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

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

www.hevs.ch

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2008

Yvan Epiney

Système embarqué sur processeur SAM9

Professeur	Pierre Pompili
Expert	Daniel Rossier

Sion, le 21 novembre 2008

HES-SO Valais

SI	TV	EE	IG	EST
X	X			

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.DB
mam/30/06/2008

Confidentiel / *Vertraulich* ☐ oui / ja ☒ non / nein

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2007/2008**	No PS / *Nr. PS* **it/2008/20**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Ecole hôte	Etudiant / *Student* **Yvan Epiney**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Ecole hôte
Professeur / *Dozent* **Pierre Pompili**	Expert / *Experte (données complètes)* **Daniel Rossier**	

Titre / *Titel*

Système embarqué sur processeur SAM9

Description et Objectifs / *Beschreibung und Ziele*

Les systèmes embarqués sont au cœur des produits tels les consoles de jeu, les appareils-photo numériques, les unités de contrôles dans l'automobile ou les systèmes multimédias. Toutes ces applications ont en commun d'exécuter des calculs intensifs, d'être hautement réactif et de consommer un minimum d'énergie.

Les processeurs ARM9 de dernière génération intègrent dans leur cœur une technologie RISC, des unités de traitement spécialisées et des bus parallèles distribués associés à des contrôleurs DMA dans le but d'éviter les goulets d'étranglement qui se produisent habituellement dans les processeurs.

Le processeur 91SAM9263 d'ATMEL, basé sur un cœur ARM9, intègre dans un seul circuit pratiquement toutes fonctionnalités exigées pour les systèmes embarqués multimédia : contrôleur LCD, coprocesseur graphique, interfaces USB, interface audio et contrôleurs DMA.

L'objectif du travail de diplôme est de développer une carte-mère à base du processeur 91SAM9263, avec comme but ultime, le portage d'un noyau Linux.

— Mise en route de la carte développée lors du projet de semestre
— Portage d'un moniteur minimal
— Avaluation des performances liées à l'utilisation des bus parallèles
— Portage du noyau Linux.

Signature ou visa / *Unterschrift oder Visum* Resp. de la filière *Leiter des Studieng.:* ..... Etudiant/*Student*:	Délais / *Termine* Attribution du thème / *Ausgabe des Auftrags:* 01.09.2008 Remise du rapport / *Abgabe des Schlussberichts:* 21.11.2008, 12:00 Exposition publique / *Ausstellung Diplomarbeiten:* 28.11.2008 Défense orale / *Mündliche Verfechtung:* semaine/Woche 49

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

Yvan Epiney 21 novembre 2008

Système embarqué sur processeur SAM9

SAM9 basierde Embedded System

Objectif

Développer et réaliser une carte mère multimédia à base d'un microcontrôleur AT91SAM9263.

Porter un moniteur minimal ainsi que le système GNU/Linux sur la nouvelle carte.

Evaluer les performances du microcontrôleur et en particulier les gains liés à l'utilisation de ses capacités de traitement parallèle.

Résultats

Développer et réaliser une carte mère multimédia à base d'un microcontrôleur AT91SAM9263.

Porter un moniteur minimal ainsi que le système GNU/Linux sur la nouvelle carte.

Evaluer les performances du microcontrôleur et en particulier les gains liés à l'utilisation de ses capacités de traitement parallèle.

Mots-clés

Système embarqué, AT91SAM9263, AT91Bootstrap, U-Boot, Linux pour ARM, NFS

Ziel

Entwicklung und Durchführung einer Mainboard die auf einem AT91SAM9263 Microkontroller basiert ist.

Adaptierung eines Monitors sowie das GNU / Linux-System auf die neue Karte.

Bewerbung der Leistung des Mikrocontrollers und insbesondere die Gewinne im Zusammenhang mit der Nutzung von Kapazitäten für die Parallelverarbeitung

Reslutate

Ein Motherboard wurde entwickelt und die folgenden Elemente wurden getestet: Speisung, JTAG, NOR Flash, RAM, RS232 und Ethernet.

U-boot wurde als Monitor adaptiert. Sowie ein funktionelles GNU/Linux System, mit einem Root File System erreichbar durch NFS .

Die Leistungen konnten leider nicht bewertet werden.

Schlüsselwörter

Embedded System, AT91SAM9263, AT91Bootstrap, U-Boot, Linux für ARM, NFS

AT91SAM9263 embedded system

Projet de diplôme 2008
Etudiant : Epiney Yvan
Enseignant : Pompili Pierre

Hes·so VALAIS WALLIS

Table des matières

Préface

Ce rapport a pour but d'expliquer le travail effectué durant mon projet de diplôme.

J'espère qu'il pourra aider les personnes devant travailler avec le microcontrôleur AT91SAM9263 dans le futur en leur exposant certain pièges/bugs de ce composant.

De même ce rapport peut être utile aux développeurs intéressés aux sujets suivants:

- Développement croisé
- Bootstrap
- U-Boot
- Linux pour ARM

Remerciements

Je tiens à remercier l'ensemble de la section Infotronic et plus particulièrement Pierre Pompili et Marc Pignat ainsi qu'Olivier Walpen et Steve Gallay de l'atelier électronique pour leur aide. Je souhaite également remercier mon expert, Daniel Rossier pour le temps consacré à la lecture de ce rapport.

Merci également à ma mère pour la correction de celui-ci, à mon frère pour l'impression couleur et à l'ensemble de mes camardes pour leur soutient dans l'adversité et les bons moments passés.

Sans oublier l'ensemble de communauté open source pour leur aide et leurs efforts constants dans le développement de solutions ouvertes et performantes. Bien qu'impossible de les citer tous, un grand coup de chapeau aux développeurs du noyau Linux, de Debian et aux utilisateurs des sites AT91portal et Linux4Sam.

1 Introduction

Les systèmes embarqués tendent à devenir omniprésents dans la vie de tous. Du navigateur GPS à la console de jeu en passant par les natels et autres appareils photos, ces produits ont comme dénominateur commun toujours plus de puissance de calcul pour une consommation de plus en plus faible.

Pour ce type d'application, Atmel propose la famille de microcontrôleurs AT91SAM9. Ces derniers intègrent en plus de leur coeur ARM9 un nombre impressionnant de contrôleurs dans un seul composant. Avec leurs interfaces LCD, audio, USB, carte mémoire et leur coprocesseur graphique, les AT91SAM9 sont idéaux pour le développement de solution embarquée multimédia.

L'objectif de ce travail de diplôme est de développer une carte-mère basée sur le microcontrôleur AT91SAM9263, avec en ligne de mire, le portage d'un système GNU/Linux.

En résumé les tâches suivantes sont demandées:

- Mise en route de la carte développée durant le projet de semestre
- Portage d'un moniteur minimal
- Portage d'un système GNU/Linux
- Evaluation des performances liées à l'utilisation des bus parallèles

2 Conception de la carte

2.1 Architecture générale

L'architecture logique de la carte s'organise naturellement autour du **AT91SAM9263** puisque ce dernier intègre la quasi totalité des microcontrôleurs requis en plus du coeur ARM9.

Cependant plusieurs composants et connecteurs doivent être ajoutés pour obtenir un système opérationnel. Tout d'abord un circuit d'alimentation capable de fournir l'énergie à l'ensemble de la carte, de la mémoire et des transceivers (phy) qui permettent de faire la conversion entre les signaux propres au microcontrôleur et ceux transmis sur les lignes (niveaux logiques, fréquence, ...).

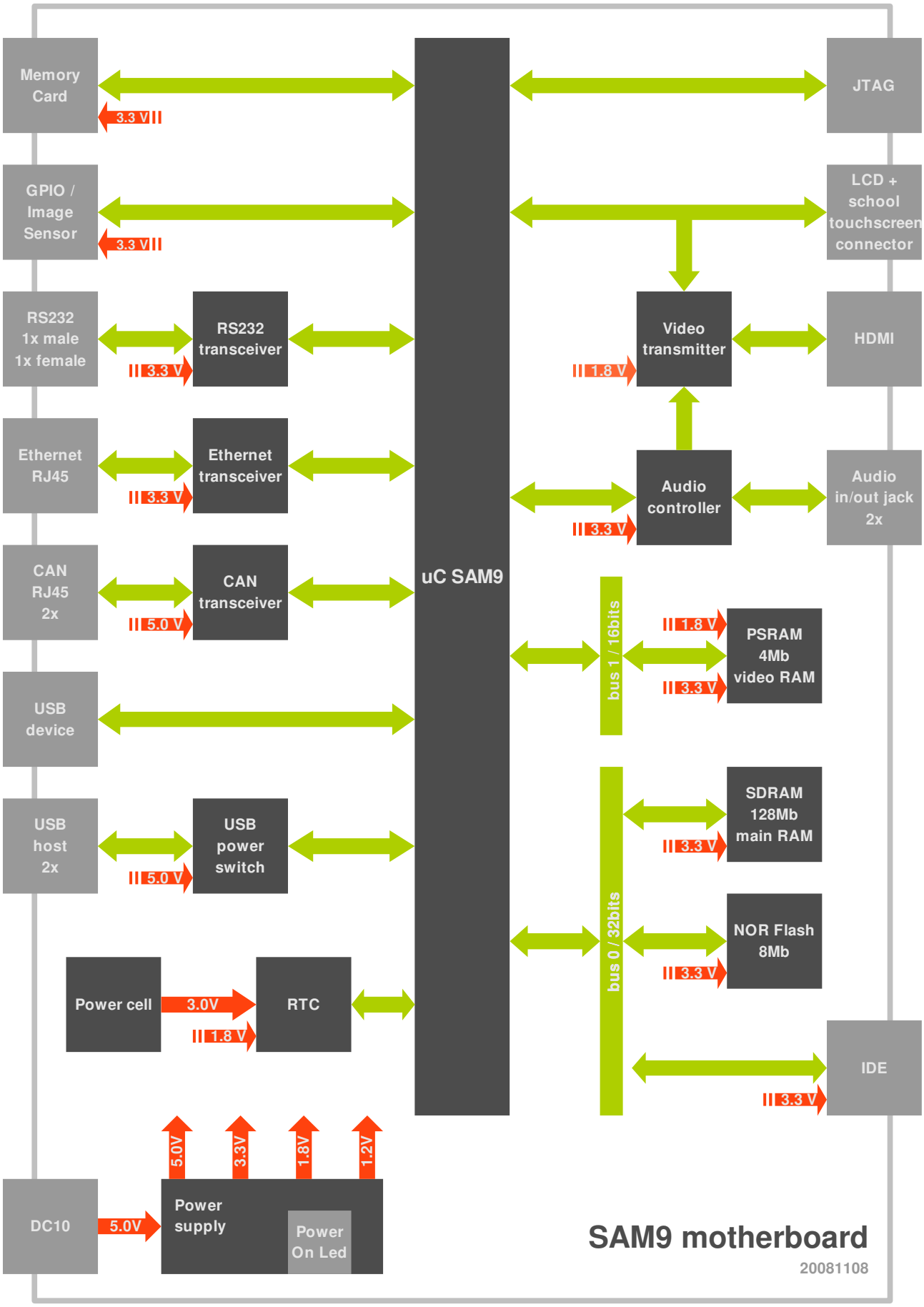
De plus bien que disposant d'un contrôleur audio AC97 (codec audio) le SAM9 ne peut fournir des signaux audio directement utilisables. C'est pourquoi nous disposons d'un contrôleur audio externe.

De même avec les signaux vidéo. Le contrôleur LCD interne au SAM9 fournit des signaux en RGB plus les signaux de contrôle. Mais aujourd'hui peu d'appareils grand public sont capables de les utiliser directement. La carte fournira donc en plus du classique RGB des signaux HDMI™ au travers d'un contrôleur vidéo. Le choix s'est porté sur le HDMI™ car celui-ci tend à devenir l'interface standard dans le multimédia et possède une connectique plus simple que le DVI tout en lui étant compatible à l'aide d'un adaptateur.

En résumé la carte disposera de :

- un circuit d'alimentation
- un microcontrôleur SAM9
- une interface de programmation/débugage JTAG
- une horloge temps réel
- une mémoire principale SDRAM
- une mémoire secondaire PSRAM faisant office de RAM vidéo
- une mémoire Flash
- un émetteur vidéo HDMI™
- un contrôleur audio
- deux ports séries RS232
- une interface Ethernet 10/100 Mbits/s
- une interface CAN
- trois connecteurs USB, 2 host et 1 device
- un slot pour carte mémoire
- un connecteur IDE pour disque dur externe
- un connecteur pour capteur d'image CMOS, faisant également office de GPIO
- une interface pour la gestion d'écran tactile
- quatre boutons et cinq LEDs

La page suivante présente une vue générale des composants intégrés sur la carte.


Memory Card
3.3 V
GPIO / Image Sensor
3.3 V
RS232 1x male 1x female
RS232 transceiver
3.3 V
Ethernet RJ45
Ethernet transceiver
3.3 V
CAN RJ45 2x
CAN transceiver
5.0 V
USB device
USB host 2x
USB power switch
5.0 V
Power cell
3.0V
1.8 V
RTC
uC SAM9
JTAG
LCD + school touchscreen connector
Video transmitter
1.8 V
HDMI
Audio controller
3.3 V
Audio in/out jack 2x
bus 1 / 16bits
1.8 V
3.3 V
PSRAM 4Mb video RAM
bus 0 / 32bits
SDRAM 128Mb main RAM
3.3 V
NOR Flash 8Mb
3.3 V
IDE
3.3 V
5.0V
3.3V
1.8V
1.2V
DC10
5.0V
Power supply
Power On Led
SAM9 motherboard
20081108

2.2 Alimentation électriques

Le circuit d'alimentation doit pouvoir fournir à l'ensemble des composants présents sur la carte les niveaux de tension requis avec des courants suffisants et un minimum de bruit. De plus il faut également veiller à disposer d'un rendement acceptable.

La tension de 5.0V est directement fournie par l'alimentation externe qui est également de 5.0V et est uniquement filtrée à l'aide d'une ferrite.

Le 3.3V est fournit par un convertisseur step-down, le **ADP2107ACPZ-3.3**. Un tel convertisseur a été choisi car ceux-ci disposent d'un très bon rendement à ces niveaux de tension 5.0V → 3.3V. Cependant il nécessite un design du PCB spécifique ainsi que l'utilisation de condensateurs spéciaux (low ESR). La tension est filtrée à l'entrée et à la sortie du composant.

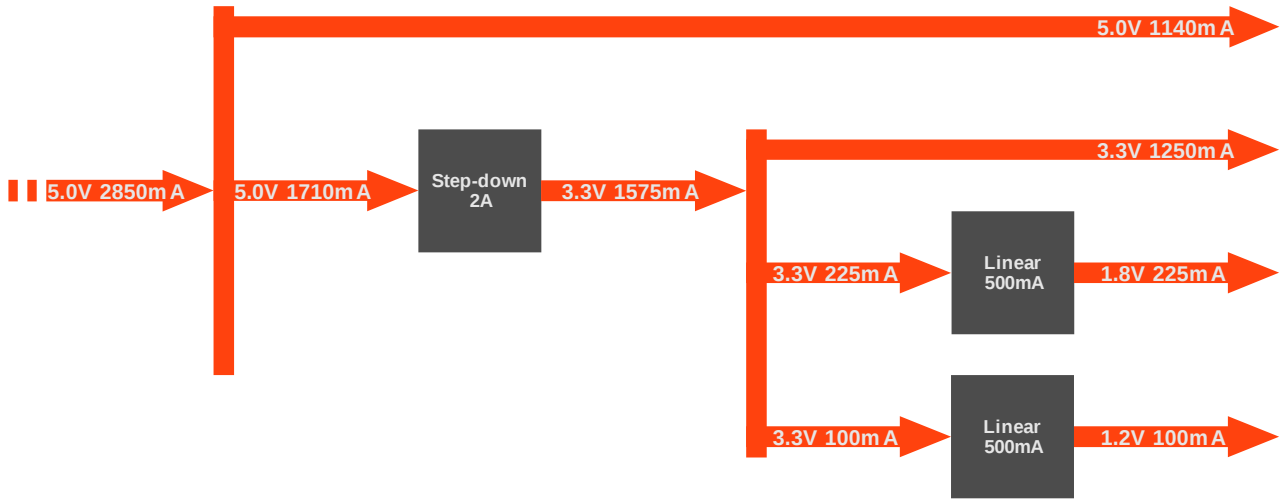
Les tensions de 1.8V et 1.2V sont fournies chacune par un convertisseur linéaire le **ADP1715ARMZ**. Des convertisseurs linéaires ont étés choisis car ils sont plus simples que des step-down et que le rendement de ces derniers devient médiocre à basse tension. La tension d'entrée est de 3.3V plutôt que 5.0V afin de minimiser les pertes qui sont égales au courant fois la chute de tension. Les tensions sont également filtrées à l'entrée et à la sortie du composant. La tension de sortie de ce composant est réglée par un pont diviseur selon la formule suivante:

$$Vout = 0.8\left(1 + \frac{R1}{R2}\right)$$

En fonction de cette équation les couples de résistances 150/120 Ω pour le 1.8V et 100/200 Ω pour le 1.2V sont utilisés.

Le dimensionnement du circuit a été fait en fonction du pire des cas (worst case design) où toutes les fonctionnalités de la carte sont activées et où chaque composant exécute son opération la plus consommatrice de courant. Bien que cette situation soit hautement improbable il n'est pas acceptable de risquer un comportement incohérent ou l'arrêt de certains composants à cause d'une alimentation insuffisante.

De plus certaines parties ont été volontairement surdimensionnées pour des raisons de coûts (composant produit à plus large échelle) et de simplicité (composant identique). Par exemple l'étage 3.3V → 1.2V peut fournir 500mA soit bien plus que les 105mA nécessaires ceci afin d'avoir un montage identique à l'étage 3.3V → 1.8V, à la boucle de contre-réaction près.

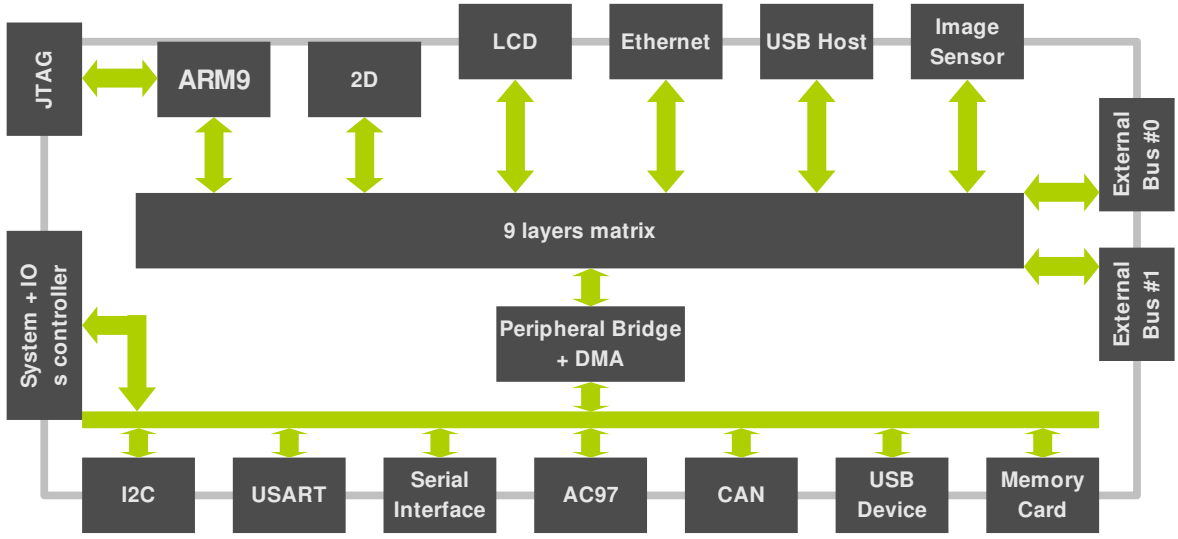


Alimentation électrique, niveaux de tensions et courants maximums

L'ensemble des valeurs ayant permis le dimensionnement de l'alimentation se trouve en annexe C, page 69.

2.3 Micro-contrôleur

L'ensemble de la carte s'organise autour microcontrôleur **AT91SAM9263**. L'image suivante est une simplification de l'architecture interne du microcontrôleur où seules les fonctionnalités utilisées ont étés représentées.



Microcontrôleur AT91SAM9263, architecture simplifiée

Le coeur du microcontrôleur est composé d'une matrice de 9 bus 32 bits permettant l'échange de données entre les différents contrôleurs haute vitesse. Chacun d'entre eux dispose d'un contrôleur DMA (direct memory access) afin de pouvoir lui déléguer les transferts de données et se concentrer sur les traitements.

L'unité centrale est un processeur RISC **ARM926EJ-S** 32 bits à 5 niveaux de pipeline capable d'opérer jusqu'à 240MHz. De plus ce dernier dispose d'une MMU (memory management unit) et d'un support pour l'exécution de code Java (Jazelle).

Le micro-contrôleur intègre un contrôleur graphique 2D afin de décharger le CPU des opérations graphiques simples : dessin de lignes, polygones et transfert de block à l'intérieur du framebufffer.

Les périphériques séries sont interconnectées à la matrice 9 bus par l'intermédiaire d'un pont.

Notez qu'une part non négligeable des pins du microcontrôleur est multiplexée, une pin a plus d'une utilité. Ce multiplexage influe de manière importante sur le choix des fonctionnalités.

Les pins non assignées peuvent être utilisées comme entrée/sortie généraliste par l'intermédiaire du contrôleur d'entrée/sortie (PIO controler). Elles peuvent en plus générer des interruptions lors d'un changement de niveau logique ou être configurées comme pullup.

L'AT91SAM9263 nécessite en plus de l'alimentation électrique de deux oscillateurs, un principal et un second pour le démarrage et le mode veille. Ce dernier doit toujours osciller à 32'768 Hz. Quand à l'oscillateur principal, le choix est libre. Celui-ci s'est porté sur oscillateur à 16MHz, ce qui permet de configurer facilement les PLLs afin d'avoir une fréquence ronde.

2.4 JTAG

Développé initialement pour le test des courts-circuit sur les cartes électroniques, le JTAG a évolué au fil des ans comme une interface de programmation et de débugage de uC.

L'AT91SAM9263 intègre en plus du JTAG un support de débugage plus évolué, nommé embeded trace module (ETM). Cependant ETM empêche d'utiliser pleinement le bus #0, de plus l'école ne dispose pas de matériel compatible. C'est pourquoi seul le JTAG est implémenté.

Il est nécessaire d'avoir un adaptateur externe entre le PC qui communique le plus souvent à l'aide d'une interface USB et l'interface JTAG de la carte.

Le connecteur sur la carte est de type 2x5 afin d'être compatible avec les outils utilisés pour la carte ARMEBS3. Le pining du connecteur JTAG se trouve à la page JTAG.

L'interface JTAG utilisée est le BDI2000 d'Abatron AG. Son utilisation est décrite au chapitre 5, page 30.

Notez que le signal *BOARD_RESET* est bidirectionnel. Cela signifie qu'il permet à l'interface de développement d'être informé d'un reset de l'utilisateur ou de procéder elle-même à un reset de la carte.

JTAG, schéma de principe

2.5 Horloge temps réel

Dans tout système moderne la gestion du temps tient un rôle important. Celle-ci s'étend du calcul d'intervalle de temps très court, pour la commutation de processus, jusqu'au maintient de l'heure utilisée par nombre de logiciels, tant système qu'utilisateur.

Malheureusement le SAM9 ne dispose que de timer temps réel (RTT) et non d'une horloge temps réel (RTC). Concrètement cela signifie que les RTTs sont réinitialisées lors du reset du uC et ne peuvent garder une trace de l'heure courante. C'est pourquoi nous disposons d'une RTC externe, le **DS1374U-18**

Ce composant dispose de deux alimentations, l'une principale et la seconde de sauvegarde afin de continuer à fonctionner même lorsque la carte n'est pas alimentée. La commutation entre les deux sources s'effectue automatiquement lorsque le niveau de tension de l'alimentation principale tombe en dessous d'une certaine valeur, typiquement 1.6V dans notre cas.

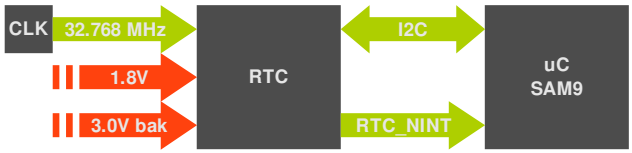
L'horloge temps réel se compose du chip lui-même plus un quartz externe à 32.768kHz. La communication entre le uC et la RTC s'effectue au travers d'une connexion I^2C. L'entrée reset *RST* dispose en interne d'une pullup et peut être laissée non connectée. De plus le composant peut émettre une interruption à destination du uC lorsque l'horloge a atteint un certain temps.

En plus de son compteur principal sur 32 bits, le DS1374U-18 dispose d'un second compteur sur 24 bits

utilisable comme alarme ou watchdog. Si ce dernier est inutilisé, il peut servir de petite mémoire permanente.

Notez que ce composant existe en trois variantes -18, -3 et -33 qui correspondent respectivement aux tensions d'alimentation 1.8V, 3.0V et 3.3V. Etant donné que nous disposons du 1.8V et que plus la tension d'alimentation est basse moins le composant consomme de courant, le -18 a été choisi.

L'alimentation de sauvegarde est composée d'une pile bouton pouvant fournir 325mAh. Etant donné que la RTC consomme 700nA en mode veille, la pile est apte à l'alimenter durant 53 années (déchargement dû au vieillissement négligé, < 1% / année).



Horloge temps réel, schéma de principe

2.6 Mémoires

Le SAM9 possède deux bus, EBI0 et EBI1 (external bus interface) selon la dénomination d'Atmel. Ces bus permettent d'interfacer au microcontrôleur de la mémoire, des coprocesseurs et/ou des cartes d'extension (mezza).

La mémoire principale est connectée sur le bus #0. Il s'agit de quatre barrettes de 32MB de RAM Samsung **K45560832E-TC75**. Chacune d'elles fournit 8 bits de données et c'est en jouant sur les adresses qu'il est possible d'avoir un bus de 32 bits. La taille de la RAM est de 128MB.

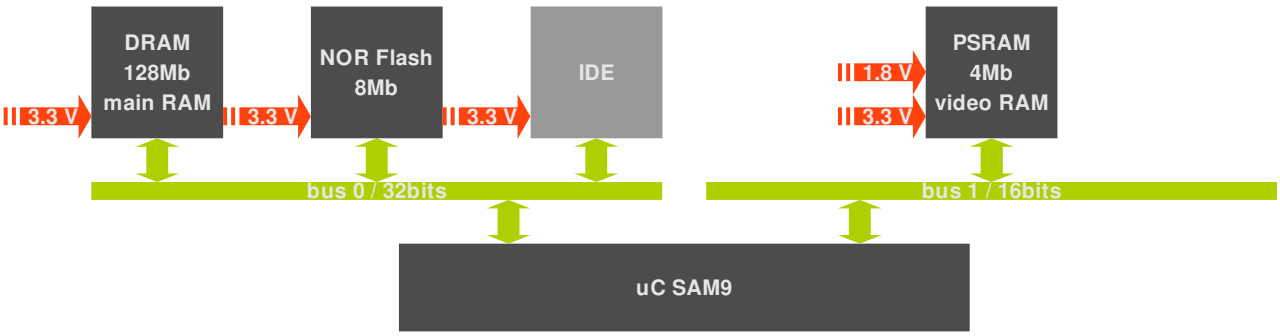
De la mémoire non volatile est également connectée sur le bus #0, de la Flash NOR d'Intel **TE28F128-J3** d'une capacité de 4MB.

Ces deux composants proviennent des surplus de l'école et n'ont pas fait l'objet d'un choix

Le bus mémoire #0 intègre aussi un connecteur **IDE** pour pouvoir connecter un disque dur externe.

Bien qu'il ait été intéressant de délester une partie des fonctionnalités du premier bus sur le second, plusieurs conflits de multiplexage (en particulier avec le contrôleur Ethernet) nous oblige à utiliser ce dernier en mode 16 bits uniquement. Du coup le second bus perd passablement de son intérêt.

Cependant ce dernier est tout de même utilisé afin d'y connecter la RAM vidéo, de la PRAM **MT45W2MW16PGA** à 4MB . Ceci permet de réduire le goulet d'étranglement dû à la mémoire puisque le système peut effectuer simultanément des opérations en mémoire principale et en mémoire vidéo.



Bus mémoire, schéma de principe

2.7 Multimedia

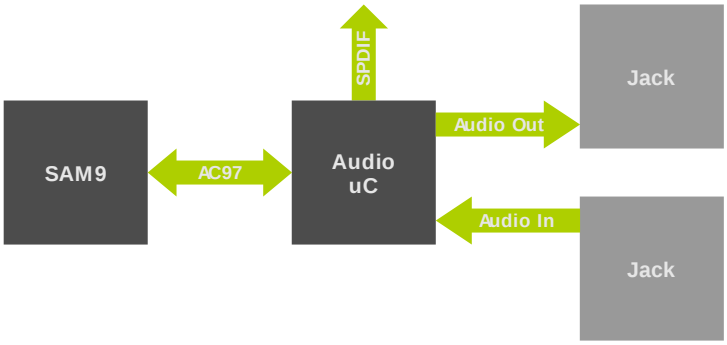
2.7.1 Audio

L'AT91SAM9263 intègre un contrôleur digital audio permettant de gérer le codec AC97. Les signaux AC97 étant purement digitaux, ils doivent être adaptés par un contrôleur, le **AD1981B**.

La carte dispose de deux connecteurs JACK, un pour la sortie audio et le second pour l'entrée audio.

De plus le contrôleur audio possède une sortie SPDIF qui est réutilisée par le contrôleur vidéo puisque les signaux HDMI sont composites, image et son.

Le pining des connecteurs JACK est disponible à la page 72.



Audio, schéma de principe

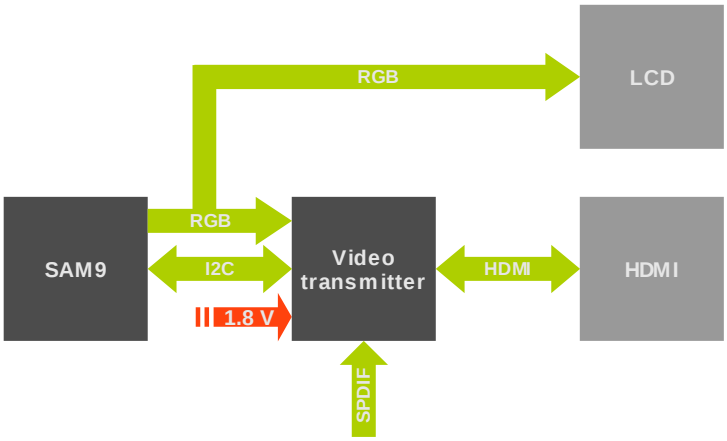
2.7.2 Video

L'émetteur vidéo de l'AT91SAM9263 fournit des signaux en RGB. Bien qu'il soit possible d'atteindre une qualité de 24bpp (bits par pixel) des conflits dû au multiplexage des entrées/sorties nous forcent à nous contenter de 16bpp.

Ces signaux peuvent directement être connectés à un écran LCD, mais comme mentionné au chapitre 2.1, il a été décidé de disposer également d'une sortie HDMI. L'émetteur retenu est le **AD9889**. La configuration de ce dernier s'effectue par le bus I²C.

De plus bien qu'acceptant des entrées audio/vidéo aux niveaux logiques 3.3V, les signaux de commande du composant utilisent les niveaux 1.8V. Les entrées sont donc adaptées par l'intermédiaire d'un pont diviseur. Les sorties par un level translator.

Le pining du connecteur HDMI est disponible à la page 72, tandis que le connecteur LCD à la page 71.



Vidéo, schéma de principe

2.8 Interfaces réseaux

2.8.1 Ethernet

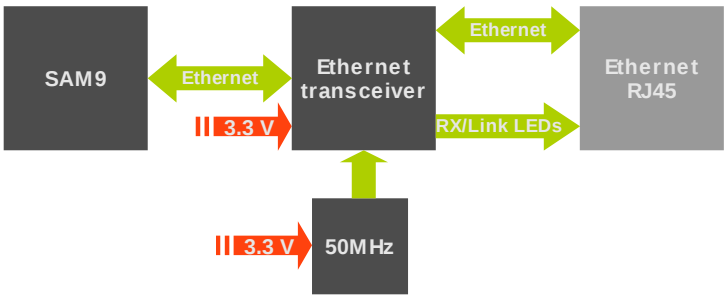
Nul besoin de présenter Ethernet tant il s'est imposé comme le LAN par excellence.

Le SAM9 intègre un contrôleur Ethernet 10/100 Mbit/s, cependant les signaux doivent être adaptés par un transceiver avant d'être transmis.

Deux types de transceivers existent, les MII (media independent interface) et les RMII (reduced media independent interface). Comme son nom l'indique, un RMII nécessite moins de ligne d'interfaçage, de 6 à 10 au lieu de 16 pour un MII. En contrepartie l'horloge d'un transceiver RMII doit avoir une fréquence double à celui d'un MII, soit 50MHz.

Etant donné que certains signaux Ethernet sont multiplexés avec des signaux vidéo RGB, un transceiver RMII a été choisi, le **DP83848C**.

Pour Ethernet utilise un connecteur RJ45 spécial avec filtre et séparation galvanique interne. De plus ce dernier dispose de deux LEDs de notification pilotées par les signaux *ETH_ACTIVITY_LED* et *ETH_LINK_LED,* fournis par le transceiver, signalant respectivement l'envoi/réception de données et la présence au sein d'un réseau Ethernet.



Ethernet, schéma de principe

La configuration du contrôleur Ethernet s'effectue lors du reset de la carte à l'aide de résistance de 2.2kΩ soit en pullup, soit en pulldown (strap options). Pour une utilisation en mode RMII avec autonégotiation de la vitesse la pin #39 (MII_MODE) doit être mis à la masse au travers d'une pulldown.

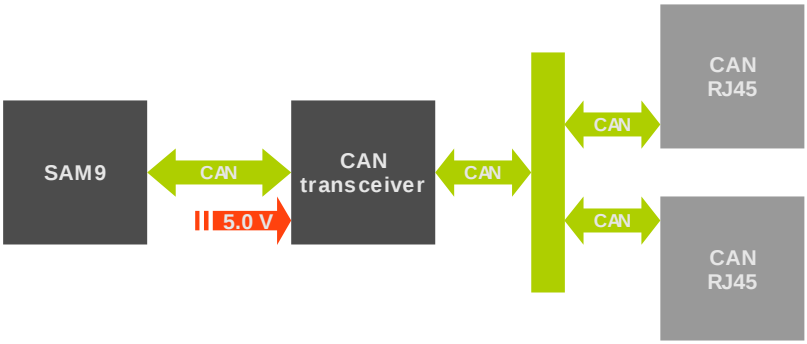
2.8.2 CAN

Développé initialement par Bosch dans le but de réduire le câblage interne des véhicules, le CAN de part ses qualités: robustesse, simplicité, faible coût, s'est imposé comme l'un des acteurs majeurs des bus de terrain.

De même que les signaux ethernet, ceux CAN ne peuvent être transmis tel quel sur le bus mais doivent être adaptés par un transceiver. Ce dernier transforme les signaux logiques en une paire de tension codant un symbole (high et low) puisque le CAN utilise un codage différentiel lui assurant une bonne immunité électromagnétique.

Le transceiver retenu est le **MCP2551**. Bien qu'il soit possible d'alimenter ce composant avec une tension de 3.3V, cette solution ne garantit pas le respect de niveau de tension définit dans la norme CAN (ISO-11898). C'est pourquoi il est alimenté en 5.0V.

Deux connecteurs RJ45 simples sont implémentés sur la carte, ceci afin de pouvoir se greffer n'importe où dans un bus CAN. Le pining, disponible à la page 73, est identique à celui utilisé au sein de l'école.



CAN, schéma de principe

2.9 Entrées / Sorties

2.9.1 Ports séries

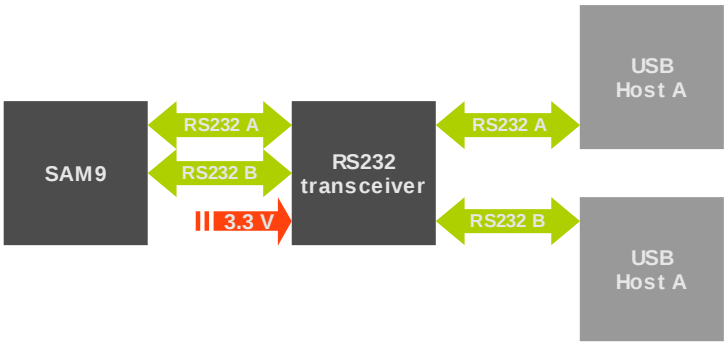
Bien que de plus en plus souvent remplacé par l'USB sur les PC modernes, le port série reste de part sa simplicité et sa robustesse le compagnon fidèle du programmeur, en particulier au travers d'un émulateur de terminal.

Le contrôleur USART intégré au SAM9 peut gérer jusqu'à 3 ports séries. Cependant à cause du multiplexage des périphériques (conflit avec MCI0) et de la relative inutilité d'un troisième port série, seul deux sont implémentés. L'USART0 et l'USART2 désignés respectivement comme port série A et port série B.

Les signaux séries du uC SAM9 ne peuvent être directement connectés au port série mais doivent être adaptés par un transceiver. Le **ADM3202ARN** a été retenu de par sa disponibilité et son faible prix.

Seuls les signaux tx (envoi) et rx (réception) sont câblés étant donné que les autres sont très rarement utilisés. Puisque le transceiver peut gérer deux entrées et deux sorties, un unique transceiver est partagé entre le deux ports séries.

Deux connecteurs différents sont utilisés. Un de type DE9 femelle, identiques à ceux présents sur les périphériques. Le second de type DE9 mâle, similaire à ceux présents sur un PC.



Ports séries, schéma de principe

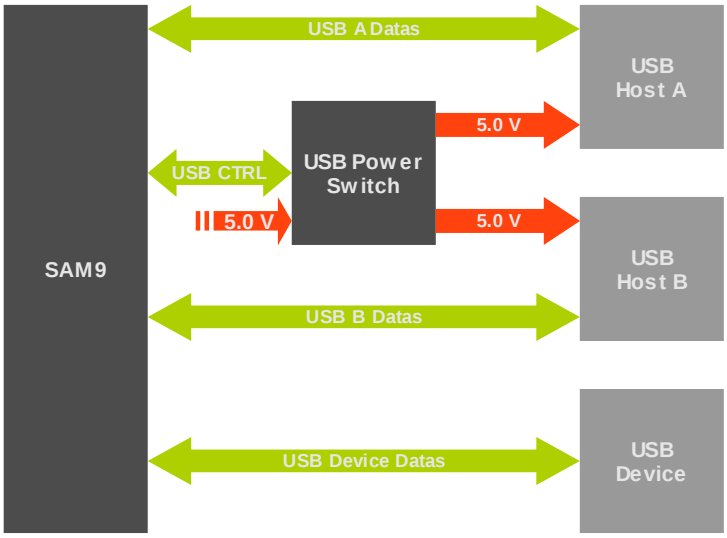
2.9.2 USB

Avec le SAM9 il est possible de disposer de 3 connecteurs USB, deux en mode host (master) et un en mode device (slave). Au vu de l'importance croissante de l'USB**,** tous sont implémentés. Les signaux USB ne nécessitent pas d'adaptation, juste un filtrage passif.

L'USBen mode device rend par exemple possible d'utiliser la carte comme périphérique pour PC. En mode device, la pin d'alimentation est connectée au SAM9 via une résistance de protection afin de pouvoir générer une interruption lors de la connexion de la carte comme esclave, signal *ETH_DEV_CNX*.

En mode host un composant supplémentaire, le power switch **LM3526M-L** permet de piloter l'alimentation des ports USB et de détecter le branchement à chaud de périphériques. Les signaux *ETH_n_CNX* indiquent qu'un périphérique est connecté et les *~ETH_n_EN* permettent de désactiver ou non l'alimentation de ce dernier. Cette fonctionnalité peut s'avérer utile afin de monter et démonter automatiquement et sans risque une clef USB par exemple.

Le pining des ports **USB** est disponible en page 73.



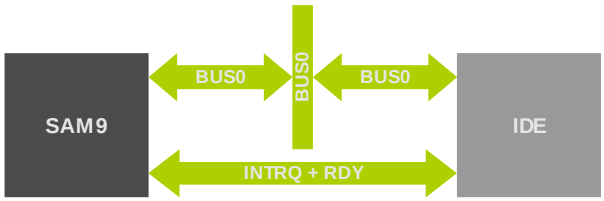
USB, schéma de principe

2.9.3 IDE

Pour augmenter la mémoire de masse de la carte sans obligatoirement passer pas le réseau (NFS), il est possible de connecter un disque dur IDE. Le connecteur **IDE** partage le bus 0 avec la mémoire principale et la mémoire Flash. De plus les signaux supplémentaires interrupt request *IDE_INTRQ* et ready *IDE_RDY* sont directement connectés au AT91SAM9263.

Notez qu'il est nécessaire d'alimenter de manière externe le disque dur car aucune alimentation n'est fournie par le connecteur.

Le pining du connecteur IDE est disponible à la page 74.



IDE, schéma de principe

2.9.4 Memory card

Les cartes mémoire deviennent de plus courantes sur nombre d'appareil mobile, natel, console de jeux, appareil photo, ... C'est donc tout naturellement que l'AT91SAM9263 dispose d'un contrôleur interne pour ce type d'équipement : le multimedia card interface (MCI). Ce contrôleur peut gérer les cartes SD et MMC. Les signaux peuvent directement être reliés au connecteur MCI.

La pin #11 est connectée en tant qu'entrée sur le SAM9 afin de détecter l'insertion ou le retrait d'une carte.

Les autres pins correspondent aux adresses/données et signaux de contrôle standards des cartes multimédia. La connectique complète est disponible à la page 73.

Connecteur SDCard, schéma de principe

2.9.5 Capteur d'image / GPIO

Le SAM9 dispose en interne d'un contrôleur de caméra CMOS (ISI). Cependant afin de ne pas alourdir inutilement la connectique de la carte, il a été décidé de le multiplexer avec le GPIO (general purpose input output).

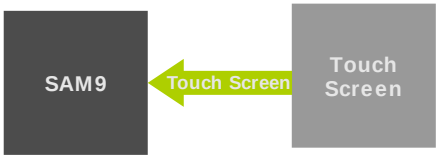
Le pinning du port GPIO est compatible avec les périphériques développés à l'école et est diponible à la page 73.

GPIO / ISI, schéma de principe

2.9.6 Touch screen

Etant donné qu'il est possible d'utiliser directement (sans mezza) les écrans de laboratoire il est intéressant de pouvoir aussi accéder nativement à leurs fonctionnalités tactiles. Ceci est possible grâce au contrôleur série (SPI) intégré au AT91SAM9263.

Le pinning du touch screen est disponible avec celui du connecteur LCD à la page 71.



Touch screen, schéma de principe

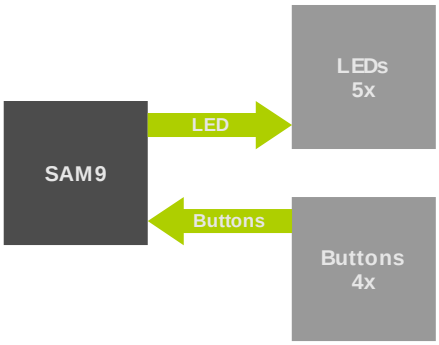
2.9.7 Boutons / LEDs

La carte dispose de 4 boutons et de 4 LEDs configurables par l'utilisateur. Chacun des boutons est susceptible de générer une interruption via le PIO contrôler. Ils sont actifs à l'état bas.

Les LEDs choisies (597-3xx1-5xx) fonctionnent avec un courant nominal de 20mA ce qui n'est pas un

problème pour la LED d'alimentation. Cependant l'AT91SAM9263 ne peut fournir plus de 8mA en sortie. Mais pour ne pas complexifier la carte, il a été décidé après test de ne pas rajouter de transistor, les LEDs s'allumant suffisamment.

De plus une 5ème LED est à la disposition du système d'exploitation comme LED d'activité. En cas de non utilisation cette dernière peut être utilisée comme les 4 autres.



LEDs, boutons, schéma de principe

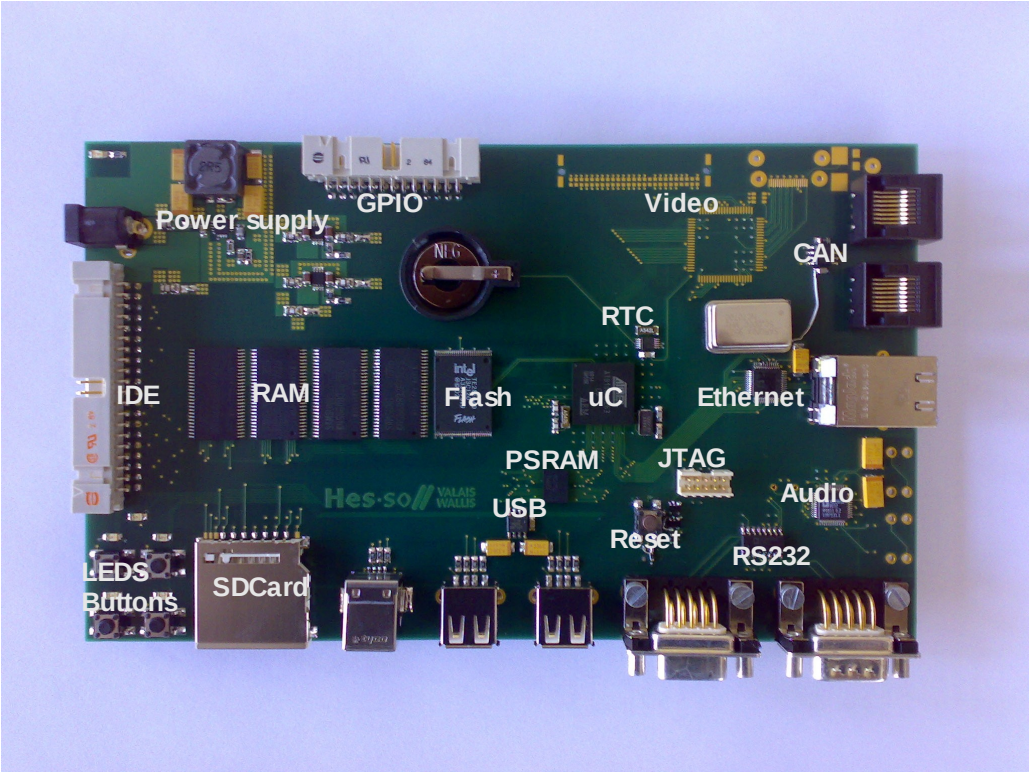
2.10 Références

- Datasheets relatifs aux différants composants:
  cdrom/board/datasheets/
- Carte ARMEBS3 de l'Hes-so/Valais et mezza assosciées:
  cdrom/board/doc/armebs3/
- Documentaion de la carte d'évaluation d'Atmel AT91SAM9263_EK:
  cdrom/board/doc/Evaluation_board_AT91SAM9263_EK.pdf

3 Réalisation de la carte

Le routage a été effectué par Olivier Walpen de l'atelier électronique. Huit couches ont étés nécessaires pour pouvoir connecter l'ensemble des composants. Il n'était donc pas possible de le réaliser avec les moyens de l'école. C'est pourquoi le PCB a été réalisé par l'entreprise Euro Circuit. Olivier Walpen a également soudé les deux BGAs ainsi que le circuit d'alimentation. Le reste de la carte a été monté par mes soins.

L'image suivante montre l'implémentation des circuit-intégrés et des connecteurs sur la carte.



Implémentation des composants sur la carte

On peut remarquer que la partie vidéo n'a pas été montée par manque de temps. Cependant rien ne l'empêche une fois le système opérationnel.

3.1 Tests électriques

Une fois la carte montée, des tests ont étés effectués afin de vérifier l'exactitude des niveaux de tension ainsi que la consommation de courant. Ceux-ci ont étés faits deux fois. Une première fois avec uniquement le circuit d'alimentation afin de prévenir tout dommage aux composants en aval en cas d'erreur. Puis la seconde avec circuit d'alimentation a été connecté au reste de la carte.

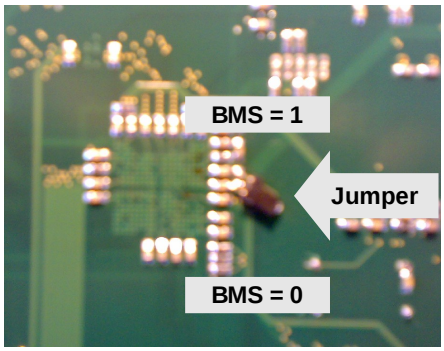
A la fréquence de 240MHz pour le coeur ARM et de 120MHz pour les contrôleur de périphériques avec la RAM et la Flash actives, la carte consomme en moyenne 320mA.

Les détails des tests sont disponibles en annexe E, page 75.

3.2 Errata

3.2.1 BMS

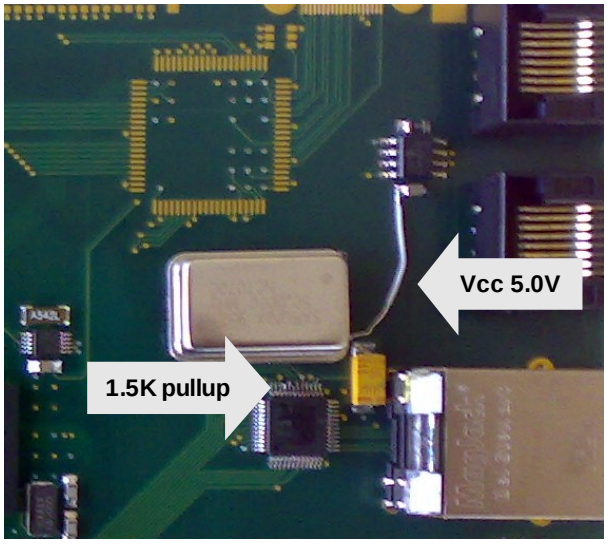
L'AT91SAM9263 dispose de deux options de boot en fonction de la valeur de sa pin PB3, BMS (Boot Mode Select), lors du reset. Cependant cette pin n'était connectée qu'au contrôleur audio selon sa seconde affectation un fois le boot passé. Il a donc été décidé de monter un jumper où une des pattes est à la masse, la seconde à 3.3V et la dernière connectée au BMS à travers une résitance. A noter qu'une valeur trop élevée de la résistance ne permet pas de tirer suffisament bas le niveau logique du BMS. Après plusieurs essais 10kΩ se trouve être une bonne valeur.



Jumper pour le BMS

3.2.2 Ethernet

L'oscillateur 50MHz doit être alimenté avec une tension de 5.0V et non 3.3V pour pouvoir fournir un niveau de tension suffisant en sortie. L'alimentation étant située dans un plan à l'intérieur de la carte, la solution a été de le dessouder, de plier la pin d'alimentation, de le ressouder et de connecter la pin 'volante' à l'alimentation 5.0V du transceiver CAN situé à proximité. De plus un résistance 1.5kΩ doit être connectée en pull-up sur la pin #30 du transceiver Ethernet DP83848. Celle-ci correspond à la pin data du bus de configuration du composant (MDIO).



Ethernet, corrections

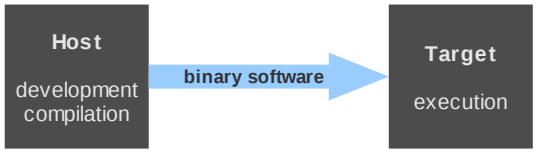
3.2.3 RS232

Les signaux TX et RX des port séries ont étés inversés. Il est donc nécessaire d'utiliser un null modem pour pouvoir communicer correctement.

La solution la plus simple à ce problème est de conserver un routage identique et de monter le connecteur femmelle à la place du mâle et inversément.

4 Développement croisé

Le **cross-development** (développement croisé) consiste à écrire et compiler une application sur une certaine machine, un **host** (hôte), pour une **target** (cible) aux spécifications différentes. Ces différences peuvent être tant matériel (processeur différent) que logiciel (système d'exploitation différent).



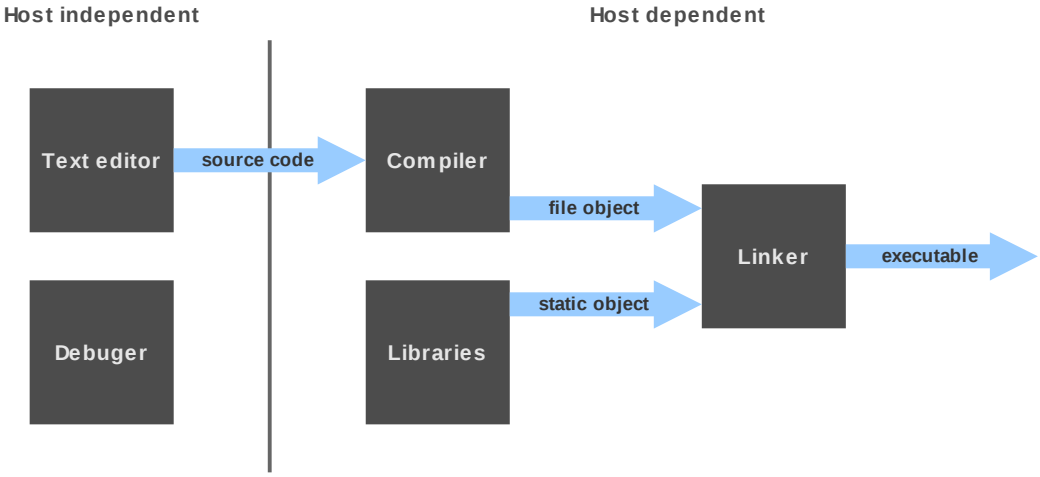
Schématisation du cross-development

Lors du développement de système embarqué le cross-development est très employé pour les raisons suivantes:

- Vitesse: Les cibles sont conçues afin d'optimiser leurs coûts et leur consomation électrique, non leur puissance de calcul.
- Mémoire: La construction d'exécutable est une opération gourmande en mémoire.
- Disponibilité: Certains outils peuvent ne pas être disponibles pour toutes les architectures.
- Simplicité: L'environement de développement est bien plus riche et convivial sur l'hôte.

4.1 Toolchain

En informatique on parle souvent de toolchain pour désigner l'ensemble des outils nécessaires à la production d'un executable. Ceux-ci se composent généralement d'un éditeur de texte, d'un compilateur, d'un linker, d'un ensemble de librairies et d'un débugeur.



Exemple de toolchain

4.2 ELDK

La toolchain utilisée est **ELDK** (Embedded Linux Development Kit). C'est la toolchain par défaut du project U-Boot, décrit au chapitre 7, page 47. Elle contient de nombreux utilitaires GNU tel que gcc, gdb, binutils, ...

Notez qu'elle est disponible avec la Glibc ou uClibc comme librairie C principale. Pour des questions de compatibilité des binaires au seins de la section Infotronic, nous utilisons la version avec la Glibc.

4.2.1 Mise en place de l'environement

La première chose à faire est de récupérer l'archive qui se présente en format iso ainsi que la somme de contrôle md5. Puis à l'aide de cette dernière on vérifie l'intégrité de la toolchain. Normalement un message devrait s'afficher et nous informer que les sources ont étées correctement téléchargées.

```
$ cd ~
$ wget ftp://ftp.denx.de/pub/eldk/4.1/arm-linux-x86/iso/arm-2007-01-21.iso
$ wget ftp://ftp.denx.de/pub/eldk/4.1/arm-linux-x86/iso/MD5SUM
$ md5sum -c MD5SUM
```

Un nouveau répertoire avec accès en mode user est crée dans `/opt`. L'image iso est montée dans `media/cdrom0` à l'aide de l'option `-o loop` puisque nous ne disposons pas du cd-rom réel. Puis la toolchain est installée avec la commande `cdrom/install` en spécifiant le répertoire précédemment créé à l'aide de `-d`. Pour finir par l'image iso est démontée du système de fichier.

```
$ sudo mkdir /opt/eldk
$ sudo chown USR_NAME /opt/eldk
$ sudo mount -o loop ~/arm-2007-01-21.iso /media/cdrom0
$ /media/cdrom0/install -d /opt/eldk
$ sudo umount /media/cdrom0
```

Par la suite bashrc (script de configuration du shell) est modifié afin de définir une nouvelle variable d'environement: CROSS_COMPILE. Celle-ci contient l'en-tête du double des commandes spécifiques à la toolchain arm-linux. La variable d'environement PATH est également modifiée afin que ces commandes soient accessibles sans avoir à rentrer leur chemin complet. Puis le script de configuration est exécuté à l'aide de `source`, ou bash est redémarré pour que ces changements soient pris en compte.

```
$ echo # ELDK >> ~/.bashrc
$ echo export CROSS_COMPILE=arm-linux-  >> ~/.bashrc
$ echo PATH=$PATH:/opt/eldk/usr/bin:/opt/eldk/bin  >> ~/.bashrc
$ source ~/.bashrc
```

4.2.2 Test

Une fois la toolchain installée, il est possible de tester si elle est capable de compiler une simple application, un 'hello world' par exemple.

```
$ arm-linux-gcc -o hello hello.c
```

Le résultat est un fichier exécutable. Grâce à la commande `file` le type du ficher peut être déterminé.

```
$ file hello
```

Le résultat devrait fortement ressembler à ceci:

```
 hello: ELF 32-bit LSB executable, ARM, version 1, dynamically linked (uses shared libs), for
GNU/Linux 2.4.3, not stripped
```

Cependant ce test ne garanti en rien que cette application s'exécute réellement correctement mais uniquement que la toolchain a été correctement installée et configurée.

4.3 Références

- Manuel de ELDK:
  ftp://ftp.denx.de/pub/eldk/4.2/ppc-linux-x86/distribution/README.html

4.4 Sources

- Toolchain ELDK:
  cdrom/eldk/src/arm-2007-01-21.iso
  ftp://ftp.denx.de/pub/eldk/4.1/arm-linux-x86/iso/arm-2007-01-21.iso
  ftp://ftp.denx.de/pub/eldk/4.1/arm-linux-x86/iso/MD5SUM

5 JTAG

Le JTAG permet d'accéder directement à de nombreuses ressources du microcontrôleur. C'est donc l'outil par excellence pour les opérations de test bas niveau et de lecture/écriture en mémoire. Dans notre cas il est utilisé afin de copier une image du bootstrap et du bootloader dans la mémoire Flash.

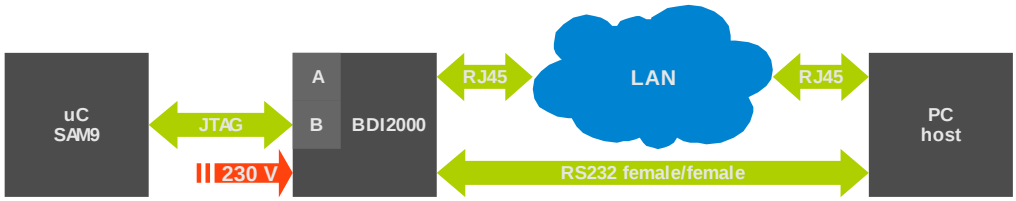
Il n'est pas possible de connecter directement le JTAG au PC hôte. Pour ce faire nous utilisons une interface de débugage, le BDI2000 d'Abatron AG, disponible à l'école.

5.1 Mise en place

Avant de pouvoir uitiliser **BDI2000** il faut en premier lieu copier quelques outils à partir de la disquette fournie avec. On commence donc par créer un nouveau répertoire où les copier. Puis on insère la disquette et on la monte dans le système de fichier. On copie l'intégralité de la disquette dans ce répertoire et on finit par démonter la disquette du système de fichier.

```
$ mkdir ~/bdi2000
$ sudo mount /dev/fd0 /media/floppy0
$ cp -r /media/flopy0 ~/bdi2000/
$ sudo umount /dev/fd0
```

Ensuite on connecte le BDI2000 à l'hôte à l'aide d'un câble série croisé femèle/femèle ainsi qu'au LAN avec un câble RJ45. Quand à la cible elle est connectée par un câble JTAG au BDI2000 à son connecteur TARGET B. Pour finir le BDI2000 est alimenté par une prise secteur. Le schéma suivant montre ces différentes connections.



Connections du BDI2000 avec l'hôte et la cible

Notez que le PC n'est pas connecté directement au BDI2000 par le câble RJ45 mais au travers d'un switch. Ceci afin que l'hôte puisse accéder à internet et au BDI2000 simultanément.

Une entrée est rajoutée à la table ARP afin d'accéder au BDI2000 à l'aide de son adresse IP. L'adresse MAC du BDI2000 est constituée de l'adress 00:0C:01 suivie des six premiers numéros de série. Ce qui nous donne `00:0C:01:96:75:73` pour le BDI2000 de l'école. Etant donné que l'hôte et le BDI2000 se trouvent sur le même sous-réseau, cette opération suffit. On peut tester à l'aide d'un simple `ping` si l'entrée est correcte.

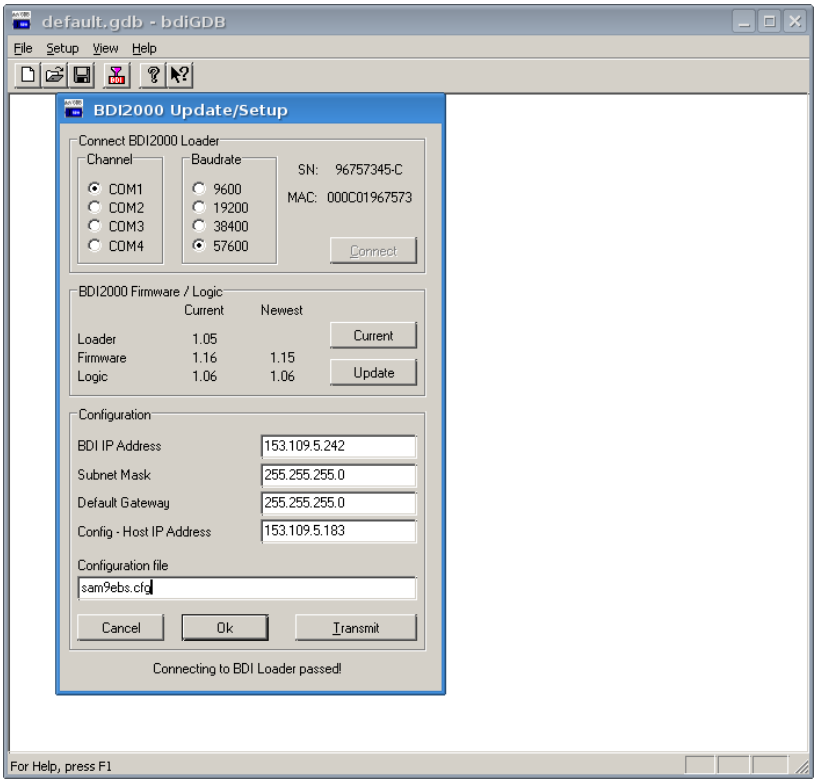
```
$ sudo arp -s 153.109.5.242 00:0c:01:96:75:73
$ ping 153.109.5.242
```

Normalement le BDI2000 est configurable sous Linux en ligne de commande à l'aide de l'utilitaire

`bdisetup`, présent dans le répertoire éponyme. Cependant après plusieurs tentatives infructueuses il a été décidé de le configurer à l'aide de l'outil Windows sous Linux grâce à Wine.

On commence par installer `wine`, puis on lance l'outil de configuration avec ce dernier.

```
$ sudo aptitude install wine
$ wine ~/bdi2000/B20ARMGD.EXE
```



bdiGDB exécuté sous Linux à l'aide de Wine

La connection série est configurée sur le port correspondant (COM1) ainsi que le baudrate à 57600. Puis il faut se connecter au BDI2000 avec le bouton 'Connect'. Le numéro de série et l'adresse MAC doivent normallement s'afficher.

Ensuite if faut compléter les adresses IP du BDI2000 et de l'hôte. L'adresse IP du BDI2000 a été assignée statiquement par l'école et est 153.109.5.242. Quand à celle de l'hôte, il est possible de connaître grâce à la commande `ifconfig`.

```
$ sudo ifconfig
```

Pour finir il faut indiquer le fichier de configuration à lire lors du démarage du BDI2000 et on lui transmet ces informations grâce au bouton 'Transmit'.

5.2 Fichier de configuration

Lors de sa mise sous-tension le BDI2000 va récupérer via **TFTP** le fichier de configuration et le lire. Celui-ci doit donc être fournit par un serveur TFTP. Sur la majorité des systèmes Linux le serveur `atftpd` est automatiquement démaré par le daemon `inetd`. Il reste plus qu'à copier ce fichier dans le répertoire associé

à `atftpd`, généralement `/var/lib/tftpboot`. La description des registres propre au microcontrôleur est également nécessaire au bon fonctionnement du BDI2000. Elle est donc aussi copiée dans le répertoire du seveur TFTP.

```
$ sudo cp ~/bdi2000/sam9ebs.cfg /var/lib/tftpboot/
$ sudo cp ~/bdi2000/reg926e.cfg /var/lib/tftpboot/
```

Le fichier de configuration est composé de cinq sections:

- [INIT] Définit une liste de commandes exécutées à chaque reset de la cible.
- [TARGET] Définit les paramètres spécifiques à la cible: processeur, organisation de la mémoire, fréquence, ...
- [HOST] Définit les paramètres spécifiques à l'hôte: adresse IP, fichier à charger, format de ce dernier, ...
- [FLASH] Définit les caractéristiques relatives à la Flash, type de Flash, fichier à charger, format de ce dernier, ...
- [REGS] Indique le fichier de définition des registres à utiliser en fonction du processeur.

Le fichier de configuration utilisé est disponible en annexe G.3, page 79.

Dans la section INIT certains registres sont configurés grâce à la commande WM32 (write memory 32 bits), dont la syntaxe est: `WM32 address value`. Le watchdog est désactivé afin d'éviter des resets involontaires et les wait-states du contrôleur sont configurés Flash au maximum. De plus une Led est activée afin de signaler que le microcontrôleur a déjà été partiellement configurer par le JTAG.

Dans la section TARGET le type de processeur est indiqué aisni qu'il s'agit d'un système petit boutiste. Le fichier signale aussi au BDI2000 de toujours utiliser le clock le plus rapide, mode adaptative.

Dans la section HOST l'adresse IP de l'hôte est définie, obligatoire. Le nom et le format du fichier sont également définis, facultatif. Ces derniers permettent seulement d'utiliser la version courte de la commande de chargement. Pour finir la configuration indique que ce fichier sera chargé manuellement par l'utilisateur.

Dans la section FLASH le type de Flash utilisée est défini, STRATAX16 pour les Flash de type Intel 28Fxxx ainsi que la la largeur du bus de donnée: 16. Le fichier et son format sont aussi définis de manière identique à la section HOST.

Pour finir dans la section REGS il est indiqué qu'il faut utiliser le jeu de registres spécifiques AT91SAM926.

Ce fichier s'avère adéquat afin d'écrire en mémoire. Mais il ne peut pas être utilisé tel quel pour du débugage à cause des opérations de la section INIT. Si ces opérations sont nécessaires pour écrire en mémoire, elles peuvent également masquer ceraines faiblesses de l'application chargée d'effectuer le bootstrap. L'initialisation du contrôleur Flash sera effectué par le fichier de configuration et pas forcément par le bootstrap.

La méthode la plus simple afin d'y remédier est de commenter l'ensemble de la section INIT lors du débugage, de sauvegarder le fichier et d'écraser celui présent dans `/var/lib/tftpboot`. Et d'effectuer l'opération inverse lorsque l'on souhaite écrire en mémoire. Pour mémoire lorsque la section INIT n'est pas commentée, une Led rouge s'allume.

5.3 Tests

Afin de pouvoir tester le bon fonctionnement du JTAG et du BDI2000, une application simpliste a été écrite et complilée. Cette dernière se borne à faire clignoter les cinq Leds rouges de la carte. et est disponible en annexe G.1, page 78. Elle est chargeable soit en RAM interne, soit en Flash. Ceci est possible grâce à la configuration minimale effectuée dans la section INIT.

Cette application est complilée à l'aide de la toolchain ELDK (Embeded Linux Development Kit), utilisée par U-Boot et présentée au chapitre 4.2. Le Makefile correspondant est quand à lui disponible en annexe G.2, page 78.

```
$ cd ~/jtag/testapps/ledtest
$ make
$ sudo cp ledtest.bin /var/lib/tftpboot/
```

Une fois l'application compilée et copiée dans le répertoire d'`atftpd`, il faut se connecter au BDI2000 via une connection `telnet`.

```
$ telnet 153.109.5.242
```

Une fois l'initialisation finie, il est possible de charger l'application `ledtest.bin` dans la RAM interne du micro-controlleur à l'adresse 0x00300000 ou 0x00500000 à l'aide de `load address [file] [format]`. Puis l'application peut être lancée avec la commande `go.` Normalllemnt les cinq Leds devraient clignoter.

```
Core#0> load 0x00300000
Core#0> go 0x00300000
```

Il est également possible de tester si l'application a été correctement chargée en affichant le contenu de la mémoire à l'adresse 0x00300000 (ou 0x00500000) à l'aide de la commande `md address` (memory display). Celui-ci devrait être similaire au contenu du fichier .bin, visible grâce à `KHexEdit` sous Debian/KDE.

```
Core#0> md 0x00300000
```

Les mêmes opérations sont possibles avec la mémoire Flash à l'adresse 0x10000000, où est connectée notre mémoire Flash (EBI0_CS0). La seule exception est l'utilisation de la commande `prog`, similaire à `load` mais pour la Flash.

```
Core#0> prog 0x10000000
Core#0> go 0x10000000
```

Notez qu'en cas d'erreur d'écriture dans la Flash, la réinitialisation d'un ou plusieurs secteurs peut être nécessaire à l'aide de la commande `erase address`. La taille des secteurs sur la Flash utiliée est de 0x20000.

```
Core#0> erase 0x10000000
```

5.4 Débuger à l'aide du BDI2000

Le BDI2000 permet de concert avec `gdb` de débuger un programme.

Pour ce faire il faut démarrer `gdb` puis se connecter au BDI2000 avec `target remote IP_ADDR:2001` comme commande. Cette opération à pour conséquence d'arrêter l'exécution du programme en cours. Pour le redémarrer il faut appuyer sur `c` pour continue.

```
$ gdb
(gdb) target remote 153.109.5.242:2001
(gdb) c
```

Cependant le programme charger est un `.bin` et ces derniers ne disposent d'aucun symbole de débugage. Il faut donc charger les symboles de débugage du `.elf` avec la commande `file`. Par la suite lorsque le programme s'arrêtera inopinément il sera possible de savoir dans quelle fonction avec `info symbol program_counter`.

```
$ cd ~/u-boot
$ gdb
(gdb) file ./u-boot
(gdb) info symbol 0x23f0050c
```

Une autre commande très utile et suffisamment peu explicite pour être mentionnée est la suivante:

```
find path_to_examine -exec grep -q "string_to_search" {} \; -print
```

Celle-ci effectue une recherche dans tous le répertoire `path_to_examine` et affiche tous les fichiers où est `string_to_search` présent.

5.5 Références

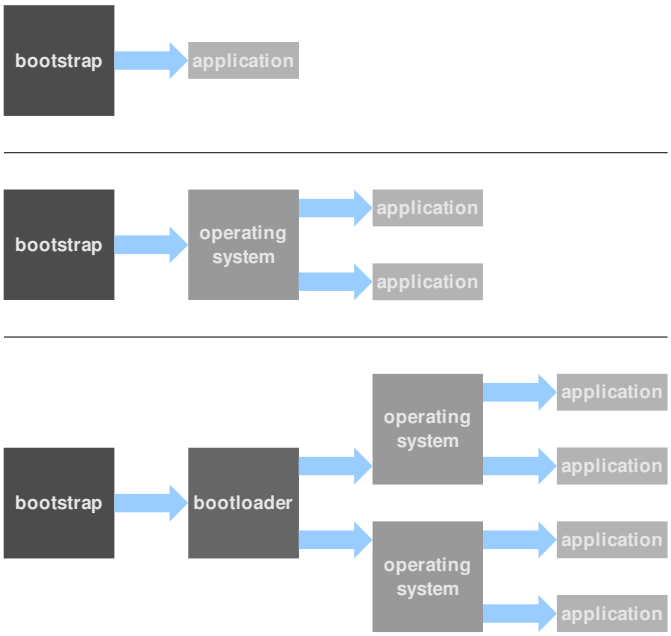
- Manuel BDI2000:
  cdrom/bdi2000/doc/manual.pdf
  http://www.abatron.ch/fileadmin/user_upload/products/pdf/ManCwPWS-2000C.pdf

5.6 Sources

- Outils pour le BDI2000:
  cdrom/bdi2000/src/gdbarm21.zip
- BDI2000 – fichier de configuration:
  cdrom/bdi2000/src/sam9ebs.cfg
- Application ledtest:
  cdrom/bdi2000/testapp/ledtest/

6 Bootstrap

Le bootstrap est l'ensemble des actions bas-niveau à effectuer lors de la mise en route d'un système avant le lancement d'un logiciel plus important. Ce dernier peut être directement l'application finale, un système d'exploitation ou un bootloader. La figure suivante illustre ces différentes possibilités.



Différentes stratégies de boot

Dans notre cas la dernière solution a été retenue car elle offre la plus grande souplesse.

6.1 Le bootstrap en bref

Lors du bootstrap, seul les opérations indispensables au chargement du bootloader sont effectuées. Elles peuvent se résumer de la manière suivante:

- Définition des vecteurs d'interruptions
- Initialisation du pointeur de pile
- Initialisation du segment bss
- Désactivation du watchdog
- Configuration des contrôleurs mémoire
- Configuration des PLLs
- Configuration des différents clock
- Chargement et exécution du bootloader en mémoire

6.2 Stratégies de boot de l'AT91SAM9263

L'AT91SAM9263 démarre toujours à l'adresse mémoire 0x0000 0000. Afin d'avoir malgré tout une certaine souplesse, cette adresse peut correspondre à différents emplacements mémoire en fonction des valeurs du registre REMAP et du BMS (Boot Mode Select).

La valeur du BMS lors du boot permet de choisir entre le périphérique connecté sur l'EBI0 sélectionnable avec le CS0 (BMS = 0) et le firmware interne stocké en ROM (BMS = 1). La valeur du BMS correspond à la valeur de la pin #K13 (AC97RX/BMS) de l'AT91SAM9263.

Une fois le REMAP effectué de manière logicielle l'adresse 0x0000 0000 pointe sur la SRAM interne. Le tableau suivant résume les correspondances en fonction du REMAP et du BMS.

Mappage interne de l'adresse 0x0000 0000

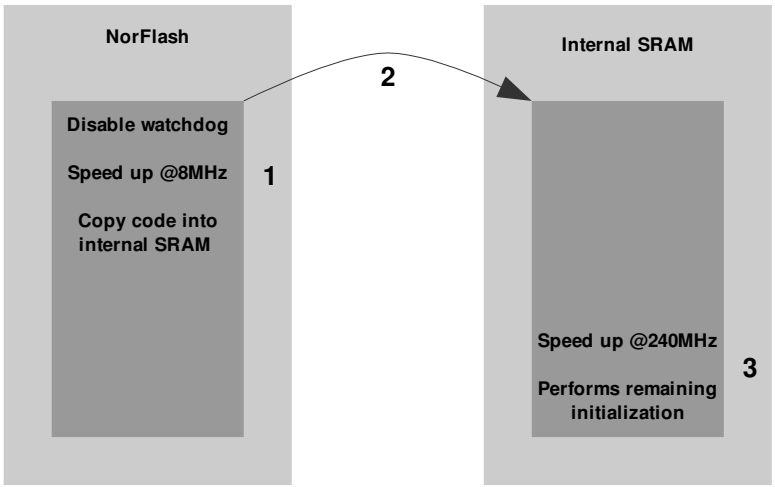
Address	*REMAP = 0*		*REMAP = 1*
	BMS = 0	*BMS = 1*	
0x0000 0000	EBI0_NCS0 (NOR Flash)	ROM (firmware)	SRAM C

6.2.1 Boot depuis la Flash NOR

La méthode la plus simple consiste à copier le code du bootstrap au début de la Flash et à positionner le BMS à zéro. Ainsi lors du prochain reset de la carte, c'est ce code qui sera exécuté.

Cependant un bug matériel de l'AT91SAM9263 (révision A et B) nous empêche de modifier les paramètres du contrôleur de mémoire Flash lorsque du code est exécuté à partir de cette mémoire. C'est justement ce qui est fait lors de la configuration des wait states du SMC (Static Memory Controller, contrôleur de la mémoire Flash) avant de changer d'horloge principale.

Néanmoins une solution existe. Elle consiste à booté en Flash et à effectuer le strict minimum (**1**). Puis à se recopier SRAM interne (**2**). A partir de ce moment il est possible d'augmenter la fréquence et de mener à bien l'initialisation de la carte (**3**).



Solution au contrôleur Flash bugué lors du bootstrap

Bien que la taille du bootstrap soit relativement modeste (5.9 kB), le copier à la fréquence de 32'768 Hz (slow clock) prend un temps loin d'être négligeable. Afin d'accélérer cette opération, il est tout de même possible d'augmenter la fréquence dans la limite des wait states par défaut du contrôleur de Flash avant de

commencer la réallocation du bootstrap en RAM.

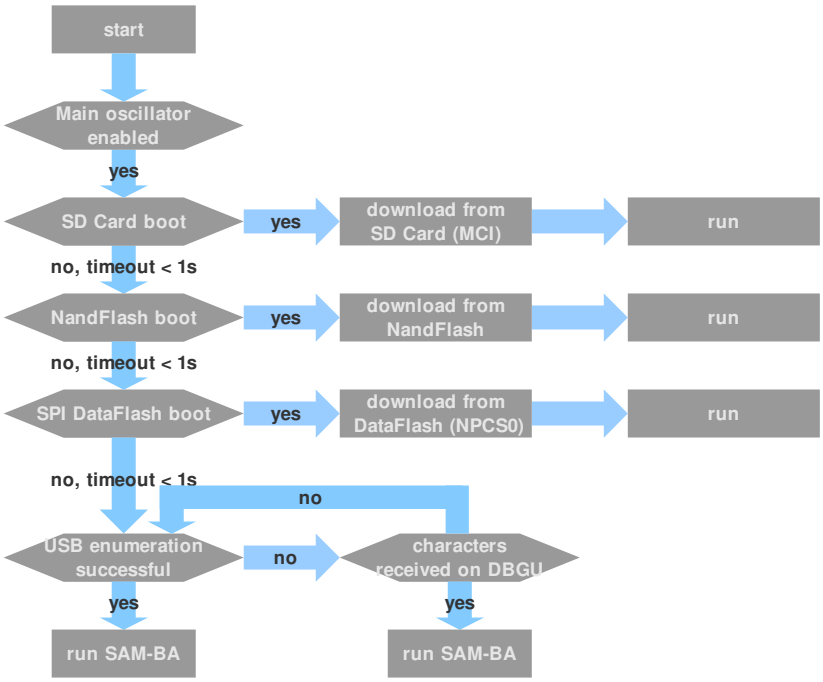
Etant donné que notre Flash NOR nécessite des temps d'attentes maximums de 120ns et que les wait states par défaut correspondants sont de 1 cycle, la fréquence maximum est donc de 8MHz.

$$\frac{1}{8*10^{6}\,Hz} = 125*10^{-9}\,s$$

Celle-ci est facilement atteignable avec notre quartz principal à 16MHz, en configurant le prescaler à 2 dans le registre PMC_MCKR (0xFFFFFC30).

6.2.2 Boot depuis le firmware

La seconde possibilité est de booté à l'aide du firmware interne avec le BMS à l'état haut. Celui-ci commence par effectuer quelques opérations d'initialisation dont l'initialisation des clocks. Seules certaines fréquences sont supportées pour l'auto-détection dont 16MHz. La liste complète des fréquences possibles se trouve au point 13.3 du datasheet du AT91SAM926. Une fois l'initialisation effectuée, la séquence de boot se poursuit en tentant de trouver du code bootable sur différents supports: SDCard, NandFlash, SPI DataFlash. Si aucune application bootable n'est trouvée, celui-ci essaye de trouver un périphérique USB. Si oui, le programme de boot SAM-BA (SAM boot assistant) est démarré. Dans le cas contraire le firmeware écoute si un caractère a été reçu sur le port série de débugage (DBGU). Dans l'affirmative il démarre SAM-BA, sinon le programme réessaye de trouver un périphérique USB et la séquence boucle indéfiniment. Le schéma suivant résume l'algorithme de boot.



Algorithme du firmeware de boot

Dans notre cas les possibilités sont plus limitées. En effet nous ne disposons pas de NandFlash, ni de DataFlash et les pins de débug ne sont malheureusement pas connectées. De plus un bug dans la révision A du microcontrôleur rend toute tentative de boot à partir de la NandFlash ansi que d'une SDCard impossible.

Ensuite le logiciel SAM-BA n'est disponible que pour Windows. Bien qu'une tentative de port sous Linux existe, celle-ci n'est pas facilement utilisable. C'est pourquoi nous avons décidé de privilégier le démarrage à partir de la Flash NOR.

6.3 Préparation des sources

Plutôt que de coder à partir de zéro un bootstrap, il est plus simple de se baser sur un code existant et l'adapter à nos besoins. Le bootstrap disponible sur le site Linux4Sam a servi de base.

L'archive est récupérée et vérifiée grâce à sa somme de contrôle md5.

```
$ cd ~/bootstrap/src
$ wget ftp://www.linux4sam.org/pub/at91bootstrap/AT91Bootstrap1.10.zip
$ wget ftp://www.linux4sam.org/pub/at91bootstrap/AT91Bootstrap1.10.zip.md5
$ md5sum -c AT91Bootstrap1.10.zip.md5
$ unzip AT91Bootstrap1.10.zip -d bootstrap
$ mv AT91Bootstrap1.10 ../
```

Une fois l'archive rapatriée et décompressée, le projet le plus proche de notre carte, le AT91SAM9263EK, est copié. Puis le répertoire, le projet (sous-répertoire), les fichiers sources sont renommés.

```
$ cd ~/bootstrap/AT91Bootstrap1.10/board
$ cp -r at91sam9263ek/ sam9ebs/
$ cd sam9ebs
$ mv at91sam9263ek.c sam9ebs.c
$ mv -r nandflash/ nor/
$ mv norflash/at91sam9263ek.h nor/sam9ebs.h
```

Ensuite le Makefile est modifié afin d'utiliser la bonne toolchain (ELDK) et de compiler l'application avec le sous-projet de notre carte.

```
# bootstrap/board/sam9ebs/nor/Makefile

#...

CROSS_COMPILE = /opt/eldk/usr/bin/arm-linux-

#...

BOARD=sam9ebs
PROJECT=nor

#...
```

Il faut également modifier le fichier `include/part.h` afin d'utiliser le nouveau fichier `sam9ebs.h` au lieu de `at91sam9263.h`.

```
/* bootstrap/include/part.h                                          */

#ifdef AT91SAM9263
#include "AT91SAM9263_inc.h"
/* #include "at91sam9263ek.h"                                        */
#include "sam9ebs.h"
#endif
```

Et pour finir modifier une macro dans le fichier `include/norflash.h` qui est défninie correctement que pour les systèmes avec un seul bus mémoire.

```
/* bootstrap/include/norflash.h                                          */

/* #define AT91_NORFLASH_BASE        AT91C_EBI_CS0                       */
 #define AT91_NORFLASH_BASE          AT91C_EBI0_CS0
```

A partir de maintenant l'application bien que non fonctionnelle devrait compiler sans erreur.

Récapitulatif des fichiers source modifiés

File	*Satus*
crt0_gnu.S	modified
include/part.h	modified
include/norflash.h	modified
board/sam9ebs/sam9ebs.c	new, based on board/at91sam9263ek/at91sam9263ek.c
board/sam9ebs/nor/Makefile	new, based on board/at91sam9263ek/nandflash/Makefile
board/sam9ebs/nor/sam9ebs.h	new, based on board/at91sam9263ek/nandflash/at91sam9263ek.h

Les fichiers `crt0_gnu.S`, `sam9ebs.h`, `main.c` et `sam9ebs.c` où d'importantes modifications ont été effectuées sont disponibles en annexe H, page 80.

6.4 Le bootstrap en détail

Le bootstrap est une des seules applications C à ne pas avoir la fonction main() comme point d'entrée mais un bout de code assembleur. La cause est double. Premièrement le bootstrap doit définir les vecteurs d'interruption. Deuxièmement un code en C s'attend à disposer de certains services: pointeur de pile initialisé, segment bss (emplacement des variables non initialisées) mis à zéro. C'est pourquoi l'application de boot commence par exécuter le code contenu dans `crt0_gnu.S`.

Les sous-chapitres suivants décrivent de manière générique chaque étape importante de notre bootstrap. La définition des macros spécifiques à la carte est décrit au chapitre suivant.

6.4.1 Vecteurs d'interruption

La première chose définie est la table des vecteurs d'interruption qui n'est rien d'autre qu'une liste de sauts inconditionnels vers un gestionnaire d'interruption (interrupt handler). Durant le bootstrap seul l'interruption reset est gérée, les autres sautent à un gestionnaire qui boucle indéfiniment sur lui-même.

```
/* bootstrap/crt0_gnu.S                                                  */

reset:

/* verctors table                                                        */
_exception_vectors:
      b       reset_vector   /* reset                                   */
      b       undef_vector   /* undefined instruction                   */
      /* ... */

undef_vector:
      b       undef_vector   /* loop endless                            */
/* ... */
```

6.4.2 Initialisation de la pile

Puis la pile est initialisée ce qui permet de gérer l'appel est le retour des fonctions pour lesquelles le link register n'est pas suffisant.

```
/* bootstrap/crt0_gnu.S                                                 */

/* init the stack                                                       */
_init_stack:
      ldr     sp, =TOP_OF_MEM
```

6.4.3 Réallocation en RAM interne

Ensuite pour contourner le bug du contrôleur Flash de l'AT91SAM9263 le code se recopie en RAM interne. Cette opération pouvant être longue, le watchdog est désactivé. Si la macro CFG_QUICK_START est définie la fréquence de l'horloge est augmentée avec la sous-routine `speed_up_osc`, décrite un peu plus bas. Ensuite les adresses du début du programme (_stext) et de la fin (_edata) sont chargées dans des registres. Ces adresses définissent les bornes du programme. Après s'être recopié entièrement en RAM interne un remap est effectué. A partir de ce moment l'application s'exécute depuis la RAM interne.

```
/* bootstrap/crt0_gnu.S                                                 */

_relocate_to_sram:

      /* relocation can be slow, disable the watchdog                   */
      ldr     r1, =AT91C_WDTC_WDMR
      mov     r2, #0x00008000
      str     r2, [r1]

#ifdef CFG_QUICK_START
      /* speed up clock at max possible frequancy before relocate       */
      bl      speed_up_osc

#endif/* CFG_QUICK_START */

      /* copy all boot app into internal sram                           */
      ldr     r1, =_stext                 /* start of boot code         */
      ldr     r2, =_edata                 /* end of boot code           */
      mov     r4, #0x00300000             /* internal sram address      */

relocate_to_sram_loop_:
      cmp     r1, r2
      ldrne   r3, [r1], #4
      strne   r3, [r4], #4
      bne     relocate_to_sram_loop_

      /* remap memmories                                                */
      ldr     r1, =AT91C_MATRIX_MRCR
      ldr     r2, =0x000001FF
      str     r2, [r1]
```

La sous-routine `speed_up_osc`, configure le main oscillator register du microcontrôleur puis boucle tant que le signal d'horloge ne s'est pas stabililsé. Une fois fait elle configure le main clock register afin que le microcontrôleur utilise la nouvelle horloge comme base de temps.

```
/* bootstrap/crt0_gnu.S                                                 */


/* increase clock before code relocation                                */
speed_up_osc:
      /* setup main osc                                                 */
      ldr     r1, =AT91C_PMC_MOR
      ldr     r2, =CFG_QUICK_START_OSC
      str     r2, [r1]
```

```
      /* wait until osc is stabilized: 1 == PMC_SR.0                    */
speed_up_osc_stabilize_loop_:
      ldr     r1, =AT91C_PMC_SR
      ldr     r2, [r1]
      and     r2, r2, #0x00000001
      cmp     r2, #0x00000001
      bne     speed_up_osc_stabilize_loop_

      /* switch clock from slow clock to main                           */
      ldr     r1, =AT91C_PMC_MCKR
      mov     r2, #0x00000005
      str     r2, [r1]

      mov     pc, lr                /* return */
```

6.4.4 Finalisation de l'initialisation low level

A partir de ce moment les données du segment data sont recopiées en RAM. Le segment bss utilisé pour les variables C non initialisée est mis à zéro. Et pour finir le code assembleur se branche sur la fonction C `main()`.

```
/* bootstrap/crt0_gnu.S                                                 */

/* copy the data section in RAM at .data link address                   */
_init_data:
      ldr     r2, =_lp_data
      ldmia   r2, {r1, r3, r4}
1:
      cmp     r3, r4
      ldrcc   r2, [r1], #4
      strcc   r2, [r3], #4
      bcc     1b

/* initialize the bss segment                                           */
_init_bss:
      adr     r2, _lp_bss
      ldmia   r2, {r3, r4}
      mov     r2, #0
1:
      cmp     r3, r4
      strcc   r2, [r3], #4
      bcc     1b

/* branch on C code Main function (with interworking)                   */
_branch_main:

      ldr     r4, = main
      mov     lr, pc
      bx      r4
```

6.4.5 Initialisation des horloges

La première fonction exécutée par main est hw_`init()`. Celle-ci commence par configurer la PLLA. Puis le main clock est configuré pour utiliser le signale de la PLLA comme base pour l'horloge principale. Pour finir la PLLB est aussi initialisée. Les valeurs des macros PLLA_SETTINGS et PLLB_SETTINGS sont décrites à la section 6.5.1, page 44.

```
/* bootstrap/board/sam9ebs/sam9ebs.c                                    */

      /* configure PLLA = MOSC * (PLL_MULA + 1) / PLL_DIVA              */
```

```
	pmc_cfg_plla( PLLA_SETTINGS, PLL_LOCK_TIMEOUT ) ;

	/* switch MCK on PLLA output PCK = PLLA = 2 * MCK                */
	pmc_cfg_mck( MCKR_SETTINGS, PLL_LOCK_TIMEOUT ) ;

	/* configure PLLB                                                */
	pmc_cfg_pllb( PLLB_SETTINGS, PLL_LOCK_TIMEOUT ) ;
```

6.4.6 Initialisation de la RAM

Le contrôleur de la RAM est initialisé. Tout d'abord la matrice interne du microcontrôleur est configurée pour pouvoir gérer le bus sur lequel la RAM est connectée: EBI0.

```
/* bootstrap/board/sam9ebs/sam9ebs.c                                  */

	writel
	(
		readl( AT91C_BASE_CCFG + CCFG_EBI0CSA )      |
		(1 << 16)                                    |
		AT91C_EBI_CS1A_SDRAMC,
		AT91C_BASE_CCFG + CCFG_EBI0CSA
	) ;
```

Puis la fonction `sdram_init()` configure le contrôleur avec les valeurs spécifiques à notre RAM Intel K45560832E-TC75.

Sa configuration physique:
- le nombre de bits de colonne: 10
- le nombre de bits de ligne: 13
- le nombre de banque: 4
- la taille du bus de donnée: 32

Le nombre de wait state des différents délais sont aussi initialisés. Par exemple le datasheet de la RAM impose un temps minimum de 65ns pour le row cycle delay. Sachant que la RAM opère à la moitié de la fréquence du coeur ARM, donc à 120MHz, il faut au minimum 8 wait states pour ce délai.

$$65\text{ns} \leq \frac{8}{120\text{MHz}}$$

```
/* bootstrap/board/sam9ebs/sam9ebs.c                                  */

	/* configure SDRAM controller                                    */
	sdram_init
	(
		AT91C_SDRAMC_NC_10           | /* column addr A0-9 */
		AT91C_SDRAMC_NR_13           | /* row addr A0-12   */
		AT91C_SDRAMC_CAS_3           | /* RAM has 2 & 3 cas*/
		AT91C_SDRAMC_NB_4_BANKS      | /* RAM has 4 banks  */
		AT91C_SDRAMC_DBW_32_BITS     | /* bus 32 bits width*/
		AT91C_SDRAMC_TWR_15          | /* FIXME            */
		AT91C_SDRAMC_TRC_8           | /* row cycle 65ns   */
		AT91C_SDRAMC_TRP_3           | /* row precharge 20ns*/
		AT91C_SDRAMC_TRCD_3          | /* row 2 column 20ns*/
		AT91C_SDRAMC_TRAS_6          | /* row active 45ns  */
		AT91C_SDRAMC_TXSR_15,          /* FIXME            */
		(MASTER_CLOCK * 11)/1000000,   /* refresh timer <64ms*/
		AT91C_SDRAMC_MD_SDRAM          /* no low power     */
	) ;
```

Notez que deux valeurs de timing n'ont pas pu être déterminées et ont étés définies à leur valeur maximale. A l'avenir il serait bien de les optimiser expérimentalement.

Les pins multiplexées utilisées par la RAM sont initialisées par la fonction `sdram_hw_init()`. Cette fonction définit une structure qui décrit chaque pin. Cette structure est passée en argument de la fonction `pio_setup()`.

```
/* bootstrap/board/sam9ebs/sam9ebs.c                                    */

void sdramc_hw_init( void )
{
	/* RAM PIOs pinning definition                                  */
	const struct pio_desc sdramc_pio[] =
	{
		{ "D16", AT91C_PIN_PD( 16 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
		/* ... */
		{ "D31", AT91C_PIN_PD( 31 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
		{ (char *) 0, 0, 0, PIO_DEFAULT, PIO_PERIPH_A },
	} ;

	/* configure SDRAMC PIO                                         */
	pio_setup( sdramc_pio ) ;
}
```

6.4.7 Initialisation de la Flash NOR

L'initialisation de la mémoire Flash se fait au travers de la fonction `norflash_hw_init()`. Celle-ci se contente d'écrire dans les registres du contrôleur des valeurs définies dans le fichier de configuration `norflash.h`. Celui-ci est décrit plus bas dans la section 6.5.2, page 44.

6.4.8 Chargement d'U-Boot

L'initialisation arrivée à son terme, le bootstrap charge U-Boot en mémoire et se branche dessus.

```
 /* bootstrap/main.c                                                   */

/*
	2nd step: load bootloader from media
*/

 /* load from Norflash in RAM                                          */
#ifdef CFG_NORFLASH
	load_norflash( IMG_ADDRESS, IMG_SIZE, JUMP_ADDR ) ;
#endif

/* ... */

/*
	3rd step: branch to the image address
*/
	return JUMP_ADDR ;
```

6.5 Fichier de configuration

6.5.1 Configuration du bootstrap

Le fichier de configuration commence par indiquer que le bootstrap doit configurer les aspects bas niveau de la carte ainsi que la RAM. Il définit également que de la Flash NOR est utilisée.

```
/* bootstrap/board/sam9ebs/nor/sam9ebs.h                                */

/* init standard hardware                                               */
#define CFG_SDRAM
#define CFG_HW_INIT

/* use norflash memory to load u-boot                                   */
#undef CFG_DATAFLASH
#undef CFG_NANDFLASH
#define CFG_NORFLASH
```

Puis il indique que l'on souhaite utiliser l'oscillateur à 16MHz avec une division à 2 comme horloge principale avant que le remap ne soit effectué.

```
/* bootstrap/board/sam9ebs/nor/sam9ebs.h                                */

/* enable quick start, configure oscillator before copy app into ram    */
#define CFG_QUICK_START
/* use main oscillator with a prescaller of 2: 16MHz / 2 => 8MHz        */
#define CFG_QUICK_START_OSC  0x00000801
```

Il définit également les valeurs des PLLs. La PLLA est configurée de manière à produire un signal à 240MHz soit la fréquence maximum du coeur ARM. Quand à la PLLB elle produit un signal à 48MHz qui est obligatoire pour pouvoir utiliser les périphériques USB.

```
/* bootstrap/board/sam9ebs/nor/sam9ebs.h                                */

/* DIVA = 1, MULA = 15 + 1, OUTA = 01 ; 16*(14+1(implicit))/1=240 MHz   */
#define PLLA_SETTINGS        0x200E7F01

/* DIVB = 1, MULB = 3 + 1, OUTB = 00, USBDIV = 00 ; 16*(3+1)/1=48 MHz   */
#define PLLB_SETTINGS        0x00033F01
```

Les paramètres pour le chargement d'U-Boot: adresse en Flash, taille et adresse de destination en RAM sont aussi définit dans ce fichier.

```
/* bootstrap/board/sam9ebs/nor/sam9ebs.h                                */

/* u-boot address in flash without flash offset                         */
#define IMG_ADDRESS          0x00010000

/* u-boot size in Flash                                                 */
#define       IMG_SIZE       0x00200000

/* u-boot destination address in ram                                    */
#define JUMP_ADDR            0x23F00000
```

6.5.2 Configuration de la Flash NOR

Les timings de la Flash Nor sont de 120ns pour l'établissement des données après que le signal chip select soit passé à l'état bas.

```
/* bootstrap/includ/norflash.h                                          */

/*
      15 * ( 1 / 120e6 ) >= 120 ns
*/
#define AT91C_FLASH_NWE_SETUP              (15 << 0)
#define AT91C_FLASH_NCS_WR_SETUP    (15 << 8)
#define AT91C_FLASH_NRD_SETUP              (15 << 16)
#define AT91C_FLASH_NCS_RD_SETUP    (15 << 24)
```

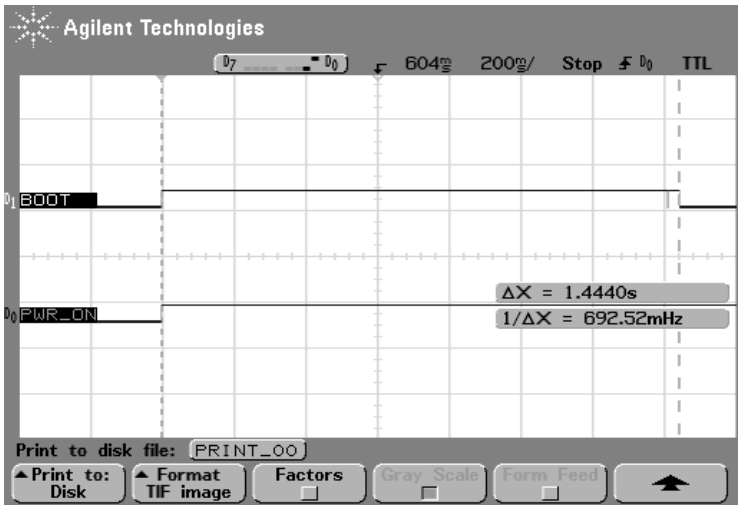
6.6 Tests

6.6.1 Chargement de ledtest.bin

Pour tester le bon fonctionnement du bootstrap l'application ledtest ,déjà utilisée pour tester le BDI2000, a été chargée en RAM interne dans un premier temps, puis en RAM principale. Le chargement s'est fait depuis la Flash. Le fonctionnement de l'application indique que le bootstrap a correctement configuré les contrôleurs Flash et RAM.

6.6.2 Durée du bootstrap

Afin d'avoir un bonne idée du temps mis par le microcontrôleur pour effectuer le bootstrap, lorsque la macro CFG_VERBOSE est définie, la pin #5 du GPIO est drivée à 1 au début du boot et à 0 à la fin. Ainsi il est facile d'effectuer une mesure de temps avec un oscilloscope. Le temps de bootstrap est d'environ 1.44 seconde.



Durée du bootstrap

6.7 Patch

Plutôt que de stocker l'ensemble des sources de l'application, il est tout à fait possible de ne stocker que les changements apportés au projet de base. Un patch a donc été créé pour l'application du bootstrap. Il se situe dans `bootstrap/src/Bootstrap-v1.10_sam9ebs.patch`.

Les commandes suivantes permettent de le créer.

```
$ cd ~/bootstrap
$ unzip src/AT91Bootstrap1.10.zip -d patch_base
$ diff -rupN patch_base/Bootstrap-v1.10 Bootstrap-v1.10/ > Bootstrap1.10_sam9ebs.patch
$ mv Bootstrap-v1.10_sam9ebs.patch src/
$ rm -r patch_base
```

Et celles-ci après de l'appliquer.

```
$ cd ~/bootstrap
$ unzip src/unzip src/AT91Bootstrap1.10.zip
$ cd Bootstrap1.10/
$ patch -p1 < ../src/ Bootstrap-v1.10_sam9ebs.patch
```

6.8 Références

- Linux4SAM:
  http://www.linux4sam.org/twiki/bin/view/Linux4SAM
- Application Note AT91Bootstrap:
  ~/bootstrap/doc/AT91Bootstrap.pdf
  http://www.atmel.com/dyn/resources/prod_documents/doc6277.pdf
- GNU device drivers, AT91SAM9263ek patches pages:
  ~/bootstrap/doc/nor_workaround.pdf
  http://www.gnudd.com/wd/at91sam9263ek-nor.html
- Pages de manuel de diff:
  $ man diff,
  (konqueror) man:diff
- Pages de manuel de patch:
  $ man patch,
  (konqueror) man:patch

6.9 Sources

- AT91Bootstrap:
  ftp://www.linux4sam.org/pub/at91bootstrap/AT91Bootstrap1.10.zip
  ftp://www.linux4sam.org/pub/at91bootstrap/AT91Bootstrap1.10.zip.md5
  cdrom/bootstrap/src/AT91Bootstrap1.10.zip
- Patch pour AT91Bootstrap:
  cdrom/bootstrap/src/Bootstrap-v1.10_sam9ebs.patch

7 U-Boot

Un bootloader (chargeur d'amorçage) est une application qui permet de lancer un ou plusieurs systèmes d'exploitation à partir de différents médias: disque, réseau, clef USB, ...

7.1 Choix d'U-Boot

Loin d'être le seul bootloader possible, U-Boot reste malgré tout le choix le plus complet et le plus largement utilisé dans les projets open source embarqués. Tous les autres bootloaders envisagés n'arrivent pas à rassembler autant de qualités qu'U-Boot. Le tableau suivant résume les autres possibilités ainsi que leur principale faiblesse.

Autres bootloaders

Bootloader	*Inconvénients*
ABLE	Non open source
Blob	Uniquement pour StrongARM
bootldr	Support uniquement pour Flash
redboot	Moins courant et documenté qu'U-Boot

7.2 Préparation des sources

Les sources de la dernière version stable (2008.10-rc1 en date du 21 novembre 2008) d'U-Boot sont téléchargées depuis le site denx.de. Puis elles sont détarées.

```
$ mkdir ~/u-boot
$ cd ~/u-boot
$ wget ftp://ftp.denx.de/pub/u-boot/u-boot-2008.10-rc1.tar.bz2
$ tar xvjf u-boot-2008.10-rc1.tar.bz2
```

7.3 Personnalisation d'U-Boot

7.3.1 Makefile

Pour commencer il faut indiquer dans le `Makefile` que pour toute architecture ARM, on utilise la toolchain ELDK présentée au chapitre 4.2, page 28.

```
ifeq ($(ARCH),arm)
CROSS_COMPILE = /opt/eltk/usr/bin/arm-linux-
endif
```

Puis toujours dans le `Makefile` il faut définir une nouvelle cible propre à la carte. Cette cible est basée sur les cibles des cartes d'évaluation d'Atmel.

```
sam9ebs_config       :       unconfig
      @$(MKCONFIG) $(@:_config=) arm926ejs sam9ebs atmel at91
```

Dernière modification du `Makefile`, le répertoire `exemples` pose des problèmes de compilation. Etant donné qu'il nous est d'aucune utilité, la solution la plus simple et de le commenter.

```
# SUBDIRS      = tools \
#                examples \
#                post \
#                post/cpu
SUBDIRS        = tools \
                 post \
                 post/cpu
```

7.3.2 Fichier de configuration

U-boot se base sur un fichier de configuration propre à chaque carte pour créer l'application. Plutôt que de partir d'un fichier vide on peut se baser sur un fichier existant, le fichier de la carte AT91SAM9263EK.

```
$ cd ~/u-boot/include/configs
$ cp at91sam9263ek.h sam9ebs.h
```

Ce fichier a été entièrement remis en forme et est disponible en annexe I, page 94. Les définitions importantes sont expliquées ci-après.

Tout d'abord, il faut indiquer à U-Boot de ne pas effectuer d'initialisation bas niveau de la carte et de ne pas se recopier en RAM. Ces deux opérations ont déjà été effectuées par le bootstrap.

```
#define CONFIG_SKIP_LOWLEVEL_INIT
#define CONFIG_SKIP_RELOCATE_UBOOT
```

Ensuite on définit le type de processeur et de microcontrôleur utilisé.

```
#define CONFIG_ARM926EJS                    1
#define CONFIG_AT91SAM9263                  1
#define CONFIG_SAM9EBS                      1
#undef CONFIG_USE_IRQ
```

Il faut également définir les valeurs des clocks ainsi que la valeur de configuration des PLLs. Ces valeurs sont identiques à celles utilisées par le bootstrap.

```
#define AT91_SLOW_CLOCK                     32768
#define AT91_MAIN_CLOCK                     240000000
#define AT91_MASTER_CLOCK                   120000000
#define CFG_HZ                              1000000

#define PLLAR_VAL                           0x200F7F01
#define PLLBR_VAL                           0x00033F01
#define MCKR_VAL                            0x00000102

#define CFG_USE_MAIN_OSCILLATOR             1
```

Puis les informations relatives à la taille et à l'emplacement d'U-Boot en Flash sont définies. Notez que l'adresse de l'environnement doit être un multiple de la taille des secteurs. Dans le cas contraire U-Boot n'est pas capable de sauvegarder une configuration. C'est pour cela que la taille d'U-Boot a été largement surévaluée. L'adresse de base d'U-Boot plus cette taille nous donne 0x0040 000 qui se trouve être le début d'un secteur.

```
#define CFG_MONITOR_BASE                0x10010000
#define CFG_MONITOR_LEN                 0x30000
#define CFG_ENV_IS_IN_FLASH             1
#define CFG_ENV_ADDR                    ( CFG_MONITOR_BASE + CFG_MONITOR_LEN )
#define CFG_ENV_SECT_SIZE               0x20000
```

Ensuite quelques constantes en relation avec la ligne de commande tel que le prompt et la taille du tampon sont définies.

```
#define CFG_CONSOLE_IS_IN_ENV           1
#define CONFIG_BOOTDELAY                3
#define CFG_PROMPT                      "U-Boot> "
#define CFG_CBSIZE                      256
#define CFG_MAXARGS                     16
#define CFG_PBSIZE                      ( CFG_CBSIZE + sizeof(CFG_PROMPT) + 16 )
```

Puis il faut inclure `config_cmd_default.h` afin d'avoir accès aux commandes de base. De plus quelques commandes utiles mais qui sont en option sont ajoutées comme le support du DHCP, de l'USB.

```
#include <config_cmd_default.h>

#define CONFIG_CMDLINE_TAG              1
#define CONFIG_SETUP_MEMORY_TAGS        1
#define CONFIG_INITRD_TAG               1

#define CONFIG_CMD_PING                 1
#define CONFIG_CMD_DHCP                 1
#define CONFIG_CMD_DIAG                 1
#define CONFIG_CMD_USB                  1
```

L'adresse à laquelle le noyau sera chargé est aussi définie.

```
#define CFG_LOAD_ADDR                   0x24000000
#define CFG_MEMTEST_START               PHYS_SDRAM
#define CFG_MEMTEST_END                 ( CFG_MEMTEST_START + PHYS_SDRAM_SIZE \
                                         - 262144 )
```

L'adresse de base ainsi que la taille de la RAM sont également définies.

```
#define CONFIG_NR_DRAM_BANKS            1
#define PHYS_SDRAM                      0x20000000
#define PHYS_SDRAM_SIZE                 0x08000000
```

Les informations affichées lors du boot par le réseau avec BOOTP doivent également être définies.

```
#define CONFIG_BOOTP_BOOTFILESIZE       1
#define CONFIG_BOOTP_BOOTPATH           1
#define CONFIG_BOOTP_GATEWAY            1
#define CONFIG_BOOTP_HOSTNAME           1
```

Pour finir la sortie d'U-Boot est configurée sur l'USART0 avec un baudrate à 115200.

```
#define CONFIG_ATMEL_USART              1
#define CONFIG_USART0                   1
#undef CONFIG_USART1
#undef CONFIG_USART2
#undef CONFIG_USART3
#define CONFIG_BAUDRATE                 115200
#define CFG_BAUDRATE_TABLE              { 115200 , 19200, 38400, 57600, 9600 }
```

Certaines options pour Ethernet sont également définies. Entre autre l'utilisation du transceiver en mode RMII ainsi que l'adresse MAC de la carte. Celle choisie commence par 00:00:17, cette suite est normalement réservée à des fins de test et de développement. La suite de l'adresse quand à elle doit être choisie de manière à ne pas rentrer en conflit avec les autres cartes développées aux seins de l'école. Arbitrairement

l'adresse FF:00:00 a été choisie.

```
#define CONFIG_MACB                     1
#define CONFIG_RMII                     1
#define CONFIG_NET_MULTI                1
#define CONFIG_NET_RETRY_COUNT          5
#define CONFIG_RESET_PHY_R              1
#define CONFIG_ETHADDR                  00:00:17:FF:00:00
```

7.3.3 Répertoire du projet

Après avoir définit un nouveau fichier de configuration il faut créer un nouveau répertoire pour le projet. Pour ce faire le répertoire de la carte AT91SAM9263EK `board/atmel/at91sam9263ek` est copié sous le nom `sam9ebs` et le fichier `at91sam9263ek.c` renommé en `sam9ebs.c` selon la cible du `Makefile`.

```
$ cd ~/u-boot
$ cp -r board/atmel/at91sam9263ek board/atmel/sam9ebs
$ cd board/atmel/sam9ebs
$ mv at91sam926ek.c sam9ebs.c
```

Dans le `Makefile` locale il faut également modifier la liste des fichiers objets à créer, remplacer `at91sam9263.o` par `sam9ebs.o`.

```
# ~/u-boot/board/atmel/sam9ebs/Makefile

COBJS-y += sam9ebs.o
```

Le fichier `sam9ebs.c` nécessite quand à lui une seule modification. Une adresse est assignée au transceiver Ethernet en fonction de ses strap options (option de démarrage). Ces options dépendent des valeurs par défaut du composant et des éventuelles pullup ou pulldown montées sur la carte. Dans notre cas l'adresse du transceiver est 0x01. Cependant la valeur utilisée par la carte AT91SAM9263EK est 0x00. Il faut donc changer cette adresse dans la fonction `macb_eth_initialize`.

```
# ~/u-boot/board/atmel/sam9ebs/sam9ebs.c

int board_eth_init(bd_t *bis)
{
	int rc = 0;
#ifdef CONFIG_MACB
	rc = macb_eth_initialize(0, (void *)AT91_BASE_EMAC, 0x01);
#endif
	return rc;
}
```

7.3.4 Modification de la link address

Etant donné qu'U-Boot n'est pas chargé en mémoire par un loader (chargeur), aucune réallocation d'adresse n'est effectuée. En clair toutes les adresses du programme sont calculées à partir de l'adresse présente dans le fichier `board/sam9ebs/config.mk`. Il faut donc modifier cette adresse afin qu'elle corresponde à l'emplacement où l'application du bootstrap chargera U-Boot.

7.3.5 Compilation

Il faut commencer par nettoyer l'environnement à l'aide de la cible make `distclean`. Ensuite on configure le projet U-Boot avec notre fichier de configuration `sam9ebs_config`. Pour finir on lance `make`.

```
$ cd ~/u-boot
$ make distclean
$ make sam9ebs_config
$ make
```

Le résultat est plusieurs fichiers u-boot:

- u-boot au format ELF, exécutable Linux
- u-boot.bin au format binaire ARM
- u-boot.srec au format binaire pour processeur Motorola
- u-boot.map description bas niveau de l'application

Le fichier u-boot est copié dans la mémoire Flash de la carte à l'aide du BDI2000 décrit au chapitre 5, page 30.

7.4 Chargement de Linux

Une fois U-Boot copié en Flash à l'adresse convenue par l'application bootstrap il faut définir trois variables afin de pouvoir booter Linux à partir du réseau:

- bootcmd définit la liste des commandes automatiquement exécutées à la fin du délai de réflection
- bootfile définit l'application à charger
- bootargs définit les arguments passés à cette application

Ces variables sont définies à l'aide de la commande U-Boot `setenv variable=value`.

Pour Linux le bootfile est `zImage.img` comme expliqué plus tard au chapitre 8.4.1, page 55.

Bootcmd est définit à `dhcp ; bootm`. La commande `dhcp` est d'abord exécutée, celle-ci charge le fichier spécifié par bootfile à l'adresse définie dans le fichier `sam9ebs.h` par la macro `CFG_LOAD_ADDR`. Puis la commande `bootm` exécute l'application à partir de cette adresse mémoire.

Bootargs quand à lui passe à Linux les arguments suivant:

- `console=ttyS1,115200` utilise le port série femelle avec un baudrate à `115200`
- `root=/dev/nfs` monte le système de fichier racine par NFS
- `init=/bin/bash` démarre bash une fois le système initialisé
- nfsroot=153.109.5.141 :/mnt/rootfs indique l'adresse du serveur NFS et le répertoire du système de fichier sur l'hôte
- `ip=::::::dhcp` indique que l'adresse IP de la carte est spécifiée par DHCP

```
U-Boot> setenv bootfile zImage.img
U-Boot> setenv bootcmd dhcp ; bootm
U-Boot> setenv bootargs console=ttyS1,115200 root=/dev/nfs init=/bin/bash
                            nfsroot=153.109.5.141:/mnt/rootfs ip=::::::dhcp
```

Pour plus d'information la commande `help` fournit la liste des commandes disponibles avec une courte description.

7.5 Patch

De manière identique au bootstrap un patch a été créé pour U-Boot. Il se situe dans `u-boot/src/u-boot-2008.10-rc1_sam9ebs.patch`.

Les commandes suivantes permettent de le créer.

```
$ cd ~/u-boot/src
$ tar xvjf u-boot-2008.10-rc1.tar.bz2
$ mv u-boot-2008.10-rc1 ../base
$ diff -rupN base/ u-boot-2008.10-rc1/ > u-boot-sam9ebs.patch
```

Et celles-ci après de l'appliquer.

```
$ cd u-boot/src
$ tar xvjf ./src/u-boot-2008.10-rc1.tar.bz2
$ patch -p1 < ./src/u-boot-2008.10-rc1-sam9ebs.patch
```

7.6 Références

- Manuel d'U-Boot:
  http://www.denx.de/wiki/DULG/Manual
- Fichier d'aide d'U-Boot:
  cdrom/u-boot/doc/README
- Sources du projet ARMEBS3:
  https://svn.hevs.ch/trac/armebs3/login/browser/u-boot/trunk
- GNU device drivers, AT91SAM9263ek patches pages:
  http://www.gnudd.com/wd/at91sam9263ek-nor.html

7.7 Sources

- U-Boot:
  ftp://ftp.denx.de/pub/u-boot/
  cdrom/u-boot/src/u-boot-2008.10-rc1.tar.bz2
- Patch pour la carte sam9ebs:
  cdrom/u-boot/src/ u-boot-2008.10-rc1_sam9ebs.patch

8 Système GNU/Linux

L'utilisation croissante de Linux dans des systèmes embarqués, son caractère open source ainsi que la qualité des outils de développement associé en font le candidat par excellence pour être porté sur cette carte.

8.1 Préparation des sources

Les sources de la dernière version stable (2.6.27.4 en date du 21 novembre 2008) sont téléchargées depuis le site kernel.org, le patch pour système AT91 depuis maxim.org.za. Après avoir détaré les sources, celles-ci sont patchées.

```
$ mkdir ~/linux
$ cd ~/linux
$ mkdir src
$ cd src
$ wget http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.27.4.tar.bz2
$ wget http://maxim.org.za/AT91RM9200/2.6/2.6.27-at91.patch.gz
$ tar xvjf linux-2.6.27.4.tar.bz2
$ mv linux-2.6.27.4 ../
$ cd ../linux-2.6.27.4
$ zcat ../src/2.6.27-at91.patch.gz | patch -p1
```

8.2 Configuration de l'environnement de construction

Dans le Makefile à la racine des sources du noyau, les champs ARCH et CROSS_COMPILE sont définis de manière à compiler le noyau pour une cible ARM et à utiliser la toolchain ELDK, préinstallée au chapitre ELDK, page 28.

```
# linux/linux-2.6.27.4/Makefile

#...

ARCH                ?= arm
CROSS_COMPILE       ?=/opt/eldk/usr/bin/arm-linux-

#...
```

8.3 Configuration du noyau

Le fichier de configuration de la carte d'évaluation AT91SAM9263EK présent dans `arch/arm/configs` est copié dans le même répertoire sous un nouveau nom: `sam9ebs_defconfig`.

```
$ cd ~/linux/linux-2.6.27.4/
$ cp arch/arm/configs/at91sam9263ek_defconfig arch/arm/configs/sam9ebs_defconfig
```

Cette configuration nécessite cependant quelques adaptations à l'aide de menuconfig.

8.3.1 Menuconfig

Menuconfig est une cible make spéciale du noyau Linux permettant de modifier graphiquement la configuration du noyau. Le paquet ncurses-dev (bilbliothèque graphique pour interface textuelle) est nécessaire, il faut donc l'installer.

```
$ sudo apt-get install ncurses-dev
```

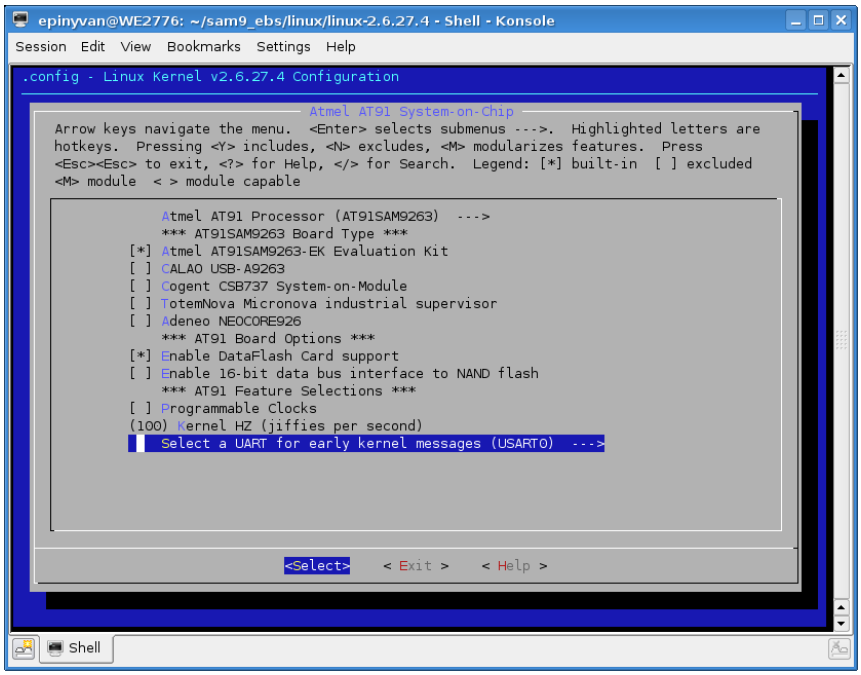
Elle se lance simplement avec la commande `make menuconfig` dans le répertoire du noyau.

```
$ cd ~/linux/linux-2.6.27.4/
$ make menuconfig
```

Une fois lancé, le fichier de configuration sam9ebs_defconfig est chargé. Dans un premier temps, seul les modifications strictement nécessaires sont effectuées.

Il faut modifier le port série utilisé par le noyau comme sortie standard et sélectionner USART0 comme port série plutôt que DGBU. Cette option se situe dans:

```
System Type
    Atmel AT91 System-on-Chip
        Select a UART for early kernel messages
```



Menuconfig à l'action

Il faut également activer le support du DHCP afin de pouvoir monter correctement par NFS le système de fichier racine (root files system) avec assignation automatique de l'adresse IP de la carte.

```
Networking support
      Networking options
            TCP/IP networking
                  IP: kernel level autoconfiguration
```

Une fois ces modifications effectuées, la configuration est sauvegardée dans le répertoire `arch/arm/configs/` sous `sam9ebs_defconfig`.

8.4 Compilation du noyau

Puis le noyau peut être compilé. La cible `sam9ebs_defconfig` doit tout d'abord être exécutée afin d'écrire la configuration dans le fichier `.config`, qui est le fichier réellement utilisé lors de la compilation. La cible `clean` est lancée pour garantir la propreté de l'environnement. Pour finir la construction de l'image, puis des modules sont lancés avec respectivement les cibles `zImage`, et `modules`.

```
$ cd ~/linux/ linux-2.6.27.4
$ make sam9ebs_defconfig
$ make clean
$ make zImage
$ make modules
```

La compilation effectuée, une première image est disponible à la racine des sources: `vmlinux`. Celle-ci est de taille plutôt conséquente 4MB. Il est donc préférable d'utiliser la seconde compressée: `zImage` dont la taille n'est que de 1.4MB. `zImage` est présent dans le répertoire `arch/arm/boot`. Cependant ce gain de taille se payera lors du démarrage de Linux par le temps nécessaires à sa décompression.

8.4.1 Création de l'image

L'image du noyau s'attend à être exécutée à partir d'une adresse connue dans la mémoire. U-boot comprend dans son répertoire `tools/` un outil permettant d'indiquer au noyau cette adresse: `mkimage`. Une fois l'image copiée depuis `arch/arm/boot` dans le répertoire `tools/` d'U-Boot, il faut exécuter `mkimage` avec les arguments suivants.

```
$ cd u-boot/tools
$ cp ~/linux/linux-2.6.27.4/arch/arm/boot/zImage ./
$ ./mkimage -A arm -O linux -T kernel -C none -a 0x25000000 -e 0x25000000 -n
"Linux_2.6.27.4_arm" -d zImage zImage.img
```

Les arguments de mkimage sont:

- -A pour l'architecture
- -O pour le système d'exploitation
- -T pour le type d'image
- -C pour la compression
- -a pour l'adresse où le noyau sera décompressé
- -e pour l'adresse du point d'entrée
- -n pour définir un nom à l'image
- -d pour les données à la source de l'image

Notez que cette adresse est celle à partir d'où le noyau s'exécutera une fois décompressé, elle doit donc être différente de l'adresse où U-Boot à copier l'image compressée.

8.5 Système de fichiers

Le noyau Linux n'est qu'une pièce d'un système GNU/Linux fonctionnel. De nombreux autres outils, fichiers de configuration et périphériques sont nécessaires. Ceux-ci sont regroupés dans le système de fichier.

Plusieurs outils permettent de construire un système de fichier à partir des sources dont Buildroot et OpenEmbedded. Cependant Buildroot utilise la uClibc comme librairie C standard. Pour des raisons de compatibilité des binaires il a été choisi de ne pas utilisé Buildroot. De son côté OpenEmbedded est extrêmement gourmant en temps et espace mémoire.

L'autre possibilité est de récupérer les binaires d'une distribution déjà compilée pour notre architecture. L'outil `debootstrap` de Debian permet cela.

8.5.1 Debootstrap

Debootstrap est un outil qui permet de créer un système Debian de base dans une cible à partir d'un miroir à l'aide de scripts.

Fonctionnement de debootstrap

La première chose à faire est d'installer `debootstrap`.

```
$ sudo apt-get install debootstrap
```

Ensuite il faut créer le futur point de montage de notre système de fichier `rootfs` dans le répertoire `/mnt` de préférence. Puis un pseudo disque est créé à l'aide de `dd` qui prend en argument un ficher d'entrée `if`, un fichier de sortie `of`, la taille des secteurs `bs` et leur nombre `count`. Le fichier d'entrée est `/dev/zero` afin d'avoir un pseudo disque vierge. Celui de sortie est notre nouveau fichier `/opt/rawdisk`. Quand aux secteurs leur nombre est de 500'000 et leur taille de 512 bytes, ce qui donne 256MB. Cette taille est suffisante puisque le système de base de fichier occupera à terme 114MB.

```
$ cd /mnt/
$ sudo mkdir rootfs
$ sudo dd if=/dev/zero of=/opt/rawdisk bs=512 count=500000
```

Une fois le fichier créé, il faut le formater en ext2 avec `mkfs.ext2`. Ext2 est le système de fichier non journalisé par défaut sous Linux. Cette commande affiche un avertissement nous indiquant que le fichier n'est pas un périphérique réel. A la question 'Voulez-vous continuer ?' répondez oui. Puis il faut monter le pseudo disque `/opt/rawfs` sur `/mnt/rootfs` avec l'option `-o loop`.

```
$ sudo mkfs.ext2 /opt/rawdisk
$ sudo mount -o loop /opt/rawdisk/ rootfs/
```

La commande `debootstrap` prend en argument l'architecture de la cible `--arch`, la version de la distribution, le répertoire cible et le miroir. L'architecture est `arm`. La version est `lenny`, qui est la version testing à l'heure actuelle (21 novembre 2008). Le répertoire est notre système de fichier monté `/mnt/rootfs`. Le miroir est celui de l'ETH à Zurich. De plus l'option `--foreign` est utilisée afin d'indiquer

à debootstrap que l'architecture cible est différente de l'hôte et que seul la première partie du debootstrap doit être effectuée. Sans celà debootstrap échoue losqu'il tente de se chrooté (changement de la racine du système) dans le nouveau système de fichier pour terminer l'installation du système de fichier. Ceci est normal puisque les binaires du nouvel environnement ont étés compilés pour un processeur ARM et non un x86.

```
$ sudo debootstrap --foreign --arch arm testing /mnt/rootfs/ http://ftp.ch.debian.org/debian/
```

Debootstrap n'a cependant pas créé les fichiers spéciaux `/dev/console` et `/dev/ttyS1` nécessaires afin d'avoir une console valide. Ceux-ci doivent être créés manuellement avec `mknod` qui prend en argument le nom d'un nouveau fichier spécial, son type et leur numéro de majeur et de mineur. Le type peut être soit `c` pour caractère, périphérique série (console), soit `b` pour bloque (disque dur). Dans notre cas les deux sont des périphériques en mode caractère. Le numéro de majeur permet au noyau de faire correspondre le périphérique à un driver. C'est pourquoi des périphériques semblables auront vraisemblablement le même numéro de majeur. Le numéro de mineur quand à lui permet de distingué de manière unique un périphérique servi par un driver. Les numéros utilisés sont 5,1 pour la console et 4,28 pour ttyS0.

```
$ sudo mknod /mnt/rootfs/dev/console c 5 1
$ sudo mknod /mnt/rootfs/dev/ttyS1 c 4 28
```

Pour finir le propriétaire du répertoire `rootfs` est changé afin de pouvoir y lire et écrire en tant que simple utilisateur.

```
$ sudo chown -R YOUR_NAME:YOUR_NAME /mnt/rootfs
```

Notez qu'à l'avenir suite à un changement dans les appels systèmes, l'architecture ARM ne sera plus supportée par la branche `arm` mais `armel`.

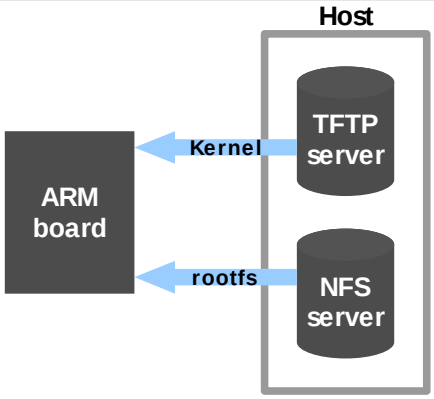
8.6 Test

Lors du démarage Linux s'attend à trouver et exécuter différents fichiers tel que `inittab`, fichiers que debootstrap n'a pas créé. La solution la plus simple est d'ajouter **`init=/bin/bash`** à la ligne d'arguments passée au noyau par U-Boot.

La mémoire Flash disponible étant plutôt limitée (8MB) il est impossible d'y copier le système de fichier, c'est pourquoi nous y accédons par NFS (Network File System).

Pour ce faire il faut installer `nfs-kernel-server`, le serveur atftpd quand à lui est normalement déjà installé.

```
$ apt-get install nfs-kernel-server
```



Boot avec noyau et système de fichier fournit par l'hôte

Afin que la cible puisse accéder au serveur NFS la ligne suivante doit être rajoutée au fichier `/etc/exports` de l'hôte. Celle-ci commence par indiquer le répertoire du système de fichier de la cible suivie du couple adresse/masque définissant le sous-réseau où se trouvent l'hôte et de la cible. Le tout suivit de quelques options afin que NFS fonctionne correctement.

```
# /etc/exports

/mnt/rootfs 153.109.5.0/255.255.255.0(async,rw,no_root_squash,no_all_squash,subtree_check)
```

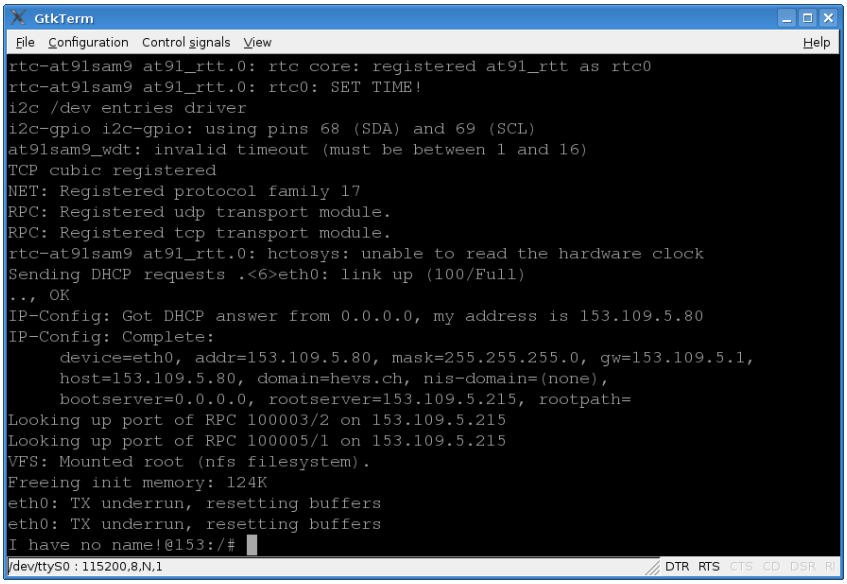
Pour que la nouvelle configuration soit prise en compte, il faut redémarrer le serveur grâce au script `nfs-kernel-server` présent dans `/etc/init.d` en lui passant `restart` en argument.

```
$ sudo /etc/init.d/nfs-kernel-server restart
```

De même en phase de développement il est intéressant de disposer d'un noyau sur l'hôte plutôt qu'en Flash à cause des changements fréquents. Le noyau doit donc être copié dans le répertoire `/var/lib/tftpboot`.

Il faut également vérifier que les variables d'environnemnts soient définies correctement pour le démarrage de Linux par U-Boot comme expliqué au chapitre 7.4, page 51.

A ce stade le système GNU/Linux peut booté. Il ne faut pas oublier de connecter la cible à l'hôte à l'aide d'un port série et de démarrer `GTKterm` avec un baudrate de 115200.



Bash une fois Linux démarré

Un schéma complet de la séquence du boot est disponible en annexe F, page 77.

Il est désormé possible de créer une première application cross développée. Pour ce faire un simple 'hello world' est écrit sur l'hôte. Puis cette application est compilée à l'aide de toolchain ELDK et copiée dans le système de fichier de la cible.

```
$ cd ~
$ /opt/eldk/usr/bin/arm-linux-gcc -o hello hello.c
$ cp ~/hello /mnt/rootfs/home/
```

Il ne reste plus qu'à l'exécuter depuis la cible.

```
target$ /home/hello
```

```
I have no name!@153:/home# ./hello

hello everybody
```

Hello world exécuté sur la cible

8.7 Copie du noyau en Flash

Afin d'accélérer la séquence de boot, il est possible de le copier l'image du noyau en Flash plutôt que de la récupérer depuis le serveur TFTP.

Pour commencer il faut dans la console d'U-Boot charger l'image du noyau en RAM à une adresse connue, typiquement 0x2400 0000 grâce à la commande `dhcp`.

```
U-Boot> dhcp
```

```
U-Boot> dhcp

macb0: link up, 100Mbps full-duplex (lpa: 0x45e1)
BOOTP broadcast 1
DHCP client bound to address 153.109.5.80
Using macb0 device
TFTP from server 153.109.5.215; our IP address is 153.109.5.80
Filename 'zImage.img'.
Load address: 0x24000000
Loading: ##################################################################
         ############################
done
Bytes transferred = 1358616 (14bb18 hex)
```

Exécution de la commande dhcp

Puis il faut définir la variable d'environnement ipaddr à l'aide de `setenv ipaddr` IP_ADDR. Cette adresse est indiquée par U-Boot lors de l'exécution de `tftp`.

```
U-Boot> setenv ipaddr 153.109.5.80
```

Ensuite il faut effacer une partie de la Flash pour y écrire le noyau. Il faut veiller à ne pas écraser le code du Bootstrap et d'U-Boot. De plus certains secteurs sont en lecture seule, marqué par RO et visible avec la commande `flinfo`.

Une fois l'adresse choisie, il faut effectuer une remise à zéro avec `erase` de la taille d'un noyau également indiquée par la commande `dhcp`. Après celà on copie avec `cp.b` le noyau de son adresse en RAM à sa nouvelle adresse en Flash avec le bon nombre de bytes.

```
U-Boot> erase 0x10400000 +0x14bb18
U-Boot> cp.b 0x24000000 0x10400000 0x14bb18
```

Pour finir on modifie la commande de boot afin de booter à partir de l'emplacement du noyau.

```
U-Boot> setenv bootcmd bootm 0x24000000
```

Naturellement il ne faut pas oublier de sauvegarder la nouvelle configuration.

```
U-Boot> saveenv
```

8.8 Références

- Linux Kernel:
  http://kernel.org/
- Compilation croisée avec ARM:
  http://www.arm.linux.org.uk/docs/kerncomp.php
- Introduction à U-Boot:
  http://www.linuxdevices.com/articles/AT5085702347.html
- Wiki Debian:
  http://wiki.debian.org/EmDebian/CrossDebootstrap
- Page de manuel de debootstrap:
  $ man debootstrap,
  (konqueror) man: debootstrap

8.9 Sources

- Linux Kernel 2.6.27.4:
  http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.27.4.tar.bz2
  http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.27.4.tar.bz2.sign
  cdrom/linux/src/linux-2.6.27.4.tar.bz2
- AT91 2.6.27.x kernel patch:
  http://maxim.org.za/AT91RM9200/2.6/2.6.27-at91.patch.gz
  cdrom/linux/src/2.6.27-at91.patch.gz
- Fichier de configuration sam9ebs_deffconfig:
  cdrom/linux/src/sam9ebs_deffconfig

9 Conclusion

Les objectifs fixés ont pu être en majeur partie atteints. La carte a été réalisée. U-Boot a pu être porté avec succès sur celle-ci et ceux malgré les bugs présents sur le microcontrôleur et dans les sources du bootstrap. Le système GNU/Linux à également pu être porté sur la nouvelle carte. Cependant aucune mesure de performance n'a peu être réalisée. De plus le bon fonctionnement de tous les périphériques n'a pas pu être validé.

A l'avenir il serait intéressant de poursuivre les tests sur les périphériques non testés. De même évaluer les capacités de traitement parallèle de l'AT91SAM9263 est une priorité. De son côté le système GNU/Linux porté bien qu'opérationnel reste très primitif et mérite des améliorations.

Vendredi, le 21 novembre 2008
Yvan Epiney

Annexes

A Liste des composants

A.1 List of components

#	Part	Description	Retailer	Reference	Price
1	U1	ADP2107, step-down converter 2A	Farnell	1461532	7.35
2	U2..3	ADP1715ARMZ, linear regulator 500mA	Farnell	1461508	3.10
2	U4..5	NC7SZ08M5, and gate	Farnell	1013807	1.25
1	U6	NC7WZ07P, buffer	Farnell	1470998	0.42
1	U7	MAX823, reset controller	Farnell	1188047	4.95
1	U8	AT91SAM9263B-CU, uC	MSC	AT91SAM9263B-CU	24.00
1	U9	DS1374, RTC	Farnell	1379774	5.25
1	U10	AD9889, HDMI transmitter	Digikey	AD9889BBSTZ-80-ND	10.92
1	U11	SN74AUP1T97, level shifter	Farnell	1053539	1.20
1	U12	AD1981B, audio uC	Analog Devices	sample	
1	U13	DP83848, Ethernet transceiver	Farnell	1286816	8.30
1	U14	Quartz 50MHz	School		
1	U15	LM3526M-L, USB power switch	Farnell	1202978	3.15
1	U16	MCP2551-I/SN, CAN transceiver	Farnell	9758569	2.90
1	U17	ADM3202A, RS232 transceiver	Farnell	1333250	3.20
4	U18..21	K4S560832E-TC75, SDRAM 32MB	School		
1	U22	28F128J3A, Flash 16MB	School		
1	U23	MT45W2MW16PGA70-WT, PSRAM 4MB	Digikey	557-1316-1-ND	6.84
1	J1	DC10B, Power connector	School		
1	J2	CONNB2X5-2MM-FCI, JTAG connector	School		
1	J3	MOLEX 47266, HDMI connector	Farnell	1516663	2.50
1	J4	SMC-B-2X25MC, LCD/Touch screen connector	School		
2	J5..6	JACK-S6-3.5, audio Jack connector	Farnell	149932	3.10
1	J7	IDC2X20MC, IDE connector	School		
1	J8	SDMF-10915W011, SD Card interface	School		
1	J9	RJ45-0810-1X1T, Ethernet RJ45 connector	School		
1	J10	USB-B-SMD, USB Device connector	Farnell	1321918	1.85
2	J11, J13	USB-A-SMD, USB Host connector	Farnell	1308874	0.925
2	J12, J14	RJ45-8PBV, CAN RJ45 connector	School		
1	J15	IDC2X13MC, GPIO connector	School		
1	J16	SUB-D-9F-2.54-MH, RS232 Female connector	School		
1	J17	SUB-D-9M-2.54-MH, RS232 Male connector	School		
1	G1	K107, batteries support / 20 3.2	School		
1	LD1	LED_SMD_XX1101W, Power on Led - green	School		
5	LD2..6	LED_SMD_XX1101W, User's Leds - red	School		
5	S1..5	PUSHBUTT-DTSM, User's & reset button	School		

2	X1, X3	CRYSTAL-FC-255, 32768Hz	School		
1	X2	CRYSTAL-CS10, 16MHz	School		
1	X4	CRYSTAL-SS3, 24.576MHz			
1	L1	SRR1208-2R5M, Self 2.5uH	Distrelec	350565	2.26
7	LC1..7	NFE31P-33, Ferrite SMD	School		
5	FB1..5	RS0805, Ferrite SMD 2200 Ohms	Farnell	1635726	0.35
1	F1	FUSE-SMD154, 64mA	Farnell	9922121	2.25
2	C90, C92	CAPS0805-MM, 15p	School		
2	C36..37	CAPS0805-MM, 17p	School		
4	C34..35, C64..65	CAPS0805-MM, 22p	School		
1	C91	CAPS0805-MM, 33p	School		
4	C96..97, C100..101	CAPS0805-MM, 47p	School		
1	C4	CAPS0805-MM, 120p	School		
4	C62..63, C66..67	CAPS0805-MM, 270p	School		
5	C39, C73..76	CAPS0805-MM, 470p	School		
1	C3	CAPS0805-MM, 1n	School		
1	C41	CAPS0805-MM, 2n	School		
1	C42	CAPS0805-MM, 20n	School		
1	C40	CAPS0805-MM, 47n	School		
92	C5, C12..33, C38, C43..61, C77..80, C89, C93, C98..99, C102..141, C143	CAPS0805-MM, 100n	School		
8	C68, C71..72, C81, C81..86	CAPS0805-MM, 1u	School		
4	C8..11	CAPS0805-MM, 2.2u	School		
3	C1..2, C6	CAPS6032P (TAJ-C), 10u low ESR	School		
1	C82	CAPS6032P (TAJ-C), 10u	School		
3	C1..2, C6	CAPS6032P (TAJ-C), 10u low ESR	School		
2	C94..95	CAPS6032P (TAJ-C), 33u	School		
2	C69..70	CAPS6032P (TAJ-C), 100u	Farnell	1432275	1.30
4	R70..72, R74	RS0805-MM, 0	School		
1	R1	RS0805-MM, 10	School		
2	R50..51	RS0805-MM, 27	School		
4	R54..55, R59..60	RS0805-MM, 39	School		
1	R3	RS0805-MM, 56	School		
1	R6	RS0805-MM, 100	School		
4	R5, R20, R46, R48	RS0805-MM, 120	School		
4	R4, R19, R45, R47	RS0805-MM, 150	School		
5	R65..69	RS0805-MM, 160	School		
1	R7	RS0805-MM, 200	School		
1	R16	RS0805-MM, 510	School		
1	R21	RS0805-MM, 887	School		
4	R24..26, R28	RS0805-MM, 1k	School		

3	R17..18, R53	RS0805-MM, 1.5k	School		
1	R15	RS0805-MM, 1.96k	Farnell	157-5825	1.25
4	R22.23, R42, R44	RS0805-MM, 2.2k	School		
5	R27, R29..31, R58	RS0805-MM, 4.7k	School		
1	R43	RS0805-MM, 4.87k	Farnell	157-5866	1.25
9	R8..12, R14, R32..33, R38	RS0805-MM, 10k	School		
9	R13, R39..41, R49, R56..57, R61..62	RS0805-MM, 15k	School		
1	R52	RS0805-MM, 22k	School		
1	R73	RS0805-MM, 47k	School		
4	R34..37	RS0805-MM, 68k	School		
1	R2	RS0805-MM, 70k	School		
1	R63..64	RS0805-MM, 100k	School		
	Total	Without school provided materials and delivery charges			**110.94**

Notes: All the prices are in swiss franc, for components payable in other currency, the following rates have been applied : 1€ = 1.60Chf, 1$ = 1.20Chf

B Caractéristiques des ICs

B.1 U4..5 And Gate NC7SZ08M5

Parameter	*Value, **typic***	*Unit*
Supply voltage	1.65 .. **3.3** .. 5.50	V

B.2 U6 Buffer NC7WZ07P

Parameter	*Value, **typic***	*Unit*
Supply voltage	1.65 .. **3.3** .. 5.50	V

B.3 U7 Reset Controller MAX823

Parameter	*Value, **typic***	*Unit*
Supply voltage	1.00 .. **3.3** .. 5.50	V

B.4 U8 uC AT91SAM9263

Parameter	*Value, **typic***	*Unit*
Core supply voltage	1.08 .. **1.2** .. 1.32	V
Bus supply voltage	1.65 .. 1.95 | 3.0 .. **3.3** .. 3.6	V
I/Os supply voltage #0	2.7 .. **3.3** .. 3.6	V
I/Os supply voltage #1	1.65 .. **3.3** .. 3.6	V
Power consumption, max	281	mA

B.5 U9 RTC DS1374U-18

Parameter	*Value, **typic***	*Unit*
Operation supply voltage	1.71 .. **1.8** .. 1.89	V
Standby supply voltage	1.3 .. **3.0** .. 3.7	V
Power-fail voltage	1.51 .. *1.6* .. 1.71	V
Operation power consumption, max	150	uA
Standby power consumption, max	700	nA
Size of binary counter	32	bit
I^2C address	0x68	/

B.6 U10 HDMI Transmitter AD9889

Parameter	*Value, **typic***	*Unit*
Supply voltage	1.71 .. **1.8** .. 1.89	V
Power consumption, max	205	mA
I^2C address, depends of power down	0x72 | 0x7A	/

B.7 U11 Level Shifter SN74AUP1T98DCK

Parameter	*Value, **typic***	*Unit*
Supply voltage	2.30 .. **3.3** .. 3.60	V

B.8 U12 Audio uC AD1981B

Parameter	*Value, **typic***	*Unit*
Supply voltage, analog	4.50 .. **5.00** .. 5.50	V
Supply voltage, digital	3.00 .. **3.30** .. 3.47	V
Analog power consumption, typic	50	mA
Digital power consumption, typic	46	mA

B.9 U13 Ethernet Transceiver DP83848C

Parameter	*Value, **typic***	*Unit*
Supply voltage	-0.5 .. **3.3** .. 4.2	V
Power consumption	92	mA

B.10 U15 USB Power Switch LM3526M-L

Parameter	*Value, **typic***	*Unit*
Supply voltage	2.7 .. **5.0** .. 5.5	V
Power consumption, max	200	uA

B.11 U16 CAN Transceiver MCP2551-SN

Parameter	*Value, **typic***	*Unit*
Supply voltage	4.5 .. **5.0** .. 5.5	V
Power consumption, max	75	mA

B.12 U17 RS232 Transceiver ADM3202ARN

Parameter	*Value, **typic***	*Unit*
Supply voltage	3.0 .. **3.3** .. 5.5	V
Power consumption, max	10	mA

B.13 U18..21 DRAM K45560832E-TC75

Parameter	*Value, **typic***	*Unit*
Supply voltage	3.0 .. **3.3** .. 3.6	V
Power consumption, max (refresh)	180	mA
Maximum operating frequency (period)	133 (7.52)	MHz (ns)
Refresh frequency (period)	15.624 (64)	Hz (ms)
Capacity, organization	32 x 8	Mbits
Column, number of address bit	10	/
Row, number of address bit	13	/
Number of banks	4	/

B.14 U22 Flash NOR TE28F128-J3

Parameter	*Value, **typic***	*Unit*
Supply voltage	2.7 .. **3.3** .. 3.6	V
Power consumption, max (refresh)	80	mA
Read/Write cycle time (min)	120	ns

B.15 U23 PSRAM MT45W2MW16PGA

Parameter	*Value, **typic***	*Unit*
Core supply voltage	1.7 .. **1.8** .. 1.95	V
I/Os supply voltage	1.7 .. **3.3** .. 3.6	V
Power consumption, max	20	mA
Access time, max	70	ns

C Consomation électrique

C.1 Backup power consumption

Voltage	*Component*	*Action*	*Consumption [mA]*
1.8 V			
	DS1374-18 / RTC	Running without squared clock signal	0.7
			0.7

C.2 Maximum power consumption

Voltage	*Component*	*Action*	*Consumption [mA]*
1.2 V			
	AT91SAM9263 / ARM9 core	Running full load algorithm	70.0
	AT91SAM9263 / uC core	See table on the next page for more info	34.9
			104.9
1.8 V			
	DS1374-18 / RTC	Reseting and interrupting	0.3
	AD9889 / video transmitter	80Mhz random pattern	205.0
	MT45W2MW16PGA / PSRAM	Asynchronous random R/W	20.0
			225.3
3.3 V			
	ADP1715ARMZ / 1.2V power supply	Approximation, without loss	104.9
	ADP1715ARMZ / 1.8V power supply	Approximation, without loss	225.3
	AD1981B / audio uC – digital power supply	Not max but typic	46.0
	TE28F128J3C / Flash NOR	Erasing	80.0
	K4S560832E / SDRAM 4x	Refreshing (each 64ms)	720.0
	DP83848C / Ethernet transceiver	10BASE-T full duplex (typic)	92.0
	ADM32202 / RS232 transceiver	Running with 3kΩ load	10.0
	Multimedia Card	Estimation	100.0
	GPIO	Estimation	200.0
			1578.2
5.0 V			
	ADP2107 / 3.3V power supply	Approximation with a yield of 92%	1715.5
	LM3526 / USB power switch	USB device full load, 2x 500mA	1000.0
	MCP2551 / CAN transceiver	Sending, dominant	75.0
	AD1981B / audio uC – analog power supply	Not max but typic	50.0
			2840.5

Note：Power consumption of small components such logic gates and so on was ignored.

C.3 uC SAM9 detailed power consumption

Voltage / Part used	*Component part*	*Consumption per MHz [uA/MHz]*	*Consumption [mA] @ 120MHz*
1.2 V			
2	PIO A, B (I/O port controller)	5	1.20
3	PIO C, D, E (I/O port controller)	14	5.04
2	USART (serial port)	13	3.12
2	USB host	12	2.88
1	USB device	9	1.08
1	I2C (RTC & video transmitter configuration)	2	0.24
1	SPI (touch screen)	9	1.08
1	MCI (multimedia card interface)	13	1.56
1	CAN	50	6.00
1	Ethernet	40	4.80
1	LCD	45	5.40
1	ISI (image sensor interface)	8	0.96
1	AC97 (audio)	13	1.56
			34.92

Note : The peripheral clock equals ½ of the ARM9 core clock which reach a maximum of 240MHz in our case.

D Connectique

D.1 JTAG : 2x5 connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	VCC	3.3V	power supply	2	NC	GND	force power on
3	TMS	input	test mode select	4	NC	GND	return clock
5	TCK	input	clock	6	RST	input, low	test reset
7	TDO	output	data output	8	BOARD_RST	I/O, low	board reset
9	TDI	input	data input	10	GND	GND	ground

D.2 LCD / touch screen : 2x25MC connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1a	GND	GND	ground	1b	GND	GND	ground
2a	GND	GND	ground	2b	GND	GND	ground
3a	LCD_V_SYNC	output	row pulse	3b	LCD_H_SYNC	output	column pulse
4a	LCD_DOT_CK	output	pixel clock	4b	LCD_EN	output	data enable
5a	GND	GND	ground	5b	RED_5	output	pixel red bit 5
6a	RED_4	output	pixel red bit 4	6b	RED_3	output	pixel red bit 3
7a	RED_2	output	pixel red bit 3	7b	RED_1	output	pixel red bit 1
8a	RED_0	output	pixel red bit 0	8b	GND	GND	ground
9a	GREEN_5	output	pixel green bit 5	9b	GREEN_4	output	pixel green bit 4
10a	GREEN_3	output	pixel green bit 3	10b	GREEN_2	output	pixel green bit 2
11a	GREEN_1	output	pixel green bit 1	11b	GREEN_0	output	pixel green bit 0
12a	GND	GND	ground	12b	BLUE_5	output	pixel blue bit 5
13a	BLUE_4	output	pixel blue bit 4	13b	BLUE_3	output	pixel blue bit 3
14a	BLUE_2	output	pixel blue bit 2	14b	BLUE_1	output	pixel blue bit 1
15a	BLUE_0	output	pixel blue bit 0	15b	GND	GND	ground
16a	NC	/	not connected	16b	NC	/	not connected
17a	GND	GND	ground	17b	TS_CS	output, low	touch scrn cs
18a	GND	GND	ground	18b	TS_DIN	output	touchscrn data in
19a	GND	GND	ground	19b	TS_DOUT	input	touchscrn data out
20a	GND	GND	ground	20b	TS_DCLK	output	touchscrn data clock
21a	GND	GND	ground	21b	TS_IRQ	input	touchscrn iterrupt
22a	GND	GND	ground	22b	NC	/	not connected
23a	GND	GND	ground	23b	NC	/	not connected
24a	NC	/	not connected	24b	GND	GND	ground
25a	GND	GND	ground	25b	GND	GND	ground

D.3 Video : HDMI connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	Data2+	output	data 2 +	2	Data2 Shield	GND	data 2 protection
3	Data2-	output	data 2 -	4	Data1+	output	data 1 +
5	Data1 Shield	GND	data 1 protection	6	Data1-	output	data 1 -
7	Data0+	output	data 0 +	8	Data0 Shield	GND	data 0 protection
9	Data0-	output	data 0 -	10	Clock+	output	clock +
11	Clock Shield	GND	clock protection	12	Clock-	output	clock -
13	NC	GND	consumer electronic control	14	Reserved	GND	not connected
15	SCL	I/O	I2C clock	16	SDA	I/O	I2C data
17	CEC Ground	GND	CEC ground	18	+5V	power supply	5V / 50mA max
19	HPD	input	hot plug detect				

D.4 Audio : JACK connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	GND	GND	ground	2	DATA_L	I/O	left channel
3	DATA_R	I/O	right channel				

D.5 RS232 : DE9 female connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	NC	/	not connected	2	RX	input	received data
3	TX	output	transmitted data	4	NC	/	not connected
5	NC	/	not connected	6	NC	/	not connected
7	NC	/	not connected	8	NC	/	not connected
9	NC	/	not connected				

D.6 RS232 : DE9 male connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	NC	/	not connected	2	TX	output	transmitted data
3	RX	input	received data	4	NC	/	not connected
5	NC	/	not connected	6	NC	/	not connected
7	NC	/	not connected	8	NC	/	not connected
9	NC	/	not connected				

D.7 Ethernet : RJ45 connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	TCT	/	3.3V filtered	2	TD+	output	tx+
3	TD-	output	tx-	4	RD+	input	rx+
5	RD-	input	rx-	6	RCT	/	3.3V filtered

D.8 CAN : RJ45 connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	NC	/	not connected	2	NC	/	not connected
3	NC	/	not connected	4	CAN_H	I/O	CAN high level
5	CAN_L	I/O	CAN low level	6	NC	/	not connected

D.9 USB : connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	VBUS	5.0V	power supply	2	D-	I/O	data-
3	D+	I/O	data+	4	GND	GND	ground

D.10 Memory Card : MCI connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	DA3	I/O	card data 3	2	CDA	I/O	card command
3	GND	GND	ground	4	VCC	3.3V	power supply
5	CK	output	clock	6	GND	GND	ground
7	DA0	I/O	card data 0	8	DA1	I/O	card data 1
9	DA2	I/O	card data 2	10	NC	3.3V, pullup	read only
11	CARD_CD	input, low	card detection				

D.11 Image Sensor, GPIO : 2x13 connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	IPCK / PA0	input / I/O	image data clock	2	IHSYNC / PA1	input / I/O	image horiz. sync.
3	IVSYNC / PA2	input / I/O	image vertical sync.	4	IMCK / PA5	output / I/O	image ref. clock
5	ISD0 / PA6	input / I/O	image data bit 0	6	ISD1 / PD0	input / I/O	image data bit 1
7	ISD2 / PD1	input / I/O	image data bit 2	8	ISD3 / PD2	input / I/O	image data bit 3
9	ISD4 / PD3	input / I/O	image data bit 4	10	GND	GND	ground
11	ISD5 / PD4	input / I/O	image data bit 5	12	GND	GND	ground
13	ISD6 / PD5	input / I/O	image data bit 6	14	GND	GND	ground
15	ISD7 / PD6	input / I/O	image data bit 7	16	GND	GND	ground
17	ISD8 / PD8	input / I/O	image data bit 8	18	GND	GND	ground
19	ISD9 / PD9	input / I/O	image data bit 9	20	GND	GND	ground
21	ISD10 / PD10	input / I/O	image data bit 10	22	GND	GND	ground
23	ISD11 / PD11	input / I/O	image data bit 11	24	GND	GND	ground
25	- / PD12	- / I/O	- / I/O	26	VCC	3.3V	power supply

D.12 IDE : 2x20 connector

Pin #	*Signal*	*Type*	*Description*	*Pin #*	*Signal*	*Type*	*Description*
1	RESET	output, low	reset	2	GND	GND	ground
3	Data7	I/O	data 7	4	Data8	I/O	data 8
5	Data6	I/O	data 6	6	Data9	I/O	data 9
7	Data5	I/O	data 5	8	Data10	I/O	data 10
9	Data4	I/O	data 4	10	Data11	I/O	data 11
11	Data3	I/O	data 3	12	Data12	I/O	data 12
13	Data2	I/O	data 2	14	Data13	I/O	data 13
15	Data1	I/O	data 1	16	Data14	I/O	data 14
17	Data0	I/O	data 0	18	Data15	I/O	data 15
19	GND	GND	ground	20	NC	/	key
21	NC	/	data request	22	GND	GND	ground
23	I/O write	output, low	data write	24	GND	GND	ground
25	I/O read	output, low	data read	26	GND	GND	ground
27	HRDY	input	device ready	28	NC	3.3V, pullup	cable select
29	NC	3.3V, pullup	data acknowledge	30	GND	GND	ground
31	IRQ	input	interrupt request	32	NC	/	not connected
33	ADDR1	output	address 1	34	NC	/	gpio dma
35	ADDR0	output	address 0	36	ADDR2	output	address 2
37	CS1	output, low	chip select 1	38	CS2	output, low	chip select 1
39	Activity	/	nc, (activity)	40	GND	GND	ground

E Procédures de test

E.1 Power supply (only power circuit connected)

#	*Description*	*Expected results*	*Measured results*	*Actions*
	The board is supplied with current limit but no jumpers JP1..4 are not inserted to isolate the rest of the board of a possible error.			
201	Current measurement on J1, power supply connector	Low current [A]	0.025	
202	Voltage control on J4.2	1.08..**1.2**..1.32 [V]	**1.5970**	R6 200 => 100 [Ω]
	New test		1.1980	
203	Voltage control on J3.2	1.71..**1.8**..1.89 [V]	1.7970	
204	Voltage control on J1.2	3.00..**3.3**..3.47 [V]	3.301	
205	Voltage control on J2.2	4.50..**5.0**..5.5 [V]	4.997	
206	Power on Led test	Led shines	yes	
207	Voltage control on Led LD1	Below 2.8 [V]	2.214	

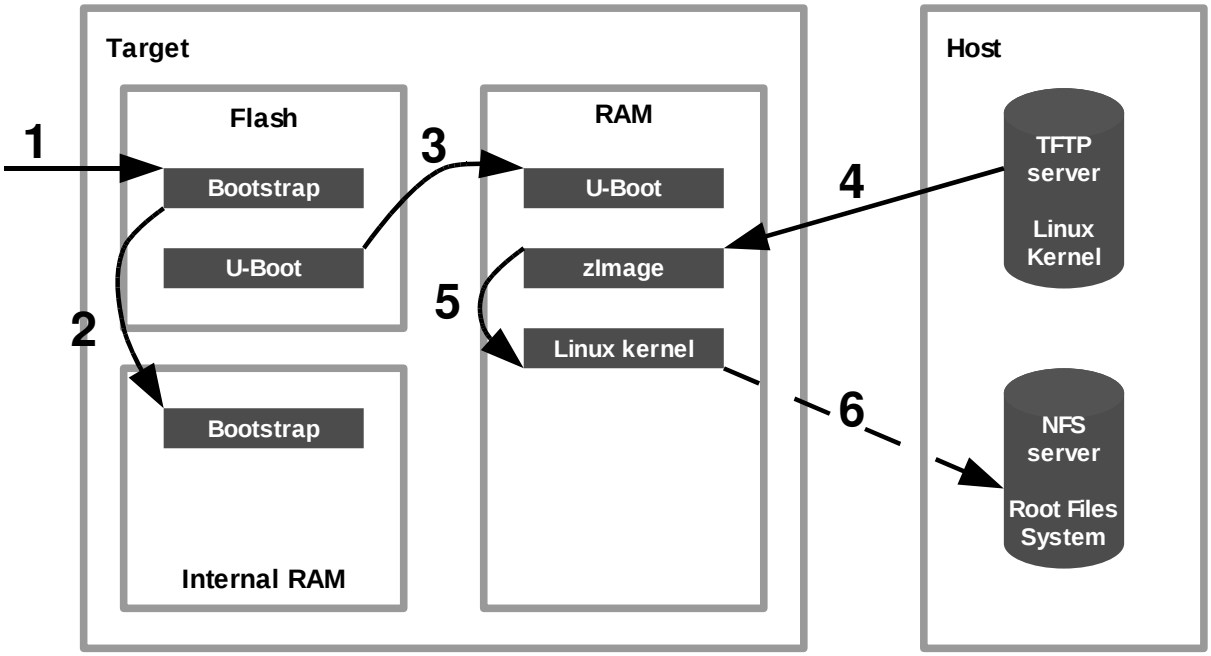
E.2 Power supply (board fully powered)

#	*Description*	*Expected results*	*Measured results*	*Actions*
	The board is supplied with current limitation and jumpers JP1..4 are inserted.			
301	Current measurement on J1, power supply connector	Medium current	0.3120	
302	Voltage control on J4.2	1.08..**1.2**..1.32 [V]	1.192	
303	Voltage control on J3.2	1.71..**1.8**..1.89 [V]	1.800	
304	Voltage control on J1.2	3.00..**3.3**..3.47 [V]	3.303	
305	Voltage control on J2.2	4.50..**5.0**..5.5 [V]	4.995	

306	Power on Led test	Led shines	yes
307	Voltage control on Led LD1	Below 2.8 [V]	1.087

F Séquance de boot

1. When BMS equals zero, the boot sequence starts at the beginning of the Flash. Only the minimal initialization is done and the main clock is configured to the highest possible frequency: 8MHz
2. Due to the AT91SAM9263's bugged Flash controller, the bootstrap code is copied into the internal RAM. After copying a remap is performed and code is now executed from internal RAM. The initialization could be now fully finished.
3. Then the bootstrap application copies U-Boot from the Flash into the RAM and jump to.
4. U-Boot downloads the compressed Linux kernel zImage into the RAM from the TFTP server present on the host machine.
5. After that the kernel unpacks itself in RAM and performs the system initialization.
6. Finally the Linux kernel mounts the root files system through NFS which is also provide by the host machine.



G JTAG test sources

G.1 ledtest.c

```
/*
       dummy app wich make leds blinking
*/

#define LEDS                    0x00f80000
#define PIOB_ENABLE             0xFFFFF400
#define PIOB_OUT_ENABLE         0xFFFFF410
#define PIOB_OUT_SET            0xFFFFF430
#define PIOB_OUT_RESET          0xFFFFF434

int main( void )
{
       int     i ;
       int     *bp ;

       /* init piob to drive leds                                    */
       bp = (int*) PIOB_ENABLE ;
       *bp = LEDS ;

       bp = (int*) PIOB_OUT_ENABLE ;
       *bp = LEDS ;

       bp = (int*) PIOB_OUT_SET ;
       *bp = LEDS ;

       for( ;; )
       {
              for( i = 100000 ; i >= 0 ; --i ) ;

              bp = (int*) PIOB_OUT_RESET ;
              *bp = LEDS ;

              for( i = 100000 ; i >= 0 ; --i ) ;

              bp = (int*) PIOB_OUT_SET ;
              *bp = LEDS ;
       }

       return 0 ;
}
```

G.2 Makefile for ledtest.bin

```
# Makefile/ledtest

CROSS_COMPILE=/opt/eldk/usr/bin/arm-linux-

BIN=ledtest
TARGET=AT91SAM9263
CC=$(CROSS_COMPILE)gcc
LD=$(CROSS_COMPILE)gcc
CCFLAGS=-g -mcpu=arm9 -O0 -Wall -D$(TARGET)

OBJCOPY=$(CROSS_CMPILE)objcopy
LDFLAGS+=-nostartfiles -nostdlib  -Wl,-Map=$(BIN).map,--cref
LDFLAGS+=-T ../elf32-littlearm.lds

SRC+= $(wildcard *.c)
OBJ= $(SRC:.c=.o)

all : $(BIN)
```

```
rebuild : clean all

$(BIN) : $(OBJ)
	$(LD) $(LDFLAGS) -n -o $(BIN).elf $(OBJ)
	$(OBJCOPY) --strip-debug --strip-unneeded $(BIN).elf -O binary $(BIN).bin

%.o: %.c
	@$(CC) -o $@ -c $(CFLAGS) -D$(TARGET) $<

clean :
	@rm -f *.o *.map
```

G.3 BDI2000 configuration file for ledtest.bin

```
; bdiGDB configuration for SAM9EBS board / ledtest.bin

[INIT]
WM32            0xFFFFFD44              0x00008000      ; disable watchdog
WM32            0xFFFFF400              0x00800000      ; config leds' as pio
WM32            0xFFFFF410              0x00800000      ; config leds' pio as output
WM32            0xFFFFE400              0x3F3F3F3F      ; config smc setup time
WM32            0xFFFFE404              0x7F7F7F7F      ; config smc pulse time
WM32            0xFFFFE408              0x01FF01FF      ; config smc cycle time
WM32            0xFFFFF430              0x00800000      ; set on status led


[TARGET]
RESET           NONE                                    ; disable reset
CPUTYPE         ARM9E                                   ; AT91SAM926EJ-S
CLOCK           0                                       ; adaptative
ENDIAN          LITTLE                                  ; little endian
VECTOR          CATCH 0x1f                              ; catch d_abort, p_abort, swi, undef and Reset
PROMPT          sam9ebs>                                ; set the Telnet prompt


[HOST]
IP              153.109.5.183                           ; host ip
FILE            ledtest.bin                             ; file to load in memory
FORMAT          BIN                                     ; format of file to load in memory
LOAD            MANUAL                                  ; load by user


[FLASH]
CHIPTYPE        STRATAX16                               ; intel's 28Fxxx compliant flash
BUSWIDTH        16                                      ; config flash bus width
FILE            ledtest.bin                             ; file to load in flash
FORMAT          BIN                                     ; format of file to load in flash


[REGS]
FILE            reg926e.def                             ; register for AT91SAM926
```

H Bootstrap sources

H.1 bootstrap/**crt0_gnu.S**

```
/*
 crt0_gnu.S - startup code
 based on the AT91Bootstrap framework from Atmel

 this version allows to boot correctly and quite quickly from norflash

 Copyright (c) 2006, Atmel Corporation
 Copyright (c) 2008, Yvan Epiney - http://www.hevs.ch/

 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions are met:

 - Redistributions of source code must retain the above copyright notice,
 this list of conditions and the disclaimer below.

 Atmel's name may not be used to endorse or promote products derived from
 this software without specific prior written permission.

 DISCLAIMER: THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
 OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/


.section start
        .text

        #include "include/part.h"


/* application startup entry point                                          */

        .globl reset
        .align 4
reset:

/* verctors table                                                           */
_exception_vectors:
        b       reset_vector    /* reset                                 */
        b       undef_vector    /* undefined instruction                 */
        b       swi_vector      /* software interrupt                    */
        b       pabt_vector     /* prefetch abort                        */
        b       dabt_vector     /* data abort                            */
.word           _edata          /* size of the image for SAM-BA          */
        b       irq_vector      /* IRQ : read the AIC                    */
        b       fiq_vector      /* FIQ                                   */

undef_vector:
        b       undef_vector
swi_vector:
        b       swi_vector
pabt_vector:
        b       pabt_vector
dabt_vector:
        b       dabt_vector
```

```
rsvd_vector:
        b       rsvd_vector
irq_vector:
        b       irq_vector
fiq_vector:
        b       fiq_vector
reset_vector:

/* init the stack                                                       */
_init_stack:
        ldr     sp, =TOP_OF_MEM

#ifdef CFG_VERBOSE
/*
when testing we drive the GPIO pin #5 as high level during the bootstrap
*/
        /* enable gpio data pin #5                                      */
        ldr     r1, =AT91C_PIOE_PER
        mov     r2, #0x00000001
        str     r2, [r1]

        /* set gpio data pin as outpug #5                               */
        ldr     r1, =AT91C_PIOE_OER
        mov     r2, #0x00000001
        str     r2, [r1]

        /*  drive gpio data pin at 1                                    */
        ldr     r1, =AT91C_PIOE_SODR
        mov     r2, #0x00000001
        str     r2, [r1]

#endif /* CFG_VERBOSE */

#ifdef CFG_NORFLASH
/*
when running from NOR, we must relocate to internal sram prior to
resetting the clocks and SMC timings
*/
_relocate_to_sram:

        /* relocation can be slow, disable the watchdog                 */
        ldr     r1, =AT91C_WDTC_WDMR
        mov     r2, #0x00008000
        str     r2, [r1]

#ifdef CFG_QUICK_START
        /* speed up clock at max possible frequancy before relocate     */
        bl      speed_up_osc

#endif  /* CFG_QUICK_START */

        /* copy all boot app into internal sram                    */
        ldr     r1, =_stext                 /* start of boot code       */
        ldr     r2, =_edata                 /* end of boot code         */
        mov     r4, #0x00300000             /* internal sram address    */

relocate_to_sram_loop_:
        cmp     r1, r2
        ldrne   r3, [r1], #4
        strne   r3, [r4], #4
        bne     relocate_to_sram_loop_

        /* remap memmories                                              */
        ldr     r1, =AT91C_MATRIX_MRCR
        ldr     r2, =0x000001FF
        str     r2, [r1]

        /*
        since here code is executed from internal sram
        */

        ldr     pc, =_setup_clocks

#endif /* CFG_NORFLASH */
```

```
_setup_clocks:
/* test if main oscillator is enabled                                   */
        ldr     r0,=AT91C_PMC_SR
        ldr     r1, [r0]
        ldr     r2,=AT91C_PMC_MOSCS
        ands    r1, r1, r2
        bne     _switch_to_mosc

/* enable the main oscillator                                           */
_enable_mosc:
        ldr     r0,=AT91C_PMC_MOR
        mov     r1, #(0x40 << 8)
        ldr     r2,=AT91C_CKGR_MOSCEN
        orr     r1, r1, r2
        str     r1, [r0]
        ldr     r0,=AT91C_PMC_SR
1:
        ldr     r1, [r0]
        ldr     r2,=AT91C_PMC_MOSCS
        ands    r1, r1, r2
        beq     1b

/* test if MCK == SLOW CLOCK                                            */
_switch_to_mosc:
        ldr     r0,=AT91C_PMC_MCKR
        ldr     r1,=AT91C_PMC_CSS
        ldr     r2, [r0]
        and     r2, r2, r1
        mov     r1, #0
        cmp     r1, r2
/* no => do nothing                                                     */
        bne     _init_bss
/* yes => switch to the main oscillator                                 */
        ldr     r1,=AT91C_PMC_CSS_MAIN_CLK
        ldr     r2,=AT91C_PMC_PRES_CLK
        orr     r1, r1, r2
        str     r1, [r0]
        ldr     r0,=AT91C_PMC_SR
1:
        ldr     r1, [r0]
        ldr     r2,=AT91C_PMC_MCKRDY
        ands    r1, r1, r2
        beq     1b

/* copy the data section in RAM at .data link address                   */
_init_data:
        ldr     r2, =_lp_data
        ldmia   r2, {r1, r3, r4}
1:
        cmp     r3, r4
        ldrcc   r2, [r1], #4
        strcc   r2, [r3], #4
        bcc     1b

/* initialize the bss segment                                           */
_init_bss:
        adr     r2, _lp_bss
        ldmia   r2, {r3, r4}
        mov     r2, #0
1:
        cmp     r3, r4
        strcc   r2, [r3], #4
        bcc     1b

/* branch on C code Main function (with interworking)                   */
_branch_main:

        ldr     r4, = main
        mov     lr, pc
        bx      r4

/* branch to the application at the end of the bootstrap init           */
_go:
```

```
        ldr     r1, =MACH_TYPE
        mov     lr, pc
        bx      r0

/* increase clock before code relocation                                */
speed_up_osc:
        /* setup main osc                                               */
        ldr     r1, =AT91C_PMC_MOR
        ldr     r2, =CFG_QUICK_START_OSC
        str     r2, [r1]

        /* wait until osc is stabilized: 1 == PMC_SR.0                  */
speed_up_osc_stabilize_loop_:
        ldr     r1, =AT91C_PMC_SR
        ldr     r2, [r1]
        and     r2, r2, #0x00000001
        cmp     r2, #0x00000001
        bne     speed_up_osc_stabilize_loop_

        /* switch clock from slow clock to main                         */
        ldr     r1, =AT91C_PMC_MCKR
        mov     r2, #0x00000005
        str     r2, [r1]

        mov     pc, lr                /* return */


        .align
_lp_data:
        .word _etext
        .word _sdata
        .word _edata

_lp_bss:
        .word _sbss
        .word _ebss
```

H.2 bootstrap/board/sam9ebs/nor/**sam9ebs.h**

```
/*
 sam9ebs.h - definition file of bootstraping app
 based on the AT91Bootstrap framework from Atmel

 Copyright (c) 2006, Atmel Corporation
 Copyright (c) 2008, Yvan Epiney - http://www.hevs.ch/

 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions are met:

 - Redistributions of source code must retain the above copyright notice,
 this list of conditions and the disclaimer below.

 Atmel's name may not be used to endorse or promote products derived from
 this software without specific prior written permission.

 DISCLAIMER: THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
 OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/


#ifndef SAM9EBS_H
#define SAM9EBS_H


/*
        application settings
*/

/* debug definition                                                     */
#define CFG_VERBOSE

/* init standard hardware                                               */
#define CFG_SDRAM
#define CFG_HW_INIT

/* use norflash memory to load u-boot                                   */
#undef CFG_DATAFLASH
#undef CFG_NANDFLASH
#define CFG_NORFLASH


/*
        norflash to internal sram relocation optimisation
*/

/* enable quick start, configure oscillator before copy app into ram    */
#define CFG_QUICK_START
/* use main oscillator with a prescaller of 2: 16MHz / 2 => 8MHz        */
#define CFG_QUICK_START_OSC   0x00000801


/*
        bootstraping settings
*/

/* u-boot address in flash without flash offset                         */
#define IMG_ADDRESS           0x00010000
```

```
/* u-boot size in Flash                                                 */
#define IMG_SIZE              0x00200000            /* u-boot.bin       */

/* u-boot destination address in ram                                    */
#define JUMP_ADDR             0x23F00000            /* RAM              */

/* TODO define own type                                                 */
#define MACH_TYPE             1202                  /* AT91SAM9263-EK   */


/*
        clock & timing settings
*/

/* DIVA = 1, MULA = 15 + 1, OUTA = 01 ; 16*(14+1(implicit))/1=240 MHz   */
#define PLLA_SETTINGS         0x200E7F01

/* DIVB = 1, MULB = 3 + 1, OUTB = 00, USBDIV = 00 ; 16*(3+1)/1=48 MHz   */
#define PLLB_SETTINGS         0x00033F01

/* MCK = PLLA / 2 = 240MHz / 2                                          */
#define MASTER_CLOCK          ( 240000000 / 2 )

/* switch MCK on PLLA output PCK = PLLA = 2 * MCK                       */
#define MCKR_SETTINGS         ( AT91C_PMC_CSS_PLLA_CLK | \
                              AT91C_PMC_PRES_CLK |  AT91C_PMC_MDIV_2 )

/* timeout value                                                        */
#define PLL_LOCK_TIMEOUT      1000000


/*
        ram basic settings
*/

#define AT91C_EBI_SDRAM       AT91C_EBI0_SDRAM
#define AT91C_BASE_SDRAMC     AT91C_BASE_SDRAMC0

/* NOTE timing settings are located in bootstrap/board/sam9ebs.c        */


/*
        norflash timing settings
*/

/* NOTE this settings are located in bootstrap/include/norflash.h       */


#endif /* SAM9EBS_H */
```

H.3 bootstrap/include/**norflash.h**

```
/* ----------------------------------------------------------------------------
 *         ATMEL Microcontroller Software Support  -  ROUSSET  -
 * ----------------------------------------------------------------------------
 * Copyright (c) 2007, Stelian Pop <stelian.pop@leadtechdesign.com>
 * Copyright (c) 2007 Lead Tech Design <www.leadtechdesign.com>
 * Copyright (c) 2008, Yvan Epiney - http://www.hevs.ch/
 *
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the disclaiimer below.
 *
 * Atmel's name may not be used to endorse or promote products derived from
 * this software without specific prior written permission.
 *
 * DISCLAIMER: THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 * DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
 * OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ----------------------------------------------------------------------------
 * File Name           : norflash.h
 * Object              : ATMEL Norflash Header File
 * Creation            :
 *----------------------------------------------------------------------------
*/
#ifndef _NORFLASH_H
#define _NORFLASH_H

#define AT91_NORFLASH_BASE          AT91C_EBI0_CS0

/*
       15 * ( 1 / 120e6 ) >= 120 ns
*/
#define AT91C_FLASH_NWE_SETUP       (15 << 0)
#define AT91C_FLASH_NCS_WR_SETUP    (15 << 8)
#define AT91C_FLASH_NRD_SETUP       (15 << 16)
#define AT91C_FLASH_NCS_RD_SETUP    (15 << 24)

#define AT91C_FLASH_NWE_PULSE       (15 << 0)
#define AT91C_FLASH_NCS_WR_PULSE    (15 << 8)
#define AT91C_FLASH_NRD_PULSE       (15 << 16)
#define AT91C_FLASH_NCS_RD_PULSE    (15 << 24)

/*
       15 * ( 1 / 120e6 ) >= 360 ns
*/
#define AT91C_FLASH_NWE_CYCLE       (45 << 0)
#define AT91C_FLASH_NRD_CYCLE       (45 << 16)

void norflash_hw_init(void);
int load_norflash(unsigned int img_addr, unsigned int img_size, unsigned int img_dest);

#endif
```

H.4 bootstrap/**main.c**

```
/*
 main.h - core function of bootstraping app
 based on the AT91Bootstrap framework from Atmel

 Copyright (c) 2006, Atmel Corporation
 Copyright (c) 2008, Yvan Epiney - http://www.hevs.ch/

 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions are met:

 - Redistributions of source code must retain the above copyright notice,
 this list of conditions and the disclaimer below.

 Atmel's name may not be used to endorse or promote products derived from
 this software without specific prior written permission.

 DISCLAIMER: THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
 OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/


#include "include/part.h"
#include "include/main.h"
#include "include/debug.h"
#include "include/dataflash.h"
#include "include/nandflash.h"
#include "include/norflash.h"


int main( void )
{
#ifdef CFG_VERBOSE
        int     *bp ;
#endif

/*
        1st step: hardware initialization
*/
#ifdef CFG_HW_INIT
        hw_init() ;
#endif


/*
        2nd step: load bootloader from media
*/
/* load from Dataflash in RAM                                           */
#ifdef CFG_DATAFLASH
        load_df( AT91C_SPI_PCS_DATAFLASH, IMG_ADDRESS, IMG_SIZE, JUMP_ADDR ) ;
#endif

/* load from Nandflash in RAM                                           */
#ifdef CFG_NANDFLASH
        load_nandflash( IMG_ADDRESS, IMG_SIZE, JUMP_ADDR ) ;
#endif

/* load from Norflash in RAM                                            */
#ifdef CFG_NORFLASH
        load_norflash( IMG_ADDRESS, IMG_SIZE, JUMP_ADDR ) ;
#endif
```

```
#ifdef CFG_VERBOSE
        /* clear GPIO pin #5, use for timestamp test                    */
        bp = (int*) AT91C_PIOE_CODR ;
        *bp = 0x00000001 ;

#endif /* CFG_VERBOSE */


/*
        3rd step: branch to the image address
*/
        return JUMP_ADDR ;
}
```

H.5 bootstrap/board/sam9ebs/**sam9ebs.c**

```
/*
 sam9ebs - functions collection of bootstraping app
 based on the AT91Bootstrap framework from Atmel

 Copyright (c) 2006, Atmel Corporation
 Copyright (c) 2008, Yvan Epiney - http://www.hevs.ch/

 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions are met:

 - Redistributions of source code must retain the above copyright notice,
 this list of conditions and the disclaimer below.

 Atmel's name may not be used to endorse or promote products derived from
 this software without specific prior written permission.

 DISCLAIMER: THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
 OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/


#include "../../include/part.h"
#include "../../include/gpio.h"
#include "../../include/pmc.h"
#include "../../include/debug.h"
#include "../../include/sdramc.h"
#include "../../include/main.h"

#ifdef CFG_NANDFLASH
#include "../../include/nandflash.h"
#endif
#ifdef CFG_NORFLASH
#include "../../include/norflash.h"
#endif
#ifdef CFG_DATAFLASH
#include "../../include/dataflash.h"
#endif

#ifdef CFG_VERBOSE

#define PIO_LED_0            AT91C_PIN_PB( 19 )
#define PIO_LED_1            AT91C_PIN_PB( 20 )
#define PIO_LED_2            AT91C_PIN_PB( 21 )
#define PIO_LED_3            AT91C_PIN_PB( 22 )
#define PIO_LED_STATUS       AT91C_PIN_PB( 23 )

#endif  /* CFG_VERBOSE */


static inline unsigned int get_cp15( void )
{
        unsigned int value ;
        __asm__( "mrc p15, 0, %0, c1, c0, 0" : "=r" ( value ) ) ;
        return value ;
}

static inline void set_cp15( unsigned int value )
{
        __asm__( "mcr p15, 0, %0, c1, c0, 0" : : "r" ( value) ) ;
}
```

```
#ifdef CFG_HW_INIT
/*
This function performs very low level HW initialization, it is invoked as
soon as possible during the c_startup
NOTE the bss segment must be initialized
*/
void hw_init( void )
{
        const struct pio_desc hw_pio[] =
        {
#ifdef CFG_VERBOSE
                { "LED_0",     PIO_LED_0,     0, PIO_DEFAULT,       PIO_OUTPUT },
                { "LED_1",     PIO_LED_1,     0, PIO_DEFAULT,       PIO_OUTPUT },
                { "LED_2",     PIO_LED_2,     0, PIO_DEFAULT,       PIO_OUTPUT },
                { "LED_3",     PIO_LED_3,     0, PIO_DEFAULT,       PIO_OUTPUT },
                { "LED_STATUS", PIO_LED_STATUS, 0, PIO_DEFAULT,     PIO_OUTPUT },

#endif  /* CFG_VERBOSE */
                { (char *) 0, 0, 0, PIO_DEFAULT, PIO_PERIPH_A },
        } ;

        /* disable watchdog                                                 */
        writel( AT91C_WDTC_WDDIS, AT91C_BASE_WDTC + WDTC_WDMR ) ;

        /* At this stage the main oscillator is supposed to be enabled      */

        /* configure PLLA = MOSC * (PLL_MULA + 1) / PLL_DIVA                */
        pmc_cfg_plla( PLLA_SETTINGS, PLL_LOCK_TIMEOUT ) ;

        /* switch MCK on PLLA output PCK = PLLA = 2 * MCK                   */
        pmc_cfg_mck( MCKR_SETTINGS, PLL_LOCK_TIMEOUT ) ;

        /* configure PLLB                                                   */
        pmc_cfg_pllb( PLLB_SETTINGS, PLL_LOCK_TIMEOUT ) ;

        /* configure the PIO controller to output PCK0                      */
        pio_setup( hw_pio ) ;

        /* configure the EBI0 Slave Slot Cycle to 64                        */
        writel
        (
                ( readl( ( AT91C_BASE_MATRIX + MATRIX_SCFG4 ) ) & ~0xFF ) |
                0x40, ( AT91C_BASE_MATRIX + MATRIX_SCFG4 )
        ) ;

#ifdef CFG_VERBOSE
        pio_set_value( PIO_LED_STATUS, 1 ) ;

#endif  /* CFG_VERBOSE */

#ifdef CFG_SDRAM
        /*
        matrix initialization
        RAM is 3.3V powered, VDDIOMSEL = 1
        EBI0CS1 is RAM assigned
        */
        writel
        (
                readl( AT91C_BASE_CCFG + CCFG_EBI0CSA )     |
                (1 << 16)                                   |
                AT91C_EBI_CS1A_SDRAMC,
                AT91C_BASE_CCFG + CCFG_EBI0CSA
        ) ;

        /* configure SDRAM controller                                       */
        sdram_init
        (
                AT91C_SDRAMC_NC_10          | /* column addr A0-9          */
                AT91C_SDRAMC_NR_13          | /* row addr A0-12            */
                AT91C_SDRAMC_CAS_3          | /* RAM has 2 & 3 cas         */
                AT91C_SDRAMC_NB_4_BANKS     | /* RAM has 4 banks           */
                AT91C_SDRAMC_DBW_32_BITS    | /* bus 32 bits width         */
                AT91C_SDRAMC_TWR_15         | /* FIXME                     */
```

```
                AT91C_SDRAMC_TRC_8          | /* row cycle 65ns          */
                AT91C_SDRAMC_TRP_3          | /* row precharge 20ns       */
                AT91C_SDRAMC_TRCD_3         | /* row 2 column 20ns        */
                AT91C_SDRAMC_TRAS_6         | /* row active 45ns          */
                AT91C_SDRAMC_TXSR_15,         /* FIXME                     */
                (MASTER_CLOCK * 11)/1000000,  /* refresh timer <64ms       */
                AT91C_SDRAMC_MD_SDRAM         /* no low power              */
        ) ;
#endif  /* CFG_SDRAM */
}
#endif  /* CFG_HW_INIT */


#ifdef CFG_SDRAM
/*
this function performs SDRAMC HW initialization
*/
void sdramc_hw_init( void )
{
        /* RAM PIOs pinning definition                                      */
        const struct pio_desc sdramc_pio[] =
        {
                { "D16", AT91C_PIN_PD( 16 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D17", AT91C_PIN_PD( 17 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D18", AT91C_PIN_PD( 18 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D19", AT91C_PIN_PD( 19 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D20", AT91C_PIN_PD( 20 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D21", AT91C_PIN_PD( 21 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D22", AT91C_PIN_PD( 22 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D23", AT91C_PIN_PD( 23 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D24", AT91C_PIN_PD( 24 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D25", AT91C_PIN_PD( 25 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D26", AT91C_PIN_PD( 26 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D27", AT91C_PIN_PD( 27 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D28", AT91C_PIN_PD( 28 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D29", AT91C_PIN_PD( 29 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D30", AT91C_PIN_PD( 30 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { "D31", AT91C_PIN_PD( 31 ), 0, PIO_DEFAULT, PIO_PERIPH_A },
                { (char *) 0, 0, 0, PIO_DEFAULT, PIO_PERIPH_A },
        } ;

        /* configure SDRAMC PIO                                             */
        pio_setup( sdramc_pio ) ;
}
#endif /* CFG_SDRAM */

#ifdef CFG_DATAFLASH
/*
this function performs DataFlash HW initialization
*/
void df_hw_init( void )
{
        /* configure PIOs                                                   */
        const struct pio_desc df_pio[] =
        {
                { "MISO",  AT91C_PIN_PA( 0 ), 0, PIO_DEFAULT, PIO_PERIPH_B },
                { "MOSI",  AT91C_PIN_PA( 1 ), 0, PIO_DEFAULT, PIO_PERIPH_B },
                { "SPCK",  AT91C_PIN_PA( 2 ), 0, PIO_DEFAULT, PIO_PERIPH_B },
                { "NPCS0", AT91C_PIN_PA( 5 ), 0, PIO_DEFAULT, PIO_PERIPH_B },
                { (char *) 0, 0, 0, PIO_DEFAULT, PIO_PERIPH_A },
        } ;

        /* Configure the PIO controller                                     */
        pio_setup( df_pio ) ;
}
#endif /* CFG_DATAFLASH */



#ifdef CFG_NANDFLASH
/*
this function performs NandFlash HW initialization
*/
```

```
void nandflash_hw_init( void )
{
	/* configure PIOs                                                */
	const struct pio_desc nand_pio[] =
	{
		{ "RDY_BSY", AT91C_PIN_PA( 22 ), 0, PIO_PULLUP, PIO_INPUT },
		{ "NANDCS",  AT91C_PIN_PD( 15 ), 0, PIO_PULLUP, PIO_OUTPUT },
		{ (char *) 0, 0, 0,  PIO_DEFAULT, PIO_PERIPH_A },
	};

	/*
	setup smart media, first enable the address range of CS3 in HMATRIX
	user interface
	*/
	writel
	(
		readl( AT91C_BASE_CCFG + CCFG_EBI0CSA )     |
		AT91C_EBI_CS3A_SM,
		AT91C_BASE_CCFG + CCFG_EBI0CSA
	) ;

	/* configure SMC CS3                                             */
	writel
	(
		AT91C_SM_NWE_SETUP                  |
		AT91C_SM_NCS_WR_SETUP               |
		AT91C_SM_NRD_SETUP                  |
		AT91C_SM_NCS_RD_SETUP,
		AT91C_BASE_SMC0 + SMC_SETUP3
	) ;

	writel
	(
		AT91C_SM_NWE_PULSE                  |
		AT91C_SM_NCS_WR_PULSE               |
		AT91C_SM_NRD_PULSE                  |
		AT91C_SM_NCS_RD_PULSE,
		AT91C_BASE_SMC0 + SMC_PULSE3
	) ;

	writel
	(
		AT91C_SM_NWE_CYCLE                  |
		AT91C_SM_NRD_CYCLE,
		AT91C_BASE_SMC0 + SMC_CYCLE3
	) ;

	writel
	(
		AT91C_SMC_READMODE                  |
		AT91C_SMC_WRITEMODE                 |
		AT91C_SMC_NWAITM_NWAIT_DISABLE      |
		AT91C_SMC_DBW_WIDTH_SIXTEEN_BITS    |
		AT91C_SM_TDF,
		AT91C_BASE_SMC0 + SMC_CTRL3
	) ;

	/* configure the PIO controller                                  */
	writel( ( 1 << AT91C_ID_PIOA ), PMC_PCER + AT91C_BASE_PMC ) ;
	writel( ( 1 << AT91C_ID_PIOCDE ), PMC_PCER + AT91C_BASE_PMC ) ;

	pio_setup(nand_pio);
}



/*
configure SMC in 16 bits mode
*/
void nandflash_cfg_16bits_dbw_init( void )
{
	writel
	(
		readl( AT91C_BASE_SMC0 + SMC_CTRL3 ) |
```

```
                AT91C_SMC_DBW_WIDTH_SIXTEEN_BITS,
                AT91C_BASE_SMC0 + SMC_CTRL3 ) ;
}

/*
configure SMC in 8 bits mode
*/
void nandflash_cfg_8bits_dbw_init( void )
{
        writel
        (
                ( readl( AT91C_BASE_SMC0 + SMC_CTRL3 ) & ~( AT91C_SMC_DBW ) ) |
                AT91C_SMC_DBW_WIDTH_EIGTH_BITS,
                AT91C_BASE_SMC0 + SMC_CTRL3
        ) ;
}

#endif /* CFG_NANDFLASH */


#ifdef CFG_NORFLASH
/*
this function performs NorFlash HW initialization
*/
void norflash_hw_init( void )
{
        /* setup configure                                              */
        writel
        (
                AT91C_FLASH_NWE_SETUP               |
                AT91C_FLASH_NCS_WR_SETUP            |
                AT91C_FLASH_NRD_SETUP               |
                AT91C_FLASH_NCS_RD_SETUP,
                AT91C_BASE_SMC0 + SMC_SETUP0
        ) ;

        /* pulse configure                                              */
        writel
        (
                AT91C_FLASH_NWE_PULSE               |
                AT91C_FLASH_NCS_WR_PULSE            |
                AT91C_FLASH_NRD_PULSE               |
                AT91C_FLASH_NCS_RD_PULSE,
                AT91C_BASE_SMC0 + SMC_PULSE0
        ) ;

        /* cycle configure                                              */
        writel
        (
                AT91C_FLASH_NWE_CYCLE               |
                AT91C_FLASH_NRD_CYCLE,
                AT91C_BASE_SMC0 + SMC_CYCLE0
        ) ;

        /* mode configure                                               */
        writel
        (
                AT91C_SMC_READMODE                  |
                AT91C_SMC_WRITEMODE                 |
                AT91C_SMC_NWAITM_NWAIT_DISABLE      |
                AT91C_SMC_BAT_BYTE_WRITE            |
                AT91C_SMC_DBW_WIDTH_SIXTEEN_BITS    |
                ( AT91C_SMC_TDF & ( 1 << 16 ) ),
                AT91C_BASE_SMC0 + SMC_CTRL0
        ) ;
}

#endif /* CFG_NORFLASH */
```

I U-Boot sources

I.1 u-boot/include/configs/**sam9ebs.h**

```
/*
 * (C) Copyright 2007-2008
 * Stelian Pop <stelian.pop@leadtechdesign.com>
 * Lead Tech Design <www.leadtechdesign.com>
 *
 * (C) Copyright 2008, Yvan Epiney
 * hes-so//Valais http://www.hevs.ch/
 *
 * Configuation settings for the SAM9EBS hes-so//Valais school board.
 *
 * See file CREDITS for list of people who contributed to this
 * project.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA 02111-1307 USA
 */

#ifndef __CONFIG_H
#define __CONFIG_H

/*
       do not perform low level init and do not relocate u-boot into RAM,
       this is already done by AT91bootstrap
*/
#define CONFIG_SKIP_LOWLEVEL_INIT
#define CONFIG_SKIP_RELOCATE_UBOOT

/*
       cpu config
       defines wich chipset is used
*/
#define CONFIG_ARM926EJS                    1
#define CONFIG_AT91SAM9263                  1
#define CONFIG_SAM9EBS                      1
#undef CONFIG_USE_IRQ

/*
       clocks config
       NOTE although that clocks are already define into the bootstrap app,
       u-boot need that this constant are defined to compile and work well.
       for value signification see bootstrap code
*/
#define AT91_SLOW_CLOCK                     32768
#define AT91_MAIN_CLOCK                     240000000
#define AT91_MASTER_CLOCK                   120000000
#define CFG_HZ                              1000000

#define PLLAR_VAL                           0x200F7F01
#define PLLBR_VAL                           0x00033F01
#define MCKR_VAL                            0x00000102

#define CFG_USE_MAIN_OSCILLATOR             1
```

```
/*
       u-boot monitor config
*/
#define CFG_MONITOR_BASE                    0x10010000
#define CFG_MONITOR_LEN                     0x30000
#define CFG_ENV_IS_IN_FLASH                 1
#define CFG_ENV_ADDR                        ( CFG_MONITOR_BASE + CFG_MONITOR_LEN )
#define CFG_ENV_SECT_SIZE                   0x20000


/*
       u-boot menu config
       defines command line environnement
*/
#define CFG_CONSOLE_IS_IN_ENV               1
#define CONFIG_BOOTDELAY                    3
#define CFG_PROMPT                          "U-Boot> "
#define CFG_CBSIZE                          256
#define CFG_MAXARGS                         16
#define CFG_PBSIZE                          ( CFG_CBSIZE + sizeof(CFG_PROMPT) + 16 )

/*
       tools
       defines default command line tool + some usefull dhcp, usb and
       diagnostic tools
*/
#include <config_cmd_default.h>

#define CONFIG_CMDLINE_TAG                  1
#define CONFIG_SETUP_MEMORY_TAGS            1
#define CONFIG_INITRD_TAG                   1

#define CONFIG_CMD_PING                     1
#define CONFIG_CMD_DHCP                     1
#define CONFIG_CMD_DIAG                     1
#define CONFIG_CMD_USB                      1

/*
       misc mem config
       defines where kernel will be load and memory emplaement and size
*/
#define CFG_LOAD_ADDR                       0x24000000
#define CFG_MEMTEST_START                   PHYS_SDRAM
#define CFG_MEMTEST_END                     ( CFG_MEMTEST_START + PHYS_SDRAM_SIZE \
                                             - 262144 )

/*
       flash config
       defines flash address and size
*/
#define CFG_FLASH_CFI                       1
#define CONFIG_FLASH_CFI_DRIVER             1
#define PHYS_FLASH_1                        0x10000000
#define PHYS_FLASH_SIZE                     0x00200000
#define CFG_FLASH_BASE                      PHYS_FLASH_1
#define CFG_MAX_FLASH_BANKS                 1
#define CFG_MAX_FLASH_SECT                  256
#define CFG_FLASH_ERASE_TOUT                ( 5 * CFG_HZ / 1000 )
#define CFG_FLASH_WRITE_TOUT                ( 5 * CFG_HZ / 1000 )

/*
       sdram config
       RAM address start @ 0x20000000 and has 128MB (0x08000000)
*/
#define CONFIG_NR_DRAM_BANKS                1
#define PHYS_SDRAM                          0x20000000
#define PHYS_SDRAM_SIZE                     0x08000000

/*
       bootp config

*/
#define CONFIG_BOOTP_BOOTFILESIZE    1
```

```
#define CONFIG_BOOTP_BOOTPATH               1
#define CONFIG_BOOTP_GATEWAY                1
#define CONFIG_BOOTP_HOSTNAME               1

/*
       RS232 config
       use usart0 @ 115200 bauds as std u-boot output
*/
#define CONFIG_ATMEL_USART                  1
#define CONFIG_USART0                       1
#undef CONFIG_USART1
#undef CONFIG_USART2
#undef CONFIG_USART3
#define CONFIG_BAUDRATE                     115200
#define CFG_BAUDRATE_TABLE                  { 115200 , 19200, 38400, 57600, 9600 }

/*
       ethernet config
       use ethernet in rmii mode
*/
#define CONFIG_MACB                         1
#define CONFIG_RMII                         1
#define CONFIG_NET_MULTI                    1
#define CONFIG_NET_RETRY_COUNT              5
#define CONFIG_RESET_PHY_R                  1
#define CONFIG_ETHADDR                      00:00:17:FF:00:00

/*
       usb config
       defines usb config
*/
#define CONFIG_USB_OHCI_NEW                 1
#define LITTLEENDIAN                        1
#define CONFIG_DOS_PARTITION                1
#define CFG_USB_OHCI_CPU_INIT               1
#define CFG_USB_OHCI_REGS_BASE              0x00a00000
#define CFG_USB_OHCI_SLOT_NAME              "at91sam9263"
#define CFG_USB_OHCI_MAX_ROOT_PORTS         2
#define CONFIG_USB_STORAGE                  1

/*
       spi & dataflash config
       unused but necessary for compilation
*/
#define CFG_SPI_WRITE_TOUT                  (5*CFG_HZ)
#define CFG_MAX_DATAFLASH_BANKS             1
#define CFG_DATAFLASH_LOGIC_ADDR_CS0        0xC0000000
#define AT91_SPI_CLK                        15000000
#define DATAFLASH_TCSS                      (0x1a << 16)
#define DATAFLASH_TCHS                      (0x1 << 24)

/*
       malloc() pool config
*/
#define ROUND(A, B)                         (((A) + (B)) & ~((B) - 1))
#define CFG_MALLOC_LEN                      ( 2*1024*1024 )
#define CONFIG_STACKSIZE                    ( 2*1024*1024 )
#define CFG_GBL_DATA_SIZE                   128

/*
       checks define
*/
#ifdef CONFIG_USE_IRQ
#error CONFIG_USE_IRQ unsupported !!!
#endif

#endif /* __CONFIG_H */
```

J Schématique de la carte

INDEX OF PAGES FOR THE SAM9 MOTHERBOARD

NUMBER	NAME	DESCRIPTION	COMMENTS
1	Index	Description of sheets	Current page
2	Revision	Description of revision changes	Current revision and major changes mades
3	Power supply	Power supply parts	Contains power parts for 5V / 3.3V / 1.8V and 1.2V
4	JTAG / Reset	JTAG and reset parts	Contains JTAG, reset button and controller parts
5	uC 1	First part of SAM9	Contains power, oscillators and RTC
6	uC 2	Second part of SAM9	Contains bus 0 & 1, port A
7	uC 3	Third part of SAM9	Contains port B, C, D & E
9	Video	Video transmitter	Contains video transmitter, HDMI & LCD connectors
10	Audio	Audio uC	Contains audio uC, JACK in & out connectors
11	IDE / Memory Card	IDE & SD/MMC card	Contains IDE exernal HD & SD/MMC card connectors
12	Ethernet	Ethernet PHYSICAL	Contains physical ethernet transceiver & connector
13	USB / CAN	USB & CAN ports	Contains 2 USB host, 1 USB device & 1 CAN
14	RS232 / GPIO / JTAG	Serial & Parallel ports + JTAG	Contains RS232, GPIO & JTAG connectors
15	Basic IOs	User I/O (LEDs, buttons)	Contains 5 LEDs & 4 buttons
16	Memory 1	SDRAM & Flash Memory	Contains the primary RAM (128MB) & Flash (16MB)
17	Memory 2	PSRAM	Contains the video RAM (4MB)

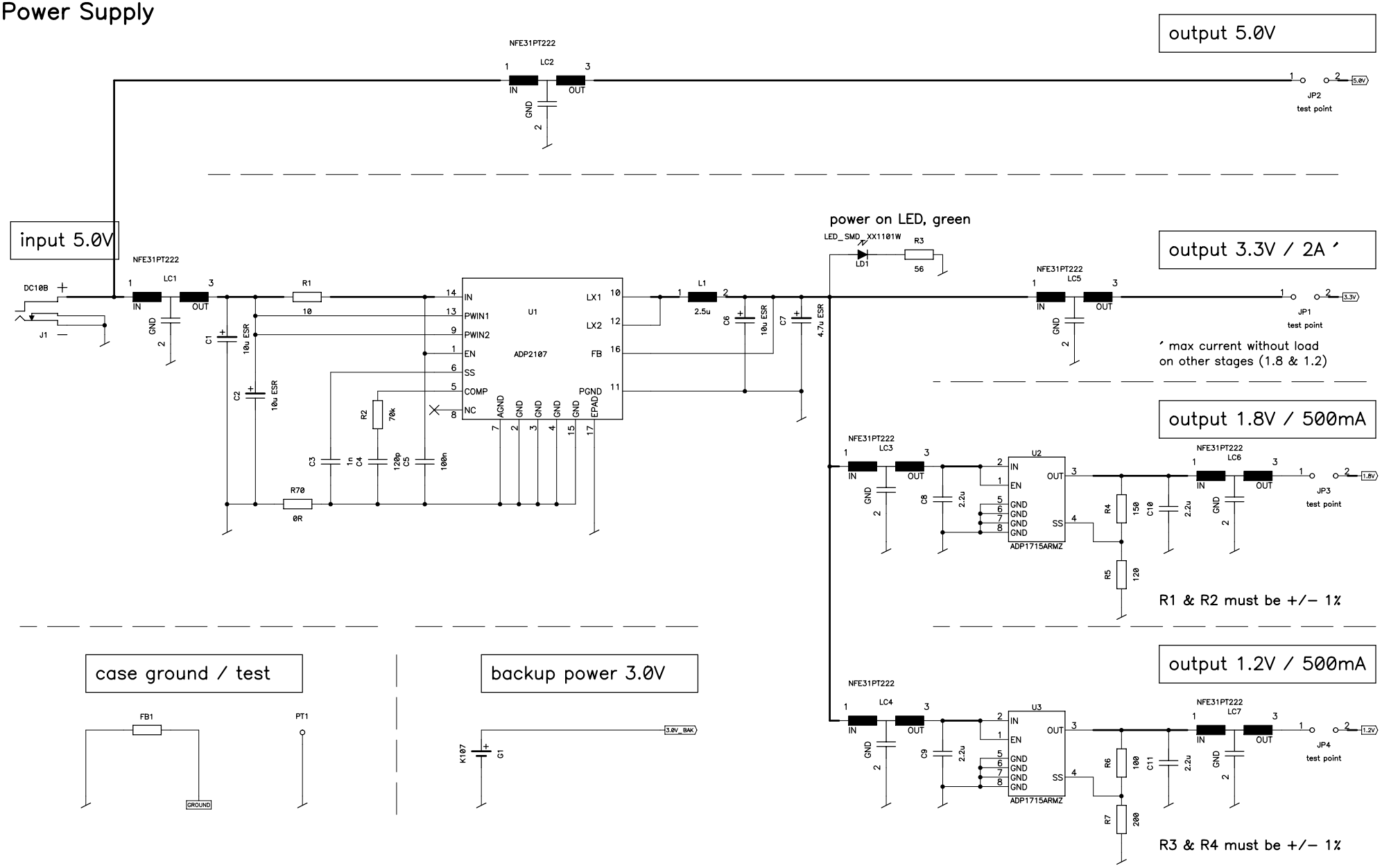
SAM9 motherboard Index	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/16	./sam9/schematic/ SAM9_ motherboard_ bis.sch

REVISION HISTORY FOR THE SAM9 MOTHERBOARD

Revision	Major changes	Path
1.0	Prototype board	SAM9_moterboard.sch

SAM9 motherboard	DES	31.05.2008 Epiney Yvan
Revision	REV	V1.0
HAUTE ECOLE VALAISANNE	2/16	./sam9/schematic/ SAM9_motherboard_bis.sch

Power Supply

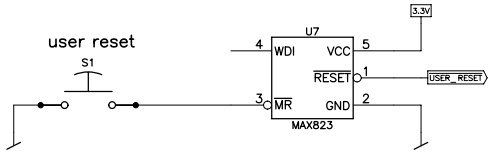


IMPORTANT :
this design has to be implement in accordance
of all the Analog Devices componants layout advices

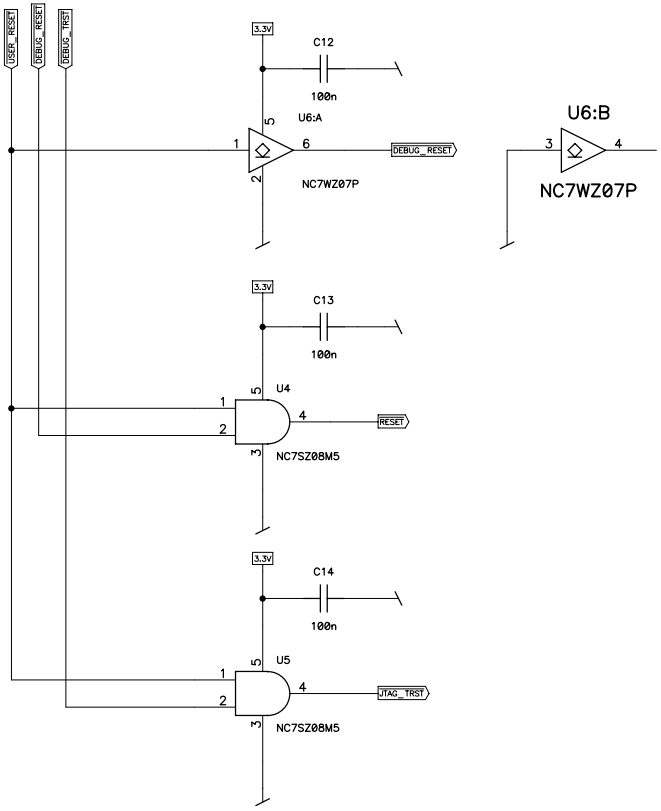
SAM9 motherboard Power Supply	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	3/16	./sam9/schematic/ SAM9_motherboard_bis.sch

JTAG / Reset

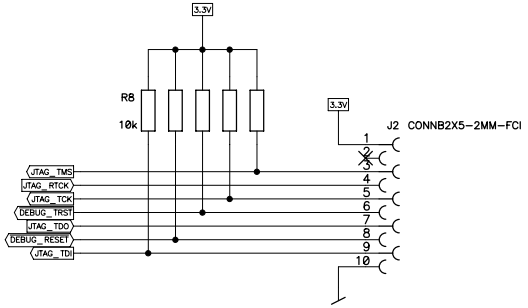
JTAG / Reset



NOTE :
MAX823 is not decoupled by a capacitor to detect small power fault



JTAG connector

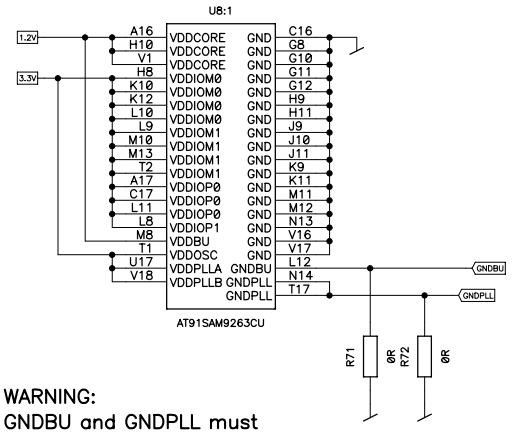


NOTE :
pullup resistors on JTAG TCK, TDI and TMS signals will not be populate

SAM9 motherboard JTAG / Reset	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	4/16	./sam9/schematic/ SAM9_motherboard_bis.sch

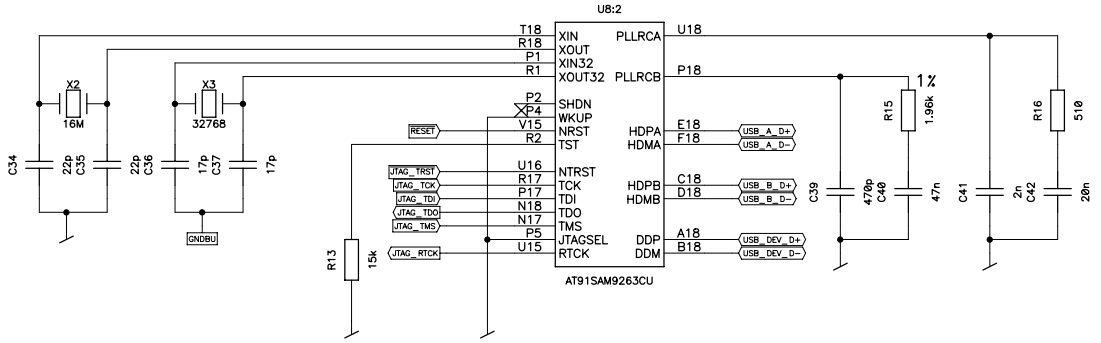
uController 1

Power Supply

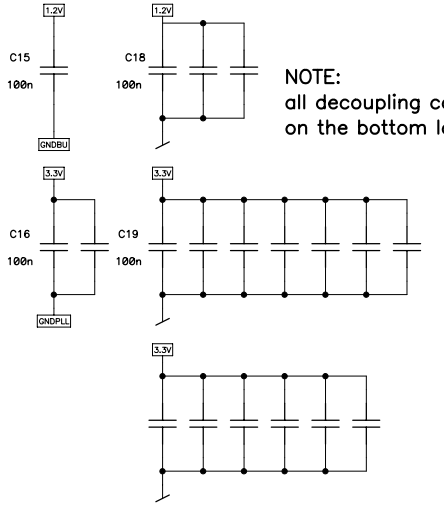


WARNING:
GNDBU and GNDPLL must
be separated from GND

System Ctrl / USB

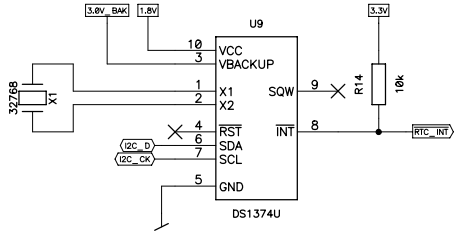


decoupling caps

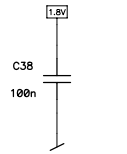


NOTE:
all decoupling caps are
on the bottom layer

Real Time Clock



decoupling cap

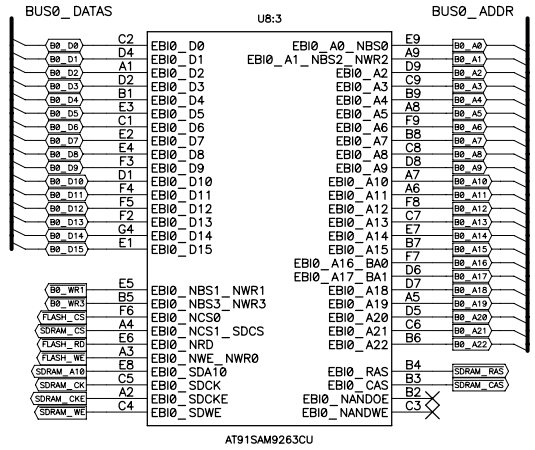


NOTE:
$\overline{\text{RST}}$ has an internal pullup for $\overline{\text{RST}}$,
SQW is not used

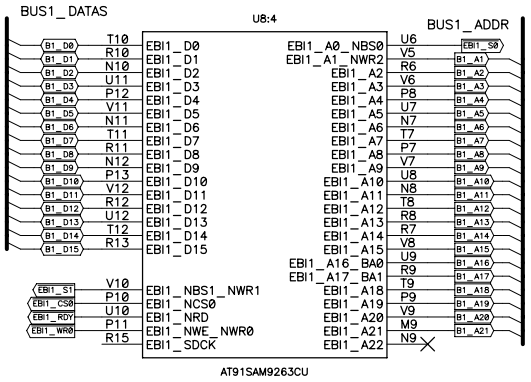
SAM9 motherboard uController_1	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	5/16	./sam9/schematic/ SAM9_motherboard_bis.sch

uController 2

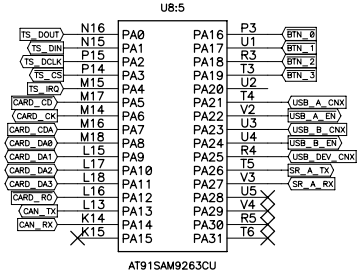
EBI0



EBI1



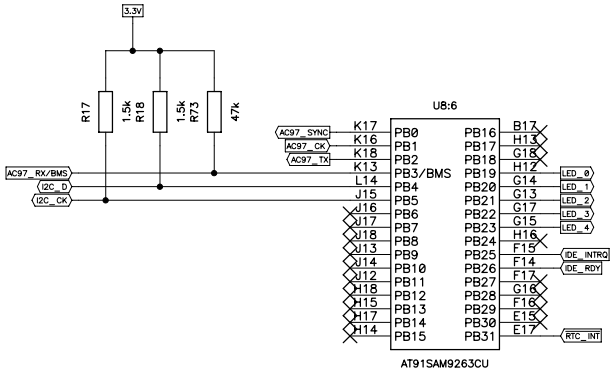
PA: CAN/MCI/BTNs...



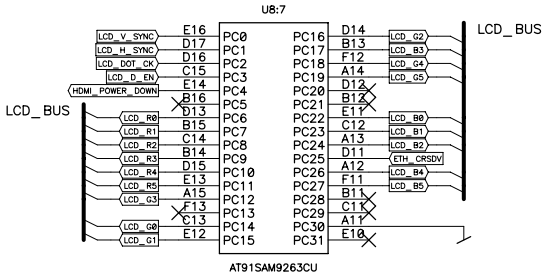
SAM9 motherboard uController_2	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	6/16	./sam9/schematic/ SAM9_motherboard_bis.sch

uController 3

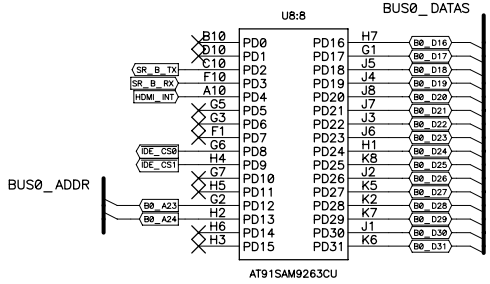
PB: AC97/I2C/LEDs...



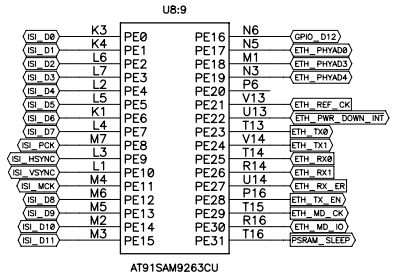
PC: LCD/ETH...



PD: EBI0/USART/IDE...



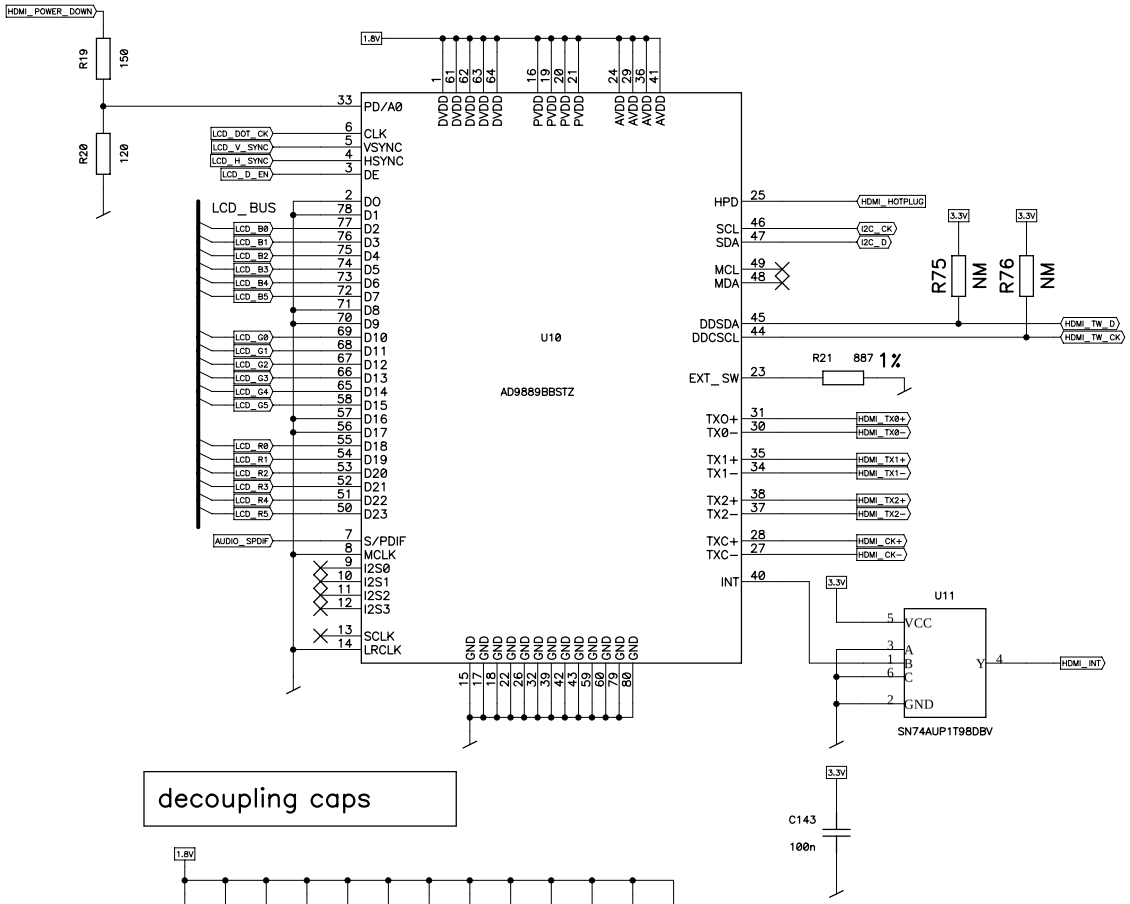
PE: ETH/ISI...



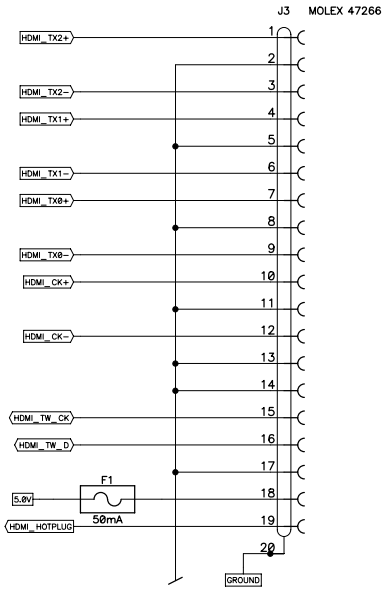
SAM9 motherboard uController_3	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	7/16	./sam9/schematic/ SAM9_motherboard_bis.sch

Video

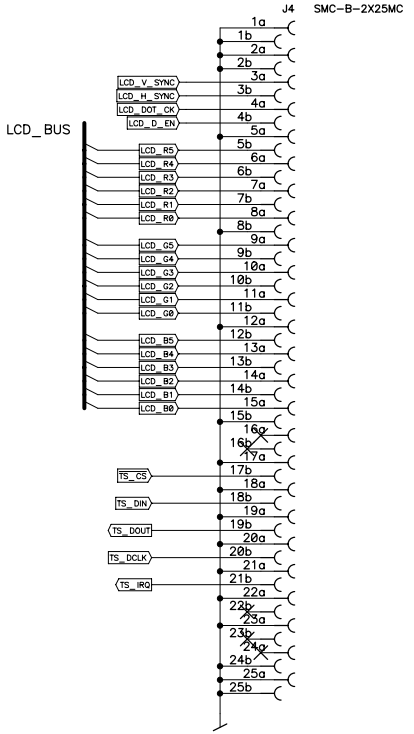
Video transmitter



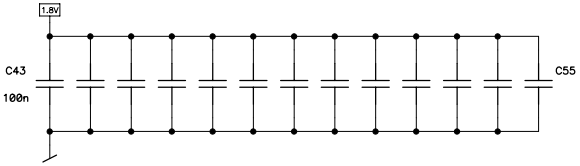
HDMI connector



LCD/touch connector



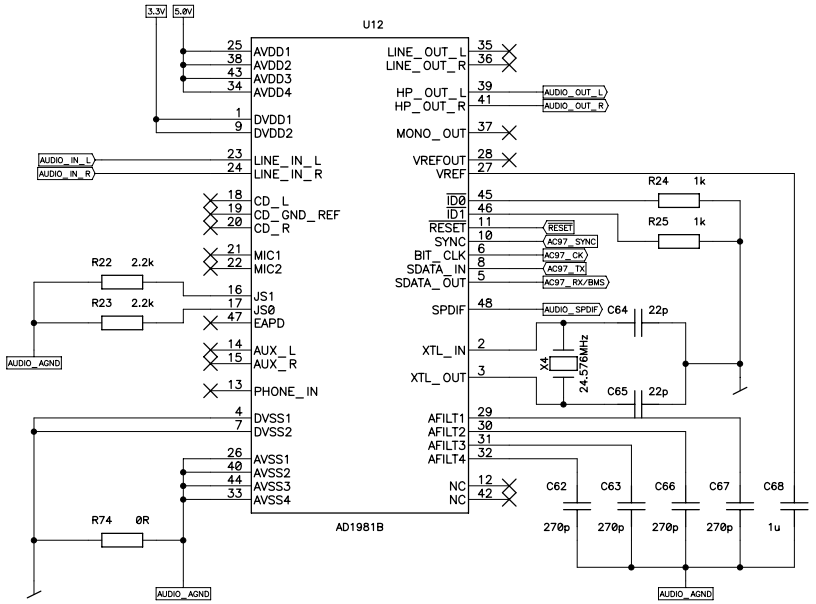
decoupling caps



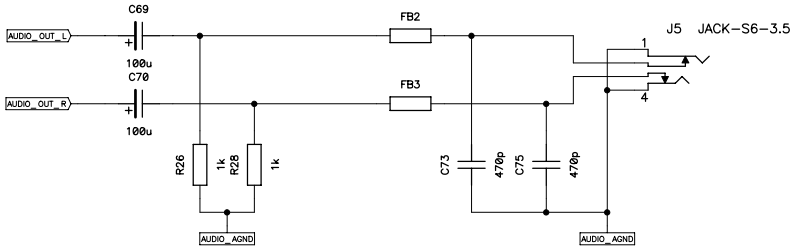
SAM9 motherboard Video	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	8/16	./sam9/schematic/ SAM9_motherboard_bis.sch

Audio

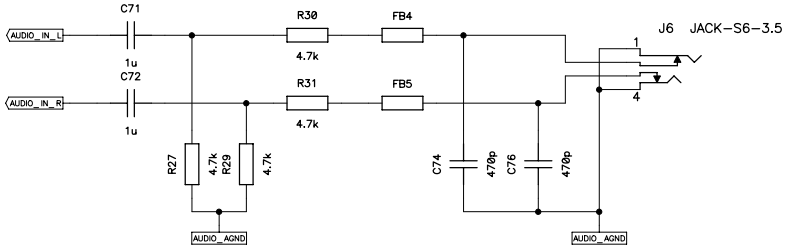
Audio Controller



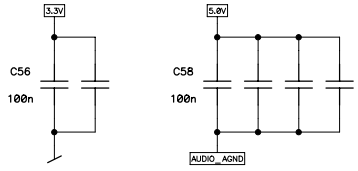
Audio Out



Audio In



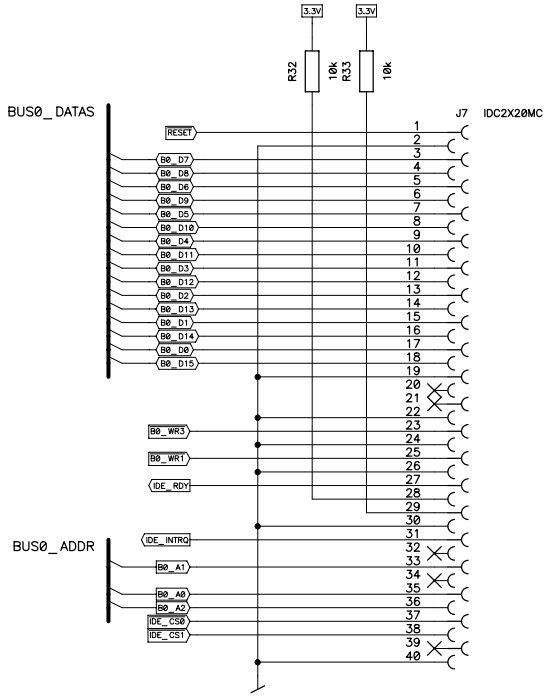
decoupling caps



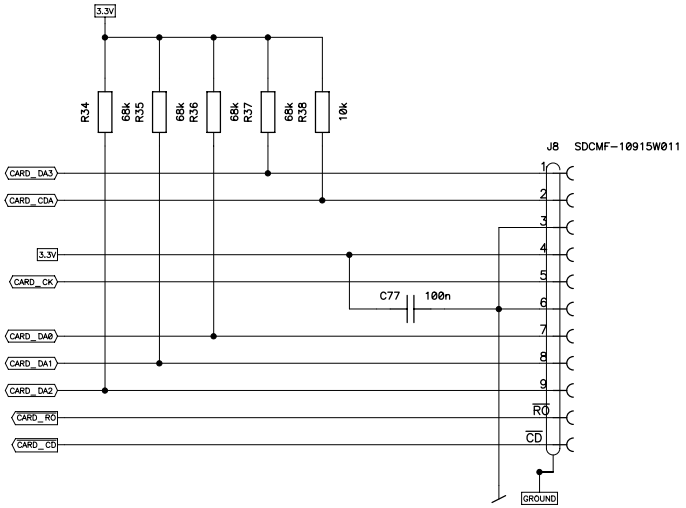
SAM9 motherboard Audio	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	9/16	./sam9/schematic/ SAM9_motherboard_bis.sch

IDE / Card Interface

IDE connector



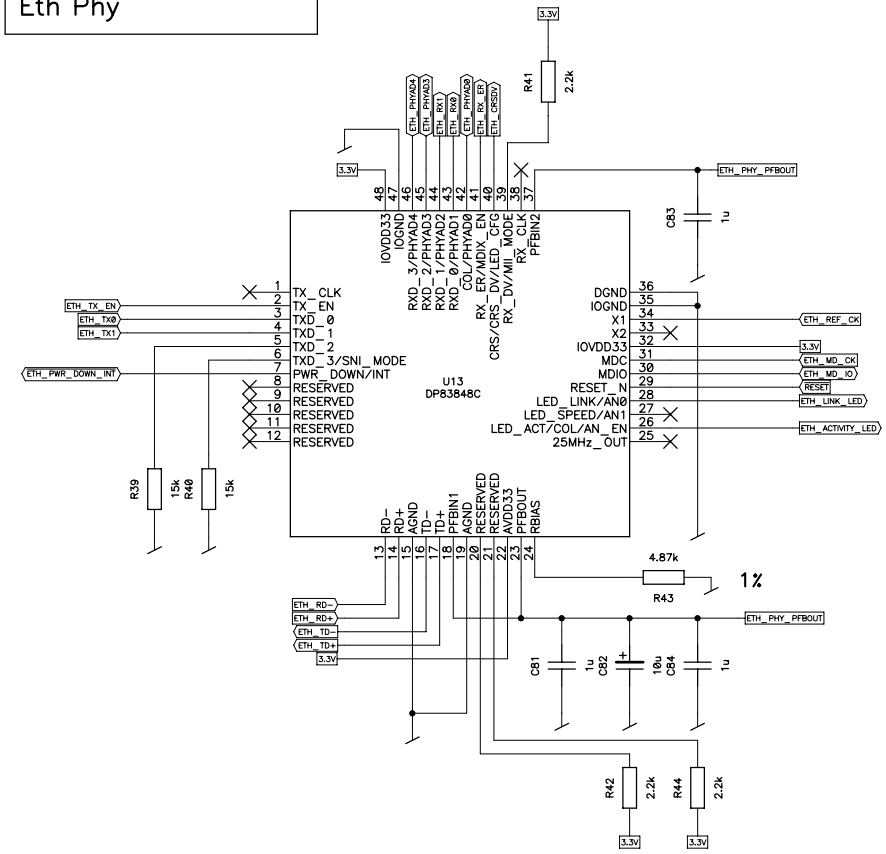
Card Iterface



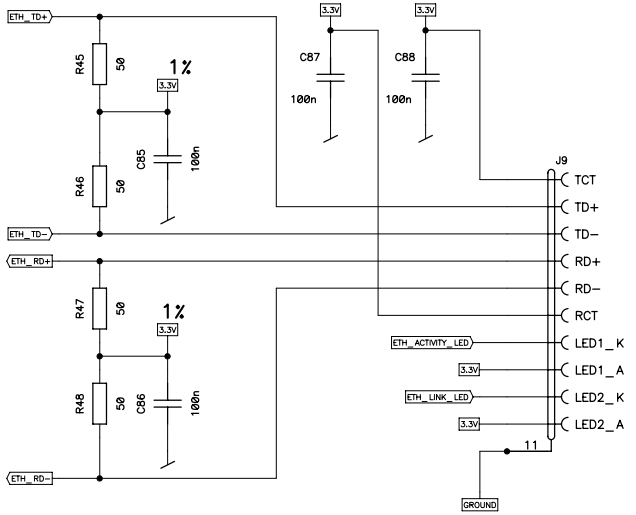
SAM9 motherboard IDE / Card Interface	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	10/16	./sam9/schematic/ SAM9_motherboard_bis.sch

Ethernet

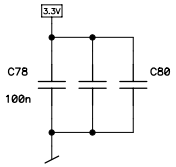
Eth Phy



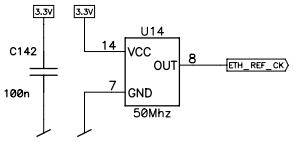
Eth Connector



decoupling caps



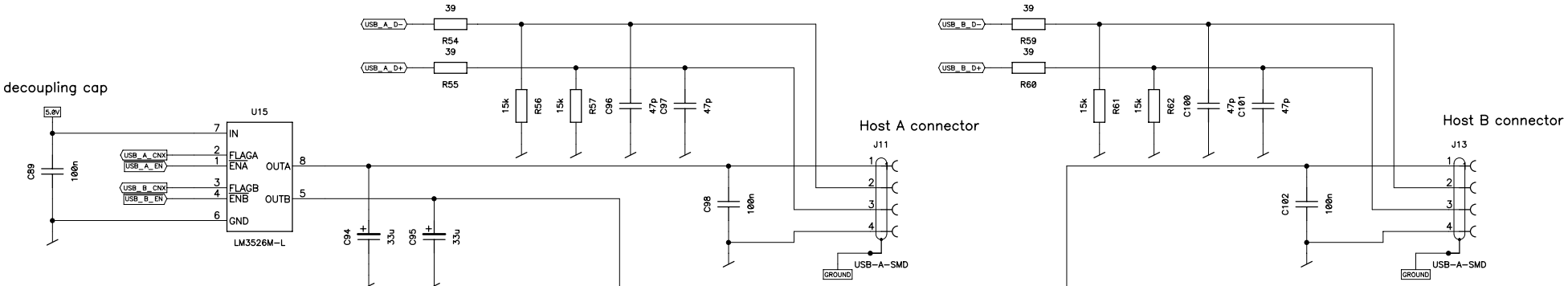
50Mhz Quartz



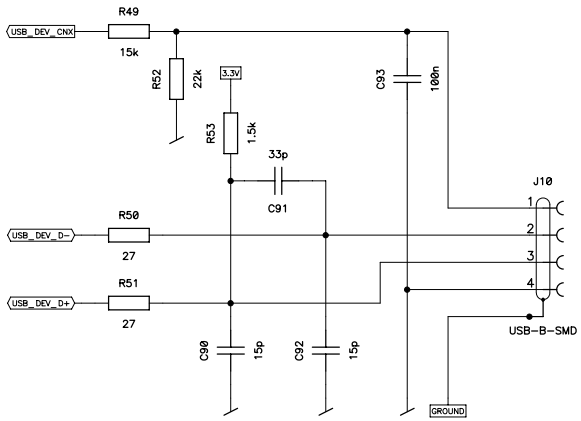
SAM9 motherboard Ethernet	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	11/16	./sam9/schematic/ SAM9_motherboard_bis.sch

USB / CAN

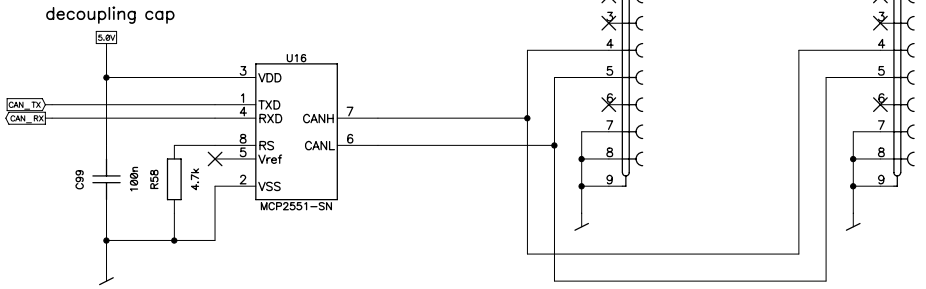
USB Host A & B



USB Device



CAN

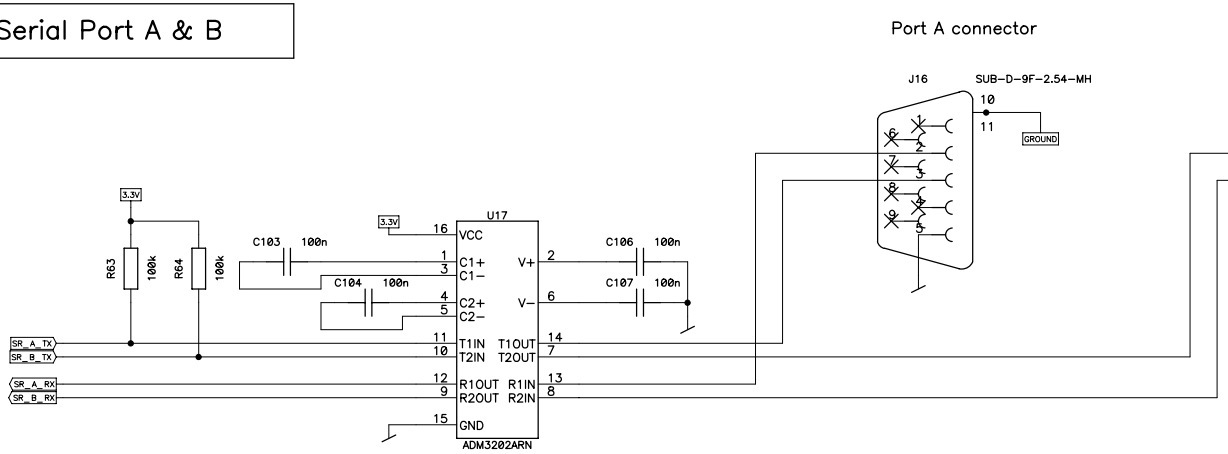


NOTE :
CAN connector is compliant with the
HES-SO/Valais connecting scheme

SAM9 motherboard USB / CAN	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	12/16	./sam9/schematic/ SAM9_ motherboard_ bis.sch

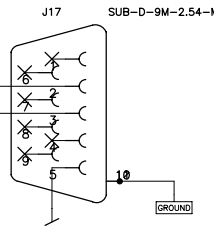
Serial / GPIO / JTAG

Serial Port A & B

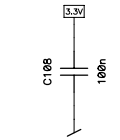


Port A connector

Port B connector

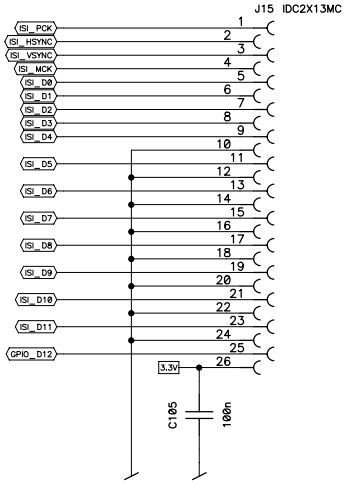


decoupling cap



NOTE :
only transmitted data (TxD) and received data (RxD) are implemented

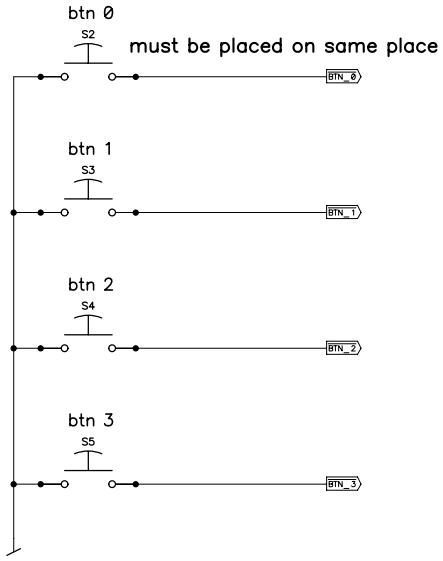
GPIO



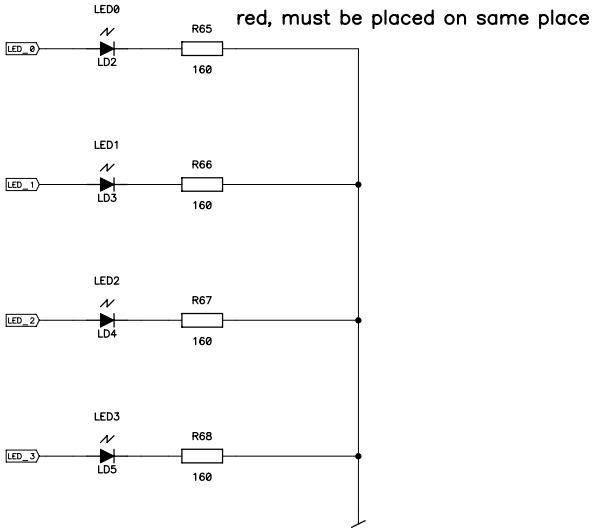
SAM9 motherboard Serial / GPIO	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	13/16	./sam9/schematic/ SAM9_motherboard_bis.sch

Basic IOs

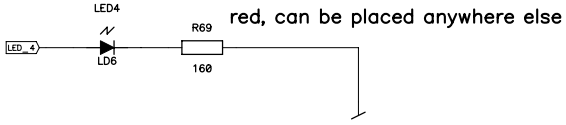
Buttons



LEDs



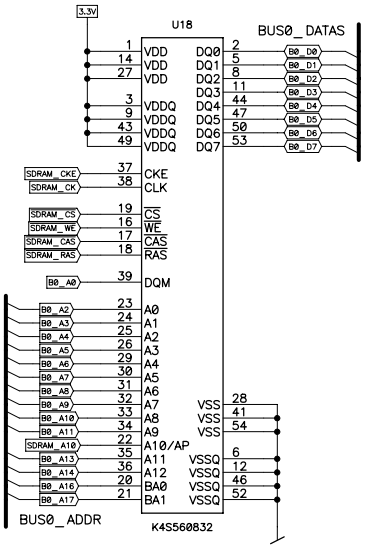
Activity LED

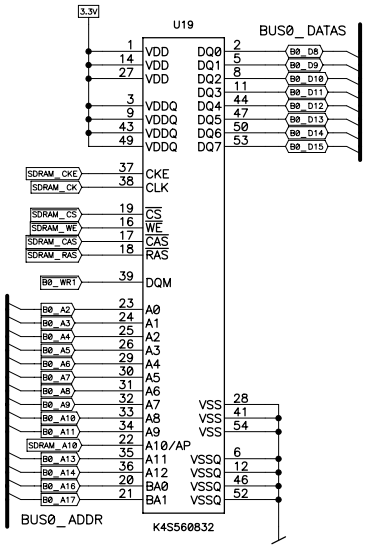


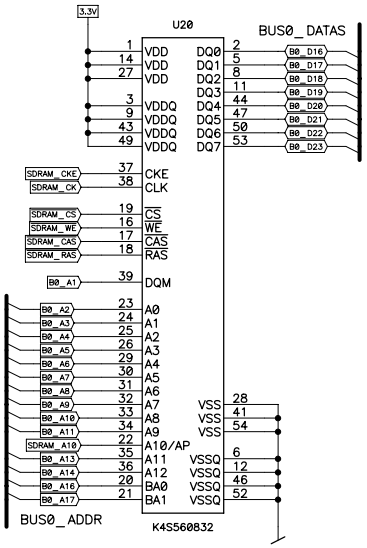
SAM9 motherboard Basic IOs	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	14/16	./sam9/schematic/ SAM9_motherboard_bis.sch

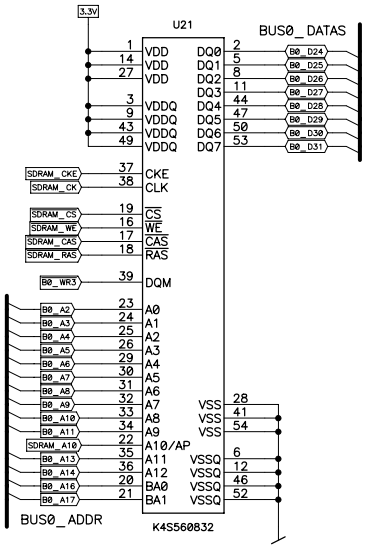
Memory 1

SDRAM, 128MB

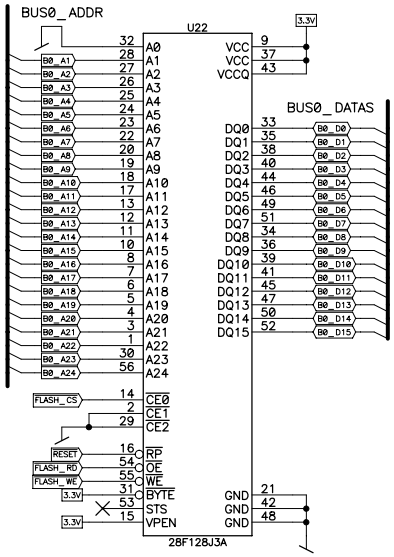




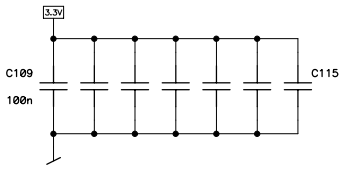


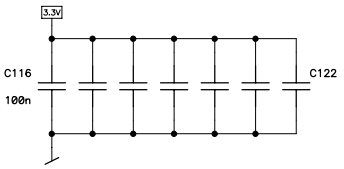


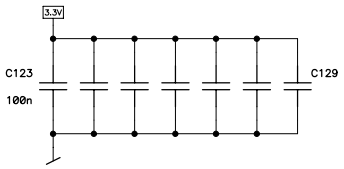
Flash, 16MB

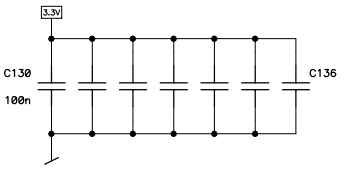


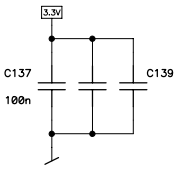
decoupling caps







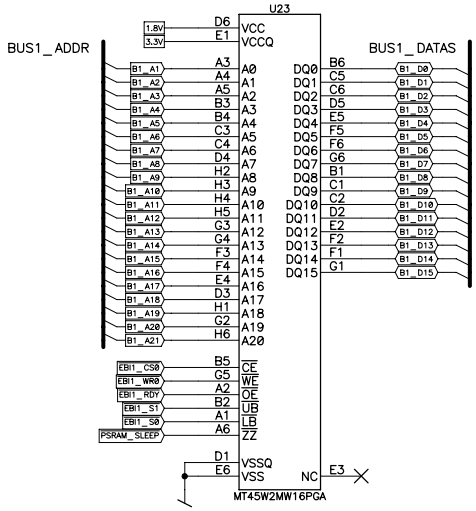




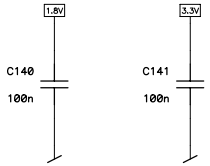
SAM9 motherboard Memory 1	DES	31.05.2008 Epiney Yvan
	REV	V1.0
HAUTE ECOLE VALAISANNE	15/16	./sam9/schematic/ SAM9_motherboard_bis.sch

Memory 2

PSRAM, 4MB



decoupling caps



SAM9 motherboard Memory 2		DES	31.05.2008 Epiney Yvan
		REV	V1.0
HAUTE ECOLE VALAISANNE		16/16	./sam9/schematic/ SAM9_motherboard_bis.sch