Interopérabilité en émulation et prototypage matériel

A. Blampey

▶ To cite this version:

A. Blampey. Interopérabilité en émulation et prototypage matériel. Micro et nanotechnologies/Microélectronique. Institut National Polytechnique de Grenoble - INPG, 2006. Français. NNT : . tel-00163987

HAL Id: tel-00163987
https://theses.hal.science/tel-00163987

Submitted on 19 Jul 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__/__/__/__/__/__/__/__/__/__|

THESE

pour obtenir le grade de

DOCTEUR de l'INP Grenoble

Spécialité : Micro et nano électronique

préparée au sein du laboratoire TIMA et de l'entreprise STMicroelectronics
dans le cadre de **l'Ecole Doctorale**
«Electronique, Electrotechnique, Automatique et Traitement du Signal»

présentée et soutenue publiquement par

Alexandre BLAMPEY

le 6 Décembre 2006

Interopérabilité en émulation et prototypage matériel

*Directeur de thèse : **M. Ahmed Amine Jerraya***
*Co-directeur de thèse : **M. Joseph Bulone***

Composition du jury :

M. Frédéric Pétrot	Président
M. Emmanuel Boutillon	Rapporteur
M. Michel Robert	Rapporteur
M. Luc Burgun	Examinateur
M. Ahmed Amine Jerraya	Directeur de thèse
M. Joseph Bulone	Co-directeur de thèse

Je tiens à remercier M. Ahmed Amine Jerraya, directeur de recherche au CNRS et responsable du groupe SLS du laboratoire TIMA, pour m'avoir accepté dans son groupe et encadré tout au long de cette thèse.

Je remercie M. Joseph Bulone, responsable des activités d'émulation matérielle chez STMicroelectronics pour m'avoir accueilli au sein de son équipe dans le cadre de ma CIFRE. Merci Joseph de m'avoir accordé ta confiance, de m'avoir proposé un sujet de recherche si passionnant et de m'avoir encadré et conseillé durant ces trois années.

Je remercie M. Frédéric Pétrot de m'honorer en présidant le jury de ma thèse. Je remercie également M. Emmanuel Boutillon et M. Michel Robert pour avoir bien voulu juger cette thèse en acceptant d'en être les rapporteurs. Je remercie M. Luc Burgun, président de la société EVE, de me faire l'honneur de participer à ce jury de thèse en qualité d'examinateur.

Je tiens également à remercier tous mes collègues de bureau qui m'ont fait part de leurs expériences, procuré de bons conseils et aidé dans mes recherches. Donc un grand merci à Helena Krupnova, Nathalie Zaghlan, Christophe Leclerc, Arnaud Richard, Nicolas Mareau, Etienne Lantreibecq et Guillaume Tribet.

Egalement un grand merci à toutes les personnes que j'ai cotoyées durant ces trois années et qui ont contribué à la création d'une superbe ambiance de travail tant chez STMicroelectronis qu'au sein du groupe SLS. Merci donc à mes amis de TIMA, Lobna Kriaa, Iuliana Bacivarov, Mohamed-Wassim Youssef, Youssef Atat, Marius Bonaciu et Patrice Gerin. Merci également à mes amis de ST, Maxime Fiandino, Lorenzo Pieralisi, Vincent Heinrich, Frédéric Hunsinger, Thomas Kunlin, Isabelle Carnel, Ludovic Chotard, Jose Sanches, Alain Bellon, Christophe Dumontier et Bruno Spy.

Enfin, merci à Caroline Barbaray et Roland Blampey pour leurs relectures et toutes leurs remarques et critiques constructives.

Résumé

Cette thèse introduit un nouveau concept dans la vérification des circuits au niveau RTL : l'interoperabilité entre simulateurs HDL, émulateurs matériel et, plateformes de prototypage. Cette thèse permet de bénéficier, dans les processus de vérification du RTL, à la fois de la capacité de déboguage des émulateurs et simulateurs HDL et de l'excellente vitesse d'exécution des plateformes de prototypage. Afin d'atteindre cet objectif, la notion d'état d'un circuit est introduite. Cette thèse présente des outils permettant d'intégrer l'interopérabilité comme une fonctionnalité des circuits à vérifier. Cela permet de rendre interopérable entre elles toutes les machines de vérification travaillant au niveau RTL. L'idée principale de l'interopérabilité consiste en la réalisation des tests sur plateforme de prototypage rapide tout en réalisant périodiquement des sauvegardes d'état. Lorsqu'une erreur apparait, on réalise le déboguage du circuit sur un émulateur rapide, ou sur un simulateur HDL économique. Le test sera exécuté en partant de la dernière sauvegarde précédent l'instant d'apparition du problème. De plus, cette thèse propose une technique permettant de concentrer l'effort de déboguage sur le sous-ensemble défectueux ce qui, d'une part, permet d'augmenter la vitesse sur simulateur et d'autre part, réduit le nombre de domaines nécessaires sur émulateur multi-domaines et augmente ainsi le nombre de déboguages simultanés.

Mots clés

Emulation matérielle, prototypage matériel, simulation HDL, interopérabilité, SceMi, état d'un circuit, captures et restaurations d'états, vérification RTL.

Abstract

This thesis defines a new concept in RTL verification : interoperability between HDL simulators, hardware emulators and hardware prototyping platforms. The main purpose is to benefit from both good speed of hardware prototyping platforms and debug capabilities of hardware emulators and HDL simulators. To achieve this purpose, this thesis introduces the notion of design state. Then, a tool dedicated to interoperability is presented. This tool add interoperability to design functionnalities. This make all machines working at RTL level interoperables with each others. Tha main idea of interoperability is to lunch tests on fast prototyping platforms while periodically saving design state. When a bug will be faced, debug will be performed using a fast emulator or a low cost HDL simulator. The test will restart from the last saved database done just before bug time. Finally, this thesis also introduce a method to focalize debug effort only on the sub-module containing the bug. This improve HDL simulation speed and reduce number of required emulator domains which allow to perform more debugs in the same time using the same emulator.

Key words

Hardware emulation, hardware prototyping, RTL simulation, RTL verification, interoperability, SceMi, design state, save and restore.

Sommaire

Liste des figures..9

Liste des tableaux...10

Chapitre I - Défis et limitations des techniques de vérification des systèmes monopuces...11

1.1) Contexte : conception des systèmes sur puces complexes...12
1.1.1) Evolution des technologies silicium...12
1.1.2) Conception des systèmes monopuces..12
1.2) Nécessité de la vérification..14
1.3) Techniques de vérification des systèmes sur puces...15
1.3.1) Différentes techniques de vérification...15
1.3.2) Complémentarité et besoin de coopération des techniques de vérification...............16
1.4) Contributions...17
1.4.1) Formalisation de la notion d'état d'un circuit et introduction du concept d'interopérabilité en émulation matérielle..17
1.4.2) Définition et semi-automatisation d'un nouveau flot de vérification, orienté interopérabilité, adapté à la vérification des systèmes sur puce.............................19
1.4.3) Validation de l'approche à l'aide d'une expérimentation industrielle sur le circuit STMicroelectronics hls25..19
1.5) Plan du mémoire..19

Chapitre II - Etat de l'art des différentes techniques de vérification et de leurs coopérations...20

2.1) Introduction...22
2.2) Différents niveaux de vérification...22
2.2.1) Introduction aux différents niveaux de vérification..22
2.2.2) Vérification au niveau composant...23
2.2.3) Vérification de l'intégration des composants..23
2.2.4) Vérification du système...24
2.2.5) Synthèse : caractérisation des besoins en fonction du niveau de vérification...........25
2.3) Différentes techniques de vérification..25
2.3.1) Critères de comparaisons..26
2.3.2) Preuve formelle..26
2.3.3) Simulation..27
2.3.4) Emulation et prototypage matériel..28
2.3.4.1) Concept...29
2.3.4.2) Flot de mise en oeuvre...29
2.3.5) Co-émulation..31

2.3.5.1) Co-émulation en mode "vecteurs de test"...32
2.3.5.2) Co-émulation avec synchronisation "cycle à cycle".......................................33
2.3.5.3) Co-émulation avec synchronisation "clairsemée"..33
2.3.5.4) Accélération...34
2.3.5.5) Co-émulation transactionnelle..34
2.3.6) Emulation avec banc de test intégré ("Self Test Bench" - mode STB)....................35
2.3.7) Emulation avec dépendances extérieures ("In Circuit Emulation" - mode ICE).......36
2.4) Etat de l'art des machines d'émulation/prototypage matériel utilisées dans l'industrie........36
2.4.1) Machines d'émulation..37
2.4.1.1) Solutions Mentor Graphics : Celaro et VStation...37
2.4.1.2) Solutions Cadence : Palladium et Xserver..37
2.4.1.3) Synthèse..38
2.4.2) Plateformes de prototypage..38
2.4.2.1) Solutions Eve : ZeBu...39
2.4.2.2) Solutions Mentor Graphics : MP3 et MP4..39
2.4.2.3) Solutions ProDesign : ChipIt..40
2.4.2.4) Solutions Hardi : Haps..40
2.4.2.5) Solutions Flexody : FlexCube...40
2.4.2.6) Synthèse..41
2.5) Synthèse...41
2.6) Nécessité de faire coopérer les différentes techniques de vérification................................43
2.6.1) Sélection du meilleur compromis techniques/machines..43
2.6.2) Besoin de coopération..44
2.7) Coopérations existantes...46
2.7.1) Interface de co-émulation SceMi...46
2.7.2) Flot de conception TLM...47
2.7.2.1) TLM et Co-émulation à synchronisation "cycle à cycle"...............................48
2.7.2.2) TLM et Co-émulation transactionnelle..48
2.7.2.3) Limitations : reproductibilité et longueur des séquences de tests...................49
2.7.3) Solutions internes aux différents fabricants..50
2.7.4) Solutions maisons spécifiques à un circuit...50
2.7.4.1) Problème de cohérence des modèles..50
2.7.4.2) Disparité des possibilités d'accès à l'état d'un circuit......................................52
2.7.4.3) Conclusion : charge de travail élevée et faible réutilisabilité.........................53
2.7.5) Recherches dans le domaine...54
2.8) Nécessité d'une solution générique applicable à toutes les machines dans tous les modes de fonctionnement...54
2.9) Gestion d'un parc de machines...55
2.9.1) Durée de mise en oeuvre...55
2.9.2) Utilisation non optimale des machines..55
2.9.3) Coût financier et temporel de la vérification..56
2.10) Conclusion : nécessité d'un flot automatisé de vérification orienté interopérabilité.........56

Chapitre III - Concept d'interopérabilité en émulation matérielle et flot de vérification associé..57
3.1) Introduction...59
3.2) Notion d'état d'un circuit...59
3.3) Extraction des variables d'état : sélection du modèle de référence....................................60

3.3.1) Hétérogénéité du niveau d'abstraction des modèles des composants du circuit........60
3.3.2) Problèmes de cohérence d'état liés à disparité des résultats de synthèse..................60
3.3.3) Conclusion : la caractérisation de l'état d'un circuit ne peut se faire qu'après synthèse 61
3.4) Accès à l'état d'un circuit...62
3.4.1) Cahier des charges...62
3.4.1.1) Préservation de la vitesse : capture d'état dynamique....................................62
3.4.1.2) Gestion de la bande passante : non altération des co-émulations...................63
3.4.1.3) Gestion des horloges contrôlées...63
3.4.1.4) Limitation du temps d'accès et contrôlabilité du pas d'échantillonnage..........63
3.4.1.5) Gestion du banc de test et des dépendances externes....................................64
3.4.2) Proposition d'une solution d'accès aux ressources mémoire....................................65
3.4.2.1) Duplication des ressources mémoire..65
3.4.2.2) Gestion des horloges...65
3.4.2.2.1) Norme SceMi : exploitation des concepts uclock et cclock.................66
3.4.2.2.2) Norme SceMi : présentation et extension du concept «cycle stamp»…68
3.4.2.3) Extraction d'état par chaîne de scan..69
3.4.2.4) Restauration d'état...70
3.4.2.5) Cellules spécifiques à l'interopérabilité...71
3.4.3) Fichier de sauvegarde d'état..71
3.4.3.1) Structure d'arbre..72
3.4.3.2) Fichiers de description et fichiers de sauvegarde..73
3.4.4) Transacteur lié à l'échange d'état...73
3.4.5) Gestion des signaux d'entrée/sortie...75
3.4.5.1) Problèmes de bande passante et de volume de stockage................................75
3.4.5.2) Compression temps réel..76
3.4.5.2.1) Limitation en surface..76
3.4.5.2.2) Segmentation du bus en ensembles..76
3.4.5.2.3) Codage de Golomb...78
3.4.5.2.4) Gestion des débordements..80
3.4.5.2.5) Fichier de sauvegarde de l'évolution des signaux d'entrée/sortie..........80
3.4.5.2.6) Implémentation et performances..80
3.4.6) Format de sauvegarde d'état..82
3.5) Concept d'interopérabilité..82
3.6) Proposition d'un flot de prototypage semi-automatisé orienté interopérabilité..................82
3.6.1) Flot d'obtention de la netlist de référence..83
3.6.2) Flot de prototypage..84
3.6.3) Outil d'interopérabilité : modification automatique d'une "netlist gate"...................85
3.7) Flot de vérification orienté interopérabilité..87
3.7.1) Architecture d'un système sur puce..87
3.7.2) Phase de vérification et phase de déboguage...88
3.7.3) Optimisation : travail sur un sous-ensemble..90
3.7.3.1) Optimisation des ressources, de la vitesse et du coût du déboguage..............91
3.7.3.2) Obtention de l'environnement du sous-ensemble défectueux.........................93
3.7.4) Apports de la technique proposée..93
3.7.4.1) Apports vis à vis du circuit en validation...93
3.7.4.2) Apports vis à vis de la gestion d'un parc de machines...................................94
3.8) Conclusion..96

Chapitre IV - Application de l'interopérabilité au circuit hls25........................97
4.1) Présentation du circuit hls25..98
4.2) Objectifs de la vérification..98
4.3) Plateforme de vérification..98
4.3.1) Sélection du couple technique/machine..98
4.3.2) Architecture du banc de test...99
4.4) Apport de l'interopérabilité...100
4.5) Mise en oeuvre de plateformes interopérables...101
4.5.1) Instrumentation du circuit...101
4.5.2) Intégration du transacteur d'interopérabilité...101
4.5.3) Plateformes Palladium II...103
4.5.4) Plateformes ZeBu-XL..105
4.5.5) Plateforme NcSim/zTide..107
4.6) Déboguage sur un sous-ensemble...108
4.6.1) Echantillonnage des signaux d'interface..108
4.6.2) Phase de déboguage...109
4.7) Conclusions...110

Chapitre V - Conclusions et perspectives...112
5.1) Conclusions...113
5.1.1) Vérification d'une IP...114
5.1.2) Vérification d'un SoC...114
5.2) Perspectives...115
5.2.1) Extension de la bibliothèque de l'outil «InteroperabiltyCompile»...........................115
5.2.2) Essais sur des cas d'application plus complexes..115
5.2.3) Exploitation des possibilités offertes par les FPGAs Xilinx....................................115
5.2.4) Gestion de plusieurs d'horloges indépendantes..116
5.2.5) Contrôleur mémoire partagé...117

Bibliographie..118

Liste des figures

Figure 1 : Flot de conception d'un système sur puce..13
Figure 2 : Ecart entre les capacités de production, conception et vérification..................................14
Figure 3 : Comparaison des flots de conception avec et sans recours à l'émulation/prototypage......16
Figure 4 : Flot de vérification orienté SoC et interopérabilité...18
Figure 5 : Flot de mise en oeuvre d'une plateforme d'émulation/prototypage matériel.....................30
Figure 6 : Comparaison entre niveau signal et niveau transactionnel..34
Figure 7 : Obtention de la machine idéale par coopération des machines existantes.........................45
Figure 8 : Mise en évidence d'un besoin de coopération entre émulation et prototypage.................45
Figure 9 : Architecture d'une plateforme de co-émulation SceMi...47
Figure 13 : Disparité des primitives d'émulation/prototypage matériel...52
Figure 14 : Exemple d'horloges générées avec SceMi..66
Figure 15 : Sélection des signaux échantillonnés..67
Figure 16 : Concept SceMi "cycle stamp"...68
Figure 17 : Exemple de compteur ustamp et cstamp..69
Figure 18 : Introduction d'une chaîne de scan dans un circuit..70
Figure 19 : Modélisation des variables d'état à l'aide d'un arbre..72
Figure 20 : Transacteur permettant l'accès à l'état d'un circuit...74
Figure 21 : Environnement graphique de pilotage des captures/restaurations d'état..........................75
Figure 22 : Génération d'un mot compressé...77
Figure 23 : Exemple de segmentation d'un bus et transmissions de messages associés.....................78
Figure 24 : Taille du code en fonction du nombre à coder...79
Figure 25 : Architecture du système de compression temps réel...81
Figure 26 : Flot d'obtention de la netlist de référence..83
Figure 27 : Flot de mise en oeuvre de plateformes de vérification interopérables............................85
Figure 28 : Principe de l'outil d'interopérabilité..86
Figure 29 : Architecture du système sur puce Nomadik...87
Figure 30 : Comparaison entre le flot classique et le flot proposé...88
Figure 31 : Flot de déboguage en trois passes..89
Figure 32 : Stratégie d'utilisation d'un émulateur collaborant avec une plateforme de prototypage .90
Figure 34 : Utilisation d'un parc de machines d'émulation/prototypage matériel..............................92
Figure 36 : Allocation dynamique des ressources d'émulation..95
Figure 37 : Architecture de la plateforme de test du circuit hls25...99
Figure 38 : Principe du module bridge appliqué au circuit hls25...102
Figure 39 : Intégration du logiciel hls25 et du logiciel d'interopérabilité..103
Figure 40 : Fichier "bridge" spécifique au module de décodage émulé...109
Figure 41 : Architecture d'une plateforme interopérable à plusieurs horloges indépendantes........116

Liste des tableaux

Tableau 1 : Besoins de la vérification en fonction du type de vérification..25
Tableau 2 : Tableau comparatif des émulateurs utilisés dans l'industrie..38
Tableau 3 : Tableau comparatif des plateformes de prototypage utilisées dans l'industrie................41
Tableau 4 : Comparaison des techniques de vérification du RTL..42
Tableau 5: Exemple de code de Golomb...79
Tableau 6 : Taux de compression en fonction du taux d'utilisation du bus..81
Tableau 7 : Durée de la vérification en fonction de la technique sélectionnée..................................98
Tableau 8 : Impact de l'interopérabilité sur un émulateur Palladium II...104
Tableau 9 : Impact de l'interopérabilité sur une machine ZeBu-XL..106

Chapitre I - Défis et limitations des techniques de vérification des systèmes monopuces

1.1) Contexte : conception des systèmes sur puces complexes....12
1.1.1) Evolution des technologies silicium....12
1.1.2) Conception des systèmes monopuces....12
1.2) Nécessité de la vérification....14
1.3) Techniques de vérification des systèmes sur puces....15
1.3.1) Différentes techniques de vérification....15
1.3.2) Complémentarité et besoin de coopération des techniques de vérification....16
1.4) Contributions....17
1.4.1) Formalisation de la notion d'état d'un circuit et introduction du concept d'interopérabilité en émulation matérielle....17
1.4.2) Définition et semi-automatisation d'un nouveau flot de vérification, orienté interopérabilité, adapté à la vérification des systèmes sur puce....19
1.4.3) Validation de l'approche à l'aide d'une expérimentation industrielle sur le circuit STMicroelectronics hls25....19
1.5) Plan du mémoire....19

1.1) Contexte : conception des systèmes sur puces complexes

Cette thèse s'inscrit dans le contexte de la conception des systèmes monopuces. La conjonction de l'évolution des applications et des technologies de fabrication du silicium fait que la tendance est à l'intégration de plus en plus de fonctions sur des puces contenant de plus en plus de transistors. Il est prévu de pouvoir supporter des applications telles que le traitement vidéo haute définition temps réel grâce à des puces ayant des architectures de plus en plus complexes et utilisant des composants hétérogènes (mémoires, DSP, processeurs, unités de calcul spécifiques).

1.1.1) Evolution des technologies silicium

La conception de cette nouvelle génération de circuits engendre de nouveaux défis. En effet, la complexité de conception des nouveaux systèmes hétérogènes ne cessent d'augmenter. De plus, le coût des technologies silicium augmente exponentiellement [CHA99]. Ainsi, dans [INT06] on apprend que les organismes publiques, toutes nations confondues, ont consacrés huit cents millions de dollars en 2000 pour la mise au point des technologies 130nm, deux milliards en 2002 pour les technologies 90nm et quatre millards en 2005 pour les technologies 65nm. Au final, un circuit ne sera rentable que s'il est produit en plusieurs centaines de milliers voire en millions d'exemplaires.

Cette augmentation des coûts technologiques est telle que même les géants de l'industrie doivent former des alliances pour partager ces coûts et pouvoir ainsi rester sur le marché. Ainsi, Freescale, Philips et STMicrolectronics, trois entreprises classées parmi les dix plus gros producteurs de semiconducteurs, ont construit une usine commune en France sur le site de Crolles.

D'autre part, la durée de vie du marché d'un circuit est de plus en plus court, de l'ordre de quelques mois. Seuls les fabricants arrivant en premier sur un marché obtiendront la majorité de ce marché et pourront vendre leurs circuits en grands volumes, donc amortir le coût de leur technologie et faire des bénéfices. Par exemple, dans [DAL00], l'entreprise NEC a dû retarder le lancement d'un circuit d'environ deux mois, ce qui lui a occasionné un manque à gagner estimé à cinquante millions de dollars.

En conclusion, pour qu'une entreprise puisse rester viable, il est important qu'elle soit la première à proposer un circuit répondant à un besoin. Pour satisfaire cette condition, les fabricants de semiconducteurs doivent réussir à réduire leur durée de conception, ce qui est d'autant plus difficile vu le paradoxe d'avoir une complexité de conception en augmentation incessante et une durée de vie des circuits en constante diminution.

1.1.2) Conception des systèmes monopuces

Classiquement, la conception d'un circuit intégré (ASIC) passe par le développement d'une partie matérielle puis par le développement du logiciel embarqué. Cependant, avec les nouveaux défis lancés par la conception de systèmes monopuces complexes, un tel flot séquentiel n'est plus concevable. On ne peut plus attendre que le matériel soit prêt avant de développer ce logiciel embarqué, sa conception doit démarrer au plus tôt. Au final, la conception des circuits actuels se décompose en cinq étapes (figure 1) et commence à un haut niveau d'abstraction.

Au plus haut niveau d'abstraction, on s'intéresse à la fonctionnalité, indépendamment de l'implémentation finale, ce qui correspond à la conception au niveau système (étape 1). Durant cette phase, on recherche les algorithmes et les représentations de données les mieux adaptés aux besoins, aux spécifications. On obtient ainsi une spécification fonctionnelle fréquemment validée par simulation informatique.

Une fois la spécification fonctionnelle correctement définie, il faut trouver une architecture implémentant les algorithmes précédemment déterminés. Cette étape (étape 2) du flot de conception détermine les fonctionnalités qui seront implémentées en matériel et celles qui seront logicielles. En

général, les composants nécessitant des performances élevées sont réalisés par des modules matériels alors que les composants nécessitant essentiellement de la flexibilité sont implémentés en logiciel. Au final, cette étape permet l'obtention des spécifications de chacun des composants du système.

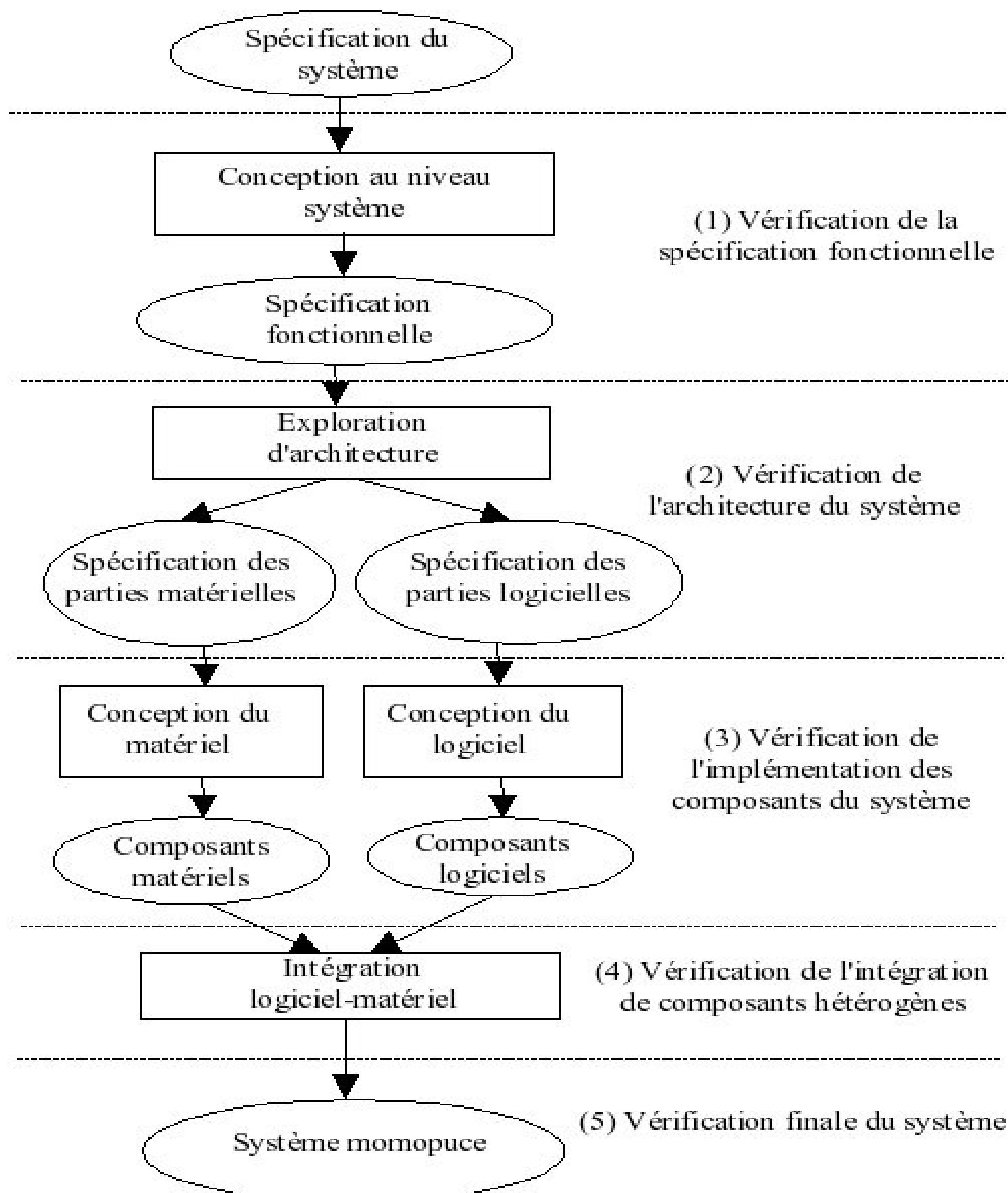


Figure 1 : Flot de conception d'un système sur puce

L'étape suivante (étape 3) est celle de conception des composants matériels et de développement des logiciels embarqués. Pour cette étape, un gain de temps important est obtenu en réutilisant des composants existants.

Une fois tous les composants développés et individuellement vérifiés, il faut les intégrer et vérifier (étape 4) que la communication entre les différents composants se déroule correctement. Enfin, il faut s'assurer que le système complet ainsi obtenu répond bien au cahier des charges, que le système a la fonctionnalité et les performances attendues (étape 5). Enfin, une fois cette dernière étape de conception effectuée, le circuit peut partir dans une fonderie afin d'être produit en grand nombre.

1.2) Nécessité de la vérification

A chaque étape de conception, les concepteurs doivent s'assurer que les nouveaux composants ou les nouveaux détails de réalisation assurent une fonctionnalité correcte, c'est à dire que le système est toujours conforme à ses spécifications.

De part leur complexité et leur hétérogénéité, la vérification des sytèmes monopuces est une tâche complexe et exigeante. Par conséquent, il faut recourir à des techniques spécifiques et adaptées permettant la vérification de plusieurs aspects : aspects fonctionnel et architectural, aspects logiciel et matériel, et communication. Au final, vérifier un système monopuce requiert de nombreuses compétences techniques. Cette vérification s'articule autour de cinq points illustrés par la figure 1 :

- Vérification de la spécification fonctionnelle
- Vérification de l'architecture du système
- Vérification de l'implémentantion des composants du système
- Vérification de l'intégration des composants
- Vérification du système complet dans son environnement de fonctionnement avant mise en fabrication et production.

La vérification peut occuper jusqu'à 70% du temps de conception [EVA03], cette étape est donc un élément clé dans la phase de conception d'un système monopuce. La vérification a un coût important en terme de temps, mais aussi au niveau financier. Une prise de conscience de l'importance de cette phase a émergé lors de la médiatisation de l'erreur de conception du Pentium d'Intel en 1994 [CLA96]. Le coût de cette erreur est estimé à quatre cent millions de dollars. Ce cas n'est pas isolé, de nombreux exemples d'erreurs de conception ont été rapportés dans la littérature (soixante quatre erreurs répertoriées dans le Pentium III [INT00], six dans l'AMD Athlon [AMD00]). Globalement, plus vite une erreur est détectée et plus son coût de correction est faible. Ceci est d'autant plus vrai lors du passage de la description d'un circuit à la réalisation du premier circuit physique en silicium. Cette phase de production nécessite la réalisation d'un jeu de masques spécifique à la technologie utilisée. Les jeux de masques utilisés par les dernières technologies sont très onéreux (de l'ordre du million de dollars [DAL00]), il est donc capital que les circuits partant en fabrication ne comportent plus aucune erreur sinon, il faudra financer plusieurs jeux de masques pour le même circuit, ce qui va augmenter le coût de conception, obliger les entreprises à diminuer leur marge pour vendre ce circuit, et donc rendre l'amortissement plus difficile.

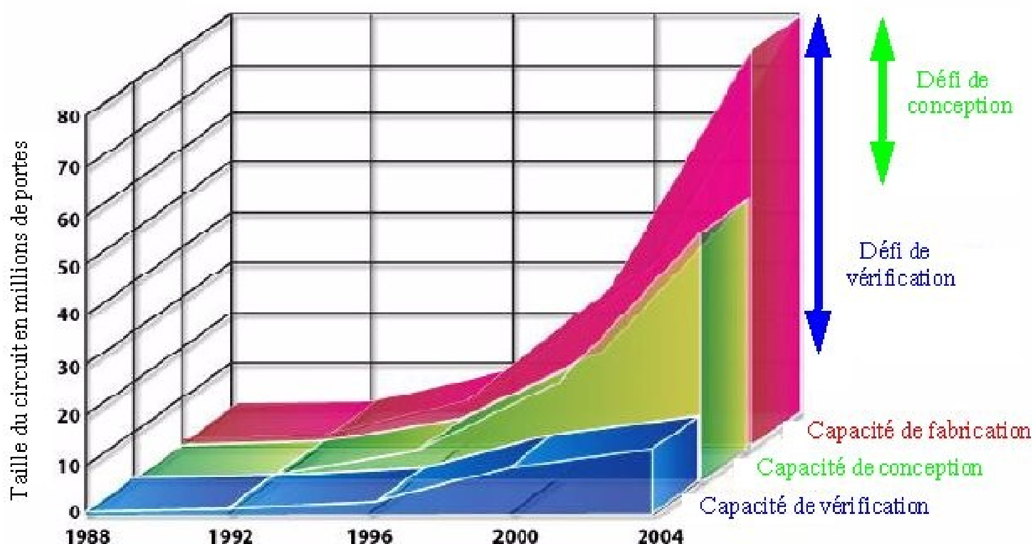


Figure 2 : Ecart entre les capacités de production, conception et vérification

D'autre part, depuis ses débuts, la technologie de conception des circuits intégrés suit la loi de Moore [MOO65] qui prévoit un doublement de la capacité d'intégration tous les dix-huit mois. Cependant, la capacité de développement des circuits monopuces n'a pas une aussi forte croissance que la capacité d'intégration. Ceci est encore plus vrai pour la capacité de vérification, comme le montre la figure 2 tirée d'une publication du SIA (Semiconductor Industry Association). Selon cette publication, il n'est actuellement pas possible de vérifier correctement l'ensemble des circuits conçus, la majorité des circuits nécessitent plusieurs essais de fabrication avant d'assurer leur fonctionnalité, ce qui va engendrer des retards de mise en vente et des surcoûts de conception.

Pour conclure, la vérification est une étape essentielle dans la conception des circuits monopuces. Un des défis actuels consiste à améliorer les techniques de vérification, à augmenter la productivité des techniques, réduire la durée et le coût de la vérification.

1.3) Techniques de vérification des systèmes sur puces

1.3.1) Différentes techniques de vérification

Il existe plusieurs techniques de vérification : la vérification formelle, la simulation, la co-émulation, l'émulation et le prototypage. Chacune de ces techniques utilise un modèle différent.

La vérification formelle introduit des notations formelles caractérisant plusieurs aspects du système. Cela permet de vérifier certaines propriétés. Cette technique est très nécessiteuse en puissance de calcul et a une très faible vitesse d'exécution. De ce fait, elle n'est pas utilisable pour l'ensemble de la vérification.

La simulation emploie des modèles d'exécution et de calcul sur ordinateur. Cette technique offre une grande observabilité, une grande souplesse, une grande flexibilité. Elle permet de rapidement et facilement modifier le modèle d'une fonction, d'un composant dans la représentation du système. Cependant, avec des modèles à bas niveau d'abstraction, très proches des composants physiques, la vitesse de simulation devient vite très limitée, ce qui ne permet pas d'exécuter des tests d'une grande longueur (en terme de cycles d'horloge du système monopuce).

Pour pallier à ce manque de vitesse, des techniques d'émulation et prototypage matériel sont apparues. L'émulation utilise des modèles physiques qui imitent le comportement du matériel. Les modèles utilisés offrent une assez grande observabilité du système, proches de celles des simulateurs mais leur vitesse d'exécution est beaucoup plus rapide. Un émulateur pourra fonctionner à 1MHz là où une simulation fonctionne à 10Hz.

Parmi les autres techniques de vérification, on trouve également la co-émulation qui utilise à la fois la simulation et l'émulation : une partie du circuit est simulée, l'autre est émulée. Cette technique permet de réaliser une vérification rapide (par rapport à une simulation pure) des composants dont le développement est terminé. Ces composants sont alors émulés alors que les composants non disponibles au niveau RTL sont simulés à l'aide d'un modèle fonctionnel à plus ou moins haut niveau d'abstraction.

Enfin, le prototypage est une technique proche de l'émulation, consistant à réaliser un prototype du système avec des modèles physiques très rapides (FPGA) mais qui n'offrent quasiment pas d'observabilité. Le prototypage vise à vérifier la fonctionnalité du système final, à aider au développement des logiciels embarqués. Cette technique est celle qui offre la meilleure vitesse d'exécution (jusqu'à plusieurs dizaines de méga hertz). Elle intervient, en général, lorsque le matériel a atteint un certain niveau de maturité. De plus, il arrive que l'environnement du circuit soit impossible à modéliser. Dans ce cas, il faut vérifier le système dans son environnement, en temps réel. Le prototypage est la seule technique offrant une vitesse d'exécution suffisante pour couvrir ce besoin.

La figure 3 montre le gain apporté par ces techniques d'émulation et prototypage matériel : l'émulation accélère le déboguage du matériel, le prototypage permet d'anticiper le développement

du logiciel et le circuit est prêt plus rapidement. De plus, l'émulation évite de fabriquer un circuit pour le déboguer. D'une part, le déboguage est plus économique et d'autre part, il est plus rapide car la visibilité offerte par l'émulateur est bien meilleure que celle d'un circuit. Ainsi, pour la vérfification d'un circuit STMicroelectronics DVB-S2 [URA05], vérification à laquelle j'ai contribué, l'émulation a été utilisée. Pour vérifier ce circuit, il a fallu passer environ deux cents scénarios de test, chaque scénario nécessitant environ cent trente milliards de cycles. Une simulation au niveau RTL aurait nécessité environ deux mille sept cents ans par test. L'émulation a permis de réaliser ces tests avec une moyenne de sept heures par test. Cet exemple montre bien la nécessité de recourir à des techniques d'émulation pour accélérer la vérification.

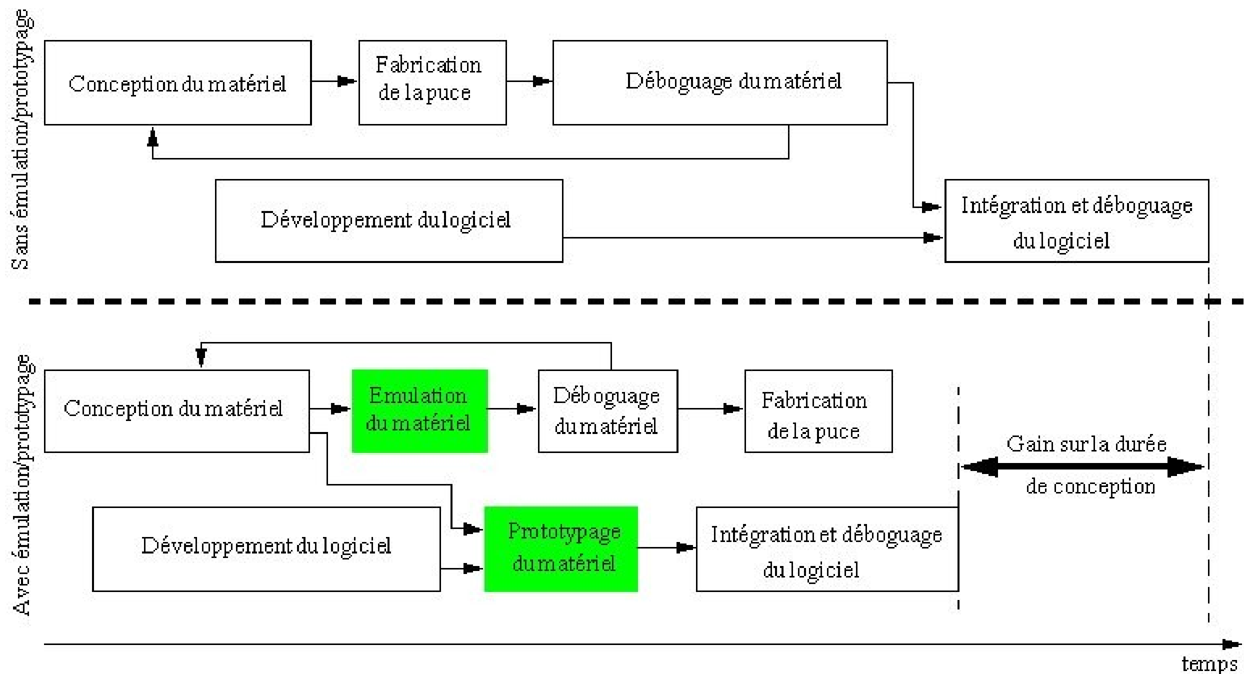


Figure 3 : Comparaison des flots de conception avec et sans recours à l'émulation/prototypage

En conclusion, il existe de nombreuses techniques de vérification RTL. Les plus rapides sont l'émulation et le prototypage matériel. Cependant, les machines associées à ces techniques sont très onéreuses, voir inabordables pour la majorité des entreprises (d'après [LAR04], l'ordre de grandeur du coût annuel d'un émulateur est le million de dollars). Au final, on ne peut pas compenser l'écart entre la capacité de vérification et celle de conception par une utilisation massive d'un gigantesque parc d'émulateurs et de plateformes de prototypage car une telle solution n'est pas financièrement abordable. Il est donc essentiel d'optimiser l'utilisation d'un petit parc de machines, d'utiliser ces machines avec des techniques efficaces de manière à minimiser la durée de vérification des circuits.

1.3.2) Complémentarité et besoin de coopération des techniques de vérification

Si l'on compare les différentes techniques de vérification on remarque qu'aucune solution ne couvre parfaitement l'ensemble des besoins. La simulation est propice au déboguage, de mise en oeuvre rapide mais a une faible vitesse d'exécution. L'émulation offre à la fois vitesse, rapidité de mise en oeuvre et observabilité mais est très onéreuse. Le prototypage offre quant à lui une excellente vitesse d'exécution mais une observabilité quasi nulle, reste à un coût élevé et nécessite une longue durée de mise en oeuvre.

Idéalement, il serait souhaitable d'avoir une machine à bas coût, offrant la vitesse

d'exécution des plateformes de prototypage avec la flexibilité, la rapidité de mise en oeuvre et l'observabilité d'un simulateur. Malheureusement une telle machine n'existe pas et souvent il faut recourir à plusieurs techniques utilisant plusieurs machines avant d'identifier un bogue.

Par exemple, pour la validation d'un circuit STMicroelectronics DVB-S2 [URA05], validation à laquelle j'ai contribué, deux cents scénarios de test ont été exécutés. Chaque test a nécessité cent trente milliards de cycles d'horloge. Pour qu'un test puisse être réalisé en moins d'une journée, il fallait obtenir une fréquence de fonctionnement d'au moins 3MHz. Seules les plateformes de prototypage offrent une telle vitesse d'exécution, une machine ZeBu-ZV fonctionnant à 8MHz a donc été utilisée. Un des tests a révélé une erreur, erreur impossible à identifier sur la plateforme de prototypage. Il a alors fallu rejouer le scénario avec une machine plus propice au déboguage. Un émulateur Palladium fonctionnant à 800kHz, a été utilisé (rejouer un test nécessite alors 70h) et a permis d'observer les communications entre modules du circuit. Le module défectueux a ainsi été identifié et débogué. Cet exemple montre bien la complémentarité entre la haute vitesse d'exécution des plateformes de prototypage et la grande capacité de déboguage des émulateurs.

En conclusion, la validation des circuits nécessite la coopération de plusieurs techniques de vérification. Malheureusement, ces techniques complémentaires reposent sur des machines non prévues pour coopérer. Chaque machine possède son propre modèle d'exécution, son propre format de programmation, sa propre interface de communication (API). Ainsi, il est très difficile d'obtenir la coopération nécessaire et il faut souvent utiliser des techniques sur des machines non adaptées. Cela engendre des situations non optimales quant à l'efficacité de la vérification.

1.4) Contributions

1.4.1) Formalisation de la notion d'état d'un circuit et introduction du concept d'interopérabilité en émulation matérielle

Les précédents paragraphes ont introduit plusieurs problèmes concernant la vérification des systèmes monopuces. Notamment, il est stratégique de réduire la durée et le coût de la vérification. Pour cela, on peut optimiser l'utilisation des machines d'émulation et prototypage matériel. Le précédent paragraphe a montré un besoin de coopération entre les différentes techniques de vérification ; or, actuellement ces techniques reposent sur des machines non prévues pour coopérer.

La première contribution de cette thèse propose un moyen de coopération efficace entre les plateformes de prototypage, les émulateurs et les simulateurs HDL. Pour cela, la notion d'état d'un circuit va être clairement définie ce qui va permettre l'introduction d'un nouveau concept, celui d'interopérabilité en émulation et prototypage matériel. La notion d'interopérabilité consiste en la capacité d'arrêter un scénario de test sur une machine et de le reprendre, à partir du point d'arrêt, sur une autre machine ayant des propriétés mieux adaptées au problème rencontré.

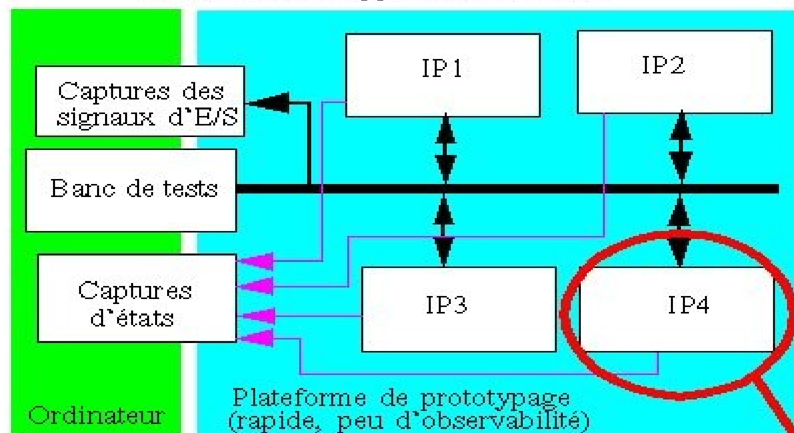
La deuxième contribution de cette thèse propose une solution générique permettant l'accès en lecture et écriture à l'état d'un circuit. Grâce à cela, toutes les machines utilisées (simulateur, émulateur et plateforme de prototypage) seront interopérables les unes avec les autres.

La figure 4 illustre les deux idées principales de cette thèse. Considèrons un circuit que l'on veut vérifier en utilisant les bienfaits de l'interopérabilité. Le circuit est constitué d'un assemblage de blocs «IP» qui communiquent tous entre eux via un bus. Le circuit est complexe et nécessite des séquences de test très longues, représentant plusieurs millards de cycles d'horloge. De plus, tous les modules ne sont pas matures, on sait qu'il faudra détecter et corriger des erreurs matérielles qui pourront se manifester tant en début, qu'au milieu ou en fin de test. Devant la longueur du test, le prototypage, offrant une grande vitesse d'exécution, semble adapté. Néanmoins, l'émulation semble également nécessaire vis à vis du déboguage.

L'interopérabilité va permettre de résoudre ce dilemme quant au choix de la machine la

mieux adaptée. En effet, grâce aux outils développés dans cette thèse, on va ajouter au circuit la possibilité de sauvegarder périodiquement l'état du système. Ainsi, le test sera exécuté en prototypage et bénéficiera de l'excellente vitesse d'exécution de la plateforme et, une sauvegarde d'état sera réalisée toutes les dix minutes. De plus, les stimulis du circuit proviennent d'un banc de tests logiciel. Ces stimulations sont enregistrées en temps réel.

Lorsqu'une erreur apparaîtra, on pourra alors rejouer le test sur un émulateur à partir de la dernière sauvegarde réalisée avant l'instant d'apparition du problème. On pourra ainsi bénéficier des capacités de déboguage de l'émulateur sans avoir à rejouer le test dans son intégralité.



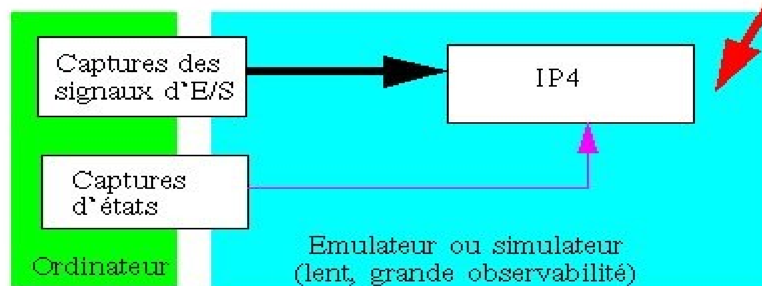


Figure 4 : Flot de vérification orienté SoC et interopérabilité

La deuxième idée principale de cette thèse concerne le déboguage. Dans notre exemple, on a échantillonné les communications du bus reliant tous les modules du circuit. Si l'on peut rapidement identifier quel module est la source du problème, on peut alors concentrer l'effort de déboguage uniquement sur ce sous-ensemble. Ce concept offre deux avantages. D'une part, les émulateurs sont des machines multi-utilisateurs, c'est à dire que plusieurs circuits peuvent être validés en même temps sur la même machine. En travaillant sur un sous-ensemble, moins de ressources seront nécessaires au déboguage, ce qui permettra d'augmenter le nombre de vérifications parallèles.

D'autre part, la vitesse d'une simulation HDL dépend de la taille du circuit. Ainsi, travailler avec un gros circuit sur une longue séquence de test n'est pas concevable vue la durée de simulation nécessaire. Cependant, en travaillant sur un sous-ensemble, on autorise à nouveau l'utilisation d'un simulateur pour réaliser le déboguage.

1.4.2) Définition et semi-automatisation d'un nouveau flot de vérification, orienté interopérabilité, adapté à la vérification des systèmes sur puce

La mise en oeuvre du concept d'interopérabilité nécessite d'accéder à l'état des circuits à vérifier. Pour cela, il faut instrumenter ces circuits. Il s'agit d'un processus complexe augmentant le nombre d'étapes nécessaires à la mise en oeuvre de plateformes d'émulation ou prototypage matériel. Afin que ce surcoût d'étapes n'affecte pas la durée globale de mise en oeuvre, la troisième contribution de cette thèse propose un flot de mise en oeuvre semi-automatisé. Ce flot permet une obtention simple et rapide de plateformes interopérables.

D'autre part, l'interopérabilité en émulation matériel ouvre de nouvelles perspectives quant aux stratégies de vérification. La quatrième contribution de cette thèse consiste en la proposition d'un nouveau flot de vérification exploitant les nouvelles possibilités offertes par l'interopérabilité. L'idée principale est d'exploiter la haute vitesse d'exécution des plateformes de prototypage pour l'exécution des tests ainsi que les hautes capacités de déboguage des émulateurs et simulateurs pour l'identification des bogues. Ce flot, adapté à la vérification des systèmes sur puce, a deux apports stratégiques. D'une part, la durée de vérification sera réduite et d'autre part, le taux d'utilisation des machines sera amélioré, les parcs de machines seront mieux exploités. En particulier, la stratégie proposée favorise l'utilisation massive de plateformes de prototypage à coût modéré avec quelques émulateurs très onéreux. Actuellement, l'émulation est l'outil dominant. En suivant la stratégie proposée, la répartition entre émulateurs et plateformes de prototypage va donner l'avantage au prototypage et donc réduire le coût de la vérification. Enfin, pour les opérations de déboguage, le flot de vérification proposé autorise, là ou classiquement on doit obligatoirement utiliser un émulateur, l'utilisation au choix d'un simulateur ou d'un émulateur. On offre ainsi le choix entre déboguage rapide (émulateur) où déboguage économique (simulateur).

1.4.3) Validation de l'approche à l'aide d'une expérimentation industrielle sur le circuit STMicroelectronics hls25

Les approches proposées par cette thèse laissent entrevoir d'importantes améliorations, tant sur la durée que sur le coût de la vérification. Cependant, pour convaincre des industriels, une thérorie seule ne suffit pas, il faut des cas concrets. Afin de valider les gains théroriques, la dernière contribution consiste en la validation des théories proposées sur un cas réel. Chez STMicroelectronics, l'équipe en charge de la validation du circuit de transmission DVB-S2 «hls25» a été grandement intéressée par le concept d'interopérabilité et le flot de vérification proposé. Ce circuit va donc servir comme preuve de l'efficacité de la méthode. En pratique, une plateforme de prototypage ZeBu-XL, un émulateur Palladium II et un simulateur NcSim ont été utilisés et rendus interopérables.

1.5) Plan du mémoire

Ce mémoire de thèse s'articule autour de cinq chapitres :

- Le premier chapitre, qui se termine ici, est un chapitre d'introduction.
- Le second chapitre propose un état de l'art détaillé des techniques de vérification et de leurs coopérations. Ce chapitre fait notamment ressortir les besoins de coopération non satisfaits par les techniques et machines utilisées dans l'industrie.
- Le troisième chapitre introduit le concept d'interopérabilité en émulation et prototypage matériel et présente un nouveau flot de vérification associé à ce nouveau concept.
- Le quatrième chapitre présente le cas d'application «hls25».
- Le dernier chapitre analyse les résultats de ces travaux de thèse et donne des perspectives.

Chapitre II - Etat de l'art des différentes techniques de vérification et de leurs coopérations

2.1) Introduction....................22
2.2) Différents niveaux de vérification....................22
  2.2.1) Introduction aux différents niveaux de vérification....................22
  2.2.2) Vérification au niveau composant....................23
  2.2.3) Vérification de l'intégration des composants....................23
  2.2.4) Vérification du système....................24
  2.2.5) Synthèse : caractérisation des besoins en fonction du niveau de vérification....................25
2.3) Différentes techniques de vérification....................25
  2.3.1) Critères de comparaisons....................26
  2.3.2) Preuve formelle....................26
  2.3.3) Simulation....................27
  2.3.4) Emulation et prototypage matériel....................28
    2.3.4.1) Concept....................29
    2.3.4.2) Flot de mise en oeuvre....................29
  2.3.5) Co-émulation....................31
    2.3.5.1) Co-émulation en mode "vecteurs de test"....................32
    2.3.5.2) Co-émulation avec synchronisation "cycle à cycle"....................33
    2.3.5.3) Co-émulation avec synchronisation "clairsemée"....................33
    2.3.5.4) Accélération....................34
    2.3.5.5) Co-émulation transactionnelle....................34
  2.3.6) Emulation avec banc de test intégré ("Self Test Bench" - mode STB)....................35
  2.3.7) Emulation avec dépendances extérieures ("In Circuit Emulation" - mode ICE)....................36
2.4) Etat de l'art des machines d'émulation/prototypage matériel utilisées dans l'industrie....................36
  2.4.1) Machines d'émulation....................37
    2.4.1.1) Solutions Mentor Graphics : Celaro et VStation....................37
    2.4.1.2) Solutions Cadence : Palladium et Xserver....................37
    2.4.1.3) Synthèse....................38
  2.4.2) Plateformes de prototypage....................38
    2.4.2.1) Solutions Eve : ZeBu....................39
    2.4.2.2) Solutions Mentor Graphics : MP3 et MP4....................39
    2.4.2.3) Solutions ProDesign : ChipIt....................40
    2.4.2.4) Solutions Hardi : Haps....................40
    2.4.2.5) Solutions Flexody : FlexCube....................40
    2.4.2.6) Synthèse....................41
2.5) Synthèse....................41
2.6) Nécessité de faire coopérer les différentes techniques de vérification....................43
  2.6.1) Sélection du meilleur compromis techniques/machines....................43
  2.6.2) Besoin de coopération....................44
2.7) Coopérations existantes....................46
  2.7.1) Interface de co-émulation SceMi....................46
  2.7.2) Flot de conception TLM....................47

2.7.2.1) TLM et Co-émulation à synchronisation "cycle à cycle".......................................48
2.7.2.2) TLM et Co-émulation transactionnelle...48
2.7.2.3) Limitations : reproductibilité et longueur des séquences de tests...........................49
2.7.3) Solutions internes aux différents fabricants..50
2.7.4) Solutions maisons spécifiques à un circuit..50
2.7.4.1) Problème de cohérence des modèles..50
2.7.4.2) Disparité des possibilités d'accès à l'état d'un circuit...52
2.7.4.3) Conclusion : charge de travail élevée et faible réutilisabilité.................................53
2.7.5) Recherches dans le domaine..54
2.8) Nécessité d'une solution générique applicable à toutes les machines dans tous les modes de fonctionnement..54
2.9) Gestion d'un parc de machines...55
2.9.1) Durée de mise en oeuvre..55
2.9.2) Utilisation non optimale des machines...55
2.9.3) Coût financier et temporel de la vérification...56
2.10) Conclusion : nécessité d'un flot automatisé de vérification orienté interopérabilité................56

2.1) Introduction

Ce chapitre introduit dans un premier temps les différents objectifs de la vérification d'un système monopuce tout au long du flot de conception. Après avoir mis en évidence les besoins de vérification au niveau RTL, les différentes techniques de vérification utilisées pour répondre à ces besoins seront présentées. Cet état de l'art est centré sur les techniques utilisées en milieu industriel, spécialement sur les techniques d'émulation et prototypage matériel.

Cet état de l'art permet la caractérisation de la machine de vérification idéale afin de la comparer aux machines existantes. Cette comparaison montre que les performances de la machine idéale peuvent être obtenues en améliorant la coopération des machines existantes.

Dans un second temps, ce chapitre dresse un état de l'art des capacités de coopération existantes. Cet état de l'art fait ressortir les limitations des solutions actuelles et montre les principaux axes à améliorer en matière de coopération.

2.2) Différents niveaux de vérification

Dans la création d'un système, il faut toujours s'assurer que celui-ci fonctionne correctement. La création d'un système sur puce passe par deux étapes principales : une phase de conception et une phase de fabrication [WEB99]. Chacune de ces deux phases nécessite un processus de vérification adapté. Dans la phase de conception, la vérification a pour but de s'assurer que le système fournit bien les fonctionnalités désirées, qu'il est conforme aux spécifications. Dans le processus de fabrication, la vérification vise à s'assurer que le produit fabriqué est identique au produit conçu. Cette thèse visant à améliorer la vérification de la conception, la vérification de la fabrication ne sera pas plus détaillée.

2.2.1) Introduction aux différents niveaux de vérification

Le chapitre précédent a introduit la vérification qui peut être subdivisée en cinq sous-ensembles, comme le montre la figure 1.

Les deux premiers sous-ensembles ont un haut niveau d'abstraction. Dans la phase de conception au niveau système on détermine et vérifie les algorithmes mis en jeu par le système. Une fois ces algorithmes déterminés, l'étape suivante consiste à déterminer l'architecture la mieux adaptée. La vérification consiste alors à vérifier que l'architecture retenue supporte bien les algorithmes mis en jeu, qu'il ne va pas y avoir des problèmes de congestion de données. Cette étape permet la définition des spécifications de l'ensemble des composants (logiciels et matériels) du système.

A partir des étapes suivantes, le niveau d'abstraction diminue puisque l'on conçoit les différents composants du système. Cette thèse est focalisée sur l'utilisation des techniques d'émulation et prototypage matériel. De plus, les émulateurs ont comme point d'entrée des descriptions RTL de circuits. Ainsi, les étapes suivantes de conception et vérification sont essentielles vis à vis du sujet considéré et seront détaillées dans les prochains paragraphes.

D'autre part, chaque type de vérification va être caractérisée en fonction de ses besoins. Les paramètres répertoriés sont le niveau de maturité du circuit (la probabilité d'apparition d'une erreur), la complexité des tests nécessaires, la longueur des séquences de tests et la taille (en nombre de portes logiques) de l'ensemble en cours de vérification. Globalement, on peut déjà affirmer que plus on avance dans le processus de conception et plus le nombre d'erreurs à corriger diminue mais, le nombre de cycles nécessaires au test augmente ainsi que la complexité des tests [PET00][PET01].

2.2.2) Vérification au niveau composant

Une fois fixée la spécification de chaque composant, il faut les concevoir ou adapter des composants déjà existants. Rentre alors en jeu un premier type de vérification fonctionnelle au niveau du bloc. On vérifie alors que les spécifications du module sont bien respectées, que le module fournit bien la fonction pour laquelle il a été conçu et qu'il peut être connecté au reste du circuit, que ses ports de communication respectent bien les protocoles prévus [ABR98].

Cette vérification fonctionnelle nécessite dans la plupart des cas de courtes séquences de test avec un gros effort de déboguage [PET00][PET01]. Dans l'article [ABR98], un ordre de grandeur est donné concernant le nombre d'erreurs contenues dans le code VHDL d'un bloc matériel venant d'être développé : deux erreurs pour cent lignes de code. L'article fournit également un exemple pour lequel l'effort de vérification a consisté à trouver environ deux mille quatre cents erreurs réparties dans cent vingt mille lignes de code pour un circuit de huit cent mille portes logiques. Si l'on ramène ce nombre d'erreurs par rapport à la taille d'un circuit (en nombre de portes), on obtient l'encadrement suivant :

$$\frac{Taille_{Circuit}}{1000} < nb_{erreurs} < \frac{Taille_{Circuit}}{100}$$

D'autre part, les modules ont souvent une petite taille. Les plus gros modules nécessitent jusqu'à deux millions de portes logiques, ce qui se traduit, dans le pire cas, par environ quelques milliers d'erreurs à identifier dans le code HDL. Les séquences de test utilisées sont relativement courtes, de l'ordre de quelques centaines ou milliers de cycle, ce qui ne nécessite pas une énorme puissance de calcul. L'exécution d'un test en quelques minutes est acceptable, ce qui correspond à des fréquences de fonctionnement minimum de l'ordre de 100Hz.

Pour conclure sur la vérification au niveau composant, étant donné le grand nombre d'erreurs à identifier, les outils adaptés doivent offrir une grande souplesse d'utilisation, des temps de recompilation très courts et une grande observabilité. Vu la faible longueur des séquences de test, la vitesse d'exécution nécessaire est faible (100Hz au minimum), ce qui impose une puissance de calcul modérée. Néanmoins, la vitesse d'exécution peut devenir critique lorsque l'on a énormément de tests à passer de type non-regressions. Dans ce cas, la moindre correction d'une erreur impose de repasser l'intégralité des tests.

2.2.3) Vérification de l'intégration des composants

Une fois l'ensemble des composants développé, il faut les assembler afin de réaliser le système sur puce. Cette étape d'intégration nécessite également une procédure de vérification adaptée consistant à s'assurer que l'ensemble des composants arrive à communiquer entre eux, c'est à dire qu'ils soient tous capables de se synchroniser et qu'ils exécutent les bons protocoles de communication [ABR98]. A ce niveau, on s'assure également que chaque composant accède correctement à son espace d'adressage théorique et qu'il assure bien sa fonctionnalité. Cela peut sembler redondant vu que la fonctionnalité de chaque composant a déjà été testée au préalable. Cependant, l'intégration fait souvent apparaitre des scénarios non prévus dans la précédente étape de test et fait ainsi ressortir plusieurs bogues.

Durant cette étape de vérification, on assemble l'ensemble des composants du système, on travaille donc avec des circuits complets pouvant atteindre une taille de plusieurs dizaines de millions de portes logiques. Vérifier une telle quantité de portes, au niveau RTL, implique de recourir à de grandes puissances de calcul.

Concernant les besoins d'observabilité, de déboguage, ceux-ci sont également importants. En effet, si l'on considère qu'il y a «n» composants dans le système considéré, il y a alors «n x (n-1)» intéractions à vérifier [PET00],[PET01]. Soit «Pi» la probabilité qu'un composant fonctionne

correctement. La probabilité que l'assemblage des «n» composants ne contienne aucune erreur est alors donné par la loi de Bernoulli : $P(circuit_{ok})=\prod Pi$

Dans les systèmes intégrés actuels, on trouve fréquement plus d'une dizaine de composants, ce qui signifie, en admettant que Pi soit de l'ordre 98%, que la probabilité que l'assemblage des composants fonctionne correctement est en dessous de 82%. Ceci montre bien l'importance de cette vérification. Soit «Ni» le nombre d'erreurs du composant «i» estimé par la méthode de la section précédente (deux erreurs pour cent lignes de code), le nombre d'erreurs restantes au niveau du composant «i» est alors donné par l'équation suivante : «NBi = (1 - Pi) x Ni». On en déduit alors que le nombre d'erreurs à detecter à cette étape de vérification est donné par la somme des erreurs restantes dans chacun des modules : $Nb_{erreurs}=\sum_{i=0}^{n-1}(1-P_i)\times N_i$

Le terme «Pi» varie en fonction de la complexité du composant «i» et, du niveau de réutilisabilité du composant. Si un composant a déjà été utilisé dans d'autres circuits, il a déjà subi de nombreuses vérifications, sa fiabilité est donc très élevée, ce qui correspond à un «Pi» proche de 1. En reprenant l'exemple du circuit de l'article [ABR98], circuit contenant environ deux mille quatre cents erreurs pour un circuit de huit cents mille portes, et en considérant que 99% des erreurs ont été corrigées par la vérification au niveau composant, il reste alors environ vingt-quatre erreurs à identifier à ce niveau de vérification. Etant donné que les circuits sont de plus en plus gros, qu'ils intègrent de plus en plus de composants, composants de plus en plus complexes, il est évident que le nombre d'erreurs à détecter à cette étape de vérification va aller en augmentant. Globalement, on peut considérer qu'il reste à détecter environ dix erreurs par million de portes.

Enfin, concernant la longueur des séquences de test, celles-ci restent souvent relativement courtes. En effet, on cherche ici à vérifier la communication entre composants, la longueur des séquences de test est donc directement dépendante des protocoles de communication qui nécessitent en général moins d'un millier de cycles par échange. Cela se traduit par une vitesse d'exécution minimale de l'ordre de 1kHz.

Pour conclure, la vérification de l'intégration nécessite de grandes puissances de calcul vue la taille des circuits à vérifier. Le nombre d'erreurs est conséquent, les techniques de vérifications adaptées doivent donc fournir une grande observabilité.

2.2.4) Vérification du système

Une fois l'intégration validée, la dernière étape de vérification avant la mise en production consiste à s'assurer que le système complet assure bien la fonctionnalité pour laquelle il a été conçu avec les performances attendues [ABR98]. Par exemple, dans le cas d'un codeur vidéo temps réel, on vérifie à ce niveau que le codage fonctionne correctement, que les fichiers générés peuvent bien être correctement décodés par les différents lecteurs du commerce. On vérifie également l'aspect temps réel et le niveau de qualité obtenu par rapport au cahier des charges.

Cette étape de vérification est importante car les erreurs non détectées à cette étape se retrouveront dans le système final, ce qui engendrera des coûts et des délais de correction très importants, comme cela a été expliqué dans le premier chapitre.

L'idéal, à ce niveau de vérification, est de tester le système dans son environnement de fonctionnement, si possible avec un utilisateur étranger au projet. En effet, un tel utilisateur trouvera probablement des conditions, des scénarios d'utilisation imprévus et donc non testés par les différentes séquences de test préalablement passées. D'autre part, avoir un prototype du circuit capable de fonctionner dans le véritable environnement de fonctionnement est très utile vis à vis de la validation et de la finalisation des logiciels embarqués.

Concernant les besoins de la vérification fonctionnelle du système, on travaille ici avec un système complet, ce qui implique de grandes puissances de calcul, comme pour la vérification de

l'intégration. De plus, l'aspect vérification dans l'environnement physique du circuit impose une nouvelle contrainte, une contrainte temps réel, ce qui va encore augmenter l'importance de la puissance de calcul nécessaire. En pratique, peu d'outils permettent de satisfaire cette contrainte en vitesse; il faut obligatoirement recourir à du prototypage.

Typiquement, la mise en oeuvre du système complet dans un but de validation nécessite des séquences de test relativement longues. Par exemple, pour valider un codeur vidéo, il faudra le tester sur plusieurs séquences de quelques minutes pour le valider. Avec une fréquence de fonctionnement de l'ordre de 100MHz, 60s de vidéo représentent six milliards de cycles d'horloge. Durant mes travaux de thèse, j'ai pu participer à la vérification fonctionnelle d'un circuit de réception satellite [URA05] visant à valider les performances d'un système de code correcteur d'erreurs. Plusieurs centaines de séquences de test ont été nécessaires, chacune d'entre elles nécessitant environ cent-cinquante milliards de cycles d'horloge. Vu la longueur des séquences de test, afin d'exécuter chacun des tests en un temps raisonnable (12h par test), il fallait un système fonctionnant au moins à 3MHz. Cet exemple montre bien le besoin en vitesse des plateformes de vérification fonctionnelle.

Enfin, les besoins en capacité de déboguage sont faibles à ce niveau de vérification. En effet, on vérifie les performances du système, les différents composants sont normalement matures, fiables, débarrassés de toutes erreurs matérielles. Cependant, bien que la probabilité qu'une erreur matérielle subsiste soit faible, cette probabilité n'est pas nulle. Les outils les mieux adaptés n'auront donc pas besoin de fournir une grande observabilité mais les capacités d'observabilité ne devront pas être nulles pour autant.

Pour conclure, la vérification complète d'un circuit complexe nécessite avant tout une plateforme de vérification très rapide (plusieurs mégahertz), capable d'accepter des gros circuits (plusieurs dizaines de millions de portes) et offrant peu de possibilité de déboguage matériel.

2.2.5) Synthèse : caractérisation des besoins en fonction du niveau de vérification

Les précédents paragraphes ont fait ressortir plusieurs aspects de la vérification résumés dans le tableau 1. Tout d'abord, plus on avance dans la vérification du système et plus la taille des éléments en cours de vérification augmente, les besoins en vitesse de la plateforme de vérification augmentent également ainsi que la longueur et la complexité des séquences de test. Seul les besoins au niveau de la capacité du déboguage matériel diminuent. Différentes techniques répondent à ces besoins et seront présentées dans les prochains paragraphes.

Vérification\besoins	*Taille de l'élément testé*	*Vitesse*	*Déboguage*
Composant	+ (<1M portes)	+ (>100Hz)	+++
Intégration	+++ (>1M portes)	++ (>1kHz)	++
Système	+++ (>1M portes)	+++ (>1MHz)	+

Tableau 1 : Besoins de la vérification en fonction du type de vérification

2.3) Différentes techniques de vérification

Avec l'accroissement de la complexité et de la taille des circuits sont nées plusieurs techniques de vérification, chacune d'elles répondant aux nouveaux besoins du moment.

2.3.1) Critères de comparaisons

Chacune des techniques de vérification est caractérisée par un coût financier, une durée de mise en oeuvre, une vitesse d'exécution, un niveau d'observabilité, un niveau de contrôlabilité et enfin un niveau de répétabilité.

Le **coût financier** de la vérification est un critère important. Les entreprises ont des budgets qu'elles ne peuvent pas dépasser, il est donc essentiel de bien maîtriser le coût de la vérification. Pour bien considérer ce point, il faut avoir une vision large du procédé de fabrication d'un circuit intégré : plus une erreur matérielle est tardivement détectée, plus son coût de correction est élevé. Par conséquent, allouer un gros budget à la vérification peut s'avérer utile quand on sait que le prix d'un émulateur est amorti si une seule erreur matérielle critique est trouvée avant la fabrication du premier circuit. Cependant, les prix des outils de vérification varient dans une large gamme de prix allant de quelques milliers de dollars pour une licence perpétuelle [RIZ03] au million de dollars pour une licence annuelle [LAR04]. Le paramètre financier n'est donc pas anodin dans le choix des outils.

La **durée de mise en place** de la plateforme de vérification est un autre critère primordial. En fonction de la méthode de vérification utilisée, de la taille et de la complexité du circuit, elle peut varier de quelques dizaines de minutes à plusieurs mois. La prise en compte de ce paramètre peut donc être négligeable dans certains cas et devenir capital dans d'autres. De plus, ce paramètre doit être estimé en association avec la vitesse d'exécution et la longueur des séquences de test. En effet, si pour réaliser une vérification on hésite entre deux techniques, l'une fonctionnant à 1kHz par exemple et l'autre à 1MHz et que la séquence de test nécessite un millard de cycles d'horloge, la durée du test sera donc d'environ douze jours pour la première méthode et de dix-sept minutes pour la deuxième. La deuxième méthode sera donc rentable si l'écart entre les durées de mise en place des deux plateformes n'excède pas douze jours, que l'on prêt à accepter le coût en ressources humaines et qu'il ne faille pas répéter cette séquence ultérieurement.

La **vitesse d'exécution,** est un autre paramètre dont l'importance vient d'être démontrée. Elle doit être considérée conjointement avec la durée de mise en place de la plateforme de vérification et la longueur des séquences de test. Dans les cas de vérification temps réel avec des composants externes, la vitesse peut devenir une contrainte imposant une solution.

L'**observabilité**, la **contrôlabilité** et la **répétabilité** définissent la puissance de déboguage matériel, c'est à dire l'efficacité de la technique pour la mise en évidence des erreurs matérielles. L'observabilité est la capacité d'observer les interactions entre les différents composants du système. La contrôlabilité est la capacité de suspendre l'exécution du modèle, de forcer les valeurs de certains paramètres au cours de l'exécution. La répétabilité est la capacité de reproduire un scénario de test avec un niveau de précision donné. Ce point est particulièrement important car si l'on ne peut pas rejouer un des scénarios de test, il se peut que l'on ne puisse pas reproduire des bogues ce qui rendra leur correction impossible.

Au vu de ces nombreux paramètres, il n'est pas aisé de choisir la méthode la plus adaptée au problème considéré. Avant d'aller plus loin sur ce sujet, il convient de présenter les différentes techniques de vérification existantes puis de les comparer.

2.3.2) Preuve formelle

La vérification formelle sert à prouver, de manière mathématique, qu'une description de circuit possède certaines propriétés. Il y a essentiellement deux types de vérification formelle [STA94][GEO01], le déboguage de la spécification qui vérifie si tous les besoins sont bien inclus et la vérification de l'implémentation qui vérifie si la spécification est bien implémentée.

Cette technique n'est encore que peu utilisée dans la conception des sytèmes monopuces. Les principaux obstacles [ROS98] sont :

- La complexité du processus de vérification est très grande, ce qui restreint l'application seulement aux cas simples.
- Cette technique n'est pas encore bien automatisée, une grande interaction entre concepteurs et outils est requise. Seuls des spécialistes peuvent utiliser cette technique.
- Il est difficile de prendre en compte l'environnement.
- Un système monopuce implique des composants logiciels et matériels qui ont des caractéristiques très différentes du point de vue de la vérification formelle. La matériel est un système synchrone parallèle alors que le logiciel est un système asynchrone séquentiel. Ainsi, ces deux types de composants ne sont pas traités par les mêmes techniques, il n'est possible de traiter que du logiciel ou du matériel mais pas un système complet intégrant à la fois du logiciel et du matériel.

Cette technique est difficile à mettre en oeuvre d'où sa faible utilisation. Par contre, entre les mains d'un spécialiste, cette technique de vérification est à faible coût puisqu'elle ne nécessite qu'un simple ordinateur pour effectuer les calculs.

Au niveau de l'efficacité, cette technique peut se révéler très bonne si le modèle formel du système incorpore de nombreux paramètres, c'est à dire s'il est à un bas niveau d'abstraction. La preuve formelle est reproductible quel que soit le niveau d'abstraction du modèle formel. De plus, elle est très souple, très flexible puisque les modèles sont construits à la main. Par contre, cette flexibilité est aussi un inconvénient car la construction des modèles est un travail difficile, fastidieux et coûteux en temps.

Enfin, la vitesse de vérification est assez faible. Ainsi, cette technique ne peut pas être appliquée aux gros circuits, la vérification d'un système monopuce ne peut pas reposer uniquement sur la preuve formelle.

Dans l'industrie, cette technique est couramment utilisée au niveau composant pour vérifier que la «netlist» obtenue après synthèse assure bien la même fonctionnalité que celle décrite dans les fichiers VHDL/Verilog.

2.3.3) Simulation

La simulation informatique est une approche basée sur l'utilisation d'un modèle comportemental et informatique du système en cours de développement. Pour un même système, plusieurs modèles à différents niveaux d'abstraction peuvent être utilisés, ce qui permet de vérifier le comportement d'un système plus ou moins rapidement à différents degrés de précision. Plus le modèle est précis, plus les calculs pour la simulation sont nombreux et par conséquent, plus l'exécution est lente.

Cette technique de vérification est la technique la plus utilisée dans les conceptions des circuits numériques monopuces, elle intervient à six différents niveaux d'abstraction [CLO02].

Le **niveau spécification fonctionnelle** modélise le comportement global du système sans aucune précision vis à vis de sa réalisation finale. Ici, on travaille à un très haut niveau d'abstraction, une simulation à ce niveau permet de très rapidement simuler une spécification fonctionnelle et permettra, par son analyse, de mettre en évidence les besoins du système.

Le **niveau architectural** modélise le système comme un ensemble de modules travaillant en parallèle et communiquant entre eux. A ce niveau, les différentes tâches du système sont allouées à des sous-systèmes. Chaque sous-système est modélisé au niveau fonctionnel. La granularité concernant les interactions est au niveau transactionnel (TLM). Ce type de simulation est particulièrement utile pour l'exploration d'architecture et le développement des parties logicielles du système.

Le **niveau micro-architecture** correspond à la même simulation qu'au niveau architectural sauf qu'ici, les interactions entre sous-systèmes ne sont plus des transactions mais des signaux. La

précision du modèle est donc au cycle d'horloge prêt au niveau de la communication entre sous-systèmes. Ce niveau de modélisation permet la réalisation de premières mesures de performances ainsi que le développement des pilotes de bas niveau («drivers») des logiciels embarqués.

Le **niveau RTL** modélise un circuit comme un ensemble de registres et de relations logiques entre registres. Ce modèle est à bas niveau d'abstraction, le système entier est simulé au cycle d'horloge prêt. Ce niveau est particulièrement utilisé pour la mise au point des sous-ensembles matériels qui composent le système.

Le **niveau porte logique** décrit le système complet comme un assemblage de portes logiques. Ce niveau est obtenu après synthèse. De nos jours, les outils permettant le passage du niveau RTL au niveau porte logique sont automatisés et suffisament fiables pour ne pas avoir à travailler à ce niveau pour la conception d'un ASIC. Cependant, dans les flots d'émulation qui nécessitent eux aussi une phase de synthèse, le niveau de fiabilité est moins élevé et il s'avère parfois nécessaire d'effectuer des simulations à ce niveau pour trouver une erreur de synthèse.

Le **niveau analogique** est le plus bas niveau d'abstraction utilisé en simulation. A ce niveau existent des outils d'extraction de paramètres électriques à partir du plan de masse. On travaille ici avec des modèles précis des transistors, dépendant de la technologie utilisée.

Pour conclure, la simulation est donc utile à toutes les phases de conception. Cette technique est très efficace de part sa grande souplesse, sa grande flexibilité, observabilité, contrôlabilité et un temps de mise en oeuvre souvent court. Cependant, plus on avance dans le processus de conception et plus la précision du modèle nécessaire augmente, ce qui implique plus de calculs et donc une vitesse d'exécution moindre. La simulation trouve donc ses limites lorsqu'il faut jouer de longues séquences de tests à un bas niveau d'abstraction.

Au niveau RTL, niveau sur lequel se focalise cette thèse, la simulation offre une observabilité et une contrôlabilité maximale puisque tous les signaux du circuits sont observables et forçables. La simulation au niveau RTL permet une reproductibilité au niveau cycle d'horloge.

Au niveau RTL, il faut compter une dizaine d'instructions CPU pour simuler l'équivalent d'une porte logique [PET00][PET01]. On en déduit facilement l'équation qui donne la fréquence maximale du système : $f_{circuit} = \frac{f_{CPU}}{(10 \, x \, nb_{portes})}$ La fréquence d'un circuit simulé au niveau RTL est donc inversement proportionnelle à la taille du circuit. Les ordinateurs actuels offrent des fréquences de fonctionnement de plusieurs giga-hertz. Sur des petits composants (moins de dix mille portes logiques), on pourra espérer atteindre la dizaine de kilohertz comme fréquence de fonctionnement en simulation. Avec un circuit complet ayant une taille moyenne aujourd'hui de dix millions de portes logiques, la fréquence du système simulé ne dépassera pas 10Hz, ce qui est largement insuffisant pour opérer une vérification nécessitant de longues séquences de test. Au niveau RTL, la simulation est donc particulièrement bien adaptée pour la vérification au niveau composant.

Concernant l'aspect financier, il faut intégrer le coût de l'ordinateur, le coût des licences des logiciels et le coût de développement des modèles. Au final, le coût global sera de l'ordre du millier de dollars par an, voir plusieurs dizaines de milliers de dollars pour une licence perpetuelle [RIZ03]. Globalement, bien qu'ayant un coût variable en fonction des outils et modèles utilisés, la simulation est une technique de vérification à faible coût comparée aux techniques d'émulation qui seront prochainement présentées.

2.3.4) Emulation et prototypage matériel

Au niveau RTL, la simulation informatique précédemment présentée est certes très souple d'utilisation, possède une très grande efficacité de déboguage, mais a un inconvénient majeur : sa lente vitesse d'exécution. La vitesse de simulation des gros circuits sur les ordinateurs les plus

puissants du marché ne dépassera pas quelques dizaines de hertz. Lorsque l'on réalise une vérification au niveau système où que l'on développe des logiciels spécifiques au circuit, on doit faire fonctionner le circuit sur plusieurs millions, voire milliards, de cycles. A 10Hz, simuler dix millions de cycles nécessite environ douze jours ce qui est inacceptable. Pour pallier à cette limitation, on utilise des techniques d'émulation et de prototypage matériel.

2.3.4.1) Concept

Emulation et prototypage matériel reposent sur des machines spécifiques et reconfigurables, capables de reproduire le comportement physique d'un circuit avec une précision au niveau du cycle d'horloge. Ces appareils sont basés sur l'utilisation de composants offrant une reconfigurabilité, à savoir des FPGAs [RIZ03], des réseaux de processeurs spécialisés [LAR04], des composants spécifiques comme des FPICs (Field Programmable Interconnect Component) [APT02] ou des FPGAs modifiés et adaptés aux besoins de l'émulation [MEN03]. La principale caractéristique des émulateurs et plateformes de prototypage est d'offrir une grande vitesse d'exécution par rapport aux techniques précédemment présentées.

La nuance entre émulation et prototypage se situe au niveau de la mise en oeuvre et de la capacité de déboguage des machines. Les émulateurs sont avant tout des machines conçues pour réaliser un déboguage matériel rapide et efficace. Elles ont des flots de mise en oeuvre assez rapides, de l'ordre de quelques heures. Il est possible de réaliser plusieurs essais par jour, l'obtention d'une plateforme de vérification fonctionnelle nécessite quelques jours de travail, quelques semaines dans les cas les plus complexes. D'autre part, les émulateurs offrent une très grande observabilité, proche de celle des simulateurs HDL (tous les signaux). Globalement, un émulateur offre les mêmes caractéristiques qu'un simulateur HDL sauf que sa vitesse d'exécution est bien plus rapide et peut atteindre quelques mégahertz. Par contre, l'obtention de telles performances impose aux machines de recourir à de nombreux composants spécialement conçus pour le besoin. Ainsi, les émulateurs sont très onéreux, de l'ordre du million de dollars pour une licence annuelle [RIZ03].

Les plateformes de prototypage sont, quant à elles, des solutions basées FPGAs. Ces composants sont reconfigurables et simples à utiliser. De plus, ils sont très répandus et donc coûtent nettement moins cher que les composants spécifiques des émulateurs. Les plateformes de prototypage sont donc plus abordables que les émulateurs (ordre de grandeur : 100k$ pour une machine de grande capacité [RIZ03]) et offrent une excellente vitesse d'exécution (quelques dizaines de mégahertz), souvent supérieure à celle des émulateurs. Par contre, l'utilisation de composants génériques se paye sur l'observabilité. Les meilleures solutions permettent, au mieux, d'observer les registres du circuit sur une courte fenêtre temporelle. D'autre part, le partitionnement d'un circuit entre les différents FPGAs de la plateforme n'est pas aisé et engendre des temps de mise en oeuvre assez longs, pouvant atteindre plusieurs mois.

Emulateurs et plateformes de prototypage supportent de nombreux modes de fonctionnement. Certaines solutions sont meilleures que d'autres en fonction du mode. Bien choisir une machine d'émulation/prototypage matériel dépend donc, d'une part de l'étape de vérification à réaliser, et d'autre part du mode d'émulation sélectionné. Les prochains paragraphes présentent en détail l'ensemble des techniques de vérification associées à ces machines.

2.3.4.2) Flot de mise en oeuvre

Avant de détailler les différents types d'émulation et afin de mesurer leurs différents impacts sur la durée de mise en oeuvre il convient de présenter le flot de mise en oeuvre d'une plateforme d'émulation/prototypage matériel (figure 5).

La première étape du flot de prototypage est une étape d'adaptation du circuit à l'émulateur cible [SAS04]. Il faut identifier les composants non synthétisables qui demanderont un traitement

particulier. Typiquement, les descriptions VHDL/Verilog des mémoires utilisées par les équipes de conception ne sont pas synthétisables sur émulateur. Pour chacun de ces composants il faut donc réaliser un modèle spécifique à la machine utilisée.

En plus d'adapter le circuit à l'émulateur, il faut créer un banc de tests. Ce banc de tests peut être réalisé en matériel, en logiciel ou les deux. Globalement, un test logiciel est plus souple d'utilisation, plus rapide à écrire, plus efficace mais s'exécute plus lentement qu'un test matériel. Ce point sera détaillé dans les prochains paragraphes traitant des techniques de co-émulation. La partie matérielle synthétisable du banc de test est traitée comme le circuit à vérifier.

Une fois l'ensemble des composants matériels disponibles, le flot d'émulation/prototypage débute par l'étape de synthèse qui transforme la description HDL des parties matérielles du circuit en un assemblage de primitives matérielles élémentaires (bascules, verrous, portes logiques NAND, NOR, AND, OR, XOR, etc) supportées par l'émulateur. Le résultat de synthèse est couramment appelé «netlist».

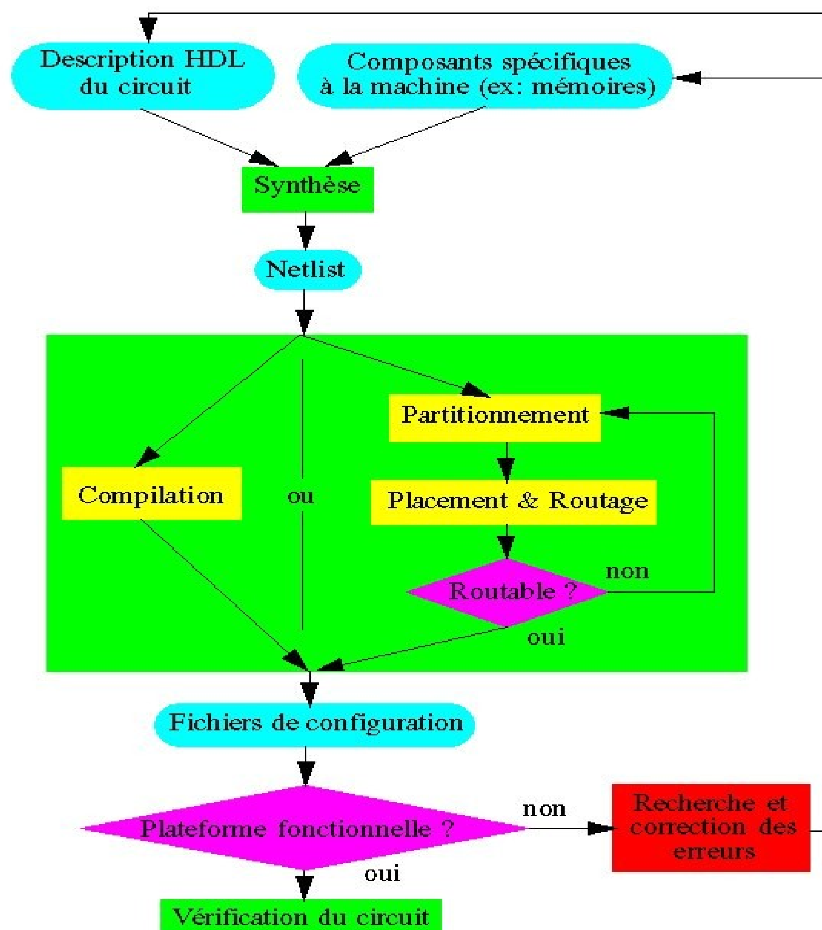


Figure 5 : Flot de mise en oeuvre d'une plateforme d'émulation/prototypage matériel

Vient ensuite la génération des fichiers de configuration de la machine d'émulation/prototypage. Les machines du commerce sont basées soit sur l'utilisation de FPGAs soit sur l'utilisation d'un réseau de processeurs. Par conséquent, pour les solutions basées processeur, la «netlist» va être transformée en plusieurs programmes pour chacun des processeurs, on parle d'étape de compilation. Le Palladium de Cadence est un exemple d'émulateur utilisant ce flot. Cette

phase de compilation est généralement assez rapide, de l'ordre de plusieurs dizaines de minutes à quelques heures. Ainsi, dans [CAD05], on apprend que compiler trente millions de portes logiques pour l'émulateur Palladium nécessite environ une heure sur un seul ordinateur.

Concernant les solutions orientées FPGA, le passage d'une «netlist» aux fichiers de configuration des FPGAs est beaucoup plus long et complexe. Tout d'abord, il faut partitionner le circuit, c'est à dire le découper en plusieurs sous-circuits, chaque sous-ensemble étant implémenté dans un FPGA. Cette étape est particulièrement ardue car il faut couper le circuit aux bons endroits afin de minimiser les signaux de communication entre FPGAs. Eventuellement, cette étape peut nécessiter une modification, une configuration de la plateforme [SAS04]. Certaines plateformes de prototypage comme le MP3 et le MP4 d'Aptix [APT02] sont constituées d'un fond de panier sur lequel on insère des FPGAs. En fonction du circuit, il faudra installer un certain nombre de FPGAs, et les placer de manière à optimiser les communications. Selon la solution de prototypage choisie, le partitionnement est plus ou moins automatisé ou assisté. Une fois le circuit partitionné, chaque FPGA devra être placé et routé. Il n'est pas rare qu'un mauvais choix de partitionnement fasse qu'un FPGA ne soit pas routable. Dans ce cas, il faut revoir le partitionnement et réitérer le processus de placement et routage. Ceci peut nécessiter plusieurs semaines, voire plusieurs mois, de travail [BIG04].

Une fois les fichiers de configurations générés, il faut vérifier que le circuit émulé fonctionne correctement. On compare alors les résultats donnés par un test de référence avec ceux préalablement obtenus par simulation du même test. S'ils concordent, la plateforme de vérification est prête sinon, une ou plusieurs erreurs sont à trouver. Elles peuvent provenir d'un des modèles spécifiques à la machine (erreur d'un modèle mémoire par exemple), de la phase de synthèse (mauvaise directive de synthèse ou erreur de l'outil utilisé), un problème de temps de propagation ou de composants de l'émulateur défectueux. Au final, les causes d'erreurs sont multiples et peuvent nécessiter une longue période d'investigations (plusieurs semaines dans le pire des cas). L'observabilité fournie par chaque machine impacte fortement ce paramètre. Une fois les erreurs détectées et corrigées, il ne reste plus qu'à réitérer le flot jusqu'à obtention d'une plateforme d'émulation/prototypage fonctionnelle.

2.3.5) Co-émulation

La co-émulation combine l'émulation/prototypage et la simulation. Il s'agit d'une technique couramment utilisée et caractérisée par des performances en vitesse souvent faibles vis à vis des possibilités des machines. Le concept de base est d'émuler/prototyper les parties du circuit dont la description au niveau RTL est terminée et de simuler, à un plus ou moins haut niveau d'abstraction, les composants manquants ainsi que le banc de tests. Pour cela, la plateforme d'émulation/prototypage retenue doit être capable de travailler conjointement avec un simulateur.

Cette technique permet l'utilisation au plus tôt des bénéfices en vitesse des émulateurs et plateformes de prototypage. En outre, recourir à la co-émulation offre également plusieurs avantages concernant le banc de tests. Celui-ci peut en effet être décrit à un assez haut niveau d'abstraction, implémenté en C, C++ ou SystemC, ce qui est plus simple et plus rapide à écrire qu'un test matériel synthétisable et offre souvent une efficacité bien supérieure. Cependant, l'environnement logiciel ne peut avoir la même vitesse d'exécution que l'environnement matériel donc, avec ce type d'émulation, les émulateurs/plateformes de prototypage fonctionnent à vitesse réduite. La vitesse globale du test va alors dépendre de plusieurs paramètres :

- vitesse d'exécution des environnements logiciel et matériel
- qualité de l'interface de communication
- nombre de synchronisations entre les deux environnements

Au niveau de la qualité de l'interface de communication, plusieurs aspects sont à considérer et impactent sur les performances. Certaines solutions fonctionnent avec un mécanisme dit «ping

pong» : lorsqu'un environnement travaille, l'autre est au repos, chacun des deux environnements se passant régulièrement la main. Les machines exploitant ce mécanisme fournissent alors une accélération minime, mais permettent une grande répétabilité des résultats. D'autres machines, au contraire, font fonctionner les deux environnements logiciel et matériel en parallèle, chaque environnement travaillant à son rythme, le plus rapide attendant parfois le plus lent. Cette solution est beaucoup plus rapide que la précédente mais peut poser des problèmes de répétabilité. Ce point sera détaillé dans le paragraphe concernant la co-émulation transactionnelle. Enfin, le nombre de synchronisations entre les deux environnements joue sur les performances, surtout lorque la solution de co-émulation fonctionne en «ping pong». Plus il est élevé et plus la plateforme est lente. Ce nombre de synchronisations varie en fonction du type de co-émulation, types qui vont être présentés dans les paragraphes suivants.

Pour finir sur les performances en fréquence, recourir à une technique de co-émulation permet d'obtenir des fréquences comprises entre quelques kilohertz et plusieurs mégahertz. Cette large gamme de fréquence dépend essentiellement du type de co-émulation utilisée, de la qualité de l'interface de communication et de la vitesse d'exécution de la machine. Plus il y aura des intèractions entre logiciel et matériel et plus le système sera lent. Dans le cas de plateforme d'émulation/prototypage mettant en jeu de nombreuses intèractions entre logiciel et matériel, le facteur limitant en vitesse sera le lien de communication, les qualités en vitesse des machines n'auront que très peu d'impact sur la vitesse globale du système. Au contraire, lorsqu'il y a peu d'intèractions, les qualités de l'émulateur impactent fortement la fréquence de fonctionnement du système émulé.

2.3.5.1) Co-émulation en mode "vecteurs de test"

Ce type de co-émulation est la plus simple du point de vue de l'environnement logiciel. Le circuit à valider est entièrement émulé. A chaque cycle d'horloge, on positionne les signaux d'entrée du circuit avec des valeurs particulières pour lesquels on a calculé les signaux de sortie qui seront obtenus au cycle d'horloge suivant. Ainsi, à chaque cycle d'horloge, on applique un vecteur d'entrée sur les ports entrant du circuit et on compare les valeurs des ports de sorties avec le vecteur de sortie prédéterminé.

Les vecteurs de test étant prédéterminés, l'environnement logiciel n'a aucun calcul à réaliser, il se cantonne juste à lire les vecteurs et à les envoyer à l'environnement matériel. Les performances sont donc directement liées au débit de l'interface de communication. L'ordre de grandeur des performances est de plusieurs dizaines de kilohertz.

Du point de vue applicatif, ce mode de co-émulation est utilisé principalement dans deux cas. D'une part, les outils d'ATPG (Automated Test Pattern Generation) permettent de générer des vecteurs de test qui seront utilisés par les testeurs, via des chaînes de scan, afin de vérifier les circuits intégrés fabriqués. Les vecteurs générés par ces outils ne sont pas tous corrects du premier coup, il faut donc les vérifier. La co-émulation en mode «vecteurs de test» est alors particulièrement bien adaptée.

D'autre part, afin d'identifier des bogues, on peut avoir à reproduire plusieurs fois un scénario de test. Dans une co-émulation, l'environnement logiciel est systématiquement un facteur limitant au niveau de la vitesse, les co-émulations en mode «vecteur de test» sont celles pour lesquelles l'environnement logiciel a un impact minimal. Ainsi, si l'on doit rejouer une co-émulation avec «synchronisation cycle à cycle», certaines solutions comme les plateformes de prototypage ZeBu [EVE05] proposent d'enregistrer les interactions entre environnements logiciel et matériel et de rejouer les scénarios en co-émulation de type «vecteurs de test». La charge de l'environnement logiciel est ainsi réduite et les scénarios sont rejoués plus rapidement.

2.3.5.2) Co-émulation avec synchronisation "cycle à cycle"

Dans une co-émulation avec synchronisation cycle à cycle, les horloges du circuit émulé sont générées dans l'environnement logiciel. Ainsi, l'environnement matériel est complétement piloté par l'environnement logiciel, les signaux d'entrée/sortie du circuit sont mis à jour à chaque cycle d'horloge. Ce type de co-émulation est très simple à mettre en oeuvre et offre tous les avantages de la co-émulation. Cependant, dans ce cas, le nombre d'intèractions entre les environnements logiciel et matériel est important. Ainsi, les performances au niveau vitesse sont fortement réduites. Typiquement, de telles co-émulations fonctionnent avec des fréquences comprises entre 1kHz et 100kHz. Les facteurs limitant sont principalement, en plus des nombreuses synchronisations, le nombre de signaux d'entrée/sortie, la bande passante de l'infrastructure de communication et la charge de calcul de la partie simulée. Dans ce cas, la vitesse d'exécution de la machine utilisée n'influence pas la fréquence de fonctionnement du système.

Au niveau applicatif, cette méthode de vérification est souvent utilisée avec des tests comportementaux. Les concepteurs décrivent avec un langage HDL (VHDL et/ou Verilog) le comportement d'un module de test. Le module de test n'est pas synthétisable et ne peut donc pas être émulé; il est exécuté sur un simulateur HDL.

D'autre part, les bancs de tests associés à ce type de co-émulation peuvent également utiliser un plus haut niveau d'abstraction. notamment, le banc de tests peut utiliser des outils spécifiques à la vérification comme Specman Elite [CAD06a]. La seule contrainte imposée est que le banc de tests génère toutes les horloges du circuit et se synchronise avec le circuit à chaque cycle d'horloge.

La co-émulation «cycle à cycle» permet la mise en oeuvre d'un test à performance moyenne avec un temps de développement du banc de tests relativement court. Les performances en vitesse ne sont pas suffisantes pour réaliser une vérification au niveau système mais cette technique est très utile pour la vérification au niveau composant qui nécessite un gros effort de déboguage matériel. Dans l'industrie, l'ensemble des émulateurs supporte et plébiscite cette technique. Par contre, bien que supportée par les plateformes de prototypage, elle ne présente que peu d'intérêt dans ce cas, du fait de leur faible capacité de déboguage et que leur avantage potentiel en vitesse n'est pas vraiment exploité dans ce mode.

2.3.5.3) Co-émulation avec synchronisation "clairsemée"

Dans la co-émulation à synchronisation cycle à cycle, les signaux d'entrée/sortie du circuit émulés sont mis à jour à chaque cycle d'horloge, ce qui impose de nombreuses communications entre émulateur et ordinateur. Cependant, les données sur les signaux d'entrée/sortie ne changent pas systématiquement à chaque cycle d'horloge donc une partie de ces communications n'est pas nécessaire et consomme de la bande passante inutilement ce qui, au final, ralentit le système. Une co-émulation clairsemée vise à réduire ces communications superflues. Pour cela, les horloges du circuit ne sont plus gérées par l'environnement logiciel mais par l'émulateur. Des signaux de contrôle servent alors à synchroniser émulateur et ordinateur.

La mise en oeuvre de cette méthode n'est pas très aisée et demande certaines conditions pour être réalisable. En effet, pour assurer les points de synchronisation, il faut pouvoir déterminer les instants de rendez-vous, ce qui peut se révéler très complexe. Globalement, cette technique n'est applicable que dans le cas où l'un des deux environnements matériel ou logiciel est le maître de la commnucation et l'autre un esclave.

Pour mieux comprendre le principe, voici un exemple. Imaginons un opérateur matériel qui consomme une donnée, la traite en dix cycles puis, une fois le traitement réalisé retourne le résultat et consomme une nouvelle donnée. Dans ce cas on peut utiliser une co-émulation clairsemée et synchroniser le matériel avec le logiciel tous les dix cycles d'horloge. Le matériel est alors le maître de la communication, tous les dix cycles, il impose l'échange de données.

Au niveau des performances, cette solution peut atteindre plusieurs centaines de kilohertz, voire quelques mégahertz dans le cas de système nécessitant un faible nombre d'interactions entre émulateur et ordinateur. La vitesse d'exécution de la machine utilisée impacte fortement la fréquence de fonctionnement du système.

Pour conclure, bien qu'offrant de bonnes performances, la co-émulation à synchronisation clairsemée est peu utilisée vue ses contraintes de mise en oeuvre liées à des développements spécifiques. Cependant, elle peut convenir à toutes les étapes de vérification.

2.3.5.4) Accélération

L'accélération est une co-émulation où l'utilisation de l'émulateur est transparente vis à vis de l'utilisateur. On parle d'accélération lorsque l'on travaille avec un circuit et un banc de test décrit en langage matériel (VHDL et/ou Verilog) et qu'une partie du code est synthétisable alors que l'autre est comportementale mais non synthétisable. Lorsque l'utilisateur va lancer le flot de mise en oeuvre, le code synthétisable sera émulé et le code non synthétisable sera simulé à l'aide d'un simulateur HDL. Pendant les phases d'exécution des tests, l'utilisateur pilotera l'émulateur depuis le simulateur et ne travaillera uniquement que dans l'environnement de simulation.

En pratique, l'accélération est une technique de vérification au niveau composant, comme la co-émulation à synchronisation cycle à cycle. Les performances sont de l'ordre de quelques kilohertz. Seuls les émulateurs supportent cette technique.

2.3.5.5) Co-émulation transactionnelle

La principale limitation de la co-émulation à synchronisation cycle à cycle précédemment présentée réside dans le nombre élevé de synchronisations entre environnement logiciel et environnement matériel. La co-émulation clairsemée cherche à réduire cet impact mais, la gestion des signaux échangés est souvent très complexe. L'idée de base de la co-émulation transactionnelle est de réduire ce nombre de synchronisations au minimum nécessaire, comme dans une co-émulation clairsemée mais, en s'affranchissant de la complexité de la gestion des signaux d'interface. Pour cela, la communication va être gérée à un plus haut niveau d'abstraction. Ainsi, on va utiliser un mécanisme de communication basé sur des évènements et non plus des signaux, les messages échangés sont appelés des transactions. Une transaction transmet des actions, des commandes au lieu de dérouler un protocole de communication au niveau signal (figure 6).

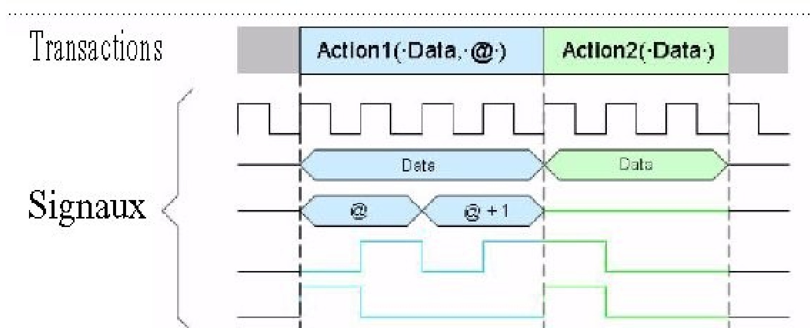


Figure 6 : Comparaison entre niveau signal et niveau transactionnel

Par exemple, un test exécuté dans l'environnement logiciel veut récuper le contenu d'un registre du circuit émulé. Il va alors lancer une transaction qui comportera les informations «lire

registre x». Lorsque le matériel va recevoir ce message, il va alors dérouler tout le protocole de bus nécessaire pour accéder à la donnée désirée (ce qui peut nécessiter de nombreux cycles d'horloge) puis va répondre au test logiciel par une transaction contenant les informations «registre x : valeur». La transaction a donc complétement abstrait le protocole de communication de l'interface d'entrée/sortie du circuit à vérifier.

Lors de la mise en application de ce concept, les utilisateurs doivent recourir à des modules matériels capables de comprendre les transactions et de les convertir en signaux et vis et versa. Ces convertisseurs matériels sont nommés «**transacteurs**» [KUD01], [SCE04]. Leur réalisation est complexe et représente la principale difficulté des équipes de vérification désireuses de recourir à cette performante technique de co-émulation.

L'architecture d'une co-émulation transactionnelle comporte un environnement logiciel et un environnement
matériel communiquant à l'aide de canaux de communication. Une norme nommée SceMi (Standard CoEmulation Modeling Interface) a standardisé ces canaux. Ce point sera détaillé dans la section «2.7.1) Interface de co-émulation SceMi».

Dans le cas d'application engendrant peu de transactions, ce type de co-émulation peut atteindre les fréquences maximales offertes par les différents émulateurs et plateformes de prototypage. La gamme de fréquence varie donc de quelques kilohertz (grand nombre de transactions, environnement logiciel complexe, faible débit de l'interface de communication) à plusieurs mégahertz (peu de transactions, important débit de l'interface de communication, grande vitesse d'exécution de la machine).

Les applications les mieux adaptées à ce type de co-émulation sont les circuits gérant des flux de données comme les circuits de télécommunication.

De plus, il est à noter que la contrainte de développement des transacteurs peut être réduite par l'utilisation d'une bibliothèque de transacteurs. Développer et valider la bibliothèque nécessite une longue période mais, une fois celle-ci développée, la durée de mise en place d'une plateforme de vérification transactionnelle devient très courte.

Enfin, ce type de co-émulation peut avoir un inconvénient plus ou moins génant dans certaines applications en fonction de la méthode de synchronisation émulateur/ordinateur utilisée. Le problème est celui de la reproduction des scénarios. En effet, les solutions qui font fonctionner émulateur et environnement logiciel en parallèle ne garantissent pas une répétabilité au cycle prêt des scénarios de tests contrairement aux solutions de type «ping pong» [CAD05]. Ainsi, en rejouant un scénario, si l'environnement logiciel a une charge de travail différente de celle rencontrée en jouant le test pour la première fois, les transactions ne seront pas échangées au même instant (au même cycle d'horloge) du point de vue du matériel. Ainsi, certaines erreurs rencontrées durant la première vérification risquent de ne pas se reproduire. Par conséquent, l'utilisation de la co-émulation transactionnelle avec des solutions à fonctionnement logiciel/matériel parallèle offre de très bonnes performances en vitesse mais, nécessite une bonne compréhension du système de vérification pour prévenir des problèmes de reproductibilité. Notamment, écrire des transacteurs capable de suspendre le matériel lorsque le logiciel est trop lent à répondre peut prévenir des problèmes de reproductibilité.

Pour conclure, étant donné la contrainte rencontrée, recourir à une co-émulation transactionnelle est bien adapté pour vérifier l'intégration du système et pour tester les performances des algorithmes.

2.3.6) Emulation avec banc de test intégré ("Self Test Bench" - mode STB)

Dans une émulation avec banc de test intégré, connue dans l'industrie comme mode STB (Self Test Bench), le circuit à vérifier et son banc de test sont tous deux des composants matériels émulés. L'émulateur ou la plateforme de prototypage fonctionne alors de manière complétement

autonome.

Dans ce mode, les machines fonctionnent à pleine vitesse. Avec un émulateur, la fréquence de fonctionnement est comprise entre plusieurs centaines de kilohertz et quelques méga-hertz. Avec une plateforme de prototypage, la gamme de fréquence s'étend de quelques méga-hertz à plusieurs dizaines (voire centaines pour les meilleurs solutions) de méga-hertz. Cette technique est donc particulièrement bien indiquée pour les vérifications nécessitant de longues séquences de test à savoir la vérification au niveau système. Cette technique est également bien adaptée pour la mise au point des logiciels embarquées. Typiquement, on utilise un émulateur en mode STB pour la vérification au niveau système et une plateforme de prototypage en mode STB pour le développement des logiciels embarqués.

La principale difficulté de cette technique d'émulation réside dans le développement du banc de test synthétisable. Cela représente une grosse charge de travail car il faut développer un composant matériel de test spécifique et le valider.

2.3.7) Emulation avec dépendances extérieures ("In Circuit Emulation" - mode ICE)

Une émulation avec dépendances extérieures, connue dans l'industrie comme mode ICE (In Circuit Emulation), est une émulation comparable à celle en mode STB sauf qu'ici, l'émulateur est connecté à un environnement physique extérieur, le circuit fonctionne alors en temps réel. Par exemple, pour valider un circuit de traitement vidéo, on peut utiliser une émulation en mode ICE. Le circuit est implémenté dans un émulateur ou une plateforme de prototypage, la machine de vérification étant reliée à une caméra vidéo ainsi qu'à un moniteur. Un tel mode permet de tester le circuit dans son environnement final.

Du point de vue applicatif, utiliser un tel mode exige souvent de très hautes performances en vitesse. Ce mode est donc majoritairement utilisé avec des plateformes de prototypage, essentiellement pour le développement des logiciels embarqués ou le développement des logiciels associés au circuit (drivers).

Par rapport au mode STB, cette technique réduit la contrainte de développement d'un banc de test synthétisable vu que l'on utilise le véritable environnement du circuit. Par contre, il n'est pas toujours possible de réaliser un prototype suffisament rapide et il faut souvent pouvoir diminuer la fréquence de fonctionnement de l'environnement.

D'autre part, la contrainte temps réel du mode ICE fait qu'il n'est pas possible de ralentir la plateforme pour des besoins de déboguage. Couramment une mémoire est utilisée pour stocker les états consécutifs de quelques signaux. Quand cette mémoire est pleine, la trace enregistrée est transférée sur un ordinateur pour exploitation. Ainsi, on peut au mieux obtenir une courte fenêtre d'observation temporelle de quelques signaux. Ceci limite fortement le déboguage et explique pourquoi cette technique est avant tout utilisée pour le développement et la validation des logiciels associés au circuit.

2.4) Etat de l'art des machines d'émulation/prototypage matériel utilisées dans l'industrie

Les performances des différentes techniques d'émulation et prototypage matériel sont directement liées aux performances des machines utilisées. Si l'on veut comparer ces techniques, il est impossible de les considérer indépendamment des machines d'émulation et prototypage. Les deux prochains paragraphes vont donc dresser un état de l'art dans ce domaine afin de pouvoir ensuite mesurer le niveau couverture des besoins de vérification.

2.4.1) Machines d'émulation

Le marché de l'émulation est dominé par deux grosses compagnies, Cadence et Mentor Graphics, qui ont développé ou racheté l'ensemble des solutions existantes. Quelle que soit la solution considérée, toutes ces machines sont très onéreuses (surtout les plus récentes qui sont les plus performantes), les licences annuelles se négocient en millions de dollars [RIZ03][LAR04].

2.4.1.1) Solutions Mentor Graphics : Celaro et VStation

La société Mentor Graphics commercialise deux solutions couramment utilisées, les machines Celaro et VStation.

La machine Celaro est basée sur l'utilisation de plusieurs centaines de FPGAs spécifiques aux besoin de l'émulation. Ces FPGAs sont nommés AVC (Accelerated Verification Chip), et ont été développés dans l'objectif d'offrir une grande observabilité contrairement aux FPGAs standards. La machine est capable d'émuler jusqu'à trente millions de portes logiques dans sa dernière version (CelaroPro). Elle supporte toutes les techniques d'émulation à l'exception de la co-émulation transactionnelle. En mode ICE ou STB, cette machine permet d'atteindre typiquement des fréquences de l'ordre de 100kHz. La capacité de déboguage est excellente puisque l'ensemble des signaux du circuit émulé sont observables. Enfin, la durée de mise en oeuvre est assez courte. Il faut compter quelques heures pour réaliser synthèse, placement et routage d'un circuit. On peut réaliser un essai par jour ce qui permet l'obtention d'une plateforme de vérification fonctionnelle en quelques jours de travail.

VStation est une autre solution, basée sur des FPGAs standards [MEN06]. Cette machine se décline en deux modèles, l'un orienté pour les émulations en mode STB et ICE (VStation PRO) l'autre, orienté pour les co-émulations transactionnelles (VStation TBX). Les capacités de déboguage sont comparables à celle de Celaro. Le modèle TBX peut fonctionner en accélérateur (utilisation transparente de l'émulateur depuis un environnement de simulation), ce qui décuple la facilité de déboguage vu qu'il n'est pas nécessaire de maitriser plusieurs environnements de déboguage. La durée de compilation est d'environ cinq millions de portes par heure (à l'aide d'une ferme d'ordinateurs) ce qui permet une mise en oeuvre très rapide. La machine a une capacité de cent vingt millions de portes logiques et offre des fréquences d'environ 1MHz en mode ICE/STB et jusqu'à 500kHz en co-émulation.

Enfin, ces deux machines sont en fin de vie. La société Mentor Graphics annonce depuis quelque temps l'arrivée d'une nouvelle machine, Veloce, combinant le meilleur de la technologie Celaro et VStation. Cependant aucune présentation publique n'est encore disponible.

2.4.1.2) Solutions Cadence : Palladium et Xserver

De son coté, la société Cadence propose également deux familles d'émulateurs, la famille Palladium et la famille Xserver.

Présent sur le marché depuis Janvier 2002, l'émulateur Palladium est basé sur l'utilisation parallèle de plusieurs dizaines de milliers de microprocesseurs spécialement conçus pour les besoins de l'émulation [CAD06b]. La plus petite machine permet d'émuler jusqu'à huit millions de portes logiques. En combinant plusieurs machines, on peut étendre la capacité d'émulation jusqu'à deux cents cinquante-six millions de portes. Cette machine offre dans sa première version, des fréquences de fonctionnant allant jusqu'à 2MHz, 4Mhz dans sa deuxième version. Toutes les techniques d'émulation et de co-émulations sont supportées. La capacité de déboguage est excellente, comparable à celle des solutions concurrentes. Enfin, la durée de mise en oeuvre est aussi très bonne, un simple ordinateur permet de compiler jusqu'à cinq millions de portes logiques par heure. Un des gros avantage de Palladium est d'être une machine multidomaines et multi-

utilisateurs, c'est à dire que plusieurs circuits peuvent être émulés en même temps sur la même machine. Cela permet d'optimiser l'utilisation de la machine.

Quant à elle, la machines Xserver est à la fois basée sur des FPGAs et sur un réseau de co-processeurs. Les FPGAs sont utilisés pour implémenter des centaines de milliers de co-processeurs. Cette architecture nommée RCC (ReConfigurable Computing) procure à cette machine un fonctionnement comparable à celui d'un simulateur orienté évènement qui parallélise ses calculs. Grâce à cette technologie, les mêmes fichiers obtenus après compilation servent à la fois au simulateur Xsim et à l'émulateur Xserver. Ainsi, il est aisé de passer de simulation à émulation puis de revenir en simulation sans rien avoir à recompiler. Xserver est la référence en machine d'accélération. Toutes les techniques d'émulation et de co-émulation sont supportées, la fréquence maximale du système est de 1MHz. La capacité de déboguage est comparable à celle des autres solutions. La mise en oeuvre est assez rapide (quelques heures pour une compilation) mais nécessite, par contre, une ferme d'ordinateurs.

2.4.1.3) Synthèse

Le tableau suivant résume les principales caractéristiques des émulateurs du marché.

Machines	***Celaro Pro***	***VStation***	***Palladium II***	***Xserver***
Déboguage	Tous les signaux	Tous les signaux	Tous les signaux	Tous les signaux
Capacité d'émulation en portes logiques	30M	120M	256M	50M
Vitesse de compilation	500k portes par heure	5M portes par heure	30M portes par heure	Non communiquée
Puissance de compilation	Un ordinateur	Ferme d'ordinateurs	Un ordinateur	Ferme d'ordinateurs
Multi-utilisateurs	Oui	Non	Oui	Oui
Co-émulation «cycle à cycle»	1kHz 20kHz	5kHz 200kHz	1kHz 20kHz	10kHz 20kHz
Co-émulation transactionnelle	Non	20kHz 100kHz	10kHz 200kHz	10kHz 150kHz
Emulation ICE/STB	100kHz 500kHz	500kHz 1MHz	100kHz 3MHz	100kHz 1MHz

Tableau 2 : Tableau comparatif des émulateurs utilisés dans l'industrie

Sur ce tableau récapitulatif, on remarque que Celaro, la machine la plus ancienne, est dépassée par les autres solutions. Par contre, les performances des machines Palladium, Xserver et VStation sont très proches. La différence entre ces machines se fait surtout sentir sur la durée de compilation et la puissance de calcul nécessaire.

2.4.2) Plateformes de prototypage

Le marché de la plateforme de prototypage industriel est partagé entre cinq entreprises : Eve, Mentor Graphics, ProDesign, Hardi et Flexody.

2.4.2.1) Solutions Eve : ZeBu

La société Eve propose une famille nommée ZeBu [EVE06]. La première machine fut le modèle ZeBu-ZV, une carte de prototypage utilisant deux FPGAs Xilinx Virtex2, la carte s'installant sur le bus PCI d'un PC. Ensuite est apparu le modèle ZeBu-XL, basé sur l'utilisation massive de FPGAs Xilinx Virtex2, offrant à la machine une capacité de prototypage de cinquante millions de portes logiques (64 FPGAs). Enfin, suite à la sortie d'une nouvelle génération de FPGA (Xilinx Virtex4), la carte ZeBu-ZV a été remplacé par le ZeBu-UF, une carte PCI intégrant jusqu'à quatre FPGAs et offrant une capacité de prototypage de six millions de portes.

Quelle que soit la machine Eve considérée, ces machines ont toutes, les mêmes capacités de déboguage, à savoir, une visibilité sur l'ensemble des registres et mémoires du circuit. L'utilisateur à le choix entre effectuer un déboguage dynamique (sélection à la volée des signaux observés) et lent (fréquence à quelques kilohertz) ou un déboguage statique (sélection à la compilation des signaux observés). En mode statique, le déboguage n'altère pas la vitesse d'exécution de la plateforme mais, seule une courte fenêtre temporelle est observable.

Concernant la mise en oeuvre, il faut dans un premier temps synthétiser le circuit (une heure par million de portes) puis partionner, placer et router chaque FPGA (une heure par FPGA). Le partitionnement est l'étape difficile dans le flot de mise en oeuvre des plateformes de prototypage, il faut souvent de nombreuses itérations avant d'obtenir un bon partionnement. Les outils Eve assistent fortement l'utilisateur dans cette étape et font que les machines Eve sont les plus faciles à mettre en oeuvre. Néanmoins, on ne peut pas faire plus d'une itération de partitionnement, placement et routage par jour de travail, ce qui impose plusieurs semaines avant l'obtention d'une plateforme de vérification fonctionnelle dans les cas les plus complexes.

Les machines ZeBu supportent l'ensemble des techniques d'émulation et de co-émulation et offrent une fréquence d'émulation très élevée, jusqu'à 10MHz pour les co-émulations et jusqu'à 100MHz pour les émulations STB avec ZeBu-UF.

2.4.2.2) Solutions Mentor Graphics : MP3 et MP4

Courant 2005, Mentor Graphics a racheté les outils de la société Aptix qui commercialisait depuis 1995 deux familles, la famille MP3 et la famille MP4 [APT06]. Ces deux solutions sont en fin de vie mais sont encore très répandues dans l'industrie. Elles se présentent comme un fond de panier sur lequel l'utilisateur vient connecter des composants, essentiellement des FPGAs et des modules mémoire. Cependant, il est possible de connecter de nombreux autres ASICs sur une machine Aptix. Afin de relier ensemble les différents modules connectés, la machine utilise des composants d'interconnection programmables nommés FPIC (Field Programmable Interconnect Component). Le fond de panier du MP3 (respectivement MP4) est divisé en trois (respectivement quatre) régions, chaque région étant reliée aux autres via les FPICs.

Au niveau de la capacité, les machines Aptix sont limitées par le nombre de FPGAs que l'on peut installer, le modèle MP3 permet de connecter jusqu'à douze FPGAs, le modèle MP4 accepte jusqu'à vingt FPGAs ce qui représente une capacité d'environ douze millions de portes pour le modèle MP3 et vingt millions de portes pour le MP4.

Les capacités de déboguage sont quant à elles médiocres. Seul un nombre limité de signaux sont observables à l'aide d'un analyseur logique : cette machine n'est pas adaptée au déboguage matériel. De plus, ce manque d'observabilité rend la mise en oeuvre de la plateforme très difficile. Le temps de cycle par FPGA est commun à l'ensemble des solutions de prototypage (environ une heure pour la synthèse et une heure pour le placement et routage), par contre le partitionnement n'est que très partiellement assisté et est très laborieux. Des outils comme Certify [SYN06] sont souvent utilisés dans l'industrie pour faciliter cette étape. Cependant, malgré ce type d'outil, le partitionnement reste difficile. Il faut compter plusieurs mois de travail avant l'obtention d'une

plateforme fonctionnelle [BIG04].

Les machines Aptix supportent en théorie tous les modes d'émulation et de co-émulation cependant, en pratique, de part les propriétés des machines, seuls les mode ICE et STB sont utilisés. Enfin, Aptix annonce des émulations en mode STB avec des fréquences pouvant aller jusqu'à 120MHz.

2.4.2.3) Solutions ProDesign : ChipIt

De son coté, la société ProDesign propose également une solution de prototypage avec sa famille ChipIt [PRO06]. Les plateformes ChipIt reposent sur l'utilisation de plusieurs FPGAs Xilinx, la dernière évolution du ChipIt (Platinium V4) exploite les FPGAs Virtex4.

Le Platinium V4 peut intégrer jusqu'à vingt et un FPGAs ce qui représente une capacité de prototypage d'environ vingt et un millions de portes logiques.

L'ensemble des techniques d'émulation et de co-émulation sont supportées. ProDesign annonce de hautes performances en vitesse, jusqu'à 100MHz en mode ICE et 200MHz en mode STB. Les capacités de déboguage sont assez pauvres (les mêmes que celles des Aptix) sur tous les modèles sauf sur la dernière évolution, le Platinium V4. Cette machine, associée à l'outil Siloti de Novas [NOV06] offre une visibilité sur l'ensemble des signaux du circuit (seuls les registres sont observés, le logiciel Siloti reconstruit les signaux combinatoires). Par contre, la capture de l'état des registres du circuit impacte la vitesse d'exécution de la plateforme, le système est arrêté entre chaque cycle d'horloge, la fréquence de prototypage tombe alors à environ 1kHz.

Enfin le partitionnement est ici encore l'étape difficile, les contraintes de mise en oeuvre sont similaires à celles des machines Aptix (au mieux une itération de partionnement placement et routage par jour) et, il faut compter plusieurs semaines, voire mois, de travail pour obtenir une plateforme fonctionnelle.

2.4.2.4) Solutions Hardi : Haps

La société Hardi propose une famille de plateformes de prototypage nommée HAPS [HAR06]. Cette famille est constituée d'un ensemble de cartes intégrant des FPGAs Xilinx, les cartes s'interconnectant les unes avec les autres. Cette solution est la plus flexible du marché, celle qui offre le plus de souplesse quant à l'interconnection entre FPGAs. Cela se traduit par de très hautes performances en fréquence, jusqu`à 200MHz en mode ICE ou STB. Par contre, cette grande flexibilité impacte la mise en oeuvre. En plus de la classique difficulté au niveau du partitionnement entre FPGAs, l'utilisateur doit lui même choisir comment il va interconnecter les cartes, combien de bits sont alloués pour chaque lien de communication. La suite logicielle est très pauvre. Au final, cette solution est d'un coté la plus performante en vitesse d'exécution mais d'un autre coté la plus difficile à mettre en oeuvre. Il faut compter plusieurs mois de travail avant l'obtention d'une plateforme fonctionnelle. Etant donné que l'on peut interconnecter autant de cartes que l'on veut, la capacité de la machine est illimitée. Cependant, plus on ajoute des cartes et plus les performances en vitesse vont diminuer. D'autre part, cette solution ne supporte que les émulations en mode ICE et STB, les capacités de déboguage sont quasi nulles, comparables à celles des machines Aptix.

2.4.2.5) Solutions Flexody : FlexCube

La société Flexody propose une solution nommée FlexCube [FLE06]. Cette solution est encore basée sur l'utilisation de plusieurs FPGAs Xilinx. Le système se compose de multiples cartes filles que l'on assemble sur une carte mère. La topologie du système permet d'obtenir des fréquences jusqu'à 100MHz en émulation STB ou ICE. La machine permet l'assemblage d'au plus seize FPGAs, ce qui lui procure une capacité de prototypage de dix huit millions de portes logiques.

Selon Flexody, toutes les techniques d'émulation et de co-émulation sont (vont) être supportées. Pour la mise en oeuvre, le partionnement est là encore le point difficile. Difficulté et durée de mise en oeuvre sont comparables à celles des machines Aptix et ProDesign.

La capacité de déboguage est limitée. Flexody fournit des modules matériels de déboguage à insérer dans le circuit, ce qui permet l'observation de quelques signaux à l'aide d'un analyseur logique embarqué.

2.4.2.6) Synthèse

Le tableau suivant résume les principales caractéristiques des différentes plateformes de prototypage rencontrées dans l'industrie. L'ensemble des machines offrent des performances en vitesse comparables. La grosse différence est au niveau de la mise en oeuvre et de la capacité de déboguage où les machines ZeBu proposent de meilleures performances.

Machines	ZeBu	ChipIt Platinium V4	MP4 et MP3	FlexCube	HAPS
Déboguage	- Tous les registres - Tous les signaux avec Siloti (vitesse max 1kHz)	- Quelques signaux sur analyseur logique - Tous les signaux avec Siloti (vitesse max 1kHz)	Quelques signaux sur analyseur logique	Quelques signaux sur analyseur logique	Quelques signaux sur analyseur logique
Capacité en portes logiques	- 50M (ZeBu-XL) - 6M (ZeBu-UF)	21M	- 12M (MP3) - 20M (MP4)	18M	Illimitée
Partitionnement assisté	++++	++	++	++	+
Durée de mise en oeuvre	Quelques semaines	Quelques mois	Quelques mois	Quelques mois	Plusieurs mois
Co-émulation «cycle à cycle»	5kHz 500kHz	5kHz 500kHz	1kHz 30kHz	Oui Pas de chiffre communiqué	Non
Co-émulation transactionnelle	200kHz 20MHz	200kHz 20MHz	25kHz 8MHz	Annoncée	Non
Emulation ICE/STB	1MHz 100MHz	1MHz 100MHz	1MHz 100MHz	10Mz 100MHz	10Mz 200MHz

Tableau 3 : Tableau comparatif des plateformes de prototypage utilisées dans l'industrie

2.5) Synthèse

Chacune des différentes techniques présentées précédemment possède ses propres caractéristiques et est adaptée à une ou plusieurs étapes du flot de conception et vérification des systèmes monopuces.

La vérification formelle est très flexible mais trop complexe à mettre en oeuvre à bas niveau d'abstraction. Cette technique est bien adaptée pour les premières étapes de vérification qui se passent à un haut niveau d'abstraction. Cette thèse adresse les problèmes de vérification des circuits décrit au niveau RTL et, comme la vérification formelle est mal adaptée à ce problème, cette technique ne sera pas plus développée.

Le tableau suivant résume les quatre aspects fondamentaux (vitesse d'exécution, possibilités de déboguage matériel, coût financier de la vérification et difficultée de mise en oeuvre) des différentes techniques de vérification RTL couramment utilisées dans l'industrie.

La simulation offre une grande souplesse, simplicité d'utilisation, grande observabilité, un temps de mise en place assez court. De part ses qualités, la simulation est l'outil fondamental dans la conception des systèmes monopuces. Cependant, cette technique a une faible vitesse d'exécution au niveau RTL ce qui limite son utilisation. Elle est en revanche bien adaptée pour la vérification au niveau composant.

Techniques	**Fréquences**	**Mise en oeuvre rapide**	**Coût modéré**	**Déboguage matériel**	**Applications**	**Machines**
Simulation	1Hz 10kHz	++++	++++	++++	Composants	NcSim, ModelSim
Accélération	1kHz 50kHz	+++	+	++++	Composants Intégration	Xserver, Palladium, VStation
Co-émulation «cycle à cycle»	1kHz 50kHz	+++	+	+++	Composants Intégration	Toutes
Co-émulation «vecteur de test»	10kHz 100kHz	+++	+	+++	Tout	Toutes
Emulateur en co-émulation transactionnelle	10kHz 1MHz	++	+	+++	Composants Intégration Système	Xserver, Palladium, VStation
Plateforme de prototypage en co-émulation transactionnelle	10kHz 10MHz	+	++	+	Système logiciels embarqués	ZeBu, ChipIt
Emulateur en mode ICE/STB	100kHz 2MHz	+++	+	+++	Intégration Système Logiciels embarqués	Palladium, Celaro, Vstation, Xserver
Plateforme de prototypage mode ICE/STB	500kHz 200MHz	+	++	+	Système Logiciels embarqués	MP3, MP4, ChipIt, ZeBu, FlexCube, HAPS

Tableau 4 : Comparaison des techniques de vérification du RTL

L'émulation, quant à elle, a des possibilités très riches. Les techniques d'accélération et de co-émulation, associées à un émulateur offrent souplesse, observabilité et une vitesse d'éxécution supérieure par rapport à la simulation moyennant un flot de mise en oeuvre plus ou moins complexe et un important budget de fonctionnement. Les différentes techniques de co-émulation peuvent intervenir à toutes les étapes de validation travaillant au niveau RTL. Les techniques de co-

émulations, associées à une plateforme de prototypage ont un intérêt plus limité. Les plateformes de prototypage ne permettent pas un déboguage facile du matériel. Ces plateformes offrent avant tout une grande vitesse d'exécution. Seules les co-émulations transactionnelles sont utiles avec ce type de machine car elles ne contrecarrent pas le seul apport de la plateforme de prototypage, à savoir une grande vitesse d'exécution. Ce point sera repris et détaillé dans la section «2.7.2) Flot de conception TLM». Enfin, la mise en oeuvre d'une co-émulation transactionnelle sur une plateforme de prototypage nécessitera plusieurs semaines, voire mois, de travail de part la complexité de mise en oeuvre de ce type de machine. Du point de vue financier, le coût restera modéré. Cette technique est adaptée à la vérification au niveau système ainsi qu'au développement des logiciels embarqués.

L'émulation matérielle, en mode STB ou ICE est une solution offrant vitesse et observabilité avec une durée de mise en oeuvre acceptable (quelques jours) mais souffre par le coût financier des machines. L'émulation en mode ICE nécessite également un investissement puisqu'il faut réaliser des bancs de tests synthétisables. Ces techniques sont adaptées à la vérification de l'intégration ainsi qu'à la vérification fonctionnelle du sytème. Elles peuvent également servir au développement des logiciels embarqués mais n'offrent pas des performances aussi bonnes que celles obtenues avec une plateforme de prototypage.

Enfin, le prototypage virtuel en mode STB ou ICE est la technique de vérification offrant les meilleures possibilités au niveau de la vitesse d'éxécution avec un buget de fonctionnement modéré. Cette technique souffre de ses possibilités de déboguage quasi nulles et une mise en place complexe et, par conséquent, longue. Cette technique est bien adaptée au développement des logiciels embarqués, ainsi qu'à la mesure des performances du système.

2.6) Nécessité de faire coopérer les différentes techniques de vérification

Dans le tableau de comparaison des techniques de vérification du précédent paragraphe, on remarque qu'aucune technique et qu'aucune machine ne couvre parfaitement l'ensemble des besoins des différents niveaux de vérification. Les machines les plus polyvalentes sont les émulateurs, c'est à dire les machines les plus chères. De plus, même en utilisant un émulateur qui couvre, avec plus ou moins d'efficacité, les besoins de vérification aux niveaux composant, intégration et système, les performances en vitesse restent souvent insuffisantes pour le développement des logiciels embarqués. Pour couvrir l'ensemble des besoins avec un maximum d'efficacité, il faut donc obligatoirement utiliser plusieurs techniques d'émulation et au moins deux machines différentes.

De plus, aucune technique de vérification n'est infaillible, la vérification au niveau composant peut laisser passer une erreur matérielle qui ne se manifestera que pendant la phase de développement des logiciels embarqués sur plateforme de prototypage. Or, identifier un bogue matériel à l'aide d'une plateforme de prototypage requiert énormément de temps vu les faibles capacités d'observabilité qui caractérisent ces machines et les astuces de déboguage qu'il faut employer. Dans ce cas, l'idéal serait de revenir en émulation ou en simulation afin d'utiliser une technique mieux adaptée au problème de déboguage rencontré.

Idéalement, il serait pratique d'utiliser une seule et unique machine pour l'ensemble de la vérification, d'avoir une machine qui offre à la fois une très grande vitesse d'exécution, une excellente capacité de déboguage (comparable à celles des simulateurs HDL), supporte l'ensemble des techniques de vérification et reste financièrement abordable. Malheureusement, l'état de l'art a montré que cette machine n'existe pas.

2.6.1) Sélection du meilleur compromis techniques/machines

Actuellement, aucune solution d'émulation et prototypage matérielle utilisée dans l'industrie ne supporte l'ensemble des modes de fonctionnement avec de bonnes performances. En outre, aux limitations intrasèques de ces techniques s'ajoutent les contraintes propres aux propriétés et

caractéristiques des machines sur lesquelles elle sont basées.

Finalement, à chaque étape de vérification, il faut sélectionner la meilleure combinaison technique de vérification et machine la supportant de façon à optimiser les facteurs temporels et économiques.

Ce choix est souvent très difficile vu les nombreux paramètres à considérer : capacité, environnement de vérification et capacité de communication, coût financier, durée de mise en oeuvre, vitesse d'exécution, maturité du circuit, flexibilité, observabilité, contrôlabilité, reproductibilité et difficulté d'utilisation.

Dans un premier temps, il faut lister les machines ayant une capacité suffisante pour implémenter le circuit. Ensuite, parmi les solutions possibles, il faut analyser l'environnement de vérification, voir si le test nécessite une connection à un environnement externe (capteurs, moniteurs, etc) ou à un ordinateur pour une co-émulation. Une fois déterminées les capacités de communication et d'interconnections nécessaires, il faut estimer le niveau de maturité du circuit afin d'estimer le nombre d'erreurs matérielles subsistant dans le circuit.

En fonction de l'étape de vérification à réaliser, de la longueur des séquences de test, on sélectionne alors une technique de vérification. On peut alors terminer le choix de la machine parmi celles remplissant tous les critères déjà mentionnés en prenant en compte l'estimation du nombre d'erreurs matérielles, la durée de mise en oeuvre des machines, la durée de recompilation après correction d'une erreur, la vitesse d'exécution et la durée de déboguage (qui dépend des capacités d'observabilité, contrôlabilité et reproductibilité). Les articles [PET00] et [PET01] proposent une équation permettant une estimation de la durée de vérification en fonction de ces différents paramètres :

$$t_{vérification} = t_{compilation} + \frac{nb_{cycles}}{f_{circuit}} \times nb_{erreurs} \times (t_{déboguage} + t_{recompilation})$$

Une partie des paramètres intervenant dans cette équation dépend de la facilité d'utilisation des machines et de l'expérience des utilisateurs, ils peuvent être interdépendants. En effet, un utilisateur sera peut être plus rapide et plus efficace en travaillant sur une solution dont l'utilisation est certes difficile, mais dont il a l'expérience, qu'avec une solution plus simple à manipuler mais qu'il ne connait pas.

Enfin, il faut s'assurer que le coût va satisfaire les contraintes budgétaires, sinon il faudra faire une nouvelle sélection, peut être moins optimale quant à la durée de vérification mais surtout moins onéreuse.

Pour conclure, la sélection d'une bonne combinaison technique/machine est très difficile car il faut envisager de nombreux paramètres interdépendants dont certain sont estimatifs (expérience des utilisateurs, facilité d'utilisation), d'autres statistiques (estimation du nombre d'erreurs matérielles, durée de mise en oeuvre) ou déterministes (taille du circuit, longueur des séquences de test).

2.6.2) <u>Besoin de coopération</u>

Sélectionner une bonne combinaison technique/machine n'est pas aisé. De plus, une mauvaise évaluation d'un ou plusieurs paramètres peut conduire au choix d'une technique et/ou d'une machine d'émulation/prototypage mal adaptée au problème. Il faudra alors changer de stratégie ce qui a un coût. Cela exige, d'une part, la disponibilité d'une autre machine et d'autre part, de perdre du temps à la mise en oeuvre d'une nouvelle plateforme de vérification. Ainsi, ce mauvais choix peut avoir des conséquences catastrophiques vu que la durée de vérification peut considérablement augmenter, ce qui engendrera un retard dans la mise sur le marché du circuit et donc un grand manque à gagner.

Au final, il serait souhaitable de s'affranchir de cette difficulté de choix de techniques et

machines. D'autre part, le prototypage apporte globalement une très grande vitesse d'exécution alors que l'émulation et la simulation apporte une excellente capacité de déboguage : ces techniques sont complémentaires, comme le montre la figure 7 qui résume les principales caractéristiques des plateformes de prototypage (MP3, MP4, ZeBu-XL, ChipIt-Platinium, FlexCube), des émulateurs (Palladium, Xserver, VStation) et des cartes de prototypage (ZeBu-ZV, ZeBu-UF, ChipIt-Silver). Si l'on combine ensemble le meilleur de chaque machine, on obtient les propriétés de la machine idéale.

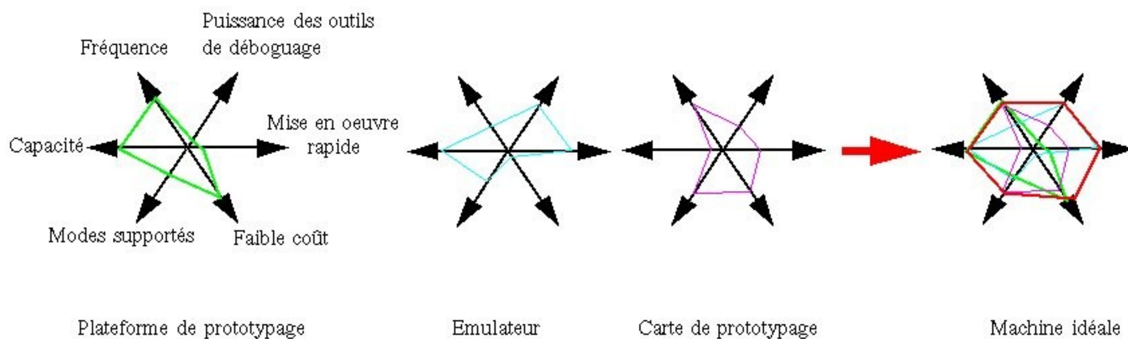


Figure 7 : Obtention de la machine idéale par coopération des machines existantes

Il y a donc un besoin de coopération entre les différentes techniques et machines de vérification afin, qu'à chaque instant, on puisse utiliser à la fois la technique et la machine la mieux adaptée au problème rencontré (figure 8).

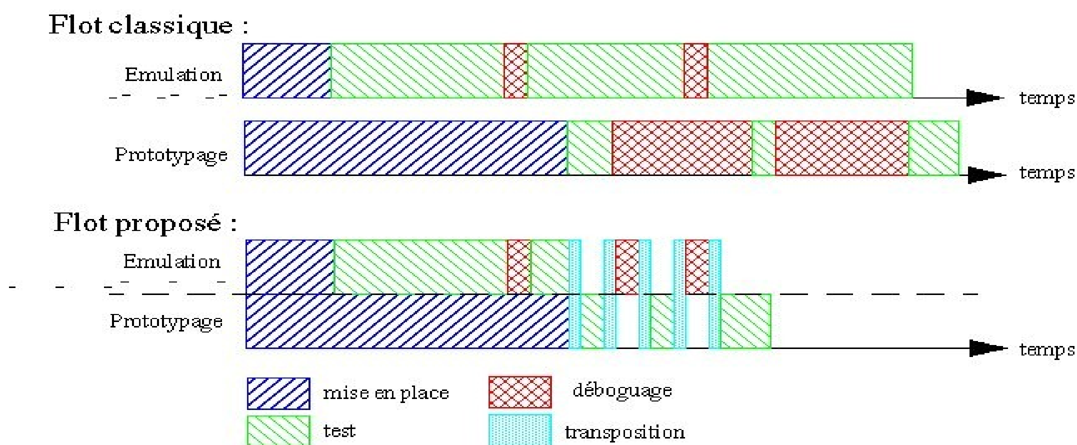


Figure 8 : Mise en évidence d'un besoin de coopération entre émulation et prototypage

Classiquement, pour une estimation du nombre d'erreurs matérielles faible (dix erreurs) dont la mise en évidence nécessite de longues séquences de test, le choix entre recourir à un émulateur ou à une plateforme de prototypage en mode STB ne sera pas aisé, les deux méthodes aboutiront sur une durée de vérification comparable. Si l'on réalise cette vérification avec un émulateur, le test sera long mais le déboguage rapide. Avec une plateforme de prototypage, ce sera l'inverse, c'est à dire que le test sera rapide mais le déboguage sera long et laborieux. De plus, si l'estimation du nombre d'erreurs est mauvaise, l'écart entre les deux méthodes se creusera, en faveur de l'émulation si ce

nombre a été sous-estimé, en faveur du prototypage s'il a été sur-estimé.

Par contre, si on pouvait faire coopérer émulateur et plateforme de prototypage, on s'affranchirait d'une part du dilemme de choix entre les deux solutions et d'autre part, on réduirait la durée de vérification. En effet, l'émulateur sera opérationnel avant la plateforme de prototypage, le test débutera donc sur émulateur. Une fois la plateforme de prototypage disponible, les tests seront exécutés sur cette machine et bénéficieront de sa grande vitesse d'exécution. Lorsqu'une erreur se produira, on arrêtera le test pour le relancer à partir de l'instant d'arrêt, sur émulateur. On bénéficiera alors des capacités de déboguage de l'émulateur.

La coopération entre techniques et machines de vérification est donc nécessaire puisque cela permet :

- de s'affranchir de la difficile sélection d'une combinaison technique/machine
- de démarrer au plus tôt la vérification
- de bénéficier du meilleur de chaque machine, à savoir la vitesse du prototypage et les capacités de déboguage de l'émulation
- de réduire la durée de vérification

Néanmoins, cette coopération a un coût puisqu'il faut mettre en place, en parallèle, deux plateformes de vérification basées sur deux machines différentes.

2.7) Coopérations existantes

La section, «2.6) Nécessité de faire coopérer les différentes techniques de vérification», a montré le besoin de faire coopérer techniques et machines de vérification afin d'améliorer l'efficacité de la vérification. Les prochains paragraphes vont dresser un état de l'art des moyens de coopération existants afin de mettre en évidence que ces moyens sont insuffisants et qu'il faut développer toute une stratégie sur ce point.

2.7.1) Interface de co-émulation SceMi

Afin de faciliter la mise en place des co-émulations transactionnelles, plusieurs compagnies, dont STMicroelectronics, réunies depuis novembre 2000 dans le comité "Accellera" ont défini une norme pour les interfaces transactionnelles. Cette norme s'appelle SceMi [SCE04] (Standard Co-Emulation Modeling Interface), la version 1.0 est sortie en mai 2003. Elle définit une infrastructure de communication transactionnelle de telle sorte que l'utilisateur ait juste à utiliser des macros matérielles et logicielles prédéfinies pour faire communiquer les deux environnements logiciel et matériel. Cette norme fournit des fonctions (ou "proxy") logicielles qui communiquent avec des ports matériels. L'un des principaux objectifs de cette norme est de définir un protocole standard de communication à haute performance indépendant de toutes les interfaces propriétaires (API) proposées par les différents fournisseurs d'émulateurs. Ainsi, grâce à cette norme, une co-émulation transactionnelle est portable d'une plateforme d'émulation/prototypage à une autre, aucune modification de code n'est nécessaire. D'autre part, cette norme permet de simplifier l'écriture des transacteurs étant donné que les communications sont entièrement gérées par les macros SceMi.

La figure 9 résume la structure d'une co-émulation transactionnelle utilisant SceMi. Pour développer un transacteur, il suffit d'utiliser les macros matérielles et d'écrire l'adaptateur matériel qui fera le lien entre les bus du circuit et les macros SceMi. SceMi en propose quatre macros matérielles :

- un port de réception de données «SceMiInPort»
- un port d'émission de données «SceMiOutPort»
- un module générateur d'horloge «SceMiClockPort»
- un module de contrôle des horloges «SceMiClockControl»

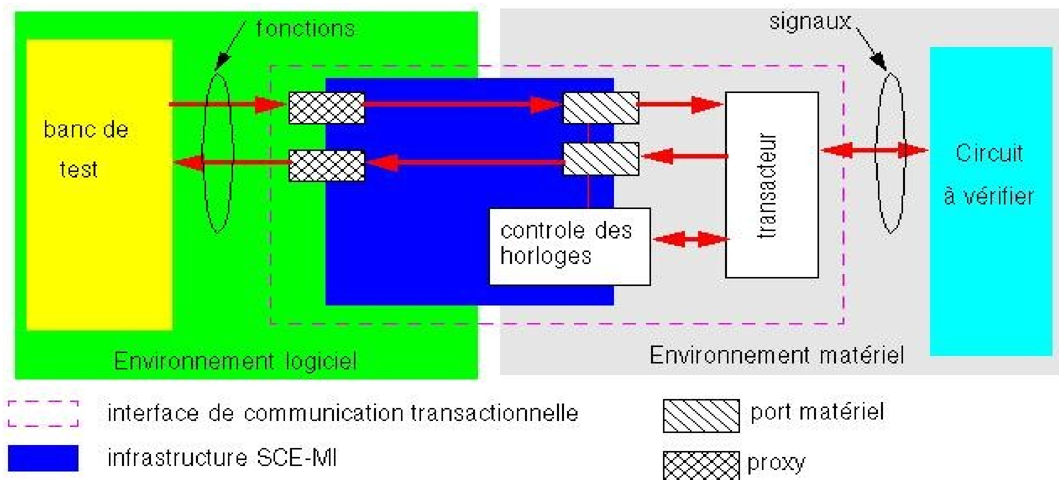


Figure 9 : Architecture d'une plateforme de co-émulation SceMi

Les modules de gestion d'horloge offrent à SceMi une grande efficacité. En effet, les horloges du circuit sont entièrement contrôlées par le transacteur. Cela permet l'utilisation d'algorithmes complexes dans la transmission de données. Par exemple, pour des besoins d'optimisation de bande passante, une transaction peut contenir des informations compressées, le transacteur devra donc décompresser ces informations avant de pouvoir réaliser la transaction. Si l'algorithme de compression est complexe, que le message nécessite plusieurs cycles d'horloge avant d'être décodé et que, le circuit en vérification nécessite immédiatement les données associées à cette transaction, le transacteur peut alors suspendre les horloges du circuit le temps de finir le décodage. Ce système de contrôle d'horloges est aussi très pratique dès que le circuit émulé nécessite une réponse immédiate de l'environnement logiciel mais que celui-ci n'est pas prêt à fournir les éléments attendus. Dans ce cas, les horloges du circuit sont suspendues.

2.7.2) Flot de conception TLM

Une méthodologie novatrice de conception de circuits intégrés, notamment développée par STMicroelectronics, permet une coopération des techniques de simulation, co-simulation et co-émulation.

Cette méthodologie dite «TLM» (Transactionnal Level Modeling) cherche à modéliser à un haut niveau d'abstraction les circuits en cours de développement [SCH03]. Cette modélisation est réalisée en SystemC, un système de classes C++ adapté à la modélisation des systèmes sur puce. Cette technique permet de créer un modèle de simulation des différents modules qui composeront le système sur puce. La communication entre les différents modèles est modélisée à un niveau transactionnel. Cette méthodologie TLM permet de modéliser le comportement de chaque composant du système et permet ainsi de vérifier au plus tôt si le système va être capable de tenir ses spécifications, si les algorithmes utilisés offriront les performances attendues. De plus, la stratégie TLM permet l'utilisation de tests efficaces et simples à écrire grâce au haut niveau d'abstraction utilisé. Cette technique est particulièrement efficace pour l'exploration d'architecture, quelques semaines de travail étant suffisantes pour concevoir un modèle fonctionnel du circuit. Le haut niveau d'abstraction permet l'obtention de fréquences de simulation importante, de l'ordre de la centaine de kilohertz. Une fois validée l'architecture du circuit, les plateformes TLM servent au développement des logiciels embarqués et des drivers.

D'autre part, l'écriture d'un banc de test efficace est une tâche souvent difficile, longue et fastidieuse. Réutiliser des bancs de test est essentiel pour accélérer la vérification, donc la

conception d'un circuit. Avec TLM, un banc de test global au circuit est développé avant même la conception des composants matériels et logiciels. Ainsi, au lieu de développer un banc de test pour chaque composant, on utilisera, pour la vérification de chaque bloc, le banc de test au niveau système.

Lorsque la description au niveau RTL d'un composant matériel est prête, on peut la vérifier en remplacant, dans le système TLM, le modèle TLM du composant par son modèle RTL. Ainsi, on passe d'une simulation de haut niveau à une co-simulation utilisant d'un coté des modèles TLM et de l'autre des modèles RTL simulés sur un simulateur HDL. Cette technique est très avantageuse puisqu'elle utilise le même jeu de tests pour toute la vérification. De plus, outre la vérification d'un composant en particulier, on réalise en permance une vérification fonctionnelle du système complet.

Lorsqu'une grande partie des composants matériels du circuit sont développés, les techniques de co-émulation peuvent accélérer la vérification de ces composants dans un environnement TLM. Au lieu d'utiliser un simulateur HDL, on peut utiliser un émulateur ou une plateforme de prototypage pour faire fonctionner la partie matérielle.

Deux stratégies sont envisageables, elles sont notamment présentées dans un livre écrit par STMicroelectronics [GHE05].

2.7.2.1) TLM et Co-émulation à synchronisation "cycle à cycle"

Le matériel travaille au niveau signal, le TLM au niveau transactionnel. Le passage signal/transaction nécessite un composant spécifique. Si cette adaptation est faite au niveau logiciel, on parle de BFM (Bus Fonctional Model). La figure 10, tirée de [GHE05], illustre ce concept qui a été utilisé pour la vérification d'un circuit de décodage vidéo MPEG2 nommé LCMPEG.

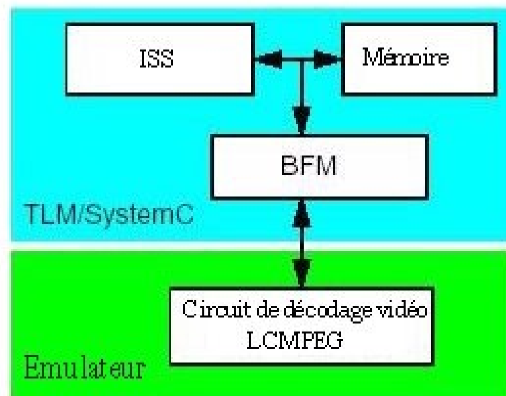


Figure 10 : Plateforme TLM avec co-émulation à synchronisation cycle à cycle

En utilisant de tels convertisseurs, on réalise une co-émulation à synchronisation cycle à cycle. D'après [GHE05], écrire ce convertisseur logiciel est assez rapide et la co-émulation ainsi réalisée permet d'accélérer d'environ trente fois la plateforme de vérification par rapport à une plateforme de simulation pure.

2.7.2.2) TLM et Co-émulation transactionnelle

L'autre possibilité d'accélération de la plateforme consiste à utiliser un convertisseur transaction/signal matériel appelé transacteur. Un tel convertisseur est plus compliqué à réaliser qu'un convertisseur logiciel (BFM) mais, permet d'améliorer grandement les performances de la plateforme de vérification (figure 11). En effet, dans ce cas, on réalise une co-émulation transactionnelle. Au niveau des performances, [GHE05] montre que l'utilisation de cette technique peut accélérer d'un facteur trois cents la plateforme de vérification.

D'autre part, en plus d'augmenter les performances en vitesse, l'utilisation d'une plateforme TLM avec co-émulation transactionnelle à l'avantage d'être portable. En effet, contrairement à la plateforme de co-émulation à synchronisation cycle à cycle qui demande l'écriture d'un BFM spécifique à l'API de chaque machine de simulation/émulation/prototypage visée, la plateforme TLM avec co-émulation transactionnelle utilise le lien de communication SceMi. Ainsi, théoriquement, sans aucune modification de code, on peut porter la plateforme de vérification sur toute machine supportant la norme SceMi. Cela permet de faire facilement coopérer une plateforme de prototypage avec un simulateur ou un émulateur. En effet, si la vérification d'un circuit nécessite le passage d'une centaine de tests indépendants les uns des autres, une plateforme de prototypage sera utilisée pour sa grande vitesse d'exécution et son coût modéré. Lorsqu'un test détectera un problème, on pourra alors le rejouer en simulation ou en émulation afin de facilement et rapidement déboguer le matériel.

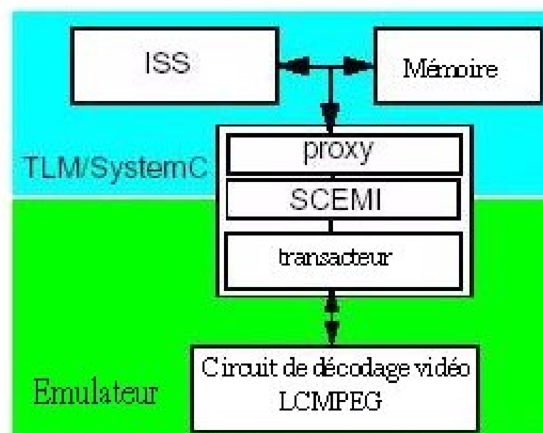


Figure 11 : Plateforme TLM avec co-émulation transactionnelle

2.7.2.3) <u>Limitations : reproductibilité et longueur des séquences de tests</u>

La stratégie TLM est très intéressante puisque le même banc de tests est utilisé du début à la fin de la conception d'un circuit. De plus, le développement tant des logiciels embarqués que des drivers du circuit peut commencer très tôt dans le flot de conception, il n'est pas nécessaire d'attendre que le matériel soit disponible.

Cependant, cette stratégie trouve ses limites lorsqu'il faut appliquer de longues séquences de test et que la majorité du cicruit est décrit au niveau RTL. Même avec une co-émulation transactionnelle, la fréquence d'une plateforme TLM intégrant des composants RTL ne dépassera pas 50kHz. Ainsi, si des tests de non régression nécessite des milliards de cycles d'horloge, leur exécution nécessitera plusieurs jours, voire plusieurs semaines, ce qui peut ne pas être acceptable.

D'autre part, travailler au niveau transactionnel permet d'augmenter la vitesse d'exécution mais fait perdre de la précision à la plateforme de vérification ce qui est le plus grand inconvénient de cet environnement. Une co-émulation transactionnelle est reproductible au niveau de la transaction mais pas forcément au niveau cycle. Cela signifie qu'en cas de reproduction d'un test, l'ensemble des transactions se répèteront dans le même ordre mais, du point de vue RTL, elles ne se dérouleront pas forcément toutes au même cycle d'horloge par rapport au premier test. Ainsi, si le circuit comporte une erreur liée à des temps de propagation, cette erreur peut s'avérer difficilement reproductible. De plus, en co-émulation transactionnelle, le déboguage est intrusif. Une partie de la bande passante est utilisée pour extraire les données observées et la charge de l'ordinateur associé augmente afin de les traiter. Globalement la plateforme ralentit, ce qui peut faire fonctionner un

circuit qui ne marchait pas au préalable.

Enfin, il est à noter que la méthode de conception TLM permet certes une coopération des techniques de simulation, co-simulation et co-émulation, c'est à dire qu'il est facile d'exécuter le même test avec chacun de ces environnements. Néanmoins, il n'est pas possible, au cours d'un test donné, de changer d'environnement. Ainsi, par exemple, on ne peut pas débuter un test à 10MHz en simulation TLM pure et passer en co-émulation transactionnelle au bout d'une heure pour des besoins de précision et de déboguage. La coopération est donc limitée.

2.7.3) Solutions internes aux différents fabricants

Plusieurs fournisseurs de machines ont vu la complémentarité qui existe entre émulateur/plateforme de prototypage et simulateur et ont développé des solutions en ce sens.

Tout d'abord l'émulateur Xserver exécute le même code que le simulateur Xsim. L'utilisation du même modèle d'exécution sur émulateur et simulateur permet de commencer facilement un test dans un environnement, d'arrêter ce test et de le reprendre dans l'autre environnement. Ceci est très pratique pour des phases de déboguage. On utilise la haute vitesse d'exécution de l'émulateur pour atteindre un point critique puis, à partir de ce point, on utilise le simulateur pour déboguer. Le déboguage nécessite souvent de longues heures d'investigation sur quelques centaines ou milliers de cycles d'horloge. Avec cette technique, on ne bloque pas l'émulateur pendant la longue étape de déboguage, d'autres tests peuvent être joués. De plus, ce changement d'environnement est possible avec des émulations en mode STB ainsi qu'avec les co-émulations. Le principal inconvénient de cette solution est la dépendance au modèle d'exécution, cette solution ne convient pas à un autre simulateur HDL que Xsim et un autre émulateur que Xserver.

L'émulateur Palladium présente des capacités analogues, à savoir qu'il est possible d'arrêter une émulation en mode STB et de la continuer avec un simulateur de l'architecture Palladium. La coopération est moins forte que dans le cas Xserver (coopération seulement en mode STB) et est encore spécifique à la machine, donc non générique. De plus, les performances sur simulateur d'architecture Palladium sont plus faibles que celles obtenues sur un simulateur HDL tel que NcSim.

Au final, le principal problème de ces coopérations est la dépendance au modèle d'exécution. La coopération existante est liée aux modèles d'exécution utilisés par les logiciels et machines. Si l'on veut augmenter la coopération, il faut donc lever cette dépendance au modèle d'exécution ou rendre cohérents tous les modèles utilisés par l'ensemble des logiciels et machines.

2.7.4) Solutions maisons spécifiques à un circuit

Les coopérations proposées dans l'industrie sont faibles. Ainsi, il peut être envisagé d'obtenir une coopération de différentes techniques pour un projet spécifique. Les prochains paragraphes vont montrer les difficultés de mise en oeuvre de cette méthode ainsi que son faible intérêt et sa faible rentabilité.

2.7.4.1) Problème de cohérence des modèles

La première étape de mise en oeuvre d'une plateforme d'émulation/prototypage matériel consiste à réaliser une synthèse des fichiers HDL décrivant le circuit. Le résultat est une description du circuit au niveau portes logiques, les primitives utilisées étant spécifiques à la technologie de la plateforme de vérification.

Cette synthèse peut être réalisée à l'aide de plusieurs outils (exemples d'outils de synthèse FPGA : Synplify Pro, DC FGPA, Precision RTL) qui peuvent fournir des résultats assez différents.

D'une part, en fonction de la plateforme cible, les primitives utilisées pourront être différentes et d'autre part, à primitives équivalentes, les outils peuvent ne pas réaliser les mêmes optimisations, simplifications, ce qui engendre un problème de cohérence des modèles d'exécution.

Ces incohérences peuvent avoir cinq origines : simplification de registres, duplication de registres, ajout de ressources de multiplexage, hétérogénéité des primitives matérielles et non préservation des noms et hiérarchies.

Durant la synthèse, les outils cherchent à minimiser les ressources matérielles utilisées, notamment en supprimant les ressources inutiles. Par exemple, si un registre reçoit en entrée un signal constant, la sortie de ce registre sera aussi constant, ce registre peut donc être supprimé. En fonction de l'efficacité des outils, certaines simplifications seront vues par l'un et non par l'autre. Cela peut donc engendrer un problème de cohérence des ressources mémoires.

D'autre part, les outils de synthèse cherchent également à optimiser la fréquence de fonctionnement du circuit. Pour répondre à ce besoin, il peut arriver que des registres soient dupliqués. En effet, imaginons que la sortie d'un registre soit reliée à l'entrée des deux autres registres dont l'un d'eux est physiquement éloigné. Le temps de propagation entre ces registres serait assez important. Si le registre source est dupliqué et que le doublon est rapproché du registre de destination le plus lointain, le temps de propagation sera ainsi réduit (figure 12). En fonction des contraintes de fréquence imposées aux outils de synthèse et des choix de placement pris par ces mêmes outils, là encore on peut obtenir deux résultats de synthèse qui diffèrent quant au nombre de registres utilisés.

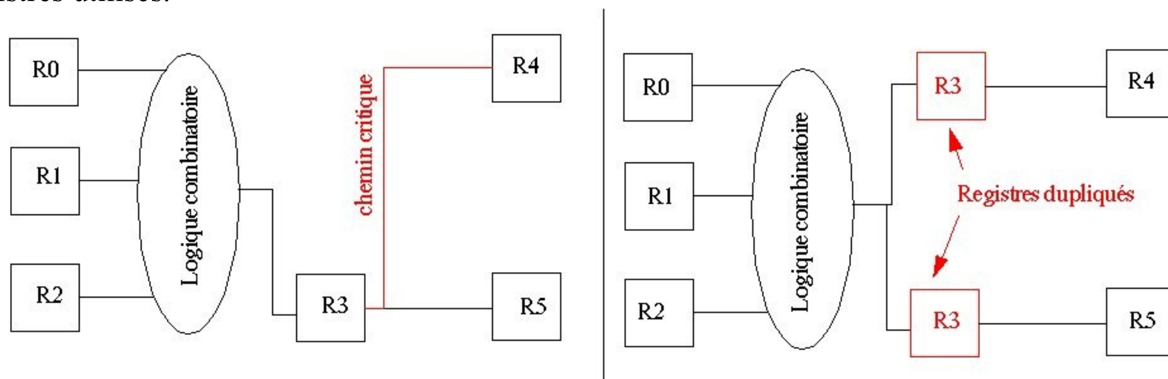


Figure 12 : Duplication de registre

Un autre problème de cohérence potentiel est lié à l'architecture de la plateforme de vérification. Les plateformes de prototypage reposent sur l'utilisation de FPGAs. Les outils de mise en oeuvre doivent assurer la communication entre ces différents FPGAs et, lorsque le nombre de signaux échangés entre FPGAs dépasse le nombre de ports des composants, il faut recourir à des techniques de multiplexage/démultiplexage, ce qui va ajouter des ressources au circuit. Par contre, les émulateurs utilisent des architectures spécifiques et ne rencontrent pas ce problème de multiplexage. Ainsi, le multiplexage peut engendrer un problème de cohérence entre deux plateformes de prototypage ou une plateforme de prototypage et un émulateur.

Dans le même esprit, les plateformes n'ont pas forcément les mêmes primitives. Ainsi, par exemple, si la sortie d'un registre doit être inversée, un synthétiseur peut, si les primitives le permettent, directement utiliser la sortie inversée d'un registre (figure 13). Dans le cas de la solution 2, le signal "Q" sera supprimé au profit de son signal complémentaire. L'état du circuit est défini par la valeur des signaux de sortie des registres dans cet exemple. Ces registres n'ayant pas rigoureuseument les mêmes ports de sorties, ces deux solutions n'ont pas rigoureusement les mêmes variables d'états. Ce problème de non homogénéité des primitives est très courant pour les

mémoires et les entrées/sorties.

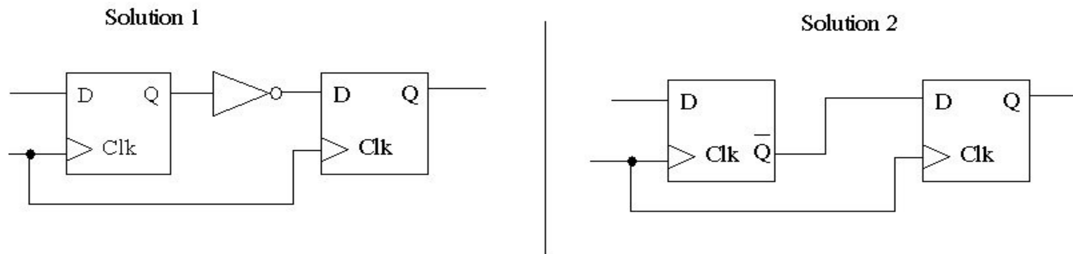


Figure 13 : Disparité des primitives d'émulation/prototypage matériel

Enfin, les outils de synthèse renomment souvent des signaux et deux «netlists» peuvent être équivalentes au niveau des ressources allouées mais différer en terme des noms et de hiérarchies, ce qui ne simplifie pas le problème de cohérence des modèles.

Pour conclure, l'étape de synthèse, en lien avec la topologie et la technologie de la plateforme d'émulation/prototypage visée impacte fortement la cohérence des modèles exécutés sur deux machines différentes.

2.7.4.2) Disparité des possibilités d'accès à l'état d'un circuit

En supposant que, pour un circuit donné, on ait réussi à rendre cohérents les différents modèles utilisés par les machines et logiciels visés, encore faut il être capable d'accéder à l'état d'un modèle, à l'état du circuit. La notion d'état d'un circuit sera précisément présentée dans le chapitre III, section «3.2) Notion d'état d'un circuit». Ici, nous ne nous intéressons pas à l'état à proprement parlé, mais au moyen d'extraire et d'initialiser un état, ce qui signifie, pour une machine d'émulation/prototypage ou un simulateur HDL, observer et forcer certains signaux clés.

Tout d'abord, on peut envisager d'utiliser les capacités de déboguage. A ce niveau, les simulateurs HDL permettent l'observation et le forçage de l'ensemble des signaux du circuit. Les machines d'émulation présentent des capacités de déboguage comparables. Cependant, il faut distinguer le mode ICE des autres modes de fonctionnement. En effet, en mode ICE l'émulateur fonctionne avec un environnement extérieur et, la fréquence du circuit émulé ne peut être diminuée, même pour des besoins de déboguage, sous peine de perdre la synchronisation entre l'émulateur et son environnement. En mode ICE, l'observabilité est donc réduite. Les signaux observés sont transmis dans une mémoire tampon. Lorsque cette mémoire est pleine, soit l'acquisition s'arrête, soit les nouvelles données effacent les précédentes. Ce mécanisme procure juste une courte fenêtre temporelle d'observation sur quelques signaux. Si la mémoire est trop petite, il ne sera pas possible d'observer tous les signaux nécessaires à une capture d'état. Enfin, dans les autres modes d'émulation, il est à noter que plus on observe des signaux et plus l'émulateur est ralenti. Observer tous les signaux caractérisant l'état d'un circuit risque donc d'annuler les bénéfices en vitesse de l'émulateur.

D'autre part, concernant les émulateurs, on peut signaler que le Palladium est capable, uniquement en mode STB, de sauvegarder l'état du système et de restaurer cet état. La sauvegarde d'état se fait machine arrêtée. Par contre, le format de sauvegarde est spécifique à la machine, le constructeur ne fournit aucune information à ce sujet, ces fichiers de sauvegarde ne sont donc pas exploitables.

De leur coté, les plateformes de prototypages ont des capacités de déboguage beaucoup plus pauvres. Les meilleurs solutions offrent une visibilité sur l'ensemble des registres plus quelques signaux combinatoires à déclarer avant le flot de synthèse, placement et routage. Au mieux, seuls

les registres sont forçables sur les meilleures solutions. De plus, on distingue l'observation statique et l'observation dynamique. Dans le premier cas (statique), les signaux observés sont sélectionnés avant l'étape de placement et routage, ces signaux sont alors routés vers une mémoire tampon. Ceci met en oeuvre le même mécanisme que celui utilisé dans une émulation en mode ICE. Le nombre de signaux caractérisant l'état d'un circuit est trop important pour que cette technique puisse être utilisée. En mode dynamique, l'utilisateur sélectionne, pendant le test, les signaux qu'il veut voir. Cette technique utilise la possibilité de «readback» des FPGAs, la fréquence de prototypage tombe alors à quelques kilohertz pendant l'observation.

D'autre part, les machines ZeBu proposent également, en mode STB, de capturer l'état du système pour le restaurer ultérieurement. Là encore, la capture se fait machine arrêtée. Ces machines exploitent également les possibilités de «readback» offertes par la dernière génération de FPGAs Xilinx. Cela permet, sans arrêter la machine, de capturer l'état de tous les registres à un instant donné, cette capture étant sauvegardée dans les mémoires internes des FPGAs [XIL06]. De nombreux cycles d'horloge sont nécessaires au transfert du contenu de ces mémoires vers un ordinateur. Cela interdit d'enchaîner les captures sur plusieurs cycles consécutifs sans arrêter la machine.

Enfin, on peut mentionner l'outil Siloti [NOV06] qui permet une visibilité complete (à l'exception des mémoires RAMs) de l'ensemble des signaux du circuit. Les techniques de DFT (Design For Test) génèrent une chaîne de scan qui relie tous les registres d'un circuit. Cette étape de DFT est obligatoire dans la conception des circuits, elle sert à vérifier les circuits fabriqués. Siloti utilise cette structure DFT comme base de sa méthode DFD (Design For Debug). L'outil insère des contrôleurs de chaîne de scan qui vont permettre, aux instants désirés, d'arrêter le circuit et de capturer l'état des registres via les chaînes de scan. L'outil est alors capable de recalculer l'ensemble des signaux combinatoires. Cette méthode est intéressante. Cependant il faut, d'une part, que le circuit prototypé ait déjà subit l'étape de DFT et d'autre part que la solution de prototypage soit compatible avec l'outil (actuellement seules les solutions Eve et ProDesign le sont). Enfin, cette méthode oblige l'arrêt du test pendant la capture et n'augmente pas les capacités de forçage des signaux.

Pour conclure, il y a une grande disparité des capacités d'observation et d'initialisation des signaux d'un circuit simulé, émulé ou prototypé. Cela rend quasi impossible la coopération des machines. De plus, quand bien même on décide de faire coopérer des machines accédant de façon satisfaisante aux signaux, la capture réduira obligatoirement la vitesse d'exécution du système. Ainsi, il est impossible, avec les moyens actuels, de capturer un état en mode ICE sauf, si l'on accepte d'arrêter la machine, de perdre les synchronisations et de ne pas aller plus loin dans le test en cours.

2.7.4.3) <u>Conclusion : charge de travail élevée et faible réutilisabilité</u>

Les deux sections précédentes ont montré que d'une part il est difficile de rendre cohérents les modèles exécutés sur deux environnements de vérification différents et que d'autre part, il est aussi extrêmement ardu de pouvoir accéder facilement, en lecture et écriture, à l'ensemble des variables des modèles utilisés. En particulier, avec les moyens actuels, il est impossible de sauvegarder un état de circuit avec une machine fonctionnant en mode ICE, or c'est ce mode qui nécessite le plus de coopération (ce point sera justifié dans la section «3.4.1.1) Préservation de la vitesse : capture d'état dynamique»).

En conclusion, faire coopérer plusieurs techniques reposant sur des logiciels et machines non prévus pour cela demandera une charge de travail élevée et donc un énorme investissement. De plus, la cohérence obtenue sera dépendante des propriétés du circuit et des machines utilisées. Si l'on change le circuit ou une machine, tout est à refaire. Le travail sera coûteux et faiblement réutilisable, ce n'est donc pas rentable pour une entreprise.

2.7.5) Recherches dans le domaine

L'utilisation de machines industrielles d'émulation et de prototypage matériel engendre de gros budgets de fonctionnement que peu d'entreprises peuvent se permettre. Pour un laboratoire de recherche, le coût d'un parc de machine n'est pas envisageable. Les laboratoires ont, au mieux, une seule solution industrielle.

Cette contrainte budgétaire fait qu'il n'y a quasiment aucune recherche dans le domaine. La plupart des laboratoires de recherche qui ont des activités de prototypage cherche essentiellement à developper leur propre solution, moins performante que les solutions industrielles mais surtout moins chères.

Néanmoins, un travail sur le sujet a été réalisé à l'université de Californie [KIR99], [KIR00]. Les auteurs de ce travail (Kirovski, Potkonjak et Guerra) mettent en évidence la complémentarité existant entre la capacité de déboguage des simulateurs HDL et la grande vitesse d'exécution des émulateurs. Ils proposent un algorithme complexe permettant de trouver les variables minimales dont la connaissance suffit pour une sauvegarde d'état. La méthode repose sur une modélisation des circuits à l'aide d'un graphe SDF (Synchronous Data Flow). Le circuit est ensuite instrumenté, chaque variable identifiée est ramenée sur un port du circuit. Une capture d'état est réalisée sur plusieurs cycles consécutifs, le système prenant en compte l'existence de dépendances entre les données dans une fenêtre temporelle. Le résultat est intéressant puisque très peu de matériel est ajouté pour accéder à un état. Néanmoins, l'approche est très complexe et demande une durée de calcul acceptable sur des petits circuits de quelques centaines de portes mais bien trop longue dans le cas de circuits de plusieurs millions de portes. De plus, l'approche est adaptée aux circuits de type traitement de données sans contrôle complexe (DSP), mais elle n'est pas généralisable à l'architecture complexe des circuits actuels. Enfin, quand bien même cette technique n'augmente quasiment pas le nombre de ports des circuits (deux ports ajoutés dans [KIR99] pour un circuit de mille portes logiques), on arriverait alors à plus de deux mille ports ajoutés sur un circuit d'un million de portes si le rapport entre la taille d'un circuit et le nombre de ports ajoutés est sensiblement constant. Ceci n'est pas du tout acceptable, l'approche n'est donc valable que dans le cas de petits circuits très simples.

2.8) Nécessité d'une solution générique applicable à toutes les machines dans tous les modes de fonctionnement

Si l'on regarde l'ensemble des approches existantes pour faire coopérer des techniques de vérification du RTL, on s'aperçoit qu'aucune solution n'est commune à l'ensemble des modes de fonctionnement (le mode ICE n'est jamais supporté). Les coopérations existantes sont limitées à quelques machines et logiciels conçus pour coopérer mais généralement les machines n'ont pas été élaborées dans ce but.

Pour améliorer cette coopération, il faut développer une solution générique, qui ne dépende pas des propriétés des machines ni du modèle d'exécution, de manière à pouvoir facilement changer de plateforme au cours de l'exécution d'un test. De plus, cette solution générique doit pouvoir fonctionner quelle que soit la technique de vérification RTL utilisée.

L'approche de Kirovski [KIR99] et du logiciel Siloti [NOV06] instrumentant le circuit afin d'utiliser une stratégie DFD (Design For Debug) est intéressante car elle permet de s'affranchir du problème de cohérence des modèles d'exécution ainsi que des problèmes d'accès aux variables d'état. Cette idée servira de point départ pour l'obtention d'une solution générique.

Enfin, toutes les stratégies existantes de vérification orientées coopération font l'hypothèse que l'utilisateur sait à quel moment apparaît une erreur et qu'il est alors capable d'arrêter la machine et de transfèrer le test sur une autre machine plus adaptée au déboguage. Cette hypothèse est fausse. En général, les ingénieurs savent qu'un test a échoué mais, il faut souvent plusieurs heures d'analyse

avant d'identifier précisément l'instant et le lieu d'apparition d'un bogue. Cela implique une capacité d'échantillonnage de l'état du circuit. Comme tous les modes doivent être supportés, le mode ICE impose que cet échantillonnage se fasse sans arrêter ni même ralentir le test. Il doit donc être réalisé dynamiquement, en temps réel.

2.9) Gestion d'un parc de machines

Afin de tenir les défis de délais de conception, les grands groupes industriels de microélectronique sont obligés de recourir à un parc de machines d'émulation et prototypage, malgré leurs prix très élevés. La gestion de ce parc est aussi un défi. Vu les coûts engendrés, il faut trouver les bons compromis sur le nombre et la variété des machines et, tant que possible, optimiser leur utilisation. Cette optimisation n'est pas simple vu le grand nombre de paramètres intervenant dans les choix, comme nous l'avons vu dans la section «2.6.1) Sélection du meilleur compromis techniques/machines».

2.9.1) Durée de mise en oeuvre

La durée de mise en oeuvre des machines est probablement le facteur le plus pénalisant dans la gestion d'un parc de machines. Les précédents paragraphes ont montré que cette durée dépend de la machine utilisée ainsi que du mode de fonctionnement choisi. Globalement, il faut compter quelques semaines à un mois de travail pour mettre en oeuvre un émulateur et plusieurs mois (jusqu'à six mois) pour mettre en oeuvre une plateforme de protrotypage [BIG04]. Ainsi, il peut arriver que le temps de mise en oeuvre d'une plateforme soit plus important que la durée des tests effectués sur cette plateforme [PET00], [PET01], ce qui conduit à un mauvais rendement d'utilisation.

2.9.2) Utilisation non optimale des machines

La section «2.6.1) Sélection du meilleur compromis techniques/machines» a traité de l'importance de l'estimation des paramètres de sélection d'une technique et d'une machine de vérification. Notamment, il a été montré qu'une mauvaise estimation des paramètres peut conduire à une importante augmentation de la durée de vérification. Dans ce cas, on peut changer de stratégie et utiliser une technique plus adaptée. Cependant, tout changement de stratégie a un coût. Ceci exige, d'une part, la disponibilité d'une autre machine et d'autre part, de perdre du temps à la mise en oeuvre d'une nouvelle plateforme de vérification.

La disponibilité d'une machine adaptée traduit, soit que l'on a de la chance, que le projet tombe dans une période creuse ou, que l'on a volontairement surdimensionné le parc de machines de manière à garantir la disponibilité d'une machine de vérification. De plus, le changement de plateforme fait perdre du temps. Le taux d'utilisation du parc en est donc affecté et n'est pas optimal.

Si aucune autre machine de vérification n'est disponible, il faudra persister dans l'utilisation d'une technique non adaptée. Là encore, on aura une utilisation non optimale des machines.

Enfin, devant le prix élevé des machines, en particulier des émulateurs, les entreprises ne pourront pas se permettre d'en acquérir un grand nombre. Les émulateurs, machines les plus polyvalentes, risquent d'être souvent saturées. Cela oblige certains projets à utiliser d'autres solutions moins adaptées aux besoins.

Pour conclure, quelle que soit la stratégie d'investissement et d'utilisation d'un parc de machines, l'utilisation est non optimale. D'une part, on ne fait pas toujours fonctionner ces dernières dans leurs modes de prédilection, d'efficacité maximale et d'autre part, il faut pouvoir gérer les pics de besoin de ressources, en particulier la journée.

2.9.3) Coût financier et temporel de la vérification

Si l'on regarde l'aspect efficacité de la vérification, le surdimensionnement du parc de machine est la meilleure solution. Les émulateurs sont les machines les plus polyvalentes donc les machines à favoriser dans un parc. Cependant, économiquement parlant, cette stratégie est très onéreuse vu le grand nombre de machines à acquérir, particulièrement le grand nombre d'émulateurs. De plus, la charge d'utilisation des machines n'est pas constante sur une année. Avec une telle stratégie, l'ensemble du parc peut n'être saturé que quelques mois par an ce qui conduit à une utilisation non optimale des machines et un coût de vérification très élevé.

Au contraire, limiter le nombre de machine est plus économique mais va imposer des délais, des attentes avant qu'un projet puisse utiliser les ressources. De plus, cette stratégie impose l'utilisation de techniques sur des machines non adaptées, ce qui rallonge la durée de vérification.

Pour conclure, les personnes en charge d'un parc de machines doivent trouver le bon compromis entre le coût financier et le coût temporel de la vérification. Actuellement, il est impossible d'obtenir une vérification rapide et efficace avec juste la machine la moins chère du marché.

2.10) Conclusion : nécessité d'un flot automatisé de vérification orienté interopérabilité

Etant donné qu'il est actuellement impossible de réaliser une vérification rapide et efficace pour un coût modéré, il est essentiel de trouver de nouvelles méthodes afin d'améliorer ce constat. Il faut à la fois chercher à réduire la durée et le coût de la vérification.

La section «2.9) Gestion d'un parc de machines» a mis en évidence une gestion non optimale des parcs de machines d'émulation/prototypage. Ces machines sont utilisées dans des modes pour lesquels elles sont mal, voire non, adaptées. La partie «2.6) Nécessité de faire coopérer les différentes techniques de vérification» a montré qu'en faisant coopérer des techniques et des machines, on peut améliorer l'efficacité de la vérification et donc réduire à la fois durée et coût de la vérification.

Les machines et techniques de vérification actuelles ont été pensées de manière à répondre à un besoin de vérification (vérification au niveau composant, vérification de l'intégration, au niveau système, développement de logiciels) mais n'ont pas été conçues pour collaborer entre elles alors qu'elles présentent toutes des propriétés complémentaires. Pour compenser cette limitation, il faut donc développer une nouvelle stratégie de vérification basée sur la coopération des techniques et machines de vérification. Le prochain chapitre de cette thèse va développer cette idée et proposer une nouvelle stratégie de vérification. La notion de coopération des techniques et machines de vérification sera précisée et dénommée par le terme «interopérabilité».

De plus les sections «2.7) Coopérations existantes» et «2.8) Nécessité d'une solution générique applicable à toutes les machines dans tous les modes de fonctionnement» ont montré d'une part, l'absence d'une solution générique permettant l'obtention de plateformes interopérables et d'autre part, la complexité pour la mise au point d'une telle solution. Cette mise au point nécessite l'utilisation du concept DFD («Design For Debug») et impose d'instrumenter les circuits à valider. Etant donnée la taille des circuits (plusieurs dizaines de millions de portes), cette instrumentation ne peut pas être réalisée manuellement, il faudra automatiser cette étape.

Dans le prochain chapitre, en plus de proposer une nouvelle stratégie de vérification orientée interopérabilité, cette thèse proposera une stratégie d'instrumentation des circuits ainsi qu'un outil automatisant ce processus.

Chapitre III - Concept d'interopérabilité en émulation matérielle et flot de vérification associé

3.1) Introduction....59
3.2) Notion d'état d'un circuit....59
3.3) Extraction des variables d'état : sélection du modèle de référence....60
3.3.1) Hétérogénéité du niveau d'abstraction des modèles des composants du circuit....60
3.3.2) Problèmes de cohérence d'état liés à disparité des résultats de synthèse....60
3.3.3) Conclusion : la caractérisation de l'état d'un circuit ne peut se faire qu'après synthèse...61
3.4) Accès à l'état d'un circuit....62
3.4.1) Cahier des charges....62
3.4.1.1) Préservation de la vitesse : capture d'état dynamique....62
3.4.1.2) Gestion de la bande passante : non altération des co-émulations....63
3.4.1.3) Gestion des horloges contrôlées....63
3.4.1.4) Limitation du temps d'accès et contrôlabilité du pas d'échantillonnage....63
3.4.1.5) Gestion du banc de test et des dépendances externes....64
3.4.2) Proposition d'une solution d'accès aux ressources mémoire....65
3.4.2.1) Duplication des ressources mémoire....65
3.4.2.2) Gestion des horloges....65
3.4.2.2.1) Norme SceMi : exploitation des concepts uclock et cclock....66
3.4.2.2.2) Norme SceMi : présentation et extension du concept «cycle stamp»....68
3.4.2.3) Extraction d'état par chaîne de scan....69
3.4.2.4) Restauration d'état....70
3.4.2.5) Cellules spécifiques à l'interopérabilité....71
3.4.3) Fichier de sauvegarde d'état....71
3.4.3.1) Structure d'arbre....72
3.4.3.2) Fichiers de description et fichiers de sauvegarde....73
3.4.4) Transacteur lié à l'échange d'état....73
3.4.5) Gestion des signaux d'entrée/sortie....75
3.4.5.1) Problèmes de bande passante et de volume de stockage....75
3.4.5.2) Compression temps réel....76
3.4.5.2.1) Limitation en surface....76
3.4.5.2.2) Segmentation du bus en ensembles....76
3.4.5.2.3) Codage de Golomb....78
3.4.5.2.4) Gestion des débordements....80
3.4.5.2.5) Fichier de sauvegarde de l'évolution des signaux d'entrée/sortie....80
3.4.5.2.6) Implémentation et performances....80
3.4.6) Format de sauvegarde d'état....82
3.5) Concept d'interopérabilité....82
3.6) Proposition d'un flot de prototypage semi-automatisé orienté interopérabilité....82
3.6.1) Flot d'obtention de la netlist de référence....83
3.6.2) Flot de prototypage....84
3.6.3) Outil d'interopérabilité : modification automatique d'une "netlist gate"....85
3.7) Flot de vérification orienté interopérabilité....87

3.7.1) Architecture d'un système sur puce......87
3.7.2) Phase de vérification et phase de déboguage......88
3.7.3) Optimisation : travail sur un sous-ensemble......90
3.7.3.1) Optimisation des ressources, de la vitesse et du coût du déboguage......91
3.7.3.2) Obtention de l'environnement du sous-ensemble défectueux......93
3.7.4) Apports de la technique proposée......93
3.7.4.1) Apports vis à vis du circuit en validation......93
3.7.4.2) Apports vis à vis de la gestion d'un parc de machines......94
3.8) Conclusion......96

3.1) Introduction

Les précédents chapitres ont mis en évidence la nécessité de recourir à différentes techniques d'émulation et prototypage matériel afin d'accélérer la vérification des systèmes sur puce. L'état de l'art a également fait ressortir un besoin de coopération entre les différentes techniques d'émulation matérielle, coopération qui à ce jour est quasi inexistante.

Ce chapitre va dans un premier temps répondre à ce manque de coopération en formalisant la notion d'état d'un circuit. Cette notion permettra de dégager un nouveau concept : l'interopérabilité en émulation et prototypage matériel. Ce chapitre va également montrer que l'exploitation de ce nouveau concept d'interopérabilité en émulation matérielle nécessite l'accès à de nombreux éléments du circuit. Ce nombre est tellement important qu'il est obligatoire d'automatiser les flots de mise en oeuvre des plateformes de vérification intéropérables. Dans un second temps sera donc proposé un flot automatisé de mise en oeuvre de plateformes de vérification interopérables.

Enfin, le chapitre précédent a également montré que le concept d'interopérabilité en émulation materielle ne peut être utilisable qu'en étant associé à un flot de vérification adapté. Ce chapitre se terminera donc par la proposition d'un nouveau flot de vérification orienté interopérabilité et, la preuve de son efficacité supérieure à celle des stratégies actuelles.

3.2) Notion d'état d'un circuit

Afin de faire coopérer les différentes techniques de vérification utilisant l'émulation et/ou le prototypage matériel, il faut assurer une cohérence entre les modèles éxécutés sur les différentes plateformes de vérification mises en jeu. La coopération maximale sera obtenue si on est capable d'arrêter un test utilisant une plateforme de vérification et de le reprendre sur une autre plateforme permettant l'utilisation de techniques mieux adaptées à la résolution des problèmes rencontrés. Cela implique la capacité d'accéder à l'état des circuits en cours de vérification : il faut être capable de capturer et de restaurer l'état des circuits.

L'état d'un circuit est caractérisé par des variables d'état. La connaissance de ces variables détermine parfaitement l'état du circuit à un instant donné et l'ensemble des signaux du circuit en sont déduits. Pour obtenir la coopération désirée, il faut donc précisément identifier ces variables.

L'ensemble des plateformes d'émulation/prototypage matériel ainsi que les simulateurs HDL utilisent comme point d'entrée de leur flot respectif une description du circuit au niveau RTL. Le modèle final exécuté sur une plateforme de vérification est donc un raffinement d'une description RTL. Le niveau RTL modélise un circuit comme un ensemble d'éléments mémoire reliés entre eux par de la logique combinatoire. Au final, une description RTL d'un circuit est une combinaison de machine d'états finis. Les dernières sont caractérisées par l'état de leurs éléments mémoires (registres, verrous, RAM), l'évolution de leurs signaux d'entrée et l'état des horloges pour les machines synchrones. Ces données caractérisent donc les variables d'états du niveau RTL. Il faut maintenant vérifier leur compatibilité avec les modèles utilisés en simulation et en émulation/prototypage matériel.

Les simulateurs actuels (NcSim, ModelSim) exécutent le code RTL en travaillant par événement. Si à un instant donné, on arrête la simulation et que l'on modifie la valeur de l'ensemble des éléments mémoire du circuit, ainsi que les valeurs des signaux d'entrée, cela va engendrer de nombreux évènements qui vont faire recalculer au simulateur l'ensemble des signaux combinatoires qui préparent l'état suivant. On peut donc ainsi, bien restaurer l'état d'un circuit en simulation.

En émulation/prototypage matériel, le modèle RTL est soit exécuté sur une machine qui a un fonctionnement comparable à celui d'un simulateur mais qui réalise les calculs en parrallèle (Palladium, Xserver) soit, le modèle est implémenté sur des composants programmables de type FPGA. Dans ce cas, le fonctionnement du circuit correspond parfaitement à celui du modèle RTL

au niveau des éléments mémoires. Seuls les signaux combinatoires changent en fonction des primitives disponibles. Quelle que soit la configuration, il y a donc toujours compatibilité entre les variables d'état de la description RTL et les variables d'états des modèles exécutés.

En conclusion, il faut retenir que l'état d'un circuit est caractérisé par trois éléments :

- l'état de l'ensemble des éléments mémoires (registres, verrous, RAM)
- l'évolution des signaux d'entrée
- l'état des horloges (phases, fréquences, formes d'onde)

3.3) Extraction des variables d'état : sélection du modèle de référence

Le précédent paragraphe a clairement identifié les variables d'état d'un circuit. En font partie les ressources mémoire. Il faut donc les identifier parmi l'ensemble des signaux mis en jeu dans le circuit. Ceci doit être réalisé à partir d'un modèle de réfèrence. Les prochains paragraphes présentent les différentes contraintes quant à la sélection de ce modèle de référence.

3.3.1) Hétérogénéité du niveau d'abstraction des modèles des composants du circuit

La mise en oeuvre d'une plateforme d'émulation/prototypage matériel peut se faire à partir de modèles décrits à deux niveaux d'abstraction et en utilisant différents langages de description. Les deux niveaux d'abstraction sont le niveau RTL décrit avec un langage HDL (VHDL ou Verilog) et le niveau portes logiques pouvant utiliser trois langages (VHDL, Verilog, EDIF). Vue la taille et la complexité des circuits actuels, il est courant que ceux-ci soient réalisés par différentes équipes. Chacune livre un sous-ensemble du circuit dans l'un des formats précédemment mentionnés.

On pourrait envisager d'extraire les éléments mémoire à partir de ces différentes données. Cependant, identifier les éléments mémoire d'un circuit à partir de sa description RTL n'est pas forcément aisé du fait du haut niveau d'abstraction. La description au niveau des portes logiques utilise des primitives liées à la plateforme d'exécution. Parmi elles se trouvent des primitives mémoires. Ce niveau de description est donc plus adapté à l'identification des variables d'état.

D'autre part, une fois récupéré l'ensemble des fichiers de description du circuit, la première étape de mise en oeuvre d'une plateforme d'émulation/prototypage matériel consiste à réaliser une synthèse du circuit. Le résultat est une description au niveau portes logiques du circuit, les primitives utilisées étant spécifiques à la technologie de la plateforme de vérification. Etant donné que l'étape de synthèse est obligatoire et qu'elle unifie les différentes descriptions des sous-ensembles du circuit en un seul fichier de description au niveau portes logiques, il devient évident que la caractérisation des éléments mémoires doit se faire après l'étape de synthèse.

3.3.2) Problèmes de cohérence d'état liés à disparité des résultats de synthèse

Dans le chapitre II, la section «2.7.4.1) Problème de cohérence des modèles» a mis en évidence cinq raisons de non cohérences des résultats de synthèse :

- simplification de registres
- duplication de registres
- ajout de ressources de multiplexage
- hétérogénéité des primitives matérielles
- non préservation des noms et hierarchies

Ainsi, la caractérisation de l'état d'un circuit est certes plus facile à réaliser à partir d'un résultat de synthèse mais, il n'y a aucune garantie quant à la cohérence de l'extraction des variables d'états entre deux résultats de synthèse différents.

3.3.3) Conclusion : la caractérisation de l'état d'un circuit ne peut se faire qu'après synthèse

Une première solution pour pallier au problème de cohérence des différents résultats de synthèse serait d'avoir un outil capable de créer une sorte de filtre rendant compatible entre elles plusieurs «netlists» au niveau des variables d'état. Ainsi, pour changer de plateforme de vérification, il faudrait sauvegarder l'état du circuit dans le premier environnement, appliquer un filtre d'adaptation sur le fichier de sauvegarde pour le rendre compatible avec le deuxième environnement et enfin restaurer l'état du circuit sur la deuxième plateforme de vérification. Cependant, une telle solution nécessite le développement d'un outil capable de comparer les ressources mémoires de deux netlists, ce qui engendre une énorme charge de travail. Cette solution n'a donc pas été retenue.

La proposition précédente engendre un format de sauvegarde d'état par plateforme de vérification et, la difficulté réside dans la cohérence de ces formats de sauvegarde. Il serait plus simple de gérer un seul et unique format de sauvegarde d'état compatible avec toutes les plateformes de vérification : il n'y aurait plus de problème de cohérence. Il faut donc utiliser un seul et unique résultat de synthèse comme référence pour la caractérisation des variables d'état.

D'autre part, une fois les variables d'état identifiées, il va falloir déployer une architecture spécifique permettant l'accès en lecture et écriture à ces variables. Les flots de mise en oeuvre des plateformes d'émulation/prototypage matériel garantissent une cohérence minimale entre le modèle qu'elles exécutent et le modèle RTL qui aura servi de point d'entrée au flot. Cette cohérence minimale est au niveau des signaux d'entrée/sortie. Cela signifie que même si les modèles exécutés sur deux plateformes différentes d'émulation/prototypage matériel présentent une grande disparité quant aux ressources mémoires utilisées, ces deux environnements produiront le même comportement au niveau des sorties du circuit. Ainsi, si l'architecture du circuit inclut la possibilité d'accéder aux variables d'état et que cet accès est ramené sur les ports d'entrée/sortie du circuit, on s'affranchit de l'ensemble des disparités des étapes de synthèse spécifiques à chaque plateforme de vérification.

Au final, la netlist de référence devra être obtenue indépendamment de toute plateforme de vérification. Cela impose de réaliser une synthèse sans contrainte de placement, fréquence et partitionnement. Cette netlist de référence servira donc à la caractérisation des variables d'état et sera instrumentée de manière à permettre l'accès à ces variables. La netlist instrumentée servira alors de point d'entrée au flot spécifique de chacune des plateformes d'émulation/prototypage matériel mises en jeu. La complexité du travail de caractérisation des variables d'état est donc ainsi ramenée à lire une description de circuit au niveau portes logiques et à identifier l'ensemble des ressources mémoires allouées dans cette description. Cette solution est plus simple à mettre en oeuvre que la première méthode proposée car il n'y a pas besoin de filtre. Par contre, elle impose la réalisation d'une synthèse supplémentaire, à savoir, la synthèse de référence, ce qui va ajouter quelques heures de travail au flot de vérification. Néanmoins ce surcoût reste acceptable.

D'autre part, le contexte de ce travail de recherche est la conception des systèmes sur puce ASIC. En conséquence, il faudra tôt ou tard réaliser une synthèse orientée ASIC. Cette étape de synthèse étant la seule obligatoire dans le flot de conception, on peut l'utiliser pour caractériser les variables d'état du circuit.

Enfin, il est à noter que les outils de synthèse dédiés aux émulateurs (exemple : HdlIce pour Palladium) sont souvent très rapides par rapport aux autres outils de synthèse. De plus la synthèse pour émulateur n'impose pas de contrainte spécifique à la machine : par conséquent, ce genre d'outil est particulièrement bien adapté à l'obtention rapide du modèle de référence du circuit permettant la caractérisation des variables d'état. Dans la suite de cette thèse, ce modèle de référence sera généré par l'outil HdlIce.

L'inconvénient majeur de la méthode proposée réside dans la disparité des primitives

utilisées par les différentes plateformes d'émulation/prototypage matériel. En effet, avant d'obtenir une implémentation du circuit sur plateforme de vérification, il faudra réaliser deux synthèses : la synthèse de référence puis la synthèse de la netlist de référence sur les primitives de la plateforme cible. Il se peut que la première synthèse conduise à un résultat qui donnera de moins bonnes performances en terme de ressources et vitesse sur la deuxième synthèse. Par exemple, l'émulateur Palladium n'incorpore pas de primitive de multiplication alors que les FPGAs sont munis de multiplieurs rapides. Ainsi, si un circuit utilise un opérateur de multiplication, cet opérateur sera réalisé à l'aide d'additionneurs sous Palladium. Utiliser le résultat de synthèse Palladium comme point d'entrée d'une synthèse FPGA impliquera donc une non utilisation des multiplieurs des FPGAs. Heureusement ce problème d'hétérogénéité des ressources ne concerne que quelques ressources particulières. Par conséquent, si un problème de ressource apparaît à cause de ce point, un traitement particulier de quelques primitives pourrait être envisagé (notamment pour la gestion des multiplieurs).

3.4) Accès à l'état d'un circuit

3.4.1) Cahier des charges

Le précédent paragraphe a introduit le besoin de recourir à une architecture spécifique permettant l'accès en lecture et écriture aux variables d'états. De nombreuses solutions sont envisageables pour répondre à ce défi. Les prochains paragraphes vont donc mettre en évidence l'ensemble des besoins afin de déterminer les caractéristiques de l'architecture permettant l'accès aux variables d'état ce qui permettra de réduire l'espace des solutions.

3.4.1.1) Préservation de la vitesse : capture d'état dynamique

Les techniques d'émulation/prototypage matériel sont nombreuses et ont toutes des contraintes spécifiques. Si on veut les faire coopérer, il faut être capable de transposer l'état d'une machine vers une autre quelles que soient les contraintes de fonctionnement.

Les techniques utilisant des dépendances externes (émulation en mode ICE) sont parmi les plus contraignantes puisque le circuit prototypé est relié à un environnement extérieur et qu'il n'est absolument pas possible de modifier la vitesse d'exécution du prototype sous peine de perdre la synchronisation avec l'environnement extérieur. Ce genre d'application est particulièrement nécessiteux en coopération car, comme on ne peut pas altérer la fréquence de fonctionnement du système, les possibilités de déboguage sont fortement réduites. Prenons l'exemple suivant : un circuit de traitement vidéo est prototypé à 30MHz et est stimulé par une caméra. Les besoins en vitesse d'un tel environnement de vérification font que la machine choisie est une plateforme de prototypage basée FPGA tel un MP4 d'Aptix. Ce genre de machine possède une très faible capacité de déboguage à savoir que seuls quelques signaux sont observables et ce, uniquement sur une fenêtre temporelle de capture. Par conséquent, si le système comporte une erreur matérielle, il faudra probablement rejouer le scénario de nombreuses fois en modifiant les signaux observés à chaque itération avant de pouvoir clairement identifier l'origine du problème. Dans le pire des cas, l'erreur peut ne pas se reproduire, si elle est liée au contenu de l'image filmée, vu que ce contenu ne sera pas rigoureusement le même entre deux essais.

Au contraire, en capturant l'état du circuit quelques cycles d'horloges avant l'instant d'apparition du bogue et en échantillonnant les signaux issus de la caméra on sera capable de reproduire ce test dans un environnement plus adapté comme un émulateur ou un simulateur. L'identification de l'erreur ne demandera alors qu'une seule et unique passe et ne présentera pas le problème de reproductibilité.

Ainsi, pour couvrir ce genre de cas, il faut être capable de capturer l'état des circuits émulés

sans arrêter les machines, il faut donc réaliser des captures d'état dynamiques, des captures d'état en temps réel. Ceci inclut la récupération des valeurs de l'ensemble des éléments mémoires et de l'état des horloges à l'instant de capture, mais également un échantillonnage en temps réel des signaux entrant en provenance d'un environnement extérieur à la plateforme d'émulation/prototypage.

Enfin, vu le nombre de données impliquées par une capture d'état (autant que de bits mémoire), il devient vite évident qu'il ne sera pas possible d'extraire un état en un seul et unique cycle d'horloge. Ce point n'engendre pas de nouvelles contraintes étant donné qu'il n'y a aucun intérêt à capturer l'état d'un circuit à tous les cycles d'horloges.

3.4.1.2) Gestion de la bande passante : non altération des co-émulations

L'extraction de l'état d'un circuit nécessite un lien de communication afin de stocker cet état sur le disque dur d'un ordinateur. Aujourd'hui, l'ensemble des plateformes d'émulation/prototypage matériel intègre un lien de co-émulation qui semble parfaitement adapté à cet acheminement des captures d'état.

D'autre part, la capture d'état est également nécessaire dans les co-émulations, plus particulièrement dans les co-émulations transactionnelles. La section «2.3.5.5) Co-émulation transactionnelle» a montré que les performances de ce type de vérification dépendent fortement de la charge de calcul de l'environnement logiciel et de la bande passante du lien de communication entre logiciel et matériel. Altérer la bande passante peut retarder l'échange de certaines transactions. Si ce retard est très important, la vérification peut passer à coté de de certaines erreurs. Ainsi, une surcharge d'utilisation de la bande passante peut se révéler fortement intrusif.

L'extraction d'état doit avoir un impact minimal sur les co-émulations ce qui impose d'utiliser uniquement les temps morts du lien de communication. Ainsi, le temps d'extraction ne sera pas déterministe mais dépendra du taux d'utilisation de la bande passante.

En conclusion, le système d'extraction d'état devra intégrer un contrôleur de bande passante capable de suspendre la transmission d'un état si cette transmission commence à devenir intrusive. En outre, l'architecture d'extraction sera capable de mémoriser un état afin d'attendre que le bus de transfert soit libre pour transmettre l'état capturé.

3.4.1.3) Gestion des horloges contrôlées

Parmi les contraintes de conception, l'économie d'énergie est une des plus importantes. Cela est même essentiel dans des circuits pour téléphone portable car baisser la consommation d'un circuit permet d'augmenter l'autonomie du téléphone.

Une des techniques classiques d'économie d'énergie consiste à couper les horloges d'un ou plusieurs modules non utilisés. Ainsi, lorsque l'on réalise un accès en lecture ou en écriture à l'état d'un circuit, il se peut qu'une partie du circuit ait ses horloges coupées. Par conséquent, même si le circuit est un circuit synchrone, l'architecture d'accès à l'état ne pourra pas utiliser les mêmes domaines d'horloges que ceux du circuit, les accès devant pouvoir se faire même si les horloges du circuit sont coupées.

3.4.1.4) Limitation du temps d'accès et contrôlabilité du pas d'échantillonnage

Si l'on veut obtenir des coopération exploitables, il ne faut pas consommer le bénéfice apporté par la coopération des techniques en changeant de machine. Cela impose de capturer et restaurer un état en un temps acceptable.

En général, l'initialisation d'un environnement d'émulation/prototypage matériel nécessite quelques minutes. Il serait donc bien de conserver cet ordre de grandeur, ce qui signifie être capable de restaurer l'état d'un circuit en environ une minute.

Concernant le temps d'une capture d'état, analysons les besoins afin de déterminer ce qui est acceptable. Typiquement, les besoins de coopération se font sentir sur les longs tests. Si une erreur arrive après douze heures de test, il faut reproduire cette erreur dans un environnement plus propice au déboguage et si possible, redémarrer le test quelques cycles d'horloges avant l'instant d'apparition du bogue. Si l'on travaille sur des tests de plusieurs heures, échantillonner l'état du circuit en cours de vérification avec une granularité de l'ordre de quelques minutes est largement suffisant.

Considérons un exemple utilisant une plateforme de prototypage pour l'exécution du test et un émulateur pour le déboguage. Si l'environnement de déboguage a une vitesse d'exécution comparable à celui de test, la durée de travail sur émulateur nécessitera quelques minutes, ce qui est tout à fait acceptable. Si au contraire, on constate un important écart entre la vitesse d'exécution des deux environnements de vérifications, par exemple un prototype à 100MHz et un émulateur à 100kHz, rejouer une fenêtre de test qui durerait initialement une minute sur plateforme de prototypage nécessitera dix-sept heures en émulation, ce qui est nettement moins acceptable.

Cependant, on peut adopter une stratégie résolvant ce problème. Lorsqu'une erreur se manifeste, on peut dans un premier temps rejouer le test entre deux captures sur l'environnement de test rapide. Dans cette passe, on va augmenter le nombre de captures d'état dans la fenêtre rejouée. Cela nécessite de ralentir la vitesse d'exécution du test de manière à tenir la cadence de capture. Ainsi, on trouvera un instant de reprise de test beaucoup plus proche de l'instant d'apparition de l'erreur. Dans l'exemple précédent, si l'on réalise vingt captures d'état dans la deuxième passe, cette dernière nécessitera environ vingt minutes au lieu d'une dans le test intial. La séquence nécessaire au déboguage jouée sur émulateur sera vingt fois plus courte et demandera environ cinquante minutes. Le temps global de déboguage sera alors d'environ une heure, ce qui est tout à fait acceptable. Cette stratégie sera plus amplement développée dans le paragrapge «3.4.2) Phase de vérification et phase de déboguage».

En conclusion, un temps d'accès à l'état d'un circuit de l'ordre de la minute est souhaitable. De plus, afin d'augmenter l'efficacité du déboguage, il est nécessaire de pouvoir parfaitement contrôler le pas d'échantillonnage des captures. L'architecture de capture devra donc être capable de ralentir, voire d'arrêter le circuit lors des reproductions de test afin de tenir la cadence d'échantillonnage visée.

3.4.1.5) Gestion du banc de test et des dépendances externes

Les émulateurs et plateformes de prototypage offrent de nombreux modes de fonctionnement. Afin de procurer un maximum de coopération entre l'ensemble des techniques il est important que la capture et la restauration d'état soit possible quel que soit le mode de fonctionnement des machines. Parmi les différents modes de fonctionnement on trouve les co-émulations qui stimulent le circuit prototypé à l'aide d'un banc de test logiciel exécuté sur un ordinateur relié à la machine d'émulation/prototypage.

Capturer l'état d'un banc de test logiciel n'est pas aisé, d'autant plus que celui-ci peut être décrit à différents niveaux d'abstraction et avec plusieurs langages (TLM, C, C++, VHDL, Verilog). Durant l'étape de vérification, le but est de vérifier le comportement du circuit, le fonctionnement du banc de test n'a aucune importance. Par conséquent, si un besoin de coopération entre techniques se fait sentir, afin d'augmenter l'efficacité du déboguage en rejouant le test dans un environnement adapté, on n'est pas obligé d'utiliser le banc de test associé au circuit. Une modélisation des stimulations du circuit suffit à reproduire le test.

De plus, lorsque la plateforme d'émulation/prototypage matériel est relié à un véritable composant externe tel une caméra vidéo, la seule possibilité pour reproduire au bit et au cycle près les stimulations générées par de tels composants est de les enregistrer.

En conclusion, les parties du circuit à vérifier sont celles qui présentent potentiellement des

bogues et sont obligatoirement émulées/prototypées. L'ensemble des éléments hors émulateurs sont juste là pour stimuler le circuit à tester. Assurer une coopération implique de reproduire ces stimulations. Afin d'assurer ce besoin, il faut donc enregistrer, pendant la phase de test, l'ensemble des signaux de stimulations qui ne sont pas générés par l'émulateur. La reproduction des scénarios de test se fera alors dans un environnement de co-émulation de type «vecteurs de test», les vecteurs fournis étant l'ensemble des stimulations du circuit à chaque cycle d'horloge.

Enfin, échantillonner les signaux d'entrée d'un circuit peut représenter un grand volume de données et nécessiter une grande passante pour le transfert sur ordinateur. Dans certains cas, il sera souhaitable que le système d'échantillonnage puisse également compresser en temps réel ces données.

3.4.2) Proposition d'une solution d'accès aux ressources mémoire

Les précédents paragraphes ont introduit les besoins de l'architecture d'accès aux variables d'état. Il a notamment été vu que l'accès en lecture aux ressources mémoires (registres, verrous, RAMs) doit être réalisé dynamiquement, c'est à dire sans arrêter le système. De plus, la gestion de la bande passante des liens de co-émulations impose que l'extraction d'une capture d'état puisse se faire sur un nombre de cycles d'horloge variable, dépendant du taux d'utilisation de la bande passante. Ces deux contraintes motivent le déploiement d'une architecture adaptée qui va être présentée dans les prochains paragraphes.

3.4.2.1) Duplication des ressources mémoire

L'architecture d'accès à l'état d'un circuit doit pouvoir mémoriser un état complet. Pour cela, il faut doubler la capacité mémoire du circuit. Etant donné le volume de données à enregistrer en un seul et unique cycle d'horloge, il n'est pas possible d'utiliser une RAM pour stocker un état. En effet, les circuits actuels peuvent contenir plusieurs centaines de milliers de registres sans compter plusieurs giga-octets de mémoire RAM. Mémoriser un tel volume de données nécessiterait une mémoire d'un seul et unique mot de plusieurs giga-bits de large, ce qui est irréalisable. De plus, cela générerait de nombreuses contraintes de routage pour amener les signaux échantillonnés sur le port d'écriture de la mémoire.

La seule solution viable trouvée consiste à dédoubler toutes les ressources mémoire. A chaque bascule et verrou, on associe une bascule de capture et à chaque RAM on associe également une RAM de capture. En fonctionnement normal, chaque écriture dans un élément mémoire engendre l'écriture de la même donnée dans la mémoire doublon. Lorsque l'on capture un état, l'écriture dans les doublons est suspendue. Il suffit alors d'extraire les valeurs des mémoires doublons afin d'extraire l'état du circuit. Une fois extrait cette action terminée, il faudra rafraichir l'état des RAMs doublons et les autoriser à nouveau en écriture avant de pouvoir effectuer une nouvelle capture d'état.

3.4.2.2) Gestion des horloges

L'accès en lecture à l'état du circuit émulé/prototypé doit se faire indépendamment des horloges du circuit qui peuvent être coupées pour des besoins d'économie d'énergie. La mémorisation d'un état doit se faire en un seul front d'horloge, l'horloge utilisée étant spécifique à l'architecture de capture d'état ce qui implique des problèmes d'instant d'échantillonnage.

D'autre part, pour reproduire un test, il faut mémoriser l'état des horloges, c'est à dire connaître pour chaucne des horloges sa phase, sa fréquence et sa forme d'onde.

La norme SceMi, évoquée dans le chapitre II, permet de résoudre simplement et efficacement ces deux problèmes. Les deux prochains paragraphes vont détailler cette solution. De

plus, utiliser des hypothèses SceMi présente un gros avantage d'implémentation vu que cette norme de communication est supportée par l'ensemble des solutions récentes en matière d'émulation/prototypage matériel.

3.4.2.2.1) Norme SceMi : exploitation des concepts uclock et cclock

La norme SceMi propose un mécanisme d'horloges intéressant reposant sur deux types d'horloge (figure 14), une horloge non contrôlée dite «uclock» (uncontrolled clock) et des horloges contrôlées de type «cclock» (controlled clock).

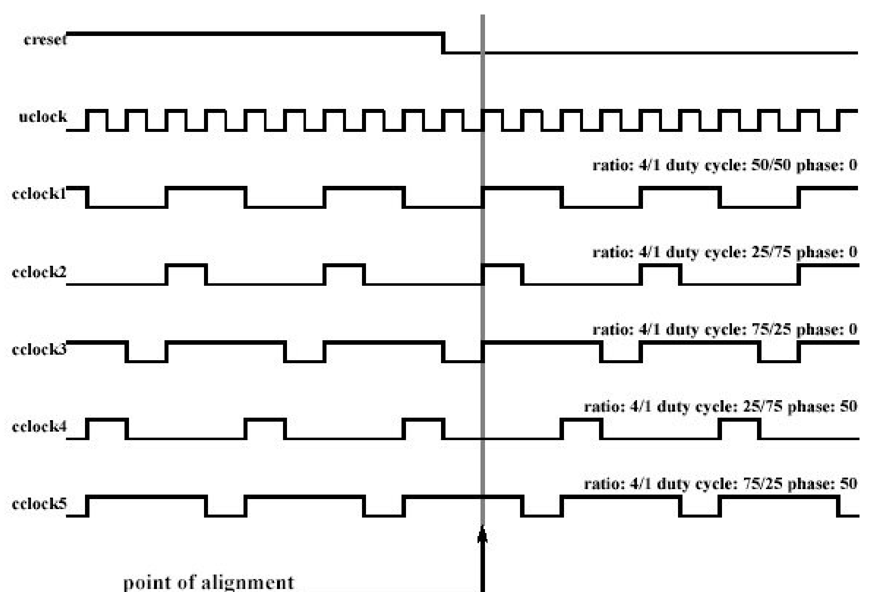


Figure 14 : Exemple d'horloges générées avec SceMi

L'horloge uclock est réservée à la gestion de la communication entre ordinateur et émulateur. Cette horloge est libre, c'est à dire que l'utilisateur n'a pas la possibilité de l'arrêter. L'ensemble des horloges du circuit en cours de vérification sont quant à elles des horloges de type cclock. Les horloges cclock offrent plusieurs propriétés par rapport à l'horloge uclock. Tout d'abord elles peuvent être arrêtées par le transacteur contrôlant le lien de communication. De plus, elles sont toutes synchrones avec uclock c'est à dire que chacune des horloges cclock a obligatoirement ses fronts actifs en même qu'un front montant de l'horloge uclock. Le précédent graphique montre cinq horloges générées par SceMi. Chacune de ces horloges possède sa propre forme d'onde, fréquence et phase mais, toutes ont leurs fronts montants et descendants en même temps qu'un front montant de l'horloge uclock.

En se plaçant dans un contexte SceMi, on peut utiliser l'horloge uclock comme horloge d'échantillonnage d'état. On réalisera alors une lecture synchrone de l'état du circuit et on s'affranchit ainsi des problèmes de changement de domaine d'horloge.

Cependant, l'horloge de capture d'état uclock est beaucoup plus rapide que les horloges cclock du circuit. Si cette horloge de capture d'état échantillonne l'entrée des registres, la capture ainsi réalisée verra des valeurs transitoires qui ne seront jamais prises par les registres. Les registres doublons pourraient donc mémoriser un faux état (figure 15).

Imaginons un registre actif sur front montant d'horloge cclock. L'horloge cclock utilisée est quatre fois plus lente que l'horloge uclock. Le signal arrivant sur l'entrée «D» du registre provient d'une partie du circuit utilisant une horloge plus rapide que celle pilotant le registre. Ainsi, si l'on

regarde les valeurs prises par le signal «Dech», échantillonnage de l'entrée du registre sur front montant de uclock, cet échantillonnage prend toutes les valeurs de «D» or, ces valeurs ne seront pas toutes transmises à la sortie «Q» de la bascule.

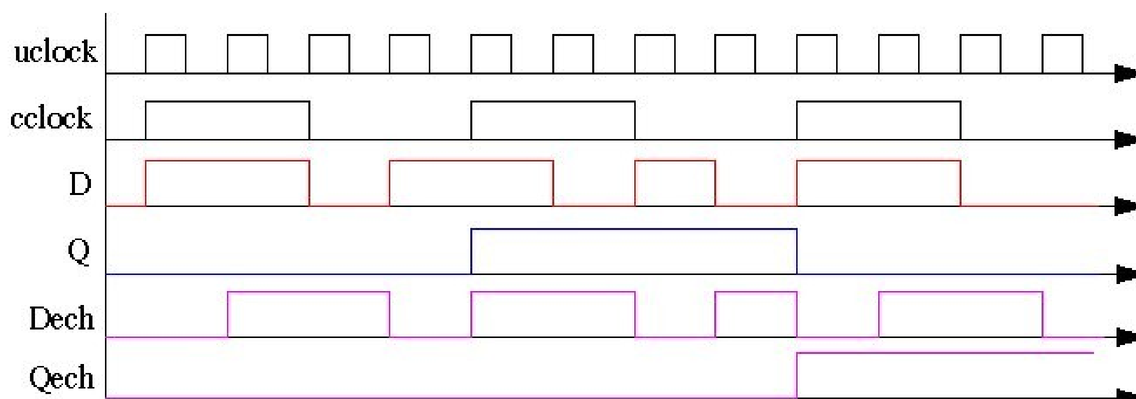


Figure 15 : Sélection des signaux échantillonnés

Une première solution consisterait à valider un échantillonnage si un front de cclock est détecté. Une telle solution serait fortement coûteuse en ressource matérielle. Par contre, en échantillonnant la sortie des registres du circuit (signal «Qech»), on mémorise l'état précédent tout en s'affranchissant de ce problème de signaux transitoires grâce à la stabilité de la sortie des registres. Ainsi, toute écriture dans un élément mémoire du circuit se traduit par la copie des nouvelles valeurs dans les doublons au cycle d'horloge de capture d'état (uclock) suivant.

D'autre part, il est très difficile de synchroniser deux horloges de manière à ce qu'elles aient leurs fronts synchrones comme le prévoit SceMi. En pratique, les solutions SceMi ne garantissent ce synchronisme que sur des petites parties du circuit. L'ensemble des horloges du circuit de type cclock sont dérivées de l'horloge de capture d'état uclock. Ainsi, les fronts de l'horloge uclock sont en avance sur ceux des horloges cclock. La lecture d'une variable d'état se fera donc toujours un peu avant le front faisant évoluer cette donnée, les temps de pré-position et de maintien seront systématiquement respectés. Le contexte SceMi permet donc de résoudre les problèmes d'échanges de données entre deux domaines d'horloge différents en réglant l'instant d'échantillonnage.

Le système de capture d'état doit, en outre, pouvoir fonctionner quel que soit le type d'émulation utilisé, ce qui inclut les co-émulations transactionnelles basées sur SceMi. Dans ce cas, on ne peut ni contrôler ni suspendre l'horloge uclock. Cependant, la section «3.4.1.4) Limitation du temps d'accès et contrôlabilité du pas d'échantillonnage» a mis en évidence un besoin de contrôlabilité de la totalité des horloges du circuit lorsque l'on effectue une deuxième passe de capture d'état. Il faut donc rendre contrôlable l'horloge uclock pour ce genre d'application. Une solution à ce problème consiste à utiliser une surcouche SceMi. La première couche SceMi est liée à l'architecture d'accès à l'état du circuit qui fonctionne sur horloge uclock. Elle gènère une horloge cclock qui servira d'horloge uclock pour la deuxième couche. La deuxième couche SceMi est liée au circuit. Le transacteur de ce circuit fonctionne donc sur une horloge uclock qui est en fait l'horloge cclock de la première couche. Le circuit fonctionne lui sur les horloges cclock de la deuxième couche SceMi. Une telle solution répond parfaitement au problème recontré. Pour sa mise en application, il faudra réaliser un générateur d'horloges SceMi, ce qui ne représente pas une grande charge de travail. Ce générateur a en effet été réalisé en une journée pour les besoins d'un essai.

En conclusion, l'horloge uclock défini par SceMi conviendra parfaitement comme horloge d'échantillonnage d'état.

3.4.2.2.2) Norme SceMi : présentation et extension du concept «cycle stamp»

Maintenant que l'on sait quand échantillonner l'état d'un circuit, il faut pouvoir enregistrer l'état des horloges, c'est à dire mémoriser la phase de chacune des horloges du circuit au moment de la capture. Le précédent paragraphe a limité l'utilisation des horloges au cadre SceMi où chacune des horloges cclock est synchrone avec l'horloge uclock.

En plus du concept d'horloges uclock et cclock, SceMi définit également un concept de «cycle stamp». Ce «cycle stamp» est un compteur permettant de savoir précisément où l'on est dans la vérification du point de vue du circuit (figure 18).

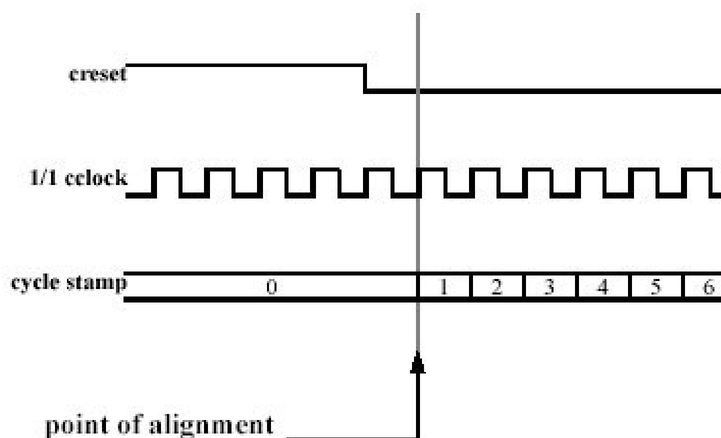


Figure 16 : Concept SceMi "cycle stamp"

Ce compteur est incrémenté sur front montant d'une horloge 1/1 cclock qui présente des propriétés analogues à celle de l'uclock vis à vis des autres horloges cclock, c'est à dire que l'ensemble des horloges cclock ont leurs fronts actifs synchrones avec un front montant de l'horloge 1/1 cclock. Par contre, l'horloge 1/1 cclock, comme toute horloge de type cclock peut être suspendue. Le compteur «cycle stamp» est initialisé à zéro et est incrémenté seulement à partir du point d'alignement. Le point d'alignement est l'instant pour lequel le signal de reset est relaché. A cet instant, l'état des horloges est parfaitement connu.

Ainsi, le «cycle stamp» est le nombre de cycles d'horloge 1/1 cclock effectué depuis le point d'alignement. Vu que l'on connait l'état des horloges au point d'alignement, que l'on connait aussi le cycle d'évolution des horloges et leur périodicité en terme de nombre de cycles de 1/1 cclock, le «cycle stamp» caractérise parfaitement l'état des horloges cclock.

Cependant, on peut avoir à capturer l'état d'un circuit interagissant avec un transacteur. Dans ce cas, l'état du transacteur est également à capturer. La connaissance du «cycle stamp» permet de caractériser l'état du circuit mais pas celui du transacteur car le «cycle stamp» n'est pas incrémenté à tous les fronts de uclock. Pour répondre à ce problème, on peut redéfinir le «cycle stamp» comme un compteur incrémenté depuis le point d'alignement sur front montant de uclock. Cependant, un tel «cycle stamp» perd l'information d'état des horloges cclock puisque ce nouveau compteur est incrémenté même quand les horloges cclock sont suspendues.

La solution finale caractérisant parfaitement l'instant d'échantillonnage et l'état des horloges repose donc sur deux compteurs, que nous appelerons par la suite «ustamp» (uncontrolled cycle stamp) et «cstamp» (controlled cycle stamp) (figure 17). Le compteur «ustamp» est un «cycle stamp» incrémenté à chaque front montant d'horloge uclock et identifie l'instant d'échantillonnage. Le compteur «cstamp» correspond au «cycle stamp» SceMi et caractérise l'état des horloges à un instant de capture.

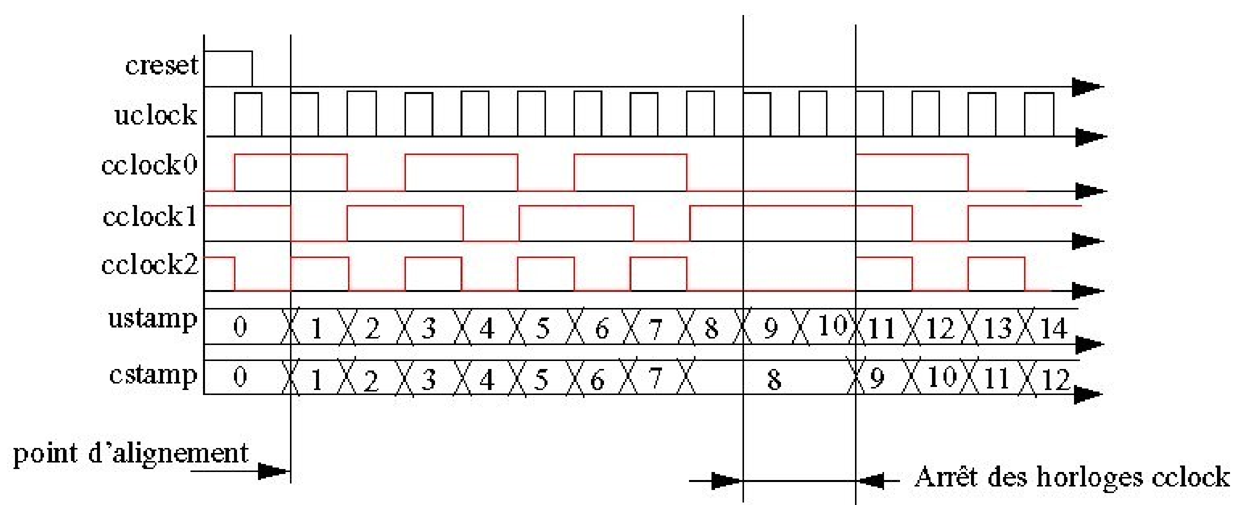


Figure 17 : Exemple de compteur ustamp et cstamp

3.4.2.3) Extraction d'état par chaîne de scan

Maintenant que l'on a résolu le problème de mémorisation de l'état d'un circuit, il faut extraire les données. On pourrait envisager d'insérer un bus dédié à ce besoin. Cependant, une telle solution aura un mauvais impact sur la plateforme de vérification, particulièrement sur les plateformes de prototypage multi FPGAs. En effet, ajouter un bus augmente le nombre de signaux d'interconnection entre les FPGAs, ce qui peut augmenter le facteur de multiplexage et ainsi diminuer la vitesse d'exécution du circuit prototypé. Au final, pour minimiser l'impact de l'architecture d'accès à l'état d'un circuit, cette architecture doit insérer le minimum de composants dans le circuit. L'idéal serait de se contenter d'un seul et unique fil pour accéder à un état.

Avec cette hypothèse d'un seul bit d'extraction, il faut vérifier qu'il est bien possible de capturer un état en quelques minutes au maximum. La taille des circuits émulés peut atteindre jusqu'à vingt millions de portes logiques et incorporer jusqu'à un million de registres et plusieurs dizaines de méga-octets de mémoire RAM. L'ordre de grandeur du volume de données à extraire est donc d'environ cent millions de bits pour les plus gros circuits. Typiquement, les environnements nécessitant des captures d'états fonctionnent au moins à 100kHz, et vraisemblablement à plusieurs méga-hertz. Le précédent chapitre à montré que l'horloge de capture d'état est l'horloge la plus rapide du système. En prenant l'hypothèse du pire cas, à savoir capturer cent millions de bits à une fréquence de 100kHz, il faudra environ dix-sept minutes pour réaliser une capture. Ce temps reste acceptable si les tests durent une à plusieurs journées. A priori, les besoins de coopération se font surtout sentir avec les plateformes de prototypage qui, pour un tel circuit, auront une fréquence de fonctionnement minimale d'au moins 1MHz. A cette cadence, une capture d'état nécessitera moins de deux minutes, ce qui est tout à fait correct vis à vis des contraintes précédemment identifiées. L'hypothèse d'utiliser un seul bit pour accéder à l'état d'un circuit est donc validée.

Une technique classique de test des circuits remplie parfaitement les objectifs fixés : l'utilisation des chaînes de scan [TOR02], [SIR04] (figure 18). Afin de vérifier les circuits fabriqués, cette technique consiste à relier ensemble toutes les bascules du circuit afin de constituer un immense registre à décalage. Ce registre à décalage sert alors à insérer des vecteurs de test ainsi qu`à extraire les vecteurs résultats du test. Cette technique présente l'avantage d'être efficace et peu coûteuse en surface vu qu'elle n'ajoute qu'un mutliplexeur par registre du circuit.

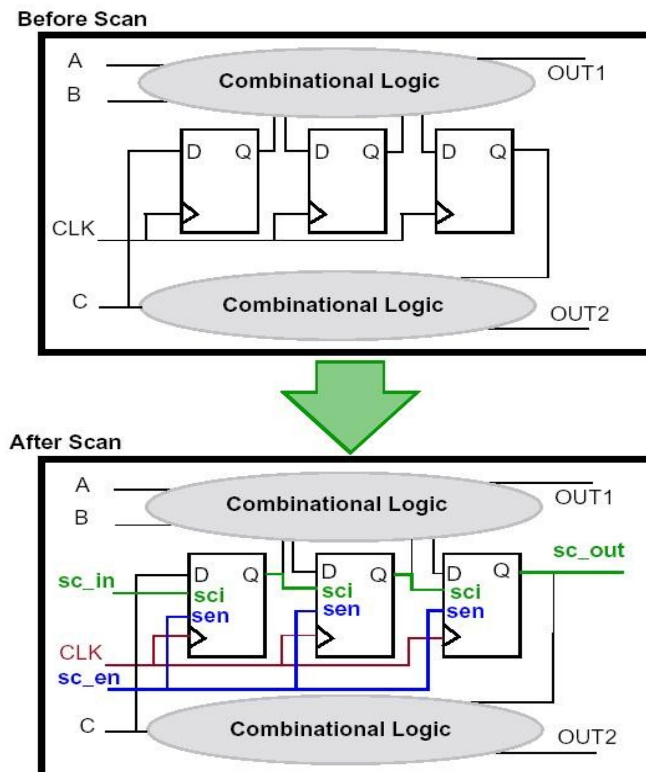


Figure 18 : Introduction d'une chaîne de scan dans un circuit

On peut reprendre ce concept pour l'extraction d'état du circuit, en reliant chacun des registres doublons entre eux. Pour les mémoires RAM, l'idée est aussi valable. Pour cela, il faudra ajouter un contrôleur mémoire à chacune d'elles, le contrôleur devrant transmettre les données sur la chaîne de scan.

3.4.2.4) Restauration d'état

Les précédents paragraphes ont identifié les besoins de la capture d'état, il faut également identifier ceux de la restauration d'état. Etant donné que cette dernière a lieu dans un environnement de déboguage, c'est à dire sans contrainte de stimulation temps réel ou d'intrusivité, la seule contrainte est d'initialiser le système en un temps acceptable, c'est à dire une durée de l'ordre de quelques minutes.

Les environnements de reproduction vont être principalement utilisés pour faciliter le déboguage. Le deuxième chapitre a dressé l'état de l'art sur les capacités de déboguage des machines et globalement, plus elles sont grandes et plus la vitesse d'exécution est faible.

Ainsi, deux cas apparaissent : la restauration d'état sur des machines à haute vitesse d'éxécution et la restauration d'état sur les machines à vitesse d'exécution plus modérée, voire lente.

Si l'environnement de reproduction est un environnement rapide (prototypage ou émulation) avec une vitesse d'exécution de l'ordre du méga-hertz, on se retrouve avec des contraintes comparables à celle de la capture d'état. Par conséquent, étant donné que l'on a inséré des chaînes de scan pour l'extraction des états, on peut utiliser ces mêmes chaînes pour la restauration des états. Le temps de restauration sera acceptable.

Au contraire, si l'environnement de reproduction est plutôt lent (simulation, accélération, émulation), c'est à dire avec des vitesses d'exécution inférieures à la centaine de kilohertz, on ne pourra plus utiliser les chaînes de scan, le temps d'initialisation deviendrait bien trop important. Le

cas le plus extrême est celui de la simulation qui a une vitesse d'exécution au mieux de l'ordre de 100Hz avec les gros circuits. Si l'on reprend l'exemple du précédent paragraphe où la taille des éléments mémoire atteint cent millions de bits, il faudrait alors environ quarante jours de simulation pour intialiser l'état du circuit par des possibilités architecturales, ce qui est complétement inconcevable.

La solution la plus efficace serait de pouvoir générer une base de données pré-initialisée avec l'état désiré pour chacune des machines visées. Cependant, une telle solution n'est pas viable car elle nécessiterait d'une part, un générateur par machine et d'autre part, une connaissance des formats des bases de données utilisées par chacune des machines or, ces formats ne sont pas publics.

Rappelons que l'un des gros avantages des machines spécialisées dans le déboguage est d'offrir une grande observabilité et contrôlabilité. Ces machines permettent notamment de forcer la valeur des signaux à l'aide d'une simple ligne de commande. On peut donc envisager d'utiliser des scripts afin d'initialiser un état.

Des mesures de temps de forçage de signaux à l'aide de scripts ont été réalisées avec le simulateur HDL NcSim de Cadence ainsi qu'avec l'émulateur Palladium de Cadence. En simulation, on a mesuré une capacité moyenne de forçage de registres d'environ 60 kbits/s. Avec Palladium, la mesure a donné environ 3kbits/s. Ainsi, avec la méthode proposée, initialiser un million de registres nécessiterait moins de vingt secondes en simulation et moins de sept minutes en émulation. Il est à noter que les temps d'initialisation des mémoires RAMs sont très rapides, de l'ordre de quelques secondes pour les plus grosses RAMs de plus de 10Mo. Recourir à des scripts est donc une solution viable pour restaurer un état dans un environnement à faible vitesse d'exécution.

En conclusion, en fonction de la vitesse d'exécution de la plateforme de déboguage, on utilisera soit des chaînes de scan, soit des scripts pour restaurer un état. Dans tous les cas, le temps d'initialisation restera acceptable, c'est à dire inférieur à la dizaine de minutes.

3.4.2.5) Cellules spécifiques à l'interopérabilité

La mise en oeuvre du concept d'interopérabilité nécessite d'instrumenter les circuits afin d'accèder en lecture et écriture à leur état. Cette instrumentation repose sur l'utilisation de mémoires doublons reliées par une chaîne de scan.

La synthèse génère une description d'un circuit comme un assemblage de primitives spécifiques à la machine ciblée. Dans le cadre de cette thèse, la synthèse de référence est réalisée par l'outil HdlIce de Cadence, outil de synthèse spécifique à l'émulateur Palladium. La bibliothèque de primitives est nommée «qtref».

Le processus d'instrumentation du circuit consiste à remplacer chaque primitive mémoire de la «qtref» par son modèle interopérable. Ce point sera détaillé dans la section «3.6.3) Outil d'interopérabilité : modification automatique d'une "netlist gate"». La mise en oeuvre de ce processus a nécessité le développement d'une bibliothèque de cellules spécifiques à l'interopérabilité. Ainsi, pour chaque bascule et verrous de la bibliothèque «qtref», un modèle interopérable a été développé. Chacun des modèles se voit ajouté des ports de scan. De même, une bibliothèque de mémoires RAM interopérable a été conçue.

3.4.3) Fichier de sauvegarde d'état

Maintenant que l'on sait comment accéder à l'état d'un circuit, il faut formaliser les fichiers de sauvegarde d'état associés. Ces fichiers dépendent de la façon dont l'ensemble des doublons ont été reliés afin de constituer les chaînes de scan. De plus, la représentation choisie doit permettre d'extraire facilement et rapidement l'état d'un sous-ensemble du circuit. Cette nécessité sera précisée dans la section «3.7.3) Optimisation : travail sur un sous-ensemble».

3.4.3.1) Structure d'arbre

La figure 19 montre la structure d'arbre proposée pour répondre à ce besoin. Les noeuds de l'arbre représentent les modules du circuit, les feuilles les variables d'état, c'est à dire des bascules, des verrous ou des mémoires RAM. Cet arbre est ordonné et est parcouru en profondeur. Pour un module donné, on trouve d'abord les feuilles du module avant ses noeuds. Les feuilles sont classées dans un premier temps par domaine d'horloge, c'est à dire que tous les éléments appartenant au même domaine d'horloge sont les uns à coté des autres. Dans la figure 19, considérons le module 2 qui a comme feuilles deux bascules (FF2 et FF3) et un verrou (L2). Si FF2 et L2 sont sur un premier domaine d'horloge et FF3 sur un second, on rencontre donc le groupe constitué de L2 et FF2 puis le groupe constitué de FF3.

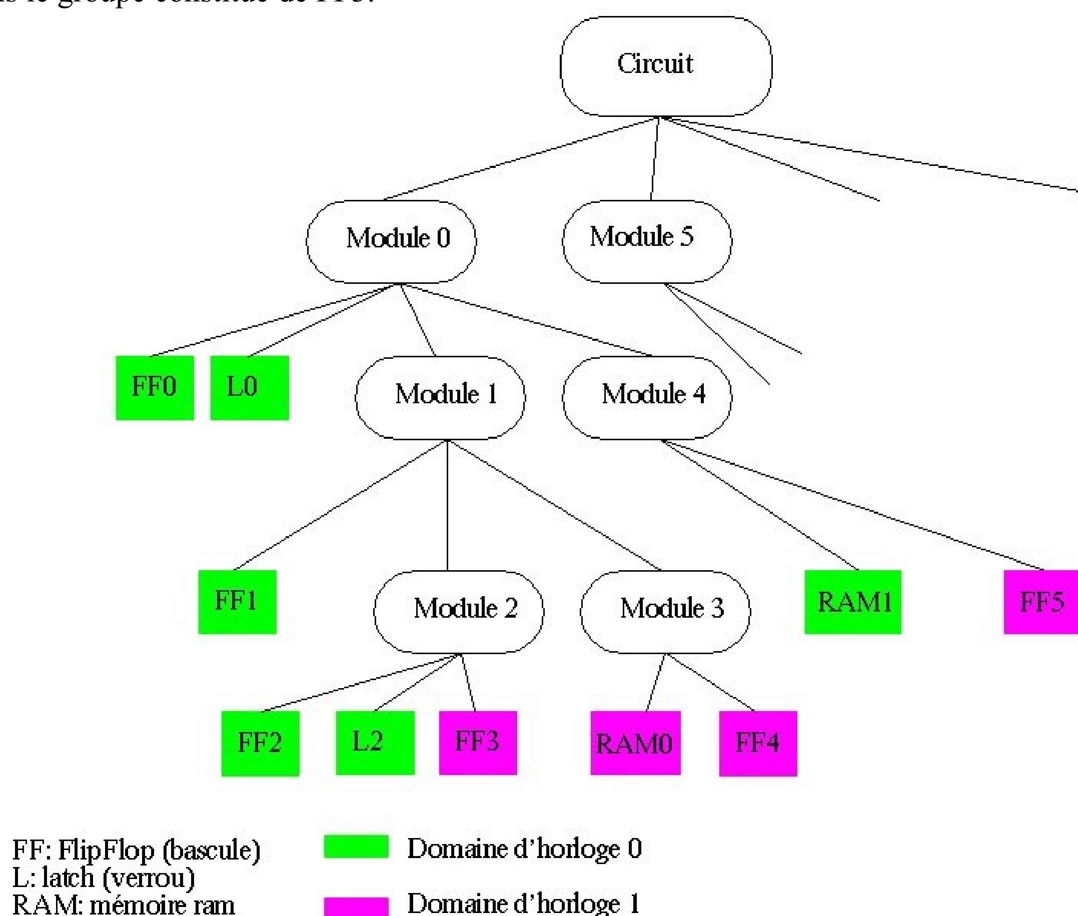


Figure 19 : Modélisation des variables d'état à l'aide d'un arbre

Dans un second temps, il faut ordonner les feuilles appartenant au même domaine d'horloge. Le critère retenu est la proximité spatiale. La synthèse initiale permettant la caractérisation des variables d'état présentée dans la section «3.3.3) Conclusion : la caractérisation de l'état d'un circuit ne peut se faire qu'après synthèse» fournit une première description du circuit au niveau portes logiques donnant la topologie des registres. On en déduit donc l'ordonnancement spatial.

Après avoir rencontré toutes les feuilles d'un module, on trouve les noeuds de ce module qui sont organisés eux aussi par proximité spatiale à partir du résultat de synthèse de référence.

Cette représentation d'arbre offre une grande souplesse pour l'extraction de l'état d'un sous-ensemble du circuit puisque pour cela, il suffit d'extraire un sous-arbre. La granularité est au niveau de l'ensemble des variable d'état d'un même domaine d'horloge d'un module.

3.4.3.2) Fichiers de description et fichiers de sauvegarde

En prévision d'une évolution de la technique de capture d'état, il a été décidé d'utiliser, pour accéder à l'état d'un circuit, deux chaînes de scan distinctes, une pour les bascules et verrous, l'autre pour les mémoires RAM. En effet, les FPGAs Xilinx offrent une capacité de capture dynamique de l'état des registres : en utilisant ceci, il n'est alors plus nécessaire d'instrumenter les bascules et verrous. Il n'y a alors plus qu'à s'occuper des mémoires RAM.

Par conséquent, le modèle d'arbre proposé dans la section précédente n'est pas tout à fait adapté car l'architecture de capture d'états va générer deux flux qu'il faudra réunir selon un ordre bien précis. Pour s'affranchir de ce problème, la solution consiste à utiliser deux fichiers de sauvegarde, donc deux arbres, un pour les bascules et verrous, l'autre pour les mémoires RAM.

D'autre part, la section «3.2.2.4) Restauration d'état» a montré la nécessité de recourir à des scripts d'initialisation pour restaurer l'état d'un circuit sur un simulateur ou une plateforme d'émulation relativement lente. Pour pouvoir les écrire, il faut connaître le nom de l'ensemble des variables d'état. La structure d'arbre précédemment présentée contient cette information.

Au final l'étape de caractérisation des variables d'état générera deux fichiers de description, un fichier des description des registres et verrous et un fichier de description des mémoires. Ces deux fichiers se présentent sous la forme de fichier texte où chaque ligne donne le nom, incluant la hierarchie d'accès, d'une feuille de l'arbre de modélisation. Si l'on reprend le circuit du précédent paragraphe, le fichier de description des bascules et verrous sera comme ceci :

- circuit.module_0.FF0
- circuit.module_0.L0
- circuit.module_0.module_1.FF1
- circuit.module_0.module_1.module_2.FF2
- circuit.module_0.module_1.module_2.L1
- circuit.module_0.module_1.module_2.FF3
- circuit.module_0.module_1.module_3.FF4
- circuit.module_0.module_4.FF5
- circuit.module_5.......

De plus, afin de réduire la taille de ce fichier, on peut envisager une optimisation lorsqu'on rencontre consécutivement plusieurs bascules appartenant au même vecteur. Au lieu d'avoir <chemin>.vecteur[0], <chemin>.vecteur[2], ... <chemin>.vecteur[n], on peut imaginer une seule ligne du type <chemin>.vecteur[0 :n].

Les deux fichiers seront nommés <nom_du circuit>.<ustamp>.<cstamp>.<reg ou mem>.st. Avec cette convention, on identifie le circuit correspondant aux fichiers de sauvegarde, les horloges et l'instant de capture sont caractérisés par la valeur des compteurs ustamp et cstamp présentés dans la section «3.4.2.2.2) Norme SceMi : présentation et extension du concept «cycle stamp»». Le type de fichier de sauvegarde est précisé par l'extension «reg.st» pour les registres et verrous et, «mem.st» pour les mémoires. Les fichiers de sauvegarde sont des fichiers binaires qui contiennent les valeurs des variables d'état ordonnées selon les arbres de description.

Enfin, lorsque le circuit a des dépendances externes, c'est à dire des signaux de stimulation non générés par la plateforme d'émulation/prototypage matériel, il faut également les enregistrer ce qui va engendrer un autre fichier de sauvegarde. Ce point sera précisé dans la section «3.4.5) Gestion des signaux d'entrée/sortie».

3.4.4) Transacteur lié à l'échange d'état

La section «3.4.2.2) Gestion des horloges» a montré l'intérêt de se placer dans un environnement SceMi. D'autre part, SceMi est une norme de communication entre émulateur/plateforme de prototypage et ordinateur supportée par l'ensemble des solutions

industrielles excepté les solutions de prototypage HAPS. SceMi est l'API de communication qui couvre le plus grand nombre de solutions, on a donc choisi de bâtir l'architecture de sauvegarde/restauration d'état à partir de cette norme. Ainsi, la solution sera portable sur la quasi totalité des solutions industrielles en matière d'émulation/prototypage.

Le système permettant l'accès à l'état du circuit se décompose en une partie logicielle et une partie matérielle (figure 20). Selon SceMi, la partie matérielle se nomme transacteur. L'API de communication entre les deux environnements est l'API SceMi. Elle permet de relier un environnement matériel avec un programme écrit en C++.

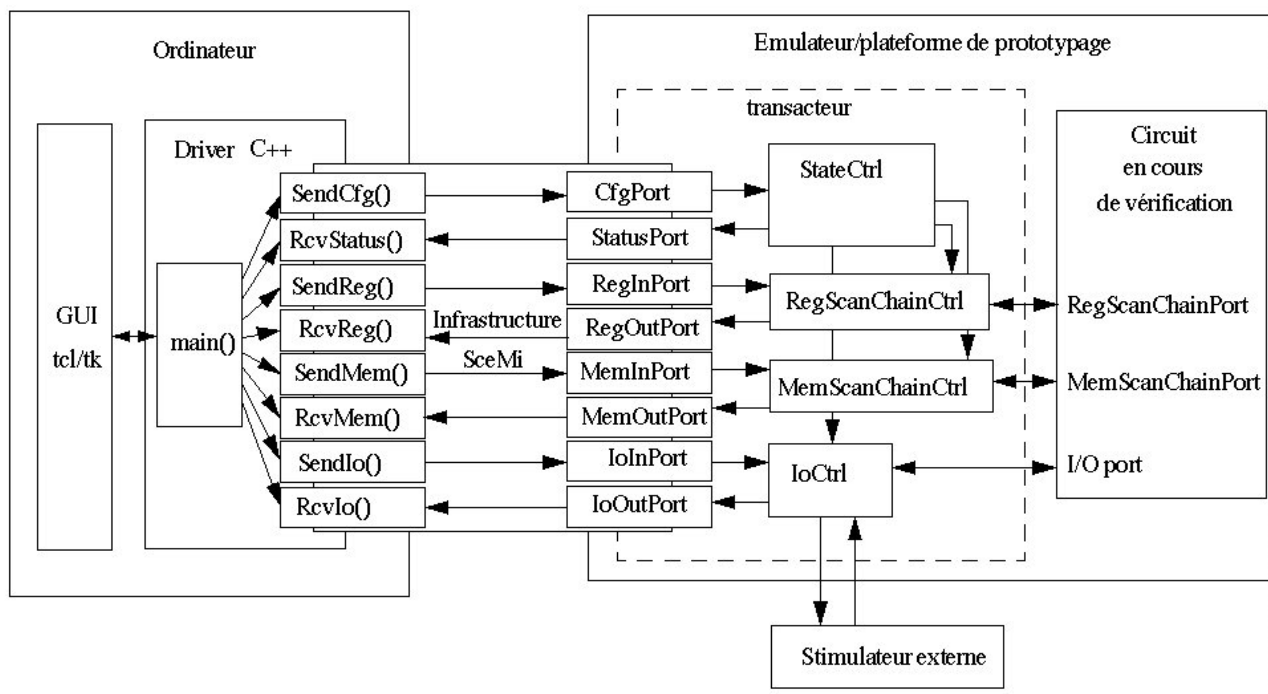


Figure 20 : Transacteur permettant l'accès à l'état d'un circuit

Pour lancer le système, il faut exécuter un programme en C++. Ce programme (main) va alors appeler d'une part un environnement graphique (GUI) et d'autre part un pilote (driver) qui gère les communications avec le matériel. Le GUI, écrit en tcl/tk, permet à l'utilisateur de configurer le système, de sélectionner les instants d'échantillonnage de l'état du circuit. Le driver pilote l'environnement matériel à l'aide de huit fonctions associées à des canaux de communication SceMi.

Le choix d'avoir un environnement utilisateur en tcl/tk a été motivé par le fait que l'ensemble des solutions industrielles offre la possibilité d'utiliser des scripts écrits dans ce langage. L'outil proposé reste ainsi dans la philosophie des outils existants, il sera facile à mettre en place vu que les utilisateurs sont familiers avec ce langage. L'environnement utilisateur permet de programmer le matériel, indique la fréquence de capture des états, propose toute une panoplie de triggers (figure 21). L'interface gère également les restaurations d'état.

Les instructions utilisateurs sont transmises par un canal de configuration à un contrôleur matériel «StateCtrl». Ce module envoie périodiquement (dix fois par seconde) un statut des opérations d'accès à l'état du circuit ainsi que l'état d'avancement du test. Lorsque l'utilisateur demande une capture ou une restauration d'état, ou si un instant de capture programmé est atteint, ce contrôleur va alors activer deux autres modules. Le premier est nommé «RegScanChainCtrl» et gère l'accès aux bascules et verrous du circuit. Il possède deux canaux spécifiques pour d'une part communiquer avec l'environnement logiciel et d'autre part accéder aux bases de données. Le deuxième module est nommé «MemScanChainCtrl» et gère, de la même manière, l'accès aux

mémoires du circuit et la communication avec les bases de données correspondantes.

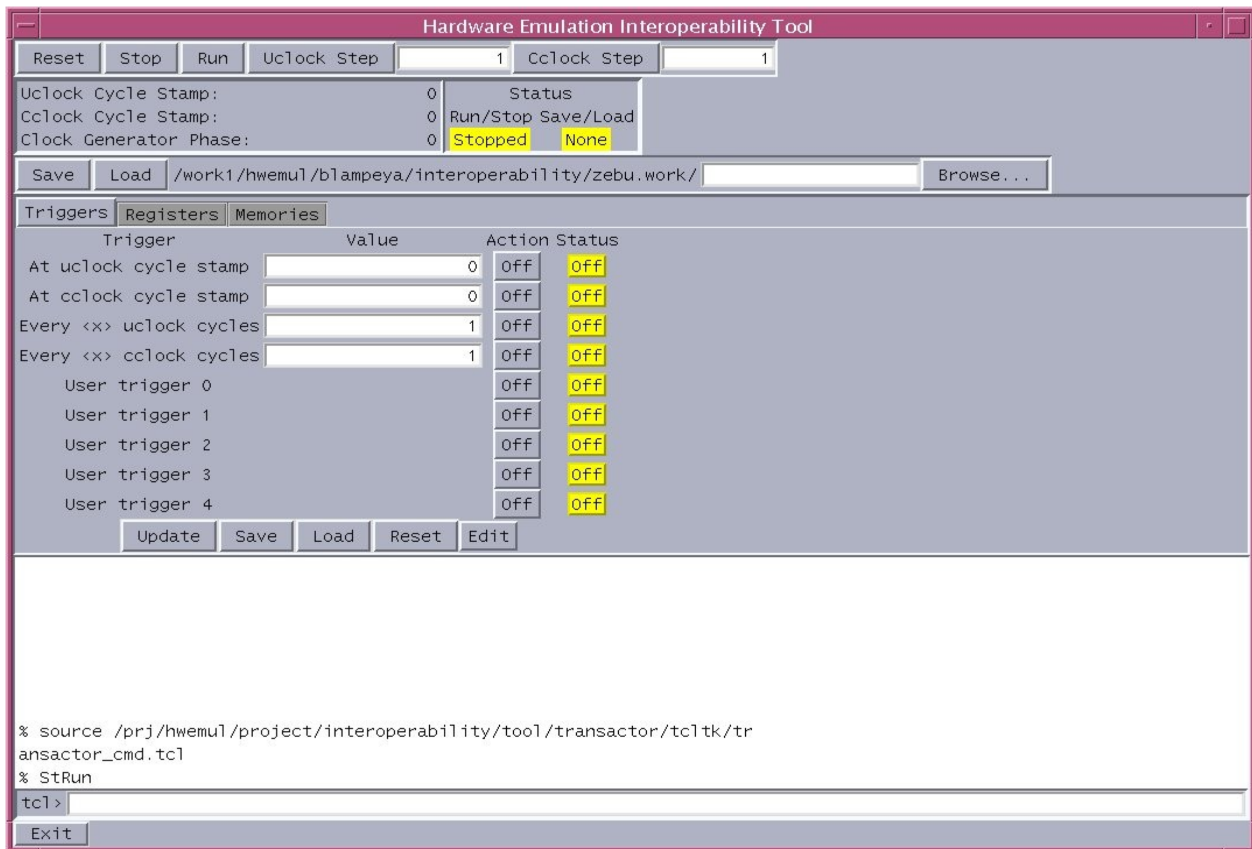


Figure 21 : Environnement graphique de pilotage des captures/restaurations d'état

Enfin, un dernier module nommé «IoCtrl» échantillonne les données de stimulation en provenance de l'extérieur de l'émulateur. Lorsqu'on reproduit un test, ce module s'occupe de fournir les données de stimulation au circuit. Il sera détaillé dans le prochain paragraphe.

3.4.5) Gestion des signaux d'entrée/sortie

Les précédents paragraphes ont montré la nécessité d'échantillonner les signaux de stimulation du circuit non générés par la plateforme d'émulation/prototypage. Par exemple, si l'on veut reproduire les tests d'un circuit de traitement vidéo stimulé par une caméra, il faut enregistrer le flux envoyé par cette caméra.

3.4.5.1) Problèmes de bande passante et de volume de stockage

Un tel échantillonnage peut générer un grand volume de données et, il se peut que la bande passante du lien de communication émulateur/ordinateur ne soit pas suffisante.

D'autre part, même si la bande passante permet l'échantillonnage, le volume de données accumulé sur plusieurs heures peut engendrer des problèmes de stockage. Par exemple, imaginons un circuit émulé à 1MHz et utilisant quatre bits pour communiquer avec un environnement logiciel. Il faudra une bande passante de 4Mbits/s pour échantillonner la communication, ce qui n'est absolument pas un problème, les APIs SceMi actuelles proposant des bandes passantes bien supérieures. Si l'on considère un test de 48h, cet échantillonnage engendrera un volume de données de 84,37Go, ce qui n'est pas rien à stocker.

Il est donc souhaitable de réduire le volume de données. On peut s'affranchir en partie du problème de stockage si le banc de test est capable de détecter une erreur. Dans ce cas, lorsque l'on réalise une nouvelle capture d'état et qu'il n'y a pas eu d'erreur détectée jusque là, on peut effacer les échantillons stockés. Seule l'évolution des signaux à partir du point de capture précédent l'apparition d'une erreur est importante. Cependant, une telle hypothèse ne peut pas se généraliser. On aura donc intérêt à intégrer dans le module d'échantillonnage un système de compression temps réel.

3.4.5.2) Compression temps réel

Les systèmes de compression sont d'autant plus efficaces qu'ils implémentent des algorithmes adaptés aux données rencontrées. Il faut donc caractériser les données que l'on va devoir échantillonner afin de développer le compresseur optimisé.

D'autre part, chez STMicroelectronics, la majorité des plateformes de vérification, ayant des dépendances externes, utilisent également un bus pour communiquer entre le circuit en cours de validation et un environnement logiciel utilisant une carte d'application spécifique. Ce type de plateforme sert à valider des sous-circuits (IPs) dans leur environnement logiciel final. Dans ce cas, le processeur de la carte d'application lance des tâches au circuit et s'occupe de récupérer les résultats. Il se passe alors de nombreux cycles sans aucune communication entre les deux environnements, d'où un possible gain sur l'échantillonnage de ces signaux.

Les prochains paragraphes vont proposer une architecture générique capable de compresser en temps réel des données échangées sur bus. Bien sur, pour avoir un maximum d'efficacité, il faudrait développer un compresseur adapté à chaque type de bus. De plus, un compresseur adapté peut nécessiter des calculs complexes et donc engendrer une architecture tout aussi complexe et longue à développer. L'architecture qui va être proposée se veut simple, de mise en oeuvre rapide, et offrir des performances correctes bien que non optimales, mais avant tout utilisable quel que soit le bus considéré. De plus la compression proposée est sans perte, elle permet de reconstruire l'intégralité des signaux observés avec une précision au niveau cycle d'horloge.

3.4.5.2.1) Limitation en surface

Instrumenter le circuit afin de réduire le volume de données échantillonnées est intéressant mais, cette instrumentation ne doit pas occuper énormément de ressource. Vu le prix des solutions industrielles d'émulation/prototypage matériel, il ne faudrait pas avoir à surdimensionner les machines afin que celles-ci puissent intégrer le compresseur. La taille des circuits actuels est de l'ordre de plusieurs millions de portes logiques, il serait souhaitable que la taille du compresseur reste insignifiante par rapport à la taille des circuits, c'est à dire que la taille du compresseur soit inférieure ou égale à 1% de la taille de ces circuits, soit moins de dix mille portes logiques.

3.4.5.2.2) Segmentation du bus en ensembles

L'algorithme de compression proposé fait l'hypothèse que la bande passante du lien de communication échantillonné n'est pas saturée et qu'il n'y a pas de communication à tous les cycles d'horloge. C'est grâce à cette non saturation que l'on va pouvoir compresser.

La compression repose en outre sur une segmentation du bus en plusieurs ensembles. Si on regarde l'organisation d'un bus, celui est souvent composé de signaux de données, de signaux d'adresses et de signaux de contrôle. Durant une transmission de type «burst», tous ces signaux n'évoluent pas en même temps. En général, les signaux de contrôle évoluent en premier pour initier la communication, puis les signaux d'adresses sont mis à jour. Commence alors la transmission des données qui se termine par un acquittement sur les signaux de contrôle.

L'algorithme proposé fonctionne de la façon suivante : à chaque fois qu'un groupe de

signaux évolue, on transmet la nouvelle valeur de ce groupe ainsi que le nombre de cycles d'horloge écoulés depuis la dernière évolution (figure 22).

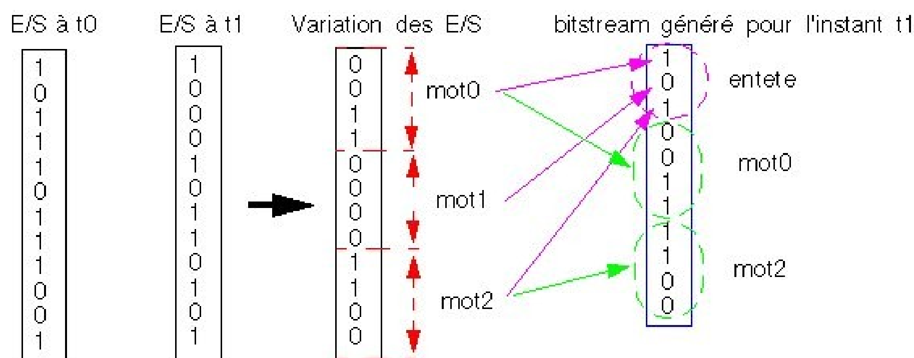


Figure 22 : Génération d'un mot compressé

Dans l'exemple de la figure 22, le bus échantillonné est segmenté en trois ensembles de taille identique, à savoir 4bits. On le compare chaque nouvelle échantillon avec le précédent, on obtient ainsi la variation entre deux cycles. Chaque groupe ayant évolué doit transmettre sa nouvelle valeur. Dans l'exemple, le premier et le dernier groupe ont évolué mais «mot1» n'a pas changé. Il faut donc transmettre juste la nouvelle valeur de «mot0» et de «mot2».

Un mot compressé contient d'abord un en-tête indiquant les groupes qui vont être mis à jour. Dans l'exemple, cet en-tête contient un «1» pour les groupes «mot0» et «mot2» qui sont concernés par la mise à jour et un «0» pour le groupe «mot1» qui n'évolue pas. Après cette entête, on trouve les nouvelles valeurs des groupes mis à jour.

Si, entre deux cycles d'horloges, les signaux observés n'évoluent pas, on ne transmet aucun message. L'en-tête présenté dans l'exemple n'est donc pas complêt, il lui manque une information temporelle, à savoir le nombre de cycles d'horloge écoulé depuis la dernière évolution. La figure 23 montre les mots compressés transmis durant une opération effectuée sur un bus que l'on échantillonne. Le bus observé est utilisé pour transmettre un bloc de quatre données. Il est segmenté en deux groupes, un groupe contenant les signaux de contrôle (Groupe Ctrl) et un groupe contenant les signaux de données (Groupe Data).

A un instant donné, l'initiateur de la transaction cherche à établir la communication en activant le signal «Dreq». Cela se traduit, au niveau de l'échantillonnage, par la transmission du message suivant : <x> cycles d'horloges depuis le dernier message, le groupe Ctrl évolue, le groupe Data reste stable (entête : 10), la nouvelle valeur du groupe Ctrl est «100». Deux cycles d'horloge plus tard, la cible de la transaction répond et transmet une première donnée (signal Req au niveau haut), les bus de données et d'adresses sont mis à jour. Cela se traduit par un nouveau message : deux cycles d'horloges depuis le dernier message, les deux groupes évoluent (entête : 11), la nouvelle valeur du groupe Ctrl est «110», la nouvelle valeur du groupe Data est «D1 A1».

Ainsi à chaque évolution d'un des deux ensembles, un nouveau message est émis, ce message contient en premier le nombre de cycles réalisé depuis l'envoi du dernier message, suivi d'une entête donnant les groupes concernés par les modifications, suivi des valeurs des groupes qui ont évolués.

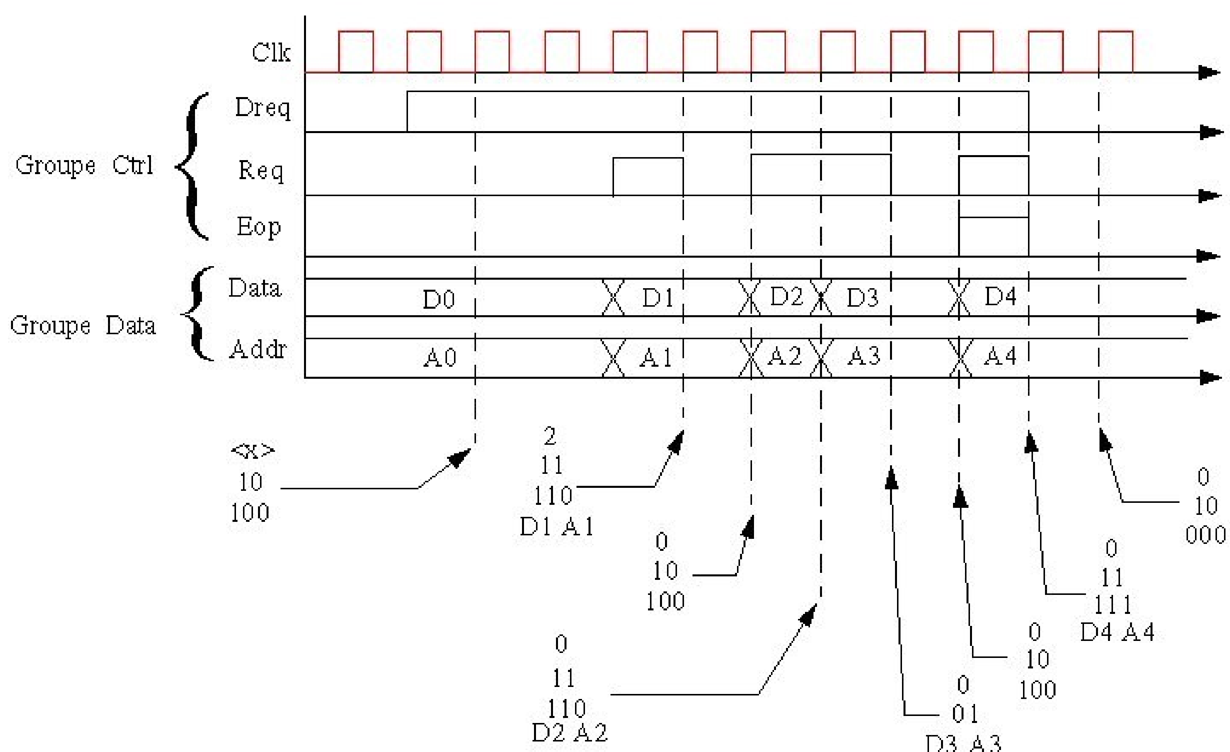


Figure 23 : Exemple de segmentation d'un bus et transmissions de messages associés

Vis à vis des besoins de reproductibilité des scénarios de test, seul l'échantillonnage des signaux d'entrée du circuit est nécessaire. Cependant, s'il n'y a pas de contrainte de bande passante, il peut être judicieux d'échantillonner également les signaux de sortie. Ainsi, lorsque l'on reproduira un scénario, on pourra vérifier que les mêmes réponses sont fournies. Si ce n'est pas le cas, on détecterait alors un problème lié aux plateformes d'émulation/prototypage utilisées et non pas un problème de circuit.

3.4.5.2.3) Codage de Golomb

Le nombre de cycles d'horloge séparant deux transmissions de valeur d'un groupe de signaux nécessite un certain nombre de bits. On ne peut pas utiliser un codage à taille fixe pour transmettre cette information. En effet, si on utilise un codage à taille fixe et vu qu'il peut y avoir de très longue plage de non communication, il faudra facilement prévoir 16, 32 bits ou plus pour coder cette information. Ainsi, si une séquence de test fait des transmissions tous les dix cycles, le coût de ce compteur risquerait de consommer tout le bénéfice de la compression.

Il est donc important d'utiliser un codage à taille variable. Dans la littérature, on trouve le codage de Golomb [CHA01] [CHA03] qui propose des propriétés intéressantes (tableau 5). Chaque mot du code est constitué d'un préfix de «N» bits et d'un corps également de taille «N» bits. Le préfixe est composé d'une suite de «1» suivi d'un «0». Le corps est quant à lui quelconque. La fonction donnant la valeur «V» d'un mot de code dont la longueur du préfixe est «N» et le corps a pour valeur «C» est la suivante : $V=2^N-2+C$.

Groupe	*Longueur de la séquence*	*Préfixe*	*Corps*	*Mot de code*
A1	0	0	0	00
	1		1	01
A2	2	10	00	1000
	3		01	1001
	4		10	1010
	5		11	1011
A3	6	110	000	110000
	7		001	110001
	8		010	110010
	9		011	110011
	10		100	110100
	11		101	110101
	12		110	110110
	13		111	110111
...	...	...	...	...

Tableau 5: Exemple de code de Golomb

Le codage retenu est donc un code à longueur logarithmique (figure 24). La taille en bits d'un mot codé est donnée par l'équation suivante : $N=E[ln2(V+2)]$. La taille «N» d'un nombre codé est donnée par la partie entière du logarithme en base deux de la valeur du nombre incrémentée de deux.

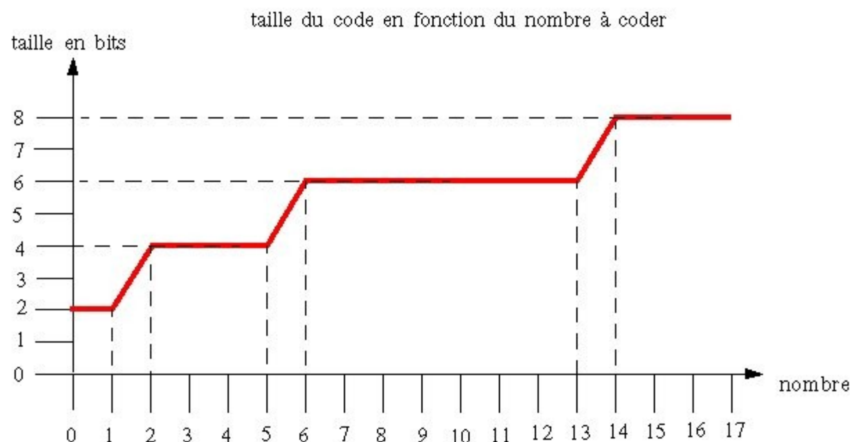


Figure 24 : Taille du code en fonction du nombre à coder

Un compteur de golomb a été réalisé dans le cadre de cette thèse afin de compter le nombre de cycles d'horloge entre deux transmissions de données, la valeur du compteur utilise le code présenté. Un tel compteur avec valeur sur au plus trente-deux bits nécessite deux cents soixante-quinze portes logiques dont soixante-neuf bascules. Cette taille est négligeable et donc tout à fait acceptable.

3.4.5.2.4) Gestion des débordements

Malgré une compression temps réel, il peut arriver que la bande passante du lien de communication ne soit pas suffisante pour transmettre l'intégralité des données échantillonnées. Dans ce cas, il faut gérer les débordements de capacité. Plusieurs scénarios ont été identifiés.

Le premier cas est celui où la plateforme d'émulation/prototypage matériel a des dépendances externes. Dans ce cas, il est impossible de ralentir le système, il faudra abandonner des données. La reproductibilité du test ne sera donc pas totale, on ne pourra rejouer qu'une partie du scénario, partie liée aux données capturées. Le système d'acquisition doit donc pouvoir indiquer les débordements.

Dans les autres cas où l'on peut ralentir la plateforme de vérification, il faudra que l'utilisateur choisisse entre vitesse et reproductibilité. Si l'on définit la reproductibilité comme priorité, le système d'acquisition devra pourvoir contrôler les horloges du circuit. SceMi offre cette capacité via le mécanisme «uclock - cclock» précédemment présenté. Dans l'architecture proposée dans la section «3.4.4) Transacteur lié à l'échange d'état», le module de gestion des signaux d'entrée/sortie communique avec le contrôleur d'état. Ce contrôleur pilote les macros SceMi et, est capable de suspendre les horloges du circuit si besoin est.

D'autre part, avec l'algorithme proposé, pour connaître la valeur des signaux échantillonnés à un instant particulier, il faut lire le fichier de sauvegarde depuis son début et calculer, à partir des valeurs des Golomb, le cycle d'émulation. Si l'on veut restaurer un état à partir d'une capture réalisée après plusieurs heures de fonctionnement, amener les signaux d'entrée à la bonne valeur pourra nécessiter un certain temps. Afin de réduire ce temps d'initialisation, il est utile de fragmenter les données échantillonnées en plusieurs fichiers. Pour chaque fichier, on connaîtra le cycle d'acquisition du premier échantillon du fichier. La fragmentation minimale sera au niveau de la capture d'état, c'est à dire qu'à chaque nouvelle capture d'état on changera de fichier de sauvegarde pour les signaux d'entrée échantillonnés.

3.4.5.2.5) Fichier de sauvegarde de l'évolution des signaux d'entrée/sortie

Les trois sections précédentes ont expliqué le codage proposé pour l'échantillonnage avec compression des signaux d'entrée du circuit. Cet échantillonnage se traduit par la transmission de messages dont voici l'organisation :
1 – ustamp – groupe 0 – groupe 1 – .... - groupe «n» ou 0 – delta_cycle – en-tête – groupes modifiés
Le premier bit d'un message indique si le message est un point de départ pour un nouveau fichier de sauvegarde (bit au niveau haut) ou si le message correspond à une évolution des signaux à ajouter au fichier de sauvegarde actuel (bit au niveau bas).

Dans le cas de la transmission d'un message de démarrage d'un fichier de sauvegarde, le message émis contiendra la valeur du «ustamp», codée sur 64 bits, suivi de la valeur de tous les signaux échantillonnés à l'instant «ustamp».

Dans le cas d'un message d'évolution des signaux à ajouter au fichier de sauvegarde en cours d'écriture, le message contiendra d'abord le nombre de cycles d'horloge effectué depuis la dernière transmission (delta_cycle). Ce nombre est codé avec le code à longueur variable de Golomb. Ensuite, on trouve un en-tête indiquant les groupes concernés par la transmission. Si le bit «i» de l'en-tête est au niveau haut, alors la nouvelle valeur du groupe «i» sera transmise après l'en-tête.

3.4.5.2.6) Implémentation et performances

Les performances du système proposé dépendent essentiellement de la qualité du groupement de signaux ainsi que du taux d'utilisation de la bande passante du canal échantillonné. Si la bande passante est quasi saturée et que l'on échantillonne aussi bien les signaux d'entrée que

ceux de sortie, les messages compressés peuvent être plus volumineux que ceux échantillonnésà cause des bits d'entête rajoutés, le système ne sera donc pas intéressant dans un tel cas.

Considèrons un exemple avec un bus nécessitant 10bits de contrôle, 32bits de données et 8bits d'adresse. On groupe ensemble les signaux de contrôle d'une part, les signaux d'adresse et de données d'autre part. En moyenne, la transmission d'une transaction sur ce bus se traduit par l'envoi de quatre messages de capture, trois concernant le groupe de contrôle et un concernant les deux groupes de signaux. Avec cette hypothèse, en considérant qu'une transmission se fait en environ six cycles d'horloges, la taille «T» des messages transmis est donnée par l'équation suivante : $T=E[ln2(C+2)]+98$, «C» étant le nombre de cycles d'horloges entre deux transmissions. Le tableau 6 donne le taux de compression du système en fonction du taux d'occupation de la bande passante du lien de communication échantillonné. On remarque l'efficacité du système proposé puisque dans les conditions de cet exemple, on obtient un taux minimum de compression de 67% lorsque le lien de communication est saturé (C=0).

Taux d'occupation du bus	*Taux de compression*
100%	**67%**
86%	**72%**
75%	**75%**
60%	**80%**
50%	**83%**
40%	**87%**
30%	**90%**
20%	**93%**
10%	**97%**

Tableau 6 : Taux de compression en fonction du taux d'utilisation du bus

Une architecture générique implémentant cette stratégie de compression a été développée dans le cadre de cette thèse. Cette architecture se décompose en deux sous-ensembles (figure 25).

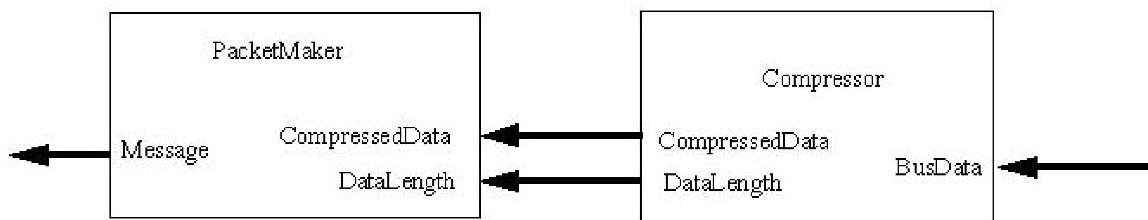


Figure 25 : Architecture du système de compression temps réel

Le premier sous-ensemble est le codeur à proprement parler. Il échantillonne le bus à chaque cycle d'horloge, compare le nouvel échantillon avec le précédent et génère le mot compressé correspondant, ainsi que la longueur de ce mot codé. Le deuxième sous-ensemble va récupérer les mots compressés ainsi que leur taille. Les transferts de données entre émulateurs et ordinateurs se font par mots de taille fixe. Ainsi, le module «PacketMaker» va former des packets de taille fixe pour les transmettre, via SceMi, à un ordinateur pour stockage sur disque dur.

Pour un bus utilisant 32bits de données, 32 bits d'adresse et 10bits de contrôle, bus segmenté en deux ensembles, le système de compression nécessite quatre mille portes logiques dont sept cents bascules. Cette taille est tout à fait acceptable vis à vis des contraintes de limitation en surface.

3.4.6) Format de sauvegarde d'état

En résumé, une sauvegarde d'état engendrera cinq fichiers différents précédemment présentés. Tout d'abord, durant l'instrumentation du circuit, un fichier de caractérisation des variables d'état de type bascule et verrou et un autre fichier de caractérisation des variables d'état de type mémoire RAM seront créés. Ces fichiers seront nécessaires pour recourir à des scripts d'initialisation/capture d'état sur des environnements lents comme des simulateurs.

De plus, chaque capture d'état engendrera un fichier de sauvegarde pour les bascules et verrous, un fichier de sauvegarde pour l'état des mémoires RAM et, si besoin est, un fichier d'enregistrement des évolutions des signaux d'entrée du circuit entre deux instants de capture d'état.

3.5) Concept d'interopérabilité

Maintenant que l'on est capable de changer de plateforme de vérification, ce qui permet une plus grande coopération entre les différentes techniques basées sur des solutions d'émulation/prototypage matériel, on peut définir un nouveau concept : le concept d'interopérabilité en émulation et prototypage matériel.

Dans la suite de cette thèse, on considérera qu'une plateforme de vérification «A» (simulation, accélération, émulation ou prototypage) est intéropérable avec une autre plateforme «B» si, à tout instant de la vérification, on est capable d'une part, de capturer l'état d'une partie du circuit émulé sur «A» ainsi que l'évolution des signaux d'entrée de la partie considérée et d'autre part, que l'on est capable de reproduire au cycle près le scénario de test de la partie considérée sur la plateforme «B» et ce à partir de n'importe quel instant de sauvegarde.

Pour bien assimiler ce nouveau concept, considérons l'exemple suivant. Un circuit utilisant un million de portes logiques est prototypé avec une machine ZeBu-ZV, en mode STB. Cela signifie que le banc de test ainsi que le circuit en cours de vérification sont implémentés dans la plateforme de prototypage ZeBu-ZV. La plateforme fonctionne à 7MHz. Les caractéristiques du ZeBu-ZV font que les possibilités de déboguage sont limitées. On lance alors un scénario de test d'une durée de dix heures. Toutes les dix minutes, on réalise une capture d'état du circuit ainsi que de son banc de test. A la fin du test, après analyse des résultats, on s'aperçoit qu'une erreur matérielle s'est produite au bout de huit heures et trente-sept minutes de test. Il faut alors observer ce qui se passe dans le circuit à cet instant afin d'identifier le bogue. On décide d'utiliser un émulateur Palladium, machine adaptée au déboguage, pour caractériser le problème. Pour cela, on va implémenter sur Palladium à la fois le banc de test du circuit et le circuit et, on va restaurer l'état à partir de la capture réalisée au bout de huit heures et trente minutes de test sur ZeBu-ZV. La plateforme Palladium fonctionne à 700kHz, il faut donc soixante-dix minutes sur Palladium pour reproduire l'erreur avec une observabilité totale. Dans ce cas, on dira que la machine ZeBu-ZV est intéropérable avec la machine Palladium.

Le concept d'interopérabilité offre de nouvelles possibilités quant aux stratégies de déboguage. Les prochains paragraphes vont dans un premier temps proposé un flot d'instrumentation semi-automatisé permettant d'obtenir l'interopérabilité entre toutes les machines supportant la norme SceMi. Dans un second temps, une nouvelle stratégie de vérification, exploitant les richesses promises par ce concept d'interopérabilité sera présentée.

3.6) Proposition d'un flot de prototypage semi-automatisé orienté interopérabilité

Les précédents paragraphes ont introduit le concept d'interopérabilité en émulation matériel ainsi qu'une solution architecturale permettant l'obtention de cette interopérabilité sur toute solution d'émulation/prototypage matériel supportant la norme de co-émulation SceMi.

Afin d'utiliser cette solution générique, il faut modifier l'architecture des circuits à vérifier c'est à dire les instrumenter de manière à ce qu'ils supportent l'architecture d'interopérabilité. Cette opération nécessite la modification de plusieurs milliers de bascules et verrous ainsi que l'ajout de plusieurs dizaines de contrôleurs mémoires. Il n'est donc pas envisageable de réaliser manuellement cette instrumentation, il est faut l'automatiser.

Les prochains paragraphes vont donc présenter un flot semi-automatisé permettant la mise en oeuvre de plateformes de vérification toutes interopérables les unes avec les autres.

3.6.1) Flot d'obtention de la netlist de référence

La section «3.3.3) Conclusion : la caractérisation de l'état d'un circuit ne peut se faire qu'après synthèse» a montré la nécessité de caractériser les variables d'état à partir d'un résultat de synthèse indépendant de la plateforme utilisée et de ses performances fréquentielles. Ce sera donc le point de départ de l'ajout de l'architecture d'interopérabilité.

La première étape du flot de mise en oeuvre d'un ensemble de plateformes d'émulation/prototypage interopérables consiste donc à réaliser cette synthèse de référence. Plusieurs outils peuvent convenir pour cette étape. On peut réaliser une synthèse ASIC avec un outil comme dc_shell de Synopsys. Cependant, bien que répondant aux besoins, la synthèse ASIC reste relativement longue, de l'ordre d'une heure de synthèse par million de portes logiques. Les synthèses pour émulateur sont beaucoup plus rapides. Par exemple, avec l'outil HdlIce, outil de synthèse spécifique à l'émulateur Palladium, il faut compter environ dix minutes pour synthétiser un million de portes logiques. De plus, les synthèses pour émulateur n'introduisent pas de contrainte de plateforme, on peut donc utiliser ces outils pour réaliser la synthèse de référence. Pour les cas pratiques de cette thèse, on utilisera l'outil de synthèse HdlIce pour la synthèse de référence.

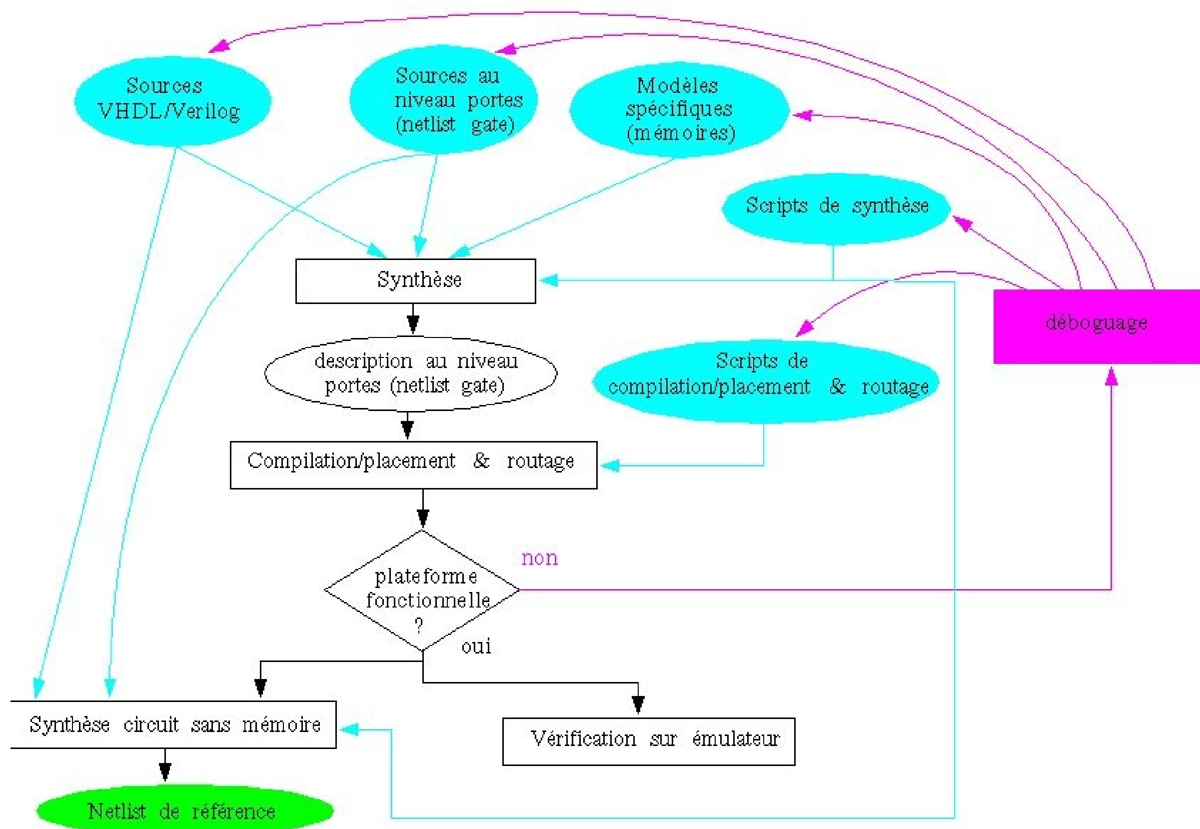


Figure 26 : Flot d'obtention de la netlist de référence

De plus, la mise en oeuvre d'une plateforme d'émulation n'est pas toujours aisée. Souvent, il faut plusieurs jours, voire semaines, de travail avant qu'une plateforme fonctionne. Cette durée de mise en oeuvre est due à diverses raisons. Notamment, les composants sont décrits pour être synthétisés en ASIC et non pour être émulés. Ainsi, certains composants doivent être modifiés avant de pouvoir passer en émulation, particulièrement les mémoires. Au final, il faut souvent plusieurs itérations avant d'obtenir une plateforme fonctionnelle. D'ailleurs, dans [BIG04], on apprend que chez STMicroelectronics, tout circuit passe obligatoirement en émulation avant d'aller en prototypage. L'étape d'émulation garantit que le circuit est synthétisable avant d'aller en prototypage où l'identification des bogues est très hardue.

En conclusion, afin d'être absolument certain de la fiabilité du résultat, il faudra attendre que le circuit fonctionne en émulation avant de valider la synthèse de référence. De plus, cela permettra de démarrer la vérification en émulation avant la fin du flot d'émulation/prototypage orienté interopérabilité (figure 26). Dans le flot proposé, la netlist de référence est obtenue après synthèse des fichiers de description du circuit en excluant les modèles mémoires. La gestion des mémoires en émulation/prototypage est toujours un point délicat. Chaque machine possède des primitives mémoire spécifiques qui demandent une gestion adaptée. Les modèles mémoires ne sont souvent pas compatibles entre deux solutions d'émulation/prototypage. En particulier, les outils de synthèse dédiés à l'émulation ne comprennent pas les descriptions des mémoires ASIC. Par conséquent, afin de s'affranchir de tout problème lié aux mémoires, la synthèse de référence n'intègre pas les mémoires. Celles-ci sont vues comme des boîtes noires.

3.6.2) Flot de prototypage

Le précédent paragraphe a permis l'obtention d'une netlist de référence, c'est à dire une description du circuit comme un assemblage de portes logiques. Ce processus s'intègre dans un flot de prototypage spécifique à l'interopérabilité (figure 27).

La netlist de référence sert de point d'entrée à un outil d'interopérabilité qui sera détaillé dans la prochaine section. Ce fichier n'intègre pas les mémoires du circuit. Cependant, les mémoires ne peuvent pas être complétement ignorées vu qu'elles font partie des variables d'état. Un fichier de description des mémoires est donc à fournir à l'outil d'interopérabilité. La composition de ce fichier sera également précisé au prochain paragraphe.

L'outil d'interopérabilité va automatiquement instrumenter le circuit, ajouter l'architecture de capture et restauration d'état. Le résultat de cet outil est d'une part, un nouveau fichier de description au niveau porte logique (netlist gate) du circuit instrumenté et, d'autre part, des scripts de simulation. Ces scripts permettront d'initialiser un état sans utiliser l'architecture d'accès aux variables d'état vu que cette architecture est trop lente en simulation, comme vu dans la section «3.4.2.4) Restauration d'état».

La netlist instrumentée du circuit va ensuite servir de point d'entrée au flot classique d'émulation/prototypage matériel. Si la bibliothèque de primitives de la plateforme cible diffère de celle utilisée pour la synthèse de référence, il faudra dans un premier temps synthétiser la bibliothèque de référence avec les primitives de la plateforme cible. Ensuite, on enchaîne les étapes classiques de synthèse, à savoir partitionnement, placement et routage ou compilation. On obtient ainsi plusieurs plateformes de vérification. Celles utilisant des plateformes de prototypages sont les plus rapides donc dédiées à l'exécution des scénarios de test. Les plateformes d'émulation offrent à la fois une bonne vitesse d'exécution et de grande capacité de déboguage, ces plateformes peuvent donc servir à l'exécution des scénarios de test et au déboguage du circuit. La simulation, de part ses propriétés, est réservée au déboguage.

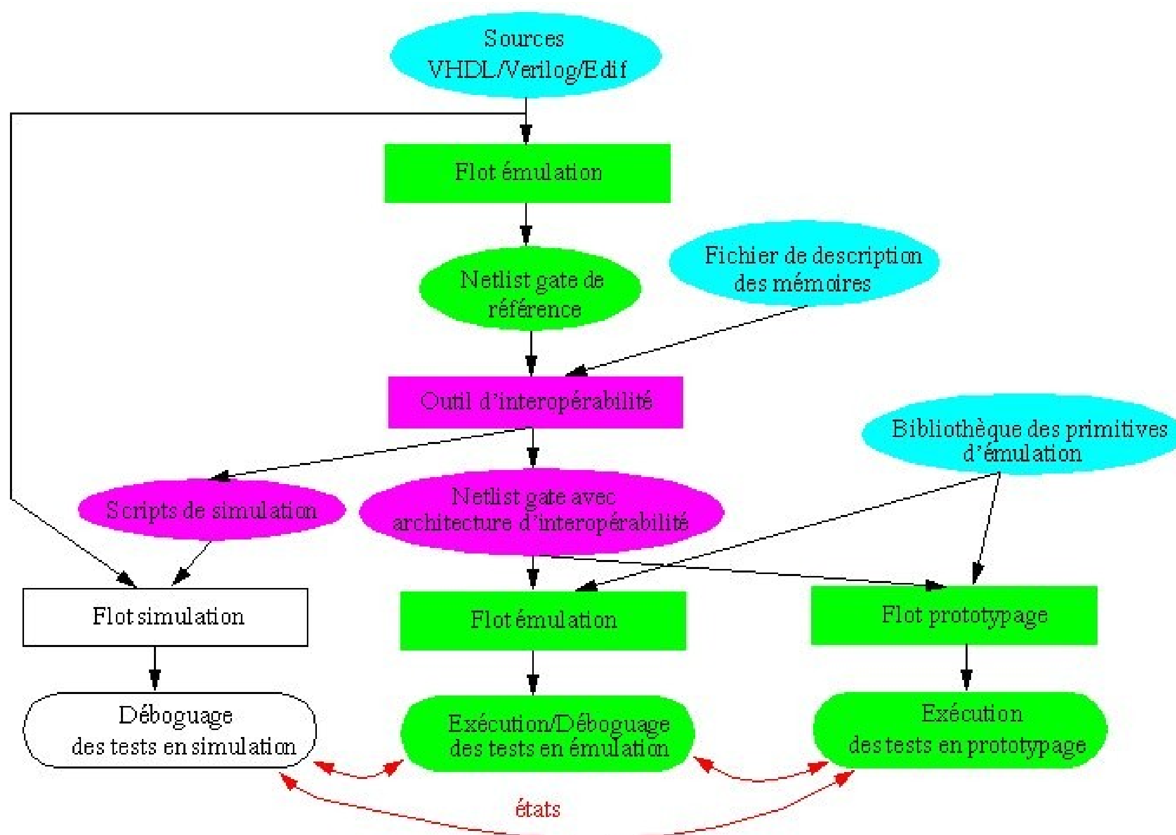


Figure 27 : Flot de mise en oeuvre de plateformes de vérification interopérables

Enfin, le point d'entrée des différents flots d'émulation/prototypage est une description fiable vu les contraintes imposées dans le précédent paragraphe. Par conséquent, la mise en oeuvre des simulations et émulations avec capacité d'interopérabilité est très rapide, de l'ordre de quelques heures. Pour le prototypage, les contraintes de partitionnement, placement et routage font que le délai de mise en oeuvre restera assez élevé, de l'ordre de plusieurs jours voire semaines.

3.6.3) Outil d'interopérabilité : modification automatique d'une "netlist gate"

L'outil d'interopérabilité, nommé «InteroperabilityCompile», a pour mission d'instrumenter le circuit à vérifier en lui ajoutant une architecture d'accès, en lecture et écriture, aux variables d'état du circuit. Son action se déroule en quatre étapes (figure 28).

Dans un premier temps, l'outil lit la netlist de référence. Pour chacune des primitives utilisées dans ce fichier, il compare la primitive lue avec celles référencées dans sa bibliothèque. Si elle s'y trouve, c'est que cette primitive est une bascule ou un verrou, il la remplace par son modèle orienté interopérabilité.

De même, lorsque l'outil rencontrera une boîte noire, il lira le fichier de description des mémoires qui liste les noms des primitives mémoire instanciées et donne, pour chacune sa taille en bits ainsi que son nombre de ports de lecture et écriture. L'écriture de ce fichier est la tâche ajoutée à l'utilisateur par rapport au flot classique. Etant donné que, quelle que soit la machine considérée, la gestion des mémoires exige un traitement manuel, l'écriture de ce fichier devrait se faire très facilement. L'outil comparer alors le nom de la boîte noire rencontrée avec ceux du fichier mémoires. Une fois le modèle mémoire identifié, l'outil intégre automatiquement le contrôleur mémoire orienté interopérabilité adapté.

Dans un deuxième temps, l'outil relie entre elles les primitives orientées interopérabilité afin de créer deux chaînes de scan, une pour les bascules et verrous, l'autre pour les mémoires RAM. Les chaînes créées respecteront l'organisation présentée dans la section «3.4.3.1) Structure d'arbre», à savoir que les éléments seront regroupés par module et, à l'intérieur d'un module, par domaine d'horloge puis par proximité spatiale. Cette organisation implique que l'outil analyse les arbres d'horloge de chaque composant avant de générer les chaînes de scan.

L'outil génére ensuite la netlist finale du circuit incorporant les capacités d'interopérabilité. De plus, à ce niveau, les variables d'état sont également caractérisées. Il crée donc les deux fichiers de description associés, c'est à dire le fichier de description des bascules et verrous et celui de description des mémoires RAM.

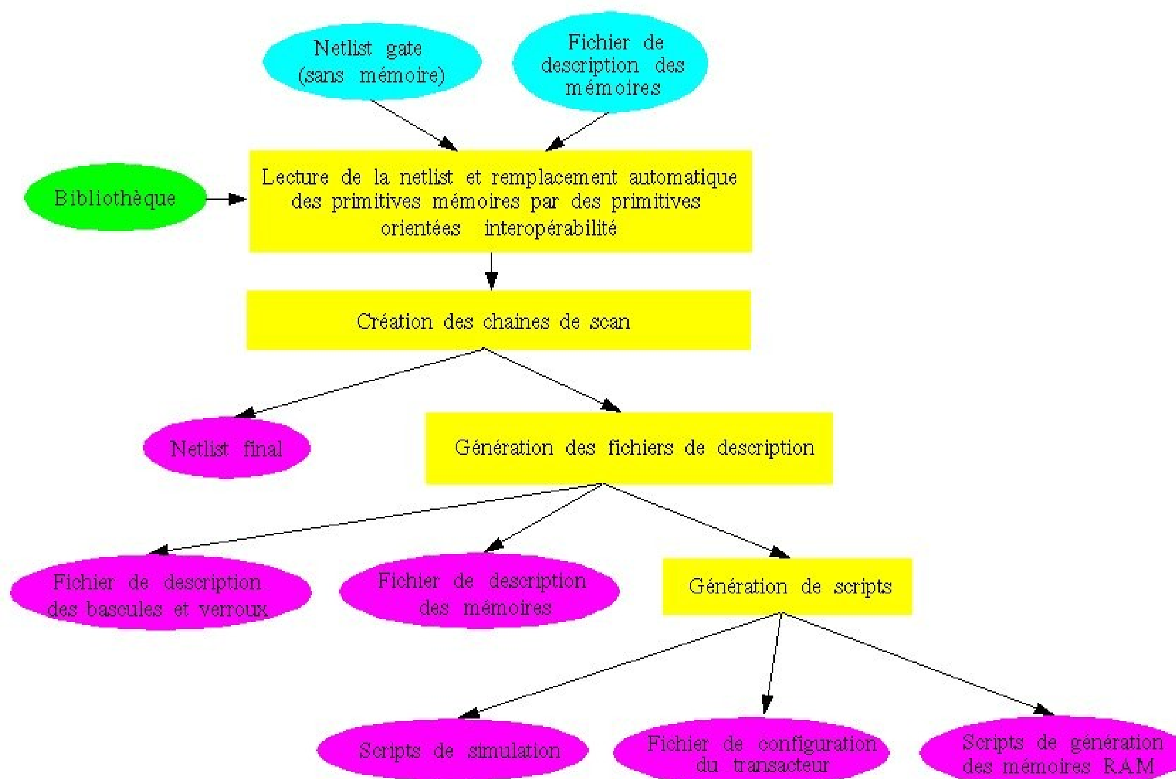


Figure 28 : Principe de l'outil d'interopérabilité

Enfin vient l'étape d'écriture des scripts. Tout d'abord, un script pour la simulation est généré. Pour le moment seul le simulateur NcSim de chez Cadence est supporté. Lorsque l'on démarre une simulation, il faut exécuter ce script qui rajoute alors deux commandes : une fonction de capture d'état «save_state» et une fonction de restauration d'état «load_state».

D'autre part, le logiciel crée également un fichier de configuration pour le transacteur qui pilote l'accès à l'état du circuit. Le transacteur réalisé communique avec l'environnement logiciel avec des mots de taille fixe égale à 160bits. Le fichier de configuration indique le nombre d'échanges qu'il faudra effectuer durant un transfert d'état ainsi que le nombre de bits de bourrage.

Enfin, l'outil est capable d'écrire quelques scripts pour les outils de génération de modèles mémoire. Par exemple, les RAMs double port sont supportées. Si le logiciel identifie ce type de mémoire, il générera un script pour Coregen (outil de génération de mémoires pour FPGA Xilinx), ainsi qu'un modèle mémoire pour Palladium.

Dans la pratique et dans un premier temps, le logiciel réalisé travaille sur une netlist de référence obtenue à l'aide de l'outil de synthèse HdlIce, outil spécifique à l'émulateur Palladium de

Cadence. HdlIce fournit des netlists écrites en Verilog utilisant des primitives de la bibliothèque «qtref». Cette première version de l'outil d'interopérabilité ne supporte que ce format.

Il est également à noter que HdlIce utilise seulement une dizaine de primitives de type bascule/verrou dans ses synthèses. La bibliothèque de l'outil d'interopérabilité ne contient donc qu'une dizaine d'éléments ce qui a permis un développement rapide en seulement trois jours.

Pour conclure, le logiciel d'interopérabilité développé s'intègre parfaitement avec les outils utilisés chez STMicroelectronics. Cependant, en l'état actuel, il est dépendant de l'émulateur Palladium. Si nécessaire, il évoluera pour supporter d'autres langages que Verilog pour la «netlist» d'entrée, et qu'il connaisse d'autres bibliothèques que la «qtref».

3.7) Flot de vérification orienté interopérabilité

Maintenant que l'on sait facilement et rapidement mettre en place des plateformes d'émulation/prototypage interopérables, il faut exploiter au mieux les nouvelles possibilités offertes. Aujourd'hui, les circuits à vérifier sont des systèmes sur puce. Les prochains paragraphes vont proposer un nouveau flot de vérification adapté aux systèmes sur puce qui sera bien entendu orienté interopérabilité et permettra une réduction de la durée de vérification et des coûts associés grâce à la coopération des différentes techniques et machines d'émulation/prototypage matériel.

3.7.1) Architecture d'un système sur puce

Afin de proposer un flot de vérification adapté aux systèmes sur puce, il faut, dans un premier temps, caractériser l'architecture d'un tel système.

Classiquement, un système monopuce est constitué d'un assemblage de composants (ou IPs) qui communiquent entre eux via un réseau de communication. Ce réseau peut être un bus ou un réseau sur puce (NoC). La figure 29 donne un exemple d'architecture de système sur puce avec le circuit Nomadik STn8810 de chez STMicroelectronics [STM06].

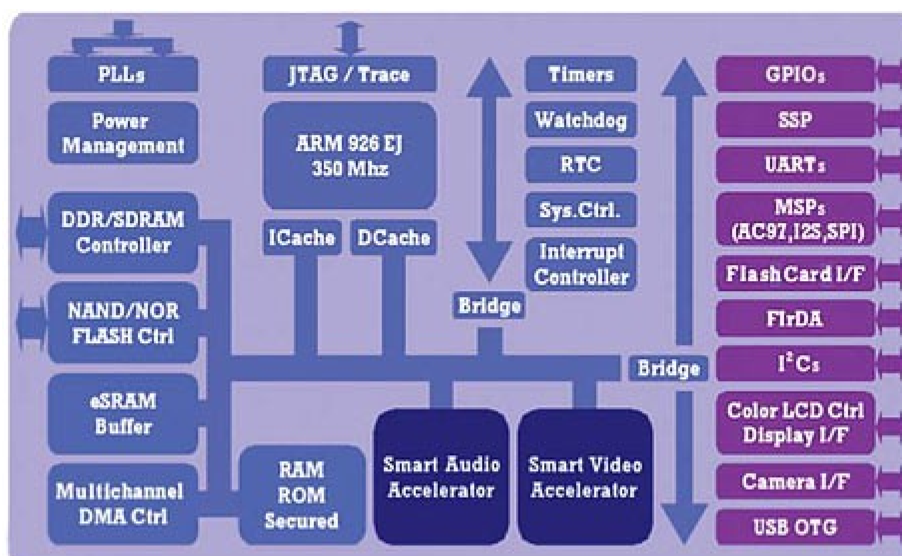


Figure 29 : Architecture du système sur puce Nomadik

Dans ce genre d'architecture, on rencontre des composants matériel, comme des «timers», des mémoires (eSRAM Buffer) ou des processeurs (ARM926), ces derniers exécutant des logiciels embarqués.

Dans l'architecture du Nomadik, on remarque qu'il y a plusieurs réseaux de communication

reliés entre eux par des passerelles (bridge). Au final, ce système sur puce est un assemblage de plusieurs sous-systèmes, chaque sous-système étant défini comme un assemblage de composants autour d'un unique réseau de communication.

Ainsi, lorsqu'une erreur apparaît dans le système, cette erreur ne concerne généralement qu'un seul sous-système du circuit, voire un seul composant. Travailler sur un composant est souvent plus simple que de travailler avec un circuit complet car il y a moins d'interactions à considérer. Une stratégie de vérification consiste donc à identifier le sous-ensemble défectueux et à concentrer l'effort de déboguage uniquement sur ce point. Ce concept va être développé dans les prochains paragraphes.

3.7.2) Phase de vérification et phase de déboguage

Le flot de vérification proposé repose sur la complémentarité existante entre les différentes solutions industrielles d'émulation et prototypage matériel, comme cela a été mis en évidence dans le deuxième chapitre. D'une part, les plateformes de prototypage offrent une très grande vitesse d'exécution mais des capacités de déboguage très faibles, quasi nulles pour certaines solutions. D'autre part, les machines d'émulation offrent d'excellentes capacités de déboguage, proches de celles des simulateurs, avec une bonne vitesse d'exécution qui est néanmoins inférieure (facteur dix à cent) à celle d'une plateforme de prototypage.

Classiquement, on exécute un scénario de test et on débogue ce test sur la même machine. En prototypage, l'exécution sera rapide mais le déboguage sera lent, ce sera l'inverse en émulation. Ainsi, il n'est pas toujours aisé de déterminer quelle solution sera la meilleure, cela dépendra essentiellement du nombre d'erreurs qui subsistent dans le circuit lorsque celui-ci arrive en émulation/prototypage. L'idée qui hémerge naturellement de ce constat est d'utiliser la vitesse d'exécution des plateformes de prototypage et la capacité de déboguage des émulateurs au lieu de choisir entre l'une des deux machines (figure 30).

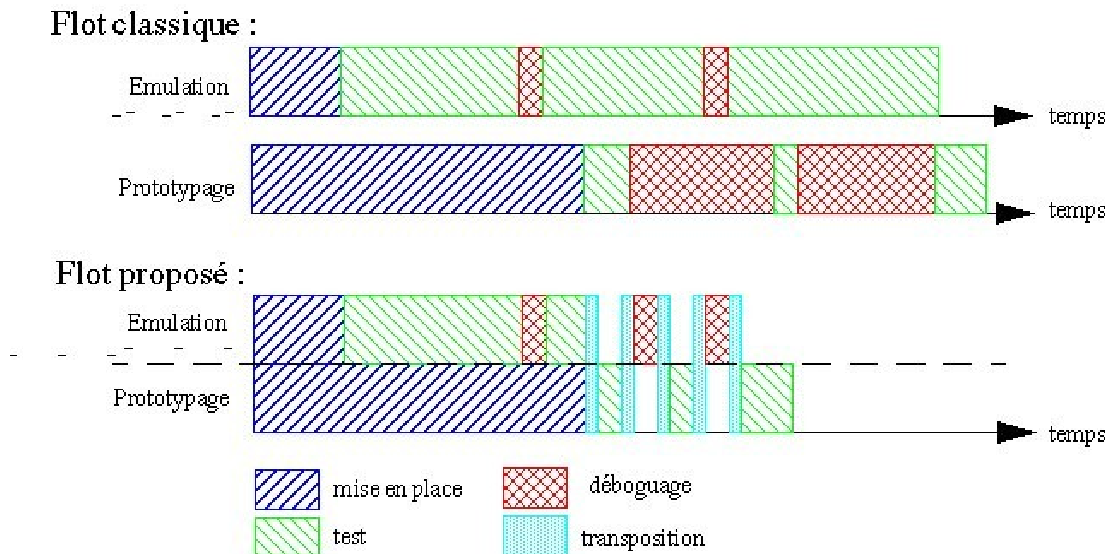


Figure 30 : Comparaison entre le flot classique et le flot proposé

Cette thèse propose donc une stratégie de vérification en deux phases. La première consiste à l'exécution d'un scénario de test, la seconde au déboguage des erreurs.

Durant la phase de test, qui se déroule sur une machine rapide, donc une plateforme de prototypage, on va périodiquement réaliser des captures de l'état du circuit.

Si le résultat du scénario de test est incorrect, il va falloir passer en phase de déboguage. On rejouera donc le scénario de test sur une machine plus adaptée au déboguage, à savoir un simulateur ou un émulateur. Bien entendu, le scénario de test sera démarré à partir de la dernière capture d'état précédent l'instant d'apparition de l'erreur.

Si l'écart de vitesse entre la plateforme d'exécution et la plateforme de déboguage est très important et que le nombre de captures d'état est faible, il se peut que rejouer une partie d'un scénario de test nécessite une longue durée. Par exemple, si la phase de test s'est déroulée sur une plateforme ayant une vitesse d'exécution de 10MHz et que celle de déboguage a une vitesse d'exécution de 100kHz, reproduire cinq minutes de test nécessitera environ huit heures et vingt minutes. Cette durée est acceptable. En considérant que l'erreur survienne au bout d'une heure de test en prototypage, il aurait fallu plus de quatre jours de test en émulation avant d'arriver au problème. D'autre part, l'expérience des équipes d'émulation/prototypage matériel de chez STMicroelectronics a montré qu'il faut souvent plusieurs jours pour trouver une erreur sur une plateforme de prototypage. Donc huit heures de déboguage comparées à plusieurs jours est tout à fait acceptable.

Cependant, on peut encore réduire cette durée. Lorsque l'on capture périodiquement l'état d'un circuit, on échantillonne en même temps l'évolution des signaux d'entrée du circuit. En rejouant le test, on utilise les enregistrements de ces signaux au lieu d'un environnement extérieur. On peut ainsi se permettre d'altérer la vitesse d'exécution du système sans modifier l'intégrité du test. Par conséquent, on pourra rejouer une première fois le scénario de test sur la plateforme de prototypage en contrôlant la vitesse d'exécution de manière à augmenter le nombre de captures d'état dans la fenêtre concernée par le bogue (figure 31).

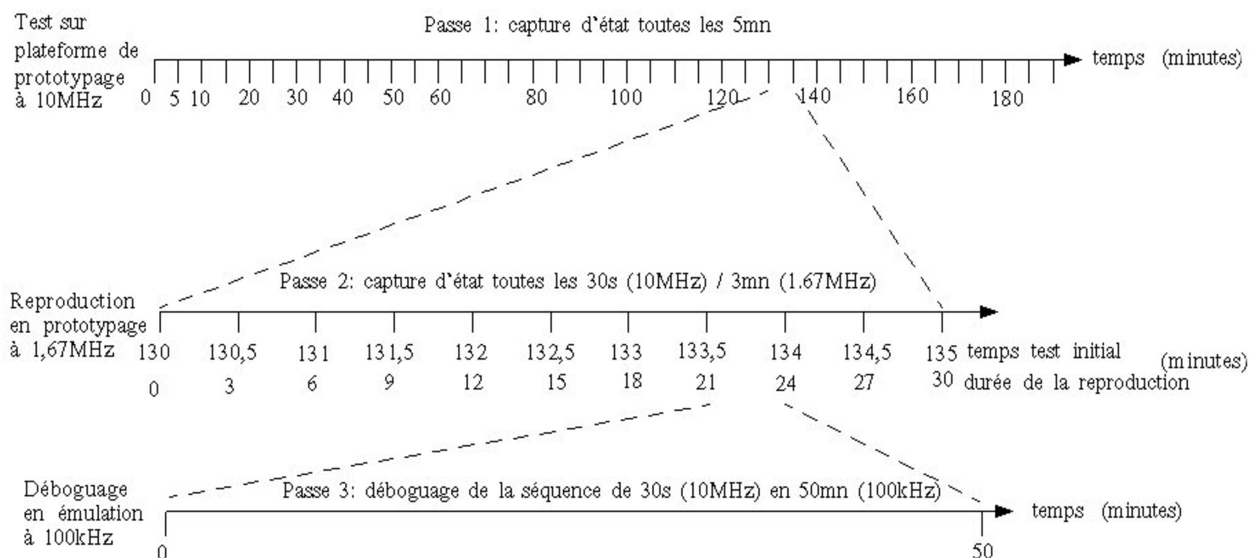


Figure 31 : Flot de déboguage en trois passes

En considérant qu'une capture d'état nécessite trois minutes sur la plateforme de prototypage, on décide de réaliser dix nouvelles captures d'état dans la fenêtre comportant l'apparition du bogue. Cette nouvelle passe prendra environ trente minutes, la fréquence moyenne de la plateforme de prototypage va descendre à 1.67MHz. On va ainsi obtenir une capture d'état beaucoup plus proche de l'instant d'apparition du bogue. Il faudra ensuite rejouer seulement trente secondes de test initial, ce qui nécessitera environ cinquante minutes en émulation à 100kHz. Au final, avec cette technique à trois passes, la détection d'un bogue aura nécessité une heure et vingt

minutes au lieu de plusieurs journées.

Enfin, la mise en oeuvre d'une plateforme d'émulation est toujours plus rapide que celle d'une plateforme de prototypage. Par exemple, dans [BIG04], on apprend qu'il faut environ six mois de travail pour mettre en oeuvre une plateforme de prototypage Aptix avec un circuit d'environ sept millions de portes alors qu'il ne faut que quelques semaines pour la mise en oeuvre de ce même circuit sur un émulateur. Par conséquent, la phase d'exécution des tests démarrera toujours en émulation, en attendant que la machine de prototypage soit prête (figure 32). Dans cette figure, un projet de vérification démarre au temps «t0». Les fichiers sources subissent le flot d'interopérabilité afin d'obtenir une description instrumentée du circuit qui va servir de point d'entrée aux flots d'émulation de prototypage. Au temps «t1», la plateforme d'émulation est opérationnelle, celle de prototypage est toujours dans sa phase de mise en oeuvre. Les tests débutent alors en émulation. Au temps «t2», la plateforme de prototypage est à son tour opérationnelle. On applique alors la stratégie proposée par ce paragraphe, à savoir d'utiliser la plateforme de prototypage pour l'exécution des scénarios de test et l'émulation pour déboguer.

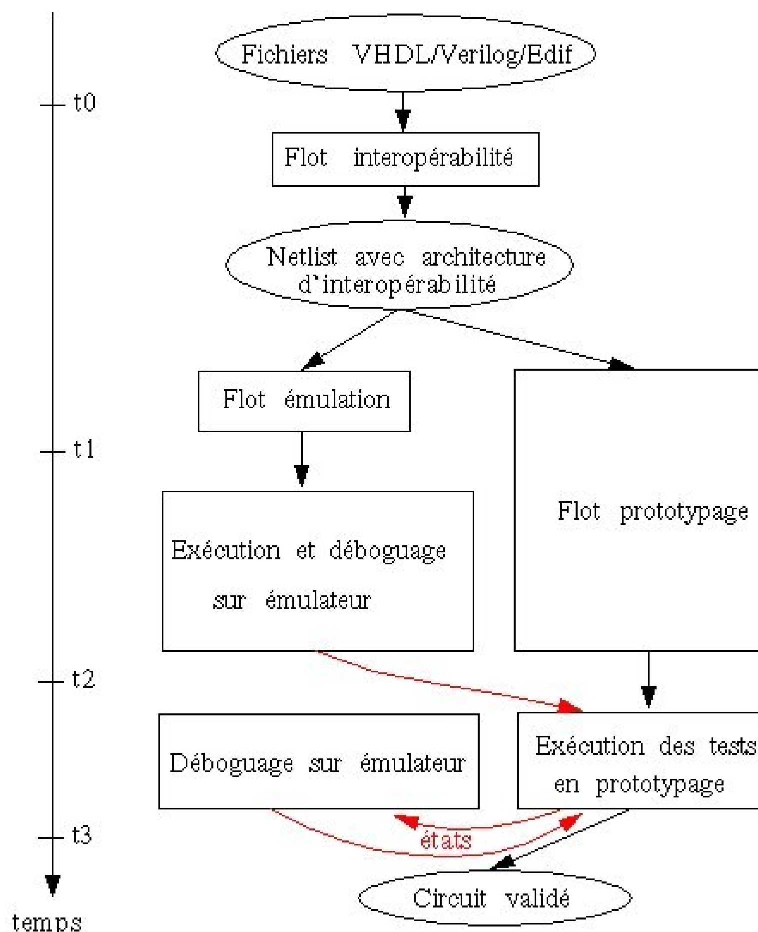


Figure 32 : Stratégie d'utilisation d'un émulateur collaborant avec une plateforme de prototypage

3.7.3) Optimisation : travail sur un sous-ensemble

La stratégie de vérification présentée permet de réduire la durée de vérification par la coopération d'une machine adaptée au déboguage (émulateur ou éventuellement simulateur) et d'une machine offrant une grande vitesse d'exécution (plateforme de prototypage). L'obtention de

cette coopération nécessite l'ajout d'une architecture d'interopérabilité à l'architecture du circuit en cours de vérification. Cela se traduit par une augmentation de la taille du circuit (jusqu'à 30%) durant la vérification (le chapitre IV donnera des chiffres précis sur cette augmentation de surface). Ainsi, la stratégie proposée peut engendrer un surdimensionnement de la capacité des machines que l'on veut utiliser. Dans le cas d'une plateforme de prototypage, qui a un coût modéré, rajouter un ou deux FPGAs n'est pas un problème tant que l'on ne dépasse pas la capacité d'acceuil du système. Par contre, pour les émulateurs qui sont tous très onéreux, voir inabordables pour la plupart des entreprises, augmenter la capacité de la machine peut s'avérer un réel problème.

Un enjeux important est donc d'optimiser l'utilisation des machines à coût élevé afin de réduire ou au moins limiter le coût du déboguage.

3.7.3.1) Optimisation des ressources, de la vitesse et du coût du déboguage

Pour atteindre les améliorations visées, on peut exploiter l'architecture modulaire des systèmes sur puce. En général, lorsqu'une erreur se produit, la première étape de déboguage consiste à identifier le module défectueux. On peut envisager ensuite de concentrer le travail de déboguage sur ce sous-ensemble en n'émulatant/simulant que cette partie du circuit (figure 33).

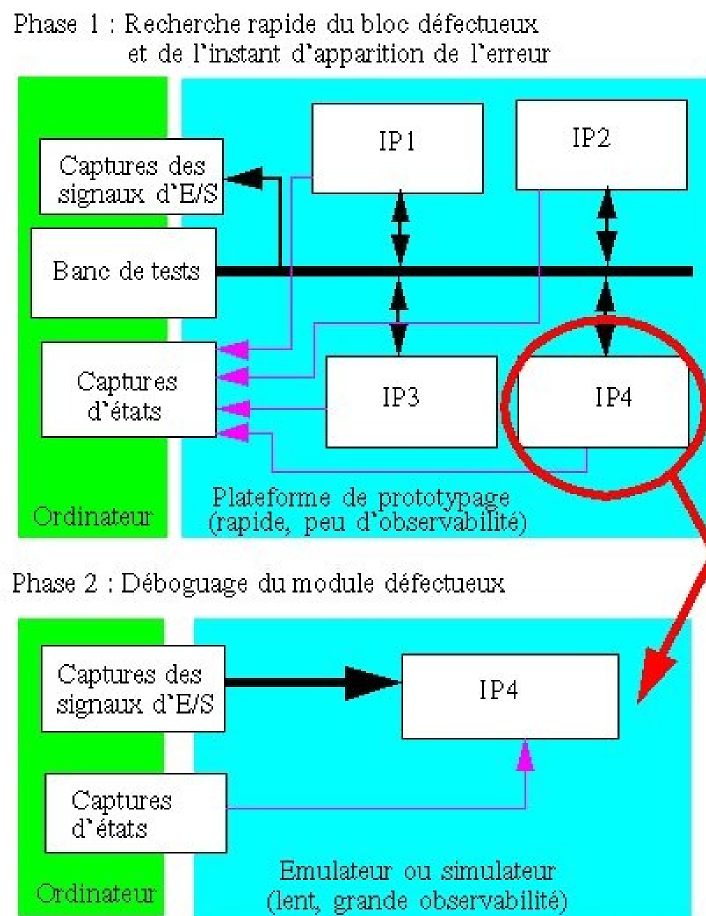


Figure 33 : Flot de vérification orienté SoC et interopérabilité

Avec une telle stratégie, on va pouvoir atteindre plusieurs objectifs. Tout d'abord, vu que l'on travaille sur un sous-ensemble du circuit, le déboguage ne nécessitera pas une grande capacité d'émulation. De plus, avec les émulateurs multi-domaines et multi-utilisateurs (Palladium, Veloce),

si on n'utilise que quelques domaines pour déboguer un circuit, on pourra facilement déboguer plusieurs sous-ensembles de circuit en parallèle. Ainsi, on optimisera fortement l'utilisation des machines onéreuses. La figure 34 montre la stratégie d'utilisation d'un parc de machines d'émulation/prototypage matériel. L'idée est d'utiliser un maximum de plateformes de prototypage en coopération avec un minimum d'émulateurs sur lesquels on effectue la mise en oeuvre des plateformes interopérables ainsi que le déboguage en parallèle des sous-systèmes défectueux.

D'autre part, travailler sur un sous-ensemble au lieu d'un circuit complet va permettre d'améliorer la vitesse d'exécution du système de déboguage. En émulation, il n'y a pas de lien simple entre la taille du circuit et sa vitesse d'exécution. Cette vitesse est en effet souvent contenue dans une plage spécifique à la machine et à la technique d'émulation/prototypage utilisée. Cependant, la complexité des différents domaines d'horloge a tendance à faire baisser les performances. En travaillant sur un sous-ensemble, il y a une grande probabilité que le sous-ensemble considéré n'utilise pas toutes les horloges du circuit, ce qui se traduira par une meilleure vitesse d'exécution.

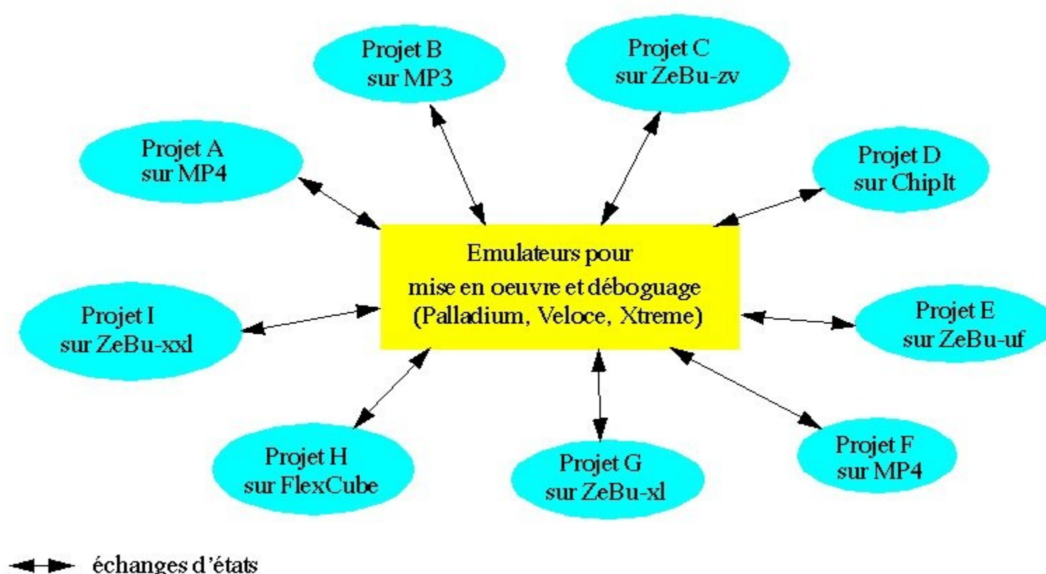


Figure 34 : Utilisation d'un parc de machines d'émulation/prototypage matériel

En simulation, la vitesse d'exécution est, quant à elle, une fonction linéaire de la taille du circuit [PET00], [PET01]. D'après ces articles, il faut environ dix instructions CPU pour simuler une porte logique. Ainsi, réduire la taille du circuit simulé permet d'améliorer la vitesse de simulation. Si l'on réduit le circuit à déboguer à un sous-ensemble de moins de cent mille portes, on peut alors espérer une vitesse de simulation de plusieurs kilohertz, ce qui peut être tout à fait suffisant pour déboguer.

Pour conclure, travailler sur un sous-ensemble permet :

- de réduire la durée de vérification
- d'optimiser l'utilisation des ressources d'émulation
- de limiter le nombre d'émulateurs dans un parc de machine
- d'améliorer la vitesse d'exécution des environnements de déboguage
- d'autoriser le recours à la simulation comme outil de déboguage
- de réduire le coût de la vérification
- d'offrir le choix entre un déboguage rapide (émulation) ou économique (simulation)

3.7.3.2) Obtention de l'environnement du sous-ensemble défectueux

Le précédent paragraphe a montré les multiples intérêts de travailler, en déboguage, avec uniquement un sous-ensemble défecteux. La difficulté majeure de la technique proposée réside dans la modélisation de l'environnement du sous-ensemble. Il faut en effet être capable de reproduire au cycle près les signaux de stimulation du sous-ensemble. L'autre difficulté, commune aux méthodes classiques, réside dans l'identification du module défectueux.

Les différents composants des systèmes sur puce (SoC) communiquent grâce à un réseau de communication. Souvent, lorsque l'on recherche un module défectueux, on suit le parcours des données à travers les différents modules. On analyse ses messages entrant et sortant de chacun afin de s'assurer qu'il fonctionne correctement. La connaissance des échanges effectués sur le réseau de communication est donc indispensable. Elle sera également nécessaire pour reproduire les stimulations du module défectueux.

Souvent, les réseaux de communication internes aux systèmes sur puce ont des bandes passantes très élevées, les bus de communication utilisant plusieurs dizaines, voire centaines de bits. Echantillonner de tels bus va engendrer un énorme volume de données. Il est peu probable qu'une telle acquisition puisse être réalisée en temps réel sur une plateforme de prototypage fonctionnant à plusieurs méga-hertz.

Afin de résoudre ce problème, une stratégie multi-passes est à nouveau proposée. La première passe est celle d'exécution du scénario de test à pleine vitesse incluant des captures d'état périodiques. La deuxième passe affine la connaissance de la fenêtre temporelle contenant l'erreur. Dans cette passe, on va rejouer le scénario de test entre les deux captures entourant l'instant d'apparition de l'erreur. En plus, durant cette passe, on va échantillonner l'ensemble des signaux échangés sur le réseau de communication. On effectuera éventuellement des captures d'état supplémentaires, conformément à la stratégie présentée dans la section «3.7.2) Phase de vérification et phase de déboguage». La fréquence du système prototypé diminue. A partir de l'échantillonnage des signaux du réseau de communication, on identifie le module problématique. On pourra alors, dans la troisième passe, procéder au déboguage sur émulateur ou simulateur du module défecteux.

3.7.4) Apports de la technique proposée

La technique de vérification proposée, orientée système sur puce et interopérabilité, a deux apports stratégiques, un vis à vis du circuit en cours de validation et l'autre vis à vis de la gestion d'un parc de machines.

3.7.4.1) Apports vis à vis du circuit en validation

La statégie proposée augmente l'efficacité du déboguage en favorisant la coopération des techniques et machines d'émulation/prototypage matériel. Ceci implique une réduction de la durée de vérification. Un important apport de la méthode est de rendre reproductible, au cycle d'horloge près, des scénarios de test qui normalement ne sont pas reproductible à ce niveau de précision (co-émulation transactionnelle ou émulation en mode ICE).

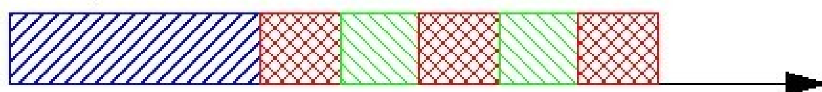
De plus, la capacité de sauvegarde et restauration d'état peut également réduire la durée de développement des logiciels embarqués. En effet, traditionnellement, le développement des logiciels embarqués démarre par la mise au point de la séquence de démarrage («boot»), puis par la validation des différentes tâches logicielles. A chaque correction du programme embarqué, il faut rejouer toute la séquence antérieure au problème qui vient d'être corrigé. Ainsi, plus on avance dans le développement et plus la séquence préalable à l'instant qui nous intéresse va être longue. Cette exécution préalable n'a pas d'intérêt vis à vis de la vérification vu que cette séquence a déjà été validée.

La solution proposée pour réduire cette séquence de démarrage consiste à capturer l'état du circuit une fois passée la séquence d'exécution déjà validée. Lorsque l'on modifiera le programme, la modification n'affectera pas le code exécuté durant la séquence de démarrage. Ainsi, en restaurant l'état du circuit à partir d'une capture d'état, on peut sauter la séquence de démarrage (figure 35).

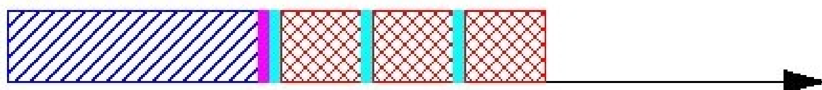
Bien évidement, avant de mettre en marche le système après une restauration d'état, il faudra mettre à jour la mémoire programme avec la nouvelle version du code à exécuter. Cette opération peut être complexe si plusieurs mémoires caches sont utilisées par le processeur embarqué.

La structure d'arbre proposée comme format des fichiers de sauvergarde d'état fait qu'il sera très simple de modifier un fichier de sauvegarde pour que celui-ci intègre la nouvelle version du code logiciel à exécuter : il suffit de remplacer un bloc continu de données dans le fichier de sauvegarde d'état des mémoires.

Méthode classique :



Méthode proposée :



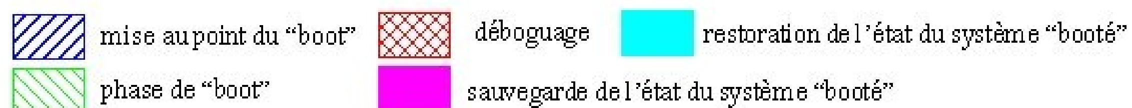


Figure 35 : Réduction de la durée de développement des logiciels embarqués par capture et restauration de l'état "booté" du circuit

Pour conclure, l'interopérabilité associée à de bonnes stratégies permet non seulement de réduire la durée de déboguage du matériel mais également la durée de développement des logiciels embarqués.

3.7.4.2) <u>Apports vis à vis de la gestion d'un parc de machines</u>

Le deuxième apport majeur de ce chapitre est un apport économique. En effet, la stratégie de vérification présentée optimise l'utilisation des émulateurs à très haut coût de fonctionnement et favorise l'utilisation massive de plateformes de prototypage à coût modéré. Avec la technique proposée, un parc de machines principalement dédié au déboguage du matériel utilisera plus de plateformes de prototypage que d'émulateurs là où, avec les techniques actuelles, on favorisait l'émulation.

De plus, la stratégie proposée donne le choix, pour le déboguage matériel, entre la rapidité en utilisant un émulateur, et la maitrise des coûts en utilisant un simulateur là où, avec les techniques actuelles, l'utilisation d'un simulateur n'est pas envisageable.

Enfin, l'interopérabilité offre également une nouvelle possibilité d'optimisation de la gestion d'un parc de machines en autorisant une gestion dynamique des ressources. Chez les grands industriels du semi-conducteur comme STMicroelectronics, il arrive que durant une période, plusieurs projets nécessitent des ressources d'émulation/prototypage en même temps. Dans certains cas, les besoins peuvent dépasser les capacités disponibles. Il faut alors arbitrer l'accès aux machines. Cependant les projets prioritaires n'utilisent pas forcément les ressources allouées en

permanance : on pourrait réaliser une partie des tests des projets moins prioritaires dans ces temps morts.

Réduire les temps morts lorsqu'il y a pénurie de ressources disponibles est une priorité. Actuellement, cela demande énormément d'efforts de communication entre les différentes équipes utilisant la même ressource et peut engendrer des conflits.

De plus, les projets qui utilisent des ressources pour une longue durée peuvent bloquer la mise au point d'autres projets. Par exemple, considérons un circuit dont la validation nécessite toutes les capacités d'un émulateur. Le mode de fonctionnement est STB, le scénario de test dure une journée entière. Durant la journée d'exécution du test, aucun autre projet ne peut être lancé sur la machine. La mise au point d'une plateforme de vérification nécessite de nombreux essais de quelques minutes. Ainsi, si un projet est sur le point de fonctionner, on réalise un court essai, on corrige l'erreur, relance un flot de compilation de quelques heures et on peut refaire un essai. Dans une journée, on peut réaliser trois à quatre essais de cinq minutes. Dans le cas présent, une journée de travail sera perdue étant donné qu'on ne pourra pas accéder à l'émulateur. Il serait donc intéressant que les courts essais de quelques minutes soient prioritaires par rapport aux tests en cours. Rallonger de dix minutes un test de plusieurs dizaines d'heures n'est pas contraignant.

L'interopérabilité permet de répondre à ce problème dans certains cas, en autorisant une gestion dynamique des ressources (figure 36). Les cas concernés sont les vérifications pour lesquels il est possible de capturer à la fois l'état du circuit ainsi que celui de tout son banc de test. Ceci concerne seulement les émulations en mode STB.

Figure 36 : Allocation dynamique des ressources d'émulation

Dans la figure 36, on observe l'utilisation d'un émulateur multi-domaines et multi-utilisateurs. La machine considérée possède sept domaines et peut donc supporter jusqu'à sept utilisateurs en parallèle. Un projet «A» nécessite quatres domaines pour un test long. Dans la journée, un test «D» doit utiliser trois domaines pour un essai de quelques minutes. Le projet «D» va alors suspendre l'exécution du projet «A», une capture d'état est réalisée. Quand l'essai du projet «D» est terminé, le projet «A» est relancé à partir de la capture d'état.

Le projet «B» est un projet à faible priorité, il est donc sans arrêt interrompu par des projets de priorité supérieure. Cependant, lorsque le projet «A» termine et qu'il libère quatres domaines, le projet «B» va utiliser les ressources libérées au lieu d'attendre que ses domaines initialement alloués (domaines 4 à 6) soient disponibles.

Ainsi, grâce à l'interopérabilité, on peut envisager la création d'un serveur de ressources qui, en fonction des priorités des différents projets désirant accéder à une machine, choisira les projets actifs et leurs ressources allouées à chaque instant de manière à maximiser l'utilisation des machines.

En conclusion, l'interopérabilité associée à une bonne stratégie permet d'optimiser l'utilisation d'un parc de machines et de limiter le nombre d'émulateurs très onéreux. Le coût de la vérification est donc réduit.

3.8) Conclusion

Ce chapitre a défini la notion d'état d'un circuit ce qui a permis d'introduire le concept d'interopérabilité en émulation/prototypage matériel. L'interopérabilité permet une coopération entre les différentes techniques et machines d'émulation/prototypage ce qui, associé avec un nouveau flot de vérification, permet à la fois de réduire la durée et le coût de la vérification.

Afin d'utiliser cette notion d'interopérabilité, ce chapitre a proposé une architecture spécifique permettant l'accès en lecture et écriture à l'état d'un circuit. Grâce à cette architecture, toute machine d'émulation/prototypage supportant l'interface de communication SceMi est interopérable.

Enfin ce chapitre a présenté un outil d'interopérabilité qui ajoute automatiquement l'architecture d'accès à l'état d'un circuit à l'architecture du circuit à vérifier. La solution proposée est générique ; elle ne dépend pas des propriétés spécifiques de la plateforme de vérification ciblée. Cependant, cette qualité se paye au niveau des ressources nécessaires, à savoir que la taille des circuits émulés/prototypés va augmenter. Cette hausse peut atteindre 40% de la taille du circuit dans le cas des circuits utilisant énormément de mémoire. De plus, le flot de vérification proposé nécessite plus d'étapes de mise en oeuvre par rapport aux flots classiques.

Il est important de mesurer l'impact de ces inconvénients afin de s'assurer que la technique proposée est bénéfique. Théoriquement, les inconvénients sont minimes. En effet, les étapes de mise en oeuvre supplémentaires ont été automatisées, donc ne surchargent pas le travail des utilisateurs. L'augmentation de taille peut quant à elle avoir un impact négligeable. En prototypage, la taille de la logique combinatoire fait qu'en ce qui concerne les ressources FPGA, en général, les LUTs sont quasi toutes utilisées alors qu'il reste un bon nombre de bascules disponibles. De plus les FPGAs sont rarement pleins à 95%, une moyenne de 70% est courante. L'interopérabilité ajoute essentiellement des registres, l'augmentation de ressources devrait donc bien se passer dans ce cas. En émulation, les machines fonctionnent par domaine de capacité fixe. La granularité est de l'ordre du million de portes logiques. Ainsi augmenter la taille du circuit de 20% peut ne pas nécessiter l'utilisation d'un domaine supplémentaire. Dans tous les cas, le prochain chapitre va prouver, par des chiffres mesurés sur un cas pratique, la validité de la solution proposée.

Chapitre IV - Application de l'interopérabilité au circuit hls25

4.1) Présentation du circuit hls25....................98
4.2) Objectifs de la vérification....................98
4.3) Plateforme de vérification....................98
    4.3.1) Sélection du couple technique/machine....................98
    4.3.2) Architecture du banc de test....................99
4.4) Apport de l'interopérabilité....................100
4.5) Mise en oeuvre de plateformes interopérables....................101
    4.5.1) Instrumentation du circuit....................101
    4.5.2) Intégration du transacteur d'interopérabilité....................101
    4.5.3) Plateformes Palladium II....................103
    4.5.4) Plateformes ZeBu-XL....................105
    4.5.5) Plateforme NcSim/zTide....................107
4.6) Déboguage sur un sous-ensemble....................108
    4.6.1) Echantillonnage des signaux d'interface....................108
    4.6.2) Phase de déboguage....................109
4.7) Conclusions....................110

4.1) Présentation du circuit hls25

Le circuit hls25 est un circuit de télécommunication intégrant un système de code correcteur d'erreurs conforme à la norme DVBS2. Le système offre la transmission de données à haut débit avec avec une grande fiabilité (taux d'erreur inférieur à 10^{-7}). Pour atteindre ce faible taux, le circuit hls25 utilise deux algorithmes de code correcteur [URA05].

La mise en oeuvre des algorithmes de code correcteur est très complexe, gourmande en nombre de calculs et requiert donc une architecture adaptée. Ainsi, hls25 intègre trois cents soixante processeurs de type SIMD (Single Instruction Multiple Data) travaillant tous en parrallèle. Chaque processeur peut communiquer avec plusieurs petites mémoires de type SRAM. Grâce à cette structure, le système peut travailler avec des paquets de 64800 bits.

L'ensemble du circuit fonctionne avec une seule horloge de fréquence 300MHz et utilise environ un million et demi de portes logiques dont quarante mille registres.

4.2) Objectifs de la vérification

Le circuit hls25 est la deuxième évolution du circuit hls23. Les modules, en grande partie, ont donc déjà été vérifiés à plusieurs reprises et présentent un grand niveau de fiabilité. La vérification n'est donc pas centré sur le déboguage matériel.

Les évolutions du circuit cherchent à réduire la taille et la consommation du système en optimisant, et donc en réduisant, le nombre de calculs réalisés par le bloc correcteur d'erreur. L'objectif de la vérification est donc d'assurer que la nouvelle architecture permet bien d'obtenir la même efficacité de correction d'erreurs que la version d'origine, c'est à dire que le taux d'erreur reste à 10^{-7}.

4.3) Plateforme de vérification

4.3.1) Sélection du couple technique/machine

Le circuit hls25 travaille par paquet de données (trame de 64800 bits), il lui faut environ cent mille cycles d'horloge pour traiter une trame. La vérification du taux d'erreur de 10^{-7} sur la transmission nécessite l'utilisation d'un million de trames par test. De plus, le circuit possède une vingtaine de configurations différentes qui devront toutes être validées.

Au final, il faut réaliser vingt tests d'un million de trames ce qui représente cent millards de cycles d'horloge par test, soit un total de deux mille millards de cycles. Un si grand nombre de cycles impose que l'environnement de test ait une haute fréquence de fonctionnement afin de réaliser la vérification en un temps raisonnable. Le tableau suivant donne une estimation de la durée de vérification en fonction de la technique sélectionnée.

Technique	*Fréquence*	*Durée de vérification d'une trame*	*Durée totale de vérification (20 tests de 1M trames)*
Simulation HDL	10Hz	2h 46mn	6342ans
Co-émulation «cycle à cycle»	5kHz	20s	13ans
Emulation STB	1MHz	0,1s	23jours et 4h
Prototypage STB	> 5MHz	< 20ms	< 4jours et 15h

Tableau 7 : Durée de la vérification en fonction de la technique sélectionnée

Le tableau précédent montre que seule une vérification à banc de test intégré (STB) aboutit

à une durée de vérification acceptable. L'écart entre émulation et prototypage est estimé à une vingtaine de jours. Hls23 a déjà été testé en prototypage à l'aide d'une machine ZeBu-ZV. Son évolution hls25 est assez proche du circuit d'origine, il ne faudra donc pas plus d'un jour ou deux pour adapter les scripts de mise en oeuvre. En prenant en compte durée de vérification et durée de mise en oeuvre, le prototypage est donc une solution plus avantageuse et plus économique que l'émulation.

Par contre, cette nouvelle version nécessite plus de ressources matérielles que la précédente et ne peut plus rentrer dans une machine ZeBu-ZV c'est pourquoi une plateforme de prototypage ZeBu-XL a été retenue.

4.3.2) Architecture du banc de test

Pour un test donné, la vérification d'un taux d'erreur de 10^{-7} nécessite l'utilisation d'un million de trames auxquelles il faut ajouter un code correcteur. Il faut ensuite bruiter les données obtenues puis les envoyer au circuit pour décodage. Enfin, afin de mesurer le taux d'erreur, il faut comparer chaque trame décodée avec la trame d'origine. Tout ce traitement doit se faire en temps réel. Seul un banc de test synthétisable et embarqué dans la plateforme de prototypage permettra de tenir la cadence désirée. La figure 37 montre l'architecture de la solution retenue.

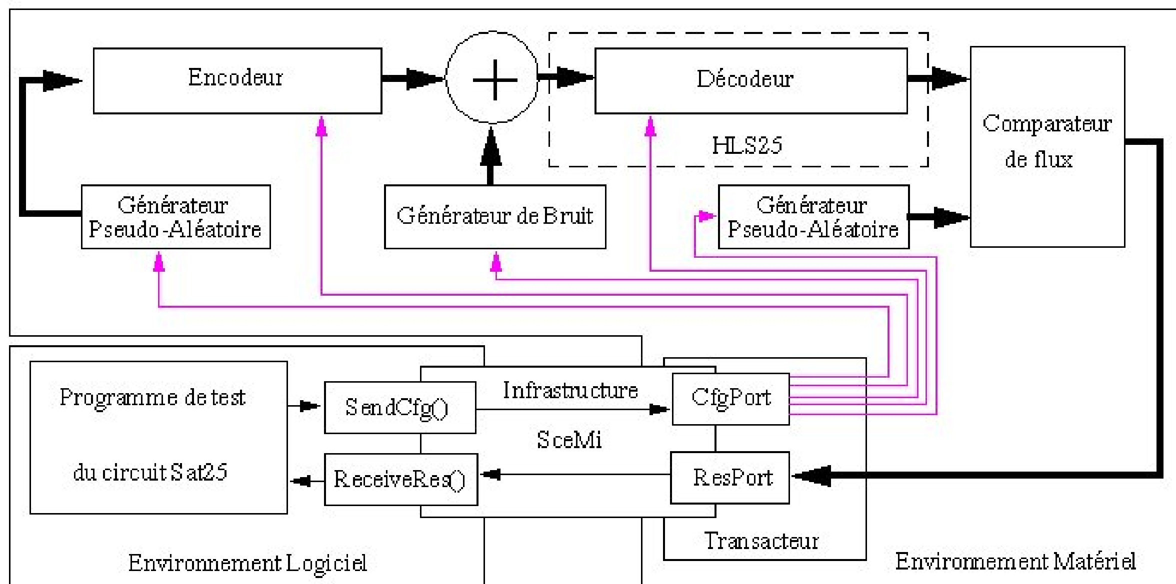


Figure 37 : Architecture de la plateforme de test du circuit hls25

Le système se compose d'un générateur pseudo-aléatoire de données qui alimente un codeur qui ajoute le code correcteur d'erreurs. Les données ainsi obtenues traversent alors un générateur de bruit dont la sortie est connectée au circuit hls25. Les données en sortie sont comparées aux données d'origine. Vue la taille des données et la latence de traitement, il n'est pas envisageable d'utiliser une immense FIFO pour stocker les données générées. La solution retenue consiste à utiliser un deuxième générateur, identique au premier, et regénérant les données d'origines avec un retard égal à la latence de traitement. Ensuite, le résultat de chacune de ces comparaisons doit être enregistré. Pour cela, un transacteur est connecté au banc de test matériel afin de transmettre les résultats à un ordinateur pour stockage sur disqe dur. De plus, durant la phase de «reset», ce transacteur permet d'entrer les paramètres du générateur de bruit et la configuration du circuit hls25 pour chacun des tests.

4.4) Apport de l'interopérabilité

La vérification de hls23, la précédente version du circuit, reposait notamenent sur une machine ZeBu-ZV fonctionnant à 6MHz. Durant les tests de hls23, un bogue matériel a été rencontré. Il se produisait dans une seule configuration, après qu'environ cent mille trames avaient été correctement traitées (c'est à dire au bout d'une heure de test).

L'identification de cette erreur a representé un véritable défi. Les concepteurs constataient que le circuit fonctionnait bizarrement à partir d'un certain point mais n'avaient aucune idée quant à l'origine de l'erreur. La stratégie suivante a alors été adoptée : tracer l'ensemble des signaux d'interface entre les principaux modules à savoir, tracer les signaux en sortie du générateur de trame, de l'encodeur, du décodeur. L'obtention des traces nécessitant de rejouer le test depuis le début, il a fallu une heure pour amener le système aux alentours de l'instant d'apparition du bogue. Les capacités de déboguage dynamique du ZeBu-ZV ont ensuite été utilisées pour capturer les signaux. La fréquence du système est alors tombée à 10kHz, la capture aura nécessité une journée. Après deux jours d'analyse des signaux, le module défectueux a été identifé.

La suite du processus de déboguage nécessitait d'observer plus en détail le module problématique. Malheureusement, les capacités de déboguage du ZeBu-ZV étaient insuffisantes. Un émulateur Palladium est donc intervenu pour finir le déboguage. Là encore, il a fallu rejouer complétement le test pour amener le système quelques cycles avant l'apparition du bogue puis, il a fallu tracer l'ensemble des signaux du module problématique. L'environnement Palladium fonctionnait à 1MHz, soit six fois moins vite que la plateforme ZeBu-ZV. Amener le système à l'instant d'apparition du bogue a donc nécessité plus de six heures. Enfin, il n'aura fallu que quelques minutes au Palladium pour tracer les signaux désirés et, après deux jours d'analyse, l'erreur était identifiée et corrigée.

Si on fait le bilan de cette expérience, on constate que :

- Amener le système à l'instant problématique a nécessité une heure sur ZeBu-ZV et six heures sur Palladium.
- Tracer des signaux a nécessité une journée sur ZeBu-Zv et une dizaine de minute sur Palladium
- Le déboguage a nécessité six jours de travail dont deux entièrement consacrés à la reproduction de l'erreur.
- Si l'erreur s'était manifestée sur les derniers cycles du scénario de test, le temps consacré a rejouer ces tests auraient été encore plus important : environ quarante heures de déboguage supplémentaires auraient été nécessaires, ce qui aurait impliqué huit jours de travail dont quatre entièrement consacrés à la reproduction de l'erreur.

Si l'interopérabilité avait existé à l'époque, l'identification de cette erreur aurait été accélérée. En effet, en capturant toutes les dix minutes l'état du système, on aurait pu directement réaliser le déboguage sur Palladium à partir d'une de ces captures. L'obtention des signaux d'interface puis de ceux du module défectueux aurait alors nécessité au plus quelques heures au lieu de deux jours, ce qui aurait réduit de 30% la durée de déboguage. Dans l'hypothèse d'une erreur survenant tardivement dans le test, ce gain s'éleverait à 50%.

En conclusion, l'interopérabilité doit permettre d'accélérer l'identification des erreurs c'est pourquoi cette stratégie a été déployée dans la validation du circuit hls25. Cela a nécessité la mise en oeuvre d'une plateforme d'émulation interopérable pour les potentiels besoins de déboguage. Un émulateur Palladium II a été sélectionné dans ce but.

4.5) Mise en oeuvre de plateformes interopérables

Le précédent paragraphe a montré l'intérêt d'avoir une machine ZeBu-XL interopérable avec un émulateur Palladium II. De plus, afin de prouver que l'interopérabilité fonctionne également avec un simulateur HDL, une plateforme de simulation interopérable va également être mise en oeuvre. Cette dernière utilisera le simulateur HDL de Cadence NcSim et l'outil de gestion SceMi zTide d'Eve.

Les prochains paragraphes vont détailler la mise en place de ces trois environnements interopérables. De plus, la section «3.7.3.1) Optimisation des ressources, de la vitesse et du coût du déboguage» propose de concentrer l'effort de déboguage d'un circuit sur seulement le sous-ensemble défectueux. La mise en évidence d'erreurs complexes pourrait bénéficier de cette stratégie. Ainsi, deux méthodes de déployement de cette technique seront présentées.

4.5.1) Instrumentation du circuit

La mise en oeuvre de plateformes interopérables commence obligatoirement par l'obtention de la netlist de référence. Comme expliqué dans la section «3.6.1) Flot d'obtention de la netlist de référence», l'outil de synthèse «HdlIce» de Cadence dédié à l'émulateur Palladium a été retenu comme synthétiseur de référence. La mise au point de cette synthèse aura été réalisée en une journée de travail, la synthèse en elle-même nécessitant une heure de calculs sur un ordinateur puissant.

La netlist ainsi obtenue contient une description complête du circuit hls25, de son banc de tests et de ses mémoires. Tout cet ensemble doit être instrumenté à l'aide de l'outil «InteroperabilityCompile» présenté dans la section «3.6.3) Outil d'interopérabilité : modification automatique d'une "netlist gate"».

Chaque machine d'émulation/prototypage possède son propre format de description des mémoires. Ces modèles ne sont pas compatibles entre deux machines. Ainsi, il n'est pas nécessaire que l'outil «InteroperabilityCompile» modifie les primitives mémoire utilisées par HdlIce vue que ces dernières ne seront pas comprises par les outils utilisés dans les flots de prototypage. Par conséquent, il faut préciser les modules mémoire à l'outil «InteroperabilityCompile» afin qu'il les remplace par des boîtes noires et qu'il insère leurs signaux de scan associés. L'instrumentation du circuit hls25 dépourvue de ces mémoires aura finalement nécessité 5s.

En outre, hls25 utilise huit types de mémoire (ROMs et RAMs) pour chacune desquelles il a fallu développer un modèle interopérable. Ces modèles ont été construits à partir d'une mémoire générique à «W» ports d'écriture synchrones et «R» ports de lecture asynchrones. Au final, une heure de travail aura suffit à l'écriture des huit modèles.

En conclusion l'instrumentation du circuit hls25 aura nécessité un peu plus d'une heure de travail, la principale difficulté étant l'écriture des wrappers mémoires.

4.5.2) Intégration du transacteur d'interopérabilité

Une fois le circuit hls25 instrumenté, il faut lui ajouter le transacteur d'interopérabilité en charge de la gestion des captures et restaurations d'états. En plus d'avoir instrumenté le circuit, l'outil «InteroperabilityCompile» a généré des fichiers de configuration de ce transacteur. Ainsi, pour mettre en oeuvre une plateforme interopérable, il faut inclure les fichiers RTL de ce transacteur à la netlist interopérable du circuit pour en réaliser une synthèse adaptée à la machine ciblée.

Il faut également insérer ce transacteur dans la plateforme. Pour faciliter ce point, la stratégie préconisée par la norme SceMi est utilisée. Quel que soit le circuit à émuler/prototyper utilisant au moins un transacteur, le «top» du système sera obligatoirement un module Verilog

nommé «bridge» dans lequel on trouvera uniquement une instanciation du circuit, des transacteurs et des générateurs d'horloge SceMi (figure 38). Ainsi, pour ajouter le transacteur d'interopérabilité à un système, il suffira d'ajouter une instance dans ce fichier. Si le système initial n'utilise pas de transacteurs avant interopérabilité, il faudra alors créer un fichier bridge. Dans tous les cas, avec cette méthode, l'intégration du transacteur d'interopérabilité ne demande qu'une dizaine de minutes, le temps d'éditer le fichier «bridge».

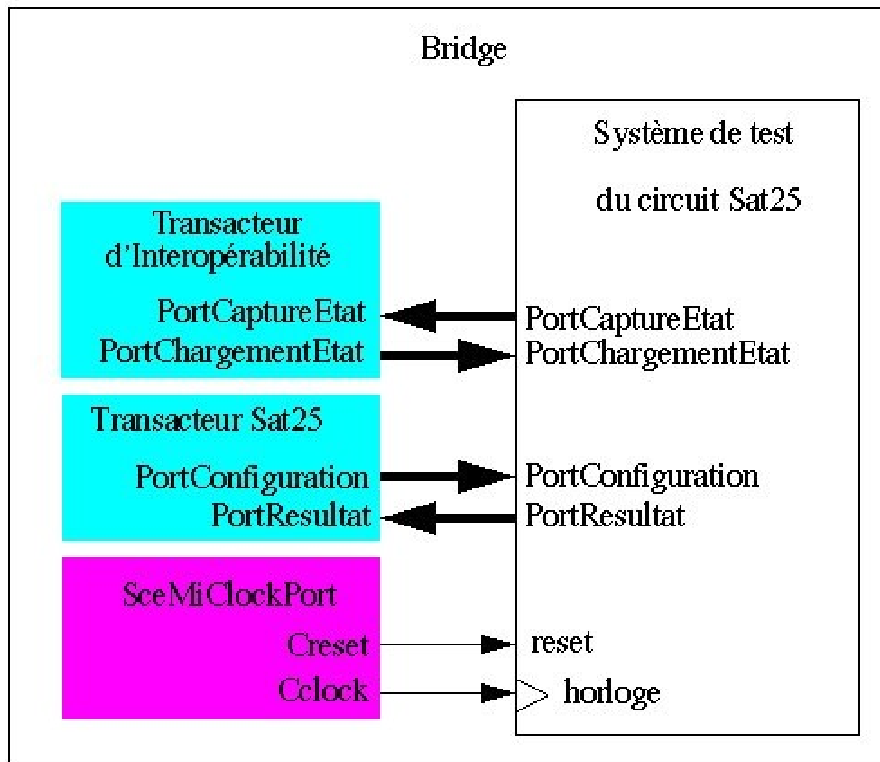


Figure 38 : Principe du module bridge appliqué au circuit hls25

Les plateformes hls25 interopérables utilisent deux transacteurs, un lié au banc de test du circuit et un destiné à la gestion des états. Au niveau matériel, l'intégration de plusieurs transacteurs SceMi indépendants ne pose aucun problème grâce à la stratégie du module «bridge». L'intégration au niveau logiciel est quant à elle un peu plus complexe. En effet, si les transacteurs sont indépendants, ils sont reliés à des programmes eux aussi indépendants, chacun d'entre eux cherchant à initialiser le matériel et à gérer l'ensemble des communications SceMi.

Afin d'éviter tout problème, la stratégie suivante a été déployée : aucune application SceMi liée à un transacteur ne doit initialiser la connection avec le matériel, un pointeur sur l'infrastructure SceMi lui sera fourni. Par contre, chaque application est en charge d'initialiser ses canaux de communication. De plus, chacune des applications est encapsulée dans un objet C++.

L'environnement logiciel lié à l'interopérabilité fonctionne à l'aide de deux threads : l'un lié à l'utilisateur et l'autre lié aux communications SceMi. Ce deuxième thread est un esclave du matériel, c'est à dire que le matériel est toujours à l'initiative des communications. Pour que le logiciel d'interopérabilité fonctionne correctement, il suffit que la méthode Scemi «ServiceLoop()» soit appelée régulièrement. Cette méthode exécute, si besoin est, les communications entre les environnements logiciel et matériel. Vu que toutes les applications SceMi utilisent obligatoirement cette méthode, il suffit d'initialiser le logiciel d'interopérabilité pour que celui-ci évolue en même temps que d'autres logiciels SceMi.

Au final, chaque logiciel applicatif doit obligatoirement intégrer deux fonctions, une pour initialiser ses canaux de communication (InitSceMiChannels) et une pour exécuter le test (RunTest), cette fonction appelant autant que possible la méthode «ServiceLoop()».

La fonction «main()» connectera dans un premier temps le logiciel à la machine puis, initialisera le logiciel d'interopérabilité suivi du logiciel applicatif (figure 39). Elle lancera enfin le test en utilisant la fonction «RunTest()» du logiciel applicatif. Cette fonction ne rendra la main qu'à la fin du test.

```
int main(int argc, char * argv[]) {

        t_InterTrans * InterTrans = NULL; // pointer on interoperability transactor object
        t_hls25Trans * hls25Trans = NULL; // pointer on hls25 transactor object
        SceMi * scemi = NULL; // pointer on scemi infrastructure

        scemi = InitHadware( .... ); // connect to hardware side

        InterTrans = new t_InterTrans; // create interoperability transactor object
        InterTrans->InitSceMiChannels(); // init transactor channels

        hls25Trans = new  t_hls25Trans; // create hls25 transactor object
        hls25Trans->InitSceMiChannels(); // init transactor channels

        hls25Trans->RunTest(); // Launch test

        // End of test
        delete  hls25Trans;
        delete  InterTrans;
        CloseHardware( ... ); // close scemi channels, release hardware

        return 0;
}
```

Figure 39 : Intégration du logiciel hls25 et du logiciel d'interopérabilité

4.5.3) <u>Plateformes Palladium II</u>

Deux environnements Palladium II ont été mis en oeuvre. Le premier correspond à celui présenté dans la section «4.3.2) Architecture du banc de test» et n'intègre pas l'architecture d'interopérabilité. Le second intègre les capacités d'interopérabilité. La comparaison de ces deux plateformes permet de vérifier que l'interopérabilité ne perturbe pas le fonctionnement du circuit et permet de mesurer son impact sur la durée de mise en oeuvre, la vitesse d'exécution, et l'utilisation des ressources d'un émulateur (tableau suivant). Plusieurs points ressortent de cette comparaison.

Tout d'abord, en émulation, le coût de mise en oeuvre de l'interopérabilité est quasi nul. L'outil d'instrumentation travaille avec la bibliothèque de primitives «qtref». Durant la synthèse de la «netlist» instrumentée, le synthétiseur HdlIce comprend toutes les primitives instanciées et n'a quasi aucun travail à faire d'où la faible durée de cette étape (trois minutes dans le cas de hls25). L'instrumentation a triplé le nombre de portes du circuit. De ce fait, la compilation est un peu plus longue. Néanmoins, cette durée supplémentaire reste faible et donc acceptable. Ainsi, recourir à l'interopérabilité n'augmente quasiment pas la durée de mise en oeuvre d'une plateforme d'émulation.

			Sans interopérabilité	***Avec interopérabilité***	***Coût de l'interopérabilité***
Durée de synthèse		Netlist de reference	1h10mn	1h10mn	+0%
		Netlist instrumentée	-	3mn	-
		Total	1h10mn	1h13mn	+4,1%
Durée d'instrumentation			-	5s	-
Compilation			4mn	5mn	+25%
Durée totale du flot			1h14mn	1h21mn	+9%
Ecriture des modèles mémoires			1h	1h	0%
Nombre de domaine Palladium II			1 (utilisation 45%)	2 (utilisation 55%)	+100%
Taille du circuit	***Mémoires***	Mémoires	105929 Portes	211336 Portes	+99%
		Contrôleurs	-	502145 Portes	-
		Total	105929 Portes	713481 Portes	+574%
	Transacteurs		859 Portes	25264 Portes	+2841%
	Registres	hls25 + TB	44428 Registres	88856 Registres	+100%
		Transacteur	157 Registres	1760 Registres	+1021%
		Contrôleurs mémoires	-	19048 Registres	-
		Total	44585 Registres (311633 Portes)	109664 Registres (769164 Portes)	+146%
	Logique	hls25 + TB	312944 Portes	1006868 Portes	+222%
		Transacteurs	222 Portes	11628 Portes	+5138%
		Contrôleurs mémoires	-	235781 Portes	-
		Total	313166 Portes	1254277 Portes	+301%
	Total		730728 Portes	2234777 Portes	+206%
Fréquence			1,4MHz	650kHz	-54%
Temps de capture/restauration			-	5s	-

Tableau 8 : Impact de l'interopérabilité sur un émulateur Palladium II

Le deuxième impact révélé par le tableau concerne la taille du circuit qui a triplé (+206%) suite à l'étape d'instrumentation. Le circuit hls25 est probablement un des pires cas concernant l'augmentation de surface. En effet, ce circuit est constitué de plusieurs dizaines de mémoires, qui représentent initialement 15% de la surface du circuit. Elles ont une grande largeur de mots, une faible profondeur et sont donc de petite taille (moyenne de 2ko par RAM). En instrumentant le circuit, chacune de ces mémoires reçoit un contrôleur d'interopérabilité qui est également de petite taille (plus petit qu'un test de type BIST). Cependant, dans la cas particulier de hls25, ces

contrôleurs sont plus gros que les mémoires et, vu le nombre de mémoires utilisées, ils sont responsables d'une énorme augmentation de surface (+574%). Si on avait réalisé les RAMs à l'aide de registres instrumentés comme les registres du circuit (donc sans utiliser les contrôleurs mémoires), ce coût en surface aurait été plus faible (estimation à +300%). De plus, les BISTs sont également plus grands que les mémoires. De manière à ne pas augmenter la surface du circuit, les concepteurs de hls25 ont intégré des BISTs partagés. Ainsi, toutes les RAMs du même type sont vérifiées par le même module de BIST. Dans le cadre de l'interopérabilité, il serait intéressant de développer des contrôleurs mémoire partagés utilisant la même stratégie.

D'autre part, durant l'instrumentation, chaque registre est remplacé par son modèle interopérable qui utilise deux registres et quatre multiplexeurs. Classiquement, la logique occupe une place bien plus importante que celle des registres. Ainsi, le coût de la logique ajoutée par l'interopérabilité devrait être faible. Dans le cas de hls25, ce coût est beaucoup plus important car ce circuit utilise autant d'éléments logiques que de registres. Ainsi, en dédoublant chaque registre, l'instrumentation a également doublé la taille de la logique.

Sans interopérabilité, hls25 nécessite un seul domaine Palladium II utilisé à 45%. Après l'instrumentation, il faut ajouter un deuxième domaine, l'ensemble étant rempli à 55%. La nécessité du second domaine s'est jouée à quelques ressources près. Si le circuit avait eu des propriétés plus «standards», c'est à dire utiliser des RAMs plus grandes et moins nombreuses ainsi qu'une plus grande proportion de logique par rapport aux registres, l'augmentation de surface aurait été moindre et la version interopérable aurait tenu sur un seul domaine. Le coût final en surface de l'interopérabilité se mesure avec le nombre de domaines utilisés et, dans le cas de hls25 ce coût est d'un domaine. Avec un circuit plus «classique», ce coût aurait pu être nul.

Enfin, l'interopérabilité semble fortement impacter la fréquence de fonctionnement du circuit émulé puisque celle-ci est divisée par deux. Néanmoins, là encore il faut considérer les propriétés exceptionnelles de hls25 qui utilise très peu de logique et qui a donc des chemins combinatoires très courts. Un tel circuit est obligatoirement grandement sensible à un allongement de ses chemins combinatoires, d'où l'impact constaté sur la vitesse. Là encore, avec un circuit aux propriétés plus «classiques», la fréquence aurait été probablement moins affectée. Dans tous les cas, la fréquence de l'environnement interopérable reste parfaitement acceptable.

En conclusion, l'interopérabilité en émulation a un coût quasi nul sur la durée de mise en oeuvre. Dans le cas de hls25, l'interopérabilité impacte fortement fréquence et taille du circuit. Néanmoins, cette influence est en grande partie due à des propriétés rares de l'architecture du circuit qui font de cet exemple probablement un cas extrême vis à vis de l'interopérabilité. On peut donc espérer que cet impact sera plus faible sur des circuits aux propriétés plus «classiques».

Enfin, une capture ou une restauration d'état a nécessité environ 5s. Une capture d'état occupe environ 400ko de données ce qui, à 650kHz, doit être transmis en 4,92s. La durée mesurée est donc tout à fait acceptable et conforme aux prévisions.

4.5.4) **Plateformes ZeBu-XL**

Comme pour l'émulation avec Palladium II, deux plateformes ZeBu-XL ont été mises en oeuvre afin de mesurer l'impact de l'interopérabilité en prototypage (tableau suivant).

Comme en émulation, l'interopérabilité augmente la taille du circuits prototypé. Les chiffres obtenus en prototypage avec «hls25» sont comparables à ceux obtenus en émulation et s'expliquent toujours par les propriétés «originales» du circuit (répartition logique/registres, nombre et taille des mémoires). Ainsi, prototypé la version interopérable de «hls25» nécessite deux FPGAs supplémentaires. Cet impact de taille est sans conséquence, que ce soit sur la vitesse d'exécution, la durée de mise en oeuvre ou l'utilisation des ressources, ce qui va être justifié dans les prochaines lignes.

			Sans interopérabilité	***Avec interopérabilité***	***Coût de l'interopérabilité***
Durée de synthèse		Netlist de reference	30mn	1h10mn	+133%
		Netlist instrumentée	-	1h50mn	-
		Total	30mn	3h	+500%
Durée d'instrumentation			-	3mn	-
Partitionnement, placement & routage			1h30mn	2h	+33%
Durée totale du flot			2h	5h	+150%
Ecriture des modèles mémoires			1h	1h	0%
Nombre de FPGAs			4	10	+150%
Taille du circuit	***Mémoires***		1047 BRAMs	2094 BRAMs	+100%
	Registres	hls25 + TB	44428 Registres	88856 Registres	+100%
		Transacteur	157 Registres	1760 Registres	+1021%
		Contrôleurs mémoires	-	19048 Registres	-
		Total	44585 Registres	109664 Registres	+146%
	Logique	hls25 + TB	183980 LUTs	197475 LUTs	+7%
		Transacteurs	31 LUTs	2380 LUTs	+7577%
		Contrôleurs mémoires	-	57672 LUTs	-
		Total	184011 LUTs	257527 LUTs	+40%
Fréquence			800kHz	800kHz	0%
Temps de capture/restauration			-	4s	-

Tableau 9 : Impact de l'interopérabilité sur une machine ZeBu-XL

Considérons d'abord l'aspect durée de mise en oeuvre. D'une part, il faut compter une heure de travail supplémentaire pour l'obtention de la «netlist» de référence. D'autre part, l'interopérabilité impose l'utilisation de six FPGAs supplémentaires qu'il faut placer et router, d'où une augmentation de la durée de mise en oeuvre. Néanmoins, ce point négatif peut être annulé. En effet, l'ensemble des FPGAs peut être placé et routé en parallèle sur plusieurs ordinateurs. Dans ce cas, le temps global de placement et routage ne dépassera pas une heure et l'interopérabilité n'aura alors qu'un très faible impact sur la durée de mise en oeuvre.

Le doublement du nombre de RAMs est en parti responsable de l'ajout des six FPGAs supplémentaires. Hls25 intègre plusieurs dizaines de mémoires qui ne sont pas bien adaptées aux primitives FPAGs. Ainsi quatre FPGAs ont été alloués uniquement pour supporter les mémoires doublons. Les primitives BRAMs sont toutes allouées mais leur espace d'adressage n'est qu'à moitié utilisé. Au final, ce surcoût FPGA serait moindre avec des mémoires remplissant mieux les

BRAMs. De plus, vu que quatres FPGAs sont utilisés uniquement pour leurs BRAMs, on aurait pu réaliser certaines RAMs à l'aide de registres et LUTs : cela aurait aisni réduit le nombre de FPGAs supplémentaires.

D'autre part, la machine ZeBu-XL est mono-utilisateur, ce qui signifie que quel que soit le nombre de FPGAs utilisés sur la plateforme, on ne peut pas utiliser les autres FPGAs pour prototyper un autre circuit. Tant que la machine peut contenir la version instrumentée d'un circuit, l'augmentation de taille n'est donc pas un problème.

Enfin, il est à signaler que l'augmentation de taille n'a pas altéré la fréquence de fonctionnement du système. La stratégie consistant à utiliser une chaîne de scan pour accéder à l'état du circuit n'a augmenté le nombre de signaux inter-FPGAs que d'une dizaine d'unités, ce qui n'a pas impacté le facteur de multiplexage. Néanmoins, ce facteur aurait pu être affecté par la modification du partitionnement engendrée par l'augmentation de la taille du circuit.

Hls25 travaille avec des bus internes allant jusqu'à un millier de bits de large. En partitionnant ce circuit, les bus doivent être distribués à travers l'ensemble des FPGAs ce qui engendre un grand nombre de signaux d'interconnection et un taux de multiplexage élevé. Ces propriétés expliquent d'une part la faible fréquence obtenue en prototypage (seulement 800kHz) et d'autre part le faible impact du changement de partitionnement.

Pour conclure, le prototypage de hls25 a montré que l'interopérabilité fonctionne. Il a été possible de capturer et restaurer un état sur une plateforme ZeBu-XL en environ 4s. Cette durée est tout à fait acceptable et conforme à la théorie. De plus, l'impact de l'interopérabilité sur la durée de mise en oeuvre est négligeable. Dans le cas de hls25, l'augmentation de taille n'a pas altéré la fréquence du système. Néanmoins, il faudrait réaliser d'autres expériences avec des circuits ayant un faible niveau de multiplexage sans interopérabilité afin de valider le faible coût sur la fréquence du système.

4.5.5) Plateforme NcSim/zTide

Afin de valider le passage émulation/prototypage vers simulation HDL, un environnement de simulation a été mis en oeuvre, basé sur le simulateur NcSim de Cadence. La section «3.4.2.4) Restauration d'état» a montré, qu'en simulation, la gestion des états doit se faire à l'aide de scripts car l'utilisation de l'architecture d'interopérabilité est dans ce cas trop lente. Ainsi, pour la mise en oeuvre de cet environnement, le circuit hls25 non instrumenté et son transacteur ont été compilés pour NcSim. Dans cette application, il est souhaitable de conserver le même environnement logiciel que celui utilisé en émulation et prototypage. Ainsi, le transacteur d'interopérabilité est remplacé par une version adaptée à la simulation où les fonctions de capture et restauration d'états sont gérées par le logiciel et non par le matériel.

Afin d'assurer la portabilité de l'environnement, la communication entre logiciel et simulateur se fait par la norme SceMi. Avec ses machines ZeBu, la société Eve fournit un outil nommé zTide qui permet de connecter un logiciel SceMi avec n'importe quel simulateur HDL via la norme DPI du langage Verilog. Cet outil est très simple d'utilisation (il suffit d'inclure la bonne bibliothèque à la compilation du logiciel) et offre de très bonnes performances.

Sur cette plateforme, l'interopérabilité a un coût quasi nul puisque, par rapport à une plateforme non interopérable, il suffit de compiler, en plus du circuit, le transacteur d'interopérabilité dédié à la simulation. Avec cet environnement, une capture d'état nécessite 21s et une restauration 33s, ce qui est tout à fait acceptable et cohérent avec la thérorie.

4.6) Déboguage sur un sous-ensemble

L'efficacité de la méthode de déboguage proposée dans la section «3.7.3.1) Optimisation des ressources, de la vitesse et du coût du déboguage», consistant à concentrer l'effort de déboguage sur le sous-ensemble défectueux, n'a malheureusement pu être démontrée dans cette application car aucune erreur complexe n'a été rencontrée durant la vérification de hls25.

Néanmoins, ce cas d'application a permis de valider le concept. La section «3.7.2) Phase de vérification et phase de déboguage» propose un flot de déboguage en trois étapes. La première consiste à exécuter le test en réalisant périodiquement des captures d'état du système, par exemple toutes les cinq minutes. La deuxième étape cherche à identifier le module défecteux lorsqu'une erreur se produit. Pour cela, les bus de communication entre les différents modules composant le circuit sont observés lorsque le scénario de test est rejoué à partir de la sauvegarde précédente l'instant d'apparition de l'erreur. Enfin, une fois le sous-ensemble identifé, la dernière étape consiste à le déboguer.

4.6.1) Echantillonnage des signaux d'interface

La plateforme de vérification hls25 est orientée flot de données, elle est constituée de cinq différents modules (générateur de données pseudo-aléatoires, encodeur, générateur de bruit, décodeur, comparateur de flux) travaillant les uns à la suite des autres, chaque module communiquant uniquement avec son fournisseur de données et son client. Ainsi, travailler sur un de ces cinq sous-ensembles impose d'enregistrer en temps réel tous les signaux d'interface de ces modules. Tous ces modules communiquent par des paquets de six bits, plus un bit de contrôle, ce qui représente au total trente-cinq bits à enregistrer en temps réel. De plus, il n'y a pas d'échange systématique à tous les cycles : ainsi, il est possible de compresser facilement cet échantillonnage à l'aide de la méthodologie proposée dans la section «3.4.5) Gestion des signaux d'entrée/sortie», ce qui permettra de capturer les signaux désirés sur plateforme de prototypage en minimisant l'impact sur la vitesse d'exécution.

Dans le cas de hls25, les fréquences obtenues en émulation et en prototypage sont très proches. Un émulateur permet de facilement observer les signaux désirés avec un faible impact sur la fréquence de fonctionnement (1,4MHz sans observation, 200kHz avec observation). Au final, deux stratégies sont envisageables pour réaliser cette capture : on peut soit utiliser les capacités d'observation de l'émulateur Palladium II, soit utiliser le transacteur d'échantillonnage de bus sur la plateforme de prototypage ZeBu-XL.

L'avantage de recourir à l'émulateur pour tracer les signaux d'interface réside dans la simplicité de l'environnement puisqu'il n'y a aucune modification supplémentaire à apporter au circuit. L'inconvénient de cette méthode se situe au niveau de la vitesse de capture qui est plus faible que celle obtenue avec l'autre méthode. De plus, les enregistrements obtenus sont au format VCD, un format texte dérivé du Verilog, qui occupe un gros volume comparé au format binaire de la deuxième méthode.

La deuxième méthode, basée sur l'utilisation d'un transacteur de capture et compression temps réel, offre une meilleure vitesse d'exécution et engendre des bases de données plus petites. Le seul inconvénient de cette technique réside dans la mise en oeuvre puisqu'il faut connecter les signaux que l'on veut observer au transacteur. Heureusement, l'outil «InteroperabilityCompile» présenté dans la section «3.6.3) Outil d'interopérabilité : modification automatique d'une "netlist gate"» résoud simplement ce problème. En effet, il suffit de passer en paramètre à l'outil les différents modules que l'on veut utiliser comme sous-ensemble de déboguage pour que celui-ci connecte les signaux d'interface au transacteur et génère les différents paramètres du compresseur.

Dans la plateforme de vérification initiale, plusieurs modules de hls25 sont configurés au début du test via un transacteur. Ces paramètres sont constants durant chaque test, il n'est donc pas

nécessaire d'échantillonner ces entrées de configuration c'est pourquoi, on spécifiera à l'outil «InteroperabilityCompile» de ne pas connecter les entrées de configuration des modules au transacteur.

Une fois les modifications réalisées, il faut relancer le flot de partitionnement, placement et routage, ce qui peut nécessiter plusieurs heures. De plus, il est à noter que le routage des signaux d'interface vers le transacteur peut augmenter le facteur de multiplexage de la plateforme de prototypage et ainsi réduire la fréquence du système. Dans le cas de hls25, le routage de quarante-neuf bits supplémentaires n'a pas affecté le facteur de multiplexage (50), la vitesse de la plateforme est resté à 800kHz. Le transfert temps réel des données nécessite une bande passante d'environ 3,5Mo/s qui est bien inférieure à la capacité de l'interface transactionnelle. Ainsi, la capture des signaux d'interfaces n'a absolument pas affecté la vitesse du système prototypé.

4.6.2) Phase de déboguage

Après avoir enregistré les signaux d'interface entre deux captures d'état, il ne reste plus qu'à déboguer le sous-ensemble problématique sur émulateur ou sur simulateur HDL.

Néanmoins, cette étape de déboguage nécessite la mise en oeuvre d'une nouvelle plateforme de vérification. Pour cela, l'outil «InteroperabilityCompile» facilite le travail. En effet, lorsque l'on a spécifié à l'outil les différents modules que l'on désire utiliser comme sous-ensemble, celui-ci a d'une part connecté les signaux d'interface des sous-modules au transacteur mais a également généré deux modules «bridge» pour chacun des modules. Un de ces fichiers est spécifique à la simulation HDL, l'autre à l'émulation. Chaque fichier «bridge» est un fichier Verilog décrivant un module utilisant le sous-module sur lequel on concentre l'effort de déboguage, ce module étant connecté à un générateur d'horloge SceMi et à un transacteur fournissant les signaux de stimulation. Les réponses du module ne sont pas connectées. La figure 40 illustre ce concept avec le module de décodage.

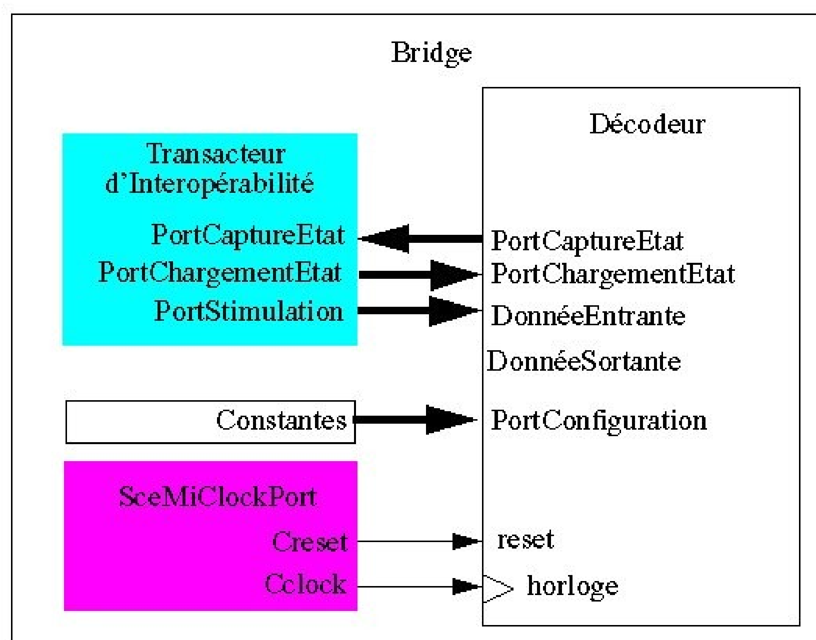


Figure 40 : Fichier "bridge" spécifique au module de décodage émulé

Le module de décodage doit être configuré au début d'un test. Dans la plateforme de vérification du système hls25, ceci est fait via un transacteur. Ces paramètres étant fixes durant le test, ils sont remplacés dans notre cas par des constantes déclarées dans le fichier «bridge». Enfin, si

l'on rejoue le scénario de test sur un émulateur, le chargement d'état se fera via l'architecture d'interopérabilité. Par contre, si l'on travaille sur un simulateur HDL, l'initialisation d'état se fera par script. Ainsi, pour chaque module il existe deux fichiers «bridge». Dans la version spécifique à la simulation, le transacteur d'interopérabilité n'intègre pas des ports de capture/restauration d'état contrairement à celui utilisé dans la version spécifique à l'émulation.

La partie logicielle associée au transacteur supporte le format binaire associé au système de compression ainsi que le format VCD. Dans le cas d'un fichier VCD, ce dernier est converti au format binaire au fur et à mesure que le test avance.

De plus, il est à noter que le transacteur permettant la stimulation du sous-module existe en deux versions. La première est destinée aux émulateurs, la décompression du flux de stimulation étant purement matérielle. Dans la deuxième version destinée aux simulateurs HDL, la décompression est réalisée par du logiciel, ce qui améliore la vitesse de simulation par rapport à un traitement matériel simulé.

Enfin, dans le cas de hls25, la configuration d'un module n'est plus passé par un transacteur lorsque l'on travaille sur un sous-ensemble. Ces paramètres ont été figés à la compilation.

4.7) Conclusions

L'application «hls25» a permis de valider plusieurs points développés dans cette thèse. Tout d'abord, il a été prouvé qu'il est possible de capturer en temps réel et de restaurer l'état d'un circuit sur un simulateur HDL, un émulateur et une plateforme de prototypage. De plus, les techniques d'accès à l'état d'un circuit (architecture ou script) ont parfaitement tenu leur objectif de vitesse puisqu'il ne faut qu'à peine 20s en simulation et 5s en émulation/prototypage pour changer l'état du système.

De plus, cet exemple a validé les outils et le flot permettant de concentrer l'effort de déboguage sur uniquement un sous-ensemble du circuit. Malheureusement, aucune erreur complexe n'est apparu durant la vérification de hls25 et il n'a pas été possible de mesurer l'impact de cette stratégie sur la durée d'identification des erreurs.

Enfin, ce cas d'application a prouvé que l'interopérabilité à un coût quasi nul sur la durée de mise en oeuvre.

Par contre, l'IP «hls25» s'est révélée être un cas particulier pour lequel l'instrumentation du circuit à un fort coût en surface. Néanmoins, de part la haute granularité des machines, cette augmentation n'a finalement aucun impact en prototypage et qu'un faible impact en émulation. De plus, l'augmentation de surface en émulation peut être annulé en utilisant une stratégie offerte par l'interopérabilité, à savoir, travailler sur un sous-ensemble.

Ce cas d'application a enfin mis en évidence que l'interopérabilité peut impacter la fréquence de fonctionnement du système. On constate une diminution de celle-ci en émulation et une stagnation en prototypage. Ce résultat est surprenant et s'explique par les propriétés «originales» de «hls25». A priori, on s'attend à une stagnation de la fréquence en émulation et une légère diminution en prototypage. Sur ce point, il faudrait réaliser de nouvelles expériences avec d'autres circuits. Dans tous les cas, diminuer la fréquence du système émulé n'est pas un inconvénient majeur puisqu'avec l'interopérabilité, l'émulation n'est utilisée que pour le déboguage. Lorsque l'on trace des signaux avec un émulateur, celui-ci travaille certes rapidement, mais pas à sa vitesse maximale. Ainsi, la durée de déboguage ne sera pas obligatoirement altérée. En prototypage, impacter sur la fréquence est plus ennuyeux. L'architecture d'interopérabilité développée cherche à minimiser ce point, ce qui a été confirmé par le cas «hls25». Cependant, il faudrait réaliser d'autres cas pour confirmer ce faible impact.

La vérification de «hls25» n'a pas rencontré d'erreur matérielle. Ainsi, il n'a pas été possible de mesurer l'efficacité de la méthode de déboguage proposée par cette thèse. Néanmoins, même sans expérience, il est évident que cet impact sera significatif puisque l'on peut s'affranchir de

rejouer des dizaines d'heures de test dans les opérations de déboguage.

Enfin, une hypothèse de l'interopérabilité a été mise en défaut par ce test : dans le cas de «hls25», la plateforme d'émulation est plus rapide que celle de prototypage, ce qui limite l'intérêt de recourir au prototypage. Cette faible vitesse d'exécution de la plateforme de prototypage s'explique par un facteur de multiplexage élevé (50). Le circuit consomme énormément de mémoires : quatre FPGAs ont été alloués uniquement pour leurs ressources en BRAMs. Ainsi, si on avait eu plus de temps, on aurait modélisé une partie des mémoires à l'aide de bascules et de LUTs à la place de BRAMs : cela aurait permis de diminuer le nombre de FPGAs nécessaire et d'augmenter leurs taux de remplissage. Ainsi, il y aurait eu moins de signaux à router entre chaque FPGA, le facteur de multiplexage aurait été moindre et, au final, la vitesse d'exécution aurait été supérieure. En optimisant bien le partitionnement, la version «hls23» tenait dans deux FPGAs Virtex2 et fonctionnait à 6MHz. Avec des optimisations analogues, on peut espérer faire tenir «hls25» sur trois Virtex2 et atteindre une fréquence de fonctionnement de 4MHz : on serait alors à nouveau dans les hypothèses de l'interopérabilité. Enfin, les nouvelles plateformes de prototypage utilisent des FPGAs Virtex 4 : «hls25» nécessite deux Virtex4 pour être prototypé. Ainsi, sur un ZeBu-UF, on peut espérer améliorer encore plus la vitesse d'exécution et atteindre plus de 10MHz.

Chapitre V - Conclusions et perspectives

5.1) Conclusions..113
5.1.1) Vérification d'une IP...114
5.1.2) Vérification d'un SoC..114
5.2) Perspectives...115
5.2.1) Extension de la bibliothèque de l'outil «InteroperabiltyCompile»................................115
5.2.2) Essais sur des cas d'application plus complexes..115
5.2.3) Exploitation des possibilités offertes par les FPGAs Xilinx..115
5.2.4) Gestion de plusieurs d'horloges indépendantes..116
5.2.5) Contrôleur mémoire partagé...117

5.1) Conclusions

Les technologies actuelles permettent l'intégration de systèmes de plus en plus complexes sur une seule puce. Ceci induit un accroissement du temps de conception alors que paradoxalement, la concurrence économique impose des temps de mise sur le marché de plus en plus courts. Afin de limiter ce décalage et d'accroître la productivité, l'industrie cherche à améliorer ses procédés de conception et en particulier ses techniques de vérification vu que celle-ci représente jusqu'à 70% de l'effort de conception.

Les techniques de simulation, d'émulation et de prototypage sont massivement déployées afin de tenir les défis de la vérification. Cependant, elles sont très onéreuses au point que l'aspect financier représente un facteur limitant cette étape.

Ainsi, cette thèse a eu pour objectif l'amélioration des stratégies d'utilisation de ces techniques afin de réduire à la fois durée et coût de vérification.

Dans un premier temps, cette thèse a mis en évidence un besoin de coopération entre les différentes techniques de vérification. Ceci représente un véritable défi étant donné que les machines utilisées ne sont pas conçues pour coopérer. Ainsi, afin de palier à cette limitation, la notion d'état d'un circuit a été définie et a permis l'introduction d'un nouveau concept, celui d'interopérabilité en émulation et prototypage matériel.

Dans un deuxième temps, cette thèse a proposé une solution générique permettant l'accès en lecture et écriture à l'état d'un circuit. Ainsi, toutes les machines (simulateur, émulateur et plateforme de prototypage) peuvent être rendues interopérables les unes avec les autres, quel que soit leur mode de fonctionnement.

La mise en oeuvre du concept d'interopérabilité nécessite en outre d'instrumenter les circuits à vérifier. Ce procédé étant complexe, l'outil «InteroperabilityCompile» a été développé afin d'automatiser cette étape.

De plus, l'interopérabilité en émulation matériel a ouvert de nouvelles perspectives quant aux stratégies de vérification. Cette thèse a donc proposé un nouveau flot de vérifications basé interopérabilité adapté à la vérification des systèmes sur puce. L'idée principale est d'exploiter la haute vitesse d'exécution des plateformes de prototypage pour l'exécution des tests ainsi que les hautes capacités de déboguage des émulateurs et simulateurs pour l'identification des bogues. Cette nouvelle stratégie a deux apports stratégiques. D'une part, la durée de vérification est réduite et d'autre part, le taux d'utilisation des machines est amélioré, les parcs de machines sont mieux exploités. Lors des opérations de déboguage, le flot de vérification proposé autorise en outre l'utilisation au choix, d'un simulateur ou d'un émulateur, dans des cas où classiquement, seul un émulateur serait utilisable. On offre ainsi le choix entre un déboguage rapide (émulateur) ou économique (simulateur).

Enfin, l'application «hls25» a permis de valider les concepts, flots et outils développés dans cette thèse. L'interopérabilité a été déployée avec succès entre un simulateur HDL (NcSim), un émulateur (Palladium II) et une plateforme de prototypage (ZeBu-XL) et a atteint ses objectifs d'amélioration de la coopération entre techniques et machines de vérification. De plus, le cas «hls25» a également prouvé que l'interopérabilité est rapide à mettre en oeuvre, la durée d'instrumentation ainsi que l'augmentation des durées des synthèses, compilations, placements et routages étant très faible par rapport à la durée globale de mise en route d'une plateforme de vérification. Néanmoins, ce succès est atténué par une importante augmentation de surface. Le bilan final doit distinguer deux cas, à savoir, si l'on vérifie une IP (circuit de petite taille) ou un SoC (circuit de grande taille).

5.1.1) Vérification d'une IP

Dans le cas de la vérification d'une IP comme «hls25», le coût de l'interopérabilité est au final faible et tout à fait acceptable. En effet, bien que les ressources matérielles nécessaires puissent tripler, ceci n'a quasiment aucun impact concernant les machines.

Les émulateurs sont des machines multi-domaines qui permettent de vérifier en parallèle plusieurs circuits de différentes tailles. Ces domaines ont une capacité fixe, par exemple 1,6M de portes pour un Palladium II. Cette granularité fait que le coût en surface de l'interopérabilité fera augmenter le taux de remplissage des domaines utilisés et, dans le pire cas, nécessitera un domaine de plus par rapport à une version sans interopérabilité. De plus, la stratégie proposée dans la section «3.7.3) Optimisation : travail sur un sous-ensemble» permet de travailler en déboguage sur un sous-ensemble du circuit. Ainsi, non seulement l'interopérabilité a un faible coût en émulation mais, associée à une bonne méthode, elle réduit le nombre de domaines nécessaires durant les phases de déboguage. D'autre part, le cas «hls25» a prouvé que l'interopérabilité peut ralentir la fréquence de fonctionnement de l'émulateur. Cependant, avec la stratégie de vérification développée dans cette thèse, les émulateurs sont utilisés pour faire du déboguage sur de courtes séquences. Ainsi, vu que l'on ne travaille que sur de courtes fenêtres temporelles, cette diminution de fréquence n'est pas génante.

En prototypage, le coût de l'interopérabilité est quasi nul dans le cas de la vérification d'une IP. En effet, les machines ont des capacités de plusieurs dizaines de millions de portes logiques. Ces machines ne sont pas multi-utilisateurs, donc passer de 15% à 30% d'utilisation de la capacité n'a aucun impact sur la gestion d'un parc. De plus, la stratégie d'accès à l'état du circuit via chaîne de scan fait que le nombre de signaux inter-FPGAs n'est augmenté, au plus, que d'une dizaine d'éléments, ce qui n'affecte pas le facteur de multiplexage et donc, n'altère pas la fréquence de fonctionnement. Le seul véritable inconvénient de l'interopérabilité dans ce cas concerne le partitionnement. En effet, l'augmentation de taille des modules peut obliger à modifier le partitionnement par rapport à une plateforme sans interopérabilité. Le nouveau partitionnement peut alors se révéler moins bon quant au nombre de signaux inter-FPGAs et au facteur de multiplexage associé.

Pour conclure, dans le cadre de la vérification d'un petit circuit de type IP, l'interopérabilité remplit tous ses objectifs avec comme seul petit risque, celui de diminuer la fréquence des plateformes de prototypage et donc de rallonger quelque peu la durée des tests. Dans tous les cas, quand bien même ces tests sont un peu plus longs que le minimum, l'efficacité de la stratégie de déboguage doit conduire à une durée globale de vérification, identification et correction des erreurs bien meilleure que celles obtenues par les méthodes actuelles.

5.1.2) Vérification d'un SoC

Le coût de l'interopérabilité est beaucoup plus lourd dans le cas de la vérification d'un SoC. En effet, ces circuits ont une grande taille qui peut, avant même instrumentation, nécessiter 90% des ressources d'un émulateur ou d'une plateforme de prototypage. Ainsi, si l'interopérabilité fait trop grossir ces circuits, il se peut qu'il soit impossible de déployer cette technique sur ce genre de circuit.

Néanmoins, on peut nuancer ce constat. En effet, la tendance fait que les capacités des machines de prototypage sont de plus en plus conséquentes. Par exemple, une machine ZeBu-XL peut intégrer jusqu'à soixante-quatre FPGAs Virtex2, ce qui procure une capacité de quarante millions de portes logiques. L'évolution de cette machine ZeBu-XXL, annoncée fin 2006, devrait au moins doubler cette capacité en utilisant des FPGAs Virtex4. Ainsi, ce type de machine devrait pouvoir facilement absorber des SoCs instrumentés en vu de l'interopérabilité. La seule contrainte est d'avoir à investir dans les plus grosses configurations des plateformes de prototypage et

éventuellement de réduire les capacités en émulation.

La stratégie de déboguage proposée dans cette thèse permet de travailler en émulation sur un sous-ensemble du circuit. Economiquement parlant, puisque les plateformes de prototypage coûtent bien moins chères que les émulateurs, il est plus intéressant d'investir dans une plateforme de prototypage surdimensionnée en travaillant, grâce à l'interopérabilité, conjointement avec un émulateur de faible capacité plutôt que d'investir dans un émulateur de grande taille capable de supporter plusieurs SoCs en parallèle.

Pour conclure, recourir aux techniques d'interopérabilité pour la vérification d'un SoC remplit parfaitement ses objectifs mais, nécessite d'investir dans les plateformes de prototypage à de plus grande capacité afin de supporter l'augmentation de surface liée à l'instrumentation.

5.2) Perspectives

Le précédent paragraphe a mis en évidence la principale limitation de l'interopérabilité : l'augmentation de surface dans les plateformes de prototypage. Plusieurs idées sont envisageables pour atténuer, voire supprimer, cette limitation. De plus, d'autres améliorations ont été identifiées, certaines sont déjà en cours de réalisation. L'ensemble des perspectives vont être présentées dans les prochains paragraphes.

5.2.1) Extension de la bibliothèque de l'outil «InteroperabiltyCompile»

L'outil «InteroperabiltyCompile» est aujourd'hui lié au synthétiseur HdlIce de Cadence. Il serait bien de lever cette dépendance, ce qui implique que l'outil puisse travailler avec d'autres langages que Verilog et comprenne d'autres bibliothèques de primitives que la qtref de Palladium. Toutes les plateformes de prototypage travaillent avec le format Edif sur la bibliothèque de primitives Xilinx Virtex2 et/ou Virtex4. Il serait donc fortement utile que l'outil puisse également supporter ces formats, cela permettrait en particulier de déployer l'interopérabilité sur un ensemble simulation/prototypage sans nécessiter d'outils d'émulation.

5.2.2) Essais sur des cas d'application plus complexes

L'IP hls25 a permis de valider le flot d'interopérabilité et ses outils associés et a montré que les techniques d'échanges d'état proposées par cette thèse fonctionnent correctement. Néanmoins, ce circuit et son banc de tests sont d'une faible complexité (une seule horloge, pas de dépendances externes). Il serait donc intéressant de valider l'interopérabilité sur des cas plus complexes, couvrant l'ensemble des difficultés fréquement rencontrées et théoriquement supportées.

De plus, les propriétés de hls25 au niveau de la taille des mémoires et de la proportion entre éléments logiques et registres sont assez rares. Ainsi, la mesure de l'impact en surface de l'interopérabilité correspond probablement au pire cas. Il faudrait réaliser des mesures sur des circuits aux propriétés plus classiques afin de déterminer le coût moyen de cette technique.

Enfin, la vérification de «hls25» n'a détecté aucune erreur matérielle. Il n'a donc pas été possible de mesurer directement l'efficacité de la méthode de déboguage proposée et qu'elle n'a pas apporté dans ce cas le gain de productivité escompté. C'est pourquoi, un autre cas d'application et un déployement serait souhaitable.

5.2.3) Exploitation des possibilités offertes par les FPGAs Xilinx

L'expérience réalisée avec l'IP hls25 prototypée sur une machine ZeBu-XL a montré que l'interopérabilité peut, dans certain cas, faire tripler la taille du système. Pour vérifier une IP qui, en général, est assez petite, ce point n'engendre pas de problème vu la forte capacité des machines de

prototypage. On constate juste une légère hausse de la durée de synthèse, placement et routage.

Par contre, dans le cas d'un système monopuce (SoC) de plusieurs dizaines de millions de portes utilisant sans instrumentation quasi toutes les ressources de la machine, cette augmentation de taille peut se révéler catastrophique au point de ne pas pouvoir déployer l'interopérabilité.

Néanmoins, une solution semble être envisageable. En effet, l'ensemble des plateformes de prototypage est basé sur des FPGAs Xilinx qui offrent une possibilité de «readback» dynamique des registres. Cela signifie, que ces FPGAs permettent de capturer dynamiquement l'état des registres mais, malheureusement, pas celui des mémoires. Eve, avec ses machines ZeBu, exploitent déjà cette technologie. D'autres compagnies semblent suivre le même axe. Ainsi, il semble intéressant de travailler sur un outil capable de convertir le bitstream Xilinx en une sauvegarde d'état. De même, afin de restaurer un état, il sera intéressant d'avoir un outil capable de modifier les fichiers de configurations des FPGAs en fonction d'un fichier de sauvegarde d'état à charger. Ainsi, pour rendre une plateforme de prototypage interopérable, il suffirait d'instrumenter seulement les mémoires au lieu des mémoires et registres.

5.2.4) <u>Gestion de plusieurs d'horloges indépendantes</u>

Une hypothèse forte de cette thèse est de n'utiliser que des horloges conformes à SceMi, ce qui signifie que toutes les horloges sont générées à partir de l'horloge «uclock». Cette hypothèse permet de couvrir une grande partie de la vérification.

Néanmoins, cette technique n'assure pas une vérification complète. En effet, une partie des erreurs matérielles est contenue dans les interfaces entre domaines d'horloges. Afin de mettre en évidence ces erreurs, il faut parfois utiliser des horloges complétement indépendantes entre elles. De telles horloges ne sont pas conformes à SceMi, l'interopérabilité ne peut donc pas être actuellement déployée sur ce genre de plateformes.

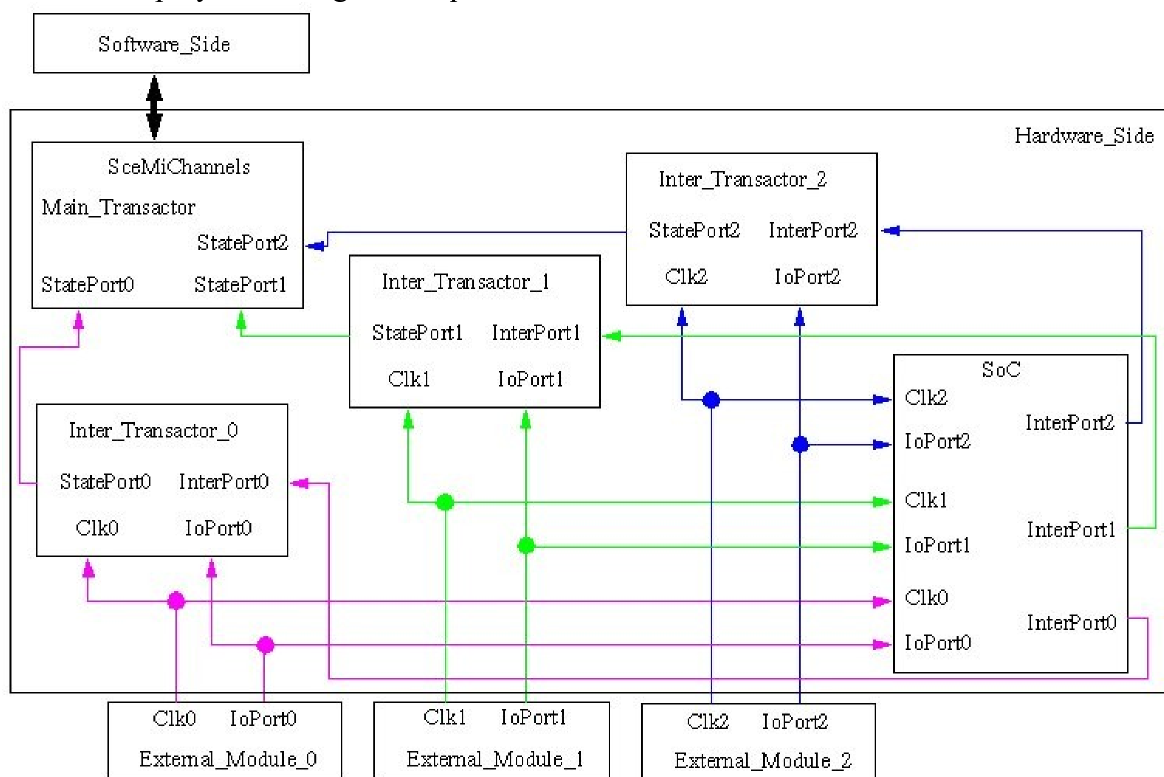


Figure 41 : Architecture d'une plateforme interopérable à plusieurs horloges indépendantes

Par contre, si on considère chaque domaine d'horloge séparement, toutes les horloges d'un

même domaine sont conformes à SceMi. Avec la technique d'instrumentation proposée dans cette thèse, on peut capturer l'état d'un sous-ensemble du circuit. Ainsi, il serait intéressant de modifier l'outil «InteroperabiltyCompile» de manière à ce qu'il puisse gérer plusieurs domaines d'horloge SceMi indépendants. L'interopérabilité permettrait alors de reproduire le comportement d'un sous-ensemble du circuit fonctionnant sur le même domaine d'horloge.

La mise en oeuvre d'une telle plateforme (figure 41) implique l'utilisation d'un transacteur d'interopérabilité par domaine d'horloge et, pour chaque domaine, d'échantillonner tous les signaux en provenance de l'extérieur ou d'un autre domaine. D'autre part, le lien de communication SceMi entre émulateur et ordinateur fonctionne obligatoirement avec une seule et unique horloge «uclock». Ainsi, si plusieurs transacteurs fonctionnant sur des horloges indépendantes doivent être mis en oeuvre, un seul poura être connecté à l'ordinateur. Cette limitation impose de définir une hiérarchie dans les transacteurs. On trouvera donc un transacteur «principal» en charge des communications avec l'ordinateur et communiquant avec l'ensemble des transacteurs d'interopérabilité.

5.2.5) <u>Contrôleur mémoire partagé</u>

L'instrumentation de l'IP hls25 a multiplié par six la taille des mémoires, les contrôleurs mémoires représentant la principale contribution de cette augmentation de surface. La section «4.5.3) Plateforme Palladium II» a montré l'intérêt de développer un contrôleur mémoire partagé entre l'ensemble des mémoires du même genre, comme cela est fait avec les BISTs. Dans le cas de hls25, une mémoire est utilisée cinquante fois et, le contrôleur associé à cette mémoire est plus gros que la mémoire. En économisant le coût de quarante neuf contrôleurs, l'impact de l'interopérabilité sur la surface aurait été bien plus faible.

Néanmoins, l'idée initiale d'utiliser un contrôleur mémoire par mémoire garantit que l'interopérabilité ajoutera au plus six signaux à router entre plusieurs FPGAs lorsque l'on va utiliser une plateforme de prototypage. Cet aspect est très important car si on augmente considérablement ce nombre de signaux, on va obligatoirement augmenter le facteur de multiplexage et donc affaiblir la fréquence du circuit prototypé. Les mémoires d'un même type peuvent être réparties sur plusieurs FPGAs. Ainsi, en utilisant un contrôleur partagé, on risque d'augmenter fortement le nombre de signaux inter-FPGAs.

Pour pallier à cet inconvénient, on pourrait envisager d'utiliser les bus des BISTs. Cependant, en émulation/prototypage, la fonctionnalité de ces modules n'est jamais vérifiée, les signaux de contrôle des BISTs sont collés à zéro et les synthétiseurs suppriment souvent ces modules par simplification. Ainsi, bien que présent au niveau RTL, ces bus ne se retrouveront pas sur les plateformes d'émulation/prototypage et ne seront donc pas exploitables.

Au final, la solution pertinente consiste au développement d'un contrôleur mémoire partagé qui serait utilisé dans chaque FPGA intégrant au moins une mémoire cible. En émulation, il n'y a pas de contrainte de partitionnement, on utiliserait donc un seul et unique contrôleur par type de mémoire. Par contre, la mise en oeuvre de cette stratégie nécessite un traitement post-partitionnement. En général, les FPGAs sont rarement remplis au dela de 70% ; il devrait donc être possible de rajouter un petit contrôleur mémoire. La principale difficulté sera dans l'automatisation du traitement post-partitionnement.

Bibliographie

[APT06] http://www.aptix.com/products/overview.htm

[BIG04] Bigot A., Charpentier F., Krupnova H., Sans I. - «Deploying Hardware Platforms for SoC Validation : An Industrial Case Study» - IEEE Transactions on Field Programmable Logic and its applications (FPL), 2004

[CAD06a] http://www.cadence.com/products/functional_ver/specman_elite/index.aspx

[CAD06b] http://www.cadence.com/products/functional_ver/acel_emul/index.aspx

[CHA99] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd - «Surviving the SoC Revolution – A Guide to Platform-Based Design» - Kluwer Academic Publisher, 1999

[CHA01] A. Chandra, K. Chakrabarty - «System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Golomb Codes» - IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems - Vol20, No3, 03/2001, Pages 355 - 367

[CHA03] A. Chandra, K. Chakrabarty - «A Unified Approach to Reduce SOC Test Data Volume, Scan Power and Testing Time» - IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems - Vol22, No3, 03/2003, Pages 352 - 362

[EVA03] Evans Data Corporation - «Embedded Systems Development Survey» - Volume 1, 2003, http://www.evansdata.com/n2c/surveys/embedded_toc_2003_2.shtml

[EVE06] http://www.eve-team.com

[FLE06] http://www.flexody.com

[GHE05] Frank Ghenassia, Livre «Transaction-Level Modeling with SystemC : TLM Concepts and Applications for Embedded Systems»

[HAR06] http://www.hardi.com/haps/haps.htm

[INT06] http://www.intel.com/technology/silicon/nanotechnology.htm

[KIR98] D. Kirovski, M. Potkonjak, L.M. Guerra, «Functional Debugging of Systems-On-Chip» - IEEE International Conference on Computer-Aided Design (ICCAD), 1998

[KIR99] D. Kirovski, M. Potkonjak, L.M. Guerra, «Improving the Observability and Controllability of Datapaths for Emulation-Based Debugging», IEEE Transactions on Computer-Aided Design of Integrated Circuits ans Systems, Vol18, No 11, November 1999

[KIR00] D. Kirovski, M. Potkonjak, L.M. Guerra - «Cut-Based Functional Debugging for Programmable Systems-om-Chip», IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol8, No 1, February 2000

[KUD01] M. Kudlugi, S. Hassoun, C. Selvidge, D. Pryor - «A Transaction-Based Unified Simulation/Emulation Architecture for Functional Verification» - IEEE Transactions on Design Automation Conference (DAC), 2001, pages 623 – 628

[LAR04] Cédric Lardière - «Système d'émulation et d'accélération : l'habit ne fait plus le moine» - Electronique Internationnal, N576 – 28 Octobre 2004, pages 27

[LUD04] Ludewig R., Hollstein T., Schutz F., Glesner M. - «Rapid Prototyping of an integrated Testing and Debugging Unit» - Proceedings of the 15th IEEE International Workshop on Rapid System Prototyping (RSP), 2004

[MEN06] http://www.mentor.com/products/fv/emulation/

[NOV06] http://www.novas.com/Siloti_Visibility_Enhancement.html

[PAP03] N. Papendreou, M. Varsamou, T. Antonakopoulos - «xDSL Systems Prototyping using a Flexible Emulation Environment» - Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping (RSP), 2003

[PET00] G.D. Peterson, «Predicting the performance of SoC verification technologies», VHDL International Users Forum Fall Workshop, 2000. Proceedings, 18-20 Oct, pages 17-24

[PET01] G.D. Peterson, «Designing the design process : applying performance evaluation to verification technologies», SoutheastCon 2001. Proceedings. IEEE , 30 March-1 April 2001, pages 21-28

[POT95] Potkonak M., Dey S., Wakabayashi K. - «Design-For-Debugging of Application Specific Designs» - IEEE International Conference on Computer-Aided Design (ICCAD), 1995

[PRO06] http://www.prodesign-europe.com/ce/CHIPitPlatinumEditionV4.htm

[RIZ03] Lauro Rizzatti - «Choosing an emulation tool» http://www.eetimes.fr/bus/news/showArticle.jhtml?articleID=171202288

[SCE04] «Standard Co-Emulation Modeling Interface (SCE-MI) Reference manual v1.0.8», http://www.verilog.org/itc/, 2004

[SCH03] HJ. Schlebusch, G. Smith, D. Sciuto. D. Gajski, C. Mielenz, CK. Lennard, F. Ghenassia, S. Swan, J. Kunkel - «Transaction based design : Another Buzzword or the Solution to a Design Probelm ?», Proceedings of the Design Automation and Test in Europe Conference and Exhibition, DATE, 2003

[SIR04] Siripokarpirom R., Mayer-Lindenberg F. - «Hardware-Assisted Simulation and Evaluation of IP Cores Using FPGA-based Rapid Prototyping Boards» - Proceedings of the 15th IEEE International Workshop on Rapid System Prototyping, 2004

[URA05] P. Urard, L. Paumier, P. Georgelin , T. Michel , V. Lebars , E. Yeo , B. Gupta - «A 135Mbps DVB-S2 Compliant Codec Based on 64800-bit LDPC and BCH Codes» - Proceedings of the 42nd annual conference on Design automation (DAC), 2005

[TOR02] Torre E., Garcia M., Riesgo T., Torroja Y., Uceda J. - «Non-intrusive debbugging using the JTAG interface of FPGA-based prototypes» - IEEE International Symposium. on Industrial Electronics (ISIE), 2002

[STM06] http://www.st.com/stonline/products/literature/bd/12071/stn8810.htm

[SYN06] http://www.synplicity.com/products/certify/

[XIL06] http://direct.xilinx.com/bvdocs/appnotes/xapp138.pdf

Résumé

Cette thèse introduit un nouveau concept dans la vérification des circuits au niveau RTL : l'interoperabilité entre simulateurs HDL, émulateurs matériel et, plateformes de prototypage. Cette thèse permet de bénéficier, dans les processus de vérification du RTL, à la fois de la capacité de déboguage des émulateurs et simulateurs HDL et de l'excellente vitesse d'exécution des plateformes de prototypage. Afin d'atteindre cet objectif, la notion d'état d'un circuit est introduite. Cette thèse présente des outils permettant d'intégrer l'interopérabilité comme une fonctionnalité des circuits à vérifier. Cela permet de rendre interopérable entre elles toutes les machines de vérification travaillant au niveau RTL. L'idée principale de l'interopérabilité consiste en la réalisation des tests sur plateforme de prototypage rapide tout en réalisant périodiquement des sauvegardes d'état. Lorsqu'une erreur apparait, on réalise le déboguage du circuit sur un émulateur rapide, ou sur un simulateur HDL économique. Le test sera exécuté en partant de la dernière sauvegarde précédent l'instant d'apparition du problème. De plus, cette thèse propose une technique permettant de concentrer l'effort de déboguage sur le sous-ensemble défectueux ce qui, d'une part, permet d'augmenter la vitesse sur simulateur et d'autre part, réduit le nombre de domaines nécessaires sur émulateur multi-domaines et augmente ainsi le nombre de déboguages simultanés.

Mots clés

Emulation matérielle, prototypage matériel, simulation HDL, interopérabilité, SceMi, état d'un circuit, captures et restaurations d'états, vérification RTL.

Abstract

This thesis defines a new concept in RTL verification : interoperability between HDL simulators, hardware emulators and hardware prototyping platforms. The main purpose is to benefit from both good speed of hardware prototyping platforms and debug capabilities of hardware emulators and HDL simulators. To achieve this purpose, this thesis introduces the notion of design state. Then, a tool dedicated to interoperability is presented. This tool add interoperability to design functionnalities. This make all machines working at RTL level interoperables with each others. Tha main idea of interoperability is to lunch tests on fast prototyping platforms while periodically saving design state. When a bug will be faced, debug will be performed using a fast emulator or a low cost HDL simulator. The test will restart from the last saved database done just before bug time. Finally, this thesis also introduce a method to focalize debug effort only on the sub-module containing the bug. This improve HDL simulation speed and reduce number of required emulator domains which allow to perform more debugs in the same time using the same emulator.

Key words

Hardware emulation, hardware prototyping, RTL simulation, RTL verification, interoperability, SceMi, design state, save and restore.

Adresses du laboratoire et de l'entreprise

Laboratoire TIMA, 46 Avenur Félix Viallet, 38031 Grenoble cedex, France
STMicroelectronics, 12, rue Jules Horowitz, BP217 F-38019 Grenoble cedex, France

ISBN: 978-2-84813-103-0