Auteur : Spinelli Isaia

Prof : [Etienne](https://cyberlearn.hes-so.ch/user/view.php?id=104149&course=1) Messerli

Ing : Sébastien Masle

Date : 05.06.2020

Salle : A09 (maison) – HEIG-VD

Classe : SOCF

Implémentation de l’algorithme de hachage MD5

[Système Soc intégré avec fpga (socf)](https://cyberlearn.hes-so.ch/course/view.php?id=14116#section-1)

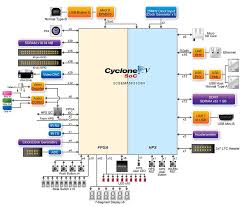


Table des matières

[Objectif - 2 -](#_Toc43304198)

[Introduction - 2 -](#_Toc43304199)

[Références - 2 -](#_Toc43304200)

[Conception - 3 -](#_Toc43304201)

[Plan d’adressage - 3 -](#_Toc43304202)

[Gestion globale - 4 -](#_Toc43304203)

[Open et close - 4 -](#_Toc43304204)

[Read - 5 -](#_Toc43304205)

[memcpy(footprint, (const void \*)addr\_footprint, FOOTPRINT\_SIZE ); - 5 -](#_Toc43304206)

[Write - 5 -](#_Toc43304207)

[Réalisation - 6 -](#_Toc43304208)

[Description VHDL - 6 -](#_Toc43304209)

[Création du driver - 7 -](#_Toc43304210)

[Probe et exit - 7 -](#_Toc43304211)

[Open - 7 -](#_Toc43304212)

[Close - 7 -](#_Toc43304213)

[Read - 7 -](#_Toc43304214)

[Write - 7 -](#_Toc43304215)

[Padding - 7 -](#_Toc43304216)

[Validation - 8 -](#_Toc43304217)

[Premier test - 8 -](#_Toc43304218)

[Second test - 8 -](#_Toc43304219)

[Aligné et Pas aligné - 9 -](#_Toc43304220)

[Annexes - 10 -](#_Toc43304221)

[Conclusion - 10 -](#_Toc43304222)

[Difficultés rencontrées - 10 -](#_Toc43304223)

# Objectif

Ce laboratoire a pour but de réaliser notre première conception d’un système réel en implémentant l’algorithme de hachage MD5.

# Introduction

MD5 est un algorithme classique de hachage utilisé dans de nombreuses applications. Le but est de réaliser une implémentation FPGA pour accélérer le calcul de cet algorithme. Notre application doit pouvoir calculer le hash de n'importe quel fichier.

Si on a une implémentation software du MD5, on peut taper : **#./md5 mon\_fichier**. On obtient comme résultat un hash de 128b bits. Exemple : 595F44fec1e92a71d3e9e77456ba80d1

Votre objectif est de créer un device (et son driver) de telle manière que l’on puisse l’exécuter sur linux. Exemple : **#cat mon\_fichier > /dev/votre\_device\_MD5** Suivi de **#cat /dev/votre\_device\_MD5** et de récupérer un hash. Exemple : 595F44fec1e92a71d3e9e77456ba80d1

Pour nous simplifier la réalisation de ce laboratoire, on nous a fournis une IP de l'algorithme MD5 permettant de l'implémenter dans la partie FPGA du SoC. L'IP fournie permet de calculer le hash pour un bloc de 512 bits.

# Références

Différentes implémentations logicielles peuvent être trouvées sur internet, comme par exemple sur le site *https://rosettacode.org/wiki/MD5* qui dispose d’un vaste catalogue d’implémentation.

Sur l’algorithme en lui-même, il y a également beaucoup de documentation. Wikipédia est un point de départ très utile *https://en.wikipedia.org/wiki/MD5* ; si vous voulez tout savoir sur l’algorithme, le standard est disponible sur https://tools.ietf.org/html/rfc1321.

# Conception

Dans ce chapitre, je vais présenter comment je pense concevoir le système complet demandé.

## Plan d’adressage

Voici mon plan d’adressage :



La constante à l’offset 0x00 (0xDEADBEEF), disponible uniquement en lecture, permet de m’assurer que la communication fonctionne bien.

Les « wb » de l’offset 0x00 à 0x3C, disponible uniquement en écriture, permet de transmettre les 512 bits à envoyer au composant MD5. Je les ai bien alignées fait suivre afin de pouvoir utiliser une fonction comme « memcpy » au lieu d’effectuer personnellement plusieurs copies.

À l’offset 0x100, au bit 0, il y a l’enable du composant MD5 disponible en lecture et en écriture. Celui-ci permet directement de contrôler l’activation du composant MD5.

Ensuite, « busy » est disponible uniquement en lecture à l’offset 0x104 sur le bit 0. Ce signal « busy » permet d’indiquer si le composant MD5 est en cours de traitement pour calculer un footprint. Il est important car il est nécessaire de recharger la valeur d’initialisation du composant MD5 par le dernier footprint calculé.

Finalement, la sortie footprint du composant MD5 est disponible uniquement en lecture à l’offset 0x200 jusqu’à 0x20C. Encore une fois, ceux-ci sont aligné afin de lire les 16 bytes plus rapidement avec la fonction « memcpy ».

Les adresses non utilisé et les adresse réservée, mentionnée comme « Reserved », indique qu’une écriture et une lecture n’est pas pris en charge à ces offsets.

## Gestion globale

Afin de bien comprendre les différentes étapes du système, j’ai décider de faire un chronogramme :

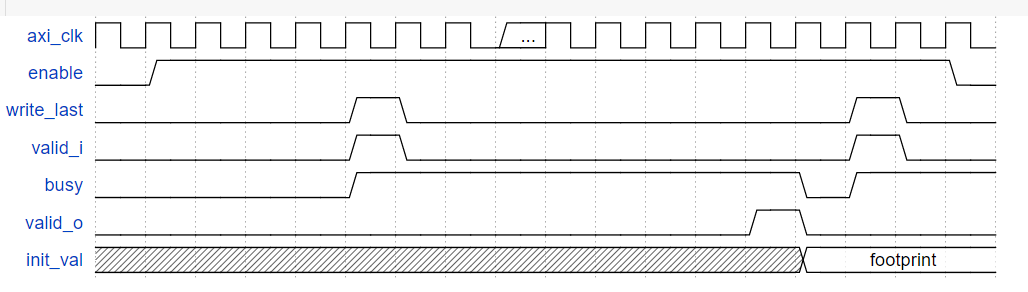


Figure 0‑1 : Chronogramme

Pour commencer, il est important d’activer l’enable avant le premier traitement.

Juste avant de mettre à jour les dernières données (32 bits) d’un bloc, il est important de vérifier que le signal « busy » est à l’état bas, comme présenté avec les flèches jaune. L’écriture des 32 derniers bits d’un bloc est représenté par le signal « write\_last ».

Une fois que les dernières données du bloc ont été envoyées, cela active directement le signal valid\_i le temps d’un seul coup de clock. Une fois ceci fait, le composant MD5 va pouvoir traiter le bloc de 512bits précédemment passé.

En attendant la fin de l’algorithme de hachage, comment indiqué par le cadre vert, il est déjà possible de préremplir les données du prochain bloc. Cependant, les derniers 32 bits du bloc ne doivent pas être envoyés avant que le signal busy soit à 0.

Le signal busy est à 1 lorsque le composant MD5 est en traitement d’un bloc. Une fois ce bloc traité, le signal « valid\_o » se lève à 1 pendant 1 coup de clock et le signal busy se remet à 0.

Au moment où le signal « valid\_o » se lève, un footprint valide est généré. Dans le cas ou d’autres bloc doivent être traité, il est nécessaire de mettre à jour la valeur d’initialisation du composant MD5 par la valeur du précèdent footprint. Ceci peut donc être fait directement dans le VHDL.

Une fois tous les blocs traités, il est préférable de désactiver le composant MD5 en descendant le signal enable.

### Open et close

Avant chaque lecture ou écriture sur un device, celui-ci doit impérativement être ouvert. De même, après chaque lecture ou écriture, le device doit être refermé. Il est aussi possible lors de l’ouverture ou fermeture de savoir si cela pour une opération de lecture ou d’écriture. Étant donné que la fonction d’écriture et lecture peuvent s’appeler « récursivement » dans le cas ou beaucoup de donné doivent être gérée, il peut être intéressant de gérer le signal enable lors de l’ouverture et la fermeture du device.

De ce fait, je pense activer le composant MD5 lors de l’ouverture du device dans le cas ou une opération d’écriture est demandé.

\*ADDR\_ENABLE\_MD5 = 1 *(enable\_s <= 1 / busy\_s <= 0 / init\_val <= VAL\_INIT\_CST)*

*Remarque : à gauche voici en bref ce que le code C va effectuer. À droite, voici en bref ce que code VHDL va effectuer.*

Donc le composant MD5 sera activé lors de l’ouverture du device si une opération de lecture est souhaitée. Dans le code VHDL, il sera aussi possible de mettre le signal busy à 0 et d’assigner à la valeur d’initialisation du composant la valeur désirée.

Dans le cas de la fermeture du device :

\*ADDR\_ENABLE\_MD5 = 0  *(enable\_s <= 0 / busy\_s <= 0 / init\_val <= VAL\_INIT\_CST)*

*Il est intéressant de désactiver le composant en mettre le signal enalbe à 0. Afin de simplifier le code, à chaque écriture du signal enable, le signal busy et init\_val seront mis à jour.*

### Read

Dans la fonction read du device, nous souhaitons simplement lire la valeur du dernier footprint en sortie du composant MD5. De ce fait, il suffire de lire les 16 bytes qui se suivent dans le plan d’adressage précédemment présent. Voici à quoi ressemlbra le code :

*memcpy(footprint, (const void \*)addr\_footprint, FOOTPRINT\_SIZE );*

« footprint » sera un tableau de 16 bytes où l’on souhaite recevoir la valeur du footprint. « Addr\_footprint » sera l’adresse virtuelle correspondant au plan d’adressage afin de pointer sur la valeur du footprint. Finalement, FOOTPRINT\_SIZE vaudra 16 afin de bien lire les 16 bytes du footprint.

### Write

Lors de l’écriture dans un device, une application de l’algorithme de hachage MD5 est souhaité. Voici les opérations qui devront être effectuées, comme précédemment, à gauche représente le code C et à droite (en parenthèse) le code VHDL.

Lecture et préparation des paquets de 512 bits avec padding si nécessaire

Boucle sur chaque paquet de 512 bits

Envoie des (512-32) premiers bits  *(wb\_s(idx+31 down to idx) <= wb\_32\_s)*

Tant que le md5Core est occupé, attend

Envoie des 32 derniers bits du paquet *(wb\_s(511 down to 480) <= wb\_32\_s* ***/*** *valid\_s <= 1* ***/****busy <= 1)*

… *( if valid\_o\_s = ‘1’ --> busy <= 0* ***/*** *init\_val\_i <= footprint\_s* ***+ init\_val\_i****)*

Attend la dernière transaction avant de terminer

Voici en bref les opérations qui devront être effectués dans la fonction d’écriture du device.

*Remarque : Il est essentiel d’additionner par 4 vecteurs la valeur d’init avec la valeur du footprint de sortie afin d’obtenir un bon footprint (sans les overflow) à assigner dans l’init val pour le prochain paquet. Ceci n’est pas fait dans l’IP, de ce fait, il est préférable de le faire dans le VHDL comme ajouté en rouge.*

# Réalisation

Dans ce chapitre, je vais présenter comment j’ai réellement effectué le système complet.

## Description VHDL

Pour commencer, j’ai repris le projet du laboratoire « AXI light » afin de directement bénéficier de l’implémentation de la communication entre le HPS et la FPGA. Cependant, je tiens à préciser qu’afin d’éviter de perdre trop de temps, je n’ai pas corrigé les quelques erreurs dans cet IP étant donné que les lectures et les écriture s’effectuait tout même.

Ensuite, je me suis familiarisé avec l’IP fourni « Md5Core.v » grâce au test bench. Ultérieurement, j’ai enlevé de mon ancienne implémentation de l'AXI light toutes les entrées/sorties non utilisées.

Après avoir une implémentation plus propre du bus AXI light, j’ai déclaré et instancié le composant fourni « Md5Core ». J’ai donc créé tous les signaux utiles à la gestion de ce composant.

Ensuite j’ai pu respecter mon plan d’adressage en ajouter les différents cas de lecture et d’écriture.

Finalement, j’ai ajouté les quelques lignes de gestion des signaux liés au composant « Md5Core » comme présenté dans le chapitre de conception.

Remarque : Le code VHDL complet est en annexe «axi4lite\_slave ».

## Création du driver

### Probe et exit

La fonction probe est appelée lors de l’insertion du module. Cette fonction va me permettre de mapper l’adresse physique du bus AXI light (0xFF200000) en adresse virtuelle afin d’accéder à ces adresses depuis le driver.

Ensuite, je vais lire le contenu à l’adresse de base et je vérifie que le contenu vaut bien la constante fixé dans le plan d’adressage (0xDEADBEEF). Dans le cas où la valeur lue n’est pas correcte, la fonction échoue.

La fonction exit est appelée lors de la suppression du module. Dans ce cas, il est important de supprimer toutes les ressources allouée dans le driver.

### Open

Comme présenté dans le chapitre de conception, cette fonction va être appelée avant toutes lectures et écritures dans le device. Elle va permettre d’activer le composant MD5 en écrivant ‘1’ à l’adresse du enable dans le cas ou le device est ouvert pour une écriture.

### Close

Comme présenté dans le chapitre de conception, cette fonction va être appelée après toutes lectures et écritures dans le device. Elle va permettre de désactiver le composant MD5 en écrivant ‘0’ à l’adresse du enable dans le cas où le device est fermé après une écriture.

Cependant, dans le cas où le messages à traiter est aligné, il est nécessaire d’ajouter un dernier paquet qui correspond à un padding avec la taille total du message en bits sur les 64 dernier bits du paquet.

### Read

Comme présenté dans le chapitre de conception, cette fonction va être appelée lors d’une lecture du device. De ce fait, elle lit les 4 mots de 32 bits qui représente le footprint du dernier traitement de hachage. Ensuite, elle transforme ces derniers en hexadécimal dans une string, afin de retourner cette string dans le buffer utilisateur. Ceci va donc comme prévu afficher à l’écran la chaine en hexadécimal de la valeur du dernier footing correcte.

### Write

Comme présenté dans le chapitre de conception, cette fonction va être appelée lors d’une écriture du device. De ce fait, elle va gérer toutes l’écriture de l’entrée sur le device dans le composant MD5. Elle réalise ceci en écrivant paquet par paquet de 512 bits vers le composant. De plus, si l’entrée n’est pas alignée, elle ajoute les données nécessaire au padding.

Vous pouvez observer le fonction « write » commenté en annexe.

### Padding

Pour bien comprendre le padding je me suis aidé du site (<https://fthb321.github.io/MD5-Hash/MD5OurVersion2.html>).

Après plusieurs test et débogage, je me suis un peu perdu dans mon code. C’est pourquoi je pense que ma façon de gérer le padding n’est pas très jolie.

Malgré le manque de charme de ma gestion du padding, elle tout de même fonctionnelle d’après mon premier test de validité et c’est qui est le plus important dans un premier temps.

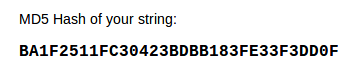
Si vous souhaitez comprendre ma gestion, vous pouvez observer la fonction « write » et « close ». Le code est commenté et en annexe.

## Validation

### Premier test

Afin de faire un premier test, je souhaitais le bon fonctionnement du système complet avec le simple texte « 123 ». Cependant, avec la commande echo, il ajoute un « lineFeed » (retour à la ligne). De ce fait, j’ai vérifié sur un outil en ligne la résultat du texte « 123<LineFeed> ».

Après quelques corrections, ce résultat correspondait bien au résultat lu avec un cat sur mon device.

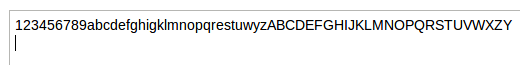


Ce test n’a pas tout de suite fonctionné. Il a fallu ajouter l’addition des 4 vecteurs, expliqué dans l’algorithme MD5 qui n’est pas dans l’IP, ce que je n’avais pas vu. Après avoir ajouté cette addition dans le VHDL, ce test fonctionnait.

### Second test

Après avoir réussi à générer un hash correcte avec un unique paquet, je souhaitais faire de même avec un paquet de plus afin de m’assurer que tout se passe bien avec plus d’un paquet.

De ce fait, j’ai voulu tester cette entrée :



*Remarque : Toujours avec un retour à la ligne à la fin.*

Cette entrée comporte 488 bits. D’après le site précédemment mentionné « fthb321 » , il est nécessaire d’ajouter un autre paquet afin d’ajouter la taille en bits (488) sur 64 bits dans le message. De ce fait, deux paquets doivent être envoyées en comptant le padding.

Cependant, le résultat attendu était incorrect. Il y avait encore une erreur dans mon système qui faisait que plus d’un paquet ne fonctionnait pas.

Afin de comprendre d’où pourrait venir l’erreur, j’ai décider de débugger mon système en utilisant l’outil Signal Tap de Quartus.

Après la configuration, l’initialisation et la recompilation du projet, j’ai pu faire fonctionner Signal Tap. Ce qui, je dois l’admettre, prend énormément de temps uniquement pour une recompilation rapide :



Malgré ceci, j’ai continué à chercher où pourrait se trouver le problème.

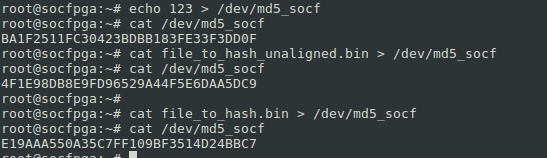
J’ai compris que les assignations faites lors d’un nouveau footprint valide n’était pas faite de manière attendue. De ce fait, j’ai contrôlé mon code VHDL. J’ai pu voir pourquoi les assignations n’était pas faite de manière espérée.

En fait, je faisais l’addition des 4 vecteurs, la concaténation des résultats et l’assignation du résultat dans le même process, sur le même flanc montant. Cependant, les valeurs ne sont pas réellement opérationnelles avant la fin du process. C’est pourquoi, j’ai ajouté deux « std\_logic » afin de résoudre ce problème. A voir aux ligne 278 – 331 de mon code de l’IP en annexe.

Après ce changement, toutes les futurs vérifications fût correcte. En effet, la gestion de mon padding n’est pas splendide mais correcte.

### Aligné et Pas aligné

On peut voir ci-dessous le bon fonctionnement du système avec les vérifications des entrées « 123\n », le fichier non alignée fourni et le fichier aligné fourni.



Les footprints relus correspondes bien avec les hash attendus.

# Annexes

Voici la liste dans l’ordre des annexes :

1. Code du driver
2. Code de mon IP VHDL

# Conclusion

Je dois avouer qu’au début du laboratoire j’étais perdu et avais peur de tout le travail demandé. Finalement, sans prendre en compte les temps de compilations extrêmement long, j’ai beaucoup apprécié ce laboratoire.

## Difficultés rencontrées

* Le débogage
* Le temps de compilation
* La compréhension du fonctionnement de l’IP

J’ai réussi à mettre en place toutes les étapes qui m’était demandé dans ce laboratoire. Les description VHDL sont synthétisable et intégrable. Je suis particulièrement fier d’avoir réussi à faire complétement le travail demandé.

Je tiens à remercier M. Masle pour tout le temps qu’il m’a consacré à m’aider.

Date : 17.06.20

Nom de l’étudiant : Spinelli Isaia