Nancy-Université
Université Henri Poincaré

Master Systèmes Embarqués et Energie
Faculté des sciences et technologies
BP70239
54506 VANDOEUVRE LES NANCY

Université Henri Poincaré, Nancy I
Faculté des Sciences et Technologies

Master Systèmes Embarqués et énergie
Spécialité Systèmes embarqués
Année universitaire 2010-2011

Modernisation d'un produit de dépistage visuel (Scolatest)

Mémoire présenté par RODRIGUEZ Francisco
Soutenu le 15/09/2011

Stage effectué ESSILOR INTERNATIONAL
ADRESSE
Tuteur industriel : BIJOU Ludovic
Tuteur universitaire : WEBER Serge

Bâtiment 1er cycle - Boulevard des Aiguillettes - B.P. 70239 - 54506 VANDOEUVRE-LES-NANCY CEDEX
Téléphone : 03.83.68.41.03 - Télécopie : 03.83.68.41.53

Nancy-Université
Université Henri Poincaré

Master Systèmes Embarqués et Energie
Faculté des sciences et technologies
BP70239
54506 VANDOEUVRE LES NANCY

RESUME.

Dans le cadre du dépistage visuel, on propose de faire évoluer un produit déjà existant. Le produit, SCOLATEST, est une machine très simple faite autours d'un petit moteur qui fait défiler des images au moyen d'une télécommande dans le quel on peut choisir le sens de la rotation. La télécommande est reliée par un fil à la boite de l'appareil. Le produit est totalement manuel et de fonctionnalité réduite, le but de ce projet est de moderniser et réduire les coûts, pour ce faire, j'ai ajouté un écran tactile, une télécommande technologie sans fils et une carte électronique avec un système linux embarqué qui permet de gérer une application.

L'application consiste à faire défiler des images mais en permettant de modifier différents paramètres comme la langue et les images qu'on veut utiliser pendant le test. L'application a été développée avec des bibliothèques graphiques multiplateformes (GTK et QT), et Le système est implémenté sur une cible arm.

MOTS-CLES : Linux, Embarqué, multiplateforme, arm, wrapper.

ABSTRACT:

This project deals with the improve of a product of visual screening. The product, SCOLATEST is a very simple machine made around a small motor that rotates images using a remote control who allow you choose the direction of rotation. The remote control is connected by a wire to the box of the device. The product is totally manual and with limited functionalities, the aim of this project is to modernize and to reduce the cost, to do it, I have added a remote control wireless, a touch screen and an electronic card with an embedded system Linux.

The application consists to rotate images conceived specially for the test, allowing to configure some parameters such as the language. The application has been developed with graphical libraries multi platforms (GTK, Qt), and the system is implemented on a target arm.

KEYWORDS : Linux, Embedded, Multiplatform, arm, Wrapper.

Nancy-Université
Université Henri Poincaré

Master Systèmes Embarqués et Energie
Faculté des sciences et technologies
BP70239
54506 VANDOEUVRE LES NANCY

Remercîments

Tout d'abord, je voudrais remercier toutes les personnes d'ESSILOR INTERNATIONAL qui m'ont aidé à accomplir ce projet dans de bonnes conditions.

Je voudrais remercier plus particulièrement mon encadrant, Mr. Ludovic Bijou, pour sa disponibilité et sa patience. Il m'a apporté, non seulement des connaissances rattachées à mon domaine d'étude, mais aussi il m'a beaucoup aidé à m'améliorer en langue Française, indispensable à la rédaction de ce rapport.

Nancy-Université
Université Henri Poincaré

Master Systèmes Embarqués et Energie
Faculté des sciences et technologies
BP70239
54506 VANDOEUVRE LES NANCY

Sommaire.

Bâtiment 1er cycle - Boulevard des Aiguillettes - B.P. 70239 - 54506 VANDOEUVRE-LES-NANCY CEDEX
Téléphone : 03.83.68.41.03 - Télécopie : 03.83.68.41.53

Avant-propos

La formation du master en systèmes embarqués et énergie (SEE) à la faculté de sciences de Nancy, est complétée par des stages académiques. Ces stages complètent la formation théorique reçue par des cas pratiques sur le terrain, et préparent l'étudiant à son insertion professionnelle.

C'est dans cette optique que j'ai effectué un stage dans la société ESSILOR INTERNATINAL basé à Ligny-en-Barrois et avec une durée d'un an.

Le but de ce stage est la « Modernisation d'un produit pour le dépistage visuel (SCOLATEST)». La principale motivation pour la réalisation de ce projet est de donner solution à une préoccupation communément rencontrée dans le monde industriel, la réduction des coûts de production, avec l'utilisation de nouvelles technologies.

Objectifs

1. *Recherche des outils pour développement embarqué / linux*
2. *Prototype application*
3. *Utilisation Windows pour le développement application*
4. *Prototype application sur Windows*
5. *Recherche d'une cible hard + écran*
6. *Application sur cible*

Planning de taches.

1. ***Simulation haut niveau:***
   - *Développement application changement écran sur PC*
     - ➔ *génération d'un exécutable qui tourne sur PC sans simulation*
   - *Utilisation librairie graphique*
     - ➔ *Librairie graphique Qt*
     - ➔ *Librairie graphique GTK*
     - ➔ *Wrapper pour la librairie graphique*
   - *Réception des événements télécommande*
2. ***Simulation du système :***
   - *Installation Qemu*
   - *Portage linux allégé*
   - *Gestion USB*
3. ***Cible :***
   - *Gestion du boot.*
4. ***Outils de développement***
5. ***Electronique :***
   - *Choix carte électronique*
     - ➔ *USB*
     - ➔ *Processeur (ARM? / X86 ? / PIC ?)*
     - ➔ *carte SD*
   - *Choix de l'écran*

INTRODUCTION.

Aujourd'hui grâce à l'évolution de l'informatique embarquée, plus précisément les systèmes linux embarqué et les applications multi plateforme, Essilor veut continuer son innovation, avec la modernisation et conception de nouveaux outils pour le dépistage visuel, en utilisant ces nouvelles technologies.

Pourquoi un système Linux embarqué? : *L'avantage de l'utilisation d'un système Linux est que :*

- *Le code source du noyau et des programmes est accessible à tous. Cela veut dire qu'un utilisateur ou un programmeur peut modifier, ajouter ou corriger un logiciel via son code source, librement, sans aucune restriction.*
- *Toutes les distributions Linux sont disponibles gratuitement ou téléchargeables par Internet, ce qui est important pour les systèmes destinés à être fabriqués en grande série.*

Aujourd'hui on trouve des systèmes embarqués un peu par tout des PDA, des téléphones mobiles, des robots, les routeurs / serveurs, appareils électroniques et comme dans notre cas un produit de dépistage visuel.

Et pourquoi une application multi plateforme? :

Imaginons une application, par exemple un jeu vidéo. Le développeur souhait diffuser sa production sur le plus de machines possibles. Cependant, ils existent plusieurs machines avec des architectures et des OS différentes qui n'acceptent pas une simple importation du code d'origine. Il faut souvent reprogrammer une partie du logiciel, ou de fois son intégralité. En termes de temps et de moyens humains ce n'est pas raisonnable.

Un exemple de cela on le trouve sur le marché de la téléphonie mobile: Apple, Google via Android, Blackberry, Samsung, LG ou Nokia avec Symbian. Matériellement, intègrent presque tous une architecture basée sur les processeurs ARM (Advanced RISC Machines). Pourtant malgré une conception similaire, du processeur jusqu'à l'écran, on ne peut pas par exemple implémenter une application androïde développée en Java pour un Iphone car java n'est pas disponible sur le iPhone et réciproquement l'Objective C n'est pas disponible sous Android. Dans le cas particulier d'Essilor on a un peu le même problème. On a différentes produits avec d'architectures et d'OS différents et on voulait que l'application puisse être facilement implémentée dans ces différentes plateformes.

I. Description de l'Entreprise

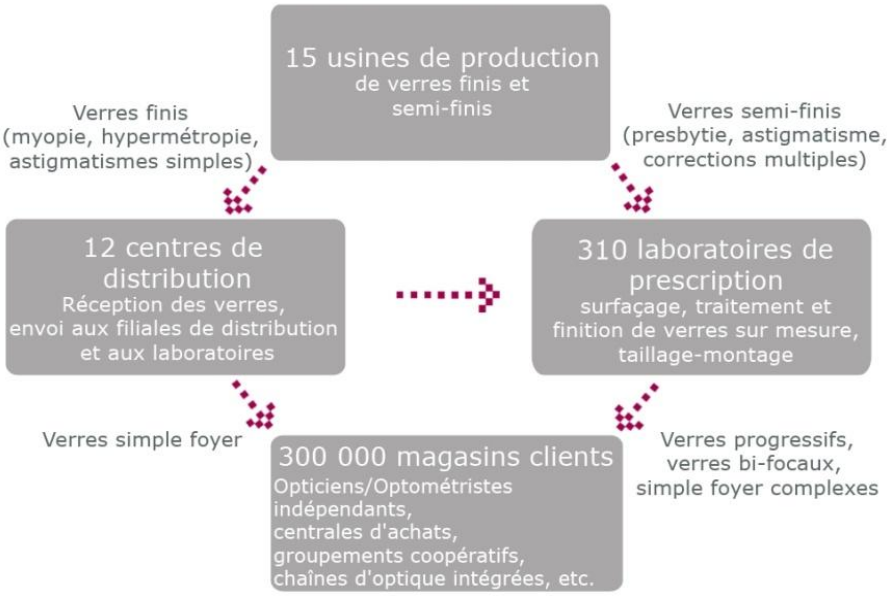
Historique

Essilor International SA est une société française qui produit des verres et de l'équipement ophtalmique. La société est basée à Paris, en France, et est cotée à la bourse Euronext de Paris. Faisant partie des 40 plus grandes entreprises de commerce de Paris, elle fait partie de l'indice CAC 40.

Essilor est la société à l'origine du Varilux, le premier verre progressif qui corrige la presbytie en donnant au porteur une correction en vision de près, en vision intermédiaire et en vision de loin1. La société s'est formée par la fusion des compagnies Essel et Silor en 1972. Essilor est aujourd'hui présent dans plus de 100 pays, répartis sur les cinq continents. Ses activités sont essentiellement axées sur la recherche, développement et la production de verres. La société est le plus grand fabricant de verres ophtalmiques du monde, dominant le marché sur tous les continents, et se positionne comme la quatrième plus grande société de matériel médical en Europe.

Organisation mondiale d'Essilor

Une couverture intégrale, le réseau Essilor est conçu de façon à former un maillage complet. De la fabrication du produit à son acheminement en magasin, le groupe est présent à chaque étape du processus. Les principaux acteurs du groupe Essilor



Implantations dans le monde

Plus de 34 000 collaborateurs travaillent quotidiennement au développement du groupe dans 100 pays.

Usines

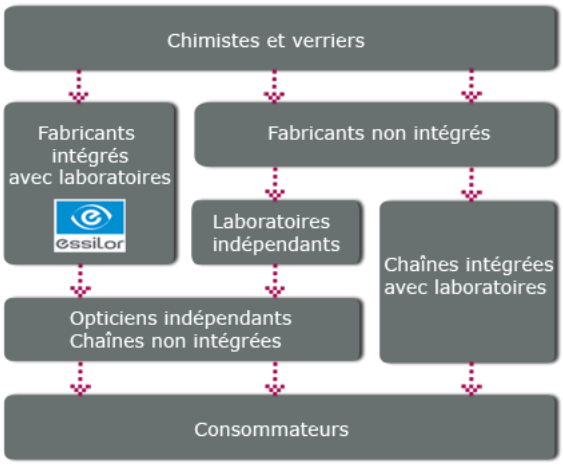
La qualité Essilor partout dans le monde, les usines Essilor approvisionnent les marchés et les filiales en verres finis et semi-finis. 12 des 15 usines fabriquent des verres à destination de plusieurs continents, ce qui implique de très nombreux flux logistiques.

Les sites de production de série d'Essilor :

Chiffres clés :

- *220 millions de verres fabriqués en moyenne.*
- *300 000 références différentes*
- *15 usines implantées dans le monde entier*

Le verre : du fabricant au consommateur.



Essilor Instruments

L'activité de la Division Instruments d'Essilor a deux spécialités :

- *Les **appareils de taillage-montage** destinés aux opticiens / optométristes et aux laboratoires de prescription.*
- ***Les équipements de dépistage** des défauts visuels physiologiques qui s'adressent aux institutions de la médecine préventive.*

La Division Instruments en chiffres :

- *700 collaborateurs à travers le monde*
- *Présence dans près de 70 pays*
- *Une structure de Recherche et Développement répartie sur 3 continents*
- *3 marques : Essilor, National Optronics, Stereo Optical*

Les équipements de taillage-montage

Essilor est le numéro un mondial des appareils de taillage des verres finis destinés aux opticiens / optométristes et aux laboratoires de prescription.

Ces instruments d'optique permettent d'effectuer toutes les étapes depuis l'examen de vu au montage des lunettes en passant par la prise de mesure, le taillage ou encore le perçage des verres.

Mr Blue, la chaîne numérique comprenant un lecteur-centreur-bloqueur et une meuleuse.

Le dépistage

Essilor Instruments développe au travers de la marque Essilor en Europe et Stereo Optical en Amérique du nord, une variété d'équipements de dépistage visuel.

Ces derniers trouvent leur application notamment dans la médecine du travail, les contrôles liés au permis de conduire et le dépistage scolaire.

En tant que numéro un mondial dans ce secteur, Essilor participe à des campagnes de sensibilisation et de dépistage en Afrique et en Asie.

Visiotest, permet d’évaluer la performance visuelle des porteurs de progressifs.

II. Présentation du sujet.

Dans le cadre du dépistage visuel, on propose de faire évoluer un produit déjà existant.

III.1. SCOLATEST.

Le produit, SCOLATEST, est une machine très simple fait autour d'un petit moteur qui fait tourner des images au moyen d'une commande dans le quel on peut choisir le sens de la rotation des images. La commande est reliée par un fil de 2,5 m (distance nécessaire pour le test) à la boite de l'appareil. Le SCOLATEST a été conçue pour être utilisé dans les écoles pour un test de vue des enfants.

III.1.1. Caractéristiques :

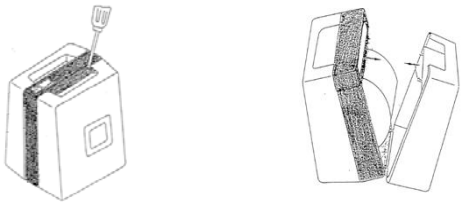
- *Dépistage de l'amétropie chez l'enfant dès l'âge de 3 ans.*
- *Disponible en deux versions : 3/6 ans et 3/16 ans*
- *Dépistage de fortes hypermétropies grâce aux lunettes +2 D et +4 D.*
- *Présentation ludique des optotypes pour une participation active de l'enfant.*
- *Prise de mesure rapide, facile et fiable.*
- *Chaque Scolatest est équipé au recto, d'une échelle de Shéridan, au verso d'une échelle morphoscopique.*

III.1.2. Utilisation.

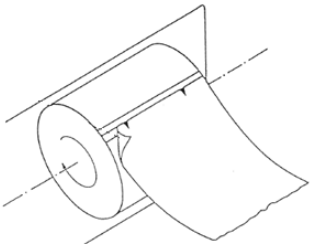
Le Scolatest doit être installé de préférence dans une salle moyennement éclairée en évitant toute source de lumière intense pouvant gêner le déroulement du dépistage. Le Scolatest sera posé sur une table et sera à hauteur des yeux de l'enfant examiné. Le câble de la télécommande sera alors déroulé perpendiculairement à la face avant de l'appareil. Le câble permet de déterminer précisément une distance de 2,50 m. L'examinateur et l'enfant seront donc parfaitement positionnés lorsqu'ils se trouveront dans le plan de la prise de

raccord du manipulateur. Un paquet de fiches de dépistage sera placé à côté de l'examinateur.

Pour changer de test (changer les images), il fallait de démonter l'appareil :



Et changer la bande de test :



Comme on le voit le produit est totalement manuel et de fonctionnalité réduite, le but de ce projet est de moderniser et réduire les coûts.

Le SCOLATEST a été totalement numérisé, j'ai ajouté un écran tactile et une carte électronique avec un système linux embarqué qui permet de gérer une application multi plateforme.

Le système a un dual OS grâce à la capacité de la carte électronique de donner support à plusieurs OS et la possibilité de démarrer le système à partir de la mémoire interne de la carte ou d'une mémoire SD. Le système principal est micro système linux que j'ai porté de l'A à la Z.

L'application consiste à faire tourner des images mais en permettant de modifier différents paramètres comme la langue, les images qu'on veut utiliser et le mode de fonctionnement, manuel (on fait défiler les images avec une télécommande sans fil) ou automatique (les images défilent automatiquement avec un intervalle de temps prédéfini).

III.2. Schéma General :

La carte électronique utilisée est la MINI2440 de FriendlyARM, la télécommande est un TARGUS technologie sans fil, le système d'exploitation utilisé est un microsystème linux et l'application peut être implémentée en utilisant la bibliothèque graphique Qt ou bien GTK et sur n'importe quelle plateforme (Linux, Windows, Android).

Pour la réalisation de l'application en combinant deux bibliothèques graphiques j'ai développé un wrapper qu'on pourrait traduire en français par adaptateur est habituellement un programme qui « enveloppe » l'exécution d'un autre programme, pour lui préparer un environnement particulier, c'est-à-dire, le Wrapper fait d'interface entre l'application et les bibliothèques graphique, cela permet à l'application de ne pas dépendre d'une bibliothèque graphique en particulière.

III.3. Carte Electronique : Mini2440 (FriendlyARM).

Cette carte architecturée autour d'un ARM9 Samsung cadencé à 400MHz présente un concentré de ressources pour un prix raisonnable (environ 65 Euros).

Caractéristiques technique :

- ***Dimensions** : 100 x 100 mm*
- ***CPU** : 400 MHz Samsung S3C2440A ARM920T (Max freq. 533 MHz)*
- ***RAM** : 64 MB SDRAM, 32 bit 100 MHz Bus*

- ***Flash** : 64 MB / 128 MB / 256 MB / 1GB NAND Flash and 2 MB NOR Flash with BIOS*
- ***EEPROM** : 1024 Byte 24C08 (I2C)*
- ***Ext. Memory** : SD-Card socket*
- ***Serial Ports** : 1x DB9 connector (RS232), total : 3x serial ports on the PCB*
- ***USB** : 1x USB-A Host, 1x USB-B Device*
- ***Audio Output** : 3.5 mm stereo jack*
- ***Audio Input** : on PCB + condenser microphone*
- ***Ethernet** : RJ-45 10/100M (DM9000)*
- ***LCD Interface***
- ***Touch Panel** : 4 wire resistive*
- ***User Inputs** : 6x push buttons and 1x A/D port*
- ***User Outputs** : 4x LEDs*
- ***Expansion** : 40 pin System Bus, 34 pin GPIO (2.0mm)*
- ***Debug** : 10 pin JTAG (2.0mm)*
- ***OS Support** :Android, Linux 2.6, Windows CE 5 and 6*

J'ai opté par la mini2440 avec un écran tactile de 7" :

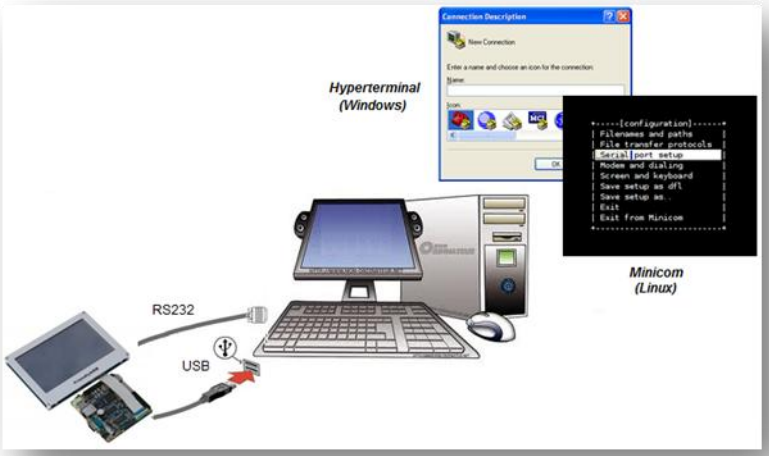
La carte vient avec windowsCE5.net préinstallé et avec un touchscreen calibré, la connexion avec le PC en mode console se fait à travers du port série.

La mini2440 possède deux types de mémoire Flash, une NOR et une NAND, la NOR vient avec un bootloader SUPERVIVI préinstallé, et la NAND est l'espace destiné à l'utilisateur. Donc le boot peut se faire à partir de la NOR ou bien de la NAND, le choix se fait à travers d'un petit interrupteur. Le boot sur la NOR (SUPERVIVI) permet de partitionner la NAND et charger un noyau et une image root.

III. Réalisation

La MINI2440 donne support à plusieurs OS comme Android, Linux 2.6 et Windows, ce qui est important pour l'implémentation d'une application multiplateforme.

Pour l'installation d'un OS premièrement, il faut se connecter à la Cible, la connexion se fait à travers du port série, cette connexion permet d'installer le noyau, le file-système de base et de gérer l'envoie des fichiers en utilisant la console de la cible (minicom sous Linux et HyperTerminal sous Windows) à travers d'une connexion USB.



IV.1. Installation des OS dans le SCOLATEST.

Comme OS pour ce système embarqué j'ai pris QTOPIA et le micro système linux que j'ai développé pendant le premier semestre de ce stage (voir annexes). QTOPIA est implémenté sur la NAND de la carte et le micro système linux sur une SD. Comme bootloader j'utilise U-boot.

IV.1.1. QTOPIA .

Qtopia est une plateforme libre développée par Qt Software, basée sur la bibliothèque Qt pour les systèmes embarques linux et son installation sur la MINI2440 peut se résumer de la façon suivant :

1. ***Connexion à Hyper terminal ou minicom***
2. ***Boot de la carte à partir de la NOR***
3. ***Formatage de la NAND :*** *Option « **x** »sur le BIOS*

4. ***Installation du Bootloader :*** *Option* ***« v ».***
5. ***Installation du Kernel Linux :*** *option* ***« K »***
6. ***Installation du système de fichiers racine Linux :*** *option* ***«y »***
7. ***Configuration du bootloader :*** *Supervivi permet de passer certains paramètres au kernel pour le bon démarrage du système.*
8. ***Redémarrage de la carte.***

Le processus d'installation d'un OS sur la Mini2440 est détaillé dans les annexes.

Les sources de Qtopia sont disponibles sur le site « friendlyarm.net ». J'ai utilisé « root-qtopia-128M.img » comme racine et « zImage_A70 » (fournie avec la mini2440) comme image du kernel.

IV.1.2. Micro système linux.

Pour l'installation sur la SD il faut tout d'abord faire deux partions, kernel et rootfs, la première partition en format FAT16 pour le kernel et la deuxième en format ext3 pour le système de fichiers racine, pour ce faire j'ai utilisé gparted (Gnome partition editor) qui est un éditeur de partitions libre, qui permet de créer, formater, supprimer et déplacer des partitions sur un le disque.

Pour la racine il faut juste copier le root file réalisé dans la première partie de cette stage dans SD (partition rootfs) :

```
Sudo cp –R * /media/rootfs
```

Mais le noyau il faut le recompiler avec la «toolchaine » fournie avec la carte (arm-linux-gcc-4.3.2).

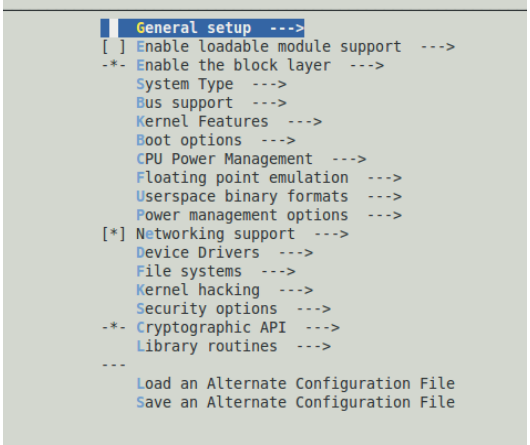
Les sources du noyau sont disponibles sur le site de la carte avec le fichier de configuration. Sur une console on fait comme suit :

- *On copie le fichier de configuration :*

```
Sudo cp config_mini2440_a70 .config
```

- *On rentre au menu de configuration:*

```
Make ARCH=arm menuconfig
```



Et on fait les modifications suivantes:

- *Désactiver « Enable loable module support »*
- *Sélectionner le microprocesseur « System Type/ARM system type/Sansung S3C2410, S3C2412, S3C2413, S3C2440, S3C2442, S3C2443*
- *Activer le format Ext3 "File systems/Ext3 journalling file system support"*
- *Image de demurrage: "File systems/Device Drivers/Graphics support/Bootup logo/Standard 224-color Linux logo"*
- *Support USB: "File systems/Device Drivers/USB support"*
- *Support SD: "File systems/Device Drivers/MMC/SD/SDIO card support"*

- *On compile*

```
Sudo make ARCH=arm CROSS_COMPILE=arm-linux- all uImage
```

Apres la compilation on trouve l'image du kernel dans le répertoire ~/linux-2.6.32.2/arch/arm/boot/uImage

- *On copie l'image dans la SD :*

```
Sudo cp uImage /media/kernel
```

Ensuite on peut configurer le bootloader. J'ai utilisé « le microsystème linux » (boot sur la SD) comme système de démarrage par défaut en configurant U-boot comme suit :

```
Setenv bootargs console=ttySAC0,115200 root=/dev/mmcblk0p2 rw
rootfstype=ext3 mini2440=3tb rootdelay=1 init=/sbin/init

Setenv bootcmd mmcinit \; fatload mmc 0:1 0x32000000 uimage.bin \;
bootm 0x32000000

saveenv
```

Pour faire le boot à partir de la NAND (Qtopia) il faut juste modifier la variable « root » :

```
Setenv bootargs root=/dev/mtdblock3

saveenv
```

IV.2. Développement de l'application multiplateforme pour le dépistage visuel.

Les avantages d'un mode de développement multiplateforme sont nombreux :

- *Un code est réalisé une seule fois*
- *Il est facile à maintenir et à adapter sur une nouvelle plateforme*
- *Les équipes spécialisées sur chaque plateforme sont réduites.*

Dans cette étude j'ai plusieurs plateformes (Windows, Android, Qtopia, micro-Linux, Emdebian) avec une architecture similaire (ARM).

Pour permettre aux logiciels d'avoir le plus d'audience possible on peut recourir à trois techniques :

- ***Les langages compilés avec des bibliothèques multiplateformes**, l'avantage des programmes compilés avec ces bibliothèques est que ceux-ci sont directement utilisables chez l'utilisateur final. En fournissant la bibliothèque avec le produit, il n'est pas nécessaire d'utiliser un interpréteur ou une machine virtuelle. Parmi les bibliothèques multiplateformes les plus populaires on trouve QT et GTK.*
- ***Les langages interprétés,** Ces langages sont interprétés à chaque fois que le programme est utilisé, du coup si l'interpréteur existe pour une plate-forme donnée, le programme fonctionnera sur cette plate-forme.*

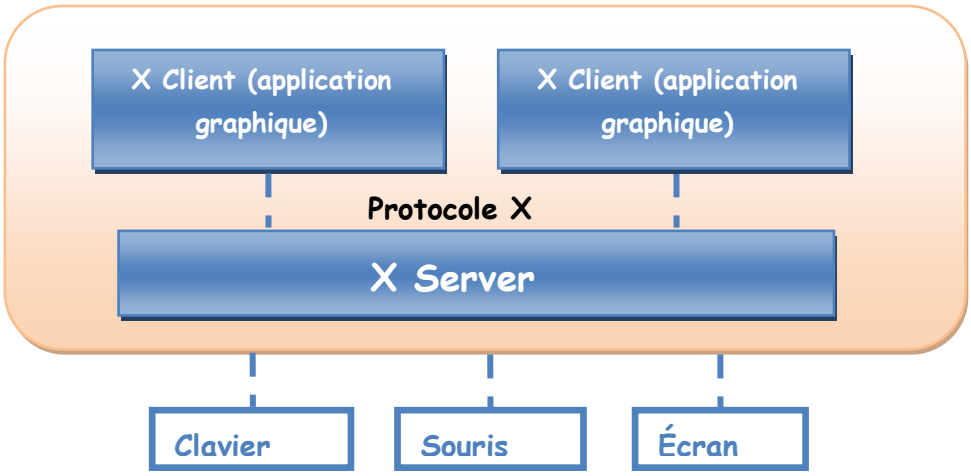
- ***Les langages semi-interprétés,*** *ces langages sont compilés vers un code intermédiaire qui est interprété. Cette approche présente les mêmes avantages que les codes interprétés mais possède une plus grande vitesse d'exécution. Le principal exemple est le langage Java. Ce langage est compilé en bytecode Java qui est ensuite interprété par une machine virtuelle.*

Pour ce projet j'ai travaillé avec de ***bibliothèques graphiques multiplateformes.***

Une bibliothèque graphique est une bibliothèque logicielle spécialisée dans les fonctions graphiques. Elle permet d'ajouter des fonctions graphiques à un programme. Il existe de nombreuses bibliothèques permettant de programmer une interface graphique pour un logiciel. Et la portabilité, la rapidité d'exécution, la rapidité et le coût de développement, la stabilité et la licence du logiciel dépendront du choix de la bibliothèque graphique.

La gestion d'une interface graphique pour les systèmes linux se base sur le X Window System ou X11 ou simplement X qui est un environnement graphique de type « fenêtré » qui gère l'interaction homme-machine par l'écran et la souris. Il est possible d'installer un serveur X sur la plupart des systèmes d'exploitation, dont Windows.

X fonctionne suivant le modèle client/serveur, où le logiciel serveur X tourne sur une machine qui est dotée d'un écran, d'un clavier et d'une souris (terminal X) ; il reçoit et sert des requêtes d'affichage, d'entrées de texte et de déplacement de souris sur un port logiciel. Un logiciel client X (logiciel graphique) se connecte au serveur X et lui envoie ses requêtes d'affichages en utilisant le protocole X. Le client est simplement l'application logicielle (application de dépistage visuel) qui utilise alors le protocole X pour déléguer au serveur X les taches d'IHM.

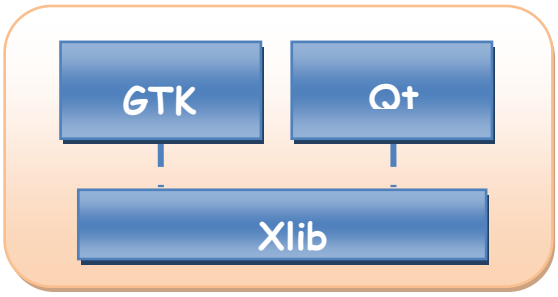


La programmation pour X peut se faire à différents niveaux. Le plus bas est celui du protocole X, où le client et le serveur échangent des données mais cela manque d'abstraction, la gestion de l'IHM se fait en utilisant des fonctions plutôt bas-niveau (ce n'est pas implémentée au niveau utilisateur mais au niveau du noyau), donc cela prendra plus de temps et elle ne sera pas portable.

Le niveau au-dessus est celui de la bibliothèque X ou Xlib. La Xlib propose une traduction assez directe des requêtes du protocole X en appels de fonctions. Elle permet notamment de créer et de manipuler des fenêtres, de dessiner à l'intérieur par l'intermédiaire d'un « contexte graphique » et de recevoir des événements (clavier, souris, exposition des fenêtres, messages inter-clients, etc.) Pour la Xlib, une fenêtre est un rectangle dans lequel on peut dessiner, et qu'on peut soulever ou baisser par rapport aux autres (on peut aussi imbriquer des fenêtres). La programmation directe en Xlib n'est guère adaptée que pour des environnements à l'environnement graphique très limité.

Le niveau suivant est celui des bibliothèques comme GTK+ et Qt qui sont surcouches de la Xlib.

Ces bibliothèques sont orientées à objets (mais entièrement en C) et permettent de manipuler des widgets, qui sont des fenêtres munies de ressources et de méthodes, leur permettant de réagir « toutes seules » à certains événements, elles agissent d'une surcouche d'une bibliothèque de niveau inférieur (xlib) et offrent ainsi une interface de développement plus confortable.



IV.2.1. Qt.

Qt est un ensemble de bibliothèques ou plutôt un framework (ensemble d'outils pour développer vos programmes plus efficacement) multiplateforme, Qt est donc constituée d'un

ensemble de bibliothèques, appelées "modules". On peut y trouver entre autres ces fonctionnalités :

- ***Module GUI** : pour création de fenêtres.*
- ***Module OpenGL** : Qt peut ouvrir une fenêtre contenant de la 3D gérée par OpenGL.*
- ***Module de dessin** : pour dessiner en 2D.*
- ***Module réseau :** Qt fournit une batterie d'outils pour accéder au réseau.*
- ***Module SVG :** possibilité de créer des images et animations vectorielles, à la manière de Flash.*
- ***Module de script :** Qt supporte le Javascript (ou ECMAScript),*
- ***Module SQL :** permet un accès aux bases de données*

IV.2.2. GTK.

GTK+ est une des plus importantes bibliothèques utilisées sous Linux. Elle est portable, c'est-à-dire utilisable sous Linux, Mac OS et Windows. GTK+ est utilisable en C.

*GTK+ est la bibliothèque de prédilection pour ceux qui écrivent des applications pour Gnome sous Linux, mais elle fonctionne aussi sous KDE. C'est la bibliothèque utilisée par Firefox par exemple, pour ne citer que lui, elle est constituée de plusieurs bibliothèques indépendantes développées par l'équipe de **GTK+**, comme **GLib, Pango** et **ATK**.*

IV.3. Installation de la bibliothèque graphique sur la Mini2440.

Comme bibliothèque principale j'utilise QT par être un peu plus évolué que GTK, son installation peut être résumée de la façon suivante :

- *Tout d'abord il faut télécharger les sources depuis "qt.nokia.com", j'ai pris **« qt-everywhere-opensource-src-4.7.3 ».***
- *On décompresse les sources :*

```
Sudo tar zxvf qt-embedded-linux-opensource-src-
4.7.3.tar.gz
```

- *On établie la toolchain à utiliser en éditant le fichier «~/qt-embedded-linux-opensource-src-4.7.3/mkspecs/qws/linux-arm-g++/qmake.conf ». J'ai utilisé la configuration suivante :*

```
#
# qmake configuration for building with arm-linux-g++
#

include(../../common/g++.conf)
include(../../common/linux.conf)
include(../../common/qws.conf)

# modifications to g++.conf
QMAKE_CC                = arm-linux-gcc
QMAKE_CXX               = arm-linux-g++
QMAKE_LINK              = arm-linux-g++
QMAKE_LINK_SHLIB        = arm-linux-g++

# modifications to linux.conf
QMAKE_AR                = arm-linux-ar cqs
QMAKE_OBJCOPY           = arm-linux-objcopy
QMAKE_STRIP             = arm-linux-strip

#compilateur mini2440

QMAKE_CFLAGS_RELEASE += -msoft-float -D__GCC_FLOAT_NOT_NEEDED -march=armv4 -mtune=arm920t
QMAKE_CXXFLAGS_RELEASE += -msoft-float -D__GCC_FLOAT_NOT_NEEDED -march=armv4 -mtune=arm920t
QMAKE_CFLAGS_DEBUG += -msoft-float -D__GCC_FLOAT_NOT_NEEDED -march=armv4 -mtune=arm920t
QMAKE_CXXFLAGS_DEBUG += -msoft-float -D__GCC_FLOAT_NOT_NEEDED -march=armv4 -mtune=arm920t

load(qt_config)
```

- *On exécute le script CONFIGURE :*

```
./configure -embedded arm -xplatform qws/linux-arm-g++ -
   prefix /usr/local/Qt -qt-mouse-tslib -little-endian
```

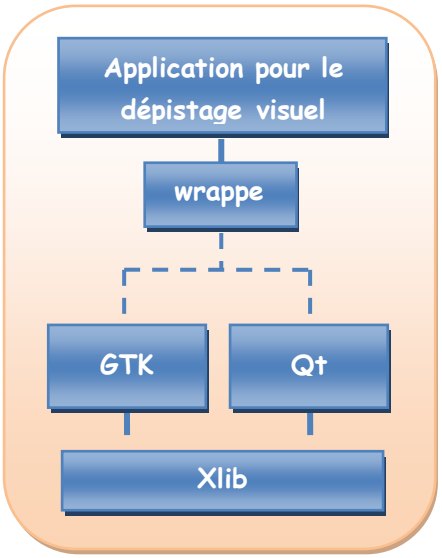
- *Make*
- *Sudo make install*

Comme on voit dans le fichier configuration le compilateur utilisé est celui fournit avec la carte (compilation croisée). Apres la compilation on place le résultat dans le répertoire « /lib » du système de fichiers racine « rootfs »

IV.4. Interaction entre les bibliothèques graphiques.

Aujourd'hui pour le développement d'une application graphique existent différentes alternatives (Nano-X Window System, Qt, Gtk, WxWidgets, OpenGL, SDL, etc.) et l'évolution rapide de l'informatique fait penser qu'on aura de nouvelles bibliothèques graphiques ou l'évolution des déjà existantes, par cela, ce projet se base en le développement d'une

application qui puisse s'adapter à n'importe quelle bibliothèque, en assurant ainsi son existence malgré l'évolution constant de l'informatique embarqué. Pour ce faire, j'ai développé un Wrapper qui consiste en une application qui masque la complexité de la bibliothèque graphique utilisée en proposant un jeu de fonctions intuitives, ainsi, le développeur n'a pas à connaître le fonctionnement de la bibliothèque utilisé pour modifier l'application de dépistage visuel. Seule une interface composée de certaines fonctions (créer_bouton, créer_fenetre, etc) sont nécessaires au développeur. Cela introduit un autre niveau d'abstraction, un développeur n'a donc pas à se soucier de la façon dont une bibliothèque fonctionne.



Le Wrapper sert d'interface entre l'application et la bibliothèque graphique, il se charge de rediriger les appels des fonctions de l'application vers la bibliothèque choisie.

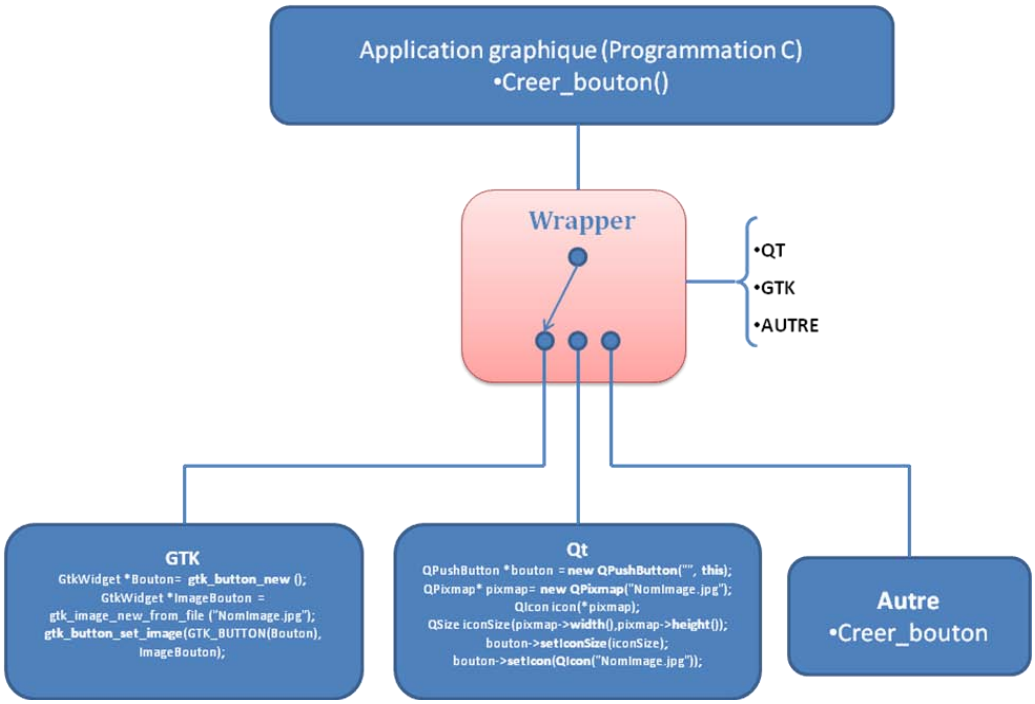
Par exemple pour la création d'un bouton avec GTK on doit utiliser les fonctions suivantes :

```
GtkWidget *Bouton=  gtk_button_new ();
GtkWidget *ImageBouton = gtk_image_new_from_file
                ("NomImage.jpg");
gtk_button_set_image(GTK_BUTTON(Bouton), ImageBouton);
```

Et pour la création d'un bouton avec Qt on fait comme suite :

```
QPushButton *bouton = new QPushButton("", this);
QPixmap* pixmap= new QPixmap("NomImage.jpg");
        bouton->setIconSize(iconSize);
    bouton->setIcon(QIcon("NomImage.jpg"));
```

Comme on observe chaque bibliothèque graphique a une structure particulier, donc la fonction du Wrapper est d'adapter les paramètres de la fonction ***creer_bouton()****, de l'application du dépistage visuel, pour faire appel à la bibliothèque graphique choisit.*

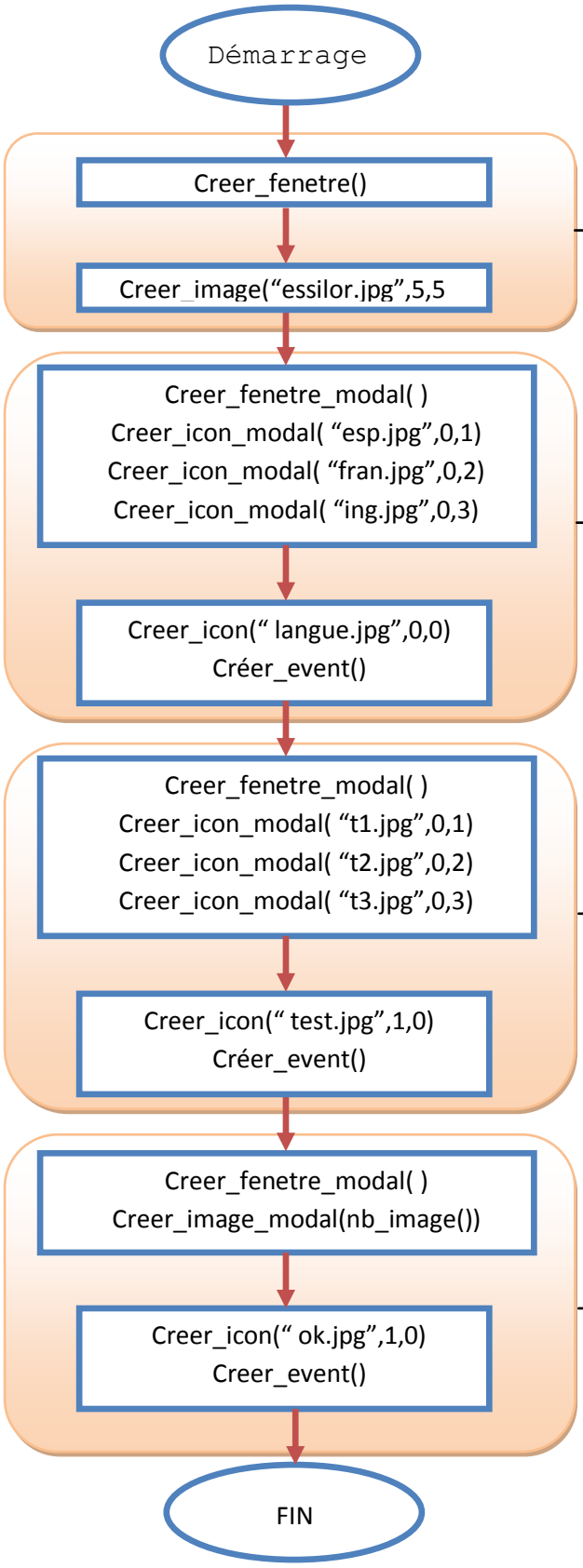


La sélection de la bibliothèque graphique se fait à travers d'une variable déclarée dans le wrapper, cette application supporte deux bibliothèques QT et GTK. L'avantage de l'utilisation du wrapper est que pour ajouter une autre bibliothèque on doit modifier que le wrapper. Donc pour n'importe quelle bibliothèque l'application reste la même.

L'application possède les fonctions suivantes :

- ***Creer_fenetre()***
- ***Creer_icon(image, position x, position y) :*** *Pour créer un bouton dans une fenêtre.*
- ***Creer_image(image, position X, position Y) :****pour insérer images dans la fenêtre.*
- ***Creer_event() :*** *Pour associer un bouton avec une fenêtre modal.*
- ***Creer_fenetre_modal() :*** *Pour créer fenêtres modales.*
- ***Creer_icon_modal(image, position x, position y) :*** *pour créer un bouton dans une fenêtre modale.*
- ***Créer_image_modal(image) :*** *Pour créer une image dans une fenêtre modale.*

IV.4.1. Diagramme fonctionnel de l'application.



Création de la fenêtre principale et insertion d'une image (logo essilor)

Création d'une fenêtre modale pour la configuration de la langue, avec trois boutons, un pour chaque langue supportée par l'application, et un événement pour lié le bouton « langue » (dans la fenêtre principal) avec cette fenêtre modale.

Création d'une fenêtre modale pour la configuration du test, avec trois boutons, un pour chaque test supporté par l'application, et un événement pour lier le bouton « test » (dans la fenêtre principal) avec cette fenêtre modal.

Création d'une fenêtre modale pour le test, avec l'insertion des images. L'argument de Creer_image_modal() est nb_image() qui est une fonction implémentée au niveau wrapper qui permet d'ouvrir le répertoire qui contient les images.

Et à travers d'un événement on lie le bouton «ok » (dans la fenêtre principale) à cette fenêtre modale.

Pour simplifier le diagramme je n'ai pas représenté les événements des boutons de la configuration de la langue et du type de test.

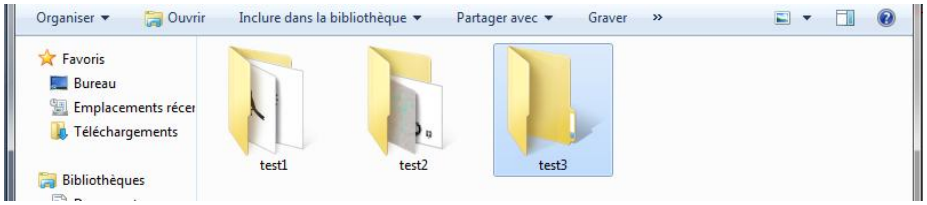
Ces évenementsts consistent en l'écriture d'un mot clef (Ex : espagnol si on choisit l'espagnol, test1 si on choisit le test 1) dans un fichier .txt. Ce fichier sera lu avant d'accéder au test pour le paramétrer en fonction de la configuration. Voir le code dans les annexes.

IV.5. Description de l'application.

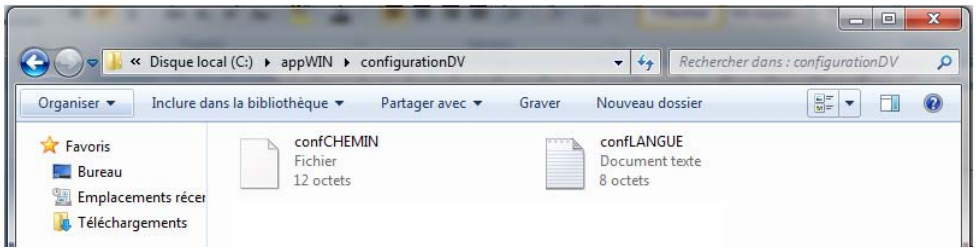
L'application reste la même pour linux et pour windows, la seule différence vient de la compilation.

L'application consiste en 2 étapes, configuration et test, la première permet de faire la configuration et la deuxième le test. Les répertoires avec les images et les répertoires destinés à la configuration doivent être dans le même dossier que les fichiers .c et .h de l'application.

Où le répertoire image contient les dossiers avec les images pour chaque test :



Et configutationDV contient 2 ficher .txt :



1. ***confCHEMIN** : contient le chemin du répertoire pour le test choisit, ce fichier est écrit à travers du bouton TEST dans la fenêtre configuration.*
2. ***confLANGUE** : contient l'information de la langue choisit dans l'étape de configuration.*

Le but de l'application est de faire un test de dépistage visuel, lequel consiste à faire défiler une série d'images spécialement conçues pour le test. Au démarrage du produit un écran ESSILOR sera affiché ainsi qu'une barre de paramétrages.

Lors de l'appui sur « langue » une fenêtre modale de 3 choix apparaît (Français, English, Espagnol), l'utilisateur en choisi 1.

Lors de l'appui sur « test », une fenêtre modale de x choix apparaît (Folder1 … FolderX), dans cette fenêtre seront affiché les répertoires contenant les images à afficher.

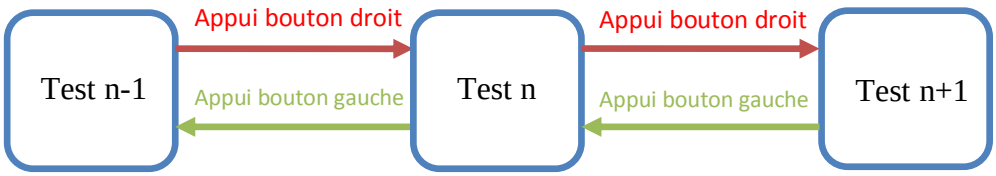
La navigation dans les menus peut se faire via la télécommande. Lors de l'appui sur le bouton droit, la sélection passe de langue (Français à Espagnol à English) à test (test 1 à test 2 à test 3) puis à bouton de validation et ainsi de suite. L'appui sur bouton gauche permet de valider la sélection.

Au prochain démarrage du produit, par défaut, la langue sera la langue sélectionnée auparavant ; de même pour la sélection des tests. Lors de l'appui sur « bouton validation », on passe à l'affichage des tests.

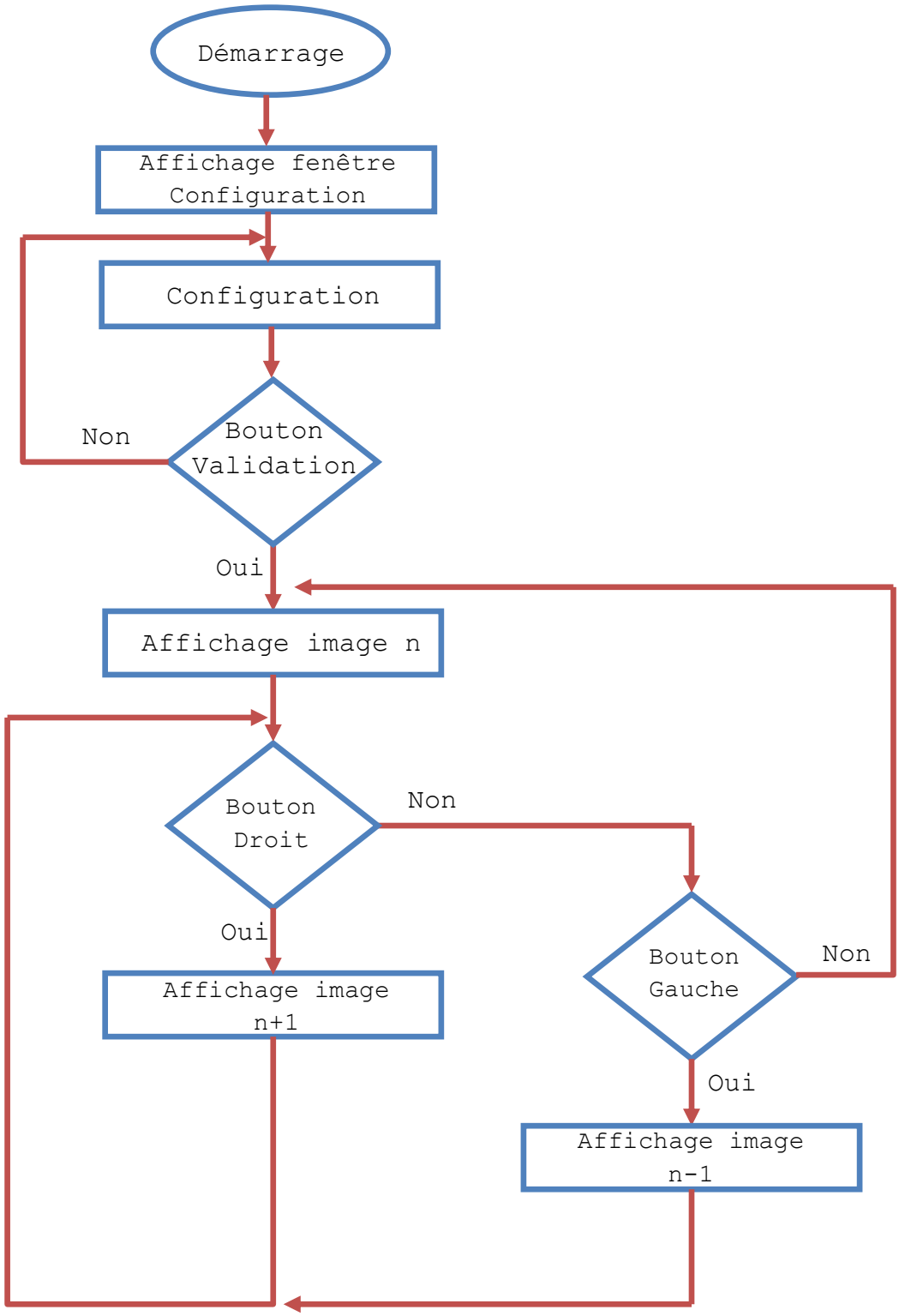
IV.5.1. Exécution du Test.

Après l'appui sur « bouton validation », le premier test est affiché.

Si on appuie sur le bouton droit de la télécommande, l'image 2 est affichée, puis la 3 sur le prochain appui etc. Et si on appuie sur le bouton gauche de la télécommande, l'image précédente est affichée, puis celle d'avant sur le prochain appui.



IV.6. Diagramme fonctionnel du système.



IV.7. Outils de développement.

IV.7.1. Installation de GTK.

Sous Windows :

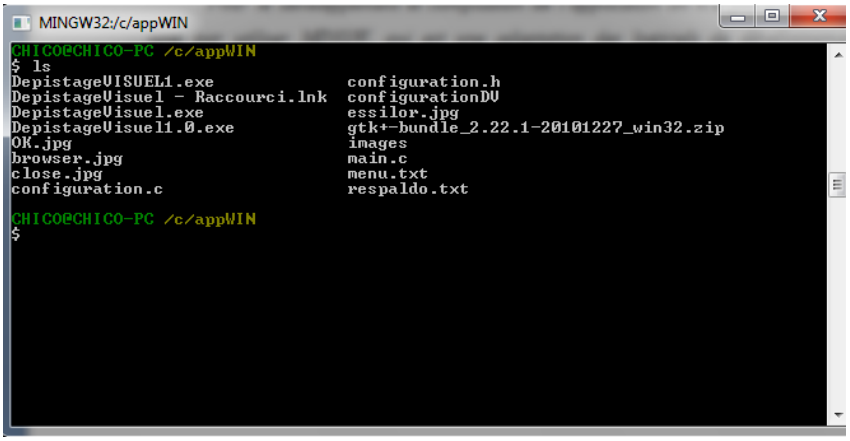
On télécharge les sources depuis ***www.gtk.org,*** *dans notre cas* ***gtk+-bundle_2.22.1-20101227_win32*** *et on décompresse le fichier dans le disque C.*

gtk+-bundle_2.22.1-20101227_win32.zip

Pour le développement et compilation de l'application, on a pris ECLIPSE, et comme compilateur MINGW, qui est une adaptation de logiciels de développement et de compilation du GNU (GCC - GNU Compiler Collection), à la plate-forme Win32.

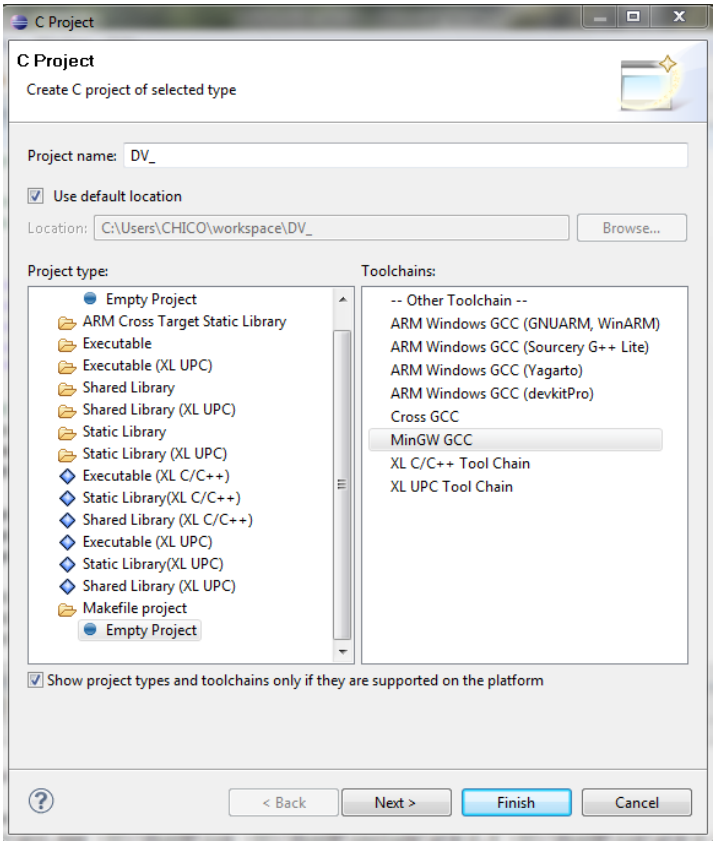
On peut le télécharger depuis http://www.mingw.org/

Avec MINGW on a sous WINDOWS toutes les commandes traditionnelles LINUX disponibles :



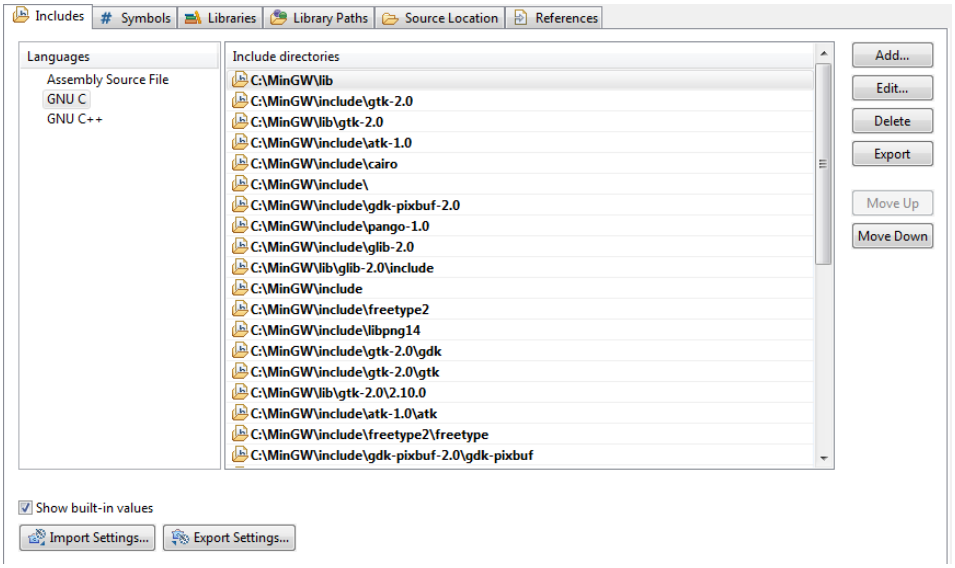
Développement avec Eclipse :

1. ***Créer un projet :*** *File->New->Project->C Project->Makefile Project->MinGW GCC->Finish*

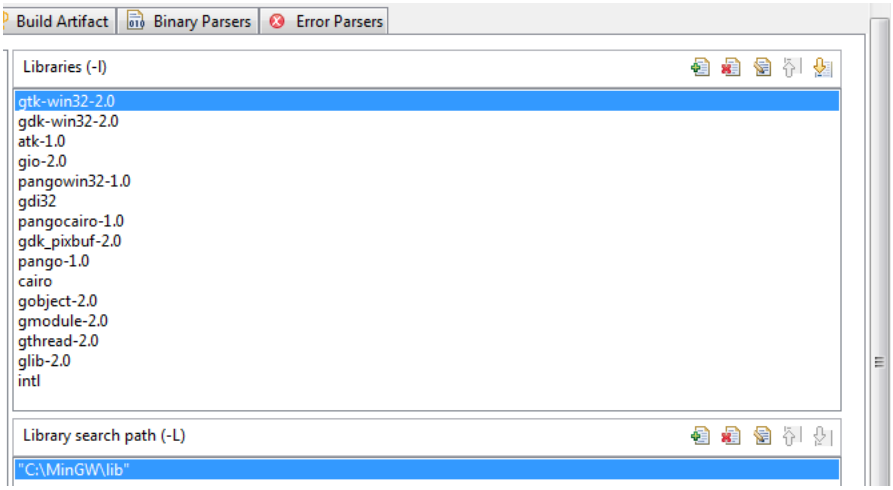


2. *Ajouter les bibliothèques GTK :*

- *Project->Properties->C/C++ General->Paths and Symbols->Include->GNU C-> et on ajoute les bibliothèques GTK*

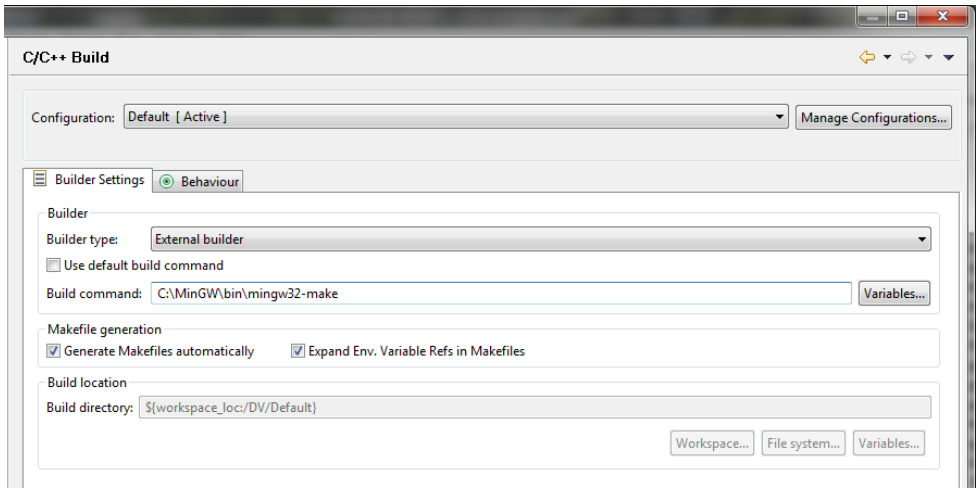


- *Project->Properties->C/C++ Build->Settings-> Tools Settings->Mingw C Linker->Libraries->on met les librairies suivantes:*



3. *Configuration du compilateur :*

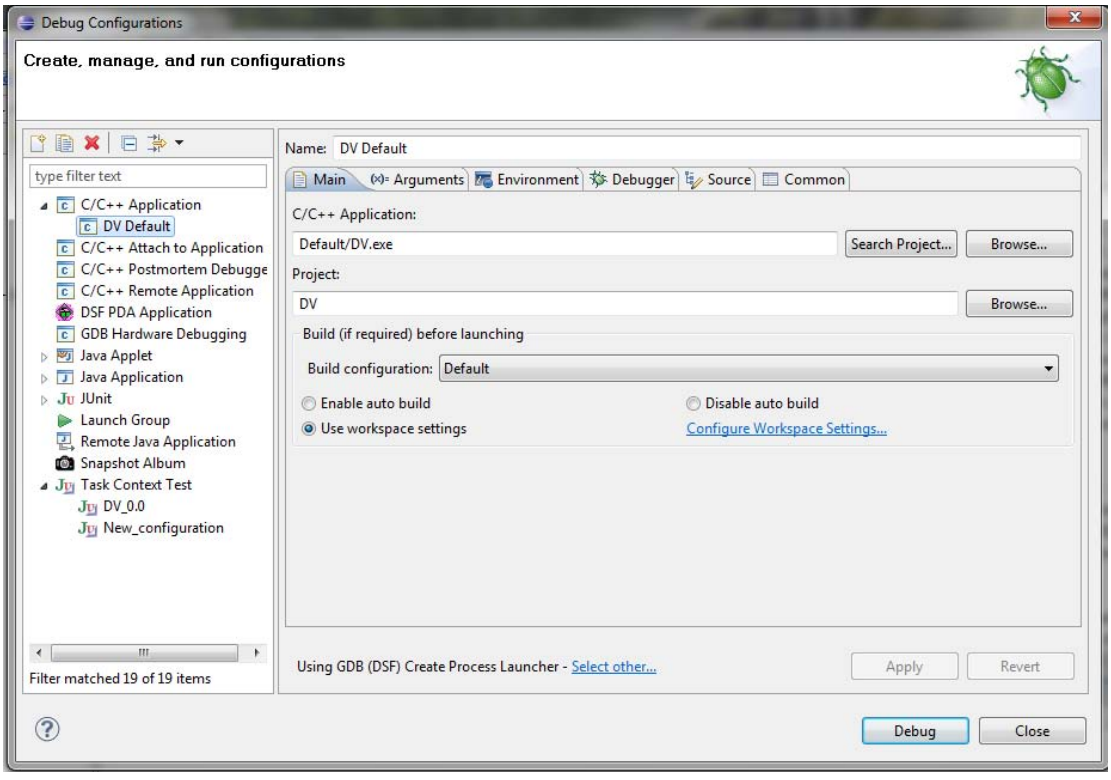
- *Project->Properties->C/C++ BUILD->On utilise la configuration suivante :*



- *Project->Properties->C/C++ BUILD->Discovery Options->GCC C Compiler->dans Compiler invocation command on met C:\MinGW\bin\gcc.exe.*
- *Project->Properties->C/C++ BUILD->Discovery Options->GCC C Compiler->Settings-> Tools Settings->GCC Assembler->on met: C:/MinGW/bin/as.*
- *Project->Properties->C/C++ BUILD->Discovery Options->GCC C Compiler->Settings->Tools Settings->GCC C++ compiler->on met: C:/MinGW/bin/g++.*

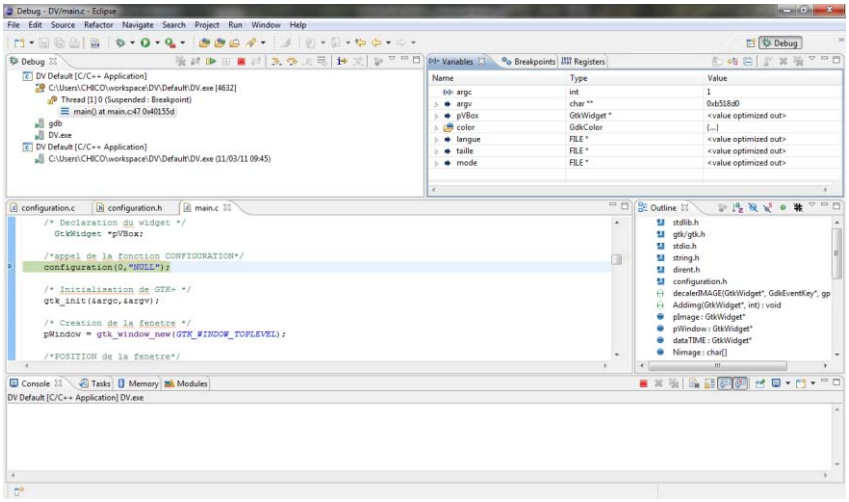
- *Project->Properties->C/C++ BUILD->Discovery Options->GCC C Compiler->Settings-> Tools Settings->GCC C compiler->on met: C:/MinGW/bin/gcc.*
- *Project->Properties->C/C++ BUILD->Discovery Options->GCC C Compiler->Settings-> Tools Settings->GCC C++ Linker->on met: C:/MinGW/bin/gcc.*
- *Project->Properties->C/C++ BUILD->Discovery Options->GCC C Compiler->Settings->Binary Parsers-> on cache: PE windows Parser, ELF Parser*

4. ***Debug de l'application** : Pour debugger l'application on a besoin du plugin CDT - http://donwload.eclipse.org/tools/cdt/releases/eclipse3.1*
   - *Help->Install new software->work with: CDT - http://donwload.eclipse.org/tools/cdt/releases/eclipse3.1.*
5. ***Compilation:** Project->Build Project*
6. ***Configuration de Simulation:** Run->Debug Configuration->Double clic sur c/c++ Application-> et on utilise la configuration suivante :*



7. *Simulation :*

-  *: Pour le débogage*
-  *: Pour la simulation*



La compilation de l'application peut aussi se faire depuis un terminal avec la commande suivante :

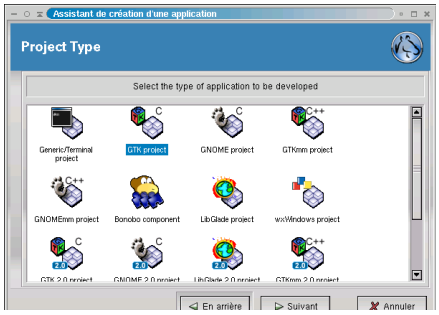
```
$gcc -Wall -g main.c configuration.c configuration.h  -o
DepistageVisuel.exe  -mwindows `pkg-config -cflags -libs gtk+-2.0`
```

Sous Ubuntu :

Sous Linux on a utilisé Anjuta qui est un environnement de développement intégré (IDE) pour le C et le C++ sur GNU/Linux. Il a été écrit pour les bibliothèques GTK+/GNOME et il fournit un nombre important de fonctions avancées de programmation. Il inclut un système de gestion de projet, de création d'application interactive, une interface au débogueur, et un puissant éditeur de code avec une navigation efficace et de la coloration syntaxique. Ajunta peut être installé à partir Logithèque Ubuntu.

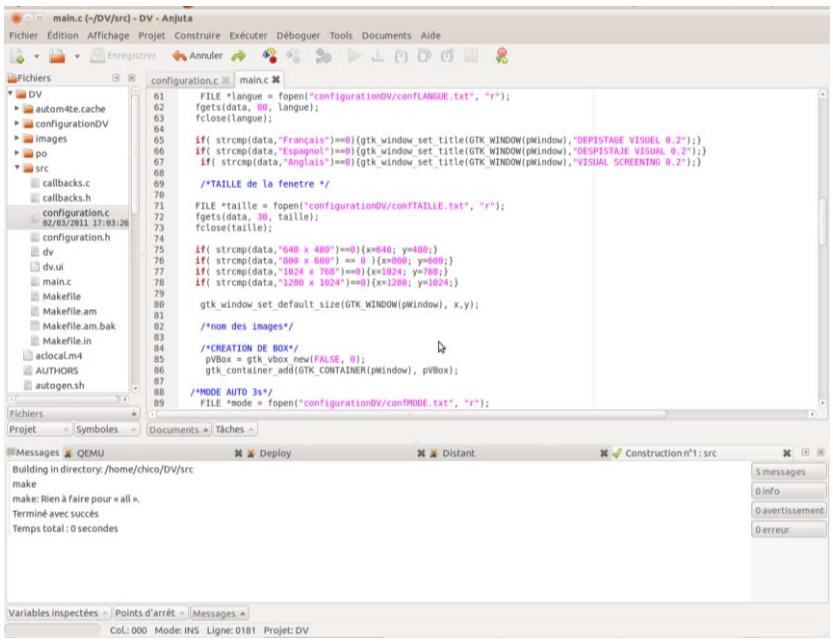
L'installation d'Anjuta inclus tous qu'il faut pour le développement d'une application GTK.

Anjuta permet de créer projets GTK :



Création d'un projet avec anjuta

1. *Démarrez anjuta.*
2. *Créez un projet en choisissant Nouveau Projet dans le menu Fichier.*
3. *Dans l'assistant, cliquez sur 'Suivant' et choisissez GTK Project, puis recliquez sur 'Suivant'*
4. *Saisissez les informations demandées, laissez les options telles qu'elles sont et après quelques clics supplémentaires, vous devriez être en présence d'un message (en bas) vous annonçant que vous pouvez construire votre projet.*
5. *En double-cliquant sur le nom d'un fichier, vous pouvez l'ouvrir dans l'éditeur.*
6. *On construit le projet en choisissant 'Construire' dans le menu Construire (ou F10). Des messages défilent alors dans la zone dédiée à cet effet et vous indiquent finalement, que tout s'est bien déroulé en N secondes.*
7. *On lance l'exécutable généré en choisissant 'Exécuter' dans le même menu (ou F3).*



IV.7.2. Installation de Qt.

Voici une suite d'instructions pour l'installation des logiciels permettant de faire des programmes avec Qt. La version OpenSource de Qt 4 est disponible. On peut la télécharger depuis http://qt.nokia.com/downloads*:*

- ***Pour Windows** : Pour pouvoir exécuter directement les applications, il faut ajouter le path d'exécution (C:\Qt\2010.04\bin) dans le system path Windows XP : Propriétés système à Avancé à Variables d'environnement*
- ***Pour MacOSX** : Il faut également installer XCode.*
- ***Pour Linux** : utiliser le gestionnaire de packages de votre distribution.*

Lors de l'installation, il est important d'installer MinGW. L'installation est particulièrement gourmande : 1 à 2 Go et plusieurs minutes d'installation.

IDE

L'IDE QtCreator est de loin préférable à l'utilisation de tout autre IDE car il intègre plusieurs mécanismes propres à Qt. De plus, il est relativement comparable aux autres IDE. Finalement, on le retrouve sur toutes les plateformes.

Dans ce projet pour avoir un IDE commun avec les autres bibliothèques graphiques j'ai utilisé eclipse pour le développement.

ECLIPSE.

*L'installation est très simple, il existe un plugin spécialement fait pour QT. Dans Eclipse, Aide> Software Updates> Trouver (**http:// artis.imag.fr/ Membres/ Xavier.Decoret/ resources/qt/eclipse/updates**) et installer.*

IV. Conclusion.

- *L'évolution de l'informatique pour l'embarqué permet d'optimiser et réduire les coûts dans le développement de produits pour le dépistage visuel.*
- *L'émulation permet de développer, sur différentes architectures cibles, sans avoir besoin d'un matériel dédié pour chaque architecture (permet de décoller le travail electronique et informatique).*
- *Dans le cadre de ce stage nous avons voulu démontrer qu'il était possible de repenser le développement d'applications en définissant :*
    - *Les facteurs communs à chacune de ces machines (du point de vue matériel et logiciel).*
    - *En choisissant une méthode de programmation la plus portable possible avec l'utilisation d'un langage natif de type C ou C++.*
    - *En trouvant des solutions de prises en main des SDK pour intégrer du code natif.*

V. *Référeces Bibliografiques.*

- **Ouvrage :**
  - ***Pierre Ficheux***. *Linux Embarqué. Paris : EYROLLES*
  - ***Scott Granneman***. *Linux l'essentiel du code et des commandes. PARIS: Pearson.*
  - **Stephen Randy Davis**. C++ pour les nuls. PARIS : FIRST.
  - **Jean-François Pillou.** Développement Logiciel**.** PARIS, Dunod, 2006.
- **Article :**
  - **Fleur Brosseau.** « Virtualisation avec KVM/Qemu ». Linux Pratique (#61). P 50-54.
  - **Fleur Brosseau.** « Le système X-Windows ». Linux Pratique (#61). P 57-61.
  - **Denis Bodor**. « MINI2440». Open silicium (#2). P 6-10.
  - **Alexandre Courbot**. « Noyau». Linux (#132). P 48-55.

VI. VII.Annexes.

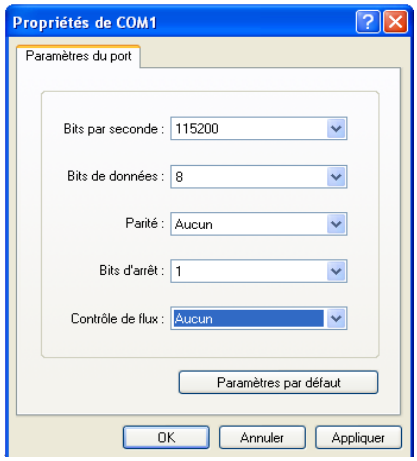
VII.1. Installation d'un OS en utilisant une PC Windows (XP).

Connexion avec HyperTerminal (Windows)

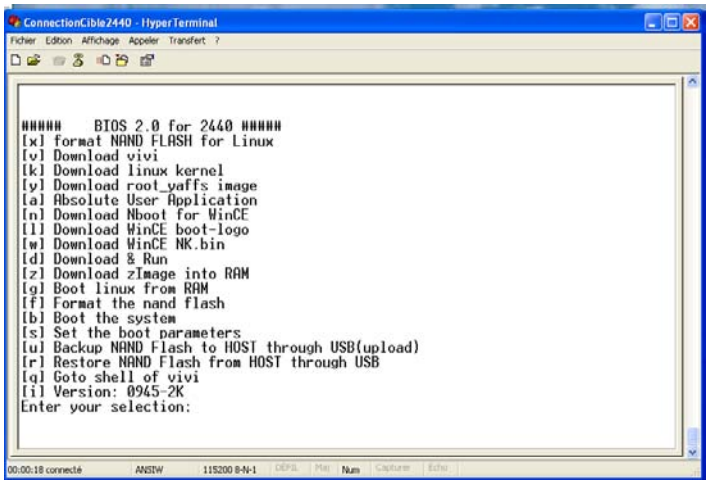
- *On ouvre HyperTerminal*

- *On choisit le port COM à utiliser*

- *On configure la connexion avec les paramètres suivants :*



- *Avec l'interrupteur (S2) en NOR on démarre la carte.*



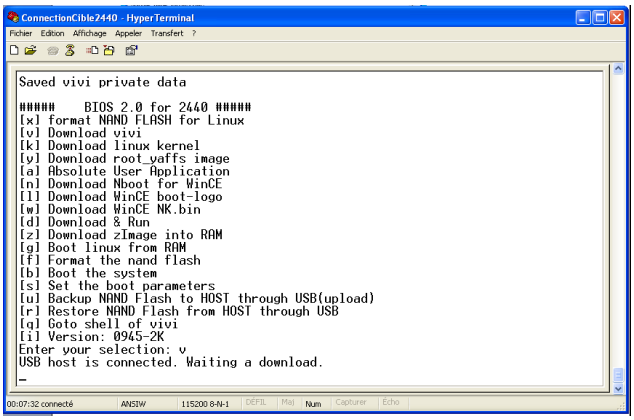
On rentre au BIOS, nommé Supervivi qui vient préinstaller sur la MINI2440. Ce BIOS est issu du vivi bootloader Samsung original, il est implémenté en forme d'une interface de menu. Supervivi peut être flashé soit dans la Flash NOR en utilisant l'interface JTAG ou dans la Flash NAND avec le port série.

L'interface des menus du Supervivi est principalement utilisée pour le flashage des systèmes et le débogage, il peut aussi être utilisé pour le paramétrage et le partitionnement.

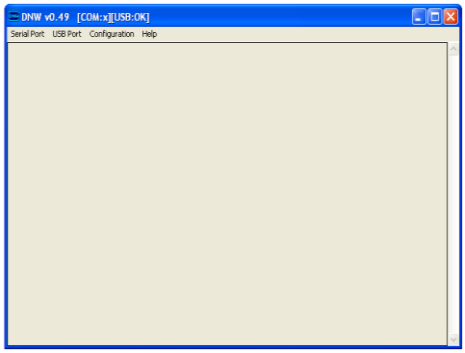
Si le Supervivi a été flashé à la NAND, il peut reconnaître automatiquement le système Linux ou WinCE ou d'autres systèmes qui a été flashe à la Flash NAND et de les exécuter.

Installation de WinCE.

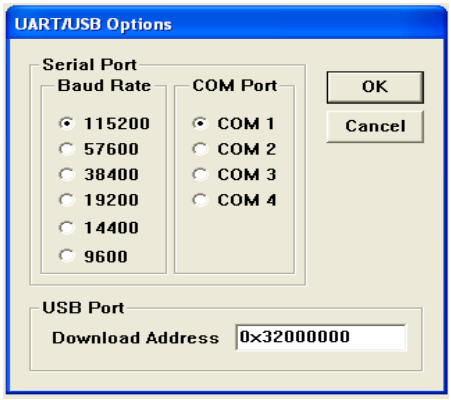
1. ***Connexion à Hyper terminal***
2. ***Boot de la carte à partir de la NOR***
3. ***Formatage de la NAND :*** *Option « **x** »sur le BIOS*
4. ***Installation du Bootloader :*** *Option **« v »***



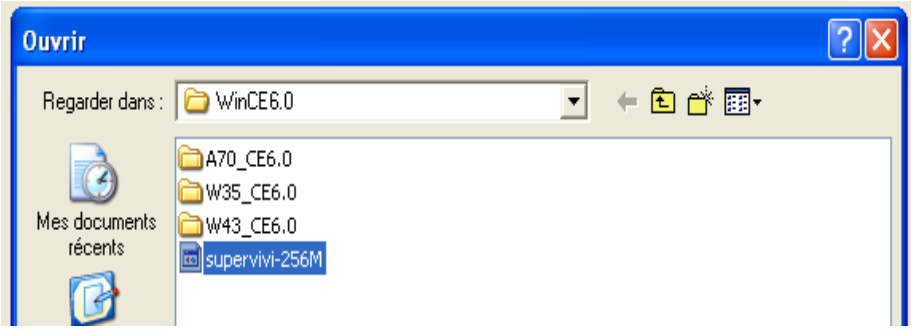
En choisissant l'option « v » (Download Bootloader), ***supervivi*** *se met à l'attend du fichier pour le copier sur la NAND. Les envoies de fichiers se fait via USB avec un logiciel fourni avec la MINI2440, le DNW. Le logiciel n'a pas besoin être installé, c'est une exécutable « dnw.exe » qu'on trouve dans le CD (Windows Tools/dnw) fourni avec la carte. En faisant double clic sur l'exécutable :*



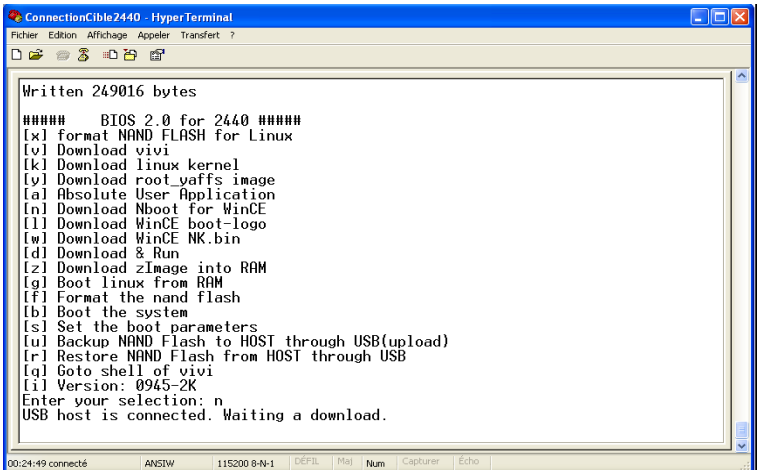
Il faut bien le configurer (configuration->options) :



Pour envoyer le fichier sollicité par l'HyperTerminal on fait « USB port-> Transmit», et on cherche le fichier à envoyer (dans ce cas SUPERVIVI) :



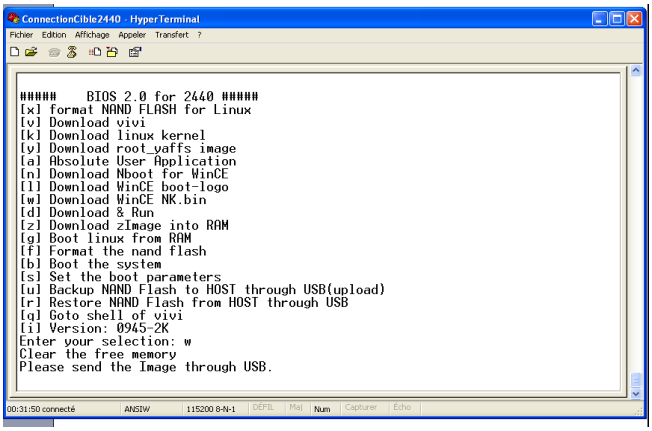
5. ***Installation de Nboot*** *: Option « n » :*



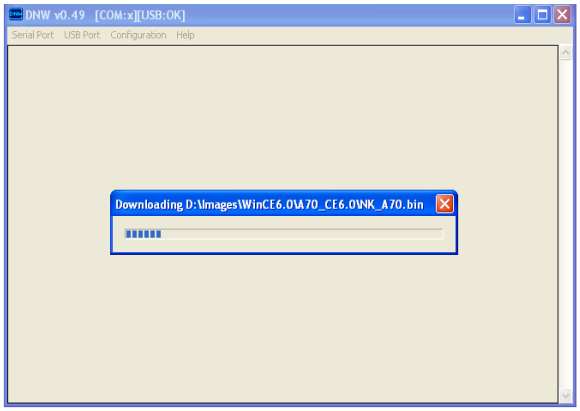
Nboot *est*

De la même façon que pour l'installation du bootloader on utilise le DNW pour envoyer le fichier.

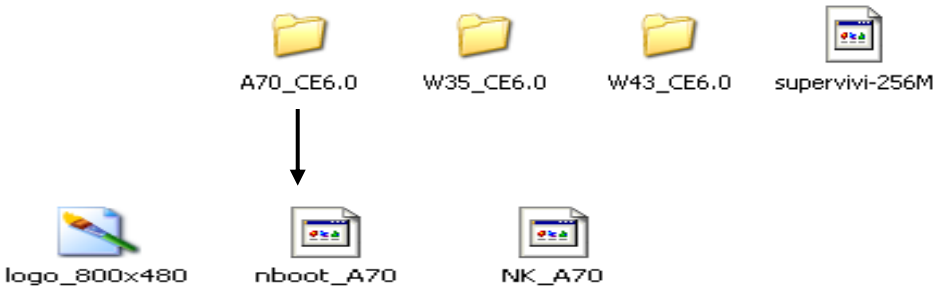
6. Installer WinCE kernel : option « w »



Et en utilisant DNW on envoi le Kernel :



L'image du kernel et du Nboot dépendent des caractéristiquement de la carte, ils existent 3 versions, A70 (écran 7''), W35 (écran 3.5'') et W43 (écran 4.3'') :



Dans mon cas A70.

7. Redémarrer avec la NAND : La MINI2440 démarre avec WINCE.
Installation d'Android.

9. Connexion à Hyper terminal

10. Boot de la carte à partir de la NOR

11. Formatage de la NAND : *Option « **x** »sur le BIOS*

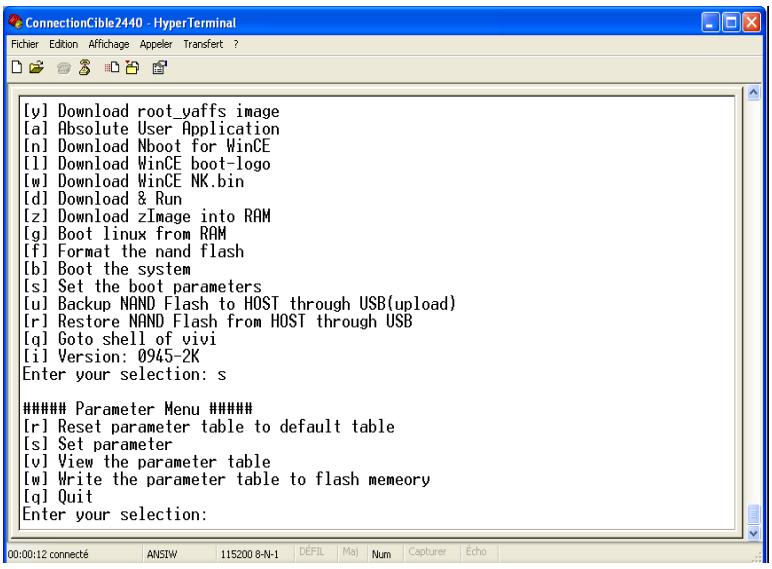
12. Installation du Bootloader : *Option **« v »,** de la même façon que pour l'installation de WINCE on utilise DNW et on envoie le bootloader de notre choix.*

13. Installation du Kernel Linux : *option **« K »***

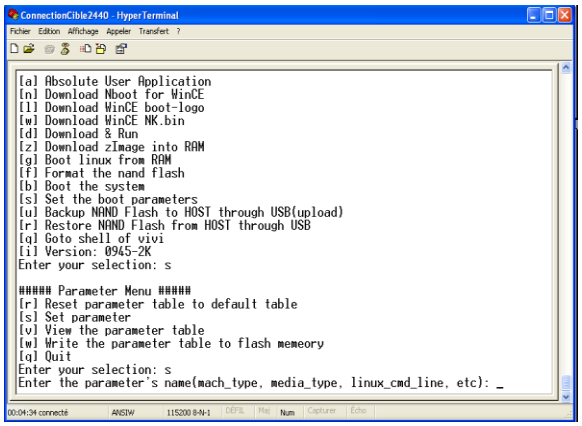
14. Installation du système de fichiers racine Linux : *option **«y »***

15. Configuration du bootloader : *Supervivi permet de passer certains paramètres au kernel pour le bon démarrage du système **:***

- ***Accéder au Sub-menu BOOT PAMETERS*** *: Option **« s »***



- ***Ecrire des paramètres :*** *option* ***« s »***



Tout d'abord on écrit le nom du paramètre, dans notre cas : ***linux_cmd_line****, ensuite on rentre le paramètre* ***(''noinitrd root=/dev/mtdblock2 init=/linuxr console=ttySAC0,115200'')****.*

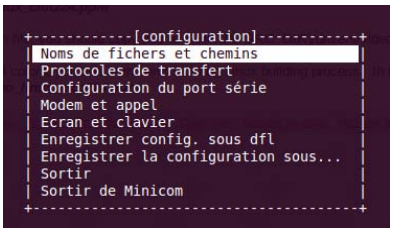
- ***Enregistrer sur le flash*** *: option* ***«w»***

8. ***Redémarrer sur la NAND :*** *La MINI2440 démarre avec Android****.***

Pour l'installation de n'importe quelle autre OS basé sur un noyau Linux on peut procéder de la même façon que pour l'installation d'Android.

Installation d'un OS En utilisant une PC Linux (UBUNTU)

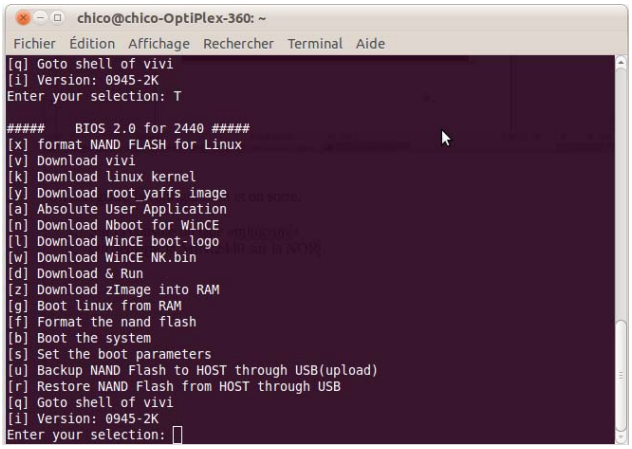
1. ***Configuration de minicom :*** *Dans une console on écrit « minicom -s »*



Dans configuration du port série, on met la configuration suivante :

On enregistre la configuration et on sorte.

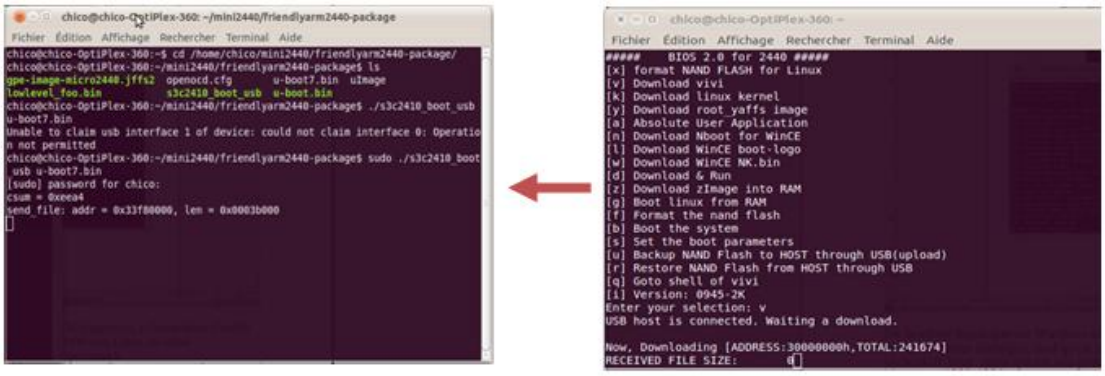
2. ***Sur la console on écrit** «minicom »*
3. ***On démarre la MINI2440 sur la NOR :***



De la même façon que sur Windows on arrive au BIOS (Supervivi), et l'installation d'un OS se fait de la même façon, sauf qu'on n'a pas l'outil DNW sous Linux, on utilise « s3c2410_boot_usb » qui est une petite application open source, téléchargeable sur la net.

Par exemple pour installer un bootloader en utilisant « s3c2410_boot_usb » on fait comme suit :

- *On choisit l'option « v » sur SUPERVIVI*
- *On ouvre un autre terminal et on tape «./s3c2410_boot_usb **nom_du_bootloader** »*



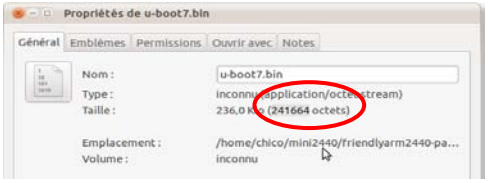
Boot à partir d'une carte SD

Supervivi ne permet pas de faire le boot à partir d'une carte SD, donc pour ce faire j'ai utilisé un autre bootloader U-boot.

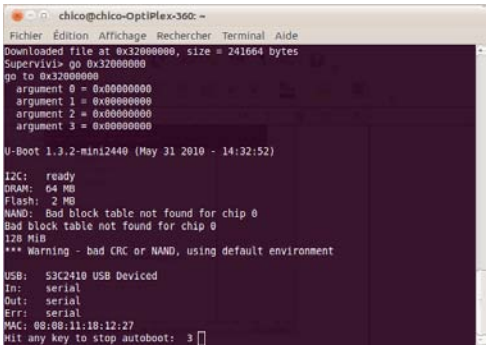
U-boot est un bootloader multiplateforme open source, avec un support complet pour le chargement et la gestion des images de démarrage, tels que le noyau Linux. U-boot est fourni avec la MINI2440.

Installation d'U-boot dans la NAND :

1. ***Démarrer la carte sur la NOR, et sur une console on écrit « minicom »***
2. ***Shell SUPERVIVI :*** *le shell nous permet de spécifier certains paramétrés importants à l'heure du flashage de la NAND, comme la taille du fichier et la position dans la mémoire dans laquelle on veut flasher :*
   - ***Accéder au Shell :*** *option « q »*
   - ***Charger le fichier : « load ram 0x32000000 241664 u »*** *où 241664 est la taille du fichier et 0x32000000 est la position de mémoire où je voudrais stocker le fichier.*

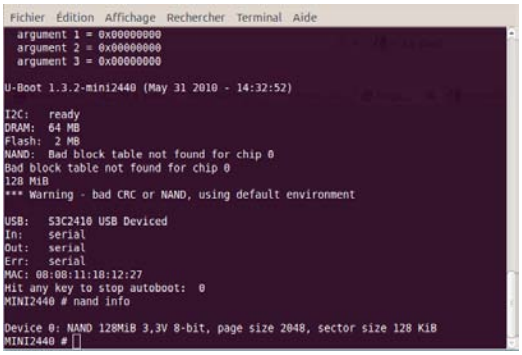


   - ***Envoi du fichier :*** *on ouvre une autre console et avec s3c2410_boot_usb on envoie le fichier :* ***« sudo ./s3c2410_boot_usb u-boot7.bin »,*** *où u-boot7 est le nom du bootloader (dans mon cas).*
   - ***Démarrer U-boot : « go 0x32000000 »,*** *si un délai de démarrage est défini, U-Boot affichera alors le texte "Hit any key to stop autoboot", comme il n'y a aucune OS installé dans tous les cas on arrive à la console d'U-Boot. Cette console permet d'afficher les composantes matérielles, initialiser des variables d'environnement ou démarrer un système d'exploitation.*

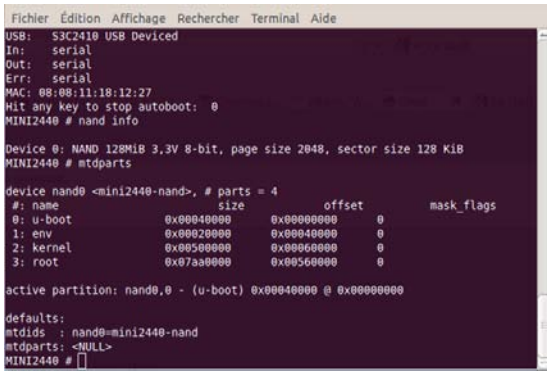


Par exemple :

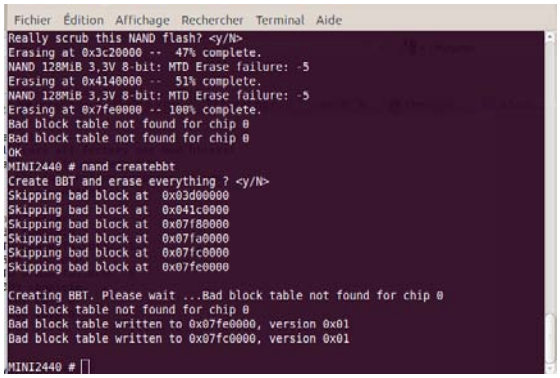
- ***« nand info »*** *:*



- ***« mtdparts »*** *:*



- ***Formatage de la NAND :*** *« nand scrub »*
- ***Création d'un tableau Bad Block (BBT) :*** *Comme tous les dispositifs de mémoire flash NAND, peuvent contenir un certain nombre de défauts de fabrication qui se traduisent par des mauvais blocs, on a besoin de distinguer entre les blocs usine mauvais et usés, pour ce faire in existe un outil qui permet de créer un tableau avec cette information (BBT) :* ***« nand createbbt »***



- ***Initialiser l'emplacement des variables d'environnement : « dynenv set 40000 »***

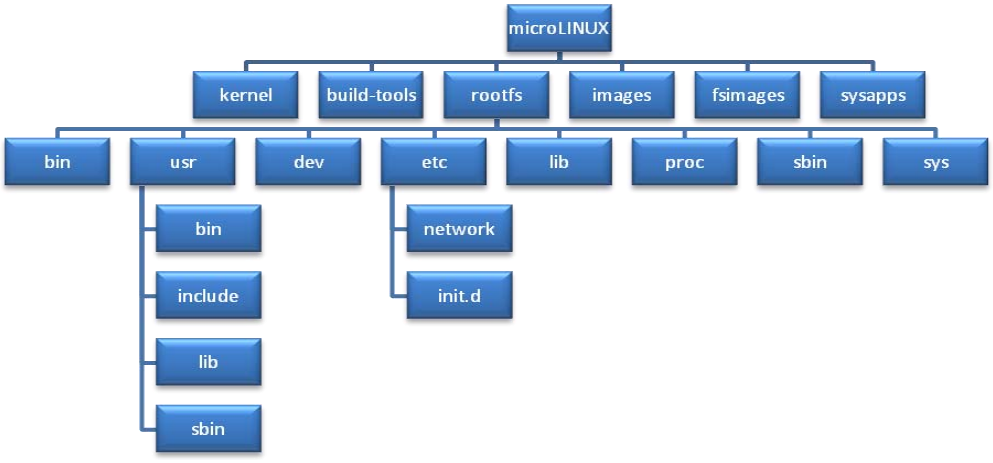
- ***Écrire dans la NAND : « nand write 0x32000000 u-boot ».*** *u-boot est le nom de la partition et non du bootloader que dans mon cas est U-boot7.bin*
- ***Etablir les variables d'environnement :*** *Les deux variables plus importantes sont* ***bootargs et bootcmd.*** ***bootargs*** *permet de passer quelques paramètres au noyau, comme le placement d'init, la console à utiliser, le disque que ferra de racine (pour la carte SD--> mmcblk0p2) et* ***bootcmd*** *l'indique au bootloader où se trouve l'image du noyau. J'ai utilisé la configuration suivante pour un démarrage sur la carte SD, formaté avec 2 partitions, kernel et rootfs, la première partition en format FAT16 pour le kernel et la deuxième en format ext3 pour le système de fichiers racine :*
  - *setenv bootargs console=ttySAC0,115200 root=/dev/mmcblk0p2 rw rootfstype=ext3 mini2440=3tb rootdelay=1 init=/sbin/init*
  - *setenv bootcmd mmcinit \; fatload mmc 0:1 0x32000000 uimage.bin \; bootm 0x32000000.*
- ***Enregistrer : « saveenv »***

VII.2. Creation d'un micro système linux (de l'A à la Z).

Pour la création de notre système embarqué depuis zéro on a utilisé la structure suivante :

a) Créer l'environnement de développement :

On commence par créer l'ensemble de répertoires qui contiendront tout qu'il faut pour la construction de notre système. On peut le résumer de la façon suivante :



Où :

- ***Kernel :*** *Les sources du noyau.*
- ***Build-tools :*** *Tout ce qui est nécessaire pour construire la chaine de compilation croisé*
- ***Rootfs :*** *Système de fichiers du système embarqué.*
- ***Tools :*** *Les outils de compilation croisée*
- ***Images :*** *Les images du noyau.*
- ***Fsimages:*** *Les images du système de fichiers.*
- ***Sysapps :*** *Les applications qu'on veut installer sur le système embarqué.*

b) Créer la 'toolchain' (compilateur) :

La chaîne de compilation a été construite de façon automatique à l'aide de l'outil ptxdist4. Cet outil se charge de télécharger les sources et patches nécessaires pour une chaîne donnée. Il suffit de lui préciser l'architecture cible.

On commence par télécharger la dernier version de PTXdist : les fichiers ***ptxdist-X.X.X.tgz*** *et* ***ptxdist-X.X.X.patches.tgz****, et on le place dans le répertoire build-tools. On les décompresse et on compile le programme.*

```
$tar xvzf ptxdist-X.X.X.tgz
$tar xvzf ptxdist-x.x.x-patches.tgz
```

```
$cd ptxdist-x.x.x
$./configure --prefix=$microLINUX/build-tools
$make
$sudo make install
```

Ptxdist est installé dans ***build-tools/bin****. Après, il faut le configurer à l'aide de la commande* ***ptxdist setup*** *et indiquer où on veut que la chaîne de compilation soit placée une fois qu'elle sera construite. Il faut entrer* ***~microLINUX/tools*** *pour l'option* ***Project SearchPath*** *et* ***~microLINUX/build-tools/src*** *pour* ***Source directory****.*

Ensuite, il faut donner les détails de la chaîne de compilation qu'on veut construire. Pour cela on télécharge un projet PTXdist qui contient les fichiers de configuration qui

fonctionnent très bien pour notre plate-forme (ARM), ***OSELAS.Toolchain-x.x.x.tar.bz2****, on le place dans le répertoire build-tools. On le décompresse et on s'y positionne pour exécuter ptxdist.*

```
$tar xvjf OSELAS.Toolchain-x.x.x.tar.bz2
$cd OSELAS.Toolchain-x.x.x/
```

Le répertoire ptxconfigs contient les fichiers de configuration en question. Celui qu'on a utilisé se nomme :

arm-xscale-linux-gnueabi_gcc-4.1.2_glibc-2.5_linux-2.6.18.ptxconfig

En visualisant son contenu, on observe que la chaîne a les caractéristiques suivantes :

- *Compilateur gcc 4.1.2 supportant C et C++*
- *Bibliothèque libc 2.5 basé sur les en-têtes du noyau 2.6.18 (elle devrait donc fonctionner avec les noyaux récents).*

Ce fichier nécessite quelques adaptations qui dépendent de la version de PTXdist utilisé et de l'emplacement d'installation. La fin du fichier, on trouve :

PTXCONF_CONFIGFILE_VERSION="x.x"

Il faut mettre la version utilisé, et pour les informations sur les répertoires où placer les chaînes de compilation :

PTXCONF_PREFIX_FIRST="/…./….."

On met: " ***microLINUX/tools****"*

On dit à PTXdist quel fichier utiliser pour la configuration :

```
$Ptxdist select ptxconfigs/arm-xscale-linux-gnueabi_gcc-
4.1.2_glibc-2.5_linux-2.6.18.ptxconfig
```

Cette chaîne peut encore être adaptée en tapant ***ptxdist menuconfig***

On lance la compilation de la chaîne à l'aide de ***ptxdist go*** *(depuis le répertoire OSELAS.Toolchain-x.x.x/)*

A la fin de la compilation, la chaine se trouve dans le répertoire ***tools/arm-xscale-linux-gnueabi/gcc-4.1.2-glibc-2.5-kernel-2.6.18/bin.***

En plus cette chaine contient un embryon de système de fichier pour notre système embarqué, un répertoire ***lib*** *qui contient* ***libc*** *déjà compilé pour notre système.*

c) Construire le noyau Linux

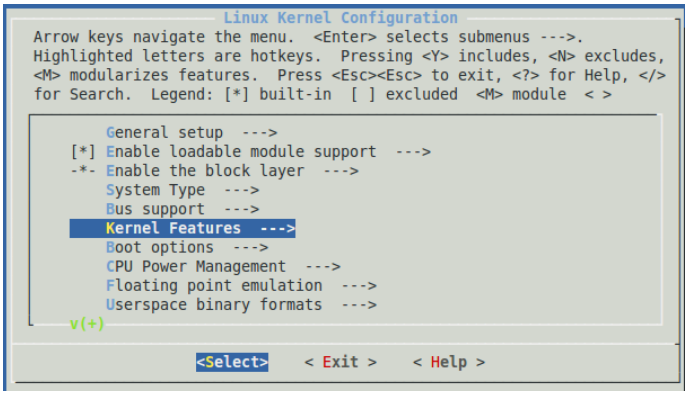
Tout d'abord pour obtenir un système fonctionnel il faut compiler le noyau Linux. On a choisi la dernière version à la date de l'écriture de ce document, 2.6.35. On télécharge les sources linux-2.6.35.tar.bz2, on les met dans le répertoire kernel, et on le décompresse.

```
$tar xvjf linux-2.6.35.tar.bz2
$cd linux-2.6.35/
```

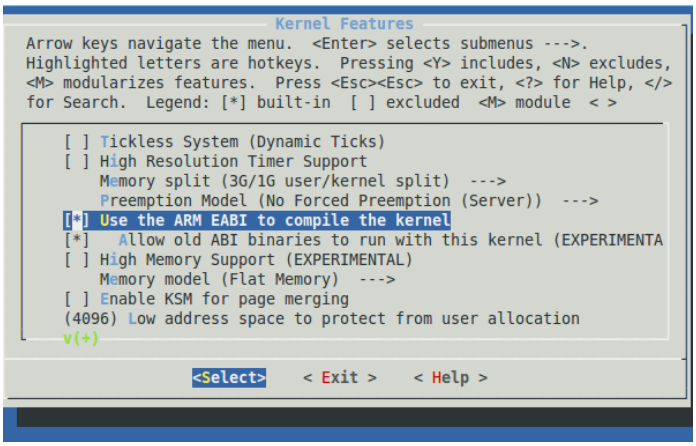
On va implémenter notre système sur une cible ARM et le compilateur et ses différents outils seront exécutés dans un environnement x86 (Linux ou Windows). Il est donc nécessaire de compiler le paquetage à partir des sources en spécifiant les nouvelles options de génération. Pour ce faire, le script configure inclus dans les paquetages permet de spécifier le type d'architecture sur lequel s'exécute l'outil ainsi que le type d'architecture cible (option --ARCH)

```
$make ARCH=versatile-defconfig
$make ARCH=arm menuconfig
```

Avec la commande ***make menuconfig,*** *nous avons accès aux options de configuration en mode console :*



Ici on trouve différentes options pour le bon fonctionnement du système, dans notre cas il faut juste activer le support pour EABI (embedded-application binary interface), dans kernel features -> Use the arm EABI to compile Kernel



EABI est un ABI définie par un consensus entre les fournisseurs de multiples compilateurs des processeurs ARM (ARM, CodeSourcery, IRA, etc).

ABI décrit l'interface de bas niveau entre une application et un système d'exploitation, entre une application et ses bibliothèques, ou entre les composants d'une application.

Ensuite la compilation :

```
$make ARCH=arm CROSS_COMPILE=~/microLINUX/ tools/arm-xscale-
linux-gnueabi/gcc-4.1.2-glibc-2.5-kernel-2.6.18/bin/arm-xscale-
linux-gnueabi-
```

*Une fois la compilation terminée, le noyau (compressé) se trouve dans : **~/microLINUX/kernel/linux-2.6.28/arch/arm/boot,** il s'appelle **zImage** et ne pèse que 1,3 Mo, ce qui est assez approprié pour un système embarqué. On le stocke dans **images/linux-2.6.35**.*

On teste le noyau avec QEMU.

QEMU

Le principal défaut du développement industriel est la nécessité de disposer d'un matériel dédié. Pour implémenter une distribution Linux pour une autre architecture que le

PC/x86 classique, il faudra disposer de la carte adéquate. De même, si on veut développer une application pour cette même architecture, on devra également disposer du matériel.

L'utilisation d'un émulateur comme QEMU résout ce problème. Il ne coûte rien, les sources sont disponibles, il est facile à installer sur les principaux systèmes d'exploitation (Linux, Windows, Mac OS X, UNIX). QEMU est un puissant outil de test et de validation automatique pour des environnements contraints tels que le transport (ferroviaire et aéronautique) ou la défense.

QEMU permet de faire fonctionner plusieurs systèmes d'exploitation sur une seule et même machine, sans avoir besoin de multiplier les disques durs. Pour ce faire, on crée sur le système hôte un disque dur virtuel, un fichier de quelque Mo sur lequel pourra être installé le système invité, virtuel.

QEMU fonctionne sous les systèmes d'exploitation Linux, Mac OS X, Unix et Windows et support les plateformes x86, PPC, Sparc et ARM.

Utilisation.

-boot x	*Indique sur le type de média que vous utilisez pour démarrer l'installation ou le système. Ainsi (à la place de x) a est la disquette, d désigne le Cdrom et c le disque C:*
-m xxx	*Permet d'allouer une part limité de la RAM pour l'usage de Qemu* *Par exemple* ***-m 256*** *limite la taille de la RAM utilisée à 256Mo*
-net user	*Option permettant de gérer le réseau*
-cdrom x	*Prend en charge le lecteur de CD. Par exemple* ***-cdrom /dev/cdrom***
-localtime	*La machine virtuelle est à l'heure locale (au lieu de GMT). À utiliser si vous constatez une différence avec l'heure de votre installation GNU/Linux*
-hda x	*Désigne le disque dur principal :* ***hda*** *correspond au disque C:\,* ***Hdb*** *correspond à D:\, X est le nom donné au fichier image du disque virtuel*

-enable-audio	*Carte son pris en charge.*
-k x	*Change la disposition du clavier : **-k fr** pour avoir le français. Si votre PC est déjà en français, cela n'est normalement pas utile.*
-M	*Type machine à émuler*
-kenel	*L'image du noyau*
-full-screen	*Démarre en mode plein écran*
-qemu	*Commande pour l'architecture x86*
- qemu-system-arm	*Commande pour l'architecture ARM*

La commande suivante permet de tester notre noyau :

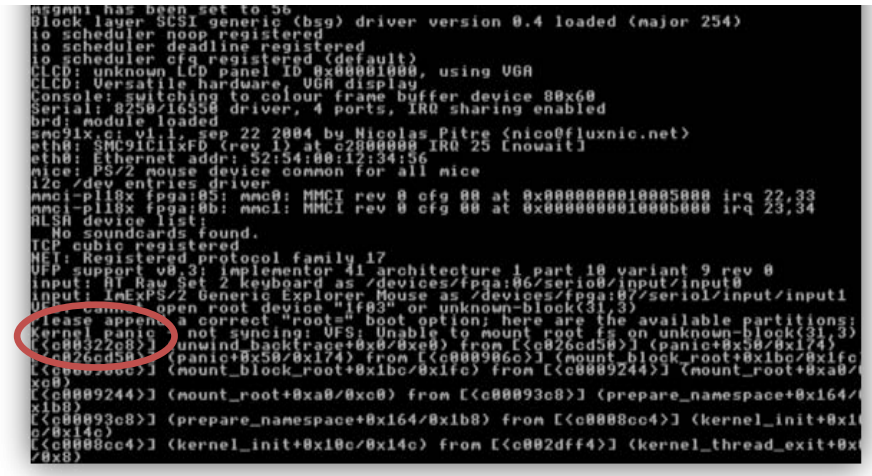
```
$qemu-system-arm  -M versatilepb –m 128M –kernel
     ~/microLINUX/images/linux-2.6.35/zImage
```

La première option (-M) de la commande nous permet d'en savoir plus.

```
$ qemu-system-arm -M ?
Supported machines are:
integratorcp ARM Integrator/CP (ARM926EJ-S) (default)
versatilepb ARM Versatile/PB (ARM926EJ-S)
versatileab ARM Versatile/AB (ARM926EJ-S)
realview   ARM RealView Emulation Baseboard (ARM926EJ-S)
akita      Akita PDA (PXA270)
spitz      Spitz PDA (PXA270)
borzoi     Borzoi PDA (PXA270)
terrier    Terrier PDA (PXA270)
cheetah    Palm Tungsten|E aka. Cheetah PDA (OMAP310)
```

Les noms ci-dessus correspondent à des cartes du marché. QEMU permet donc de simuler entièrement (ou presque) de nombreuses cartes industrielles. Dans notre cas, nous avons choisi la carte ARM Versatile-PB car elle est très bien supportée par Linux.

Voici le résultat :



On observe le démarrage du noyau, qui donne un message d'erreur, ***kernel panic****, car on ne l'a pas encore spécifié le système de fichiers racine.*

Qemu est capable de lancer le noyau directement à partir de son image. Sur le véritable système, cette image est présente sur le système de fichiers, et on utilise un programme d'amorçage (bootloader) pour le récupérer et le charger en mémoire.

d) Construire le système de fichiers racine

Le système de fichiers de notre système embarqué est dans le répertoire ***rootfs****. Le contenu de ce répertoire va être copié sur la carte de mémoire flash qui, une fois insérée dans le système, fera office de racine. Mais dans un premier temps on a transformé ce répertoire en une image de disque, qu'ensuite on peut passer à Qemu pour qu'il la fasse apparaître comme le premier disque dur.*

Tout d'abord on commence par copier les répertoires créés par PTXdist et qui contiennent la libc.

LIBC

La bibliothèque standard de C est une collection maintenant normalisée d'en-têtes et de routines utilisées pour implémenter des opérations courantes, telles que les entrées/sorties et la gestion des chaînes de caractères, dans le langage C.

```
$cd ~/microLINUX/tools/arm-xscale-linux-gnueabi/gcc-4.1.2-glibc-
      2.5-kernel-2.6.18/sysroot-arm-xscale-linux-gnueabi

                $cp -R *  ~/microLINUX /rootfs
```

Ensuite, on ajoute les répertoires manquants :

- ***/etc :*** *Contient les commandes et les fichiers nécessaires à l'administrateur du système* ***(inittab, ld.so.conf)****.*
- ***/sbin et /bin :*** *Les commandes essentielles à la configuration du système sont en général localisées dans le répertoire* ***/sbin****. Le meilleur exemple en est la commande* ***init*** *qui correspond au premier processus lancé par le noyau. Ce répertoire ne contient normalement pas de commande directement accessible à un utilisateur non privilégié. Certaines commandes essentielles sont également situées sur* ***/bin****, comme la commande* ***mount*** *qui permet le montage du système de fichier principal (root filesystem) au démarrage.*
- ***/lib :*** *Ce répertoire contient les bibliothèques partagées indispensables au fonctionnement du système, donc utilisées par les programmes situés sur* ***/bin*** *et* ***/sbin****.*
- ***/usr :*** *Hiérarchie secondaire.*
- ***/usr/bin :*** *Contient la majorité de fichiers binaires et commandes utilisateurs (exemple* **qemu-system-arm** *).*
- ***/usr/include :*** *Fichiers d'en-tête C et C++.*
- ***/var :*** *Le répertoire* ***/var*** *contient des fichiers et sous-répertoires dont l'encombrement peut augmenter fortement au cours du fonctionnement d'un système Linux classique. C'est la raison pour laquelle ce répertoire est habituellement situé sur une partition dédiée. Dans le cas d'un système embarqué, l'utilisation du répertoire sera limitée aux sujets suivants :*
    - *fichiers de fonctionnement ou pid-files,*
    - *fichiers verrous ou lock-files,*
    - *fichiers de trace ou log-files.*

Les deux premiers sont très peu consommateurs en espace. La partie fichier de trace doit être étudiée avec soin. Avec le répertoire ***/tmp****, c'est le seul répertoire dont la taille est susceptible d'augmenter de manière significative durant la vie du système.*

```
$cd ~/microLINUX /rootfs

$mkdir bin dev proc sbin sys tmp var etc/rc.d etc/network
    usr/sbin var/lib var/lock var/log var/run var/tmp
```

Il doit être possible d'écrire et lire dans tmp pour tous les utilisateurs :

```
$chmod 1777 tmp
$chmod 1777 var/tmp
```

e) BusyBox :

Afin d'avoir à notre disposition les outils de base nécessaires au bon fonctionnement d'un système Linux, nous allons utiliser BusyBox, qui, en un seul exécutable de 750 Ko, implémente plusieurs dizaines d'utilitaires. BusyBox implémente même une version simplifiée d'init ***(****est la première commande exécutée par le noyau une fois que le démarrage de celui-ci est terminé. C'est init qui est alors responsable de gérer les logins, de lancer des shells, etc. Tous les processus du système sont donc des descendants d'init****).***

Pour l'installation on copie les sources busybox-1.10.1.tar.bz2 dans ~microLINUX/sysappas/src, les décompresse, les compile à l'aide du compilateur croisé et les installer dans ~microLINUX/rootfs (avec l'outil make menuconfig).

```
$tar xvjf busybox-1.10.1.tar.bz2

$cd busybox-1.10.1/

$make ARCH=arm CROSS_COMPILE=~/microLINUX/ tools/arm-
   xscale-linux-gnueabi/gcc-4.1.2-glibc-2.5-kernel-
   2.6.18/bin/arm-xscale-linux-gnueabi-  defconfig

$make menuconfig

$make ARCH=arm CROSS_COMPILE=~/microLINUX/ tools/arm-
   xscale-linux-gnueabi/gcc-4.1.2-glibc-2.5-kernel-
   2.6.18/bin/arm-xscale-linux-gnueabi-  defconfig
```

f) Configuration d'init

Par défaut, la commande init implémentée dans BusyBox exécute les actions qui correspondent au fichier inittab. On va créer ce fichier inittab (dans ~microLINUX/rootfs/etc/inittab):

```
::sysinit:/etc/rc.d/rcS
tty1::askfirst:/bin/sh
::restart:/sbin/init
```

::ctrlaltdel:/sbin/reboot

::shutdown:/bin/umount -a –r

Au démarrage du système, cet inittab lance le script /etc/rc.d/rcS. On le crée en lui donnant les permissions d'exécution :

!/bin/sh

mount –t proc none /proc

mount –t sysfs none /sys

/sbin/mdev –s

Ifconfig lo up

Ifconfig eth0 XXX.XXX.XXX.XXX netmask XXX.XXX.XXX.XXX

Route add defaut gw XXX.XXX.XXX.XXX

```
$cd ~/microLINUX/rootfs/etc/rc.d

$touch rcS

$chmod a+x rcS
```

g) *Création de l'image*

```
$cd ~/microLINUX/rootfs

$sudo find . | cpio -o  --format=newc | gzip >
../rootfs.img.gz
```

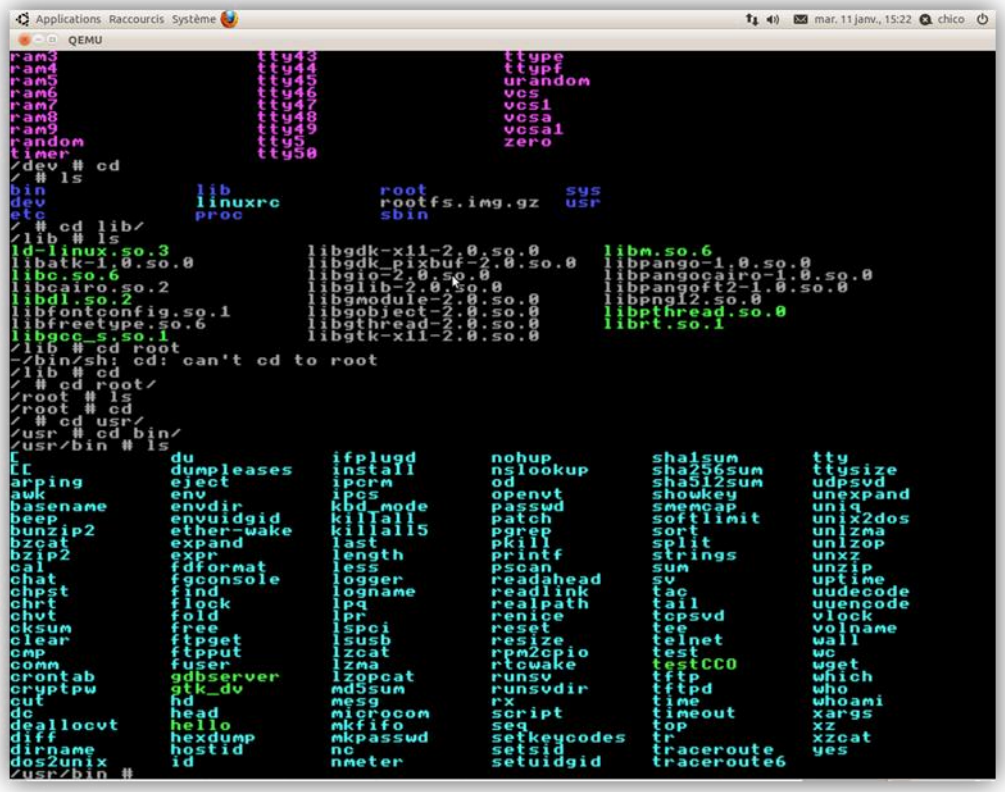
h) *Test*

On lance l'image grâce à la commande suivante :

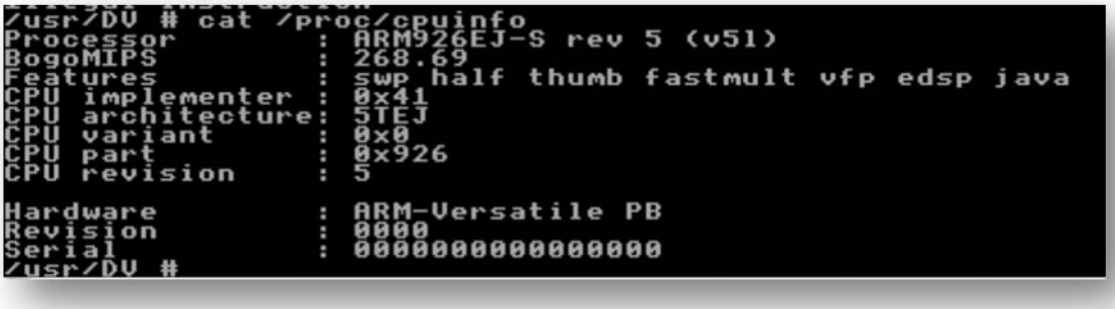
```
$qemu-system-arm -M versatilepb -m 256M -kernel
~/microLINUX/images/linux-2.6.28/zImage -initrd rootfs.img.gz  -
append "root=/dev/sda rdinit=sbin/init"
```

L'option -append permet de passer des paramètres au noyau. Sur le système « réel », c'est le bootloader qui passe ces paramètres au noyau. Ici, Qemu fait office de bootloader. Les options sélectionnées indiquent que c'est /dev/sda qui doit faire office de racine.

Voici le résultat :



On observe comme les commandes classiques comme ls et cd sont réalisées à l'aide de BusyBox, et avec le command ***cat /proc/cpuinfo*** *on peut vérifie les caractéristiques du CPU :*



On voit bien qui s'agit d'un processor ARM, dans ce cas l'ARM926EJ-S, avec un Hardware ARM-Versatile PB.

VII.4. Commandes linux de base.

Éditeurs de texte

- ***emacs*** *: Éditeur de texte*
- ***nano*** *: Éditeur de texte*
- ***vi*** *: Éditeur de texte présent dans presque tous les systèmes Unix.*

Fichiers et répertoires

- ***cd*** *: Change le répertoire courant. (Change Directory)*
- ***cp*** *: Copie un fichier, peut copier une liste de fichiers dans un autre répertoire en conservant leur nom.*
- ***dd:*** *Effectue une copie d'un fichier avec possibilité de conversion du format .*
- ***dir*** *: Equivalent à ls, n'existait pas à l'origine d'UNIX*
- ***du*** *: Affiche l'utilisation du disque.*
- ***df*** *: Affiche l'utilisation des disques.*
- ***ln*** *: Crée un lien avec un autre dossier ou fichier.*
- ***ls*** *: Affiche la liste des fichiers dans le dossier courant ou d'un autre dossier.*
- ***mkdir*** *: Crée un ou plusieurs répertoires*
- ***mv*** *: Déplace (ou renomme) un fichier, y compris si c'est un répertoire, peut déplacer une liste de fichiers dans un autre répertoire en conservant leur nom.*
- ***pwd*** *: Affiche le chemin du dossier courant.*
- ***rm*** *: Supprime un/des fichier(s) ou des répertoires (avec l'option -r).*
- ***touch*** *: Change la date de modification d'un fichier, en le créant s'il n'existait pas.*
- ***lsof*** *: Affiche la liste des fichiers ouverts.*

Manipulations d'archives et compressions

- ***bzip2/bunzip2 :*** *Compresse et décompresse des fichiers*
- ***cpio :*** *Copie de fichiers à partir de ou vers une archive cpio/tar*
- ***gzip/gunzip :*** *Compresse et décompresse des fichiers. (Gnu ZIPper)*
- ***tar*** *: Archiveur, capable de fonctionner avec bzip2 ou gzip.*
- ***zip/unzip :*** *Compresse et décompresse des fichiers. (ZIP)*

Gestion des disques/points de montage

- ***mount*** *: Attache un système de fichiers sur un point de montage.*
- ***umount*** *: Détache un système de fichiers.*

Manipulation de texte

- ***cat*** *: Concatène des fichiers texte. Peut aussi servir à simplement afficher ou lire un fichier.*
- ***echo*** *: Affiche une ligne de texte donnée en paramètre.*
- ***grep et egrep*** *: Affiche les lignes qui contiennent une expression régulière donnée, egrep (grep étendu) est plus riche en possibilités.*
- ***tail*** *: Affiche les dernières lignes d'un fichier. (opposé de head)*

Permissions

- ***chmod*** *: Change les permissions en lecture, écriture et/ou exécution d'un fichier.*
- ***chown*** *: Change le propriétaire d'un fichier.*

Shells

- ***bash*** *(GNU) : Shell compatible sh de GNU (Bourne Again Shell)*
- ***sh*** *: Shell standard (bourne Shell)*

Utilisateurs

- ***su*** *: Commence un nouveau shell ou une autre commande en changeant l'utilisateur. (Super-User, Switch User)*
- ***sudo*** *: Exécute un processus avec les droits d'un autre utilisateur selon les règles définies dans le fichier /etc/sudoers*

Compilation

- ***gcc -o prog prog.c :*** *Compilation C du fichier prog.c, exécutable dans le fichier prog*