Alcatel
e-Business Networking Division
1, route du Dr Albert Schweitzer
67408 Illkirch Cedex

Encadrant industriel
Philippe MARTEAU

ISIAL
Faculté des Sciences
BP - 239
54506 Vandoeuvre Cedex

Encadrant universitaire
Alexandre SIMON

Jean-Julien Claudon

Étude et adaptation d'un driver de mémoire Flash

DESS Informatique, option Ingénierie des Réseaux & Systèmes
Stage du 15 avril au 31 août 2002

Remerciements

Je souhaite remercier en premier lieu Philippe Marteau, responsable du stage, pour son accueil et son encadrement durant ces quelques mois, ainsi que Yves Ludwig, responsable de l'équipe, qui a proposé initialement le sujet du stage

Je remercie Alexandre Simon, encadrant universitaire du stage.

Je remercie également l'ensemble du personnel du service qui a toujours été disponible pour répondre à mes questions.

Enfin, je remercie en tengwar tous les stagiaires :

Table des matières

Table des figures

Liste des tableaux

Introduction

Le stage clôturant l'année du DESS d'informatique en ingénierie des réseaux et systèmes peut porter sur un vaste ensemble de techniques et d'objectifs : implantation de protocoles réseaux, systèmes temps-réels, interfaces *web*, sécurité ou encore spécification, développement et intégration logiciel...

Pour ce qui est de moi, du quinze avril au trente août 2002, je me suis rapproché de la partie basse de l'informatique : le matériel. J'ai effectué mon stage au sein de la société *Alcatel Business Systems* sise à Illkirch, à côté de la belle ville de Strasbourg.

Cette société est spécialisée dans les technologies de télécommunication, et conçoit du matériel de pointe, tant en téléphonie qu'en informatique. C'est dans l'optique de contribuer au développement d'un de ces matériels - un autocommutateur privé - que durant les quatres mois et demi du stage j'ai effectué la mission qui m'a été confiée, c'est-à-dire l'étude d'un nouveau pilote pour le principal médium de stockage utilisé, la mémoire Flash

Plus précisemment, cela a consisté en l'étude du pilote de gestion mémoire de *Linux*, son adaptation aux spécificités du matériel et sa comparaison avec le pilote existant en vue d'un éventuel remplacement.

Après avoir donné une vue d'ensemble du contexte du stage en présentant la situation passée et présente de l'entreprise ainsi que l'équipe au sein de laquelle j'ai été, j'introduirai la technologie des mémoires Flash et ce qui existe pour la gérer. La partie suivante montrera le travail réalisé et les résultats qu'on peut en tirer par rapport aux objectifs initiaux

Première partie

Présentation et études préliminaires

Chapitre 1

Présentation de l'entreprise

Dans lequel on expose le contexte industriel et le sujet du stage en lui-même.

1.1 Historique de l'entreprise et situation actuelle

À la fin du XIX^{e} siècle , la Compagnie générale d'électricité (CGE) a été créée dans le but de concurrencer des sociétés telles que Siemens où General Electric. À la fin du XX^{e} siècle , la CGE après de nombreux changements - comme on peut le voir dans le bref historique du tableau 1.1 - prend sa dénomination actuelle : *Alcatel*.

Alcatel devient, en 2001, le numéro un mondial des fournisseurs d'infrastructures de télécoms. Le groupe occupe des positions de premier plan sur ses marchés tels que l'accès rapide à Internet et les réseaux optiques. La vision à long terme et le haut niveau technologique du Groupe lui ont permis d'acquérir les positions qu'il détient aujourd'hui.

Les solutions proposés par Alcatel vont du simple service téléphonique aux réseaux multimédia les plus sophistiqués, et ses clients sont aussi bien les opérateurs de télécoms, les fournisseurs de services Internet, les entreprises que le grand public.

1898	Création de la Compagnie générale d'électricité (CGE).
1925	Absorption de la Compagnie générale des câbles de Lyon.
1966	Absorption de la société alsacienne de constructions atomiques, de télécommunications et d'électronique (Alcatel).
1984	Contrôle par le groupe CGE du holding Thomson Telecommunications.
1985	Fusion de la Compagnie industrielle des téléphones avec Alcatel / Thompson Telecommunications sous le nom d'Alcatel.
1987	Privatisation de la CGE.
1988	Alliance Alsthom / General Electric Company.
1991	La CGE prend le nom d'Alcatel Alsthom.
1995	Serge Tchuruck devient président directeur général d'Alcatel Alsthom, et entreprend de réorganiser la société pour la recentrer sur les télécommunications.
1998	Alcatel Alsthom prend le nom *Alcatel*.

TAB. 1.1 – Quelques points de l'histoire d'Alcatel

Alcatel exerce son activité dans plus de cent trente pays. La répartition géographique de son chiffre d'affaire pour 2001 a été de 51% pour l'Europe, de 19% pour les États-Unis et de 13% pour l'Asie, région la plus dynamique.

D'un point de vue économique, Alcatel se divise en cinq entités :

- *Networking*, qui regroupe les activités Réseaux, mobiles et fixes, incluant les commutateurs et les routeurs. En 2001, Alcatel conserve sa position de leader mondial dans la technologie ADSL et a livré huit millions de lignes.
- *Optics*, qui englobe l'ensemble des solutions optiques. En 2001, Alcatel a dominé à la fois le marché des réseaux optiques terrestres avec 14.2% de part de marché, et le marché des réseaux optiques sous-marins avec 41% de part de marché. Alcatel s'est ainsi affirmé comme le leader mondial du marché global des réseaux optiques.
- *e-Business Group*, qui regroupe les activités de communication d'entreprise et se subdivise en trois : l'*e-Networking Division* (eND) pour les solutions de convergence voix-données, l'*e-Application Division* (eAD) pour le développement d'applications e-business, et la *Mobile Phone Division* (mPD) pour les terminaux mobiles entreprises et grand public.
- *Space and Components*, qui concerne les satellites et leurs composants associés. Elle possède 15% de part du marché des satellites de télécommunications.

– *Nexans*, qui concerne la partie câble d Alcatel et qui a été introduite en bourse en juin 2001.

1.2 Recherche & développement

Le budget 2001 de R&D s'élève à 2,9 milliards d'euros représentant 11,3% du chiffre d'affaire du groupe. L'activité de R&D mobilise 22 000 ingénieurs dans le monde. Alcatel a maintenu en 2001 son effort de recherche et développement à un haut niveau, avec la double ambition de dégager les moyens nécessaires pour consolider ses avantages compétitifs tout en focalisant ses investissements sur des domaines jugés prioritaires pour l'avenir.

Ces orientations se concrétisent dans la priorité accordée aux réseaux métropolitains DWDM (*Dense Wavelength Division Multiplexing*) et aux routeurs-commutateurs optiques, aux réseaux dorsaux large bande, aux équipements d'accès ainsi qu'aux logiciels de gestion de réseaux et de services. Les partenariats, tel celui signé avec Fujitsu pour développer l'infrastructure des réseaux mobiles de troisième génération (UMTS, *Universal Mobile Telecommunication System*) illustrent cette politique.

Par ailleurs, dans un souci de proximité des marchés géographiques en croissance, Alcatel a décidé l'ouverture en Chine du sixième centre de recherche du groupe. Sur des sujets très spécialisés ou des projets de long terme, Alcatel collabore avec plus de cent instituts et centres de recherche universitaires dans le monde. En France, l'INRIA (Institut national de recherche en informatique et automatique) en est le premier partenaire.

1.3 Un groupe en crise

En l'an 2000, la croissance exceptionnelle du marché et de l'engouement général pour les nouvelles technologies ont permis des records. La forte augmentation du chiffre d affaire dans tous les secteurs et l'optimisme général au sujet des nouvelles technologies ont fait bondir le cours de l'action à 94.

Au cœur de la crise des télécommunications, le titre est à peine à 15. La désillusion du WAP, les retards successifs du GPRS et le prix des licences UMTS sont autant de facteurs ayant alimenté cette crise. Les opérateurs, premiers clients d Alcatel, étant par conséquent surendettés, n'investissent plus. Les équipementiers eux-mêmes ont également perdu beaucoup dans la course à l'acquisition de *start-up*.

Dans ce contexte de crise, la direction prend des décisions majeures :

- Le 27 avril 2001, la direction a annonçé l'externalisation de l'ensemble de la production de téléphones portables. Deux usines sont concernées : celle de Laval vendue à Flextronics, et celle d'Illkirch réorientée vers la production de composants optiques.
- Le 26 juin 2001, dans le Wall Street Journal, Serge Tchuruck, PDG d'Alcatel, a déclaré «Nous souhaitons être très bientôt une entreprise sans usine». D'ici la fin 2002, le nombre d'usines détenues par le groupe devrait passer de 120 à 12.
- En février 2002, des pertes de plus de 5 milliards d'euros ont été annoncées pour 2001.
- Le 25 avril 2002, Serge Tchuruck dit à propos du site d'Illkirch «On réduit les emplois, c'est en cours. Il n'y a pas de fermeture annonçable mais l'activité s'est réduite d'un quart en un an».

Les effectifs totaux du groupe devraient passer de 99000 fin décembre 2001 à 83000 fin 2002, soit une baisse de 19.2%. Le prix de l'action chute, les plans de restructurations se succèdent et les avertissements sur les résultats se multiplient : Alcatel est en pleine tourmente. Le manque de visibilité du marché associé à l'absence de transparence des intentions de la direction rendent l'avenir de l entreprise incertain.

1.4 Le site d'Illkirch

L'établissement d'Illkirch (voir photo en figure 1.1) est un pôle de marketing, R&D et industrialisation ainsi que de services opérationnels au sein de l'*e-Business Group*. Il compte environ 1600 personnes et a la charge de concevoir des systèmes de communication d'entreprises et des terminaux professionnels.

Son activité principale, jusque avril 2001, était la production des terminaux GSM. Une réorientation vers la production de sous systèmes optiques développés par Alcatel Optronics prévoyait de déplacer totalement la production GSM à l'établissement Flextronics Laval pour juin 2002.

Cependant, aujourd'hui, la reconversion du site de production dans la fabrication de sous-ensembles optronics paraît incertaine. En effet, la croissance du marché prévue par Alcatel optronics en avril 2001 dernier ne s'est pas concrétisée, et son chiffre d'affaire prévisionnel 2002 a été divisé par trois

FIG. 1.1 – Photo du site d'Illkirch

par rapport à 2001. Même si les composants optiques restent une activité prometteuse, la prévision de retour aux perspectives 2002 est a priori repoussée en 2004, et sans production certaine. C'est donc la pérennité même du site qui est aujourd'hui en jeu.

1.5 Le département SME et l'équipe infra

Le département de recherche & développement SME (*Small and Medium Entreprises*) travaille, comme son nom l'indique, pour une clientèle de petites et moyennes entreprises (PME). Ce département se divise en services dédiés à l'architecture, le développemtn des logiciels, la gestion des outils, la validation et enfin la partie matérielle.

Ce départment a la charge des solutions de communications pour le marché des PME, qui sont basées sur la ligne de produits Alcatel Office. Cette ligne est constituée des autocommutateurs 4200 C/D/E et de l'OmniPCX Office, appelé en interne Alizé. Les membres du département SME ont aussi à charge :
- les systèmes d'exploitation des autocommutateurs,
- les logiciels de configuration

– les application CTI (*Computer & Telephony Intergration*, c'est-à-dire l'intégration voix-données)

C'est au sein de ce département que j'ai effectué mon stage, et plus particulirement au sein de l'équipe infrastructure. Celle-ci s'occupe de la plateforme Alizé, *i.e.* l'autocommutateur privé OmniPcx. Elle doit fournir une plateforme générique de type Linux pour toutes les applications embarquées (gestion de la téléphonie, voix sur IP, accés internet, ...). Pour cela, le personnel de l'équipe adapte et fait évoluer la plateforme matérielle et l'environnement Linux standard, et s'occupe aussi d'un certain nombre des applications - ce qui en fait une équipe dont le domaine de compétence est très large, et qui est fortement sollicitée par les autres équipes.

1.6 La plateforme Alizé

La plateforme sur laquelle j'ai travaillé est l'Alizé. C'est un autocommutateur privé (PABX, *Private Automatic Branch eXchange*) qui a donc pour fonction principale de commuter et router les appels téléphoniques. Ce type de matériel permet en outre diverses fonctionnalités supplémentaires selon les ajouts de cartes additionnelles, permettant de gérer par exemple la messagerie vocale, l'accès internet ou la voix sur IP, ...Il s'agit au final d'un équipement «tout en un».

Au niveau matériel, la plateforme est architecturée autour d'un processeur AMD SC520, dit Aspen, à 100 ou 133 MHz, ou d'un processeur PIII à 700 MHz. Le BIOS (*Basic Input/Output System*) est contenu dans une mémoire Flash NOR de 512ko et le système d'exploitation est stocké dans une mémoire Flash NAND de 32Mo, puis chargé en mémoire vive, 32 à 256 Mo de SDRam étant disponibles selon le modèle. Sur la carte principale - appelée carte *main* - on peut ajouter une carte fille possédant de la mémoire Flash supplémentaire. Cette carte est appelée *XMem* et sa présence sera à prendre en compte à un moment donné du travail d'adaptation du pilote...

La plateforme logicielle associée évolue constamment pour suivre les corrections de bugs, ajouts de fonctionnalités ou optimisations des traitements, ...Elle est basée sur un Linux modifié temps réel : RT/Linux 2.0. La version du noyau actuellement en cours d'utilisation est 2.4.18. Ce noyau a été modifié à de nombreux endroits pour correspondre aux besoins matériels et logiciels.

1.7 Sujet du stage

Comme je l'ai mentionné en 1.6 à propos de l'Alizé, le système d'exploitation est stocké dans une mémoire Flash NAND avant d'être chargé en mémoire vive. Pour accéder à cette mémoire Flash, un pilote est utilisé. Celui-ci a été développé par la société DataLight et acheté par Alcatel en *Buyout*, c'est-à-dire en l'état, avec possibilité de modification, de développement et d'intégration au sein des produits Alcatel sans reversement de *royalties* - mais aussi sans support ni maintenance de la part de DataLight. Encore très récemment, des corrections ont dû être apportées à ce pilote après que des problèmes de sécurité ont été soulevés.

De même que précédemment le driver FlashFX avait remplacé celui utilisé alors - à savoir un produit de la socitété M-Systems -, il y a aujourd'hui une étude comparative à réaliser pour déterminer si un éventuel changement est possible…

Avec la série 2.4.x du noyau Linux, il y a une gestion générique des technologies de mémoires - les *Memory Technology Devices* ou MTD - inclue et maintenue. Il est donc dans l'intérêt d'Alcatel de savoir si le pilote de mémoire Flash des MTD peut être adapté avantageusement pour leur système, en vue d'un remplacement de celui existant.

L'objectif du stage était défini en deux temps : tout d'abord réaliser l'adaptation propre au matériel pour faire fonctionner le pilote Linux sur la plateforme Alizé, et ensuite apporter des éléments de comparaison avec le pilote actuel.

Chapitre 2

Présentation de la technologie Flash

Dans lequel on montre certaines des particularités de cette mémoire non-volatile de faible coût.

2.1 Vue d'ensemble

La mémoire Flash est une mémoire non-volatile, ce qui signifie que les données inscrites restent rémanentes même en l'absence de toute source d'énergie. Leur robustesse, leur faible consommation d'énergie en cours d'utilisation et leurs dimensions réduites en font un support très bien adapté aux mobiles par exemple, et aux systèmes embarqués en général.

2.2 Aspect matériel

Bien que le détail de l'aspect matériel de ce support ne soit pas fondamental pour la suite, la lecture de ce point permettra de comprendre d'où viennent certaines particularités du médium dont il faudra tenir compte.

La mémoire Flash est un type de circuit EEPROM (*Electrically erasable programmable read-only memory*). Elle fonctionne avec une grille de colonnes et de lignes ayant deux transistors à chaque intersection. Ceux-ci sont séparés par une mince couche d'oxide. Un de ces transistors est dénommé «porte flottante» et l'autre «porte de contrôle» [1]. La porte flottante est uniquement liée à la ligne par la porte de contrôle. Tant que ce lien est en place, la cellule a pour valeur 1, comme l'illustre la figure 2.1. Pour passer cette valeur à zéro, il existe le processus du *Fowler-Nordheim tunneling.*

Il s'agit d'utiliser une charge électrique pour faire agir la porte flottante comme un canon à électrons, et ainsi faire passer les électrons excités de l'autre côté de la couche de séparation. On obtient alors une charge négative qui fait barrière entre la porte flottante et la porte de contrôle.

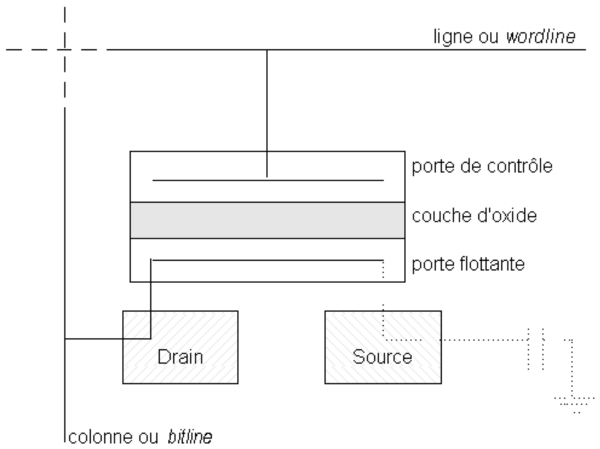

FIG. 2.1 – État d'une cellule de la Flash ayant la valeur 1

Un testeur de cellules (*cell sensor*) relève les charges qui passent par la porte flottante, et si le flot qui passe à travers la porte est au dessus d'un seuil de 50% la valeur de la cellule est considérée comme étant à 1, mais si les charges négatives dépassent le seuil, la cellule vaudra zéro, comme illustré en figure 2.2. Un support vierge aura toutes ses portes ouvertes, *i.e.* chacune de ses cellules à 1.

L'effacement se réalise en appliquant un champs électrique à toute la Flash, ou à une des parties prédéterminées appelée *bloc*. Le fait de permettre un retour des bits à 1 par bloc est le grand avantage de la mémoire Flash par rapport à la technologie EEPROM dont elle dérive, laquelle souffre d'une trop grande lenteur, devant agir bit à bit.

[1] respectivement *floating gate* et *control gate*

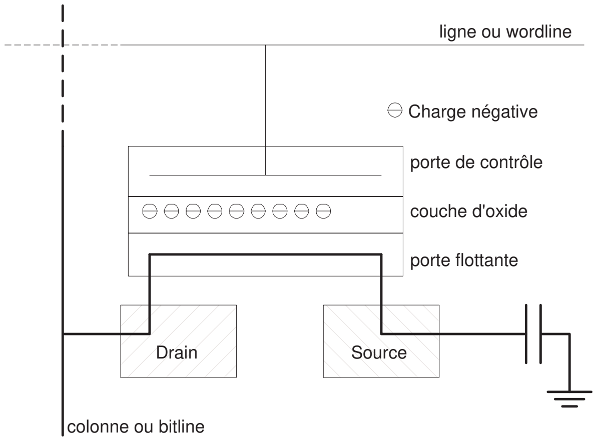

FIG. 2.2 – État d'une cellule de la Flash ayant la valeur 0

2.3 Les deux visages de la Flash

Il y a deux types de Flash[2] qui diffèrent sur plusieurs points : les Flash NOR et NAND. C'est sur le second type qu'a porté mon travail.

La différence technologique principale entre les deux est la manière d'accéder aux données. Alors que la Flash NOR propose un bus pour l'accès direct aux données, la NAND utilise une interface d'entrées/sorties à accès séquentiel. Cela implique par exemple que le code stocké sur une Flash NAND ne peut pas être exécuté directement depuis celle-ci, mais doit au préalable être placé en mémoire vive. Ce type est donc plus approprié pour le stockage de masse [3].

Le choix d'utiliser la mémoire de type NAND est dans la plupart des cas lié à son faible coût, qui justifie le fait de s'accommoder des lenteurs au niveau des opérations de lecture et d'écriture. Les points de comparaisons entre ces mémoires sont repris dans le tableau 2.1.

2.4 Les composants de la Flash

Je présente le diagramme des composants de la Flash, regroupés en blocs logiques en figure 2.3. Cette section a pour but de montrer l'organisation

[2] J'utiliserai l'appellation Flash plutôt que mémoire Flash par la suite
[3] stockage de données par opposition au stockage de code exécutable

	FLASH	
	NOR	NAND
Interface	Bus	I/O
Taille	Grande	Petite
Coût	Fort	Faible
Vitesse de lecture	Rapide	Lent
Vitesse d'écriture	Rapide	Lent
Vitesse d'effacement	Lent	Rapide
Consommation d'énérgie	Forte	Faible
Exécution depuis la Flash	Oui	Non

TAB. 2.1 – Quelques points de comparaison entre NOR & NAND

interne faisant le lien entre les points d'accès disponibles pour le contrôle de la Flash et la grille de cellules où sont stockés physiquement les valeurs des bits.

Le rôle de chacun des blocs logiques qu'on trouve dans la Flash est :

- *I/O control* & *Logic control* gérent les informations en entrée et en sortie des broches respectives de la Flash ;
- *Control circuit* effectue les opérations de lecture et de programmation en fonction des données, et en règle générale s'occupe du traitement en cours, comme la génération de tensions ou les indications de disponibilités et de fin de traitement ;
- *Registers* : registres d'adresse, de commande et de status. Le registre d'adresse, indique la ligne et la colonne dans la grille ;
- *Memory cell array* : grille de cellules.

À gauche du schéma en figure 2.3, on peut voir les broches qui seront utilisées pour interagir avec la grille de mémoire. Le rôle de chacune des broches est le suivant :

- *I/O 1..8* : ces broches correspondent aux bits du port d'entrée/sortie ;
- *CE* indique si le bus est utilisé pour envoyer une commande (*Command Enable*). Dans le cas contraire, cela signifie que le dispositif conduit des données. À noter que cette broche est inversée, c'est-à-dire qu'une valeur positive indique *Command Disable* et non *Enable* ;
- *ALE/CLE*, respectivement pour *Address Latch Enable* et pour *Command Latch Enable*. Dans le cas où le port n'est pas utilisé pour les données, cette broche indique si les entrées/sorties concernent des adresses ou des commandes ;

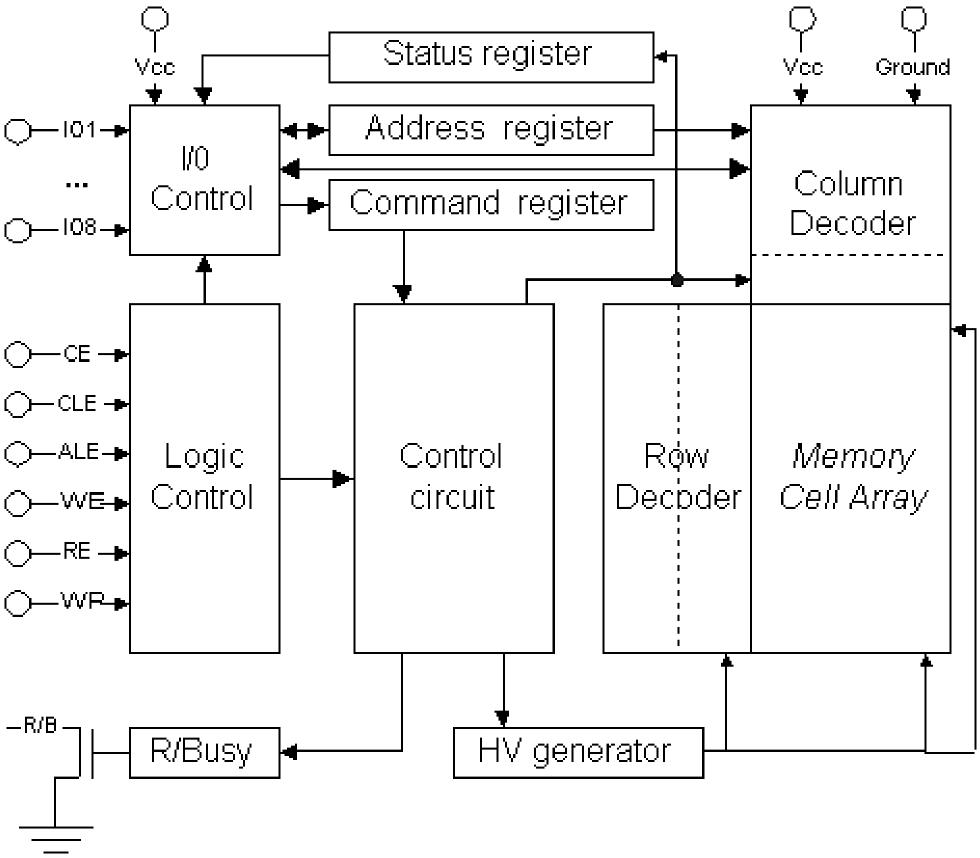

FIG. 2.3 – Diagramme des blocs logiques constitutifs de la Flash

- *WE/RE*, respectivement pour *Write Enable* et *Read Enable* ;
- *WP* indique la protection en écriture, *Write Protect* ;
- *R/B* est le signal *Ready/Busy* permettant de savoir si la Flash est occupée par un traitement ou disponible ;
- *Vcc* & *Ground* sont simplement la source d'énergie et la masse.

2.5 Spécificités du médium

Ces spécificités découlent de la technique utilisée pour le stockage des données dans la grille de cellules de la Flash, comme présenté en section 2.2.

La gestion des données présentes sur la Flash est compliquée par certaines contraintes inhérentes à la mise en œuvre de cette technologie. La contrainte la plus importante étant qu'on ne peut écrire au dessus des données existantes qu'après une opération d'effacement.

Ce problème s'accompagne de l'obligation de réaliser l'effacement par blocs entiers de taille fixe - et de taille relativement importante, typiquement 64ko - ce qui implique en outre de recopier toutes les autres données du bloc effacé, y compris celles des pages voisines n'ayant pas de rapport avec la mise à jour en cours.

Le nombre d'effacements possibles pour un bloc n'est pas infini. En général, un bloc ayant subit un million de cycles d'écriture/effacement - voire beaucoup moins - devient de plus en plus sujet à des erreurs d'écriture jusqu'à n'être en fin de compte plus du tout effaçable. Il faudra donc penser à utiliser de manière uniforme les blocs de la Flash pour augmenter sa durée de vie - ce qu'on appelle le *wear levelling*.

2.6 Conclusion

Les avantages de la mémoire Flash en font un support apprécié. Cependant, la technique sous-jacente permettant d'obtenir les qualités du médium s'accompagne de contraintes d'utilisation particulières.

Les drivers doivent intégrer dans leur design la gestion de ces contraintes pour fournir aux couches supérieures une méthode simple d'accès au médium.

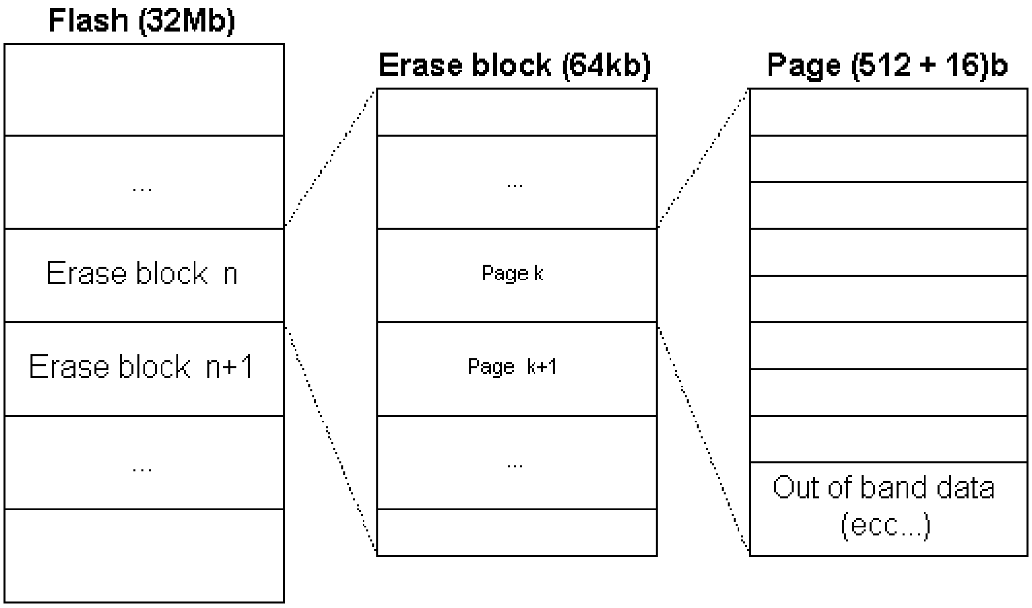

FIG. 2.4 – Structure physique des données sur la Flash

Chapitre 3

Présentation des drivers existants

Dans lequel on découvre les architectures et les différents niveaux d'abstractions des deux pilotes existants.

3.1 Rappels

Avant de parler des drivers existant, je présente quelques notions de base utiles par la suite. Les rappels portent sur ce qu'est un pilote de périphérique sous Linux, sur une vue superficielle des systèmes de fichiers et sur les modules du noyau...

3.1.1 Les pilotes

Le rôle primordial des pilotes au sein du noyau Linux est l'accès au matériel, mais un rôle plus global est de faire la transition entre les applications (le code utilisé pour le contrôle d'un matériel étant partagé par ceux qui l'utilisent) qui font la demande d'un appel système, et le matériel. Les pilotes peuvent se regrouper en catégories selon le type de matériel auquel on cherche à accéder et les services proposés.

Il y a deux types de drivers, liés à deux types de périphériques :

Les périphériques en **mode caractères**, ou *char devices*, sont ceux dont l'accès est réalisé similairement à l'accès à un fichier. Les fonctionnalités pro-

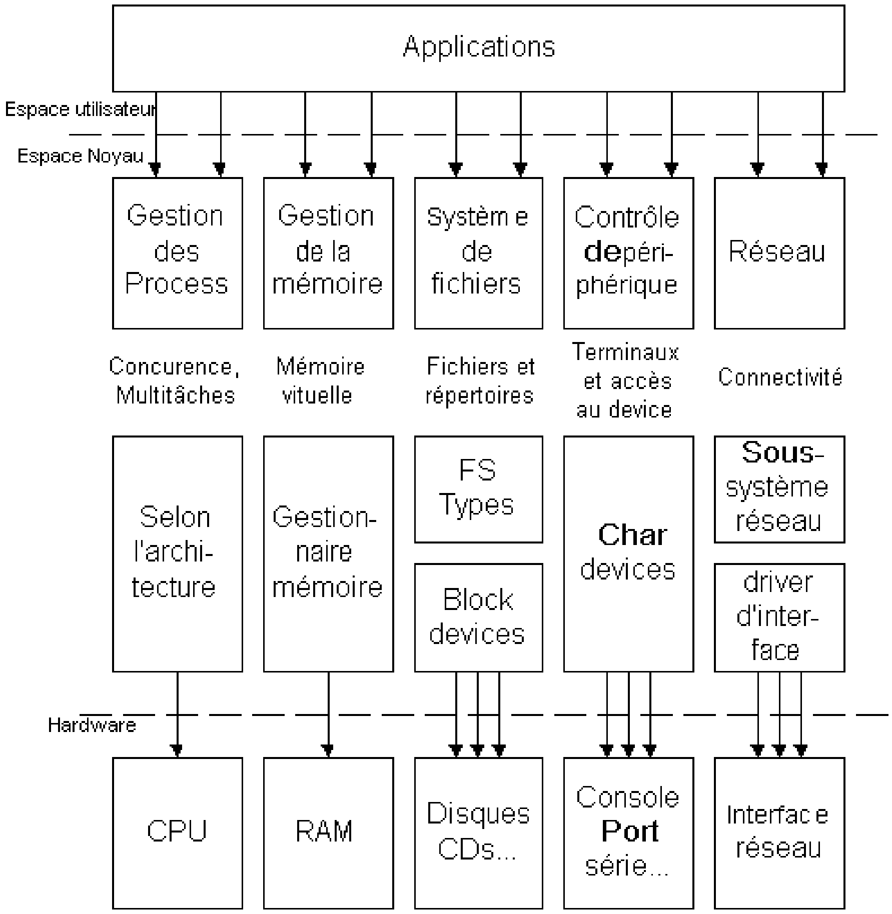

FIG. 3.1 – Les drivers au sein du noyau Linux : on peut voir, de haut en bas, la partie du noyau concernée, les fonctionnalités implementées, les parties logicielles et matérielles. Aux extremités se trouvent les applications et le matériel en soit.

posées reflètent ce comportement et regroupent généralement les opérations d'ouverture, de fermeture, de lecture et d'écriture.

Ils sont la plupart du temps adaptés aux matériels simples, et sont plus facile à comprendre que ceux en mode bloc par exemple. Parmi les périphériques de ce type, on peut citer entre autre la console ou le port parallèle

Les périphériques en **mode bloc**, ou *block devices*, quant à eux sont ceux qui peuvent supporter un système de fichier. Dans la plupart des systémes Unix, on ne peux y accéder que par bloc, *i.e.* par une tranche de taille fixe - un bloc étant souvent 1ko de données.

Que ce soit pour les périphériques en mode bloc ou en mode caractère, il faut passer par une phase d'enregistrement auprès du système pour être pris en compte par la suite et permettre l'appel aux fonctionnalités. Lors de l'insterion du module, la fonction d'initialisation utilise register_blk_device ou register_chr_device pour le faire, comme on peut le voir en figure 3.2 pour un *block device*.

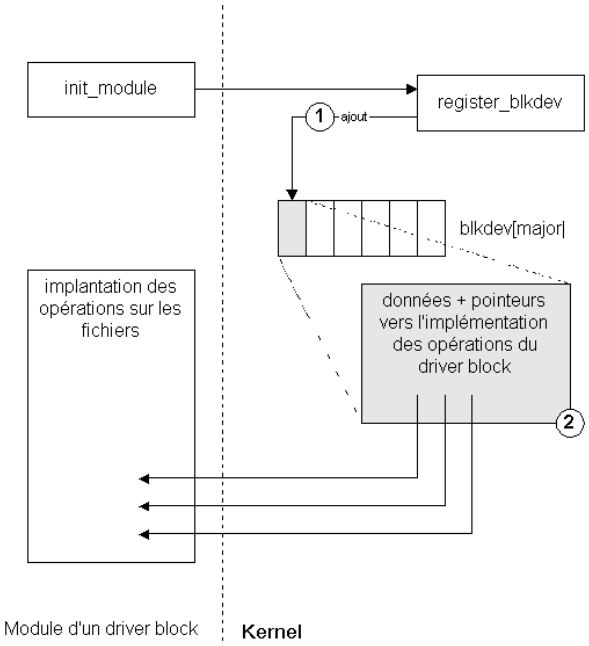

FIG. 3.2 – Exemple d'enregistrement d'un périphérique en mode bloc

La différence entre *block devices* et *char devices* tient essentiellement dans la manière dont les données sont gérées par le noyau, donc par l'interface logicielle liant le noyau et le pilote :

- un périphérique bloc est accédé par un nœud du système de fichier, les différences entre nœuds étant transparentes aux niveaux supérieurs ;
- en mode caractères, il n'y a pas de mise en tampon (*buffering*) des données alors qu'en mode bloc il y a utilisation d'un cache.

3.1.2 Les systèmes de fichiers

Sous Linux, tous les fichiers sont accédés via un système de fichier virtuel, *Virtual Filesystem Switch* ou VFS. Il s'agit d'une couche qui implémente les actions génériques des différents systèmes de fichiers pour amener à l'exécution du code approprié au traitement de la requête. Les pilotes de prériphériques se placent tout comme les systèmes de fichiers en aval de VFS dans le sens où ils correspondent à un type de fichier spécial (*device special files*) parmi les autres types de fichiers (réguliers, répertoires, liens symboliques…).

Écrire un pilote de périphérique sous Linux revient à écrire un ensemble de fonctions et à s'enregistrer auprès de VFS. Tous les périphériques contrôlés par un même driver vont être référencés en tant que fichier spécial qui porte un nombre majeur le liant au driver, et différenciés au sein de ce majeur par un nombre mineur[1].

Le système de fichier qu'on trouve le plus abondamment sous Linux est *ext2* (*Second Extended File System*). C'est aussi avec ce système *ext2* que fonctionne l'arborescence de fichiers au sein des PABX, en incluant de plus la possibilité d'avoir le système de fichiers compressé.

Il existe aussi des systèmes de fichiers journalisés, comme *ext3* par exemple. Parmi ceux-ci, JFFS & JFFS2[2] sont intéressants dans la mesure où ils ont été spécifiquement développés en vue d'être utilisé au-dessus des Flash.

Le principe consiste à ce que le système de fichiers soit une liste de nœuds, chacun contenant des informations à propos du fichier dont il fait partie, que ce soit le nom du fichier, des données de celui-ci ou d'autres informations nécessaires. Dans le cas où le nœud contient des données, il indique en outre la position dans le fichier où ces données doivent apparaître - cela permet l'écriture par dessus les anciennes données. On peut de même indiquer combien de données on souhaite supprimer du fichier, pour avoir la possibilité de le tronquer.

[1] *major and minor number*

[2] JFFS a été développé par *Axis Telecommunications AB* et JFFS2, qui se base sur le premier, est poursuivi par *Red Hat*.

La reconstruction des fichiers se fait en construisant la carte à partir de tous les nœuds et en appliquant les modifications successives. L'exemple simple en figure 3.3 montre les nœuds recomposant un fichier dont une partie du contenu à été remplacé.

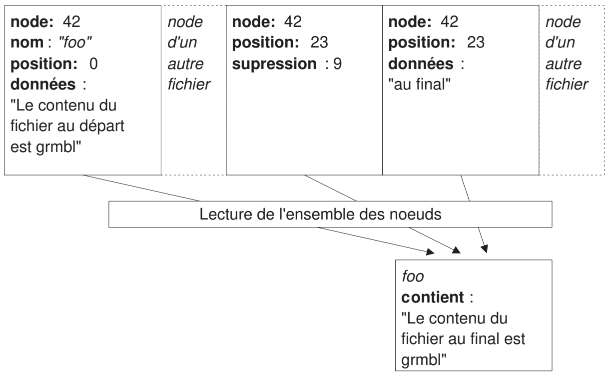

FIG. 3.3 – Exemple de fichier vu par un système à structure de journal

3.1.3 Les modules noyaux

Un module kernel est une partie des fonctionnalités du noyau (avec les privilèges qui s'y rapportent) qui n'y est pas obligatoirement inclue dés le démarrage, mais qui peut être chargée ou déchargée par la suite par l'appel des commandes *insmod* et *rmmod*.

La plupart des pilotes se présentent sous la forme d'un ou plusieurs modules, chacun apportant au sein du noyau un ensemble de fonctionnalités. Outre le fait de rendre disponible aux autres programmes ces fonctions, le chargement d'un module est une phase durant laquelle des initialisations et demandes de ressources, telles des demandes d'IRQ (canal d'interruption) ou de plage d'I/O (zone d'entrées/sorties), peuvent être accomplies.

3.2 Le driver FlashFX

3.2.1 Vue d'ensemble

Le driver FlashFX a été développé par la société *DataLight, Inc.* spécialisée dans les logiciels pour systèmes embarqués. La première version a été lancée en 1993. Le pilote utilisé à Alcatel sur les Alizés actuellement est un driver FlashFX 4.0 modifié. Celui-ci a été acheté en *buyout.*

L'achat en *buyout* signifie qu'Alcatel peut utiliser le code et les sources du pilote, le modifier et l'inclure sur les produits destinés à la vente sans avoir à payer quoi que ce soit de supplémentaire. Cette question des *royalties* est un aspect non négligeable dans le choix de l'utilisation de ce pilote plutôt qu'un autre.

Ce pilote a pour but d'offrir les méthodes d'accès standards aux applications, de manière transparente. Il est aussi conçu pour être portable à tout type de processeur et de systèmes d'exploitations - par exemple DOS, QNX, Linux et Windows CE...

Il permet aux applications de voir la Flash comme un disque accessible en lecture et en écriture. En ce sens, il ne fournit pas le niveau de service d'un système de fichiers mais une émulation d'un accès par secteurs sur lequel un *filesystem* (FS) peut être placé.

Ce driver a été choisi en concurrence avec d'autres. Il remplace le driver précédent, celui de la société M-Systems. Le choix a aussi porté à la base sur le fait d'utiliser des Flash NAND directement ou une des autres solutions matérielles à base de Flash disponibles, comme le *DiskOnChip* qui intégre de la correction d'erreur matérielle, *i.e.* n'ayant pas à être gérée logiciellement, ou encore la solution de type *SmartMedia* qui fournit un dispositif et une connectique permettant d'utiliser directement le support Flash de la même manière qu'un disque IDE. Des raisons - outre celles purement techniques - de coût et de pérennité ont conduit au choix d'utiliser les mémoires sans enrobage, laissant ainsi toute latitude quant à la façon de la gérer, ce qui n'aurait pas été le cas avec une solution matérielle propriétaire.

3.2.2 Architecture

La hiérarchie générale des couches du driver FlashFX est présentée en figure 3.4. L'accès au matériel depuis le système d'exploitation (ou depuis le BIOS) se fait en traversant les différents niveaux d'abstractions.

Je présente les différentes couches mises en œuvre dans l'accès au support matériel de plus bas niveau en présentant la ou les fonctionnalités remplies par chacunes.

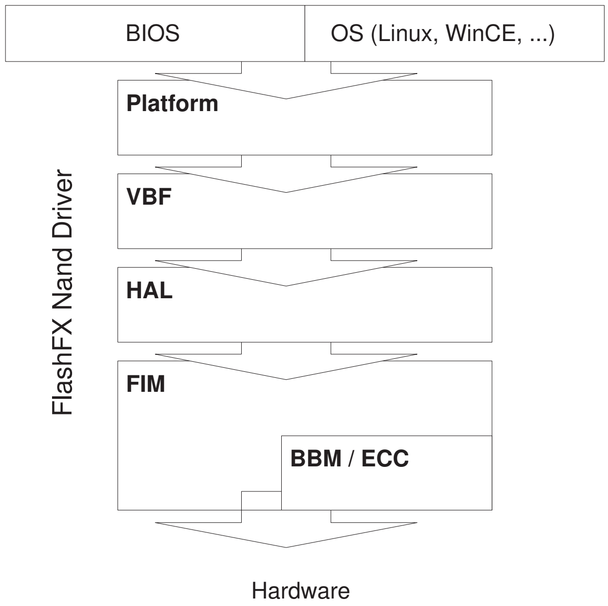

FIG. 3.4 – Architecture des composants de FlashFX

Platform La couche *Platform* est celle avec laquelle le système à affaire. Elle consiste à transformer des appels dépendants du système d'exploitation en des appels indépendants. Tout ce qui se trouve en deça de cette couche est transparent pour les applications ou pour le système.

Variable Block Flash (VBF) Cette couche a pour rôle de placer et d'aller chercher les données sur la Flash en garantissant la consistance des données, *i.e.* en fournissant un service protégé contre une coupure brutale ou un arrêt du CPU. Le VBF voit la Flash comme un ensemble de région d'environ 500ko, composées d'unités. Une unité, qui est la forme la plus simple de stockage, est formée d'un en-tête, d'un certain nombre de pages et d'une liste d'allocation.

Hardware Abstraction Layer (HAL) Il s'agit d'une simple interface définie pour faciliter la personnalisation par rapport à un matériel spécifique. Ses tâches sont l'initialisation matérielle, les algorithmes de lecture/écriture et d'autres besoins spécifiques au *hardware*.

Flash Interface Module (FIM) Ce module défini les caractéristiques d'une Flash particulière. Il doit pouvoir fournir des informations comme la taille des zones effaçables et des méthodes comme l'algorithme de programmation et d'effacement.

Bad Block Managment (BBM) & **Error Correction Code** (ECC) Cette partie du code s'occupe de masquer aux autres niveaux les blocs de la Flash qui sont détectés comme étant erronés. Je décrirai plus en détail le fonctionnement de cette partie dans le point 3.2.3, mais pour résumer son principe, on peut dire qu'il fournit trois procédures :

1. appliquer un code de correction aux données,
2. transférer les données d'un mauvais bloc venant d'être détecté vers un bloc valide,
3. faire en sorte que l'accés à un bloc ayant été transféré soit bien routé vers sa nouvelle position.

Deux autres points font partie des mécanismes intrinsèques de l'architecture du driver : la collecte de blocs obsolètes (*Garbage Collection*) et l'uniformisation de l'utilisation du support (*wear levelling*).

Une des contraintes vue précédemment en 2.5 impose de ne pas écrire et réécrire toujours au même endroit au sein d'un bloc, comme il est possible de le faire avec un disque dur conventionnel. Au lieu de cela, quand un bloc doit être réécrit, on place le nouveau contenu à un nouvel endroit de la Flash. Une table maintient le marquage des blocs valides et de ceux obsolètes. Il arrive un moment où une zone d'effacement contient un excés de blocs invalides, et doit effectuer la collecte de cet espace pour permettre des écritures futures, d'où le mécanisme de *Garbage Collection*.

Le processus consiste à copier les données valides de la zone d'effacement dans un zone précédemment liberée, puis à effacer la zone venant d'être collectée, qui devient alors disponible pour recevoir des données valides par la suite. La gestion des informations s'y rapportant se fait au niveau de la couche *Variable Flash Block*, conçue - et brevetée - par DataLight pour améliorer les performances de ce principe.

Une seconde contrainte est liée au fait qu'une zone de la mémoire Flash ne peut être impliquée que dans un nombre limité de cycle écriture/effacement. Typiquement, l'ordre de grandeur de cette limitation est de 100 000 cycles, mais on peut trouver des variations allant de 10 000 à 1 000 000… Pour augmenter la durée de vie de la Flash, il est parfois nécessaire de déplacer des données qui ne sont pas concernées par un changement, de manière à uniformiser l'utilisation du médium. Ce procédé est appelé *wear levelling*.

3.2.3 Gestion des blocs erronés

Le but du *bad block managment* est de faire apparaître aux couches supérieures le support comme exempt d'erreurs. L'accès a une partie de la Flash doit se faire du point de vue des niveaux supérieurs de la même manière lors d'un accès sans problème de détection d'un mauvais bloc que lors d'un accès ayant conduit à traiter l'erreur.

Le principe est d'effectuer la recopie des informations de tout le bloc fautif, en ayant entre temps corrigé la page qui a déclenché l'erreur. La copie des informations est placée à un endroit réservé de la Flash, comme montré en figure 3.5. Il faut noter que cette place réduit d'autant la capacité de la Flash, or l'espace disponible est une ressource qui fait parfois défaut sur les PABX pour embarquer de nouvelles évolutions du système...

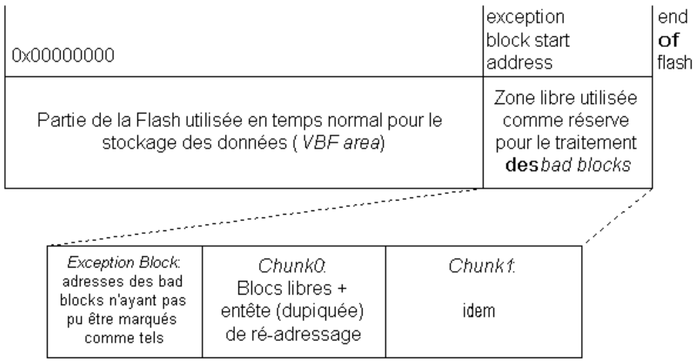

FIG. 3.5 – Zone réservée par FlashFX pour la gestion des blocs erronés

Il est nécessaire de réserver cette place car cette gestion doit toujours pouvoir se faire, le bloc marqué comme mauvais n'étant plus accédé par la suite car on garde la trace de ces blocs et de leur nouvel emplacement. Un exemple de ce comportement est visible en figure 3.6

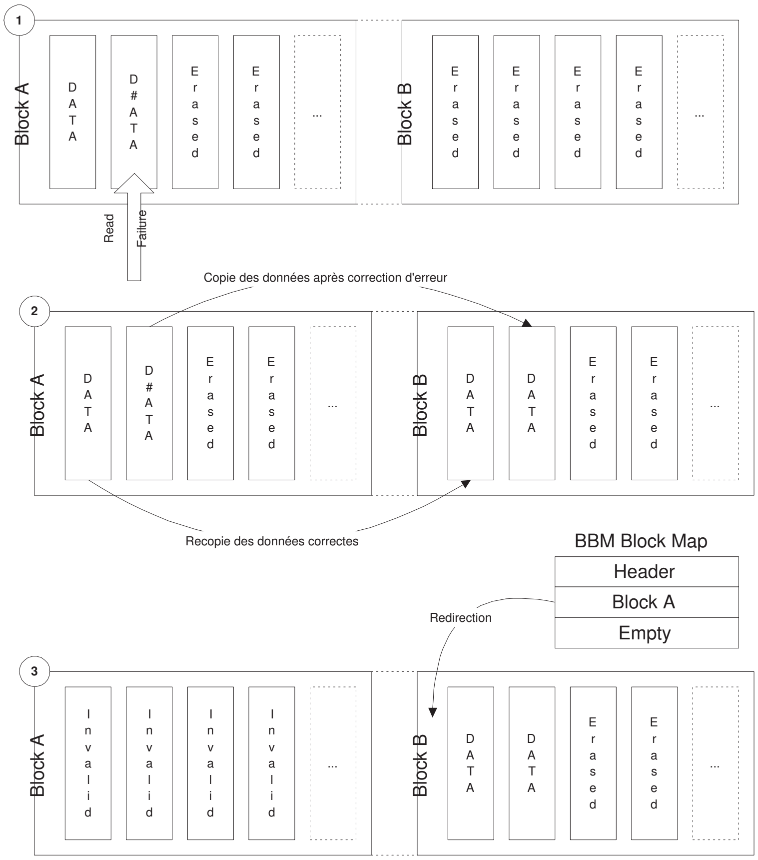

FIG. 3.6 – Exemple de gestion d'un mauvais bloc détecté sur une lecture

3.3 Le driver Linux

3.3.1 Présentation

À partir de la série 2.4.x du noyau Linux, parmi les autres drivers disponibles, il se trouve l'ensemble des *Memory Technology Devices* (MTD). L'objectif des concepteurs et des contributeurs de cette partie du noyau a été de fournir un sous-système générique pour tous les différents périphériques de mémoire. Cela dit, comme on peut le lire en introduction du site présentant les MTD (voir la référence [MTD]), la principale cible était les mémoires Flash.

Tout comme FlashFX, le pilote Linux est conçu de manière à fournir une interface générique entre le driver de plus bas niveau (l'accès au matériel en tant que tel) et les couches supérieures du système. Ceci simplifie l'introduction de nouveaux drivers spécifiques.

La partie la plus basse n'a besoin d'aucune connaissance concernant le format de stockage utilisé, comme le système de fichier mis en place par exemple, et ne va s'occuper que de proposer les opérations de bases pour la lecture, l'écriture et l'effacement.

3.3.2 Architecture

La figure 3.7 présente l'architecture des *Memory Technology Devices*. Les tâches attribuées à chacun des composants en relation avec la Flash NAND[3] sont les suivantes :

- **mtdcore** : Le «cœur» des MTD ne contient pas énormément de fonctions, mais un ensemble de structure communes, les fonctions d'ajout et supression de *mtd devices*, et le procédé de création et de traitement de l'entrée `mtd` dans `/proc`. Pour la mise en place d'un système de notfications, il fournit aussi le (dés)enregistrement de clients auprès d'un device ;
- **mtdchar** et **mtdblock** : Ces deux modules proposent les fonctionnalités propres respectivement aux pilotes en mode caractère (lecture, écriture et positionnement comme dans un fichier,…) et en mode bloc (ouverture et relâchement du périphérique, lecture/écriture/supression

[3] Je ne parle pas ici des parties liées aux Flash NOR ou aux autres supports mémoires pris en compte par le driver MTD…

bufferisées,…). Tout deux implantent aussi le traitement des requêtes d'*ioctl* et leur enregistrement en tant que *char device* et *block device*

- **nand_ecc** : On trouve dans ce module les fonctions de calcul du code de correction d'erreur, et logiquement, la fonction de correction des données ;
- **nand** : Le driver générique de Flash NAND inclu les algorithmes de commande de la Flash, la fonction d'attente générique basée sur celle spécifique du module suivant, les algorithmes de lecture et d'écriture de pages ou de données additionnelles aux pages, avec ou sans correction d'erreur, ainsi que d'effacement, et l'initialisation de la Flash ;
- **nandspec** : Cette couche est l'implémentation des fonctions spécifiques au matériel, qui doit proposer au minimum les fonctions *hwcontrol* et *dev_reday*. Le module est aussi évidemment en charge de l'initialisation des structures et de la réservation des ressources. Cette partie est sollicitée par la partie *nand* générique pour des opérations atomiques sur la Flash.

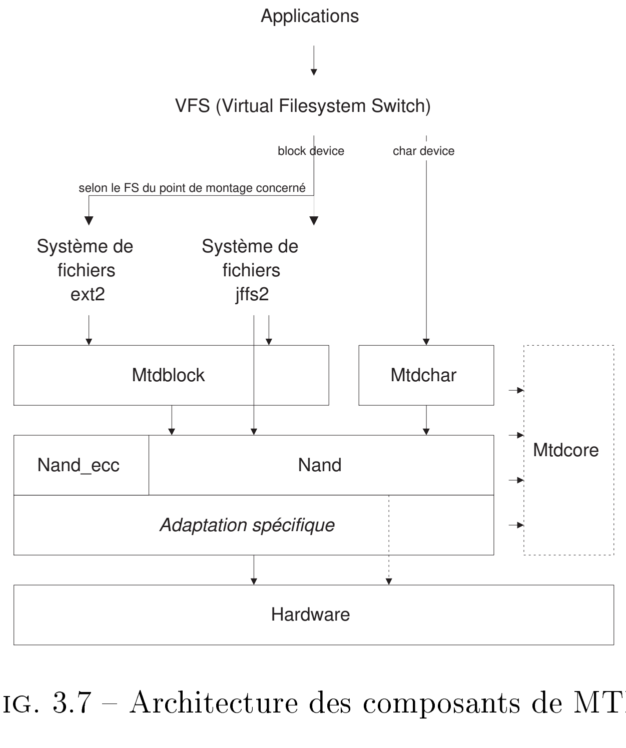

FIG. 3.7 – Architecture des composants de MTD

Il y a au moins une chose n'apparaissant pas dans cette architecture, car il ne s'agit pas à proprement parler d'une partie du driver : JFFS2. Comme il l'a déjà été dit dans le point 3.1.2, c'est un système de fichiers journalisé (ou du moins un système de fichier à structure de journal - *log-structured filesystem*) qui est spécialement prévu pour être placé au-dessus des mémoires Flash. Contrairement au comportement normal qui est de se placer en amont du périphérique en mode bloc, *i.e.* en utilisant des fonctionnalités comme celles proposées par *mtdblock*, JFFS2 va court-circuiter la hiérarchie établie et ne se servir de ce module que pour accéder au structure de niveau inférieur.

L'approche habituelle encore récemment pour l'utilisation des Flash en systèmes embarqués était d'utiliser un pseudo-système de de fichier tenant compte de la technologie pour émuler un *block device* et d'y placer ensuite le système de fichier voulu. Tout l'intérêt de JFFS est d'agir intelligement de bout en bout par rapport à la connaissance du support et de ses contraintes.

Le périphérique en mode bloc inclu dans les MTD ne propose pas les fonctions le rendant propre à supporter efficacement un système de fichier comme *ext2* sur Flash. Il s'agit plus d'une commodité de développement. Une approche plus complexe est donc requise, d'où la présence de JFFS2 lié aux MTD.

On peut aussi noter la présence dans le driver d'une partie *Flash Translation Layer* qui reprend - il s'agit gloablement d'un portage - les mécanismes ayant été développés par M-Systems pour leur solution *DiskOnChip*. Avant l'utilisation de FlashFX, le logiciel en provenance de M-Systems était en fonction dans les PABX. Néanmoins, je n'envisagerai pas par la suite son utilisation, d'une part car c'est une partie logicielle brevetée et d'autre part car (selon la liste de diffusion concernant le driver MTD) le port n'a été réalisé que pour des raisons de compatibilité, et qu'il n'est pas destiné à être suivi pour développement.

3.4 Conclusion

Cette présentation des différents points abordés dans l'un et l'autre des drivers permet de se rendre compte du résultat à obtenir dans l'adaptation de la couche spécifique du driver des MTD pour fournir un niveau de service égal à celui du driver FlashFX.

La partie suivante du rapport va montrer tout d'abord cette adaptation, puis ses résultats…

Deuxième partie

Travail réalisé

Chapitre 4

Adaptation du driver Linux

*Dans lequel on adapte le driver en fonction du matériel
et où on «débogue» aussi du kernel.*

4.1 Préparations préalables

Pour la phase de développement et de test, je travaille sur le PABX avec une installation qui n'est pas la même que celle d'une machine en production. Alors qu'un PABX est normalement autonome, avec un système correspondant à une version bien précise (*release*), je l'ai configuré pour ma part de manière à faire un démarrage par le réseau et utiliser un système de développement, voire de *rescue* qui propose un système minimal.

Je n'ai pas produit mes propres systèmes en utilisant la chaîne de production, qui réalise grâce à une constellation de scripts l'ensemble des compilations et assemblages en fonction de l'environnement défini et du contrôle de source [1]. J'ai pu me contenter des systèmes prédéfinis (*install*, *dev* et *rescue*) pour mes besoins, modifiant après coup certains points (comme l'arborescence dans le répértoire `/dev` par exemple...).

J'ai cependant effectué un certain nombre de modifications pour personnaliser le noyau et y intégrer correctement la partie MTD de Linux. J'ai

[1]Le système de contrôle de source est basé sur SCCS, avec une interface graphique baptisée *TeamWare*. Sans entrer dans le détail, j'ai recupéré pour travailler l'arborescence de fichiers liés au noyau Linux et à son patch RTLinux, et je n'ai que peu interagit avec *TeamWare*, n'ayant pas à faire remonter des modifications et travaillant sur des parties non sujettes à des mises à jour fréquentes.

entrepris d'automatiser les modifications permettant - à partir de l'arborescence récupérée par le système de contrôle de source pour un environnement de travail concernant Linux/RTLinux - d'appliquer en une fois l'ensemble des changements et de générer des scripts utilisé par la suite en fonction des répertoire de travail en cours. Le script créé dans cette optique n'est plus actuellement à jour, mais il s'agissait avant tout d'une commodité personnelle et non d'un outil destiné à la chaîne de production... Ses actions synthétisaient :

- Décompresser la dernière version CVS du driver MTD (des images journalières étant disponibles sous forme d'archive sur le site [MTD]) et copier les sources au sein de l'arborescence en tenant compte du système de contrôle de versions ;
- Désactiver la compilation de linux/fs/inflate_fs qui entrait en conflit par la suite ;
- Supprimer la partie linux/fs/jffs2/zlib qui est reportée au niveau de linux/lib/zlib et modifier linux/lib/Config.in pour ajouter les options de compilations correspondantes ;
- Faire une copie du fichier d'en-têtes linux/fs/jffs2/crc32.h pour le rendre aussi accessible dans linux/include/linux pour permettre la compilation de JFFS2 ;
- Appliquer le patch mtd qui réalise d'autres modifications à divers endroits du noyau.

Cela étant fait, on peut recompiler le noyau

- en le configurant par make xconfig et en lui indiquant les options de compilation pour les MTD et JFFS (voir en annexe B.4)
- en lançant la compilation par une rêgle spécifique à Alcatel, à savoir make kernel.plk, qui réalisera les dépendances nécessaires et traitera le résultat de la compilation comme voulu

4.2 Spécifications matérielles

Les opérations qu'on peut réaliser sur la Flash (écrire une page, effacer un bloc, consulter le status, demander l'identifiant du constructeur et du modèle, etc.) sont décrites dans les spécifications que le constructeur fournit avec la puce mémoire. Ces spécifications se trouvent sous la forme de chronogrammes, indiquant les entrées et les sorties possibles et nécessaires d'effectuer pour mener à bien une opération, ainsi que certains exemples pour des cas spéciaux (*e.g.* le comportement attendu si on réalise une opération de lecture durant une phase d'effacement.).

Le chronogramme pour l'effacement d'un bloc, un exemple parmi d'autres, est reproduit en figure 4.1, et les valeur temporelle qui lui sont associées sont reportées dans le tableau 4.1 - tous deux p.38.

D'autres part, il a fallu tenir compte des spécifications de la plateforme Alizé. Celle-ci sont regroupées dans un document technique [ALZ] interne à Alcatel, disponible via le centre de documentation en ligne. Parmi l'ensemble des données fournies, comme la carte de la mémoire ou la carte des entrées/sorties, j'ai extrait celles pertinentes à la réalisation technique de mon travail, c'est-à-dire quelques lignes réparties sur moins d'une dizaines des cent pages que compte le document, mais dont il faut comprendre la teneur.

4.3 Driver réalisé

Avec du recul, l'adaptation est réélement simple. L'implémentation (disponible en annexes A.1 et A.2) reposant sur la mise à disposition pour le driver générique, *i.e.* le module *nand* des MTD, des fonctions hwcontrol qui doit transmettre correctement le signal de contrôle spécifique à la Flash - voir en figure 4.2 -, et dev_ready qui indique si la Flash est prête ou occupée.

Les difficultés que j'ai rencontrées sont de deux ordres : 1° trouver quelle information s'applique au cas voulu, parmi les sources de renseigements - *e.g.* il n'est pas évident de savoir si on est concerné par ce qui est dit dans un extrait de message de la liste de diffusion : parle-t-on de NOR ou de NAND ? S'agit-t-il d'un problème générique ou spécifique à un matériel ? Est-ce encore valable ou rendu obsolète ? Est-ce en interaction avec une configuration particulière du noyau ? Tant de choses rendant l'information inapplicable hors du contexte. Avec du recul, il est plus facile de faire un tri, mais c'est une expérience à acquérir et un flou demeure parfois 2° À cette échelle de granularité des opérations, le plus petit changement peux rendre l'ensemble inopérant, instable ou fautif. Faut-il alors chercher les cause du côté des délais ou de l'ordre d'envoi des commandes ? de la programmation des I/O ou de la présence de la XMem[2] ?

Au final, les implémentations minimales ont suffit, *i.e.* les possibilités du driver quant à la personnalisation de la plupart des étapes de son *modus operandi* n'ont pas eu à être utilisées. Il a cependant fallu que je modifie des détails du driver générique (devenant ainsi légérement spécifique, mais dans une mesure acceptable) et que j'ajoute une phase d'initialisation dépendant

[2] la carte supportant de la mémoire Flash additionnelle, comme introduite en 1.6

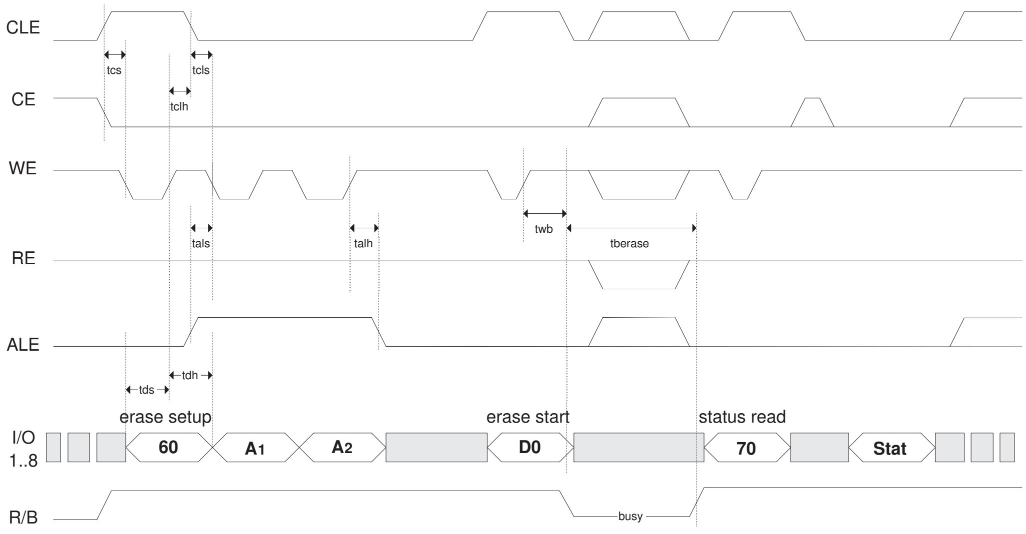

FIG. 4.1 – Chronogramme spécifiant la méthode d'effacement d'un bloc

Symbole	Paramètre	Min	Max
tcls	CLE Setup Time	0	-
tclh	CLE Hold Time	10 ns	-
tcs	CE Setup Time	0	-
tch	CE Hold Time	10 ns	-
tals	ALE Setup Time	0	-
talh	ALE Hold Time	10 ns	-
tds	Data Setup Time	20 ns	-
tdh	Data Hold Time	10 ns	-
twb	WE High to Busy	-	200 ns
tberase	Block Erasing Time	3 ms	20 ms

TAB. 4.1 – Conditions temporelles à respecter lors de l'effacement

de la présence d'une carte additionnelle de l'Alizé pour une programmation correcte des I/O.

```
~~~

/* Convert to fit PIO */
SREG_signals = PIO_convert[signals & PINS_ALL];

/* Write the signals to the Signal Register of XMEM */
NAND_OUTBYTE (NAND_CMD, SREG_signals);

/* Write also these signals though the Aspen */
/* if ever there is no XMEM */
PIOSignals = NAND_GET_PIO(PORT_R_PIODATA);
PIOSignals &= ~PINS_ASPEN_ALL;

/* CE signal is inverted on CPU board */
PIOSignals |= (SREG_signals ^ PINS_ASPEN_CE);
NAND_SET_PIO(PIOSignals, PORT_W_PIODATA);

~~~
```

FIG. 4.2 – Envoi d'un signal de contrôle à la Flash : extrait de la fonction aspenUpdateFlashSignals() dont le paramètre signals va diriger les broches de la Flash. Il s'agit là d'un point clé de l'adaptation car c'est le type d'opération sollicité par la partie générique du driver. Les entrées/sorties doivent pour cela être correctement préparées à remplir leur fonction.

4.4 Debugging

Mon environnement de travail - voir en figure 4.3 - était constitué d'une station Sun et d'un PABX Alizé à base de processeur Aspen. J'appliquais mes diverses tentatives de développement sur le PABX, recoupant les cas d'erreurs par les messages kernel pour progresser vers un logiciel fonctionnel. Quand j'en suis arrivé à avoir des *Oops*[3] aléatoires, et que je les ai situés au beau milieu de la gestion des *slabs*, c'est-à-dire dans la gestion de la mémoire vive, des caches et autres sous Linux, il a fallu mettre en place un système plus puissant pour éliminer les erreurs.

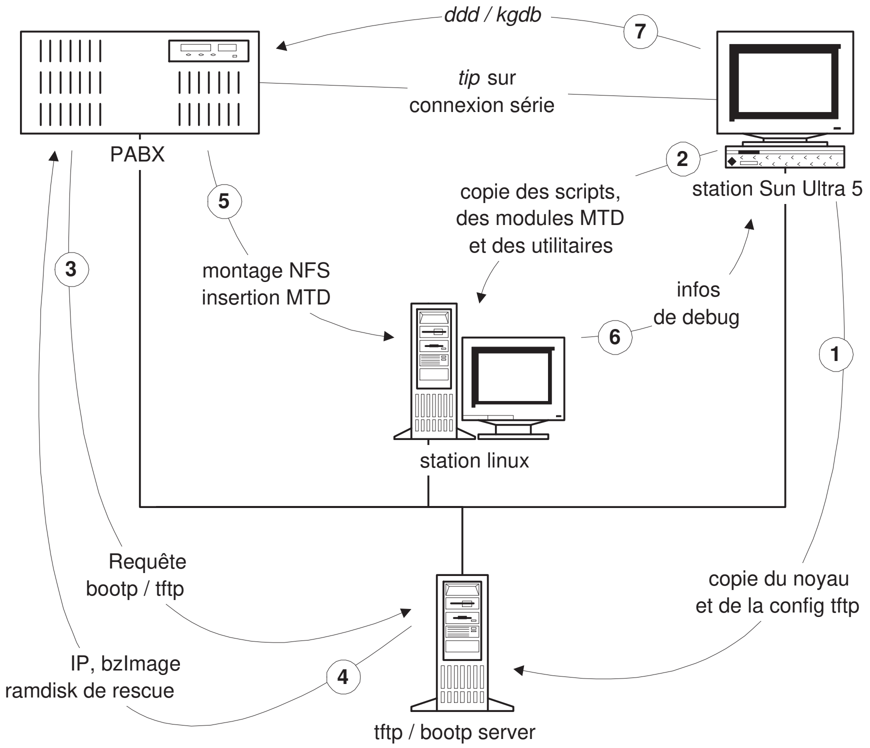

FIG. 4.3 – Installation pour le *debugging* du noyau sur le PABX

Le détail des actions pour pouvoir tracer le déroulement du noyau sont énumérées ci-dessous :

1. Une fois le kernel et les modules compilés, il faut placer le noyau compressé, *i.e.* le fichier *bzImage*, sur le serveur *tftp*. Il faut aussi créer le

[3] une erreur grave au sein du noyau qui dès lors ne garantit plus la cohérence de ses structures internes, souvent suivi d'un message de *Panic !* nécessitant le redémarrage de la machine.

fichier de configuration *tftp*, qui spécifie entre autre la taille du noyau à charger. Pour cela, j'ai créé un script automatisant la configuration - voir en annexe B.2.

2. Il faut placer les modules devant être chargés par la suite dans un répértoire partageable via NFS. J'utilise une machine Linux où j'ai rendu partageable un répértoire MTD. J'y ai aussi mis des scripts réalisant les opérations nécessaires à l'insertion des modules.

3. On peut maintenant démarrer le PABX. Au démarrage, il effectue une requête bootp pour déterminer son adresse IP, puis une requête tftp pour charger dans un premier temps son noyau - celui qui vient d'y être placé - et ensuite son ramdisk.

4. Le serveur bootp/tftp lui envoie son adresse, son noyau et son ramdisk. Un problème que j'ai rencontré à ce point, et qui m'aurait longtemps laissé perplèxe s'il ne s'était agit d'un bug connu dont on avait la solution, survient quand la taille du noyau, en héxadécimal, finit par *0xE0* et que la version du BIOS est inférieure à 8.12. Il appert alors que le chargement via tftp boucle...
Cette installation étant destinée à tester le driver MTD, j'utilise comme système le ramdisk de *rescue*, qui charge un minimum de choses, en particulier qui ne charge pas le driver FlashFX. Bien qu'il soit sous la forme de module, ôter celui-ci de la mémoire pose problème avec le ramdisk de développement, car divers périphériques sont montés en *loopback* et occupés (*busy*) par la suite, donc non démontables...

5. En utilisant la liaison série et l'utilitaire *tip*, j'accède au Linux tournant sur le PABX. Je monte via NFS le répertoire partagé de la station Linux, et je lance le script d'insertion des modules MTD - voir en annexe B.3 et B.1 - qui va en outre ajouter une ligne au fichier /etc/fstab, permettant ainsi par la suite de se connecter via *telnet* au PABX, ce qui était jusque là impossible car les pseudo-terminaux n'étaient pas disponibles.

6. Si le script d'insertion des modules est lancé avec l'option *-gdb*, des données seront générées pour permettre le debuggage. En effet, le debugger nécessite certaines informations concernant le placement en mémoire des modules qui ne peuvent être connues qu'à l'insertion de ceux-ci. Un ensemble de commandes va, durant cette phase, être rassemblé dans un fichier sur la station Linux, ainsi que les commandes de chargement des autres symboles du noyau.

7. L'ensemble de l'installation est quasiment près à être debuggé. Les étapes restant à accomplir sont : terminer la session *tip*, se connecter via *telnet* sur le PABX et lancer *gdbstart*. Lancer sur la station l'interface *ddd* en lui spécifiant de tourner au dessus de *i486-pc-linux-kgdb*

pour homogénéiser les environnements, le paramétrer pour fonctionner sur la ligne série (terminal utilisé et débit de la ligne), et enfin charger les commandes du script produit à l'étape précédente. La table des symboles du noyau est chargée, et chaque table des symboles des modules est chargée.

4.5 Conclusion

L'adaptation du driver Linux n'est pas une chose compliquée en soit. Une fois compris quel type d'adaptation et de paramétrage il faut réaliser, il s'agit d'implémenter les spécifications concernant la Flash pour rendre effectives les opérations diverses.

Il faut cependant préciser que le tri entre les sources d'informations appropriées et celles non-valides rend les choses confuses et l'immersion dans le sujet plus difficile - ou du moins plus longue - que cela ne devrait. De même, les erreurs liées à la programmation noyau peuvent être une vraie source de blocage.

Il y a en outre un point qui m'a pris du temps au début de mon stage et que je n'ai pas abordé ici : Alcatel a construit en interne une carte ISA supportant deux Flash NAND à des fins de test. Bien que suivant les spécifications matérielles fournies, je n'ai pas réussi à en faire quelque chose (*i.e.* pas même à accéder à l'identifiant de la Flash).

Ayant abouti à un driver utilisable pour la Flash, j'ai pu passer à la suite logique du travail, au vu des objectifs, qui a consisté à comparer les deux solutions désormais disponibles.

Chapitre 5

Comparatif entre le driver réalisé et l'existant

Dans lequel on aborde les résultats concrets
à retirer du travail réalisé.

5.1 Préambule

Cette partie devant aboutir à une réponse concernant l'utilisabilité du driver implémenté, il faut s'axer sur des critères de comparaison. Je m'attache essentiellement aux points suivants :

- les *différences fondamentales* qu'il faut garder à l'esprit pour ne pas jauger sur un même critère des choses incomparables ;
- les *performances* quantifiables qui permettent une vue chiffrée de l'utilisabilité et de l'efficacité du système, ainsi que l'*espace* physique obtenu sur le médium, indicateur de gain ;
- le *coût et l'évolutivité* engendré par une nouvelle solution.

5.2 Différences intrinsèques

Le driver FlashFX étant conçu pour être non seulement simplement adaptable aux différents matériels mais aussi aux différents systèmes d'exploitation au-dessus de lui, il a au moins un niveau d'abstraction supplémentaire dans son architecture. Il intégre de plus clairement une couche de translation. Ces deux niveaux hiérarchiques ne se retrouvent pas dans le driver MTD qui d'une part est lié au système Linux, et d'autre part déporte la couche de translation, et ce qu'elle implique (gestion des blocs erronés, *wear levelling*...) au niveau du système de fichier JFFS2.

Une comparaison directe n'est donc pas évidente, car ces différences ont des implications dans des étapes diverses des fonctions, en particulier dans le service rendu. Ainsi, placer *ext2* sur *mtdblock* n'est pas comparable à placer *ext2* sur FlashFX car dans le second cas on aura *garbage collection, wear levelling* et *bad block management*, un ensemble de mécanismes incomparable à la simple gestion de cache du *block device* du premier cas. De même, comparer vis-à-vis de *jffs2* devrait prendre en compte le fait qu'on ait là un système à structure de journal, mais cette différence n'est pas - pour moi - quantifiable dans ce contexte...

5.3 Évolutivité et coût

Bien qu'il subsiste quelques problèmes ponctuels avec le driver en cours d'utilisation, il fonctionne suffisamment bien pour être en production à grande échelle. Cela dit, quand une modification est à apporter, c'est au sein d'Alcatel que doivent se faire les améliorations, car le driver a divergé depuis longtemps de ce qu'était capable d'apporter DataLight en terme de support ou de maintenance.

Un intérêt dans l'utilisation du driver de Linux est donc de profiter de la dynamique de la communauté de développeurs pour avoir une auto-évolutivité du produit, et concentrer l'effort au niveau des adaptations vraiment spécifiques au matériel.

D'un côté comme de l'autre, c'est-à-dire considérant l'achat en *buyout* ou l'utilisation de logiciel en licence GPL[1], le coût ne sera pris en compte que dans l'aspect de l'évolutivité du produit, et donc en faveur des MTD (après une phase de mise à jour pour atteindre le niveau actuel de FlashFX bien sûr...)

[1] pour certaines partie en Dual GPL/MPL mais peu importe

5.4 Tests de performances

J'ai utilisé une partie des tests de performances et non régressions préparés un peu auparavant par Antoine Bantzhaff, un autre stagiaire. La partie vérifiant le système de fichiers sur la Flash repose sur *stressFs*, un utilitaire qui réalise de nombreux accès en lecture, écriture et suppression. Cette action est complétée par une exécution concurrentielle (quatre processus en parallèle). Les détails de paramétrages de *stressFs* sont dans le script de lancement en annexe C.1.

L'aspect de non régression du test va se retrouver au niveau des logs dans le signalement des erreurs. Dans tous les cas, le nombre d'erreurs obtenus est zéro, ce qui semble montrer que le comportement en cours de fonctionnement supporte correctement le système de fichier. Il faut tout de même nuancer cette position :

- on ne sait pas si le cas d'un bloc erroné a été rencontré - cette fonction n'étant accessible que par un utilitaire dépendant de FlashFX ;
- on ne teste pas les opérations de montage/démontage successifs. Cela ne correspond de toute façon pas au comportement du PABX, mais le tester pourrait mettre en évidence des vices cachés ;
- on ne teste pas la validité de part et d'autre d'un redémarrage ;
- on ne teste pas les arrêts brutaux (coupure de courant) mais dans le cas de l'utilisation de JFFS2, le système étant journalisé, il est a priori prévu pour bien gérer la situation ;
- il n'y a pas eu de profonde réflexion sur ce que devrais être une procédure de test/validation complète de la Flash.

Pour ce qui est des performances, les résultats sont présentés dans le tableau 5.1, dont on peut trouver le détail en annexes C.2, C.3 et C.4.

	ext2/Ffx	ext2/Mtd	jffs2/Mtd
Écriture			
Petits fichiers	1.74 Mo/s	1.20 Mo/s	0.10 Mo/s
Grands fichiers	1.70 Mo/s	0.68 Mo/s	0.10 Mo/s
Lecture			
Petits fichiers	0.73 Mo/s	0.18 Mo/s	0.37 Mo/s
Grands fichiers	1.13 Mo/s	0.69 Mo/s	0.44 Mo/s
Suppression			
Petits fichiers	0.08 Mo/s	0.85 Mo/s	0.16 Mo/s
Grands fichiers	0.36 Mo/s	2.01 Mo/s	0.39 Mo/s

TAB. 5.1 – Résultats moyens arrondis des tests de performances

On remarque avec ces résultats que l'utilisation de JFFS2 implique un écroulement des performances qui rend inapplicable son intégration en l'état. Le «pourquoi» de cet écroulement fait partie des points que j'aurais aimé avoir eu le temps d'approfondir. En l'état actuel, ce fait est rédibitoire.

Un autre point de vue sur les performances se rapporte au gain possible en espace disponible sur le médium. Il s'agit d'une ressource fortement mise à partie dans les systèmes en production, car elle doit entre autre permettre d'héberger des fonctionnalités supplémentaires du système embarqué.

La gestion des blocs erronés dans l'architecture de FlashFX implique de réserver une certaine place pour faire face au pourcentage de déficiences prévues par le constructeur, et on retrouve cette réservation d'espace de manière quantifiée. Pour une comparaison, il n'est malheureusement pas possible de calculer l'implication de l'utilisation de JFFS2 en terme de perte de place, car il s'agit d'un point variable dépendant de l'utilisation en cours. Une partie est réservée là aussi pour mener à bien les opérations de collecte d'espace - mais les documents ne sont pas clair sur la quantité, et il n'y a pas de preuve formelle qu'elle soit suffisante dans tous les cas - et varie selon la quantité de données recouvertes ou obsolètes de la structure de journal…

5.5 Autres points à traiter

D'autres points à prendre en compte en vue d'une éventuelle intégration, pour estimer en partie l'effort de travail, sont :

- la facilité de mise à jour au sein de l'environnement sous contrôle de source des changements introduits dans la branche CVS en cours ;
- les essais d'utilisation des partitions sur la mémoire supplémentaire des cartes additionnelles XMem - le code du driver et les tests que j'ai fait n'ont pas porté sur cette partie, mais j'estime que cela ne devrait pas soulever de problèmes ;
- l'adaptation à l'architecture à base de PIII est présente dans mon travail, mais sans doute incomplète et absolument non-testée. Là encore, il s'agit d'un point qui ne devrait pas être problématique ;
- le compteur d'utilisation du module *mtdblock* n'est pas décrémenté à un endroit que je n'ai pas identifié. C'est un problème très mineur mais pas «cosmétique» ;
- on rencontre des problèmes si on tente d'utiliser de manière altérnée les deux drivers FlashFX & MTD. Ce n'est guère dérangeant en soit, mais ça indique des défauts dans l'architecture de l'un ou de l'autre ;

- la plateforme Alizé utilise un système Linux modifié pour le temps-réel par un patch RTLinux, il faudra donc étudier les implications de ce contexte du point de vue du driver. On pourrait penser que les deux domaines sont distincts en connaissant le principe de RTLinux, mais il faut savoir qu'une dérive d'horloge a été observée et liée au fonctionnement de FlashFX, ce qui n'avait rien d'intuitif, donc qu'on peut éventuellement faire apparaître le même genre de comportement ;
- le port de FlashFX au niveau du BIOS consiste en grande partie en une copie des sources et des modifications ponctuelles. Il faudra déterminer si un tel port est possible avec les MTD.
- ...

5.6 Conclusion

Une conclusion suffisante serait de dire qu'en l'absence de piste pour rendre les performances viables, l'intégration est tout simplement impossible.

Pour compléter cette affirmation par trop manichéenne, on peut supposer qu'il est possible d'obtenir des performances normales en gardant l'option - qui n'en est pas vraiment une si on veut un système fiable - de garder JFFS2. Alors, il reste, pour envisager de partir sur cette solution, à déterminer l'effort de travail restant à faire : intégration du driver au niveau du BIOS et du démarrage, compression du système de fichier, etc.

Bien que ce driver ne soit pas poursuivi à la suite de ce stage, il fera l'objet d'une seconde estimation lors de la convergence future des plateformes, qui doivent se recentrer sur Linux et réunifier certains points. Il sera temps de vérifier l'état d'avancement au niveau Linux des drivers et de poursuivre l'investigation...

Conclusion

Mon travail durant ce stage de fin d'études a porté sur un pilote de mémoire Flash. Les objectifs ont été de réaliser l'étude et l'adaptation d'un pilote sous Linux, puis sa comparaison avec le pilote actuellement utilisé en vue de connaître les possibilités de remplacement.

L'adaptation du driver m'a pris plus de temps que prévu, mais j'ai produit une version viable au final. Le délai a par conséquent été réduit pour la seconde partie du travail, et j'aurais aimé approfondir certains points, mais cela a cependant suffit pour apporter des éléments de réponses satisfaisants.

La présentation du bilan de stage a permis d'exposer le travail réalisé et de soumettre aux questions et remarques au reste de l'équipe les résultats obtenus quant à la comparaison des deux pilotes. Les conclusions dégagées attestent que les inconvénients subsistant dans le nouveau driver et que l'effort de travail nécessité pour une intégration ne justifient pas de poursuivre pour l'instant l'investigation. Néanmoins, une convergence des plateformes étant prévue et centrée sur le système Linux, la question pourra être redébattue dans le future, en profitant des développements actifs de la communauté Linux durant le temps écoulé...

Pour conclure sur une note personnelle, j'ai apprécié d'avoir l'opportunité de faire ce stage et d'être impliqué dans un driver, car il s'agit d'un type de problème relativement peu courant auquel je n'avais pas encore eu à me confonter. Dans le cadre de la formation, l'ensemble de la partie système est une bonne base pour attaquer concrétement le problème.

Ce sujet a été enrichissant et j'espère continuer à l'avenir à explorer les différents aspects de l'informatique.

Bibliographie

[RUB98] Rubini, Alessandro. *Linux Device Drivers.* O'Reilly. 1998.

[KHG] Johnson, Michael K. *The Linux Kernel Hacker Guide.*
`http://tldp.org/LDP/khg/HyperNews/get`.

[MTD] Woodhouse, David & Gleixner, Thomas. *MTD Subsystem for Linux.*
`http://linux-mtd.infradead.org`.

[JFFS] Woodhouse, David. *The Journalling Flash File System.*
JFFS2 : `http://sources.redhat.com/jffs2/jffs2.pdf`
JFFS1 : `http://www.developer.axis.com/software/jffs/`

[ALZ] *ALIZE Rack : CPU/Applicative CPU/Controller Input/Output Handbook.*

Troisième partie

Annexes

Annexe A

Sources

A.1 Alcalz

```
/*
 * Board-specific glue layer for NAND flash on ALIZE
 */

#include <linux/slab.h>
#include <linux/module.h>
#include <asm/io.h>
#include <linux/ioport.h>
#include <linux/mtd/mtd.h>
#include <linux/mtd/nand.h>
#include <linux/mtd/partitions.h>
#include <linux/interrupt.h>
#include <asm/delay.h>

#include "alcalz.h"
#include "alcalz_piii.h"

void alcalz_cleanup(void);
int  alcalz_init(void);

/* *** Protoz for implementation *** */
/* Aspen */
void aspenUpdateFlashSignals(unsigned char);
int  aspenDeviceReady(void);
```

```
void aspenDecodingDetection(void);
/* PIII */
void piii_UpdateFlashSignals(unsigned char);
int  piii_DeviceReady(void);
void piii_DecodingDetection(void);

/* *** Implementation choices *** */
void (*updateFlashSignals)(unsigned char)=aspenUpdateFlashSignals;
int (*alcalz_dev_ready)(void) = aspenDeviceReady;
void (*xmemDecodingDetection)(void)=aspenDecodingDetection;

/* Mtd structures */
#define NUM_PARTITIONS 1
static struct mtd_info *alcalz_mtd = NULL;
static struct mtd_partition partition_info[] = {
        { name:   "ALCALZ flash partition",
          offset: 0,
          size:   512/*bytes*/ * 32/*pages*/ * 2048/*blocks*/ }
};

/* To request io region */
static int io_base = 0x400;
static int io_range = 4;
MODULE_PARM(io_base,"i");
MODULE_PARM(io_range,"i");

void alcalz_hwcontrol(int cmd){
        unsigned char sig;
        /* iok debug clisti cli(); */
        sig=NAND_INBYTE(NAND_CMD);
        switch(cmd){
        case NAND_CTL_SETCLE: sig|= PINS_CLE; break;
        case NAND_CTL_CLRCLE: sig&=~PINS_CLE; break;
        case NAND_CTL_SETALE: sig|= PINS_ALE; break;
        case NAND_CTL_CLRALE: sig&=~PINS_ALE; break;
        case NAND_CTL_SETNCE: sig|= PINS_CE;  break;
        case NAND_CTL_CLRNCE: sig&=~PINS_CE;  break;
        }
        updateFlashSignals(sig);
        /* iok debug clisti sti(); */
}

/* *** Aspen Implementation *** */
```

```
void aspenDecodingDetection (void) {
  unsigned int uireg;
  unsigned char ucreg;
  unsigned int PIOSignals ;

  /* Aspen do not drive the Device Select */
  uireg = 0;
  uireg = NAND_GET_PIO(PORT_R_PIODIR);
  /* The CE signal must be set */
  NAND_SET_PIO(uireg | 0x0007, PORT_W_PIODIR);
  PIOSignals = NAND_GET_PIO(PORT_R_PIODATA);
  PIOSignals |= 0x0001; /* set bit CE */
  NAND_SET_PIO(PIOSignals, PORT_W_PIODATA);
  uireg |= 0x0006;
  uireg &= 0xFFFE;
  /* CE Programmed as input */
  NAND_SET_PIO(uireg, PORT_W_PIODIR);
  /* Just let the Aspen do the job */
  uireg = NAND_GET_PIO(PORT_R_PIODIR);
  /* select Flash device on CPU board */
  NAND_OUTBYTE(NAND_DEV_SEL, 0);
  /* set CE of this flash */
  ucreg = PINS_CE;
  NAND_OUTBYTE (NAND_CMD, ucreg);
  /* Read CE pin of flash device through PIO16 of Aspen  */
  /* CE=0 means chip enabled. CE=1 means CE not enabled. */
  uireg = 0;
  uireg = NAND_GET_PIO(PORT_R_PIODATA);
  uireg &= 0x0001;

  if(uireg==0) {
    /* The XMem board is present. */
    /* PIO16 of ASPEN already set */
    /* iok printk("alcalz-dbg: Xmem  board detected\n" );*/
  } else {
    /* The CE of CPU flash device need to be */
    /* controlled by PIO16 of ASPEN */
    uireg = 0;
    uireg = NAND_GET_PIO(PORT_R_PIODIR);
    NAND_SET_PIO(uireg | 0x0007, PORT_W_PIODIR);
    uireg = NAND_GET_PIO(PORT_R_PIODIR);
    /* iok printk("alcalz-dbg: No Xmem board detected\n" );*/
```

```
  }

  /* Free the flash device */
  ucreg = 0;
  NAND_OUTBYTE(NAND_CMD, ucreg);
}

void aspenUpdateFlashSignals(unsigned char signals) {
        unsigned int PIOSignals;
        unsigned char SREG_signals;
        unsigned char PIO_convert[8] = {
          /* nothing        */ 0,
          /* CE             */ PINS_ASPEN_CE,
          /* CLE            */ PINS_ASPEN_CLE,
          /* CE + CLE       */ PINS_ASPEN_CE
                             | PINS_ASPEN_CLE,
          /* ALE            */ PINS_ASPEN_ALE,
          /* ALE + CE       */ PINS_ASPEN_ALE
                             | PINS_ASPEN_CE,
          /* ALE + CLE      */ PINS_ASPEN_ALE
                             | PINS_ASPEN_CLE,
          /* ALE + CE + CLE */ PINS_ASPEN_ALE
                             | PINS_ASPEN_CE
                             | PINS_ASPEN_CLE
        };

        SREG_signals = PIO_convert[signals & PINS_ALL];
        /* Write the signals to the Signal Register of XMEM */
        NAND_OUTBYTE (NAND_CMD, SREG_signals);
        /* Write also these signals though the Aspen */
        /* if ever there is no XMEM */
        PIOSignals = NAND_GET_PIO(PORT_R_PIODATA);
        PIOSignals &= ~PINS_ASPEN_ALL;
        /* CE signal is inverted on CPU board */
        PIOSignals |= (SREG_signals ^ PINS_ASPEN_CE);
        NAND_SET_PIO(PIOSignals, PORT_W_PIODATA);
}

int aspenDeviceReady(void) {
        unsigned char reg;
        reg = NAND_GET_PIO(PORT_R_PIODATA);
        return(reg&PINS_RDY_CPU);;
}
```

```
/* *** PIII Implementation *** */

void piii_DecodingDetection(void) {
  return;
}

void piii_UpdateFlashSignals(unsigned char signals) {
  unsigned int PIOSignals;
  unsigned int Lsignals;
  unsigned int flags;
  /* Warning, CE signal is inverted on CPU board */
  struct {
    unsigned int bits;
    unsigned char label[10];
    } PIO_convert[8] = {
    /* nothing        */ { 0, "-"},
    /* CE             */ { PINS_PIII_CE,
                           "        CE"},
    /* CLE            */ { PINS_PIII_CLE,
                           "    CLE   "},
    /* CE + CLE       */ { PINS_PIII_CE
                         | PINS_PIII_CLE,
                         "    CLE-CE"},
    /* ALE            */ { PINS_PIII_ALE,
                         "ALE       "},
    /* ALE + CE       */ { PINS_PIII_ALE
                         | PINS_PIII_CE, "ALE  -  CE"},
    /* ALE + CLE      */ { PINS_PIII_ALE
                         | PINS_PIII_CLE,
                         "ALE-CLE   "},
    /* ALE + CE + CLE */ { PINS_PIII_ALE
                         | PINS_PIII_CE
                         | PINS_PIII_CLE,
                         "ALE-CLE-CE"}
  };

  Lsignals = PIO_convert[signals & PINS_ALL].bits;
  LOCK_GPIO_ICH(flags);
  PIOSignals= NAND_GET_PIOPIII(cGP_LVL);
  PIOSignals &= ~PINS_PIII_ALL;
  /* Warning, CE signal is inverted on CPU board */
  PIOSignals |= (Lsignals ^ PINS_PIII_CE);
```

```
  NAND_SET_PIOPIII(PIOSignals,cGP_LVL);
  UNLOCK_GPIO_ICH(flags);
}

/* definitively shouldnt be that simple
* this'll have to be adapted */
void piii_NANDReady(void) {
        unsigned char reg;
        reg = NAND_GET_PIO(PORT_R_PIODATA);
        return(reg&PINS_RDY_CPU);;
}

/* *** Module stuff *** */

int __init alcalz_init(void) {
        struct nand_chip *this=NULL;
        printk(KERN_INFO "\n\nalcalz-dbg: \
               Alcalz_Init [%s %s %s:%s]\n",
               __DATE__,__TIME__,"Opt",
#ifdef CONFIG_MTD_NAND_ECC
#ifdef CONFIG_JFFS2_FS_NAND
               "ECC+JFFS2NAND"
#else
               "ECC"
#endif
#else
               "NoECC"
#endif
               );

        if(check_region(io_base,io_range) != 0) {
          printk(KERN_INFO "alcalz: IO %x-%x in Use\n",
                 io_base,
                 io_base+io_range-1);
          return 1;
        }
        request_region(io_base,io_range,"Alcalz Nand flash");
        /* iok debug */
        printk("alcalz-dbg: request_region %x-%x done\n",
                        io_base,
                        io_base+io_range-1);

        /* Allocate memory for MTD device structure */
```

```
/* and private data */
alcalz_mtd = kmalloc(sizeof(struct mtd_info)+
                     sizeof(struct nand_chip),
                     GFP_KERNEL);
if (!alcalz_mtd) {
  printk("alcalz: Unable to alloc \
         ALCALZ NAND MTD dev struct\n");
  alcalz_cleanup();
  return -ENOMEM;
}

/* Get pointer to private data */
this = (struct nand_chip *)(&alcalz_mtd[1]);

/* Initialize structures */
memset((char *)alcalz_mtd, 0, sizeof(struct mtd_info));
memset((char *)this,       0, sizeof(struct nand_chip));

/* Link the private data with the MTD structure */
alcalz_mtd->priv = this;

/* Set address of NAND IO lines */
this->IO_ADDR_R = NAND_RD_PORT;
this->IO_ADDR_W = NAND_WR_PORT;
/* Set address of hardware control function */
this->hwcontrol = alcalz_hwcontrol;
this->dev_ready = alcalz_dev_ready;
/* 15 us command delay time */
this->chip_delay = 15;

/* iok debug */ aspenDecodingDetection();
/* Scan to find existence of the device */
if(nand_scan(alcalz_mtd)) {
        alcalz_cleanup();
        return -ENXIO;
}

/* Allocate memory for internal data buffer */
this->data_buf =
    kmalloc(sizeof(unsigned char)*
            (alcalz_mtd->oobblock+alcalz_mtd->oobsize),
            GFP_KERNEL);
if (!this->data_buf) {
```

```
            printk ("alcalz: Unable to alloc NAND data buffer\n");
            alcalz_cleanup();
            return -ENOMEM;
        }

        /* Allocate memory for internal data buffer */
        this->data_cache =
            kmalloc(sizeof(unsigned char)*
                    (alcalz_mtd->oobblock+alcalz_mtd->oobsize),
                    GFP_KERNEL);
        if(!this->data_cache) {
            printk ("alcalz: Unable to alloc NAND data cache\n");
            alcalz_cleanup();
            return -ENOMEM;
        }
        this->cache_page = -1;

        /* Register the partitions */
        add_mtd_partitions(alcalz_mtd,
                           partition_info,
                           NUM_PARTITIONS);
        printk("alcalz-dbg: add_mtd_device done\n");

        return 0;
}
module_init(alcalz_init);

#ifdef MODULE
void __exit alcalz_cleanup(void) {
        struct nand_chip *this;
        if(alcalz_mtd) {
                /* priv data ptr */
                this = (struct nand_chip *)(&alcalz_mtd[1]);
                /* iok debug
                   if(this->data_buf) kfree(this->data_buf);
                   if(this->data_cache) kfree(this->data_cache);
                   / iok debug */
                del_mtd_partitions(alcalz_mtd);
                del_mtd_device(alcalz_mtd);
                if(alcalz_mtd) kfree(alcalz_mtd);
        }

        release_region(io_base,io_range);
```

```
        /* iok debug */
        printk("alcalz-dbg: release_region %x-%x done\n",
               io_base,
               io_base+io_range-1);
        printk("alcalz-dbg: *** alcalz_cleanup ***\n");
}
module_exit(alcalz_cleanup);
#endif

MODULE_LICENSE("GPL");
MODULE_AUTHOR("iok");
MODULE_DESCRIPTION("Board-specific glue layer \
                   for NAND flash on ALIZE");
```

A.2 Alcalz header

```
#ifndef ALCALZ_H_INCLUDED
#define ALCALZ_H_INCLUDED

/*
 * Definition used for NAND flash access
 * on ALIZE. Cf. Datalight FlashFx driver
 * (RT_PTF: drivers/ffx/nand.h)
 *
 * Used by alcalz.c
 * and nand.c if CONFIG_TARGET_ALIZE defined
 */

#include <asm/sc520_mmcr.h>

/* Definition of nand commands */
#define CMD_DATA_INPUT          0x80
#define CMD_READ_MODE1          0x00
#define CMD_READ_MODE2          0x50
#define CMD_RESET               0xFF
#define CMD_PROGRAM             0x10
#define CMD_BLOCK_ERASE         0x60
#define CMD_START_ERASE         0xD0
#define CMD_RESUME              0xD0
```

```
#define CMD_SUSPEND             0xB0
#define CMD_STATUS              0x70
#define CMD_REGISTER_RD         0xE0
#define CMD_READ_ID             0x90

/* Definition of IO port */
#define NAND_DEV_SEL    0x400
/* Device Select register located on XMEM       */
#define NAND_CMD        0x401
/* Command register located on XMEM             */
#define NAND_RD_PORT    0x402
/* Data Port for READ located on XMEM           */
#define NAND_WR_PORT    0x403
/* Data Port for write located on XMEM          */

/* Definition of Alize registers to access */
/* FLASH devices on XMEM   */
/* NEVER MODIFY these values which are used */
/* as INDEXES in a table !! */
#define PINS_CE         0x01
/* Chip Enable          */
#define PINS_CLE        0x02
/* Command Latch Enable */
#define PINS_ALE        0x04
/* Address latch Enable */
#define PINS_LATCH      (PINS_CLE | PINS_ALE)
/* Used to clear CLE and ALE */
#define PINS_ALL        (PINS_CE|PINS_CLE|PINS_ALE)
/* Used to clear CE, CLE and ALE */

/* definitions of ASPEN register bits to access FLASH devices */
#define PINS_ASPEN_CE   0x01
/* Chip Enable          */
#define PINS_ASPEN_CLE  0x02
/* Command Latch Enable */
#define PINS_ASPEN_ALE  0x04
/* Address latch Enable */
#define PINS_ASPEN_ALL  (PINS_ASPEN_CE|PINS_ASPEN_CLE|PINS_ASPEN_ALE)
 /* Used to clear CE, CLE and ALE */

/* definitions of PIII PIO bits to access FLASH devices */
#define PINS_PIII_CE    (0x04 << 23)
/* Chip Enable          */
```

```
#define PINS_PIII_CLE   (0x01 << 23)
/* Command Latch Enable */
#define PINS_PIII_ALE   (0x02 << 23)
/* Address latch Enable */
#define PINS_PIII_ALL   (PINS_PIII_CLE|PINS_PIII_ALE|PINS_PIII_CE)
/* Used to clear CE, CLE and ALE */

/* Definition of PIO to use for monitoring flash's RDY pin */
#define PINS_RDY_CPU    0x0080
/* PIO23                */
#define PINS_RDY_XMEM   0x0010
/* PIO20                */

/* iok debug */
#define INP(x)                          inb(x)
#define OUTP(adr,dat)                   outb(dat,adr)

/* Definition of PIO access */
#define NAND_SET_PIO(sig, port)     SET_MMCR_W((sig), (port))
#define NAND_GET_PIO(port)          GET_MMCR_W((port))
#define NAND_GET_PIOPIII(port)      inl((port))
#define NAND_SET_PIOPIII(sig,port)  outl((sig),(port));
#define NAND_INBYTE(a)              INP((a))
#define NAND_OUTBYTE(a, d)          OUTP((a), (d))
#define PORT_W_PIODATA              mmcr_reg->pio.rw_piodata31_16
#define PORT_W_PIODIR               mmcr_reg->pio.rw_piodir31_16
#define PORT_R_PIODATA              mmcr_reg->pio.rw_piodata31_16
#define PORT_R_PIODIR               mmcr_reg->pio.rw_piodir31_16

#endif /* ALCALZ_H_INCLUDED */
```

Annexe B

Scripts

B.1 insalcalz

```
#!/bin/bash

function usage() {
    echo "Usage: $0 [-gdb SCRIPTDIR] [[BASEDIR] ENVDIR MTDDIR]"
    exit
}

function f_insmod() {
    if [ $# -eq 0 ];then
        echo "Usage: function f_insmod MODNAME"
        exit
    fi
    insmod ./$1.o
}

function f_gdbinsmod() {
    if [ $# -lt 2 ];then
        echo "Usage: function f_gdbinsmod SCRIPT_PATH MODNAME"
        exit
    fi
    if [ ! -e $2_d.o ];then
        echo "Module $2_d.o not found, trying to make it"
        i486-pc-linux-ld -d -r $2.o -o $2_d.o
    fi
```

```
    if [ ! -e $2_d.o ];then
        echo "Module $2_d.o not found, trying non gdb module"
        f_insmod $2
    else
        gdbinsmod $1 `pwd`/$2_d.o
    fi
}


if [ ! "$1" != "--help" ];then echo -n "Help:";usage; fi

if [ $# -gt 5 ];then echo -n "Too many args($#):";usage; fi

INSMOD=f_insmod
if [ ! "$1" != "-gdb" ];then
        shift
        GDBSCRIPT_PATH=$1
        /bin/rm -f $GDBSCRIPT_PATH/load*.o
        INSMOD="f_gdbinsmod ${GDBSCRIPT_PATH} "
        shift
fi
echo "INSMOD=${INSMOD}"

if [ $# -gt 3 ];then echo -n "Too many args($#):";usage; fi

if [ $# -eq 0 ];then
        BASEDIR=/users/claudon1/tw/M200
        ENVDIR=claudon1_CLEANSE
        MTDDIR=${ENVDIR}/code/linux/mtd
        echo -n "No arguments: assume base="
        echo "$BASEDIR env=$ENVDIR mtd=$MTDDIR"
else
        if [ $# -eq 3 ];then
                BASEDIR=$1
                shift
        fi
        ENVDIR=$1
        MTDDIR=$2
        echo "Setting: base=$BASEDIR env=$ENVDIR mtd=$MTDDIR"
fi

# if flashfx driver remains, unload it
umount /dev/fla >& /dev/null
```

```
umount /dev/flb >& /dev/null
rmmod flashfx >& /dev/null

# create /dev/mtd*
if [ ! -e /dev/mtdblock0 ];then
        source ${BASEDIR}/${MTDDIR}/util/MAKEDEV.noseq
fi

# insert modules
cd ${BASEDIR}/${ENVDIR}/code/modules_install
cd lib/modules/2.4.17-rt3.1/kernel/drivers/mtd/

# linux/drivers/mtd
${INSMOD} mtdcore
${INSMOD} mtdblock
${INSMOD} mtdpart
${INSMOD} mtdchar

# linux/drivers/mtd/nand
cd nand
if [ -f ./nand_ecc.o ];then ${INSMOD} nand_ecc; fi
${INSMOD} nand
${INSMOD} alcalz

cd ${BASEDIR}/${ENVDIR}/code/modules_install
cd lib/modules/2.4.17-rt3.1/kernel/fs/jffs
${INSMOD} jffs
# Zlib kernel version not found implies no jffs2
cd ../jffs2
${INSMOD} zlib
${INSMOD} jffs2
cd ${BASEDIR}

if [ "$GDBSCRIPT_PATH" != "" ];then
    cd ${GDBSCRIPT_PATH}
    echo "set remotebaud 115200" > loadall.o
    echo "target remote /dev/term/a" >> loadall.o
    echo -n "add-symbol-file ${BASEDIR}/${ENVDIR}/" >>loadall.o
    echo "code/linux/vmlinux" >>loadall.o
    for I in `ls load_*`;do cat $I;echo "";done >> loadall.o
    echo "To debug, do {"
    echo "     kill remaining tip to the alize"
    echo "     telnet the alize"
```

```
    echo "     gdbstart -s 115200 -t /dev/ttyS0"
    echo "     ddd --debugger i486-pc-linux-kgdb &"
    echo "     change loadall.o if not usign /dev/term/a"
    echo "     (gdb) source ${GDBSCRIPT_PATH}/loadall.o"
    echo "}"
fi

# Add to fstab and mount devpts so that telnet
# dont tell us "all network ports in use"
echo -n "none           /dev/pts        devpts" >> /etc/fstab
echo "  mode=0622       0 0" >> /etc/fstab
mount /dev/pts
```

B.2 alzbzi

```
#!/bin/bash

function printUsage() {
    echo -n "Usage: bash $0 [-rescue] ENVDIR"
    echo ' [NAME] [DHS2NUMBER]'
}

ASSUME_ENVDIR="../claudon1_CLEANSE"

if [ "$1" != "-rescue" ];then
    RAMDISKANDSIZE="initrd_dev.gz;3487830"
    ROOTLINE=
else
    RAMDISKANDSIZE="initrd_rescue.gz;3349798"
    ROOTLINE="root=/dev/hda1,console=ttyS0;28;CL;100;200"
    shift
fi

if [ $# -lt 1 ];then
    printUsage;
    echo ""
    echo "No ENVDIR, assume ${ASSUME_ENVDIR}"
    ENVDIR=${ASSUME_ENVDIR}
else
    ENVDIR=$1
```

```
fi

cd $ENVDIR
GRU=`pwd`
if [ ! -d ${GRU}/code ];then
    echo "${GRU} doesnt seem to be valid I_RTLINUX env"
    exit 0
fi

if [ $# -gt 1 ];then
    NAME=$2
else
    NAME=iok
fi

if [ $# -gt 2 ];then
    ALZ=$3
else
    ALZ=153
fi

if [ $# -gt 3 ];then
    echo 'Warning: too many args'
fi

SRC=$GRU/code/linux/arch/i386/boot
TRG=/tools/linux/tftpboot/alize/dhs2alz${ALZ}.d

/bin/cp -f $SRC/bzImage $TRG/bzImage.${NAME}
SIZ=`wc -c $SRC/bzImage | sed -e 's/^ *\([0-9]*\) .*/\1/'`
cat $TRG/dhs2alz${ALZ}.iok | \
    sed -e 's/bzImage;[0-9]*;/bzImage;${SIZ}/' >! \
    $TRG/dhs2alz${ALZ}.${NAME}
/bin/mv -f $TRG/bzImage $TRG/bzImage.previous
/bin/ln -sf $TRG/bzImage.${NAME} $TRG/bzImage
/bin/rm -f $TRG/dhs2alz${ALZ}.${NAME}
/bin/rm -f $TRG/dhs2alz${ALZ}
/bin/rm -f $TRG/../dhs2alz${ALZ}
echo -n "/tftpboot/alize/dhs2alz${ALZ}.d/bzImage;" > \
     $TRG/dhs2alz${ALZ}.${NAME}
echo -n $SIZ >> $TRG/dhs2alz${ALZ}.${NAME}
echo ";BIN;1046016;1046016" >> \
     $TRG/dhs2alz${ALZ}.${NAME}
```

```
echo "/tftpboot/alize/dhs2alz${ALZ}.d" >> \
     $TRG/dhs2alz${ALZ}.${NAME}
echo "/${RAMDISKANDSIZE};RAMDISK;4194304;0" >> \
     $TRG/dhs2alz${ALZ}.${NAME}
if [ "${ROOTLINE}" != "" ];then
    echo "${ROOTLINE}" >> $TRG/dhs2alz${ALZ}.${NAME}
fi
echo "boot;0;BOOT;0;1048576" >> \
     $TRG/dhs2alz${ALZ}.${NAME}
/bin/cp -f $TRG/dhs2alz${ALZ}.${NAME} \
           $TRG/dhs2alz${ALZ}
if [ -f $TRG/../dhs2alz${ALZ} ];then
    /bin/mv $TRG/../dhs2alz${ALZ} \
            $TRG/../dhs2alz${ALZ}.old`date '+20%y%m%d%H%M'`
fi
/bin/ln -s $TRG/dhs2alz${ALZ} $TRG/../dhs2alz${ALZ}

# modules debuggables
MODINST="$GRU/code/modules_install"
KERMODS_STR="${MODINST}/lib/modules/2.4.17-rt3.1/kernel"
# warning: untested with KERMODS containing spaces in it...
KERMODS=`echo ${KERMODS_STR} | sed -e "s/ /\\ /g"`
echo "KERMODS=${KERMODS}"
# cp ${GRU}/code/linux/fs/jffs2/zlib.o \
#    ${KERMODS}/fs/jffs2
cp ${GRU}/code/linux/lib/zlib_inflate/zlib_inflate.o \
   ${KERMODS}/fs/jffs2
cp ${GRU}/code/linux/lib/zlib_deflate/zlib_deflate.o \
   ${KERMODS}/fs/jffs2
for D in `echo ${KERMODS}/drivers/mtd/ \
               ${KERMODS}/drivers/mtd/nand/ \
               ${KERMODS}/fs/jffs/ ${KERMODS}/fs/jffs2/`
do
    cd $D
    /bin/rm -f *_d.o
    for F in `ls *o`;do
        BASENAME=`basename $F .o`
        i486-pc-linux-ld -d -r $F -o ${BASENAME}_d.o
    done
done

echo "Kernel: bzImage.${NAME}"
echo "From  : ${SRC}"
```

```
echo "To    : ${TRG}"
echo "Tftp  :"
cat $TRG/dhs2alz${ALZ}.${NAME}
```

B.3 insmtd

```
#!/bin/bash
TOPDIR=`pwd`
# only those we need, faster than doing each ones
./mknod /dev/mtd0 c 90 0
./mknod /dev/mtdblock0 b 31 0
cd mtd
insmod -f ./mtdcore.o
insmod -f ./mtdchar.o
insmod -f ./mtdblock.o
insmod -f ./mtdpart.o
cd nand
insmod -f ./nand_ecc.o
insmod -f ./nand.o
insmod -f ./alcalz.o
cd ../../jffs2
insmod ./zlib_inflate.o
insmod ./zlib_deflate.o
insmod ./jffs2.o
```

B.4 Options de compilation du noyau

```
# MTD Memory Technology Devices
CONFIG_MTD=m
CONFIG_MTD_DEBUG=y
CONFIG_MTD_DEBUG_VERBOSE=3
CONFIG_MTD_PARTITIONS=m
CONFIG_MTD_CHAR=m
CONFIG_MTD_BLOCK=m
CONFIG_MTD_NAND=m
CONFIG_MTD_NAND_ECC=y
CONFIG_MTD_NAND_VERIFY_WRITE=y
CONFIG_MTD_NAND_ECC_JFFS2=y
CONFIG_MTD_NAND_ALCALZ=m
# JFFS & JFFS2 Filesystems
CONFIG_JFFS_FS=m
CONFIG_JFFS_FS_VERBOSE=3
CONFIG_JFFS_PROC_FS=y
CONFIG_JFFS2_FS=m
CONFIG_JFFS2_FS_DEBUG=3
CONFIG_JFFS2_FS_NAND=m
# Zlib
# CONFIG_ZLIB_FS_INFLATE is not set
CONFIG_ZLIB_INFLATE=m
CONFIG_ZLIB_DEFLATE=m
```

B.5 Session d'utilisation du driver

```
bash# bash insmtd
alcalz-dbg: Alcalz_Init [Aug 27 2002 11:55:01 BAZ-ECC+JFFS2NAND]
alcalz-dbg: request_region 400-403 done
NAND device: Manufacture ID: 0x98, Chip ID: 0x75 (Toshiba TC58256FT/DC)
Creating 1 MTD partitions on "Toshiba TC58256FT/DC":
0x00000000-0x02000000 : "ALCALZ flash partition"
alcalz-dbg: add_mtd_device done
JFFS2 version 2.1. (C) 2001, 2002 Red Hat, Inc., designed by Axis Communications A

bash# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 02000000 00004000 "ALCALZ flash partition"
```

```
bash# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
02f8-02ff : serial(auto)
03f8-03ff : serial(auto)
0400-0403 : Alcalz Nand flash
0cf8-0cff : PCI conf1
fc00-fcff : PCI device 100b:0020
  fc00-fcff : eth0
fd00-fdff : PCI device 100b:0020
  fd00-fdff : eth1
fe00-fe1f : PCI device 1064:2101

bash# ls /dev/mtd*
/dev/mtd0  /dev/mtdblock0

bash# cd mtd-utils
bash# ./eraseall /dev/mtd0
Erased 32768 Kibyte @ 0 -- 100% complete.

bash# mount -t jffs2 /dev/mtdblock0 /mnt/flash

bash# mount
/dev/ram0 on / type ext2 (rw)
none on /proc type proc (rw)
none on /dev/pts type devpts (rw,mode=0622)
172.25.55.121:/MTD on /mnt/shared type nfs (rw,addr=172.25.55.121,addr=172.25.55.1
/dev/mtdblock0 on /mnt/flash type jffs2 (rw)

bash# touch me
bash# echo etang > narcisse
bash# ls -l
total 0
-rw-r--r--   1 root     root            0 Sep 22 06:28 me
-rw-r--r--   1 root     root            6 Sep 22 06:28 narcisse
```

```
bash# cat narcisse
etang
```

Annexe C

Tests de performances

C.1 Script d’appel à stressFs

```
#!/bin/bash

alias chattr=’echo \!* >& /dev/null’
alias lsattr=’echo \!* >& /dev/null’

function Usage() {
    echo -n "Usage: $0 [LogPrefix] [DevName]"
    echo " [MountPoint] [TimeToRun] [NumberOfInstances]"
    exit 0
}

if [ ! "$1" != "--help" ];then Usage; fi
if [ $# -gt 4 ];then Usage; fi

SMALL_FILE_MIN_SIZE=10
SMALL_FILE_MAX_SIZE=80
BIG_FILE_MIN_SIZE=300
BIG_FILE_MAX_SIZE=400
MIN_USE="1024"
MAX_USE="4096"

if [ "$1" != "" ];then
    LOG_PREFIX=$1
    shift
```

```
else
    LOG_PREFIX="LogFs_"
fi

if [ "$1" != "" ];then
    DEV_NAME=$1
    shift
else
    DEV_NAME="/dev/mtdblock0"
fi

if [ "$1" != "" ];then
    MOUNT_POINT=$1
    shift
else
    MOUNT_POINT="/mnt/flash"
fi

if [ "$1" != "" ];then
    TIME_TO_RUN=21
    shift
else
    TIME_TO_RUN=0
fi

if [ "$1" != "" ];then
    NB_PARA=$1
    shift
else
    NB_PARA=4
fi

echo "($0 Dev=${DEV_NAME} Mnt=${MOUNT_POINT} Ttr=${TIME_TO_RUN}) x${NB_PARA}"
echo "Results in ${LOG_PREFIX}n"
I=0;
while [ $I -lt $NB_PARA ];do
    I=$(( $I + 1 ))
    LOG_NAME="./${LOG_PREFIX}$I"
    ./stressFs  -f  -t ${TIME_TO_RUN} \
                    -s $SMALL_FILE_MIN_SIZE \
                    -S $SMALL_FILE_MAX_SIZE \
                    -b $BIG_FILE_MIN_SIZE \
                    -B $BIG_FILE_MAX_SIZE \
```

```
                    -l$LOG_NAME \
                    -d $DEV_NAME \
                    -m $MOUNT_POINT \
                    -u$MIN_USE \
                    -U$MAX_USE >$LOG_NAME &
done
```

C.2 FFX & ext2

```
Results of WRITE operations
---------------------------
Number of files written : 2485

SMALL files:
  Sum of write      : 95.195312 MBytes
  Best    thoughput : 7.136057 MB/s
  Average thoughput : 1.743164 MB/s
  Worst   thoughput : 0.011843 MB/s
  Number of ERRORS  : 0

BIG files:
  Sum of write      : 95.520508 MBytes
  Best    thoughput : 6.925583 MB/s
  Average thoughput : 1.700195 MB/s
  Worst   thoughput : 0.152265 MB/s
  Number of ERRORS  : 0

Results of READ operations
--------------------------
Number of files read    : 4846

SMALL files:
  Sum of read       : 185.105469 MBytes
  Best    thoughput : 8.296946 MB/s
  Average thoughput : 0.729492 MB/s
  Worst   thoughput : 0.005680 MB/s
  Number of ERRORS  : 0

BIG files:
  Sum of read       : 185.133789 MBytes
```

```
  Best    thoughput : 8.469722 MB/s
  Average thoughput : 1.126953 MB/s
  Worst   thoughput : 0.096867 MB/s
  Number of ERRORS  : 0

Results of REMOVE operations
---------------------------
Number of files removed : 2447

SMALL files:
  Sum of remove     : 93.551758 MBytes
  Best    thoughput : 26.886250 MB/s
  Average thoughput : 0.082031 MB/s
  Worst   thoughput : 0.005196 MB/s
  Number of ERRORS  : 0

BIG files:
  Sum of remove     : 93.628906 MBytes
  Best    thoughput : 25.173882 MB/s
  Average thoughput : 0.365234 MB/s
  Worst   thoughput : 0.067190 MB/s
  Number of ERRORS  : 0
```

C.3 MTD & ext2

```
Results of WRITE operations
---------------------------
Number of files written : 2482

SMALL files:
  Sum of write      : 95.134766 MBytes
  Best    thoughput : 7.011629 MB/s
  Average thoughput : 1.207031 MB/s
  Worst   thoughput : 0.002457 MB/s
  Number of ERRORS  : 0

BIG files:
  Sum of write      : 95.188477 MBytes
  Best    thoughput : 7.042601 MB/s
```

```
  Average thoughput : 0.683594 MB/s
  Worst   thoughput : 0.010793 MB/s
  Number of ERRORS  : 0

Results of READ operations
--------------------------
Number of files read    : 4839

SMALL files:
  Sum of read       : 184.816406 MBytes
  Best    thoughput : 52.759928 MB/s
  Average thoughput : 0.176758 MB/s
  Worst   thoughput : 0.000625 MB/s
  Number of ERRORS  : 0

BIG files:
  Sum of read       : 184.823242 MBytes
  Best    thoughput : 8.478881 MB/s
  Average thoughput : 0.686523 MB/s
  Worst   thoughput : 0.013779 MB/s
  Number of ERRORS  : 0

Results of REMOVE operations
----------------------------
Number of files removed : 2442

SMALL files:
  Sum of remove     : 93.439453 MBytes
  Best    thoughput : 88.329185 MB/s
  Average thoughput : 0.852539 MB/s
  Worst   thoughput : 0.002558 MB/s
  Number of ERRORS  : 0

BIG files:
  Sum of remove     : 93.297852 MBytes
  Best    thoughput : 254.585480 MB/s
  Average thoughput : 2.008789 MB/s
  Worst   thoughput : 0.043991 MB/s
  Number of ERRORS  : 0
```

C.4 MTD & JFFS2

```
Results of WRITE operations
---------------------------
Number of files written : 2472

SMALL files:
  Sum of write      : 94.741211 MBytes
  Best    thoughput : 0.707839 MB/s
  Average thoughput : 0.098633 MB/s
  Worst   thoughput : 0.005021 MB/s
  Number of ERRORS  : 0

BIG files:
  Sum of write      : 94.877930 MBytes
  Best    thoughput : 0.499349 MB/s
  Average thoughput : 0.100586 MB/s
  Worst   thoughput : 0.033384 MB/s
  Number of ERRORS  : 0

Results of READ operations
--------------------------
Number of files read    : 4827

SMALL files:
  Sum of read       : 184.226562 MBytes
  Best    thoughput : 33.227839 MB/s
  Average thoughput : 0.375000 MB/s
  Worst   thoughput : 0.004184 MB/s
  Number of ERRORS  : 0

BIG files:
  Sum of read       : 184.523438 MBytes
  Best    thoughput : 8.393923 MB/s
  Average thoughput : 0.440430 MB/s
  Worst   thoughput : 0.025852 MB/s
  Number of ERRORS  : 0

Results of REMOVE operations
---------------------------
Number of files removed : 2439
```

```
SMALL files:
  Sum of remove     : 93.298828 MBytes
  Best    thoughput : 8.814669 MB/s
  Average thoughput : 0.160156 MB/s
  Worst   thoughput : 0.005554 MB/s
  Number of ERRORS  : 0

BIG files:
  Sum of remove     : 93.297852 MBytes
  Best    thoughput : 3.537023 MB/s
  Average thoughput : 0.393555 MB/s
  Worst   thoughput : 0.090400 MB/s
  Number of ERRORS  : 0
```