Titre: Title:	Environnement de validation en temps réel basé sur des assertions pour les systèmes matériels
Auteur: Author:	Kevin Peterson
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Peterson, K. (2005). Environnement de validation en temps réel basé sur des assertions pour les systèmes matériels [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7663/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7663/
Directeurs de recherche: Advisors:	Yvon Savaria
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

ENVIRONNEMENT DE VALIDATION EN TEMPS RÉEL BASÉ SUR DES ASSERTIONS POUR LES SYSTÈMES MATÉRIELS.

KEVIN PETERSON
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
AOÛT 2005

Your file Votre référence
ISBN: 978-0-494-16831-8
Our file Notre référence
ISBN: 978-0-494-16831-8

UNIVERSITÉ DE MONTRÉAL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ENVIRONNEMENT DE VALIDATION EN TEMPS RÉEL BASÉ SUR DES ASSERTIONS POUR LES SYSTÈMES MATÉRIELS

présenté par: PETERSON Kevin
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
A été dûment accepté par le jury d'examen constitué de :

M. KHOUAS Abdelhakim, Ph.D., président
M. SAVARIA Yvon, Ph.D., membre et directeur de recherche
M. AUDET Yves, Ph.D., membre

REMERCIEMENTS

En premier lieu, je tiens à remercier mon directeur de recherche Yvon Savaria pour m'avoir accueilli au sein du GRM et pour ses conseils et recommandations pour mon projet de recherche. Sa grande expérience m'a aidé à mieux fixer mes priorités et échéanciers pour la réalisation du projet de sonde intégrée.

Je tiens également à remercier le Conseil de recherches en sciences naturelles et génie du Canada pour leur support financier pendant les deux dernières années. Ce financement m'a permis de me concentrer principalement sur mes activités de recherche.

Il m'apparaît important de remercier mes anciens collègues de la compagnie Hyperchip, soit Martin Montpetit, Christian Mercier et Michel Forest, sans qui l'élaboration de mon sujet de recherche aurait été impossible. L'immersion dans cet environnement privilégié de test de circuits électroniques m'a fait découvrir une problématique que j'ai tenté de résoudre par mon projet.

Je m'en voudrais d'oublier l'aide apportée par Sébastien Regimbal et Jean-François Lemire à mon arrivée à l'École Polytechnique. Leurs activités de recherche m'ont fait découvrir la vérification fonctionnelle de circuits intégrés, que j'ai appliquée à la validation de circuits intégrés.

Finalement, une attention particulière va pour ma conjointe Caroline Dalpé, qui a su me soutenir tout au long de mes études de maîtrise.

RÉSUMÉ

La complexité et la vitesse des circuits numériques n'ont cessé de croître au cours des dernières années. Les langages comme le VHDL et le Verilog et les logiciels de placement et routage permettent l'élaboration en un temps relativement court de circuits complexes fonctionnant à haute vitesse. Parallèlement, des simulateurs de plus en plus puissants permettent de déceler plusieurs erreurs de conception tôt dans le processus de design. Par contre, ces outils ne peuvent complètement remplacer la fabrication et la validation d'un prototype en conditions réelles. La détection et la localisation d'erreurs dans un tel prototype se heurtent à des problèmes d'observabilité du fonctionnement interne du circuit. Il est toutefois possible de résoudre certains de ces problèmes par l'ajout d'instrumentation sur puce, aussi appelée sonde intégrée.

Ce mémoire apporte une contribution au niveau du déverminage de circuits intégrés fonctionnant en conditions réelles en proposant un environnement complet de vérification basé sur des assertions, nommé « *Assertion-Based Runtime Debugger* » (ABRD). Le développement de cet environnement comporte deux principaux aspects. En premier lieu, la conception et l'étude de l'impact de la sonde matérielle, la « *On-Chip Hardware Probe* » (OCHP), ajoutée au circuit vérifié visent à minimiser l'impact de celle-ci sur le fonctionnement normal du circuit. Il importe que la sonde permette une visibilité maximale du fonctionnement interne du circuit, tout en minimisant la bande passante utilisée sur les signaux externes et en ajoutant un minimum de capacité parasite sur les lignes échantillonnées. Un contrôle des régions couvertes par la sonde à chaque coup d'horloge ajoute une flexibilité permettant la localisation plus précise des sources d'erreur et le suivi en temps réel des opérations internes dans différents secteurs du système validé.

Le second aspect porte sur le développement d'un circuit reconfigurable, le « *Assertion-Checker FPGA* » (ACF), permettant le contrôle de la sonde, ainsi que le stockage et l'analyse en temps réel des données recueillies. L'utilisation de vérificateurs d'assertions synthétisés permet la détection et la localisation précises des erreurs de

fonctionnement du système vérifié. Les assertions, décrites en OVL (« Open Verification Library »), un langage standard développé pour la vérification sur simulateur, permettent une description simple et sans ambiguïté du fonctionnement des zones vérifiées. On peut même envisager de réutiliser les mêmes assertions ayant servi à vérifier le circuit sur simulateur, lors des étapes de conception. Le développement d'un compilateur d'assertions dédié à cet environnement de test permet également d'accélérer le processus de traduction des assertions en code VHDL adapté au circuit validé, pour programmer les changements de configuration de la sonde en temps réel et démarrer et arrêter le stockage des données de déverminage. L'implantation de la logique de vérification dans un FPGA permet quant à elle l'utilisation d'une quantité essentiellement illimitée d'assertions pour vérifier à divers niveaux de précision les composantes d'un circuit ou différents systèmes comportant le même type de sonde.

L'environnement de validation a été analysé sur différents modèles. Les performances et impacts de la sonde intégrée ont été analysés sur simulateur en utilisant un processeur Cordic. Les performances du processeur avec et sans la sonde ont été analysées lors de simulations analogiques pour vérifier la valeur et l'effet des capacités parasites induites par l'ajout du circuit de déverminage. Le circuit reconfigurable a quant à lui fait l'objet d'analyses sur simulateur numérique ainsi que sur une plate-forme de prototypage. Dans les deux cas, un circuit modélisant un bus ISA a servi d'exemple pour l'essai de l'environnement de test. Des assertions décrivant le fonctionnement complet du circuit ont été écrites et synthétisées dans un FPGA Spartan™ 3S200 de Xilinx. Un autre FPGA du même type contenait le modèle de bus ISA. Après avoir modifié le circuit ISA en changeant une de ses portes logiques, l'efficacité du système a été tout d'abord démontrée sur le simulateur ModelSim™ de Mentor Graphics, puis finalement en environnement réel sur la plateforme de prototypage, ce qui démontre la validité du concept présenté.

ABSTRACT

Digital circuits have grown in complexity and speed during the last decades. Hardware description languages such as VHDL and Verilog, as well as place and route software tools have reduced the time required to design complex high-speed circuits. At the same time, more powerful simulators allowed for early detection of design errors. But these tools cannot completely replace fabrication and validation of a prototype working in real-world conditions. Detection and localization of design errors in those prototypes are made difficult because of a lack of observability of the internal operation of the circuit. It is possible to partly solve these difficulties by adding on-chip instrumentation, also named integrated probe, to a device under validation (DUV).

This thesis brings a contribution in the field of integrated circuits debugging methods. It allows debugging them in real-world conditions by proposing a complete assertion-based verification environment, the Assertion-Based Runtime Debugger (ABRD). The development of this environment includes two main aspects. The first aspect includes both the design and impact analysis of the hardware probe: the On-Chip Hardware Probe (OCHP). This probe, added to the DUV, is designed to try to minimize the impact on the normal behavior of the circuit. It is important that the probe allows a maximum visibility of the internal operations of the circuit, while minimizing the bandwidth used on the external signals and adding a minimum parasitic capacitance to the sampled signals. Control over the zones covered by the probe in one clock cycle increases flexibility in the debugging process. This is done by allowing a more precise localization of error sources and a real-time follow-up of the internal operations in different zones of the validated system.

The second aspect of the ABRD debug environment deals with the development of a re-configurable circuit, the Assertion-Checker FPGA (ACF). The ACF is used to control the probe and to store and analyze the monitored signals in real-time. Synthesized assertion checkers permit a precise detection and localization of errors in the validated system. The assertions, written in OVL ("Open Verification Library"), a standard

language developed for verification on simulators, allow a simple and unambiguous description of the behavior of the circuit. It is even possible to consider re-using assertions developed during the design process. The translation of assertions in VHDL can be speed-up by developing a dedicated assertion compiler that will help swapping real-time configurations and capture debugging data. Implementing the debug logic in a FPGA allows the use of an essentially unlimited number of assertions to verify it at different levels of abstraction. It is also possible to use the same debug circuit to validate different systems where the OCHP is implemented.

This validation environment was analyzed on different models. The integrated probe performance and impacts were analyzed in simulation using a Cordic processor. The performances of this processor, with and without the probes, were compared with analog simulations to check the effect of the parasitic capacitance induced by the debug circuit. The assertion-checking circuit was analyzed both on a digital simulator and on prototyping boards. In both cases, an ISA bus circuit was used to validate the test environment. Assertions describing the entire system behavior have been written and synthesized in a Xilinx Spartan™ 3S200 FPGA. Another FPGA of the same type contained the ISA bus model. After modifying a logic gate in the ISA model, it was possible to confirm the debug system efficiency, first with Mentor Graphics' ModelSim™ simulator and then with the prototyping boards. This demonstrates the validity of the concept described in this document.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SIGLES ET DES ABRÉVIATIONS

ABRD : Assertion-Based Runtime Debugger

ACF : Assertion-Checker FPGA

ASIC : Application-Specific Integrated Circuit

BIST : Built-In Self-Test

DUV : Design Under Verification

FIFO : First-In First-Out

FPGA : Field Programmable Gate Array

GRM : Groupe de Recherche en Microélectronique

HDL : Hardware Description Language

ICE : In-Circuit Emulator

IP : Intellectual Property

IRQ : Interrupt ReQuest

JTAG : Joint Test Action Group

LFSR : Linear Feedback Shift Register

OCHP : On-Chip Hardware Probe

OVL : Open Verification Library

RTL : Register Transfert Level

SOC : System-On-Chip

VHDL : VLSI Hardware Description Language

VLSI : Very Large Scale Integrated Circuit

LISTE DES ANNEXES

Chapitre 1

INTRODUCTION

La validation et le déverminage de circuits intégrés et de systèmes fonctionnant en environnement réel, à des fréquences de l'ordre des centaines de mégahertz, représentent des défis de taille pour les équipes d'ingénierie. Une des difficultés majeures consiste à identifier et à localiser la source d'une erreur lors de la validation d'un prototype. En effet, les erreurs survenant à cette étape du flot de développement sont souvent intermittentes et surviennent habituellement seulement lorsque le système fonctionne à pleine vitesse [41]. L'accès en temps réel limité, voire inexistant, aux signaux internes du circuit ajoute à la complexité du problème. En effet, une erreur qui affecte une section interne du système peut prendre plusieurs cycles d'horloge avant d'apparaître aux interfaces externes [18] et le lien entre cet effet et la source de l'erreur peut être très difficile à établir. En plus, la localisation précise du problème se trouve grandement compromise par le fait que le comportement erroné ne se reproduit pas toujours sur le simulateur, en raison des différences entre le modèle simulé et le circuit réel.

Une méthodologie de déverminage efficace implique l'utilisation d'instrumentation sur puce, c'est-à-dire d'un module incorporé au circuit permettant d'accéder à ses signaux internes et à ses interfaces. L'instrumentation de base se compose d'un ou plusieurs blocs, utilisés pour collecter et accumuler des données internes sélectionnées, ce qui permet de retracer le fonctionnement du circuit dans le temps, à l'aide d'un outil de déverminage extérieur. On applique habituellement une technique de marquage temporel, pour permettre une reconstitution fidèle du flot de données par l'outil de déverminage. Des versions plus sophistiquées, mais également plus coûteuses en surface utilisée, compressent les données avant de les transférer à l'outil de déverminage [26]. Un filtre de données sélectionne les signaux devant être échantillonnés, conjointement avec des déclencheurs matériels. Ce filtre sélectionne des événements particuliers qui déclenchent le début et la fin de l'acquisition des données. Il limite le stockage requis à un intervalle

de temps défini, ce qui réduit la quantité de mémoire tampon nécessaire dans le circuit et le temps de transfert des données vers l'extérieur. Ces caractéristiques sont importantes pour s'assurer que l'on transmet seulement l'information pertinente à l'outil de vérification [41].

Il faut toutefois résoudre plusieurs difficultés pour ajouter des fonctionnalités de déverminage à un design. Évidemment, on ne peut pas connecter un analyseur logique aux signaux internes d'un circuit intégré. Il faut donc absolument utiliser des broches du circuit pour accéder à ces signaux, mais cette technique exige de faire des compromis, car les plots externes d'un circuit représentent un coût très important [3]. Il faut en même temps considérer que la détection rapide d'une erreur de fonctionnement et la localisation précise de sa source nécessitent l'accès à un grand nombre de signaux internes. Une solution idéale devra maximiser le nombre de signaux internes visibles et permettre l'envoi en temps réel de ces signaux à l'outil de déverminage, tout en utilisant un nombre restreint de plots externes. En raison du nombre limité de signaux externes disponibles simultanément, un changement rapide de la zone accédée permettra d'observer le comportement de plusieurs parties du circuit en utilisant les mêmes ressources, ce qui permet par exemple de suivre l'évolution d'un calcul ou d'une transaction dans différents modules d'un SoC.

Comme certaines erreurs de délai surviennent seulement dans une configuration de circuit particulière et à la fréquence d'horloge nominale, le comportement du système pourra différer entre un prototype comportant une sonde intégrée et un produit final ne comportant pas de sonde. Ce problème appelé « effet de sonde » trouve sa source dans les différences de routage induites par l'ajout d'un moniteur matériel à un système [10]. Il faut donc éviter les solutions impliquant l'ajout d'une sonde seulement sur un prototype du système validé, car des erreurs de délai pourraient survenir dans une nouvelle version du circuit ne comportant pas de sonde, sans aucun moyen de localiser la source de cette erreur!

Aux difficultés matérielles d'un environnement de déverminage en temps réel s'ajoutent celles liées à la configuration de l'outil et à l'interprétation des données

recueillies par l'ingénieur de test. Déterminer et programmer les événements adéquats permettant de démarrer ou d'arrêter le stockage de données de déverminage peut s'avérer difficile si le lien entre le comportement échantillonné et les données recueillies par l'instrumentation sur puce ne passe pas par un langage évolué. Plusieurs chercheurs proposent depuis une dizaine d'années des méthodes pour simplifier l'interface entre la sonde matérielle et l'usager. L'interprétation des données recueillies reste cependant ardue, car le comportement du circuit affiché sur des chronogrammes doit être comparé soit au code HDL ou à la spécification du circuit.

L'objectif du travail présenté dans ce mémoire est d'apporter une contribution au niveau des méthodes utilisées pour la validation de circuits numériques fonctionnant en temps réel et comportant une sonde intégrée. La méthode de validation et de déverminage, le « *Assertion-Based Runtime Debugger* » (ABRD), est présentée, en mettant l'emphase sur la sonde matérielle proprement dite, nommée « *On-Chip Hardware Probe* » (OCHP) et sur le circuit de déverminage externe, le « *Assertion-Checker FPGA* » (ACF), utilisant des assertions synthétisables.

La OCHP présentée consiste en un module d'instrumentation sur puce utilisant peu de ressources matérielles dans le circuit validé. Sa conception visait cinq objectifs. Premièrement, une minimisation de l'effet de sonde, en réduisant le plus possible la longueur des connexions ajoutées au circuit existant et en diminuant la charge ajoutée à chaque ligne échantillonnée. Deuxièmement, un changement rapide de la zone échantillonnée, en un coup d'horloge, pour permettre de suivre en temps réel l'évolution d'un calcul ou d'une transaction dans différentes parties du système, en utilisant les mêmes signaux externes. Troisièmement, l'utilisation d'un nombre réduit de plots pour le contrôle de la sonde et la transmission de données de déverminage au ACF. Quatrièmement, une transmission parallèle et en temps réel des données de déverminage. Les données échantillonnées à un temps donné arrivent simultanément au ACF et les données du prochain cycle d'horloge suivent immédiatement. Finalement, un protocole de communication simple et rapide est proposé pour la configuration et le contrôle de la

sonde. Ce protocole permet un nombre variable de signaux de déverminage et de contrôle selon le circuit validé, ce qui permet de réutiliser le ACF avec plusieurs designs.

Le ACF, quant à lui, sert à la fois d'unité de stockage des données de déverminage et d'analyseur du comportement du circuit vérifié, pour déceler les erreurs de fonctionnement éventuelles. L'utilisation d'un FPGA pour implanter ce circuit permet la programmation d'une quantité illimitée de séries d'assertions, pour décrire le comportement de plusieurs sections d'un même circuit, ou de la même section, mais à différents niveaux de précision. La souplesse du protocole de communication de la OCHP permet également d'utiliser le ACF pour vérifier plusieurs circuits différents, en autant qu'ils comportent un module d'instrumentation sur puce de type OCHP. Une autre particularité de ce système est l'utilisation d'assertions synthétisables pour décrire le fonctionnement du circuit vérifié et pour générer des événements déclenchant et arrêtant le stockage des données de déverminage. Les assertions permettent une description simple et sans ambiguïté du comportement d'une ou de plusieurs zones du circuit vérifié [19]. En utilisant un langage standard comme le OVL d'Accelera [2] pour les assertions, il devient possible de réutiliser des assertions définies à l'étape de la conception pour la vérification fonctionnelle sur simulateur, diminuant d'autant le temps de configuration du ABRD. Un compilateur d'assertions, comme ceux disponibles sur le marché [5][20], simplifie la création des fichiers binaires nécessaires pour programmer différentes séries d'assertions synthétisées dans un FPGA.

L'environnement de déverminage a été validé sur différents modèles. L'implantation de la sonde configurable dans un processeur a permis l'analyse de la valeur et de l'effet des capacités parasites, ainsi que de la fréquence d'opération maximale, et ce pour différents nombres de plots externes, de signaux internes échantillonnés et de configurations possibles. Un modèle de bus ISA a servi à la validation de l'environnement complet, au moyen d'assertions OVL synthétisées dans le ACF. Pour vérifier la précision et l'efficacité de la détection et de la localisation d'une erreur de fonctionnement, une méthode d'injection de mutants dans le modèle de bus ISA fut utilisée. Une fois le circuit synthétisé pour aller dans un FPGA Spartan^{TM} 3S200 de

Xilinx, une de ses portes logiques a été changée, modifiant du même coup le fonctionnement d'une partie du circuit. Une simulation de ce circuit mutant sur ModelSim™, de Mentor Graphics a permis de valider le fonctionnement du système de déverminage. L'essai sur deux cartes de prototypage, l'une contenant le ACF avec les assertions synthétisées et l'autre contenant le bus ISA mutant, a quant à lui démontré le fonctionnement du système dans un environnement réel et par le fait même validé le concept.

Ce mémoire est divisé en cinq chapitres. Le chapitre 1 introduit une vue d'ensemble des travaux présentés dans ce document. Le chapitre 2 présente une revue de littérature sur les thèmes des technologies actuelles d'instrumentation sur puce et des techniques de vérification par assertions. Le chapitre 3 introduit l'environnement de validation en temps réel basé sur des assertions développé. L'environnement en tant que tel est présenté avec des remarques critiques et les extensions possibles de la méthode. Le chapitre 4 présente l'application de l'environnement de validation sur différents exemples. L'analyse des performances de la sonde matérielle fait l'objet d'une première partie, alors que le fonctionnement du système complet est démontré sur simulateur et sur une plate-forme de prototypage en seconde partie. Finalement, le chapitre 5 conclut ce mémoire.

Chapitre 2

REVUE DE LITTÉRATURE

Ce chapitre présente une revue de littérature sur le sujet de la vérification des circuits numériques au moyen d'instrumentation sur puce et sur la vérification par assertions. En premier lieu, une comparaison des différents systèmes d'instrumentation sur puce existants sera effectuée. Ensuite, un survol des techniques de vérification par assertions est effectué, en mettant l'accent sur les langages utilisés par les outils commerciaux actuels.

2.1 Instrumentation sur puce

Au cours des dernières années, les laboratoires de recherche universitaires et l'industrie ont conçu plusieurs techniques d'instrumentation sur puce, en réponse au besoin d'outils de déverminage offrant une meilleure observabilité des comportements internes des circuits. Ces techniques se regroupent en trois grandes catégories, soit :

a) Modules d'instrumentation sur puce basés sur des chaînes de balayage ou le port Boundary Scan (JTAG);
b) Modules d'instrumentation sur puce dédiés aux systèmes sur puce ou « System-on-Chip » (SoC) comportant des processeurs embarqués;
c) Modules d'instrumentation sur puce pour les circuits implantés sur des FPGA.

Chacune de ces solutions tente de répondre à des besoins particuliers, en profitant des avantages d'un type de circuit particulier et en utilisant les ressources offertes par une implémentation, mais dans chaque cas, on peut dénoter certaines limitations à la méthode utilisée. Cette section examinera les différentes solutions développées par les équipes de recherche, en faisant ressortir leurs particularités et leurs limitations.

2.1.1 Instrumentation sur puce basée sur une chaîne de scan ou Boundary Scan

Beaucoup de modules d'instrumentation sur puce utilisent des chaînes de scan et parfois le port Boundary Scan [24] pour accéder aux signaux internes d'un circuit pour la validation. Cette méthode a le grand avantage de nécessiter un minimum de plots sur le circuit mais, selon les particularités de l'implémentation, comporte certains désavantages pour le déverminage en temps réel de circuits haute vitesse.

2.1.1.1 Modules avec contrôle sur le fonctionnement du circuit

Les laboratoires de recherche de la compagnie Philips, aux Pays-Bas [14][35][37], ont développé un système de déverminage de prototypes basé sur trois principes importants :

1. L'accès aux broches fonctionnelles du circuit à valider

   Il importe en effet d'exercer les fonctionnalités du circuit avec des stimuli réels et de récolter les réponses du circuit pour analyse. Les chercheurs proposent deux méthodes pour envoyer et recevoir les données externes : soit en connectant le circuit dans son environnement de fonctionnement prévu ou en le connectant sur un testeur de circuit, du genre utilisé pour les tests de fabrication [35].

2. L'accès aux signaux internes et aux éléments de mémoire du circuit

   Si un comportement erroné est détecté sur les interfaces externes du circuit, l'équipe de validation voudra faire un « zoom sur l'erreur » en accédant aux signaux internes du circuit, de même qu'à ses éléments de mémoire. La méthode proposée implique de rendre accessibles par le port Boundary Scan des chaînes de balayage parcourant le circuit [14].

3. Un contrôle sur le fonctionnement du circuit (i.e. : démarrage, interruption et exécution pas-à-pas de l'horloge)

   L'accès aux signaux internes du circuit à l'aide de chaînes de balayage pour localiser la source d'une erreur de fonctionnement implique un contrôle sur l'horloge du circuit. En effet, il faut plusieurs coups d'horloge pour extraire la valeur d'un signal au moyen d'une chaîne de balayage. Si le circuit continuait son fonctionnement normal pendant ce temps, il serait impossible de connaître les

valeurs prises par le signal entre le moment de l'échantillonage et le moment où se termine le transfert des données au travers de la chaîne de balayage. Il faut donc exécuter la séquence suivante pendant un nombre de coups d'horloge suffisant pour obtenir le comportement erroné : une interruption de l'horloge, suivie d'un balayage des éléments de mémoire et de l'exécution d'un autre coup d'horloge, et ainsi de suite. L'interruption de l'horloge s'effectue au moyen de moniteurs programmables, qui attendent l'occurrence d'un événement particulier dans le système pour interrompre l'horloge [37].

Cette méthode comporte deux lacunes : en premier lieu, le temps d'acquisition des données peut devenir très élevé à mesure que le nombre de signaux internes échantillonné augmente. De plus, comme certains types d'erreurs, comme les erreurs de délai notamment, surviennent seulement lorsque le circuit fonctionne à vitesse nominale, une exécution pas-à-pas passerait à côté de ce genre d'erreur.

2.1.1.2 Modules avec mémoire tampon (FIFO)

Pour éviter d'avoir à interrompre l'horloge d'un circuit pour en balayer les éléments de mémoire, une équipe de recherche de l'Université Mälarden en Suède [17] a développé un système d'instrumentation sur puce comportant une sonde avec une mémoire tampon de type FIFO.

La méthode a plusieurs similitudes avec celle présentée précédemment. L'extraction des données internes s'effectue toujours au moyen d'un lien sériel et le système comporte un moniteur d'événements internes pour le déclenchement. La différence réside dans le fait qu'au lieu d'interrompre l'horloge à la détection d'un événement, la sonde démarre plutôt le stockage de signaux présélectionnés dans une FIFO, pour extraction ultérieure par le logiciel de déverminage [17].

Cette méthode permet de détecter et localiser des erreurs de fonctionnement à vitesse nominale, contrairement à la précédente, mais a un désavantage : la FIFO ayant une taille limitée, le nombre de données de déverminage stockées l'est également. Si le temps entre le début du stockage et l'apparition de l'erreur dépasse la taille de la FIFO, il peut manquer des données pour en localiser la source.

2.1.1.3 Norme IEEE P1500

Le besoin d'accéder aux signaux d'interface des systèmes sur puce, habituellement composés de plusieurs modules de propriété intellectuelle (IP) a mené au développement de la norme IEEE P1500 [28]. Cette norme dérive de la norme IEEE 1149.1 [24], portant sur le Boundary Scan, qui sert au test des circuits imprimés. Un parallèle existe en effet entre un SoC et un circuit imprimé : dans les deux cas, des modules indépendants s'interconnectent sur un même circuit.

Le principe du JTAG s'applique donc particulièrement bien au test des SoC. Les signaux d'interface entre les IP, de même que leurs signaux internes peuvent s'accéder au moyen d'une chaîne de balayage configurable de la même façon qu'une chaîne JTAG. Il suffit d'envelopper le IP d'un « core wrapper » comportant des registres de balayage et un module de contrôle standard. Ceci permet d'accéder aux signaux pertinents de l'IP. En reliant en série les « core wrappers » de tous les modules du SoC, on a un système d'instrumentation sur puce permettant d'accéder à toutes les interfaces internes et même à certains signaux internes des IP.

Cette technique comporte plusieurs avantages. Un « core wrapper » utilisant un protocole standard peut être réutilisé dans plusieurs SoC avec un minimum d'adaptation. La souplesse de la norme permet l'ajout de fonctionnalités spécifiques à un IP, comme l'interruption de l'horloge ou le test intégré (BIST). L'instruction « bypass » accélère le transfert de données de déverminage en passant au travers de modules non-échantillonnés en un coup d'horloge. Des logiciels de déverminage peuvent servir pour valider des SoC provenant de manufacturiers différents en utilisant le même protocole de communication et le même jeu d'instructions.

Par contre, le IEEE P1500 garde les désavantages inhérents aux systèmes sériels. Il nécessite soit d'interrompre l'horloge du circuit ou de stocker les données dans une mémoire tampon.

2.1.1.4 Modules hybrides (« Core Wrapper » et accès interne)

Pour pallier aux problèmes des systèmes d'instrumentation sur puce sériels tout en en gardant les avantages, on peut utiliser des modules hybrides [36]. Ces modules

comportent une enveloppe aussi appelée un « *core wrapper* » compatible IEEE P1500, couplés à des bus de test parallèles. Pour des tests pas-à-pas, la chaîne de balayage peut servir, mais en plus, les bus de test parallèles, liés aux signaux de l'enveloppe, par un système de multiplexage programmable par une instruction P1500, permettent d'échantillonner en temps réel des signaux et de les transmettre à l'extérieur en parallèle.

Un tel système permet une observabilité simultanée de plusieurs signaux internes et leur transfert en temps réel à l'outil de déverminage. Par contre, le changement de la zone échantillonnée demande l'envoi d'une instruction P1500 à tous les modules IP du SoC, ce qui peut prendre plusieurs coups d'horloge et ne permet pas de suivre l'exécution d'une transaction dans différents secteurs du circuit.

2.1.2 Instrumentation sur puce dédiée pour SoC avec processeurs embarqués

Une classe distincte de modules d'instrumentation sur puce a fait son apparition au cours des dernières années. Il s'agit des modules dédiés aux SoC comportant des processeurs embarqués. Comme la plupart des systèmes sur puce possèdent au moins un processeur, ces modules concernent une large part des circuits intégrés actuels. Un SoC avec un processeur embarqué a quelques particularités que la sonde intégrée doit tenir en compte :

- Le processeur constitue souvent le cœur du circuit et il contrôle le fonctionnement des autres modules;
- Le SoC comprend un ou des bus de données pour interconnecter les différents modules;
- Le SoC comporte la plupart du temps des banques de mémoire intégrées, impossible à accéder directement de l'extérieur.

2.1.2.1 Émulateurs « In-Circuit » (ICE)

L'utilisation croissante de systèmes embarqués pour diverses applications a amené le développement d'émulateurs dits « *in-circuit* » (ICE) qui permettent de déverminer le code embarqué au moyen des fonctions suivantes [38]:

- Arrêt et démarrage de programme;

- Traçage de code;
- « *Breakpoints* » logiciels ou matériels;
- Inspection de registres ou adresses mémoire;
- Inspection de ports du processeur.

Les ICE fonctionnent habituellement par un port dédié du processeur (souvent le port JTAG). L'envoi d'instruction de déverminage et la collecte des réponses s'effectuent avec un logiciel qui permet de suivre l'exécution du programme à même le code source [7], soit à haut niveau ou en assembleur.

L'avènement de systèmes sur puce comportant des processeurs embarqués a mené tout naturellement à l'adoption des ICE pour le déverminage de ces derniers. Des méthodes pour accéder aux registres sans interrompre l'exécution du logiciel existent maintenant, par le biais de registres parallèles (« *shadows registers* »)[9].

Un émulateur « *in-circuit* » seul ne permet toutefois pas de trouver toutes les causes d'erreur de fonctionnement d'un SoC, car il manque un accès aux bus de données du processeur.

2.1.2.2 ICE avec moniteur de bus

Un problème d'observabilité inhérent aux circuits intégrés a modifié les techniques de déverminage des systèmes embarqués sur puce. En effet, les bus de données, autrefois accessibles par un analyseur logique directement sur le circuit imprimé, font maintenant partie d'une puce enfermée dans un boîtier [10]. D'autres architectures de déverminage, comprenant un moniteur de bus, résolvent ce problème d'observabilité [30][41]. Un moniteur de bus analyse les transactions ayant lieu sur un bus de données et effectue une opération lorsqu'une certaine transaction ou une certaine séquence de transactions prédéterminée survient. L'opération effectuée peut être l'interruption du programme, le stockage d'un état dans des registres de déverminage ou l'arrêt de l'horloge du système [30]. Le ICE peut par la suite être utilisé pour produire des données utiles pour le déverminage du circuit.

Ce genre de fonctionnalité ajoute de la visibilité sur les opérations internes du SoC, mais l'utilisation d'un ICE sériel nécessite toujours l'interruption soit du programme ou

de l'horloge du processeur pour le déverminage pas-à-pas, avec les désavantages exposés à la section 2.1.1.1.

2.1.2.3 Norme IEEE-ISTO 5001 ™

La norme Nexus 5001 Forum™, développée par le *IEEE Industry Standards and Technology Organization* (ISTO), résulte d'une tentative d'améliorer et de normaliser les diverses approches de déverminage sur puce pour les SoC comportant des processeurs embarqués. Cette norme édicte des règles de base destinées aux concepteurs de processeurs embarqués qui désirent ajouter à leur design des fonctionnalités permettant de suivre en temps réel le flot des instructions et des données, avec un impact minimal sur l'exécution du logiciel [23]. Cette norme a été conçue pour être indépendante du type de processeur et d'architecture et pour supporter des systèmes à plusieurs processeurs. La norme définit également des signaux standards d'interface et un protocole de transfert des données de déverminage [31]. Les données peuvent être transférées à l'aide d'un port JTAG standard ou d'un port auxiliaire à large bande passante [23].

La norme prévoit quatre classes d'implémentation des fonctionnalités de déverminage. Un module IEEE-ISTO 5001 de classe 1 ne comprend qu'un port JTAG pour contrôler l'exécution du programme embarqué, comme pour les ICE ordinaires. À partir de la classe 2, un port auxiliaire à large bande passante, comportant de 1 à 16 signaux parallèles permet l'envoi d'instructions et la lecture de données de déverminage. Les classes 3 et 4 permettent la lecture et l'écriture en temps réel de données sur le bus du processeur et prévoient un tampon pour emmagasiner les réponses du processeur ou les données échantillonnées [27].

Cette technique de déverminage a comme principal avantage l'indépendance du protocole de communication et des ports externes par rapport aux manufacturiers de processeurs embarqués. Le même logiciel de déverminage peut donc servir pour déverminer différentes familles de processeurs. En plus, le port auxiliaire IEEE-ISTO 5001 permet de régler une partie du problème de l'interruption du programme pour le transfert de données de déverminage, tel que décrit à la section précédente.

Par contre, ce port auxiliaire ne permet d'accéder en parallèle qu'au bus de données du processeur, et non aux autres signaux internes du circuit, rendant difficile le déverminage d'autres modules que le processeur. En plus, comme le bus auxiliaire partage sa bande passante avec les données de déverminage logicielles provenant du CPU, il peut y avoir un débordement de la mémoire tampon, si les données recueillies ne sont pas transmises assez rapidement pour vider ce dernier. Ceci peut résulter en une perte d'information.

2.1.3 Instrumentation sur puce pour FPGA

L'augmentation exponentielle du nombre de portes logiques contenues dans les FPGA actuels amène les concepteurs de circuits à utiliser de plus en plus cette technologie pour implanter des circuits propriétaires. Le coût relativement peu élevé des FPGA par rapport à un ASIC, leurs performances souvent comparables pour la plupart des applications industrielles, ainsi que la réduction du temps de mise en marché d'un nouveau produit [21] expliquent également l'engouement pour ces circuits programmables, pour les petites et moyennes productions.

Les FPGA modernes ont des particularités que les concepteurs de modules d'instrumentation sur puce ont eu tôt fait d'exploiter :

- Ils comportent la plupart du temps un port *Boundary Scan* pour leur programmation et l'accès à des registres internes [15];
- Ils possèdent plusieurs entrées/sorties pouvant se connecter à n'importe quel signal interne du circuit [15];
- Ils peuvent être reprogrammés pour changer les zones échantillonnées. Certains FPGA permettent même la reprogrammation pendant le fonctionnement du circuit;
- Les états internes de plusieurs FPGA sont directement accessibles de l'extérieur [22].

Deux types de modules d'instrumentation sur puce existent pour les FPGA. Ils diffèrent par la méthode d'échantillonnage des données de déverminage : soit en série ou en parallèle.

2.1.3.1 Échantillonnage série

La méthode la plus simple d'implantation d'une sonde intégrée dans un FPGA reste l'utilisation de l'infrastructure de programmation existante, basée sur un port *Boundary Scan* [15][21][22]. Dans une telle implémentation, des registres de déverminage accessibles par le port JTAG sont ajoutés au circuit. Ces registres permettent un accès aux signaux internes du circuit sans interférer avec le fonctionnement normal de celui-ci. Des chercheurs de l'Universidad Politécnica de Madrid, en Espagne [15], proposent deux manières d'implanter de tels registres : avec la première méthode, les signaux internes peuvent être routés à des cellules de sortie non utilisées du circuit, les rendant de ce fait directement accessibles par le JTAG, en mode « EXTEST ». La seconde méthode implique la création de modules de déverminage utilisant la logique interne du FPGA.

Cette seconde méthode a comme principal avantage que le nombre de signaux échantillonnés n'est pas limité par le nombre de broches disponibles. Par ailleurs, avec un FPGA disposant de beaucoup de ressources non utilisées, les cellules logiques peuvent servir d'éléments de mémoire tampon pour stocker les données de déverminage avant la transmission [15]. La connaissance de la structure interne du FPGA permet en plus de créer des outils de programmation et de déverminage polyvalents, qui utilisent des modèles de circuit développés en langage haut niveau comme le Java pour synthétiser le circuit, le simuler et déverminer le prototype [22]. Toutefois, comme pour les modules d'instrumentation sur puce pour ASICs décrits dans les sections 2.1.1.1 et 2.1.1.2, l'obligation de transmettre les données par lien sériel reste une limitation importante de ce genre de technique pour le déverminage en temps réel.

2.1.3.2 Échantillonnage parallèle : Xilinx Open Core™

Le fabriquant d'analyseurs logiques Agilent Technologies et le manufacturier de FPGA Xilinx ont récemment uni leur forces pour offrir une solution novatrice pour le déverminage en temps réel de FPGA [3][4]. Cette technique, nommée B4655A FPGA Dynamic Probe™, utilise des circuits spécialisés présents dans les dernières générations de FPGA Xilinx (le Xilinx Open Core™) pour échantillonner et transmettre en parallèle à un analyseur logique l'état des signaux internes d'un FPGA, et ce en temps réel et sans

latence. Les plus récents analyseurs logiques de marque Agilent supportent une application logicielle permettant de programmer les sondes de type B4655A.

Ce type de sonde permet de multiplexer jusqu'à 32 banques de 4 à 128 signaux internes sur un nombre égal de signaux externes du FPGA, sur lesquels se connecte une sonde de l'analyseur logique. Le changement de la banque de signaux échantillonnée s'effectue au moyen d'une commande envoyée par l'instrument sur le port Boundary Scan du FPGA. Dans certaines conditions, la bande passante peut être multipliée par 2 en envoyant les données sur les deux fronts de l'horloge. L'analyseur logique se charge d'extraire l'état de chaque signal et de l'afficher dans un chronogramme [3].

Cette technologie présente plusieurs avantages. Comme le Xilinx Open Core fait partie intégrante du FPGA, le module d'instrumentation sur puce n'a pas d'effet de sonde indésirable qui affecterait le fonctionnement du circuit. L'utilisation de la technique de compression des données sur deux fronts d'horloge double la bande passante de la sonde. Le changement de la zone échantillonnée s'effectue facilement par l'usager avec l'interface de l'analyseur logique [4]. Ce système a cependant quelques limitations : le changement de zone échantillonnée demande une intervention de l'usager, ce qui augmente significativement le temps de réaction. Même en automatisant le changement de zone, l'utilisation du port JTAG pour envoyer la commande empêche une modification instantanée de la configuration. Plusieurs coups d'horloge sont nécessaires avant que la nouvelle instruction prenne effet, ce qui compromet le suivi en temps réel d'une transaction dans différentes zones du circuit.

2.2 Vérification par assertions

Aucune des solutions décrites dans la section précédente ne profite des avantages offerts par les techniques de vérification basées sur des assertions. Si l'on fait abstraction de quelques outils de déverminage vérifiant des assertions hors-ligne [34] ou de vérificateurs d'assertions synthétisés directement dans le circuit vérifié [16], aucun environnement matériel de déverminage performant n'utilise des assertions complexes pour identifier et localiser des erreurs dans un prototype.

Cette section présente les principes de base et les avantages des assertions, suivis d'une présentation des techniques logicielles de vérification par assertions et d'une présentation des langages spécialisés disponibles sur le marché. Le principe d'assertions synthétisables sera ensuite expliqué, pour terminer avec la description du fonctionnement d'outils commerciaux permettant d'accélérer la simulation en utilisant des accélérateurs matériels incluant des vérificateurs d'assertions synthétisés.

2.2.1 Principes de base

Une assertion (aussi appelée « propriété ») est un énoncé sur le comportement escompté d'un design qui doit rester vrai tout au long du processus de vérification. Son rôle principal est de s'assurer de la conformité du design avec l'intention du concepteur [19]. Les assertions varient en complexité. Par exemple, on pourrait vérifier une assertion telle que : « Les signaux A et B ne doivent jamais prendre l'état bas (0) au même moment » aussi bien que des énoncés transactionnels complexes du type : « À la réception d'une commande d'écriture PCI, une commande d'écriture mémoire de type séquentielle doit être émise dans l'intervalle de 5 à 36 cycles d'horloge subséquents » [29].

Les propriétés se décomposent en trois couches distinctes [18]:

1. Couche booléenne

   La couche booléenne comprend des expressions logiques composées de variables appartenant au design. Par exemple, la couche booléenne de l'expression : « Les signaux A et B sont mutuellement exclusifs » s'écrirait de la façon suivante en Verilog :

   ```
   !(A & B)
   ```

2. Couche temporelle

   La couche temporelle exprime les relations entre les expressions booléennes dans le temps. Si on exprime l'exemple précédent de la façon suivante : « Les signaux A et B sont toujours mutuellement exclusifs », on enlève l'ambiguïté sur le moment où la propriété doit être vraie. En Verilog, on peut exprimer cette expression ainsi :

   ```
   always !(A & B)
   ```

3. Couche de vérification

   La couche de vérification définit comment la description du comportement faite par les deux autres couches doit être utilisée lors de la vérification. En d'autres mots, elle définit si la propriété doit être vérifiée par l'outil, si elle sert de contrainte à la vérification ou si elle sert à définir un événement qui détermine les temps d'acquisition d'information de couverture de test. Si on assume que la propriété doit être vérifiée par l'outil, l'expression précédente s'écrit maintenant :

```
assert always !(A & B)
```

On distingue deux types de propriétés : l'assertion, une expression devant être vérifiée dans un design et la contrainte, une condition qui limite les comportements considérés lors de la vérification [18].

Finalement, les assertions existent sous deux formes : les assertions déclaratives et les assertions procédurales. La forme déclarative existe à même la structure du design et est toujours active. Les assertions procédurales sont quant à elles vérifiées seulement quand un chemin particulier est pris dans une séquence d'événements séquentiels [18].

2.2.2 Avantages des assertions

L'utilisation d'assertions pour la vérification des circuits intégrés comporte cinq principaux avantages [19]:

1. Augmentation de l'observabilité du circuit

   Dans les bancs d'essais classiques, un stimulus propre à activer une erreur doit être suivi de stimuli servant à propager les effets de l'erreur aux interfaces du circuit, où le comportement erroné sera détecté. Les assertions imbriquées dans le code HDL améliorent l'observabilité des comportements internes du circuit, en vérifiant le comportement des modules internes au lieu de vérifier seulement les signaux aux interfaces du circuit.

2. Réduction du temps de déverminage

   Les méthodes de vérification traditionnelles ne détectent pas les erreurs directement là où elles surviennent. Cela oblige les équipes de vérification à consacrer un temps important pour retracer la source d'un comportement

inattendu, ce qui réduit d'autant le temps de mise en marché du produit. Comme les assertions augmentent l'observabilité du circuit, elles permettent de trouver les erreurs exactement quand et où elles surviennent.

3. Amélioration de l'intégration de modules de propriété intellectuelle (IP)

   Un concepteur de module IP qui fournit des assertions décrivant le comportement acceptable aux interfaces de son module permet d'améliorer l'intégration de ce dernier dans un SoC. En effet, les concepteurs du SoC ont ainsi accès à un outil pour valider que le reste du circuit interagit de la façon prévue avec le module IP. Tout comportement sortant du cadre décrit par les assertions constitue une violation du protocole d'interface avec le IP.

4. Amélioration de l'efficacité de la vérification

   Dans les bancs d'essais traditionnels, les simulations sont programmées pour vérifier des cas de test prédéfinis. Si un cas de test échoue, il est exécuté de nouveau après identification et correction du problème. Habituellement, après la correction d'une erreur, les vérificateurs arrêtent de chercher des cas particuliers associés au problème et passent au problème suivant. Les assertions, quant à elles, continuent sans cesse de vérifier un comportement donné, quel que soit le cas de test. Cela permet de détecter une même erreur, survenant dans d'autres conditions. En outre, les assertions spécifiées peuvent servir à plusieurs outils de vérification, qu'ils soient propriétaires ou commerciaux.

5. Amélioration de la communication par la documentation

   Les assertions intégrées au code HDL fournissent une excellente source de documentation sur le comportement du circuit. D'autres ingénieurs qui reprennent un design doté d'assertions ont accès à une source d'information sur la façon correcte d'interfacer celui-ci au reste du système.

Bien que les programmeurs utilisent les assertions depuis plusieurs années, celles-ci ne sont apparues que récemment dans le domaine de la vérification des circuits intégrés. À l'heure actuelle, on utilise surtout la vérification basée sur des assertions en simulation

et avec les outils de vérification formelle (qui prouvent mathématiquement des propriétés du circuit) [8].

2.2.3 Méthodes de vérification par assertions

Cette section effectue un survol des techniques logicielles de vérification par assertions en considérant les deux techniques les plus utilisées en industrie, soit la vérification d'assertions en simulation et la vérification formelle [8].

2.2.3.1 Vérification par simulations

Les techniques traditionnelles de simulation utilisent habituellement une approche de type boîte noire, qui dépend de la connaissance du comportement des entrées et des sorties du circuit. Les concepteurs partent d'une spécification haut niveau décrivant l'architecture et la fonctionnalité du circuit de même qu'une définition des entrées/sorties et la transforment en code RTL synthétisable. L'équipe de vérification part de cette même spécification pour définir des cas de test qui servent à produire des bancs d'essai cherchant à vérifier le comportement correct du circuit lorsqu'on exerce toutes ses fonctionnalités [1].

Cette approche a comme principal désavantage la difficulté grandissante à retracer la source d'une erreur de comportement détectée aux interfaces des circuits vérifiés (« Device Under Verification » ou DUV), à mesure qu'augmente leur complexité. Une approche « boîte blanche », qui permet une visibilité et un contrôle sur l'intérieur du circuit, s'avère plus efficace dans le cas de circuits complexes [1].

La vérification de type « boîte blanche » repose sur l'expression par le concepteur du comportement prévu du cœur du circuit. Des assertions intégrées au code HDL se prêtent particulièrement bien à la description des comportements permis et interdits de chaque structure du design. Plusieurs simulateurs supportent maintenant la vérification d'assertions pendant la simulation [1][5][33]. Ces assertions, placées directement dans le code source, sont vérifiées pendant la simulation, soit en permanence, dans le cas d'assertions déclaratives, ou lorsqu'un chemin particulier est pris dans le code, dans le cas d'assertions procédurales [18].

L'ajout d'assertions au code RTL augmente l'efficacité du déverminage des simulations car l'assertion va immédiatement se déclencher au moment où survient le problème et dans la zone précise où il survient [8]. L'ajout d'assertions dans un design existant ou un nouveau design selon la méthodologie décrite en [19] améliore donc réellement l'efficacité de la vérification fonctionnelle, tel que décrit dans la section 2.2.2.

2.2.3.2 Vérification formelle

Si la simulation peut déceler la présence d'une erreur, elle ne peut jamais prouver son absence. Le fait qu'une assertion n'ait jamais rapporté une violation ne prouve pas qu'elle ne puisse pas être violée, car il faudrait pour ce faire exercer toutes les combinaisons d'états possibles du système. Or, ce nombre d'états augmente de façon exponentielle à mesure que croît la complexité des designs. Les outils de vérification formelle peuvent prouver mathématiquement que, pour un circuit donné et considérant quelques hypothèses sur le comportement des signaux d'entrée, une assertion restera toujours valide [8].

Dans un système réel, les entrées ne pourront pas prendre toutes les combinaisons de valeurs possibles. Il est donc possible, en utilisant des contraintes bien définies, d'explorer tous les états possibles du système pour prouver que les assertions restent toujours vraies. La validité de la preuve dépend bien sûr de la validité des contraintes. Si les contraintes sont trop restrictives et négligent des états qui pourraient survenir aux interfaces du circuit, la preuve perd sa validité. Les contraintes imposées aux entrées d'un design deviennent donc des assertions à vérifier aux sorties du design qui s'y connecte.

Pour pallier à l'explosion du nombre d'états possibles au sein de circuits de complexité grandissante, des outils de preuve semi-formelle ont fait leur apparition. Ces logiciels combinent un simulateur et un outil de preuve formelle. Les informations de simulation intermédiaires servent de point de départ à la preuve formelle.

Les outils de vérification formelle ne peuvent pas remplacer complètement les simulateurs. Ces derniers restent des outils très puissants pour couvrir l'essentiel de la fonctionnalité d'un design et trouver la plupart des erreurs de fonctionnement. La preuve formelle sert plutôt à explorer des cas particuliers que l'analyse de couverture a identifiés

comme étant difficiles à atteindre en simulation. Ces conditions difficiles à recréer en simulation permettent de déterminer des contraintes qui ciblent les cas non couverts [8].

2.2.4 Langages spécialisés

Plusieurs sociétés travaillent présentement à développer des langages dédiés aux assertions, comme le Property Specification Language (PSL) et le Open Verification Library (OVL) [2], tous deux en voie de standardisation par Accellera. Une version améliorée du OVL se retrouve par ailleurs intégrée au langage de vérification System Verilog, développé par la compagnie Synopsys [12]. Grâce à de tels langages, on peut traduire la spécification fonctionnelle en propriétés décrivant le comportement du design, de façon expressive et sans ambiguïté [18]. Cette section tentera de décrire la structure des deux langages les plus populaires à l'heure actuelle : le OVL et le PSL [13].

2.2.4.1 Open Verification Library (OVL)

La Open Verification Library (OVL) est composée d'un groupe de moniteurs d'assertions décrits en Verilog et en VHDL, tels que définis par le comité Accelera Open Verification Library. Les moniteurs d'assertions OVL peuvent aussi bien servir à la vérification en simulation qu'aux outils de preuve formelle et semi-formelle [19]. Le Tableau 1 présente les vérificateurs d'assertions présents dans la version 2003 de la OVL [2].

Tableau 1 : Moniteurs d'assertion OVL [2]

assert_always	assert_no_underflow
assert_always_on_edge	assert_odd_parity
assert_change	assert_one_cold
assert_cycle_sequence	assert_one_hot
assert_decrement	assert_proposition
assert_delta	assert_quiescent_state
assert_even_parity	assert_range
assert_fifo_index	assert_time

assert_frame	assert_transition
assert_handshake	assert_unchange
assert_implication	assert_width
assert_increment	assert_win_change
assert_never	assert_win_unchange
assert_next	assert_window
assert_no_overflow	assert_zero_one_hot
assert_no_transition	

Cette bibliothèque a comme principal avantage qu'elle ne nécessite pas de préprocesseur pour l'utiliser, car les moniteurs d'assertions sont écrits en Verilog ou en VHDL comportemental. L'utilisateur n'a donc pas à attendre que les simulateurs supportent le nouveau standard, comme pour d'autres langages spécialisés [19]. En plus, des modifications mineures aux moniteurs d'assertions peuvent les rendre synthétisables.

Par exemple, la propriété : « Le signal B ne peut jamais devenir actif pendant les 4 cycles après que A ait été actif » s'exprime de la façon suivante en OVL :

```
assert_cycle_sequence #(0,4) A_B (clk, 'TRUE, A, {4{!B}});
```

2.2.4.2 Property Specification Language (PSL)

Le Property Specification Language (PSL), issu du langage Sugar développé par IBM, procure une manière lisible et mathématiquement précise de décrire des propriétés. Comme le OVL, le PSL peut aussi bien servir aux simulateurs qu'aux outils de preuve formelle [19]. Le PSL nécessite toutefois des simulateurs spécialisés qui supportent ce nouveau langage. Pour cette raison, les assertions PSL apparaissent souvent dans le code RTL sous forme de « pragma », qui peuvent être ignorés par les simulateurs incompatibles [13].

Le PSL est un langage beaucoup plus large et expressif que le OVL. Contrairement à ce dernier, comportant un nombre fixe de types de moniteurs d'assertions, il permet une plus grande souplesse dans la description des propriétés. Le PSL comporte différents

opérateurs et mots clés qui permettent de composer des assertions complexes. Le Tableau 2 montre les principaux opérateurs PSL, en ordre de priorité :

Tableau 2 : Opérateurs PSL en ordre de priorité [19]

Opérateurs HDL	Opérateurs Verilog ou VHDL	
Opérateur d'horloge	@	Spécifie l'expression de contrôle sur le moment d'évaluation de l'expression
Opérateurs d'expressions Sugar	;	Concaténation temporelle
	[*]	Répétitions consécutives
	[=]	Répétitions non consécutives
	[->]	Répétition goto
Opérateurs d'implication de séquence	:	Fusion de séquence
	|	Disjonction de séquence
	&	Connexion de séquence sans vérification de durée
	&&	Connexion de séquence avec vérification de durée
Opérateurs d'implication de la couche de fondation	|->	Suffixe d'implication faible
	|-> !	Suffixe d'implication forte
	|=>	Suffixe d'implication suivante faible
	|=> !	Suffixe d'implication suivante forte
Opérateurs d'occurrence de la couche de fondation	always	Doit toujours rester vraie
	never	Ne doit jamais être vraie
	eventually!	Doit être vraie à un moment indéfini dans le futur
	next next!	Doit être vraie à un moment défini du futur ou dans une plage

	next_a next_a! next_e next_e! next_event next_event_a next_event_a! next_event_e next_event_e!	de temps futur.
	within within! within!_ within_	Doit être vraie après qu'une séquence soit complétée, jusqu'à une condition de fin
	whilenot whilenot! whilenot!_ whilenot_	Doit être vraie à partir du cycle courant jusqu'à une condition de fin
Opérateurs de fin	abort	Doit être vraie, mais les obligations futures peuvent être annulées par un événement donné
	until until! until!_ until_	Doit être vraie jusqu'à un événement donné
	before before! before!_ before_	Doit être vrai à un moment avant un événement donné
Opérateurs LTL	X X! F G [U] [W]	Pareil à next Pareil à next! Pareil à eventually! Pareil à always Pareil à until! Pareil à until

Par exemple, la propriété : « Le signal B ne peut jamais devenir actif pendant les 4 cycles après que A ait été actif » s'exprime de la façon suivante en PSL :

```
assert always {A} |=> {!B [*4]} @(posedge clk);
```

2.2.5 Assertion synthétisables

La section 2.2.2 a démontré les avantages de l'utilisation d'assertions dans la vérification fonctionnelle de circuits, avant leur fabrication. De ces avantages découle un intérêt pour l'utilisation d'assertions pour la vérification de circuits implantés sur silicium. Diverses techniques profitant des avantages offerts par les assertions pour la vérification de circuits réels ont vu le jour dernièrement [5][16][20]. Les sous-sections suivantes décrivent deux techniques pertinentes au projet ABRD : les vérificateurs d'assertions intégrés au circuit et les compilateurs d'assertions.

2.2.5.1 Vérificateurs d'assertions intégrés au circuit

L'idée d'utiliser des assertions pour la vérification de prototypes sur puce a mené au développement de vérificateurs d'assertions ajoutés à même le circuit synthétisé [16]. Cette méthode génère des moniteurs d'assertions à partir des couches booléenne et temporelle de la propriété à vérifier. Pour reproduire la couche booléenne, les signaux impliqués sont insérés dans des portes logiques correspondant à l'équation booléenne de l'assertion. La couche temporelle peut être reproduite en utilisant des registres à décalage, qui impriment un retard aux signaux qui en précèdent d'autres dans l'évaluation de l'équation booléenne. Par exemple, si on veut vérifier qu'un signal B doit être actif trois coup d'horloge après le signal A, il suffit d'ajouter trois registres à décalage au signal B entrant dans un comparateur ayant comme autre entrée le signal A non registré.

Cette méthode permet la vérification de comportements internes du circuit en temps réel à partir de propriétés synthétisées. Par contre, plusieurs limitations viennent réduire l'intérêt de la méthode. En premier lieu, les vérificateurs d'assertions implantés dans le circuit ne peuvent être modifiés une fois le prototype fabriqué. L'ajout ou la suppression de propriétés s'avère donc impossible, ce qui réduit significativement la flexibilité du système. En second lieu, les vérificateurs d'assertions utilisent une surface importante sur

le circuit validé, surtout dans le cas d'assertions complexes ou pipelinées. Cette surface, à laquelle s'ajoute celle de la mécanique d'extraction des résultats des vérificateurs, serait utilisée à meilleur escient en ajoutant une sonde matérielle permettant d'extraire les signaux internes pour leur vérification à l'extérieur du circuit.

2.2.5.2 Compilateurs d'assertions

La génération de vérificateurs d'assertions synthétisés comporte certaines difficultés si l'usager utilise une méthode manuelle. Ces difficultés et l'intérêt du marché pour la vérification par assertions ont amené des compagnies concevant des outils de simulation à créer des compilateurs d'assertions [5][20], qui traduisent des propriétés exprimées dans un langage spécialisé en code HDL synthétisable, ajouté à un design vérifié. Ces vérificateurs d'assertions synthétisés servent notamment dans des plates-formes d'accélération matérielle couplées à des simulateurs. Les assertions implantées dans le design permettent ainsi de détecter en temps réel des violations de son comportement prévu. Les moniteurs peuvent être laissés dans la version finale du circuit pour des fins de test [5]. Les compilateurs d'assertions supportent les langages spécialisés les plus courants, comme le PSL et le OVL.

Ces compilateurs ont comme principal avantage la simplification du processus de synthèse des vérificateurs d'assertions. La plupart des assertions exprimées en langage spécialisé peuvent être transformées en code RTL, que l'utilisateur peut facilement ajouter au design validé. Par contre, les méthodes de vérification proposées par les concepteurs de ces outils se basent toutes sur des vérificateurs d'assertions ajoutés au circuit vérifié. Ces techniques comportent une limitation importante du fait de l'augmentation de la surface du circuit qu'elles engendrent. Un effet de sonde important peut aussi avoir lieu si le produit final ne comporte pas de vérificateurs.

Chapitre 3

ENVIRONNEMENT DE VALIDATION EN TEMPS RÉEL BASÉ SUR DES ASSERTIONS

3.1 Méthode proposée : Assertion-Based Runtime Debugger (ABRD)

La solution proposée se nomme « Assertion-Based Runtime Debugger » (ABRD). Elle utilise un module spécial ajouté au design (« On-Chip Hardware Probe » (OCHP)) qui se connecte à une carte de vérification externe composée d'un FPGA (« *Assertion Checker* FPGA » (ACF)) et d'un processeur. La Figure 1 illustre l'architecture du ABRD. Tous les signaux pouvant être échantillonnés dans le circuit intégré sont branchés sur le OCHP. Ce module est programmable et peut stocker plusieurs configurations différentes. Le ACF peut changer dynamiquement la configuration active de la OCHP, de façon à ce que seules les valeurs des signaux choisis à un temps donné se retrouvent sur le port d'entrée/sortie.

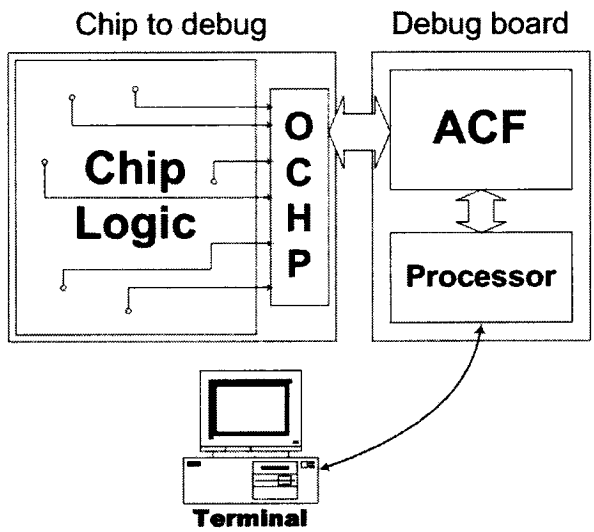

Figure 1 : Architecture de l'environnement de déverminage

Le ACF se synchronise sur l'horloge du circuit sous test (ou « Design under Test », DUT) pour mettre à jour un chronomètre, utilisé pour marquer chaque série de données recueillies. Le processeur, connecté à un terminal externe par le biais d'un lien série (UART), peut accéder au ACF pour : le programmer, changer la configuration de l'environnement de déverminage, vérifier l'état des vérificateurs d'assertions, recueillir des statistiques sur le nombre d'assertions qui passent et qui échouent et obtenir l'état des signaux du circuits dans le temps, dans le but de localiser la source d'une erreur. Le processeur a également la possibilité de vérifier des assertions plus complexes en différé, grâce aux données stockées dans la mémoire d'assertions du ACF.

3.2 Sonde intégrée : On-Chip Hardware Probe (OCHP)

Le diagramme bloc de la On-Chip Hardware Probe apparaît à la Figure 2. La OCHP consiste en trois blocs. Le Data Filter se connecte à tous les signaux susceptibles d'être échantillonnées dans le circuit intégré. Il sert à envoyer les signaux voulus à un temps donné au I/O Port. Ces signaux sont déterminés par la configuration sélectionnée dans les Configuration Registers. Ces registres contiennent plusieurs configurations différentes. La programmation des registres et le changement de configuration s'effectuent en temps réel par le biais du I/O Port.

Le I/O Port se connecte au ACF par l'intermédiaire de broches du boîtier du circuit intégré. Le nombre de broches varie selon trois facteurs dépendant du design : le nombre de signaux internes échantillonnables (N), le nombre de signaux envoyés simultanément en un coup d'horloge (P) et le nombre de configurations différentes que peut contenir la OCHP (C).

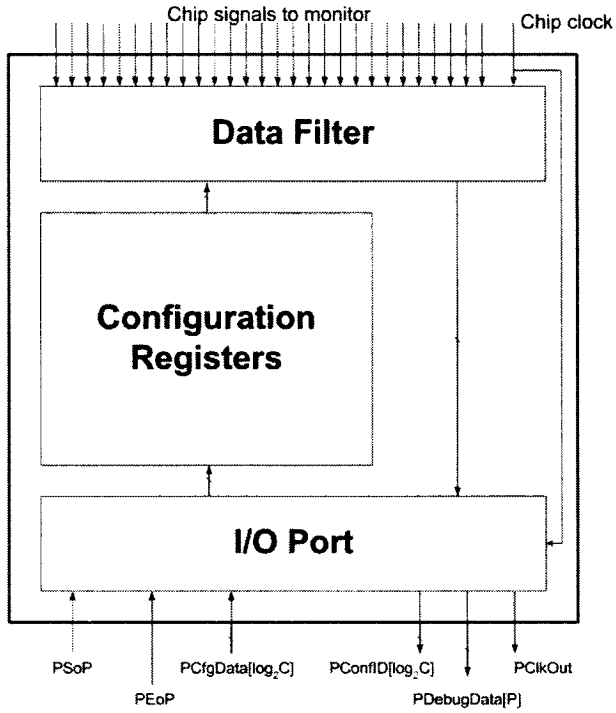

Figure 2 : Diagramme bloc de la OCHP

Les signaux d'entrée du I/O Port ont la fonctionnalité suivante : les signaux PSoP et PEoP indiquent la nature des données entrantes, envoyées via les broches nommées PCfgData. La programmation des Configuration Registers et le changement de configuration nécessitent seulement $\log_2 C$ broches de type PCfgData. Il existe en plus trois groupes de signaux de sortie. PClkOut envoie l'horloge du DUT au ACF. Les signaux PConfID (au nombre de $\log_2 C$) indiquent la configuration présentement sélectionnée dans la OCHP. Les broches PDebugData (au nombre de P) transmettent quant à eux en temps réel au ACF les signaux internes sélectionnés.

3.2.1 I/O Port

Le I/O Port comprend la logique de contrôle des Configuration Registers et connecte les différents signaux d'interface avec l'extérieur du circuit au Data Filter.

3.2.1.1 Signaux d'entrée/sortie

Le Tableau 3 présente les descriptions des différents signaux d'entrée/sortie de la OCHP avec l'extérieur.

Tableau 3: Description des signaux d'entrée/sortie de la OCHP

Signal	Type	Niveau actif	Description
Clk	Input	N/A	Horloge provenant du circuit vérifié. Les communications de la sonde avec le ACF sont synchronisées sur cette horloge. Cette horloge est envoyée au ACF par le signal PClkOut.
Reset_n	Input	0	Signal d'initialisation de la OCHP. Un niveau actif sur ce signal efface le contenu des Configuration Registers et remet la configuration courante à 0.
PClkOut	Output	N/A	Horloge envoyée au ACF pour synchroniser le *Debug Port Interface*, le *Timestamp Timer*, les *Assertion Checkers*, le *Event Generator*, la *Assertion RAM* et le *Configuration Manager*.
PintSignal[N]	Input	N/A	Signaux internes du circuit, accessibles par la OCHP. Ces signaux sont envoyés au ACF selon la configuration des Configuration Registers sélectionnée par le registre de configuration courante.
PDebugData[P]	Output	N/A	Bus de données de largeur P bits. Les données collectées par la OCHP sont transmises au ACF par ce bus.
PConfID[$\log_2$C]	Output	N/A	Bus de largeur $\log_2$C bits indiquant la configuration présentement activée dans la OCHP.
PSoP	Input	1	Signal indiquant le début d'un paquet de configuration de la OCHP ou, lorsque actif en même temps que PEoP, un changement de configuration active.
PEoP	Input	1	Signal indiquant la fin d'un paquet de configuration de la OCHP ou, lorsque actif en même temps que PSoP, un changement de configuration active.
PCfgData[$\log_2$C]	Input	N/A	Bus de largeur $\log_2$C bits servant à transmettre les paquets de configuration de la OCHP ou le numéro de la configuration sélectionnée.

3.2.1.2 Protocole de communication

Les bus de sortie PDebugData et PConfID envoient leurs données à chaque coup d'horloge, selon la configuration sélectionnée. Ils ne doivent donc suivre aucun protocole de communication particulier.

Le protocole de communication de la OCHP permet deux genres de transactions, qui utilisent les signaux d'entrée du I/O Port (PSoP, PEoP et PCfgData) : le chargement des paramètres de configuration de la OCHP et le changement de configuration.

a) Chargement des paramètres de configuration

Pour charger les paramètres de configuration de la OCHP, les données doivent être sérialisées pour former des paquets. Un paquet doit avoir le format suivant :

PClkIn	1	2	...	log2N + 1	log2N + 2	...	P*log2N + 1
PCfgData[log_2C-1]	Conf #	X	...	X	X	...	X
PCfgData[log_2C-2]		X	...	X	X	...	X
...		X	...	X	X	...	X
PCfgData[0]		MSB Pin P-1	...	LSB Pin P-1	MSB Pin P-2	...	LSB Pin 0
PSoP	1	0	...	0	0	...	0
PEoP	0	0	...	0	0	...	1

Figure 3 : Format d'un paquet de configuration

« Pin P-1 » correspond au bit le plus significatif du port PDebugData, alors que « Pin 0 » correspond au bit le moins significatif du même port.

Les signaux doivent respecter les contraintes suivantes :

- Les signaux sont échantillonnés au front montant de PClkOut.
- Les signaux PSoP et PEoP ne doivent être actifs que pendant un coup d'horloge.
- Aucune interruption ne peut avoir lieu pendant un paquet. Il doit y avoir une nouvelle donnée à chaque coup d'horloge.
- Le premier mot du paquet constitue le numéro de la configuration programmée. Le signal PSoP doit être à 1 lorsque ce mot est envoyé sur le bus PCfgData.
- Les mots suivants indiquent le numéro du signal interne connecté à un signal PDebugData. Chaque signal interne connecté à la sonde se voit attribuer un numéro d'identification. Le bit le plus significatif du numéro d'identification du

signal interne est envoyé en premier, et le bit le plus significatif du port PDebugData est le premier programmé.

- Le dernier mot du paquet est le bit le moins significatif du numéro d'identification du signal interne connecté au bit le moins significatif du port PDebugData. Le signal PEoP doit être à 1 lorsque ce mot est envoyé sur le bus PCfgData.
- Les mots de configuration de tous les signaux PDebugData doivent être envoyés, l'un après l'autre, du bit le plus significatif au bit le moins significatif. Il n'est pas possible de sauter un ou des mots de configuration.
- Les bits les plus significatifs du bus PCfgData ont une valeur de 0 du second au dernier mot d'un paquet. Seul le bit le moins significatif est lu par la OCHP.

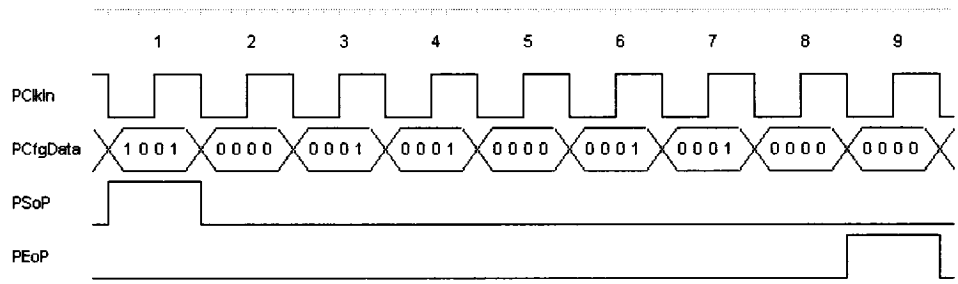

Figure 4 : Exemple d'envoi d'un paquet de configuration à la OCHP

La Figure 4 montre un exemple de l'envoi d'un paquet de configuration à une OCHP comportant 16 configuration différentes (d'où le bus PCfgData de 4 bits), 16 signaux internes (d'où les 4 bits de configuration pour chaque broche PDebugData) et 2 broches PDebugData. Les nombres C=16, N=16 et P=2 de cet exemple n'ont aucun lien entre eux.

Pour la configuration #9 (0b1001), la broche PDebugData[1] est connectée au signal interne #6 (0b0110) tandis que la broche PDebugData[0] est connectée au signal interne #12 (0b1100).

b) Changement de configuration

Pour changer dynamiquement la configuration sélectionnée dans la OCHP, il suffit d'indiquer le numéro de la configuration souhaitée sur le bus PCfgData en maintenant les signaux PSoP et PEoP à 1 en même temps.

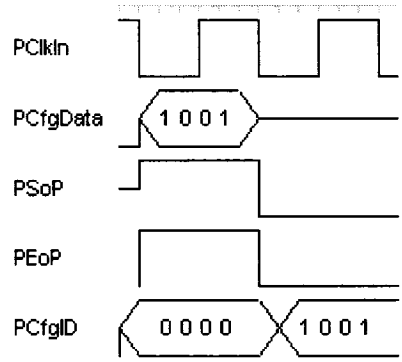

Figure 5 : Exemple de changement de configuration de la OCHP

La Figure 5 montre un exemple de changement de configuration. La configuration initiale (0b0000) est changée pour la configuration #9 (0b1001). On remarque l'effet du changement sur le signal PConfID.

3.2.2 Configuration Registers

Les registres de configuration (« Configuration Registers ») servent à stocker un nombre C de configurations différentes des signaux internes envoyés au ACF par l'intermédiaire du bus PDebugData. Chaque bit du port PDebugData se voit attribuer $C*\log_2 N$ bits de configuration qui identifient, pour une des C configurations actuellement sélectionnée, lequel des N signaux internes échantillonnables est envoyé sur la broche. Pour permettre une programmation sérielle des configurations, les bits de registre d'une configuration donnée sont mis en série, du bit le plus significatif de la broche PDebugData de poids plus élevé jusqu'au bit le moins significatif de la broche PDebugData de poids moins élevé. La configuration présentement sélectionnée connecte ces registres de configuration aux circuits de multiplexage du Data Filter, qui déterminent les signaux internes envoyés au ACF par l'intermédiaire des signaux PDebugData.

3.2.3 Data Filter

Le Data Filter contient le circuit de multiplexage des signaux internes échantillonnés jusqu'aux signaux PDebugData. Pour réduire la taille occupée sur le circuit, il est préférable d'utiliser un multiplexage à logique trois états, composé de cellules telles qu'illustrées à la Figure 6. On y remarque un décodeur d'adresse qui permet d'activer un tampon à trois états permettant de connecter le signal à échantillonner sur un bit du port PDebugData.

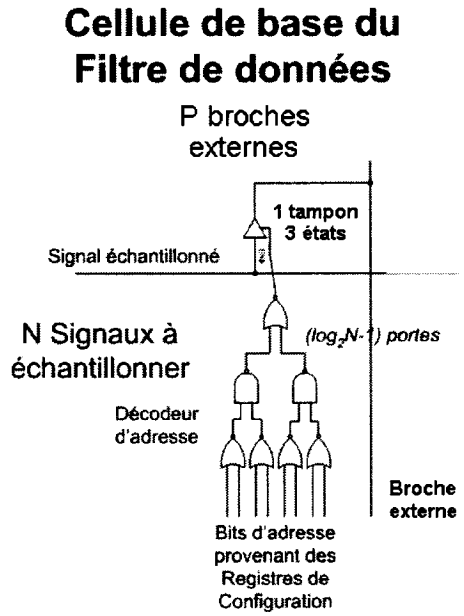

Figure 6 : Cellule de base du Data Filter

3.3 FPGA vérificateur d'assertions : Assertion-Checker FPGA (ACF)

La Figure 7 montre le diagramme bloc du ACF. Cette unité consiste en sept blocs. Le *Debug Port Interface* se connecte au I/O Port de la OCHP. Elle reçoit en temps réel les données collectées par la OCHP et lui transmet des données de configuration lorsque nécessaire. Le *Timestamp Timer* se synchronise sur le signal d'horloge provenant de la OCHP et ajoute les informations temporelles aux données stockées dans la *Assertion RAM*. Le *Event Generator* (composé de nEG moniteurs d'événements) utilise les données provenant de la OCHP pour démarrer et arrêter les *Assertion Checkers* (au nombre de

nAC). Ce module peut aussi changer en tout temps la configuration de la OCHP et contrôle un signal d'interruption (IRQ) connecté au processeur de la carte de vérification. Les *Assertion Checkers* peuvent stocker des résultats identifiés par rapport au temps par le *Timestamp Timer*, dans la *Assertion RAM*, pour consultation par un processeur hôte.

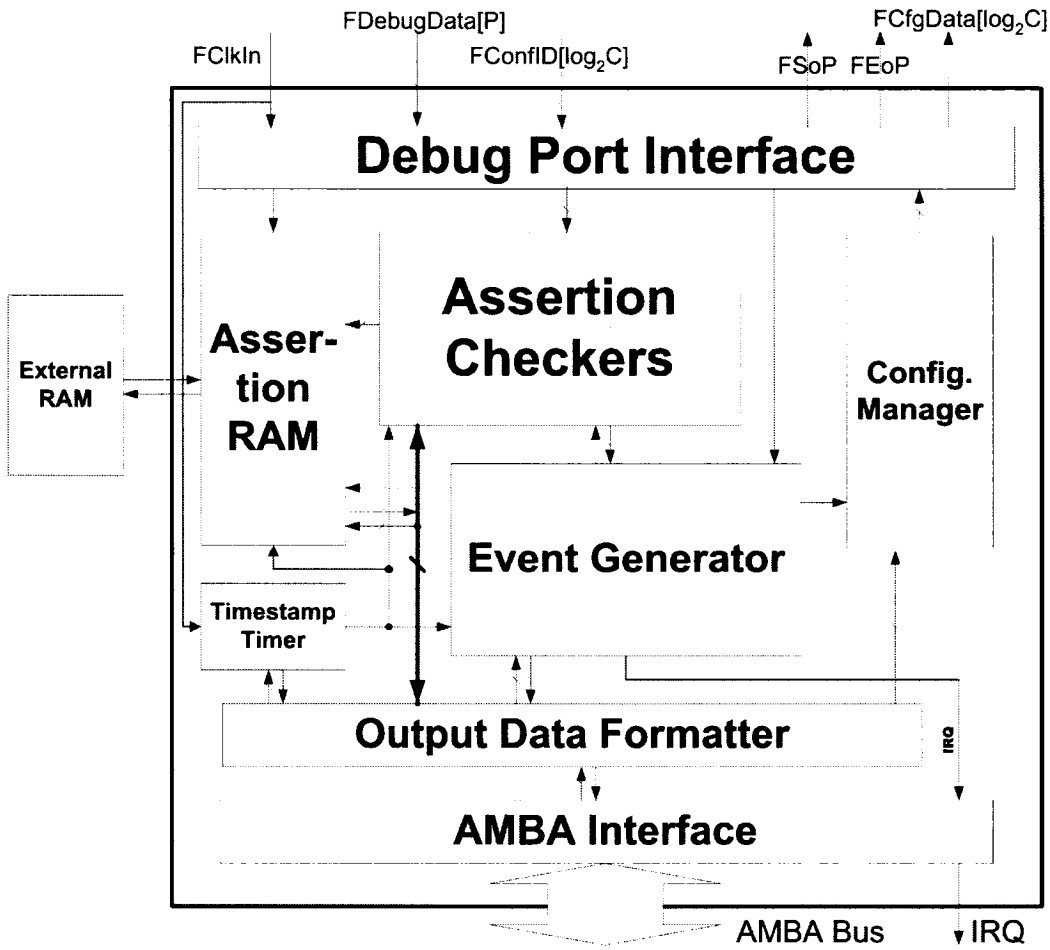

Figure 7 : Diagramme bloc du ACF

Le *Output Data Formatter* a pour rôle de bâtir des réponses aux requêtes du processeur, concernant l'état des *Assertion Checkers* ou du *Event Generator* ou les données stockées dans la *Assertion RAM*. Les requêtes et les réponses transitent via le

AMBA Interface. Le processeur a finalement la capacité de configurer le ACF, qui à son tour configure la OCHP, avec l'aide du *Configuration Manager*.

3.3.1 Fonctionnement du Assertion-Checker FPGA (ACF)

Cette section décrit le fonctionnement des différents modules composant le ACF. Notons que le nombre de signaux internes échantillonnables dans le circuit testé (N), le nombre de signaux envoyés simultanément en un coup d'horloge (P) et le nombre de configurations différentes que peut contenir la OCHP du circuit testé (C), ainsi que le nombre d'*Assertion Checkers* (nAC) et le nombre de moniteur d'événements du *Event Generator* (nEG) sont déterminés à la création du code RTL du ACF, d'après les paramètres propres au circuit vérifié. Le codeur devra donc concevoir du code VHDL permettant un changement de ces paramètres au moment de la compilation.

3.3.1.1 Signaux d'entrée/sortie

Le Tableau 4 présente les descriptions des différents signaux d'entrée/sortie du ACF avec l'extérieur.

Tableau 4: Description des signaux d'entrée/sortie du ACF

Signal	Type	Niveau actif	Description
FClkIn	Input	N/A	Horloge provenant de la OCHP du circuit vérifié. Les signaux de sortie du *Debug Port Interface*, le *Timestamp Timer*, les *Assertion Checkers*, le *Event Generator*, la *Assertion RAM* et le *Configuration Manager* sont synchronisés sur cette horloge.
FDebugData[P]	Input	N/A	Bus de données de largeur P bits. Les données collectées par la OCHP sont transmises au ACF par ce bus.
FConfID[$\log_2$C]	Input	N/A	Bus de largeur $\log_2$C bits indiquant la configuration présentement activée dans la OCHP.
FSoP	Output	1	Signal indiquant le début d'un paquet de configuration de la OCHP ou, lorsque actif en même temps que FEoP, un changement de configuration active.

FEoP	Output	1	Signal indiquant la fin d'un paquet de configuration de la OCHP ou, lorsque actif en même temps que FSoP, un changement de configuration active.
FCfgData[$\log_2$C]	Output	N/A	Bus de largeur $\log_2$C bits servant à transmettre les paquets de configuration de la OCHP ou le numéro de la configuration sélectionnée.
FRAMClk	Output	N/A	Horloge de la *Assertion RAM* externe. Ce signal est branché directement au signal FClkIn.
FRAMAddr[28]	Output	N/A	Bus d'adresse de la *Assertion RAM* externe.
FRAMData[32]	I/O	N/A	Bus de données bidirectionnel de la *Assertion RAM* externe. Les données sont présentées en même temps que l'adresse lors d'une écriture et sont lues au coup d'horloge suivant lors d'une lecture.
FRAMWE	Output	1	Signal indiquant qu'on écrit à un emplacement mémoire de la *Assertion RAM* externe. Ce signal doit être à 0 pour une lecture.
FRAMOE	Output	1	Signal indiquant qu'on lit un emplacement mémoire de la *Assertion RAM* externe. Ce signal doit être à 0 pour une écriture.
FHCLK	Input	N/A	Horloge provenant du processeur. Cette horloge synchronise les transferts sur le bus AMBA. Tous les signaux du bus sont échantillonnés au front montant de FHCLK.
FHRESETn	Input	0	Ce signal de reset, actif bas, sert à réinitialiser le ACF et le bus AMBA.
FHADDR[31:0]	Input	N/A	Bus d'adressage du bus AMBA. Les plages d'adresses réservées pour le ACF sont définies à la section 3.3.2.
FHTRANS[1:0]	Input	N/A	Indication du type de transfert ayant lieu sur le bus AMBA. Le ACF supporte seulement les types IDLE et NONSEQUENTIAL (voir [6]).
FHWRITE	Input	1	Signal indiquant la direction du transfert sur le bus AMBA. Un 1 indique une écriture et un 0 indique une lecture.
FHSIZE[2:0]	Input	N/A	Indication de la taille du transfert sur le bus AMBA. Le ACF supporte seulement les transferts de 32 bits (word, voir [6]).
FHBURST[2:0]	Input	N/A	Indication que le transfert fait partie d'un « burst » sur le bus AMBA. Non supporté par le ACF.
FHPROT[3:0]	Input	N/A	Indication du niveau de protection du transfert sur le bus AMBA. Non supporté par le ACF.

FHWDATA[31:0]	Input	N/A	Bus de données provenant du processeur vers le ACF sur le bus AMBA. Les données écrites par le processeur passent par ce bus.
FHSELx	Input	1	Signal de sélection du ACF sur le bus AMBA. Un transfert démarre lorsque ce signal est à 1.
FHRDATA[31:0]	Output	N/A	Bus de données envoyées au processeur par le ACF sur le bus AMBA. Les données lues par le processeur passent par ce bus.
FHREADY	Output	1	Signal indiquant que le ACF a terminé un transfert sur le bus AMBA. Ce signal peut être tenu à 0 pour étendre un transfert (si on attend une réponse du ACF).
FHRESP[1:0]	Output	N/A	Indication envoyée par le ACF sur le statut d'un transfert sur le bus AMBA. Le ACF supporte seulement les messages OK et ERROR (voir [6]).
FIRQ	Output	1	Signal d'interruption allant au processeur. Ce signal provient du *Event Generator*.

3.3.1.2 Debug Port Interface

Le *Debug Port Interface* consiste en la jonction des signaux d'entrée FClkIn, FDebugData, FConfID avec le *Event Generator*, les *Assertion Checkers*, le *Timestamp Timer* et la *Assertion RAM* et des signaux de sortie FSoP, FEoP et FCfgData avec le *Configuration Manager*. La gestion des protocoles de communications utilisés par ces signaux est effectuée par les modules internes auxquels ils sont reliés.

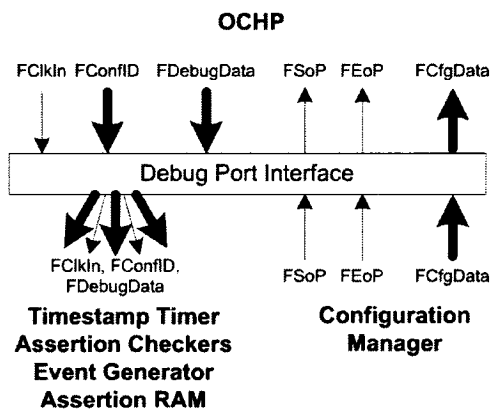

Figure 8 : Schéma de principe du Debug Port Interface

3.3.1.3 Timestamp Timer

Le *Timestamp Timer* sert au marquage temporel des données de déverminage et consiste en un compteur de 8 à 32 bits (nombre de bits déterminé à la compilation du code VHDL) qui s'incrémente de 1 à chaque front montant de FClkIn et reboucle à 0 lorsqu'il atteint une valeur maximale prédéterminée. Le compteur a un signal de remise à zéro synchrone, relié à un signal interne accessible par le registre de contrôle du ACF (ACF Control Register, section 3.3.2.1).

Le compte maximal du compteur avant le rebouclage à zéro est configurable par l'usager, via le registre de valeur maximale du *Timestamp Timer* (*Timestamp Timer* Maximum Count Register, section 3.3.2.2). L'écriture d'une valeur dans ce registre remet le compteur à zéro. Lorsque la valeur 0 se trouve dans le registre *Timestamp Timer* Maximum Count, le compteur du *Timestamp Timer* reste toujours à 0.

La valeur du temps courant calculée par le *Timestamp Timer* est lisible en tout temps, via le registre de valeur du temps courant du *Timestamp Timer* (*Timestamp Timer* Current Time, section 3.3.2.2). La lecture de cette valeur ne doit pas affecter le décompte.

La Figure 9 montre un exemple de fonctionnement du *Timestamp Timer*, pour une valeur de 0x07 dans le registre de maximum du *Timestamp Timer*. Un reset synchrone survient au début du chronogramme, suivi par le décompte de 0x00 à 0x07 et le rebouclage à 0x00.

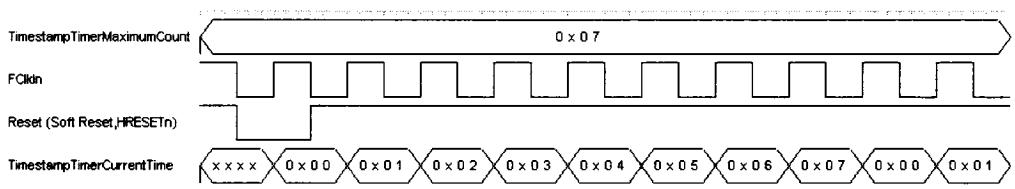

Figure 9 : Exemple de fonctionnement du TimestampTimer

3.3.1.4 Assertion Checkers

Les *Assertion Checkers* constituent le cœur du ACF. Ils proviennent d'assertions, écrites dans un langage spécialisé tel que le PSL ou le OVL, et compilées pour être

synthétisées dans le FPGA. Ces vérificateurs d'assertions valident en temps réel le fonctionnement de la zone du circuit présentement échantillonnée. Les moniteurs d'événements du *Event Generator* gèrent l'activation des *Assertion Checkers*. L'état des vérificateurs d'assertions peut également être retourné au *Event Generator* pour activer ou arrêter le stockage des données de déverminage par la *Assertion RAM*, ou changer la zone d'échantillonage du circuit. Il est également possible de stocker l'état de ces vérificateurs d'assertions dans la *Assertion RAM* pour consultation ultérieure.

Sur le plan architectural, les *Assertion Checkers* sont vus comme un nombre variable (nAC, nombre déterminé à la compilation, mais ne pouvant excéder 256 pour des raisons d'adressage) de blocs logiques séquentiels comportant deux signaux de contrôle (FClkIn et un signal de reset distinct pour chaque bloc), les signaux entrants du *Debug Port Interface* (bus FDebugData et FConfID), un nombre variable de signaux allant vers d'autres *Assertion Checkers* et/ou vers le *Event Generator* et deux registres accessibles en lecture seule (un registre d'état du *Assertion Checker* (1 bit) et un registre optionnel de Timestamp/Failure Count (0-32 bits)).

La remise à zéro d'un bloc d'*Assertion Checker* se fait par une écriture dans le registre de remise à zéro des *Assertion Checkers* (AC Reset, section 3.3.2.3), par la commande Soft Reset (section 3.3.2.1) ou par un niveau actif du signal FHRESETn. Cette écriture provoque un niveau actif sur le signal de reset pendant un coup d'horloge.

Le registre d'état d'un bloc d'*Assertion Checker* est accessible par la lecture du registre d'état des *Assertion Checkers* (AC State, section 3.3.2.3).

Le registre optionnel Timestamp/Failure count est accessible via la lecture du registre de Timestamp (AC Timestamp/Failure Counter, section 3.3.2.3).

3.3.1.5 Event Generator

Les moniteurs d'événements du *Event Generator* ont trois fonctions principales. En premier lieu, ils gèrent l'activation des *Assertion Checkers*, selon l'état du circuit donné par les signaux de déverminage et les vérificateurs d'assertions. Ces données servent en second lieu au *Event Generator* pour déterminer quelle zone du circuit doit être échantillonnée à un temps donné. Selon ces informations, le *Event Generator* envoie le

numéro de configuration à sélectionner au *Configuration Manager*. L'état des moniteurs d'événements sert troisièmement à démarrer et à arrêter le stockage des données de déverminage dans la *Assertion RAM*. En plus, de façon optionnelle, un moniteur d'événements peut être connecté à un signal externe d'IRQ.

Sur le plan architectural, les moniteurs d'événements du *Event Generator* sont vus comme un nombre variable (nEG, nombre déterminé à la compilation, mais ne pouvant excéder 256) de blocs logiques séquentiels comportant deux signaux de contrôle (FClkIn et un signal de reset distinct pour chaque bloc), les signaux entrants du *Debug Port Interface* (bus FDebugData et FConfID), un nombre variable de signaux allant vers d'autres moniteurs d'événements et/ou *Assertion Checkers* et deux registres accessibles en lecture seule (un registre d'état du moniteur d'événements (1 bit) et un registre optionnel de Timestamp/Counter (0-32 bits)) et un signal de sortie allant vers la *Assertion RAM* et le multiplexeur d'IRQ.

La remise à zéro d'un bloc moniteur d'événements se fait par une écriture dans le registre de remise à zéro des *Event Generator*s (EG Reset, section 3.3.2.4), par la commande Soft Reset (section 3.3.2.1) ou par un niveau actif du signal FHRESETn. Cette écriture provoque un niveau actif sur le signal de reset pendant un coup d'horloge.

Le registre d'état du moniteur d'événements est accessible par la lecture d'un registre d'état des *Event Generator*s (EG State, section 3.3.2.4).

Le registre optionnel Timestamp/Counter est accessible via la lecture du registre de Timestamp (EG Timestamp/Counter, section 3.3.2.4).

Les signaux de sortie des moniteurs d'événements sont multiplexés vers le signal FIRQ, selon la valeur inscrite au registre d'IRQ du *Event Generator* (EG IRQ Pin Event Number, section 3.3.2.4), si le bit No IRQ de ce registre est à 0. Le signal FIRQ prend la valeur 1 au front montant du signal de sortie du moniteur sélectionné, et est remis à zéro par une écriture d'un 1 au bit correspondant du registre de contrôle du ACF (ACF Control Register, section 3.3.2.1).

3.3.1.6 Configuration Manager

Le *Configuration Manager* sert à stocker les différentes configurations de la OCHP envoyées par le processeur jusqu'à leur écriture dans la OCHP. Ce module gère également les changements de configuration active, conjointement avec le *Event Generator.*

L'écriture des mots de configuration des différentes broches FDebugData dans la mémoire interne du *Configuration Manager* s'effectue en écrivant un mot de 32 bits à une des adresses dédiées à la Configuration RAM sur le bus AMBA (voir section 3.3.2.5). Chaque configuration a une adresse réservée sur le bus AMBA. L'identification de la broche s'effectue par les 8 bits les plus significatifs du mot écrit, alors que les 24 bits les moins significatifs identifient le signal interne connecté à la pin. L'information inscrite dans la mémoire interne du *Configuration Manager* (Configuration RAM) consiste en fait en les $\log_2 N$ bits les moins significatifs du mot écrit sur le bus AMBA. L'adresse à laquelle cette information est stockée dans la Configuration RAM correspond à l'équation :

$$(AdresseAMBA - 0x00000100) * 2 * \log_2 P + Valeur\ des\ \log_2 P\ MSB$$

Le registre Configuration Stored (section 3.3.2.5) identifie les configurations pour lesquelles on a stocké au moins un mot de configuration dans la Configuration RAM. Le *Configuration Manager* consulte ce registre pour n'envoyer que les configurations stockées dans la RAM lors d'une commande Dump cfg to OCHP (section 3.3.2.1). L'envoi de cette commande démarre l'envoi de paquets de configuration à la OCHP, selon le protocole défini à la section 3.3.3.

La Figure 10 montre des exemples de mots de configurations et des valeurs correspondantes inscrites dans la Configuration RAM et dans le registre Configuration stored, pour un nombre de configurations possibles de 16, un nombre de signaux FDebugData de 2 et un nombre de signaux internes échantillonables de 16. L'écriture à l'adresse AMBA 0x00000101 permet de stocker les mots de la configuration #1. Le premier stockage associe le signal interne #4 à la broche FDebugData[0], alors que le second stockage associe le signal interne #15 à la broche FDebugData[1]. Les troisième

et quatrième stockages concernent la configuration #9 et associent respectivement les signaux internes #12 et #6 aux broches FDebugData[0] et FDebugData[1].

C = 16, N = 16, P = 2

# Écriture	Avant	1	2	3	4
Configuration Stored	0x0000	0x0002	0x0002	0x0202	0x0202
AMBA Write Address	x	0x00000101	0x00000101	0x00000109	0x00000109
AMBA Write Data	x	0x00000004	0x0100000F	0x0000000C	0x01000006
Conf. RAM Address	x	0x002	0x003	0x014	0x015
Conf. RAM Data	x	0x4	0xF	0xC	0x6

Figure 10 : Exemple de mots de configuration pour C=16, N=16 et P=2

La Figure 11 montre un exemple d'envoi des mots de configuration de la Figure 10 à la OCHP, provoqué par la commande Dump cfg to OCHP. Comme seuls les bits correspondant aux configurations #1 et #9 sont à 1 dans le registre Configuration stored, le *Configuration Manager* n'envoie que les paquets correspondant à ces deux configurations. Dans le haut de la Figure 11, on remarque le premier paquet, destiné à la configuration #1 (premier mot du paquet avec la valeur 0b0001) et programmant la broche FDebugData[1] (bit le plus significatif du port). Cette broche est programmée avec la valeur 0xF (seul le bit le moins significatif du bus FCfgData sert à programmer les mots de configuration, ce qui donne les mots 0b0001, 0b0001, 0b0001 et 0b0001, provenant de l'écriture #2 de la Figure 10). Cette configuration est suivie de la programmation du bit le moins significatif du port, qui prend la valeur 0x4 (mots 0b0000, 0b0001, 0b0000 et 0b0000, provenant de l'écriture #1 de la Figure 10). Le second paquet, destiné à la configuration #9 (premier mot du paquet suivant à 0b1001, dans le bas de la figure), suit immédiatement. L'ordre d'envoi des paquets n'a pas d'importance, en autant que tous les bits du port FDebugData soient programmés pour une configuration donnée.

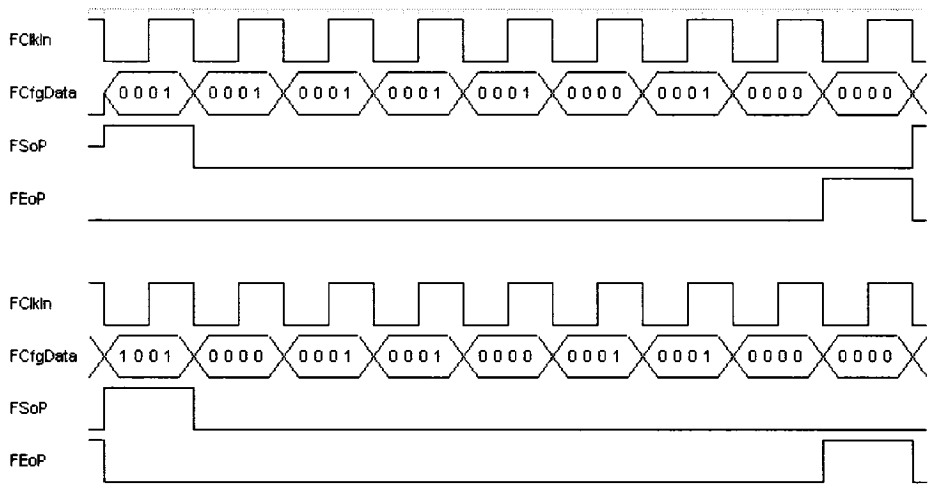

Figure 11 : Exemple de Dump Configuration pour C=16, N=16 et P=2

Le changement de configuration active s'effectue quant à lui sur demande du *Event Generator* (les signaux utilisés dépendent des moniteurs d'événements implantés). Ce changement de configuration s'effectue selon le protocole défini à la section 3.3.3.

3.3.1.7 Assertion RAM

La *Assertion RAM* se comporte comme l'unité de stockage d'un oscilloscope ou d'un analyseur logique. Elle comprend une mémoire externe synchronisée sur FClkIn et comportant un port de données bidirectionnel. La section 3.3.3 décrit le protocole de transfert des données à cette mémoire. Jusqu'à 32 signaux provenant aussi bien des bus FDebugData et FConfID que des *Assertion Checkers* ou du *Event Generator* peuvent être routés à la *Assertion RAM* pour stockage dans la mémoire externe. Ces signaux sont déterminés à la compilation.

Deux compteurs permettent de gérer respectivement l'incrémentation des adresses de stockage des données (compteur AssertionRAMAddrCounter) et le nombre de données restant à stocker (compteur AssertionRAMSizeCounter).

Le compteur AssertionRAMAddrCounter est initialisé au début d'un stockage et incrémente jusqu'à la valeur inscrite dans le registre RAM Storage Size (section 3.3.2.6) moins 1, puis reboucle à zéro si le stockage n'est pas terminé.

Le compteur AssertionRAMSizeCounter quant à lui, est initialisé à la valeur inscrite dans le registre RAM Storage Size (section 3.3.2.6) moins 1 en mode normal (bit Single Shot à 0, voir section 3.3.2.6) ou à la valeur inscrite dans le registre RAM Trigger Delay (section 3.3.2.6) en mode Single Shot. Ce compteur décrémente ensuite jusqu'à 0 et le stockage cesse à ce moment. Si une valeur de 0 apparaît dans le registre RAM Storage Size, la *Assertion RAM* est désactivée.

Il existe plusieurs modes d'activation du signal de déclenchement (« trigger »), déterminés par le registre RAM Trigger type (section 3.3.2.6). Dans tous les modes, le signal de déclenchement doit préalablement être armé avec la commande Arm Trigger (section 3.3.2.1) pour pouvoir être déclenché. Une fois le stockage démarré, à chaque coup d'horloge, le compteur AssertionRAMAddrCounter est incrémenté. La valeur courante du *Timestamp Timer* est mise dans le registre RAM End of Storage Timestamp (section 3.3.2.6), alors que l'adresse de stockage courante est mise dans le registre RAM End of Storage Pointer (section 3.3.2.6).

En mode normal (Manual Trigger = 0; Store until Event = 0 et Single Shot = 0), une fois le signal de déclenchement armé, à chaque front montant du signal d'événement du moniteur d'événement identifié par le registre RAM Trigger Event Number (section 3.3.2.6), le signal de déclenchement démarre le stockage des données. Le compteur AssertionRAMAddrCounter est initialisé à 0, le compteur AssertionRAMSizeCounter est mis à la valeur inscrite dans le registre RAM Storage Size (section 3.3.2.6) moins 1 et la valeur courante du *Timestamp Timer* est mise dans le registre RAM Trigger Timestamp (section 3.3.2.6). À chaque coup d'horloge, le compteur AssertionRAMSizeCounter est décrémenté. Lorsque ce dernier atteint 0, la *Assertion RAM* enregistre une dernière valeur puis le stockage cesse. Le signal de déclenchement reste néanmoins armé.

La Figure 12 montre un exemple de fonctionnement en mode normal. Le signal de déclenchement est armé au point a, un événement active le signal de déclenchement au point b, le stockage cesse au point d et un autre événement re-déclenche le signal de déclenchement au point e, ce qui écrase les données précédemment stockées.

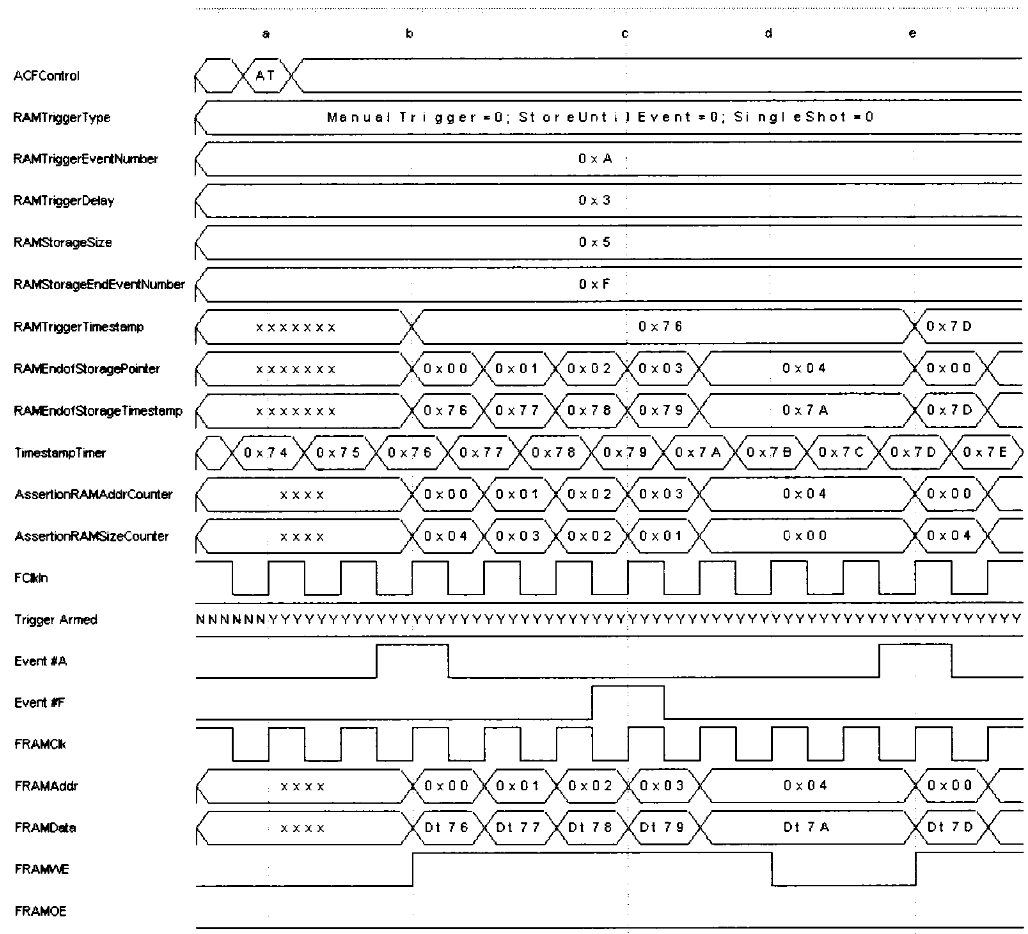

Figure 12 : Exemple de fonctionnement de la *Assertion RAM* en mode normal

Si un front montant de l'événement sélectionné survient avant que le compteur AssertionRAMSizeCounter n'ait atteint 0, le stockage recommence. La Figure 13 montre un exemple de ce qui survient dans ce cas. Le premier déclenchement survient au point b, mais un second survient au point c, ce qui réinitialise les compteurs, stocke un nouveau Timestamp de déclenchement et prolonge le stockage, qui se termine au point d.

Cette situation peut survenir lorsqu'un événement survient fréquemment et qu'on désire seulement conserver un exemple de la dernière apparition d'une situation donnée.

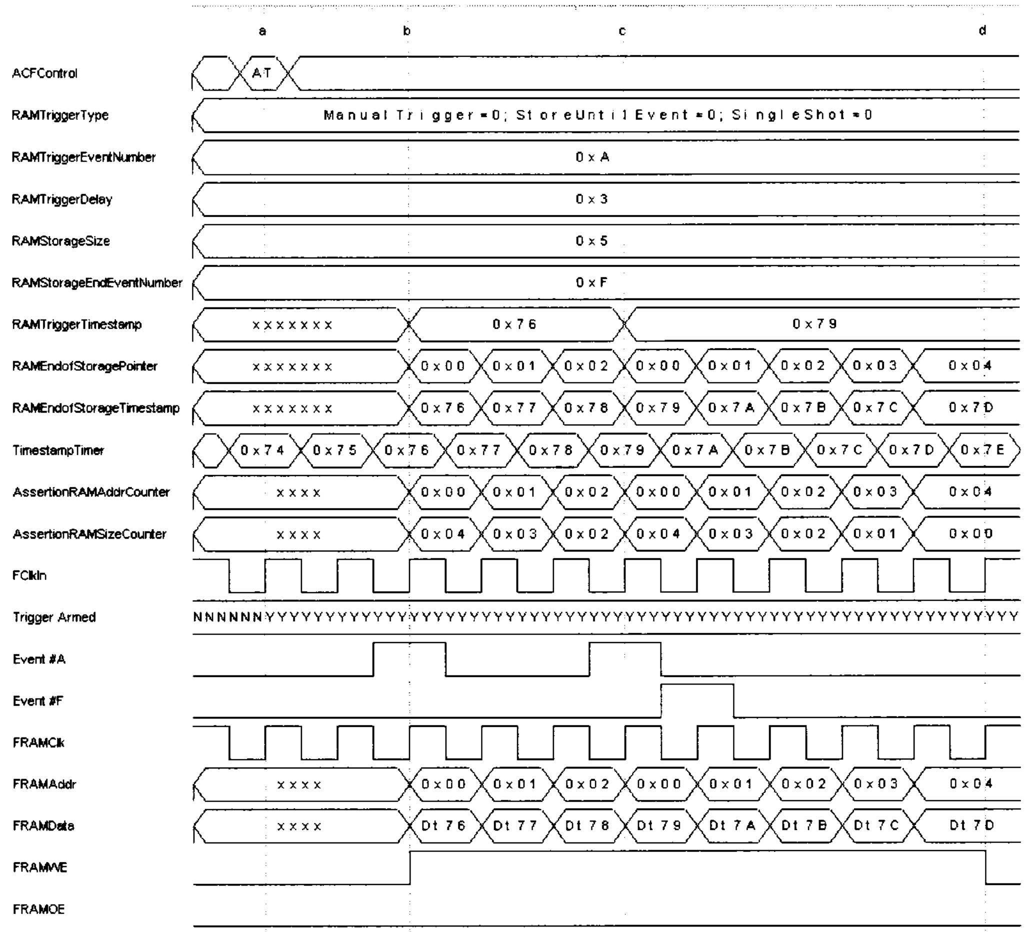

Figure 13 : Exemple de déclenchement survenant avant la fin du stockage en mode normal

Le stockage peut être interrompu par la commande Stop Storage (section 3.3.2.1). Le compteur AssertionRAMSizeCounter est alors mis à 0, ce qui termine le stockage. La Figure 14 montre un exemple d'une fin de stockage prématurée causée par cette commande. Le déclenchement survient au point b et le stockage est interrompu par la commande Stop storage au point c.

Cette commande peut servir si aucune autre façon d'interrompre le stockage n'a été configurée ou si un événement attendu pour arrêter le stockage ne survient pas et qu'on désire conserver les données accumulées jusqu'à maintenant.

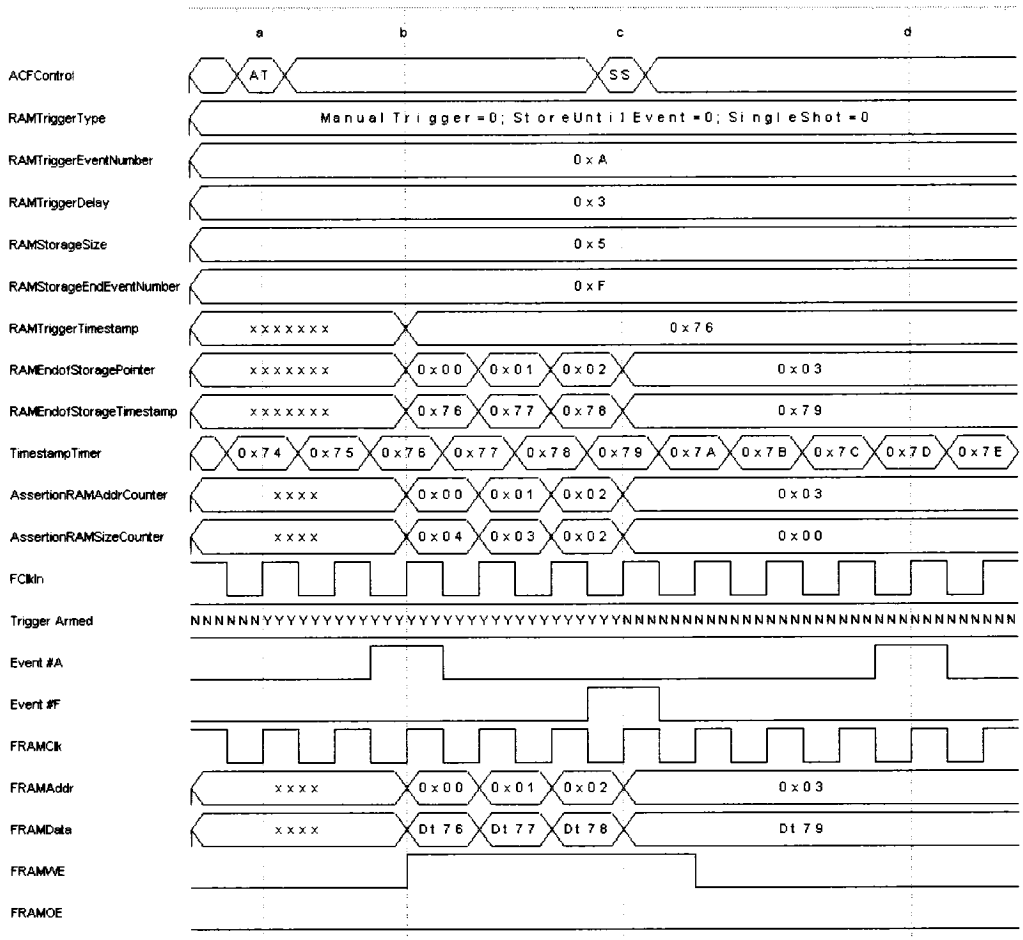

Figure 14 : Exemple de fin de stockage déclenché par la commande Stop storage

Le mode Single Shot (Single Shot = 1), a deux particularités. En premier lieu, un délai peut être programmé dans le registre RAM Trigger Delay (section 3.3.2.6) pour garder des données enregistrées avant l'activation du signal de déclenchement. De plus, le signal de déclenchement est désarmé une fois déclenché, ce qui empêche l'écrasement des données par une autre activation du signal de déclenchement. Le compteur AssertionRAMAddrCounter est initialisé à 0 dès l'armement du signal de déclenchement et le stockage commence immédiatement. Le compteur AssertionRAMSizeCounter est quant à lui initialisé à la valeur du registre RAM Storage Size - RAM Trigger Delay (section 3.3.2.6) à l'armement du signal de déclenchement, mais ne commence pas à décrémenter avant l'activation du signal de déclenchement. Cette particularité permet de

garder les données stockées pendant le délai préprogrammé précédant l'activation du signal de déclenchement, inclusivement.

La Figure 15 montre un exemple de fonctionnement du mode Single Shot. Le signal de déclenchement est armé au point a, ce qui déclenche immédiatement le stockage des données en mémoire et l'initialisation du compteur AssertionRAMSizeCounter. À l'activation du signal de déclenchement, au point b, le compteur AssertionRAMSizeCounter commence à décrémenter, puis le stockage cesse lorsque sa valeur atteint 0, au point c. On remarque aussi que le signal de déclenchement est désarmé au point b.

Ce mode est particulièrement utile si on veut garder la trace d'un événement spécifique et des conditions ayant mené à son apparition. Comme le signal de déclenchement est désarmé une fois le stockage démarré, la réapparition de l'événement déclencheur ne recommence pas le stockage et les données ne sont donc pas écrasées. En plus, le délai préprogrammé permet d'accumuler des informations sur l'état du circuit avant l'apparition de la condition de déclenchement. Ces informations peuvent servir à reproduire l'état du circuit en simulation, pour recréer les conditions menant à une erreur de fonctionnement.

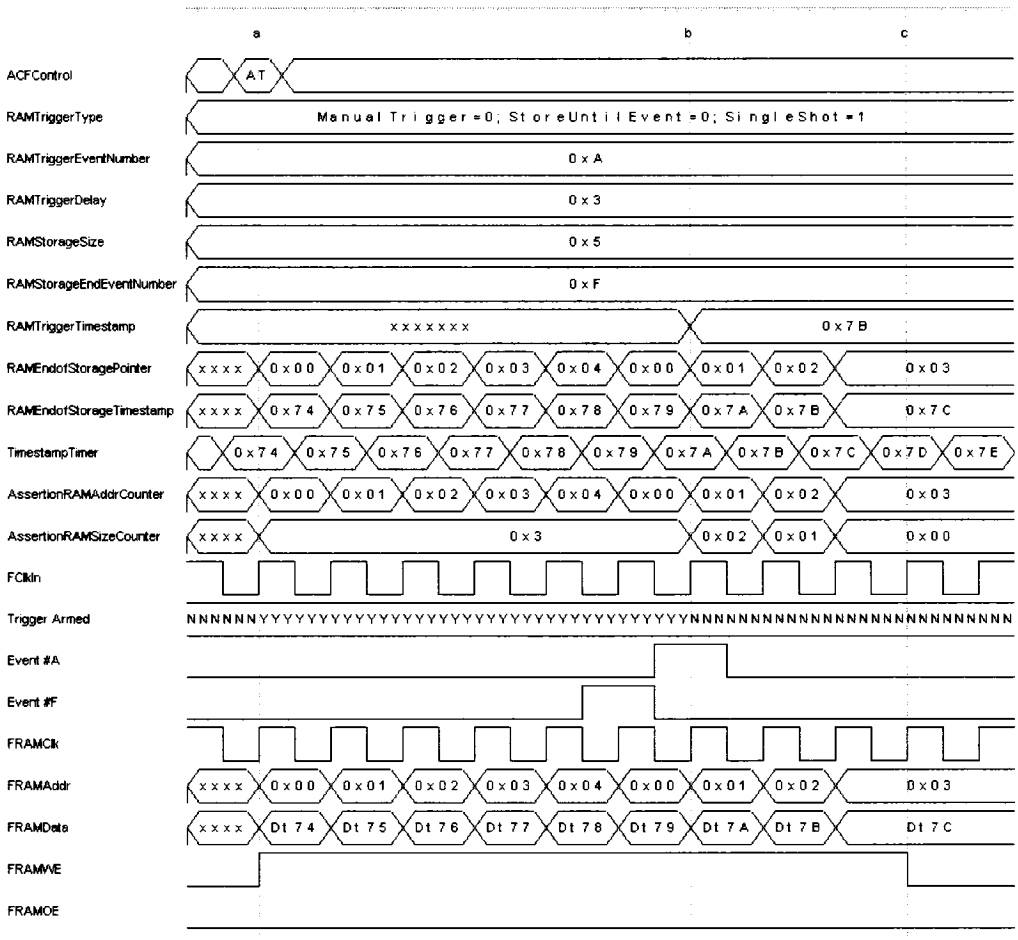

Figure 15 : Exemple de fonctionnement du mode Single Shot de la *Assertion RAM*

Le mode Store until event (Store until Event = 1) permet de continuer le stockage des données jusqu'à l'apparition d'un événement, sélectionné par le registre RAM Storage End Event Number (section 3.3.2.6). Il est possible que les données stockées depuis l'apparition du signal de déclenchement soient écrasées si le compteur AssertionRAMAddrCounter reboucle avant l'apparition de l'événement sélectionné.

La Figure 16 montre un exemple de fonctionnement du mode Store until event. Le signal de déclenchement est activé au point b, puis le stockage continue jusqu'à l'apparition de l'événement attendu, au point c.

Ce mode peut servir à accumuler des données entre l'apparition de deux conditions dans le circuit (la condition de déclenchement du stockage peut aussi servir à arrêter

l'écriture en mémoire). Par exemple, dans un circuit de télécommunication, le stockage pourrait démarrer à l'apparition du signal de début d'un paquet et s'interrompre à l'apparition du signal de fin de paquet, ce qui permet de stocker l'état du circuit pendant la réception d'un paquet.

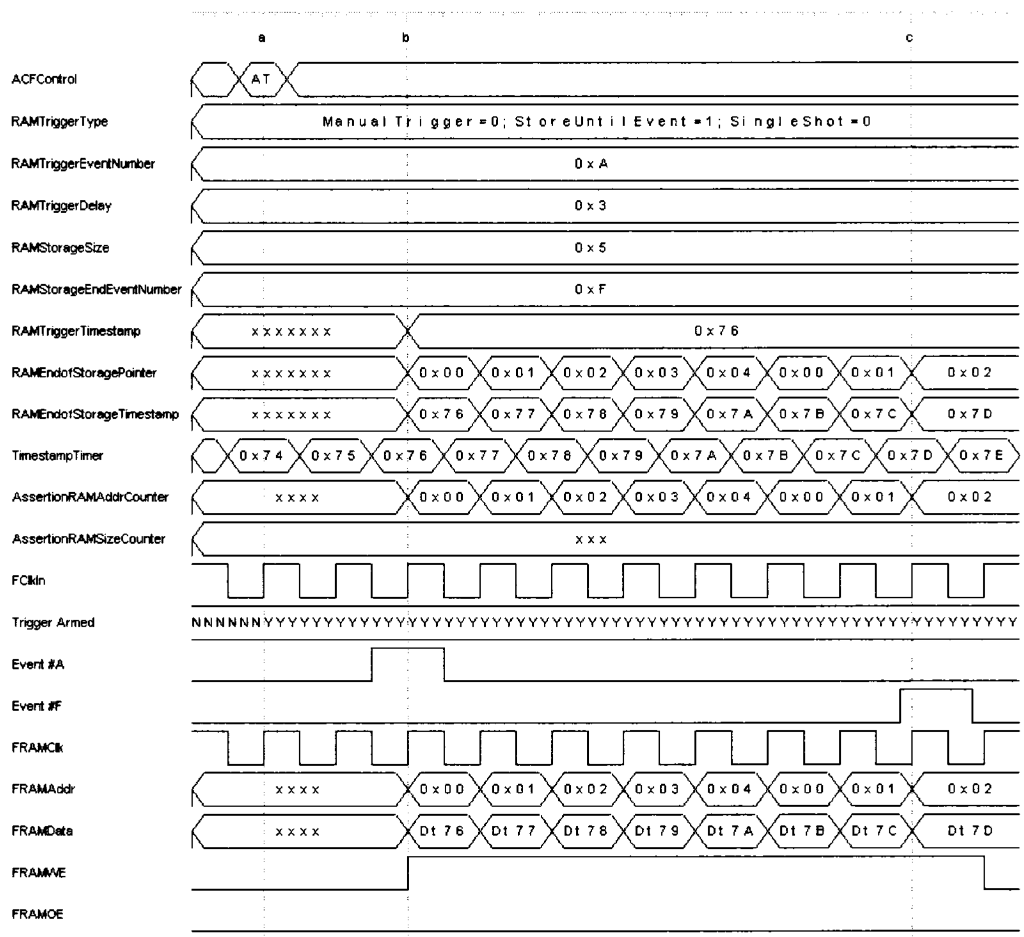

Figure 16 : Exemple de fonctionnement du mode Store until event de la *Assertion RAM*

Un dernier mode de fonctionnement, le mode Manual trigger (Manual Trigger = 1) permet de désactiver le signal de déclenchement automatique branché sur un moniteur d'événements. Une fois armé par la commande Arm Trigger, le signal de déclenchement doit absolument être activé par la commande Force Trigger (section 3.3.2.1) pour

déclencher le stockage. La fin du stockage est déterminée par le mode sélectionné dans le registre RAM Trigger type (section 3.3.2.6).

La Figure 17 montre un exemple du mode Manual trigger. Le signal de déclenchement est armé au point a, puis il est déclenché par la commande Force Trigger au point b. Le stockage s'effectue ensuite normalement jusqu'au point c.

Ce mode peut servir si le ACF est utilisé conjointement avec un appareil qui injecte des données dans le circuit. Cet appareil pourrait, par l'intermédiaire du processeur connecté au ACF, démarrer le stockage des données lorsqu'il envoie un certain type de données.

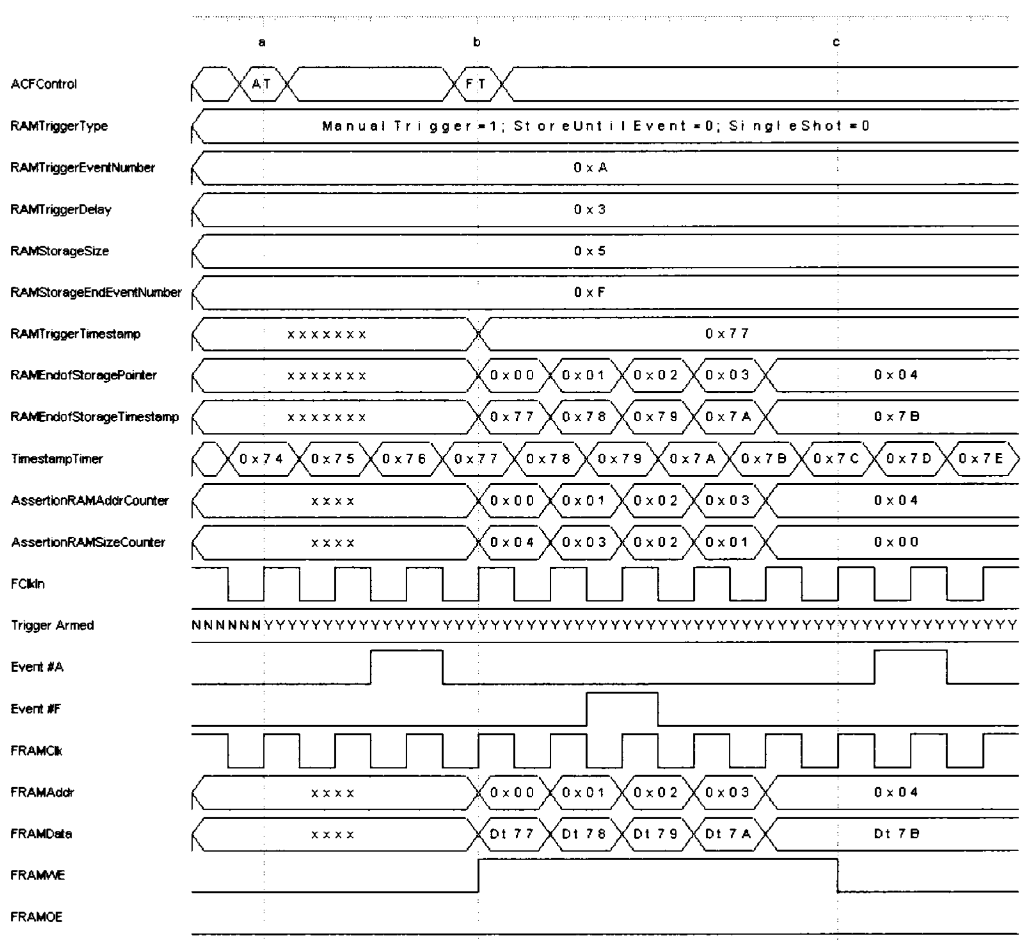

Figure 17 : Exemple de fonctionnement du mode Force Trigger de la *Assertion RAM*

3.3.1.8 Output Data Formatter

On peut voir le *Output Data Formatter* comme un gros multiplexeur/démultiplexeur entre les registres et le bus AMBA. Le *Output Data Formatter* sert à faire le lien entre les registres des différents modules du ACF (voir section 3.3.2) et la *AMBA Interface*. Ce module contient les décodeurs d'adresses pour connecter un registre du ACF à la *AMBA Interface*. Toutes les transactions d'écriture et de lecture des registres du ACF effectuées par le processeur passent par cette interface.

3.3.1.9 AMBA Interface

La *AMBA Interface* a pour fonction de gérer le protocole de communication AMBA AHB (section 3.3.3 et chapitre 3 de [6]) lors des lectures et écritures de registres par le processeur. Cette interface est reliée au *Output Data Formatter* pour les lectures et les écritures. Le processeur constitue le maître sur le bus, alors que le ACF se comporte comme un esclave (slave).

Tel que décrit dans la section 3.3.1.1, la *AMBA Interface* supporte seulement les transactions de type IDLE et NONSEQUENTIAL. Les transferts en BURST ne sont donc pas supportés. En plus, l'interface n'effectue que des transferts de taille WORD, soit 32 bits de large et n'implémente pas de niveau de protection pour les transferts. Comme statuts de transfert, la AMBA interface n'envoie que des messages de type OK ou ERROR. Un message ERROR survient lorsque le processeur tente d'accéder une adresse non associée à un registre du ACF ou lorsqu'il essaie d'effectuer une transaction non permise sur un registre (par exemple : lire un registre Écriture seulement ou écrire un registre Lecture seulement). Par contre, le signal HREADY peut être utilisé pour aider à synchroniser les domaines d'horloge.

La *AMBA Interface* assure la synchronisation entre les deux domaines d'horloge (FClkIn et FHCLK) lors des opérations de lecture/écriture de registres.

3.3.2 Description des registres du ACF

Cette section présente une description des registres du ACF, incluant, lorsque pertinent, le format des données, la plage d'adresses sur le bus AMBA et le type d'accès

au registre (lecture et/ou écriture). Comme la plupart des registres sont accessibles par le bus AMBA de 32 bits, les données sont présentées sous ce format, bien que les registres puissent nécessiter moins de bits. Les bits non utilisés par un registre sur le bus de données AMBA prennent la valeur 0 en lecture et ne sont pas considérés en écriture. Sauf mention contraire, les bits des registres prennent la valeur 0 au reset (signal FHRESETn ou Soft Reset, voir annexe B).

3.3.2.1 Registres généraux du ACF

Le *Assertion Checker* FPGA comporte trois registres généraux accessibles par le bus AMBA, qui servent à informer de l'état général du ACF et à exécuter des commandes globales.

a) ACF IDCode : Ce registre de 32 bits contient une valeur constante déterminée à la compilation, qui permet d'identifier le code programmé dans le FPGA.
b) ACF Status : Ce registre comporte 6 bits indiquant l'état courant du ACF.
c) ACF Control : Ce registre comporte 8 bits servant chacun à envoyer une commande générale au ACF.

3.3.2.2 Registres du Timestamp Timer

Le *Timestamp Timer* comporte deux registres accessibles par le bus AMBA, qui permettent de déterminer la valeur maximale de décompte du temps et de consulter la valeur présente du décompte.

a) Timestamp Timer Maximum Count : Ce registre de 8 à 32 bits (selon la largeur du compteur du *Timestamp Timer*) permet d'indiquer la valeur maximale atteignable par le compteur avant le rebouclage à zéro.
b) Timestamp Timer Current Time : Ce registre de 8 à 32 bits indique la valeur actuelle du compteur du *Timestamp Timer*.

3.3.2.3 Registres des Assertion Checkers

Chaque *Assertion Checker* comporte trois registres accessibles par le bus AMBA. Ces registres permettent de lire le bit d'état du Assertion Checker, d'indiquer une information optionnelle (Timestamp, compteur, etc..) et de remettre à zéro l'Assertion Checker.

a) AC State : Ce registre de 1 bit indique l'état du Assertion Checker.
b) ACTimestamp/Failure Counter : Ce registre de 0-32 bits indique une information sur le *Assertion Checker* (Timestamp, compteur, etc..)
c) AC Reset : Ce registre de $\log_2$nAC + 1 bits sert à remettre à zéro un ou tous les *Assertion Checkers.*

3.3.2.4 Registres du Event Generator

Le *Event Generator* comporte un registre global accessible par le bus AMBA, qui permet de déterminer quel moniteur d'événements est relié au signal FIRQ. De plus, chaque moniteur d'événements comporte trois registres accessibles par le bus AMBA. Ces registres permettent de lire le bit d'état du moniteur d'événements, d'indiquer une information optionnelle (Timestamp, compteur, etc..) et de remettre à zéro le moniteur d'événements.

a) EG IRQ Pin Event Number : Ce registre de $\log_2$nEG + 1 bits sert à indiquer si le signal FIRQ peut être activé et, le cas échéant, quel moniteur d'événements commande le signal FIRQ.
b) EG State : Ce registre de 1 bit indique l'état du moniteur d'événements.
c) EG Timestamp/Counter : Ce registre de 0-32 bits indique une information sur le moniteur d'événements (Timestamp, compteur, etc..)
d) EG Reset : Ce registre de $\log_2$nEG + 1 bits sert à remettre à zéro un ou tous les moniteurs d'événements.

3.3.2.5 Registres du Configuration Manager

Le *Configuration Manager* comporte deux registres principaux, qui permettent d'accéder à Configuration RAM et de déterminer si une configuration spécifique a été stockée. Le bus AMBA n'a accès qu'aux registres de la Configuration RAM.

a) Configuration RAM : Ces registres permettent d'accéder chaque configuration à stocker dans la Configuration RAM. L'adresse AMBA correspond à la configuration accédée.

b) Configuration stored : Ce registre de C bits sert à indiquer au *Configuration Manager* que les données de configuration d'une configuration sont stockées dans la Configuration RAM.

3.3.2.6 Registres de la Assertion RAM

La *Assertion RAM* comporte 9 registres accessibles par le bus AMBA. Ces registres permettent de configurer le signal de déclenchement, la taille des données stockées et l'événement de fin de stockage, en plus de permettre de consulter les valeurs de timestamp et de pointeur du signal de déclenchement et de la fin de stockage et les données stockées dans la *Assertion RAM*.

a) RAM Trigger type : Ce registre comporte 2 bits permettant d'indiquer le type de déclenchement et la condition de fin de stockage de la *Assertion RAM*.

b) RAM Trigger Event Number : Ce registre de $\log_2$nEG bits identifie le générateur d'événements qui active le signal de déclenchement de la *Assertion RAM*.

c) RAM Trigger Delay : Ce registre de 8-32 bits permet de déterminer le délai entre le début du stockage et l'apparition du signal de déclenchement, pour un déclenchement de type Single Shot (section a)). Les données stockées dans la *Assertion RAM* avant le déclenchement, pendant le délai indiqué par ce registre, sont conservées (voir section 3.3.1.7).

d) RAM Storage Size : Ce registre de 8-32 bits permet de déterminer le nombre de coups d'horloge pendant lequel le stockage est effectué après le déclenchement, en incluant le délai indiqué par le registre RAM Trigger Delay (section c)). Cette valeur détermine la valeur à laquelle le compteur d'adresse de la *Assertion RAM* reboucle.

e) RAM Storage End Event Number : Ce registre de $\log_2$nEG bits identifie le générateur d'événements qui provoque la fin du stockage dans la *Assertion RAM*, lorsque le bit Store Until Event du registre RAM Trigger Type est actif (section a)).

f) RAM Trigger Timestamp : Ce registre de 8 à 32 bits indique le temps du *Timestamp Timer* lors de la dernière activation du signal de déclenchement.

g) RAM End of Storage Pointer : Ce registre de 8-32 bits indique l'adresse de la *Assertion RAM* à laquelle la dernière donnée stockée a été écrite. Si le stockage est en cours, l'adresse indiquée est l'adresse de stockage actuelle.

h) RAM End of Storage Timestamp : Ce registre de 8 à 32 bits indique le temps du *Timestamp Timer* lors de la fin du stockage en mémoire. Si le stockage est en cours, ce registre indique le temps actuel du *Timestamp Timer.*

i) *Assertion RAM* : Chaque adresse correspond à une case mémoire de la *Assertion RAM.* La lecture d'une adresse provoque la lecture de la case mémoire correspondante de la *Assertion RAM.* Si un stockage est en cours, il est arrêté par une lecture à une des adresses de la *Assertion RAM.* Une lecture provoque également le désarmement du signal de déclenchement, qui doit être réarmé par la commande Arm Trigger (annexe B).

3.3.3 Protocoles de communication du ACF

Le *Debug Port Interface* suit le protocole de communication de la OCHP, décrit à la section 3.2.1.2.

L'accès à la *Assertion RAM* externe suit un protocole simplifié. En effet, les écritures s'effectuent en un cycle d'horloge, alors que les lectures s'effectuent en deux cycles. Pour une écriture, l'adresse est mise sur le bus FRAMAddr, la donnée sur FRAMData, le signal FRAMWE est mis à 1 et le signal FRAMOE est mis à 0, à temps pour le front montant de FRAMClk. Pour une lecture, l'adresse est mise sur le bus FRAMAddr avec le signal FRAMWE à 0 et FRAMOE à 1 à temps pour le premier coup d'horloge. La donnée est lue sur le bus FRAMData au coup d'horloge suivant. La Figure 18 montre un exemple d'écriture suivie d'une lecture de la *Assertion RAM.*

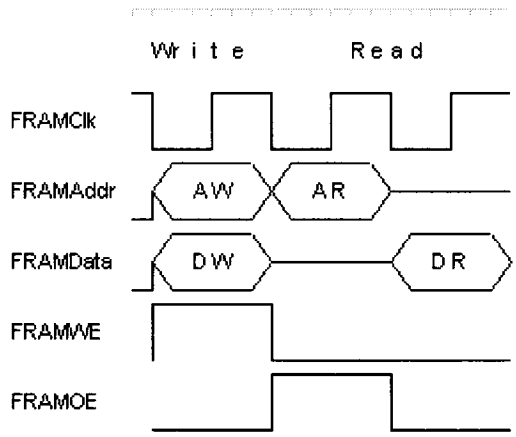

Figure 18 : Exemple d'écriture et de lecture de la *Assertion RAM*

La Figure 19 montre le chronogramme de plusieurs transferts sur le bus AMBA. Toute lecture (ou écriture) comporte deux phases : la phase d'adressage et la phase de données. Pendant la phase d'adressage, l'adresse est présentée sur le bus HADDR pour être lue au front montant de HCLK. Dans le cas d'une lecture, la donnée demandée apparaît sur le bus HRDATA au coup d'horloge suivant, à moins que le signal HREADY soit mis à 0, ce qui donne un coup d'horloge de plus pour l'obtenir. Dans le cas d'une écriture, la donnée à écrire est lue sur le bus HWDATA au coup d'horloge suivant la lecture de l'adresse. Si le ACF n'est pas prêt à lire la donnée, il met le signal HREADY à 0, pour que le processeur laisse la donnée sur HWDATA un coup d'horloge de plus. Pour plus de précisions sur ce protocole de bus, consulter la norme AMBA [6].

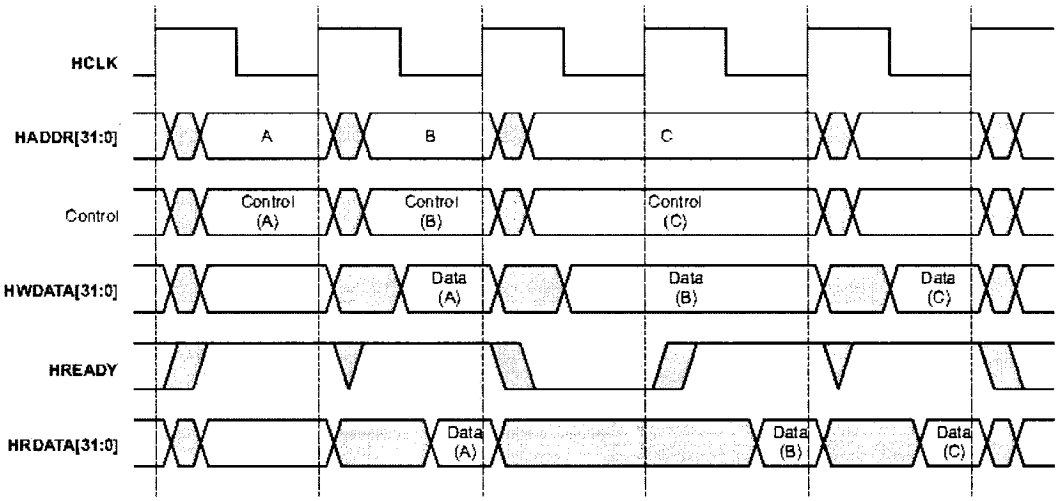

Figure 19 : Chronogramme de plusieurs transferts sur le bus AMBA AHB [6].

3.4 Compilateur d'assertions

Les paramètres de configuration dépendent à la fois du DUT et de la nature des vérificateurs d'assertions que l'usager désire implanter. Pour rendre la génération de ces paramètres la plus simple possible, il serait utile de fournir un compilateur d'assertions à l'usager. Un fichier décrivant la correspondance entre les bits de configuration et les signaux internes du design, la largeur du Port d'entrée/sortie et la fréquence d'horloge devra être créé pour chaque circuit. Le compilateur aura également besoin d'un autre fichier, contenant des assertions exprimées dans un langage tel que PSL ou OVL. Avec ces fichiers, le compilateur pourra traduire les assertions en code synthétisable pour programmer le ACF et les paramètres de configuration pour la OCHP. Le compilateur devra déterminer les situations pendant lesquelles le *Event Generator* devra changer la configuration. Lorsque des assertions ne peuvent se vérifier en raison de la limitation de la bande passante des signaux de sortie, le compilateur avertira l'usager. Le compilateur produira finalement un fichier de routage du FPGA et un fichier de données de configuration de la OCHP, qui sont envoyés au ACF par le processeur de la carte de vérification.

Ce compilateur pourrait être couplé à un compilateur d'assertions commercial [5][20] pour effectuer la synthèse des expressions PSL ou OVL, et prendrait en charge l'optimisation de la configuration de la OCHP et la production des générateurs d'événements du ACF.

3.5 Discussion

Dans cette section, certaines remarques sur la méthodologie sont présentées. Ensuite, des extensions à l'environnement de déverminage sont proposées.

3.5.1 Avantages de la méthodologie

La méthodologie proposée corrige certaines lacunes des autres systèmes d'instrumentation sur puce :

- L'accès parallèle et instantané aux données de déverminage élimine le besoin d'une mémoire tampon qui limiterait la quantité de données stockables avant le transfert à l'outil de déverminage. Le ACF a donc accès à toutes les données nécessaires aux vérificateurs d'assertions.
- L'accès direct aux données permet également de détecter l'effet d'erreurs de synchronisation dans le circuit, car le circuit peut fonctionner à vitesse nominale, dans sa configuration finale. Les chemins critiques peuvent ainsi être exercés en situation réelle de fonctionnement. L'accès aux signaux à vitesse nominale reste toutefois sujette aux limitations expliquées dans la section suivante.
- Le bus de configuration de $\log_2 C$ signaux permet un changement de la zone échantillonnée dans le circuit en un temps minimal (1 coup d'horloge). Il est donc possible de suivre l'exécution d'un calcul ou le cheminement d'une transaction dans différentes zones du circuit en temps réel. Cette particularité évite d'avoir à répéter une transaction plusieurs fois en changeant à chaque fois la zone d'échantillonnage.

Plusieurs avantages peuvent être retirés de l'utilisation de vérificateurs d'assertions synthétisés :

- Les assertions définies à l'étape de la conception du circuit peuvent, à condition d'utiliser des signaux connectés à la OCHP, être réutilisées dans le ACF pour vérifier le comportement du prototype. Cette technique a l'avantage d'économiser le temps d'interprétation et de codification de la spécification fonctionnelle du circuit, car les équipes de design ou de vérification ont déjà effectué cette tâche.
- L'implantation des vérificateurs d'assertions synthétisés dans un FPGA permet l'utilisation de plusieurs séries d'assertions selon les besoins, pour affiner la localisation d'une source d'erreur dans un circuit ou pour vérifier différents circuits munis d'une sonde reconfigurable de type OCHP.
- Le traitement en temps réel des assertions dans le ACF permet de maximiser la bande passante de la sonde intégrée et de diminuer la surface occupée par celle-ci. En effet, un traitement hors-ligne des assertions nécessite l'envoi d'une plus grande quantité de données de déverminage à l'extérieur du circuit, car les zones échantillonnées ne sont pas ciblées en fonction des assertions présentement actives. Cela implique que les données de déverminage potentiellement utilisables par tous les vérificateurs d'assertions doivent être envoyées à l'outil de déverminage, alors que certaines données ne sont nécessaires qu'à certains moments. Le changement de configuration de la sonde par le *Configuration Manager* permet donc de maximiser la bande passante, en évitant l'envoi sur le bus de données non nécessaires. En plus, une sélection précise des données permet d'éviter l'ajout d'une mémoire tampon, qui serait nécessaire pour stocker toutes les données nécessaires en attendant leur transmission sur le bus de déverminage.

3.5.2 Limitations de la méthodologie

L'architecture proposée pour la sonde reconfigurable peut nécessiter le routage de lignes relativement longues, en particulier pour amener des signaux situés à l'opposé du circuit vers la sonde. La longueur de ces signaux augmente le délai de propagation, qui peut dépasser une période d'horloge dans les circuits fonctionnant à très haute fréquence. De plus, le temps de montée et de descente des signaux numériques est affecté par la

grande capacité électrique des traces et par la vitesse de commutation des broches du circuit. Toutefois, des solutions peuvent être envisagées pour contourner ces limitations :

- Placer des tampons de retransmission le long d'une ligne pour régénérer le signal. Un délai supplémentaire s'ajoute par contre sur le signal.
- Pipeliner les signaux avec des registres répartis le long du trajet du signal. Les vérificateurs d'assertions devront toutefois tenir compte du retard induit par ces registres. Dans certains cas, le changement de la zone échantillonnée pourra être retardé de quelques coups d'horloge.
- Dédier des broches du bus de déverminage à des groupes de signaux, et les répartir uniformément sur la surface du circuit, au lieu de permettre à tous les signaux d'aller à toutes les broches. Cette solution diminue la longueur des traces mais enlève de la flexibilité pour le multiplexage des signaux de déverminage.
- Utiliser des paires de broches différentielles pour les signaux à haute fréquence. Cette solution double de nombre de broches du port de déverminage mais permet d'échantillonner et de transmettre des signaux à plus haute fréquence qu'avec une seule broche.
- Pour les signaux impossibles à router à l'extérieur, utiliser un algorithme de compression des données pour une transmission à plus basse fréquence.

La bande passante, bien qu'optimisée par la OCHP, reste limitée. Certaines assertions, en particulier celles impliquant des bus de grande largeur, ne peuvent être vérifiées en raison du trop grand nombre de signaux impliqués simultanément. L'emploi de mémoire tampon ou le transfert d'une partie de la logique de déverminage dans le DUV pourrait solutionner ce problème.

Finalement, dans le cas de plusieurs changements rapides de la zone échantillonnée, un mécanisme d'anticipation s'avère nécessaire, pour prévoir un coup d'horloge d'avance le changement de configuration. Cette anticipation permet d'envoyer les nouvelles configurations sur le bus PcfgData à temps pour le changement.

Chapitre 4

APPLICATION DE LA MÉTHODE

La méthode de validation en temps réel au moyen d'assertions a été appliquée sur différents circuits. Ce chapitre présente les applications faites de la méthode développée dans ce mémoire. Cette section sera organisée en fonction du type d'analyse effectuée sur la méthode. En premier lieu, une analyse des effets de la sonde intégrée sur un processeur Cordic développé au GRM sera présentée. Ensuite, un bus ISA simple a été utilisé pour appliquer la méthode de détection et de localisation d'erreurs, injectées sous forme de mutants.

4.1 Analyse des effets de la sonde intégrée

L'effet de sonde de la OCHP sur un modèle de processeur Cordic développé au GRM[25] a fait l'objet d'analyses sur le simulateur analogique de Cadence. Cette sous-section présente tout d'abord le processeur Cordic utilisé pour l'analyse, suivi d'une description de la méthode d'analyse de l'effet de sonde et des résultats de simulation.

4.1.1 Présentation du circuit : Processeur Cordic

L'algorithme CORDIC (COordinate Rotation Digital Computing) est une méthode simple et efficace qui permet de calculer une gamme de fonctions complexes. S'appuyant sur une technique à décalage-addition et à rotation vectorielle, l'algorithme calcule par approximation la plupart des fonctions basées sur la trigonométrie. Beaucoup de calculatrices commerciales utilisent cet algorithme pour résoudre des fonctions telles que sinus, cosinus, arc tangente et racine carrée. Les systèmes de communication numériques l'utilisent également afin de produire rapidement des conversions polaires-cartésiennes sur des signaux échantillonnés. En bref, l'algorithme repose sur des opérations mathématiques élémentaires qui, en plus d'offrir une grande efficacité de calcul, simplifient énormément l'implantation sur des plates-formes logicielles ou matérielles [25].

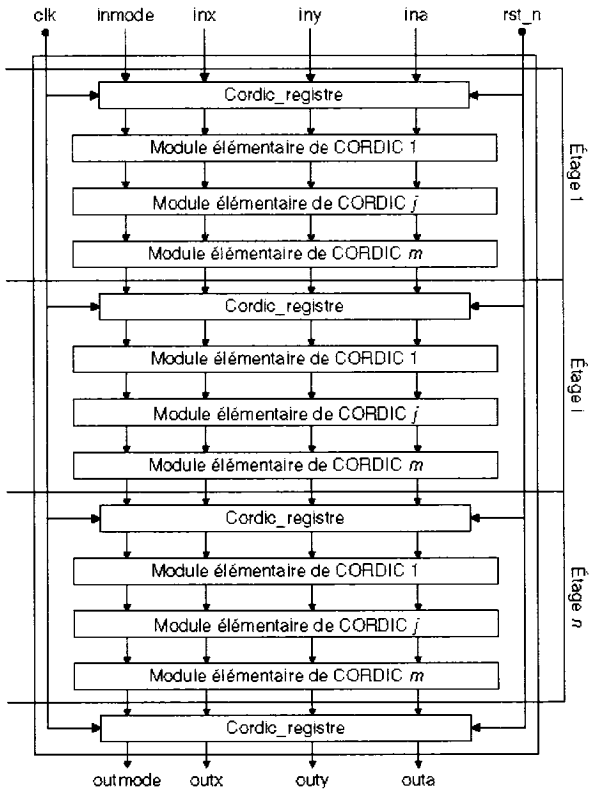

Figure 20 : Architecture du processeur Cordic

La Figure 20 montre l'architecture du processeur Cordic utilisé pour l'analyse. Pour simplifier les étapes de synthèse, le nombre d'étages a été limité à 4, avec un seul module élémentaire de CORDIC par étage. Le Tableau 5 décrit les bus de données du processeur Cordic, avant l'ajout de la OCHP.

Tableau 5 : Entrées/sorties du processeur Cordic

Signal	Type	Niveau actif	Description
Clk	Input	N/A	Horloge du processeur.
Rst_n	Input	0	Signal d'initialisation du module (asynchrone).
Inmode	Input	N/A	Mode de calcul : vectoriel (0) ou rotation (1).
Inx	Input	N/A	Position initiale en x.
Iny	Input	N/A	Position initiale en y.
Ina	Input	N/A	Angle A.
Outmode	Output	N/A	Mode de calcul pour obtenir les positions et angles résultants: vectoriel (0) ou rotation (1).
Outx	Output	N/A	Position résultante en x.
Outy	Output	N/A	Position résultante en y.
Outa	Output	N/A	Angle résultant.

Ce processeur permet d'effectuer rapidement des opérations trigonométriques, à l'aide d'opérations simples telles que des additions, des décalages et des comparaisons.

4.1.2 Analyse de l'effet de sonde de la OCHP

Une version de la OCHP comportant les paramètres C=8, N=52 et P=16 a servi pour effectuer l'analyse de l'effet de sonde dans le processeur Cordic. 52 des 64 signaux internes du processeur ont été connectés au Data Filter de la sonde. Le code VHDL du processeur Cordic et de la OCHP apparaissent en annexe.

La modélisation de façon réaliste du processeur Cordic, avec et sans la OCHP, a suivi les étapes suivantes : synthèse logique, placement et routage, extraction des valeurs de capacité parasites et simulation analogique des cas critiques. Ces étapes sont décrites plus en détail ci-dessous.

4.1.2.1 Synthèse logique, placement et routage

La synthèse logique, le placement et le routage du modèle VHDL du processeur Cordic ont été effectués selon les étapes décrites dans [11]. En résumé, le logiciel Design Analyzer de Synopsys a servi pour compiler et synthétiser le code VHDL du processeur Cordic muni d'une sonde de type OCHP. Ensuite, le placement et le routage ont été

réalisés avec le logiciel SOCEncounter, de Cadence. La Figure 21 montre le résultat du routage du processeur Cordic, avec la OCHP. Cette version sera appelée version OCHP dans le reste du chapitre.

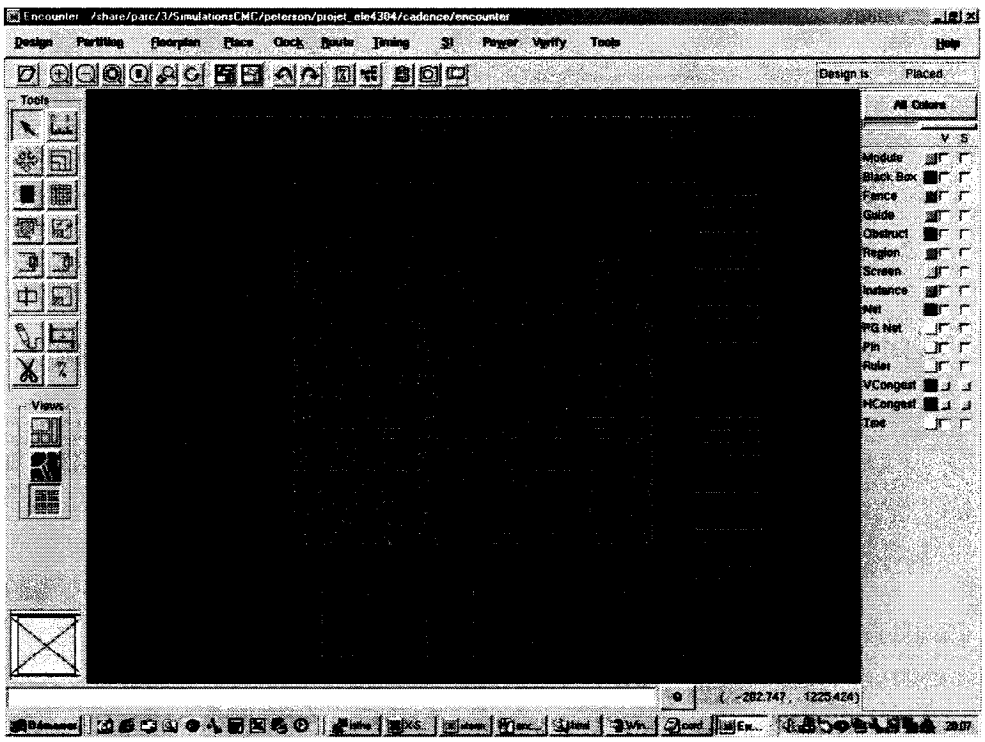

Figure 21 : Routage du processeur Cordic avec la OCHP

Pour effectuer un calcul le plus juste possible de l'effet de sonde, le même placement et routage a servi pour la version du processeur Cordic sans OCHP. D'une façon manuelle, les fils et éléments logiques de la OCHP ont été retirés du fichier DEF représentant le design routé. Cette méthode permet d'éliminer seulement les effets parasites des composantes de la sonde intégrée, en conservant le placement et le routage du reste du circuit. En conséquence, les capacités parasites dans le reste du circuit demeurent inchangées. La Figure 22 montre les résultats du routage du processeur Cordic, après le retrait des éléments de la OCHP. Cette version sera appelée version sans OCHP dans le reste du chapitre. Si on compare avec la Figure 21, on remarque dans le coin supérieur droit une zone plus noire, là où les fils et éléments logiques ont été retirés.

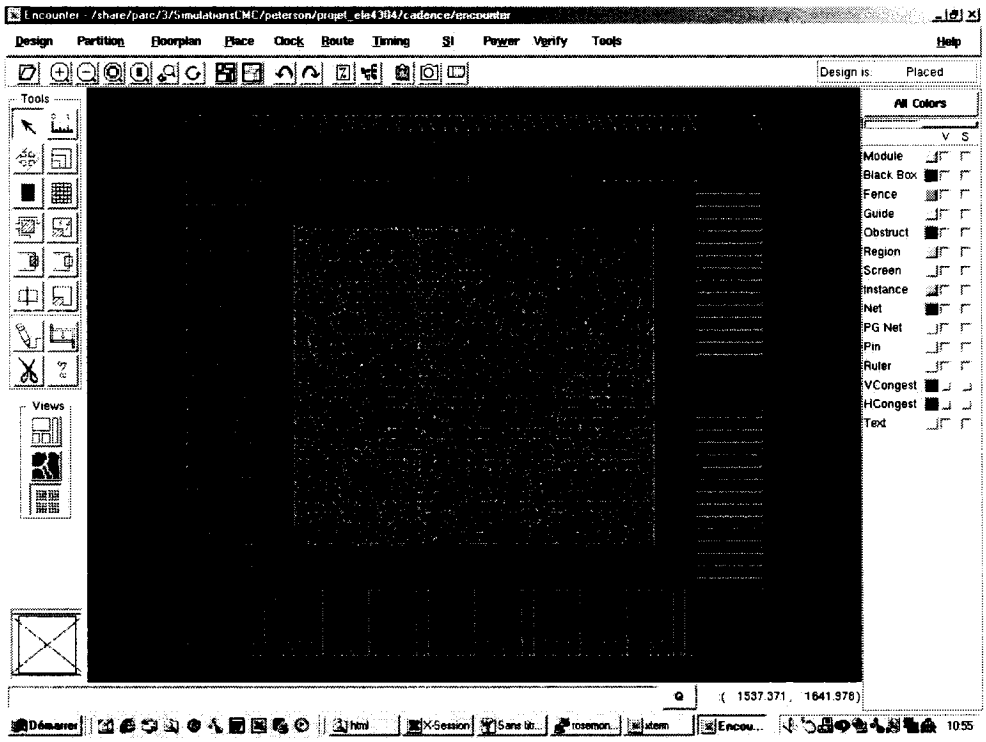

Figure 22 : Routage du processeur Cordic après le retrait de la OCHP

4.1.2.2 Extraction de la valeur des capacités parasites

Pour effectuer un calcul précis de l'effet de sonde de la OCHP, le logiciel Analog Design Environment de Cadence a servi pour extraire la valeur des capacités parasites des deux versions du routage du processeur Cordic (OCHP et sans OCHP). Une nouvelle « netlist », comportant les valeurs des capacités parasites, a été générée pour chaque version du processeur. Divers scripts, dont le code source apparaît en annexe, ont été utilisés pour :

- Identifier les nœuds électriques directement liés à la sonde OCHP;
- Trouver les nœuds électriques dont la capacité parasite avec les autres nœuds est la plus élevée;
- Calculer la somme des capacités parasites sur les nœuds identifiés, en assumant le pire cas;

Pour un fil donné dont l'environnement est déterminé, le pire cas de la capacité parasite apparente sur un nœud électrique est observé lorsque tous les autres nœuds environnants changent au même moment que le nœud affecté, en prenant une valeur logique inverse. Pour calculer la capacité parasite totale, plusieurs modèles suggèrent de supposer que les lignes environnantes sont reliées à la masse [39]. Cette capacité parasite implique un signal fixe sur toutes les lignes environnantes. Le calcul de la capacité parasite totale équivalente lorsque toutes les lignes changent simultanément, avec une polarité de transition inverse de la ligne étudiée, implique alors de sommer toutes les capacités trouvées entre les lignes et de les multiplier par 2. Cette valeur donne le pire cas de la capacité parasite équivalente, rapportée à la masse.

4.1.2.3 Simulation analogique des nœuds critiques

Le logiciel Analog Design Environment de Cadence a également servi à simuler un modèle analogique des nœuds critiques, avec la capacité parasite de pire cas. Trois nœuds candidats ont fait l'objet d'une modélisation et d'une simulation, pour les versions OCHP et sans OCHP. La Figure 23 et la Figure 24 montrent le modèle d'un chemin critique du processeur sans et avec la OCHP. On y remarque, à la sortie du registre de gauche, le condensateur représentant la capacité parasite totale.

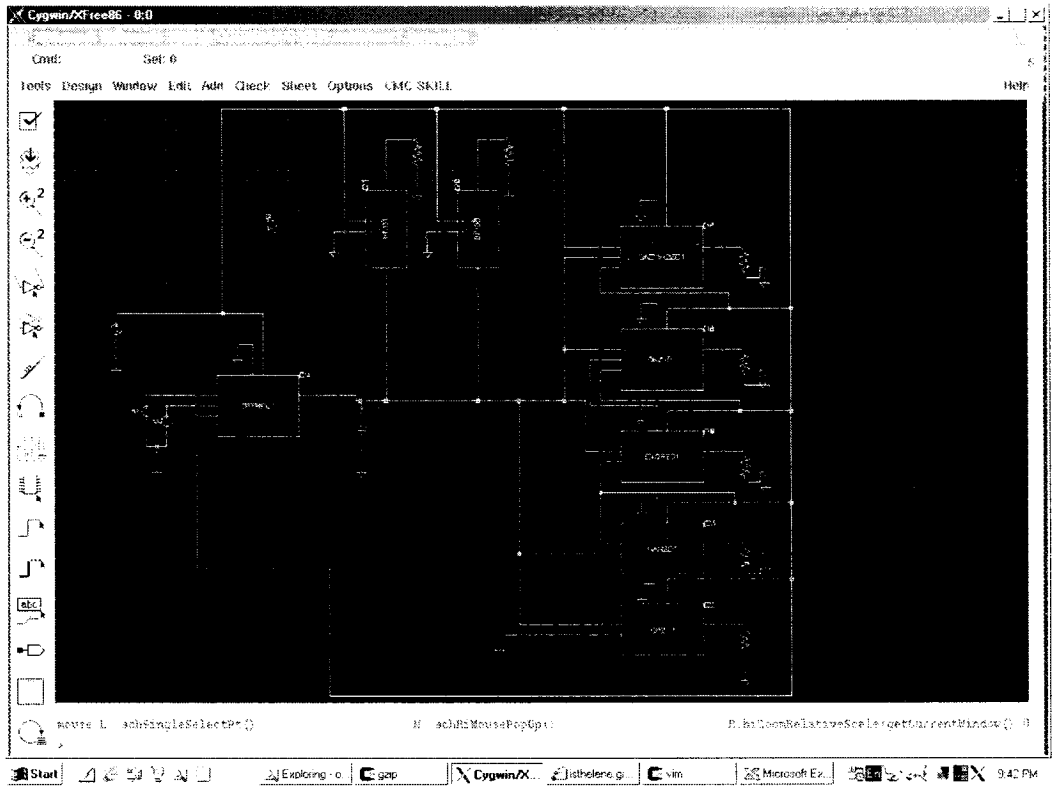

Figure 23 : Modèle d'un chemin critique du processeur sans la OCHP

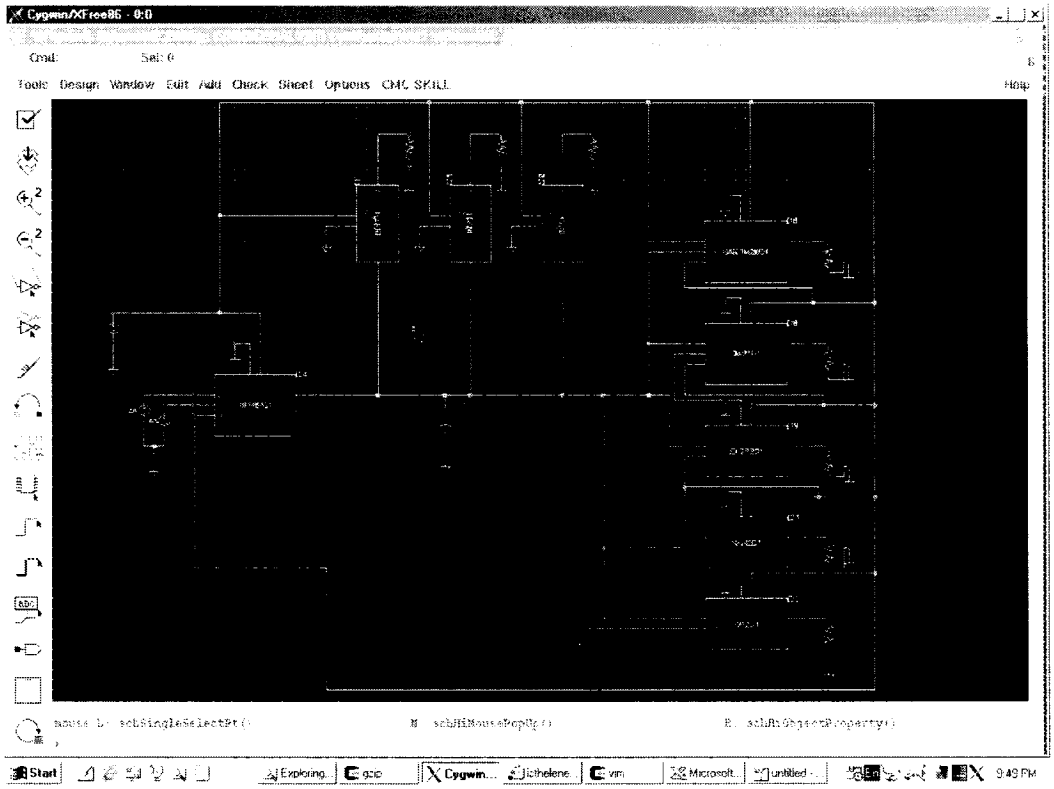

Figure 24 : Modèle d'un chemin critique du processeur Cordic avec la OCHP

La Figure 25 et la Figure 26 montrent les résultats de la simulation des modèles schématisés ci-dessus, à une fréquence de 250 MHz. On remarque que la forme du signal n'est pas affectée par la capacité parasite induite par la sonde. En plus, le temps de montée du signal, déduit de la tension maximale atteinte avant d'amorcer la descente, varie d'une façon négligeable.

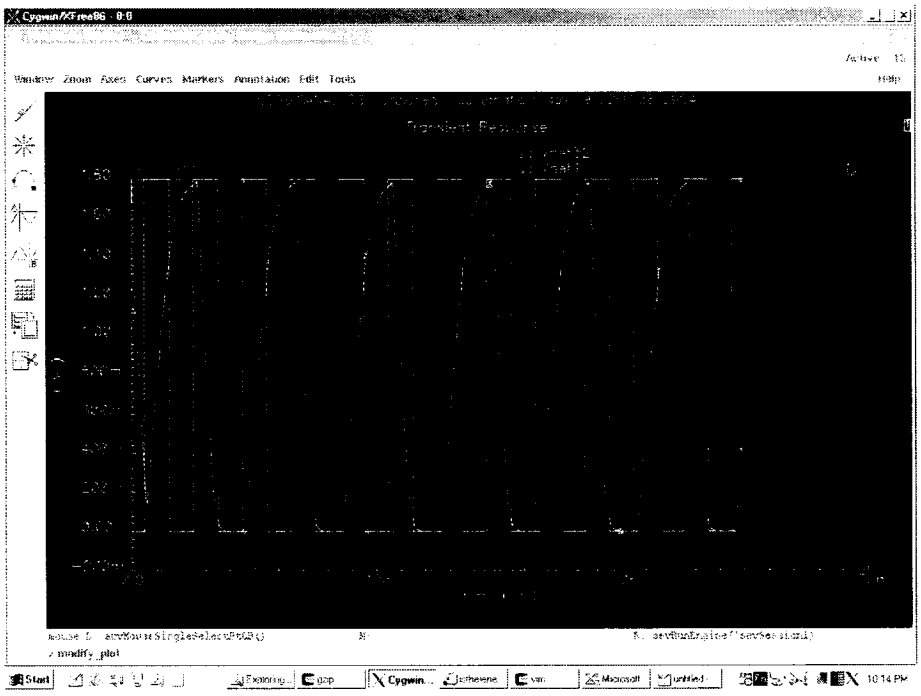

Figure 25 : Simulation d'un chemin critique du processeur Cordic sans la OCHP

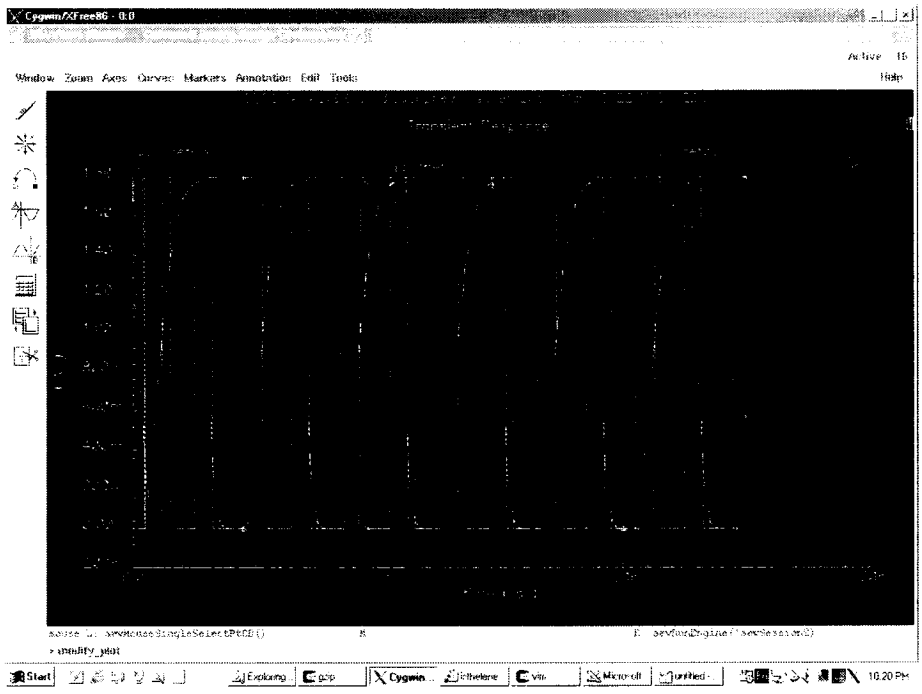

Figure 26 : Simulation d'un chemin critique du processeur Cordic avec la OCHP

Le Tableau 6 résume les résultats de l'analyse de l'effet de sonde de la OCHP pour les chemins critiques directement reliés à la sonde.

Tableau 6 : Variation de la capacité parasite pour les chemins critiques du processeur Cordic (à 250 MHz)

Nom du nœud	Capacité parasite sans OCHP (F)	Capacité parasite avec OCHP (F)	% variation
int_a_2\]\[7_	1,32E-13	1,37E-13	3,42%
n669	9,16E-14	9,17E-14	0,13%
n2308	7,46E-14	7,47E-14	0,09%

4.2 Détection et localisation d'erreurs

La méthode de détection et de localisation d'erreurs de fonctionnement au moyen de vérificateurs d'assertions synthétisés a été validée à l'aide d'un modèle de bus ISA simple, provenant de l'Université de Californie à Riverside [42]. Ce modèle a comme avantage un fonctionnement suffisamment complexe pour permettre l'écriture de plusieurs assertions, tout en étant synthétisable dans un FPGA de taille moyenne. Le processeur Cordic ne pouvait servir à cette démonstration, car il comporte des unités arithmétiques comportant des bus de données très larges, ce qui aurait monopolisé la bande passante de la sonde OCHP, au détriment des signaux de contrôle internes.

4.2.1 Présentation du circuit : Modèle de bus ISA

Le modèle de bus ISA implémente un contrôleur de bus, couplé à un chemin de données qui effectue une simple addition sur les données écrites. Une sonde de type OCHP a été ajoutée au modèle, pour échantillonner tous les signaux internes disponibles. Le code VHDL du modèle de bus ISA et de la OCHP apparaît en annexe. La OCHP ajoutée au circuit a les paramètres suivants : C=4, N=62 et P=16.

Le contrôleur sert à gérer les opérations de lecture et d'écriture sur le bus ISA. Il fonctionne en accord avec la machine à états illustrée à la Figure 27. Une transaction suit la séquence suivante :

- À partir de l'état S0, une demande de transaction est initiée lorsque le signal ALE_P prend la valeur 1 pendant un coup d'horloge. Le contrôleur passe alors à l'état S1.
- Si l'adresse présente sur le bus ADDRESS_P n'est pas dans la plage 0xA000-0xA001, les signaux Wem et Rem du chemin de données restent à 0 et le contrôleur reste à l'état S1. Sinon, le contrôleur passe à l'état S2 et le bus est prêt à identifier le type de transaction au coup d'horloge suivant.
- Une demande d'écriture est initiée en mettant le signal IOW_P à 0 pendant deux coups d'horloge, avec l'adresse 0xA000 prête sur le bus ADDRESS_P. La donnée à écrire doit être prête sur le bus DIN_P. Ces conditions vont faire passer le contrôleur de l'état S2 aux états S3 et S3A, successivement.
- Une demande de lecture est initiée en mettant le signal IOR_P à 0 pendant deux coups d'horloge, avec l'adresse 0xA001 prête sur le bus ADDRESS_P. Le signal IOW_P a priorité sur IOR_P. La donnée lue apparaît sur le bus DOUT_P au coup d'horloge suivant. Ces conditions vont faire passer le contrôleur de l'état S2 aux états S4 et S4A, successivement.
- La transaction se termine au coup d'horloge suivant par le retour à l'état S0.

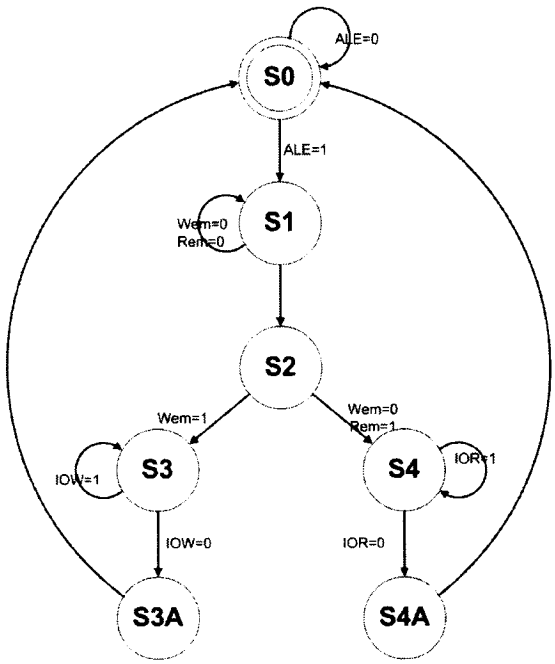

Figure 27 : Machine à états du contrôleur du bus ISA

Le chemin de données, quant à lui, prend la donnée écrite sur le bus DIN_P, la stocke dans un registre, ajoute 1 à la valeur de ce registre et stocke le résultat dans un second registre.

La Figure 28 donne un exemple d'une transaction d'écriture (de la valeur 0x33), suivie d'une lecture (de la valeur 0x34).

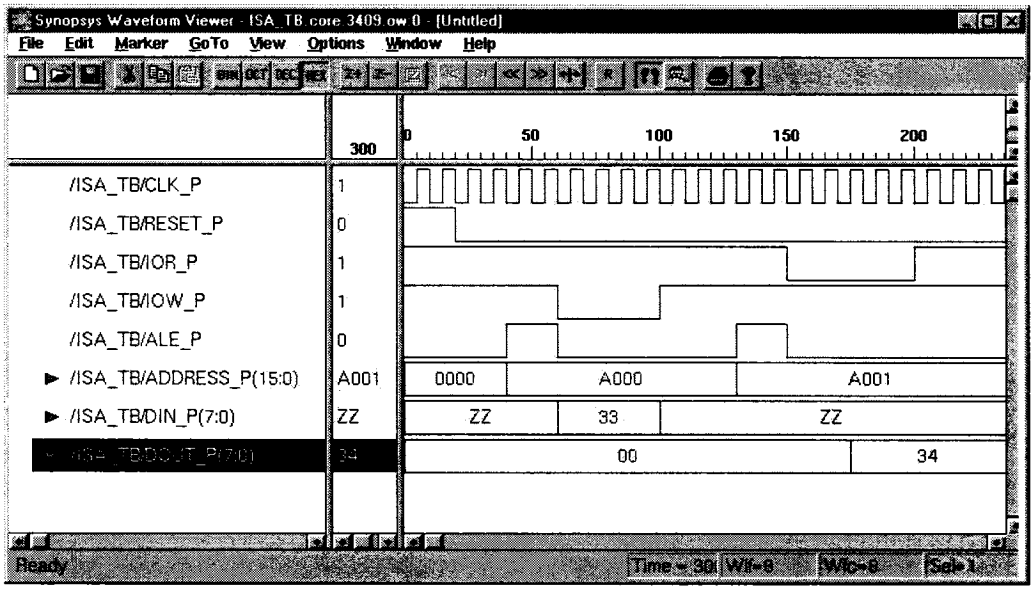

Figure 28 : Chronogramme d'une écriture et d'une lecture sur le bus ISA[42]

Ce modèle de bus gère des transactions se déroulant dans différentes parties du circuit, ce qui permet de démontrer l'utilité du ACF pour changer la zone d'échantillonnage en temps réel. Tous les signaux internes de contrôle et les chemins de données ont été reliés à la OCHP, selon l'adressage présenté dans le tableau suivant :

Tableau 7 : Assignation des signaux internes du bus ISA au Data Filter de la OCHP

Numéro PIntSignal (décimal)	Numéro PIntSignal (hexadécimal)	Signal du modèle ISA
61	0x3D	WE_sig
60	0x3C	RE_sig
59	0x3B	Ald_sig
58	0x3A	Dld_sig
57	0x39	Dst_sig
56	0x38	IOR_P
55	0x37	IOW_P
54	0x36	ALE_P
53-38	0x35-0x26	ADDRESS_P(15:0)
37-30	0x25-0x1E	DIN_P(7:0)
29-22	0x1D-0x16	IntDOUT_P(7:0)
21-14	0x15-0x0E	Datapath/reg2adder
13-6	0x0D-0x06	Datapath/adder2reg
5-3	0x05-0x03	Controller/CStateV
2-0	0x02-0x00	Controller/NStateV

4.2.2 Assertions synthétisées

Trois séries d'assertions, couvrant des fonctionnalités différentes du bus ISA, ont servi à en décrire le comportement. Les vérificateurs d'assertions synthétisés, ainsi que les générateurs d'événements nécessaires aux changements dynamiques de la configuration de la OCHP, ont été ajoutés au code synthétisable du ACF. Une description de la partie commune du ACF servant aux trois séries d'assertions apparaît ci-dessous, suivie d'une description spécifique des séries d'assertions. Le code VHDL du ACF et des vérificateurs d'assertions apparaît en annexe.

4.2.2.1 Synthèse du ACF

La version synthétisée du ACF implémente tous les modules décrits à la section 3.3.1, à l'exception de l'interface AMBA. Comme la plate-forme de prototypage ne comporte pas de processeur, ce bus devenait superflu. L'accès aux registres du ACF s'effectue avec une interface propriétaire, composée d'un clavier PS/2 et d'un affichage 7 segments.

Pour rendre l'ajout des vérificateurs d'assertions et des générateurs d'événements le plus simple possible, une enveloppe (« wrapper »), se connectant aux modules Data Formatter et *Configuration Manager*, regroupe les signaux de contrôle et de données. Cette enveloppe, le *ac_eg_module*, permet d'instancier un nombre variable de vérificateurs d'assertions et de générateurs d'événements, sans affecter les signaux des autres modules. Le compilateur d'assertions n'a donc pas à tenir compte du reste du ACF lors de la création du code VHDL.

4.2.2.2 Séries d'assertions du bus ISA

Les différentes fonctionnalités du modèle de bus ISA peuvent être décrites au moyen de trois séries d'assertions. Chaque série d'assertions est associée à un générateur d'événements, qui détermine les changements de la zone échantillonnée selon le comportement actuel du circuit. Les zones prédéterminées par 4 configurations programmées dans la OCHP par le *Configuration Manager*, couvrent tous les secteurs du circuit vérifiés par les assertions. Le générateur d'événements agit comme une machine à états, qui décide, avec une anticipation d'un cycle d'horloge, la zone à échantillonner

pour suivre la transaction dans les divers secteurs du circuit. Le code VHDL de chaque série d'assertions et des générateurs d'événements associés apparaît en annexe.

La première série d'assertions décrit le fonctionnement de la machine à états du contrôleur et le comportement des entrées/sorties du bus. Le Tableau 8 montre les quatre configurations de la sonde OCHP nécessaires à la vérification de la première série d'assertions.

Tableau 8 : Configuration de la OCHP du bus ISA pour la première série d'assertions

Configuration #0																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	CStateV			Ald	Dld	Dst	ALE_P	DOUT(3:0)				ADDR(15:12)				ADDR(0)
PIntSignal OCHP	5	4	3	59	58	57	54	25	24	23	22	53	52	51	50	38
Configuration #1																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	CStateV			Ald	Dld	Dst	Wem	Rem	IOW	IOR		ADDR(15:12)				ADDR(0)
PIntSignal OCHP	5	4	3	59	58	57	61	60	55	56		53	52	51	50	38
Configuration #2																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	CStateV			Ald	Dld	Dst	ALE_P	IOW	DIN(3:0)				reg2adder(3:0)			
PIntSignal OCHP	5	4	3	59	58	57	54	55	33	32	31	30	17	16	15	14
Configuration #3																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	CStateV			Ald	Dld	Dst	ALE_P	IOR	adder2reg(3:0)				DOUT(3:0)			
PIntSignal OCHP	5	4	3	59	58	57	54	56	9	8	7	6	25	24	23	22

Dans les quatre cas, les signaux CStateV, Ald, Dld et Dst permettent de suivre l'état du contrôleur, pour vérifier si les changements d'états correspondent à la machine à états illustrée à la Figure 27 et pour que le *Event Generator* puisse changer la zone échantillonnée en fonction de l'état du circuit.

La configuration #0 comprend en plus les signaux ALE_P, DOUT(3:0) et une partie du bus ADDRESS_P. Cette configuration permet de vérifier si le contrôleur passe bien de l'état S0 à l'état S1 lorsque ALE_P=1 et permet également de vérifier une partie du bus de sortie lors du passage de l'état S4A (lecture du bus) à l'état S0.

La configuration #1 comprend les signaux Wem, Rem, IOW, IOR et une partie du bus ADDRESS_P. Cette configuration permet de vérifier si le contrôleur passe bien de l'état S1 à l'état S2 (et si Wem ou Rem prend une valeur différente de 0) lorsque l'adresse est dans la plage 0xA000-0xA001.

La configuration #2 comprend les signaux ALE_P, IOW, DIN(3:0) et l'entrée de l'additionneur *reg2adder(3:0)*. Cette configuration permet de vérifier si le chemin de données charge la donnée présente sur le bus DIN dans le registre d'entrée de l'additionneur et si le contrôleur passe de l'état S2 à S3 et S3A lors d'une écriture sur le bus (IOW=0).

La configuration #3 comprend les signaux ALE_P, IOR, DOUT(3:0) et la sortie de l'additionneur *adder2reg(3:0)*. Cette configuration permet de vérifier si la donnée présente à la sortie de l'additionneur est chargée dans le registre de sortie du chemin de données et si le contrôleur passe de l'état S2 à S4 et S4A lors d'une lecture sur le bus (IOR=0).

Le générateur d'événements doit gérer les changements de configuration en temps réel pendant le fonctionnement du circuit. Pour ce faire, une machine à quatre états, soit un par configuration, a été créée. Elle réagit selon le schéma montré à la Figure 29.

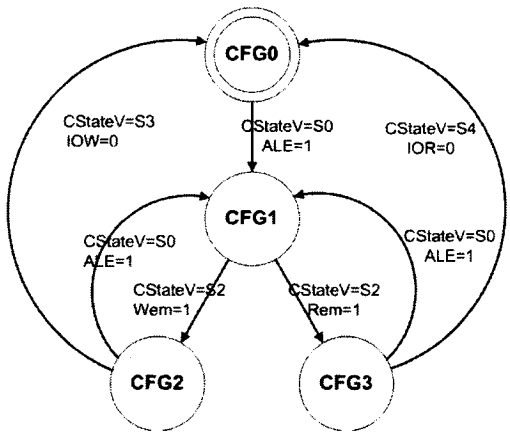

Figure 29 : Machine à états du générateur d'événements de la première série d'assertions

Le Tableau 9 décrit les assertions de la première série, avec le numéro de la configuration où elles sont vérifiées. Dû à un problème de synthèse de l'assertion OVL *assert_next*, un pipelinage de un coup d'horloge a dû être ajouté pour l'expression antécédente des assertions 0 à 11 et l'assertion *assert_implication* a été utilisée à la place. Pour les assertions 37 et 38, les valeurs de DIN et de adder2reg, respectivement, ont été pipelinées, pour comparer avec la dernière valeur et non avec la valeur actuelle.

Les assertions 0 à 12 servent à vérifier si le contrôleur du bus se comporte conformément à la machine à états de la Figure 27. Les assertions 13 à 33 vérifient si les signaux de contrôle internes Ald, Dld et Dst prennent la bonne valeur selon l'état du contrôleur. Les assertions 34 à 36 vérifient si les signaux de contrôle internes Wem et Rem prennent la bonne valeur selon l'état du contrôleur et l'adresse présente sur le bus ADDRESS_P. L'assertion 37 vérifie si la donnée présente sur la moitié du bus DIN est bien écrite dans le registre d'entrée lors d'une écriture. L'assertion 38 vérifie quant à elle si la donnée présente sur la moitié du bus de sortie de l'additionneur est bien écrite dans le registre de sortie lors d'une lecture.

Tableau 9 : Première série d'assertions du bus ISA

Numéro	Vérifiée lorsque PconfID =	Assertion
0	0,2,3	Assert_implication(Lst(CstateV=S0&&ALE=0),CstateV=S0)
1	0,2,3	Assert_implication(Lst(CstateV=S0&&ALE=1),CstateV=S1)
2	0,1,2,3	Assert_implication(Lst(CstateV=S1),CstateV=S2)
3	1	Assert_implication(Lst(CstateV=S2&&Wem=0&&Rem=0),CstateV=S2)
4	1	Assert_implication(Lst(CstateV=S2&&Wem=1),CstateV=S3)
5	1	Assert_implication(Lst(CstateV=S2&&Wem=0&&Rem=1),CstateV=S4)
6	1,2	Assert_implication(Lst(CstateV=S3&&IOW=1),CstateV=S3)
7	1,2	Assert_implication(Lst(CstateV=S3&&IOW=0),CstateV=S3a)
8	0,1,2,3	Assert_implication(Lst(CstateV=S3a),CstateV=S0)
9	1,3	Assert_implication(Lst(CstateV=S4&&IOR=1),CstateV=S4)
10	1,3	Assert_implication(Lst(CstateV=S4&&IOR=0),CstateV=S4a)
11	0,1,2,3	Assert_implication(Lst(CstateV=S4a),CstateV=S0)
12	0,1,2,3	Assert_never(CstateV=S5)
13	0,1,2,3	Assert_implication(CstateV=S0,Ald=0)
14	0,1,2,3	Assert_implication(CstateV=S0,Dld=0)
15	0,1,2,3	Assert_implication(CstateV=S0,Dst=0)
16	0,1,2,3	Assert_implication(CstateV=S1,Ald=1)
17	0,1,2,3	Assert_implication(CstateV=S1,Dld=0)
18	0,1,2,3	Assert_implication(CstateV=S1,Dst=0)
19	0,1,2,3	Assert_implication(CstateV=S2,Ald=1)
20	0,1,2,3	Assert_implication(CstateV=S2,Dld=0)
21	0,1,2,3	Assert_implication(CstateV=S2,Dst=0)
22	0,1,2,3	Assert_implication(CstateV=S3,Ald=0)
23	0,1,2,3	Assert_implication(CstateV=S3,Dld=0)
24	0,1,2,3	Assert_implication(CstateV=S3,Dst=0)
25	0,1,2,3	Assert_implication(CstateV=S3a,Ald=0)
26	0,1,2,3	Assert_implication(CstateV=S3a,Dld=1)

27	0,1,2,3	Assert_implication(CstateV=S3a,Dst=0)
28	0,1,2,3	Assert_implication(CstateV=S4,Ald=0)
29	0,1,2,3	Assert_implication(CstateV=S4,Dld=0)
30	0,1,2,3	Assert_implication(CstateV=S4,Dst=0)
31	0,1,2,3	Assert_implication(CstateV=S4a,Ald=0)
32	0,1,2,3	Assert_implication(CstateV=S4a,Dld=0)
33	0,1,2,3	Assert_implication(CstateV=S4a,Dst=1)
34	0,1	Assert_implication(CstateV=S2&&ADDR(15:12)=1010&&ADDR(0)=0, Wem=1&&Rem=0)
35	0,1	Assert_implication(CstateV=S2&&ADDR(15:12)=1010&&ADDR(0)=1, Wem=0&&Rem=1)
36	0,1	Assert_implication(CstateV=S2&&ADDR(15:12)!=1010,Wem=0&&Rem=0)
37	2	Assert_implication(CstateV=S3a,Lst(DIN)=reg2adder)
38	3	Assert_implication(CstateV=S4a,Lst(adder2reg)=DOUT)

La seconde série d'assertions décrit le comportement du comparateur et de l'additionneur du chemin de données. Le Tableau 10 montre les quatre configurations de la sonde OCHP nécessaires à la vérification de la seconde série d'assertions.

Tableau 10 : Configuration de la OCHP du bus ISA pour la seconde série d'assertions

Configuration #0																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	CStateV			ALE_P	Wem	Rem						ADDR(15:12)				ADDR(0)
PIntSignal OCHP	5	4	3	54	61	60						53	52	51	50	38
Configuration #1																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	CStateV			Ald	Wem	Rem	IOR					ADDR(15:12)				ADDR(0)
PIntSignal OCHP	5	4	3	59	61	60	56					53	52	51	50	38
Configuration #2																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	CStateV				Wem	Rem	IOR									
PIntSignal OCHP	5	4	3		61	60	56									
Configuration #3																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	reg2adder(7:0)								adder2reg(7:0)							
PIntSignal OCHP	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6

Dans les trois premiers cas, le signal CStateV permet de suivre l'état du contrôleur, pour vérifier si les changements d'états correspondent à la machine à états illustrée à la Figure 27 et pour que le *Event Generator* puisse changer la zone échantillonnée en fonction de l'état du circuit.

La configuration #0 comprend en plus les signaux ALE_P, Wem, Rem et une partie du bus ADDRESS_P. Cette configuration permet de vérifier si le contrôleur passe bien de l'état S0 à l'état S1 lorsque ALE_P=1 et permet également de vérifier si les signaux Wem et Rem changent en fonction de l'adresse présente sur ADDRESS_P.

La configuration #1 comprend les signaux Ald, Wem, Rem, IOR et une partie du bus ADDRESS_P. Cette configuration permet de vérifier si le contrôleur entre dans l'état de lecture du bus ISA (S4) lorsque le signal IOR=0 et que ADDRESS_P=0xA001.

La configuration #2 comprend les signaux Wem, Rem et IOR. Cette configuration permet de vérifier si la machine à états passe de l'état S4 à S4A lors d'une lecture du bus.

La configuration #3 comprend les signaux d'entrée et de sortie de l'additionneur. Cette configuration permet de vérifier si l'additionneur ajoute effectivement 1 à la donnée présente à son entrée.

Le générateur d'événements doit gérer les changements de configuration en temps réel pendant le fonctionnement du circuit. Pour ce faire, une machine à quatre états, soit un par configuration, a été créée. Elle réagit selon le schéma montré à la Figure 30.

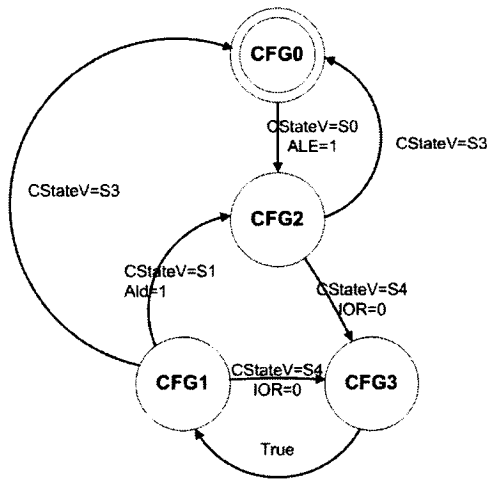

Figure 30 : Machine à états du générateur d'événements de la seconde série d'assertions

Le Tableau 11 décrit les assertions de la deuxième série, avec le numéro de la configuration où elles sont vérifiées. Les assertions 0 à 2 vérifient si les signaux de contrôle internes Wem et Rem prennent la bonne valeur selon l'état du contrôleur et

l'adresse présente sur le bus ADDRESS_P. L'assertion 3 vérifie si l'additionneur ajoute bien 1 à la valeur présente au registre d'entrée.

Tableau 11 : Seconde série d'assertions du bus ISA

Numéro	Vérifiée lorsque PconfID =	Assertion
0	0,1	Assert_implication(CstateV=S2&&ADDR(15:12)=1010&&ADDR(0)=0, Wem=1&&Rem=0)
1	0,1	Assert_implication(CstateV=S2&&ADDR(15:12)=1010&&ADDR(0)=1, Wem=0&&Rem=1)
2	0,1	Assert_implication(CstateV=S2&&ADDR(15:12)!=1010,Wem=0&&Rem=0)
3	3	Assert_implication(PconfID=3,reg2adder+1=adder2reg)

La dernière série d'assertions décrit le comportement des bus internes du chemin de données. Le Tableau 12 montre les trois configurations de la sonde OCHP nécessaires à la vérification de la troisième série d'assertions.

Tableau 12 : Configuration de la OCHP du bus ISA pour la troisième série d'assertions

Configuration #0																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	CStateV			IOW	IOR											
PIntSignal OCHP	5	4	3	55	56											
Configuration #1																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	DIN(7:0)								reg2adder(7:0)							
PIntSignal OCHP	37	36	35	34	33	32	31	30	21	20	19	18	17	16	15	14
Configuration #2																
PDebugData	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Signal bus ISA	adder2reg(7:0)								DOUT(7:0)							
PIntSignal OCHP	13	12	11	10	9	8	7	6	29	28	27	26	25	24	23	22

La configuration #0 comprend les signaux CStateV, IOW et IOR. Cette configuration permet de vérifier si le contrôleur va passer de l'état S3 à S3A ou de S4 à S4A (écriture ou lecture du bus).

La configuration #1 comprend les signaux DIN et reg2adder. Cette configuration permet de vérifier si la donnée présente à l'entrée du bus est bien écrite dans le registre d'entrée du chemin de données lors d'une écriture du bus.

La configuration #2 comprend les signaux adder2reg et DOUT. Cette configuration permet de vérifier si la donnée présente à la sortie de l'additionneur est bien écrite dans le registre de sortie lors d'une lecture du bus.

Le générateur d'événements doit gérer les changements de configuration en temps réel pendant le fonctionnement du circuit. Pour ce faire, une machine à trois états, soit un par configuration, a été créée. Elle réagit selon le schéma montré à la Figure 31.

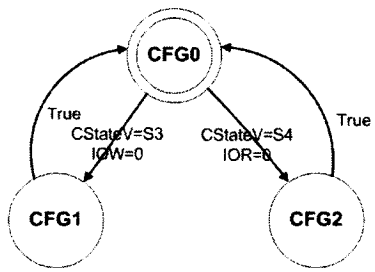

Figure 31 : Machine à états du générateur d'événements de la troisième série d'assertions

Le Tableau 13 décrit les assertions de la troisième série, avec le numéro de la configuration où elles sont vérifiées. Pour les assertions 0 et 1, les valeurs de DIN et de adder2reg, respectivement, ont été pipelinées, pour comparer avec la dernière valeur et non avec la valeur actuelle. L'assertion 0 vérifie si la donnée présente sur le bus DIN est bien écrite dans le registre d'entrée lors d'une écriture. L'assertion 1 vérifie quant à elle si la donnée présente sur à la sortie de l'additionneur est bien écrite dans le registre de sortie lors d'une lecture.

Tableau 13 : Troisième série d'assertions du bus ISA

Numéro	Vérifiée lorsque PconfID =	Assertion
0	1	Assert_implication(PconfID=1,Lst(DIN)=reg2adder)
1	2	Assert_implication(PconfID=2,Lst(adder2reg)=DOUT)

4.2.3 Application de la méthode à la détection de mutants sur simulateur

Le modèle de bus ISA a subi une légère modification, localisée dans un secteur précis du circuit, pour démontrer la validité de la méthode de validation présentée. Deux versions modifiées du circuit, appelées des circuits mutants, ont servi à tester le système. Cette sous-section présente la méthode de génération des mutants utilisée, suivie des résultats de simulation de la détection et la localisation des erreurs injectées.

4.2.3.1 Création des mutants

Les mutants ont été générés en roulant un script sur le fichier VHDL de synthèse du modèle de bus ISA. Ce script, qui apparaît en annexe, effectue les opérations suivantes :

- Déterminer quels blocs logiques du Spartan3^{TM} font partie du modèle de bus et lesquels font partie de la sonde OCHP ajoutée à ce dernier.
- Choisir au hasard un des blocs logiques appartenant au modèle de bus et en modifier la logique en changeant la configuration de sa « look-up table » (LUT).

De cette façon, chaque mutant ne comporte qu'une seule modification, précisément localisée et qui affecte le fonctionnement normal du modèle. Étant donné le caractère aléatoire des mutants, certains d'entre eux affectaient de manière trop fondamentale le fonctionnement du bus ISA pour être utiles à la démonstration de la méthode de validation. En effet, dans certains cas, le circuit ne fonctionnait tout simplement plus, rendant la détection d'erreurs impossible. Dans d'autres cas, un bit restait collé à une valeur fixe à une interface externe, une erreur aisément détectable sans l'outil de vérification. Cette sous-section s'attarde donc à deux mutants, qui affectent respectivement les zones suivantes du modèle ISA :

- Mutant #1 : Additionneur du chemin de données
- Mutant #4 : Comparateur du chemin de données

4.2.3.2 Détection d'erreurs sur simulateur

Pour la détection des erreurs induites par les deux mutants, le ACF a tout d'abord été configuré par l'écriture dans les registres du *Configuration Manager* et l'envoi d'une commande pour la programmation de la sonde OCHP. Une fois la programmation

complétée, la *Assertion RAM* a été configurée pour réagir à l'échec d'une des assertions, en mode « One Shot » et avec un délai de 4 coups d'horloge, pour stocker les données de déverminage de la première erreur de fonctionnement. La Figure 32 montre l'écriture dans les registres du *Configuration Manager* (signaux *dut/we_n*, *dut/regaddr* et *dut/regdatain*), alors que la Figure 33 montre l'envoi de la commande de programmation de la OCHP, suivie du début de l'envoi des paquets de configuration (signaux *dut/psop, dut/peop* et *dut/pcfgdata*), pour la première série d'assertions (voir section 4.2.2.2). La procédure de détection des erreurs pour chaque mutant est décrite ci-après.

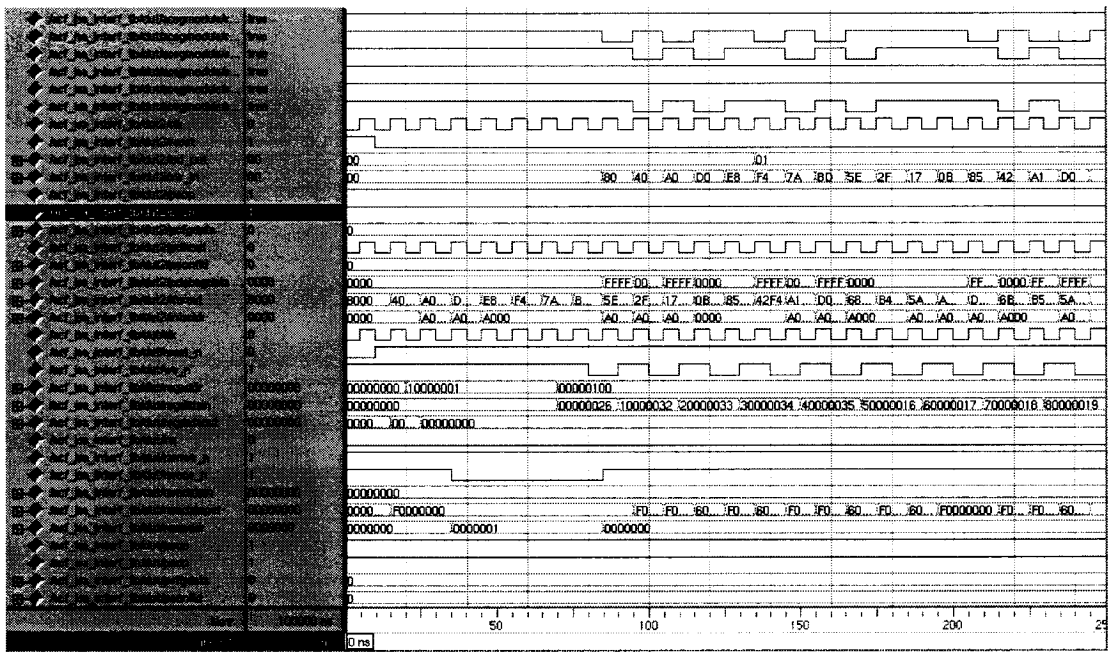

Figure 32 : Écriture dans les registres du *Configuration Manager* du ACF

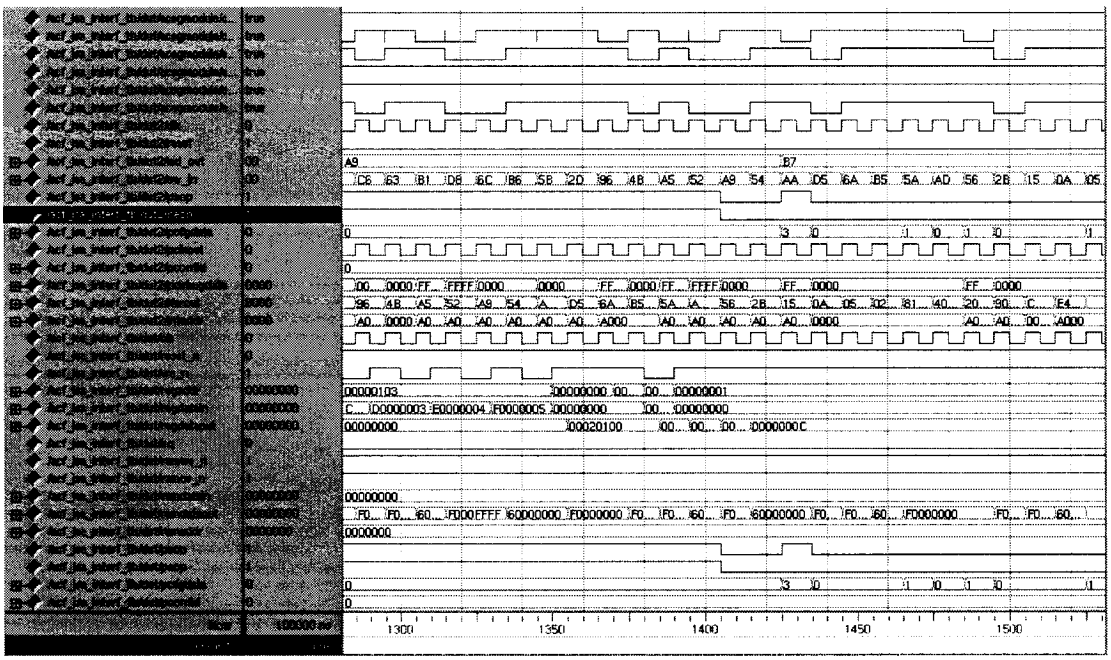

Figure 33 : Envoi de la configuration à la OCHP par le ACF

- Mutant #1 : Additionneur du chemin de données

L'erreur induite dans le mutant #1 affectait l'additionneur du chemin de données, qui, au lieu d'ajouter 1 à la donnée écrite dans le registre d'entrée du bus ISA, y soustrayait 1. Cette erreur a été détectée par la seconde série d'assertions (voir section 4.2.2.2). Une fois la OCHP configurée et la *Assertion RAM* armée, les vérificateurs d'assertions ont procédé à la détection, et par la suite, le stockage des données de déverminage a démarré grâce au *Event Generator*. La Figure 34 montre la détection de cette erreur dans le bloc *acegmodule* (le signal *dut/acegmodule/acvalid*, passe de 0xF à 0x7, correspondant au déclenchement du 3ème détecteur d'assertions). On remarque que la donnée recueillie au registre *addr2reg* (registre de sortie du bus ISA) n'est pas égale à la donnée du registre *reg2addr* plus 1, mais moins 1. (0x1D au lieu de 0x1F). Ces données sont extraites des données du signal *dut/acegmodule/pdebugdata*. La Figure 35 confirme l'erreur de fonctionnement en illustrant le fonctionnement du circuit normal. On remarque sur les deux figures que la zone échantillonnée change dynamiquement (signal *dut/acegmodule/pconfid*, troisième à partir du haut sur le chronogramme) en fonction des données de déverminage.

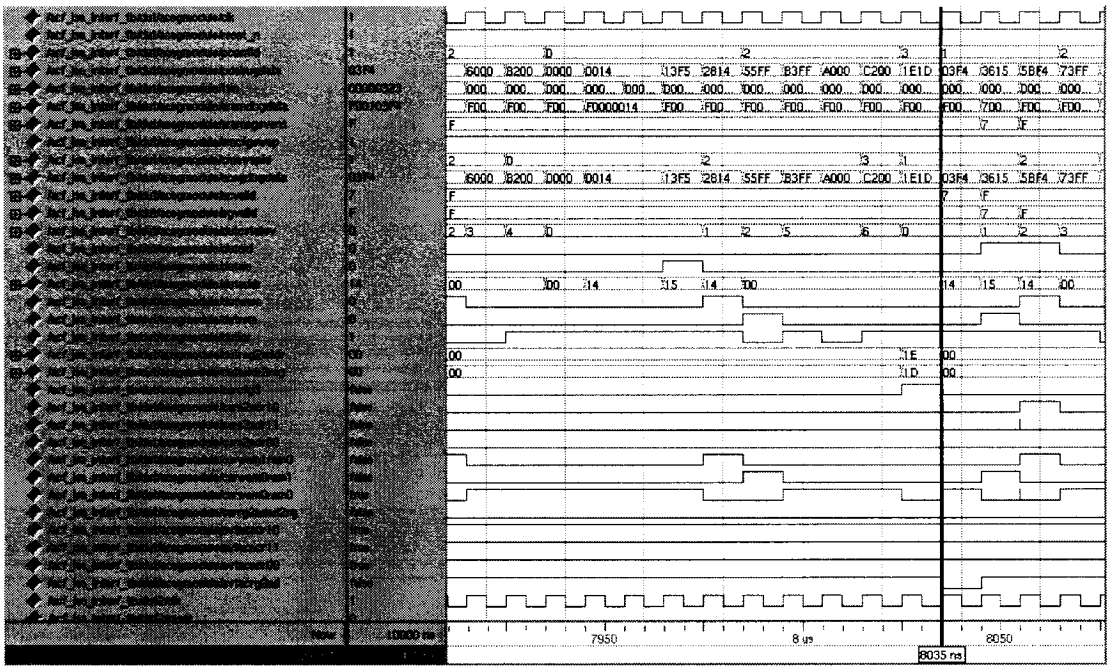

Figure 34 : Détection par les *Assertion Checkers* de l'erreur induite par le mutant #1

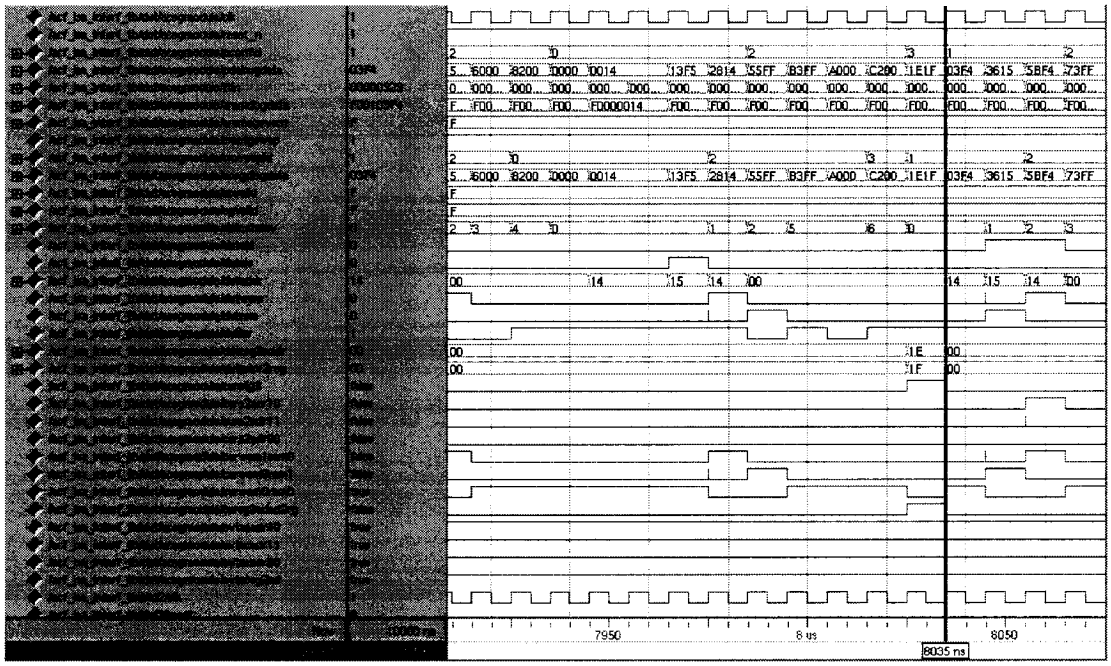

Figure 35 : Fonctionnement normal de l'additionneur du bus ISA

La détection de cette erreur par le vérificateur d'assertions numéro 3 de la seconde série d'assertions interrompt le stockage des données de déverminage après 10 coups d'horloge. On remarque dans la Figure 36 que le signal de déclenchement de la *Assertion RAM*, armé jusqu'au temps 8065 ns (signal *dut/regdataout*) à 0x00000015, se désarme par la suite, grâce à la détection de l'erreur de fonctionnement. L'identification du détecteur d'assertion déclenché et les données de déverminage stockées permettent de localiser la source de l'erreur dans l'additionneur du chemin de données, car :

- La donnée du registre *reg2adder* n'est pas augmentée de 1 mais retranchée de 1 une fois dans le registre *adder2reg*;
- Le vérificateur d'assertion numéro 3 a échoué, ce qui signifie que la valeur de sortie de l'additionneur ne respecte pas les règles.

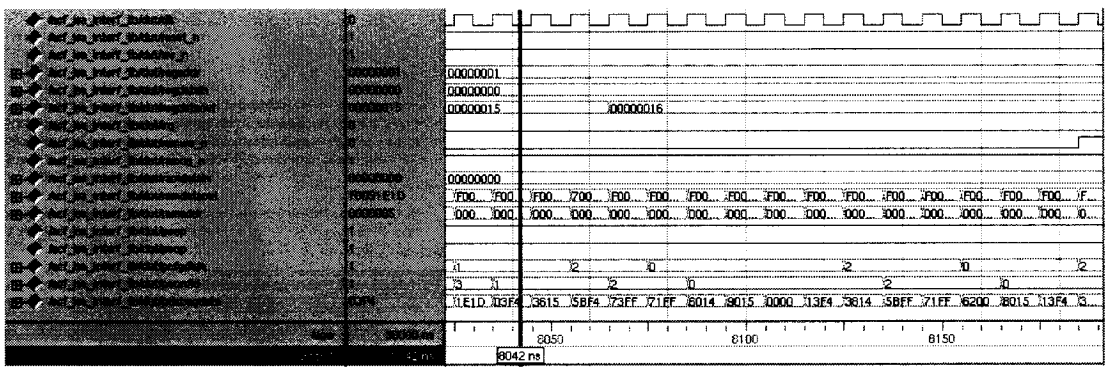

Figure 36 : Stockage des données de déverminage pour la localisation du mutant #1

- Mutant #4 : Comparateur du chemin de données.

L'erreur induite dans le mutant #4 affectait le comparateur du chemin de données, qui ne détecte plus la validité d'une transaction d'écriture sur le bus. Cette erreur a été détectée par la première série d'assertions (voir section 4.2.2.2). Une fois la OCHP configurée et la *Assertion RAM* armée, les vérificateurs d'assertions ont procédé à la détection, et par la suite, le stockage des données de déverminage a démarré grâce au *Event Generator*. La Figure 37 montre la détection de cette erreur dans le bloc *acegmodule* (le signal *dut/acegmodule/acvalid*, passe de 0x7FFFFFFFFF à

0x7BFFFFFFFF, correspondant au déclenchement du 34ème détecteur d'assertions). On y remarque, au temps 7535 ns, que malgré que la machine à états du bus ISA (signal *dut/acegmodule/dutcstatev*) soit à l'état 0x2 et que l'adresse sur le bus *dut/acegmodule/dutaddr* soit à 0x14, le signal *dut/acegmodule/dutwem* ne monte pas à 1. Ces données sont extraites des données du signal *dut/acegmodule/pdebugdata*. La Figure 38 confirme l'erreur de fonctionnement en illustrant le fonctionnement du circuit normal. On remarque sur les deux figures que la zone échantillonnée change dynamiquement (signal *dut/acegmodule/pconfid*) en fonction des données de déverminage. Le fait que l'assertion numéro 34 ait détecté l'erreur permet une localisation précise de la source, car l'activation du signal *wem* en fonction de l'adresse du bus ISA est commandé par le comparateur du chemin de données.

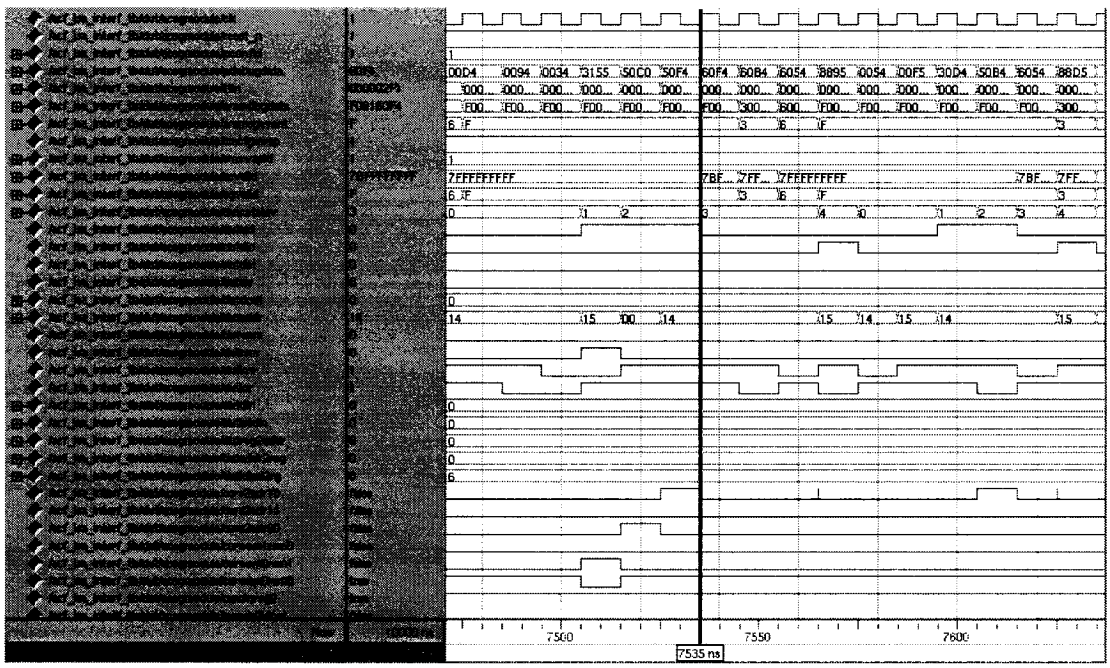

Figure 37 : Détection par les *Assertion Checkers* de l'erreur induite par le mutant #4

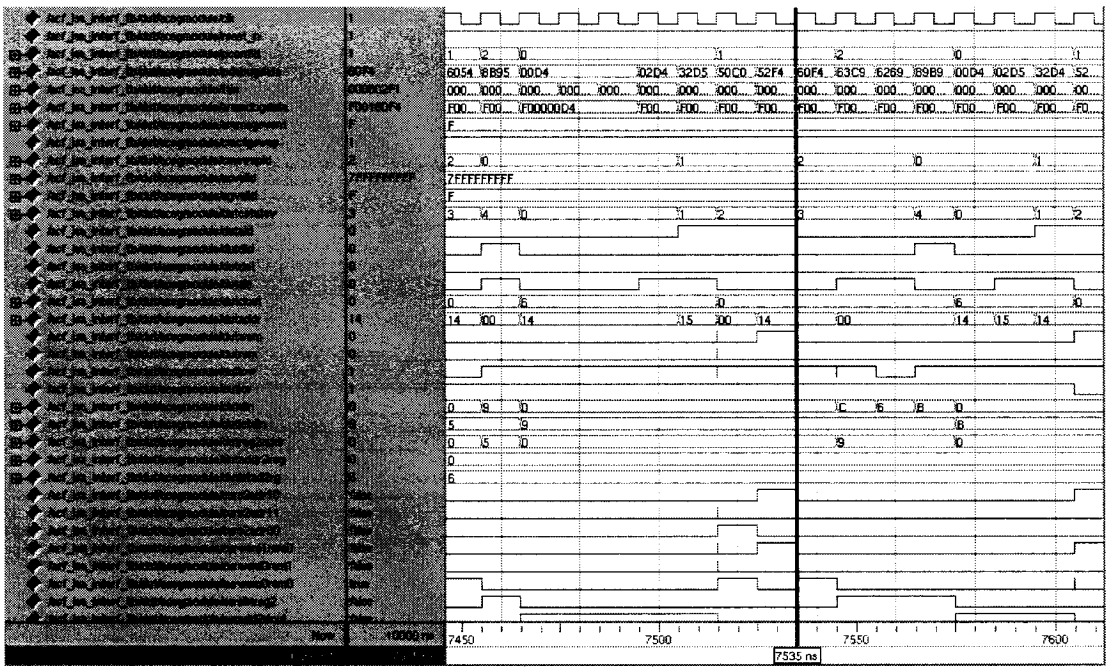

Figure 38 : Fonctionnement normal du comparateur du bus ISA

4.2.4 Application de la méthode à la détection de mutants sur plate-forme de prototypage

Un des mutants générés à l'étape précédente a été utilisé pour appliquer la méthode de détection d'erreurs sur une plate-forme de prototypage. Cette sous-section décrit le circuit de prototypage utilisé et montre les résultats pratiques obtenus.

4.2.4.1 Description de la plate-forme de prototypage

Pour implanter et valider le ABRD en environnement réel, deux cartes de prototypage de la compagnie Digilent [40], comportant chacune un FPGA Spartan 3S200™ (item 1 de la Figure 39) de Xilinx ont été connectés ensemble, par le connecteur A2 (item 20). Dans l'une des cartes, appelée par après carte ISA, le modèle de bus ISA a été implanté, alors que la seconde carte (carte ACF) contenait le ACF. L'affichage composé de quatre modules 7 segments (item 10) servait d'interface visuelle pour l'usager, alors que l'entrée

de données s'effectuait à l'aide d'un clavier standard d'ordinateur, connecté au port PS/2 de la carte (item 9). Les deux cartes fonctionnent avec un oscillateur de 50MHz. L'horloge du ACF provenait de l'oscillateur de la carte ISA, via le signal PClkOut. L'émulation de transactions sur le bus ISA s'effectuait à l'aide d'un LFSR synthétisé dans le FPGA correspondant, alors que les données entrantes sur le bus ISA étaient données par la valeur des interrupteurs de la carte de prototypage (item 11). La remise à zéro de chaque carte pouvait s'effectuer avec un des boutons poussoirs (item 13).

Pour effectuer la validation, le mutant #1, décrit à la section 4.2.3.1 a été synthétisé dans la carte ISA, alors qu'un ACF contenant la seconde série d'assertions (voir section 4.2.2.2) a été synthétisé dans la carte ACF. Le code VHDL synthétisable du bus ISA et du ACF (avec l'interface PS/2 et l'affichage 7 segments) apparaissent en annexe.

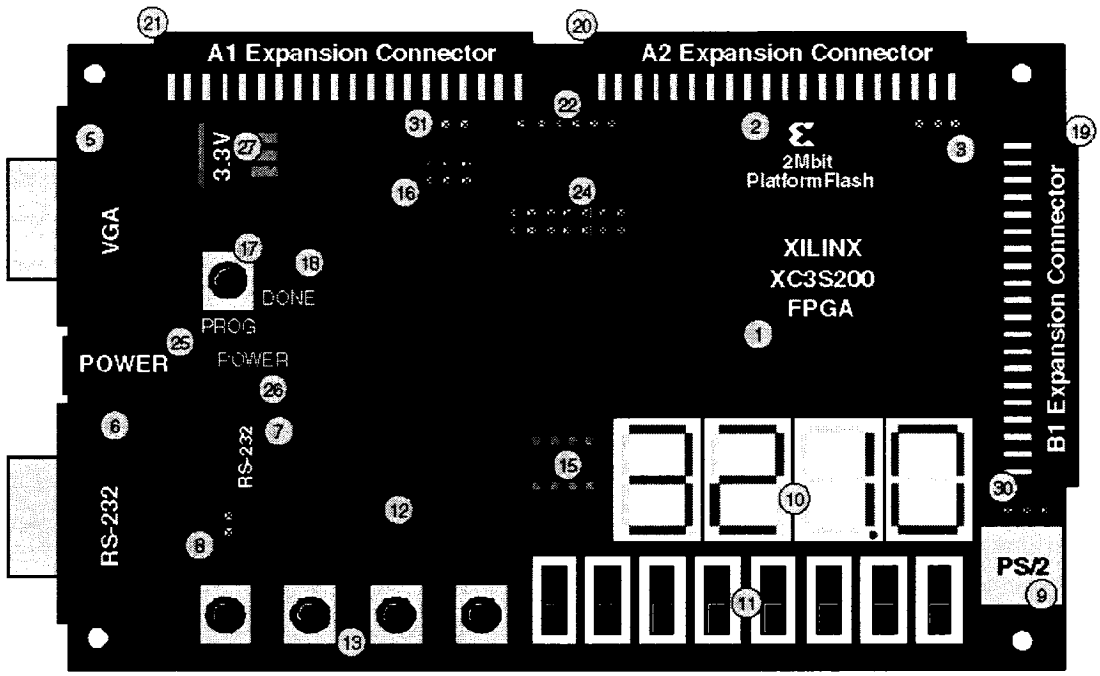

Figure 39 : Plateforme de prototypage de Digilent [40]

Le ACF, avec les interfaces usager et la seconde série d'assertions, prenait les ressources suivantes du Spartan 3S200™ (extrait du rapport de placement et routage):

```
Device utilization summary:
```

```
Number of External IOBs            47 out of 173    27%
   Number of LOCed External IOBs   47 out of 47    100%

Number of Slices                 1418 out of 2880   49%
   Number of SLICEMs               76 out of 960     7%

Number of BUFGMUXs                  1 out of 8      12%
```

4.2.4.2 Localisation d'erreurs sur la plate-forme de prototypage

La configuration initiale du ACF a impliqué l'écriture (via le clavier PS/2) des registres du *Configuration Manager* avec les données décrites à la section 4.2.2.2). La *Assertion RAM* a été mise en mode « One Shot », avec un délai de stockage de 4 coups d'horloge et un stockage de 16 adresses mémoire (voir section 3.3.2.6). Ensuite, les vérificateurs d'assertions et les générateurs d'événements ont été remis à zéro et le signal de déclenchement de la *Assertion RAM* a été armé (section 3.3.2.1).

Le registre d'état du ACF (section 3.3.2.1) a été consulté jusqu'à la constatation que le signal de déclenchement n'était plus armé, ce qui signifiait qu'un vérificateur d'assertions s'est déclenché et que des données de déverminage se trouvent dans la *Assertion RAM*.

Pour localiser la source de l'erreur, il a tout d'abord fallu consulter les registres RAM Trigger Timestamp et RAM End of Storage Pointer (section 3.3.2.6), pour savoir le temps d'apparition de l'erreur et l'adresse de fin de stockage (qui permettait de déduire l'adresse de début de stockage, sachant la taille de stockage configurée). Les données de déverminage ont ensuite été lues dans la *Assertion RAM*, ce qui a permis de constater que le vérificateur d'assertions numéro 3 s'était déclenché. En consultant la donnée précédente, l'effet de l'erreur a pu être constaté, car on retrouvait la donnée du registre *reg2adder* et celle du registre *adder2reg*, comme lors de la simulation. Le fonctionnement du système en environnement réel a ainsi pu être démontré.

Chapitre 5

CONCLUSION

L'introduction de ce mémoire a présenté la problématique de la validation de circuits intégrés fonctionnant en environnement réel, à des fréquences de l'ordre des centaines de Mégahertz. Il y était mentionné que l'instrumentation sur puce constitue une méthodologie de déverminage efficace, mais que l'utilisation de cette technique comporte plusieurs difficultés, autant matérielles que logicielles. Le projet de recherche décrit dans ce mémoire avait pour objectif d'apporter une contribution au niveau des méthodes de validation des circuits fonctionnant en conditions réelles et comportant une sonde intégrée. Ce mémoire a tout d'abord présenté une revue de littérature concernant les techniques d'instrumentation sur puce existant à l'heure actuelle, ainsi qu'une présentation de la vérification matérielle au moyen d'assertions. La description des techniques d'instrumentation sur puce développées au cours des dernières années permet d'en faire ressortir les principes de base, ainsi que les limitations, que le travail présenté dans ce mémoire tente d'atténuer. La description de la vérification par assertions démontre quant à elle la puissance de ce moyen de description des fonctionnalités d'un circuit pour clarifier la spécification, éliminer les ambiguïtés et réutiliser à différents niveaux d'abstraction les outils de déverminage. L'utilisation de vérificateurs d'assertions synthétisables permet d'étendre le champ d'action de cette méthode à la validation de circuits dans un environnement réel.

La revue de littérature a été suivie d'une description de la méthode de validation et de déverminage « Assertion-Based Runtime Debugger » (ABRD), développée au GRM. La méthode présentée dans ce mémoire est une contribution originale permettant d'utiliser la clarté descriptive des assertions pour faciliter la détection et la localisation d'erreurs de fonctionnement d'un circuit intégré. Ce mémoire a décrit les deux principaux aspects du ABRD, soit la sonde matérielle proprement dite, la OCHP et le circuit de déverminage utilisant des assertions synthétisables, le ACF.

Deux circuits ont servi pour l'évaluation de la méthode de validation. En premier lieu, un processeur Cordic, développé au GRM, a servi de modèle pour évaluer l'impact de la sonde OCHP sur le fonctionnement d'un circuit réel. Après analyse des résultats de simulation analogique, il est possible de conclure que la sonde a un effet négligeable sur un tel processeur, fonctionnant à une fréquence de 250 MHz. Cette affirmation peut être appuyée par le fait qu'à la suite d'une simulation analogique de l'effet des capacités parasites sur l'intégrité des signaux internes du processeur, un effet non perceptible sur le temps de montée et de descente a été remarqué. Une simulation sur le même circuit, duquel seuls les blocs logiques et les fils de la OCHP ont été retirés, a permis de faire la comparaison de la forme d'onde des signaux critiques, avec et sans la sonde. Une telle sonde peut donc servir dans des circuits fonctionnant à des vitesses de plusieurs centaines de Mégahertz.

Un second circuit, soit un modèle de bus ISA synthétisable, a servi à démontrer l'efficacité du ACF pour détecter et localiser une erreur de fonctionnement induite volontairement. Le modèle du bus ISA, augmenté d'une sonde de type OCHP, a tout d'abord été synthétisé pour être implanté dans un FPGA Spartan 3S200™ de Xilinx. Le fichier de synthèse a ensuite été modifié pour insérer un mutant qui modifiait le fonctionnement d'une section spécifique du circuit. L'efficacité de trois séries d'assertions pour détecter et localiser précisément le mutant dans deux circuits modifiés a été démontrée sur un simulateur numérique. Dans les deux cas, les résultats de simulation démontrent clairement que les vérificateurs d'assertions synthétisés peuvent détecter une erreur de fonctionnement et en localiser la source. Le modèle de bus ISA ainsi que le ACF ont finalement été implantés sur une plate-forme de prototypage comportant deux FPGA Spartan 3S200™, pour prouver l'applicabilité de la méthode sur un circuit réel, fonctionnant à fréquence nominale. Cette plate-forme a détecté et aidé à localiser une erreur de fonctionnement induite dans un des circuits mutants simulés précédemment, ce qui a permis de démontrer que la méthode est applicable sur des modèles réels.

Plusieurs travaux peuvent découler de la méthode présentée dans ce mémoire. Une version modifiée du ACF pourrait faciliter le déverminage de circuits au moyen d'un

analyseur logique. La description plus claire du circuit au moyen d'assertions et le changement dynamique de la zone échantillonnée offerts par le ACF peuvent accélérer le déverminage de circuits intégrés. L'analyseur logique effectue quant à lui les fonctions de stockage et d'affichage des données de déverminage de façon plus efficace. Associer un ACF à un analyseur logique permettrait un prétraitement des données et la gestion des zones échantillonnées, pour améliorer l'utilisation de la bande passante de la sonde intégrée.

On peut également songer à déporter une partie des vérificateurs d'assertions ou des générateurs d'événements directement dans le circuit sous test. On pourrait ainsi diminuer le besoin d'un nombre élevé de signaux externes pour vérifier la fonctionnalité du circuit, déterminer les zones à échantillonner ou démarrer le stockage de données de déverminage. Cette solution pourrait également permettre d'éviter le routage de signaux à trop haute fréquence au travers de plots externes. Par contre, cette méthode a pour désavantage l'augmentation de la surface du circuit, mais un tel compromis peut être envisagé pour améliorer l'observabilité interne du circuit en utilisant un minimum de signaux externes.

Certains vérificateurs d'assertions synthétisables peuvent prendre beaucoup d'espace dans le ACF. On pourrait envisager de faire un traitement ultérieur des données de déverminage au moyen de vérificateurs d'assertions logiciels, pour diminuer le travail du FPGA vérificateur d'assertions et augmenter le nombre d'assertions appliquées au circuit sous test. Cette solution a toutefois le désavantage d'augmenter l'intervalle entre l'apparition d'une erreur de fonctionnement et sa détection. Une analyse plus poussée des impacts de ces solutions permettra de trouver le meilleur compromis entre l'observabilité, la bande passante et le temps de déverminage.

RÉFÉRENCES

[1] 0-IN DESIGN AUTOMATION. 2003. *Assertion-Based Verification for Complex Designs*. In. *0-In Website*. [En ligne]. http://www.0-in.com/whitepapers/Archer_Whitepaper.pdf. (Page consultée le 5 décembre 2004).

[2] ACCELERA. 2003. *OVL Reference Manual (June 2003)*. In. *Accelera OVL Technical Commitee Website*. [En ligne]. http://www.eda.org/ovl/docs/OVL_Ref_June2003.zip. (Page consultée le 5 décembre 2004).

[3] AGILENT TECHNOLOGIES. 2004. *Agilent Technologies B4655A FPGA Dynamic Probe Data Sheet*. In. *Agilent Website* [En ligne]. http://we.home.agilent.com/cgi-bin/bvpub/agilent/reuse/cp_ObservationLogRedirector.jsp?NAV_ID=-536893754.536883660.00&LANGUAGE_CODE=eng&CONTENT_KEY=403773&COUNTRY_CODE=US&CONTENT_TYPE=AGILENT_EDITORIAL. (Page consultée le 5 décembre 2004).

[4] AGILENT TECHNOLOGIES. 2004. *Frequently Asked Questions, B4655A FPGA Dynamic Probe*. In. *Agilent Website* [En ligne]. http://we.home.agilent.com/cgi-bin/bvpub/agilent/reuse/cp_ObservationLogRedirector.jsp?NAV_ID=-536893754.536883660.00&LANGUAGE_CODE=eng&CONTENT_KEY=457834&COUNTRY_CODE=US&CONTENT_TYPE=AGILENT_EDITORIAL. (Page consultée le 5 décembre 2004).

[5] ALDEC INC. 2003. *Riviera IPT Hardware Acceleration Platform*. In. *Aldec Website* [En ligne]. http://www.aldec.com/Riviera/riviera_ipt.htm. (Page consultée le 5 décembre 2004).

[6] ARM. 1999. *AMBA Specification Rev. 2.0*. In. *ARM Website* [En ligne]. http://www.arm.com/products/solutions/AMBA_Spec.html. (Page consultée le 5 décembre 2004).

[7] BANNATYNE, R. 1998 « Debugging Aids for Systems-on-a-Chip ». Northcon/98 Conference Proceedings, 107-111.

[8] BERGERON, J. 2003. Writing Testbenches : Functional Verification of HDL Models, Second Edition, Boston : Kluwer Academic Publishers. 475p.

[9] BILONG, C. XIAOLANG, Y. 2003 « Method of Using Shadow Registers in designing an on-chip Debug Unit of a Microprocessor ». Proceedings. 5th International Conference on ASIC, Volume 1 , 393-396.

[10] BROKISH, C. KERTIS, D. 2002. *Adapting traditional embedded debug strategies to SoC designs*, In. *EETimes*. [En ligne]. http://www.eetimes.com/in_focus/embedded_systems/OEG20020531S0030. (Page consultée le 17 juillet 2003)

[11] CANTIN, M.-A. LAFRANCE, L.-P. MBAYE, M.M. REGIMBAL, S. TAILLEFER, F. SAVARIA, Y. 2004. *Guide pour le Placement et Routage*, In. *Groupe de Recherche en Microélectronique*.

[12] CERNY, E. 2004 « Design for Verification with System Verilog ». NEWCAS 2004 Tutorial, 45 p.

[13] DATTA, K. DAS, P.P. 2004 « Assertion Based Verification Using HDVL ». Proceedings. 17th International Conference on VLSI Design, 319-325.

[14] DE BOER, W. VERMEULEN, B. 2004 « Silicon Debug : Avoid Needless Respins ». IEEE/CPMT/SEMI 29th International Electronics Manufacturing Technology Symposium, 277-281.

[15] DE LA TORRE, E. GARCIA, M. RIESGO, T. TORROJA, Y. UCEDA, J. 2002 « Non-intrusive debugging using the JTAG interface of FPGA-based prototypes ». Proceedings of the 2002 IEEE International Symposium on Industrial Electronics, Volume 2 , 666-671.

[16] DRECHSLER, R. 2003 « Synthesizing Checkers for On-line Verification of System-on-Chip Designs ». Proceedings of the 2003 International Symposium on Circuits and Systems, Volume 4 , 748-751.

[17] EL SHOBAKI, M. 2002 « On-Chip Monitoring of Single- and Multiprocessor Hardware Real-Time Operating Systems ». 8th International Conference on Real-Time Computing Systems and Applications.

[18] FOSTER, H.D. 2003. *Property Specification : The key to an Assertion-Based Verification Platform*, In *Electronic Design Automation Website*. [En ligne]. http://www.eda.org/edps/edp03/submissions/hf_edps03.pdf. (Page consultée le 5 décembre 2004)

[19] FOSTER, H.D. KROLNIK, A.C. LACEY, D.J. 2003. *Assertion-Based Design*. Boston : Kluwer Academic Publishers. 363p.

[20] GOERING, R. 2003. *0-In's assertion compiler is multilingual*, In. *EEDesign*. [En ligne]. http://www.eedesign.com/news/OEG20031110S0047. (Page consultée le 2 janvier 2004)

[21] GRAHAM, P.S. 2001. *Logical Hardware Debuggers for FPGA-Based Systems*, Brigham Young University. 246p.

[22] HUTCHINGS, B. NELSON, B. 1999 « Developing and Debugging FPGA Applications in Hardware with JHDL ». Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers, Volume 1, 554-558.

[23] IEEE-ISTO. 1999. *The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface.* In *IEEE-ISTO Nexus 5001 Website* [En ligne] http://www.nexus5001.org/standard2.html. (Page consultée le 5 décembre 2004)

[24] IEEE. 2001. *1149.1-2001 Test Access Port and Boundary Scan Architecture.* Piscataway, NJ, IEEE.

[25] LAFRANCE, L.-P. MBAYE, M.M. REGIMBAL, S. 2004. *Algorithme de Cordic*, In. *Groupe de Recherche en Microélectronique Website*. [En ligne]. http://www.grm.polymtl.ca/~savaria/ele4304/laboratoires/projet_ele4304_h2004.pdf. (Page consultée le 3 avril 2004)

[26] LEATHERMAN, R. ABLEIDINGER, B. STOLLON, N. 2003. *Method offers snapshot of SoC operation*, In. *EETimes*. [En ligne]. http://www.eetimes.com/story/OEG20020418S0017. (Page consultée le 17 juillet

2003)

[27] MACNAMEE, C. HEFFERNAN, D. 2000. « Emerging on-chip debugging techniques for real-time embedded systems », IEEE Computing & Control Engineering Journal. 11 :6. 295-303.

[28] MARINISSEN, E.J. GOEL, S.K. LOUSBERG, M. 2000 « Wrapper Design for Embedded Core Test ». Proceedings International Test Conference, 911-920.

[29] MAXFIELD, C. 2002. *Walking the assertion maze*, In. *EEDesign*. [En ligne]. http://www.eedesign.com/printableActicle?doc_id=OEG20020815S0035. (Page consultée le 7 octobre 2002)

[30] MOERMAN, E. BOCQ, S. VERFAILLIE, J. 2003 « Debug Architecture for System on Chip taking full advantage of the Test Access Port ». Proceedings The Eighth IEEE European Test Workshop, 155-159.

[31] OKEEFFE, H. 2000. *IEEE-ISTO 5001TM-1999, The Nexus 5001 ForumTM Standard providing the Gateway to the Embedded Systems of the Future*, In. *Ashling Microsystems Website*. [En ligne]. http://www.ashling.com/pdf_papers/NexusGateway.PDF. (Page consultée le 5 décembre 2004)

[32] RASHINKAR, P. PATERSON, P. SINGH, L. 2001. *System-on-a-chip Verification, Methodology and Techniques*. Boston : Kluwer Academic Publishers. 372p.

[33] SYNOPSYS. 2002. Assertion-Based Verification. In. *Synopsys Website*. [En ligne]. http://www.synopsys.com/products/simulation/assertion_based_wp.pdf. (Page consultée le 5 décembre 2004)

[34] TABBARA, B. HSU, Y. BAKEWELL, G. SANDLER, S. 2003 « Assertion-Based Hardware Debugging ». DVCON 2003.

[35] VAN ROOTSELAAR, G.J. VERMEULEN, B. 1999 « Silicon Debug : Scan Chains Alone Are Not Enough ».Proceedings International Test Conference, 1999, 892-902.

[36] VARMA, P. BHATIA, S. 1998 « A Structured Test Re-Use Methodology for Core-Based System Chips ».Proceedings International Test Conference, 1998, 294-302.

[37] VERMEULEN, B. VAN ROOTSELAAR, G.J. 2000 « Silicon Debug of a Co-Processor Array for Video Applications ». Proceedings IEEE International High-Level Design Validation and Test Workshop, 2000, 47-52.

[38] WALTERS, G. KING, E. KESSINGER, R. 1998 « Processor Design and Implementation of Real-Time Testing of Embedded Systems ». Proceedings, 17th Digital Avionics Systems Conference, Volume 1 , B44/1-B44/8.

[39] WESTE, N.H.E. ESHRAGHIAN, K. 1994. *Principles of CMOS VLSI Design : A Systems Perspective, Second Edition.* Reading : Addison-Wesley Publishing Company. 702p.

[40] XILINX. 2004. Spartan 3 Starter Kit Board User Guide. In. *Digilent Website.* [En ligne]. http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD-rm.pdf. (Page consultée le 17 septembre 2004)

[41] YORK, R. 1999. *Real Time Debug for System-on-Chip Devices*, In. *ARM Website.* [En ligne]. http://www.arm.com/Pro+Peripherals/ArchExt/RTDebug. (Page consultée le 5 décembre 2004)

[42] ZHANG, W. 2001. *ISA bus interface design (ISA.vhd)*, In. *UCR Website.* [En ligne]. www.cs.ucr.edu/content/esd/labs/tutorial. (Page consultée le 7 septembre 2004)

ANNEXES

ANNEXE A : Article ISCAS 2004

ASSERTION-BASED ON-LINE VERIFICATION AND DEBUG ENVIRONMENT FOR COMPLEX HARDWARE SYSTEMS

K. Peterson[1] and Y. Savaria[1]

[1]Electrical and Computer Engineering Dept., École Polytechnique de Montréal, P.O. Box 6079, Station Centre-Ville, Montréal, Québec, Canada, H3C 3A7

ABSTRACT

This paper describes a debug environment for high performance integrated circuits and systems, running in real-world conditions. With the proposed environment, debug data is collected by a built-in debug hardware module (or integrated probe) and transferred to an external debugger using a dedicated debug port. The external debugger, composed of a FPGA and a processor, uses real-time assertion-based verification techniques to ensure that the system acts according to its specifications. Dynamic changes in the probe configuration allow higher monitoring resolution of critical parts of the circuit or system and improve the use of the debug port bandwidth. This paper discusses advantages and limitations of this technique.

1. INTRODUCTION

Validation and debug of high speed integrated circuits (ICs) and systems running at frequencies over 100MHz poses significant challenges to engineering teams. A major challenge is to identify and locate the source of a bug when validating a prototype. Bugs tracked at this step in the development flow can be intermittent and usually occur only when the system is running at full-speed [1]. Moreover, a bug affecting an internal part of the system can take many cycles to appear at its interfaces [2].

Efficient debug methodologies often involve the use of on-chip instrumentation, composed of embedded circuitry that provides access to the inner signals and interfaces of a system. Basic instrumentation is composed of one or more sets of blocks used to collect and aggregate selected internal data that allows tracing important signals over time. A time stamping technique is applied, to allow accurate reconstitution of the data flow by the debug tool. In more sophisticated methods, data compression is performed prior to transfer to the debugger [3]. Data filtering selects signals to be monitored, along with suitable hardware triggers, recognizable events that trigger begin and end of data capture. Data filtering also limits collection to the targeted time frames. These features are important to ensure that only relevant information is transmitted to the debugger [1].

Several difficulties have to be overcome when adding debug capabilities to a design. Of course, it is not possible to connect a logic analyzer on internal signals in a chip. Pins of the IC can be used, but it may force routing signals off-chip at maximum frequency. In addition, many pins must be added to the chip package and, if the final chip does not include these external connections, the system behavior can be different. This last problem is called the "probe-effect". It is caused by the differences in timing induced by the addition of an intrusive monitor. Over the last decade, researchers have proposed several approaches to create non-intrusive monitors for highly integrated hardware systems [4].

The next section presents related work in on-chip debug techniques and describes some commonly used assertion-based verification methods. It is followed in Section 3 by a description of the proposed assertion-based on-line verification method. Section 4 discusses the expected advantages and the limitations to be overcome when implementing the method. Finally, we conclude by suggesting steps that would further validate the proposed methodology.

2. RELATED WORK

Existing on-line on-chip debugging techniques mainly rely on monitoring bus transactions in

ANNEXE A : Article ISCAS 2004

embedded System-on-Chip devices (SoC). These solutions allow tracing code execution in an embedded processor with event recognition hardware, used to synchronize a debugger with the system [4]. Some existing methods even allow to control software execution [3][7]. Other methods include hardware breakpoints to monitor bus activity in a SoC, and interrupt an embedded CPU when the bus contains specific values. Complex sequences can also be recognized with a state sequencer [8].

The Nexus 5001 Forum™ standard, developed by the IEEE Industry Standards and Technology Organization (ISTO), is an attempt to expand and standardize the diverse approaches to on-chip debugging. This standard provides guidelines for embedded processors developers to add the capability to trace instructions and data with a minimum impact on the embedded software execution. Pin interface standards, as well as a debug data transfer protocol are also defined. The data can be transferred using a standard JTAG port or using a higher bandwidth auxiliary debug port [5]. The standard is designed to be processor and architecture independent, and to support multi-core or multi-processor designs [6].

None of the solutions described above take advantage of assertion-based verification techniques. On-chip synthesized checkers have been proposed [9] and commercial tools exist that use assertions for off-line analysis of debug data [10] or to assist verification in hardware accelerators [12][13], but no on-line hardware debugger uses assertions to track and identify bugs in real-time. An assertion (also called "property") is a statement about a design's intended behavior that is expected to always be true [2][11]. For example, an assertion could be: when signal A is set to 1, signal B must be cleared to 0 within 2 clock cycles. An interface or a component in a design can be described using many such statements.

Assertions have been used in software for many years and have recently been introduced in hardware verification. At this time, assertion-based verification is mostly performed during simulation and while using formal (mathematical theorem proving) tools [14]. Special assertion languages are now being developed, such as the Property Specification Language (PSL) and Open Verification Library (OVL). They are both progressing towards standardization by Accellera. These languages allow translating a functional specification into powerful and expressive properties [15].

3. PROPOSED METHOD

3.1. Overview

The solution we propose is called an Assertion-Based Runtime Debugger (ABRD). It uses special hardware added to the design ("On-Chip Hardware Probe") that connects to debug hardware resources composed of a FPGA (the "Assertion Checker FPGA") and a processor. Several advanced FPGAs contain a processor on-chip. Using such a FPGA would allow fitting all the debug resources in one device.

Figure 1 illustrates the architecture of the ABRD. All signals to be monitored in an IC are connected to the On-Chip Hardware Probe (OCHP). The OCHP is configurable and can store many different configuration patterns. The configuration can be changed dynamically by the Assertion Checker FPGA (ACF), so only the signals needed at a specific time are transferred to the debug port. The ACF uses a clock from the device under test (DUT) to update a timer, in order to timestamp the debug data. The processor, connected to an external terminal via a UART, can access the ACF to program it, change the configuration settings, check the state of the various assertion checkers, get statistics on the number of passing and failing assertions, and obtain debug data to track the source of a bug. Some assertions can also be verified off-line with the data stored in the ACF RAM.

ANNEXE A : Article ISCAS 2004

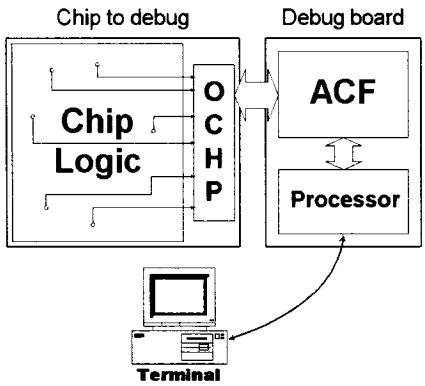

Figure 1. Architecture of the debug environment

3.2. On-Chip Hardware Probe

The block diagram of the On-Chip Hardware Probe appears in Figure 2. The OCHP consists in the following blocks. The Data Filter is connected to all the signals that can be monitored in the chip. It is responsible to send the relevant signal states to the I/O Port. The appropriate signals are determined by the current configuration selected in the Configuration Registers. These registers hold different configurations that are programmed and can be switched at runtime via the I/O Port.

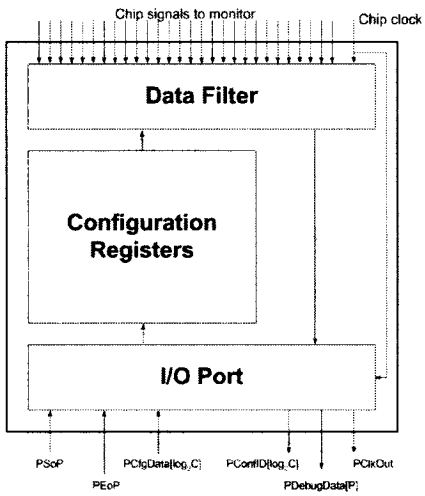

Figure 2. Block diagram of the OCHP

The I/O port is connected to the ACF by a number of pins. That number varies according to three design dependent factors: the number of internal signals monitored in the chip (N), the number of output debug data pins (P) and the number of different configurations that can be stored in the OCHP (C).

The input pins are used as follows: the PSoP and PEoP signals indicate the nature of the incoming data, which is sent via the PCfgData pins. Only $\log_2 C$ PCfgData pins are needed to perform the configuration programming and switching tasks. There are also three groups of output pins. The PClkOut pin sends the DUT clock to the ACF. The PConfID signals ($\log_2 C$ pins are needed) indicate the current configuration selected in the OCHP. The PDebugData pins transmit the selected internal signals states in real-time to the ACF.

3.3. Assertion Checker FPGA

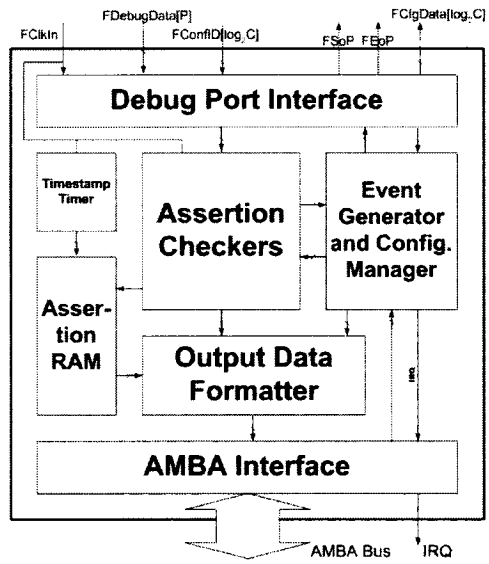

Figure 3. Block diagram of the ACF

Figure 3 shows the block diagram of the ACF. This unit consists in the following blocks. The Debug Port Interface is connected to the OCHP debug port. It receives the collected data in real-time and transmits the configuration settings to the OCHP when needed. The system clock of the chip under verification is sent to the ACF via the Debug Port to synchronize the Timestamp Timer. The debug data is used by the Event Generator, which has the capability to dynamically change the configuration in the OCHP. The Event Generator also starts and stops the Assertion Checkers and

controls an interrupt request pin (IRQ) going to the on-board processor. The Assertion Checkers can store debug data along with timestamps in the Assertion RAM, for later consultation. Upon request from the processor via the AMBA Interface, the Output Data Formatter can get the state of the Assertion Checkers or the Event Generators, as well as the data stored in the Assertion RAM, and send this information to the processor. The processor also has the capability to configure the ACF that in turn, configures the OCHP.

3.4. ABRD Compiler

The configuration settings are dependent on the DUT and the assertion checkers to be implemented. In order to make the configuration of the ABRD as simple as possible, a compiler is provided. A file describing the mapping of the signals connected to the Data Filter in the OCHP as well as the Debug Port width and clock frequency must be created for each design. The compiler also needs another file, containing the assertions expressed in an assertion language, like PSL or OVL. With these files, the compiler translates the assertions in RTL code, for implementation in the ACF, and in configuration settings for the OCHP. The compiler optimizes the acquisition rate for each signal and determines the conditions for when dynamic changes in the configuration are made by the Event Generator. Warnings are printed when assertions cannot be correctly verified with the available bandwidth. The resulting FPGA routing file and the configuration file are uploaded in the ACF by the processor.

4. DISCUSSION

The ABRD has several advantages over existing debug methods. The use of assertion languages to describe the expected behavior allows verification engineers to use the same properties that were added in the RTL code at the design stage for on-line debug purposes. A designer following the rules of assertion-based design, explained in [2], and planning the routing of the debug signals to the OCHP, from existing assertions, will get the full advantage of the time savings provided by the ABRD.

The flexibility offered by the FPGA and the compiler allows the use of different kinds of assertions, depending on the debug needs. For example, an erroneous behavior may be detected by general assertions, and afterwards, the source of the bug can be investigated with assertions covering a precise part of a chip.

Using a FPGA to implement assertion checkers instead of off-line processing of debug data in software helps to maximize the OCHP bandwidth. A transaction can be followed in real-time in different parts of the DUT by dynamically changing the probe configuration according to the needs of the active assertions checkers.

The OCHP is a minimally-intrusive way to get the required visibility through pins and system interfaces, without using too many connections. This advantage can be demonstrated by comparing the cost estimates of the OCHP with the cost of routing all monitored signals to external pins, for different values of the N, P and C parameters. The estimates were made considering the use of gates from the TSMC standard cell library, in 0.18µm technology. We also considered that many assertion patterns described in [2] can be checked using fewer than 16 simultaneous probed signals.

Table 1 presents the results for 64 and 128 internal signals, with different numbers of debug pins and stored configurations. Since total number of pins is limited in chip packages, using pins for other purposes than debug presents significant advantages. In an industrial design the authors were exposed to, 128 pins per chip were reserved for debug purposes. A smaller package, at a lower cost, could have been used by implementing the debug hardware presented in this paper.

Table 1. Cost estimates for different OCHP configurations

N	P	C	OCHP die area (mm^2)	NAND2 equivalent	OCHP Pins needed	Pins saved	Operating Frequency (MHz)
64	8	8	0.08	6904	17	47	215
64	16	4	0.11	9250	23	41	212
64	16	8	0.17	13687	25	39	207
128	16	8	0.32	26222	25	103	165

Table 1 also shows that the gate count (in NAND2 equivalent) and the die area used by the OCHP are small, compared to the size of modern designs. In today's multi-million gates ASICs and FPGAs, using up to 26000 gates for debug purposes is an acceptably small overhead. This

ANNEXE A : Article ISCAS 2004

methodology will become more and more advantageous with the evolution of lithography, since on-chip gate size decrease faster than pads size.

A few drawbacks can be seen in this technique in its current form. The performance of the integrated probe is limited by its loading effect on the system, as well as the limited bandwidth and latency of long signals that may need to be routed to the Data Filter. A complete solution will have to take these factors into account when routing signals to the OCHP and generating Assertion Checkers in the ACF. The problem is partly solved by inserting buffers near the monitored signals and distributing the Data Filter logic in different points of the design, in the vicinity of clusters of monitored signals.

Preliminary results with a OCHP monitoring 64 signals in a real world design implemented in a Xilinx XCV800 FPGA shows that the probe adds an average of 0.79 ns to the signal propagation delay, for an average degradation of performances of 14%. It is important to note that no optimization was made, no buffers were added, and the Data Filter was concentrated in one location. Performance issues and their solutions will be discussed in more details in a future paper.

5. CONCLUSION

An assertion-based debug environment for high-speed complex hardware systems was presented. The advantages of this method for bug identification and localization were discussed. This architecture is currently implemented on a real design and simulations are performed to validate the methodology. Hardware implementation on a prototyping board will follow.

6. ACKNOWLEDGEMENTS

We would like to thank the Natural Sciences and Engineering Research Council of Canada for their financial support.

7. REFERENCES

[1] R. York, J. Sharp, "Real Time Debug for System-on-Chip Devices", *White Paper*, ARM Ltd., Cambridge, UK, June 1999.

[2] Foster, H.D., A.C. Krolnik, and D.J. Lacey, Assertion-Based Design, Kluwer Academic Publishers, Boston, 2003.

[3] R. Leatherman, B. Ableidinger, and N. Stollon, "Method offers snapshot of SoC operation", EE Times, CMP Media LLC, Manhasset, NY, April 2003.

[4] M. El Shobaki, "On-Chip Monitoring of Single- and Multiprocessor Hardware Real-Time Operating Systems", 8th International Conference on Real-Time Computing Systems and Applications, 2002.

[5] IEEE Industry Standards and Technology Organization, The Nexus 5001 Forum™ Standard for a Global Embedded Processor Debug Interface, IEEE-ISTO, Piscataway, NJ, December 1999.

[6] H. O'Keeffe, "IEEE-ISTO 5001™-1999, The Nexus 5001 ForumTM Standard providing the Gateway to the Embedded Systems of the Future", White Paper, Ashling Microsystems Ltd., Limerick, Ireland, January 2000.

[7] C. MacNamee and D. Heffernan, "Emerging on-chip debugging techniques for real-time embedded systems", Computing & Control Engineering Journal, IEE, Stevenage, UK, pp. 295-303, December 2003.

[8] C. Brokish and D. Kertis, "Adapting traditional embedded debug strategies to SoC designs", EE Times, CMP Media LLC, Manhasset, NY, June 2002.

[9] R. Drechler, "Synthesizing checkers for on-line verification of System-on-Chip designs", Proceedings of the 2003 IEEE ISCAS, IEEE, Piscataway, NJ, vol. 4, pp. 748-751, May 2003.

[10] B. Tabbara, Y.-C. Hsu, G. Bakewell and S. Sandler, "Assertion-Based Hardware Debugging", *White Paper*, DVCon 2003, San Jose, CA, February 2003.

[11] C. Maxfield, "Walking the assertion maze", EE Design, CMP Media LLC, Manhasset, NY, August 2002.

[12] 0-In Design Automation inc., "Assertion-Based Verification for Complex Designs", *White Paper*, 0-In Design Automation inc., San Jose, CA, January 2003.

[13] Aldec inc., "Riviera IPT Hardware Acceleration Platform", Aldec inc., Henderson, NV, 2003.

[14] Bergeron, J., Writing Testbenches: Functional Verification of HDL Models, Kluwer Academic Publishers, Boston, 2003.

[15] H.D. Foster, "Property Specification: The key to an Assertion-Based Verification Platform", Electronic Design Processes 2003, Monterey, CA, 2003.

ANNEXE B : Description des registres du ACF

Cette annexe présente une description fonctionnelle des registres du ACF, incluant, lorsque pertinent, le format des données, la plage d'adresses sur le bus AMBA et le type d'accès au registre (lecture et/ou écriture). Comme la plupart des registres sont accessibles par le bus AMBA de 32 bits, les données sont présentées sous ce format, bien que les registres puissent nécessiter moins de bits. Les bits non utilisés par un registre sur le bus de données AMBA prennent la valeur 0 en lecture et ne sont pas considérés en écriture. Sauf mention contraire, les bits des registres prennent la valeur 0 au reset (signal FHRESETn ou Soft Reset).

B.1. REGISTRES GÉNÉRAUX DU ACF

Le Assertion Checker FPGA comporte trois registres généraux accessibles par le bus AMBA, qui servent à informer de l'état général du ACF et à exécuter des commandes globales.

a) ACF IDCode

Ce registre de 32 bits contient une valeur constante déterminée à la compilation, qui permet d'identifier le code programmé dans le FPGA.

- Type d'accès : Lecture seulement;
- Adresse AMBA : 0x00000000;
- Format des données : Constante de 32 bits déterminée à la compilation.

b) ACF Status

Ce registre comporte 6 bits indiquant l'état courant du ACF.

- Type d'accès : Lecture seulement;
- Adresse AMBA : 0x00000001;
- Format des données : Chaque bit indiqué à la Figure 1 représente un état du système :
  - IRQ set : Le signal d'IRQ est présentement actif.
  - Cfg dumped : Les données de configuration ont été envoyées au OCHP.
  - Dumping cfg : Le ACF est en train d'envoyer les données de configuration au OCHP.

ANNEXE B : Description des registres du ACF

- Cfg stored : Les données de configuration d'au moins une configuration du OCHP sont stockées dans la RAM de configuration. Il prend la valeur 1 si au moins un bit du registre Configuration stored est à 1 (section B.5).
- Storing debug data : Le ACF est en train de stocker des données de déverminage provenant du Debug Port Interface. En d'autres mots, le signal de déclenchement a été activé et le stockage des données est en cours.
- Trigger armed : Le signal de déclenchement de la Assertion RAM est armé.

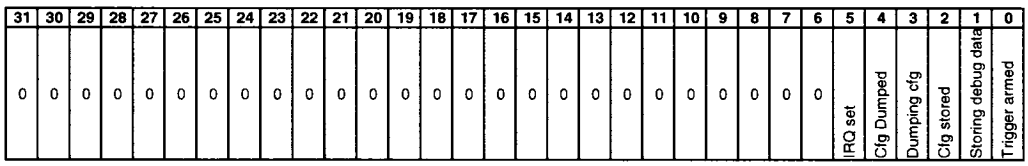

Figure 1: Format des données du registre ACF Status

c) ACF Control

Ce registre comporte 6 bits indiquant l'état courant du ACF.

- Type d'accès : Écriture seulement;
- Adresse AMBA : 0x00000010;
- Format des données : Chaque bit indiqué à la Figure 2 représente une commande générale pouvant être envoyée au ACF. L'écriture d'une valeur à un ou plusieurs bit(s) de ce registre prend effet immédiatement. Le registre ne garde pas la valeur inscrite.
  - Soft reset : Remise à zéro générale du ACF (Assertion Checkers, Event Generator, Configuration Manager, Assertion RAM, etc.). L'écriture d'un 1 à ce bit a le même effet qu'un niveau actif sur le signal externe FHRESETn.
  - Dump cfg to OCHP : Envoi des données de configuration au OCHP. L'écriture d'un 1 à ce bit provoque l'envoi des données de configuration (préalablement programmées dans la RAM de configuration) au OCHP.

ANNEXE B : Description des registres du ACF

- Reset OCHP cfg : Remise à zéro de la configuration du OCHP. L'écriture d'un 1 à ce bit remet à zéro le registre Configuration stored (section B.5).
- Reset Timestamp Timer : Remise à zéro du compteur du Timestamp Timer. L'écriture d'un 1 à ce bit remet immédiatement à zéro de compteur du Timestamp Timer.
- Clear IRQ : Remise à zéro du signal FIRQ. L'écriture d'un 1 à ce bit remet à zéro le signal FIRQ.
- Stop storage : Arrêt du stockage de données de déverminage. L'écriture d'un 1 à ce bit provoque l'arrêt immédiat du stockage des données de déverminage dans la Assertion RAM. Le signal de déclenchement doit être réarmé après cette commande.
- Force trigger : Activation du signal de déclenchement de la Assertion RAM. L'écriture d'un 1 à ce bit provoque le déclenchement du stockage dans la Assertion RAM. Le signal de déclenchement doit préalablement avoir été armé pour que Force trigger prenne effet. L'arrêt du stockage s'effectue selon les conditions programmées dans les registres de contrôle de la Assertion RAM (section B.6) ou par la commande Stop storage.
- Arm trigger : Armement du signal de déclenchement de la Assertion RAM. L'écriture d'un 1 à ce bit arme le signal de déclenchement de la Assertion RAM. Le signal de déclenchement s'active selon les conditions programmées dans les registres de contrôle de la Assertion RAM (section B.6) ou par la commande Force Trigger.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Soft reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Dump cfg to OCHP	Reset OCHP cfg	Reset Timestamp Time	Clear IRQ	Stop storage	Force trigger	Arm trigger

Figure 2: Format des données du registre ACF Control

ANNEXE B : Description des registres du ACF

B.2. REGISTRES DU TIMESTAMP TIMER

Le Timestamp Timer comporte deux registres accessibles par le bus AMBA, qui permettent de déterminer la valeur maximale de décompte du temps et de consulter la valeur présente du décompte.

a) Timestamp Timer Maximum Count

Ce registre de 8 à 32 bits (selon la largeur du compteur du Timestamp Timer) permet d'indiquer la valeur maximale atteignable par le compteur avant le rebouclage à zéro.

- Type d'accès : Écriture et lecture;
- Adresse AMBA : 0x00000020;
- Format des données : Entier de 8 à 32 bits. Les bits les plus significatifs prennent la valeur 0 en lecture et sont ignorés en écriture si l'entier comporte moins de 32 bits.

b) Timestamp Timer Current Time

Ce registre de 8 à 32 bits indique la valeur actuelle du compteur du Timestamp Timer.

- Type d'accès : Lecture seulement;
- Adresse AMBA : 0x00000021;
- Format des données : Entier de 8 à 32 bits. Les bits les plus significatifs prennent la valeur 0 en lecture si l'entier comporte moins de 32 bits.

B.3. REGISTRES DES ASSERTION CHECKERS

Chaque Assertion Checker comporte trois registres accessibles par le bus AMBA. Ces registres permettent de lire le bit d'état du Assertion Checker, d'indiquer une information optionnelle (Timestamp, compteur, etc..) et de remettre à zéro l'Assertion Checker.

a) AC State

Ce registre de 1 bit indique l'état du Assertion Checker.

- Type d'accès : Lecture seulement;
- Adresse AMBA : De 0x00001000 à 0x000010FF (possibilité de 256 AC);

ANNEXE B : Description des registres du ACF

- Format des données : 1 bit, indiqué sur le LSB du bus de données AMBA. Les bits les plus significatifs prennent la valeur 0 en lecture.

b) ACTimestamp/Failure Counter

Ce registre de 0-32 bits indique une information sur le Assertion Checker (Timestamp, compteur, etc..)

- Type d'accès : Lecture seulement;
- Adresse AMBA : De 0x00001800 à 0x000018FF (possibilité de 256 AC);
- Format des données : Donnée de 0-32 bits. Les bits les plus significatifs non utilisés prennent la valeur 0 en lecture.

c) AC Reset

Ce registre de $\log_2 nAC + 1$ bits sert à remettre à zéro un ou tous les Assertion Checkers.

- Type d'accès : Écriture seulement;
- Adresse AMBA : 0x00000011;
- Format des données : Le bit le plus significatif sert à remettre à zéro tous les Assertion Checkers. La valeur indiquée sur les bits les moins significatifs identifie un Assertion Checker spécifique que l'on désire remettre à zéro.

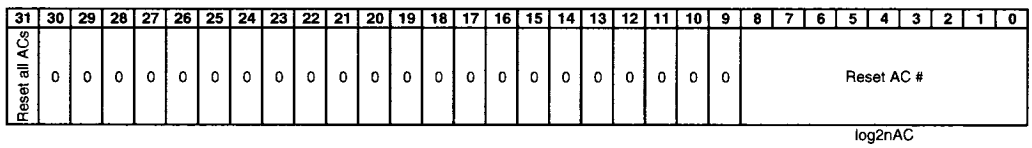

Figure 3: Format des données du registre AC Reset

B.4. REGISTRES DU EVENT GENERATOR

Le Event Generator comporte un registre global accessible par le bus AMBA, qui permet de déterminer quel moniteur d'événements est relié au signal FIRQ. De plus, chaque moniteur d'événements comporte trois registres accessibles par le bus AMBA. Ces registres permettent de lire le bit d'état du moniteur d'événements, d'indiquer une

ANNEXE B : Description des registres du ACF

information optionnelle (Timestamp, compteur, etc..) et de remettre à zéro le moniteur d'événements.

a) EG IRQ Pin Event Number

Ce registre de $\log_2$nEG + 1 bits sert à indiquer si le signal FIRQ peut être activé et, le cas échéant, quel moniteur d'événements commande le signal FIRQ.

- Type d'accès : Écriture et lecture;
- Adresse AMBA : 0x00000035;
- Format des données : Le bit le plus significatif sert à indiquer que le signal IRQ ne peut être activé par aucun moniteur d'événements. Les $\log_2$nEG bits les moins significatifs indiquent quel moniteur d'événements commande le signal FIRQ.

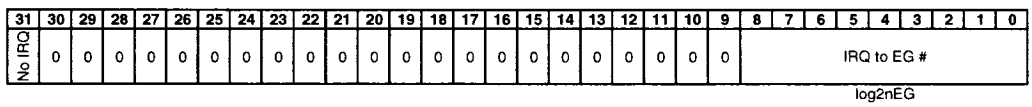

Figure 4: Format des données du registre EG IRQ Pin Event Number

b) EG State

Ce registre de 1 bit indique l'état du moniteur d'événements.

- Type d'accès : Lecture seulement;
- Adresse AMBA : De 0x00002000 à 0x000020FF (possibilité de 256 EG);
- Format des données : 1 bit, indiqué sur le LSB du bus de données AMBA. Les bits les plus significatifs prennent la valeur 0 en lecture.

c) EG Timestamp/Counter

Ce registre de 0-32 bits indique une information sur le moniteur d'événements (Timestamp, compteur, etc..)

- Type d'accès : Lecture seulement;
- Adresse AMBA : De 0x00002800 à 0x000028FF (possibilité de 256 EG);
- Format des données : Donnée de 0-32 bits. Les bits les plus significatifs non utilisés prennent la valeur 0 en lecture.

ANNEXE B : Description des registres du ACF

d) EG Reset

Ce registre de $\log_2$nEG + 1 bits sert à remettre à zéro un ou tous les moniteurs d'événements.

- Type d'accès : Écriture seulement;
- Adresse AMBA : 0x00000012;
- Format des données : Le bit le plus significatif sert à remettre à zéro tous les moniteurs d'événements. La valeur indiquée sur les bits les moins significatifs identifie un moniteur d'événements spécifique que l'on désire remettre à zéro.

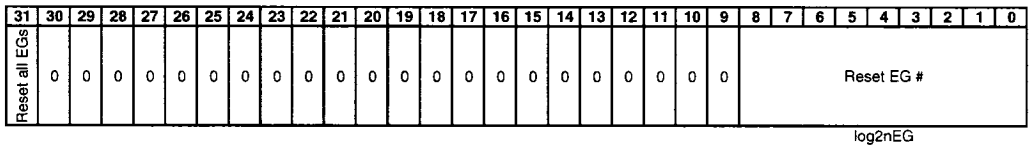

Figure 5: Format des données du registre EG Reset

B.5. REGISTRES DU CONFIGURATION MANAGER

Le Configuration Manager comporte deux registres principaux, qui permettent d'accéder à Configuration RAM et de déterminer si une configuration spécifique a été stockée. Le bus AMBA n'a accès qu'aux registres de la Configuration RAM.

a) Configuration RAM

Ces registres permettent d'accéder chaque configuration à stocker dans la Configuration RAM. L'adresse AMBA correspond à la configuration accédée.

- Type d'accès : Écriture seulement;
- Adresse AMBA : De 0x00000100 à 0x000001FF (256 configuration possibles);
- Format des données : Les bits les plus significatifs indiquent le numéro de la broche FDebugData configurée, alors que les bits les moins significatifs indiquent le numéro du signal interne relié à cette pin. Une écriture dans un des registres Configuration RAM provoque une écriture immédiate dans cette mémoire, à l'adresse correspondant au numéro de la configuration indiqué par l'adresse du registre et au numéro de la broche indiqué par les $\log_2$P bits les plus significatifs du bus de données AMBA.

ANNEXE B : Description des registres du ACF

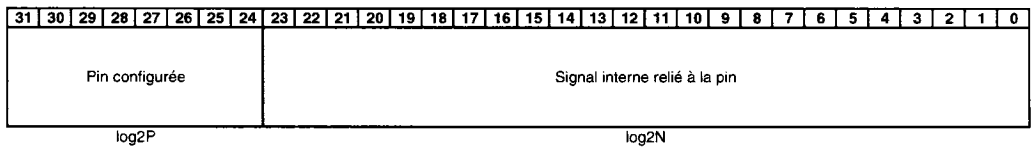

Figure 6: Format des données de la Configuration RAM

b) Configuration stored

Ce registre de C bits sert à indiquer au Configuration Manager que les données de configuration d'une configuration sont stockées dans la Configuration RAM.

- Type d'accès : Interne au ACF;
- Adresse AMBA : N/A;
- Format des données : Chaque bit correspond à une des configurations possibles. Une valeur 1 à un bit signifie qu'au moins une broche a été programmée pour une configuration donnée.

B.6. REGISTRES DE LA ASSERTION RAM

La Assertion RAM comporte 9 registres accessibles par le bus AMBA. Ces registres permettent de configurer le signal de déclenchement, la taille des données stockées et l'événement de fin de stockage, en plus de permettre de consulter les valeurs de timestamp et de pointeur du signal de déclenchement et de la fin de stockage et les données stockées dans la Assertion RAM.

a) RAM Trigger type

Ce registre comporte 2 bits permettant d'indiquer le type de signal de déclenchement et la condition de fin de stockage de la Assertion RAM.

- Type d'accès : Écriture et lecture;
- Adresse AMBA : 0x00000030;
- Format des données : Chaque bit indiqué à la Figure 7 représente une configuration de la Assertion RAM, décrite ci-dessous :

ANNEXE B : Description des registres du ACF

- Manual Trigger : Une valeur 1 à ce bit signifie que seul le signal de déclenchement manuel offert par le bit Force Trigger du registre de contrôle du ACF (section B.1) peut déclencher le stockage dans la Assertion RAM.
- Store until event : Une valeur 0 à ce bit signifie qu'une fois le signal de déclenchement activé, le stockage dans la Assertion RAM continue jusqu'au remplissage de la mémoire à la taille maximale indiquée par le registre RAM Storage Size (section d)). Une valeur 1 signifie quant à elle que le stockage continue jusqu'au front montant du signal de sortie du générateur d'événements sélectionné par le registre RAM Storage End Event Number (section e)) ou l'envoi d'une commande Stop Storage (section B.1).
- Single shot : Une valeur 1 à ce bit signifie que le stockage dans la Assertion RAM n'a lieu qu'une fois, c'est-à-dire que le signal de déclenchement n'est pas réarmé à la fin du stockage pour un éventuel écrasement des données par une autre activation du signal de déclenchement.

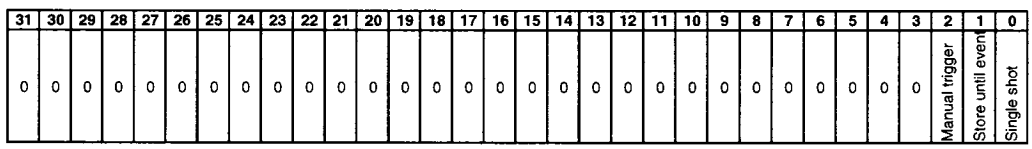

Figure 7: Format des données du registre RAM Trigger Type

b) RAM Trigger Event Number

Ce registre de $\log_2$nEG bits identifie le générateur d'événements qui active le signal de déclenchement de la Assertion RAM.

- Type d'accès : Écriture et lecture;
- Adresse AMBA : 0x00000031;
- Format des données : Les $\log_2$nEG bits les moins significatifs du bus AMBA identifient le numéro du générateur d'événements qui active le signal de déclenchement de la Assertion RAM.

c) RAM Trigger Delay

ANNEXE B : Description des registres du ACF

Ce registre de 8-32 bits permet de déterminer le délai entre le début du stockage et l'apparition du signal de déclenchement, pour un déclenchement de type Single Shot (section a)). Les données stockées dans la Assertion RAM avant le déclenchement, pendant le délai indiqué par ce registre, sont conservées.

- Type d'accès : Écriture et lecture;
- Adresse AMBA : 0x00000032;
- Format des données : Un entier de 8-32 bits (les moins significatifs du bus AMBA) indique le délai en nombre de coups d'horloge entre le début du stockage et l'apparition du signal de déclenchement.

d) RAM Storage Size

Ce registre de 8-32 bits permet de déterminer le nombre de coups d'horloge pendant lequel le stockage est effectué après le déclenchement, en incluant le délai indiqué par le registre RAM Trigger Delay (section c)). Cette valeur détermine la valeur à laquelle le compteur d'adresse de la Assertion RAM reboucle.

- Type d'accès : Écriture et lecture;
- Adresse AMBA : 0x00000033;
- Format des données : Un entier de 8-32 bits (les moins significatifs du bus AMBA) indique le nombre de coups d'horloge pendant lequel le stockage est effectué après le déclenchement, en incluant le délai indiqué par le registre RAM Trigger Delay.

e) RAM Storage End Event Number

Ce registre de $\log_2 nEG$ bits identifie le générateur d'événements qui provoque la fin du stockage dans la Assertion RAM, lorsque le bit Store Until Event du registre RAM Trigger Type est actif (section a)).

- Type d'accès : Écriture et lecture;
- Adresse AMBA : 0x00000034;
- Format des données : Les $\log_2 nEG$ bits les moins significatifs du bus AMBA identifient le numéro du générateur d'événements qui provoque la fin du stockage dans la Assertion RAM.

ANNEXE B : Description des registres du ACF

f) RAM Trigger Timestamp

Ce registre de 8 à 32 bits indique le temps du Timestamp Timer lors de la dernière activation du signal de déclenchement.

- Type d'accès : Lecture seulement;
- Adresse AMBA : 0x00000040;
- Format des données : Entier de 8 à 32 bits. Les bits les plus significatifs prennent la valeur 0 en lecture si l'entier comporte moins de 32 bits.

g) RAM End of Storage Pointer

Ce registre de 8-32 bits indique l'adresse de la Assertion RAM à laquelle la dernière donnée stockée a été écrite. Si le stockage est en cours, l'adresse indiquée est l'adresse de stockage actuelle.

- Type d'accès : Lecture seulement;
- Adresse AMBA : 0x00000041;
- Format des données : Adresse de 32 bits de la case mémoire concernée sur le bus AMBA (section i)).

h) RAM End of Storage Timestamp

Ce registre de 8 à 32 bits indique le temps du Timestamp Timer lors de la fin du stockage en mémoire. Si le stockage est en cours, ce registre indique le temps actuel du Timestamp Timer.

- Type d'accès : Lecture seulement;
- Adresse AMBA : 0x00000042;
- Format des données : Entier de 8 à 32 bits. Les bits les plus significatifs prennent la valeur 0 en lecture si l'entier comporte moins de 32 bits.

i) Assertion RAM

Chaque adresse correspond à une case mémoire de la Assertion RAM. La lecture d'une adresse provoque la lecture de la case mémoire correspondante de la Assertion RAM. Si un stockage est en cours, il est arrêté par une lecture à une des adresses de la

ANNEXE B : Description des registres du ACF

Assertion RAM. Une lecture provoque également le désarmement du signal de déclenchement, qui doit être réarmé par la commande Arm Trigger (section B.1).

- Type d'accès : Lecture seulement;
- Adresse AMBA : De 0x10000000 à 0x1FFFFFFF (2^{28} adresses possibles);
- Format des données : Les données envoyées sur le bus AMBA sont celles lues à l'adresse correspondante de la Assertion RAM.

ANNEXE C : Sources OCHP

```
------------------------------------------------------------
-- Description : OCHP top level
--
-- File   : ochp.vhd
-- Author : K. Peterson
-- Date   : Jan 27 2004
--
-- Projet : Assertion-Based Runtime Debugger
------------------------------------------------------------

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

entity ochp is
  PORT (
    -- Control signals
    clk                    : IN    std_logic;
    reset_n                : IN    std_logic;

    -- Internal signals
    PIntSignal             : IN    std_logic_vector((N-1) downto 0);

    -- I/O Port input signals
    PSoP                   : IN    std_logic;
    PEoP                   : IN    std_logic;
    PCfgData               : IN    std_logic_vector(Cwid downto 0);

    -- I/O Port output signals
    PClkOut                : OUT   std_logic;
    PConfID                : OUT   std_logic_vector(Cwid downto 0);
    PDebugData             : OUT   std_logic_vector((P-1) downto 0)
    );
end ochp;

architecture RTL of ochp is

    component dfslice
    PORT (
      -- Debug signals
      monsignal            : IN    std_logic_vector((N-1) downto 0);
      debugpin             : OUT   std_logic;

      -- Configuration signals
      confaddr             : IN    std_logic_vector(Nwid downto 0)
    );
    end component;

    component crcell
    GENERIC (
      curconf              : integer
    );

    PORT (
      -- Control signals
      clk                  : IN    std_logic;
      reset_n              : IN    std_logic;
      shift_en             : IN    std_logic;

      -- Data signals
      serialconfin         : IN    std_logic;
      serialconfout        : OUT   std_logic;
      dfaddrbit            : OUT   std_logic;

      -- Configuration signals
      confaddr             : IN    std_logic_vector(Cwid downto 0)
    );
    end component;
```

ANNEXE C : Sources OCHP

```
    component crctrl
    PORT (
      -- Control signals
      clk                    : IN    std_logic;
      reset_n                : IN    std_logic;

      -- ACF signals
      sop                    : IN    std_logic;
      eop                    : IN    std_logic;
      cfgdata                : IN    std_logic_vector(Cwid downto 0);

      -- Configuration signals
      serialcfgdata          : OUT   std_logic;
      curprog                : OUT   std_logic_vector((C-1) downto 0);
      curconfaddr            : OUT   std_logic_vector(Cwid downto 0)
    );
    end component;

    signal   intsig        : std_logic_vector((N-1) downto 0);
    signal   debpin        : std_logic_vector((P-1) downto 0);
    signal   addrbus       : std_logic_vector((P*(Nwid+1)-1) downto 0);
    signal   cfgsdata      : std_logic_vector(((C*(P*(Nwid+1)))-1) downto 0);
    signal   curprg        : std_logic_vector((C-1) downto 0);
    signal   cfgdatamux    : std_logic;
    signal   curconf       : std_logic_vector(Cwid downto 0);

begin

    CONFREGCTRL: crctrl
    port map(
      clk=>clk,
      reset_n=>reset_n,
      sop=>PSoP,
      eop=>PEoP,
      cfgdata=>PCfgData,
      serialcfgdata=>cfgdatamux,
      curprog=>curprg,
      curconfaddr=>curconf
    );

    -- Configuration Registers generation
    CONFREG: for i in (C-1) downto 0 generate
      CRSLICE: for j in 0 to (P*(Nwid+1)-1) generate
        CRCELL0: if j = 0 generate
          CRC0: crcell
          generic map(i)
          port map(
            clk=>clk,
            reset_n=>reset_n,
            shift_en=>curprg(i),
            serialconfin=>cfgdatamux,
            serialconfout=>cfgsdata((i*P*(Nwid+1))+j),
            dfaddrbit=>addrbus(j),
            confaddr=>curconf
          );
        end generate;
        CRCELLx: if j > 0 generate
          CRCx: crcell
          generic map(i)
          port map(
            clk=>clk,
            reset_n=>reset_n,
            shift_en=>curprg(i),
            serialconfin=>cfgsdata((i*P*(Nwid+1))+j-1),
            serialconfout=>cfgsdata((i*P*(Nwid+1))+j),
            dfaddrbit=>addrbus(j),
            confaddr=>curconf
          );
        end generate;
      end generate;
```

ANNEXE C : Sources OCHP

```
    end generate;

    -- Data Filter generation
    DFSL: for j in (P-1) downto 0 generate
      DFC: dfslice
      port map(
        monsignal=>intsig,
        debugpin=>debpin(j),
        confaddr=>addrbus(((j*(Nwid+1))+Nwid) downto (j*(Nwid+1)))
      );
    end generate;

    -- Asynchronous assignation of signals

    intsig     <= PIntSignal;
    PClkOut    <= clk;
    PConfID    <= curconf;
    PDebugData <= debpin;

end RTL;
```

ANNEXE C : Sources OCHP

```
------------------------------------------------------------
-- Description : On-Chip Hardware Probe parameters file
--
-- File   : ochp_param.vhd
-- Author : K. Peterson
-- Date   : Jan 21 2004
--
-- Proj   : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

package ochptypes is
  constant C:         integer := 8;     -- Number of configurations
  constant Cwid:      integer := 2;     -- log2C - 1
  constant N:         integer := 64;    -- Number of internal signals
  constant Nwid:      integer := 5;     -- log2N - 1
  constant P:         integer := 16;    -- Number of external pins
  constant Pwid:      integer := 3;     -- log2P - 1
end ochptypes;
```

ANNEXE C : Sources OCHP

```
------------------------------------------------------------
-- Description : OCHP Configuration Registers controller
--
-- File   : confregctrl.vhd
-- Author : K. Peterson
-- Date   : Jan 21 2004
--
-- Proj   : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use work.ochptypes.all;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

ENTITY crctrl IS

  PORT (
    -- Control signals
    clk                    : IN    std_logic;
    reset_n                : IN    std_logic;

    -- ACF signals
    sop                    : IN    std_logic;
    eop                    : IN    std_logic;
    cfgdata                : IN    std_logic_vector(Cwid downto 0);

    -- Configuration signals
    serialcfgdata          : OUT   std_logic;
    curprog                : OUT   std_logic_vector((C-1) downto 0);
    curconfaddr            : OUT   std_logic_vector(Cwid downto 0)
    );
end crctrl;


ARCHITECTURE rtl OF crctrl IS

  SIGNAL curprogreg        : std_logic_vector((C-1) downto 0);
  SIGNAL curconfaddrreg    : std_logic_vector(Cwid downto 0);

BEGIN

  conf_reg_ctrl : PROCESS (clk,reset_n)
  BEGIN
    IF reset_n = '0' THEN
      FOR i IN (C-1) downto 0 LOOP
        curprogreg(i)     <= '0';
      END LOOP;
      FOR i IN Cwid downto 0 LOOP
        curconfaddrreg(i) <= '0';
      END LOOP;
    ELSIF clk'event AND clk = '1' THEN
      IF sop = '1' AND eop = '1' THEN
        curconfaddrreg    <= cfgdata;
        FOR i IN (C-1) downto 0 LOOP
          curprogreg(i)   <= '0';
        END LOOP;
      ELSIF sop = '1' THEN
--        curconfaddrreg    <= cfgdata;
        FOR i IN (C-1) downto 0 LOOP
          IF cfgdata = CONV_STD_LOGIC_VECTOR(i,cfgdata'length) THEN
            curprogreg(i) <= '1';
          ELSE
            curprogreg(i) <= '0';
          END IF;
        END LOOP;
      ELSIF eop = '1' THEN
        FOR i IN (C-1) downto 0 LOOP
          curprogreg(i)   <= '0';
```

ANNEXE C : Sources OCHP

```
        END LOOP;
      END IF;
    END IF;
  END PROCESS;

  -- Asynchronous signal assignations

  serialcfgdata     <= cfgdata(0);
  curprog           <= curprogreg;
  curconfaddr       <= curconfaddrreg;

END rtl;
```

ANNEXE C : Sources OCHP

```
------------------------------------------------------------
-- Description : OCHP Configuration Register basic cell
--
-- File   : confregcell.vhd
-- Author : K. Peterson
-- Date   : Jan 27 2004
--
-- Proj   : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use work.ochptypes.all;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

ENTITY crcell IS

  GENERIC (
    curconf                : integer
  );

  PORT (
    -- Control signals
    clk                    : IN    std_logic;
    reset_n                : IN    std_logic;
    shift_en               : IN    std_logic;

    -- Data signals
    serialconfin           : IN    std_logic;
    serialconfout          : OUT   std_logic;
    dfaddrbit              : OUT   std_logic;

    -- Configuration signals
    confaddr               : IN    std_logic_vector(Cwid downto 0)
    );
end crcell;

ARCHITECTURE rtl OF crcell IS

  SIGNAL cfgreg            : std_logic;

BEGIN

  conf_reg_cell : PROCESS (clk,reset_n)
  BEGIN
    IF reset_n = '0' THEN
      cfgreg      <= '0';
    ELSIF clk'event AND clk = '1' THEN
      IF shift_en = '1' THEN
        cfgreg      <= serialconfin;
      END IF;
    END IF;
  END PROCESS;

  -- Asynchronous Data Filter address bit assignation

  dfaddrbit   <= cfgreg  WHEN confaddr = CONV_STD_LOGIC_VECTOR(curconf,confaddr'length)
ELSE
                 'Z';

  serialconfout <= cfgreg;

END rtl;
```

ANNEXE C : Sources OCHP

```
------------------------------------------------------------
-- Description : OCHP Data Filter basic cell
--
-- File   : datafiltercell.vhd
-- Author : K. Peterson
-- Date   : Jan 27 2004
--
-- Proj   : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use work.ochptypes.all;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

ENTITY dfslice IS

  PORT (
    -- Debug signals
    monsignal              : IN    std_logic_vector((N-1) downto 0);
    debugpin               : OUT   std_logic;

    confaddr               : IN    std_logic_vector(Nwid downto 0)
    );
end dfslice;

ARCHITECTURE rtl OF dfslice IS

    component dfcell
    GENERIC (
      celladdr             : integer
    );

    PORT (
      -- Debug signals
      monsignal            : IN    std_logic;
      debugpin             : OUT   std_logic;

      -- Configuration signals
      confaddr             : IN    std_logic_vector(Nwid downto 0)
    );
    end component;

BEGIN

  DFSLICE: for i in (N-1) downto 0 generate
    DFC: dfcell
      generic map(i)
      port map(
        monsignal=>monsignal(i),
        debugpin=>debugpin,
        confaddr=>confaddr
      );
  end generate;

END rtl;
```

ANNEXE C : Sources OCHP

```
------------------------------------------------------------
-- Description : OCHP Data Filter basic cell
--
-- File   : datafiltercell.vhd
-- Author : K. Peterson
-- Date   : Jan 21 2004
--
-- Proj   : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use work.ochptypes.all;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

ENTITY dfcell IS

  GENERIC (
    celladdr               : integer
  );

  PORT (
    -- Debug signals
    monsignal              : IN    std_logic;
    debugpin               : OUT   std_logic;

    -- Configuration signals
    confaddr               : IN    std_logic_vector(Nwid downto 0)
    );
end dfcell;


ARCHITECTURE rtl OF dfcell IS

BEGIN

  debugpin    <= monsignal  WHEN confaddr =
CONV_STD_LOGIC_VECTOR(celladdr,confaddr'length) ELSE
                'Z';

END rtl;
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

capacitor_sum.scr

```
awk 'BEGIN{
}
{
if(NF > 2) {
  if(match(toupper($3),"INSTANCE")) {
    instname = $4;
  }

  if(match(toupper($(NF-2)),"CAPACITOR")) {
    net = substr($0,length($1)+3,length($0));
    net = substr(net,1,match(net,"capacitor")-4);
    tot[net] = tot[net] + substr($(NF-1),3,length($(NF-1)));
    numcap[net]++;
    capname[net] = capname[net]" "instname;
  }
}
}
END{
  for(elem in tot) {
    printf("Net: %s  Total Cap: %sF  Num. Cap: %d   Caps:
%s\n",elem,tot[elem],numcap[elem],capname[elem]);
  }
}' $1 | sort
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

netnames.scr

```
awk 'BEGIN{
flagnets = 0;
total = 0;
}
{
  if(match($0,"^NETS ")) {
    flagnets = 1;
  }
  if(match($0,"-") && (flagnets == 1)) {
    netname = $2;
  }
  if(match($0,"debugpin_tri A ")) {
    tot[netname]++;
  }
}
END{
  for(elem in tot) {
    printf("Net: %s  Total pins: %d\n",elem,tot[elem]);
    total = total + tot[elem];
  }
  printf("Total pins: %d\n",total);
}' $1 | sort
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

netpins.scr

```
awk 'BEGIN{
flagnets = 0;
total = 0;
indx = 0;
while(getline<"netsraw") {
  netlist[indx++] = $0"$";
}
}
{
  if(match($0,"^NETS ")) {
    flagnets = 1;
  }
  if(match($0,"-") && (flagnets == 1)) {
    flagnetlist = 0;
    for(i=0;i<indx;i++) {
      if(match($2,netlist[i])) { flagnetlist = 1;}
    }
  }
  if(flagnetlist) {
    print $0;
  }
}
END{
}' $1
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

parasitic_ochp.scr

```
awk 'BEGIN{
  flagochp = 0;
  numinst = 0;
  numochp = 0;
  totprob = 0;
  incrinst = 0;
  printf("*********************************\n");
  printf("* Parasitic capacitor extractor *\n");
  printf("*********************************\n");
  system("date");
  printf("\n\n\n");
}
{
if(NF > 2) {
  if(match(toupper($3),"INSTANCE")) {
    instname = $4;
    flagochp = 0;
    numinst++;

    if(match(instname,"debugpin_tri$")) {
      flagochp = 1;
      numochp++;
    }
  }

  if(match(toupper($(NF-2)),"CAPACITOR")) {
    net = substr($0,length($1)+3,length($0));
    net = substr(net,1,match(net,"capacitor")-4);
    totcap[net] = totcap[net] + substr($(NF-1),3,length($(NF-1)));
    numcap[net]++;
    capname[net] = capname[net]" "instname;
  }

  if(match($1,"^x")) {
    if(flagochp) {
      netname = substr($2,2,length($2));
      ochplist[netname] = instname" "ochplist[netname];
      ochpcnt[netname]++;
    }
    net = substr($0,1,(length($0)-length($(NF))-3));
    net = substr(net,length($1)+3,length($0));
    instnets[instname] = net;
    insttype[instname] = $(NF);
  }
}
}
END{

  prct = (numinst-numochp) / 100;

  for(ochp in ochplist) {
    incrinst = 0;
    nextincr = prct * 10;
    printf("\n\n##########################################################\n");
    system("date");
    printf("OCHP net: %s   Num. inst.: %d\n",ochp,ochpcnt[ochp]);
    split(ochplist[ochp],ochpinst," ");
    for(i=1;i<=ochpcnt[ochp];i++) {
      printf("  Probe %d: %s\n",i,ochpinst[i]);
    }
    totprob = totprob + ochpcnt[ochp];
    printf("  Total probes: %d/%d (%f%%)\n\n",totprob,numochp,(totprob/numochp)*100);
    printf("  Parasitic caps related to this net:\n");
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
    for(parcap in totcap) {
      if(match(parcap,"^"ochp" ") || match(parcap," "ochp"$")) {
        printf("\n    Between nets: %s    Parasitic cap.: %sF\n",parcap,totcap[parcap]);
        if(match(parcap,"^"ochp" ")) {
          othernet = substr(parcap,length(ochp)+2,length(parcap));
        }
        if(match(parcap," "ochp"$")) {
          othernet = substr(parcap,1,length(parcap)-length(ochp)-1);
        }
        for(instname in instnets) {
          if(match(instnets[instname],"^"othernet" ") || match(instnets[instname],"
"othernet" ") || match(instnets[instname]," "othernet"$")) {
            printf("      Non-probed net capacitor connected to inst.: %s
(%s)\n",instname,insttype[instname]);
            printf("        Connected to nets: %s\n",instnets[instname]);
          }
        }
      }
      incrinst = incrinst + numcap[parcap];
      if(incrinst > nextincr) {
        nextincr = nextincr + (prct*10);
        printf("Analyzed %d/%d instances
(%d%%)\n",incrinst,numinst,1+(incrinst/numinst)*100);
      }
    }
    printf("\n");
    for(instname in instnets) {
      if(match(instnets[instname],"^"ochp" ") || match(instnets[instname]," "ochp" ") ||
match(instnets[instname]," "ochp"$")) {
        printf("    Instance present on probed net: %s
(%s)\n",instname,insttype[instname]);
        printf("    Connected to nets: %s\n\n",instnets[instname]);
      }
      incrinst++;
      if(incrinst > nextincr) {
        nextincr = nextincr + (prct*10);
        printf("Analyzed %d/%d instances
(%d%%)\n",incrinst,numinst,1+(incrinst/numinst)*100);
      }
    }
    printf("Analyzed %d/%d instances (%d%%)\n",incrinst,numinst,1+(incrinst/numinst)*100);
  }
}' $1 #| sort
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

parasitic_ochp_noprobe.scr

```
awk 'BEGIN{
  numinst = 0;
  numochp = 0;
  totprob = 0;
  incrinst = 0;
  printf("*********************************************\n");
  printf("* Parasitic capacitor extractor (no probes) *\n");
  printf("*********************************************\n");
  system("date");
  printf("\n\n\n");
  while(getline<"newnetnames") {
    ochplist[$2] = $1;
    posnet[$2] = $3;
    numochp = numochp + $4;
    ochpcnt[$2] = $4;
  }
}
{
if(NF > 2) {
  if(match(toupper($3),"INSTANCE")) {
    instname = $4;
    flagochp = 0;
    numinst++;
  }

  if(match(toupper($(NF-2)),"CAPACITOR")) {
    net = substr($0,length($1)+3,length($0));
    net = substr(net,1,match(net,"capacitor")-4);
    totcap[net] = totcap[net] + substr($(NF-1),3,length($(NF-1)));
    numcap[net]++;
    capname[net] = capname[net]" "instname;
  }

  if(match($1,"^x")) {
    net = substr($0,1,(length($0)-length($(NF))-3));
    net = substr(net,length($1)+3,length($0));
    instnets[instname] = net;
    insttype[instname] = $(NF);
    for(indx in ochplist) {
      if(indx == instname) {
        split(net,tmpnet," ");
        newnetname[indx] = tmpnet[posnet[indx]];
      }
    }
  }
}
}
END{

  prct = (numinst-numochp) / 100;

  for(ochp in ochplist) {
    incrinst = 0;
    nextincr = prct * 10;
    printf("\n\n############################################################\n");
    system("date");
    printf("OCHP net: %s   Num. inst.: %d\n",newnetname[ochp],ochpcnt[ochp]);
    split(ochplist[ochp],ochpinst,"-");
    for(i=1;i<=ochpcnt[ochp];i++) {
      printf("  Probe %d: %s\n",i,ochpinst[i]);
    }
    totprob = totprob + ochpcnt[ochp];
    printf("  Total probes: %d/%d (%f%%)\n\n",totprob,numochp,(totprob/numochp)*100);
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
    printf("  Parasitic caps related to this net:\n");
    for(parcap in totcap) {
      if(match(parcap,"^"newnetname[ochp]" ") || match(parcap," "newnetname[ochp]"$")) {
        printf("\n    Between nets: %s    Parasitic cap.: %sF\n",parcap,totcap[parcap]);
        if(match(parcap,"^"newnetname[ochp]" ")) {
          othernet = substr(parcap,length(newnetname[ochp])+2,length(parcap));
        }
        if(match(parcap," "newnetname[ochp]"$")) {
          othernet = substr(parcap,1,length(parcap)-length(newnetname[ochp])-1);
        }
        for(instname in instnets) {
          if(match(instnets[instname],"^"othernet" ") || match(instnets[instname],"
"othernet" ") || match(instnets[instname]," "othernet"$")) {
            printf("      Non-probed net capacitor connected to inst.: %s
(%s)\n",instname,insttype[instname]);
            printf("        Connected to nets: %s\n",instnets[instname]);
          }
        }
      }
      incrinst = incrinst + numcap[parcap];
      if(incrinst > nextincr) {
        nextincr = nextincr + (prct*10);
        printf("Analyzed %d/%d instances
(%d%%)\n",incrinst,numinst,1+(incrinst/numinst)*100);
      }
    }
    printf("\n");
    for(instname in instnets) {
      if(match(instnets[instname],"^"newnetname[ochp]" ") || match(instnets[instname],"
"newnetname[ochp]" ") || match(instnets[instname]," "newnetname[ochp]"$")) {
        printf("    Instance present on probed net: %s
(%s)\n",instname,insttype[instname]);
        printf("    Connected to nets: %s\n\n",instnets[instname]);
      }
      incrinst++;
      if(incrinst > nextincr) {
        nextincr = nextincr + (prct*10);
        printf("Analyzed %d/%d instances
(%d%%)\n",incrinst,numinst,1+(incrinst/numinst)*100);
      }
    }
    printf("Analyzed %d/%d instances (%d%%)\n",incrinst,numinst,1+(incrinst/numinst)*100);
  }
}' $1 #| sort
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

parcapsum.scr

```
awk 'BEGIN{
flagfirstnet = 0;
}
{
  if(match($0,"^OCHP net:")) {
    if(flagfirstnet) {
      printf("Total parasitic cap. on net %s: %sF\n",netname,totalcap);
    }
    netname = $3;
    totalcap = 0;
    flagfirstnet = 1;
  }

  if(match($0,"Parasitic cap.:")) {
    totalcap = totalcap + substr($(NF),1,length($(NF))-1);
  }

}
END{
  printf("Total parasitic cap. on net %s: %sF\n",netname,totalcap);
}' $1 | grep -v " 0F$" | sort
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

findnewnetnames.scr

```
awk 'BEGIN{
}
{
  if(match($0,"^OCHP net:")) {
    netname = $3;
    flaginst = 0;
  }

  if(match($1,"^Probe")) {
    probename[netname] = $3"-"probename[netname];
    probeinc[netname]++;
  }

  if(match($0,"Instance present on probed net") && !match($6,"debugpin_tri")) {
    instname[netname] = $6;
    flaginst = 1;
  }

  if((flaginst == 1) && match($0,"Connected to nets")) {
    for(i=4;i<(NF+9);i++) {
      if(match($(i),"^"netname"$")) {
        posnet[netname] = i-3;
      }
    }
    flaginst = 2;
  }

}
END{
  for(nets in probename) {
    if(instname[nets] != "") {
      printf("%s %s %s %d\n",probename[nets],instname[nets],posnet[nets],probeinc[nets]);
    }
  }
}' $1
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

def2para.scr

```
awk -v parasit=$2 'BEGIN{
flagnets = 0;
total = 0;
}
{
  if(match($0,"^NETS ")) {
    flagnets = 1;
  }
  if(match($0,"-") && (flagnets == 1)) {
    netname = $2;
  }
  if(match($0,"debugpin_tri A ")) {
    tot[netname]++;
    firstdfc = match($0,"ochprb..DFC_")+8;
    secdfc = match($0,"DFC_[0-9]*..debugpin");
    dfcnum = substr($0,firstdfc,secdfc-2-firstdfc);
    dfcnum = dfcnum" "substr($0,secdfc,match($0,"debugpin")-2-secdfc);
    netdfc[dfcnum] = netname;
  }
}
END{
  while(getline<parasit) {
    if(match($0,"^OCHP net")) {
      netname = $3;
    }
    if(match($1,"Probe") && (pincnt[netname] == 0)) {
      firstdfc = match($0,"ochprb...DFC_")+9;
      secdfc = match($0,"DFC_[0-9]*...debugpin");
      dfcnum = substr($0,firstdfc,secdfc-3-firstdfc);
      dfcnum = dfcnum" "substr($0,secdfc,match($0,"debugpin")-3-secdfc);
      if(netdfc[dfcnum]) {
        printf("Netname: %s is %s in %s\n",netdfc[dfcnum],netname,parasit);
      }
      pincnt[netname]++;
    }
  }
}' $1 | sort
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

tracelength.scr

```
awk 'BEGIN{
flagnets = 0;
total = 0;
}
{
  if(match($0,"^NETS ")) {
    flagnets = 1;
    flagdebpin = 0;
    flagtrace = 0;
  }
  if(match($0,"-") && (flagnets == 1)) {
    netname = $2;
  }
  if(flagnets == 1) {
    if(match($1,";")) {flagdebpin = 0;}

#    if(match($0,"debugpin_tri A ")) {
      flagdebpin = 1;
#    }

    if(flagdebpin == 1) {
      if(match($2,"^ROUTED$")) {
        xorig = 5;
        flagtrace = 1;
      }

      if(match($1,"^NEW$")) {
        xorig = 4;
        flagtrace = 1;
      }

      if(flagtrace == 1) {
        if(NF >= (xorig+5)) {
          if(match($(xorig+4),"^\*$")) {
            subtrace = $(xorig+5) - $(xorig+1);
          }
          if(match($(xorig+5),"^\*$")) {
            subtrace = $(xorig+4) - $(xorig);
          }
          tracelength[netname] = tracelength[netname] + subtrace;
        }
      }

    }

  }

}
END{
  longuest = 0;
  for(nets in tracelength) {
    printf("Net: %s  Total trace length: %d\n",nets,tracelength[nets]);
    if(tracelength[nets] > longuest) {
      longuest = tracelength[nets];
      longname = nets;
    }
  }
  printf("Longuest trace net name: %s (%d)\n",longname,longuest);
}' $1 | sort
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

findbufloc.scr

```
awk 'BEGIN{
flagnets = 0;
indx = 0;
while(getline<"probenetlocs") {
  if(match($2,"^A$")) {
    source[++indx]=$3;
    dest[indx]=$1;
  }
  else {
    source[++indx]=$1;
    dest[indx]=$3;
  }
}
}
{
  if(match($0,"^NETS ")) {
    flagnets = 1;
  }
  if(!flagnets) {
    if(match($5,"PLACED")) {
      locx[$2] = $7;
      locy[$2] = $8;
    }
  }
}
END{
for(i=1;i<=indx;i++) {
  nms = source[i];
  nmd = dest[i];
  print "Dest: "nmd" "locx[nmd]" "locy[nmd]"  Source: "nms" "locx[nms]" "locy[nms];
}
}' $1 | sort
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

bufnetnames.scr

```
awk 'BEGIN{
flagnets = 0;
total = 0;
indx = 0;
while(getline<"bufsraw") {
  buflist[indx++] = $0;
}
}
{
  if(match($0,"^NETS ")) {
    flagnets = 1;
  }
  if(match($0,"-") && (flagnets == 1)) {
    netname = $2;
  }
  if(match($0," A )")) {
    for(i=0;i<indx;i++) {
      if(match($0,buflist[i]" A )")) {
        print buflist[i]" "netname;
      }
    }
  }
}
END{
}' $1 | sort
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

netnamesfinder.scr

```
awk -v buflist=$1 -v problist=$2 'BEGIN{
flagnets = 0;
total = 0;
indx = 0;
while(getline<buflist) {
  bufnam[$6] = $2;
  nambuf[$2] = $6;
}
while(getline<problist) {
  for(buf in bufnam) {
    if(match($3,"^Z")) {
      probnam = $6;
      probbuf = $2;
    }
    else {
      probnam = $2;
      probbuf = $6;
    }
    if(match(bufnam[buf],probbuf)) {
      prob[buf] = probnam;
    }
  }
}
}
{
if(NF > 2) {
  if(match(toupper($3),"INSTANCE")) {
    instname = $4;
    flagochp = 0;

    for(buf in bufnam) {
      if(instname == bufnam[buf]) {
        flagochp = 1;
        tmpbuf = buf;
      }
    }
  }

  if(match($1,"^x")) {
    if(flagochp) {
      netname = substr($2,2,length($2));
      ochpnet[netname] = tmpbuf;
    }
    net = substr($0,1,(length($0)-length($(NF))-3));
    net = substr(net,length($1)+3,length($0));
    instnets[instname] = net;
    insttype[instname] = $(NF);
  }
}
}
END{
  for(inst in instnets) {
    netcnt = split(instnets[inst],tmpnet," ");
    for(i=1;i<=netcnt;i++) {
      if((ochpnet[tmpnet[i]]) && (!nambuf[inst]) && (!flag[tmpnet[i]])) {
        tmpbuf = ochpnet[tmpnet[i]];
        flag[tmpnet[i]]++;
        printf("%s %s %d 1\n",prob[tmpbuf],inst,i);
      }
    }
  }
}' $3
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

matchparanet.scr

```
Netname: n2262 is 28874 in parasitic_bfrnear
Netname: n2265 is 28878 in parasitic_bfrnear
Netname: n2267 is 26476 in parasitic_bfrnear
Netname: n2269 is 28524 in parasitic_bfrnear
Netname: n2271 is 26104 in parasitic_bfrnear
Netname: n2273 is 28520 in parasitic_bfrnear
Netname: n2275 is 26110 in parasitic_bfrnear
Netname: n2277 is 28516 in parasitic_bfrnear
Netname: n2279 is 3960 in parasitic_bfrnear
Netname: n2281 is 2924 in parasitic_bfrnear
Netname: n2285 is 30331 in parasitic_bfrnear
Netname: n2314 is 23435 in parasitic_bfrnear
Netname: n2321 is 20287 in parasitic_bfrnear
Netname: n630 is 22781 in parasitic_bfrnear
Netname: n634 is 20189 in parasitic_bfrnear
Netname: n637 is 9662 in parasitic_bfrnear
Netname: n640 is 22794 in parasitic_bfrnear
Netname: n643 is 20208 in parasitic_bfrnear
Netname: n646 is 22800 in parasitic_bfrnear
Netname: n650 is 20250 in parasitic_bfrnear
Netname: n677 is 22816 in parasitic_bfrnear
Netname: n699 is 20448 in parasitic_bfrnear
Netname: n702 is 20464 in parasitic_bfrnear
Netname: n705 is 20477 in parasitic_bfrnear
Netname: n708 is 20491 in parasitic_bfrnear
Netname: n711 is 20500 in parasitic_bfrnear
Netname: n714 is 20512 in parasitic_bfrnear
Netname: n720 is 20529 in parasitic_bfrnear
Netname: ndfc0 is 10482 in parasitic_bfrnear
Netname: ndfc2 is 20547 in parasitic_bfrnear
Netname: ndfc24 is 20315 in parasitic_bfrnear
Netname: ndfc25 is 20332 in parasitic_bfrnear
Netname: ndfc26 is 10463 in parasitic_bfrnear
Netname: ndfc28 is 10461 in parasitic_bfrnear
Netname: ndfc29 is 10456 in parasitic_bfrnear
Netname: ndfc3 is 20381 in parasitic_bfrnear
Netname: ndfc30 is 10454 in parasitic_bfrnear
Netname: ndfc48 is 30376 in parasitic_bfrnear
Netname: ndfc49 is 30339 in parasitic_bfrnear
Netname: ndfc50 is 30335 in parasitic_bfrnear
Netname: ndfc62 is 10445 in parasitic_bfrnear
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

matchparabfrnear.scr

```
grep 28874 parasitic_bfrnear_*
grep 28878 parasitic_bfrnear_*
grep 26476 parasitic_bfrnear_*
grep 28524 parasitic_bfrnear_*
grep 26104 parasitic_bfrnear_*
grep 28520 parasitic_bfrnear_*
grep 26110 parasitic_bfrnear_*
grep 28516 parasitic_bfrnear_*
grep 3960 parasitic_bfrnear_*
grep 2924 parasitic_bfrnear_*
grep 30331 parasitic_bfrnear_*
grep 23435 parasitic_bfrnear_*
grep 20287 parasitic_bfrnear_*
grep 22781 parasitic_bfrnear_*
grep 20189 parasitic_bfrnear_*
grep 9662 parasitic_bfrnear_*
grep 22794 parasitic_bfrnear_*
grep 20208 parasitic_bfrnear_*
grep 22800 parasitic_bfrnear_*
grep 20250 parasitic_bfrnear_*
grep 22816 parasitic_bfrnear_*
grep 20448 parasitic_bfrnear_*
grep 20464 parasitic_bfrnear_*
grep 20477 parasitic_bfrnear_*
grep 20491 parasitic_bfrnear_*
grep 20500 parasitic_bfrnear_*
grep 20512 parasitic_bfrnear_*
grep 20529 parasitic_bfrnear_*
grep 10482 parasitic_bfrnear_*
grep 20547 parasitic_bfrnear_*
grep 20315 parasitic_bfrnear_*
grep 20332 parasitic_bfrnear_*
grep 10463 parasitic_bfrnear_*
grep 10461 parasitic_bfrnear_*
grep 10456 parasitic_bfrnear_*
grep 20381 parasitic_bfrnear_*
grep 10454 parasitic_bfrnear_*
grep 30376 parasitic_bfrnear_*
grep 30339 parasitic_bfrnear_*
grep 30335 parasitic_bfrnear_*
grep 10445 parasitic_bfrnear_*
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

matchparanoprobe.scr

```
grep 28781 parasitic_noprobe_*
grep 28785 parasitic_noprobe_*
grep 26359 parasitic_noprobe_*
grep 28431 parasitic_noprobe_*
grep 19641 parasitic_noprobe_*
grep 28427 parasitic_noprobe_*
grep 26175 parasitic_noprobe_*
grep 28423 parasitic_noprobe_*
grep 3947 parasitic_noprobe_*
grep 3112 parasitic_noprobe_*
grep 1766 parasitic_noprobe_*
grep 22718 parasitic_noprobe_*
grep 20003 parasitic_noprobe_*
grep 22207 parasitic_noprobe_*
grep 9666 parasitic_noprobe_*
grep 9661 parasitic_noprobe_*
grep 22217 parasitic_noprobe_*
grep 3998 parasitic_noprobe_*
grep 22223 parasitic_noprobe_*
grep 19932 parasitic_noprobe_*
grep 22234 parasitic_noprobe_*
grep 20162 parasitic_noprobe_*
grep 20173 parasitic_noprobe_*
grep 20195 parasitic_noprobe_*
grep 20208 parasitic_noprobe_*
grep 20229 parasitic_noprobe_*
grep 20239 parasitic_noprobe_*
grep 20240 parasitic_noprobe_*
grep 9550 parasitic_noprobe_*
grep 20273 parasitic_noprobe_*
grep 20038 parasitic_noprobe_*
grep 20047 parasitic_noprobe_*
grep 10322 parasitic_noprobe_*
grep 10311 parasitic_noprobe_*
grep 10318 parasitic_noprobe_*
grep 20114 parasitic_noprobe_*
grep 10308 parasitic_noprobe_*
grep 30283 parasitic_noprobe_*
grep 10290 parasitic_noprobe_*
grep 30242 parasitic_noprobe_*
grep 10330 parasitic_noprobe_*
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

parasitic_bfrnear_matched

```
parasitic_bfrnear_def2para:Netname: n2262 is 28874 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 28874: 1.11757e-14F
parasitic_bfrnear_def2para:Netname: n2265 is 28878 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 28878: 1.61276e-14F
parasitic_bfrnear_def2para:Netname: n2267 is 26476 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 26476: 1.40082e-14F
parasitic_bfrnear_def2para:Netname: n2269 is 28524 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 28524: 1.15748e-14F
parasitic_bfrnear_def2para:Netname: n2271 is 26104 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 26104: 1.85514e-14F
parasitic_bfrnear_def2para:Netname: n2273 is 28520 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 28520: 2.14549e-14F
parasitic_bfrnear_def2para:Netname: n2275 is 26110 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 26110: 6.53801e-15F
parasitic_bfrnear_def2para:Netname: n2277 is 28516 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 28516: 8.47956e-15F
parasitic_bfrnear_def2para:Netname: n2279 is 3960 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 3960: 1.63743e-14F
parasitic_bfrnear_def2para:Netname: n2281 is 2924 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 2924: 1.26518e-14F
parasitic_bfrnear_def2para:Netname: n2285 is 30331 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 30331: 1.1322e-14F
parasitic_bfrnear_def2para:Netname: n2314 is 23435 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 23435: 2.77151e-14F
parasitic_bfrnear_def2para:Netname: n2321 is 20287 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20287: 1.36658e-13F
parasitic_bfrnear_def2para:Netname: n630 is 22781 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 22781: 5.34214e-15F
parasitic_bfrnear_def2para:Netname: n634 is 20189 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20189: 7.85946e-15F
parasitic_bfrnear_def2para:Netname: n637 is 9662 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 9662: 4.47866e-15F
parasitic_bfrnear_def2para:Netname: n640 is 22794 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 22794: 2.19063e-15F
parasitic_bfrnear_def2para:Netname: n643 is 20208 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20208: 2.03027e-15F
parasitic_bfrnear_def2para:Netname: n646 is 22800 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 22800: 6.12741e-15F
parasitic_bfrnear_def2para:Netname: n650 is 20250 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20250: 1.52185e-14F
parasitic_bfrnear_def2para:Netname: n677 is 22816 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 22816: 4.83811e-14F
parasitic_bfrnear_def2para:Netname: n699 is 20448 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20448: 4.11943e-15F
parasitic_bfrnear_def2para:Netname: n702 is 20464 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20464: 8.56372e-15F
parasitic_bfrnear_def2para:Netname: n705 is 20477 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20477: 1.8966e-15F
parasitic_bfrnear_def2para:Netname: n708 is 20491 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20491: 8.30624e-15F
parasitic_bfrnear_def2para:Netname: n711 is 20500 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20500: 1.25428e-15F
parasitic_bfrnear_def2para:Netname: n714 is 20512 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20512: 1.24792e-15F
parasitic_bfrnear_def2para:Netname: n720 is 20529 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20529: 1.86033e-15F
parasitic_bfrnear_def2para:Netname: ndfc0 is 10482 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 10482: 1.43877e-14F
parasitic_bfrnear_def2para:Netname: ndfc2 is 20547 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20547: 2.99571e-15F
parasitic_bfrnear_def2para:Netname: ndfc24 is 20315 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20315: 7.73643e-15F
parasitic_bfrnear_def2para:Netname: ndfc25 is 20332 in parasitic_bfrnear
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20332: 5.83836e-15F
parasitic_bfrnear_def2para:Netname: ndfc26 is 10463 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 10463: 5.9962e-14F
parasitic_bfrnear_def2para:Netname: ndfc28 is 10461 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 10461: 4.76924e-15F
parasitic_bfrnear_def2para:Netname: ndfc29 is 10456 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 10456: 1.37217e-15F
parasitic_bfrnear_def2para:Netname: ndfc3 is 20381 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 20381: 9.16993e-14F
parasitic_bfrnear_def2para:Netname: ndfc30 is 10454 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 10454: 7.46825e-14F
parasitic_bfrnear_def2para:Netname: ndfc48 is 30376 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 30376: 1.67228e-14F
parasitic_bfrnear_def2para:Netname: ndfc49 is 30339 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 30339: 4.0812e-15F
parasitic_bfrnear_def2para:Netname: ndfc50 is 30335 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 30335: 2.52848e-14F
parasitic_bfrnear_def2para:Netname: ndfc62 is 10445 in parasitic_bfrnear
parasitic_bfrnear_parcapsum:Total parasitic cap. on net 10445: 2.24454e-14F
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

parasitic_noprobe_matched

```
parasitic_noprobe_def2para:Netname: n2262 is 28781 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 28781: 1.11637e-14F
parasitic_noprobe_def2para:Netname: n2265 is 28785 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 28785: 1.54973e-14F
parasitic_noprobe_def2para:Netname: n2267 is 26359 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 26359: 1.37107e-14F
parasitic_noprobe_def2para:Netname: n2269 is 28431 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 28431: 6.92934e-15F
parasitic_noprobe_def2para:Netname: n2271 is 19641 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 19641: 1.5111e-15F
parasitic_noprobe_def2para:Netname: n2273 is 28427 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 28427: 2.09218e-14F
parasitic_noprobe_def2para:Netname: n2275 is 26175 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 26175: 6.39427e-15F
parasitic_noprobe_def2para:Netname: n2277 is 28423 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 28423: 8.45452e-15F
parasitic_noprobe_def2para:Netname: n2279 is 3947 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 3947: 1.63493e-14F
parasitic_noprobe_def2para:Netname: n2281 is 3112 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 3112: 1.2621e-14F
parasitic_noprobe_def2para:Netname: n2285 is 1766 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 1766: 6.16173e-15F
parasitic_noprobe_def2para:Netname: n2314 is 22718 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 22718: 2.73606e-14F
parasitic_noprobe_def2para:Netname: n2321 is 20003 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20003: 1.32143e-13F
parasitic_noprobe_def2para:Netname: n630 is 22207 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 22207: 5.2052e-15F
parasitic_noprobe_def2para:Netname: n634 is 9666 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 9666: 7.49189e-15F
parasitic_noprobe_def2para:Netname: n637 is 9661 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 9661: 3.45501e-15F
parasitic_noprobe_def2para:Netname: n640 is 22217 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 22217: 2.16559e-15F
parasitic_noprobe_def2para:Netname: n643 is 3998 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 3998: 1.42233e-15F
parasitic_noprobe_def2para:Netname: n646 is 22223 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 22223: 4.89254e-15F
parasitic_noprobe_def2para:Netname: n650 is 19932 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 19932: 1.27323e-14F
parasitic_noprobe_def2para:Netname: n677 is 22234 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 22234: 4.54962e-14F
parasitic_noprobe_def2para:Netname: n699 is 20162 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20162: 2.74802e-15F
parasitic_noprobe_def2para:Netname: n702 is 20173 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20173: 8.54736e-15F
parasitic_noprobe_def2para:Netname: n705 is 20195 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20195: 1.75894e-15F
parasitic_noprobe_def2para:Netname: n708 is 20208 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20208: 8.28168e-15F
parasitic_noprobe_def2para:Netname: n711 is 20229 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20229: 1.22923e-15F
parasitic_noprobe_def2para:Netname: n714 is 20239 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20239: 1.09759e-15F
parasitic_noprobe_def2para:Netname: n720 is 20240 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20240: 2.92275e-16F
parasitic_noprobe_def2para:Netname: ndfc0 is 9550 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 9550: 1.43905e-14F
parasitic_noprobe_def2para:Netname: ndfc2 is 20273 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20273: 7.40978e-16F
parasitic_noprobe_def2para:Netname: ndfc24 is 20038 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20038: 6.47154e-15F
parasitic_noprobe_def2para:Netname: ndfc25 is 20047 in parasitic_noprobe
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20047: 5.79072e-15F
parasitic_noprobe_def2para:Netname: ndfc26 is 10322 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 10322: 5.92369e-14F
parasitic_noprobe_def2para:Netname: ndfc28 is 10311 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 10311: 4.58653e-15F
parasitic_noprobe_def2para:Netname: ndfc29 is 10318 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 10318: 1.34713e-15F
parasitic_noprobe_def2para:Netname: ndfc3 is 20114 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 20114: 9.15841e-14F
parasitic_noprobe_def2para:Netname: ndfc30 is 10308 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 10308: 7.46181e-14F
parasitic_noprobe_def2para:Netname: ndfc48 is 30283 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 30283: 1.67109e-14F
parasitic_noprobe_def2para:Netname: ndfc49 is 10290 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 10290: 2.41618e-15F
parasitic_noprobe_def2para:Netname: ndfc50 is 30242 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 30242: 2.43718e-14F
parasitic_noprobe_def2para:Netname: ndfc62 is 10330 in parasitic_noprobe
parasitic_noprobe_parcapsum:Total parasitic cap. on net 10330: 2.24203e-14F
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

cordic_ochp_near_candidates

```
------------------------------------------------------------
Candidate 1:

OCHP net: 20287   Num. inst.: 1
  Probe 1: ochprb#2fDFC_15#2fDFC_23#2fdebugpin_tri
  Total probes: 6/42 (14.285714%)

    Instance present on probed net: U4612 (OAI21M20D1_g19)
    Connected to nets: 4544 20287 8357 4536 4535 8433

    Instance present on probed net: U3333 (BUFD1_g36)
    Connected to nets: 20287 10305 10304 33400

    Instance present on probed net: U129 (OA21D1_g9)
    Connected to nets: 20287 20291 29855 1555 1554 8360

    Instance present on probed net: U3205 (EXOR2D1_g29)
    Connected to nets: 20287 20315 8908 8907 8906

    Instance present on probed net: U3234 (INVD0_g45)
    Connected to nets: 20287 20280 20279 20276

    Instance present on probed net: U3235 (INVD1_g13)
    Connected to nets: 20287 2283 2282 33389

    Instance present on probed net: U368 (OR2D1_g22)
    Connected to nets: 20315 20287 8263 8262 26082

    Instance present on probed net: U370 (NAN2D1_g11)
    Connected to nets: 10287 20287 2830 1673 1672

    Instance present on probed net: reg_etage_1#2fouta_reg_7_ (DFFRPQ1_g32)
    Connected to nets: 26051 9577 20287 22825 9568 10304

------------------------------------------------------
Candidate 2:

OCHP net: 20381   Num. inst.: 1
  Probe 1: ochprb#2fDFC_0#2fDFC_3#2fdebugpin_tri
  Total probes: 11/42 (26.190476%)

    Instance present on probed net: U3328 (INVD0_g45)
    Connected to nets: 20386 20385 20382 20381

    Instance present on probed net: UDFC3 (BUFD1_g36)
    Connected to nets: 20381 10474 20382 33422

    Instance present on probed net: reg_etage_3#2foutmode_reg (DFFRPQ1_g32)
    Connected to nets: 26051 20381 9557 21098 9556 9553

    Instance present on probed net: U3441 (OR2D1_g22)
    Connected to nets: 9478 20381 8237 8236 8235

    Instance present on probed net: U3468 (OR2D1_g22)
    Connected to nets: 20381 33554 8223 8218 20582

------------------------------------------------------------
Candidate 3:

OCHP net: 10454   Num. inst.: 1
  Probe 1: ochprb#2fDFC_0#2fDFC_30#2fdebugpin_tri
  Total probes: 10/42 (23.809524%)
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
    Instance present on probed net: U3326 (BUFD1_g36)
    Connected to nets: 10296 10295 24964 10454

    Instance present on probed net: U3327 (INVD1_g13)
    Connected to nets: 10454 2826 2825 20380

    Instance present on probed net: UDFC30 (BUFD1_g36)
    Connected to nets: 10454 10453 10452 33112

-----------------------------------------------------
Candidate 4:

OCHP net: 10463   Num. inst.: 1
  Probe 1: ochprb#2fDFC_0#2fDFC_26#2fdebugpin_tri
  Total probes: 23/42 (54.761905%)

    Instance present on probed net: U4609 (OAI21M20D1_g19)
    Connected to nets: 10463 29864 4534 4533 4532 8368

    Instance present on probed net: U4618 (OAI21M20D1_g19)
    Connected to nets: 10463 29867 23435 4543 4542 8810

    Instance present on probed net: UDFC26 (BUFD1_g36)
    Connected to nets: 10463 10462 10779 33108

    Instance present on probed net: U3349 (BUFD1_g36)
    Connected to nets: 10316 10315 10314 10463

    Instance present on probed net: U3350 (MUX2D1_g23)
    Connected to nets: 29864 10033 10463 8436 8435 8434

    Instance present on probed net: U3351 (MUX2D1_g23)
    Connected to nets: 19761 8439 10463 8438 8437 9969

    Instance present on probed net: U3352 (NAN2D1_g11)
    Connected to nets: 29868 10463 1793 1792 1791

    Instance present on probed net: U3353 (OR2D1_g22)
    Connected to nets: 10463 29880 8241 8240 9971

    Instance present on probed net: U1629 (NOR2D1_g14)
    Connected to nets: 10463 19813 3995 3994 29862
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
--------------------------------------------------------------------------------
-- Title         : Cordic Slice Mode Vecteur et Rotation + OCHP
-- Project       : ELE4304 Projet H2004
--------------------------------------------------------------------------------
-- File          : cordic_ochp.vhd
-- Authors       : Sebastien REGIMBAL  <regimbal@vlsi103.vlsi.polymtl.ca>
--               : Kevin PETERSON      <peterson@vlsi103.vlsi.polymtl.ca>
-- Created       : 2004/02/24
-- Last modified : 2004/04/01
--------------------------------------------------------------------------------
-- Description :
--
--------------------------------------------------------------------------------
-- Modification history :
-- 2004/02/24 : created
-- 2004/04/01 : added On-Chip Hardware Probe
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.ochptypes.all;

entity cordic is
    generic (N_XY            : integer := 16;
             N_A             : integer := 16;
             N_ETAGE         : integer := 4;
             N_ITR_PAR_ETAGE : integer := 1);
    port (clk        : in std_logic;
          rst_n      : in std_logic;
          inmode     : in std_logic;          -- 1=mode Rotation 0=mode vecteur
          inx        : in std_logic_vector(N_XY-1 downto 0);
          iny        : in std_logic_vector(N_XY-1 downto 0);
          ina        : in std_logic_vector(N_A-1 downto 0);
          outmode    : out std_logic;
          outx       : out std_logic_vector(N_XY-1 downto 0);
          outy       : out std_logic_vector(N_XY-1 downto 0);
          outa       : out std_logic_vector(N_A-1 downto 0);

          -- I/O Port input signals
          PSoP       : IN std_logic;
          PEoP       : IN std_logic;
          PCfgData   : IN std_logic_vector(Cwid downto 0);

          -- I/O Port output signals
          PClkOut    : OUT std_logic;
          PConfID    : OUT std_logic_vector(Cwid downto 0);
          PDebugData : OUT std_logic_vector((P-1) downto 0)
          );

end cordic;

architecture rtl of cordic is
    -- Declaration des constantes de arctan
    constant N_MAX_ITERATION : integer := 12;
    constant N_ITR_TOTALE    : integer := N_ETAGE*N_ITR_PAR_ETAGE;
    type atan_typ is array (1 to N_MAX_ITERATION) of std_logic_vector(N_A-1 downto 0);
    constant atan_table : atan_typ   := ("0001011010000000", -- 45.00000000000000
                                         "0000110101001000", -- 26.56505117707799
                                         "0000011100000100", -- 14.03624346792648
                                         "0000001110010000", -- 7.12501634890180
                                         "0000000111001001", -- 3.57633437499735
                                         "0000000011100101", -- 1.78991060824607
                                         "0000000001110010", -- 0.89517371021107
                                         "0000000000111001", -- 0.44761417086055
                                         "0000000000011100", -- 0.22381050036854
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
                                    "0000000000001110", -- 0.11190567706621
                                    "0000000000000111", -- 0.05595289189380
                                    "0000000000000011");-- 0.02797645261700);
signal atan_const : atan_typ;

-- Declaration des fils internes
constant N_REG   : integer := 1 + N_XY + N_XY + N_A;
type int_xy_typ   is array (integer range <>) of std_logic_vector(N_XY-1 downto 0);
type int_a_typ    is array (integer range <>) of std_logic_vector(N_A-1 downto 0);
type int_mode_typ is array (integer range <>) of std_logic;

signal int_mode  : int_mode_typ(1 to (N_ETAGE + N_ETAGE*N_ITR_PAR_ETAGE +1));
signal int_x     : int_xy_typ(1 to (N_ETAGE + N_ETAGE*N_ITR_PAR_ETAGE +1));
signal int_y     : int_xy_typ(1 to (N_ETAGE + N_ETAGE*N_ITR_PAR_ETAGE +1));
signal int_a     : int_a_typ(1 to (N_ETAGE + N_ETAGE*N_ITR_PAR_ETAGE +1));
signal ochprobes : std_logic_vector((N-1) downto 0);

-- Declaration du module combinatoire
component cordic_rot_vec
   generic (N_XY   : integer := 16;
            N_A    : integer := 16;
            ITR    : integer := 1);
   port (inmode  : in std_logic;         -- 1=mode Rotation 0=mode vecteur
         inx     : in std_logic_vector(N_XY-1 downto 0);
         iny     : in std_logic_vector(N_XY-1 downto 0);
         ina     : in std_logic_vector(N_A-1 downto 0);
         consta  : in std_logic_vector(N_A-1 downto 0);
         outmode : out std_logic;
         outx    : out std_logic_vector(N_XY-1 downto 0);
         outy    : out std_logic_vector(N_XY-1 downto 0);
         outa    : out std_logic_vector(N_A-1 downto 0)
         );
end component;

-- Declaration du registre
component cordic_reg
   generic (N_XY  : integer := 16;
            N_A   : integer := 16);
   port (clk     : in  std_logic;
         rst_n   : in  std_logic;
         inmode  : in  std_logic;
         inx     : in  std_logic_vector(N_XY-1 downto 0);
         iny     : in  std_logic_vector(N_XY-1 downto 0);
         ina     : in  std_logic_vector(N_A-1 downto 0);
         outmode : out std_logic;
         outx    : out std_logic_vector(N_XY-1 downto 0);
         outy    : out std_logic_vector(N_XY-1 downto 0);
         outa    : out std_logic_vector(N_A-1 downto 0)
         );
end component;

component ochp
  port (
    -- Control signals
    clk                    : IN    std_logic;
    reset_n                : IN    std_logic;

    -- Internal signals
    PIntSignal             : IN    std_logic_vector((N-1) downto 0);

    -- I/O Port input signals
    PSoP                   : IN    std_logic;
    PEoP                   : IN    std_logic;
    PCfgData               : IN    std_logic_vector(Cwid downto 0);

    -- I/O Port output signals
    PClkOut                : OUT   std_logic;
    PConfID                : OUT   std_logic_vector(Cwid downto 0);
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
        PDebugData              : OUT   std_logic_vector((P-1) downto 0)
        );
    end component;

begin  -- rtl
    int_mode(1) <= inmode;
    int_x(1)    <= inx;
    int_y(1)    <= iny;
    int_a(1)    <= ina;
    atan_const    <= atan_table;

    gen_etage : for itr0 in 1 to N_ETAGE generate
        -- Instanciaton du registre
        reg_etage : cordic_reg
            generic map (N_XY => N_XY, N_A => N_A)
            port map (clk          => clk,
                      rst_n        => rst_n,
                      inmode       => int_mode(itr0 +(itr0-1)*N_ITR_PAR_ETAGE),
                      inx          => int_x(itr0 +(itr0-1)*N_ITR_PAR_ETAGE),
                      iny          => int_y(itr0 +(itr0-1)*N_ITR_PAR_ETAGE),
                      ina          => int_a(itr0 +(itr0-1)*N_ITR_PAR_ETAGE),
                      outmode      => int_mode(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+1),
                      outx         => int_x(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+1),
                      outy         => int_y(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+1),
                      outa         => int_a(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+1)
                      );
        -- Instanciation des modules Cordic
        gen_cor_mod1 : for  itr1 in 1 to N_ITR_PAR_ETAGE generate
            cor_module : cordic_rot_vec
                generic map (N_XY => N_XY,
                             N_A  => N_A,
                             ITR  => ((itr0-1)*N_ITR_PAR_ETAGE+itr1))
                port map (inmode  => int_mode(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+itr1),
                          inx     => int_x(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+itr1),
                          iny     => int_y(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+itr1),
                          ina     => int_a(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+itr1),
                          consta  => atan_const((itr0-1)*N_ITR_PAR_ETAGE+itr1),
                          outmode => int_mode(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+itr1+1),
                          outx    => int_x(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+itr1+1),
                          outy    => int_y(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+itr1+1),
                          outa    => int_a(itr0 +(itr0-1)*N_ITR_PAR_ETAGE+itr1+1)
                          );

        end generate gen_cor_mod1;
    end generate gen_etage;

    -- Instanciation du registre de sortie
    reg_out : cordic_reg
        generic map (N_XY => N_XY, N_A => N_A)
        port map (clk          => clk,
                  rst_n        => rst_n,
                  inmode       => int_mode(N_ETAGE + N_ETAGE*N_ITR_PAR_ETAGE +1),
                  inx          => int_x(N_ETAGE + N_ETAGE*N_ITR_PAR_ETAGE +1),
                  iny          => int_y(N_ETAGE + N_ETAGE*N_ITR_PAR_ETAGE +1),
                  ina          => int_a(N_ETAGE + N_ETAGE*N_ITR_PAR_ETAGE +1),
                  outmode      => outmode,
                  outx         => outx,
                  outy         => outy,
                  outa         => outa
                  );

    ochprb: ochp
        port map (clk          => clk,
                  reset_n      => rst_n,
                  PIntSignal   => ochprobes,
                  PSoP         => PSoP,
                  PEoP         => PEoP,
                  PCfgData     => PCfgData,
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
                PClkOut       => PClkOut,
                PConfID       => PConfID,
                PDebugData    => PDebugData
                );

    -- Connexion des sondes
    ochprobes(0)         <= int_mode(1);
    ochprobes(1)         <= int_mode(2);
    ochprobes(2)         <= int_mode(3);
    ochprobes(3)         <= int_mode(4);

    ochprobes(31 downto 16) <= int_a(2);
    ochprobes(47 downto 32) <= int_a(3);
    ochprobes(63 downto 48) <= int_a(4);

end rtl;
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
------------------------------------------------------------
-- Description : On-Chip Hardware Probe parameters file
--
-- File    : ochp_param.vhd
-- Author : K. Peterson
-- Date    : Jan 21 2004
--
-- Proj    : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

package ochptypes is
  constant C:         integer := 8;     -- Number of configurations
  constant Cwid:      integer := 2;     -- log2C - 1
  constant N:         integer := 64;    -- Number of internal signals
  constant Nwid:      integer := 5;     -- log2N - 1
  constant P:         integer := 16;    -- Number of external pins
  constant Pwid:      integer := 3;     -- log2P - 1
end ochptypes;
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
--------------------------------------------------------------------------------
-- Title         : Cordic Slice Mode Vecteur et Rotation
-- Project       : ELE4304 Projet H2004
--------------------------------------------------------------------------------
-- File          : cordic_slice.vhd
-- Author        : Sebastien REGIMBAL  <regimbal@vlsi103.vlsi.polymtl.ca>
-- Created       : 2004/02/24
-- Last modified : 2004/02/24
--------------------------------------------------------------------------------
-- Description :
--
--------------------------------------------------------------------------------
-- Modification history :
-- 2004/02/24 : created
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity cordic_rot_vec is
    generic (N_XY   : integer := 16;
             N_A    : integer := 16;
             ITR    : integer := 1);
    port (inmode  : in std_logic;     -- 1=mode Rotation 0=mode vecteur
          inx     : in std_logic_vector(N_XY-1 downto 0);
          iny     : in std_logic_vector(N_XY-1 downto 0);
          ina     : in std_logic_vector(N_A-1 downto 0);
          consta  : in std_logic_vector(N_A-1 downto 0);
          outmode : out std_logic;
          outx    : out std_logic_vector(N_XY-1 downto 0);
          outy    : out std_logic_vector(N_XY-1 downto 0);
          outa    : out std_logic_vector(N_A-1 downto 0));
end cordic_rot_vec;

architecture rtl of cordic_rot_vec is
   function shiftVector (signal   val      : in std_logic_vector(N_XY-1 downto 0);
                         constant nb_shift : in integer) return std_logic_vector is
       variable result : std_logic_vector(N_XY-1 downto 0);
   begin  -- shiftVector
       if val(N_XY - 1) = '0' then
           -- Shift positive
           result := (others => '0');
       else
           -- shift negative
           result := (others => '1');
       end if;
       result(N_XY-ITR downto 0) := val(N_XY-1 downto ITR-1);
       return result;
   end shiftVector;
begin  -- rtl

    update_outputs : process (inx,iny,ina,inmode,consta)
        variable dx : std_logic_vector(N_XY-1 downto 0);
        variable dy : std_logic_vector(N_XY-1 downto 0);
    begin  -- process update_outputs
        dx := shiftVector(inx,ITR);
        dy := shiftVector(iny,ITR);

        if inmode = '1' then
            if ina(N_A-1) = '1' then
                outx <= inx + dy;
                outy <= iny - dx;
                outa <= ina + consta;
            else
                outx <= inx - dy;
                outy <= iny + dx;
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
                outa <= ina - consta;
            end if;
        else
            if iny(N_XY-1) = '1' then
                outx <= inx - dy;
                outy <= iny + dx;
                outa <= ina - consta;
            else
                outx <= inx + dy;
                outy <= iny - dx;
                outa <= ina + consta;
            end if;
        end if;

        outmode <= inmode;
    end process update_outputs;
end rtl;
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

```
-------------------------------------------------------------------------------
-- Title         : Cordic Register
-- Project       : ELE4304 Projet H2004
-------------------------------------------------------------------------------
-- File          : cordic_reg.vhd
-- Author        : Sebastien REGIMBAL  <regimbal@vlsi103.vlsi.polymtl.ca>
-- Created       : 2004/02/24
-- Last modified : 2004/02/24
-------------------------------------------------------------------------------
-- Description :
--
-------------------------------------------------------------------------------
-- Modification history :
-- 2004/02/24 : created
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity cordic_reg is
    generic (N_XY  : integer := 16;
             N_A   : integer := 16);
    port (clk     : in  std_logic;
          rst_n   : in  std_logic;
          inmode  : in  std_logic;
          inx     : in  std_logic_vector(N_XY-1 downto 0);
          iny     : in  std_logic_vector(N_XY-1 downto 0);
          ina     : in  std_logic_vector(N_A-1 downto 0);
          outmode : out std_logic;
          outx    : out std_logic_vector(N_XY-1 downto 0);
          outy    : out std_logic_vector(N_XY-1 downto 0);
          outa    : out std_logic_vector(N_A-1 downto 0)
          );
end cordic_reg;

architecture rtl of cordic_reg is

begin  -- rtl
    reg_q : process (clk, rst_n)

    begin  -- process reg_q
        -- activities triggered by asynchronous reset (active low)
        if rst_n = '0' then
            outmode <= '0';
            outx    <= (others => '0');
            outy    <= (others => '0');
            outa    <= (others => '0');
        -- activities triggered by rising edge of clock
        elsif clk'event and clk = '1' then
            outmode <= inmode;
            outx    <= inx;
            outy    <= iny;
            outa    <= ina;
        end if;
    end process reg_q;
end rtl;
```

ANNEXE D : Scripts et résultats des analyses de l'effet de sonde

cordic_ochp_analysis_040519.xls

Net name	Par. Cap. No probe	Par. Cap. OCHP	% variation	Trace length no probe	Trace length OCHP	% variation	Vmax no probe	Vmax OCHP	% variation		
int_a_2\N10_	2,70E-13	2,65E-13	-1,77%	1548000	2406400	55,45%	1,54	1,43	-7,14%	Candidat3	12587
int_a_3\N2_	2,56E-13	2,14E-13	-16,38%	1487400	2022800	36,00%					24276
int_a_2\N7_	2,43E-13	2,74E-13	12,64%	2065320	2388120	15,63%	1,55	1,45	-6,45%	Candidat1	24234
int_a_3\N5_	2,24E-13	2,39E-13	6,41%	1529780	1827780	19,48%	1,65	1,58	-4,24%	Candidat4	24328
int_a_2\N13_	1,77E-13	1,85E-13	4,85%	1603260	1854180	15,65%				Candidat2	31621
int_a_3\N6_	1,64E-13	2,41E-13	46,69%	1461540	1586220	8,53%					24338
int_a_3\N3_	1,57E-13	2,23E-13	42,46%	1520780	2102500	38,25%					24286
int_a_3\N1_	1,39E-13	2,31E-13	66,31%	1365440	1954720	43,16%					24296
int_a_3\N0_	1,36E-13	2,04E-13	49,56%	1258000	1791820	42,43%					24318
n676	7,65E-14	1,53E-13	99,89%	844500	1433760	69,78%					12591
int_a_2\N7_	1,32E-13	1,37E-13	3,42%	1488980	1488980	0,00%	1,78	1,77	-0,56%	Candidat1	20287
n669	9,16E-14	9,17E-14	0,13%	928160	928160	0,00%	1,8	1,8	0,00%	Candidat2	20381
n2308	7,46E-14	7,47E-14	0,09%	657360	657360	0,00%	1,79	1,78	-0,56%	Candidat3	10454
n675	5,92E-14	6,00E-14	1,22%	703820	703820	0,00%			#DIV/0!	Candidat4	10463

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Assertion-Checker FPGA top file
--
-- File   : acf_top.vhd
-- Author : K. Peterson
-- Created: Sep 16 2004
-- Modif. : Sep 22 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
use work.acftypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY acf_top IS
  PORT (clk         : IN  std_logic;
        reset_n     : IN  std_logic;

        -- ACF CPU Interface
        we_n        : IN  std_logic;
        regaddr     : IN  std_logic_vector(31 downto 0);
        regdatain   : IN  std_logic_vector(31 downto 0);
        regdataout  : OUT std_logic_vector(31 downto 0);
        irq         : OUT std_logic;

        -- Assertion RAM Interface
        ramwe_n     : OUT std_logic;
        ramoe_n     : OUT std_logic;
        ramdatain   : IN  std_logic_vector((dbgbuswidth-1) downto 0);
        ramdataout  : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        ramaddr     : OUT std_logic_vector(assramwid downto 0);

        -- OCHP Interface
        -- I/O Port input signals
        PSoP        : OUT std_logic;
        PEoP        : OUT std_logic;
        PCfgData    : OUT std_logic_vector(Cwid downto 0);

        -- I/O Port output signals
        PConfID     : IN  std_logic_vector(Cwid downto 0);
        PDebugData  : IN  std_logic_vector((P-1) downto 0)
  );

END acf_top;

ARCHITECTURE rtl OF acf_top IS

  COMPONENT cm_ram
    PORT (clk        : IN  std_logic;

          -- Configuration RAM I/O
          ramwe_n    : IN  std_logic;
          ramoe_n    : IN  std_logic;
          ramdatain  : IN  std_logic_vector(Nwid downto 0);
          ramdataout : OUT std_logic_vector(Nwid downto 0);
          ramaddr    : IN  std_logic_vector((Pwid+Cwid+1) downto 0)
    );
  END COMPONENT;

  COMPONENT acfgenregs
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- ACF regs control
          we_n       : IN  std_logic;
```

ANNEXE E : Sources ACF

```
        -- ACF regs status signals
        trig_armed  : IN  std_logic;
        storing_dat : IN  std_logic;
        cfg_stored  : IN  std_logic;
        dumping_cfg : IN  std_logic;
        cfg_dumped  : IN  std_logic;
        irq_set     : IN  std_logic;

        -- ACF regs ext. control signals
        arm_trigger : OUT std_logic;
        force_trig  : OUT std_logic;
        stop_stor   : OUT std_logic;
        clr_irq     : OUT std_logic;
        reset_tt    : OUT std_logic;
        reset_cm    : OUT std_logic;
        dump_cfg    : OUT std_logic;
        soft_reset  : OUT std_logic;

        -- ACF regs reg data input
        regdatain   : IN  std_logic_vector(31 downto 0);
        regaddr     : IN  std_logic_vector(1 downto 0);

        -- ACF regs data output
        regdataout  : OUT std_logic_vector(31 downto 0)
  );
end COMPONENT;

COMPONENT ac_eg_module
  PORT (clk         : IN  std_logic;
        reset_n     : IN  std_logic;

        -- OCHP debug signals input
        PConfId     : IN  std_logic_vector(Cwid downto 0);
        PDebugData  : IN  std_logic_vector((P-1) downto 0);

        -- AC controller signals
        ACCwe_n     : IN  std_logic;
        ACCout_addr : IN  std_logic_vector(nACwid downto 0);
        ACCdat_type : IN  std_logic;
        ACCresetin  : IN  std_logic_vector(31 downto 0);
        ACCdatar    : OUT std_logic_vector((dbgbuswidth-1) downto 0);

        -- EG controller signals
        EGCwe_n     : IN  std_logic;
        EGCout_addr : IN  std_logic_vector(nEGwid downto 0);
        EGCdat_type : IN  std_logic;
        EGCtransdes : IN  std_logic;
        EGCclrirq   : IN  std_logic;
        EGCcfgin    : IN  std_logic_vector(31 downto 0);
        EGCdatar    : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        EGCirq      : OUT std_logic;

        -- Timestamp Timer signals
        TTin        : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Assertion RAM signals
        ARAMdbgdata : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        ARAMegevent : OUT std_logic_vector((nEG-1) downto 0);

        -- Configuration Manager signals
        CMcfgswap   : OUT std_logic;
        CMswapid    : OUT std_logic_vector(Cwid downto 0)
  );
end COMPONENT;

COMPONENT assertram_ctrl
  PORT (clk         : IN  std_logic;
        reset_n     : IN  std_logic;
```

ANNEXE E : Sources ACF

```
        -- AssertRAM control signals
        we_n        : IN  std_logic;
        read_ram    : IN  std_logic;
        arm_trigger : IN  std_logic;
        force_trig  : IN  std_logic;
        stop_stor   : IN  std_logic;

        -- AssertRAM status
        trig_armed  : OUT std_logic;
        storing_dat : OUT std_logic;
        regdataout  : OUT std_logic_vector(31 downto 0);

        -- AssertRAM reg data input
        regdatain   : IN  std_logic_vector(31 downto 0);
        regaddr     : IN  std_logic_vector(3 downto 0);
        aramaddr    : IN  std_logic_vector(31 downto 0);

        -- AssertRAM dbg data input
        dbgdata_in  : IN  std_logic_vector((dbgbuswidth-1) downto 0);
        egevent_in  : IN  std_logic_vector((nEG-1) downto 0);
        ttin        : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Assertion RAM I/O
        ramwe_n     : OUT std_logic;
        ramoe_n     : OUT std_logic;
        ramdatain   : IN  std_logic_vector((dbgbuswidth-1) downto 0);
        ramdataout  : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        ramaddr     : OUT std_logic_vector(assramwid downto 0)
    );
end COMPONENT;

COMPONENT cm_ctrl
    PORT (clk       : IN  std_logic;
        reset_n     : IN  std_logic;

        -- CM control signals
        we_n        : IN  std_logic;
        eg_cfgswap  : IN  std_logic;
        dump_cfg    : IN  std_logic;

        -- CM cfg status
        cfg_stored  : OUT std_logic;
        dumpingcfg  : OUT std_logic;
        cfg_dumped  : OUT std_logic;

        -- CM cfg data input
        cfgswapid   : IN  std_logic_vector(Cwid downto 0);
        cfgdatain   : IN  std_logic_vector(31 downto 0);
        cfgaddr     : IN  std_logic_vector(Cwid downto 0);

        -- Configuration RAM I/O
        ramwe_n     : OUT std_logic;
        ramoe_n     : OUT std_logic;
        ramdatain   : IN  std_logic_vector(Nwid downto 0);
        ramdataout  : OUT std_logic_vector(Nwid downto 0);
        ramaddr     : OUT std_logic_vector((Pwid+Cwid+1) downto 0);

        -- I/O Port output signals
        FSoP        : OUT   std_logic;
        FEoP        : OUT   std_logic;
        FCfgData    : OUT   std_logic_vector(Cwid downto 0)
    );
end COMPONENT;

COMPONENT dataformatter
    PORT (clk       : IN  std_logic;
        reset_n     : IN  std_logic;

        -- ACF regs control
        we_n        : IN  std_logic;
```

ANNEXE E : Sources ACF

```
        -- ACF general registers signals
        GRwe_n       : OUT std_logic;
        GRregdatain  : OUT std_logic_vector(31 downto 0);
        GRregaddr    : OUT std_logic_vector(1 downto 0);
        GRregdataout : IN  std_logic_vector(31 downto 0);

        -- Timestamp Timer signals
        TTwe_n       : OUT std_logic;
        TTcfgin      : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        TTmaxcnt     : IN  std_logic_vector((dbgbuswidth-1) downto 0);
        TTcurtime    : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Assertion Checkers signals

        ACwe_n       : OUT std_logic;
        ACout_addr   : OUT std_logic_vector(nACwid downto 0);
        ACrdat_type  : OUT std_logic;
        ACresetin    : OUT std_logic_vector(31 downto 0);
        ACdatar      : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Event Generator signals
        EGwe_n       : OUT std_logic;
        EGout_addr   : OUT std_logic_vector(nEGwid downto 0);
        EGrdat_type  : OUT std_logic;
        EGtransdest  : OUT std_logic;

        EGcfgin      : OUT std_logic_vector(31 downto 0);
        EGdatar      : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Configuration Manager signals
        CMwe_n       : OUT std_logic;
        CMcfgdatain  : OUT std_logic_vector(31 downto 0);
        CMcfgaddr    : OUT std_logic_vector(Cwid downto 0);

        -- Assertion RAM signals
        ARwe_n       : OUT std_logic;
        ARread_ram   : OUT std_logic;
        ARregdataout : IN  std_logic_vector(31 downto 0);
        ARregdatain  : OUT std_logic_vector(31 downto 0);
        ARregaddr    : OUT std_logic_vector(3 downto 0);
        ARaramaddr   : OUT std_logic_vector(31 downto 0);

        -- ACF registers data
        regaddr      : IN  std_logic_vector(31 downto 0);
        regdatain    : IN  std_logic_vector(31 downto 0);
        regdataout   : OUT std_logic_vector(31 downto 0)
  );
end COMPONENT;

COMPONENT timestamptimer
  GENERIC (
    dbginfowidth    : integer
  );
  PORT (clk         : IN  std_logic;
        reset_n     : IN  std_logic;

        -- AC/EG control signals
        clrtimer    : IN  std_logic;
        we_n        : IN  std_logic;

        -- AC/EG debug data input
        cfgin       : IN  std_logic_vector((dbginfowidth-1) downto 0);

        -- AC/EG debug data output
        ttmaxcnt    : OUT std_logic_vector((dbginfowidth-1) downto 0);
        ttcurtime   : OUT std_logic_vector((dbginfowidth-1) downto 0)

  );
end COMPONENT;
```

ANNEXE E : Sources ACF

```
  SIGNAL MasterReset_n : std_logic;

  SIGNAL ARtrig_armed  : std_logic;
  SIGNAL ARstoring_dat : std_logic;
  SIGNAL CMcfg_stored  : std_logic;
  SIGNAL CMdumping_cfg : std_logic;
  SIGNAL CMcfg_dumped  : std_logic;
  SIGNAL EGirq_set     : std_logic;
  SIGNAL ARarm_trigger : std_logic;
  SIGNAL ARforce_trig  : std_logic;
  SIGNAL ARstop_stor   : std_logic;
  SIGNAL EGclr_irq     : std_logic;
  SIGNAL GRreset_cm    : std_logic;
  SIGNAL CMreset_cm    : std_logic;
  SIGNAL CMdump_cfg    : std_logic;
  SIGNAL GRsoft_reset  : std_logic;
  SIGNAL ARAMdbgdata   : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL ARAMegevent   : std_logic_vector((nEG-1) downto 0);
  SIGNAL CMcfgswap     : std_logic;
  SIGNAL CMswapid      : std_logic_vector(Cwid downto 0);
  SIGNAL CMramwe_n     : std_logic;
  SIGNAL CMramoe_n     : std_logic;
  SIGNAL CMramdatain   : std_logic_vector(Nwid downto 0);
  SIGNAL CMramdataout  : std_logic_vector(Nwid downto 0);
  SIGNAL CMramaddr     : std_logic_vector((Pwid+Cwid+1) downto 0);
  SIGNAL GRwe_n        : std_logic;
  SIGNAL GRregdatain   : std_logic_vector(31 downto 0);
  SIGNAL GRregaddr     : std_logic_vector(1 downto 0);
  SIGNAL GRregdataout  : std_logic_vector(31 downto 0);

  SIGNAL TTwe_n        : std_logic;
  SIGNAL TTcfgin       : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL TTmaxcnt      : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL TTcurtime     : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL ACwe_n        : std_logic;
  SIGNAL ACout_addr    : std_logic_vector(nACwid downto 0);
  SIGNAL ACrdat_type   : std_logic;
  SIGNAL ACresetin     : std_logic_vector(31 downto 0);
  SIGNAL ACdatar       : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL EGwe_n        : std_logic;
  SIGNAL EGout_addr    : std_logic_vector(nEGwid downto 0);
  SIGNAL EGrdat_type   : std_logic;
  SIGNAL EGtransdest   : std_logic;
  SIGNAL EGcfgin       : std_logic_vector(31 downto 0);
  SIGNAL EGdatar       : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL CMwe_n        : std_logic;
  SIGNAL CMcfgdatain   : std_logic_vector(31 downto 0);
  SIGNAL CMcfgaddr     : std_logic_vector(Cwid downto 0);
  SIGNAL ARwe_n        : std_logic;
  SIGNAL ARread_ram    : std_logic;
  SIGNAL ARregdataout  : std_logic_vector(31 downto 0);
  SIGNAL ARregdatain   : std_logic_vector(31 downto 0);
  SIGNAL ARregaddr     : std_logic_vector(3 downto 0);
  SIGNAL ARaramaddr    : std_logic_vector(31 downto 0);
  SIGNAL TTclrtimer    : std_logic;

BEGIN

  MasterReset_n <= '0' WHEN (reset_n = '0') OR
                            (GRsoft_reset = '1') ELSE
                   '1';

  CMreset_cm    <= '0' WHEN (MasterReset_n = '0') OR
                            (GRreset_cm = '1') ELSE
                   '1';

  irq           <= EGirq_set;

  CMRAM: cm_ram
```

ANNEXE E : Sources ACF

```
  port map (clk=>clk,
            ramwe_n=>CMramwe_n,
            ramoe_n=>CMramoe_n,
            ramdatain=>CMramdataout,
            ramdataout=>CMramdatain,
            ramaddr=>CMramaddr
  );

GENREGS: acfgenregs
  port map (clk=>clk,
            reset_n=>MasterReset_n,
            we_n=>GRwe_n,
            trig_armed=>ARtrig_armed,
            storing_dat=>ARstoring_dat,
            cfg_stored=>CMcfg_stored,
            dumping_cfg=>CMdumping_cfg,
            cfg_dumped=>CMcfg_dumped,
            irq_set=>EGirq_set,
            arm_trigger=>ARarm_trigger,
            force_trig=>ARforce_trig,
            stop_stor=>ARstop_stor,
            clr_irq=>EGclr_irq,
            reset_tt=>TTclrtimer,
            reset_cm=>GRreset_cm,
            dump_cfg=>CMdump_cfg,
            soft_reset=>GRsoft_reset,
            regdatain=>GRregdatain,
            regaddr=>GRregaddr,
            regdataout=>GRregdataout
  );

ACEGMODULE: ac_eg_module
  port map (clk=>clk,
            reset_n=>MasterReset_n,
            PConfId=>PConfID,
            PDebugData=>PDebugData,
            ACCwe_n=>ACwe_n,
            ACCout_addr=>ACout_addr,
            ACCdat_type=>ACrdat_type,
            ACCresetin=>ACresetin,
            ACCdatar=>ACdatar,
            EGCwe_n=>EGwe_n,
            EGCout_addr=>EGout_addr,
            EGCdat_type=>EGrdat_type,
            EGCtransdes=>EGtransdest,
            EGCclrirq=>EGclr_irq,
            EGCcfgin=>EGcfgin,
            EGCdatar=>EGdatar,
            EGCirq=>EGirq_set,
            TTin=>TTcurtime,
            ARAMdbgdata=>ARAMdbgdata,
            ARAMegevent=>ARAMegevent,
            CMcfgswap=>CMcfgswap,
            CMswapid=>CMswapid
  );

ASSERTRAM: assertram_ctrl
  port map (clk=>clk,
            reset_n=>MasterReset_n,
            we_n=>ARwe_n,
            read_ram=>ARread_ram,
            arm_trigger=>ARarm_trigger,
            force_trig=>ARforce_trig,
            stop_stor=>ARstop_stor,
            trig_armed=>ARtrig_armed,
            storing_dat=>ARstoring_dat,
            regdataout=>ARregdataout,
            regdatain=>ARregdatain,
            regaddr=>ARregaddr,
            aramaddr=>ARaramaddr,
```

ANNEXE E : Sources ACF

```
            dbgdata_in=>ARAMdbgdata,
            egevent_in=>ARAMegevent,
            ttin=>TTcurtime,
            ramwe_n=>ramwe_n,
            ramoe_n=>ramoe_n,
            ramdatain=>ramdatain,
            ramdataout=>ramdataout,
            ramaddr=>ramaddr
  );

CONFMGR: cm_ctrl
  port map (clk=>clk,
            reset_n=>CMreset_cm,
            we_n=>CMwe_n,
            eg_cfgswap=>CMcfgswap,
            dump_cfg=>CMdump_cfg,
            cfg_stored=>CMcfg_stored,
            dumpingcfg=>CMdumping_cfg,
            cfg_dumped=>CMcfg_dumped,
            cfgswapid=>CMswapid,
            cfgdatain=>CMcfgdatain,
            cfgaddr=>CMcfgaddr,
            ramwe_n=>CMramwe_n,
            ramoe_n=>CMramoe_n,
            ramdatain=>CMramdatain,
            ramdataout=>CMramdataout,
            ramaddr=>CMramaddr,
            FSoP=>PSoP,
            FEoP=>PEoP,
            FCfgData=>PCfgData
  );

DATAFORM: dataformatter
  port map (clk=>clk,
            reset_n=>MasterReset_n,
            we_n=>we_n,
            GRwe_n=>GRwe_n,
            GRregdatain=>GRregdatain,
            GRregaddr=>GRregaddr,
            GRregdataout=>GRregdataout,
            TTwe_n=>TTwe_n,
            TTcfgin=>TTcfgin,
            TTmaxcnt=>TTmaxcnt,
            TTcurtime=>TTcurtime,
            ACwe_n=>ACwe_n,
            ACout_addr=>ACout_addr,
            ACrdat_type=>ACrdat_type,
            ACresetin=>ACresetin,
            ACdatar=>ACdatar,
            EGwe_n=>EGwe_n,
            EGout_addr=>EGout_addr,
            EGrdat_type=>EGrdat_type,
            EGtransdest=>EGtransdest,
            EGcfgin=>EGcfgin,
            EGdatar=>EGdatar,
            CMwe_n=>CMwe_n,
            CMcfgdatain=>CMcfgdatain,
            CMcfgaddr=>CMcfgaddr,
            ARwe_n=>ARwe_n,
            ARread_ram=>ARread_ram,
            ARregdataout=>ARregdataout,
            ARregdatain=>ARregdatain,
            ARregaddr=>ARregaddr,
            ARaramaddr=>ARaramaddr,
            regaddr=>regaddr,
            regdatain=>regdatain,
            regdataout=>regdataout
  );

TSTAMPTIMER: timestamptimer
```

ANNEXE E : Sources ACF

```
    generic map (dbgbuswidth)
    port map (clk=>clk,
              reset_n=>MasterReset_n,
              clrtimer=>TTclrtimer,
              we_n=>TTwe_n,
              cfgin=>TTcfgin,
              ttmaxcnt=>TTmaxcnt,
              ttcurtime=>TTcurtime
    );

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Timestamp Timer
--
-- File   : tstamptimer.vhd
-- Author : K. Peterson
-- Date   : Jul 1 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY timestamptimer IS
  GENERIC (
    dbginfowidth   : integer   := 32

  );
  PORT (clk        : IN  std_logic;
        reset_n    : IN  std_logic;

        -- AC/EG control signals
        clrtimer   : IN  std_logic;
        we_n       : IN  std_logic;

        -- AC/EG debug data input
        cfgin      : IN  std_logic_vector((dbginfowidth-1) downto 0);

        -- AC/EG debug data output
        ttmaxcnt   : OUT std_logic_vector((dbginfowidth-1) downto 0);
        ttcurtime  : OUT std_logic_vector((dbginfowidth-1) downto 0)
  );

END timestamptimer;

ARCHITECTURE rtl OF timestamptimer IS
  CONSTANT DEF_MAXCNT : std_logic_vector((dbginfowidth-1) downto 0) := (OTHERS => '1');
  CONSTANT ZERO       : std_logic_vector((dbginfowidth-1) downto 0) := (OTHERS => '0');

  SIGNAL maxcntreg :    std_logic_vector((dbginfowidth-1) downto 0);
  SIGNAL timerreg  :    std_logic_vector((dbginfowidth-1) downto 0);

BEGIN

  ttmaxcnt <= maxcntreg;
  ttcurtime <= timerreg;

  maxcnt_reg: PROCESS (clk, reset_n)
  BEGIN

    IF (reset_n = '0') THEN
      maxcntreg <= DEF_MAXCNT;

    ELSIF clk'event AND clk = '1' THEN

      IF we_n = '0' THEN
        maxcntreg <= cfgin;
      END IF;

    END IF;

  END PROCESS;

  timerincr: PROCESS (clk, reset_n, clrtimer)
  BEGIN

    IF (reset_n = '0') OR (clrtimer = '1') THEN
```

ANNEXE E : Sources ACF

```
      timerreg  <= ZERO;

    ELSIF clk'event AND clk = '1' THEN

      IF timerreg >= maxcntreg THEN
        timerreg <= ZERO;
      ELSE
        timerreg <= timerreg + 1;
      END IF;

    END IF;

  END PROCESS;

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Assertion Checkers + Event Generators
--               module
--               Synthesized AC and EG specific to ISA
--               bus interface + OCHP design
--
-- File   : ac_eg_module_ISA_ochp_set1.vhd
-- Author : K. Peterson
-- Created: Oct 29 2004
-- Modif. : Oct 31 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
use work.acftypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY ac_eg_module IS
  PORT (clk          : IN  std_logic;
        reset_n      : IN  std_logic;

        -- OCHP debug signals input
        PConfId      : IN  std_logic_vector(Cwid downto 0);
        PDebugData   : IN  std_logic_vector((P-1) downto 0);

        -- AC controller signals
        ACCwe_n      : IN  std_logic;
        ACCout_addr  : IN  std_logic_vector(nACwid downto 0);
        ACCdat_type  : IN  std_logic;
        ACCresetin   : IN  std_logic_vector(31 downto 0);
        ACCdatar     : OUT std_logic_vector((dbgbuswidth-1) downto 0);

        -- EG controller signals
        EGCwe_n      : IN  std_logic;
        EGCout_addr  : IN  std_logic_vector(nEGwid downto 0);
        EGCdat_type  : IN  std_logic;
        EGCtransdes  : IN  std_logic;
        EGCclrirq    : IN  std_logic;
        EGCcfgin     : IN  std_logic_vector(31 downto 0);
        EGCdatar     : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        EGCirq       : OUT std_logic;

        -- Timestamp Timer signals
        TTin         : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Assertion RAM signals
        ARAMdbgdata  : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        ARAMegevent  : OUT std_logic_vector((nEG-1) downto 0);

        -- Configuration Manager signals
        CMcfgswap    : OUT std_logic;
        CMswapid     : OUT std_logic_vector(Cwid downto 0)
  );

END ac_eg_module;

ARCHITECTURE rtl OF ac_eg_module IS

    component assert_always
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
```

ANNEXE E : Sources ACF

```
        port (
          clk, reset_n : IN std_ulogic;
          test_expr    : IN boolean;
          valide       : OUT std_ulogic);
    end component;

    component assert_never
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
        port (
          clk, reset_n : IN std_ulogic;
          test_expr    : IN boolean;
          valide       : OUT std_ulogic);
    end component;

    component assert_implication
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
        port (
          clk, reset_n                 : IN std_ulogic;
          antecedent_expr, consequent_expr : IN boolean;
          valide                       : OUT std_ulogic);
    end component;


  COMPONENT ac_harness
    GENERIC (
      dbginfowidth   : integer   := 32;
      dbginfotype    : integer   := 0;  -- Default: Timestamp register
      failedlatched  : integer   := 0   -- Default: Failed not latched
    );
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- Assertion checker control signals
          clrfail    : IN  std_logic;
          clrinforeg : IN  std_logic;
          ACenable   : IN  std_logic;

          -- Assertion checker debug data input
          dbginput   : IN  std_logic_vector((dbginfowidth-1) downto 0);
          validin    : IN  std_logic;

          -- Assertion checker debug data output
          dbginfo    : OUT std_logic_vector((dbginfowidth-1) downto 0);
          failed_n   : OUT std_logic);
  END COMPONENT;

  COMPONENT ac_ctrl
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- AC control signals
          we_n       : IN  std_logic;
          out_addr   : IN  std_logic_vector(nACwid downto 0);
          rdat_type  : IN  std_logic;

          -- AC debug data input
          statein    : IN  std_logic_vector((nAC-1) downto 0);
          ttin       : IN  std_logic_vector(((dbgbuswidth*nAC)-1) downto 0);
          resetin    : IN  std_logic_vector(31 downto 0);
```

ANNEXE E : Sources ACF

```
        -- AC debug data output
        datar       : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        resetout    : OUT std_logic_vector((nAC-1) downto 0)
  );
end COMPONENT;

COMPONENT eg_ctrl
  PORT (clk         : IN  std_logic;
        reset_n     : IN  std_logic;

        -- EG control signals
        we_n        : IN  std_logic;
        out_addr    : IN  std_logic_vector(nEGwid downto 0);
        rdat_type   : IN  std_logic;
        transdest   : IN  std_logic;
        clrirq      : IN  std_logic;

        -- EG debug data input
        statein     : IN  std_logic_vector((nEG-1) downto 0);
        ttin        : IN  std_logic_vector(((dbgbuswidth*nEG)-1) downto 0);
        cfgin       : IN  std_logic_vector(31 downto 0);

        -- EG debug data output
        datar       : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        resetout    : OUT std_logic_vector((nEG-1) downto 0);
        irq         : OUT std_logic
  );
end COMPONENT;

SIGNAL ACEGconfid  : std_logic_vector(Cwid downto 0);
SIGNAL ACEGdbgdata : std_logic_vector((P-1) downto 0);
SIGNAL ACstate     : std_logic_vector((nAC-1) downto 0);
SIGNAL ACdbginfo   : std_logic_vector(((dbgbuswidth*nAC)-1) downto 0);
SIGNAL ACreset     : std_logic_vector((nAC-1) downto 0);
SIGNAL EGstate     : std_logic_vector((nEG-1) downto 0);
SIGNAL EGdbginfo   : std_logic_vector(((dbgbuswidth*nEG)-1) downto 0);
SIGNAL EGreset     : std_logic_vector((nEG-1) downto 0);
SIGNAL ACEGramdata : std_logic_vector((dbgbuswidth-1) downto 0);
SIGNAL ACEGcfgswap : std_logic;
SIGNAL ACEGcfgnum  : std_logic_vector(Cwid downto 0);
SIGNAL ACvalid     : std_logic_vector((nAC-1) downto 0);
SIGNAL EGvalid     : std_logic_vector((nEG-1) downto 0);

-- DUT Debug signals
SIGNAL DUTCStateV  : std_logic_vector(2 downto 0);
SIGNAL DUTAld      : std_logic;
SIGNAL DUTDld      : std_logic;
SIGNAL DUTDst      : std_logic;
SIGNAL DUTALE      : std_logic;
SIGNAL DUTDout     : std_logic_vector(3 downto 0);
SIGNAL DUTAddr     : std_logic_vector(4 downto 0);
SIGNAL DUTWem      : std_logic;
SIGNAL DUTRem      : std_logic;
SIGNAL DUTIOW      : std_logic;
SIGNAL DUTIOR      : std_logic;
SIGNAL DUTDin      : std_logic_vector(3 downto 0);

SIGNAL DUTLstDin   : std_logic_vector(3 downto 0);
SIGNAL DUTreg2addr : std_logic_vector(3 downto 0);
SIGNAL DUTaddr2reg : std_logic_vector(3 downto 0);
SIGNAL DUTLstad2rg : std_logic_vector(3 downto 0);

-- Assertions expressions signals
SIGNAL CurS0       : boolean;
SIGNAL LstS0       : boolean;
SIGNAL CurS1       : boolean;
SIGNAL LstS1       : boolean;
SIGNAL CurS2       : boolean;
SIGNAL LstS2       : boolean;
SIGNAL CurS3       : boolean;
```

ANNEXE E : Sources ACF

```
  SIGNAL LstS3        : boolean;
  SIGNAL CurS3a       : boolean;
  SIGNAL LstS3a       : boolean;
  SIGNAL CurS4        : boolean;
  SIGNAL LstS4        : boolean;
  SIGNAL CurS4a       : boolean;
  SIGNAL LstS4a       : boolean;
  SIGNAL CurS5        : boolean;
  SIGNAL CurS0ALE0    : boolean;
  SIGNAL LstS0ALE0    : boolean;
  SIGNAL CurS0ALE1    : boolean;
  SIGNAL LstS0ALE1    : boolean;
  SIGNAL CurS2We0Re0  : boolean;
  SIGNAL LstS2We0Re0  : boolean;
  SIGNAL CurS2We1     : boolean;
  SIGNAL LstS2We1     : boolean;
  SIGNAL CurS2We0Re1  : boolean;
  SIGNAL LstS2We0Re1  : boolean;
  SIGNAL CurS3IOW1    : boolean;
  SIGNAL LstS3IOW1    : boolean;
  SIGNAL CurS3IOW0    : boolean;
  SIGNAL LstS3IOW0    : boolean;
  SIGNAL CurS4IOR1    : boolean;
  SIGNAL LstS4IOR1    : boolean;
  SIGNAL CurS4IOR0    : boolean;
  SIGNAL LstS4IOR0    : boolean;
  SIGNAL CurAld0      : boolean;
  SIGNAL CurAld1      : boolean;
  SIGNAL CurDld0      : boolean;
  SIGNAL CurDld1      : boolean;
  SIGNAL CurDst0      : boolean;
  SIGNAL CurDst1      : boolean;
  SIGNAL CurS2Adr10   : boolean;
  SIGNAL CurS2Adr11   : boolean;
  SIGNAL CurS2Adr00   : boolean;
  SIGNAL CurWem1Rem0  : boolean;
  SIGNAL CurWem0Rem1  : boolean;
  SIGNAL CurWem0Rem0  : boolean;
  SIGNAL CurDinReg2   : boolean;
  SIGNAL CurAdd2Dout  : boolean;

  -- Event generators expressions signals
  SIGNAL EvtACStateM : boolean;
  SIGNAL EvtACAlDlDs : boolean;
  SIGNAL EvtACAdDat  : boolean;
  SIGNAL EvtACFail   : boolean;

BEGIN

  -- Debug signals assignations
  DUTCStateV  <= PDebugData(15 downto 13);
  DUTAld      <= PDebugData(12);
  DUTDld      <= PDebugData(11);
  DUTDst      <= PDebugData(10);
  DUTALE      <= PDebugData(9) WHEN not (PConfID = "01") ELSE
                 '0';
  DUTDout     <= PDebugData(8 downto 5) WHEN (PConfID = "00") ELSE
                 PDebugData(3 downto 0) WHEN (PConfID = "11") ELSE
                 "0000";
  DUTAddr     <= PDebugData(4 downto 0) WHEN (PConfID(1) = '0') ELSE
                 "00000";
  DUTWem      <= PDebugData(9) WHEN (PConfID = "01") ELSE
                 '0';
  DUTRem      <= PDebugData(8) WHEN (PConfID = "01") ELSE
                 '0';
  DUTIOW      <= PDebugData(7) WHEN (PConfID = "01") ELSE
                 PDebugData(8) WHEN (PConfID = "10") ELSE
                 '1';
  DUTIOR      <= PDebugData(6) WHEN (PConfID = "01") ELSE
                 PDebugData(8) WHEN (PConfID = "11") ELSE
```

ANNEXE E : Sources ACF

```
                '1';
DUTDin      <= PDebugData(7 downto 4) WHEN (PConfID = "10") ELSE
                "0000";
DUTreg2addr <= PDebugData(3 downto 0) WHEN (PConfID = "10") ELSE
                "0000";
DUTaddr2reg <= PDebugData(7 downto 4) WHEN (PConfID = "11") ELSE
                "0000";

-- Assertions test expressions assignations
CurS0       <= TRUE WHEN DUTCStateV = "000" ELSE
                FALSE;
CurS1       <= TRUE WHEN DUTCStateV = "001" ELSE
                FALSE;
CurS2       <= TRUE WHEN DUTCStateV = "010" ELSE
                FALSE;
CurS3       <= TRUE WHEN DUTCStateV = "011" ELSE
                FALSE;
CurS3a      <= TRUE WHEN DUTCStateV = "100" ELSE
                FALSE;
CurS4       <= TRUE WHEN DUTCStateV = "101" ELSE
                FALSE;
CurS4a      <= TRUE WHEN DUTCStateV = "110" ELSE
                FALSE;
CurS5       <= TRUE WHEN DUTCStateV = "111" ELSE
                FALSE;
CurS0ALE0   <= TRUE WHEN ((PConfID = "00") OR (PConfID = "10") OR
                          (PConfID = "11")) AND
                          (DUTCStateV = "000") AND (DUTALE = '0') ELSE
                FALSE;
CurS0ALE1   <= TRUE WHEN ((PConfID = "00") OR (PConfID = "10") OR
                          (PConfID = "11")) AND
                          (DUTCStateV = "000") AND (DUTALE = '1') ELSE
                FALSE;
CurS2We0Re0 <= TRUE WHEN (PConfID = "01") AND
                          (DUTCStateV = "010") AND (DUTWem = '0') AND
                          (DUTRem = '0') ELSE
                FALSE;
CurS2We1    <= TRUE WHEN (PConfID = "01") AND
                          (DUTCStateV = "010") AND (DUTWem = '1') ELSE
                FALSE;
CurS2We0Re1 <= TRUE WHEN (PConfID = "01") AND
                          (DUTCStateV = "010") AND (DUTWem = '0') AND
                          (DUTRem = '1') ELSE
                FALSE;
CurS3IOW1   <= TRUE WHEN ((PConfID = "01") OR (PConfID = "10")) AND
                          (DUTCStateV = "011") AND (DUTIOW = '1') ELSE
                FALSE;
CurS3IOW0   <= TRUE WHEN ((PConfID = "01") OR (PConfID = "10")) AND
                          (DUTCStateV = "011") AND (DUTIOW = '0') ELSE
                FALSE;
CurS4IOR1   <= TRUE WHEN ((PConfID = "01") OR (PConfID = "11")) AND
                          (DUTCStateV = "101") AND (DUTIOR = '1') ELSE
                FALSE;
CurS4IOR0   <= TRUE WHEN ((PConfID = "01") OR (PConfID = "11")) AND
                          (DUTCStateV = "101") AND (DUTIOR = '0') ELSE
                FALSE;
CurAld0     <= TRUE WHEN DUTAld = '0' ELSE
                FALSE;
CurAld1     <= TRUE WHEN DUTAld = '1' ELSE
                FALSE;
CurDld0     <= TRUE WHEN DUTDld = '0' ELSE
                FALSE;
CurDld1     <= TRUE WHEN DUTDld = '1' ELSE
                FALSE;
CurDst0     <= TRUE WHEN DUTDst = '0' ELSE

                FALSE;
CurDst1     <= TRUE WHEN DUTDst = '1' ELSE
                FALSE;
CurS2Adr10  <= TRUE WHEN ((PConfID = "00") OR (PConfID = "01")) AND
```

ANNEXE E : Sources ACF

```
                        (DUTCStateV = "010") AND (DUTAddr = "10100") ELSE
                    FALSE;
CurS2Adr11  <= TRUE WHEN ((PConfID = "00") OR (PConfID = "01")) AND
                        (DUTCStateV = "010") AND (DUTAddr = "10101") ELSE
                    FALSE;
CurS2Adr00  <= TRUE WHEN ((PConfID = "00") OR (PConfID = "01")) AND
                        (DUTCStateV = "010") AND
                        (NOT(DUTAddr(4 downto 1) = "1010")) ELSE
                    FALSE;
CurWem1Rem0 <= TRUE WHEN (PConfID = "01") AND
                        (DUTWem = '1') AND (DUTRem = '0') ELSE
                    FALSE;
CurWem0Rem1 <= TRUE WHEN (PConfID = "01") AND
                        (DUTWem = '0') AND (DUTRem = '1') ELSE
                    FALSE;
CurWem0Rem0 <= TRUE WHEN (PConfID = "01") AND
                        (DUTWem = '0') AND (DUTRem = '0') ELSE
                    FALSE;
CurDinReg2  <= TRUE WHEN DUTLstDin = DUTreg2addr ELSE
                    FALSE;
CurAdd2Dout <= TRUE WHEN DUTLstad2rg = DUTDout ELSE
                    FALSE;


-- Assertions expressions pipelining
ASSERTEXPPIPE: PROCESS (clk, reset_n)
BEGIN
  IF reset_n = '0' THEN
    LstS0       <= FALSE;
    LstS1       <= FALSE;
    LstS2       <= FALSE;
    LstS3       <= FALSE;
    LstS3a      <= FALSE;
    LstS4       <= FALSE;
    LstS4a      <= FALSE;
    LstS0ALE0   <= FALSE;
    LstS0ALE1   <= FALSE;
    LstS2We0Re0 <= FALSE;
    LstS2We1    <= FALSE;
    LstS2We0Re1 <= FALSE;
    LstS3IOW1   <= FALSE;
    LstS3IOW0   <= FALSE;
    LstS4IOR1   <= FALSE;
    LstS4IOR0   <= FALSE;
    DUTLstDin   <= "0000";
    DUTLstad2rg <= "0000";

  ELSIF clk'event AND clk = '1' THEN
    LstS0       <= CurS0;
    LstS1       <= CurS1;
    LstS2       <= CurS2;
    LstS3       <= CurS3;
    LstS3a      <= CurS3a;
    LstS4       <= CurS4;
    LstS4a      <= CurS4a;
    LstS0ALE0   <= CurS0ALE0;
    LstS0ALE1   <= CurS0ALE1;
    LstS2We0Re0 <= CurS2We0Re0;
    LstS2We1    <= CurS2We1;
    LstS2We0Re1 <= CurS2We0Re1;
    LstS3IOW1   <= CurS3IOW1;
    LstS3IOW0   <= CurS3IOW0;
    LstS4IOR1   <= CurS4IOR1;
    LstS4IOR0   <= CurS4IOR0;
    IF (CurS3a = TRUE) THEN
      DUTLstDin   <= DUTDin;
    END IF;
    IF (CurS4a = TRUE) THEN
      DUTLstad2rg <= DUTaddr2reg;
    END IF;
```

ANNEXE E : Sources ACF

```
  END IF;
END PROCESS;


-- Event generators expressions assignations
EvtACStateM <= TRUE WHEN ACstate(12 downto 0) = "1111111111111" ELSE
               FALSE;
EvtACAlDlDs <= TRUE WHEN ACstate(33 downto 13) = "11111111111111111111" ELSE
               FALSE;
EvtACAdDat  <= TRUE WHEN ACstate(38 downto 34) = "11111" ELSE
               FALSE;
EvtACFail   <= TRUE WHEN (EvtACStateM = TRUE) AND
                         (EvtACAlDlDs = TRUE) AND
                         (EvtACAdDat = TRUE) ELSE
               FALSE;



ACEGconfid  <= PConfId;
ACEGdbgdata <= PDebugData;
ARAMdbgdata <= ACEGramdata;
ARAMegevent <= EGstate;

ACEGramdata((P+Cwid) downto P) <= PConfId;

ACEGramdata((P-1) downto 0) <= PDebugData;
ACEGramdata(31 downto (32-nEG)) <= EGstate;
ACEGramdata((31-nEG) downto (P+Cwid+1)) <= (others => '0');


-- Assertion checkers instanciations
ACS0ALE0S0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS0ALE0,
                consequent_expr=>CurS0,
                valide=>ACvalid(0));

ACS0ALE1S1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS0ALE1,
                consequent_expr=>CurS1,
                valide=>ACvalid(1));

ACS1S2: assert_implication
      generic map(0)

      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS1,
                consequent_expr=>CurS2,
                valide=>ACvalid(2));

ACS2WEM0REM0S2: assert_implication
      generic map(0)

      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS2We0Re0,
                consequent_expr=>CurS2,
                valide=>ACvalid(3));

ACS2WEM1S3: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS2We1,
```

ANNEXE E : Sources ACF

```
                consequent_expr=>CurS3,
                valide=>ACvalid(4));

ACS2REM1S4: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS2We0Re1,
                consequent_expr=>CurS4,
                valide=>ACvalid(5));

ACS3IOW1S3: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS3IOW1,
                consequent_expr=>CurS3,
                valide=>ACvalid(6));

ACS3IOW0S3A: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS3IOW0,
                consequent_expr=>CurS3a,
                valide=>ACvalid(7));

ACS3AS0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS3a,
                consequent_expr=>CurS0,
                valide=>ACvalid(8));

ACS4IOR1S4: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS4IOR1,
                consequent_expr=>CurS4,
                valide=>ACvalid(9));

ACS4IOR0S4A: assert_implication
      generic map(0)

      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS4IOR0,
                consequent_expr=>CurS4a,
                valide=>ACvalid(10));

ACS4AS0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS4a,
                consequent_expr=>CurS0,
                valide=>ACvalid(11));

ACNOS5: assert_never
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>CurS5,
                valide=>ACvalid(12));

ACS0ALD0: assert_implication
      generic map(0)
      port map (clk=>clk,
```

ANNEXE E : Sources ACF

```
                reset_n=>reset_n,
                antecedent_expr=>CurS0,
                consequent_expr=>CurAld0,
                valide=>ACvalid(13));

ACS0DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS0,
                consequent_expr=>CurDld0,
                valide=>ACvalid(14));

ACS0DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS0,
                consequent_expr=>CurDst0,
                valide=>ACvalid(15));

ACS1ALD1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS1,
                consequent_expr=>CurAld1,
                valide=>ACvalid(16));

ACS1DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS1,
                consequent_expr=>CurDld0,
                valide=>ACvalid(17));

ACS1DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS1,
                consequent_expr=>CurDst0,
                valide=>ACvalid(18));

ACS2ALD1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2,
                consequent_expr=>CurAld1,
                valide=>ACvalid(19));

ACS2DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2,
                consequent_expr=>CurDld0,
                valide=>ACvalid(20));

ACS2DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2,
                consequent_expr=>CurDst0,
                valide=>ACvalid(21));

ACS3ALD0: assert_implication
```

ANNEXE E : Sources ACF

```
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3,
                consequent_expr=>CurAld0,
                valide=>ACvalid(22));

ACS3DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3,
                consequent_expr=>CurDld0,
                valide=>ACvalid(23));

ACS3DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3,
                consequent_expr=>CurDst0,
                valide=>ACvalid(24));

ACS3aALD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3a,
                consequent_expr=>CurAld0,
                valide=>ACvalid(25));

ACS3aDLD1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3a,
                consequent_expr=>CurDld1,
                valide=>ACvalid(26));

ACS3aDST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3a,
                consequent_expr=>CurDst0,
                valide=>ACvalid(27));

ACS4ALD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4,
                consequent_expr=>CurAld0,
                valide=>ACvalid(28));

ACS4DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4,

                consequent_expr=>CurDld0,
                valide=>ACvalid(29));

ACS4DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4,
                consequent_expr=>CurDst0,
```

ANNEXE E : Sources ACF

```
                valide=>ACvalid(30));

ACS4aALD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4a,
                consequent_expr=>CurAld0,
                valide=>ACvalid(31));

ACS4aDLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4a,
                consequent_expr=>CurDld0,
                valide=>ACvalid(32));

ACS4aDST1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4a,
                consequent_expr=>CurDst1,
                valide=>ACvalid(33));

ACS2ADDR10: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2Adr10,
                consequent_expr=>CurWem1Rem0,
                valide=>ACvalid(34));

ACS2ADDR11: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2Adr11,
                consequent_expr=>CurWem0Rem1,
                valide=>ACvalid(35));

ACS2ADDR00: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2Adr00,
                consequent_expr=>CurWem0Rem0,
                valide=>ACvalid(36));

ACS3aDINREG2ADDR: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3a,
                consequent_expr=>CurDinReg2,
                valide=>ACvalid(37));

ACS4aADDR2REGDOUT: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4a,
                consequent_expr=>CurAdd2Dout,
                valide=>ACvalid(38));


ACHARNESS: for i in (nAC-1) downto 0 generate
  HARN: ac_harness
        generic map (32,0,0)
```

ANNEXE E : Sources ACF

```
        port map(clk=>clk,
                 reset_n=>ACreset(i),
                 clrfail=>'0',
                 clrinforeg=>'0',
                 ACenable=>'1',
                 dbginput=>TTin,
                 validin=>ACvalid(i),
                 dbginfo=>ACdbginfo((((i+1)*dbgbuswidth)-1) downto (i*dbgbuswidth)),
                 failed_n=>ACstate(i)
        );
end generate;

ACCTRL: ac_ctrl
  port map (clk=>clk,
            reset_n=>reset_n,
            we_n=>ACCwe_n,
            out_addr=>ACCout_addr,
            rdat_type=>ACCdat_type,
            statein=>ACstate,
            ttin=>ACdbginfo,
            resetin=>ACCresetin,
            datar=>ACCdatar,
            resetout=>ACreset
  );


-- Event generators instanciation
EGACSTATE: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACStateM,
                valide=>EGvalid(0));

EGACALDDLDDST: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACAlDlDs,
                valide=>EGvalid(1));

EGACADDRDATA: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACAdDat,
                valide=>EGvalid(2));

EGACFAIL: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACFail,
                valide=>EGvalid(3));


EGHARNESS: for i in (nEG-1) downto 0 generate
  HARN: ac_harness
        generic map (32,0,0)
        port map(clk=>clk,
                 reset_n=>EGreset(i),
                 clrfail=>'0',
                 clrinforeg=>'0',
                 ACenable=>'1',
                 dbginput=>TTin,
                 validin=>EGvalid(i),
                 dbginfo=>EGdbginfo((((i+1)*dbgbuswidth)-1) downto (i*dbgbuswidth)),
                 failed_n=>EGstate(i)
        );
```

ANNEXE E : Sources ACF

```
end generate;

EGCTRL: eg_ctrl
  port map (clk=>clk,
            reset_n=>reset_n,
            we_n=>EGCwe_n,
            out_addr=>EGCout_addr,
            rdat_type=>EGCdat_type,
            transdest=>EGCtransdes,
            clrirq=>EGCclrirq,
            statein=>EGstate,
            ttin=>EGdbginfo,
            cfgin=>EGCcfgin,
            datar=>EGCdatar,
            resetout=>EGreset,
            irq=>EGCirq

  );


CMcfgswap    <= ACEGcfgswap;
CMswapid     <= ACEGcfgnum;


CFGSWAPCTRL: PROCESS(clk,reset_n,PDebugData)  -- Project 2, Version 1, Revision 0
BEGIN
  IF reset_n = '0' THEN
    ACEGcfgswap <= '1';
    ACEGcfgnum  <= "00";
  ELSIF clk'event AND clk = '1' THEN
    ACEGcfgswap <= '1';
    CASE ACEGcfgnum IS
      WHEN "00" => -- CFG0
        IF (DUTCStateV = "000") AND (DUTALE = '1') THEN
          ACEGcfgnum <= "01";
        ELSE
          ACEGcfgnum <= "00";
        END IF;

      WHEN "01" => -- CFG1
        IF (DUTCStateV = "010") AND (DUTWem = '1') THEN
          ACEGcfgnum <= "10";

        ELSIF (DUTCStateV = "010") AND (DUTRem = '1') THEN
          ACEGcfgnum <= "11";
        ELSE
          ACEGcfgnum <= "01";
        END IF;

      WHEN "10" => -- CFG2
        IF (DUTCStateV = "000") AND (DUTALE = '1') THEN
          ACEGcfgnum <= "01";
        ELSIF (DUTCStateV = "011") AND (DUTIOW = '0') THEN
          ACEGcfgnum <= "00";
        ELSE
          ACEGcfgnum <= "10";
        END IF;

      WHEN "11" => -- CFG3
        IF (DUTCStateV = "000") AND (DUTALE = '1') THEN
          ACEGcfgnum <= "01";
        ELSIF (DUTCStateV = "101") AND (DUTIOR = '0') THEN
          ACEGcfgnum <= "00";
        ELSE
          ACEGcfgnum <= "11";
        END IF;

      WHEN OTHERS =>
        ACEGcfgnum <= "00";
```

ANNEXE E : Sources ACF

```
      END CASE;
    END IF;
  END PROCESS;
END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : ACF general registers
--
-- File   : dataformatter.vhd
-- Author : K. Peterson
-- Created: Sep 12 2004
-- Modif. : Sep 13 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.acftypes.all;
use work.ochptypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY dataformatter IS
  PORT (clk         : IN  std_logic;
        reset_n     : IN  std_logic;

        -- ACF regs control
        we_n        : IN  std_logic;

        -- ACF general registers signals
        GRwe_n      : OUT std_logic;
        GRregdatain : OUT std_logic_vector(31 downto 0);
        GRregaddr   : OUT std_logic_vector(1 downto 0);
        GRregdataout : IN  std_logic_vector(31 downto 0);

        -- Timestamp Timer signals
        TTwe_n      : OUT std_logic;
        TTcfgin     : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        TTmaxcnt    : IN  std_logic_vector((dbgbuswidth-1) downto 0);
        TTcurtime   : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Assertion Checkers signals
        ACwe_n      : OUT std_logic;
        ACout_addr  : OUT std_logic_vector(nACwid downto 0);
        ACrdat_type : OUT std_logic;
        ACresetin   : OUT std_logic_vector(31 downto 0);
        ACdatar     : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Event Generator signals
        EGwe_n      : OUT std_logic;
        EGout_addr  : OUT std_logic_vector(nEGwid downto 0);
        EGrdat_type : OUT std_logic;
        EGtransdest : OUT std_logic;
        EGcfgin     : OUT std_logic_vector(31 downto 0);
        EGdatar     : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Configuration Manager signals
        CMwe_n      : OUT std_logic;
        CMcfgdatain : OUT std_logic_vector(31 downto 0);

        CMcfgaddr   : OUT std_logic_vector(Cwid downto 0);

        -- Assertion RAM signals
        ARwe_n      : OUT std_logic;
        ARread_ram  : OUT std_logic;
        ARregdataout : IN  std_logic_vector(31 downto 0);
        ARregdatain : OUT std_logic_vector(31 downto 0);
        ARregaddr   : OUT std_logic_vector(3 downto 0);
        ARaramaddr  : OUT std_logic_vector(31 downto 0);

        -- ACF registers data
        regaddr     : IN  std_logic_vector(31 downto 0);
        regdatain   : IN  std_logic_vector(31 downto 0);
```

ANNEXE E : Sources ACF

```
        regdataout   : OUT std_logic_vector(31 downto 0)
  );

END dataformatter;

ARCHITECTURE rtl OF dataformatter IS

  CONSTANT cACFIDCODE     : std_logic_vector(31 downto 0) := X"00000000";
  CONSTANT cACFSTATUS     : std_logic_vector(31 downto 0) := X"00000001";
  CONSTANT cACFCONTROL    : std_logic_vector(31 downto 0) := X"00000010";
  CONSTANT cTTMAXCOUNT    : std_logic_vector(31 downto 0) := X"00000020";
  CONSTANT cTTCURTIME     : std_logic_vector(31 downto 0) := X"00000021";
  CONSTANT cACSTATE       : std_logic_vector(31 downto 0) := X"00001000";
  CONSTANT cACTTCOUNTER   : std_logic_vector(31 downto 0) := X"00001800";
  CONSTANT cACRESET       : std_logic_vector(31 downto 0) := X"00000011";
  CONSTANT cEGIRQEVENT    : std_logic_vector(31 downto 0) := X"00000035";
  CONSTANT cEGSTATE       : std_logic_vector(31 downto 0) := X"00002000";
  CONSTANT cEGTTCOUNTER   : std_logic_vector(31 downto 0) := X"00002800";
  CONSTANT cEGRESET       : std_logic_vector(31 downto 0) := X"00000012";
  CONSTANT cCFGRAM        : std_logic_vector(31 downto 0) := X"00000100";
  CONSTANT cRAMTRIGTYPE   : std_logic_vector(31 downto 0) := X"00000030";
  CONSTANT cRAMTRIGEVT    : std_logic_vector(31 downto 0) := X"00000031";
  CONSTANT cRAMTRIGDELAY  : std_logic_vector(31 downto 0) := X"00000032";
  CONSTANT cRAMSTORSIZE   : std_logic_vector(31 downto 0) := X"00000033";
  CONSTANT cRAMSTRENDEVT  : std_logic_vector(31 downto 0) := X"00000034";
  CONSTANT cRAMTRIGTSTMP  : std_logic_vector(31 downto 0) := X"00000040";
  CONSTANT cRAMENDPTR     : std_logic_vector(31 downto 0) := X"00000041";
  CONSTANT cRAMENDTSTMP   : std_logic_vector(31 downto 0) := X"00000042";
  CONSTANT cASSERTRAM     : std_logic_vector(31 downto 0) := X"10000000";

  CONSTANT cREGDATZEROS   : std_logic_vector(31 downto 0) := X"00000000";
  CONSTANT cGRADDRZEROS   : std_logic_vector(1 downto 0)  := "00";
  CONSTANT cDBGINFZEROS   : std_logic_vector((dbgbuswidth-1) downto 0) :=
                            (OTHERS => '0');
  CONSTANT cNACWIDZEROS   : std_logic_vector(nACwid downto 0) :=
                            (OTHERS => '0');
  CONSTANT cNEGWIDZEROS   : std_logic_vector(nEGwid downto 0) :=
                            (OTHERS => '0');
  CONSTANT cCWIDZEROS     : std_logic_vector(Cwid downto 0) :=
                            (OTHERS => '0');
  CONSTANT cARADDRZEROS   : std_logic_vector(3 downto 0) := "0000";

BEGIN

  reg_mgr: PROCESS (clk, reset_n)
  BEGIN

    IF (reset_n = '0') THEN
      regdataout   <= cREGDATZEROS;
      GRwe_n       <= '1';
      GRregdatain  <= cREGDATZEROS;
      GRregaddr    <= cGRADDRZEROS;
      TTwe_n       <= '1';
      TTcfgin      <= cDBGINFZEROS;
      ACwe_n       <= '1';
      ACout_addr   <= cNACWIDZEROS;
      ACrdat_type  <= '0';
      ACresetin    <= cREGDATZEROS;
      EGwe_n       <= '1';
      EGout_addr   <= cNEGWIDZEROS;
      EGrdat_type  <= '0';
      EGtransdest  <= '0';
      EGcfgin      <= cREGDATZEROS;
      CMwe_n       <= '1';
      CMcfgdatain  <= cREGDATZEROS;
      CMcfgaddr    <= cCWIDZEROS;
      ARwe_n       <= '1';
      ARread_ram   <= '0';
      ARregdatain  <= cREGDATZEROS;
      ARregaddr    <= cARADDRZEROS;
```

ANNEXE E : Sources ACF

```
  ARaramaddr    <= cREGDATZEROS;

ELSIF clk'event AND clk = '1' THEN
  GRwe_n        <= '1';
  GRregdatain   <= cREGDATZEROS;
  GRregaddr     <= cGRADDRZEROS;
  TTwe_n        <= '1';
  TTcfgin       <= cDBGINFZEROS;
  ACwe_n        <= '1';
  ACout_addr    <= cNACWIDZEROS;
  ACrdat_type   <= '0';
  ACresetin     <= cREGDATZEROS;
  EGwe_n        <= '1';
  EGout_addr    <= cNEGWIDZEROS;
  EGrdat_type   <= '0';
  EGtransdest   <= '0';
  EGcfgin       <= cREGDATZEROS;
  CMwe_n        <= '1';
  CMcfgdatain   <= cREGDATZEROS;
  CMcfgaddr     <= cCWIDZEROS;
  ARwe_n        <= '1';
  ARread_ram    <= '0';
  ARregdatain   <= cREGDATZEROS;
  ARregaddr     <= cARADDRZEROS;
  ARaramaddr    <= cREGDATZEROS;

  -- ACF General Registers
  IF (regaddr = cACFIDCODE) OR
     (regaddr = cACFSTATUS) OR
     (regaddr = cACFCONTROL) THEN
    GRwe_n        <= we_n;
    GRregdatain   <= regdatain;
    GRregaddr     <= regaddr(4) & regaddr(0);
    regdataout    <= GRregdataout;

  -- Timestamp Timer Registers
  ELSIF regaddr = cTTMAXCOUNT   THEN
    TTwe_n        <= we_n;
    TTcfgin       <= regdatain;
    regdataout    <= TTmaxcnt;
  ELSIF regaddr = cTTCURTIME    THEN
    regdataout    <= TTcurtime;

  -- Assertion Checkers Registers
  ELSIF (regaddr >= cACSTATE) AND
        (regaddr < (cACSTATE + CONV_STD_LOGIC_VECTOR(nAC,32)))   THEN
    ACout_addr    <= regaddr(nACwid downto 0);
    ACrdat_type   <= '0';
    regdataout    <= ACdatar;
  ELSIF (regaddr >= cACTTCOUNTER) AND
        (regaddr < (cACTTCOUNTER + CONV_STD_LOGIC_VECTOR(nAC,32)))   THEN
    ACout_addr    <= regaddr(nACwid downto 0);
    ACrdat_type   <= '1';
    regdataout    <= ACdatar;
  ELSIF regaddr = cACRESET      THEN
    ACwe_n        <= we_n;
    ACresetin     <= regdatain;

  -- Event Generators Registers
  ELSIF regaddr = cEGIRQEVENT   THEN
    EGwe_n        <= we_n;
    EGtransdest   <= '1';
    EGcfgin       <= regdatain;
    regdataout    <= EGdatar;
  ELSIF (regaddr >= cEGSTATE) AND
        (regaddr < (cEGSTATE + CONV_STD_LOGIC_VECTOR(nEG,32)))   THEN
    EGout_addr    <= regaddr(nEGwid downto 0);
    EGrdat_type   <= '0';
    EGtransdest   <= '0';
    regdataout    <= EGdatar;
```

ANNEXE E : Sources ACF

```
      ELSIF (regaddr >= cEGTTCOUNTER) AND
            (regaddr < (cEGTTCOUNTER + CONV_STD_LOGIC_VECTOR(nEG,32)))   THEN
        EGout_addr   <= regaddr(nEGwid downto 0);
        EGrdat_type  <= '1';
        EGtransdest  <= '0';
        regdataout   <= EGdatar;
      ELSIF regaddr = cEGRESET      THEN
        EGwe_n       <= we_n;
        EGtransdest  <= '0';
        EGcfgin      <= regdatain;

      -- Configuration RAM Registers
      ELSIF (regaddr >= cCFGRAM) AND
            (regaddr < (cCFGRAM + CONV_STD_LOGIC_VECTOR(C,32)))   THEN
        CMwe_n       <= we_n;
        CMcfgdatain  <= regdatain;
        CMcfgaddr    <= regaddr(Cwid downto 0);

      -- Assertion RAM Registers
      ELSIF (regaddr = cRAMTRIGTYPE) OR
            (regaddr = cRAMTRIGEVT) OR
            (regaddr = cRAMTRIGDELAY) OR
            (regaddr = cRAMSTORSIZE) OR
            (regaddr = cRAMSTRENDEVT) OR
            (regaddr = cRAMTRIGTSTMP) OR
            (regaddr = cRAMENDPTR) OR
            (regaddr = cRAMENDTSTMP) THEN
        ARwe_n       <= we_n;
        regdataout   <= ARregdataout;
        ARregdatain  <= regdatain;
        ARregaddr    <= regaddr(6) & regaddr(2 downto 0);

      -- Assertion RAM
      ELSIF (regaddr >= cASSERTRAM) THEN
        ARread_ram   <= '1';
        regdataout   <= ARregdataout;
        ARaramaddr   <= regaddr;
      ELSE
      END IF;
    END IF;

  END PROCESS;

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Configuration Manager control logic
--
-- File   : cm_ctrl.vhd
-- Author : K. Peterson
-- Created: Jul 26 2004
-- Modif. : Sep 21 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY cm_ctrl IS
  PORT (clk        : IN  std_logic;
        reset_n    : IN  std_logic;

        -- CM control signals
        we_n       : IN  std_logic;
        eg_cfgswap : IN  std_logic;
        dump_cfg   : IN  std_logic;

        -- CM cfg status
        cfg_stored : OUT std_logic;
        dumpingcfg : OUT std_logic;
        cfg_dumped : OUT std_logic;

        -- CM cfg data input
        cfgswapid  : IN  std_logic_vector(Cwid downto 0);
        cfgdatain  : IN  std_logic_vector(31 downto 0);
        cfgaddr    : IN  std_logic_vector(Cwid downto 0);

        -- Configuration RAM I/O
        ramwe_n    : OUT std_logic;
        ramoe_n    : OUT std_logic;
        ramdatain  : IN  std_logic_vector(Nwid downto 0);
        ramdataout : OUT std_logic_vector(Nwid downto 0);
        ramaddr    : OUT std_logic_vector((Pwid+Cwid+1) downto 0);

        -- I/O Port output signals
        FSoP       : OUT   std_logic;
        FEoP       : OUT   std_logic;
        FCfgData   : OUT   std_logic_vector(Cwid downto 0)
  );

END cm_ctrl;

ARCHITECTURE rtl OF cm_ctrl IS

  CONSTANT READTIME     : integer                          := 2;
  CONSTANT CNFSTOZERO   : std_logic_vector((C-1) downto 0) := (others => '0');
  CONSTANT CFGZERO      : std_logic_vector(Cwid downto 0)  := (others => '0');
  CONSTANT CFGMAX       : std_logic_vector(Cwid downto 0)  :=
                          CONV_STD_LOGIC_VECTOR(C-1,Cwid+1);
  CONSTANT PINMAX       : std_logic_vector(Pwid downto 0)  :=
                          CONV_STD_LOGIC_VECTOR(P-1,Pwid+1);
  CONSTANT PINZERO      : std_logic_vector(Pwid downto 0)  := (others => '0');
  CONSTANT SIGMAX       : std_logic_vector(Nwid downto 0)  :=
                          CONV_STD_LOGIC_VECTOR(Nwid,Nwid+1);
  CONSTANT SIGZERO      : std_logic_vector(Nwid downto 0)  := (others => '0');

  TYPE    t_cfg_state  IS (READY,RDCFGADDR,RDCFGDATA,PKTSTART,
                           RDNXTPIN,NXTPINDATA,NXTPINSTART,PKTWRD,PKTEND);

  SIGNAL  CnfStored     : std_logic_vector((C-1) downto 0);
```

ANNEXE E : Sources ACF

```
  SIGNAL  cfg_state     : t_cfg_state;
  SIGNAL  CurCfg        : std_logic_vector(Cwid downto 0);
  SIGNAL  CurCfgPin     : std_logic_vector(Pwid downto 0);
  SIGNAL  CurCfgSig     : std_logic_vector(Nwid downto 0);
  SIGNAL  CurCfgData    : std_logic_vector(Nwid downto 0);

  SIGNAL  Intramwe_n    : std_logic;
  SIGNAL  Intramoe_n    : std_logic;
  SIGNAL  Intramdata    : std_logic_vector(Nwid downto 0);
  SIGNAL  Intramaddr    : std_logic_vector((Pwid+Cwid+1) downto 0);

  SIGNAL  Intcfgdumped  : std_logic;

  SIGNAL  IntCfgData    : std_logic_vector(Cwid downto 0);

BEGIN

  cfg_stored <= '0' WHEN (CnfStored = CNFSTOZERO) ELSE
                '1';

  dumpingcfg <= '0' WHEN (cfg_state = READY) ELSE
                '1';

  cfg_dumped <= Intcfgdumped;

  ramwe_n    <= Intramwe_n;
  ramoe_n    <= Intramoe_n;
  ramdataout <= Intramdata;
  ramaddr    <= Intramaddr;

  FCfgData <= cfgswapid WHEN (eg_cfgswap = '1') AND
                             (cfg_state = READY) ELSE
              IntCfgData;

  FSoP     <= '1'       WHEN ((eg_cfgswap = '1') AND
                             (cfg_state = READY)) OR
                             (cfg_state = PKTSTART) ELSE
              '0';

  FEoP     <= '1'       WHEN ((eg_cfgswap = '1') AND
                             (cfg_state = READY)) OR
                             (cfg_state = PKTEND) ELSE
              '0';


  cfg_mgr: PROCESS (clk, reset_n)
  BEGIN

    IF (reset_n = '0') THEN
      Intramwe_n   <= '1';
      Intramoe_n   <= '1';
      Intramdata   <= cfgdatain(Nwid downto 0);
      Intramaddr   <= cfgaddr &
                      cfgdatain(cfgdatain'high downto (cfgdatain'high-Pwid));
      Intcfgdumped <= '0';
      CnfStored    <= CNFSTOZERO;
      cfg_state    <= READY;
      CurCfg       <= CFGMAX;
      CurCfgPin    <= PINMAX;
      CurCfgSig    <= SIGMAX;
      CurCfgData   <= SIGZERO;
      IntCfgData   <= CFGZERO;

    ELSIF clk'event AND clk = '1' THEN

      Intramwe_n   <= '1';
      Intramoe_n   <= '1';
      IntCfgData   <= CFGZERO;

      CASE cfg_state IS
```

ANNEXE E : Sources ACF

```
WHEN READY =>
  IF (dump_cfg = '1') THEN
    CurCfg     <= CFGMAX;
    CurCfgPin  <= PINMAX;
    CurCfgSig  <= SIGMAX;
    cfg_state  <= RDCFGADDR;
  ELSIF (we_n = '0') THEN
    Intramwe_n <= '0';
    Intramdata <= cfgdatain(Nwid downto 0);
    Intramaddr <= cfgaddr &
                  cfgdatain(cfgdatain'high downto (cfgdatain'high-Pwid));
    CnfStored(CONV_INTEGER(cfgaddr)) <= '1';
    cfg_state  <= READY;
  ELSE
    cfg_state  <= READY;
  END IF;

WHEN RDCFGADDR =>
  IF CnfStored(CONV_INTEGER(CurCfg)) = '0' THEN
    IF CurCfg = CFGZERO THEN
      cfg_state <= READY;
      Intcfgdumped <= '1';
    ELSE
      CurCfg    <= CurCfg - 1;
      cfg_state <= RDCFGADDR;
    END IF;
  ELSE
    Intramoe_n <= '0';
    Intramaddr <= CurCfg &
                  CurCfgPin;
    cfg_state <= RDCFGDATA;
  END IF;

WHEN RDCFGDATA =>
  Intramoe_n <= '0';
  IntCfgData <= CurCfg;
  cfg_state <= PKTSTART;

WHEN PKTSTART =>
  CurCfgData <= ramdatain((Nwid-1) downto 0) & '0';
  IntCfgData(0) <= ramdatain(Nwid);
  CurCfgSig  <= CurCfgSig - 1;
  cfg_state <= PKTWRD;

WHEN PKTWRD =>
  IF CONV_INTEGER(CurCfgSig) > READTIME THEN
    CurCfgSig  <= CurCfgSig - 1;
    cfg_state <= PKTWRD;
  ELSE
    IF (CurCfgPin = PINZERO) THEN
      IF (CurCfgSig = SIGZERO) THEN
        cfg_state <= PKTEND;
      ELSE
        CurCfgSig  <= CurCfgSig - 1;
        cfg_state <= PKTWRD;
      END IF;
    ELSE
      CurCfgPin <= CurCfgPin - 1;
      cfg_state <= RDNXTPIN;
    END IF;
  END IF;
  IntCfgData(0) <= CurCfgData(Nwid);
  CurCfgData <= CurCfgData((Nwid-1) downto 0) & '0';

WHEN RDNXTPIN =>
  Intramoe_n <= '0';
  Intramaddr <= CurCfg &
                CurCfgPin;
  IntCfgData(0) <= CurCfgData(Nwid);
```

ANNEXE E : Sources ACF

```
          CurCfgData <= CurCfgData((Nwid-1) downto 0) & '0';
          cfg_state <= NXTPINDATA;

        WHEN NXTPINDATA =>
          Intramoe_n <= '0';
          IntCfgData(0) <= CurCfgData(Nwid);
          CurCfgSig  <= SIGMAX;
          cfg_state <= NXTPINSTART;

        WHEN NXTPINSTART =>
          CurCfgData <= ramdatain((Nwid-1) downto 0) & '0';
          IntCfgData(0) <= ramdatain(Nwid);
          CurCfgSig  <= CurCfgSig - 1;
          cfg_state <= PKTWRD;

        WHEN PKTEND =>
          IF CurCfg = CFGZERO THEN
            cfg_state <= READY;
            Intcfgdumped <= '1';
          ELSE
            CurCfg    <= CurCfg - 1;
            cfg_state <= RDCFGADDR;
            CurCfgPin <= PINMAX;
            CurCfgSig <= SIGMAX;
          END IF;

        WHEN OTHERS =>
          cfg_state <= READY;
      END CASE;

    END IF;

  END PROCESS;

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Configuration Manager control logic
--
-- File   : cm_ram.vhd
-- Author : K. Peterson
-- Created: Sep 22 2004
-- Modif. : Sep 24 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY cm_ram IS
  PORT (clk        : IN  std_logic;

        -- Configuration RAM I/O
        ramwe_n    : IN  std_logic;
        ramoe_n    : IN  std_logic;
        ramdatain  : IN  std_logic_vector(Nwid downto 0);
        ramdataout : OUT std_logic_vector(Nwid downto 0);
        ramaddr    : IN  std_logic_vector((Pwid+Cwid+1) downto 0)
  );

END cm_ram;

ARCHITECTURE rtl OF cm_ram IS

  TYPE    mem_type     IS array (((2**(Pwid+Cwid+2))-1) downto 0) of std_logic_vector(Nwid
downto 0);

  SIGNAL  mem           : mem_type;

BEGIN

  ramdataout <= mem(CONV_INTEGER(ramaddr));

  memory: PROCESS (clk, ramwe_n,ramaddr)
  BEGIN

    IF clk'event AND clk = '1' THEN

      IF ramwe_n = '0' THEN
        mem(CONV_INTEGER(ramaddr)) <= ramdatain;

      END IF;

    END IF;

  END PROCESS;

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Assertion RAM control logic
--
-- File   : assertram_ctrl.vhd
-- Author : K. Peterson
-- Created: Jul 30 2004
-- Modif. : Sep 21 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
use work.acftypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY assertram_ctrl IS
  PORT (clk         : IN  std_logic;
        reset_n     : IN  std_logic;

        -- AssertRAM control signals
        we_n        : IN  std_logic;
        read_ram    : IN  std_logic;
        arm_trigger : IN  std_logic;
        force_trig  : IN  std_logic;
        stop_stor   : IN  std_logic;

        -- AssertRAM status
        trig_armed  : OUT std_logic;
        storing_dat : OUT std_logic;
        regdataout  : OUT std_logic_vector(31 downto 0);


        -- AssertRAM reg data input
        regdatain   : IN  std_logic_vector(31 downto 0);
        regaddr     : IN  std_logic_vector(3 downto 0);
        aramaddr    : IN  std_logic_vector(31 downto 0);

        -- AssertRAM dbg data input
        dbgdata_in  : IN  std_logic_vector((dbgbuswidth-1) downto 0);
        egevent_in  : IN  std_logic_vector((nEG-1) downto 0);
        ttin        : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Assertion RAM I/O
        ramwe_n     : OUT std_logic;
        ramoe_n     : OUT std_logic;
        ramdatain   : IN  std_logic_vector((dbgbuswidth-1) downto 0);
        ramdataout  : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        ramaddr     : OUT std_logic_vector(assramwid downto 0)
  );

END assertram_ctrl;

ARCHITECTURE rtl OF assertram_ctrl IS

  CONSTANT cRAMTRIGTYPE  : std_logic_vector(regaddr'high downto 0) := "0000";
  CONSTANT cRAMTRIGEVT   : std_logic_vector(regaddr'high downto 0) := "0001";
  CONSTANT cRAMTRIGDELAY : std_logic_vector(regaddr'high downto 0) := "0010";
  CONSTANT cRAMSTORSIZE  : std_logic_vector(regaddr'high downto 0) := "0011";
  CONSTANT cRAMSTRENDEVT : std_logic_vector(regaddr'high downto 0) := "0100";
  CONSTANT cRAMTRIGTSTMP : std_logic_vector(regaddr'high downto 0) := "1000";
  CONSTANT cRAMENDPTR    : std_logic_vector(regaddr'high downto 0) := "1001";
  CONSTANT cRAMENDTSTMP  : std_logic_vector(regaddr'high downto 0) := "1010";

  CONSTANT cSINGLESHOT   : integer                                 := 0;
  CONSTANT cSTORUNTILEVT : integer                                 := 1;
  CONSTANT cMANUALTRIG   : integer                                 := 2;
```

ANNEXE E : Sources ACF

```
  CONSTANT cTRIGDEFAULT  : std_logic_vector(2 downto 0)               :=
                           (others => '0');
  CONSTANT cEVENTZERO    : std_logic_vector(nEGwid downto 0)          :=
                           (others => '0');
  CONSTANT cSTORZERO     : std_logic_vector(assramwid downto 0)       :=
                           (others => '0');
  CONSTANT cTTZERO       : std_logic_vector((dbgbuswidth-1) downto 0) :=
                           (others => '0');
  CONSTANT cDATAZERO     : std_logic_vector(31 downto 0) := X"00000000";

  SIGNAL  RamTrigType    : std_logic_vector(2 downto 0);
  SIGNAL  RamTrigEvent   : std_logic_vector(nEGwid downto 0);
  SIGNAL  RamTrigDelay   : std_logic_vector(assramwid downto 0);
  SIGNAL  RamStorSize    : std_logic_vector(assramwid downto 0);
  SIGNAL  RamStorEndEvt  : std_logic_vector(nEGwid downto 0);
  SIGNAL  RamTrigTStamp  : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL  RamEndPointer  : std_logic_vector(assramwid downto 0);
  SIGNAL  RamEndTStamp   : std_logic_vector((dbgbuswidth-1) downto 0);

  SIGNAL  RamAddrCountr  : std_logic_vector(assramwid downto 0);
  SIGNAL  RamSizeCountr  : std_logic_vector(assramwid downto 0);

  SIGNAL  TrigArmed      : std_logic;
  SIGNAL  StoringData    : std_logic;

  SIGNAL  Intramwe_n     : std_logic;
  SIGNAL  Intramoe_n     : std_logic;
  SIGNAL  Intramaddr     : std_logic_vector(assramwid downto 0);

  SIGNAL  IntCntrZero    : std_logic;
  SIGNAL  IntTrigger     : std_logic;
  SIGNAL  IntStopStor    : std_logic;
  SIGNAL  IntReadRAM     : std_logic;

BEGIN

  trig_armed  <= TrigArmed;

  storing_dat <= StoringData;

  ramwe_n     <= Intramwe_n;
  ramoe_n     <= Intramoe_n;
  ramaddr     <= Intramaddr;

  regdataout  <= ramdatain
                   WHEN (IntReadRAM = '1') ELSE
                 cDATAZERO(31 downto (RamTrigType'high+1)) & RamTrigType
                   WHEN (regaddr = cRAMTRIGTYPE) ELSE
                 cDATAZERO(31 downto (RamTrigEvent'high+1)) & RamTrigEvent
                   WHEN (regaddr = cRAMTRIGEVT) ELSE
                 cDATAZERO(31 downto (RamTrigDelay'high+1)) & RamTrigDelay
                   WHEN (regaddr = cRAMTRIGDELAY) ELSE
                 cDATAZERO(31 downto (RamStorSize'high+1)) & RamStorSize
                   WHEN (regaddr = cRAMSTORSIZE) ELSE
                 cDATAZERO(31 downto (RamStorEndEvt'high+1)) & RamStorEndEvt
                   WHEN (regaddr = cRAMSTRENDEVT) ELSE
                 RamTrigTStamp
                   WHEN (regaddr = cRAMTRIGTSTMP) ELSE
                 cDATAZERO(31 downto (RamEndPointer'high+1)) & RamEndPointer
                   WHEN (regaddr = cRAMENDPTR) ELSE
                 RamEndTStamp
                   WHEN (regaddr = cRAMENDTSTMP) ELSE
                 X"00000000";

  IntCntrZero <= '1' WHEN (RamSizeCountr = cSTORZERO) ELSE
                 '0';

  IntTrigger  <= TrigArmed AND (force_trig OR (
                                NOT (RamTrigType(cMANUALTRIG) OR
```

ANNEXE E : Sources ACF

```
                                    egevent_in(CONV_INTEGER(RamTrigEvent)))));

IntStopStor <= StoringData AND (stop_stor OR (
                                    RamTrigType(cSTORUNTILEVT) AND
                                    NOT egevent_in(CONV_INTEGER(RamStorEndEvt)))
                                    OR IntCntrZero);


reg_mgr: PROCESS (clk, reset_n)
BEGIN

  IF (reset_n = '0') THEN
    RamTrigType   <= cTRIGDEFAULT;
    RamTrigEvent  <= cEVENTZERO;
    RamTrigDelay  <= cSTORZERO;
    RamStorSize   <= cSTORZERO;
    RamStorEndEvt <= cEVENTZERO;
    ramdataout  <= cTTZERO;

  ELSIF clk'event AND clk = '1' THEN

    ramdataout  <= dbgdata_in;

    IF we_n = '0' THEN
      CASE regaddr IS
        WHEN cRAMTRIGTYPE  =>
          RamTrigType   <= regdatain(RamTrigType'high downto 0);
        WHEN cRAMTRIGEVT   =>
          RamTrigEvent  <= regdatain(RamTrigEvent'high downto 0);
        WHEN cRAMTRIGDELAY =>
          RamTrigDelay  <= regdatain(RamTrigDelay'high downto 0);
        WHEN cRAMSTORSIZE  =>
          RamStorSize   <= regdatain(RamStorSize'high downto 0);
        WHEN cRAMSTRENDEVT =>
          RamStorEndEvt <= regdatain(RamStorEndEvt'high downto 0);
        WHEN others        =>
          NULL;
      END CASE;
    END IF;

  END IF;

END PROCESS;

assertram_mgr: PROCESS (clk, reset_n)

BEGIN

  IF (reset_n = '0') THEN
    RamTrigTStamp <= cTTZERO;
    RamEndPointer <= cSTORZERO;
    RamEndTStamp  <= cTTZERO;
    RamAddrCountr <= cSTORZERO;
    RamSizeCountr <= cSTORZERO;
    TrigArmed     <= '0';
    StoringData   <= '0';
    Intramwe_n    <= '1';
    Intramoe_n    <= '1';
    Intramaddr    <= cSTORZERO;
    IntReadRAM    <= '0';

  ELSIF clk'event AND clk = '1' THEN

    IntReadRAM    <= '0';
    Intramwe_n    <= '1';
    Intramoe_n    <= '1';
    Intramaddr    <= RamAddrCountr;

    IF arm_trigger = '1' THEN
```

ANNEXE E : Sources ACF

```
        TrigArmed   <= '1';
        RamSizeCountr <= RamStorSize;
      END IF;

      IF read_ram = '1' THEN
        IF StoringData = '1' THEN
          RamEndPointer <= RamAddrCountr;
          RamEndTStamp  <= ttin;
        END IF;
        TrigArmed   <= '0';
        StoringData <= '0';
        IntReadRAM  <= '1';
        Intramwe_n  <= '1';
        Intramoe_n  <= '0';
        Intramaddr  <= aramaddr(Intramaddr'high downto 0);
        RamSizeCountr <= RamStorSize;

      ELSIF IntTrigger = '1' THEN
        StoringData <= '1';
        Intramwe_n  <= '0';
        RamTrigTStamp <= ttin;

        IF RamTrigType(cSINGLESHOT) = '1' THEN
          TrigArmed     <= '0';
          RamSizeCountr <= RamStorSize - RamTrigDelay;
          IF RamAddrCountr < (RamStorSize - 1) THEN
            RamAddrCountr <= RamAddrCountr + 1;
          ELSE
            RamAddrCountr <= cSTORZERO;
          END IF;
        ELSE
          RamAddrCountr <= cSTORZERO;
          RamSizeCountr <= RamStorSize;
        END IF;

      ELSIF IntStopStor = '1' THEN
        StoringData <= '0';
        Intramwe_n  <= '1';
        RamEndPointer <= RamAddrCountr;
        RamEndTStamp  <= ttin;
        RamSizeCountr <= RamStorSize;

      ELSIF StoringData = '1' THEN
        StoringData <= '1';
        Intramwe_n  <= '0';
        IF RamAddrCountr < (RamStorSize - 1) THEN
          RamAddrCountr <= RamAddrCountr + 1;
        ELSE
          RamAddrCountr <= cSTORZERO;
        END IF;

        IF NOT (RamTrigType(cSTORUNTILEVT) = '1') THEN
          RamSizeCountr <= RamSizeCountr - 1;
        END IF;

      ELSIF (TrigArmed = '1') AND (RamTrigType(cSINGLESHOT) = '1') THEN
        Intramwe_n  <= '0';
        IF RamAddrCountr < (RamStorSize - 1) THEN
          RamAddrCountr <= RamAddrCountr + 1;
        ELSE
          RamAddrCountr <= cSTORZERO;
        END IF;
      END IF;

    END IF;

  END PROCESS;

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : ACF general registers
--
-- File   : acfgenregs.vhd
-- Author : K. Peterson
-- Created: Sep 9 2004
-- Modif. : Sep 9 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.acftypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY acfgenregs IS
  PORT (clk         : IN  std_logic;
        reset_n     : IN  std_logic;

        -- ACF regs control
        we_n        : IN  std_logic;

        -- ACF regs status signals
        trig_armed  : IN  std_logic;
        storing_dat : IN  std_logic;
        cfg_stored  : IN  std_logic;
        dumping_cfg : IN  std_logic;
        cfg_dumped  : IN  std_logic;
        irq_set     : IN  std_logic;

        -- ACF regs ext. control signals
        arm_trigger : OUT std_logic;
        force_trig  : OUT std_logic;
        stop_stor   : OUT std_logic;
        clr_irq     : OUT std_logic;
        reset_tt    : OUT std_logic;
        reset_cm    : OUT std_logic;
        dump_cfg    : OUT std_logic;
        soft_reset  : OUT std_logic;

        -- ACF regs reg data input
        regdatain   : IN  std_logic_vector(31 downto 0);
        regaddr     : IN  std_logic_vector(1 downto 0);

        -- ACF regs data output
        regdataout  : OUT std_logic_vector(31 downto 0)
  );

END acfgenregs;

ARCHITECTURE rtl OF acfgenregs IS

  CONSTANT cIDCODEADDR   : std_logic_vector(regaddr'high downto 0) := "00";
  CONSTANT cSTATUSADDR   : std_logic_vector(regaddr'high downto 0) := "01";
  CONSTANT cCONTROLADDR  : std_logic_vector(regaddr'high downto 0) := "10";

  CONSTANT cTRIGARMED    : integer := 0;
  CONSTANT cSTORINGDAT   : integer := 1;
  CONSTANT cCFGSTORED    : integer := 2;
  CONSTANT cDUMPINGCFG   : integer := 3;
  CONSTANT cCFGDUMPED    : integer := 4;
  CONSTANT cIRQSET       : integer := 5;

  CONSTANT cARMTRIGGER   : integer := 0;
  CONSTANT cFORCETRIG    : integer := 1;
  CONSTANT cSTOPSTOR     : integer := 2;
  CONSTANT cCLRIRQ       : integer := 3;
```

ANNEXE E : Sources ACF

```
  CONSTANT cRESETTT       : integer := 4;
  CONSTANT cRESETCM       : integer := 5;
  CONSTANT cDUMPCFG       : integer := 6;
  CONSTANT cSOFTRESET     : integer := 31;

  CONSTANT cZEROS         : std_logic_vector(31 downto 0) := X"00000000";

  SIGNAL   IDCODE         : std_logic_vector(31 downto 0);
  SIGNAL   StatusVector   : std_logic_vector(31 downto 0);

BEGIN

  IDCODE(31 downto 16)       <= CONV_STD_LOGIC_VECTOR(ACFProject,16);
  IDCODE(15 downto 8)        <= CONV_STD_LOGIC_VECTOR(ACFVersion,8);
  IDCODE(7 downto 0)         <= CONV_STD_LOGIC_VECTOR(ACFRevision,8);

  StatusVector(31 downto 6) <= "00000000000000000000000000";
  StatusVector(cTRIGARMED)  <= trig_armed;
  StatusVector(cSTORINGDAT) <= storing_dat;
  StatusVector(cCFGSTORED)  <= cfg_stored;
  StatusVector(cDUMPINGCFG) <= dumping_cfg;
  StatusVector(cCFGDUMPED)  <= cfg_dumped;
  StatusVector(cIRQSET)     <= irq_set;

  regdataout <= IDCODE       WHEN (regaddr = cIDCODEADDR) ELSE
                StatusVector WHEN (regaddr = cSTATUSADDR) ELSE
                cZEROS;

  reg_mgr: PROCESS (clk, reset_n)
  BEGIN

    IF (reset_n = '0') THEN
      arm_trigger <= '0';
      force_trig  <= '0';
      stop_stor   <= '0';
      clr_irq     <= '0';
      reset_tt    <= '0';
      reset_cm    <= '0';
      dump_cfg    <= '0';
      soft_reset  <= '0';

    ELSIF clk'event AND clk = '1' THEN

      IF (we_n = '0') AND (regaddr = cCONTROLADDR) THEN
        arm_trigger <= regdatain(cARMTRIGGER);
        force_trig  <= regdatain(cFORCETRIG);
        stop_stor   <= regdatain(cSTOPSTOR);
        clr_irq     <= regdatain(cCLRIRQ);
        reset_tt    <= regdatain(cRESETTT);
        reset_cm    <= regdatain(cRESETCM);
        dump_cfg    <= regdatain(cDUMPCFG);
        soft_reset  <= regdatain(cSOFTRESET);
      ELSE
        arm_trigger <= '0';
        force_trig  <= '0';
        stop_stor   <= '0';
        clr_irq     <= '0';
        reset_tt    <= '0';
        reset_cm    <= '0';
        dump_cfg    <= '0';
        soft_reset  <= '0';
      END IF;

    END IF;

  END PROCESS;

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Synthesisable assertions harness
--
-- File   : ac_harness.vhd
-- Author : K. Peterson
-- Date   : Jun 29 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY ac_harness IS
  GENERIC (
    dbginfowidth   : integer   := 32;
    dbginfotype    : integer   := 0;  -- Default: Timestamp register
    failedlatched  : integer   := 0   -- Default: Failed not latched
    );
  PORT (clk        : IN  std_logic;
        reset_n    : IN  std_logic;

        -- Assertion checker control signals
        clrfail    : IN  std_logic;
        clrinforeg : IN  std_logic;
        ACenable   : IN  std_logic;

        -- Assertion checker debug data input
        dbginput   : IN  std_logic_vector((dbginfowidth-1) downto 0);
        validin    : IN  std_logic;

        -- Assertion checker debug data output
        dbginfo    : OUT std_logic_vector((dbginfowidth-1) downto 0);
        failed_n   : OUT std_logic);
END ac_harness;

ARCHITECTURE rtl OF ac_harness IS
  CONSTANT DBGINFO_TIMESTAMP : integer    := 0; -- Timestamp register
  CONSTANT DBGINFO_FAILCNTR  : integer    := 1; -- Fail counter
  CONSTANT FAILED_NOTLATCHED : integer    := 0;
  CONSTANT FAILED_LATCHED    : integer    := 1;

  SIGNAL   failreg           : std_logic;
  SIGNAL   dbginforeg        : std_logic_vector((dbginfowidth-1) downto 0);
BEGIN

  dbginfo  <= dbginforeg;

  failed_n <= failreg  WHEN failedlatched = FAILED_LATCHED ELSE
              validin  WHEN ACenable = '1'                 ELSE
              '1';

  assertion_failed : PROCESS (clk, reset_n, clrfail)
  BEGIN

    IF (reset_n = '0') OR (clrfail = '1') THEN

      failreg <= '1';

    ELSIF clk'event AND clk = '1' THEN

      IF ACenable = '1' THEN
        IF validin = '0' THEN
          failreg <= '0';
        END IF;
      END IF;
```

ANNEXE E : Sources ACF

```
    END IF;

  END PROCESS;


  assertion_dbginfo : PROCESS (clk, reset_n, clrinforeg)
  BEGIN

    IF (reset_n = '0') OR (clrinforeg = '1') THEN

      FOR i IN (dbginfowidth-1) downto 0 LOOP
        dbginforeg(i) <= '0';
      END LOOP;

    ELSIF clk'event AND clk = '1' THEN

      IF ACenable = '1' THEN
        IF dbginfotype = DBGINFO_TIMESTAMP THEN
          IF failedlatched = FAILED_LATCHED THEN
            IF (failreg = '1') AND (validin = '0') THEN
              dbginforeg <= dbginput;
            END IF;
          ELSE
            IF validin = '0' THEN
              dbginforeg <= dbginput;
            END IF;
          END IF;
        ELSIF dbginfotype = DBGINFO_FAILCNTR THEN
          IF validin = '0' THEN
            dbginforeg <= dbginforeg + 1;
          END IF;
        END IF;
      END IF;

    END IF;

  END PROCESS;

END rtl;
```

ANNEXE E : Sources ACF

```
--*- mode: VHDL; vhdl-basic-offset: 2; vhdl-upper-case-keywords: t; indent-tabs-mode: nil
-*-
--
-- Editorial notes:
-- The top line of this file is used to set the GNU Emacs vhdl-mode to
--   a) Use indentation of 2
--   b) Use upper case keywords.
--   c) Avoid using the tab character.

-- ------------------------------------------------------------------------
--
--        ASSERT_ALWAYS
--
-- ------------------------------------------------------------------------
--  NAME
--           ASSERT_ALWAYS - An invariant concurrent assertion to ensure
--                           that its argument always evaluates TRUE
--
-- ------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.Numeric_Std.ALL;

--LIBRARY accellera;
USE work.ovl_assert.ALL;

ENTITY assert_always IS
  GENERIC (
-- pragcomment translate_off
--    severity_lvl : severity_level := FAILURE;
-- pragcomment translate_on
    options      : integer        := 0
-- pragcomment translate_off
--    ;
--    msg          : string         := "ASSERT ALWAYS VIOLATION"
-- pragcomment translate_on
    );
  PORT (clk, reset_n : IN std_ulogic;
        test_expr    : IN boolean;
        valide       : OUT std_ulogic);
END assert_always;

ARCHITECTURE ovl OF assert_always IS
-- pragcomment translate_off
  SIGNAL valid : std_ulogic := '1';
  SIGNAL rst_n : std_ulogic;
-- pragcomment translate_on
BEGIN
-- pragcomment translate_off
--  ASSERT valid = '1' REPORT msg SEVERITY severity_lvl;

  rst_n <= --ovl_reset_n WHEN ovl_reset_n_enable ELSE
           reset_n;

  valide <= valid;

  PROCESS
  BEGIN
    WAIT UNTIL clk'EVENT AND clk = '1';
    valid <= '1';
    IF (rst_n = '1') THEN
      IF (test_expr = FALSE) THEN
        valid <= '0';
      ELSE
        valid <= '1';
      END IF;
    ELSE
      valid <= '1';
    END IF;
```

ANNEXE E : Sources ACF

```
  END PROCESS;
-- pragcomment translate_on
END ovl;  -- assert_always
```

ANNEXE E : Sources ACF

```
--*- mode: VHDL; vhdl-basic-offset: 2; vhdl-upper-case-keywords: t; indent-tabs-mode: nil
-*-
--
-- Editorial notes:
-- The top line of this file is used to set the GNU Emacs vhdl-mode to
--   a) Use indentation of 2
--   b) Use upper case keywords.
--   c) Avoid using the tab character.

------------------------------------------------------------------------------
--
--       ASSERT_IMPLICATION
--
------------------------------------------------------------------------------
-- NAME
--          ASSERT_IMPLICATION - An invariant concurrent assertion to ensure
--              that a logical implication always evaluates TRUE.
--
-- USAGE
--          assert_implication [#(severity_level, options, msg)]
--              inst_name (clk, reset_n, antecedent_expr, consequent_expr);
--
------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.Numeric_Std.ALL;

--LIBRARY accellera;
USE work.ovl_assert.ALL;

ENTITY assert_implication IS
  GENERIC (
-- pragcomment translate_off
--    severity_lvl : severity_level := FAILURE;
-- pragcomment translate_on
    options      : integer        := 0
-- pragcomment translate_off
--    ;
--    msg          : string         := "ASSERT IMPLICATION VIOLATION"
-- pragcomment translate_on
    );
  PORT (clk, reset_n                      : IN std_ulogic;
        antecedent_expr, consequent_expr : IN boolean;
        valide                            : OUT std_ulogic);
END assert_implication;
--
ARCHITECTURE ovl OF assert_implication IS
-- pragcomment translate_off
  SIGNAL valid : std_ulogic := '1';
  SIGNAL rst_n : std_ulogic;
-- pragcomment translate_on
BEGIN
-- pragcomment translate_off
--
  rst_n <= --ovl_reset_n WHEN ovl_reset_n_enable ELSE
           reset_n;

  valide <= '1' WHEN (valid = '1') ELSE
            '0';
--
  PROCESS
  BEGIN
    WAIT UNTIL clk'EVENT AND clk = '1';
    valid <= '1';
    IF (rst_n = '1') THEN
      IF NOT ((antecedent_expr = FALSE) OR (consequent_expr = TRUE)) THEN
        valid <= '0';
--        ASSERT FALSE REPORT msg SEVERITY severity_lvl;
      END IF;
```

ANNEXE E : Sources ACF

```
    END IF;
  END PROCESS;
-- pragcomment translate_on
END ovl;  -- assert_implication
```

ANNEXE E : Sources ACF

```
--*- mode: VHDL; vhdl-basic-offset: 2; vhdl-upper-case-keywords: t; indent-tabs-mode: nil
-*-
--
-- Editorial notes:
-- The top line of this file is used to set the GNU Emacs vhdl-mode to
--   a) Use indentation of 2
--   b) Use upper case keywords.
--   c) Avoid using the tab character.

-- ----------------------------------------------------------------------------
--
--        ASSERT_NEVER
--
-- ----------------------------------------------------------------------------
--  NAME
--           ASSERT_NEVER - An invariant concurrent assertion to ensure
--                          that its argument never evaluates TRUE
--
-- ----------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.Numeric_Std.ALL;

--LIBRARY accellera;
USE work.ovl_assert.ALL;

ENTITY assert_never IS
  GENERIC (
-- pragcomment translate_off
--    severity_lvl : severity_level := FAILURE;
-- pragcomment translate_on
    options      : integer        := 0
-- pragcomment translate_off
--    ;
--    msg          : string         := "ASSERT NEVER VIOLATION"
-- pragcomment translate_on
    );
  PORT (clk, reset_n : IN std_ulogic;
        test_expr    : IN boolean;
        valide       : OUT std_ulogic);
END assert_never;

ARCHITECTURE ovl OF assert_never IS
-- pragcomment translate_off
  SIGNAL valid : std_ulogic := '1';
  SIGNAL rst_n : std_ulogic;
-- pragcomment translate_on
BEGIN
-- pragcomment translate_off
--
--  ASSERT valid = '1' REPORT msg SEVERITY severity_lvl;
--
  rst_n <= --ovl_reset_n WHEN ovl_reset_n_enable ELSE
           reset_n;

  valide <= valid;
--
  PROCESS
  BEGIN
    WAIT UNTIL clk'EVENT AND clk = '1';
    valid <= '1';
    IF (rst_n = '1') THEN
      IF (test_expr = TRUE) THEN
        valid <= '0';
      END IF;
    ELSE
      valid <= '1';
    END IF;
  END PROCESS;
-- pragcomment translate_on
```

ANNEXE E : Sources ACF

```
END ovl;  -- assert_never
```

ANNEXE E : Sources ACF

```
--*- mode: VHDL; vhdl-basic-offset: 2; vhdl-upper-case-keywords: t; indent-tabs-mode: nil
-*-
--
-- Editorial notes:
-- The top line of this file is used to set the GNU Emacs vhdl-mode to
--   a) Use indentation of 2
--   b) Use upper case keywords.
--   c) Avoid using the tab character.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.Numeric_Std.ALL;

PACKAGE ovl_assert IS

-- pragma translate_off
--  SIGNAL ovl_reset_n                  : std_ulogic := '1';
--  SIGNAL ovl_reset_n_enable           : boolean    := FALSE;
--  SIGNAL ovl_end_of_simulation_signal : std_ulogic := '0';
-- pragma translate_on

  TYPE OVL_BOOLEAN_VECTOR IS ARRAY (integer RANGE <>) OF boolean;

--  COMPONENT assert_always
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT ALWAYS VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN boolean);
--  END COMPONENT;
--
--
--  COMPONENT assert_always_on_edge
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      edge_type    : integer        := 0;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT ALWAYS ON EDGE VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n              : IN std_ulogic;
--      sampling_event, test_expr : IN boolean);
--  END COMPONENT;
--
--
--  COMPONENT assert_change
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 1;
--      num_cks      : integer        := 1;
--      flag         : integer        := 0;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT CHANGE VIOLATION"
```

ANNEXE E : Sources ACF

```
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      start_event  : IN boolean;
--      test_expr    : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_cycle_sequence
--    GENERIC (
---- pragma translate_off
--      severity_lvl        : severity_level := FAILURE;
---- pragma translate_on
--      num_cks             : integer        := 1;
--      necessary_condition : integer        := 0;
--      options             : integer        := 0
---- pragma translate_off
--      ;
--      msg                 : string         := "ASSERT CYCLE SEQUENCE VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n   : IN std_ulogic;
--      event_sequence : IN OVL_BOOLEAN_VECTOR((num_cks-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_decrement
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 1;
--      value        : integer        := 1;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT DECREMENT VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_delta
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 1;
--      min          : integer        := 1;
--      max          : integer        := 1;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT DELTA VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_even_parity
--    GENERIC (
---- pragma translate_off
```

ANNEXE E : Sources ACF

```
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 1;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT EVEN PARITY VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_fifo_index
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      depth        : integer        := 1;
--      push_width   : integer        := 1;
--      pop_width    : integer        := 1;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT FIFO INDEX VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      push         : IN unsigned((push_width-1) DOWNTO 0);
--      pop          : IN unsigned((pop_width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_frame
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      min_cks      : integer        := 0;
--      max_cks      : integer        := 0;
--      flag         : integer        := 0;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT FRAME VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n           : IN std_ulogic;
--      start_event, test_expr : IN boolean);
--  END COMPONENT;
--
--
--  COMPONENT assert_handshake
--    GENERIC (
---- pragma translate_off
--      severity_lvl   : severity_level := FAILURE;
---- pragma translate_on
--      min_ack_cycle  : integer        := 0;
--      max_ack_cycle  : integer        := 0;
--      req_drop       : integer        := 0;
--      deassert_count : integer        := 0;
--      max_ack_length : integer        := 0;
--      options        : integer        := 0
---- pragma translate_off
--      ;
--      msg            : string         := "ASSERT HANDSHAKE VIOLATION"
```

ANNEXE E : Sources ACF

```
---- pragma translate_on
--       );
--     PORT (
--       clk, reset_n : IN std_ulogic;
--       req, ack     : IN boolean);
--   END COMPONENT;
--
--
--   COMPONENT assert_implication
--     GENERIC (
---- pragma translate_off
--       severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--       options      : integer        := 0
---- pragma translate_off
--       ;
--       msg          : string         := "ASSERT IMPLICATION VIOLATION"
---- pragma translate_on
--       );
--     PORT (
--       clk, reset_n                       : IN std_ulogic;
--       antecedent_expr, consequent_expr : IN boolean);
--   END COMPONENT;
--
--
--   COMPONENT assert_increment
--     GENERIC (
---- pragma translate_off
--       severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--       width        : integer        := 1;
--       value        : integer        := 1;
--       options      : integer        := 0
---- pragma translate_off
--       ;
--       msg          : string         := "ASSERT INCREMENT VIOLATION"
---- pragma translate_on
--       );
--     PORT (
--       clk, reset_n : IN std_ulogic;
--       test_expr    : IN unsigned((width-1) DOWNTO 0));
--   END COMPONENT;
--
--
--   COMPONENT assert_never
--     GENERIC (
---- pragma translate_off
--       severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--       options      : integer        := 0
---- pragma translate_off
--       ;
--       msg          : string         := "ASSERT NEVER VIOLATION"
---- pragma translate_on
--       );
--     PORT (
--       clk, reset_n : IN std_ulogic;
--       test_expr    : IN boolean);
--   END COMPONENT;
--
--
--   COMPONENT assert_next
--     GENERIC (
---- pragma translate_off
--       severity_lvl      : severity_level := FAILURE;
---- pragma translate_on
--       num_cks           : integer        := 1;
--       check_overlapping : integer        := 1;
--       only_if           : integer        := 0;
--       options           : integer        := 0
```

ANNEXE E : Sources ACF

```
---- pragma translate_off
--       ;
--       msg                : string         := "ASSERT NEXT VIOLATION"
---- pragma translate_on
--       );
--     PORT (
--       clk, reset_n : IN std_ulogic;
--       start_event  : IN boolean;
--       test_expr    : IN boolean);
--   END COMPONENT;
--
--
--   COMPONENT assert_no_overflow
--     GENERIC (
---- pragma translate_off
--       severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--       width        : integer        := 1;
--       min          : integer        := 0;
--       max          : integer        := -1;
--       options      : integer        := 0
---- pragma translate_off
--       ;
--       msg          : string         := "ASSERT NO OVERFLOW VIOLATION"
---- pragma translate_on
--       );
--     PORT (
--       clk, reset_n : IN std_ulogic;
--       test_expr    : IN unsigned((width-1) DOWNTO 0));
--   END COMPONENT;
--
--
--   COMPONENT assert_no_transition
--     GENERIC (
---- pragma translate_off
--       severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--       width        : integer        := 1;
--       options      : integer        := 0
---- pragma translate_off
--       ;
--       msg          : string         := "ASSERT NO TRANSITION VIOLATION"
---- pragma translate_on
--       );
--     PORT (
--       clk, reset_n : IN std_ulogic;
--       test_expr    : IN unsigned((width-1) DOWNTO 0);
--       start_state  : IN unsigned((width-1) DOWNTO 0);
--       next_state   : IN unsigned((width-1) DOWNTO 0));
--   END COMPONENT;
--
--
--   COMPONENT assert_no_underflow
--     GENERIC (
---- pragma translate_off
--       severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--       width        : integer        := 1;
--       min          : integer        := 0;
--       max          : integer        := -1;
--       options      : integer        := 0
---- pragma translate_off
--       ;
--       msg          : string         := "ASSERT NO UNDERFLOW VIOLATION"
---- pragma translate_on
--       );
--     PORT (
--       clk, reset_n : IN std_ulogic;
--       test_expr    : IN unsigned((width-1) DOWNTO 0));
--   END COMPONENT;
```

ANNEXE E : Sources ACF

```
--
--  COMPONENT assert_odd_parity
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 1;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT ODD PARITY VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_one_cold
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 32;
--      inactive     : integer        := 2;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT ONE COLD VIOLATION"
---- pragma translate_on
--      );
--
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_one_hot
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 32;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT ONE HOT VIOLATION"
---- pragma translate_on
--      );
--
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_proposition
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT PROPOSITION VIOLATION"
---- pragma translate_on
--      );
--    PORT (
```

ANNEXE E : Sources ACF

```
--      reset_n    : IN std_ulogic;
--      test_expr  : IN boolean);
--  END COMPONENT;
--
--
--  COMPONENT assert_quiescent_state
--    GENERIC (
---- pragma translate_off
--      severity_lvl             : severity_level := FAILURE;
---- pragma translate_on
--      width                    : integer        := 1;
--      options                  : integer        := 0;
---- pragma translate_off
--      msg                      : string         := "ASSERT QUIESCENT_STATE VIOLATION";
---- pragma translate_on
--      ASSERT_END_OF_SIMULATION : integer        := 0
--      );
--    PORT (
--      clk, reset_n             : IN std_ulogic;
--      state_expr, check_value  : IN unsigned((width-1) DOWNTO 0);
--      sample_event             : IN boolean);
--  END COMPONENT;
--
--
--  COMPONENT assert_range
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 1;
--      min          : integer        := 1;
--      max          : integer        := -1;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT RANGE VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_time
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      num_cks      : integer        := 1;
--      flag         : integer        := 0;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT TIME VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      start_event  : IN boolean;
--      test_expr    : IN boolean);
--  END COMPONENT;
--
--
--  COMPONENT assert_transition
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 1;
```

ANNEXE E : Sources ACF

```
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT TRANSITION VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN unsigned((width-1) DOWNTO 0);
--      start_state  : IN unsigned((width-1) DOWNTO 0);
--      next_state   : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_unchange
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 1;
--      num_cks      : integer        := 1;
--      flag         : integer        := 0;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT UNCHANGE VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      start_event  : IN boolean;
--      test_expr    : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;
--
--
--  COMPONENT assert_width
--    GENERIC (
---- pragma translate_off
--      severity_lvl     : severity_level := FAILURE;
---- pragma translate_on
--      min_cks, max_cks : integer        := 1;
--      options          : integer        := 0
---- pragma translate_off
--      ;
--      msg              : string         := "ASSERT WIDTH VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr    : IN boolean);
--  END COMPONENT;
--
--
--  COMPONENT assert_win_change
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width        : integer        := 1;
--      options      : integer        := 0
---- pragma translate_off
--      ;
--      msg          : string         := "ASSERT WIN CHANGE VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      start_event  : IN boolean;
--      test_expr    : IN unsigned((width-1) DOWNTO 0);
```

ANNEXE E : Sources ACF

```
--      end_event     : IN boolean);

--  END COMPONENT;
--
--
--  COMPONENT assert_win_unchange
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width         : integer        := 1;
--      options       : integer        := 0
---- pragma translate_off
--      ;
--      msg           : string         := "ASSERT WIN UNCHANGE VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      start_event  : IN boolean;

--      test_expr     : IN unsigned((width-1) DOWNTO 0);
--      end_event     : IN boolean);
--  END COMPONENT;
--
--
--  COMPONENT assert_window
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      options       : integer        := 0
---- pragma translate_off
--      ;
--      msg           : string         := "ASSERT WINDOW VIOLATION"
---- pragma translate_on
--      );
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      start_event  : IN boolean;
--      test_expr     : IN boolean;
--      end_event     : IN boolean);
--  END COMPONENT;
--
--
--  COMPONENT assert_zero_one_hot
--    GENERIC (
---- pragma translate_off
--      severity_lvl : severity_level := FAILURE;
---- pragma translate_on
--      width         : integer        := 32;
--      options       : integer        := 0
---- pragma translate_off
--      ;
--      msg           : string         := "ASSERT ZERO ONE HOT VIOLATION"
---- pragma translate_on
--      );
--
--    PORT (
--      clk, reset_n : IN std_ulogic;
--      test_expr     : IN unsigned((width-1) DOWNTO 0));
--  END COMPONENT;

END ovl_assert;
```

ANNEXE E : Sources ACF

```
--*- mode: VHDL; vhdl-basic-offset: 2; vhdl-upper-case-keywords: t; indent-tabs-mode: nil
-*-
--
-- Editorial notes:
-- The top line of this file is used to set the GNU Emacs vhdl-mode to
--   a) Use indentation of 2
--   b) Use upper case keywords.
--   c) Avoid using the tab character.

-- pragcomment translate_off
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.Numeric_Std.ALL;

LIBRARY Std;
USE STD.TextIO.ALL;

--LIBRARY accellera;
USE work.ovl_assert.ALL;
-- pragcomment translate_on

PACKAGE ovl_assert_util IS
-- pragcomment translate_off
  FUNCTION xorr (V  : unsigned) RETURN std_ulogic;
--  FUNCTION has_x (A : unsigned) RETURN boolean;
--
--  -- Image functions for reporting value of test_expr etc.
--  FUNCTION Image(In_Image       : unsigned) RETURN string;
--  FUNCTION HexImage(In_Image    : unsigned) RETURN string;
--  FUNCTION HexImage(InStrg      : string) RETURN string;
--  FUNCTION Image(In_Image       : std_logic_vector) RETURN string;
--  FUNCTION Image(In_Image       : OVL_BOOLEAN_VECTOR) RETURN string;
--  FUNCTION BinHexImage(In_Image : unsigned) RETURN string;
-- pragcomment translate_on
END ovl_assert_util;


PACKAGE BODY ovl_assert_util IS
-- pragcomment translate_off
  FUNCTION xorr (V : unsigned) RETURN std_ulogic IS
    VARIABLE reduce : std_ulogic;
  BEGIN
    FOR i IN V'RANGE LOOP
      IF i = V'LEFT THEN
        reduce := V(i);
      ELSE
        reduce := reduce XOR V(i);
      END IF;
    END LOOP;
    RETURN reduce;
  END xorr;


--  FUNCTION has_x (A : unsigned) RETURN boolean IS
--  BEGIN
--    FOR ii IN A'RANGE LOOP
--      IF (is_x(A(ii))) THEN
--        RETURN(TRUE);                   -- has X
--      END IF;
--    END LOOP;
--    RETURN(FALSE);                      -- does not have X
--  END has_x;
--
--  FUNCTION Image(In_Image : unsigned) RETURN string IS
--  BEGIN
--    RETURN Image(std_logic_vector(In_Image));
--  END Image;
--
--  FUNCTION Image(In_Image : std_logic_vector) RETURN string IS
--    VARIABLE L : line;                  -- access type
```

ANNEXE E : Sources ACF

```
--    VARIABLE W : string(1 TO In_Image'LENGTH) := (OTHERS => ' ');
--  BEGIN
--    IEEE.Std_Logic_TextIO.WRITE(L, In_Image);
--    W(L.ALL'RANGE) := L.ALL;
--    Deallocate(L);
--    RETURN W;
--  END Image;
--
--  FUNCTION HexImage(InStrg : string) RETURN string IS
--    SUBTYPE  Int03_Typ IS integer RANGE 0 TO 3;
--    VARIABLE Result : string(1 TO ((InStrg'LENGTH - 1)/4)+1) :=
--      (OTHERS => '0');
--    VARIABLE StrTo4 : string(1 TO Result'LENGTH * 4) :=
--      (OTHERS => '0');
--    VARIABLE MTspace : Int03_Typ;       --  Empty space to fill in
--    VARIABLE Str4    : string(1 TO 4);
--    VARIABLE Group_v : natural := 0;
--  BEGIN
--    MTspace                              := Result'LENGTH * 4 - InStrg'LENGTH;
--    StrTo4(MTspace + 1 TO StrTo4'LENGTH) := InStrg;  -- padded with '0'
--    Cnvrt_Lbl : FOR I IN Result'RANGE LOOP
--      Group_v := Group_v + 4;  -- identifies end of bit # in a group of 4
--      Str4    := StrTo4(Group_v - 3 TO Group_v);     -- get next 4 characters
--      CASE Str4 IS
--        WHEN "0000" => Result(I) := '0';
--        WHEN "0001" => Result(I) := '1';
--        WHEN "0010" => Result(I) := '2';
--        WHEN "0011" => Result(I) := '3';
--        WHEN "0100" => Result(I) := '4';
--        WHEN "0101" => Result(I) := '5';
--        WHEN "0110" => Result(I) := '6';
--        WHEN "0111" => Result(I) := '7';
--        WHEN "1000" => Result(I) := '8';
--        WHEN "1001" => Result(I) := '9';
--        WHEN "1010" => Result(I) := 'A';
--        WHEN "1011" => Result(I) := 'B';
--        WHEN "1100" => Result(I) := 'C';
--        WHEN "1101" => Result(I) := 'D';
--        WHEN "1110" => Result(I) := 'E';
--        WHEN "1111" => Result(I) := 'F';
--        WHEN "ZZZZ" => Result(I) := 'Z';             -- added 8/23/02
--        WHEN OTHERS => Result(I) := 'X';
--      END CASE;  --  Str4
--    END LOOP Cnvrt_Lbl;
--
--    RETURN Result;
--  END HexImage;
--
--  FUNCTION HexImage(In_Image : unsigned) RETURN string IS
--  BEGIN
--    RETURN HexImage(Image(In_Image));
--  END HexImage;
--
--  FUNCTION Image(In_Image : OVL_BOOLEAN_VECTOR) RETURN string IS
--    VARIABLE slv : std_logic_vector(In_Image'RANGE);
--  BEGIN
--    FOR i IN In_Image'RANGE LOOP
--      IF In_Image(i) THEN
--        slv(i) := '1';
--      ELSE
--        slv(i) := '0';
--      END IF;
--    END LOOP;
--
--    RETURN Image(slv);
--  END Image;
--
--  -- Return both binary and hex version of unsigned
--  FUNCTION BinHexImage(In_Image : unsigned) RETURN string IS
--  BEGIN
```

ANNEXE E : Sources ACF

```
--     RETURN """" & Image(In_Image) & """=16#" & HexImage(In_Image) & "#";
--   END BinHexImage;
---- pragcomment translate_on
END ovl_assert_util;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Assertion Checkers control logic
--
-- File   : ac_ctrl.vhd
-- Author : K. Peterson
-- Date   : Jul 1 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.acftypes.all;

USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY ac_ctrl IS
  PORT (clk        : IN  std_logic;
        reset_n    : IN  std_logic;

        -- AC control signals
        we_n       : IN  std_logic;
        out_addr   : IN  std_logic_vector(nACwid downto 0);
        rdat_type  : IN  std_logic;

        -- AC debug data input
        statein    : IN  std_logic_vector((nAC-1) downto 0);
        ttin       : IN  std_logic_vector(((dbgbuswidth*nAC)-1) downto 0);
        resetin    : IN  std_logic_vector(31 downto 0);

        -- AC debug data output
        datar      : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        resetout   : OUT std_logic_vector((nAC-1) downto 0)
  );

END ac_ctrl;

ARCHITECTURE rtl OF ac_ctrl IS

  COMPONENT ac_eg_comm
    GENERIC (
      dbginfowidth   : integer;
      nMD            : integer;
      MDwid          : integer
    );
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- AC/EG control signals
          we_n       : IN  std_logic;
          out_addr   : IN  std_logic_vector(MDwid downto 0);
          rdat_type  : IN  std_logic;

          -- AC/EG debug data input
          statein    : IN  std_logic_vector((nMD-1) downto 0);
          ttin       : IN  std_logic_vector(((dbginfowidth*nMD)-1) downto 0);
          resetin    : IN  std_logic_vector(31 downto 0);

          -- AC/EG debug data output
          datar      : OUT std_logic_vector((dbginfowidth-1) downto 0);
          resetout   : OUT std_logic_vector((nMD-1) downto 0)
    );
  end COMPONENT;
BEGIN

  ACEGCOMM: ac_eg_comm
    generic map(dbgbuswidth,nAC,nACwid)
    port map (clk=>clk,
```

ANNEXE E : Sources ACF

```
                reset_n=>reset_n,
                we_n=>we_n,
                out_addr=>out_addr,
                rdat_type=>rdat_type,
                statein=>statein,

                ttin=>ttin,
                resetin=>resetin,
                datar=>datar,
                resetout=>resetout
    );

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Event Generators control logic
--
-- File   : eg_ctrl.vhd
-- Author : K. Peterson
-- Created: Jul 1 2004
-- Modif. : Jul 30 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.acftypes.all;

USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY eg_ctrl IS
  PORT (clk        : IN  std_logic;
        reset_n    : IN  std_logic;

        -- EG control signals
        we_n       : IN  std_logic;
        out_addr   : IN  std_logic_vector(nEGwid downto 0);
        rdat_type  : IN  std_logic;
        transdest  : IN  std_logic;
        clrirq     : IN  std_logic;

        -- EG debug data input
        statein    : IN  std_logic_vector((nEG-1) downto 0);
        ttin       : IN  std_logic_vector(((dbgbuswidth*nEG)-1) downto 0);
        cfgin      : IN  std_logic_vector(31 downto 0);

        -- EG debug data output
        datar      : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        resetout   : OUT std_logic_vector((nEG-1) downto 0);
        irq        : OUT std_logic
  );

END eg_ctrl;

ARCHITECTURE rtl OF eg_ctrl IS

  COMPONENT ac_eg_comm
    GENERIC (
      dbginfowidth   : integer;
      nMD            : integer;
      MDwid          : integer
    );
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- AC/EG control signals
          we_n       : IN  std_logic;
          out_addr   : IN  std_logic_vector(MDwid downto 0);
          rdat_type  : IN  std_logic;

          -- AC/EG debug data input
          statein    : IN  std_logic_vector((nMD-1) downto 0);
          ttin       : IN  std_logic_vector(((dbginfowidth*nMD)-1) downto 0);
          resetin    : IN  std_logic_vector(31 downto 0);

          -- AC/EG debug data output
          datar      : OUT std_logic_vector((dbginfowidth-1) downto 0);
          resetout   : OUT std_logic_vector((nMD-1) downto 0)
    );
  end COMPONENT;
```

ANNEXE E : Sources ACF

```
  CONSTANT TRANSDEST_COMM : std_logic := '0';
  CONSTANT TRANSDEST_IRQ  : std_logic := '1';

  SIGNAL we_ncomm     : std_logic;
  SIGNAL datarcomm    : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL irqegnum     : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL irqreg       : std_logic;

BEGIN

  we_ncomm <= we_n    WHEN transdest = TRANSDEST_COMM ELSE
              '1';

  datar <= datarcomm WHEN transdest = TRANSDEST_COMM ELSE
           irqegnum;

  irq <= irqreg       WHEN irqegnum(irqegnum'high) = '0' ELSE
         '0';

  ACEGCOMM: ac_eg_comm
    generic map(dbgbuswidth,nEG,nEGwid)
    port map (clk=>clk,
              reset_n=>reset_n,
              we_n=>we_ncomm,
              out_addr=>out_addr,
              rdat_type=>rdat_type,
              statein=>statein,
              ttin=>ttin,
              resetin=>cfgin,
              datar=>datarcomm,
              resetout=>resetout
    );

  select_read: PROCESS (clk, reset_n)
  BEGIN

    IF (reset_n = '0') THEN
      irqegnum(irqegnum'high) <= '1';
      FOR i IN (irqegnum'high-1) downto 0 LOOP
        irqegnum(i) <= '0';
      END LOOP;

    ELSIF clk'event AND clk = '1' THEN
      IF (we_n = '0') and (transdest = TRANSDEST_IRQ) THEN
        irqegnum(irqegnum'high) <= cfgin(cfgin'high);
        irqegnum(nEGwid downto 0) <= cfgin(nEGwid downto 0);
      END IF;
    END IF;

  END PROCESS;

  irq_handle: PROCESS (clk, reset_n, clrirq)
  BEGIN

    IF (reset_n = '0') OR (clrirq = '1') THEN
      irqreg <= '0';

    ELSIF clk'event AND clk = '1' THEN
      IF irqegnum(irqegnum'high) = '0' THEN
        IF statein(CONV_INTEGER(irqegnum(nEGwid downto 0))) = '0' THEN
          irqreg <= '1';
        END IF;
      END IF;
    END IF;

  END PROCESS;

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Assertion Checkers and Event Generators
--               common logic
--
-- File   : ac_eg_common.vhd
-- Author : K. Peterson
-- Created: Jul 1 2004
-- Modif. : Aug 19 2004

--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY ac_eg_comm IS
  GENERIC (
    dbginfowidth   : integer   := 32;
    nMD            : integer   := 256; -- Number of AC or EG
    MDwid          : integer   := 7    -- log2nMD-1
  );
  PORT (clk        : IN  std_logic;
        reset_n    : IN  std_logic;

        -- AC/EG control signals
        we_n       : IN  std_logic;
        out_addr   : IN  std_logic_vector(MDwid downto 0);
        rdat_type  : IN  std_logic;

        -- AC/EG debug data input
        statein    : IN  std_logic_vector((nMD-1) downto 0);
        ttin       : IN  std_logic_vector(((dbginfowidth*nMD)-1) downto 0);
        resetin    : IN  std_logic_vector(31 downto 0);

        -- AC/EG debug data output
        datar      : OUT std_logic_vector((dbginfowidth-1) downto 0);
        resetout   : OUT std_logic_vector((nMD-1) downto 0)
  );

END ac_eg_comm;

ARCHITECTURE rtl OF ac_eg_comm IS
  CONSTANT RDAT_STATEREG : std_logic  := '0'; -- State register
  CONSTANT RDAT_TTREG    : std_logic  := '1'; -- Timestamp/counter register

  TYPE     timeArray    is array ((nMD-1) downto 0) of std_logic_vector((dbginfowidth-1)
downto 0);

  SIGNAL   ttarray       : timeArray;

BEGIN


  reset_ctrl: PROCESS (clk, reset_n)
  BEGIN

    IF (reset_n = '0') THEN
      FOR i IN (nMD-1) downto 0 LOOP
        resetout(i) <= '0';
      END LOOP;

    ELSIF clk'event AND clk = '1' THEN

      IF we_n = '0' THEN
        FOR i IN (nMD-1) downto 0 LOOP
          IF resetin(resetin'high) = '1' THEN
```

ANNEXE E : Sources ACF

```
            resetout(i) <= '0';
          ELSE
            IF CONV_INTEGER(resetin(MDwid downto 0)) = i THEN
              resetout(i) <= '0';
            ELSE
              resetout(i) <= '1';
            END IF;
          END IF;
        END LOOP;
      ELSE
        FOR i IN (nMD-1) downto 0 LOOP
          resetout(i) <= '1';
        END LOOP;
      END IF;

    END IF;

  END PROCESS;

  select_read: PROCESS (out_addr,rdat_type,statein,ttin,ttarray)
  BEGIN
    FOR i in (nMD-1) downto 0 LOOP
      ttarray(i) <= ttin((((i+1)*dbginfowidth)-1) downto (i*dbginfowidth));
    END LOOP;

    IF rdat_type = RDAT_STATEREG THEN
      datar(0) <= statein(CONV_INTEGER(out_addr));
      FOR i IN datar'high downto 1 LOOP
        datar(i) <= '0';
      END LOOP;
    ELSE
      datar    <= ttarray(CONV_INTEGER(out_addr));
    END IF;
  END PROCESS;

END rtl;
```

ANNEXE E : Sources ACF

```
------------------------------------------------------------
-- Description : Assertion Checker FPGA      parameters file
--               Customized for ISA bus interface
--
-- File    : acf_param_ISA_ochp_set1.vhd
-- Author : K. Peterson
-- Created: Oct 29 2004
-- Modif. : Oct 31 2004
--
-- Proj    : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

package ochptypes is
  constant C:         integer := 4;     -- Number of configurations
  constant Cwid:      integer := 1;     -- log2C - 1
  constant N:         integer := 62;    -- Number of internal signals
  constant Nwid:      integer := 5;     -- log2N - 1
  constant P:         integer := 16;    -- Number of external pins
  constant Pwid:      integer := 3;     -- log2P - 1
end ochptypes;

package acftypes is
  constant nAC:          integer := 39;     -- Number of Assertion Checkers
  constant nACwid:       integer := 5;     -- log2nAC - 1
  constant nEG:          integer := 4;     -- Number of Event Generators
  constant nEGwid:       integer := 1;     -- log2nEG - 1
  constant dbgbuswidth:  integer := 32;
  constant assramwid:    integer := 27;

  constant ACFProject:   integer := 2; -- ISA bus interface
  constant ACFVersion:   integer := 1; -- Assertion set 1
  constant ACFRevision:  integer := 0;

end acftypes;
```

ANNEXE F : Sources Bus ISA

```
------------------------------------------------------------
-- Description : ISA bus interface validation
--
-- File   : ISA_spartan.vhd
-- Author : K. Peterson
-- Created: Oct  6 2004
-- Modif. : Oct 12 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
use work.acftypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY ISA_spartan IS
  PORT (clk          : IN  std_logic;
        reset        : IN  std_logic;

        -- LED display signals
        led_out    : out std_logic_vector(7 downto 0);

        -- Data input and switches
        sw_in      : in  std_logic_vector(7 downto 0);

        -- OCHP Interface
        -- I/O Port input signals
        PSoP         : IN  std_logic;
        PEoP         : IN  std_logic;
        PCfgData     : IN  std_logic_vector(Cwid downto 0);

        -- I/O Port output signals
        PClkOut      : OUT std_logic;
        PConfID      : OUT std_logic_vector(Cwid downto 0);
        PDebugData   : OUT std_logic_vector((P-1) downto 0)
  );

END ISA_spartan;

ARCHITECTURE rtl OF ISA_spartan IS

  COMPONENT ISA_ochp
    PORT (
        CLK_P:           in std_logic;
        RESET_P:         in std_logic;
        IOR_P:           in std_logic;
        IOW_P:           in std_logic;
        ALE_P:           in std_logic;
        ADDRESS_P:       in std_logic_vector(15 downto 0);
        DIN_P:           in std_logic_vector(7 downto 0);
        DOUT_P:          out std_logic_vector(7 downto 0);

        -- OCHP Interface
        -- I/O Port input signals
        PSoP         : IN  std_logic;
        PEoP         : IN  std_logic;
        PCfgData     : IN  std_logic_vector(Cwid downto 0);

        -- I/O Port output signals
        PClkOut      : OUT std_logic;
        PConfID      : OUT std_logic_vector(Cwid downto 0);
        PDebugData   : OUT std_logic_vector((P-1) downto 0)
    );
  END COMPONENT;

  component LFSR
```

ANNEXE F : Sources Bus ISA

```
    port (
        clk     :       in std_logic;
        reset   :       in std_logic;
        q       :       out std_logic_vector(15 downto 0)
  );
  end component;

  SIGNAL lfsrout     : std_logic_vector(15 downto 0);
  SIGNAL Intaddr     : std_logic_vector(15 downto 0);

BEGIN

  Intaddr <= X"A000" WHEN lfsrout(13) = '1' ELSE
             X"A001" WHEN lfsrout(12) = '1' ELSE
             X"0000";

  DUT: ISA_ochp
    port map (CLK_P=>clk,
              RESET_P=>reset,
              IOR_P=>lfsrout(15),
              IOW_P=>lfsrout(14),
              ALE_P=>lfsrout(7),
              ADDRESS_P=>Intaddr,
              DIN_P=>sw_in,
              DOUT_P=>led_out,
              PSoP=>PSoP,
              PEoP=>PEoP,
              PCfgData=>PCfgData,
              PClkOut=>PClkOut,
              PConfID=>PConfID,
              PDebugData=>PDebugData
    );

  EVTGEN: LFSR
    port map (clk=>clk,
              reset=>reset,
              q=>lfsrout
    );

END rtl;
```

ANNEXE F : Sources Bus ISA

```
------------------------------------------------------------
-- Description : ISA bus interface with integrated probe
--
-- File   : ISA_ochp.vhd
-- Authors: Weijun Zhang, K. Peterson
-- Created: May 2001
-- Modif. : Oct 6 2004
--
-- Proj   : Assertion-Based Runtime Debugger
------------------------------------------------------------

-----------------------------------------------------------------
-- ISA bus interface design (ISA.vhd)
-- by Weijun Zhang, 05/2001
-----------------------------------------------------------------

-- 8-bit adder ----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adder is
port(   num1:  in std_logic_vector(7 downto 0);
        num2:  in std_logic_vector(7 downto 0);
        sum:    out std_logic_vector(7 downto 0)
);
end adder;

architecture behv of adder is
begin

    sum <= num1 + num2;

end behv;

-- Comparator -----------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity comparator is
port(   addr_ld:       in std_logic;
        addr_in:       in std_logic_vector(15 downto 0);
        w_match:       out std_logic;
        r_match:       out std_logic
);
end comparator;

architecture behv of comparator is
begin

    process(addr_ld, addr_in)
    begin
        if(addr_ld='1') then
            if (addr_in="1010000000000000") then
                w_match <= '1';
                r_match <= '0';
            elsif (addr_in="1010000000000001") then
                w_match <= '0';
                r_match <= '1';
            else
                w_match <= '0';
                r_match <= '0';
            end if;
        else
            w_match <= '0';
```

ANNEXE F : Sources Bus ISA

```
            r_match <= '0';
        end if;
    end process;

end behv;

-- Data Register ------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity data_reg is
port(   clock:          in std_logic;
        reset:          in std_logic;
        load:           in std_logic;
        data_in:        in std_logic_vector(7 downto 0);
         data_out:      out std_logic_vector(7 downto 0)
);
end data_reg;

architecture behv of data_reg is
begin
    process(clock, reset, load, data_in)
        begin
         if(reset = '1') then
             data_out <= "00000000";
         elsif(clock'event and clock = '1') then
             if(load = '1') then
                 data_out <= data_in;
                        end if;
         end if;

    end process;
end behv;

-------------------------------------------------------------------------
-- Data Path of ISA bus interface
-------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity datapath is
port(   dp_clk:         in std_logic;
        dp_rst:         in std_logic;
        A_ld:           in std_logic;
        D_ld:           in std_logic;
        D_st:           in std_logic;
        ADDR_P:         in std_logic_vector(15 downto 0);
        DIN:            in std_logic_vector(7 downto 0);
        Write_match:    out std_logic;
        Read_match:     out std_logic;
        DOUT:           out std_logic_vector(7 downto 0);
         ochp_data:       out std_logic_vector(15 downto 0)
);
end datapath;

architecture struct of datapath is

component data_reg is
port(   clock:          in std_logic;
        reset:          in std_logic;
        load:           in std_logic;
        data_in:        in std_logic_vector(7 downto 0);
         data_out:      out std_logic_vector(7 downto 0)
);
```

ANNEXE F : Sources Bus ISA

```
end component;

component comparator is
port(   addr_ld:        in std_logic;
        addr_in:        in std_logic_vector(15 downto 0);
        w_match:        out std_logic;
        r_match:        out std_logic
);
end component;

component adder is
port(    num1:  in std_logic_vector(7 downto 0);
         num2:  in std_logic_vector(7 downto 0);
         sum:    out std_logic_vector(7 downto 0)
);
end component;

signal reg2adder, adder2reg, one: std_logic_vector(7 downto 0);

begin

    one <= "00000001";

    ochp_data <= reg2adder & adder2reg;

    U1: comparator port map (A_ld, ADDR_P, Write_match, Read_match);
    U2: data_reg port map (dp_clk, dp_rst, D_ld, DIN, reg2adder);
    U3: adder port map (reg2adder, one, adder2reg);
    U4: data_reg port map (dp_clk, dp_rst, D_st, adder2reg, DOUT);

end struct;

-------------------------------------------------------------------
-- FSM controller for ISA bus interfacing
-------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity controller is
port(   ctrl_clk:       in std_logic;
        ctrl_rst:       in std_logic;
        IOR:            in std_logic;
        IOW:            in std_logic;
        ALE:            in std_logic;
        WE_m:           in std_logic;
        RE_m:           in std_logic;
        Ald:            out std_logic;
        Dld:            out std_logic;
        Dst:            out std_logic;
         ochp_data:      out std_logic_vector(5 downto 0)
);
end controller;

architecture fsm of controller is

    type states is (S0, S1, S2, S3, S3a, S4, S4a, S5);
    signal nState, cState: states;

    signal nStateV, cStateV: std_logic_vector(2 downto 0);
begin

    nStateV   <= "000" when nState = S0 ELSE
                 "001" when nState = S1 ELSE
                 "010" when nState = S2 ELSE
                 "011" when nState = S3 ELSE
                 "100" when nState = S3a ELSE
                 "101" when nState = S4 ELSE
```

ANNEXE F : Sources Bus ISA

```
                "110" when nState = S4a ELSE
                "111";

cStateV   <= "000" when cState = S0 ELSE
                "001" when cState = S1 ELSE
                "010" when cState = S2 ELSE
                "011" when cState = S3 ELSE
                "100" when cState = S3a ELSE
                "101" when cState = S4 ELSE
                "110" when cState = S4a ELSE
                "111";

ochp_data <= cStateV & nStateV;

state_reg: process(ctrl_clk, ctrl_rst)
begin
    if (ctrl_rst = '1') then
        cState <= S0;
    elsif (ctrl_clk'event and ctrl_clk = '1') then
        cState <= nState;
    end if;
end process;

comb_logic: process(cState, IOR, IOW, ALE, WE_m, RE_m)
begin

  case cState is

    when S0 =>      Ald <= '0';         -- waiting for ALE signal
                    Dld <= '0';
                    Dst <= '0';
                    if (ALE='1') then
                            nState <= S1;
                    else
                            nState <= S0;
                    end if;

    when S1 =>      Ald <= '1';         -- waiting for Address
                    Dld <= '0';
                    Dst <= '0';
                    nState <= S2;

    when S2 =>      Ald <= '1';         -- decide operation
                    Dld <= '0';         -- Read or Write
                    Dst <= '0';
                    if (WE_m='1') then
                            nState <= S3;
                    elsif (RE_m='1') then
                            nState <= S4;
                    else
                            nState <= S2;
                    end if;

    when S3 =>      Ald <= '0';         -- ready to Write
                    Dld <= '0';
                    Dst <= '0';
                    if (IOW='0') then
                            nState <= S3a;
                    else
                            nState <= S3;
                    end if;

    when S3a =>     Ald <= '0';         -- do Write operation
                    Dld <= '1';         -- then go back
                    Dst <= '0';
                    nState <= S0;

    when S4 =>      Ald <= '0';         -- ready to Read
                    Dld <= '0';
                    Dst <= '0';
```

ANNEXE F : Sources Bus ISA

```
                        if (IOR='0') then
                                nState <= S4a;
                        else
                                nState <= S4;
                        end if;

        when S4a =>     Ald <= '0';         -- do Read operation
                        Dld <= '0';         -- then go back
                        Dst <= '1';
                        nState <= S0;

        when others =>  Ald <= '0';         -- go back to initial
                        Dld <= '0';
                        Dst <= '0';
                        nState <= S0;

      end case;

    end process;

end fsm;

-------------------------------------------------------------------
-- ISA bus interface ( FSM + Datapath )
-- VHDL structural modeling
-------------------------------------------------------------------

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

entity ISA_ochp is
port(   CLK_P:          in std_logic;
        RESET_P:        in std_logic;
        IOR_P:          in std_logic;
        IOW_P:          in std_logic;
        ALE_P:          in std_logic;
        ADDRESS_P:      in std_logic_vector(15 downto 0);
        DIN_P:          in std_logic_vector(7 downto 0);

        DOUT_P:         out std_logic_vector(7 downto 0);

        -- OCHP Interface
        -- I/O Port input signals
        PSoP        : IN  std_logic;
        PEoP        : IN  std_logic;
        PCfgData    : IN  std_logic_vector(Cwid downto 0);

        -- I/O Port output signals
        PClkOut     : OUT std_logic;
        PConfID     : OUT std_logic_vector(Cwid downto 0);
        PDebugData  : OUT std_logic_vector((P-1) downto 0)

);
end ISA_ochp;

architecture struct of ISA_ochp is

  COMPONENT ochp
    PORT (
      -- Control signals
      clk                   : IN    std_logic;
      reset_n               : IN    std_logic;

      -- Internal signals
      PIntSignal            : IN    std_logic_vector((N-1) downto 0);

      -- I/O Port input signals
```

ANNEXE F : Sources Bus ISA

```
      PSoP                    : IN    std_logic;
      PEoP                    : IN    std_logic;
      PCfgData                : IN    std_logic_vector(Cwid downto 0);

      -- I/O Port output signals
      PClkOut                 : OUT   std_logic;
      PConfID                 : OUT   std_logic_vector(Cwid downto 0);
      PDebugData              : OUT   std_logic_vector((P-1) downto 0)
    );
  END COMPONENT;

component controller is
port(   ctrl_clk:       in std_logic;
        ctrl_rst:       in std_logic;
        IOR:            in std_logic;
        IOW:            in std_logic;
        ALE:            in std_logic;
        WE_m:           in std_logic;
        RE_m:           in std_logic;
        Ald:            out std_logic;
        Dld:            out std_logic;
        Dst:            out std_logic;
         ochp_data:       out std_logic_vector(5 downto 0)
);
end component;

component datapath is
port(   dp_clk:         in std_logic;
        dp_rst:         in std_logic;
        A_ld:           in std_logic;
        D_ld:           in std_logic;
        D_st:           in std_logic;
        ADDR_P:         in std_logic_vector(15 downto 0);
        DIN:            in std_logic_vector(7 downto 0);
        Write_match:    out std_logic;
        Read_match:     out std_logic;
        DOUT:           out std_logic_vector(7 downto 0);
         ochp_data:       out std_logic_vector(15 downto 0)
);
end component;


signal WE_sig, RE_sig: std_logic;

signal Ald_sig, Dld_sig, Dst_sig: std_logic;

signal reset_n: std_logic;
signal IntDOUT_P: std_logic_vector(7 downto 0);
signal PIntSignal: std_logic_vector((N-1) downto 0);

begin


  DOUT_P <= IntDOUT_P;

  reset_n <= not RESET_P;
  PIntSignal((N-1) downto (N-40)) <= WE_sig & RE_sig &
                                     Ald_sig & Dld_sig & Dst_sig &
                                     IOR_P & IOW_P & ALE_P &
                                     ADDRESS_P & DIN_P & IntDOUT_P;

  PROBE: ochp
    port map (clk=>CLK_P,
              reset_n=>reset_n,
              PIntSignal=>PIntSignal,
              PSoP=>PSoP,
              PEoP=>PEoP,
              PCfgData=>PCfgData,
              PClkOut=>PClkOut,
              PConfID=>PConfID,
```

ANNEXE F : Sources Bus ISA

```
            PDebugData=>PDebugData
    );

    U0: controller port map(CLK_P,RESET_P,IOR_P,IOW_P,ALE_P,
                            WE_sig,RE_sig,Ald_sig,Dld_sig,Dst_sig,
                              PIntSignal(5 downto 0));
    U1: datapath port map(CLK_P,RESET_P,Ald_sig,Dld_sig,Dst_sig,
                            ADDRESS_P,DIN_P,WE_sig,RE_sig,IntDOUT_P,
                              PIntSignal((N-41) downto 6));

end struct;
```

ANNEXE F : Sources Bus ISA

```
------------------------------------------------------------
-- Descrition : Description vhdl comp d'un registre LFSR
--              de 16 bits "Linear Feedback Shift Register"
--
-- File    : LFSR.vhd
-- Author : A. Khouas, S. Regimbal
-- Date    : 13 oct 2003
--
-- Proj    : Test d'architectures de test integre "BIST",
--           laboratoire #4 du cours ELE6306.
------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;


entity LFSR is
    port (
        clk     :       in std_logic;
        reset   :       in std_logic;
        q       :       out std_logic_vector(15 downto 0));
end LFSR;

architecture BEH of LFSR is

signal  d       :       std_logic_vector(15 downto 0);

-- la valeur de l'etat initial du LFSR
constant lfsr_init : std_logic_vector(15 downto 0) := "1000000000000000";

begin

        process (clk, reset)

        begin
           if (reset='1') then -- Reset
              d <= lfsr_init;

           elsif (clk'event and clk = '1') then
              d(15) <= d(14) xor d(13) xor d(11) xor d(0);
              for i in 14 downto 0 loop
                 d(i) <= d(i+1) ;
              end loop ;

           end if ;

        end process;

        q <= d ;

end BEH;
```

ANNEXE G : Script d'injection de mutants

```
nawk -v filenam=$1 -v newfile=$2 'BEGIN{
  print"********************"
  print"* Mutant Injector *"
  print"********************"
  print" "
  print"Source file: "filenam
  print"Destination file: "newfile
  print" "

  flaglut = 0;
  numlut = 0;
  while(getline<filenam) {
    if(match($0,"cellRef LUT")&&!match($0,"debugpin")&&!match($0,"addrbus")) {
      numlut++;
    }
  }
  srand();
  mutantnum = int(numlut*rand());
  print"# LUT: "numlut
  print"Mutant location: LUT #"mutantnum
  print" "
  lutnum = 0;
}
{
  if(match($0,"cellRef LUT")&&!match($0,"debugpin")&&!match($0,"addrbus")) {
    flaglut = 1;
    lutnum++;
  }

  if(flaglut) {
    if(match($0,"property init")) {
      if(lutnum==mutantnum){
        gsub("string ..","string \"0");
      }
      flaglut = 0;
    }
  }

  print $0 > newfile
}
END{
  print"Done"
  print" "
}' $1
```

ANNEXE G : Script d'injection de mutants

```
********************
* Mutant Injector *
********************

Source file: ../ISA_spartan_good/ISA_spartan_good.edf
Destination file: ISA_spartan_mut1.edf

# LUT: 52
Mutant location: LUT #7

Done

********************
* Mutant Injector *
********************

Source file: ../ISA_spartan_good/ISA_spartan_good.edf
Destination file: ISA_spartan_mut3.edf

# LUT: 52
Mutant location: LUT #41

Done

********************
* Mutant Injector *
********************

Source file: ../ISA_spartan_good/ISA_spartan_good.edf
Destination file: ISA_spartan_mut4.edf

# LUT: 52
Mutant location: LUT #11

Done

********************
* Mutant Injector *
********************

Source file: ../ISA_spartan_good/ISA_spartan_good.edf
Destination file: ISA_spartan_mut5.edf

# LUT: 52
Mutant location: LUT #36

Done
```

ANNEXE H : Sources Assertions Bus ISA

```
------------------------------------------------------------
-- Description : Spartan 3s200 board ACF interface
--
-- File   : spartan_interf_lowmem.vhd
-- Author : K. Peterson
-- Created: Nov 10 2004
-- Modif. : Nov 10 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
use work.acftypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY spartan_interf IS
  PORT (clk          : IN  std_logic;
        reset        : IN  std_logic;

        -- PS2 keyboard signals
        PS2_clk    : in  std_logic;  -- Keyboard Clock Line
        PS2_Data   : in  std_logic;  -- Keyboard Data Line

        -- 4X Seven segments display signals
        digit_out : out std_logic_vector(3 downto 0);
        seg_out   : out std_logic_vector(7 downto 0);

        -- LED display signals
        led_out    : out std_logic_vector(7 downto 0);

        -- Display control switch
        pb_in      : in  std_logic;

--Using Spartan block RAM for the moment... Max 64*32 bits
--        -- SRAM signals
--        sramwe_n   : out std_logic;
--        sramoe_n   : out std_logic;
--        sramaddr   : out std_logic_vector(17 downto 0);
--        sramio     : inout std_logic_vector(31 downto 0);
--        sramce_n   : out std_logic_vector(1 downto 0);
--        sramub_n   : out std_logic_vector(1 downto 0);
--        sramlb_n   : out std_logic_vector(1 downto 0);

        -- ACF to OCHP interface
        PSoP       : OUT std_logic;
        PEoP       : OUT std_logic;
        PCfgData   : OUT std_logic_vector(Cwid downto 0);

        -- I/O Port output signals
        PConfID    : IN  std_logic_vector(Cwid downto 0);
        PDebugData: IN  std_logic_vector((P-1) downto 0)
  );

END spartan_interf;

ARCHITECTURE rtl OF spartan_interf IS

  COMPONENT acf_top
    PORT(clk          : IN  std_logic;
         reset_n      : IN  std_logic;

         -- ACF CPU Interface
         we_n         : IN  std_logic;
         regaddr      : IN  std_logic_vector(31 downto 0);
         regdatain    : IN  std_logic_vector(31 downto 0);
         regdataout   : OUT std_logic_vector(31 downto 0);
```

ANNEXE H : Sources Assertions Bus ISA

```
        irq         : OUT std_logic;

        -- Assertion RAM Interface
        ramwe_n     : OUT std_logic;
        ramoe_n     : OUT std_logic;
        ramdatain   : IN  std_logic_vector((dbgbuswidth-1) downto 0);
        ramdataout  : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        ramaddr     : OUT std_logic_vector(assramwid downto 0);

        -- OCHP Interface
        -- I/O Port input signals
        PSoP        : OUT std_logic;
        PEoP        : OUT std_logic;
        PCfgData    : OUT std_logic_vector(Cwid downto 0);

        -- I/O Port output signals
        PConfID     : IN  std_logic_vector(Cwid downto 0);
        PDebugData  : IN  std_logic_vector((P-1) downto 0)
    );
  END COMPONENT;

  COMPONENT PS2_Ctrl
    PORT(clk       : in  std_logic;  -- System Clock
         Reset     : in  std_logic;  -- System Reset
         PS2_clk   : in  std_logic;  -- Keyboard Clock Line
         PS2_Data  : in  std_logic;  -- Keyboard Data Line
         DoRead    : in  std_logic;  -- From outside when reading the scan code
         Scan_Err  : out std_logic;  -- To outside : Parity or Overflow error
         Scan_DAV  : out std_logic;  -- To outside when a scan code has arrived
         Scan_Code : out std_logic_vector(7 downto 0) -- Eight bits Data Out
    );
  END COMPONENT;

---- For validation
--    component LFSR
--      port (
--          clk     :       in std_logic;
--          reset   :       in std_logic;
--          q       :       out std_logic_vector(15 downto 0)
--    );
--    end component;

  CONSTANT cDATAHEXWID   : integer := 8;
  CONSTANT cASSERTRAMSIZE : integer := 16;  -- X 32 bits

  CONSTANT cADDRESSCHAR  : std_logic_vector(7 downto 0) := "00000100";
  CONSTANT cWRITECHAR    : std_logic_vector(7 downto 0) := "11000110";
  CONSTANT cREADCHAR     : std_logic_vector(7 downto 0) := "11110100";
  CONSTANT cDATACHAR     : std_logic_vector(7 downto 0) := "10000100";
  CONSTANT cSPACECHAR    : std_logic_vector(7 downto 0) := "11111111";
  CONSTANT c0CHAR        : std_logic_vector(7 downto 0) := "00000011";
  CONSTANT c1CHAR        : std_logic_vector(7 downto 0) := "10011111";
  CONSTANT c2CHAR        : std_logic_vector(7 downto 0) := "00100101";
  CONSTANT c3CHAR        : std_logic_vector(7 downto 0) := "00001101";
  CONSTANT c4CHAR        : std_logic_vector(7 downto 0) := "10011001";
  CONSTANT c5CHAR        : std_logic_vector(7 downto 0) := "01001001";
  CONSTANT c6CHAR        : std_logic_vector(7 downto 0) := "01000001";
  CONSTANT c7CHAR        : std_logic_vector(7 downto 0) := "00011111";
  CONSTANT c8CHAR        : std_logic_vector(7 downto 0) := "00000001";
  CONSTANT c9CHAR        : std_logic_vector(7 downto 0) := "00001001";
  CONSTANT cACHAR        : std_logic_vector(7 downto 0) := "00010001";
  CONSTANT cBCHAR        : std_logic_vector(7 downto 0) := "11000001";
  CONSTANT cCCHAR        : std_logic_vector(7 downto 0) := "11100101";
  CONSTANT cDCHAR        : std_logic_vector(7 downto 0) := "10000101";
  CONSTANT cECHAR        : std_logic_vector(7 downto 0) := "01100001";
  CONSTANT cFCHAR        : std_logic_vector(7 downto 0) := "01110001";

  CONSTANT cKEYB_0       : std_logic_vector(7 downto 0) := X"45";
  CONSTANT cKEYB_1       : std_logic_vector(7 downto 0) := X"16";
  CONSTANT cKEYB_2       : std_logic_vector(7 downto 0) := X"1E";
```

ANNEXE H : Sources Assertions Bus ISA

```
  CONSTANT cKEYB_3        : std_logic_vector(7 downto 0) := X"26";
  CONSTANT cKEYB_4        : std_logic_vector(7 downto 0) := X"25";
  CONSTANT cKEYB_5        : std_logic_vector(7 downto 0) := X"2E";
  CONSTANT cKEYB_6        : std_logic_vector(7 downto 0) := X"36";
  CONSTANT cKEYB_7        : std_logic_vector(7 downto 0) := X"3D";
  CONSTANT cKEYB_8        : std_logic_vector(7 downto 0) := X"3E";
  CONSTANT cKEYB_9        : std_logic_vector(7 downto 0) := X"46";
  CONSTANT cKEYB_A        : std_logic_vector(7 downto 0) := X"1C";
  CONSTANT cKEYB_B        : std_logic_vector(7 downto 0) := X"32";
  CONSTANT cKEYB_C        : std_logic_vector(7 downto 0) := X"21";
  CONSTANT cKEYB_D        : std_logic_vector(7 downto 0) := X"23";
  CONSTANT cKEYB_E        : std_logic_vector(7 downto 0) := X"24";
  CONSTANT cKEYB_F        : std_logic_vector(7 downto 0) := X"2B";
  CONSTANT cKEYB_R        : std_logic_vector(7 downto 0) := X"2D";
  CONSTANT cKEYB_W        : std_logic_vector(7 downto 0) := X"1D";
  CONSTANT cKEYB_BACKSP   : std_logic_vector(7 downto 0) := X"66";
  CONSTANT cKEYB_ESCAPE   : std_logic_vector(7 downto 0) := X"76";
  CONSTANT cKEYB_ENTER    : std_logic_vector(7 downto 0) := X"5A";
  CONSTANT cKEYB_KEYUP    : std_logic_vector(7 downto 0) := X"F0";

  CONSTANT cZEROSREGS     : std_logic_vector(31 downto 0) := X"00000000";
  CONSTANT cZEROSKEYB     : std_logic_vector(23 downto 0) := X"000000";
  CONSTANT cZEROSSEGS     : std_logic_vector(7 downto 0)  := X"00";

  TYPE    t_disp_state IS (DATAR,ENTERRADDR,ENTERWADDR,ENTERWDATA);

  TYPE    t_segreg     IS ARRAY (3 downto 0) OF std_logic_vector(7 downto 0);

  SIGNAL  disp_state    : t_disp_state;

  SIGNAL ACFreset_n     : std_logic;
  SIGNAL ACFwe_n        : std_logic;
  SIGNAL ACFregaddr     : std_logic_vector(31 downto 0);
  SIGNAL ACFregdatain   : std_logic_vector(31 downto 0);
  SIGNAL ACFregdataout  : std_logic_vector(31 downto 0);
  SIGNAL ACFirq         : std_logic;
  SIGNAL ACFramwe_n     : std_logic;
  SIGNAL ACFramoe_n     : std_logic;
  SIGNAL ACFramdatain   : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL ACFramdataout  : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL ACFramaddr     : std_logic_vector(assramwid downto 0);
  SIGNAL PS2DoRead      : std_logic;
  SIGNAL PS2Scan_Err    : std_logic;
  SIGNAL PS2Scan_DAV    : std_logic;
  SIGNAL PS2Scan_Code   : std_logic_vector(7 downto 0);

  SIGNAL addrreg        : std_logic_vector(31 downto 0);
  SIGNAL datawreg       : std_logic_vector(31 downto 0);
  SIGNAL swpcounter     : std_logic_vector(15 downto 0);
  SIGNAL keybcounter    : std_logic_vector(23 downto 0);

  SIGNAL segreg         : t_segreg;
  SIGNAL keybindex      : integer range 0 to 7;
  SIGNAL sweepindex     : integer range 0 to 3;

---- For validation
--  SIGNAL lfsrout        : std_logic_vector(15 downto 0);

-- Assertion RAM made of Block RAM
  TYPE   mem_type     IS array ((cASSERTRAMSIZE-1) downto 0) of
std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL assertmem      : mem_type;

  FUNCTION SevenSeg (V : std_logic_vector(3 downto 0))
           RETURN std_logic_vector IS
  begin
    case V is
      when  x"0"  => return c0CHAR;
      when  x"1"  => return c1CHAR;
      when  x"2"  => return c2CHAR;
```

ANNEXE H : Sources Assertions Bus ISA

```
      when  x"3"  => return c3CHAR;
      when  x"4"  => return c4CHAR;
      when  x"5"  => return c5CHAR;
      when  x"6"  => return c6CHAR;
      when  x"7"  => return c7CHAR;
      when  x"8"  => return c8CHAR;
      when  x"9"  => return c9CHAR;
      when  x"A"  => return cACHAR;
      when  x"B"  => return cBCHAR;
      when  x"C"  => return cCCHAR;
      when  x"D"  => return cDCHAR;
      when  x"E"  => return cECHAR;
      when  x"F"  => return cFCHAR;
      when others => return "00000000";
    end case;
  end FUNCTION;

  FUNCTION KeybDecode (K : std_logic_vector(7 downto 0))
          RETURN std_logic_vector IS
  begin
    case K is
      when cKEYB_0 => return x"0";
      when cKEYB_1 => return x"1";
      when cKEYB_2 => return x"2";
      when cKEYB_3 => return x"3";
      when cKEYB_4 => return x"4";
      when cKEYB_5 => return x"5";
      when cKEYB_6 => return x"6";
      when cKEYB_7 => return x"7";
      when cKEYB_8 => return x"8";
      when cKEYB_9 => return x"9";
      when cKEYB_A => return x"A";
      when cKEYB_B => return x"B";
      when cKEYB_C => return x"C";
      when cKEYB_D => return x"D";
      when cKEYB_E => return x"E";
      when cKEYB_F => return x"F";
      when others  => return x"0";
    end case;
  end FUNCTION;

BEGIN

  ACFreset_n <= not reset;

  ACF: acf_top
    port map (clk=>clk,
              reset_n=>ACFreset_n,
              we_n=>ACFwe_n,
              regaddr=>ACFregaddr,
              regdatain=>ACFregdatain,
              regdataout=>ACFregdataout,
              irq=>ACFirq,
              ramwe_n=>ACFramwe_n,
              ramoe_n=>ACFramoe_n,
              ramdatain=>ACFramdatain,
              ramdataout=>ACFramdataout,
              ramaddr=>ACFramaddr,
              PSoP=>PSoP,
              PEoP=>PEoP,
              PCfgData=>PCfgData,
              PConfID=>PConfID,
--              PDebugData=>lfsrout
              PDebugData=>PDebugData
    );

  PS2KEYB: PS2_Ctrl
    port map (clk=>clk,
              Reset=>reset,
              PS2_clk=>PS2_Clk,
```

ANNEXE H : Sources Assertions Bus ISA

```
            PS2_Data=>PS2_Data,
            DoRead=>PS2DoRead,
            Scan_Err=>PS2Scan_Err,
            Scan_DAV=>PS2Scan_DAV,
            Scan_Code=>PS2Scan_Code
    );

---- For validation
--    STATEGEN: LFSR
--        port map (clk=>clk, reset=>reset, q=>lfsrout);


  ACFregaddr   <= addrreg;
  ACFregdatain <= datawreg;

  keyboard_mgr: PROCESS (clk, reset)
  BEGIN

    IF (reset = '1') THEN
      addrreg      <= cZEROSREGS;
      datawreg     <= cZEROSREGS;
      PS2DoRead    <= '0';
      keybindex    <= 0;
      disp_state   <= DATAR;
      ACFwe_n      <= '1';
      keybcounter  <= cZEROSKEYB;

    ELSIF clk'event AND clk = '1' THEN
      IF (PS2Scan_DAV = '1') AND (PS2DoRead = '0') THEN
        PS2DoRead  <= '1';
        ACFwe_n    <= '1';

        CASE disp_state IS
          WHEN DATAR =>
            CASE PS2Scan_Code IS

              WHEN cKEYB_R       =>
                disp_state  <= ENTERRADDR;
                keybindex   <= 7;

              WHEN cKEYB_W       =>
                disp_state  <= ENTERWADDR;
                keybindex   <= 7;

              WHEN OTHERS        =>
                disp_state  <= DATAR;
            END CASE;

          WHEN ENTERRADDR =>
            CASE PS2Scan_Code IS
              WHEN cKEYB_ENTER  =>
                disp_state <= DATAR;


              WHEN cKEYB_ESCAPE =>
                disp_state <= DATAR;

              WHEN cKEYB_BACKSP =>
                IF keybindex < 7 THEN
                  keybindex <= keybindex + 1;
                END IF;
                disp_state <= ENTERRADDR;

              WHEN cKEYB_KEYUP =>
                disp_state <= ENTERRADDR;

              WHEN OTHERS        =>

                FOR i IN 7 DOWNTO 0 LOOP
                  IF i = keybindex THEN
```

ANNEXE H : Sources Assertions Bus ISA

```
            addrreg((((i+1)*4)-1) downto i*4) <=
              KeybDecode(PS2Scan_Code);
          END IF;
        END LOOP;
        IF keybindex > 0 THEN
          keybindex <= keybindex - 1;
        END IF;
        disp_state <= ENTERRADDR;

    END CASE;

  WHEN ENTERWADDR =>
    CASE PS2Scan_Code IS
      WHEN cKEYB_ENTER  =>
        keybindex  <= 7;
        disp_state <= ENTERWDATA;

      WHEN cKEYB_ESCAPE =>
        disp_state <= DATAR;

      WHEN cKEYB_BACKSP =>
        IF keybindex < 7 THEN
          keybindex <= keybindex + 1;
        END IF;
        disp_state <= ENTERWADDR;

      WHEN cKEYB_KEYUP =>
        disp_state <= ENTERWADDR;

      WHEN OTHERS       =>

        FOR i IN 7 DOWNTO 0 LOOP
          IF i = keybindex THEN
            addrreg((((i+1)*4)-1) downto i*4) <=
              KeybDecode(PS2Scan_Code);
          END IF;
        END LOOP;
        IF keybindex > 0 THEN
          keybindex <= keybindex - 1;
        END IF;
        disp_state <= ENTERWADDR;

    END CASE;

  WHEN ENTERWDATA =>
    CASE PS2Scan_Code IS
      WHEN cKEYB_ENTER  =>
        ACFwe_n      <= '0';
        disp_state <= DATAR;

      WHEN cKEYB_ESCAPE =>
        disp_state <= DATAR;

      WHEN cKEYB_BACKSP =>
        IF keybindex < 7 THEN
          keybindex <= keybindex + 1;
        END IF;
        disp_state <= ENTERWDATA;

      WHEN cKEYB_KEYUP =>
        disp_state <= ENTERWDATA;

      WHEN OTHERS       =>

        FOR i IN 7 DOWNTO 0 LOOP
          IF i = keybindex THEN
            datawreg((((i+1)*4)-1) downto i*4) <=
              KeybDecode(PS2Scan_Code);
          END IF;
        END LOOP;
```

ANNEXE H : Sources Assertions Bus ISA

```
            IF keybindex > 0 THEN
              keybindex <= keybindex - 1;
            END IF;
            disp_state <= ENTERWDATA;

          END CASE;

        WHEN OTHERS =>
          disp_state <= DATAR;
          keybindex  <= 0;
      END CASE;

    ELSIF (PS2DoRead = '1') THEN
      IF keybcounter = X"111111" THEN
        PS2DoRead <= '0';
        keybcounter <= X"000000";
      ELSE
        keybcounter <= keybcounter + 1;
      END IF;
    END IF;
  END IF;

END PROCESS;

display_mgr: PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    FOR i IN 3 downto 0 LOOP
      segreg(i) <= not cZEROSSEGS;
    END LOOP;
    led_out <= cZEROSSEGS;
  ELSIF clk'event AND clk = '1' THEN
    led_out <= cZEROSSEGS;
    IF (disp_state = ENTERRADDR) AND (keybindex = 7) THEN
      led_out(7) <= '1';
      FOR i IN 3 downto 1 LOOP
        segreg(i) <= cSPACECHAR;
      END LOOP;
      segreg(0) <= cREADCHAR;
    ELSIF (disp_state = ENTERWADDR) AND (keybindex = 7) THEN
      led_out(7) <= '1';
      FOR i IN 3 downto 1 LOOP
        segreg(i) <= cSPACECHAR;
      END LOOP;
      segreg(0) <= cWRITECHAR;
    ELSIF (disp_state = ENTERWDATA) AND (keybindex = 7) THEN
      led_out(7) <= '1';
      FOR i IN 3 downto 1 LOOP
        segreg(i) <= cSPACECHAR;
      END LOOP;
      segreg(0) <= cDATACHAR;
    ELSIF disp_state = DATAR THEN
      FOR i IN 3 downto 0 LOOP
        IF pb_in = '0' THEN
          segreg(i) <= SevenSeg(ACFregdataout((((i+1)*4)-1) downto i*4));
          led_out(i) <= '1';
        ELSE
          segreg(i) <= SevenSeg(ACFregdataout((((i+5)*4)-1) downto (i+4)*4));
          led_out(i+4) <= '1';
        END IF;
      END LOOP;
    ELSIF (disp_state = ENTERRADDR) OR (disp_state = ENTERWADDR) THEN
      led_out(keybindex) <= '1';
      FOR i IN 3 downto 0 LOOP
        IF (keybindex < 3) THEN
          segreg(i) <= SevenSeg(addrreg((((i+1)*4)-1) downto i*4));
        ELSE
          segreg(i) <= SevenSeg(addrreg((((i+5)*4)-1) downto (i+4)*4));
        END IF;
      END LOOP;
```

ANNEXE H : Sources Assertions Bus ISA

```
      ELSIF disp_state = ENTERWDATA THEN
        led_out(keybindex) <= '1';
        FOR i IN 3 downto 0 LOOP
          IF (keybindex < 3) THEN
            segreg(i) <= SevenSeg(datawreg((((i+1)*4)-1) downto i*4));
          ELSE
            segreg(i) <= SevenSeg(datawreg((((i+5)*4)-1) downto (i+4)*4));
          END IF;
        END LOOP;
      END IF;
    END IF;
  END PROCESS;


--Using Spartan block RAM for the moment... Max 64*32 bits
--  sramce_n <= clk & clk WHEN ACFramwe_n = '0' ELSE
--              "00";
--  sramub_n <= "00";
--  sramlb_n <= "00";
--  sramaddr <= ACFramaddr(sramaddr'high downto 0);
--  sramio   <= ACFramdataout WHEN ACFramwe_n = '0' ELSE (others => 'Z');
--  ACFramdatain <= sramio;
--
--  sram_ctrl: PROCESS(clk,reset)
--  BEGIN
--    IF reset = '1' THEN
--      sramwe_n <= '1';
--      sramoe_n <= '1';
--    ELSIF clk'event AND clk = '1' THEN  -- Sur front montant
--      sramwe_n <= '1';
--      sramoe_n <= '1';
--      IF ACFramwe_n = '0' THEN
--        sramwe_n <= '0';
--      ELSE
--        sramoe_n <= '0';
--      END IF;
--    END IF;
--  END PROCESS;

  ACFramdatain <= assertmem(CONV_INTEGER(ACFramaddr));

  bram_ctrl: PROCESS(clk,ACFramwe_n,ACFramaddr)
  BEGIN
    IF clk'event AND clk = '1' THEN

      IF ACFramwe_n = '0' THEN
        assertmem(CONV_INTEGER(ACFramaddr)) <= ACFramdataout;
      END IF;

    END IF;
  END PROCESS;


  SWEEP: PROCESS(clk,reset)
  BEGIN
    IF reset = '1' THEN
      digit_out   <= "1110";
      sweepindex <= 0;
      seg_out          <= segreg(sweepindex);
      swpcounter           <= "0000000000000000";
    ELSIF clk'event AND clk = '1' THEN
      swpcounter <= swpcounter + 1;
      IF swpcounter >= "1111111111111111" THEN
        swpcounter   <= "0000000000000000";
        FOR i IN 3 downto 0 LOOP
          IF i = sweepindex THEN
            digit_out(i) <= '0';
          ELSE
            digit_out(i) <= '1';
          END IF;
```

ANNEXE H : Sources Assertions Bus ISA

```
        END LOOP;
        seg_out          <= segreg(sweepindex);

        sweepindex <= sweepindex + 1;
        IF sweepindex >= 3 THEN
          sweepindex <= 0;
        END IF;
      END IF;
    END IF;
  END PROCESS;

END rtl;
```

ANNEXE H : Sources Assertions Bus ISA

```
------------------------------------------------------------
-- Descrition : Description vhdl comp d'un registre LFSR
--              de 16 bits "Linear Feedback Shift Register"
--
-- File    : LFSR.vhd
-- Author : A. Khouas, S. Regimbal
-- Date    : 13 oct 2003
--
-- Proj    : Test d'architectures de test integre "BIST",
--           laboratoire #4 du cours ELE6306.
------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;


entity LFSR is
    port (
        clk     :       in std_logic;
        reset   :       in std_logic;
        q       :       out std_logic_vector(15 downto 0));
end LFSR;

architecture BEH of LFSR is

signal  d       :       std_logic_vector(15 downto 0);

-- la valeur de l'etat initial du LFSR
constant lfsr_init : std_logic_vector(15 downto 0) := "1000000000000000";

begin

        process (clk, reset)

        begin
           if (reset='1') then -- Reset
              d <= lfsr_init;

           elsif (clk'event and clk = '1') then
              d(15) <= d(14) xor d(13) xor d(11) xor d(0);
              for i in 14 downto 0 loop
                 d(i) <= d(i+1) ;
              end loop ;

           end if ;

        end process;

        q <= d ;

end BEH;
```

ANNEXE H : Sources Assertions Bus ISA

```
-- PS2_Ctrl.vhd
-- ------------------------------------------------
--   Simplified PS/2 Controller  (kbd, mouse...)
-- ------------------------------------------------
-- Only the Receive function is implemented !
-- (c) ALSE. http://www.alse-fr.com


library IEEE;
  use IEEE.Std_Logic_1164.all;
  use IEEE.Numeric_Std.all;

-- --------------------------------------
    Entity PS2_Ctrl is
-- --------------------------------------
  generic (FilterSize : positive := 8);
  port( Clk       : in  std_logic;  -- System Clock
        Reset     : in  std_logic;  -- System Reset
        PS2_Clk   : in  std_logic;  -- Keyboard Clock Line
        PS2_Data  : in  std_logic;  -- Keyboard Data Line
        DoRead    : in  std_logic;  -- From outside when reading the scan code
        Scan_Err  : out std_logic;  -- To outside : Parity or Overflow error
        Scan_DAV  : out std_logic;  -- To outside when a scan code has arrived
        Scan_Code : out std_logic_vector(7 downto 0) -- Eight bits Data Out
        );
end PS2_Ctrl;

---- --------------------------------------
--    Architecture Plain_Wrong of PS2_Ctrl is
---- --------------------------------------
---- Comments : Bad solution !
---- 3 clock domains, no global reset, disguised FSM...
---- There is also a bad problem with the synchronous reset
---- which is not on the right clock domain !
---- Note : the VHDL style has been fixed...
--
--  signal Bit_Cnt    : unsigned (3 downto 0);
--  signal Shift_Reg  : std_logic_vector(8 downto 0);
--  signal Read_CHAR  : std_logic;
--  signal Ready_set  : std_logic;
--  signal PS2_Clk_f  : std_logic;
--  signal Filter     : std_logic_vector(FilterSize-1 downto 0);
--
--begin
--
--Scan_Err <= '0';  -- not used in this architecture
--

--  -- Clock domain #1 , async reset #1
--
--  process (DoRead, Ready_set)
--  begin
--    if DoRead = '1' then
--      Scan_DAV <= '0';
--    elsif rising_edge(Ready_set) then
--      Scan_DAV <= '1';
--    end if;
--  end process;
--
--
--  -- Clock domain #2, no reset !
--
---- This process filters the raw clock signal coming from the keyboard
---- using a shift register and two 4-inputs AND gates
---- Implies a 320 ns delay
--  Clock_filter : process (Clk)
--  begin
--    if rising_edge (Clk) then
--      Filter <= PS2_Clk & Filter(Filter'high downto 1);
--      if Filter = "11111111" then
```

ANNEXE H : Sources Assertions Bus ISA

```
--        PS2_Clk_f <= '1';
--      elsif Filter = "00000000" then
--        PS2_Clk_f <= '0';
--      end if;
--    end if;
--  end process Clock_filter;
--
--  -- Clock domain #3, partial synchronous reset !
--  -- PS2_Data is not resynchronized
--
--  --This process Reads in serial data coming from the keyboard
--  process(PS2_Clk_f)
--  begin
--    if rising_edge (PS2_Clk_f) then

--      if RESET = '1' then
--        Bit_Cnt  <= (others => '0');
--        Read_CHAR <= '0';
--      else
--        if PS2_Data = '0' and Read_CHAR = '0' then
--          Read_CHAR <= '1';
--          Ready_set <= '0';
--        else
--          -- Shift in next 8 data bits to assemble a scan code
--          if Read_CHAR = '1' then
--            if Bit_Cnt < 9 then
--              Bit_Cnt  <= Bit_Cnt + 1;
--              Shift_Reg <= PS2_Data & Shift_Reg(8 downto 1);
--              Ready_set <= '0';
--            else -- End of scan code character, so set flags and exit loop
--              Scan_Code <= Shift_Reg (7 downto 0);
--              Read_CHAR <= '0';
--              Ready_set <= '1';
--              Bit_Cnt     <= (others => '0');
--            end if;
--          end if;
--        end if;
--      end if;
--    end if;
--  end process;
--
--end Plain_Wrong;


-- -------------------------------------
    Architecture ALSE_RTL of PS2_Ctrl is
-- -------------------------------------
-- (c) ALSE. http://www.alse-fr.com
-- Author : Bert Cuzeau.
-- Fully synchronous solution, same Filter on PS2_Clk.
-- Still as compact as "Plain_wrong"...
-- Possible improvement : add TIMEOUT on PS2_Clk while shifting
-- Note: PS2_Data is resynchronized though this should not be
-- necessary (qualified by Fall_Clk and does not change at that time).
-- Note the tricks to correctly interpret 'H' as '1' in RTL simulation.

  signal PS2_Datr  : std_logic;

  subtype Filter_t is std_logic_vector(FilterSize-1 downto 0);
  signal Filter    : Filter_t;
  signal Fall_Clk  : std_logic;
  signal Bit_Cnt   : unsigned (3 downto 0);
  signal Parity    : std_logic;
  signal Scan_DAVi : std_logic;

  signal S_Reg     : std_logic_vector(8 downto 0);

  signal PS2_Clk_f : std_logic;

  Type   State_t is (Idle, Shifting);
```

ANNEXE H : Sources Assertions Bus ISA

```
  signal State : State_t;

begin

Scan_DAV <= Scan_DAVi;

-- This filters digitally the raw clock signal coming from the keyboard :
--   * Eight consecutive PS2_Clk=1 makes the filtered_clock go high
--   * Eight consecutive PS2_Clk=0 makes the filtered_clock go low
-- Implies a (FilterSize+1) x Tsys_clock delay on Fall_Clk wrt Data
-- Also in charge of the re-synchronization of PS2_Data

process (Clk,Reset)
begin
  if Reset='1' then
    PS2_Datr  <= '0';
    PS2_Clk_f <= '0';
    Filter    <= (others=>'0');
    Fall_Clk  <= '0';
  elsif rising_edge (Clk) then
    PS2_Datr <= PS2_Data and PS2_Data; -- also turns 'H' into '1'
    Fall_Clk <= '0';
    Filter   <= (PS2_Clk and PS2_CLK) & Filter(Filter'high downto 1);
    if Filter = Filter_t'(others=>'1') then
      PS2_Clk_f <= '1';
    elsif Filter = Filter_t'(others=>'0') then
      PS2_Clk_f <= '0';
      if PS2_Clk_f = '1' then
        Fall_Clk <= '1';
      end if;
    end if;
  end if;
end process;


-- This simple State Machine reads in the Serial Data
-- coming from the PS/2 peripheral.

process(Clk,Reset)
begin

  if Reset='1' then
    State     <= Idle;
    Bit_Cnt   <= (others => '0');
    S_Reg     <= (others => '0');
    Scan_Code <= (others => '0');
    Parity    <= '0';
    Scan_Davi <= '0';
    Scan_Err  <= '0';

  elsif rising_edge (Clk) then

    if DoRead='1' then
      Scan_Davi <= '0'; -- note: this assgnmnt can be overriden
    end if;

    case State is

      when Idle =>
        Parity  <= '0';
        Bit_Cnt <= (others => '0');
        -- note that we dont need to clear the Shift Register
        if Fall_Clk='1' and PS2_Datr='0' then -- Start bit
          Scan_Err <= '0';
          State <= Shifting;
        end if;

      when Shifting =>
          if Bit_Cnt >= 9 then
            if Fall_Clk='1' then -- Stop Bit
```

ANNEXE H : Sources Assertions Bus ISA

```
            -- Error is (wrong Parity) or (Stop='0') or Overflow

            Scan_Err  <= (not Parity) or (not PS2_Datr) or Scan_DAVi;
            Scan_Davi <= '1';
            Scan_Code <= S_Reg(7 downto 0);
            State <= Idle;
          end if;
        elsif Fall_Clk='1' then
          Bit_Cnt  <= Bit_Cnt + 1;
          S_Reg <= PS2_Datr & S_Reg (S_Reg'high downto 1); -- Shift right
          Parity <= Parity xor PS2_Datr;
        end if;

      when others => -- never reached
        State <= Idle;

    end case;

    --Scan_Err <= '0'; -- to create an on-purpose error on Scan_Err !

  end if;

end process;

end ALSE_RTL;
```

ANNEXE H : Sources Assertions Bus ISA

```
------------------------------------------------------------
-- Description : Assertion Checkers + Event Generators
--               module
--               Synthesized AC and EG specific to ISA
--               bus interface + OCHP design
--
-- File   : ac_eg_module_ISA_ochp_set1.vhd
-- Author : K. Peterson
-- Created: Oct 29 2004
-- Modif. : Oct 31 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
use work.acftypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY ac_eg_module IS
  PORT (clk          : IN  std_logic;
        reset_n      : IN  std_logic;

        -- OCHP debug signals input
        PConfId      : IN  std_logic_vector(Cwid downto 0);
        PDebugData   : IN  std_logic_vector((P-1) downto 0);

        -- AC controller signals
        ACCwe_n      : IN  std_logic;
        ACCout_addr  : IN  std_logic_vector(nACwid downto 0);
        ACCdat_type  : IN  std_logic;
        ACCresetin   : IN  std_logic_vector(31 downto 0);
        ACCdatar     : OUT std_logic_vector((dbgbuswidth-1) downto 0);

        -- EG controller signals
        EGCwe_n      : IN  std_logic;
        EGCout_addr  : IN  std_logic_vector(nEGwid downto 0);
        EGCdat_type  : IN  std_logic;
        EGCtransdes  : IN  std_logic;
        EGCclrirq    : IN  std_logic;
        EGCcfgin     : IN  std_logic_vector(31 downto 0);
        EGCdatar     : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        EGCirq       : OUT std_logic;

        -- Timestamp Timer signals
        TTin         : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Assertion RAM signals
        ARAMdbgdata  : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        ARAMegevent  : OUT std_logic_vector((nEG-1) downto 0);

        -- Configuration Manager signals
        CMcfgswap    : OUT std_logic;
        CMswapid     : OUT std_logic_vector(Cwid downto 0)
  );

END ac_eg_module;

ARCHITECTURE rtl OF ac_eg_module IS

    component assert_always
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
```

ANNEXE H : Sources Assertions Bus ISA

```
        port (
          clk, reset_n : IN std_ulogic;
          test_expr    : IN boolean;
          valide       : OUT std_ulogic);
    end component;

    component assert_never
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
        port (
          clk, reset_n : IN std_ulogic;
          test_expr    : IN boolean;
          valide       : OUT std_ulogic);
    end component;

    component assert_implication
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
        port (
          clk, reset_n                 : IN std_ulogic;
          antecedent_expr, consequent_expr : IN boolean;
          valide                       : OUT std_ulogic);
    end component;


  COMPONENT ac_harness
    GENERIC (
      dbginfowidth   : integer   := 32;
      dbginfotype    : integer   := 0;  -- Default: Timestamp register
      failedlatched  : integer   := 0   -- Default: Failed not latched
    );
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- Assertion checker control signals
          clrfail    : IN  std_logic;
          clrinforeg : IN  std_logic;
          ACenable   : IN  std_logic;

          -- Assertion checker debug data input
          dbginput   : IN  std_logic_vector((dbginfowidth-1) downto 0);
          validin    : IN  std_logic;

          -- Assertion checker debug data output
          dbginfo    : OUT std_logic_vector((dbginfowidth-1) downto 0);
          failed_n   : OUT std_logic);
  END COMPONENT;

  COMPONENT ac_ctrl
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- AC control signals
          we_n       : IN  std_logic;
          out_addr   : IN  std_logic_vector(nACwid downto 0);
          rdat_type  : IN  std_logic;

          -- AC debug data input
          statein    : IN  std_logic_vector((nAC-1) downto 0);
          ttin       : IN  std_logic_vector(((dbgbuswidth*nAC)-1) downto 0);
          resetin    : IN  std_logic_vector(31 downto 0);
```

ANNEXE H : Sources Assertions Bus ISA

```
        -- AC debug data output
        datar       : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        resetout    : OUT std_logic_vector((nAC-1) downto 0)
  );
end COMPONENT;

COMPONENT eg_ctrl
  PORT (clk         : IN  std_logic;
        reset_n     : IN  std_logic;

        -- EG control signals
        we_n        : IN  std_logic;
        out_addr    : IN  std_logic_vector(nEGwid downto 0);
        rdat_type   : IN  std_logic;
        transdest   : IN  std_logic;
        clrirq      : IN  std_logic;

        -- EG debug data input
        statein     : IN  std_logic_vector((nEG-1) downto 0);
        ttin        : IN  std_logic_vector(((dbgbuswidth*nEG)-1) downto 0);
        cfgin       : IN  std_logic_vector(31 downto 0);

        -- EG debug data output
        datar       : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        resetout    : OUT std_logic_vector((nEG-1) downto 0);
        irq         : OUT std_logic
  );
end COMPONENT;

SIGNAL ACEGconfid  : std_logic_vector(Cwid downto 0);
SIGNAL ACEGdbgdata : std_logic_vector((P-1) downto 0);
SIGNAL ACstate     : std_logic_vector((nAC-1) downto 0);
SIGNAL ACdbginfo   : std_logic_vector(((dbgbuswidth*nAC)-1) downto 0);
SIGNAL ACreset     : std_logic_vector((nAC-1) downto 0);
SIGNAL EGstate     : std_logic_vector((nEG-1) downto 0);
SIGNAL EGdbginfo   : std_logic_vector(((dbgbuswidth*nEG)-1) downto 0);
SIGNAL EGreset     : std_logic_vector((nEG-1) downto 0);
SIGNAL ACEGramdata : std_logic_vector((dbgbuswidth-1) downto 0);
SIGNAL ACEGcfgswap : std_logic;
SIGNAL ACEGcfgnum  : std_logic_vector(Cwid downto 0);
SIGNAL ACvalid     : std_logic_vector((nAC-1) downto 0);
SIGNAL EGvalid     : std_logic_vector((nEG-1) downto 0);

-- DUT Debug signals
SIGNAL DUTCStateV  : std_logic_vector(2 downto 0);
SIGNAL DUTAld      : std_logic;
SIGNAL DUTDld      : std_logic;
SIGNAL DUTDst      : std_logic;
SIGNAL DUTALE      : std_logic;
SIGNAL DUTDout     : std_logic_vector(3 downto 0);
SIGNAL DUTAddr     : std_logic_vector(4 downto 0);
SIGNAL DUTWem      : std_logic;
SIGNAL DUTRem      : std_logic;
SIGNAL DUTIOW      : std_logic;
SIGNAL DUTIOR      : std_logic;
SIGNAL DUTDin      : std_logic_vector(3 downto 0);

SIGNAL DUTLstDin   : std_logic_vector(3 downto 0);
SIGNAL DUTreg2addr : std_logic_vector(3 downto 0);
SIGNAL DUTaddr2reg : std_logic_vector(3 downto 0);
SIGNAL DUTLstad2rg : std_logic_vector(3 downto 0);

-- Assertions expressions signals
SIGNAL CurS0       : boolean;
SIGNAL LstS0       : boolean;
SIGNAL CurS1       : boolean;
SIGNAL LstS1       : boolean;
SIGNAL CurS2       : boolean;
SIGNAL LstS2       : boolean;
SIGNAL CurS3       : boolean;
```

ANNEXE H : Sources Assertions Bus ISA

```
  SIGNAL LstS3        : boolean;
  SIGNAL CurS3a       : boolean;
  SIGNAL LstS3a       : boolean;
  SIGNAL CurS4        : boolean;
  SIGNAL LstS4        : boolean;
  SIGNAL CurS4a       : boolean;
  SIGNAL LstS4a       : boolean;
  SIGNAL CurS5        : boolean;
  SIGNAL CurS0ALE0    : boolean;
  SIGNAL LstS0ALE0    : boolean;
  SIGNAL CurS0ALE1    : boolean;
  SIGNAL LstS0ALE1    : boolean;
  SIGNAL CurS2We0Re0  : boolean;
  SIGNAL LstS2We0Re0  : boolean;
  SIGNAL CurS2We1     : boolean;
  SIGNAL LstS2We1     : boolean;
  SIGNAL CurS2We0Re1  : boolean;
  SIGNAL LstS2We0Re1  : boolean;
  SIGNAL CurS3IOW1    : boolean;
  SIGNAL LstS3IOW1    : boolean;
  SIGNAL CurS3IOW0    : boolean;
  SIGNAL LstS3IOW0    : boolean;
  SIGNAL CurS4IOR1    : boolean;
  SIGNAL LstS4IOR1    : boolean;
  SIGNAL CurS4IOR0    : boolean;
  SIGNAL LstS4IOR0    : boolean;
  SIGNAL CurAld0      : boolean;
  SIGNAL CurAld1      : boolean;
  SIGNAL CurDld0      : boolean;
  SIGNAL CurDld1      : boolean;
  SIGNAL CurDst0      : boolean;
  SIGNAL CurDst1      : boolean;
  SIGNAL CurS2Adr10   : boolean;
  SIGNAL CurS2Adr11   : boolean;
  SIGNAL CurS2Adr00   : boolean;
  SIGNAL CurWem1Rem0  : boolean;
  SIGNAL CurWem0Rem1  : boolean;
  SIGNAL CurWem0Rem0  : boolean;
  SIGNAL CurDinReg2   : boolean;
  SIGNAL CurAdd2Dout  : boolean;

  -- Event generators expressions signals
  SIGNAL EvtACStateM : boolean;
  SIGNAL EvtACAlDlDs : boolean;
  SIGNAL EvtACAdDat  : boolean;
  SIGNAL EvtACFail   : boolean;

BEGIN

  -- Debug signals assignations
  DUTCStateV  <= PDebugData(15 downto 13);
  DUTAld      <= PDebugData(12);
  DUTDld      <= PDebugData(11);
  DUTDst      <= PDebugData(10);
  DUTALE      <= PDebugData(9) WHEN not (PConfID = "01") ELSE
                 '0';
  DUTDout     <= PDebugData(8 downto 5) WHEN (PConfID = "00") ELSE
                 PDebugData(3 downto 0) WHEN (PConfID = "11") ELSE
                 "0000";
  DUTAddr     <= PDebugData(4 downto 0) WHEN (PConfID(1) = '0') ELSE
                 "00000";
  DUTWem      <= PDebugData(9) WHEN (PConfID = "01") ELSE
                 '0';
  DUTRem      <= PDebugData(8) WHEN (PConfID = "01") ELSE
                 '0';
  DUTIOW      <= PDebugData(7) WHEN (PConfID = "01") ELSE
                 PDebugData(8) WHEN (PConfID = "10") ELSE
                 '1';
  DUTIOR      <= PDebugData(6) WHEN (PConfID = "01") ELSE
                 PDebugData(8) WHEN (PConfID = "11") ELSE
```

ANNEXE H : Sources Assertions Bus ISA

```
                '1';
DUTDin      <= PDebugData(7 downto 4) WHEN (PConfID = "10") ELSE
                "0000";
DUTreg2addr <= PDebugData(3 downto 0) WHEN (PConfID = "10") ELSE
                "0000";
DUTaddr2reg <= PDebugData(7 downto 4) WHEN (PConfID = "11") ELSE
                "0000";

-- Assertions test expressions assignations
CurS0       <= TRUE WHEN DUTCStateV = "000" ELSE
                FALSE;
CurS1       <= TRUE WHEN DUTCStateV = "001" ELSE
                FALSE;
CurS2       <= TRUE WHEN DUTCStateV = "010" ELSE
                FALSE;
CurS3       <= TRUE WHEN DUTCStateV = "011" ELSE
                FALSE;
CurS3a      <= TRUE WHEN DUTCStateV = "100" ELSE
                FALSE;
CurS4       <= TRUE WHEN DUTCStateV = "101" ELSE
                FALSE;
CurS4a      <= TRUE WHEN DUTCStateV = "110" ELSE
                FALSE;
CurS5       <= TRUE WHEN DUTCStateV = "111" ELSE
                FALSE;
CurS0ALE0   <= TRUE WHEN ((PConfID = "00") OR (PConfID = "10") OR
                          (PConfID = "11")) AND
                          (DUTCStateV = "000") AND (DUTALE = '0') ELSE
                FALSE;
CurS0ALE1   <= TRUE WHEN ((PConfID = "00") OR (PConfID = "10") OR
                          (PConfID = "11")) AND
                          (DUTCStateV = "000") AND (DUTALE = '1') ELSE
                FALSE;
CurS2We0Re0 <= TRUE WHEN (PConfID = "01") AND
                          (DUTCStateV = "010") AND (DUTWem = '0') AND
                          (DUTRem = '0') ELSE
                FALSE;
CurS2We1    <= TRUE WHEN (PConfID = "01") AND
                          (DUTCStateV = "010") AND (DUTWem = '1') ELSE
                FALSE;
CurS2We0Re1 <= TRUE WHEN (PConfID = "01") AND
                          (DUTCStateV = "010") AND (DUTWem = '0') AND
                          (DUTRem = '1') ELSE
                FALSE;
CurS3IOW1   <= TRUE WHEN ((PConfID = "01") OR (PConfID = "10")) AND
                          (DUTCStateV = "011") AND (DUTIOW = '1') ELSE
                FALSE;
CurS3IOW0   <= TRUE WHEN ((PConfID = "01") OR (PConfID = "10")) AND
                          (DUTCStateV = "011") AND (DUTIOW = '0') ELSE
                FALSE;
CurS4IOR1   <= TRUE WHEN ((PConfID = "01") OR (PConfID = "11")) AND
                          (DUTCStateV = "101") AND (DUTIOR = '1') ELSE
                FALSE;
CurS4IOR0   <= TRUE WHEN ((PConfID = "01") OR (PConfID = "11")) AND
                          (DUTCStateV = "101") AND (DUTIOR = '0') ELSE
                FALSE;
CurAld0     <= TRUE WHEN DUTAld = '0' ELSE
                FALSE;
CurAld1     <= TRUE WHEN DUTAld = '1' ELSE
                FALSE;
CurDld0     <= TRUE WHEN DUTDld = '0' ELSE
                FALSE;
CurDld1     <= TRUE WHEN DUTDld = '1' ELSE
                FALSE;
CurDst0     <= TRUE WHEN DUTDst = '0' ELSE

                FALSE;
CurDst1     <= TRUE WHEN DUTDst = '1' ELSE
                FALSE;
CurS2Adr10  <= TRUE WHEN ((PConfID = "00") OR (PConfID = "01")) AND
```

ANNEXE H : Sources Assertions Bus ISA

```
                        (DUTCStateV = "010") AND (DUTAddr = "10100") ELSE
                FALSE;
CurS2Adr11  <= TRUE WHEN ((PConfID = "00") OR (PConfID = "01")) AND
                        (DUTCStateV = "010") AND (DUTAddr = "10101") ELSE
                FALSE;
CurS2Adr00  <= TRUE WHEN ((PConfID = "00") OR (PConfID = "01")) AND
                        (DUTCStateV = "010") AND
                        (NOT(DUTAddr(4 downto 1) = "1010")) ELSE
                FALSE;
CurWem1Rem0 <= TRUE WHEN (PConfID = "01") AND
                        (DUTWem = '1') AND (DUTRem = '0') ELSE
                FALSE;
CurWem0Rem1 <= TRUE WHEN (PConfID = "01") AND
                        (DUTWem = '0') AND (DUTRem = '1') ELSE
                FALSE;
CurWem0Rem0 <= TRUE WHEN (PConfID = "01") AND
                        (DUTWem = '0') AND (DUTRem = '0') ELSE
                FALSE;
CurDinReg2  <= TRUE WHEN DUTLstDin = DUTreg2addr ELSE
                FALSE;
CurAdd2Dout <= TRUE WHEN DUTLstad2rg = DUTDout ELSE
                FALSE;


-- Assertions expressions pipelining
ASSERTEXPPIPE: PROCESS (clk, reset_n)
BEGIN
  IF reset_n = '0' THEN
    LstS0       <= FALSE;
    LstS1       <= FALSE;
    LstS2       <= FALSE;
    LstS3       <= FALSE;
    LstS3a      <= FALSE;
    LstS4       <= FALSE;
    LstS4a      <= FALSE;
    LstS0ALE0   <= FALSE;
    LstS0ALE1   <= FALSE;
    LstS2We0Re0 <= FALSE;
    LstS2We1    <= FALSE;
    LstS2We0Re1 <= FALSE;
    LstS3IOW1   <= FALSE;
    LstS3IOW0   <= FALSE;
    LstS4IOR1   <= FALSE;
    LstS4IOR0   <= FALSE;
    DUTLstDin   <= "0000";
    DUTLstad2rg <= "0000";

  ELSIF clk'event AND clk = '1' THEN
    LstS0       <= CurS0;
    LstS1       <= CurS1;
    LstS2       <= CurS2;
    LstS3       <= CurS3;
    LstS3a      <= CurS3a;
    LstS4       <= CurS4;
    LstS4a      <= CurS4a;
    LstS0ALE0   <= CurS0ALE0;
    LstS0ALE1   <= CurS0ALE1;
    LstS2We0Re0 <= CurS2We0Re0;
    LstS2We1    <= CurS2We1;
    LstS2We0Re1 <= CurS2We0Re1;
    LstS3IOW1   <= CurS3IOW1;
    LstS3IOW0   <= CurS3IOW0;
    LstS4IOR1   <= CurS4IOR1;
    LstS4IOR0   <= CurS4IOR0;
    IF (CurS3a = TRUE) THEN
      DUTLstDin   <= DUTDin;
    END IF;
    IF (CurS4a = TRUE) THEN
      DUTLstad2rg <= DUTaddr2reg;
    END IF;
```

ANNEXE H : Sources Assertions Bus ISA

```
  END IF;
END PROCESS;


-- Event generators expressions assignations
EvtACStateM <= TRUE WHEN ACstate(12 downto 0) = "1111111111111" ELSE
               FALSE;
EvtACAlDlDs <= TRUE WHEN ACstate(33 downto 13) = "11111111111111111111" ELSE
               FALSE;
EvtACAdDat  <= TRUE WHEN ACstate(38 downto 34) = "11111" ELSE
               FALSE;
EvtACFail   <= TRUE WHEN (EvtACStateM = TRUE) AND
                         (EvtACAlDlDs = TRUE) AND
                         (EvtACAdDat = TRUE) ELSE
               FALSE;



ACEGconfid  <= PConfId;
ACEGdbgdata <= PDebugData;
ARAMdbgdata <= ACEGramdata;
ARAMegevent <= EGstate;

ACEGramdata((P+Cwid) downto P) <= PConfId;

ACEGramdata((P-1) downto 0) <= PDebugData;
ACEGramdata(31 downto (32-nEG)) <= EGstate;
ACEGramdata((31-nEG) downto (P+Cwid+1)) <= (others => '0');


-- Assertion checkers instanciations
ACS0ALE0S0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS0ALE0,
                consequent_expr=>CurS0,
                valide=>ACvalid(0));

ACS0ALE1S1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS0ALE1,
                consequent_expr=>CurS1,
                valide=>ACvalid(1));

ACS1S2: assert_implication
      generic map(0)

      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS1,
                consequent_expr=>CurS2,
                valide=>ACvalid(2));

ACS2WEM0REM0S2: assert_implication
      generic map(0)

      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS2We0Re0,
                consequent_expr=>CurS2,
                valide=>ACvalid(3));

ACS2WEM1S3: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS2We1,
```

ANNEXE H : Sources Assertions Bus ISA

```
                consequent_expr=>CurS3,
                valide=>ACvalid(4));

ACS2REM1S4: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS2We0Re1,
                consequent_expr=>CurS4,
                valide=>ACvalid(5));

ACS3IOW1S3: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS3IOW1,
                consequent_expr=>CurS3,
                valide=>ACvalid(6));

ACS3IOW0S3A: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS3IOW0,
                consequent_expr=>CurS3a,
                valide=>ACvalid(7));

ACS3AS0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS3a,
                consequent_expr=>CurS0,
                valide=>ACvalid(8));

ACS4IOR1S4: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS4IOR1,
                consequent_expr=>CurS4,
                valide=>ACvalid(9));

ACS4IOR0S4A: assert_implication
      generic map(0)

      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS4IOR0,
                consequent_expr=>CurS4a,
                valide=>ACvalid(10));

ACS4AS0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>LstS4a,
                consequent_expr=>CurS0,
                valide=>ACvalid(11));

ACNOS5: assert_never
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>CurS5,
                valide=>ACvalid(12));

ACS0ALD0: assert_implication
      generic map(0)
      port map (clk=>clk,
```

ANNEXE H : Sources Assertions Bus ISA

```
                reset_n=>reset_n,
                antecedent_expr=>CurS0,
                consequent_expr=>CurAld0,
                valide=>ACvalid(13));

ACS0DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS0,
                consequent_expr=>CurDld0,
                valide=>ACvalid(14));

ACS0DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS0,
                consequent_expr=>CurDst0,
                valide=>ACvalid(15));

ACS1ALD1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS1,
                consequent_expr=>CurAld1,
                valide=>ACvalid(16));

ACS1DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS1,
                consequent_expr=>CurDld0,
                valide=>ACvalid(17));

ACS1DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS1,
                consequent_expr=>CurDst0,
                valide=>ACvalid(18));

ACS2ALD1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2,
                consequent_expr=>CurAld1,
                valide=>ACvalid(19));

ACS2DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2,
                consequent_expr=>CurDld0,
                valide=>ACvalid(20));

ACS2DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2,
                consequent_expr=>CurDst0,
                valide=>ACvalid(21));

ACS3ALD0: assert_implication
```

ANNEXE H : Sources Assertions Bus ISA

```
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3,
                consequent_expr=>CurAld0,
                valide=>ACvalid(22));

ACS3DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3,
                consequent_expr=>CurDld0,
                valide=>ACvalid(23));

ACS3DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3,
                consequent_expr=>CurDst0,
                valide=>ACvalid(24));

ACS3aALD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3a,
                consequent_expr=>CurAld0,
                valide=>ACvalid(25));

ACS3aDLD1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3a,
                consequent_expr=>CurDld1,
                valide=>ACvalid(26));

ACS3aDST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3a,
                consequent_expr=>CurDst0,
                valide=>ACvalid(27));

ACS4ALD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4,
                consequent_expr=>CurAld0,
                valide=>ACvalid(28));

ACS4DLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4,

                consequent_expr=>CurDld0,
                valide=>ACvalid(29));

ACS4DST0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4,
                consequent_expr=>CurDst0,
```

ANNEXE H : Sources Assertions Bus ISA

```
                valide=>ACvalid(30));

ACS4aALD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4a,
                consequent_expr=>CurAld0,
                valide=>ACvalid(31));

ACS4aDLD0: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4a,
                consequent_expr=>CurDld0,
                valide=>ACvalid(32));

ACS4aDST1: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4a,
                consequent_expr=>CurDst1,
                valide=>ACvalid(33));

ACS2ADDR10: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2Adr10,
                consequent_expr=>CurWem1Rem0,
                valide=>ACvalid(34));

ACS2ADDR11: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2Adr11,
                consequent_expr=>CurWem0Rem1,
                valide=>ACvalid(35));

ACS2ADDR00: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2Adr00,
                consequent_expr=>CurWem0Rem0,
                valide=>ACvalid(36));

ACS3aDINREG2ADDR: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS3a,
                consequent_expr=>CurDinReg2,
                valide=>ACvalid(37));

ACS4aADDR2REGDOUT: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS4a,
                consequent_expr=>CurAdd2Dout,
                valide=>ACvalid(38));


ACHARNESS: for i in (nAC-1) downto 0 generate
  HARN: ac_harness
        generic map (32,0,0)
```

ANNEXE H : Sources Assertions Bus ISA

```
        port map(clk=>clk,
                 reset_n=>ACreset(i),
                 clrfail=>'0',
                 clrinforeg=>'0',
                 ACenable=>'1',
                 dbginput=>TTin,
                 validin=>ACvalid(i),
                 dbginfo=>ACdbginfo((((i+1)*dbgbuswidth)-1) downto (i*dbgbuswidth)),
                 failed_n=>ACstate(i)
        );
end generate;

ACCTRL: ac_ctrl
  port map (clk=>clk,
            reset_n=>reset_n,
            we_n=>ACCwe_n,
            out_addr=>ACCout_addr,
            rdat_type=>ACCdat_type,
            statein=>ACstate,
            ttin=>ACdbginfo,
            resetin=>ACCresetin,
            datar=>ACCdatar,
            resetout=>ACreset
  );



-- Event generators instanciation
EGACSTATE: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACStateM,
                valide=>EGvalid(0));

EGACALDDLDDST: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACAlDlDs,
                valide=>EGvalid(1));

EGACADDRDATA: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACAdDat,
                valide=>EGvalid(2));

EGACFAIL: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACFail,
                valide=>EGvalid(3));


EGHARNESS: for i in (nEG-1) downto 0 generate
  HARN: ac_harness
        generic map (32,0,0)
        port map(clk=>clk,
                 reset_n=>EGreset(i),
                 clrfail=>'0',
                 clrinforeg=>'0',
                 ACenable=>'1',
                 dbginput=>TTin,
                 validin=>EGvalid(i),
                 dbginfo=>EGdbginfo((((i+1)*dbgbuswidth)-1) downto (i*dbgbuswidth)),
                 failed_n=>EGstate(i)
        );
```

ANNEXE H : Sources Assertions Bus ISA

```
end generate;

EGCTRL: eg_ctrl
  port map (clk=>clk,
            reset_n=>reset_n,
            we_n=>EGCwe_n,
            out_addr=>EGCout_addr,
            rdat_type=>EGCdat_type,
            transdest=>EGCtransdes,
            clrirq=>EGCclrirq,
            statein=>EGstate,
            ttin=>EGdbginfo,
            cfgin=>EGCcfgin,
            datar=>EGCdatar,
            resetout=>EGreset,
            irq=>EGCirq

  );


CMcfgswap    <= ACEGcfgswap;
CMswapid     <= ACEGcfgnum;


CFGSWAPCTRL: PROCESS(clk,reset_n,PDebugData)  -- Project 2, Version 1, Revision 0
BEGIN
  IF reset_n = '0' THEN
    ACEGcfgswap <= '1';
    ACEGcfgnum  <= "00";
  ELSIF clk'event AND clk = '1' THEN
    ACEGcfgswap <= '1';
    CASE ACEGcfgnum IS
      WHEN "00" => -- CFG0
        IF (DUTCStateV = "000") AND (DUTALE = '1') THEN
          ACEGcfgnum <= "01";
        ELSE
          ACEGcfgnum <= "00";
        END IF;

      WHEN "01" => -- CFG1
        IF (DUTCStateV = "010") AND (DUTWem = '1') THEN
          ACEGcfgnum <= "10";

        ELSIF (DUTCStateV = "010") AND (DUTRem = '1') THEN
          ACEGcfgnum <= "11";
        ELSE
          ACEGcfgnum <= "01";
        END IF;

      WHEN "10" => -- CFG2
        IF (DUTCStateV = "000") AND (DUTALE = '1') THEN
          ACEGcfgnum <= "01";
        ELSIF (DUTCStateV = "011") AND (DUTIOW = '0') THEN
          ACEGcfgnum <= "00";
        ELSE
          ACEGcfgnum <= "10";
        END IF;

      WHEN "11" => -- CFG3
        IF (DUTCStateV = "000") AND (DUTALE = '1') THEN
          ACEGcfgnum <= "01";
        ELSIF (DUTCStateV = "101") AND (DUTIOR = '0') THEN
          ACEGcfgnum <= "00";
        ELSE
          ACEGcfgnum <= "11";
        END IF;

      WHEN OTHERS =>
        ACEGcfgnum <= "00";
```

ANNEXE H : Sources Assertions Bus ISA

```
      END CASE;
    END IF;
  END PROCESS;
END rtl;
```

ANNEXE H : Sources Assertions Bus ISA

```
------------------------------------------------------------
-- Description : Assertion Checkers + Event Generators
--               module
--               Synthesized AC and EG specific to ISA
--               bus interface + OCHP design
--
-- File   : ac_eg_module_ISA_ochp_set2.vhd
-- Author : K. Peterson
-- Created: Oct 31 2004
-- Modif. : Oct 31 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
use work.acftypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY ac_eg_module IS
  PORT (clk          : IN  std_logic;
        reset_n      : IN  std_logic;

        -- OCHP debug signals input
        PConfId      : IN  std_logic_vector(Cwid downto 0);
        PDebugData   : IN  std_logic_vector((P-1) downto 0);

        -- AC controller signals
        ACCwe_n      : IN  std_logic;
        ACCout_addr  : IN  std_logic_vector(nACwid downto 0);
        ACCdat_type  : IN  std_logic;
        ACCresetin   : IN  std_logic_vector(31 downto 0);
        ACCdatar     : OUT std_logic_vector((dbgbuswidth-1) downto 0);

        -- EG controller signals
        EGCwe_n      : IN  std_logic;
        EGCout_addr  : IN  std_logic_vector(nEGwid downto 0);
        EGCdat_type  : IN  std_logic;
        EGCtransdes  : IN  std_logic;
        EGCclrirq    : IN  std_logic;
        EGCcfgin     : IN  std_logic_vector(31 downto 0);
        EGCdatar     : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        EGCirq       : OUT std_logic;

        -- Timestamp Timer signals
        TTin         : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Assertion RAM signals
        ARAMdbgdata  : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        ARAMegevent  : OUT std_logic_vector((nEG-1) downto 0);

        -- Configuration Manager signals
        CMcfgswap    : OUT std_logic;
        CMswapid     : OUT std_logic_vector(Cwid downto 0)
  );

END ac_eg_module;

ARCHITECTURE rtl OF ac_eg_module IS

    component assert_always
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
```

ANNEXE H : Sources Assertions Bus ISA

```
        port (
          clk, reset_n : IN std_ulogic;
          test_expr    : IN boolean;
          valide       : OUT std_ulogic);
    end component;

    component assert_implication
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
        port (
          clk, reset_n                 : IN std_ulogic;
          antecedent_expr, consequent_expr : IN boolean;
          valide                       : OUT std_ulogic);
    end component;


  COMPONENT ac_harness
    GENERIC (
      dbginfowidth   : integer   := 32;
      dbginfotype    : integer   := 0;  -- Default: Timestamp register
      failedlatched  : integer   := 0   -- Default: Failed not latched
    );
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- Assertion checker control signals
          clrfail    : IN  std_logic;
          clrinforeg : IN  std_logic;
          ACenable   : IN  std_logic;

          -- Assertion checker debug data input
          dbginput   : IN  std_logic_vector((dbginfowidth-1) downto 0);
          validin    : IN  std_logic;

          -- Assertion checker debug data output
          dbginfo    : OUT std_logic_vector((dbginfowidth-1) downto 0);
          failed_n   : OUT std_logic);
  END COMPONENT;

  COMPONENT ac_ctrl
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- AC control signals
          we_n       : IN  std_logic;
          out_addr   : IN  std_logic_vector(nACwid downto 0);
          rdat_type  : IN  std_logic;

          -- AC debug data input
          statein    : IN  std_logic_vector((nAC-1) downto 0);
          ttin       : IN  std_logic_vector(((dbgbuswidth*nAC)-1) downto 0);
          resetin    : IN  std_logic_vector(31 downto 0);

          -- AC debug data output
          datar      : OUT std_logic_vector((dbgbuswidth-1) downto 0);
          resetout   : OUT std_logic_vector((nAC-1) downto 0)
    );
  end COMPONENT;

  COMPONENT eg_ctrl
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- EG control signals
          we_n       : IN  std_logic;
          out_addr   : IN  std_logic_vector(nEGwid downto 0);
```

ANNEXE H : Sources Assertions Bus ISA

```
        rdat_type   : IN  std_logic;
        transdest   : IN  std_logic;
        clrirq      : IN  std_logic;

        -- EG debug data input
        statein     : IN  std_logic_vector((nEG-1) downto 0);
        ttin        : IN  std_logic_vector(((dbgbuswidth*nEG)-1) downto 0);
        cfgin       : IN  std_logic_vector(31 downto 0);

        -- EG debug data output
        datar       : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        resetout    : OUT std_logic_vector((nEG-1) downto 0);
        irq         : OUT std_logic
    );
  end COMPONENT;

  SIGNAL ACEGconfid  : std_logic_vector(Cwid downto 0);
  SIGNAL ACEGdbgdata : std_logic_vector((P-1) downto 0);
  SIGNAL ACstate     : std_logic_vector((nAC-1) downto 0);
  SIGNAL ACdbginfo   : std_logic_vector(((dbgbuswidth*nAC)-1) downto 0);
  SIGNAL ACreset     : std_logic_vector((nAC-1) downto 0);
  SIGNAL EGstate     : std_logic_vector((nEG-1) downto 0);

  SIGNAL EGdbginfo   : std_logic_vector(((dbgbuswidth*nEG)-1) downto 0);
  SIGNAL EGreset     : std_logic_vector((nEG-1) downto 0);
  SIGNAL ACEGramdata : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL ACEGcfgswap : std_logic;
  SIGNAL ACEGcfgnum  : std_logic_vector(Cwid downto 0);
  SIGNAL ACvalid     : std_logic_vector((nAC-1) downto 0);
  SIGNAL EGvalid     : std_logic_vector((nEG-1) downto 0);

  -- DUT Debug signals
  SIGNAL DUTCStateV  : std_logic_vector(2 downto 0);
  SIGNAL DUTAld      : std_logic;
  SIGNAL DUTALE      : std_logic;
  SIGNAL DUTAddr     : std_logic_vector(4 downto 0);
  SIGNAL DUTWem      : std_logic;
  SIGNAL DUTRem      : std_logic;
  SIGNAL DUTIOR      : std_logic;
  SIGNAL DUTreg2addr : std_logic_vector(7 downto 0);
  SIGNAL DUTaddr2reg : std_logic_vector(7 downto 0);

  -- Assertions expressions signals
  SIGNAL CurCfg3     : boolean;
  SIGNAL CurS2Adr10  : boolean;
  SIGNAL CurS2Adr11  : boolean;
  SIGNAL CurS2Adr00  : boolean;
  SIGNAL CurWem1Rem0 : boolean;
  SIGNAL CurWem0Rem1 : boolean;
  SIGNAL CurWem0Rem0 : boolean;
  SIGNAL CurRg2AdAd2Rg : boolean;

  -- Event generators expressions signals
  SIGNAL EvtACAdr10  : boolean;
  SIGNAL EvtACAdr11  : boolean;
  SIGNAL EvtACAdr00  : boolean;

  SIGNAL EvtACRg2Ad  : boolean;

BEGIN

  -- Debug signals assignations
  DUTCStateV  <= PDebugData(15 downto 13);
  DUTAld      <= PDebugData(12) WHEN (PConfID = "01") ELSE
                 '0';
  DUTALE      <= PDebugData(12) WHEN (PConfID = "00") ELSE
                 '0';
  DUTAddr     <= PDebugData(4 downto 0) WHEN (PConfID(1) = '0') ELSE
                 "00000";
  DUTWem      <= PDebugData(11) WHEN not(PConfID = "11") ELSE
```

ANNEXE H : Sources Assertions Bus ISA

```
                '0';
DUTRem      <= PDebugData(10) WHEN not(PConfID = "11") ELSE
                '0';
DUTIOR      <= PDebugData(9) WHEN (PConfID = "01") OR (PConfID = "10") ELSE
                '1';
DUTreg2addr <= PDebugData(15 downto 8) WHEN (PConfID = "11") ELSE
                "00000000";
DUTaddr2reg <= PDebugData(7 downto 0) WHEN (PConfID = "11") ELSE
                "00000000";

-- Assertions test expressions assignations
CurCfg3     <= TRUE WHEN (PConfID = "11") ELSE
                FALSE;
CurS2Adr10  <= TRUE WHEN ((PConfID = "00") OR (PConfID = "01")) AND
                         (DUTCStateV = "010") AND (DUTAddr = "10100") ELSE
                FALSE;
CurS2Adr11  <= TRUE WHEN ((PConfID = "00") OR (PConfID = "01")) AND
                         (DUTCStateV = "010") AND (DUTAddr = "10101") ELSE
                FALSE;
CurS2Adr00  <= TRUE WHEN ((PConfID = "00") OR (PConfID = "01")) AND
                         (DUTCStateV = "010") AND
                         (NOT(DUTAddr(4 downto 1) = "1010")) ELSE
                FALSE;
CurWem1Rem0 <= TRUE WHEN not(PConfID = "11") AND
                         (DUTWem = '1') AND (DUTRem = '0') ELSE
                FALSE;
CurWem0Rem1 <= TRUE WHEN not(PConfID = "11") AND
                         (DUTWem = '0') AND (DUTRem = '1') ELSE
                FALSE;
CurWem0Rem0 <= TRUE WHEN not(PConfID = "11") AND
                         (DUTWem = '0') AND (DUTRem = '0') ELSE
                FALSE;
CurRg2AdAd2Rg <= TRUE WHEN DUTaddr2reg = (DUTreg2addr+1) ELSE
                FALSE;



-- Event generators expressions assignations
EvtACAdr10  <= TRUE WHEN ACstate(0) = '1' ELSE
                FALSE;
EvtACAdr11  <= TRUE WHEN ACstate(1) = '1' ELSE
                FALSE;
EvtACAdr00  <= TRUE WHEN ACstate(2) = '1' ELSE
                FALSE;
EvtACRg2Ad  <= TRUE WHEN ACstate(3) = '1' ELSE
                FALSE;



ACEGconfid  <= PConfId;
ACEGdbgdata <= PDebugData;
ARAMdbgdata <= ACEGramdata;
ARAMegevent <= EGstate;

ACEGramdata((P+Cwid) downto P) <= PConfId;
ACEGramdata((P-1) downto 0) <= PDebugData;
ACEGramdata(31 downto (32-nEG)) <= EGstate;
ACEGramdata((31-nEG) downto (P+Cwid+1)) <= (others => '0');


-- Assertion checkers instanciations
ACS2ADDR10: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2Adr10,
                consequent_expr=>CurWem1Rem0,
                valide=>ACvalid(0));

ACS2ADDR11: assert_implication
```

ANNEXE H : Sources Assertions Bus ISA

```
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2Adr11,
                consequent_expr=>CurWem0Rem1,
                valide=>ACvalid(1));

ACS2ADDR00: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurS2Adr00,
                consequent_expr=>CurWem0Rem0,
                valide=>ACvalid(2));

ACS4aRG2ADAD2RG: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurCfg3,
                consequent_expr=>CurRg2AdAd2Rg,
                valide=>ACvalid(3));



ACHARNESS: for i in (nAC-1) downto 0 generate
  HARN: ac_harness
        generic map (32,0,0)
        port map(clk=>clk,
                 reset_n=>ACreset(i),
                 clrfail=>'0',
                 clrinforeg=>'0',
                 ACenable=>'1',
                 dbginput=>TTin,
                 validin=>ACvalid(i),
                 dbginfo=>ACdbginfo((((i+1)*dbgbuswidth)-1) downto (i*dbgbuswidth)),
                 failed_n=>ACstate(i)
        );
end generate;

ACCTRL: ac_ctrl
  port map (clk=>clk,
            reset_n=>reset_n,
            we_n=>ACCwe_n,
            out_addr=>ACCout_addr,
            rdat_type=>ACCdat_type,
            statein=>ACstate,
            ttin=>ACdbginfo,
            resetin=>ACCresetin,
            datar=>ACCdatar,
            resetout=>ACreset
  );


-- Event generators instanciation
EGACSTATE: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACAdr10,
                valide=>EGvalid(0));

EGACALDDLDDST: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACAdr11,
                valide=>EGvalid(1));

EGACADDRDATA: assert_always
```

ANNEXE H : Sources Assertions Bus ISA

```
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACAdr00,
                valide=>EGvalid(2));

EGACFAIL: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACRg2Ad,
                valide=>EGvalid(3));


EGHARNESS: for i in (nEG-1) downto 0 generate
  HARN: ac_harness
        generic map (32,0,0)
        port map(clk=>clk,
                 reset_n=>EGreset(i),
                 clrfail=>'0',
                 clrinforeg=>'0',
                 ACenable=>'1',
                 dbginput=>TTin,
                 validin=>EGvalid(i),
                 dbginfo=>EGdbginfo((((i+1)*dbgbuswidth)-1) downto (i*dbgbuswidth)),
                 failed_n=>EGstate(i)
        );
end generate;

EGCTRL: eg_ctrl
  port map (clk=>clk,
            reset_n=>reset_n,
            we_n=>EGCwe_n,
            out_addr=>EGCout_addr,
            rdat_type=>EGCdat_type,
            transdest=>EGCtransdes,
            clrirq=>EGCclrirq,
            statein=>EGstate,
            ttin=>EGdbginfo,
            cfgin=>EGCcfgin,
            datar=>EGCdatar,
            resetout=>EGreset,
            irq=>EGCirq

  );


CMcfgswap   <= ACEGcfgswap;
CMswapid    <= ACEGcfgnum;


CFGSWAPCTRL: PROCESS(clk,reset_n,PDebugData)  -- Project 2, Version 1, Revision 0
BEGIN
  IF reset_n = '0' THEN
    ACEGcfgswap <= '1';
    ACEGcfgnum  <= "00";
  ELSIF clk'event AND clk = '1' THEN
    ACEGcfgswap <= '1';
    CASE ACEGcfgnum IS
      WHEN "00" => -- CFG0
        IF (DUTCStateV = "000") AND (DUTALE = '1') THEN
          ACEGcfgnum <= "10";
        ELSE
          ACEGcfgnum <= "00";
        END IF;

      WHEN "01" => -- CFG1
        IF (DUTCStateV = "001") AND (DUTAld = '1') THEN
          ACEGcfgnum <= "10";
        ELSIF (DUTCStateV = "101") AND (DUTIOR = '0') THEN
```

ANNEXE H : Sources Assertions Bus ISA

```
          ACEGcfgnum <= "11";
        ELSIF (DUTCStateV = "011") THEN
          ACEGcfgnum <= "00";
        ELSE
          ACEGcfgnum <= "01";
        END IF;

      WHEN "10" => -- CFG2
        IF (DUTCStateV = "101") AND (DUTIOR = '0') THEN
          ACEGcfgnum <= "11";
        ELSIF (DUTCStateV = "011") THEN
          ACEGcfgnum <= "00";
        ELSE
          ACEGcfgnum <= "10";
        END IF;

      WHEN "11" => -- CFG3
        ACEGcfgnum <= "01";

      WHEN OTHERS =>
        ACEGcfgnum <= "00";

    END CASE;
  END IF;
END PROCESS;
END rtl;
```

ANNEXE H : Sources Assertions Bus ISA

```
------------------------------------------------------------
-- Description : Assertion Checkers + Event Generators
--               module
--               Synthesized AC and EG specific to ISA
--               bus interface + OCHP design
--
-- File   : ac_eg_module_ISA_ochp_set3.vhd
-- Author : K. Peterson
-- Created: Oct 31 2004
-- Modif. : Oct 31 2004
--
-- Projet : Assertion-Based Runtime Debugger
-- Module : Assertion-Checker FPGA
------------------------------------------------------------

LIBRARY ieee;
use work.ochptypes.all;
use work.acftypes.all;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY ac_eg_module IS
  PORT (clk          : IN  std_logic;
        reset_n      : IN  std_logic;

        -- OCHP debug signals input
        PConfId      : IN  std_logic_vector(Cwid downto 0);
        PDebugData   : IN  std_logic_vector((P-1) downto 0);

        -- AC controller signals
        ACCwe_n      : IN  std_logic;
        ACCout_addr  : IN  std_logic_vector(nACwid downto 0);
        ACCdat_type  : IN  std_logic;
        ACCresetin   : IN  std_logic_vector(31 downto 0);
        ACCdatar     : OUT std_logic_vector((dbgbuswidth-1) downto 0);

        -- EG controller signals
        EGCwe_n      : IN  std_logic;
        EGCout_addr  : IN  std_logic_vector(nEGwid downto 0);
        EGCdat_type  : IN  std_logic;
        EGCtransdes  : IN  std_logic;
        EGCclrirq    : IN  std_logic;
        EGCcfgin     : IN  std_logic_vector(31 downto 0);
        EGCdatar     : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        EGCirq       : OUT std_logic;

        -- Timestamp Timer signals
        TTin         : IN  std_logic_vector((dbgbuswidth-1) downto 0);

        -- Assertion RAM signals
        ARAMdbgdata  : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        ARAMegevent  : OUT std_logic_vector((nEG-1) downto 0);

        -- Configuration Manager signals
        CMcfgswap    : OUT std_logic;
        CMswapid     : OUT std_logic_vector(Cwid downto 0)
  );

END ac_eg_module;

ARCHITECTURE rtl OF ac_eg_module IS

    component assert_always
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
```

ANNEXE H : Sources Assertions Bus ISA

```
        port (
          clk, reset_n : IN std_ulogic;
          test_expr    : IN boolean;
          valide       : OUT std_ulogic);
    end component;

    component assert_implication
        generic (
--          severity_lvl : integer;
          options      : integer
--          ;
--          msg          : string
        );
        port (
          clk, reset_n                : IN std_ulogic;
          antecedent_expr, consequent_expr : IN boolean;
          valide                      : OUT std_ulogic);
    end component;


  COMPONENT ac_harness
    GENERIC (
      dbginfowidth   : integer   := 32;
      dbginfotype    : integer   := 0;  -- Default: Timestamp register
      failedlatched  : integer   := 0   -- Default: Failed not latched
    );
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- Assertion checker control signals
          clrfail    : IN  std_logic;
          clrinforeg : IN  std_logic;
          ACenable   : IN  std_logic;

          -- Assertion checker debug data input
          dbginput   : IN  std_logic_vector((dbginfowidth-1) downto 0);
          validin    : IN  std_logic;

          -- Assertion checker debug data output
          dbginfo    : OUT std_logic_vector((dbginfowidth-1) downto 0);
          failed_n   : OUT std_logic);
  END COMPONENT;

  COMPONENT ac_ctrl
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- AC control signals
          we_n       : IN  std_logic;
          out_addr   : IN  std_logic_vector(nACwid downto 0);
          rdat_type  : IN  std_logic;

          -- AC debug data input
          statein    : IN  std_logic_vector((nAC-1) downto 0);
          ttin       : IN  std_logic_vector(((dbgbuswidth*nAC)-1) downto 0);
          resetin    : IN  std_logic_vector(31 downto 0);

          -- AC debug data output
          datar      : OUT std_logic_vector((dbgbuswidth-1) downto 0);
          resetout   : OUT std_logic_vector((nAC-1) downto 0)
    );
  end COMPONENT;

  COMPONENT eg_ctrl
    PORT (clk        : IN  std_logic;
          reset_n    : IN  std_logic;

          -- EG control signals
          we_n       : IN  std_logic;
          out_addr   : IN  std_logic_vector(nEGwid downto 0);
```

ANNEXE H : Sources Assertions Bus ISA

```
        rdat_type   : IN  std_logic;
        transdest   : IN  std_logic;
        clrirq      : IN  std_logic;

        -- EG debug data input
        statein     : IN  std_logic_vector((nEG-1) downto 0);
        ttin        : IN  std_logic_vector(((dbgbuswidth*nEG)-1) downto 0);
        cfgin       : IN  std_logic_vector(31 downto 0);

        -- EG debug data output
        datar       : OUT std_logic_vector((dbgbuswidth-1) downto 0);
        resetout    : OUT std_logic_vector((nEG-1) downto 0);
        irq         : OUT std_logic
    );
  end COMPONENT;

  SIGNAL ACEGconfid  : std_logic_vector(Cwid downto 0);
  SIGNAL ACEGdbgdata : std_logic_vector((P-1) downto 0);
  SIGNAL ACstate     : std_logic_vector((nAC-1) downto 0);
  SIGNAL ACdbginfo   : std_logic_vector(((dbgbuswidth*nAC)-1) downto 0);
  SIGNAL ACreset     : std_logic_vector((nAC-1) downto 0);
  SIGNAL EGstate     : std_logic_vector((nEG-1) downto 0);

  SIGNAL EGdbginfo   : std_logic_vector(((dbgbuswidth*nEG)-1) downto 0);
  SIGNAL EGreset     : std_logic_vector((nEG-1) downto 0);
  SIGNAL ACEGramdata : std_logic_vector((dbgbuswidth-1) downto 0);
  SIGNAL ACEGcfgswap : std_logic;
  SIGNAL ACEGcfgnum  : std_logic_vector(Cwid downto 0);
  SIGNAL ACvalid     : std_logic_vector((nAC-1) downto 0);
  SIGNAL EGvalid     : std_logic_vector((nEG-1) downto 0);

  -- DUT Debug signals
  SIGNAL DUTCStateV  : std_logic_vector(2 downto 0);
  SIGNAL DUTIOW      : std_logic;
  SIGNAL DUTIOR      : std_logic;
  SIGNAL DUTDin      : std_logic_vector(7 downto 0);
  SIGNAL DUTLstDin   : std_logic_vector(7 downto 0);
  SIGNAL DUTreg2addr : std_logic_vector(7 downto 0);
  SIGNAL DUTaddr2reg : std_logic_vector(7 downto 0);
  SIGNAL DUTDout     : std_logic_vector(7 downto 0);

  -- Assertions expressions signals
  SIGNAL CurCfg1     : boolean;
  SIGNAL CurCfg2     : boolean;
  SIGNAL CurDinRg2Ad : boolean;
  SIGNAL CurAd2RgDout: boolean;

  -- Event generators expressions signals
  SIGNAL EvtACDin    : boolean;
  SIGNAL EvtACDout   : boolean;

BEGIN

  -- Debug signals assignations
  DUTCStateV  <= PDebugData(15 downto 13) WHEN (PConfID = "00") ELSE
                 "000";
  DUTIOW      <= PDebugData(12) WHEN (PConfID = "00") ELSE
                 '1';
  DUTIOR      <= PDebugData(11) WHEN (PConfID = "00") ELSE
                 '1';
  DUTDin      <= PDebugData(7 downto 0) WHEN (PConfID = "00") ELSE
                 PDebugData(15 downto 8) WHEN (PConfID = "01") ELSE
                 "00000000";
  DUTreg2addr <= PDebugData(7 downto 0) WHEN (PConfID = "01") ELSE
                 "00000000";
  DUTaddr2reg <= PDebugData(15 downto 8) WHEN (PConfID = "10") ELSE
                 "00000000";
  DUTDout     <= PDebugData(7 downto 0) WHEN (PConfID = "10") ELSE
                 "00000000";
```

ANNEXE H : Sources Assertions Bus ISA

```
-- Assertions test expressions assignations
CurCfg1      <= TRUE WHEN (PConfID = "01") ELSE
                FALSE;
CurCfg2      <= TRUE WHEN (PConfID = "10") ELSE
                FALSE;
CurDinRg2Ad <= TRUE WHEN DUTLstDin = DUTreg2addr ELSE
                FALSE;
CurAd2RgDout <= TRUE WHEN DUTaddr2reg = DUTDout ELSE
                FALSE;

-- Assertions expressions pipelining
ASSERTEXPPIPE: PROCESS (clk, reset_n)
BEGIN
  IF reset_n = '0' THEN
    DUTLstDin   <= "00000000";

  ELSIF clk'event AND clk = '1' THEN
    IF (PConfID = "00") THEN
      DUTLstDin   <= DUTDin;
    END IF;
  END IF;
END PROCESS;



-- Event generators expressions assignations
EvtACDin     <= TRUE WHEN ACstate(0) = '1' ELSE
                FALSE;
EvtACDout    <= TRUE WHEN ACstate(1) = '1' ELSE
                FALSE;



ACEGconfid  <= PConfId;
ACEGdbgdata <= PDebugData;
ARAMdbgdata <= ACEGramdata;
ARAMegevent <= EGstate;

ACEGramdata((P+Cwid) downto P) <= PConfId;
ACEGramdata((P-1) downto 0) <= PDebugData;
ACEGramdata(31 downto (32-nEG)) <= EGstate;
ACEGramdata((31-nEG) downto (P+Cwid+1)) <= (others => '0');


-- Assertion checkers instanciations
ACS2ADDR10: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurCfg1,
                consequent_expr=>CurDinRg2Ad,
                valide=>ACvalid(0));

ACS2ADDR11: assert_implication
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                antecedent_expr=>CurCfg2,
                consequent_expr=>CurAd2RgDout,
                valide=>ACvalid(1));



ACHARNESS: for i in (nAC-1) downto 0 generate
  HARN: ac_harness
        generic map (32,0,0)
        port map(clk=>clk,
                 reset_n=>ACreset(i),
                 clrfail=>'0',
                 clrinforeg=>'0',
```

ANNEXE H : Sources Assertions Bus ISA

```
                ACenable=>'1',
                dbginput=>TTin,
                validin=>ACvalid(i),
                dbginfo=>ACdbginfo((((i+1)*dbgbuswidth)-1) downto (i*dbgbuswidth)),
                failed_n=>ACstate(i)
        );
end generate;

ACCTRL: ac_ctrl
  port map (clk=>clk,
            reset_n=>reset_n,
            we_n=>ACCwe_n,
            out_addr=>ACCout_addr,
            rdat_type=>ACCdat_type,
            statein=>ACstate,
            ttin=>ACdbginfo,
            resetin=>ACCresetin,
            datar=>ACCdatar,
            resetout=>ACreset
  );


-- Event generators instanciation
EGACDIN: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACDin,
                valide=>EGvalid(0));

EGACDOUT: assert_always
      generic map(0)
      port map (clk=>clk,
                reset_n=>reset_n,
                test_expr=>EvtACDout,
                valide=>EGvalid(1));



EGHARNESS: for i in (nEG-1) downto 0 generate
  HARN: ac_harness
        generic map (32,0,0)
        port map(clk=>clk,
                 reset_n=>EGreset(i),
                 clrfail=>'0',
                 clrinforeg=>'0',
                 ACenable=>'1',
                 dbginput=>TTin,
                 validin=>EGvalid(i),
                 dbginfo=>EGdbginfo((((i+1)*dbgbuswidth)-1) downto (i*dbgbuswidth)),
                 failed_n=>EGstate(i)
        );
end generate;

EGCTRL: eg_ctrl
  port map (clk=>clk,
            reset_n=>reset_n,
            we_n=>EGCwe_n,
            out_addr=>EGCout_addr,

            rdat_type=>EGCdat_type,
            transdest=>EGCtransdes,
            clrirq=>EGCclrirq,
            statein=>EGstate,
            ttin=>EGdbginfo,
            cfgin=>EGCcfgin,
            datar=>EGCdatar,
            resetout=>EGreset,
            irq=>EGCirq
```

ANNEXE H : Sources Assertions Bus ISA

```
    );

  CMcfgswap   <= ACEGcfgswap;
  CMswapid    <= ACEGcfgnum;

  CFGSWAPCTRL: PROCESS(clk,reset_n,PDebugData)  -- Project 2, Version 3, Revision 0
  BEGIN
    IF reset_n = '0' THEN
      ACEGcfgswap <= '1';
      ACEGcfgnum  <= "00";
    ELSIF clk'event AND clk = '1' THEN
      ACEGcfgswap <= '1';
      CASE ACEGcfgnum IS
        WHEN "00" => -- CFG0
          IF (DUTCStateV = "011") AND (DUTIOW = '0') THEN
            ACEGcfgnum <= "01";
          ELSIF (DUTCStateV = "101") AND (DUTIOR = '0') THEN
            ACEGcfgnum <= "10";
          ELSE
            ACEGcfgnum <= "00";
          END IF;

        WHEN "01" => -- CFG1
          ACEGcfgnum <= "00";

        WHEN "10" => -- CFG2
          ACEGcfgnum <= "00";

        WHEN OTHERS =>
          ACEGcfgnum <= "00";

      END CASE;
    END IF;
  END PROCESS;
END rtl;
```

ANNEXE H : Sources Assertions Bus ISA

```
------------------------------------------------------------
-- Description : Assertion Checker FPGA      parameters file
--               Customized for ISA bus interface
--
-- File    : acf_param_ISA_ochp_set1.vhd
-- Author : K. Peterson
-- Created: Oct 29 2004
-- Modif. : Oct 31 2004
--
-- Proj    : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

package ochptypes is
  constant C:         integer := 4;     -- Number of configurations
  constant Cwid:      integer := 1;     -- log2C - 1
  constant N:         integer := 62;    -- Number of internal signals
  constant Nwid:      integer := 5;     -- log2N - 1
  constant P:         integer := 16;    -- Number of external pins
  constant Pwid:      integer := 3;     -- log2P - 1
end ochptypes;

package acftypes is
  constant nAC:          integer := 39;     -- Number of Assertion Checkers
  constant nACwid:       integer := 5;     -- log2nAC - 1
  constant nEG:          integer := 4;     -- Number of Event Generators
  constant nEGwid:       integer := 1;     -- log2nEG - 1
  constant dbgbuswidth:  integer := 32;
  constant assramwid:    integer := 27;

  constant ACFProject:   integer := 2; -- ISA bus interface
  constant ACFVersion:   integer := 1; -- Assertion set 1
  constant ACFRevision:  integer := 0;

end acftypes;
```

ANNEXE H : Sources Assertions Bus ISA

```
------------------------------------------------------------
-- Description : Assertion Checker FPGA      parameters file
--               Customized for ISA bus interface
--
-- File    : acf_param_ISA_ochp_set2.vhd
-- Author : K. Peterson
-- Created: Oct 31 2004
-- Modif. : Oct 31 2004
--
-- Proj    : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

package ochptypes is
  constant C:         integer := 4;     -- Number of configurations
  constant Cwid:      integer := 1;     -- log2C - 1
  constant N:         integer := 62;    -- Number of internal signals
  constant Nwid:      integer := 5;     -- log2N - 1
  constant P:         integer := 16;    -- Number of external pins
  constant Pwid:      integer := 3;     -- log2P - 1
end ochptypes;

package acftypes is
  constant nAC:          integer := 4;     -- Number of Assertion Checkers
  constant nACwid:       integer := 1;     -- log2nAC - 1
  constant nEG:          integer := 4;     -- Number of Event Generators
  constant nEGwid:       integer := 1;     -- log2nEG - 1
  constant dbgbuswidth:  integer := 32;
  constant assramwid:    integer := 27;

  constant ACFProject:   integer := 2; -- ISA bus interface
  constant ACFVersion:   integer := 2; -- Assertion set 2
  constant ACFRevision:  integer := 0;

end acftypes;
```

ANNEXE H : Sources Assertions Bus ISA

```
------------------------------------------------------------
-- Description : Assertion Checker FPGA      parameters file
--               Customized for ISA bus interface
--
-- File    : acf_param_ISA_ochp_set3.vhd
-- Author : K. Peterson
-- Created: Oct 31 2004
-- Modif. : Oct 31 2004
--
-- Proj    : Assertion-Based Runtime Debugger
------------------------------------------------------------

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
--Use ieee.std_logic_unsigned.all;

package ochptypes is
  constant C:         integer := 4;     -- Number of configurations
  constant Cwid:      integer := 1;     -- log2C - 1
  constant N:         integer := 62;    -- Number of internal signals
  constant Nwid:      integer := 5;     -- log2N - 1
  constant P:         integer := 16;    -- Number of external pins
  constant Pwid:      integer := 3;     -- log2P - 1
end ochptypes;

package acftypes is
  constant nAC:          integer := 2;     -- Number of Assertion Checkers
  constant nACwid:       integer := 0;     -- log2nAC - 1
  constant nEG:          integer := 2;     -- Number of Event Generators
  constant nEGwid:       integer := 0;     -- log2nEG - 1
  constant dbgbuswidth:  integer := 32;
  constant assramwid:    integer := 27;

  constant ACFProject:   integer := 2; -- ISA bus interface
  constant ACFVersion:   integer := 3; -- Assertion set 3
  constant ACFRevision:  integer := 0;

end acftypes;
```