Grenoble INP – ENSIMAG
École Nationale Supérieure d'Informatique et de Mathématiques Appliquées

Rapport de projet de fin d'études

Effectué chez Elsys-Design

Design FPGA et Vérification UVM

Eduardo Tomasi Ribeiro
3e année – Option SEOC

04 mars 2019 – 16 août 2019

Elsys Design
Space Antipolis 1, 2323 Chemin Saint-Bernard
06220 Vallauris Cedex

Responsable de stage
Nicolas Herve
Tuteur de l'école
Stephane Mancini

Résumé

Afin de permettre à ses nouveaux ingénieurs d'être rapidement opérationnels sur les missions, Elsys-Design a mis en place une plate-forme de développement servant d'outil d'autoformation. Récemment cette plate-forme a été mise en avant pour valoriser le partenariat existant avec ARM. La société utilise cette formation en interne pour ses ingénieurs et, potentiellement, ses clients. Cette formation est destinée aux ingénieurs en micro-électronique désireux d'acquérir de nouvelles compétences en intégration système et/ou vérification.

Dans ce contexte, les objectifs du stage sont de prendre en main cette plate-forme, de contribuer à son amélioration et de répondre aux besoins en développement basé sur une plate-forme et en faire sa vérification. Lors du stage, les travaux suivants sont définis : l'intégration d'un IP existant avec une architecture basée sur un processeur ARM Cortex-M0 et son bus, l'écriture des drivers nécessaires à son utilisation, la simulation et l'implémentation sur FPGA. Ensuite, un plan de vérification de l'IP mise en œuvre est défini, puis un environnement de vérification développé afin de mettre à exécution le plan.

Afin d'augmenter un peu plus la difficulté du stage, et de rapidement développer les compétences y associées, il a été décidé de réorienter le stage vers quelque chose de plus audacieux. Un accélérateur cryptographique a été mis en œuvre dans le but de faire des calculs RSA de façon rapide, sécurisée et efficace.

Les différentes tâches parcourues lors du stage afin de réaliser cette formation sont les suivantes : différents modules seront développés en VHDL et intégrés avec le DesignStart Cortex-M0 de chez ARM en utilisant les sous-systèmes des bus APB et AHB. L'IP doit être ensuite vérifié en utilisant des tests dirigés. Une deuxième étape de vérification doit être faite, cette fois-ci en utilisant la méthodologie UVM, basée sur la programmation orientée objet, permettant de réaliser une vérification plus complète de l'IP.

Mots-clés : vérification UVM, AHB, ARM, design, FPGA, RSA, cryptographie

Abstract

In order to allow its new engineers to quickly be operational in their missions, Elsys-Design has developed a platform serving as a self-training tool. Recently, this platform was highlighted to enhance the partnership with ARM. It can be used by its own engineers and, potentially, by its clients, specially microelectronic engineers wanting to acquire new skills in system integration and/or verification.

In this context, the goal of the internship is to contribute to the improvement of this platform, as well as to answer the new requirements of the market in terms of UVM verification and platform-based development. Throughout the training, the following tasks are defined: the integration of an existing IP with a Cortex-M0 processor-based architecture and its bus, the definition of all the drivers needed to its use, its simulation and its implementation in a FPGA. Finally, a verification plan must be defined so a verification environment can be developed and executed.

Afterwards, in order to increase the internship's difficulty and to quickly develop the related skills, the internship has been reoriented to something more challenging. A cryptographic accelerator has been implemented so RSA computations can be done in a quick, secure and effective way.

Several tasks have been done throughout the internship so this training can be completed: different IPs were developed and integrated with the ARM's DesignStart Cortex-M0 using its APB and AHB subsystems. These IPs must, then, be verified using unit tests. A second verification stage is done, this time using the UVM methodology, based on the Object-Oriented Programming, allowing to achieve a more complete verification of the IP.

Keywords: UVM verification, AHB, ARM, design, FPGA, RSA, cryptography

Table des matières

1 Introduction **6**

2 Présentation de l'entreprise **7**
2.1 Groupe Advans 7
2.2 Avisto 7
2.3 Mecagine 7
2.4 Elsys-Design 7

3 Projet ETNA **9**
3.1 L'environnement ETNA 9
3.2 Problème posé 11

4 Mise en œuvre d'un filtre **12**
4.1 Filtres à réponse impulsionnelle finie (FIR) 12
4.2 Développement du module FIR 13
4.2.1 Architecture systolique 13
4.2.2 Implémentation en haut niveau 14
4.2.3 Implémentation RTL 14

5 Intégration au projet ETNA **17**
5.1 Advanced High-Performance Bus (AHB) 17
5.1.1 Maître AHB-Lite 18
5.1.2 Esclaves AHB-Lite 19
5.1.3 Modes d'opération du protocole AHB-Lite 19
5.2 Intégration avec le système ARM DesignStart 20
5.2.1 Interface 21
5.2.2 Driver 22

6 Vérification **23**
6.1 Universal Verification Methodology (UVM) 23
6.1.1 Classes UVM 23
6.1.2 Métriques de vérification 24
6.2 Architecture mise en place 25
6.3 Analyse des résultats 26

7 Accélérateur cryptographique **28**
7.1 Chiffrement RSA 28
7.2 Développement du module RSA 29
7.2.1 Opération d'exponentiation modulaire 30
7.2.2 Opération de multiplication modulaire 32
7.3 Optimisations 34
7.3.1 Additionneurs 34
7.3.2 Comparateurs 38
7.4 Résultats 38

8 Avancement et état du projet **40**

9 Prise en compte de l'impact environnemental et sociétal **41**

10 Conclusion **42**

Annexes **44**

A Altera DE2-115 **44**

B Utilisation du module FIR **46**

C Signaux de filtrage en C **47**

D Signaux de l'IP FIR RTL **48**

E Test du driver du filtre FIR **49**
E.1 Filtre passe-haut 49
E.2 Filtre passe-bas 50

F Résultats de la vérification du FIR **51**

G Schémas-bloc de l'accélérateur cryptographique **52**
G.1 Exponentiation modulaire 52
G.2 Multiplication modulaire 52
G.3 Transformation de Montgomery 53
G.4 Réduction de Montgomery 53

1 Introduction

Elsys-Design, en tant que société de services, doit s'adapter en permanence aux changements du tissu industriel au niveau national. La tendance actuelle se situe autour des métiers de la micro électronique et plus particulièrement la vérification numérique, ce qui représente une importante partie du *flow* de conception FPGA (Field-Programmable Gate Array) et ASIC (Application Specific Integrated Circuit).

Afin de permettre à ses nouveaux ingénieurs d'être rapidement opérationnels sur les missions, Elsys a mis en place une plate-forme de développement servant d'outil d'autoformation. Récemment cette plate-forme a été mise en avant pour valoriser le partenariat existant avec ARM. La société utilise cette formation en interne pour ses ingénieurs et, potentiellement, ses clients désireux d'acquérir des nouvelles compétences en intégration système et/ou vérification. La formation s'adresse aussi aux entreprises soucieuses de développer, en leurs seins, une méthodologie et un langage commun et robuste pour leurs ingénieurs.

Dans ce contexte, les objectifs du stage sont de prendre en main cette plate-forme, de contribuer à l'améliorer et répondre aux besoins en développement basé sur une plate-forme et vérification UVM (Universal Verification Methodology). Lors de cette formation, les travaux suivants sont définis : intégration d'un IP (Intelectual Property) existant avec une architecture basée sur un processeur ARM Cortex-M0 et son bus, l'écriture des drivers nécessaires à son utilisation, la simulation et l'implémentation sur FPGA. Ensuite, un plan de vérification est défini afin de permettre le développement d'un environnement de vérification.

Toutefois, afin d'augmenter la complexité du stage, il a été décidé de le réorienter vers quelque chose de plus audacieux. Un système d'accélération de calcul cryptographique a été choisi pour être mis en œuvre rapidement et avec un fonctionnement efficace et sécurisé. Le but est de développer des compétences en cryptographie autour du projet ETNA et, potentiellement, la création d'un IP commercialisable. Cette implémentation a pris un tiers du temps de stage, lors que le projet ETNA en a pris deux tiers.

Le rapport est articulé ainsi : dans la Section 2, il est présenté le groupe Advans ainsi que les différentes entreprises qui le composent — notamment Elsys Design. Dans la Section 3 le projet ETNA et son environnement est présenté aussi bien que les objectifs attendus et les travails à réaliser lors du stage. La Section 4 présente la première partie du stage, qui s'agit de la mise en œuvre d'un IP de filtrage. L'intégration de cet IP au projet ETNA est présenté dans la Section 5 et sa vérification, en utilisant la méthodologie UVM, dans 6. La mise en œuvre de l'accélérateur cryptographique est dans la Section 7. L'état du projet et les détails de son avancement sont dans 8, ainsi qu'une réflexion sur l'impact environnemental dans la Section 9. Finalement, une conclusion sur les apports du projet est faite dans la Section 10.

2 Présentation de l'entreprise

Dans cette section, est présentée la société Elsys-Design, où le stage s'est déroulé. Elsys-Design fait partie du Groupe Advans, ainsi que deux autres sociétés. Le stage s'est passé à Sophia Antipolis, mais Elsys est présente dans plusieurs pays et dans plusieurs villes de France.

2.1 Groupe Advans

Le groupe Advans a été créé en 2002 et est un acteur européen de la sous-traitance en projets complexes, spécialisé dans les domaines des systèmes électroniques, des logiciels applicatifs et réseau et de la mécanique [1].

En plus de la prestation de services, le groupe conçoit des solutions technologiques et il offre un soutien à l'innovation, en accompagnant des jeunes sociétés innovantes et prometteuses dans différents marchés.

Fondé et managé entièrement par des ingénieurs, le groupe est présent dans 7 pays et emploie plus d'un millier d'ingénieurs qui interviennent pour de grandes sociétés internationales et de Petites et Moyennes Entreprises (PME) technologiques évoluant au sein de nombreuses entreprises. Il est, aujourd'hui, constitué par trois sociétés : Avisto (logiciels applicatifs), Mecagine (mécanique) et Elsys-Design (spécialiste des systèmes électroniques).

2.2 Avisto

Avisto est la société de Recherche et Développement (R&D) et de services du groupe Advans dont l'expertise est le développement de logiciels applicatifs. Présente dans 9 villes, elle est spécialiste dans la réalisation de projets mettant en œuvre des technologies Objet, Web et Mobile. La société intervient dans les domaines des systèmes d'information, du web, des télécommunications, du logiciel industriel, de l'applicatif embarqué, du temps réel et du réseau. La société couvre toutes les étapes de la vie d'un logiciel, de la spécification des besoins des clients à la préparation de ses évolutions futures [3].

2.3 Mecagine

Mecagine est une société d'ingénierie et de conseil en mécanique des structures et des systèmes, présente uniquement à Paris. Elle se spécialise dans la conception, le calcul de structures, la simulation thermique et l'optimisation. La société gère les projets clients dans les phases d'analyse, d'étude, de conception, de calcul et de prototype jusqu'à l'industrialisation [6].

2.4 Elsys-Design

Elsys-Design, créé en 2000 à Paris par deux ingénieurs nommés François AGNETTI et Radomir JOVANOVIC, est la société spécialiste en conception de systèmes électroniques (métiers du matériel, du logiciel et des systèmes embarqués) du groupe Advans. Les compétences d'Elsys couvrent toutes les étapes de la conception matérielle, du développement de logiciels embarqués et de la conception de systèmes, de la faisabilité à la validation finale de systèmes complets. La société est aussi présente en Serbie, sous le nom de Elsys Eastern Europe [4].

L'équipe est constituée par des ingénieurs spécialistes dans tous les métiers de la conception de systèmes électroniques et logiciels :

- Architecture hardware et software de cartes et calculateurs ;
- Électronique de puissance, analogique ;
- Conception d'ASIC, de FPGA et de circuits intégrés ;
- Développement de logiciel embarqué temps réel ;
- Banc de test, intégration, validation, test et industrialisation.

Elsys est certifiée `ISO-9001:v2015` et `EN-9100`, dans le but de fournir les plus haut niveaux possibles de satisfaction à ses salariés, clients et candidats.

Ainsi que dans les autres sociétés du groupe, les ingénieurs de chez Elsys sont suivis par des responsables d'affaires, eux-mêmes venus d'une formation technique, ce qui garanti l'ADN technique du groupe. Le rôle de ces responsables est de garantir le développement professionnel et la progression de carrière des ingénieurs dont ils assurent l'accompagnement.

Depuis sa création, la société a défini la R&D comme un axe stratégique majeur. Elle a reçu le label Société Innovante en 2002 de l'organisme public français ANVAR (désormais Bpifrance). La société appartient également au réseau « Bpifrance Excellence », composé d'entreprises « qui ont soif de liberté, qui innovent et qui ont une envie d'entreprendre sans limite. »

3 Projet ETNA

Dans le cadre du partenariat entre *Elsys Eastern Europe* et ARM, mis en place en 2017, Elsys propose aux clients de la société ARM de bénéficier d'un support technique en conception ainsi que des formations dans le cadre du développement de System on Chip (SoC) intégrant des cœurs ARM.

Afin de former et spécialiser ses ingénieurs, via un exemple concret et réel, à la fois sur la méthodologie de vérification UVM, et sur le développement basé sur une plate-forme, ELSYS Design a mis au point une plate-forme de développement servant d'outil d'autoformation. Dernièrement, cette plate-forme a été mise en avant pour valoriser le partenariat existant avec ARM. Cette plate-forme est nommée ETNA (*ELSYS' Training Not ARM's*), intégrant le besoin de spécialisation et de formation, ainsi que la mise en valeur et l'utilisation des ressources disponibles via le partenariat avec ARM.

Lors de cette formation, les participants apprennent à rendre un composant IP existant compatible avec une architecture basée sur un processeur ARM et son bus, l'intégrer au système, à écrire ses drivers en C, le simuler et l'implémenter sur FPGA. Dans un deuxième temps, un plan de vérification est défini, puis un environnement de vérification UVM est développé afin de mettre à exécution le plan. Enfin une partie de la vérification est intégrée comme logiciel de test pour la plate-forme FPGA.

Cette section se divise ainsi : l'environnement de travail ETNA est présenté dans la Section 3.1, et le travail à réaliser, ainsi que les objectifs attendus dans la Section 3.2.

3.1 L'environnement ETNA

Les premières versions d'ETNA sont basées sur le *DesignStart* du microcontrôleur Cortex-M0 de chez ARM. Il est prévu de mettre en place une version d'ETNA basée sur le *DesignStart* du Cortex-M3 et, éventuellement, sur d'autres cœurs de processeur. *DesignStart* est un paquet RTL (Register Transfer Level) qui contient le processeur ARM intégré a un sous-système AHB (Advanced High-Performance Bus) qui contient d'autres composants (Figure 1), permettant de modifier et mettre en œuvre l'architecture sur un ASIC. Il a été créé par ARM afin de fournir une expérience plus rapide et facile de développement basé sur ses processeurs.

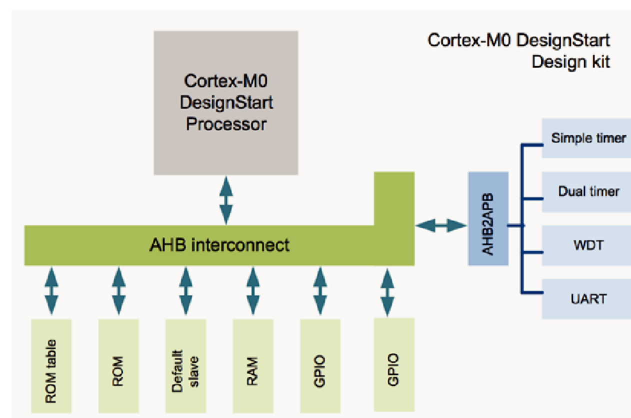


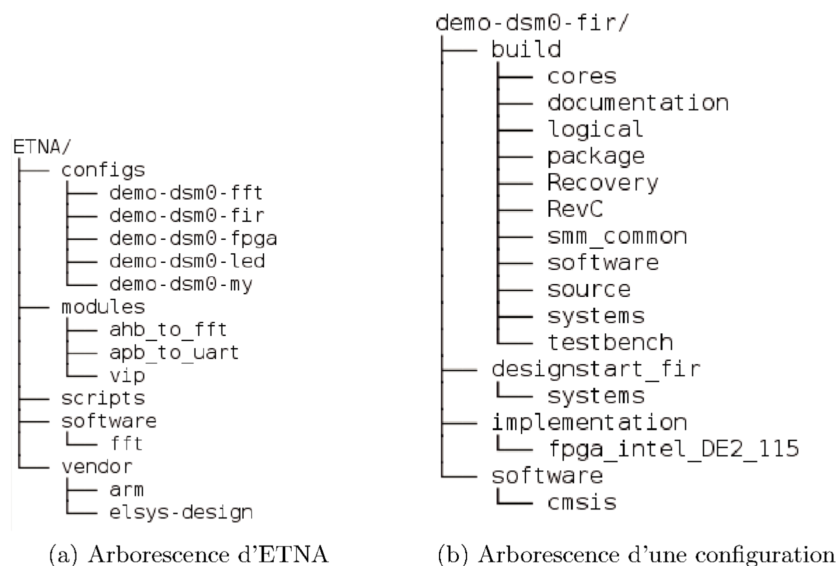
FIGURE 1 – Schéma bloc simplifié du DesignStart Cortex-M0 (Source : ARM Community)

Afin de faciliter son utilisation, ETNA est utilisée dans une machine virtuelle disposant d'un système d'exploitation CentOS 7. Cette machine permet de conserver les configurations des différentes étapes du projet : développement, intégration et vérification. La machine dispose aussi de tous les outils nécessaires à l'avancement du projet : Quartus, ModelSim et des différents utilitaires. Pour gérer les différentes versions, l'avancement et pour permettre le partage du projet, un dépôt git a été mis en place sur GitLab.

Actuellement, la plate-forme ETNA est implémentée sur le kit de développement Altera DE2-115 de chez Terasic. Pour que cette implémentation soit possible, un certain nombre de modifications et adaptations sur le code source du *DesignStart* Cortex-M0 ont dû être fait. Des travaux ont été initiés pour un portage sur Xilinx, mais n'ont pas été finalisés.

La plate-forme DE2-115 dispose d'une puce FPGA Cyclone IV, de chez Altera, de mémoires FLASH (8Mo), SRAM (2Mo) et SDRAM (2×64Mo), de LEDs, d'un écran LCD, de ports d'entrées/sorties et nombreux autres périphériques. L'un des objectifs de ce stage est de rendre disponible tous ces périphériques depuis le processeur ARM. Des images supplémentaires sont fournies dans l'annexe A.

ETNA est composée de 5 dossiers principaux : `vendor`, `modules`, `software`, `script` et `configs`. L'arborescence venue de cette organisation est présentée sur la Figure 2a. Le dossier `vendor` contient les différents composants du projet, à la fois ceux mis à disposition par ARM pour le *DesignStart* du Cortex-M0 et ceux développés par Elsys-Design, comme le module de la transformation de Fourier rapide (FFT) et du filtre FIR (Finite Impulse Response). Le répertoire `modules` contient les modules permettant d'interfacer les composants développés avec les bus AHB et APB (Advanced Peripheral Bus) du *DesignStart*. Dans `software`, l'ensemble de pilotes C et d'autres fichiers utilisés pour le développement des composants de chez Elsys sont disponibles. Finalement, le dossier `configs` contient tous les différents fichiers propres à chaque configuration de la plate-forme.



(a) Arborescence d'ETNA (b) Arborescence d'une configuration

FIGURE 2 – Arborescences dans ETNA

Dans un dossier de configuration, on ne met que les fichiers qui ne pourront pas être utilisés par d'autres configurations, ainsi que des fichiers disponibles dans le répertoire `vendor` qui ont été modifiés pour que la configuration soit réalisable. Aussi dans ce dossier, un `Makefile` doit être mis à disposition pour que la configuration soit automatiquement construite. La construction d'une configuration commence par la copie des fichiers originaux du *DesignStart*, suivie de la copie des fichiers des composants de chez Elsys. Elle se finit par le remplacement de ces fichiers par les fichiers modifiés.

L'arborescence d'une configuration est décrite par la Figure 2b. Dans les dossiers `software` et `designstart_`<config>, les fichiers de chez ARM modifiés doivent être mis. Le répertoire `implementation` contiendra les fichiers spécifiques aux logiciels Quartus et ModelSim. Finalement, le dossier `build` contient la configuration construite, avec la copie de tous les fichiers nécessaires à son fonctionnement.

3.2 Problème posé

Ayant une plateforme assez large permettant l'entraînement sur les plus diverses sujets et technologies, il a été décidé, comme objectif, d'avoir une démonstration avec les entrées et sorties de la carte FPGA. D'entre elles, il a été choisi d'utiliser les entrées et sorties audio. Cette démonstration comprend l'utilisation des modules servant à la capture et au traitement audio depuis le processeur d'ETNA.

Afin d'atteindre l'objectif du stage, 4 tâches principales sont définies. D'abord, le développement d'un IP de filtrage dans le but de traiter les données qui seront captées. Pour cela, des implémentations existantes ont été étudiées, ainsi que des façons efficaces de le faire. Ensuite, l'intégration de cet IP à l'environnement ETNA pour qu'il puisse être accédé depuis le processeur, ce qui comprend le développement d'une interface entre l'IP et le bus, ainsi que l'écriture de son driver. La vérification du filtre, dans le but d'assurer que l'IP suit les spécifications, en utilisant la méthodologie UVM. Finalement, ayant déjà une bonne prise en main de l'environnement de travail, refaire les trois dernières tâches pour un IP de capture audio.

Cependant, lors que la vérification du filtre a été réalisée, il a été décidé d'augmenter la complexité du stage et de le réorienter vers quelque chose de plus audacieux. Ainsi, un nouveau objectif est créé : la mise en œuvre d'un accélérateur cryptographique capable d'exécuter l'algorithme RSA. Cet accélérateur doit être capable de réaliser des chiffrages à 2048 bits à une fréquence de 50 MHz. Pour cela, 3 tâches ont été définies : étudier l'algorithme RSA et les recherches faites dessus afin de réaliser ses calculs de façon plus performante, ainsi que les implémentations déjà existantes ; sa mise en place RTL en envisageant la carte FPGA ; et son optimisation dans le but d'arriver à la performance souhaitée.

Enrichir la plateforme ETNA est le but général du stage, dans l'intention de présenter à ARM les compétences des ingénieurs de chez Elsys Design. Ces deux objectifs permettront alors l'enrichissement souhaité — ainsi que le développement personnel sur les sujets de design FPGA, vérification et sécurité matérielle — et, possiblement, la création d'un IP commercialisable par l'entreprise.

4 Mise en œuvre d'un filtre

Afin de permettre la prise en main de l'environnement ETNA et son enrichissement, la première tâche lors du stage a été le développement d'un module de filtrage numérique. Cette section a pour but de présenter le type de filtre mis en place — les filtres à réponse impulsionnelle finie — dans 4.1, ainsi que son implémentation dans 4.2.

4.1 Filtres à réponse impulsionnelle finie (FIR)

Dans l'électronique, les filtres sont utilisés pour accentuer les signaux à une certaine gamme de fréquence, ainsi que pour atténuer les signaux à d'autres gammes de fréquence. Les filtres ont aussi un gain associé aux fréquences auxquelles ils agissent. Par exemple, si l'on a un signal composé de deux sinusoïdes à l'entrée d'un filtre : l'une en basse fréquence et l'autre en haute fréquence (comme un bruit) ; ce filtre peut être configuré pour avoir un gain très bas pour les hautes fréquences et un gain élevé pour les basses fréquences. De cette façon, le bruit ne sera presque plus perceptible à la sortie.

Les filtres peuvent être classifiés en analogique et numérique. Les filtres analogiques sont faits en utilisant des composants électriques et électroniques, comme des résistances, des condensateurs ou des amplificateurs opérationnels. Les filtres numériques utilisent un calculateur numérique pour faire le traitement des échantillons d'entrée, ainsi que d'autres composants comme des mémoires pour stocker les coefficients. L'un des avantages des filtres numériques est qu'ils sont configurables par logiciel, ce qui les rend facilement réutilisables. De plus, ils sont plus facilement implémentés, et ne souffrent pas avec le vieillissement et les différences de température. Finalement, ils sont plus performants et peuvent fonctionner de différentes façons.

Les filtres numériques sont utilisés dans plusieurs applications, comme la communication sans fil, le traitement d'image et le biomédical pour faire de la suppression ou de l'annulation du bruit. Sans ce type de filtre il ne serait pas possible, par exemple, d'avoir une communication propre, parce que des bruits sont ajoutés lors du passage par le canal.

Ils peuvent être classifiés en deux groupes : les filtres à réponse impulsionnelle finie (FIR) et les filtres à réponse impulsionnelle infinie (IIR). On ne s'intéresse qu'au premier groupe, les filtres du type IIR ne seront pas traités dans ce document.

Les filtres FIR sont des filtres dont la réponse impulsionnelle est limitée en durée, autrement dit, sa réponse est zéro à l'infini. Cela arrive parce qu'ils ne dépendent pas des sorties passées, ce qui les rend naturellement stables, d'où sa grande popularité.

Son fonctionnement est assez simple : sa sortie est la somme pondérée des plus récentes valeurs d'entrée, ce qui est montré sur l'équation 1. On peut noter que son calcul est le même que celui d'une convolution discrète. La quantité de coefficients, ainsi que des valeurs d'entrée prises en compte, est définie par la taille du filtre.

$$y[n] = \sum_{i=0}^{N} c_i \cdot x[n-i] \tag{1}$$

Un filtre FIR peut être implémenté en utilisant des délais, des multiplicateurs et des additionneurs. Un exemple de ce filtre avec 4 coefficients est montré sur la Figure 3 en utilisant une architecture simple.

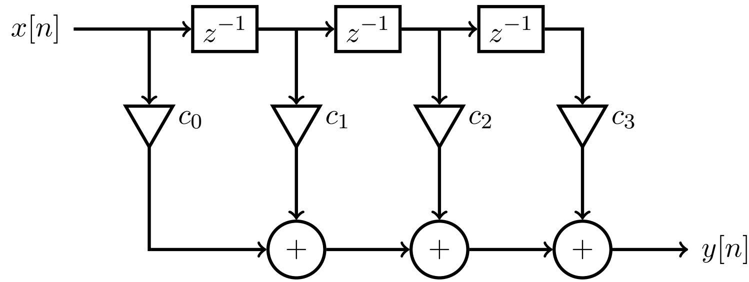


Figure 3 – Architecture usuelle d'un filtre FIR : exemple avec 4 coefficients

On peut noter que le filtre, mis en œuvre de cette façon, croît linéairement selon sa taille, en ajoutant un nouveau délai, un nouveau multiplicateur et un nouveau additionneur à chaque nouveau coefficient. Avec cette architecture, on peut avoir le filtre avec n'importe quelle configuration (passe-haut, passe-bas, entre autres) et taille. En revanche, elle n'est pas optimale pour une implémentation en matériel à cause de son chemin critique qui passe pas tous les additionneurs. Pour avoir des filtres plus performants, d'autres architectures doivent être utilisées, ce qui sera discuté dans la section 4.2.

4.2 Développement du module FIR

Dans cette section, les solutions mises en places pour le filtre FIR seront présentées. Elle se divise ainsi : l'architecture décidée sera présentée et discutée dans 4.2.1, ensuite, une brève présentation de l'implémentation faite en C pour la validation du filtre en 4.2.2 et, finalement, sa mise en œuvre en RTL en 4.2.3.

4.2.1 Architecture systolique

Les architectures systoliques consistent à des aboutements de cellules identiques contenant des structures répétitives de l'architecture originale. De cette façon, des circuits complexes peuvent être obtenus avec un effort de conception réduit [5]. Dans cette architecture, chaque cellule reçoit des données des cellules voisines, effectue un calcul, puis transmet les résultats à des cellules voisines un temps de cycle plus tard. Seules les cellules à la frontière du réseau communiquent avec l'extérieur (dans ce cas, les entrées et sorties du filtre).

En appliquant cette solution au filtre FIR, on peut noter que les multiplications et additions se répètent à chaque coefficient. De cette façon, on peut les regrouper en une seule cellule, qui reçoit une entrée à être multipliée par le coefficient, et autre qui lui sera ajoutée. De plus, pour que toutes les cellules reçoivent les bonnes données au bon moment, trois registres leur sont ajoutés. La cellule finale est indiquée sur la Figure 4. Finalement, il faut enchaîner ces cellules de façon à avoir la taille du filtre souhaité. Autrement dit, le signal y_{out} d'une cellule sera l'entrée y_{in} de la cellule suivante, ainsi que le signal x_{out} sera l'entrée x_{in} de la suivante.

Les registres dans les cellules permettent de diviser le chemin critique que l'on avait avant, ce qui permet aussi l'augmentation de la fréquence avec laquelle le filtre fonctionne. Par contre, cette architecture entraîne aussi des inconvénients : l'addition des registres crée un délai au début de son fonctionnement. Quand le filtre est initialisé, il n'a des valeurs que dans sa

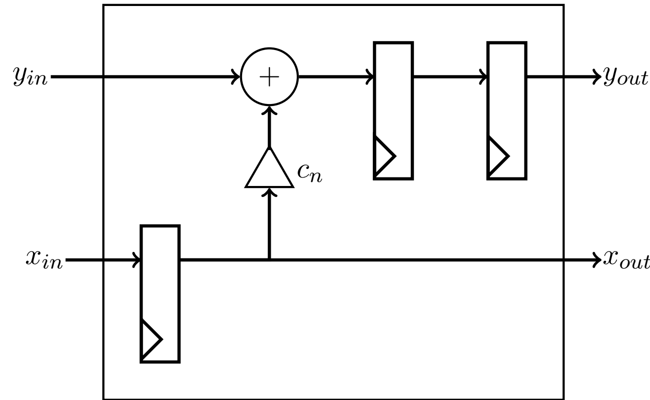


FIGURE 4 – Cellule de l'architecture systolique du FIR

première cellule (celle qui reçoit les entrées). Il faut alors attendre deux fois la taille du filtre (numéro de coefficients et, par conséquence, de cellules) cycles d'horloge pour avoir une valeur valide à sa sortie. Cette attente peut ne pas être importante, si l'on arrive à bien augmenter la fréquence.

4.2.2 Implémentation en haut niveau

Ayant décidé l'architecture à utiliser, un filtre FIR a été mis en place en C, en utilisant le point fixe, pour que, ensuite, le filtre en HDL (Hardware Description Language) puisse être validé. Ce filtre a été validé en regardant ses valeurs de sortie lors d'un impulse à l'entrée : elles doivent avoir les mêmes valeurs que ses coefficients. Ensuite, avec un échelon à l'entrée, la sortie doit être la somme des coefficients du filtre. Finalement, avec ces deux fonctionnements validés, un signal composé par plusieurs sinusoïdes a été mis à l'entrée et, selon la configuration du filtre, la sortie doit être le composant de haute ou basse fréquence de ce signal. Ces mêmes étapes de validation ont été utilisées pour la validation du filtre en HDL.

Le fonctionnement du filtre en C est montré dans l'Annexe C. Dans cet exemple, le signal d'entrée est composé d'une sinusoïde de fréquence 1Hz et amplitude égale à 3, et d'une sinusoïde de fréquence 600Hz et amplitude 1. Le filtre, quand configuré pour ignorer les signaux de fréquence plus haute que 200 Hz, a le résultat montré sur la Figure 4c, où le signal de sortie n'est composé que par la sinusoïde de basse fréquence. Quand on le configure comme un filtre passe-haut, on a la sortie de la Figure 4b.

Sa mise en œuvre est importante pour la validation des modules faits en RTL. Elle montre le fonctionnement que l'on veut d'un filtre FIR et on peut alors comparer sa sortie avec celle du filtre en RTL pour savoir s'il fonctionne comme souhaité.

4.2.3 Implémentation RTL

En connaissant le fonctionnement souhaité du filtre, et ayant une façon de le valider, sa mise en œuvre dans des langages de description de matériel est possible. Alors, on a décidé de le faire en utilisant le langage VHDL pour le design, et le SystemVerilog pour la vérification.

Deux modules ont été mis en places : le premier avec la description de la cellule de l'architecture systolique (présenté en 4.2.1) et le deuxième, le *top-level*, qui, en aboutant plusieurs cellules, met en place effectivement le filtre. Son diagramme bloc est montré sur la Figure 5.

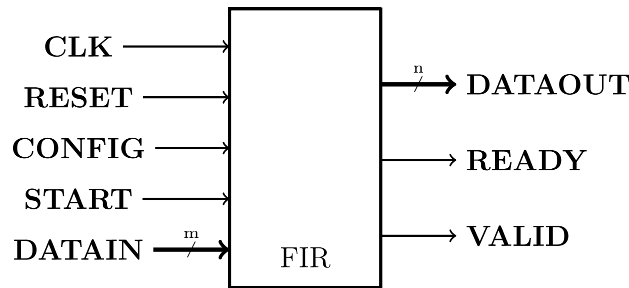


FIGURE 5 – Diagramme de bloc des entrées et sorties du module FIR implémenté

Signaux Le signal d'entrée CONFIG sert à configurer le filtre, c'est-à-dire modifier la valeur de ses coefficients, pour que l'on puisse changer son fonctionnement. Lors qu'il est mis à 1, le filtre arrête le filtrage (s'il est en train de le faire), et reçoit les valeurs des coefficients par le signal DATAIN. C'est aussi à travers ce signal-ci que le filtre reçoit les échantillons utilisés lors du filtrage.

Les calculs de filtrage sont commencés quand le signal START est mis à niveau logique haut. La description de tous les signaux est présente dans la Table 1.

Signal	**Description**
CLK	Signal d'horloge.
RESET	Signal de réinitialisation actif haut.
CONFIG	Signal indiquant une étape de configuration du filtre.
START	Signal utilisé pour initialiser le filtrage.
DATAIN	Bus en taille configurable par où le filtre reçoit les données, à la fois les coefficients lors d'une configuration et les échantillons lors d'un filtrage.
DATAOUT	Bus en taille configurable où le résultat du filtrage est écrit.
READY	Signal indiquant que le dernier transfert a été fini.
VALID	Signal qui indique la validité des données en sortie .

TABLE 1 – Description des entrées et sorties du module FIR

Tous les signaux sont à 1 bit, sauf DATAIN et DATAOUT, dont la taille doit être configurée avant la synthèse du filtre. Cela arrive parce que le filtre est fait en utilisant l'arithmétique virgule fixe, ce que veut dire qu'il faut définir la quantité de bits pour la partie décimale, ainsi que pour la partie entière. Le module a été créé pour être le plus réutilisable possible, alors la taille de ces signaux n'a pas été fixée. De plus, la taille du filtre, c'est-à-dire le nombre de coefficients, est aussi modifiable et doit aussi être choisie avant la synthèse en sachant que augmenter sa taille augmentera aussi le délai.

Machine à états Ce module fonctionne avec une machine à quatre états : *IDLE*, *CONF*, *FILT* et *DONE* (Figure 6). Dans l'état *IDLE*, rien ne se passe et l'IP attend. Lors de la réception d'un niveau haut du signal CONFIG, le filtre passe à l'état *CONF*, où il restera jusqu'à recevoir tous ses coefficients. La configuration étant finie, il passe à l'état *DONE*, où la sortie READY est mise à 1 et, ensuite, il retourne à l'état *IDLE*. Le temps qu'il reste dans l'état *CONF* est défini par la taille du filtre.

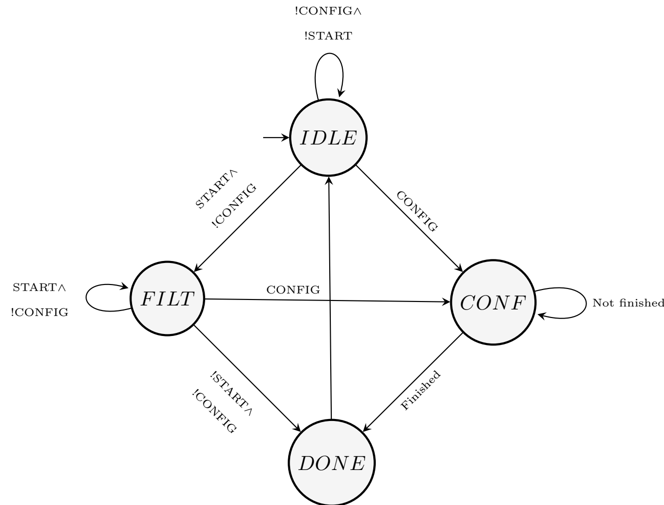


FIGURE 6 – Machine à états du module FIR

Depuis l'état *IDLE*, quand l'entrée START est mise à 1, la machine passe à l'état *FILT*, où le filtrage est fait. La machine reste à cet état jusqu'à recevoir un signal CONFIG égal à 1, en indiquant une nouvelle configuration, où un 0 sur le signal START, ce qui indique la fin du filtrage. Des formes d'onde de l'utilisation du filtre sont disponible dans l'annexe B.

La simulation d'un filtre de taille 21 mis en place avec deux configurations différentes peut être vue dans l'Annexe D. Il est notable que, au début des deux simulation, il y a une étape de configuration (lorsque le signal CONFIG est égal à 1), où les coefficients du filtre sont envoyés par le signal DATAIN, afin d'avoir une configuration passe-bas ou une configuration passe-haut. Ensuite, le signal CONFIG passe à 0 et le filtre indique que l'étape de configuration est finie en mettant le signal READY à 1. Le filtre commence à recevoir alors les échantillons à filtrer sur DATAIN et, dans quelques cycles d'horloge, le signal VALID est mis à 1, indiquant que les données sur DATAOUT sont valides.

Le comportement de l'IP est équivalent à celui du filtre fait en C : un filtre passe-bas réduit l'amplitude des hautes fréquences et le passe-haut réduit l'amplitude des basses fréquences. Il est possible, donc, de passer à la prochaine étape du stage, l'intégration de cet IP au projet ETNA en utilisant le bus AHB de chez ARM.

5 Intégration au projet ETNA

Après avoir mis en place l'IP de filtrage, il est nécessaire de faire son intégration au projet ETNA. C'est-à-dire de l'intégrer au microcontrôleur Cortex-M0 via le bus AHB de chez ARM, afin de le rendre accessible depuis le microprocesseur. Alors, cette section présente le protocole de communication AHB dans la Section 5.1 et le travail d'intégration dans 5.2.

5.1 Advanced High-Performance Bus (AHB)

Cette sous-section présente le protocole AHB de chez ARM et est divisée ainsi : les signaux utilisées par un maître seront décrits en 5.1.1 et ceux utilisées par les esclaves en 5.1.2. Finalement, le fonctionnement d'un transfert est décrit en 5.1.3.

Le protocole de bus AHB fait partie de la famille de protocoles format ouvert AMBA (Advanced Microcontroller Bus Architecture) de chez ARM. Il a été développé pour les modules de haute performance comme les processeurs, les mémoires sur puce et les interfaces des mémoires. Le protocole offre un support à plusieurs maîtres et fourni des opérations de haute bande passante. Cependant, la version de DesignStart utilisée par ETNA dispose de la version *Lite* du protocole AHB, qui ne supporte qu'un seul maître.

AHB-Lite met en place des fonctionnalités nécessaires pour les systèmes de haute performance, ce qui inclut les transferts en burst, le fonctionnement en un seul front d'horloge (front montant), une implémentation non-tristate et un bus de données configurable de 64 a 1024 bits.

La Figure 7 montre le diagramme bloc d'un bus AHB-Lite avec trois esclaves. L'interconnexion du bus est composée d'un décodeur d'adresse et un multiplexeur esclave-à-maître. Le décodeur prend en entrée l'adresse envoyée par le maître pour que le bon esclave puisse être choisi et les bonnes sorties sélectionnées via le multiplexeur.

Les esclaves n'ayant pas besoin d'une haute bande passante peuvent être inclus dans le système AHB, mais pour des raisons de performance ils sont plutôt mis dans le sous-système APB, un autre bus présent dans le microcontrôleur fait pour des transferts en basse bande passante. Dans *DesignStart*, le sous-système APB est mis a disposition dans le sous-système AHB.

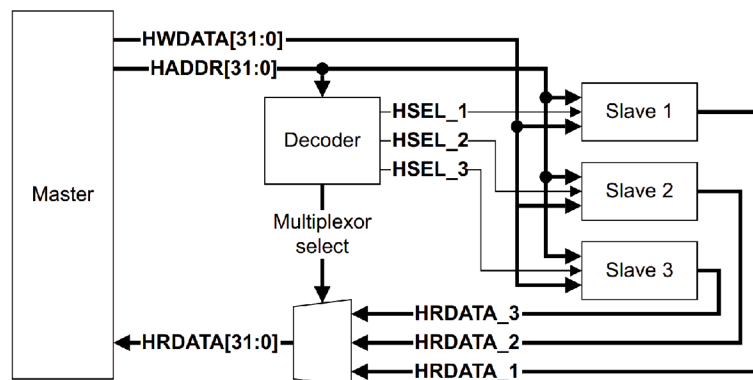


Figure 7 – Diagramme bloc du bus AHB-Lite (Source : spécification AMBA 3 AHB-Lite)

5.1.1 Maître AHB-Lite

Dans le protocole AHB-Lite, le maître est responsable de fournir les adresses et les données nécessaires pour la réalisation des écritures et lectures. Son interface est illustrée par la Figure 8.

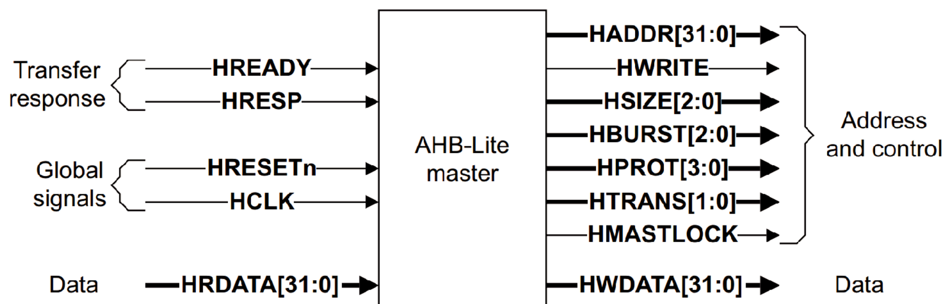


Figure 8 – Interface du maître (Source : spécification du protocole AMBA 3 AHB-Lite)

Sur le côté gauche de l'interface, on trouve les signaux d'entrée du maître. De même, les signaux de sortie sont à sa droite. Les signaux HCLK et HRESETn sont globaux et viennent alors du processeur Cortex-M0. Il est important de souligner que le signal HRESETn est le seul signal actif bas du protocole. Tous les signaux d'un maître AHB-Lite sont décrits dans la Table 2.

Signal	**Source**	**Description**
HCLK	Global	Signal d'horloge responsable de synchroniser tous les transferts du bus, qui se passent sur son front montant.
HRESETn	Global	Signal de réinitialisation, actif à l'état bas.
HADDR	Maître	Bus de 32 bits indiquant l'adresse à laquelle le transfert doit avoir lieu.
HWRITE	Maître	Signal qui définit la direction du transfert. L'opération est une écriture quand ce signal est égal à 1 et une lecture quand égal à 0.
HWDATA	Maître	Bus de 32 bits utilisé pour les transferts d'écriture.
HRDATA	Esclave	Bus de 32 bits utilisé pour les transferts de lecture.
HSIZE	Maître	Signal utilisé pour indiquer la taille du transfert, allant jusqu'à 1024 bits.
HBURST	Maître	Signal indiquant si le transfert courant est unique ou partie d'un burst. Des bursts de taille 4, 8 et 16 sont supportés.
HTRANS	Maître	Signal de classification du transfert.
HPROT	Maître	Signal fournisseur d'informations du niveau de protection du transfert.
HMASTLOCK	Maître	Signal indiquant l'atomicité des transactions.
HREADY	Esclave	Signal de périphérique prêt.
HRESP	Esclave	Signal qui indique si le transfert a échoué ou a été réussi.

Table 2 – Description des signaux d'entrées et sorties du maître AHB-Lite

Outre que utilisé par les esclaves, le signal HADDR est utilisé par le décodeur du bus pour sélectionner le bon esclave cible et les bonnes entrées du maître. Le signal HTRANS peut avoir 4 valeurs : *IDLE*, utilisée quand il n'y a pas de transfert de données ; *BUSY*, pour créer des états d'attente au milieu des bursts ; *NONSEQUENTIAL* pour signaler le début d'un burst ou des transferts uniques ; et *SEQUENTIAL* indiquant les transferts séquentiels d'un burst.

Les esclaves ne sont pas obligés d'utiliser les signaux de contrôle de protection (HPROT) et de transferts bloqués (HMASTLOCK). Alors, ces signaux n'ont pas été utilisés lors du stage[1]. Les signaux d'entrée issus des esclaves seront discutés dans la section suivante (5.1.2).

5.1.2 Esclaves AHB-Lite

Les modules esclaves, dont l'interface est présentée sur la Figure 9, sont responsables de répondre à tous les transferts leur étant dirigés, initialisés par le maître du système. Ils utilisent le signal de sélection HSELn, issu du décodeur, pour savoir si le transfert leur est destiné ou non.

Les esclaves doivent signaler au maître si le transfert a été réussi, en mettant le signal HRESP à 0, ou échoué, en le mettant à 1. De plus, ils peuvent étendre les transferts en utilisant le signal HREADYOUT. Ce signal est utilisé pour indiquer au maître qu'un transfert a été fini, en le mettant au niveau haut. Alors, si ce signal est mis à 0, le transfert sera étendu — c'est-à-dire qu'il fera attendre le maître jusqu'à ce qu'il soit prêt et valide.

Tous les autres signaux issus du maître du bus ont été présentés dans la section 5.1.1.

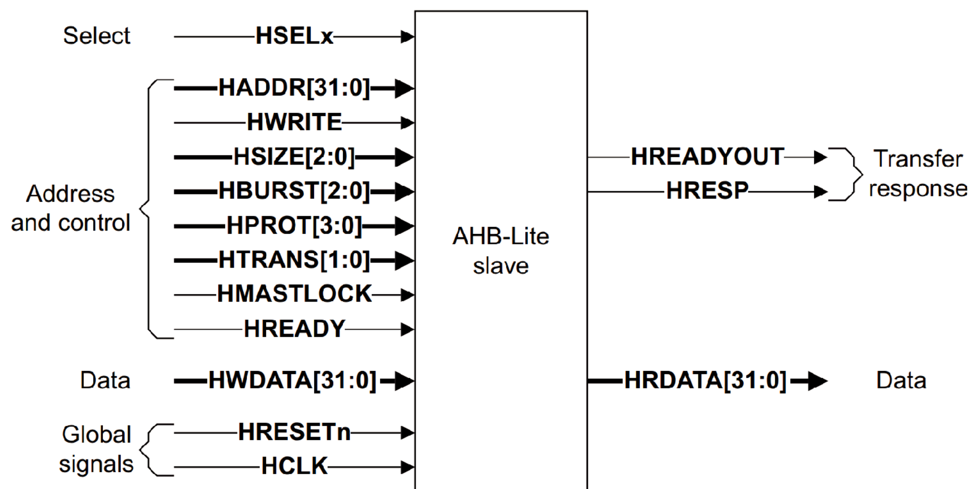


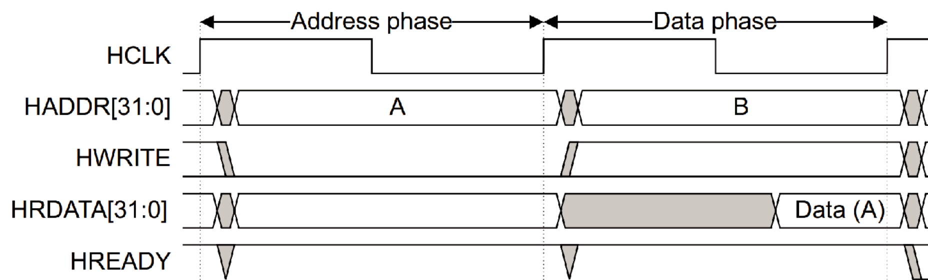
FIGURE 9 – Interface d'un esclave (Source : spécification du protocole AMBA 3 AHB-Lite)

5.1.3 Modes d'opération du protocole AHB-Lite

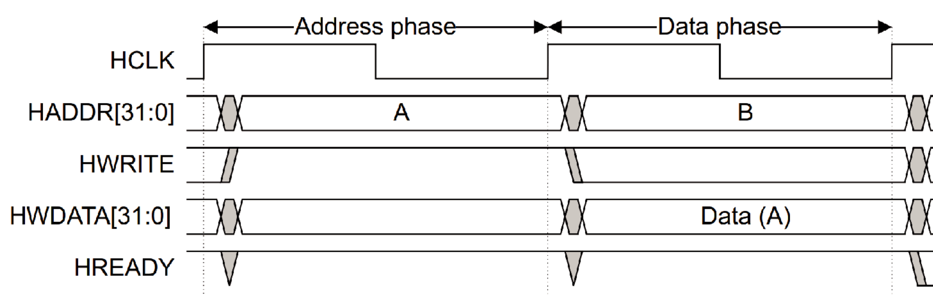
Un transfert du protocole AHB-Lite est fait en deux phases. La première est nommée *Address Phase*, et c'est où le maître envoie, sur HADDR, l'adresse où l'écriture ou la lecture sera faite. Cette phase dure un cycle d'horloge mais peut être étendue. La deuxième phase s'appelle *Data Phase*, et c'est où les données du transfert seront écrites sur ses respectifs signaux, HWDATA ou HRDATA. La durée de cette phase est contrôlée par le signal HREADY.

Comme indiqué dans la section 5.1.1, le signal HWRITE définit la direction du transfert. S'il est mis à 0, c'est une lecture qui sera faite. Quand il est à 1, c'est un transfert d'écriture. Le fonctionnement d'une écriture est illustré sur la Figure 10a et celui d'une lecture sur la Figure 10b.

[1] Pour plus d'informations, référer à la spécification du protocole disponible sur le site d'ARM.



(a) Transfert de lecture



(b) Transfert d'écriture

FIGURE 10 – Fonctionnement des transferts (Source : spécification AMBA 3 AHB-Lite)

Brièvement, ce qui se passe lors d'un transfert est le suivant : dans un premier cycle d'horloge, le maître écrit l'adresse et les signaux de contrôle (notamment HWRITE) sur le bus. Dans le cycle suivant, le esclave sélectionné par le décodeur d'adresses échantillonne ces signaux et commence à traiter et à envoyer la bonne réponse. Finalement, dans le troisième cycle, la réponse sera échantillonnée par le maître.

Cet exemple est celui le plus simple possible, qui montre comme les deux phases du protocole se produisent dans des cycles d'horloge différents. La phase d'adressage d'un transfert se passe lors de la phase de données du transfert précédent. Ce fonctionnement est fondamental pour garantir l'essence pipelinée du bus et les hautes performances desquelles il est capable.

5.2 Intégration avec le système ARM DesignStart

Cette section parle de l'intégration du module précédemment présenté avec *DesignStart* Cortex-M0, en utilisant le bus AHB. Pour tous les autres modules développés ultérieurement, l'intégration peut être faite en suivant les mêmes démarches, avec des adaptations propres à chaque cas.

Pour intégrer le module développé au *DesignStart* Cortex-M0, il faut, d'abord, mettre en place une interface entre lui et le bus AHB. Cette interface sera, ensuite, enveloppée dans un autre module, pour faire sa connexion avec le filtre. L'interface créée est présentée dans la Section 5.2.1. Ensuite, l'écriture du driver permettant son accéss depuis le processeur est fait, et sa présentation est faite dans la Section 5.2.2

5.2.1 Interface

Le rôle de l'interface est de permettre l'utilisation du FIR à partir des signaux du bus AHB. Donc, elle reçoit en entrée et envoie en sortie les mêmes signaux présents dans le protocole AHB, ce qui est indiqué sur la Figure 11.

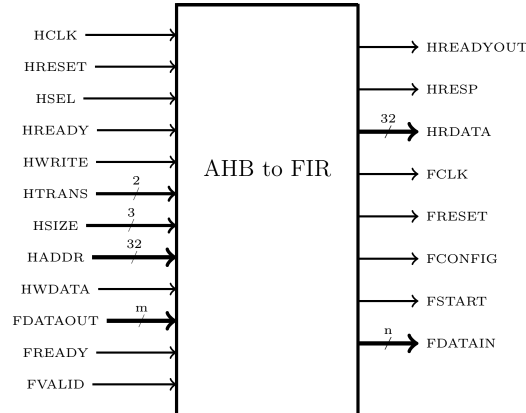


FIGURE 11 – Diagramme de bloc des entrées et sorties de l'interface *AHB to FIR*

Pour gérer cette communication et les divers modes d'opérations du bus, une machine à états a été mise en place. Cette machine à états dispose de quatre états : *IDLE*, *WRITE*, *READ* et *ERROR* (Figure 12).

L'interface est initialisée dans l'état *IDLE*, où rien n'est fait jusqu'à la réception d'un transfert non-idle (HTRANS $\neq$ "00"). Pour aller à l'état *WRITE*, quatre conditions doivent être remplies (en plus du transfert non-idle) : le module est sélectionné (HSEL $= 1$), le dernier transfert est fini (HREADY $= 1$), un transfert est du type écriture (HWRITE $= 1$) et l'adresse du transfert doit être valide. Ces conditions sont représentées par la transition *Write transfer* sur l'automate. Les conditions de la transition *Read transfer* sont les mêmes, à l'exception du type de transfert, qui doit être une lecture (HWRITE $= 0$).

Le dernier état est l'état d'erreur (*ERROR*). Cet état sert à indiquer au maître qu'il a eu un problème avec le traitement du dernier transfert. Le problème peut être une adresse inexistante, une opération interdite — l'écriture dans une adresse de lecture seulement ou une lecture dans une adresse d'écriture seulement — ou une configuration de transfert pas acceptée par l'interface.

Comme le protocole AHB a une nature pipelinée, plusieurs transferts peuvent être enchaînés. C'est-à-dire que plusieurs transferts peuvent être faits en séquence (dans des cycles consécutifs d'horloge). Alors, si l'on est dans l'état *READ*, il est possible de passer à l'état *WRITE* au cycle suivant, par exemple, sans passer par *IDLE*. La seule exception est l'état *ERROR*, puisque la signalisation d'une erreur prend toujours deux cycles d'horloge, en retardant d'un cycle le traitement d'un transfert qui vient juste après. Tout cela est représenté dans la machine à états avec des transitions entre tous les états.

Avec l'interface prête, il est possible de créer le module qui sera instancié dans le sous-système du bus AHB présent dans DesignStart. Ce module a été appelé AHB_FIR, et possède tous les signaux d'entrée et sortie d'un esclave, présentés dans la section 5.1.2. La plage d'adresse choisie pour ce module est de 0x00100000 jusqu'à 0x0010FFFF.

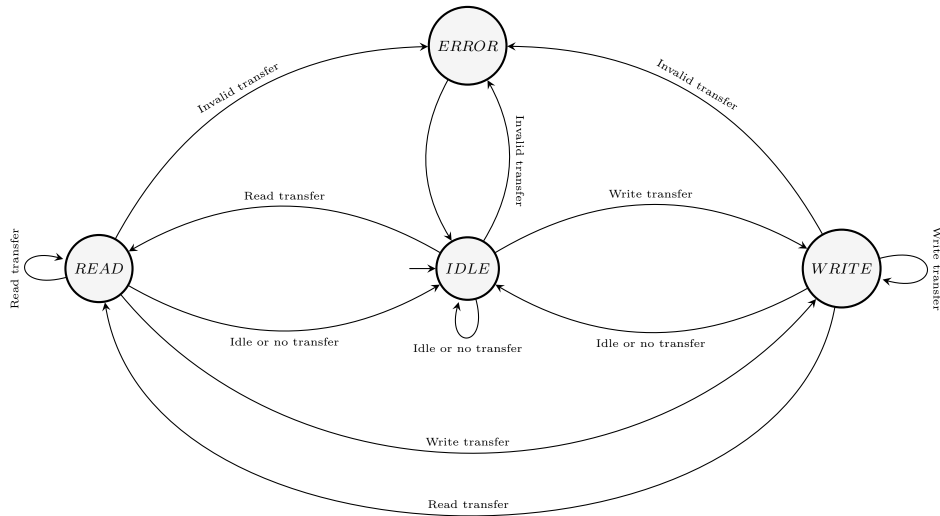


FIGURE 12 – Machine à états de l'interface entre le filtre et le bus

5.2.2 Driver

La dernière étape avant de pouvoir utiliser et visualiser les signaux de sortie est l'écriture d'un driver correspondant au module. Ce driver, réalisé en C, a trois objectifs principaux : la définition des adresses des différents registres internes au module développé ; la définition des fonctions utiles pour pouvoir utiliser le filtre (fonctions d'écriture, lecture et configuration) et la réalisation d'un test du module.

Le driver a deux fonctions principales : l'une qui configure les coefficients et l'autre qui envoie les échantillons à filtrer. Les deux fonctions reçoivent en paramètre un tableau de flottants avec les données à envoyer. Ces données seront d'abord transformées en virgule fixe pour qu'elles puissent êtres envoyées au filtre. La synchronisation de l'envoi avec la fréquence d'échantillonage du filtre est faite avec des interruptions du processeur.

Après la définition du driver, il a été fait des tests pour le valider. Les tests consistent en initialiser le filtre dans différentes configurations, en définissant le type de filtrage qui sera fait, puis une écriture successive des échantillons d'un signal qui sera filtré dans le respectif registre. En regardant les signaux de sortie du filtre, il est possible de vérifier le bon fonctionnement du driver et de l'intégration faite. Les résultats des tests sont disponibles dans l'Annexe E.

Une fois ceux-ci validés, il est temps d'appliquer la méthodologie de vérification UVM afin d'assurer que tous les aspects de la spécification sont respectés.

6 Vérification

La vérification est l'étape du processus de conception matérielle où il est vérifié que l'implémentation met en œuvre correctement la spécification. C'est la tâche la plus longue du développement matériel et qui a, par conséquent, une grande importance à la qualité, le temps et le coût de production. La vérification est essentielle pour minimiser les risques et augmenter la fiabilité des systèmes.

Les spécifications fonctionnelles d'un système sont celles qui définissent les fonctions d'un système ou de ses composants, décrites en fonction de ses entrées et sorties. L'équipe de conception traduit le texte définissant les fonctionnalités en code RTL, ce qui crée une marge d'erreur d'interprétation du texte. La vérification fonctionnelle sert alors pour garantir qu'aucune erreur n'a été faite, soit d'interprétation, soit de conception.

Cette section se divise ainsi : la Section 6.1 présente la méthodologie de vérification UVM, la Section 6.2 présente l'architecture de vérification décidée, lors que la Section 6.3 parlera des résultats de la vérification.

6.1 Universal Verification Methodology (UVM)

L'UVM est une méthodologie de vérification fonctionnelle orientée essentiellement aux simulations. Sortie en 2011, l'UVM est devenue l'une des plus importantes méthodologies de vérification qui utilisent le langage SystemVerilog, grâce à l'utilisation de pratiques de la programmation orientée objet, telle que la réutilisation des composants. Le but est de faciliter la création des environnements de test et, par conséquent, améliorer la productivité.

SystemVerilog est un langage très grand et complexe, et l'UVM est une très grande librairie de ce langage, où des classes de base peuvent être trouvées. En les donnant, l'UVM fournit un moyen d'avoir un environnement modulaire où chaque composant a une responsabilité définie. La librairie fournit aussi des façons de customiser les composants importés, tout en fournissant des lignes directrices pour assurer la cohérence de l'environnement.

La vérification est basée sur un document qui s'appelle plan de vérification. Ce document est extrait de la spécification du système, en traduisant toutes les informations en fonctionnalités qui seront vérifiées. La vérification ne finit que quand toutes les fonctionnalités présentes dans le plan de vérification ont été suffisamment testées. Le plan n'est pas définitif, il doit être changé lors du processus de vérification, mais il est impératif qu'il soit bien écrit et complet.

Cette section est divisée ainsi : dans la sous-section 6.1.1 les classes de l'environnement de vérification UVM sont présentées et les métriques utilisées dans 6.1.2.

6.1.1 Classes UVM

L'UVM est une librairie SystemVerilog qui fournit des classes de base afin de permettre l'implémentation d'un environnement de vérification. Les principales classes, qui seront visitées au long de cette sous-section, sont : le *Sequencer*, le *Driver*, le *Monitor*, le *Scoreboard* et l'*Agent*. La Figure 13 présente une disposition typique des environnements UVM.

Les composants se communiquent entre eux et avec le DUT (*Design Under Test*) en utilisant des transactions. Une transaction contient des entrées et des sorties, qui seront traduites en signaux d'entrées et sorties du DUT. Pourtant, dans la plupart des cas, une seule transaction n'est pas suffisante pour créer le scénario de test souhaité, ce qui rend nécessaire la création d'un ensemble de transactions. Une série de transactions est alors appelé séquence.

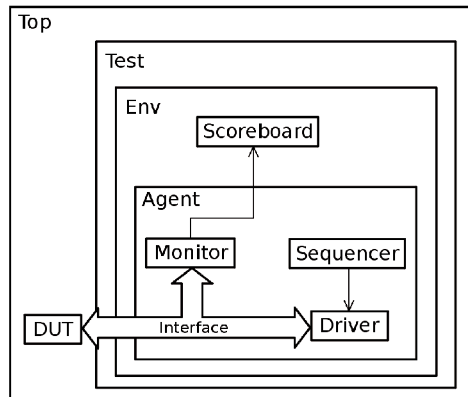


FIGURE 13 – Schéma-bloc d'un environnement UVM typique

Le *Sequencer* est le composant responsable de générer des séquences. Les transactions peuvent être définies par l'utilisateur, de façon aléatoire ou combinées de plusieurs façons. Il est aussi responsable de la configuration et l'initialisation de l'environnement de test et du DUT. Les séquences sont envoyées à un *Driver*.

Le *Driver* est responsable de la communication de l'environnement de tests vers le DUT. Il reçoit les transactions d'un *Sequencer* et les transforme en signaux au niveau des portes. Il est aussi responsable de la synchronisation entre le *Sequencer* et le DUT, afin d'assurer que les transactions soient reçues et envoyées au bon moment. Cette interaction est observée par un autre composant, le *Monitor*, dont le principal objectif est la traduction des signaux de sortie en information pertinente aux autres composants — des transactions.

Tous ces composants sont regroupés dans un autre composant, l'*Agent*. Un *agent* est considéré actif quand il insère des données dans le DUT — il a, par conséquent, un *Sequencer* et un *Driver* — et passif quand il ne fait qu'observer les entrées et sorties avec un *Monitor*.

L'objectif du *Scoreboard* est la vérification du fonctionnement approprié du design à niveau fonctionnel. C'est-à-dire qu'il vérifie si le DUT met en place — ou non — les fonctionnalités décrites sur le plan de vérification. Il reçoit les entrées du DUT, fait une prédiction des valeurs de sortie, et les compare avec les valeurs sortantes du DUT. C'est un élément crucial de l'environnement puisque c'est lui qui fait le calcul des métriques intéressantes à la vérification.

6.1.2 Métriques de vérification

Afin de savoir quels scénarios ont été testés, lesquels il reste tester et si la vérification est complète, il est nécessaire l'utilisation des métriques de vérification. Il existe plusieurs métriques et le choix dépend du DUT à vérifier, de la méthodologie utilisée et des logiciels disponibles. L'UVM se sert, surtout, des métriques de couverture fonctionnelle.

La couverture fonctionnelle est la métrique qui permet la perception de combien du système a été testé par l'environnement. Elle est définie par l'utilisateur en utilisant des « points de couverture », où on définit certaines conditions à être couvertes avant la fin de la simulation. Ces conditions peuvent être, entre autres, des valeurs, des séquences ou des combinaisons.

Au long de la simulation, quand les conditions des points de couverture sont atteintes, ils sont « couverts ». Après avoir fait une certaine quantité de tests, il est possible de générer un rapport pour analyser quelles fonctionnalités ont été couvertes et de planifier la suite afin de couvrir les fonctionnalités manquantes.

Ayant une façon de suivre la couverture fonctionnelle, il faut savoir si le système exécute bien ces fonctions. Pour cela, deux méthodes peuvent être utilisés : des vérifications basées sur des assertions ou des modèles. Les assertions sont une façon d'insérer dans le code des comportements légaux ou illégaux, qui seront signalés comme réussis ou échoués. Elles peuvent être utilisées pour garantir que, par exemple, pour une entrée définie, la sortie doit être toujours une sortie définie. L'autre façon de le faire est d'utiliser un modèle. Il s'agit de la création d'un modèle abstrait ou partiel qui reproduit le comportement souhaité. De cette manière, il est possible de comparer le comportement de ce modèle avec celui du système testé.

6.2 Architecture mise en place

Le filtre a été développé en envisageant son intégration avec le microcontrôleur Cortex-M0. Pour cela, l'architecture choisie a aussi été réfléchie pour qu'elle puisse être utilisée lors que le DUT a été intégré. Alors, il a été décidé d'utiliser trois agents différentes : un actif et deux passifs. L'architecture est illustrée sur la Figure 14.

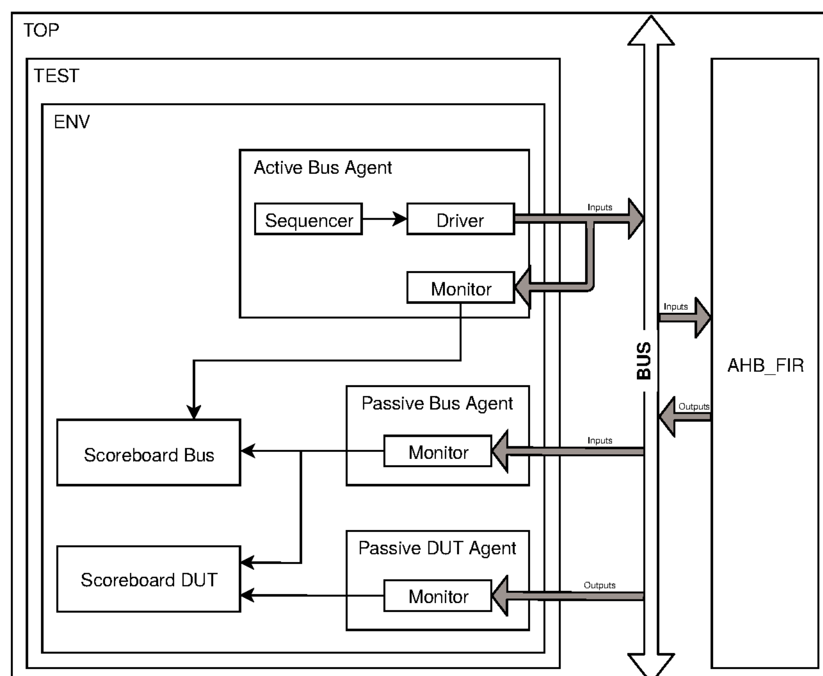


FIGURE 14 – Schéma-bloc de l'environnement UVM mis en place pour la vérification du filtre

L'agent actif est utilisé pour créer les séquences de test souhaitées et les envoyer au DUT. De plus, cet agent surveille les données envoyées au DUT, afin de les utiliser dans un *Scoreboard*. La création des séquences est faite avec un *Sequencer* et leurs envoi et surveillance avec un *Driver* et un *Monitor*, respectivement.

Les deux agents passifs sont composés par un *Monitor* chacun. L'un entre eux est utilisé pour surveiller les entrées du DUT et est appelé *Passive Bus Agent* (PBA), et l'autre surveille ses sorties et s'appelle *Passive DUT Agent* (PDA).

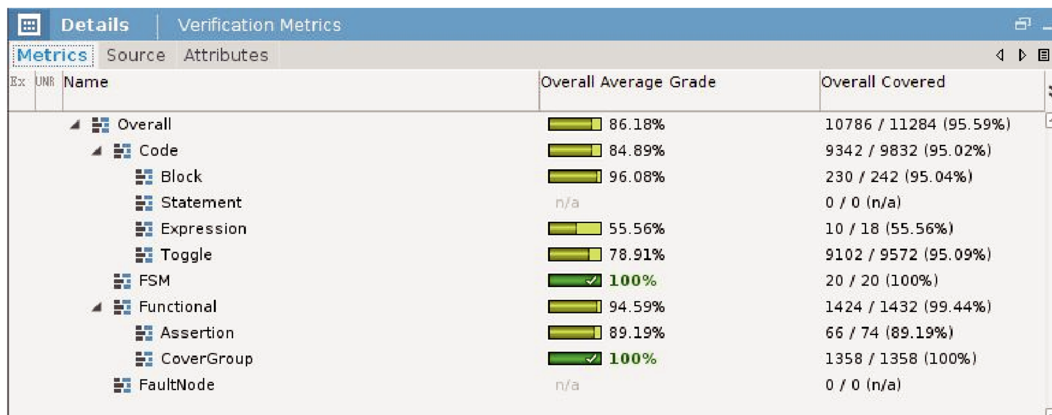
Deux scoreboards sont utilisés afin de faire la validation du système. Le premier, appelé *Scoreboard Bus*, reçoit en entrée les données des *Monitors* de l'agent actif et de l'agent PBA. Il vérifie que le DUT reçoit les mêmes séquences envoyées par l'agent actif. Cela est utilisé afin d'assurer que le bus de communication entre l'agent et le DUT n'est pas fautif.

Le *Scoreboard DUT* reçoit en entrée les données qui rentrent effectivement dans le DUT et ses sorties. Son but est de vérifier que les sorties du DUT correspond aux sorties attendues. Pour cela, deux méthodes sont utilisées : des comparaisons avec un modèle abstrait et des assertions des propriétés.

Cette architecture permet alors la vérification des fonctionnalités souhaitées, ainsi qu'une réutilisation des agents pour la vérification de cet IP lors qu'il sera intégré au Cortex-M0. Il ne reste plus qu'à l'exécuter et analyser ses résultats, ce qui est présenté dans la section 6.3.

6.3 Analyse des résultats

Le logiciel utilisé pour voir les métriques de la vérification est le *Cadence Metrics*. En utilisant ce logiciel, il est possible de regarder des diverses métriques de vérification, comme la couverture de code et la couverture fonctionnelle. Les résultats globaux avec toutes les métriques disponibles de la vérification du filtre sont présentés sur la Figure 15.



Details | Verification Metrics

Metrics | Source | Attributes

Name	Overall Average Grade	Overall Covered
Overall	86.18%	10786 / 11284 (95.59%)
Code	84.89%	9342 / 9832 (95.02%)
Block	96.08%	230 / 242 (95.04%)
Statement	n/a	0 / 0 (n/a)
Expression	55.56%	10 / 18 (55.56%)
Toggle	78.91%	9102 / 9572 (95.09%)
FSM	100%	20 / 20 (100%)
Functional	94.59%	1424 / 1432 (99.44%)
Assertion	89.19%	66 / 74 (89.19%)
CoverGroup	100%	1358 / 1358 (100%)
FaultNode	n/a	0 / 0 (n/a)

FIGURE 15 – Capture d'écran des résultats sur Cadence Metrics

Le test a été réalisé avec 10000 séquences aléatoires et les métriques intéressantes à la vérification UVM sont les fonctionnelles, qui sont représentées sur la Figure 15 par *Assertion* et *CoverGroup*. Le *CoverGroup* indique combien des différentes entrées valides ont été testées lors de la simulation. Dans le test fait, toutes les valeurs d'entrée ont été testées. Cependant, pour tester l'intégration avec le bus AHB, il faut tester aussi des séquences d'entrées. Pour cela, les assertions sont utilisées.

Les assertions sont divisées en deux groupes : couverture et *assert*. Les couvertures sont utilisées pour s'assurer que des séquences ont été couvertes en entrée, lorsque les *asserts* sont utilisés pour vérifier que des différentes propriétés sont garanties pour des différentes entrées. Les résultats individuels sont dans l'Annexe F.

D'après les résultats, on conclut que 94.5% des fonctionnalités (CoverGroups et assertions) ont été couvertes. De toutes les fonctionnalités testées, 100% des assertions de propriété ont été vraies pendant toute la simulation. Il faut alors faire des tests avec une plus grande quantité de séquences afin d'assurer la couverture de toutes les fonctionnalités. A part cela, la plupart des propriétés ont été validées et la phase de vérification a été conclue.

Ayant l'IP de filtrage bien intégré au projet ETNA et déjà validé en utilisant la méthodologie UVM, la prochaine étape serait la mise en œuvre d'un module de captation audio afin d'avoir une démonstration sur la carte FPGA. Cependant, il a été décidé de changer cet objectif et le réorienter vers quelque chose de plus complexe. La démonstration audio a été alors mise de côté et la mise en œuvre d'un accélérateur cryptographique a été commencée. Son implémentation est présentée dans la section suivante.

7 Accélérateur cryptographique

Après une prise en main réussie du projet ETNA et validée par Elsys-Design, les objectifs du stage ont été revus pour intégrer aux tâches de design plus complexes. La mise en œuvre d'un accélérateur cryptographique a été alors choisie en raison de sa complexité et des aspects convenables au sujet du stage — comme l'utilisation de grosses données et la quantité de calculs nécessaires.

La cryptographie fournit des mécanismes puissants pour protéger des données, au prix d'une grande puissance de calcul quand exécutée dans des ordinateurs standards. Le calcul d'une transaction qui requiert une forte cryptographie peut prendre quelques secondes à être finis par un processeur commun, ce qui l'empêche aussi de réaliser d'autres calculs. Il est possible alors de mettre en place un système avec du matériel et une architecture spécialisés dans le but de faire ces calculs dans une vitesse comparable aux solutions présentes dans le marché. Ces systèmes sont appelés accélérateurs matériels.

Le système envisagé lors du stage doit réaliser, de manière efficace, le calcul d'opérations modulaires — tel que l'exponentiation et la multiplication modulaires — afin de mettre en place l'algorithme de cryptographie RSA. Cela dit, l'objectif à moyen terme est le développement de compétences en cryptographie autour du projet ETNA et, à long terme, la création d'un IP commercialisable.

Cependant, il est connu que les performances des implémentations FPGA sont beaucoup plus faibles que celles des implémentations ASIC. C'est une des raisons pour laquelle le FPGA est utilisé plutôt pour le prototypage. C'est aussi pourquoi, même si des bonnes performances sont atteintes, il n'est pas demandé d'avoir des résultats exorbitants, et des temps cibles ne sont pas définis.

Cette section parle de l'implémentation d'un accélérateur de cryptographie RSA, spécialisé dans le calcul des exponentiations modulaire, et est divisée ainsi : la Section 7.1 présente l'algorithme de cryptographie RSA, la Section 7.2 présente les étapes suivies lors de la mise en œuvre du système.

7.1 Chiffrement RSA

La cryptographie est l'étude de techniques mathématiques servant à la sécurité de l'information, tel que la confidentialité, l'authentification et l'intégrité des données. Son utilisation date des milliers d'années, mais c'est la prolifération des ordinateurs et des systèmes de communication qui a permis la croissance de son utilisation et de l'importance que le secteur a subi les dernières décennies. Lors du stage, une attention particulière a été portée sur la confidentialité des données, service utilisé pour garder le contenu de l'information secret à tous sauf ceux autorisés à l'avoir, garantie par son chiffrage.

En 1976, Diffie et Hellmann ont publié un article qui a introduit le concept de chiffrage à clé publique, ce qui a révolutionné l'histoire de la cryptographie. Aussi dans cet article, ils ont fourni une nouvelle méthode d'échange de clés basée sur l'intraitabilité des logarithmes discrets. Cela a généré une grande activité dans la communauté et, deux ans après, en 1978, Rivest, Shamir et Adleman ont découvert le premier schéma pratique de chiffrage et signature à clés publiques. Cet algorithme, aussi connu comme RSA, est basé sur l'intraitabilité de la factorisation de grands nombres premiers et il est, jusqu'au jour présent, l'un des algorithmes les plus utilisés, dû à sa difficulté de cassage [2].

Le RSA est un algorithme de cryptographie asymétrique, ce qui veut dire qu'il y a une distinction entre les clés publiques et privées. Les clés publiques sont partagées entre tous les participants et sont utilisées pour chiffrer un message pour un destinataire particulier ou pour vérifier l'authenticité d'un message d'une personne. Les clés privées, par contre, n'appartiennent qu'à un seul participant et servent pour déchiffrer ou pour signer un message.

Ce qui différencie le RSA des autres systèmes cryptographiques asymétriques est la méthode utilisée pour calculer les clés, de façon à garantir un chiffrement difficile à casser. Le calcul consiste à utiliser des grands nombres premiers pour trouver les valeurs qui composent les clés : le modulus (N), l'exposant de la clé publique (e) et l'exposant de la clé privée (d). Les valeurs doivent être telles que l'Equation 2 soit vérifiée.

$$m = (m^e)^d \mod N = (m^d)^e \mod N \tag{2}$$

Le chiffrement RSA exécute alors des exponentiations modulaires afin de calculer les messages chiffrés à l'aide de l'exposant e tout en garantissant une façon de recalculer les messages en clair à l'aide de l'exposant d (cette fois-ci appliquée au message chiffré). On définit m comme étant le message en clair et c le message chiffré, tels que l'Equation 3 et l'Equation 4 correspondent, respectivement, aux fonctions de chiffrage et déchiffrage.

$$c \equiv m^e \mod N \tag{3}$$

$$m \equiv c^d \mod N \tag{4}$$

La sécurité intrinsèque au RSA vient de l'utilisation de très grands nombres premiers, de sorte que la factorisation des nombres produits par eux soit impraticable. Il est recommandé alors d'utiliser des nombres de l'ordre de 2048 ou 4096 bits.

Dû à la grande taille des données utilisées, le calcul de l'exponentiation modulaire ne peut pas se faire de façon naïve – c'est-à-dire d'abord calculer m^e (ou c^d) et ensuite le reste modulo N –, parce que cela demanderait de manipuler des nombres trop grands pour être stockés dans un ordinateur, avec une utilisation exorbitante des ressources (temps et mémoire). Il existe, pourtant, des méthodes efficaces pour le calcul de l'exponentiation modulaire, qui seront explorées dans la sous-section 7.2.

7.2 Développement du module RSA

Les calculs de l'algorithme de chiffrage RSA ont été définis dans la section 7.1. Il s'agit des exponentiations modulaires avec des valeurs pré-sélectionnées, qui peuvent être décomposées en plusieurs multiplications modulaires. Cependant, ces opérations peuvent être assez coûteuses – soit en logiciel, soit en matériel – si leur implémentation n'est pas bien réfléchie.

En raison de l'utilisation de grands nombres, l'optimisation des calculs des opérations modulaires est une partie importante de la mise en œuvre de cette implémentation. Cela rend aussi important le choix de l'architecture à mettre en place, en considérant qu'elle a un grand impacte sur la quantité de ressources – registres, additionneurs, etc – utilisés. A l'effet de comparaison des résultats avec des produits commerciaux, il sera considéré une implémentation à 2048 bits.

Cette section présente les différentes étapes de conception et développement de l'IP . Elle se divise ainsi : l'opération d'exponentiation modulaire, ainsi que son architecture matériel, est présenté dans la sous-section 7.2.1 ; l'opération de multiplication modulaire et son architecture dans 7.2.2.

7.2.1 Opération d'exponentiation modulaire

Le modulo est l'opération mathématique qui calcule le reste de la division entre deux entiers. L'arithmétique modulaire fournit des propriétés intéressantes, puisqu'il est possible de travailler sur les restes plutôt que sur les nombres, ce qui permet les calculs d'être faits de façon plus efficace. Pour cela, les opérations modulaires sont beaucoup utilisées par des algorithmes de cryptographie, comme le RSA.

L'opération d'exponentiation modulaire est simplement une exponentiation où toutes les opérations de multiplication sont modulaires. Les heuristiques d'exponentiation utilisées pour calculer m^e peuvent être aussi appliquées pour calculer $m^e \bmod N$. Des techniques de mise en œuvre de cette opération seront vues dans cette section, avec quelques détails de leurs mise en place en matériel et vulnérabilité aux attaques.

Soient m, e et N des entiers, le calcul binaire de l'exponentiation $m^e \bmod N$ peut être fait en parcourant les bits de l'exposant de deux façons, selon la direction avec laquelle les bits sont parcourus : de droite à gauche ou de gauche à droite. L'opération est définie telle que

$$m^e \bmod N = ((m^{2^0 e_0} \times m^{2^1 e_1}) \times ...) \times m^{2^{n-1} e_{n-1}} \bmod N,$$

où toute multiplication est aussi modulaire. De cette manière, la quantité de multiplications nécessaires est de l'ordre logarithmique.

Une méthode pour calculer l'exponentiation modulaire est indiquée sur l'Algorithme 1. A chaque itération, la variable c est mise à jour en fonction des bits de e lues jusqu'à présent. Pour cela, cette méthode porte le nom d'exponentiation binaire. Dans cet algorithme, l'exposant est parcouru bit-à-bit et le carré de la dernière valeur de c est calculé. Une multiplication modulaire est faite sur la variable c si le bit est 1. Si le bit est à 0, le calcul est fait sur un registre « faux » c'. Ne pas ajouter ce registre crée une vulnérabilité qui peut être exploré par des attaques par canaux auxiliaires (temporaux et de consommation) pour découvrir certaines caractéristiques du système — la clé privée, dans ce cas.

Algorithme 1 Exponentiation binaire avec un registre faux

Entrées: m, e, N, n
Sorties: $c = m^e \bmod N$

1: **if** $e_{n-1} = 1$ **then**
2: $c \leftarrow m$
3: **else**
4: $c \leftarrow 1$
5: **end if**
6: $c' \leftarrow c$
7: **for** $i = n - 2$ **downto** 0 **do**
8: $c \leftarrow c \times c \mod N$
9: **if** $e_i = 1$ **then**
10: $c \leftarrow c \times m \mod N$
11: **else**
12: $c' \leftarrow c \times m \mod N$
13: **end if**
14: **end for**
15: **return** c

Pourtant, cette implémentation a aussi une vulnérabilité : la création d'une valeur non utilisée rend possible les attaque par fautes. C'est-à-dire que, si une faute générée au moment de la multiplication par M (lignes 10 et 12) n'est pas propagée, le bit lu est 0 et, au cas échéant, le bit est égal à 1. Ces attaques sont plus difficiles à réaliser que celles citées antérieurement, mais elles sont quand même possibles. Pour les éviter, il est possible d'appliquer un algorithme appelé *Montgomery Ladder*, indiqué sur l'Algorithme 2. Cet algorithme peut être fait en parcourant l'exposant dans les deux directions.

Ainsi que le dernier algorithme, le *Montgomery Ladder* parcourt les bits de e de gauche à droite. La différence est que, indépendamment de la valeur du bit, deux multiplications modulaires différentes sont faites en parallèle, l'une sur $R0$ et l'autre sur $R1$, de telle façon que le résultat de l'exponentiation soit dans le registre $R0$ lors de la dernière itération.

Algorithme 2 Exponentiation binaire avec *Montgomery Ladder*

Entrées: m, e, N, n
Sorties: $R0 = m^e \bmod N$

1: $R0 \leftarrow 1$
2: $R1 \leftarrow m$
3: **for** $i = n-1$ **downto** 0 **do**
4:     **if** $e_i = 1$ **then**
5:         $R0 \leftarrow R0 \times R1 \mod N$
6:         $R1 \leftarrow R1 \times R1 \mod N$
7:     **else**
8:         $R1 \leftarrow R0 \times R1 \mod N$
9:         $R0 \leftarrow R0 \times R0 \mod N$
10:     **end if**
11: **end for**
12: **return** $R0$

Ce qui fait cette implémentation résistante aux attaque par fautes, c'est qu'il y a toujours une des multiplications qui est dépendante de la valeur actuelle de l'autre registre. En autres mots, cet algorithme n'insère pas des valeurs fausses et, au cas où une faute est créée dans le système, elle sera toujours propagée, sans révéler la valeur de la clé. Celui-là est donc l'algorithme choisi pour l'implémentation du RSA.

Une autre propriété intéressante de cette méthode est le fait qu'il est toujours possible de vérifier la propriété

$$R1 = R0 \times m,$$

malgré que $R1$ n'est jamais calculé de la sorte. Cela peut être utilisé comme une façon d'assurer la sûreté et la sécurité de l'IP.

La mise en œuvre matériel de l'algorithme d'exponentiation est faite alors de la façon suivante. D'abord, les signaux d'entrée sont presque les mêmes de l'algorithme précèdent : `m`, le message en clair ; `e`, l'exposant, qui peut être la clé privée ou la clé publique ; `N`, le modulus choisi ; `n` la quantité de bits nécessaire à représenter le modulus ; et `start` pour initialiser le calcul. Il y a deux signaux de sorties : `res`, qui prend la valeur du registre `R0` ; et `done` pour indiquer que le calcul est fini et la valeur sur la sortie `res` est le résultat.

Dans l'exponentiation, en plus des registres `R0` et `R1`, un registre à décalage `key_shift` est créé pour parcourir les exposants bit à bit. Le module est composé de deux sous-modules pour faire les deux multiplications en parallèle. Il est aussi composé d'une unité de contrôle, qui gère les signaux internes — comme les entrées des sous-modules, les compteurs et les multiplexeurs — et le signal de sortie `done`.

Les signaux d'entrée `n` et `start` ne sont pas utilisés directement dans le calcul, mais dans l'unité de contrôle. Lors que le signal `start` est mis à 1, la configuration de l'opération est faite : les registres de `R0` et `R1` sont initialisés à 1 et à `m`, respectivement. Ensuite, les multiplications sont initialisées et tous les registres sont mis à jour avant d'initialiser les multiplications suivantes. Pour le RSA à 2048 bits, 2048 multiplications modulaires parallèles sont faites jusqu'à avoir le résultat de l'exponentiation. L'architecture résultante est illustré dans l'Annexe G.1.

L'entrée `one` sera expliquée ultérieurement, ainsi que l'implémentation des sous-modules de multiplication modulaire, qui sera expliqué dans la sous-section suivante.

7.2.2 Opération de multiplication modulaire

La multiplication modulaire est définie comme le calcul de c tel que

$$c = a \times b \bmod N,$$

soient a, b et N des entiers tels que $0 \leq a, b < N$ et le résultat c aussi un nombre entier avec $0 \leq c < N$. C'est une opération assez importante pour l'algorithme RSA, étant donné qu'elle sera répétée 2048 fois afin de calculer l'exponentiation modulaire. Il faut, donc, qu'elle soit efficace en temps et en ressources utilisés. Trois méthodes pour calculer le produit c seront discutées : multiplication et division, multiplications et divisions entrelacées et multiplication de Montgomery.

La première méthode, multiplication et division, est la plus simple d'entre elles. Il s'agit de multiplier a et b afin d'obtenir une valeur intermédiaire

$$c' = a \times b$$

telle que $0 \leq c' < N^2$. Ensuite, cette valeur est divisée par le modulus N et le reste de cette division est le résultat attendu de la multiplication modulaire.

C'est une méthode facile à implémenter, mais avec des inconvénients qui rendent son utilisation impraticable avec l'algorithme RSA. Un exemple est la quantité de ressources utilisés : soit N une valeur représentable sur n bits, il est nécessaire $2n$ bits pour représenter la valeur intermédiaire c'. C'est-à-dire que, pour le RSA à 2048 bits, il sera nécessaire l'utilisation des registres à 4096 bits, ce qui représente une utilisation importante des ressources disponibles. Cette implémentation est intéressante aux cas où la valeur c' est nécessaire, ce qui n'est pas le cas du RSA.

La méthode des multiplications et réductions entrelacées fournit une manière de faire la multiplication modulaire en utilisant moins de ressources que la solution précédente. Il s'agit de l'exploitation de la représentation binaire des nombres et des propriétés de l'arithmétique modulaire, qui disent que

$$a + b \bmod N = a \bmod N + b \bmod N.$$

La multiplication peut être alors écrite comme

$$\begin{aligned} c' = a \times b = a \times \sum_{i=0}^{n-1}(b_i 2^i) = \sum_{i=0}^{n-1}(a \times b_i)2^i \\ = 2(\cdots 2(2(0 + a \times b_{k-1}) + a \times b_{k-2}) + \cdots) + a \times b_0, \end{aligned}$$

où chaque multiplication partielle doit être divisée par le modulus afin de garder les valeurs dans un intervalle approprié.

Ce format crée un algorithme d'additions et multiplications par 2 présenté sur l'Algorithme 3. Sachant que $0 \leq a, b, c < N$, alors c à la fin de chaque itération sera dans l'intervalle $0 \leq c \leq 3N - 3$. Il faut alors au maximum 2 soustractions par N afin de calculer le modulo et réduire le résultat à $0 \leq c < N$.

Algorithme 3 Multiplication binaire avec multiplications et divisions entrelacées

Entrées: a, b, N, n
Sorties: $c = a \times b \bmod N$

```
1: c ← 0
2: for i = n − 1 downto 0 do
3:     c ← 2c + a × b_i
4:     c ← c mod N
5: end for
6: return c
```

L'implémentation matérielle de cet algorithme requiert un décalage, un produit partiel et une addition (ligne 3), et 2 soustractions avec test du signe pour effectuer le modulo. Le décalage et les produits partiels consomment très peu de ressources matériels. Les additions et les divisions par N, en revanche, sont des opérations très coûteuses. Les additions sont cruciales pour l'opération de multiplication et seront traitées dans une autre section. Les divisions, par contre, peuvent être évitées en utilisant la multiplication de Montgomery.

Montgomery définit le N-résidu d'un entier a relativement à R, soit $R > N$, comme

$$\bar{a} = a \times R \bmod N,$$

opération qui sera appelée transformation de Montgomery. L'algorithme de Montgomery offre alors une méthode efficiente de calculer

$$(\bar{a} \times \bar{b})R^{-1} \bmod N,$$

où R, N sont des entiers premiers entre eux tels que $R > N$, et $\bar{a}$ et $\bar{b}$ sont les N-résidus des entiers a et b en relation à R. N étant produit de grands nombres premiers — dans le cas du RSA —, il n'est pas produit de 2 et donc toute puissance de 2 est première avec N. L'astuce de cet algorithme alors est de remplacer les divisions modulo N par des divisions par une puissance de 2 — faites très efficacement au moyen de décalages à droite —, en choisissant $R = 2^n$ tel que $2^{n-1} \leq N < 2^n$ [7].

L'algorithme de Montgomery exploite les propriétés des résidus pour calculer rapidement le N-résidu de la multiplication de deux entiers dont les N-résidus sont connus. De là, la multiplication de Montgomery est définie comme le N-résidu

$$\begin{aligned}\bar{c} &= (a \times b)R \bmod N \\ &= (aR \times bR)R^{-1} \bmod N \\ &= (\bar{a} \times \bar{b})R^{-1} \bmod N.\end{aligned}$$

L'inverse de la transformation de Montgomery, c'est-à-dire le calcul d'un nombre entier étant donné son N-résidu par rapport à R, s'appelle réduction de Montgomery et peut être fait aussi efficacement par une division par R. Donc, le résultat de la multiplication modulaire peut être calculé à partir de son N-résidu en calculant son inverse

$$\begin{aligned}c &= \bar{c} \times R^{-1} \bmod N \\ &= ((a \times b)R)R^{-1} \bmod N \\ &= a \times b \bmod N.\end{aligned}$$

Le plus important de cette méthode, c'est que la multiplication ne se fait qu'avec des multiplications et divisions par R, qui peuvent être calculées très rapidement puisque R est une puissance de 2. Il s'agit alors des décalages à gauche — pour les multiplications par 2 — et à droite — pour les divisions par R.

L'architecture matérielle mise en place pour cette méthode est illustrée sur l'Annexe G.2. D'après cette architecture, il est nécessaire 2 additionneurs à 2048 bits, ainsi qu'un comparateur, afin d'exécuter la multiplication modulaire. Ce sont des opérations très coûteuses en taille et en délai de propagation, ce qui rend nécessaire leur optimisation afin de réussir la performance nécessaire. Dans la sous-section 7.3, les méthodes utilisées pour les optimiser — et même pour faire disparaître la comparaison — sont discutées.

7.3 Optimisations

L'algorithme RSA est souvent utilisé avec des nombres géants afin d'assurer la sécurité. Par conséquent, tous les calculs faits pendant l'exécution se font sur ces grands nombres, ce qui le rend assez lent et, en implémentation matérielle, assez grand en surface et ressources. Dans le but de le rendre plus performant, deux optimisations sont faites : l'une sur toutes les additions de l'algorithme, présentée dans la Section 7.3.1, et l'autre sur les comparaisons, dans 7.3.2.

7.3.1 Additionneurs

Dans l'étape de synthèse logique, les logiciels font de leur mieux afin de traduire le code HDL sous forme de porte logique. Une partie importante de cette étape est la synthèse des opérations d'addition, qui est faite différemment selon le logiciel utilisé et l'application faite. De plus, dans le cas du FPGA, l'architecture elle-même est optimisée pour faire des additions du type RCA (Ripple Carry Adder). Dans le cadre du stage, le logiciel utilisé a été *Intel Quartus Prime*.

Les additions sont normalement traduites sous forme des additionneurs parallèles à propagation de retenue — ou *Ripple Carry Adder* (RCA), en anglais. L'avantage de cet additionneur est sa simplicité d'implémentation, il s'agit des additionneurs complets un bit enchaînés de telle façon que chaque additionneur attend la retenue propagée par la dernière étape afin de calculer ses sorties. La Figure 16 montre l'implémentation d'un RCA à 4 bits. Les bits a_0 et b_0 représentent les bits de poids faible des nombres à sommer et le résultat de l'addition est représenté par les bits $s_0 - s_3$.

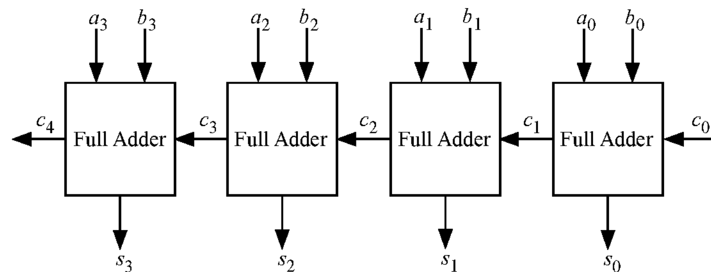


FIGURE 16 – Additionneur parallèle à propagation de retenue (RCA) à 4 bits

Son plus grand inconvénient vient aussi de la façon avec laquelle il est implémenté, car il dépend de la propagation de la retenue de module en module. C'est-à-dire que le bit de poids fort du résultat n'est connu avec certitude qu'après la retenue est passée par tous les additionneurs complets. Par conséquent, le résultat final n'est disponible, au pire, qu'après un délai de $2(n-1)+3$ portes logiques, où n est le nombre de bits de l'additionneur.

Pour un additionneur à 4 bits, par exemple, le délai nécessaire à calculer tous les bits du résultat est égal à 9 portes logiques. Le circuit de l'additionneur complet est présenté sur la Figure 17, où il est possible de voir le délai de 3 portes pour le calcul de la retenue, et de 2 portes pour la somme. Le chemin critique de ce circuit passe alors par les portes de calcul des retenues. Alors, cette méthode n'est pas conseillée pour des circuits dépassant quelques bits avec des contraintes de temps, ce qui est le cas du RSA.

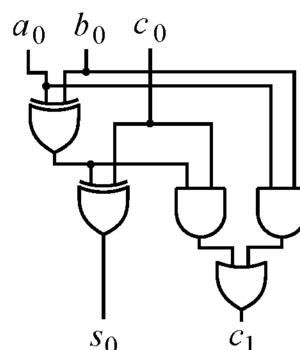


FIGURE 17 – Additionneur complet utilisé dans le RCA

Afin de réduire le délai de propagation et augmenter la fréquence d'exécution, il est possible d'utiliser d'autres circuits additionneurs. Dans le cadre du stage, l'additionneur parallèle à retenue anticipée — ou *Carry Lookahead Adder* (CLA) — a été étudié et mis en place. Il s'agit d'un circuit qui fait le calcul des retenues en avance, dans le but de ne pas les attendre passer dans tous les modules. Il est possible de le faire en se basant sur le fait qu'une retenue n'est générée qu'en deux cas : quand les bits a_i et b_i sont égaux à 1 ou quand l'un des deux bits est égal à 1 et la retenue d'entrée est aussi à 1. Son circuit est illustré sur la Figure 18.

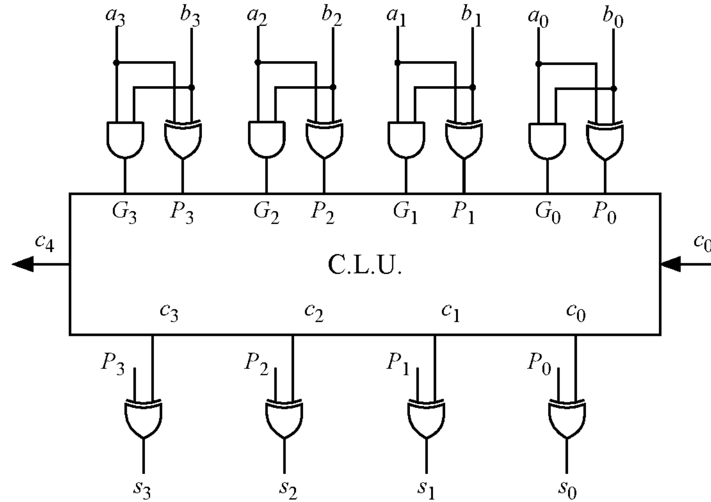


FIGURE 18 – Additionneur parallèle à retenue anticipée (CLA) à 4 bits

Deux nouvelles valeurs intermédiaires sont nécessaires afin de faire l'addition. La retenue générée (G_i) indique si la somme génère une retenue, indépendamment de la retenue d'entrée, lorsque la retenue propagée (P_i) indique si la somme propagera une retenue en entrée. Les deux valeurs sont calculées telles que

$$G_i = a_i \cdot b_i$$
$$P_i = a_i \oplus b_i.$$

Les retenues et les bits qui composent le résultat sont calculés tels que

$$c_{i+1} = G_i + P_i \cdot c_i \tag{5}$$
$$s_i = P_i \oplus c_i. \tag{6}$$

Il est possible de dérouler l'Equation 5 afin de paralléliser tous les calculs. De cette façon, on peut calculer tous les retenues sur un CLA à 4 bits telles que

$$c_1 = G_0 + P_0 \cdot c_i \tag{7}$$
$$c_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot c_0 \tag{8}$$
$$c_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot c_0 \tag{9}$$
$$c_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0. \tag{10}$$

Ces calculs sont faits dans le *Carry Lookahead Unit* (CLU) et leur circuit est représente sur la Figure 19. Ainsi, le délai nécessaire au calcul de c_i est de 3 portes logiques : 1 porte pour faire le calcul des G_i et 2 pour faire le calcul de l'Équation 10. Finalement, le délai pour le calcul du résultat de la somme (Equation 6), est constant et égal à 6 portes logiques, ce qui ne semble pas très avantageux par rapport aux 9 portes logiques du RCA mais, quand on l'applique à un additionneur à 2048 bits, la différence est beaucoup plus importante.

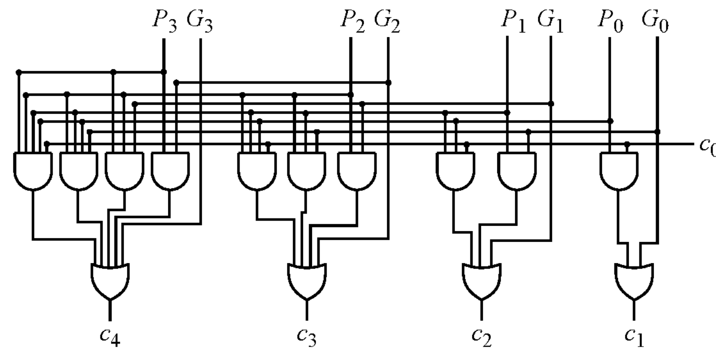


FIGURE 19 – Circuit décrivant le calcul des retenues dans le CLU

De plus, l'additionneur CLA peut être cascadé afin d'avoir des meilleures performances. Il s'agit de créer plusieurs niveaux de CLU à 4 bits de façon à avoir encore plus de calculs en parallèle. Sur la Figure 20, un exemple de CLA à 16 bits cascadé avec deux niveaux est présenté, où `CLU1` est le premier niveau et `CLU2` le deuxième. En enchaînant quatre `CLU2`, il est possible d'avoir un additionneur à 64 bits, avec un délai de 14 portes logiques, lors que le RCA à 64 bits présente un délai de 129 portes. Avec un troisième niveau, ce délai peut être réduit à 10 portes.

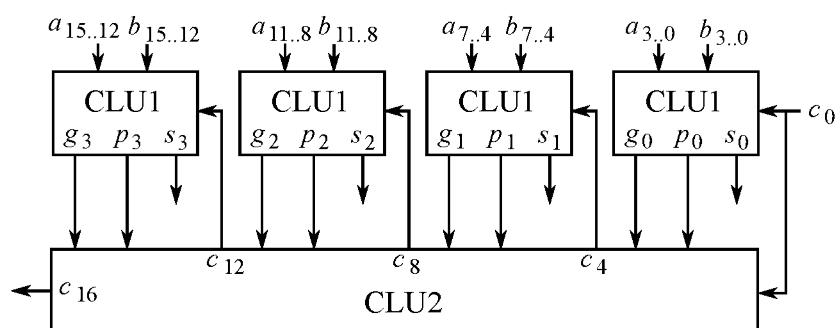


FIGURE 20 – CLA à 16 bits cascadé avec deux niveaux de CLU

L'inconvénient de cette méthode est la quantité de portes logiques utilisées, ainsi que sa complexité, quand plusieurs niveaux de CLU sont nécessaires. L'impact de cette méthode sur le RSA — en délai de propagation et en surface — sera présenté ultérieurement dans la Section 7.4.

7.3.2 Comparateurs

Ainsi que pour les additionneurs, les comparaisons sont synthétisées différemment selon le logiciel et l'application. Pour le RSA, il est souvent fait des opérations qui comparent si les résultats intermédiaires sont plus grands ou égal à N afin de faire des calculs modulo N. Cela est synthétisé par le logiciel comme une soustraction $x - N$, où le signal du résultat indique si x est plus grand ou plus petit que N. Ces soustractions sont implémentées sous la forme des additions et, par conséquent, des additionneurs RCA.

L'utilisation des additionneurs RCA et leur remplacement par des additionneurs CLA (en raison de leurs grands délais de propagation) ont été déjà vus dans la Section 7.3.1. Une façon d'optimiser les comparaisons est leur remplacement par des soustractions plus efficientes.

En revanche, il est possible de l'optimiser davantage. Il peut être prouvé que le résultat des multiplications faites, sans les comparaisons, sont toujours dans un intervalle limité — entre 0 et $2R$. Cela veut dire qu'il est possible d'enlever les comparaisons du calcul, au coût d'un bit de plus et de, au plus, 2 soustractions par N à la fin de la multiplication modulaire afin d'avoir le résultat modulo N.

Ce changement permet une augmentation importante de la fréquence maximale de l'IP puisque, avec la comparaison, une addition et une soustraction étaient faites successivement, ce qui avait un délai de propagation assez grand. En l'enlevant, il a été possible de réduire le temps des chemins critiques à la moitie et, par conséquent, doubler les fréquences d'utilisation.

7.4 Résultats

La première implémentation du module RSA a été assez simple. Il n'y avait aucune optimisation, c'est-à-dire que les additionneurs RCA et les comparateurs étaient toujours utilisés. L'objectif était d'exécuter l'algorithme RSA à 2048 bits dans le meilleur temps possible, sans contrainte de temps. Pourtant, on a quand même défini une contrainte de temps car les outils essaient à tout prix de l'atteindre, ce qui nous a rendu possible de voir tout de suite ce qui devait être améliore. Il a été choisi alors la fréquence native du cristal FPGA, 50 MHz.

En changeant les opérations d'addition RCA pour des additions CLA, il a été possible d'augmenter de façon importante la performance. Pour une implémentation à 256 bits, l'IP était déjà 62% plus performant qu'avant, ce qui est présenté sur la Table 3.

Implémentation	Fréquence maximale	Temps de synthèse	Quantité d'ALUTs
RSA RCA 128	42.79 MHz	00 : 01 : 28	4 486 / 114 480
RSA CLA 128	42.98 MHz	00 : 02 : 10	6 042 / 114 480
RSA RCA 256	24.15 MHz	00 : 06 : 35	8 683 / 114 480
RSA CLA 256	39.26 MHz	00 : 05 : 22	12 112 / 114 480
RSA RCA 512	-	01 : 19 : 54	150 491 / 114 480
RSA CLA 1024	27.3 MHz	01 : 25 : 30	51 336 / 114 480

Table 3 – Résultats de la synthèse avec des différents additionneurs (avec des comparateurs)

Il est notable que l'implémentation avec des additionneurs CLA permet l'utilisation des fréquences beaucoup plus hautes, vu que l'utilisation de l'algorithme RSA à plus de 512 bits sont déjà trop grandes pour la carte FPGA — même si le CLA utilise, normalement, plus de ressources. Cela est dû au fait que, on mettant une contrainte temporelle, les outils essaient à tout prix de l'atteindre, même si cela résulte en une augmentation de la quantité de ressources utilisés.

Toutefois, il n'est pas possible de réaliser des calculs à la fréquence native du FPGA, vu que la fréquence maximale atteinte à 1024 bits utilisant le CLA est de 27.3 MHz. En revanche, cette fréquence-là est plus que suffisante pour les *smartcards*, qui fonctionnent souvent entre 1 et 5 MHz, vu que la basse consommation énergétique est très envisagée. De plus, il est à noter que les FPGA sont utilisés pour faire le prototypage, et non les produits finaux. C'est-à-dire que, quand implémenté en ASIC, l'implémentation sera beaucoup plus performante.

Cependant, en essayant toujours d'améliorer la performance, une autre optimisation a été faite, par rapport aux comparateurs. Dans les multiplications modulaires ils ont été enlevés lorsque, dans les opérations de transformation de Montgomery, ils ont été remplacés par des additionneurs CLA. Cela a permis une duplication des fréquences maximales. Les nouvelles valeurs sont présentées sur la Table 4.

Taille	Fréquence maximale	Quantité d'ALUTs	Temps de calcul (50 MHz)
RSA 1024	55.27 MHz	52 361 / 114 480	21.12 ms
RSA 2048	48.88 MHz	101 766 / 114 480	$\sim$ 84 ms

Table 4 – Résultats finaux de la synthèse

Il n'est toujours pas possible de réaliser les calculs RSA avec la fréquence du cristal FPGA, mais la performance est assez proche. Pourtant, le module développé fonctionne avec des fréquences beaucoup plus élevées que celles utilisés dans le marché, ce qui permet la réduction de la fréquence. Le module développé est quand même beaucoup plus performant qu'une bonne partie des produits existantes dans le marché, réalisant le calcul en un cinquième du temps du produit le plus rapide à 50 MHz, et en 2 fois le temps à 5 MHz. La Table 5 présente une comparaison du temps de calcul de l'IP RSA développé pour le projet ETNA avec les produits commercialisés.[2]

Produit	Temps de calcul
RSA ETNA (50 MHz)	$\sim$ 84 ms
Softlock SLCOS InfineonSLE78	426.26 ms
JavaCOS A40	573.78 ms
RSA ETNA (5 MHz)	$\sim$ 840 ms
NXP J2D081 80K	2002.7 ms
Oberthur ID-ONE Cosmo 64 RSA v5.4	2927.22 ms

Table 5 – Temps de calcul RSA à 2048 bits des différents produits commercialisés

Une dernière optimisation était prévue, en utilisant le théorème des restes chinois afin d'augmenter les performances du module, mais le temps de stage n'a pas été suffisant. Il reste, alors, comme une suggestion de continuation de ce projet.

[2] https://www.fi.muni.cz/~xsvenda/jcalgtest/comparative-table.html

8 Avancement et état du projet

Les objectifs du stage étaient de faire du design FPGA orienté à une plate-forme et de la vérification UVM. Il était souhaité d'avoir une moitié du stage dédié à chacun d'entre eux. Cependant, l'étape de vérification a pu être faite en beaucoup moins de temps que prévu — environ 1 mois — et les objectifs ont été revus.

D'abord, le stage suivait la méthodologie Agile, divisé par des sprints de deux semaines et géré avec l'outil *Redmine*. Au début de chaque sprint, tous les deux semaines, des tâches étaient définies en considérant ce qui a été fait lors du dernier sprint. Cela permettait une bonne adaptation des semaines de travail selon le déroulement effectif du projet. Au fil du temps, la rigueur de la méthodologie a été relaxée car les objectifs initiaux avaient été remplis, et de nombreuses directions étaient possibles.

Les objectifs initiaux étaient d'avoir une démonstration sur la carte FPGA, capable de capturer et filtrer des signaux audio. Au cours du stage, j'ai demandé à revoir les objectifs du stage, puisqu'on est passé d'une phase où on souhait que certaines choses soient faites et ajoutées à l'existant à une phase où je devenais, en quelque sorte, propriétaire du projet. Je voulais, alors, un sujet plus complexe et convenable aux clients de l'entreprise, tels que NXP et Schneider Electric.

Les tâches ont été changés alors au fur et a mesure afin de suivre les objectifs définis. Le premier mois de stage a été utilisé pour la prise en main de l'environnement, ainsi que l'étude de l'environnement ETNA, le deuxième mois pour l'implémentation et intégration du filtre et le troisième pour l'étude et l'application de la vérification UVM. A la suite du changement d'objectif, le quatrième et cinquième mois ont été dédiés à l'étude, développement et optimisation de l'accélérateur RSA. Les dernières semaines ont été dédiées à ce rapport et à la soutenance.

Finalement, même avec de gros changements du planning, le projet a été bien développé. L'environnement a pu être enrichi avec deux nouveaux périphériques : un filtre FIR, son interface avec le bus AHB et son driver ; et un accélérateur de calculs RSA, capable de faire des calculs sur 2048 bits très efficacement. De plus, un premier environnement UVM a été mis en place pour la plateforme, servant d'exemple et de base pour les prochains utilisateurs, ainsi que les modules développés pourront servir comme point de départ pour des projets futurs.

9 Prise en compte de l'impact environnemental et sociétal

Mon travail chez Elsys est fait à la fois sur l'ordinateur, où je n'ai qu'un écran, et sur la carte FPGA. Ce sont, alors, les seuls consommateurs directs d'énergie pendant toute la journée. Encore, la carte FPGA n'est allumée que quand je l'utilise, ce qui n'est pas souvent. De plus, l'environnement est assez illuminé par le soleil, ce qui rend parfois inutile l'utilisation des ampoules.

À propos des déplacements faits pour le stage, je n'utilise que les transports en commun. Le stage se passe à Vallauris, et actuellement j'habite à Nice. Je prend, donc, un car qui fait le trajet de Nice à Sophia-Antipolis, puis un bus qui fait le trajet entre l'arrêt du car et les locaux de l'entreprise. Le trajet est assez long à cause des bouchons, et cela me prend environ 80 minutes, deux fois tous les jours. Cependant, la plupart des voitures ne contiennent qu'une ou deux personnes, alors le covoiturage serait une bonne solution pour réduire le trafic et, aussi, la pollution. Le transport en commun est, donc, un choix intéressant et moins agressif.

Les politiques de l'entreprise à propos du développement durable ne sont pas assez claires. Cependant, il y a une forte consommation de café en capsules Nespresso, dont l'enveloppe est fait en aluminium et causent, alors, une pollution importante et de la déforestation si les déchets ne sont pas recyclés. Heureusement, l'entreprise met à disposition des employés des poubelles spécifiques aux capsules pour qu'elles puissent être recyclées.

Finalement, comme mon stage n'a pas un objectif commercial, et ne sert qu'en faveur de l'entreprise comme une vitrine technologique, il aura un impact très indirect sur la société.

10 Conclusion

La plateforme ETNA a été enrichie avec deux nouveaux périphériques, conforme aux objectifs initiaux du stage, développés en deux parties : l'une pour la mise en œuvre d'un filtre FIR et l'autre pour un module RSA. La première partie, le développement et vérification du filtre, a été intéressante pour la prise en main de l'environnement, ainsi que pour apprendre la méthodologie de vérification UVM. La deuxième et dernière partie, la mise en place de l'accélérateur cryptographique, a été assez intéressante à cause de sa complexité, ce qui m'a fait un grand défis et m'a permis d'améliorer mes compétences.

Il a été le premier stage que j'ai fait dans le domaine de la microélectronique, ce qui m'a permis une grande croissance professionnelle et, surtout, technique. J'ai été en contact constant avec des professionnels déjà expérimentés dans le sujet, ainsi qu'avec d'autres stagiaires et des nouveaux employés. Cela m'a rendu possible d'échanger et de comprendre la vision des différents types de professionnels.

Avec un projet complexe, j'ai pu développer de façon importante mes compétences en développement matériel ayant comme cible les cartes FPGA. Même si j'avais déjà quelques expériences académiques, elles n'étaient pas suffisantes pour m'insérer dans le marché. Ce stage m'a permis, alors, de combler cette lacune technique et de me préparer pour être un professionnel du domaine.

J'ai aussi appris plus de l'étape de vérification du flow de conception matériel. Auparavant, je n'avais jamais fait de la vérification, mais j'étais conscient de son importance. Ce stage m'a permis d'ajouter la méthodologie UVM dans mes compétences techniques, ce qui m'est très intéressant, vu que c'est l'étape la plus longue de la conception.

De plus, la cryptographie est un sujet qui m'intéressait beaucoup, et mettre en place un accélérateur RSA m'a fait approfondir mes connaissances en sécurité numérique et matérielle. J'ai aussi pu comprendre quelques concepts qui ne m'étaient pas très clairs précédemment, comme la cryptographie à clés publiques et la signature RSA. Finalement, l'environnement ETNA et la méthodologie de travail m'ont permis d'améliorer mes compétences en logiciel, en développant des scripts afin d'automatiser les tests.

Sur le plan personnel, le plus grand apport de ces 6 mois de stage a été la relation avec les gens au bureau, malgré les différences culturelles et linguistiques. L'ambiance était très conviviale et je pouvais parler français toute la journée, ce qui m'a fait améliorer de manière significative mes compétences en français.

Références

[1] Where Passion Leads to Excellence | ADVANS GROUP. `http://www.advans-group.com/`. Accédé : 01/04/2019.

[2] Paul C. van Oorschot Alfred J. Menezes and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[3] Développement Logiciel | AViSTO. `https://www.avisto.com/fr`. Accédé : 02/04/2019.

[4] Systèmes Electroniques et logiciels | ELSYS Design. `https://www.elsys-design.com/fr`. Accédé : 02/04/2019.

[5] Patrice et Raimbault Frédéric Lavenir, Dominique et Quinton. Architectures systoliques et parallelisme de données. *IRISA*, pages 1–17.

[6] L'imagination au service de la mécanique | MECAGINE. `https://www.mecagine.com/fr`. Accédé : 02/04/2019.

[7] Peter L. Montgomery. Modular multiplicatin without trial division. *Mathematics of Computation*, pages 519–521, 1985.

Appendices

A Altera DE2-115

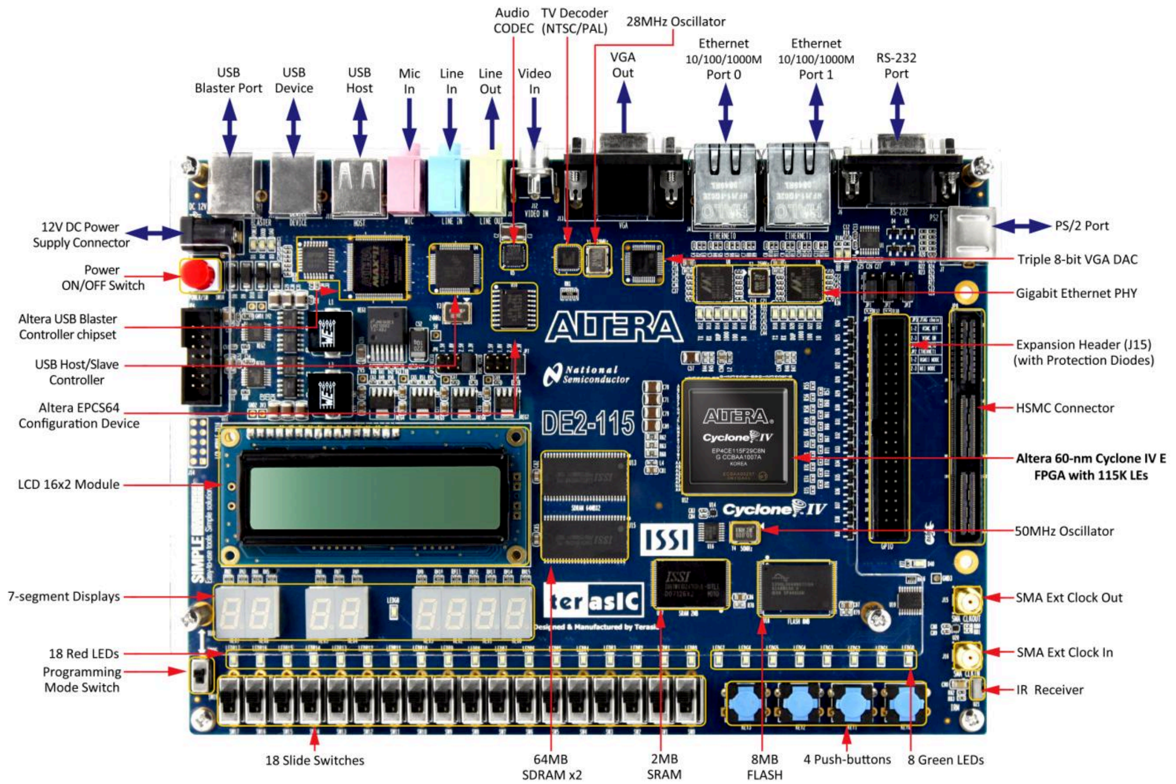


FIGURE 1 – Plate-forme Altera DE2-115 (Source : documentation technique Altera DE2-115)

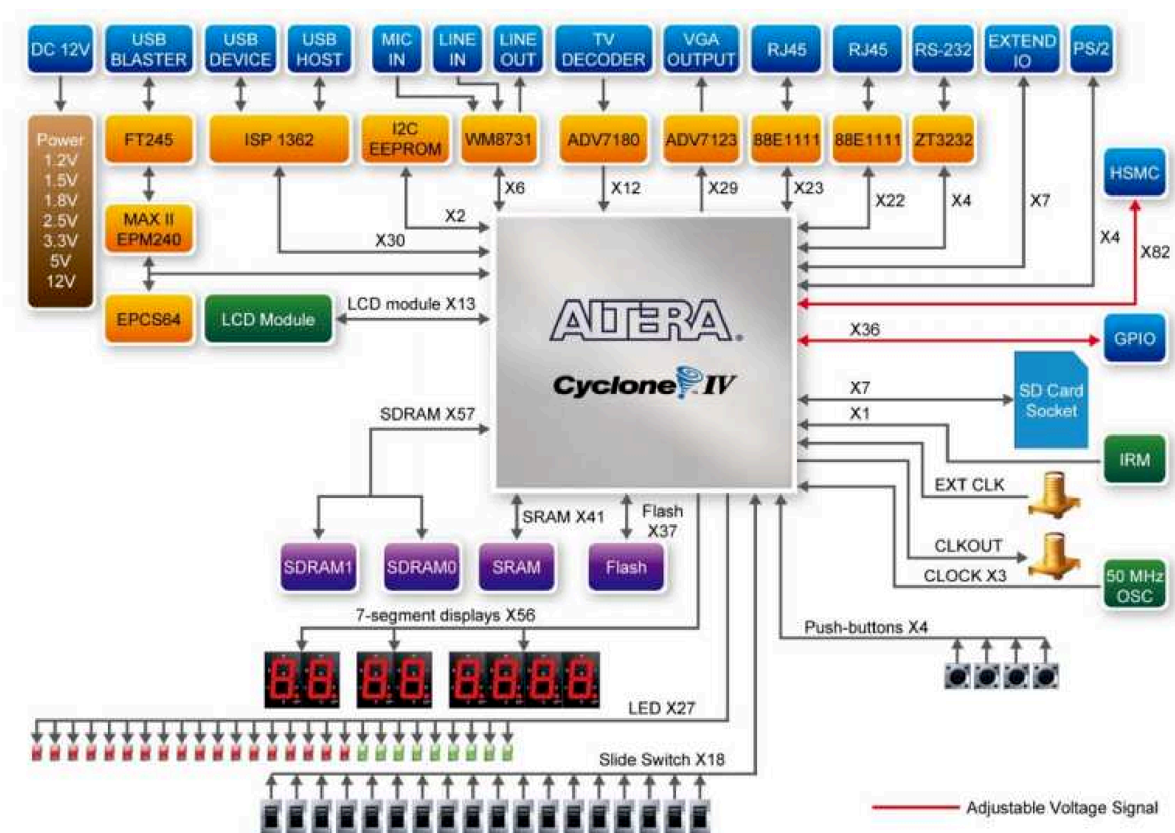
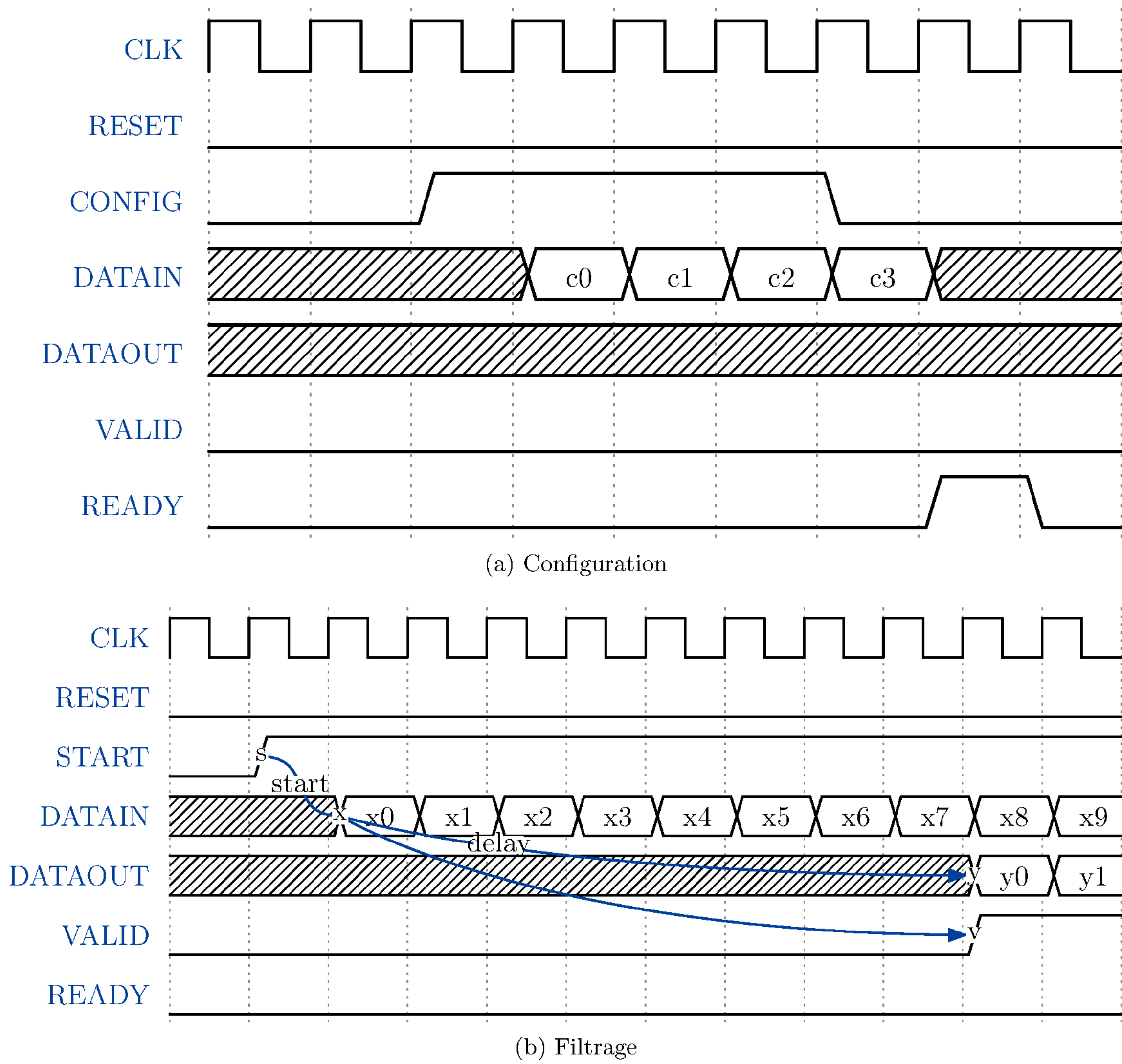


FIGURE 2 – Schéma bloc de la carte DE2-115 (Source : documentation technique Altera DE2-115)

B Utilisation du module FIR

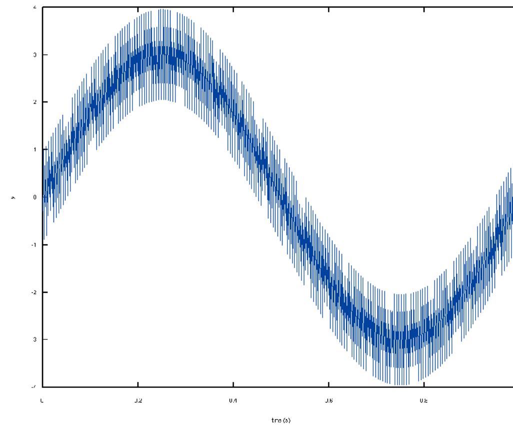


(a) Configuration

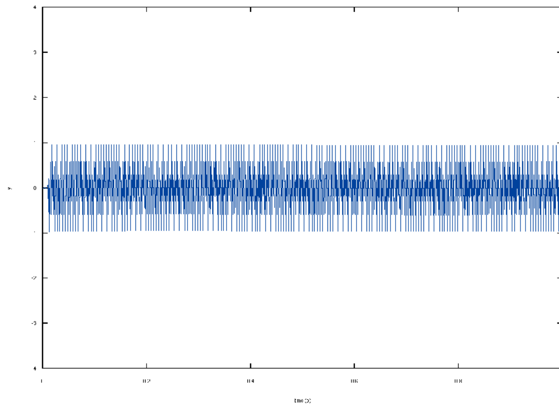
(b) Filtrage

FIGURE 3 – Formes d'onde de l'utilisation du filtre

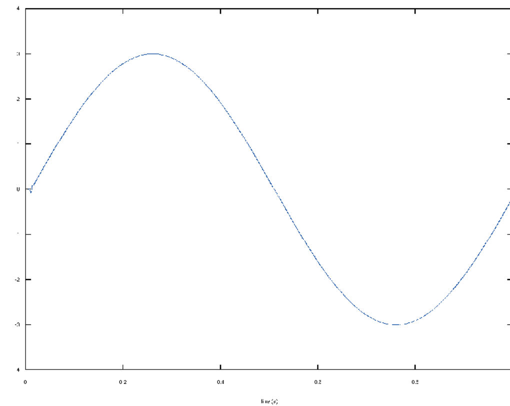
C Signaux de filtrage en C



(a) Signal d'entrée



(b) Signal de sortie d'un filtre passe-haut

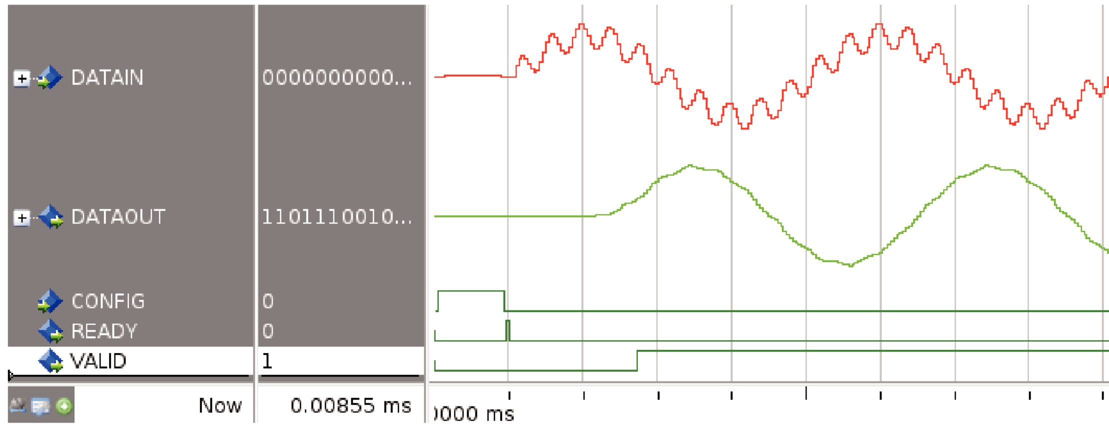


(c) Signal de sortie d'un filtre passe-bas

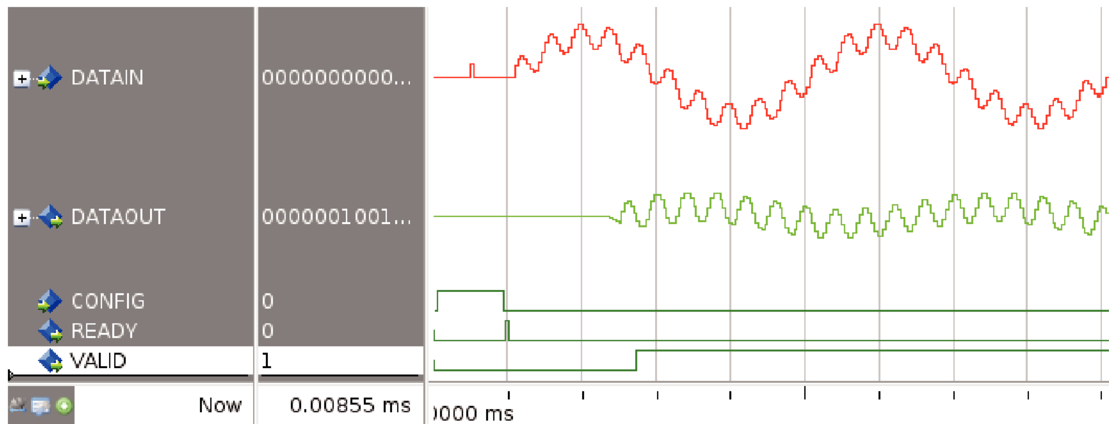
FIGURE 4 – Graphe en fonction du temps des signaux d'entrée et sortie du filtre dans différentes configurations.

D Signaux de l'IP FIR RTL

La Figure 5 illustre la simulation d'un filtre à 21 coefficients avec deux configurations différentes. La Figure 5a illustre les entrées et sorties du filtre avec une configuration passe-bas, lorsque la configuration passe-haut est illustrée sur la Figure 5b.



(a) Filtre passe-bas



(b) Filtre passe-haut

FIGURE 5 – Formes d'onde des signaux d'entrée (DATAIN) et sortie (DATAOUT) du filtre numérique dans différentes configurations

E Test du driver du filtre FIR

E.1 Filtre passe-haut

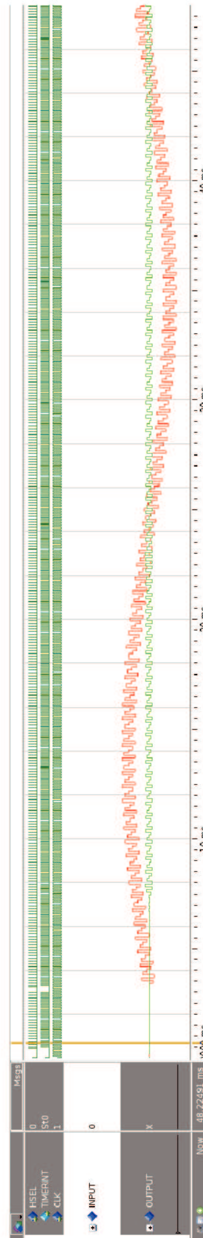


FIGURE 6 – Test du driver avec une configuration passe-haut

E.2 Filtre passe-bas

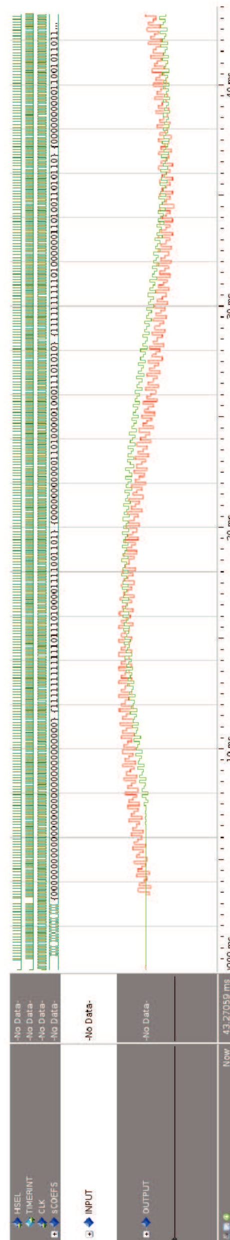


FIGURE 7 – Test du driver avec une configuration passe-bas

F Résultats de la vérification du FIR

La Figure 8 présente les résultats individuels de la vérification du filtre. Cette vérification a été faite avec 10000 valeurs aléatoires en entrée. Les assertions commencées par `COVER` indiquent les couvertures de séquences d'entrée et celles commencées par `ASSERT`, les assertions de propriétés.

On peut noter que presque toutes les fonctionnalités ont été couvertes, à l'exception des fonctionnalités 1.3.2 et 2.1.3. Il est aussi possible de noter que toutes les assertions ont été vérifiées — à l'exception de celles qui n'ont pas été couvertes.

Assertions

Ex	UNR	Name	Overall Average Grade	Assertion Status Grade
		(no filter)	(no filter)	(no filter)
		ASSERT_3_3_2_READ_WRITEONLY_REGISTERS	100%	100%
		ASSERT_3_3_1_WRITE_REGISTERS	100%	100%
		ASSERT_3_2_2_WRITE_READONLY_REGISTERS	100%	100%
		ASSERT_3_2_1_READ_REGISTERS	100%	100%
		ASSERT_2_2_4_INVALID_AFTER_CONFIG	100%	100%
		ASSERT_2_2_3_INVALID_AFTER_FINISHED	100%	100%
		ASSERT_2_2_2_VALID_AFTER_START	100%	100%
		ASSERT_2_1_4_NOT_FILTERING_WHILE_CONFIG	100%	100%
		ASSERT_2_1_3_FILTERING_QUICK_FINISH	! 0%	0%
		ASSERT_2_1_2_FILTERING_STOP_SIGNAL	100%	100%
		ASSERT_2_1_1_FILTERING_STARTS_IDLE	100%	100%
		ASSERT_1_3_3_RESET_WHILE_FILTERING	100%	100%
		ASSERT_1_3_2_RESET_WHILE_CONFIG	! 0%	0%
		ASSERT_1_3_1_RESET_WHILE_IDLE	100%	100%
		ASSERT_1_2_3_CONFIG_DOES_NOT_STOP	100%	100%
		ASSERT_1_2_2_CONFIG_WHILE_FILTERING	100%	100%
		ASSERT_1_2_1_CONFIG_WHILE_IDLE	100%	100%
		ASSERT_1_1_RESET_OUTPUTS	100%	100%
		ASSERT_ONLY_NON_SEQ	100%	100%
		COVER_3_1_5_READ_AFTER_WRITE	100%	n/a
		COVER_3_1_4_WRITE_AFTER_READ	100%	n/a
		COVER_3_1_3_WRITE_AFTER_WRITE	100%	n/a
		COVER_3_1_2_READ_AFTER_READ	100%	n/a
		COVER_2_2_4_INVALID_AFTER_CONFIG	100%	n/a
		COVER_2_2_3_INVALID_AFTER_FINISHED	100%	n/a
		COVER_2_2_2_VALID_AFTER_START	100%	n/a
		COVER_2_1_4_NOT_FILTERING_WHILE_CONFIG	100%	n/a
		COVER_2_1_3_FILTERING_QUICK_FINISH	! 0%	n/a
		COVER_2_1_2_FILTERING_STOP_SIGNAL	100%	n/a
		COVER_2_1_1_FILTERING_STARTS_IDLE	100%	n/a
		COVER_1_3_3_RESET_WHILE_FILTERING	100%	n/a
		COVER_1_3_2_RESET_WHILE_CONFIG	! 0%	n/a
		COVER_1_3_1_RESET_WHILE_IDLE	100%	n/a
		COVER_1_2_3_CONFIG_DOES_NOT_STOP	100%	n/a
		COVER_1_2_2_CONFIG_WHILE_FILTERING	100%	n/a
		COVER_1_2_1_CONFIG_WHILE_IDLE	100%	n/a
		COVER_1_1_POWER_ON_RESET	100%	n/a

FIGURE 8 – Résultats de la couverture fonctionnelle du filtre

G Schémas-bloc de l'accélérateur cryptographique

G.1 Exponentiation modulaire

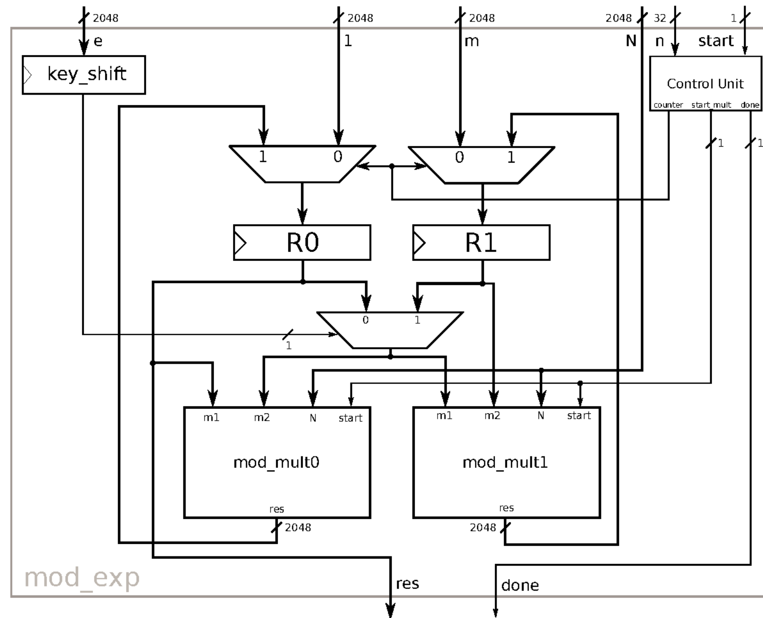


Figure 9 – Schéma bloc de l'exponentiation modulaire

G.2 Multiplication modulaire

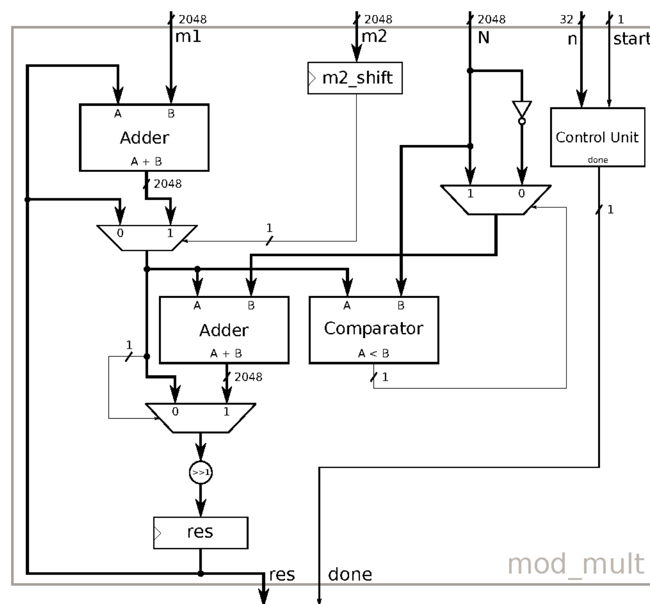


Figure 10 – Schéma bloc de la multiplication modulaire

G.3 Transformation de Montgomery

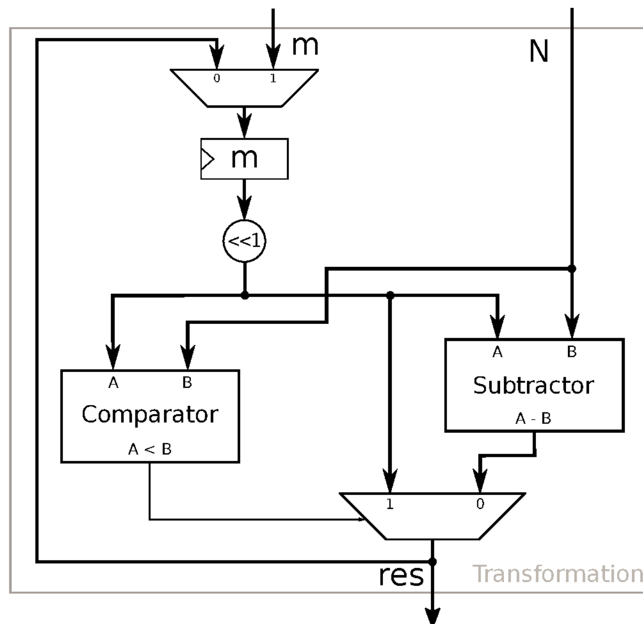


Figure 11 – Schéma bloc de la transformation de Montgomery

G.4 Réduction de Montgomery

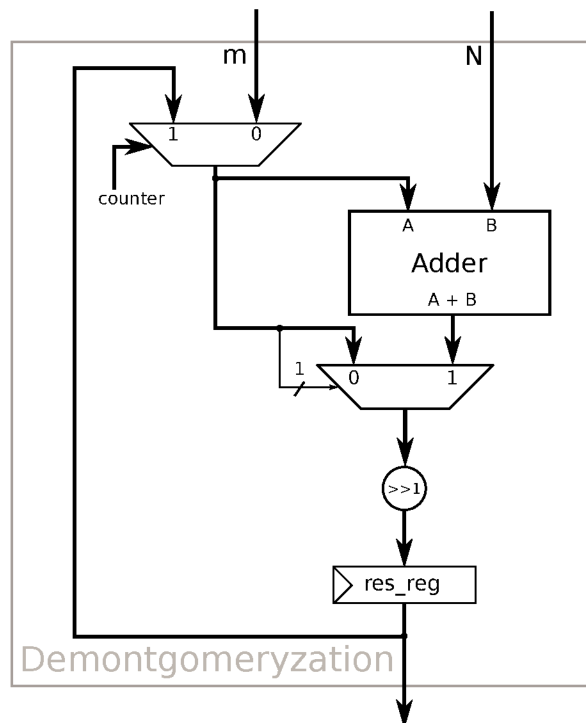


Figure 12 – Schéma bloc de la réduction de Montgomery