ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M. Ing.

PAR
NORMAND LECLERC

CONCEPTION D'UNE PLATEFORME DE TESTS DE CIRCUITS D'INTÉGRATION DIRECTE SUR TRANCHE

MONTRÉAL, LE 29 SEPTEMBRE 2003

CE MÉMOIRE À ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Claude Thibeault, directeur de mémoire
Département de génie électrique de l'École de technologie supérieure

M. Yvon Savaria, codirecteur de mémoire
Département de génie électrique de l'École Polytechnique de Montréal

M. Jean Belzile, président du jury
Département de génie électrique de l'École de technologie supérieure

M. René Toutant, Principal Staff Engineer
Hyperchip – Engineering Department

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY

LE 23 JUILLET 2003

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

Erinyes

« Les Furies, en Grec Erinyes, ou euphémique Eumenides, étaient des esprits vengeurs de la justice. Leurs noms, qui au cours du temps en est venu à être fixé au nombre de trois, étaient Alecto, Magaera et Tisiphone. Leur tâche était de punir les crimes qui n'étaient pas à la portée de la justice des hommes. » (Bulfinch's Mythology)

CONCEPTION D'UNE CARTE DE TESTS DE CIRCUITS D'INTÉGRATION DIRECTE SUR TRANCHE

Normand Leclerc

SOMMAIRE

L'intégration directe sur tranche est une technique de fabrication de puces électroniques pour laquelle une seule puce couvre la grande majorité de la surface d'une tranche. Cette technique présente un très grand potentiel d'intégration mais comporte des risques technologiques importants. Malgré ces risques, la compagnie Hyperchip croit qu'il serait possible d'améliorer son produit en utilisant une telle technologie.

Dans le cadre d'une coopération universitaire, Hyperchip a conçu un certain nombre de puces qui contiennent des structures de tests ainsi que des stratégies pour contourner les différents problèmes potentiels. Ces puces n'ont, jusqu'à présent, jamais été vérifiées. Dans le but de valider les idées implantées dans ces démonstrateurs, une carte de test est nécessaire. Cette carte doit être assez flexible pour permettre la vérification de tous les démonstrateurs présents et futurs.

Il existe plusieurs cartes sur le marché mais aucune d'entre elles ne satisfait l'exigence du nombre de ports demandés par la spécification préliminaire. La conception d'une carte dédiée est donc requise. Ce projet porte sur la conception de cette carte de test dédiée : Erinyes.

Les spécifications de deux démonstrateurs seront utilisées pour guider la conception : le démo 4 et le démo 5. Le démo 4 présente un mécanisme de tolérance aux défectuosités de fabrication des puces d'intégration directe sur tranche. Le démo 5 quant à lui, explore les problèmes liés à la diaphonie et à la température.

Étant donné le contexte particulier entourant ce projet, son étendue et ses contributions ont été limitées à la conception logique du matériel, à la programmation des circuits intégrés de type FPGA et à la planification des modifications nécessaires au système d'exploitation µCLinux.

AVANT-PROPOS

L'intégration directe sur tranche est un sujet de la microélectronique qui reste encore inexploité. Cette technique offre certains avantages qui ont poussé la compagnie montréalaise Hyperchip à fonder un groupe de recherche sur le sujet. Pour permettre de tester certaines puces de démonstration, ce groupe a demandé à l'École de technologie supérieure, en tant que partenaire de recherche, de concevoir une carte de test.

Le projet fut entamé en juin 2001 pour être abandonné un an plus tard, en juin 2002. La dite carte n'a jamais vu le jour. Le travail de conception schématique ainsi que la planification de la carte de circuits imprimés sont terminés. La durée du travail restant a été évaluée à une année/homme de travail.

Les remerciements vont aux professeurs Claude Thibeault et Yvon Savaria, superviseurs du projet, M. Karl Fecteau, responsable du projet chez Hyperchip, M. Sébastien Clavier, étudiant stagiaire, M. Patrick Francq, M. Patrick Vincent, M. Philippe-Pierre Marion et M. René Toutant, ingénieurs chez Hyperchip.

Je tiens également à remercier mes parents et frères pour leur support sans lequel il m'aurait été impossible d'accéder aux études supérieures.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS ET DES SIGLES

AN	Analogique à Numérique
BDM	Background Debug Mode
CAN	Controlled Area Network
CC	Courant Continu
cPCI	Compact PCI
DCM	Digital Clock Manager
DDR	Double Data Rate
DIMM	Dual In-line Memory Module
DLL	Delay Lock Loop
EIA	Electronic Industries Alliance
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
IrDA	Infrared Data Assiciation
JTAG	Joint Test Action Group
LVCMOS	Low Voltage Complementary Metal-Oxide Semiconductor
LVPECL	Low Voltage Positive Emitter Coupled Logic
LVTTL	Low Voltage ToTem pole Logic
NA	Numérique à Analogique
PCI	Peripherial Component Interconnection
PCMCIA	Personal Computer Memory Card International Association
SCSI	Small Computer System Interface
SoC	System on Chip
USB	Universal Serial Bus
ZBT	Zero Bus Turnaround

CONTRIBUTIONS

La carte de test conçue dans le cadre de ce projet conduira à la fabrication d'un nouvel outil qui permettra de valider les solutions trouvées au problème de la fabrication de puce d'intégration sur tranche. La carte est assez flexible pour permettre de simuler un environnement de test à la fois analogique et digital.

Le placement préliminaire ainsi que la planification des couches de la carte ont été faits. Ce travail permettra aux éventuels concepteurs de la carte de planifier le routage. Le routage devra cependant être manuellement dirigé pour que les contraintes énoncées dans le présent document soient respectées.

Au cours du projet, un banc de test pour le 4e démo fut complété. Ce banc de test a servi à valider le routage et la fonctionnalité de la puce avant sa soumission pour fabrication. En raison de l'utilisation de structures à multiples dimensions, ce code n'est pas synthétisable. Malgré ce fait, certaines fonctions pourront être reprises pour l'écriture du microcode du FPGA.

La structure du banc de tests du démo 5 a été élaborée sans toutefois avoir été réalisée. Cette structure est accompagnée d'une méthodologie de test. Ces deux éléments forment l'essentiel du fonctionnement d'un banc de vérification. Le travail de conception du microprogramme pour la vérification du 5e démo ne consiste qu'à composer en VHDL les idées énoncées.

Le microprogramme servant au contrôle et à la communication entre les différentes composantes de la carte a été développé et simulé. Ce code pourra servir à la fois de code de vérification de la carte et de code de base pour le test.

Le noyau de Linux est une structure relativement complexe et sa modification peut être difficile. Le travail de planification des modifications du système d'exploitation µCLinux a été élaboré pour permettre de les effectuer de manière structurée.

INTRODUCTION

Au début des années 60, la marine américaine a lancé des recherches sur le contrôle de son armement en temps de guerre. Le problème était de conserver ce contrôle même si l'Amérique était touchée par une bombe atomique. En 1969, l'Internet, un petit réseau de recherche militaire, était créé. Ce réseau de 4 machines reliait l'Université de Californie à Los Angeles, la Stanford Research Institute, l'Université de Californie à Santa Barbara et l'Université de l'Utah. Aujourd'hui, le réseau Internet compte plus de 15 millions de serveurs branchés sur des liens terrestres et aériens allant de 50Kbps à plus de 155Mbps.

Avec un nombre aussi important de serveurs, il est essentiel d'avoir un système efficace de transport de l'information. De nombreuses décisions doivent être prises pour acheminer les paquets à destination. À chaque nœud, il faut trouver le meilleur segment. Le pouvoir décisionnel de l'Internet provient des routeurs. Ces équipements très rapides et dispendieux doivent subvenir à un besoin de plus en plus grandissant. La plupart des équipements en place sont déjà surchargés et depuis déjà quelque temps, il est possible d'observer des engorgements à certains nœuds.

Pour faire face à la nouvelle demande, une jeune compagnie montréalaise propose un système de routage très flexible entièrement basé sur le protocole IP. Les routeurs Hyperchip devraient offrir un débit jusqu'à 1000 fois supérieur à celui des routeurs existants. Pour conserver son avance dans le domaine, une équipe de recherche et développement de la compagnie se penche vers l'intégration à l'échelle de la tranche.

L'Office Québécois de la langue française[8] définit l'intégration à l'échelle de la tranche comme « *une technique de fabrication de circuits consistant à diffuser sur une seule tranche un système électronique complet ayant de 25 à 100 fois la capacité des microprocesseurs actuels* ». Le système actuel proposé par Hyperchip repose sur la

coopération de nombreuses puces réparties sur plusieurs cartes. Or, lorsqu'un signal sort d'un circuit imprimé, il est exposé aux interférences du milieu et par conséquent, il est ralenti, limitant la vitesse du système. Une solution à ce problème est de concevoir un système complet tenant sur une seule puce. Il serait possible, par exemple, de concevoir une carte du routeur qui aurait le même pouvoir d'opération que trois autres fonctionnant en parallèle. L'utilisation de cette technique comporte des risques technologiques importants. Si ces éléments de risque se manifestent déjà au niveau des techniques d'intégrations plus conventionnelles, ils se trouvent amplifiés à l'échelle de la tranche. Le rendement, l'interférence entre les signaux, la dissipation de la chaleur ainsi que les gradients de température sont parmi les éléments de risques les plus importants.

La fabrication des puces électroniques se base sur des procédés de fabrication effectués en plusieurs étapes. Ces procédés sont réalisés dans une chambre blanche où l'atmosphère y est filtrée. La quantité de poussière en circulation, dans ces chambres, y est de moins d'une particule par mètre cube. Malgré toutes les précautions prises lors de la fabrication, pour chacune des étapes, il y a un risque d'introduction de défectuosités. Il est à toute fin pratique, impossible de fabriquer une tranche sur laquelle toutes les cellules sont parfaites. De ce fait, le rendement anticipé d'une puce d'intégration à l'échelle de la tranche est très faible. Par conséquent, pour être en mesure de réaliser de telles puces, il est essentiel d'introduire des structures de tolérance aux défectuosités qui permettront d'augmenter le rendement.

La densité des circuits est de plus en plus importante mais ceux-ci ne semblent pas diminuer de taille. Bien au contraire, les circuits intégrés tendent à prendre des superficies de plus en plus grandes. Ceci signifie que les signaux doivent traverser de plus en plus grandes distances tandis que l'espacement entre eux devient de plus en plus restreint. Ceci a pour effet de rendre ces signaux de plus en plus sujets aux effets d'interférences (diaphonie). Ce phénomène prend encore plus d'importance dans une application d'intégration directe sur tranche. Les cellules des extrémités de la tranche

deviennent très éloignées les unes des autres et les signaux les reliant se transforment littéralement en lignes de transmission très longues. Les effets d'interférence peuvent prendre des proportions telles qu'ils viennent perturber les stratégies conventionnelles de synchronisation, qui doivent être ajustées en conséquence.

Outre le rendement et la diaphonie, la dissipation de la chaleur des circuits devient également un problème. L'augmentation du nombre de transistors ainsi que l'accélération des circuits se traduisent par une augmentation du nombre de transitions par secondes. Se faisant, il y a une diminution considérable du temps de refroidissement, se traduisant par un accroissement non négligeable de la température. Cet accroissement de la température est déjà une préoccupation pour les circuits complexes réalisés via des techniques d'intégration conventionnelles. Par exemple, un microprocesseur Pentium 4 d'Intel© cadencé à 2 gigahertz doit dissiper une puissance thermique d'environ 75 watt. Pour dissiper toute cette chaleur, ces puces doivent être munies d'un dissipateur de chaleur actif. L'intégration à l'échelle de la tranche amène une dimension nouvelle en raison de gradients de température anticipés. En effet, les puces étant très complexes, elles sont généralement subdivisées par fonctions. De ce fait, il est rare d'utiliser une puce à 100% en tout temps. Se faisant, il y a création de gradients de température sur la surface de la puce[1]. Parmi les conséquences prévues de tels gradients, notons dans un premier temps que d'importants gradients peuvent causer de sérieuses contraintes mécaniques et endommager le circuit. D'autre part, puisque la vitesse des transistors diminue lorsque la température augmente, les gradients de température peuvent affecter la synchronisation de la puce en introduisant des biais de manière inégale. Le courant électrique ne voyage donc pas partout à la même vitesse et une désynchronisation est inévitable.

Face à ces problématiques, le groupe de recherche en intégration directe sur tranche fut mandaté afin d'estimer l'impact réel de ces effets et dans le cas échéant, de proposer des solutions afin de rendre la fabrication d'une puce d'intégration directe sur tranche

possible. Pour valider leurs hypothèses ainsi que leurs solutions, le groupe, a conçu 5 puces de démonstrations. Ces démonstrateurs (ou démos) contiennent des structures spéciales permettant d'investiguer les effets énoncés plus haut.

L'objectif de ce projet de maîtrise est de concevoir une carte qui agirait comme banc de test pour les derniers démonstrateurs appelés ci-après démos 4 et 5. La carte doit être simple d'utilisation avec une interface conviviale et flexible, permettant de simuler un environnement de test pour les démos. La carte devait permettre d'effectuer tous les tests nécessaires sur les deux démonstrateurs d'intérêt, en plus de permettre la vérification des démos futurs. Étant donné le contexte particulier entourant ce projet, son étendue et ses contributions ont été limitées à la conception logique du matériel, à la programmation des circuits intégrés de type FPGA et à la planification des modifications nécessaires au système d'exploitation μCLinux.

Le reste du mémoire est structuré comme suit. Pour bien comprendre l'architecture de la carte de tests, les architectures des démos 4 et 5 seront respectivement présentées aux chapitres 1 et 2. Il sera ensuite question au chapitre 3 d'une analyse des besoins ainsi que de l'inventaire des cartes disponibles sur le marché en rapport avec la spécification préliminaire. Enfin, l'architecture comme telle de la carte de test sera examinée en détail d'un point de vue matériel (chapitre 4) et d'un point de vue logiciel (chapitre 5). Suivront finalement la conclusion et les recommandations.

CHAPITRE 1

LE DÉMO 4 : TOLÉRANCE AUX DÉFECTUOSITÉS DE FABRICATION

Pour permettre la réalisation d'un circuit à intégration directe sur tranche, il faut doter la puce d'un mécanisme qui lui permettrait de survivre à une ou plusieurs défectuosités de fabrication. Le mécanisme pourrait être statique ou dynamique. La fabrication des mémoires utilise déjà un mécanisme statique de tolérance aux défectuosités où les cellules défectueuses sont remplacées physiquement par des cellules fonctionnelles avant l'étape de l'encapsulation.

Le démo 4 est une implémentation de reconfiguration dynamique. La reconfiguration dynamique s'effectue de façon entièrement électronique. Elle fait partie intégrante de la logique de la puce. Le démo 4 fut également utilisé pour le développement d'une interface JTAG[4]. Cette interface devait être réutilisée par la suite dans certaines des puces de la compagnie Hyperchip.

Ce démo a été conçu dans le cadre du cours *Conception de circuits électroniques intégrés II* de l'école Polytechnique de Montréal à l'automne 2001. L'auteur du présent document avait comme tâche, au sein de l'équipe de conception, de concevoir une méthode de test des structures implémentées.

1.1 Système de reconfiguration

Pour la simplicité de la conception, l'application vise un circuit hautement parallélisé où toutes les cellules sont identiques. Il est composé de 9 cellules reliées entre elles par des bus de données segmentés en 3 sous-bus d'une largeur de 4 bits ou quartets. Le mécanisme de remplacement permet la substitution d'une cellule ou d'un quartet.

Les cellules et les bus du démo 4 peuvent être remplacées par un voisin immédiat ou un voisin secondaire dans la direction horizontale ou verticale. Ce remplacement peut être partiel ou total. La figure 1 montre un exemple de remplacement complet d'une cellule. Sur cette illustration, la cellule 2 est déclarée défectueuse. La cellule 1 fut choisie comme cellule de remplacement, alors les cellules 3, 4 et 5 doivent maintenant communiquer avec la cellule 1 plutôt que la cellule 2. Dans une matrice de cellules plus grande, un tel remplacement affecte toutes les cellules d'une ligne ou d'une colonne. Il y aura un effet de décalage des cellules.

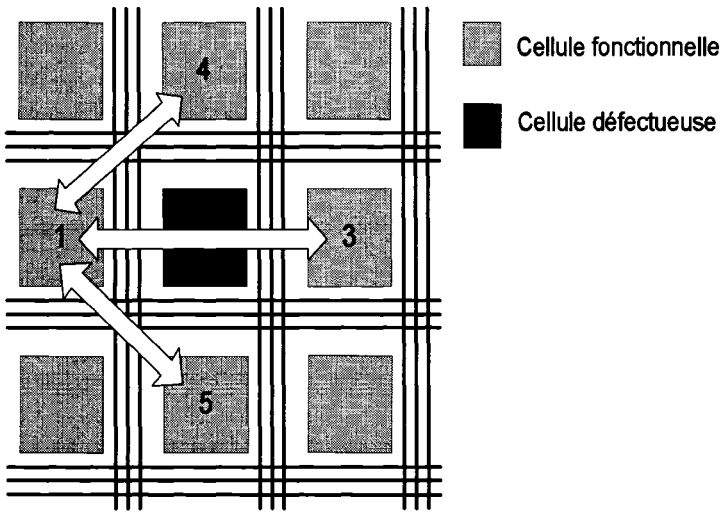


Figure 1 Exemple de remplacement complet d'une cellule.

Il serait également possible d'effectuer le remplacement d'un quartet en transférant le trafic vers un quartet fonctionnel. La structure de reconfiguration du démo 4 ne permet pas la substitution partielle d'un quartet; une ligne complète doit être remplacée. La figure 2 montre deux substitutions de quartets. Il est possible de remplacer un quartet à l'intérieur d'un même bus ou d'effectuer le remplacement d'un quartet par celui d'un autre bus.

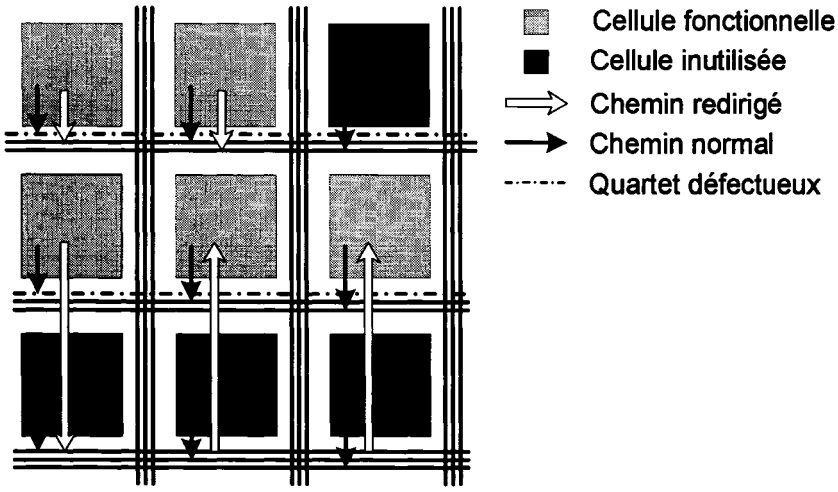


Figure 2 Exemple de remplacement d'un quartet.

1.2 Interface de communication

L'interface JTAG est une interface de communication sérielle synchrone conçue pour permettre de forcer ou d'extraire des données à l'intérieur d'une puce dans le but d'en faire la vérification. L'interface possède 5 signaux de contrôle[4]:

a. nTRST : signal de réinitialisation asynchrone;
b. TCK : signal d'horloge;
c. TMS : signal de contrôle de la machine à états finis;
d. TDI : Signal d'entrée des données;
e. TDO : Signal de sortie des données.

Les données sont introduites par la ligne TDI et sortent par la ligne TDO. Les données sont stockées dans des chaînes de registres à décalage. Il existe deux types de chaînes de registres. La première est appelée registre d'instructions. Cette chaîne permet de

contenir le mode d'opération dans lequel le JTAG fonctionne. La seconde est la chaîne de test qui sert à introduire ou retirer des données de la puce sous test.

La sélection du type de chaîne de registres se fait par l'intermédiaire du Test Access Port (TAP). Ce dernier possède son propre signal de contrôle, TMS, qui permet de traverser les différents états de la machine. La figure 3 représente la machine à états finis TAP.

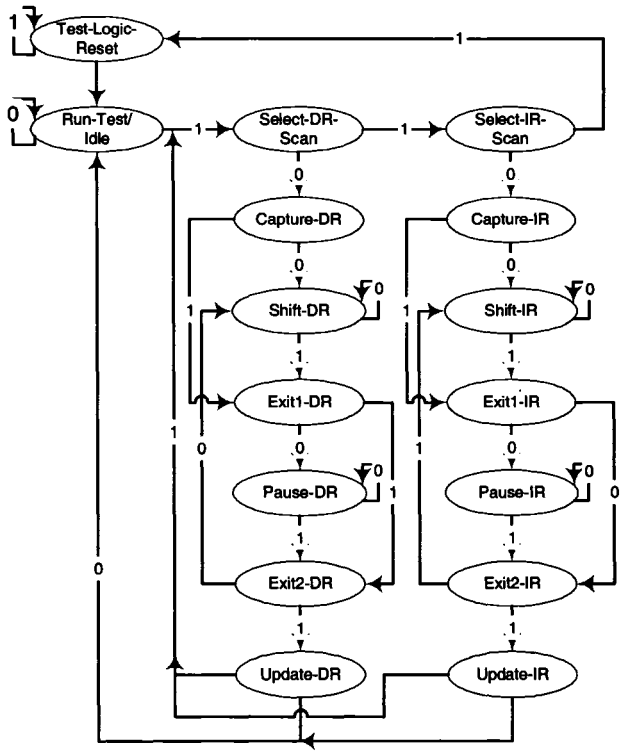


Figure 3 Représentation de la machine à états finis TAP.

Lors de la sélection d'un type de chaîne, la ligne d'entrée TDI, la ligne de sortie TDO ainsi que l'horloge lui sont assignées. De cette façon, une seule chaîne de registres ne peut être active à la fois.

1.2.1 Fonctionnement du JTAG

Lors de la mise sous tension ou lors de l'assertion de la ligne nTRST, le JTAG est placé dans son état de réinitialisation. Aucune chaîne de registres n'est active et le système est automatiquement en mode BYPASS. À l'aide du signal TMS, le TAP est placé dans son état de capture (Capture-DR ou Capture-IR). Lorsque la machine entre dans cet état, les registres actifs sont remis dans leur état de initial sauf pour la chaîne des données qui sera chargée des valeurs du bus. À ce moment, sur chacun des fronts montants de l'horloge TCK, la valeur présente à l'entrée TDI est chargée dans la chaîne de registres. Les données préalablement chargées dans les registres sont quant à eux, transférés vers la sortie : TDO. Ce n'est que lorsque le TAP entre dans le mode Update-DR ou Update-IR que les données présentes dans la chaîne sont effectivement transférées au circuit logique de la puce. Lorsque les données transmises ou capturées, le TAP peut être replacé dans sont état de repos : Run-test/Idle.

1.2.2 Implémentation dans le démo 4

Chacune des cellules du démo 4 compte deux chaînes de registres de test: une chaîne de configuration comprenant 18 registres et une chaîne de données de 24 registres. L'implémentation du JTAG dans le démo 4 comprend un registre d'instructions d'une largeur de 2 bits pour contenir les instructions suivantes :

a. BYPASS (*00*): Dans ce mode, un registre seul est placé entre TDI et TDO. Tous les bits sont ignorés et tout simplement retransmis vers le voisin si présent;
b. SAMPLE_PRELOAD (*01*): Cette instruction est utilisée pour charger les données voulues dans les chaînes de registres de configuration ou de données;

c. EXTEST (*10*): Le mode EXTEST force le JTAG à appliquer les données présentes dans les registres aux différents bus ou aux différentes structures de configuration. La chaîne de registres des données est sélectionnée;
d. CONFIG (*11*) : Pour sélectionner la chaîne de registres qui contrôle les structures de configuration, CONFIG est utilisée.

Tous les contrôleurs JTAG sont branchés en une longue chaîne de 9 éléments; les lignes TDO étant connectées aux lignes TDI des cellules suivantes. Tous les signaux nTRST, TCK et TMS sont communs. La figure 4 illustre le branchement du JTAG de chacune des 9 cellules du démo 4.

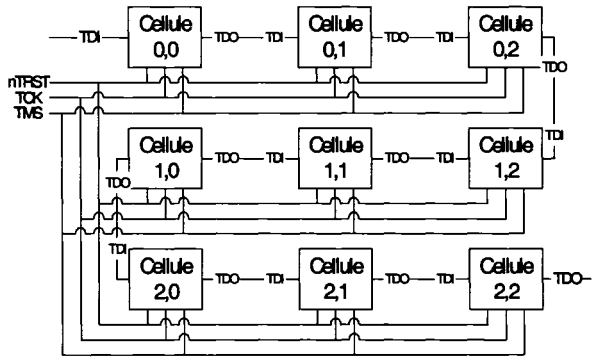


Figure 4 Branchement du JTAG à l'intérieur du démo 4.

Un tel branchement signifie que pour envoyer des données à une et une seule cellule, il faut tout d'abord placer toutes les autres en mode BYPASS. Le test sera plus long que si les cellules étaient indépendantes et les vecteurs devront être augmentés pour tenir compte des registres des cellules en mode BYPASS.

1.3 Méthodologie de test

Les mécanismes de reconfiguration du démo 4 reposent sur le bon fonctionnement du JTAG. Si une cellule coupe la chaîne, la puce au complet devient inutilisable et les stratégies de configuration implémentées dans la puce ne peuvent être testées. Le test est divisé en trois grandes parties : le test du JTAG, le test du bus et le test des structures de configuration. Chacun de ces tests devrait être fait dans l'ordre présenté dans ce document. Le banc de test utilisé lors de la vérification du démo 4 avant soumission pour fabrication est présenté à l'annexe 2. Le lecteur pourra se référer en annexe 3 pour obtenir un aperçu graphique des résultats de simulations.

1.3.1 Test du JTAG

Le test du contrôleur JTAG se fait en quatre temps: le test du registre d'instruction, le test du registre BYPASS, le test de la chaîne de configuration et finalement, le test de la chaîne des données.

1.3.1.1 Le registre d'instruction

Le test du JTAG commence par le registre d'instruction de chacune des cellules. Lorsque le TAP entre dans le mode de capture du registre d'instruction, le registre est préchargé du code binaire *01*. Il est donc possible d'effectuer une première vérification de ce registre en chargeant une instruction quelconque pour en relire le contenu initial. Si une cellule coupe la chaîne, les données de toutes celles qui la précèdent seront détruites. Il est donc facile d'isoler la cellule défectueuse. Cependant, aucun mécanisme de tolérance du JTAG n'est implanté dans ce démo et une défectuosité du registre d'instruction d'une cellule signifie que la puce est inutilisable.

Advenant un échec, les autres tests ne pourront être effectués que sur les cellules qui précèdent celle qui a coupé la chaîne. En effet, puisque le registre d'instruction de toutes les cellules est sélectionné en même temps, un bris dans la chaîne ne permettra pas à l'instruction de voyager au-delà de la coupure.

1.3.1.2 Le registre BYPASS

Le test de ce registre s'effectue en chargeant le code BYPASS dans le registre d'instruction et en y laissant passer un train de bits pour le relire enfin en sortie. Le vecteur de test envoyé ne doit pas être modifié lors de son transfert. Ce vecteur devrait être formé de 19 bits. Il y a 9 cellules et chacun des bits doit ressortir de la chaîne. Un vecteur de test dont seul le bit le moins significatif est non nul, permettrait de déceler un problème. Lorsqu'une cellule brise la chaîne, tous les bits du vecteur seront changés qu'il s'agisse d'un bit collé à 1 ou à 0. La figure 5 illustre cette idée. Six bits sont transférés dans les 3 registres, les 6 bits sortants seront changés par la coupure.

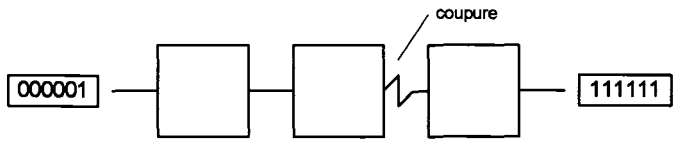


Figure 5 Résultat d'un bris dans une chaîne de registres

Le registre BYPASS n'est présent que pour accélérer les communications intercellulaires du JTAG. N'étant composé que d'un seul registre, il offre un délai réduit en comparaison avec les autres chaînes de registres.

1.3.1.3 La chaîne de configuration

La chaîne de configuration est composée de 18 registres. Ces registres sont remis à zéro lorsque le TAP entre dans son mode de capture. Pour son test, il faut procéder comme dans le cas du registre BYPASS et construire un vecteur de test qui soit assez long pour permettre au résultat de sortir.

Le test idéal de ce registre se fait cellule par cellule. Ceci est obtenu en plaçant toutes les cellules en mode BYPASS sauf celle sous test. Le vecteur doit alors passer à travers les registres de configuration de la cellule cible et les registres BYPASS des autres cellules. La chaîne ainsi créée demanderait un vecteur de 26 bits (18 registres de configuration, 8 registres BYPASS). Cependant, pour permettre au résultat de sortir de la chaîne, la longueur de ce vecteur devra être doublée et augmentée de 1 bit. Une fois de plus, un vecteur dont seul le bit le moins significatif est 1 permettrait la détection d'un problème dans la chaîne. La figure 6 montre le test de la chaîne de registres de la cellule centrale.

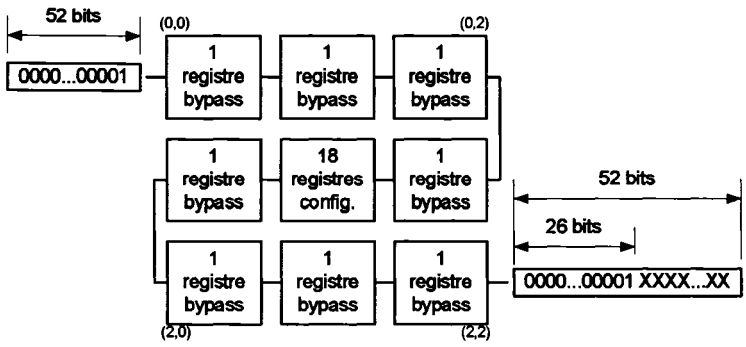


Figure 6 Test de la chaîne de registres de configuration

Une seule modification dans le vecteur de test montre que la cellule ne peut être reconfigurée. Elle pourrait cependant être utilisable si elle passe le test de la chaîne de

registres des données. Advenant un échec au registre BYPASS, il serait tout de même possible de tester les chaînes de configuration. En plaçant toutes les cellules en mode de configuration, un test global peut être initié. Il serait par contre impossible de localiser une cellule défectueuse par cette méthode. La figure 7 montre le test de la cellule centrale advenant un échec dans le test des registres BYPASS.

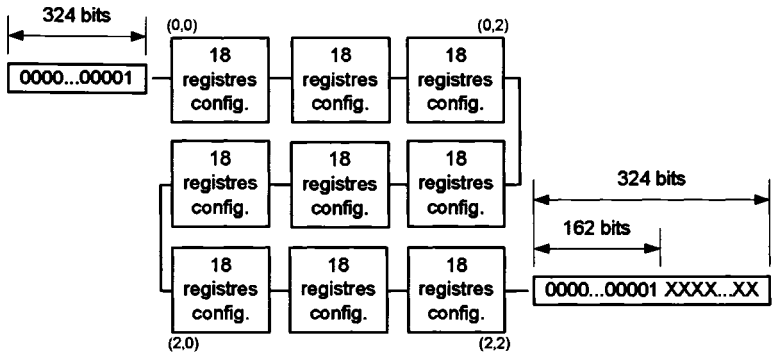


Figure 7 Test de la chaîne de configuration avec erreur dans la chaîne BYPASS

1.3.1.4 La chaîne des données

Le test de la chaîne des données est très semblable au test de la chaîne de configuration. Elle n'est cependant pas mise à zéro lorsque le TAP entre dans le mode de capture; elle charge plutôt les données du bus. Autre différence: la chaîne comporte 24 éléments plutôt que 18.

Le test idéal utiliserait alors un vecteur de 65 bits : 24 registres de données et 8 registres BYPASS qui doivent être doublés et augmentés de 1 bit pour permettre d'extraire le vecteur initialement transmis. Ce vecteur pourrait encore une fois être composé que d'un seul au bit le moins significatif 1, le reste étant à 0. Une seule modification dans le vecteur signifie que la cellule est inutilisable et doit être remplacée. Il est également

possible de tester cette chaîne de façon globale dans le cas où la puce n'ait pas passé le test du mode BYPASS. Dans un tel cas, il est une fois de plus impossible de localiser la cellule défectueuse.

1.3.1.5 Tests avancés

Advenant un échec dans l'un ou l'autre des tests énoncés plus haut, il est possible d'utiliser des chemins alternatifs dans le but de compléter le test en faute. Lors d'un échec au test du registre BYPASS par exemple, il est possible d'utiliser une des deux autres chaînes de registres pour trouver la cellule défectueuse. Ainsi, le test de la chaîne de configuration et le test de la chaîne des données deviennent des façons de sonder la puce pour les cellules défectueuses en mode BYPASS. De la même façon, il serait possible d'utiliser la chaîne des données pour isoler la ou les cellules qui ont une chaîne de configuration défectueuse.

Considérons par exemple une puce possédant un bris dans le mode BYPASS mais dont toutes les chaînes de configuration sont parfaites: les cellules sont tout d'abord placées en mode configuration puis successivement, elles sont testées en mode BYPASS. Lorsque le vecteur de test change, cela signifie que la dernière cellule mise en BYPASS est celle qui est défectueuse. La recherche peut continuer en plaçant cette cellule en mode de configuration.

Il est cependant possible qu'une cellule possède une ou plusieurs défectuosités dans le registre BYPASS ainsi que dans les deux chaînes. Il serait alors possible de tester les cellules une à une selon différentes combinaisons de modes pour isoler celles qui sont défectueuses. Lorsque toutes les cellules défectueuses ont été trouvées, il est alors possible de les placer sur la chaîne de registres qui permettra de laisser passer les données au travers de la cellule. La figure 8 montre le test de la cellule (2,1) sachant que

la cellule (0,3) ne passe pas le test du registre BYPASS et que la cellule (1,0) ne passe ni le test du registre BYPASS ni le test des registres de configuration.

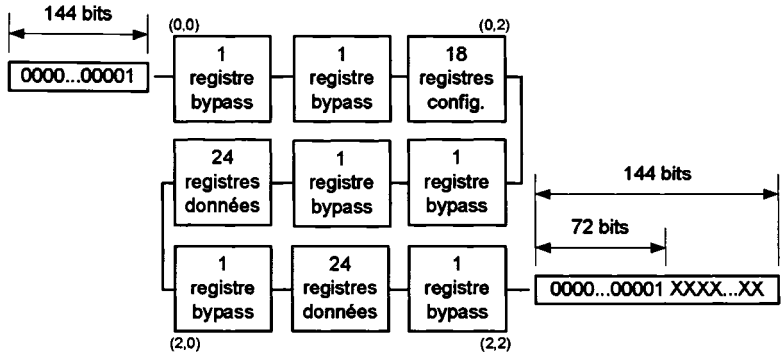


Figure 8 Test de la chaîne de registres avec un chemin alternatif

1.4 Test du bus

Le test du bus est une opération plus longue. Pour chacune des structures choisies, il faut vérifier que les valeurs présentes sur le bus sont les valeurs prévues. Pour permettre de tester la configuration, un code de 4 bits est assigné à chaque cellule. Comme il y a deux orientations, les codes seront inversés sur la verticale. Ainsi, pour une cellule possédant le code 0001 à l'horizontale, le code 1110 lui sera assigné sur la verticale. Les entrées de reconfiguration non connectées des cellules sont fixées à la masse (code 0000).

Comme il y a 9 cellules et que le code 0000 est déjà assigné, il faut s'attendre à avoir deux cellules qui partagent un même code, l'un vertical et l'autre horizontal. Il faudra cependant faire attention que deux cellules ayant un code commun ne soient pas adjacentes. Cela permet d'identifier plus facilement un problème.

Les bus possèdent des inverseurs comme portes de transfert qui viendront modifier les valeurs injectées sur les bus. Un inverseur est présent à chaque interface de sortie horizontale ou verticale d'une cellule. Comme le montre la figure 9, les données d'une cellule seront inversées trois fois sur le bus. Chacune des directions ainsi que toutes les combinaisons de directions par orientation devront être testées.

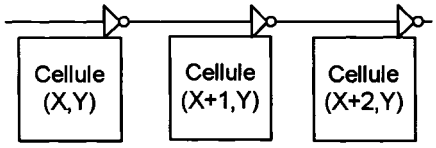


Figure 9 Inverseurs présents sur un bus

1.4.1 Méthodologie

Les codes choisis pour les cellules sont illustrés à la figure 10. Pour chacune des directions, toutes les données de chacune des cellules sont extraites pour les comparer avec les codes prévus. Ces codes sont donnés dans les tableaux I à IV.

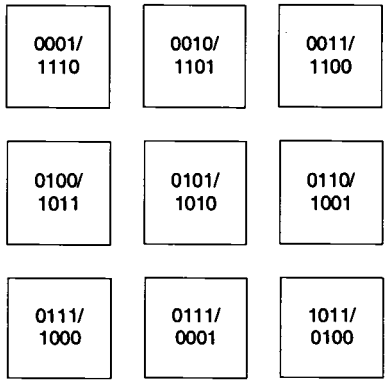


Figure 10 Assignation des codes aux cellules (horizontal/vertical)

Les données ne sont transmises du JTAG au bus de configuration ou du bus des données que lorsque la cellule est en mode EXTEST. Dans ce mode, les données du bus sont également transmises vers les registres de données. Comme les sorties ne sont pas modifiées tant que le TAP n'entre dans sont état UPDATE-DR, il sera possible d'opérer dans ce mode pour effectuer le test.

Le test se fait donc comme suit :

a. Précharger les cellules de leurs codes respectifs;
b. Pour chacune des configurations;
    i. charger la configuration dans les cellules;
    ii. placer toutes les cellules en mode EXTEST;
    iii. charger les codes et observer la sortie.

Pour chacune des configurations, un tableau des résultats ainsi qu'une figure montrant le flux des données de la configuration choisie sont donnés.

Tableau I

Résultats prévus, tests du bus, configuration horizontale:
quartet 1 gauche, quartet 2 gauche; verticale: quartet 1 haut, quartet 2 haut

Cellule	Horizontal		Vertical	
	Quartet 1	*Quartet 2*	*Quartet 1*	*Quartet 2*
(0,0)	1101	0011	0100	1000
(0,1)	1100	1111	0101	0001
(0,2)	0000	0000	0110	0100
(1,0)	1010	0110	0111	1111
(1,1)	1001	1111	1110	1111
(1,2)	0000	0000	1011	1111
(2,0)	0001	1011	0000	0000
(2,1)	0000	1111	0000	0000
(2,2)	0000	0000	0000	0000

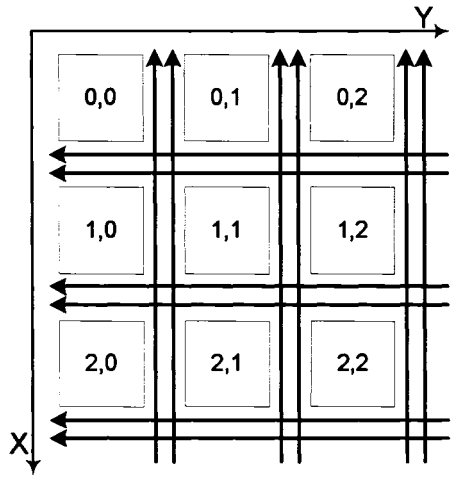


Figure 11 Test du bus, configuration horizontale:
quartet 1 gauche, quartet 2 gauche; verticale: quartet 1 haut, quartet 2 haut

Tableau II

Résultats prévus, tests du bus, configuration horizontale:
quartet 1 droite, quartet 2 gauche; verticale: quartet 1 haut, quartet 2 bas

Cellule	Horizontal		Vertical	
	Quartet 1	*Quartet 2*	*Quartet 1*	*Quartet 2*
(0,0)	0000	0011	0100	0000
(0,1)	1111	1111	0101	0000
(0,2)	0001	0000	0110	0000
(1,0)	0000	0110	0111	0010
(1,1)	1111	1111	1110	0010
(1,2)	0100	0000	1011	0011
(2,0)	0000	1011	0000	0100
(2,1)	1111	1111	0000	0101
(2,2)	0111	0000	0000	0110

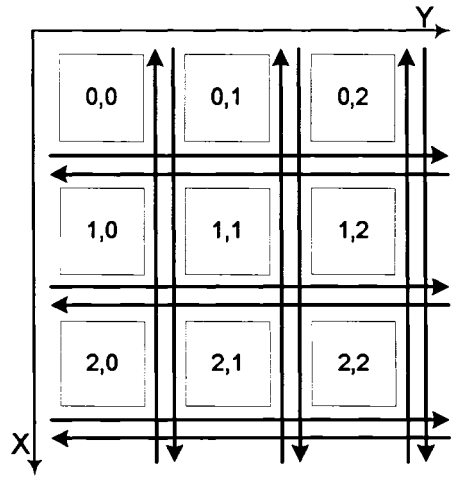


Figure 12 Test du bus, configuration horizontale:
quartet 1 droite, quartet 2 gauche; verticale: quartet 1 haut, quartet 2 bas

Tableau III

Résultats prévus, tests du bus, configuration horizontale:
quartet 1 gauche, quartet 2 droite; verticale: quartet 1 bas, quartet 2 haut

Cellule	Horizontal		Vertical	
	Quartet 1	*Quartet 2*	*Quartet 1*	*Quartet 2*
(0,0)	1101	0000	0000	1000
(0,1)	1100	1110	0000	0001
(0,2)	0000	1101	0000	0100
(1,0)	1010	0000	1111	1111
(1,1)	1001	1011	1111	1111
(1,2)	0000	1010	1111	1111
(2,0)	0001	0000	1110	0000
(2,1)	0100	1000	1101	0000
(2,2)	0000	0001	1100	0000

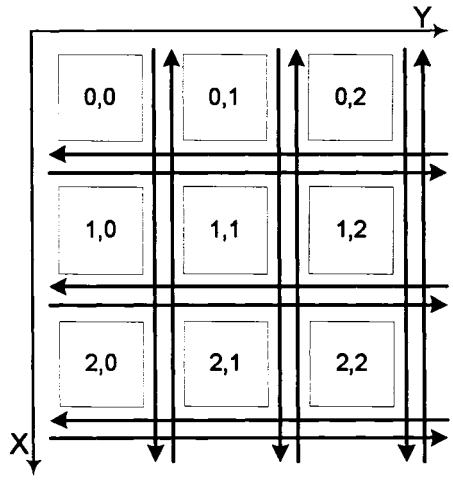


Figure 13 Test du bus, configuration horizontale:
quartet 1 gauche, quartet 2 droite; verticale: quartet 1 bas, quartet 2 haut

Tableau IV

Résultats prévus, tests du bus, configuration horizontale:
quartet 1 droite, quartet 2 droite; verticale: quartet 1 bas, quartet 2 bas

Cellule	Horizontal		Vertical	
	Quartet 1	*Quartet 2*	*Quartet 1*	*Quartet 2*
(0,0)	0000	0000	0000	0000
(0,1)	1111	1110	0000	0000
(0,2)	0001	1101	0000	0000
(1,0)	0000	0000	1111	0001
(1,1)	1111	1011	1111	0010
(1,2)	0100	1010	1111	0011
(2,0)	0000	0000	1110	0100
(2,1)	1111	1000	1101	0101
(2,2)	0111	0001	1100	0110

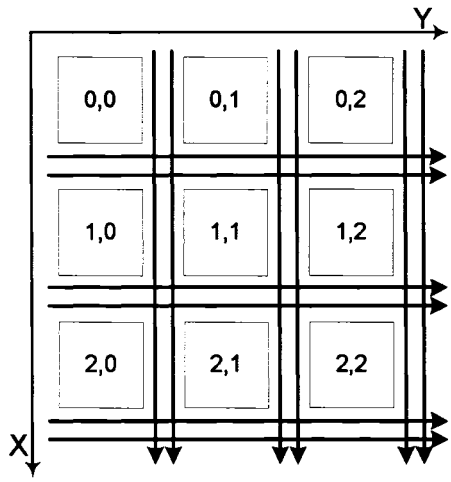


Figure 14 Test du bus, configuration horizontale:
quartet 1 droite, quartet 2 droite; verticale: quartet 1 bas, quartet 2 bas

En suivant cette méthode, il est facile d'isoler un bus défectueux et de trouver un bus de remplacement. Cependant, il n'y a pas assez de cellules à l'intérieur de la puce pour permettre le remplacement d'un quartet ou d'une cellule sans affecter les tests qui suivront. Les tests des structures de configuration devront être faits en tenant compte de

défectuosités si présentes. Seulement après ces tests sera-t-il possible de remplacer une cellule ou un quartet.

1.5 Test des structures de configuration

De façon similaire au test des bus, le test des structures de configuration se fait en modifiant les registres de configuration pour en observer le résultat sur les entrées ou les sorties de chaque cellule. Il faut effectuer le test de tous les chemins possibles et ainsi vérifier tous les mécanismes de configuration de la puce. Les cellules porteront le même code que celui qui leur a été assigné dans le test du bus. Les structures de configuration verticales sont branchées aux bus horizontaux et que les structures horizontales sont connectées au bus verticaux.

La vérification s'effectue cellule par cellule. Pour chacune des configurations, il faut s'assurer que les données soient transmises vers les bons quartets. Les données en réception de toutes les cellules doivent être vérifiées par la suite pour s'assurer qu'il n'y ait aucune cellule en contention avec les mécanismes de reconfiguration.

1.6 Sommaire

Dans une application d'intégration directe sur tranche, la probabilité qu'il n'y ait aucune défectuosité est plutôt faible. Un seul problème sur la tranche signifie que le système au grand complet est inutilisable. Pour palier à ce problème, il faut concevoir des circuits qui permettront de rendre le système tolérant aux défectuosités de manufacture. Cette circuiterie devrait permettre le remplacement des structures défaillantes pour rendre une application WSI possible.

Le démo 4 est un exemple de circuit d'intégration directe sur tranche avec tolérance aux défectuosités. Il est formé de 9 cellules identiques entrecoupées de bus horizontaux et de bus verticaux qui ont chacun deux directions. Les bus sont divisés en sous bus d'une largeur de 4 bits ou quartets. Son système de communication avec le monde extérieur est une interface JTAG (IEEE-1149.1). Par cette interface, il sera possible d'injecter ou d'extraire les données de la puce. Cependant, cette interface ne possède aucun mécanisme de tolérance aux défaillances. Une défectuosité dans sa logique pourrait rendre la puce complètement inutilisable.

Les structures de configuration implantées dans le démo 4 permettent le remplacement d'une cellule ou d'un quartet. Le remplacement d'une cellule peut être partiel tandis que le remplacement d'un quartet est total.

Le test du démo 4 se fait en 3 étapes : le test du contrôleur JTAG, le test des bus et le test des structures de configuration. Le test du contrôleur JTAG est subdivisé en 4 sous étapes : le test du registre d'instruction, le test du registre BYPASS, le test de la chaîne de configuration et le test de la chaîne des données. Le test des bus permet de procéder à la vérification des portes de transfert de sortie des cellules ainsi que des bus eux même. Le test des structures de configuration quant à lui s'assure que toutes les cellules soient en mesure de transférer leurs données vers les autres quartets.

CHAPITRE 2

LE DÉMO 5 : DIAPHONIE ET TEMPÉRATURE

Le 5e démo contient des structures de tests qui représentent différentes situations possibles affectant un bus dans une puce d'intégration directe sur tranche. Ce démo est une mise en oeuvre d'idées permettant de mesurer les effets d'interférences inter-signaux. Il implémente également des systèmes permettant d'imiter l'effet des gradients de température sur de longues lignes de transmission.

2.1 Structure du démo 5

Le démo 5 se présente comme une puce de type « Pin Grid Array » (PGA) de 120 broches dont seulement 84 sont utilisées. L'interface de communication avec l'extérieur est formée de 50 entrées dont 9 analogiques, et de 22 sorties. Les bus de données ont une largeur de 9 bits et la cadence d'opération se situe autour de 125 MHz. Il est capable de capturer les données sur un front montant ou un front descendant de son horloge.

Il se compose de plusieurs blocs internes dont une interface au FPGA, une matrice de configurations de bus, différentes structures de mesures temporelles précises, différentes structures logiques contre les interférences, des éléments thermiques et des diodes de mesure de température. La description détaillée de ces modules dépasse le cadre de ce document. Il est néanmoins nécessaire de les présenter de manière minimale pour permettre la compréhension des méthodologies de test.

2.1.1 Interface FPGA

L'interface FPGA est le port d'entrée des données et est également l'interface par laquelle la structure de test est sélectionnée. Les vecteurs provenant de l'extérieur sont synchronisés sur l'horloge interne. Les signaux de sélection de structure doivent être présents au même moment que les vecteurs de test. Il est possible de sélectionner les données présentes aux registres sur un front montant ou un front descendant.

2.1.2 Matrices des structures

Le démo 5 renferme 4 structures de test : les structures A, B, C et E. Les structures A et C sont identiques mis à part la topologie de leur bus de donnés. Les deux topologies de bus ont été conçues pour imiter des situations qu'il serait possible de rencontrer dans une puce qui possède de longues lignes parallèles. La structure B est une structure à laquelle on a ajouté un synchroniseur qui choisit automatiquement le meilleur front d'horloge pour la capture des données. La structure E est composée d'un système très précis de mesure de délais de propagation.

2.1.2.1 Structure A et C

Tel qu'il a été mentionné plus haut, la seule différence entre la structure A et la structure C est leur bus. Le but visé par le test de ces deux structures est de permettre la mesure d'une précision moyenne du délai de transmission (avec ou sans les effets de diaphonie) dans de longues lignes à l'intérieur d'un circuit imprimé. La figure 15 présente une vue globale des structures A et C et la logique les encapsulant. Elle illustre la segmentation de la logique combinatoire et des registres. Un bus de données de 9 bits, des signaux de sélection de structures (deux bits), ainsi qu'un signal d'un bit contrôlant la fonctionnalité

d'un oscillateur interne proviennent du FPGA. Le bus de données et les signaux de sélection sont échantillonnés par le bloc d'interface FPGA qui va aiguiller les données vers la structure choisie. Les données sont échantillonnées par la structure cible, émis sur le bus de longues lignes et re-échantillonnées à la sortie de ce bus. L'oscillateur interne permet d'ajuster les horloges de la structure. De cette façon, il est possible d'échantillonner les données en sortie du bus sur une phase différente de celle utilisée pour échantillonner les données en entrée. Les données transmises dans le bus sont comparées à l'autre extrémité avec les valeurs d'origine. Lorsqu'une erreur est détectée, elle est immédiatement signalée. Les données sont également retransmises à la sortie pour analyse.

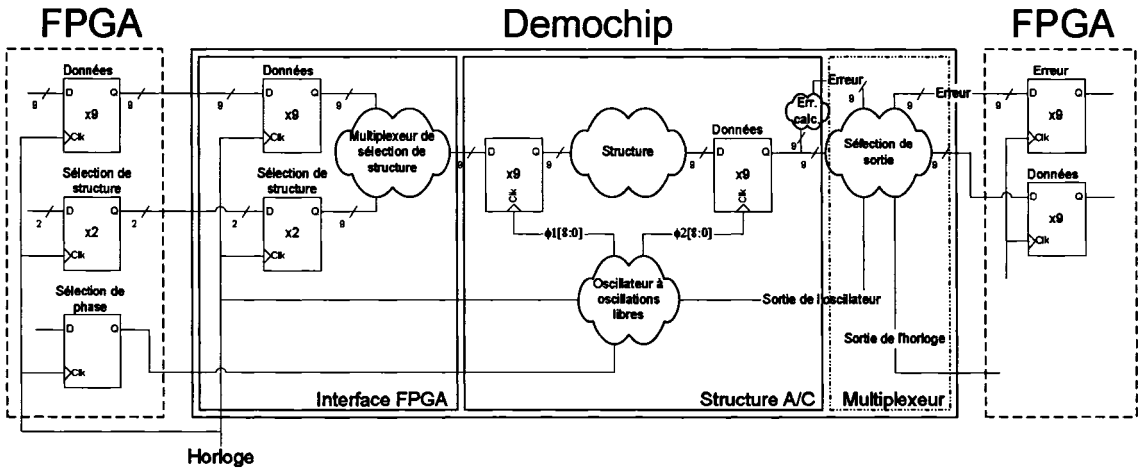


Figure 15 Segmentation des structures A et C

2.1.2.2 Structure B

La structure B permet de vérifier le fonctionnement d'une re-synchronisation sur des données qui traversent le bus. La figure 16 montre la segmentation de la structure B ainsi que la logique qui l'entoure. L'interface FPGA ainsi que le multiplexeur de sortie sont identiques à ceux des structures A et C. Tout comme les structures A et C, les

données sont échantillonnées de part et d'autre du bus à longues lignes. La différence se trouve à la sortie de la structure. Un synchroniseur tente de trouver la bonne phase de capture des données en choisissant entre le front montant et le front descendant de l'horloge. Les données qui entrent dans la structure peuvent être transmises sur différentes phases de l'horloge permettant de forcer une désynchronisation entre les registres d'entrée et ceux en sortie. Les données sont retransmises à la sortie pour une vérification externe. Pour cette structure, le vecteur d'erreur n'indique pas la validité des données mais transporte plutôt les signaux d'horloge interne utilisés pour la synchronisation. Tout comme les structures A et C, des registres de synchronisation sont placés de part et d'autre du bus. Il est possible d'utiliser un signal d'horloge externe pour contrôler les registres de réception pour permettre d'effectuer une synchronisation manuelle.

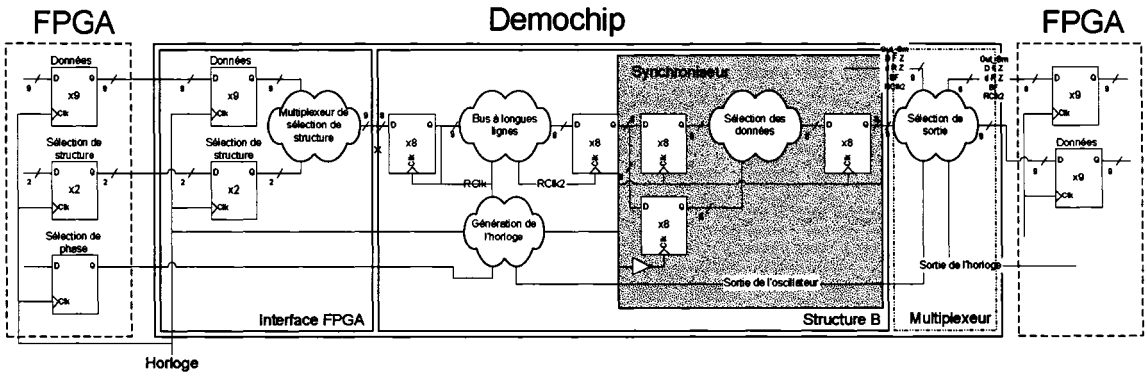


Figure 16 Segmentation de la structure B

2.1.2.3 Structure E

La structure E permet un test de mesure précise de l'effet de diaphonie. La figure 17 illustre la division de la structure E et la logique environnante. L'interface FPGA et le multiplexeur y sont toujours représentés. Les données sont toujours échantillonnées de part et d'autre du bus à longues lignes. La structure E diffère des autres structures du

fait qu'elle contient un vernier. Ce dispositif de mesure très précis est capable de faire des différences de l'ordre des picosecondes (10^{-12} secondes). Les vecteurs sont transmis à travers la structure et le résultat est comparé avec ce qui a été envoyé à l'origine. Chacun des signaux de données peut être placé sur une phase différente de l'horloge. Le résultat ainsi que les bits de détection d'erreur sont retransmis à la sortie. Étant capable d'effectuer des mesures temporelles très précises, cette structure permettra de connaître le délai de chacun des signaux transmis sur le bus. Il sera donc possible de mesurer les effets d'interférences dans un très long bus.

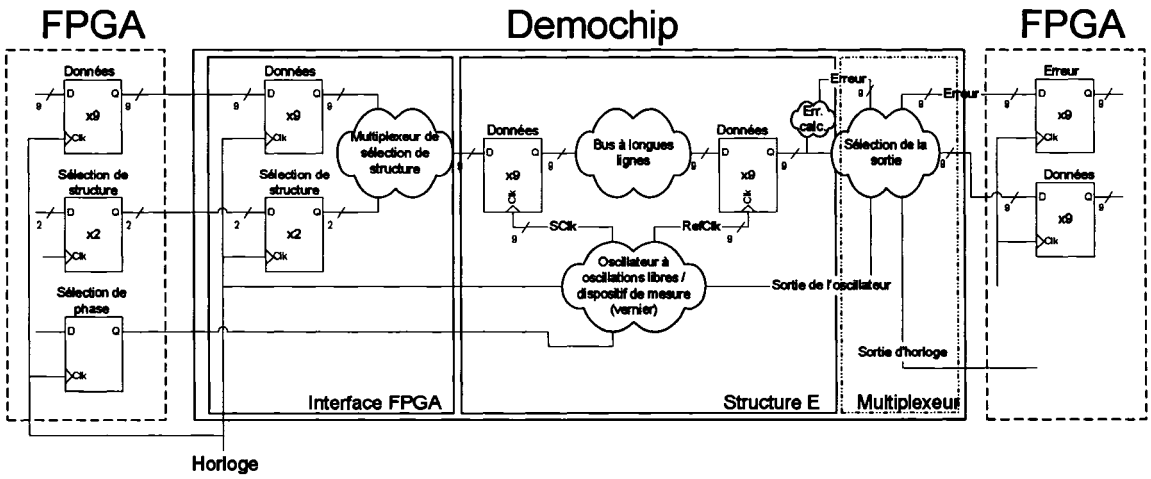


Figure 17 Segmentation de la structure E

2.1.3 Les éléments thermiques

Le démo 5 ne contient pas assez de logique pour augmenter sa température de façon considérable. Pour simuler une activité importante et soutenue, des éléments chauffants sont incorporés dans les structures pour permettre d'augmenter la température sur une partie de la puce. Ainsi, il offre une possibilité de tester l'effet de gradients de température sur la vitesse des signaux. Les éléments thermiques couvrent plus de la moitié des bus à longues lignes. Ils ne devraient jamais tous être maintenu actifs. La

consommation en courant serait excessive et les structures chauffantes pourraient être endommagées. Ces éléments thermiques ont été conçus pour être actifs de façon intermittente. Leur contrôle provient du FPGA.

2.1.4 Diodes de mesure de température

Les diodes de mesure de température sont des transistors dont la grille et le drain sont court-circuités pour simuler une diode. Cette diode agira comme limiteur de courant sensible à la température de la puce. Des circuits intégrés existants permettent la mesure de la température des puces d'aujourd'hui comme certains processeurs et certains FPGA. Le processeur Pentium® de la compagnie Intel serait un bon exemple de puce possédant une diode de mesure de température Cependant, la technologie de fabrication des transistors n'est pas nécessairement celle utilisée dans le démo 5. Pour effectuer la capture de la température, une calibration sera nécessaire. La capture de la température se fera donc en comparant le courant provenant d'une puce démo au repos avec le courant de la puce en fonction.

2.2 Méthodologies de tests

Comme il a été mentionné plus haut, le démo 5 a été conçu pour fonctionner à une fréquence d'environ 125 MHz. Avant de commencer les tests, il faut s'assurer que l'horloge d'entrée du démo 5 soit propre et stable. Les mesures faites par les différentes structures sont de l'ordre des picosecondes. La gigue (« jitter ») diminue la précision des mesures et ainsi fausse les résultats.

Mise à part les structures A et C, chacune des structures possède sa propre procédure de tests, les structures A et C possédant une procédure identique. Notons que certains signaux de contrôle ne sont pas échantillonnés par des registres et par conséquent ne

doivent changer que lors d'un changement de test. Il faut également laisser suffisamment de temps à ces signaux pour s'assurer que les valeurs soient stables. Malgré que les signaux de sélection des structures soient synchrones, il est plus prudent de ne pas changer leur état pendant les tests. Il serait trop facile de mélanger la sortie d'une structure avec le test d'une autre ou tout simplement d'assigner des signaux asynchrones aux mauvaises structures.

Les tensions fournies aux entrées analogiques doivent être propres et stables. Une forte variation de ces références introduit de l'imprécision dans les mesures. Ces tensions sont utilisées comme références pour les mesures de délais et pour contrôler les délais induits volontairement à l'interne sur certains signaux d'horloge. Ces délais augmentent lorsque la tension diminue. Il existe une tension minimale appelée tension de seuil sous laquelle les délais deviennent infinis. Cette tension est étroitement liée à la fabrication de la puce et doit être trouvée de façon empirique. Il a été établi par simulation que cette tension se situe entre 0.7 et 0.8 volt.

2.2.1 Structures A et C

Avant de démarrer les tests, il faut d'abord sélectionner la structure à tester. Le choix du front d'échantillonnage des registres d'entrée a, pour le moment, peu d'importance. Il est important d'avoir une tension stable sur les lignes analogiques. Un test rapide des structures de mesures temporelles permettra de voir si la tension de seuil est atteinte: l'horloge en sortie devrait osciller.

Le test s'effectue avec des vecteurs provenant d'un générateur de nombres pseudo aléatoires. Les données reçues doivent correspondre aux données envoyées. Il ne doit y avoir aucune erreur à la sortie pour une tension de référence supérieure à la tension de seuil. Si une erreur survient, il se peut que la phase des registres d'entrée soit incorrecte.

Il faut alors modifier la phase d'échantillonnage. Les vitesses de propagation des signaux sur la carte de test peuvent affecter les performances. Si une telle erreur survient, des tests aidés d'un oscilloscope sont conduits pour vérifier que toutes les données arrivent à temps aux entrées de la puce.

Il se peut cependant qu'une erreur ne soit pas signalée mais que les vecteurs reçus soient différents de ceux qui sont transmis. Dans ce cas, il faut essayer de changer de phase d'acquisition des données au FPGA. Étant donné qu'il n'y a pas de registres en sortie, des délais supplémentaires de multiplexage entrent en jeu et peuvent créer ce comportement.

Les tests sont effectués en rafales. Pour chaque tension analogique, un nombre important de vecteurs doit être transmis. À 125 MHz, 100 millions de vecteurs prend un peu moins d'une seconde à traverser la structure. Les données sont échantillonnées sur une phase ($\phi 1$) de l'horloge avant d'être transmises dans le bus à longues lignes. À la sortie, la tension de référence permettra de sélectionner la phase d'échantillonnage des registres à l'autre extrémité du bus ($\phi 2$). Les tensions de référence devraient être augmentées par petits incréments jusqu'à l'apparition d'erreurs. Sitôt l'erreur signalée, la mesure du délai de propagation peut être mesurée.

2.2.2 Structure B

Après avoir sélectionné la structure B, il faut placer les tensions analogiques de référence à la tension de seuil pour que les structures de mesures temporelles puissent fonctionner efficacement. La phase de capture des données à l'entrée ne devrait pas être différente de ce qui a été utilisé lors des tests des structures A et C.

Le vecteur d'erreur possède déjà toutes les informations relatives à la sélection de phase du synchroniseur. Il est possible de les visualiser à l'aide d'un analyseur logique ou d'un oscilloscope rapide. Des vecteurs provenant d'un générateur de nombres aléatoires peuvent maintenant être transmis. Le FPGA devra faire sa propre vérification pour détecter les erreurs. La phase d'acquisition des données à l'intérieur du FPGA ne devrait pas différer de celle utilisée pour les structures A et C.

Si des erreurs sont détectées, il faut alors passer en mode manuel pour savoir si la structure n'est pas endommagée. Sur l'une des deux alternances, les erreurs devraient disparaître faute de quoi la structure B est déclarée inutilisable.

Pour chacune des phases sélectionnées, des données pseudo-aléatoires sont transmises à travers la structure. Il est essentiel d'utiliser un grand nombre de vecteurs pour effectuer les tests. Tout comme pour le test des structures A et C, les données sont échantillonnées sur une phase différente de part et d'autre du bus (RClk1 et RClk2) mais cette fois-ci, pour effectuer une désynchronisation. Immédiatement placé après les registres de désynchronisation, le synchroniseur devrait toujours trouver la bonne phase pour échantillonner peu importe la désynchronisation imposée. Si une erreur survient, c'est que la limite du synchroniseur est atteinte. En revenant d'une phase en arrière, il est maintenant possible d'utiliser la tension de référence pour déterminer avec plus de précision le délai minimal supporté par le synchroniseur.

2.2.3 Structure E

Les causes de la diaphonie sont les transitions rapides des signaux dans une configuration de lignes rapprochées. Lorsqu'un signal change brusquement d'état, un courant et une tension sont induits dans les lignes avoisinantes. Les signaux ainsi affectés seront appelés victimes tandis que le signal changeant est l'agresseur. Cet effet

est dû à deux phénomènes : l'inductance mutuelle et la capacitance mutuelle. Deux signaux qui produisent une transition dans le même sens (transition positive ou transition négative) subiront une interférence constructive. Si deux signaux produisent une transition différente, ils subiront une interférence destructive. La figure 18 illustre la propagation d'un signal ne subissant aucune interférence. À la figure 19, le signal V subit une interférence constructive sur son front montant; ce front possède une transition plus rapide. Sur la figure 20, le front descendant du signal V subit une interférence destructive ralentissant sa transition.

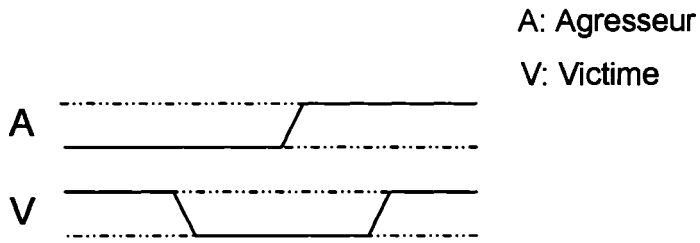


Figure 18 Propagation d'un signal sans interférence

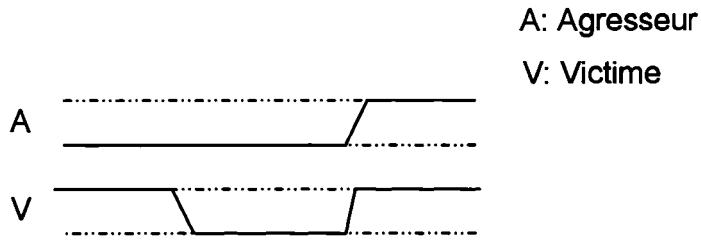


Figure 19 Propagation d'un signal avec interférence constructive

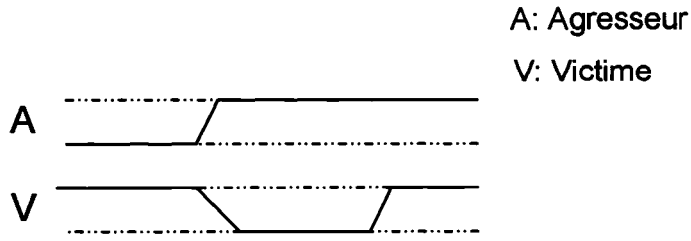


Figure 20 Propagation d'un signal avec une interférence destructive

Le but des tests de la structure E est de vérifier l'effet de la diaphonie sur les temps de montée et le temps de descente des transitions sur un signal. Contrairement aux autres structures, le test de la structure E ne peut être fait par force brute. Étant très flexible, cette structure ouvre un trop grand éventail de possibilités et le test complet prendrait plusieurs mois. Le test de la structure E doit donc être fait de façon méthodique et les vecteurs de tests devront être choisies. Le présent document ne se concentrera que sur certains vecteurs représentants, selon l'auteur, les cas les plus intéressants.

Après la sélection de la structure E, on ajuste la tension de référence à la tension de seuil. Le test des structures de mesures temporelles devrait être fait. La puce doit être remise dans l'état présenté ci-après au début de chacun des tests.

Il faut tout d'abord sélectionner les phases d'émission et de réception. Toutes les phases des registres d'émission doivent être placées sur le même étage de la ligne à délais. La phase des registres de réception doit être placée au même étage que ceux de l'émission. Le déphasage initial doit être minimal. Le test préliminaire de la structure peut se faire avec des vecteurs provenant d'un générateur de nombres pseudo-aléatoires. Ces vecteurs sont envoyés dans la structure et le résultat est acquis à la sortie. Aucune erreur ne devrait être détectée.

Comme il l'a été mentionné, seuls les cas les plus intéressants seront évalués. Les tests seront catégorisés comme suit : position de la victime, position des agresseurs et le nombre d'agresseurs.

2.2.3.1 Position de la victime

Les tests de cette catégorie se concentrent sur le déplacement de la ligne qui subit l'interférence. Dans ce cas, toutes les autres lignes seront des agresseurs. Les vecteurs utilisés dans ces tests peuvent être vu comme une rotation du vecteur de base: *011111111*.

2.2.3.1.1 Interférence destructive

Pour les tests, toutes les phases des registres d'émission sont identiques et constantes. Seule la phase des registres de réception est variable. La structure même du démo demande que pour tous les tests, les vecteurs soient envoyés un très grand nombre de fois pour une même phase. Comme ce sont les transitions qui causent les interférences, deux tests différents peuvent être faits en alternance successive. Par exemple, le changement du vecteur *111101111* à son complément *000010000* attaque une transition positive (au centre) avec des transitions négatives. De même, une transition du vecteur *000010000* vers son complément *111101111* attaque une transition négative avec des transitions positives. Encore une fois, l'ordre du million de répétitions semble raisonnable. La figure 21 illustre le test lors d'une transition du vecteur *111101111* vers le vecteur *000010000* ; les agresseurs A*n* devraient influencer la victime V.

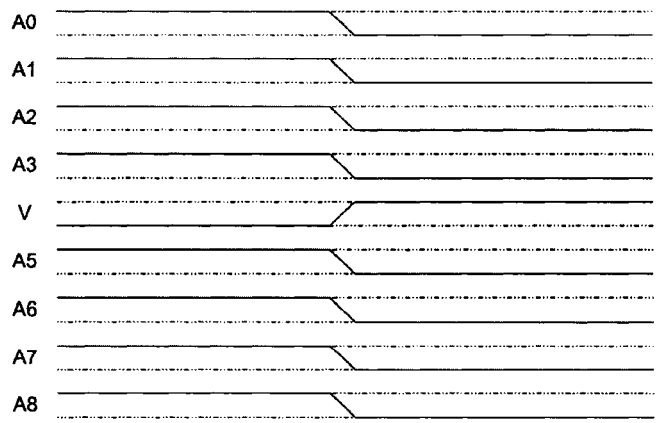


Figure 21 Test d'interférence destructive de la position de la victime

Pour chacun des vecteurs, la phase des registres de réception est balayée du dernier étage de la ligne à délais vers le premier pour trouver le point de défaillance. Lors de l'apparition d'une erreur, des modifications sur la tension de référence permettra de trouver avec plus de précision le temps de propagation. Après avoir ajusté la ligne à délais sur la phase qui précède l'erreur, la tension est incrémentée jusqu'à défaillance. Le vernier pourra alors être utilisé.

Le vernier est ajusté pour avoir une période de plus en plus courte. Un nombre suffisant de vecteurs doit être envoyé pour chacune des modifications du vernier. Lorsqu'une erreur survient, le délai peut être mesuré. Il est possible que les tests d'interférences constructives et les tests d'interférences destructives ne produisent pas des erreurs à la même différence de phase. Il faudra alors reprendre le test d'interférences qui n'a pas produit d'erreur là ou il a été laissé et continuer jusqu'à ce que ce type d'interférence n'arrive à un résultat erroné.

2.2.3.1.2 Interférence constructive

Le test de l'interférence constructive se fait de façon très semblable au test d'interférence destructive à l'exception que la victime doit être retardée en rapport avec

les agresseurs. En effet, si la victime et les agresseurs subissaient la même transition au même moment, l'effet d'interférence constructive ne pourrait être détectée.

Pour ce test, les vecteurs *000000000* et *111111111* sont utilisés. Il est alors possible de mesurer les effets d'interférences constructives tant sur une transition montante que descendante. La victime est caractérisée par un léger retard par rapport aux autres signaux. Le retard doit, lors d'une interférence suffisante, disparaître et tous les signaux doivent arriver aux registres de sortie en même temps. Tel que vu à la section 2.1.2.3, chacun des signaux de la structure 5 permet l'échantillonnage d'un signal sur une phase différente des autres. La figure 22 illustre le test d'interférence constructive sur une transition descendante pour le signal central (V).

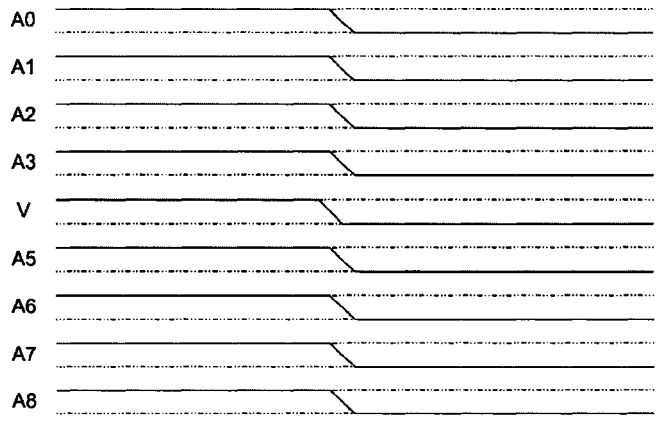


Figure 22 Test d'interférence constructive de la position de la victme

La mesure est effectuée de la même façon qu'il a été vu dans la section 2.2.3.1.1. Ce test peut être fait pour plusieurs phases de départ permettant de trouver le délai minimal pour lequel la victime ne sera pas influencée par ses agresseurs ainsi que la différence de phase qui offre une interférence maximale.

2.2.3.2 Position de l'agresseur

Comme son nom l'indique, ce test s'effectue avec une victime immobile et un agresseur mobile. La position de la victime sera la première ligne de donnée (ligne 0) ce qui permet le test d'une ligne éloignée d'une distance de 8 signaux. L'agresseur prend tour à tour les positions 1 à 8.

2.2.3.2.1 Interférence destructive

Il est également possible d'effectuer les tests des interférences destructives sur front montant et sur front descendant en alternance tel que vu dans la section précédente. La génération des vecteurs de tests d'interférences constructives peut être vue comme un décalage vers la droite d'un bit partant de la position 8 et allant vers la position 0. Le tableau V présente les vecteurs à utiliser pour ce test. Ces vecteurs alternés avec le vecteur *100000000* permettent la génération des transitions nécessaires. Le test est fait en suivant les mêmes étapes que celles qui ont été vu dans le test de la position de la victime.

Tableau V

Vecteurs de tests, position de l'agresseur; interférence destructive

Interférence destructive	
Position	*Vecteur*
A1	010000000
A2	001000000
A3	000100000
A4	000010000
A5	000001000
A6	000000100
A7	000000010
A8	000000001

2.2.3.2.2 Interférence constructive

La victime possède encore un retard par rapport aux autres signaux. Ce signal est choisi comme étant celui de la première ligne de donnée comme le test précédent. En utilisant la différence de phase maximale trouvée lors du test de la position de la victime, il est possible de mesurer l'impact de la distance entre deux signaux sur la diaphonie à l'intérieur d'une puce. Le vecteur *000000000* utilisé en alternance avec ceux du tableau VI permettra de mesurer les effets d'interférence constructive sur les transitions montantes et descendantes.

Tableau VI

Vecteurs de tests, position de l'agresseur; interférence constructive

Interférence constructive	
Position	*Vecteur*
A1	110000000
A2	101000000
A3	100100000
A4	100010000
A5	100001000
A6	100000100
A7	100000010
A8	100000001

2.2.3.3 Nombre des agresseurs

Encore une fois, ce sont les vecteurs plutôt que les phases des émetteurs qui sont modifiés. Cette fois-ci, la victime sera placée au centre du bus et les agresseurs sont tout autour. Un par un, en alternance de chaque coté de la victime, les agresseurs sont enlevés de l'intérieur vers l'extérieur de la structure. La raison de ne pas procéder de l'extérieur vers l'intérieur en est une de précision. Les agresseurs les plus près influencent de façon beaucoup plus importante que ceux en périphérie ainsi l'effet d'enlever les agresseur de l'extérieur vers l'intérieur pourrait passer inaperçu.

2.2.3.3.1 Interférence destructive

Les tests d'interférences destructives sur front montants et front descendants peuvent encore être faits en alternance. Pour chacun des tests, il faut réinitialiser le bus des données avec la valeur qui permettra d'obtenir l'effet voulu. Par exemple, le vecteur de test d'interférences constructives *111001111* requiert une valeur préchargée de *000010000*. Les tests d'interférences destructives sont automatiquement faits lors du préchargement du bus de données. En effet, il est facile de remarquer que la transition obtenue par les vecteurs du tableau VII et de la valeur de préchargement cause des transitions sur les mêmes agresseurs et la même victime. Le test se fait de la même façon que ce qui a été vu à la section 2.2.3.2.

Tableau VII

Vecteurs de tests, nombre d'agresseurs, interférence destructive

Interférence destructive	
Position	*Vecteur*
A4	111101111
A4	111001111
A4	111000111
A4	110000111
A4	110000011
A4	100000011
A4	100000001
A4	000000001

2.2.3.3.2 Interférence constructive

Pour effectuer les tests d'interférence constructive, les vecteurs présentés au tableau VIII sont utilisées. La valeur de préchargement pour ce test sera le vecteur nul: *000000000*. Ces valeurs sont à un bit près identique aux vecteurs présentés précédemment. Étant la victime, le bit central de chaque vecteur est échantillonné en retard Les mesures sont toujours effectuées tels qu'il a été discuté dans les sections précédentes.

Tableau VIII

Vecteurs de tests, nombre d'agresseurs, interférence constructive

Interférence constructive	
Position	*Vecteur*
A4	111111111
A4	111011111
A4	111010111
A4	110010111
A4	110010011
A4	100010011
A4	100010001
A4	000010001

2.2.4 L'utilisation des éléments thermiques

Les tests vus dans les sections 2.2.1, 2.2.2 et 2.2.3 peuvent tous être refaits avec des éléments chauffants actifs. En effectuant les mêmes tests, il est alors possible de comparer les résultats avec et sans gradient. Il est important de rappeler que les éléments thermiques ne doivent jamais rester actifs pendant une trop longue période de temps pour éviter d'endommager la puce. La température du démo 5 devrait toujours rester sous observation lors de l'utilisation de ces éléments. Dans le cas d'une chaleur excessive, ces éléments doivent être immédiatement arrêtés.

2.3 Sommaire

Le démo 5 offre un gabarit de tests permettant d'analyser le comportement de signaux parcourant de grandes distances. Il incorpore 4 structures de tests. Les structures A et C sont très semblables et ont pour but le test des délais de propagation des signaux dans de longues lignes. La structure B propose une méthode de synchroniser la réception des données sur le bon front de l'horloge. La structure E permet de tester la diaphonie.

Pour permettre l'étude des effets de gradients dans une puce d'intégration à la tranche, le démo 5 est muni d'éléments thermiques qui permettent d'élever la température de la puce. Il est alors possible de créer un écart de température assez important pour mesurer les effets de gradients sur de longues lignes.

CHAPITRE 3

BESOINS

Dans les chapitres précédents, il a été question des deux principales puces que la carte de test doit vérifier. Ces deux puces ont des besoins propres en nombre d'entrées/sorties, en cadence d'horloge, en stabilité des signaux, etc. La carte de test doit accommoder les besoins de toutes les puces tant présentes que futures. Les besoins présents ont été définis en rapport avec les exigences requises du démo 5. Le 4^{e} démo ne possède pas vraiment de signaux qui requièrent une vitesse élevée et son nombre de signaux digitaux est largement inférieur à celui requis pour le démo 5.

Comme il a été vu au chapitre précédent, le démo 5 est monté sur un boîtier de type PGA de 120 broches, dont 94 broches sont utilisées: 63 digitales, 17 analogiques et 14 d'alimentation. Les lignes digitales doivent supporter la vitesse nominale de 125MHz. Sur les 17 lignes analogiques, 9 lignes sont utilisées comme tensions de référence et les 8 restantes transportent un courant. Il n'est pas exclu que les prochains démos aient une dimension différente et qu'ils nécessitent plus signaux.

Les tensions de référence doivent être des plus stables possibles. Selon des simulations faites sur les générateurs internes de fréquence, un changement d'environ 0.05 volts dans les tensions les plus basses peut faire varier la fréquence d'une magnitude de plus de 20MHz (cf. tableau IX). Ces lignes analogiques devraient donc être filtrées pour avoir le moins de bruit possible.

Tableau IX

Fréquence d'oscillation des lignes à délais

Tension	*V*	0.74	0.75	0.8	0.85	0.9	0.95	1	1.05	1.1	1.15	1.2	1.25	1.3	1.35
Fréquence	*MHz*	31	39	64	86	103	118	129	138	145	149	153	155	158	160
Tension	*V*	1.4	1.45	1.5	1.55	1.6	1.65	1.7	1.75	1.8	1.85	1.9	1.95	2	
Fréquence	*MHz*	161	163	164	165	166	167	168	169	170	170	171	171	172	

Pour l'alimentation des démos, une tension de 1.8V est requise. Cette alimentation doit être assez forte pour subvenir aux besoins en courant de la puce. Comme l'alimentation est de tension peu élevée, elle doit être filtrée d'avantage. La pollution par les interfaces de communication ou des interfaces de tests branchés sur la carte ne pourrait être tolérée.

La carte de tests doit être capable d'effectuer la capture de la température des démos. Cette information est transmise sous forme de courant. Idéalement, il faudrait que la carte soit capable elle-même d'arrêter tous les tests en cours lors d'une surchauffe de la puce.

3.1 Besoins futurs

La plus grande problématique de la carte est qu'elle doit permettre le test des prochains démos. Ces puces n'ont pas encore de spécifications et les tests qu'elles conduiront ne sont encore que de vagues idées. Cependant, des ports d'expansion permettraient de combler certains besoins futurs. Tel qu'il a été mentionné plus haut, le nombre de signaux pourrait être supérieur à celui du 5^{e} démo. Ces signaux pourraient également être poussés à des vitesses supérieures. Il faut donc préparer une spécification qui permette d'ajouter un certain nombre de signaux digitaux et qui puisse facilement modifier la fréquence d'opération des tests.

Les prochains démos pourraient contenir des portions analogiques plus ou moins rapides pouvant requérir une génération de signaux. Des convertisseurs digitaux/analogiques

permettraient de générer ces signaux à même la carte. Il pourrait également être possible de devoir effectuer la capture de certaines formes d'ondes. L'ajout de convertisseurs analogiques/digitaux serait donc un atout intéressant.

Une autre possibilité envisagée est d'effectuer un test d'interaction entre deux démos. Il serait, par exemple, possible de simuler la transmission/réception d'un système de communication. La carte devrait permettre le branchement de deux démos ou plus sur la même carte et de prévoir un moyen de passer l'information de l'une à l'autre.

3.2 Spécifications préliminaires de la carte

La spécification de la carte proposée plus bas est en grande partie composée d'éléments nécessaires au test des démos actuels mais en comporte également plusieurs jugés désirables pour les démos futurs.

La spécification préliminaire est comme suit:

a. Conception autour d'un FPGA
    i. Le FPGA doit être suffisamment rapide pour effectuer facilement des tests à une fréquence de 125MHz;
    ii. Le FPGA doit posséder suffisamment de ports d'entrées/sorties pour permettre le branchement d'au moins deux démos (un minimum de 220 broches) sans multiplexage.
b. Une horloge programmable
    i. L'horloge qui alimente le FPGA et le démo doit permettre une plage de fréquence allant d'environ 1MHz à une fréquence supérieure à 125MHz; La gigue sur le signal d'horloge doit être inférieure à 300ps;
c. Sources de tension programmables

i. La carte doit être capable de fournir 9 tensions programmables aux démos;
ii. Ces tensions doivent être très stables.

d. Acquisition de données
   i. La carte doit permettre des acquisitions de données tant numériques qu'analogiques;
   ii. Deux convertisseurs numériques et deux convertisseurs analogiques au minimum seront requis;
   iii. Les convertisseurs doivent être présent en deux catégories : rapides et précis;
   iv. La carte doit posséder un capteur de température avec référence externe.

e. Mémoire
   i. La carte doit posséder une certaine quantité de mémoire pour permettre un stockage de vecteurs de tests ainsi que des résultats;
   ii. Cette quantité dépend du nombre de tests à effectuer en ligne ainsi que de la vitesse de communication avec l'extérieur;
   iii. Une mémoire de type flash pourrait être utile pour le stockage à moyen ou long terme de vecteurs ou de résultats.

f. Interfaces
   i. Un port JTAG doit être présent pour la programmation du FPGA ainsi que le diagnostic;
   ii. La carte de test doit posséder au moins une interface de communication autre que le JTAG. Cette interface pourrait être du type EIA-RS232, FireWire, Ethernet ou encore USB;
   iii. De nombreux points de tests doivent être présent pour permettre le branchement d'appareils d'analyse comme des analyseurs logiques. Idéalement, chaque signal de contrôle devrait posséder son point de test.

g. Alimentation
   i. Une seule tension d'entrée doit être fournie à la carte;

ii. Une protection indépendante sur chacune des tensions ainsi qu'une protection générale doit être présente.

h. Expansion

i. La carte doit permettre une expansion future. Elle doit si possible, être légèrement plus performante que la spécification en prévision de test ultérieurs.

3.3 Les cartes existantes

Une recherche fut entreprise dans le but de trouver une carte de développement rencontrant les besoins énoncés ci-haut. Cette recherche fut effectuée en sachant que certains besoins spécifiques, comme l'acquisition de données, ne pouvaient être tous comblés. Les cartes disponibles sont généralement conçues pour des applications de traitement où les données sont déjà numérisées. Par contre, il devait être possible d'ajouter les fonctionnalités manquantes en utilisant les ports d'extension disponibles sur les cartes.

Les principaux critères de recherche ont été les points a, b et e, c'est-à-dire le nombre et la vitesse des ports d'entrées/sorties, une horloge flexible et facilement modifiable, ainsi que la quantité et la vitesse de la mémoire disponible. Cette recherche à conduit à une liste de cartes disponibles dont seuls les plus intéressantes seront décrites ci-après.

3.3.1 Alpha Data

Alpha Data a mis sur le marché deux cartes basées sur le Virtex-II: la ADM-XRC-II/4000-4CES et la ADM-XRC-II/6000-4CES. Ces cartes sont conçues autour d'un XC2V4000 ou d'un XC2V6000 sur quelques grades de vitesse (4, 5 et 6). Cependant,

lors de la recherche, seul le grade 4 était disponible et aucune date n'était connue pour la sortie des grades supérieurs.

Elles possèdent 146 ports d'entrées/sorties sur deux connecteurs de série QSH de la compagnie Samtec et 64 sur un connecteur de type SCSI Pn4 pour un total de 210. Ces cartes possèdent un oscillateur programmable de 0 à 100MHz; les fréquences supérieures pourraient être générées par le FPGA, ou encore être fournies par les connecteurs d'expansion. L'oscillateur de référence de 14MHz est caractérisé d'une gigue de 10ps. Leur capacité de mémoire est de 6 mégaoctets sous forme de mémoire statique de type ZBT. Une mémoire de type FLASH de 8 mégabits est disponible. Ces cartes ne possèdent pas d'interface de communication vers l'extérieur. Il faudrait en implémenter un ce qui réduit davantage le nombre d'entrées/sorties disponibles.

Les produits de Alpha Data ne remplissent pas les spécifications soumises de plusieurs façons :

a. cette carte ne possède pas le nombre requis de ports d'entrées/sorties compte tenu du manque d'une interface de communication;
b. l'oscillateur programmable n'est pas assez rapide;
c. les connecteurs d'expansion ne sont pas adéquats. Les deux connecteurs QSH sont placés dans le même coin et le connecteur Pn4 est un connecteur pour un câble de type SCSI qui ne pourrait pas servir comme port d'expansion.

3.3.2 SIDSA

La compagnie SIDSA possède une carte de développement pour le processeur ARM: CARMeN (Core ARM EmulatioN). CARMeN est conçue autour du Virtex-E 2000 de grade 6. Les grades supérieurs 7 et 8 n'étaient pas encore disponibles. Elle possède un maximum de 300 ports d'entrées/sorties et est cadencée à une fréquence fixe de 30MHz.

Il serait possible de multiplier cette horloge en utilisant les DLL du Virtex-E. Ces DLL sont caractérisés d'une gigue de ±60ps.

CARMeN possède une capacité de 2 mégaoctets de mémoire statique de 15ns et une mémoire de type FLASH de 8 mégabits. Il est possible de lui ajouter de la mémoire dynamique par un connecteur DIMM de 168 broches. Elle possède une multitude de ports de communication: EIA-RS232, USB, CAN, SmartCARD, IrDA, PCI, PCMCIA et MutliICE.

La carte CARMeN ne peut être utilisée pour les raisons suivantes :

a. cette carte est basée sur le Virtex-E. Tel qu'il a été mention plus haut, le grade 7 de ce FPGA permet en moyenne une cadence de 130MHz. Nous sommes presque certain qu'avec un grade 6 nous n'obtiendrons pas la performance requise;
b. la vitesse maximale de l'oscillateur est de 30MHz. Pour obtenir les fréquences supérieures, il faut utiliser les DLL internes du FPGA. Les DLL du Virtex-E ne peuvent que multiplier par 2 ou diviser par certains facteurs prédéfinis;

3.3.3 Catalina research incorporated

Cette compagnie se spécialise dans le traitement du signal. Les cartes conçues par Catalina research sont très performantes. Il y a présentement deux cartes basées sur le Virtex : la Chameleon VME et la Cheetah VME.

La Chameleon VME est une carte de 4 Virtex-E cadencés à une vitesse fixe de 266MHz. Elle possède 28 mégaoctets de mémoire dynamique et un total de 56 mégaoctets est possible.

La Cheetah VME est une carte spécialisée dans les transformées de Fourier rapides (FFT). Elle est conçue autour d'un Virtex-E d'une vitesse d'opération fixe à 120MHz. Elle possède deux modules à FFT et une mémoire de 28 mégaoctets de mémoire statique au maximum.

Outre leur trop grande capacité de traitement, ces deux cartes ne peuvent être utilisées étant donné leur format compact PCI (cPCI). Aucun port d'entrées/sorties des FPGA n'est disponible à l'extérieur de cette interface.

3.3.4 ErSTElectronics

La carte EVALXCV/XCVE est conçue autour d'un Virtex ou d'un Virtex-E. Elle possède deux oscillateurs fixes sur support qui permet un changement simple et rapide de fréquence. Elle permet également de fournir une horloge externe par 3 connecteurs prévus à cet effet. Elle possède une capacité de mémoire de type ZBT de 9 mégabits. La carte permet l'accès à 146 ports d'entrées/sorties sur 3 connecteurs.

Elle ne possède pas de ports dédiés à une communication externe réduisant d'avantage le nombre de ports disponibles. Outre le manque de ports d'entrée/sortie, la EVALXCV/XCVE ne peut être utilisée de par la taille de sa mémoire.

3.3.5 Insight

Insight possède une carte de développement basée sur le Virtex-II : la INS2044-VIRTEXII. Cette carte supporte du XC2V40 jusqu'au XC2V1000 dans un format de 256 broches. Ce boîtier peut supporter un maximum de 172 ports d'entrées/sorties.

Le FPGA possède 4 entrées d'horloges: une de 25MHz, une de 100MHz et deux externes. La carte est capable d'accepter jusqu'à 32 mégaoctets de mémoire dynamique

de type DDR. Elle possède ses propre régulateurs de 3.3V, 2.5V, 1.8V et 1.5V et une interface RS232.

Cette carte ne pourrait être utilisée puisque son nombre d'entrées/sorties est trop bas. L'ajout de la mémoire enlève au FPGA des broches qui pourraient autrement être utilisées pour le test.

3.3.6 Nallatech

Il existe deux cartes intéressantes chez Nallatech: la Ballynuey3 et la Benera. La carte Balleynuey3 est une plateforme de développement de haute performance. Composée de deux FPGA, un Virtex-II et un Spartan-II, elle se spécialise dans le traitement de l'image. Elle possède de 4 à 32 mégaoctets de mémoire statique de type ZBT mais ne possède que 66 ports d'entrées/sorties qui est nettement insuffisant en plus de n'être disponible que dans un format de carte PCI.

La carte Benera, quant à elle, est conçue autour d'un Virtex-E et d'un Virtex-II. Elle possède 3 horloges programmables et 16 mégaoctets de mémoire de type flash. Aucune mémoire tant statique que dynamique n'est présente sur la carte. Il serait cependant possible d'y en attacher en utilisant quelques uns des 400 ports d'entrées/sorties disponibles.

Benera n'est disponible qu'en format cPCI ce qui veut dire que bon nombre de ses entrées/sorties ne sont disponibles que pour la communication du bus. De par sont format, elle ne pourrait être utilisée.

3.4 Résultat

La recherche permis d'arriver à la conclusion qu'aucune carte existante ne pouvait adéquatement répondre aux exigences demandées par la spécification. Il fallait donc concevoir une carte qui pouvait respecter ces exigences. À chacun des différents points de la spécification, un degré d'importance a été associé. Le tableau X illustre cette classification. Le chapitre 4 fera une description exhaustive des composantes choisies pour répondre à chacun des points.

Tableau X

Importance des points de la spécification

Spécification préliminaire	Degré d'importance	Choix de composantes / architecture
FPGA > 125Mhz	Nécessaire	Virtex-II de Xilinx
220 entrées/sorties aux démos min.	Nécessaire	240 sont prévus
Horloge programmable, 1-200MHz	Hautement désirable	AMI Semiconducteur (0-300MHz)
Sources de tenson programmables	Hautement désirable	Potentiomètres digitaux 10bits
Convertisseurs NA/AN rapides	Optionnel	Ajoutés par expansion
Convertisseurs NA/AN lents	Optionnel	Ajoutés par expansion
Capteurs de température	Désirable	Capteurs à référence externe
Grande quantité de mémoire (8Mo min.)	Désirable	Mémoire statique 200MHz
Interface de communication	Nécessaire	Port ethernet
Une seule alimentation	Hautement désirable	Multiples convertisseurs DC-DC
Expansion	Désirable	Disponible

3.5 Sommaire

Le test des démos demande une spécification très exigeante. Étant donné que la puce la plus complexe, jusqu'à maintenant, est le démo 5, la spécification de la carte a été élaborée pour tenir compte principalement de ses besoins. Outre le nombre important de ports d'entrées/sorties à haute vitesse, les tests demandent une génération de tensions analogiques très stables et une capture de la température de la puce. La spécification préliminaire énoncée est une ébauche des fonctionnalités que la carte de test devrait avoir. Elle est partiellement composée d'éléments désirés par la compagnie pour des projets futurs.

Malgré les besoins spécifiques de la spécification, il aurait été possible qu'une carte de développement FPGA déjà disponible sur le marché, comble la majorité des besoins pour effectuer le test des démos. Les fonctionnalités absentes mais nécessaires auraient pus être ajoutées en utilisant des ports d'expansion. La plupart des cartes disponibles sur le marché sont basées sur le Virtex de première génération qui aurait permis d'effectuer le test à la cadence désirée de 125MHz. Il aurait donc été possible d'envisager l'achat d'une carte de développement basée sur ce type de FPGA.

Un éventail de cartes a été évalué selon les critères de vitesse, de nombre de ports d'entrées/sorties, de flexibilité de l'horloge et de la quantité de mémoire disponible. Toutes les cartes évaluées ont la même défaillance: elles possèdent un nombre insuffisant de ports d'entrées/sorties.

CHAPITRE 4

ERINYES, PHYSIQUE

Ce chapitre porte sur l'aspect physique (« hardware ») de la carte Erynies. Le banc de test est constitué d'une carte maîtresse et de deux mezzanines. L'avantage d'utiliser des mezzanines est de personnaliser, à un coût plus modique, le banc de test pour chacun des démos. Chaque mezzanine peut supporter son propre démo avec tous les composants nécessaires pour les tests. De cette façon, tous les convertisseurs numériques/analogiques ainsi que les convertisseurs analogiques/numériques peuvent être ajoutés ultérieurement dans le design d'une mezzanine. Seuls les capteurs de température ainsi que les tensions analogiques stables resteront sur la carte mère.

Pour permettre la communication avec l'extérieur, la carte est munie d'un microcontrôleur. Ce dernier permet également de programmer et de contrôler le FPGA.

4.1 Spécification détaillée

Chacun des composants de la carte fut choisie avec la performance en tête. Ils ont également été choisis pour permettre d'optimiser le temps de développement tant matériel que logiciel. La nouvelle carte peut être divisée en 4 parties: le CPU, le FPGA, les mezzanines et l'alimentation. Une vue globale est représentée à la figure 23. Les détails et les schémas électroniques de la carte peuvent être retrouvés respectivement en annexe 4 et annexe 5.

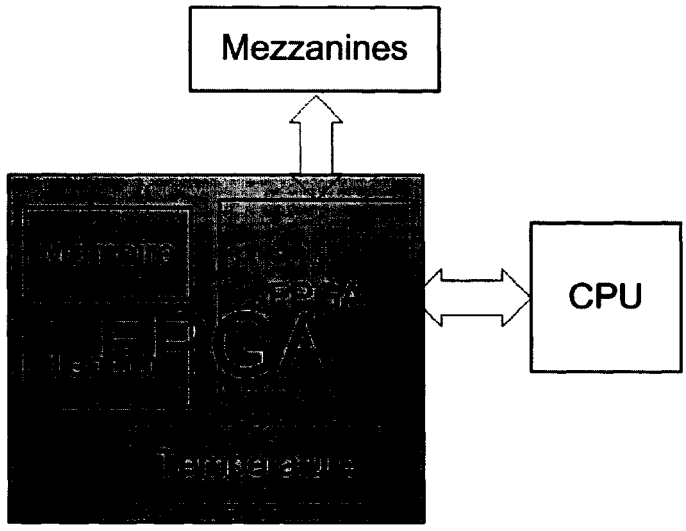


Figure 23 Représentation schématique de l'architecture proposée

4.1.1 Section CPU

Cette section de la carte est le point de communication avec l'extérieur. Elle sert également de contrôleur pour le FPGA. La section CPU a grandement été calquée sur la carte de développement de Motorola du ColdFire MCF5272: la MCF5272C3. Les pages 2 à 19 de l'annexe 5 présentent cette section.

4.1.1.1 Microcontrôleur

Le microcontrôleur choisi est un ColdFire MCF5272 de Motorola cadencé à 66MHz. Le ColdFire est un processeur 32bits possédant le jeu d'instructions de la série des 68000. Les microprocesseurs Motorola sont réputés pour leur facilité d'intégration et leur support logiciel. Bon nombre de compilateurs et d'assembleurs gratuits sont disponibles pour cette architecture. La grande force de ces processeurs est leur habileté à adapter la

largeur et la synchronisation de leur bus, ce qui les rend compatibles à presque n'importe quel périphérique.

Le MCF5272 possède plusieurs interfaces de communication qui le rend parfait pour les tâches de contrôle. Les interfaces disponibles sont: deux ports EIA-RS232, deux ports USB 1.0 et une interface Ethernet 10/100Mbps. Seules les interfaces EIA-RS232 et Ethernet seront utilisées.

Ce microcontrôleur possède un port de déboguage à double fonction. Il peut être utilisé comme un port d'accès JTAG ou comme un port *Background Debug Mode* (BDM). Pour ce type de processeur, il est beaucoup plus avantageux d'utiliser le BDM. Ce mode de fonctionnement permet le contrôle total du microcontrôleur. Il est possible de lui faire exécuter son programme instruction par instruction. Il est également possible d'ajouter ou modifier du code dans les mémoires attachées au microcontrôleur. La première version du logiciel sera chargée dans la mémoire flash à l'aide de ce port.

Une des principales raisons du choix de ce microcontrôleur est la disponibilité du noyau du système d'exploitation Linux pour cette plateforme. La version embarquée de Linux, appelée µCLinux, roule déjà sur la carte de développement de Motorola pour le MCF5272. µCLinux possède tous les logiciels de base de Linux comme un éditeur de commandes, un éditeur de texte, toutes les commandes requises pour le traitement des fichiers ainsi que tous les principaux logiciels de réseautique comme telnet et ftp. L'avantage d'utiliser un système d'exploitation existant est d'éliminer le développement du support logiciel des interfaces de contrôle et de communication.

Le microcontrôleur demande, lors de la mise sous tension, que certaines broches aient un niveau précis pour lui offrir une configuration de base. Pour éviter l'erreur dans l'assignation des broches de configuration, il fut choisi de les placer sur des

interrupteurs. De cette façon, il sera possible de modifier la configuration de base en tout temps.

4.1.1.2 Mémoire

Le processeur possède sa propre mémoire. Comme il un contrôleur de mémoire dynamique et que ce type de mémoire est peu coûteuse, le ColdFire aura 64 mégaoctets de mémoire vive, soit le maximum supporté par la puce. Une telle capacité de mémoire du coté du processeur permet d'emmagasiner bon nombre de vecteurs et de résultats de test pour le transfert vers les mémoires flash. Elle permet également d'être à l'aise pour l'exécution des logiciels de contrôle. Cette mémoire est organisée en deux puces de 256Mb x 16 d'une vitesse maximale de 100MHz. La synchronisation de ce type de mémoire est stricte et ne doit être ralentie d'aucune façon.

Pour permettre le stockage du système d'exploitation, du microprogramme (« firmware ») du FPGA et des données provenant du test, la carte mère est dotée de mémoire de type flash. Ces mémoires de la compagnie AMD ont un cycle d'écriture garanti d'un million de cycles. La carte possède 3 puces de 32 mégabits organisées en 2MB x 16. Chacune de ces puces est utilisée pour stocker son propre type de données : système d'exploitation/logiciels, données et microprogramme du FPGA.

4.1.1.3 Contrôle de la température du FPGA

Le FPGA contrôle souvent des tests qui nécessitent des fréquences d'horloges relativement rapides. Selon le degré d'utilisation de cette puce, la température du FPGA sera plus ou moins élevée. Il est possible que certains tests augmentent sa température jusqu'à un point de non fonctionnement. Pour éviter une surchauffe du FPGA, un mécanisme de contrôle de la température sera implanté sur la carte de test. Le

microcontrôleur fera une évaluation périodique de la température pour modifier la vitesse de rotation du ventilateur en conséquence.

Il fut choisi que le contrôle de la température du FPGA soit pris en charge par le microcontrôleur. La capture de la température est effectuée par le MAX1669 de MAXIM. Cette puce est capable de transmettre la température interne d'un circuit possédant une diode similaire à celle du Pentium® III d'Intel avec une précision de ±3 degré C. Cette puce possède un contrôle de ventilateur intégré qui élimine le besoin d'une circuiterie externe. Elle possède en plus deux lignes d'interruptions, une comme avertissement et l'autre lors de la surchauffe. Lors d'un avertissement, la vitesse de rotation du ventilateur devra immédiatement être de 100%. Lors d'une surchauffe, le FPGA devra être placé en mode d'hibernation pour cesser toute activité et le laisser refroidir pour éviter de l'endommager.

4.1.1.4 Indicateurs d'état

Le microcontrôleur est branché à 8 diodes électroluminescentes bicolores. Ces diodes servent d'indicateurs d'activité qui permettent, d'un coup d'œil, de connaître l'état dans lequel le microcontrôleur se trouve. Pour le moment, ces indicateurs n'ont pas encore de signification précise. Ces dernières seront définies lors du développement logiciel de la carte. Dans un système embarqué, il est primordial de ne jamais laisser ces indicateurs inactifs.

Le calcul des résistances attachées aux diodes électroluminescentes bicolores en est un délicat. La diode ne fait pas chuter la tension à ses bornes de la même valeur pour les deux couleurs. Une longueur d'onde plus courte demandera une tension plus forte. Selon les spécifications, la couleur vert chute de 2.03 à 2.6V, la couleur ambre de 1.93 à 2.6V et la couleur rouge de 1.72 à 2.5V. Le courant requis est de 10mA pour toutes les

couleurs. Les sorties du microcontrôleur ne peuvent fournir que 4mA. Il a donc fallu ajouter des portes de transfert. Ces portes de transfert perdent environ 100mV en sortie par rapport à la tension d'alimentation. Pour satisfaire à toutes les diodes, des résistances de 110 Ohm ont été choisies offrant, à 10mA, environ 2.1V aux bornes de la diode.

4.1.1.5 Ports d'expansion

Deux connecteurs d'expansion sont prévus pour permettre d'ajouter certains périphériques à la section CPU. Toutes les lignes de données ainsi que les lignes d'adresses se rendent sur ces connecteurs. Les lignes de sélection ainsi que les lignes d'interruption qui ne sont pas utilisées sont également dirigées vers ceux-ci. Il serait possible, par exemple, de brancher des mémoires statiques ou encore des convertisseurs numériques/analogiques ou analogiques/numériques.

4.1.2 Section FPGA

Cette portion de la carte est le coeur du test. Elle comprend tous les accessoires nécessaires au test des démos présents. Les schémas de cette section sont présentés aux pages 20 à 65 de l'annexe 5.

4.1.2.1 FPGA

Comme il a été mentionné, le FPGA choisi est un Virtex de 2 millions de portes logiques de la seconde génération des Virtex de Xilinx: le XC2V2000. Le grade de vitesse de cette puce est de 5 ce qui permet de concevoir un additionneur 16 bits pouvant fonctionner à plus de 239MHz. Un grade supérieur peut évidemment être utilisé.

La deuxième génération du Virtex possède plusieurs points intéressants. Ces puces renferment des composantes de manipulation d'horloges : les DCM ou « Digital Clock Manager ». Ces modules permettent d'effectuer des opérations de multiplication ou de division sur l'horloge ainsi que d'en modifier la phase. Ces caractéristiques permettent de synchroniser la logique du FPGA sur les données en sortie des démos. Les DCM sont probablement capable de générer l'horloge des démos. Ces structures sont caractérisées d'une gigue variant entre 100 et 300ps.

Outre la génération flexible d'horloges, la série Virtex-II possède des entrées/sorties dont les terminaisons sont contrôlées de façon digitale: les DCI ou Digitally Controlled Impedance. Ces terminaisons sont programmées par deux résistances de référence, deux par banque d'entrées/sorties.

Une autre caractéristique intéressante du Virtex-II est de supporter la programmation partielle. Il est possible de ne remplacer qu'une portion du microprogramme permettant ainsi de changer des portions du microprogramme sans procéder à une remise à zéro du FPGA. Ceci permettrait par exemple de charger différents modules de tests tout en conservant les interfaces de base.

La dimension du FPGA ne fut pas choisie pour la quantité de logique disponible mais plutôt pour le nombre de ports d'entrées/sorties. Il fut établi que le test devait se faire sans multiplexage des ports. Selon la spécification énoncée au chapitre 3, un nombre de 220 ports d'entrées/sorties au minimum est requis. Cependant, il ne faut pas oublier que le microcontrôleur, la mémoire statique, les capteurs de température et finalement les potentiomètres digitaux ont besoin de leurs propres ports.

Une contrainte imposée par le FPGA est celle de la division des domaines de puissance à l'intérieur de celui-ci. Le FPGA est subdivisé en 8 banques dont la moitié comprend 72

ports d'entrées/sorties alors que l'autre moitié en comprend 84. Chacune de ces banques possèdent son alimentation propre se situant entre 1.8V et 3.3V. De par cette organisation, il faut prévoir les connections du FPGA aux composantes avoisinantes qui n'ont pas toutes les mêmes tensions d'alimentation. La division des domaines de puissance du FPGA sera discutée à la section 4.3.1.

Le format choisi est le « flip-chip » de 896 connexions qui permet d'obtenir un maximum de 624 ports pour le XC2V2000. Ce nombre de ports devrait être amplement suffisant pour permettre le branchement de touts les composants ainsi que les démos à tester.

4.1.2.2 Mémoire

Le XC2V2000 est capable d'instancier un maximum de 512 banques de 18Kbits soit un total de 129 024 octets de mémoire. Bien qu'elle soit très rapide, cette quantité n'est pas suffisante pour permettre le stockage des vecteurs des résultats et de test. Des banques de mémoires statiques ont donc été prévues: un total de 3 banques pouvant contenir chacune 9 mégaoctets. Les banques sont composées de 4 puces de mémoire 8 mégabits « syncburst » de la compagnie MICRON configurées comme des 512Kbits x 18 de grade 5. Ces puces de type synchrone sont capables de fournir ou absorber des données en rafales et ce, à une cadence maximale de 200MHz. Chacune des 3 banques d'une largeur de 18 bits est contrôlée indépendamment des autres. Cet arrangement permet de doubler ou tripler la bande passante lorsque requis. Il est possible de spécialiser chacune des banques pour le stockage de données distinctes pour permettre une utilisation simultanée des 3 banques.

4.1.2.3 Horloge programmable

Le FPGA est connecté à une horloge programmable. Fabriquée par AMI Semiconductors, le FS7140 est un générateur d'horloge très flexible capable de générer un signal de 0 à 300MHz avec une gigue maximale de 300ps. Cette gigue ne dépend uniquement que des diviseurs internes choisis. La sortie de cette horloge est un signal PECL de 3.3V. Elle est contrôlée par une interface I^2C provenant du microcontrôleur.

Deux connecteurs de type BNC ont été prévus pour apporter une horloge de l'extérieur. Ces entrées offrent une flexibilité supplémentaire et une porte de sortie advenant un problème de la génération de l'horloge du FS7140.

Étant donné que la sortie du générateur est de type PECL et que les démos ont une entrée d'horloge de type LVCMOS à 1.8V, les démos ne pourront pas être alimentés par cette puce. Le FPGA devra fournir l'horloge du générateur par un de ses ports. Lorsque cette horloge n'est pas altérée par les DCM, elle n'est pas détériorée et il n'aura pas de gigue additionnée. Cependant, les délais de routage viennent ajouter un déphasage qui pourrait désynchroniser les démos du FPGA. Il est alors possible d'échantillonner les données sur un front différent ou encore d'utiliser les DCMs pour déphaser les registres d'entrées et de sorties du FPGA par rapport à l'horloge transmise vers les démos.

4.1.2.4 Tensions analogiques stables

Le circuit de tensions analogiques est décomposé en 2 parties: la référence et le circuit diviseur. La référence provient d'une puce, la ADR420 de Analog Devices, offrant une tension stable et filtrée de 2.048V. Ce circuit peut générer un courant de sortie de 10mA au maximum. Le circuit de division est composé de 18 potentiomètres digitaux de 10Kohm et d'une précision de 10 bits. Bien que les démos ne demandent aucun courant

sur leurs entrées de contrôle de lignes à délais, il en est tout autre pour les potentiomètres digitaux. Si un seul ADR420 était utilisé, les potentiomètres digitaux demanderaient un courant de 66.3mA au total dépassant largement la limite de 10mA du circuit. En séparant les potentiomètres digitaux en trois groupes de 6 potentiomètres chacun, le courant total maximal demandé à un circuit de référence est d'environ 7.4mA.

Les AD5231 de Analog Devices sont des potentiomètres linéaires fonctionnant sur une interface sérielle de type SPI fournie par le FPGA. Les potentiomètres possèdent une mémoire non volatile qui permet de conserver les valeurs ajustées même après une remise à zéro.

4.1.2.5 Capture de température des démos

Comme il a été mentionné au chapitre 3, les puces sous test possèdent des diodes qui permettent d'évaluer leur température. La capture de la température des démos est effectuée par des ADM1023 de Analog Devices. Ces capteurs de température ont une précision de ±1°C sur une résolution de 0.125°C. Ils possèdent deux lignes d'interruption, une pour l'alerte et une pour la surchauffe. Le seuil de déclenchement des interruptions ainsi que leur hystérésis sont programmables.

Il fut choisi d'utiliser le FPGA comme port d'accès pour la capture de la température plutôt que d'utiliser des lignes du microcontrôleur. Ce branchement fut fait en grande partie pour isoler la section CPU du test. Un autre avantage de ce branchement est de permettre l'arrêt immédiat des éléments thermiques lors d'une surchauffe.

4.1.2.6 Indicateurs d'état

Pour permettre de suivre d'un coup d'oeil l'évolution des tests au niveau du FPGA, 12 diodes électroluminescentes bicolores sont branchées au FPGA. Ces diodes servent à indiquer l'état dans lequel le FPGA se trouve.

Le calcul des résistances est fait de façon similaire à ce qui a été montré à la section 4.1.1.4. Avec une résistance de 110ohm, une tension maximale de 2.2V sera appliquée aux bornes de la diode lorsqu'un courant de 10mA les traverse. Les sorties du FPGA branchées aux diodes devront être de type LVCMOS 3.3V qui est capable de fournir un courant de 24mA.

4.1.3 Section mezzanines

Pour le test des démos 4 et 5, deux mezzanines sont nécessaires. Ces mezzanines sont très semblables. Essentiellement, elles sont composées d'un support à force d'insertion nulle, de points de tests et de connecteurs à haute vitesse.

Les connecteurs choisis pour fixer les mezzanines sont des *QSTRIP* de la compagnie Samtec. Ces connecteurs offrent une excellente isolation signal à signal en plus d'être mécaniquement stables[12]. Ils possèdent, tout comme les connecteurs MICTOR (Matched Impedance ConnecTOR), une ligne centrale d'isolation pour le retour vers la masse. La figure 24 montre une vue de plan d'un connecteur QSTRIP.

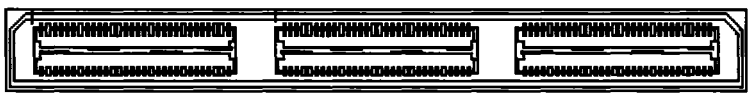


Figure 24 Vue de plan d'un connecteur QSTRIP

Les connecteurs permettent le branchement de plusieurs formats de mezzanines. De façon symétrique, deux connecteurs dans la partie supérieure et deux connecteurs dans la partie inférieure permettent le branchement des mezzanines principales. Ces connecteurs ne contiennent qu'une seule tension de 1.8V et sont prévus pour le test des démos existants.

En plus des 4 connecteurs principaux, il y a deux autres connecteurs secondaires au centre de la carte (cf. figure 33, page 82). Ces connecteurs permettent d'amener les tensions de 3.3 et 5V sur les mezzanines. Le Virtex-II ne supporte pas les tensions supérieures à 3.3V. La tension de 5V est prévue pour les tensions de référence des convertisseurs numériques/analogiques et analogiques/numériques lorsque requises. Tous les signaux de 3.3V inutilisés ont été acheminés vers ces connecteurs. Par l'arrangement de ces connecteurs, il est possible de concevoir des cartes mezzanines possédant de 1 à 6 connecteurs et supportant 3 tensions différentes.

Pour limiter la longueur des traces des tensions stables et des capteurs de température, ces puces sont placées près des connecteurs. Des traces plus courtes diminuent les risques de pollution sur les tensions de référence et diminuent la résistance des traces pour la capture de la température (les capteurs de température fonctionnent sur une base de courant plutôt que sur une base de tension).

4.1.4 L'alimentation

La partie alimentation d'Erinyes a été en majeure partie dessinée par les ingénieurs d'Hyperchip. Ce module provient, en fait, de l'alimentation des cartes de lignes de la compagnie. Ces cartes reçoivent une tension de 48V en courant continu qui sera convertie en plusieurs tensions plus faibles. Le système d'alimentation est séparée en trois parties: le séquenceur, la régulation et la supervision.

Le séquenceur, le SHM4804, est un circuit de remplacement à chaud (« hot swap », remplacement lorsque le système est en marche) de Summit Microelectronics Inc. Erinyes n'a pas vraiment besoin de ce genre ce circuit étant donné qu'il ne s'agit que d'une carte autonome. Cependant, ce type de circuit possède une fonctionnalité très intéressante: il permet de démarrer des convertisseurs DC/DC en une séquence programmable. Certaines puces ont besoin de plusieurs tensions d'alimentation. Dans le cas d'Erinyes, seul le FPGA se retrouve dans cette situation. Selon la spécification du Virtex-II, le FPGA peut consommer jusqu'à 600mA instantanément par banque si leur alimentation arrive avant l'alimentation de la portion de contrôle (VCCAux). Dans une séquence, il sera possible d'alimenter chacune des portions du FPGA pour éviter un stress momentané sur l'alimentation.

En évitant ainsi cette demande de courant momentanée, il est possible de choisir des pièces qui fourniront un courant moindre et ainsi diminuer les coûts rattachés à l'alimentation. Un autre bienfait d'alimenter le système petit à petit est de limiter le bruit sur la carte au démarrage pour s'assurer que chacune des puces reçoivent leur signal de remise à zéro.

La partie de régulation est composée de plusieurs convertisseurs DC/DC. La première couche de convertisseurs permet d'abaisser la tension de 48V pour la convertir en

tensions de 3.3 et de 5V. Ces tensions passeront à travers une seconde couche de convertisseurs qui génèreront à leur tour les tensions de 1.5 et de 1.8V. Ces modules sont tous capable de produire un courant d'au moins 5 ampères, la première couche étant capable d'en fournir 25.

La supervision est faite par deux LCT1326 de la compagnie Linear Technology. Ces puces assurent que les tensions nominales soient atteintes avant de permettre le démarrage de la carte. Si une des tensions venait à s'écraser, elles arrêteraient le système en activant la ligne de remise à zéro. La redémarrage manuel est également possible par le biais d'un bouton poussoir.

4.2 La programmation du Virtex

Un des importants aspects de la fonctionnalité de la carte est sa capacité de programmer le FPGA par le biais du microcontrôleur. Il existe plusieurs façons de programmer le FPGA. Outre le port JTAG, il est possible d'utiliser le mode « Master/Slave Serial » et le mode « SelectMAP » en mode maître ou esclave.

La norme JTAG (IEEE-1149.1) est largement utilisée pour la programmation en circuit et pour le test. C'est un protocole de communication série qui ne requiert que 5 signaux pour fonctionner, tel qu'il a été vu au chapitre 2. Pour programmer le FPGA dans ce mode, il faut implémenter un logiciel de contrôle JTAG.

Le mode Master/Slave Serial est également un mode de programmation sériel. Pour programmer le FPGA dans ce mode, 5 signaux sont également nécessaires: DIN, CCLK, PROG_B, INIT_B et DONE. Un logiciel de contrôle est encore nécessaire.

Le mode SelectMAP est le mode de programmation le plus rapide mais celui qui requiert le plus grand nombre de signaux. Comme il s'agit d'une programmation en parallèle, 8 bits de données sont nécessaires en plus des signaux de contrôles qui sont au nombre de 7, soient PROG_B, INIT_B, CCLK, CS_B, RDRW_B et DONE. Ces lignes de contrôlent sont dédiées à la programmation et par conséquent, leur utilisation n'affecte pas le nombre total de ports d'entrées/sorties disponibles. En branchant les 8 bits de données directement sur le bus du microcontrôleur, il est possible de programmer le Virtex en utilisant les signaux de contrôle du ColdFire. Ces lignes de données peuvent être utilisées par la suite pour permettre la communication entre le FPGA et le microcontrôleur. La programmation en SelectMAP peut se faire en mode maître ou esclave. En mode maître, le FPGA fournit l'horloge et contrôle le flux de l'information. En mode esclave, il reçoit son horloge et permet le contrôle de sa programmation par un agent externe.

Le mode SelectMAP esclave a été choisi pour programmer le FPGA. Ce mode est plus rapide que les modes sériels et il pourrait être possible de programmer le FPGA sans avoir à développer de logiciel d'interface complexe.

4.2.1 Programmation en mode SelectMAP esclave

Pour permettre d'établir la compatibilité entre le microcontrôleur et le FPGA, il faut tout d'abord analyser les chronogrammes de communication de chacun. Dans le mode SelectMAP esclave, le programme du FPGA est transféré octet par octet suivant un certain protocole de transmission. Le chronogramme de cette communication est illustré à la figure 25.

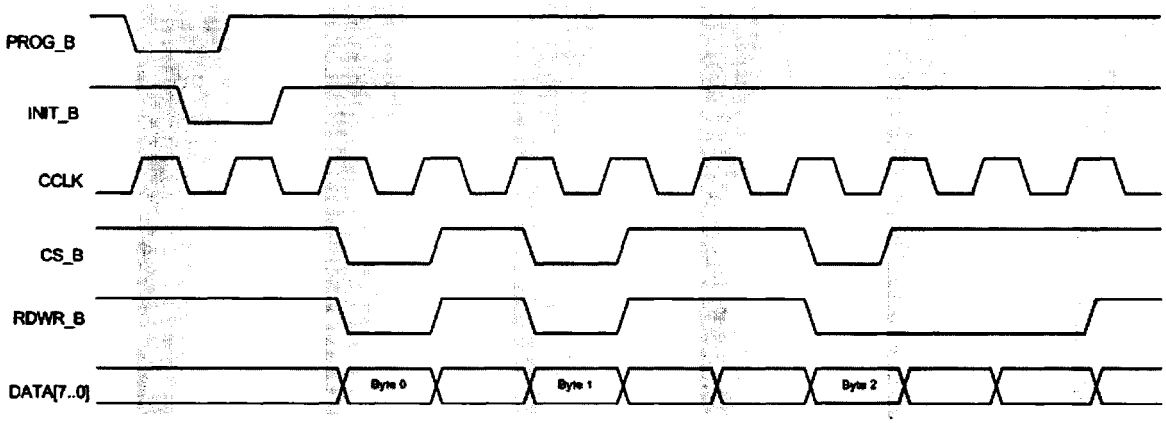


Figure 25 Chronogramme de programmation du Virtex-II

La programmation est lancée de deux façons: automatiquement lors de la mise sous tension ou en abaissant la ligne PROG_B. Immédiatement après l'activation de la programmation, le FPGA abaisse la ligne INIT_B indiquant le début de l'effacement de la mémoire. La ligne INIT_B se relève sitôt que la mémoire est complètement effacée; le FPGA entre alors dans le mode de programmation. Il est cependant possible d'interrompre temporairement le FPGA immédiatement après l'effacement de la mémoire en forçant la ligne INIT_B basse.

Le FPGA ne chargera un octet que si ses lignes CS_B et RDWR_B sont actives au front montant de CCLK. Cependant, pour ne pas interrompre la programmation, lors d'un front montant de l'horloge, la ligne RDWR_B doit toujours être active lorsque la ligne CS_B est active sans quoi, le FPGA sort du mode de programmation. Ce mode se termine normalement lorsque tous les bits ont été transférés (pour le XC2V2000, 7 492 000 bits).

Le FPGA possède une vérification de type CRC-16[1] pour lui permettre de valider le programme reçu. Une fois le programme chargé, il compare ce CRC avec celui qui est

[1] CRC : Cyclic Redundancy Check, cf. Virtex-II Platform FPGA Handbook v1.2, p.393

présent dans le flux de données. Si les deux CRC diffèrent, la ligne INIT_B s'abaisse et le FPGA tombe en mode d'attente. Le processus doit alors recommencer en abaissant la ligne PROG_B. Lorsque le FPGA accepte le programme envoyé, il active sa ligne DONE et démarre automatiquement.

4.2.2 Branchement du FPGA au microcontrôleur

Le bus du microprocesseur ColdFire fonctionne tel qu'illustré à la figure 26:

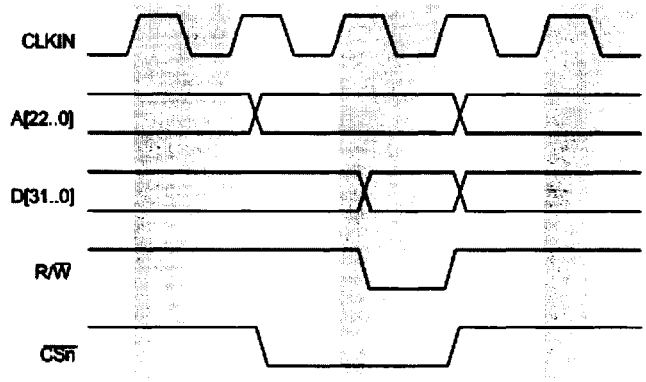


Figure 26 Chronogramme d'accès en écriture du microprocesseur ColdFire

En comparant les figures 25 et 26 il est d'évidence que le chronogramme du bus du microprocesseur n'est pas directement compatible au chronogramme de programmation du FPGA. En effet, la sélection de puce (CSn) arrive un coup d'horloge trop tôt par rapport avec les données et la ligne de permission d'écriture (R/W) et reste active pendant deux coups d'horloge alors qu'un seul est requis. Un tel décalage de la sélection de puce ferait sortir le FPGA du mode de programmation. De plus, une durée de la ligne de permission d'écriture de cette longueur chargerait le FPGA d'un octet de trop par accès. Nous ne pouvons donc pas nous servir directement des signaux de contrôle du bus du microcontrôleur.

Il faudrait donc concevoir un mécanisme de contrôle des signaux CS_B, PROG_B, INIT_B et RDRW_B. Il fut choisi d'utiliser les 8 premiers bits du bus de données du microprocesseur pour effectuer le transfert. Les signaux CS_B, PROG_B, INIT_B, RDRW_B et DONE seront branchés sur des entrées/sorties générales du microcontrôleur. Ces lignes peuvent être contrôlées une à une de façon tout à fait indépendante. L'horloge de programmation du FPGA (CCLK) sera fournie par les fronts montants du signal de sélection. De cette façon, les lignes CS_B et RDRW_B pourront être maintenues sur un niveau bas. Le débit sera contrôlé par le signal CSn pour chaque accès. Un tel branchement permettra d'obtenir un chronogramme tel que celui de la figure 27.

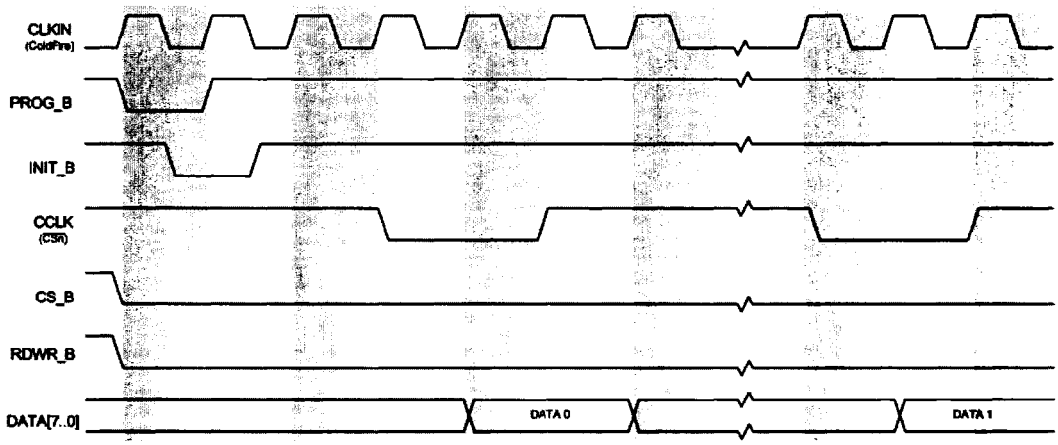


Figure 27 Chronogramme de programmation résultant du branchement FPGA - microcontrôleur

4.2.3 Caractérisation temporelle

Il faut également considérer les caractéristiques temporelles des signaux utilisés. Les informations de temps de préparation et temps de maintien seront essentielles pour valider la compatibilité entre FPGA et microcontrôleur. Le tableau XI montre les caractéristiques temporelles des signaux de programmation du Virtex-II :

Tableau XI

Caractéristiques de synchronisation de la programmation du Virtex-II[2]

	Description	Symbole	Valueur	Unités
CCLK				
	D_{0-7} Setup/Hold	T_{SMDCC}/T_{SMCCD}	5.0/0.0	ns, min
	CS_B Setup/Hold	T_{SMCSCC}/T_{SMCCCS}	7.0/0.0	ns, min
	RDWR_B Setup/Hold	T_{SMCCW}/T_{SMWCC}	7.0/0.0	ns, min
	BUSY Propagation Delay	T_{SMCKBY}	12.0	ns, max
	Maximum Frequency	F_{CC}_SelectMAP	66	MHz, max
	Maximum Frequency with no handshake	F_{CCNH}	66	MHz, max

Le tableau XII montre les caractéristiques temporelles du bus du microcontrôleur.

[2] Virtex-II Platform FPGA Handbook v1.2, p. 364.

Tableau XII

Caractéristiques temporelles du bus du microcontrôleur MCF5272 ColdFire[3]

Nom	Characteristiques[1]	0-66 Mhz		Unité
		Min	Max	
B6a	SDCLK to chip selects (CS[6:0]) valid	-	12	ns
B6b	SDCLK to byte enables (BS[3:0]) valid	-	9.5	ns
B6c	SDCLK to output enable (OE) valid	-	9	ns
B6d	SDCLK to write enable (R/W) balid	-	8	ns
B6e	SDCLK to reset output (RSTO) valid	-	13.5	ns
B7a	SDCLK to control output (CS[6:0], OE) invalid (output hold)	1.5	-	ns
B7b	SDCLK to control output (BS[3:0], R/W) invalid (output hold)	1	-	ns
B7c	SDCLK to reset output (RSTO) invalid (output hold)	4	-	ns
	Address and attribute Outputs			
B8	SDCLK to address (A[22:0]) valid	-	13	ns
B9	SDCLK to address (A[22:0]) invalid (output hold)	1.5	-	ns
	Data Outputs			
B11	SDCLK to data output (D[31:0]) valid	-	11	ns
B12[2]	SDCLK to data output (D[31:0]) invalid (output hold)	1	-	ns
B13	SDCLK to data output (D[31:0]) high impedance	-	6	ns

[1] All timing references to SDCLK are given to its rising edge when bit 3 of the SDRAM control register is 0.

[2] Data output is held for one CPU clock period after deassertion of BS[3:0].

Comme les lignes PROG_B, INIT_B, CS_B et RDRW_B peuvent être maintenues sur plusieurs cycles et qu'elles ne dépendent pas des caractéristiques temporelles du bus du microcontrôleur, l'information relative à ces signaux peut être ignorée. Nous pouvons également ignorer les temps de maintien puisque le tableau XII montre que le FPGA n'en a aucunement besoin.

Le tableau XI montre que les données doivent être préparées au moins 5.0ns avant le front montant de l'horloge CCLK. L'horloge est fournie par le signal CSn dont le front montant arrive, selon le tableau XII, à un minimum de 1.5ns après la montée de l'horloge du microcontrôleur. Les données du ColdFire sont valides à un maximum de 11ns après le front de son horloge. Il est à noter que les données du microcontrôleur

[3] MCF5272 ColdFire® Integrated Microprocessor User's Manual (MCF5272UM.pdf), page 23-8, table 23-8, Processor Bus Output Timing Specifications.

passent à travers une porte de transfert. Tel que montré dans le tableau XIII, le temps de propagation de cette porte est dans le pire des cas, de 3ns pour une tension de 3.3V. À 66MHz, la période d'un cycle a une largeur de 15ns.

Le temps de préparation effectif devient : 15 - (1.5 + 11) = 2.5. Les données seront donc prêtes 2.5ns avant le front montant de CSn ou CCLK. Le temps de préparation n'est donc pas respecté.

Tableau XIII

Spécifications temporelles de la porte de transfert[4]

PARAMETER	DE (INPUT)	À (OUTPUT)	V_{CC} = 1.8 V	V_{CC} = 2.5 V ± 0.2 V		V_{CC} = 2.7 V		V_{CC} = 3.3 V ± 0.3 V	
			TYP	MIN	MAX	MIN	MAX	MIN	MAX
t_{pd}	A or B	B or A	¶	1	3.7		3.6	1	3
t_{en}	OE	A or B	¶	1	5.7		5.4	1	4.4
t_{dis}	OE	A or B	¶	1	5.2		4.6	1	4.1

¶ L'information n'était pas disponible à la date de publication

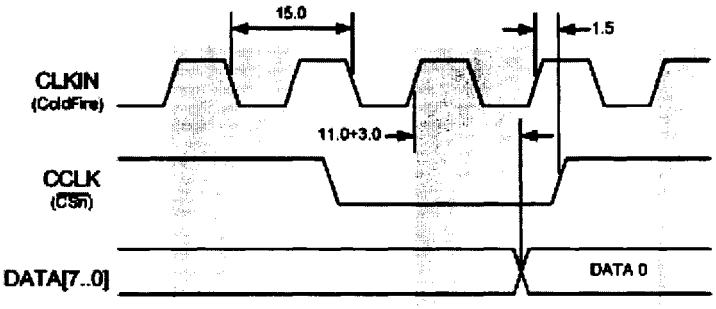


Figure 28 Caractérisation temporelle du chronogramme de programmation

Pour les calculs de temps de préparation, il fut choisi d'employer le pire cas. Il ne manque que 2.5ns pour que ce temps soit respecté. Il pourrait être suffisant de dire que

[4] SN74ALVCH16245 16-bit bus transceiver with 3-state outputs, Texas Instruments, SCES015H, revised version September 2001.

les puces performeront mieux que dans ce pire cas mais il serait plus prudent de s'assurer qu'un tel cas pourrait fonctionner.

Pour permettre une communication sans faille, une modification est nécessaire. Le bus du ColdFire est très flexible et permet de préparer les données pour le front de l'horloge qui précède le signal CSn. Le chronogramme du bus du microcontrôleur résultant est illustré à la figure 29.

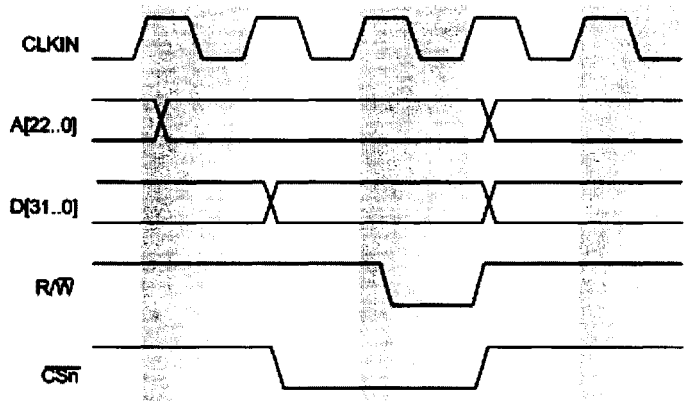


Figure 29 Chronogramme de communication avec préparation d'adresse

En utilisant ce mode de contrôle du bus, le temps de préparation est augmenté d'un cycle soit 15ns qui est largement suffisant pour respecter les contraintes exigées par le FPGA.

4.3 Conception de la carte de circuits imprimés

La conception physique de la carte est une des étapes parmi les plus délicates. La qualité des signaux voyageant sur la carte est dépendante de plusieurs facteurs dont le positionnement des pièces, la subdivision des couches de la carte, le routage des signaux.

4.3.1 Placement préliminaire

Le placement des pièces est la première étape, après la réalisation schématique. Le placement détermine si les signaux peuvent être routés ou non et par conséquent, joue un rôle important dans l'intégrité de ceux-ci. Le placement préliminaire de la figure 30 montre une carte d'environ 400 x 250 mm.

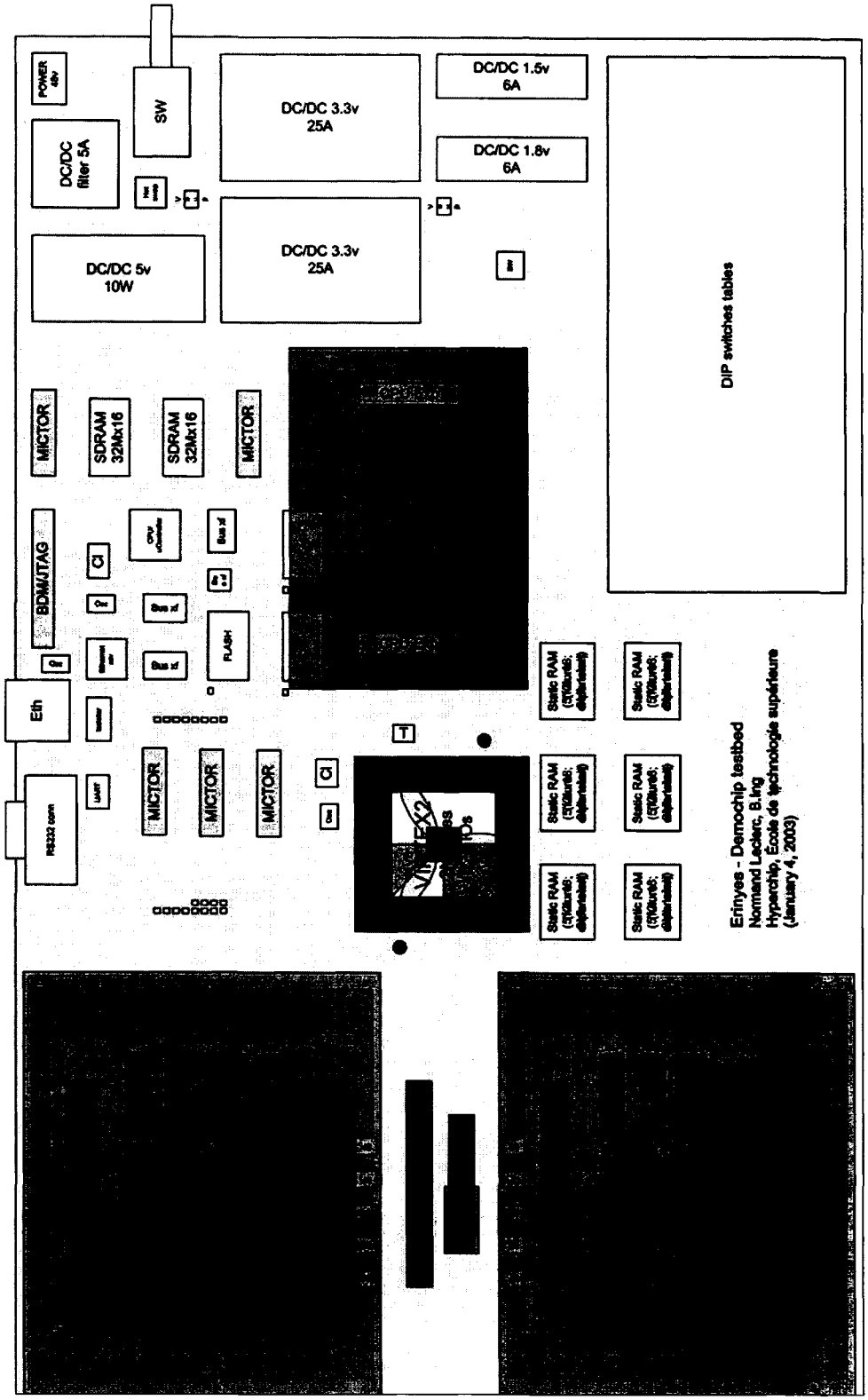


Figure 30 Placement préliminaire de la carte mère

Les mezzanines ont une dimension de 90 x 90 mm comme montré à la figure 31. Chacune des parties de la carte est isolée par rapport aux autres pour diminuer les interférences. Toutes les connexions vers des câbles ont été placées horizontalement et près de la partie supérieure de la carte pour éviter les encombrements de fils et pour augmenter la liberté de mouvement de l'utilisateur autour de la carte.

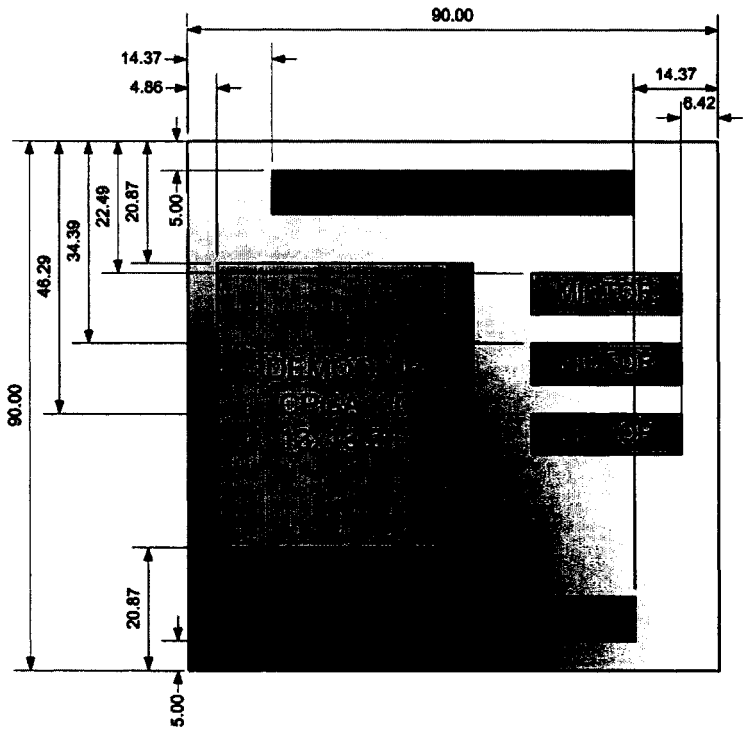


Figure 31 Placement préliminaire de la mezzanine

4.3.1.1 L'alimentation

L'alimentation fut placée complètement à droite de la carte, l'interrupteur le plus près que possible du haut pour ne pas encombrer l'utilisateur. Les tensions élevées ont été placées le plus près possible du filtre et les plus basses ont été éloignées de l'entrée de

l'alimentation externe. Ce placement assure que le filtre et les convertisseurs à tension plus élevée n'entrent pas en interférence avec les tensions plus faibles.

4.3.1.2 Le microcontrôleur

Un peu plus près du centre se trouve la section CPU. Il s'agit d'un domaine d'alimentation unique de 3.3V. Cette section étant de tension relativement élevée, elle est plus résistante aux interférences que les convertisseurs à courant continu peuvent causer. Elle est également moins sensible que le reste du système étant donné sa vitesse d'horloge moins élevée. Tel que vu précédemment, la section CPU possède une cadence de 66MHz alors que les autres sections peuvent fonctionner jusqu'à 300MHz.

4.3.1.3 Le FPGA

Vers le bas de la carte se trouve la section FPGA. Elle possède plusieurs domaines de puissance dont les deux plus faibles et en plus, elle reçoit les deux horloges. Sa position ainsi que son orientation devient une opération assez complexe.

Pour planifier les domaines de puissance, il a fallu tenir compte de deux principaux facteurs. Le premier facteur est que les broches de programmation du mode SelectMAP sont placées dans les banques 4 et 5. Ces banques doivent donc obligatoirement avoir une tension de 3.3V. Le second est que les démos ont besoin d'un nombre suffisant de ports d'entrées/sorties. Un nombre d'au moins 110 par démo serait idéal.

Le Virtex-II comporte 8 banques d'entrées / sorties, 4 de 84 ports et 4 de 72 ports. Il fut choisi d'utiliser deux banques de 84 ports et une de 72 pour les démos. La banque de 72 ports sera divisée en deux et distribuée aux deux mezzanines pour offrir un maximum de 120 entrées/sorties à chacun des démos. Il est donc possible de faire le test de puces

d'une dimension légèrement supérieure. Ces banques ont été choisies à l'opposé de celles réservées pour la programmation du FPGA. Ainsi, les banques 2, 7 et de façon tout à fait arbitraire la banque 1, ont été placées à 1.8V. Toutes les autres banques possèderont une tension de 3.3V. La figure 32 montre l'organisation des domaines de puissance choisie lors de la conception.

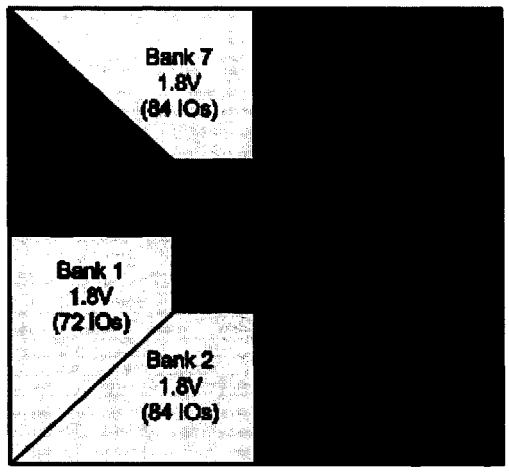


Figure 32 Domaines de puissance du FPGA

Une fois le FPGA divisé, les connecteurs pour les mezzanines peuvent être placés pour permettre la conception de cartes de différentes tailles. La disposition des connecteurs tel qu'illustré à la figure 33 offre une grande stabilité mécanique pour à peu près toutes les dimensions de mezzanine.

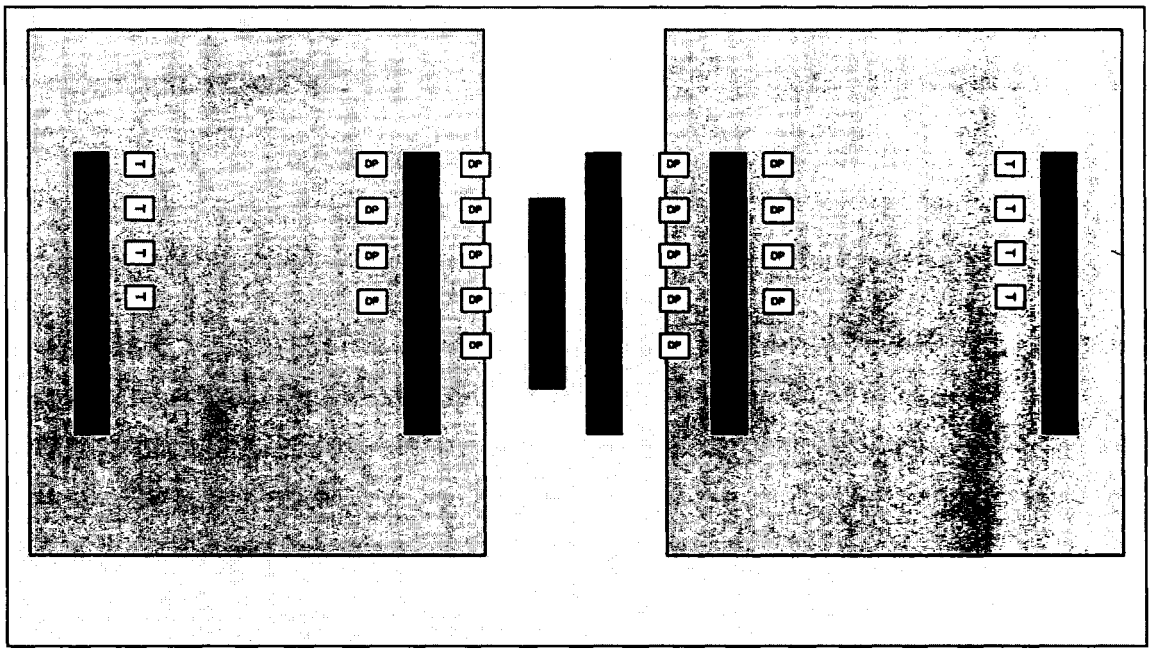


Figure 33 Disposition des connecteurs des mezzanines

4.3.1.4 La mémoire périphérique au FPGA

Pour assurer la synchronisation entre le FPGA et les mémoires, il faut que ces dernières soient localisées près du FPGA pour éviter de trop longs délais de propagation des signaux sur la carte de circuits imprimés. Il faut s'assurer que les temps de préparation et de maintien soient respectés. Les temps de maintien peuvent être ignorés pour la simple raison que ces derniers sont très petits voire nuls en comparaison avec la période de l'horloge utilisée.

Le délai de propagation d'un signal dépend des caractéristiques physiques de la carte de circuits imprimés. Ces caractéristiques physiques forment la constante diélectrique de la carte. Le calcul de cette constante est comme suit [3]:

$$\varepsilon_e = \frac{\varepsilon_r + 1}{2} + \frac{\varepsilon_r - 1}{2}\left(1 + \frac{12H}{W}\right)^{-1/2} + F - 0.217(\varepsilon_r - 1)\frac{T}{\sqrt{WH}} \tag{4.1}$$

$$F = \begin{cases} 0.02(\varepsilon_r - 1)\left(1 - \frac{W}{H}\right)^2 & pour \frac{W}{H} < 1 \\ 0 & pour \frac{W}{H} > 1 \end{cases} \tag{4.2}$$

avec ε_r, la constante diélectrique de l'isolant utilisé dans la conception de la carte, H la hauteur du conducteur au dessus d'un plan de masse, W la largeur du conducteur et T l'épaisseur de ce dernier. La vitesse de propagation et finalement le délai de propagation d'un signal dans une trace de cuivre caractérisée par une constante diélectrique se calcule :

$$v = \frac{c}{\sqrt{\varepsilon_r}} \tag{4.3}$$

$$TD = \frac{\text{distance}\sqrt{\varepsilon_r}}{c} \tag{4.4}$$

La constante diélectrique du matériau visé est de 3.55. La vitesse de propagation d'un signal dans une trace ainsi caractérisée sur une distance d'un pouce devient :

$$v = \frac{c}{\sqrt{\varepsilon_r}} = \frac{3\times10^8 \text{ m/s}}{\sqrt{3.55}} = 1.59\times10^8 \text{ m/s}$$

$$TD = \frac{\text{distance}\sqrt{\varepsilon_r}}{c} = \frac{2.54\times10^{-3}\text{m}\cdot\sqrt{3.55}}{1.59\times10^8 \text{ m/s}} = 301\text{ps}$$

Le calcul de la distance maximale entre le FPGA et ses mémoires se fait en deux temps. Premièrement, il faut analyser la distance maximale pour une écriture vers les mémoires. Une fois fait, il faut ensuite déterminer la distance maximale pour une lecture provenant des mémoires. Les délais maximaux sont calculés en utilisant les équations :

$$T_{W\max} = T - t_{sSRAM} - t_{coFPGA} \tag{4.5}$$

$$T_{R\max} = T - t_{sFPGA} - t_{coFPGA} - t_{coSRAM} \tag{4.6}$$

avec T, la période de l'horloge utilisée, t_{sSRAM}, le temps de préparation des mémoires statiques, t_{sFPGA}, le temps de préparation du FPGA, t_{coFPGA}, le temps que prend un signal à sortir du FPGA et t_{coSRAM}, le temps que prend un signal à sortir de la mémoire. Le t_{coFPGA} quant à lui se calcule comme suit :

$$t_{coFPGA} = t_{pp} + t_{driver} + t_{clk} \tag{4.7}$$

où t_{pp} est le temps de base de calcul pour le FPGA, t_{driver} est le délai du au choix de type de port de sortie et t_{clk} est le délai relié au type d'horloge utilisé à l'intérieur du FPGA. Dans le cas du XC2V2000-5, le t_{pp} est de 2.4ns pour une sortie de type LVTTL à transitions rapides de 12mA qui utilisent les DCM. Pour utiliser une sortie LVCMOS 3.3V à transitions rapides de 12mA, le t_{driver} est de 0.05ns. L'horloge utilisée est de type LVPECL 3.3V qui donne un t_{clk} de 0.6ns. Le calcul devient donc :

$$t_{coFPGA} = t_{PP} + t_{driver} + t_{clk} = 2.4\text{ns} + 0.05\text{ns} + 0.6\text{ns} = 3.05\text{ns}$$

Les mémoires utilisées ont un grade de 5. Selon les spécifications de MICRON, leur temps de préparation (t_s) est de 1.5ns et la période minimale possible d'opération des mémoires est de 5ns (200MHz). Le délai maximal devient donc :

$$T_{W\max} = T - t_{sSRAM} - t_{coFPGA} = 5.0\text{ns} - 1.5\text{ns} - 3.05\text{ns} = 0.45\text{ns}$$

Ce délai maximal nous donne, pour l'écriture, une distance maximale de

$$\frac{450\text{ ps}}{301\ ^{\text{ps}}/_{\text{po}}} = 1.50\text{ pouces}$$

Pour le délai maximal en lecture, le XC2V2000-5 donne un t_{pp} de 1.70ns pour une entrée LVTTL standard utilisant les gestionnaires d'horloges (DCM), un t_{driver} nul pour une entrée LVCMOS 3.3V et un t_{clk} de 0.6ns pour une horloge de type LVPECL. Les mémoires ont un t_{coSRAM} de 3.1ns au maximum. Le calcul de T_{RMax} devient :

$$\begin{aligned} t_{coFPGA} &= t_{PP} + t_{driver} + t_{clk} = 1.7\text{ns} + 0\text{ns} + 0.6\text{ns} = 2.3\text{ns} \\ T_{R\max} &= T - t_{sFPGA} - t_{coFPGA} - t_{coSRAM} = 5.0\text{ns} - 2.3\text{ns} - 3.1\text{ns} = -0.4\text{ns} \end{aligned} \qquad (4.8)$$

Il manque donc 0.4ns pour pouvoir respecter le temps de préparation du FPGA et ce, sans même avoir ajouté le délai du à la distance des mémoires. Ce calcul nous montre qu'il ne sera pas possible d'utiliser les mémoires à la fréquence maximale de 200MHz sans utiliser les DCM pour effectuer une acquisition retardée. Il serait par exemple possible de faire une acquisition retardée de 90 degrés sur l'horloge qui pourrait se traduire, dans l'équation (4.8) en diminuant le temps de préparation du FPGA (t_{sFPGA}) à 1.05ns. Ainsi,

$$T_{R\max} = T - t_{sFPGA} - t_{coFPGA} - t_{coSRAM} = 5.0\text{ns} - 1.05\text{ns} - 3.1\text{ns} = 850\text{ps}$$

Ce délai maximal nous donne, pour l'écriture, une distance maximale de

$$\frac{850\text{ ps}}{301\ ^{\text{ps}}/_{\text{po}}} = 2.82\text{ pouces}$$

Comme la distance pour l'écriture est moins élevée, elle sera prise comme distance maximale absolue.

Pour minimiser les distances, il fut choisi de placer les banques de mémoires statiques sur les deux faces de la carte: six sur la face supérieure et six sur la face inférieure. Les

puces de la face supérieure ne seront pas soudées. Ces espaces seront réservés pour expansion future.

4.3.2 Prévision des couches de la carte des circuits imprimés

Un aspect important de la viabilité d'un système électronique est la prévision des couches de la carte de circuits imprimés. Elle doit avoir suffisamment de couches pour permettre un routage propre et suffisamment de plans de masse pour isoler les signaux les uns des autres.

Pour éviter les émissions d'ondes électromagnétiques vers les mezzanines, aucun signal ne se sera placé sur les faces extérieures de la carte (microruban[5]). Tous les signaux voyageront à l'intérieur du circuit imprimé près d'un plan de masse. L'agencement des plans d'alimentation et des plans de masse a pour effet de créer un plan capacitif agissant de la même manière qu'un condensateur de petite valeur en amortissant les petites fluctuations rapides. Ils ont l'avantage d'agir de façon globale contrairement à une dispersion de petits condensateurs qui auraient chacun une influence locale.

Le réflexe premier lors du design d'une carte est de séparer les plans de masse; un par tension. Tous les plans de masse doivent cependant être branchés à un seul et même endroit. Cette règle permet d'éviter que le bruit d'un domaine de puissance ne vienne perturber les autres domaines. Malgré que le FPGA possède plusieurs domaines d'alimentation, il n'y a aucune division en banque des broches de retour vers la masse. Il serait donc impossible d'effectuer une division propre des domaines de puissance.

Aucun plan de masse ne sera spécialisé; tous les plans de masse seront branchés ensemble par les broches des composantes et des trous d'interconnexion. En procédant

---

[5] Microruban (microstrip) : Signal de surface

ainsi, l'impédance du chemin de retour des signaux se trouve à être diminué. Pour les tensions stables, il est prévu que le plan de masse des références des tensions analogiques soit situé sur un plan de surface.

Au total, 16 couches seront utilisées pour la carte mère. Les tensions les moins élevées seront placées au centre de la carte. La carte sera composée de 6 plans de masse (dont deux sur les surfaces), 4 plans de puissance et 6 plans de signaux. La largeur et l'espacement des traces jouent aussi un rôle important dans la stabilité des signaux. Deux traces à transitions rapides trop près l'une de l'autre peuvent causer de la diaphonie. Pour les vitesses de 66 et 125MHz, une largeur standard de 5 millièmes de pouces et une distance de 10 millièmes de pouces sont raisonnables. Cette configuration devrait assurer un minimum d'interférence entre les lignes. L'annexe 6 présente la distribution des couches de la carte mère.

Le premier modèle de carte mezzanine aura 12 couches. De ces 12 couches, 4 seront destinées aux signaux, 3 aux alimentations et 5 pour les plans de masse. Une fois de plus, tous les signaux seront routés à l'intérieur de la carte pour éviter toute émission d'ondes électromagnétiques et les plans d'alimentation seront séparés des plans de masse par un plan condensateur. Les traces auront 5 millièmes de pouces de largeur et seront distancés de 10 millièmes de pouces. La distribution des couches des cartes mezzanines est également présentée à l'annexe 6.

4.3.3 Routage des signaux sensibles

Certains signaux de la carte ont besoin d'une attention particulière lors du routage. Ils peuvent influencer les autres signaux ou être influencé par les autres signaux. Ce sont les horloges et les tensions stables.

4.3.3.1 Les horloges

Les horloges sont probablement les signaux les plus importants de la carte. Chacune des pièces doit recevoir un front d'horloge franc et toutes doivent la recevoir à peu près en même temps. Erinyes possède deux horloges: une de 66Mhz pour la section CPU, l'autre programmable jusqu'à 300MHz placée sur la section FPGA. L'horloge de 66MHz provient d'un générateur, le MPC905, qui possède une sortance (« fanout ») de 6. Deux des sorties sont utilisées pour la distribution de cette horloge sur la carte, l'une se rendant au microcontrôleur, l'autre au FPGA. Le signal d'horloge programmable est de type LVPECL 3.3V. Ce type de ligne nécessite un routage particulier qui demande que les deux lignes complémentaires aient le plus possible la même impédance. Étant de nature différentielle, elle possède déjà une certaine immunité au bruit. Cependant, elle pourrait influencer les lignes qui l'entourent. Cette horloge est par la suite redistribuée vers les mezzanines en un format LVCMOS 1.8V.

Pour éviter toute pollution ayant un rapport avec les horloges, il est préférable que leur chemin soit le plus direct que possible et que ces signaux soient séparés d'une trace au potentiel de la masse d'une largeur de 20 millièmes de pouces. Ceci offrira une meilleure immunité au bruit.

Certains signaux provenant du FPGA et allant sur les mezzanines ont été dédiés comme des transporteurs d'horloges. Ces signaux sont identifiés deux à deux pour permettre la transmission d'horloges différentielles. Tout comme l'horloge programmable, il faut prévoir que ces signaux aient, par paire, une impédance semblable.

4.3.3.2 Les tensions stables

La position des potentiomètres digitaux permet déjà de limiter les interférences provenant des autres signaux. Les courtes lignes sont moins influencées par ce qui les entoure. Il a été prévu que ces potentiomètres soient disposés sur un plan de masse en surface et segmenté pour leur procurer une protection supplémentaire. Il serait intéressant de savoir si la diaphonie est assez importante pour augmenter la distance de ces signaux par rapport aux signaux digitaux. Le calcul de l'indice de diaphonie maximum est donné par [1]:

$$C_{tlk} < \frac{1}{1+(D/H)^2} \tag{4.9}$$

où D est la distance entre le centre de deux traces et H la hauteur de la trace en rapport à un plan de masse. Les traces de la carte auront une largeur de 5 millièmes de pouce, seront espacées les unes des autres d'une distance de 10 millièmes et seront espacés par une distance de 4 millièmes d'un plan de masse. La distance centre à centre de deux trace (D) est donc de 20 millièmes et la hauteur (H) est de 4 millièmes. L'indice de diaphonie devient :

$$C_{tlk} < \frac{1}{1+(D/H)^2} = \frac{1}{1+(0.020/0.004)^2} = 38.46\times10^{-3}$$

ce qui représente un coefficient d'environ 4%. Rappelons le tableau IX du chapitre 3 représentant les fréquences d'oscillation des lignes à délais en fonction de la tension de contrôle.

Tableau XIV

Fréquence d'oscillation des lignes à délais

Tension	*V*	0.74	0.75	0.8	0.85	0.9	0.95	1	1.05	1.1	1.15	1.2	1.25	1.3	1.35
Fréquence	*MHz*	31	39	64	86	103	118	129	138	145	149	153	155	158	160
Tension	*V*	1.4	1.45	1.5	1.55	1.6	1.65	1.7	1.75	1.8	1.85	1.9	1.95	2	
Fréquence	*MHz*	161	163	164	165	166	167	168	169	170	170	171	171	172	

En extrapolant les données du tableau XIV, nous avons la pire variation suivante:

$$\frac{39\text{MHz}-31\text{MHz}}{750\text{mV}-740\text{mV}}=0.8\,\text{MHz}/\text{mV}$$

Les potentiomètres digitaux sont placés sous les mezzanines. Les signaux des mezzanines ont une tension de 1.8V ce qui se traduit, pour 4%, par une diaphonie d'environ 72mV. Une variation de moins d'un mégahertz est souhaitable ce qui correspond à 1.25mV soit environ 0.07% de 1.8V. De l'équation (4.9), il est possible de déduire que :

$$D=\sqrt{\frac{H^2}{C_{tlk}}-H^2} \tag{4.10}$$

$$D=\sqrt{\frac{0.004^2}{0.0007}-0.004^2}\approx 151.11\times 10^{-3}$$

Par conséquent, une distance minimale centre à centre de 150 millièmes de pouces est requise entre les traces de tensions stables et les signaux à 1.8V.

4.3.4 Les connecteurs

Les signaux traversant des connecteurs sont très souvent sujets à subir de la diaphonie. Le faible espacement entre les connexions et le manque d'isolation en sont les

principales causes. Il est important de choisir la position des lignes sensibles qui traverseront les connecteurs. Les connecteurs choisis ont déjà une excellente spécification quant à la diaphonie. Cependant, les tests de diaphonie sur ces connecteurs sont réalisés avec une configuration masse-signal-masse, c'est-à-dire que chaque signal est espacé d'une connexion à la masse. Pour permettre d'obtenir les spécifications donnée par le fabriquant, tous les signaux à l'exception des signaux de capture de température, ont été espacés par la masse. Il fut également jugé bon d'espacer les signaux critiques comme les lignes d'horloge, les lignes de tensions stables ainsi que la ligne de remise à zéro par deux broches connectées à la masse. Le nombre de broches des connecteurs le permet facilement. Ceci a pour but d'augmenter d'avantage l'isolation et de s'assurer que ces signaux ne seront pas altérés par une diaphonie. La distribution des signaux sur les connecteurs est présentée à l'annexe 7.

4.4 Sommaire

La carte de tests est composée d'une carte mère et de deux mezzanines. Les pièces prévues pour le test des futurs démos pourront être ajoutées sur d'éventuelles mezzanines. Les points qui suivent résument les différents aspects importants de la carte.

- La portion CPU est composée d'un microcontrôleur, le ColdFire MCF5272 de Motorola. Il permet de programmer et contrôler le FPGA en plus de permettre une communication avec l'ordinateur auquel la carte est attachée.

- La section FPGA est formée d'un Xilinx XC2V2000. Il contrôle toutes les composantes attachées aux démos comme les potentiomètres digitaux et les capteurs de températures.

- La carte possède plusieurs convertisseurs DC/DC permettant d'abaisser la tension d'entrée de 48V vers les tensions requises par la carte.

- Parce qu'il est le plus rapide et d'interface facile, le mode SelectMAP esclave fut choisi comme méthode de programmation du FPGA. Il a été possible de brancher le FPGA sur le bus du microcontrôleur. Cependant, pour permettre un tel branchement, il a fallu effectuer certaines modifications dans le branchement et la synchronisation du bus du microcontrôleur.

- La carte a 400mm de long sur 250mm de large et sa première mezzanine est de 90mm de coté. Les sections sont clairement séparées les unes des autres pour offrir une bonne isolation entre les différents domaines d'horloges et de tensions et par la même occasion de permettre un routage plus facile.

- La carte de circuits imprimés est formée de 16 couches. De ces 16 couches, 6 sont utilisées pour les signaux, 6 pour les plans de masse dont 2 en surface et 4 pour les plans d'alimentation.

- Les traces ont une largeur de 5 millièmes de pouces et sont espacés de 10 millièmes de pouces.

- Pour respecter les contraintes de synchronisation des mémoires statiques, il a fallu placer certaines puces sur la face opposée pour ainsi limiter la longueur des traces.

- Les horloges sont espacées de 20 millièmes de pouces par une ligne placée au potentiel de la masse pour une isolation accrue.

- Les tensions analogiques sont espacées d'au moins 150 millièmes de pouces des signaux avoisinants (1.8V).

- Les connecteurs choisis ont une excellente isolation contre de genre de phénomène. Pour accroître le potentiel isolateur, les signaux qui traversent le connecteur sont séparés d'un signal à la masse. Les horloges et les tensions analogiques sont espacées d'une broche mise à la masse supplémentaire.

CHAPITRE 5

ERINYES, LOGIQUE

Le chapitre qui suit présente les lignes directrices majeures pour une éventuelle conception de la carte de tests. Il décrit en détail les principaux éléments qui devront être réalisés.

Le logiciel de la carte est composé de deux microprogrammes, un premier pour le FPGA et un second pour le microcontrôleur. Le microprogramme de la section CPU est la porte d'entrée de la carte de tests. Elle permet la communication du système avec l'extérieur et le contrôle sur la carte. Le microprogramme du Virtex a comme tâche le contrôle de l'environnement de test des démos; il constitue le banc de test pour les démos. Le seul microprogramme qui peut changer d'un démo à l'autre est celui du FPGA. Le logiciel de la section CPU devrait rester stable une fois qu'il aura été programmé et débogué.

5.1 Microprogramme

Le pouvoir de test d'Erinyes provient du microprogramme contenu dans le FPGA. Pour arriver à simuler un environnement pour les démos, le Virtex doit contrôler et communiquer avec plusieurs périphériques autour de lui. Pour donner une certaine structure au développement du microprogramme, ce dernier fut divisé en plusieurs parties. Ces parties ne fonctionnent pas toutes à la même fréquence et devront donc parfois être synchronisées sur l'horloge principale. La structure proposée est celle d'un système sur puce (SoC). Cette structure permet de libérer rapidement les modules de contrôle pour ne pas engorger le flot de données. La gestion de la structure peut devenir complexe dû au fait que tous ces modules fonctionnent de façon indépendante. La figure 34 présente une vue schématique de la division du microprogramme.

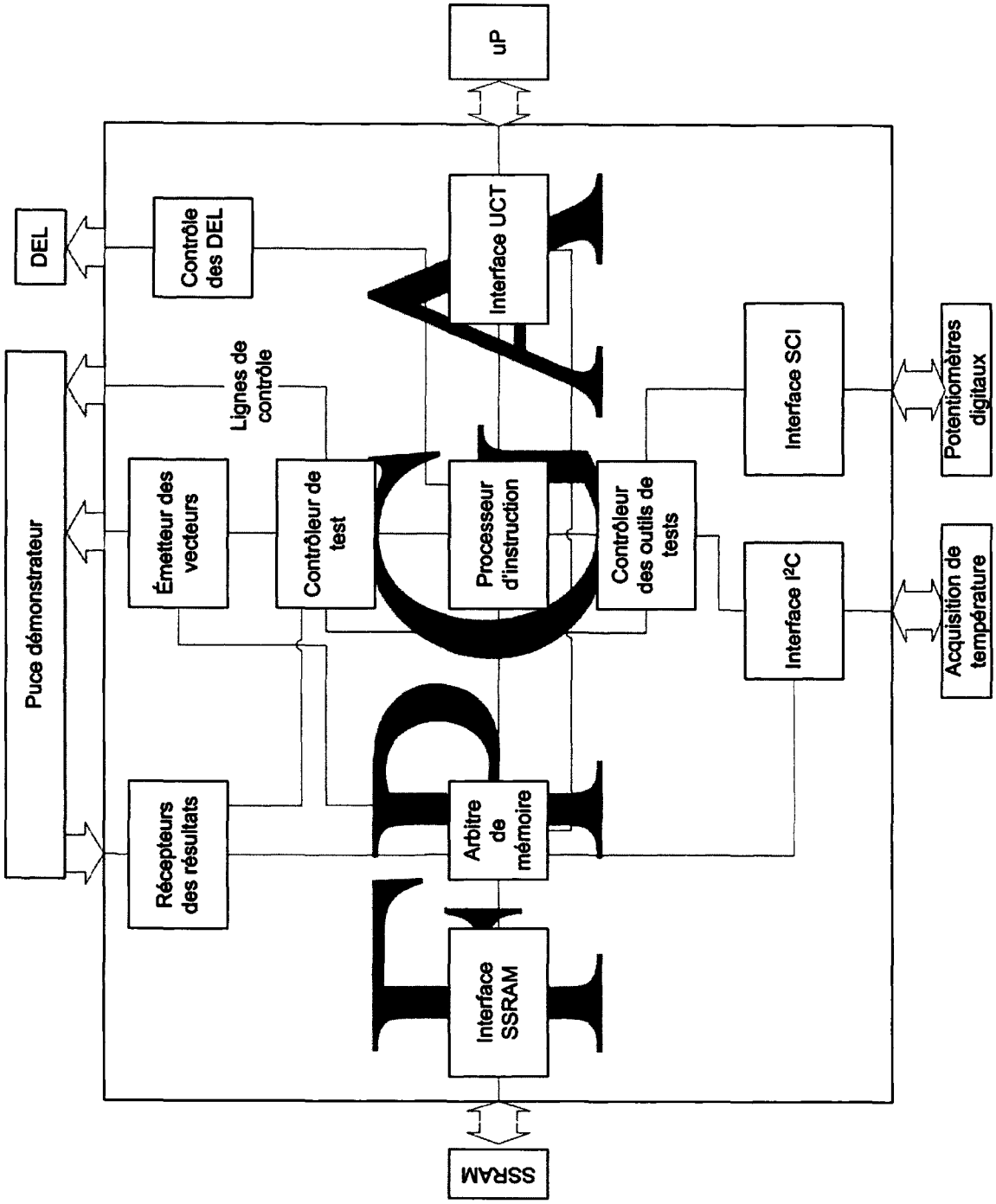


Figure 34 Division schématique du microprogramme

5.1.1 Interface CPU

L'interface CPU permet au FPGA de recevoir ses instructions et ses vecteurs de test. Par cette interface, il lui est également possible de transférer les résultats stockés dans ses mémoires. Cette interface est formée de 23 lignes d'adresses et de 32 lignes de données. Elle possède également une ligne de permission et une ligne d'interruption.

Pour assurer une bonne communication entre le microcontrôleur et l'interface, cette dernière fonctionne à une cadence de 66MHz. Elle permet donc un transfert théorique de 251 mégaoctets par secondes. Les tests du démo 5 demande un bus d'une largeur de 9 bits sur 125MHz pour un total de 134 mégaoctets par secondes. Le microcontrôleur ne permet cependant pas l'exploitation de toute cette bande passante. Ce dernier a besoin de plusieurs coups d'horloges pour effectuer un seul transfert. Il est donc impossible d'effectuer une communication directe entre le FPGA et le microcontrôleur sans l'utilisation de mémoire.

5.1.2 Interface SSRAM

Les mémoires statiques ont comme utilité la sauvegarde temporaire des vecteurs pour le test, et le stockage des vecteurs résultants de ces tests. Pour simplifier le design de cette section, elle doit fonctionner à la même cadence que celle des tests. Cette interface en regroupe en fait 3. Chacune de ces 3 interfaces est indépendante et permet le contrôle d'un maximum de 4 puces de mémoires statiques. Ces interfaces possèdent un bus d'adresses de 19 bits et deux bus de données de 9 bits chacun. Tous les signaux de contrôle des puces sont connectés à cette interface pour permettre d'exploiter toutes les possibilités des puces mémoires. L'horloge des puces mémoires est fournie par cette interface et devrait être de même fréquence que l'horloge principale.

Selon les calculs énoncés dans la section 4.3.1.4, les DCM (« Digital Clock Manager ») devront être utilisés pour minimiser le temps de propagation du signal. L'utilisation de ce mécanisme permettrait d'augmenter la vitesse d'acquisition de ces mémoires en les déphasant les unes par rapport aux autres.

5.1.3 Contrôleur de test du démo

Cette section du microprogramme contient le code du banc de test. Le contrôleur n'est qu'un séquenceur qui commande les différentes interfaces nécessaires au test. Il doit également stimuler les lignes de contrôle des démos. Ce contrôleur est différent pour chacun des démos à tester. Il repose sur deux modules secondaires servant d'interface: l'émetteur et le récepteur de vecteurs. Ces deux modules n'ont comme fonction que de transmettre au démo les vecteurs de test et recevoir du démo les vecteurs résultats. Elles se chargent de les transférer vers les mémoires ou le microcontrôleur lorsque le débit le permet.

5.1.4 Arbitre de mémoire

Ce module permet de contrôler l'accès à la mémoire statique reliée au FPGA. Pour des questions de vitesse, plusieurs périphériques pourront directement accéder à l'espace de stockage prévu. Le microprocesseur et le démo peuvent utiliser la mémoire de façon directe. Pour éviter tout conflit, l'arbitre fait la gestion du trafic des données. Selon la configuration choisie pour les puces de mémoire, il est possible de permettre l'accès aux mémoires pour plus d'un périphérique à la fois. Cet arbitre doit être aussi flexible que l'interface des mémoires statiques le permet.

5.1.5 Contrôleur des capteurs de température

Les capteurs de température demandent une interface I^2C. Cette interface sérielle à 2 fils est idéale pour les communications à basse vitesse. Elle fonctionne à une fréquence maximale de 400kHz et comme cette interface ne supporte qu'un seul maître sur la ligne, aucune ligne de permission n'est requise. Chaque module I^2C possède sa propre adresse logique dans la chaîne. Pour permettre une interruption de haute priorité, une ligne est ajoutée à cette interface. Cette ligne avertira le FPGA que le démo est en train de surchauffer et que les tests doivent être immédiatement interrompus.

5.1.6 Contrôleur des potentiomètres digitaux

Le contrôle des potentiomètres digitaux se fait par une interface sérielle d'une cadence maximale de 50MHz. Elle possède 3 signaux principaux: l'horloge, l'émission des données et la réception des données. Ce type d'interface supporte plusieurs maîtres sur une ligne et est donc muni de signaux de permission. Chaque potentiomètre possède sa propre ligne de permission. Tous les potentiomètres partagent deux lignes supplémentaires: une qui permet de précharger les potentiomètres avec une valeur centrale, l'autre pour avertir le FPGA que la commande est complétée.

5.1.7 Interface DEL

L'interface des diodes électroluminescente permettra de changer la couleur des indicateurs d'états du FPGA. Elle doit supporter plusieurs modes d'opération permettant de sélectionner plusieurs séquences en plus de permettre le contrôle indépendant des diodes.

5.2 Microprogramme de validation

Un microprogramme de vérification de la carte a été écrit pour permettre une validation facile des fonctionnalités de base d'Erinyes. Ce microprogramme contient tous les modules décrits plus haut à l'exception des modules de tests des démos. Le code VHDL du microprogramme est présenté en annexe 8. Le code a été compilé et synthétisé en utilisant les outils Synopsys version 7.2 et Xilinx Foundation Series (ISE) version 4.2i. Le rapport d'utilisation du FPGA donné par ISE est le suivant :

```
Device utilization summary:

   Number of External DIFFMs           1 out of 312     1%
   Number of External DIFFSs           1 out of 312     1%
   Number of External IOBs           262 out of 624    41%
      Number of LOCed External IOBs   63 out of 262    24%

   Number of RAMB16s                   4 out of 56      7%
   Number of SLICEs                  753 out of 10752   7%

   Number of BUFGMUXs                  4 out of 16     25%
   Number of DCMs                      5 out of 8      62%
   Number of TBUFs                   807 out of 5376   15%
```

5.2.1 Interface CPU

L'interface CPU n'est rien d'autre qu'une simple détection d'adresse et d'un synchroniseur. Les données sont envoyées du microcontrôleur sur une horloge de 50MHz. Elles sont par la suite synchronisées sur l'horloge principale de 125MHz pour être transmises au module correspondant à l'adresse transmise. Les deux sources d'horloge sont complètement indépendantes.

Étant donné que la communication entre le FPGA et le CPU s'effectue selon un certain protocole, il n'est pas essentiel que les deux horloges soient en phase. Les données sont capturées sur l'horloge la plus rapide soit 125MHz.

L'interface CPU possède un bus d'adresse et deux bus de données. Cette structure est nécessaire pour permettre une communication bidirectionnelle entre le FPGA et le microcontrôleur. Chacune des interfaces CPU possède ses propres portes à trois états sur le bus de données en sortie pour un branchement commun vers la sortie du FPGA. Les broches en sorties de ce dernier sont également munies de portes à trois états sensibles sur le signal de permission OE du microcontrôleur.

Les figures 35 et 36 montrent deux simulations de l'interface CPU pour le module de contrôle des potentiomètres digitaux. L'horloge de base du FPGA est de 125 MHz et l'horloge du microcontrôleur est de 66MHz. La figure 35 montre un cycle d'écriture à l'adresse 0x2A5000 tandis que la figure 36 montre un accès en lecture à l'adresse 0x2A5004.

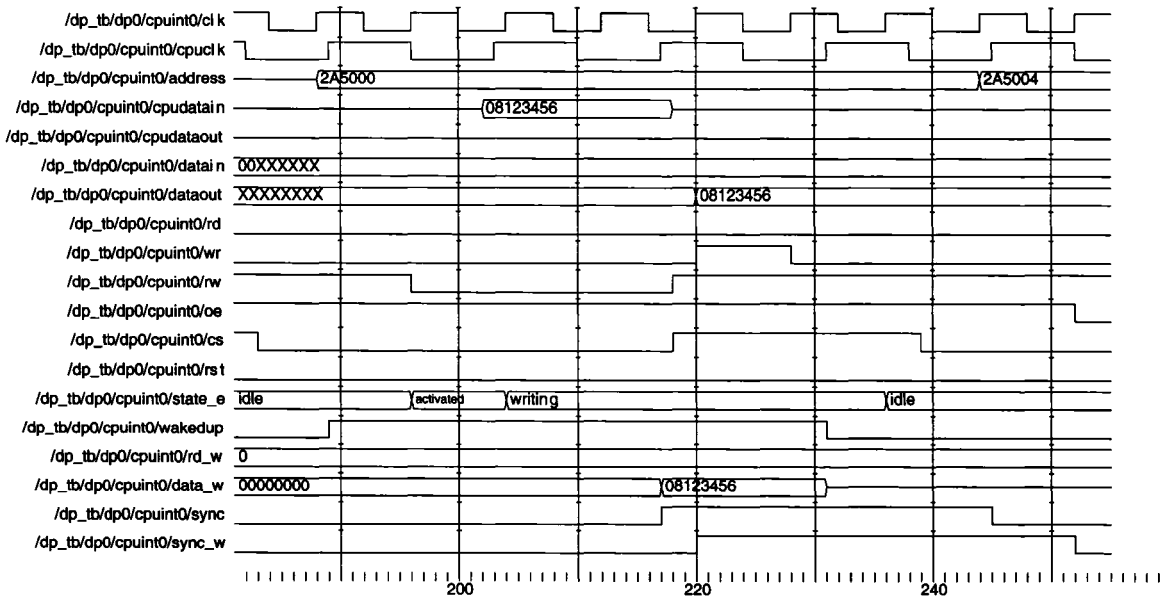


Figure 35 Simulation de l'interface CPU: écriture

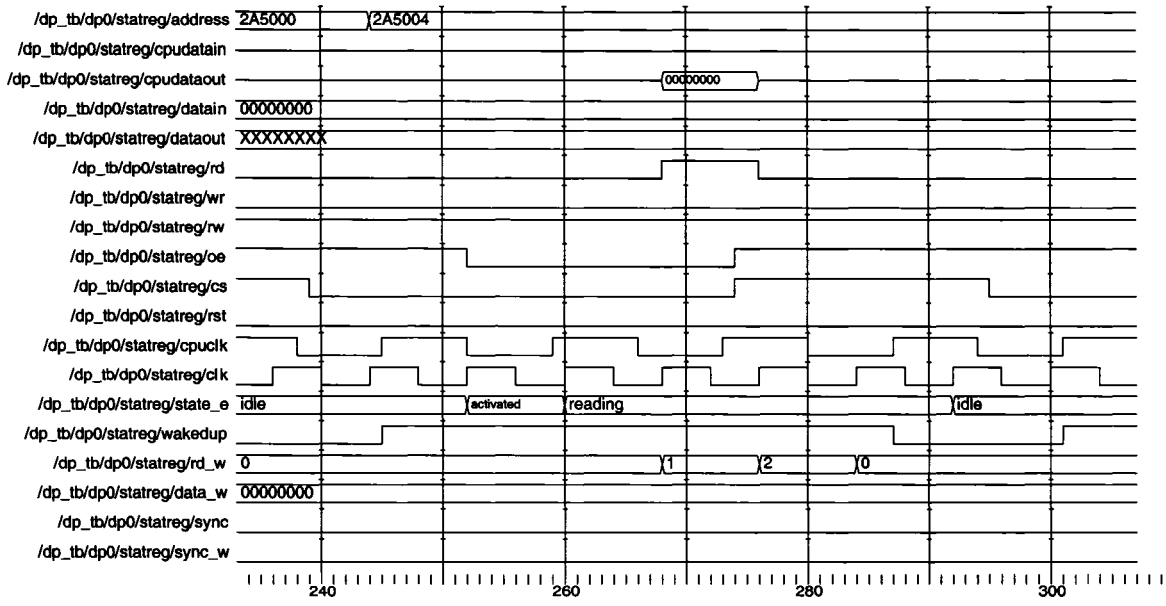


Figure 36 Simulation de l'interface CPU: lecture

5.2.2 Arbitre de mémoire

L'arbitre de mémoire implémenté permet d'accéder à trois banques de mémoire indépendantes de façon simultanée. L'arbitre permet donc au microcontrôleur ainsi qu'à deux autres modules d'accéder aux trois banques de mémoires.

La quantité de mémoire disponible au FPGA dépasse largement la quantité d'adresses disponibles sur le bus du microcontrôleur. Des 23 lignes d'adresse, 19 sont réservées pour l'accès aux mémoires. Comme chacune des banques possède 4 puces mémoires, il aurait été nécessaire d'utiliser toutes les lignes pour permettre d'accéder à une seule banque de mémoire.

Pour permettre à ce dernier d'accéder à la totalité de la mémoire, chacune des puces des différentes banques peuvent être accédées de façon tout à fait indépendante. L'arbitrage se fait par la configuration d'une matrice d'accès. Cette matrice assigne une ou plusieurs banques à une seule interface. La configuration s'effectue par le port du microcontrôleur. Elle permet également de restreindre l'écriture sur un demi mot et d'activer l'une au l'autre des puces mémoire de la banque. Cette configuration est contenue dans six registres : trois pour la configuration des banques et trois pour la configuration des ports d'accès. Les divisions des bits des deux types de registres sont présentées aux tableaux XV, et XVI.

Tableau XV

Bits de configuration des banques de mémoires de l'arbitre

ssramCtrl

Bits	***Nom***	***Description***
5-2	cSel	Sélection des puces de mémoire actives Sélection de type binaire, 1 actif, 0 inactif. (5) - contrôle de la puce 3 (4) - contrôle de la puce 2 (3) - contrôle de la puce 1 (2) - contrôle de la puce 0
1-0	ssramState	Sélection du port d'accès ssramStat=0: banque inactive ssramStat=1: banque connectée à l'interface CPU ssramStat=2: banque connectée à l'interface 1 ssramStat=3: banque connectée à l'interface 2

Tableau XVI

Bits de configuration des ports d'accès de l'arbitre

portCtrl

Bits	***Nom***	***Description***
1-0	byte	Permission d'écriture des données (0) - permission d'écriture sur le mot le moins significatif (1) - permission d'écriture sur le mot le plus significatif 0 = protection contre écriture 1 = écriture permise

Les mémoires sont de type « syncburst » ce qui offre la possibilité d'effectuer une salve de transactions de suite chacune sur un cycle d'horloge. Pour simplifier la gestion de signaux de contrôle, les mémoires sont toujours opérées en mode salve. Le comportement reste le même qu'il s'agisse d'un accès simple ou multiple. Comme l'accès aux mémoires ne peut se faire de façon instantanée, il a fallu insérer deux temps

morts (wait state) au microcontrôleur pour permettre une lecture directe des mémoires par le microcontrôleur.

Chacune des banques possède son propre DCM pour permettre de déphaser l'horloge des puces mémoire. De cette façon, il sera possible de synchroniser les données en tenant compte des délais de propagation des signaux dans la carte de circuits imprimés.

Les figures 37 et 38 montrent une simulation de l'arbitre. La figure 37 présente la configuration de ce dernier. Les trois premier accès programment les permissions d'accès des banques ainsi que la sélection des puces qui seront activées dans la banque. Les trois autres accès configurent les permissions d'écritures des demi mots. Dans cette simulation, la banque 0 est associée aux port CPU, la banque 1 au port 1 et la banque 2 au port 2. La banque 0 aura 2 puces actives soient les puces 2 et 3, la banque 1 en aura une seule : la puce 3 et la banque 1 aura également une seule puce active : la puce 2. La figure 38 montre un accès lecture, un accès écriture et un second accès lecture.

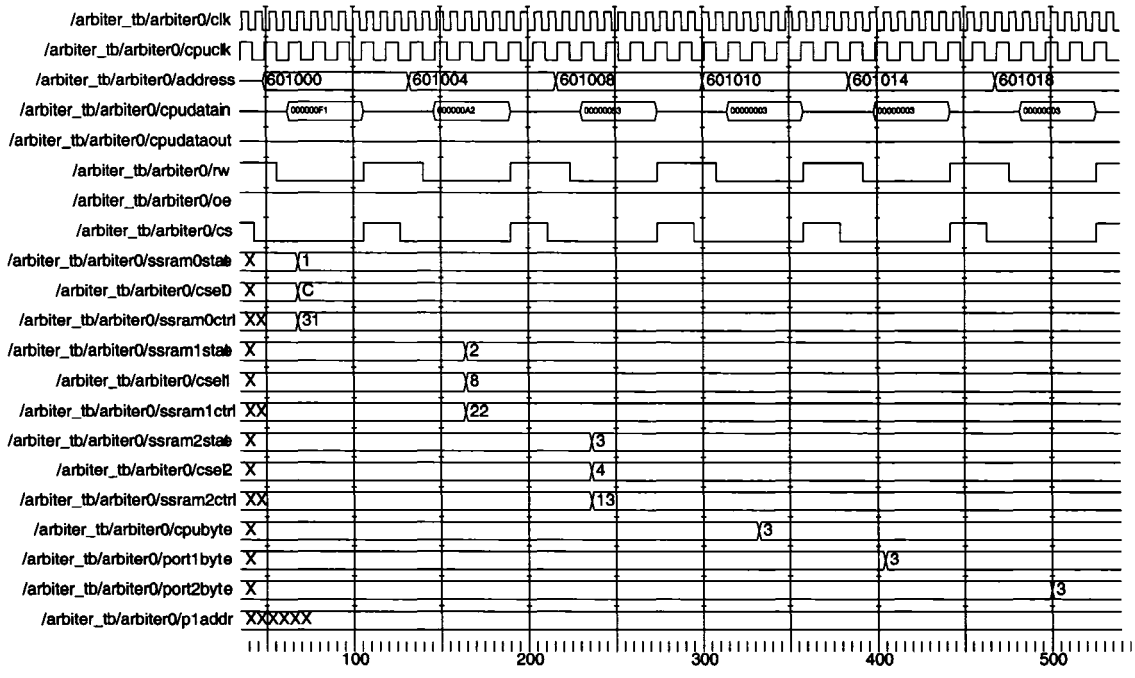


Figure 37 Simulation de l'arbitre: configuration

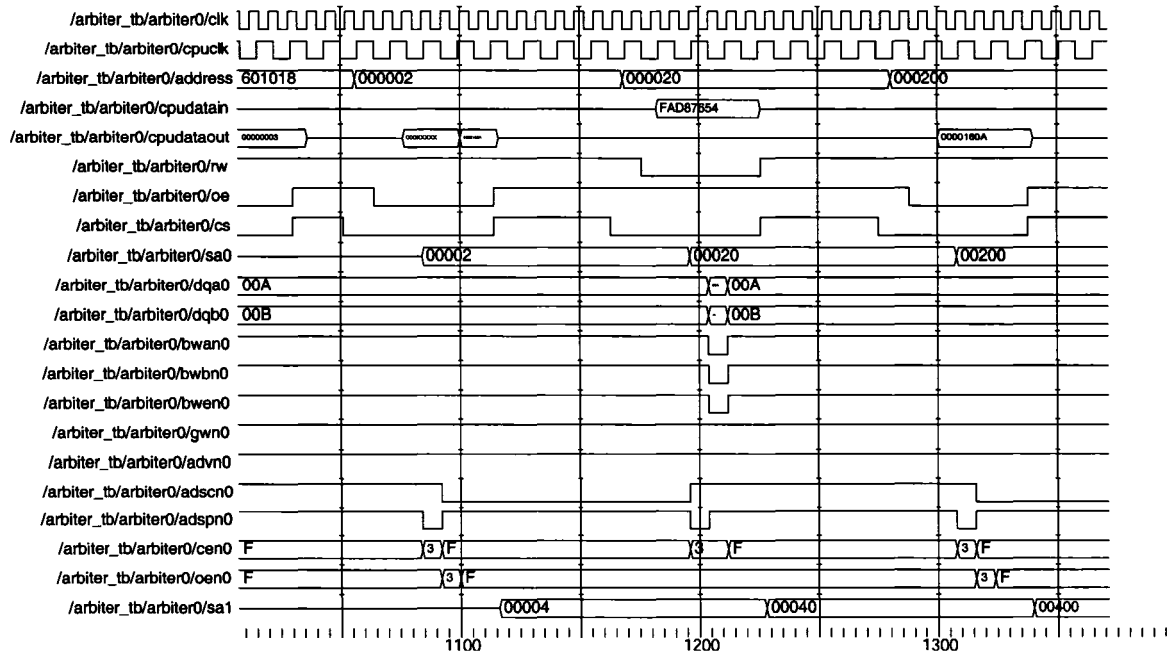


Figure 38 Simulation de l'arbitre: lecture et écriture

5.2.3 Contrôleur des capteurs de température

La communication avec les capteurs de température et le FPGA est très lente par rapport à l'horloge du système. Il était essentiel de concevoir cette interface aussi autonome que possible. Cette interface possède une queue de commandes pour la sortie et une mémoire tampon en entrée. Il est possible d'envoyer plusieurs instructions de suite à l'interface et de recueillir les résultats plus tard. La commande envoyée à l'interface est d'une largeur de 24 bits. La segmentation de la commande est présentée au tableau XVII. Un registre de statut permet de connaître l'état de l'interface. La description de ce registre est donnée au tableau XVIII.

Tableau XVII

Bits de commande de l'interface des capteurs de température

commande

Bits	***Nom***	***Description***
23-17	adr	Adresse du capteur
16	RW	Commande RW=0: lecture RW=1: écriture
15-0	dta	Donnée à transmettre

Tableau XVIII

Bits du registre de statut de l'interface des capteurs de température

tempSensStatus

Bits	***Nom***	***Description***
1	alert	Drapeau d'alerte de température.
		alert=0: comportement normal
		alert=1: capteur en alerte
0	dataRdy	Indicateur de donnée
		dataRdy=0: aucune donnée n'est prête dataRdy=1: une donnée est prête à être lue.

Les figures 39 et 40 montrent une simulation lors d'un mode d'opération normal. À la figure 39, une commande d'écriture est envoyée au capteur 0x42. Le train de bit transmis vers les capteurs est illustré à la figure 40. Cette dernière figure montre que la transmission des données s'effectue en trois temps; un par tranche de huit bits. Pour chacune de ces tranches, le capteur doit répondre en maintenant sa ligne de donnée basse sinon, la transmission est interrompue et la commande oubliée. Pour chacune des commandes transmises, un résultat est placé dans le tampon des résultats. Pour s'assurer que la commande est bien reçue, le microcontrôleur pourra lire le résultat dont la forme est présentée au tableau XIX. Il est à noter que lors d'une erreur dans la transmission, le champ *dta* des résultats montre le reste de la commande non transmise. La figure 42 présente une simulation de lecture d'un résultat d'une commande interrompue.

Tableau XIX

Bits du résultat de l'interface des capteurs de température

résultat

Bits	*Nom*	*Description*
23-17	adr	Adresse du capteur
16	statut	Statut de la commande envoyée statut=0: aucune erreur statut=1: une erreur s'est produite dans la transmission
15-0	dta	Données reçues lors d'une commande de lecture

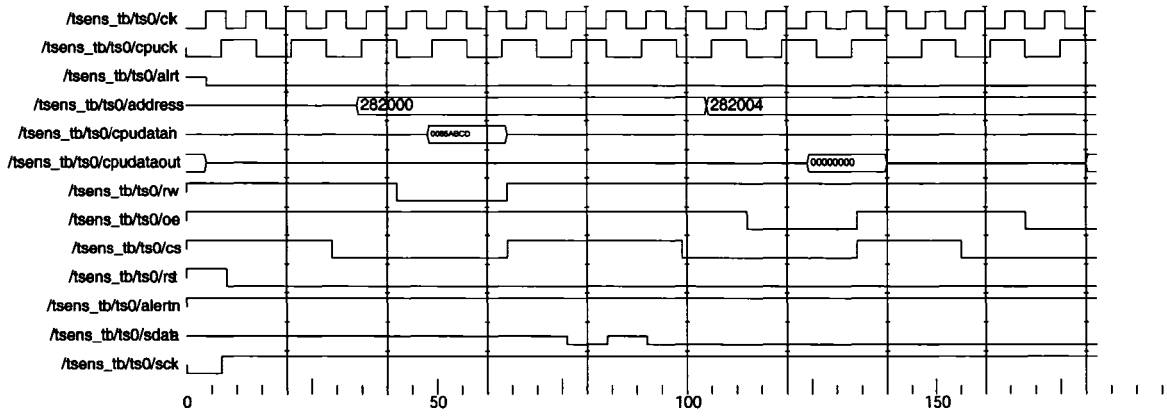


Figure 39 Simulation de l'interface des capteurs de température: émission de la commande

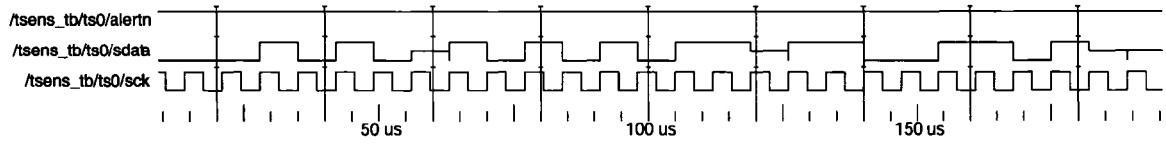


Figure 40 Simulation de l'interface des capteurs de température: transmission de la commande

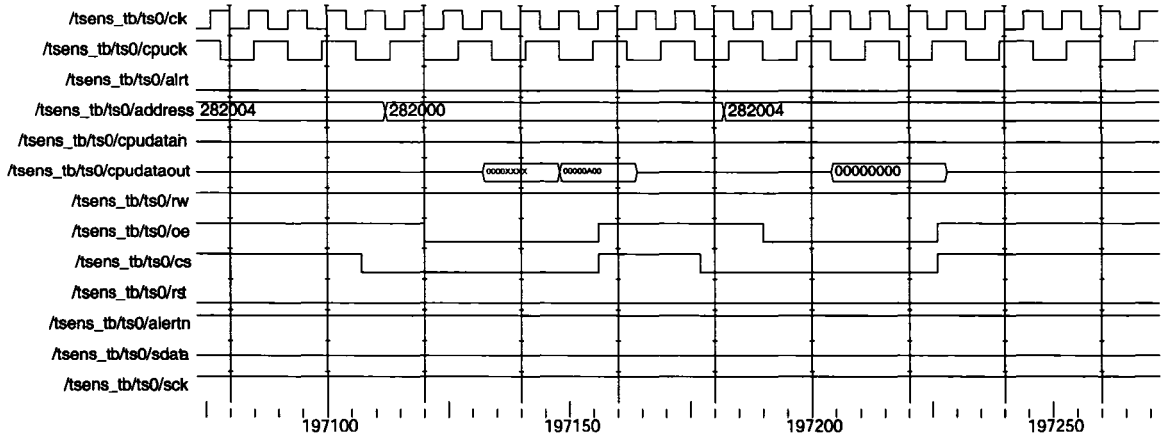


Figure 41 Simulation de l'interface des capteurs de température: lecture du résultat

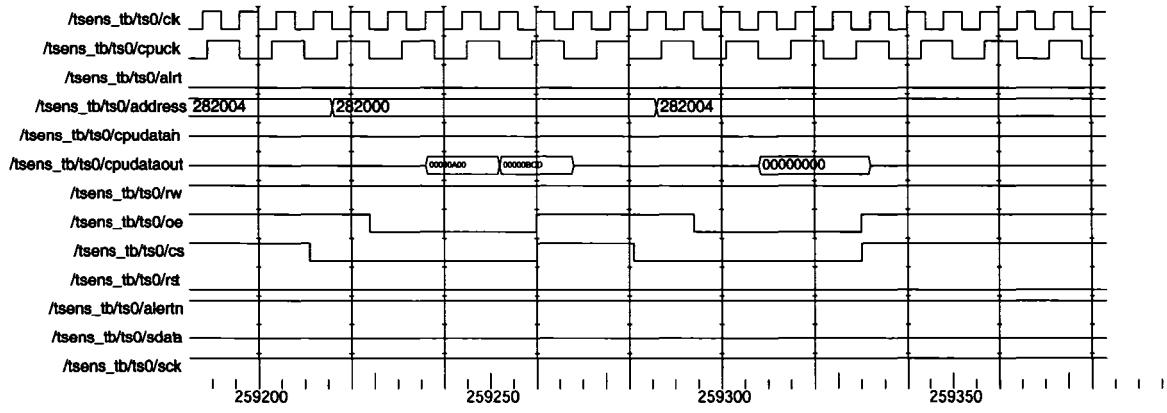


Figure 42 Interface des capteurs de température: commande interrompue

Les capteurs de température peuvent avertir le FPGA que l'une des puces sous test est en surchauffe. Dans un tel cas, l'interface ajoutera automatiquement une commande de réception de l'alerte dans la queue de transmission. Cette commande est en fait une adresse réservée à cette fin. Lorsque l'adresse 0xC est envoyée, le capteur en alerte retourne son adresse plus un bit supplémentaire dans le LSB dans le champ de donnée. Ainsi, le résultat de la commande sera formé de l'adresse 0xC, le bit de statut et finalement l'adresse du capteur en alerte. L'alerte est également signalée au FPGA par

le registre de statut et par un signal interne qui pourra être branché à la ligne d'interruption du microcontrôleur.

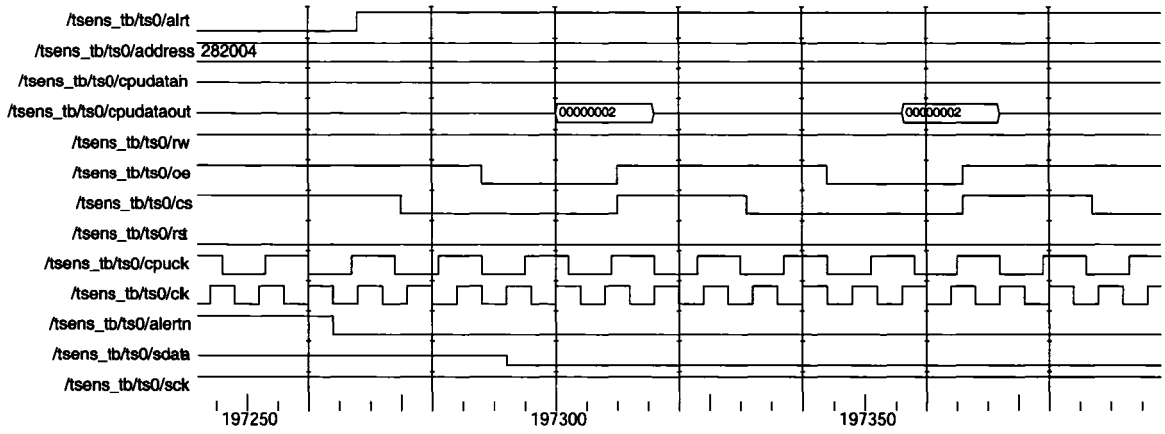


Figure 43 Interface des capteurs de température: génération d'une alerte

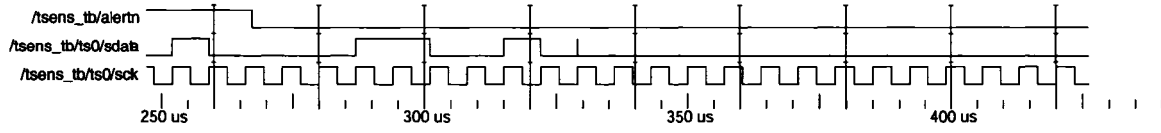


Figure 44 Interface des capteurs de température: réception d'une alerte

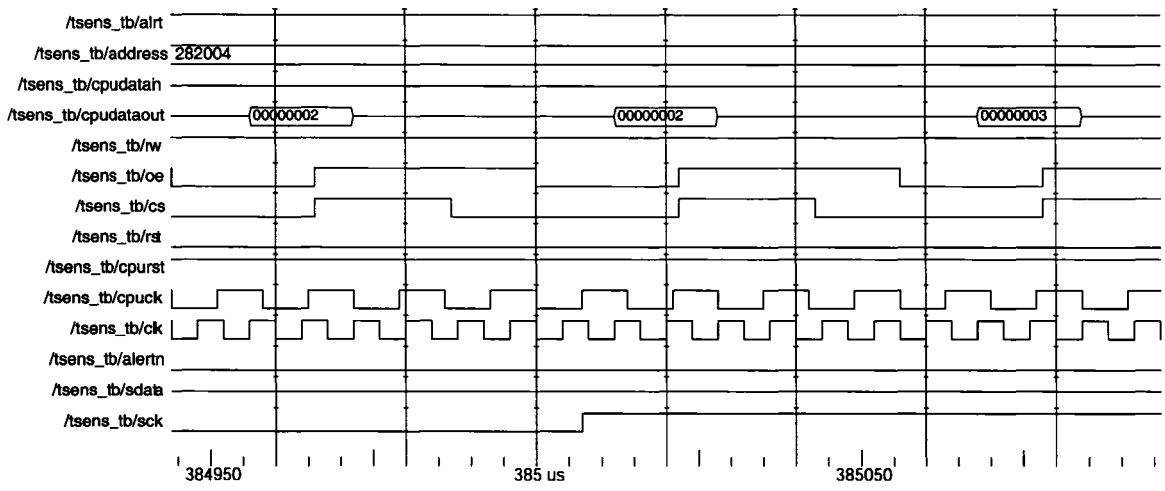


Figure 45 Interface des capteurs de température: capture de l'alerte

L'horloge de l'interface des capteurs de température est générée par un compteur de cycles de l'horloge du microcontrôleur. Les délais de propagation sur une horloge de 100KHz sont négligeables et par conséquent, il reste acceptable de procéder ainsi.

5.2.4 Contrôleur des potentiomètres digitaux

Tout comme l'interface des capteurs de température, l'interface des potentiomètres digitaux est tout à fait autonome. Elle reçoit sa donnée à transmettre, la place dans une queue et place les résultats dans un tampon prévu à cet effet pour une lecture ultérieure. Le format de la commande est donné au tableau XX et le format du mot de statut au tableau XXI.

Tableau XX

Bits de commande de l'interface des potentiomètres digitaux

commande

Bits	***Nom***	***Description***
28-24	dev	Adresse du potentiomètre
23-0	dta	Donnée à transmettre

Tableau XXI

Bits du registre de statut de l'interface des potentiomètres digitaux

tempSensStatus

Bits	***Nom***	***Description***
0	dataRdy	Indicateur de donnée
		dataRdy=0: aucune donnée n'est prête dataRdy=1: une donnée est prête à être lue.

Les figures 46, 47 et 48 présentent une simulation de l'interface. La figure 13 montre la réception de la commande par l'interface et le début de la transmission. La commande 0x123456 est transmise à la puce 8 qui est confirmée par la position du bit de la sélection de puce (spiCs). Le protocole SPI fonctionne sur le principe que lorsqu'il y a une donnée transmise, il y a simultanément une donnée qui est reçue. La figure 47 montre la transmission des bits sur spiDo ainsi qu'une simulation de la réception d'une donnée sur spiDi. La figure 48 montre enfin la lecture de la donnée présente dans le tampon de réception.

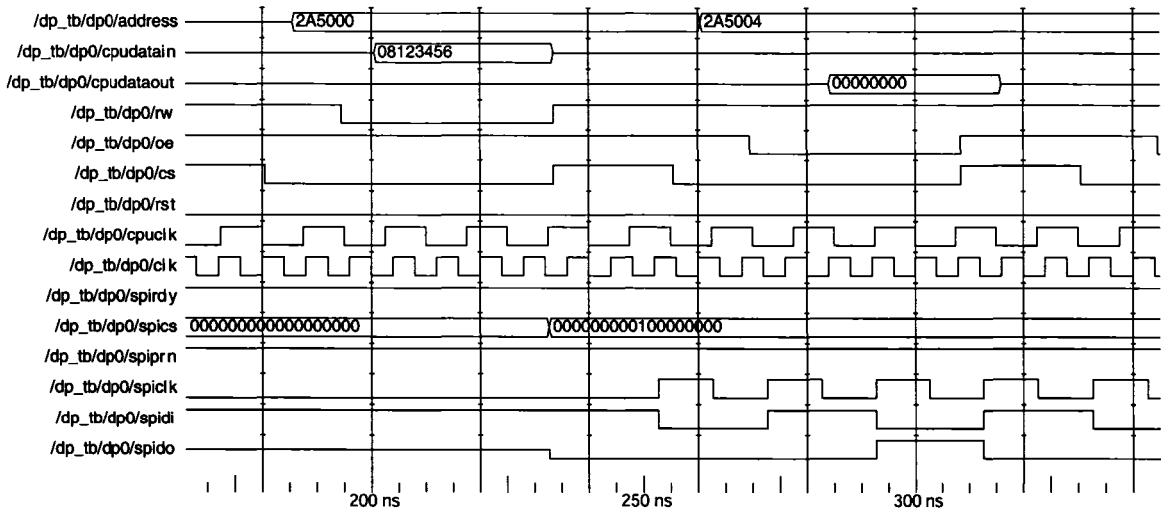


Figure 46 Interface des potentiomètres digitaux: commande d'écriture

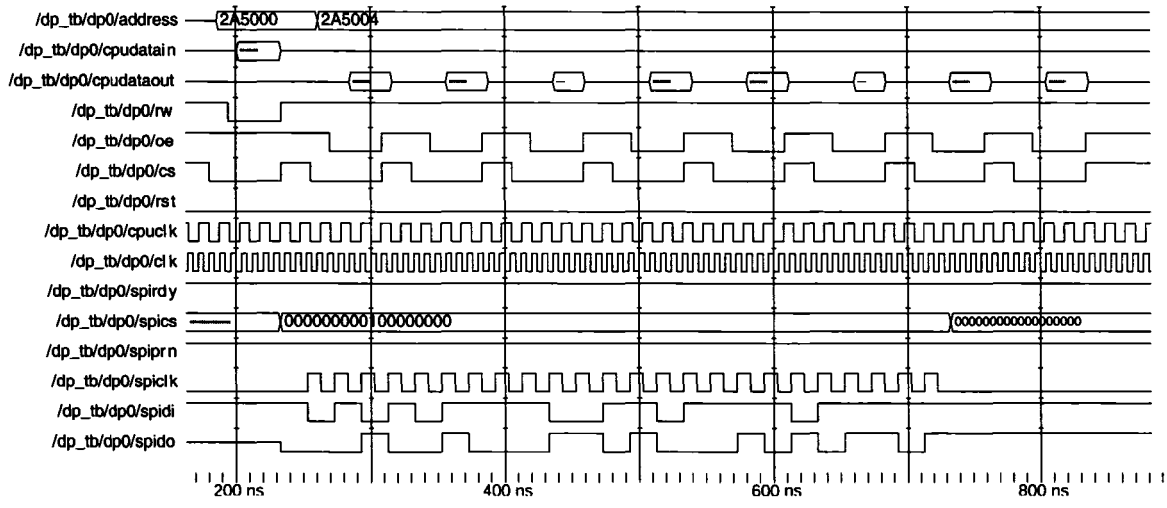


Figure 47 Interface des potentiomètres digitaux : transmission

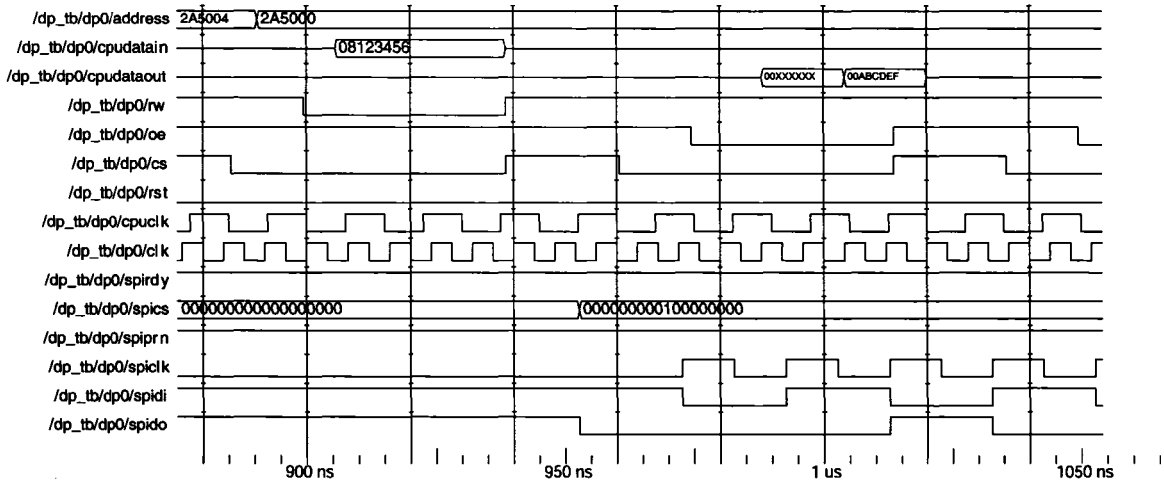


Figure 48 Interface des potentiomètres digitaux : lecture

Cette interface a recourt à un DCM pour la génération de son horloge. En effet, puisque celle-ci est de fréquence relativement élevée, les délais de propagation ont une influence non négligeable. Le DCM utilise l'horloge de 66MHz du microcontrôleur pour générer une fréquence de 50MHz qui sera utilisée par l'interface.

5.2.5 Interface DEL

L'interface DEL permet de placer les diodes électroluminescentes dans plusieurs états. Le banc de test de l'interface en défini 7 : quatre états de clignotement entre les deux couleurs, un état d'une couleur, un état de l'autre couleur et finalement un état hors circuit. Les états de clignotements sont générés par un compteur de 26 bits sur l'horloge de 66MHz. Les bits les plus significatifs sont branchés aux diodes pour offrir une fréquences d'oscillation minimale d'environ 2Hz. La figure 49 montre la transmission de la commande à l'interface et la figure 50 montre le résultat sur les différentes lignes des diodes. Il et à noter que pour la simulation, la fréquence de 66MHz est divisée par un facteur de 256 dans le but de limiter le temps de simulation.

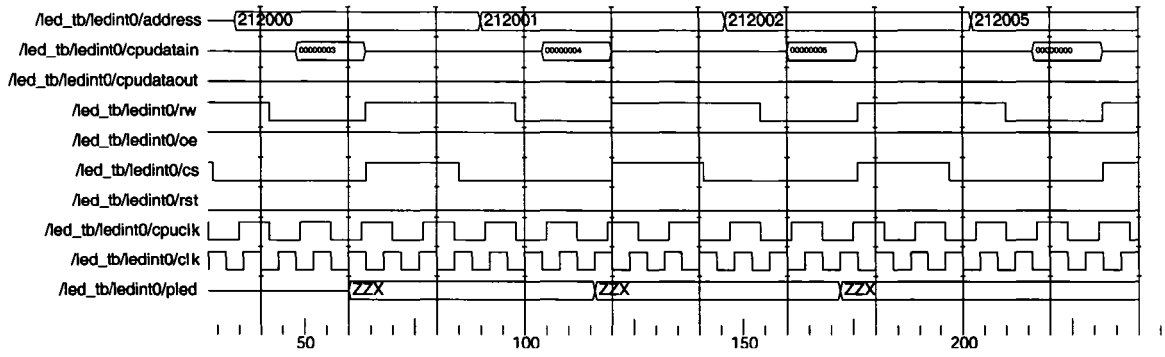


Figure 49 Simulation de l'interface DEL: commande

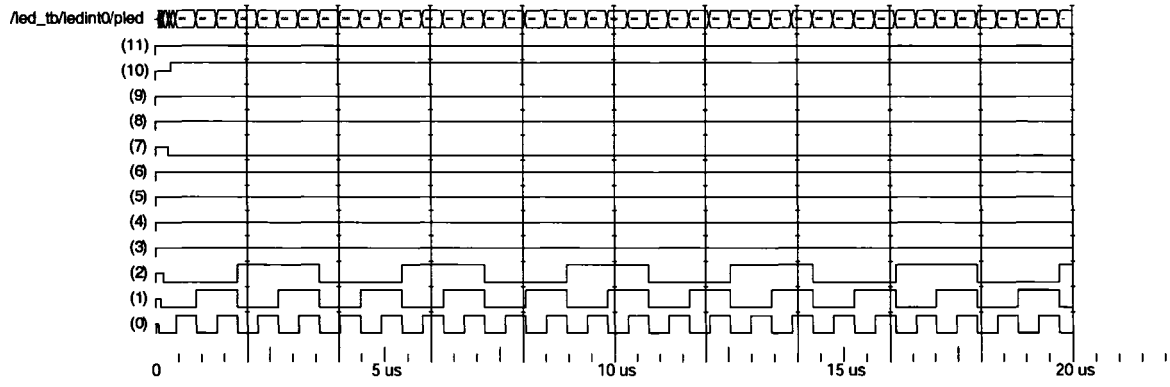


Figure 50 Simulation de l'interface DEL: résultat

5.3 Logiciel

Étant donné que le microcontrôleur ne prend aucunement part au contrôle du test, il n'est pas nécessaire d'opérer en temps réel. Tel qu'il a été mentionné dans la section 4.1.1.1, le microcontrôleur exécutera le système d'exploitation µCLinux. Cette distribution possède la majorité des fonctionnalités de la distribution Linux. Pour le développement des logiciels de la carte, cette distribution a recourt à la transcompilation. La transcompilation est l'habilité de concevoir du code destiné à fonctionner sur une certaine architecture de microprocesseur en utilisant une autre architecture. Ainsi, il sera

possible de créer des applications pour le ColdFire à partir d'un PC. Pour ce faire, l'ensemble de développement de μCLinux doit être installé sur un ordinateur roulant Linux. Cet ensemble de développement est tout à fait gratuit et comprend le noyau, le système d'exploitation et les outils pour la transcompilation.

5.3.1 Modifications du noyau

Le noyau de base de la distribution de μCLinux a été développé pour supporter un grand nombre de cartes de développement. Dans le cas du ColdFire MCF5272, il est possible d'utiliser 4 configurations de base. La configuration qui sert de patron est celle de la carte de développement MC5272C3 de Motorola. Comme il a été vu précédemment, Erinyes diffère légèrement de cette carte de développement. Par conséquent, le noyau Linux devra être modifié pour tenir compte de ces différences. La plupart des options du noyau peuvent facilement être ajoutés ou retranchés par l'intermédiaire du script de configuration. Ce script offre une interface graphique ou textuelle conviviale qui permet aisément de naviguer à travers la configuration. Cependant, pour certaines modifications, il sera nécessaire de modifier et même de créer de nouveaux fichiers pour compléter certaines parties du code. Lors de la création des fichiers, il faudra s'assurer que tous les fichiers Makefile associés au répertoire de création des nouveaux fichiers les incorporent dans la compilation.

La distribution de μCLinux comprend deux versions du noyau de Linux. La version première est la 2.0.0. La branche 2.0 existe depuis relativement longtemps. Elle est encore disponible pour les systèmes existants développés sur cette version. La version 2.0 n'est plus augmentée depuis quelques temps et par conséquent, ne supporte pas tous les périphériques existants sur les cartes de développement. Erinyes devra utiliser la branche 2.4 du noyau. Cette version, plus récente, est probablement légèrement moins

stable que la version 2.0 mais a comme principal avantage de supporter la majorité des périphériques. Elle ne demandera que des modifications mineures.

Pour garder la structure du noyau intacte, les modifications doivent être faites de façon méthodiques et à l'image de ce qui est déjà fait. L'utilisation exhaustive des variables de type #DEFINE ainsi que les directives de compilation #IFDEF et #IFNDEF est fortement recommandée. La procédure à suivre est partiellement décrite dans la distribution de μCLinux[6]. Il est fortement conseillé de lire également le fichier de méthodologie de code de Linux[7]. Avant même d'amorcer les modifications telles quelles, il faut ajouter Erinyes aux modèles de cartes basées sur le MCF5272. Une fois ajoutée, les fichiers peuvent être modifiés pour refléter l'architecture de la carte.

La branche des services où les pilotes devraient être écrits est la branche char. Cette branche définit les pilotes de traitement de caractères comme les ports séries, parallèles, les consoles etc. La création telle quelle des pilotes dépasse les objectifs de ce document et est laissée à la discrétion du lecteur.

5.3.1.1 Espace mémoire

La première modification est de configurer l'espace mémoire que le microcontrôleur peut utiliser. La mémoire dynamique et la mémoire flash de démarrage devront être positionnées chacune au bon endroit. Ces données se retrouvent dans les fichiers crt0_ram.S et ram.ld du répertoire vendeur nouvellement créé. Le fichier crt0_ram.S contient une variable permettant de spécifier directement la quantité de mémoire vive contenue sur la carte. Il serait préférable de l'utiliser plutôt que de se fier à la routine de détection automatique de la mémoire.

---

[6] cf. fichier uClinux-dist-20020701/uClinux-dist/Documentation/Adding-Platforms-HOWTO.

[7] cf. fichier uClinux-dist-20020701/uClinux-dist/linux-2.4.x/Documentation/CodingStyle.

L'espace de stockage en mémoire de type flash est déjà implémenté dans le noyau de µCLinux. Il est nécessaire de spécifier la compagnie de mémoire flash utilisée par Erinyes dans le script de configuration. Erinyes utilise les mémoires de la compagnie AMD. Une fois la mémoire flash sélectionnée, la largeur des mots et la dimension peuvent être laissées à la détection automatique. Les algorithmes de détection de la dimension se basent sur les informations des mots de contrôle des mémoires flash. Cette méthode de détection est des plus fiables.

Le système d'exploitation possède un système de fichiers spécifiquement écrits pour les mémoires flash. Ce support est nommé ROMFS et possède plusieurs variantes de système de fichiers. Pour permettre de supporter les 3 puces de mémoires flash, il faut modifier le fichier blkmem.c[8] pour inclure les plages mémoires et créer les fichiers périphériques. Le reste des modifications se fait dans le script de configuration dans les sections ROMFS et MSDOSFS. Comme le système d'exploitation n'utilise pas la totalité de la mémoire flash du système, il est conseillé d'y réserver une certaine plage pour le stockage des logiciels d'application. Il serait par exemple possible de la séparer en deux : un mégaoctet pour le noyau et 3 pour les applications.

Étant donné que la mémoire de type flash ne permet d'effacer ses cases qu'un nombre limité de fois, il est proposé de prévoir un espace de stockage temporaire en mémoire RAM en utilisant un disque virtuel. Lorsque le fichier doit être conservé, il pourra être copié dans la mémoire flash. Il est possible d'ajouter le support pour les disques virtuels par le script de configuration du noyau. Une taille de 4 à 8 mégaoctets est raisonnable.

---

[8] cf. répertoire uClinux-dist-20020701/uClinux-dist/linux-2.4.x/drivers/block/

5.3.1.2 Interfaces de communication

La carte de développement supporte 2 ports série de type EIA-RS232 alors qu'Erinyes n'en possède qu'un seul. Linux détecte lui-même la validité des deux ports série par le test des registres de statut de l'interface. Comme les ports séries du ColdFire ont une double fonction, il faut tout d'abord s'assurer que les broches du port 0 soient configurées comme un port série alors que les signaux du port 1 doivent être configurés comme des entrées/sorties directes. Le port 1 devrait également être enlevé du fichier de configuration des ports séries pour éviter une auto-détection. Tout ce travail est effectué dans la routine mcfrs_irqinit() du fichier mcfserial.c[9].

Le support pour l'interface I^2C est partiel pour le MCF5272. Le microcontrôleur ne supporte pas directement cette interface. Les routines sont définies en utilisant des fonctions encapsulées qui contrôlent les signaux, ce qui permet l'émulation facile et rapide de l'interface. Tous les fichiers se retrouvent dans le répertoire /uClinux-dist-20020701/uClinux-dist/linux-2.4.x/drivers/i2c. Un nouveau fichier devra être écrit à partir d'une des définitions existantes. Le fichier de base recommandé est celui qui émule cette interface à partir du port parallèle: i2c-philips-par.c. Le fichier i2c-core.c devra également être modifié pour permettre d'inclure les bonnes fonctions d'encapsulation.

Le noyau de µCLinux supporte déjà une interface pour les diodes électroluminescentes. Le fichier de support à modifier est ledman.c[10]. Il faut s'assurer que le port A du microcontrôleur est bien placé en sorties directes.

---

[9] cf. répertoire uClinux-dist-20020701/uClinux-dist/linux-2.4.x/drivers/char/

[10] cf répertoire uClinux-dist-20020701/uClinux-dist/linux-2.4.x/drivers/char/

5.3.1.3 Communication FPGA

Le branchement du FPGA au microcontrôleur est l'interface qui demande le plus de modifications. Étant donné qu'il est branché sur le bus du MCF5272, il est possible d'y accéder directement par une écriture en mémoire. Cependant, pour conserver la structure du noyau et pour utiliser le service des interruptions, il faut créer un pilote. Par ce pilote, il serait alors possible de communiquer avec le FPGA de la même façon qu'il est possible de communiquer avec tous les autres périphériques.

Il sera nécessaire de créer 2 gestionnaires de périphériques pour les communications du Virtex. Comme il y a deux protocoles de communication distincts, un pour la programmation et l'autre pour la communication, il faudra créer deux pilotes de communication. Une attention particulière doit être prise pour l'assignation du mode des broches à double fonction du port A et B du microcontrôleur.

5.3.1.4 Contrôle de la température

Il sera nécessaire de créer un autre pilote pour permettre les interruptions du capteur de température attaché au FPGA. Ce pilote dépend de l'interface I2C et par conséquent, doit être installé une fois l'interface initialisée. Le pilote devra prendre en charge les deux interruptions et agir immédiatement lorsqu'une d'entre elles survient soit en forçant le ventilateur à sa vitesse maximale ou en plaçant le FPGA en mode dormant. Cependant, ce pilote ne devrait pas être responsable d'établir la vitesse de rotation du ventilateur; un *daemon*[11] s'en chargera. Il devra également permettre la lecture de la température ainsi que l'écriture de l'hystérésis des alarmes.

---

[11] Daemon: logiciel performant certaines tâches en arrière-plan sans intervention humaine

5.3.2 Logiciels d'application

Après les modifications au noyau du système d'exploitation, il faut développer les applications qui permettront de contrôler la plateforme de test. Pour que l'interfaçage soit plus convivial, il serait préférable d'écrire un seul logiciel qui permettra de programmer le FPGA, programmer l'horloge, lire la température du FPGA, et de contrôler le test. Cette application devra posséder une interface usager textuelle qui permettra l'utilisation de la console comme moyen de communication. L'avantage d'utiliser une interface textuelle est qu'il sera possible d'effectuer les tests à partir des deux interfaces de communication en utilisant, sur le PC hôte, des applications de communication par port série ou tout simplement par un logiciel supportant le protocole telnet.

Le transfert des vecteurs de test se fera par le protocole FTP (File Transfer Protocol). Les fichiers pourront être stockés sur un disque virtuel ou encore dans la mémoire flash des données. Le logiciel d'application pourra aller lire ces fichiers et transmettre les vecteurs au FPGA. Il recueillera par la suite les résultats pour les sauvegarder dans un second fichier qui pourra être stocké en mémoire puis transféré vers l'ordinateur hôte.

Pour permettre de contrôler la température du FPGA, il faut concevoir un petit logiciel qui travaillera en arrière plan pour contrôler la vitesse de rotation du ventilateur. Ce *daemon* devra périodiquement vérifier la température et modifier la vitesse du ventilateur en conséquence. Il pourra également lancer des messages diagnostiques dans les fichiers journaux par l'entremise du *daemon syslogd*.

5.4 Sommaire

Le microprogramme du FPGA est le centre nerveux de l'environnement de test des démos. Il contrôle chacune des composantes qui y sont reliées. Ce logiciel est subdivisé en plusieurs parties logiques. Ces parties logiques agiront de façon indépendante à la manière d'un SoC pour permettre de libérer les interfaces de contrôle. Les parties du microprogramme du Virtex sont: l'interface CPU, l'interface des mémoires statiques, le contrôleur de tests des démos, l'arbitre de mémoire, le contrôleur des périphériques, interface des diodes électroluminescentes et le processeur d'instructions. Le code de vérification de la carte Erinyes renferme déjà les blocs de base qui formeront le microprogramme. Toutes les interfaces à l'exception de celle des puces démo ont été écrites et simulées.

Le logiciel de la section du microcontrôleur est le système d'exploitation Linux dans sa version embarquée. Pour lui permettre de fonctionner correctement et d'exploiter toutes les fonctionnalités de la carte, il faut modifier certaines parties de son code. La plupart des options sont insérées ou retirées du noyau par un script de configuration. Cependant, certaines modifications requièrent des changements dans les fichiers source existants et parfois même la création de nouveaux fichiers. Ces modifications doivent tous être faits en accord avec la structure du noyau.

Il faut modifier Linux pour permettre d'utiliser toute la plage mémoire disponible. Les modifications permettront également d'utiliser les puces de mémoires flash comme des disques de stockage. Linux doit aussi être modifié pour enlever la seconde interface sérielle, pour permettre d'accéder aux périphériques de l'interface I2C et pour permettre le contrôle des diodes électroluminescentes. Il est nécessaire d'ajouter des pilotes pour communiquer avec le FPGA et son contrôle de température.

Une fois les microprogrammes terminés, les logiciels d'applications doivent être écrits. Un logiciel de contrôle des tests permettra à l'utilisateur de la carte de charger les vecteurs, lancer les tests et recueillir les résultats. Le contrôle de la température du FPGA sera fait par un *daemon*.

CONCLUSION

Le taux d'utilisation de l'Internet ne cesse d'augmenter. Les infrastructures en place sont déjà dépassées et bientôt, elles ne suffiront plus à la demande. Pour s'attaquer à ce problème, la compagnie Hyperchip développe des routeurs capable d'offrir un débit 1000 fois supérieurs à celui des routeurs présentement disponibles. Pour atteindre ce débit, la compagnie a développé un système hautement parallélisé comprenant plusieurs circuits distribués sur quelques cartes. Mais pour pouvoir rester compétitive et continuer de combler les besoins en routage, il lui faut trouver des moyens d'accélérer d'avantage l'architecture de ses routeurs actuels.

Pour conserver leur avance, un groupe de recherche d'Hyperchip s'est tourné vers l'intégration directe sur tranche. Cette technique consiste à concevoir un système complet à l'intérieur d'un immense circuit intégré. Comme les signaux ne changent jamais de milieu, ils voyagent à travers le système à la vitesse maximale permise par la technologie de fabrication de la puce. Cependant, la technique d'intégration directe sur tranche comporte de nombreux problèmes. Le groupe de recherche a donc développé certaines stratégies permettant de les contourner. Quelques-unes de ces idées ont été implantées dans des démonstrateurs qui n'avaient jusqu'ici jamais été testés. Pour prouver la validité de ces stratégies, il est nécessaire d'effectuer la vérification de ces puces.

Pour procéder à la vérification des démos, il a fallu concevoir une carte capable de les analyser. Elle devait être assez flexible pour permettre le test de futurs démonstrateurs dont les spécifications ne sont pas encore connues. Pour le moment, il n'y a que 2 démonstrateurs pour lesquels il existe une spécification: le 4e et le 5e démonstrateur.

Le démo 4 s'attaque au problème de la conception même d'un système à intégration directe sur tranche. Les méthodes de conception des circuits intégrés ne sont pas

parfaites. Elles sont un ensemble de procédés complexes au cours desquels des erreurs peuvent se glisser affectant directement le rendement de fabrication. Il est, à toute fin pratique, impossible d'obtenir une tranche sans aucune défectuosité. De ce fait, il serait impossible de concevoir un système d'intégration directe sur tranche sans mécanisme de tolérance aux défectuosités. Ce démo renferme des structures de reconfiguration permettant de survivre à une ou plusieurs défectuosités dans la grille de cellules d'un système hautement parallélisé.

Outre les problèmes de manufacture, l'intégration directe sur tranche apporte des problèmes dus à la dimension du circuit imprimé. Le 5e démonstrateur offre des structures permettant de tester certains problèmes reliés à la distance que doit parcourir un signal à travers une puce de grande dimension, ainsi qu'à travers une puce qui subit un gradient de température important. Dans une puce de petite dimension, l'effet de gradients est négligeable mais sur un système d'intégration directe sur tranche, ces gradients peuvent causer d'importantes contraintes mécaniques et des erreurs de synchronisation.

Pour amener à la conception d'une carte capable d'effectuer tous ces tests, une spécification préliminaire fut définie. Cette spécification basée sur les besoins du démo 5, fut augmentée en prévision de tests sur de futurs démonstrateurs. Cette spécification permit d'effectuer une recherche dans les cartes existantes pour déterminer si l'une d'entre elle pouvait satisfaire au minimum les exigences imposées. Étant donné qu'aucune carte existante ne possède un nombre suffisant de ports d'entrées/sorties, il fut décidé d'en concevoir une.

La carte est en fait constituée de 3 cartes: une carte mère et deux mezzanines. La conception n'inclurait que le strict nécessaire pour le test du 5e démo, le reste pouvant être ajouté ultérieurement sur de nouvelles mezzanines. La carte mère est sectionnée en quatre parties: le CPU, le FPGA, l'interface avec les mezzanines et l'alimentation.

La section CPU est conçue autour d'un microcontrôleur MCF5272 attaché à une mémoire dynamique de 64 mégaoctets En plus de permettre la communication avec l'extérieur, le microcontrôleur aura comme fonction de programmer et contrôler le FPGA et sera responsable de la gestion de sa température en plus du contrôle de son horloge programmable.

La partie FPGA de la carte se compose d'un XC2V2000 de Xilinx. Il sera branché à 3 banques de mémoires statiques chacune d'une capacité maximale de 9 mégaoctets. Ces mémoires ont été placées de façon à minimiser la distance des signaux vers le FPGA. Son horloge sera contrôlée par un oscillateur programmable capable de générer des fréquences de 0 à 300MHz. Des potentiomètres digitaux permettront de sélectionner les tensions analogiques stables aux démos et des capteurs feront la capture de la température des démonstrateurs.

Le support des démos se fait par les mezzanines. La section mezzanine est tout simplement composée de 6 connecteurs haute vitesse. Ces connecteurs sont placés de façon à permettre un branchement solide de mezzanines de différentes dimensions.

La carte mère est formée de 16 couches. Il n'y a aucun signal de surface pour limiter les émissions électromagnétiques vers les mezzanines. Les signaux auront une largeur de 5 millièmes de pouces et seront espacés de 10 millièmes

La carte a besoin de deux microprogrammes distincts: un pour le FPGA et un pour le microcontrôleur. Le microprogramme du FPGA est séparé en sections autonomes reliées entre elles tel un système sur puce. Une grande partie du microprogramme du FPGA a été écrit pour permettre le test de la carte lorsqu'elle sera assemblée. Le microprogramme du microcontrôleur consiste en une version embarquée du système

d'exploitation Linux: µCLinux. Pour que ce système permette d'utiliser toutes les fonctionnalités d'Erinyes, il faudra procéder à certaines modifications.

Ce projet à été amené au stade de développement le plus avancé possible, considérant les différentes contraintes particulières au contexte. La complexité atteinte par le système de test élaboré est telle que nous estimons que sa mise en œuvre finale et son déboguage dépassent largement le cadre d'un seul projet de maîtrise.

RECOMMANDATIONS

La conception d'Erinyes a été orientée vers la facilité du développement et du déboguage tant matériel que logiciel. Dans cet ordre d'idée, il est possible d'énumérer certaines recommandations.

La planification du routage de la carte n'est pas encore planifiée. Il faut prévoir l.'emplacement des condensateurs de découplages par rapport au placement préliminaire. Le routage des signaux stables peut être effectués en surface pour les éloigner le plus possible des signaux à transitions rapides.

Le code µCLinux devrait tout d'abord être testé qu'avec des modifications mineures. Les modifications obligatoires sont celles de la dimension de la mémoire (flash et dynamique) ainsi que les modifications aux ports de communication sérielle EIA-RS232. Erinyes est grandement inspirée de la carte de développement du Coldfire MC5272 de la compagnie Motorola. µCLinux est déjà conçu pour tourner sur cette plateforme et par conséquent pourra fonctionner sur Erinyes qu'avec ces changements.

Le FPGA ne pourra démarrer sans que sa ligne PWRDWN_B ne soit activée. De ce fait, la première modification apportée à µCLinux devrait être le contrôle de la température du FPGA. Il faudra tester adéquatement que ce dernier fonctionne avant même d'amorcer l'écriture du pilote de configuration du FPGA. Le Virtex est une pièce très dispendieuse et est difficile à remplacer.

L'arbitre de mémoire développé n'est en fait qu'un multiplexeur. Le développement d'un arbitre plus flexible pourrait s'avérer très utile. La carte permet le contrôle de chacune des puces de façon indépendante. Il serait ainsi possible d'élaborer un arbitre permettant de choisir la largeur du bus de données ainsi que la largeur du bus d'adresses

pour chacune des interfaces. Un tel arbitre augmenterait la flexibilité de la carte en rapport au test.

BIBLIOGRAPHIE

[1] Ahmed LAKHSASI, Mohammed BOUGATAYA, Rapport technique sur les gradients de température d'une puce d'intégration à l'échelle de la tranche, POLY/HYP/UQAH-H01-A, Université du Québec à Hull, avril 2001.

[2] Howard JOHNSON, Martin GRAHAM, High-Speed Digital Design, A Handbook of Black Magic, Prentice Hall Inc, New Jersey, 1993

[3] Stephen H. HALL, Garrett W. HALL, James A. MCCALL, High-Speed Digital System Design, A Handbook of Interconnect Theory and Design Practices, John Wiley & Sons Inc, 2000

[4] IEEE Standard Test Access Port and Boundary-Scan Architecture, Institute of Electrical and Electronic Engineers, IEEE Std 1149.1-2001.

[5] Karl FECTEAU, Stéphane NADEAU, Demochip 5 V2 specifications, Hyperchip, 2001

[6] MENG Lu, Bing QIU, Guy HACHÉ, Ara TALASLIAN, Renaud TIENNOT, Normand LECLERC, Conception de structures tolérantes aux défauts d'un système parallèle dans une application WSI, École Polytechnique de Montréal, 2001.

[7] Intel Microprocessor Hall of Fame, Intel Corporation, http://www.intel.com/intel/intelis/museum/exhibit/hist_micro/hof/hof_main.htm

[8] Grand dictionnaire terminologique, Office Québécois de la langue française, http://www.granddictionnaire.com/btml/fra/R_MotClef/index800_1.asp, juillet 2002.

[9] Virtex-II Platform FPGA Handbook v1.2, Xilinx Inc., http://direct.xilinx.com/bvdocs/publications/ds031.pdf, juillet 2002.

[10] MCF5272 ColdFire® Integrated Microprocessor User's Manual, Motorola semiconductors, MCF5272UM.pdf.

[11] SN74ALVCH16245 16-bit bus transceiver with 3-state outputs, Texas Instruments, SCES015H, revised version September 2001

[12] Test Report #99644, REV 1.1, RF CHARACTERIZATION, SAMTEC, 12 janvier 2000, QSHQTHRF.pdf

ANNEXE 1

Classification des défauts de manufacture

Défectuosités de manufacture

Il est possible de classifier les défauts de manufacture en deux catégories : les défectuosités locales et les défectuosités globales.

Les défectuosités globales couvrent une grande surface de la tranche. De tels défauts affectent plusieurs puces et peuvent rendre la tranche complètement inutilisable. Ces défectuosités peuvent être causées par: une mauvaise manipulation de la tranche qui causerait une égratignure sur sa surface, un mauvais alignement de masque, une surexposition ou sous-exposition aux agents oxydants ou dopants, la diffusion des régions ioniques, etc.

Les défauts globaux sont plus fréquents lors de l'application d'un nouveau procédé de fabrication. Après perfectionnement, les défectuosités globales sont plus rares et par conséquent, ne constituent pas une réelle menace pour l'application d'intégration directe sur tranche. Les véritables problèmes de l'intégration directe sur tranche sont les défectuosités locales.

Les défectuosités locales peuvent être représentés comme des structures manquantes ou des circuits ouverts. Elles peuvent également apparaître comme des structures indésirables ou des courts-circuits. Un défaut local n'affecte généralement qu'une seule puce. Ces défectuosités se retrouvent au hasard, distribuées sur la surface de la tranche ne détruisant qu'un nombre limité de cellules.

Même dans les procédés de fabrication matures, de telles défectuosités ne sont pas rares. Elles proviennent principalement de dépôts d'impuretés ou de poussières sur les masques lors de la photolithographie ou directement sur la surface de la tranche à l'une ou l'autre des étapes du processus.

Dans une application d'intégration directe sur tranche, les défauts de manufacture n'affectent pas seulement les cellules; ils affectent également les interconnections. Une seule défectuosité affectant une structure sur la tranche rendrait le système complètement inutilisable. Sans mécanisme de tolérance aux défectuosités, il est impossible d'entrevoir un système tenant sur une tranche.

ANNEXE 2

Banc de test du 4e démonstrateur

```
# WSI demochip3
# Normand Leclerc, Ðcole de technologie sup\211rieure
# Hyperchip May 2001

# Tests makefile

COMPILER=vhdlan -nc

# jtag_a is unused
jtagb: touched/jtagb
cella: touched/cella
cellb: touched/cellb
demochip: touched/demochip
testa: touched/testa
testb: touched/testb
wsi: touched/wsi
tests: testa testb wsi

DATE=`date | awk '{ printf("%s.%s.%d.%d", $$1, $$2, $$3, $$6) }'`

backup: testbenches
        cp Makefile $?
        bkp /home/projet2/normlec/WSI/Synopsys/$?

clean:
        rm -fr touched/* work/*


# Synopsys packages
SYN_PKG_WC=Syn_package/tpz973gwc_components.vhdl \
        Syn_package/vst_n18_sc_tsm_c4_wc_components.vhdl
SYN_PKG_BC=Syn_package/tpz973gbc_components.vhdl \
        Syn_package/vst_n18_sc_tsm_c4_bc_components.vhdl
SYN_PKG_TYP=Syn_package/vst_n18_sc_tsm_c4_typ_components.vhdl \
        Syn_package/tpz973gtc_components.vhdl

# set default
SYN_PKG=$(SYN_PKG_TYP)

# JTAG_B
JTAG_PACKAGES=package/TAP_pack.vhd package/Instruction.vhdl package/jtag.vhdl
JTAGB=src/BR.vhdl src/data_register.vhdl src/BSR.vhdl src/CFGR_new.vhdl \
        src/CFG_cell_new.vhdl src/IR.vhdl src/IR_cell.vhdl src/decode.vhdl \
        src/TAP.vhdl src/TAP_output.vhdl src/control_new.vhdl src/JTAG_B.vhdl
touched/jtagb: $(SYN_PKG) $(JTAG_PACKAGES) $(JTAGB)
        $(COMPILER) -spc $?
        touch $@

# bus and reconfiguration
RECONF=package/bus_arch.vhdl bus_arch_rtl/Bus_Structure_H.vhdl \
        bus_arch_rtl/Bus_Structure_V.vhdl bus_arch_rtl/Con_Structure_H.vhdl \
        bus_arch_rtl/Con_Structure_V.vhdl bus_arch_rtl/StructureHV.vhdl
touched/reconf: $(SYN_PKG) $(RECONF)
        $(COMPILER) -spc $(RECONF)
        touch $@

# CELL_A
CELL_A=src/CELL_A.vhdl
touched/cella: touched/jtagb touched/reconf $(CELL_A)
        $(COMPILER) -spc $(CELL_A)
        touch $@

# CELL_B
CELL_B=src/CELL_B.vhdl
touched/cellb: touched/jtagb touched/reconf $(CELL_B)
        $(COMPILER) -spc $(CELL_B)
        touch $@
```

```
# demochip package
DEMOCHIP_PKG=package/DemoChip_pak.vhdl
touched/demochippkg: touched/cella touched/cellb $(DEMOCHIP_PKG)
	$(COMPILER) -spc $(DEMOCHIP_PKG)
	touch $@

# Demochip
DEMOCHIP=src/DemoChip.vhdl src/DemoChip_chip.vhdl
touched/demochip: touched/demochippkg $(DEMOCHIP)
	$(COMPILER) -spc $(DEMOCHIP)
	touch $@


# TESTS

# global tests definitions
TEST_DEFS=testbenches/PackageBaseDefs_p.vhd testbenches/PackageBaseDefs_b.vhd
touched/defs: $(TEST_DEFS)
	$(COMPILER) $?
	touch $@

# JTAG routines
JTAG_BASIC=testbenches/PackageJTAG_p.vhd testbenches/PackageJTAG_b.vhd
touched/jtagbasic: touched/defs $(JTAG_BASIC)
	$(COMPILER) $(JTAG_BASIC)
	touch $@

# Configuration definitions
CONFIG_DEFS=testbenches/PackageConfig_p.vhd testbenches/PackageConfig_b.vhd
touched/config: touched/defs touched/jtagbasic $(CONFIG_DEFS)
	$(COMPILER) $(CONFIG_DEFS)
	touch $@

# cells definitions
CELLS_DEFS=testbenches/cells/PackageCells_p.vhd \
	testbenches/cells/PackageCells_b.vhd
touched/cellsdefs: touched/jtagbasic $(CELLS_DEFS)
	$(COMPILER) $(CELLS_DEFS)
	touch $@

# cells test procedures
CELL_TESTS=testbenches/cells/PackageCellsProc_p.vhd \
	testbenches/cells/PackageCellsProc_b.vhd
touched/cellsproc: touched/defs touched/jtagbasic \
	touched/config $(CELL_TESTS)
	$(COMPILER) $(CELL_TESTS)
	touch $@

# CELL_A testbench
CELL_A_TB=testbenches/cells/EntityCellA_tb.vhd \
	testbenches/cells/BehavioralCellA_tb.vhd
touched/testa: touched/cella touched/defs touched/cellsdefs \
	touched/cellsproc $(CELL_A_TB)
	$(COMPILER) $(CELL_A_TB)
	touch $@

# CELL_B testbench
CELL_B_TB=testbenches/cells/EntityCellB_tb.vhd \
	testbenches/cells/BehavioralCellB_tb.vhd
touched/testb: touched/cellb touched/defs touched/cellsdefs \
	touched/cellsproc $(CELL_B_TB)
	$(COMPILER) $(CELL_B_TB)
	touch $@


# WSI tests definitions
```

```
WSI_DEFS=testbenches/wsi/PackageWSI_p.vhd testbenches/wsi/PackageWSI_b.vhd
touched/wsidefs: touched/defs $(WSI_DEFS)
	$(COMPILER) $(WSI_DEFS)
	touch $@

# wsi JTAG procedures
WSI_JTAG=testbenches/wsi/PackageWSIJTAG_p.vhd \
	testbenches/wsi/PackageWSIJTAG_b.vhd
touched/wsijtag: touched/defs touched/jtagbasic touched/wsidefs $(WSI_JTAG)
	$(COMPILER) $(WSI_JTAG)
	touch $@

# wsi procedures
WSI_PROC=testbenches/wsi/PackageWSIProc_p.vhd \
	testbenches/wsi/PackageWSIProc_b.vhd
touched/wsiproc: touched/defs touched/jtagbasic touched/config \
	touched/wsijtag $(WSI_PROC)
	$(COMPILER) $(WSI_PROC)
	touch $@

# Bus test procedure
WSI_BUS_TEST=testbenches/wsi/PackageWSIBusTest_p.vhd \
	testbenches/wsi/PackageWSIBusTest_b.vhd
touched/wsibustest: touched/wsiproc $(WSI_BUS_TEST)
	$(COMPILER) $(WSI_BUS_TEST)
	touch $@

# Configuration test procedure
WSI_CFG_TEST=testbenches/wsi/PackageWSICfgTest_p.vhd \
	testbenches/wsi/PackageWSICfgTest_b.vhd
touched/wsicfgtest: touched/wsiproc $(WSI_CFG_TEST)
	$(COMPILER) $(WSI_CFG_TEST)
	touch $@

# WSI testbench
WSI=testbenches/wsi/EntityWSI_tb.vhd testbenches/wsi/BehavorialWSI_tb.vhd
touched/wsi: touched/demochip touched/defs touched/jtagbasic touched/wsidefs\
	touched/wsijtag touched/wsiproc touched/wsibustest \
	touched/wsicfgtest $(WSI)
	$(COMPILER) $(WSI)
	touch $@
```

```
-- WSI demochip3 basic definitions package
-- Normand Leclerc, Đcole de technologie sup\211rieure
-- Hyperchip June 2001

library ieee;
library std;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use std.textio.all; use std.textio;

package baseDefs_pkg is

-- basic simulation variables
  constant cyclePeriod_c: time := 10 ns;


-- usefull new types
  subtype bundle_typ is std_logic_vector(3 downto 0);


-- misc constants
  constant GROUNDED: bundle_typ := (others=>'0');
  constant VSS: bundle_typ := (others=>'0');
  constant VDD: bundle_typ := (others=>'1');

-- Cells value assignment
  constant CELL00: bundle_typ := "0001";
  constant CELL01: bundle_typ := "0010";
  constant CELL02: bundle_typ := "0011";
  constant CELL10: bundle_typ := "0100";
  constant CELL11: bundle_typ := "0101";
  constant CELL12: bundle_typ := "0110";
  constant CELL20: bundle_typ := "1000";
  constant CELL21: bundle_typ := "1001";
  constant CELL22: bundle_typ := "0111";

-- we enlarge the grid array so we get the following configuration possible:
--   (unused)   (unused)   (-1, 1 )   (unused)   (unused)
--   (unused)   ( 0, 0 )   ( 0, 1 )   ( 0, 2 )   (unused)
-- ( 1, -1 )   ( 1, 0 )   ( 1, 1 )   ( 1, 2 )   ( 1, 3 )
--   (unused)   ( 2, 0 )   ( 2, 1 )   ( 2, 2 )   (unused)
--   (unused)   (unused)   ( 3, 1 )   (unused)   (unused)
  constant CELL_11: bundle_typ := "1110";
  constant CELL1_1: bundle_typ := "1101";
  constant CELL13: bundle_typ  := "1100";
  constant CELL31: bundle_typ  := "1011";

-- usefull functions
  procedure fprintf(file out_f: textio.text; constant string_c: string;
      constant vect_c: std_logic_vector);

  procedure fprintf(file out_f: textio.text; constant string_c: string);

end baseDefs_pkg;
```

```
-- WSI demochip3 basic definitions package
-- Normand Leclerc, Đcole de technologie sup\211rieure
-- Hyperchip June 2001


package body baseDefs_pkg is


-- usefull functions
  function vect2String(constant vect_c: std_logic_vector) return string is
    variable lin_v: line;
    variable s_v: string(1 to vect_c'length) := (others=>' ');
  begin
    ieee.std_logic_textio.write(lin_v, vect_c);
    s_v(lin_v.all'range) := lin_v.all;
    deallocate(lin_v);
    return s_v;
  end vect2String;



  procedure fprintf(file out_f: textio.text; constant string_c: string;
      constant vect_c: std_logic_vector) is
  begin
    report string_c & vect2String(vect_c);
    textio.write(out_f, "(" & time'image(now) & ") " & string_c &
      vect2String(vect_c));
  end fprintf;



  procedure fprintf(file out_f: textio.text; constant string_c: string) is
  begin
    report string_c;
    textio.write(out_f, "(" & time'image(now) & ") " & string_c);
  end fprintf;


end baseDefs_pkg;
```

```
-- WSI demochip3 cell configuration package
-- Normand Leclerc, \xcole de technologie sup\211rieure
-- Hyperchip May 2001
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;
use work.JTAG_pkg.all; use work.JTAG_pkg;

package config_pkg is
-- Cell configuration constants
-- I/Os are ordered as such (cf. specification document)
--      1
--      2
--      3
--  6 5 4

-- Bus bundles select
-- Horizontal bus
  constant h6BSel0: config_typ := "00" & "0000" & "000" & "000" & "0000" & "00";
  constant h6BSel1: config_typ := "10" & "0000" & "000" & "000" & "0000" & "00";
  constant h5BSel0: config_typ := "00" & "0000" & "000" & "000" & "0000" & "00";
  constant h5BSel1: config_typ := "01" & "0000" & "000" & "000" & "0000" & "00";

-- Vertical bus
  constant v1BSel0: config_typ := "00" & "0000" & "000" & "000" & "0000" & "00";
  constant v1BSel1: config_typ := "00" & "0000" & "000" & "000" & "0000" & "01";
  constant v2BSel0: config_typ := "00" & "0000" & "000" & "000" & "0000" & "00";
  constant v2BSel1: config_typ := "00" & "0000" & "000" & "000" & "0000" & "10";

-- configuration structure H
-- outputs
  constant vVss: config_typ    := "00" & "0000" & "000" & "000" & "0000" & "00";
  constant vlb0in: config_typ  := "00" & "0000" & "000" & "001" & "0000" & "00";
  constant vb0in: config_typ   := "00" & "0000" & "000" & "010" & "0000" & "00";
  constant vrb0in: config_typ  := "00" & "0000" & "000" & "011" & "0000" & "00";
  constant vub0in: config_typ  := "00" & "0000" & "000" & "100" & "0000" & "00";
  constant vdb0in: config_typ  := "00" & "0000" & "000" & "101" & "0000" & "00";

-- inputs
  constant vb0in1: config_typ  := "00" & "0000" & "000" & "000" & "0000" & "00";
  constant vub1in: config_typ  := "00" & "0000" & "000" & "000" & "0001" & "00";
  constant vb1in: config_typ   := "00" & "0000" & "000" & "000" & "0010" & "00";
  constant vdb1in: config_typ  := "00" & "0000" & "000" & "000" & "0011" & "00";
  constant vub2in: config_typ  := "00" & "0000" & "000" & "000" & "0100" & "00";
  constant vb2in: config_typ   := "00" & "0000" & "000" & "000" & "1000" & "00";
  constant vdb2in: config_typ  := "00" & "0000" & "000" & "000" & "1100" & "00";

-- configuration structure V
-- outputs
  constant hVss: config_typ    := "00" & "0000" & "000" & "000" & "0000" & "00";
  constant hlb0in: config_typ  := "00" & "0000" & "100" & "000" & "0000" & "00";
  constant hb0in: config_typ   := "00" & "0000" & "010" & "000" & "0000" & "00";
  constant hrb0in: config_typ  := "00" & "0000" & "110" & "000" & "0000" & "00";
  constant hub0in: config_typ  := "00" & "0000" & "001" & "000" & "0000" & "00";
  constant hdb0in: config_typ  := "00" & "0000" & "101" & "000" & "0000" & "00";

-- inputs
  constant hb0in1: config_typ  := "00" & "0000" & "000" & "000" & "0000" & "00";
  constant hub1in: config_typ  := "00" & "0100" & "000" & "000" & "0000" & "00";
  constant hb1in: config_typ   := "00" & "1000" & "000" & "000" & "0000" & "00";
  constant hdb1in: config_typ  := "00" & "1100" & "000" & "000" & "0000" & "00";
  constant hub2in: config_typ  := "00" & "0001" & "000" & "000" & "0000" & "00";
  constant hb2in: config_typ   := "00" & "0010" & "000" & "000" & "0000" & "00";
  constant hdb2in: config_typ  := "00" & "0011" & "000" & "000" & "0000" & "00";
end config_pkg;
```

```
-- WSI demochip3 cells configuration package
-- Normand Leclerc, \212cole de technologie sup\211rieure
-- Hyperchip May 2001


package body config_pkg is
end config_pkg;
```

```
-- WSI demochip3 basic JTAG package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;


package JTAG_pkg is


-- JTAG structure definitions
  constant TAPSize_c: integer := 2;
  constant nbConfigRegs_c: integer := 18;
  constant nbDataRegs_c: integer := 24;


-- JTAG types
  subtype inst_typ is std_logic_vector(TAPSize_c-1 downto 0);
  subtype config_typ is std_logic_vector(nbConfigRegs_c-1 downto 0);
  subtype data_typ is std_logic_vector(nbDataRegs_c-1 downto 0);


-- JTAG signals
  signal nTRST, TCK, TMS, TDI, TDO: std_logic;
  signal TCKEn: boolean := false;


-- JTAG output registers
  signal JTAGOutputIns: inst_typ;
  signal JTAGOutputCfg: config_typ;
  signal JTAGOutputCfg2: std_logic_vector(2*nbConfigRegs_c downto 0);
  signal JTAGOutputData: data_typ;


-- JTAG instructions
  constant bypass_c: inst_typ := (others=>'0');
  constant config_c: inst_typ := "10";
  constant samplePreload_c: inst_typ := "01";
  constant extest_c: inst_typ := "11";

-- JTAG instruction register after reset
  constant JTAGRstIR_c: inst_typ := "01";


-- JTAG routines

  procedure JTAGShift(constant data_c: in std_logic_vector;
    signal output: inout std_logic_vector; signal TDI: out std_logic);

  procedure JTAGTCKControl(signal TCK: inout std_logic);

  procedure JTAGHardReset(signal nTRST: out std_logic;
    signal TMS: out std_logic; signal TCKEn: out boolean);
  procedure JTAGSoftReset(signal TMS: out std_logic; signal TCKEn: out boolean);

  procedure JTAGWriteInstruction(constant instruction_c: in std_logic_vector;
    signal output: inout std_logic_vector; signal TCKEn: out boolean;
    signal TMS, TDI: out std_logic);

  procedure JTAGWriteData(constant data_c: in std_logic_vector;
    signal output: inout std_logic_vector; signal TCKEn: out boolean;
    signal TMS, TDI: out std_logic);

end JTAG_pkg;
```

```
-- WSI demochip3 basic JTAG package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001


package body JTAG_pkg is


-- JTAG clock control procedure
  procedure JTAGTCKControl(signal TCK: inout std_logic) is
  begin
    TCK <= '0';
    infLoop: while true loop
      if( TCKEn ) then
        TCK <= not TCK;
      end if;
      wait for baseDefs_pkg.cyclePeriod_c/2;
    end loop infLoop;
  end JTAGTCKControl;



-- JTAG reset procedures
  procedure JTAGHardReset(signal nTRST: out std_logic;
      signal TMS: out std_logic; signal TCKEn: out boolean) is
  begin
    nTRST <= '0';
    TCKEn <= true;
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    nTRST <= '1';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGHardReset;


  procedure JTAGSoftReset(signal TMS: out std_logic;
      signal TCKEn: out boolean) is
  begin
    TCKEn <= true;
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;

    TMS <= '1';
    wait for 3*baseDefs_pkg.cyclePeriod_c;

    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGSoftReset;



-- JTAG data shifting procedure
  procedure JTAGShift(constant data_c: in std_logic_vector;
    signal output: inout std_logic_vector; signal TDI: out std_logic) is
    alias outp: std_logic_vector(output'length-1 downto 0) is output;
  begin
-- Shift data fom LSB to MSB
    shift_in: for n in data_c'reverse_range loop
      wait for baseDefs_pkg.cyclePeriod_c;
      outp <= TDO & outp(outp'left downto 1);
      TDI <= data_c(n);
    end loop shift_in;
  end JTAGShift;
```

```
-- Single cell configuration routine
  procedure JTAGWriteInstruction(constant instruction_c: in std_logic_vector;
      signal output: inout std_logic_vector; signal TCKEn: out boolean;
      signal TMS, TDI: out std_logic) is
  begin
-- select Instruction Register
    TMS <= '1';
    TCKEn <= true;
    wait for 2*baseDefs_pkg.cyclePeriod_c;

--  Capture Instruction register
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;

-- get out of capture
    wait for baseDefs_pkg.cyclePeriod_c;

-- Shift instruction in
    JTAGShift(instruction_c, output, TDI);

-- Exit from Instruction register mode
    TMS <= '1';
    wait for 2*baseDefs_pkg.cyclePeriod_c;

-- return to idle mode
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGWriteInstruction;



-- Single cell data writing routine
  procedure JTAGWriteData(constant data_c: in std_logic_vector;
      signal output: inout std_logic_vector; signal TCKEn: out boolean;
      signal TMS, TDI: out std_logic) is
  begin
-- select Instruction Register
    TMS <= '1';
    TCKEn <= true;
    wait for baseDefs_pkg.cyclePeriod_c;

--  Capture data register
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;

-- get out of capture
    wait for baseDefs_pkg.cyclePeriod_c;

-- Shift data in
    JTAGShift(data_c, output, TDI);

-- Exit from data register mode
    TMS <= '1';
    wait for 2*baseDefs_pkg.cyclePeriod_c;

-- return to idle mode
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGWriteData;

end JTAG_pkg;
```

```
-- WSI demochip3 cell type A testbench
-- Normand Leclerc, Ðcole de technologie sup%rieure
-- Hyperchip May 2001

-- Implementation of middle cell (1,1)

architecture cellA_tb_behav of cellA_tb is

-- package aliases (VHDL'93)
-- alias TCHIP_pkg is work.cells_pkg;
-- alias TJTAG_pkg is work.JTAG_pkg;
-- Synopsys is NOT fully VHDL'93 compliant!

begin


-- CELL A instantiation
  cell: CELL_A
  port map(
-- JTAG
    JTAG_pkg.TDI, JTAG_pkg.nTRST, JTAG_pkg.TCK,
    JTAG_pkg.TMS, JTAG_pkg.TDO,
-- Bus structure
-- Vertical
    cells_pkg.D_1_D1,  cells_pkg.D_2_D2, cells_pkg.D_0_N,
    cells_pkg.U_0_N, cells_pkg.U_1_U2, cells_pkg.U_2_U1,
    cells_pkg.D_1_U2, cells_pkg.D_2_U1, cells_pkg.D_0_A,
    cells_pkg.U_0_A, cells_pkg.U_1_D1, cells_pkg.U_2_D2,
-- Horizontal
    cells_pkg.L_0_N, cells_pkg.L_1_L2, cells_pkg.L_2_L1,
    cells_pkg.R_1_R1, cells_pkg.R_2_R2, cells_pkg.R_0_N,
    cells_pkg.L_0_A, cells_pkg.L_1_R1, cells_pkg.L_2_R2,
    cells_pkg.R_1_L2, cells_pkg.R_2_L1, cells_pkg.R_0_A,
-- Configuration structure
-- Horizontal
    cells_pkg.V_L_B0_IN, cells_pkg.V_U_B0_IN, cells_pkg.V_D_B0_IN,
    cells_pkg.V_R_B0_IN, cells_pkg.V_L_B2_IN, cells_pkg.V_R_B2_IN,
    cells_pkg.V_L_B1_IN, cells_pkg.V_R_B1_IN, cells_pkg.G_V_B0_OUT,
    cells_pkg.G_V_B1_OUT, cells_pkg.G_V_B2_OUT,
-- Vertical
    cells_pkg.H_U_B2_IN, cells_pkg.H_D_B2_IN, cells_pkg.H_U_B1_IN,
    cells_pkg.H_D_B1_IN, cells_pkg.H_D_B0_IN, cells_pkg.H_L_B0_IN,
    cells_pkg.H_U_B0_IN, cells_pkg.H_R_B0_IN, cells_pkg.G_H_B2_OUT,
    cells_pkg.G_H_B1_OUT, cells_pkg.G_H_B0_OUT
  );



-- TCK control process
-- This process is controlled by TCKEn signal
  TCKControl: process
  begin
    JTAG_pkg.JTAGTCKControl(JTAG_pkg.TCK);
  end process TCKControl;




-----------------------
-- main test routine --
-----------------------
  testProc: process
    file file_f: textio.text open write_mode is outFileName_g;
  begin

-- Signals initialization
```

```
-- JTAG
    JTAG_pkg.TCKEn <= false;  -- disable TCK
    JTAG_pkg.nTRST <= '0';  -- maintain reset for a while
    JTAG_pkg.TMS <= '0';
    JTAG_pkg.TDI <= '0';


--------------
-- Data bus --
--------------

-- Make this cell a center cell so that there will be data from all
--  the neigbour cells

-- The simulated cell will have it's value set by JTAG
--  through it's own value or with replacing cells's
--    cells_pkg.L_0_A <= baseDefs_pkg.CELL11;
--    cells_pkg.R_0_A <= baseDefs_pkg.CELL11;
--    cells_pkg.U_0_A <= baseDefs_pkg.CELL11;
--    cells_pkg.D_0_A <= baseDefs_pkg.CELL11;

-- Set output from cell (0,1)
    cells_pkg.U_2_U1 <= baseDefs_pkg.CELL01;
-- (1,0)
    cells_pkg.L_2_L1 <= baseDefs_pkg.CELL10;
-- (1,2)
    cells_pkg.R_1_R1 <= baseDefs_pkg.CELL12;
-- (2,1)
    cells_pkg.D_1_D1 <= baseDefs_pkg.CELL21;

-- Set outputs for extrapolated cells (unused)
-- (-1,1)
    cells_pkg.U_1_U2 <= baseDefs_pkg.CELL_11;
-- (1,-1)
    cells_pkg.L_1_L2 <= baseDefs_pkg.CELL1_1;
-- (1,3)
    cells_pkg.R_2_R2 <= baseDefs_pkg.CELL13;
-- (3,1)
    cells_pkg.D_2_D2 <= baseDefs_pkg.CELL31;


-----------------------
-- Configuration bus --
-----------------------

-- Vertically arranged

-- output mux 3
-- VSS is connected from inside
    cells_pkg.H_U_B0_IN <= baseDefs_pkg.CELL01;
-- H_B0_IN is connected from inside
    cells_pkg.H_D_B0_IN <= baseDefs_pkg.CELL21;
    cells_pkg.H_L_B0_IN <= baseDefs_pkg.CELL10;
    cells_pkg.H_R_B0_IN <= baseDefs_pkg.CELL12;

-- input mux 2
-- H_B0_IN_1 is connected from inside
    cells_pkg.H_U_B1_IN <= baseDefs_pkg.CELL00;
-- H_B1_IN is connected from inside
    cells_pkg.H_D_B1_IN <= baseDefs_pkg.CELL20;

-- input mux 1
-- H_B0_IN_1 is connected from inside
    cells_pkg.H_U_B2_IN <= baseDefs_pkg.CELL02;
-- H_B2_IN is connected from inside
    cells_pkg.H_D_B2_IN <= baseDefs_pkg.CELL22;
```

```
-- Horizontally arranged
-- Vertical busses will be noted as inversed

-- output mux 4
-- VSS is internally connected
    cells_pkg.V_L_B0_IN <= not baseDefs_pkg.CELL10;
-- V_B0_IN is internally connected
    cells_pkg.V_R_B0_IN <= not baseDefs_pkg.CELL12;
    cells_pkg.V_U_B0_IN <= not baseDefs_pkg.CELL01;
    cells_pkg.V_D_B0_IN <= not baseDefs_pkg.CELL21;

-- input mux 5
-- V_B0_IN_1 is internally connected
    cells_pkg.V_L_B1_IN <= not baseDefs_pkg.CELL00;
-- V_B1_IN is internally connected
    cells_pkg.V_R_B1_IN <= not baseDefs_pkg.CELL02;

-- input mux 6
-- V_B0_IN_1 is internally connected
    cells_pkg.V_L_B2_IN <= not baseDefs_pkg.CELL20;
-- V_B2_IN is internally connected
    cells_pkg.V_R_B2_IN <= not baseDefs_pkg.CELL22;



-- begin tests
    baseDefs_pkg.fprintf(file_f, "INFO: Begining tests.");
    baseDefs_pkg.fprintf(file_f, "INFO: All numbers are in binary format.");

    proc_pkg.resetTest(proc_pkg.curTest, JTAG_pkg.TDI,
      JTAG_pkg.nTRST, JTAG_pkg.TMS, JTAG_pkg.TCKEn, JTAG_pkg.JTAGOutputIns,
      file_f);

    proc_pkg.regTest(proc_pkg.curTest, proc_pkg.curSubTest,
      JTAG_pkg.TDI, JTAG_pkg.nTRST, JTAG_pkg.TMS, JTAG_pkg.TCKEn,
      JTAG_pkg.JTAGOutputData, JTAG_pkg.JTAGOutputCfg, JTAG_pkg.JTAGOutputCfg2,
      JTAG_pkg.JTAGOutputIns, file_f);

    proc_pkg.outMuxTest(proc_pkg.curTest, JTAG_pkg.TDI,
      JTAG_pkg.nTRST, JTAG_pkg.TMS, JTAG_pkg.TCKEn, JTAG_pkg.JTAGOutputData,
      JTAG_pkg.JTAGOutputCfg, JTAG_pkg.JTAGOutputIns, file_f);

    proc_pkg.inMuxTest(proc_pkg.curTest, proc_pkg.curSubTest,
      JTAG_pkg.TDI, JTAG_pkg.nTRST, JTAG_pkg.TMS, JTAG_pkg.TCKEn,
      JTAG_pkg.JTAGOutputData, JTAG_pkg.JTAGOutputCfg, JTAG_pkg.JTAGOutputIns,
      file_f);

    baseDefs_pkg.fprintf(file_f, "INFO: Ending of tests.");


-- prevents a second test
infLoop: while( true ) loop
      wait for 100 ns;
    end loop infLoop;
  end process testProc;


end cellA_tb_behav;



configuration cellA_tb_cfg of cellA_tb is
  for cellA_tb_behav
  end for;
end cellA_tb_cfg;
```

```
-- WSI demochip3 cell type B testbench
-- Normand Leclerc, Ðcole de technologie sup&rieure
-- Hyperchip June 2001

-- Implementation of middle cell (1,1)

architecture cellB_tb_behav of cellB_tb is

-- package aliases (VHDL'93)
-- alias TCHIP_pkg is work.cells_pkg;
-- alias TJTAG_pkg is work.JTAG_pkg;
-- Synopsys is NOT fully VHDL'93 compliant!

begin

-- CELL B instantiation
  cell: CELL_B
  port map(
-- JTAG
    JTAG_pkg.TDI, JTAG_pkg.nTRST, JTAG_pkg.TCK,
    JTAG_pkg.TMS, JTAG_pkg.TDO,
-- Direct IOs
    cells_pkg.DO, cells_pkg.DI,
-- Bus structure
-- Vertical
    cells_pkg.D_1_D1,  cells_pkg.D_2_D2, cells_pkg.D_0_N,
    cells_pkg.U_0_N, cells_pkg.U_1_U2, cells_pkg.U_2_U1,
    cells_pkg.D_1_U2, cells_pkg.D_2_U1, cells_pkg.D_0_A,
    cells_pkg.U_0_A, cells_pkg.U_1_D1, cells_pkg.U_2_D2,
-- Horizontal
    cells_pkg.L_0_N, cells_pkg.L_1_L2, cells_pkg.L_2_L1,
    cells_pkg.R_1_R1, cells_pkg.R_2_R2, cells_pkg.R_0_N,
    cells_pkg.L_0_A, cells_pkg.L_1_R1, cells_pkg.L_2_R2,
    cells_pkg.R_1_L2, cells_pkg.R_2_L1, cells_pkg.R_0_A,
-- Configuration structure
-- Horizontal
    cells_pkg.V_L_B0_IN, cells_pkg.V_U_B0_IN, cells_pkg.V_D_B0_IN,
    cells_pkg.V_R_B0_IN, cells_pkg.V_L_B2_IN, cells_pkg.V_R_B2_IN,
    cells_pkg.V_L_B1_IN, cells_pkg.V_R_B1_IN, cells_pkg.G_V_B0_OUT,
    cells_pkg.G_V_B1_OUT, cells_pkg.G_V_B2_OUT,
-- Vertical
    cells_pkg.H_U_B2_IN, cells_pkg.H_D_B2_IN, cells_pkg.H_U_B1_IN,
    cells_pkg.H_D_B1_IN, cells_pkg.H_D_B0_IN, cells_pkg.H_L_B0_IN,
    cells_pkg.H_U_B0_IN, cells_pkg.H_R_B0_IN, cells_pkg.G_H_B2_OUT,
    cells_pkg.G_H_B1_OUT, cells_pkg.G_H_B0_OUT
  );


-- TCK control process
-- This process is controlled by TCKEn signal
  TCKControl: process
  begin
    JTAG_pkg.JTAGTCKControl(JTAG_pkg.TCK);
  end process TCKControl;




-----------------------
-- main test routine --
-----------------------
  testProc: process
    file file_f: textio.text open write_mode is outFileName_g;
  begin

-- Signals initialization
```

```
-- JTAG
    JTAG_pkg.TCKEn <= false;  -- disable TCK
    JTAG_pkg.nTRST <= '0';  -- maintain reset for a while
    JTAG_pkg.TMS <= '0';
    JTAG_pkg.TDI <= '0';


--------------
-- Data bus --
--------------

-- Make this cell a center cell so that there will be data from all
--  the neigbour cells

-- The simulated cell will have it's value set by JTAG
--  through it's own value or with replacing cells's
--    cells_pkg.L_0_A <= baseDefs_pkg.CELL11;
--    cells_pkg.R_0_A <= baseDefs_pkg.CELL11;
--    cells_pkg.U_0_A <= baseDefs_pkg.CELL11;
--    cells_pkg.D_0_A <= baseDefs_pkg.CELL11;

-- Set output from cell (0,1)
    cells_pkg.U_2_U1 <= baseDefs_pkg.CELL01;
-- (1,0)
    cells_pkg.L_2_L1 <= baseDefs_pkg.CELL10;
-- (1,2)
    cells_pkg.R_1_R1 <= baseDefs_pkg.CELL12;
-- (2,1)
    cells_pkg.D_1_D1 <= baseDefs_pkg.CELL21;

-- Set outputs for extrapolated cells (unused)
-- (-1,1)
    cells_pkg.U_1_U2 <= baseDefs_pkg.CELL_11;
-- (1,-1)
    cells_pkg.L_1_L2 <= baseDefs_pkg.CELL1_1;
-- (1,3)
    cells_pkg.R_2_R2 <= baseDefs_pkg.CELL13;
-- (3,1)
    cells_pkg.D_2_D2 <= baseDefs_pkg.CELL31;


----------------------
-- Configuration bus --
----------------------

-- Vertically arranged

-- output mux 3
-- VSS is connected from inside
    cells_pkg.H_U_B0_IN <= baseDefs_pkg.CELL01;
-- H_B0_IN is connected from inside
    cells_pkg.H_D_B0_IN <= baseDefs_pkg.CELL21;
    cells_pkg.H_L_B0_IN <= baseDefs_pkg.CELL10;
    cells_pkg.H_R_B0_IN <= baseDefs_pkg.CELL12;

-- input mux 2
-- H_B0_IN_1 is connected from inside
    cells_pkg.H_U_B1_IN <= baseDefs_pkg.CELL00;
-- H_B1_IN is connected from inside
    cells_pkg.H_D_B1_IN <= baseDefs_pkg.CELL20;

-- input mux 1
-- H_B0_IN_1 is connected from inside
    cells_pkg.H_U_B2_IN <= baseDefs_pkg.CELL02;
-- H_B2_IN is connected from inside
    cells_pkg.H_D_B2_IN <= baseDefs_pkg.CELL22;
```

```
-- Horizontally arranged
-- Vertical busses will be noted as inversed

-- output mux 4
-- VSS is internally connected
    cells_pkg.V_L_B0_IN <= not baseDefs_pkg.CELL10;
-- V_B0_IN is internally connected
    cells_pkg.V_R_B0_IN <= not baseDefs_pkg.CELL12;
    cells_pkg.V_U_B0_IN <= not baseDefs_pkg.CELL01;
    cells_pkg.V_D_B0_IN <= not baseDefs_pkg.CELL21;

-- input mux 5
-- V_B0_IN_1 is internally connected
    cells_pkg.V_L_B1_IN <= not baseDefs_pkg.CELL00;
-- V_B1_IN is internally connected
    cells_pkg.V_R_B1_IN <= not baseDefs_pkg.CELL02;

-- input mux 6
-- V_B0_IN_1 is internally connected
    cells_pkg.V_L_B2_IN <= not baseDefs_pkg.CELL20;
-- V_B2_IN is internally connected
    cells_pkg.V_R_B2_IN <= not baseDefs_pkg.CELL22;

-- begin tests
    baseDefs_pkg.fprintf(file_f, "INFO: Begining tests.");
    baseDefs_pkg.fprintf(file_f, "INFO: All numbers are in binary format.");

    proc_pkg.resetTest(proc_pkg.curTest, JTAG_pkg.TDI,
      JTAG_pkg.nTRST, JTAG_pkg.TMS, JTAG_pkg.TCKEn, JTAG_pkg.JTAGOutputIns,
      file_f);

    proc_pkg.regTest(proc_pkg.curTest, proc_pkg.curSubTest,
      JTAG_pkg.TDI, JTAG_pkg.nTRST, JTAG_pkg.TMS, JTAG_pkg.TCKEn,
      JTAG_pkg.JTAGOutputData, JTAG_pkg.JTAGOutputCfg, JTAG_pkg.JTAGOutputCfg2,
      JTAG_pkg.JTAGOutputIns, file_f);

    proc_pkg.outMuxTest(proc_pkg.curTest, JTAG_pkg.TDI,
      JTAG_pkg.nTRST, JTAG_pkg.TMS, JTAG_pkg.TCKEn, JTAG_pkg.JTAGOutputData,
      JTAG_pkg.JTAGOutputCfg, JTAG_pkg.JTAGOutputIns, file_f);

    proc_pkg.inMuxTest(proc_pkg.curTest, proc_pkg.curSubTest,
      JTAG_pkg.TDI, JTAG_pkg.nTRST, JTAG_pkg.TMS, JTAG_pkg.TCKEn,
      JTAG_pkg.JTAGOutputData, JTAG_pkg.JTAGOutputCfg, JTAG_pkg.JTAGOutputIns,
      file_f);

    proc_pkg.directIOTest(proc_pkg.curTest, proc_pkg.curSubTest,
      JTAG_pkg.TDI, JTAG_pkg.nTRST, JTAG_pkg.TMS, JTAG_pkg.TCKEn,
      JTAG_pkg.JTAGOutputData, JTAG_pkg.JTAGOutputCfg, JTAG_pkg.JTAGOutputIns,
      cells_pkg.DI, cells_pkg.DO, file_f);

    baseDefs_pkg.fprintf(file_f, "INFO: Ending of tests.");

-- prevents a second test
infLoop: while( true ) loop
      wait for 100 ns;
    end loop infLoop;
  end process testProc;

end cellB_tb_behav;

configuration cellB_tb_cfg of cellB_tb is
  for cellB_tb_behav
  end for;
end cellB_tb_cfg;
```

```
-- WSI demochip3 test cells package
-- Normand Leclerc, Đcole de technologie sup\211rieure
-- Hyperchip May 2001

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;
use work.JTAG_pkg.all; use work.JTAG_pkg;


package cells_pkg is

-- CELL A declaration
component Cell_A
  port (
-- Jtag
      TDI, nTRST, TCK, TMS : in std_logic;
      TDO : out std_logic;

--------------------------------------------------------------------------------
-- Structure du bus en V
-- Bus_Structure_V

      D_1_D1  : in  std_logic_vector(3 downto 0);
      D_2_D2  : in  std_logic_vector(3 downto 0);
      D_0_N   : in  std_logic_vector(3 downto 0);
      U_0_N   : in  std_logic_vector(3 downto 0);
      U_1_U2  : in  std_logic_vector(3 downto 0);
      U_2_U1  : in  std_logic_vector(3 downto 0);
      D_1_U2  : out std_logic_vector(3 downto 0);
      D_2_U1  : out std_logic_vector(3 downto 0);
      D_0_A   : out std_logic_vector(3 downto 0);
      U_0_A   : out std_logic_vector(3 downto 0);
      U_1_D1  : out std_logic_vector(3 downto 0);
      U_2_D2  : out std_logic_vector(3 downto 0);

-- Structure du Bus en H
-- Bus_Structure_H

      L_0_N  :  in  std_logic_vector(3 downto 0);
      L_1_L2 :  in  std_logic_vector(3 downto 0);
      L_2_L1 :  in  std_logic_vector(3 downto 0);
      R_1_R1 :  in  std_logic_vector(3 downto 0);
      R_2_R2 :  in  std_logic_vector(3 downto 0);
      R_0_N  :  in  std_logic_vector(3 downto 0);
      L_0_A  :  out std_logic_vector(3 downto 0);
      L_1_R1 :  out std_logic_vector(3 downto 0);
      L_2_R2 :  out std_logic_vector(3 downto 0);
      R_1_L2 :  out std_logic_vector(3 downto 0);
      R_2_L1 :  out std_logic_vector(3 downto 0);
      R_0_A  :  out std_logic_vector(3 downto 0);

-- Con_Structure_H

      V_L_B0_IN   : in std_logic_vector(3 downto 0);
      V_U_B0_IN   : in std_logic_vector(3 downto 0);
      V_D_B0_IN   : in std_logic_vector(3 downto 0);
      V_R_B0_IN   : in std_logic_vector(3 downto 0);
      V_L_B2_IN : in std_logic_vector(3 downto 0);
      V_R_B2_IN : in std_logic_vector(3 downto 0);
      V_L_B1_IN : in std_logic_vector(3 downto 0);
      V_R_B1_IN : in std_logic_vector(3 downto 0);
      G_V_B0_OUT : out std_logic_vector(3 downto 0);
      G_V_B1_OUT : out std_logic_vector(3 downto 0);
      G_V_B2_OUT : out std_logic_vector(3 downto 0);
```

```
-- Con_Structure_V

      H_U_B2_IN : in std_logic_vector(3 downto 0);
      H_D_B2_IN : in std_logic_vector(3 downto 0);
      H_U_B1_IN  : in std_logic_vector(3 downto 0);
      H_D_B1_IN  : in std_logic_vector(3 downto 0);
      H_D_B0_IN  : in std_logic_vector(3 downto 0);
      H_L_B0_IN  : in std_logic_vector(3 downto 0);
      H_U_B0_IN  : in std_logic_vector(3 downto 0);
      H_R_B0_IN  : in std_logic_vector(3 downto 0);
      G_H_B2_OUT : out std_logic_vector(3 downto 0);
      G_H_B1_OUT : out std_logic_vector(3 downto 0);
      G_H_B0_OUT : out std_logic_vector(3 downto 0));
  end component;


-- CELL B declaration
  component Cell_B
    port (
-- Jtag
      TDI, nTRST, TCK, TMS : in std_logic;
      TDO : out std_logic;

-------------------------------------------------------------------------------
-- Direct I/O
      DO : out std_logic_vector( 7 downto 0);
      DI : in  std_logic_vector( 3 downto 0);

-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- Structure du bus en V
-- Bus_Structure_V

      D_1_D1  : in  std_logic_vector(3 downto 0);
      D_2_D2  : in  std_logic_vector(3 downto 0);
      D_0_N   : in  std_logic_vector(3 downto 0);
      U_0_N   : in  std_logic_vector(3 downto 0);
      U_1_U2  : in  std_logic_vector(3 downto 0);
      U_2_U1  : in  std_logic_vector(3 downto 0);
      D_1_U2  : out std_logic_vector(3 downto 0);
      D_2_U1  : out std_logic_vector(3 downto 0);
      D_0_A   : out std_logic_vector(3 downto 0);
      U_0_A   : out std_logic_vector(3 downto 0);
      U_1_D1  : out std_logic_vector(3 downto 0);
      U_2_D2  : out std_logic_vector(3 downto 0);

-- Structure du Bus en H
-- Bus_Structure_H

      L_0_N  :  in  std_logic_vector(3 downto 0);
      L_1_L2 :  in  std_logic_vector(3 downto 0);
      L_2_L1 :  in  std_logic_vector(3 downto 0);
      R_1_R1 :  in  std_logic_vector(3 downto 0);
      R_2_R2 :  in  std_logic_vector(3 downto 0);
      R_0_N  :  in  std_logic_vector(3 downto 0);
      L_0_A  :  out std_logic_vector(3 downto 0);
      L_1_R1 :  out std_logic_vector(3 downto 0);
      L_2_R2 :  out std_logic_vector(3 downto 0);
      R_1_L2 :  out std_logic_vector(3 downto 0);
      R_2_L1 :  out std_logic_vector(3 downto 0);
      R_0_A  :  out std_logic_vector(3 downto 0);

-- Con_Structure_H
```

```
      V_L_B0_IN   : in std_logic_vector(3 downto 0);
      V_U_B0_IN   : in std_logic_vector(3 downto 0);
      V_D_B0_IN   : in std_logic_vector(3 downto 0);
      V_R_B0_IN   : in std_logic_vector(3 downto 0);
      V_L_B2_IN : in std_logic_vector(3 downto 0);
      V_R_B2_IN : in std_logic_vector(3 downto 0);
      V_L_B1_IN : in std_logic_vector(3 downto 0);
      V_R_B1_IN : in std_logic_vector(3 downto 0);
      G_V_B0_OUT : out std_logic_vector(3 downto 0);
      G_V_B1_OUT : out std_logic_vector(3 downto 0);
      G_V_B2_OUT : out std_logic_vector(3 downto 0);

-- Con_Structure_V

      H_U_B2_IN : in std_logic_vector(3 downto 0);
      H_D_B2_IN : in std_logic_vector(3 downto 0);
      H_U_B1_IN  : in std_logic_vector(3 downto 0);
      H_D_B1_IN  : in std_logic_vector(3 downto 0);
      H_D_B0_IN  : in std_logic_vector(3 downto 0);
      H_L_B0_IN  : in std_logic_vector(3 downto 0);
      H_U_B0_IN  : in std_logic_vector(3 downto 0);
      H_R_B0_IN  : in std_logic_vector(3 downto 0);
      G_H_B2_OUT : out std_logic_vector(3 downto 0);
      G_H_B1_OUT : out std_logic_vector(3 downto 0);
      G_H_B0_OUT : out std_logic_vector(3 downto 0));
  end component;



-- Test signals
-- Direct IOs
    signal DI: std_logic_vector(3 downto 0);
    signal DO: std_logic_vector(7 downto 0);

-- Bus structure signals
-- Horizontal bus
    signal L_0_N: std_logic_vector(3 downto 0);
    signal L_1_L2: std_logic_vector(3 downto 0);
    signal L_2_L1: std_logic_vector(3 downto 0);
    signal R_1_R1: std_logic_vector(3 downto 0);
    signal R_2_R2: std_logic_vector(3 downto 0);
    signal R_0_N: std_logic_vector(3 downto 0);
    signal L_0_A: std_logic_vector(3 downto 0);
    signal L_1_R1: std_logic_vector(3 downto 0);
    signal L_2_R2: std_logic_vector(3 downto 0);
    signal R_1_L2: std_logic_vector(3 downto 0);
    signal R_2_L1: std_logic_vector(3 downto 0);
    signal R_0_A: std_logic_vector(3 downto 0);
-- Vertical bus
    signal D_1_D1: std_logic_vector(3 downto 0);
    signal D_2_D2: std_logic_vector(3 downto 0);
    signal D_0_N: std_logic_vector(3 downto 0);
    signal U_0_N: std_logic_vector(3 downto 0);
    signal U_1_U2: std_logic_vector(3 downto 0);
    signal U_2_U1: std_logic_vector(3 downto 0);
    signal D_1_U2: std_logic_vector(3 downto 0);
    signal D_2_U1: std_logic_vector(3 downto 0);
    signal D_0_A: std_logic_vector(3 downto 0);
    signal U_0_A: std_logic_vector(3 downto 0);
    signal U_1_D1: std_logic_vector(3 downto 0);
    signal U_2_D2: std_logic_vector(3 downto 0);

-- Configuration structure
-- Horizontal configuration
    signal V_L_B0_IN: std_logic_vector(3 downto 0);
    signal V_U_B0_IN: std_logic_vector(3 downto 0);
    signal V_D_B0_IN: std_logic_vector(3 downto 0);
```

```
    signal V_R_B0_IN: std_logic_vector(3 downto 0);
    signal V_L_B2_IN: std_logic_vector(3 downto 0);
    signal V_R_B2_IN: std_logic_vector(3 downto 0);
    signal V_L_B1_IN: std_logic_vector(3 downto 0);
    signal V_R_B1_IN: std_logic_vector(3 downto 0);
    signal G_V_B0_OUT: std_logic_vector(3 downto 0);
    signal G_V_B1_OUT: std_logic_vector(3 downto 0);
    signal G_V_B2_OUT: std_logic_vector(3 downto 0);
-- Vertical Configuration
    signal H_U_B2_IN: std_logic_vector(3 downto 0);
    signal H_D_B2_IN: std_logic_vector(3 downto 0);
    signal H_U_B1_IN: std_logic_vector(3 downto 0);
    signal H_D_B1_IN: std_logic_vector(3 downto 0);
    signal H_D_B0_IN: std_logic_vector(3 downto 0);
    signal H_L_B0_IN: std_logic_vector(3 downto 0);
    signal H_U_B0_IN: std_logic_vector(3 downto 0);
    signal H_R_B0_IN: std_logic_vector(3 downto 0);
    signal G_H_B2_OUT: std_logic_vector(3 downto 0);
    signal G_H_B1_OUT: std_logic_vector(3 downto 0);
    signal G_H_B0_OUT: std_logic_vector(3 downto 0);

end cells_pkg;
```

```
-- WSI demochip3 test cells package
-- Normand Leclerc, ?cole de technologie sup\211rieure
-- Hyperchip May 2001


package body cells_pkg is
end cells_pkg;
```

```
-- WSI demochip3 test routines package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;
use work.JTAG_pkg.all; use work.JTAG_pkg;
use work.cells_pkg.all; use work.cells_pkg;
use work.config_pkg.all; use work.config_pkg;

library std;
use std.textio.all; use std.textio;

package proc_pkg is
-- test types
  type test_typ is (Reset, Registers, Output_muxes, Input_muxes, Direct_IOs);
  type subTest_typ is (Bypass, Instruction, Config, Data, Original_bundles,
    Alternate_bundles, DI_TEST, DO_TEST);

-- Simulation visualisation signal
    signal curTest: test_typ;
    signal curSubTest: subTest_typ;

-- Test procedures
  procedure writeCellConfig(constant configData_c: in std_logic_vector;
    signal output, outputInstr: inout std_logic_vector;
    signal TCKEn: out boolean; signal TMS, TDI: out std_logic);

  procedure writeCellData(constant data_c: in std_logic_vector;
    signal output, outputInstr: inout std_logic_vector;
    signal TCKEn: out boolean; signal TMS, TDI: out std_logic);

-- RESET tests
  procedure resetTest(signal cTest: out test_typ;
    signal TDI, nTRST, TMS: out std_logic; signal TCKEn: out boolean;
    signal outputIns: inout std_logic_vector; file debug_f: textio.text);

-- JTAG Registers tests
  procedure regTest(signal cTest: out test_typ; signal csTest: out subTest_typ;
    signal TDI, nTRST, TMS: out std_logic; signal TCKEn: out boolean;
    signal outputData, outputCfg, outputCfg2,
    outputIns: inout std_logic_vector; file debug_f: textio.text);

-- Output mux tests
  procedure outMuxTest(signal cTest: out test_typ;
    signal TDI, nTRST, TMS: out std_logic; signal TCKEn: out boolean;
    signal outputData, outputCfg, outputIns: inout std_logic_vector;
    file debug_f: textio.text);

-- Input mux tests
  procedure inMuxTest(signal cTest: out test_typ;
    signal cSTest: out subtest_typ;
    signal TDI, nTRST, TMS: out std_logic; signal TCKEn: out boolean;
    signal outputData, outputCfg, outputIns: inout std_logic_vector;
    file debug_f: textio.text);

-- Cell B's direct IOs tests
  procedure directIOTest(signal cTest: out test_typ;
    signal csTest: out subTest_typ; signal TDI, nTRST, TMS: out std_logic;
    signal TCKEn: out boolean; signal outputData, outputCfg,
    outputIns: inout std_logic_vector;
    signal DI: inout std_logic_vector(3 downto 0);
    signal DO: in std_logic_vector(7 downto 0); file debug_f: textio.text);

end proc_pkg;
```

```
-- WSI demochip3 test routines package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001


package body proc_pkg is

-- Cell configuration routine
  procedure writeCellConfig(constant configData_c: in std_logic_vector;
      signal output, outputInstr: inout std_logic_vector;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is
  begin
-- Send instruction
    JTAG_pkg.JTAGWriteInstruction(JTAG_pkg.config_c, outputInstr, TCKEn,
      TMS, TDI);

-- Send data
    JTAG_pkg.JTAGWriteData(configData_c, output, TCKEn, TMS, TDI);
  end writeCellConfig;


-- Cell data writing routine
  procedure writeCellData(constant data_c: in std_logic_vector;
      signal output, outputInstr: inout std_logic_vector;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is
  begin
-- Send instruction
    JTAG_pkg.JTAGWriteInstruction(JTAG_pkg.samplePreload_c, outputInstr,
      TCKEn, TMS, TDI);

-- Send data
    JTAG_pkg.JTAGWriteData(data_c, output, TCKEn, TMS, TDI);
  end writeCellData;


-- mux position to string conversion functions
  function outHMuxPos2String(constant pos_c: integer) return string is
  begin
    case pos_c is
      when 0 =>
        return "VSS";
      when 1 =>
        return "V_L_B0_IN";
      when 2 =>
        return "V_B0_IN";
      when 3 =>
        return "V_R_B0_IN";
      when 4 =>
        return "V_U_B0_IN";
      when 5 =>
        return "V_D_B0_IN";
      when others =>
        return "unknown";
    end case;
  end outHMuxPos2String;

  function outVMuxPos2String(constant pos_c: integer) return string is
  begin
    case pos_c is
      when 0 =>
        return "VSS";
      when 1 =>
        return "H_U_B0_IN";
      when 2 =>
        return "H_B0_IN";
      when 3 =>
        return "H_D_B0_IN";
```

```
    when 4 =>
      return "H_L_B0_IN";
    when 5 =>
      return "H_R_B0_IN";
    when others =>
      return "unknown";
  end case;
end outVMuxPos2String;


function inB1VMuxPos2String(constant pos_c: integer) return string is
begin
  case pos_c is
    when 0 =>
      return "H_B0_IN_1";
    when 1 =>
      return "H_U_B1_IN";
    when 2 =>
      return "H_B1_IN";
    when 3 =>
      return "H_D_B1_IN";
    when others =>
      return "unknown";
  end case;
end inB1VMuxPos2String;

function inB2VMuxPos2String(constant pos_c: integer) return string is
begin
  case pos_c is
    when 0 =>
      return "H_B0_IN_1";
    when 1 =>
      return "H_U_B2_IN";
    when 2 =>
      return "H_B2_IN";
    when 3 =>
      return "H_D_B2_IN";
    when others =>
      return "unknown";
  end case;
end inB2VMuxPos2String;


function inB1HMuxPos2String(constant pos_c: integer) return string is
begin
  case pos_c is
    when 0 =>
      return "V_B0_IN_1";
    when 1 =>
      return "V_L_B1_IN";
    when 2 =>
      return "V_B1_IN";
    when 3 =>
      return "V_R_B1_IN";
    when others =>
      return "unknown";
  end case;
end inB1HMuxPos2String;

function inB2HMuxPos2String(constant pos_c: integer) return string is
begin
  case pos_c is
    when 0 =>
      return "V_B0_IN_1";
    when 1 =>
      return "V_U_B2_IN";
    when 2 =>
```

```
        return "V_B2_IN";
      when 3 =>
        return "V_R_B2_IN";
      when others =>
        return "unknown";
    end case;
  end inB2HMuxPos2String;


----------------------
-- TESTS PROCEDURES --
----------------------

--------------------
-- TEST 1          --
-- JTAG reset test --
--------------------
  procedure resetTest(signal cTest: out test_typ;
      signal TDI, nTRST, TMS: out std_logic; signal TCKEn: out boolean;
      signal outputIns: inout std_logic_vector; file debug_f: textio.text) is

  begin

    cTest <= Reset;
    baseDefs_pkg.fprintf(debug_f, "INFO: Beginning reset tests.");

-- hard reset
    JTAG_pkg.JTAGHardReset(nTRST, TMS, TCKEn);

-- test instruction register after reset
    JTAG_pkg.JTAGWriteInstruction(JTAG_pkg.bypass_c, outputIns, TCKEn, TMS,
      TDI);

    if( outputIns /= JTAG_pkg.JTAGRstIR_c ) then
      baseDefs_pkg.fprintf(debug_f,
        "WARNING: Bad instruction output from JTAG after Hard reset: ",
        outputIns);
    else
      baseDefs_pkg.fprintf(debug_f, "INFO: JTAG Hard reset test passed: ",
        outputIns);
    end if;

-- soft reset
    JTAG_pkg.JTAGSoftReset(TMS, TCKEn);

    JTAG_pkg.JTAGWriteInstruction(JTAG_pkg.bypass_c, outputIns, TCKEn, TMS,
      TDI);

    if( outputIns /= JTAG_pkg.JTAGRstIR_c ) then
      baseDefs_pkg.fprintf(debug_f,
        "WARNING: Bad instruction output from JTAG after soft reset: ",
        outputIns);
    else
      baseDefs_pkg.fprintf(debug_f, "INFO: JTAG soft reset test passed: ",
        outputIns);
    end if;

    baseDefs_pkg.fprintf(debug_f, "INFO: End of reset tests.");
  end resetTest;


------------------------
--       TEST 2       --
-- JTAG registers test --
------------------------
  procedure regTest(signal cTest: out test_typ; signal csTest: out subTest_typ;
```

```
      signal TDI, nTRST, TMS: out std_logic; signal TCKEn: out boolean;
      signal outputData, outputCfg, outputCfg2,
      outputIns: inout std_logic_vector; file debug_f: textio.text) is

   variable config_v: std_logic_vector(JTAG_pkg.JTAGOutputCfg'range);
   alias cfgRegTest: std_logic_vector(JTAG_pkg.JTAGOutputCfg'range) is
     outputCfg2(JTAG_pkg.JTAGOutputCfg2'left downto
     JTAG_pkg.JTAGOutputCfg'left+2);

   variable data_v, data1_v:
    std_logic_vector(JTAG_pkg.JTAGOutputData'range);

   constant dataTestPatern_c: std_logic_vector(data_v'range) :=
     "11001001101011110000l010";
   constant configTestPatern_c: std_logic_vector(config_v'range) :=
     "1111101010101001l0";

  begin

    cTest <= Registers;
    baseDefs_pkg.fprintf(debug_f, "INFO: Beginning registers tests.");

-- test bypass mode
    cSTest <= Bypass;
    baseDefs_pkg.fprintf(debug_f, "INFO: Bypass test.");

    JTAG_pkg.JTAGWriteInstruction(JTAG_pkg.bypass_c, outputIns, TCKEn, TMS,
      TDI);

    data_v := dataTestPatern_c;
    JTAG_pkg.JTAGWriteData(data_v, outputData, TCKEn, TMS, TDI);

-- 1 bit will be left out and the first shifted bit SHOULD be '0'
    if( outputData /= data_v(data_v'left-2 downto 0)&"00" ) then
      baseDefs_pkg.fprintf(debug_f,
        "WARNING: Bad bypass output from JTAG: ", outputData);
    else
      baseDefs_pkg.fprintf(debug_f,
        "INFO: Bypass output from JTAG after reset: ", outputData);
    end if;


-- test configuration register
    cSTest <= Config;
    baseDefs_pkg.fprintf(debug_f, "INFO: Configuration register test.");
    JTAG_pkg.JTAGWriteInstruction(JTAG_pkg.config_c, outputIns, TCKEn, TMS,
      TDI);

-- Send data
    config_v := configTestPatern_c;
    JTAG_pkg.JTAGWriteData('0'&config_v&config_v, outputCfg2, TCKEn, TMS, TDI);

    if( cfgRegTest /= config_v )
    then
      baseDefs_pkg.fprintf(debug_f,
        "WARNING: Bad configuration output from JTAG: ", cfgRegTest);
    else
      baseDefs_pkg.fprintf(debug_f, "INFO: JTAG configuration reg test passed:"
        & " ", cfgRegTest);
    end if;


-- test data register
    cSTest <= Data;
-- can't be done, for outputs, must compare inputs when set to a known value
    baseDefs_pkg.fprintf(debug_f, "INFO: Data register test.");
```

```
    config_v := config_pkg.hvss or config_pkg.vVss or
      config_pkg.hb0in1 or config_pkg.vb0in1;
    writeCellConfig(config_v, outputCfg, outputIns, TCKEn, TMS, TDI);
    baseDefs_pkg.fprintf(debug_f, "INFO: Config sent: ", config_v);

-- shift data in
--              INPUTS              OUTPUTS             INPUTS
--            horizontal      horiz     vertical       vertical
    data_v := "0000" & "0000" & CELL11 & not CELL11 & "0000" & "0000";
    writeCellData(data_v, outputData, outputIns, TCKEn, TMS, TDI);
    baseDefs_pkg.fprintf(debug_f, "INFO: Data sent: ", data_v);

    writeCellData(data_v, outputData, outputIns, TCKEn, TMS, TDI);
    data_v := outputData(data_v'left downto data_v'left-7) & "00000000" &
      outputData(data_v'right+7 downto data_v'right);
    data1_v := (others=>'0');

    if( data_v /= data1_v ) then
      baseDefs_pkg.fprintf(debug_f,
        "WARNING: Bad data output from JTAG: ", outputData(15 downto 8));
    else
      baseDefs_pkg.fprintf(debug_f,
        "INFO: Data output from JTAG: ", outputData(15 downto 8));
    end if;

    baseDefs_pkg.fprintf(debug_f, "INFO: End of registers tests.");
  end regTest;



------------------
--    TEST 3     --
-- Output Muxes  --
------------------
  procedure outMuxTest(signal cTest: out test_typ;
    signal TDI, nTRST, TMS: out std_logic; signal TCKEn: out boolean;
    signal outputData, outputCfg, outputIns: inout std_logic_vector;
    file debug_f: textio.text) is

   type outMux_typ is array(0 to 5) of std_logic_vector(3 downto 0);
   variable outHMuxSigs_v: outMux_typ;
   variable outVMuxSigs_v: outMux_typ;

   variable config_v: std_logic_vector(JTAG_pkg.JTAGOutputCfg'range);
   variable data_v: std_logic_vector(JTAG_pkg.JTAGOutputData'range);
   alias outConH: std_logic_vector(2 downto 0) is config_v(8 downto 6);
   alias outConV: std_logic_vector(2 downto 0) is config_v(11 downto 9);

  begin

    cTest <= Output_muxes;
    baseDefs_pkg.fprintf(debug_f, "INFO: Beginning of output muxes test.");
    config_v := (others=>'0');

-- set cell's output to a known value
-- shift data in
--              INPUTS              OUTPUTS             INPUTS
--            horizontal      horiz     vertical       vertical
    data_v := "0000" & "0000" & CELL11 & not CELL11 & "0000" & "0000";
    writeCellData(data_v, outputData, outputIns, TCKEn, TMS, TDI);
    baseDefs_pkg.fprintf(debug_f, "INFO: Data sent: ", data_v);

-- inverters are on the bus, we must therefore inverse results
    outHMuxSigs_v := (not VSS, not cells_pkg.H_U_B0_IN,
      not CELL11, not cells_pkg.H_D_B0_IN, not cells_pkg.H_L_B0_IN,
      not cells_pkg.H_R_B0_IN);
    outVMuxSigs_v := (not VSS, not cells_pkg.V_L_B0_IN,
```

```
      CELL11, not cells_pkg.V_R_B0_IN, not cells_pkg.V_U_B0_IN,
      not cells_pkg.V_D_B0_IN);

-- Change outputs
outTst: for i in 0 to 5 loop
      outConH := conv_std_logic_vector(i,3);
      outConV := conv_std_logic_vector(i,3);
      writeCellConfig(config_v, outputCfg, outputIns, TCKEn, TMS, TDI);
      baseDefs_pkg.fprintf(debug_f, "INFO: Wrote config: ", config_v);
-- we need an extest to send this cell's real data to bus
      JTAG_pkg.JTAGWriteInstruction(JTAG_pkg.extest_c, outputIns, TCKEn, TMS,
        TDI);

      if( cells_pkg.L_0_A/=outHMuxSigs_v(i) or
       cells_pkg.R_0_A/=outHMuxSigs_v(i) ) then
        baseDefs_pkg.fprintf(debug_f,
          "WARNING: Bad data output from cell position " &
          outHMuxPos2String(i));
        baseDefs_pkg.fprintf(debug_f,
          "       expected to get: ", outHMuxSigs_v(i));
        baseDefs_pkg.fprintf(debug_f, "       received L_0_A: ",
          cells_pkg.L_0_A);
        baseDefs_pkg.fprintf(debug_f, "       received R_0_A: ",
          cells_pkg.R_0_A);
      else
        baseDefs_pkg.fprintf(debug_f,
          "INFO: Data output from cell position " &
          outHMuxPos2String(i) & ": ", cells_pkg.L_0_A);
      end if;
      if( cells_pkg.U_0_A/=outVMuxSigs_v(i) or
       cells_pkg.D_0_A/=outVMuxSigs_v(i) ) then
        baseDefs_pkg.fprintf(debug_f,
          "WARNING: Bad data output from cell position " &
          outVMuxPos2String(i));
        baseDefs_pkg.fprintf(debug_f,
          "       expected to get: ", outVMuxSigs_v(i));
        baseDefs_pkg.fprintf(debug_f, "       received U_0_A: ",
          cells_pkg.U_0_A);
        baseDefs_pkg.fprintf(debug_f, "       received D_0_A: ",
          cells_pkg.U_0_A);
      else
        baseDefs_pkg.fprintf(debug_f,
          "INFO: Data output from cell position " &
          outVMuxPos2String(i) & ": ", cells_pkg.U_0_A);
      end if;
    end loop outTst;

    baseDefs_pkg.fprintf(debug_f, "INFO: End of output muxes test.");
  end outMuxTest;



------------------
--   TEST 4     --
-- Input Muxes  --
------------------
  procedure inMuxTest(signal cTest: out test_typ;
    signal cSTest: out subTest_typ;
    signal TDI, nTRST, TMS: out std_logic; signal TCKEn: out boolean;
    signal outputData, outputCfg, outputIns: inout std_logic_vector;
    file debug_f: textio.text) is

   type inMux_typ is array(0 to 3) of std_logic_vector(3 downto 0);
   variable inHMux1Sigs_v: inMux_typ;
   variable inHMux2Sigs_v: inMux_typ;
   variable inVMux1Sigs_v: inMux_typ;
   variable inVMux2Sigs_v: inMux_typ;
```

```
   variable data_v: std_logic_vector(JTAG_pkg.JTAGOutputData'range);
   alias hBundle1: std_logic_vector(3 downto 0) is outputData(23 downto 20);
   alias hBundle2: std_logic_vector(3 downto 0) is outputData(19 downto 16);
   alias vBundle1: std_logic_vector(3 downto 0) is outputData(3 downto 0);
   alias vBundle2: std_logic_vector(3 downto 0) is outputData(7 downto 4);

   variable config_v:
    std_logic_vector(JTAG_pkg.JTAGOutputCfg'range);
   alias b1SelectH: std_logic is config_v(17);
   alias b2SelectH: std_logic is config_v(16);
   alias b1SelectV: std_logic is config_v(0);
   alias b2SelectV: std_logic is config_v(1);
   alias in1ConH: std_logic_vector(1 downto 0) is config_v(3 downto 2);
   alias in2ConH: std_logic_vector(1 downto 0) is config_v(5 downto 4);
   alias in1ConV: std_logic_vector(1 downto 0) is config_v(15 downto 14);
   alias in2ConV: std_logic_vector(1 downto 0) is config_v(13 downto 12);

  begin
    cTest <= Input_muxes;
    baseDefs_pkg.fprintf(debug_f, "INFO: Beginning test of input muxes.");
    config_v := (others=>'0');

-- set cell's output to a known value
-- shift data in
--              INPUTS              OUTPUTS              INPUTS
--           horizontal      horiz     vertical       vertical
    data_v := "0000" & "0000" & CELL11 & not CELL11 & "0000" & "0001";
    writeCellData(data_v, outputData, outputIns, TCKEn, TMS, TDI);
    baseDefs_pkg.fprintf(debug_f, "INFO: Data sent: ", data_v);

-- H_B0_IN_1 and V_B0_IN_1 are known to be VSS for a null configuration.
      inVMux1Sigs_v := (VSS, cells_pkg.H_U_B1_IN,
        cells_pkg.R_1_R1, cells_pkg.H_D_B1_IN);
      inVMux2Sigs_v := (VSS, cells_pkg.H_U_B2_IN,
        cells_pkg.R_2_R2, cells_pkg.H_D_B2_IN);
      inHMux1Sigs_v := (VSS, cells_pkg.V_L_B1_IN,
        cells_pkg.D_1_D1, cells_pkg.V_R_B1_IN);
      inHMux2Sigs_v := (VSS, cells_pkg.V_L_B2_IN,
        cells_pkg.D_2_D2, cells_pkg.V_R_B2_IN);


-- Change inputs
inTstj: for j in 0 to 1 loop
      if( j=0 ) then
        baseDefs_pkg.fprintf(debug_f, "INFO: Original bundles.");
        cSTest <= Original_bundles;
      else
        baseDefs_pkg.fprintf(debug_f, "INFO: Alternate bundles.");
        cSTest <= Alternate_bundles;
      end if;
  inTsti: for i in 0 to 3 loop
        in1ConH := conv_std_logic_vector(i,2);
        in2ConH := conv_std_logic_vector(i,2);
        in1ConV := conv_std_logic_vector(i,2);
        in2ConV := conv_std_logic_vector(i,2);
        writeCellConfig(config_v, outputCfg, outputIns, TCKEn, TMS, TDI);
        baseDefs_pkg.fprintf(debug_f, "INFO: Wrote config: ", config_v);

-- shift out
        writeCellData(data_v, outputData, outputIns, TCKEn, TMS, TDI);

-- bundles 1
        if( vBundle1/=inHMux1Sigs_v(i) ) then
          baseDefs_pkg.fprintf(debug_f,
            "WARNING: Bad data input from JTAG position " &
            inB1HMuxPos2String(i));
```

```
          baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
            inHMux1Sigs_v(i));
          baseDefs_pkg.fprintf(debug_f, "       received H_B1_OUT_SIG: ",
            vBundle1);
        else
          baseDefs_pkg.fprintf(debug_f,
            "INFO: Data input from JTAG position " & inB1HMuxPos2String(i) &
            ": ", hBundle1);
        end if;
        if( hBundle1/=inVMux1Sigs_v(i) ) then
          baseDefs_pkg.fprintf(debug_f,
            "WARNING: Bad data input from JTAG position " &
            inB1VMuxPos2String(i));
          baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
            inVMux1Sigs_v(i));
          baseDefs_pkg.fprintf(debug_f, "       received V_B1_OUT_SIG: ",
            hBundle1);
        else
          baseDefs_pkg.fprintf(debug_f,
            "INFO: Data input from JTAG position " & inB1VMuxPos2String(i) &
            ": ", vBundle1);
        end if;

-- bundles 2
        if( vBundle2/=inHMux2Sigs_v(i) ) then
          baseDefs_pkg.fprintf(debug_f,
            "WARNING: Bad data input from JTAG position " &
            inB2HMuxPos2String(i));
          baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
            inHMux2Sigs_v(i));
          baseDefs_pkg.fprintf(debug_f, "       received H_B2_OUT_SIG: ",
            vBundle2);
        else
          baseDefs_pkg.fprintf(debug_f,
            "INFO: Data input from JTAG position " & inB2HMuxPos2String(i) &
            ": ", hBundle2);
        end if;
        if( hBundle2/=inVMux2Sigs_v(i) ) then
        baseDefs_pkg.fprintf(debug_f,
          "WARNING: Bad data input from JTAG position " &
          inB2VMuxPos2String(i));
        baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
          inVMux2Sigs_v(i));
        baseDefs_pkg.fprintf(debug_f, "       received V_B2_OUT_SIG: ",
          hBundle2);
        else
          baseDefs_pkg.fprintf(debug_f,
            "INFO: Data input from JTAG position " & inB2VMuxPos2String(i) &
            ": ", hBundle2);
        end if;
      end loop inTsti;

-- change test variables
      inVMux1Sigs_v(2) := cells_pkg.L_1_L2;
      inVMux2Sigs_v(2) := cells_pkg.L_2_L1;
      inHMux1Sigs_v(2) := cells_pkg.U_1_U2;
      inHMux2Sigs_v(2) := cells_pkg.U_2_U1;

-- switch bundle input muxes
      b1SelectH := not b1SelectH;
      b2SelectH := not b2SelectH;
      b1SelectV := not b1SelectV;
      b2SelectV := not b2SelectV;
    end loop inTstj;

    baseDefs_pkg.fprintf(debug_f, "INFO: End of input muxes test.");
  end inMuxTest;
```

```
---------------
--   TEST 5   --
-- Direct IOs --
---------------
  procedure directIOTest(signal cTest: out test_typ;
    signal csTest: out subTest_typ; signal TDI, nTRST, TMS: out std_logic;
    signal TCKEn: out boolean; signal outputData, outputCfg,
    outputIns: inout std_logic_vector;
    signal DI: inout std_logic_vector(3 downto 0);
    signal DO: in std_logic_vector(7 downto 0); file debug_f: textio.text) is

   type inMux_typ is array(0 to 3) of std_logic_vector(3 downto 0);
   variable inHMux1Sigs_v: inMux_typ;
   variable inHMux2Sigs_v: inMux_typ;
   variable inVMux1Sigs_v: inMux_typ;
   variable inVMux2Sigs_v: inMux_typ;

   variable data_v: std_logic_vector(JTAG_pkg.JTAGOutputData'range);

-- DO is cut into 4 sections, horizontal bundle 1,2 and vertical bundles 1,2
   alias hDOb1: std_logic_vector(1 downto 0) is DO(7 downto 6);
   alias hDOb2: std_logic_vector(1 downto 0) is DO(5 downto 4);
   alias vDOb2: std_logic_vector(1 downto 0) is DO(3 downto 2);
   alias vDOb1: std_logic_vector(1 downto 0) is DO(1 downto 0);

   variable config_v:
    std_logic_vector(JTAG_pkg.JTAGOutputCfg'range);
   alias in1ConH: std_logic_vector(1 downto 0) is config_v(3 downto 2);
   alias in2ConH: std_logic_vector(1 downto 0) is config_v(5 downto 4);
   alias in1ConV: std_logic_vector(1 downto 0) is config_v(15 downto 14);
   alias in2ConV: std_logic_vector(1 downto 0) is config_v(13 downto 12);

  begin
    cTest <= Direct_IOs;
    baseDefs_pkg.fprintf(debug_f, "INFO: Beginning test direct IOs.");
    config_v := (others=>'0');

-- DO test
   csTest <= DO_TEST;

-- set cell's output to a known value
-- shift data in
--              INPUTS              OUTPUTS              INPUTS
--            horizontal      horiz     vertical       vertical
    data_v := "0000" & "0000" & CELL11 & not CELL11 & "0000" & "0000";
    writeCellData(data_v, outputData, outputIns, TCKEn, TMS, TDI);
    baseDefs_pkg.fprintf(debug_f, "INFO: Data sent: ", data_v);

-- Change inputs
inTst: for i in 0 to 3 loop
      in1ConH := conv_std_logic_vector(i,2);
      in2ConH := conv_std_logic_vector(i,2);
      in1ConV := conv_std_logic_vector(i,2);
      in2ConV := conv_std_logic_vector(i,2);
      writeCellConfig(config_v, outputCfg, outputIns, TCKEn, TMS, TDI);
      baseDefs_pkg.fprintf(debug_f, "INFO: Wrote config: ", config_v);

-- H_B0_IN_1 and V_B0_IN_1 are known to be VSS for a null configuration.
      inVMux1Sigs_v := (VSS, cells_pkg.H_U_B1_IN,
        cells_pkg.R_1_R1, cells_pkg.H_D_B1_IN);
      inVMux2Sigs_v := (VSS, cells_pkg.H_U_B2_IN,
        cells_pkg.R_2_R2, cells_pkg.H_D_B2_IN);
      inHMux1Sigs_v := (VSS, cells_pkg.V_L_B1_IN,
        cells_pkg.D_1_D1, cells_pkg.V_R_B1_IN);
      inHMux2Sigs_v := (VSS, cells_pkg.V_L_B2_IN,
```

```
        cells_pkg.D_2_D2, cells_pkg.V_R_B2_IN);

-- bundles 1
      if( vDOb1/=inHMux1Sigs_v(i)(1 downto 0) ) then
        baseDefs_pkg.fprintf(debug_f,
          "WARNING: Bad data output from IO position " &
          inB1HMuxPos2String(i));
        baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
          inHMux1Sigs_v(i)(1 downto 0) );
        baseDefs_pkg.fprintf(debug_f, "       received H_B1_OUT_SIG: ", vDOb1);
      else
        baseDefs_pkg.fprintf(debug_f,
          "INFO: Data input from IO position " & inB1HMuxPos2String(i) &
          ": ", vDOb1);
      end if;
      if( hDOb1/=inVMux1Sigs_v(i)(3 downto 2) ) then
        baseDefs_pkg.fprintf(debug_f,
          "WARNING: Bad data output from IO position " &
          inB1VMuxPos2String(i));
        baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
          inVMux1Sigs_v(i)(3 downto 2));
        baseDefs_pkg.fprintf(debug_f, "       received V_B1_OUT_SIG: ", hDOb1);
      else
        baseDefs_pkg.fprintf(debug_f,
          "INFO: Data input from IO position " & inB1VMuxPos2String(i) &
          ": ", hDOb1);
      end if;

-- bundles 2
      if( vDOB2/=inHMux2Sigs_v(i)(1 downto 0) ) then
        baseDefs_pkg.fprintf(debug_f,
          "WARNING: Bad data input from IO position " &
          inB2HMuxPos2String(i));
        baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
          inHMux2Sigs_v(i)(1 downto 0));
        baseDefs_pkg.fprintf(debug_f, "       received H_B2_OUT_SIG: ", vDOB2);
      else
        baseDefs_pkg.fprintf(debug_f,
          "INFO: Data input from IO position " & inB2HMuxPos2String(i) &
          ": ", vDOB2);
      end if;
      if( hDOb2/=inVMux2Sigs_v(i)(3 downto 2) ) then
        baseDefs_pkg.fprintf(debug_f,
          "WARNING: Bad data input from IO position " &
          inB2VMuxPos2String(i));
        baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
          inVMux2Sigs_v(i)(3 downto 2));
        baseDefs_pkg.fprintf(debug_f, "       received V_B2_OUT_SIG: ", hDOb2);
      else
        baseDefs_pkg.fprintf(debug_f,
          "INFO: Data input from IO position " & inB2VMuxPos2String(i) &
          ": ", hDOb2);
      end if;
    end loop inTst;


-- DI test
-- DI isn't really transfered to JTAG.  It can be samples and shifted out BUT
--  as soon as samplePreload command ends, the JTAG gets it's value back.
   csTest <= DI_TEST;

-- For this test to work, we must set the cell to let out own output go through
    config_v := config_pkg.hb0in or config_pkg.vb0in or
      config_pkg.hb0in1 or config_pkg.vb0in1;
    writeCellConfig(config_v, outputCfg, outputIns, TCKEn, TMS, TDI);
    baseDefs_pkg.fprintf(debug_f, "INFO: Wrote config: ", config_v);
```

```
-- Change output data
outTst: for i in 0 to 15 loop
      DI <= conv_std_logic_vector(i,4);
      JTAG_pkg.JTAGWriteInstruction(JTAG_pkg.samplePreload_c, outputIns, TCKEn,
        TMS, TDI);
      baseDefs_pkg.fprintf(debug_f, "INFO: Wrote DI: ", DI);

-- we only need to test one bundle since they all went through tests
      if( hDOb1/=DI(3 downto 2) ) then
        baseDefs_pkg.fprintf(debug_f,
          "WARNING: Bad data output from IO.");
        baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
          DI(3 downto 2) );
        baseDefs_pkg.fprintf(debug_f, "       received DO(7:6): ", vDOb1);
      else
        baseDefs_pkg.fprintf(debug_f,
          "INFO: Data input from IO: ", hDOb1);
      end if;
      if( vDOb1/=DI(1 downto 0) ) then
        baseDefs_pkg.fprintf(debug_f,
          "WARNING: Bad data output from IO.");
        baseDefs_pkg.fprintf(debug_f, "       expected to get: ",
          DI(1 downto 0) );
        baseDefs_pkg.fprintf(debug_f, "       received DO(1:0): ", vDOb1);
      else
        baseDefs_pkg.fprintf(debug_f,
          "INFO: Data input from IO: ", vDOb1);
      end if;
    end loop outTst;

    baseDefs_pkg.fprintf(debug_f, "INFO: End of direct IOs test.");
  end directIOTest;

end proc_pkg;
```

```
-- WSI demochip3 test package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;
use work.JTAG_pkg.all; use work.JTAG_pkg;


package WSIDefs_pkg is

  component DemoChip_chip
    port (
-- JTAG interface
      TCK, TMS, nTRST, TDI : in std_logic;
      TDO : out std_logic;
-- I/O interface
      DI_00, DI_02, DI_20, DI_22 : in std_logic_Vector(3 downto 0);
      DO_00, DO_02, DO_20, DO_22 : out std_logic_vector(7 downto 0);
-- TDO on first cell
      TDO_22: out std_logic
    );
  end component;


-- connection signals
  signal DI_00, DI_02, DI_20, DI_22: std_logic_vector(3 downto 0);
  signal DO_00, DO_02, DO_20, DO_22: std_logic_vector(7 downto 0);
  signal TDO_22: std_logic;


-- Demochip3 arangement
  constant nbHorizCells_c: integer := 3;
  constant nbVertCells_c: integer := 3;
  constant nbCells_c: integer := nbHorizCells_c*nbVertCells_c;


-- usefull new types
  type orien_typ is (horizontal, vertical);
  type bundles_typ is (bundle1, bundle2);
  type busSel_typ is (bus_normal, bus_alt1, bus_alt2, bus_alt3);


-- Cartesian to linear position
  function cellLinPos(constant row_c, col_c: integer) return integer;

-- string and number conversions functions
  function bundle2Num(constant bndle_c: busSel_typ) return std_logic_vector;
  function bundle2Str(constant bndle_c: bundles_typ) return string;
  function orient2Str(constant orient_c: orien_typ) return string;
  function busSel2Str(constant bndle_c: busSel_typ) return string;

end WSIDefs_pkg;
```

```
-- WSI demochip3 test package
-- Normand Leclerc, Đcol_ce de technologie sup\211rieure
-- Hyperchip May 2001


package body WSIDefs_pkg is


-- find position of cell in linear array
  function cellLinPos(constant row_c, col_c: integer) return integer is
  begin
    if( conv_std_logic_vector(row_c,8)(0) ='1' ) then
      return nbHorizCells_c*row_c+col_c;
    else
      return (nbCells_c-1)-(nbHorizCells_c*row_c+col_c);
    end if;
  end cellLinPos;



-- bundle selection to std_logiv_vector
  function bundle2Num(constant bndle_c: busSel_typ) return std_logic_vector is
  begin
    case bndle_c is
      when Bus_Normal =>
        return "00";
      when Bus_alt1 =>
        return "01";
      when Bus_alt2 =>
        return "10";
      when Bus_alt3 =>
        return "11";
      when others =>
        return "XX";
    end case;
  end bundle2Num;



-- bundle selection to string
  function bundle2Str(constant bndle_c: bundles_typ) return string is
  begin
    case bndle_c is
      when bundle1 =>
        return "bundle1";
      when bundle2 =>
        return "bundle2";
      when others =>
        return "unknown";
    end case;
  end bundle2Str;


-- orientation to string
  function orient2Str(constant orient_c: orien_typ) return string is
  begin
    case orient_c is
      when horizontal =>
        return "horizontal";
      when vertical =>
        return "vertical";
      when others =>
        return "unknown";
    end case;
  end orient2Str;
```

```
-- bus selection to string
  function busSel2Str(constant bndle_c: busSel_typ) return string is
  begin
    case bndle_c is
      when Bus_Normal =>
        return "Bus Normal";
      when Bus_alt1 =>
        return "Bus alternate 1";
      when Bus_alt2 =>
        return "Bus alternate 2";
      when Bus_alt3 =>
        return "Bus alternate 3";
      when others =>
        return "unknown";
    end case;
  end busSel2Str;


end WSIDefs_pkg;
```

```
-- WSI demochip3
-- Normand Leclerc, Ðcole de technologie sup%rieure
-- Hyperchip May 2001

-- Demochip WSI JTAG package definitions

library ieee;
use ieee.std_logic_1164.all;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;
use work.WSIDefs_pkg.all; use work.WSIDefs_pkg;
use work.JTAG_pkg.all; use work.JTAG_pkg;

package wsiJTAG_pkg is

-- WSI JTAG new types
  subtype wsiIns_typ is std_logic_vector(nbCells_c*
    JTAG_pkg.TAPSize_c-1 downto 0);
  subtype wsiCfg_typ is std_logic_vector(nbCells_c*
    JTAG_pkg.nbConfigRegs_c-1 downto 0);
  subtype wsiData_typ is std_logic_vector(nbCells_c*
    JTAG_pkg.nbDataRegs_c-1 downto 0);


-- WSI output registers
  signal JTAGOutputIns: wsiIns_typ;
  signal JTAGOutputCfg: wsiCfg_typ;
  signal JTAGOutputData: wsiData_typ;



-- JTAG routines

  procedure JTAGWriteInstruction(constant row_c, col_c: in integer;
    constant instruction_c: in std_logic_vector; signal TCKEn: out boolean;
    signal TMS, TDI: out std_logic);

  procedure JTAGWriteInstruction(constant instruction_c: in std_logic_vector;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic);

  procedure JTAGPrepareCapture(constant row_c, col_c: integer;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic);

  procedure JTAGWriteData(constant row_c, col_c: in integer;
    constant data_c: in std_logic_vector; signal TCKEn: out boolean;
    signal TMS, TDI: out std_logic);

  procedure JTAGWriteData(constant row_c, col_c: in integer;
    constant data_c: in std_logic_vector; signal output: inout std_logic_vector;
    signal TCKEn: out boolean; signal TMS, TDI: out std_logic);

  procedure JTAGWriteData(constant data_c: in std_logic_vector;
    signal TCKEn: out boolean; signal TMS, TDI: out std_logic);

end WSIJTAG_pkg;
```

```
-- Normand Leclerc, Ðcol_ce de technologie sup%rieure
-- Hyperchip May 2001

-- Demochip JTAG package definitions

package body WSIJTAG_pkg is


-- JTAG data shifting procedure
  procedure JTAGShift(constant data_c: in std_logic_vector;
    signal TDI: out std_logic) is
  begin
-- Shift data fom LSB to MSB
    shift_in: for n in data_c'reverse_range loop
      wait for baseDefs_pkg.cyclePeriod_c;
      TDI <= data_c(n);
    end loop shift_in;
  end JTAGShift;


-- Single cell configuration routine
  procedure JTAGWriteInstruction(constant row_c, col_c: in integer;
      constant instruction_c: in std_logic_vector; signal TCKEn: out boolean;
      signal TMS, TDI: out std_logic) is
    variable pos_v: integer;
  begin
-- get linear cell pos_vition
    pos_v := WSIDefs_pkg.cellLinPos(row_c, col_c);

-- select Instruction Register
    TMS <= '1';
    TCKEn <= true;
    wait for 2*baseDefs_pkg.cyclePeriod_c;

--  Capture Instruction register
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;

-- get out of capture
    wait for baseDefs_pkg.cyclePeriod_c;

-- put cells after pos_v in bypass mode
    if( pos_v<WSIDefs_pkg.nbCells_c-1 ) then
      bypassHead: for n in nbCells_c-1 downto pos_v+1 loop
        JTAGShift(bypass_c, TDI);
      end loop bypassHead;
    end if;

-- Shift instruction in
    JTAGShift(instruction_c, TDI);

-- pad rest of the cells with bypass
    if( pos_v>0 ) then
      bypassTail: for n in 1 to pos_v loop
        JTAGShift(bypass_c, TDI);
      end loop bypassTail;
    end if;

-- Exit from Instruction register mode
    TMS <= '1';
    wait for 2*baseDefs_pkg.cyclePeriod_c;

-- return to idle mode
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGWriteInstruction;
```

```
-- All cells configuration routine
  procedure JTAGWriteInstruction(constant instruction_c: in std_logic_vector;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is
  begin
-- select Instruction Register
    TMS <= '1';
    TCKEn <= true;
    wait for 2*baseDefs_pkg.cyclePeriod_c;

--  Capture Instruction register
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;

-- get out of capture
    wait for baseDefs_pkg.cyclePeriod_c;

-- Shift instruction in
    cfgLoop: for n in 1 to WSIDefs_pkg.nbCells_c loop
      JTAGShift(instruction_c, TDI);
    end loop cfgLoop;

-- Exit from Instruction register mode
    TMS <= '1';
    wait for 2*baseDefs_pkg.cyclePeriod_c;

-- return to idle mode
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGWriteInstruction;



-- All cells configuration routine for capture
  procedure JTAGPrepareCapture(constant row_c, col_c: integer;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is
    variable pos_v: integer;
  begin
-- get linear cell pos_vition
    pos_v := WSIDefs_pkg.cellLinPos(row_c, col_c);

-- select Instruction Register
    TMS <= '1';
    TCKEn <= true;
    wait for 2*baseDefs_pkg.cyclePeriod_c;

--  Capture Instruction register
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;

-- get out of capture
    wait for baseDefs_pkg.cyclePeriod_c;

-- put cells after pos_v in extest mode
    if( pos_v<WSIDefs_pkg.nbCells_c-1 ) then
      extestHead: for n in WSIDefs_pkg.nbCells_c-1 downto pos_v+1 loop
        JTAGShift(JTAG_pkg.extest_c, TDI);
      end loop extestHead;
    end if;

-- Shift instruction in
    JTAGShift(JTAG_pkg.samplePreload_c, TDI);
```

```
-- pad rest of the cells with extest
    if( pos_v>0 ) then
      extestTail: for n in 1 to pos_v loop
        JTAGShift(JTAG_pkg.extest_c, TDI);
      end loop extestTail;
    end if;

-- Exit from Instruction register mode
    TMS <= '1';
    wait for 2*baseDefs_pkg.cyclePeriod_c;

-- return to idle mode
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGPrepareCapture;


-- Single cell data writing routine
  procedure JTAGWriteData(constant row_c, col_c: in integer;
    constant data_c: in std_logic_vector; signal TCKEn: out boolean;
    signal TMS, TDI: out std_logic) is
    variable pos_v: integer;
  begin
-- get linear cell pos_vition
    pos_v := WSIDefs_pkg.cellLinPos(row_c, col_c);

-- select Instruction Register
    TMS <= '1';
    TCKEn <= true;
    wait for baseDefs_pkg.cyclePeriod_c;

--  Capture data register
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;

-- Shift data in
    JTAGShift(data_c, TDI);

-- place data on targetted cell
-- bypass mode is delaying data of one cycle
    wait for pos_v*baseDefs_pkg.cyclePeriod_c;

-- Exit from data register mode
    TMS <= '1';
    wait for 2*baseDefs_pkg.cyclePeriod_c;

-- return to idle mode
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGWriteData;


--single cell data writing routine with capture
  procedure JTAGWriteData(constant row_c, col_c: in integer;
    constant data_c: in std_logic_vector; signal output: inout std_logic_vector;
    signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is

    constant dtalen_c: integer := JTAG_pkg.data_typ'length;
    variable pos_v, idx_v: integer;
    variable cellData_v: std_logic_vector(data_c'length-1 downto 0) := data_c;

  begin
-- get linear cell position
    pos_v := WSIDefs_pkg.cellLinPos(row_c, col_c);
```

```
-- select Instruction Register
    TMS <= '1';
    TCKEn <= true;
    wait for baseDefs_pkg.cyclePeriod_c;

--  Capture data register
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;

-- get out of capture
    wait for baseDefs_pkg.cyclePeriod_c;

-- get ready to capture data
-- set cells after pos_v with given data
    if( pos_v<WSIDefs_pkg.nbCells_c-1 ) then
      idx_v := (WSIDefs_pkg.nbCells_c - pos_v - 1)*dtaLen_c-1;
      JTAGShift(cellData_v(idx_v downto 0), TDI);

-- remove shifted portion
      cellData_v := (cellData_v'left downto cellData_v'length-idx_v => '0') &
        cellData_v(cellData_v'left downto idx_v+1);
    end if;

-- capture data
    JTAG_pkg.JTAGShift(cellData_v(output'length-1 downto 0), output, TDI);

-- remove shifted portion
    cellData_v := (cellData_v'left downto cellData_v'left-output'length =>'0')
      & cellData_v(cellData_v'left downto output'length);

-- send rest of given data
    if( pos_v>0 ) then
      idx_v := pos_v*dtaLen_c-1;
      JTAGShift(cellData_v(idx_v downto 0), TDI);
    end if;

-- Exit from data register mode
    TMS <= '1';
    wait for 2*baseDefs_pkg.cyclePeriod_c;

-- return to idle mode
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGWriteData;


-- All cells data writing routine
  procedure JTAGWriteData(constant data_c: in std_logic_vector;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is
  begin
-- select Data Register
    TMS <= '1';
    TCKEn <= true;
    wait for baseDefs_pkg.cyclePeriod_c;

--  Capture Data register
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;

-- get out of capture
    wait for baseDefs_pkg.cyclePeriod_c;

-- Shift data in
    if( data_c'length < WSIJTAG_pkg.wsiData_typ'length ) then
      dtaLoop: for n in 1 to WSIDefs_pkg.nbCells_c loop
        JTAGShift(data_c, TDI);
```

```
      end loop dtaLoop;
    else
      JTAGShift(data_c, TDI);
    end if;

-- Exit from data register mode
    TMS <= '1';
    wait for 2*baseDefs_pkg.cyclePeriod_c;

-- return to idle mode
    TMS <= '0';
    wait for baseDefs_pkg.cyclePeriod_c;
    TCKEn <= false;
  end JTAGWriteData;

end WSIJTAG_pkg;
```

```
-- WSI demochip3 procedures test package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;
use work.JTAG_pkg.all; use work.JTAG_pkg;
use work.WSIJTAG_pkg.all; use work.WSIJTAG_pkg;
use work.WSIDefs_pkg.all; use work.WSIDefs_pkg;

library std;
use std.textio.all; use std.textio;


package WSIProc_pkg is

-- test types
  type test_typ is (PreTests, Order, Busses, Config);


-- Simulation visualisation signal
  signal curTest: test_typ;


-- new types
  type orienBundle_typ is array(WSIDefs_pkg.horizontal to WSIDefs_pkg.vertical)
    of bundle_typ;


-- cells initialization constants

-- cartesian config definitions
  type cartDataLong_typ is array(0 to WSIDefs_pkg.nbVertCells_c-1,
    0 to WSIDefs_pkg.nbHorizCells_c-1) of data_typ;
  type cartDataShort_typ is array(0 to WSIDefs_pkg.nbVertCells_c-1,
    0 to WSIDefs_pkg.nbHorizCells_c-1) of orienBundle_typ;

  constant cCDta_c: cartDataShort_typ := (
    (
      ( baseDefs_pkg.CELL00, not baseDefs_pkg.CELL00 ),
      ( baseDefs_pkg.CELL01, not baseDefs_pkg.CELL01 ),
      ( baseDefs_pkg.CELL02, not baseDefs_pkg.CELL02 )
    ), (
      ( baseDefs_pkg.CELL10, not baseDefs_pkg.CELL10 ),
      ( baseDefs_pkg.CELL11, not baseDefs_pkg.CELL11 ),
      ( baseDefs_pkg.CELL12, not baseDefs_pkg.CELL12 )
    ), (
      ( baseDefs_pkg.CELL20, not baseDefs_pkg.CELL20 ),
      ( baseDefs_pkg.CELL21, not baseDefs_pkg.CELL21 ),
      ( baseDefs_pkg.CELL22, not baseDefs_pkg.CELL22 )
    )
  );


-- Chip cells assignment
  constant data00_c: data_typ :=
    "00000000" & cCDta_c(0,0)(horizontal) & cCDta_c(0,0)(vertical) & "00000000";
  constant data01_c: data_typ :=
    "00000000" & cCDta_c(0,1)(horizontal) & cCDta_c(0,1)(vertical) & "00000000";
  constant data02_c: data_typ :=
    "00000000" & cCDta_c(0,2)(horizontal) & cCDta_c(0,2)(vertical) & "00000000";
  constant data10_c: data_typ :=
    "00000000" & cCDta_c(1,0)(horizontal) & cCDta_c(1,0)(vertical) & "00000000";
  constant data11_c: data_typ :=
    "00000000" & cCDta_c(1,1)(horizontal) & cCDta_c(1,1)(vertical) & "00000000";
```

```
  constant data12_c: data_typ :=
    "00000000" & cCDta_c(1,2)(horizontal) & cCDta_c(1,2)(vertical) & "00000000";
  constant data20_c: data_typ :=
    "00000000" & cCDta_c(2,0)(horizontal) & cCDta_c(2,0)(vertical) & "00000000";
  constant data21_c: data_typ :=
    "00000000" & cCDta_c(2,1)(horizontal) & cCDta_c(2,1)(vertical) & "00000000";
  constant data22_c: data_typ :=
    "00000000" & cCDta_c(2,2)(horizontal) & cCDta_c(2,2)(vertical) & "00000000";

-- linear chip config
-- CELL22 is first in JTAG chain
  constant data_c: wsiData_typ := data22_c & data21_c & data20_c &
                                  data10_c & data11_c & data12_c &
                                  data02_c & data01_c & data00_c;

-- cartesian chip config
  constant cData_c: cartDataLong_typ := (
    ( data00_c, data01_c, data02_c ),
    ( data10_c, data11_c, data12_c ),
    ( data20_c, data21_c, data22_c )
  );



-- Cell configuration routine
  procedure writeConfig(constant row_c, col_c: integer;
    constant configData_c: in std_logic_vector;
    signal TCKEn: out boolean; signal TMS, TDI: out std_logic);


-- Chip configuration routine
  procedure writeConfig(constant configData_c: in std_logic_vector;
    signal TCKEn: out boolean; signal TMS, TDI: out std_logic);


-- Cell data writing routine
  procedure writeData(constant row_c, col_c: in integer;
    constant data_c: in std_logic_vector;
    signal TCKEn: out boolean; signal TMS, TDI: out std_logic);


-- Chip data writing routine
  procedure writeData(constant data_c: in std_logic_vector;
    signal TCKEn: out boolean; signal TMS, TDI: out std_logic);


-- Chip initialization procedure
  procedure init(signal cTest: out test_typ; signal TCKEn: out boolean;
    signal TMS, TDI, nTRST: out std_logic; file debug_f: textio.text);

end WSIProc_pkg;
```

```
-- WSI demochip3 procedures test package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001


package body WSIProc_pkg is


-- Cell configuration routine
  procedure writeConfig(constant row_c, col_c: integer;
      constant configData_c: in std_logic_vector;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is
  begin
-- Send instruction
    WSIJTAG_pkg.JTAGWriteInstruction(row_c, col_c, JTAG_pkg.config_c,
      TCKEn, TMS, TDI);

-- Send data
    WSIJTAG_pkg.JTAGWriteData(row_c, col_c, configData_c, TCKEn, TMS, TDI);
  end writeConfig;



-- Chip configuration routine
  procedure writeConfig(constant configData_c: in std_logic_vector;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is
  begin
-- Send instruction
    WSIJTAG_pkg.JTAGWriteInstruction(JTAG_pkg.config_c, TCKEn, TMS, TDI);

-- Send data
    WSIJTAG_pkg.JTAGWriteData(configData_c, TCKEn, TMS, TDI);
  end writeConfig;



-- Cell data writing routine
  procedure writeData(constant row_c, col_c: in integer;
      constant data_c: in std_logic_vector;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is
  begin
-- Send instruction
    WSIJTAG_pkg.JTAGWriteInstruction(row_c, col_c, JTAG_pkg.samplePreload_c,
      TCKEn, TMS, TDI);

-- Send data
    WSIJTAG_pkg.JTAGWriteData(row_c, col_c, data_c, TCKEn, TMS, TDI);
  end writeData;


-- Chip data writing routine
  procedure writeData(constant data_c: in std_logic_vector;
      signal TCKEn: out boolean; signal TMS, TDI: out std_logic) is
  begin
-- Send instruction
    WSIJTAG_pkg.JTAGWriteInstruction(JTAG_pkg.samplePreload_c, TCKEn, TMS, TDI);

-- Send data
    WSIJTAG_pkg.JTAGWriteData(data_c, TCKEn, TMS, TDI);
  end writeData;



-- Chip initialization procedure
  procedure init(signal cTest: out test_typ; signal TCKEn: out boolean;
    signal TMS, TDI, nTRST: out std_logic; file debug_f: textio.text) is
  begin
```

```
   cTest <= PreTests;

-- reset chip
    baseDefs_pkg.fprintf(debug_f, "INFO: resetting chip (HARD).");
    JTAG_pkg.JTAGHardReset(nTRST, TMS, TCKEn);
    baseDefs_pkg.fprintf(debug_f, "INFO: done.");

-- initialize configuration JTAG chain to a known value
    baseDefs_pkg.fprintf(debug_f, "INFO: Setting config up.");
    WSIJTAG_pkg.JTAGWriteInstruction(config_c, TCKEn, TMS, TDI);
-- shifting once will zero all chain
    WSIJTAG_pkg.JTAGWriteData(2, 2, "0", TCKEn, TMS, TDI);
    baseDefs_pkg.fprintf(debug_f, "INFO: done.");

-- send cells data in
    baseDefs_pkg.fprintf(debug_f, "INFO: Setting cells data up.");
-- Send instruction
    WSIJTAG_pkg.JTAGWriteInstruction(JTAG_pkg.samplePreload_c, TCKEn, TMS, TDI);
-- Send data
    WSIJTAG_pkg.JTAGWriteData(data_c, TCKEn, TMS, TDI);
    baseDefs_pkg.fprintf(debug_f, "INFO: done.");

  end init;

end WSIProc_pkg;
```

```
-- WSI demochip3 configuration test procedures package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;
use work.JTAG_pkg.all; use work.JTAG_pkg;
use work.WSIJTAG_pkg.all; use work.WSIJTAG_pkg;
use work.WSIDefs_pkg.all; use work.WSIDefs_pkg;
use work.WSIProc_pkg.all; use work.WSIProc_pkg;


package cfgTest_pkg is

  procedure cfgTest(curTest, JTAG_pkg.JTAGOutputData, TCKEn, TMS, TDI, nTRST,
    file_f);

end cfgTest_pkg;
```

```
-- WSI demochip3 configuration test procedures package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001

package body cfgTest_pkg is

  procedure cfgTest(curTest, JTAG_pkg.JTAGOutputData, TCKEn, TMS, TDI, nTRST,
    file_f) is

    variable cfg_v: JTAG_pkg.config_typ := h6BSel0 or h5BSel1 or v1BSel0 or
      v2BSel1 or vVss or vb0in1 or hVss or hb0in1;
    alias outConfH: std_logic_vector(2 downto 0) is cfg_v(8 downto 6);
    alias outConfV: std_logic_vector(2 downto 0) is cfg_v(11 downto 9);
    alias inConfH: std_logic_vector(3 downto 0) is cfg_v(5 downto 2);
    alias inConfV: std_logic_vector(3 downto 0) is cfg_v(15 downto 12);

  begin
    cTest <= Config;
    baseDefs_pkg.fprintf(debug_f, "INFO: Starting bus test.");

-- send the same configuration to all cells
      baseDefs_pkg.fprintf(debug_f, "INFO: Configuring cells for " &
        busSel2Str(busSel) & ".");
      WSIProc_pkg.writeConfig(cfg_v, TCKEn, TMS, TDI);
      baseDefs_pkg.fprintf(debug_f, "INFO: Done.");



    baseDefs_pkg.fprintf(debug_f, "INFO: End of bus test.");
  end cfgTest;

end cfgTest_pkg;
```

```
-- WSI demochip3 bus test procedures package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001

library ieee;
library std;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use std.textio.all; use std.textio;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;
use work.JTAG_pkg.all; use work.JTAG_pkg;
use work.config_pkg.all; use work.config_pkg;
use work.WSIJTAG_pkg.all; use work.WSIJTAG_pkg;
use work.WSIDefs_pkg.all; use work.WSIDefs_pkg;
use work.WSIProc_pkg.all; use work.WSIProc_pkg;


package busTest_pkg is

-- this structure will enable us to create a test loop
  type bunArray_typ is array(WSIDefs_pkg.bundle1 to WSIDefs_pkg.bundle2) of
    WSIProc_pkg.orienBundle_typ;
  type busCellArray_typ is array(WSIDefs_pkg.bus_normal to
    WSIDefs_pkg.bus_alt3) of bunArray_typ;
  type cellArray_typ is array(0 to WSIDefs_pkg.nbVertCells_c-1,
    0 to WSIDefs_pkg.nbHorizCells_c-1) of busCellArray_typ;


-- construct the data comparison array
  constant chipDta_c: cellArray_typ := (
   (
-- cell (0,0)
     (
-- bus_normal
      (
-- bundle 1
       ( not WSIProc_pkg.cCDta_c(0,1)(horizontal),
         not WSIProc_pkg.cCDta_c(1,0)(vertical)
       ),
-- bundle 2
       ( WSIProc_pkg.cCDta_c(0,2)(horizontal),
         WSIProc_pkg.cCDta_c(2,0)(vertical)
       )
      ),
-- bus alt1
      (
-- bundle 1
       ( not WSIProc_pkg.cCDta_c(0,1)(horizontal),
         baseDefs_pkg.VSS
       ),
-- bundle 2
       ( baseDefs_pkg.VSS,
         WSIProc_pkg.cCDta_c(2,0)(vertical)
       )
      ),
-- bus alt2
      (
-- bundle 1
       ( baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(1,0)(vertical)
       ),
-- bundle 2
       ( WSIProc_pkg.cCDta_c(0,2)(horizontal),
         baseDefs_pkg.VSS
       )
      ),
-- bus alt3
```

```
      (
-- bundle 1
       ( baseDefs_pkg.VSS,
         baseDefs_pkg.VSS
       ),
-- bundle 2
       ( baseDefs_pkg.VSS,
         baseDefs_pkg.VSS
       )
      )
     ),

-- cell (0,1)
     (
-- bus_normal
      (
-- bundle 1
       ( not WSIProc_pkg.cCDta_c(0,2)(horizontal),
         not WSIProc_pkg.cCDta_c(1,1)(vertical)
       ),
-- bundle 2
       ( not baseDefs_pkg.VSS,
         WSIProc_pkg.cCDta_c(2,1)(vertical)
       )
      ),
-- bus alt1
      (
-- bundle 1
       ( not WSIProc_pkg.cCDta_c(0,2)(horizontal),
         baseDefs_pkg.VSS
       ),
-- bundle 2
       ( not WSIProc_pkg.cCDta_c(0,0)(horizontal),
         WSIProc_pkg.cCDta_c(2,1)(vertical)
       )
      ),
-- bus alt2
      (
-- bundle 1
       ( not baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(1,1)(vertical)
       ),
-- bundle 2
       ( not baseDefs_pkg.VSS,
         baseDefs_pkg.VSS
       )
      ),
-- bus alt3
      (
-- bundle 1
       ( not baseDefs_pkg.VSS,
         baseDefs_pkg.VSS
       ),
-- bundle 2
       ( not WSIProc_pkg.cCDta_c(0,0)(horizontal),
         baseDefs_pkg.VSS
       )
      )
     ),

-- cell (0,2)
     (
-- bus_normal
      (
-- bundle 1
       ( baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(1,2)(vertical)
```

```
      ),
-- bundle 2
      ( baseDefs_pkg.VSS,
        WSIProc_pkg.cCDta_c(2,2)(vertical)
      )
     ),
-- bus alt1
     (
-- bundle 1
      ( baseDefs_pkg.VSS,
        baseDefs_pkg.VSS
      ),
-- bundle 2
      ( not WSIProc_pkg.cCDta_c(0,1)(horizontal),
        WSIProc_pkg.cCDta_c(2,2)(vertical)
      )
     ),
-- bus alt2
     (
-- bundle 1
      ( WSIProc_pkg.cCDta_c(0,0)(horizontal),
        not WSIProc_pkg.cCDta_c(1,2)(vertical)
      ),
-- bundle 2
      ( baseDefs_pkg.VSS,
        baseDefs_pkg.VSS
      )
     ),
-- bus alt3
     (
-- bundle 1
      ( WSIProc_pkg.cCDta_c(0,0)(horizontal),
        baseDefs_pkg.VSS
      ),
-- bundle 2
      ( not WSIProc_pkg.cCDta_c(0,1)(horizontal),
        baseDefs_pkg.VSS
      )
     )
    )


  ), (

-- cell (1,0)
    (
-- bus_normal
     (
-- bundle 1
      ( not WSIProc_pkg.cCDta_c(1,1)(horizontal),
        not WSIProc_pkg.cCDta_c(2,0)(vertical)
      ),
-- bundle 2
      ( WSIProc_pkg.cCDta_c(1,2)(horizontal),
        not baseDefs_pkg.VSS
      )
     ),
-- bus alt1
     (
-- bundle 1
      ( not WSIProc_pkg.cCDta_c(1,1)(horizontal),
        not baseDefs_pkg.VSS
      ),
-- bundle 2
      ( baseDefs_pkg.VSS,
        not baseDefs_pkg.VSS
      )
```

```
      ),
-- bus alt2
      (
-- bundle 1
       ( baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(2,0)(vertical)
       ),
-- bundle 2
       ( WSIProc_pkg.cCDta_c(1,2)(horizontal),
         not WSIProc_pkg.cCDta_c(0,0)(vertical)
       )
      ),
-- bus alt3
      (
-- bundle 1
       ( baseDefs_pkg.VSS,
         not baseDefs_pkg.VSS
       ),
-- bundle 2
       ( baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(0,0)(vertical)
       )
      )
     ),

-- cell (1,1)
     (
-- bus_normal
      (
-- bundle 1
       ( not WSIProc_pkg.cCDta_c(1,2)(horizontal),
         not WSIProc_pkg.cCDta_c(2,1)(vertical)
       ),
-- bundle 2
       ( not baseDefs_pkg.VSS,
         not baseDefs_pkg.VSS
       )
      ),
-- bus alt1
      (
-- bundle 1
       ( not WSIProc_pkg.cCDta_c(1,2)(horizontal),
         not baseDefs_pkg.VSS
       ),
-- bundle 2
       ( not WSIProc_pkg.cCDta_c(1,0)(horizontal),
         not baseDefs_pkg.VSS
       )
      ),
-- bus alt2
      (
-- bundle 1
       ( not baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(2,1)(vertical)
       ),
-- bundle 2
       ( not baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(0,1)(vertical)
       )
      ),
-- bus alt3
      (
-- bundle 1
       ( not baseDefs_pkg.VSS,
         not baseDefs_pkg.VSS
       ),
-- bundle 2
```

```
       ( not WSIProc_pkg.cCDta_c(1,0)(horizontal),
         not WSIProc_pkg.cCDta_c(0,1)(vertical)
       )
      )
     ),

-- cell (1,2)
     (
-- bus_normal
      (
-- bundle 1
       ( baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(2,2)(vertical)
       ),
-- bundle 2
       ( baseDefs_pkg.VSS,
         not baseDefs_pkg.VSS
       )
      ),
-- bus alt1
      (
-- bundle 1
       ( baseDefs_pkg.VSS,
         not baseDefs_pkg.VSS
       ),
-- bundle 2
       ( not WSIProc_pkg.cCDta_c(1,1)(horizontal),
         not baseDefs_pkg.VSS
       )
      ),
-- bus alt2
      (
-- bundle 1
       ( WSIProc_pkg.cCDta_c(1,0)(horizontal),
         not WSIProc_pkg.cCDta_c(2,2)(vertical)
       ),
-- bundle 2
       ( baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(0,2)(vertical)
       )
      ),
-- bus alt3
      (
-- bundle 1
       ( WSIProc_pkg.cCDta_c(1,0)(horizontal),
         not baseDefs_pkg.VSS
       ),
-- bundle 2
       ( not WSIProc_pkg.cCDta_c(1,1)(horizontal),
         not WSIProc_pkg.cCDta_c(0,2)(vertical)
       )
      )
     )

   ), (

-- cell (2,0)
     (
-- bus_normal
      (
-- bundle 1
       ( not WSIProc_pkg.cCDta_c(2,1)(horizontal),
         baseDefs_pkg.VSS
       ),
-- bundle 2
       ( WSIProc_pkg.cCDta_c(2,2)(horizontal),
         baseDefs_pkg.VSS
```

```
      )
     ),
-- bus alt1
     (
-- bundle 1
      ( not WSIProc_pkg.cCDta_c(2,1)(horizontal),
        WSIProc_pkg.cCDta_c(0,0)(vertical)
      ),
-- bundle 2
      ( baseDefs_pkg.VSS,
        baseDefs_pkg.VSS
      )
     ),
-- bus alt2
     (
-- bundle 1
      ( baseDefs_pkg.VSS,
        baseDefs_pkg.VSS
      ),
-- bundle 2
      ( WSIProc_pkg.cCDta_c(2,2)(horizontal),
        not WSIProc_pkg.cCDta_c(1,0)(vertical)
      )
     ),
-- bus alt3
     (
-- bundle 1
      ( baseDefs_pkg.VSS,
        WSIProc_pkg.cCDta_c(0,0)(vertical)
      ),
-- bundle 2
      ( baseDefs_pkg.VSS,
        not WSIProc_pkg.cCDta_c(1,0)(vertical)
      )
     )
    ),

-- cell (2,1)
    (
-- bus_normal
     (
-- bundle 1
      ( not WSIProc_pkg.cCDta_c(2,2)(horizontal),
        baseDefs_pkg.VSS
      ),
-- bundle 2
      ( not baseDefs_pkg.VSS,
        baseDefs_pkg.VSS
      )
     ),
-- bus alt1
     (
-- bundle 1
      ( not WSIProc_pkg.cCDta_c(2,2)(horizontal),
        WSIProc_pkg.cCDta_c(0,1)(vertical)
      ),
-- bundle 2
      ( not WSIProc_pkg.cCDta_c(2,0)(horizontal),
        baseDefs_pkg.VSS
      )
     ),
-- bus alt2
     (
-- bundle 1
      ( not baseDefs_pkg.VSS,
        baseDefs_pkg.VSS
      ),
```

```
-- bundle 2
       ( not baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(1,1)(vertical)
       )
      ),
-- bus alt3
      (
-- bundle 1
       ( not baseDefs_pkg.VSS,
         WSIProc_pkg.cCDta_c(0,1)(vertical)
       ),
-- bundle 2
       ( not WSIProc_pkg.cCDta_c(2,0)(horizontal),
         not WSIProc_pkg.cCDta_c(1,1)(vertical)
       )
      )
     ),

-- cell (2,2)
    (
-- bus_normal
      (
-- bundle 1
       ( baseDefs_pkg.VSS,
         baseDefs_pkg.VSS
       ),
-- bundle 2
       ( baseDefs_pkg.VSS,
         baseDefs_pkg.VSS
       )
      ),
-- bus alt1
      (
-- bundle 1
       ( baseDefs_pkg.VSS,
         WSIProc_pkg.cCDta_c(0,2)(vertical)
       ),
-- bundle 2
       ( not WSIProc_pkg.cCDta_c(2,1)(horizontal),
         baseDefs_pkg.VSS
       )
      ),
-- bus alt2
      (
-- bundle 1
       ( WSIProc_pkg.cCDta_c(2,0)(horizontal),
         baseDefs_pkg.VSS
       ),
-- bundle 2
       ( baseDefs_pkg.VSS,
         not WSIProc_pkg.cCDta_c(1,2)(vertical)
       )
      ),
-- bus alt3
      (
-- bundle 1
       ( WSIProc_pkg.cCDta_c(2,0)(horizontal),
         WSIProc_pkg.cCDta_c(0,2)(vertical)
       ),
-- bundle 2
       ( not WSIProc_pkg.cCDta_c(2,1)(horizontal),
         not WSIProc_pkg.cCDta_c(1,2)(vertical)
       )
      )
     )
    )
  );
```

```
-- bus test procedure
  procedure busTest(signal cTest: out test_typ;
    signal output: inout JTAG_pkg.data_typ; signal TCKEn: out boolean;
    signal TMS, TDI, nTRST: out std_logic; file debug_f: textio.text);

end busTest_pkg;
```

```
-- WSI demochip3 procedures test package
-- Normand Leclerc, Ðcole de technologie sup\211rieure
-- Hyperchip May 2001

package body busTest_pkg is

  procedure busTest(signal cTest: out test_typ;
    signal output: inout JTAG_pkg.data_typ; signal TCKEn: out boolean;
    signal TMS, TDI, nTRST: out std_logic; file debug_f: textio.text) is

    variable cfg_v: JTAG_pkg.config_typ := h6BSel0 or h5BSel0 or v1BSel0 or
      v2BSel0 or vb0in or vb1in or vb2in or hb0in or hb1in or hb2in;
    alias hbSel: std_logic_vector(1 downto 0) is cfg_v(17 downto 16);
    alias vbSel: std_logic_vector(1 downto 0) is cfg_v(1 downto 0);

    alias hb1: baseDefs_pkg.bundle_typ is
      output(output'left downto output'left-3);
    alias hb2: baseDefs_pkg.bundle_typ is
      output(output'left-4 downto output'left-7);
    alias vb1: baseDefs_pkg.bundle_typ is
      output(output'right+3 downto output'right);
    alias vb2: baseDefs_pkg.bundle_typ is
      output(output'right+7 downto output'right+4);

    type bunData_typ is array(WSIDefs_pkg.bundle1 to WSIDefs_pkg.bundle2) of
      baseDefs_pkg.bundle_typ;
    type orienBunData_typ is array(WSIDefs_pkg.horizontal to
      WSIDefs_pkg.vertical) of bunData_typ;
    variable bundleData_v: orienBunData_typ;

  begin

    cTest <= Busses;
    baseDefs_pkg.fprintf(debug_f, "INFO: Starting bus test.");

    busSelect: for busSel in WSIDefs_pkg.Bus_Normal to WSIDefs_pkg.Bus_alt3 loop

-- update configuration
      hbSel := WSIDefs_pkg.bundle2Num(busSel);
      vbSel := WSIDefs_pkg.bundle2Num(busSel);

-- separator
      baseDefs_pkg.fprintf(debug_f, "------------");

-- send the same configuration to all cells
      baseDefs_pkg.fprintf(debug_f, "INFO: Configuring cells for " &
        busSel2Str(busSel) & ".");
      WSIProc_pkg.writeConfig(cfg_v, TCKEn, TMS, TDI);
      baseDefs_pkg.fprintf(debug_f, "INFO: Done.");

-- scan cells
      rows: for i in 0 to WSIDefs_pkg.nbVertCells_c-1 loop
        columns: for j in 0 to WSIDefs_pkg.nbHorizCells_c-1 loop

          baseDefs_pkg.fprintf(debug_f, "INFO: Testing cell (" &
            integer'image(i) & "," & integer'image(j) & ").");

-- prepare cell for capture and do it
          WSIJTAG_pkg.JTAGPrepareCapture(i, j, TCKEn, TMS, TDI);
          WSIJTAG_pkg.JTAGWriteData(i, j, WSIProc_pkg.data_c, output, TCKEn,
            TMS, TDI);

-- prepare vectors
          bundleData_v := ( (hb1, hb2), (vb1, vb2) );

-- compare received data
          orient: for o in WSIDefs_pkg.horizontal to WSIDefs_pkg.vertical loop
```

```
            bndle: for b in WSIDefs_pkg.bundle1 to WSIDefs_pkg.bundle2 loop
              if( bundleData_v(o)(b) /= chipDta_c(i, j)(busSel)(b)(o) ) then
                baseDefs_pkg.fprintf(debug_f, "WARNING: Bad " & orient2Str(o) &
                  " data output for " & WSIDefs_pkg.bundle2Str(b) & ": ",
                  bundleData_v(o)(b));
                baseDefs_pkg.fprintf(debug_f, "         expected: ",
                  chipDta_c(i, j)(busSel)(b)(o));
              else
                baseDefs_pkg.fprintf(debug_f, "INFO: " & orient2Str(o) &
                  " data output for " & WSIDefs_pkg.bundle2Str(b) & ": ",
                  bundleData_v(o)(b));
              end if;
            end loop bndle;
          end loop orient;

        end loop columns;
      end loop rows;

    end loop busSelect;

-- separator
      baseDefs_pkg.fprintf(debug_f, "------------");

    baseDefs_pkg.fprintf(debug_f, "INFO: End of bus test.");
  end busTest;

end busTest_pkg;
```

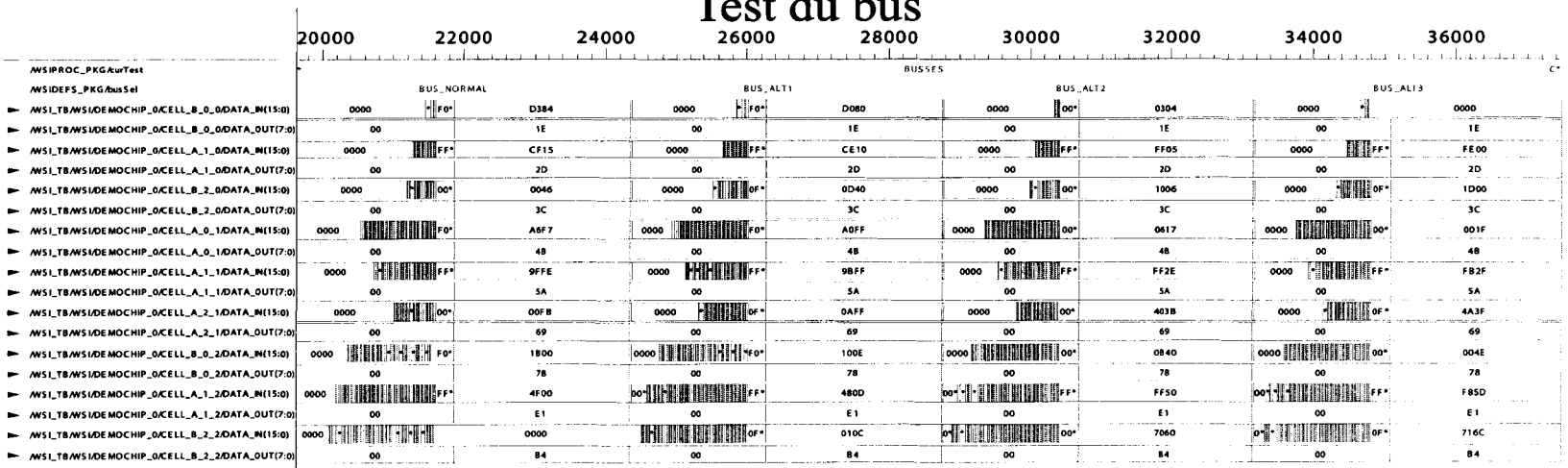
```
-- WSI demochip3 testbench
-- Normand Leclerc, Ðcole de technologie sup&rieure
-- Hyperchip May 2001

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.baseDefs_pkg.all; use work.baseDefs_pkg;
use work.WSIDefs_pkg.all; use work.WSIDefs_pkg;
use work.JTAG_pkg.all; use work.JTAG_pkg;
use work.WSIJTAG_pkg.all; use work.WSIJTAG_pkg;
use work.WSIProc_pkg.all; use work.WSIProc_pkg;
use work.busTest_pkg.all; use work.busTest_pkg;

library std;
use std.textio.all; use std.textio;

library tpz973gtc ;
library vst_n18_sc_tsm_c4_typ ;
use vst_n18_sc_tsm_c4_typ.components.all;
use tpz973gtc.all;

entity WSI_tb is
  generic(constant outFileName_g: string := "WSI_tb.txt");
end WSI_tb;
```

```
-- WSI demochip3 testbench
-- Normand Leclerc, Ecole de technologie sup&rieure
-- Hyperchip May 2001
architecture WSI_behav of WSI_tb is

begin

-- WSI Chip instantiation
  wsi: DemoChip_chip port map (
    JTAG_pkg.TCK, JTAG_pkg.TMS, JTAG_pkg.nTRST,
    JTAG_pkg.TDI, JTAG_pkg.TDO,
    WSIDefs_pkg.DI_00,WSIDefs_pkg.DI_02,WSIDefs_pkg.DI_20,WSIDefs_pkg.DI_22,
   WSIDefs_pkg.DO_00,WSIDefs_pkg.DO_02,WSIDefs_pkg.DO_20,WSIDefs_pkg.DO_22,
   WSIDefs_pkg.TDO_22
  );

-- TCK control process
-- This process is controlled by TCKEn signal
  TCKControl: process
  begin
    JTAG_pkg.JTAGTCKControl(JTAG_pkg.TCK);
  end process TCKControl;

-----------------------
-- main test routine --
-----------------------
  testProc: process
    file file_f: textio.text open write_mode is outFileName_g;
  begin

-- Signals initialization
-- JTAG
    JTAG_pkg.TCKEn <= false;  -- disable TCK
    JTAG_pkg.nTRST <= '0';  -- maintain reset for a while
    JTAG_pkg.TMS <= '0';
    JTAG_pkg.TDI <= '0';
    WSIDefs_pkg.DI_00 <= (others=>'0');
    WSIDefs_pkg.DI_02 <= (others=>'0');
    WSIDefs_pkg.DI_20 <= (others=>'0');
    WSIDefs_pkg.DI_22 <= (others=>'0');

   init(WSIProc_pkg.curTest, JTAG_pkg.TCKEn, JTAG_pkg.TMS, JTAG_pkg.TDI,
     JTAG_pkg.nTRST, file_f);

-- begin tests
    baseDefs_pkg.fprintf(file_f, "INFO: Begining of tests.");

--    busTest_pkg.busTest(curTest, JTAG_pkg.JTAGOutputData, TCKEn, TMS, TDI,
      nTRST, file_f);

    cfgTest_pkg.cfgTest(curTest, JTAG_pkg.JTAGOutputData, TCKEn, TMS, TDI,
      nTRST, file_f);

    baseDefs_pkg.fprintf(file_f, "INFO: End of tests.");

-- prevents a second test
    wait for 500 ns;
    assert false
      report "Simulation ended, stopping simulator." severity failure;
  end process testProc;

end WSI_behav;

configuration WSI_tb_cfg of WSI_tb is
  for WSI_behav
  end for;
end WSI_tb_cfg;
```

ANNEXE 3

Résultats théoriques
de tests du démonstrateur 4

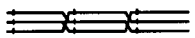
Test du bus



Note: les collones sont alternées configuration/données valides.

DATA_IN: Données provenant du bus. Format (par quartet): Horizontal$_1$ Horizontal$_2$ Vertical$_2$ Vertical$_1$.
DATA_OUT: Données transmises sur le bus. Format (par quartet): Horizontal/Vertical.

Structure du bus horizontal:



Structure du bus vertical:

BUS_NORMAL:
Bus horizontaux: 1 et 2 vers la gauche.
Bus verticaux: 1 et 2 vers le haut.

BUS_ALT1:
Bus horizontaux: 1 vers la gauche, 2 vers la droite.
Bus verticaux: 1 vers le bas, 2 vers le haut.

BUS_ALT2:
Bus horizontaux: 1 vers la droite, 2 vers la gauche.
Bus verticaux: 1 vers le haut, 2 vers le bas.

BUS_ALT3:
Bus horizontaux: 1 vers la droite, 2 vers la droite.
Bus verticaux: 1 vers le bas, 2 vers le bas.

Il y a inversion lorsque les données changent de la région délimitée par la cellule. Par exemple, il y a inversion lorsque la donnée passe de la cellule (0,0) à la cellule (0,1) et il y a une seconde inversion lorsque cette donnée arrive à la cellule (0,2). Ainsi, les données horizontales transmises par la cellule (0,1) seront vues comme inversées pour la cellule (0,0) alors que celles de la cellule (0,2), inversées deux fois, seront directes.

Exemples:

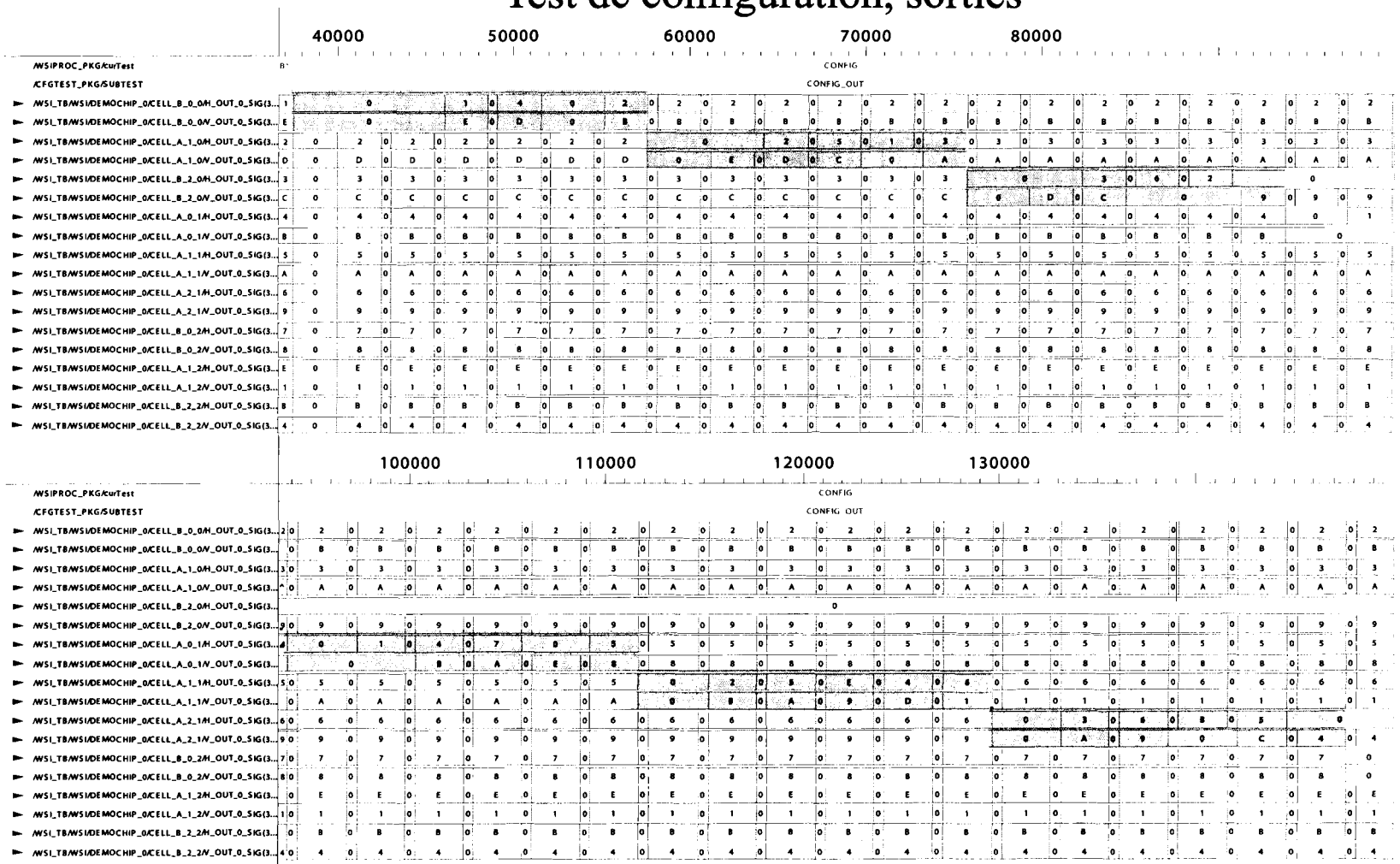
BUS_NORMAL, cellule A(0,1)

Donnée horizontale du bus 1 provenant de la droite soit 5 de A(1,1). La donnée est inversée une fois donc A.
Donnée horizontale du bus 2 provenant de la droite soit le bus 1 de la cellule A(1,1) qui provient de la cellule A(2,1) qui transmet 6. La donnée est inversée deux fois ce qui donne toujours 6.
Donnée verticale du bus 1 provenant du bas soit 8 de A(0,2). La donnée est inversée une fois donc 7.
Donnée verticale du bus 2 provenant du bas soit le bus 1 de la cellule A(0,2) qui est mis à la masse donnant un 0. Cette valeur est inversée une fois pour donner F.
Le vecteur résultat écrit dans le format énoncé de DATA_IN devient: A6F7.

BUS_ALT2, cellule B(0,2)

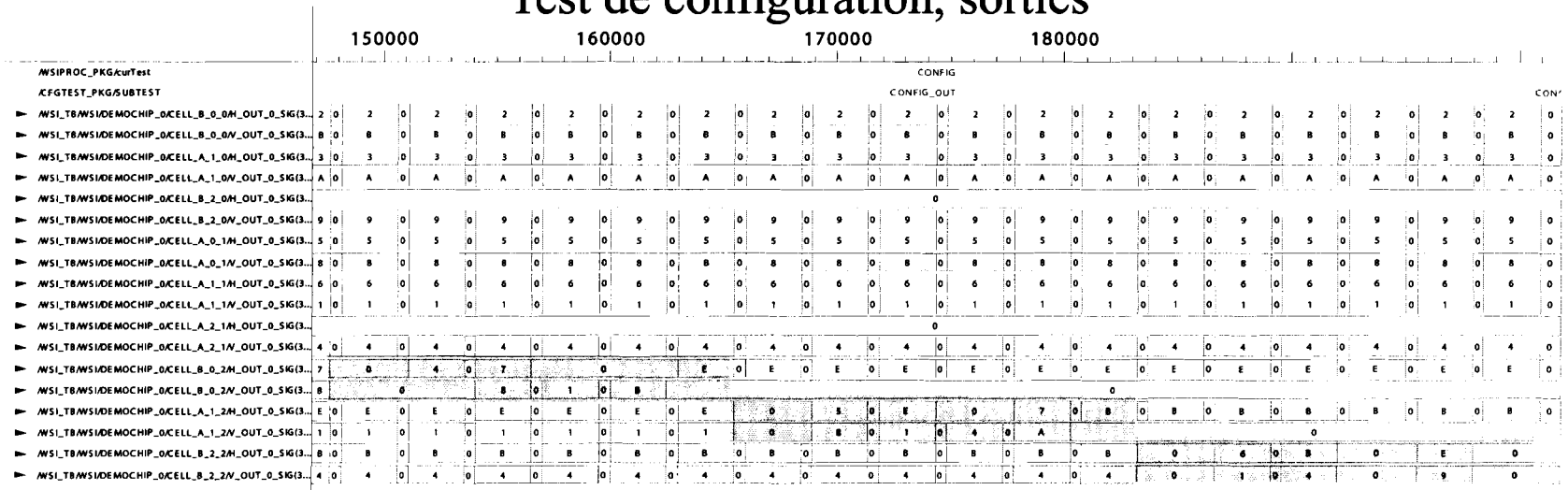
Donnée horizontale du bus 1 provenant de la gauche qui est forcée à 0. Cette donnée n'est pas inversée.
Donnée horizontale du bus 2 provenant de la droite soit le bus 1 de la cellule A(1,2) qui provient de la cellule B(2,2) qui transmet B. Cette donnée est inversée deux fois donnant toujours B.
Donnée verticale du bus 1 provenant du bas qui est forcé à 0 et ne subissant aucune inversion.
Donnée verticale du bus 2 provenant du haut soit B de A(0,1). La donnée est inversée ce qui donne 4.
Le vecteur résultant écrit dans le format énoncé devient: 0B40.

Test de configuration, sorties



Note: Les résultats sont montrés sur deux lignes la première étant la sortie du multiplexeur horizontal (configuration verticale) et la seconde étant la sortie du multiplexeur vertical (configuration horizontale).

Test de configuration, sorties



Note: Les résultats sont montrés sur deux lignes la première étant la sortie du multiplexeur horizontal (configuration verticale) et la seconde étant la sortie du multiplexeur vertical (configuration horizontale).

Multiplexeurs de sortie horizontale (configuration verticale):

1. VSS
2. Quartet 0 horizontal de la cellule du haut
3. Quartet 0 horizontal
4. Quartet 0 horizontal de la cellule du bas
5. Quartet 0 horizontal de la cellule de gauche
6. Quartet 0 horizontal de la cellule de droite

Multiplexeurs de sortie verticale (configuration horizontale):

1. VSS
2. Quartet 0 vertical de la cellule de gauche
3. Quartet 0 vertical
4. Quartet 0 vertical de la cellule de droite
5. Quartet 0 vertical de la cellule du haut
6. Quartet 0 vertical de la cellule du bas

Exemples:

Cellule B(2,0) @ 76000.
Multiplexeur de sortie horizontale:

1. VSS (0).
2. Quartet 0 horizontal de la cellule du haut qui n'existe pas donc 0.
3. Quartet 0 horizontal soit B(2,0) de valeur 3.
4. Quartet 0 horizontal de la cellule du bas soit A(2,1) de valeur 6.
5. Quartet 0 horizontal de la cellule de gauche soit A(1,0) de valeur 2.
6. Quartet 0 horizontal de la cellule de droite qui n'existe pas donc 0.

Multiplexeur de sortie verticale:

1. VSS (0).
2. Quartet 0 vertical de la cellule gauche soit A(1,0) de valeur D.
3. Quartet 0 vertical soit B(2,0) de valeur C.
4. Quartet 0 vertical de la cellule droite qui n'existe pas donc 0.
5. Quartet 0 vertical de la cellule du haut qui n'existe pas donc 0.
6. Quartet 0 vertical de la cellule du bas soit A(2,1) de valeur 9.

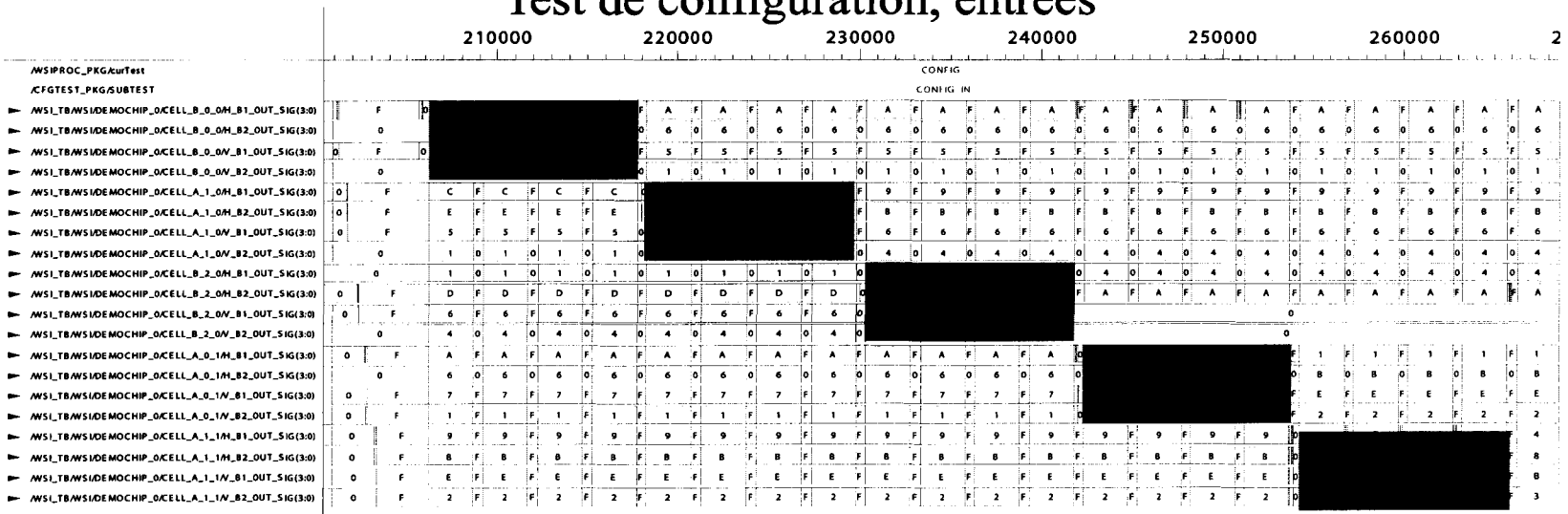
Cellule A(1,2) @ 1766000.
Multiplexeur de sortie horizontale:

1. VSS (0).
2. Quartet 0 horizontal de la cellule du haut soit A(1,1) de valeur 5.
3. Quartet 0 horizontal soit B(1,2) de valeur E.
4. Quartet 0 horizontal de la cellule du bas qui n'existe pas donc 0.
5. Quartet 0 horizontal de la cellule de gauche soit A(0,2) de valeur 7.
6. Quartet 0 horizontal de la cellule de droite soit B(2,2) de valeur B.

Multiplexeur de sortie verticale:

1. VSS (0).
2. Quartet 0 vertical de la cellule gauche soit A(0,2) de valeur 8.
3. Quartet 0 vertical soit B(1,2) de valeur 1.
4. Quartet 0 vertical de la cellule droite soit B(2,2) de valeur 4.
5. Quartet 0 vertical de la cellule du haut soit A(1,1) de valeur A.
6. Quartet 0 vertical de la cellule du bas qui n'existe pas donc 0.

Test de configuration, entrées



Note: Le résultat de chacune des cellules est présenté sur 4 lignes. La sortie de la configuration verticale (sortie horizontale) des quartets 1 et 2 horizontaux sont suivis par la sortie de la configuration horizontale (sortie verticale) des quartets 1 et 2 verticaux.

Configuration verticale:

0. Sortie du multiplexeur attaché à l'émetteur horizontal
1. Sortie du multiplexeur attaché au récepteur horizontal de la cellule du haut
2. Sortie du multiplexeur attaché au récepteur horizontal
3. Sortie du multiplexeur attaché au récepteur horizontal de la cellule du bas

Configuration horizontale:

0. Sortie du multiplexeur attaché à l'émetteur vertical de la cellule
1. Sortie du multiplexeur attaché au récepteur vertical de la cellule de gauche
2. Sortie du multiplexeur attaché au récepteur vertical
3. Sortie du multiplexeur d'entrée vertical de la cellule de droite.

Exemple:

Cellule A(0,1) @ 242000.
Configuration verticale:
Quartet 1:

0. Sortie de l'émetteur horizontal de A(0,1), 4.
1. Récepteur du quartet 1 horizontal de B(0,0) qui reçoit sa donnée de A(1,0), 2 inversé: D.
2. Récepteur du quartet 1 horizontal de A(0,1) recevant sa donnée de A(1,1), 5 inversé: A.
3. Récepteur du quartet 1 horizontal de B(0,2) dont la donnée provient de A(1,2), E inversé: 1.

Quartet 2:

0. Sortie de l'émetteur horizontal de A(0,1), 4.
1. Récepteur du quartet 2 horizontal de B(0,0) qui reçoit sa donnée de B(2,0), 3 inversé 2 fois: 3.
2. Récepteur du quartet 2 horizontal de A(0,1) recevant sa donnée de A(2,1), 6 inversé 2 fois: 6.
3. Récepteur du quartet 2 horizontal de B(0,2) dont la donnée provient de B(2,2), B inversé 2 fois: B.

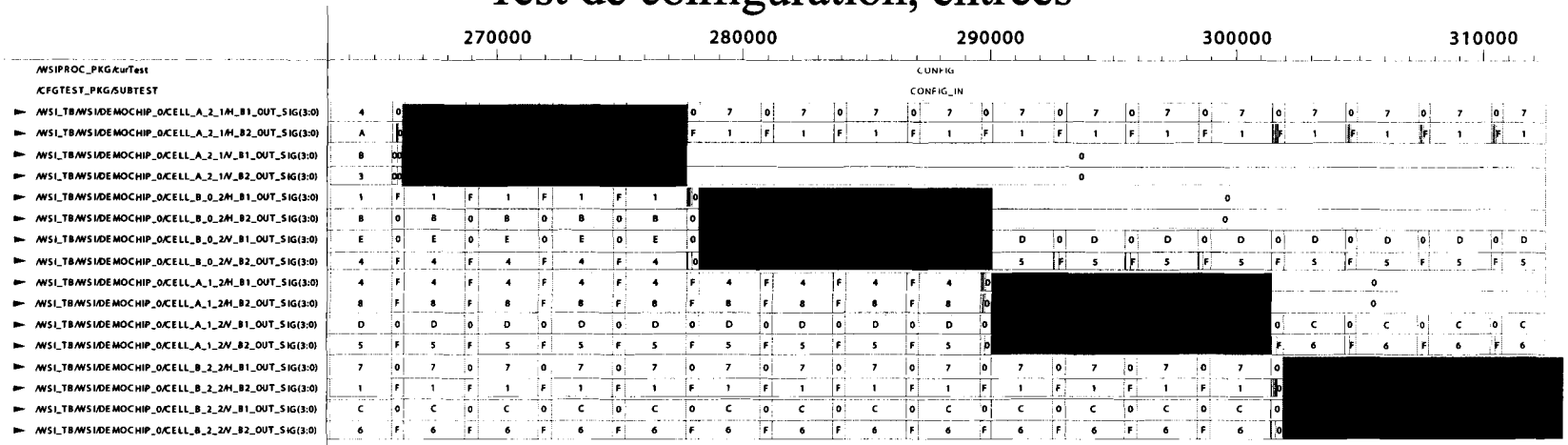
Configuration horizontale:
Quartet 1:

0. Sortie de l'émetteur vertical de A(0,1), B.
1. Récepteur du quartet 1 vertical d'une cellule inexistante, donc 0.
2. Récepteur du quartet 1 vertical de A(0,1) recevant sa donnée de B(0,1), 8 inversé: 7.
3. Récepteur du quartet 1 vertical de A(1,1) dont la donnée provient de A(1,2), 1 inversé: E.

Quartet 2:

0. Sortie de l'émetteur vertical de A(0,1), B.
1. Récepteur du quartet 2 vertical d'une cellule inexistante, donc 0.
2. Récepteur du quartet 2 vertical de A(0,1) recevant sa donnée de B(0,0), E inversé: 1.
3. Récepteur du quartet 2 vertical de A(1,1) dont la donnée provient de A(1,0), D inversé: 2.

Test de configuration, entrées



Note: Le résultat de chacune des cellules est présenté sur 4 lignes. La sortie de la configuration verticale (sortie horizontale) des quartets 1 et 2 horizontaux sont suivis par la sortie de la configuration horizontale (sortie verticale) des quartets 1 et 2 verticaux.

Configuration verticale:

0. Sortie du multiplexeur attaché à l'émetteur horizontal
1. Sortie du multiplexeur attaché au récepteur horizontal de la cellule du haut
2. Sortie du multiplexeur attaché au récepteur horizontal
3. Sortie du multiplexeur attaché au récepteur horizontal de la cellule du bas

Configuration horizontale:

0. Sortie du multiplexeur attaché à l'émetteur vertical de la cellule
1. Sortie du multiplexeur attaché au récepteur vertical de la cellule de gauche
2. Sortie du multiplexeur attaché au récepteur vertical
3. Sortie du multiplexeur d'entrée vertical de la cellule de droite.

Exemple:

Cellule A(1,2) @ 290000.
Configuration verticale:
Quartet 1:

0. Sortie de l'émetteur horizontal de A(1,2), E.
1. Récepteur du quartet 1 horizontal de A(1,1) qui reçoit sa donnée de A(2,1), 6 inversé: 9.
2. Récepteur du quartet 1 horizontal de A(1,2) recevant sa donnée de B(2,2), B inversé: 4.
3. Récepteur du quartet 1 horizontal d'une cellule inexistante, donc 0.

Quartet 2:

0. Sortie de l'émetteur horizontal de A(1,2), E.
1. Récepteur du quartet 2 horizontal de A(1,1) qui reçoit sa donnée de A(0,1), 4 inversé: B.
2. Récepteur du quartet 2 horizontal de A(1,2) recevant sa donnée de B(0,2), 7 inversé: 8.
3. Récepteur du quartet 2 horizontal d'une cellule inexistante, donc 0.

Configuration horizontale:
Quartet 1:

0. Sortie de l'émetteur vertical de A(1,2), 1.
1. Récepteur du quartet 1 vertical de B(0,2) recevant sa donnée de B(0,0), E inversé 2 fois: E.
2. Récepteur du quartet 1 vertical de A(1,2) recevant sa donnée de A(1,0), D inversé 2 fois: D.
3. Récepteur du quartet 1 vertical de B(2,2) dont la donnée provient de A(2,0), C inversé 2 fois: C.

Quartet 2:

0. Sortie de l'émetteur vertical de A(0,1), B.
1. Récepteur du quartet 2 vertical de B(0,2) recevant sa donnée de A(0,1), B inversé: 4.
2. Récepteur du quartet 2 vertical de A(1,2) recevant sa donnée de A(1,1), A inversé: 5.
3. Récepteur du quartet 2 vertical de B(2,2) dont la donnée provient de A(2,1), 9 inversé: 6.

Résultats textuels du banc de test du démonstrateur 4.

```
(0 ns) INFO: Begining of tests.
(0 ns) INFO: JTAG test.
(0 ns) INFO: resetting chip (HARD).
(20 ns) INFO: done.
(20 ns) INFO: Instruction register sanity check
(270 ns) INFO: done
(270 ns) INFO: Bypass and instruction registers test
(270 ns) INFO: Sending vector: 101010101
(520 ns) INFO: done
(520 ns) INFO: Configuration chain test
(520 ns) INFO: Testing cell (0,0).
(1370 ns) INFO: Testing cell (0,1).
(2220 ns) INFO: Testing cell (0,2).
(3070 ns) INFO: Testing cell (1,0).
(3920 ns) INFO: Testing cell (1,1).
(4770 ns) INFO: Testing cell (1,2).
(5620 ns) INFO: Testing cell (2,0).
(6470 ns) INFO: Testing cell (2,1).
(7320 ns) INFO: Testing cell (2,2).
(8170 ns) INFO: done
(8170 ns) INFO: Data chain test
(8170 ns) INFO: Testing cell (0,0).
(9140 ns) INFO: Testing cell (0,1).
(10110 ns) INFO: Testing cell (0,2).
(11080 ns) INFO: Testing cell (1,0).
(12050 ns) INFO: Testing cell (1,1).
(13020 ns) INFO: Testing cell (1,2).
(13990 ns) INFO: Testing cell (2,0).
(14960 ns) INFO: Testing cell (2,1).
(15930 ns) INFO: Testing cell (2,2).
(16900 ns) INFO: done
(16900 ns) INFO: End of JTAG test.
(16900 ns) INFO: resetting chip (HARD).
(16920 ns) INFO: done.
(16920 ns) INFO: Setting config up.
(17230 ns) INFO: done.
(17230 ns) INFO: Setting cells data up.
(19700 ns) INFO: done.
(19700 ns) INFO: Starting bus test.
(19700 ns) ------------
(19700 ns) INFO: Configuring cells for Bus Normal.
(21630 ns) INFO: Done.
(24100 ns) INFO: Testing cell (0,0).
(24100 ns) INFO: horizontal data output for bundle1: 1101
(24100 ns) INFO: horizontal data output for bundle2: 0011
(24100 ns) INFO: vertical data output for bundle1: 0100
(24100 ns) INFO: vertical data output for bundle2: 1000
(24100 ns) INFO: Testing cell (0,1).
(24100 ns) INFO: horizontal data output for bundle1: 1100
(24100 ns) INFO: horizontal data output for bundle2: 1111
(24100 ns) INFO: vertical data output for bundle1: 0101
(24100 ns) INFO: vertical data output for bundle2: 0001
(24100 ns) INFO: Testing cell (0,2).
(24100 ns) INFO: horizontal data output for bundle1: 0000
(24100 ns) INFO: horizontal data output for bundle2: 0000
(24100 ns) INFO: vertical data output for bundle1: 0110
(24100 ns) INFO: vertical data output for bundle2: 0100
(24100 ns) INFO: Testing cell (1,0).
(24100 ns) INFO: horizontal data output for bundle1: 1010
(24100 ns) INFO: horizontal data output for bundle2: 0110
(24100 ns) INFO: vertical data output for bundle1: 0111
(24100 ns) INFO: vertical data output for bundle2: 1111
(24100 ns) INFO: Testing cell (1,1).
(24100 ns) INFO: horizontal data output for bundle1: 1001
```

```
(24100 ns) INFO: horizontal data output for bundle2: 1111
(24100 ns) INFO: vertical data output for bundle1: 1110
(24100 ns) INFO: vertical data output for bundle2: 1111
(24100 ns) INFO: Testing cell (1,2).
(24100 ns) INFO: horizontal data output for bundle1: 0000
(24100 ns) INFO: horizontal data output for bundle2: 0000
(24100 ns) INFO: vertical data output for bundle1: 1011
(24100 ns) INFO: vertical data output for bundle2: 1111
(24100 ns) INFO: Testing cell (2,0).
(24100 ns) INFO: horizontal data output for bundle1: 0001
(24100 ns) INFO: horizontal data output for bundle2: 1011
(24100 ns) INFO: vertical data output for bundle1: 0000
(24100 ns) INFO: vertical data output for bundle2: 0000
(24100 ns) INFO: Testing cell (2,1).
(24100 ns) INFO: horizontal data output for bundle1: 0100
(24100 ns) INFO: horizontal data output for bundle2: 1111
(24100 ns) INFO: vertical data output for bundle1: 0000
(24100 ns) INFO: vertical data output for bundle2: 0000
(24100 ns) INFO: Testing cell (2,2).
(24100 ns) INFO: horizontal data output for bundle1: 0000
(24100 ns) INFO: horizontal data output for bundle2: 0000
(24100 ns) INFO: vertical data output for bundle1: 0000
(24100 ns) INFO: vertical data output for bundle2: 0000
(24100 ns) ------------
(24100 ns) INFO: Configuring cells for Bus alternate 1.
(26030 ns) INFO: Done.
(28500 ns) INFO: Testing cell (0,0).
(28500 ns) INFO: horizontal data output for bundle1: 1101
(28500 ns) INFO: horizontal data output for bundle2: 0000
(28500 ns) INFO: vertical data output for bundle1: 0000
(28500 ns) INFO: vertical data output for bundle2: 1000
(28500 ns) INFO: Testing cell (0,1).
(28500 ns) INFO: horizontal data output for bundle1: 1100
(28500 ns) INFO: horizontal data output for bundle2: 1110
(28500 ns) INFO: vertical data output for bundle1: 0000
(28500 ns) INFO: vertical data output for bundle2: 0001
(28500 ns) INFO: Testing cell (0,2).
(28500 ns) INFO: horizontal data output for bundle1: 0000
(28500 ns) INFO: horizontal data output for bundle2: 1101
(28500 ns) INFO: vertical data output for bundle1: 0000
(28500 ns) INFO: vertical data output for bundle2: 0100
(28500 ns) INFO: Testing cell (1,0).
(28500 ns) INFO: horizontal data output for bundle1: 1010
(28500 ns) INFO: horizontal data output for bundle2: 0000
(28500 ns) INFO: vertical data output for bundle1: 1111
(28500 ns) INFO: vertical data output for bundle2: 1111
(28500 ns) INFO: Testing cell (1,1).
(28500 ns) INFO: horizontal data output for bundle1: 1001
(28500 ns) INFO: horizontal data output for bundle2: 1011
(28500 ns) INFO: vertical data output for bundle1: 1111
(28500 ns) INFO: vertical data output for bundle2: 1111
(28500 ns) INFO: Testing cell (1,2).
(28500 ns) INFO: horizontal data output for bundle1: 0000
(28500 ns) INFO: horizontal data output for bundle2: 1010
(28500 ns) INFO: vertical data output for bundle1: 1111
(28500 ns) INFO: vertical data output for bundle2: 1111
(28500 ns) INFO: Testing cell (2,0).
(28500 ns) INFO: horizontal data output for bundle1: 0001
(28500 ns) INFO: horizontal data output for bundle2: 0000
(28500 ns) INFO: vertical data output for bundle1: 1110
(28500 ns) INFO: vertical data output for bundle2: 0000
(28500 ns) INFO: Testing cell (2,1).
(28500 ns) INFO: horizontal data output for bundle1: 0100
(28500 ns) INFO: horizontal data output for bundle2: 1000
(28500 ns) INFO: vertical data output for bundle1: 1101
(28500 ns) INFO: vertical data output for bundle2: 0000
(28500 ns) INFO: Testing cell (2,2).
```

```
(28500 ns) INFO: horizontal data output for bundle1: 0000
(28500 ns) INFO: horizontal data output for bundle2: 0001
(28500 ns) INFO: vertical data output for bundle1: 1100
(28500 ns) INFO: vertical data output for bundle2: 0000
(28500 ns) ------------
(28500 ns) INFO: Configuring cells for Bus alternate 2.
(30430 ns) INFO: Done.
(32900 ns) INFO: Testing cell (0,0).
(32900 ns) INFO: horizontal data output for bundle1: 0000
(32900 ns) INFO: horizontal data output for bundle2: 0011
(32900 ns) INFO: vertical data output for bundle1: 0100
(32900 ns) INFO: vertical data output for bundle2: 0000
(32900 ns) INFO: Testing cell (0,1).
(32900 ns) INFO: horizontal data output for bundle1: 1111
(32900 ns) INFO: horizontal data output for bundle2: 1111
(32900 ns) INFO: vertical data output for bundle1: 0101
(32900 ns) INFO: vertical data output for bundle2: 0000
(32900 ns) INFO: Testing cell (0,2).
(32900 ns) INFO: horizontal data output for bundle1: 0001
(32900 ns) INFO: horizontal data output for bundle2: 0000
(32900 ns) INFO: vertical data output for bundle1: 0110
(32900 ns) INFO: vertical data output for bundle2: 0000
(32900 ns) INFO: Testing cell (1,0).
(32900 ns) INFO: horizontal data output for bundle1: 0000
(32900 ns) INFO: horizontal data output for bundle2: 0110
(32900 ns) INFO: vertical data output for bundle1: 0111
(32900 ns) INFO: vertical data output for bundle2: 0001
(32900 ns) INFO: Testing cell (1,1).
(32900 ns) INFO: horizontal data output for bundle1: 1111
(32900 ns) INFO: horizontal data output for bundle2: 1111
(32900 ns) INFO: vertical data output for bundle1: 1110
(32900 ns) INFO: vertical data output for bundle2: 0010
(32900 ns) INFO: Testing cell (1,2).
(32900 ns) INFO: horizontal data output for bundle1: 0100
(32900 ns) INFO: horizontal data output for bundle2: 0000
(32900 ns) INFO: vertical data output for bundle1: 1011
(32900 ns) INFO: vertical data output for bundle2: 0011
(32900 ns) INFO: Testing cell (2,0).
(32900 ns) INFO: horizontal data output for bundle1: 0000
(32900 ns) INFO: horizontal data output for bundle2: 1011
(32900 ns) INFO: vertical data output for bundle1: 0000
(32900 ns) INFO: vertical data output for bundle2: 0100
(32900 ns) INFO: Testing cell (2,1).
(32900 ns) INFO: horizontal data output for bundle1: 1111
(32900 ns) INFO: horizontal data output for bundle2: 1111
(32900 ns) INFO: vertical data output for bundle1: 0000
(32900 ns) INFO: vertical data output for bundle2: 0101
(32900 ns) INFO: Testing cell (2,2).
(32900 ns) INFO: horizontal data output for bundle1: 0111
(32900 ns) INFO: horizontal data output for bundle2: 0000
(32900 ns) INFO: vertical data output for bundle1: 0000
(32900 ns) INFO: vertical data output for bundle2: 0110

(32900 ns) ------------
(32900 ns) INFO: Configuring cells for Bus alternate 3.
(34830 ns) INFO: Done.
(37300 ns) INFO: Testing cell (0,0).
(37300 ns) INFO: horizontal data output for bundle1: 0000
(37300 ns) INFO: horizontal data output for bundle2: 0000
(37300 ns) INFO: vertical data output for bundle1: 0000
(37300 ns) INFO: vertical data output for bundle2: 0000
(37300 ns) INFO: Testing cell (0,1).
(37300 ns) INFO: horizontal data output for bundle1: 1111
(37300 ns) INFO: horizontal data output for bundle2: 1110
(37300 ns) INFO: vertical data output for bundle1: 0000
(37300 ns) INFO: vertical data output for bundle2: 0000
(37300 ns) INFO: Testing cell (0,2).
```

```
(37300 ns) INFO: horizontal data output for bundle1: 0001
(37300 ns) INFO: horizontal data output for bundle2: 1101
(37300 ns) INFO: vertical data output for bundle1: 0000
(37300 ns) INFO: vertical data output for bundle2: 0000
(37300 ns) INFO: Testing cell (1,0).
(37300 ns) INFO: horizontal data output for bundle1: 0000
(37300 ns) INFO: horizontal data output for bundle2: 0000
(37300 ns) INFO: vertical data output for bundle1: 1111
(37300 ns) INFO: vertical data output for bundle2: 0001
(37300 ns) INFO: Testing cell (1,1).
(37300 ns) INFO: horizontal data output for bundle1: 1111
(37300 ns) INFO: horizontal data output for bundle2: 1011
(37300 ns) INFO: vertical data output for bundle1: 1111
(37300 ns) INFO: vertical data output for bundle2: 0010
(37300 ns) INFO: Testing cell (1,2).
(37300 ns) INFO: horizontal data output for bundle1: 0100
(37300 ns) INFO: horizontal data output for bundle2: 1010
(37300 ns) INFO: vertical data output for bundle1: 1111
(37300 ns) INFO: vertical data output for bundle2: 0011
(37300 ns) INFO: Testing cell (2,0).
(37300 ns) INFO: horizontal data output for bundle1: 0000
(37300 ns) INFO: horizontal data output for bundle2: 0000
(37300 ns) INFO: vertical data output for bundle1: 1110
(37300 ns) INFO: vertical data output for bundle2: 0100
(37300 ns) INFO: Testing cell (2,1).
(37300 ns) INFO: horizontal data output for bundle1: 1111
(37300 ns) INFO: horizontal data output for bundle2: 1000
(37300 ns) INFO: vertical data output for bundle1: 1101
(37300 ns) INFO: vertical data output for bundle2: 0101
(37300 ns) INFO: Testing cell (2,2).
(37300 ns) INFO: horizontal data output for bundle1: 0111
(37300 ns) INFO: horizontal data output for bundle2: 0001
(37300 ns) INFO: vertical data output for bundle1: 1100
(37300 ns) INFO: vertical data output for bundle2: 0110
(37300 ns) ------------
(37300 ns) INFO: End of bus test.
(37300 ns) ------------
(37300 ns) INFO: Configuration test.
(37300 ns) ............
(37300 ns) INFO: Outputs.
(37300 ns) INFO: Configuring cells for Bus Normal; reading self.
(39230 ns) INFO: Done.
(39230 ns) -
(39230 ns) INFO: Testing cell (0,0).
(39790 ns) INFO: Configuration 0
(42260 ns) INFO: horizontal config output for bundle1: 0000
(42260 ns) INFO: horizontal config output for bundle2: 0000
(42260 ns) INFO: vertical config output for bundle1: 0000
(42260 ns) INFO: vertical config output for bundle2: 0000
(42820 ns) INFO: Configuration 1
(45290 ns) INFO: horizontal config output for bundle1: 0000
(45290 ns) INFO: horizontal config output for bundle2: 0000
(45290 ns) INFO: vertical config output for bundle1: 0000
(45290 ns) INFO: vertical config output for bundle2: 0000
(45850 ns) INFO: Configuration 2
(48320 ns) INFO: horizontal config output for bundle1: 1110
(48320 ns) INFO: horizontal config output for bundle2: 1110
(48320 ns) INFO: vertical config output for bundle1: 0001
(48320 ns) INFO: vertical config output for bundle2: 0001
(48880 ns) INFO: Configuration 3
(51350 ns) INFO: horizontal config output for bundle1: 1101
(51350 ns) INFO: horizontal config output for bundle2: 1101
(51350 ns) INFO: vertical config output for bundle1: 0100
(51350 ns) INFO: vertical config output for bundle2: 0100
(51910 ns) INFO: Configuration 4
(54380 ns) INFO: horizontal config output for bundle1: 0000
(54380 ns) INFO: horizontal config output for bundle2: 0000
```

```
(54380 ns) INFO: vertical config output for bundle1: 0000
(54380 ns) INFO: vertical config output for bundle2: 0000
(54940 ns) INFO: Configuration 5
(57410 ns) INFO: horizontal config output for bundle1: 1011
(57410 ns) INFO: horizontal config output for bundle2: 1011
(57410 ns) INFO: vertical config output for bundle1: 0010
(57410 ns) INFO: vertical config output for bundle2: 0010
(57410 ns) -
(57410 ns) INFO: Testing cell (0,1).
(57960 ns) INFO: Configuration 0
(60430 ns) INFO: horizontal config output for bundle1: 0000
(60430 ns) INFO: horizontal config output for bundle2: 0000
(60430 ns) INFO: vertical config output for bundle1: 0000
(60430 ns) INFO: vertical config output for bundle2: 0000
(60980 ns) INFO: Configuration 1
(63450 ns) INFO: horizontal config output for bundle1: 1110
(63450 ns) INFO: horizontal config output for bundle2: 1110
(63450 ns) INFO: vertical config output for bundle1: 0000
(63450 ns) INFO: vertical config output for bundle2: 0000
(64000 ns) INFO: Configuration 2
(66470 ns) INFO: horizontal config output for bundle1: 1101
(66470 ns) INFO: horizontal config output for bundle2: 1101
(66470 ns) INFO: vertical config output for bundle1: 0010
(66470 ns) INFO: vertical config output for bundle2: 0010
(67020 ns) INFO: Configuration 3
(69490 ns) INFO: horizontal config output for bundle1: 1100
(69490 ns) INFO: horizontal config output for bundle2: 1100
(69490 ns) INFO: vertical config output for bundle1: 0101
(69490 ns) INFO: vertical config output for bundle2: 0101
(70040 ns) INFO: Configuration 4
(72510 ns) INFO: horizontal config output for bundle1: 0000
(72510 ns) INFO: horizontal config output for bundle2: 0000
(72510 ns) INFO: vertical config output for bundle1: 0001
(72510 ns) INFO: vertical config output for bundle2: 0001
(73060 ns) INFO: Configuration 5
(75530 ns) INFO: horizontal config output for bundle1: 1010
(75530 ns) INFO: horizontal config output for bundle2: 1010
(75530 ns) INFO: vertical config output for bundle1: 0011
(75530 ns) INFO: vertical config output for bundle2: 0011
(75530 ns) -
(75530 ns) INFO: Testing cell (0,2).
(76070 ns) INFO: Configuration 0
(78540 ns) INFO: horizontal config output for bundle1: 0000
(78540 ns) INFO: horizontal config output for bundle2: 0000
(78540 ns) INFO: vertical config output for bundle1: 0000
(78540 ns) INFO: vertical config output for bundle2: 0000
(79080 ns) INFO: Configuration 1
(81550 ns) INFO: horizontal config output for bundle1: 1101
(81550 ns) INFO: horizontal config output for bundle2: 1101
(81550 ns) INFO: vertical config output for bundle1: 0000
(81550 ns) INFO: vertical config output for bundle2: 0000
(82090 ns) INFO: Configuration 2
(84560 ns) INFO: horizontal config output for bundle1: 1100
(84560 ns) INFO: horizontal config output for bundle2: 1100
(84560 ns) INFO: vertical config output for bundle1: 0011
(84560 ns) INFO: vertical config output for bundle2: 0011
(85100 ns) INFO: Configuration 3
(87570 ns) INFO: horizontal config output for bundle1: 0000
(87570 ns) INFO: horizontal config output for bundle2: 0000
(87570 ns) INFO: vertical config output for bundle1: 0110
(87570 ns) INFO: vertical config output for bundle2: 0110
(88110 ns) INFO: Configuration 4
(90580 ns) INFO: horizontal config output for bundle1: 0000
(90580 ns) INFO: horizontal config output for bundle2: 0000
(90580 ns) INFO: vertical config output for bundle1: 0010
(90580 ns) INFO: vertical config output for bundle2: 0010
(91120 ns) INFO: Configuration 5
```

```
(93590 ns) INFO: horizontal config output for bundle1: 1001
(93590 ns) INFO: horizontal config output for bundle2: 1001
(93590 ns) INFO: vertical config output for bundle1: 0000
(93590 ns) INFO: vertical config output for bundle2: 0000
(93590 ns) -
(93590 ns) INFO: Testing cell (1,0).
(94100 ns) INFO: Configuration 0
(96570 ns) INFO: horizontal config output for bundle1: 0000
(96570 ns) INFO: horizontal config output for bundle2: 0000
(96570 ns) INFO: vertical config output for bundle1: 0000
(96570 ns) INFO: vertical config output for bundle2: 0000
(97080 ns) INFO: Configuration 1
(99550 ns) INFO: horizontal config output for bundle1: 0000
(99550 ns) INFO: horizontal config output for bundle2: 0000
(99550 ns) INFO: vertical config output for bundle1: 0001
(99550 ns) INFO: vertical config output for bundle2: 0001
(100060 ns) INFO: Configuration 2
(102530 ns) INFO: horizontal config output for bundle1: 1011
(102530 ns) INFO: horizontal config output for bundle2: 1011
(102530 ns) INFO: vertical config output for bundle1: 0100
(102530 ns) INFO: vertical config output for bundle2: 0100
(103040 ns) INFO: Configuration 3
(105510 ns) INFO: horizontal config output for bundle1: 1010
(105510 ns) INFO: horizontal config output for bundle2: 1010
(105510 ns) INFO: vertical config output for bundle1: 0111
(105510 ns) INFO: vertical config output for bundle2: 0111
(106020 ns) INFO: Configuration 4
(108490 ns) INFO: horizontal config output for bundle1: 1110
(108490 ns) INFO: horizontal config output for bundle2: 1110
(108490 ns) INFO: vertical config output for bundle1: 0000
(108490 ns) INFO: vertical config output for bundle2: 0000
(109000 ns) INFO: Configuration 5
(111470 ns) INFO: horizontal config output for bundle1: 1000
(111470 ns) INFO: horizontal config output for bundle2: 1000
(111470 ns) INFO: vertical config output for bundle1: 0101
(111470 ns) INFO: vertical config output for bundle2: 0101
(111470 ns) -
(111470 ns) INFO: Testing cell (1,1).
(111990 ns) INFO: Configuration 0
(114460 ns) INFO: horizontal config output for bundle1: 0000
(114460 ns) INFO: horizontal config output for bundle2: 0000
(114460 ns) INFO: vertical config output for bundle1: 0000
(114460 ns) INFO: vertical config output for bundle2: 0000
(114980 ns) INFO: Configuration 1
(117450 ns) INFO: horizontal config output for bundle1: 1011
(117450 ns) INFO: horizontal config output for bundle2: 1011
(117450 ns) INFO: vertical config output for bundle1: 0010
(117450 ns) INFO: vertical config output for bundle2: 0010
(117970 ns) INFO: Configuration 2
(120440 ns) INFO: horizontal config output for bundle1: 1010
(120440 ns) INFO: horizontal config output for bundle2: 1010
(120440 ns) INFO: vertical config output for bundle1: 0101
(120440 ns) INFO: vertical config output for bundle2: 0101
(120960 ns) INFO: Configuration 3
(123430 ns) INFO: horizontal config output for bundle1: 1001
(123430 ns) INFO: horizontal config output for bundle2: 1001
(123430 ns) INFO: vertical config output for bundle1: 1110
(123430 ns) INFO: vertical config output for bundle2: 1110
(123950 ns) INFO: Configuration 4
(126420 ns) INFO: horizontal config output for bundle1: 1101
(126420 ns) INFO: horizontal config output for bundle2: 1101
(126420 ns) INFO: vertical config output for bundle1: 0100
(126420 ns) INFO: vertical config output for bundle2: 0100
(126940 ns) INFO: Configuration 5
(129410 ns) INFO: horizontal config output for bundle1: 0001
(129410 ns) INFO: horizontal config output for bundle2: 0001
(129410 ns) INFO: vertical config output for bundle1: 0110
```

```
(129410 ns) INFO: vertical config output for bundle2: 0110
(129410 ns) -
(129410 ns) INFO: Testing cell (1,2).
(129940 ns) INFO: Configuration 0
(132410 ns) INFO: horizontal config output for bundle1: 0000
(132410 ns) INFO: horizontal config output for bundle2: 0000
(132410 ns) INFO: vertical config output for bundle1: 0000
(132410 ns) INFO: vertical config output for bundle2: 0000
(132940 ns) INFO: Configuration 1
(135410 ns) INFO: horizontal config output for bundle1: 1010
(135410 ns) INFO: horizontal config output for bundle2: 1010
(135410 ns) INFO: vertical config output for bundle1: 0011
(135410 ns) INFO: vertical config output for bundle2: 0011
(135940 ns) INFO: Configuration 2
(138410 ns) INFO: horizontal config output for bundle1: 1001
(138410 ns) INFO: horizontal config output for bundle2: 1001
(138410 ns) INFO: vertical config output for bundle1: 0110
(138410 ns) INFO: vertical config output for bundle2: 0110
(138940 ns) INFO: Configuration 3
(141410 ns) INFO: horizontal config output for bundle1: 0000
(141410 ns) INFO: horizontal config output for bundle2: 0000
(141410 ns) INFO: vertical config output for bundle1: 1011
(141410 ns) INFO: vertical config output for bundle2: 1011
(141940 ns) INFO: Configuration 4
(144410 ns) INFO: horizontal config output for bundle1: 1100
(144410 ns) INFO: horizontal config output for bundle2: 1100
(144410 ns) INFO: vertical config output for bundle1: 0101
(144410 ns) INFO: vertical config output for bundle2: 0101
(144940 ns) INFO: Configuration 5
(147410 ns) INFO: horizontal config output for bundle1: 0100
(147410 ns) INFO: horizontal config output for bundle2: 0100
(147410 ns) INFO: vertical config output for bundle1: 0000
(147410 ns) INFO: vertical config output for bundle2: 0000
(147410 ns) -
(147410 ns) INFO: Testing cell (2,0).
(147910 ns) INFO: Configuration 0
(150380 ns) INFO: horizontal config output for bundle1: 0000
(150380 ns) INFO: horizontal config output for bundle2: 0000
(150380 ns) INFO: vertical config output for bundle1: 0000
(150380 ns) INFO: vertical config output for bundle2: 0000
(150880 ns) INFO: Configuration 1
(153350 ns) INFO: horizontal config output for bundle1: 0000
(153350 ns) INFO: horizontal config output for bundle2: 0000
(153350 ns) INFO: vertical config output for bundle1: 0100
(153350 ns) INFO: vertical config output for bundle2: 0100
(153850 ns) INFO: Configuration 2
(156320 ns) INFO: horizontal config output for bundle1: 1000
(156320 ns) INFO: horizontal config output for bundle2: 1000
(156320 ns) INFO: vertical config output for bundle1: 0111
(156320 ns) INFO: vertical config output for bundle2: 0111
(156820 ns) INFO: Configuration 3
(159290 ns) INFO: horizontal config output for bundle1: 0001
(159290 ns) INFO: horizontal config output for bundle2: 0001
(159290 ns) INFO: vertical config output for bundle1: 0000
(159290 ns) INFO: vertical config output for bundle2: 0000
(159790 ns) INFO: Configuration 4
(162260 ns) INFO: horizontal config output for bundle1: 1011
(162260 ns) INFO: horizontal config output for bundle2: 1011
(162260 ns) INFO: vertical config output for bundle1: 0000
(162260 ns) INFO: vertical config output for bundle2: 0000
(162760 ns) INFO: Configuration 5
(165230 ns) INFO: horizontal config output for bundle1: 0000
(165230 ns) INFO: horizontal config output for bundle2: 0000
(165230 ns) INFO: vertical config output for bundle1: 1110
(165230 ns) INFO: vertical config output for bundle2: 1110
(165230 ns) -
(165230 ns) INFO: Testing cell (2,1).
```

```
(165720 ns) INFO: Configuration 0
(168190 ns) INFO: horizontal config output for bundle1: 0000
(168190 ns) INFO: horizontal config output for bundle2: 0000
(168190 ns) INFO: vertical config output for bundle1: 0000
(168190 ns) INFO: vertical config output for bundle2: 0000
(168680 ns) INFO: Configuration 1
(171150 ns) INFO: horizontal config output for bundle1: 1000
(171150 ns) INFO: horizontal config output for bundle2: 1000
(171150 ns) INFO: vertical config output for bundle1: 0101
(171150 ns) INFO: vertical config output for bundle2: 0101
(171640 ns) INFO: Configuration 2
(174110 ns) INFO: horizontal config output for bundle1: 0001
(174110 ns) INFO: horizontal config output for bundle2: 0001
(174110 ns) INFO: vertical config output for bundle1: 1110
(174110 ns) INFO: vertical config output for bundle2: 1110
(174600 ns) INFO: Configuration 3
(177070 ns) INFO: horizontal config output for bundle1: 0100
(177070 ns) INFO: horizontal config output for bundle2: 0100
(177070 ns) INFO: vertical config output for bundle1: 0000
(177070 ns) INFO: vertical config output for bundle2: 0000
(177560 ns) INFO: Configuration 4
(180030 ns) INFO: horizontal config output for bundle1: 1010
(180030 ns) INFO: horizontal config output for bundle2: 1010
(180030 ns) INFO: vertical config output for bundle1: 0111
(180030 ns) INFO: vertical config output for bundle2: 0111
(180520 ns) INFO: Configuration 5
(182990 ns) INFO: horizontal config output for bundle1: 0000
(182990 ns) INFO: horizontal config output for bundle2: 0000
(182990 ns) INFO: vertical config output for bundle1: 1011
(182990 ns) INFO: vertical config output for bundle2: 1011
(182990 ns) -
(182990 ns) INFO: Testing cell (2,2).
(183470 ns) INFO: Configuration 0
(185940 ns) INFO: horizontal config output for bundle1: 0000
(185940 ns) INFO: horizontal config output for bundle2: 0000
(185940 ns) INFO: vertical config output for bundle1: 0000
(185940 ns) INFO: vertical config output for bundle2: 0000
(186420 ns) INFO: Configuration 1
(188890 ns) INFO: horizontal config output for bundle1: 0001
(188890 ns) INFO: horizontal config output for bundle2: 0001
(188890 ns) INFO: vertical config output for bundle1: 0110
(188890 ns) INFO: vertical config output for bundle2: 0110
(189370 ns) INFO: Configuration 2
(191840 ns) INFO: horizontal config output for bundle1: 0100
(191840 ns) INFO: horizontal config output for bundle2: 0100
(191840 ns) INFO: vertical config output for bundle1: 1011
(191840 ns) INFO: vertical config output for bundle2: 1011
(192320 ns) INFO: Configuration 3
(194790 ns) INFO: horizontal config output for bundle1: 0000
(194790 ns) INFO: horizontal config output for bundle2: 0000
(194790 ns) INFO: vertical config output for bundle1: 0000
(194790 ns) INFO: vertical config output for bundle2: 0000
(195270 ns) INFO: Configuration 4
(197740 ns) INFO: horizontal config output for bundle1: 1001
(197740 ns) INFO: horizontal config output for bundle2: 1001
(197740 ns) INFO: vertical config output for bundle1: 1110
(197740 ns) INFO: vertical config output for bundle2: 1110
(198220 ns) INFO: Configuration 5
(200690 ns) INFO: horizontal config output for bundle1: 0000
(200690 ns) INFO: horizontal config output for bundle2: 0000
(200690 ns) INFO: vertical config output for bundle1: 0000
(200690 ns) INFO: vertical config output for bundle2: 0000
(200690 ns) INFO: Done.
(200690 ns) ............
(200690 ns) INFO: Inputs.
(200690 ns) INFO: Configuring cells for Bus Normal; reading assigned bundles.
(200690 ns) INFO: Config for (0,0): 001010010010101000
```

```
(201250 ns) INFO: Config for (0,1): 011010010010101000
(201800 ns) INFO: Config for (0,2): 111010010010101000
(202340 ns) INFO: Config for (1,0): 001010010010101010
(202850 ns) INFO: Config for (1,1): 011010010010101010
(203370 ns) INFO: Config for (1,2): 111010010010101010
(203900 ns) INFO: Config for (2,0): 001010010010101011
(204400 ns) INFO: Config for (2,1): 011010010010101011
(204890 ns) INFO: Config for (2,2): 111010010010101011
(205370 ns) INFO: Done.
(205370 ns) -
(205370 ns) INFO: Testing cell (0,0).
(205930 ns) INFO: sent configuration 0
(208400 ns) INFO: horizontal config input for bundle1: 1110
(208400 ns) INFO: horizontal config input for bundle2: 1110
(208400 ns) INFO: vertical config input for bundle1: 0001
(208400 ns) INFO: vertical config input for bundle2: 0001
(208960 ns) INFO: sent configuration 1
(211430 ns) INFO: horizontal config input for bundle1: 0000
(211430 ns) INFO: horizontal config input for bundle2: 0000
(211430 ns) INFO: vertical config input for bundle1: 0000
(211430 ns) INFO: vertical config input for bundle2: 0000
(211990 ns) INFO: sent configuration 2
(214460 ns) INFO: horizontal config input for bundle1: 0100
(214460 ns) INFO: horizontal config input for bundle2: 1000
(214460 ns) INFO: vertical config input for bundle1: 1101
(214460 ns) INFO: vertical config input for bundle2: 0011
(215020 ns) INFO: sent configuration 3
(217490 ns) INFO: horizontal config input for bundle1: 0101
(217490 ns) INFO: horizontal config input for bundle2: 0001
(217490 ns) INFO: vertical config input for bundle1: 1010
(217490 ns) INFO: vertical config input for bundle2: 0110
(217490 ns) -
(217490 ns) INFO: Testing cell (0,1).
(218040 ns) INFO: sent configuration 0
(220510 ns) INFO: horizontal config input for bundle1: 1101
(220510 ns) INFO: horizontal config input for bundle2: 1101
(220510 ns) INFO: vertical config input for bundle1: 0010
(220510 ns) INFO: vertical config input for bundle2: 0010
(221060 ns) INFO: sent configuration 1
(223530 ns) INFO: horizontal config input for bundle1: 0100
(223530 ns) INFO: horizontal config input for bundle2: 1000
(223530 ns) INFO: vertical config input for bundle1: 0000
(223530 ns) INFO: vertical config input for bundle2: 0000
(224080 ns) INFO: sent configuration 2
(226550 ns) INFO: horizontal config input for bundle1: 0101
(226550 ns) INFO: horizontal config input for bundle2: 0001
(226550 ns) INFO: vertical config input for bundle1: 1100
(226550 ns) INFO: vertical config input for bundle2: 1110
(227100 ns) INFO: sent configuration 3
(229570 ns) INFO: horizontal config input for bundle1: 0110
(229570 ns) INFO: horizontal config input for bundle2: 0100
(229570 ns) INFO: vertical config input for bundle1: 1001
(229570 ns) INFO: vertical config input for bundle2: 1011
(229570 ns) -
(229570 ns) INFO: Testing cell (0,2).
(230110 ns) INFO: sent configuration 0
(232580 ns) INFO: horizontal config input for bundle1: 1100
(232580 ns) INFO: horizontal config input for bundle2: 1100
(232580 ns) INFO: vertical config input for bundle1: 0011
(232580 ns) INFO: vertical config input for bundle2: 0011
(233120 ns) INFO: sent configuration 1
(235590 ns) INFO: horizontal config input for bundle1: 0101
(235590 ns) INFO: horizontal config input for bundle2: 0001
(235590 ns) INFO: vertical config input for bundle1: 0000
(235590 ns) INFO: vertical config input for bundle2: 0000
(236130 ns) INFO: sent configuration 2
(238600 ns) INFO: horizontal config input for bundle1: 0110
```

```
(238600 ns) INFO: horizontal config input for bundle2: 0100
(238600 ns) INFO: vertical config input for bundle1: 0001
(238600 ns) INFO: vertical config input for bundle2: 1101
(239140 ns) INFO: sent configuration 3
(241610 ns) INFO: horizontal config input for bundle1: 0000
(241610 ns) INFO: horizontal config input for bundle2: 0000
(241610 ns) INFO: vertical config input for bundle1: 0100
(241610 ns) INFO: vertical config input for bundle2: 1010
(241610 ns) -
(241610 ns) INFO: Testing cell (1,0).
(242120 ns) INFO: sent configuration 0
(244590 ns) INFO: horizontal config input for bundle1: 1011
(244590 ns) INFO: horizontal config input for bundle2: 1011
(244590 ns) INFO: vertical config input for bundle1: 0100
(244590 ns) INFO: vertical config input for bundle2: 0100
(245100 ns) INFO: sent configuration 1
(247570 ns) INFO: horizontal config input for bundle1: 0000
(247570 ns) INFO: horizontal config input for bundle2: 0000
(247570 ns) INFO: vertical config input for bundle1: 1101
(247570 ns) INFO: vertical config input for bundle2: 0011
(248080 ns) INFO: sent configuration 2
(250550 ns) INFO: horizontal config input for bundle1: 0111
(250550 ns) INFO: horizontal config input for bundle2: 0001
(250550 ns) INFO: vertical config input for bundle1: 1010
(250550 ns) INFO: vertical config input for bundle2: 0110
(251060 ns) INFO: sent configuration 3
(253530 ns) INFO: horizontal config input for bundle1: 1110
(253530 ns) INFO: horizontal config input for bundle2: 0010
(253530 ns) INFO: vertical config input for bundle1: 0001
(253530 ns) INFO: vertical config input for bundle2: 1011
(253530 ns) -
(253530 ns) INFO: Testing cell (1,1).
(254050 ns) INFO: sent configuration 0
(256520 ns) INFO: horizontal config input for bundle1: 1010
(256520 ns) INFO: horizontal config input for bundle2: 1010
(256520 ns) INFO: vertical config input for bundle1: 0101
(256520 ns) INFO: vertical config input for bundle2: 0101
(257040 ns) INFO: sent configuration 1
(259510 ns) INFO: horizontal config input for bundle1: 0111
(259510 ns) INFO: horizontal config input for bundle2: 0001
(259510 ns) INFO: vertical config input for bundle1: 1100
(259510 ns) INFO: vertical config input for bundle2: 1110
(260030 ns) INFO: sent configuration 2
(262500 ns) INFO: horizontal config input for bundle1: 1110
(262500 ns) INFO: horizontal config input for bundle2: 0010
(262500 ns) INFO: vertical config input for bundle1: 1001
(262500 ns) INFO: vertical config input for bundle2: 1011
(263020 ns) INFO: sent configuration 3
(265490 ns) INFO: horizontal config input for bundle1: 1011
(265490 ns) INFO: horizontal config input for bundle2: 0011
(265490 ns) INFO: vertical config input for bundle1: 0100
(265490 ns) INFO: vertical config input for bundle2: 1000
(265490 ns) -
(265490 ns) INFO: Testing cell (1,2).
(266020 ns) INFO: sent configuration 0
(268490 ns) INFO: horizontal config input for bundle1: 1001
(268490 ns) INFO: horizontal config input for bundle2: 1001
(268490 ns) INFO: vertical config input for bundle1: 0110
(268490 ns) INFO: vertical config input for bundle2: 0110
(269020 ns) INFO: sent configuration 1
(271490 ns) INFO: horizontal config input for bundle1: 1110
(271490 ns) INFO: horizontal config input for bundle2: 0010
(271490 ns) INFO: vertical config input for bundle1: 0001
(271490 ns) INFO: vertical config input for bundle2: 1101
(272020 ns) INFO: sent configuration 2
(274490 ns) INFO: horizontal config input for bundle1: 1011
(274490 ns) INFO: horizontal config input for bundle2: 0011
```

```
(274490 ns) INFO: vertical config input for bundle1: 0100
(274490 ns) INFO: vertical config input for bundle2: 1010
(275020 ns) INFO: sent configuration 3
(277490 ns) INFO: horizontal config input for bundle1: 0000
(277490 ns) INFO: horizontal config input for bundle2: 0000
(277490 ns) INFO: vertical config input for bundle1: 0111
(277490 ns) INFO: vertical config input for bundle2: 0001
(277490 ns) -
(277490 ns) INFO: Testing cell (2,0).
(277990 ns) INFO: sent configuration 0
(280460 ns) INFO: horizontal config input for bundle1: 1000
(280460 ns) INFO: horizontal config input for bundle2: 1000
(280460 ns) INFO: vertical config input for bundle1: 0111
(280460 ns) INFO: vertical config input for bundle2: 0111
(280960 ns) INFO: sent configuration 1
(283430 ns) INFO: horizontal config input for bundle1: 0000
(283430 ns) INFO: horizontal config input for bundle2: 0000
(283430 ns) INFO: vertical config input for bundle1: 1010
(283430 ns) INFO: vertical config input for bundle2: 0110
(283930 ns) INFO: sent configuration 2
(286400 ns) INFO: horizontal config input for bundle1: 1110
(286400 ns) INFO: horizontal config input for bundle2: 0100
(286400 ns) INFO: vertical config input for bundle1: 0001
(286400 ns) INFO: vertical config input for bundle2: 1011
(286900 ns) INFO: sent configuration 3
(289370 ns) INFO: horizontal config input for bundle1: 1101
(289370 ns) INFO: horizontal config input for bundle2: 0101
(289370 ns) INFO: vertical config input for bundle1: 0000
(289370 ns) INFO: vertical config input for bundle2: 0000
(289370 ns) -
(289370 ns) INFO: Testing cell (2,1).
(289860 ns) INFO: sent configuration 0
(292330 ns) INFO: horizontal config input for bundle1: 0001
(292330 ns) INFO: horizontal config input for bundle2: 0001
(292330 ns) INFO: vertical config input for bundle1: 1110
(292330 ns) INFO: vertical config input for bundle2: 1110
(292820 ns) INFO: sent configuration 1
(295290 ns) INFO: horizontal config input for bundle1: 1110
(295290 ns) INFO: horizontal config input for bundle2: 0100
(295290 ns) INFO: vertical config input for bundle1: 1001
(295290 ns) INFO: vertical config input for bundle2: 1011
(295780 ns) INFO: sent configuration 2
(298250 ns) INFO: horizontal config input for bundle1: 1101
(298250 ns) INFO: horizontal config input for bundle2: 0101
(298250 ns) INFO: vertical config input for bundle1: 0100
(298250 ns) INFO: vertical config input for bundle2: 1000
(298740 ns) INFO: sent configuration 3
(301210 ns) INFO: horizontal config input for bundle1: 1100
(301210 ns) INFO: horizontal config input for bundle2: 0110
(301210 ns) INFO: vertical config input for bundle1: 0000
(301210 ns) INFO: vertical config input for bundle2: 0000
(301210 ns) -
(301210 ns) INFO: Testing cell (2,2).
(301690 ns) INFO: sent configuration 0
(304160 ns) INFO: horizontal config input for bundle1: 0100
(304160 ns) INFO: horizontal config input for bundle2: 0100
(304160 ns) INFO: vertical config input for bundle1: 1011
(304160 ns) INFO: vertical config input for bundle2: 1011
(304640 ns) INFO: sent configuration 1
(307110 ns) INFO: horizontal config input for bundle1: 1101
(307110 ns) INFO: horizontal config input for bundle2: 0101
(307110 ns) INFO: vertical config input for bundle1: 0100
(307110 ns) INFO: vertical config input for bundle2: 1010
(307590 ns) INFO: sent configuration 2
(310060 ns) INFO: horizontal config input for bundle1: 1100
(310060 ns) INFO: horizontal config input for bundle2: 0110
(310060 ns) INFO: vertical config input for bundle1: 0111
```

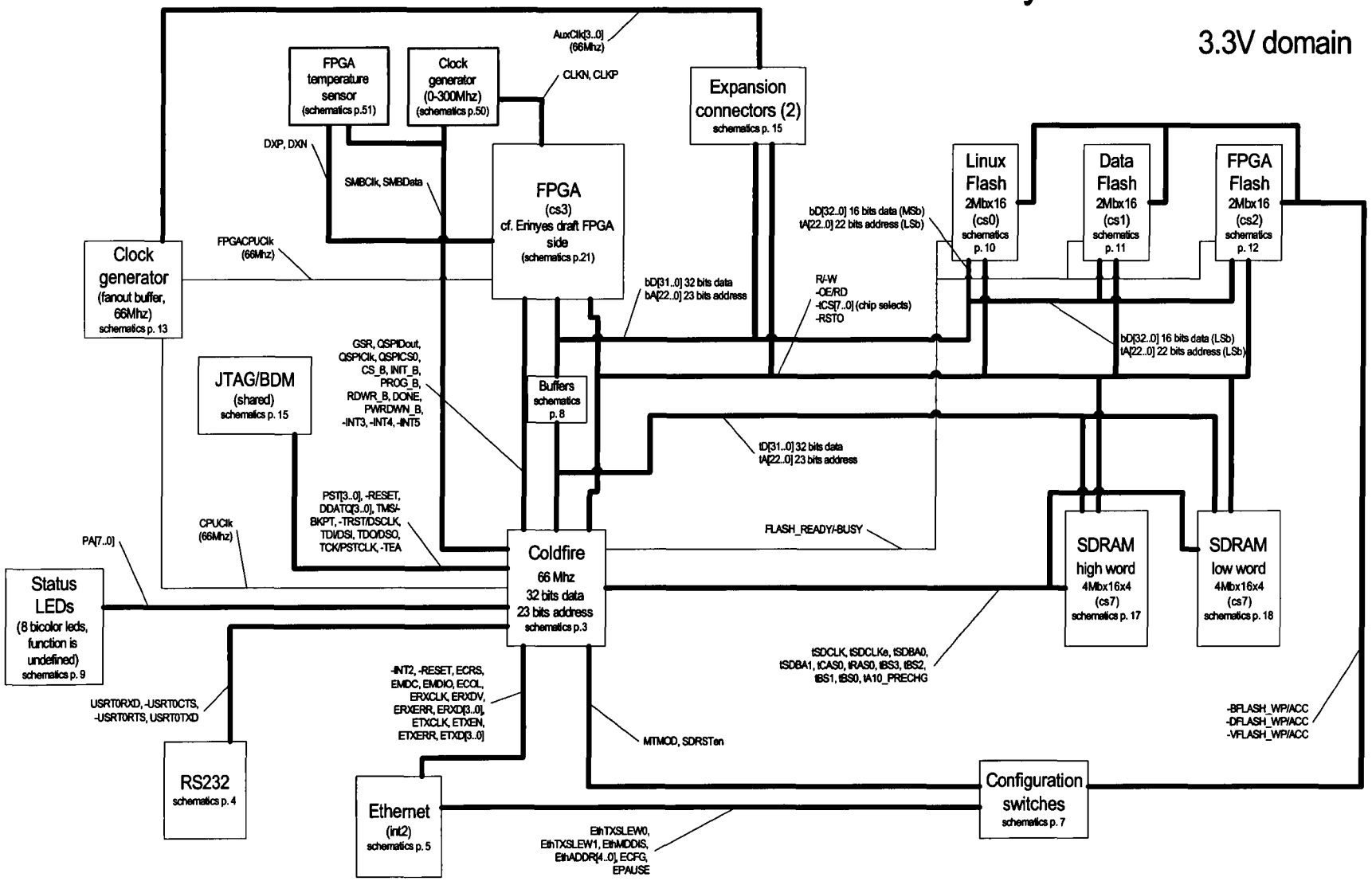
```
(310060 ns) INFO: vertical config input for bundle2: 0001
(310540 ns) INFO: sent configuration 3
(313010 ns) INFO: horizontal config input for bundle1: 0000
(313010 ns) INFO: horizontal config input for bundle2: 0000
(313010 ns) INFO: vertical config input for bundle1: 0000
(313010 ns) INFO: vertical config input for bundle2: 0000
(313010 ns) INFO: End of configuration test.
(313010 ns) INFO: End of tests.
```

ANNEXE 4

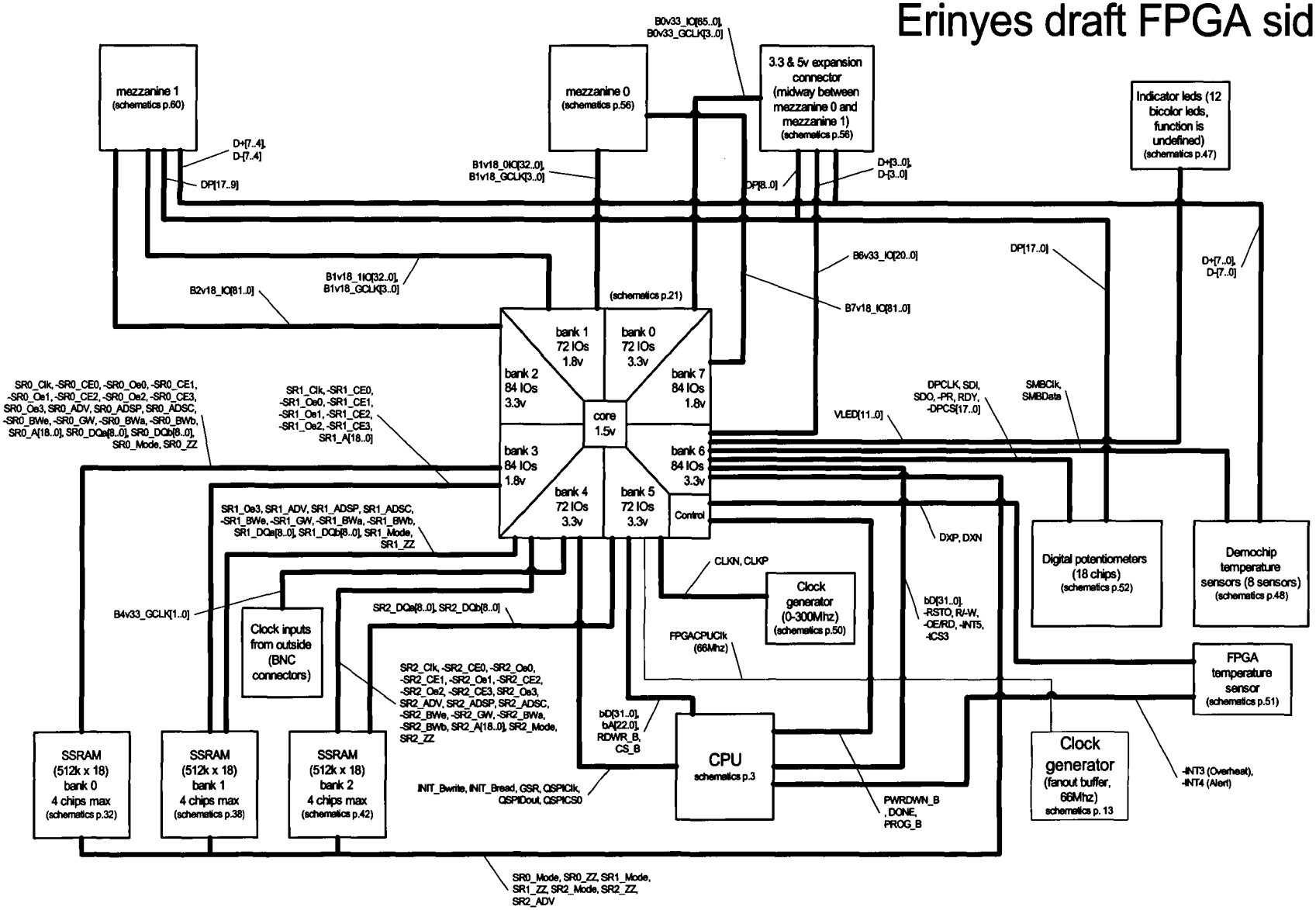
Représentation par blocs de la carte de tests

Erinyes draft CPU side

3.3V domain

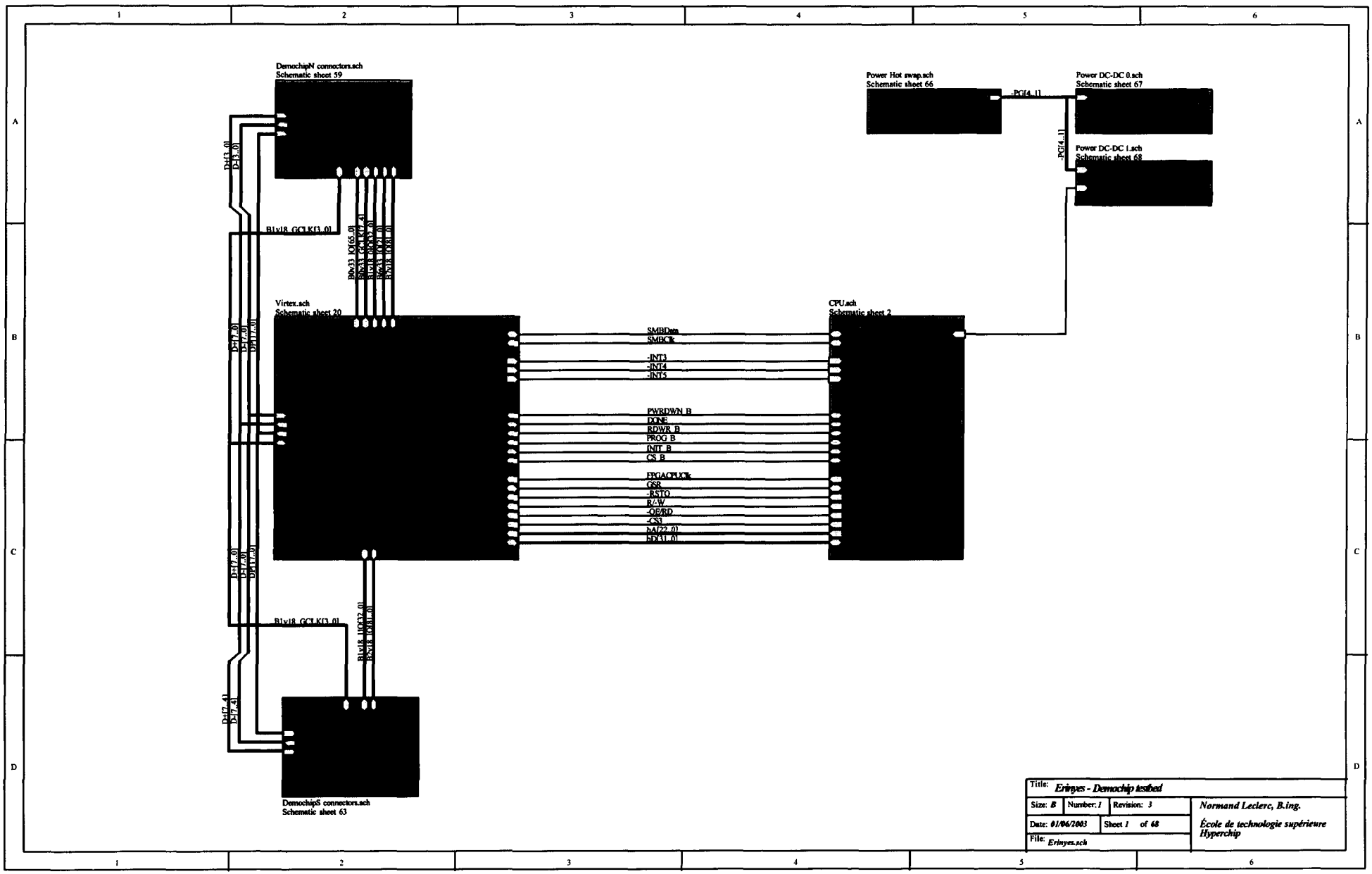


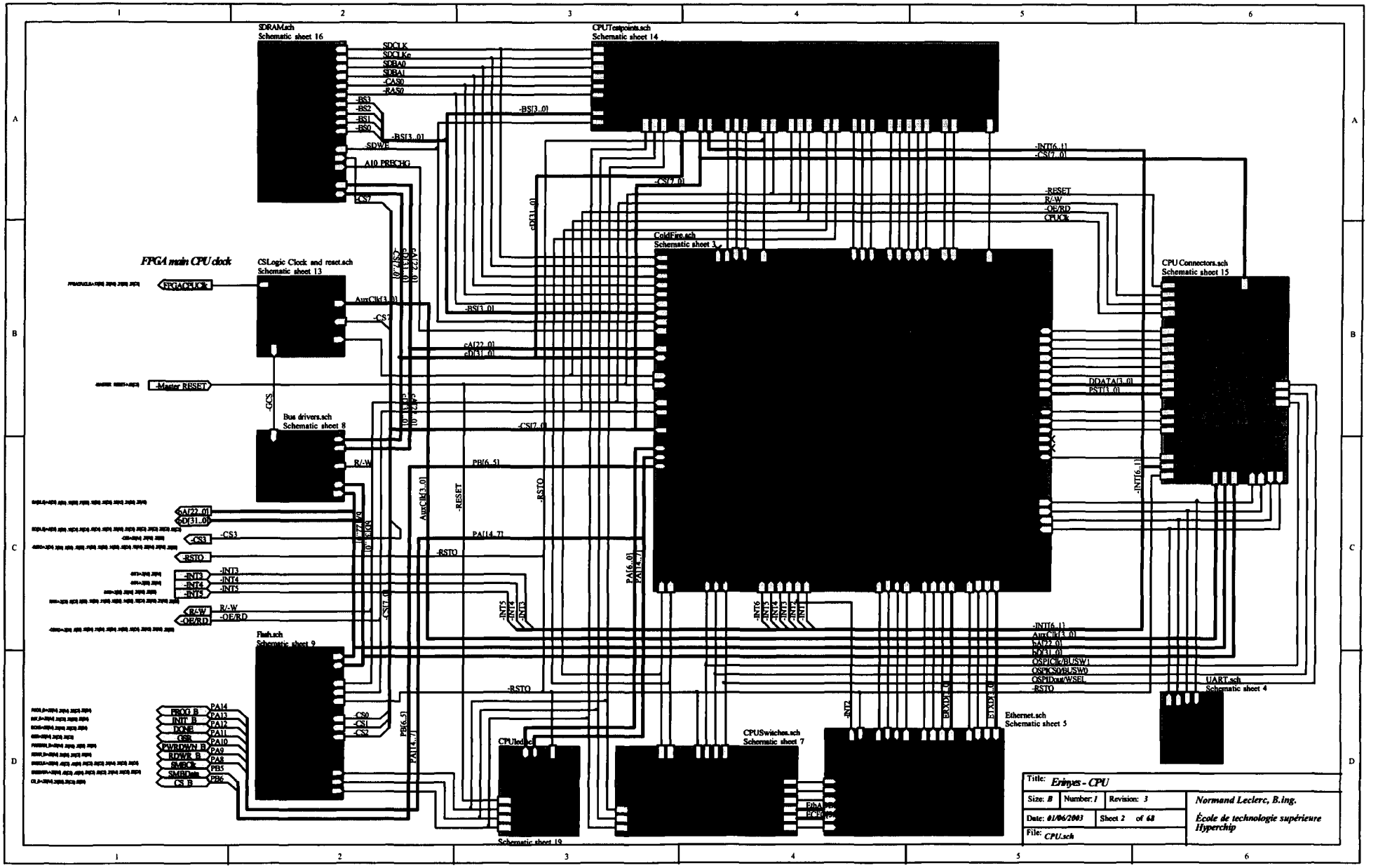
Erinyes draft FPGA side

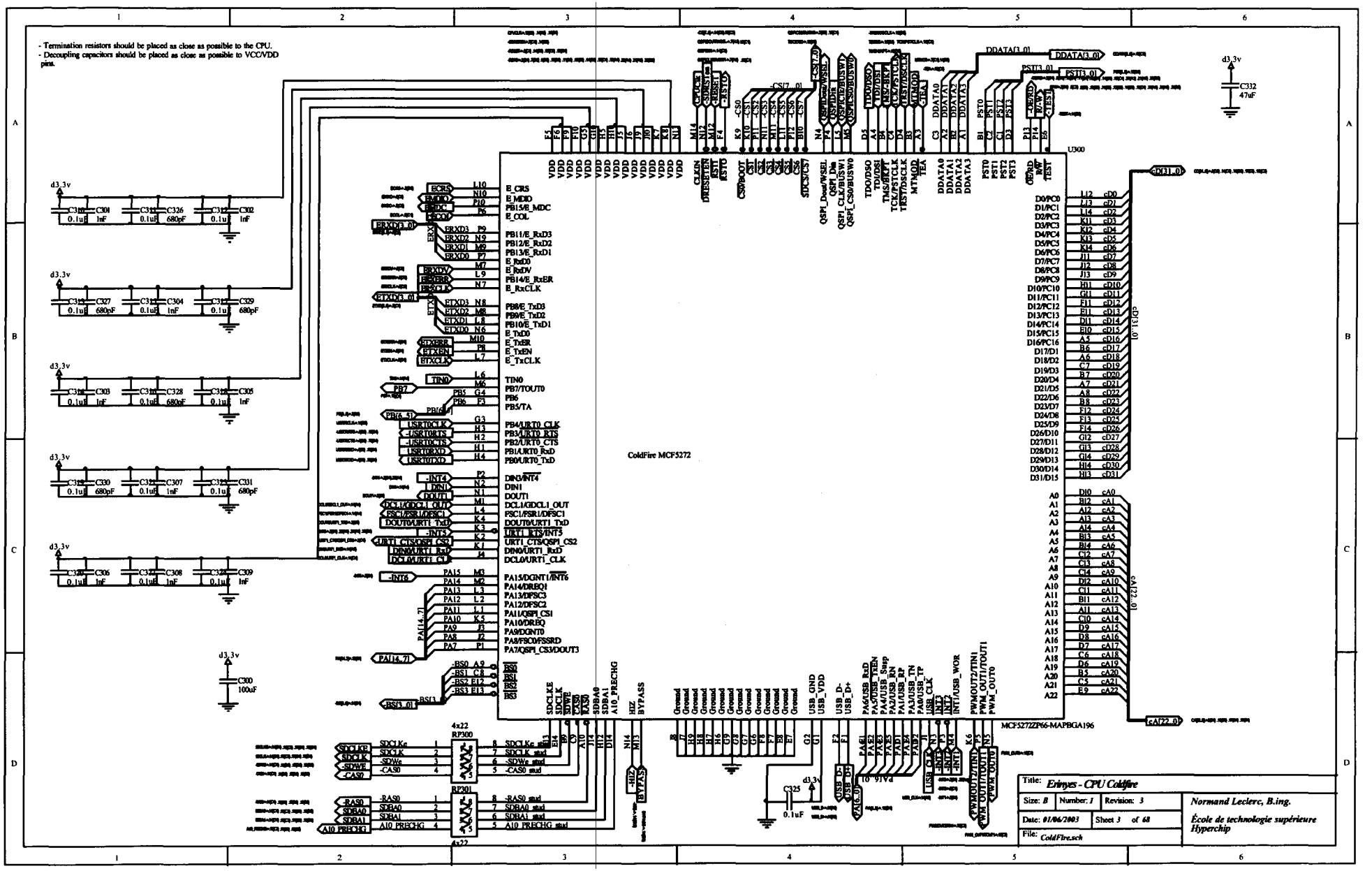


ANNEXE 5

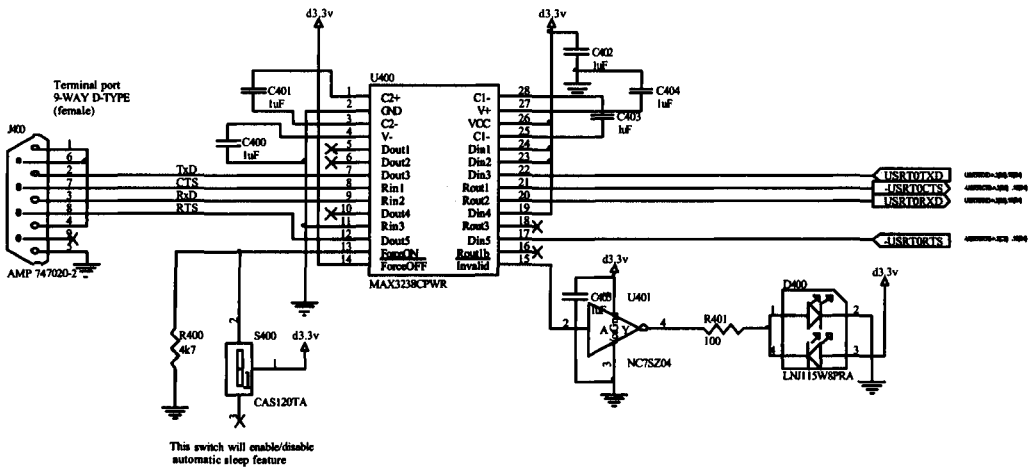
Schémas électroniques de la carte de tests


DemochipN connectors.sch
Schematic sheet 59
Power Hot swap.sch
Schematic sheet 66
-PGI4..11
Power DC-DC 0.sch
Schematic sheet 67
Power DC-DC 1.sch
Schematic sheet 68
Virtex.sch
Schematic sheet 20
CPU.sch
Schematic sheet 2
SMBData
SMBClk
-INT3
-INT4
-INT5
PWRDWN_B
DONE
RDWR_B
PROG_B
INIT_B
CS_B
FPGACPUClk
GSR
-RSTO
R/-W
-OE/RD
-CS1
BA[22..0]
BD[31..0]
B1v18_GCLKI3..0]
DemochipS connectors.sch
Schematic sheet 63
Title: Erinyes - Demochip testbed
Size: B
Number: 1
Revision: 3
Date: 01/06/2003
Sheet 1 of 68
File: Erinyes.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

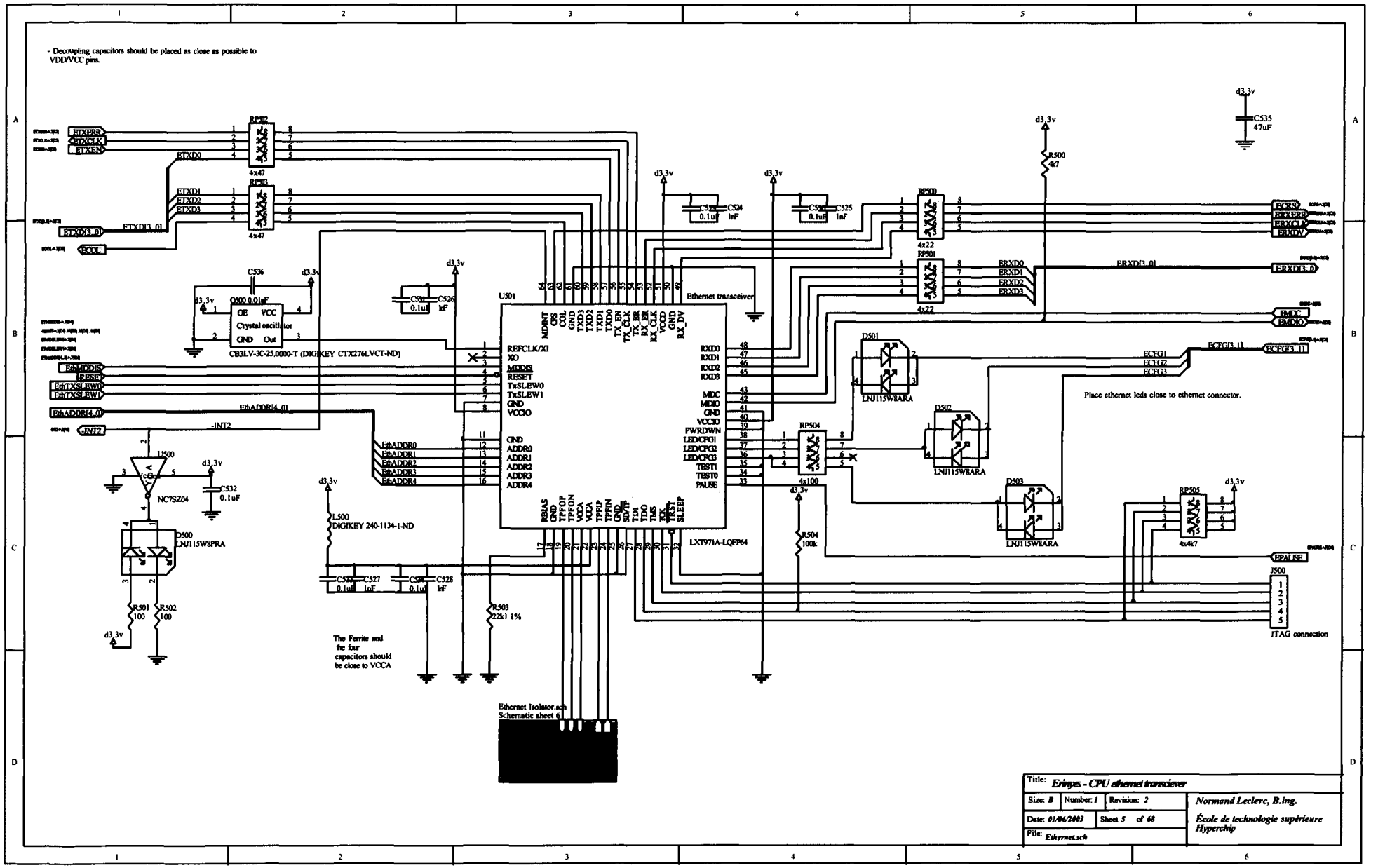

SDRAM.sch
Schematic sheet 16
CPUTestpoints.sch
Schematic sheet 14
ColdFire.sch
Schematic sheet 3
CPU Connectors.sch
Schematic sheet 15
CSLogic Clock and reset.sch
Schematic sheet 13
FPGA main CPU clock
FPGACLOCK
-Master RESET
Bus drivers.sch
Schematic sheet 8
Flash.sch
Schematic sheet 9
UART.sch
Schematic sheet 4
Ethernet.sch
Schematic sheet 5
CPUSwitches.sch
Schematic sheet 7
CPUled.sch
Schematic sheet 19
SDCLK
SDCLKe
SDBA0
SDBA1
-CAS0
-RAS0
-BS3
-BS2
-BS1
-BS0
-BS[3..0]
SDWE
A10 PRECHG
-CS7
-RESET
R/-W
-OE/RD
CPUCK
-INT[6..1]
-CS7_0
cD[31..0]
cA[22..0]
DDATA[3..0]
BST[3..0]
AuxClk[3..0]
-CS3
-RSTO
-INT3
-INT4
-INT5
-INT6
PB[6..5]
PA[14..7]
PA[6..0]
-GCS
-RESET
PROG_B
INIT_B
DONE
GSR
PWRDWN_B
RDWR_B
BUSY
SMBDat
CS_B
PA14
PA13
PA12
PA11
PA10
PA9
PA8
PB5
PB6
-CS0
-CS1
-CS2
OSPICLK/BUSW1
OSPICS0/BUSW0
OSPIDout/WSEL
-INT2
DRXD[3..0]
ETXD[3..0]
Title: Erinyes - CPU
Size: B
Number: 1
Revision: 3
Date: 01/06/2003
Sheet 2 of 68
File: CPU.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

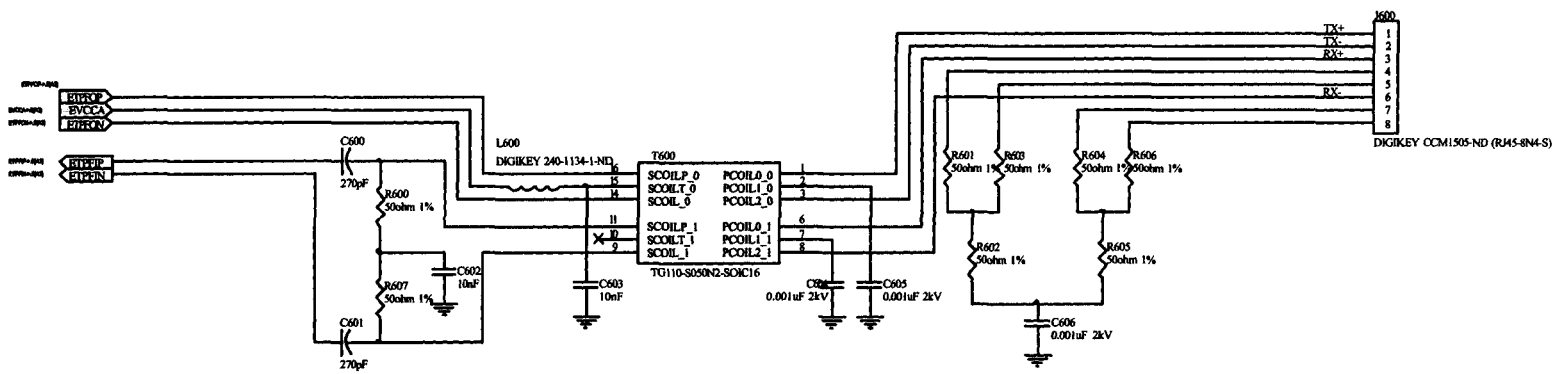

- Termination resistors should be placed as close as possible to the CPU.
- Decoupling capacitors should be placed as close as possible to VCC/VDD pins.
ColdFire MCF5272
MCF5272ZP66-MAPBGA196
U300
RP300
RP301
4x22
Title: Entrées - CPU Coldfire
Size: B
Number: 1
Revision: 3
Date: 01/06/2003
Sheet 3 of 68
File: ColdFire.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

- Decoupling capacitors should be placed as close as possible of VCC.
- All capacitors should be placed as close as possible to chip pins.
- All capacitors should be tantalum.



Title: *Eringes - CPU UART*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *4* of *68*		*École de technologie supérieure*
File: *UART.sch*			*Hyperchip*

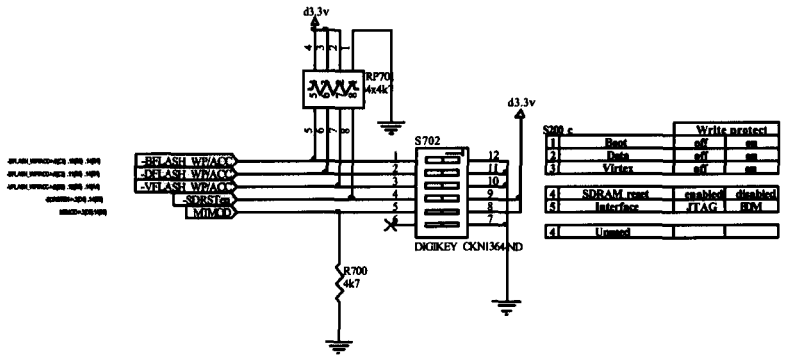

- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
ETXERR
ETXCLK
ETXEN
ETXD0
ETXD1
ETXD2
ETXD3
ETXD[3..0]
ECOL
RP502
RP503
4x47
C536
Crystal oscillator
CB3LV-3C-25.0000-T (DIGIKEY CTX276LVCT-ND)
U501
Ethernet transceiver
REFCLK/XI
XO
MDDIS
RESET
TxSLEW0
TxSLEW1
GND
VCCIO
EthMDDIS
-RESET
EthTXSLEW0
EthTXSLEW1
EthADDR[4..0]
-INT2
EthADDR0
EthADDR1
EthADDR2
EthADDR3
EthADDR4
ADDR0
ADDR1
ADDR2
ADDR3
ADDR4
MDINT
CRS
COL
TXD3
TXD2
TXD1
TXD0
TX_EN
TX_ER
RX_ER
RX_CLK
RX_DV
RXD0
RXD1
RXD2
RXD3
MDC
MDIO
PWRDWN
LEDCFG1
LEDCFG2
LEDCFG3
TEST1
TEST0
PAUSE
RBIAS
TPFOP
TPFON
VCCA
TPFIP
TPFIN
TDI
TDO
TMS
TCK
TRST
SLEEP
LXT971A-LQFP64
U500
NC7SZ04
D500
LNJ115W8PRA
L500
DIGIKEY 240-1134-1-ND
R501
100
R502
100
R503
22k1 1%
R504
100k
R500
4k7
RP504
4x100
RP500
RP501
4x22
D501
D502
D503
LNJ115W8ARA
ERXD0
ERXD1
ERXD2
ERXD3
ERXD[3..0]
ECRS
ERXERR
ERXCLK
ERXDV
RMDC
RMDIO
ECFG1
ECFG2
ECFG3
ECFG[3..1]
Place ethernet leds close to ethernet connector.
RP505
4x4k7
EPAUSE
J500
JTAG connection
C522
C527
C528
C529
C530
C531
C532
C533
C534
C535
C524
C525
C526
0.1uF
1nF
47uF
The Ferrite and the four capacitors should be close to VCCA
Ethernet Isolator.sch
Schematic sheet 6
Title: Eringes - CPU ethernet transciever
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 5 of 68
File: Ethernet.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip



Title: *Erinyes - CPU ethernet transciever isolation and connector*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.Ing.*
Date: *01/06/2003*	Sheet *6* of *60*		*École de technologie supérieure Hyperchip*
File: *Ethernet Isolator.sch*			

- Every table should be transfered to PCB near corresponding switch block.

CPU

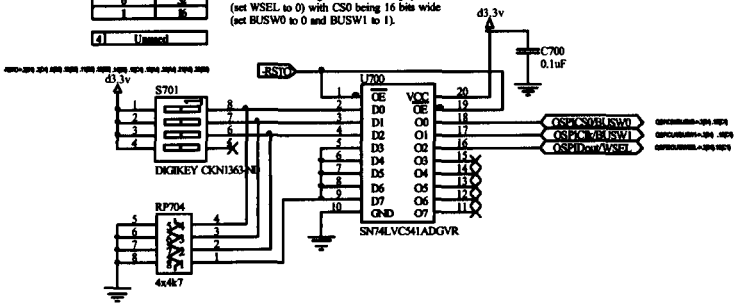


S700 c		
		Write protect
1	Boot	off / on
2	Data	off / on
3	Virtex	off / on
4	SDRAM reset	enabled / disabled
5	Interface	JTAG / SEM
4	Unused	

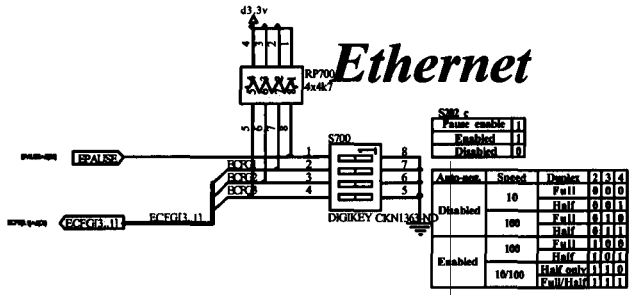
Bus width selection (S204_c)		
BUSW0 (1)	BUSW1 (2)	Size
0	0	32
0	1	16
1	0	8
1	1	invalid

WSEL (3)	Size
0	32
1	16
4	Unused

Bus should be configured as a 32bits physical (set WSEL to 0) with CS0 being 16 bits wide (set BUSW0 to 0 and BUSW1 to 1).

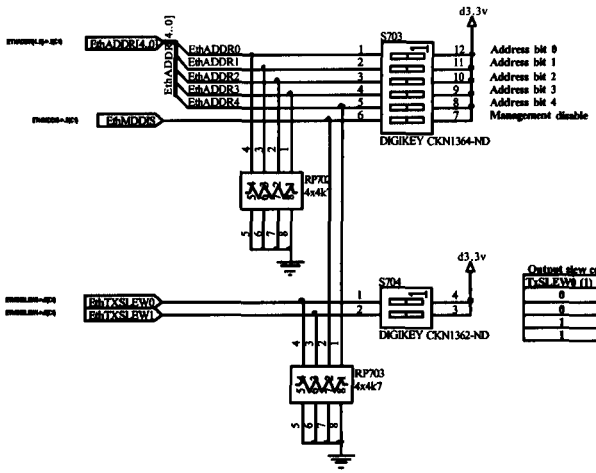


Ethernet



S702_c	
Pause enable	1
Enabled	1
Disabled	0

Auto-neg.	Speed	Duplex	2	3	4
Disabled	10	Full	0	0	0
		Half	0	0	1
	100	Full	0	1	0
		Half	0	1	1
Enabled	100	Full	1	0	0
		Half	1	0	1
	10/100	Half only	1	1	0
		Full/Half	1	1	1

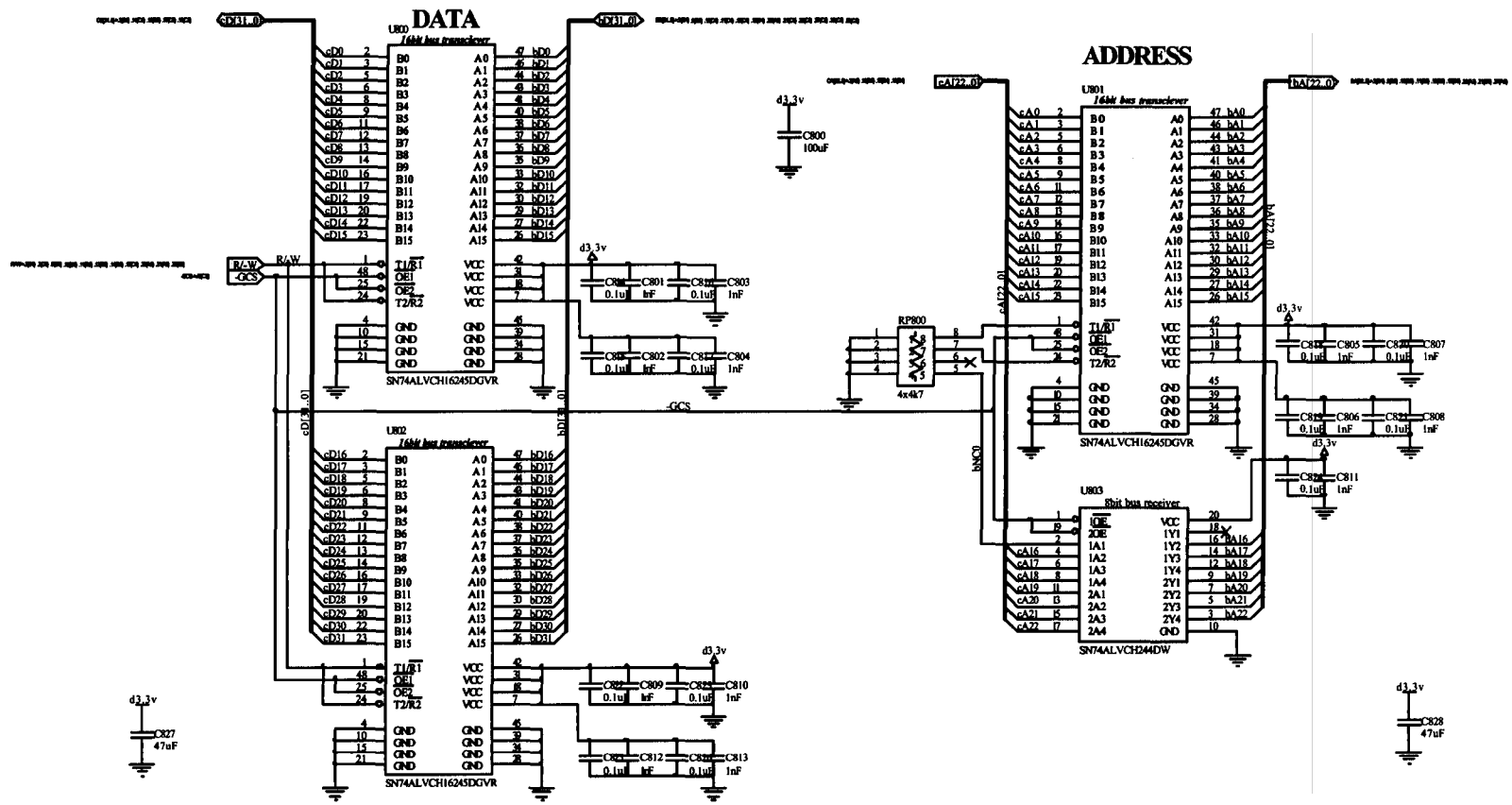


Output slew control (S203_c)		
TxSLEW0 (1)	TxSLEW1 (2)	Slew rate
0	0	2.5ns
0	1	3.1ns
1	0	3.7ns
1	1	4.3ns

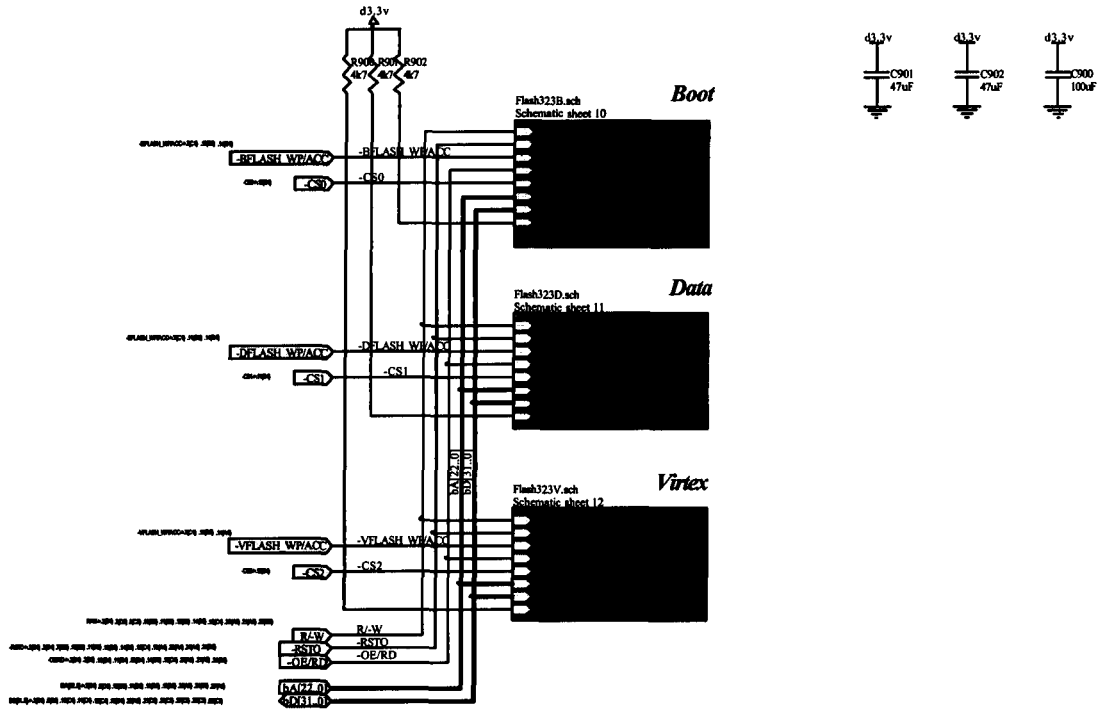
Title: *Eringes - CPU configuration switches*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.Ing.*
Date: *01/06/2003*	Sheet *7* of *68*		*École de technologie supérieure Hyperchip*
File: *CPUSwitches.sch*			

- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- Place one 47uF close to a pair of buffers

- Place each 1uF somewhere between bus transceivers and CPU

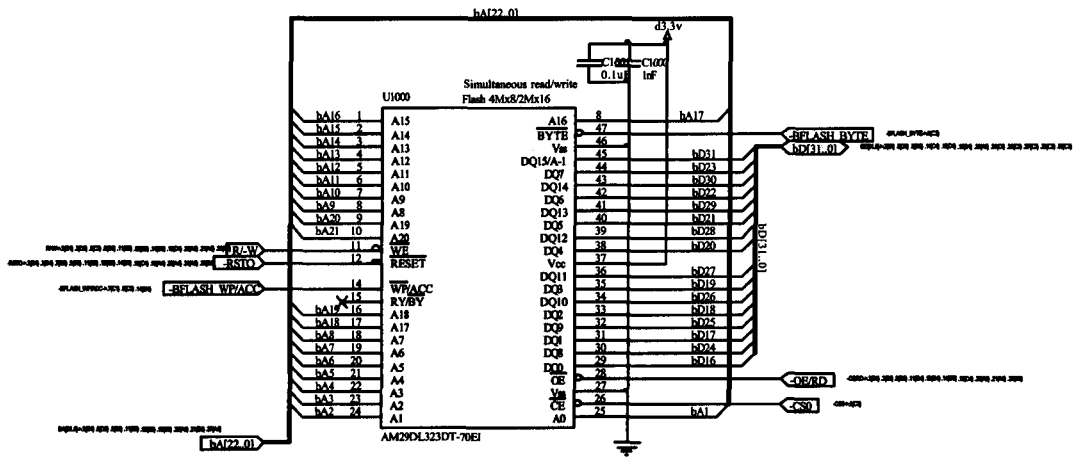


Title: *Erinyes - CPU data/address buffers*			
Size: *B*	Number: *1*	Revision: *3*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *8* of *68*		*École de technologie supérieure Hyperchip*
File: *Bus drivers.sch*			



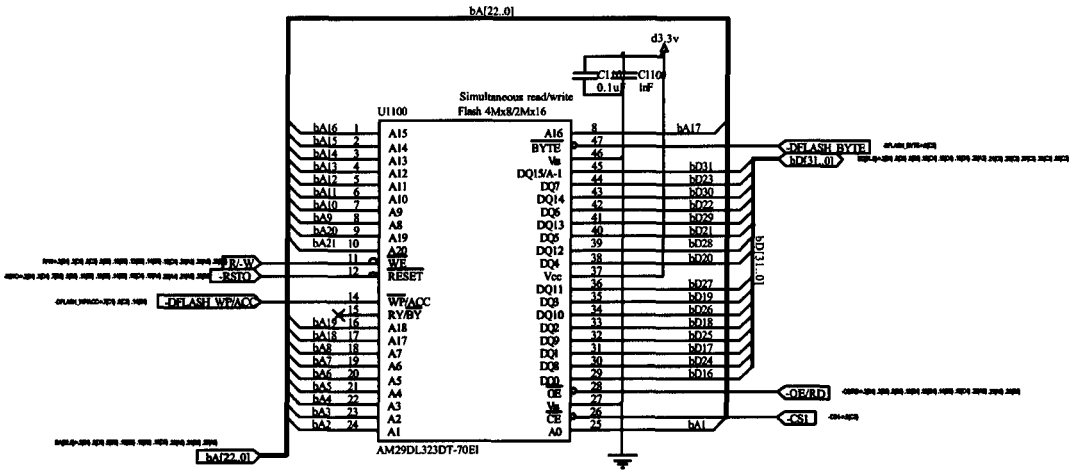
Title: *Eringes - CPU flashes*			
Size: *B*	Number: *1*	Revision: *3*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *9* of *68*		*École de technologie supérieure Hyperchip*
File: *Flash.sch*			

- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- Termination resistors should be placed as close as possible to the chip.



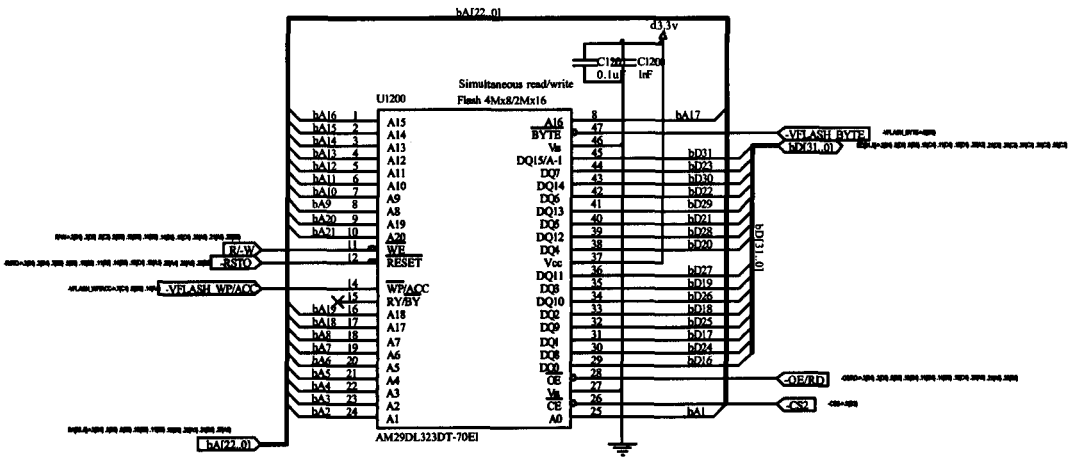
Title: *Erinyes - CPU boot flash*			
Size: *B*	Number: *1*	Revision: *3*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *10* of *68*		*École de technologie supérieure Hyperchip*
File: *Flash323B.sch*			

- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- Termination resistors should be placed as close as possible to the



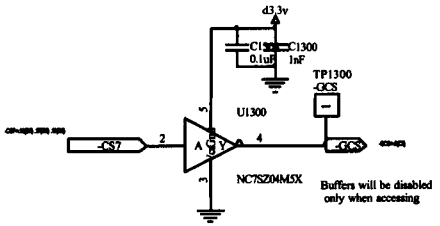
Title: *Erinyes - CPU data flash*			
Size: *B*	Number: *1*	Revision: *3*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *11* of *68*		*École de technologie supérieure Hyperchip*
File: *Flash323D.sch*			

- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- Termination resistors should be placed as close as possible to the

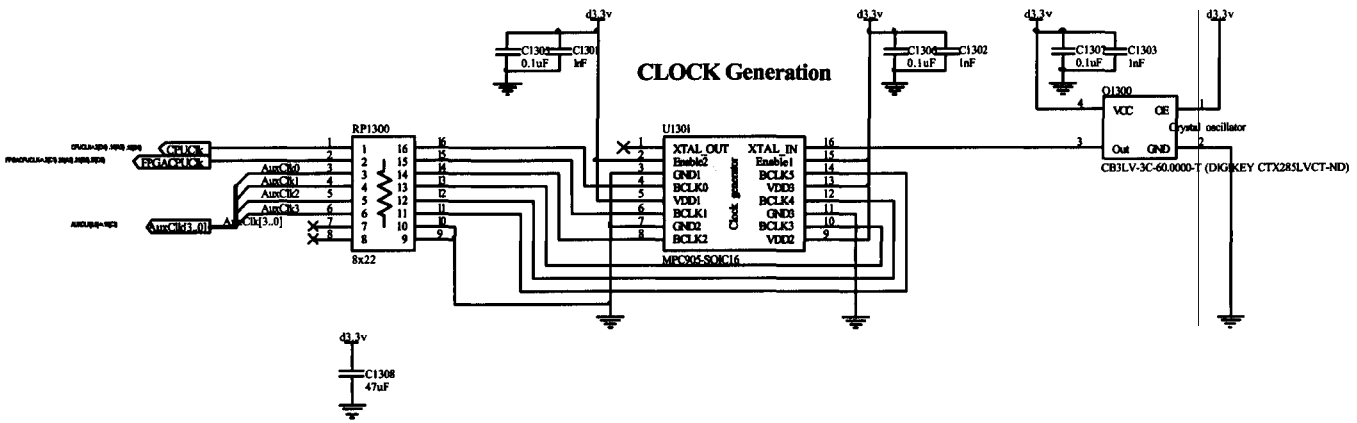


Title: *Erinyes - CPU virtex firmware flash*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *12* of *68*		*École de technologie supérieure Hyperchip*
File: *Flash323V.sch*			

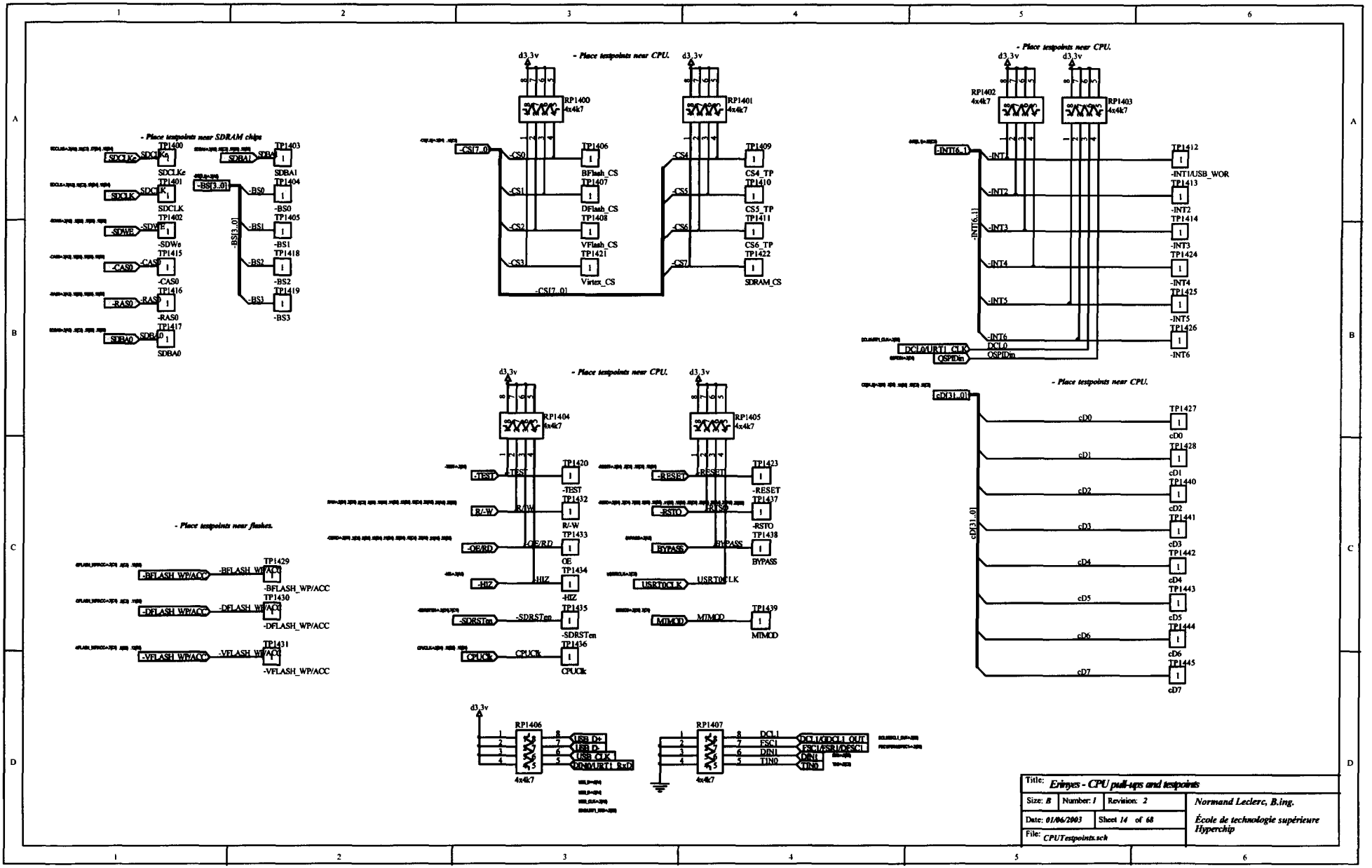
Chip selects logic

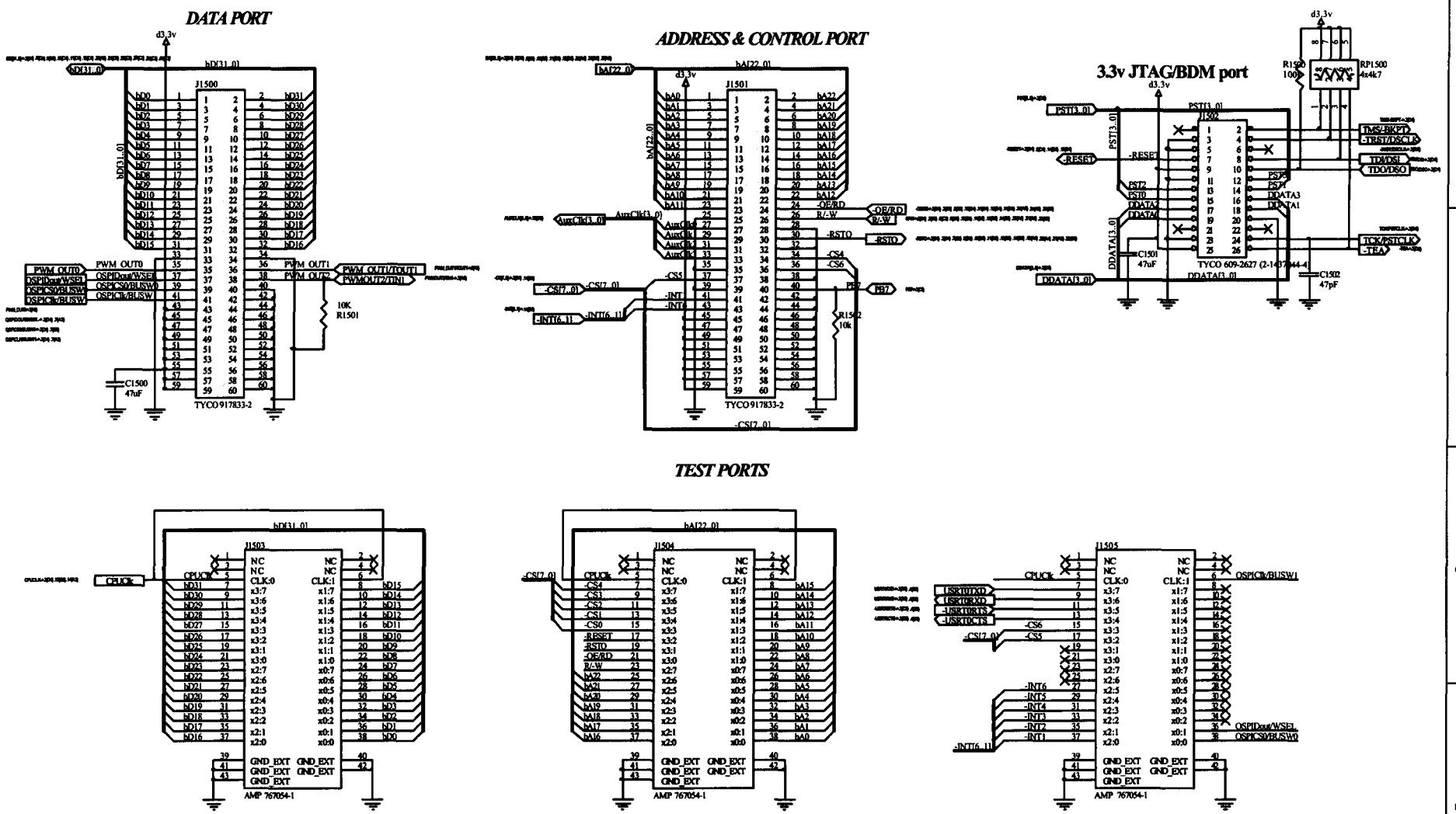


CLOCK Generation

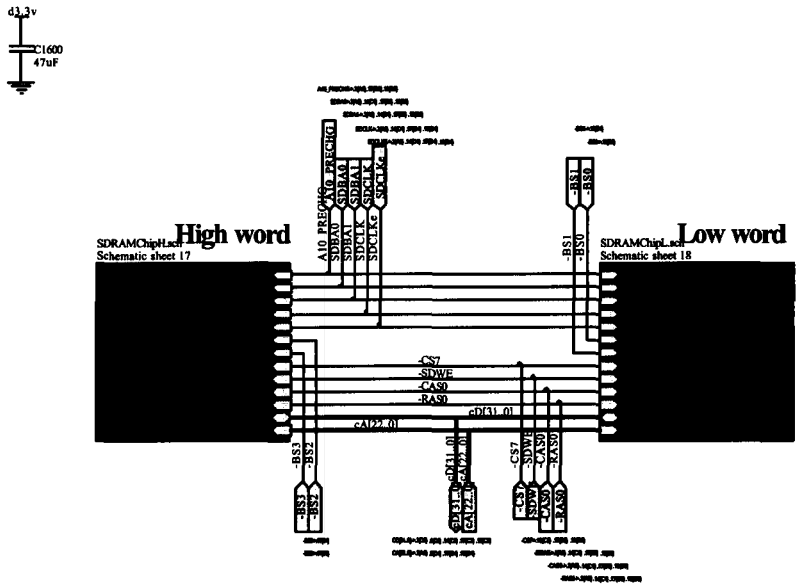


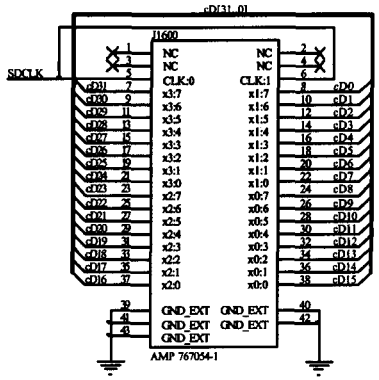
Title: *Eringes - CPU CS logic, reset and clock generation*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.Ing.*
Date: *01/06/2003*	Sheet *13* of *68*		*École de technologie supérieure Hyperchip*
File: *CSLogic Clock and reset.sch*			

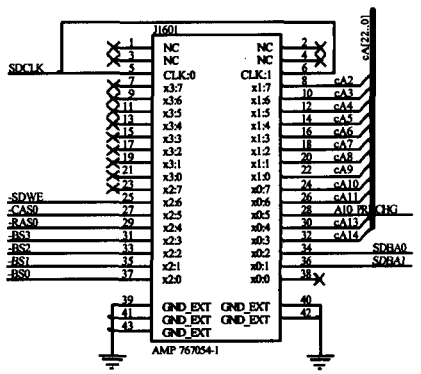

- Place testpoints near SDRAM chips
SDCLKe
SDCLK
-SDWe
-CAS0
-RAS0
SDBA0
-BS[3..0]
SDBA1
-BS0
-BS1
-BS2
-BS3
d3,3v
- Place testpoints near CPU.
RP1400 4x4k7
RP1401 4x4k7
-CS[7..0]
BFlash_CS
DFlash_CS
VFlash_CS
Virtex_CS
CS4_TP
CS5_TP
CS6_TP
SDRAM_CS
RP1402 4x4k7
RP1403 4x4k7
-INT[6..1]
-INT1/USB_WOR
-INT2
-INT3
-INT4
-INT5
-INT6
DCL0/URT1_CLK
OSPIDin
- Place testpoints near flashes.
-BFLASH_WP/ACC
-DFLASH_WP/ACC
-VFLASH_WP/ACC
RP1404 4x4k7
RP1405 4x4k7
-TEST
R/-W
-OE/RD
-HIZ
-SDRSTen
CPUClk
-RESET
-RSTO
BYPASS
USRT0CLK
MTMOD
cD[31..0]
cD0
cD1
cD2
cD3
cD4
cD5
cD6
cD7
RP1406 4x4k7
USB_D+
USB_D-
USB_CLK
DIN0/URT1_RxD
RP1407 4x4k7
DCL1
FSC1
DIN1
TIN0
DCL1/ODCL1_OUT
FSC1/ESSRI/DFSC1
Title: Erinyes - CPU pull-ups and testpoints
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 14 of 68
File: CPUTestpoints.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip


DATA PORT
ADDRESS & CONTROL PORT
3.3v JTAG/BDM port
TEST PORTS
J1500
J1501
J1502
J1503
J1504
J1505
TYCO917833-2
AMP 767054-1
TYCO 609-2627 (2-1437644-4)
Title: Eriyes - CPU connectors
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 15 of 68
File: CPU Connectors.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- Place one 47uF close to SDRAMs

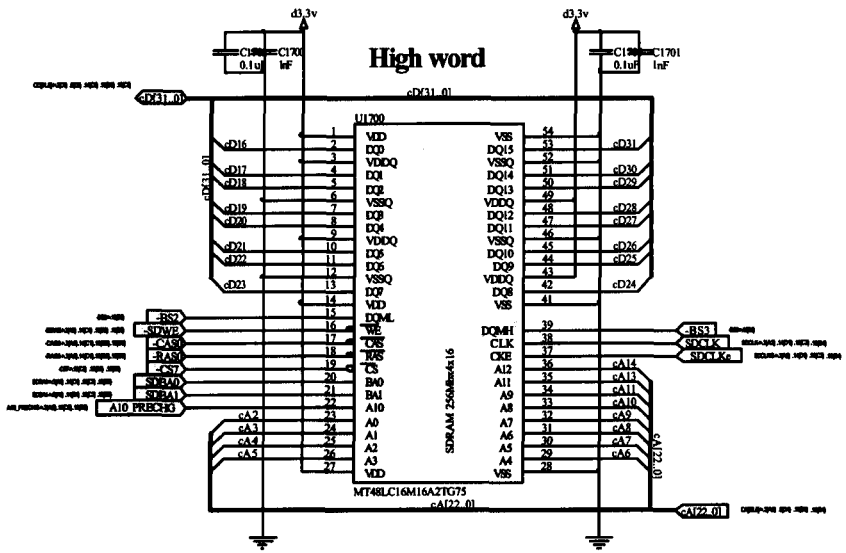






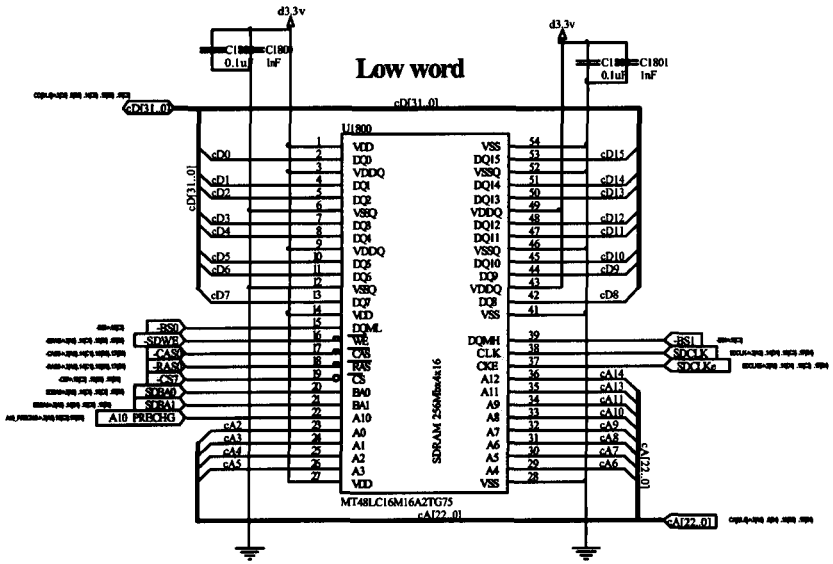
Title: *Eriyes - CPU SDRAM*			
Size: *B*	Number: *1*	Revision: *3*	*Normand Leclerc, B.Ing.*
Date: *01/06/2003*	Sheet *16* of *68*		*École de technologie supérieure Hyperchip*
File: *SDRAM.sch*			

- As the ColdFire accepts busses of 16 or 32 bits, one or two SDRAM chips can be soldered but 2 is prefered. This configuration gives 64MB of available RAM with no possible expansion.
- Decoupling capacitors should be placed as close as possible to VDDQ pins.

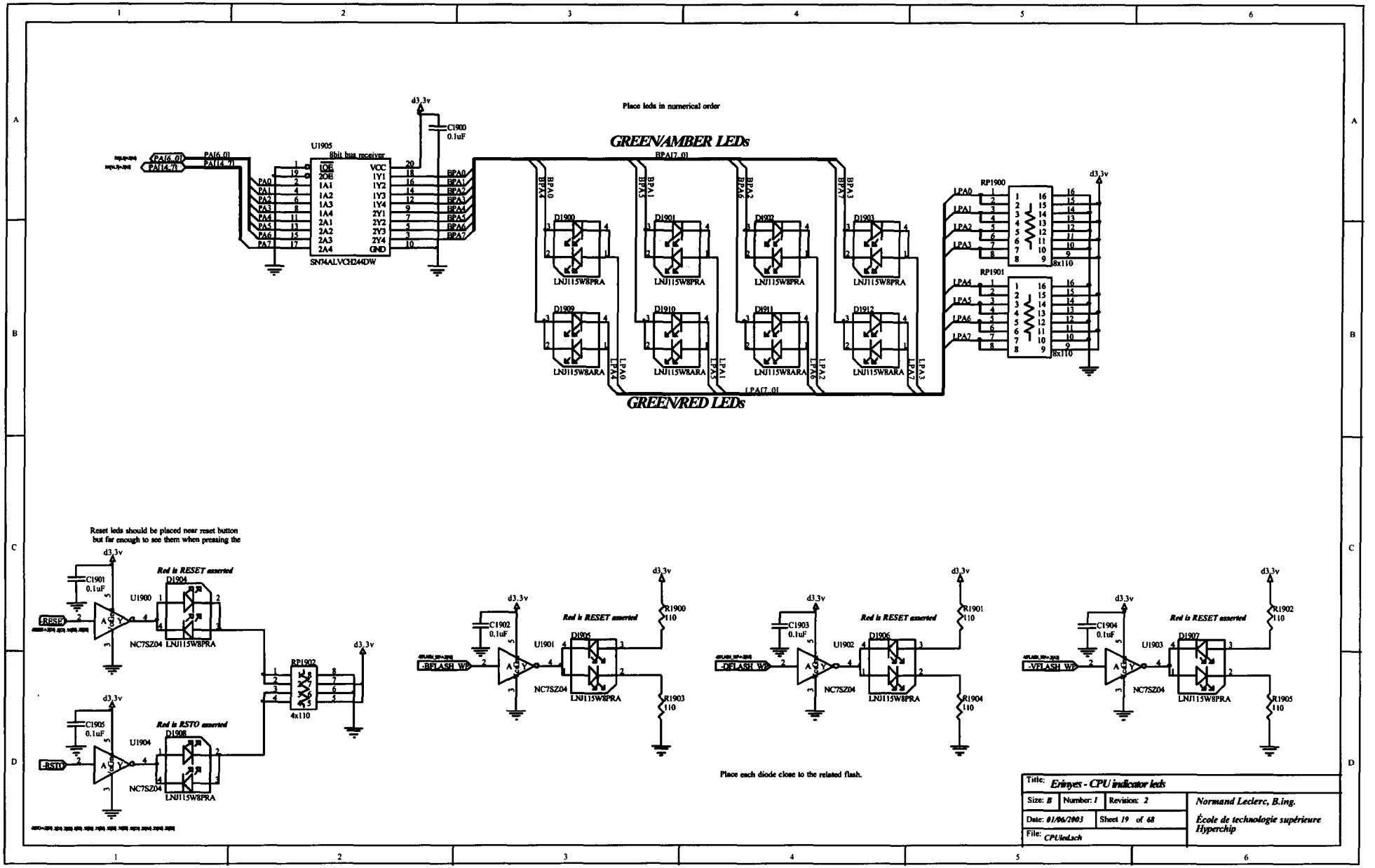


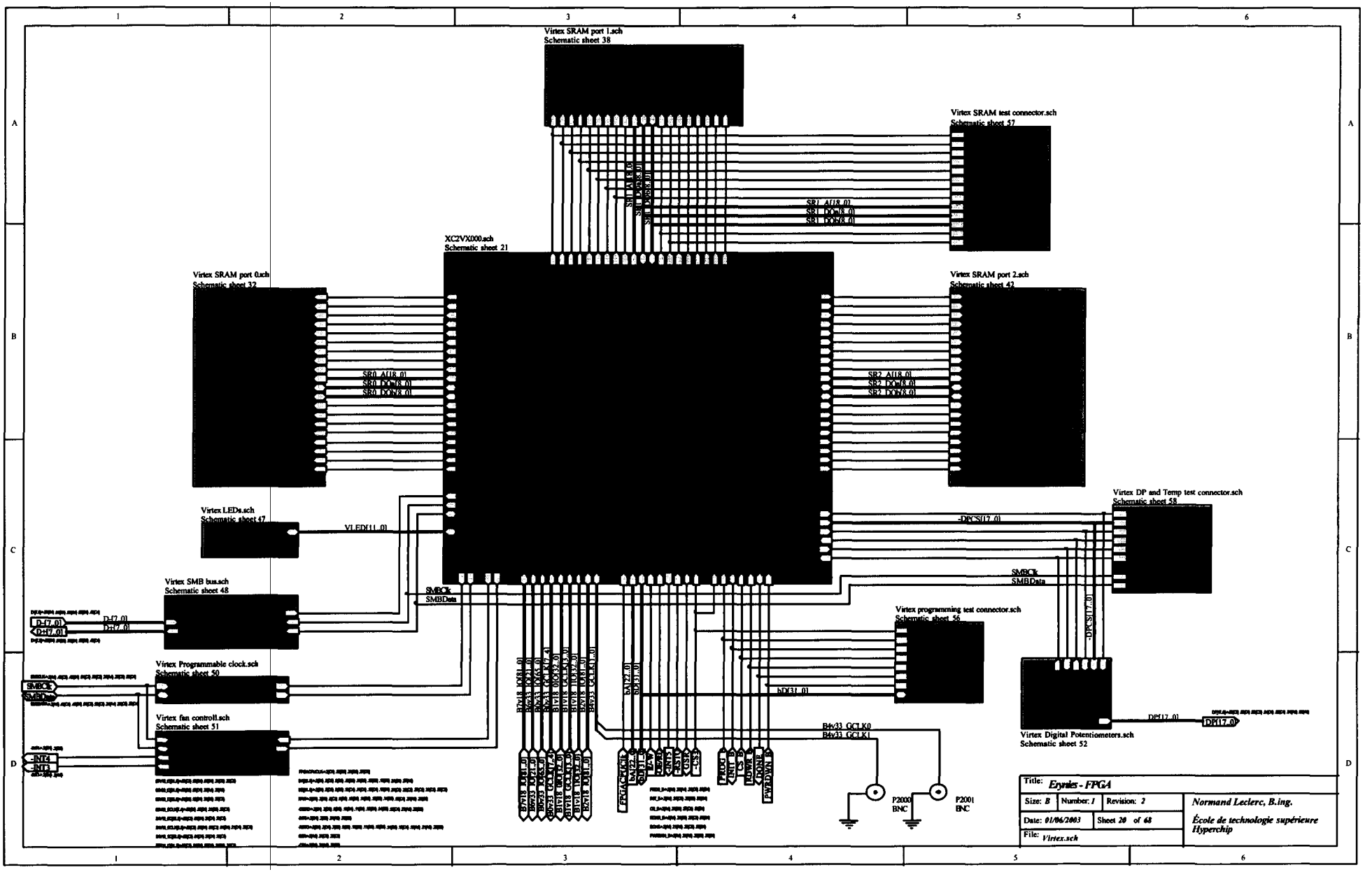
Title: *Eriroyes - CPU high word SDRAM*			
Size: *B*	Number: *1*	Revision: *3*	*Normand Leclerc, B.Ing.*
Date: *01/06/2003*	Sheet *17* of *68*		*École de technologie supérieure Hyperchip*
File: *SDRAMChip1.sch*			

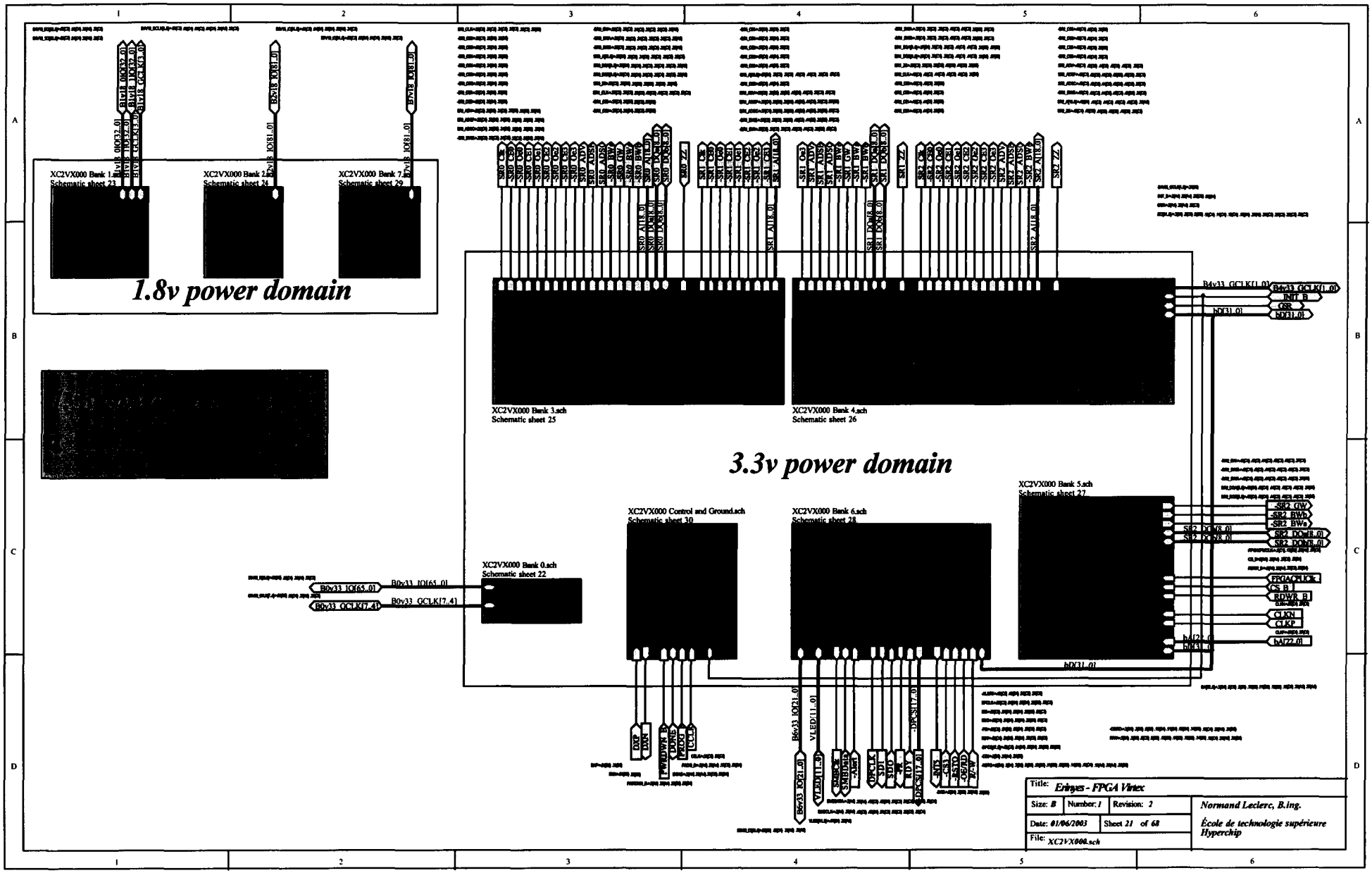
- As the ColdFire accepts busses of 16 or 32 bits, one or two SDRAM chips can be soldered but 2 is prefered. This configuration gives 64MB of available RAM with no possible expansion.
- Decoupling capacitors should be placed as close as possible to VDDQ pins.

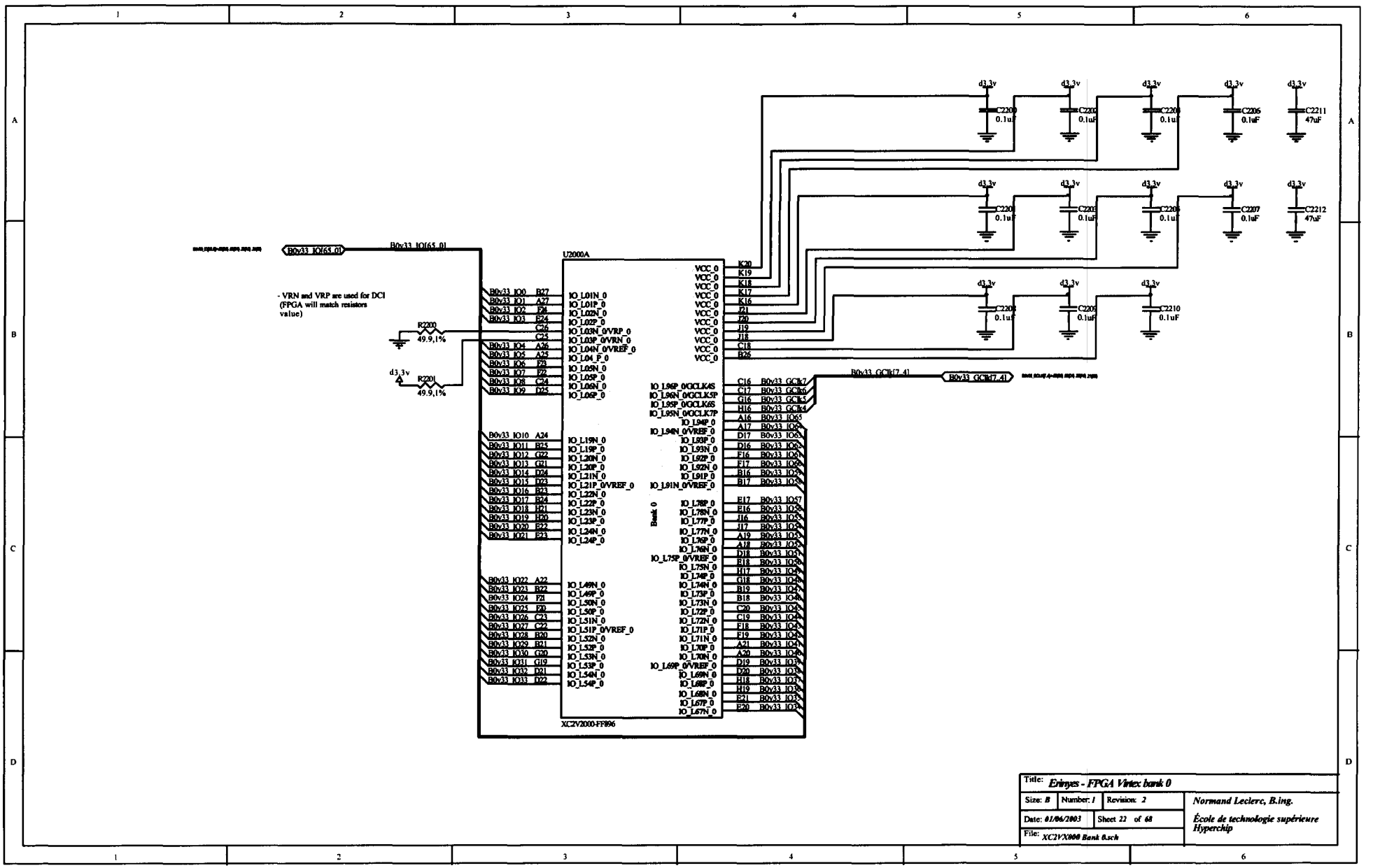


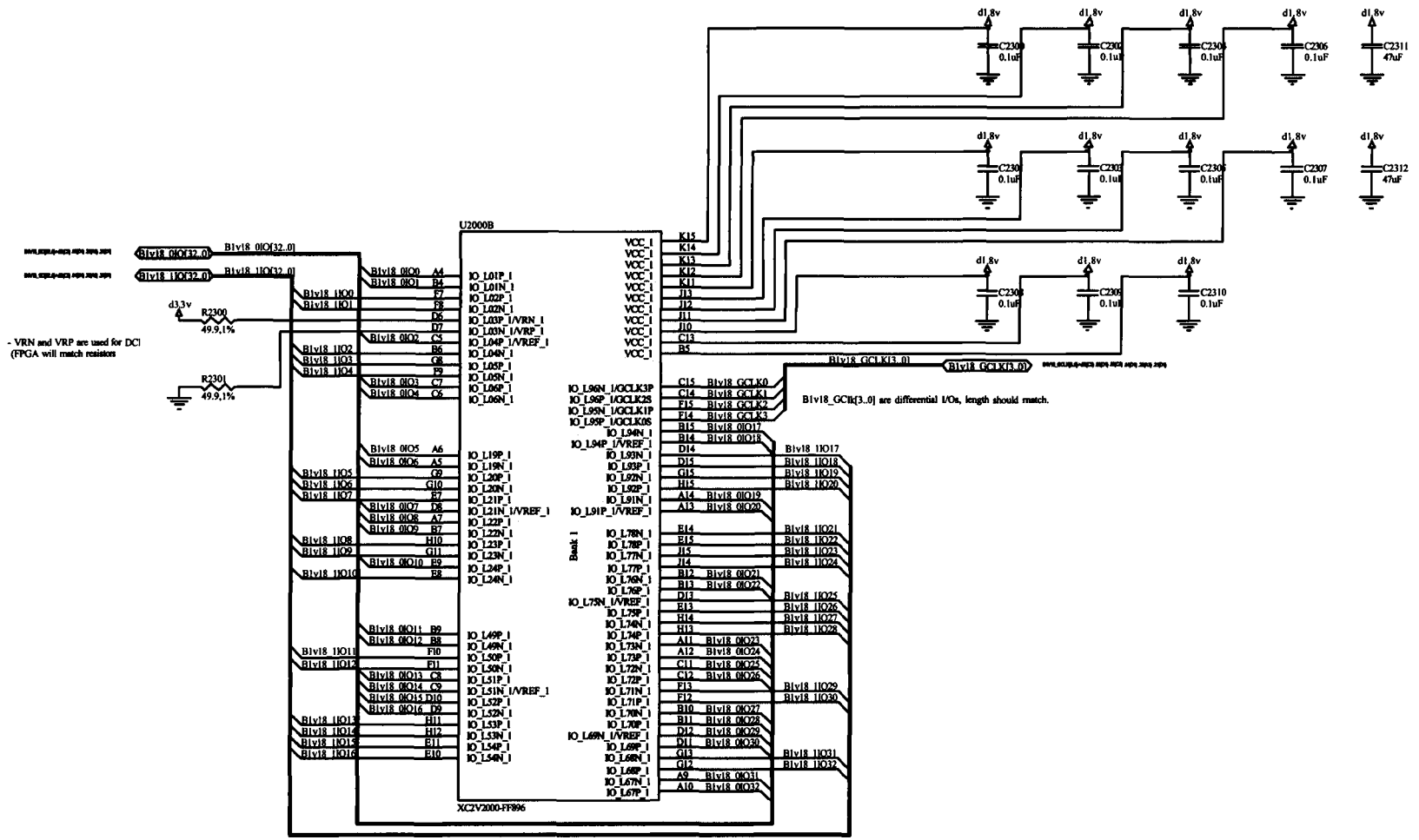
Title: *Eriyes - CPU low word SDRAM*			
Size: *B*	Number: *1*	Revision: *3*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *18* of *68*		*École de technologie supérieure Hyperchip*
File: *SDRAMChipL.sch*			

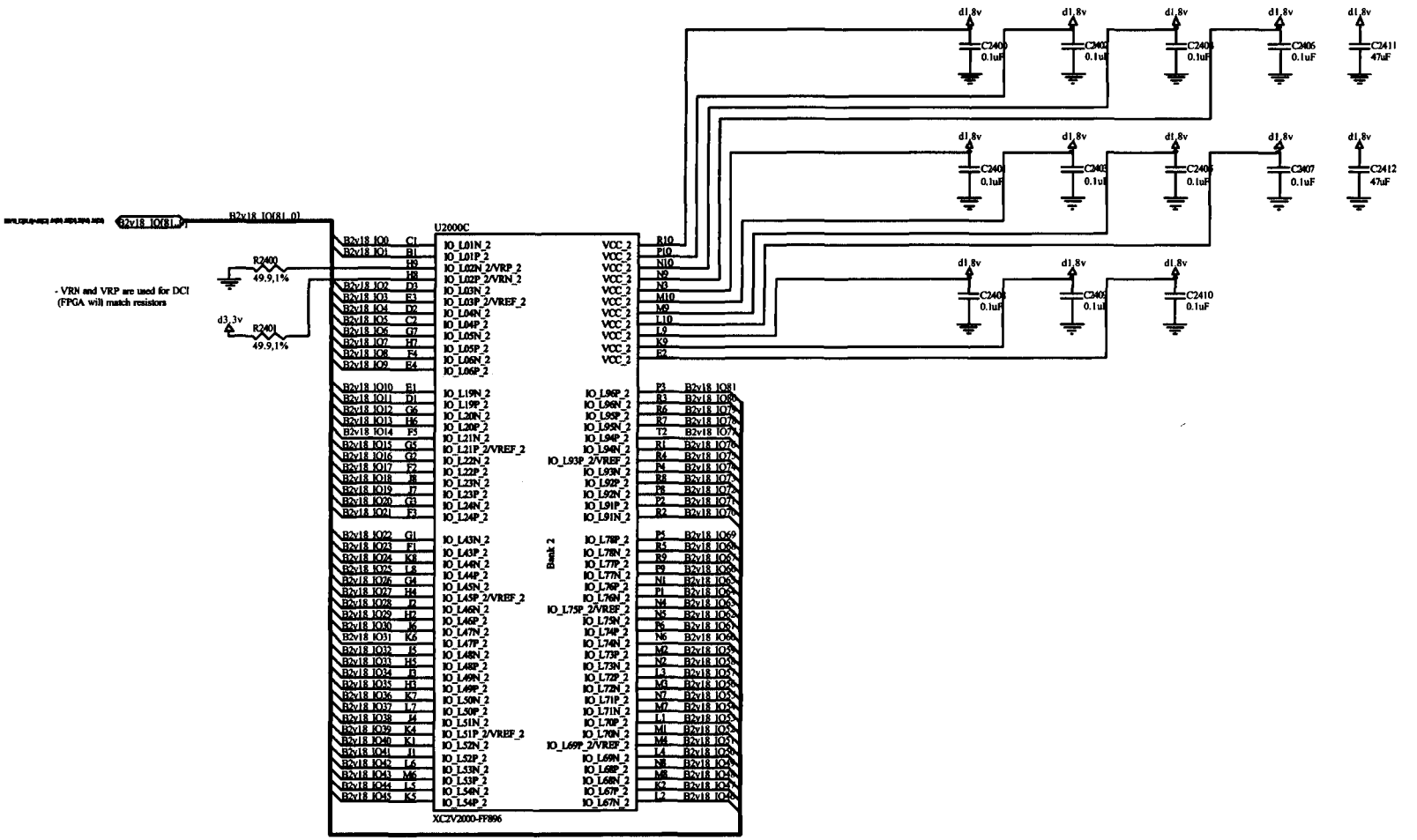

Place leds in numerical order
GREEN/AMBER LEDs
GREEN/RED LEDs
U1905
8bit bus receiver
SN74ALVCH24DW
C1900
0.1uF
d3,3v
RP1900
RP1901
8x110
D1900
D1901
D1902
D1903
D1909
D1910
D1911
D1912
LNJ115W8PRA
LNJ115W8ARA
Reset leds should be placed near reset button
but far enough to see them when pressing the
Red is RESET asserted
Red is RSTO asserted
RESET
-RSTO
NC7SZ04
RP1902
4x110
-BFLASH_WP
-DFLASH_WP
-VFLASH_WP
R1900
R1901
R1902
R1903
R1904
R1905
110
Place each diode close to the related flash.
Title: Erinyes - CPU indicator leds
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 19 of 68
File: CPUled.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

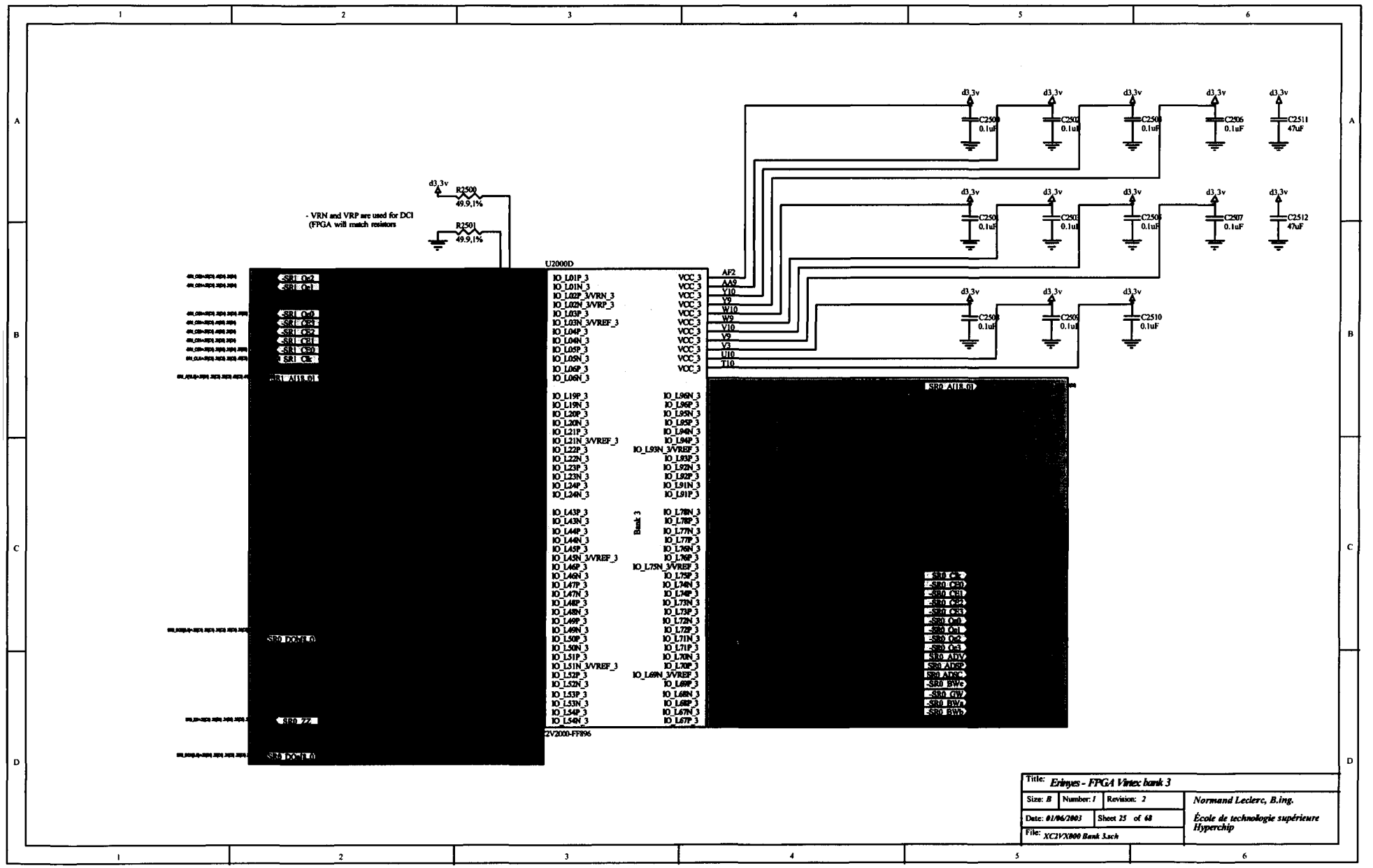

Virtex SRAM port 1.sch
Schematic sheet 38
Virtex SRAM test connector.sch
Schematic sheet 57
SR1_A[18..0]
SR1_DO[8..0]
SR1_DO[8..0]
XC2VX000.sch
Schematic sheet 21
Virtex SRAM port 0.sch
Schematic sheet 32
Virtex SRAM port 2.sch
Schematic sheet 42
SR0_A[18..0]
SR0_DO[8..0]
SR0_DO[8..0]
SR2_A[18..0]
SR2_DO[8..0]
SR2_DO[8..0]
Virtex LEDs.sch
Schematic sheet 47
VLED[11..0]
Virtex DP and Temp test connector.sch
Schematic sheet 58
-DPCS[17..0]
SMBClk
SMBData
Virtex SMB bus.sch
Schematic sheet 48
D-I7..0]
D+I7..0]
Virtex programming test connector.sch
Schematic sheet 56
bDO31..0]
-DPCS[17..0]
Virtex Programmable clock.sch
Schematic sheet 50
SMBClk
SMBData
Virtex fan controll.sch
Schematic sheet 51
-INT4
-INT3
B4v33_GCLK0
B4v33_GCLK1
P2000
BNC
P2001
BNC
DPI17..0]
DPI17..0
Virtex Digital Potentiometers.sch
Schematic sheet 52
Title: Eeyrles - FPGA
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 20 of 68
File: Virtex.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

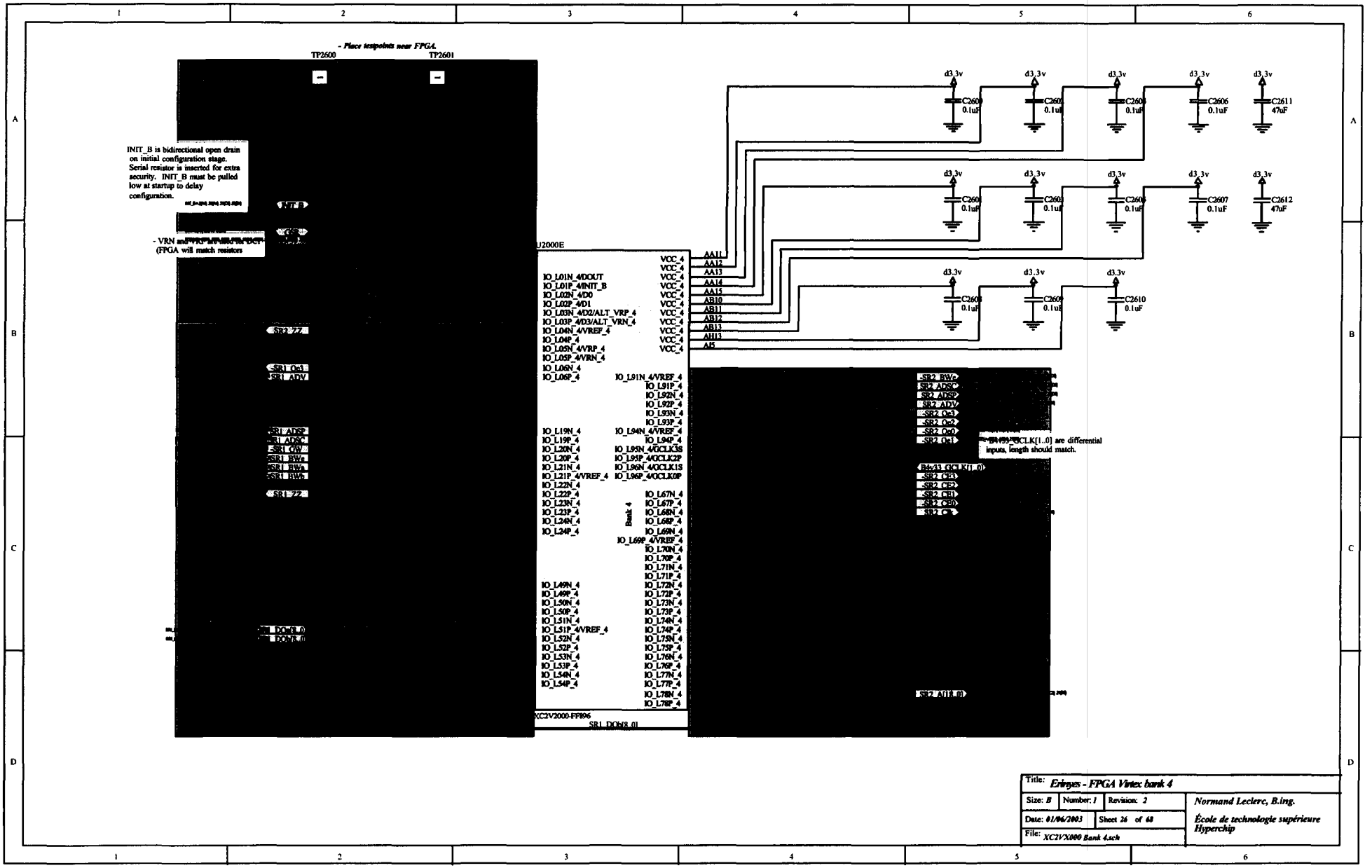

1.8v power domain
3.3v power domain
XC2VX000 Bank 1.sch
Schematic sheet 23
XC2VX000 Bank 2.sch
Schematic sheet 24
XC2VX000 Bank 7.sch
Schematic sheet 29
XC2VX000 Bank 3.sch
Schematic sheet 25
XC2VX000 Bank 4.sch
Schematic sheet 26
XC2VX000 Bank 5.sch
Schematic sheet 27
XC2VX000 Bank 6.sch
Schematic sheet 28
XC2VX000 Control and Ground.sch
Schematic sheet 30
XC2VX000 Bank 0.sch
Schematic sheet 22
Title: Eriyes - FPGA Virtex
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 21 of 68
File: XC2VX000.sch
Normand Leclerc, B.Ing.
École de technologie supérieure
Hyperchip

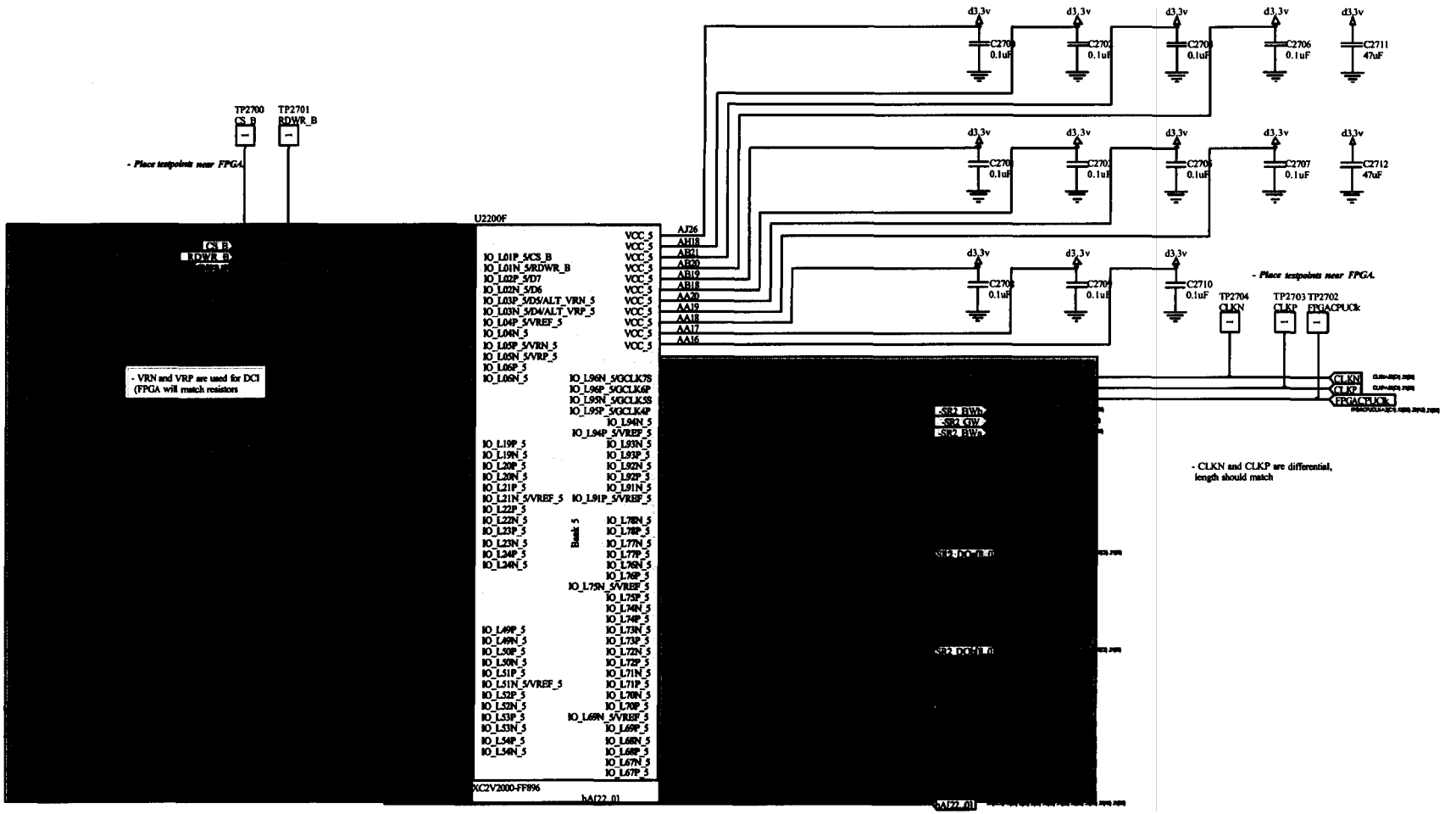

B0v33_IO165_01
B0v33_IO165_01
- VRN and VRP are used for DCI (FPGA will match resistors value)
R2200
49.9,1%
R2201
49.9,1%
d3.3v
U2000A
XC2V2000-FF896
Bank 0
B0v33_GCLK7_41
B0v33_GCLK7_41
C2209 0.1uF
C2208 0.1uF
C2207 0.1uF
C2206 0.1uF
C2211 47uF
C2212 47uF
C2210 0.1uF
Title: Eriryes - FPGA Virtex bank 0
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 22 of 68
File: XC2VX000 Bank 0.sch
Normand Leclerc, B.ing.
École de technologie supérieure Hyperchip


U2000B
XC2V2000-FF896
Bank 1
d1.8v
C2309 0.1uF
C2305 0.1uF
C2304 0.1uF
C2306 0.1uF
C2311 47uF
C2302 0.1uF
C2303 0.1uF
C2308 0.1uF
C2307 0.1uF
C2312 47uF
C2301 0.1uF
C2309 0.1uF
C2310 0.1uF
d3.3v
R2300 49.9,1%
R2301 49.9,1%
- VRN and VRP are used for DCI (FPGA will match resistors
Bivl8_GCLKI3..0]
B1v18_GCIk[3..0] are differential I/Os, length should match.
Title: Erinyes - FPGA Virtex bank 1
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 23 of 68
File: XC2VX000 Bank 1.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

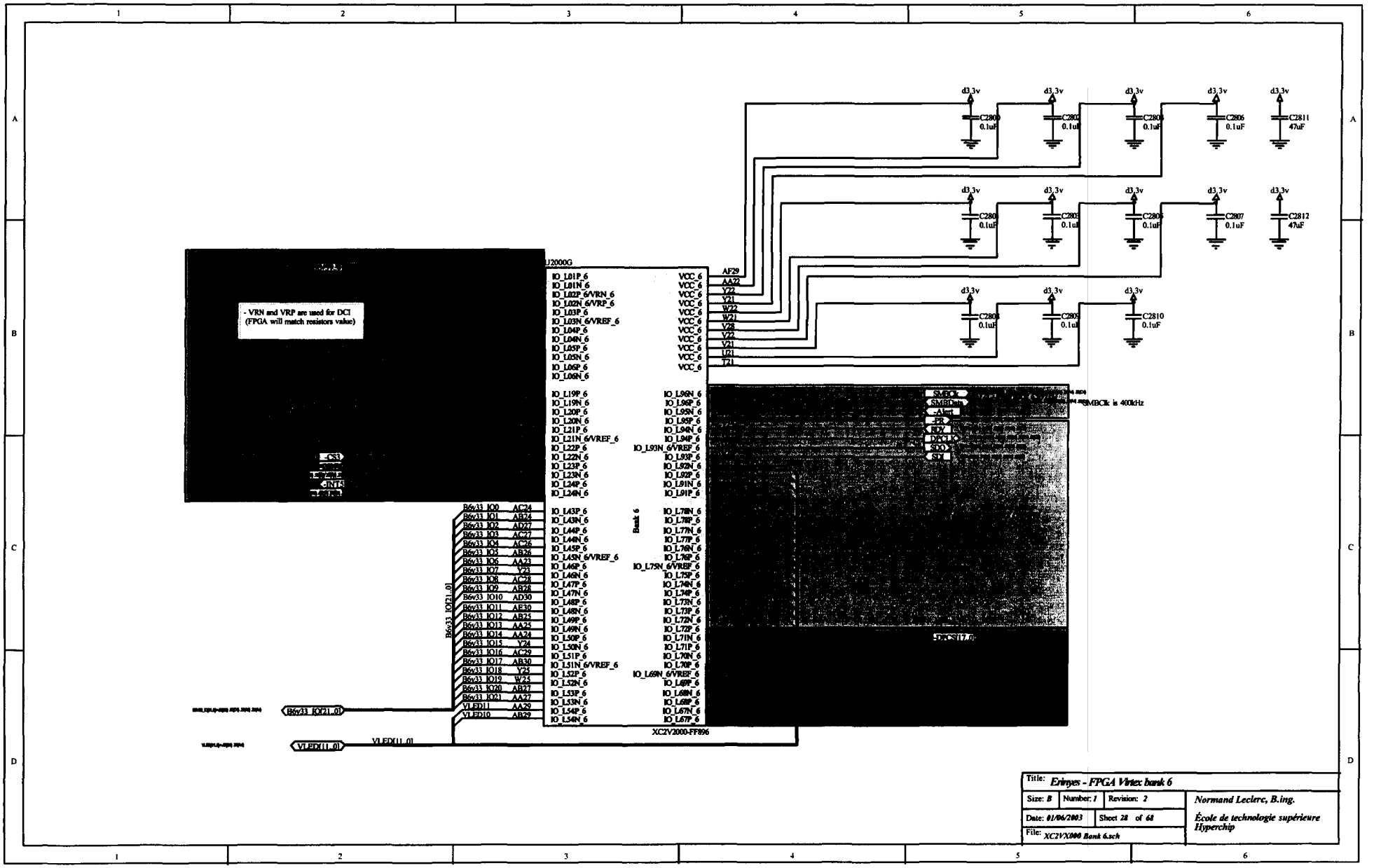

d1.8v
C2400 0.1uF
C2401 0.1uF
C2402 0.1uF
C2403 0.1uF
C2404 0.1uF
C2405 0.1uF
C2406 0.1uF
C2407 0.1uF
C2408 0.1uF
C2409 0.1uF
C2410 0.1uF
C2411 47uF
C2412 47uF
R2400 49.9,1%
R2401 49.9,1%
d3.3v
- VRN and VRP are used for DCI
(FPGA will match resistors
B2v18_IO[81_0]
U2000C
Bank 2
XC2V2000-FF896
Title: Eriyes - FPGA Virtex bank 2
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 24 of 68
File: XC2VX000 Bank 2.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

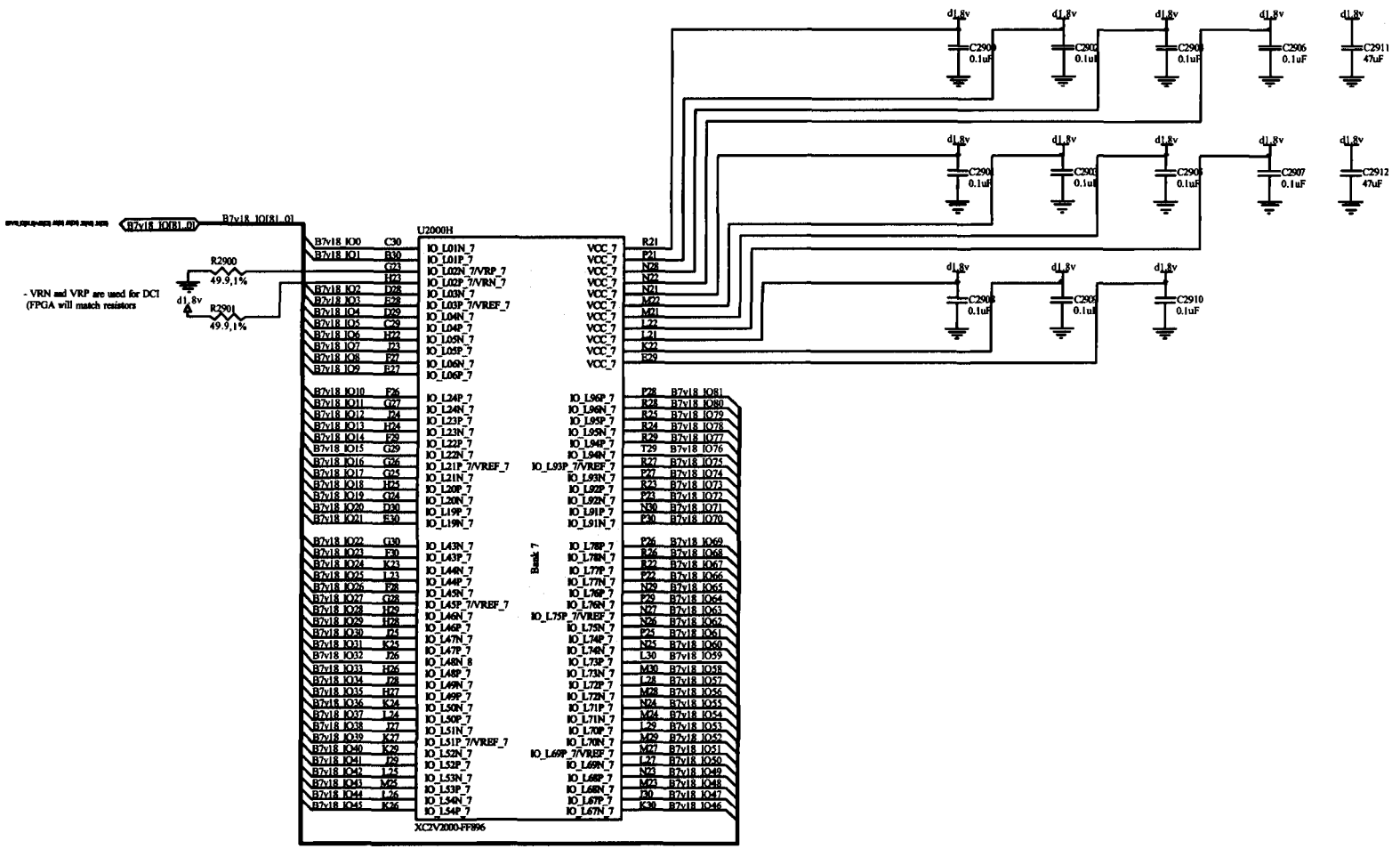

- VRN and VRP are used for DCI
(FPGA will match resistors
R2500
49.9,1%
R2501
49.9,1%
d3.3v
C2500
C2501
C2502
C2503
C2504
C2505
C2506
C2507
C2508
C2509
C2510
C2511
C2512
0.1uF
47uF
U2000D
SR1_D[2]
SR1_D[3]
SR1_D[4]
SR1_CE2
SR1_CE3
SR1_CE4
SR1_CE0
SR1_CE1
SR1_CK
SR1_A[18..0]
SR0_D[0..31]
SR0_ZZ
SR1_D[0..31]
IO_L01P_3
IO_L01N_3
IO_L02P_3/VRN_3
IO_L02N_3/VRP_3
IO_L03P_3
IO_L03N_3/VREF_3
IO_L04P_3
IO_L04N_3
IO_L05P_3
IO_L05N_3
IO_L06P_3
IO_L06N_3
IO_L19P_3
IO_L19N_3
IO_L20P_3
IO_L20N_3
IO_L21P_3
IO_L21N_3/VREF_3
IO_L22P_3
IO_L22N_3
IO_L23P_3
IO_L23N_3
IO_L24P_3
IO_L24N_3
IO_L43P_3
IO_L43N_3
IO_L44P_3
IO_L44N_3
IO_L45P_3
IO_L45N_3/VREF_3
IO_L46P_3
IO_L46N_3
IO_L47P_3
IO_L47N_3
IO_L48P_3
IO_L48N_3
IO_L49P_3
IO_L49N_3
IO_L50P_3
IO_L50N_3
IO_L51P_3
IO_L51N_3/VREF_3
IO_L52P_3
IO_L52N_3
IO_L53P_3
IO_L53N_3
IO_L54P_3
IO_L54N_3
VCC_3
AF2
AA9
V10
Y9
W10
W9
V10
V9
V3
U10
T10
Bank 3
IO_L96N_3
IO_L96P_3
IO_L95N_3
IO_L95P_3
IO_L94N_3
IO_L94P_3
IO_L93N_3/VREF_3
IO_L93P_3
IO_L92N_3
IO_L92P_3
IO_L91N_3
IO_L91P_3
IO_L78N_3
IO_L78P_3
IO_L77N_3
IO_L77P_3
IO_L76N_3
IO_L76P_3
IO_L75N_3/VREF_3
IO_L75P_3
IO_L74N_3
IO_L74P_3
IO_L73N_3
IO_L73P_3
IO_L72N_3
IO_L72P_3
IO_L71N_3
IO_L71P_3
IO_L70N_3
IO_L70P_3
IO_L69N_3/VREF_3
IO_L69P_3
IO_L68N_3
IO_L68P_3
IO_L67N_3
IO_L67P_3
SR0_A[18..0]
SR0_CK
SR0_CE0
SR0_CE1
SR0_CE2
SR0_CE3
SR0_OE0
SR0_OE1
SR0_OE2
SR0_OE3
SR0_ADV
SR0_ADSP
SR0_ADSC
SR0_BWc
SR0_GW
SR0_BWa
SR0_BWb
2V2000-FF896
Title: Erinyes - FPGA Virtex bank 3
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 25 of 68
File: XC2VX800 Bank 3.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip


- Place testpoints near FPGA
TP2600
TP2601
INIT_B is bidirectional open drain on initial configuration stage. Serial resistor is inserted for extra security. INIT_B must be pulled low at startup to delay configuration.
- VRN and
(FPGA will match resistors
U2000E
VCC_4
IO_L01N_4/DOUT
IO_L01P_4/INIT_B
IO_L02N_4/D0
IO_L02P_4/D1
IO_L03N_4/D2/ALT_VRP_4
IO_L03P_4/D3/ALT_VRN_4
IO_L04N_4/VREF_4
IO_L04P_4
IO_L05N_4/VRP_4
IO_L05P_4/VRN_4
IO_L06N_4
IO_L06P_4
IO_L19N_4
IO_L19P_4
IO_L20N_4
IO_L20P_4
IO_L21N_4
IO_L21P_4/VREF_4
IO_L22N_4
IO_L22P_4
IO_L23N_4
IO_L23P_4
IO_L24N_4
IO_L24P_4
IO_L49N_4
IO_L49P_4
IO_L50N_4
IO_L50P_4
IO_L51N_4
IO_L51P_4/VREF_4
IO_L52N_4
IO_L52P_4
IO_L53N_4
IO_L53P_4
IO_L54N_4
IO_L54P_4
IO_L91N_4/VREF_4
IO_L91P_4
IO_L92N_4
IO_L92P_4
IO_L93N_4
IO_L93P_4
IO_L94N_4/VREF_4
IO_L94P_4
IO_L95N_4/GCLK3S
IO_L95P_4/GCLK2P
IO_L96N_4/GCLK1S
IO_L96P_4/GCLK0P
IO_L67N_4
IO_L67P_4
IO_L68N_4
IO_L68P_4
IO_L69N_4
IO_L69P_4/VREF_4
IO_L70N_4
IO_L70P_4
IO_L71N_4
IO_L71P_4
IO_L72N_4
IO_L72P_4
IO_L73N_4
IO_L73P_4
IO_L74N_4
IO_L74P_4
IO_L75N_4
IO_L75P_4
IO_L76N_4
IO_L76P_4
IO_L77N_4
IO_L77P_4
IO_L78N_4
IO_L78P_4
Bank 4
XC2V2000-FF896
SR1_DOb[8..0]
AA11
AA12
AA13
AA14
AA15
AB10
AB11
AB12
AB13
AH13
AJ5
d3.3v
C2600 0.1uF
C2601 0.1uF
C2602 0.1uF
C2606 0.1uF
C2611 47uF
C2603 0.1uF
C2607 0.1uF
C2612 47uF
C2604 0.1uF
C2609 0.1uF
C2610 0.1uF
SR2_BWb
SR2_ADSC
SR2_ADSP
SR2_ADV
SR2_Oe3
SR2_Oe2
SR2_Oe0
SR2_Oe1
B4v33_GCLK[1..0]
SR2_CE3
SR2_CE2
SR2_CE1
SR2_CE0
SR2_CK
SR2_A[18..0]
B4v33_GCLK[1..0] are differential inputs, length should match.
Title: Exirpes - FPGA Virtex bank 4
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 26 of 68
File: XC2VX000 Bank 4.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

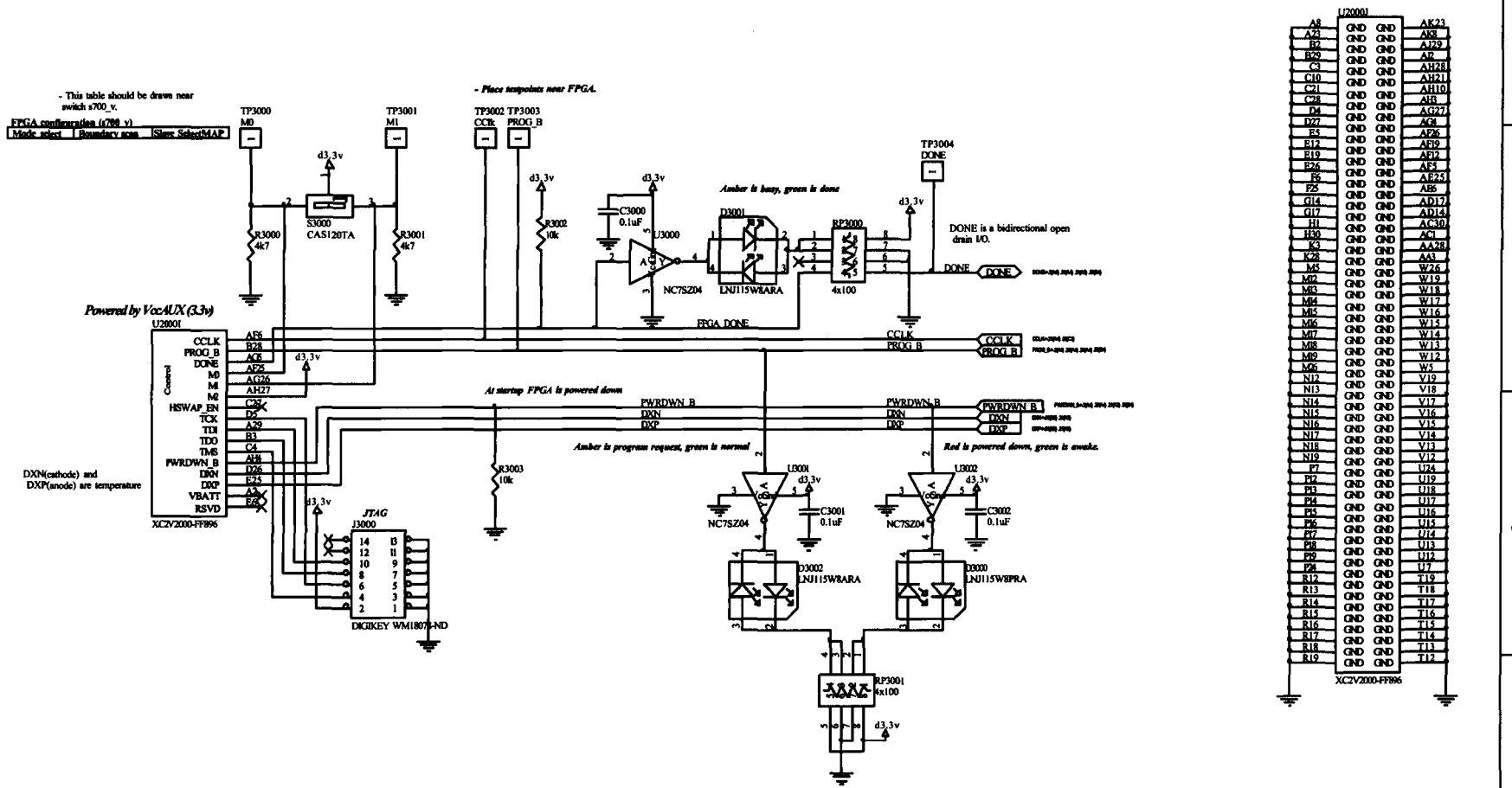


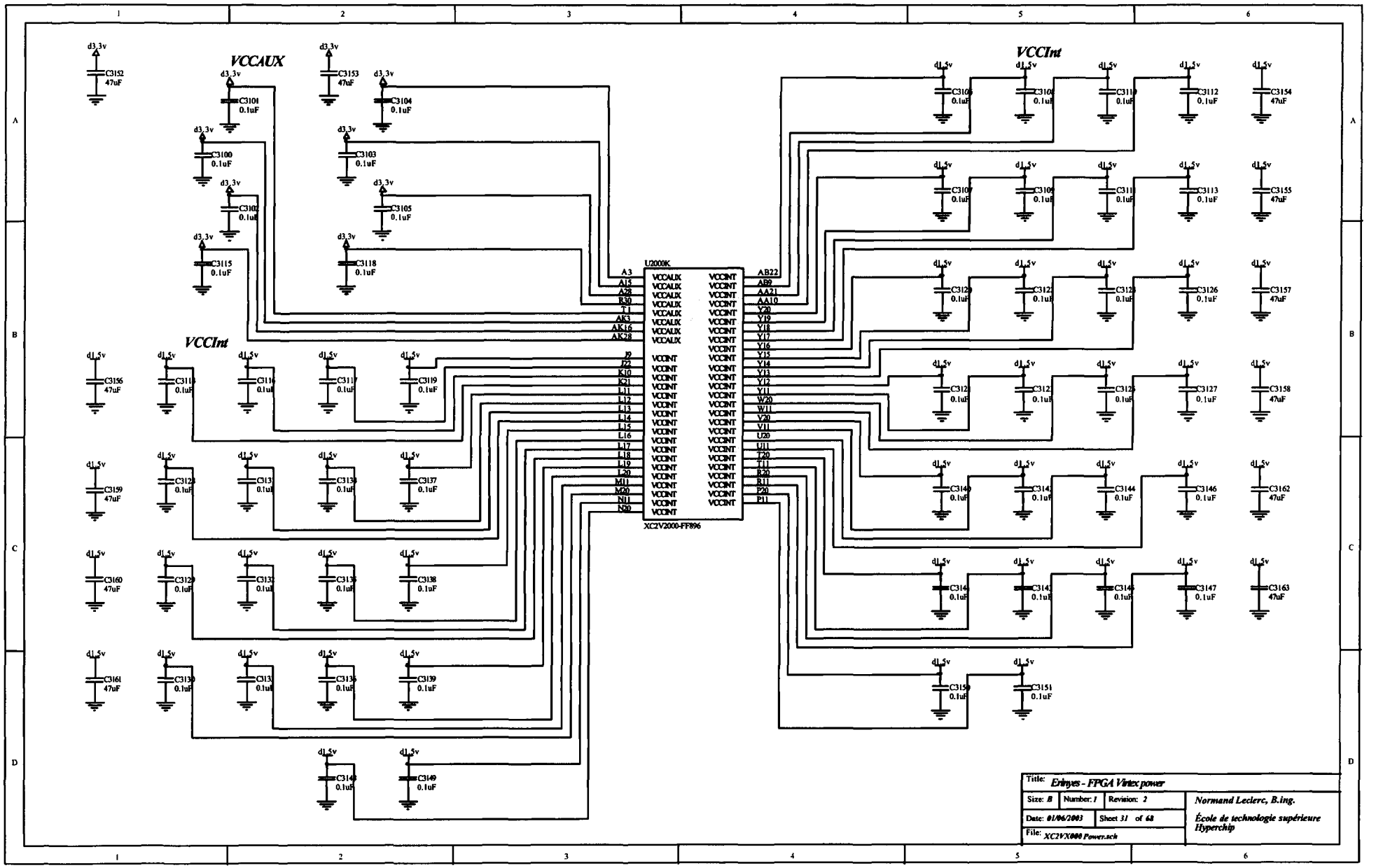
Title: *Eriques - FPGA Virtex bank 5*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *27* of *68*		*École de technologie supérieure Hyperchip*
File: *XC2VX800 Bank 5.sch*			

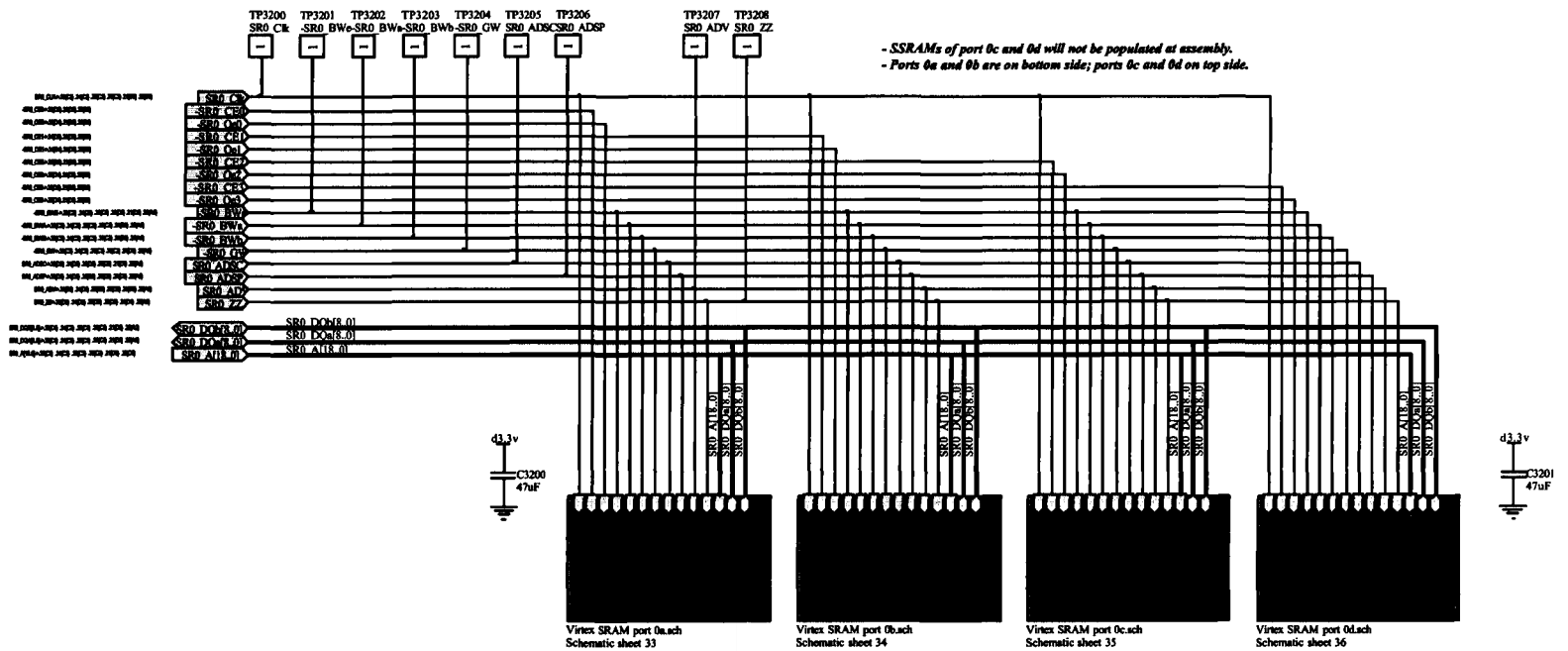

- VRN and VRP are used for DCI
(FPGA will match resistors value)
U2000G
XC2V2000-FF896
Bank 6
d3.3v
C2800 0.1uF
C2801 0.1uF
C2802 0.1uF
C2806 0.1uF
C2811 47uF
C2803 0.1uF
C2804 0.1uF
C2805 0.1uF
C2807 0.1uF
C2812 47uF
C2809 0.1uF
C2810 0.1uF
SMBC
SMBDat
Alert
SCL
SDA
DCLK
SDO
SDI
SMBClk is 400kHz
B6v33_IO0 AC24
B6v33_IO1 AB24
B6v33_IO2 AD27
B6v33_IO3 AC27
B6v33_IO4 AC26
B6v33_IO5 AB26
B6v33_IO6 AA23
B6v33_IO7 Y23
B6v33_IO8 AC28
B6v33_IO9 AB28
B6v33_IO10 AD30
B6v33_IO11 AE30
B6v33_IO12 AB25
B6v33_IO13 AA25
B6v33_IO14 AA24
B6v33_IO15 Y24
B6v33_IO16 AC29
B6v33_IO17 AB30
B6v33_IO18 Y25
B6v33_IO19 W25
B6v33_IO20 AB27
B6v33_IO21 AA27
B6v33_IO[21..0]
VLED11 AA29
VLED10 AB29
B6v33_IO[21..0]
VLED[11..0]
VLED[11..0]
Title: Eriepes - FPGA Virtex bank 6
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 28 of 68
File: XC2VX000 Bank 6.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip



Title: *Erinyes - FPGA Virtex bank 7*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *29* of *68*		*École de technologie supérieure Hyperchip*
File: *XC2VX800 Bank 7.sch*			

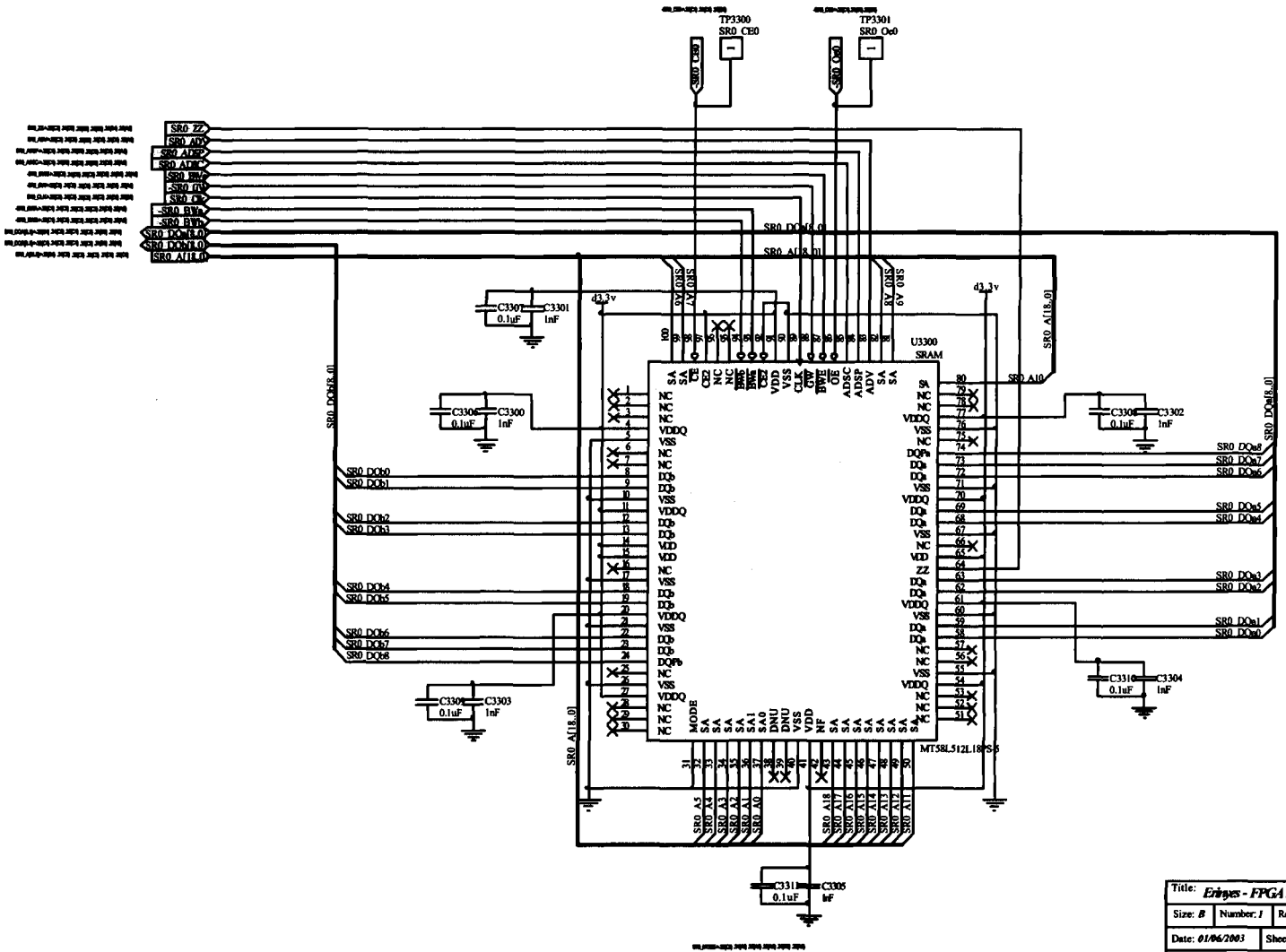

- This table should be drawn near switch s700_v.
FPGA configuration (s700_v)
Mode select
Boundary scan
Slave SelectMAP
- Place testpoints near FPGA.
TP3000 M0
TP3001 M1
TP3002 CCIk
TP3003 PROG_B
TP3004 DONE
d3,3v
S3000 CAS120TA
R3000 4k7
R3001 4k7
R3002 10k
R3003 10k
C3000 0.1uF
U3000
NC7SZ04
D3001
LNJ115W8ARA
RP3000 4x100
Amber is busy, green is done
DONE is a bidirectional open drain I/O.
DONE
FPGA_DONE
Powered by VccAUX (3.3v)
U2000I
CCLK
PROG_B
DONE
M0
M1
M2
HSWAP_EN
TCK
TDI
TDO
TMS
PWRDWN_B
DXN
DXP
VBATT
RSVD
Control
XC2V2000-FF896
At startup FPGA is powered down
PWRDWN_B
DXN
DXP
Amber is program request, green is normal
Red is powered down, green is awake.
U3001
U3002
C3001 0.1uF
C3002 0.1uF
D3002 LNJ115W8ARA
D3000 LNJ115W8PRA
RP3001 4x100
DXN(cathode) and DXP(anode) are temperature
JTAG
J3000
DIGIKEY WM1807-ND
U2000I
XC2V2000-FF896
Title: Erinyes - FPGA Virtex control and ground
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 38 of 68
File: XC2VX000 Control and Ground.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip


VCCAUX
VCCInt
VCCInt
U2000K
XC2V2000-FF896
Title: Entrees - FPGA Virtex power
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 31 of 68
File: XC2VX000 Power.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip



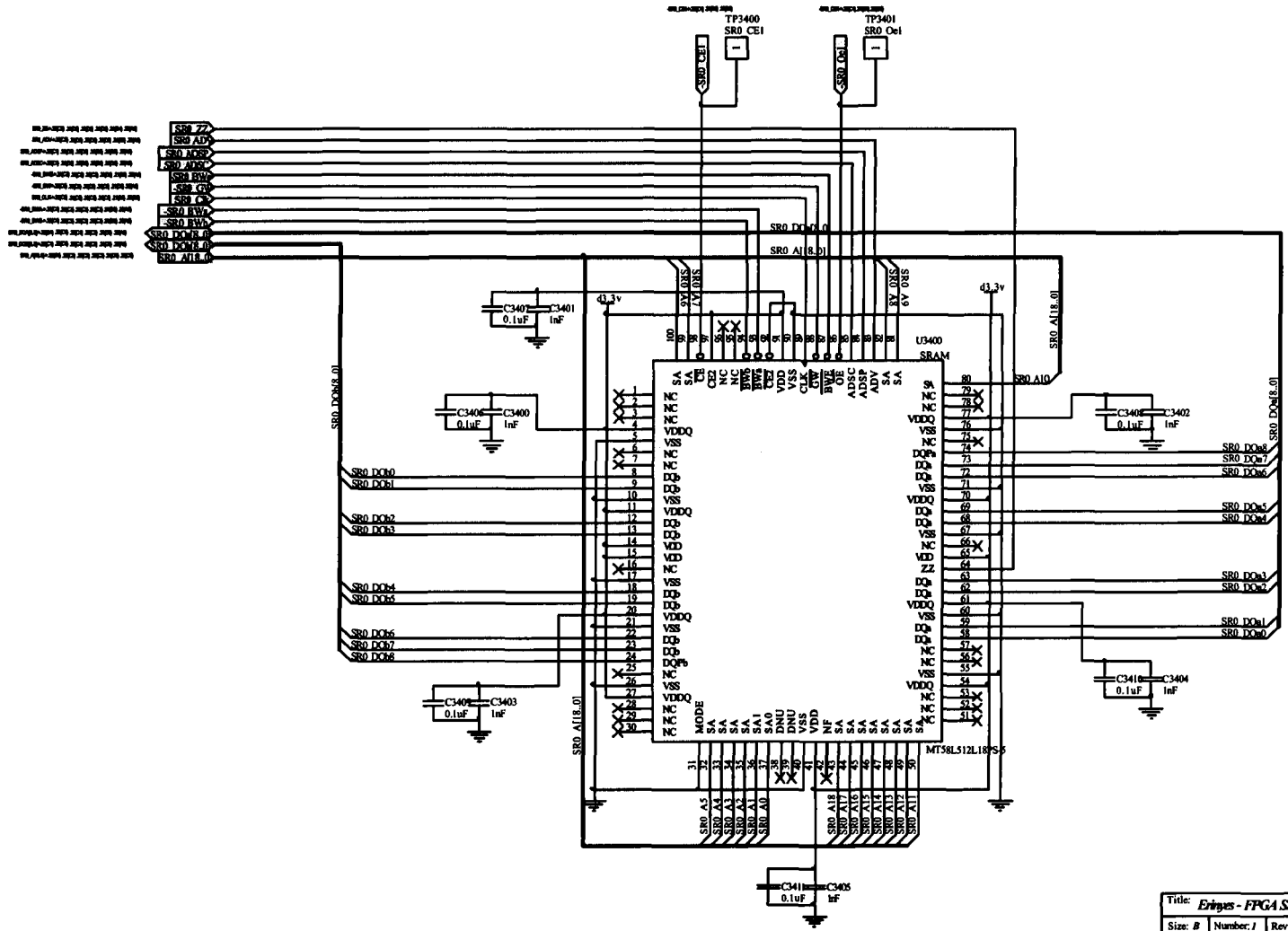
Title: *Eriques - FPGA SRAM port 0*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *32* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 0.sch*			

- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDDVCC pins.
- All capacitors should be tantalum.



Title: *Erinyes - FPGA SRAM port 0*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *33* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 0a.sch*			

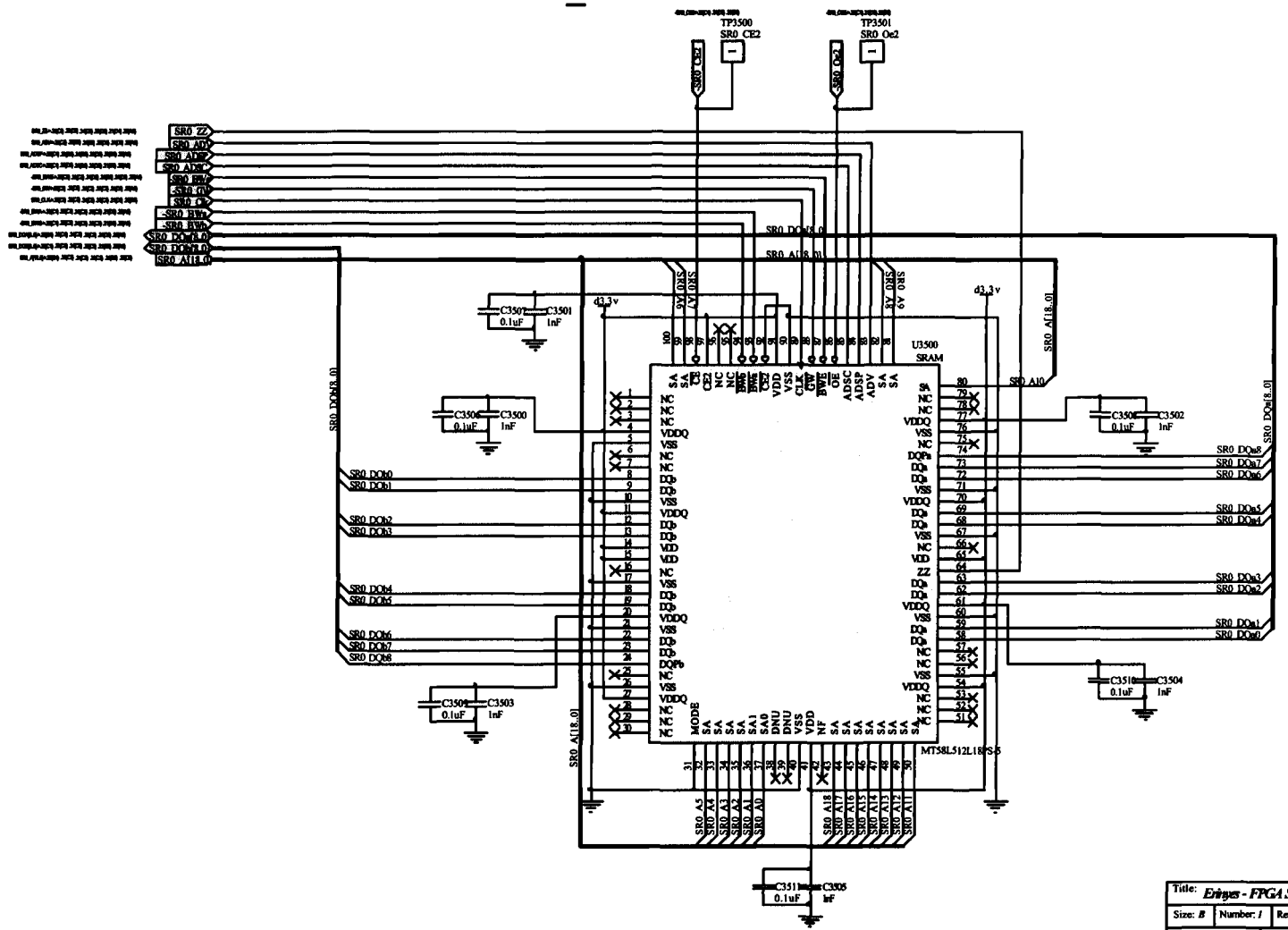
- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.



Title: *Eriques - FPGA SRAM port 0*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.Ing.*
Date: *01/06/2003*	Sheet *34* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 0b.sch*			

- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.

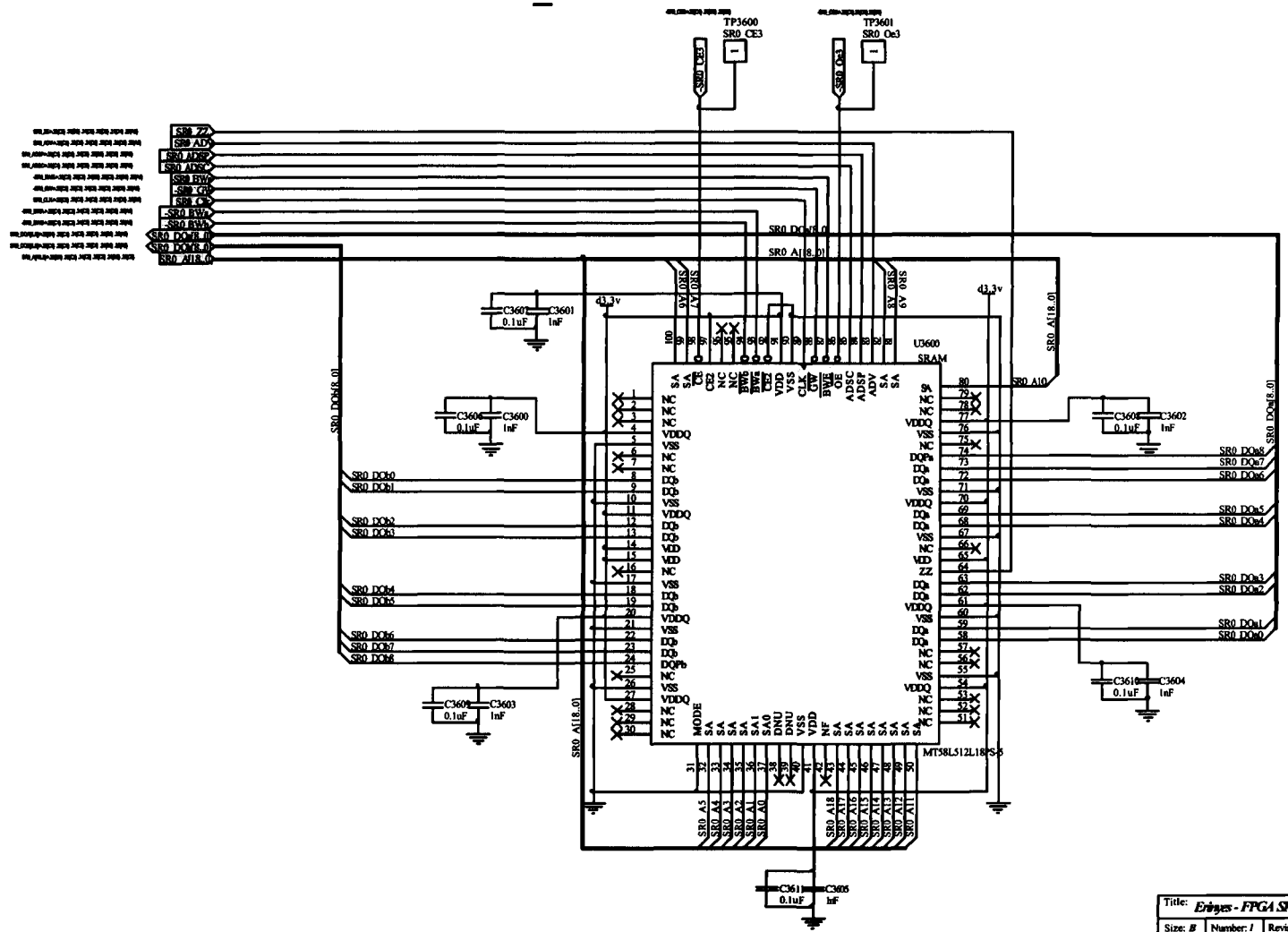
U390_v: NOT POPULATED AT ASSEMBLY



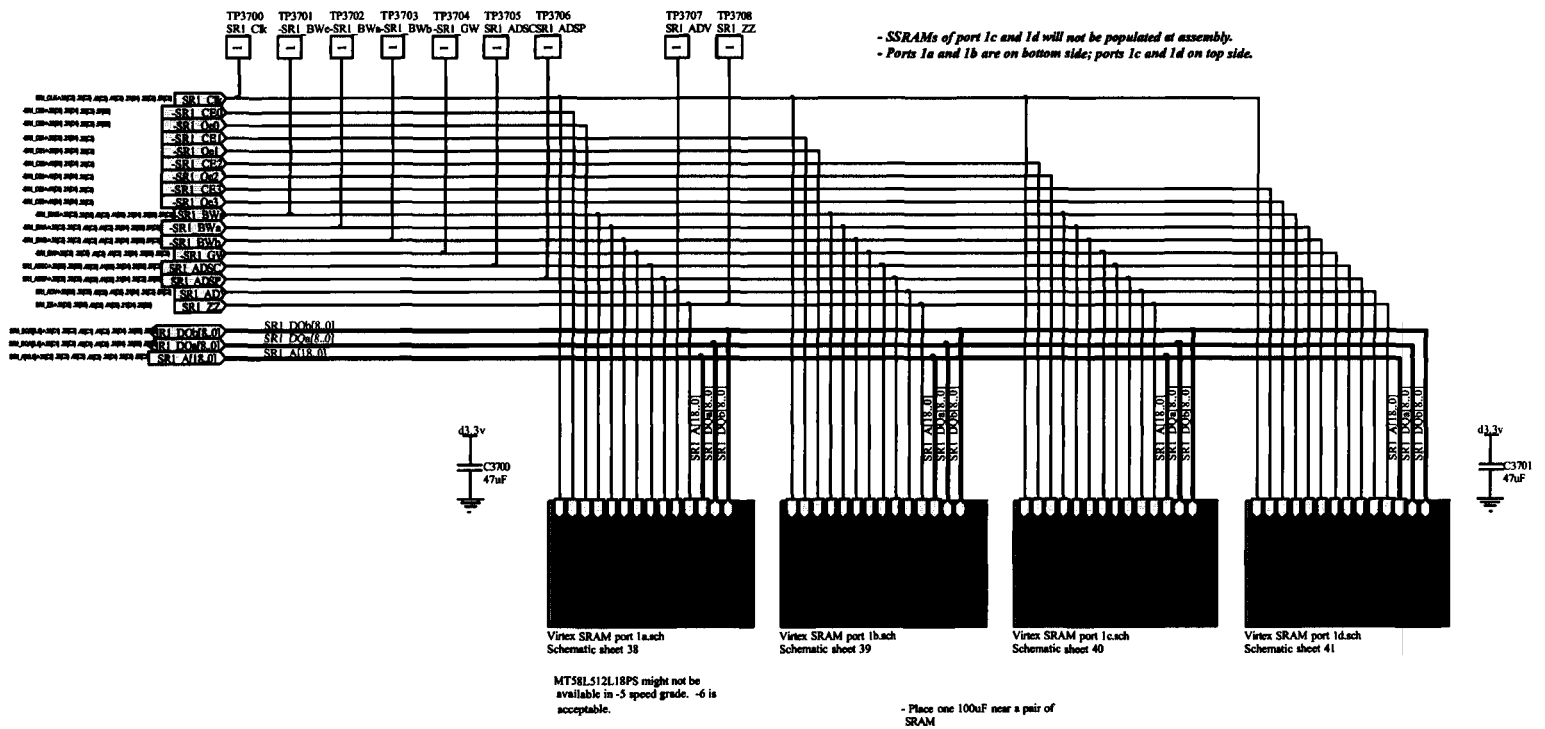
Title: *Eriques - FPGA SRAM port 0*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *35* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 0c.sch*			

- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.

U405_v: NOT POPULATED AT ASSEMBLY

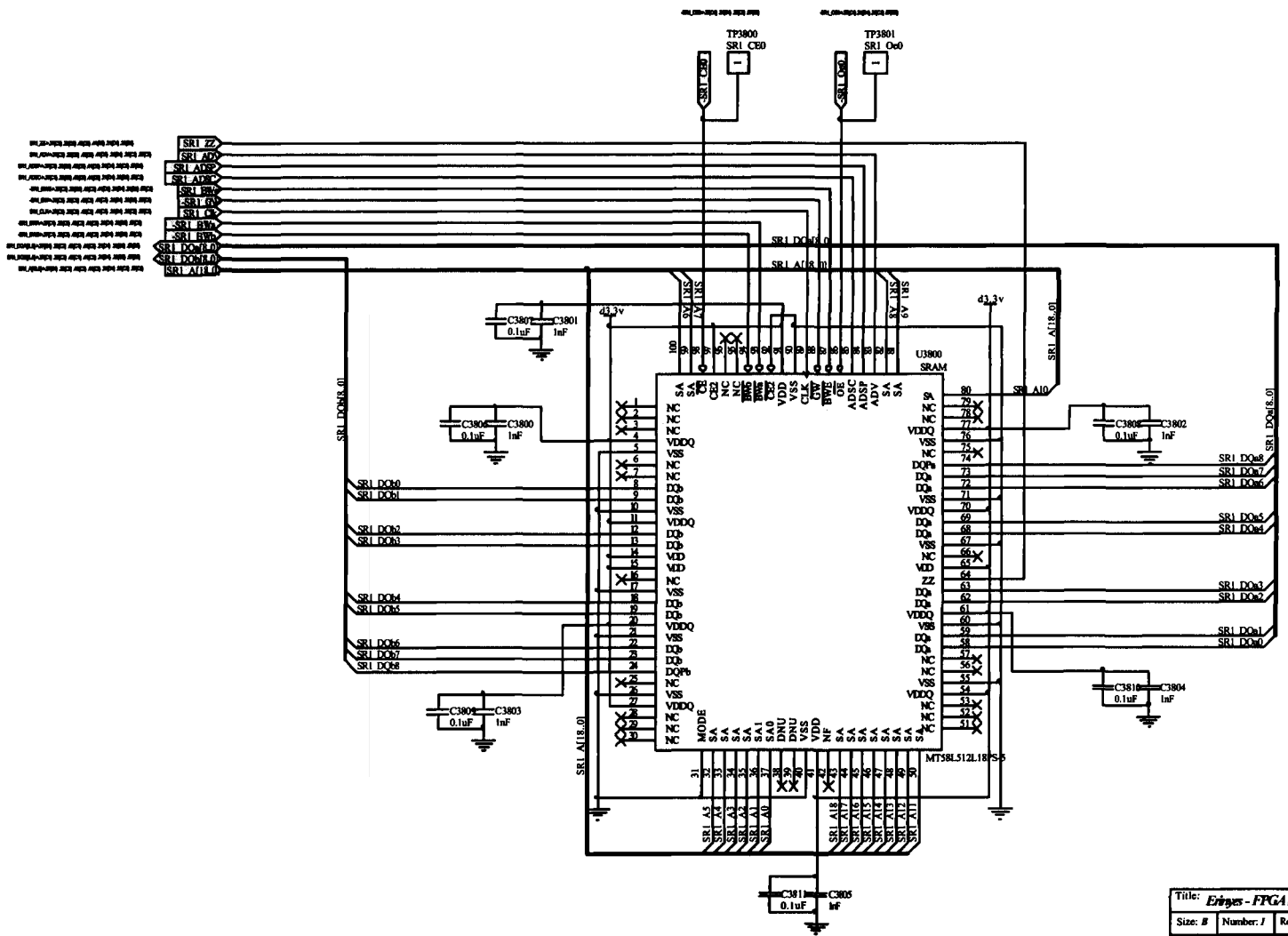


Title: *Eriyes - FPGA SRAM port 0*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *36* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 0d.sch*			



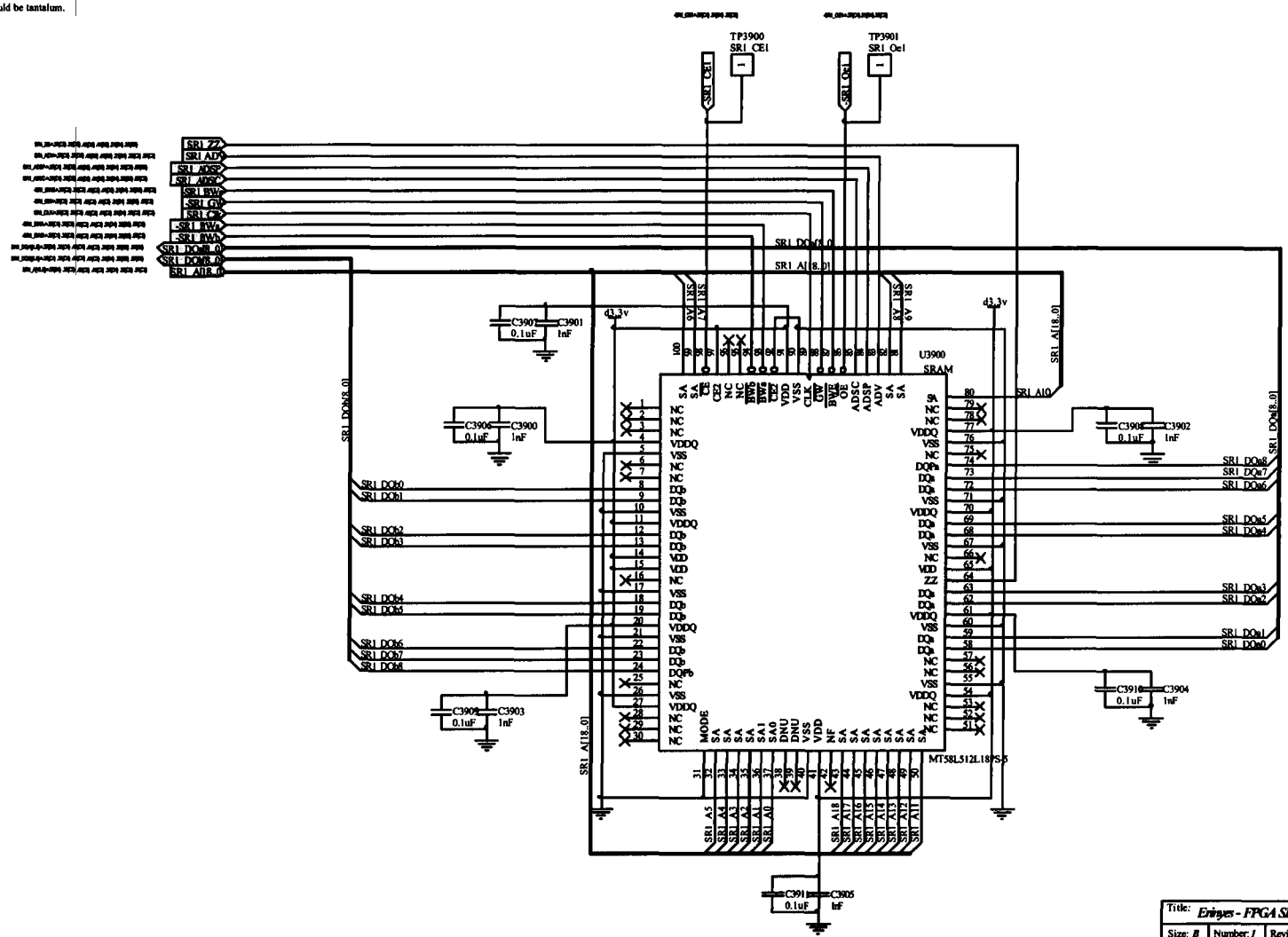
Title: *Eriques - FPGA SRAM port 1*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *37* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 1.sch*			

- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.



Title: *Erinyes - FPGA SRAM port 1*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *38* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 1a.sch*			

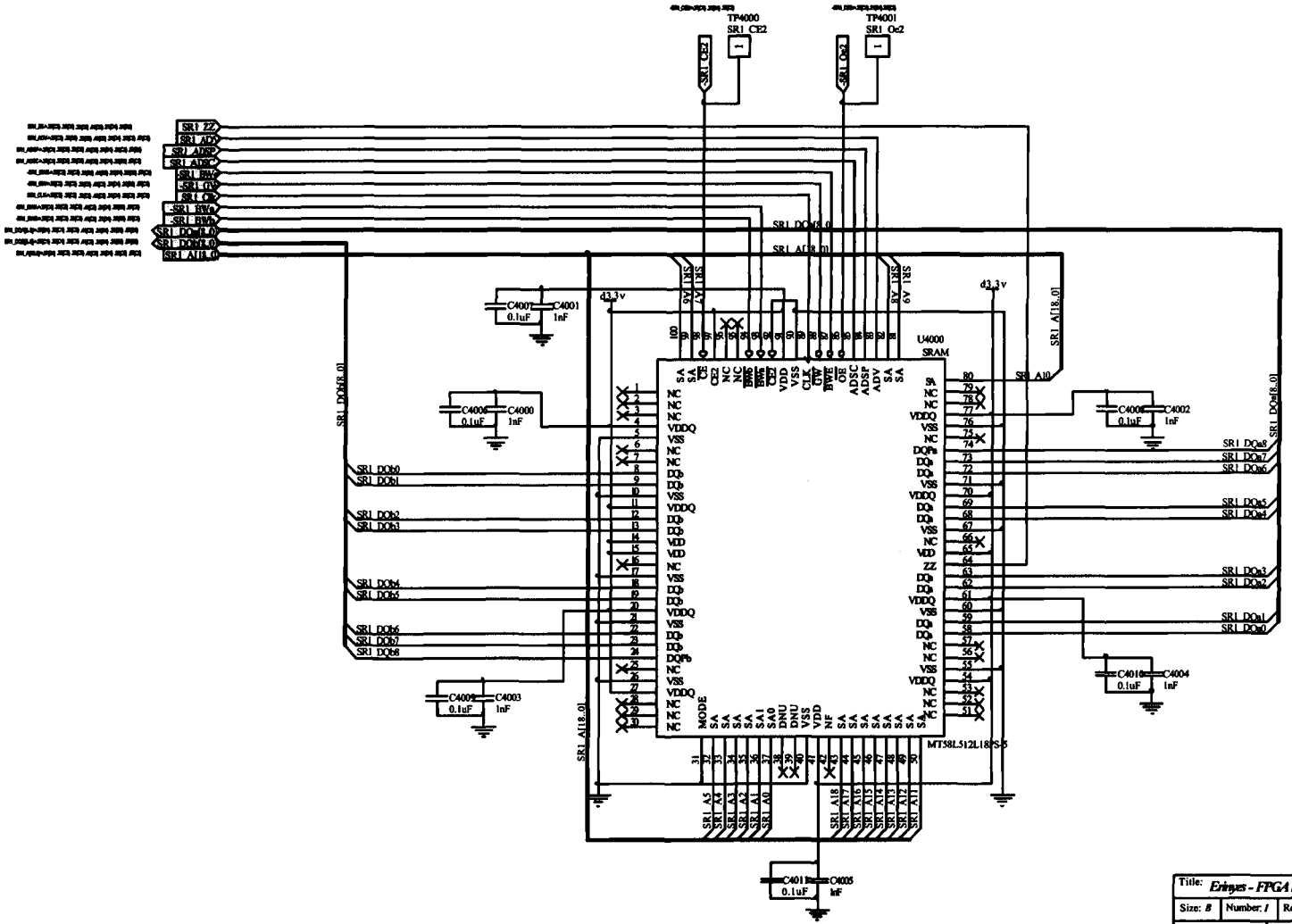
- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.



Title: *Eniyes - FPGA SRAM port 1*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *39* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 1b.sch*			

- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.

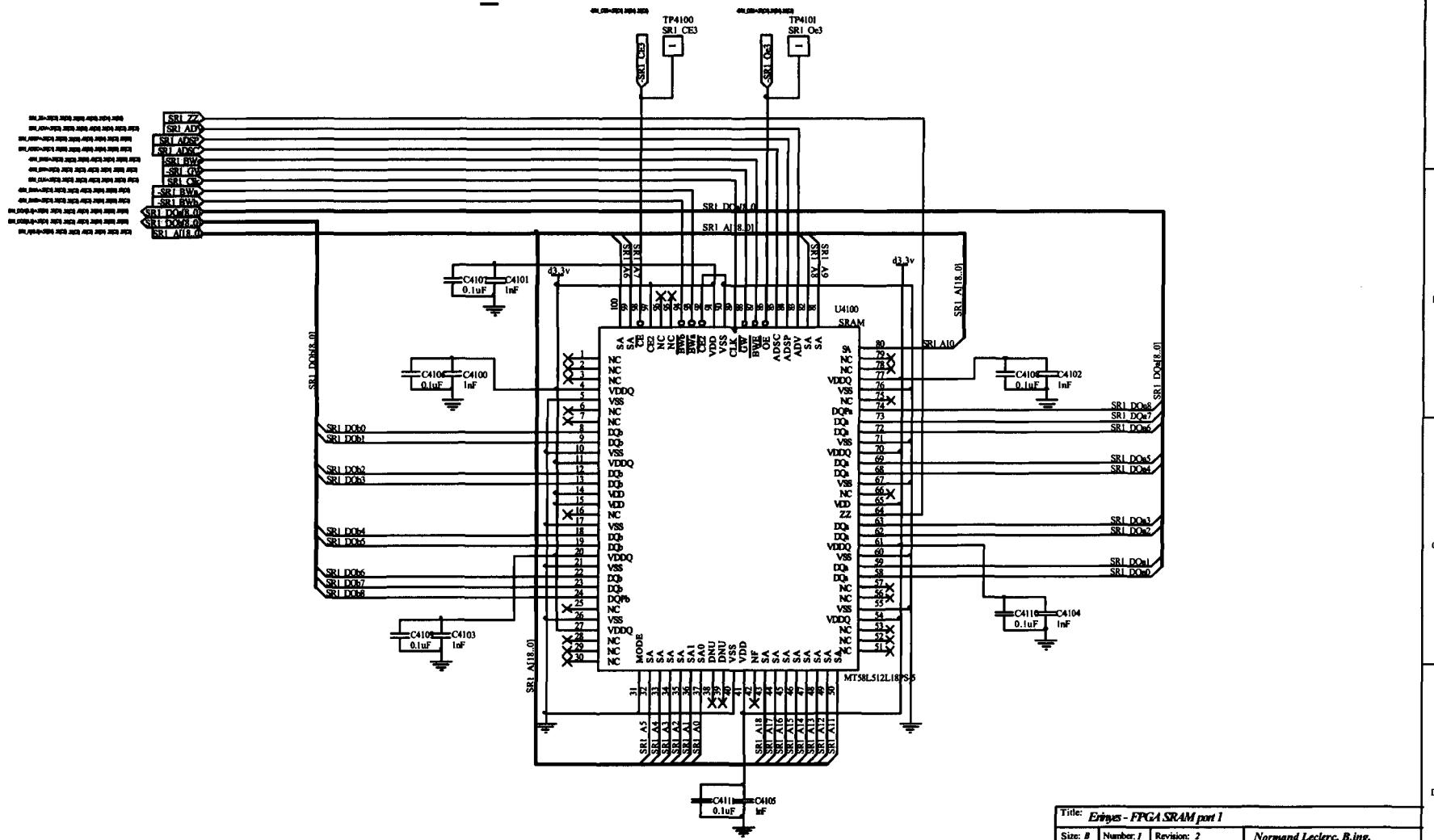
U460_v: NOT POPULATED AT ASSEMBLY



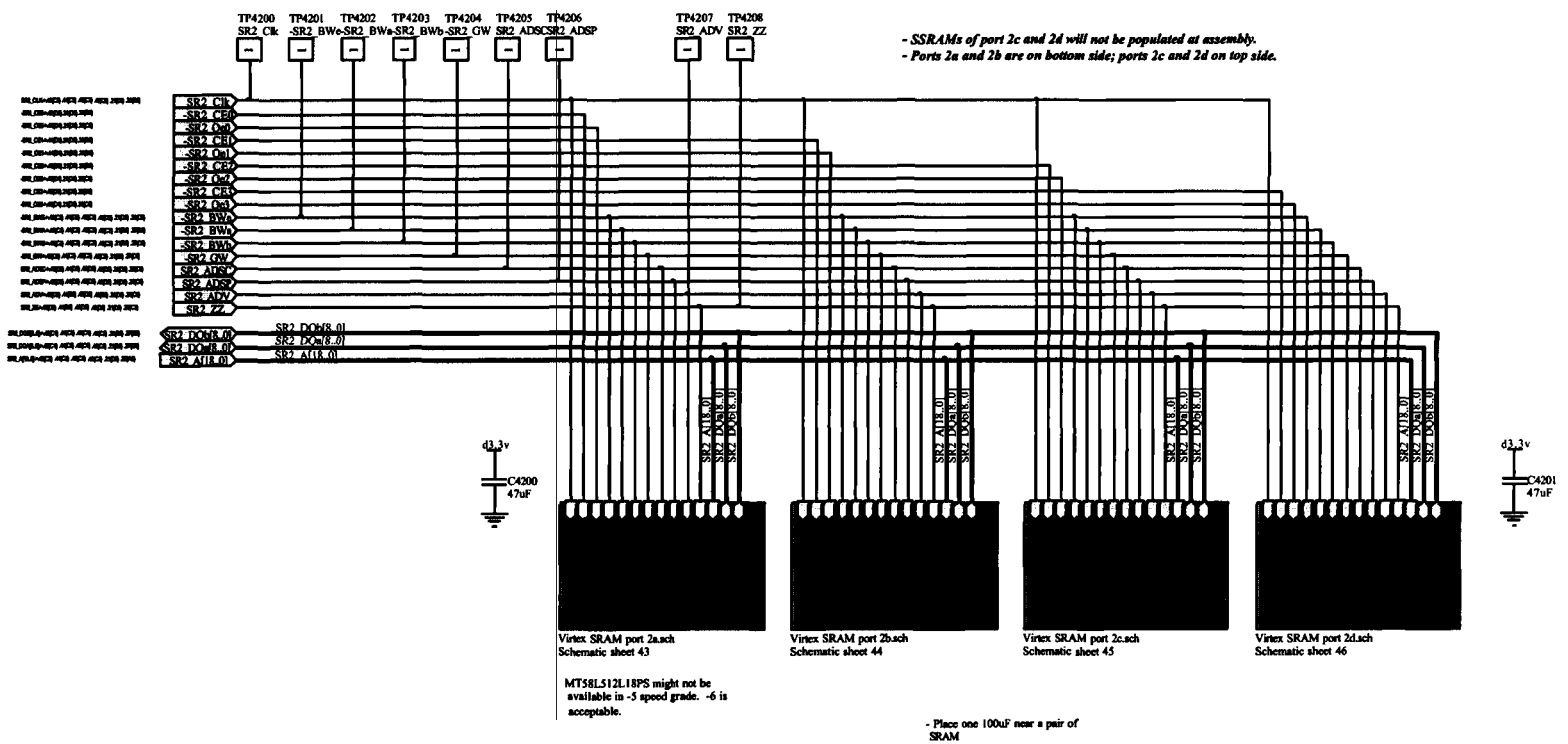
Title: *Erinyes - FPGA SRAM port 1*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *40* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 1c.sch*			

- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.

U475_v: NOT POPULATED AT ASSEMBLY

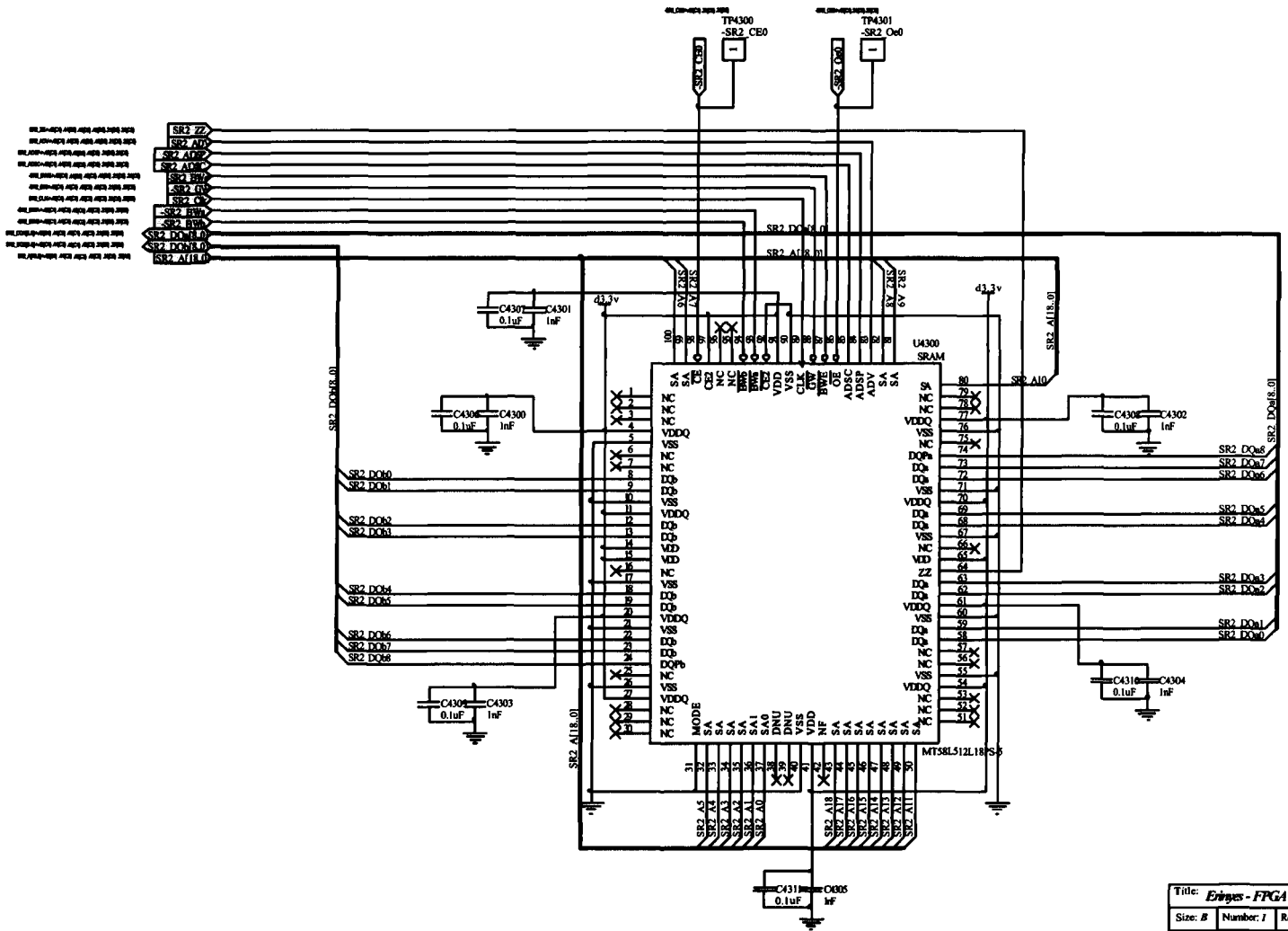


Title: *Eriques - FPGA SRAM port 1*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *41* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 1d.sch*			



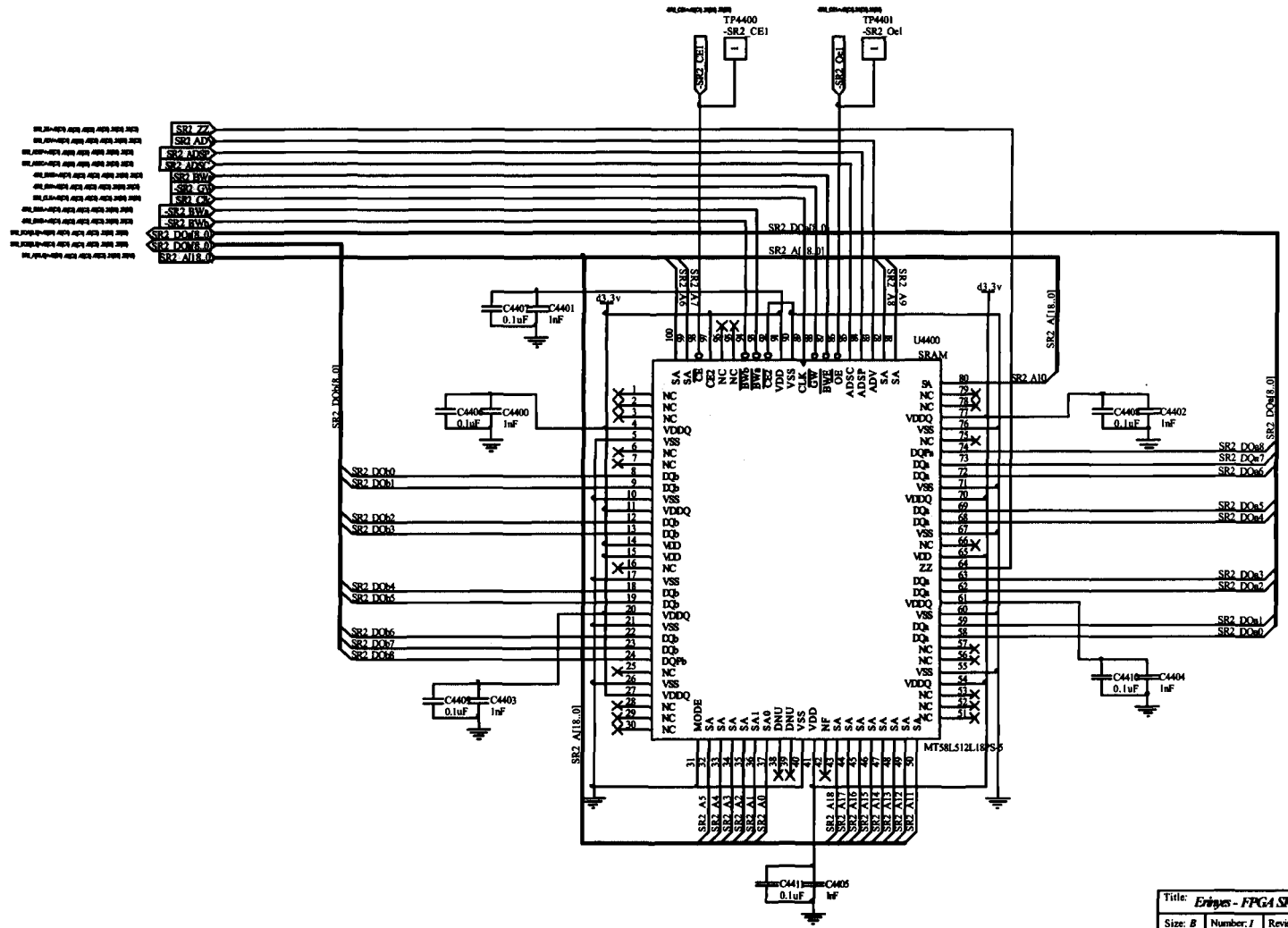
Title: *Eriyes - FPGA SRAM port 2*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *42* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 2.sch*			

- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.



Title: *Eringes - FPGA SRAM port 2*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *43* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 2a.sch*			

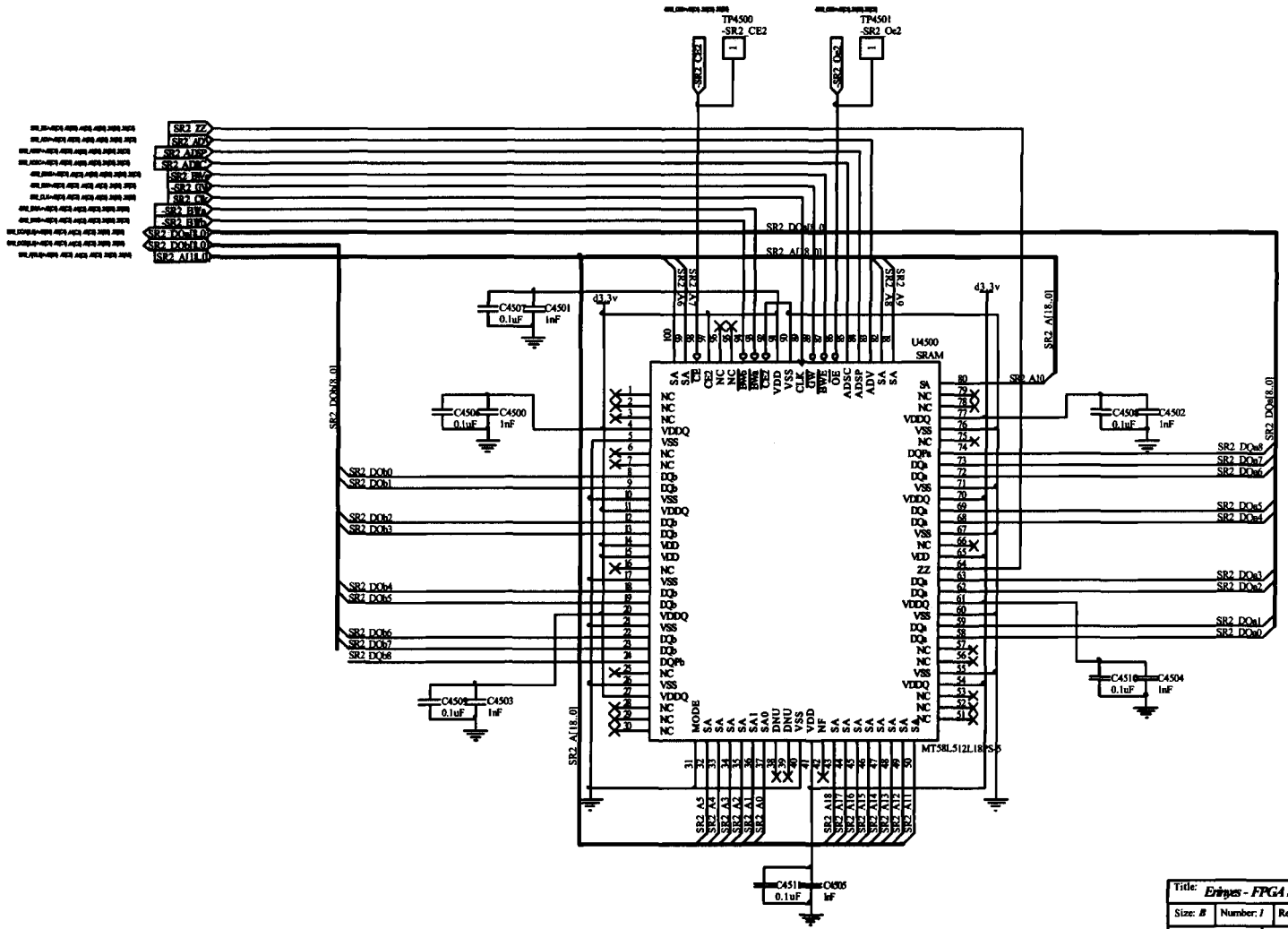
- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.



Title: *Erinyes - FPGA SRAM port 2*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *44* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 2b.sch*			

- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.

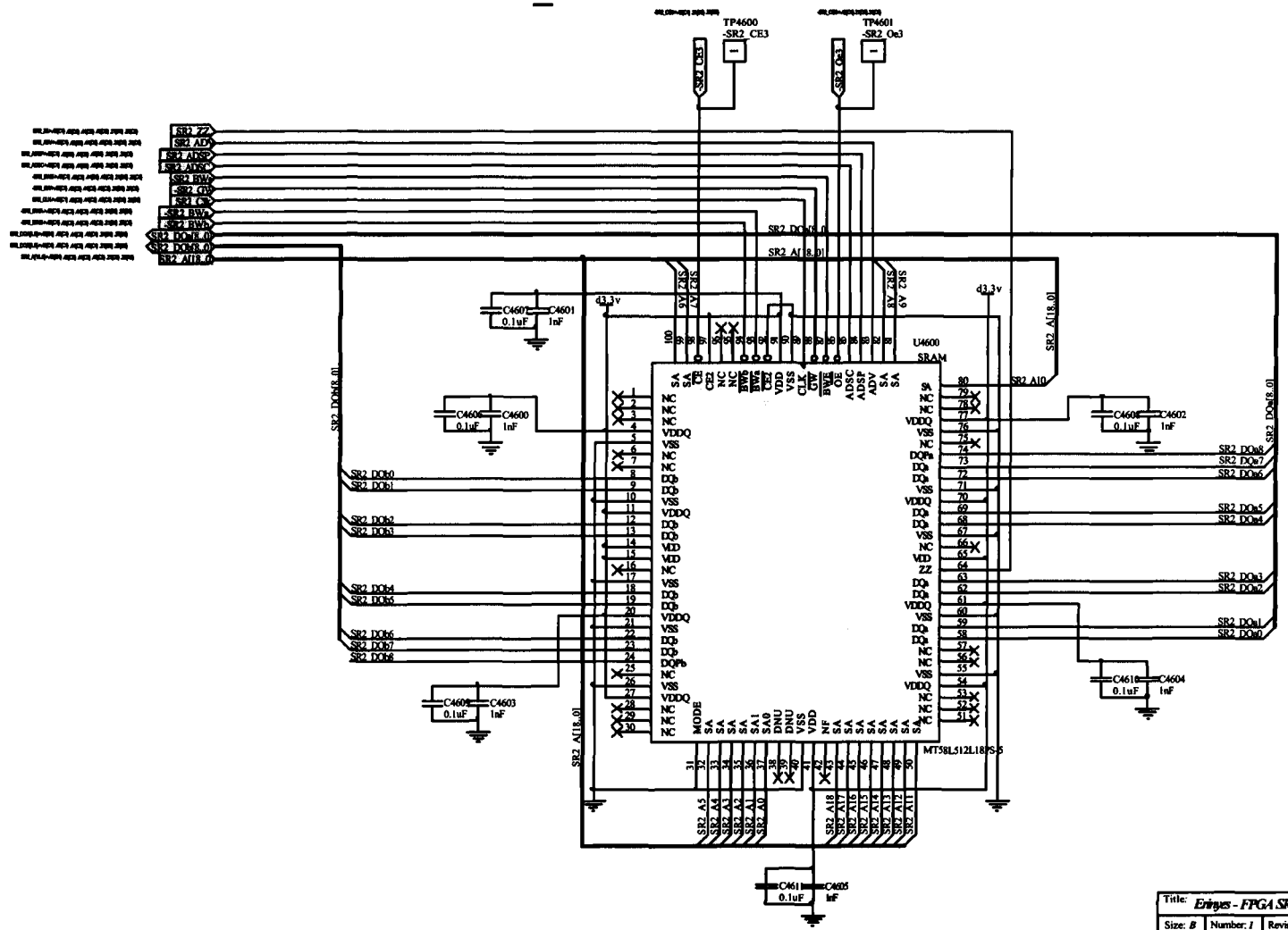
U530_v: NOT POPULATED AT ASSEMBLY



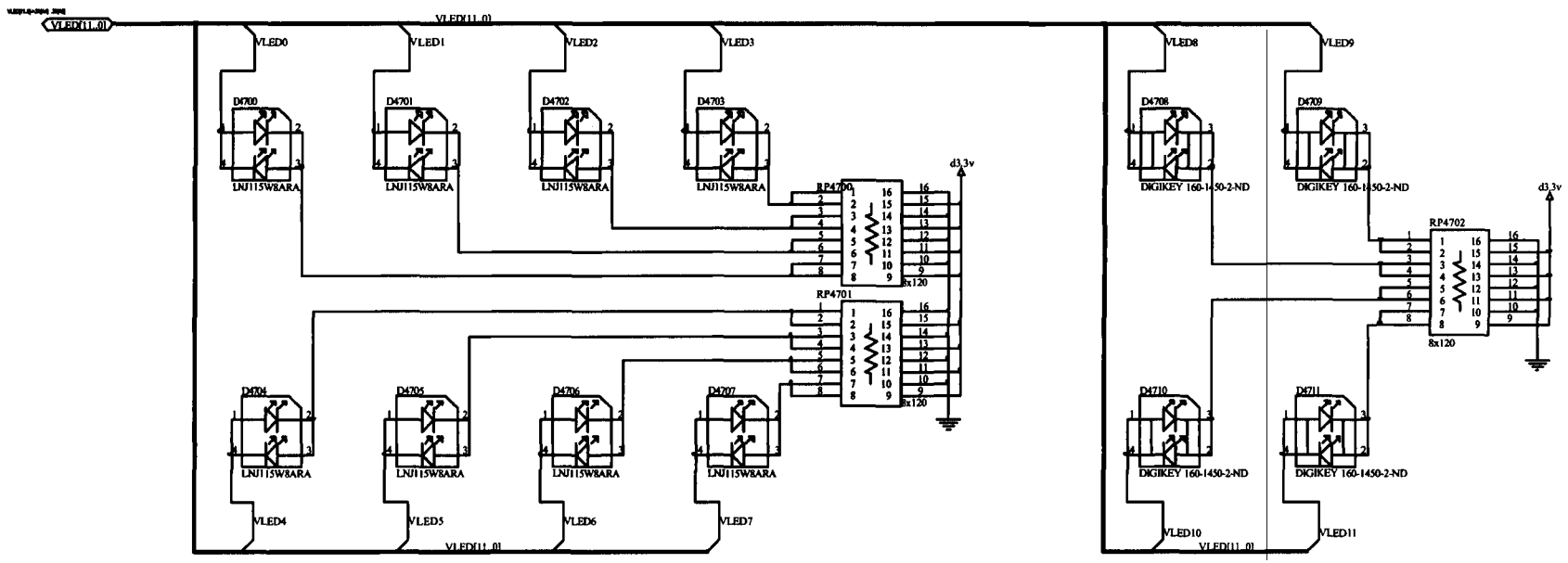
Title: *Eriyes - FPGA SRAM port 2*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *45* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 2c.sch*			

- Place termination resistors as close as possible to SRAM.
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
- All capacitors should be tantalum.

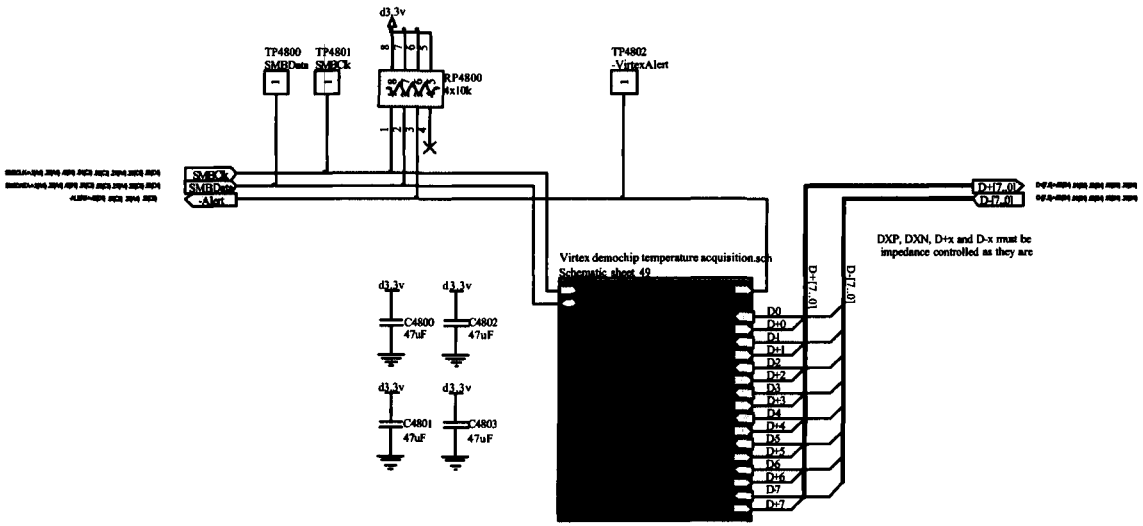
U545_v: NOT POPULATED AT ASSEMBLY



Title: *Eriyes - FPGA SRAM port 2*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *46* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM port 2d.sch*			



Title: *Erinyes - FPGA indicator leds*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *47* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex Leds.sch*			

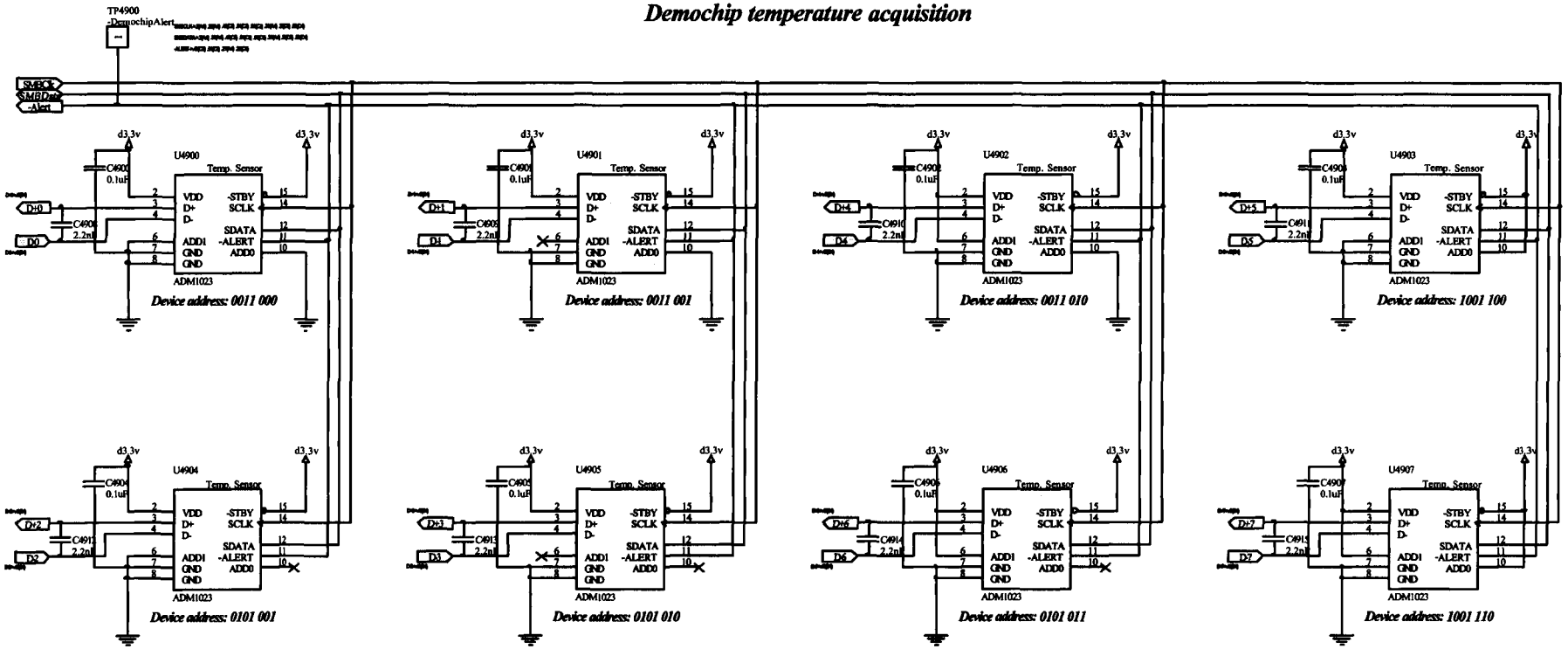


Title: *Erinyes - FPGA I2C bus*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *48* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SMB bus.sch*			

- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.

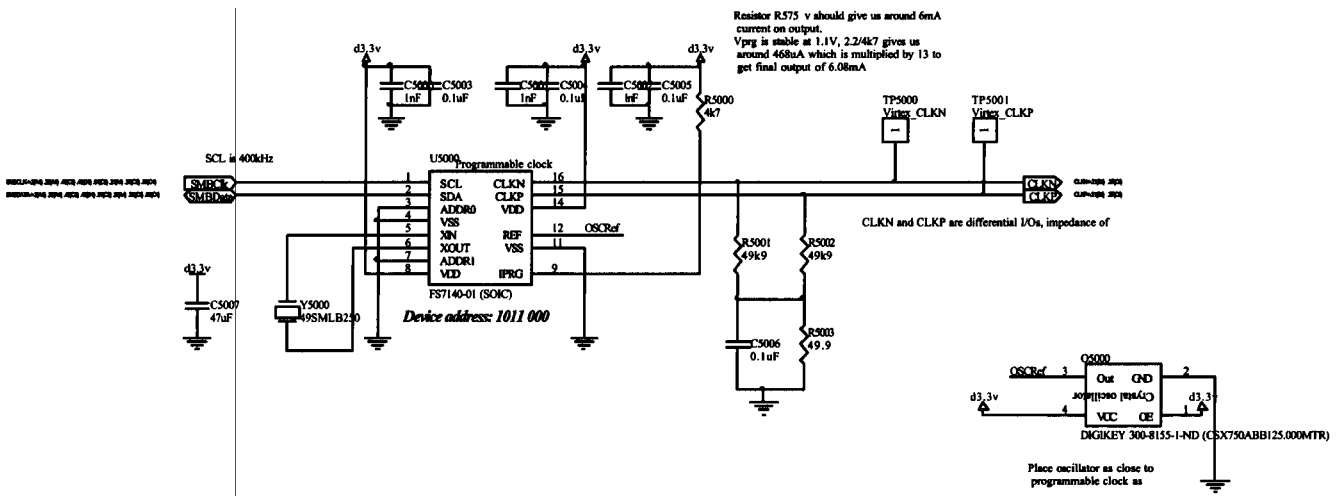
D+x and D-x must be impedance controlled as they are current based.

Demochip temperature acquisition

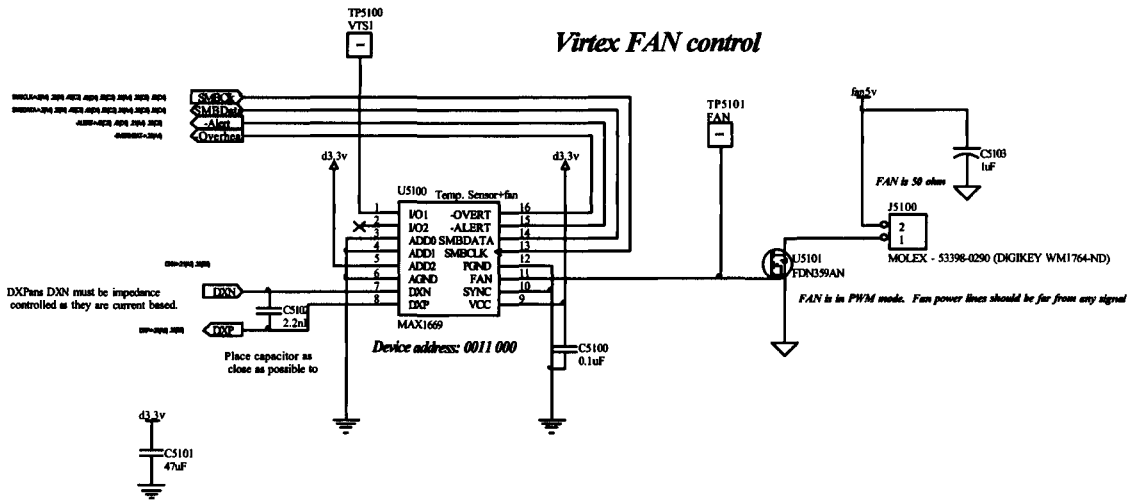


Title: *Erinyes - FPGA demochip temperature acquisition*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *49* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex demochip temperature acquisition.sch*			

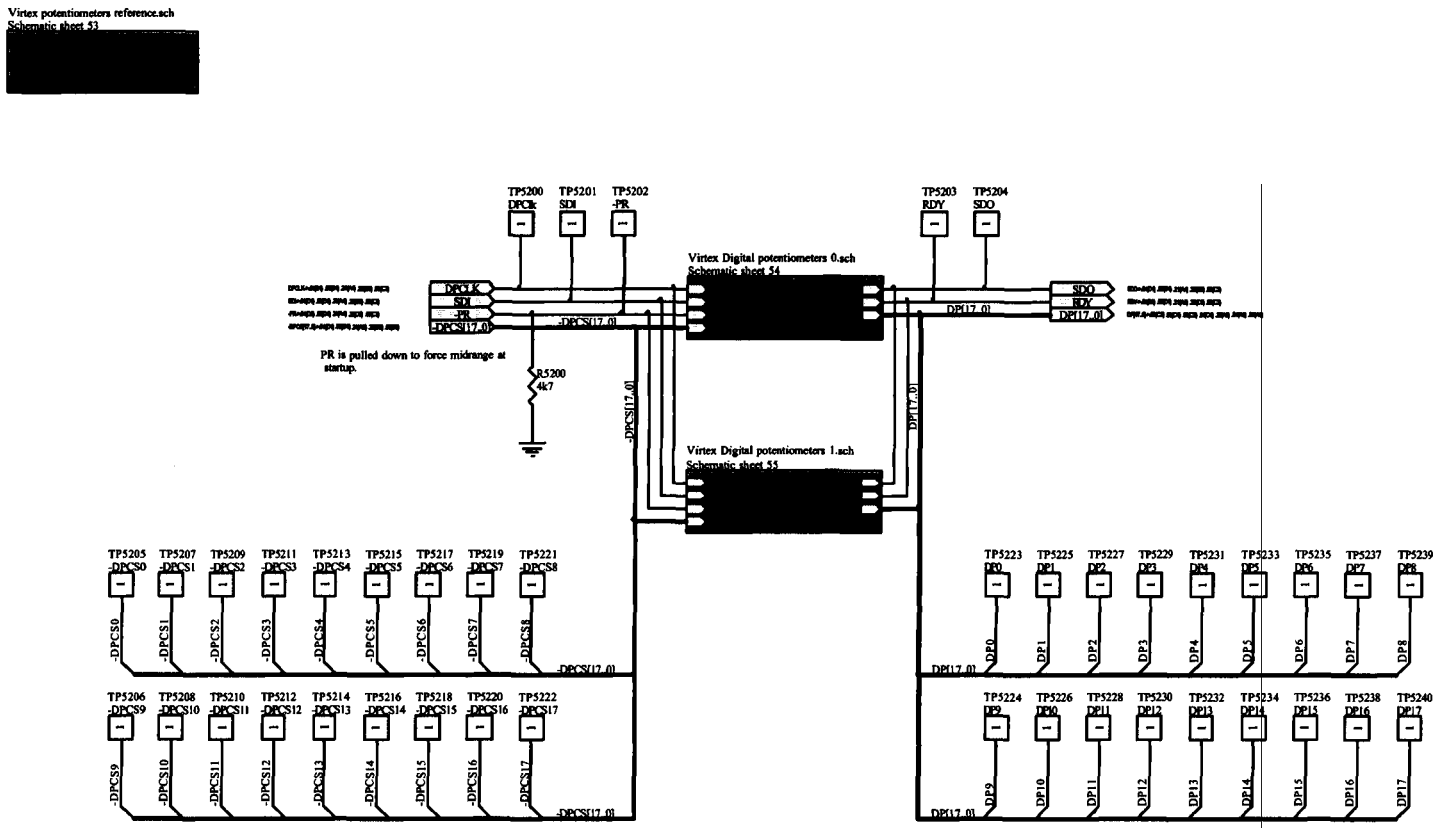
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.



Title: *Erinyes - FPGA programmable clock*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *50* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex Programmable clock.sch*			

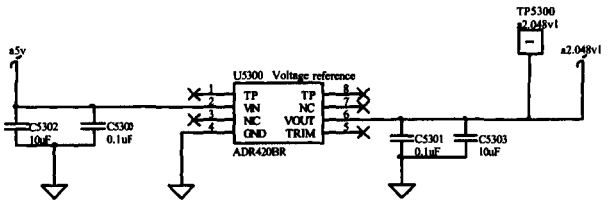


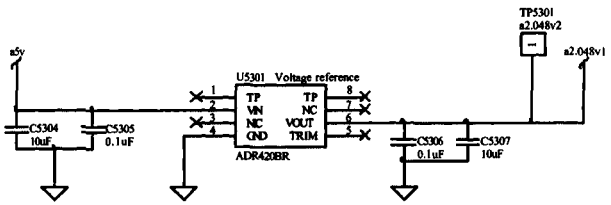
Title: *Eringes - FPGA cooling fan control*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *51* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex fan controll.sch*			

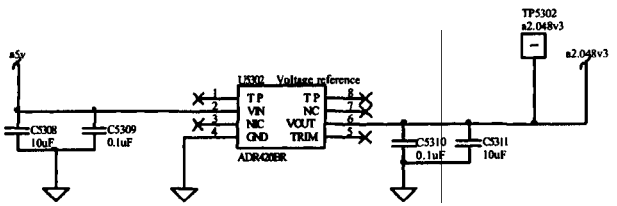


Title: *Erinyes - FPGA digital potentiometers*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *52* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex Digital Potentiometers.sch*			

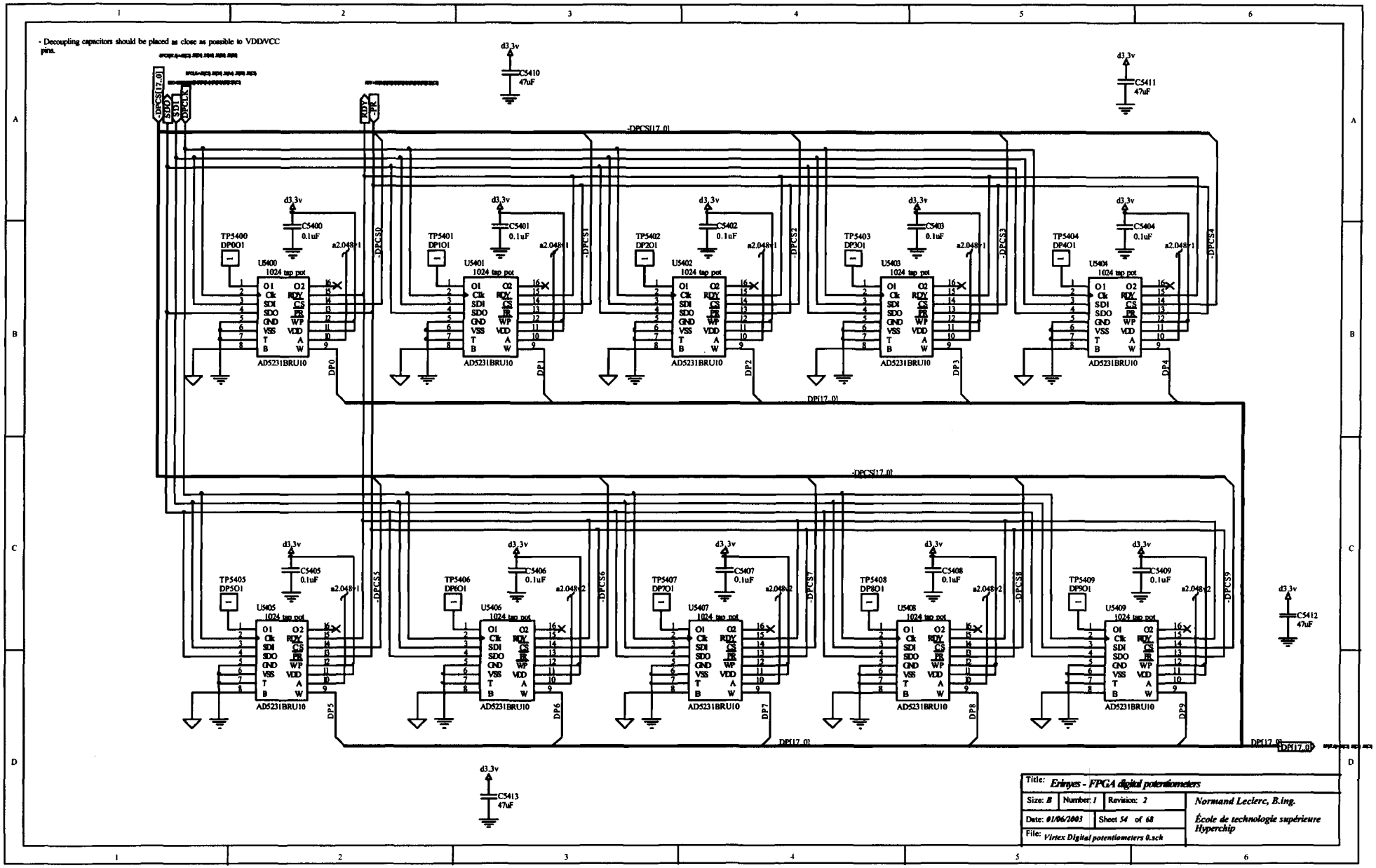
- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.

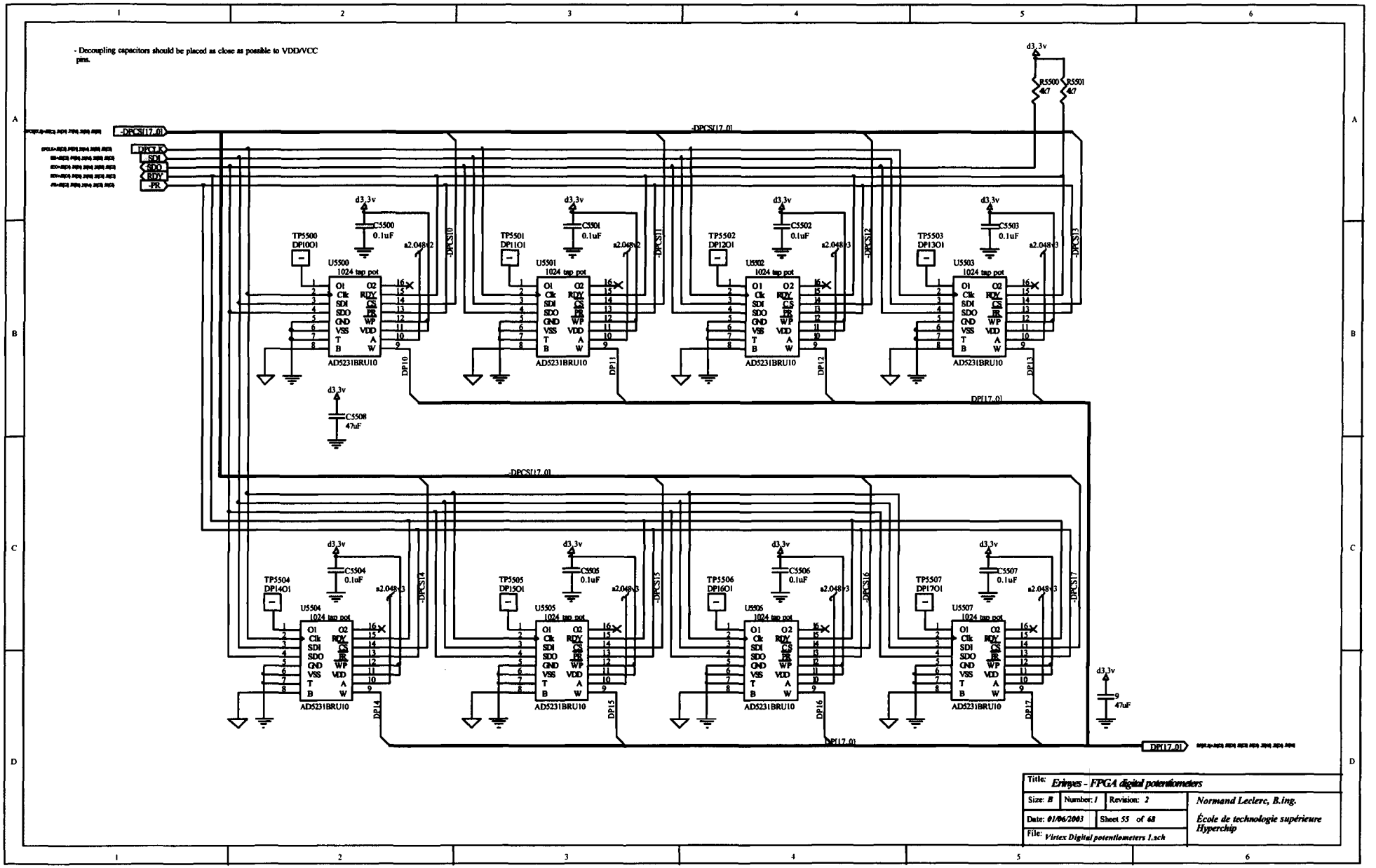


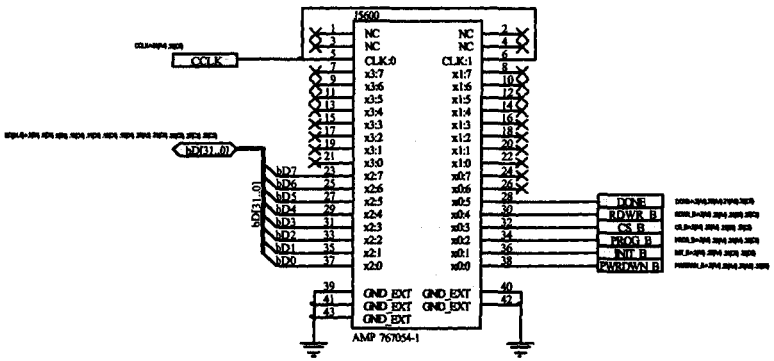




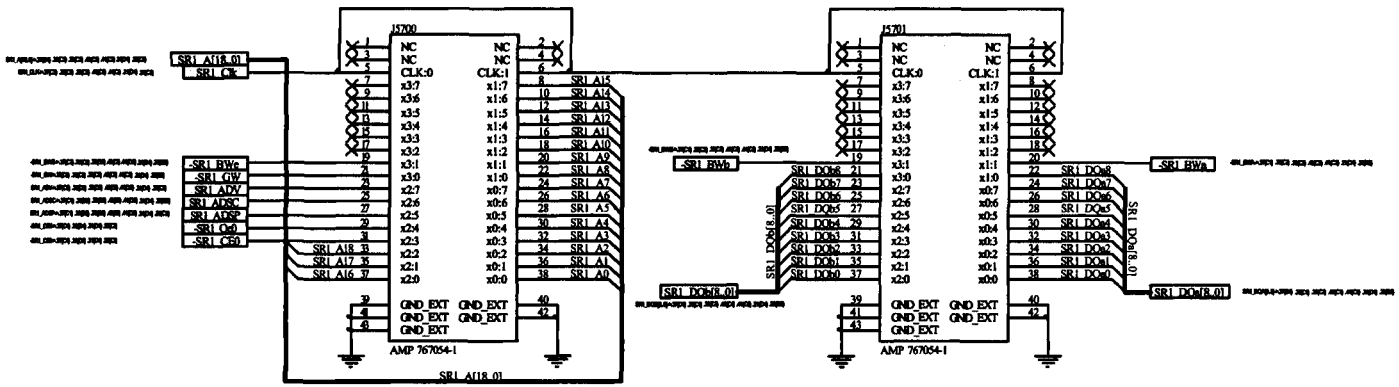
Title: *Erinyes - FPGA digital potentiometers reference*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *53* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex potentiometers reference.sch*			


- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
-DPCSU17_01
SDO
SDI
DPCLK
RDY
-PR
d3.3v
C5410
47uF
C5411
47uF
C5412
47uF
C5413
47uF
C5400
0.1uF
C5401
0.1uF
C5402
0.1uF
C5403
0.1uF
C5404
0.1uF
C5405
0.1uF
C5406
0.1uF
C5407
0.1uF
C5408
0.1uF
C5409
0.1uF
TP5400
DP0O1
TP5401
DP1O1
TP5402
DP2O1
TP5403
DP3O1
TP5404
DP4O1
TP5405
DP5O1
TP5406
DP6O1
TP5407
DP7O1
TP5408
DP8O1
TP5409
DP9O1
U5400
U5401
U5402
U5403
U5404
U5405
U5406
U5407
U5408
U5409
1024 tap pot
AD5231BRU10
a2.048v1
a2.048v2
-DPCS0
-DPCS1
-DPCS2
-DPCS3
-DPCS4
-DPCS5
-DPCS6
-DPCS7
-DPCS8
-DPCS9
DP0
DP1
DP2
DP3
DP4
DP5
DP6
DP7
DP8
DP9
DP17_0
Title: Eriyes - FPGA digital potentiometers
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 54 of 68
File: Virtex Digital potentiometers 0.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

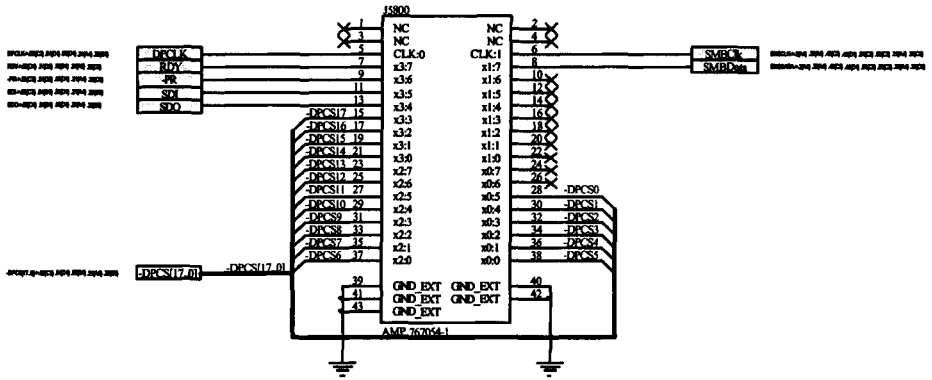

- Decoupling capacitors should be placed as close as possible to VDD/VCC pins.
d3.3v
R5500 4k7
R5501 4k7
-DPCS[17..0]
DPCLK
SDI
SDO
RDY
-PR
TP5500 DP10O1
TP5501 DP11O1
TP5502 DP12O1
TP5503 DP13O1
TP5504 DP14O1
TP5505 DP15O1
TP5506 DP16O1
TP5507 DP17O1
C5500 0.1uF
C5501 0.1uF
C5502 0.1uF
C5503 0.1uF
C5504 0.1uF
C5505 0.1uF
C5506 0.1uF
C5507 0.1uF
C5508 47uF
C9 47uF
U5500 1024 tap pot
U5501 1024 tap pot
U5502 1024 tap pot
U5503 1024 tap pot
U5504 1024 tap pot
U5505 1024 tap pot
U5506 1024 tap pot
U5507 1024 tap pot
O1 O2 Clk RDY SDI CS SDO RS GND WP VSS VDD T A B W
AD5231BRU10
DP10
DP11
DP12
DP13
DP14
DP15
DP16
DP17
DP[17..0]
Title: Erinyes - FPGA digital potentiometers
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 55 of 68
File: Virtex Digital potentiometers 1.sch
Normand Leclerc, B.Ing.
École de technologie supérieure
Hyperchip



Title: *Erinyes - Virtex programming test connector*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *56* of *68*		*École de technologie supérieure*
File: *Virtex programming test connector.sch*			*Hyperchip*



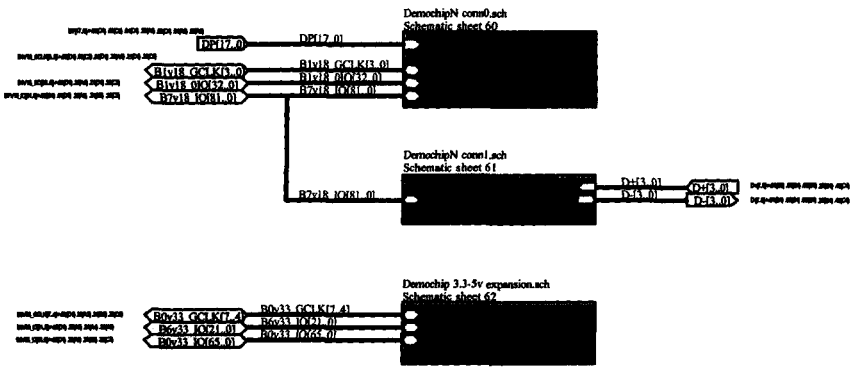
Title: *Eringyes - Virtex SRAM test connector*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *57* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex SRAM test connector.sch*			



Title: *Erinyes - Digital potentiometers and temperature acquisition test connector*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *58* of *68*		*École de technologie supérieure Hyperchip*
File: *Virtex DP and Temp test connector.sch*			

D+x and D-x must be impedance controlled as they are current based.

QStrip connector image

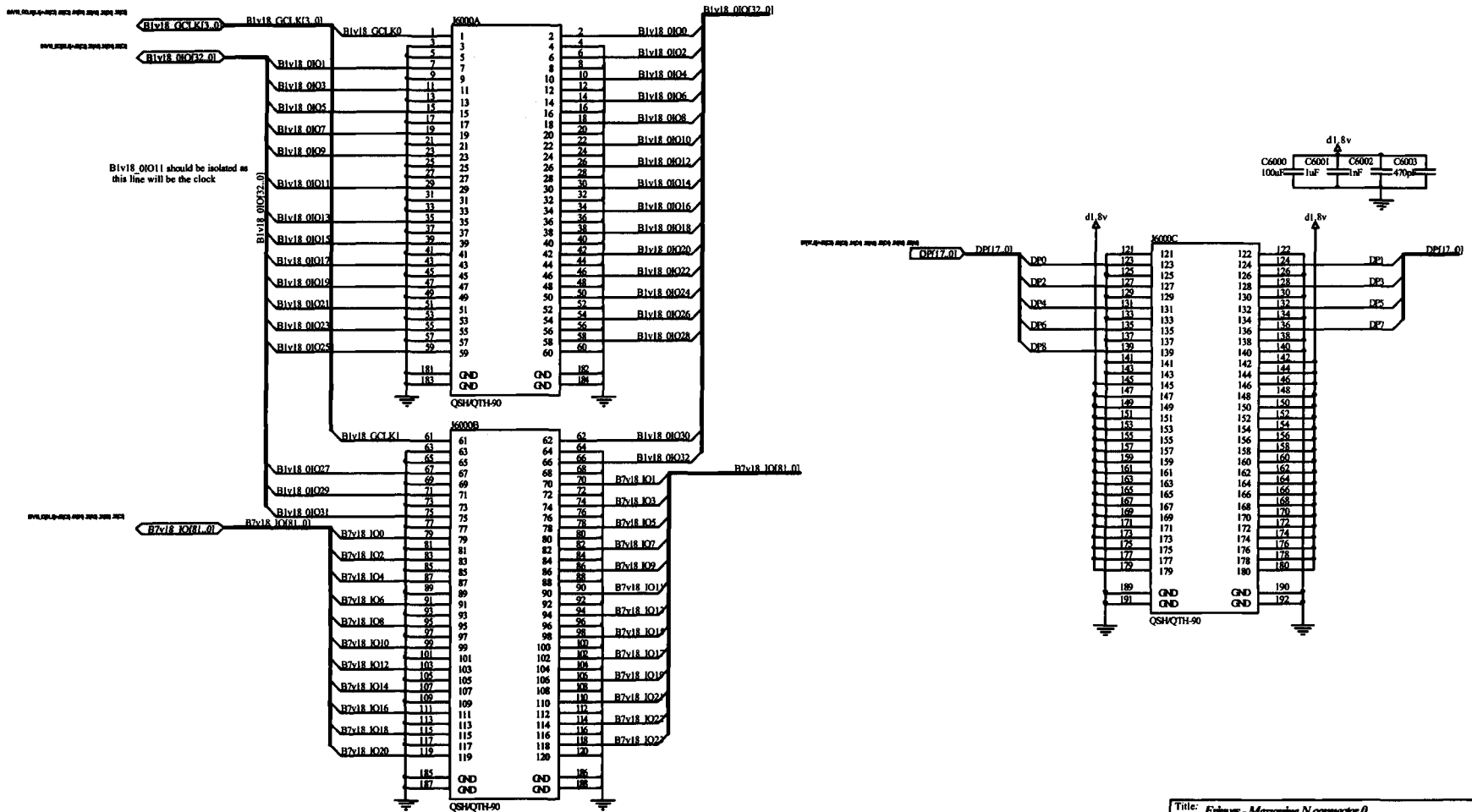


Title: *Erinyes - Mezzanine N connectors*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *59* of *68*		*École de technologie supérieure Hyperchip*
File: *DemochipN connectors.sch*			

All data lines will be switched at speeds of 125MHz or more

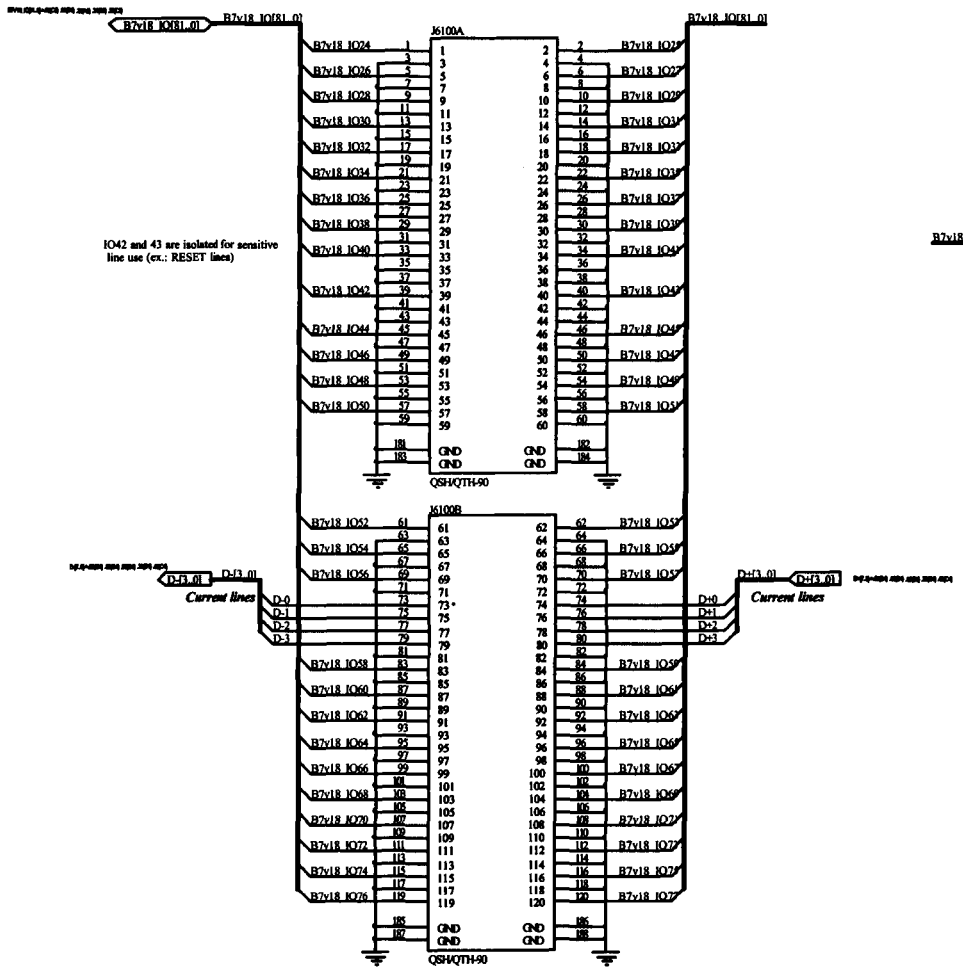
Extra care should be taken for B1v18_GCLK signals as these lines will be clock lines

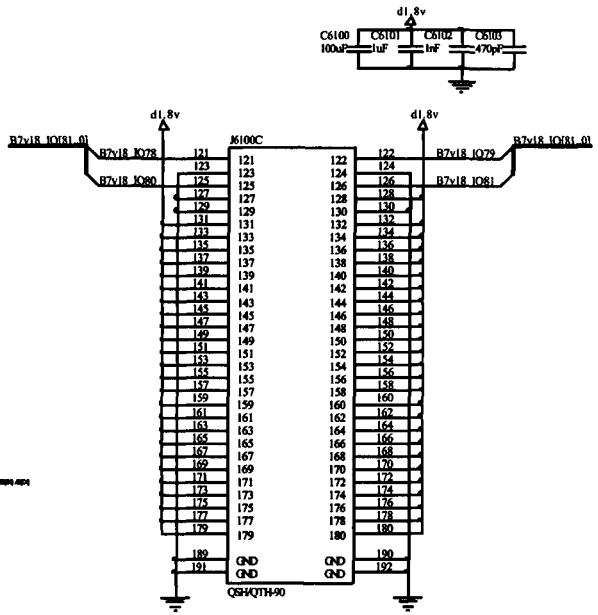
B1v18_0IO11 should be isolated as this line will be the clock



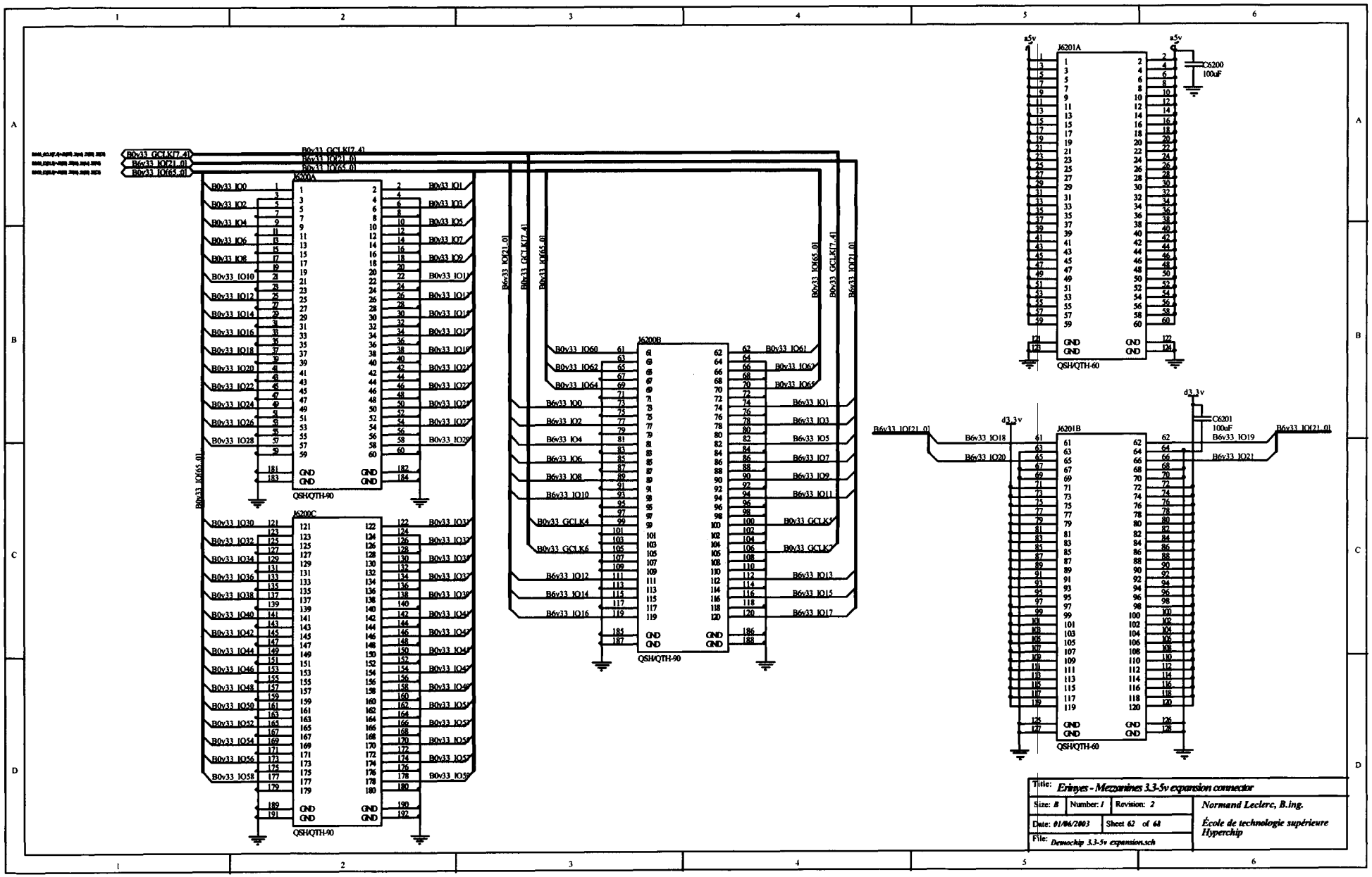
Title: *Eriyes - Mezzanine N connector 0*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *60* of *68*		*École de technologie supérieure Hyperchip*
File: *Demochip/N conn0.sch*			

D+x and D-x must be impedance controlled as they are current based.
All data lines will be switched at speeds of 125MHz or more



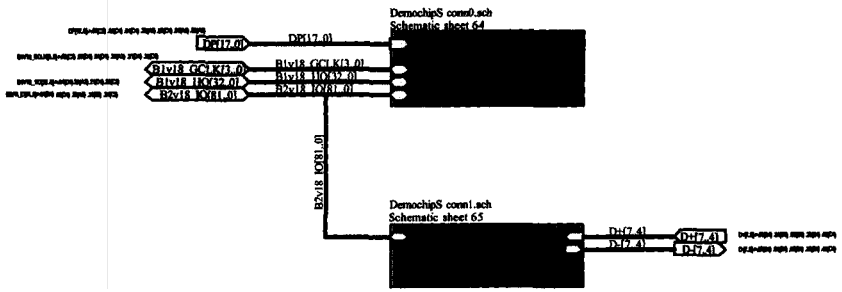


Title: *Entrées - Mezzanine N connector 1*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *61* of *68*		*École de technologie supérieure Hyperchip*
File: *DemochipN conn1.sch*			

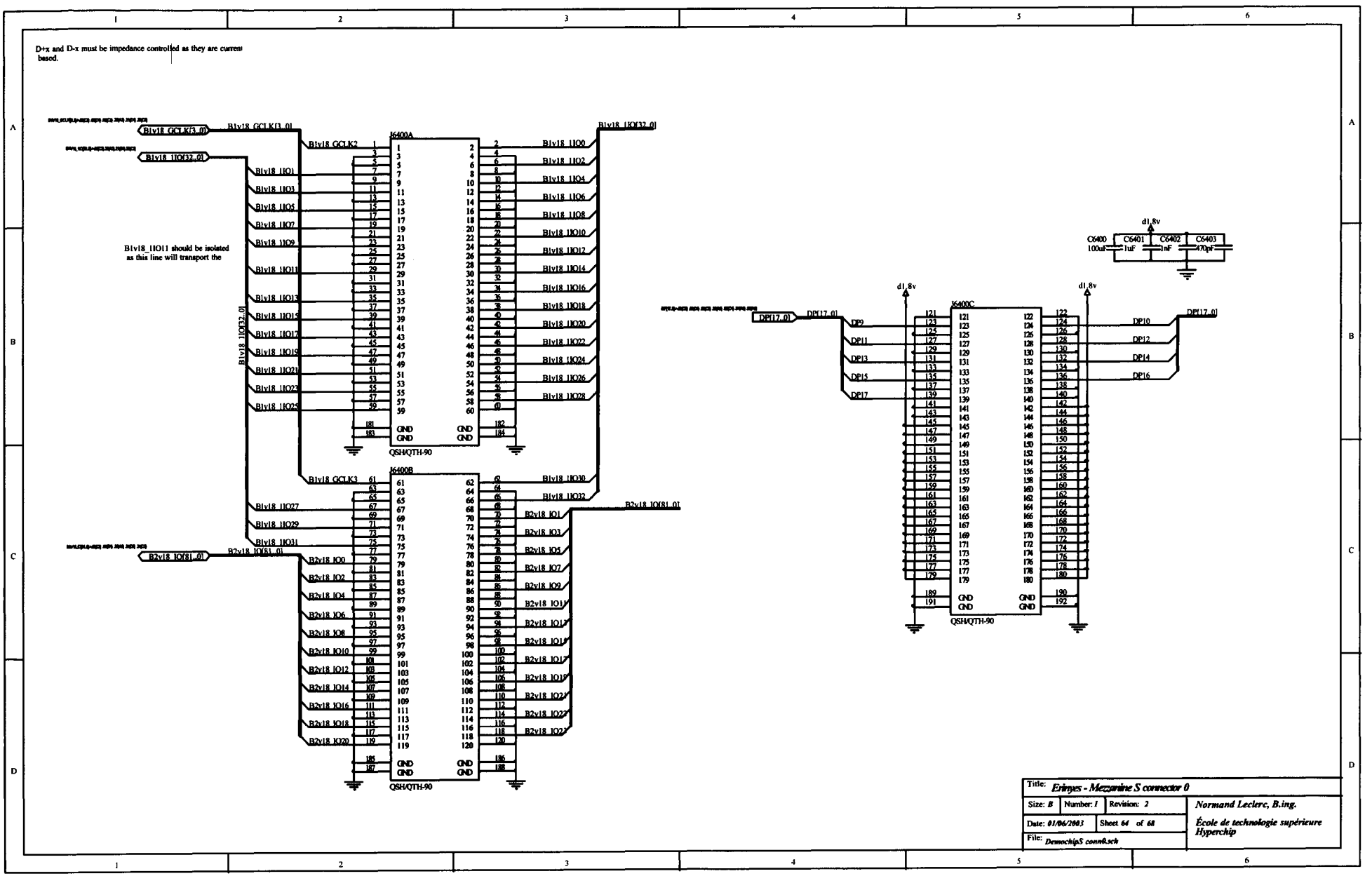

B0v33_GCLKI7..4]
B6v33_IOI21..0]
B0v33_IOI65..0]
J6200A
QSH/QTH-90
J6200B
QSH/QTH-90
J6200C
QSH/QTH-90
J6201A
QSH/QTH-60
J6201B
QSH/QTH-60
a5v
d3.3v
C6200
100uF
C6201
100uF
GND
Title: Eriyes - Mezzanines 3.3-5v expansion connector
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 62 of 68
File: Demochip 3.3-5v expansion.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

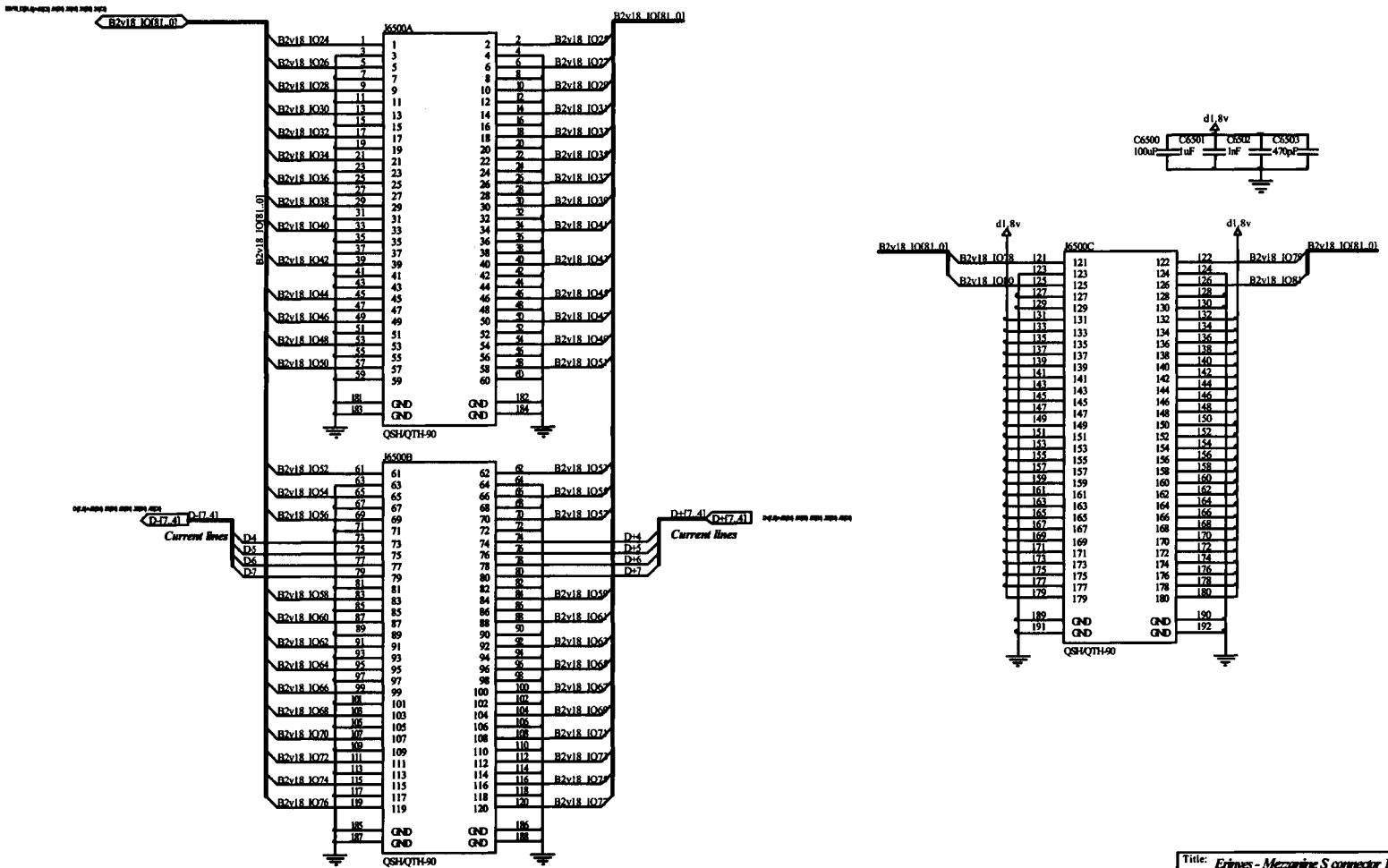
D+x and D-x must be impedance controlled as they are current based.

QStrip connector image

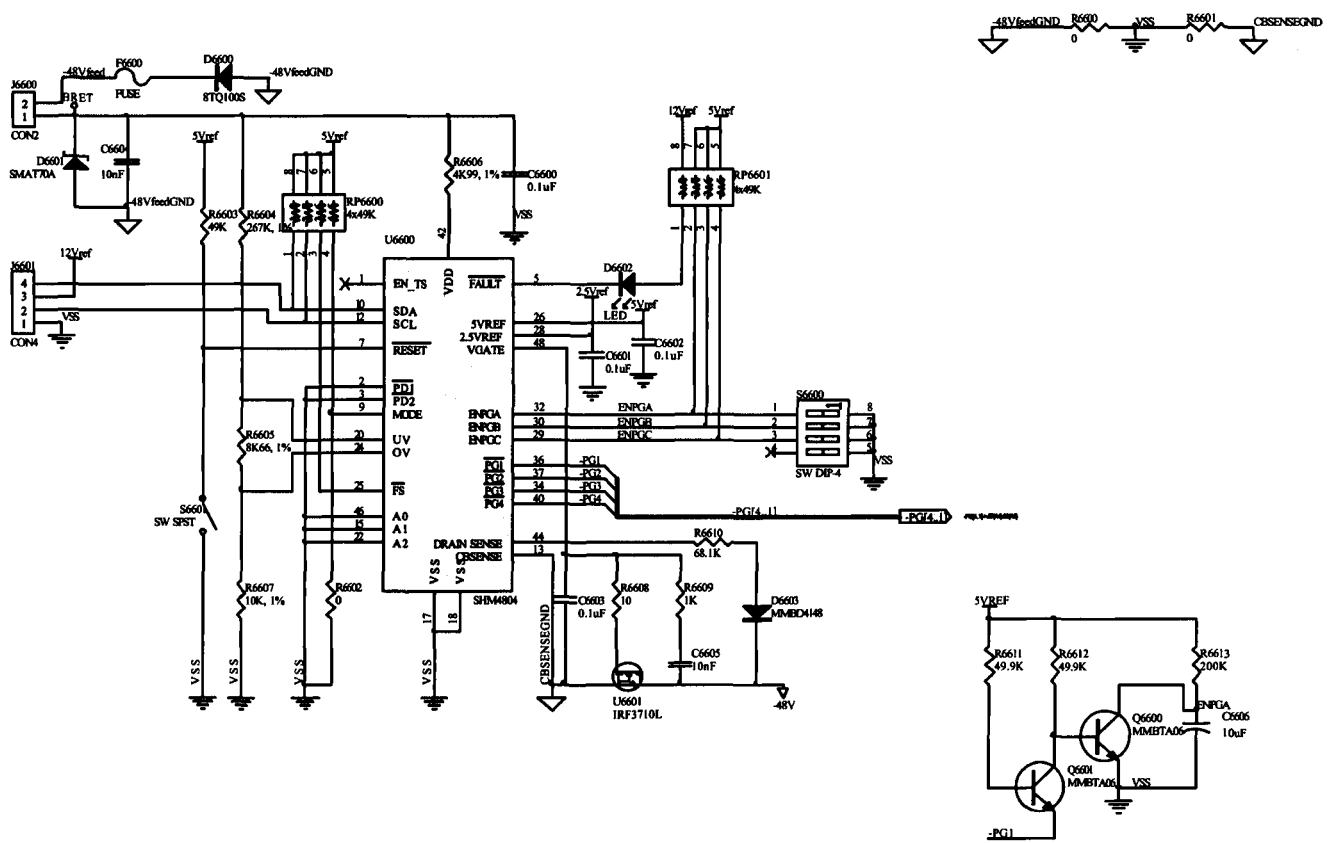


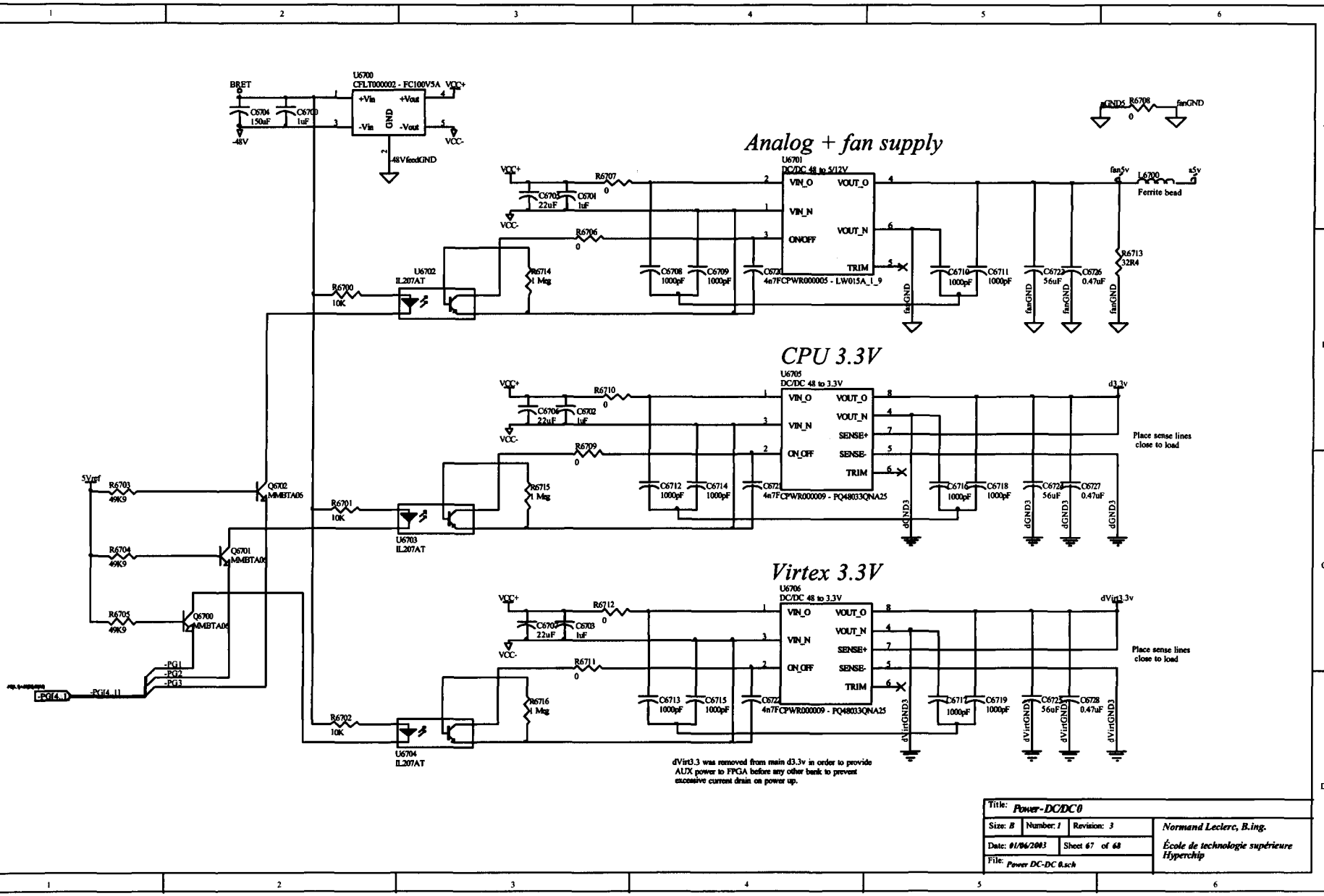
Title: *Erinyes - Mezzanine S connectors*			
Size: *B*	Number: *1*	Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet *63* of *68*		*École de technologie supérieure Hyperchip*
File: *DemochipS connectors.sch*			

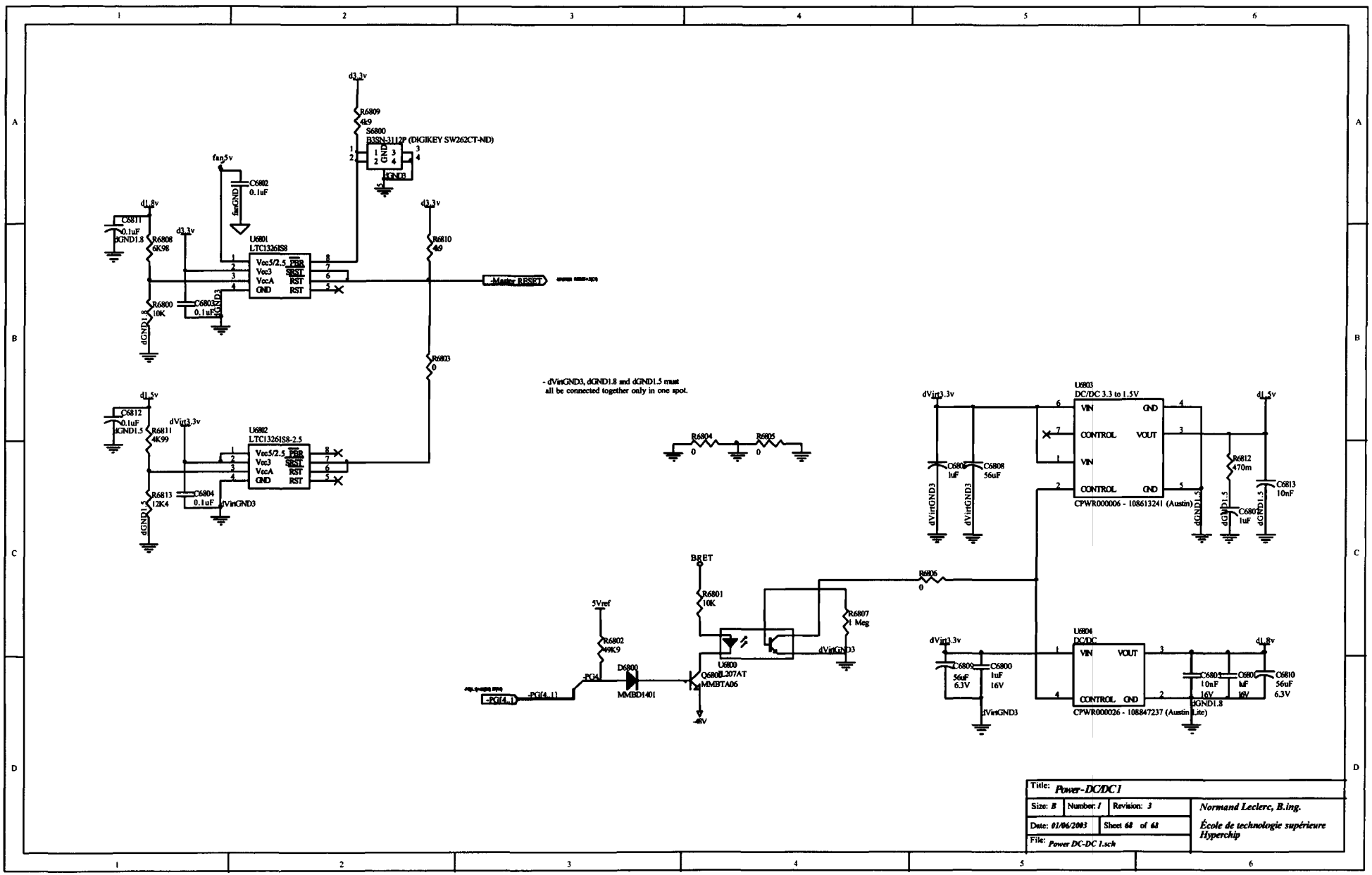

D+x and D-x must be impedance controlled as they are current based.
B1v18_IIO11 should be isolated as this line will transport the
B1v18_GCLK[3..0]
B1v18_IIO[32..0]
B2v18_IO[81..0]
J6400A
J6400B
J6400C
QSH/QTH-90
d1.8v
C6400 100uF
C6401 1uF
C6402 1nF
C6403 470pF
DP17..0
Title: Erinyes - Mezzanine S connector 0
Size: B
Number: 1
Revision: 2
Date: 01/06/2003
Sheet 64 of 68
File: DemochipS conn0.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

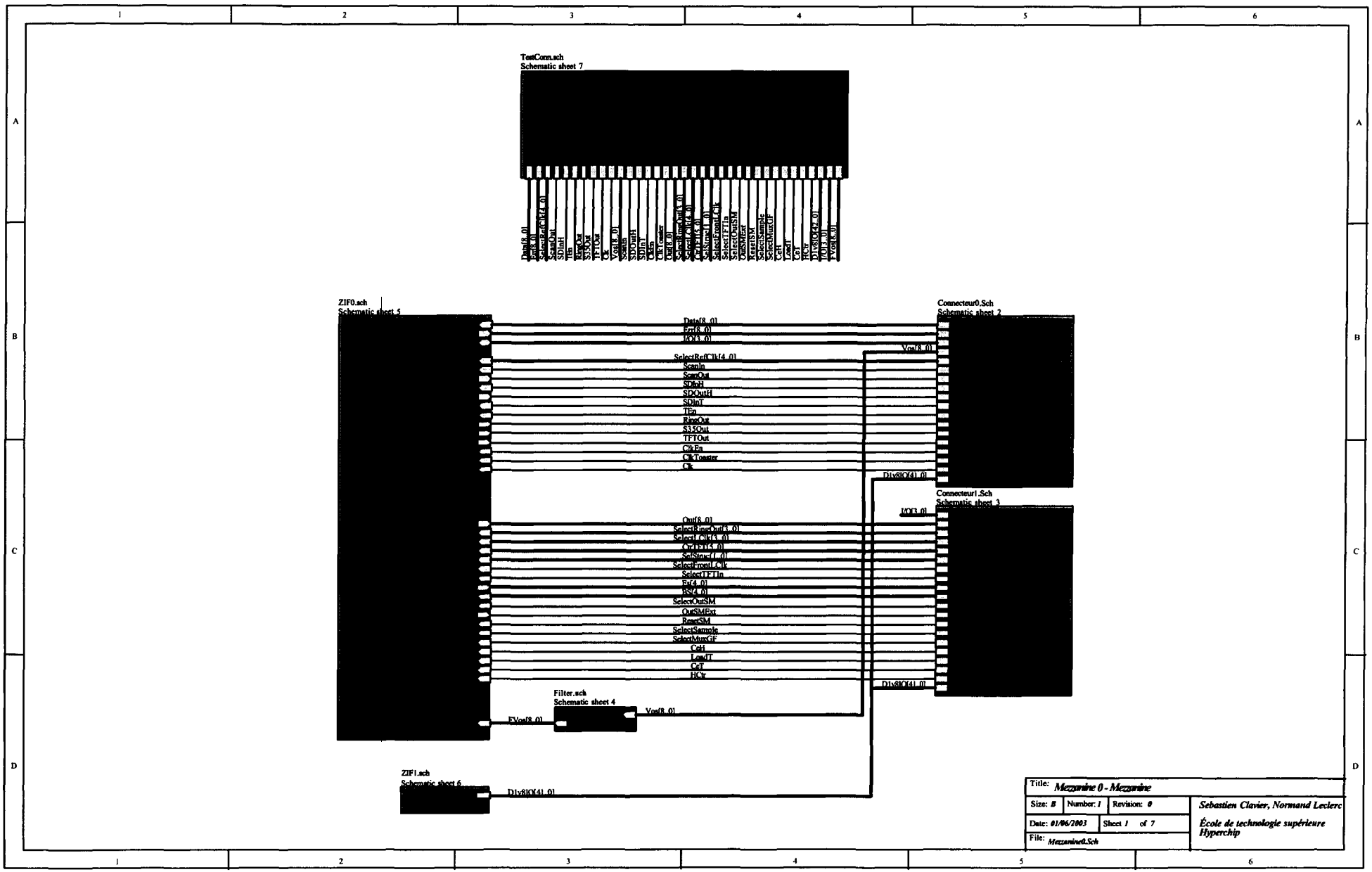


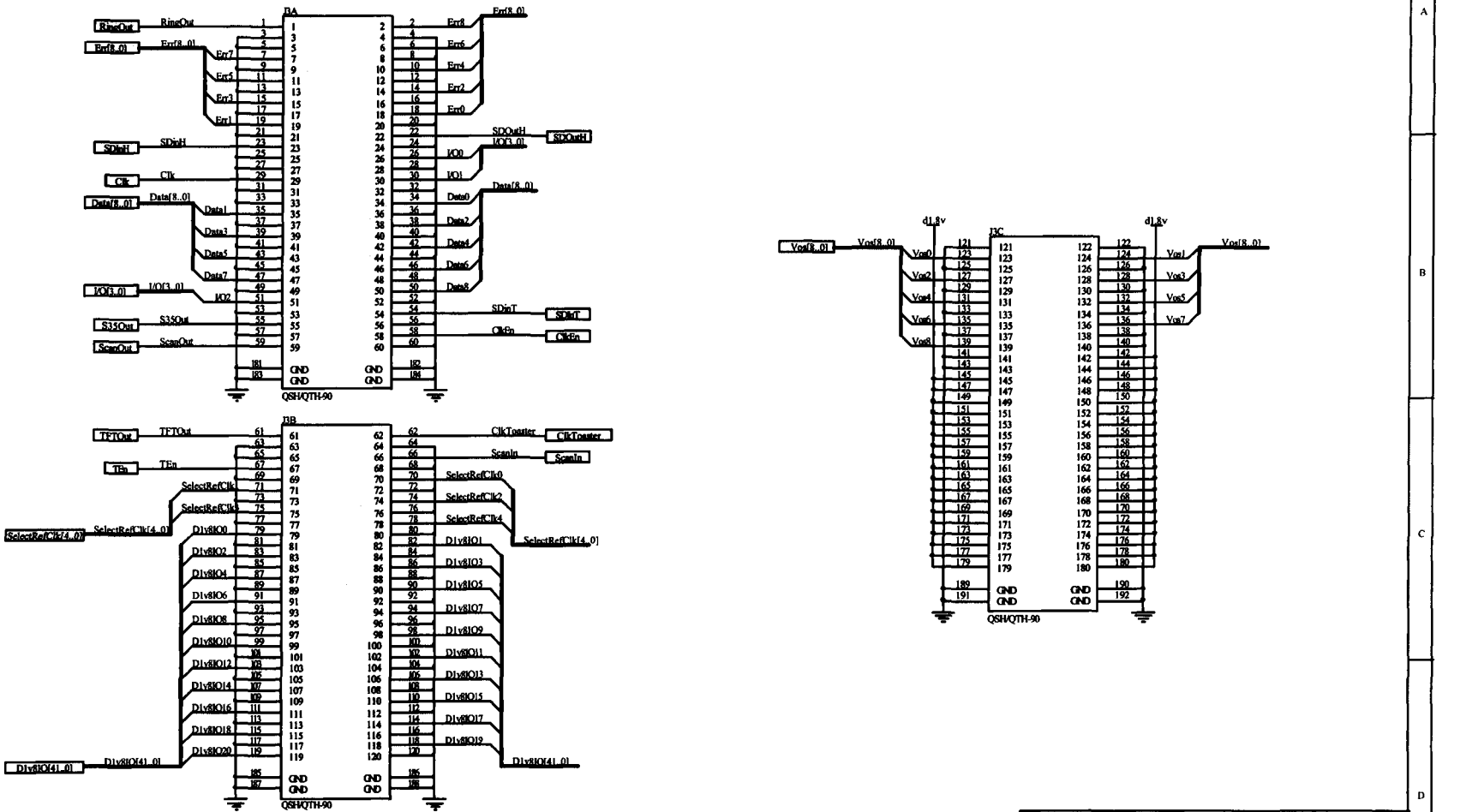
Title: *Erinyes - Mezzanine S connector 1*	
Size: *B* / Number: *1* / Revision: *2*	*Normand Leclerc, B.ing.*
Date: *01/06/2003* / Sheet *65* of *68*	*École de technologie supérieure Hyperchip*
File: *DemochipS conn1.sch*	


-48VfeedGND
R6600
0
VSS
R6601
0
CBSENSEGND
J6600
CON2
-48Vfeed
F6600
FUSE
D6600
8TQ100S
-48VfeedGND
BRET
D6601
SMAT70A
C6600
10nF
-48VfeedGND
5Vref
R6603
49K
R6604
267K, 1%
RP6600
4x49K
5Vref
R6606
4K99, 1%
C6600
0.1uF
VSS
12Vref
5Vref
RP6601
4x49K
J6601
CON4
12Vref
VSS
U6600
42
EN_TS
SDA
SCL
RESET
PD1
PD2
MODE
UV
OV
FS
A0
A1
A2
VDD
FAULT
5VREF
2.5VREF
VGATE
DRAIN SENSE
CBSENSE
VSS
VSS
SHM4804
D6602
2.5Vref
LED
5Vref
C6601
0.1uF
C6602
0.1uF
S6600
SW DIP-4
ENPGA
ENPGB
ENPGC
VSS
-PG1
-PG2
-PG3
-PG4
-PG[4..1]
S6601
SW SPST
R6605
8K66, 1%
R6607
10K, 1%
R6602
0
R6610
68.1K
C6603
0.1uF
R6608
10
R6609
1K
D6603
MMBD4148
C6605
10nF
U6601
IRF3710L
-48V
CBSENSEGND
5VREF
R6611
49.9K
R6612
49.9K
R6613
200K
Q6600
MMBTA06
ENPGA
C6606
10uF
Q6601
MMBTA06
VSS
-PG1
Title: Power - Hotswap
Size: B
Number: 1
Revision: 3
Date: 01/06/2003
Sheet 66 of 68
File: Power Hotswap.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

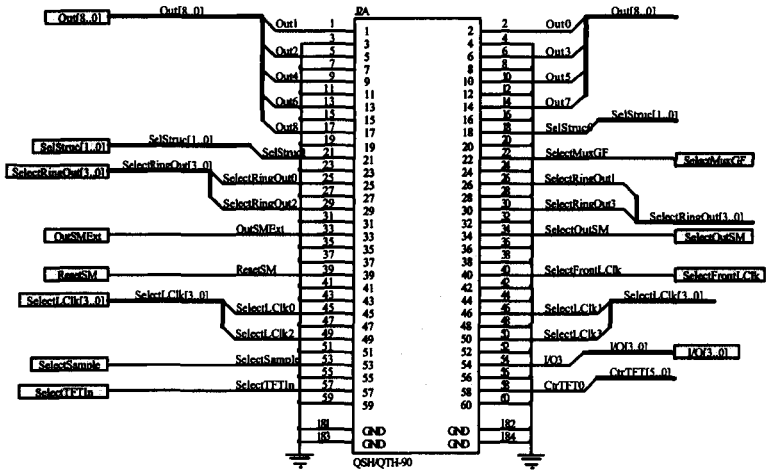

Analog + fan supply
CPU 3.3V
Virtex 3.3V
U6700
CFLT000002 - FC100V5A
BRET
-48V
48VfeedGND
VCC+
VCC-
U6701
DC/DC 48 to 5/12V
VIN_O
VIN_N
ON/OFF
VOUT_O
VOUT_N
TRIM
fan5v
L6700
Ferrite bead
R6707
R6708
fanGND
R6713
32R4
U6702
IL207AT
R6700
10K
R6714
1 Meg
U6705
DC/DC 48 to 3.3V
d3.3v
SENSE+
SENSE-
ON_OFF
R6710
R6709
R6715
U6703
R6701
dGND3
Place sense lines close to load
5Vref
R6703
49K9
R6704
R6705
Q6702
MMBTA06
Q6701
Q6700
-PG1
-PG2
-PG3
-PG[4..1]
U6706
dVirt3.3v
R6712
R6711
R6716
U6704
R6702
dVirtGND3
dVirt3.3v was removed from main d3.3v in order to provide AUX power to FPGA before any other bank to prevent excessive current drain on power up.
Title: Power-DC/DC0
Size: B
Number: 1
Revision: 3
Date: 01/06/2003
Sheet 67 of 68
File: Power DC-DC 0.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

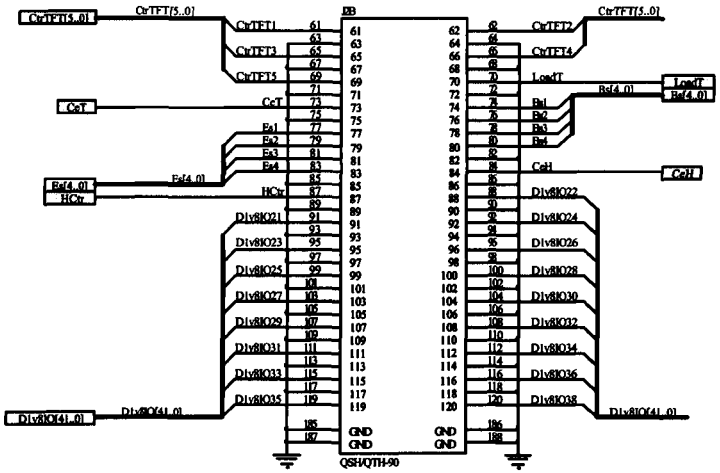

d3.3v
R6809
4k9
S6800
B3SN-3112P (DIGIKEY SW262CT-ND)
fan5v
C6802
0.1uF
fanGND
dGND3
d1.8v
C6811
0.1uF
dGND1.8
R6808
6K98
U6801
LTC1326IS8
Vcc5/2.5
Vcc3
VccA
GND
PBR
SRST
RST
RST
R6810
4k9
Master RESET
R6800
10K
C6803
0.1uF
R6803
0
d1.5v
C6812
0.1uF
dGND1.5
R6811
4K99
dVirt3.3v
U6802
LTC1326IS8-2.5
R6813
12K4
C6804
0.1uF
dVirtGND3
- dVirtGND3, dGND1.8 and dGND1.5 must all be connected together only in one spot.
R6804
0
R6805
0
BRET
R6801
10K
5Vref
R6802
49K9
PG4
PG[4..1]
D6800
MMBD1401
U6809
L207AT
Q6800
MMBTA06
R6807
1 Meg
dVirtGND3
-48V
R6806
0
U6803
DC/DC 3.3 to 1.5V
VIN
GND
CONTROL
VOUT
CPWR000006 - 108613241 (Austin)
C6807
1uF
C6808
56uF
R6812
470m
C6813
10nF
C6801
1uF
U6804
DC/DC
CPWR000026 - 108847237 (Austin Lite)
C6805
56uF
6.3V
C6800
1uF
16V
C6809
10nF
16V
C6802
1uF
16V
C6810
56uF
6.3V
Title: Power-DC/DC1
Size: B
Number: 1
Revision: 3
Date: 01/06/2003
Sheet 68 of 68
File: Power DC-DC 1.sch
Normand Leclerc, B.ing.
École de technologie supérieure
Hyperchip

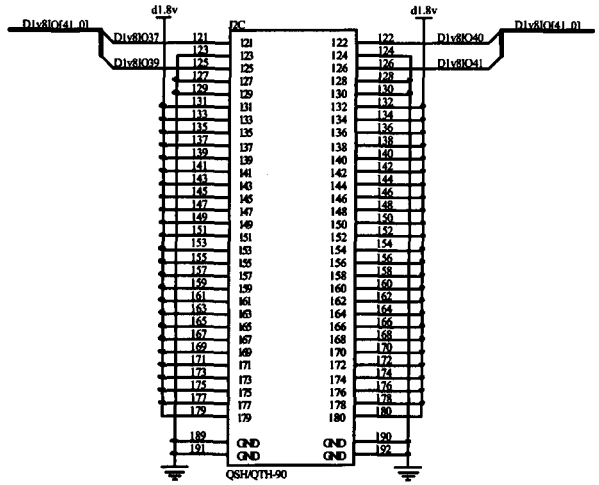

TestConn.sch
Schematic sheet 7
ZIF0.sch
Schematic sheet 5
Connecteur0.Sch
Schematic sheet 2
Connecteur1.Sch
Schematic sheet 3
Filter.sch
Schematic sheet 4
ZIF1.sch
Schematic sheet 6
Data[8..0]
Err[8..0]
IO[3..0]
Vos[8..0]
SelectRefClk[4..0]
ScanIn
ScanOut
SDInH
SDOutH
SDInT
TEn
RingOut
S35Out
TFTOut
ClkEn
ClkToggle
Clk
D1v8IO[41..0]
Out[8..0]
SelectRingOut[3..0]
SelectClk[1..0]
SelectEn[5..0]
SelStageCL[1..0]
SelectFromLClk
SelectTFTIn
Ref[4..0]
BSP[4..0]
SelectOutSM
OutSMExt
ResetSM
SelectSample
SelectMuxGF
CeH
LoadT
CeT
HClr
EVos[8..0]
Title: Mezzanine 0 - Mezzanine
Size: B
Number: 1
Revision: 0
Date: 01/06/2003
Sheet 1 of 7
File: Mezzanine0.Sch
Sebastien Clavier, Normand Leclerc
École de technologie supérieure
Hyperchip



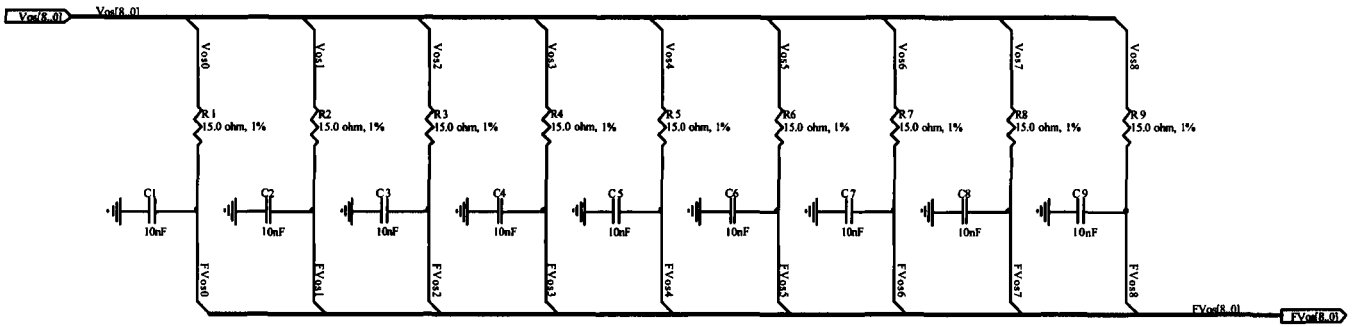
Title: *Mezzanine 0 - Connector 0*			
Size: *B*	Number: *1*	Revision: *1*	*Sebastien Clavier, Normand Leclerc*
Date: *01/06/2003*	Sheet *2* of *7*		*École de technologie supérieure Hyperchip*
File: *Connecteur0.Sch*			



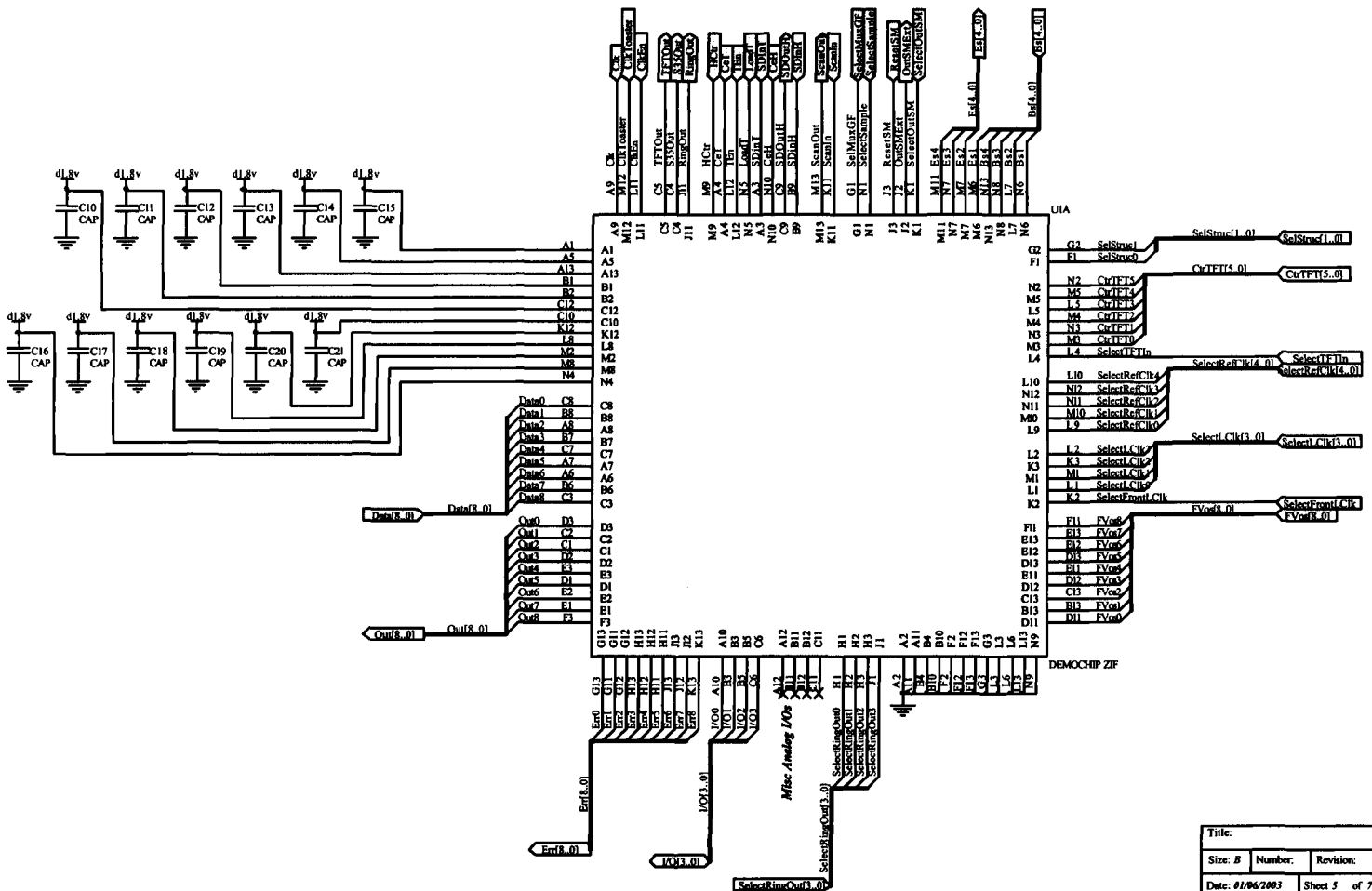




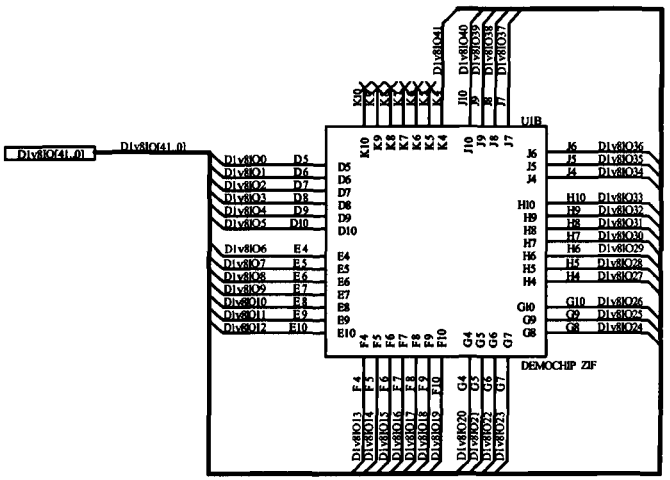
Title: *Mezzanine 0 - Connecteur 1*			
Size: *B*	Number: *1*	Revision: *0*	*Sebastien Clavier, Normand Leclerc*
Date: *01/06/2003*	Sheet *3* of *7*		*École de technologie supérieure Hyperchip*
File: *Connecteur1.Sch*			



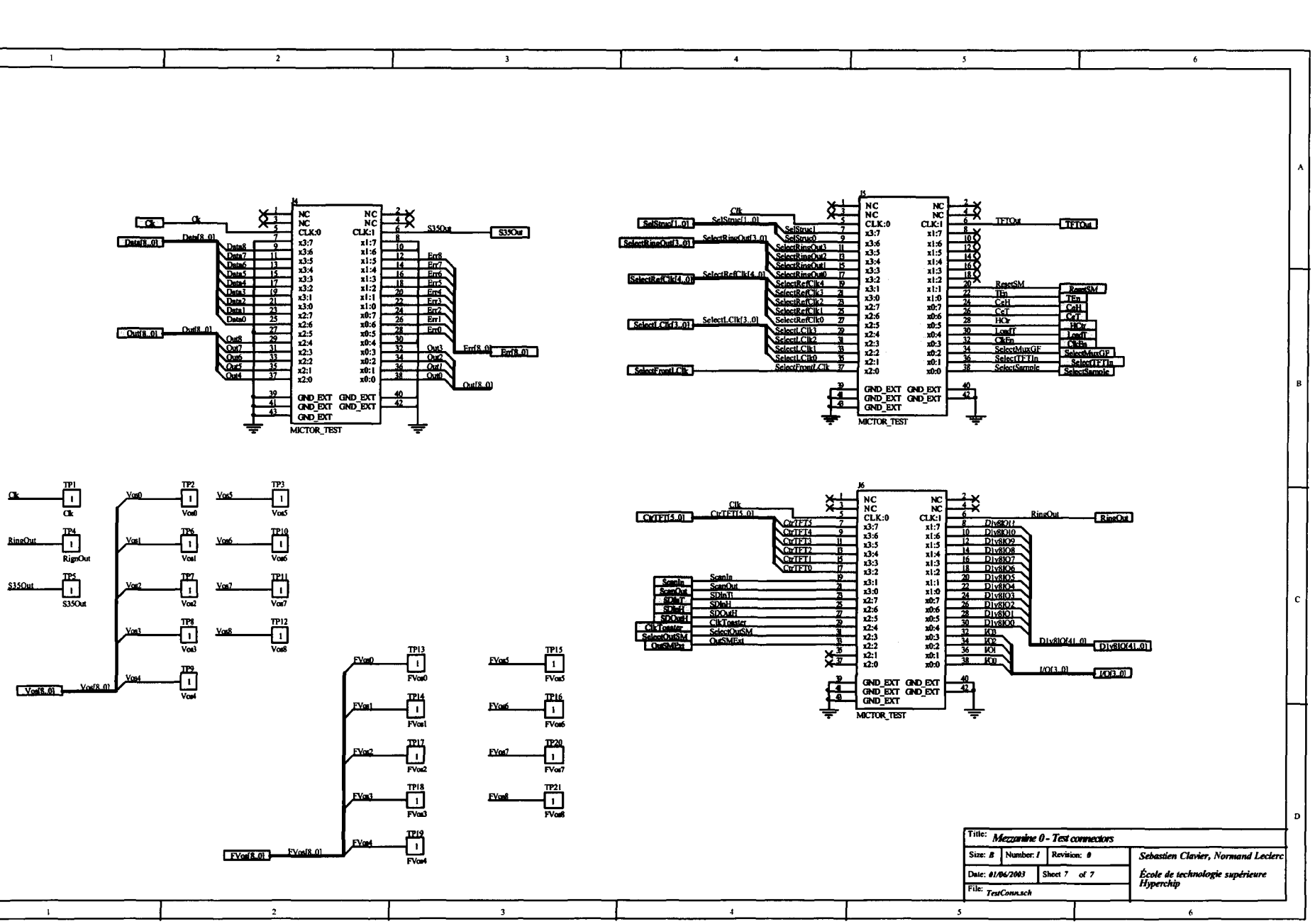
Title: *Mezzanine 0 - VOS Filters*			
Size: *B*	Number: *1*	Revision: *0*	*Sebastien Clavier, Normand Leclerc*
Date: *01/06/2003*	Sheet *4* of *7*		*École de technologie supérieure Hyperchip*
File: *Filter.sch*			



Title:			
Size: B	Number:	Revision:	Normand Leclerc, B.ing.
Date: 01/06/2003	Sheet 5 of 7		École de technologie supérieure Hyperchip
File: ZIF8.sch			



Title:			
Size: B	Number:	Revision:	*Normand Leclerc, B.ing.*
Date: *01/06/2003*	Sheet 6 of 7		*École de technologie supérieure Hyperchip*
File: *ZIF1.sch*			


J4
MICTOR_TEST
J5
MICTOR_TEST
J6
MICTOR_TEST
TP1
TP2
TP3
TP4
TP5
TP6
TP7
TP8
TP9
TP10
TP11
TP12
TP13
TP14
TP15
TP16
TP17
TP18
TP19
TP20
TP21
Title: Mezzanine 0 - Test connectors
Size: B
Number: 1
Revision: 0
Date: 01/06/2003
Sheet 7 of 7
File: TestConn.sch
Sebastien Clavier, Normand Leclerc
École de technologie supérieure
Hyperchip

ANNEXE 6

Répartition des couches
de la carte de circuits imprimés

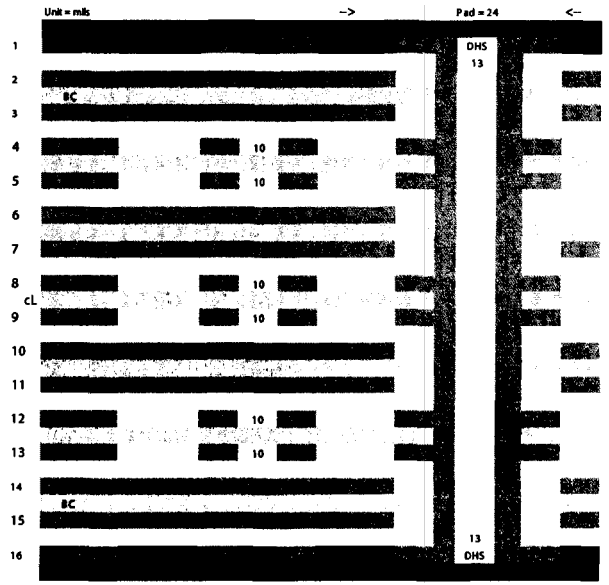
	VERSION:	3	BOARD THICK. SPEC.
	AUTHOR:	PFrancq	DIMENSIONS :
	PROJECT :	Erinyes	Cu Thickness (mils) :
v5	MATERIAL :	FR406 unless otherwise specified.	
	Dk :	see below (Eff. Dk)	

HYPERCHIP

LY #	Assignment	Book	Cu Weight	Nom. Mat. Th.	Cu Th.	Nom. Dk	Eff. Dk	Comments	Wtrace (mils) ART WORK SE W	Wtrace (mils) ART WORK Diff W	Wtrace (mils) ART WORK Diff S	Wtrace (mils) ART WORK Diff W	Zo (Ohms) Spec. SE	Zo (Ohms) Spec. Diff	Zo (Ohms) Calc. SE	Zo (Ohms) Calc. Diff	Zo (Ohms) Tol.
				0.6	1.9												
L 1	Pads		h		0.6	3.6		6	6 /	12	/ 6		50	100			+/- 10%
		P		4													
L 2	3V3	PLN	1		1.2												
		C		4													
L 3	GND	PLN	1		1.2												
		P		4													
L 4			h		0.6	3.55		5	5 /	10	/ 5		50	100			+/- 10%
		C		4													
L 5			h		0.6	3.55		5	5 /	10	/ 5		50	100			+/- 10%
		P		4													
L 6	1V5	PLN	1		1.2												
		C		2													
L 7	GND	PLN	1		1.2												
		P		4													
L 8			h		0.6	3.55		5	5 /	10	/ 5		50	100			+/- 10%
	cL	C		4													
L 9			h		0.6	3.55		5	5 /	10	/ 5		50	100			+/- 10%
		P		4													
L 10	GND	PLN	1		1.2												
		C		2													
L 11	1V8	PLN	1		1.2												
		P		4													
L 12			h		0.6	3.55		5	5 /	10	/ 5		50	100			+/- 10%
		C		4													
L 13			h		0.6	3.55		5	5 /	10	/ 5		50	100			+/- 10%
		P		4													
L 14	GND	PLN	1		1.2												
		C		4													
L 15	3V3, 5V	PLN	1		1.2												
		P		4													
L 16			h		0.6	3.6		6	6 /	12	/ 6		50	100			+/- 10%
				0.6	1.9												
				57.2	18.2												

After Lamination 70.40

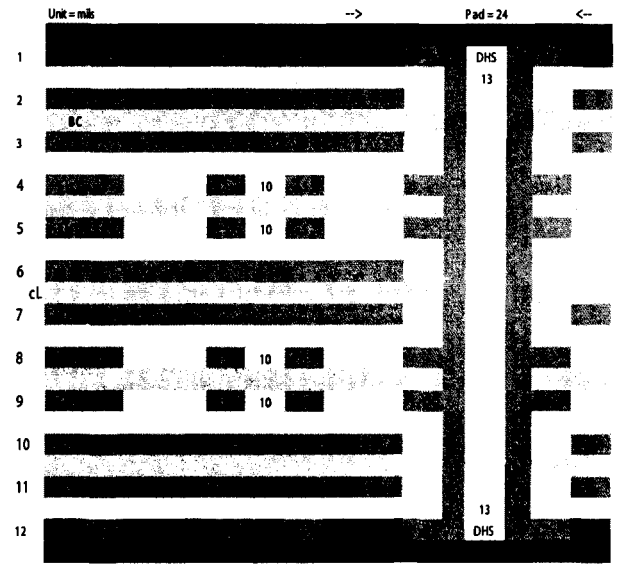
After Plating 75.40



Comments on Stack up

Revisions				
Version #		Date	Peop	Comments

LY #	Assignment	Book	Cu Weight	Nom. Mat. Th.	Cu Th.	Nom. Dk	Eff. Dk	Comments	Wtrace (mils) ART WORK				Zo (Ohms) Spec.		Calc.		Tol.
									SE	Diff			SE	Diff	SE	Diff	
									W	W	S	W					
				0.6													
					1.9												
L 1	Pads		h		0.6	3.6			6	6 /	12	/ 6	50	100			+/- 10%
		P		4													
L 2	1V8	PLN	1		1.2												
		C		4													
L 3	GND	PLN	1		1.2												
		P		4													
L 4			h		0.6	3.55			5	5 /	10	/ 5	50	100			+/- 10%
		C		4													
L 5			h		0.6	3.55			5	5 /	10	/ 5	50	100			+/- 10%
		P		4													
L 6	1V8	PLN	1		1.2												
	cL	C		2													
L 7	GND	PLN	1		1.2												
		P		4													
L 8			h		0.6	3.55			5	5 /	10	/ 5	50	100			+/- 10%
		C		4													
L 9			h		0.6	3.55			5	5 /	10	/ 5	50	100			+/- 10%
		P		4													
L 10	GND	PLN	1		1.2												
		C		2													
L 11	1V8	PLN	1		1.2												
		P		4													
L 12			h		0.6	3.6			6	6 /	12	/ 6	50	100			+/- 10%
					1.9												
				0.6													
				41.2	14.6												
After Lamination 50.80																	
After Plating 55.80																	



Comments on Stack up

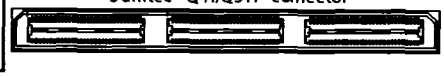
Revisions					
Version #			Date	Peop	Comments

ANNEXE 7

Répartition des signaux
sur les connecteurs de mezzanines

Connectors are Samtec QTH/QSH type
Virtex has 8 banks, each has it's own power domain
Connectors integral ground plane for connector 0 and 1 are connected to VSS
1v8 is a [illegible] connected to 3 banks of Virtex and mezzanines

Samtec QTH/QSH connector



Connecteurs des mezzanines

Connector pin	CONNECTOR 1	Connector pin	Connector pin	CONNECTOR	Connector pin
1		2	1		2
3		4	3		4
5		6	5		6
7		8	7		8
9		10	9		10
11		12	11		12
13		14	13		14
15		16	15		16
17		18	17		18
19		20	19		20
21		22	21		22
23		24	23		24
25		26	25		26
27		28	27		28
29		30	29		30
31		32	31		32
33		34	33		34
35		36	35		36
37		38	37		38
39		40	39		40
41		42	41		42
43		44	43		44
45		46	45		46
47		48	47		48
49		50	49		50
51		52	51		52
53		54	53		54
55		56	55		56
57		58	57		58
59		60	59		60
61		62	61		62
63		64	63		64
65		66	65		66
67		68	67		68
69		70	69		70
71		72	71		72
73		74	73	Temperature diode 0, cathode / Temperature diode 0, anode	74
75		76	75	Temperature diode 1, cathode / Temperature diode 1, anode	76
77		78	77	Temperature diode 2, cathode / Temperature diode 2, anode	78
79		80	79	Temperature diode 3, cathode / Temperature diode 3, anode	80
81		82	81		82
83		84	83		84
85		86	85		86
87		88	87		88
89		90	89		90
91		92	91		92
93		94	93		94
95		96	95		96
97		98	97		98
99		100	99		100
101		102	101		102
103		104	103		104
105		106	105		106
107		108	107		108
109		110	109		110
111		112	111		112
113		114	113		114
115		116	115		116
117		118	117		118
119		120	119		120
121		122	121		122
123		124	123		124
125		126	125		126
127		128	127		128
129		130	129		130
131		132	131		132
133		134	133		134
135		136	135		136
137		138	137		138
139		140	139		140
141		142	141		142
143		144	143		144
145		146	145		146
147		148	147		148
149		150	149		150
151		152	151		152
153		154	153		154
155		156	155		156
157		158	157		158
159		160	159		160
161		162	161		162
163		164	163		164
165		166	165		166
167		168	167		168
169		170	169		170
171		172	171		172
173		174	173		174
175		176	175		176
177		178	177		178
179		180	179		180
181		182	181		182
183		184	183		184
185		186	185		186
187		188	187		188
189		190	189		190
191		192	191		192

Implémentation du testbed
Connecteurs des mezzanines

Connectors are Samtec QTH/QSH type
Virtex has 8 banks, each has it's own power domain
Connectors integral ground plane for connector 2 and 3 are connected to VSS

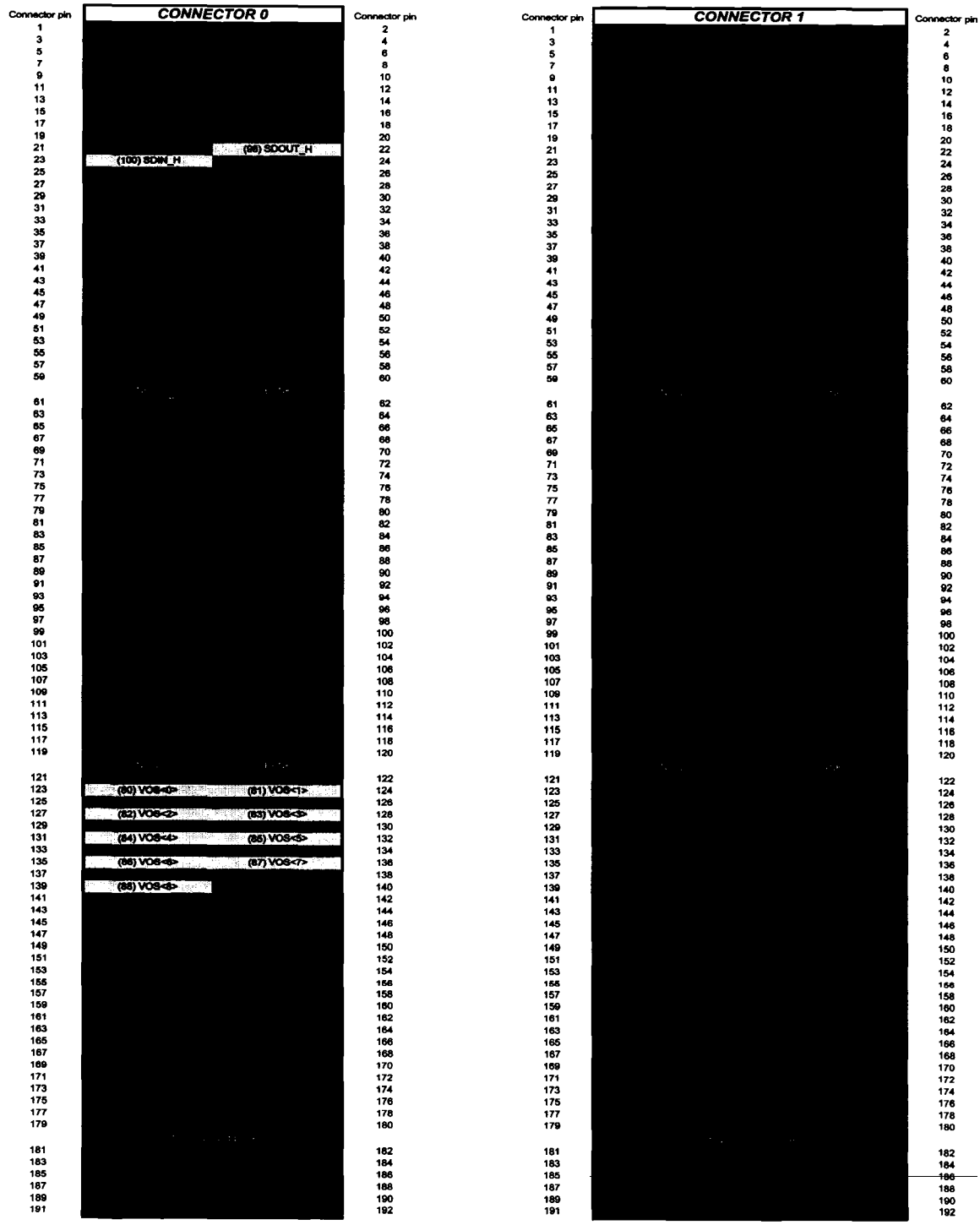
Samtec QTH/QSH connector

Connector pin	CONNECTOR 2	Connector pin	Connector pin	CONNECTOR 3	Connector pin
1		2	1		2
3		4	3		4
5		6	5		6
7		8	7		8
9		10	9		10
11		12	11		12
13		14	13		14
15		16	15		16
17		18	17		18
19		20	19		20
21		22	21		22
23		24	23		24
25		26	25		26
27		28	27		28
29		30	29		30
31		32	31		32
33		34	33		34
35		36	35		36
37		38	37		38
39		40	39		40
41		42	41		42
43		44	43		44
45		46	45		46
47		48	47		48
49		50	49		50
51		52	51		52
53		54	53		54
55		56	55		56
57		58	57		58
59		60	59		60
61		62	61		62
63		64	63		64
65		66	65		66
67		68	67		68
69		70	69		70
71		72	71		72
73		74	73		74
75		76	75		76
77		78	77		78
79		80	79		80
81		82	81		82
83		84	83		84
85		86	85		86
87		88	87		88
89		90	89		90
91		92	91		92
93		94	93		94
95		96	95		96
97		98	97		98
99		100	99		100
101		102	101		102
103		104	103		104
105		106	105		106
107		108	107		108
109		110	109		110
111		112	111		112
113		114	113		114
115		116	115		116
117		118	117		118
119		120	119		120
121		122	121		122
123		124	123		124
125		126	125		126
127		128	127		128
129		130			
131		132			
133		134			
135		136			
137		138			
139		140			
141		142			
143		144			
145		146			
147		148			
149		150			
151		152			
153		154			
155		156			
157		158			
159		160			
161		162			
163		164			
165		166			
167		168			
169		170			
171		172			
173		174			
175		176			
177		178			
179		180			
181		182			
183		184			
185		186			
187		188			
189		190			
191		192			

Implémentation du testbed
Connecteurs des mezzanines

Connectors are Samtec QTH/QSH type
Virtex has 8 banks, each has it's own power domain
Connectors integral ground plane for connector 0 and 1 are connected to VSS

Number in parentesis are corresponding DEMOCHIP die pads. They are not CONNECTOR pins.



ANNEXE 8

Code VHDL de base
de la carte de test Erinyes

```
-- Erinyes
--
-- CPU definitions package
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

package cpu_pak is

  subtype cpuAddress_typ is std_logic_vector(22 downto 0);
  subtype cpuData_typ is std_logic_vector(31 downto 0);

  constant cpuRead_c: std_logic:= '1';
  constant cpuWrite_c: std_logic:= '0';

end package cpu_pak;
```

```
-- Erinyes
--
-- CPU definitions package
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

package body cpu_pak is


end cpu_pak;
```

```
-- Erinyes
--
-- CPU Interface: Address decoder
--
-- Purpose:
--    Address detection scheme
--
-- I/O:
--    address: input address to be compared
--    ce: output clock enable signal
--    cs: clock enable
--    clk: clock signal
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity addressDecoder is
  generic(
    constant WIDTH: integer := 24;
    constant TRIG_ADDRESS_LOW: std_logic_vector := x"0";
    constant TRIG_ADDRESS_HIGH: std_logic_vector := x"0"
  );
  port(
    address: in std_logic_vector(WIDTH-1 downto 0);
    ce: out std_logic;
    cs: in std_logic;
    clk: in std_logic
  );
end addressDecoder;
```

```
-- Erinyes
--
-- CPU Interface: Address decoder
--
-- Purpose:
--    Address detection scheme
--
-- I/O:
--    address: input address to be compared
--    ce: clock enable
--    cs: chip select
--    clk: clock signal
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture rtl of addressDecoder is

  signal active: std_logic;
  signal cmpLow: std_logic_vector(address'length downto 0);
  signal cmpHigh: std_logic_vector(cmpLow'range);
  signal cmpAddr: std_logic_vector(cmpLow'range);

begin

  singleAddressGen: if( TRIG_ADDRESS_LOW=TRIG_ADDRESS_HIGH ) generate
    active <= '1' when cs='0' and address=TRIG_ADDRESS_LOW else '0';
  end generate singleAddressGen;



  rangeAddressGen: if( TRIG_ADDRESS_LOW/=TRIG_ADDRESS_HIGH ) generate
-- comparisons substractions
    cmpAddr <= '0' & address;
    cmpLow <= cmpAddr-TRIG_ADDRESS_LOW;
    cmpHigh <= TRIG_ADDRESS_HIGH-cmpAddr;
    active <= not (cs or cmpLow(cmpLow'high) or cmpHigh(cmpHigh'high));
  end generate rangeAddressGen;


  output: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      rstIf: if( active='1' ) then
        ce <= '1';
      else
        ce <= '0';
      end if rstIf;
    end if clkIf;
  end process ;

end rtl;
```

```
-- Erinyes
--
-- CPU Interface
--
-- Purpose:
--  CPU data synchronization
--
-- I/O:
--    address: input address to be compared
--    cpuDataIn: incoming cpu data bus
--    cpuDataOut: outgoing cpu data bus
--    dataIn: interface data input
--    dataOut: interface data output
--    rd: cpu read request
--    wr: cpu write request
--    rw: cpu RW signal
--    oe: cpu output enable
--    cs: clock enable
--    rst: reset signal
--    cpuClk: cpu clock signal
--    clk: clock signal
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity cpuInt is
  generic(
    constant ADDR_WIDTH: integer := 24;
    constant CPU_DATA_WIDTH: integer := 32;
    constant DATA_WIDTH: integer := 32;
    constant TRIG_ADDRESS_HIGH: std_logic_vector := x"0";
    constant TRIG_ADDRESS_LOW: std_logic_vector := x"0"
  );
  port(
    address: in std_logic_vector(ADDR_WIDTH-1 downto 0);
    cpuDataIn: in std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
    cpuDataOut: out std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
    dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
    dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
    rd: out std_logic;
    wr: out std_logic;
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic
  );
end cpuInt;
```

```
-- Erinyes
--
-- CPU Interface
--
-- Purpose:
--  CPU data synchronization
--
-- I/O:
--    address: input address to be compared
--    cpuDataIn: incoming cpu data bus
--    cpuDataOut: outgoing cpu data bus
--    dataIn: interface data input
--    dataOut: interface data output
--    ce: clock enable output signal
--    rw: cpu RW signal
--    oe: cpu output enable
--    cs: clock enable
--    cpuClk: cpu clock signal
--    clk: clock signal
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture rtl of cpuInt is

  component addressDecoder is
    generic(
      constant WIDTH: integer := 24;
      constant TRIG_ADDRESS_LOW: std_logic_vector := x"0";
      constant TRIG_ADDRESS_HIGH: std_logic_vector := x"0"
    );
    port(
      address: in std_logic_vector(WIDTH-1 downto 0);
      ce: out std_logic;
      cs: in std_logic;
      clk: in std_logic
    );
  end component addressDecoder;

-- data width difference
  constant widthDiff_c: integer := CPU_DATA_WIDTH-DATA_WIDTH;

-- machine states
  type machStates_typ is ( idle, activated, reading, writing );
  signal state_e: machStates_typ;

-- internal signals
  signal wakedUp: std_logic;
  signal rd_w: std_logic_vector(1 downto 0);
  signal data_w: std_logic_vector(dataIn'range) := (others=>'0');
  signal sync, sync_w: std_logic;
  constant zeroPad_c: std_logic_vector(widthDiff_c-1 downto 0)
                    := (others=>'0');
begin

  ad0: addressDecoder
  generic map( ADDR_WIDTH, TRIG_ADDRESS_HIGH,
               TRIG_ADDRESS_LOW )
  port map( address, wakedUp, cs, cpuClk );

  machine: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      rstIf: if( rst='1' ) then
        state_e <= idle;
      else -- rstIf
```

```
      machCase: case state_e is
        when idle =>
          actIf: if( wakedUp='1' ) then
            state_e <= activated;
          end if actIf;
        when activated =>
          readWriteIf: if( oe='0' ) then
            state_e <= reading;
          elsif( rw='0' ) then -- readWriteIf
            state_e <= writing;
          end if readWriteIf;
        when others =>
          idleIf: if( oe='1' and rw='1' ) then
            getIdle: if( wakedUp='0' ) then
              state_e <= idle;
            end if getIdle;
          end if idleIf;
      end case machCase;
    end if rstIf;
  end if clkIf;
end process machine;

read: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    readIf: if( state_e=reading and oe='0' ) then
      rd_w(0) <= '1';
      cpuDataOut <= zeroPad_c & dataIn;
    else -- readIf
      rd_w(0) <= '0';
      cpuDataOut <= (others=>'Z');
    end if readIf;
    rd_w(1) <= rd_w(0);
  end if clkIf;
end process read;
rd <= rd_w(0) and not rd_w(1);

cpuWriteSync: process(cpuClk)
begin
  clkIf: if( rising_edge(cpuClk) ) then
    rstIf: if( rst='1' ) then
      data_w <= (others=>'0');
    elsif( state_e=writing ) then -- rstIf
      data_w <= cpuDataIn(data_w'range);
      sync <= '1';
    else -- rstIf
      sync <= '0';
    end if rstIf;
  end if clkIf;
end process cpuWriteSync;

write: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    writeIf: if( sync_w='0' and sync='1') then
      dataOut <= data_w;
      wr <= '1';
    else -- writeIf
      wr <= '0';
    end if writeIf;
    sync_w <= sync;
  end if clkIf;
end process write;

end rtl;
```

```
-- Erinyes
--
-- CPU Front end
--
-- Purpose:
--  CPU buffers instantiation
--
-- CPU side:
--    cpuAddress: input address
--    cpuData: incoming cpu data bus
--    cpuRw: cpu write request
--    cpuOe: cpu output enable
--    cpuCs: clock enable
--    cpuRst: reset signal
--    cpuClk: cpu clock signal
--    cpuInt: cpu interrupt line
--    cpuGsr: global reset
--    cpuInitB: fpga programming initialization
--    cpuCsB: fpga programming csB
--    cpuRdWrB: fpga programming write enable
--
-- FPGA side:
--    cpuAddress: input address
--    dataIn: incoming cpu data bus
--    dataOut: outgoing cpu data bus
--    rw: cpu write request
--    oe: cpu output enable
--    cs: clock enable
--    rst: reset signal
--    clk: cpu clock signal
--    int: cpu interrupt line
--    gsr: global reset
--    initBin: fpga programming initialization incoming
--    initBout: fpga programming initialization outgoing
--    initBt: tristate enable
--    csB: fpga programming csB
--    rdWrB: fpga programming write enable
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on


entity cpuFrontEnd is
  generic(
    constant ADDR_WIDTH: integer := 24;
    constant CPU_DATA_WIDTH: integer := 32
  );
  port(
    cpuAddress: in std_logic_vector(ADDR_WIDTH-1 downto 0);
    cpuData: inout std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
    cpuRw: in std_logic;
    cpuOe: in std_logic;
    cpuCs: in std_logic;
    cpuRst: in std_logic;
    cpuClk: in std_logic;
    cpuInt: out std_logic;
```

```
    cpuGsr: in std_logic;
    cpuInitB: inout std_logic;
    cpuCsB: in std_logic;
    cpuRdWrB: in std_logic;

    address: out std_logic_vector(ADDR_WIDTH-1 downto 0);
    dataIn: out std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
    dataOut: in std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
    rw: out std_logic;
    oe: out std_logic;
    cs: out std_logic;
    rst: out std_logic;
    clk: out std_logic;
    int: in std_logic;
    gsr: out std_logic;
    initBin: out std_logic;
    initBout: in std_logic;
    initBt: in std_logic;
    csB: out std_logic;
    rdWrB: out std_logic
  );
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of cpuFrontEnd: entity is "cpuData, cpuInitB";
end cpuFrontEnd;
```

```
-- Erinyes
--
-- CPU Front end
--
-- Purpose:
--  CPU buffers instantiation
--
-- CPU side:
--    cpuAddress: input address
--    cpuData: incoming cpu data bus
--    cpuRd: cpu read request
--    cpuRw: cpu write request
--    cpuOe: cpu output enable
--    cpuCs: clock enable
--    cpuRst: reset signal
--    cpuClk: cpu clock signal
--    cpuInt: cpu interrupt line
--    cpuGsr: global reset
--    cpuInitB: fpga programming initialization
--    cpuCsB: fpga programming csB
--    cpuRdWrB: fpga programming write enable
--
-- FPGA side:
--    cpuAddress: input address
--    dataIn: incoming cpu data bus
--    dataOut: outgoing cpu data bus
--    rd: cpu read request
--    rw: cpu write request
--    oe: cpu output enable
--    cs: clock enable
--    rst: reset signal
--    clk: cpu clock signal
--    int: cpu interrupt line
--    gsr: global reset
--    initB: fpga programming initialization
--    csB: fpga programming csB
--    rdWrB: fpga programming write enable
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of cpuFrontEnd is

  component ibufLvc33 is
    port ( o: out std_logic_vector; i: in std_logic_vector );
  end component ibufLvc33;

  component iobufLvc33 is
    port ( o: out std_logic_vector; io: inout std_logic_vector;
           i: in std_logic_vector; t: in std_logic );
  end component iobufLvc33;

  component ibuf_lvcmos33 is
    port ( o: out std_logic; i: in std_logic );
  end component ibuf_lvcmos33;

  component obuf_lvcmos33_f_12 is
    port ( o: out std_logic; i: in std_logic );
  end component obuf_lvcmos33_f_12;

  component iobuf_lvcmos33_f_12 is
    port ( o: out std_logic; io: inout std_logic;
           i: in std_logic; t: in std_logic );
  end component iobuf_lvcmos33_f_12;

  attribute syn_black_box: boolean;
```

```
  attribute syn_black_box of ibuf_lvcmos33: component is true;
  attribute syn_black_box of obuf_lvcmos33_f_12: component is true;
  attribute syn_black_box of iobuf_lvcmos33_f_12: component is true;


-- interconnect
  signal oe_w: std_logic;

begin

  addBuf: ibufLvc33
  port map( address, cpuAddress);

  dataBuf: iobufLvc33
  port map( dataIn, cpuData, dataOut, oe_w );

  rwBuf:  ibuf_lvcmos33
  port map( rw, cpuRw );

  oeBuf:  ibuf_lvcmos33
  port map( oe_w, cpuOe );
  oe <= oe_w;

  csBuf:  ibuf_lvcmos33
  port map( cs, cpuCs );

  rstBuf:  ibuf_lvcmos33
  port map( rst, cpuRst );

  clkBuf:  ibuf_lvcmos33
  port map( clk, cpuClk );

  intBuf:  obuf_lvcmos33_f_12
  port map( cpuInt, int );

  gsrBuf:  ibuf_lvcmos33
  port map( gsr, cpuGsr );

  initBbuf:  iobuf_lvcmos33_f_12
  port map( initBin, cpuInitB, initBout, initBt );

  csBbuf:  ibuf_lvcmos33
  port map( csB, cpuCsB );

  rdWrBbbuf:  ibuf_lvcmos33
  port map( rdWrB, cpuRdWrB );

end struct;
```

```
-- Erinyes
--
-- SPI interface
--
-- Purpose:
--    SPI communication module
--
-- I/O:
--  Internal signals:
--    device: device address
--    dataIn: data channel (in)
--    dataOut: data channel (out)
--    rdy: indicates that the interface is ready
--    ce: clock enable
--    rst: reset
--    clk50MHz: smbus clock
--    clk: internal clock
--  External signals:
--    sDi: SPI input pin
--    sDo: SPI output pin
--    sRdy: SPI ready signal
--    sCs: chip select signal
--    sPrN: preset signal
--    sClk: chip synchronization clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on


entity spiInt is
  generic(
    DATA_WIDTH: integer:= 24;
    NB_DEVICES: integer:= 18
  );
  port(
    device: in std_logic_vector;
    dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
    dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
    rdy: out std_logic;
    ce: in std_logic;
    rst: in std_logic;
    clk50MHz: in std_logic;
    sDi: in std_logic;
    sDo: out std_logic;
    sRdy: in std_logic;
    sCs: out std_logic_vector(NB_DEVICES-1 downto 0);
    sPrN: out std_logic;
    sClk: out std_logic
  );
end spiInt;
```

```
-- Erinyes
--
-- SPI interface
--
-- Purpose:
--    SPI communication module
--
-- I/O:
--  Internal signals:
--    device: device address
--    dataIn: data channel (in)
--    dataOut: data channel (out)
--    rdy: indicates that the interface is ready
--    ce: clock enable
--    rst: reset
--    clk50MHz: smbus clock
--  External signals:
--    sDi: SPI input pin
--    sDo: SPI output pin
--    sRdy: SPI ready signal
--    sCs: chip select signal
--    sPrN: preset signal
--    sClk: chip synchronization clock
--
--  NOTE: Wait for rdy to go low before deassertion of ce.
--
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture rtl of spiInt is

  component ibuf_lvcmos33 is
    port( o: out std_logic; i:in std_logic );
  end component ibuf_lvcmos33;

  component obuf_lvcmos33_f_12 is
    port( o: out std_logic; i:in std_logic );
  end component obuf_lvcmos33_f_12;

  component obufLvc33 is
    port( o: out std_logic_vector; i:in std_logic_vector );
  end component obufLvc33;


  attribute syn_black_box: boolean;
  attribute syn_black_box of ibuf_lvcmos33: component is true;
  attribute syn_black_box of obuf_lvcmos33_f_12: component is true;


-- data shifting register
  signal data_w: std_logic_vector(dataIn'range);
  signal sCs_w: std_logic_vector(NB_DEVICES-1 downto 0);

-- bit counter
  signal count: std_logic_vector(log2m1(NB_DEVICES) downto 0);

-- internal signals
  signal rdy_w: std_logic;
  signal sClkEn: std_logic;
  signal rstB: std_logic;
  signal clk50MhzGated: std_logic;
  signal sRdy_w: std_logic;
  signal sDi_w, sDo_w: std_logic;
  signal sCsBuf_w: std_logic_vector(NB_DEVICES-1 downto 0);
```

```
begin

  sPrNbuf: obuf_lvcmos33_f_12
  port map( sPrN, rstB );
  rstB <= not rst;

  sRdyBuf: ibuf_lvcmos33
  port map( sRdy_w, sRdy );


  clockEnable: process(clk50MHz)
  begin
    clkIf: if( falling_edge(clk50MHz) ) then
      enable: if( sRdy_w='1' and rdy_w='0' ) then
        sClkEn <= '1';
      else -- enable
        sClkEn <= '0';
      end if enable;
    end if clkIf;
  end process clockEnable;

  sClkBuf: obuf_lvcmos33_f_12
  port map( sClk, clk50MhzGated );
  clk50MhzGated <= clk50MHz and sClkEn;



  counter: process(clk50Mhz)
  begin
    clkIf: if( rising_edge(clk50Mhz) ) then
      rstIf: if( rdy_w='1' ) then
        count <= (others=>'0');
      elsif( rdy_w='0' ) then -- rstIf
        count <= count + 1;
      end if rstIf;
    end if clkIf;
  end process counter;



  rdySig: process(clk50Mhz)
  begin
    clkIf: if( rising_edge(clk50Mhz) ) then
      rstIf: if( rst='1' ) then
        rdy_w <= '1';
      elsif( ce='1' ) then -- rstIf
        rdy_w <= '0';
      else -- rstIf
        stopIf: if( count=DATA_WIDTH-1 ) then
          rdy_w <= '1';
        end if stopIf;
      end if rstIf;
    end if clkIf;
  end process rdySig;
  rdy <= rdy_w;



  assert( NB_DEVICES<=sCs_w'length )
  report "Bad number of devices for sCs width, please verify!"
  severity failure;

  csAssign: process(clk50MHz)
  begin
    clkIf: if( rising_edge(clk50MHz) ) then
```

```
      ceIf: if( ce='1' ) then
        sCsBuf_w <= sCs_w;
      elsif( rdy_w='1' ) then -- ceIf
        sCsBuf_w <= (others=>'0');
      end if ceIf;
    end if clkIf;
  end process csAssign;
  csGen: for i in sCs_w'range generate
    sCs_w(i) <= '1' when i=conv_integer(device) else '0';
  end generate csGen;

  sCsBuf: obufLvc33
  port map( sCs, sCsBuf_w );




  shift: process(clk50Mhz)
  begin
    clkIf: if( rising_edge(clk50Mhz) ) then
      ceIf: if( ce='1' ) then
        data_w <= dataIn;
      elsif( rdy_w='0' and sRdy_w='1' ) then -- ceIf
        data_w <= data_w(data_w'high-1 downto 0) & sDi_w;
      end if ceIf;
    end if clkIf;
  end process shift;
  sDo_w <= data_w(data_w'high);
  dataOut <= data_w;

  sDiBuf: ibuf_lvcmos33
  port map( sDi_w, sDi );

  sDoBuf: obuf_lvcmos33_f_12
  port map( sDo, sDo_w );

end rtl;
```

```
-- Erinyes
--
-- SPI interface testbench
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity spiInt_tb is
end spiInt_tb;


architecture behav of spiInt_tb is


  component spiInt is
    generic(
      DATA_WIDTH: integer:= 24;
      NB_DEVICES: integer:= 18
    );
    port(
      device: in std_logic_vector;
      dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
      dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
      rdy: out std_logic;
      ce: in std_logic;
      rst: in std_logic;
      clk50MHz: in std_logic;
      clk: in std_logic;
      sDi: in std_logic;
      sDo: out std_logic;
      sRdy: in std_logic;
      sCs: out std_logic_vector(NB_DEVICES-1 downto 0);
      sPrN: out std_logic;
      sClk: out std_logic
    );
  end component;


-- timing constants
  constant fpgaPeriod_c: time:= 8 ns;
  constant spiPeriod_c: time:= 20 ns;

-- signal structure
  type spiSig_typ is record
    dev: std_logic_vector(4 downto 0);
    dta: std_logic_vector(23 downto 0);
    ce: std_logic;
  end record;
  signal spiSig_rec: spiSig_typ;

-- internal signals
  signal dataOut: std_logic_vector(spiSig_rec.dta'range);
  signal rdy, rst: std_logic;
  signal clk50MHz: std_logic:= '0';
  signal clk: std_logic:= '0';
  signal sDi, sDo, sRdy, sClk: std_logic;
  signal sCs: std_logic_vector(17 downto 0);
  signal sPrN: std_logic;

-- data constant
  signal sdiVect: std_logic_vector(dataOut'range)
```

```
                := x"abcdef";

-- usefull procedures
  procedure send(constant dev: in integer;
                 constant dta: in std_logic_vector(23 downto 0);
                 signal sig_rec: out spiSig_typ ) is
  begin
    sig_rec.dev <= conv_std_logic_vector(dev, 5);
    sig_rec.dta <= dta;
    sig_rec.ce <= '1';
    wait on rdy'transaction;
    sig_rec.ce <= '0';
    wait on rdy'transaction;
    wait for fpgaPeriod_c;
  end procedure send;


begin

  spi0: spiInt
  generic map( 24, 18 )
  port map( spiSig_rec.dev, spiSig_rec.dta, dataOut, rdy,
            spiSig_rec.ce, rst, clk50MHz, clk,
            sDi, sDo, sRdy, sCs, sPrN, sClk );


  clk <= not clk after fpgaPeriod_c/2;
  clk50MHz <= not clk50MHz after spiPeriod_c/2;


  dataIn: process(clk50MHz)
  begin
    clkIf: if( rising_edge(clk50MHz) ) then
      shiftIf: if( rdy='0' ) then
        sdiVect <= sdiVect(sdiVect'high-1 downto 0) & sdiVect(sdiVect'high);
        sDi <= sdiVect(sdiVect'high);
      end if;
    end if clkIf;
  end process dataIn;


  test: process
  begin
    rst <= '1';
    sRdy <= '1';  -- to unlock interface
    spiSig_rec.ce <= '0';
    wait for spiPeriod_c;
    rst <= '0';
    wait for spiPeriod_c;

    send(4, x"123456", spiSig_rec);
    send(2, x"568432", spiSig_rec);

  end process test;

end behav;
```

```
-- Erinyes
--
-- SMBus interface
--
-- Purpose:
--    SMBus communication module
--
-- I/O:
--  Internal signals:
--    device: device address
--    dataIn: data channel (in)
--    dataOut: data channel (out)
--    rdy: indicates that the interface is ready
--    rw: read/write signal
--    ce: clock enable
--    rst: reset
--    clk100kHz: smbus clock
--    clk: internal clock
--  External signals:
--    sData: data signal
--    sClk: chip synchronization clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on


entity smbInt is
  port(
    device: in std_logic_vector(6 downto 0);
    dataIn: in std_logic_vector(15 downto 0);
    dataOut: out std_logic_vector(15 downto 0);
    rdy: out std_logic;
    rw: in std_logic;
    ce: in std_logic;
    rst: in std_logic;
    clk100kHz: in std_logic;
    clk: in std_logic;
    sData: inout std_logic;
    sClk: out std_logic
  );
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of smbInt: entity is "sData";
end smbInt;
```

```
-- Erinyes
--
-- SMBus interface
--
-- Purpose:
--    SMBus communication module
--
-- I/O:
--  Internal signals:
--    device: device address
--    dataIn: data channel (in)
--    dataOut: data channel (out)
--    rw: read/write signal
--    ce: clock enable
--    clk: internal clock
--  External signals:
--    sData: data signal
--    sClk: chip synchronization clock
--
--  NOTE:  When data is read, lower byte is the effective data, high
--         byte is status (-1 error, 0 good).
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture rtl of smbInt is


  component obuf_lvcmos33_f_12 is
    port( o: out std_logic; i: in std_logic );
  end component obuf_lvcmos33_f_12;

  component iobuf_lvcmos33_f_12 is
    port( o: out std_logic; io: inout std_logic;
          i: in std_logic; t: in std_logic );
  end component iobuf_lvcmos33_f_12;

  attribute syn_black_box: boolean;
  attribute syn_black_box of obuf_lvcmos33_f_12: component is true;
  attribute syn_black_box of iobuf_lvcmos33_f_12: component is true;


-- state machine enumerations
  type states_typ is ( idle, sndAddr, sndReg, wrData, rdData, done, abort );
  signal state_e: states_typ;

-- each frame is 9 clock cycles
  constant frame_c: std_logic_vector(3 downto 0):= conv_std_logic_vector(9, 4);

-- smbus constants
  constant read_c: std_logic:= '1';
  constant write_c: std_logic:= '0';

-- Alert acknowledge
  constant ara_c: std_logic_vector(6 downto 0)

                := "0001100";

-- bit counter
  signal count: std_logic_vector(3 downto 0);

-- status register
  signal status: std_logic_vector(7 downto 0);

-- In/Out signals
  signal sDataIn, sDataOut: std_logic;
```

```
-- shift registers
  signal address: std_logic_vector(device'range);
  signal rw_w: std_logic;
  signal data_w: std_logic_vector(7 downto 0);
  signal reg: std_logic_vector(7 downto 0);

-- clock enable
  signal ce100kHz: std_logic;
  signal c100kHz: std_logic_vector(1 downto 0);
  signal sDataOutEn: std_logic;


-- readability procedure
  procedure sendData(signal sDataOutEn: out std_logic;
                     signal sDataOut: out std_logic;
                     constant data: in std_logic) is
  begin
    ackIf: if( count=frame_c-1 ) then
-- prepare for ack
      sDataOutEn <= '1';
    else -- ackIf;
      sDataOutEn <= '0';
      sDataOut <= data;
    end if ackIf;
  end sendData;


begin

  sDataBuf: iobuf_lvcmos33_f_12
  port map( sDataIn, sData, sDataOut, sDataOutEn );

  sClkBuf: obuf_lvcmos33_f_12
  port map( sClk, clk100kHz );



  rwSig: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      ceIf: if( ce='1' ) then -- rstIf
        rw_w <= rw;
      end if ceIf;
    end if clkIf;
  end process rwSig;



  stateMachine: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      rstIf: if( rst='1' ) then
        state_e <= idle;
        status <= (others=>'0');
      else -- rstIf
        stateCase: case( state_e ) is
          when idle =>
            ceIf: if( ce='1' ) then
              rdy <= '0';
              state_e <= sndAddr;
            else --ceIf
              rdy <= '1';
            end if ceIf;
          when sndAddr =>
            addrIf: if( count=frame_c ) then
```

```
            state_e <= sndReg;
            ackAddrIf: if( sDataIn/='0' ) then
              state_e <= abort;
            else -- ackAddrIf;
              araIf: if( device=ara_c ) then
                state_e <= rdData;
              else -- araIf
                state_e <= sndReg;
              end if araIf;
            end if ackAddrIf;
          end if addrIf;
        when sndReg =>
          sndRegIf: if( count=frame_c ) then
            ackRegIf: if( sDataIn/='0' ) then
              state_e <= abort;
            elsif ( rw=read_c ) then -- ackRegIf;
              state_e <= rdData;
            else -- ackRegIf
              state_e <= wrData;
            end if ackRegIf;
          end if sndRegIf;
        when wrData =>
          wrDataIf: if( count=frame_c ) then
            ackDataIf: if( sDataIn/='0' ) then
              state_e <= abort;
            else -- ackDataIf
              state_e <= done;
            end if ackDataIf;
          end if wrDataIf;
        when rdData =>
          rdDataIf: if( count=frame_c ) then
            state_e <= done;
          end if rdDataIf;
        when done =>
           state_e <= idle;
        when others =>
-- aborted means error occured, signal it in status
          status <= (others=>'1');
          state_e <= idle;
      end case stateCase;
    end if rstIf;
  end if clkIf;
end process stateMachine;



ce100kHzGen: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    c100kHz <= c100kHz(0) & clk100kHz;
  end if clkIf;
end process ce100kHzGen;
ce100kHz <= c100kHz(0) and not c100kHz(1);



bitCount: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    rstIf: if( rst='1' or count=frame_c) then
      count <= (others=>'0');
    elsif( ce100kHz='1' and state_e/=idle ) then -- rstIf
      count <= count+1;
```

```
      end if rstIf;
    end if clkIf;
  end process bitCount;



  addrShift: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      ceIf: if( ce='1' ) then
        address <= device;
      elsif( ce100kHz='1' and state_e=sndAddr ) then --ceIf
-- rw is transmitted with address
        address <= address(address'high-1 downto 0) & rw_w;
      end if ceIf;
    end if clkIf;
  end process addrShift;



  regShift: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      ceIf: if( ce='1' ) then

        reg <= dataIn(15 downto 8);
      elsif( ce100kHz='1' and state_e=sndReg ) then --ceIf
-- rw is transmitted with address
        reg <= reg(reg'high-1 downto 0) & '0';
      end if ceIf;
    end if clkIf;
  end process regShift;



  dataShift: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      ceIf: if( ce='1' and rw=write_c ) then
        data_w <= dataIn(7 downto 0);
      elsif( ce100kHz='1' ) then -- ceIf
        stateIf: if( state_e=wrData ) then
          data_w <= data_w(data_w'high-1 downto 0) & '0';
        elsif( state_e=rdData ) then
          data_w <= data_w(data_w'high-1 downto 0) & sDataIn;
        end if stateIf;
      end if ceIf;
    end if clkIf;
  end process dataShift;
  dataOut <= status & data_w;



  sDataCtrl: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      rstIf: if( rst='1' ) then
        sDataOutEn <= '1';
      else -- rstIf
```

```
        stateCase: case( state_e ) is
          when idle =>
            ceIf: if( ce='1' ) then
-- send start
              sDataOutEn <= '0';
              sDataOut <= '0';
            end if ceIf;
          when sndAddr =>
            sendData(sDataOutEn, sDataOut,
                     address(address'high));
          when sndReg =>
            sendData(sDataOutEn, sDataOut,
                     reg(reg'high));
          when wrData =>
            sendData(sDataOutEn, sDataOut,
                     data_w(data_w'high));
          when done =>
             sDataOutEn <= '1';
          when others =>
            sDataOutEn <= '1';
        end case stateCase;
      end if rstIf;
    end if clkIf;
  end process sDataCtrl;


end rtl;
```

```
-- Erinyes
--
-- SMBus interface testbench
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity smbusInt_tb is
end smbusInt_tb;


architecture behav of smbusInt_tb is

  component smbInt is
    port(
      device: in std_logic_vector(6 downto 0);
      dataIn: in std_logic_vector(15 downto 0);
      dataOut: out std_logic_vector(15 downto 0);
      rdy: out std_logic;
      rw: in std_logic;
      ce: in std_logic;
      rst: in std_logic;
      clk100kHz: in std_logic;
      clk: in std_logic;
      sData: inout std_logic;
      sClk: out std_logic
    );
  end component;


  type smbusSig_typ is record
    device: std_logic_vector(6 downto 0);
    data: std_logic_vector(15 downto 0);
    rw: std_logic;
    ce: std_logic;
  end record;
  signal smbSig_rec: smbusSig_typ;


-- clock periods
  constant fpgaPeriod_c: time:= 8 ns;
  constant smbusPeriod_c: time:= 10 us;


  signal clk: std_logic:= '0';
  signal clk100kHz: std_logic:= '0';
  signal rdy, rst: std_logic;
  signal sData, sClk: std_logic;
  signal dataOut: std_logic_vector(smbSig_rec.data'range);


  procedure sendData(constant addr: in std_logic_vector(6 downto 0);
                     constant dta: in std_logic_vector(15 downto 0);
                     signal smbSig_rec: inout smbusSig_typ ) is
  begin
    rdyIf: if( rdy='0' ) then
      wait on rdy'transaction;
    else -- rdyIf
      smbSig_rec.ce <= '1';
      smbSig_rec.rw <= '0';
      smbSig_rec.data <= dta;
```

```
      smbSig_rec.device <= addr;
      wait for fpgaPeriod_c;
      smbSig_rec.ce <= '0';
      wait for fpgaPeriod_c;
      wait on rdy'transaction;
      wait on rdy'transaction;
    end if rdyIf;
  end procedure sendData;


  procedure getData(constant addr: in std_logic_vector(6 downto 0);
                    constant dta: in std_logic_vector(15 downto 0);
                    signal smbSig_rec: inout smbusSig_typ ) is
  begin
    rdyIf: if( rdy='0' ) then
      wait on rdy'transaction;
    else -- rdyIf
      smbSig_rec.ce <= '1';
      smbSig_rec.rw <= '1';
      smbSig_rec.data <= dta;
      smbSig_rec.device <= addr;
      wait for fpgaPeriod_c;
      smbSig_rec.data <= (others=>'Z');
      smbSig_rec.ce <= '0';
      wait for fpgaPeriod_c;
      wait on rdy'transaction;
      wait on rdy'transaction;
      wait for fpgaPeriod_c;
    end if rdyIf;
  end procedure getData;



begin

  smb0: smbInt
  port map( smbSig_rec.device, smbSig_rec.data, dataOut, rdy, smbSig_rec.rw,
            smbSig_rec.ce, rst, clk100kHz, clk, sData, sClk );

  clk <= not clk after fpgaPeriod_c/2;
  clk100kHz <= not clk100kHz after smbusPeriod_c/2;

  test: process
  begin

    sData <= 'L';
    rst <= '1';
    smbSig_rec.ce <= '0';
    smbSig_rec.rw <= '0';
    smbSig_rec.device <= (others=>'0');
    smbSig_rec.data <= (others=>'Z');
    wait for 2*smbusPeriod_c;
    rst <= '0';
    wait for fpgaPeriod_c;

    sendData("101"&x"3", x"5678", smbSig_rec);
    sendData("010"&x"6", x"a2dc", smbSig_rec);
    getData("100"&x"5", x"3400", smbSig_rec);
    wait for fpgaPeriod_c;

  end process test;

end behav;
```

```
-- Erinyes
--
-- SSRAM Interface: Data transfer and control
--
-- Purpose:
--    Controls the SSRAM chips.
--
-- Internal signals
--    address: outgoing address
--    cSel: chip selection
--    dataAIn: write data
--    dataBIn: write data
--    dataAOut: read data
--    dataBOut: read data
--    rdy: data ready signal
--    byte: byte operation signal
--    rw: operation signal
--    ce: selection signal
--    rst: reset signal
--    addrPtr: internal address pointer
--    clk: clock signal
--
-- I/O:
--    sa: address lines
--    dqa: data line a
--    dqb: data line b
--    bwaN: byte write byte a
--    bwbN: byte write byte b
--    bweN: byte write enable
--    gwN: global write
--    advN: synchronous address advance
--    adscN: synchronous address status controller
--    adspN: synchronous address status processor
--    ceNx: clock enables
--    oeNx: output enables
--    zz: snooze mode enable
--    cko: output clock signal
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

entity ssramInt is
  generic(
    constant DATA_A_WIDTH: integer := 9;
    constant DATA_B_WIDTH: integer := 9;
    constant ADDR_WIDTH: integer := 19;
    constant DCM_PHASE_SHIFT: integer := 0
  );
  port(
    address: in std_logic_vector(ADDR_WIDTH-1 downto 0);
    cSel: in std_logic_vector(3 downto 0);
    dataAIn: in std_logic_vector(DATA_A_WIDTH-1 downto 0);
    dataBIn: in std_logic_vector(DATA_B_WIDTH-1 downto 0);
    dataAOut: out std_logic_vector(DATA_A_WIDTH-1 downto 0);
    dataBOut: out std_logic_vector(DATA_B_WIDTH-1 downto 0);
    rdy: out std_logic;
```

```
    byte: in std_logic_vector(1 downto 0);
    rw: in std_logic;
    ce: in std_logic;
    addrPtr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
    sleep: in std_logic;
    clk: in std_logic;

    sa: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa: inout std_logic_vector(8 downto 0) := (others=>'Z');
    dqb: inout std_logic_vector(8 downto 0) := (others=>'Z');

    bwaN: out std_logic;
    bwbN: out std_logic;
    bweN: out std_logic;
    gwN: out std_logic;

    advN: out std_logic;
    adscN: out std_logic;
    adspN: out std_logic;

    ceN: out std_logic_vector(3 downto 0);
    oeN: out std_logic_vector(3 downto 0);
    zz: out std_logic;
    cko: out std_logic
  );
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of ssramInt: entity is "dqa, dqb";
end ssramInt;
```

```
-- Erinyes
--
-- SSRAM Interface: Data transfer and control
--
-- Purpose:
--    Controls the SSRAM chips.
--
-- Internal signals
--    address: outgoing address
--    cSel: chip selection
--    dataAIn: write data
--    dataBIn: write data
--    dataAOut: read data
--    dataBOut: read data
--    rdy: data ready signal
--    byte: byte operation signal
--    rw: operation signal
--    ce: selection signal
--    rst: reset signal
--    addrPtr: internal address pointer
--    sleep: puts SSRAM chip in sleep mode
--    clk: clock signal
--
-- I/O:
--    sa: address lines
--    dqa: data line a
--    dqb: data line b
--    bwaN: byte write byte a
--    bwbN: byte write byte b
--    bweN: byte write enable
--    gwN: global write
--    advN: synchronous address advance
--    adscN: synchronous address status controller
--    adspN: synchronous address status processor
--    ceNx: clock enables
--    oeNx: output enables
--    zz: snooze mode enable
--    cko: clock output
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture rtl of ssramInt is


  component DCM
-- synthesis translate_off
    generic (
      CLK_FEEDBACK :string := "1X";
      CLKDV_DIVIDE : real := 2.0; -- (1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 8.0, 16.0)
      CLKFX_DIVIDE : integer := 1; -- (1 to 4096)
      CLKFX_MULTIPLY : integer := 4; -- (1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 5.5,
      -- 6.0, 6.5, 7.0, 7.5, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0)
      CLKIN_DIVIDE_BY_2 : boolean := FALSE; -- (TRUE, FALSE)
      CLKOUT_PHASE_SHIFT: string := "NONE";
      DESKEW_ADJUST: string := "SYSTEM_SYNCHRONOUS";
      DFS_FREQUENCY_MODE: string := "LOW";
      DLL_FREQUENCY_MODE: string := "LOW";
      DSS_MODE: string := "NONE";
      DUTY_CYCLE_CORRECTION : Boolean := TRUE; -- (TRUE, FALSE)
      FACTORY_JF : bit_vector := X"C080";
      PHASE_SHIFT: integer := 0;
      STARTUP_WAIT :boolean := FALSE
    );
-- synthesis translate_on
    port(
```

```
      CLK0 : out std_logic;
      CLK180 : out std_logic;
      CLK270 : out std_logic;
      CLK2X : out std_logic;
      CLK2X180 : out std_logic;
      CLK90 : out std_logic;
      CLKDV : out std_logic;
      CLKFX : out std_logic;
      CLKFX180 : out std_logic;
      LOCKED : out std_logic;
      PSDONE : out std_logic;
      STATUS : out std_logic_vector(7 downto 0);
      CLKFB : in std_logic;
      CLKIN : in std_logic;
      DSSEN : in std_logic;
      PSCLK : in std_logic;
      PSEN : in std_logic;
      PSINCDEC: in std_logic;
      RST : in std_logic
    );
  end component DCM;

  attribute xc_props :string;
  attribute xc_props of dcm0 : label is
          "PHASE_SHIFT=" & integer'image(DCM_PHASE_SHIFT);
  attribute syn_black_box: boolean;
  attribute syn_black_box of dcm: component is true;


  component iobufLvc33 is
    port ( o: out std_logic_vector; io: inout std_logic_vector;
           i: in std_logic_vector; t: in std_logic );
  end component iobufLvc33;

  component obufLvc33 is
    port ( o: out std_logic_vector; i: in std_logic_vector );
  end component obufLvc33;

  component obuf_lvcmos33_f_12  is
    port ( o: out std_logic; i: in std_logic);
  end component obuf_lvcmos33_f_12;
  attribute syn_black_box of obuf_lvcmos33_f_12: component is true;



-- static values (change this and die!)
  constant dataWidth_c: integer := 9;
  constant addressWidth_c: integer := 19;

  type states_typ is (latchAddress, read, write, deselect);
  signal state: states_typ;

-- internal signals
   signal sa_w: std_logic_vector(sa'range);
   signal bwaN_w, bwbN_w, bweN_w: std_logic;
   signal advN_w: std_logic_vector(1 downto 0);
   signal adscN_w, adspN_w: std_logic;
   signal ceN_w: std_logic;
   signal ceNb_w: std_logic_vector(ceN'range);
   signal oeN_w: std_logic;
   signal oeNb_w: std_logic_vector(oeN'range);
   signal cko_w: std_logic;
   signal writeEn: std_logic;
   signal dqaOut, dqaIn: std_logic_vector(dqa'range);
   signal dqbOut, dqbIn: std_logic_vector(dqb'range);
```

```
-- internal address counter
  signal addrCnt: std_logic_vector(ADDR_WIDTH-1 downto 0);
  signal brstCnt: std_logic_vector(1 downto 0);

-- dcm signals
  signal clk0, clk180, clk270: std_logic;
  signal clk2x, clk2x180, clk90: std_logic;
  signal clkDv, clkFx, clkFx180: std_logic;
  signal locked, psDone: std_logic;
  signal stat: std_logic_vector(7 downto 0);
  signal dssEn, psClk, psEn, psIncDec: std_logic:= '0';


begin

  dcm0: DCM
-- synthesis translate_off
  generic map( PHASE_SHIFT => DCM_PHASE_SHIFT )
-- synthesis translate_on
  port map( clk0, clk180, clk270, clk2x, clk2x180, clk90,
            clkDv, clkFx, clkFx180, locked, psDone, stat,
            cko_w, clk, dssEn, psClk, psEn, psIncDec, '0');



  dqaBuf: iobufLvc33
  port map( dqaIn, dqa, dqaOut, writeEn );

  dqbBuf: iobufLvc33
  port map( dqbIn, dqb, dqbOut, writeEn );



-- let clock and sleep pass through
  cko_w <= clk0;

  ckoBuf: obuf_lvcmos33_f_12
  port map( cko, clk0 );

  zzBuf:  obuf_lvcmos33_f_12
  port map( zz, sleep );



  machine: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      rstIf: if( ce='0' ) then
        state <= latchAddress;
      else -- rstIf
        machCase: case( state ) is
          when latchAddress =>
            readStateIf: if( rw='1' ) then
              state <= read;
            else -- readStateIf
              state <= write;
            end if readStateIf;
          when read =>
            keepReadIf: if( rw='0' ) then
              state <= latchAddress;
            end if keepReadIf;
          when write =>
            keepWriteIf: if( rw='1' ) then
              state <= latchAddress;
            end if keepWriteIf;
          when others =>
```

```
          state <= latchAddress;
      end case machCase;
    end if rstIf;
  end if clkIf;
end process machine;



addressCount: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    countIf: if( state=latchAddress ) then
      addrCnt <= address;
      addrPtr <= addrCnt;
    else -- countIf
      addrCnt <= addrCnt + 1;
    end if countIf;
  end if clkIf;
end process addressCount;



burstCount: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    rstIf: if( adspN_w='0' ) then
      brstCnt <= (others=>'1');
    else -- rstIf
      brstCnt <= brstCnt - 1;
    end if rstIf;
  end if clkIf;
end process burstCount;



adsc: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    firstLatchIf: if( state=latchAddress ) then
      activeIf: if( ce='1' ) then
        adscN_w <= '1';
      end if activeIf;
    elsif( state=read and rw='0' ) then -- firstLatchIf
      adscN_w <= '0';
    else -- firstLatchIf
      addrLatchIf: if( ce='1' and rw='0' and
                       brstCnt=x"0" ) then
        adscN_w <= '0';
      else -- addrLatchIf
        adscN_w <= '1';
      end if addrLatchIf;
    end if firstLatchIf;
  end if clkIf;
end process adsc;

adscNbuf: obuf_lvcmos33_f_12
port map( adscN, adscN_w );



adsp: process(clk)
begin
```

```
  clkIf: if( rising_edge(clk) ) then
    firstLatchIf: if( state=latchAddress ) then
      activeIf: if( ce='1' ) then
        adspN_w <= '0';
      end if activeIf;
    elsif( state=read ) then -- firstLatchIf
      addrLatchIf: if( rw='1' and brstCnt=x"0" ) then
        adspN_w <= '0';
      else -- addrLatchIf
        adspN_w <= '1';
      end if addrLatchIf;
    else -- firstLatchIf
      adspN_w <= '1';
    end if firstLatchIf;
  end if clkIf;
end process adsp;

adspNbuf: obuf_lvcmos33_f_12
port map( adspN, adspN_w );



clockEnables: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    enableIf: if( state=latchAddress ) then
      ceN_w <= not ce;
    elsif( state=read ) then -- enableIf
      addrLatchIf: if( rw='0' and brstCnt=x"0" ) then
        ceN_w <= '0';
      else -- addrLatchIf
        ceN_w <= '1';
      end if addrLatchIf;
    elsif( state=write ) then -- enableIf
    else -- enableIf
      ceN_w <= '1';
    end if enableIf;
  end if clkIf;
end process clockEnables;

ceNgen: for i in ceN'range generate
  ceNb_w(i) <= not ceN_w nand cSel(i);
end generate ceNgen;

ceNbuf: obufLvc33
port map( ceN, ceNb_w );



adv: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    advanceIf: if( state/=latchAddress ) then
      advN_w(0) <= '0';
      advN_w(1) <= advN_w(0);
    else -- advanceIf
      advN_w <= (others=>'1');
    end if advanceIf;
  end if clkIf;
end process adv;

advNbuf: obuf_lvcmos33_f_12
port map( advN, advN_w(1) );
```

```
saX: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    latchIf: if( state=latchAddress ) then
      activeIf: if( ce='1' ) then
        sa_w(ADDR_WIDTH-1 downto 0) <= address;
      end if activeIf;
    elsif( brstCnt=x"0" ) then
      sa_w(ADDR_WIDTH-1 downto 0) <= addrCnt;
    end if latchIf;
  end if clkIf;
end process saX;

saBuf: obufLvc33
port map( sa, sa_w );



writePerm: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    readIf: if( state/=write ) then
      bweN_w <= '1';
      bwaN_w <= '1';
      bwbN_w <= '1';
    else -- readIf
      bweN_w <= '0';
      bwaN_w <= not byte(0);
      bwbN_w <= not byte(1);
    end if readIf;
  end if clkIf;
end process writePerm;

bweNbuf: obuf_lvcmos33_f_12
port map( bweN, bweN_w );

bwaNbuf: obuf_lvcmos33_f_12
port map( bwaN, bwaN_w );

bwbNbuf: obuf_lvcmos33_f_12
port map( bwbN, bwbN_w );

gwNbuf: obuf_lvcmos33_f_12
port map( gwN, '1' );



outputEnable: process(clk)
begin
  clkIf: if( rising_edge(clk) ) then
    enableIf: if( state=read ) then
      oeN_w <= '0';
    else -- enableIf
      oeN_w <= '1';
    end if enableIf;
  end if clkIf;
end process outputEnable;

oeNgen: for i in ceN'range generate
  oeNb_w(i) <= not oeN_w nand cSel(i);
end generate oeNgen;

oeNbuf: obufLvc33
```

```
  port map( oeN, oeNb_w );


  dqX: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      writeIf: if( state=write ) then
        writeEn <= '0';
      else -- writeIf
        writeEn <= '1';
      end if writeIf;
    end if clkIf;
  end process dqX;
  dqaOut(dataAIn'range) <= dataAIn;
  dqbOut(dataBIn'range) <= dataBIn;



  dataX: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      readIf: if( state=read ) then
        rdy <= '1';
        dataAOut <= dqaIn(dataAIn'range);
        dataBOut <= dqbIn(dataBIn'range);
      else -- readIf
        rdy <= '0';
      end if readIf;
    end if clkIf;
  end process dataX;

end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ssramInt_tb is
end ssramInt_tb;

architecture behav of ssramInt_tb is

  component ssramInt is
    generic(
      constant DATA_A_WIDTH: integer := 9;
      constant DATA_B_WIDTH: integer := 9;
      constant ADDR_WIDTH: integer := 19
    );
    port(
      address: in std_logic_vector(ADDR_WIDTH-1 downto 0);
      cSel: in std_logic_vector(3 downto 0);
      dataAIn: in std_logic_vector(DATA_A_WIDTH-1 downto 0);
      dataBIn: in std_logic_vector(DATA_B_WIDTH-1 downto 0);
      dataAOut: out std_logic_vector(DATA_A_WIDTH-1 downto 0);
      dataBOut: out std_logic_vector(DATA_B_WIDTH-1 downto 0);
      rdy: out std_logic;
      byte: in std_logic_vector(1 downto 0);
      rw: in std_logic;
      ce: in std_logic;
      addrPtr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
      sleep: in std_logic;
      clk: in std_logic;

      sa: out std_logic_vector(18 downto 0);
      dqa: inout std_logic_vector(8 downto 0);
      dqb: inout std_logic_vector(8 downto 0);

      bwaN: out std_logic;
      bwbN: out std_logic;
      bweN: out std_logic;
      gwN: out std_logic;

      advN: out std_logic;
      adscN: out std_logic;
      adspN: out std_logic;

      ceN: out std_logic_vector(3 downto 0);
      oeN: out std_logic_vector(3 downto 0);
      zz: out std_logic;
      cko: out std_logic
    );
  end component;

-- constants
  signal addrWidth_c: integer := 19;
  signal dataAWidth_c: integer := 9;
  signal dataBWidth_c: integer := 9;

-- internal signals
  signal clk: std_logic := '0';
  signal address: std_logic_vector(addrWidth_c-1 downto 0);
  signal cSel: std_logic_vector(3 downto 0);
  signal dataAIn: std_logic_vector(dataAWidth_c-1 downto 0);
  signal dataAOut: std_logic_vector(dataAWidth_c-1 downto 0);
  signal dataBIn: std_logic_vector(dataBWidth_c-1 downto 0);
  signal dataBOut: std_logic_vector(dataBWidth_c-1 downto 0);
  signal rdy: std_logic;
  signal byte: std_logic_vector(1 downto 0);
  signal rw: std_logic;
```

```
  signal ce: std_logic;
  signal addrPtr: std_logic_vector(addrWidth_c-1 downto 0);
  signal sleep: std_logic;
  signal sa: std_logic_vector(18 downto 0);
  signal dqa: std_logic_vector(8 downto 0);
  signal dqb: std_logic_vector(8 downto 0);
  signal bwaN, bwbN, bweN, gwN: std_logic;
  signal advN, adscN, adspN: std_logic;
  signal ceN: std_logic_vector(3 downto 0);
  signal oeN: std_logic_vector(3 downto 0);
  signal zz, cko: std_logic;
  signal mode, ce2N: std_logic;


begin

  clk <= not clk after 5 ns;

  ssramInt0: ssramInt
  generic map( 9, 9, 19 )
  port map( address, cSel, dataAIn, dataBIn, dataAOut, dataBOut, rdy, byte, rw, ce,
            addrPtr, sleep, clk, sa, dqa, dqb, bwaN, bwbN, bweN, gwN, advN,
            adscN, adspN, ceN, oeN, zz, cko );


  test: process
  begin
    mode <= '0';
    cSel <= "1111";

    sleep <= '0';
    rw <= '1';
    byte <= "11";
    dataAIn <= (others=>'0');
    dataBIn <= (others=>'0');
    wait for 5 ns;
    ce <= '0';
    wait for 30 ns;

-- WRITE
    report "Single writes";

    address <= "110" & x"1234";
    dataAIn <= '1' & x"23";
    dataBIn <= '0' & x"78";
    byte <= "11";
    rw <= '0';
    ce <= '1';
    wait for 10 ns;
    ce <= '0';
    wait for 10 ns;

    address <= "101" & x"1234";
    dataAIn <= '1' & x"23";
    dataBIn <= '0' & x"78";
    byte <= "11";
    rw <= '0';
    ce <= '1';
    wait for 10 ns;
    ce <= '0';
    wait for 50 ns;

    report "Burst writes";
    address <= "111" & x"1234";
    dataAIn <= '1' & x"23";
    dataBIn <= '0' & x"78";
    byte <= "11";
```

```
    rw <= '0';
    ce <= '1';
    burstWLoop: for i in 1 to 16 loop
      dataAIn <= dataAIn + 1;
      dataBIn <= dataBIn + 1;
      wait for 10 ns;
    end loop burstWLoop;
    ce <= '0';
    wait for 100 ns;

-- READ

    report "Single reads";
    address <= "110" & x"1234";
    rw <= '1';
    ce <= '1';
    wait for 10 ns;
    ce <= '0';
    wait for 10 ns;

    address <= "101" & x"1234";
    rw <= '1';
    ce <= '1';
    wait for 10 ns;
    ce <= '0';
    wait for 50 ns;

    report "Burst reads";
    address <= "111" & x"1234";
    rw <= '1';
    ce <= '1';
    burstRLoop: for i in 1 to 16 loop
      wait for 10 ns;
    end loop burstRLoop;
    ce <= '0';
    wait for 100 ns;


-- READ/WRITE/READ/WRITE
    report "Read/write/read/write";
    address <= "110" & x"1234";
    dataAIn <= '1' & x"23";
    dataBIn <= '0' & x"78";
    byte <= "11";
    rw <= '0';
    ce <= '1';
    wait for 10 ns;

    address <= "110" & x"1235";
    rw <= '1';
    wait for 10 ns;

    address <= "110" & x"1236";
    dataAIn <= '1' & x"23";
    dataBIn <= '0' & x"78";
    byte <= "11";
    rw <= '0';
    wait for 10 ns;

    address <= "110" & x"1237";
    rw <= '1';
    wait for 10 ns;
    ce <= '0';
    wait for 100 ns;


-- burst read, write burst read
```

```
    report "Burst read/write/burst read";
    address <= "111" & x"1234";
    rw <= '1';
    ce <= '1';
    burstR1Loop: for i in 1 to 16 loop
      wait for 10 ns;
    end loop burstR1Loop;

    address <= "110" & x"1236";
    byte <= "11";
    rw <= '0';
    wait for 10 ns;
    rw <= '1';
    dataAIn <= '1' & x"23";
    dataBIn <= '0' & x"78";
    wait for 10 ns;

    address <= "111" & x"1234";
    rw <= '1';
    ce <= '1';
    burstR2Loop: for i in 1 to 16 loop
      wait for 10 ns;
    end loop burstR2Loop;
    ce <= '0';
    wait for 100 ns;

  end process test;

end behav;
```

```
-- Erinyes
--
-- Digital potentiometers control
--
-- Purpose:
--    Serves as a controller for the digital potentiometers
--
-- I/O:
--  Internal signals:
--    device: device address
--    dataIn: data channel (in)
--    dataOut: data channel (out)
--    dataRdy: data available for reading
--    dataRd: data reading signal
--    ce: clock enable
--    rst: reset
--    clk50Mhz: internal spi clock
--    clk: internal clock
-- SPI side signals
--    spiDi: spi data in
--    spiDo: spi data out
--    spiRdy: interface ready signal
--    spiCs: chip select
--    spiPrN: spi preset
--    spiClk: spi clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

library work;
use work.utils_pak.all;

entity dPotentiometers is
  generic(
    QUEUE_LENGTH: integer:= 18;
    DATA_WIDTH: integer:= 24;
    NB_DEVICES: integer:= 18
  );
  port(
    device: in std_logic_vector;
    dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
    dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
    dataRdy: out std_logic;
    dataRd: in std_logic;
    ce: in std_logic;
    rst: in std_logic;
    clk50MHz: std_logic;
    clk: in std_logic;
    spiDi: in std_logic;
    spiDo: out std_logic;
    spiRdy: in std_logic;
    spiCs: out std_logic_vector;
    spiPrN: out std_logic;
    spiClk: out std_logic
  );
end dPotentiometers;
```

```
-- Erinyes
--
-- Digital potentiometers control
--
-- Purpose:
--    Serves as a controller for the digital potentiometers
--
-- I/O:
--  Internal signals:
--    device: device address
--    dataIn: data channel (in)
--    dataOut: data channel (out)
--    ce: clock enable
--    rst: reset
--    clk50Mhz: internal spi clock
--    clk: internal clock
-- SPI side signals
--    spiDi: spi data in
--    spiDo: spi data out
--    spiRdy: interface ready signal
--    spiCs: chip select
--    spiPrN: spi preset
--    spiClk: spi clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of dPotentiometers is

  component fifo is
    generic (
      WIDTH: integer:= 8;
      DEPTH: integer:= 16
    );
    port (
      wData: in std_logic_vector(WIDTH-1 downto 0);
      rData: out std_logic_vector(WIDTH-1 downto 0);
      mty: out std_logic;
      we: in std_logic;
      rd: in std_logic;
      rst: in std_logic;
      clk: in std_logic
    );
  end component fifo;



  component spiInt is
    generic(
      DATA_WIDTH: integer:= 24;
      NB_DEVICES: integer:= 18
    );
    port(
      device: in std_logic_vector;
      dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
      dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
      rdy: out std_logic;
      ce: in std_logic;
      rst: in std_logic;
      clk50MHz: in std_logic;
      sDi: in std_logic;
      sDo: out std_logic;
      sRdy: in std_logic;
      sCs: out std_logic_vector(NB_DEVICES-1 downto 0);
```

```
      sPrN: out std_logic;
      sClk: out std_logic
    );
  end component spiInt;


  component sigTrans is
    port(
      d: in std_logic;
      q: out std_logic;
      clk1: in std_logic;
      clk2: in std_logic
    );
  end component sigTrans;


-- geometry constants
  constant devWidth_c: integer:= log2m1(NB_DEVICES);
  constant dataWidth_c: integer:= devWidth_c+DATA_WIDTH+1;

-- internal signals
  signal wData: std_logic_vector(dataWidth_c-1 downto 0);
  signal rData: std_logic_vector(dataWidth_c-1 downto 0);
  signal outFifoIn: std_logic_vector(DATA_WIDTH-1 downto 0);
  signal dataRdy_w, shiftOut: std_logic;
  signal mty: std_logic;
  signal rd: std_logic;
  signal spiReady: std_logic;
  signal spiCe: std_logic;
  signal spiRst: std_logic;
  signal spiShifting: std_logic;
  signal doneShift: std_logic;



begin

  wData <= dataIn & device;

  fifoIn: fifo
  generic map( dataWidth_c, NB_DEVICES )
  port map( wData, rData, mty, ce, rd, rst, clk );

  dataRdy <= not dataRdy_w;
  shiftOut <= not dataRdy_w and dataRd;
  fifoOut: fifo
  generic map( DATA_WIDTH, NB_DEVICES )
  port map( outFifoIn, dataOut(outFifoIn'range), dataRdy_w, doneShift, shiftOut, rst, clk
);

  spi0: spiInt
  generic map( DATA_WIDTH, NB_DEVICES )
  port map( rData(device'range), rData(rData'high downto device'length), outFifoIn,
            spiReady, spiCe, spiRst, clk50MHz, spiDi, spiDo, spiRdy,
            spiCs, spiPrN, spiClk );

  rst0: sigTrans
  port map( rst, spiRst, clk, clk50Mhz );


  fifoReady: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      fifoRdyIf: if( mty='0' or spiCe='1' ) then
        spiCe <= spiReady;
      else -- fifoRdyIf
        spiCe <= '0';
```

```
      end if fifoRdyIf;
    end if clkIf;
  end process fifoReady;

  fifoRead: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      spiRdyIf: if( spiReady='1' and mty='0' and spiCe='0' ) then
        rd <= '1';
      elsif( rd='1' ) then -- spiRdyIf
        rd <= '0';
      end if spiRdyIf;
    end if clkIf;
  end process fifoRead;

  outFifoWrite: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      spiShifting <= spiReady;
      doneShift <= spiReady and not spiShifting;
    end if clkIf;
  end process outFifoWrite;

end struct;
```

```
-- Erinyes
--
-- Digital potentiometers
--
-- Purpose:
--    Serves as a controller for the digital potentiometers
--
-- I/O:
--  Internal signals:
--    address: cpu address
--    cpuDataIn: incoming cpu data bus
--    cpuDataOut: outgoing cpu data bus
--    rw: cpu read/write
--    oe: cpu output enable
--    cs: cpu chip select
--    rst: reset
--    cpuClk: cpu clock
--    clk: internal clock
-- SPI side signals
--    spiDi: spi data in
--    spiDo: spi data out
--    spiRdy: interface ready signal
--    spiCs: chip select
--    spiPrN: preset signal
--    spiClk: spi clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on
library work;
use work.utils_pak.all;

library synplify;
use synplify.all;
entity dp is
  generic(
    TRIG_ADDRESS: std_logic_vector:= x"0";
    TRIG_ADDRESS_STATUS: std_logic_vector:= x"0"
  );
  port(
    address: in std_logic_vector(22 downto 0);
    cpuDataIn: in std_logic_vector(31 downto 0);
    cpuDataOut: out std_logic_vector(31 downto 0);
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic;
    spiDi: in std_logic;
    spiDo: out std_logic;
    spiRdy: in std_logic;
    spiCs: out std_logic_vector(17 downto 0);
    spiPrN: out std_logic;
    spiClk: out std_logic
  );
end dp;
```

```
-- Erinyes
--
-- Digital potentiometers
--
-- Purpose:
--    Serves as a controller for the digital potentiometers
--
-- I/O:
--  Internal signals:
--    address: cpu address
--    cpuData: cpu data bus
--    rw: cpu read/write
--    oe: cpu output enable
--    cs: cpu chip select
--    rst: reset
--    cpuClk: cpu clock
--    clk: internal clock
-- SPI side signals
--    spiDi: spi data in
--    spiDo: spi data out
--    spiRdy: interface ready signal
--    spiCs: chip select
--    spiPrN: preset signal
--    spiClk: spi clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003
architecture rtl of dp is

  component DCM
-- synthesis translate_off
    generic (
      CLK_FEEDBACK :string := "1X";
      CLKDV_DIVIDE : real := 2.0; -- (1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 8.0, 16.0)
      CLKFX_DIVIDE : integer := 1; -- (1 to 4096)
      CLKFX_MULTIPLY : integer := 4; -- (1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 5.5,
      -- 6.0, 6.5, 7.0, 7.5, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0)
      CLKIN_DIVIDE_BY_2 : boolean := FALSE; -- (TRUE, FALSE)
      CLKOUT_PHASE_SHIFT: string := "NONE";
      DESKEW_ADJUST: string := "SYSTEM_SYNCHRONOUS";
      DFS_FREQUENCY_MODE: string := "LOW";
      DLL_FREQUENCY_MODE: string := "LOW";
      DSS_MODE: string := "NONE";
      DUTY_CYCLE_CORRECTION : Boolean := TRUE; -- (TRUE, FALSE)
      FACTORY_JF : bit_vector := X"C080";
      PHASE_SHIFT: integer := 0;
      STARTUP_WAIT :boolean := FALSE
    );
-- synthesis translate_on
    port(
      CLK0 : out std_logic;
      CLK180 : out std_logic;
      CLK270 : out std_logic;
      CLK2X : out std_logic;
      CLK2X180 : out std_logic;
      CLK90 : out std_logic;
      CLKDV : out std_logic;
      CLKFX : out std_logic;
      CLKFX180 : out std_logic;
      LOCKED : out std_logic;
      PSDONE : out std_logic;
      STATUS : out std_logic_vector(7 downto 0);
      CLKFB : in std_logic;
      CLKIN : in std_logic;
      DSSEN : in std_logic;
      PSCLK : in std_logic;
```

```
    PSEN : in std_logic;
    PSINCDEC: in std_logic;
    RST : in std_logic
  );
end component DCM;

attribute xc_props :string;
attribute xc_props of dcm0 : label is "CLKFX_MULTIPLY=15," &
                                      "CLKFX_DIVIDE=20";
attribute syn_black_box: boolean;
attribute syn_black_box of DCM: component is true;


component bufG
  port( o: out std_logic; i: in std_logic );
end component bufG;
attribute syn_black_box of bufG: component is true;



component cpuInt is
  generic(
    constant ADDR_WIDTH: integer := 24;
    constant DATA_WIDTH: integer := 32;
    constant TRIG_ADDRESS_HIGH: std_logic_vector := x"0";
    constant TRIG_ADDRESS_LOW: std_logic_vector := x"0"
  );
  port(
    address: in std_logic_vector(ADDR_WIDTH-1 downto 0);
    cpuDataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
    cpuDataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
    dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
    dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
    rd: out std_logic;
    wr: out std_logic;
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic
  );
end component cpuInt;

component dPotentiometers is
  generic(
    QUEUE_LENGTH: integer:= 18;
    DATA_WIDTH: integer:= 24;
    NB_DEVICES: integer:= 18
  );
  port(
    device: in std_logic_vector;
    dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
    dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
    dataRdy: out std_logic;
    dataRd: in std_logic;
    ce: in std_logic;
    rst: in std_logic;
    clk50MHz: std_logic;
    clk: in std_logic;
    spiDi: in std_logic;
    spiDo: out std_logic;
    spiRdy: in std_logic;
    spiCs: out std_logic_vector(17 downto 0);
    spiPrN: out std_logic;
    spiClk: out std_logic
  );
```

```
  end component dPotentiometers;

-- dcm signals
  signal clk0, clk180, clk270: std_logic;
  signal clk2x, clk2x180, clk90: std_logic;
  signal clkDv, clkFx, clkFx180: std_logic;
  signal clkfb: std_logic;
  signal locked, psDone: std_logic;
  signal stat: std_logic_vector(7 downto 0);
  signal dssEn, psClk, psEn, psIncDec: std_logic:= '0';

-- cpu interface
  signal dta: std_logic_vector(31 downto 0);
  signal dataIn: std_logic_vector(31 downto 0);
  signal dataOut: std_logic_vector(31 downto 0);
  signal rd, wr: std_logic;

-- interconnect
  signal clk50MHz: std_logic;
  signal dev: std_logic_vector(4 downto 0);
  signal dataIn_w: std_logic_vector(23 downto 0);
  signal statRd, statWr: std_logic;
  signal dataRdy: std_logic;
  signal rst_w: std_logic;

-- internal status register
  signal statRead: std_logic_vector(31 downto 0):= (others=>'0');
  signal statWrite: std_logic_vector(statRead'range):= (others=>'0');

begin

  rst_w <= rst or not locked;

  bufg0: bufG
  port map( clk50MHz, clkFx );

  dcm0: DCM
-- synthesis translate_off
  generic map( CLK_FEEDBACK=>"NONE", CLKFX_MULTIPLY=>15, CLKFX_DIVIDE=>20 )
-- synthesis translate_on
  port map( clk0, clk180, clk270, clk2x, clk2x180, clk90,
            clkDv, clkFx, clkFx180, locked, psDone, stat,
            clkfb, cpuClk, dssEn, psClk, psEn, psIncDec, rst );

  cpuInt0: cpuInt
  generic map( address'length, cpuDataIn'length, TRIG_ADDRESS, TRIG_ADDRESS )
  port map( address, cpuDataIn, cpuDataOut, dataIn, dataOut, rd, wr, rw,
            oe, cs, rst_w, cpuClk, clk );

  statReg: cpuInt
  generic map( address'length, cpuDataIn'length, TRIG_ADDRESS_STATUS,
               TRIG_ADDRESS_STATUS  )
  port map( address, cpuDataIn, cpuDataOut, statRead, statWrite,
            statRd, statWr, rw, oe, cs, rst_w, cpuClk, clk );

  dev <= dataOut(28 downto 24);
  dataIn <= x"00" & dataIn_w;
  dpot0: dPotentiometers
  generic map( 18, 24, 18 )
  port map( dev, dataOut(23 downto 0), dataIn_w, dataRdy, rd,
            wr, rst_w, clk50MHz, clk, spiDi, spiDo, spiRdy,
            spiCs, spiPrN, spiClk );

  statRead <= (0=>dataRdy, others=>'0');

end rtl;
```

```
-- Erinyes
--
-- Digital potentiometers control testbench
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library unisim;
use unisim.all;

library work;
use work.utils_pak.all;


entity dPotentiometers_tb is
end dPotentiometers_tb;


architecture behav of dPotentiometers_tb is

  component dPotentiometers is
    generic(
      QUEUE_LENGTH: integer:= 18;
      DATA_WIDTH: integer:= 24;
      NB_DEVICES: integer:= 18
    );
    port(
      device: in std_logic_vector;
      dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
      dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
      ce: in std_logic;
      rst: in std_logic;
      clk50Mhz: std_logic;
      clk: in std_logic;
      spiDi: in std_logic;
      spiDo: out std_logic;
      spiRdy: in std_logic;
      spiCs: out std_logic_vector;
      spiClk: out std_logic
    );
  end component dPotentiometers;


-- simulation constants
  constant fpgaPeriod_c: time:= 8 ns;
  constant spiPeriod_c: time:= 20 ns;
  constant nbDevices_c: integer:= 18;
  constant dataWidth_c: integer:= 24;

-- simulation structure
  type spiSig_typ is record
    dev: std_logic_vector(log2m1(nbDevices_c) downto 0);
    data: std_logic_vector(dataWidth_c-1 downto 0);
    ce: std_logic;
  end record spiSig_typ;
  signal spiSig_rec: spiSig_typ;

-- internal signals
  signal dataOut: std_logic_vector(spiSig_rec.data'range);
  signal rst: std_logic;
  signal clk50MHz: std_logic:= '0';
```

```
  signal clk: std_logic:= '0';
  signal spiDo: std_logic;
  signal spiCs: std_logic_vector(nbDevices_c-1 downto 0);
  signal spiClk: std_logic;
  signal sdiVect: std_logic_vector(spiSig_rec.data'range)
                    := x"fedcba";


-- simulation procedure
  procedure send(constant dev: in integer;
                 constant data: in std_logic_vector;
                 signal spiSig_rec: out spiSig_typ ) is
  begin
    spiSig_rec.dev <= conv_std_logic_vector(dev,
                        log2ml(nbDevices_c)+1);
    spiSig_rec.data <= data;
    spiSig_rec.ce <= '1';
    wait for fpgaPeriod_c;
    spiSig_rec.ce <= '0';
    wait for fpgaPeriod_c;
  end procedure send;


begin

  dp0: dPotentiometers
  generic map( nbDevices_c, dataWidth_c, nbDevices_c )
  port map( spiSig_rec.dev, spiSig_rec.data, dataOut, spiSig_rec.ce, rst,
            clk50MHz, clk, sDiVect(sDiVect'high), spiDo, '1', spiCs, spiClk );

  clk <= not clk after fpgaPeriod_c/2;
  clk50MHz <= not clk50MHz after spiPeriod_c/2;


  sdiGen: process(spiClk)
  begin
    clkIf: if( rising_edge(spiClk) ) then
      csIf: if( conv_integer(spiCs)/=0 ) then
        sDiVect <= sDiVect(sDiVect'high-1 downto 0) & sDiVect(sDiVect'high);
      end if csIf;
    end if clkIf;
  end process sdiGen;


  test: process
  begin

    spiSig_rec.ce <= '0';
    spiSig_rec.dev <= (others=>'0');
    spiSig_rec.data <= (others=>'0');
    rst <= '1';
    wait for spiPeriod_c;
    rst <= '0';
    wait for spiPeriod_c;

    send(5, x"123456", spiSig_rec);
    wait for 32*spiPeriod_c;

    send(8, x"48573d", spiSig_rec);
    wait for 32*spiPeriod_c;

    send(15, x"abd726", spiSig_rec);
    wait for 32*spiPeriod_c;

  end process test;

end behav;
```

```
-- Erinyes
--
-- LED interface
--
-- Purpose:
--    LED control module
--
-- I/O:
--  Internal signals:
--    led: LED address
--    state: led state
--    ce: clock enable
--    rst: reset
--    cpuClk: cpu clock
--    clk: internal clock
--  External signals:
--    pLed: LED power pin
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;


entity ledInt is
  generic(
    NB_STATES: integer:= 5;
    NB_DEVICES: integer:= 12
  );
  port(
    led: in std_logic_vector;
    state: in std_logic_vector;
    ce: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic;
    pLed: out std_logic_vector(NB_DEVICES-1 downto 0)
  );
end ledInt;
```

```
-- Erinyes
--
-- LED interface
--
-- Purpose:
--    LED control module
--
-- I/O:
--  Internal signals:
--    led: LED address
--    state: led state
--    rst: reset
--    cpuClk: cpu clock
--    clk: internal clock
--  External signals:

--    pLed: LED power pin
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;


architecture rtl of  ledInt is

  component ledMux is
    port(
      sel: in std_logic_vector;
      power: in std_logic_vector;
      output: out std_logic
    );
  end component ledMux;


-- led data holder
-- converted later into RAM by synplify
  subtype ledData_typ is std_logic_vector(log2m1(NB_STATES) downto 0);
  type ledDataArray_typ is array(pLed'range) of ledData_typ;
  signal ledData: ledDataArray_typ;


-- clock division counter
-- dividing 66MHz by 2^26 should give a clock of around 1Hz
-- synchronization is not a factor for LED timer counter
  signal clkDiv: std_logic_vector(25 downto 0) := (others=>'0');
  signal powerVect: std_logic_vector(NB_STATES-1 downto 0);


begin

-- power 0 is unused by multiplexer
  powerVect <= clkDiv(clkDiv'high
                      downto clkDiv'high-powerVect'high+3) &
                      "100";

  clockDivision: process(cpuClk)
  begin
    clkIf: if( rising_edge(cpuClk) ) then
      clkDiv <= clkDiv+1;
```

```
    end if clkIf;
  end process clockDivision;


  ledMuxGen: for i in pLed'range generate
    ledMuxi: ledMux
    port map( ledData(i), powerVect, pLed(i) );
  end generate ledMuxGen;


  stateChange: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      rstIf: if( rst='1' ) then
        ledData <= (others=>(others=>'0'));
      elsif( ce='1' ) then  -- rstIf
        ledData(conv_integer(led)) <= state;
      end if rstIf;
    end if clkIf;
  end process stateChange;

end rtl;
```

```
-- Erinyes
--
-- LED interface: multiplexer
--
-- Purpose:
--    LED power multiplexer
--
-- I/O:
--  Internal signals:
--    sel: selection
--    power: power input vector
--    output: output power
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on


entity ledMux is
  port(
    sel: in std_logic_vector;
    power: in std_logic_vector;
    output: out std_logic
  );
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of ledMux: entity is "output";
end ledMux;
```

```
-- Erinyes
--
-- LED interface: multiplexer
--
-- Purpose:
--    LED power multiplexer
--
-- I/O:
--  Internal signals:
--    sel: selection
--    power: power input vector
--    output: output power
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;


architecture rtl of ledMux is

 component obuft_lvcmos33_f_12 is
    port( o: out std_logic; i: in std_logic; t: in std_logic );
  end component obuft_lvcmos33_f_12;
  attribute syn_black_box: boolean;
  attribute syn_black_box of obuft_lvcmos33_f_12: component is true;

  signal output_w: std_logic;
  signal selNotNull: std_logic;

begin


  obufLed: obuft_lvcmos33_f_12
  port map( output, output_w, selNotNull );

  selection: process(sel, power)
  begin
-- keep 0 for off
    zeroOutIf: if( sel/=0 ) then
      output_w <= power(conv_integer(sel));
    end if zeroOutIf;
  end process selection;

  selNotNull <= '1' when sel=0 else '0';

end rtl;
```

```
-- Erinyes
--
-- LED interface
--
-- Purpose:
--    Controls the LEDs
--
-- I/O:
--  Internal signals:
--    address: cpu address bus
--    cpuDataIn: Incoming cpuData bus
--    cpuDataOut: outgoing cpuData bus
--    rw: read/write signal
--    oe: output enable signal
--    cs: chip select signal
--    rst: reset
--    cpuClk: cpu clock
--    clk: internal clock
--  External signals:
---   pLed: LED power signals
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;

entity led is
  generic(
    NB_STATES: integer:= 5;
    TRIG_ADDRESS: std_logic_vector:= x"0"
  );
  port(
    address: in std_logic_vector(22 downto 0);
    cpuDataIn: in std_logic_vector(31 downto 0);
    cpuDataOut: out std_logic_vector(31 downto 0);
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic;
    pLed: out std_logic_vector(11 downto 0)
  );
  attribute black_box_tri_pins : string;
  attribute black_box_tri_pins of led: entity is "pLed[11:0]";
end led;
```

```
-- Erinyes
--
--  LED interface
--
-- Purpose:
--    Controls the LEDs
--
-- I/O:
--  Internal signals:
--    address: cpu address bus
--    cpuDataIn: Incoming cpuData bus
--    cpuDataOut: Outgoing cpuData bus
--    rw: read/write signal
--    oe: output enable signal
--    cs: chip select signal
--    rst: reset
--    cpuClk: cpu clock
--    clk: internal clock
--  External signals:
---   pLed: LED power signals
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;

architecture rtl of led is

  component cpuInt is
    generic(
      constant ADDR_WIDTH: integer := 24;
      constant DATA_WIDTH: integer := 32;
      constant TRIG_ADDRESS_HIGH: std_logic_vector := x"0";
      constant TRIG_ADDRESS_LOW: std_logic_vector := x"0"
    );
    port(
      address: in std_logic_vector(ADDR_WIDTH-1 downto 0);
      cpuDataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
      cpuDataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
      dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
      dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
      rd: out std_logic;
      wr: out std_logic;
      rw: in std_logic;
      oe: in std_logic;
      cs: in std_logic;
      rst: in std_logic;
      cpuClk: in std_logic;
      clk: in std_logic
    );
  end component cpuInt;


  component ledInt is
    generic(
      NB_STATES: integer:= 5;
      NB_DEVICES: integer:= 12
    );
    port(
      led: in std_logic_vector;
```

```
      state: in std_logic_vector;
      ce: in std_logic;
      rst: in std_logic;
      cpuClk: in std_logic;
      clk: in std_logic;
      pLed: out std_logic_vector(NB_DEVICES-1 downto 0)
    );
  end component ledInt;


-- interconnect
  signal dataIn: std_logic_vector(log2m1(NB_STATES) downto 0);
  signal led: std_logic_vector(3 downto 0);
  signal state: std_logic_vector(dataIn'range);
  signal rd, wr: std_logic;


begin

  assert( TRIG_ADDRESS(led'range)=0 )
  report "WARNING: interface was made to have all zeros for bits " &
         integer'image(led'high) & " downto 0 in base address; not " &
         "all LED will be accessible!";

  cpuInt0: cpuInt
  generic map( 23, state'length, TRIG_ADDRESS,
      TRIG_ADDRESS+conv_std_logic_vector(12, TRIG_ADDRESS'length) )
  port map( address, cpuDataIn, cpuDataOut, dataIn, state, rd, wr, rw,
            oe, cs, rst, cpuClk, clk );

  ledInt0: ledInt
  generic map( NB_STATES, 12 )
  port map( led, state, wr, rst, cpuClk, clk, pLed );


  ledDecode: process(cpuClk)
  begin
    clkIf: if( rising_edge(cpuClk) ) then
      led <= address(led'range);
    end if clkIf;
  end process ledDecode;

end rtl;
```

```
-- Erinyes
--
--   LED Interface, testbench
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;
use work.cpu;
use work.simCpu_pak.all;

entity led_tb is
end led_tb;


architecture behav of led_tb is

  component CPU is
    generic(
      CLOCK_PERIOD: time := cpuClockPeriod_c;
      WAIT_STATE: integer := 0
    );
    port(
      address: out address_typ;
      data: inout data_typ;
      rw: out std_logic;
      cs: out std_logic;
      oe: out std_logic;
      rst: in std_logic;
      clk: out std_logic
    );
  end component CPU;


  component led is
    generic(
      NB_STATES: integer:= 5;
      TRIG_ADDRESS: std_logic_vector:= x"0"
    );
    port(
      address: in std_logic_vector(22 downto 0);
      cpuDataIn: in std_logic_vector(31 downto 0);
      cpuDataOut: out std_logic_vector(31 downto 0);
      rw: in std_logic;
      oe: in std_logic;
      cs: in std_logic;
      rst: in std_logic;
      cpuClk: in std_logic;
      clk: in std_logic;
      pLed: out std_logic_vector(11 downto 0)
    );
  end component led;



-- some constants
  constant fpgaClockPeriod_c: time:= 8 ns;
  constant nbLeds_c: integer:= 12;
  constant devAddr_c: address_typ:= "010"&x"12000";
```

```
-- interconnect
  signal address: address_typ;
  signal dataIn, dataOut: data_typ;
  signal rw, cs, rst, cpuRst, cpuClk, oe: std_logic;
  signal pLed: std_logic_vector(nbLeds_c-1 downto 0);
  signal clk: std_logic:= '0';

begin

  cpu0: cpu
  port map( address, dataIn, rw, cs, oe, cpuRst, cpuClk );

  ledInt0: led
  generic map( 6, devAddr_c )
  port map( address, dataIn, dataOut, rw, oe, cs,
            rst, cpuClk, clk, pLed );


  clk <= not clk after fpgaClockPeriod_c/2;
  cpuRst <= not rst;


  test: process
  begin
    rst <= '1';
    wait for fpgaClockPeriod_c;
    rst <= '0';
    wait for fpgaClockPeriod_c;

    report "Sending data to LED 0.";
    cpuWrite(devAddr_c, x"00000003", cpuSig_rec);

    report "Sending data to LED 1.";
    cpuWrite(devAddr_c+1, x"00000004", cpuSig_rec);

    report "Sending data to LED 2.";
    cpuWrite(devAddr_c+2, x"00000005", cpuSig_rec);

    report "Sending data to LED 5.";
    cpuWrite(devAddr_c+5, x"00000000", cpuSig_rec);

    report "Sending data to LED 7.";
    cpuWrite(devAddr_c+7, x"00000001", cpuSig_rec);

    report "Sending data to LED 10.";
    cpuWrite(devAddr_c+10, x"00000002", cpuSig_rec);
    wait;

  end process test;

end behav;
```

```
-- Erinyes
--
-- Arbiter package
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

package arbiter_pak is

  subtype ssramIntState_typ is std_logic_vector(1 downto 0);
  constant unused_c: ssramIntState_typ:= "00";
  constant port0_c: ssramIntState_typ:= "01";
  constant port1_c: ssramIntState_typ:= "10";
  constant port2_c: ssramIntState_typ:= "11";

end package arbiter_pak;
```

```
-- Erinyes
--
-- Arbiter package
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

package body arbiter_pak is

end arbiter_pak;
```

```
-- Erinyes
--
-- Basic memory element
--
-- Purpose:
--  Building block for memory components
--
-- I/O:
--   dp0: output port 0
--   dCe: clock enable port 0
--   sp0: output port 1
--   sCe: clock enable port 1
--   a: writing address (also sp0 address)
--   d: writing data
--   dpra: dp0 address
--   clk: port 0 clock
--   wClk: write clock
--   we: write wnable
--
-- NOTE:
--  Synplify is used to determine which RAM primitive to use.
--  PRIMITIVE can be one of "block_ram", "select_ram", "registers" or "no_rx_check"
--
--   block_ram: Block RAM
--   select_ram: LUT
--   registers: LUT through logic
--   no_rx_check: Block RAM without conflict resolving
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library synplify;
use synplify.all;

entity baseMem is
  generic(
    WIDTH: integer:= 8;
    DEPTH: integer:= 16;
    PRIMITIVE: string:= "select_ram"
  );
  port (
    dp0 : out std_logic_vector;
    dCe: in std_logic;
    sp0 : out std_logic_vector;
    sCe: in std_logic;
    a : in std_logic_vector;
    d : in std_logic_vector;
    dpra : in std_logic_vector;
    clk: in std_logic;
    wClk : in std_logic;
    we : in std_logic
  );
end baseMem;
```

```
-- Erinyes
--
-- Basic memory element
--
-- Purpose:
--  Building block for memory components
--
-- I/O:
--   dp0: output port 0
--   dCe: clock enable port 0
--   sp0: output port 1
--   sCe: clock enable port 1
--   a: writing address (also sp0 address)
--   d: writing data
--   dpra: dp0 address
--   clk: port 0 clock
--   wClk: write clock
--   we: write wnable
--
-- NOTE:
--  Synplify is used to determine which RAM primitive to use.
--  PRIMITIVE can be one of "block_ram", "select_ram", "registers" or "no_rx_check"
--
--   block_ram: Block RAM
--   select_ram: LUT
--   registers: LUT through logic
--   no_rx_check: Block RAM without conflict resolving
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003
architecture behav of baseMem is
-- memory array declaration
--   This is invalid VHDL for synthesis.  Synopsys will transform this
-- code for synthesis on target FPGA using the specified primitive.
  subtype cell_typ is std_logic_vector(WIDTH-1 downto 0);
  type memArray_typ is array(DEPTH-1 downto 0) of cell_typ;
  signal memArray_a: memArray_typ;

-- synopsys attributes for synthesis
  attribute syn_ramstyle: string;
  attribute syn_ramstyle of memArray_a: signal is PRIMITIVE;

begin
  sp0Port: process(wClk)
  begin
    clkIf: if( rising_edge(wClk) ) then
      ceIf: if( sCe='1' ) then
        weIf: if( we='1' ) then
          memArray_a(conv_integer(a)) <= d;
        else -- weIf
          sp0 <= memArray_a(conv_integer(a));
        end if weIf;
      end if ceIf;
    end if clkIf;
  end process sp0Port;

  dp0Port: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      ceIf: if( dCe='1' ) then
        dp0 <= memArray_a(conv_integer(dpra));
      end if ceIf;
    end if clkIf;
  end process dp0Port;

end behav;
```

```
-- Erinyes
--
-- FIFO buffer
--
-- I/O:
--   wData: writing Data
--   rData: reading Data
--   mty: empty flag
--   fl: fifo full flag
--   we: write wnable
--   rd: output enable
--   rst: reset signal
--   clk: clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;

entity fifo is
  generic (
    WIDTH: integer:= 8;
    DEPTH: integer:= 16
  );
  port (
    wData: in std_logic_vector(WIDTH-1 downto 0);
    rData: out std_logic_vector(WIDTH-1 downto 0);
    mty: out std_logic;
    fl: out std_logic;
    we: in std_logic;
    rd: in std_logic;
    rst: in std_logic;
    clk: in std_logic
  );
end fifo;
```

```
-- Erinyes
--
-- FIFO buffer
--
-- I/O:
--   wData: writing Data
--   rData: reading Data
--   mty: empty flag
--   fl: fifo full flag
--   we: write wnable
--   rd: output enable
--   rst: reset signal
--   clk: clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture rtl of fifo is

  component baseMem is
    generic (
      WIDTH: integer:= 8;
      DEPTH: integer:= 16;
      PRIMITIVE: string:= "select_ram"
    );
    port (
      dp0 : out std_logic_vector;
      dCe: in std_logic;
      sp0 : out std_logic_vector;
      sCe: in std_logic;
      a : in std_logic_vector;
      d : in std_logic_vector;
      dpra : in std_logic_vector;
      clk: in std_logic;
      wClk : in std_logic;
      we : in std_logic
    );
    end component baseMem;


-- memory pointers
  signal rdPtr, wrPtr: std_logic_vector(log2m1(DEPTH) downto 0);

-- internal signals
  signal sp0: std_logic_vector(wData'range);
  signal rData_w: std_logic_vector(rData'range);

-- flags management arithmetics
  signal subs: std_logic_vector(rdPtr'length downto 0);
  signal oldSign: std_logic;


begin

  ram0: baseMem
  generic map ( WIDTH, 2**rdPtr'length, "select_ram" )
  port map ( rData, rd, sp0, we, wrPtr, wData,
             rdPtr, clk, clk, we );


-- substract rPtr from wPtr
  subs <= (wrPtr(wrPtr'high)&wrPtr) -
          (rdPtr(rdPtr'high)&rdPtr);


  readPtr: process(clk)
```

```
  begin
    clkIf: if( rising_edge(clk) ) then
      rstIf: if( rst='1' ) then
        rdPtr <= (others=>'0');
      elsif( rd='1' ) then -- rstIf
        rdPtr <= rdPtr + 1;
      end if rstIf;
    end if clkIf;
  end process readPtr;


  writePtr: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      rstIf: if( rst='1' ) then
        wrPtr <= (others=>'0');
      elsif( we='1' ) then -- rstIf
        wrPtr <= wrPtr + 1;
      end if rstIf;
    end if clkIf;
  end process writePtr;


  oldSignKeep: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      oldSign <= subs(subs'high);
    end if clkIf;
  end process oldSignKeep;


  fullEmpty: process(rdPtr, wrPtr, oldSign)
  begin
    cmpIf: if( rdPtr=wrPtr ) then
      mty <= not oldSign;
      fl <= oldSign;
    else -- cmpIf
      mty <= '0';
      fl <= '0';
    end if cmpIf;
  end process fullEmpty;

end rtl;
```

```
-- Erinyes
--
-- FIFO buffer testbench
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.utils_pak.all;

entity fifo_tb is
end fifo_tb;


architecture behav of fifo_tb is

  component fifo is
    generic (
      WIDTH: integer:= 8;
      DEPTH: integer:= 16
    );
    port (
      wData: in std_logic_vector(WIDTH-1 downto 0);
      rData: out std_logic_vector(WIDTH-1 downto 0);
      mty: out std_logic;
      we: in std_logic;
      rd: in std_logic;
      rst: in std_logic;
      clk: in std_logic
    );
  end component;


-- utility constants
  constant fpgaPeriod_c: time:= 8 ns;
  constant fifoWidth_c: integer:= 24;
  constant fifoDepth_c: integer:= 18;


-- fifo signals
  type fifoSig_typ is record
    wData: std_logic_vector(fifoWidth_c-1 downto 0);
    rData: std_logic_vector(fifoWidth_c-1 downto 0);
    we: std_logic;
    rd: std_logic;
  end record fifoSig_typ;
  signal fifoSig_rec: fifoSig_typ;

-- internal signals
  signal clk: std_logic:= '0';
  signal rst: std_logic;
  signal mty: std_logic;


-- utility functions
  procedure send(constant d: in std_logic_vector(fifoSig_rec.wData'range);
                 signal fifoSig_rec: out fifoSig_typ) is
  begin
    fifoSig_rec.wData <= d;
    fifoSig_rec.rd <= '0';
    fifoSig_rec.we <= '1';
```

```
    wait for fpgaPeriod_c;
    fifoSig_rec.we <= '0';
  end procedure send;

  procedure get(signal fifoSig_rec: inout fifoSig_typ) is
  begin
    fifoSig_rec.we <= '0';
    fifoSig_rec.rd <= '1';
    wait for fpgaPeriod_c;
    fifoSig_rec.rd <= '0';
  end procedure get;

  procedure getSend(constant d: in std_logic_vector(fifoSig_rec.wData'range);
                    signal fifoSig_rec: out fifoSig_typ) is
  begin
    fifoSig_rec.wData <= d;
    fifoSig_rec.rd <= '1';
    fifoSig_rec.we <= '1';
    wait for fpgaPeriod_c;
    fifoSig_rec.we <= '0';
    fifoSig_rec.rd <= '0';
  end procedure getSend;

begin

  fifo0: fifo
  generic map( fifoWidth_c, fifoDepth_c )
  port map( fifoSig_rec.wData, fifoSig_rec.rData, mty,
            fifoSig_rec.we, fifoSig_rec.rd, rst, clk );

  clk <= not clk after fpgaPeriod_c/2;

  test: process
  begin
    rst <= '1';
    fifoSig_rec.rData <= (others=>'Z');
    wait for fpgaPeriod_c;
    rst <= '0';

    send(conv_std_logic_vector(37, fifoWidth_c),
        fifoSig_rec);
    get(fifoSig_rec);

    fill: for i in 1 to fifoDepth_c loop
      send(conv_std_logic_vector(i, fifoWidth_c),
           fifoSig_rec);
    end loop fill;

    roll: for i in 1 to 2*fifoDepth_c loop
      getSend(conv_std_logic_vector(i, fifoWidth_c),
           fifoSig_rec);
    end loop roll;

  end process test;

end behav;
```

```
-- Erinyes
--
-- Memory arbiter: switch matrix
--
-- Purpose:
--   Connects SSRAM interface to selected port
--
-- I/O:
--   port0addr: port0 address
--   port0dataAin: port0 data A input
--   port0dataBin: port0 data B input
--   port0dataAout: port0 data A output
--   port0dataBout: port0 data B output
--   port0rdy: port0 data ready
--   port0byte: port0 byte write enable
--   port0rw: port0 rw signal
--   port0ce: port0 clock enable
--   port1addr: port0 address
--   port1dataAin: port1 data A input
--   port1dataBin: port1 data B input
--   port1dataAout: port1 data A output
--   port1dataBout: port1 data B output
--   port1rdy: port0 data ready
--   port1byte: port0 byte write enable
--   port1rw: port0 rw signal
--   port1ce: port0 clock enable
--   port2addr: port0 address
--   port2dataAin: port2 data A input
--   port2dataBin: port2 data B input
--   port2dataAout: port2 data A output
--   port2dataBout: port2 data B output
--   port2rdy: port0 data ready
--   port2byte: port0 byte write enable
--   port2rw: port0 rw signal
--   port2ce: port0 clock enable
--   clk: interface clock
--   ssramState: ssram status enumeration
--   zz: ssram sleep signal
--   addr: ssram address port
--   aIn: ssram data A input
--   bIn: ssram data B input
--   aOut: ssram data A output
--   bOut ssram data B output
--   rdy: ssram data ready
--   byte: ssram byte write enable
--   rw: ssram read/write
--   ce: ssram clock enable
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

-- synthesis translate_off
library unisim;
use unisim.bufe;
-- synthesis translate_on

library work;
use work.arbiter_pak.all;

entity arbiterSwitch is
  port (
    port0addr: in std_logic_vector;
    port0dataAin: in std_logic_vector;
```

```
    port0dataBin: in std_logic_vector;
    port0dataAout: out std_logic_vector;
    port0dataBout: out std_logic_vector;
    port0rdy: out std_logic;
    port0byte: in std_logic_vector;
    port0rw: in std_logic;
    port0ce: in std_logic;

    port1Addr: in std_logic_vector;
    port1DataAin: in std_logic_vector;
    port1DataBin: in std_logic_vector;
    port1DataAout: out std_logic_vector;
    port1DataBout: out std_logic_vector;
    port1rdy: out std_logic;
    port1byte: in std_logic_vector;
    port1rw: in std_logic;
    port1ce: in std_logic;

    port2Addr: in std_logic_vector;
    port2DataAin: in std_logic_vector;
    port2DataBin: in std_logic_vector;
    port2DataAout: out std_logic_vector;
    port2DataBout: out std_logic_vector;
    port2rdy: out std_logic;
    port2byte: in std_logic_vector;
    port2rw: in std_logic;
    port2ce: in std_logic;

    clk: in std_logic;
    ssramState: in  ssramIntState_typ;

    zz: out std_logic;
    addr: out std_logic_vector;
    aIn: out std_logic_vector;
    bIn: out std_logic_vector;
    aOut: in std_logic_vector;
    bOut: in std_logic_vector;
    rdy: in std_logic;
    byte: out std_logic_vector;
    rw: out std_logic;
    ce: out std_logic
  );
end arbiterSwitch;
```

```
-- Erinyes
--
-- Memory arbiter: switch matrix
--
-- Purpose:
--   Connects SSRAM interface to selected port
--
-- I/O:
--   port0addr: port0 address
--   port0dataAin: port0 data A input
--   port0dataBin: port0 data B input
--   port0dataAout: port0 data A output
--   port0dataBout: port0 data B output
--   port0rdy: port0 data ready
--   port0byte: port0 byte write enable
--   port0rw: port0 rw signal
--   port0ce: port0 clock enable
--   port1addr: port0 address
--   port1dataAin: port1 data A input
--   port1dataBin: port1 data B input
--   port1dataAout: port1 data A output
--   port1dataBout: port1 data B output
--   port1rdy: port0 data ready
--   port1byte: port0 byte write enable
--   port1rw: port0 rw signal
--   port1ce: port0 clock enable
--   port2addr: port0 address
--   port2dataAin: port2 data A input
--   port2dataBin: port2 data B input
--   port2dataAout: port2 data A output
--   port2dataBout: port2 data B output
--   port2rdy: port0 data ready
--   port2byte: port0 byte write enable
--   port2rw: port0 rw signal
--   port2ce: port0 clock enable
--   ssramState: ssram status enumeration
--   zz: ssram sleep signal
--   addr: ssram address port
--   aIn: ssram data A input
--   bIn: ssram data B input
--   aOut: ssram data A output
--   bOut ssram data B output
--   rdy: ssram data ready
--   byte: ssram byte write enable
--   rw: ssram read/write
--   ce: ssram clock enable
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture rtl of arbiterSwitch is

  component bufeBus is
    port( o: out std_logic_vector; e: in std_logic; i: in std_logic_vector );
  end component bufeBus;

  component bufe is
    port( o: out std_logic; e: in std_logic; i: in std_logic );
  end component bufe;

  attribute syn_black_box: boolean;
  attribute syn_black_box of bufe: component is true;
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of bufe: component is "o";
```

```
-- enable signals
  signal port0enable: std_logic := '0';
  signal port1enable: std_logic := '0';
  signal port2enable: std_logic := '0';


begin

  enableSelect: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      selectCase: case ssramState is
        when port0_c =>
          port0enable <= '1';
          port1enable <= '0';
          port2enable <= '0';
        when port1_c =>
          port0enable <= '0';
          port1enable <= '1';
          port2enable <= '0';
        when port2_c =>
          port0enable <= '0';
          port1enable <= '0';
          port2enable <= '1';
        when others =>
          port0enable <= '0';
          port1enable <= '0';
          port2enable <= '0';
      end case selectCase;
    end if clkIf;
  end process enableSelect;

  zz <= '1' when ssramState=unused_c else '0';


-- port0
  port0addrBuf: bufeBus
  port map( addr, port0enable, port0addr );
  port0dataAinBuf: bufeBus
  port map( aIn, port0enable, port0dataAin );
  port0dataBinBuf: bufeBus
  port map( bIn, port0enable, port0dataBin );
  port0dataAoutBuf: bufeBus
  port map( port0dataAout, port0enable, aOut );
  port0dataBoutBuf: bufeBus
  port map( port0dataBout, port0enable, bOut );
  port0rdyBuf: bufe
  port map( port0rdy, port0enable, rdy );
  port0byteBuf: bufeBus
  port map( byte, port0enable, port0byte );
  port0rwBuf: bufe
  port map( rw, port0enable, port0rw );
  port0ceBuf: bufe
  port map( ce, port0enable, port0ce );

-- port1
  port1addrBuf: bufeBus
  port map( addr, port1enable, port1addr );
  port1dataAinBuf: bufeBus
  port map( aIn, port1enable, port1dataAin );
  port1dataBinBuf: bufeBus
  port map( bIn, port1enable, port1dataBin );
  port1dataAoutBuf: bufeBus
  port map( port1dataAout, port1enable, aOut );
  port1dataBoutBuf: bufeBus
  port map( port1dataBout, port1enable, bOut );
  port1rdyBuf: bufe
```

```
  port map( port1rdy, port1enable, rdy );
  port1byteBuf: bufeBus
  port map( byte, port1enable, port1byte );
  port1rwBuf: bufe
  port map( rw, port1enable, port1rw );
  port1ceBuf: bufe
  port map( ce, port1enable, port1ce );

-- port2
  port2addrBuf: bufeBus
  port map( addr, port2enable, port2addr );
  port2dataAinBuf: bufeBus
  port map( aIn, port2enable, port2dataAin );
  port2dataBinBuf: bufeBus
  port map( bIn, port2enable, port2dataBin );
  port2dataAoutBuf: bufeBus
  port map( port2dataAout, port2enable, aOut );
  port2dataBoutBuf: bufeBus
  port map( port2dataBout, port2enable, bOut );
  port2rdyBuf: bufe
  port map( port2rdy, port2enable, rdy );
  port2byteBuf: bufeBus
  port map( byte, port2enable, port2byte );
  port2rwBuf: bufe
  port map( rw, port2enable, port2rw );
  port2ceBuf: bufe
  port map( ce, port2enable, port2ce );

end rtl;
```

```
-- Erinyes
--
-- Memory arbiter
--
-- Purpose:
--   Controls access to external memory
--
-- I/O:
--
--  address: cpu address bus
--  cpuDataIn: incoming cpu data bus
--  cpuDataOut: outgoing cpu data bus
--  rw: cpu rw signal
--  oe: cpu output enable
--  cs: cpu chip select
--  cpuClk: cpu clock
--
--  pXaddr: port x address
--  pXdataIn: port x data input
--  pXdataOut: port x data output
--  pXrdy: port x data ready
--  pXrw: port x rw signal
--  pXce: port x clock enable signal
--
--  rst: reset input
--  clk: clock input
--
--  saX: ssram X address lines
--  dqaX: ssram X data line a
--  dqbX: ssram X data line b
--  bwaNX: ssram X byte write byte a
--  bwbNX: ssram X byte write byte b
--  bweNX: ssram X byte write enable
--  gwNX: ssram X global write
--  advNX: ssram X synchronous address advance
--  adscNX: ssram X synchronous address status controller
--  adspNX: ssram X synchronous address status processor
--  ceNX: ssram X clock enable
--  oeNX: ssram X output enable
--  zzX: ssram X snooze mode enable
--  cko: ssrams output clock signal
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.arbiter_pak.all;
use work.cpu_pak.all;
use work.utils_pak.all;

-- synthesis translate_off
library unisim;
use unisim.dcm;
-- synthesis translate_on

entity arbiter is
  generic (
    SSRAM0_CTRL_ADDRESS: std_logic_vector:= "110"&x"01000";
    SSRAM1_CTRL_ADDRESS: std_logic_vector:= "110"&x"01004";
    SSRAM2_CTRL_ADDRESS: std_logic_vector:= "110"&x"01008";
    CPU_CTRL_ADDRESS: std_logic_vector:= "110"&x"01010";
```

```
    PORT1_CTRL_ADDRESS: std_logic_vector:= "110"&x"01014";
    PORT2_CTRL_ADDRESS: std_logic_vector:= "110"&x"01018";
    CPU_A_WIDTH: integer:= 9;
    CPU_B_WIDTH: integer:= 9;
    PORT1_A_WIDTH: integer:= 9;
    PORT1_B_WIDTH: integer:= 9;
    PORT2_A_WIDTH: integer:= 9;
    PORT2_B_WIDTH: integer:= 9
  );
  port (
    address: in std_logic_vector(22 downto 0);
    cpuDataIn: in std_logic_vector(31 downto 0);
    cpuDataOut: out std_logic_vector(31 downto 0);
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    cpuClk: in std_logic;

    p1addr: in std_logic_vector(22 downto 0);
    p1dataIn: in std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
    p1dataOut: out std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
    p1rdy: out std_logic;
    p1rw: in std_logic;
    p1ce: in std_logic;

    p2addr: in std_logic_vector(22 downto 0);
    p2dataIn: in std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
    p2dataOut: out std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
    p2rdy: out std_logic;
    p2rw: in std_logic;
    p2ce: in std_logic;

    rst: in std_logic;
    clk: in std_logic;

-- SSRAM0
    sa0: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa0: inout std_logic_vector(8 downto 0) := (others=>'Z');
    dqb0: inout std_logic_vector(8 downto 0) := (others=>'Z');

    bwaN0: out std_logic;
    bwbN0: out std_logic;
    bweN0: out std_logic;
    gwN0: out std_logic;

    advN0: out std_logic;
    adscN0: out std_logic;
    adspN0: out std_logic;

    ceN0: out std_logic_vector(3 downto 0);
    oeN0: out std_logic_vector(3 downto 0);
    zz0: out std_logic;
    cko0: out std_logic;

-- SSRAM1
    sa1: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa1: inout std_logic_vector(8 downto 0) := (others=>'Z');
    dqb1: inout std_logic_vector(8 downto 0) := (others=>'Z');

    bwaN1: out std_logic;
    bwbN1: out std_logic;
    bweN1: out std_logic;
    gwN1: out std_logic;

    advN1: out std_logic;
    adscN1: out std_logic;
    adspN1: out std_logic;
```

```
    ceN1: out std_logic_vector(3 downto 0);
    oeN1: out std_logic_vector(3 downto 0);
    zz1: out std_logic;
    cko1: out std_logic;

-- SSRAM2
    sa2: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa2: inout std_logic_vector(8 downto 0) := (others=>'Z');
    dqb2: inout std_logic_vector(8 downto 0) := (others=>'Z');

    bwaN2: out std_logic;
    bwbN2: out std_logic;
    bweN2: out std_logic;
    gwN2: out std_logic;

    advN2: out std_logic;
    adscN2: out std_logic;
    adspN2: out std_logic;

    ceN2: out std_logic_vector(3 downto 0);
    oeN2: out std_logic_vector(3 downto 0);
    zz2: out std_logic;
    cko2: out std_logic
  );
end arbiter;
```

```
-- Erinyes
--
-- Memory arbiter
--
-- Purpose:
--   Controls access to external memory
--
-- I/O:
--
--  address: cpu address bus
--  cpuDataIn: incoming cpu data bus
--  cpuDataOut: outgoing cpu data bus
--  rw: cpu rw signal
--  oe: cpu output enable
--  cs: cpu chip select
--  cpuClk: cpu clock
--
--  pXaddr: port x address
--  pXdataIn: port x data input
--  pXdataOut: port x data output
--  pXrdy: port x data ready
--  pXrw: port x rw signal
--  pXce: port x clock enable signal
--
--  rst: reset input
--  clk: clock input
--
--  saX: ssram X address lines
--  dqaX: ssram X data line a
--  dqbX: ssram X data line b
--  bwaNX: ssram X byte write byte a
--  bwbNX: ssram X byte write byte b
--  bweNX: ssram X byte write enable
--  gwNX: ssram X global write
--  advNX: ssram X synchronous address advance
--  adscNX: ssram X synchronous address status controller
--  adspNX: ssram X synchronous address status processor
--  ceNX: ssram X clock enable
--  oeNX: ssram X output enable
--  zzX: ssram X snooze mode enable
--  ckoX: ssram X output clock signal
--
-- cpuAddr: cpu address
-- cpuData: cpu data
-- cpuRW: cpu rw signal
-- cpuCs: cpu chip select
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of arbiter is

  component cpuInt is
    generic(
      constant ADDR_WIDTH: integer := 24;
      constant CPU_DATA_WIDTH: integer := 32;
      constant DATA_WIDTH: integer := 32;
      constant TRIG_ADDRESS_HIGH: std_logic_vector := x"0";
      constant TRIG_ADDRESS_LOW: std_logic_vector := x"0"
    );
    port(
      address: in std_logic_vector(ADDR_WIDTH-1 downto 0);
      cpuDataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
      cpuDataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
      dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
      dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
```

```
    rd: out std_logic;
    wr: out std_logic;
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic
  );
end component cpuInt;


component ssramInt is
  generic(
    constant DATA_A_WIDTH: integer := 9;
    constant DATA_B_WIDTH: integer := 9;
    constant ADDR_WIDTH: integer := 19;
    constant DCM_PHASE_SHIFT: integer := 0
  );
  port(
    address: in std_logic_vector(ADDR_WIDTH-1 downto 0);
    cSel: in std_logic_vector(3 downto 0);
    dataAIn: in std_logic_vector(DATA_A_WIDTH-1 downto 0);
    dataBIn: in std_logic_vector(DATA_B_WIDTH-1 downto 0);
    dataAOut: out std_logic_vector(DATA_A_WIDTH-1 downto 0);
    dataBOut: out std_logic_vector(DATA_B_WIDTH-1 downto 0);
    rdy: out std_logic;
    byte: in std_logic_vector(1 downto 0);
    rw: in std_logic;
    ce: in std_logic;
    addrPtr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
    sleep: in std_logic;
    clk: in std_logic;

    sa: out std_logic_vector(18 downto 0);
    dqa: inout std_logic_vector(8 downto 0);
    dqb: inout std_logic_vector(8 downto 0);

    bwaN: out std_logic;
    bwbN: out std_logic;
    bweN: out std_logic;
    gwN: out std_logic;

    advN: out std_logic;
    adscN: out std_logic;
    adspN: out std_logic;

    ceN: out std_logic_vector(3 downto 0);
    oeN: out std_logic_vector(3 downto 0);
    zz: out std_logic;
    cko: out std_logic
  );
end component ssramInt;


component arbiterSwitch is
  port (
    port0Addr: in std_logic_vector;
    port0DataAin: in std_logic_vector;
    port0DataBin: in std_logic_vector;
    port0DataAout: out std_logic_vector;
    port0DataBout: out std_logic_vector;
    port0rdy: out std_logic;
    port0byte: in std_logic_vector;
    port0rw: in std_logic;
    port0ce: in std_logic;
```

```
      port1Addr: in std_logic_vector;
      port1DataAin: in std_logic_vector;
      port1DataBin: in std_logic_vector;
      port1DataAout: out std_logic_vector;
      port1DataBout: out std_logic_vector;
      port1rdy: out std_logic;
      port1byte: in std_logic_vector;
      port1rw: in std_logic;
      port1ce: in std_logic;

      port2Addr: in std_logic_vector;
      port2DataAin: in std_logic_vector;
      port2DataBin: in std_logic_vector;
      port2DataAout: out std_logic_vector;
      port2DataBout: out std_logic_vector;
      port2rdy: out std_logic;
      port2byte: in std_logic_vector;
      port2rw: in std_logic;
      port2ce: in std_logic;

      clk: in std_logic;
      ssramState: in  ssramIntState_typ;

      zz: out std_logic;
      addr: out std_logic_vector;
      aIn: out std_logic_vector;
      bIn: out std_logic_vector;
      aOut: in std_logic_vector;
      bOut: in std_logic_vector;
      rdy: in std_logic;
      byte: out std_logic_vector;
      rw: out std_logic;
      ce: out std_logic
    );
  end component arbiterSwitch;

  constant addrWidth_c: integer:= 19;
  constant dataAwidth_c: integer:= 9;
  constant dataBwidth_c: integer:= 9;


-- ssram0 signals
  signal ssram0state: ssramIntState_typ := (others=>'0');
  signal cSel0: std_logic_vector(3 downto 0);
  signal ssram0ctrl: std_logic_vector(ssram0state'length+
                                      cSel0'length-1 downto 0);
  signal addr0: std_logic_vector(addrWidth_c-1 downto 0);
  signal a0in, a0out: std_logic_vector(dataAwidth_c-1 downto 0);
  signal b0in, b0out: std_logic_vector(dataBwidth_c-1 downto 0);
  signal rdy0: std_logic;
  signal byte0: std_logic_vector(1 downto 0);
  signal rw0, ce0: std_logic;
  signal aPtr0: std_logic_vector(addr0'range);
  signal slp0: std_logic;


-- ssram1 signals
  signal ssram1state: ssramIntState_typ := (others=>'0');
  signal cSel1: std_logic_vector(3 downto 0);
  signal ssram1ctrl: std_logic_vector(ssram1state'length+
                                      cSel1'length-1 downto 0);
  signal addr1: std_logic_vector(addrWidth_c-1 downto 0);
  signal a1in, a1out: std_logic_vector(dataAwidth_c-1 downto 0);
  signal b1in, b1out: std_logic_vector(dataBwidth_c-1 downto 0);
  signal rdy1: std_logic;
  signal byte1: std_logic_vector(1 downto 0);
```

```
  signal rw1, ce1: std_logic;
  signal aPtr1: std_logic_vector(addr1'range);
  signal slp1: std_logic;


-- ssram2 signals
  signal ssram2state: ssramIntState_typ := (others=>'0');
  signal cSel2: std_logic_vector(3 downto 0);
  signal ssram2ctrl: std_logic_vector(ssram2state'length+
                                      cSel2'length-1 downto 0);
  signal addr2: std_logic_vector(addrWidth_c-1 downto 0);
  signal a2in, a2out: std_logic_vector(dataAwidth_c-1 downto 0);
  signal b2in, b2out: std_logic_vector(dataBwidth_c-1 downto 0);
  signal rdy2: std_logic;
  signal byte2: std_logic_vector(1 downto 0);
  signal rw2, ce2: std_logic;
  signal aPtr2: std_logic_vector(addr2'range);
  signal slp2: std_logic;


-- port CPU
  signal cpuPortIn: std_logic_vector(CPU_A_WIDTH+CPU_B_WIDTH-1 downto 0);
  signal cpuPortOut: std_logic_vector(cpuPortIn'range);
  signal cpuDataAin, cpuDataAout: std_logic_vector(CPU_A_WIDTH-1 downto 0);
  signal cpuDataBin, cpuDataBout: std_logic_vector(CPU_B_WIDTH-1 downto 0);
  signal cpuByte: std_logic_vector(byte0'range);
  signal cpuRdy, cpuCeDecoded: std_logic;
  signal cpuRd, cpuWr, cpuRw: std_logic;


-- port1
  signal port1dataIn: std_logic_vector(p1dataIn'range);
  signal p1dataAin, p1dataAout: std_logic_vector(a0in'range);
  signal p1dataBin, p1dataBout: std_logic_vector(b0in'range);
  signal port1byte: std_logic_vector(byte1'range);
  signal port1ceDecoded: std_logic;


-- port2
  signal port2dataIn: std_logic_vector(p2dataIn'range);
  signal p2dataAin, p2dataAout: std_logic_vector(a0in'range);
  signal p2dataBin, p2dataBout: std_logic_vector(b0in'range);
  signal port2byte: std_logic_vector(byte2'range);
  signal port2ceDecoded: std_logic;


-- CPU interfaces
  signal ssram0rd, ssram0wr: std_logic;
  signal ssram1rd, ssram1wr: std_logic;
  signal ssram2rd, ssram2wr: std_logic;
  signal port0rd, port0wr: std_logic;
  signal port1rd, port1wr: std_logic;
  signal port2rd, port2wr: std_logic;


begin

  ssram0: ssramInt
  generic map( dataAwidth_c, dataBwidth_c, addrWidth_c )
  port map( addr0, cSel0, a0In, b0In, a0Out, b0Out, rdy0, byte0, rw0, ce0, aPtr0, slp0,
clk,
            sa0, dqa0, dqb0,
            bwaN0, bwbN0, bweN0, gwN0,
            advN0, adscN0, adspN0,
            ceN0, oeN0, zz0, cko0 );

  ssram1: ssramInt
```

```
  generic map( dataAwidth_c, dataBwidth_c, addrWidth_c )
  port map( addr1, cSel1, a1In, b1In, a1Out, b1Out, rdy1, byte1, rw1, ce1, aPtr1, slp1,
clk,
            sa1, dqa1, dqb1,
            bwaN1, bwbN1, bweN1, gwN1,
            advN1, adscN1, adspN1,
            ceN1, oeN1, zz1, cko1 );

  ssram2: ssramInt
  generic map( dataAwidth_c, dataBwidth_c, addrWidth_c )
  port map( addr2, cSel2, a2In, b2In, a2Out, b2Out, rdy2, byte2, rw2, ce2, aPtr2, slp2,
clk,
            sa2, dqa2, dqb2,
            bwaN2, bwbN2, bweN2, gwN2,
            advN2, adscN2, adspN2,
            ceN2, oeN2, zz2, cko2 );

-- a 3x3x18 multiplexer would be too large and slow.
-- selection will be made through internal 3-state buffers

  p1dataAin <= p1dataIn(p1dataAin'range);
  p1dataBin <= p1dataIn(p1dataIn'high downto p1dataAin'length);
  p1dataOut <= p1dataBout & p1dataAout;
  p2dataAin <= p2dataIn(p2dataAin'range);
  p2dataBin <= p2dataIn(p2dataIn'high downto p2dataAin'length);
  p2dataOut <= p2dataBout & p2dataAout;
  ssram0Switch: arbiterSwitch
  port map( address(addr0'range),
            cpuDataAin, cpuDataBin, cpuDataAout, cpuDataBout,
            cpuRdy, cpuByte, cpuRw, cpuCeDecoded,
            p1addr(addr1'range),
            p1dataAin, p1dataBin, p1dataAout, p1dataBout,
            p1rdy, port1byte, p1rw, port1ceDecoded,
            p2addr(addr2'range),
            p2dataAin, p2dataBin, p2dataAout, p2dataBout,
            p2rdy, port2byte, p2rw, port2ceDecoded,
            clk, ssram0state, slp0, addr0, a0in, b0in, a0out, b0out,
            rdy0, byte0, rw0, ce0 );

  ssram1Switch: arbiterSwitch
  port map( address(addr0'range),
            cpuDataAin, cpuDataBin, cpuDataAout, cpuDataBout,
            cpuRdy, cpuByte, cpuRw, cpuCeDecoded,
            p1addr(addr1'range),
            p1dataAin, p1dataBin, p1dataAout, p1dataBout,
            p1rdy, port1byte, p1rw, port1ceDecoded,
            p2addr(addr2'range),
            p2dataAin, p2dataBin, p2dataAout, p2dataBout,
            p2rdy, port2byte, p2rw, port2ceDecoded,
            clk, ssram1state, slp1, addr1, a1in, b1in, a1out, b1out,
            rdy1, byte1, rw1, ce1 );

  ssram2Switch: arbiterSwitch
  port map( address(addr0'range),
            cpuDataAin, cpuDataBin, cpuDataAout, cpuDataBout,
            cpuRdy, cpuByte, cpuRw, cpuCeDecoded,
            p1addr(addr1'range),
            p1dataAin, p1dataBin, p1dataAout, p1dataBout,
            p1rdy, port1byte, p1rw, port1ceDecoded,
            p2addr(addr2'range),
            p2dataAin, p2dataBin, p2dataAout, p2dataBout,
            p2rdy, port2byte, p2rw, port2ceDecoded,
            clk, ssram2state, slp2, addr2, a2in, b2in, a2out, b2out,
            rdy2, byte2, rw2, ce2 );
```

```
-- 3 very first bits define the ssram base address
-- The total address span is from 000 0000 - 5ff ffff
-- On 23 bits, this means that the 3 MSB from 000 to 101 are reserved
-- for ssram memory

-- asynchronous address decode
  port1ceDecoded <= p1ce when p1Addr(p1addr'high downto p2addr'high-2)<"110" else '0';
  port2ceDecoded <= p2ce when p2Addr(p1addr'high downto p2addr'high-2)<"110" else '0';


  portCpu: cpuInt
  generic map( 3, cpuDataIn'length, cpuPortIn'length, "000", "101" )
  port map( address(address'high downto address'high-2), cpuDataIn,
            cpuDataOut, cpuPortIn, cpuPortOut, cpuRd, cpuWr, rw, oe,
            cs, rst, cpuClk, clk );

  cpuCeDecoded <= cpuRd or cpuWr;
  cpuRw <= cpuRd;
  cpuDataAin <= cpuPortOut(cpuDataAin'range);
  cpuDataBin <= cpuPortOut(cpuPortOut'high downto cpuDataAin'length);
  cpuPortIn <= cpuDataBout & cpuDataAout;


  ssram0Control: cpuInt
  generic map( address'length, cpuDataIn'length, ssram0ctrl'length,
               SSRAM0_CTRL_ADDRESS, SSRAM0_CTRL_ADDRESS )
  port map( address, cpuDataIn, cpuDataOut, ssram0ctrl, ssram0ctrl,
            ssram0rd, ssram0wr, rw, oe, cs, rst, cpuClk, clk );
    ssram0state <= ssram0ctrl(ssram0state'range);
    cSel0 <= ssram0ctrl(ssram0ctrl'high downto ssram0state'length);

  ssram1Control: cpuInt
  generic map( address'length, cpuDataIn'length, ssram1ctrl'length,
               SSRAM1_CTRL_ADDRESS, SSRAM1_CTRL_ADDRESS )
  port map( address, cpuDataIn, cpuDataOut, ssram1ctrl, ssram1ctrl,
            ssram1rd, ssram1wr, rw, oe, cs, rst, cpuClk, clk );
    ssram1state <= ssram1ctrl(ssram1state'range);
    cSel1 <= ssram1ctrl(ssram1ctrl'high downto ssram1state'length);

  ssram2Control: cpuInt
  generic map( address'length, cpuDataIn'length, ssram2ctrl'length,
               SSRAM2_CTRL_ADDRESS, SSRAM2_CTRL_ADDRESS )
  port map( address, cpuDataIn, cpuDataOut, ssram2ctrl, ssram2ctrl,
            ssram2rd, ssram2wr, rw, oe, cs, rst, cpuClk, clk );
    ssram2state <= ssram2ctrl(ssram2state'range);
    cSel2 <= ssram2ctrl(ssram2ctrl'high downto ssram2state'length);

  cpuControl: cpuInt
  generic map( address'length, cpuDataIn'length, cpuByte'length,
               CPU_CTRL_ADDRESS, CPU_CTRL_ADDRESS )
  port map( address, cpuDataIn, cpuDataOut, cpuByte, cpuByte,
            port0rd, port0wr, rw, oe, cs, rst, cpuClk, clk );

  port1Control: cpuInt
  generic map( address'length, cpuDataIn'length, port1byte'length,
               PORT1_CTRL_ADDRESS, PORT1_CTRL_ADDRESS )
  port map( address, cpuDataIn, cpuDataOut, port1byte, port1byte,
            port1rd, port1wr, rw, oe, cs, rst, cpuClk, clk );

  port2Control: cpuInt
  generic map( address'length, cpuDataIn'length, port2byte'length,
               PORT2_CTRL_ADDRESS, PORT2_CTRL_ADDRESS )
  port map( address, cpuDataIn, cpuDataOut, port2byte, port2byte,
            port2rd, port2wr, rw, oe, cs, rst, cpuClk, clk );

end struct;
```

```
-- Erinyes
--
-- Memory arbiter testbench
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.arbiter_pak.all;
use work.cpu_pak.all;
use work.simCpu_pak.all;
use work.cpu;
use work.utils_pak.all;

entity arbiter_tb is
end arbiter_tb;


architecture behav of arbiter_tb is

  component CPU is
    generic(
      CLOCK_PERIOD: time := cpuClockPeriod_c;
      WAIT_STATE: integer := 0
    );
    port(
      address: out address_typ;
      data: inout data_typ;
      rw: out std_logic;
      cs: out std_logic;
      oe: out std_logic;
      rst: in std_logic;
      clk: out std_logic
    );
  end component CPU;


  component arbiter is
    generic (
      SSRAM0_CTRL_ADDRESS: std_logic_vector:= "110"&x"01000";
      SSRAM1_CTRL_ADDRESS: std_logic_vector:= "110"&x"01004";
      SSRAM2_CTRL_ADDRESS: std_logic_vector:= "110"&x"01008";
      CPU_CTRL_ADDRESS: std_logic_vector:= "110"&x"01010";
      PORT1_CTRL_ADDRESS: std_logic_vector:= "110"&x"01014";
      PORT2_CTRL_ADDRESS: std_logic_vector:= "110"&x"01018";
      CPU_A_WIDTH: integer:= 9;
      CPU_B_WIDTH: integer:= 9;
      PORT1_A_WIDTH: integer:= 9;
      PORT1_B_WIDTH: integer:= 9;
      PORT2_A_WIDTH: integer:= 9;
      PORT2_B_WIDTH: integer:= 9
    );
    port (
      address: in std_logic_vector(22 downto 0);
      cpuDataIn: in std_logic_vector(31 downto 0);
      cpuDataOut: out std_logic_vector(31 downto 0);
      rw: in std_logic;
      oe: in std_logic;
      cs: in std_logic;
      cpuClk: in std_logic;
```

```
      p1addr: in std_logic_vector(22 downto 0);
      p1dataIn: in std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
      p1dataOut: out std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
      p1rdy: out std_logic;
      p1rw: in std_logic;
      p1ce: in std_logic;

      p2addr: in std_logic_vector(22 downto 0);
      p2dataIn: in std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
      p2dataOut: out std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
      p2rdy: out std_logic;
      p2rw: in std_logic;
      p2ce: in std_logic;

      rst: in std_logic;
      clk: in std_logic;

-- SSRAM0
      sa0: out std_logic_vector(18 downto 0) := (others=>'0');
      dqa0: inout std_logic_vector(8 downto 0) := (others=>'Z');
      dqb0: inout std_logic_vector(8 downto 0) := (others=>'Z');

      bwaN0: out std_logic;
      bwbN0: out std_logic;
      bweN0: out std_logic;
      gwN0: out std_logic;

      advN0: out std_logic;
      adscN0: out std_logic;
      adspN0: out std_logic;

      ceN0: out std_logic_vector(3 downto 0);
      oeN0: out std_logic_vector(3 downto 0);
      zz0: out std_logic;
      cko0: out std_logic;

-- SSRAM1
      sa1: out std_logic_vector(18 downto 0) := (others=>'0');
      dqa1: inout std_logic_vector(8 downto 0) := (others=>'Z');
      dqb1: inout std_logic_vector(8 downto 0) := (others=>'Z');

      bwaN1: out std_logic;
      bwbN1: out std_logic;
      bweN1: out std_logic;
      gwN1: out std_logic;

      advN1: out std_logic;
      adscN1: out std_logic;
      adspN1: out std_logic;

      ceN1: out std_logic_vector(3 downto 0);
      oeN1: out std_logic_vector(3 downto 0);
      zz1: out std_logic;
      cko1: out std_logic;

-- SSRAM2
      sa2: out std_logic_vector(18 downto 0) := (others=>'0');
      dqa2: inout std_logic_vector(8 downto 0) := (others=>'Z');
      dqb2: inout std_logic_vector(8 downto 0) := (others=>'Z');

      bwaN2: out std_logic;
      bwbN2: out std_logic;
      bweN2: out std_logic;
      gwN2: out std_logic;

      advN2: out std_logic;
      adscN2: out std_logic;
```

```
      adspN2: out std_logic;

      ceN2: out std_logic_vector(3 downto 0);
      oeN2: out std_logic_vector(3 downto 0);
      zz2: out std_logic;
      cko2: out std_logic
    );
  end component arbiter;


-- clock period
  constant fpgaClkPeriod_c: time:= 8 ns;

-- control addresses
  constant ssram0Ctrl_c: std_logic_vector(22 downto 0)
                       :="110"&x"01000";
  constant ssram1Ctrl_c: std_logic_vector(22 downto 0)
                       :="110"&x"01004";
  constant ssram2Ctrl_c: std_logic_vector(22 downto 0)
                       :="110"&x"01008";

  constant port0Ctrl_c: std_logic_vector(22 downto 0)
                       :="110"&x"01010";
  constant port1Ctrl_c: std_logic_vector(22 downto 0)
                       :="110"&x"01014";
  constant port2Ctrl_c: std_logic_vector(22 downto 0)
                       :="110"&x"01018";


-- interconnect
  signal p1dataOut: std_logic_vector(9+9-1 downto 0);
  signal p1rdy: std_logic;
  signal p2dataOut: std_logic_vector(9+9-1 downto 0);
  signal p2rdy: std_logic;
  signal clk: std_logic:= '0';



-- SSRAM1
  signal sa0: std_logic_vector(18 downto 0) := (others=>'0');
  signal dqa0: std_logic_vector(8 downto 0) := (others=>'Z');
  signal dqb0: std_logic_vector(8 downto 0) := (others=>'Z');
  signal bwaN0: std_logic;
  signal bwbN0: std_logic;
  signal bweN0: std_logic;
  signal gwN0: std_logic;
  signal advN0: std_logic;
  signal adscN0: std_logic;
  signal adspN0: std_logic;
  signal ceN0: std_logic_vector(3 downto 0);
  signal oeN0: std_logic_vector(3 downto 0);
  signal zz0: std_logic;
  signal cko0: std_logic;

-- SSRAM1
  signal sa1: std_logic_vector(18 downto 0) := (others=>'0');
  signal dqa1: std_logic_vector(8 downto 0) := (others=>'Z');
  signal dqb1: std_logic_vector(8 downto 0) := (others=>'Z');
  signal bwaN1: std_logic;
  signal bwbN1: std_logic;
  signal bweN1: std_logic;
  signal gwN1: std_logic;
  signal advN1: std_logic;
  signal adscN1: std_logic;
  signal adspN1: std_logic;
  signal ceN1: std_logic_vector(3 downto 0);
  signal oeN1: std_logic_vector(3 downto 0);
```

```
  signal zz1: std_logic;
  signal cko1: std_logic;

-- SSRAM2
  signal sa2: std_logic_vector(18 downto 0) := (others=>'0');
  signal dqa2: std_logic_vector(8 downto 0) := (others=>'Z');
  signal dqb2: std_logic_vector(8 downto 0) := (others=>'Z');
  signal bwaN2: std_logic;
  signal bwbN2: std_logic;
  signal bweN2: std_logic;
  signal gwN2: std_logic;
  signal advN2: std_logic;
  signal adscN2: std_logic;
  signal adspN2: std_logic;
  signal ceN2: std_logic_vector(3 downto 0);
  signal oeN2: std_logic_vector(3 downto 0);
  signal zz2: std_logic;
  signal cko2: std_logic;



-- arbiter procedures records
  type arbsig_typ is record
    addr: std_logic_vector(22 downto 0);
    dataIn: std_logic_vector(9+9-1 downto 0);
    rw: std_logic;
    ce: std_logic;
  end record arbsig_typ;
  signal arb1sig_rec: arbSig_typ;
  signal arb2sig_rec: arbSig_typ;


-- cpu interface
  signal address: address_typ;
  signal cpuDataIn, cpuDataOut: data_typ;
  signal rw, oe, cs, rst, cpuRst, cpuClk: std_logic;



-- readability procedures
  procedure write(constant a: in std_logic_vector(arb1sig_rec.addr'range);
                  constant d: in std_logic_vector(19 downto 0);
                  signal sig_rec: out arbSig_typ ) is
  begin
    sig_rec.addr <= a;
    sig_rec.dataIn <= d(arb1sig_rec.dataIn'range);
    sig_rec.rw <= '0';
    sig_rec.ce <= '1';
    wait for fpgaClkPeriod_c;
    sig_rec.ce <= '0';
    wait for fpgaClkPeriod_c;
  end procedure write;


  procedure read(constant a: in std_logic_vector(arb1sig_rec.addr'range);
                 signal sig_rec: out arbSig_typ ) is
  begin
    sig_rec.addr <= a;
    sig_rec.rw <= '1';
    sig_rec.ce <= '1';
    wait for fpgaClkPeriod_c;
    sig_rec.ce <= '0';
    wait for fpgaClkPeriod_c;
  end procedure read;
```

```
begin

  cpu0: cpu
  generic map( WAIT_STATE=>2 )
  port map( address, cpuDataIn, rw, cs, oe, cpuRst, cpuClk );


  clk <= not clk after fpgaClkPeriod_c/2;

  arbiter0: arbiter
  port map( address, cpuDataIn, cpuDataOut, rw, oe, cs, cpuClk,
            arb1sig_rec.addr,  arb1sig_rec.dataIn,  p1dataOut,  p1rdy,  arb1sig_rec.rw,
arb1sig_rec.ce,
            arb2sig_rec.addr,  arb2sig_rec.dataIn,  p2dataOut,  p2rdy,  arb2sig_rec.rw,
arb1sig_rec.ce,
            rst, clk,
            sa0, dqa0, dqb0, bwaN0, bwbN0, bweN0, gwN0, advN0, adscN0, adspN0,
            ceN0, oeN0, zz0, cko0,
            sa1, dqa1, dqb1, bwaN1, bwbN1, bweN1, gwN1, advN1, adscN1, adspN1,
            ceN1, oeN1, zz1, cko1,
            sa2, dqa2, dqb2, bwaN2, bwbN2, bweN2, gwN2, advN2, adscN2, adspN2,
            ceN2, oeN2, zz2, cko2 );


-- simulate ssram
  dqa0 <= '0'&x"0a" when bweN0='1' else (others=>'Z');
  dqb0 <= '0'&x"0b" when bweN0='1' else (others=>'Z');
  dqa1 <= '0'&x"1a" when bweN1='1' else (others=>'Z');
  dqb1 <= '0'&x"1b" when bweN1='1' else (others=>'Z');
  dqa2 <= '0'&x"2a" when bweN2='1' else (others=>'Z');
  dqb2 <= '0'&x"2b" when bweN2='1' else (others=>'Z');


  cpuRst <= not rst;

  test: process
  begin
    rst <= '1';
    wait for cpuClockPeriod_c;
    rst <= '0';
    wait for cpuClockPeriod_c;

    report "Writing config.";
    cpuWrite(ssram0ctrl_c, x"000000f1", cpuSig_rec);
    cpuWrite(ssram1ctrl_c, x"000000a2", cpuSig_rec);
    cpuWrite(ssram2ctrl_c, x"00000053", cpuSig_rec);
    cpuWrite(port0ctrl_c, x"00000003", cpuSig_rec);
    cpuWrite(port1ctrl_c, x"00000003", cpuSig_rec);
    cpuWrite(port2ctrl_c, x"00000003", cpuSig_rec);

    report "Reading config.";
    cpuRead(ssram0ctrl_c, cpuSig_rec);
    cpuRead(ssram1ctrl_c, cpuSig_rec);
    cpuRead(ssram2ctrl_c, cpuSig_rec);
    cpuRead(port0ctrl_c, cpuSig_rec);
    cpuRead(port1ctrl_c, cpuSig_rec);
    cpuRead(port2ctrl_c, cpuSig_rec);

    report "Start of test, reading.";
    cpuRead("000"&x"00002", cpuSig_rec);
    read( "000"&x"00004", arb1sig_rec);
    read( "000"&x"00008", arb2sig_rec);

    report "Writing...";
--    write( "000"&x"00020", x"12345", arb0sig_rec);
    cpuWrite("000"&x"00020", x"fad87654", cpuSig_rec);
    write( "000"&x"00040", x"12345", arb1sig_rec);
```

```
    write( "000"&x"00080", x"12345", arb2sig_rec);

    report "Reading...";
    cpuRead("000"&x"00200", cpuSig_rec);
    read( "000"&x"00400", arb1sig_rec);
    read( "000"&x"00800", arb2sig_rec);

    wait;

  end process test;

end behav;
```

```
-- Erinyes
--
-- Temperature sensors interface
--
-- Purpose:
--    Used to read/write data from/to the sensors
--
-- I/O:
--  Internal signals:
--    alrt: alert has been reported, interface
--          on hold until alert cleared
--    device: device address
--    dataIn: data channel (in)
--    dataOut: data channel (out)
--    dataRdy: indicates that data is ready
--    busy: indicates that the interface is busy
--    rd: get data
--    rw: read/write signal
--    ce: clock enable
--    rst: reset
--    clk100kHz: smbus clock
--    clk: internal clock
--  External signals:
---   alertN: overheat alert signal
--    sData: data signal
--    sClk: chip synchronization clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity tempSensor is
  generic( FIFO_DEPTH: integer:= 16 );
  port(
    alrt: out std_logic;
    device: in std_logic_vector(6 downto 0);
    dataIn: in std_logic_vector(15 downto 0);
    dataOut: out std_logic_vector(15 downto 0);
    dataRdy: out std_logic;
    busy: out std_logic;
    rd: in std_logic;
    wr: in std_logic;
    ce: in std_logic;
    rst: in std_logic;
    clk100kHz: in std_logic;
    clk: in std_logic;
    alertN: in std_logic;
    sData: inout std_logic;
    sClk: out std_logic
  );
end tempSensor;
```

```
-- Erinyes
--
-- Temperature sensors interface
--
-- Purpose:
--    Used to read/write data from/to the sensors
--
-- I/O:
--  Internal signals:
--    alrt: alert has been reported, interface
--          on hold until alert cleared
--    device: device address
--    dataIn: data channel (int)
--    dataOut: data channel (int)
--    dataRdy: indicates that data is ready
--    rd: get data
--    rw: read/write signal
--    ce: clock enable
--    rst: reset
--    clk100kHz: smbus clock
--    clk: internal clock
--  External signals:
--    alertN: overheat alert signal
--    sData: data signal
--    sClk: chip synchronization clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture rtl of tempSensor is

  component smbInt is
    port(
      device: in std_logic_vector(6 downto 0);
      dataIn: in std_logic_vector(15 downto 0);
      dataOut: out std_logic_vector(15 downto 0);
      rdy: out std_logic;
      rw: in std_logic;
      ce: in std_logic;
      rst: in std_logic;
      clk100kHz: in std_logic;
      clk: in std_logic;
      sData: inout std_logic;
      sClk: out std_logic
    );
  end component;

  component fifo is
    generic (
      WIDTH: integer:= 8;
      DEPTH: integer:= 16
    );
    port (
      wData: in std_logic_vector(WIDTH-1 downto 0);
      rData: out std_logic_vector(WIDTH-1 downto 0);
      mty: out std_logic;
      we: in std_logic;
      rd: in std_logic;
      rst: in std_logic;

      clk: in std_logic
    );
  end component;


-- alert response address
```

```
  constant ara_c: std_logic_vector(6 downto 0)
               := "0001100";

-- interconnect
  signal outFifoIn, outFifoOut: std_logic_vector(23 downto 0);
  signal outmty, outWe, outRd: std_logic:= '0';
  signal inFifoIn: std_logic_vector(15 downto 0);
  signal inmty, inWe, inRd: std_logic;
  signal smbDev: std_logic_vector(6 downto 0);
  signal smbDataIn, smbDataOut: std_logic_vector(15 downto 0);
  signal smbRdy, smbRw, smbCe: std_logic;
  signal curDevLatch: std_logic;
  signal currentDevice: std_logic_vector(6 downto 0);
  signal alerted, alertCe: std_logic;
  signal pending: std_logic;

  signal sClk_w: std_logic;


begin

-- 24 bits in output fifo, 7 bits address, 1 bit command, 8 bits register,
--     8 bits data
  outFifo: fifo
  generic map( 24, FIFO_DEPTH )
  port map( outFifoIn, outFifoOut, outMty, outWe, outRd, rst, clk );

-- 16 bits in input fifo, 1 bit status, 7 bits address, 8 bits data
-- write commands will be in fifo as well (status purpose)
  inFifo: fifo
  generic map( 16, FIFO_DEPTH*2 )
  port map( inFifoIn, dataOut, inMty, inWe, rd, rst, clk );



  smbInt0: smbInt
  port map( smbDev, smbDataIn, smbDataOut, smbRdy, smbRw, smbCe, rst, clk100kHz,
            clk, sData, sClk_w );
  sClk <= sClk_w;



-- data is ready when incomming fifo is not empty
  dataRdy <= not inMty;



-- outgoing fifo input/output is divided in 3
  outFifoMux: process(alerted, device, dataIn, ce, alertCe, wr)
  begin
    muxIf: if( alerted='1' ) then
-- send read command with alert address
      outFifoIn <= ara_c & '1' & dataIn;
      outWe <= alertCe;
    else -- muxIf
      outFifoIn <= device & not wr & dataIn;
      outWe <= ce;
    end if muxIf;
  end process outFifoMux;

  smbDev <= outFifoOut(23 downto 17);
  smbDataIn <= outFifoOut(smbDataIn'range);
  smbRw <= outFifoOut(16);



-- pack incoming fifo data
```

```
  inFifoIn <= currentDevice & smbDataOut(smbDataOut'high) &
              smbDataOut(7 downto 0);


  alertSig: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      alerted <= not alertN;
      alertCe <= alerted nor alertN; -- rising_edge(alerted)
    end if clkIf;
  end process alertSig;
  alrt <= alerted;


  outFifoRead: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      readIf: if( smbRdy='1' and smbCe='0' and outMty='0' ) then
        outRd <= '1';
      else  -- readIf
        outRd <= '0';
      end if readIf;
    end if clkIf;
  end process outFifoRead;


  curDev: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      curDevLatch <= outRd;
      outFifoReadIf: if( curDevLatch='1' ) then
        currentDevice <= outFifoOut(23 downto 17);
      end if outFifoReadIf;
    end if clkIf;
  end process curDev;



  smbStart: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      sendIf: if( smbRdy='1' and outMty='0' ) then
        smbCe <= '1';
      else -- sendIf
        smbCe <= '0';
      end if sendIf;
    end if clkIf;
  end process smbStart;



  fifoInLatch: process(clk)
  begin
    clkIf: if( rising_edge(clk) ) then
      latchIf: if( smbRdy='0' ) then
        pending <= '1';
      elsif( pending='1' and smbRdy='1' ) then -- latchIf
        inWe <= '1';
        pending <= '0';
      else -- latchIf
        inWe <= '0';
      end if latchIf;
    end if clkIf;
  end process fifoInLatch;


end rtl;
```

```
-- Erinyes
--
-- Temperature sensors interface
--
-- Purpose:
--    Used to read/write data from/to the sensors
--
-- I/O:
--  Internal signals:
--    alrt: alert has been reported, interface
--          on hold until alert cleared
--    address: cpu address bus
--    cpuDataIn: incoming cpuData bus
--    cpuDataOut : outgoing cpuData bus
--    rw: read/write signal
--    oe: output enable signal
--    cs: chip select signal
--    rst: reset
--    cpuClk: cpu clock
--    clk: internal clock
--  External signals:
---   alertN: overheat alert signal
--    sData: data signal
--    sClk: chip synchronization clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

library work;
use work.utils_pak.all;

entity tSens is
  generic(
    FIFO_DEPTH: integer:= 16;
    CLK_DIVISOR: integer:= 625;
    TRIG_ADDRESS: std_logic_vector:= x"0";
    TRIG_ADDRESS_STATUS: std_logic_vector:= x"0"
  );
  port(
    alrt: out std_logic;
    address: in std_logic_vector(22 downto 0);
    cpuDataIn: in std_logic_vector(31 downto 0);
    cpuDataOut: out std_logic_vector(31 downto 0);
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic;
    alertN: in std_logic;
    sData: inout std_logic;
    sClk: out std_logic
  );
end tSens;
```

```
-- Erinyes
--
-- Temperature sensors interface
--
-- Purpose:
--    Used to read/write data from/to the sensors
--
-- I/O:
--  Internal signals:
--    alrt: alert has been reported, interface
--          on hold until alert cleared
--    address: cpu address bus
--    cpuDataIn: incoming cpuData bus
--    cpuDataOut: outgoing cpuData bus
--    rw: read/write signal
--    oe: output enable signal
--    cs: chip select signal
--    rst: reset
--    cpuClk: cpu clock
--    clk: internal clock
--  External signals:
---   alertN: overheat alert signal
--    sData: data signal
--    sClk: chip synchronization clock
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003
architecture rtl of tSens is

  component bufG is
     port( o: out std_logic; i: in std_logic );
  end component bufG;

  component cpuInt is
    generic(
      constant ADDR_WIDTH: integer := 24;
      constant DATA_WIDTH: integer := 32;
      constant TRIG_ADDRESS_HIGH: std_logic_vector := x"0";
      constant TRIG_ADDRESS_LOW: std_logic_vector := x"0"
    );
    port(
      address: in std_logic_vector(ADDR_WIDTH-1 downto 0);
      cpuDataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
      cpuDataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
      dataIn: in std_logic_vector(DATA_WIDTH-1 downto 0);
      dataOut: out std_logic_vector(DATA_WIDTH-1 downto 0);
      rd: out std_logic;
      wr: out std_logic;
      rw: in std_logic;
      oe: in std_logic;
      cs: in std_logic;
      rst: in std_logic;
      cpuClk: in std_logic;
      clk: in std_logic
    );
  end component cpuInt;

  component tempSensor is
    generic( FIFO_DEPTH: integer:= 16 );
    port(
      alrt: out std_logic;
      device: in std_logic_vector(6 downto 0);
      dataIn: in std_logic_vector(15 downto 0);
      dataOut: out std_logic_vector(15 downto 0);
      dataRdy: out std_logic;
      rd: in std_logic;
```

```
      wr: in std_logic;
      ce: in std_logic;
      rst: in std_logic;
      clk100kHz: in std_logic;
      clk: in std_logic;
      alertN: in std_logic;
      sData: inout std_logic;
      sClk: out std_logic
    );
  end component tempSensor;

-- clock division counter
  signal clk100KHz, clk100KHz_w: std_logic:= '0';
  signal count: std_logic_vector(log2m1(CLK_DIVISOR) downto 0);

-- interconnect
  signal dataIn: std_logic_vector(23 downto 0);
  signal dataOut: std_logic_vector(23 downto 0);
  signal dataRdy: std_logic;
  signal rd, wr: std_logic;
  signal alrt_w: std_logic;
  signal tSensDataOut: std_logic_vector(15 downto 0);

-- internal status register
  signal statRd, statWr: std_logic;
  signal statRead: std_logic_vector(31 downto 0):= (others=>'0');
  signal statWrite: std_logic_vector(statRead'range):= (others=>'0');

begin
  cpuInt0: cpuInt
  generic map( address'length, dataIn'length,
               TRIG_ADDRESS, TRIG_ADDRESS )
  port map( address, cpuDataIn, cpuDataOut, dataOut, dataIn, rd, wr, rw,
            oe, cs, rst, cpuClk, clk );
    dataOut <= "00000000" & tSensDataOut;

  statReg: cpuInt
  generic map( address'length, statRead'length,
               TRIG_ADDRESS_STATUS, TRIG_ADDRESS_STATUS )
  port map( address, cpuDataIn, cpuDataOut, statRead, statWrite, statRd, statWr, rw,
            oe, cs, rst, cpuClk, clk );

  tempSens0: tempSensor
  generic map( 16 )
  port map( alrt_w, dataIn(22 downto 16), dataIn(15 downto 0),
            tSensDataOut, dataRdy, rd, dataIn(23), wr,
            rst, clk100KHz, clk, alertN, sData, sClk );
  alrt <= alrt_w;

  bufg0: bufG
  port map( clk100KHz, clk100KHz_w );

  cntr: process(cpuClk)
  begin
    clkIf: if( rising_edge(cpuClk) ) then
      rstIf: if( rst='1' or count=0 ) then
        count <= conv_std_logic_vector(CLK_DIVISOR-1, count'length);
        clk100KHz_w <= not clk100KHz_w;
      else -- rstIf
        count <= count - 1;
      end if rstIf;
    end if clkIf;
  end process cntr;

  statRead <= (0=>dataRdy, 1=>alrt_w, others=>'0');

end rtl;
```

```
-- Erinyes
--
-- Temperature sensors interface testbench
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library unisim;
use unisim.all;

library work;
use work.utils_pak.all;
use work.cpu;
use work.simCpu_pak.all;

entity tSens_tb is
end tSens_tb;


architecture behav of tSens_tb is

  component CPU is
    generic(
      CLOCK_PERIOD: time := cpuClockPeriod_c;
      WAIT_STATE: integer := 0
    );
    port(
      address: out address_typ;
      data: inout data_typ;
      rw: out std_logic;
      cs: out std_logic;
      oe: out std_logic;
      rst: in std_logic;
      clk: out std_logic
    );
  end component CPU;


  component tSens is
    generic(
      FIFO_DEPTH: integer:= 16;
      CLK_DIVISOR: integer:= 625;
      TRIG_ADDRESS: std_logic_vector:= x"0";
      TRIG_ADDRESS_STATUS: std_logic_vector:= x"0"
    );
    port(
      alrt: out std_logic;
      address: in std_logic_vector(22 downto 0);
      cpuDataIn: in std_logic_vector(31 downto 0);
      cpuDataOut: out std_logic_vector(31 downto 0);
      rw: in std_logic;
      oe: in std_logic;
      cs: in std_logic;
      rst: in std_logic;
      cpuClk: in std_logic;
      clk: in std_logic;
      alertN: in std_logic;
      sData: inout std_logic;
      sClk: out std_logic
    );
  end component tSens;
```

```
-- test constants
  constant fpgaClockPeriod_c: time := 8 ns;
  constant devAddr_c: std_logic_vector(22 downto 0)
                    := "010"&x"82000";
  constant devAddr1_c: std_logic_vector(22 downto 0)
                    := "010"&x"82004";


-- interconnect
  signal alrt: std_logic;
  signal address: std_logic_vector(22 downto 0);
  signal cpuDataIn, cpuDataOut: std_logic_vector(31 downto 0);
  signal rw, oe, cs, rst, cpuRst: std_logic;
  signal cpuClk: std_logic;
  signal clk: std_logic:= '0';
  signal alertN, sData, sClk: std_logic;

begin

  cpu0: cpu
  generic map( 8 ns, 1 )
  port map( address, cpuDataIn, rw, cs, oe, cpuRst, cpuClk );


  ts0: tSens
  generic map( 16, 250, devAddr_c, devAddr1_c )
  port map( alrt, address, cpuDataIn, cpuDataOut, rw, oe,
            cs, rst, cpuClk, clk, alertN, sData, sClk );

  clk <= not clk after fpgaClockPeriod_c/2;
  cpuRst <= not rst;


  test: process
  begin
    alertN <= '1'; -- no alert
    rst <= '1';
    sData <= 'L';
    wait for fpgaClockPeriod_c;
    rst <= '0';
    wait for fpgaClockPeriod_c;

    report "Sending data write command";
    cpuWrite(devAddr_c, x"0085abcd", cpuSig_rec);
    wait for cpuClockPeriod_c;

    pollStatLoop: while( cpuDataOut(0)/='1' ) loop
      cpuRead(devAddr1_c, cpuSig_rec);
    end loop pollStatLoop;

    wait for 10*fpgaClockPeriod_c;

    report "Read data from fifo";
    cpuRead(devAddr_c, cpuSig_rec);
    cpuRead(devAddr1_c, cpuSig_rec);
    wait for 1000*fpgaClockPeriod_c;

    report "Sending data write command with abort from sensors";
    cpuWrite(devAddr_c, x"0085abcd", cpuSig_rec);
    wait for cpuClockPeriod_c;

    sData <= 'H';
    pollStatLoop1: while( cpuDataOut(0)/='1' ) loop
      cpuRead(devAddr1_c, cpuSig_rec);
    end loop pollStatLoop1;
```

```
    sData <= 'L';

    wait for 10*fpgaClockPeriod_c;

    report "Read data from fifo";
    cpuRead(devAddr_c, cpuSig_rec);
    cpuRead(devAddr1_c, cpuSig_rec);
    wait for 1000*fpgaClockPeriod_c;

    report "Generating an alert";
    alertN <= '0';

    pollStatLoop2: while( cpuDataOut(0)/='1' ) loop
      cpuRead(devAddr1_c, cpuSig_rec);
    end loop pollStatLoop2;

    wait;

  end process test;

end behav;
```

```
-- Erinyes
--
-- Utility functions package
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

package utils_pak is

-- log2-1
-- Takes an integer as parameter, returns the log2(n)-1
  function log2m1(a: integer) return integer;

end package utils_pak;
```

```
-- Erinyes
--
-- Utility functions package
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

package body utils_pak is

-- log2-1
-- Takes an integer as parameter, returns the log2(n)-1
  function log2m1(a: integer) return integer is
    variable n_v: integer:= 0;
    variable a_v: integer:= a;
  begin
    divLoop: while( a_v>1 ) loop
      n_v := n_v+1;
      a_v := a_v/2;
    end loop divLoop;
    return n_v;
  end function log2m1;

end utils_pak;
```

```
-- Erinyes
--
-- Misc: Inter clock domain signal transfer
--
-- Purpose:
--    Passes a signal through two clock domains
--
-- I/O:
--    d: input signal to be transfered (clk1)
--    q: output signal (clk2)
--    clk1: clock signal
--    clk2: clock signal
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

entity sigTrans is
  port(
    d: in std_logic;
    q: out std_logic;
    clk1: in std_logic;
    clk2: in std_logic
  );
end sigTrans;
```

```
-- Erinyes
--
-- Misc: Inter clock domain signal transfer
--
-- Purpose:
--    Passes a signal through two clock domains
--
-- I/O:
--    d: input signal to be transfered (clk1)
--    q: output signal (clk2)
--    clk1: clock signal
--    clk2: clock signal
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture rtl of sigTrans is

-- internal signals declarations
  signal clk1Sample: std_logic := '0';
  signal clk1Hold: std_logic := '0';
  signal clk2Sample: std_logic := '0';

begin

  clk1Sampler: process(clk1)
  begin
    clkIf: if( rising_edge(clk1) ) then
      clk1Sample <= d;
    end if clkIf;
  end process clk1Sampler;

  clk1Holder: process(clk1)
  begin
    clkIf: if( rising_edge(clk1) ) then
      rstIf: if( clk2Sample='1' ) then
        clk1Hold <= '0';
      else -- rstIf
        clk1Hold <= clk1Sample or clk1Hold;
      end if rstIf;
    end if clkIf;
  end process clk1Holder;

  clk2Sampler: process(clk2)
  begin
    clkIf: if( rising_edge(clk2) ) then
      clk2Sample <= clk1Hold;
    end if clkIf;
  end process clk2Sampler;

  q <= clk2Sample;

end rtl;
```

```
-- Erinyes
--
-- CPU simulation
--
-- Purpose:
--    CPU access routines for simulation.
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

package simCpu_pak is

-- CPU definitions
constant cpuClockPeriod_c: time := 15 ns;
constant addressWidth_c: integer := 23;
constant dataWidth_c: integer:= 32;
constant cpuRead_c: std_logic := '1';
constant cpuWrite_c: std_logic := '0';

-- cpu types
  subtype address_typ is std_logic_vector(addressWidth_c-1 downto 0);
  subtype data_typ is std_logic_vector(dataWidth_c-1 downto 0);

-- CPU access structure
  type cpuSignals_typ is record
    address : address_typ;
    data : data_typ;
    rw : std_logic;
  end record;
  signal cpuSig_rec : cpuSignals_typ;
  signal cpuAccessDone : std_logic := '0';


-- procedures

-- cpu write access subroutine
-- synopsys: cpuWrite(address, data);
--    This routine sends a write command to the cpu
  procedure cpuWrite(constant address_c: in address_typ;
                     constant data_c: in data_typ;
                     signal cpuSig_rec: out cpuSignals_typ);
  procedure cpuRead(constant address_c: in address_typ;
                    signal cpuSig_rec: out cpuSignals_typ);

end simCpu_pak;
```

```
-- Erinyes
--
-- CPU simulation
--
-- Purpose:
--    CPU access routines for simulation.
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

package body simCpu_pak is

-- cpu write access subroutine
-- synopsys: cpuWrite(address, data);
--    This routine simulates a write command from the cpu
  procedure cpuWrite(constant address_c: in address_typ;
                     constant data_c: in data_typ;
                     signal cpuSig_rec: out cpuSignals_typ) is
    variable cpuSigTemp_rec: cpuSignals_typ;
  begin
    cpuSigTemp_rec.address := address_c;
    cpuSigTemp_rec.data := data_c;
    cpuSigTemp_rec.rw := cpuWrite_c;
-- start transaction
    cpuSig_rec <= cpuSigTemp_rec;
    wait on cpuAccessDone'transaction;
  end cpuWrite;


-- cpu read access subroutine
-- synopsys: cpuRead(address);
--    This routine simulated  a read command from the cpu
  procedure cpuRead(constant address_c: in address_typ;
                    signal cpuSig_rec: out cpuSignals_typ) is
    variable cpuSigTemp_rec: cpuSignals_typ;
  begin
    cpuSigTemp_rec.address := address_c;
    cpuSigTemp_rec.rw := cpuRead_c;
-- start transaction
    cpuSig_rec <= cpuSigTemp_rec;
    wait on cpuAccessDone'transaction;
  end cpuRead;

end simCpu_pak;
```

```
-- Erinyes
--
-- Tools: CPU simulation entity
--
-- Purpose:
--    Eases CPU accesses
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

library work;
use work.simCpu_pak;
use work.simCpu_pak.all;


entity CPU is
  generic(
    CLOCK_PERIOD: time := cpuClockPeriod_c;
    WAIT_STATE: integer := 0
  );
  port(
    address: out address_typ;
    data: inout data_typ;
    rw: out std_logic;
    cs: out std_logic;
    oe: out std_logic;
    rst: in std_logic;
    clk: out std_logic
  );
end CPU;
```

```
-- Erinyes
--
-- Tools: CPU simulation entity
--
-- Purpose:
--    Eases CPU accesses
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture behav of CPU is

  signal clk_w: std_logic:= '0';
  signal address_w: address_typ := (others=>'Z');
  signal data_w: data_typ := (others=>'Z');
  signal cs_w: std_logic := '1';
  signal oe_w: std_logic := '1';
  signal rw_w: std_logic := '1';


  procedure waitClk(constant ws: in integer) is
  begin
    waitState: for i in 1 to ws loop
      wait on clk_w'transaction;
      if( clk_w='0' ) then
        wait on clk_w'transaction;
      end if;
    end loop waitState;
  end waitClk;

  procedure waitClk is
  begin
    waitClk(1);
  end waitClk;


begin

  clk_w <= not clk_w after cpuClockPeriod_c/2;
  clk <= clk_w;

-- cpu access process
-- needed by cpu access subroutines
--   This process takes CPU requests from the testbench and sets the CPU
--  lines to their correct values
-- CPU accesses are done through an instance of cpuSignals_typ
  cpuAccess: process
  begin
    wait on cpuSig_rec'transaction;
    waitClk;
    wait for 8 ns;
    cs_w <= '0';
    wait for 5 ns;
    address_w <= cpuSig_rec.address;
    waitClk;
    wait for 7 ns;
    case cpuSig_rec.rw is
      when cpuRead_c =>
        rw_w <= cpuSig_rec.rw;
        oe_w <= '0';
-- fill this portion in
      when cpuWrite_c =>
        rw_w <= cpuSig_rec.rw;
        wait for 6 ns;
        data_w <= cpuSig_rec.data;
      when others =>
```

```
-- Do nothing
    end case;
    waitClk(2+WAIT_STATE);
    wait for 1 ns;
    rw_w <= '1';
    cs_w <= '1';
    oe_w <= '1';
    data_w <= (others=>'Z');
--    address_w <= (others=>'Z');
--  unlock process
    simCpu_pak.cpuAccessDone <= not simCpu_pak.cpuAccessDone;
  end process cpuAccess;


-- reset management
  cs <= cs_w when rst='1' else '1';
  oe <= oe_w when rst='1' else '1';
  rw <= rw_w when rst='1' else '1';
  address <= address_w; -- when rst='1' else (others=>'Z');
  data <= data_w when rst='1' else (others=>'Z');

end behav;
```

```
-- Erinyes
--
-- 3-State bus buffer
--
-- Purpose:
--   Generates an output enabled bus
--
-- I/O:
--  o: output
--  e: enable
--  i: input
--
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

-- synthesis translate_off
library unisim;
use unisim.bufe;
-- synthesis translate_on

entity bufeBus is
  port(
    o: out std_logic_vector;
    e: in std_logic;
    i: in std_logic_vector
  );
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of bufeBus: entity is "o";
end bufeBus;
```

```
-- Erinyes
--
-- 3-State bus buffer
--
-- Purpose:
--   Generates an output enabled bus
--
-- I/O:
--  o: output
--  e: enable
--  i: input
--
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of bufeBus is

  component bufe is
    port( o: out std_logic; e: in std_logic; i: in std_logic );
  end component bufe;
  attribute syn_black_box: boolean;
  attribute syn_black_box of bufe: component is true;


begin

  assert o'length=i'length
  report "Error, input/output width mismatch!"
  severity error;


  bufeBusGen: for j in i'range generate
    bufeIns_j: bufe
    port map( o(j), e, i(j) );
  end generate bufeBusGen;

end struct;
```

```
-- Erinyes
--
-- 3.3 volt input buffer
--
-- I/O:
--  o: output
--  i: input
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

entity ibufLvc33 is
  port(
    o: out std_logic_vector;
    i: in std_logic_vector
  );
end ibufLvc33;
```

```
-- Erinyes
--
-- 3.3 volt input buffer
--
-- I/O:
--  o: output
--  i: input
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of ibufLvc33 is

  component ibuf_lvcmos33 is
    port ( o: out std_logic; i: in std_logic );
  end component ibuf_lvcmos33;
  attribute syn_black_box: boolean;
  attribute syn_black_box of ibuf_lvcmos33: component is true;


begin

  assert o'length=i'length
  report "Error, input/output width mismatch!"
  severity error;


  busGen: for j in i'range generate
    buffer_i: ibuf_lvcmos33
    port map( o(j), i(j) );
  end generate busGen;

end struct;
```

```
-- Erinyes
--
-- 1.8 volt input/output buffer
--
-- I/O:
--  o: output
--  io: input/output
--  i: input
--  t: tristate enable
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

entity iobufLvc18 is
  port(
    o: out std_logic_vector;
    io: inout std_logic_vector;
    i: in std_logic_vector;
    t: in std_logic
  );
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of iobufLvc18: entity is "o";
end iobufLvc18;
```

```
-- Erinyes
--
-- 1.8 volt input/output buffer
--
-- I/O:
--  o: output
--  io: input/output
--  i: input
--  t: tristate enable
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of iobufLvc18 is

  component iobuf_lvcmos18_f_12 is
    port ( o: out std_logic; io: inout std_logic;
           i: in std_logic; t: in std_logic );
  end component iobuf_lvcmos18_f_12;
  attribute syn_black_box: boolean;
  attribute syn_black_box of iobuf_lvcmos18_f_12: component is true;


begin

  assert o'length=i'length
  report "Error, input/output width mismatch!"
  severity error;


  busGen: for j in i'range generate
    buffer_i: iobuf_lvcmos18_f_12
    port map( o(j), io(j), i(j), t );
  end generate busGen;

end struct;
```

```
-- Erinyes
--
-- 3.3 volt input/output buffer
--
-- I/O:
--  o: output
--   io: input/output
--  i: input
--  t: tristate enable
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

entity iobufLvc33 is
  port(
    o: out std_logic_vector;
    io: inout std_logic_vector;
    i: in std_logic_vector;
    t: in std_logic
  );
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of iobufLvc33: entity is "o";
end iobufLvc33;
```

```
-- Erinyes
--
-- 3.3 volt input/output buffer
--
-- I/O:
--  o: output
--  io: input/output
--  i: input
--  t: tristate enable
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of iobufLvc33 is

  component iobuf_lvcmos33_f_12 is
    port ( o: out std_logic; io: inout std_logic;
           i: in std_logic; t: in std_logic );
  end component iobuf_lvcmos33_f_12;
  attribute syn_black_box: boolean;
  attribute syn_black_box of iobuf_lvcmos33_f_12: component is true;


begin

  assert o'length=i'length
  report "Error, input/output width mismatch!"
  severity error;


  busGen: for j in i'range generate
    buffer_i: iobuf_lvcmos33_f_12
    port map( o(j), io(j), i(j), t );
  end generate busGen;

end struct;
```

```
-- Erinyes
--
-- 3.3 volt output buffer
--
-- I/O:
--  o: output
--  i: input
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

entity obufLvc33 is
  port(
    o: out std_logic_vector;
    i: in std_logic_vector
  );
end obufLvc33;
```

```
-- Erinyes
--
-- 3.3 volt output buffer
--
-- I/O:
--  o: output
--  i: input
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of obufLvc33 is

  component obuf_lvcmos33_f_12 is
    port ( o: out std_logic; i: in std_logic );
  end component obuf_lvcmos33_f_12;
  attribute syn_black_box: boolean;
  attribute syn_black_box of obuf_lvcmos33_f_12: component is true;


begin

  assert o'length=i'length
  report "Error, input/output width mismatch!"
  severity error;


  busGen: for j in i'range generate
    buffer_i: obuf_lvcmos33_f_12
    port map( o(j), i(j) );
  end generate busGen;

end struct;
```

```
-- Erinyes
--
-- Erinyes, mezzanine 1.8v
--
-- Purpose:
--   Instanciate mezzanine buffers
--
-- I/O:
--  Internal:
--  v18i: mezzanine 1.8v input
--  v18o: mezzanine 1.8v outpu
--  v18t: mezzanine 1.8v tristate
--  v18gClki: mezzanine 1.8v global clock input
--  v18gClko: mezzanine 1.8v global clock output
--  v18gClkt: mezzanine 1.8v global clock tristate
--  v18i1: mezzanine 1.8v input
--  v18o1: mezzanine 1.8v outpu
--  v18t1: mezzanine 1.8v tristate
--  External:
--  v18io: mezzanine 1.8v IOs
--  v18gClkIo: mezzanine 1.8v global clock
--  v18io1: mezzanine 1.8v IOs
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;

library work;
use work.arbiter_pak.all;
use work.cpu_pak.all;
use work.utils_pak.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

entity mezzanine18 is
  port (
    v18io: inout std_logic_vector(32 downto 0);
    v18gClkIo: inout std_logic_vector(1 downto 0);
    v18io1: inout std_logic_vector(81 downto 0);

    v18i: out std_logic_vector(32 downto 0);
    v18o: in std_logic_vector(32 downto 0);
    v18t: in std_logic_vector(32 downto 0);
    v18gClki: out std_logic_vector(1 downto 0);
    v18gClko: in std_logic_vector(1 downto 0);
    v18gClkt: in std_logic_vector(1 downto 0);
    v18i1: out std_logic_vector(81 downto 0);
    v18o1: in std_logic_vector(81 downto 0);
    v18t1: in std_logic_vector(81 downto 0)
  );
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of mezzanine18: entity is
            "v18io, v18gClkIo, v18io1";
end mezzanine18;
```

```
-- Erinyes
--
-- Erinyes, mezzanine 1.8v
--
-- Purpose:
--   Instanciate mezzanine buffers
--
-- I/O:
--  Internal:
--  v18i: mezzanine 1.8v input
--  v18o: mezzanine 1.8v outpu
--  v18t: mezzanine 1.8v tristate
--  v18gClkI: mezzanine 1.8v global clock input
--  v18gClkO: mezzanine 1.8v global clock output
--  v18gClkT: mezzanine 1.8v global clock tristate
--  v18i1: mezzanine 1.8v input
--  v18o1: mezzanine 1.8v outpu
--  v18t1: mezzanine 1.8v tristate
--  External:
--  v18Io: mezzanine 1.8v IOs
--  v18gClkIo: mezzanine 1.8v global clock
--  v18Io1: mezzanine 1.8v IOs
--
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of mezzanine18 is

  component iobuf_lvcmos18_f_12 is
    port( o: out std_logic; io: inout std_logic;
          i: in std_logic; t: in std_logic );
  end component iobuf_lvcmos18_f_12;
  attribute syn_black_box: boolean;
  attribute syn_black_box of iobuf_lvcmos18_f_12: component is true;


begin

  v18gen: for i in v18io'range generate
    v18buf_i: iobuf_lvcmos18_f_12
    port map( v18i(i), v18io(i), v18o(i), v18t(i) );
  end generate v18gen;

  v18gClkGen: for i in v18gClkIo'range generate
    v18buf_i: iobuf_lvcmos18_f_12
    port map( v18gClkI(i), v18gClkIo(i), v18gClkO(i), v18gClkT(i) );
  end generate v18gClkGen;

  v181gen: for i in v18io1'range generate
    v181buf_i: iobuf_lvcmos18_f_12
    port map( v18i1(i), v18io1(i), v18o1(i), v18t1(i) );
  end generate v181gen;

end struct;
```

```
-- Erinyes
--
-- Erinyes, mezzanine 3v3
--
-- Purpose:
--   Instanciate mezzanine buffers
--
-- I/O:
--  Internal:
--    b0v33i: 3.3 volt Input
--    b0v33o: 3.3 volt Output
--    b0v33t: 3.3 volt tristate
--    b0v33gClkI: 3.3 volt global clock pins input
--    b0v33gClkO: 3.3 volt global clock pins output
--    b0v33gClkT: 3.3 volt global clock pins tristate
--    b4v33gClkI: 3.3 volt global clock pins input
--    b4v33gClkO: 3.3 volt global clock pins output
--    b4v33gClkT: 3.3 volt global clock pins tristate
--    b6v33i: 3.3 volt Input
--    b6v33o: 3.3 volt Output
--    b6v33t: 3.3 volt tristate
--  External:
--    b0v33io: 3.3 volt IOs
--    b0v33gClkIo: 3.3 volt global clock pins
--    b4v33gClkIo: 3.3 volt global clock pins
--    b6v33io: 3.3 volt IOs
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003
library ieee;
use ieee.std_logic_1164.all;

library work;
use work.arbiter_pak.all;
use work.cpu_pak.all;
use work.utils_pak.all;
-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

entity mezzanine33 is
  port (
    b0v33io: inout std_logic_vector(65 downto 0);
    b0v33gClkIo: inout std_logic_vector(7 downto 4);
    b4v33gClkIo: inout std_logic_vector(3 downto 0);
    b6v33io: inout std_logic_vector(21 downto 0);

    b0v33i: out std_logic_vector(65 downto 0);
    b0v33o: in std_logic_vector(65 downto 0);
    b0v33t: in std_logic_vector(65 downto 0);
    b0v33gClkI: out std_logic_vector(7 downto 4);
    b0v33gClkO: in std_logic_vector(7 downto 4);
    b0v33gClkT: in std_logic_vector(7 downto 4);
    b4v33gClkI: out std_logic_vector(3 downto 0);
    b4v33gClkO: in std_logic_vector(3 downto 0);
    b4v33gClkT: in std_logic_vector(3 downto 0);
    b6v33i: out std_logic_vector(21 downto 0);
    b6v33o: in std_logic_vector(21 downto 0);
    b6v33t: in std_logic_vector(21 downto 0)
  );
  attribute black_box_tri_pins: string;
  attribute black_box_tri_pins of mezzanine33: entity is
            "b0v33io, b0v33gClkIo, b4v33gClkIo, b6v33io";
end mezzanine33;
```

```
-- Erinyes
--
-- Erinyes, mezzanine 3v3
--
-- Purpose:
--   Instanciate mezzanine buffers
--
-- I/O:
--  Internal:
--    b0v33i: 3.3 volt Input
--    b0v33o: 3.3 volt Output
--    b0v33t: 3.3 volt tristate
--    b0v33gClkI: 3.3 volt global clock pins input
--    b0v33gClkO: 3.3 volt global clock pins output
--    b0v33gClkT: 3.3 volt global clock pins tristate
--    b4v33gClkI: 3.3 volt global clock pins input
--    b4v33gClkO: 3.3 volt global clock pins output
--    b4v33gClkT: 3.3 volt global clock pins tristate
--    b6v33i: 3.3 volt Input
--    b6v33o: 3.3 volt Output
--    b6v33t: 3.3 volt tristate
--  External:
--    b0v33io: 3.3 volt IOs
--    b0v33gClkIo: 3.3 volt global clock pins
--    b4v33gClkIo: 3.3 volt global clock pins
--    b6v33io: 3.3 volt IOs
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of mezzanine33 is

  component iobuf_lvcmos33_f_12 is
    port( o: out std_logic; io: inout std_logic;
          i: in std_logic; t: in std_logic );
  end component iobuf_lvcmos33_f_12;
  attribute syn_black_box: boolean;
  attribute syn_black_box of iobuf_lvcmos33_f_12: component is true;

begin
  b0v33gen: for i in b0v33io'range generate
    b0v33buf_i: iobuf_lvcmos33_f_12
    port map( b0v33i(i), b0v33io(i), b0v33o(i), b0v33t(i) );
  end generate b0v33gen;

  b0v33gClkGen: for i in b0v33gClkIo'range generate
    b0v33buf_i: iobuf_lvcmos33_f_12
    port map( b0v33gClkI(i), b0v33gClkIo(i), b0v33gClkO(i), b0v33gClkT(i) );
  end generate b0v33gClkGen;

  b4v33gClkGen: for i in b4v33gClkIo'range generate
    b4v33buf_i: iobuf_lvcmos33_f_12
    port map( b4v33gClkI(i), b4v33gClkIo(i), b4v33gClkO(i), b4v33gClkT(i) );
  end generate b4v33gClkGen;

  b6v33gen: for i in b6v33io'range generate
    b0v33buf_i: iobuf_lvcmos33_f_12
    port map( b6v33i(i), b6v33io(i), b6v33o(i), b6v33t(i) );
  end generate b6v33gen;
end struct;
```

```
-- Erinyes
--
-- Erinyes, test
--
-- Purpose:
--   Tests the major functions of Erinyes
--
-- I/O:
--
--  cpuAddress: cpu address bus
--  cpuData: cpu data bus
--  cpuRw: cpu rw signal
--  cpuOe: cpu output enable
--  cpuCs: cpu chip select
--  cpuRst: fpga reset pin
--  cpuOutInt: cpu interrupt line
--  cpuInClk: cpu clock
--  clkP: differential clock input
--  clkN: differential clock input
--  cpuGsr: in std_logic;
--  cpuInitB: FPGA initialization pin
--  cpuCsB: FPGA programming chip select
--  cpuRdWrB: FPGA programming read/write
--
-- Temperature sensors
--    alertN: overheat alert signal
--    sData: data signal
--    sClk: chip synchronization clock
--
-- Digital potentiometers signals
--    spiDi: spi data in
--    spiDo: spi data out
--    spiRdy: interface ready signal
--    spiCs: chip select
--    spiPrN: spi preset
--    spiClk: spi clock
--
-- SSRAM signals
--  saX: ssram X address lines
--  dqaX: ssram X data line a
--  dqbX: ssram X data line b
--  bwaNX: ssram X byte write byte a
--  bwbNX: ssram X byte write byte b
--  bweNX: ssram X byte write enable
--  gwNX: ssram X global write
--  advNX: ssram X synchronous address advance
--  adscNX: ssram X synchronous address status controller
--  adspNX: ssram X synchronous address status processor
--  ceNX: ssram X clock enable
--  oeNX: ssram X output enable
--  zzX: ssram X snooze mode enable
--  cko: ssrams output clock signal
--
-- LED interface
---   pLed: LED power signals
--
-- MEZZANINES
-- 3v3 expansion
--  b0v33io: 3.3 volt IOs
--  b0v33gClkIo: 3.3 volt global clock pins
--  b4v33gClkIo: 3.3 volt global clock pins
--  b6v33io: 3.3 volt IOs
-- North
--  b1v18nIo: North mezzanine 1.8v IOs
--  b1v18nGclkIo: North mezzanine 1.8v global clock
--  b7v18nIo: North mezzanine 1.8v IOs
-- South
```

```
--  b1v18sIo: South mezzanine 1.8v IOs
--  b1v18sGclkIo: South mezzanine 1.8v global clock
--  b2v18sIo:  South mezzanine 1.8v IOs
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.arbiter_pak.all;
use work.cpu_pak.all;
use work.utils_pak.all;

-- synthesis translate_off
library unisim;
use unisim.all;
-- synthesis translate_on

entity erinyes is
  port (
-- CPU
    cpuAddress: in std_logic_vector(22 downto 0);
    cpuData: inout std_logic_vector(31 downto 0);
    cpuRw: in std_logic;
    cpuOe: in std_logic;
    cpuCs: in std_logic;
    cpuRst: in std_logic;
    cpuOutInt: out std_logic;
    cpuInClk: in std_logic;
    clkP: in std_logic;
    clkN: in std_logic;
    cpuGsr: in std_logic;
    cpuInitB: inout std_logic;
    cpuCsB: in std_logic;
    cpuRdWrB: in std_logic;

-- TEMPERATURE SENSORS
    alertN: in std_logic;
    sData: inout std_logic;
    sClk: out std_logic;

-- DIGITAL POTENTIOMETERS
    spiDi: in std_logic;
    spiDo: out std_logic;
    spiRdy: in std_logic;
    spiCs: out std_logic_vector(17 downto 0);
    spiPrN: out std_logic;
    spiClk: out std_logic;

-- SSRAM0
    sa0: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa0: inout std_logic_vector(8 downto 0) := (others=>'Z');
    dqb0: inout std_logic_vector(8 downto 0) := (others=>'Z');
    bwaN0: out std_logic;
    bwbN0: out std_logic;
    bweN0: out std_logic;
    gwN0: out std_logic;
    advN0: out std_logic;
    adscN0: out std_logic;
    adspN0: out std_logic;
    ceN0: out std_logic_vector(3 downto 0);
    oeN0: out std_logic_vector(3 downto 0);
```

```
    zz0: out std_logic;
    cko0: out std_logic;
-- SSRAM1
    sa1: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa1: inout std_logic_vector(8 downto 0) := (others=>'Z');
    dqb1: inout std_logic_vector(8 downto 0) := (others=>'Z');
    bwaN1: out std_logic;
    bwbN1: out std_logic;
    bweN1: out std_logic;
    gwN1: out std_logic;
    advN1: out std_logic;
    adscN1: out std_logic;
    adspN1: out std_logic;
    ceN1: out std_logic_vector(3 downto 0);
    oeN1: out std_logic_vector(3 downto 0);
    zz1: out std_logic;
    cko1: out std_logic;
-- SSRAM2
    sa2: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa2: inout std_logic_vector(8 downto 0) := (others=>'Z');
    dqb2: inout std_logic_vector(8 downto 0) := (others=>'Z');
    bwaN2: out std_logic;
    bwbN2: out std_logic;
    bweN2: out std_logic;
    gwN2: out std_logic;
    advN2: out std_logic;
    adscN2: out std_logic;
    adspN2: out std_logic;
    ceN2: out std_logic_vector(3 downto 0);
    oeN2: out std_logic_vector(3 downto 0);
    zz2: out std_logic;
    cko2: out std_logic;
-- LED
    pLed: out std_logic_vector(11 downto 0);
-- MEZZANINES
-- 3v3 expansion
    b0v33io: inout std_logic_vector(65 downto 0);
    b0v33gClkIo: inout std_logic_vector(7 downto 4);
    b4v33gClkIo: inout std_logic_vector(3 downto 0);
    b6v33io: inout std_logic_vector(21 downto 0);
-- North
    b1v18nIo: inout std_logic_vector(32 downto 0);
    b1v18nGclkIo: inout std_logic_vector(1 downto 0);
    b7v18nIo: inout std_logic_vector(81 downto 0);
-- South
    b1v18sIo: inout std_logic_vector(32 downto 0);
    b1v18sGclkIo: inout std_logic_vector(3 downto 2);
    b2v18sIo: inout std_logic_vector(81 downto 0)
  );
end erinyes;
```

```
-- Erinyes
--
-- Erinyes, test
--
-- Purpose:
--   Tests the major functions of Erinyes
--
-- I/O:
--
--  cpuAddress: cpu address bus
--  cpuData: cpu data bus
--  cpuRw: cpu rw signal
--  cpuOe: cpu output enable
--  cpuCs: cpu chip select
--  cpuRst: fpga reset pin
--  cpuOutInt: cpu interrupt line
--  cpuInClk: cpu clock
--  clkP: differential clock input
--  clkN: differential clock input
--  cpuGsr: in std_logic;
--  cpuInitB: FPGA initialization pin
--  cpuCsB: FPGA programming chip select
--  cpuRdWrB: FPGA programming read/write
--
-- Temperature sensors
--    alertN: overheat alert signal
--    sData: data signal
--    sClk: chip synchronization clock
--
-- Digital potentiometers signals
--    spiDi: spi data in
--    spiDo: spi data out
--    spiRdy: interface ready signal
--    spiCs: chip select
--    spiPrN: spi preset
--    spiClk: spi clock
--
-- SSRAM signals
--  saX: ssram X address lines
--  dqaX: ssram X data line a
--  dqbX: ssram X data line b
--  bwaNX: ssram X byte write byte a
--  bwbNX: ssram X byte write byte b
--  bweNX: ssram X byte write enable
--  gwNX: ssram X global write
--  advNX: ssram X synchronous address advance
--  adscNX: ssram X synchronous address status controller
--  adspNX: ssram X synchronous address status processor
--  ceNX: ssram X clock enable
--  oeNX: ssram X output enable
--  zzX: ssram X snooze mode enable
--  cko: ssrams output clock signal
--
-- LED interface
--   pLed: LED power signals
--
-- MEZZANINES
-- 3v3 expansion
--  b0v33io: 3.3 volt IOs
--  b0v33gClkIo: 3.3 volt global clock pins
--  b4v33gClkIo: 3.3 volt global clock pins
--  b6v33io: 3.3 volt IOs
-- North
--  b1v18nIo: North mezzanine 1.8v IOs
--  b1v18nGclkIo: North mezzanine 1.8v global clock
--  b7v18nIo: North mezzanine 1.8v IOs
-- South
```

```
--  b1v18sIo: South mezzanine 1.8v IOs
--  b1v18sGclkIo: South mezzanine 1.8v global clock
--  b2v18sIo:  South mezzanine 1.8v IOs
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

architecture struct of erinyes is


  component DCM
-- synthesis translate_off
    generic (
      CLK_FEEDBACK :string := "1X";
      CLKDV_DIVIDE : real := 2.0; -- (1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 8.0, 16.0)
      CLKFX_DIVIDE : integer := 1; -- (1 to 4096)
      CLKFX_MULTIPLY : integer := 4; -- (1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 5.5,
      -- 6.0, 6.5, 7.0, 7.5, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0)
      CLKIN_DIVIDE_BY_2 : boolean := FALSE; -- (TRUE, FALSE)
      CLKOUT_PHASE_SHIFT: string := "NONE";
      DESKEW_ADJUST: string := "SYSTEM_SYNCHRONOUS";
      DFS_FREQUENCY_MODE: string := "LOW";
      DLL_FREQUENCY_MODE: string := "LOW";
      DSS_MODE: string := "NONE";
      DUTY_CYCLE_CORRECTION : Boolean := TRUE; -- (TRUE, FALSE)
      FACTORY_JF : bit_vector := X"C080";
      PHASE_SHIFT: integer := 0;
      STARTUP_WAIT :boolean := FALSE
    );
-- synthesis translate_on
    port(
      CLK0 : out std_logic;
      CLK180 : out std_logic;
      CLK270 : out std_logic;
      CLK2X : out std_logic;
      CLK2X180 : out std_logic;
      CLK90 : out std_logic;
      CLKDV : out std_logic;
      CLKFX : out std_logic;
      CLKFX180 : out std_logic;
      LOCKED : out std_logic;
      PSDONE : out std_logic;
      STATUS : out std_logic_vector(7 downto 0);
      CLKFB : in std_logic;
      CLKIN : in std_logic;
      DSSEN : in std_logic;
      PSCLK : in std_logic;
      PSEN : in std_logic;
      PSINCDEC: in std_logic;
      RST : in std_logic
    );
  end component DCM;

  attribute xc_props :string;
  attribute xc_props of dcm0 : label is "PHASE_SHIFT=0";


  component bufG
    port( o: out std_logic; i: in std_logic );
  end component bufG;

  component ibufgds_lvpecl_33 is
    port( o: out std_logic; i: in std_logic; ib: in std_logic );
  end component ibufgds_lvpecl_33;
```

```
attribute syn_black_box : boolean;
attribute syn_black_box of dcm: component is true;
attribute syn_black_box of bufg: component is true;
attribute syn_black_box of ibufgds_lvpecl_33: component is true;


component tSens is
  generic(
    FIFO_DEPTH: integer:= 16;
    CLK_DIVISOR: integer:= 625;
    TRIG_ADDRESS: std_logic_vector:= x"0";
    TRIG_ADDRESS_STATUS: std_logic_vector:= x"0"
  );
  port(
    alrt: out std_logic;
    address: in std_logic_vector(22 downto 0);
    cpuDataIn: in std_logic_vector(31 downto 0);
    cpuDataOut: out std_logic_vector(31 downto 0);
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic;
    alertN: in std_logic;
    sData: inout std_logic;
    sClk: out std_logic
  );
end component tSens;


component dp is
  generic(
    TRIG_ADDRESS: std_logic_vector:= x"0";
    TRIG_ADDRESS_STATUS: std_logic_vector:= x"0"
  );
  port(
    address: in std_logic_vector(22 downto 0);
    cpuDataIn: in std_logic_vector(31 downto 0);
    cpuDataOut: out std_logic_vector(31 downto 0);
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic;
    spiDi: in std_logic;
    spiDo: out std_logic;
    spiRdy: in std_logic;
    spiCs: out std_logic_vector(17 downto 0);
    spiPrN: out std_logic;
    spiClk: out std_logic
  );
end component dp;


component arbiter is
  generic (
    SSRAM0_CTRL_ADDRESS: std_logic_vector:= "110"&x"01000";
    SSRAM1_CTRL_ADDRESS: std_logic_vector:= "110"&x"01004";
    SSRAM2_CTRL_ADDRESS: std_logic_vector:= "110"&x"01008";
    CPU_CTRL_ADDRESS: std_logic_vector:= "110"&x"01010";
    PORT1_CTRL_ADDRESS: std_logic_vector:= "110"&x"01014";
    PORT2_CTRL_ADDRESS: std_logic_vector:= "110"&x"01018";
    CPU_A_WIDTH: integer:= 9;
```

```
    CPU_B_WIDTH: integer:= 9;
    PORT1_A_WIDTH: integer:= 9;
    PORT1_B_WIDTH: integer:= 9;
    PORT2_A_WIDTH: integer:= 9;
    PORT2_B_WIDTH: integer:= 9
  );
  port (
    address: in std_logic_vector(22 downto 0);
    cpuDataIn: in std_logic_vector(31 downto 0);
    cpuDataOut: out std_logic_vector(31 downto 0);
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    cpuClk: in std_logic;

    p1addr: in std_logic_vector(22 downto 0);
    p1dataIn: in std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
    p1dataOut: out std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
    p1rdy: out std_logic;
    p1rw: in std_logic;
    p1ce: in std_logic;

    p2addr: in std_logic_vector(22 downto 0);
    p2dataIn: in std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
    p2dataOut: out std_logic_vector(PORT1_A_WIDTH+PORT1_B_WIDTH-1 downto 0);
    p2rdy: out std_logic;
    p2rw: in std_logic;
    p2ce: in std_logic;

    rst: in std_logic;
    clk: in std_logic;

-- SSRAM0
    sa0: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa0: inout std_logic_vector(8 downto 0) := (others=>'Z');
    dqb0: inout std_logic_vector(8 downto 0) := (others=>'Z');
    bwaN0: out std_logic;
    bwbN0: out std_logic;
    bweN0: out std_logic;
    gwN0: out std_logic;
    advN0: out std_logic;
    adscN0: out std_logic;
    adspN0: out std_logic;
    ceN0: out std_logic_vector(3 downto 0);
    oeN0: out std_logic_vector(3 downto 0);
    zz0: out std_logic;
    cko0: out std_logic;

-- SSRAM1
    sa1: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa1: inout std_logic_vector(8 downto 0) := (others=>'Z');
    dqb1: inout std_logic_vector(8 downto 0) := (others=>'Z');
    bwaN1: out std_logic;
    bwbN1: out std_logic;
    bweN1: out std_logic;
    gwN1: out std_logic;
    advN1: out std_logic;
    adscN1: out std_logic;
    adspN1: out std_logic;
    ceN1: out std_logic_vector(3 downto 0);
    oeN1: out std_logic_vector(3 downto 0);
    zz1: out std_logic;
    cko1: out std_logic;

-- SSRAM2
    sa2: out std_logic_vector(18 downto 0) := (others=>'0');
    dqa2: inout std_logic_vector(8 downto 0) := (others=>'Z');
```

```
    dqb2: inout std_logic_vector(8 downto 0) := (others=>'Z');
    bwaN2: out std_logic;
    bwbN2: out std_logic;
    bweN2: out std_logic;
    gwN2: out std_logic;
    advN2: out std_logic;
    adscN2: out std_logic;
    adspN2: out std_logic;
    ceN2: out std_logic_vector(3 downto 0);
    oeN2: out std_logic_vector(3 downto 0);
    zz2: out std_logic;
    cko2: out std_logic
  );
end component arbiter;



component led is
  generic(
    NB_STATES: integer:= 5;
    TRIG_ADDRESS: std_logic_vector:= x"0"
  );
  port(
    address: in std_logic_vector(22 downto 0);
    cpuDataIn: in std_logic_vector(31 downto 0);
    cpuDataOut: out std_logic_vector(31 downto 0);
    rw: in std_logic;
    oe: in std_logic;
    cs: in std_logic;
    rst: in std_logic;
    cpuClk: in std_logic;
    clk: in std_logic;
    pLed: out std_logic_vector(11 downto 0)
  );
end component led;



component cpuFrontEnd is
  generic(
    constant ADDR_WIDTH: integer := 24;
    constant CPU_DATA_WIDTH: integer := 32
  );
  port(
    cpuAddress: in std_logic_vector(ADDR_WIDTH-1 downto 0);
    cpuData: inout std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
    cpuRw: in std_logic;
    cpuOe: in std_logic;
    cpuCs: in std_logic;
    cpuRst: in std_logic;
    cpuClk: in std_logic;
    cpuInt: out std_logic;
    cpuGsr: in std_logic;
    cpuInitB: inout std_logic;
    cpuCsB: in std_logic;
    cpuRdWrB: in std_logic;

    address: out std_logic_vector(ADDR_WIDTH-1 downto 0);
    dataIn: out std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
    dataOut: in std_logic_vector(CPU_DATA_WIDTH-1 downto 0);
    rw: out std_logic;
    oe: out std_logic;
    cs: out std_logic;
    rst: out std_logic;
    clk: out std_logic;
    int: in std_logic;
```

```
    gsr: out std_logic;
    initBin: out std_logic;
    initBout: in std_logic;
    initBt: in std_logic;
    csB: out std_logic;
    rdWrB: out std_logic
  );
end component cpuFrontEnd;


component mezzanine33 is
  port (
    b0v33io: inout std_logic_vector(65 downto 0);
    b0v33gClkIo: inout std_logic_vector(7 downto 4);
    b4v33gClkIo: inout std_logic_vector(3 downto 0);
    b6v33io: inout std_logic_vector(21 downto 0);

    b0v33i: out std_logic_vector(65 downto 0);
    b0v33o: in std_logic_vector(65 downto 0);
    b0v33t: in std_logic_vector(65 downto 0);
    b0v33gClkI: out std_logic_vector(7 downto 4);
    b0v33gClkO: in std_logic_vector(7 downto 4);
    b0v33gClkT: in std_logic_vector(7 downto 4);
    b4v33gClkI: out std_logic_vector(3 downto 0);
    b4v33gClkO: in std_logic_vector(3 downto 0);
    b4v33gClkT: in std_logic_vector(3 downto 0);
    b6v33i: out std_logic_vector(21 downto 0);
    b6v33o: in std_logic_vector(21 downto 0);
    b6v33t: in std_logic_vector(21 downto 0)
  );
end component mezzanine33;




component mezzanine18 is
  port (
    v18io: inout std_logic_vector(32 downto 0);
    v18gClkIo: inout std_logic_vector(1 downto 0);
    v18io1: inout std_logic_vector(81 downto 0);

    v18i: out std_logic_vector(32 downto 0);
    v18o: in std_logic_vector(32 downto 0);
    v18t: in std_logic_vector(32 downto 0);
    v18gClki: out std_logic_vector(1 downto 0);
    v18gClko: in std_logic_vector(1 downto 0);
    v18gClkt: in std_logic_vector(1 downto 0);
    v18i1: out std_logic_vector(81 downto 0);
    v18o1: in std_logic_vector(81 downto 0);
    v18t1: in std_logic_vector(81 downto 0)
  );
end component mezzanine18;



-- interconnect
-- CPU
  signal address: cpuAddress_typ;
  signal dataIn, dataOut: cpuData_typ;
  signal rd, rw, oe, cs, rst, rst_w: std_logic;
  signal cpuClk, cpuInt, gsr: std_logic;
  signal initBin, initBout, initBt: std_logic;
  signal csB, rdWrB: std_logic;
```

```
-- DCM
  signal clk0, clk180, clk270, clk2x, clk2x180: std_logic;
  signal clk90, clkDv: std_logic;
  signal clkFx, clkFx180, locked, psDone: std_logic;
  signal stat: std_logic_vector(7 downto 0);
  signal fClk, mClk: std_logic;
  signal dssEn, psClk, psEn, psIncDec: std_logic;

  attribute syn_isclock : boolean;
  attribute syn_isclock of clk0: signal is true;
  attribute syn_isclock of mClk: signal is true;



-- MEMORY ARBITER
  signal p1addr: cpuAddress_typ;
  signal p1dataIn, p1dataOut: std_logic_vector(17 downto 0);
  signal p1rdy, p1rw, p1ce: std_logic;
  signal p2addr: cpuAddress_typ;
  signal p2dataIn, p2dataOut: std_logic_vector(17 downto 0);
  signal p2rdy, p2rw, p2ce: std_logic;

-- TEMPERATURE SENSORS
  signal alert: std_logic;


-- MEZZANINES
  signal b0v33i, b0v33o, b0v33t: std_logic_vector(65 downto 0);
  signal b0v33gClkI, b0v33gClkO, b0v33gClkT: std_logic_vector(7 downto 4);
  signal b4v33gClkI, b4v33gClkO, b4v33gClkT: std_logic_vector(3 downto 0);
  signal b6v33i, b6v33o, b6v33t: std_logic_vector(21 downto 0);


  signal b1v18nI, b1v18nO, b1v18nT: std_logic_vector(32 downto 0);
  signal b1v18nGclkI, b1v18nGclkO, b1v18nGclkT: std_logic_vector(1 downto 0);
  signal b7v18nI, b7v18nO, b7v18nT: std_logic_vector(81 downto 0);

  signal b1v18sI, b1v18sO, b1v18sT: std_logic_vector(32 downto 0);
  signal b1v18sGclkI, b1v18sGclkO, b1v18sGclkT: std_logic_vector(3 downto 2);
  signal b2v18sI, b2v18sO, b2v18sT: std_logic_vector(81 downto 0);



-- Generic constants

-- MEMORY ARBITER
  constant ssram0ctrl_c: cpuAddress_typ := "111"&x"00100";
  constant ssram1ctrl_c: cpuAddress_typ := "111"&x"00104";
  constant ssram2ctrl_c: cpuAddress_typ := "111"&x"00108";
  constant cpuCtrl_c: cpuAddress_typ := "111"&x"0010c";
  constant port1ctrl_c: cpuAddress_typ := "111"&x"00110";
  constant port2ctrl_c: cpuAddress_typ := "111"&x"00114";
  constant cpuAwidth_c: integer := 9;
  constant cpuBwidth_c: integer := 9;
  constant port1aWidth_c: integer := 9;
  constant port1bWidth_c: integer := 9;
  constant port2aWidth_c: integer := 9;
  constant port2bWidth_c: integer := 9;

-- DIGITAL POTENTIOMETERS
  constant dpData_c: cpuAddress_typ := "111"&x"00300";
  constant dpStatus_c: cpuAddress_typ := "111"&x"00304";

-- TEMPERATURE SENSORS
  constant tempSensFifoDepth_c: integer := 16;
```

```
  constant tempSensClkDiv_c: integer := 625;
  constant tempSensData_c: cpuAddress_typ := "111"&x"00400";
  constant tempSensStatus_c: cpuAddress_typ := "111"&x"00404";

-- LED
  constant ledStates_c: integer := 5;
  constant ledData_c: cpuAddress_typ := "111"&x"00500";


begin

  rst_w <= (not rst or not locked) or not gsr;

-- ports 1 and 2 are unused for this test
  p1addr <= (others=>'0');
  p1dataIn <= (others=>'0');
  p1dataOut <= (others=>'0');
  p1rdy <= '0';
  p1rw <='0';
  p1ce <= '0';
  p2addr <= (others=>'0');
  p2dataIn <= (others=>'0');
  p2dataOut <= (others=>'0');
  p2rdy <= '0';
  p2rw <='0';
  p2ce <= '0';


-- mezzanines IO high-Z
  b0v33t <= (others=>'1');
  b0v33gClkT <= (others=>'1');
  b4v33gClkT <= (others=>'1');
  b6v33t <= (others=>'1');
-- North
  b1v18nT <= (others=>'1');
  b1v18nGclkT <= (others=>'1');
  b7v18nT <= (others=>'1');
-- South
  b1v18sT <= (others=>'1');
  b1v18sGclkT <= (others=>'1');
  b2v18sT <= (others=>'1');

-- FPGA control signals inactive
  initBout <= '0';
  initBt <= '1';



-- Clock distribution
  inBufG: ibufgds_lvpecl_33
  port map( fClk, clkP, clkN );

  bufg0: bufG
  port map( mClk, clk0 );

  dcm0: DCM
-- synthesis translate_off
  generic map( PHASE_SHIFT => 0 )
-- synthesis translate_on
  port map( clk0, clk180, clk270, clk2x, clk2x180, clk90,
            clkDv, clkFx, clkFx180, locked, psDone, stat,
            mClk, fClk, dssEn, psClk, psEn, psIncDec, '0');



-- Memory arbiter
```

```
  arb0: arbiter
  generic map(
    ssram0ctrl_c, ssram1ctrl_c, ssram2ctrl_c,
    cpuCtrl_c, port1ctrl_c, port2ctrl_c,
    cpuAwidth_c, cpuBwidth_c,
    port1aWidth_c, port1bWidth_c,
    port2aWidth_c, port2bWidth_c
  )
  port map (
    address, dataIn, dataOut, rw, oe, cs, cpuClk,
    p1addr, p1dataIn, p1dataOut, p1rdy, p1rw, p1ce,
    p2addr, p2dataIn, p2dataOut, p2rdy, p2rw, p2ce,
    rst_w, mClk,
    sa0, dqa0, dqb0, bwaN0, bwbN0, bweN0, gwN0, advN0,
    adscN0, adspN0,
    ceN0, oeN0, zz0, cko0,
    sa1, dqa1, dqb1, bwaN1, bwbN1, bweN1, gwN1, advN1,
    adscN1, adspN1,
    ceN1, oeN1, zz1, cko1,
    sa2, dqa2, dqb2, bwaN2, bwbN2, bweN2, gwN2, advN2,
    adscN2, adspN2,
    ceN2, oeN2, zz2, cko2
  );


-- Digital potentiometers
  dp0: dp
  generic map( dpData_c, dpStatus_c )
  port map(
    address, dataIn, dataOut, rw, oe, cs, rst_w,
    cpuClk, mClk,
    spiDi, spiDo, spiRdy, spiCs, spiPrN, spiClk
  );


-- Temperature sensors
  tSens0: tSens
  generic map(
    tempSensFifoDepth_c, tempSensClkDiv_c,
    tempSensData_c, tempSensStatus_c
  )
  port map(
    alert, address, dataIn, dataOut, rw, oe, cs,
    rst_w, cpuClk, mClk,
    alertN, sData, sClk
  );
  cpuInt <= not alert;


-- LED
  led0: led
  generic map( ledStates_c, ledData_c )
  port map(
    address, dataIn, dataOut, rw, oe, cs, rst_w,
    cpuClk, mClk, pLed
  );


-- CPU front-end
  cpu0: cpuFrontEnd
  generic map( 23, 32 )
  port map(
    cpuAddress, cpuData, cpuRw, cpuOe, cpuCs, cpuRst, cpuInClk,
    cpuOutInt, cpuGsr, cpuInitB, cpuCsB, cpuRdWrB,
```

```
    address, dataIn, dataOut, rw, oe, cs, rst, cpuClk,
    cpuInt, gsr, initBin, initBout, initBt, csB, rdWrB
  );

-- Mezzanines
  mez33: mezzanine33
  port map(
    b0v33io, b0v33gClkIo, b4v33gClkIo, b6v33io,
    b0v33i, b0v33o, b0v33t,
    b0v33gClkI, b0v33gClkO, b0v33gClkT,
    b4v33gClkI, b4v33gClkO, b4v33gClkT,
    b6v33i, b6v33o, b6v33t
  );

  mezNorth: mezzanine18
  port map(
    b1v18nIo, b1v18nGclkIo, b7v18nIo,
    b1v18nI, b1v18nO, b1v18nT,
    b1v18nGclkI, b1v18nGclkO, b1v18nGclkT,
    b7v18nI, b7v18nO, b7v18nT
  );

  mezSouth: mezzanine18
  port map(
    b1v18sIo, b1v18sGclkIo, b2v18sIo,
    b1v18sI, b1v18sO, b1v18sT,
    b1v18sGclkI, b1v18sGclkO, b1v18sGclkT,
    b2v18sI, b2v18sO, b2v18sT
  );

end struct;
```

```
-- Erinyes
--
-- Erinyes, test
--
-- Normand Leclerc, B.ing.
-- Ecole de technologie superieure
-- January, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.arbiter_pak.all;
use work.cpu_pak.all;
use work.utils_pak.all;
use work.simCpu_pak.all;
use work.cpu;

library unisim;
use unisim.dcm;

entity erinyes_tb is
end erinyes_tb;


architecture behav of erinyes_tb is


  component CPU is
    generic(
      CLOCK_PERIOD: time := cpuClockPeriod_c;
      WAIT_STATE: integer := 0
    );
    port(
      address: out address_typ;
      data: inout data_typ;
      rw: out std_logic;
      cs: out std_logic;
      oe: out std_logic;
      rst: in std_logic;
      clk: out std_logic
    );
  end component CPU;



  component erinyes is
    port (
-- CPU
      cpuAddress: in std_logic_vector(22 downto 0);
      cpuData: inout std_logic_vector(31 downto 0);
      cpuRw: in std_logic;

      cpuOe: in std_logic;
      cpuCs: in std_logic;
      cpuRst: in std_logic;
      cpuOutInt: out std_logic;
      cpuInClk: in std_logic;
      clkP: in std_logic;
      clkN: in std_logic;
      cpuGsr: in std_logic;
      cpuInitB: inout std_logic;
      cpuCsB: in std_logic;
      cpuRdWrB: in std_logic;
```

```
-- TEMPERATURE SENSORS
      alertN: in std_logic;
      sData: inout std_logic;
      sClk: out std_logic;

-- DIGITAL POTENTIOMETERS
      spiDi: in std_logic;
      spiDo: out std_logic;
      spiRdy: in std_logic;
      spiCs: out std_logic_vector(17 downto 0);
      spiPrN: out std_logic;
      spiClk: out std_logic;

-- SSRAM0
      sa0: out std_logic_vector(18 downto 0) := (others=>'0');
      dqa0: inout std_logic_vector(8 downto 0) := (others=>'Z');
      dqb0: inout std_logic_vector(8 downto 0) := (others=>'Z');
      bwaN0: out std_logic;
      bwbN0: out std_logic;
      bweN0: out std_logic;
      gwN0: out std_logic;
      advN0: out std_logic;
      adscN0: out std_logic;
      adspN0: out std_logic;
      ceN0: out std_logic_vector(3 downto 0);
      oeN0: out std_logic_vector(3 downto 0);
      zz0: out std_logic;
      cko0: out std_logic;

-- SSRAM1
      sa1: out std_logic_vector(18 downto 0) := (others=>'0');
      dqa1: inout std_logic_vector(8 downto 0) := (others=>'Z');
      dqb1: inout std_logic_vector(8 downto 0) := (others=>'Z');
      bwaN1: out std_logic;
      bwbN1: out std_logic;
      bweN1: out std_logic;
      gwN1: out std_logic;
      advN1: out std_logic;
      adscN1: out std_logic;
      adspN1: out std_logic;
      ceN1: out std_logic_vector(3 downto 0);
      oeN1: out std_logic_vector(3 downto 0);
      zz1: out std_logic;
      cko1: out std_logic;

-- SSRAM2
      sa2: out std_logic_vector(18 downto 0) := (others=>'0');
      dqa2: inout std_logic_vector(8 downto 0) := (others=>'Z');
      dqb2: inout std_logic_vector(8 downto 0) := (others=>'Z');
      bwaN2: out std_logic;
      bwbN2: out std_logic;
      bweN2: out std_logic;
      gwN2: out std_logic;
      advN2: out std_logic;
      adscN2: out std_logic;
      adspN2: out std_logic;
      ceN2: out std_logic_vector(3 downto 0);
      oeN2: out std_logic_vector(3 downto 0);
      zz2: out std_logic;
      cko2: out std_logic;

-- LED
      pLed: out std_logic_vector(11 downto 0);

-- MEZZANINES
-- 3v3 expansion
    b0v33io: inout std_logic_vector(65 downto 0);
```

```
    b0v33gClkIo: inout std_logic_vector(7 downto 4);
    b4v33gClkIo: inout std_logic_vector(3 downto 0);
    b6v33io: inout std_logic_vector(21 downto 0);
-- North
    b1v18nIo: inout std_logic_vector(32 downto 0);
    b1v18nGclkIo: inout std_logic_vector(1 downto 0);
    b7v18nIo: inout std_logic_vector(81 downto 0);
-- South
    b1v18sIo: inout std_logic_vector(32 downto 0);
    b1v18sGclkIo: inout std_logic_vector(3 downto 2);
    b2v18sIo: inout std_logic_vector(81 downto 0)
    );
  end component erinyes;


-- Generic constants

-- MEMORY ARBITER
  constant ssram0ctrl_c: cpuAddress_typ := "111"&x"00100";
  constant ssram1ctrl_c: cpuAddress_typ := "111"&x"00104";
  constant ssram2ctrl_c: cpuAddress_typ := "111"&x"00108";
  constant cpuCtrl_c: cpuAddress_typ := "111"&x"0010c";
  constant port1ctrl_c: cpuAddress_typ := "111"&x"00110";
  constant port2ctrl_c: cpuAddress_typ := "111"&x"00114";

-- DIGITAL POTENTIOMETERS
  constant dpData_c: cpuAddress_typ := "111"&x"00300";
  constant dpStatus_c: cpuAddress_typ := "111"&x"00304";

-- TEMPERATURE SENSORS
  constant tempSensData_c: cpuAddress_typ := "111"&x"00400";
  constant tempSensStatus_c: cpuAddress_typ := "111"&x"00404";

-- LED
  constant ledData_c: cpuAddress_typ := "111"&x"00500";

-- FPGA CLOCK
  constant fpgaClkPeriod_c: time := 8 ns;



-- interconnect
  signal clkP: std_logic := '0';
  signal clkN: std_logic;
  signal clk: std_logic;
  signal gsr: std_logic;
  signal initB: std_logic;
  signal csB: std_logic;
  signal rdWrB: std_logic;


-- cpu interface
  signal address: address_typ;
  signal cpuData: data_typ;
  signal rw, oe, cs, rst, cpuClk: std_logic;
  signal cpuInt: std_logic;

-- temperature sensors
  signal alertN, sData, sClk: std_logic;

-- digital potentiometers
  signal spiDi, spiDo, spiRdy: std_logic;
  signal spiCs: std_logic_vector(17 downto 0);
  signal spiPrN: std_logic;
  signal spiClk: std_logic;
  signal spiData: std_logic_vector(23 downto 0)
```

```
                := x"abcdef";

-- ssram0
  signal sa0: std_logic_vector(18 downto 0);
  signal dqa0, dqb0: std_logic_vector(8 downto 0);
  signal bwaN0, bwbN0, bweN0, gwN0: std_logic;
  signal advN0, adscN0, adspN0: std_logic;
  signal ceN0: std_logic_vector(3 downto 0);
  signal oeN0: std_logic_vector(3 downto 0);
  signal zz0, cko0: std_logic;

-- ssram1
  signal sa1: std_logic_vector(18 downto 0);
  signal dqa1, dqb1: std_logic_vector(8 downto 0);
  signal bwaN1, bwbN1, bweN1, gwN1: std_logic;
  signal advN1, adscN1, adspN1: std_logic;
  signal ceN1: std_logic_vector(3 downto 0);
  signal oeN1: std_logic_vector(3 downto 0);
  signal zz1, cko1: std_logic;

-- ssram2
  signal sa2: std_logic_vector(18 downto 0);
  signal dqa2, dqb2: std_logic_vector(8 downto 0);
  signal bwaN2, bwbN2, bweN2, gwN2: std_logic;
  signal advN2, adscN2, adspN2: std_logic;
  signal ceN2: std_logic_vector(3 downto 0);
  signal oeN2: std_logic_vector(3 downto 0);
  signal zz2, cko2: std_logic;

-- LED
  signal pLed: std_logic_vector(11 downto 0);


-- Mezzanines
  signal b0v33io: std_logic_vector(65 downto 0);
  signal b0v33gClkIo: std_logic_vector(7 downto 4);
  signal b4v33gClkIo: std_logic_vector(3 downto 0);
  signal b6v33io: std_logic_vector(21 downto 0);
  signal b1v18nIo: std_logic_vector(32 downto 0);
  signal b1v18nGclkIo: std_logic_vector(1 downto 0);
  signal b7v18nIo: std_logic_vector(81 downto 0);
  signal b1v18sIo: std_logic_vector(32 downto 0);
  signal b1v18sGclkIo: std_logic_vector(3 downto 2);
  signal b2v18sIo: std_logic_vector(81 downto 0);


begin


  cpu0: cpu
  generic map( WAIT_STATE=>2 )
  port map( address, cpuData, rw, cs, oe, rst, cpuClk );

  erinyes0: erinyes
  port map (
    address, cpuData, rw, oe, cs, rst, cpuInt, cpuClk, clkP,
    clkN, gsr, initB, csB, rdWrB,
    alertN, sData, sClk,
    spiDi, spiDo, spiRdy, spiCs, spiPrN, spiClk,
    sa0, dqa0, dqb0, bwaN0, bwbN0, bweN0, gwN0,
    advN0, adscN0, adspN0, ceN0, oeN0, zz0, cko0,
    sa1, dqa1, dqb1, bwaN1, bwbN1, bweN1, gwN1,
    advN1, adscN1, adspN1, ceN1, oeN1, zz1, cko1,
    sa2, dqa2, dqb2, bwaN2, bwbN2, bweN2, gwN2,
    advN2, adscN2, adspN2, ceN2, oeN2, zz2, cko2,
```

```
    pLed, b0v33io, b0v33gClkIo, b4v33gClkIo, b6v33io,
    b1v18nIo, b1v18nGclkIo, b7v18nIo,
    b1v18sIo, b1v18sGclkIo, b2v18sIo
  );


  clkP <= not clkP after fpgaClkPeriod_c/2;
  clkN <= not clkP;


-- outside simulation
  spiShift: process(spiClk)
  begin
    clkIf: if( rising_edge(spiClk) ) then
      spiData <= spiData(spiData'high-1 downto 0) & spiData(spiData'high);
    end if clkIf;

  end process spiShift;
  spiDi <= spiData(spiData'high);

-- spi devices are ready
  spiRdy <= '1';

-- no alert on smbus
  alertN <= '1';

-- fpga programming signals down
  initB <= '1';
  csB <= '0';
  rdWrB <= '0';


-- simulate ssram

  dqa0 <= '0'&x"0a" when bweN0='1' else (others=>'Z');
  dqb0 <= '0'&x"0b" when bweN0='1' else (others=>'Z');
  dqa1 <= '0'&x"1a" when bweN1='1' else (others=>'Z');
  dqb1 <= '0'&x"1b" when bweN1='1' else (others=>'Z');
  dqa2 <= '0'&x"2a" when bweN2='1' else (others=>'Z');
  dqb2 <= '0'&x"2b" when bweN2='1' else (others=>'Z');



  test: process
  begin
    gsr <= '1';
    rst <= '0';
    wait for cpuClockPeriod_c;
    rst <= '1';
    wait for cpuClockPeriod_c;

    report "Waiting for DCM to lock";
    wait for 50*cpuClockPeriod_c;


    report "Digital potentiometers test";
    cpuWrite(dpData_c, x"08123456", cpuSig_rec);
    dpPollLoop: while cpuData(0)/='1' loop
      cpuRead(dpStatus_c, cpuSig_rec);
    end loop dpPollLoop;
    cpuRead(dpData_c, cpuSig_rec);

    wait for 10*fpgaClkPeriod_c;

--    report "Temperature sensor test";
--    cpuWrite(tempSensData_c, x"0085abcd", cpuSig_rec);
--    tSpollStatLoop: while( cpuData(0)/='1' ) loop
```

```
--      cpuRead(tempSensStatus_c, cpuSig_rec);
--    end loop tSpollStatLoop;

--    wait for 10*fpgaClkPeriod_c;

    report "LED test";
    cpuWrite(ledData_c, x"00000003", cpuSig_rec);
    cpuWrite(ledData_c+1, x"00000002", cpuSig_rec);
    cpuWrite(ledData_c+6, x"00000001", cpuSig_rec);
    cpuWrite(ledData_c+11, x"00000000", cpuSig_rec);

    wait for 10*fpgaClkPeriod_c;

    report "Arbiter test";

    report "Writing config.";
    cpuWrite(ssram0ctrl_c, x"00000001", cpuSig_rec);
    cpuWrite(ssram1ctrl_c, x"00000000", cpuSig_rec);
    cpuWrite(ssram2ctrl_c, x"00000000", cpuSig_rec);
    cpuWrite(cpuCtrl_c, x"00000003", cpuSig_rec);
    cpuWrite(port1ctrl_c, x"00000003", cpuSig_rec);
    cpuWrite(port2ctrl_c, x"00000003", cpuSig_rec);

    report "Reading config back.";
    cpuRead(ssram0ctrl_c, cpuSig_rec);
    cpuRead(ssram1ctrl_c, cpuSig_rec);
    cpuRead(ssram2ctrl_c, cpuSig_rec);
    cpuRead(cpuCtrl_c, cpuSig_rec);
    cpuRead(port1ctrl_c, cpuSig_rec);
    cpuRead(port2ctrl_c, cpuSig_rec);

    report "Reading memory.";
    cpuRead("000"&x"00002", cpuSig_rec);
    report "Writing...";
    cpuWrite("000"&x"00020", x"fad87654", cpuSig_rec);
    report "Reading...";
    cpuRead("000"&x"00200", cpuSig_rec);

    report "Writing config.";
    cpuWrite(ssram0ctrl_c, x"00000000", cpuSig_rec);
    cpuWrite(ssram1ctrl_c, x"00000001", cpuSig_rec);

    report "Reading memory.";
    cpuRead("000"&x"00002", cpuSig_rec);
    report "Writing...";
    cpuWrite("000"&x"00020", x"fad87654", cpuSig_rec);
    report "Reading...";
    cpuRead("000"&x"00200", cpuSig_rec);

    report "Writing config.";
    cpuWrite(ssram1ctrl_c, x"00000000", cpuSig_rec);
    cpuWrite(ssram2ctrl_c, x"00000001", cpuSig_rec);

    report "Reading memory.";
    cpuRead("000"&x"00002", cpuSig_rec);
    report "Writing...";
    cpuWrite("000"&x"00020", x"fad87654", cpuSig_rec);
    report "Reading...";
    cpuRead("000"&x"00200", cpuSig_rec);

    report "Tests done.";
    wait;
  end process test;

end behav
```