Etat de l'art associé au projet tutoré :

Développement d'un System-On-Chip open-source

DUZAN Luc

FARACHE Gabriel

LONGO Matthieu

MICHAUD Clément

duzan@etud.insa-toulouse.fr
farache@etud.insa-toulouse.fr
longo@etud.insa-toulouse.fr
cmichaud@etud.insa-toulouse.fr

4IR 2012-2013

Résumé du projet

Dans notre projet tutoré, nous allons nous intéresser plus particulièrement au *Milkymist* SoC, qui est un SoC *open-source*, c'est à dire que le code HDL qui le décrit est libre de droit. Ce SoC a été conçu pour être programmé sur un FPGA spécifique le Virtex-4 XC4VLX25. Notre but est d'adapter le code de ce SoC pour le rendre compatible avec le Xilinx Spartan6 qui est utilisé à l'INSA de Toulouse. Nous allons aussi ajouter un module à *Milkymist* permettant d'exploiter la propriété de certains FPGA qui est de pouvoir être reprogrammé à la volée. C'est-à-dire que nous allons permettre au système *Milkymist* de pouvoir ajouter à chaud (sans redémarrer le système) des périphériques (à partir de leurs descriptions structurelles) sur le FPGA et de les connecter au SoC.

Table des matières

Introduction

Les systèmes électroniques numériques sont constitués de bascules (permettant de sauvegarder des états logiques) et de portes logiques (et, ou, non, ou exclusif par exemple) interconnectés entre elles. Ces portes sont elles-mêmes constituées de transistors qui sont donc les briques élémentaires de l'électronique numérique. En effet, elles peuvent être utilisées comme de minuscules interrupteurs pilotés. Il est possible de graver un enchevêtrement complexe de transistors sur une seule plaque de silicium, on parle de circuit intégré. Concevoir un circuit intégré consiste alors à choisir des portes logiques et des bascules et à définir des connexions entre elles afin d'obtenir le comportement désiré. Les portes logiques sont aussi décrites par des réseaux de transistors. Au final, un circuit intégré est construit en gravant un complexe réseau de transistors sur une seule plaque de silicium.

Historiquement ce réseau complexe était dessiné à la main sur des feuilles spéciales (feuilles de mylar) ce qui limitait fortement la complexité des circuits intégrés. De plus, il n'était pas possible de tester la conception sans la graver. La montée en puissance des ordinateurs a alors permis la mise en place de langages de description de matériel (appelés HDL pour *hardware description language*). Les HDL permettent de modéliser le comportement des circuits logiques et également de décrire les structures de ces circuits logiques, on peut alors simuler le comportement de la structure décrite et vérifier qu'elle se comporte de manière conforme au modèle. Il est souvent possible de déduire de manière automatique la description structurelle d'un circuit à l'aide de sa description comportementale. L'apparition de ses langages a alors révolutionné le monde de la conception des circuits intégrés. Les concepteurs décrivent alors leurs circuits grâce aux HDL, les simulent de manière virtuelle et génèrent automatiquement à l'aide de synthétiseurs les dessins physiques qui représentent le circuit intégré, ces dessins appelés netlists sont ensuite envoyés à une fonderie qui s'occupe alors de la fabrication. Une fonderie est une entreprise spécialisée dans la fabrication de circuits. Ces nouvelles techniques de production permettent alors de produire des circuits intégrés spécifiques à des attentes particulières plutôt que d'utiliser des microcontrôleurs et de les programmer afin d'obtenir le comportement attendu, il s'agit d'ASIC pour *Application Specific Integrated Circuit*. Cependant la conception et surtout la fabrication d'ASIC ne peuvent être rentabilisées que pour de gros volumes de production, de plus une erreur de conception sur un ASIC peut coûter très chère. Mais l'apparition des FPGA (pour *Field Pro-*

grammable Gate Array), qui sont des circuits intégrés logiques reprogrammables après leur conception, marque encore une fois une révolution. Ils permettent de tester de manière plus poussée la conception d'ASIC en créant des prototypes et peuvent même être vendus une fois programmés à la place d'ASIC pour des petits volumes de production. Les FPGA de nos jours sont assez performants pour contenir l'intégralité des composants nécessaires au fonctionnement d'un système complet (microprocesseur, mémoire, GPIO, ...), on parle alors de SoC. Les SoC (*Systen On Chip*) désignent un système complet qui est contenu sur une seule puce que ce soit un FPGA ou un ASIC.

Dans notre projet tutoré, nous allons nous intéresser plus particulièrement au *Milkymist* SoC, qui est un SoC *open-source*, c'est à dire que le code HDL qui le décrit est libre de droit. Ce SoC a été conçu pour être programmé sur un FPGA spécifique le Virtex-4 XC4VLX25. Notre but est d'adapter le code de ce SoC pour le rendre compatible avec le Xilinx Spartan6 qui est utilisé à l'INSA de Toulouse. Nous allons aussi ajouter un module à *Milkymist* permettant d'exploiter la propriété de certains FPGA qui est de pouvoir être reprogrammé à la volée. C'est-à-dire que nous allons permettre au système *Milkymist* de pouvoir ajouter à chaud (sans redémarrer le système) des périphériques (à partir de leurs descriptions structurelles) sur le FPGA et de les connecter au SoC.

Ce rapport ayant pour but de clarifier le travail de notre projet tutoré, nous allons dans un premier temps détailler la conception des circuits intégrés. Pour cela, nous parlerons dans un premier temps des supports pouvant recevoir des circuits intégrés (FPGA et ASIC), nous parlerons ensuite de la conception du matériel à l'aide des langages de description de matériel tel que VHDL, Verilog et Migen et nous ferons un état sur l'avancement de la communauté *open-source* dans ce domaine. Dans la deuxième partie, nous parlerons ensuite du projet *Milkymist* et de l'intention de notre contribution sur ce projet.

Chapitre 1

Conception du matériel

1.1 Langage de description de circuits logiques

1.1.1 Présentation de Verilog et VHDL

Verilog et VHDL (pour *Very high speed integrated circuit Hardware Description Language*) sont tous deux des Langages de Description de Matériel (en anglais, *Hardware Description Language* ou HDL). Les HDL utilisent une méthode de description de flux de processus résultant d'un modèle de flux de données avec des informations de synchronisation (le temps). Cette méthode consiste en une abstraction au niveau des portes logiques et des transitors. Cette méthode s'appelle *Register-Transfer Level* ou RTL et a été définie à cause de l'explosion de la complexité des circuits électroniques depuis les années 1970 (loi de Moore). En effet, les concepteurs de matériel micro-électronique avaient besoin d'une méthode de description logique des matériels numériques de plus haut niveau pour limiter la complexité de la conception et cela sans que cette dernière ne soit spécifique à une technologie en particulier. Les HDL utilisent donc cette méthode pour représenter des circuits à un niveau élevé. À partir de cette représentation des circuits, des représentations de plus bas niveaux et le câblage peuvent être déduits. Les HDL permettent donc de décrire un circuit électronique tant au niveau comportemental que structurel, c'est-à-dire les flux de signaux transitant entre les différents registres. C'est pour cette simplification dans la conception et l'utilisation des matériels que les HDL sont largement utilisés dans l'industrie.

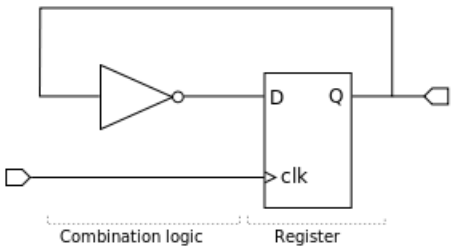

FIGURE 1.1 – Bascule D

La figure 1.1 ci-dessus montre comment est représenté un circuit. Nous constatons qu'il est ici totalement fait abstraction de la manière avec laquelle ce dernier est câblé. Nous ne connaissons que ce qui entre et ce qui sort de ce dernier. Un HDL décrira, quant à lui, le « comportement » externe, général du circuit.

La principale différence entre les HDL avec les langages de programmations traditionnels est qu'avec un HDL, contrairement aux autres types de langages procéduraux, nous pouvons modéliser de multiples processus parallèles et exprimer le temps de manière explicite. En effet, ces deux éléments (le parallélisme et le temps) sont les principaux attributs d'un matériel électronique. Un changement dans un des processus pourra donc déclencher une mise à jour dans les autres processus.

1.1.2 Les autres HDL existants

Il existe de nombreux HDL autres que les deux plus connus que sont Verilog et VHDL, tel que, Altera HDL (AHDL) qui est un langage propriétaire d'**Altera**, Hydra qui est basé sur **Haskell**, MyHDL qui est basé sur **Python** ou encore RHDL basé sur **Ruby**. Un autre HDL est SystemC, qui est, avec Verilog et VHDL, un des HDL les plus utilisés dans l'industrie. Cependant, ce dernier n'est pas un langage à part entière puisque c'est en fait un ensemble de classes **C++** fournissant les outils nécessaires à la modélisation du matériel. *Milkymist* possède aussi son propre langage Migen qui est toujours en développement, nous y reviendrons dans une sous-section ultérieure.

1.1.3 Verilog VS VHDL

Nous allons vous présenter plus en détail les deux principaux HDL que sont VHDL et Verilog. Chacun a ses atouts :

– VHDL :

  - ce langage a été conçu pour aider la conception et la spécification au niveau d'un système électronique,
  - il est plus souple que Verilog car permet, entre autre, à l'utilisateur de définir ses types, ses configurations, etc.
- Verilog :
  - conçu de base pour les concepteurs de matériel développant des FPGAs et ASICs
  - parfait pour convertir des types de données de vecteurs de bits vers des notations arithmétiques,
  - existence de supports compréhensibles pour la conception numérique de bas niveau.

Gateway Design Automation Inc. puis rendu disponible au grand public en 1990 par **Cadence**. Ce sont là les principales différences entre les deux langages. Dans les faits, VHDL et Verilog sont similaires puisqu'ils sont tous les deux des standards IEEE : *IEEE standard 1076-1987* [27] en 1987 puis *IEEE standard 1076-1993* [28] en 1993 après une mise à jour pour VHDL. Il a ensuite été mis à jour régulièrement pour arriver, en 2008 au standard *IEEE 1076* [29] (VHDL 4.0) qui est la dernière version en date. Quant à Verilog, il est devenu un standard en 1995 : *IEEE standard 1364-1995* [30] et a été mis à jour deux fois depuis : en 2001 *IEEE Standard 1364-2001* [31] (Verilog 2001) qui corrigera de nombreux problèmes puis en 2005 *IEEE Standard 1364-2005* [32] (Verilog 2005) qui corrigea quelques bogues. Historiquement, VHDL a été développé pour l'armée de l'air des États-Unis d'Amérique (contrat *F33615-83-C-1003*) par *Intermetrics, Inc.*, *Texas Instruments* et *IBM*. *Intermetrics, Inc.* fournissait les experts en langage (de programmation) et était aussi le maître d'œuvre du projet. *Texas Instruments* se concentrait sur la partie expertise en conception de puces. *IBM* quant à lui fournissait les experts en conception de systèmes informatiques. C'était donc un projet d'une certaine envergure (comme *ADA* par exemple). Verilog a quant à lui été lancé en 1984 par **Gateway Design Automation Inc.** puis rendu disponible au grand public en 1990 par **Cadence**.

1.1.4 Le HDL de Milkymist : Migen

Nous vous avions aussi parlé de Migen, le langage de *Milkymist*. Dans le futur, le SoC *Milkymist* utilisera ce langage. Le projet Migen (pour ***Mi**lkymist **gen**erator*) a été lancé en 2011 et est toujours en cours de de développement. Cet outil est fondé sur le langage **Python** et a pour but d'automatiser encore plus le processus de conception des VLSI (*Very Large Scale Integration*, c'est un processus de création de circuits intégrés). Grâce à Migen, il sera possible d'appliquer des concepts modernes tels que l'objet et la métaprogrammation [1] pour concevoir un matériel. Le résultat est plus élégant et surtout plus facile à faire évoluer, tout en réduisant les problèmes dûs aux erreurs humaines. Ce sont là les grands principes que *Milkymist* veut mettre en place dans Migen. En se

1. La méta-programmation c'est la création de programmes qui utilisent d'autres programmes pour fonctionner. Ces autres programmes sont donc les données manipulées par le méta-programme créé.

basant sur ces dernières, Migen fournira les outils nécessaires pour construire des conceptions synchrones de manière plus productive en automatisant certaines tâches comme réinitialiser les registres et en faisant abstraction du paradigme des HDL fondés sur l'évènement (le parallélisme). Migen permettra aussi d'intégrer le SoC (*System-on-Chip*) en automatisant par exemple l'interconnexion entre les puces. Migen accélèrera aussi la conception matérielle grâce au paradigme du flux de données avec une intégration semi-automatique dans le SoC.

Migen semble donc un projet très intéressant puisqu'il rend la conception plus simple et qu'il rend plus efficace le matériel. Cependant, ce projet est encore jeune et encore en développement. Toutes les fonctionnalités ne sont pas encore implémentées. Nous ne l'utiliserons donc pas dans notre projet pour éviter tout problème.

1.2 Technologie de fabrication

1.2.1 ASIC

Netlist

Une fois la conception réalisée à l'aide de langages de description matérielle, on peut obtenir une netlist du circuit intégré. Cette netlist contient dans un premier temps une liste qui définit les composants utilisés dans ce circuit intégré. Chaque composant est défini par les différents points de connexion (appelé *ports*) qu'il possède et par ses propriétés élémentaires. Un composant peut être quelque chose d'élémentaire comme un transistor ou une résistance tout comme un circuit intégré complexe. Ensuite la netlist va alors décrire des connexions entre des composants. Chaque composant décrit dans le réseau est forcément un des composants de la liste précédemment décrite, on dit alors que ce composant est une instance de la définition donnée dans la liste de composants. Les connexions sont décrites par une liste de fils. Un fil connecte un port donné d'une instance avec un autre port donné d'une autre instance. Pour limiter la redondance, on peut hiérarchiser les netlists, c'est à dire que l'on peut ranger une partie d'un réseau dans une définition d'un composant. De cette manière, si cette partie est répétée plusieurs fois dans le réseau, on peut l'insérer comme on insérerait un composant élémentaire, et la description de cette partie n'est alors donnée qu'une seule fois.

Fabrication d'ASIC

FIGURE 1.2 – Waffer comportant plusieurs ASIC (image récupérée sur http://tes-dst.com/tes-dst/services/asic-design-service/asic-design-library-a-ip.html)

À partir de la description en netlist, il est possible d'obtenir un fichier (généralement au format GDSII) qui représente le dessin physique du circuit intégré. C'est-à-dire qu'il représente entre autre les formes géométriques à graver sur le silicium. C'est ce fichier qui est envoyé aux fondeurs pour obtenir les masques

nécessaires à la gravure du système. Ce procédé utilisant le plus souvent la lithographie est très complexe et ne sera pas détaillé dans ce rapport. La fabrication d'ASIC demande un fort investissement en matériel, c'est pour cela qu'on observe une séparation entre les entreprises qui conçoivent les circuits intégrés et les fonderies car il faut une très grande production pour amortir les investissements. Seules quelques entreprises telles qu'Intel, STMicroelectronics, Texas Instruments qui produisent de très grands volumes peuvent se permettre de faire en même temps la conception et la réalisation de leurs circuits. Les ASIC ne peuvent alors être réalisés pour un prix raisonnable en petit volume.

1.2.2 FPGA

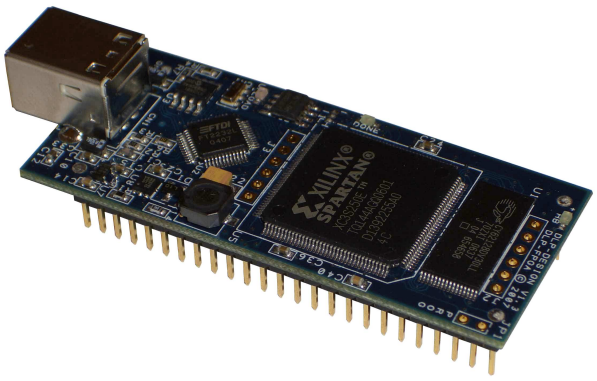

FIGURE 1.3 – Une carte de developpement équipé d'un FPGA (image récupérée sur http://www.dlpdesign.com/fpga/fpga.shtml)

Les FPGA (*Field Programmable Gate Array*) sont des circuits logiques programmables. En effet, *Field Programmage* signifie programmable sur le champ, le circuit peut être programmé après sa fabrication pour réaliser n'importe quelle fonction logique. La configuration d'un FPGA est obtenue à l'aide de langages HDL qui permettent aussi de concevoir les ASIC, c'est pour cela que les FPGA peuvent être utilisés pour faire des prototypes d'ASIC.

Les FPGA sont constitués d'un nombre très importants de blocs logiques rangés en matrice (d'où le terme *Gate Array* de FPGA). Ces blocs logiques peuvent être configurés pour effectuer des fonctions logiques élémentaires (et, ou, ou exclusif, non) ou quelques fonctions logiques plus complexes. Ils peuvent aussi être configurés en bascules ou en d'autres éléments permettant de mémoriser de l'information. Ces bloques logiques peuvent également être connectés de manière arbitraire entre eux. C'est en définissant les fonctions réalisées par ces blocs logiques et des connexions entre eux que l'on peut configurer un FPGA pour obtenir le fonctionnement attendu. Cette configuration est générée de manière

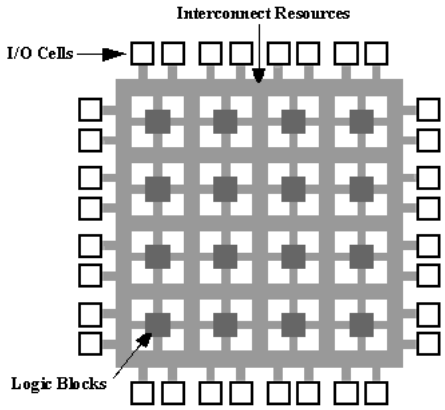

FIGURE 1.4 – Schéma représentant la structure en matrice des portes d'un FPGA (image récupérée sur http://proxacutor.free.fr/)

automatique à partir de description en HDL. De nos jours, le nombre de blocs logiques sur les FPGA se compte en millions.

Les cartes FPGA contiennent en plus du réseau de blocs logiques programmables des périphériques. Cela permet d'économiser des portes. De plus certains périphériques tels que les GPIO ne peuvent pas être programmés à l'aide de blocs logiques. Il n'est donc pas rare de voir des cartes FPGA embarquer des GPIO, de la RAM, des cartes réseaux. Dans notre projet tutoré, la différence des périphériques de la Nexys3 utilisée à l'INSA et ceux de *Milkymist* entrainent des problèmes de portabilité.

1.3 Le monde de l'*open source* et son avancement sur les FPGA

Dans cette partie, nous allons parler des matériels *open-source*. La conception est effectuée de manière communautaire ou par des sociétés privées. Elles ne sont pas soumises à des brevets ultra-restricitfs mais le plus souvent sous licence GPL (General Public License) plus souples.

Le matériel *open-source* est très peu developpé contrairement à son homologue logiciel. Il n'y a que peu d'acteurs sur cette scène alors qu'il y en a des dizaines pour l'*open-source* logiciel. Cependant, ce pan de l'*open-source* tend de plus

en plus à se développer ces derniers temps. Pour preuve, en 2010 a été publié l'**Open Source Hardware** (**OSHW** qui définit ce qu'est un matériel *open-source* et les idées derrière ce mouvement.

Il y a d'autres projets en cours, que ce soit dans le domaine des ordinateurs eux-mêmes, dans leur composants (CPU (*Central Processing Unit*, le coeur de la machine), carte graphique, ...), voire même dans d'autres secteurs tels que la téléphonie et la géolocalisation. Voici la liste des projets *open-source* relatifs aux ordinateurs : *Arduino*, *Beagle Board*, *Bug Labs*, *CuBox*, *Ethernut*, *GP32*, *Maple*, *PLAICE*, *Raspberry Pi*, *Simputer*, *Pandora*, *PC532*[2] qui date des années 1990, *Minimig* qui est une nouvelle implémentation de l'Amiga 500 utilisant des FPGA, *Milkymist One* qui est un SoC évolué implémentant tous les principes logiques à travers des fichiers de conception créés avec *Verilog*. Ces fichiers sont *open-source*, donc disponibles pour tout le monde. Dans la plupart des autres matériels *open-source*, ces fichiers sont confidentiels et propriétaires.

Nous pouvons lister différents projets : *Amber*, *LEON*, *OpenCores*, *OpenSPARC*, *OpenRISC* qui est plus un groupe de développeurs visant à créer des CPU RISC ayant des performances très importantes. Enfin, le *Milkymist SoC*, couplé au micro-processeur *LatticeMico32*, qui possède un ensemble de périphériques développés de manière indépendante et *open-source*.

La plus grande communauté à l'heure actuelle est *OpenCores* (`http://www.opencores.org`). Fondée en 1999 par Damjan Lampret, *OpenCores* a developpé *OpenRISC 1000*. Le développement s'est poursuivi jusqu'en 2007. Durant cette période *OpenCores* a été supporté par *Flextronics*. À partir de 2007, *OpenRISC 1000* est passé dans une phase de déploiement commercial grâce à *ORSoC AB*. En mars 2012, il y avait plus de 147 000 utilisateurs enregistrés et plus de 900 projets en cours sur cette architecture matérielle. *ORPSoC* est l'implémentation de référence sur cette architecture. Cependant, le *Raspberry Pi*, l'ordinateur à 29$, sorti courant 2012, grandit de plus en plus. De nombreuses cartes électroniques complémentaire au *Raspberry Pi* sont sorties, toutes à un prix raisonnable.

Il faut savoir que contrairement aux idées reçues, les entreprises qui font de l'*open-source hardware* ont de bons résultats. En effet, les 13 plus grosses sociétés qui ont ce genre d'activités ont toutes plus d'un million de dollars de chiffre d'affaire (en dollars et en 2010)[9] :

- Adafruit Industries : plus d'1 million
- Arduino : plus d'1 million
- BeagleBoard : 1 million
- Bug Labs : 1 million
- Chumby : plus d'1 million
- Dangerous Prototypes
- DIY Drones : 1 million
- Evil Mad Scientist Labs
- Liquidware : plus d'1 million

2. `http://en.wikipedia.org/w/index.php?title=List_of_open-source_hardware_projects&oldid=533515806`

- MakerBot Industries : 1 million
- Maker Shed : plus d'1 million
- Parallax : plus d'1 million
- Seeed Studios : 1 million
- Solarbotics : plus d'1 million
- SparkFun Electronics : 10 millions

Ces entreprises ne sont qu'un échantillon. Il en existe bien d'autres de ce type qui approchent ou dépassent le million de chiffre d'affaire. De plus, ces chiffres datent de 2010, on peut donc légitimement supposer que ces chiffres ont évolué de manière positive ces deux dernières années. Selon les prévisions pour 2015, le chiffre d'affaire cumulé de toutes les entreprises de matériel *open-source* devrait avoisiner le milliard de dollars.

Ce secteur a donc un avenir radieux qui s'offre à lui. De plus en plus de projets voient le jour et ceux déjà présents s'améliorent grâce à la contribution de personnes possédant ce genre de matériel car c'est bien là le leitmotiv de l'*open-source* : la progression par la participation.

Chapitre 2

Adaptation d'un *System-on-Chip* libre : *Milkymist*

2.1 Le projet *Milkymist*

Ce projet a pour origine le projet de fin d'étude de Sébastien Bourdeauducq [37] sur le design d'un System-on-Chip au Royal Institute of Technology de Stockholm en 2010. Plus précisément, il s'agissait de réaliser un SoC rapide et économe en ressources, basé sur FPGA et avec pour but principal de supporter une application de rendu d'effets video en temps réel. Ces effets peuvent être aussi bien réalisés avec un ordinateur normal. Cependant cette approche a certains inconvénients, un système embarqué a l'avantage d'être moins encombrant et plus facile à mettre en place. Le temps de démarrage et de configuration est réduit à quelques secondes contairament à un ordinateur normal qui doit d'abord lancer un système d'exploitation lourd tel que Windows et ensuite lancer un logiciel spécialisé permettant de créer des effets vidéo. De plus, on peut avoir des interfaces spécialisées qu'un PC normal n'a pas, à moins de les rajouter, ce qui peut se révéler assez cher.

Le SoC *Milkymist* est implémenté sur un FPGA et utilise le coeur LatticeMico32 (LM32)[38]. C'est un CPU 32-bit *big endian* RISC *open-source*. Le microprocesseur LM32 est assisté dans son travail par une unité de mappage de texture et par un coprocesseur programmable à virgule flottante qui est utilisé par le logiciel de synthèse video.

2.2 Le travail à réaliser

2.2.1 Portage sur le Nexys3

Un SoC est composé de blocs correspondant chacun à un module spécifique (CPU, interfaces RS232, USB, PCI, modules de traitement du signal, blocs réseaux ARM ou Ethernet, module d'encryption DES ou AES ...). Heureusement il n'est pas à chaque fois nécessaire de tout réadapter. La plupart des modules écrits pour un certain projet sont réutilisables dans d'autres, ils ont souvent été prévus pour une intégration *plug and play*. Chacun des blocs constitue un bloc Soft-IP (Soft Intellectual Property Block) en opposition aux Hard-IP Blocks (ASIC), généralement bien documenté afin de faciliter le travail de réintégration et dont la licence encadre ses conditions de réutilisation au sein d'un projet (licence propriétaire entrainant le paiement de droit ou licences opensources avec chacune leurs partucalarités). Les avantages de blocs IP sont nombreux, ils permettent de supprimer le temps d'écriture du bloc donc diminuer le temps jusqu'à la commercialisation, diminuer le nombre d'ingénieurs requis pour le projet donc diminuation des coûts et enfin rassurer les développeurs car la solution est déjà éprouvée.

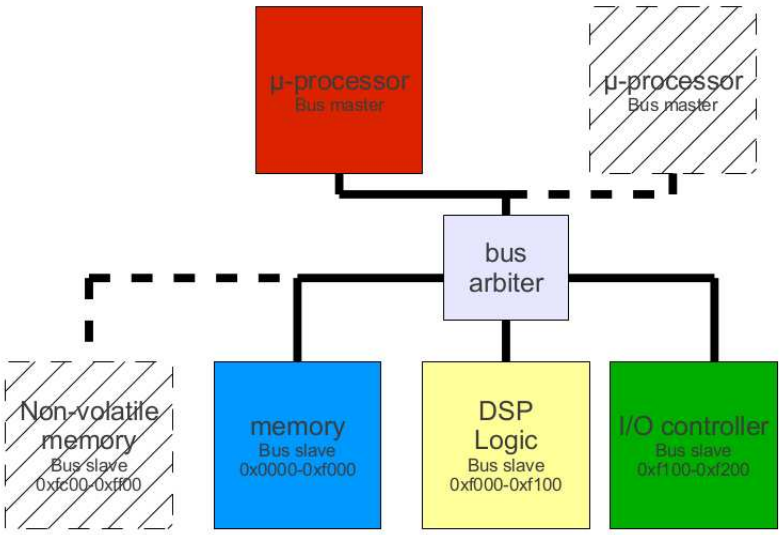

FIGURE 2.1 – Exemple d'implémentation de blocs IP dans un SoC

Seule la description du circuit (les interconnexions entre les différents modules et les entrées/sorties du FPGA) est écrite à chaque fois que l'on change de carte et/ou de FPGA. Cette description peut être codée soit en VHDL ou en Verilog tout comme la description des blocs. Il s'agit souvent d'un *memory-mapped* bus fournissant un accès aux registres de contrôle et aux mémoires. Dans notre

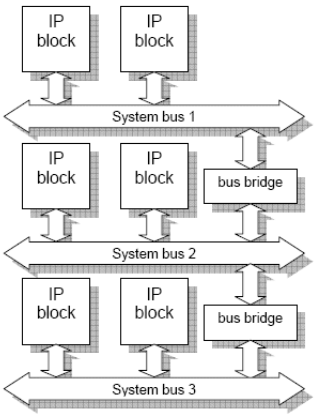

FIGURE 2.2 – Interconnexion des blocs IP par des bus

cas, c'est un CSR bus conçu par Sébastien Bourdeauducq, à l'origine du projet *Milkymist*.

Dans notre cas, nous allons réutiliser plusieurs modules déjà implémentés. Tout d'abord, l'IP-Core LatticeMico32 (architecture RISC) déjà intégré dans *Milkymist* et disponible gratuitement sous licence GNU GPL. Celui-ci nécessite tout de même quelques configurations de par sa généricité. On peut par exemple choisir selon ses besoins d'implémenter ou non, afin d'économiser des portes logiques, certaines parties dans l'unité de calcul (unité de multiplication, de décalage, activer les calculs signés, ...). On peut également changer la taille du bus, le nombre de registres, l'alimentation, le timing, la taille du cache, ... Le GPIO, l'USART et d'autres éléments seront aussi réutilisés et configuré tout comme le processeur selon nos besoins.

Dans cette partie, nous avons vu l'énorme utilité de la généricité dans la description des IPs. Cet aspect est souvent utilisé dans l'industrie afin de créer un hardware spécifique au besoin d'un projet. On peut gagner par cette méthode énormément en efficacité face aux architectures non-spécialisées, ce qui permet à résultat égal de travailler avec une fréquence plus basse et du coup de moins consommer d'énergie.

2.2.2 Adapter le système logiciel associé

Après avoir créé l'architecture qui correspond au cahier des charges du SoC, il faut encore adapter le système d'exploitation (*Operating System*) qui va le cadencer. C'est-à-dire qu'il faut récupérer un noyau existant souvent proche de l'architecture que l'on vient de réaliser et modifier le code pour qu'il soit entièrement compatible avec le nouveau matériel [34]. Il faut tout d'abord faire la différence entre un système d'exploitation classique que l'on trouve sur les PC actuels permettant le traitement de tâches multiples plus ou moins en même temps grâce au système du temps partagé (time sharing) et contenant une unité de translation des adresses physiques en adresses virtuelles (MMU) qui gère les tâches de manière avancée et possède souvent des appels système particuliers. En revanche les systèmes d'exploitation conçus pour les SoC sont souvent des systèmes temps réel (*Real Time OS*) qui effectuent une tâche en continu. En effet, ceux-ci n'ont en général qu'un seul but comme par exemple le traitement d'une vidéo dans le cas de la *Milkymist One*.

Le noyau d'un système d'exploitation étant entièrement dépendant de l'architecture du SoC, il faut généralement faire beaucoup de modifications dans le code que l'on veut porter. C'est pourquoi il est intéressant de sélectionner un noyau utilisé sur une architecture proche de la nouvelle pour limiter ces modifications qui en appellent souvent beaucoup d'autres. Concrètement, l'un des noyaux les plus utilisés pour le SoC est celui d'Unix. Les distributions fondées sur ce noyau et adaptés à ces systèmes sont appelés 'linux embedded' pour embarqué. Le noyau qui se trouve à la base de toutes les distributions unix a déjà été porté sur une multitude d'architectures différentes comme les ARM, AVR, PIC, MIPS, SPARC, PowerPC, etc... Une version très connue de ce linux spécial *System-on-Chip* est appelé **μClinux** [39].

Quelques informations sur la *Milkymist*.

Une partie de notre projet sera de porter le noyau utilisé dans la Milkymist [34] pour qu'il fonctionne sur la **Nexys3**. Il est important de noter que le projet *Milkymist* est l'un des premiers projets d'architecture matérielle complètement *open-source*. Le code de son gestionnaire de mémoire a d'ailleurs été utilisé par la NASA pour construire un système totalement différent du traitement vidéo.

La *Milkymist* utilisant le processeur open source LatticeMico32 [38], il est possible de lui associer un noyau μClinux mais aussi un noyau **RTEMS** pour *Real-Time Executive for Multiprocessor Systems*. Une multitude de drivers ont également du être développés pour dialoguer avec les nombreux périphériques présents sur la carte tels que les ports USB pour brancher une souris et/ou un

clavier, un port midi pour éventuellement ajouter des interactions avec un instrument de musique, un contrôleur VGA pour afficher le résultat du traitement, une carte son.

Une fois cette étape d'adaptation terminée nous pourrons ensuite nous lancer dans le développement de modules propres à la fonctionnalité que nous donnerons au système. Pour cela il suffira de développer les drivers qui accompagneront le module. Au final, ce travail tombera dans le domaine de l'*open-source* pour que toute la communauté puisse profiter de ce portage sur la Nexys3.

Finalement ce portage ne devrait pas être très long vu que la carte possède plus ou moins le même matériel que la *Milkymist*. C'est pourquoi nous avons décider de nous lancer dans un projet traitant un domaine en vogue : le *reconfigurable computing*.

2.2.3 Reconfiguration du matériel à la volée

Le *Reconfigurable Computing* est une idée qui est apparue dans les années 60 [33] mais qui faute de capacité technologique à cette époque n'a pas pu aboutir avant 1980-90. En effet, on a commencé à graver de plus en plus de transistors sur une même surface de substrat et les puces FPGA ont commencé à voir le jour. L'idée majeure de ce paradigme est de joindre une partie de contrôle comprenant un processeur à une zone de logique reprogrammable à chaud. Il deviendrait alors possible de dédier plusieurs parties de cette logique à des tâches bien particulières qui pourraient entièrement s'exécuter en parallèle. On aurait alors un gain de cycles d'horloge considérable par rapport à un modèle classique de Von Neumann[40] qui exécute toutes les instructions en série et décompose d'ailleurs chaque étape en *fetch-decode-execute*.

Prenons un exemple simple permettant de mettre en lumière ce gain. Nous avons une machine qui possède une logique reconfigurable à chaud. Cette machine dispose également d'une carte son. Pour lire un fichier audio un système multitâche classique sera obligé de créer une tâche dédiée à la lecture de ce fichier. À différents instants, ce processus va prendre et laisser la main au processeur pour traiter les autres tâches. Le fait de valser entre les tâches et d'exécuter le processus coûte bien évidemment du temps précieux au processeur. Combien gagnerait-t-on si on créait au lieu d'une tâche, un circuit dédié qui s'occuperait indépendamment de la lecture de ce fichier et du transit jusqu'à la carte son ?

Cette tâche n'apparaîtrait plus du tout dans le planning de l'ordonnanceur et plus aucun cycle d'horloge ne serait perdu par le processeur pour exécuter cette tâche. L'idée présentée juste au dessus est à rapprocher des puces des ordinateurs classiques chargées de transiter des données d'une mémoire à une autre sans passer par le processeur (*Direct Memory Access*).

Le temps gagné en n'effectuant plus ces petites tâches anodines serait considérable. On pourrait également imaginer reconfigurer une logique qui effectue un calcul complètement dédié. On entend par là que pour effectuer un calcul complexe, on perdrait moins de temps à avoir une logique adaptée à ce calcul que de passer par des cycles de décodage d'une instruction pour effectuer plusieurs calculs à la suite et obtenir enfin le résultat. Cette idée est résumée dans les deux graphiques ci-dessous.

On passerait d'une oprération (A+B) >> C effectuer en deux temps par un modèle de Von Neumann, à un une opération exécutée en une unité de temps avec la logique reconfigurable comme illustré ci-dessous.

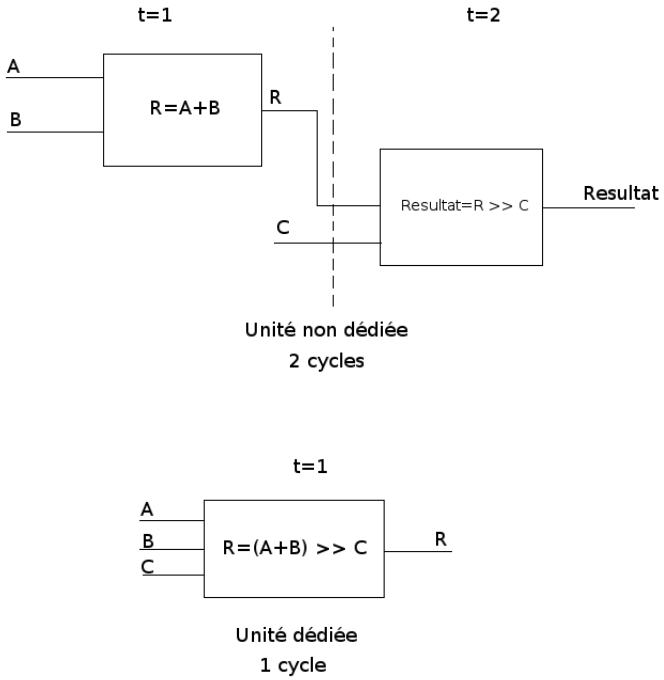

FIGURE 2.3 – Opération non-dédiée et dédiée

Conclusion

Comme nous l'avons vu au cours de ce rapport les FPGA permettent bien plus de choses que de prototyper des ASIC. Leur puissance et leur nombre de portes grandissant leur permettent maintenant d'acceuillir des systèmes complets tels que *Milkymist*. De plus, leurs facultés à pouvoir se reconfigurer à chaud ouvrent de nombreuses portes qui pourraient bien révolutionner l'informatique. Nous comptons alors ajouter un module au projet *open-source Milkymist* permettant d'utiliser cette caracteristique. Nous avons déjà réussi à porter le projet sur la carte de développement Nexys3 utilisant le FPGA Spartan6 de Xilinx fourni par l'INSA. Le module que nous allons concevoir permettra alors de rajouter à chaud des périphériques au *System-on-Chip Milkymist* selon les besoins du moment.

Bibliographie

[1] RUBOW, Erik. *Open Source Hardware*. **[en ligne]**. États-Unis d'Amérique : University of California, San Diego, Cours, 2008, 5p. Disponible sur http://cseweb.ucsd.edu/classes/fa08/cse237a/topicresearch/erubow_tr_report.pdf (Consulté le 24 janvier 2013).

[2] ACOSTA, Roberto. *Open Source Hardware*. **[en ligne]**. États-Unis d'Amérique : Massachusetts Institute of Technology, 2009, 83p. Disponible sur http://18.7.29.232/bitstream/handle/1721.1/55201/609411873.pdf (Consulté le 24 janvier 2013).

[3] BAXTER, Julius, BENNETT, Jeremy et opencores.org. *Practical System-on-Chip*. **In** : *Open Source Hardware User Group* **[en ligne]**. (Modifié le 29 mars 2012). Disponible sur http://oshug.org/presentations/openrisc-oshug17-final.pdf (Consulté le 24 janvier 2013).

[4] GAISLER, Jiri. *Industrial use of open-source IP cores. 45th IFIP WG 10.4 - Workshop on "Open Source and Dependability"*. Moorea polynésie française, 5-9 mars 2004. Disponible sur http://webhost.laas.fr/TSF/IFIPWG/Workshops&Meetings/45/06-Gaisler.pdf (Consulté le 24 janvier 2013).

[5] INMAN, Kurt and FRITSKY, Lauren. *What Is an IP Core ?*. **In** : *wiseGEEK* **[en ligne]**. Disponible sur http://www.wisegeek.com/what-is-an-ip-core.htm (Consulté le 24 janvier 2013).

[6] SALEM, Mohamed A. and KHATIB, Jamil I. An introduction to open-source hardware development. EETimes. Modifié le 07 janvier 2004. Disponible sur http://eetimes.com/electronics-news/4155052/An-introduction-to-open-source-hardware-development.

[7] OSHW. *Definition of Free Cultural Works* **[en ligne]**. (modifié le 9 janvier 2013). Disponible sur : http://freedomdefined.org/OSHW. (Consulté le 24 janvier 2013).

[8] WITZ, Jean-François, CHAMBON, Olivier. *Matériel OpenSource (OpenHardware) et la recherche*. **In** : *PLUME* **[en ligne]**. (Modifié le 29 août 2011). Disponible sur https://www.projet-plume.org/ressource/openhardware (Consulté le 24 janvier 2013).

[9] TORRONE, Phillip et FRIED, Limor. *Million dollar baby. Foo Camp East 2010* (http://foocamp10.wiki.oreilly.com/wiki/index.

php/Main_Page). Sebastopol, Californie, 25-27 juin 2010. Disponible sur http://www.adafruit.com/pt/fooeastignite2010.pdf (Consulté le 27 janvier 2013).

[10] List of open-source hardware projects. *Wikipedia* **[en ligne]**. (modifié le 17 janver 2013). Disponible sur : http://en.wikipedia.org/w/index.php?title=List_of_open-source_hardware_projects&oldid=533515806. (Consulté le 27 janvier 2013).

[11] Application-specific integrated circuit. *Wikipedia* **[en ligne]**. (Modifié le 08 janver 2013). Disponible sur : http://en.wikipedia.org/wiki/Application-specific_integrated_circuit. (Consulté le 20 janvier 2013).

[12] Programmable logic device. *Wikipedia* **[en ligne]**. (Modifié le 11 janver 2013). Disponible sur : http://en.wikipedia.org/wiki/Programmable_logic_device. (Consulté le 20 janvier 2013).

[13] POITI, Alexis. *Cours en ligne de TELECOM Paristech sur les HDL.* **[en ligne]**. France : Paristech, Cours en ligne, 2005. Disponible sur http://comelec.enst.fr/hdl/ (Consulté le 24 janvier 2013).

[14] Foundry model. *Wikipedia* **[en ligne]**. (Modifié le 09 janver 2013). Disponible sur : http://en.wikipedia.org/wiki/Foundry_model. (Consulté le 20 janvier 2013).

[15] Netlist. *Wikipedia* **[en ligne]**. (Modifié le 09 janver 2013). Disponible sur : http://en.wikipedia.org/wiki/Netlist. (Consulté le 20 janvier 2013).

[16] Register-transfer level. *Wikipedia* **[en ligne]**. (Modifié le 01 janver 2013). Disponible sur : http://en.wikipedia.org/w/index.php?title=Register-transfer_level&oldid=530759637. (Consulté le 24 janvier 2013).

[17] THOMAS, Donald E. et MOORBY, Philip R. *The Verilog® Hardware Description Language.* **[en ligne]**. Springer. Etats-Unis d'Amérique : 2002, 408p. Format google book. Disponible sur http://books.google.fr/books?id=DxQGrz7q-SwC&printsec=frontcover&hl=fr#v=onepage&q&f=false (Consulté le 24 janvier 2013).

[18] Milkymist. *Wikipedia* **[en ligne]**. (Modifié le 13 janvier 2013). Disponible sur : http://en.wikipedia.org/w/index.php?title=Milkymist&oldid=532830955 (Consulté le 24 janvier 2013).

[19] KASHAP, Satish. Lec 08 Hardware Description Language (HDL). [18 mars 2012] [enregistrement vidéo] **In :** Youtube [1h 07' 54"]. Disponible sur : http://www.youtube.com/watch?v=rdAPXzxeaxs. (Consulté le 20 janvier 2013).

[20] GIOMI, Jean-Charles et TARROUX, Gerard. *Integrated circuit fabrication using state machine extraction from ....* Brevet 5537580. 21 décembre 1994.

[21] Hardware description language. *Wikipedia* **[en ligne]**. (Modifié le 14 janvier 2013). Disponible sur : http://en.wikipedia.org/w/index.php?

title=Hardware_description_language&oldid=532956772 (Consulté le 24 janvier 2013).

[22] SMITH, Douglas J. VHDL & Verilog Compared & Contrasted Plus Modeled Example Written in VHDL, Verilog and C. In Rajesh Bawankule's Verilo Center. **[en ligne]**. (Denière mise à jour du site : 12 février 2003) Disponible sur http://www.angelfire.com/in/rajesh52/verilogvhdl.html (Consulté le 07 janvier 2013).

[23] CHAKRABARTI, Indrajit. *VERILOG Hardware Description Language* **[en ligne]**. Inde : IIT Kharagpur, Cours, 2007, 24p. Dispobible sur http://www.smdp.iitkgp.ernet.in/PDF%5CVLSI_DSP%5CIC-Verilog.pdf (Consulté le 05 janvier 2013).

[24] Verilog. *Wikipedia* **[en ligne]**. (Modifié le 7 septembre 2012). Disponible sur : http://fr.wikipedia.org/w/index.php?title=Verilog&oldid=82794321. (Consulté le 20 janvier 2013).

[25] DON, Thomas. *The Verilog Hardware Description Language*. **[en ligne]**. Etats-Unis d'Amérique : University of Iowa, Cours, 1998, 11p. Disponible sur http://www.engineering.uiowa.edu/~hpca/LectureNotes/VerilogTutorialSpring11.pdf (Consulté le 05 janvier 2013).

[26] VHDL. *Wikipedia* **[en ligne]**. (Modifié le 4 décembre 2012). Disponible sur : http://fr.wikipedia.org/w/index.php?title=VHDL&oldid=86139857. (Consulté le 20 janvier 2013).

[27] 1076-1987 - IEEE Standard VHDL Language Reference. *IEEE Standard Association* **[en ligne]**. Disponible sur http://standards.ieee.org/findstds/standard/1076-1987.html (Consulté le 20 janvier 2013).

[28] 1076-1993 - IEEE Standard VHDL Language Reference. *IEEE Standard Association* **[en ligne]**. Disponible sur http://standards.ieee.org/findstds/standard/1076-1993.html (Consulté le 20 janvier 2013).

[29] P1076 - VHDL Analysis and Standardization Group. *IEEE Standard Association* **[en ligne]**. Disponible sur http://standards.ieee.org/develop/wg/P1076.html (Consulté le 20 janvier 2013).

[30] 1364-1995 - IEEE Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language *IEEE Standard Association* **[en ligne]**. Disponible sur http://standards.ieee.org/findstds/standard/1364-1995.html (Consulté le 20 janvier 2013).

[31] 1364-2001 - IEEE Standard Verilog Hardware Description Language. *IEEE Standard Association* **[en ligne]**. Disponible sur http://standards.ieee.org/findstds/standard/1364-2001.html (Consulté le 20 janvier 2013).

[32] 1364-2005 - IEEE Standard for Verilog Hardware Description Language. *IEEE Standard Association* **[en ligne]**. Disponible sur http://standards.ieee.org/findstds/standard/1364-2005.html (Consulté le 20 janvier 2013).

[33] Reconfigurable Computing Wikipedia. *Wikipedia* **[en ligne]**. (modifié le 01 janvier 2013). Disponible sur : http://en.wikipedia.org/wiki/Reconfigurable_computing. (Consulté le 16 janvier 2013).

[34] Milkymist official website. *Milkymist* **[en ligne]**. Disponible sur : http://milkymist.org/3/. (Consulté le 20 janvier 2013).

[35] SALEH, Resve ; MIRABBASI, Shahriar ; LEMIEUX, Lemieux ; GRECU, Cristian. System-on-Chip : Reuse and Integration **[en ligne]. In :** *Proceedings of the IEEE*, vol. 94, No. 6, June 2006. Disponible sur : http://eecs.wsu.edu/~pande/Journal_Papers/Paper_IEEE_Proceedings.pdf (Consulté le 05/01/2013)

[36] M. WINZKER. *Advenced FPGA Design*. **[en ligne]** Buenos Aires : Universidad Tecnológica Nacional. Cours, 2010, 15 slides. Disponible sur : http://www.electron.frba.utn.edu.ar/dplab/CursoLP/Lecture_FPGA_5.pdf (Consulté le 05/01/2013)

[37] BOURDEAUDUCQ, Sébastien. *A performance-driven SoC architecture for video synthesis*. **[en ligne]** Master of Science Thesis in System-on-Chip Design, Stockholm, Royal Institute of Technology, 2010. 109. Disponible sur : http://milkymist.org/3/thesis/thesis.pdf (Consulté le 01/01/2013)

[38] Lattice semiconductor. *LatticeMico32 architecture* **[en ligne]**. Disponible sur : http://www.latticesemi.com/products/intellectualproperty/ipcores/mico32/index.cfm. (Consulté le 19 janvier 2013).

[39] uClinux. *Embedded Linux/Microcontroller Project* **[en ligne]**. Disponible sur : http://www.uclinux.org/. (Consulté le 20 janvier 2013).

[40] Wikipedia. *Architecture de Von Neumann* **[en ligne]**. Disponible sur : http://fr.wikipedia.org/wiki/Architecture_de_von_Neumann. (Consulté le 20 janvier 2013).