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

DUZAN Luc LONGO Mathieu FARACHE Gabriel
MICHAUD Clément

20 janvier 2012

Table des matières

Introduction

Chapitre 1

FPGA et outils de conception actuel d'ASIC

1.1 FPGA et ASIC

A FAIRE

1.2 Langages de description Verilog et VHDL

A FAIRE

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

A FAIRE

Chapitre 2

L'adaptation de Milkymist sur la Nexys3

2.1 Portabilité d'une architecture matériel

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 décider de nous lancer dans un projet traitant un domaine en vogue : le **reconfigurable computing**.

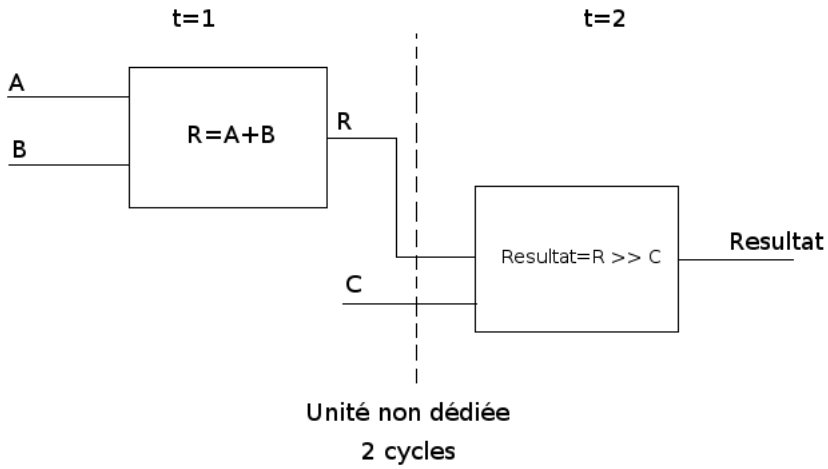
2.3 Reconfiguration du materiel à la volée

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 comprennant 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.

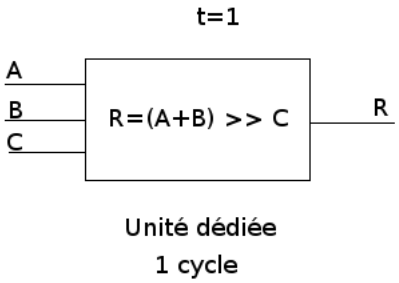
Prenons un exemple simple permettant de mettre en lumière ce gain. Nous avons une machine qui possède une logique reconfigurable à chaud. Cette machine dispose également d'une carte son. Pour lire un fichier audio un système multitâche classique sera obligé de créer une tâche dédiée à la lecture de ce 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 indépendemment 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 pésentée 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 complétement 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 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.



Conclusion