

Rapport de projet tutoré : Développement d'un System-On-Chip open-source

DUZAN Luc LONGO Matthieu FARACHE Gabriel
MICHAUD Clément

20 janvier 2012

Table des matières

1	Conception du matériel	4
1.1	Langage de description de circuit logique	4
1.1.1	Présentation de Verilog et VHDL	4
1.1.2	Les autres HDL existant	5
1.1.3	Verilog VS VHDL	6
1.1.4	Le HDL de Milkymist : Migen	6
1.2	Technologie de fabrication	8
1.2.1	ASIC	8
1.2.2	FPGA	10
1.3	Le monde de l'open source et son avancement sur les FPGA . . .	11
2	Adaptation d'un System On Chip libre : MilkyMist	14
2.1	Le projet MilkyMist	15
2.2	Le travail à réaliser	16
2.2.1	Portage sur le Nexys3	16
2.2.2	Adapter le système logiciel associé	18
2.2.3	Reconfiguration du materiel à la volée	20

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 même constituées de transistors qui sont donc les briques élémentaires de l'électronique numérique. En effet, ils peuvent être utilisés 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 (appelé 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 productions 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 cher. Mais l'apparition des

FPGA (pour *Field Programmable 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 performant pour contenir l'intégralité des composants nécessaires au fonctionnement d'un système complet (microprocesseurs, mémoire, GPIO par exemple), on parle alors de SoC. Les SoC (*System 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 Spartan 6 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 Migel 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 circuit logique

1.1.1 Présentation de Verilog et VHDL

Verilog et Very high speed integrated circuit Hardware Description Language ou VHDL sont tous deux des Hardware Description Language ou HDL soit en français Langage de Description de Matériel. 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 transistors. Cette méthode s'appelle Register-Transfer Level ou RTL et a été défini 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 matériel 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. A partir de cette représentation des circuits, des représentations de plus bas niveaux et le câblage peuvent être déduit. 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.

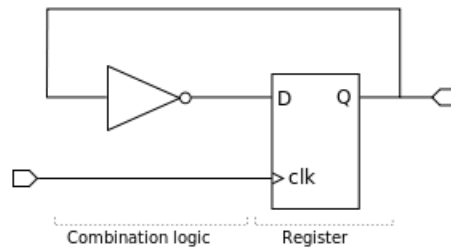


FIGURE 1.1 – Légende

Le schéma ci-dessus montre comment est représenté un circuit. Nous constatons qu'il est ici totalement fait abstraction des comment 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, générale du circuit.

La principale différence entre les HDL avec les langages de programmations traditionnels est qu'avec un HDL, contrairement aux autres type de langage procédurier, 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 existant

Il existe de nombreux HDL autre que les deux plus connu 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 ne soit 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. De part cette caractéristique, nous ne nous y attarderons pas plus. Milkymist, ceux qui font le SoC sur lequel nous allons travailler, possède aussi leur propre langage Migen qui est toujours en développement, nous y reviendrons plus loin dans cette partie.

1.1.3 Verilog VS VHDL

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

- VHDL :
 - ce langage a été fait 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 numérique développant des FPGAs et ASICs
 - parfait pour convertir des types de données de vecteurs de bits vers des notations arithmétiques,
 - existence de supports compréhensible pour la conception numérique de bas niveau.

Ce sont-là les principales différences entre les deux langages. Dans les fait, VHDL et Verilog sont similaires puisqu'ils sont tous les deux des standards IEEE : *IEEE standard 1076* en 1987 puis *IEEE standard 1076-1993* en 1993 après une mise à jour pour VHDL. Il a ensuite été mis à jour régulièrement pour arriver, en 2008 au standard *IEEE 1076-2008* (VHDL 4.0) qui est la dernière version en date. Verilog est lui devenu un standard en 1995 : *IEEE standard 1364-1995* et a été mis à jour deux fois depuis : en 2001 *IEEE Standard 1364-2001* (Verilog 2001) qui corrigera de nombreux problèmes puis en 2005 *IEEE Standard 1364-2005* (Verilog 2005) qui corrigea quelques bugs.. Historiquement, VHDL a été développé pour l'armée de l'air des Etats-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 (électronique). *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 était 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 avons aussi parlé de Migen, le langage de Milkymist. Nous n'allons pas l'utiliser mais il nous semblait intéressant d'en parler. En effet, dans le futur, le SoC Mylkymist utilisera ce langage. Le projet Migen (pour **Milkymist generator**) a été lancé en 2011 et est toujours en cours de de développement. Cet outil est basé sur le langage python et il a pour but d'automatiser encore plus le processus de conception des VSLI (Very-large-scale integration, c'est un processus de création de circuits intégrés). Grâce à Migen, il sera possible

d'appliquer des concepts modernes, tel que l'objet et la métaprogramation par exemple, pour concevoir un matériel. L'avantage étant que ce qui en résulte est plus élégant et est surtout plus facile à faire évoluer, tout en réduisant les problèmes dus aux erreurs humaines. Ce sont-là les grands principes que Milkymist veut mettre en place dans Migen. En se 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 basé sur l'évènement (le parallélisme). Migen permettra aussi d'intégrer le Soc (system-on-chips) en automatisant par exemple l'interconnexion entre les chips (puces). Migen accélérera aussi la conception matériel grâce le 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.

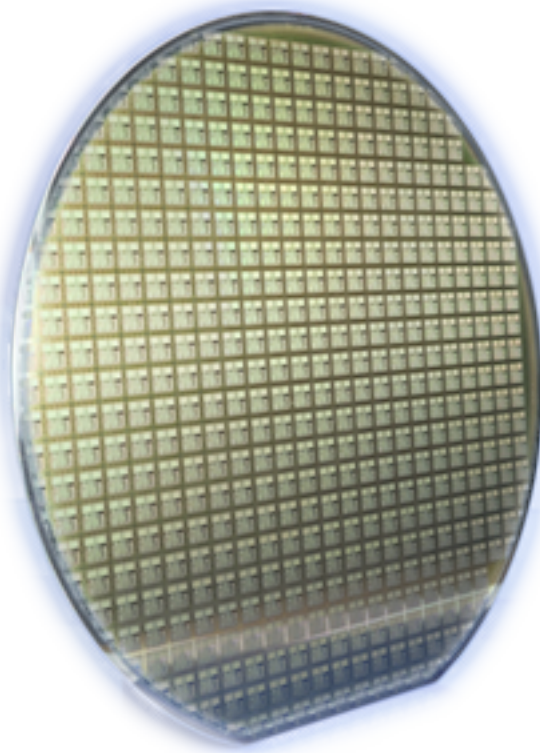
1.2 Technologie de fabrication

1.2.1 ASIC

Netlist

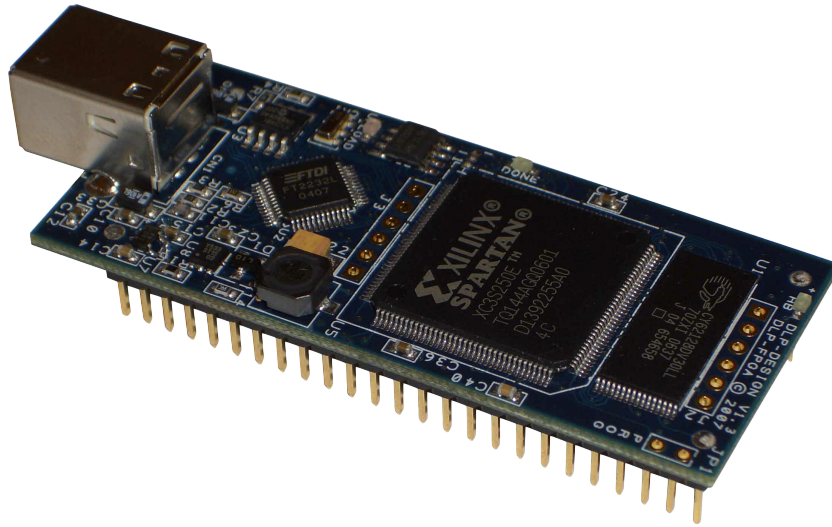
Une fois la conception réalisée à l'aide de langages de descriptions matériel, on peut obtenir une netlist du circuits intégrés. Cette netlist contient dans un premier temps, une liste qui défini les composants utilisés dans ce circuit intégré, chaque composant est définie par les différents points de connexion (appelé *port*) 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 complexe circuit intégré. 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'une 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 est alors donnée qu'une seule fois.

Fabrication d'ASIC



A 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. Ce fichier permet alors d'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 matériels, 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. Seul quelques entreprises tel que 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.

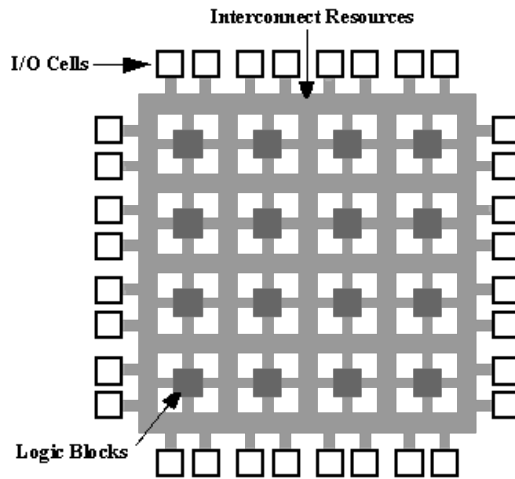
1.2.2 FPGA



Les FPGA (*Field Programmable Gate Array*) sont des circuits logiques programmable. En effet *Field Programmage* signifie programmable sur le champ, le circuit peut être programmé après sa fabrication pour réaliser n'importe qu'elle fonctions logiques. La configuration d'un FPGA est obtenu 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 constitué d'un nombre très importants de blocs logiques rangés en matrice (d'où le *Gate Array* de FPGA). C'est blocs logiques peuvent être configurés pour effectués des fonctions logiques élémentaires (et, ou, ou exclusif, non) ou quelques fonctions logiques plus complexes. Ils peuvent aussi être configurer en bascule ou en d'autres éléments permettant de mémoriser de l'information. Ces blocs 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 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 programmable des périphériques. Cela permet d'économiser des portes, de plus certains périphériques tel 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 embarquent des GPIO, de la RAM, des cartes réseaux par exemple. Dans notre projet tuteuré, la différence des périphériques utilisés par la carte FPGA utilisé à l'INSA et celle utilisé par MilkyMist entraînent des problèmes de portabilités.



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



Dans cette partie nous allons parler des matériels open-source, c'est à dire que les matériels, ou plutôt leurs conceptions, sont créés de manière communautaire ou par des sociétés privées. Ils ne sont pas soumis à des brevets ultra-restrictifs mais qui sont le plus souvent sous licence GPL (General Public License) ou assimilé plus souples.

Le matériel open-source est très peu développé 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 le **Open Source Hardware (OSHW)**. Il s'agit d'une définition de ce qu'est un matériel open-source et des idéaux de ce dernier.. 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 cœur de la machine), carte graphique, ...), voir même dans d'autres secteurs tels que la téléphonie et le GPS. 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* qui date des années 1990, *Minimig* qui est une nouvelle implémentation de l'Amiga 500 utilisant des FPGA, *Milkymist One* qui est un

carte-ordinateur implémentant tous les principes logiques à travers des fichiers de conception créés avec *Verilog*. Ces fichiers sont open-source, donc disponible à tout le monde. Dans la plupart des autres matériel open-source, ces fichiers sont confidentiels et propriétaire. Bien évidemment, cette carte utilise *Milkymist SoC* comme CPU. En parlant de CPU, nous pouvons aussi lister les différents

projets : *Amber*, *LEON*, *OpenCores*, *OpenSPARC*, *OpenRISC* qui est plus un groupe de développeur 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 SoC développé de manière indépendante et open-source. Nous constatons qu'il y a bien moins de projet open-source concernant les CPU que les ordinateurs eux-mêmes. La plus grosse communau-

ter à l'heure actuelle est OpenCores (<http://www.opencores.org>). Fondée en 1999 par Damjan Lampret, OpenCores (OC) a commandé et développé *OpenRISC 1000*. Le développement s'est poursuivi jusqu'en 2007. Durant cette période OC a été supporté par *Flextronics*. A partir de 2007, *OpenRISC 1000* est passé dans une phase de déploiement commerciale 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 matériel. *ORPSoC* est l'implémentation de SoC de référence sur cette architecture. Cependant, le *Raspberry Pi* (raspi), l'ordinateur à 29\$ sortie courant 2012 grossit de plus en plus. De nombreuses cartes électroniques complémentaires au raspi sont sorties, toutes dans un prix raisonnable. Un projet de sortir une caméra est même en cours et son prix devrait lui aussi être extrêmement raisonnable compte tenu de la technologie impliquée. Il faut sa-

voir que contrairement aux idées reçues, les entreprises qui font de l'open-source hardware ont de (très) 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) :

- 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
- 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 existe bien d'autres entreprises de ce type qui dépassent ou approche le million de chiffre d'affaire. De plus ces chiffres

datent de 2010, on peut donc légitimement supposé que ces chiffres ont évolués de manières positives ces deux dernières années. De plus, selon les prévisions, en 2015, le chiffre d'affaire cumulé de toutes les entreprises de matériel open-source devrait avoisiné le milliard de dollars. Ce secteur a donc un avenir radieux qui s'offre à lui, de plus en plus de projet voient le jour et ceux qui sont déjà présents ne font que s'améliorer grâce au travail que donne toutes les 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

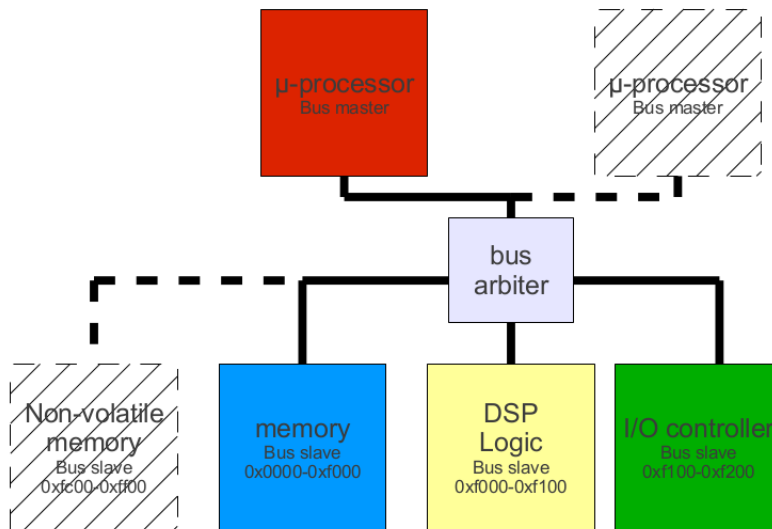
2.1 Le projet MilkyMist

Ce projet a pour origine la thèse de Sébastien Bourdeauducq 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 principal but 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 contrairement 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, il utilise le coeur LatticeMico32 (LM32). C'est un CPU 32-bit big endian RISC Opensources. Le microprocesseur ML32 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.

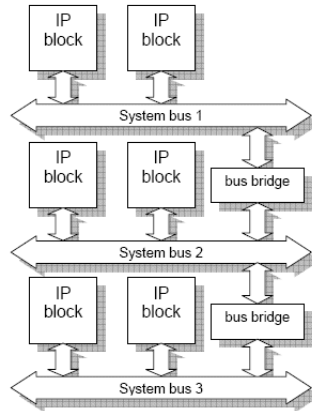
2.2 Le travail à réaliser

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 écrit pour un certain projet sont réutilisables dans d'autres, ils ont été souvent 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 entraînant le paiement de droit ou licences opensources avec chacune leurs particularités). 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 diminution des coûts et enfin rassurer les développeurs car la solution est déjà prouvée.



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 ou/et 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 cas, c'est un CSR bus conçu par Sébastien Bourdauducq, 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 dû à sa généricité quelques configurations lorsqu'on le change de FPGA (taille du bus, nombre de registres, alimentation, timing, taille du cache, ...) et on peut également 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, ...).

Va être aussi réutilisé le GPIO, l'USART, la ROM qui contiendra le BIOS, ... (à revoir cette partie là, y'a tellement de truc commenté et décommenté dans le fichier que c'est resté flou dans ma tête).

Chacun d'eux va être 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 à nos besoins. On peut gagner par cette méthode énormément en efficacité face aux architectures non-spécialisées, ce qui permet pour un résultat égal de travailler à fréquence plus basse et du coup de moins chauffer et consommer d'énergie.

2.2.2 Adapter le système logiciel associé

Après avoir créée l'architecture qui correspond au cahier des charges du SoC, il faut encore adapter ou 'porter' le système d'exploitation (Operating System en anglais) qui va le cadencer, c'est-à-dire 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. Il faut tout d'abord faire la différence entre un système d'exploitation classique que l'on trouve sur les PC actuels qui permet le traitement de tâches multiples plus ou moins en même temps grâce au système du temps partagé (time sharing), qui contient une unité de translation des adresses physiques en adresses virtuelles (MMU), qui gère les tâches de manière avancée et qui 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 **uClinux**.



Quelques informations sur la Milkymist.

Une partie de notre projet sera de porter le noyau utilisé dans la Milkymist 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, il est possible de lui associer un noyau uClinux 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 tel 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 décidé de nous lancer dans un projet traitant un domaine en vogue : le **reconfigurable computing**.

2.2.3 Reconfiguration du matériel à la volée

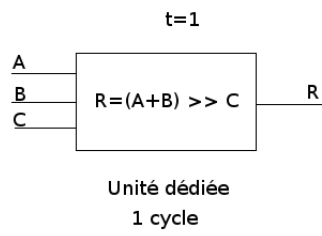
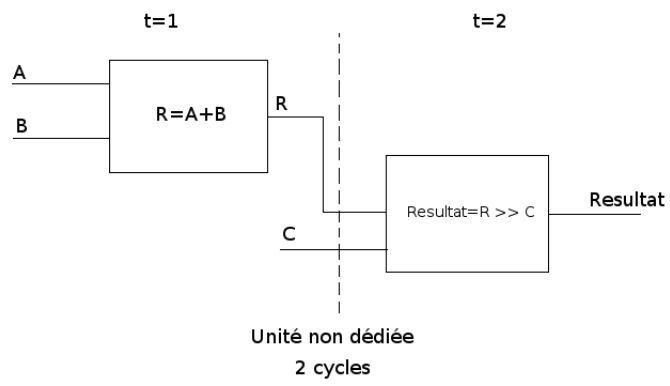
Le 'reconfigurable computing' est une idée qui est apparue dans les années 60 mais qui faute de capacité technologique à cette époque n'a pas pu aboutir avant 1980-90. En effet, on a commencé à graver de plus en plus de transistors sur une même surface de substrat et les chips **FPGA** ont commencé à 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 dédicacer 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 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. A 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 seraient 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 adapté à ce calcul que de passer par des cycles de décodage d'une instructions pour effectuer plusieurs calculs à la suite et obtenir enfin le résultat. Cette idée est résumée dans les 2 graphiques ci-dessous. On passerait d'une opération

$(A+B) \gg C$ effectuer en deux temps par un modèle de Von Neumann, à une opération exécutée en une unité de temps avec la logique reconfigurable comme illustré ci-dessous.



Conclusion

à faire