Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Domaine Sciences de l'ingénieur
Rte du Rawyl 47
CH- 1950 Sion 2
Tél. +41 27 606 85 11
Fax +41 27 606 85 75
info@hevs.ch

www.hevs.ch

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2007

Michaël Fournier

Système embarqué iPresenter2

Professeur	Pierre Pompili
Expert	Daniel Rossier

Sion, le 23 novembre 2007

HES-SO Valais

SI	TV	EE	IG	EST
X	X	X	X	

Données du travail de diplôme
Daten der Diplomarbeit

FO.0.2.02.07.DB
DD/18/05/2006

Filière / *Studiengang* : Systèmes industriels

Confidentiel / *Vertraulich* ☐

Etudiant / *Student*	**Année scolaire / *Schuljahr***	**No TD / *Nr. DA***
Michaël Fournier	2006/07	SI/2007/5

Proposé par / *vorgeschlagen von*	**Lieu d'exécution / *Ausführungsort***
HES-SO Valais, UIT	HES-SO Valais, DSI
	Expert / *Experte*

Titre / *Titel*:

Système embarqué iPresenter2

Description / *Beschreibung*:

L'unité Infotronics a conçu et développé un système embarqué, appelé iPresenter, capable de lire des fichiers JPEG depuis une clé USB et de les projeter sur un écran de présentation à l'aide d'un Beamer connecté à la sortie XVGA ou sur un poste de télévision à l'aide d'un câble idoine connecté à la sortie XVGA.

L'iPresenter actuel ne permet pas de projeter une présentation au format PowerPoint natif. Une présentation PowerPoint doit être traduite au préalable en une suite de fichier JPEG (un fichier par page de présentation) avant que l'iPresenter ne puisse jouer son rôle de présentateur.

Windows Mobile offre *PowerPoint Mobile* pour lire des présentations PowerPoint natives sans grandes restrictions. Le but de ce projet est de développer une nouvelle version de l'iPresenter capable de lire les présentations PowerPoint natives depuis une clé USB et de les projeter sur un écran de présentation, à l'aide d'un Beamer connecté sur une sortie XVGA.

Objectifs / *Ziele:*

- Concevoir l'électronique de l'iPresenter2 qui intègrera la carte fille Colibri PXA270 ou PXA320 de Toradex
- Adapter les fonctions de bas niveau pour prendre en compte les entrées-sorties de l'iPresenter2
- Porter Windows Mobile sur l'iPresenter2
- Démontrer les fonctions définies pour l'iPresenter2.

Signature ou visa / *Unterschrift oder Visum*	**Délais / *Termine***
Resp. de l'orientation ~~infotronics~~ Pompili R	Attribution du thème / Ausgabe des Auftrags: 03.09.2007
Professeur/*Dozent*: ~~Pierre~~ Pompili Pompili R	Remise du rapport / Abgabe des Schlussberichts: 23.11.2007
Etudiant/*Student*: M. Fournier	Exposition publique / Ausstellung Diplomarbeiten: 30.11.2007
	Défenses orales / Mündliche Verfechtungen Semaine 49

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats:* .....

Fournier Michaël vendredi, le 23 novembre 2007

Système embarqué - iPresenter2

Embedded System - iPresenter2

Objectif

Réaliser une carte électronique autonome embarquant l'un des modules Colibri PXA270/320 de la société Toradex. La carte devra mettre à disposition une sortie graphique, deux ports USB et quatre boutons.

Les modules devront intégrer le système d'exploitation Windows CE 5. Des fonctions de bas-niveau devront être développées pour contrôler les entrées-sorties de la carte. Un logiciel permettant la lecture de fichiers Powerpoint devra être intégré au système.

Résultats

Une carte électronique fonctionnant sur batterie a été développée et testée. Elle supporte les deux modules PXA270 et PXA320. Les entrées/sorties nécessaires sont fonctionnelles.

Au niveau logiciel, des librairies de liaisons dynamiques (dll) ont été développées pour prendre en compte les entrées/sorties de la carte. Une nouvelle image de Windows CE 5 intégrant le logiciel « Powerpoint Mobile » a été portée. Un programme permettant le pilotage de l'appareil a été conçu.

Mots-clés

iPresenter2, Powerpoint, Windows CE 5, Toradex, Colibri, PXA270, PXA320, drivers.

Ziel

Eine autonome elektronische Karte mit Colibri PXA270/320 Modulen von Toradex realisieren. Diese Karte wird über einen grafischen Ausgang und zwei USB Ausgänge verfügen müssen.

Die Module werden mit einem Windows CE 5 Betriebsystem funktionieren. Die Eingänge/Ausgänge der Karte müssen von low-level Funktionen kontrolliert werden. Eine Software, um Powerpoints zu lesen, soll im System integriert werden.

Resultate

Die elektronische Karte, die mit Batterien funktioniert, wurde entwickelt und getestet. Sie unterstützt die Module PXA270 und PXA320. Die Eingänge/Ausgänge funktionieren sehr gut.

Die Eingänge/Ausgänge werden durch dlls (dynamic link libraries) kontrolliert. Ein neues Windows CE 5 Image mit „Powerpoint Mobile" wurde portiert. Ein „MainMenu" Programm wurde auch entwickelt.

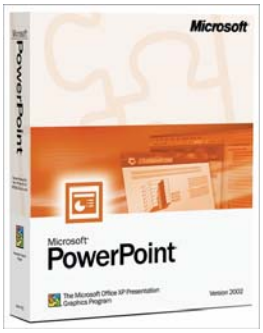
Schlüsselwörter

iPresenter2, Powerpoint, Windows CE 5, Toradex, Colibri, PXA270, PXA320, drivers.

Table des matières

1. Introduction

Rares sont les cadres et autres travailleurs du secteur tertiaire, aujourd'hui, qui n'ont jamais réalisé de présentations sur projecteur vidéo. Celles-ci sont rapides à mettre en œuvre, d'une grande qualité et propreté, et permettent d'intégrer des images, vidéos, animations ou autres contenus multimédia. La plupart des salles de présentation sont aujourd'hui équipées de « beamers », ceux-ci pouvant être connectés soit sur un poste informatique fixe présent sur les lieux, soit sur un ordinateur portable. La seconde possibilité est surtout utilisée par les utilisateurs nomades, qui connectent leur machine et n'utilisent le projecteur que pour la durée de leur présentation.



Logiciel de référence en la matière, Microsoft Powerpoint, s'est imposé et reste le logiciel le plus utilisé à ce jour sur les ordinateurs du type PC / Windows. Il a l'avantage d'être rétro compatible, ce qui permet de peu voir de ne pas se soucier des problèmes de changement de versions ou de machines qui peuvent survenir. Malgré cela, la politique actuelle de Microsoft s'appliquant aussi à Powerpoint, le codage des fichiers créés reste propriétaire, et peu de logiciels alternatifs sont capables de les lire. OpenOffice est le plus connu d'entre eux et comporte un atout non négligeable : il est libre de droit et sous licence GPL (licence publique générale).

Il est intéressant de remarquer que dans le cadre de sa politique de transparence, le format de stockage utilisé par OpenOffice est ouvert et documenté. Il a été standardisé sous le nom d'*OASIS Open Document Format* comme format bureautique universel par un organisme de normalisation. Microsoft va finalement soutenir ce format universel, ce qui permettra, par exemple, de développer des solutions tierces pour lire les présentations informatiques.

Malgré toutes les qualités de ces nouveaux outils de travail, il demeure certains problèmes, avant tout dus aux choix techniques et à la complexité croissante du matériel informatique. Comme le témoigne l'image ci-contre, la connexion d'un projecteur vidéo à un ordinateur portable semble simple : un câble suffit.

Or, on peut constater que les personnes effectuant des présentations à l'extérieur de leur entreprise se retrouvent de plus en plus souvent confrontées aux difficultés suivantes (engendrant du stress et de désagréables surprises) :

- problèmes de compatibilité des résolutions d'affichage entre le projecteur et l'ordinateur
- difficultés d'activer la sortie graphique externe de l'ordinateur portable
- difficultés de configurer le bon mode d'affichage (clone) entre l'écran de l'ordinateur portable et la sortie graphique externe
- Difficultés liées au logiciel de présentation ou à Windows

Il existe des solutions alternatives à l'ordinateur portable, telles que l'assistant personnel PDA, auquel on adjoint un connecteur pour le brancher à un projecteur. On réduit là quelque peu la complexité des outils, mais l'on en créée de nouvelles (nouveau système d'exploitation à maitriser (Windows CE), configuration de la nouvelle sortie écran, résolutions utilisées, prix très élevé, etc.)

Il faudrait disposer d'un appareil portatif dont les atouts principaux seraient :

- La simplicité d'utilisation
- la compatibilité avec la majorité des projecteurs vidéo du marché
- une configuration minimale voir nulle de l'appareil
- une taille minimale, facile à emporter, autonome
- une possibilité simple de stocker ou de transférer des présentations

2. Présentation des objectifs

La HES-So Valais, consciente de l'existence d'un marché sur ce secteur en pleine croissance, a déjà développé un prototype nommé iPresenter. Celui-ci permet d'afficher des présentations sur un projecteur vidéo en passant par une sortie VGA analogique. Equipé de batteries, il peut être emporté facilement et ne nécessite aucun câble supplémentaire. Il est possible de lui connecter des clés de stockage USB et d'en extraire les présentations. De plus, la simplicité d'utilisation a été poussée à son maximum : l'appareil est muni d'uniquement 4 boutons poussoirs permettant de naviguer entre les fichiers de la clé et de faire défiler la présentation.

Quelques défauts majeurs sont cependant existants : le plus gênant est l'impossibilité de lire de manière native les fichiers de présentation Powerpoint. L'iPresenter est basé sur un système Linux. De ce fait, il ne peut lire les fichiers basés sur le format propriétaire de Microsoft. Une conversion en une suite d'images est nécessaire. Cette opération est lourde et peut vite devenir pénible pour l'utilisateur final.

De plus, il est alimenté par piles et ne peut être transporté dans une poche de chemise, de par ses dimensions. Ces petits défauts peuvent aussi s'avérer gênants.

Le but de ce travail est de réaliser un iPresenter2, nouvelle version de cet appareil dépourvue des défauts cités ci-dessus.

Les solutions existantes permettant de lire nativement les fichiers au format de Microsoft Powerpoint sur un appareil mobile sont les suivantes :

- Utiliser l'OS Linux et extraire le code source d'OpenOffice pour utiliser les fichiers Powerpoint
- Utiliser l'OS Windows CE Mobile avec le logiciel Microsoft Powerpoint Mobile
- Utiliser un logiciel tierce partie fonctionnant sur Linux / Windows CE / …

Les logiciels tournant sous Linux et permettant de gérer les présentations sont tous compatibles avec le format Open Document. Malheureusement, seul OpenOffice est capable d'ouvrir les fichiers propriétaires Microsoft. Il serait donc possible, vu la nature libre de celui-ci, d'extraire une partie du code source liée aux présentations et de l'adapter pour notre application. La complexité et le temps à investir pour cette tâche est énorme, sans compter que Microsoft pourrait changer son format propriétaire.

Une autre possibilité envisageable constitue à changer de système d'exploitation. Windows CE est l'alternative de Microsoft, optimisée pour les systèmes embarqués. Dans sa version Mobile 5, puis 6, il est fourni par défaut avec Powerpoint Mobile, qui est aussi la version embarquée du célèbre logiciel de présentation. Si celle-ci ne s'avèrerait pas convaincante, il existe plusieurs logiciels tiers permettant la visualisation de fichiers Powerpoint fonctionnant sous Windows CE.

Avec le processeur actuel de l'iPresenter, il n'est pas imaginable de faire tourner convenablement un système Windows embarqué, les ressources étant trop limitées. Toradex, une entreprise suisse basée à Lucerne et qui opère dans le secteur des technologies d'information, commercialise des modules nommés Colibri. Ceux-ci sont capables de faire tourner des systèmes d'exploitation basés sur Windows CE.

C'est cette alternative qui a été choisie. L'objectif premier de ce travail est donc de **réaliser un prototype de l'iPresenter2, en utilisant la carte processeur de Toradex tournant sous Windows CE. Deux ports USB devront être placés sur celle-ci, ainsi qu'une sortie écran. De plus, elle devra être alimentée par une batterie capable de faire fonctionner l'appareil pendant au moins deux heures. La charge de la batterie sera réalisée via un transformateur secteur (un connecteur supplémentaire sera présent sur la carte pour brancher celui-ci).**

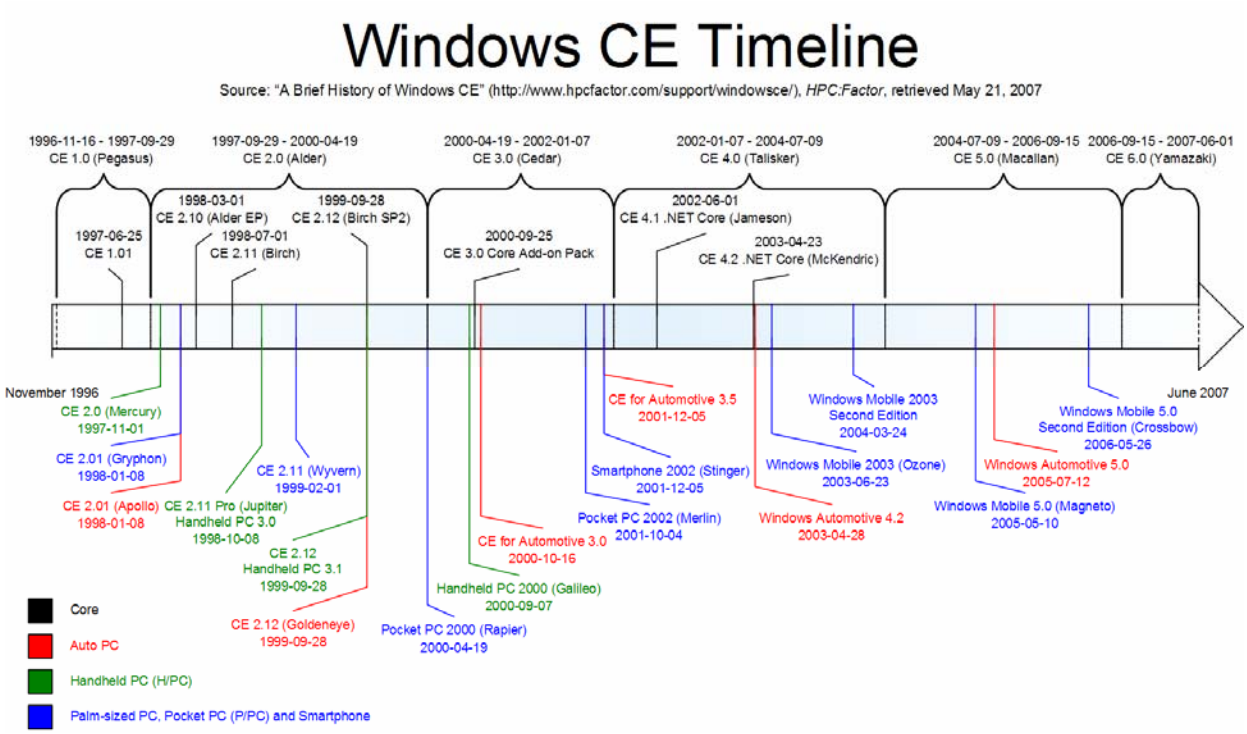
3. Windows CE et ses dérivés

Windows CE est une variation de Windows pour les systèmes embarqués et autres systèmes minimalistes, utilisée notamment dans les PC de poche ou Handheld. Il utilise un noyau distinct des autres Windows plutôt qu'une version allégée et supporte les architectures processeur Intel x86 et similaires, MIPS (jusqu'à CE 3.0), ARM et aussi Hitachi SH.

Windows CE est optimisé pour les appareils possédant une faible capacité de stockage - le noyau peut tourner avec moins d'un méga-octet de mémoire vive. Les systèmes sont souvent produits sans disque de stockage et peuvent être pensés pour ne pas pouvoir être étendus (par exemple, le système peut être écrit en ROM).

Depuis, Windows CE a évolué en un système d'exploitation basé sur des composants, embarqué et temps réel. Il n'est plus réservé aux seuls ordinateurs tenant dans la main. Beaucoup de plates-formes sont basées sur le cœur du système Windows CE telles que Microsoft's Handheld PC, Pocket PC, Smartphone et Mobile.

Le graphique ci-dessous représente les différentes versions de Windows CE (Cœur) et pour chacune d'entre elles les versions dérivées dédiées à certaines applications (Automobile, Ordinateur tenant dans la main, PocketPC, téléphones portables).



Windows Mobile 5.0 est une version répondant au nom de code "Magneto" lancée en mai 2005. C'est l'évolution de Microsoft Pocket PC. Il s'agit en fait de Windows CE 5.0 fonctionnant avec le framework .NET. Cet OS inclut entre autre une nouvelle version de Microsoft Office nommée "Office Mobile" incluant PowerPoint Mobile.

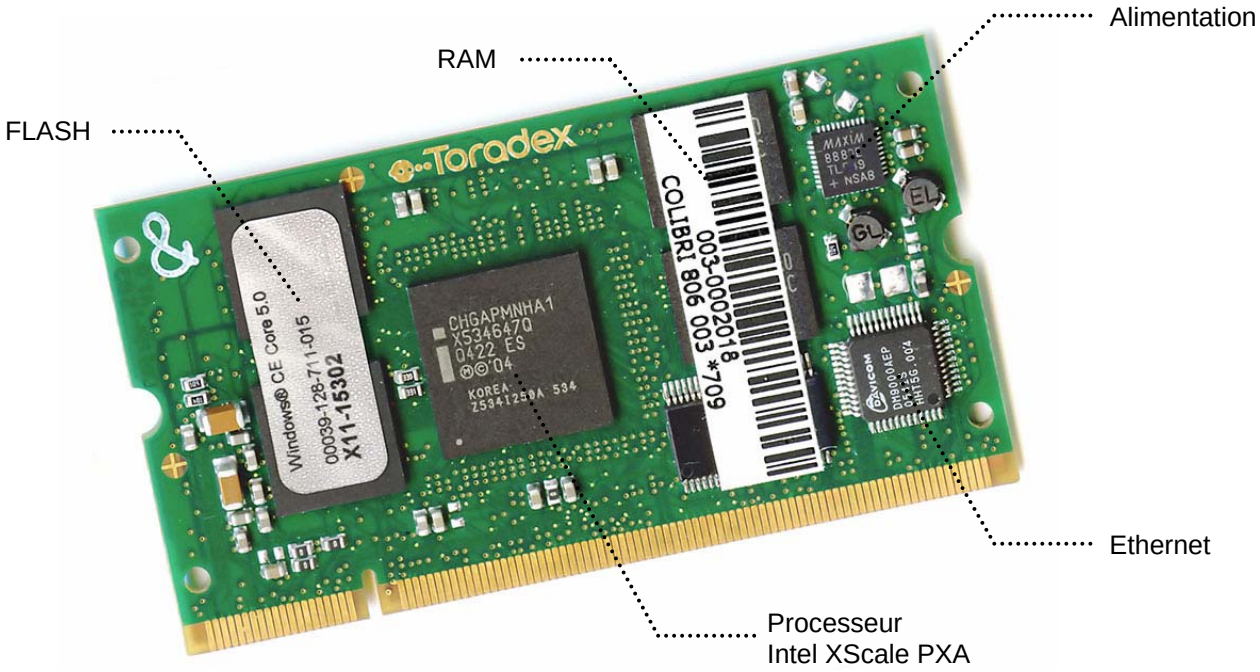
La dernière version existante de Windows CE nommée Windows Embedded CE 6.0 permet de faire tourner simultanément jusqu'à 32'000 processus au lieu de 32 sur les autres versions. De plus, il est possible d'allouer 2 GB de mémoire par processus au lieu de 32MB pour les versions antérieures.

Source : Wikipédia

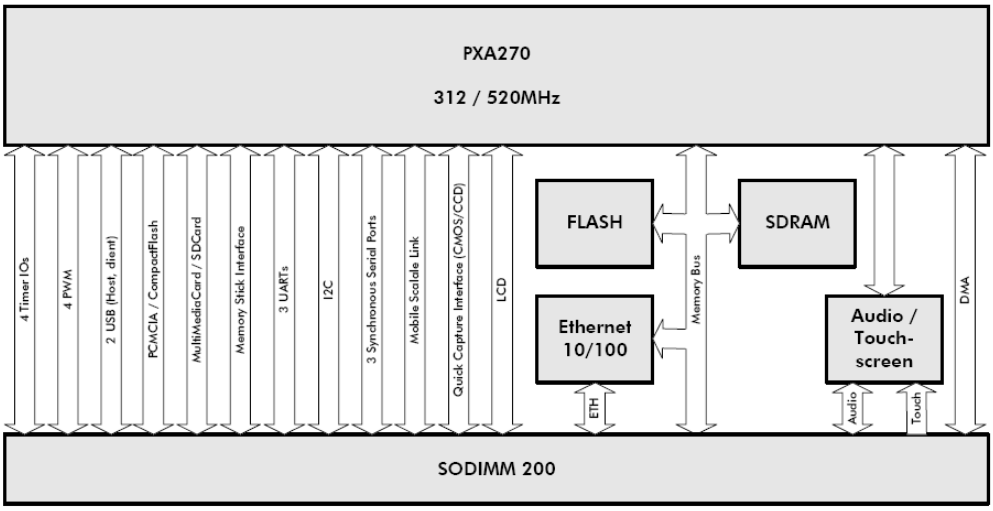
4. Les modules Toradex

4.1. Colibri PXA270 & PXA320

Comme décrit dans la présentation des objectifs, le cœur de la carte à développer est constitué par un module complet, faisant office de microcontrôleur. Cette carte électronique, aux allures de barrette de mémoire RAM d'ordinateur portable, est constituée principalement d'un processeur Intel XScale PXA.



Voici le schéma bloc du module : on constate bien la présence du processeur, de mémoire RAM et FLASH, ainsi que de 2 éléments extérieurs qui complètement les fonctionnalités de la carte : un contrôleur Ethernet et un contrôleur audio + écran tactile (se trouve au dos de la carte sur la photo).



Il existe actuellement deux versions de cette carte qui sont susceptibles de nous intéresser : le PXA270 et le PXA320. Les processeurs utilisés dans chacune d'entre elles sont différents, mais tout le reste est identique. De plus, le pin out du connecteur est compatible, ce qui permet, indifféremment, d'utiliser l'une ou l'autre carte processeur.

Les fonctionnalités ainsi que la puissance de calcul offertes par les deux modules sont différentes. Voici un descriptif complet :

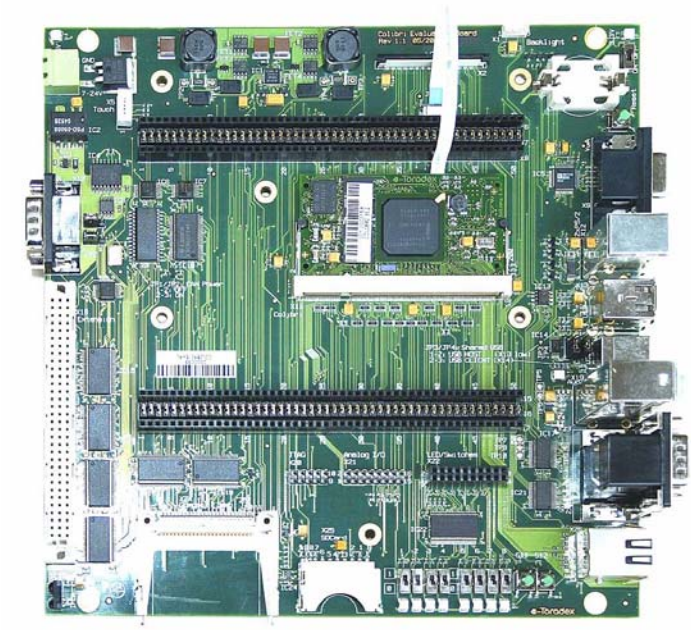
Colibri PXA 270	Colibri PXA 320
	
CPU Marvell PXA270 @ 520 MHz	**CPU** Marvell PXA320 @ 806 MHz
Mémoire 64MB SDRAM (32Bit) 32MB Flash (32Bit)	**Mémoire** 128MB DDR RAM (32Bit) 1GB Flash (8Bit)
Connectivité Bus externe 32 bits I2C / SPI CompactFlash 2xUART IrDA 100MBit Ethernet Memory Stick 1xSDCard 1xUSB Host 1xUSB Host/Device	**Connectivité** Bus externe 16 bits I2C / SPI / One-Wire CompactFlash 2xUART IrDA 100MBit Ethernet Memory Stick 2xSDCard 1xUSB Host 1xUSB Host/Device UTMI (USB 2.0 Device)
Entrées / sorties à usage général Nombre : 85	**Entrées / sorties à usage général** Nombre : 113
Interfaces multimédia LCD (XGA, 1024x768) Ecran tactile Interface caméra Entrées / sorties audio PWM	
Taille 67.6x36.7x5.2 mm Connecteur SO-DIMM 200 pins DDR1	
Logiciel Windows CE 5.0 (préinstallé) Support Linux et QNX	

On remarque la présence de deux ports USB, d'une interface pour écran LCD, d'entrées/sorties à usage général ainsi que d'une interface pour mémoire externe sur les deux cartes, ce qui correspond à nos besoins. La mémoire flash disponible diffère. Une étude plus poussée des besoins en mémoire sera nécessaire.

On constate aussi que la version 5.0 de Windows CE est préinstallée sur les modules. Cela simplifiera sûrement une partie du développement logiciel (pilotes graphiques, USB, …)

4.2. La carte d'évaluation

Toradex propose, en parallèle à ses modèles Colibri, une carte d'évaluation compatible avec les modèles PXA270 et PXA320. Celle-ci permet d'exploiter toutes les ressources mises à disposition par les modules. De plus, par une liaison USB à l'ordinateur, elle permet la programmation, le téléchargement de données, ainsi que le débogage des Colibri.



Voici les interfaces des modules Colibris qui sont implémentées sur la carte, et utilisables directement :

- 1 connexion Ethernet 10/100 Mbits/s
- 1 port USB Hôte
- 1 port USB Hôte / Client
- 2 connecteurs PS/2 (clavier, souris)
- 1 sortie écran VGA analogique
- 1 sortie écran numérique TFT: Philips LB064V02–A1
- 3 connecteurs audio : Line–In, Line–Out, Mic–In
- 1 émetteur/récepteur infrarouge IrDA
- 2 ports RS232
- 1 connecteur pour bus CAN (Philips SJA1000)
- 1 lecteur de cartes SDCard
- 1 lecteur de cartes Compact Flash

5. Spécification

Lors de la réalisation d'un projet de cette taille, il est très important de fixer clairement et complètement les spécifications à respecter. Dans notre cas, elle se divise en quatre aspects principaux.

5.1. Spécification fonctionnelle

L'iPresenter2 devra être capable :

- d'afficher des présentations du type Microsoft Powerpoint (version 97, 2000, XP et 2003) sur un projecteur vidéo dans une résolution standard de 800x600 pixels
- de lire le contenu de clés ou de périphériques de stockage USB
- de lire les présentations de manière native sans conversion de formats
- d'être totalement piloté via 4 boutons poussoirs
- de supporter le branchement d'une clé USB Wifi pour l'envoi de données sans fil sur un projecteur vidéo
- de disposer d'une quantité suffisante de mémoire interne de type FLASH pour stocker des présentations (~128MB minimum)
- d'être rechargé et d'être capable de faire défiler des présentations durant au moins 2 heures
- d'afficher son état via une ou plusieurs diodes électroluminescentes.

5.2. Spécification électrique / électronique

L'iPresenter2 devra :

- embarquer une carte processeur Intel XScale de la société Toradex
- être compatible avec les modèles PXA270 et PXA320
- générer les tensions d'alimentation nécessaires au système depuis la batterie
- gérer la charge de la batterie de manière adéquate et protégée
- protéger tous les connecteurs contre les perturbations extérieures (électricité statique, inversion de polarité, surtensions, …)

5.3. Spécification mécanique

L'iPresenter2 devra :

- adopter des dimensions minimales, idéalement pour pouvoir être transporté dans la poche d'une chemise (son épaisseur ne doit pas dépasser 20mm)
- adopter un poids minimal
- être pourvu d'un connecteur pour la liaison au projecteur (le format est à définir)
- être pourvu de deux connecteurs USB sur des faces différentes
- être pourvu d'un connecteur permettant la recharge de la batterie interne

5.4. Spécification logicielle

L'iPresenter2 devra :

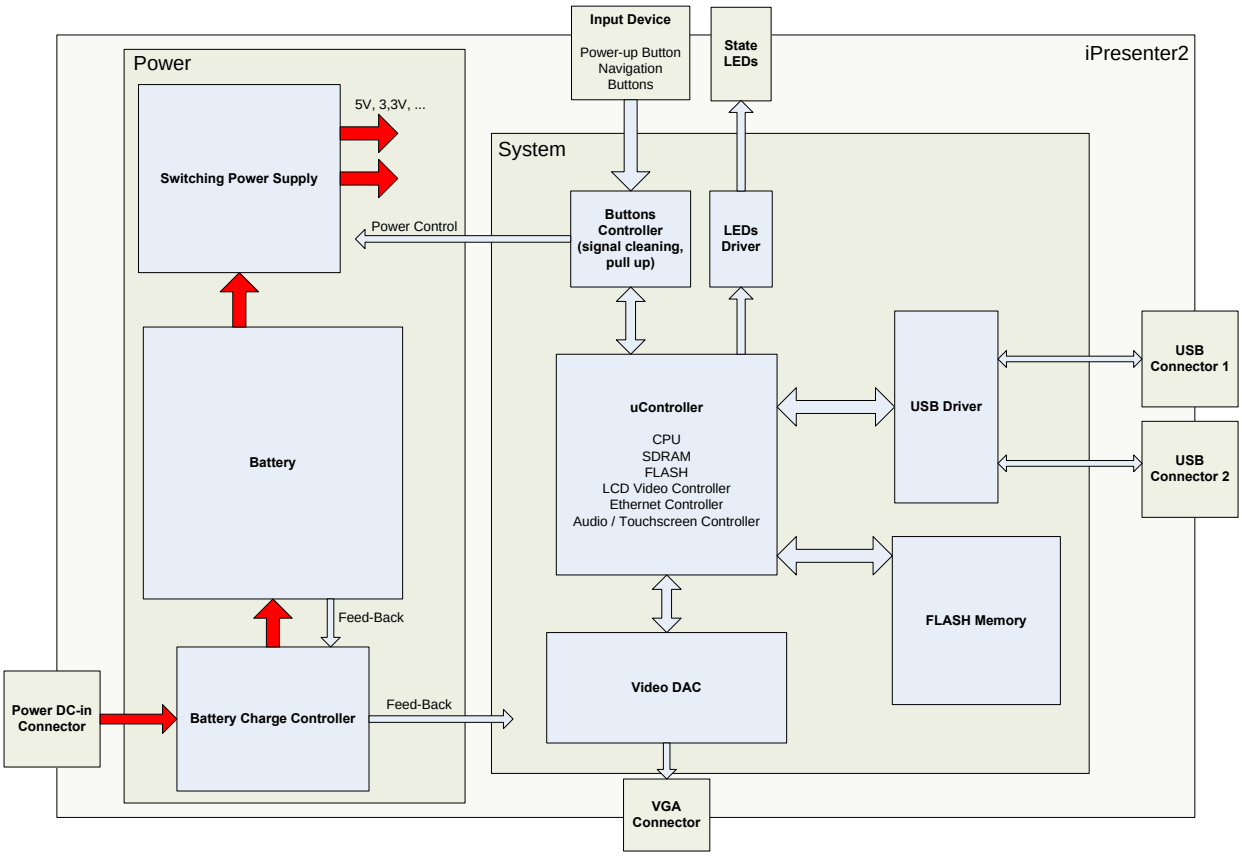
- intégrer un lecteur de fichiers Powerpoint natif
- présenter une interface utilisateur conviviale, simple et pratique à utiliser contenant :
    - un menu pilotable avec les boutons
    - la possibilité de naviguer dans la mémoire interne et sur les périphériques de stockage USB branchés
    - un pictogramme ou un texte indiquant l'état de la batterie
    - la possibilité de changer la résolution de l'écran si le matériel le permet
    - un menu timing pour permettre l'avance automatique des diapositives
    - une interface permettant le pilotage de l'avance des diapositives via les boutons

6. Schéma logique

La première étape menant à la réalisation physique d'une carte électronique est la création d'un schéma logique. Chaque bloc de celui-ci représente une entité ou une fonctionnalité que devra assurer le système. La représentation de l'iPresenter2 se divise en deux sous-systèmes. La partie alimentation (Power) et la partie intelligente (system).

Les blocs situés entre la carte et l'extérieur représentent les interfaces. On y retrouve les différents connecteurs (2x USB, écran et le connecteur d'alimentation externe) ainsi que les boutons et les LEDs.

Les flèches en rouge représentent le flux des courants d'alimentation, celles en bleu les signaux logiques entre les blocs.



La batterie est située au centre du bloc d'alimentation. Elle peut être rechargée en amenant du courant via le connecteur d'alimentation externe. Le bloc contrôleur de charge batterie permet de réguler la charge. Le signal feed-back permet au chargeur de connaître certaines informations importantes de la batterie telles que sa tension ou encore sa température. Le courant de la batterie passe au travers des alimentations à découpage qui ont pour but de fournir les bonnes tensions au reste du système.

L'élément principal de la partie intelligente est bien évidemment le microcontrôleur. Il est entouré d'interfaces permettant la communication avec les connecteurs externes. On y retrouve aussi une interface permettant de connaître l'état des boutons et de définir celui des LEDs.

Il existe deux signaux passant entre les sous-systèmes. Le premier permet soit par l'intermédiaire des boutons soit par un signal provenant du processeur de mettre sous/hors tension les alimentations de la carte. Le deuxième permet au microcontrôleur de s'informer de l'état de la batterie. Cette information sera sans doute récoltée par le chargeur de batterie.

7. Réalisation de la schématique

La deuxième étape est la création de la schématique et le choix des composants, ceci en tenant compte de tous les éléments de la spécification. Chaque bloc découvert lors de la découpe logique doit être pris séparément et transformé en circuit électronique réalisable.

Afin de simplifier et d'accélérer le design, il faudra toujours étudier la possibilité de réutiliser les éléments présents sur la carte d'évaluation de Toradex, dont le fonctionnement est prouvé.

Remarque : la totalité du circuit sera réalisée avec des composants du type SMD, et de hauteur minimale, afin de respecter la spécification. Seuls les connecteurs seront des composants du type traversant, ceci pour des raisons de solidité et de durabilité des soudures. La schématique, quand à elle, sera dessinée sur le logiciel PCAD 2006 d'Altium.

7.1. L'alimentation

Fournir la bonne tension à chaque composant et le courant nécessaire à leur fonctionnement n'est pas une tâche simple dans les systèmes embarqués fonctionnant sur batterie. Les contraintes sont nombreuses, il est donc nécessaire d'étudier précisément les besoins et les moyens à mettre en œuvre pour assurer le bon fonctionnement de l'ensemble.

La carte de démonstration de Toradex est pourvue d'alimentations à découpage surdimensionnées et faites pour être alimentées par des tensions élevées (de l'ordre de 10-14Vdc). Elles ne sont donc pas adaptées à notre application et doivent être totalement revues pour l'iPresenter2.

7.1.1. Calcul de la consommation du système

Avant de dimensionner les éléments de la batterie ou de l'étage d'alimentation, il faut calculer la consommation globale ainsi que les courants et tensions nécessaires au reste du système. L'établissement d'une spécification sur ces paramètres ne peut être effectué qu'en récupérant les données constructeur des composants connus et d'estimer celles des autres :

Bloc logique	Elément	Tension d'alimentation	Courant moyen consommé	Courant de crête consommé
CPU	Colibri PXA270 @ 520MHz Colibri PXA320 @ 806MHz	3.3V 3.3V	~ 300 mA ~ 250 mA	1100 mA 1200 mA
Vidéo	Convertisseur D/A	3.3V	~ 78 mA	87 mA
FLASH	Chip Flash NAND	3.3V	~ 1 mA	30 mA
USB	Clé USB 2.0 standard Autres périphériques (wifi)	5.0V 5.0V	~ 1 mA ~ 140 mA	40 mA 2x 500mA (norme)
DIVERS	Pull-up/down, bascules,…	5.0V	~ 10 * 0.1mA	~ 10 * 0.1mA
TOTAL	**iPresenter2**	**3.3V @ 560 MHz** **3.3V @ 806MHz** **5V**	**~ 380 mA** **~ 329 mA** **~ 141 mA**	**1218 mA** **1318 mA** **1000 mA**

On constate donc que deux tensions d'alimentation sont nécessaires :

- 3.3 V permettant d'alimenter le processeur, le convertisseur D/A et la mémoire Flash
- 5.0V pour le port USB qui permettra d'alimenter le stick mémoire et un périphérique Wifi

Le courant moyen sur la ligne d'alimentation 3.3V sera compris entre 328mA et 379mA. Celui sur le 5V sera d'environ 140mA (avec un stick USB et une clé USB Wifi consommant 700mW). Lorsque des pointes de courant auront lieu, l'alimentation devra être capable de fournir les courants suivants : jusqu'à 1317mA sur le 3.3V et 1000mA sur le 5V.

La carte doit être capable de fonctionner avec les deux processeurs Intel XScale PXA270 et 320 ; la consommation moyenne sera donc calculée avec les courants les plus élevés.

Voici un tableau récapitulatif des spécifications de l'étage d'alimentation :

Tension nécessaire	Courant moyen	Courant de crête
3.3V	**380 mA**	**1320 mA**
5.0V	**150mA**	**1000mA**

7.1.2. Calcul des tolérances de tension

Maintenant que les tensions nécessaires au système sont connues, il reste à calculer un élément important avant de dimensionner l'alimentation : les tolérances admises (fluctuations de la tension d'entrée) par les différents consommateurs de la carte. Le tableau ci-dessous recense les tolérances des éléments principaux du système :

Bloc logique	Elément	Tension nom. [V]	Tolérance [V]	Tolérance [%]
CPU	Colibri PXA270 Colibri PXA320	3.3V 3.3V	2.97 … 3.63 V 3.1 … 3.4 V	+/- 10% - 6% … + 3%
Vidéo	Convertisseur D/A	3.3V	3.0V … 3.6V	+/- 9%
FLASH	Chip Flash NAND	3.3V	2.7V … 3.6V	- 18% … + 9%
USB (norme)	Autres périphériques	5.0V	4.75V … 5.25 V	+/- 5%
TOTAL	**iPresenter2**	**3.3V (PXA 270)** **3.3V (PXA320)** **5V**	**3.0V … 3.6V** **3.1V … 3.4V** **4.75V … 5.25V**	**+/- 9%** **- 6% … + 3%** **+/- 5%**

On constate que les équipements les plus exigeants sont le processeur et le convertisseur D/A. L'établissement des tolérances de l'iPresenter2 complet est réalisé en prenant les tolérances des composants les plus critiques. Dans ce cas-ci, c'est le processeur PXA320 qui est le plus exigeant. Voici le tableau de spécification final obtenu :

Tension nécessaire	Tolérance [V]	Tolérance [%]
3.3V	**3.1 … 3.4 V**	**- 6% … + 3%**
5.0V	**4.75 … 5.25 V**	**+/- 5%**

7.1.3. Les différents types d'accumulateurs

La carte développée devra être capable de tirer son énergie d'un ou plusieurs accumulateurs. Ces derniers devront évidemment être rechargeables. Actuellement, il existe 6 principaux types d'accumulateurs, de technologies différentes. Certains sont montés en batterie pour atteindre des tensions utilisables, d'autres ne nécessitent qu'un seul élément.

Le tableau ci-dessous recense les caractéristiques principales de chacune des technologies. Cela va nous permettre de choisir plus facilement la plus adaptée pour notre carte.

Caractéristiques	Plomb Acide	Nickel Cadmium	Nickel Métal Hydride	Lithium Ion	Lithium Polymère	Alcalines (rechargeables)
Energie massique [Wh/kg]	30-50	45-80	60-120	110-160	100-130	80
Energie volumique [Wh/ltr]	75 – 120	90 – 150	175 – 330	220 – 330		-
Cycles de vie	200 à 300	1500	300 à 500	500 à 1000	200 à 300	10 à 50
Auto décharge par mois	5 %	20 %	30 %	10 %	10 %	0,3 %
Tension nominale	2 V	1,2 V	1,2 V	3,6 ou 3,7 V	3,7 V	1,5 V
Capacité maximale	4000 Ah	1500 Ah	18 Ah	4,5Ah	1,6 Ah	-
Résistance interne [mΩ]	0,3 à 100	100 à 200	200 à 300	150 à 250	200 à 300	200 à 2000
Courant max de décharge	5 C	20 C	5 C	> 2 C	> 2 C	C/2
Courant nom. de décharge	C/5	1 C	C/2	< 1C	< 1C	C/5
Sensible à l'effet mémoire	Non	Oui	Oui	Non	Non	Non
Coût moyen	25	50	60	100	100	5

Source : Wikipedia et www.ni-cd.net

Afin d'effectuer un choix judicieux, il faut dresser en parallèle à ce tableau, une liste des contraintes et des exigences pour notre application.

De toute évidence, les critères déterminants sont l'encombrement ainsi que le poids de l'accumulateur. Il devra être de taille minimale, léger, tout en étant capable de stocker une quantité d'énergie importante. Ceci élimine déjà les éléments au plomb, Ni-Cd et Ni-MH, dont l'énergie volumique ainsi que massique est peu élevée. Le faible nombre de recharges possibles ainsi que le peu de courant de décharge utilisable des accumulateurs de type alcaline en font une alternative inintéressante. Un des autres critères important est le prix des éléments. Aussi on peut constater que les éléments à base de lithium sont parmi les plus onéreux. (Coût moyen)

Malgré cela, ils restent les plus intéressants sur tout autre point de ce rapide comparatif. La technologique Lithium-Ion, plus mature, assure un plus grand nombre de cycles de vie ainsi qu'une résistance interne plus faible. Elle constitue donc le meilleur candidat pour l'iPresenter2.

A remarquer que la majorité des assistants personnels PDA actuellement sur le marché sont équipés d'accumulateurs Lithium-Ion. Leur tension nominale de 3.6 / 3.7V (3.0V au minimum à 4.1V / 4.2V complètement chargé) permet de n'utiliser qu'un seul élément pour réaliser la batterie, d'où un nouveau gain de place. Cela permet aussi évidemment de simplifier la réalisation du chargeur, mais compliquera quelque peu l'étage d'alimentation de la carte, comme on le verra par la suite.

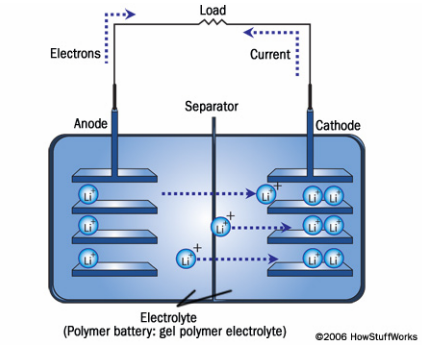
7.1.4. Les éléments Lithium-Ion

Une étude un peu plus approfondie des éléments lithium-ion va permettre de faciliter la réalisation et le choix des composants pour l'étage d'alimentation ainsi que le chargeur de batterie.

L'accumulateur lithium-ion occupe aujourd'hui une place prédominante sur le marché de l'électronique portable. Ses principaux avantages sont une densité d'énergie spécifique et volumique élevée (4 à 5 fois plus que le Ni-MH par exemple) ainsi que l'absence d'effet mémoire (aucun ou presque). Enfin, l'autodécharge est relativement faible par rapport à d'autres accumulateurs. Cependant le coût reste important et cantonne le lithium aux systèmes de petite taille.

La tension nominale des accus au lithium est de 3,6V pour les Li-ion les plus anciens ou actuels de moindre qualité et de 3,7V pour les Li-ion récents.

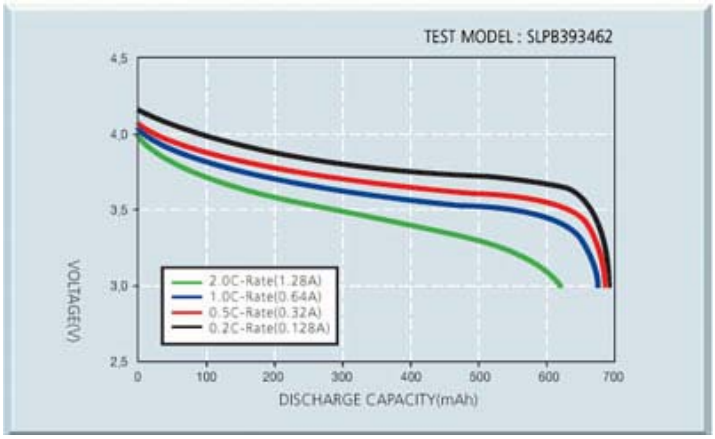
Ils ne supportent pas la surcharge aussi faible soit-elle et il faut impérativement respecter le niveau de la tension de charge. (maximum 4.1 ou 4.2 Volts selon les modèles)
La tolérance aujourd'hui admise est de +/- 0.05V par élément. Le courant de charge doit être limité entre C/2 et 1C. Dans ces conditions respectées, la charge dure entre 2 et 3 heures.



Comme pour les autres technologies de batterie, les accus au Lithium ont une tension minimale de décharge en dessous de laquelle il ne faut surtout pas les décharger sous peine de destruction des éléments. La tension basse de destruction est de 2,5 Volts. Mais en utilisation normale, il ne faut pas descendre en dessous de 2.8 Volts (la zone entre 2.5V et 2.8V est très petite).

Les accus au Lithium sont aujourd'hui capables de débiter de forts courants. Leur résistance interne s'approche maintenant de celle des accus Ni-Cd ou Ni-MH, ce qui va nous permettre d'atteindre les 2.5 ampères environ nécessaires lors des appels de courants du système.

Voici la courbe de décharge d'un élément lithium-ion :



On constate que durant la majorité de son cycle de décharge, la batterie se maintient à un niveau de tension d'environ 3.6 Volts, lorsqu'on le décharge à environ C / 2.

Les problèmes de sécurité imposent d'intégrer un système électronique de protection, embarqué le plus souvent dans chaque élément au lithium. (Il empêche une charge ou décharge trop profonde : sinon le danger peut aller jusqu'à l'explosion de l'élément).

7.1.5. Choix de la batterie (1/2)

Le choix final s'est donc porté sur une batterie de type Lithium-Ion. Une information capitale est l'autonomique désirée pour notre appareil. Celle-ci devra être d'environ 2h. La tension de sortie nominale doit être de 3.6V ou 3.7V. Un modèle prismatique semble le plus adapté à la forme du boîtier. Un élément de taille comparable aux piles AA ou AAA est aussi imaginable (cylindrique).

La capacité de la batterie ne pourra être calculée qu'après la réalisation de l'alimentation. Un critère déterminant comme le rendement n'est pas encore bien défini.

7.1.6. Conception de l'étage d'alimentation

7.1.6.1. Les différents types d'alimentation

Maintenant que les données importantes sont calculées, on peut établir un tableau final contenant toutes les spécifications de l'alimentation à réaliser.

Tension d'entrée	Tension de sortie	Tolérance de sortie	Courant nominal de sortie	Courant maximal de sortie
2.8 à 4.2V	3.3V	3.1 à 3.4V	380mA	1320mA
2.8V à 4.2V	5.0V	4.75 à 5.25 V	150mA	1000mA

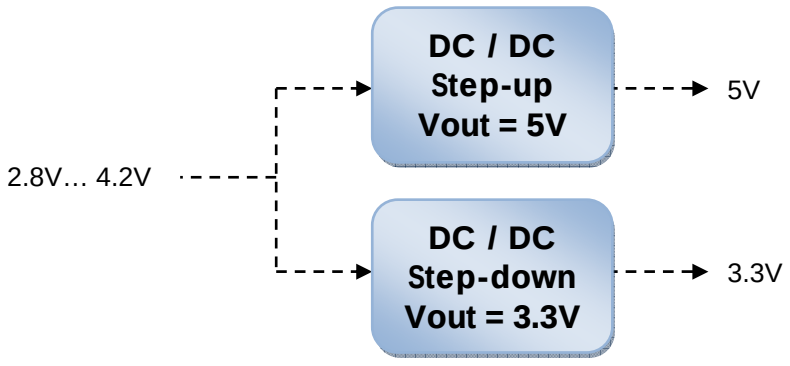
Il y a deux points importants à relever de ce tableau :
- Pour le 5V, la tension de sortie est strictement plus élevée que toute la plage d'entrée
- Pour le 3.3V, la tension de sortie est située dans la plage d'entrée.

Pour les deux tensions de sortie, il est nécessaire d'utiliser des alimentations à découpage, et ce pour plusieurs raisons évidentes :
- il faut créer une tension de sortie plus haute que celle d'entrée
- le rendement est un facteur déterminant dans la réalisation d'alimentation d'appareils portables
- Au vu des courants mis en jeu (> 1A), les pertes générées par des régulateurs linéaires créeraient un échauffement non négligeable et indésirable

Différentes topologies d'alimentation à découpage sont imaginables et existent sur le marché. Les suivantes ont été analysées :

1. Un convertisseur élévateur pour le 5V, et un abaisseur pour le 3.3V en parallèle

Cette solution est l'une des plus simples. Deux convertisseurs sont mis en parallèle en sortie de la batterie. L'un élève la tension d'entrée jusqu'à obtenir 5V. Le second l'abaisse jusqu'à 3.3V.

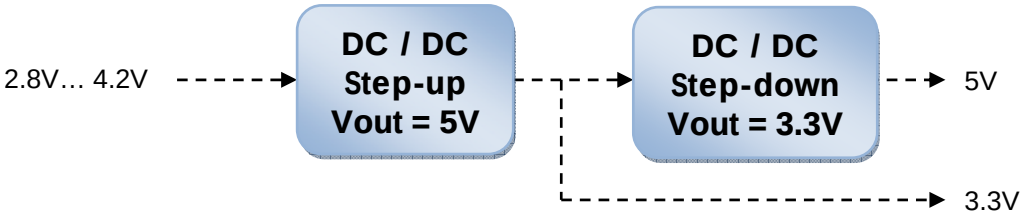


Son défaut principal se situe aussi au niveau du rendement, un abaisseur uniquement ne sera pas capable d'utiliser toute la plage de tension de l'accumulateur lithium-ion. En effet, un abaisseur a besoin d'une tension d'entrée égale voir, en pratique, supérieure à sa tension de sortie. Cela n'est pas acceptable pour notre application.

Avantages	**Inconvénients**
✓ Simplicité de mise en œuvre des convertisseurs ✓ Bruit en sortie des convertisseurs isolés ✓ Bon rendement de la ligne 5V	× Rendement médiocre sur la ligne 3.3V (utilisation partielle de la plage d'entrée)

2. Un convertisseur élévateur pour le 5V, suivi en série d'un convertisseur abaisseur pour le 3.3V

Il s'agit ici de placer les mêmes convertisseurs, mais cette fois en série, le 3.3V étant généré à partir du 5V. Cela permet de passer outre le défaut principal de la solution précédente, vu que la tension d'entrée du 2ème convertisseur sera toujours de 5V.



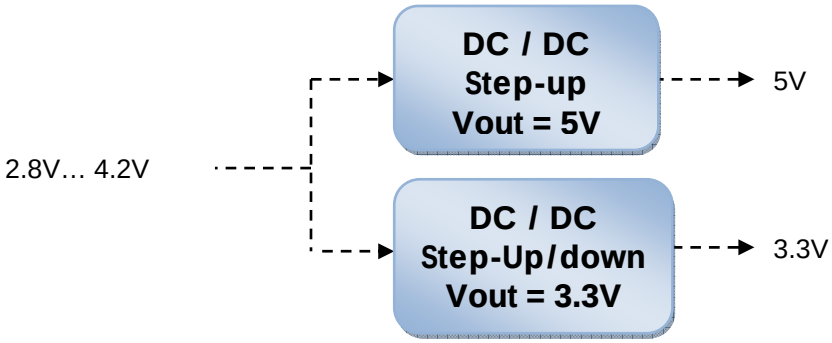
Cette solution est aussi facilement réalisable que la première. Il s'agit de convertisseurs élévateurs et abaisseurs uniquement.

Mais un gros problème persiste au niveau du rendement et du bruit généré par une telle structure.
Pour générer le 3.3V, on doit passer à travers les deux convertisseurs, et donc les pertes seront bien plus grandes. De plus, le bruit généré par le convertisseur 5V se répercutera sur la ligne 3.3V. Cette solution n'est donc, elle non plus, pas idéale pour notre application.

Avantages	**Inconvénients**
✓ Simplicité de mise en œuvre des convertisseurs ✓ Bon rendement de la ligne 5V ✓ Utilisation de toute la plage de tension de la batterie	× Rendement médiocre sur la ligne 3.3V (pertes additionnées des deux convertisseurs) × Addition du bruit des deux convertisseurs sur la ligne 3.3V

3. Un convertisseur élévateur pour le 5V, et un convertisseur élévateur-abaisseur type SEPIC pour le 3.3V

Cette solution utilise toujours un convertisseur élévateur uniquement pour la ligne 5V. Par contre la ligne 3.3V est générée à partir d'un convertisseur de type SEPIC (élévateur-abaisseur)

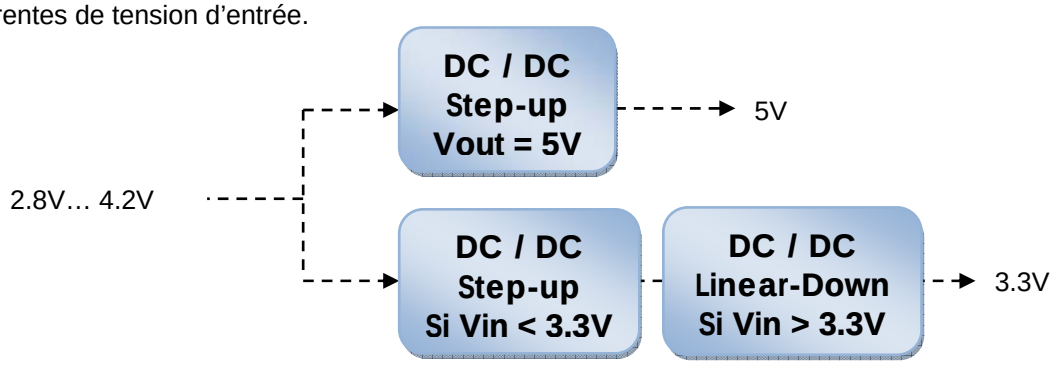


Cette solution paraît idéale. En effet, non seulement elle génère un bruit minimal sur les deux lignes (car elles sont indépendantes) mais le rendement total est très bon car la batterie est utilisée dans toute sa gamme de tension (2.8V… 4.2V).

Avantages	**Inconvénients**
✓ Bon rendement sur les deux lignes ✓ Bruit limité sur les deux lignes	× Mise en œuvre plus compliquée × Nécessite deux convertisseurs de types différents

4. Un convertisseur élévateur pour le 5V et un convertisseur élévateur couplé à un LDO à transistor (régulateur linéaire) pour le 3.3V

Cette solution constitue une alternative un peu particulière à celle du convertisseur SEPIC. Elle couple un convertisseur élévateur et un régulateur linéaire à transistor, chacun travaillant pour des gammes différentes de tension d'entrée.



Le bruit généré est nettement amélioré car lorsque la tension d'entrée dépasse 3.3V, le découpage est stoppé et le régulateur linéaire prend le relais. La différence de rendement entre cette solution et la précédente doit être étudiée.

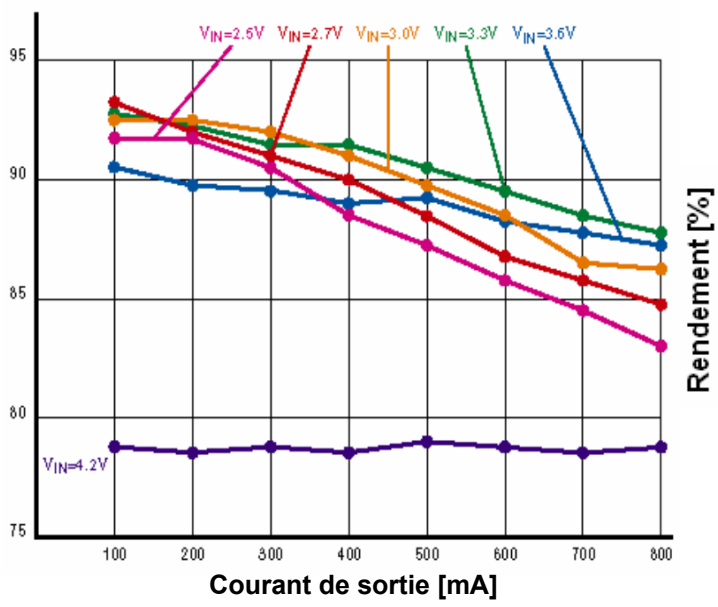
Avantages	**Inconvénients**
✓ Bon rendement sur les deux lignes ✓ Bruit limité sur les deux lignes ✓ Bruit minimal sur la ligne 3.3V	× Mise en œuvre plus compliquée × Rendement ? (régulateur linéaire)

7.1.6.2. Choix de l'alimentation

On peut déjà éliminer les solutions 1 et 2, celles-ci n'étant simplement pas utilisables. En effet, la première offre un rendement au niveau du 3.3V inférieur à 80% sur toute la gamme d'entrée de la tension. De plus, le bruit généré est doublé sur cette ligne.
La 2ème n'est pas utilisable du tout, du fait qu'un abaisseur a besoin d'une tension d'entrée supérieure à la tension de sortie. Dès que la tension d'entrée de celui-ci passera en dessous d'environ 3.6V (pertes sur la diode roue-libre à prendre en compte), la tension de sortie va chuter. (Surtout lors des forts appels de courant, ce qui provoquera soit le plantage, soit l'extinction de l'appareil.)

Reste les deux dernières solutions, qui demandent à être étudiées plus en détail.

L'avantage principal qui semble en faveur de l'alimentation SEPIC est son rendement. Plusieurs articles traitent de ce sujet précis. Il en ressort un élément principal : durant la plus grande partie de sa phase de décharge, la batterie verra sa tension de sortie stabilisée autour des 3.6V. Or, pour cette tension-là ainsi que pour toutes les tensions inférieures, la solution avec le régulateur linéaire obtient un meilleur rendement, comme le montre la figure si contre (> 89% @ Iout = 300mA). Le SEPIC lui plafonne à 80%.
Il semble donc logique de privilégier et de choisir pour notre application la solution 4 :

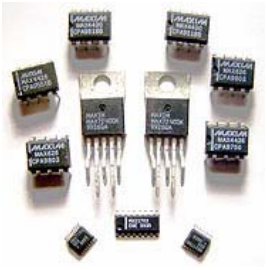


Un convertisseur élévateur pour le 5V et un convertisseur élévateur couplé à un LDO à transistor (régulateur linéaire) pour le 3.3V

7.1.6.3. Conception et calculs des composants

La structure, les tensions, les courants et les tolérances de l'alimentation étant totalement définis, il est maintenant possible de réaliser l'alimentation. Il existe plusieurs manières de réaliser des alimentations à découpage. Le plus simple et le plus fiable reste d'utiliser des circuits intégrés réalisant déjà les fonctions de découpage (p. ex. PWM) et de régulation, et d'y adjoindre quelques composants extérieurs.

Plusieurs fabricants proposent des circuits élévateurs / abaisseurs. Mais l'un des seuls étant capable de fournir un courant continu de sortie de 1 A à 1.5A par ligne, avec possibilité d'implémenter un régulateur linéaire, le tout avec un nombre minimal de composants est le MAX1703 du constructeur MAXIM.

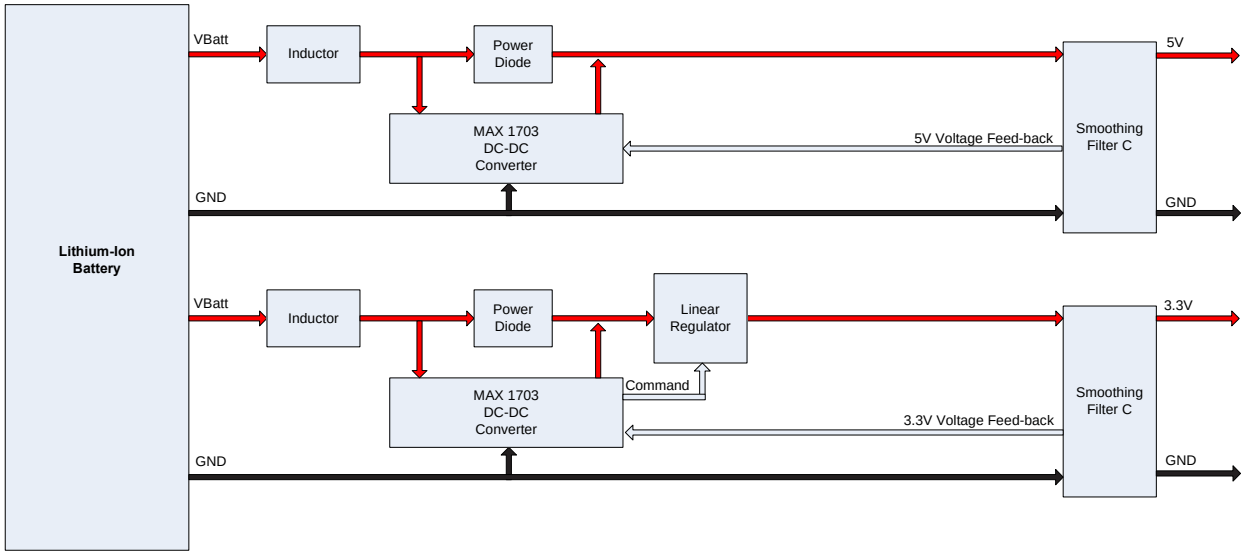


Chaque ligne de sortie sera donc composée d'un MAX1703 et de quelques éléments extérieurs, dont une bobine avec ferrite, une diode de puissance, et des condensateurs permettant de filtrer le signal. Les résistances utilisées dans les ponts diviseurs doivent avoir une précision de 1% (série E96)

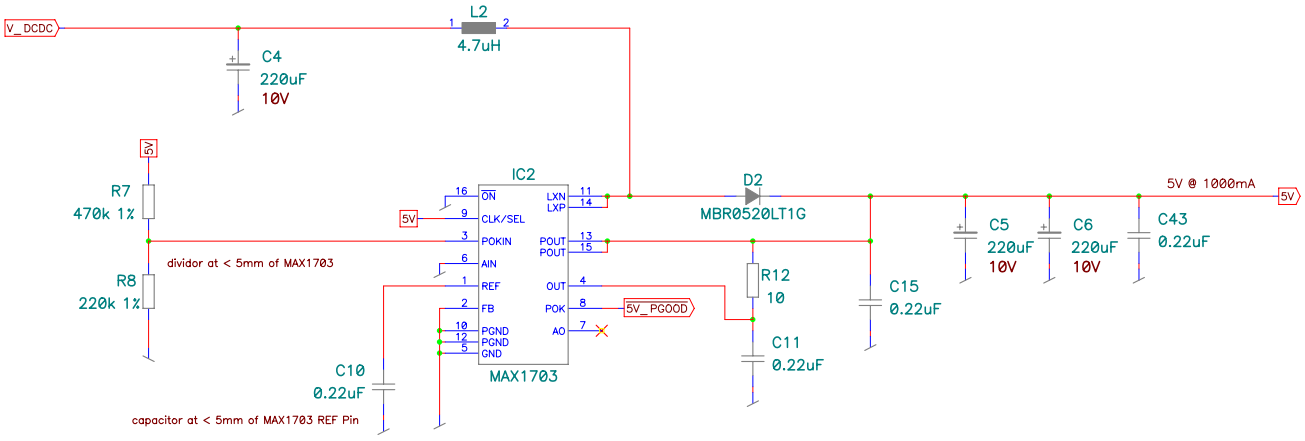
Une attention particulière doit être portée aux condensateurs de filtrage. Ceux-ci (en particulier ceux de sortie) doivent impérativement comporter une résistance série équivalente très basse (leur impédance de source interne doit être faible dans un montage tel que celui d'une alimentation à découpage) et de préférence être à base de tantale. (low ESR tantalum capacitor)

Remarque : Linear technology a mis en production fin mars 2007 un nouveau composant (LTC3533) comparable au MAX1703 mais de taille plus petite et demandant moins d'éléments extérieurs. Le prix est aussi inférieur (Des échantillons gratuits ont été commandés pour une future version de l'iPresenter2)

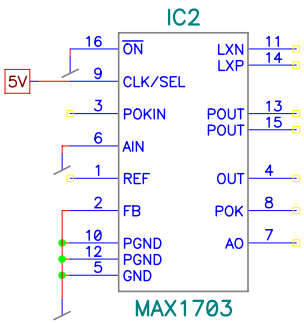
Le schéma bloc suivant, comprenant les composants physiques les plus importants ainsi que leurs interactions, facilite la réalisation de la schématique. Il a été réalisé à l'aide des datasheets du MAX1703.



L'alimentation 5V

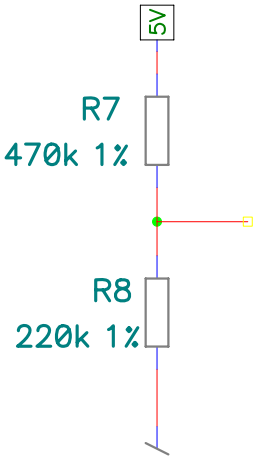


1-Cell High Power Low Noise DC-DC Step-up MAX1703



Le circuit le plus simple à calculer est celui de la ligne 5V. En effet, pour que le régulateur fournisse 5V à sa sortie, il suffit de relier la pin FB (feedback) à la masse. La pin CLK/SEL permet de mettre en route le convertisseur en mode PWM. Il faut la relier à la sortie. La pin ON permet d'activer le convertisseur. Elle sera mise à zéro, pour que, dès qu'une tension apparaisse à l'entrée, le découpage démarre. En effet, même en veille, ce circuit consomme du courant ; une autre manière sera utilisée pour enclencher/déclencher l'alimentation en amont.

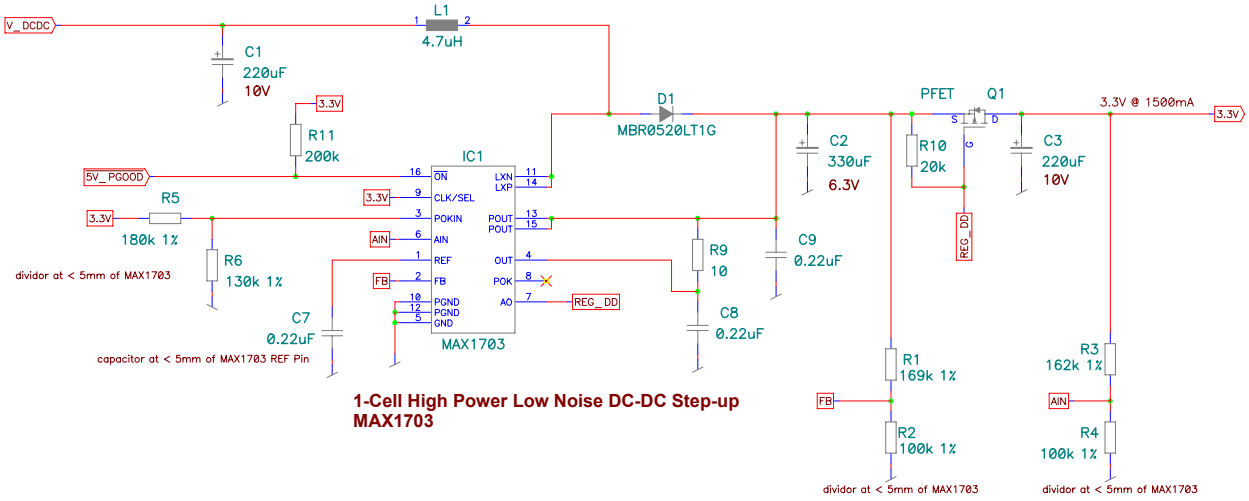
Afin de ne pas faire démarrer les deux convertisseurs simultanément, celui de la ligne 3.3V ne s'enclenche qu'au moment où le 5V est stabilisé. Il est possible de faire cela en couplant la pin ON du deuxième convertisseur à la sortie POK du premier, car le MAX1703 est équipé d'un comparateur interne prévu à cet effet. (Ne pas oublier la résistance pull-up, car la sortie POK est à collecteur ouvert). Il est possible de définir à partir de quelle tension de sortie le convertisseur 5V permet à celui 3.3V de démarrer. Ceci se fait grâce au pont diviseur sur l'entrée POKIN, composé de R7 et de R8.



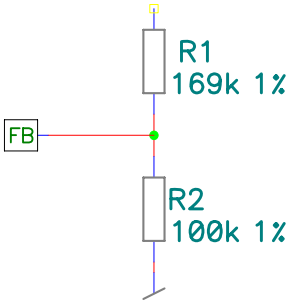
`R7 = R8 * (Vtrig5 / 1.25V – 1)`
Si on choisit Vtrig5 = 4V :

- **R7 = 470k**
- **R8 = 220k**

L'alimentation 3.3V



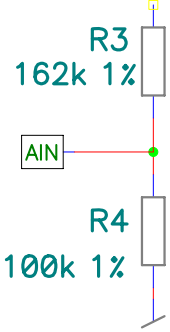
Le convertisseur 3.3V demande un peu plus de calculs. Le premier élément à fixer est la tension de sortie. On choisira une tension de sortie légèrement plus élevée, pour pouvoir la filtrer ensuite grâce au régulateur linéaire. Un pont diviseur composé de R1 et de R2 doit être placé sur l'entrée FB :



`R1 = R2 * (Vout / 1.25V – 1) ;`
Si l'on choisit Vout = 3.36V

- **R1 = 169k**
- **R2 = 100k**

Ensuite, un pont diviseur est nécessaire pour commander le régulateur linéaire. Le MAX1703 est pourvu d'un amplificateur de différence entre une tension de référence interne et l'entrée AIN. Dès que la tension sur AIN dépassera 1.25V, la sortie AO qui pilote le transistor FET limitera et filtrera la tension de sortie. Les valeurs des résistances du pont fermant la régulation sont :



`Vout = Vin * (1.25 + R7 / R8)`
Si l'on choisit Vout = 3.275V (~ au milieu de la plage de tolérance de la ligne 3.3V)

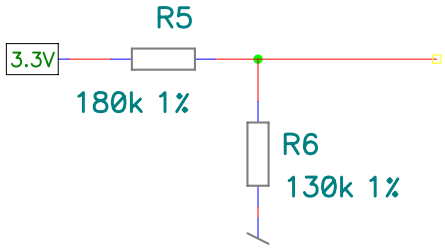
- **R3 = 162k**
- **R4 = 100k**

De même que pour le 5V, un signal indiquant que les deux alimentations sont stabilisées (POK) est nécessaire pour la suite du circuit. Un autre pont diviseur sur l'entrée POKIN doit être calculé :

```
R5 = R6 * (Vtrig33 / 1.25V – 1)
```

Si l'on choisit Vtrig33 = 3V

- R5 = 180k
- R6 = 130k

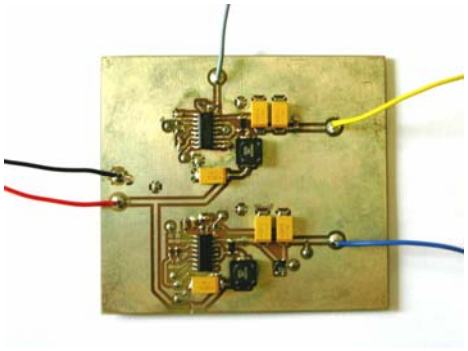


Les autres points importants, comme les distances entre les composants critiques, sont décrit dans la schématique elle-même, pour les deux convertisseurs.

7.1.6.4. Précautions lors du routage (carte de test)

La réalisation d'alimentations à découpage demande un design très rigoureux au niveau du PCB. Certaines contraintes comme la réduction maximale de la taille des boucles de courant ou encore la présence de condensateurs au tantale avec une faible résistance série (low ESR) sont nécessaires.

Afin d'assurer la qualité de la carte finale et de valider les calculs effectués, la décision a été prise de réaliser un prototype de l'alimentation. Celui-ci a été imprimé à la HEVs et les composants ont été soudés à la main.



L'ajout des condensateurs C43 et C44 ont permis de réduire le bruit à la sortie. Une optimisation de routage (réduction de la taille des pistes de signaux, rapprochement des composants, réduction de la taille des vias, diminution supplémentaire de la taille des boucles de courant, …) avant la réalisation de la carte finale est encore nécessaire.

La schématique de cette carte est disponible en annexe 01. Une version informatique au format PCAD de la schématique et du routage PCB est disponible sur le CD-ROM en annexe. (*1_Schematics*\ *PowerSupply_TestBoard.sch*).

La procédure de test complète de l'alimentation est disponible en annexe 02. Une version informatique se trouve sur le CD-ROM en annexe. (4_Documentation\ AN02_AlimTest.doc).

Les résultats du test sont réjouissants. L'alimentation est stable et le bruit reste dans les tolérances. L'élément qui manquait pour pouvoir calculer la capacité de l'accumulateur était le rendement des alimentations. Il est maintenant défini :

Rendement moyen de la ligne 3.3V	**80%**
Rendement moyen de la ligne 5V	**88%**

7.1.6.5. Choix de la batterie (2/2)

Maintenant que l'ensemble du système d'alimentation est connu, il est possible de calculer la capacité de la batterie nécessaire, grâce au tableau suivant :

Tension de sortie	Courant moyen de sortie	Rendement de la ligne	Courant à l'entrée (Vin_{moy} = 3.6V)	Capacité pour 2 heures
3.3V	380mA	~80 %	435mA	870mAh
5.0V	150mA	~88%	237mA	474mAh
TOTAL				**~1350mAh**

Un seul élément Lithium-Ion 3.7V d'une capacité d'environ 1500-2200mAh semble être un choix idéal, marge de sécurité et vieillissement de la batterie comprise.

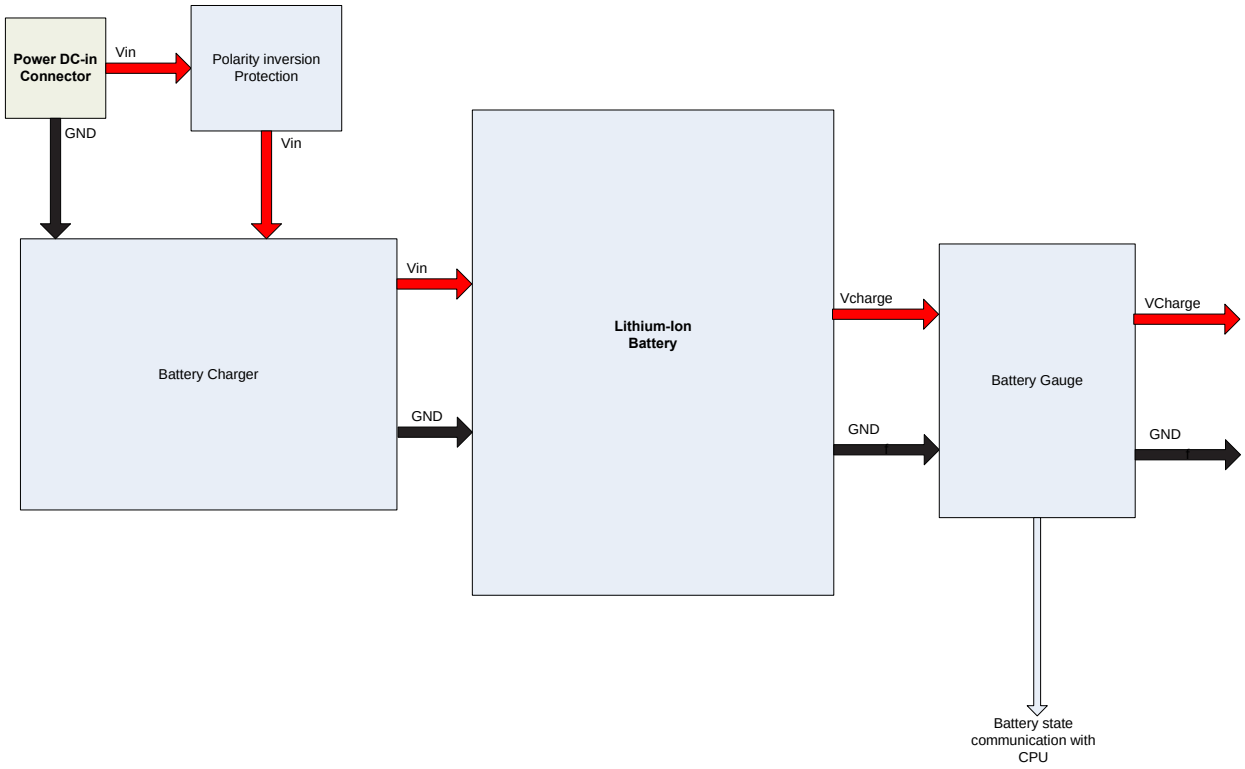
Le constructeur Varta propose un élément de 3.7V, d'une capacité de 2200mAh, cylindrique (19mm de diamètre sur 65mm de long). De plus, l'élément incorpore une protection contre les sur-courants de charge et de décharge, surtensions de charge et contre les décharges profondes. Il convient donc bien pour notre application.

Données de la batterie choisie

Capacité	**2100mAh** min, 2200mAh typique
Tension nominale	**3.7V**
Poids	**50 grammes**
Résistance interne	**150 mOhms**
Tension minimale d'opération	2.75V (**3V** recommandé)
Tension maximale d'opération	**4.2V**
Protection de surcharge	à 4.30…**4.35 V** (en 0.4…2 secondes)
Protection de décharge	à **2.42 V**…2.58V (en 0.2 secondes)
Courant de décharge maximal	**2.8 A**
Courant de charge maximal	1.6 A (**1.1 A** recommandé)
Coupure de protection de sur-courant	à **3 A**… 5 A (en 20ms max)
Cycle de vie	70% de la capacité après 300 cycles (0.5C/0.5C)

7.2. Le chargeur d'accumulateur

Une des fonctionnalités importantes est la charge de la l'accumulateur. Les éléments lithium Ion doivent être rechargés avec beaucoup de précaution et de manière extrêmement précise. On peut, par exemple, citer la tension de charge qui doit être de 4.2 volts **+/- 0.05V** par élément. Dans de bonnes conditions, la charge dure entre deux et trois heures. Pour réaliser cette charge, il vaut mieux avoir recourt à des circuits prévus à cet effet. De plus, on y ajoutera une protection contre les inversions de polarité et un circuit capable de mesurer l'état de l'accumulateur pour en informer le microcontrôleur. Voici le schéma bloc correspondant :

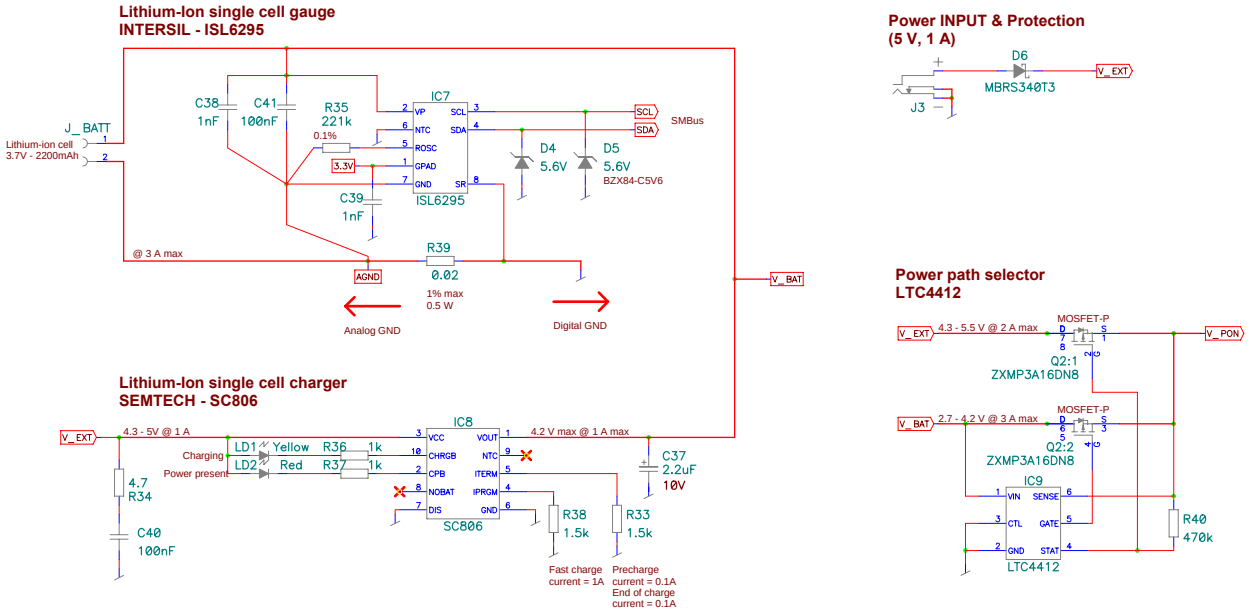


La HES-SO Valais a déjà développé un tel système de charge dont il sera possible de réutiliser certaines parties. Il est basé sur le circuit SC806 du constructeur Semtech. Il a l'avantage d'être de taille minimale (3mm * 3mm) et d'intégrer les transistors de puissance, la diode de blocage, la résistance de mesure de courant ainsi qu'un senseur de température. De plus, deux pilotes de LEDs indiquant la charge et la fin de charge sont mis à disposition.

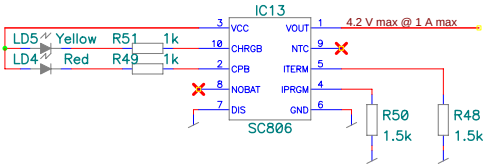
La gauge est basée sur le circuit ISL6295 d'Intersil. Il sera placé entre la batterie et le chargeur et pourra donc mesurer et comptabiliser les flux de courant dans les deux sens ainsi que sa tension. Ceci permettra d'estimer la charge de l'accumulateur. Une résistance de mesure de courant externe est nécessaire. La communication entre la gauge et le microcontrôleur s'effectue via un bus I^2C composé d'un signal d'horloge et des données.

Une subtilité mise en place dans la carte développée par la HES-SO permet, lorsqu'on branche le connecteur d'alimentation, de dériver une partie du courant de celui-ci directement sur les convertisseurs DC-DC, afin que rien ne soit tiré des batteries durant la charge. Le circuit LTC4412 de Linear permet d'implémenter cette fonctionnalité.

Voici la schématique du chargeur d'accumulateur :

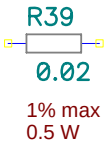


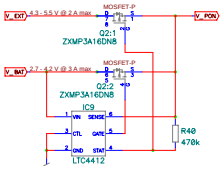
La protection contre les inversions de polarité est assurée par la diode Schottky D6 capable de supporter 3A et qui produit une faible chute de tension de 0.5V.



Le chargeur n'a presque besoin d'aucun élément externe. Les diodes électroluminescentes LD4 et LD5 avertissent que le chargeur fonctionne et que la charge est terminée. Les résistances R48 et R50 permettent de définir les courants maximaux de charge. En début et en fin de charge, un courant limité sera utilisé pour charger l'accumulateur. Celui-ci est réglable par l'intermédiaire de R48. Avec une valeur de 1.5kohms, le courant sera de 100mA. Lorsque la charge entre dans sa phase normale, le courant est limité par la résistance R50. Avec une valeur de 1.5kOhms, le courant de charge sera de 1 ampère.

Les deux lignes de communication de la gauge sont protégées contre les surtensions par les diodes Zener D4 et D5. La résistance R39 permet la mesure bidirectionnelle du courant de l'accumulateur. Sa puissance de 0.5W peut mesurer des courants allant jusqu'à 5 ampères maximum.





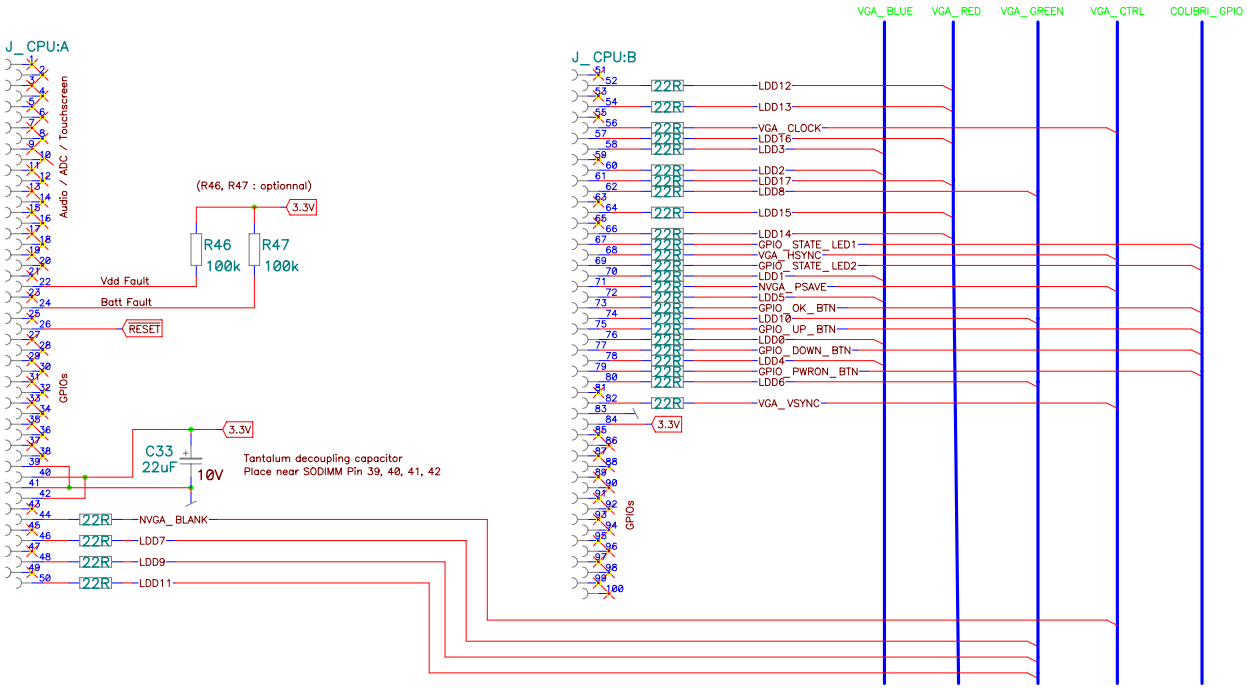
La dérivation du courant du chargeur sur les convertisseurs est donc réalisée grâce au LTC4412 ainsi que deux transistors MOS-FET de puissance. Ces deux transistors travaillent toujours de manière inversée.

7.3. Liaison à la carte Colibri

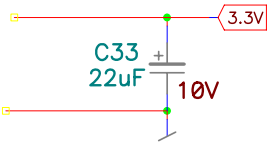
Cette partie de schématique décrit les connexions existantes entre la carte Colibri et le reste du système. On retrouve aussi tous les signaux mis à disposition par le processeur pour la sortie écran, les ports USB ainsi que les boutons et les LEDs.

Le module Colibri est compatible avec le format des barrettes de RAM de type DDR1 à 200 pins (emplacement Regular). La carte sera pourvue d'un connecteur acceptant de telles barrettes. Le datasheet du Colibri permet de connaître la fonctionnalité de toutes les pins.

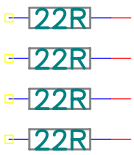
Voici la schématique de la première partie de la liaison :



La première étape est bien évidemment de fournir une alimentation à la carte Colibri. Elle nécessite une tension de 3.3V à 6 endroits du connecteur. De plus, quatre d'entre eux doivent être équipés de condensateur de découplage au tantale de 4.7 ou 22uF. Comme précisé dans la schématique, les condensateurs doivent être placés au plus près du connecteur.



Sur cette première partie de la schématique, on retrouve les signaux utilisés pour la sortie graphique ainsi que les boutons et les LEDs d'état. Chaque signal nommé LDD* correspond à un bit utilisé pour le codage RGB du signal graphique. Toutes les connexions nommées VGA_* sont des signaux de contrôle du bus graphique. Les connexions nommées GPIO_* sont utilisées soit en entrée pour connaître l'état des boutons, soit en sortie pour définir l'état des LEDs.

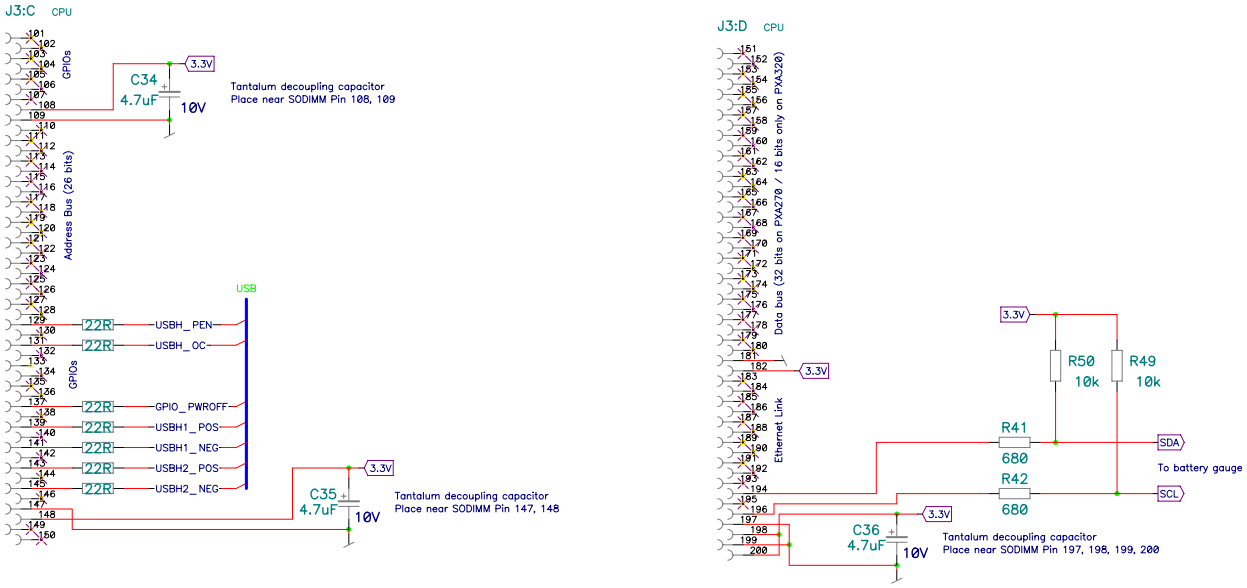


Chacune des entrées/sorties de la carte requiert une résistance en série sur la ligne de signal. Afin de simplifier les soudures et la clarté de la carte, on commandera des chips contenant chacun 4 résistances. De part ce fait là, il faut faire attention à effectuer les connexions de manière à ce que les pins placées au plus proches soient connectées à la même résistance. (n° de pin impair avec impair, n° de pin pair avec pair)

Un signal reset, qui est généré lors de la mise sous tension par un circuit spécialisé détaillé plus loin, permet une mise en route correcte de la carte lorsque l'alimentation démarre.

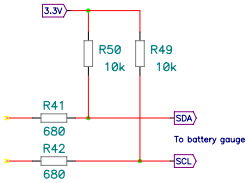
Sur les pins inutilisées, il y a des entrées/sorties à usage général ainsi que les connexions à la puce gérant l'audio, l'écran tactile ainsi que des entrées pour convertisseurs A/D. (il est possible de rajouter les résistances R46 et R47 pour forcer les signaux avertissant le processeur d'un défaut de batterie ou d'alimentation. Ces signaux ne sont pas utilisés par notre carte)

Voici la schématique de la deuxième partie de la liaison :



On voit ici les autres alimentations avec leur condensateur de découplage respectif.

Les signaux nommés USBH* sont utilisés pour la connexion aux ports USB. On y retrouve les deux lignes de données ainsi que les signaux de contrôle.



La liaison à la gauge qui contrôle l'état de l'accumulateur se fait par une liaison I^2C. Celle-ci est composée de deux signaux : un d'horloge et un autre ou passent les données. Deux résistances de 680 ohms chacune sont nécessaires en série avec les deux signaux ; une pull-up de 10kOhms par ligne est encore nécessaire.

Au niveau des connexions inutilisées, on retrouve à nouveau des entrées/sortie à usage général ainsi que des pins liées à la puce de gestion Ethernet. Il y a aussi le bus d'adresses et le bus de données. Ce dernier est soit de 32bits sur le PXA270, soit de 16 bits sur le PXA320). Dans le cas d'un bus 16 bits, les autres connexions sont converties en entrées / sorties à usage général.

Pour ce qui est de l'interchangeabilité entre le processeur PXA270 et PXA320 sur notre carte, il n'y a aucun élément à modifier. Toradex a mis à disposition sur son site une vidéo montrant que certains jumpers doivent être retirés lors du passage au PXA320. Or ces jumpers désactivent des signaux de Chip Select que notre carte n'utilise pas.

Video Tutorials

Colibri Upgrade PXA270 to PXA320

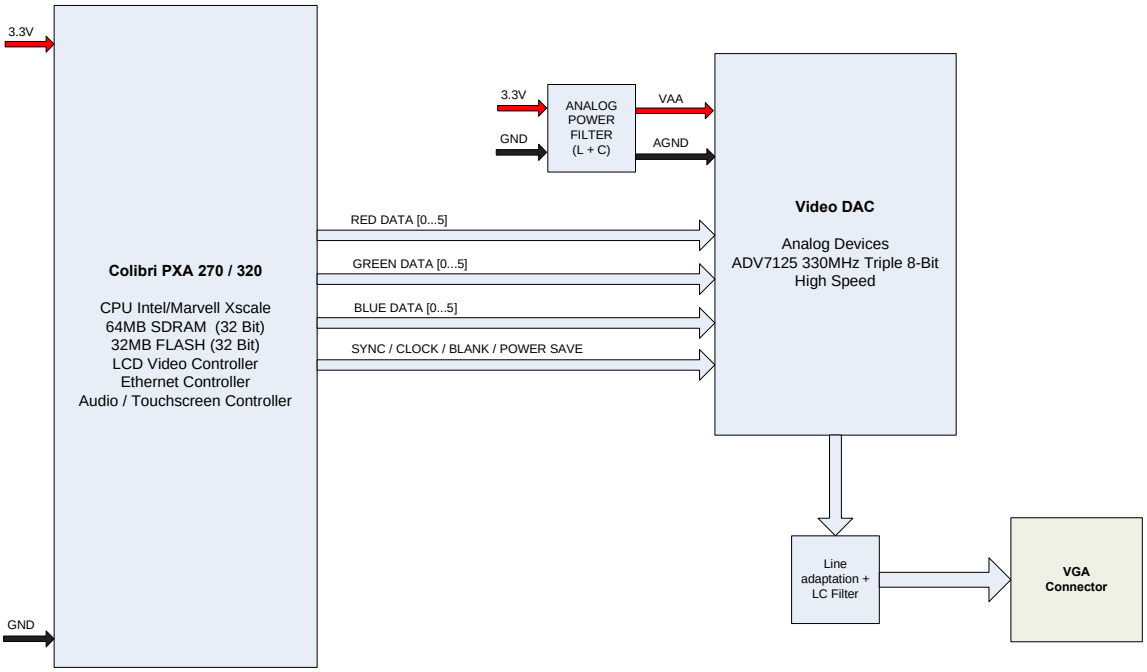
Easy Transition from Colibri PXA270 to the new Colibri PXA320 - Video - 3.8MB
Replace the Colibri PXA270 module by a PXA320 module. Show the doubling of the performance when playing a high resolution DivX video.

7.4. Sortie graphique

Afin de permettre la connexion avec un projecteur, l'appareil devra disposer d'un connecteur écran. Le standard actuel, supporté par tous les projecteurs vidéo, est **la sortie VGA analogique 15 pôles, avec un affichage en une résolution de 800x600 pixels**.

Les cartes processeur Colibri PXA270 & 320 offrent une sortie pour écran LCD numérique, avec une résolution maximale utilisable de 1024x768, correspondant au standard XVGA. Une conversion numérique/analogique (D/A) est donc nécessaire. Il existe plusieurs circuits intégrés permettant d'effectuer cette opération. Celui utilisé dans la carte de démonstration et qui sera réutilisé dans l'iPresenter2 est le ADV7125 de Analog Devices.

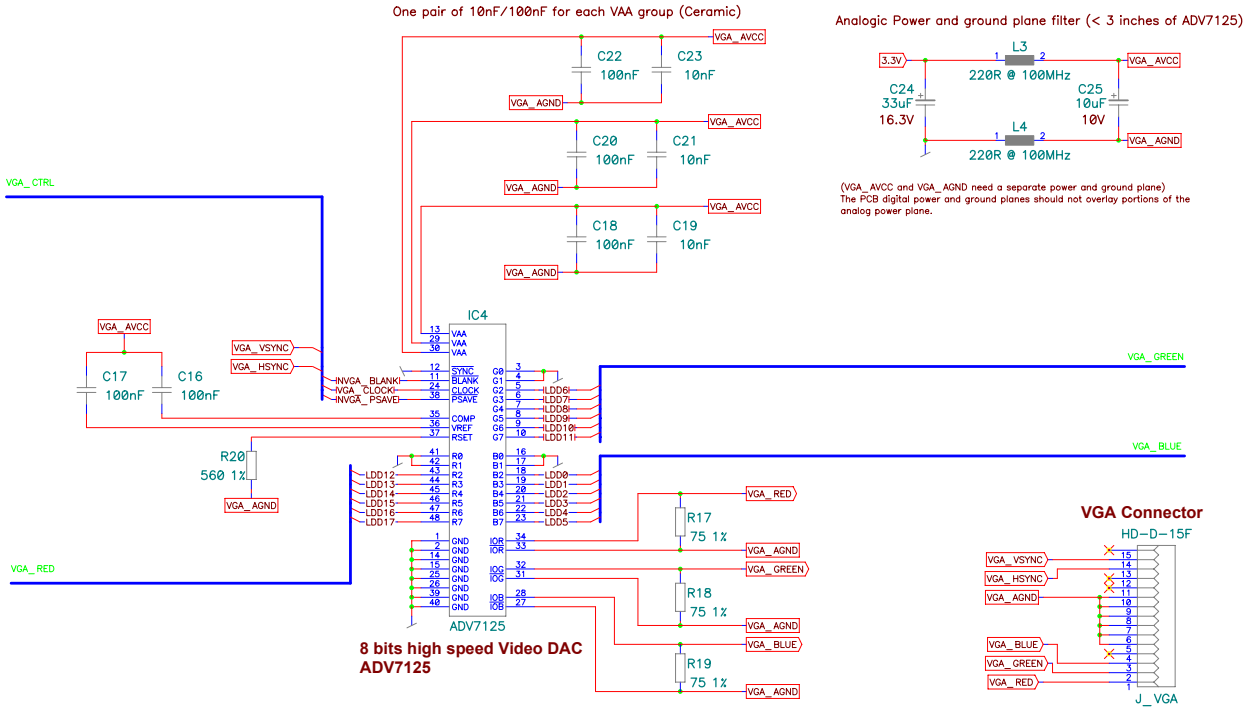
Les signaux entre le processeur et le convertisseur D/A ont été connectés de la même manière que sur la carte de démonstration. Voici le schéma bloc correspondant :



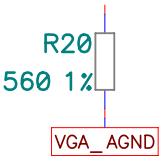
Le convertisseur vidéo ne peut être connecté directement sur l'alimentation 3.3V. Elle nécessite un filtrage particulier car la composante alternative ajoutée par le convertisseur DC/DC peut nuire aux signaux vidéo. Ce filtre est composé de ferrites et de condensateurs.

Une adaptation de ligne est nécessaire en sortie avant le connecteur VGA.

La schématique correspondante est très proche du schéma bloc. Les composants supplémentaires permettent un meilleur découplage et filtrage des alimentations analogiques (C18 à C23) et l'adaptation d'impédance (75 ohms) pour le transport des signaux sur le câble du projecteur (R17 à R19).



Les composants du filtre ont du être recalculés car les valeurs utilisées dans la carte de démonstration n'étant pas connues. La résistance R20 connectée à la pin R7 du convertisseur permet de définir la puissance du signal en sortie du convertisseur. En partant du principe que la longueur du câble liaison jusqu'au projecteur peut être parfois assez longue dans des salles de présentation (beamer au plafond), une résistance de 560 ohm s a été choisie.



7.5. Les ports USB

Le cahier des charges indique que deux ports USB devront être présents sur les côtés (différents) de l'appareil, afin de permettre la connexion de sticks mémoire USB, ou encore de sticks Wifi USB.

Les cartes processeur Colibri PXA270 & 320 offrent deux interfaces USB, dont une peut aussi fonctionner en mode client, pour la connexion à un ordinateur (ActiveSync).

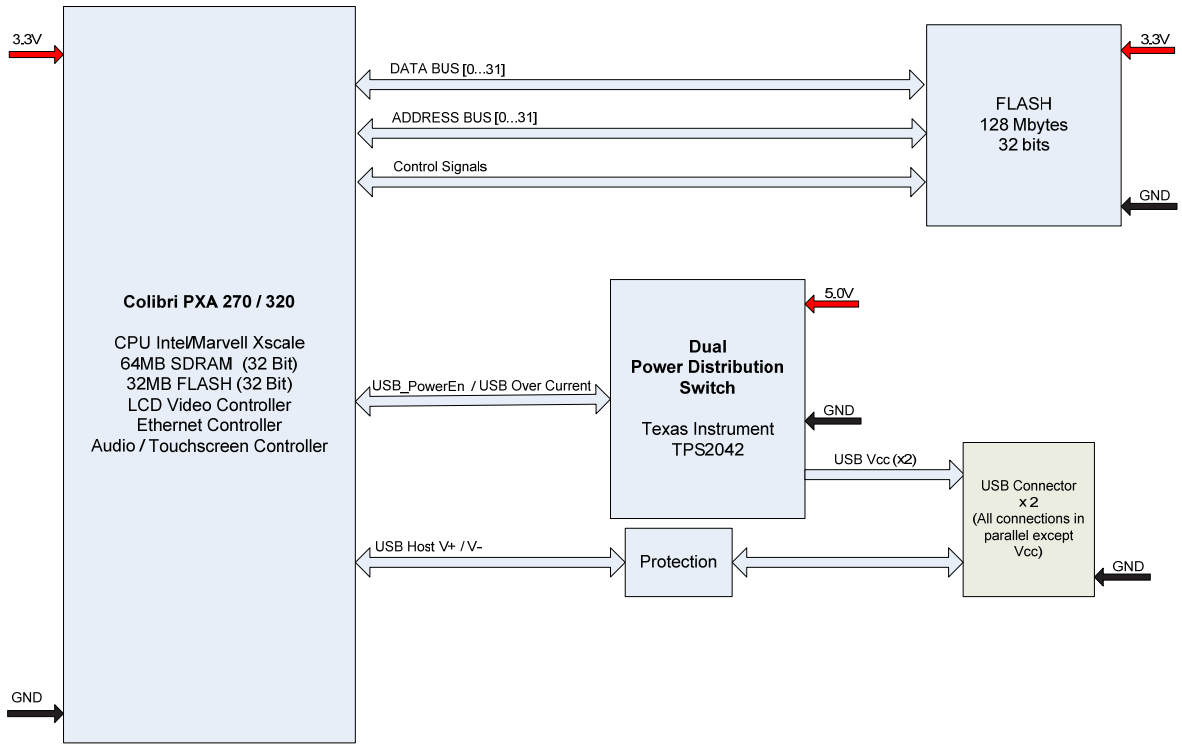
Dans notre cas, il faut configurer les deux interfaces en mode hôte. Ensuite, il est très simplement possible de relier les signaux sortant de la carte processeur directement aux connecteurs USB, moyennant certaines précautions.

Il s'agit avant tout de protéger les lignes où circulent les données des ports contre les différentes sources de bruits passagers (p. ex. décharges électrostatiques), qui peuvent endommager les entrées (de type CMOS très peu protégées) du processeur. Le circuit SN65240 de Texas Instruments permet de protéger 4 lignes simultanément et est donc idéal pour notre application.

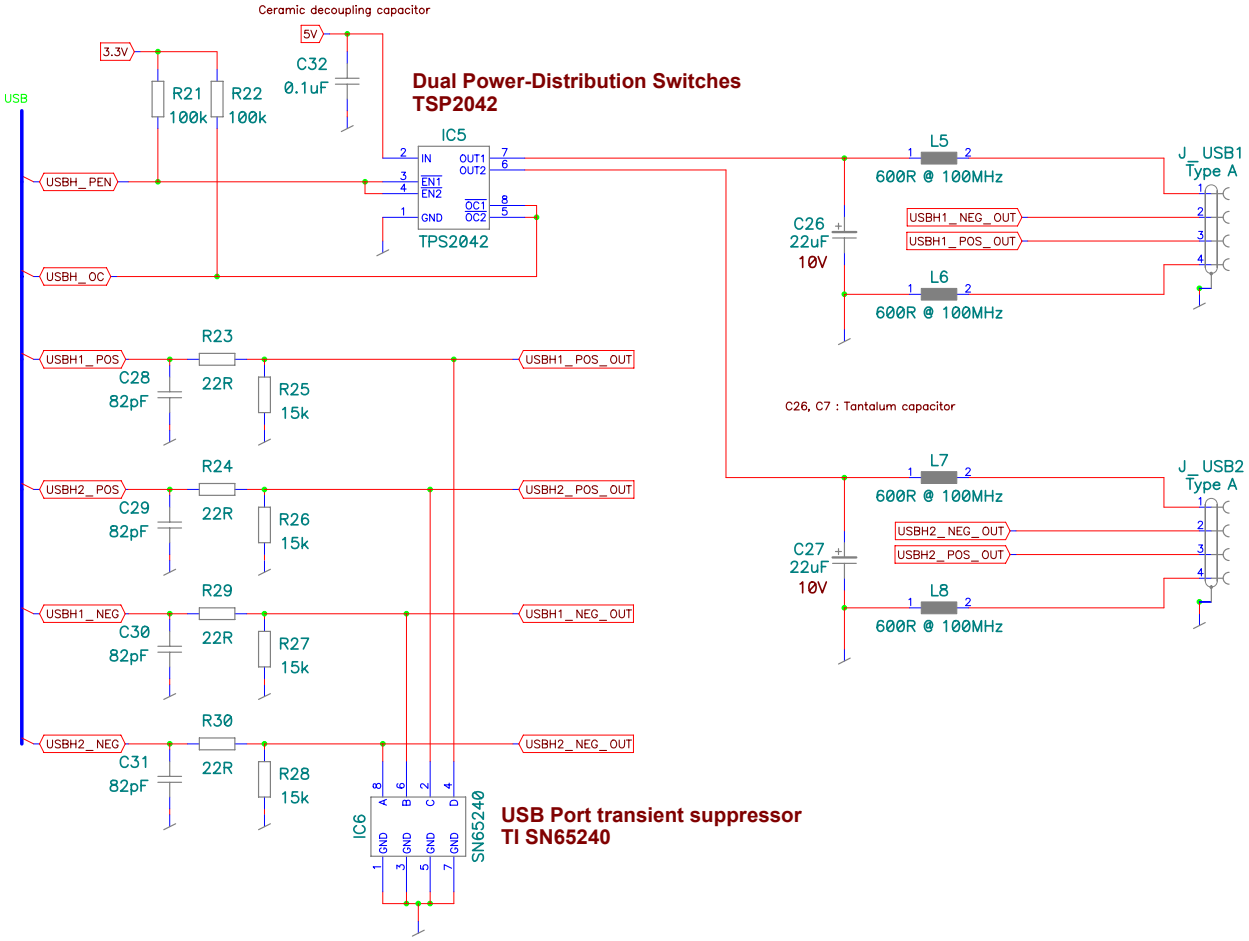
Les deux ports USB étant capables de fournir une alimentation de 5V (contrairement aux données qui circulent sous une tension de 3.3V), il faut contrôler que le courant tiré par les périphériques connectés ne dépasse pas la limite (500mA par port). Le circuit TPS2042, de Texas Instruments également, est conçu pour cette application.

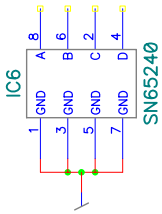
Il peut fournir 5V sur ses deux sorties (OUT1 et OUT2), celles-ci étant activables ou non par un signal logique 3.3V provenant du processeur (EN1 et EN2). Il dispose de deux sorties OC1 et OC2, qui indiquent au processeur que le courant tiré est supérieur à 500mA. De plus, il est équipé d'une protection ESD et anti courts-circuits.

Voici le schéma bloc correspondant à ces explications :



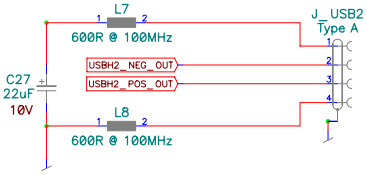
La schématique correspondante est à nouveau assez proche du schéma bloc. Des éléments supplémentaires comme des filtres (C26 à C31), condensateurs de découplage (C32) et pull-up (R21 et R22) sont nécessaires.

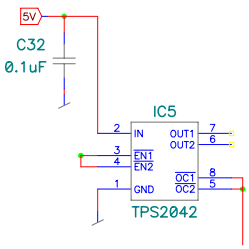




Le premier filtre est composé de 2 résistances d'un condensateur de faible capacité. Il devra être placé sur chaque ligne où circulent les données, pour satisfaire la norme USB. Le SN65240 est connecté entre chaque ligne de données et la masse. Il est composé principalement d'une diode Zener, court-circuitant à la masse les surtensions.

Un deuxième filtre est nécessaire au niveau des connecteurs eux-mêmes. Comment pour le circuit de filtrage du convertisseur vidéo, il est composé de ferrites en série. Celles-ci ont du être recalculées.





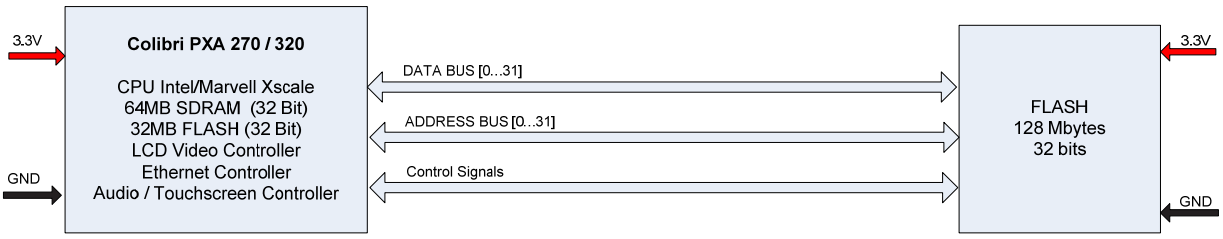
Le TPS2042 est bien alimenté par une tension de 5V. Un condensateur de découplage C32 est nécessaire. Les deux signaux d'activation des sorties EN1 et EN2 sont liés ensemble. Ainsi, les deux ports USB sont activés/désactivé lorsque le processeur le veut. (Ne pas oublier la pull-up R21). La signalisation d'un sur courant est aussi mise en commun jusqu'au processeur. (pull-up R22 nécessaire).

7.6. Mémoire Flash intégrée

Afin de disposer d'assez d'espace mémoire à l'intérieur de la carte, il faudra peut-être rajouter de la mémoire sur la carte. On constate en effet, rapidement, que sur un PXA270 la place mémoire libre sur la FLASH est restreinte :

OS Installé : Windows CE 5.0 Build 1400
Taille mémoire FLASH sur la PXA270 : 32 MBytes
Taille mémoire du Bootloader v2.3 : 42kBytes
Taille mémoire de l'image Toradex Windows CE 1.7 : 7.3 MBytes
Mémoire restante : 32 Mbytes – 7.3 Mbytes = **24.7 Mbytes**

Cet espace sera peut-être insuffisant pour certaines présentations Powerpoint. D'autant plus que les zones mémoires ne seront certainement pas contiguës. On peut donc imaginer une mémoire interne, qui sera connectée sur le bus d'adresse et de données. Le schéma ci-dessous illustre la situation :

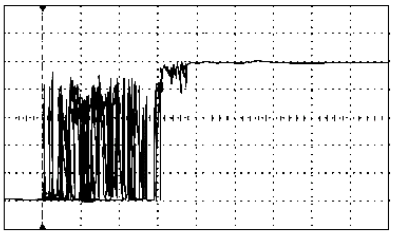


Toradex ne met pas à disposition le memory mapping de son processeur sans devoir souscrire un abonnement à son support technique. Le fabriquant a proposé d'utiliser soit une carte SD ou CompactFlash, soit le Colibri PXA320. Ce dernier est équipé de 1GByte de mémoire FLASH au lieu des 32MBytes du PXA270.

C'est cette solution qui a été choisie. La carte sera donc équipée du PXA320, donc le prix n'est que légèrement supérieur au PXA270. La différence de prix aurait été en partie compensée par l'achat d'une mémoire FLASH supplémentaire.

7.7. Les boutons / LEDs / Signal de Reset

La dernière partie de la schématique concerne les boutons et les LEDs présents sur la carte. Les boutons poussoirs ne peuvent être connectés directement sur une pin du processeur car ils sont victimes de rebonds à chaque pression (comme le montre la figure ci-contre, qui représente la tension en fonction du temps à la sortie d'un interrupteur relié à l'alimentation). Ces rebonds nuisent à la marche normale du processeur, par exemple en provoquant de nombreuses interruptions à la suite. Il existe plusieurs manières de lutter contre ce phénomène :



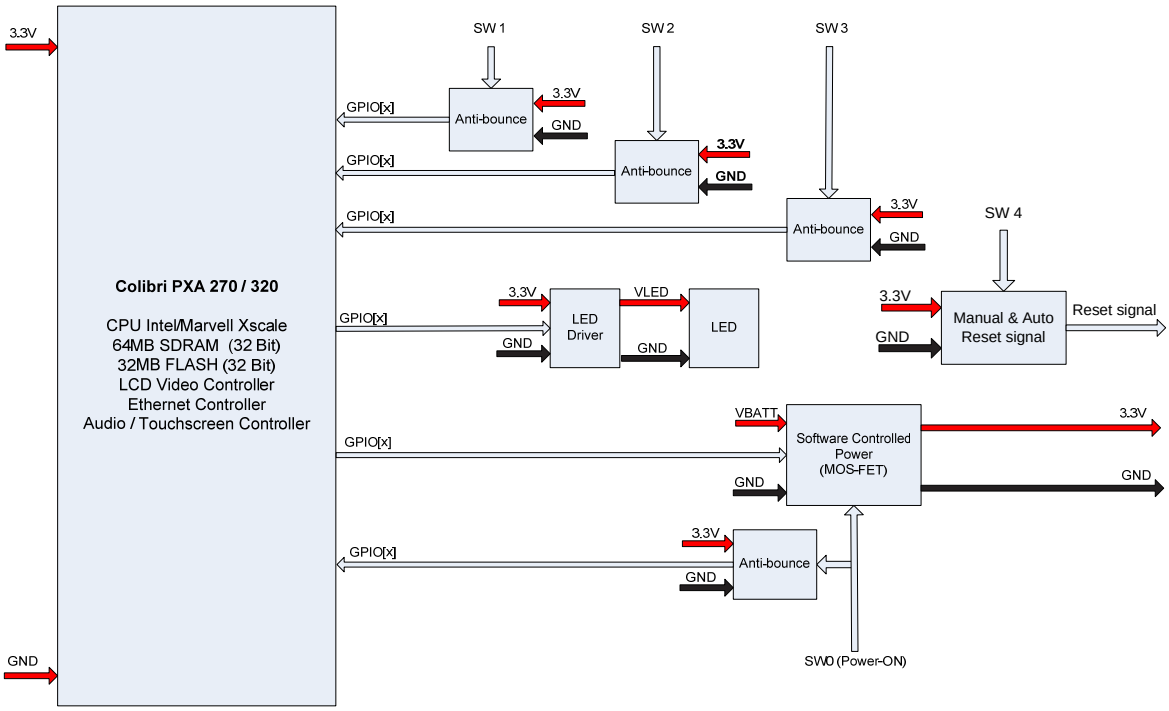
- Implémenter un timer de manière logicielle, qui va attendre la fin des rebonds avant de continuer le programme. Cette solution est simple mais ralentit le processeur, qui devra attendre la fin du timer au lieu d'effectuer d'autres tâches
- Implémenter un système anti-rebonds matériel, qui va donner au processeur un flanc net, ne comportant pas de rebonds. C'est cette solution qui a été choisie.

Pour le cas des LEDs, un signal sortant du processeur est incapable d'attaquer directement une diode électroluminescente : le courant fourni est bien trop faible. Un circuit intermédiaire est nécessaire, il pilotera l'allumage des LEDs en fonction du signal fourni par le processeur.

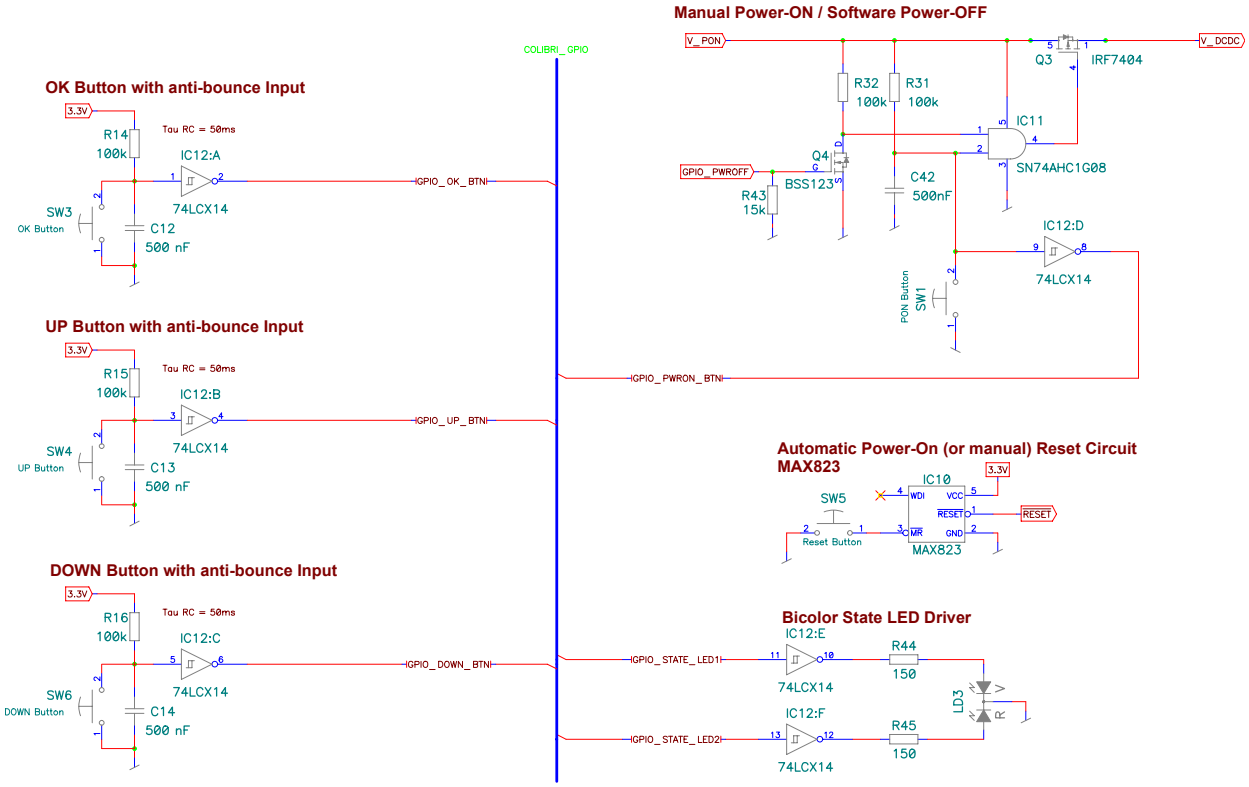
Une dernière fonctionnalité doit encore être implémentée : un signal de reset. Lorsque le processeur est mis sous tension, il requiert d'être mis dans un état de reset pendant un certain temps. On va utiliser un circuit spécial qui va détecter lorsque le signal d'alimentation est stable. A ce moment-là, il fournira un signal de reset d'une durée suffisante pour mettre à zéro le processeur (140 millisecondes). Ce circuit accepte aussi un reset manuel, qui peut être fourni via un dernier bouton poussoir.

Trois boutons (Up, Down, OK) sont lus directement par le processeur. Le bouton de remise à zéro est connecté sur l'entrée manuelle du circuit de reset. Un dernier bouton permet la mise sous-tension de la carte. La mise hors-tension est commandée par le processeur. A noté que l'état du bouton de mise sous-tension peut être lu par le processeur également.

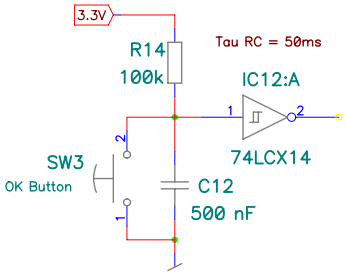
Voici le schéma bloc représentant toutes ces fonctionnalités :

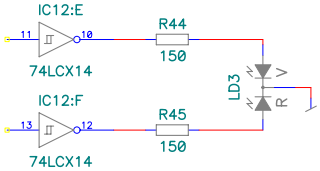


La schématique correspondante se trouve ci-dessous. La ligne centrale est un bus reliant tous les signaux des boutons et des LEDs au processeur.



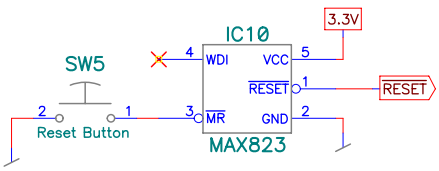
Il existe plusieurs manières de réaliser des systèmes anti-rebonds hardware. Un des plus efficaces, sans pour autant être trop coûteux, est celui présenté dans la figure de droite. Il est composé d'un circuit RC (R14-C12) qui se décharge rapidement lorsque l'on appuie sur le bouton, et se recharge lentement lorsqu'on le relâche. Il permet donc de retarder la remontée du signal lors de rebonds. Le trigger de Schmidt (IC12 :A), un circuit comportant 6 triggers dans une puce, transforme ce signal qui varie lentement (que les entrées processeurs n'apprécient pas) en un flanc net.

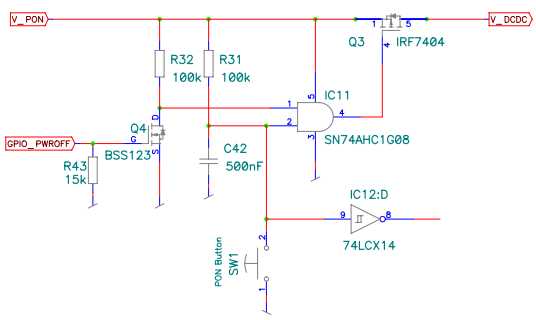




La LED choisie est un modèle bicolore (rouge/vert). Cela permet de représenter plusieurs informations avec une seule diode. Le même trigger de Schmidt, qui est capable de fournir jusqu'à 25mA par sortie, sera utilisé comme pilote de LED. L'entrée du trigger provient directement du processeur.

Le circuit MAX823 de MAXIM est conçu pour fournir un signal de reset propre d'une durée de 140ms. Il est simplement connecté à l'alimentation 3.3V qu'il surveille jusqu'à ce qu'elle soit stable. Le bouton SW5 permet une mise à zéro manuelle par l'utilisateur. Le signal nommé RESET est lié directement au processeur.





La mise sous-tension de l'appareil est assurée par le circuit ci-contre. Le courant de la batterie ne peut circuler et alimenter les convertisseurs DC-DC que si le transistor MOS-FET Q3 est à l'état passant. Celui-ci peut l'être que lorsque la porte logique ET de IC11 fournit un zéro et donc que l'une des deux entrées de la porte soit à zéro. Si rien ne se passe (interrupteur ouvert et Q4 bloqué), les résistances pull-up R31 et R32 forcent la porte ET à sortir un 1. Si l'interrupteur de mise sous tension SW1 est appuyé et maintenu, il force la 2ème entrée de la porte à zéro et provoque ainsi le passage à l'état passant de Q3. Les alimentations démarrent et le processeur peut démarrer à son tour. Lorsque le processeur en sera capable, il fournira un signal positif sur l'entrée GPIO_PWROFF afin de rendre Q4 passant, et donc de maintenir un zéro en sortie de la porte ET. L'alimentation est ainsi maintenue même si SW1 est relâché.

Le processeur est maintenant responsable de l'alimentation de la carte. A noter que le signal que donne SW1 est aussi transmis au processeur via la porte D du trigger de Schmidt. Le filtre RC est aussi présent (R31 et C42) sur cette entrée. A noter que si le bouton SW1 est relâché avant que le processeur n'ait eu le temps de maintenir l'alimentation, le système s'éteint.

7.8. Bilan

La réalisation de la schématique est maintenant terminée. Elle est disponible en annexe 03. Une version informatique au format PCAD de la schématique et du routage PCB est disponible sur le CD-ROM en annexe. (*1_Schematics\ iPresenter2.sch*).

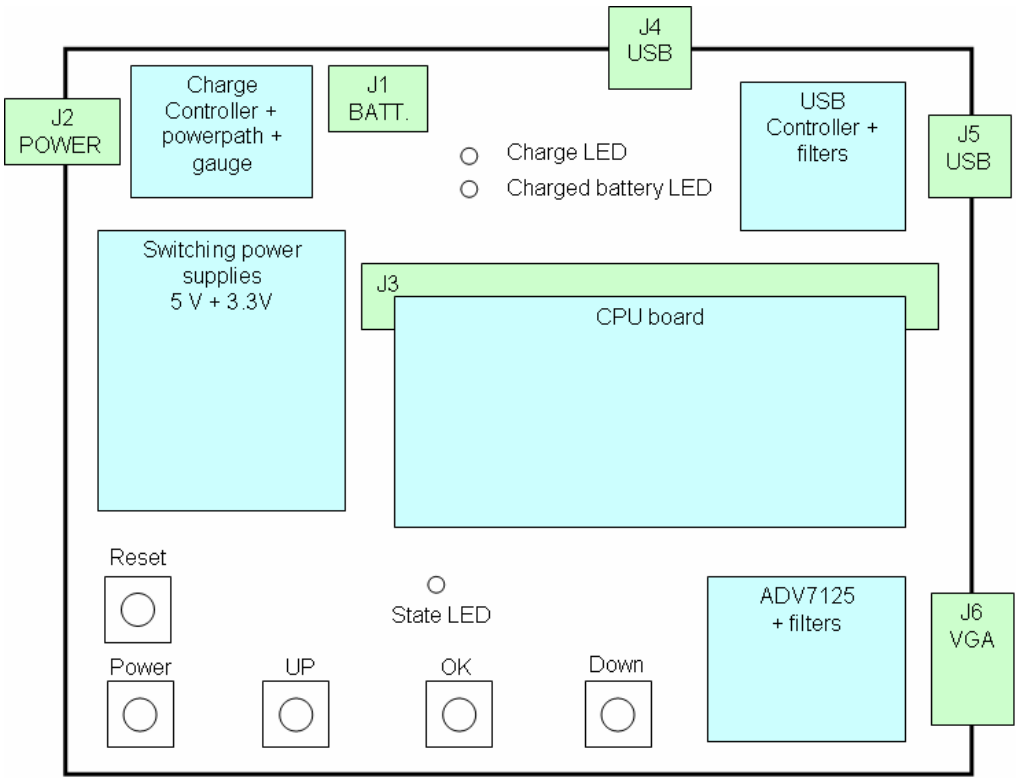
8. Placement et routage de la carte

La schématique étant terminée, il est temps de penser à l'implémentation physique des composants et des lignes sur un PCB. La carte prototype ne sera pas aussi petite que celle du produit final. Le format choisi par Mr. Pompili est de 100mm * 85mm. Compte-tenu du nombre de pistes à dessiner et de la nécessité d'avoir des plans de masse et d'alimentation, il sera nécessaire de produire un PCB d'au minimum 4 couches. La HES-So Valais à Sion est capable de fabriquer des cartes jusqu'à 2 couches, il est donc nécessaire de le faire imprimer à l'extérieur.

Un fichier Microsoft Excel, placé en annexe 04 contient la liste de tous les composants présents sur la carte selon leur dénomination et leur référence de commande. Le prix indiqué pour chacun ainsi que le total correspond au coût pour la production d'une carte.

8.1. Plan d'implantation des composants

La première étape consiste à établir un plan sommaire sur lequel la position des composants principaux est définie. Dans notre cas, il s'agit de placer les connecteurs, les boutons et LEDs ainsi que les blocs de circuits tels que alimentations, chargeur, etc…



Comme précisé dans la spécification, les ports USB sont placés sur des faces différentes et proches du circuit de protection et du distributeur de tension.

Le connecteur batterie se trouve au plus proche du circuit de charge ainsi que du connecteur d'alimentation. Les deux LEDs d'état de charge se trouvent aussi à côté.

Le connecteur VGA est placé prêt du convertisseur D/A. Les boutons sont groupés au fond de la carte avec la LED d'état.

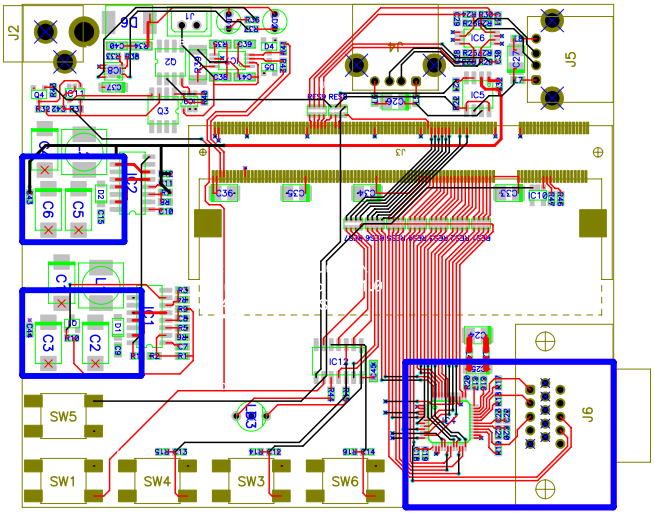
Les circuits d'alimentation ont été éloignés le plus possible de la sortie graphique pour éviter des problèmes de bruit rayonné.

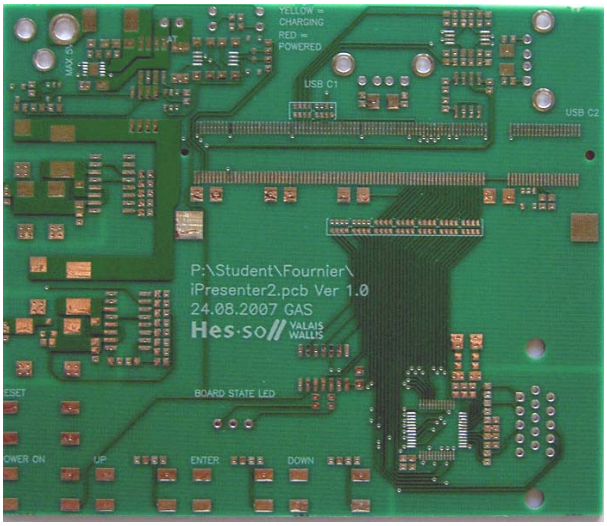
8.2. Routage et réalisation de la carte

Le routage a été effectué par l'atelier électronique de la HES-So Valais. C'est cette solution qui a été choisie car leur expérience dans ce domaine permet d'effectuer cette opération en quelques jours. Il m'aurait fallu plus d'une semaine pour arriver à un résultat sans doute inférieur. (le routage de la carte d'alimentation effectué précédemment m'avait donné une idée assez précise du temps à investir pour cette étape). L'atelier est parvenu à router la carte sur 4 couches au format désiré.

Il était très important de spécifier dans la schématique les courants qui circulent dans les lignes d'alimentation. Cela permet de déterminer quelle est la taille de la piste de cuivre correspondante.

La réalisation physique est donnée à une entreprise externe. Celle-ci a livré deux PCB 4 couches identiques avec trous métallisés et pads étamés, facilitant la soudure des composants.





8.3. Soudure des composants

Cette opération délicate a été effectuée en 2 jours. Il est important de soigner cette étape et de contrôler toutes les soudures pour ne pas perdre de temps à chercher la raison d'erreurs de fonctionnement dues à de mauvais contacts ou à des soudures froides.

Le premier composant qu'il faut souder est le chargeur de batterie SC806 car il comporte un pad thermique sur la face touchant le PCB. Cette étape a été réalisée en plaçant de la pâte à base d'étain sur le pad, en plaquant le composant dessus puis en chauffant grâce à une machine adéquate la zone jusqu'à ce que la pâte se liquéfie et soude les deux surfaces entre elles. Ensuite seulement les pattes de la puce sont soudées.

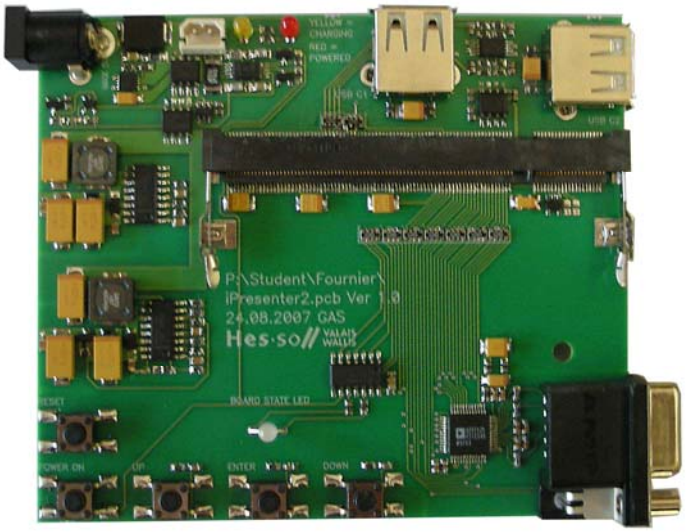
Les composants les plus petits doivent être soudés en premiers avec les grosses puces, condensateurs et self et LEDs puis enfin les connecteurs.

Un problème d'ordre mécanique a été rencontré lors de la mise en place du connecteur VGA. En effet, la place prévue pour le module Colibri est insuffisante et touche le connecteur écran (erreur de dessin du connecteur CPU). Il a fallu enlever une partie de la matière du connecteur pour faire passer le module. Le problème sera évidemment corrigé sur la version finale.

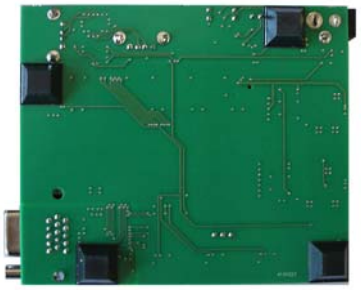
Le reste de l'opération s'est déroulé sans encombre. La grosse difficulté se situa au niveau du connecteur CPU J3, dont les pins ont dû être soudées une à une, car il était difficile de réaliser plusieurs soudures simultanément (pins dorées).

8.4. Photos de la carte

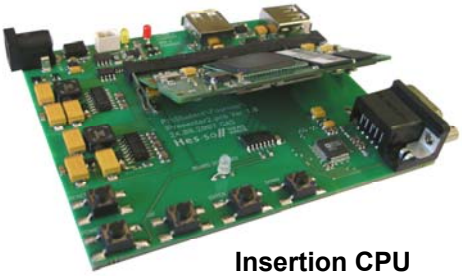
Voici quelques photos de la carte terminée. Quatre pieds en caoutchouc ont été collés sur la face inférieure pour qu'elle ne glisse pas et pour éviter les courts-circuits avec des éléments métalliques.



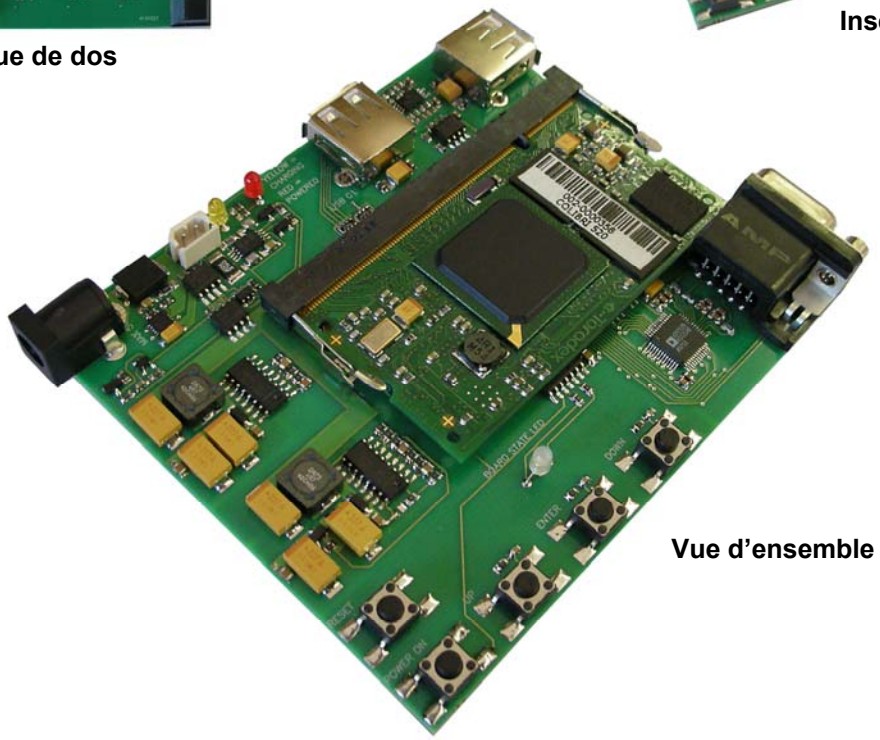
Vue de face, sans CPU



Vue de dos



Insertion CPU



Vue d'ensemble

9. Tests de la carte

9.1. Tests fonctionnels

9.1.1. Procédure de test

Afin de bien vérifier toutes les fonctionnalités de la carte maintenant soudée, il est nécessaire d'établir une procédure de test complète. Les tests doivent être effectués dans un ordre précis pour ne pas endommager le matériel en cas d'erreur. Cette procédure a été écrite avant tout branchement de la carte et les résultats sont scrupuleusement notés et contrôlés. Il est possible de poursuivre uniquement en cas de réussite du test précédent. Dans le cas contraire, des solutions devront être trouvées avant de continuer.

Certains tests se limitent à des mesures de tension ou d'impédance. Les résultats de ceux-ci sont directement notés dans la procédure de test. Dans le cas de mesures complexes à l'oscilloscope par exemple, les résultats sont expliqués dans les points ci-dessous.

La procédure de test complète se trouve en annexe 05.
Les tests N° 36 et N° 45 avaient échoués à cause de problèmes de soudures. Malgré l'attention particulière portée au contrôle de celles-ci, quelques connexions étaient mauvaises à un endroit particulièrement sensible : la sortie graphique. Des signaux de relativement haute fréquence passent dans ces fils et si la soudure n'est pas parfaite, les signaux passent mal ou ne passent plus du tout.

9.1.2. Alimentations

La carte d'essai préliminaire a permis de confirmer le bon fonctionnement du circuit d'alimentation. Il est cependant nécessaire de refaire des tests plus poussés, au vu du nouveau design utilisé sur la carte prototype de l'iPresenter2.

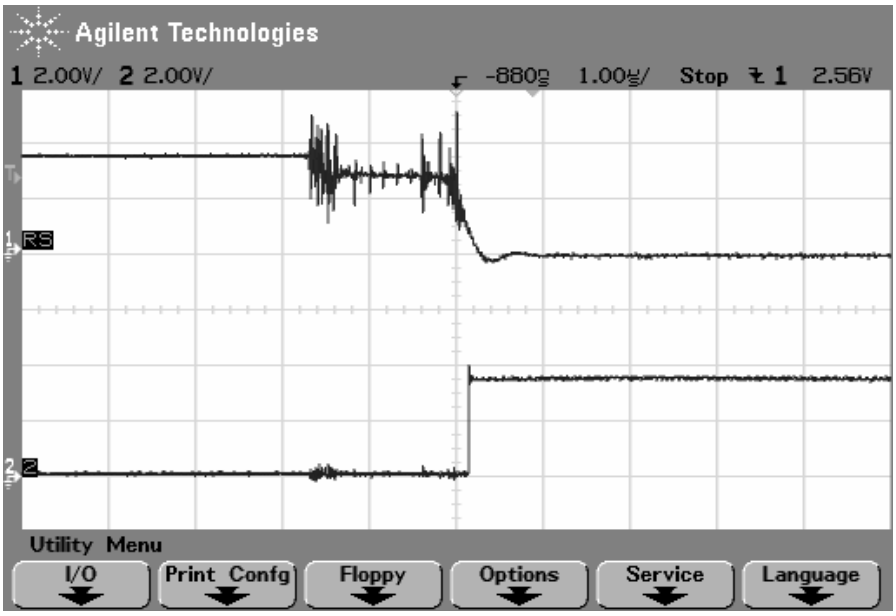
Des tests ont d'abord été effectués à vide, puis avec l'un et l'autre processeur. Les ports USB ont été chargés avec divers périphériques afin de tester leurs limites.

La procédure de test complète est disponible en annexe 06. Elle se trouve aussi sous format informatique sur le CD-ROM en annexe (4_Documentation\ AN06_AlimIPres2Test.doc).

On peut donc conclure que la stabilité des tensions reste dans les tolérances dans tous les cas de figure testés. Le test est donc un succès.

9.1.3. Dispositif anti-rebonds des boutons

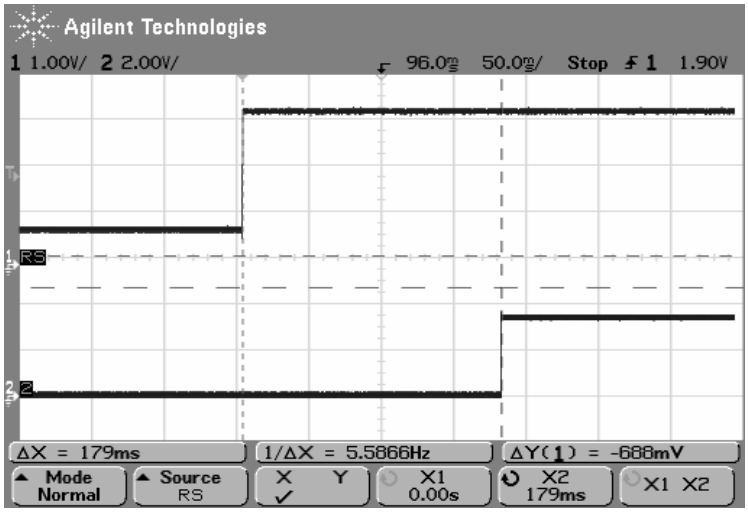
Ce test permet de vérifier que le circuit RC et le trigger de Schmidt sont bien dimensionnés et fonctionnent correctement. A l'aide de l'oscilloscope, une première sonde est placée sur la patte de sortie de l'un des boutons poussoirs. La seconde est connectée sur la pin de sortie du trigger de Schmidt correspondante. L'oscilloscope est configuré en monocoup avec trigger sur le canal de la première sonde. Voici les résultats :



On constate que le signal après le circuit RC et le trigger est très bien nettoyé. Les oscillations qui ont lieu dans le signal du premier canal sont absentes en bas. De plus, le signal est inversé, ce qui est correct. Cela nous permet d'avoir un flanc montant net lors de la pression sur un bouton.

9.1.4. Reset

Lors de la mise sous tension de la carte, un signal de reset doit être mis à zéro puis remis à Vcc dès que l'alimentation 3.3V est stable pour assurer la bonne mise en marche du microprocesseur. C'est le but du MAX823 placé sur la carte. Il assure un signal de reset d'au minimum 140ms. C'est cette donnée que nous allons vérifier. Cette mesure se fera de nouveau avec l'oscilloscope, sonde 1 placée sur la ligne de sortie de l'alimentation 3.3V et sonde 2 à l'entrée du signal de reset de la carte Colibri, mode monocoup. Voici les résultats :



Le signal de reset est bien présent sur la 2ème trace. Il maintient à un niveau bas l'entrée reset du processeur pendant 179ms. C'est un temps largement suffisant pour assurer le reset de la carte Colibri.

9.1.5. Fonctionnement des ports USB

Lors des tests de reconnaissance d'une clé USB sur les deux ports présents sur la carte, l'un a échoué. Sur le premier port (port en haut de la carte), tout type de périphérique standard est reconnu (clé USB,

clavier, souris). Par contre sur le 2ème port à droite, l'alimentation est bien présente mais aucun signal ne transite et le périphérique n'est pas reconnu.

Je pensais dans un premier temps qu'il s'agissait d'un problème de soudures ou de valeurs de composants au niveau du filtre sur les lignes de données, mais tout a été refait et contrôlé. Le résultat était le même.

Dès ce moment-là, il a fallu chercher plus loin au niveau de l'utilisation des GPIO pour comprendre les raisons de ce dysfonctionnement. Voici un tableau récapitulant l'utilisation des entrées/sorties du Colibri pour les boutons et les ports USB :

Pin #	GPIO #	Fonction par défaut	Mode par défaut	Fonction utilisée	Mode utilisé
67	17	PWM_OUT1	OUT	STATE_LED_1	OUT
69	20	nSDSC2	OUT	STATE_LED_2	OUT
73	52	CIF_DD4	I/O	GPIO_OK_BTN	IN
75	53	USB_P2_3	OUT	GPIO_UP_BTN	IN
77	82	CIF_DD5	I/O	GPIO_DOWN_BTN	IN
79	83	CIF_DD4	I/O	GPIO_PWRON_BTN	IN
129	89	USBH_PEN	OUT	USBH_PEN	OUT
131	88	USBH_OC	IN	USBH_OC	IN
137	**41**	**USB_DET**	**IN**	**GPIO_PWROFF**	**OUT**
139	USBH_P1	USB Host 1 Pos.	I/O	USBH1_POS	I/O
141	USBH_N1	USB Host 1 Nég.	I/O	USBH1_NEG	I/O
143	USBC_P2	USB Client 2 Pos.	I/O	USBH2_POS	I/O
145	USBC_N2	USB Client 2 Nég.	I/O	USBH2_NEG	I/O
194	117	SCL	OUT	SCL	OUT
196	118	SDA	I/O	SDA	I/O

On remarque que la pin 137 du module est configurée par défaut pour détecter la connexion d'un périphérique sur le port USB client. La carte iPresenter2 n'utilisant pas le port client, j'ai connecté cette pin au circuit de maintien de l'alimentation. Par défaut cette pin est configurée en entrée et attend un flanc montant qui indique qu'un périphérique est connecté sur le port USB Client. De plus, le port USB client et le 2ème port USB hôte partagent les mêmes lignes de données. Dès qu'on la configure en sortie, le port concerné se bloque et ne communique plus. Il n'est pas possible de désactiver séparément le pilote du port client de celui hôte dans la base de registre de Windows CE. La seule solution trouvée a été d'utiliser une autre pin pour le maintien de l'alimentation (GPIO37, pin 133), et de forcer la pin USB_DET à zéro. De cette manière, aucun périphérique n'est détecté et le port fonctionne en tant qu'hôte correctement.

Ce problème étant résolu, les deux ports USB sont opérationnels, il est possible de brancher les périphériques suivants directement reconnus : clavier, souris et clés de données. Le transfert d'un gros fichier sur les 2 ports via une clé de données permet de confirmer la bonne marche des ports.

9.2. Tests de performance

9.2.1. Consommation et autonomie

Afin d'évaluer l'autonomie de l'appareil, il va falloir mesurer le courant que débite la batterie lorsqu'il est en marche. Pour cela, on va utiliser une pince ampère-métrique reliée à une entrée de l'oscilloscope. Le premier élément à définir est le courant qui circule depuis la batterie pour différentes valeurs de sa tension. Aucun périphérique n'est branché sur les ports USB et le processeur n'effectue pas de tâche spécifique (bureau Windows).

Avec le processeur PXA270 :

VBatt [V]	4.09	3.99	3.90	3.80	3.72	3.60	3.50	3.40	3.20	3.00	2.80
Ibatt [mA]	378	386	390	392	385	402	407	416	438	465	500
Wbatt [W]	1.55	1.54	1.52	1.49	1.43	1.44	1.42	1.41	1.40	1.39	1.4
Ton [h]	0h00	0h31	1h11	1h58	2h57	4h52	5h22	5h28	5h39	5h42	**5h44**

Avec le processeur PXA320 :

VBatt [V]	4.08	4.00	3.89	3.81	3.71	3.59	3.48	3.41	3.19	3.01	2.82
Ibatt [mA]	369	370	375	382	390	389	394	407	429	451	492
Wbatt [W]	1.50	1.47	1.46	1.45	1.44	1.40	1.37	1.39	1.37	1.36	1.39
Ton [h]	0h00	0h34	1h18	2h09	3h16	5h23	5h54	6h01	6h13	6h16	**6h19**

On constate que la consommation de la carte à vide est inférieure mais proche de celle estimée lors du dimensionnement de la batterie (435mA @ 3.6V), ce qui est un bon point. Il est aussi logique de voir la consommation en watt de la batterie diminuer au cours de la décharge, cela est dû au régulateur linéaire. Il est maintenant nécessaire d'estimer la consommation de l'appareil dans différents cas spécifiques (tension de la batterie : 3.6V) avec :

PXA270

- une clé USB branchée (moyenne), aucune tâche processeur : **437mA (1.57W)**
- une clé USB branchée, une souris USB, aucune tâche processeur : **488mA (1.76W)**
- un programme utilisant 100% du temps CPU et rien sur les ports USB : **650mA (2.34W)**
- une clé + une souris USB et un programme utilisant 100% du temps CPU : **702mA (2.52W)**

PXA320

- une clé USB branchée (moyenne), aucune tâche processeur : **390mA (1.40W)**
- une clé USB branchée, une souris USB, aucune tâche processeur : **458 mA (1.65W)**
- un programme utilisant 100% du temps CPU et rien sur les ports USB : **773mA (2.78W)**
- une clé + une souris USB et un programme utilisant 100% du temps CPU : **831mA (2.99W)**

Premier point intéressant : notre appareil consomme moins de 3W lors d'une utilisation normale. On peut aussi voir la différence entre le PXA270 et PXA320 au repos. Ce dernier consomme environ 2% moins de courant alors qu'il tourne à une fréquence 35% supérieure ! Cela est dû à la finesse de gravure des puces, bien plus petite sur le PXA320 ainsi qu'aux précautions qu'à pris le fondeur pour limiter les courants de fuite.

La différence la plus frappante est le ratio entre la consommation à vide et celle en charge des deux processeurs. Cette fois-ci la différence de fréquence est clairement en cause. Pour le PXA270, celui-ci est de 2 : 3 alors que pour le PXA320 il est de 1 : 2.

La mesure suivante est l'une des plus importantes : l'autonomie. Elle a déjà été calculée dans le tableau ci-dessus pour un cas sans clé USB ni tâche processeur. Le même test a été effectué pour différents cas. Il faut considérer que la batterie au Lithium Ion est neuve et que donc sa capacité va diminuer quelque peu avant de se stabiliser. Il faut compter environ 20% de capacité en moins. Ce qui nous ramène à l'autonomie suivante pour le PXA270 :

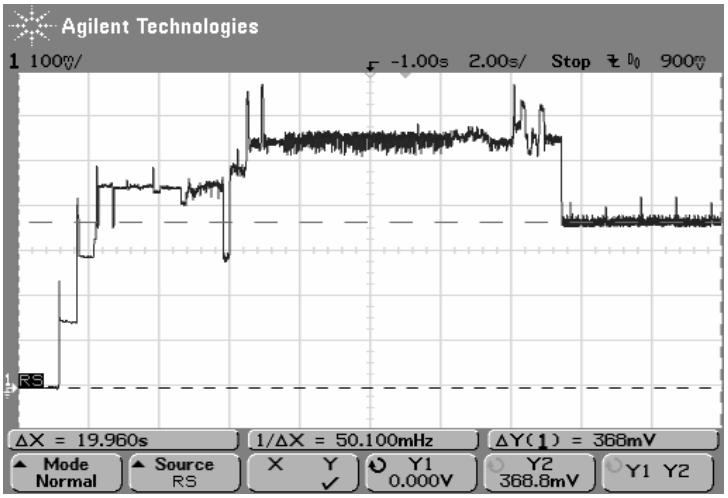
- Sans clé USB, aucun tâche processeur : 4H35
- Sans clé USB, avec un programme utilisant 100% du temps CPU : 2H34
- Avec une clé + une souris, avec un programme utilisant env. 50% du temps CPU : **3H10**

Les mêmes mesures effectuées avec le PXA320 :

- Sans clé USB branchée, aucun tâche processeur : 5H03
- Sans clé USB branchée, avec un programme utilisant 100% du temps CPU : 2H10
- Avec une clé + une souris, avec un programme utilisant env. 50% du temps CPU : **2H52**

Dans tous les cas, l'appareil a une autonomie supérieure aux 2 heures définies dans la spécification. Le test est donc un succès.

Un dernier point intéressant est la consommation de l'appareil au démarrage. Cette courbe est facilement réalisable à l'oscilloscope en mode monocoup sur une longue durée (2s/div). Processeur PXA270, aucun périphérique USB n'est connecté lors de ce test. Voici le résultat :



On remarque avant tout que la consommation au démarrage varie énormément, avec des pointes de courant à plusieurs endroits. On comprend mieux l'importance d'avoir une batterie capable de débiter de forts courants (résistance interne faible). Au milieu du graphique, on constate que la consommation au chargement de Windows Mobile est à son maximum au niveau du processeur, puis retombe dans la zone de valeurs mesurées à la page précédente. (368mA).

9.3. Coût total du prototype

Bien que n'étant astreint à aucun budget précis, il est intéressant et important de calculer le coût total que représente la carte développée. Voici les totaux calculés (hors coûts de production du PCB) :

Elément	Prix
Composants carte	214.20 CHF
Module Colibri PXA270	194.80 CHF
Module Colibri PXA320	227.50 CHF
TOTAL iPresenter2-270	**409.00 CHF**
TOTAL iPresenter2-320	**441.70 CHF**

Court Euro 19.11.2007 : 1. 1.637

Bien évidemment et comme tout produit industriel, l'achat de pièces et de modules en plus grand nombre permettrait de faire baisser le prix de manière considérable. Il est cependant important de remarquer que le coût total des composants et celui des modules sont semblables. Cela semble relativement cher. Les éléments principaux qui font grimper le prix de la carte sont :

- La batterie (55.-)
- Les 2 convertisseurs MAXIM (28.50)
- Les connecteurs CPU et graphique (35.35)

Ils représentent à eux-seuls environ 30% du prix de la carte. Les connecteurs ne peuvent être changés. On peut par contre imaginer utiliser une autre batterie moins coûteuse (et donc de plus petite capacité), ce qui semble possible au vu de la marge d'autonomie. Au niveau de l'alimentation, il faudrait

reconcevoir l'ensemble à base d'un autre circuit comme le LTC3533. Il faudra faire attention à ce que les performances restent intactes parce qu'elles sont vitales (bruit et stabilité.)

9.4. Bilan

Les alimentations sont d'une stabilité très bonne dans tous les cas de figures. Certaines modifications ont dû être apportées, principalement au niveau des ports USB. *La schématique a été corrigée et est à jour, il faudra par contre modifier le routage PCB pour la version finale.*

Tous les tests fonctionnels importants ont été menés à bien. Il est toujours probable qu'une erreur de conception soit trouvée plus tard lors de la conception logicielle (p. ex. bus SMBus, erreurs de transmission, etc..) mais les éléments vitaux semblent fonctionner correctement.

Il est temps de passer au développement du logiciel de lecture et de gestion de fichiers Powerpoint.

10. Développement logiciel sous Windows CE

10.1. Introduction

Maintenant que la partie matérielle est fonctionnelle, il est possible de débuter la partie la plus conséquente du projet : le développement logiciel. Comme dit précédemment, les cartes Colibri sont préinstallées avec un système d'exploitation Windows CE 5.0 de Microsoft fonctionnel. On peut démarrer et utiliser l'iPresenter2 car les pilotes graphiques et USB sont déjà écrits et implémentés dans le module.

Il suffit de le mettre sous tension à l'aide du bouton Power (il faut maintenir la pression sur le bouton car le système de maintien d'alimentation par le processeur n'est pas opérationnel). Ensuite, il faut connecter un clavier et une souris sur les ports USB. Le système est utilisable.

Il n'est pas contre pas possible de programmer l'iPresenter2 depuis la carte prototype, car elle est dépourvue d'un connecteur USB de type client pour la connexion à l'ordinateur. Pour cela, il faudra insérer le module dans la carte de démonstration Colibri Evaluation Board présentée au chapitre 4.2.

10.2. Cahier des charges

Afin de planifier au mieux le déroulement de la programmation logicielle, il est tout d'abord nécessaire de connaître les besoins et les objectifs à atteindre. En voici le détail :

- ✓ **Faire fonctionner et intégrer au système un lecteur de fichiers Powerpoint**
- ✓ **Etre capable de lire l'état des boutons présents sur la carte**
- ✓ **Etre capable de piloter la LED bicolore présente sur la carte**
- ✓ **Communiquer via le bus I^2C (SMBus) avec la puce de gestion batterie**
- ✓ **Gérer les accès au système de fichier de la mémoire FLASH et aux clés de données branchées sur les ports USB**
- ✓ **Développer une application graphique de haut niveau permettant l'affichage de menus et la coordination de toutes les autres fonctions ci-dessus**
- ✓ **Etre capable de changer la résolution d'affichage dans les modes VGA, SVGA et XVGA (objectif mis à jour après discussion avec Mr. Pompili)**

On constate que le travail à effectuer est très conséquent, il s'agira de le découper de manière optimale de façon à ne pas perdre de temps. Aborder sans aucune connaissance préalable la programmation sur un système d'exploitation dont le fonctionnement est flou voir inconnu est suicidaire. La première étape consistera donc à s'informer et à comprendre les bases de Windows CE.

11. L'architecture Windows CE

11.1. Généralités

Le but de ce chapitre est de comprendre les bases sur lesquelles Windows CE version 5 sont construites et d'en saisir le fonctionnement global.

Comme précédemment cité, Windows CE n'est pas un système d'exploitation livré sous la forme d'un exécutable et paramétré par la suite pour nos besoins. En effet, l'une des prérogatives fixées par Microsoft est de construire un OS pour applications mobiles très flexibles (destiné aux appareils tels que les Pocket PC, Smart Phone, systèmes embarqués dans les véhicules, écrans dans les jets privés, etc..). Il doit donc être capable de fonctionner avec une énorme quantité de matériel qui sont totalement incompatibles entre eux (processeur, mémoire, périphériques, carte graphique, etc..). De plus, sa taille doit être aussi faible que possible, pour être portable sur le plus large panel d'appareils sur le marché.

Partant de ce fait là, on comprend mieux pourquoi il n'existe pas une version unique de Windows CE. Pour rappel, une multitude de systèmes basés sur cet OS existent (Windows Mobile, Windows embedded, Windows .NET, …). Ces versions ont toute la même base mais un certain nombre de composants logiciels (tels que le framework .NET ou la suite Office Mobile) y sont ajoutés.

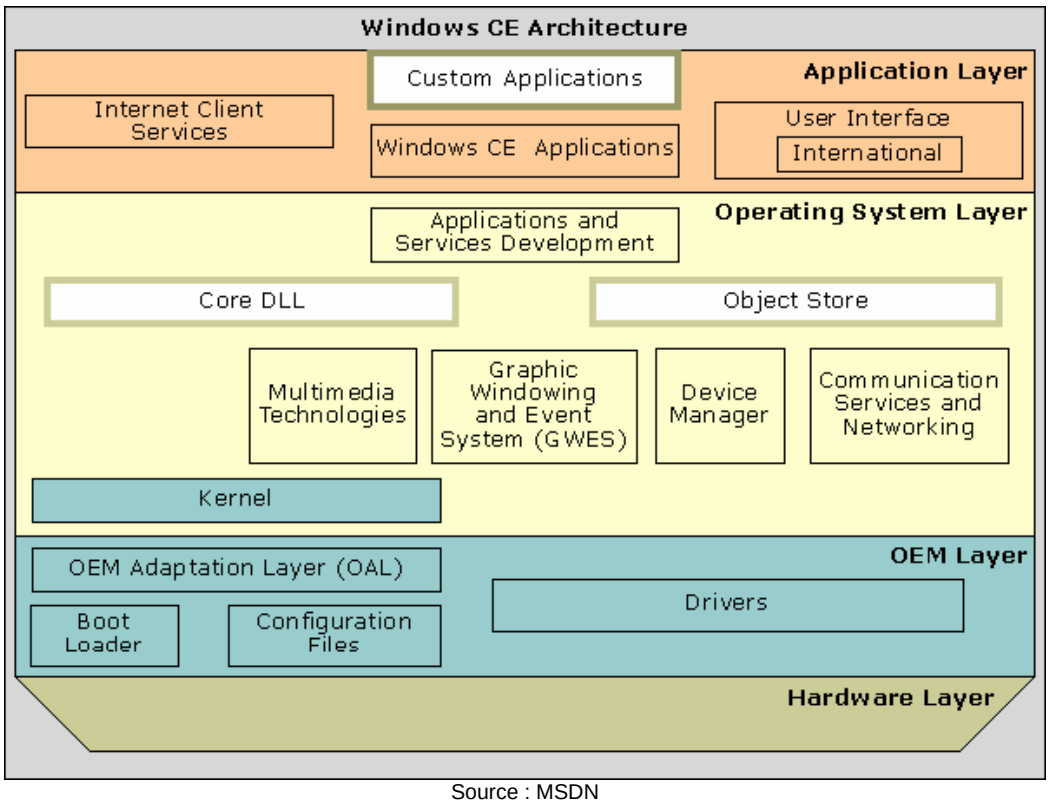
Microsoft va même plus loin en proposant, via un logiciel nommé Platform Builder, de construire un système d'exploitation Windows CE totalement personnalisable en y ajoutant uniquement les composants logiciels requis. Cela permet à chaque fabriquant de disposer de son propre système d'exploitation limité à ses besoins. Celui-ci paie ensuite une licence de l'ordre de 3 à 5 dollars pour chaque OS vendu sur ses appareils.

Cette grande modularité permet de créer des systèmes dont la taille varie entre 1 et plusieurs dizaines de Megaoctets. Celui intégré par défaut dans la carte de Toradex pèse entre 6 et 7 Mo selon la version installée.

La compréhension de l'architecture qui compose le système d'exploitation est très importante. Elle permettra de choisir quelle est la meilleure voie pour développer les applications et modules dont nous aurons besoin.

Le schéma ci-dessous décrit tous les composants qui forment l'architecture d'un système Windows CE. On voit bien que le tout est découpé en quatre blocs principaux :

- La couche matérielle, qui représente la carte électronique et tous ses composants
- La couche OEM, qui représente l'interface entre le matériel et le cœur du système d'exploitation
- La couche du système d'exploitation, comprenant le noyau du système et différents composants
- La couche d'application, qui représente les applications mises à disposition de l'utilisateur



Source : MSDN

La couche matérielle et celle d'application est claire et ne demande pas d'approfondissement pour le moment. Lors du développement logiciel, il sera certainement nécessaire de travailler avec la couche OEM et celle du système d'exploitation. Une étude plus poussée de ces deux éléments est nécessaire.

11.2. La couche OEM

Celle-ci fait clairement l'interface entre le matériel et le système d'exploitation. Elle est en général à la charge du développeur qui veut utiliser Windows CE. Il existe sur le site MSDN de Microsoft un tutorial complet sur la manière de développer la couche OEM. Cette procédure est longue, complexe et demande une connaissance poussée du fonctionnement des couchées avoisinantes.

11.2.1. Le bootloader

Comme son nom l'indique, cette partie du code est la première à s'exécuter lors du démarrage du système. Elle a pour fonction de :

- initialiser et configurer correctement les registres du processeur
- communiquer via un port UART pour le debug, pour afficher et pour piloter un menu
- permettre le téléchargement d'une nouvelle image du système d'exploitation
- démarrer Windows CE

Comprendre le fonctionnement précis d'un bootloader ne fait pas partie du cahier des charges du projet mais l'aborder permet de mieux comprendre certaines parties et fonctions présentes dans Platform Builder et permettra d'avancer plus rapidement.

Les cartes de Toradex, comme la grande majorité de celles dans le domaine des systèmes embarqués, sont composées d'un processeur (Intel XScale PXA dans notre cas), d'une certaine quantité de mémoire RAM et d'une mémoire FLASH (PDA actuels sur le marché : ~64Mo de FLASH, ~128Mo de RAM). Ces éléments sont connectés sur le bus mémoire du processeur et leur contenu est atteignable via une adresse dite adresse physique.

Il faut savoir que ces processeurs, lors de leur mise sous tension, vont chercher l'instruction présente à l'adresse 0 sur le bus mémoire. Il s'agira donc de s'arranger pour que cette adresse corresponde à une donnée dans la mémoire FLASH, qui ne s'efface donc pas lorsque la tension d'alimentation disparaît, et d'y placer des instructions et plus précisément, le bootloader.

En ce qui nous concerne au niveau du projet, le bootloader devra être utilisé pour charger de nouvelles images du système d'exploitation si nécessaire et pour modifier les états des GPIO au démarrage. Le bootloader des cartes Colibri est livré en tant que fichier binaire par Toradex, aucun source n'est disponible (ce qui est normal car il consiste une des plus-values des produits Colibri).

11.2.2. l'OAL Adaptation Layer

Cette partie de code (aussi appelée OSAL pour Operating System Abstraction Layer) fait office d'interface entre le noyau de Windows CE et le matériel présent sur la cible. Il initialise le processeur (reconfiguration des GPIO principalement pour les fonctions spéciales, MMU, horloges,…), permet l'utilisation par l'OS de ressources telles que les interruptions, timers, gestion de l'alimentation et initialise le kernel (création des stacks, initialisation mémoire, fonctions de debug,…)

Son développement est celui qui représente le plus de difficultés et de temps lors du portage de WinCE sur une nouvelle cible. Créer l'OAL à partir de zéro demande énormément de temps et de ressources, c'est pourquoi il est bien plus facile de se baser sur ceux déjà disponibles dans Platform Builder et de les modifier. C'est ce qu'a fait Toradex, en se basant sur un OAL pour Intel Xscale PXA270 déjà mis à disposition. Lors de la compilation du système d'exploitation, le code de l'OAL est intégré dans l'exécutable du noyau de Windows CE.

11.2.3. Les pilotes

Peut-être la partie la plus intéressante et celle qui nous concerne le plus au niveau du projet, les pilotes constituent une partie vitale du système d'exploitation. Ils font l'interface entre un composant matériel (p. ex. une carte graphique) et les logiciels qui veulent l'utiliser. En général, ce matériel est atteignable au travers de registres qui peuvent être écrits ou lus. L'objectif d'un driver est d'offrir aux programmeurs un accès à ces registres sous la forme d'une interface simple et normalisée.

Il en existe plusieurs types qui dépendent de la fonctionnalité du périphérique. Il est clair que le pilote à développer pour piloter une carte graphique n'aura rien à voir avec celui d'une imprimante. Windows CE a divisé les pilotes en deux catégories principales :

- Les pilotes de flux (stream drivers)
- Les pilotes natifs (native drivers)

La première information importante est qu'un pilote sous Windows CE est obligatoirement un fichier sous la forme d'une bibliothèque de liaison dynamique (DLL). Cette dernière est un fichier binaire qui contient une ou plusieurs fonctions mises à disposition pour d'autres applications.

Un pilote peut être soit chargé manuellement en mémoire grâce aux fonctions LoadDriver() et RegisterDriver() qui sont mises à disposition par la dll principale de Windows CE qui s'apelle coredll.dll. Si l'on veut qu'un driver se charge automatiquement en mémoire lors du démarrage de Windows, il suffit d'enregistrer ce dernier dans la base de registre de Windows CE dans le chemin *HKLM\Drivers\Active*. L'activation manuelle d'un pilote est surtout utilisée pour les périphériques branchables à chaud tels que ceux des ports USB ou SDIO.

Les pilotes de flux

Les pilotes de type « stream drivers » sont pensés pour fonctionner de la manière la plus optimale avec des périphériques de type « producteur-consommateur ». Ils sont particulièrement adaptés aux périphériques comme les ports séries (type UART), bus SDIO, port imprimante, USB, etc.

Leur particularité principale réside au niveau de leur interface logicielle, qui est standard et commune à tous les pilotes de flux. Un pilote de type stream driver est au final une DLL qui met à disposition les fonctions suivantes (ils en existent d'autres, moins utilisées) :

- XXX_Init
- XXX_Close
- XXX_PowerUp
- XXX_PowerDown
- XXX_IOControl
- XXX_Open
- XXX_Read
- XXX_Write
- XXX_Seek

Le préfixe XXX de chaque fonction est remplacé par 3 lettres qui servent d'identifiant unique pour chaque driver. Pour une imprimante, cela sera par exemple LPT et pour un port série COM. Lors du chargement du driver, un chiffre lui aussi assigné, par exemple COM3. Ce nom complet sert de lien entre le pilote et le logiciel qui l'utilise.

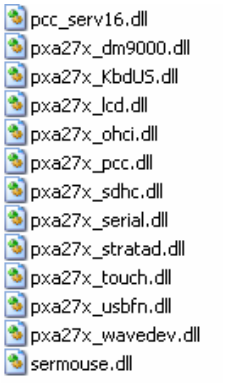
Il faut aussi savoir que les stream drivers sont utilisés comme des fichiers. Par exemple si l'on veut écrire une donnée sur un port série, il faut tout d'abord ouvrir le flux vers le port concerné en utilisant la fonction createFile() (disponible dans la dll *coredll*) puis par exemple en utilisant une autre fonction de cette dll apellée DeviceIOControl() pour envoyer une donnée.

Les pilotes natifs

Tous les pilotes qui mettent à disposition une interface logicielle différente de celle des pilotes de flux sont considérés comme pilotes natifs. Ceux-ci sont créés dans le but d'améliorer les performances lorsque les fonctions mises à disposition ne permettent pas d'effectuer toutes les actions souhaitées.

Les pilotes utilisant cette interface sont par exemple ceux du clavier ou de la souris, ceux de la carte graphique ou encore des pilotes de la batterie.

Remarque : Certains pilotes peuvent même avoir une interface de type stream driver et native driver simultanément.

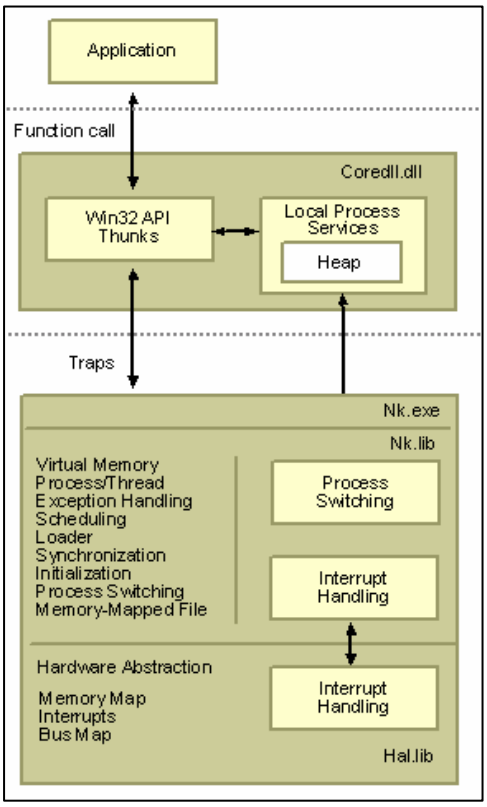


On retrouve ici à gauche tous les drivers qui sont mis à disposition par Toradex (toujours via leur BSP) pour la carte Colibri. On y retrouve le pilote de la carte graphique, celui de la carte réseau, le pilote clavier, souris, USB, infrarouge, le pilote de gestion audio, celui pour la gestion des zones tactiles, les pilotes USB et encore ceux permettant l'accès aux lecteurs de mémoires CompactFlash et SD.

Ils sont bien évidemment déjà compilés sous la forme de dll et Toradex ne donne pas leur source. A noter qu'il est possible d'acheter sur leur site via des crédits d'heures de support les pilotes pour le bus CAN, pilote I2C, SPI, RTC,… avec leur code source.

11.3. La couche du système d'exploitation

Elle représente tout ce que livre Microsoft lors l'on achète le système d'exploitation Windows CE, hormis l'interface utilisateur et les applications. L'élément principal est bien évidemment le noyau autour duquel gravite des modules permettant de faire du multimédia, des accès au système de fichier, l'interface graphique ou encore des librairies.



11.3.1. le noyau

Comme précédemment expliqué, il représente le cœur du système d'exploitation. Il est responsable de la gestion de la mémoire virtuelle, des processus (stacks privés, switching entre processus, synchronisation des tâches) et de la gestion des exceptions. L'OAL fait aussi partie du noyau. Toradex livre dans son BSP pour Platform Builder le fichier *kernel.exe* qui représente le noyau déjà compilé.

11.3.2. Core DLL

Cette librairie de liens dynamiques est la plus importante de Windows CE. Elle met à disposition des applications et des pilotes les fonctions de bases permettant par exemple de charger ou d'activer un pilote, d'accéder à certaines zones mémoires ou encore d'avoir accès à des fonctions du processus lié aux applications. Coredll a un accès privilégié au noyau, comme le montre la figure-ci contre.

11.4. Les librairies de liaisons dynamiques

Comme on a pu le voir jusqu'à maintenant, les librairies de liens dynamiques sont utilisées à de nombreux endroits dans l'architecture Windows CE. Après-tout, un pilote n'est rien d'autre qu'une DLL qui met à disposition des fonctions bien précises. Elles permettent de rendre une application modulaire comme on le verra par la suite, lors du développement du code de l'iPresenter2

De manière générale, une DLL peut contenir du code ou des ressources mises à dispositions pour d'autres applications. Un de ces avantages est celui de ne charger d'une seule fois le code de la dll en mémoire lorsque celle-ci est utilisée par plusieurs applications. Il est aussi par exemple possible de programmer des fonctions en C ou en C++ et des les utiliser dans un programme écrit en langage de plus haut niveau.

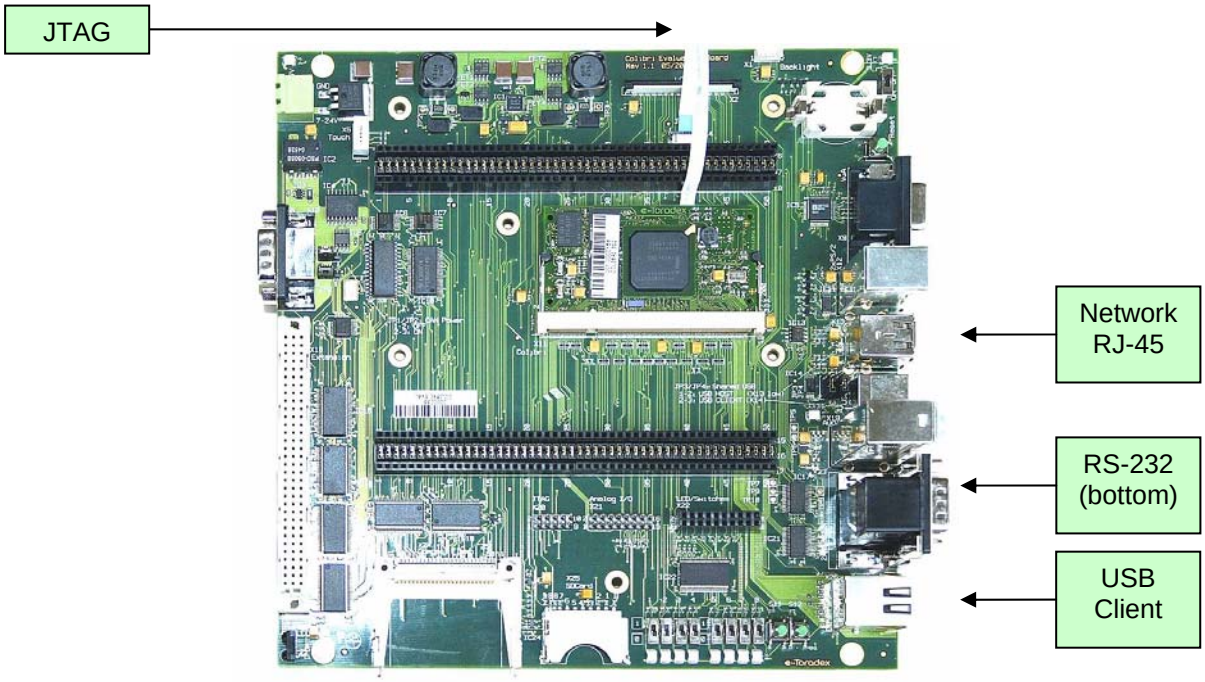
Sources :
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnce30/html/devicedr.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcecoreos5/html/wce50conKernel.asp
http://www.laboratoire-microsoft.org/articles/web/windows_ce_driver/
http://www.laboratoire-microsoft.org/articles/web/windows_ce/

12. Environnement de programmation

Maintenant que l'architecture Windows CE est plus claire, il va falloir écrire, tester et transférer de nouveaux logiciels et parties de code sur la carte. On va étudier les moyens possibles pour le faire.

12.1. La carte de développement Colibri

Comme déjà décrit au chapitre 4.2 au début de ce rapport, une carte de démonstration est livrée avec les modules Colibri. Elles permettent en premier lieu l'utilisation de toutes les ressources de ces derniers mais aussi à paramétrer, programmer et debugger le logiciel présent dans la mémoire. Voici une brève description de toutes les interfaces disponibles pour cet usage, et leur utilité.



JTAG
Cette connexion directe sur le module Colibri permet d'accéder via JTAG à la mémoire FLASH en particulier pour réécrire le contenu du bootloader ou du système d'exploitation en cas de problème.

RS-232 (bottom)
Le port série inférieur de la carte d'évaluation est connecté directement au module Colibri au travers d'une puce d'adaptation de niveaux MAX232. Il est utilisé lors du démarrage du Colibri pour naviguer et configurer le bootloader.

RJ-45 (network)
Le port réseau est connecté directement au module Colibri, qui est équipé d'une interface réseau. Connecté au PC et avec un adressage correctement configuré, il est possible d'envoyer une nouvelle image binaire du système d'exploitation Windows CE sur le Colibri par ce biais.

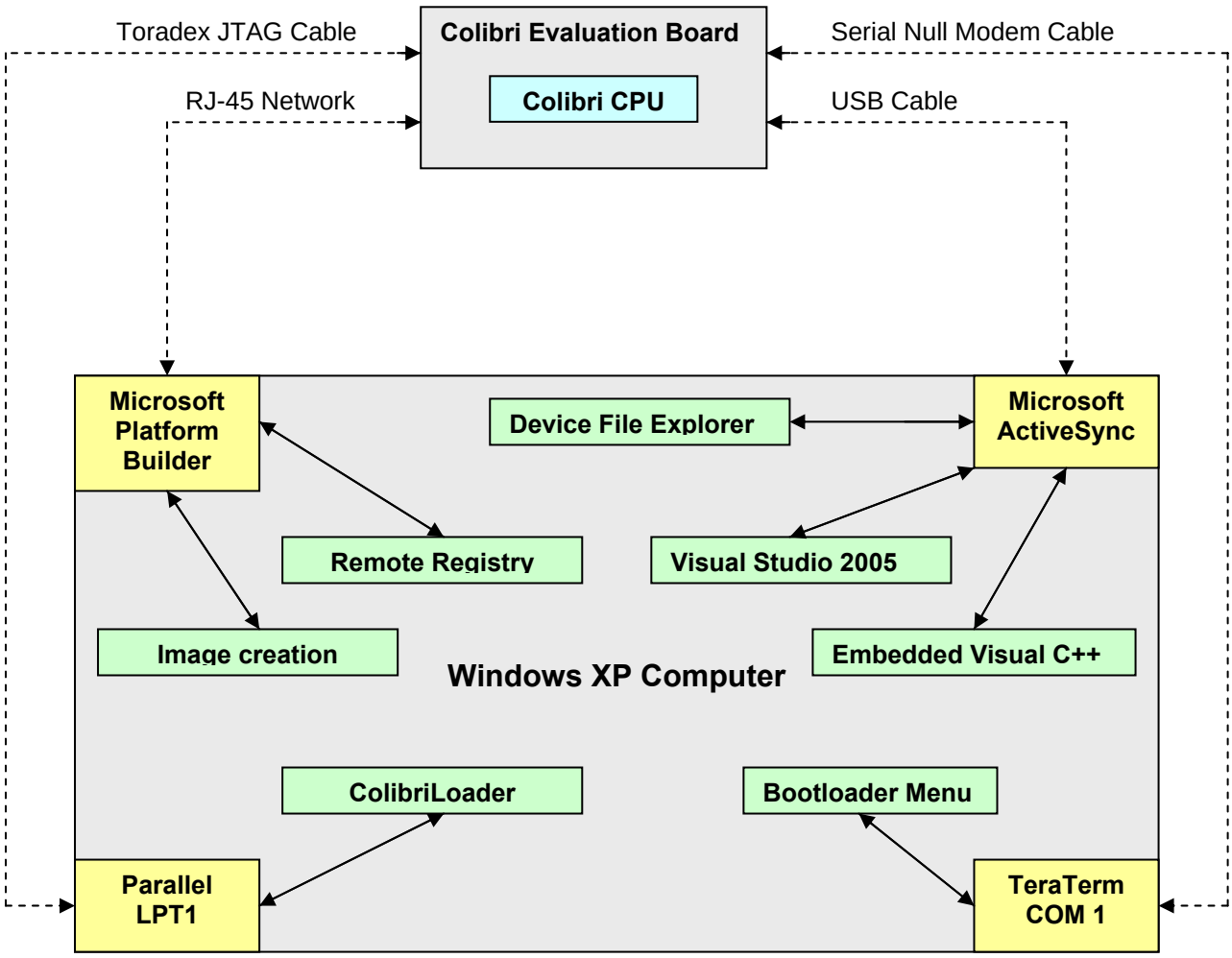
USB Client
Ce port USB client est le même que celui présent dans tous les PDA du marché ainsi que dans les téléphones du type SmartPhone et PocketPC. Il permet une liaison simple et directement reconnue par un ordinateur, au travers du logiciel ActiveSync. Ce port est utilisé pour atteindre la mémoire de l'appareil, pour transférer des données, pour éditer le registre à distance, pour programmer de nouvelles applications, etc..

12.2. Description de l'environnement

Nos besoins au niveau de la communication avec la carte sont grands. Il doit être possible d'utiliser, de modifier et de remplacer tout ce qui est dans la mémoire FLASH du Colibri. Il nous faut donc mettre en place toutes les interfaces décrites ci-dessus.

Ces interfaces seront connectées à un ordinateur qui tourne sous Microsoft Windows XP, installé avec l'image de base laboratoire utilisée à la HES-So.

Voici un schéma logique expliquant l'environnement que nous allons utiliser :



Sur l'ordinateur doivent être installés nombre d'outils permettant la communication avec la carte de démonstration. Le chapitre suivant décrit toutes les étapes permettant de retrouver l'environnement du schéma ci-dessus.

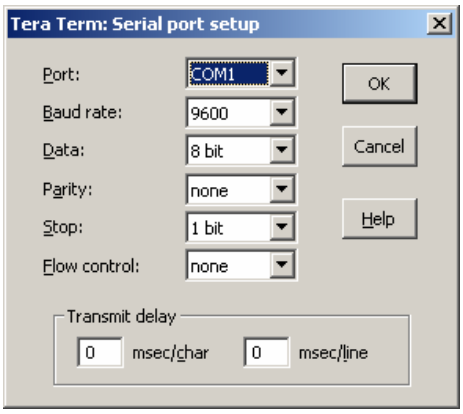
12.3. Installation des logiciels

12.3.1. Teraterm sur port série

Le premier programme à tourner sur le processeur au démarrage de la carte Colibri est le bootloader. Il permet, via un menu, d'effectuer des tâches de base comme la restauration des données du registre, l'écriture d'une nouvelle image soit dans la mémoire RAM soit dans la FLASH ou encore la configuration de l'interface réseau. Il configure aussi le mode et l'état par défaut des entrées/sorties (GPIO) du Colibri.

De plus, c'est lui qui, au démarrage, est responsable de copier l'image Windows CE en RAM et de rediriger le pointeur de programme sur celle-ci.

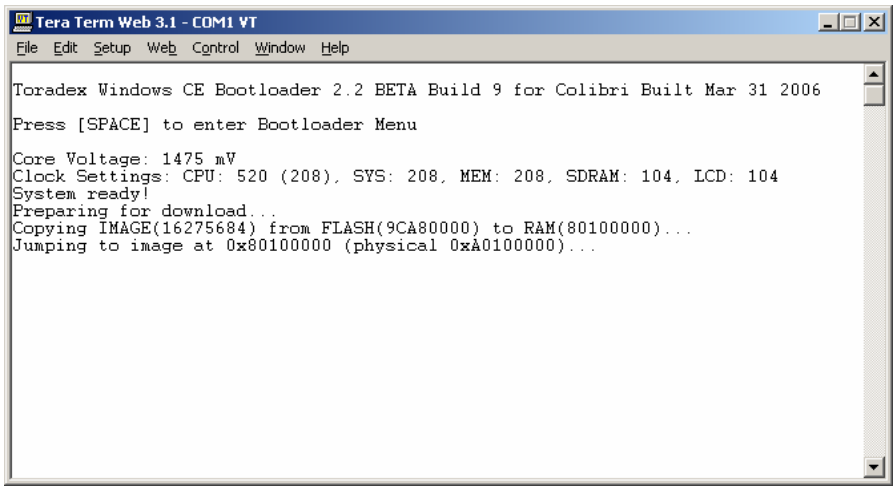
L'interface utilisateur de ce menu est atteignable via le port série inférieur de la carte de démonstration. Les caractères du menu sont envoyés via ce port et il est possible de le piloter avec les touches clavier.



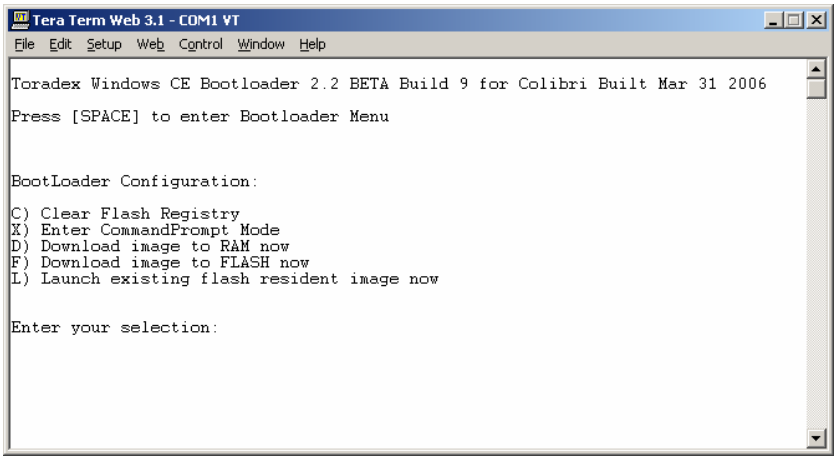
Sur l'image standard des salles de laboratoire se trouve un utilitaire nommé TeraTerm. C'est un simple terminal qui peut recevoir et envoyer des données sur le port série de l'ordinateur. (l'utilitaire HyperTerminal intégré à Windows XP fonctionne mal et n'a donc pas été utilisé.)

Après avoir démarré TeraTerm (version Pro Web 3.1.3), il faut configurer le port série. Aller dans le menu Setup ➔ Serial Port et entrer les informations ci-contre. Appuyer sur OK pour enregistrer la configuration.

La connexion entre l'ordinateur et la carte de démonstration se fait par un câble série 9 pôles de type Null-Modem (DB-9). Veillez à bien connecter le câble sur le bon port COM de l'ordinateur (celui défini dans la configuration du port série de TeraTerm). Si tout est configuré correctement, le menu suivant doit s'afficher à la mise sous tension du Colibri :



Si, lors de la mise sous tension, l'on appuie sur la touche espace, le menu suivant s'affiche sur la fenêtre du terminal et permet de configurer les éléments décrits plus haut :



- Si l'on appuie sur la touche C, toutes les modifications effectuées dans le registre de l'image Windows CE sont effacées et la configuration d'origine est restaurée.
- Si l'on appuie sur la touche X, un mini interpréteur de commande (shell) apparaît ; il permet de configurer l'adresse IP ainsi que le masque et l'interface réseau, ou encore de définir l'état des entrées/sorties (GPIO) de la carte au démarrage. Pour revenir au menu principal, taper « exit » puis enter.
- Si l'on appuie sur la touche D, le bootloader va attendre qu'une image lui soit envoyée via l'interface réseau et la transférer dans la RAM de la carte. Cela permet de travailler avec des images de test (debug) qui seront gardées en mémoire seulement jusqu'à un reset ou une mise hors tension de la carte.
- Si l'on appuie sur la touche F, le bootloader va attendre qu'une image lui soit envoyée via l'interface réseau et la transférer dans la FLASH de la carte. Cela permet d'enregistrer une image fonctionnelle qui sera utilisée à tous les démarrages.
- Si l'on appuie sur la touche L, le bootloader copie le contenu de l'image depuis la FLASH dans la RAM et déplace le pointeur de programme sur cette image.

Il est maintenant possible d'utiliser le menu du bootloader pour configurer certains éléments de notre application. Cette étape sera décrite plus loin.

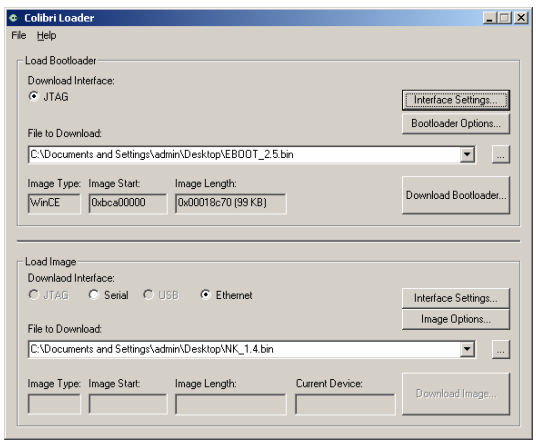
12.3.2. Colibriloader

12.3.2.1. sur le port parallèle

Lorsqu'une image défectueuse est transférée dans la mémoire RAM, il suffit d'appuyer sur le bouton reset de la carte de démonstration pour que l'image d'origine en FLASH soit rechargée. Si l'image de la FLASH est endommagée, il est encore possible de la réécrire via le bootloader.

Si le bootloader lui-même est endommagé, le seul moyen de le récupérer et de le réécrire est de passer via l'interface JTAG de la carte. Un câble constitué d'une fiche DB-25 connectée sur le port parallèle d'un côté et d'un connecteur pour câble plat 10 pôles de l'autre doit être réalisé. Le plan de câblage est le suivant :

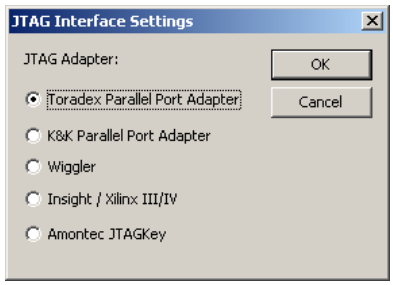
```
X13                                 Parallel Port
(3V3) 2  1 (3V3)                          (do not connect 3V3! Only needed when using other JTAG adapters)
(GND) 4  3 (nTRST) -------------------- 5     (Data 3)
(GND) 6  5 (TDI) ---------------------- 3     (Data 1)
(GND) 8  7 (TMS) ---------------------- 4     (Data 2)
(GND) 10 9 (TCK) ---------------------- 2     (Data 0)
(GND) 12 11 (NC)
(GND) 14 13 (TDO) --------------------- 11    (Busy)
(GND) 16 15 (nSysReset)                   (only needed for debugging, not used)
(GND) 18 17 (NC)
(GND) 20 19 (NC)
                                    18-25 (GND, connect all GND of X13)
```



Sur l'ordinateur, il faut lancer un logiciel créé par Toradex, ColibriLoader, qui est capable d'envoyer le fichier image du bootloader ou de Windows CE sur le port parallèle. Ce programme peut être trouvé à l'adresse suivante : http://toradex.ch/colibri_downloads/WinCE/ColibriLoader/. Il suffit de décompresser le contenu du fichier .zip et de lancer l'exécutable *ColibriLoader.exe*.

La partie du haut permet de télécharger le bootloader et celle du bas une image Windows CE. Ces deux fichiers sont aussi disponibles sur le site de Toradex à l'adresse suivante : http://toradex.ch/colibri_downloads/WinCE/Images/.
A noter qu'il est nécessaire de toujours prendre un

bootloader compatible avec l'image en flash. La description de ces compatibilités est faite dans le fichier Bootloader_ChangeLog.txt qui se situe dans le même dossier.



Avant de lancer le téléchargement d'un bootloader, il faut préciser au programme quelle interface utiliser pour le transfert. Cliquer sur le bouton « Interface settings » puis choisir « Toradex Parallel Port Adapter » dans la liste puis OK. Mettre la carte de démonstration sous tension puis cliquer sur « Download Bootloader… ». Le transfert prend environ 30 secondes.

L'autre utilité de ce programme est de restaurer l'image d'origine de la carte Toradex. La première étape consiste à télécharger l'image (lien ci-dessus) en veillant bien à en prendre une compatible avec le bootloader actuel de la carte. Ensuite, dans la partie basse du programme, choisir l'interface Ethernet, puis cliquer sur « Download Image… ». Puis il faut mettre la carte de démonstration sous tension en appuyant sur la touche espace du clavier (le terminal doit être ouvert et le câble série null modem connecté). Dans le menu, taper « F » pour télécharger l'image dans la FLASH du Colibri. Le transfert prend moins d'une minute. Une fois terminé, appuyer sur le bouton reset de la carte et fermer le programme.

12.3.2.2. avec JTAG Key

La HES-So est propriétaire d'un certain nombre d'appareils de la société Amontec nommés « JTAG Key ». Ces modules se branchent sur un port USB et mettent à disposition une interface de type JTAG. Il est possible de les connecter directement la carte de démonstration de Toradex via le connecteur 20 broches X20 au lieu du câble sur le port parallèle créé ci-dessus. Il faudra aussi choisir l'option « Amontec JTAGKey » dans les options d'interface du programme. Les deux connectiques ont été testées et fonctionnent correctement.

12.3.3. Microsoft Platform Builder

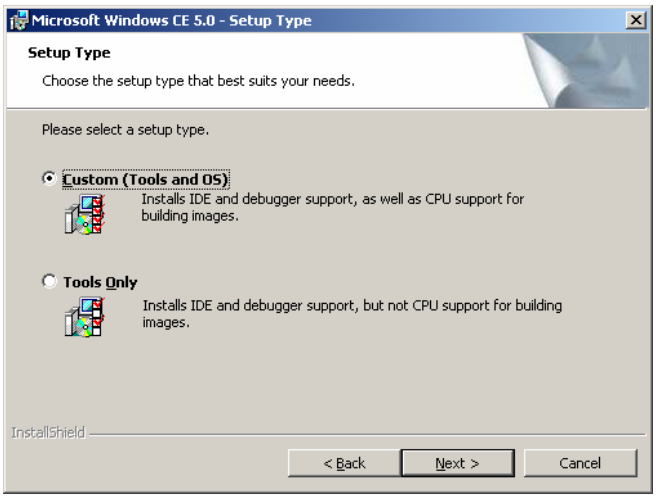
Un des objectifs de ce travail de diplôme est d'être capable de compiler une image du système d'exploitation Windows CE 5 avec des composants spécifiques. Microsoft commercialise Windows CE 5.0 sous la forme d'une application nommée Microsoft Platform Builder. Elle permet, par le biais de modules, pilotes et applications que l'on peut rajouter à un projet, de compiler une image binaire exécutable et personnalisée.

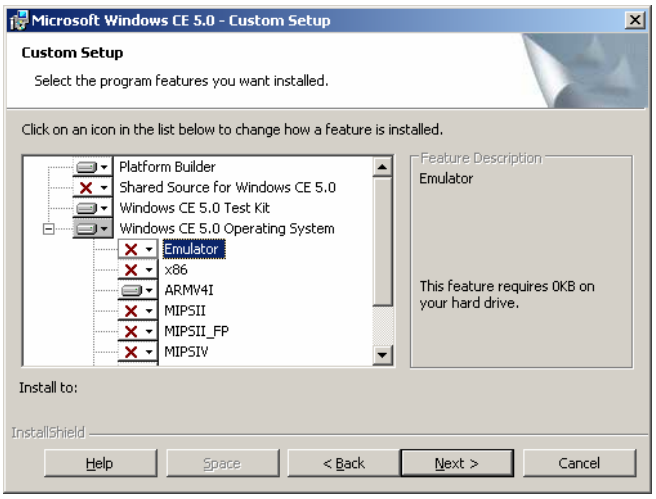
Platform Builder met aussi à disposition des utilitaires permettant par exemple de modifier à distance le registre de la cible Windows CE, de voir les processus en cours, l'état du heap, etc.

Le transfert des images compilées se fait par le biais de la connexion réseau (RJ-45), tout comme avec ColibriLoader. Ce logiciel est bien sûr payant mais il existe une version de démonstration, valable durant 120 jours, qui nous suffira amplement. La version de démonstration est disponible à l'adresse suivante :

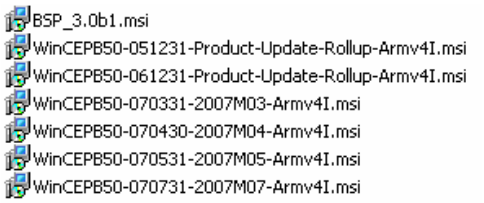
http://www.microsoft.com/downloads/details.aspx?FamilyID=486E8250-D311-4F67-9FB3-23E8B8944F3E&displaylang=en

L'installation ne pose pas de problèmes particuliers, il faut juste suivre les étapes, choisir une installation personnalisée et cocher les paramètres suivants :





Uniquement les sources pour des cibles ARMV4I (cœur des processeurs PXA) seront téléchargées et cela réduira considérablement le temps et la taille lors de l'installation. Le reste de l'installation se déroule sans encombre, aucune action spécifique n'est nécessaire.



Les fichiers ci-contre représentent tous les autres éléments à installer après Platform Builder. Tous ceux commençant par « WinCEPB50-» sont des mises à jour. Ils sont téléchargeables depuis le site de Microsoft http://msdn2.microsoft.com/en-us/embedded/aa731256.aspx.

Pour pouvoir créer une image qui fonctionne avec notre carte, il va falloir intégrer les différents pilotes et les autres parties logicielles mises à disposition par Toradex lors de la compilation. Pour ce faire, Toradex met à disposition un BSP (Board Support Package) compatible avec Platform Builder pour le Colibri. Il peut être téléchargé à l'adresse suivante : http://toradex.ch/colibri_downloads/WinCE/BSP/. Choisir si possible la dernière version (version utilisée pour le projet : v.3.0b1). L'installation est à nouveau simple et ne pose aucun problème. Suivre simplement les instructions et veillez à bien fermer Platform Builder avant de lancer l'installation du BSP.

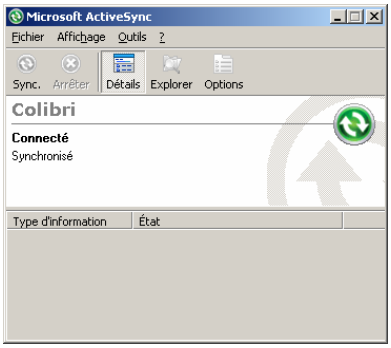
Platform Builder est maintenant opérationnel et il est possible de créer des images avec les pilotes adéquats, transférable sur le Colibri.

12.3.4. Microsoft ActiveSync

Ce logiciel, bien connu dans le monde des assistants personnels, permet principalement de synchroniser les informations de l'ordinateur et du PDA. Nous n'allons pas l'utiliser pout cette fonctionnalité-là. Le premier utilitaire intéressant proposé dans ActiveSync est l'explorateur de fichiers. Il est possible de voir, copier, créer et transférer des fichiers et des dossiers dans toute la mémoire de la carte Colibri.

Une fonctionnalité moins connue d'ActiveSync est celle de créer une interface entre les logiciels de programmation et le PDA. En effet, tous les logiciels permettant de créer et de debugger des logiciels pour applications mobiles Windows CE passent par ActiveSync pour le transfert et la communication avec le PDA.

L'installation est, elle aussi, simple et ne nécessite aucune manipulation particulière. Suivre les instructions recommandées lors de l'installation.



Ce logiciel est gratuit et téléchargeable ici : http://www.microsoft.com/downloads/details.aspx?FamilyID=9e641c34-6f7f-404d-a04b-dc09f8141141&displaylang=fr. (version utilisée pour le projet : 4.5)

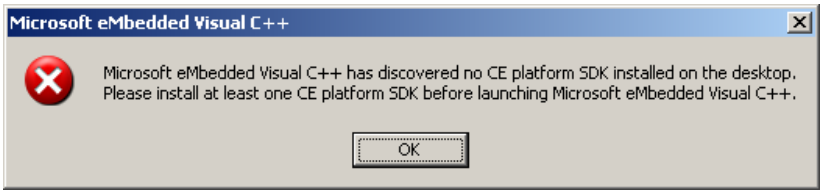
12.3.5. Microsoft embedded Visual C++ 4.0

Pour développer leurs applications Windows CE, Toradex utilise un outil de compilation et de debug nommé Embedded Visual C++. C'est une version allégée et gratuite de Visual C++ contenu dans la suite Visual Studio. Elle permet de compiler des applications simples ou graphiques orientées objet (MFC) en C++ pour Windows CE 5.0.

Le programme peut être téléchargé à l'adresse suivante :
http://www.microsoft.com/downloads/details.aspx?FamilyId=1DACDB3D-50D1-41B2-A107-FA75AE960856&displaylang=en (version utilisée pour le projet : 4.0)

Tel quel, le programme ne fonctionne pas correctement sous Windows XP, il plante lors de la création d'un projet ou de la compilation. Il est nécessaire d'installer le Service Pack 4, que l'on peut trouver à l'adresse suivante :
http://www.microsoft.com/downloads/details.aspx?FamilyID=4A4ED1F4-91D3-4DBE-986E-A812984318E5&displaylang=en

Installer en premier le programme. Il n'y a pas de manipulation spcéfique à faire, suivre simplement les instructions et accepter les paramètres par défaut. Même opération pour le service pack 4.

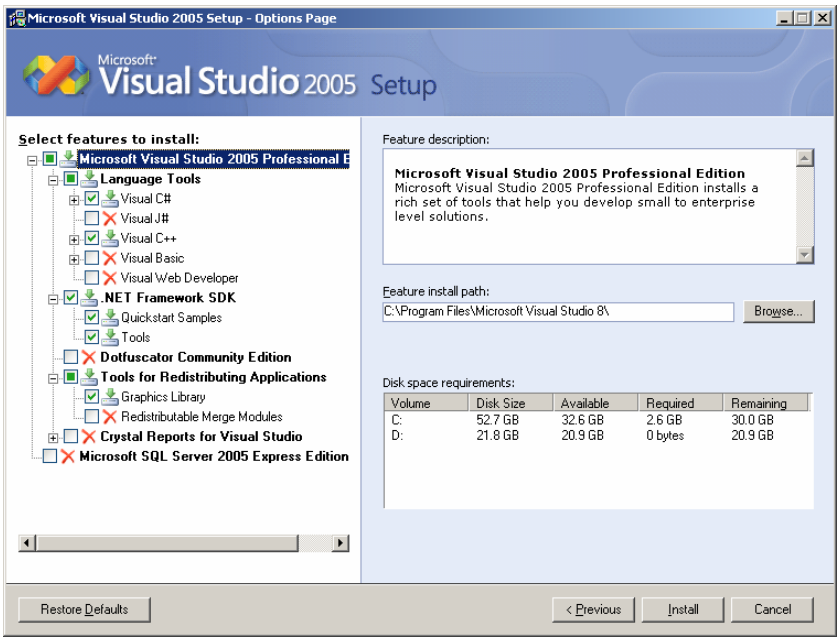


Ce message d'erreur apparaît si l'on essaie de démarrer le programme à cet instant précis. Embedded Visual C++ 4.0 est prévu pour fonctionner uniquement avec un BSP installé, pour limiter l'utilisation du logiciel au développement sur une cible reconnue. Afin de prendre en charge la carte Colibri pour le debug, il est nécessaire d'installer le SDK de Toradex. Il peut être télécharger ici : http://toradex.ch/colibri_downloads/WinCE/SDK/. (version utilisée pour le projet : 1.0). Il faut l'installer en dernier simplement en suivant les instructions et en ayant bien fermé embedded Visual C++.

Le premier environnement de programmation est maintenant opérationnel et permet de coder des applications en C++ pour Windows CE 5.0.

12.3.6. Microsoft Visual Studio 2005

Visual Studio version 2005 permet, en utilisant le Framework .NET, de programmer des applications graphiques très facilement dans les derniers langages de programmation existants tels que C# ou C++. Nous allons donc l'installer. La HES-So possède les CD et la licence pour cette suite Microsoft. La procédure d'installation est, elle-aussi, simple et ne requiert que le choix des composants suivants :



Les langages Visual Basic et Java n'ont pas été installés car ils ne seront pas utilisés dans la suite du projet.

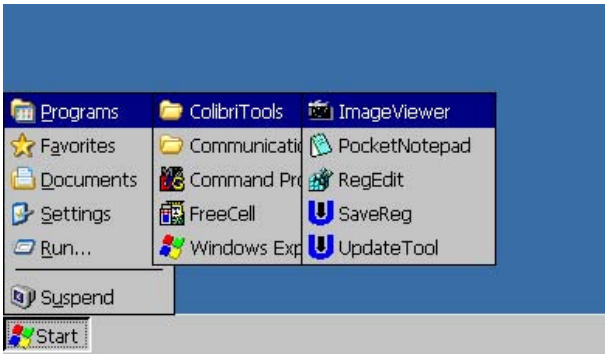
13. Lire un fichier Powerpoint

13.1. Analyse de l'image de base Colibri

Lorsque l'on insère le module dans la carte de démonstration et que l'on démarre le tout, voici l'image affichée sur la sortie écran :

Il s'agit du bureau Windows CE. Il diffère de celui des assistants personnels PDA de par la présence d'une barre de menu et d'icônes sur le bureau. Si l'on connecte une souris soit sur les ports PS/2 soit sur USB, il est possible d'utiliser Windows CE de manière assez similaire à un ordinateur de bureau Windows 9x/2000/XP.

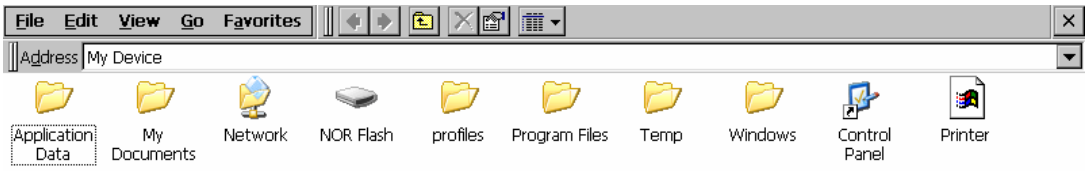
Le bouton Start regroupe des raccourcis vers les applications installées sur le système.



On y retrouve les applications suivantes :

ImageViewer : Logiciel de visualisation d'images BMP ou JPEG
PocketNotepad : Simple éditeur de texte
RegEdit : Editeur de registre simplifié
SaveReg : Outil permettant de sauver les modifications faites au registre dans la mémoire FLASH
UpdateTool : Utilitaire permettant la mise à jour des images Windows CE et du bootloader de la FLASH depuis Windows
Command Prompt : Invite de commande
Terminal : Terminal pour le port série
FreeCell : Jeux de cartes
Windows Explorer : Explorateur Windows allégé

L'icône *My Device* lance une version allégée de l'explorateur Windows et permet d'accéder au contenu de la mémoire RAM et FLASH de l'appareil.



Les dossiers systèmes en jaune sont ceux utilisés par le système d'exploitation. Le dossier « NOR FLASH » permet d'accéder à une partie de la mémoire flash du Colibri et d'y stocker des informations, une sorte de disque dur accessible en lecture/écriture. Sur le PXA270, la taille de cet emplacement mémoire est de 11Mo environ. Cela reste insuffisant pour y stocker de grosses présentations Powerpoint. De plus, on utilisera sûrement une partie de cette mémoire pour y loger l'application de gestion et des dll. Pour ce qui est du PXA320, la place en FLASH est largement suffisante, un peu moins de 1 Go.

Une recherche détaillée dans tout le contenu du système permet d'avancer qu'il n'y a pas d'application permettant de lire les fichiers Powerpoint installée sur le système.

Toutes les modifications effectuées dans les fichiers systèmes, le registre ou dans la mémoire autre que celle du dossier NOR Flash est perdue lors d'un reset de la carte. Ces modifications sont en effet effectuées en RAM et non sauvées automatiquement. Pour ce qui est des modifications du registre, il est possible de sauver le registre dans la mémoire FLASH grâce à l'utilitaire SaveReg du menu démarrer.

13.2. Analyse des différentes possibilités

Rien n'est présent d'origine sur le Colibri pour nous permettre de lire des fichiers Powerpoint. L'étude rapide préalable effectuée au chapitre 2 indique que deux solutions s'offrent à nous :

- **Trouver un moyen de faire tourner Powerpoint Mobile sur la carte de Toradex**

- **Utiliser un logiciel tierce partie capable de lire une présentation et l'installer sur l'image d'origine du Colibri**

La première solution est sans doute la plus efficace et la plus élégante. Powerpoint Mobile est une application de Microsoft lui-même conçue pour tourner sur Windows CE. Elle gère par défaut toutes les versions actuelles des présentations Powerpoint (97, 2000, XP, 2003).

Powerpoint Mobile est livré en standard avec Windows Mobile. Pour rappel, Windows Mobile est un dérivé de Windows CE 5.0 équipé du Framework .NET, de Word, Excel et Powerpoint Mobile, ainsi que quelques autres utilitaires. Partant de ce fait-là, on peut imaginer deux chemins pour faire tourner Powerpoint Mobile sur le Colibri :

- **Extraire l'application et les DLL nécessaires depuis une version de Windows Mobile et la copier sur la carte Colibri (Emplacement NOR Flash)**

- **Recréer de toute part une image de Windows CE intégrant Powerpoint Mobile et copier cette image dans la mémoire FLASH du Colibri**

La première solution a été testée et est traitée en détails ci-après. La deuxième pourrait être intéressante mais demande un gros travail. Le chapitre 16 est dédié essentiellement à la création d'une image pour la carte de Toradex.

13.3. Récupération de l'exécutable

La première solution mise en œuvre a été de récupérer l'exécutable de Microsoft Powerpoint Mobile et de l'intégrer dans l'image d'origine de Toradex. Cette méthode ne demande pas de retravailler en profondeur l'OS et pourrait très bien fonctionner. La première étape est donc de parvenir à trouver les fichiers nécessaires au fonctionnement du logiciel. J'ai essayé de la faire de différentes manières :

- Relier à l'ordinateur un PDA tournant sous Windows Mobile et extraire via ActiveSync les fichiers de la ROM ➔ impossible car les fichiers situés en ROM sont protégés et ne peuvent être extraits
- Relier à l'ordinateur un PDA tournant sous Windows Mobile et extraire la totalité de la ROM via un programme associé (XDATools) puis de recréer le système de fichier en ROM avec DumpRom) ➔ impossible de lancer les fichiers extraits car ils n'ont pas été compilés pour une cible ARMV4I (du au PDA source utilisé)
- Recréer une image du système d'exploitation Windows CE avec Platform Builder et récupérer le contenu de la compilation avant la création du fichier image. ➔ Cette procédure a abouti et est décrite en détail au chapitre ci-dessous.

13.3.1. Création d'une image avec Platform Builder

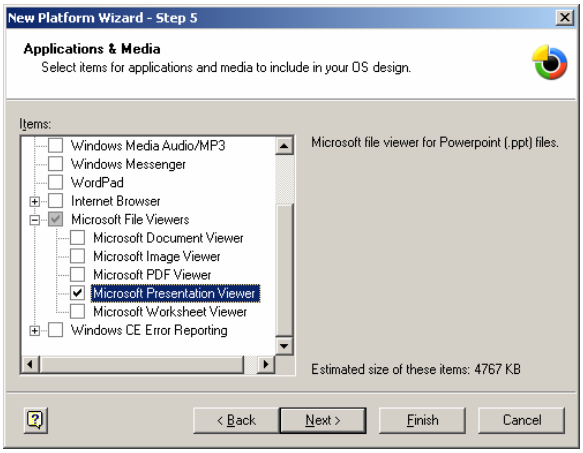
Cette solution peut paraître ambitieuse mais n'est pas si compliquée à mettre en œuvre au vu du fait que l'on doive uniquement récupérer des fichiers compilés et pas modifier des paramètres pointus lors de la compilation.

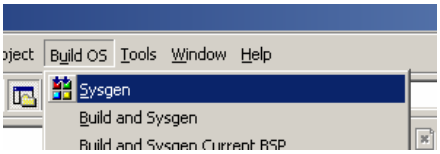
Microsoft Platform Builder 5.0 est un logiciel qui a été conçu pour générer et debugger des images de système d'exploitation Windows CE 5.0. Il permet, de manière simple par l'ajout de composants, de parvenir à un système d'exploitation opérationnel très rapidement.

Les étapes suivantes permettent de générer une image de Windows CE qui contient le lecteur Microsoft Powerpoint Mobile et toutes les librairies nécessaires à son fonctionnement :

Procédure permettant de créer une image Windows CE contenant Powerpoint Mobile

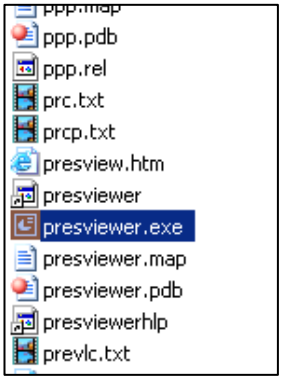
1. Lancer Microsoft Platform Builder 5.0 (procédure d'installation au chapitre 10.5.3)
2. Menu « File », cliquer sur « New Platform »
3. Dans la fenêtre qui s'affiche, cliquer sur « Next »
4. Donner un nom à l'espace de Travail puis « Next »
5. Dans la liste des BSP, cocher « Colibri : ARMV4i » puis « Next »
6. Dans la liste, choisir « Mobile Handled » puis « Next »
7. Dans la liste des « Applications & Media », cocher les éléments suivants :
   < Microsoft File Viewers ➔ Microsoft Presentation Viewer >
   puis cliquer sur « Next »
8. Cliquer sur Finish
9. Dans le menu « Build OS », cliquer sur « Sysgen ». Platform Builder va maintenant compiler tous les modules et créer une image du système d'exploitation correspondant. Cette opération dure environ 10 minutes





13.3.2. Mise en route du lecteur sur l'image de base

Lors de la phase de compilation de l'OS, Platform Builder crée un répertoire (par défaut : « C:\WINCE500\PBWorkspaces\NOM_DU_PROJET\RelDir\ Colibri_ARMV4I_Release » dans lequel on trouve tous les fichiers qui seront copiés dans la mémoire du Colibri. L'exécutable de PowerPoint Mobile se trouve dans cette liste, il est nommé « presviewer.exe ».



Il faut démarrer l'image d'origine sur la carte Colibri et connecter le câble USB entre l'ordinateur et la carte d'essai. Avec ActiveSync, copier le fichier « presviewer.exe » dans la mémoire NOR Flash, puis essayer de l'exécuter en double cliquant depuis le périphérique. Le système indique qu'un composant manque et que presviewer ne peut se lancer.

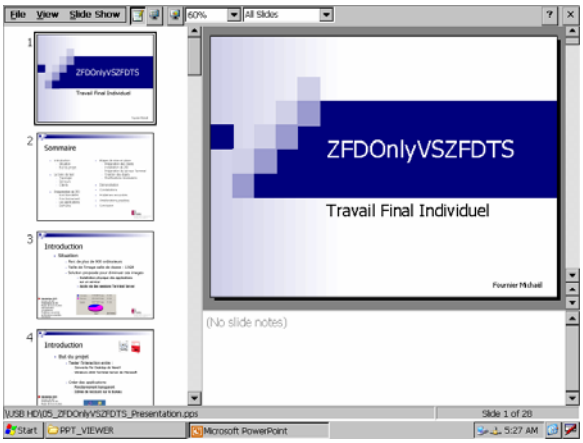
Après plusieurs heures de recherche et d'essais, j'ai réussi à isoler toutes les dll nécessaires au fonctionnement du logiciel. On voit ci-dessous la liste de celles-ci. Le fonctionnement de Powerpoint Mobile a été vérifié afin d'être certain que toutes les dépendances sont réglées et que l'application est 100% fonctionnelle.

Name	Size	Type
autoshape.dll	227 KB	Application Extension
dcdjpeg.dll	16 KB	Application Extension
dcdmz.dll	10 KB	Application Extension
dcdpng.dll	10 KB	Application Extension
image.dll	17 KB	Application Extension
jpeg.dll	56 KB	Application Extension
mfdll.dll	74 KB	Application Extension
png.dll	59 KB	Application Extension
presviewer.exe	357 KB	Application
zlib.dll	37 KB	Application Extension

Différents essais ont été fait avec des fichiers Powerpoint simples, puis avec des animations complexes et des images de haute résolution. La puissance des processeurs PXA270 et PXA320 permet une rapidité d'affichage appréciable. Le contenu multimédia devra bien sûr être limité, les vidéos n'étant par exemple pas prises en charge par le lecteur.

On peut aussi constater que l'application avec toutes ces dépendances occupe une place de 859Ko en mémoire. C'est très peu, et d'autant plus important que la place disponible dans la mémoire « NOR Flash » doit être limitée au maximum lorsque l'on utilise le PXA270.

Il est à remarquer que l'application est capable d'ouvrir les fichiers avec une extension .PPT ou .PPS. Lorsque l'on ouvre un fichier, le logiciel affiche la liste des diapositives et il faut manuellement aller lancer la présentation depuis le menu Slide Show ➔ View Show.



Pendant la présentation, un clic de souris permet de passer à la suivante, alors qu'un clic droit puis un clic gauche sur le bouton « Previous » permet de revenir en arrière dans la présentation. Lorsque tous les slides ont défilé, un écran noir s'affiche et le seul moyen de quitter la présentation est d'effectuer un clic droit puis de cliquer avec le bouton gauche sur End Show.

13.4. Bilan

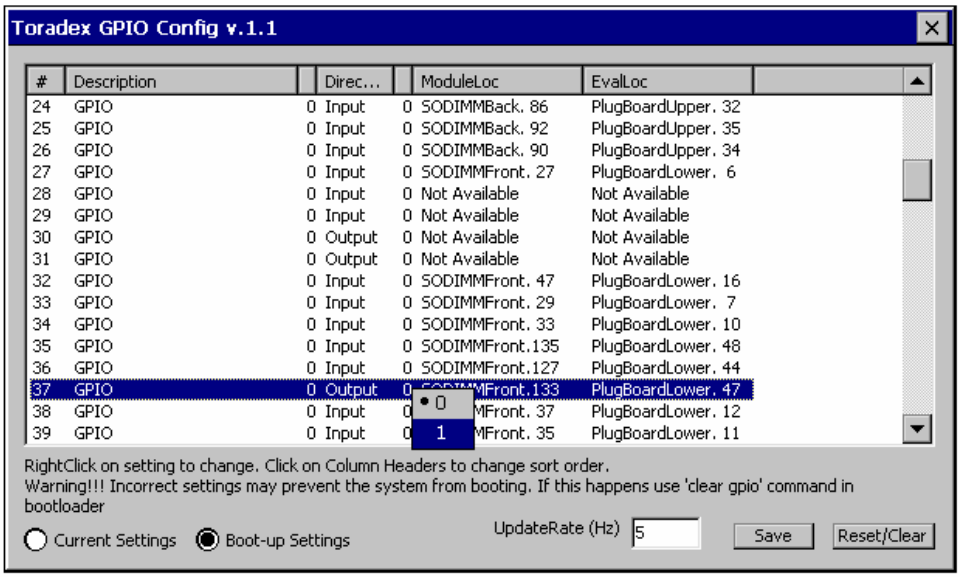
La solution développée permet un bon compromis entre les deux voies imaginées : récupérer l'exécutable et les dll nécessaires pour faire tourner Powerpoint Mobile depuis une image créée et intégrer ceux-ci à l'image d'origine. Le seul inconvénient de celle-ci sera l'intégration du framework .NET qui pourrait être utile à la suite du projet. Ce point sera traité plus tard, lors de la programmation du logiciel de gestion.

Un point important que je n'ai pas traité jusque là est la licence nécessaire pour exploiter le produit Microsoft Powerpoint Mobile avec l'image Windows CE d'origine.

14. Maintien automatique de l'alimentation

Afin de maintenir automatiquement l'alimentation lors de la mise sous tension, il faut parvenir à mettre en sortie et à l'état haut une des GPIO du Colibri au démarrage. Cette opération peut être effectuée par un petit logiciel qui se lance au démarrage et dont la seule tâche est d'écrire une valeur dans le registre de configuration puis de définir une valeur à la GPIO concernée. Cela fonctionne bien, mais il faut maintenir pressé le bouton Power pendant plus de 10 secondes, le temps que requiert Windows pour démarrer avant d'exécuter quoi que ce soit.

Cette solution n'est pas acceptable. Une autre possibilité, plus propre, est d'utiliser un utilitaire disponible à cette adresse : http://toradex.ch/colibri_downloads/WinCE/Colibri_Software/GPIOConfig/ (version utilisée pour le projet : 1.1) qui permet de définir l'état et la configuration de chaque GPIO du Colibri au démarrage. C'est le bootloader qui se chargera d'appliquer ces valeurs aux pins et donc le temps avant que l'opération soit faite est bien diminué. Copier l'exécutable GPIOConfig.exe dans la mémoire FLASH de la carte Colibri et l'exécuter depuis le Windows CE. Cliquer tout d'abord sur « Boot-Up Settings » afin que les paramètres soient appliqués à la mise sous tension. Ensuite, chercher dans la liste le GPIO numéro 37 (correspondant à la pin de maintien de l'alimentation) et cliquer avec le bouton droit de la souris sur Direction ➔ Output puis dans la colonne directement à droite, bouton droit et choisir « 1 ».



Cliquer sur le bouton Save puis fermer l'application avec la croix dans le coin supérieur droit. Depuis maintenant, à chaque démarrage, la GPIO 37 sera mise à 1 à la fin du bootloader. Il suffit de maintenir la pression sur le bouton Power environ 1-1.5 secondes pour allumer le périphérique.

15. Architecture de l'application à développer

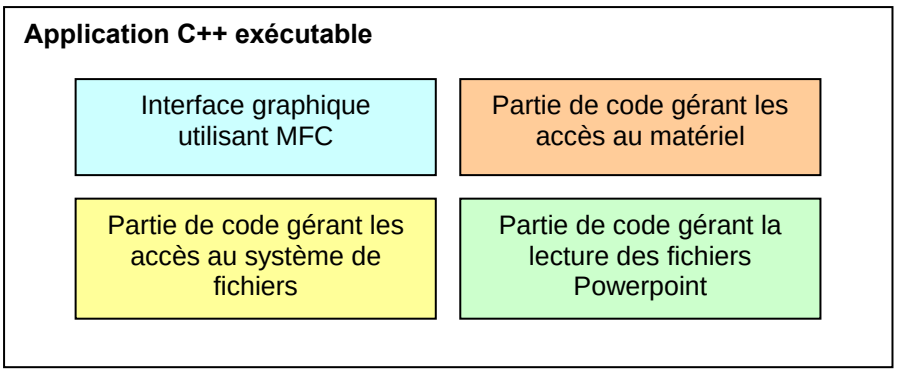
15.1. Introduction

L'application à développer étant conséquente et devant effectuer de nombreuses tâches de nature assez différentes, il est important d'analyser les différentes voies, langages et structures qu'il est possible d'utiliser. L'environnement de programmation mis en place au chapitre 10.4.3 nous permet de créer des applications qui tournent sous Windows CE de plusieurs manières. On peut par exemple imaginer programmer toute l'application avec embedded Visual C++. Il serait aussi possible d'utiliser Visual Studio 2005 avec une liberté au niveau du choix du langage C#, VB, ou C++. Vu qu'une interface graphique est nécessaire, la programmation avec embedded Visual C++ requiert l'utilisation de librairies telles que MFC.

Une analyse plus approfondie est nécessaire afin de déterminer les avantages et inconvénients de chaque solution.

15.2. Architecture plate en C++ avec MFC

L'idée de cette architecture est de regrouper au sein d'une seule application exécutable tous les éléments nécessaires au fonctionnement de l'iPresenter2. La partie interface graphique sera gérée par les classes MFC. Celles-ci ont été créées afin de simplifier le développement de programmes utilisant une interface graphique. La prise en main des classes MFC est assez difficile dans un premier temps ; un bon tutorial existe sur internet au format PDF. Il a été joint aux données du projet dans le sous répertoire *software\doc\mfc* sous la dénomination *iPres2_MFC_tutorial.pdf*.

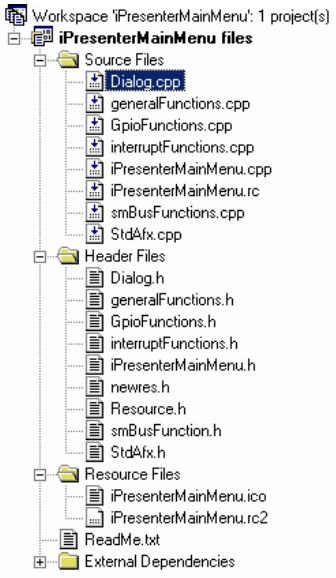


Une autre partie du code sera responsable de gérer les accès au matériel, principalement la lecture de l'état des boutons, contrôle de la LED et communication I2C.

La troisième partie de l'application sera responsable de la gestion du système de fichiers, détection du branchement de clés USB, listage du contenu de répertoire, copie et suppression de fichiers de la mémoire interne et recherche de fichiers de type Powerpoint.

La dernière partie du logiciel s'occupera de démarrer et de piloter le logiciel de lecture des fichiers Powerpoint presviewer.exe. Il faudra trouver un moyen de faire passer au slide suivant/précédent la présentation depuis l'application.

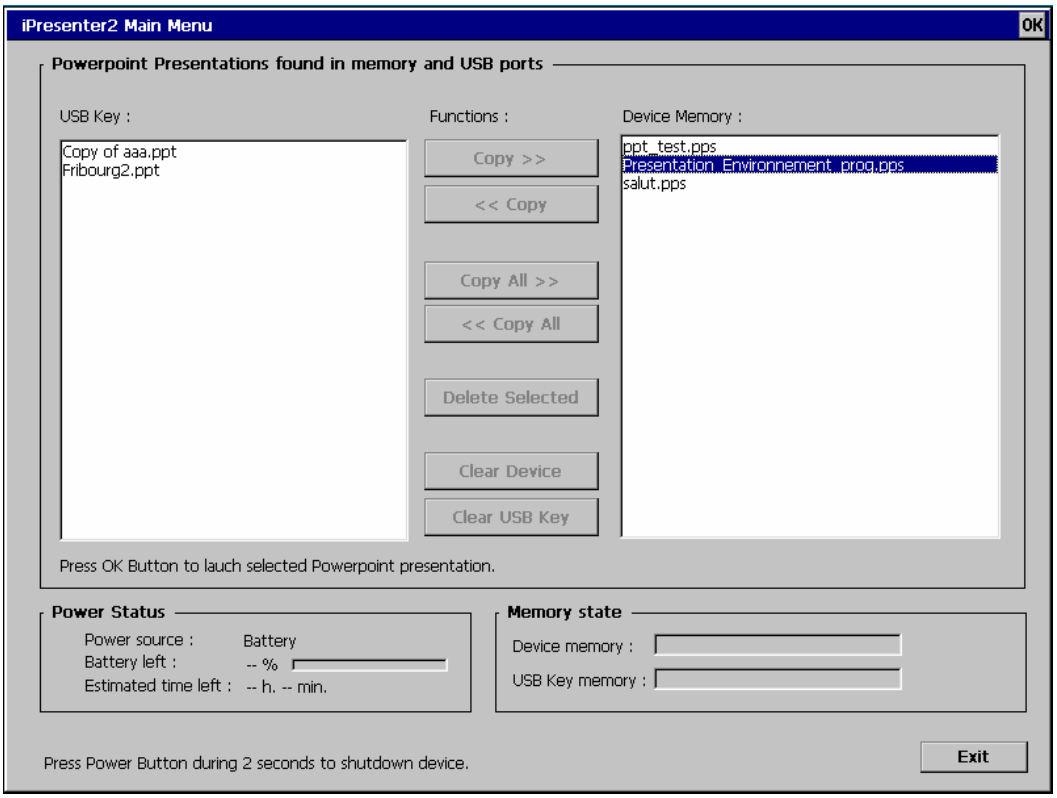
15.2.1. Réalisation de l'application



J'ai commencé à coder l'application telle que définie ci-dessus. La séparation des différentes parties du programme est réalisée en créant plusieurs fichiers source dans l'espace de travail, ceux-ci contenant des fonctions utilisables par le programme principal (image ci-contre). On y retrouve la gestion des boutons et des LEDs, l'utilisation des interruptions matérielles, la gestion de fichiers et du programme Powerpoint ainsi que la commande d'avancement / recul des diapositives.

Au cours de la programmation, j'ai eu de nombreux problèmes au niveau de l'interface graphique. En effet, les limitations liées à MFC ne permettent pas de créer et surtout de personnaliser les éléments graphiques tels que boutons ou zones de texte. La gestion des polices de caractère est restreinte et inutilisable dans un programme plein écran fait pour être lu facilement (grandes polices, choix des couleurs, etc.) Les classes MFC pour Windows CE sont plutôt faites pour créer des applications graphiques sur des PDA avec écran de taille limitée, avec un stylet et sans animation ni interaction complexes avec l'utilisateur.

Voici un aperçu de l'interface graphique créée :



15.2.2. Tests et conclusion

L'avantage de ce type de code est la rapidité d'exécution, le tout tournant en C++ natif et se trouvant dans le même exécutable. Bien qu'acceptable, l'interface n'est pas satisfaisante, de part sa rigidité et une présentation limitée à des boutons de type Windows ainsi que des listes non personnalisables.

Le codage compliqué des classes MFC, les limitations rencontrées dans la personnalisation des objets graphiques ainsi que l'impossibilité d'adapter ceux-ci à une utilisation sur un écran de grande taille constituent les défauts de cette architecture. Il faut envisager une autre solution.

Remarque : Le codage de cette application n'a pas été mené à son terme. C'est pour cela qu'aucune explication concernant le code source n'est présente dans le rapport. Il a cependant été ajouté sur le CD-ROM afin de montrer les techniques utilisées et de voir celles reprises dans l'application finale. (*5_Software\2_Developpment\iPresenterMainMenu_old*)

15.3. Architecture organisée avec librairies

Le framework .NET

Le framework .NET est un composant logiciel qui peut être installé sur les systèmes d'exploitation Windows. Il propose une architecture unifiée permettant de faciliter le travail des développeurs. Il existe une version nommée Compact Framework installable sur les systèmes Windows CE. Il est composé de deux blocs principaux : le CLR et les bibliothèques de classes.

Le CLR (Common Langage Runtime) correspond à la machine virtuelle Java pour les applications .NET. Les applications écrites pour .NET sont transformées en pseudo-code lors de la compilation et ne comportent aucune instruction relative au matériel ou au système d'exploitation. Plusieurs langages sont utilisables : C# (mélange de C++ et de Java), le J# (langage proche mais non compatible avec Java) et VB.NET. Le CLR a pour but de transformer le pseudo-code en instructions machines sur la cible installée.

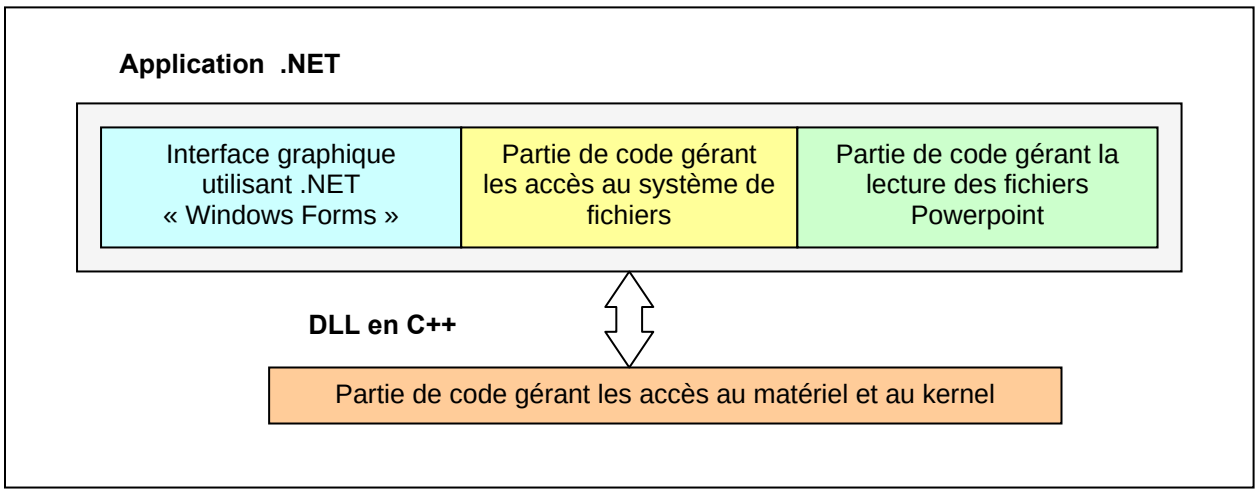
A cela sont ajoutées des bibliothèques de classes permettant la gestion des **chaînes de caractères**, entrées/sorties, **fichiers**, threads, ADO (accès au données), gestion XML et surtout les classes de type **Windows Forms** permettant de créer des fenêtres totalement personnalisables. Les éléments en gras sont ceux qui nous intéressent le plus dans le cadre du projet.

Source : Wikipedia

Architecture du programme

Le grand avantage du code en C++ natif réside d'une part dans la rapidité d'exécution du code et d'autre part dans les accès simplifiés aux fonctions de bas niveau, en particulier celles qui nous touchent comme l'accès aux registres matériels, la simulation des touches clavier/souris, les fonctions du kernel, etc.

Il faut trouver un moyen de garder cet atout tout en étant capable d'utiliser les grands avantages des langages évolués orientés objets de type C#, J# ou VB.NET, créés pour tirer parti de toutes les capacités de la plateforme .NET. L'architecture imaginée est la suivante :



Cette nouvelle architecture permet de tirer parti des avantages du code C++ et du framework .NET. Il aurait été possible de remplacer la librairie de liens dynamiques imaginée par des pilotes. Il suffirait de standardiser les fonctions mises à disposition par celle-ci et de l'intégrer dans Windows CE via la base de registre Windows.

Cela a été jugé inutile de part la complexité que cela amène au projet pour au final un gain nul ou très faible en performance.

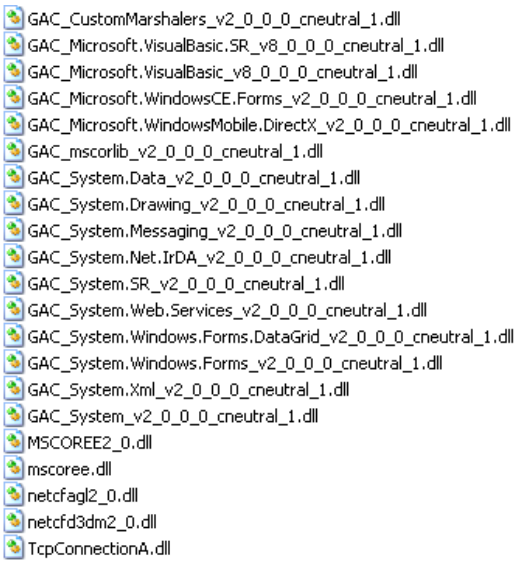
Installation du Compact .NET framework sur l'image Colibri

L'image Colibri d'origine ne comportant pas ce framework, il va falloir trouver un moyen de l'intégrer. Toradex propose un logiciel nommé « .Net Framework 2.0 Flash Installer » permettant d'ajouter le framework directement dans l'image FLASH d'origine. Bien évidemment celui-ci est payant et cher (2 heures de support = environ 330 CHF) ! Une autre solution doit être trouvée. Le Compact Framework dernière version (v2.0 SP1) est gratuit et téléchargeable depuis le site de Microsoft à l'adresse suivante : http://www.microsoft.com/downloads/details.aspx?displaylang=fr&FamilyID=0c1b0a88-59e2-4eba-a70e-4cd851c5fcc4. Il peut être installé sur la carte Colibri via ActiveSync comme n'importe quel logiciel. Seul problème, une fois la carte hors tension ou après un reset, tout est effacé car le programme est installé en mémoire RAM et le système de fichier n'est pas sauvé sur la FLASH.

Deux solutions sont envisageables pour pallier à ce problème :

- **Installer le framework .NET sur le Colibri de manière normale, repérer les fichiers installés en RAM, les copier dans la mémoire de stockage (Emplacement NOR Flash) et au démarrage les restaurer à leur emplacement initial.**
- **Recréer une image complète du système d'exploitation avec le framework installé.**

La première solution a été testée et fonctionne. Comme on peut le voir ci-contre, la liste des fichiers installés par le framework est assez longue. Un dossier sur le CD-ROM en annexe contient tous ces fichiers, car les isoler demande un travail de recherche relativement pénible. (*5_Software\1_Colibri_bin\2_dotNET_FrameworkV20_files*). Tous ces fichiers doivent être copiés dans le répertoire système \windows au démarrage pour que le framework soit considéré comme installé sur le système. Aucune entrée dans la base de registre n'est nécessaire pour faire tourner les applications. La version utilisée est la 2.0 SP1. Leur taille totale en mémoire est de l'ordre de 6 Mo.



Cela peut paraître raisonnable au vu de la taille disponible en FLASH mais il faut d'une part savoir que la place restante devient très faible pour stocker des présentations sur l'appareil (PXA270) et surtout : le temps nécessaire à la copie des fichiers de la FLASH dans le dossier système de Windows au démarrage (avant de lancer l'application .NET) est long (de l'ordre d'une quinzaine de secondes).

S'ajoute à cela le temps de chargement du bootloader et de Windows, ce qui nous ramène à un temps total de chargement depuis la pression sur le bouton Power jusqu'à l'apparition de l'application de l'ordre de 20-25 secondes. C'est long. Il est à remarquer qu'avec le PXA320, ce temps est ramené à 12-15 secondes, de part sa fréquence plus élevée (en particulier au niveau du bus principal, accélérant la copie de la FLASH vers la RAM).

Il est donc acceptable de faire cette copie si l'on travaille avec le PXA320, mais pas sur le PXA270.

Cette copie a lieu lors que l'on place tous les fichiers ci-dessus dans le répertoire « autocopy » de la mémoire FLASH, dans un sous-dossier nommé « windows ». Cela a pour effet de les copier dans le dossier système au démarrage du Colibri. Ainsi, le framework est considéré comme installé.

La solution complexe de la recréation complète d'une image Windows CE devient de plus en plus nécessaire ; en effet au niveau des temps de chargement et de la rapidité d'exécution des applications. Le chapitre suivant est consacré à la création d'une image Windows CE pour le Colibri.

16. Création d'une nouvelle image Windows CE

16.1. Introduction

La création d'une nouvelle image comprenant des fonctionnalités supplémentaires se faisant de plus en plus sentir, j'ai pris du temps pour tester et documenter cette étape. Elle n'est pas vitale au projet mais permet une chose très intéressante : se séparer de l'image de base livrée par Toradex et ajouter au Colibri n'importe quelle fonctionnalité livrée avec le système d'exploitation Windows CE.

De plus, un gain réel aura lieu au niveau des temps de démarrage et de la propreté de l'architecture, aucune copie de fichiers n'étant nécessaire après le démarrage de Windows.

Il est en premier lieu intéressant de lister les nouveaux composants à intégrer dans l'image. Après discussion avec Mr. Pompili, il s'est avéré que de nouveaux logiciels pourraient être intéressants sur l'iPresenter2. Voici la liste complète :

- Microsoft Powerpoint Mobile
- Microsoft Word Mobile
- Microsoft PDF Reader Mobile
- Microsoft Excel Mobile
- Microsoft Image Viewer
- .NET Compact Framework

L'iPresenter2 sera plus universel. Le logiciel de gestion ne permettra que de gérer les présentations Microsoft Powerpoint mais si l'on y connecte un clavier et une souris, il peut devenir un lecteur universel de tous les fichiers cités ci-dessus.

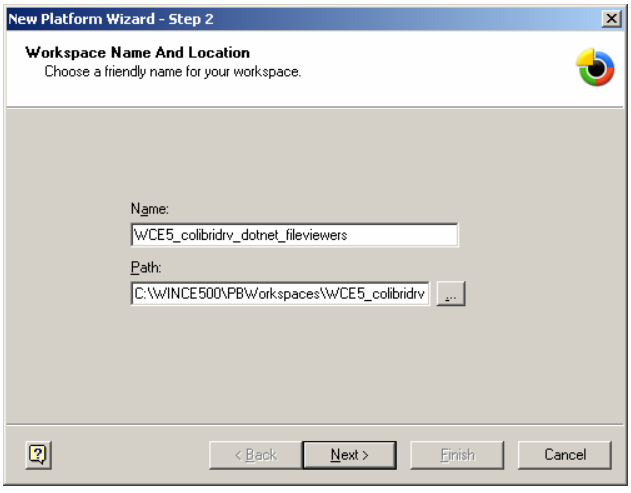
Le cas du PXA320

Toradex ne commercialise que depuis peu de temps le module PXA320. Il m'est d'ailleurs parvenu que 3 semaines avant la fin du projet, ne me laissant que peu de temps pour l'adapter à l'iPresenter2. Le BSP livré par Toradex intègre les pilotes et le noyau déjà compilé, mais uniquement pour le PXA270. Aucun BSP n'est encore disponible pour le PXA320. De ce fait, il ne sera pas possible dans le cadre du projet de créer une nouvelle image tournant sur le PXA320.

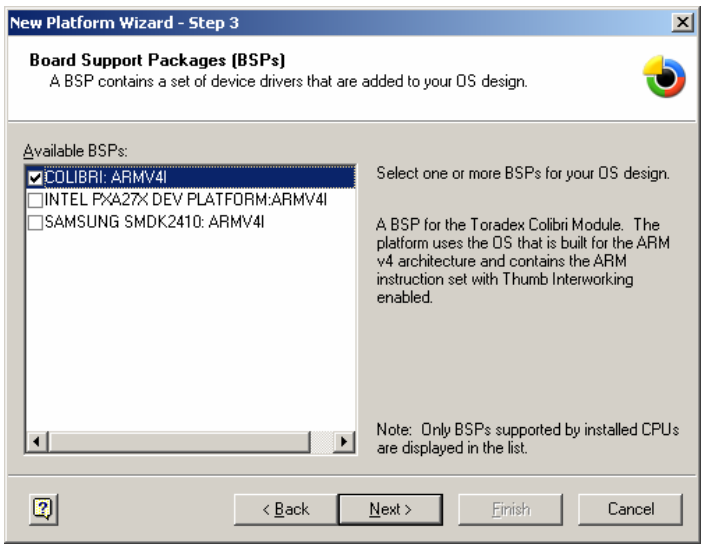
16.2. Création du projet sous Platform Builder

Les étapes suivantes décrivent pas à pas comment créer un nouveau projet permettant de compiler une image pour le module Colibri PXA270.

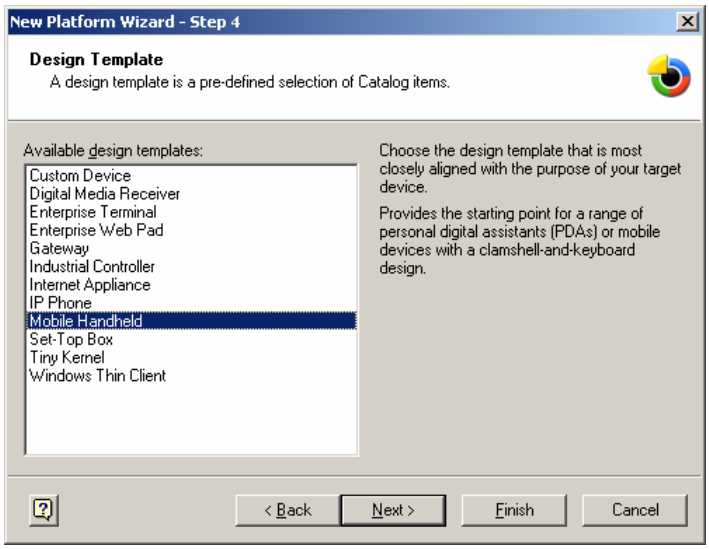
1. Démarrer Platform Builder mis à jour avec le BSP de Toradex installé
2. Cliquer sur File ➔ New Platform…
3. Un assistant s'ouvre, cliquer sur le bouton de l'écran de bienvenue « Next »
4. La fenêtre ci-contre s'affiche, donner un nom au projet ainsi qu'un emplacement sur le disque dur, puis cliquer sur « Next »



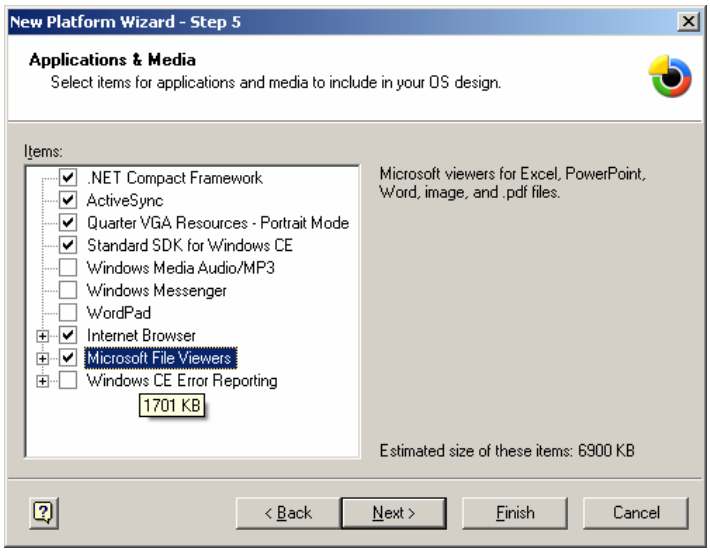
5. Choisir le BSP de Toradex, en cochant la case « Colibri : ARMV4I » puis « Next »



6. Choisir le modèle prédéfini « Mobile Handheld ». C'est celui qui correspond le mieux à l'iPresenter2. Cliquer sur « Next »



7. Dans la liste de composants à choisir, cocher l'élément « Microsoft File Viewers » en plus de ceux déjà choisis. Puis cliquer sur « Finish », puis « Next », puis « Finish »

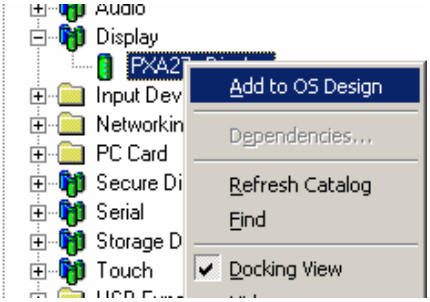


16.3. Choix des composants

Maintenant que le projet est créé avec les composants par défaut, il s'agit de rajouter à l'image tous les éléments nécessaires à la rendre démarrable. Moyennant quelques heures de support, il est possible d'obtenir de Toradex le projet sous Platform Builder avec tous les composants nécessaires choisis. C'est très utile, car si un pilote ou une fonctionnalité manque, l'image plante ou ne démarre plus du tout.

Un certain nombre d'heures a donc été nécessaire pour trouver et ajouter à l'image finale tous les éléments nécessaires pour que l'image soit presque identique à celle de Toradex, nos logiciels en plus.

Vous trouverez en annexe 07 deux pages contenant tous les composants qui doivent apparaître dans la colonne de gauche dans Platform Builder. Si l'un d'entre eux manque à votre projet, il suffit d'aller le chercher dans la colonne de droite, de faire un clic droit sur le composant et de choisir « Add to OS Design ».

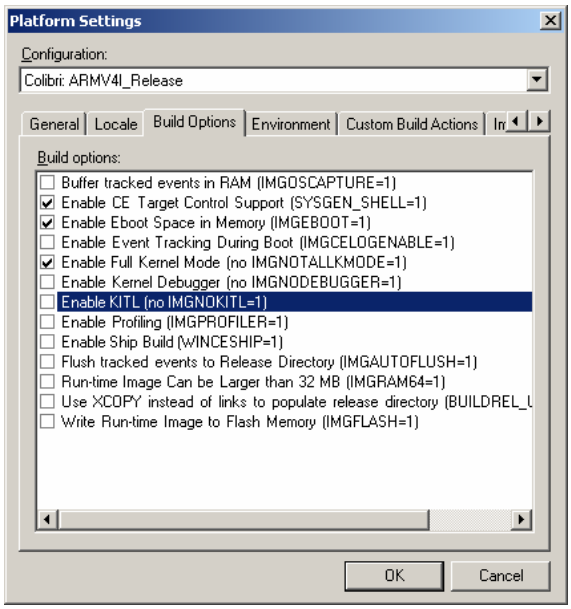


Vous trouverez sur le CD-ROM le projet complet sous Platform Builder avec tous les composants nécessaires déjà ajoutés. (*5_Software\2_Developpment\Image_PlatformBuilder*)

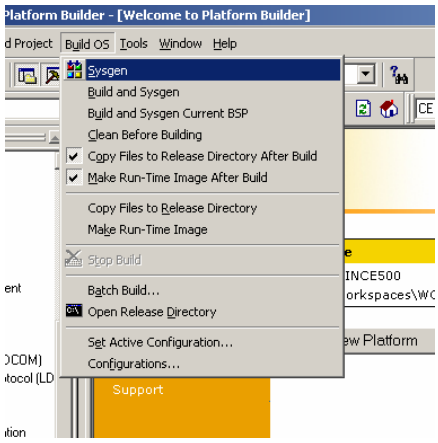
16.4. Compilation et copie en mémoire FLASH

Le but final de cette étape est d'obtenir un fichier binaire transférable dans la mémoire FLASH du Colibri. Tous les composants ajoutés au chapitre précédents constitueront cette image. A noter que le kernel n'est pas compilé et récupéré directement depuis le BSP de Toradex pour être ajouté à l'image finale.

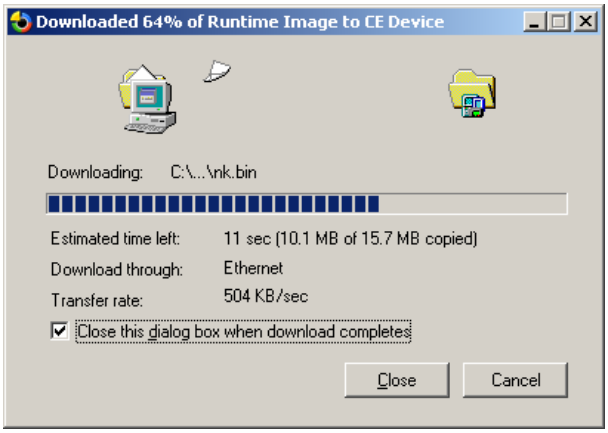
1. Dans le menu Platform ➔ Settings, Sous l'onglet « Build Options », décocher la case suivante : « Enable KITL ». Cette option (Kernel Independant Transport Layer) permet le debug du noyau via une connexion type Ethernet ou USB, elle n'est pas supportée par l'image du Colibri. Cliquer sur « OK ».



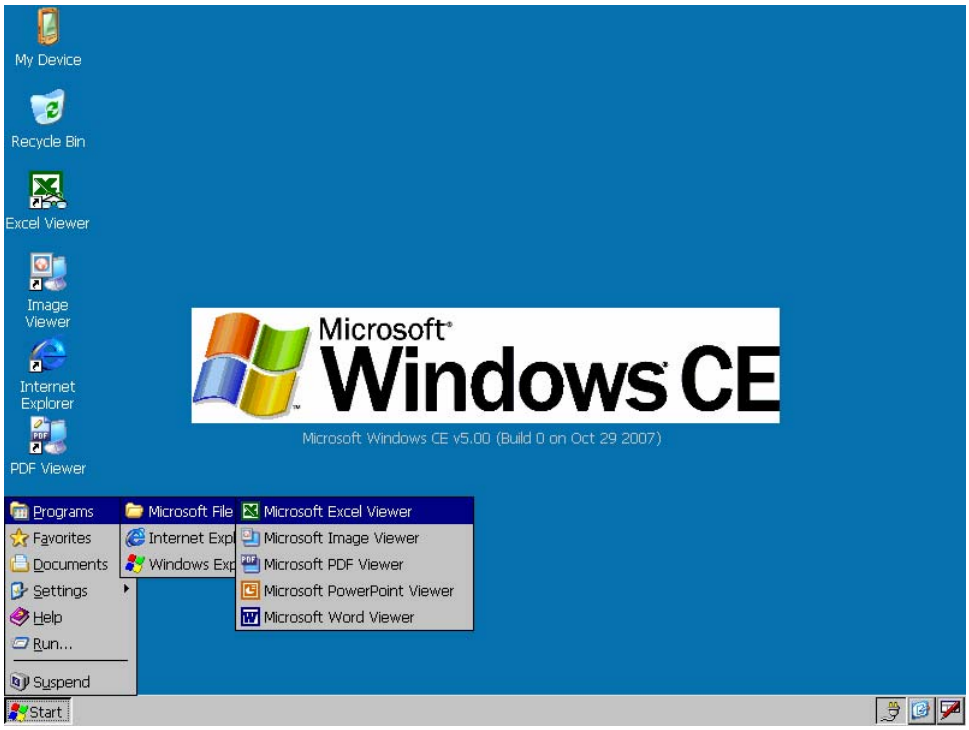
2. Dans le menu « Build OS », choisir « Sysgen ». A partir de maintenant, Platform Builder compile les éléments qui ne le sont par encore et rassemble tous les fichiers compilés dans un fichier image nommé NK.bin. Cette opération prend environ 15 minutes.



1. Connecter le câble série sur la carte de démonstration et ouvrir le terminal. (chap. 10.5.1). Appuyer sur la touche espace puis mettre sous tension la carte de démonstration
2. Appuyer sur la touche « D » pour placer l'image du nouvel OS en mémoire RAM ou sur « F » pour placer l'image du nouvel OS dans la mémoire « FLASH ». Cette opération est irréversible et le seul moyen de restaurer l'image d'origine de Toradex est d'utiliser ColibriLoader.
3. Connecter soit un câble réseau croisé entre le PC et la carte de développement, soit un câble droit en passant par un Hub ou un Switch.
4. Dans le menu « Target », cliquer sur « Attach Device ». Si tout est bien connecté, le transfert de l'image dans la mémoire du Colibri démarre.



5. A la fin du transfert, l'image compilée est lancée. Vous devriez voir cette image sur la sortie écran de la carte de démonstration.



16.5. Tests de la nouvelle image

Afin d'être certain que la nouvelle image était totalement fonctionnelle, les éléments suivants ont été testés :

- Présence sur le système et fonctionnement correcte de « Microsoft Powerpoint Viewer »
- Présence sur le système de la plateforme .NET compacte
- Présence sur le système et fonctionnement des autres logiciels de lecture
    - Microsoft Excel Viewer
    - Microsoft Image Viewer
    - Microsoft PDF Viewer
    - Microsoft Word Viewer
- Présence et fonctionnement de la partition en mémoire FLASH nommée « NOR Flash »
- Présence des utilitaires d'enregistrement du registre et de mise à jour dans le dossier \windows
- Communication avec activeSync possible
- Communication avec Platform Builder et tous les utilitaires comme l'éditeur de registre possible
- Compilation et debug d'applications via embedded Visual C++ 4.0 et Visual Studio 2005 possible

Tous ces tests ont été menés avec succès, l'image est donc opérationnelle. Elle pourra être utilisée à la place de celle de Toradex sur le PXA270.

C'est très intéressant car cela nous permet de créer n'importe quelle nouvelle image pour ce module avec des composants voulus. La carte de Toradex pourrait donc s'adapter à d'autres projets sur la même base, sans qu'aucune fonctionnalité supplémentaire ne doive être payée ou demandée à Toradex. Je pense donc que le temps investi pour réunir tous les composants nécessaires au démarrage de la carte a été particulièrement utile.

Il est à remarquer que la taille de l'image créée est environ de 15.7Mo. Le framework .NET et les logiciels de lectures ajoutés doublent donc presque la taille de l'image. La place dans la flash est cependant suffisante. (la marge laissé par Toradex permet de ne pas empiéter sur l'espace de données.)

A l'heure actuelle, le PXA270 est donc équipé d'une nouvelle image Windows CE alors que le PXA320 tourne toujours avec l'image d'origine. Cela permet de démontrer les deux possibilités au niveau de la lecture de fichiers Powerpoint ou encore du .NET Compact Framework. (copie au démarrage ou natif).

17. Développement du logiciel final

17.1. Introduction et découpage

Un petit récapitulatif des éléments clés qui sont maintenant en notre possession serait utile. Voici un tableau montrant l'infrastructure logicielle disponible sur chacun des modules Colibri :

Module	Colibri PXA270	Colibri PXA320
Lecteurs installés	Microsoft Powerpoint Mobile Microsoft Word Mobile Microsoft Excel Mobile Microsoft Image Viewer Microsoft PDF Viewer	Microsoft Powerpoint Mobile
Plateforme .NET compacte	Installée, natif	Installée, copié au démarrage
Ports USB	2, pris en charge	2, pris en charge
Mémoire interne	Environ 10MB de libre, partition « \NOR Flash »	Environ 980 MB de libre, partition « \FlashDisk »

L'idéal serait de développer une et une seule application qui pourrait être compilée pour l'une ou l'autre des cibles. Tous les deux processeurs ont un cœur ARM et les instructions ARMV4I fonctionnent sur les deux modules. Clairement, une application qui tourne sur un processeur tournera aussi sur l'autre, au niveau des instructions.

Par contre, certaines différences vont encore compliquer un peu la tâche de programmation. Typiquement les registres internes diffèrent entre les deux processeurs. Le datasheet complet du PXA270 est à ma disposition, mais pas celui du PXA320. Certaines limitations verront donc le jour lorsque le logiciel tournera sur le PXA320. Pour obtenir le datasheet complet de ce dernier, il faut signer un NDA chez Marvell (fabriquant des puces), cela n'a pu se faire dans les trois dernières semaines du projet (réception tardive du PXA320).

L'architecture du programme sera donc divisée en une ou plusieurs librairies de liaisons dynamiques C++ et un programme écrit en C#, faisant appel aux fonctions des librairies. Ce langage ayant été choisi en raison de mes connaissances plus poussées en C++ et en Java plutôt qu'en VB. C# est un langage neuf et a été conçu pour .NET. Il n'a pas un long passé comme VB, et ses concepteurs ont donc pu partir sur de nouvelles bases saines.

17.2. Les librairies de liaisons dynamiques DLL

Elles seront responsables des accès au matériel et aux fonctions bas niveaux du noyau. Voici la liste des tâches qu'elles devront effectuer :

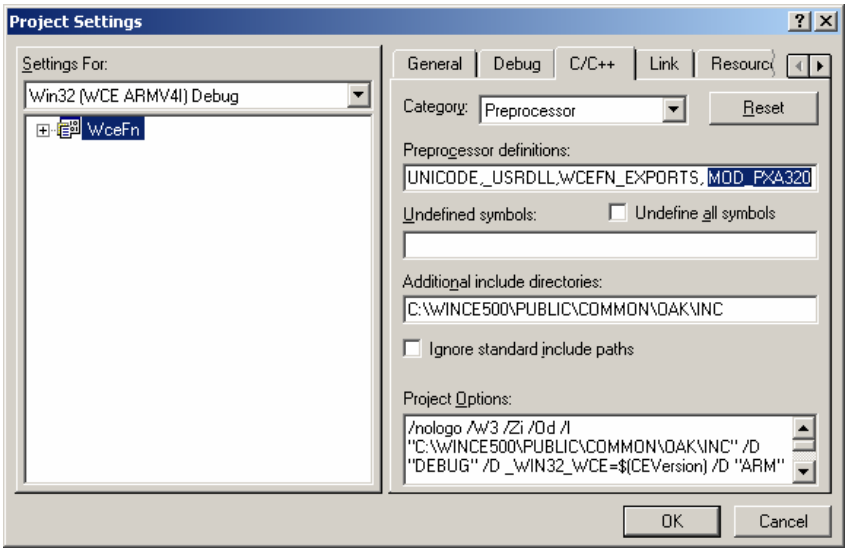
- Lire / écrire la valeur des GPIO
- Changer la direction / le mode des GPIO
- Envoyer / Recevoir des données sur le bus I2C
- Simuler une pression sur une touche clavier ou un clic de souris
- Lire / Ecrire des valeurs dans le registre Windows
- Redémarrer le noyau de Windows CE

Cette liste contient principalement deux types de fonctionnalités : les accès aux registres matériels (GPIO, I2C) et les accès au système d'exploitation (simulation de touches, registre, commandes noyau).

C'est donc ce découpage-là qui sera utilisé ; deux DLL seront créées, RegAccess.dll et WceFunc.dll.

Pour chacune de ces dll, un découpage interne a encore été mis en place, d'un côté les fonctions de bas niveau qui permettent de réaliser les fonctionnalités, et de l'autre des fonctions de plus au niveau qui seront mises à disposition pour le programme principal. Cela est très utile pour éviter d'alourdir le programme avec du code répétitif.

Au lieu d'avoir une version pour le PXA270 et une autre pour le PXA320, la compilation conditionnelle a été utilisée. Lorsque le code doit être compilé pour le 370, il suffit de rajouter une définition préprocesseur nommée MOD_PXA320 dans les options C/C++ du projet comme indiqué sur la fenêtre suivante :



Certaines fonctions dans les dll ne devront pas être disponibles depuis l'extérieur, car uniquement utiles pour le fonctionnement interne de celles-ci. Si l'on veut mettre à disposition une fonction de la dll (« exporter » cette fonction), il faudra ajouter la ligne suivante dans le fichier de définition « .h » du projet :

```
extern "C" _declspec(dllexport) int getUpButton(void);
```

En remplaçant par le nom, la valeur de retour et les paramètres de la fonction à exporter.

Tout le code des deux dll a été écrit pour s'exécuter de manière sécurisée. En clair, le programme teste toujours si l'instruction précédente s'est déroulée avec succès avant de continuer. Si une erreur a lieu, le programme appelant est dans tous les cas averti (valeur de retour booléenne ou négative par exemple).

Le code étant suffisamment documenté, je me limiterai à la description globale puis à quelques explications sur des parties de codes intéressantes à traiter, en particulier sur les raisons qui ont mené au développement de telle ou telle partie de code.

Il faut aussi noter que les deux dll ont été placées dans le dossier « Autocopy » de la mémoire FLASH, dans un sous-dossier nommé « windows ». Cela a pour effet de les copier au démarrage dans le dossier système. Ainsi l'application principale peut les utiliser directement.

17.2.1. RegAccess.dll

Cette dll permet à la base d'accéder aux registres matériels du processeur. Ceux-ci se situent à des adresses dites adresses physiques. On pourrait se dire qu'il suffit donc, par exemple avec un pointeur, d'accéder à l'adresse physique par exemple du contrôleur des GPIO (0x40E00000), et d'y insérer la valeur voulue.

Windows CE dans sa version 5 a mis en place un système de protection qui ne permet pas aux applications en mode utilisateur d'accéder directement aux registres matériels. Dès lors, notre dll va

devoir passer par un artifice utilisé dans tous les pilotes qui fonctionnent en mode user, basé sur deux fonctions :

- VirtualAlloc() qui permet de réserver un espace mémoire virtuel dans la mémoire **locale** du programme qui l'exécute
- VirtualCopy() qui va permettre de faire un lien entre une partie de la mémoire physique et la mémoire réservée auparavant. Dès lors, toute modification dans la mémoire locale réservée se répercute sur la mémoire physique et inversement.

Pour faciliter et clarifier le code, on va créer des structures de données qui correspondent exactement à celles des registres et de les lier à ceux-ci. On définira aussi en tant que constantes l'adresse de base des registres à lier et le nombre de bytes à copier.

Exemple

Définition des registres de base, taille et structure de données

```
#define CLK_REGBASE             0x41300000
#define CLK_PAGE_SIZE           4096

// Clock configuration registers
typedef volatile struct
{
                DWORD CCCR;         // Core Clock Configuration Register
                DWORD CKEN;         // Clock Enable Register
} CLK_REG;
```

Allocation de la mémoire programme et lien

```
pRegs = VirtualAlloc(0, CLK_PAGE_SIZE, MEM_RESERVE, PAGE_NOACCESS));
VirtualCopy(pRegs, (LPVOID)( CLK_REGBASE >>8), CLK_PAGE_SIZE, PAGE_READWRITE |
                        PAGE_PHYSICAL |  PAGE_NOCACHE)) ;
```

Modification de la mémoire physique en passant un élément de la structure

```
pCLKRegs->CKEN |=  (1 << 5) ;
```

L'accès aux registres est maintenant possible. Des fonctions d'un niveau un peu plus élevé permettent de modifier les registres qui concernent les GPIO. Elles ne sont pas mises à disposition du programme principal.

Elles permettent d'effectuer les actions suivantes :

- setGPIODirection() ➔ Mettre une GPIO en entrée ou en sortie
- setGPIOSpecFunc() ➔ Utiliser la GPIO comme une entrée/sortie standard ou sa fonction spécialisée associée (certaines entrées/sorties peuvent par exemple être les pins d'entrée/sorties du bus I2C)
- setGPIO() ➔ Défini la valeur d'une GPIO (0 ou 3.3V sur la pin)
- getGPIO() ➔ Récupère la valeur d'une GPIO depuis le registre d'état de celle-ci

Ce qui intéresse le programme de gestion est uniquement le contrôle de certaines GPIO. La dll mettra donc à disposition des fonctions de plus haut niveau tels que getDownButton() qui renvoie si la GPIO sur laquelle est connecté le bouton « BAS » est à l'état haut ou bas. Cela permet au programme principal de ne pas se soucier des détails d'implémentation hardware.

Le numéro des GPIO utilisées, l'adresses des registres ou encore les commandes à envoyer et recevoir sont des constantes définies dans un fichier nommé PXAReg.h. Si l'on connecte un bouton sur une autre pin, il suffira de modifier cette valeur. Toute la dll a été pensée dans ce sens pour faciliter les modifications futures.

A noter que (par chance) les registres utilisés pour le contrôle des GPIO sont les mêmes sur le PXA270 et le PXA320. Il a donc été possible de réutiliser directement le même code. Les numéros des GPIO ont par contre tous changé et ont dû être réadaptés en utilisant la compilation conditionnelle.

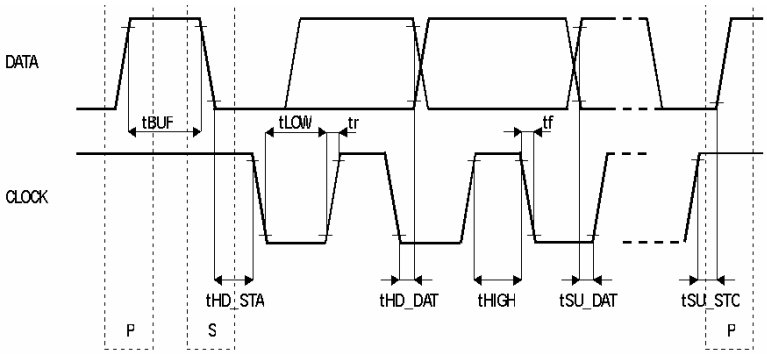
La gestion de l'I2C

De nombreuses heures ont été perdues sur ce point précis. La communication via le bus I2C est nécessaire afin de pouvoir questionner la puce ISL6295 du circuit d'alimentation. Elle est capable de nous communiquer l'état de la batterie, les courants qui circulent ou encore la température de l'appareil.

Le PXA270 est pourvu d'un contrôleur I2C interne, relié aux pins SDA et SCL du module Colibri. C'est celles-ci qui ont été câblées sur l'ISL6295 lors de la réalisation de la schématique. Hors, aucun pilote I2C n'est disponible et directement utilisable pour piloter le port. Comme d'habitude, Toradex met à disposition le code pour accéder au contrôler I2C, moyennant finance évidemment.

Dès lors, je suis d'abord parti du principe que l'envoi de quelques bytes de configuration et la lecture d'un registre par I2C ne devait pas être une tâche trop ardue à réaliser soi-même. J'ai donc recréé manuellement les signaux de clock et de données permettant la communication avec l'ISL6295 en utilisant les fonctions setGPIO() et getGPIO() précédemment mises en place.

Une étude approfondie du protocole I2C a du être réalisée pour connaître et appliquer précisément tous les timings et changements d'état pour recréer les bits de start, bit de stop, et bit d'envoi/réceptions de données. (Ce code est toujours inclus au projet du programme de gestion mais non utilisé et non documenté.)



Cela fonctionnait, il était possible de récupérer la tension de la batterie depuis la puce. Malheureusement, celle solution était vouée à l'échec à plus ou moins long terme. Cela est dû au fait que les timings du bus I2C ont besoin d'être précis. Or, lorsqu'un programme tourne sous Windows CE, il suffit qu'une interruption ou tout autre événement de priorité supérieure apparaisse pour ralentir ou momentanément stopper l'exécution du code, et donc sortir des timings acceptés par l'ISL6295. Cette dernière se mettait donc dans un état indéfini et seul un reset complet (retirer la batterie) permettait de le remettre sur les rails.

C'est donc une autre solution qui fut choisie : celle d'utiliser directement le contrôleur I2C intégré dans le processeur. Cela permet d'avoir du hardware qui s'occupe essentiellement de l'envoi/réception de données sur le bus et donc d'assurer les timings.

Des fonctions permettant l'initialisation du contrôleur, l'envoi et la réception de paquets et la scrutation des registres d'état ont du être écrites. Elles s'inspirent des procédures présentes dans le datasheet complet du PXA270 (Developper-Manual présent sur le CD-ROM en annexe), chapitre 9.4.9. (*3_Composants\1_Datasheets\Colibri*).

Ces dernières sont mises à disposition du programme principal, qui est responsable de la conversion de données. Ce choix a été fait car l'utilisation de nombres à virgules est nécessaire, cela avec une certaine précision. Le framework .NET est capable de le faire mieux que les classes de base C++.

A noter que les registres I2C du PXA270 et du PXA320 sont différents, c'est donc pour cela que lorsque l'on compile pour le PXA320, tout le code I2C est désactivé. Il faudrait connaître l'adresse de base ainsi que l'organisation des registres du 320 pour pouvoir réactiver le code. N'ayant pas le datasheet complet à disposition, la fonctionnalité est inutilisable.

Le code de la dll commenté se situe en annexe 08. Le projet visual Embedded C++ est disponible sur le CD-ROM en annexe. (*5_Software\2_Developpment\regAccess*)

17.2.2. WceFunc.dll

La deuxième dll est responsable de fonctions de plus haut niveau, touchant au kernel, à la base de registre et aux simulations d'événements.

Simulation de touches clavier/souris

Un problème est survenu lorsqu'il fallu piloter le logiciel Microsoft Powerpoint Mobile depuis l'application de gestion. Aucun touche de clavier ne permet de faire défiler une diapositive, ni de revenir directement à la précédente lorsque la présentation est ouverte. Le seul moyen possible pour faire défiler les diapositives vers l'avant est un clic gauche de souris. Pour le retour en arrière, il faut d'abord faire un clic droit de souris, et aller sur le menu contextuel sur un bouton appelé « Previous » et cliquer à nouveau avec la souris. Par chance, un clic droit suivi d'une pression sur la touche « P » permet aussi de revenir à la diapositive précédente. (le driver du clavier doit être actif)

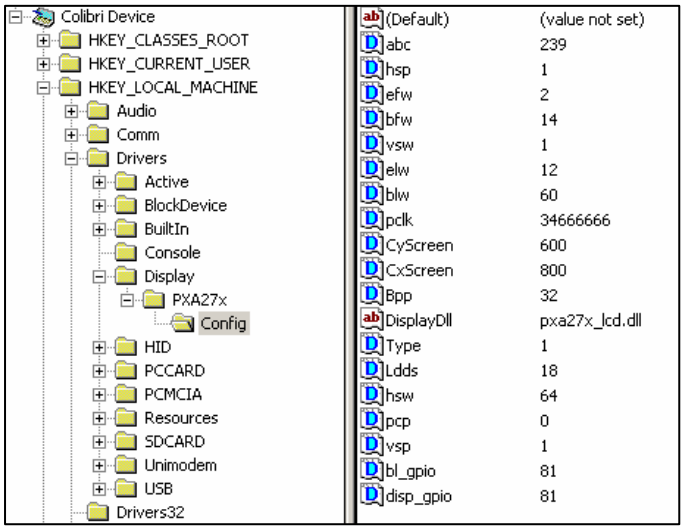
Cela est très sûrement dû au fait que Powerpoint Mobile a été conçu pour tourner sur des périphériques mobiles de type PDA ou PocketPC qui ne sont en général pas équipés de clavier. Ils ont privilégié l'utilisation du touchpad et de la souris.

Il a donc fallu mettre à disposition du programme de gestion des fonctions permettant d'émuler un clic droit et gauche de souris, ainsi qu'une pression sur la touche « P » du clavier. Des fonctions permettant le placement du curseur à un endroit précis ou encore de le cacher ont aussi été ajoutées (afin de gérer au mieux celui-ci durant les présentations.

Changement de résolution d'écran

Tout le reste du code présent dans la dll est écrit pour permettre de changer la résolution de l'écran depuis le logiciel de gestion. Beaucoup de temps a aussi été nécessaire pour faire fonctionner cette fonctionnalité.

Le contrôleur graphique intégré dans le processeur du PXA270 et PXA320 est géré par un pilote fourni par Toradex. Leur code étant apparemment bien écrit, il est possible via des clés de registres de modifier tous les paramètres de résolution, couleurs et options des pixels.



Cela devrait nous permettre de changer relativement facilement depuis notre programme la résolution actuelle de l'écran. Il faut savoir qu'il est nécessaire de redémarrer le système pour que la nouvelle résolution soit prise en compte. Ce n'est pas tout à fait vrai, en fait il suffit de redémarrer le noyau. La différence peut sembler infime mais elle permet des choses très intéressantes. En effet lorsqu'un redémarrage complet est réalisé (hardware reset), toutes les données qui étaient présentes en mémoire RAM et non enregistrées (fichiers copiés depuis la FLASH, paramètres modifiés dans la base de registres,…) sont effacés. Ce qui n'est pas le cas lorsque l'on effectue un reset du noyau uniquement (hot restart). Cela nous permet deux choses très intéressantes :

- Tous les fichiers copiés au démarrage sont gardés dans le système de fichier (gain de temps lors du redémarrage sur le PXA320)
- La base de registre n'a pas besoin d'être sauvegardée, il suffit de changer les paramètres de résolution d'écran et d'effectuer un hot restart. Ceci est un double avantage car l'on a même pas à se poser la question de comment récupérer la résolution standard d'affichage en cas de mauvais choix : il suffit d'un reset complet. Tout changement de résolution n'est gardé que durant le temps où l'appareil est allumé et ne subit pas de remise à zéro.

Un autre problème est survenu lors du passage à la résolution XVGA. Le PXA270 ou plus précisément la fréquence de fonctionnement du bus principal n'est pas suffisamment élevée pour permettre d'afficher des images à une résolution de 1024x768 à 60Hz. On arrive tout juste à atteindre les 50Hz.

C'est là que l'acquisition du PXA320 prend tout son sens, hormis l'affichage nettement plus rapide des présentations : son bus principal peut travailler jusqu'à 208MHz alors que celui du PXA270 et limité à 52MHz. Cela permet de monter confortablement en résolution et atteindre les 1024x768 / 60Hz requis.

Remarque : Il arrivait souvent, lorsque je travaillais en 1024x768 à 50Hz avec le PXA270 que l'écran se mettait à scintiller, voir que l'image complète se brouillait et disparaissait. Après quelques échanges de mails avec Toradex, il s'avère que le problème vient du fait que lorsque l'on utilise des périphériques connectés sur le bus mémoire (le contrôleur écran l'est aussi) tels que la FLASH ou le contrôleur CompactFlash, le buffer de l'écran peut se vider si les données ne lui parviennent pas suffisamment vite.

Ce tableau résume les résolutions qui seront utilisées avec les différents modules Colibri :

Module	Colibri PXA270	Colibri PXA320
Résolutions supportées	VGA (640 x 480) @ 60Hz SVGA (800 x 600) @ 60Hz XGA (1024 x 768) @ 50Hz	VGA (640 x 480) @ 60Hz SVGA (800 x 600) @ 60Hz XGA (1024 x 768) @ 60Hz
Résolution par défaut (boot)	SVGA (800 x 600) @ 60Hz	SVGA (800 x 600) @ 60Hz

La librairie devra donc mettre à disposition deux fonctions de bas niveau :

- une permettant le redémarrage du noyau de Windows CE uniquement
- une autre permettant de lire et d'écrire des valeurs dans le registre Windows

La fonction de redémarrage est extrêmement simple, elle appelle du type « trap » du noyau permettant de lui donner des commandes spécifiques, dont celle de redémarrer. Voici le détail :

```
KernelIoControl(IOCTL_HAL_REBOOT, NULL, 0, NULL, 0, NULL);
```

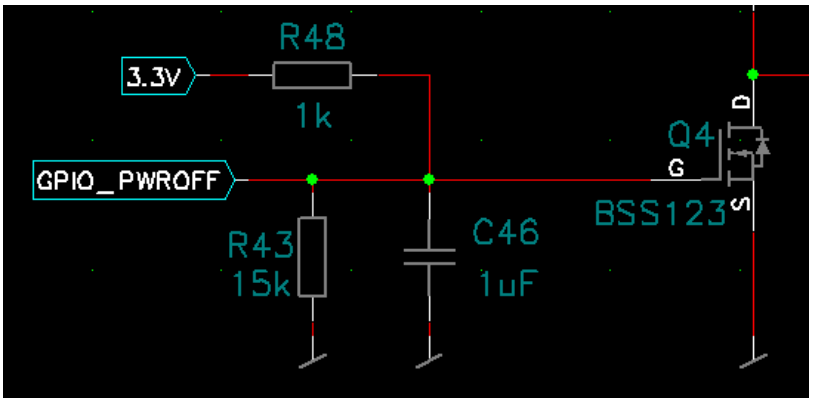
Problème au niveau de l'alimentation

Un souci non soupçonné se produit lors de l'exécution de cette commande : un redémarrage du noyau implique aussi une ré-exécution du code d'initialisation de l'OAL, lequel configure toutes les GPIO en mode « entrée / haute impédance » avant de leur redonner leur état normal. Cette opération a pour but pur et simple de faire lâcher le maintien de l'alimentation de la carte et éteint l'appareil.

La recherche d'une solution à ce problème a de nouveau engendré une perte de temps et implique une modification matérielle. Au lieu que cela soit le processeur qui maintienne l'alimentation en appliquant une tension de 3.3V sur la base du transistor Q4, ce sera directement l'alimentation qui va effectuer un auto-maintien en prélevant la tension générée par la ligne 3.3V sur la base de Q4 via une résistance.

Ainsi une très courte impulsion sur le bouton « Power » suffit à enclencher la carte (plus besoin d'attendre que le processeur démarre). Le processeur doit par contre configurer la GPIO de maintien en entrée par défaut. C'est uniquement lorsque l'on voudra éteindre l'appareil qu'il faudra appliquer un zéro en sortie sur la base de Q4, ce qui aura pour effet de couper l'alimentation. Il faut modifier l'état de la GPIO au démarrage, via le programme GPIOConfig (cf. chapitre 10.4.5)

Le condensateur de 1uF C46 permet d'effectuer un filtre RC (R48 – C46). Ce filtre retarde la remontée du signal d'alimentation après la coupure du processeur (tension résiduelle des condensateurs de découplage de la carte). Cela permet d'éviter que la carte ne redémarre.



La schématique a été modifiée en conséquence. Il faudra refaire le routage de cette partie.

Lors des modifications des entrées du registre, le code utilise les fonctions RegOpenKeyEx(), RegQueryValueEx() et RegSetValueEx(). Toutes les étapes sont contrôlées et un code d'erreur est retourné si un problème a lieu.

La fonction de plus haut niveau setDisplayResolution() va simplement mettre dans le registre une liste de valeurs prédéfinies permettant l'affichage dans chaque résolution listée ci-dessus. La fonction getDisplayResolution() permet au programme principal de savoir en quelle résolution l'appareil travaille actuellement. Ces deux dernières fonctions sont mises à disposition du programme principal.

Le code de la dll commenté se situe en annexe 09. Le projet visual Embedded C++ est disponible sur le CD-ROM en annexe. (*5_Software\2_Developpment\WceFn*)

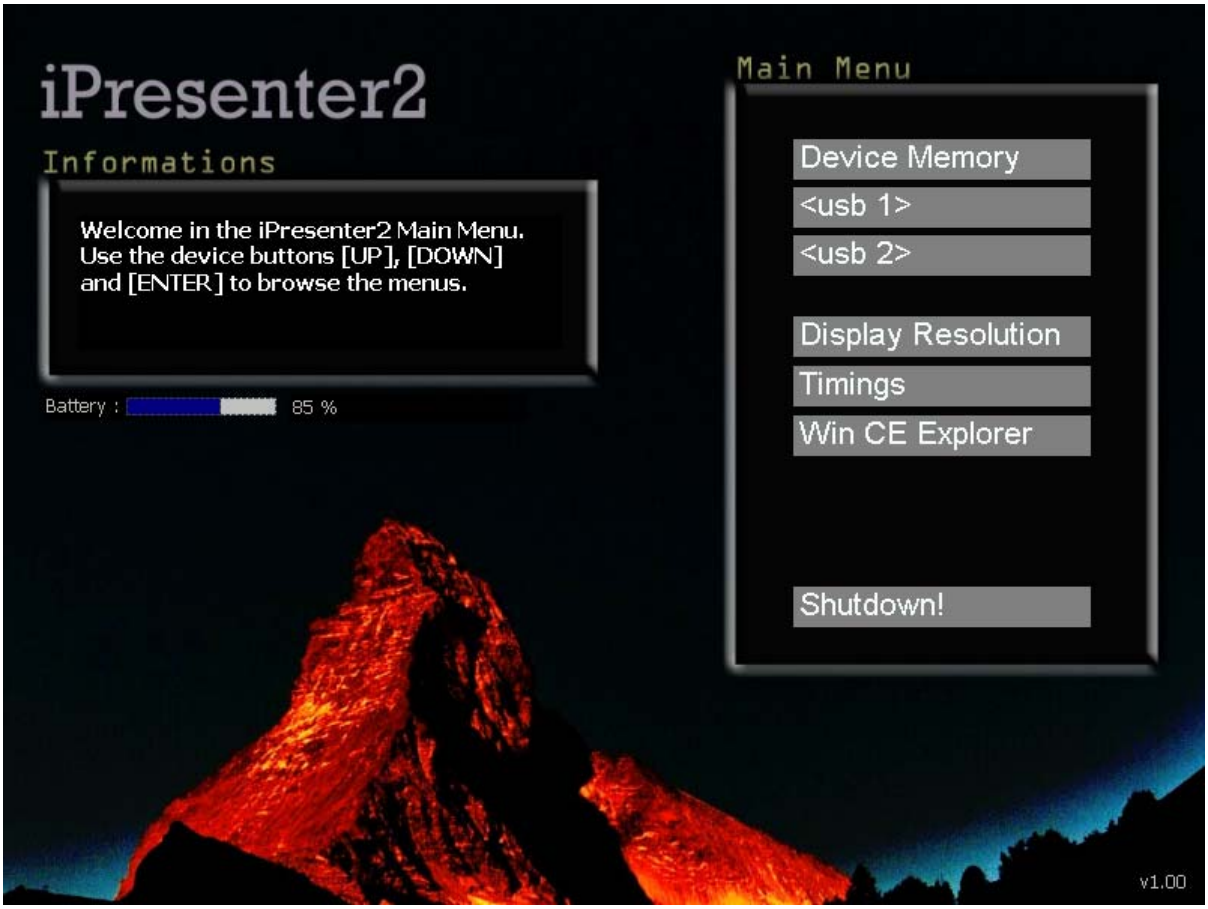
Remarque : les valeurs prédéfinies de chaque registre pour les différentes résolutions ont du être trouvées manuellement et ont été testées sur différents écrans plats, à tube, et sur la plupart des projecteurs de l'école. Aucun problème constaté. Le respect des normes VGA, SVGA et XGA semble bon.

18. Le logiciel de gestion

18.1. Présentation

Le logiciel principal est celui qui s'affiche au démarrage de l'iPresenter2. Il se doit donc d'être le plus convivial, simple et utilitaire possible. Le design a aussi son importance, point pour lequel le framework .NET permet beaucoup de libertés et de possibilités. Il a été long à mettre en œuvre, il est cependant vital que l'utilisateur est une bonne expérience avec celui-ci car c'est le seul avec lequel il interagira.

Voici un aperçu de l'écran d'accueil du programme principal, développé en C# :

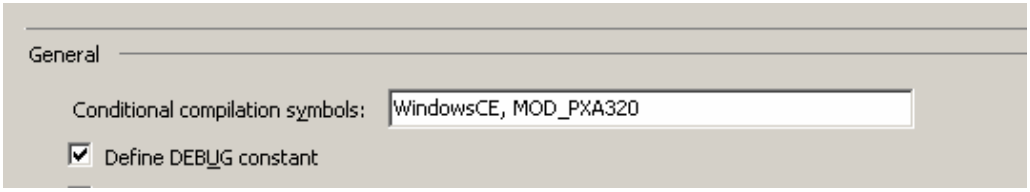


Le menu de droite permet de naviguer entre toutes les fonctionnalités développées via les touches « HAUT » et « BAS ». A chaque pression, le menu précédent ou suivant est mis en surbrillance. Une pression sur le bouton « Enter » exécute la fonctionnalité qui peut soit être un autre menu soit une action directe.

L'écran d'information en anglais donne une aide contextuelle, qui s'adapte à chaque sélection de l'utilisateur. La barre d'avancement et le pourcentage indiquent ce qui reste dans la batterie (estimation).

L'image en fond d'écran a été créée à l'aide d'Adobe Photoshop et peut être modifiée. Cela permet de changer de « skin » facilement. On pourrait imaginer proposer d'autres skins sur un site web. Cette image est disponible sur le CD-ROM au format Photoshop et JPEG. (5_Software\2_Developpment\MainMenu_Background)

A nouveau, la compilation conditionnelle a été utilisée pour éviter de devoir développer deux applications distinctes. Il faut ajouter le symbole « MOD_PXA320 » si on veut compiler pour cette cible (menu Projets ➔ iPresenterMainMenu Properties)



Encore une fois, le code est documenté de manière complète. Ce rapport se concentre donc uniquement sur le fonctionnement général, sur les raisons qui ont poussé à écrire telle ou telle partie de code ou encore des explications précises sur certaines lignes complexes.

Le code n'a pas été imprimé en annexe, car il est particulièrement lourd. En effet plus de 3000 lignes de codes ont été écrites. Il est disponible sur le CD-ROM en annexe. (*5_Software\2_Developpment\iPresenterMainMenu*).

18.2. Structure et déroulement du programme

Au vu de la taille et de la complexité du programme, j'ai jugé nécessaire d'ajouter au rapport une explication globale de son déroulement et de sa structure. Un structrogramme UML est disponible en annexe 10. Il contient toutes les classes utilisées dans le programme principal. Il est aussi disponible sur le CD-ROM en annexe (4_Documentation\ AN10_DiagrammeUML.doc). Toutes les classes décrites dans le structogramme sont documentées dans le code. Pour chaque d'entre-elle on retrouvera une description générale de son fonctionnement et de son utilité ainsi qu'une liste et une explication du rôle de chaque fonction présente dans celle-ci.

Le programme s'exécute sous la forme d'une grande boucle qui va tester l'état des boutons chaque 50 millisecondes. Si l'un d'entre eux est pressé, il exécute le code adéquat en fonction du contexte. Le contexte est donné par une variable globale, qui définit dans quel état et quelle action est entrain d'exécuter l'appareil. Cela peut paraitre lourd, mais la boucle ne fait rien si aucun bouton n'est appuyé, ce qui limite l'activité processeur au minimum. On pourrait se demander pourquoi je n'ai pas utilisé d'interruption directe depuis les boutons, le fait est qu'il n'est pas possible de les atteindre directement en C#.

Chaque sous-menu est une nouvelle fenêtre, communément appelée « Form » sous .NET. Les sous-menus (qui sont aussi des classes) sont en général passifs et mettent à disposition des fonctions pour que le programme principal puisse les animer. On retrouvera par exemple très souvent les fonctions selectNext() (qui permet de choisir l'élément suivant dans la liste lorsque l'utilisateur appuie sur le bouton « bas » ou getSelection() qui permettent de savoir sur quel objet s'est arrêté l'utilisateur lorsqu'il a appuyé sur le bouton « Enter ».

Le programme principal est stocké dans la mémoire FLASH, dans un sous-dossier appelé « app ». Il est exécuté automatiquement au démarrage du Colibri grâce au placement d'un raccourci dans le dossier « autorun » placé lui aussi dans la mémoire FLASH.

18.3. Importation des fonctions de la DLL

Notre programme C# fera très souvent appel aux fonctions des deux dll créées. Chacune d'entre elle est importée au tout début du code lors de l'initialisation de la classe principale.

Voici le code permettant de le faire :

```
        [DllImport("regAccess.dll")]
        static extern bool I2CWriteISL6295Reg(char slaveAddr, int bank,
            char regAddr, char value);
```

Elle va aller chercher une fonction nommée I2CWriteISL6295Reg() dans la dll regAccess.dll présente soit dans le même dossier que le programme lui-même, soit dans le dossier système de Windows (\windows). Il faut que « l'empreinte » de la fonction corresponde exactement à celle définie dans la DLL, sinon cela ne fonctionne pas (nom + type de retour + type des paramètres).

Ce code doit être répété pour chaque fonction utilisée depuis les dll. A noter que les fonctions concernant l'I2C sont ignorées dans le cas où la compilation a lieu pour le PXA320.

18.4. Gestion clés USB / mémoire interne

La gestion du système de fichier est relativement simple et basée sur des classes prévues à cet effet. En particulier les suivantes : `FileInfo`, `File` et `Directory`. Les méthodes GetDirectories() et GetFiles() de Directory permettent de lister le contenu d'un répertoire. Les méthodes Move() et Delete() de File permettent de déplacer ou supprimer des fichiers. Les différents attributs de FileInfo permettent de connaître la taille, l'extension ou encore le dossier parent d'un fichier précis.

La détection de la présence de clés USB se fait simplement en contrôlant si le dossier du répertoire racine \ contient des sous-dossiers nommés « USB HD » et « USB HD2 » qui sont les noms utilisés lorsque des périphériques de stockage de masse sont reconnus par Windows CE. Il peut y en avoir 2. Lorsque c'est le cas, les menus <USB 1> et/ou <USB 2> s'affichent ou disparaissent du menu principal.

Il est aussi possible d'atteindre un emplacement dans la mémoire interne, nommée « NOR Flash » sur le PXA270 et « FlashDisk » sur le PXA320. Le sous-répertoire de stockage utiliateur dans cette mémoire est défini en taut que constante au début du programme.

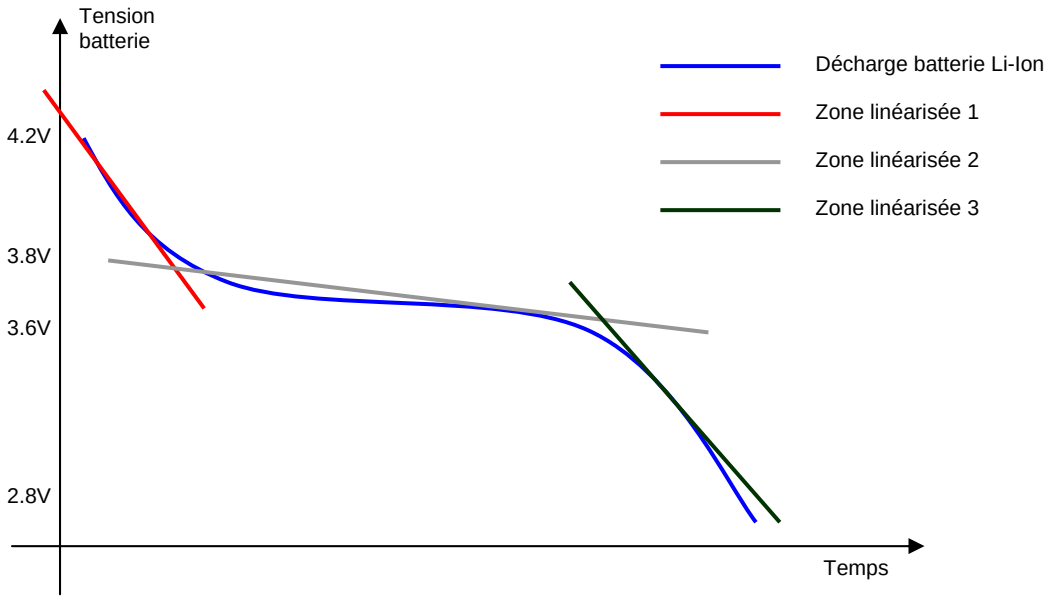
A noter que les chemins par défaut sont différents pour les deux processeurs. Des chaînes de caractères constantes ont été définies et leurs valeurs sont différentes selon le processeur utilisé (compilation conditionelle en début de programme).

18.5. Etat de la batterie

Comme précisé dans le code de la dll, le programme principal dispose de fonctions permettant d'atteindre les registres du circuit ISL6295. Il va tout d'abord procéder à l'initialisation de la puce de gestion batterie au démarrage du programme. Ensuite, un timer va aller chercher à intervalle régulier (chaque seconde) la valeur contenue dans les deux registres de la tension batterie (MSB et LSB). Le programme converti ces valeurs en tension comprise entre 2.8V et 4.2V (utilisation de nombres à virgules) via une fonction privées dans le programme principal (`getI2CBatteryVoltage()`).

L'annexe 11 décrit précisément les valeurs et les registres qui sont utilisées durant cette procédure. J'ai jugé important de l'ajouter car j'ai perdu un certain temps à comprendre comment faire fonctionner cette communication I2C.

D'une fois que la tension en volts de la batterie est connue, il faut encore la transformer en pourcentage restant avant de l'afficher. Il aurait été simple d'utiliser une relation linéaire, 2.8V correspondant à une batterie vide, 4.2V à une batterie pleine, et de faire diminuer cette valeur de manière linéaire avec la chute de tension. Cette méthode, bien qu'utilisée dans de nombreux appareils électroniques est peu précise dans le cas des batteries Lithium Ion. Leur courbe de décharge est loin d'être linéaire. Afin d'améliorer la précision de cette mesure, j'ai décidé de linéariser la courbe de décharge dans trois zones, de la manière décrite sur le graphique ci-dessous :



Une fonction nommée `getBatteryState()` est justement responsable de cette linéarisation. Elle définit ces trois zones et applique des facteurs de pente différents pour les trois zones. Elle permet d'être particulièrement sensible dans la zone de décharge de 3.8V à 3.6V, là ou la batterie se trouve durant la plus grande partie de sa phase de décharge.

D'une fois que la valeur en pourcent a été calculée, il suffit de configurer la taille de la barre bleue et de reprendre la valeur chiffrée à droite.

Le programme profite après avoir mis à jour la barre bleue et la valeur en pourcent de contrôler dans quelle plage se situe l'état de la batterie. Si elle est inférieure à 20%, une icône de batterie se met à clignoter dans le menu principal. La LED bicolore est aussi mise à jour. Si elle émet un clignotement en vert chaque seconde, cela signifique qu'il reste plus de 40% d'énergie dans la batterie. Elle se met à clignoter en rouge lorsque que ce n'est plus le cas. Le nombre de clignotement augmente à mesure que la tension baisse. Lorsque celle-ci passe en dessous de 10%, elle clignote de manière continue.

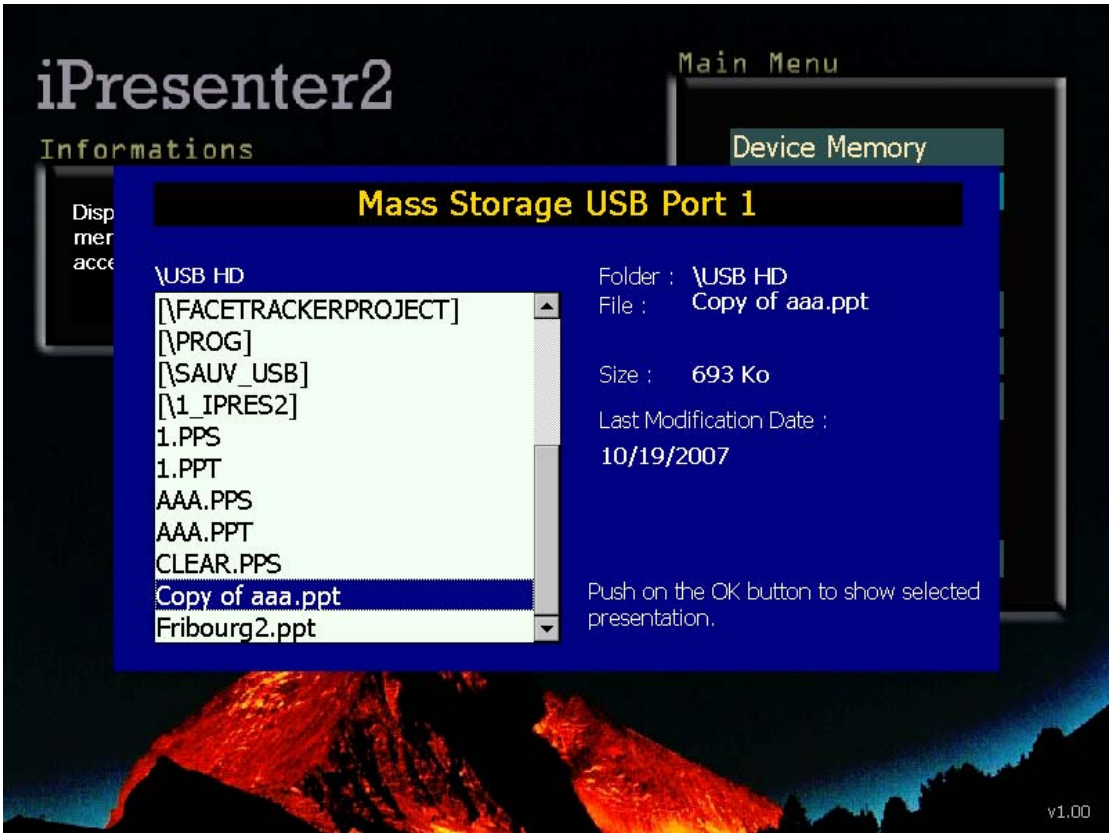
C'est un bon moyen pour l'utilisateur de savoir combien il reste d'énergie dans la batterie lorsqu'une présentation est en cours. Il lui suffit de regarder la couleur et le nombre de clignotements successifs chaque seconde sans quitter sa présentation.

Afin de préserver la batterie, le logiciel vérifie que la tension de la batterie ne soit jamais inférieure à 2.85V. (la sécurité de la batterie de déclenche à 2.8V). Si c'est le cas, l'alimentation est immédiatement coupée.

Remarque : L'ancienne version de la classe permettant d'accéder à la puce via un protocle I2C « fait maison » porte le nom d'isl6295.cs dans la hiérarchie du programme principal. Comme prédemment expliqué, elle n'est pas utilisé mais a été conservée pour montrer les méthodes utilisées.

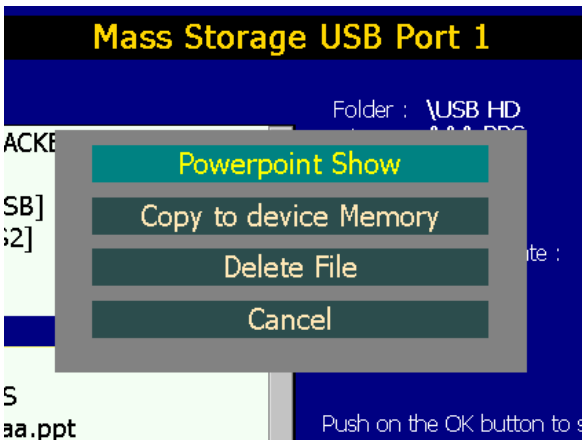
18.6. Explorateur de fichiers

Voici un aperçu de l'explorateur de fichiers. L'utilisateur a dans l'exemple ci-dessous ouvert le contenu d'une clé USB connectée.



La liste des dossiers et des fichiers est générée par la classe Directory. Les répertoires sont mis entre crochets alors que les fichiers n'ont pas de signes supplémentaires. A droite sont affichées des informations générales sur le fichier sélectionné : le nom du dossier contenant, la taille du fichier ou encore la date de dernière modification de celui-ci.

Le sous-menu ci-dessus apparaît après une pression sur « Enter » lorsqu'un fichier et sélectionné.

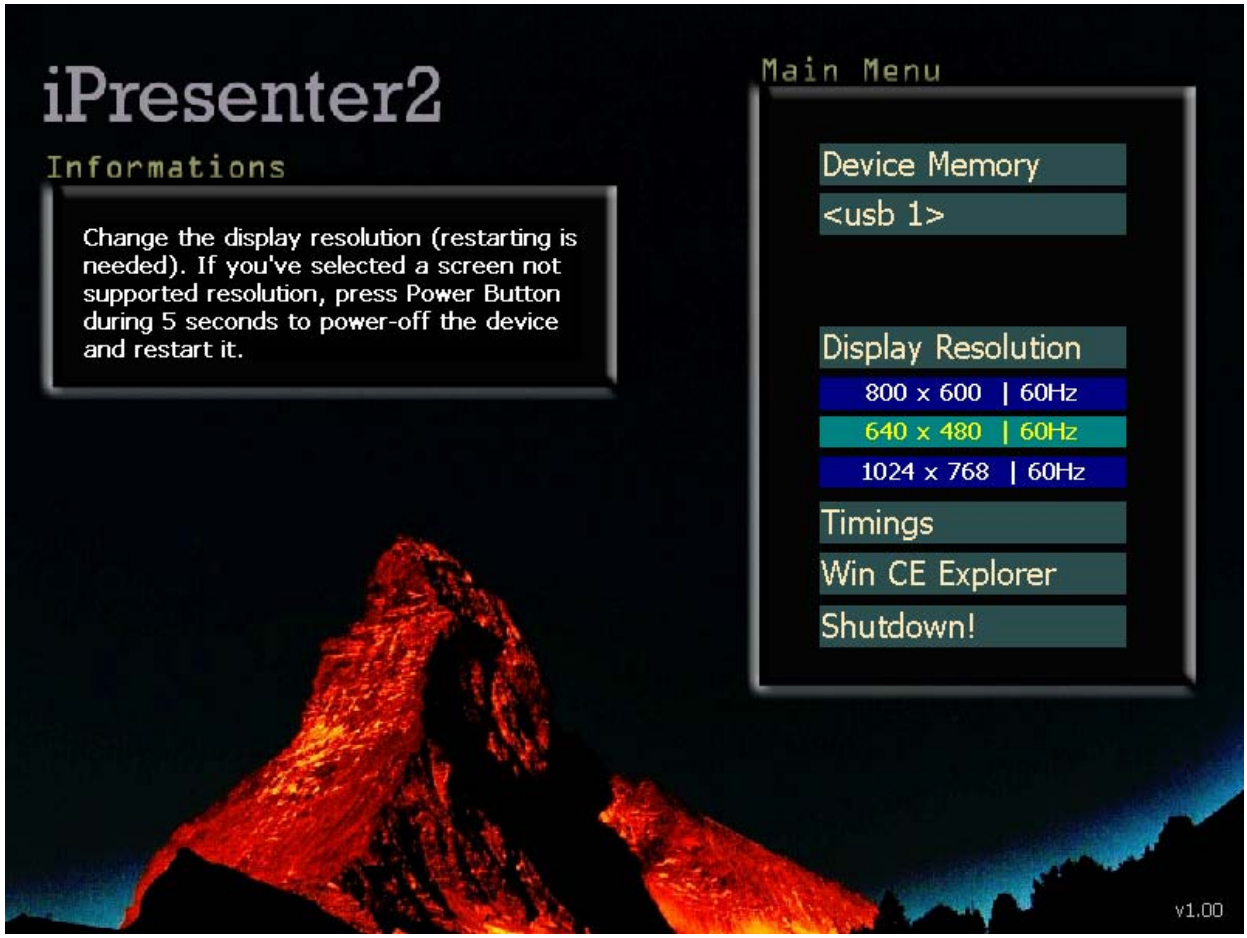


Il permet les opérations suivantes :

- Démarrer l'affichage de la présentation Powerpoint
- Copier dans la mémoire interne le fichier sélectionné. Cette fonction est la seule qui permet de remplir la mémoire interne. La copie est mise dans une structure de gestion d'erreur du type try{}…catch{}. Cela permet de détecter si une erreur s'est produite lors de la copie, par exemple due à une mémoire saturée ou un fichier du même nom existant. Un message d'erreur averti l'utilisateur dans ce cas.
- Supprimer le fichier. C'est le seul moyen de vider le contenu de la mémoire interne.
- Fermer le sous-menu et revenir à l'explorateur de fichier sans effectuer aucune action.

18.7. Réglage de la résolution de l'écran

Voici un aperçu du sous-menu de sélection de la résolution écran :

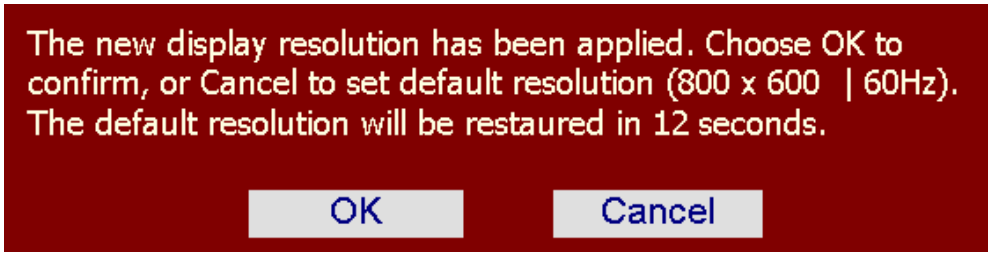


Lorsque l'utilisateur appuie sur le bouton « Enter » en ayant sélectionné le menu « Display Resolution », le sous-menu s'affichage. Il peut dès lors choisir une des trois résolutions proposées. Lorsqu'il le fait, un menu de confirmation s'affiche à l'écran.

S'il confirme le changement de résolution, le programme appelle les fonctions dll qui vont changer les valeurs du registre puis redémarrer l'appareil.

Au démarrage du programme principal, une partie de code appelle la fonction dll qui retourne la résolution actuelle. Si celle-ci n'est pas la résolution par défaut pour laquelle a été écrit le programme (800x600 pixels), il va adapter la taille des caractères, des objets et des fenêtres pour s'adapter à la nouvelle résolution.

Au redémarrage, le message suivant s'affiche sur l'écran :

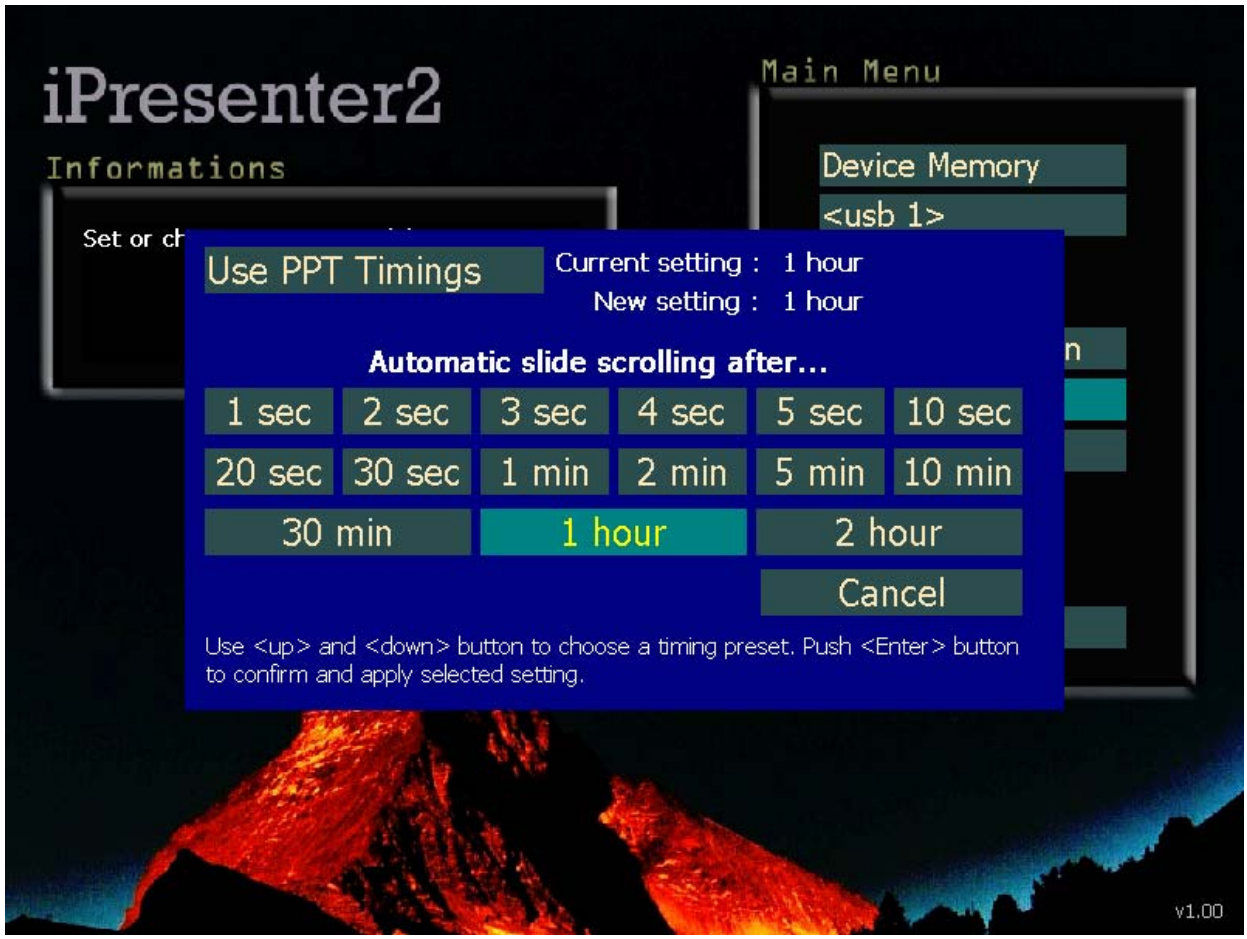


Si l'utilisateur ne le quittance pas au bout de 15 secondes, le programme réapplique la résolution par défaut. Cela permet d'éviter le choix d'une résolution non supportée.

A noter que le sous-menu de la résolution 1024x768 voit son taux de rafraîchissement varier selon que l'on compile pour l'un ou l'autre processeur (50Hz ou 60Hz). L'appel à la dll reste le même car c'est à l'intérieur de celle-ci que la modification est faite, lors de la compilation conditionnelle.

18.8. Menu Timings

L'écran suivant montre l'iPresenter2 lorsque l'utilisateur a appuyé sur « Enter » lorsque le menu sélectionné était « Timings » :



Ce menu est pilotable à l'aide des touches « haut », « bas » et « Enter ». Il permet de choisir une valeur de timing comprise entre 1 seconde et deux heures. On peut aussi désactiver cette fonctionnalité en choisissant l'option « Use PPT Timings ». Si l'on choisit l'option « Cancel », aucune modification n'est faite. Dans la partie supérieure droite, on voit la sélection d'origine à l'ouverture du menu et celle actuelle.

Cette fonctionnalité de Timings est en faite un héritage de la version 1 de l'iPresenter. Celui-ci permettait de faire défiler de manière automatique les diapositives sans intervention de l'utilisateur. Maintenant que les présentations Powerpoint sont gérées de manière native, il est plus simple de définir ces timings automatiques dans les options de transitions des diapositives. La fonctionnalité a cependant été conservée.

Si un timing a été choisi, c'est un simple objet Timer qui est activé au moment de la lecture d'une présentation. Au tick() de ce timer, un clic gauche de souris est simulé, faisant passer à la diapositive suivante. Si l'utilisateur fait défiler manuellement l'une ou l'autre diapositive, le timer est remis à zéro.

Sauvegarde d'une configuration

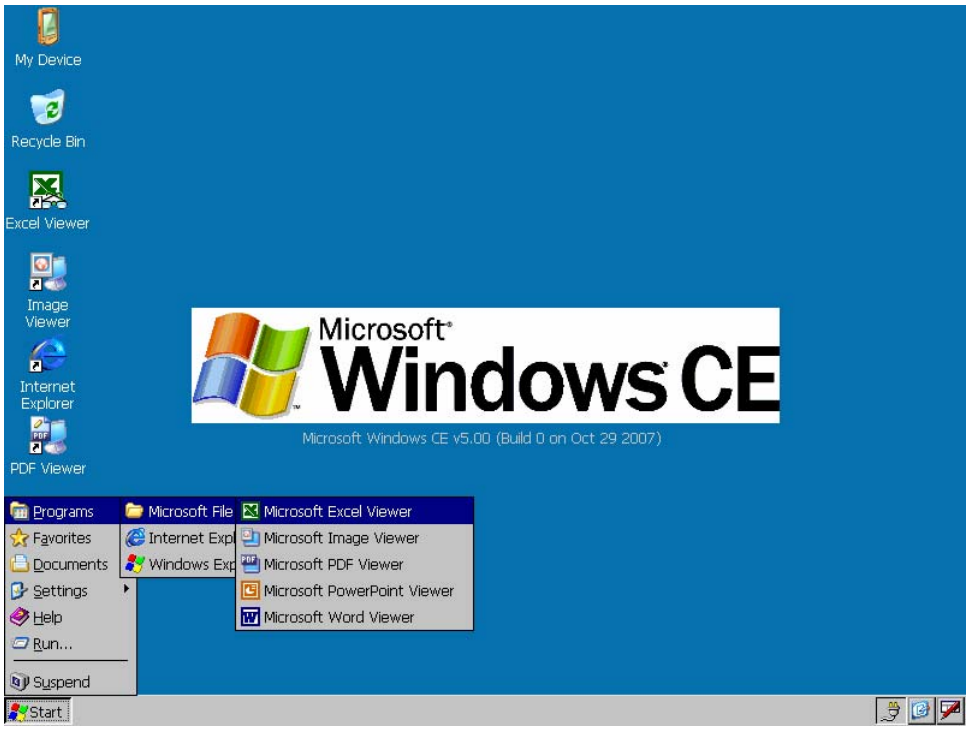
Il serait utile de conserver la sélection faite par l'utilisateur dans ce menu, même lorsque l'appareil est éteint. C'est en partant de cette nécessité-là qu'a été développée la classe « ConfigFile ». Elle permet de sauver dans un fichier différents paramètres. Elle a été écrite de manière à être totalement extensible.

Une structure de données définie dans le programme principal permet de stocker toutes les données de configuration. C'est cette structure qui est passée en paramètre à la fonction SaveConfig() de la classe. Elle s'occupera de placer ces données dans un fichier via les classes dédiées FileStream, StreamReader et StreamWriter de .NET. La fonction LoadConfig() effectue l'opération inverse.

A chaque changement de timing, la configuration est modifiée et sauvée dans la mémoire FLASH (répertoire \user\conf). Cela permet de conserver les paramètres de manière fiable. Pour l'instant le seul élément sauvé est le timing choisi mais on peut très rapidement y ajouter d'autres éléments.

A noter que l'emplacement du fichier de configuration est bien sûr paramétrable et défini comme chaîne de caractère constante au début du programme principal.

18.9. Retour sous l'explorateur Windows CE



L'image ci-dessus représente le bureau de Windows CE lors du démarrage normal de la carte Colibri PXA270 avec la nouvelle image. Une fonctionnalité intéressante serait de pouvoir quitter le menu de l'iPresenter pour se retrouver sur le bureau et ainsi utiliser tous les autres logiciels installés tels que Word, Excel et PDF Viewer.

C'est ce qui se passe lorsque l'utilisateur choisit le bouton « Windows CE Explorer » du menu principal puis appuie sur la touche « Enter ». Le menu principal se cache alors complètement et un écran semblable à celui ci-dessus apparaît.

Désactivation de l'explorer au démarrage

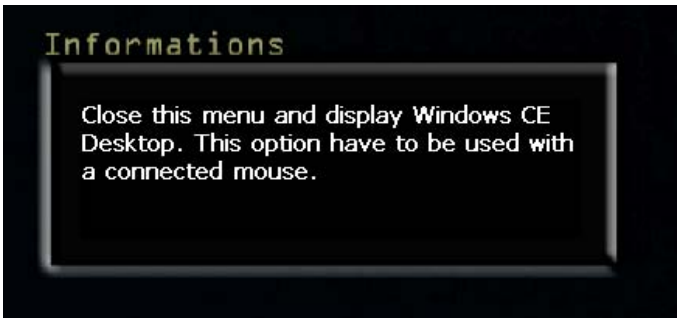
Lorsque la carte démarre, on a tout d'abord un écran blanc sur le PXA270 et une image de Toradex sur le PXA320. Puis vient le tour du chargement du bureau de Windows CE. Enfin, après une éventuelle copie de fichiers depuis la FLASH dans la RAM, l'application du menu principal s'affiche. Cela fait relativement peu propre de laisser à l'utilisateur la vision de toutes ces étapes. Une action simple permet d'affiche le premier écran puis le menu de l'iPresenter2 directement.

Il faut modifier la clé suivant du registre : HKLM\Init\APP00 et remplacer la valeur explorer.exe par _explorer.exe. Après cela, l'explorateur ne se chargera pas par défaut.

Ainsi, notre programme principal doit être au courant de l'état actuel de l'explorateur. Si il n'a encore jamais été ouvert, il ne lui suffit pas de cacher la fenêtre du programme principal, il doit encore lançer un nouveau processus nommé « explorer.exe ». Dans le cas contraire, il cache simplement sa fenêtre principale.

18.10. Zone informative

Voici une des informations que cette zone est capable d'afficher :



Elle permet d'aiguiller l'utilisateur sur les choix qu'il peut faire à un moment précis. Elle permet aussi parfois comme ci-dessus de conseiller l'utilisateur sur l'une ou l'autre option de l'application. Les informations sont mises à jour à chaque fois que l'utilisateur choisit une autre option dans le menu principal.

Toutes les chaînes de caractères qui contiennent ces informations ainsi que celles des messages d'erreurs ou d'information sont définies sous la forme de tableaux de constantes à l'initialisation du programme principal.

18.11. Extinction de l'appareil

La fonctionnalité de ce bouton est simple. Il appelle la fonction de la dll qui va écrire un zéro sur la pin de maintien de l'alimentation. Cela désactive l'ensemble du système. On ne risque rien de couper l'alimentation de la sorte car tout le programme est stocké et ne travaille qu'en mémoire RAM. A chaque démarrage, tout est rechargé depuis la FLASH.

19. Tests du logiciel final

Les DLL

Les fonctions de bas et de haut niveau ont été testées dans tous les cas de figures. Vu que toutes les opérations critiques sont testées après leur exécution, le programme est de toute façon averti des erreurs qui peuvent arriver.

La séparation en plusieurs niveaux, la mise à disposition uniquement des fonctions utiles au programme principal, la création de structures d'accès aux données, la définition de constantes dans une zone séparée et l'absence de valeurs chiffrées dans la plupart du code permet une qualité qui sera sûrement appréciée lors de modifications ou mises à jour du code.

Le programme principal

Basé sur les mêmes principes d'écriture que les dll, il est aussi assez bien structuré. Les structures try{}…catch{}.. aux endroits sensibles permettent d'éviter de plantage du programme dans certains cas. Il est clair que le niveau de sécurisation et de gestion des erreurs du programme principal est loin de celui des dll. Il faudrait passer beaucoup de temps pour en faire un logiciel parfaitement stable et sûr.

Des tests plus poussés suivants une procédure formelle comme celle effectuée sur le matériel de l'iPresenter2 doivent être faits. Le temps imparti ne m'a pas permis de mener à bien cette étape.

20. Bilan

Le meilleur moyen de dresser un bilan sur l'ensemble du projet est d'évaluer le stade actuel par rapport aux objectifs fixés ou plus précisément aux spécifications écrites au chapitre 5.

On peut affirmer que l'électronique de l'iPresenter2 est fonctionnelle. Elle a été testée selon une procédure rigoureuse. La sortie graphique affiche une image stable. Les ports USB fonctionnent et l'on peut connecter tout type de périphériques reconnus, de la souris optique à la clé de données en passant par un disque dur 2.5 pouces. La batterie peut être rechargée et l'autonomie de l'appareil dépasse les deux heures fixées dans la spécification dans tous les cas d'utilisation. Les calculs prennent en compte la baisse de capacité d'environ 20% de la batterie due à son vieillissement. Les alimentations sont stables, les tensions de sorties respectent les tolérances des composants branchés dans tous les cas. Les modules Colibri PXA270 et PXA320 peuvent être utilisés avec certaines limitations pour le PXA320.

Au niveau physique, la carte développée est un prototype est n'est donc pas de taille aussi réduite que celle voulue pour le produit final. Une deuxième version qui prendra en compte et corrigera les erreurs de conception commises lors du design initial devra être imprimée. La schématique est à jour.

L'appareil utilise Microsoft Powerpoint Mobile et gère donc tous ces formats de manière native. Il est à noter que les animations ne sont gérées que partiellement, cela étend du aux limitations du logiciel de Microsoft. La résolution de 800x600 pixels (SVGA) est gérée par défaut de manière fiable, il est même possible d'utiliser les résolutions de 640x480(VGA) et de 1024x768(XVGA). La lecture de fichiers à partir de clés USB est fonctionnelle, sur les deux ports de la carte. Il est possible de connecter un disque dur ou tout autre périphérique de stockage de masse reconnu par Windows CE.

La spécification faisait part de la volonté de pouvoir connecter un stick USB Wifi sur la carte. Le test a été réalisé avec une clé U.S.Robotics USR5422. L'appareil reconnaît l'appareil branché mais ne dispose pas des pilotes nécessaires. Il faudrait trouver un stick avec des pilotes développé pour Windows CE. La gestion des protocoles Wifi 802.11a/b/g ainsi que TCP/IP a été ajoutée à la nouvelle image créée et il devrait être relativement simple de rendre opérationnelle cette fonctionnalité. Elle n'a pas pu être testée dans le cadre du projet.

Une nouvelle image de Windows CE a été portée sur le module PXA270. Elle contient tous les éléments vitaux pour son fonctionnement (pilotes et noyau de Toradex, composants logiciels de bases et logiciels supplémentaires).

Des bibliothèques de liaisons dynamiques gèrent les accès au matériel et met à disposition de l'application principale des fonctions pour gérer celui-ci. L'application principale est écrite en langage évolué et présente une interface riche en couleurs, intuitive et évolutive.

Les objectifs principaux définis au départ sont donc en grande partie atteints. Les éléments manquants sont le portage de Windows CE sur le PXA320 et l'accès aux registres matériels de celui-ci.

21. Suite du projet

L'objectif ultime étant de créer une version commercialisable de l'iPresenter2, il faudra encore passer du temps sur différents points. Voici les plus importants :

Au niveau matériel :

- Recréer une carte électronique de plus petite taille avec un boitier design et adapté
- Effectuer une étude de marché pour contrôler si le prix total des composants, main d'œuvre et marge n'est pas trop élevé. Dans le cas contraire, retravailler les éléments critiques tels qu'alimentation, batterie et connecteurs.

Au niveau logiciel :

- Se procurer le datasheet complet (developper-datasheet) chez Marvell pour le PXA320 et compléter les bibliothèques de liaisons dynamiques pour mettre au même niveau les fonctionnalités des deux modules
- Se procurer le Board Support Package pour Platform Builder du PXA320 afin de pouvoir recréer une image complète à notre guise
- Finaliser et effectuer une procédure de test complète et rigoureuse sur l'application finale en C#. Un travail doit encore être apporté à la sécurisation du code et à la gestion des erreurs
- Régler la problématique des licences pour Powerpoint Mobile

22. Remerciements

J'aimerai remercier en particulier Mr. Pierre Pompili pour son coaching et son aide précieuse durant le projet, Mr. Mladen Stjepic pour le développement du chargeur de batterie, Mr. Pascal Sartoretti pour la commande des pièces ainsi que mes collèges de l'A309 pour leur soutien et leurs conseils.

23. Conclusion

J'ai simplement trouvé ce projet passionnant. Avoir la possibilité de développer une carte électronique complexe, pouvoir étudier en profondeur un système d'exploitation très utilisé dans le domaine mobile et développer des applications dans plusieurs langages plus ou moins évolué dans le même projet est une chance. Il est clair que le temps à investir pour mener à bien les objectifs est grand, mais acquérir des connaissances dans tous ces domaines et très important et me permet aujourd'hui d'orienter ma carrière professionnelle.

Vendredi, le 23 novembre 2007

24. Liste des annexes

- Annexe 01 : Schématique de la carte de test de l'alimentation
- Annexe 02 : Procédure de test de la carte de test de l'alimentation
- Annexe 03 : Schématique de l'iPresenter2
- Annexe 04 : Liste des composants de l'iPresenter2
- Annexe 05 : Procédure de test du matériel de l'iPresenter2
- Annexe 06 : Procédure de test de l'alimentation de l'iPresenter2
- Annexe 07 : Liste des composants logiciels pour la nouvelle image Windows CE 5
- Annexe 08 : Code de la bibliothèque de liaisons dynamiques RegAccess.dll
- Annexe 09 : Code de la bibliothèque de liaisons dynamiques WceFunc.dll
- Annexe 10 : Diagramme UML du programme principal
- Annexe 11 : Procédure d'initialisation, de lecture et d'écritures de données sur l'ISL6295

25. CD-ROM

Le CD-ROM en annexe contient les dossiers suivants :

1_Schematics : Il contient les fichiers de la schématique de l'iPresenter2 et de la carte d'alimentation. Ces fichiers peuvent être ouverts avec PCAD version 2004 ou 2006 d'Altium.

2_PCB : Il contient les fichiers de la schématique de l'iPresenter2 et de la carte d'alimentation. Ces fichiers peuvent être ouverts avec PCAD version 2004 ou 2006 d'Altium.

3_Composants : Il contient les datasheets des composants les plus importants, ainsi que le fichier Excel qui liste toutes les pièces de l'iPresenter2, leurs références de commande ainsi que les prix pour la réalisation d'une carte.

4_Documentation : Il contient ce présent rapport, le schéma d'implantation des composants, une tutorial sur la programmation MFC ainsi que des documents informatifs utilisés durant le projet

5_Software : Il contient toutes les sources et les fichiers binaires pour Windows ou pour le Colibri

1_Colibri_bin : il contient tous les fichiers exécutables utilisés sur le Colibri

2_Developpment : il contient tous les fichiers sources des programmes développés ainsi que l'image d'arrière-plan du menu principal