UNIVERSITAS INDONESIA **UNIVERSITE DE BRETAGNE SUD**

STUDI COUPLED UNTUK AKSELERATOR HARDWARE PADA SOC (SYSTEM ON CHIP) BERBASIS LEON

TESIS

A SUMARUDIN
1006803915

FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK ELEKTRO
DEPOK
JULI 2012

UNIVERSITAS INDONESIA **UNIVERSITE DE BRETAGNE SUD**

STUDI COUPLED UNTUK AKSELERATOR HARDWARE PADA SOC (SYSTEM ON CHIP) BERBASIS LEON

TESIS

Diajukan sebagai salah satu syarat untuk memperoleh gelar Magister Teknik

A SUMARUDIN
1006803915

FAKULTAS TEKNIK

PROGRAM STUDI TEKNIK ELEKTRO

KEKHUSUSAN JARINGAN INFORMASI DAN MULTIMEDIA

DEPOK

JULI 2012

HALAMAN PERNYATAAN ORISINALITAS

Tesis ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	**: A SUMARUDIN**
NPM	**: 1006803915**
Tanda Tangan	**:** 
Tanggal	**: 27 JUNI 2012**

Studi coupled..., A. Sumarudin, FT UI, 2012

LEMBAR PENGESAHAN

Tesis ini diajukan oleh

Nama	:	A Sumarudin
NPM	:	1006803915
Program Studi	:	Teknik Elektro
Judul Tesis	:	Studi coupled untuk akselerator hardware pada SoC (System on Chip) berbasis LEON

Telah berhasil dipertahankan dihadapan dewan penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Master 2 Université de Bretagne-Sud (Perancis) dan Magister Teknik Program Studi Teknik Elektro, Fakultas Teknik, Universitas Indonesia

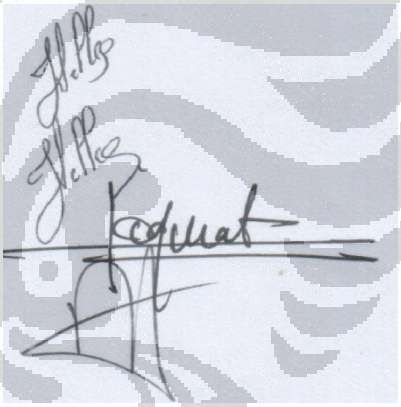
DEWAN PENGUJI

Pembimbing : Dr. Dominique HELLER

Penguji : Dr. Dominique HELLER

Penguji : Prof. Guy GOGNIAT

Penguji : Prof. Marc SEVAUX

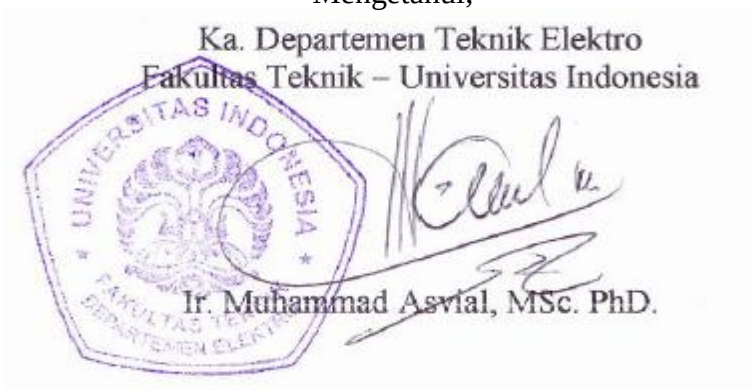


Ditetapkan di : Lorient, Prancis

Tanggal : 27 Juni 2012

Mengetahui,

Ka. Departemen Teknik Elektro
Fakultas Teknik – Universitas Indonesia



Ir. Muhammad Asvial, MSc. PhD.

Studi coupled..., A. Sumarudin, FT UI, 2012

KATA PENGANTAR

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya, saya dapat menyelesaikan tesis ini. Penulisan tesis ini dilakukan dalam rangka memenuhi salah satu syarat untuk mencapai gelar magister Teknik program studi Teknik Elektro pada Fakultas Teknik Universitas Indonesia. Saya menyadari bahwa, tanpa bantuan dan bimbingan dari berbagai pihak, dari masa perkuliahan sampai pada penyusunan tesis ini, sangatlah sulit bagi saya untuk menyelesaikan tesis ini. Oleh karena itu, saya mengucapkan terima kasih kepada:

1. Dr. Dominique Heller, selaku dosen pembimbing yang telah menyediakan waktu, tenaga, dan pikiran untuk mengarahkan saya dalam penyusunan tesis ini;
2. Lab-STICC université de bretagne sud yang telah menyediakan kesempatan untuk melakukan penelitian dan perangkat yang mendukung penelitian ini;
3. Istri tercinta Irma Mardiati dan anakku tersayang Adeevaqila Shafaa Elmardhia, yang merupakan sumber inspirasi dan motivasi.
4. orang tua dan keluarga penulis yang telah memberikan bantuan dukungan; serta
5. sahabat master 2 université de bretgane sud, alhadj abou bakar, nono woffo dan chabane yang telah banyak membantu dalam menyelesaikan tesis ini.

Akhir kata, saya berharap Tuhan Yang Maha Esa berkenan membalas segala kebaikan semua pihak yang telah membantu. Semoga tesis ini membawa manfaat bagi pengembangan ilmu.

Lorient, 27 Juni 2012

Penulis

Studi coupled..., A. Sumarudin, FT UI, 2012

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI
TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama : A Sumarudin

NPM : 1006803915

Program Studi : Jaringan Informasi dan Multimedia

Departemen : Teknik Elektro

Fakultas : Teknik

Jenis karya : Tesis

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty-Free Right*)** atas karya ilmiah saya yang berjudul :

Studi coupled untuk akselerator hardware Pada SoC (system on chip) berbasis LEON

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Lorient, Prancis

Pada tanggal : 27 Juni 2012

Yang menyatakan

(A Sumarudin)

ABSTRAK

Nama : A Sumarudin

Program Studi : Teknik Elektro

Judul : Studi *coupled* untuk akselerator hardware pada SoC (*system on chip*) berbasis LEON

Tujuan dari tesis ini adalah studi *coupled* untuk akselarator pada SoC berbasis arsistektur LEON3. *Coupling* ini menggunakan FIFO FSL (*Fast simplex Link*) bus dan direalisasikan pada FPGA virtex-6 (board *Xilinx ML605)*. Desain dasar menggunakan APB bus, karena kemudahan dalam segi protokol dan rendah konsumsi energinya. Sedangkan untuk *FSL Converter* menggunakan *Xilinx ISE Library*. IP ini ditambahkan dengan menggunakan prinsip *plug&pluy extending* dari IP GRLIB. Desain *coupled* ini menggunakan prinsip FSM (*finite state machine*), simulasi menggunakan ModelSim 6, dan logika sintesis menggunaan ISE. Test *real-time* menggunakan monitor GRMON. Hasil tes menunjukan penggunaan APB tidak bisa digunakan untuk *coupled tightly* pada LEON3 karena menggunakan AMBA V.2 berbeda dengan versi 3 yang tidak terbatas, jika dibutuhkan untuk dirubah (tanpa PReady, twait) maka harus diganti dengan menggunakan AHB Bus.

Kata Kunci:

LEON 3, GRLIB, GRMON, AMBA bus, Tightly-coupled, FSL, FSM

ABSTRACT

Name : A Sumarudin

Study Program : Electrical Engineering

Title : Study Of Coupled for Hardware Accelerator on Soc (System on Chip) Based On LEON

The objectives of the thesis are the studies the coupling of a hardware accelerator in a SoC based on leon3. Coupling created using FSL (Fast Simplex Link*)* bus with AMBA bus leon3 programmed in FPGA Virtex-6 (Xilinx ML605 board prototype). The initial design use APB bus, because of the simplicity of the APB protocol and low power. FSL converter use Xilinx ISE library. Addition of this IP uses plug&play Extending IP GRLIB. This design coupled includes design based on FSM (finite state machine), simulation using ModelSim and logic synthesis using ISE. The test is based on real-time monitor GRMon. Test results show for the APB could not be tightly coupled because of the SoC based on LEON3 using AMBA bus v2, which unlike the version 3 does not freiner if need an exchange (no pready, Twait): it must be changed using the AHB bus.

Keywords:

LEON 3, GRLIB, GRMON, AMBA bus, Tightly-coupled, FSL, FSM

RESUMES

NOM, Prénom : SUMARUDIN, A
Disciplinaire : Génie Electronique
Titre de sujet : Etude du couplage d'un accélérateur Hardware sur un SoC (System on CHIP) à base du LEON

L'objectif du stage est l'étude du couplage d'un accélérateur hardware dans un soc à base de leon3. Cette étude repose sur l'utilisation d'une Fifo FSL (*Fast Simplex Link) et* la programmation d'un FPGA Virtex-6 (carte Xilinx ML605). La conception initiale utilisant le bus APB, à cause de la simplicité du protocole APB. La FSL est une Fifo de la bibliothèque Xilinx ISE. L'addition de cette IP repose sur le *plug&pluy extending IP* de la GRLIB. Cette conception inclut la conception à base du FSM (*finite state machine*), la simulation en utilisant ModelSim et la synthèse logique utilisant ISE. Le test temps-réel repose sur le moniteur GRMon. Les résultats des tests montrent que l'APB ne peut pas être « tighly » couplé à cause du fait que le SoC à base de Leon3 utilise la version 2 de l'AMBA bus, qui contrairement à la version 3 ne permet pas de « freiner » si besoin un échange (pas de pready, twait) : il doit donc être changé en utilisant le bus AHB.

Motts-clefs: LEON 3, GRLIB, GRMON, AMBA bus, Tightly-coupled, FSL, FSM

DAFTAR ISI

DAFTAR GAMBAR

Studi coupled..., A. Sumarudin, FT UI, 2012

DAFTAR LAMPIRAN

CHAPITRE I
Introduction

1.1 Introduction Générale

L'objectif de stage est d'étudier le couplage d'un accélérateur hardware dans un Soc à base de Leon3. Le LEON est un processeur qui est utilisé dans l'industrie aérospatiale européenne et des applications militaires. Ce processeur utilise le jeu d'instruction du SPARC (*Scalable Processor Architecture*) V8 et existe en version FT (*Fault Tolerence*). Dans ce stage, nous avons d'abord essayé de coupler un coprocesseur sur le mode d'interfaçage d'une FPU (*Floating Point Unit*) : approche ASIP. Et puis, l'interface coprocesseur n'étant plus accessible dans le Leon version 3, nous avons décidé d'utiliser un Fifo reliée au bus APB.

1.2 Les objectifs

Cette étude a été découpée en plusieurs étapes :

1. Prise en main des Soc à base de Leon3 (GRLIB) et des outils associés (compilateur, Synthèse du SOC, simulation, moniteur temps-réel…)
2. Approche ASIP : interface coprocesseur (développement et test)
3. Approche « BUS based »: interface ABP2FSL et AHB2FSL (développement et test)

1.3 Plan

Le premier chapitre présente l'introduction générale sur le sujet du stage, l'objectif du stage, et le plan du rapport.

Nous présentons dans le chapitre 2 des notions importantes sur le processeur LEON : architecture du processeur et description du bus AMBA incluant l'AHB (*Advanced High-Performance Bus*) et l'APB (*Advanced Peripheral Bus*).

Ensuite, nous décrivons un couplage direct entre un processeur LEON et un coprocesseur. Il s'agit d'une approche ASIP : utilisation d'un jeu d'instruction spécifique pour l'envoi des opérandes et l'attente du résultat. Puis nous avons décidé d'introduire une Fifo à la manière d'un lien FSL du microblaze pour découpler l'exécution de processeur et du coprocesseur.

Cette présentation est suivie par le chapitre 3 consacré à l'environnement logiciel utilisé pour configurer et générer un SoC à base LEON3, simuler le Soc et puis enfin monitorer en temps réel une application.

Dans le chapitre 4, nous décrivons la conception et la validation de l'interface APB vers FSL.

Enfin, nous concluons ce stage en montrant dans quelle mesure les objectifs fixés ont été atteints et proposons un ensemble de perspectives à ces travaux.

CHAPITRE II

Description des LEON3, Amba bus, tightly coupled, FSL, Xilinx Virtex 6

2.1 LEON3

Le LEON3 est un modèle synthétisable VHDL d'un processeur 32 bits compatible avec l'architecture SPARC V8. Le modèle est hautement configurable, et particulièrement adapté pour un système-sur-une-puce (SoC/system on chip). Le code source complet est disponible sous la licence GNU GPL, permettant l'utilisation gratuite et illimitée pour la recherche et l'éducation.

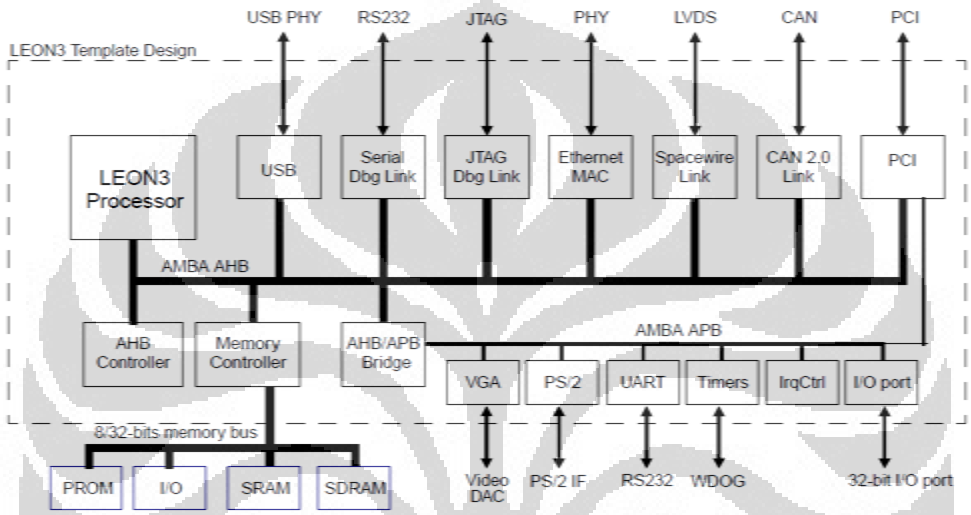


Figure 2.1: Architecture LEON 3 [(9)]

LEON3 est également disponible sous une licence commerciale à faible coût, ce qui lui permet d'être utilisé dans n'importe quelle application commerciale à une fraction du coût des cœurs d'IP comparables. Le processeur LEON3 possède les caractéristiques suivantes:

- ensemble d'instructions SPARC V8 avec des extensions V8e
- avancée **7-étage de pipeline**
- Unités se multiplient Matériel, divisé et MAC
- Hautes performances, entièrement en pipeline IEEE-754 FPU
- cache séparée d'instruction et les données (architecture Harvard) avec IGMP
- Les caches configurables: 1 - 4 voies, 1 - 256 Ko / trajet. Le remplacement aléatoire, LRR ou LRU
- l'instruction locale et les données scratch pad RAM, 1 - 512 Ko
- SPARC référence MMU (SRMMU) avec TLB configurable
- **AMBA AHB-2.0 d'interface de bus**

- Prise en charge avancée de débogage sur puce à l'instruction et le tampon des données de trace
- symétrique support multiprocesseurs (SMP)
- Power-down mode et le clock gating
- Conception robuste et entièrement synchrone d'horloge unique-bord
- Jusqu'à 125 MHz dans FPGA et 400 MHz sur 0,13 µm technologies ASIC
- Version à tolérance de pannes et SEU-preuve disponibles pour les applications spatiales
- Entièrement configurable
- Large gamme d'outils logiciels: compilateurs, noyaux, des simulateurs et des moniteurs de débogage
- Haute performance : 1.4 DMIPS/MHz, 1.8 CoreMark/MHz (gcc -4.1.2)

Le processeur LEON3 est distribué dans le cadre de la bibliothèque IP GRLIB, permettant une intégration simple dans la conception SOC complexes. GRLIB comprend également une configuration multiprocesseurs avec un maximum de 4 CPU et une large gamme de périphériques [(2)]

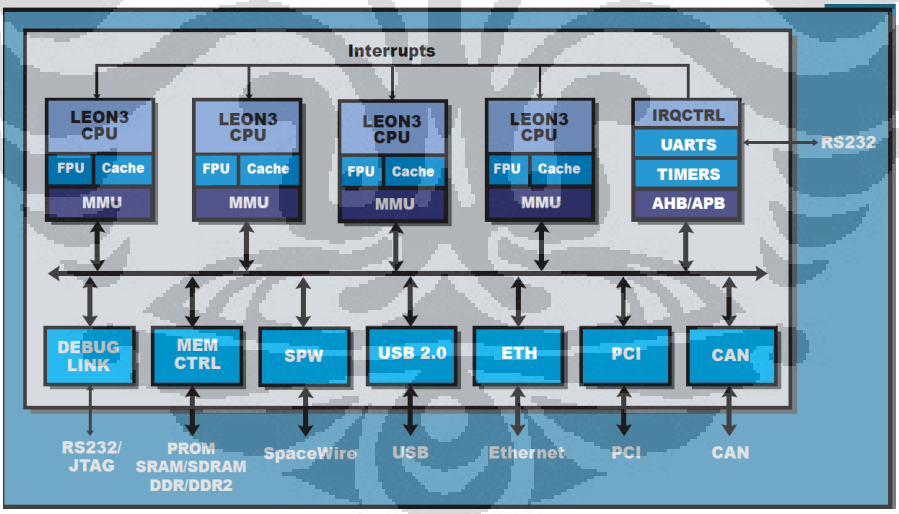


Figure 2.2: Conception complexe système multiprocesseur LEON3 [(25)]

Une description plus détaillée peut être trouvée dans [(2)].

SPARC est un processeur RISC. Une différence entre SPARC et le Berkeley RISC I & II, c'est que SPARC fournit une plus grande flexibilité à un compilateur dans sa mission de registres aux variables de programme. SPARC est plus souple car la gestion des fenêtres registre n'est pas liée à l'appel de procédure et revenir de procédure (CALL et JMPL)

Instructions, car il est sur les machines à Berkeley. Au lieu de cela, des instructions séparées (SAVE et RESTORE) fournir inscrire la gestion des fenêtres.

Un processeur SPARC comprend logiquement une unité de calcul entier (UI/integer unit), une unité à virgule flottante (FPU/ floating-point unit), et une option coprocesseur (CP/ coprocessor), chacune avec ses propres registres. Cette organisation permet aux implémentations avec la concurrence maximale entre le chiffre entier, en virgule flottante, et l'exécution d'instruction de coprocesseur. Tous les registres à l'exception des ceux des coprocesseurs ont 32 bits de largeur : Les opérandes de l'instruction sont généralement simples registres, paires de registres, ou quadruples registres. [(3)]

2.2 AMBA Bus

L'architecture avancée microcontrôleur Bus (AMBA/*Advanced Microcontroller Bus Architecture*) définit un standard de communications on chip pour la conception de haute performance microcontrôleurs embarqués.

Trois bus distincts sont définis dans la spécification AMBA, mais deux sont réellement utilisés :

1. L'avancé haute performance bus (AHB /*Advanced High-performance Bus*)
   L'AHB AMBA pour la haute-performance (BURST de données)

2. Le bus périphérique avancé (APB /*Advanced Peripheral Bus*)
   L'APB AMBA est à faible consommation périphériques. AMBA APB est optimisé pour consommation d'énergie minimale et la complexité d'interface réduite à l'appui des fonctions périphériques. Il repose sur un protocole d'échange synchrone simple à deux phases : envoi de l'adresse, puis envoi de la donnée (Pas de BURST).

Une méthodologie de test est fournie avec la spécification AMBA qui fournit une infrastructure pour le test macro cellule modulaire et l'accès au diagnostic. [(4)]

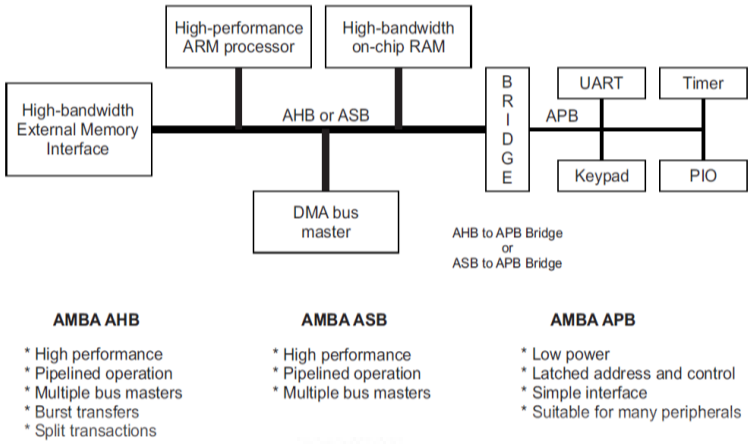


Figure 2.3: UN system standard AMBA Bus [(4)]

2.3 Tightly coupled

Il existe deux types de couplage dans un SOC : « tightly coupled » (couplage direct du type ASIP ou par Fifo ou par mémoire partagée) et « loosely coupled » (couplage indirectement via un bus et définition d'un mapping mémoire).

Une comparaison entre les principales caractéristiques d'une interface coprocesseur et une interface mappé en mémoire est donnée dans le tableau 2.1.

Dans un schéma du type **tightly couplage**, il y a un seul point de synchronisation (instruction spécifique) : envoie des opérandes et récupération du résultat.

Dans **le cas loosely coupled**, le logiciel fournit un grand bloc de données sur le coprocesseur, envoie le « start » puis attend le « done » avant de récupérer les résultats. Le bon choix entre couplage tightly et un couplage loosely dépend de l'application et l'architecture ciblée [(5)].

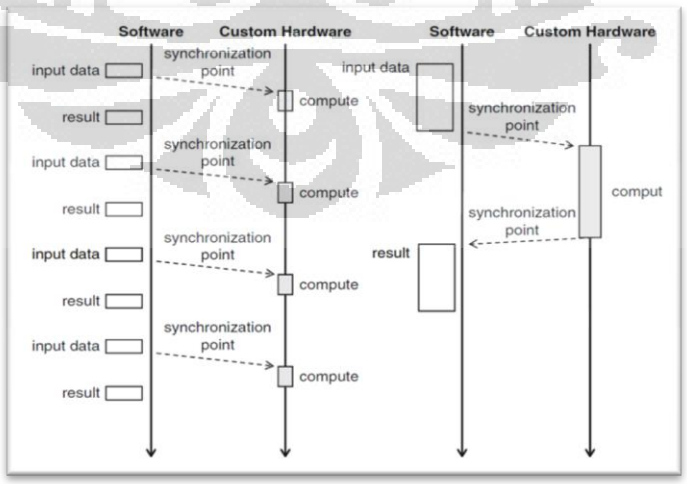


Figure 2.4: Tightly coupled et loosely coupled [(5)]

Tableau 2.1 : En comparant une interface coprocesseur avec une mappé en mémoire [(5)]

Factor	Coprocessor interface	Memory-mapped interface
Addressing	Processor-specific	On-chip bus address
Connection	Point-to-point	Shared
Latency	Fixed	Variable
Throughput	Higher	Lower
Typical-use	Tightly coupled	Loosely coupled

L'interface coprocesseur de Leon est décrite par le diagramme temporel de la figure 2.6. Le processeur indique la présence des opérandes au coprocesseur avec le signal load, puis il attend que le signal busy soit désactivé avant de continuer son exécution.

Nous avons d'abord pensé utiliser ce mode d'interfaçage pour coupler un accélérateur hardware en insérant un wrapper permettant de passer du protocole « interface copro ASIP » à un protocole en 4 phases sur niveaux pour assurer le changement de domaine de fréquence (« clock domain crossing »).

Le leon étant en attente de fin d'exécution, nous avons ensuite décidé d'ajouter une Fifo du type FSL permettant d'introduire du parallélisme d'exécution entre le leon et le coprocesseur et d'assurer le changement du domaine de fréquence (Fifo bi-clock).

Malheureusement, durant la phase d'intégration et de validation, nous nous sommes aperçu que l'interface coprocesseur n'est plus aisément accessible depuis la version 3 du Leon.

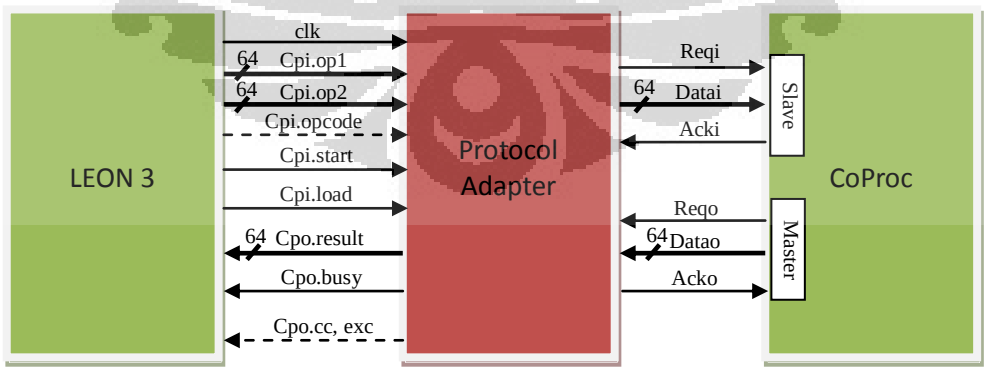


Figure 2.5: Tightly-coupled coprocessor Interface

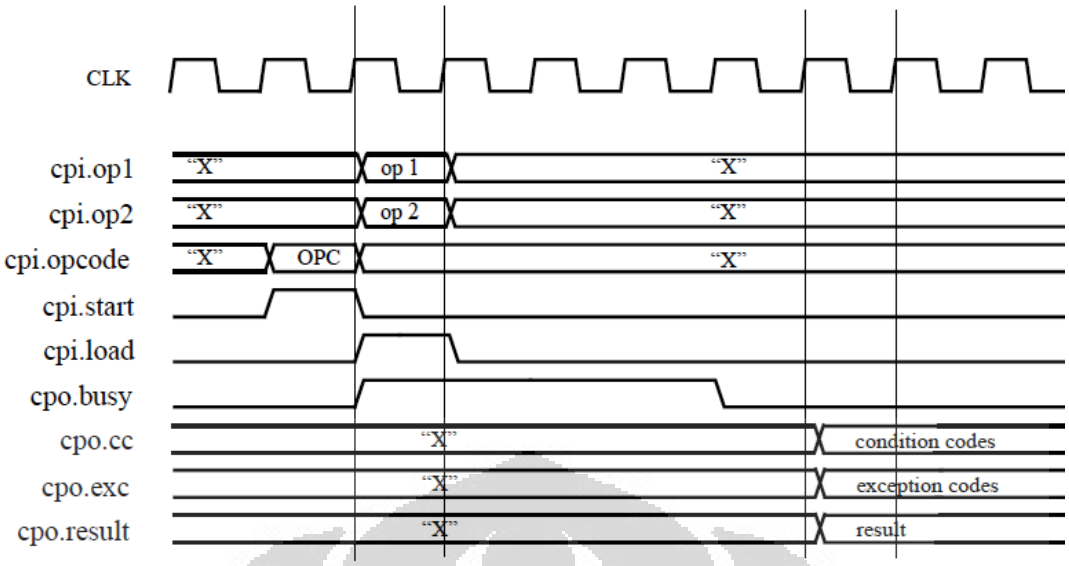


Figure 2.6: Exécution schéma unité de forme d'onde coproc [(14)]

2.4 FSL

La figure 2.7 montre le principe d »un Fifo FSL et les signaux disponibles.

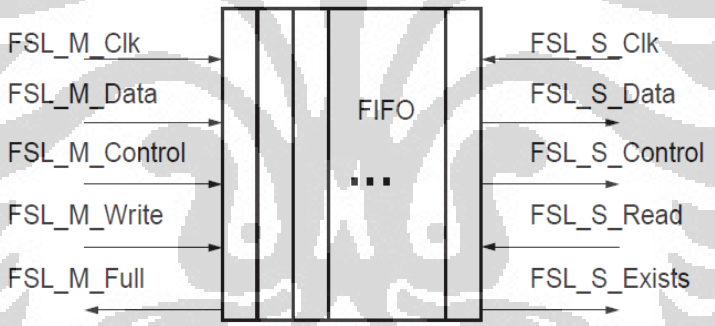


Figure 2.7: Interface FSL

L'interface est type maître pour l'écriture dans la Fifo et esclave pour la lecture. [(6)]

Dans cette étude, nous avons utilisé la FSL Version 2.0 de Ise version 13(C:\Xilinx\13.1\ISE_DS\EDK\hw\XilinxProcessorIPLib\pcores\fsl_v20_v2_11_d\hdl\vhdl) [(7)].

2.5 La Carte Xilinx Virtex ML-605

Toute la conception ci-dessus est mise en œuvre dans la carte Xilinx virtex 6 ML605. Des renseignements supplémentaires et un appui matériel est situé à: http://www.xilinx.com/ml605**[(8)].**

CHAPITRE III

GRLIB et GRMON

3.1 GRLIB

La Bibliothèque IP GRLIB est un ensemble intégré de cœurs IP réutilisables, conçu pour le système sur puce (SOC) de développement centrée autour d'une interface commune par bus. Une approche du type plug & play est utilisée pour configurer et de connecter les noyaux de propriété intellectuelle, sans la nécessité de modifier les sources. [(9)]

La figure ci-dessous montre un exemple d'un système leon3 conçu avec GRLIB et j'ai ajouté Couplage copro dans cette expérience :

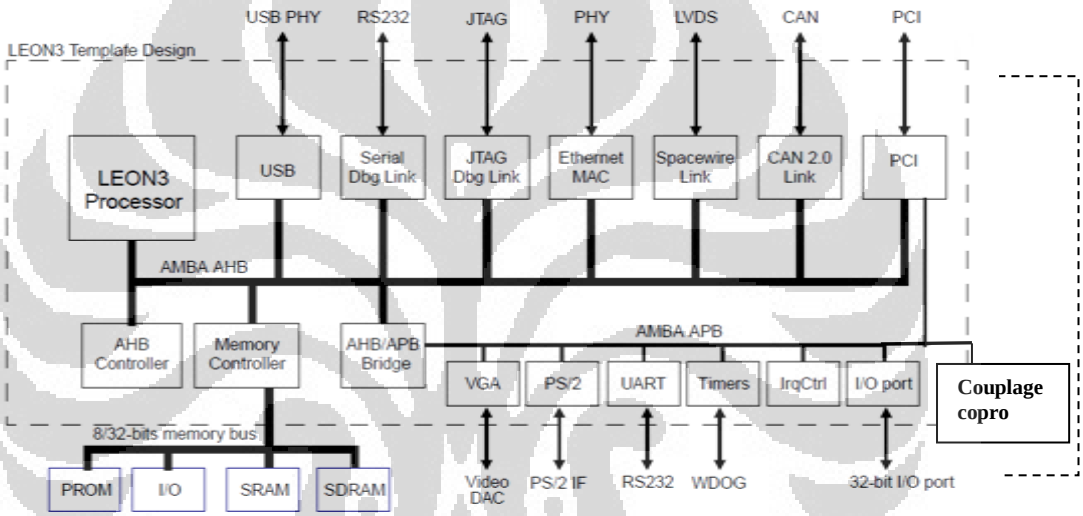


Figure 3.1 Leon3 architecture grlib extending copro couplage [(9)]

3.1.1. Configure GRLIB SoC LEON3

Dans la conception de SoC, il y a quatre étapes importantes: la configuration, la vérification, la synthèse puis l'implémentation.

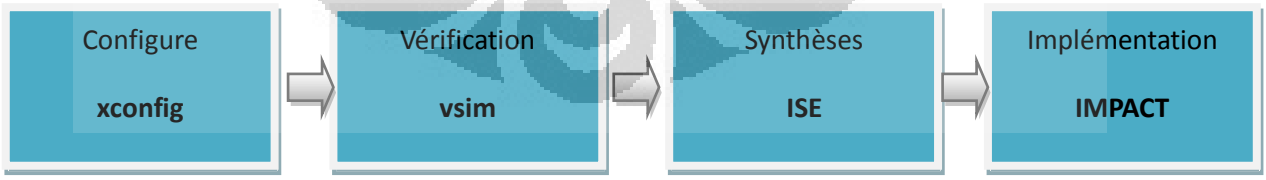


Figure 3.2: Les étapes design SoC LEON3

Pour une connaissance de base d'un soc leon et de sa configuration, le fichier README.txt du dossier grlib-gpl-1.1.0-b4113\designs\leon3-xilinx-ml605 doit être consulté. Ne pas oublier de changer le nom du dossier, pour éviter les erreurs de configuration. Ceci permet donc de restaurer revenir à la configuration par défaut quand une erreur survient.

Lorsqu'on utilise le **FPU** dans la conception, Cette conception exige une **netlist** pour la connexion entre UI (Unit Integer) et la FPU (floating Point Unit). Il faut télécharger la

netlist depuis http://www.gaisler.com/cms/index.php (download => grlib => netlist for Xilinx and altera).

Dans ce chapitre, j'ai utilisé le fichier makefile pour config. makefile pour un programme global dans le dossier ~/bin. En puis pour chaque élément du soc, il y a aussi un makefile pour faire un paramètre spécial.

Les étapes suivantes sont utilisées pour configurer l'interface graphique basée sur SoC leon3. Si vous travaillez dans Windows, alors utiliser cygwin. Exécutez la commande **$make xconfig**. Si vous pensez que votre cygwin ne fonctionne pas bien à cause de l'interface graphique, alors utiliser **$startx**, vous travaillez dans le cygwin/x.

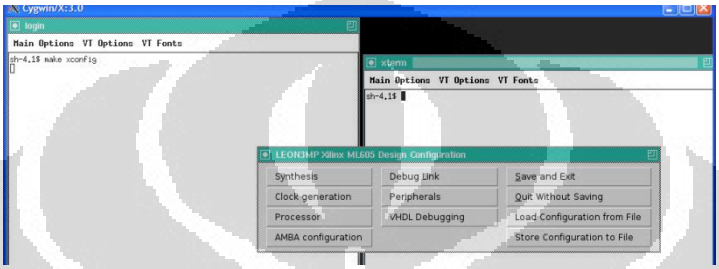


Figure 3.3: Configure SoC leon avec grlib dengan startx

- **Synthesys**

  Configuration pour la synthèse : choix de la carte et de la cible fpga.

- **Clock generation**

  Sélection de l'horloge du processeur et du bus (75MHz).

- **Processor**

  Configuration des processeurs (permet jusqu'à 4 cores) et de la FPU : GRFPU, GRFPU-LITE et MEIKO FPU. Dans cette expérience, nous avons utilisé un core avec une FPU GRFPU.

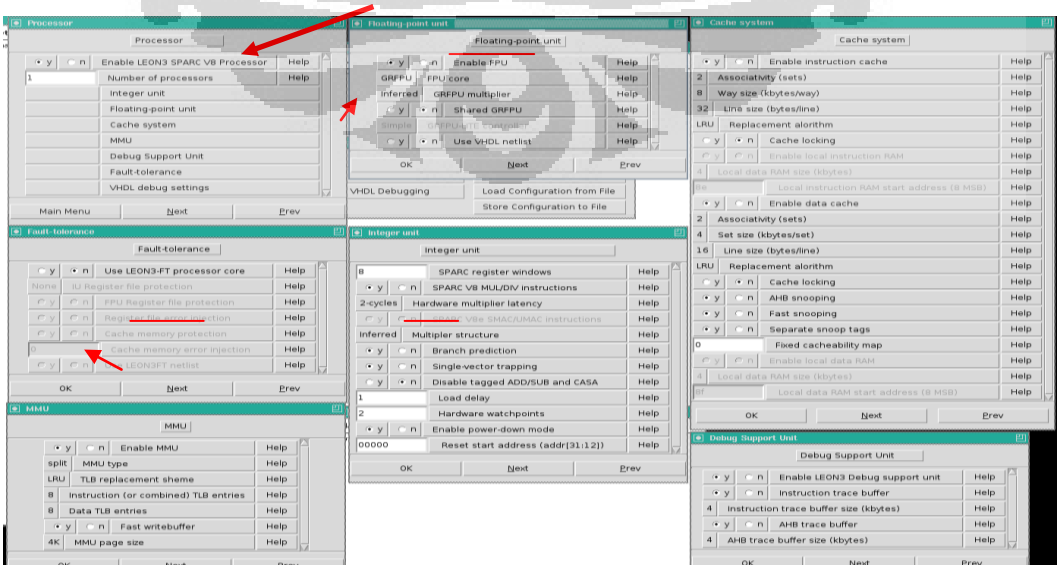


Figure 3.4: Setting grlib processor LEON3

- **Amba configuration**

  Paramètres de configuration AMBA pour création de l'adresse de surface E/S et le pont de AHB / APB. Et pouvons-nous ajouter master AHB, mais pour la haute performance du processeur permettre à la valeur 0 (pour utiliser un master AHB). Dans ce paramètre nous pouvons aussi activer ronde Robbin arbitre pour la gestion des flux de paquets.

- **debug link**

  Dans ce paramètre nous nous sommes concentrés sur le lien qui est utilisé pour déboguer le système de traçage grmon l'utilisation. J'utilise debug link et serial debug link.

- **Peripherals**

  Paramètre périphérique que nous allons développer. Dans ce cas, je l'ai laissé par défaut.

- **vhdl debugging**

  J'ai activé une UART pour le traçage.

Enfin, cliquez sur sauvegarder et quitter. Ensuite nous modifions le fichier config.vhd. Pour certains paramètres, vous pouvez éditer le fichier manuellement dans config.vhd en fournissant constante et la référence au moment de la conception dans topmodel (leon3mp.vhd) est faite pour l'interface plug and play IP.

3.1.2. Vérification avec Vsim

Après avoir configuré le soc leon3, on peut passer à une première étape de vérification de la façon suivante:

1. make distclean
2. make vsim
3. make vsim-launch ou vsim

Après lancement de la simulation, se déroulera la configuration de démarrage qui a été enregistré dans une PROM (boot loader).

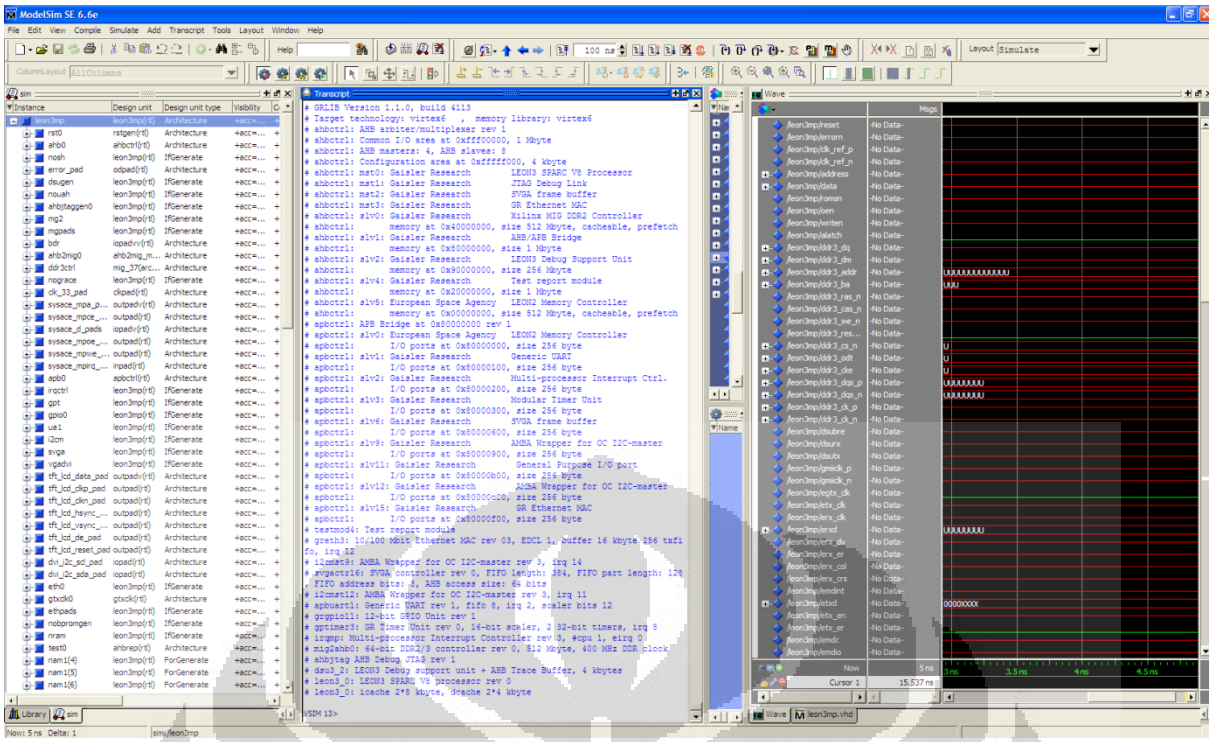


Figure 3.5: Simulation ModelSim avec la PROM d'initialisation

3.1.3. Ajouté library dans Design SoC

Si nous utilisons une nouvelle bibliothèque à partir de laquelle nous ajoutons nos IP, nous devons définir le dossier /bin/libs.txt. Pour cela, il faut remplir le fichier avec le nom du dossier bibliothèque ajouté, comme le montre ci-dessous:

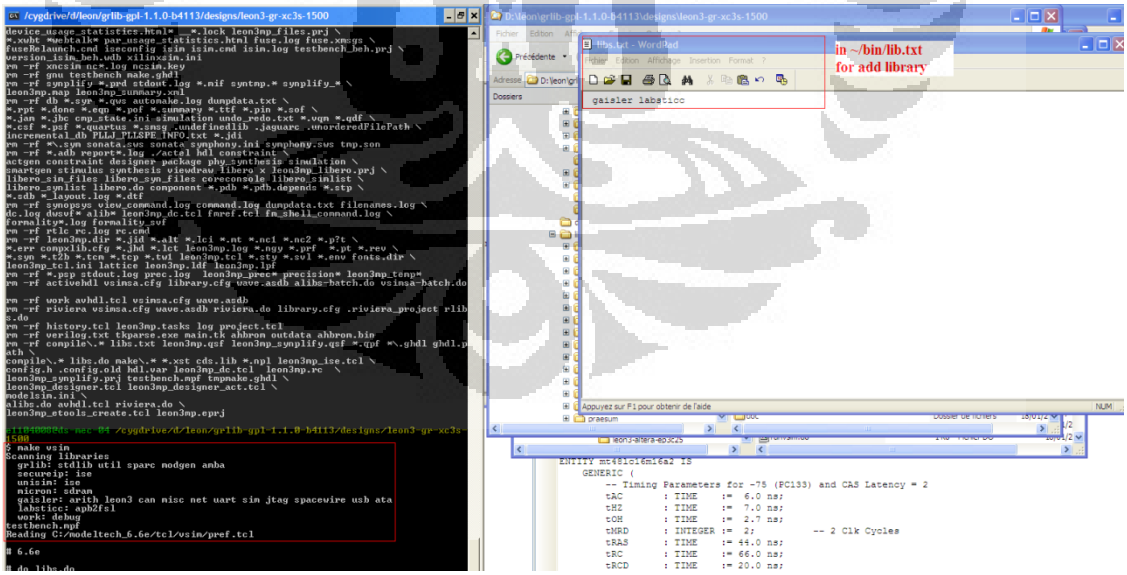


Figure 3.6: Ajout d'une Librairie dans design SoC LEON3

Dans l'exemple ci-dessus, j'ai ajouté une bibliothèque gaisler et labsticc (design coupled. Il faut aussi, ajouter le chemin du dossier de la bibliothèque (/lib/gaisler et lib/labstic) dans les fichiers **dir.txt et vhdlsync.txt**.

3.1.4. Setting code C pour vérification

Le premier code de test écrit est pour tester la FPU. Le code C pour les réglages du système sont dans ~/grlib-gpl-1.1.0-b4113/software/leon3 ensuite nous établissons sur la base de la configuration que nous réveillons avec quelques paramètres. Ceci est décrit dans la documentation de GRMON.

Le test peut aussi être réalisé par simulation ModelSim, en générant le contenu d'une rom (boot loader : prom.s) et d'une ram (application : sdram.srec) [(15)]. Nous utilisons la commande **$ make soft**. **prom.srec**, un fichier s-record pour le démarrage d'un soc et **sdram.srec** est s-record fichier pour tester l'application. Tous les fichiers sont configurés à l'aide de la commande $make soft. Cette simulation ne peut pas être exécutée si vous utilisez grfpu-lite et SpaceWire [(17)] qui ne sont disponibles que sous forme de netlist.

Les paramètres suivants du tesbench permettent pour charger prom.srec et sdram.srec.

```
constant promfile  : string  := "prom.srec";       -- rom contents
constant sdramfile : string  := "sdram.srec";      -- sdram contents
……..
address(0) <= '0';
  prom0 : for i in 0 to 1 generate
    sr0 : sram generic map (index => i+4, abits => 24, fname => promfile)
      port map (address(24 downto 1), data(15-i*8 downto 8-i*8), romsn,
            writen, oen);
 end generate;
```

tesbench.vhd

Comme indiqué, dans le fichier readme.txt du dossier design/leon3-xilinx-ml605, le modèle de simulation de la mémoire et son contrôleur mémoire est loin d'être parfait (carte ML605 encore mal supportée dans le grlib) mais on peut aussi charger le programme de l'application dans une ROM comme indiqué dans le projet gnss_sensor(https://github.com/teeshina/soc_leon3) [(16)].

3.1.5. Synthèse GRLIB dans Board

Ensuite, nous utilisons **$make mig** pour la génération d'un modèle de la mémoire DDR. Des erreurs sur le commande **$make mig** se produisent si il ya un conflit d'adresse mémoire : deux activation ou AHB adresses de mémoire AHB simultanément. Il faut donc être attentif au mapping mémoire.

Et puis la synthèse logique (génération du bitstream) se lance par la commande **$make ise** [(9)]. Ensuite, nous utilisons **$make ise-prog-fpga** pour entrer le bitstream généré dans le FPGA (dans ce cas la mémoire flash).

3.2 GRMON

Pour tracer l'exécution des programmes à distance sur le carte, nous utilisons GRMON (Gaisler Resaearch MONitor). [(10)]

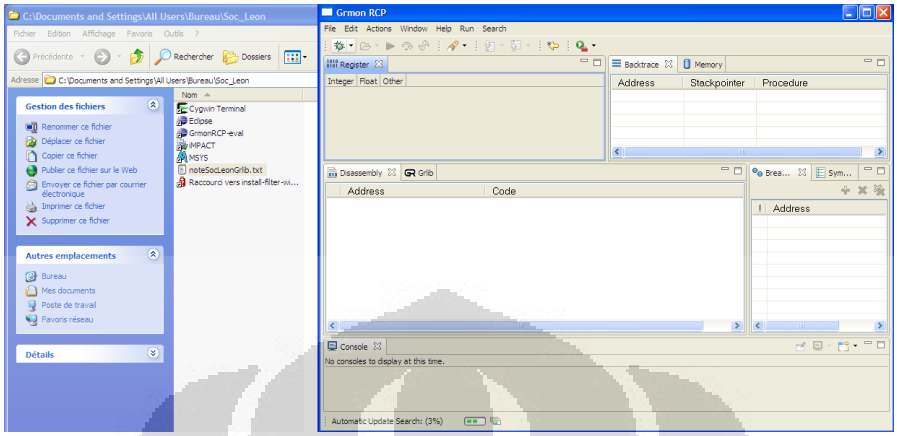


Figure 3.7: Gaisler Research MONitor

3.1.1. Configuration GRMON et GRLIDE

Le moniteur GRMON peut-être utilisé de deux façons:

1. Utilisation directe (connection carte par jtag, série ou Ethernet)
2. Utilisation via l'environnement de programmation GRLIDE (Eclipse IDE)

Pour l'utilisation directe, nous exécutons l'application grmon et nous nous connectons à la carte via le lien jtag/usb.

Dans cette expérience, j'ai utilisé du remote debugging via JTAG et fait aussi une trace à l'aide de la sortie de série. Nous pouvons également utiliser la commande dans cygwin $grmon-rcp.exe –xlusb –l. Des problèmes ont été rencontrés entre le driver de la carte Ml605 et le driver l'USB JTAG de la GRLIB. Vous avez donc du installer un filtre particulier sur l'usb pour résoudre ce problème.

Quelques commandes du monietur grmon:

1. info sys
2. info reg
3. info libs
4. load <file bit>
5. run dan reset
6. mem <addres>
7. dis <addres/main>
8. cpu en <no_cpu - 1>
9. cpu
10. float
11. cp
12. register
13. inst <length_trace>
14. icache dan dcache
15. perf <en/dis> **[(10)]**

3.1.2. Exécuter et Trace du code C avec GRMON

Le minoteur GRMOn permet de charger et éxecuter un programme, dans les formats binaire *.exe, *.elf, *.o .

Si nous utilisons, l'environnement de programmation grlide les étapes sont:

- compiler notre code C (projet-> run-> compiler-all).
- configurer et lancer le debug.

CHAPITRE IV

Design et résultats tightly-coupled co-processor FSL

Ce chapitre décrit la conception et la mise en œuvre d'un co-processeur couplé au processeur LEON3 en utilisant l'interface co-processeur des Leons puis une fifo FSL (*Fast Simplex Link)* reliée au bus AMBA APB.

4.1 Design coupled coprocessor

Le sparc V8 et les leons 1 et 2 permettent de coupler directement un coprocesseur et dispose pour cela d'une interface et d'instructions spécifiques (approche ASIP).

Instrucsions set SPARC V8 coprocessor:

cpop1 *opc*, *cregrs1* , *cregrs2* , *cregrd*

cpop2 *opc*, *cregrs1* , *cregrs2* , *cregrd*

Setting register PSR (**Processor State Register)** pour enable copro [(4)]**:**

Impl	*ver*	*icc*	Reserved	EC	EF	PIL	S	PS	ET	CWP

PSR_enable_coprocessor (CE) bit 13 doit être définie à 1 pour activer le coprocesseur, ou utiliser pser.ce = 1. Pour ce réglage peut être fait en donnant force au paramètre registre vhdl cp = 1 en configure leon3ft, leon3s, leon3sh, leon3ftsh. En changeant les valeurs de 0 entre CFG_V8 et CFG_MAC avec CFG_CP. Puis dans config.vhd en ajoutant un "*constant CFG_CP : integer := 1; -- enable cp".*

si vous utilisez le code c dans le réglage initial:

```
tmp = xgetpsr();
setpsr(tmp | (1 << 12) | (1 << 13));
tmp = xgetpsr();
if (!(tmp & (1 <<12))) return(0);
set_fsr(0);
```

Dans un premier temps, nous avons utilisé un wrapper interface co-processeur/protocole en 4 phases (Request/Acknowledge) comme le mettre la figure 4.1. Dans cette configuration, le leon est bloqué jusqu'à la désactivation du signal busy.

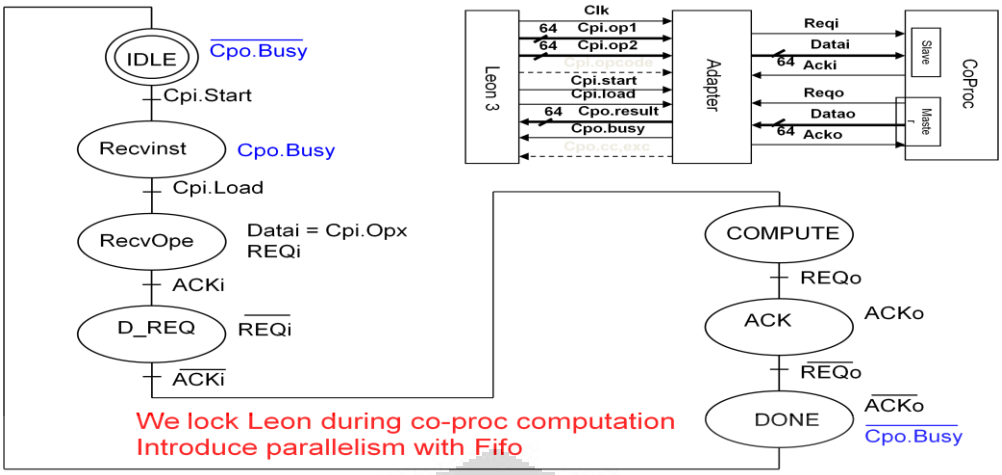


Figure 4.1: finite state machine tigtly-coupled copro.

Pour introduire du parallélisme d'exécution entre le Leon et le coprocesseur, nous avons ensuite insérer deux fifos du type FSL comme le montre la figure 4.2

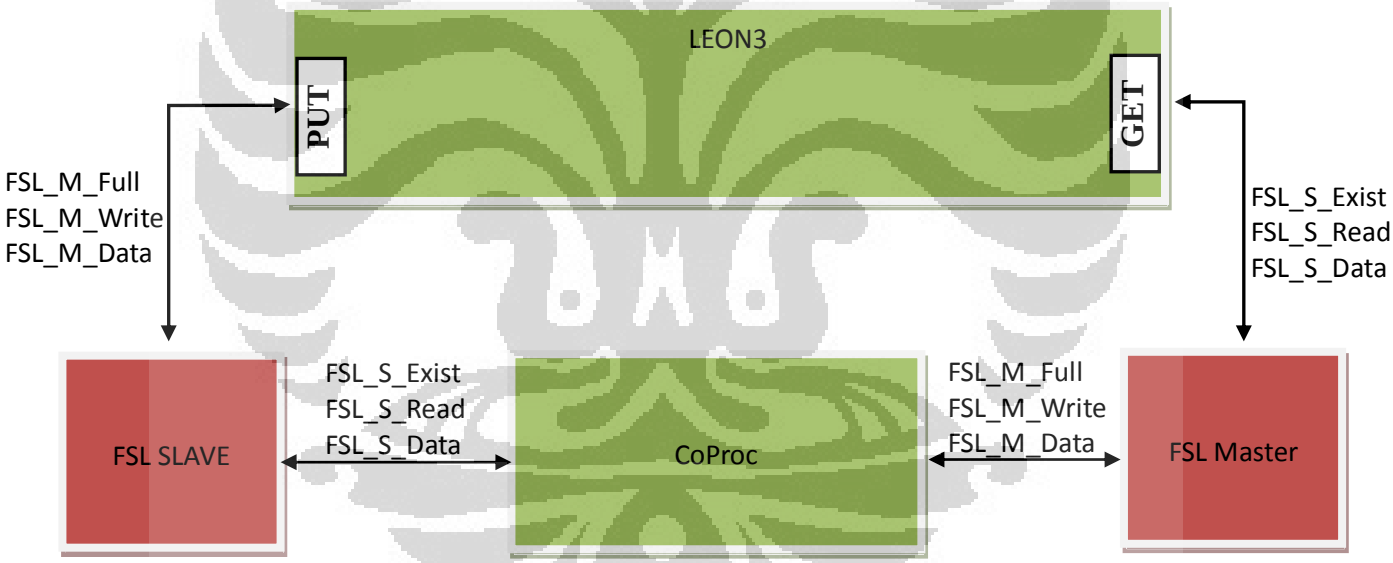


Figure 4.2: tightly-coupled FSL

FSL est une Fifo bi-clock (Master Clock, Slave Clock) et permet donc de coupler des éléments asynchrones. Les FSL sont directement couplées au processeur de synchronisation de coprocesseur qui bloque le Datapath (enable signal) si les entrées ou sorties ne sont pas prêts à être consommait / produit (FIFO empty / full) [(13)].

4.2 Design couplage copro FSL avec APB

Comme malheureusement, l'interface coprocesseur des Leons semble avoir été cachée dans le Leon3, nous nous sommes rabattu vers le bus APB en développant des wrappers ABP2ReqAck (figure 4.1) et APB2FSL.

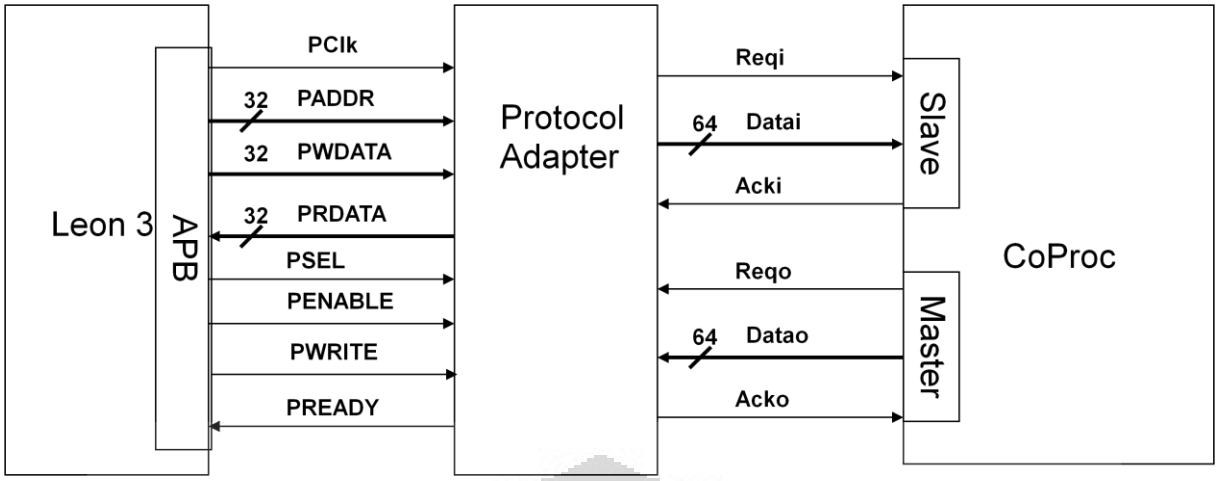


Figure 4.3: protocol adapter APB to FSL

La Figure 4.4, montre la simplicité du processus de transfert pour écrire et lire APB.

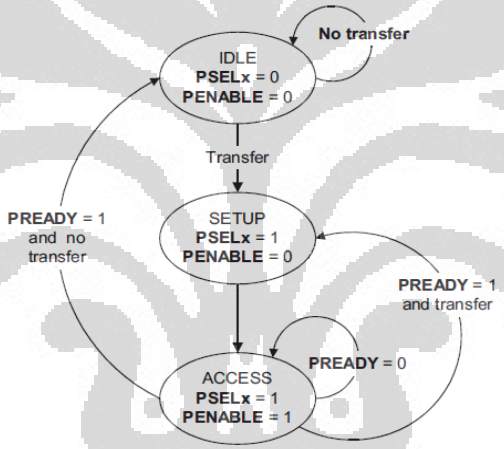


Figure 4.4 : finite state machine apb [(18)]

Étapes de transfert (lecture / écriture) des données en utilisant l'APB:

1. IDLE
   - défaut APB Etat
2. SETUP
   - Lorsque le transfert nécessaire
   - PSELx est affirmé
   - Un seul cycle
3. ACCESS
   - PENABLE est affirmé
   - Addr, write, select, et Write des données restent stables
   - Restez si PREADY = L

- Aller à IDLE si PREADY = H et aucune donnée ne
- Aller à SETUP si PREADY = H et plus de données en attente

La figure 4.5 présente un diagramme de temps de transfert de données à partir de l'APB et FSL. Le figure 4.5.a et 4.5.c du signal apb et le figure 4.5.b et 4.5.d signal de FSL.

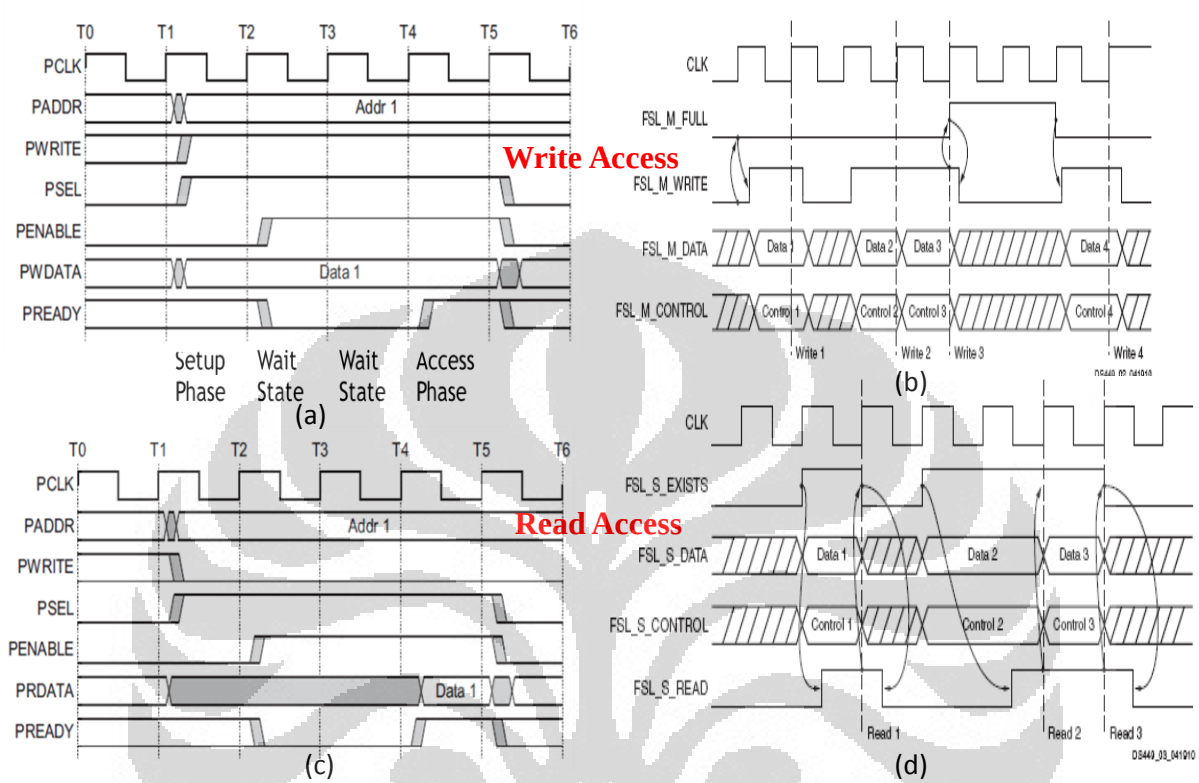


figure 4.5: APB et FSL signal [(18)]

La conversion APB vers FSL, conçue sur la base des chronogrammes (figure 4.5) du bus apb et FSL est représentée dans le diagramme de machine d'état fini (FSM), comme illustré dans la machine de moore de la figure 4.6.

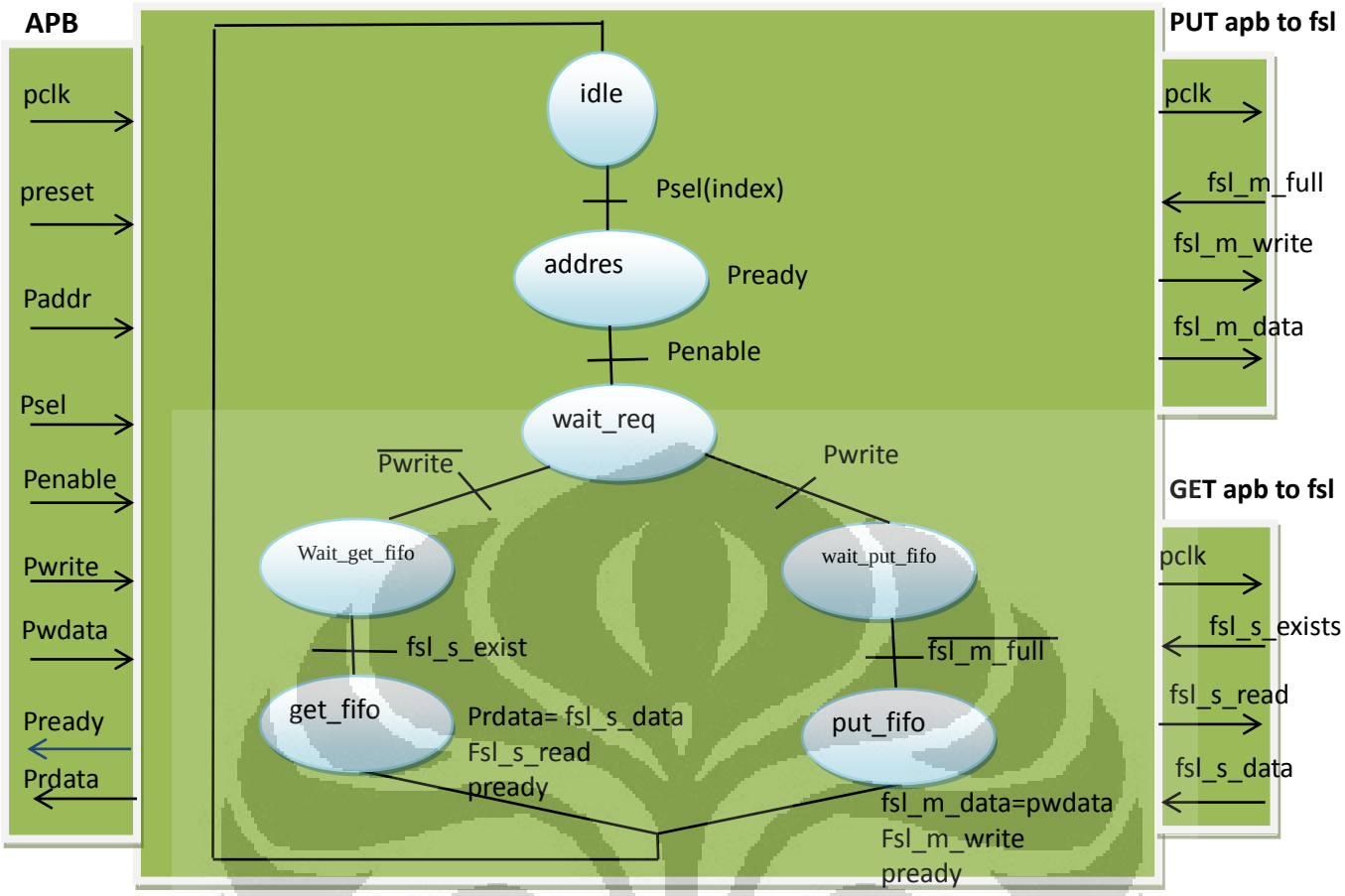


Figure 4.6: FSM APB to FSL protocol adapter

4.3 Mise en œuvre apb2fsl

Les bibliothèques et composants qui sont ajoutés doivent être définies dans le fichiers lib.txt, et dirs.txt et dans dossier ~/lib/ip_ajouter..

La mise en œuvre de l'adaptateur de protocole dans leon3 conception de SoC utilisant le principe de plug & pluy, par la définition de notre lib dans le paquet et le vendor ID et le Device ID id quand trace à l'aide GrMon. Configure dans grlib/amba/device.vhd. Ajouter device ID et le vendor ID qui est créé et fournir une du device.

```
-- Vendor codes
constant VENDOR_GAISLER    : amba_vendor_type := 16#01#;
constant GAISLER_APB2FSL   : amba_device_type := 16#120#;
constant GAISLER_DESC : vendor_description :=  "Gaisler Research        ";
constant gaisler_device_table : device_table_type := (
GAISLER_APB2FSL   => "coupled coprosessor FSL       ",
……..
others            => "Unknown Device                ");

constant gaisler_lib : vendor_library_type := (
    vendorid       => VENDOR_GAISLER,
    vendordesc     => GAISLER_DESC,
    device_table   => gaisler_device_table  );                    devices.vhd
```

Définition de l'identifiant de vendor et device id de l'apbtofsl protocole:

```
constant pconfig : apb_config_type := (
0 => ahb_device_reg (VENDOR_GAISLER, GAISLER_APB2FSL, 0, REVISION, 1),
 1 => apb_iobar(paddr, pmask));
```

apb2fsl.vhd

Le code apbtofsl.vhd est fourni dans l'annexe B.1. Nous avons défini des types pour les types de bus et le composant wrapper :

```
-- FSL conection ----------
   type apb2fsl_in_type is record
-- GET_FSL  input
     FSL_S_Read    : std_logic;
     FSL_S_Exists  : std_logic;
     FSL_S_Data    : std_logic_vector(0 to 31);
-- Put input
     FSL_M_Full    : std_logic;
     FSL_M_Control : std_logic;
   end record;
   type apb2fsl_out_type is record
-- PUT_FSL output
     FSL_M_Data    : std_logic_vector(0 to 31);
     FSL_M_Write   : std_logic;
     FSL_S_Clk     : std_logic;
-- get output
     FSL_M_Clk     : std_logic;
     FSL_S_Control : std_logic;
   end record;
component apb2fsl
   generic (
      --    APB signal
     pindex   : integer := 0;
      paddr    : integer := 0;
      pmask    : integer := 16#fff#
            );
   port (
     --  APB signal
     rst    : in  std_ulogic;
     clk    : in  std_ulogic;
     apbi   : in  apb_slv_in_type;
     apbo   : out apb_slv_out_type;
    apb2fsli : in std_logic_vector(0 to 63);
    apb2fslo : out std_logic_vector(0 to 63)
     );
end component;
```

misc.vhd

Pour ajouter ce wrapper dans l'architecture du SoC, il faut modifier le fichier ~ / design/leon3-xilinx-ml605/config.vhd:

```
-- APB to FSL interface
      constant CFG_APB2FSL_ENABLE : integer := 1;
-- FSL converter
      constant CFG_FSLCONV_ENABLE : integer := 1;
```

config.vhd

Voici les paramètres dans haut niveau:

```
-- apb2fsl
 signal apb2fsli : apb2fsl_in_type;
 signal apb2fslo : apb2fsl_out_type;
….....................................................................
--- APB FSL converter (PUT_FSL and GET_FSL)
-----------------------------------------------------------------------
  apbtofsl : if CFG_APB2FSL_ENABLE = 1 generate
 apb2fsl0 : apb2fsl generic map (pindex => 10, paddr => 10)
       port map (rstn, clkm, apbi => apbi, apbo => apbo(10), apb2fsli => apb2fsli,
apb2fslo => apb2fslo);
  end generate;
```

leon3mp.vhd

4.2 Configure IP FSL_V20

Pour la fifo FSL, nous avons utilisé les sources de Xilinx :

C:\Xilinx\13.1\ISE_DS\EDK\hw\XilinxProcessorIPLib\pcores\fsl_v20_v2_11_d\hdl\vhdl.

Le composant fsl est déclaré et instantié dans l'entité top du Soc :

```
component fsl_v20
 generic (
  C_EXT_RESET_HIGH     : integer := 1;
  C_ASYNC_CLKS         : integer := 0;
  C_IMPL_STYLE         : integer := 0;
  C_USE_CONTROL        : integer := 1;
  C_FSL_DWIDTH         : integer := 32;
  C_FSL_DEPTH          : integer := 16;
  C_READ_CLOCK_PERIOD : integer := 0
  );
 port (
  -- Clock and reset signals
  FSL_Clk : in  std_logic;
  SYS_Rst : in  std_logic;
  FSL_Rst : out std_logic;
  -- FSL master signals
  FSL_M_Clk     : in  std_logic;
  FSL_M_Data    : in  std_logic_vector(0 to C_FSL_DWIDTH-1);
  FSL_M_Control : in  std_logic;
  FSL_M_Write   : in  std_logic;
  FSL_M_Full    : out std_logic;
  -- FSL slave signals
```

```
    FSL_S_Clk     : in  std_logic;
    FSL_S_Data    : out std_logic_vector(0 to C_FSL_DWIDTH-1);
    FSL_S_Control : out std_logic;
    FSL_S_Read    : in  std_logic;
    FSL_S_Exists  : out std_logic;
    -- FIFO status signals
    FSL_Full      : out std_logic;
    FSL_Has_Data  : out std_logic;
    FSL_Control_IRQ : out std_logic
    );
end component;
```

misc.vhd

Au top niveau, la configuration est basée sur le couplage entre les composantes **PUT (apb2fslo)** et **FSL_SLAVE** et **GET (apb2fsli)** avec **FSL_MASTER**.

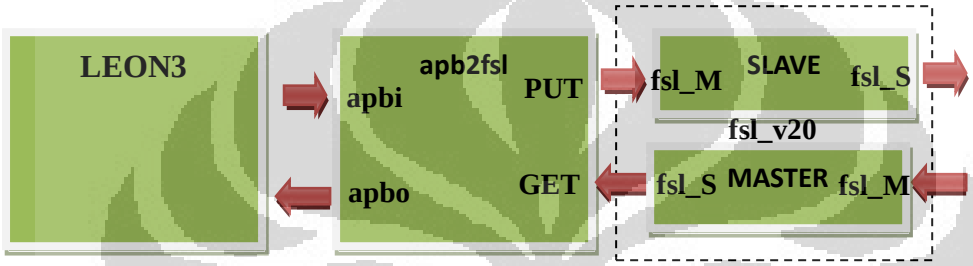


Figure 4.7: Configure toplevel apb2fsl

Figure 4.8 une configuration un top model pour apb2fsl avec fsl_V20. (Annexe B.2)

```
-----------------------------------------------------------------
---fsl converter top level
-----------------------------------------------------------------
fsl_converter : if CFG_FSLCONV_ENABLE = 1 generate
 fsl0 : fsl_v20 generic map (C_EXT_RESET_HIGH => 1, C_ASYNC_CLKS => 0,
C_IMPL_STYLE => 0, C_USE_CONTROL => 1, C_FSL_DWIDTH => 32,
C_FSL_DEPTH => 1, C_READ_CLOCK_PERIOD => 0)
  port map (clkm, rstn, FSL_M_Clk => apb2fslo.FSL_M_Clk , FSL_M_Data =>
apb2fslo.FSL_M_Data, FSL_M_Control => '0' , FSL_M_Write => apb2fslo.FSL_M_Write,
FSL_M_Full => apb2fsli.FSL_M_Full, FSL_S_Clk => apb2fslo.FSL_S_Clk , FSL_S_Data
=> apb2fsli.FSL_S_Data, FSL_S_Control => open, FSL_S_Read =>
apb2fslo.FSL_S_Read, FSL_S_Exists => apb2fsli.FSL_S_Exists);
  end generate;
```

leon3mp.vhd

4.4 Vérification apbtofsl

4.4.1 Testbench Signal SoC dans Modelsim

Le testbench du sytème complet (Soc) est fourni dans l'annexe B.3. Pour tester le wrapper, nous avons utilisé simplement une fifo comme le montre la figure 4.9.

Le résultat est montré par la figure 4.10.

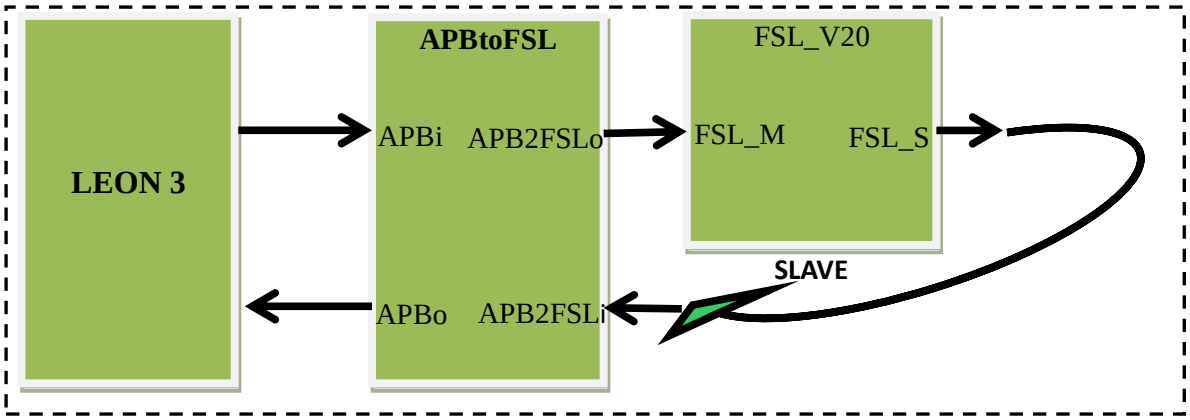


Figure 4.8: Conception tesbench system

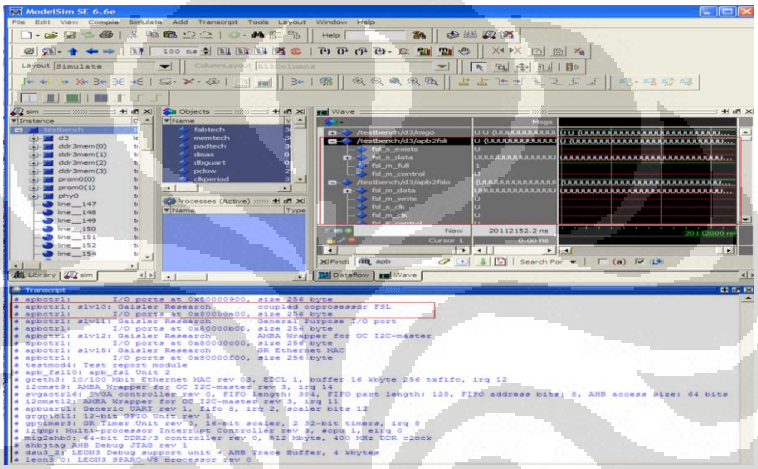


Figure 4.9 : Les résultats de tests de signaux SoC

4.4.2 Testbench Seul dans Modelsim

Comme le résultat précédent, n'était pas probant, nous avons développé un testbench pour valider le wrapper seul. (tesbench_standalone.vhd dans annexe B.4). Les signaux lecture et d'écriture apbi et apbo utilisent les procédures apbwrite() et apbread() définies dans amba_tp.vhd dans lib/grlib/amba.

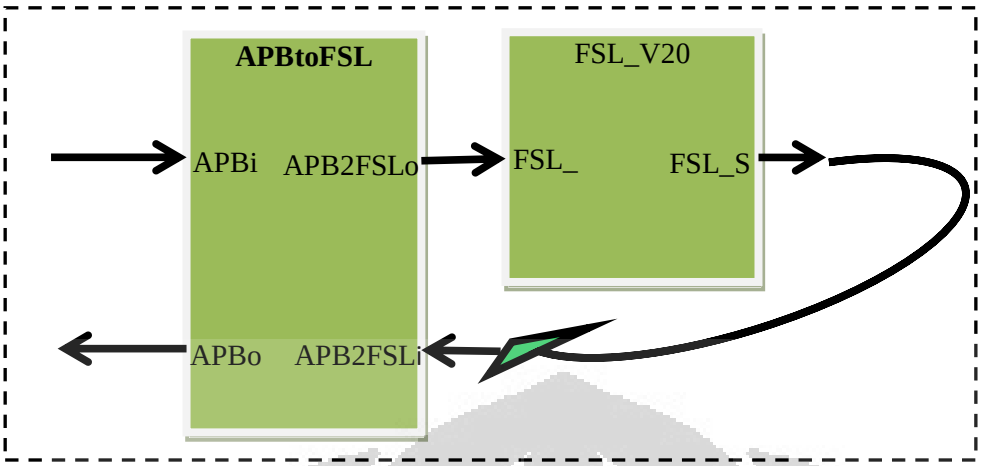


Figure 4.10: Conception testbench seul

Le développement de ce testbench a mis en évidence que le Soc à base de Leon3 utilise une vieille version du bus APT (version 2) qui ne possède pas le signal d'asservissement PReady indispensable pour notre couplage.

En suivant les résultats du test standalone:

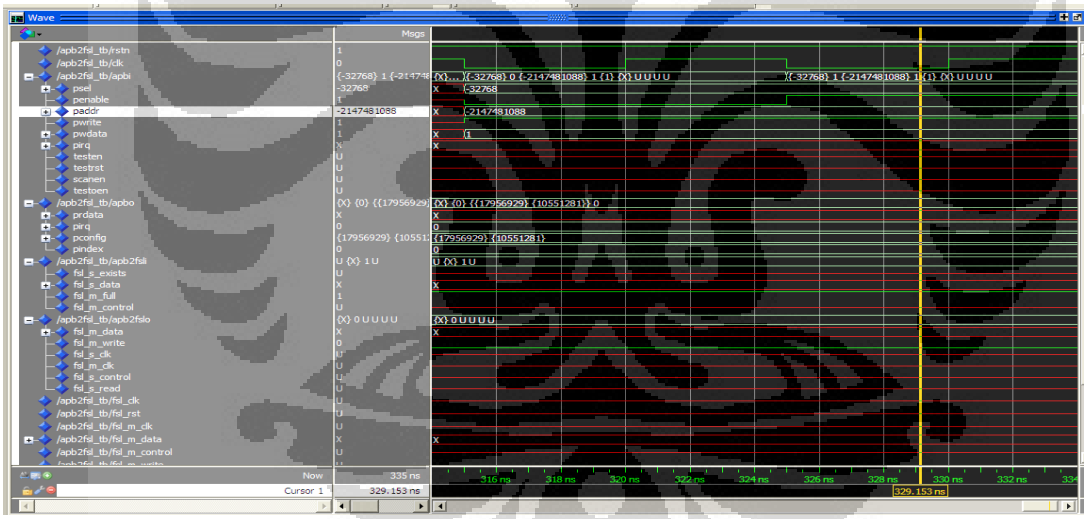


Figure 4.11: Test Seul Tesbench

4.5 Synthèse le dans FPGA

Bien que le test fonctionnel ne soit pas correct à cause de l'absence du signal PReady, j'ai fait la synthèse, la génération du bitstream à partir ISE et son téléchargement sur la carte.

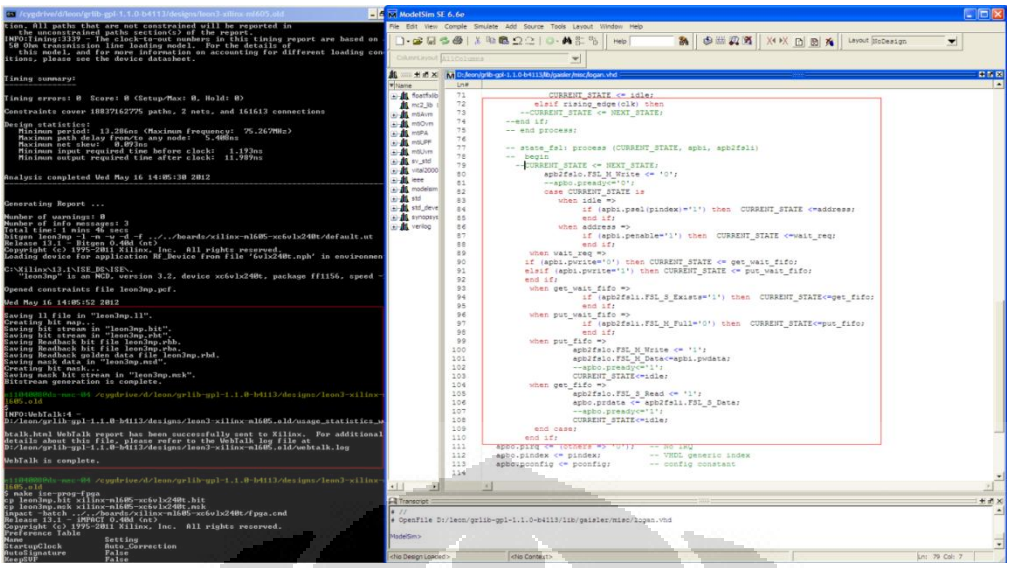


Figure 4.12: Synthése SoC LEON3 avec apbtofsl

4.6 Trace avec GRMON

En utilisant les commandes du moniteur GRMON, on peut voir la présence du wrapper dans le soc:

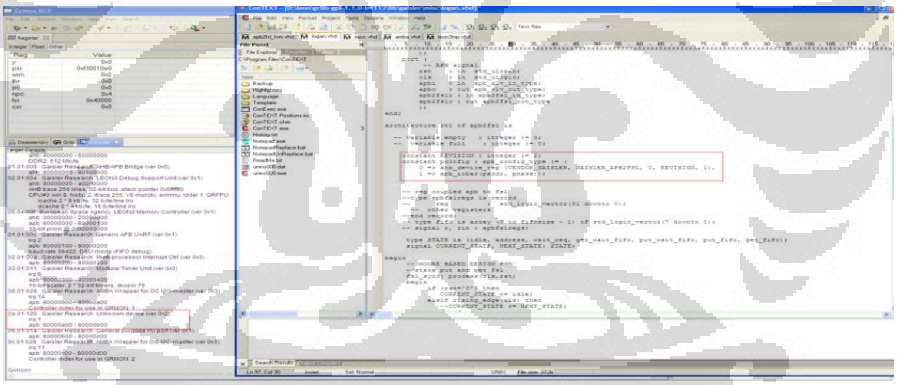


Figure 4.13: GRMON SoC LEON3 avec APBtoFSL

4.7. Propose AHB to FSL protocol adapter

L'absence du signal PREADY dans la version 2 de l'APB, nous oblige à nous rabattre sur l'utilisation du bus plus complexe AHB (présence de Hready et Hresp dans l'interface slave).

Les avantages d'utiliser AMBA AHB sont:

- haute performance
- fonctionnement en pipeline

- Les maîtres de bus multiples
- Les transferts Burst
- Les opérations split (4)

Figure 4.15 représente les signaux standards AHB:

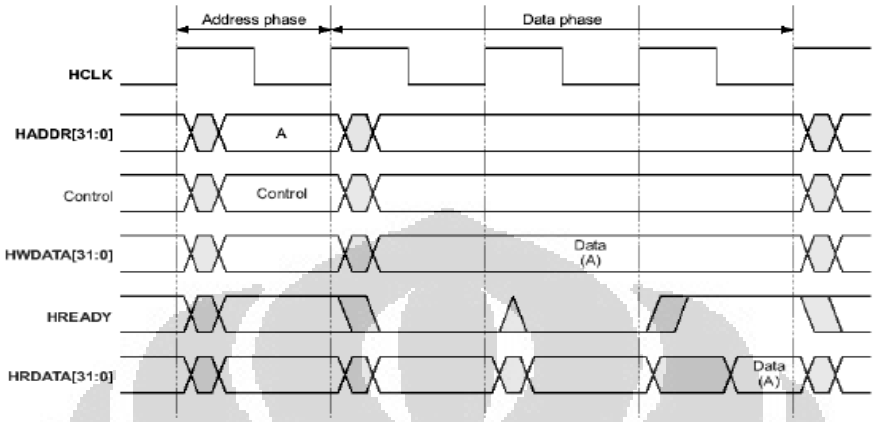


Figure 4.14: AHB standard de transfert

- HCLK: the bus clock source (rising-edge triggered)
- HRESETn: the bus (and typically system) reset signal (active low)
- HADDR: the AHB address bus
- HSELx: the select line for each slave device
- HWRITE: indicates transfer direction (Write=H, Read=L)
- HWDATA: the write data bus (can be up to 32-bits wide)
- HREADY: used to extend a transfer
- HRDATA: the read data bus (can be up to 32-bits wide)
- HRESP: status transfer (OKAY, ERROR,RETRY,SPLIT)
- CONTROL: HTRANS, HSIZE, HBURST, HPROT

Un bus de données d'écriture est utilisé pour déplacer des données du maître à un esclave, tandis qu'un bus de données de lecture est utilisé pour déplacer des données d'un esclave au transfert master. Chaque transfert se compose de:

- Un cycle d'adresse et de contrôle
- Les un ou plusieurs cycles des données.

L'adresse ne peut être prolongée et, par conséquent tous les esclaves doivent échantillonner l'adresse pendant ce temps. Les données, cependant, peut être étendue en utilisant le signal HREADY. Lorsque ce signal est à l'état LOW des états d'attente sont insérés dans le transfert. Lors d'un transfert, l'esclave indique l'état du transfert en utilisant les signaux de ré-

ponse : HRESP et OKAY. La réponse OKAY est utilisée pour indiquer que le transfert se déroule normalement et quand HREADY est HAUT ceci indique que le transfert est terminé avec succès (19). Les machines d'états finis du maître (*Master)* et de l'esclave (*Slave*) d'un transfert AHB sont:

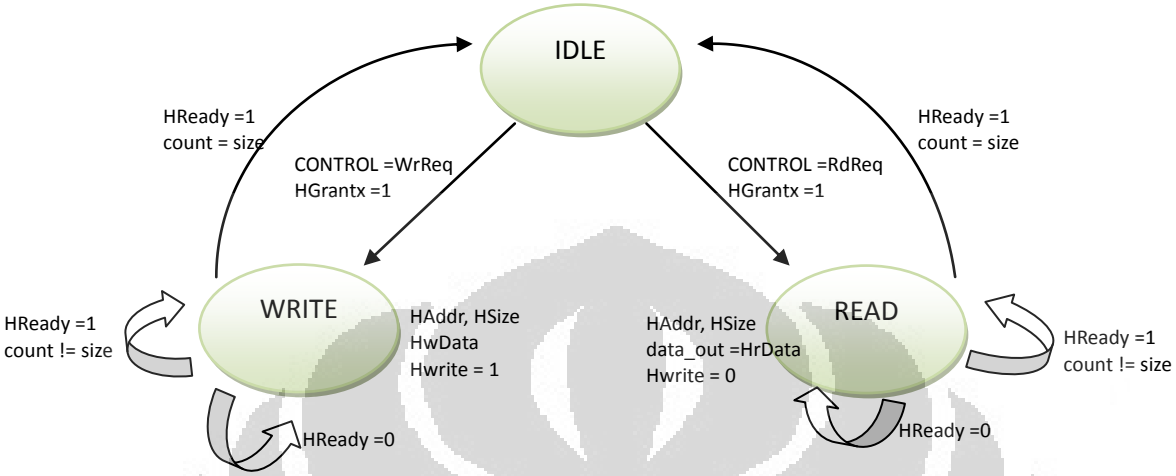


Figure 4.15: *finite state machine AHB master* [(19)]

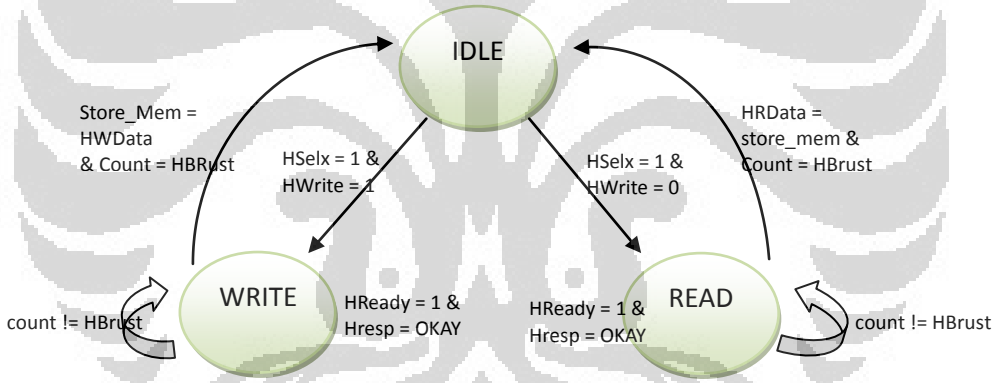


Figure 4.16: *finite state machine AHB slave* [(19)]

Dans le cas simple d'un transfert sans BURST, la machine d'état d'un wrapper AHB vers le protocole en 4 phases est :

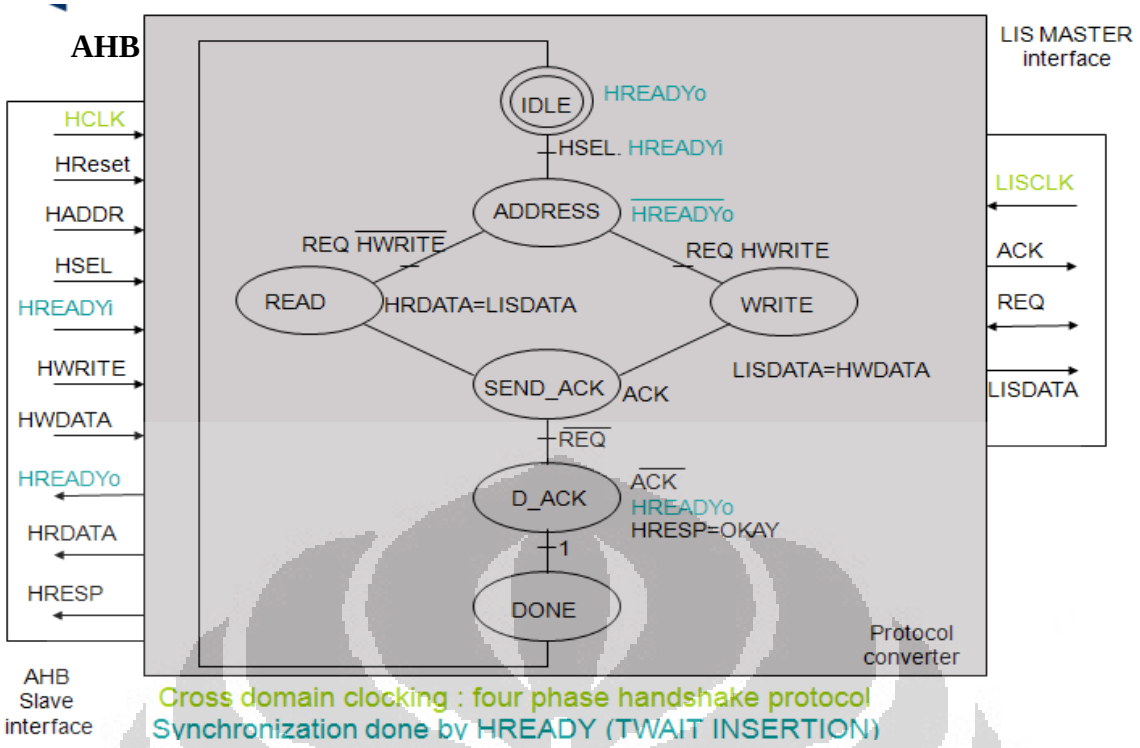


Figure 4.17: interface ahb Slave [(13)]

La wrapper AHB vers FSL est similaire à celui développé pour le bus APB et est donné la figure suivante :

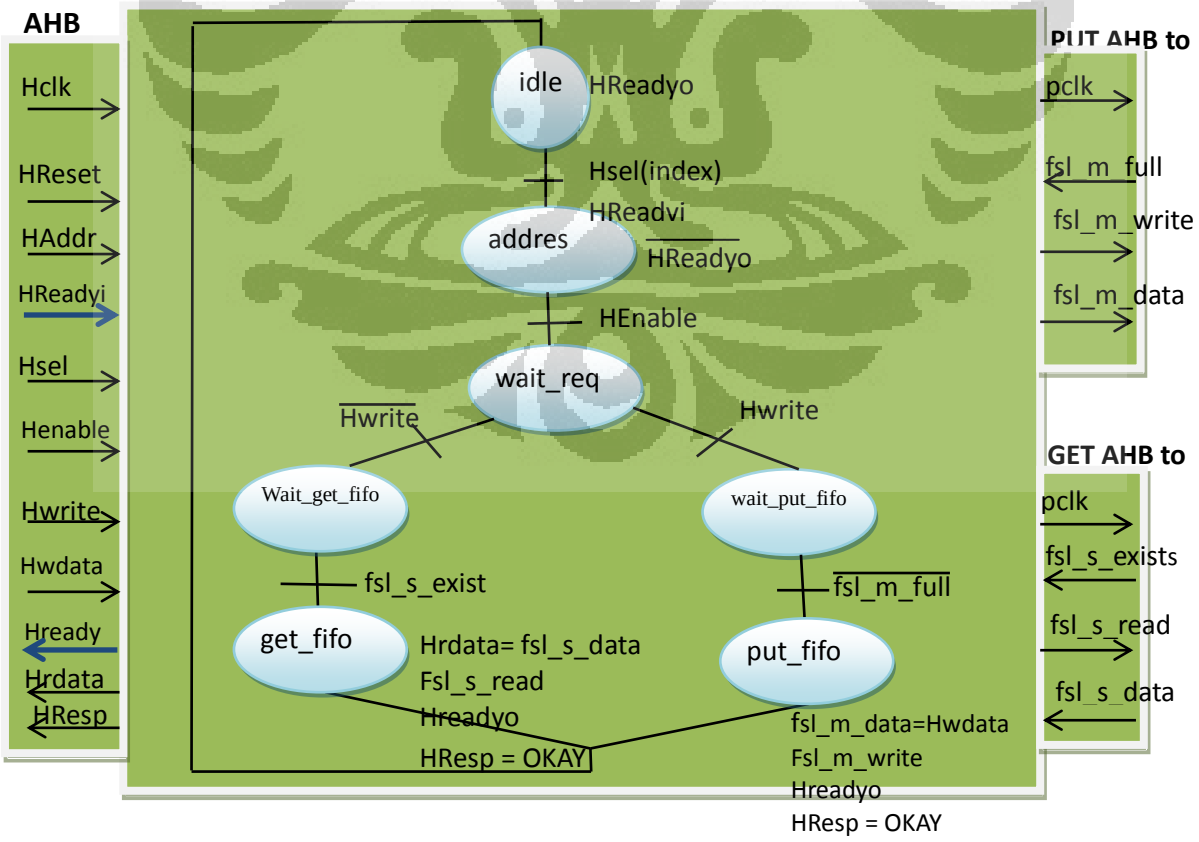


Figure 4.18: Conception Finite State Machine AHB to FSL Interface

CHAPITRE V

Conclusion et les Travaux Futurs

5.1. **Conclusion**

Ce stage a été effectué à partir du 1er Février 2012 au 22 Juin 2012 au sein du laboratoire LAB-STICC. L'objectif était d'étudier un Soc à base de Leon3 et différents modes de couplage entre un coprocesseur et le Leon 3.

Nous n'avons pas pu réaliser la solution du type ASIP car depuis le Leon3, l'interface coprocesseur des SPARC/Leon/Leon2 n'est plus accessible. Nous nous sommes alors rabattu vers le bus APB mais dans sa version actuelle la Grlib repose sur une version assez ancienne de l'AMBA (version 2.0) dans laquelle le bus APB ne dispose pas encore du signal PREADY. Or ce signal qui permet d'asservir le Leon aux périphériques (insertion de Twait) nous est indispensable pour coupler un coprocesseur asynchrone au Leon. Au final, il ne nous reste qu'un seul choix possible : le bus AHB.

Avec ce stage, j'ai acquis beaucoup d'expérience (matériel, logiciel, flot de conception) sur la conception de SoC à base de leon3.

5.2. **Travaux Futurs**

L'interface AHB2FSL conçue n'est pas optimale car elle ne permet pas d'exploiter le bus AHB à son maximum (BURST). Les Soc à base de Leon3 (GRLIB) ne disposant pas de DMA centraux du type « store and forward », la seule solution efficace est que le co-processeur est une interface de communication plus complexe du type DMA maître.

Bibliographies

(1) N.Ohba, K.Takano, "An SoC design methodology using FPGAs and embeded micro-processors," IEEE Cat. Conf Design Automation Proceedings , June. 2004

(2) **Leon3 Processor**. http://www.gaisler.com/cms/index.php

(3) The SPARC Architecture Manual. http**://www.sparc.org/standards/V8.pdf**

(4) AMBA 2.0 Specipication rev 2.0, 1999 (http://www-micro.deis.unibo.it/~magagni/amba99.pdf)

(5) patrick R. schaumont "a practical introduction to hardware/software codesign" springer USA : 2010 page 281

(6) Hans-Peter Rosinger "Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel". xilinx 2004

(7) **LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c).** **http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf**

(8) ML605 Hardware user guide. **http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf**

(9) GRLIB IP Library User's Manual. http://www.gaisler.com/products/grlib/grlib.pdf, January, 2012

(10) GRMON User's Manual. http://www.gaisler.com/doc/grmon.pdf

(11) LEON Integrated Development Environment for Eclipse. http://www.gaisler.com/doc/gr_lide.pdf

(12) *Juergen Wassner, Klaus Zahn, Ulrich Dersch*. "HARDWARE-SOFTWARE CODE-SIGN OF A TIGHTLY-COUPLED COPROCESSOR FOR VIDEO CONTENT ANALY-SIS". In IEEE sips 2010 pp 87-92: 2010

(13) Heller, Dominique. " Hardware/Software interface for GAUT HLS tool on leon/grlib plateform". Labsticc-UBS.

(14) Jiri Gaisler. "The LEON Processor User's Manual". Gaisler Research, August: 2001

(15) Buttelman, Lutz. "How to setup LEON3 VHDL simulation with Modelsim v0.1". 18 agust 2007.

(16) Tutorial - Getting started with SoC Leon 3 simulator. http://www.gnss-tracker.com/index.php?LinkID=4

(17) Jiri Gaisler, Marko Isomäki. "LEON3 GR-XC3S-1500 Template Design". Gaisler Research, October 2006

(18) AMBA® APB Protocol: version 2003-2010 ARM. http://www.arm.com

(19) Rishabh Singh Kurmi, Shruti Bhargava, Ajay Somkuwar. "Design of AHB Protocol Block for Advanced Microcontrollers". International Journal of Computer Applications (0975 – 8887), October 2011

(20) Enabling cpop interface. http://numanahsan.wordpress.com/2009/04/28/enabling-cpop-interface/

(21) floating point aritmatich http://flint.cs.yale.edu/cs422/doc/art-of-asm/pdf//CH14.PDF

(22) Arvanitakis Ioannis et Mamalis Dimitrios. "Replacing the SPARC-based core of the Leon3 HDL microprocessor model with a MIPS-based core ". open**core**s.org/usercontent,doc,1302422330

(23) CARLSSON.K."**Design and implementation of an on-chip logic analyzer".** Master's thesis CHALMERS UNIVERSITY OF TECHNOLOGY .Göteborg 2005

(24) Josë ROBERTO Sanchez Mayen. « Implementation an AC97Audio Controller IP". Master's thesis CHALMERS UNIVERSITY OF TECHNOLOGY .Göteborg 2011

(25) Leon3 Multiprocessing Cpu core.http://www.gaisler.com/doc/leon3_product_sheet.pdf

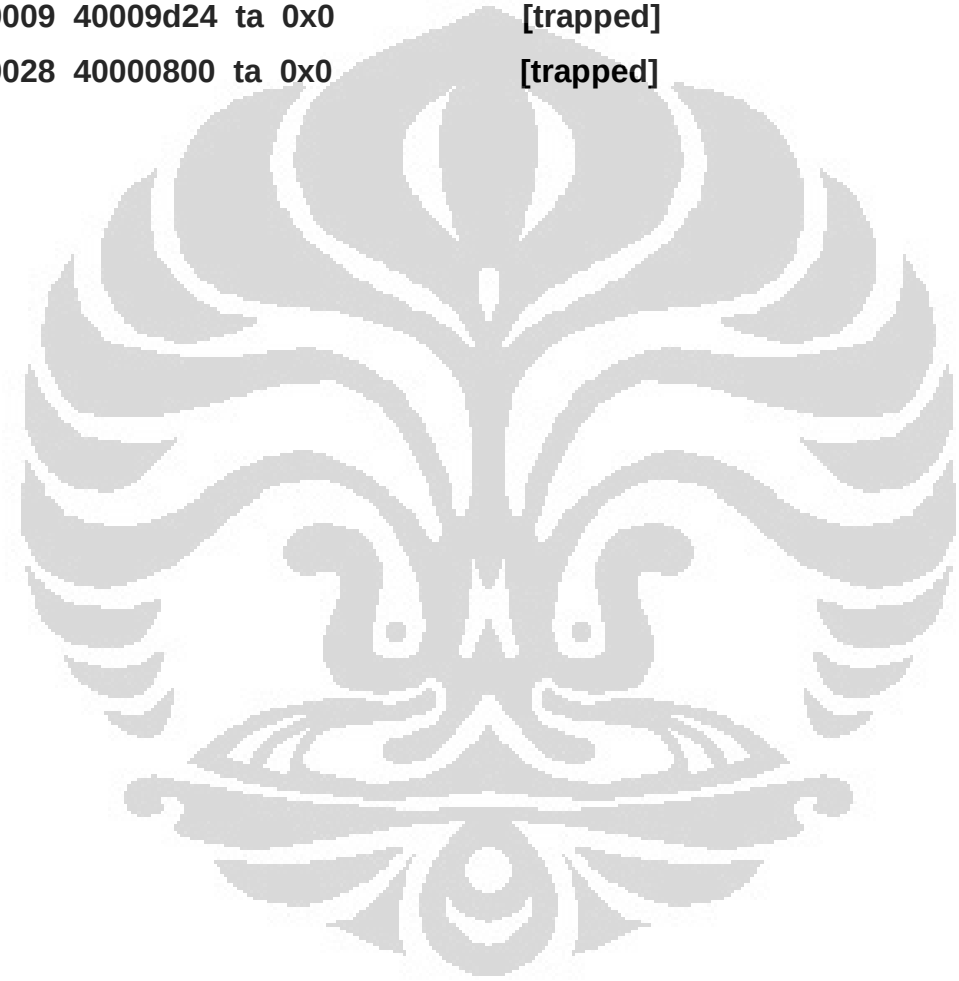
Annexes A

A.1 Trace disassemble grmon

time	address	instruction	result
10728524	4000967c	mov 13, %g1	[0000000d]
10728529	40009670	ld [%o7 + 0x4], %g1	[00100082]
10728531	40009674	andcc %g1, 0x4, %g0	[00000000]
10728532	40009678	be 0x40009670	[80000100]
10728533	4000967c	mov 13, %g1	[0000000d]
10728538	40009670	ld [%o7 + 0x4], %g1	[00100082]
10728540	40009674	andcc %g1, 0x4, %g0	[00000000]
10728541	40009678	be 0x40009670	[80000100]
10728542	4000967c	mov 13, %g1	[0000000d]
10728547	40009670	ld [%o7 + 0x4], %g1	[00100082]
10728549	40009674	andcc %g1, 0x4, %g0	[00000000]
10728550	40009678	be 0x40009670	[80000100]
10728551	4000967c	mov 13, %g1	[0000000d]
10728556	40009670	ld [%o7 + 0x4], %g1	[00100082]
10728558	40009674	andcc %g1, 0x4, %g0	[00000000]
10728559	40009678	be 0x40009670	[80000100]
10728560	4000967c	mov 13, %g1	[0000000d]
10728565	40009670	ld [%o7 + 0x4], %g1	[00100082]
10728567	40009674	andcc %g1, 0x4, %g0	[00000000]
10728568	40009678	be 0x40009670	[80000100]
10728569	4000967c	mov 13, %g1	[0000000d]
10728574	40009670	ld [%o7 + 0x4], %g1	[00100082]
10728576	40009674	andcc %g1, 0x4, %g0	[00000000]
10728577	40009678	be 0x40009670	[80000100]
10728578	4000967c	mov 13, %g1	[0000000d]
10728583	40009670	ld [%o7 + 0x4], %g1	[00100082]
10728585	40009674	andcc %g1, 0x4, %g0	[00000000]
10728586	40009678	be 0x40009670	[80000100]
10728587	4000967c	mov 13, %g1	[0000000d]
10728592	40009670	ld [%o7 + 0x4], %g1	[00100082]

……

```
10728960  400012b8  ld  [%sp + 0x6c], %o7       [40001068]
10728968  400012bc  ldd  [%sp + 0x70], %l0      [4000b400 00000003]
10728982  400012c0  retl                        [400012c0]
10728986  400012c4  sub  %sp, -144, %sp         [5fffff68]
10729002  40001070  call  0x40009d20            [40001070]
10729003  40001074  nop                         [00000000]
10729008  40009d20  mov  1, %g1                 [00000001]
10729009  40009d24  ta  0x0                     [trapped]
10729028  40000800  ta  0x0                     [trapped]
```



A.2: Readme.txt

```
This leon3 design is tailored to the Xilinx Virtex-6 ML605 board

http://www.xilinx.com/ml605

Simulation and synthesis
------------------------

The design uses the Xilinx MIG memory interface with an AHB-2.0
interface. The MIG source code cannot be distributed due to the
prohibitive Xilinx license, so the MIG must be re-generated with
coregen before simulation and synthesis can be done.

To generate the MIG and install the Xilinx unisim simulation
library, do as follows:

  make mig
  make install-secureip

This will ONLY work with ISE-13 installed, and the XILINX variable
properly set in the shell. To synthesize the design, do

  make ise

and then

  make ise-prog-fpga

to program the FPGA.

Design specifics
----------------

* Synthesis shold be done using ISE-13

* The DDR3 controller is implemented with Xilinx MIG-3.7 and
  runs of the 200 MHz clock. The DDR3 memory runs at 400 MHz
  (DDR3-800). grmon-1.1.47 or later is needed to detect the
   DDR3 memory.

* The AHB clock is generated by the MMCM module in the DDR3
  controller, and can be set to 80, 100 and 120 MHz.

* System reset is mapped to the CPU RESET button

* DSU break is mapped to GPIO south button

* LED 0 UART RX active

* LED 1 UART TX active

* LED 2 indicates processor in debug mode

* LED 3 indicates DDR3 PHY initialization done

* The GRETH core is enabled and runs without problems at both
  100 and 1000 Mbit. The 1000 Mbit operation requires the commercial
  version of grlib. The ethernet debug link is enabled by default,
  using IP address 192.168.0.52.
```

```
* 16-bit flash prom can be read at address 0. It can be programmed
  with GRMON version 1.1.16 or later.

* The system can be simulated if the secure IP models are installed:

  make install-secureip

  Then rebuild the scripts and simulation model:

  make distclean vsim

  Modelsim v6.5e or newer is required to build the secure IP models.

  The normal leon3 test bench cannot be executed as the DDR3 model
  does not support pre-loading.

* The application UART1 is connected to the USB/RS232 connector

* The JTAG DSU interface is enabled and accesible via the USB/JTAG port.
  Start grmon with -xilusb to connect.

* Output from GRMON is:

grmon -eth -ip 192.168.0.52 -u -nb

 GRMON LEON debug monitor v1.1.47 professional version

 Copyright (C) 2004-2010 Aeroflex Gaisler - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to support@gaisler.com

 ethernet startup.
 GRLIB build version: 4107

 initialising ...........
 detected frequency: 100 MHz

 Component                            Vendor
 LEON3 SPARC V8 Processor             Gaisler Research
 AHB Debug JTAG TAP                   Gaisler Research
 GR Ethernet MAC                      Gaisler Research
 Xilinx MIG DDR2 controller           Gaisler Research
 AHB/APB Bridge                       Gaisler Research
 LEON3 Debug Support Unit             Gaisler Research
 LEON2 Memory Controller              European Space Agency
 Generic APB UART                     Gaisler Research
 Multi-processor Interrupt Ctrl       Gaisler Research
 Modular Timer Unit                   Gaisler Research
 General purpose I/O port             Gaisler Research

 Use command 'info sys' to print a detailed report of attached cores

grlib> inf sys
00.01:003   Gaisler Research  LEON3 SPARC V8 Processor (ver 0x0)
             ahb master 0
01.01:01c   Gaisler Research  AHB Debug JTAG TAP (ver 0x1)
             ahb master 1
02.01:01d   Gaisler Research  GR Ethernet MAC (ver 0x0)
             ahb master 2, irq 12
             apb: 80000f00 - 80001000
```

```
             Device index: dev0
             1000 Mbit capable
             edcl ip 192.168.0.52, buffer 2 kbyte
00.01:06b   Gaisler Research  Xilinx MIG DDR2 controller (ver 0x0)
             ahb: 40000000 - 60000000
             DDR2: 512 Mbyte
01.01:006   Gaisler Research  AHB/APB Bridge (ver 0x0)
             ahb: 80000000 - 80100000
02.01:004   Gaisler Research  LEON3 Debug Support Unit (ver 0x1)
             ahb: 90000000 - a0000000
             AHB trace 256 lines, 32-bit bus, stack pointer 0x5ffffff0
             CPU#0 win 8, hwbp 2, itrace 256, V8 mul/div, srmmu, lddel 1,
GRFPU
                   icache 4 * 4 kbyte, 32 byte/line lru
                   dcache 4 * 4 kbyte, 16 byte/line lru
05.04:00f   European Space Agency  LEON2 Memory Controller (ver 0x1)
             ahb: 00000000 - 20000000
             apb: 80000000 - 80000100
             16-bit prom @ 0x00000000
01.01:00c   Gaisler Research  Generic APB UART (ver 0x1)
             irq 2
             apb: 80000100 - 80000200
             baud rate 38343, DSU mode (FIFO debug)
02.01:00d   Gaisler Research  Multi-processor Interrupt Ctrl (ver 0x3)
             apb: 80000200 - 80000300
03.01:011   Gaisler Research  Modular Timer Unit (ver 0x0)
             irq 8
             apb: 80000300 - 80000400
             8-bit scaler, 2 * 32-bit timers, divisor 100
0b.01:01a   Gaisler Research  General purpose I/O port (ver 0x1)
             apb: 80000b00 - 80000c00
grlib> fla

 Intel-style 16-bit flash on D[31:16]

 Manuf.    Intel
 Device    Strataflash P30

 Device ID ca29ffff242640e7
 User   ID ffffffffffffffff


 1 x 32 Mbyte = 32 Mbyte total @ 0x00000000


 CFI info
 flash family  : 1
 flash size    : 256 Mbit
 erase regions : 2
 erase blocks  : 259
 write buffer  : 1024 bytes
 lock-down     : yes
 region  0     : 255 blocks of 128 Kbytes
 region  1     : 4 blocks of 32 Kbytes

grlib>
```

A.3 La Carte XILINX VIRTEX ML605

Annexes B

B.1 apb2fsl.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
use grlib.devices.all;

library gaisler;
use gaisler.misc.all;

use std.textio.all;


entity apb2fsl is
    generic (
            -- APB signal
        pindex   : integer := 0;
        paddr    : integer := 0;
        pmask    : integer := 16#fff#

        );
    port (
         -- APB signal
        rst    : in  std_ulogic;
        clk    : in  std_ulogic;
        apbi   : in  apb_slv_in_type;
        apbo   : out apb_slv_out_type;
        apb2fsli : in apb2fsl_in_type;
        apb2fslo : out apb2fsl_out_type
        );
end;

architecture rtl of apb2fsl is

  -- variable empty   : integer := 0;
  --  variable full    : integer := 0;

    constant REVISION : integer := 2;
    constant pconfig : apb_config_type := (
        0 => ahb_device_reg (VENDOR_GAISLER, GAISLER_APB2FSL, 0, REVISION, 1),
        1 => apb_iobar(paddr, pmask));


    -- reg coupled apb to fsl
    --type apbfslregs is record
    --      reg       :  std_logic_vector(31 downto 0);
      --  --  other registers
    --end record;
    -- type fifo is array (0 to fifosize - 1) of std_logic_vector(7 downto 0);
    -- signal r, rin : apbfslregs;

     type  STATE  is  (idle,  address,  wait_req,  get_wait_fifo,  put_wait_fifo,  put_fifo,
get_fifo);
     signal CURRENT_STATE, NEXT_STATE: STATE;

begin
     -- MOORE BASED DESIGN FSM
     --state put and get fsl
     fsl_sync: process(clk,rst)
     begin
          if (rst='0') then
             CURRENT_STATE <= idle;
          elsif rising_edge(clk) then
            apb2fslo.FSL_M_Write <= '0';
            case CURRENT_STATE is
               when idle =>
                    if (apbi.psel(pindex)='1') then  CURRENT_STATE <=address;
                    end if;
               when address =>
                    if (apbi.penable='1') then  CURRENT_STATE <=wait_req;
```

```
                    end if;
                when wait_req =>
                    if (apbi.pwrite='0') then CURRENT_STATE <= get_wait_fifo;
                    elsif (apbi.pwrite='1') then CURRENT_STATE <= put_wait_fifo;
                    end if;
                when get_wait_fifo =>
                    if (apb2fsli.FSL_S_Exists='1') then  CURRENT_STATE<=get_fifo;
                    end if;
            when put_wait_fifo =>
                    if (apb2fsli.FSL_M_Full='0') then  CURRENT_STATE<=put_fifo;
                    end if;
            when put_fifo =>
                    apb2fslo.FSL_M_Write <= '1';
                    apb2fslo.FSL_M_Data<=apbi.pwdata;
                    --apbo.pready<='1';
                    CURRENT_STATE<=idle;
                when get_fifo =>
                    apb2fslo.FSL_S_Read <= '1';
                    apbo.prdata <= apb2fsli.FSL_S_Data;
                    --apbo.pready<='1';
                    CURRENT_STATE<=idle;
             end case;
           end if;
          apbo.pirq <= (others => '0');   -- No IRQ
  apbo.pindex <= pindex;          -- VHDL generic index
          apbo.pconfig <= pconfig;        -- config constant
       end process;
        bootmsg : report_version
          generic map ("apb_fsl" & tost(pindex) &
                       ": apb_fsl Unit " & tost(REVISION));
end;
```

B.2 FSL_V20

```
------------------------------------------------------------------------------
-- $Id: fsl_v20.vhd,v 1.1.2.1 2010/10/28 11:17:56 goran Exp $
------------------------------------------------------------------------------
-- fsl_v20.vhd - Entity and architecture
--
-- (c) Copyright [2003] - [2010] Xilinx, Inc. All rights reserved.
--
-- This file contains confidential and proprietary information
-- of Xilinx, Inc. and is protected under U.S. and
-- international copyright and other intellectual property
-- laws.
--
-- DISCLAIMER
-- This disclaimer is not a license and does not grant any
-- rights to the materials distributed herewith. Except as
-- otherwise provided in a valid license issued to you by
-- Xilinx, and to the maximum extent permitted by applicable
-- law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
-- WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
-- AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
-- BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
-- INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
-- (2) Xilinx shall not be liable (whether in contract or tort,
-- including negligence, or under any other theory of
9-- liability) for any loss or damage of any kind or nature
-- related to, arising under or in connection with these
-- materials, including for any direct, or any indirect,
-- special, incidental, or consequential loss or damage
-- (including loss of data, profits, goodwill, or any type of
-- loss or damage suffered as a result of any action brought
-- by a third party) even if such damage or loss was
-- reasonably foreseeable or Xilinx had been advised of the
-- possibility of the same.
--
-- CRITICAL APPLICATIONS
-- Xilinx products are not designed or intended to be fail-
-- safe, or for use in any application requiring fail-safe
-- performance, such as life-support or safety devices or
-- systems, Class III medical devices, nuclear facilities,
-- applications related to the deployment of airbags, or any
-- other applications that could lead to death, personal
-- injury, or severe property or environmental damage
-- (individually and collectively, "Critical
-- Applications"). Customer assumes the sole risk and
-- liability of any use of Xilinx products in Critical
-- Applications, subject only to applicable laws and
-- regulations governing limitations on product liability.
--
-- THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
-- PART OF THIS FILE AT ALL TIMES
--
------------------------------------------------------------------------------
-- Filename:        fsl_v20.vhd
--
-- Description:
--
-- VHDL-Standard:   VHDL'93
------------------------------------------------------------------------------
-- Structure:
--
fsl_v20.vhdenv\Databases\ip2\processor\hardware\doc\bram_block\bram_block_v1_00_a
--
------------------------------------------------------------------------------
-- Author:          satish
-- Revision:        $Revision: 1.1.2.1 $
-- Date:            $Date: 2010/10/28 11:17:56 $
--
-- History:
--   satish  2003-02-13    First Version
--   satish  2004-03-03    New Version
--   rolandp 2006-08-20    BRAM in asynch mode
------------------------------------------------------------------------------
-- Naming Conventions:
--      active low signals:                     "*_n"
--      clock signals:                          "clk", "clk_div#", "clk_#x"
```

```
--      reset signals:                          "rst", "rst_n"
--      generics:                               "C_*"
--      user defined types:                     "*_TYPE"
--      state machine next state:               "*_ns"
--      state machine current state:            "*_cs"
--      combinatorial signals:                  "*_com"
--      pipelined or register delay signals:    "*_d#"
--      counter signals:                        "*cnt*"
--      clock enable signals:                   "*_ce"
--      internal version of output port         "*_i"
--      device pins:                            "*_pin"
--      ports:                                  - Names begin with Uppercase
--      processes:                              "*_PROCESS"
--      component instantiations:               "<ENTITY_>I_<#|FUNC>
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
use grlib.devices.all;

library gaisler;
use gaisler.misc.all;
--use gaisler.misc.sync_fifo;
--use gaisler.misc.async_fifo;

--pragma translate_off
use std.textio.all;
--pragma translate_on

library Unisim;
use Unisim.vcomponents.all;


entity fsl_v20 is
  generic (
    C_EXT_RESET_HIGH    : integer := 1;
    C_ASYNC_CLKS        : integer := 0;
    C_IMPL_STYLE        : integer := 0;
    C_USE_CONTROL       : integer := 1;
    C_FSL_DWIDTH        : integer := 32;
    C_FSL_DEPTH         : integer := 16;
    C_READ_CLOCK_PERIOD : integer := 0
    );
  port (
    -- Clock and reset signals
    FSL_Clk : in  std_logic;
    SYS_Rst : in  std_logic;
    FSL_Rst : out std_logic;

    -- FSL master signals
    FSL_M_Clk     : in  std_logic;
    FSL_M_Data    : in  std_logic_vector(0 to C_FSL_DWIDTH-1);
    FSL_M_Control : in  std_logic;
    FSL_M_Write   : in  std_logic;
    FSL_M_Full    : out std_logic;

    -- FSL slave signals
    FSL_S_Clk     : in  std_logic;
    FSL_S_Data    : out std_logic_vector(0 to C_FSL_DWIDTH-1);
    FSL_S_Control : out std_logic;
    FSL_S_Read    : in  std_logic;
    FSL_S_Exists  : out std_logic;

    -- FIFO status signals
    FSL_Full        : out std_logic;
    FSL_Has_Data    : out std_logic;
    FSL_Control_IRQ : out std_logic
    );
end entity fsl_v20;

architecture IMP of fsl_v20 is

  component Sync_FIFO is
```

```
    generic (
      C_IMPL_STYLE : Integer;
      WordSize     : Integer;
      MemSize      : Integer);
    port (
      Reset : in Std_Logic;
      Clk   : in Std_Logic;

      WE      : in  Std_Logic;
      DataIn  : in  Std_Logic_Vector(WordSize-1 downto 0);
      Full    : out Std_Logic;
      RD      : in  Std_Logic;
      DataOut : out Std_Logic_Vector(WordSize-1 downto 0);
      Exists  : out Std_Logic);
  end component Sync_FIFO;

  component Async_FIFO is
    generic (
      WordSize : Integer;
      MemSize  : Integer;
      Protect  : Boolean);
    port (
      Reset   : in  Std_Logic;
      -- Clock region WrClk
      WrClk   : in  Std_Logic;
      WE      : in  Std_Logic;
      DataIn  : in  Std_Logic_Vector(WordSize-1 downto 0);
      Full    : out Std_Logic;
      -- Clock region RdClk
      RdClk   : in  Std_Logic;
      RD      : in  Std_Logic;
      DataOut : out Std_Logic_Vector(WordSize-1 downto 0);
      Exists  : out Std_Logic);
  end component Async_FIFO;

  component Async_FIFO_BRAM is
    generic (
      WordSize : Integer;
      MemSize  : Integer;
      Protect  : Boolean);
    port (
      Reset   : in  Std_Logic;
      -- Clock region WrClk
      WrClk   : in  Std_Logic;
      WE      : in  Std_Logic;
      DataIn  : in  Std_Logic_Vector(WordSize-1 downto 0);
      Full    : out Std_Logic;
      -- Clock region RdClk
      RdClk   : in  Std_Logic;
      RD      : in  Std_Logic;
      DataOut : out Std_Logic_Vector(WordSize-1 downto 0);
      Exists  : out Std_Logic);
  end component Async_FIFO_BRAM;

  signal sys_rst_i    : std_logic;
  signal srl_time_out : std_logic;
  signal fsl_rst_i    : std_logic;
  signal Data_In      : std_logic_vector(0 to C_FSL_DWIDTH);
  signal Data_Out     : std_logic_vector(0 to C_FSL_DWIDTH);

  signal fifo_full       : std_logic;
  -- signal fifo_half_full  : std_logic;
  -- signal fifo_half_empty : std_logic;
  signal fifo_has_data   : std_logic;

  signal fsl_s_control_i : std_logic;

  signal srl_clk : std_logic;

begin  -- architecture IMP

  SYS_RST_PROC : process (SYS_Rst) is
    variable sys_rst_input : std_logic;
  begin
    if C_EXT_RESET_HIGH = 0 then
      sys_rst_i <= not SYS_Rst;
    else
```

```
    sys_rst_i <= SYS_Rst;
  end if;
end process SYS_RST_PROC;

Rst_Delay_Async: if (C_ASYNC_CLKS /= 0) generate
  srl_clk <= FSL_M_Clk;

end generate Rst_Delay_Async;

Rst_Delay_Sync: if (C_ASYNC_CLKS = 0) generate
  srl_clk <= FSL_Clk;
end generate Rst_Delay_Sync;

POR_SRL_I : SRL16
  generic map (
    INIT => X"FFFF")
  port map (
    D   => '0',
    CLK => srl_Clk,
    A0  => '1',
    A1  => '1',
    A2  => '1',
    A3  => '1',
    Q   => srl_time_out);

POR_FF_I : FDS
  port map (
    Q => fsl_rst_i,
    D => srl_time_out,
    C => srl_Clk,
    S => sys_rst_i);

FSL_Rst <= fsl_rst_i;


------------------------------------------------------------------------------
-- Width is 1, so implement a registers
------------------------------------------------------------------------------
Only_Register : if (C_FSL_DEPTH = 1) generate
  signal fsl_s_exists_i : std_logic;
  signal fsl_m_full_i   : std_logic;
begin

  -- FSL_S_Clk and FSL_M_Clk are the same
  Sync_Clocks: if (C_ASYNC_CLKS = 0) generate

    FIFO : process (FSL_Clk) is
      variable fifo_full : std_logic;
    begin  -- process FIFO
      if FSL_Clk'event and FSL_Clk = '1' then  -- rising clock edge
        if fsl_rst_i = '1' then                -- synchronous reset (active high)
          fifo_full      := '0';
          Fsl_m_full_i   <= '1';
          Fsl_s_exists_i <= '0';
        else
          if (fifo_full = '0') then            -- Empty
            if (FSL_M_Write = '1') then
              fifo_full       := '1';
              FSL_S_Data      <= FSL_M_Data;
              fsl_s_control_i <= FSL_M_Control;
            end if;
          end if;
          if (fifo_full = '1') then            -- Has data
            if (FSL_S_Read = '1') then
              fifo_full := '0';
            end if;
          end if;
          Fsl_m_full_i   <= fifo_full;
          Fsl_s_exists_i <= fifo_full;
        end if;
      end if;
    end process FIFO;
  end generate Sync_Clocks;

  FSL_S_Exists <= fsl_s_exists_i;
  FSL_Has_Data <= fsl_s_exists_i;
```

```
  FSL_M_Full <= fsl_m_full_i;
  FSL_Full   <= fsl_m_full_i;

  FSL_S_Control   <= fsl_s_control_i;
  FSL_Control_IRQ <= fsl_s_control_i and fsl_s_exists_i;

end generate Only_Register;

Using_FIFO: if (C_FSL_DEPTH > 1) generate
begin
  -- Map Master Data/Control signal
  Data_In(0 to C_FSL_DWIDTH-1) <= FSL_M_Data;

  -- Map Slave Data/Control signal
  FSL_S_Data    <= Data_Out(0 to C_FSL_DWIDTH-1);

  -- SRL FIFO BASED IMPLEMENTATION
  Sync_FIFO_Gen : if (C_ASYNC_CLKS = 0) generate
    Use_Control: if (C_USE_CONTROL /= 0) generate

      Data_In(C_FSL_DWIDTH)        <= FSL_M_Control;
      fsl_s_control_i <= Data_Out(C_FSL_DWIDTH);

      Sync_FIFO_I1 : Sync_FIFO
        generic map (
          C_IMPL_STYLE => C_IMPL_STYLE,
          WordSize     => C_FSL_DWIDTH + 1,
          MemSize      => C_FSL_DEPTH)
        port map (
          Reset   => fsl_rst_i,
          Clk     => FSL_Clk,
          WE      => FSL_M_Write,
          DataIn  => Data_In,
          Full    => fifo_full,
          RD      => FSL_S_Read,
          DataOut => Data_Out,
          Exists  => fifo_has_data);
    end generate Use_Control;

    Use_Data: if (C_USE_CONTROL = 0) generate

      fsl_s_control_i <= '0';

      Sync_FIFO_I1 : Sync_FIFO
        generic map (
          C_IMPL_STYLE => C_IMPL_STYLE,
          WordSize     => C_FSL_DWIDTH,
          MemSize      => C_FSL_DEPTH)
        port map (
          Reset   => fsl_rst_i,
          Clk     => FSL_Clk,
          WE      => FSL_M_Write,
          DataIn  => Data_In(0 to C_FSL_DWIDTH-1),
          Full    => fifo_full,
          RD      => FSL_S_Read,
          DataOut => Data_Out(0 to C_FSL_DWIDTH-1),
          Exists  => fifo_has_data);

    end generate Use_Data;
  end generate Sync_FIFO_Gen;

  Async_FIFO_Gen: if (C_ASYNC_CLKS /= 0) generate

    Use_Control: if (C_USE_CONTROL /= 0) generate

      Data_In(C_FSL_DWIDTH)        <= FSL_M_Control;
      fsl_s_control_i <= Data_Out(C_FSL_DWIDTH);

      Use_DPRAM1: if (C_IMPL_STYLE = 0) generate
        -- LUT RAM implementation
        Async_FIFO_I1: Async_FIFO
          generic map (
            WordSize     => C_FSL_DWIDTH + 1,  -- [Integer]
            MemSize      => C_FSL_DEPTH,  -- [Integer]
            Protect      => true)         -- [Boolean]
          port map (
            Reset   => fsl_rst_i,         -- [in  Std_Logic]
```

```
            -- Clock region WrClk
            WrClk   => FSL_M_Clk,          -- [in  Std_Logic]
            WE      => FSL_M_Write,        -- [in  Std_Logic]
            DataIn  => Data_In,    -- [in  Std_Logic_Vector(WordSize-1 downto 0)]
            Full    => fifo_full,          -- [out Std_Logic]
            -- Clock region RdClk
            RdClk   => FSL_S_Clk,          -- [in  Std_Logic]
            RD      => FSL_S_Read,         -- [in  Std_Logic]
            DataOut => Data_Out,  -- [out Std_Logic_Vector(WordSize-1 downto 0)]
            Exists  => fifo_has_data);     -- [out Std_Logic]
        end generate Use_DPRAM1;

        Use_BRAM1: if (C_IMPL_STYLE /= 0) generate
          -- BRAM implementation
          Async_FIFO_BRAM_I1 : Async_FIFO_BRAM
            generic map (
              WordSize     => C_FSL_DWIDTH + 1,  -- [Integer]
              MemSize      => C_FSL_DEPTH,       -- [Integer]
              Protect      => true)              -- [Boolean]
            port map (
              Reset   => fsl_rst_i,          -- [in  Std_Logic]
              -- Clock region WrClk
              WrClk   => FSL_M_Clk,          -- [in  Std_Logic]
              WE      => FSL_M_Write,        -- [in  Std_Logic]
              DataIn  => Data_In,    -- [in  Std_Logic_Vector(WordSize-1 downto 0)]
              Full    => fifo_full,          -- [out Std_Logic]
              -- Clock region RdClk
              RdClk   => FSL_S_Clk,          -- [in  Std_Logic]
              RD      => FSL_S_Read,         -- [in  Std_Logic]
              DataOut => Data_Out,  -- [out Std_Lo gic_Vector(WordSize-1 downto 0)]
              Exists  => fifo_has_data);     -- [out Std_Logic]
        end generate Use_BRAM1;

      end generate Use_Control;

      Use_Data: if (C_USE_CONTROL = 0) generate

        fsl_s_control_i <= '0';

        Use_DPRAM0: if (C_IMPL_STYLE = 0) generate
          -- LUT RAM implementation
          Async_FIFO_I1 : Async_FIFO
            generic map (
              WordSize     => C_FSL_DWIDTH,  -- [Integer]
              MemSize      => C_FSL_DEPTH,   -- [Integer]
              Protect      => true)          -- [Boolean]
            port map (
              Reset   => fsl_rst_i,          -- [in  Std_Logic]
              -- Clock region WrClk
              WrClk   => FSL_M_Clk,          -- [in  Std_Logic]
              WE      => FSL_M_Write,        -- [in  Std_Logic]
              DataIn  => Data_In(0 to C_FSL_DWIDTH-1),   -- [in  Std_Logic_Vector(WordSize-1
downto 0)]
              Full    => fifo_full,          -- [out Std_Logic]
              -- Clock region RdClk
              RdClk   => FSL_S_Clk,          -- [in  Std_Logic]
              RD      => FSL_S_Read,         -- [in  Std_Logic]
              DataOut => Data_Out(0 to C_FSL_DWIDTH-1),  -- [out Std_Logic_Vector(WordSize-1
downto 0)]
              Exists  => fifo_has_data);     -- [out Std_Logic]
        end generate Use_DPRAM0;

        Use_BRAM0: if (C_IMPL_STYLE /= 0) generate
          -- BRAM implementation
          Async_FIFO_BRAM_I1 : Async_FIFO_BRAM
            generic map (
              WordSize     => C_FSL_DWIDTH,  -- [Integer]
              MemSize      => C_FSL_DEPTH,   -- [Integer]
              Protect      => true)          -- [Boolean]
            port map (
              Reset   => fsl_rst_i,          -- [in  Std_Logic]
              -- Clock region WrClk
              WrClk   => FSL_M_Clk,          -- [in  Std_Logic]
              WE      => FSL_M_Write,        -- [in  Std_Logic]
              DataIn  => Data_In(0 to C_FSL_DWIDTH-1),   -- [in  Std_Logic_Vector(WordSize-1
downto 0)]
              Full    => fifo_full,          -- [out Std_Logic]
```

```
              -- Clock region RdClk
              RdClk   => FSL_S_Clk,          -- [in  Std_Logic]
              RD      => FSL_S_Read,         -- [in  Std_Logic]
              DataOut => Data_Out(0 to C_FSL_DWIDTH-1),   -- [out Std_Logic_Vector(WordSize-1
downto 0)]
              Exists  => fifo_has_data);     -- [out Std_Logic]
          end generate Use_BRAM0;

      end generate Use_Data;

    end generate Async_FIFO_Gen;

    FSL_M_Full <= fifo_full or fsl_rst_i;  -- Inhibit writes during reset by
                                           -- forcing full to '1'
    FSL_S_Exists <= fifo_has_data;

    FSL_Full     <= fifo_full;
    FSL_Has_Data <= fifo_has_data;

    FSL_S_Control   <= fsl_s_control_i;
    FSL_Control_IRQ <= fsl_s_control_i and fifo_has_data;

  end generate Using_FIFO;

end architecture IMP;
```

B.3 Testbench system

```
------------------------------------------------------------------------------
--  LEON3 Demonstration design test bench
--  Copyright (C) 2004 Jiri Gaisler, Gaisler Research
------------------------------------------------------------------------------
--  This file is a part of the GRLIB VHDL IP LIBRARY
--  Copyright (C) 2003 - 2008, Gaisler Research
--  Copyright (C) 2008 - 2010, Aeroflex Gaisler
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
library gaisler;
use gaisler.libdcom.all;
use gaisler.sim.all;
library techmap;
use techmap.gencomp.all;
library micron;
use micron.components.all;
use work.debug.all;

use work.config.all;
use work.ml605.all;

entity testbench is
  generic (
    fabtech   : integer := CFG_FABTECH;
    memtech   : integer := CFG_MEMTECH;
    padtech   : integer := CFG_PADTECH;
    disas     : integer := CFG_DISAS;   -- Enable disassembly to console
    dbguart   : integer := CFG_DUART;   -- Print UART on console
    pclow     : integer := CFG_PCLOW;
    clkperiod : integer := 37           -- system clock period
    );
end;

architecture behav of testbench is
  constant promfile  : string  := "prom.srec";        -- rom contents
  constant sdramfile : string  := "sdram.srec";       -- sdram contents

  constant lresp    : boolean := false;
  constant ct       : integer := clkperiod/2;

  signal clk        : std_logic := '0';
  signal clk200p    : std_logic := '1';
  signal clk200n    : std_logic := '0';
  signal rst        : std_logic := '0';
  signal rstn1      : std_logic;
  signal rstn2      : std_logic;
  signal error      : std_logic;

  -- PROM flash
  signal address    : std_logic_vector(24 downto 0);
  signal data       : std_logic_vector(15 downto 0);
  signal romsn      : std_logic;
  signal oen        : std_ulogic;
  signal writen     : std_ulogic;
  signal iosn       : std_ulogic;

  -- DDR3 memory

  signal ddr3_dq       : std_logic_vector(DQ_WIDTH-1 downto 0);
```

```
  signal ddr3_dm       : std_logic_vector(DM_WIDTH-1 downto 0);
  signal ddr3_addr     : std_logic_vector(ROW_WIDTH-1 downto 0);
  signal ddr3_ba       : std_logic_vector(BANK_WIDTH-1 downto 0);
  signal ddr3_ras_n    : std_logic;
  signal ddr3_cas_n    : std_logic;
  signal ddr3_we_n     : std_logic;
  signal ddr3_reset_n  : std_logic;
  signal ddr3_cs_n     : std_logic_vector((CS_WIDTH*nCS_PER_RANK)-1 downto 0);
  signal ddr3_odt      : std_logic_vector((CS_WIDTH*nCS_PER_RANK)-1 downto 0);
  signal ddr3_cke      : std_logic_vector(CKE_WIDTH-1 downto 0);
  signal ddr3_dqs_p    : std_logic_vector(DQS_WIDTH-1 downto 0);
  signal ddr3_dqs_n    : std_logic_vector(DQS_WIDTH-1 downto 0);
  signal ddr3_tdqs_n   : std_logic_vector(DQS_WIDTH-1 downto 0);
  signal ddr3_ck_p     : std_logic_vector(CK_WIDTH-1 downto 0);
  signal ddr3_ck_n     : std_logic_vector(CK_WIDTH-1 downto 0);
  signal sda           : std_logic;
  signal scl           : std_logic;


  -- Debug support unit
  signal dsubre     : std_ulogic;

  -- AHB Uart
  signal dsurx      : std_ulogic;
  signal dsutx      : std_ulogic;

  -- APB Uart
  signal urxd       : std_ulogic;
  signal utxd       : std_ulogic;

  -- Ethernet signals
  signal etx_clk    : std_ulogic;
  signal erx_clk    : std_ulogic;
  signal erxdt      : std_logic_vector(7 downto 0);
  signal erx_dv     : std_ulogic;
  signal erx_er     : std_ulogic;
  signal erx_col    : std_ulogic;
  signal erx_crs    : std_ulogic;
  signal etxdt      : std_logic_vector(7 downto 0);
  signal etx_en     : std_ulogic;
  signal etx_er     : std_ulogic;
  signal emdc       : std_ulogic;
  signal emdio      : std_logic;
  signal emdint     : std_logic;
  signal egtx_clk   : std_logic;
  signal gmiiclk_p  : std_logic := '1';
  signal gmiiclk_n  : std_logic := '0';

  -- Output signals for LEDs
  signal led       : std_logic_vector(4 downto 0);

signal iic_scl_main, iic_sda_main : std_logic;
signal iic_scl_dvi, iic_sda_dvi : std_logic;
signal tft_lcd_data    : std_logic_vector(11 downto 0);
signal tft_lcd_clk_p   : std_logic;
signal tft_lcd_clk_n   : std_logic;
signal tft_lcd_hsync   : std_logic;
signal tft_lcd_vsync   : std_logic;
signal tft_lcd_de      : std_logic;
signal tft_lcd_reset_b : std_logic;
signal sysace_mpa      : std_logic_vector(6 downto 0);
signal sysace_mpce     : std_ulogic;
signal sysace_mpirq    : std_ulogic;
signal sysace_mpoe     : std_ulogic;
signal sysace_mpwe     : std_ulogic;
signal sysace_d        : std_logic_vector(7 downto 0);
signal clk_33          : std_ulogic := '0';

  signal brdyn     : std_ulogic;
  signal errorn  : std_logic;

begin
  -- clock and reset
  clk        <= not clk after ct * 1 ns;
  clk200p    <= not clk200p after 2.5 ns;
  clk200n    <= not clk200n after 2.5 ns;
  gmiiclk_p    <= not gmiiclk_p after 4 ns;
```

```
  gmiiclk_n    <= not gmiiclk_n after 4 ns;
  clk_33       <= not clk_33 after 15 ns;
  rst        <= '1', '0' after 1000 ns;
  rstn1      <= not rst;
  dsubre     <= '0';
  urxd       <= 'H';

  d3 : entity work.leon3mp
    generic map (fabtech, memtech, padtech, disas, dbguart, pclow)
    port map (
      reset     => rst,
      errorn    => error,
      clk_ref_p    => clk200p,
      clk_ref_n    => clk200n,

      -- PROM
      address   => address(24 downto 1),
      data      => data(15 downto 0),
      romsn     => romsn,
      oen       => oen,
      writen    => writen,

      -- DDR3
      ddr3_dq       => ddr3_dq,
      ddr3_dm       => ddr3_dm,
      ddr3_addr     => ddr3_addr,
      ddr3_ba       => ddr3_ba,
      ddr3_ras_n    => ddr3_ras_n,
      ddr3_cas_n    => ddr3_cas_n,
      ddr3_we_n     => ddr3_we_n,
      ddr3_reset_n  => ddr3_reset_n,
      ddr3_cs_n     => ddr3_cs_n,
      ddr3_odt      => ddr3_odt,
      ddr3_cke      => ddr3_cke,
      ddr3_dqs_p    => ddr3_dqs_p,
      ddr3_dqs_n    => ddr3_dqs_n,
      ddr3_ck_p     => ddr3_ck_p,
      ddr3_ck_n     => ddr3_ck_n,
--      sda           => sda,
--      scl           => scl,

      -- Debug Unit
      dsubre    => dsubre,

      -- AHB Uart
      dsutx     => dsutx,
      dsurx     => dsurx,

      -- PHY
      gmiiclk_p  => gmiiclk_p,
      gmiiclk_n  => gmiiclk_n,
      egtx_clk  => egtx_clk,
      etx_clk   => etx_clk,
      erx_clk   => erx_clk,
      erxd      => erxdt(7 downto 0),
      erx_dv    => erx_dv,
      erx_er    => erx_er,
      erx_col   => erx_col,
      erx_crs   => erx_crs,
      emdint    => emdint,
      etxd      => etxdt(7 downto 0),
      etx_en    => etx_en,
      etx_er    => etx_er,
      emdc      => emdc,
      emdio     => emdio,

      -- Output signals for LEDs
        iic_scl_main => iic_scl_main,
       iic_sda_main => iic_sda_main,
        dvi_iic_scl => iic_scl_dvi,
       dvi_iic_sda => iic_sda_dvi,
        tft_lcd_data => tft_lcd_data,
       tft_lcd_clk_p => tft_lcd_clk_p,
       tft_lcd_clk_n => tft_lcd_clk_n,
       tft_lcd_hsync => tft_lcd_hsync,
        tft_lcd_vsync => tft_lcd_vsync,
       tft_lcd_de => tft_lcd_de,
```

```
        tft_lcd_reset_b => tft_lcd_reset_b,
         clk_33 => clk_33,
         sysace_mpa => sysace_mpa,
        sysace_mpce => sysace_mpce,
        sysace_mpirq => sysace_mpirq,
        sysace_mpoe => sysace_mpoe,
         sysace_mpwe => sysace_mpwe,
        sysace_d => sysace_d,
         led => led
       );

  ddr3mem : for i in 0 to 3 generate
  -- u0 : entity micron.ddr3
   u0 : entity micron.MT46V16M16
        generic map(fname => sdramfile)
port map (
    --rst_n => ddr3_reset_n,
    clk => ddr3_ck_p(0),
    clk_n => ddr3_ck_n(0),
    cke => ddr3_cke(0),
    cs_n => ddr3_cs_n(0),
    ras_n => ddr3_ras_n,
    cas_n => ddr3_cas_n,
    we_n => ddr3_we_n,
    dm => ddr3_dm(i*2+1 downto i*2),
    ba => ddr3_ba,
    addr => ddr3_addr,
    dq => ddr3_dq(16*i+15 downto 16*i),
    dqs => ddr3_dqs_p(i*2+1 downto i*2)
    --dqs_n => ddr3_dqs_n(i*2+1 downto i*2),
    --tdqs_n => ddr3_tdqs_n(i*2+1 downto i*2),
    --odt => ddr3_odt(0)
);
  end generate;

--  prom0 : sram
 --   generic map (index => 6, abits => 24, fname => promfile)
 --   port map (address(24 downto 1), data(32 downto 25), romsn, writen, oen);

 --
  address (0) <= '0';
  prom0 : for i in 0 to 1 generate
     sr0 : sram generic map (index => i+4, abits => 24, fname => promfile)
       port map (address(24 downto 1), data(15-i*8 downto 8-i*8), romsn,
                 writen, oen);
 end generate;


  phy0 : if (CFG_GRETH = 1) generate
    emdio <= 'H';
    p0: phy
      generic map (address => 7)
      port map(rstn1, emdio, etx_clk, erx_clk, erxdt, erx_dv, erx_er,
               erx_col, erx_crs, etxdt, etx_en, etx_er, emdc, egtx_clk);
  end generate;

--  spimem0: if CFG_SPIMCTRL = 1 generate
--    s0 : spi_flash generic map (ftype => 4, debug => 0, fname => promfile,
--                                readcmd => CFG_SPIMCTRL_READCMD,
--                                dummybyte => CFG_SPIMCTRL_DUMMYBYTE,
--                                   dualoutput => 0)  -- Dual output is not supported in this
design
--      port map (spi_clk, spi_mosi, data(24), spi_sel_n);
--  end generate spimem0;

  error <= 'H';                         -- ERROR pull-up

  iuerr : process
  begin
    wait for 5000 us;
    if to_x01(errorn) = '1' then wait on errorn; end if;
     assert (to_x01(errorn) = '1')
   --assert (to_X01(error) = '1')
    report "*** IU in error mode, simulation halted ***"
     severity failure;
 end process;
```

```
  data <= buskeep(data) after 5 ns;

  dsucom : process
    procedure dsucfg(signal dsurx : in std_ulogic; signal dsutx : out std_ulogic) is
      variable w32 : std_logic_vector(31 downto 0);
      variable c8  : std_logic_vector(7 downto 0);
      constant txp : time := 160 * 1 ns;
    begin
      dsutx  <= '1';
      wait;
      wait for 5000 ns;
      txc(dsutx, 16#55#, txp);          -- sync uart
      txc(dsutx, 16#a0#, txp);
      txa(dsutx, 16#40#, 16#00#, 16#00#, 16#00#, txp);
      rxi(dsurx, w32, txp, lresp);

-- txc(dsutx, 16#c0#, txp);
-- txa(dsutx, 16#90#, 16#00#, 16#00#, 16#00#, txp);
-- txa(dsutx, 16#00#, 16#00#, 16#00#, 16#ef#, txp);
--
-- txc(dsutx, 16#c0#, txp);
-- txa(dsutx, 16#90#, 16#00#, 16#00#, 16#20#, txp);
-- txa(dsutx, 16#00#, 16#00#, 16#ff#, 16#ff#, txp);
--
-- txc(dsutx, 16#c0#, txp);
-- txa(dsutx, 16#90#, 16#40#, 16#00#, 16#48#, txp);
-- txa(dsutx, 16#00#, 16#00#, 16#00#, 16#12#, txp);
--
-- txc(dsutx, 16#c0#, txp);
-- txa(dsutx, 16#90#, 16#40#, 16#00#, 16#60#, txp);
-- txa(dsutx, 16#00#, 16#00#, 16#12#, 16#10#, txp);
--
-- txc(dsutx, 16#80#, txp);
-- txa(dsutx, 16#90#, 16#00#, 16#00#, 16#00#, txp);
-- rxi(dsurx, w32, txp, lresp);
    end;
  begin
    dsucfg(dsutx, dsurx);
    wait;
  end process;
end;
```

B.4 Testbench standalone

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
library gaisler;
--use gaisler.libdcom.all;
--use gaisler.sim.all;
use gaisler.misc.all;
--use gaisler.ambatest.all;
library grlib;
use grlib.stdlib.conv_std_logic_vector;
use grlib.stdlib.conv_integer;
use grlib.stdlib.conv_std_logic;
use grlib.stdlib.tost;
use grlib.stdlib."+";
use grlib.testlib.print;
use grlib.amba.all;
use grlib.AMBA_TestPackage.all;
--use grlib.at_pkg.all;
--use grlib.at_ahb_mst_pkg.all;
--library techmap;-
--use techmap.gencomp.all;
--library micron;
--use micron.components.all;
--use work.debug.all;
--use work.config.all;
--use work.ml605.all;
-- entity testbench
entity apb2fsl_tb is
end entity apb2fsl_tb;
-- architecture
architecture behavioural of apb2fsl_tb is
  -- component
  -- apb2fsl compenent
component apb2fsl
   generic (
            -- APB signal
        pindex   : integer := 0;
        paddr    : integer := 0;
        pmask    : integer := 16#fff#
          );
    port (
        --  APB signal
        rst    : in  std_ulogic;
        clk    : in  std_ulogic;
        apbi   : in  apb_slv_in_type;
        apbo   : out apb_slv_out_type;
        apb2fsli : in apb2fsl_in_type;
        apb2fslo : out apb2fsl_out_type
        );
end component;
-- ============================================================
-- fsl v20

component fsl_v20
  generic (
    C_EXT_RESET_HIGH    : integer := 1;
    C_ASYNC_CLKS        : integer := 0;
    C_IMPL_STYLE        : integer := 0;
    C_USE_CONTROL       : integer := 1;
    C_FSL_DWIDTH        : integer := 32;
    C_FSL_DEPTH         : integer := 16;
    C_READ_CLOCK_PERIOD : integer := 0
    );
  port (
    -- Clock and reset signals
    FSL_Clk : in  std_logic;
    SYS_Rst : in  std_logic;
    FSL_Rst : out std_logic;
    -- FSL master signals
    FSL_M_Clk     : in  std_logic;
    FSL_M_Data    : in  std_logic_vector(0 to C_FSL_DWIDTH-1);
    FSL_M_Control : in  std_logic;
    FSL_M_Write   : in  std_logic;
    FSL_M_Full    : out std_logic;
```

```
    -- FSL slave signals
    FSL_S_Clk     : in  std_logic;
    FSL_S_Data    : out std_logic_vector(0 to C_FSL_DWIDTH-1);
    FSL_S_Control : out std_logic;
    FSL_S_Read    : in  std_logic;
    FSL_S_Exists  : out std_logic;
    -- FIFO status signals
    FSL_Full        : out std_logic;
    FSL_Has_Data    : out std_logic;
    FSL_Control_IRQ : out std_logic
    );
end component;
--=================================================================
-- tests
   constant do_basic_read_test : boolean := true;
   constant do_basic_write_test : boolean := true;
   constant vmode               : boolean := false;
   constant sysperiod_c         : time    := 10 ns;
   --constant datareg_c           : std_logic_vector (31 downto 0) := uartaddr_c;
   --constant statusreg_c         : std_logic_vector (31 downto 0) := uartaddr_c+4;
   --constant ctrlreg_c           : std_logic_vector (31 downto 0) := uartaddr_c+8;
   --constant scalerreg_c         : std_logic_vector (31 downto 0) := uartaddr_c+12;
   --constant fifodbgreg_c        : std_logic_vector (31 downto 0) := uartaddr_c+16;
   --constant  uartaddr_c                 :  std_logic_vector  (31  downto  0)  :=  appaddr_c  +
(conv_std_logic_vector(paddr, 12)
   --                           and conv_std_logic_vector (pmask, 12));
   constant size                : integer := 16;

-- ============================================================================================
-- signal                                                                                  ===
-- ============================================================================================
signal rstn : std_ulogic := '0';
signal clk  : std_ulogic := '1';
signal apbi : apb_slv_in_type;
signal apbo : apb_slv_out_type;

-- apb2fsl
  signal apb2fsli : apb2fsl_in_type;
  signal apb2fslo : apb2fsl_out_type;
-- ================================================================
-- apb converter
 signal FSL_Clk : std_logic;
 signal FSL_Rst : std_logic;

    -- FSL master signals
signal FSL_M_Clk     : std_logic;
signal FSL_M_Data    : std_logic_vector(0 to 31);
signal FSL_M_Control : std_logic;
signal FSL_M_Write   : std_logic;
signal FSL_M_Full    : std_logic;

    -- FSL slave signals
signal FSL_S_Clk     : std_logic;
signal FSL_S_Data    : std_logic_vector(0 to 31);
signal FSL_S_Control : std_logic;
signal FSL_S_Read    : std_logic;
signal FSL_S_Exists  : std_logic;
--================================================================
-- function
--================================================================


-- ================================================================
begin
-----------------------------------------------------------------------
-- clk generation
-----------------------------------------------------------------------
clk <= not clk after (sysperiod_c/2);
rstn <= '0', '1' after 300 ns;



-----------------------------------------------------------------------
--- APB FSL converter (PUT_FSL and GET_FSL)
-----------------------------------------------------------------------
apb2fsl0 : apb2fsl generic map (pindex => 0, paddr => 10)
```

```
          port map (rstn, clk, apbi => apbi, apbo => apbo, apb2fsli => apb2fsli, apb2fslo =>
apb2fslo);
------------------------------------------------------------------------
---fsl converter
------------------------------------------------------------------------
fsl0 : fsl_v20 generic map (C_EXT_RESET_HIGH => 1, C_ASYNC_CLKS => 0, C_IMPL_STYLE => 0,
C_USE_CONTROL => 1,
                            C_FSL_DWIDTH => 32, C_FSL_DEPTH => 16, C_READ_CLOCK_PERIOD => 0)
      port map (clk, rstn, FSL_M_Clk => apb2fslo.FSL_M_Clk , FSL_M_Data =>
apb2fslo.FSL_M_Data,
                  FSL_M_Control => '0' , FSL_M_Write => apb2fslo.FSL_M_Write, FSL_M_Full =>
apb2fsli.FSL_M_Full,
                  FSL_S_Clk => apb2fslo.FSL_S_Clk , FSL_S_Data => apb2fsli.FSL_S_Data,
FSL_S_Control => open,
                  FSL_S_Read => apb2fslo.FSL_S_Read, FSL_S_Exists => apb2fsli.FSL_S_Exists);


------------------------------------------------------------------------
test_apb : process is
        variable tp : boolean;
        variable tpcounter : integer;
        variable d : std_logic_vector (31 downto 0);
        variable c : std_logic_vector (31 downto 0);
        variable e : std_logic;
        variable time0 : time;
        variable time1 : time;
        variable dummy : boolean;

begin
    wait until rstn='1';
     print ("testbench start");
------------------------------------------------------------------------
-- test write
------------------------------------------------------------------------
if do_basic_write_test then print ("test 1: writing "& time'image(now));
--for i in 0 to 32 loop
   -- d := conv_std_logic_vector (4, 32);
   d := x"00000001";
    c := x"80000a00";
    apbwrite(c, d, clk, apbi, apbo, "APBWrite",
                vmode, false, 0);
   --print ("written data at address "& tost (apbaddr_c+4)&" is "&tost(d));
 --   end loop;
end if;
------------------------------------------------------------------------
-- test read
------------------------------------------------------------------------
if do_basic_read_test then print ("test 2: read the writen values ... "& time'image(now));
   --for i in 0 to size loop
    -- with amba_tp.vhd
    APBRead(c, d, clk, apbi, apbo, "APBRead",
                vmode, false, 0);
    --print ("read data at address "& tost (apbaddr_c+1)&" is "&tost(d));
    -- end loop;
   end if;

    wait for 7500 ns;

   assert false report "testbench ended!" severity failure;
end process;
end architecture;
```

B.5 Program C pour test FPU

```
/*
 * fpu.c
 *
 *  Created on: 12 mars 2012
 *      Author: e1104008
 */
#include "leon3.h"
#include "testmod.h"
#include <math.h>
//#include <cpmath.h> //problem avec libcpmath.a

extern int xgetpsr();
extern void setpsr(int psr);

int __errno;
int main(int argc, char *argv[])

{
      volatile float a,x;
      float b= -0.25, c = 68.35,d = 45.30,e = 123.56;
      int tmp;

      tmp = xgetpsr();
      setpsr(tmp | (1 << 12) | (1 << 13));
      tmp = xgetpsr();
      if (!(tmp & (1 <<12))) return(0);
      set_fsr(0);

      //report_subtest(FPU_TEST+(get_pid()<<4));
      a = sqrt(e);
      x = exp(b);
      test_fpu();
      printf("FPU test program\n");
      printf("sqrt %f = %f\n",e,a);
      printf("\n");
      //coprocessor test
      //asm volatile (cpop1("0x00", "0x00", "0x01", "0x02"));
      //x = cpcos(10.258);
      printf("fpu 2 test test program\n");
      printf("exp %f = %f\n",b,x);
      printf("\n");
      return 0;
}

extern fpu_pipe();
extern fpu_chkft();

test_fpu()
{
      volatile double a, c, d;
      double e;
      extern volatile double a1,b1,c1;
      float b;
      int tmp;
```

```
        d = 3.0;
        e = d;
        a = *(double *)&a1 - *(double *)&b1;
        if (a != c1) fail(1);
        a = sqrt(e);
        if (fabs((a * a) - d) > 1E-15) fail(2);
        b = sqrt(e);
        if (fabs((b * b) - d) > 1E-7) fail(3);
        initfpreg();
        tmp = fpu_pipe();
        if (tmp) fail(tmp);
        tmp = fpu_chkft();
        if (tmp) fail(5);
//      if (((get_asr17() >> 10) & 0x3C0003) == 1) grfpu_test();
}

float f1x = -1.0;
double fmin1 = -1.0;
int fsr1[4] = { 0x80000000, 0 , 0, 0 };
double ftest[3] = {2.0, 3.0, 6.0 } ;
```