الجمهورية الجزائرية الديمقراطية الشعبية

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Yahia Farès de MEDEA

Faculté de Technologie

Département de Génie Électrique

Mémoire de master

Mémoire de master

***Spécialité* :Electronique des Systèmes Embarqués**

Thème:

Etude et implementation d'un IP core de l'algorithme de hachage cryptographique SHA-2 sur un circuit FPGA

Réalisé par :

Hachemi Houria
Elaoua Abrar

Encadré par :

OuldZmirli Mohamed
Toubal Abdelmoughni

ANNEE UNIVERSITAIRE: 2017/2018

Mémoire préparé au sein du Laboratoire des Systèmes Electroniques Avancés (LSEA) de l'Université de Médéa

ملخص :

في هذا المشروع، استخدمنا أداة تطوير Xilinx XPS لإنشاء بنية أجهزة تعتمد على softcoreMicroblaze ، بالإضافة إلى نواة IP تقوم بتنفيذ خوارزمية التجزئة SHA256. ثم صممنا بواسطة أداة SDK برنامج يتم تنفيذه من قبل Microblaze ، والذي يلعب دورًا في اختبار لـ IP core. تم التحقق من الاختبارات العملية على بطاقة DiglilentNexys 2 Spartan 3E.

كلمات مفتاحية:codesign، FPGA ، تجزئة التشفير ، IP core، SHA256

Résumé :

Dans ce projet, nous avons utilisé l'outil de développement XPS de Xilinx pour créer une architecture matérielle à base d'un softcore de type Microblaze, ainsi qu'un IP core qui implémente d'une manière hardware l'algorithme de hachage SHA256. Ensuite nous avons conçu par l'outil SDK un programme exécuté par le Microblaze, qui joue un rôled'un testbench de l'IP core. Des tests pratiques ont été validés sur une carte de type Diglilent Nexys 2 Spartan 3E.

Mots clés: codesign, FPGA, hachage cryptographique, IP core, SHA256

Abstract:

In this project, we used the Xilinx XPS development tool to create hardware architecture based on a Microblazesoftcore, as well as an IP core that implements the SHA256 hashing algorithm. Then we designed by the SDK tool a program executed by the Microblaze, which plays a role of a testbench of the IP core. Practical tests have been validated on a DiglilentNexys 2 Spartan 3E card.

Key words:codesign, cryptographic hashing,FPGA, IP core, SHA256

Sommaire

Résumé i

sommaire ii

Liste des figures iv

Liste des acronymes vi

Introduction générale.....1

Chapitre 1 : Fonctions de hachage cryptographiques

1 Introduction 2

2 Les fonctions de hachage 2

3 Les attaques des fonctions hachage 3

4 Présentation des fonctions de la famille SHA 4

4.1 L'algorithme SHA-0 4

4.2 Sécurité de la fonction SHA-0 4

4.3 L'algorithme SHA-1 5

4.4 Sécurité de SHA-1 5

4.5 L'algorithme de SHA-2 5

4.6 SHA-3 6

5 Description de la fonction de hachage SHA256 6

5.1 Bourrage 6

5.2 Découpage en blocs 7

5.3 Initialisations 7

5.4 Calcule du condensé (haché) 8

6 Domaines d'utilisation des fonctions de hachage cryptographiques 12

6.1 Intégrité des données 12

6.2 Protection de mots de passe 12

6.3 Signature électronique 13

6.5 Génération de nombres pseudo-aléatoires 14

7 Conclusion 14

Chapitre 2 : Systèmes embarqués sur FPGA

1 Introduction …… …….15

2 Circuits logiques programmables ……15

4.1 L'algorithme SHA-0 ……16

3 Codesign: Conception conjointe logiciel/matériel ……...17

3.1 Flotde la conception conjointe logiciel/matériel ……19

3.2 Les composants virtuels (IPs) ……19

4 Les systèmes sur puce (SoC) sur FPGA …….. 21

4.1 Le processeur MicroBlaze de Xilinx ……..23

4.2 Communications sur puce ……... 24

4.2.1 Processor Local Bus (PLB) …… 25

4.2.2 On-chip Peripheral Bus (OPB) …..…… 26

4.2.3 Bus de contrôle de périphérique (DCR)……...26

5 Les Outils de développement de Xilinx …….. 27

6 Conclusion ……27

Chapitre 3 : Développement d'un IP core pour l'algorithme de hachage cryptographique SHA256

1 Introduction …… …….28

2 Description et fonctionnement du module globale SHA256 utilisé……..28

3 Simulation du module de hachage SHA256 ...…….33

3.1 Résultat du 1er test …….. 33

3.1 Résultat du 2eme test …….. 33

3.1 Résultat du 3eme test …….. 33

4 Conception conjointe logiciel/matériel d'un IP core SHA256 …….37

4.1 Etape N°1 : Création de l'IP core ……37

4.2 Etape N°2 : Finalisation de l'IP core SHA256 ……...…….44

4.3 Etape N°3 : Intégration de l'IP core dans le système embarqué ..……49

4.4 Etape N°4 : Conception de la partie logicielle …….57

5 Résultats des tests pratiques …….58

6 Conclusion …….. 60

Conclusion générale……..61

Liste des figures

Chapitre I

Figure I 1 Principe d'une fonction de hachage ……2

Figure I 2 Historique des principales fonctions de hachage……..…3

Figure I 3 Fonctionnement de la famille SHA ……..4

Figure I 4 La fonction de hachage SHA-2 ……..……11

Figure I 5 Principe de signatures électroniques……...13

Chapitre II

Figure II 1 Les 4 familles des circuits PLD ……...……16

Figure II 2 Architecture interne du FPGA fabriqué Xilinx ……17

Figure II 3 Conception traditionnelle et lecodesign …….. 18

Figure II 4 Flot de conception dans le codesign ……..……19

Figure II 5 Architecture de base d'un système sur puce ……..…...21

Figure II 6 Architecture de base d'un SoPC sur FPGA de type XilinxZinq …….……..…...23

Figure II 7 Architecture interne du softcoreMICROBLAZE…….. ….……..…...24

Figure II 8 Communications sur puce ……...…….……25

Chapitre III

Figure III 1 Schéma bloc du module SHA256 utilisé ……28

Figure III 2 Chronogramme des signaux d'E/S dans le cas du premier bloc du message ….30

Figure III 3 Chronogramme des signaux d'E/S dans le cas du dernier bloc du message …..30

Figure III 4 Diagramme bloc du module SHA256 utilisé ……..31

Figure III 5 Architecture de calcul des mots W0-W15 ……..……32

Figure III 6 Architecture de calcul des mots W0-W15 ……..……32

Figure III 7 Résultat de simulation du 1er test. ……..34

Figure III 8 Résultat de simulation du 2eme test. ……..35

Figure III 9 Résultat de simulation du 3eme test. ……..36

Figure III 10 Xilinx Platform Studio …… 37

Figure III 11 Création d'un IP core ……38

Figure III 12 Création d'un IP core ……38

Figure III 13 Chemin de l'IP core ……..……38

Figure III 14 Nommage de l'IP core ……..…39

Figure III 15 Choix du type de bus ……..……..……..…39

Figure III 16 Services de l'IP core ……...……..…40

Figure III 17 Configuration du bus de l'interface…..……..…41

Figure III 18 Configuration du nombre de registres de l'IP core ……..…42

Figure III 19 Interconnexion de l'IP core ……...……...…42

Figure III 20 Génération les fichiers de simulation de l'IP core …..……..…43

Figure III 21 Sélection du support d'implémentation de l'IP core ……..…..…44

Figure III 22 Liste des répertoires du projet ……..…..……..…45

Figure III 23 Contenu du dossier "pcores -> sha256_ip_v1_00_a "……...…46

Figure III 24 Dossier "pcore s-> sha256_ip_v1_00_a -> hdl -> vhdl"……..…46

Figure III 25 Modification apportée au code source "user_logic"……..…49

Figure III 26 Fichier sha256_ip_v2_1_0.pao ……...…50

Figure III 27 Assistant de création ou chargement de l'IP core …… .50

Figure III 28 Import de l'IP core ……..……..…51

Figure III 29 Types de fichiers sources de l'IP core ……..……..……..…51

Figure III 30 Indication du chemin du fichier pao ……..……..…52

Figure III 31 Analyse des fichiers du projet ……..… 52

Figure III 32 Choix du bus PLB ……...…53

Figure III 33 Configuration bus SPLB de l'IP core ……..… 53

Figure III 34 Configuration paramètres bus SPLB ……..… 54

Figure III 35 Sélection mode des interruptions.……..……..… 54

Figure III 36 Visualisation dans « IP Catalog de IP core ……..……..…. 55

Figure III 37 Configuration de l'IP core ……..…. 56

Figure III 38 Nouveau composant ajouté au système ……..……..56

Figure III 39 Configuration des zones d'adresses de module sha256……..…. 57

Figure III 40 Environnement SDK avec le projet développé ……..…. 57

Figure III 41 Résultat du 1er test pratique ……..……... 58

Figure III 42 Résultat du 2eme test pratique ……..……... 58

Figure III 43 Résultat du 3eme test pratique ……..……... 59

Liste des tables

Chapitre III

Tableau III 1 description des signaux d'Entrées/Sorties du module SHA256 ……28

Acronyms

ARM : Advanced **R**isc**M**achine

ASIC: Application-**S**pecific **I**ntegrated **C**ircuit

CLB : Configurable **L**ogic **B**loc

IOB : Input/**O**utput**B**lock

PLB : Processor **L**ocal **B**us

CPU: Central **P**rocessing **U**nit

CPLD : Complex**P**rogrammable**L**ogic**D**evice

FPGA : Field Programmable Gate Arrays

ELF: Executable and **L**inking **F**ormat

SDK : Software **D**evelopment **K**it

EDK : Embedded **D**esign **k**it

FPU:Floating **P**roduction **U**nit

HDL : Hardware **D**escription **L**anguage

HW/SW : Hard**W**are/**S**oft**W**are

IBM : International **B**usiness **M**achines **C**orporation

IP : Intellectual **p**roperty

ISE : Integrated**S**ystem **E**nvironnement

OPB : On-chip **P**eripheral **B**us

JTAG : Joint **T**est **A**ction **G**roup

LUT : Look-**U**p **T**able

MD4/MD5 : Message **D**igest**4/ M**essage **D**igest**5**

NSA : National **S**ecurity **A**gency

LMB : Local **M**emory **B**us

NIST : National**I**nstitute of **S**cience and **T**echnology

O(2^{63}) : La **C**omplixité

GPIO : General-**P**urpose **I**nput/**O**utput

PLD : Programmable **L**ogic **D**evice

SHA : Secure **H**ash **A**lgorithm

SOC : System **O**n**C**hip

DCR : Device**C**ontrol **R**egister

SOPC: System **O**n**P**rogrammable **C**hip

SSL : Secure **S**ockets **L**ayer

RTL: Register **T**ransfer **L**evel

UART : Universal **A**synchronous **R**eceiver **T**ransmitter

USB : Universal **S**erial **B**us

VHDL : Very High **H**ardware **D**escription **L**anguage

VGA : Video **G**raphics **A**rray

XPS : Xilinx **P**latform **S**tudio

FSL : Fast **S**implex **L**ink

Introduction générale

Dans ce mémoire, nous nous intéressons aux fonctions de hachage cryptographiques, et plus particulièrement à la fonction SHA (Secure Hash Algorithm), qui a été conçue par la National Security Agency des Etats-Unis (NSA).

Les fonctions de hachage cryptographiques font correspondre à un message de taille arbitraire, une sortie de taille fixe (256 bits dans le cas de ce travail), appelée haché. L'idée est que toute modification, même minime, du message d'entrée, doit induire des modifications importantes et imprédictibles du haché en sortie. Par exemple, l'une des propriétés recherchées est qu'il doit être très difficile pour un attaquant de trouver une collision, c'est-à-dire deux messages distincts ayant un haché identique.**[3]**

Le présent projet consiste au développement et à l'évaluation d'un IP core pour l'algorithme de hachage SHA256, intégré dans un système embarqué à base d'un softcore de type Microblaze. Nous utiliserons une conception conjointe logiciel/matériel (conception hybride).
Enfin, nous exposons les résultats des tests pratiques réalisés avec sur une plateforme de développement de type DigilentNexys 2-500C à base d'un circuit FPGA de type Spartan 3E.

Le mémoire est organisé comme suit :
Le premier chapitre fournit un bref aperçu sur les fonctions de hachage cryptographiques de la famille SHA, et une description détaillée des différentes étapes de l'algorithme SHA-256.
Le deuxième chapitre présente la conception des systèmes embarqués, et en particulier l'environnement de la description de l'architecture matérielle/logicielle qui cible les circuits FPGA.
Dans le dernier chapitre, nous utiliserons l'outil de développement XPS de Xilinx pour créer l'architecture matérielle d'un système embarqué à base du Microblaze et d'un IP core qui implémente l'algorithme SHA256, ensuite, nous utiliserons l'outil SDK pour concevoir un programme de test de l'IP core SHA256, conçu en langage C et qui sera exécuté sur le Microblaze. Enfin, nous présentons les résultats de trois tests pratiques du système développé sur la carte de type DiglilentNexys 2 Spartan 3E.

1 Introduction

Dans ce chapitre nous présentons en général les fonctions de hachage cryptographiques de la famille SHA, et en détail l'algorithme de hachage SHA256. Les travaux présentés ici abordent à la fois la conception et l'analyse de cette fonction. Enfin, nous terminons par les domaines d'applications.

2 Les fonctions de hachage

La fonction de hachage code une empreinte, elle est censée d'identifier un message. C'est la définition d'un résumé du message qui permettra, non de le reconstituer, mais de l'identifier si on dispose au préalable d'une table de correspondance. Cela fonctionne comme une empreinte digitale qui ne permet pas de reconstituer d'autres caractéristiques d'un individu, mais qui permet de l'identifier. Les fonctions de hachage sont particulièrement utiles en cryptographie. Elles permettent notamment de diminuer la quantité d'information à crypter. Si l'image de x par la fonction de hachage est appelée l'empreinte de x, on peut par exemple ne crypter que l'empreinte. D'autre part, elles permettent de mettre au point des protocoles de signature électronique et d'authentification des messages, ainsi que de vérifier l'intégrité d'un document, au même titre que le bit de parité, qui lui-même est une fonction de hachage particulière.

Formellement, une fonction de hachage $H : \{0,1\}^{*} \longrightarrow \{0,1\}^{n}$ est une application qui transforme une chaîne de taille quelconque en une chaîne de taille fixe n, comme le montre la figure illustre sur la figure 1.**[1][2]**

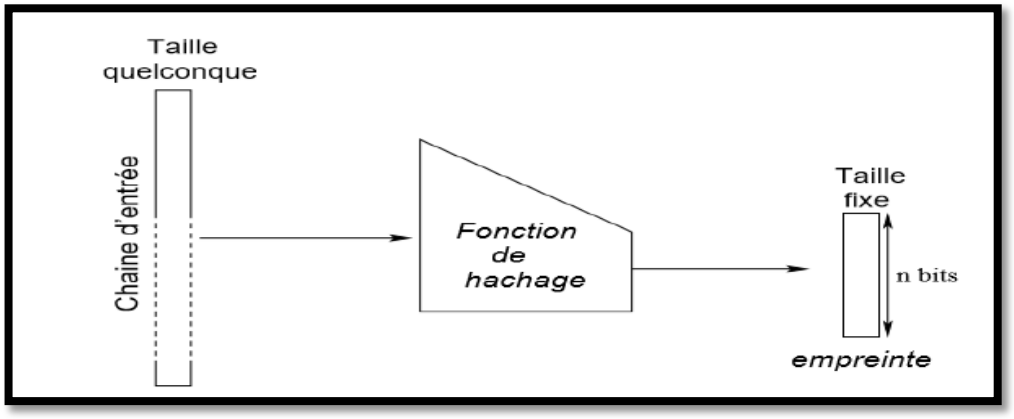


Figure I1 *Principe d'une fonction de hachage.*

La figure I.2 montre un rapide historique des principales fonctions de hachage en explicitant les dérivations de construction.

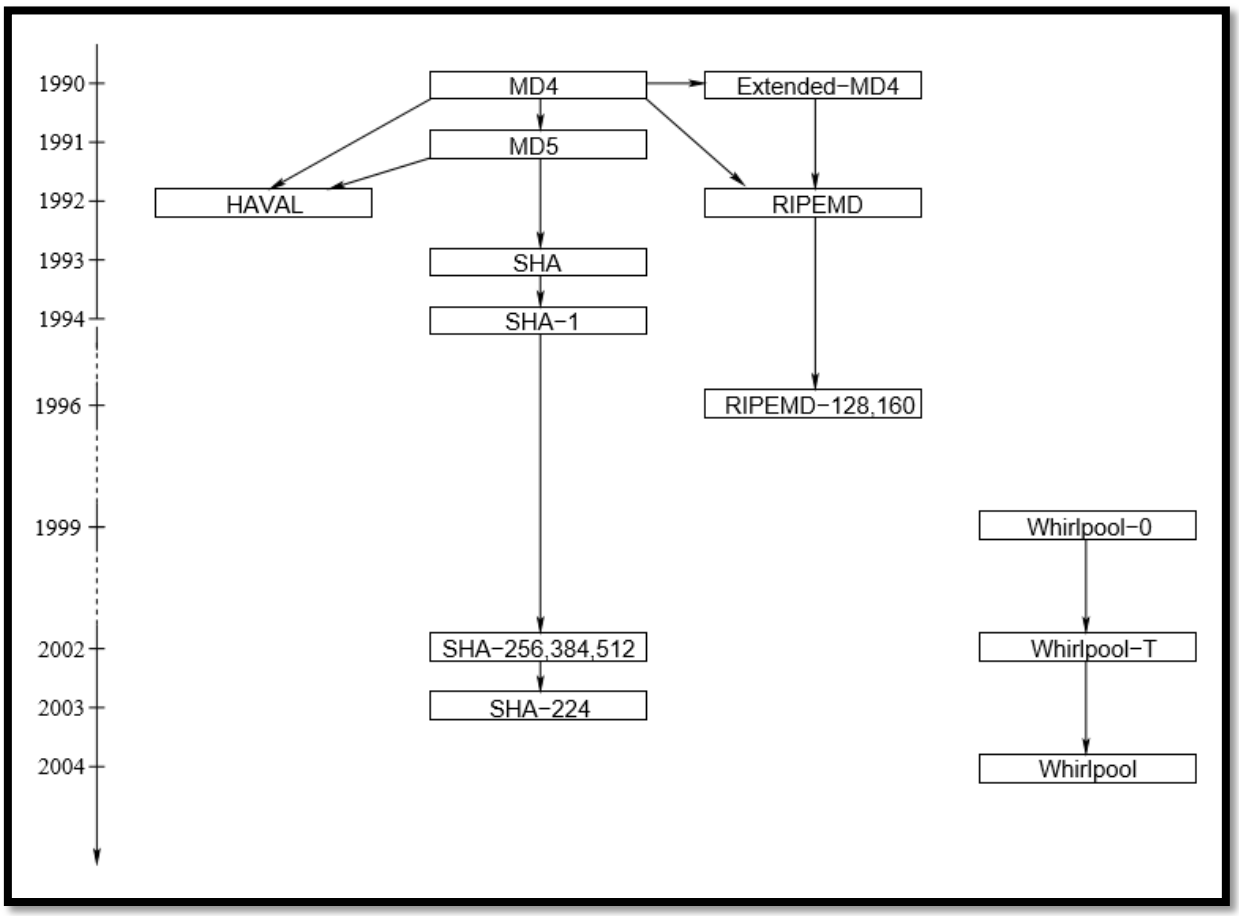


Figure I 2 *Historique des principales fonctions de hachage.*

3 Les attaques des fonctions hachage :

Il existe 2 types d'attaques spécifiques :

La collision : une collision se produit lorsque 2 fichiers (messages d'entrée) différents produisent le même condensat (empreinte). Dans ce cas il est alors possible de substituer un message d'entrée à un autre. L'algorithme de hachage SHA0 est vulnérable à la collision, c'est la raison pour laquelle il n'est plus utilisé aujourd'hui.

La pré-image : Il faut distinguer la pré-image et la pré-image secondaire. La première consiste à déterminer la valeur d'un fichier à partir de son condensat. La seconde à utiliser un condensat pour produire une valeur différente de celle à l'origine du hash**.[3]**

4 Présentation des fonctions de la famille SHA

Le gouvernement américain a normalisé au moins six algorithmes de hachage sécurisés. SHA-0 et SHA-1 ont été développées dans les années 1990, la série SHA-2 développéedans les années 2000 inclus SHA-224, -256, -384 et -512.

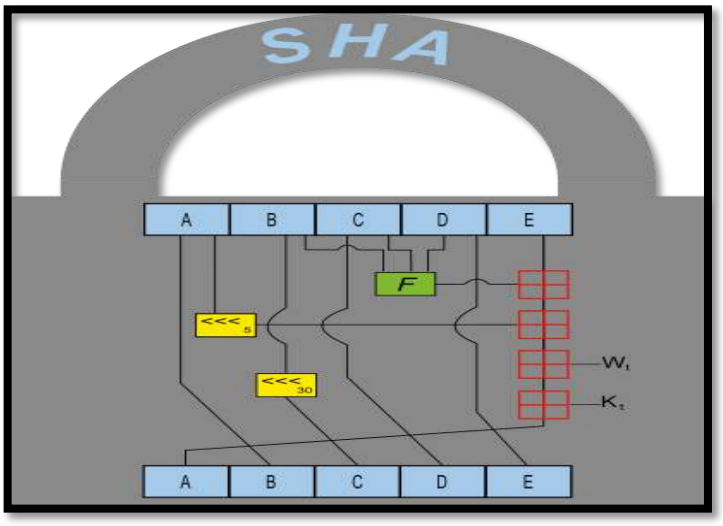


Figure I 3 *Fonctionnement de la famille SHA.*

4.1 *L'algorithme SHA-0*

D'abord publié en 1993 par le NIST, a été rapidement abandonnée à cause d'une faiblesse majeure de vulnérabilité de collision. Il a été remplacé par le SHA-1 en 1995, qui comprend une étape supplémentaire. Cet algorithmetransforme un message d'entrée jusqu'à 2^{64} -1 bits en un digest (haché) de 160bits.Cette fonctionmanipule des blocsd'entrée de 512 bits et des mots de 32 bits. **[3][4]**

4.2 *Sécurité de la fonction SHA-0 :*

Les faiblesses dans la conception de SHA-0 ont été confirmées en 1998 par la découverte par Antoine Joux et Florent Chabaud d'une attaque par collision, avec une complexité égale à 2^{61} appels à la fonction de compression contre 2^{80} (l'attaque générique des anniversaires).**[4]**

Certes, il n'existe pas actuellement une attaque connue pour la recherche de pré-image ou de seconde pré-image, mais l'utilisation de SHA-0 est tout à fait déconseillée. Elle est pourtant restée un domaine de recherche pour la communauté académique, en tant que prédécesseur de SHA-1.

4.3 L'algorithme SHA-1

Utilisé dans un certain nombre de protocoles d'Internet communs et des dispositifs de sécurité. Ceux-ci comprennent IPsec, PGP, SSL, S/MIME, TLS et SSH. Le secteur privé utilise cet algorithme de hachage pour sécuriser certaines informations sensibles. Il est prévu pour être retiré de la circulation générale par le gouvernement américain en 2010. **[4]**

4.4 *Sécurité de SHA-1 :*

La taille d'empreinte dans SHA-1 est de 160 bits. La recherche de collisions par force brute (attaque de Yuval) a donc une complexité en 2^{80} calculs d'empreintes. Récemment, une technique similaire à celle utilisée contre MD5 a permis d'obtenir des collisions en seulement d'une complexité de $O(2^{63})$ de calculs d'empreinte. On considère donc que SHA-1 est cassé, même si l'attaque reste difficile. De par sa popularité, cette fonction de hachage est encore largement utilisée.**[4][2]**

4.5 ***L'algorithmeSHA-2***

En 2002 le NIST a normalisé une nouvelle famille de fonctions de hachage, SHA-2. Quatre versions de cet algorithme permettent de calculer des empreintes de 224, 256, 384 et 512 bits. Comme ses prédécesseurs, SHA-2 utilise l'algorithme d'extension de domaine de Merkle-Damgård et la construction de Davies-Meyer. La taille des variables de chaînage est de 256 bits pour les deux premières versions et de 512 bits pour les deux dernières. Les blocs de message correspondants ont pour tailles respectives 512 et 1024 bits. De manière générale, SHA-224 et SHA-256 utilisent des registres de 32 bits, et SHA-384 et SHA-512 utilisent des registres de 64 bits. La structure générale des opérations effectuées est similaire entre ces quatre fonctions. Le bloc de message passe par une expansion permettant de générer 64 mots pour les fonctions SHA-224 et SHA-256 et 80 mots pour SHA-384 et SHA-512.**[4][5][6]**

4.6 *SHA-3*

En 2007, le NIST a lancé un concours visant à définir un nouveau standard pour les fonctions de hachage, SHA-3. Le besoin d'un nouveau standard fait suite aux nombreuses attaques essuyées pour la fonction SHA1, pour laquelle une collision peut être trouvée en $O(2^{63})$.Le standard actuel (SHA-2) étant dérivé de SHA-1 mais avec des tailles de hachés plus grandes, pourrait donc être victime du même type d'attaque.**[4][6]**

5 Description de la fonction de hachage SHA256

Les caractéristiques de la version SHA-256 qui nous intéresse par la suite dans notre travail, sont les suivantes :

Taille du message : 2^{64} bits maximum

Taille des blocs : 512 bits

Taille des mots : 32 bits

Taille du condensé : 256 bits

Niveau de sécurité attendu (collisions) : 2^{128} (par l'attaque des anniversaires)

Nombre de tours (fonction de compression) : 64. **[5]**

5.1 *Bourrage :*

Il s'agit ici de compléter le message d'entrée *M* pour qu'il soit d'une taille multiple de 512 bits. Le bourrage utilise la représentation binaire de l'entier *l* qui est la longueur en bits du message *M*. Pour ce faire on ajoute successivement :

Un bit à '1' à la fin du message *M*,

K bits à '0' de bourrage, où *k* est le plus petit entier naturel vérifiant

$l + 1 + k \equiv 448 \bmod 512$

Un bloc de 64 bits qui donne la représentation binaire de *l*.

Exemples :

Pour *M* = "abc", l = 3 × 8 = 24, $k \equiv 448 - (24 + 1) \equiv 423 \bmod 512$, donc *k*= 423 ;

On ajoute à *M*

D'abord un bit à '1',

Puis 423 bits à '0',

Puis 64 bits représentant 24 en binaire, soit « 0000 ... 0000 0011 000 »

On obtient alors un message complété dont la longueur est le plus petit multiple de 512 bits supérieur à la longueur initiale de *M*.

Pour *M* de longueur 500 bits, $k \equiv 448 - (500 + 1) \equiv -53 \bmod 512$; donc*k*= 512 - 53 = 459 ;

On ajoute à *M*

D'abord un bit à '1',

Puis 459 bits à '0',

Puis 64 bits représentant 500 en binaire, soit « 0000 ... 0000 0001 1111 0100 »

On obtient alors un message complété dont la longueur est supérieure de 512 bits au plus petit multiple de 512 bits supérieur à la longueur initiale de *M*.**[7]**

5.2 *Découpage en blocs*

Le message complété par l'étape précédente de bourrage est découpé en N blocs de 512 bits, notés $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$. Chaque bloc de 512 bits est ensuite découpé en 16 mots de 32 bits, notés $M_0^{(i)}, M_1^{(i)}, \ldots, M_{15}^{(i)}$.**[7] [8]**

5.3 *Initialisations*

Les huit variables suivantes sont affectées de valeurs initiales comme suit :

$H_0^{(0)}$=0x6a09e667 **(I.1)**

$H_1^{(0)}$= 0xbb67ae85 **(I.2)**

$H_2^{(0)}$= 0x3c6ef372 **(I.3)**

$H_3^{(0)}$= 0xa54ff53a **(I.4)**

$H_4^{(0)}$= 0x510e527f **(I.5)**

$H_5^{(0)}$= 0x9b05688c **(I.6)**

$H_6^{(0)}$= 0x1f83d9ab **(I.7)**

$H_7^{(0)}$= 0x5be0cd19 **(I.8)**

5.3.1 Calcul du condensé (haché)

Pour ce traitement on utilisera :

un tableau de 64 mots de 32 bits, notésW_0,W_1 ,…,W_{63}

huit variables, mots de 32 bits notées a,b,c,d,e,f,g,h

huit variables, mots de 32 bits contenant les valeurs de hachage, notées $H_0^{(i)}$et initialisées précédemment en $H_0^{(0)}$. Ces variables contiendront itérativement de nouvelles valeurs de hachage, $H^{(i)}$ et finalement contenir le condensé de M, dans $H^{(N)}$.

deux variables, notées T_1et T_2, mots de 32 bits.

six fonctions logiques qui traitent des mots de 32 bits. Le résultat de calcul de chaque fonction est un mot de 32 bits.

$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$ **(I.9)**

$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$ **(I.10)**

$\sum_0^{\{256\}} = \mathrm{ROTR}^{2}(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$ **(I.11)**

$\sum_1^{\{256\}} = ROTR^{6}(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$ **(I.12)**

$\sigma_0^{\{256\}} = \mathrm{ROTR}^{7}(x) \oplus ROTR^{18}(x) \oplus SHR^{3}(x)$ **(I.13)**

$\sigma_1^{\{256\}} = \mathrm{ROTR}^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$ **(I.14)**

Un tableau de 64 valeurs constantes, mots de 32 bits,$K_0^{\{256\}}$,$K_1^{\{256\}}$,…,$K_{63}^{\{256\}}$:

428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5

d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174

e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da

983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967

27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85

a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070

19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3

748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2

Le calcul du condensé (haché) se fait d'une manière itérative. On traite successivement les N blocs de M selon les étapes suivantes :

{

1. On remplit le tableau $W_t$pour t de 0 à 63, selon

$$W_t=\begin{cases} M_t^{(i)} , & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} , & 16 \le t \le 63 \end{cases} \quad \textbf{(I.15)}$$

2. On initialise les variables a,b, c, d, e, f, g et h avec les valeurs de hachage du tour précédent

a =$H_0^{(i-1)}$ **(I.16)**

b=$H_1^{(i-1)}$ **(I.17)**

c=$H_2^{(i-1)}$ **(I.18)**

d=$H_3^{(i-1)}$ **(I.19)**

e=$H_4^{(i-1)}$ **(I.20)**

f=$H_5^{(i-1)}$ **(I.21)**

g=$H_6^{(i-1)}$ **(I.22)**

h=$H_7^{(i-1)}$ **(I.23)**

3. Pour t de 0 à 63

{

T_1 = h+$\Sigma_1^{\{256\}}(e)$ +$Ch(e,f,g)$ +$K_t^{\{256\}}$+W_t **(I.18)**

T_2 =$\Sigma_0^{\{256\}}(a)$ +$Maj(a,b,c)$ **(I.24)**

$h = g$

$g = f$

$f = e$

$e = d + T_1$

$d = c$

$c = b$

$b = a$

$a = T_1 + T_2$

}

4. Calcul des valeurs de hachage intermédiaires

{

$H_0^{(i)} = a + H_0^{(i-1)}$ **(I.25)**

$H_1^{(i)} = b + H_1^{(i-1)}$ **(I.26)**

$H_2^{(i)} = c + H_2^{(i-1)}$ **(I.27)**

$$H_3^{(i)} = d + H_3^{(i-1)} \quad \textbf{(I.28)}$$

$$H_4^{(i)} = e + H_4^{(i-1)} \quad \textbf{(I.29)}$$

$$H_5^{(i)} = f + H_5^{(i-1)} \quad \textbf{(I.30)}$$

$$H_6^{(i)} = g + H_6^{(i-1)} \quad \textbf{(I.31)}$$

$$H_7^{(i)} = h + H_7^{(i-1)} \quad \textbf{(I.32)}$$

}

}

La fonction de mise à jour des registres peut etre schématisée comme le représente la figure

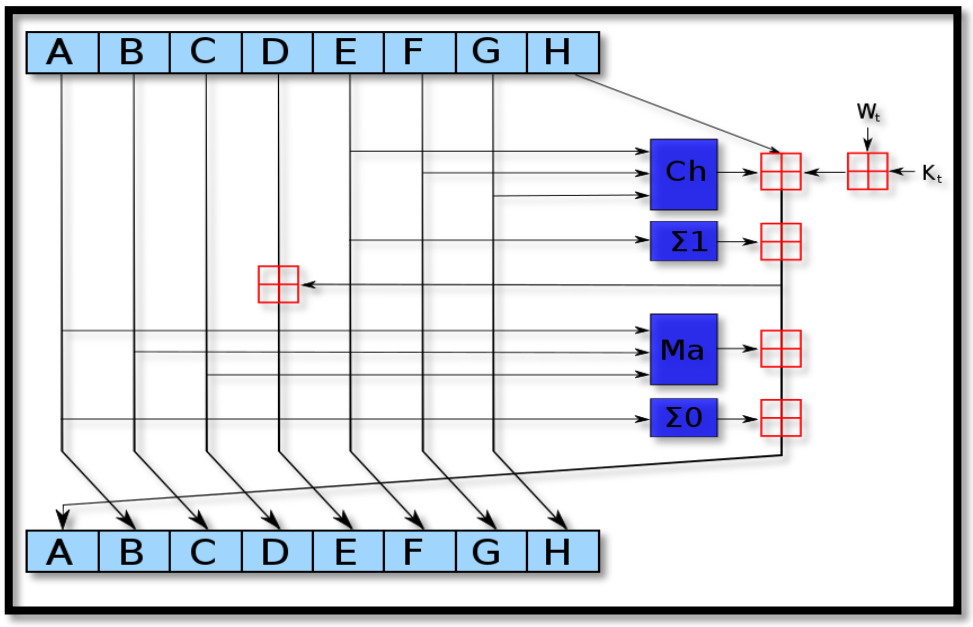


Figure I 4 *La fonction de hachage SHA-2*

Après répétition des quatre étapes ci-dessus pour les N blocs du message M, (i.e., après traitement de $M^{(N)}$), le condensé de 256 bits de M est obtenu par concaténation des valeurs

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$$

6 Domaines d'utilisation des fonctions de hachage cryptographiques :

Le domaine d'utilisation des fonctions de hachage cryptographique est multiple selon leurs propriétés spécifiques de sécurité.Nous citons dans cette partie des exemples de ces domaines.**[2][9]**

6.1 *Intégrité des données :*

Les fonctions de hachage cryptographiques sont des opérations mathématiques exécutées sur des données numériques ; en comparant le "hash" calculé (la sortie de l'exécution de l'algorithme) à une valeur de hachage connue et attendue, une personne peut déterminer l'intégrité des données. Par exemple, le calcul du hachage d'un fichier téléchargé et la comparaison du résultat à un résultat de hachage précédemment publié peuvent indiquer si le téléchargement a été modifié ou falsifié. Un aspect clé des fonctions de hachage cryptographiques est leur résistance aux collisions: personne ne devrait être capable de trouver deux valeurs d'entrée différentes qui donnent la même sortie de hachage.**[9]**

6.2 *Protection de mots de passe :*

Une autre des applications courantes des fonctions de hachage cryptographiques est la protection de mots de passe. Une grande partie des utilisateurs choisissent comme mots de passe des mots courants ou triviaux, n'assurant donc pas une entropie suffisante. Pour cela un mot de passe ne doit pas être stocké en clair sur une machine, seul le résultat du hachage (le haché) du mot de passe est donc stocké. L'ordinateur compare l'empreinte du mot de passe d'origine avec l'empreinte du mot de passe demandé pour identifier un utilisateur**.[8][9]**

6.3 *Signature électronique :*

Les schémas de signature électronique sont sans aucun doute l'application la plus importante des fonctions de hachage cryptographiques.

La signature électronique est un mécanisme analogue de la signature manuscrite permettant de garantir l'intégrité d'un document électronique et prouvant au lecteur du document l'identité de son auteur (authentification).Lorsqu'un utilisateur veut envoyer un message

signé, il calcule le haché du message, le chiffre avec sa clé privée, et le transmet avec le message. Le destinataire décrypte le condensat avec sa clé publique, calcule le haché du message reçu, et les compare. Si les deux hachés sont identiques, il s'assure de l'identité de l'émetteur et de l'intégrité du message**.[9]**

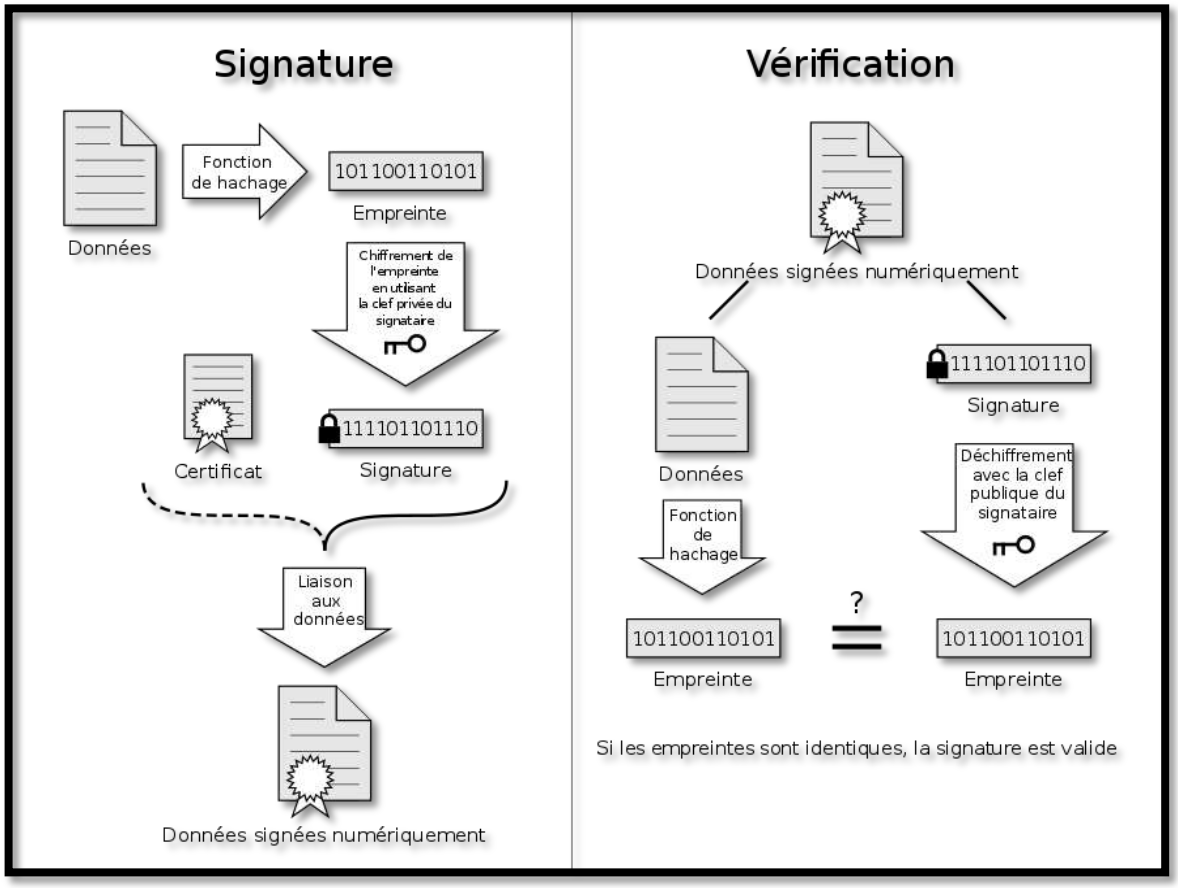


Figure I 5 *Principe de signatures électroniques.*

6.4 ***Dérivation de clé :***

Dans le cadre de la cryptographie symétrique, pour assurer une communication entre deux entités, plusieurs clés secrètes sont souvent utilisées. Une fonction de hachage peut générer plusieurs clefs différentes à partir d'une seule information secrète, appelée clef maître. L'utilisation des fonctions de hachage pour cette fonctionnalité a pour objectif d'empêcher un adversaire ayant obtenu une des clés dérivées, de déduire des informations sur la clé maître ou les autres clés dérivées. **[9]**

6.5 *Génération de nombres pseudo-aléatoires :*

Un générateur pseudo-aléatoire de nombres est un algorithme qui produit des suites de bits présentant certaines propriétés de l'aléatoire. On peut construire un générateur pseudo-aléatoire à partir d'une fonction de hachage.

Les générateurs pseudo-aléatoires sont nécessaires pour diverses applications cryptographiques, on les retrouve surtout pour la génération des clés pour le chiffrement asymétrique, la confidentialité des échanges sur les réseaux sans fil ou encore dans les signatures électroniques. **[9]**

7 Conclusion :

Dans ce chapitre, nous avons défini l'algorithme de hachage cryptographique, ainsi que ses propriétés : la résistance aux attaques recherchant des collisions, les pré-images et les secondes pré-images. Ensuite, nous avons présenté une description de la famille SHA et en particulier l'algorithme SHA-256 en détail. Enfin, nous avons terminé ce chapitre par les domaines d'utilisation.

1. Introduction

Un système embarqué peut être défini comme un système électronique et informatique autonome, qui est dédié à une tâche bien précise, possédant des ressources d'ordre spatial (taille) et énergétique (consommation) limitées. Le terme de « Système Embarqué » désigne aussi bien le matériel que le logiciel utilisé.

Les systèmes embarqués exécutent des tâches prédéfinies et ont un cahier des charges, qui peut être d'ordre :

- d'espace mémoire limité de l'ordre dequelques Mo maximum.Ils font de plus trèssouvent appel à l'informatique et aussiaux systèmes « tempsréel ».

- de consommation énergétique le plus faible possible, due à l'utilisation de sources autonomes, batteries, panneaux solaires.

- temporelle dont le temps d'exécution de tâches est déterminé.

2 Circuits logiques programmables

Les circuits logiques programmables (également appelés **PLD**) apparaissent à la fin des années 1970.Ces circuits permettent de réaliser une ou plusieurs Fonctions Logiques ou/et Fonctions Séquentielles faisant intervenir une ou plusieurs variables d'entrée. Ils sont utilisés pour remplacer l'association de plusieurs boîtiers logiques. Le câblage est simplifié, l'encombrement et le risque de pannes est réduit. Ces circuits disposent d'un certain nombre de broches d'entrées et de sorties. L'utilisateur associe ces broches aux équations logiques (plus ou moins complexes) qu'il programme dans le circuit. **[10][11]**



Figure II 1 *Les 4 familles des circuits PLD*

2.1 Circuits FPGA :

Les FPGA se sont imposés sur le marché des circuits logiques depuis la fin des années 1980. Alors qu'au début ils permettaient de remplacer quelques composantes discrètes de façon plus efficace que les PALs et PLAs, ils concurrencent de nos jours certains microprocesseurs et microcontrôleurs en complexité et en performance. Les FPGA (Field Programmable GateArrays ou "réseaux logiques programmables") sont des composants entièrement reconfigurables ce qui permet de les reprogrammer afin d'accélérer notablement certaines phases de calculs. L'avantage de ce genre de circuit est sa grande souplesse qui permet de les réutiliser à volonté dans des algorithmes différents en un temps très court.Le progrès de ces technologies permet de faire des composants toujours plus rapides et à plus haute intégration, ce qui permet de programmer des applications importantes. Cette technologie permet d'implanter un grand nombre d'applications et offre une solution d'implantation matérielle à faible coût.

Un FPGA est composé de:

- un réseau de blocs de logique programmable (Configurable Logic Block –CLB), chaque bloc pouvant réaliser des fonctions complexes de plusieurs variables, et comportant des éléments à mémoire;
- un réseau d'interconnexions programmables entre les blocs;
- des blocs spéciaux d'entrée et de sortie avec le monde extérieur (Input/Output Block –IOB).

Les FPGA sont bien distincts des autres familles de circuits programmables tout en offrant le plus haut niveau d'intégration logique. Il existe actuellement plusieurs fabricants de circuits FPGA et plusieurs technologies et principes organisationnels.

L'architecture, retenue par Xilinx, se présente sous forme de deux couches (Figure II.2):

• Une couche appelée circuit configurable.

• Une couche réseau mémoire SRAM.

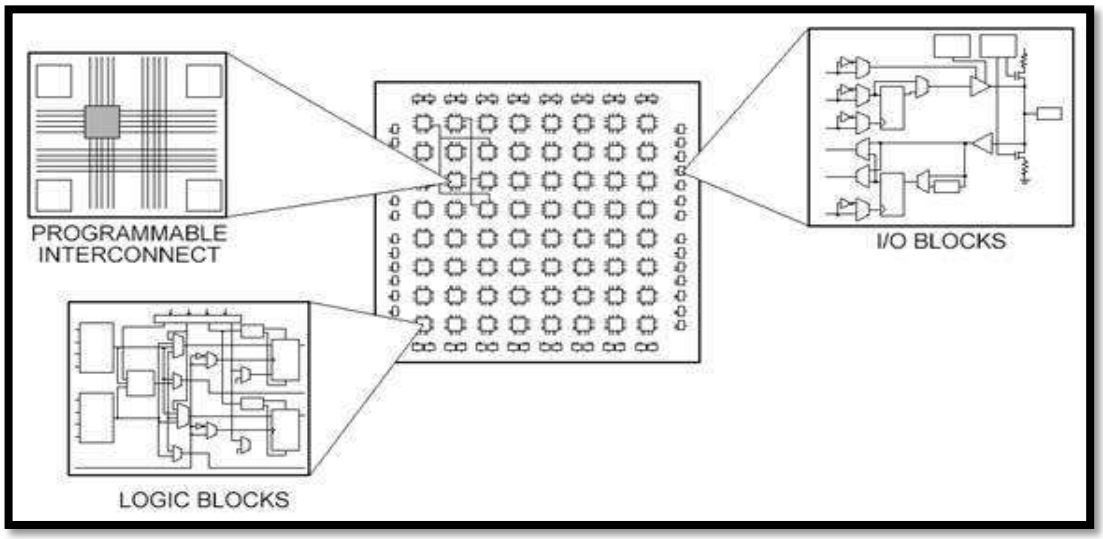


Figure II 2 *Architecture interne du FPGA fabriqué Xilinx*

Les outils de développement des FPGApermettent au concepteur de programmer le circuit à partir de la description de la fonction à réaliser. Cette description peut être textuelle, dans ce cas on utilise généralement des langages de développement : le Verilog et le VHDL. Cette description peut aussi être graphique et ce au moyen de chronogrammes, de graphes d'état et de symboles de fonction**.[12]**

3 Codesign: Conception conjointe logiciel/matériel

C'est une conception des systèmes embarqués qui intègrent à la fois des processeurs génériques capables d'exécuter des programmes (partie logicielle) et des IP (IntellectualProperty) (partie matérielle).

La conception conjointe logiciel/matériel est apparue dès le début des années 90. Dans un souci d'abaisser les coûts et d'augmenter la flexibilité d'un système spécifique, on peut faire le choix d'implanter certaines fonctions en logiciel et les fonctions les plus difficiles au niveau "performances" en matériel. Les méthodologies dites de hardware/software ou le codesign proposent dans la phase de spécification de partitionner le système, c'est à dire de décider quelles parties seront conçues en logiciel et quelles parties seront conçues en matériel. Une fois le système spécifié et partitionné, il est alors possible pendant la phase de conception de développer conjointement le matériel et le logiciel. Cette méthodologie nécessite lacoopération et la collaboration de deux équipes de développement (matériel et logiciel) afin de réduire les cycles de développements.

Les méthodologies de codesign présentent l'avantage de pouvoir effectuer des évaluations du système très tôt dans le flot de conception (dès la fin de la phase de spécification qui se matérialise par l'élaboration d'un prototype virtuel). Desétapes intermédiaires de vérification peuvent être effectuées au fur et à mesure de la progression de la conception par le biais de laco-simulation.

Le Codesign utilise le matériel et le logiciel pour une fonctionnalité à implanter :

- Le logiciel est utilisé pour sa flexibilité.
- Le matériel (i.e. FPGA et ASIC) est utilisé pour ses performances**.[4][5]**

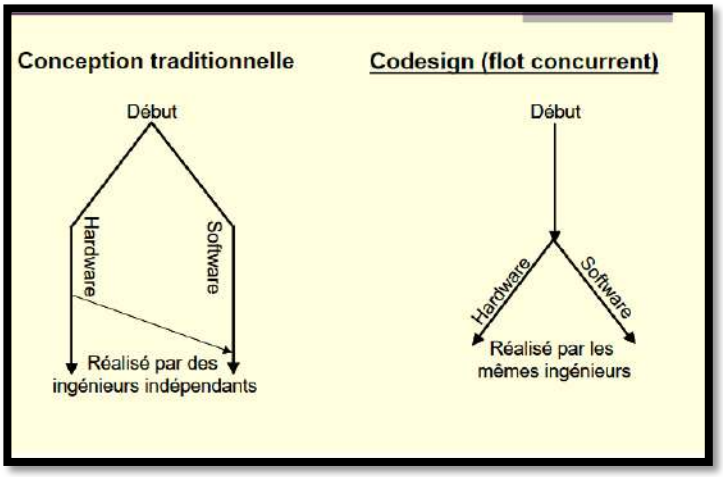


Figure II 3 *Conception traditionnelle et lecodesign*

3.1 *Flotde la conception conjointe logiciel/matériel :*

Le Codesign propose une approche structurée pour la conception d'une classe de systèmes numériques qui sont des systèmes dédiés.

Quatre grandes étapes sont distinguées dans l'approche du CoDesign :

1. *Spécification*
2. *Partitionnement*
3. *Synthèse logicielle & matérielle*
4. *Co-simulation* .**[13][14]**

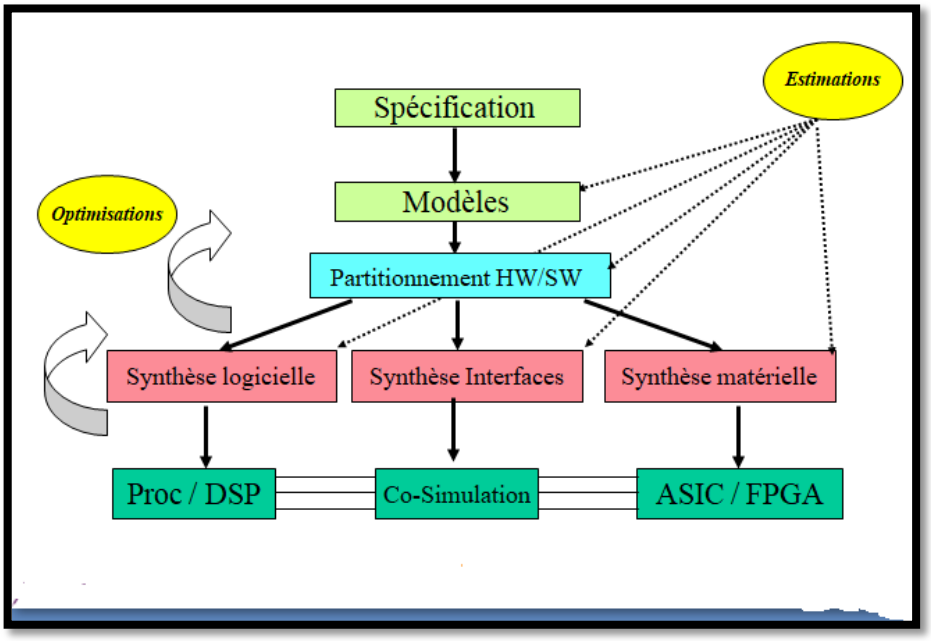


Figure II 4 *Flot de conception dans le codesign*

3.2 *Les composants virtuels (IPs) :*

L'évolution rapide de notre société conduit à l'élaboration d'équipements de plus en plus complexes, capables de traiter des flots d'informations de nature et d'origine très diverses et dont l'accès doit néanmoins rester simple et rapide. La conception et l'intégration de ce type d'équipements (systèmes) demande aujourd'hui encore des développements longs, fastidieux, coûteux et par conséquent en inadéquation avec les évolutions rapides et la flexibilité que doivent suivre les matériels électroniques. Ajouter à cela, le fait qu'il est

aujourd'hui très difficile de réutiliser la connaissance existante, l'industrie se trouve confrontée à un problème sérieux : le manque de concepteurs de systèmes expérimentés. Face à ce problème, les concepteurs essaient de favoriser la réutilisation de code, en s'orientant vers l'assemblage de blocs préconçus et pré-vérifiés désignés sous le terme de composants virtuels ou IP (pour intellectualproperty). Ce concept, né dans le milieu des années 90, a donné lieu à plusieurs vocables pour désigner ces blocs réutilisables : composants réutilisables, composants virtuels, IPs (IntellectualProperties), ou plus simplement macros.

Un composant virtuel est la spécification d'un composant réalisant une fonction bien déterminée pouvant être synthétisé, donc réutilisé, par un utilisateur n'ayant pas participé à la spécification de ce composant. Cette démarche comprend que seule une appréhension de la fonction réalisée est nécessaire, en aucune manière les détails de la spécification. Cependant cette absence de connaissance détaillée ne doit pas empêcher l'obtention par l'utilisateur de performances identiques après synthèse, à celles obtenues par le créateur de l'IP.

Un tel composant correspond aussi à un ensemble de services pour :

- l'évaluation de performances en vue de son intégration dans un système.
- les scripts de synthèse (logique, physique) qui garantissent à l'utilisateur de l'IP d'atteindre les mêmes performances physiques que celles obtenues par le fournisseur.
- les fichiers de test, à tous les niveaux d'abstraction, qui permettront de valider l'intégration de l'IP. **[15]**

Il existe trois classes (catégories) de composants virtuels matériels en fonction du niveau d'abstraction de leur description synthétisable.

1) **IP hardware :** La première est la catégorie hardware ou IP matériel, elle permet d'optimiser la puissance, la taille ou les performances et le plan pour une technologie spécifique. Elle englobe la netlist entière, le routage et l'optimisation pour une librairie technologique spécifique, un layout personnalisé ou une combinaison des deux. L'inconvénient est qu'il est moins flexible et portable car le processus est dépendant de la technologie (Plate-forme), par contre il présente l'avantage d'être prédictif. La protection est meilleure car on n'utilise pas le niveau RTL.
2) **IP firmware :** La seconde est la catégorie Firmware (ou flexible) : elle permet d'optimiser la structure et la topologie pour les performances et la superficie à travers

le placement des composants sur le plan et dans la topologie. Le cœur de type firm inclut une combinaison du RTLsynthétisable, des références de la bibliothèque technologique cible, une netlist de portes complète ou partielle. Le firmcore n'est pas routé, ce type de macro bloc est donc un compromis entre les IPs qui sont complètement logiciels et ceux complètement matériels. Les performances et la taille sont prévisibles mais la protection de la propriété intellectuelle peut ne pas être assurée.

3) **IP software :** Enfin pour la famille software (ou logiciel), le composant est livré sous sa forme HDLsynthétisable et son principal avantage est qu'il est simple d'utilisation, c'està dire flexible. L'inconvénient majeur est qu'il ne peut être prédictif en termes de superficie, puissance et temps. Egalement on diminue la protection de la propriété intellectuelle car le code RTL source est demandé par l'intégrateur.

4 Les systèmes sur puce (SoC) sur FPGA :

Un SoC est un ensemble de blocs fonctionnels intégrés dans un composant électronique avec au moins un processeur comme élément de traitement. L'idée des SoC consiste à ce que seuls les blocs de traitements particulièrement coûteux, de point de vue temps de calcul, soit réalisés en matériel(FigureII.5), l'essentiel de l'application étant implanté en logiciel et exécuté par les cœurs de processeurs. Ceci conduit à un autre problème de choix entre les blocs de traitement à réaliser en matériel et les blocs à réaliser en logiciels

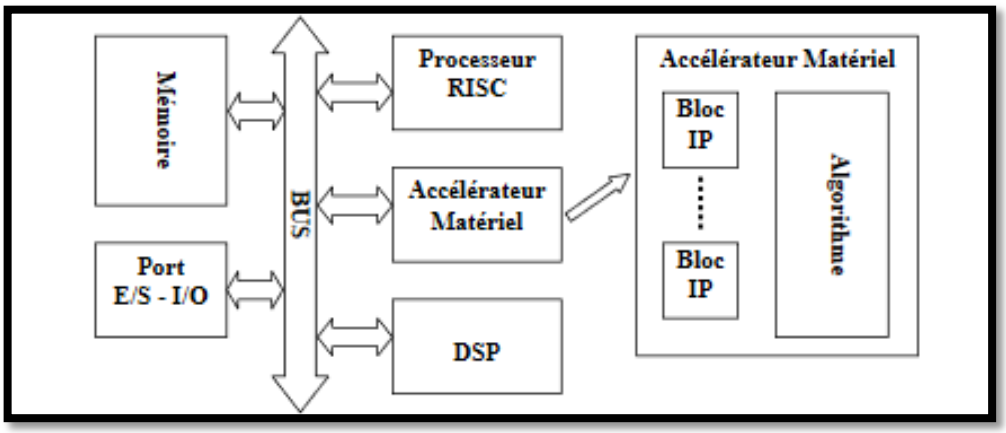


Figure II 5 *Architecture de base d'un système sur puce.*

L'approche SoC a été créée dans un premier temps pour le développement d'ASIC mais a été étendue pour le développement sur des circuits FPGA. On parle alors de SoPC.

La tendance pour les circuits FPGA est la possibilité de réaliser des systèmes sur puce en utilisant des " composants virtuels " et de concevoir ainsi des blocs de propriété intellectuelle IP qui sont par exemple des fonctions VHDL/Verilog génériques réutilisables. Un SoC est composé de plusieurs modules fonctionnels intégrés dans un même circuit.

Un SOC peut être développé :

- sans utilisation de cœur de processeur,
- autour d'un cœur de processeur
- autour d'une architecture multicore.

L'approche SoC a été empruntée du développement de systèmes complexes sur technologie ASIC. Cependant, elle a été étendue pour le développement sur cible FPGA. On parlera alors de SoPC (System On Programmable Chip).

Le code exécutable correspondant à l'application logicielle déployée sur le ou les processeurs peut être soit mémorisé dans une mémoire interne au FPGA, soit dans une mémoire externe.

Le concept de conception conjointe Matériel/Logiciel propose d'associer au sein d'un même système :

- un ou plusieurs cœurs de processeurs embarqués (softcore) ou matériel avec
- des blocs matériels dédiés: blocs IP personnalisés ou des blocs IP propriétaires.

Parmi les processeurs embarqués intégrables dans un système SoPC, on peut citer les solutions softcoreMicrBlaze et Nios respectivement développées par Xilinx et Altéra ainsi que d'autres processeurs performants (hardcore) issus du monde open source : processeurs Leon 2/3, plasma, ARM, Sun...

Les composants FPGA permettent d'implanter deux types de SoPC. Le premier type de SoPC utilise uniquement des IP soft de processeur Softcore tels que le MicroBlaze et le NIOS. Un processeur softcore est réalisé à partir de la logique programmable présente sur la puce. Le nombre de processeurs intégrés sur un même composant dépend des ressources du composant utilisé en logique programmable. Le deuxième type de FPGA en plus de la logique reconfigurable, implémente en dur un ou plusieurs cœurs de processeurs (PowerPC ou ARM). Ceci n'empêche en rien l'utilisation de processeurs Soft (netlist). Ces deux types de SoPC utilisent les mêmes flots de conception**.[16][17]**

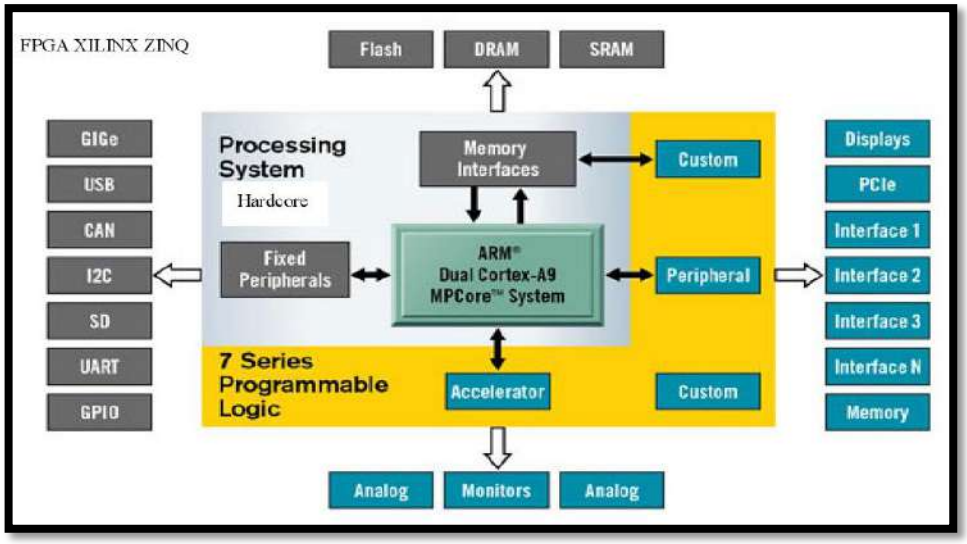


Figure II 6 *Architecture de base d'un SoPC sur FPGAde type XilinxZinq.*

4.1 Le processeur MicroBlaze de Xilinx

Le processeur MicroBlaze est paramétrable et permet l'activation sélective de fonctionnalités supplémentaires. Seule la version du processeur MicroBlaze et les versions suivantes supportent toutes les options. L'architecture interne du MicroBlaze associé au bloc IP propriétaire est comme le montre la figure II.7. Ce processeur de type softcore a pour avantage de permettre aux concepteurs de systèmes complexes d'intégrer facilement de la flexibilité dans leurs systèmes. Cette flexibilité est obtenue à l'aide des outils de programmation. Cependant, ce processeur flexible possède une puissance de calcul assez faible.

Spécialement pour la conception des SoC, Xilinx a proposé le MicroBlaze, un processeur RISC "soft IP" à 3 étages avec une architecture Harvard et avec 32 registres internes de 32 bits .Il dispose d'un bus d'instructions et de données internes et externes ((ILMB, DLMB, IOPB et DOPB). Le Processeur MicroBlaze, tout comme le NIOS, est très facilement configurable occupant selon le choix des options de 900 à 2600 éléments logiques et pouvant fonctionner sur une fourchette de fréquences autour de 80MHz. Le processeur comporte 70 options de configuration permettant à l'utilisateur de sélectionner ou de paramétrer les composants internes selon ses besoins. Parmi les options configurables, on peut citer :

- Un pipeline à 3 ou 5 niveaux,

- Utilisation des multiplieurs câblés du FPGA,
- Opérateur de division,
- Opérateur de décalage (Barrel Shifter),
- FPU (Floating Point Unit),
- Mémoires cache instructions et données,
- Logique de débug (hardware breakpoints).**[18]**

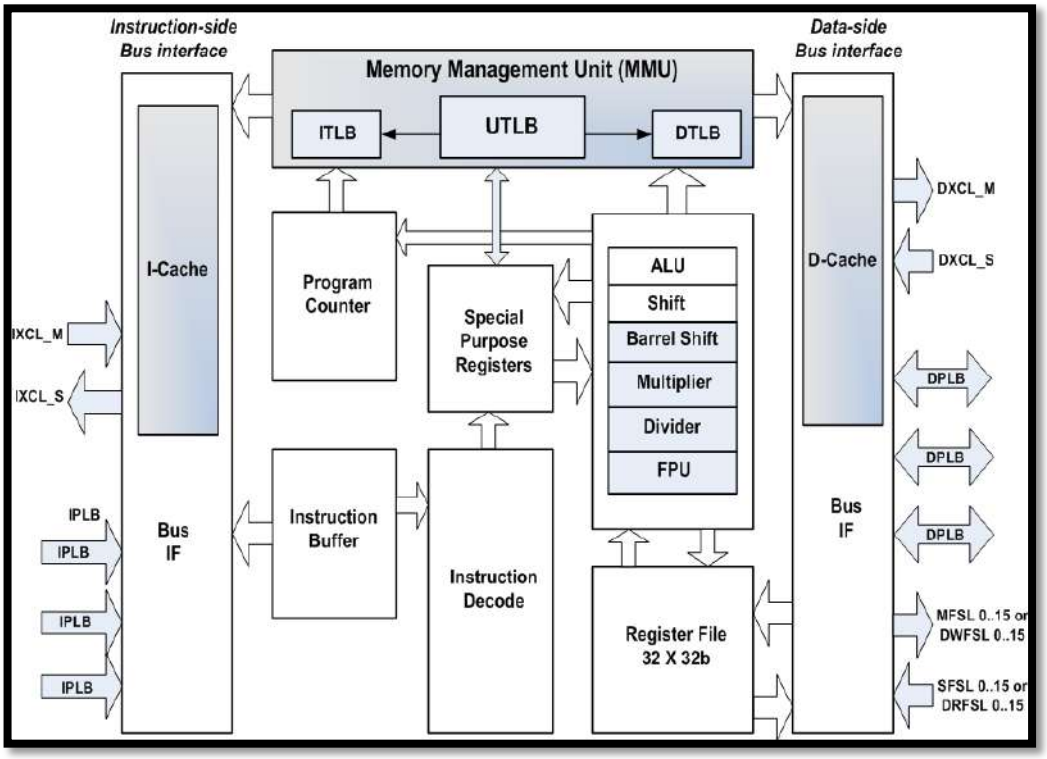


Figure II 7 *Architecture interne du softcoreMICROBLAZE*

4.2 Communications sur puce

Le bus utilisé avec le MicroBlaze est le bus OPB (On-chip Peripheral Bus). C'est un bus IBM Core-Connect utilisé aussi avec les processeurs PowerPC. Ce bus autorise un maximum de 16 maîtres et un nombre d'esclaves illimité. Il dispose d'une politique d'arbitrage (Bus multi-maîtres) paramétrable. Dans un système sur puce programmable à base de processeur MicroBlaze, le bus OPB est utilisé afin de connecter les périphériques dont les besoins en communication sont faibles. Autrement, Xilinx fournit le lien FSL (Fast Simplex Link)

permettant des accès rapides (2 fronts d'horloge) des périphériques vers le MicroBlaze et vice versa (8 connexions FSL par MicroBlaze).

Un autre type de bus LMB (Local Memory Bus) est utilisé pour accéder aux blocs RAM du FPGA. Dans le MicroBlaze, ce bus est utilisé pour les instructions et les données et il assure des accès rapides à la mémoire (1 front d'horloge).

En plus, de nombreux périphériques sont fournis avec le MicroBlaze, afin de constituer un microcontrôleur complet et personnalisable. Il y a, entre autres : contrôleur mémoire (SRAM, Flash), contrôleur mémoire SDRAM, UART lite, Timer/compteur avec fonction PWM, interface SPI, contrôleur d'interruptions, GPIO (entrées-sorties génériques), convertisseurs A/N et N/A Delta-Sigma, DMA, etc. Ces différents périphériques sont reliés par le bus PLB.**[18][19]**

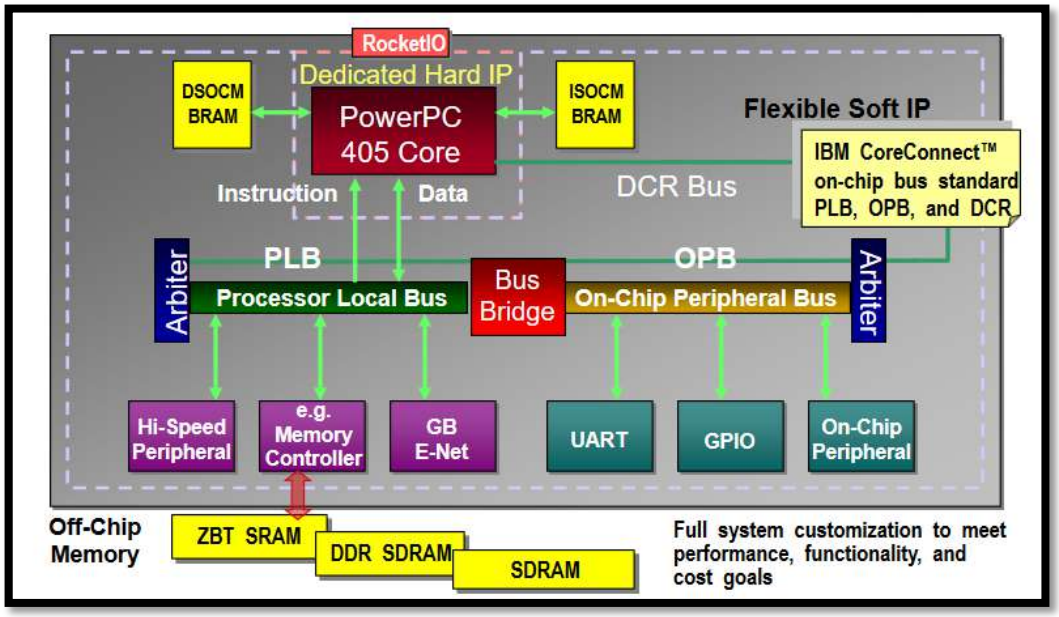


Figure II 8 *Communications sur puce*

4.2.1 ***Processor Local Bus (PLB)***

Le cœur du Power PC ™ 405 accède à des ressources système haute vitesse et hautes performances via des interfaces PLB (Processor Local Bus) sur les contrôleurs de cache d'instructions et de données. Les interfaces PLB fournissent des bus d'adresses 32 bits et de données 64 bits séparés pour les côtés instruction et données.

Le PLB prend en charge la lecture et l'écriture des transferts de données entre les dispositifs maître et esclave équipés d'une interface de bus PLB et connectés via des signaux PLB.

L'architecture de bus prend en charge plusieurs périphériques maître et esclave. Chaque maître PLB est relié au PLB via des bus d'adresse, de lecture et d'écriture séparés. Les esclaves PLB sont reliés au PLB par l'intermédiaire de bus partagés, mais découplés, d'adresses, de données de lecture et de données d'écriture et d'une pluralité de signaux de commande et d'état de transfert pour chaque bus de données.

L'accès à la PLB est accordé au moyen d'un mécanisme d'arbitrage central qui permet aux capitaines de se disputer la propriété des bus. Ce mécanisme d'arbitrage est suffisamment souple pour permettre la mise en œuvre de divers schémas de priorités. De plus, un mécanisme de verrouillage d'arbitrage est prévu pour prendre en charge les opérations atomiques commandées par le maître. Les arbitres PLB peuvent être implémentés sur le fabricFPGA et sont disponibles en tant que cœurs IP souples. Le PLB est un bus entièrement synchrone. La synchronisation de tous les signaux PLB est assurée par une seule source d'horloge partagée par tous les maîtres et esclaves connectés à la PLB.**[20][21]**

4.2.2 *On-chip Peripheral Bus (OPB)*

Le cœur du Power PC ™ 405 accède aux ressources système à faible vitesse et faible performance via le bus périphérique sur puce (OPB). L'OPB est un bus entièrement synchrone qui fonctionne indépendamment à un niveau distinct de la hiérarchie de bus. Il n'est pas prévu de se connecter directement au cœur du processeur. Les interfaces OPB fournissent une adresse séparée de 32 bits et des bus de données de 32 bits. Comme l'OPB prend en charge plusieurs périphériques maîtres, le bus d'adresses et le bus de données sont implémentés en tant que multiplexeur distribué. Le cœur du processeur peut accéder aux périphériques esclaves sur ce bus via l'unité de pont "PLB to OPB". Les périphériques qui sont des maîtres de bus OPB peuvent accéder à la mémoire sur le PLB via l'unité de pont "OPB à PLB". Les arbitres OPB peuvent être implémentés sur un fabricFPGA et sont disponibles en tant que cœurs IP souples**.[21]**

4.2.3 *Bus de contrôle de périphérique (DCR)*

Le bus DCR (Device Control Register) est conçu pour transférer des données entre les registres à usage général (GPR) de la CPU et les registres de contrôle de l'équipement (DCR) de la logique esclave DCR. Le bus DCR supprime les registres de configuration de la carte d'adresses mémoire, ce qui réduit le chargement et améliore la bande passante de la PLB. Le

bus DCR entièrement synchrone fournit un bus d'adresses de 10 bits et un bus de données de 32 bits. Le bus DCR est typiquement implémenté comme un multiplex distribué à travers la puce. L'implémentation est telle que chaque sous-unité a non seulement un chemin pour placer ses propres DCR sur le chemin de lecture DCR de la CPU, mais aussi un chemin qui contourne ses DCR et place les DCR d'une autre unité sur le chemin de lecture DCR de la CPU. Tous les DCR sont privilégiés pour la lecture et l'écriture**.[21]**

5 Les Outils de développement de Xilinx

Xilinx fournit dans son pack d'outils différents soft permettant la création de systèmes embarqués sur puce, parmi ces softs on dénombre ISE (Integrated Software Environnent) et EDK. L'outil ISE est spécialement pour la réalisation des projets d'IPs matériels à partir d'un code HDL. L'outil EDK, englobant System Generator for Processor et Xilinx Platform Studio (XPS), qui permettent d'établir un lien direct entre la partie matérielle et logicielle d'un système. Tous les deux offrent la possibilité d'avoir un bitstream pour la programmation des FPGA suivant l'application ciblée. Et dans l'objectif de tester l'environnement matériel par logiciel, Xilinx fournit l'outil SDK (Software Development Kit). Cet outil permet aux développeurs de faire tourner des applications logicielles écrites en langage C en générant des exécutables au format ELF (Executable and Linking Format) qui vont être chargés sur la carte par l'interface JTAG**.[22]**

6 Conclusion

Dans ce chapitre, nous avons exposé en premier lieu les circuits logiques programmables ainsi que leurs familles et plus précisément les circuits FPGA. Ensuite, nous avons décri la méthodologie de conception conjointe logiciel/matériel. En outre, nous avons présenté les systèmes sur puce (SoC) et leur développement sur des circuits FPGA, ainsi que les différents types d'interconnexions internes entre les différents blocs du SoC. Enfin, les outils de développement de xilinx (ISE et EDK) ont été présentés.

1 Introduction

Notre objectif principal est de réaliser et évaluer par des tests pratiques un IP core pour l'algorithme de hachage SHA256, intégré dans un système embarqué à base d'un softcore de type Microblaze. Dans ce contexte, nous avons opté pour l'algorithme de hachage un code VHDL open source, optimisé du site opencores.org**[23].**

Pour cela, nous présentons d'abord une description générale de l'architecture du module globale SHA256 ainsi que des résultats de simulations. Ensuite, nous exposons notre conception conjointe logiciel/matériel (conception hybride) et son implémentation sur une plateforme de développement de type DigilentNexys 2-500C.

Enfin, nous exposons les résultats de deux tests pratiques réalisés entre la carte FPGA et un PC interfacés entre eux par le port série (RS232).

2 Description et fonctionnement du module globale SHA256 utilisé

Dans cette partie, il sera présenté une description générale de l'architecture du module globale SHA256 utilisé dans ce projet.

La figure III.1 montre son schéma bloc. Le tableau III.1 décrit les différents signaux d'entrées/sorties.

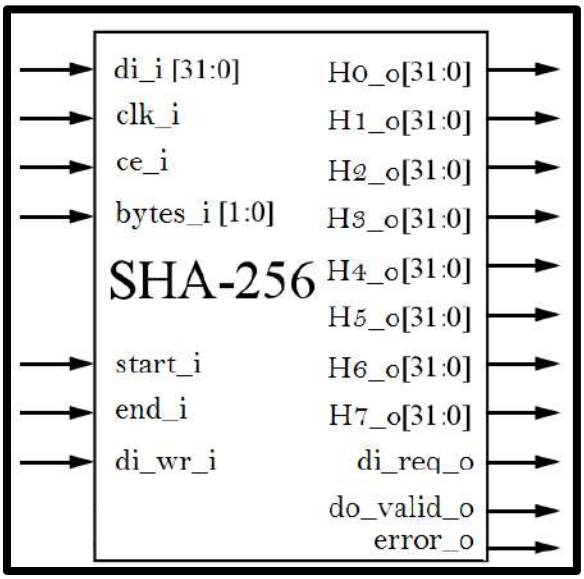


Figure III.1*Schéma bloc du module SHA256 utilisé.*

Tableau III.1 : *description des signaux d'Entrées/Sorties du module SHA256*

Nom	**Type**	**Description**
di_i [31:0]	Entrée	Donnéed'entrée (un mot de 32 bits du message)
clk_i	Entrée	Signal d'horloge
ce_i	Esntrée	Signal de validation synchrone
bytes_i	Entrée	Nombre d'octets valides dans le dernier mot du message 01 : di_i [31:24] valide ; 10 : di_i [31:16] valide 11 : di_i [31:8] valide ; 00 : di_i [31:0] valide
start_i	Entrée	Signal d'initialisation au début de chaque message
end_i	Entrée	Signal qui indique le dernier mot du message
di_wr_i	Entrée	Signal qui indique la validité de du signal d'entrée (di_i)
H0_o - H7_o [31:0]	Sorties	Valeur du message haché
di_req_o	Sortie	Signal requête données
do_valid_o	Sortie	Signal qui indique la validité du message haché
error_o	Sortie	Signal qui indique une erreur

Le signal « start _i» est mis en niveau haut au début de chaque message pour initialiser le circuit de hachage SHA256. Ce dernier active le signal « di_req_o »pour indiquer qu'il est prêt à accepter des données. Chaque mot de 32 bits est cadencé à l'entrée « di_i » sur le front montant du signal d'horloge « clk_i » lorsque le signal « di_wr_i » est activé. Ce signal « di_wr_i » est utilisé comme un accusé de réception d'une demande de données provenant du circuit SHA256. Si le circuit demande encore une nouvelle donnée (« di_req_o » niveau haut) et le signal « di_wr_i » est en niveau bas, le circuit s'arrête.

La principale différence entre les signaux « ce_i » et « di_wr_i » est que le deuxième signal ne bloque pas le circuit lorsqu'il est en attente des données, alors que le premier suspend toutes les opérations à l'intérieur du circuit.

Après l'arrivée de 16 mots sur l'entrée « di_i », le signal « di_req_o » est désactivé. Le circuit SHA-256 calcul le haché du message reçu (16 mots de 32 bits).La figure 2 montre le premier bloc de seize mots synchronisés dans le circuit.

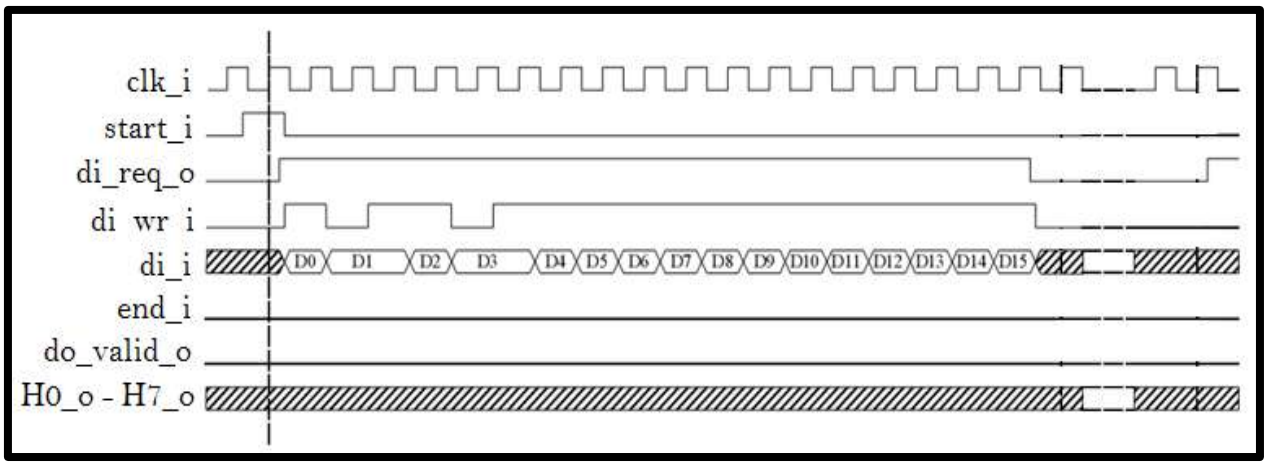


Figure III.2*Chronogramme des signaux d'E/S dans le cas du premier bloc du message.*

La norme spécifie que le nombre maximal de bits dans le message complet est 2^{64}-1. Par conséquent, le nombre maximal de mots de 32 bits pouvant être synchronisés est 2^{59}-1. Le circuit utilisé peut gérer n'importe quel nombre de mots jusqu'à 2^{59}-1.

La figure 3 montre l'arrivée du dernier bloc du message. Le signal « end_i » est activé par l'utilisateur lors de l'envoie du dernier mot.

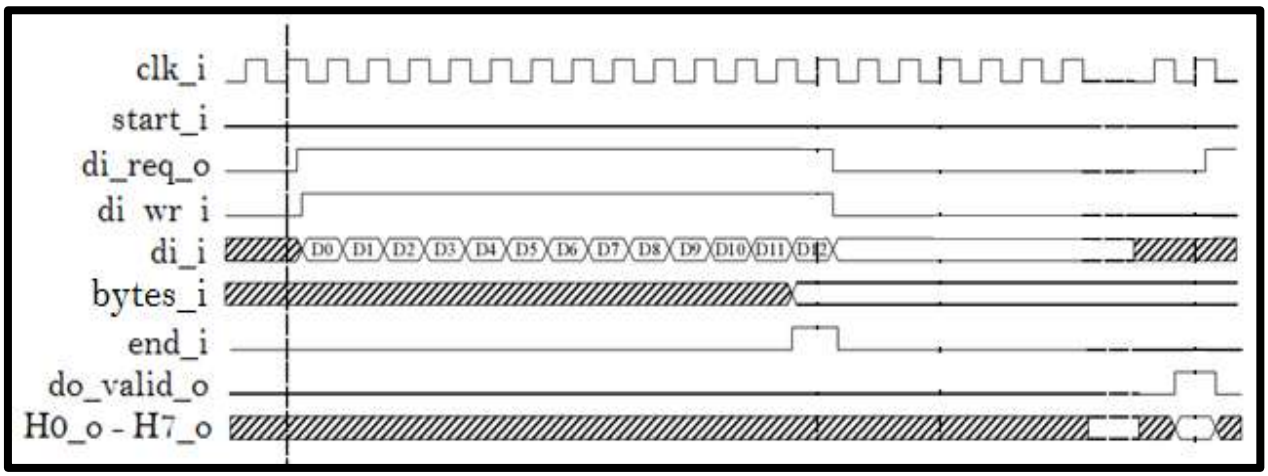


Figure III.3*Chronogramme des signaux d'E/S dans le cas du dernier bloc du message.*

On peut noter que le signal « bytes_i » est utilisé par le circuit SHA256 lorsque le signal « end_i » est haut. Ce signal est utilisé pour déterminer combien d'octets dans le dernier mot font partie des données d'entrée, c'est-à-dire, si le message d'entrée n'est pas un multiple de mots de32

bits, l'utilisateur doit ajouter des octets contenants des zéros au dernier mot en indiquant le nombre d'octets valides au circuit SHA256 par le biais du signal « bytes_i » (voir tableau III.1 pour voir comment le circuit interprète ce signal).

Si le nombre total de mots d'entrée plus trois n'est pas un multiple de 16, des octets de bourrage (pading) supplémentaires sont ajoutés par le circuit pour calculer le résumé (haché) du message comme il est spécifié dans la norme (voir chapitre 1).Les deux mots de longueur du message original sont également ajoutés par le circuit.

Le résumé de message de 256 bits est émis sur H0-H7 lorsque le signal « do_valid_o » est activé. Ce signal indique que le calcul du résumé (haché) est terminé et reste active jusqu'à l'arrivée d'un nouveau « start_i ».

La figure III.4 représente le diagramme bloc du module SHA256.

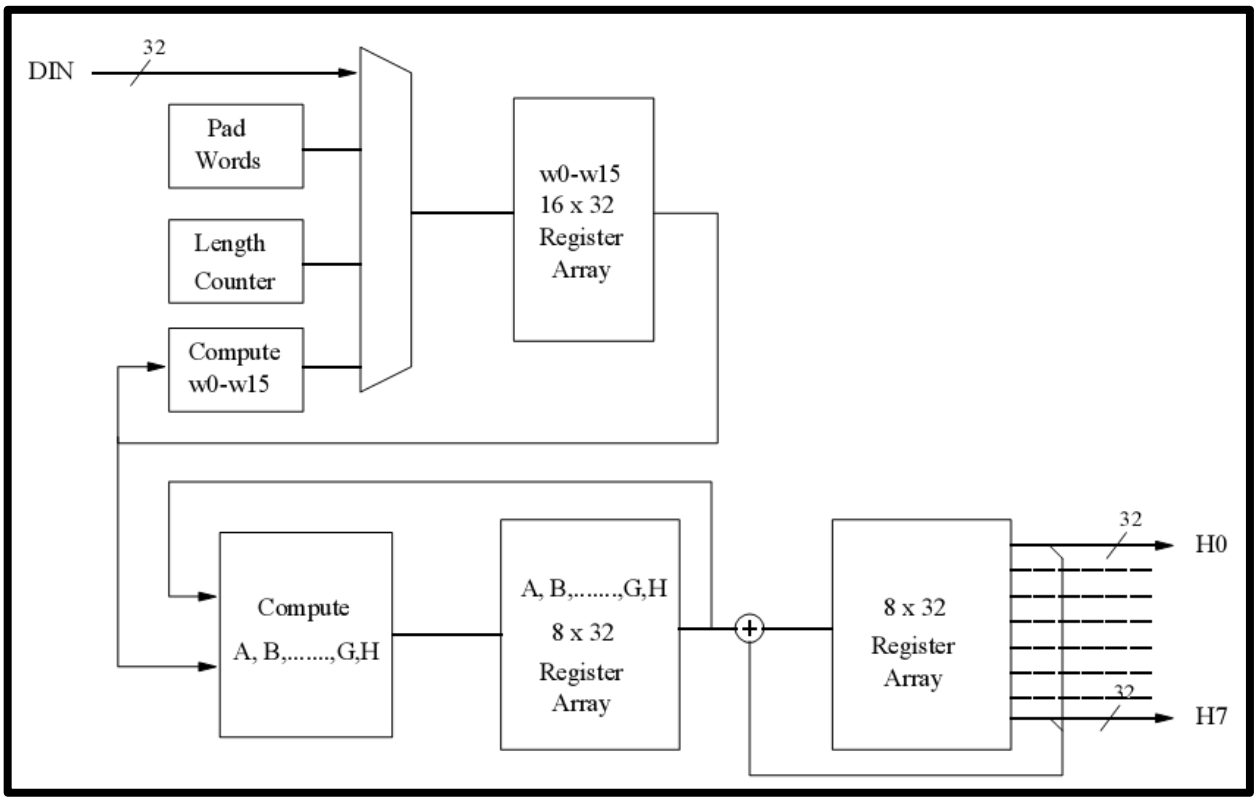


Figure III.4 Diagramme bloc du module SHA256 utilisé

Le bloc « Pad Words) effectue l'opération de bourrage. Le bloc « compute w0-w15 » est conçusuivant l'architecture représenté dans la figure III-5.

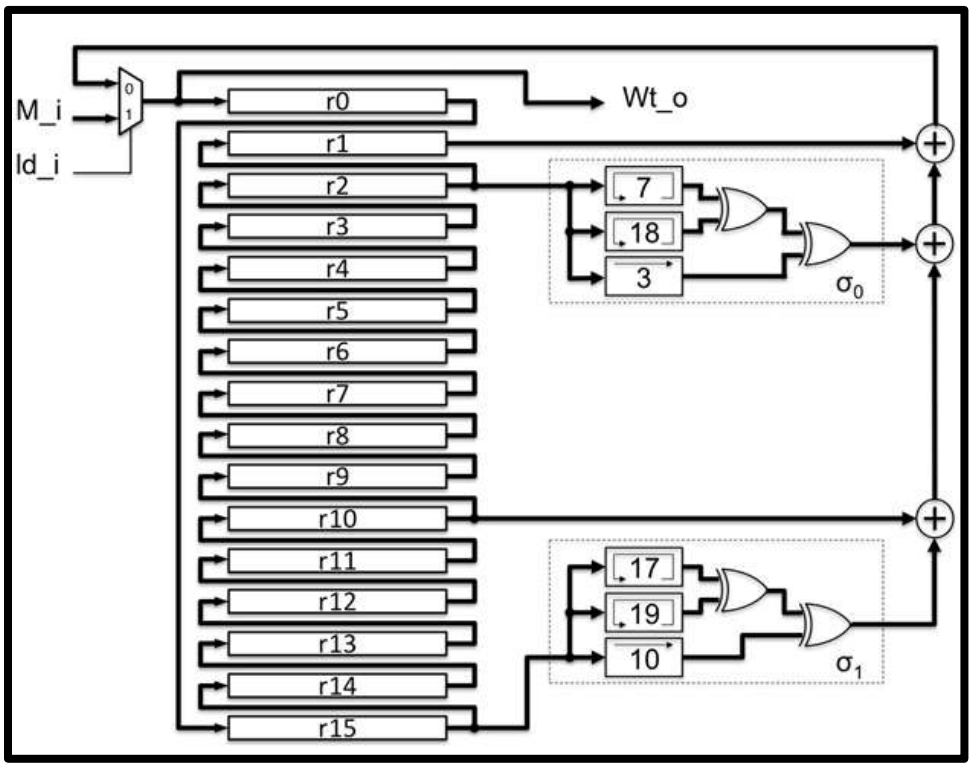


Figure III.5*Architecture de calcul des mots W0-W15.*

Le bloc « computeA,B, …G,H » est conçu suivant l'architecture représenté dans la figure III-6.

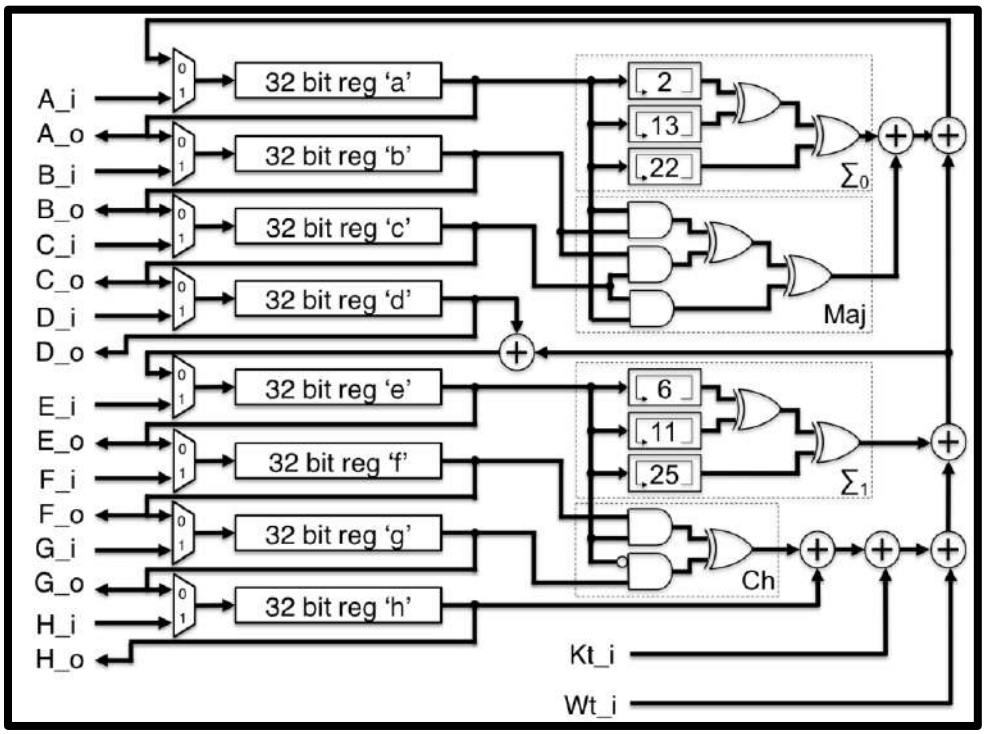


Figure III.6*Architecture de calcul des mots W0-W15.*

3 Simulation du module de hachage SHA256

Pour valider le bon fonctionnement de la fonction de hachage implémentée, nous avons effectué trois tests de simulations avec le logiciel ModelSim.

3.1 Résultat du 1er test

Nous avons effectué un premier test dont le message d'origine est la chaîne de caractères « abc ». Ce test est le premier exemple qui se trouve dans le document officiel Fips-180-2**[24].**Donc nous avons chargé le signal d'entrée par la valeur : 0x61626300 (code ASCII de la chaîne de caractères « abc » plus l'octet « 00 »pour compléter le mot à 32 bits).

Ce message contient à l'origine 3 octets, donc nous avons fixé le signal bytes_i à la valeur « 11 ». L'horloge est fixée à 50 MHz. Les autres bits de commande sont fixés comme nous l'avons expliqué auparavant.

Le résultat de hachageest identique à celui cité dans le document officiel qui est :0xBA7816BF 8F01CFEA 414140DE 5DAE2223 B00361A3 96177A9C B410FF61 F20015AD (Figure III.7).

En plus, on peut noter aussi que le temps consommé par l'opération de hachage pour cet exemple est de 65 cycles d'horloge, ou l'équivalent de 1300 ns (65*20 ns).

3.2Résultat du 2ème test

Nous avons effectué un deuxième test avec un message d'entrée est égal à 0xBD. Donc nous avons fixé le signal d'entrée di_i à 0xBD000000, et le signal bytes_i à « 01 ». Le résultat de hachageest correcte et il est de :0x68325720 aabd7c82 f30f554b 313d0570 c95accbb 7dc4b5aa e11204c0 8ffe732b (Figure III.8).

3.3 Résultat du 3ème test

Dans le troisième test, le message d'origine est le mot 0xC98C8E55.Donc nous avons fixé le signal signalbytes_i à « 00 ». Le résultat de hachage est correcte et il est de : 0x7abc22c0 ae5af26c e93dbb94 433a0e0b 2e119d01 4f8e7f65 bd56c61c cccd9504 (Figure III.9).

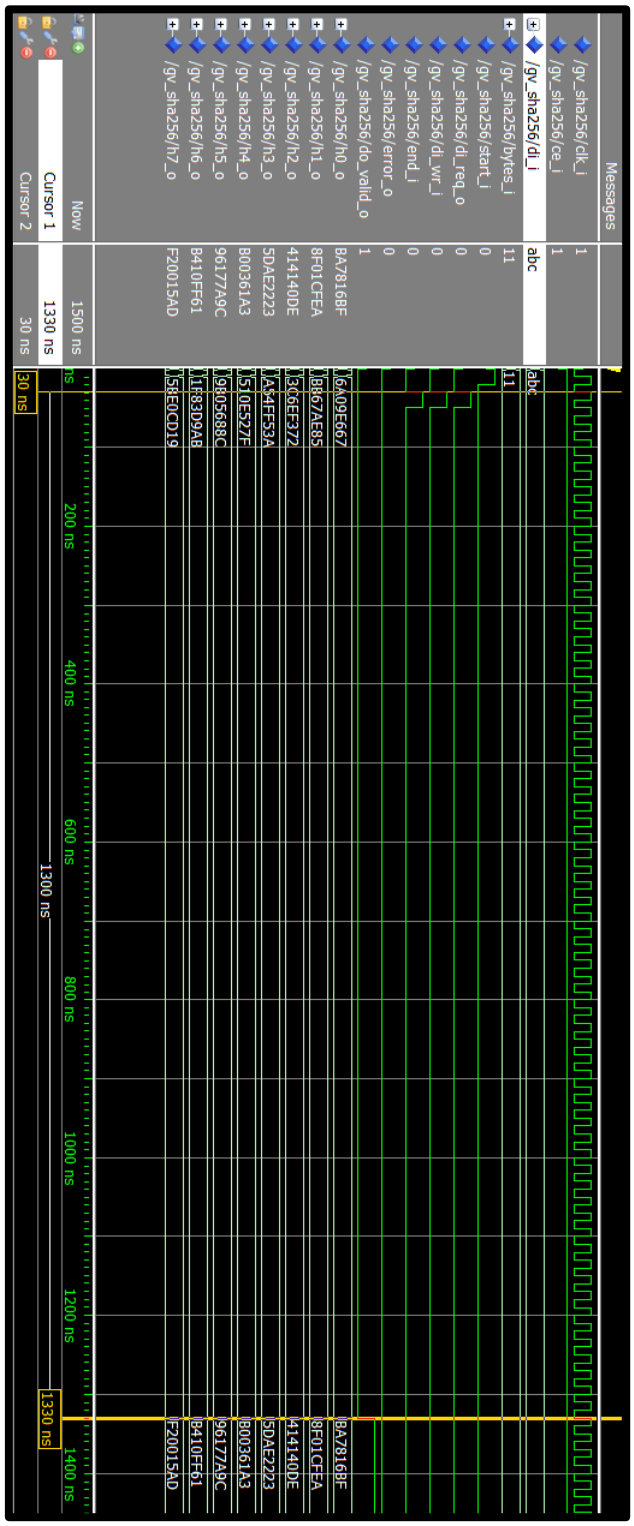


Figure III.7 Résultat de simulation du 1er test.

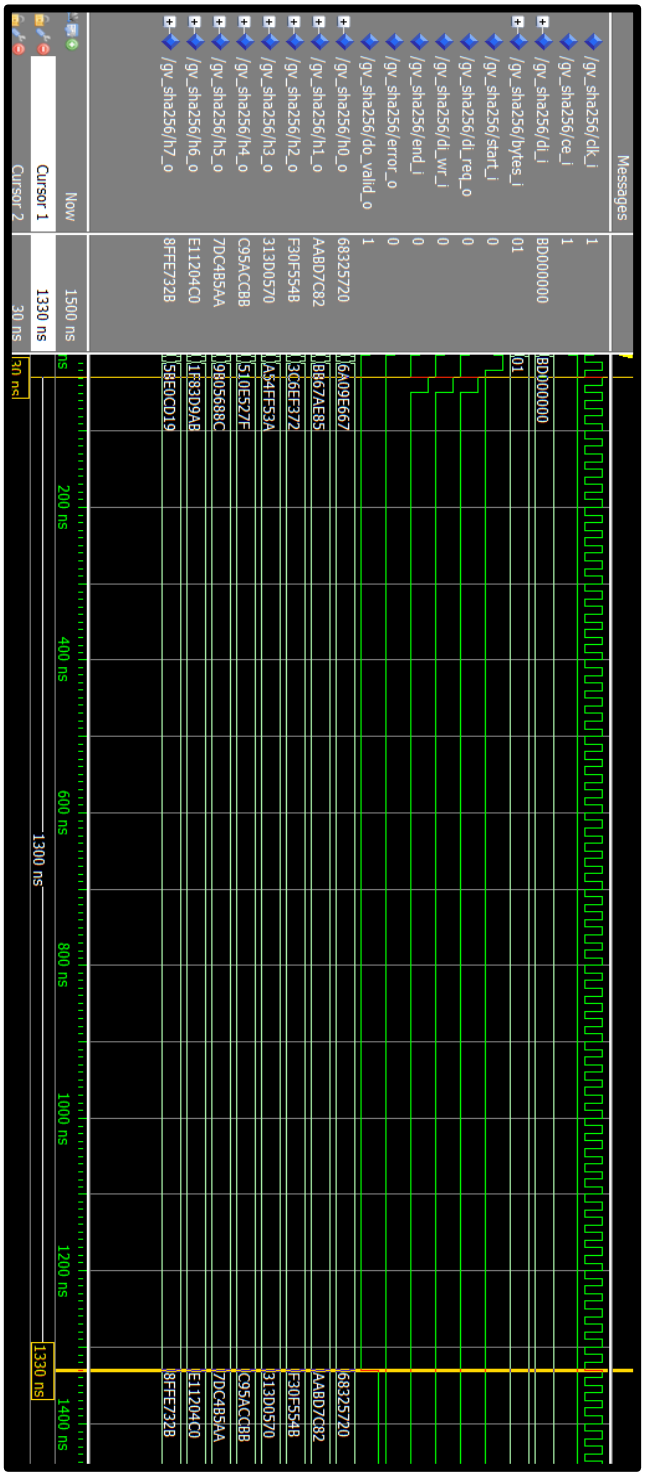


Figure III.8Résultat de simulation du 2ème test.

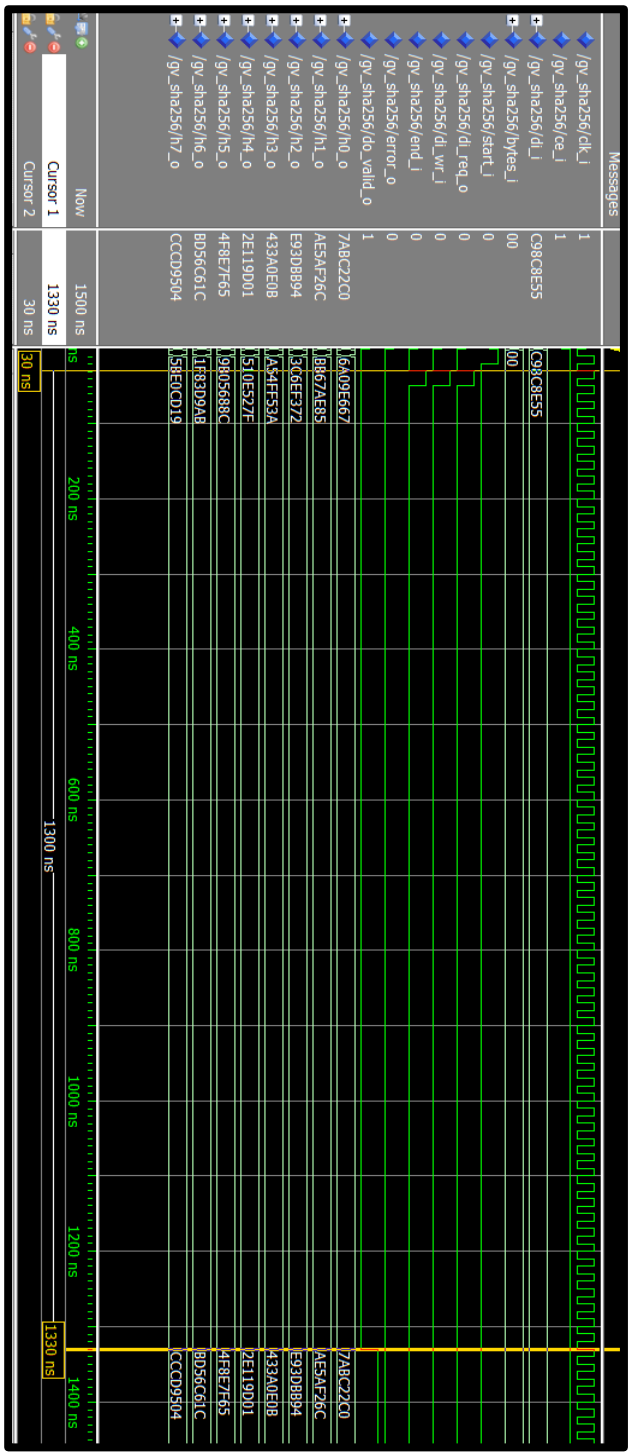


Figure III.9 *Résultat de simulation du 3ème test.*

4 Conception conjointe logiciel/matériel d'un IP core SHA256

Dans cette partie, on voudrait transformer le circuit présenté précédemment en un IP core par une conception conjointe logiciel/matériel.

Avant de créer l'IP core, on a créé un nouveau projet en utilisant l'environnement XPS composé d'un système basé sur un softcore de type Microblaze à 50 MHz, une mémoire de données de 16kB, une mémoire d'instructions de 16kB et un UART pour la communication avec le PC.

4.1 Etape N°1 : Création del'IPcore

Dans cette partie, on présente les étapes de création de l'IP coredans le but de l'intégrer avec le système embarqué à base du Microblaze. On commence par l'exécution de l'environnement XPS (Xilinx Platform Studio) (figureIII.10).

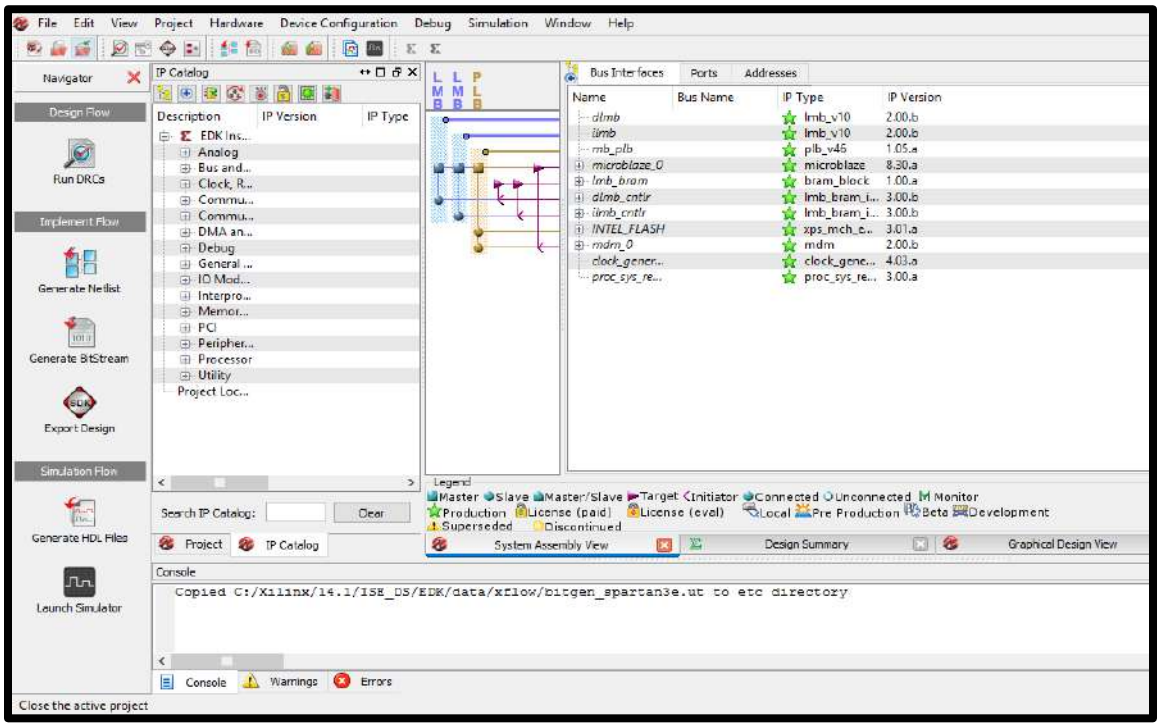


Figure III.10*Xilinx Platform Studio*

Dans cette fenêtre, on sélectionne l'élément de menu '**Hardware->Create or Import Peripheral...** Cela permet d'afficher la fenêtre "Create or Import PeripheralWizard" comme le montre la figure III.11. Cet assistant nous guidera à travers plusieurs étapes la création d'un nouveau périphérique.



Figure III.11 *Création d'un IP core*

L'étape suivante demande si on souhaite créer un nouveau périphérique ou importer un core qui est déjà créé. On sélectionne **"Create Template for a new Peripheral"** comme indique la Figure III.12.

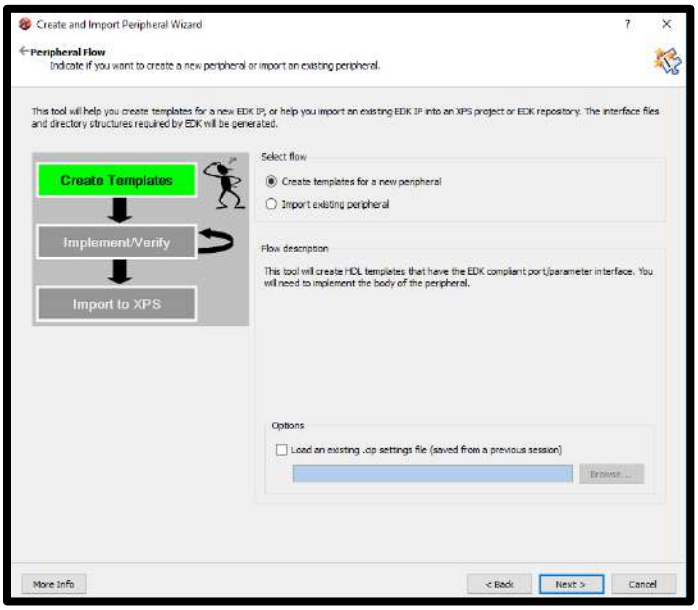


Figure III.12 *Création d'un IP core*

La fenêtre suivante nous demande où nous souhaitons stocker nos fichiers. Ici, nous avons deux options, comme le montre la figure III.13. La première option est choisie si on veut que l'IP core soit accessible d'une manière globale dans l'EDK. La deuxième option si l'IP core sera utilisé seulement dans le répertoire local, et il ne sera disponible que dans le projet. On choisit la deuxième, puis on appuie sur **"Next"**.

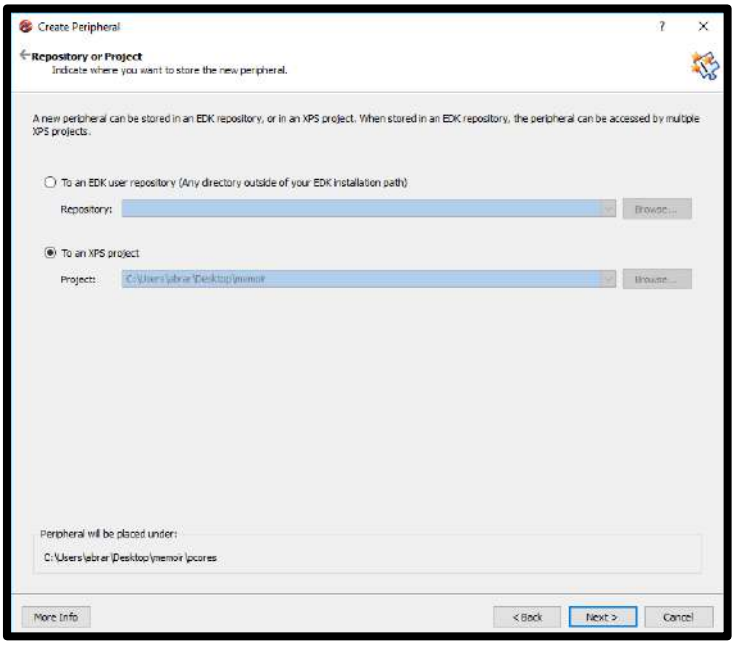


Figure III.13 *Chemin de l'IP core*

Dans la fenêtre de la figure III.14, on doit entrer le nom du périphérique et sa version. L'assistant permet de créer différentes versions du même IP core, ainsi on note que le nom de base doit contenir que des lettres minuscules et des chiffres. Dans notre cas, on choisit un nouveau IPcore nommé "**sha256_ip**", et on appuie sur "**Next**" pour continuer.

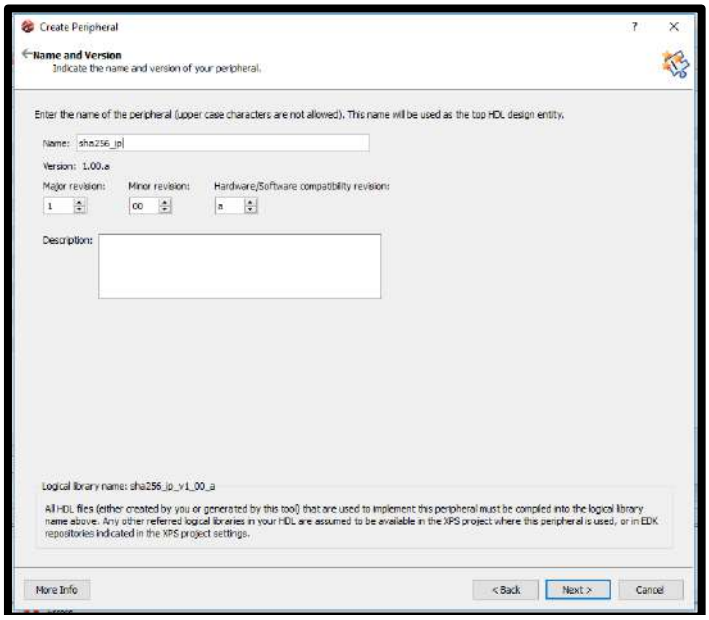


Figure III.14 *Nommage de l'IP core*

Dans cette étape, on sélectionne le type d'interface utilisée. XilinxEDK prend en charge 3 types de normes d'interface dans ses systèmes embarqués. Le premier et le plus récent est l'Architecture AXI4 qui est conçu pour différents modes de communication. Le second type est le **"Processor Local Bus (PLB 4.0)"** qui est une architecture de bus standard pour connecter le Microblaze et ses périphéries dans une structure de bus partagé. La dernière interface est le **"Fast Simplex Link (FSL)"**, qui est une liaison de type FIFO dédiée entre le processeur et le périphérique. Le Microblaze prend en charge un nombre limité de liens de FSL, mais ce canal de communication garantit la mode de communication la plus rapide entre le périphérique et le processeur. Nous avons opté pour le deuxième : **"Processor Local Bus (PLB 4.0)"** comme le montre la figure III.15, puis on appuie sur" **Next** ".

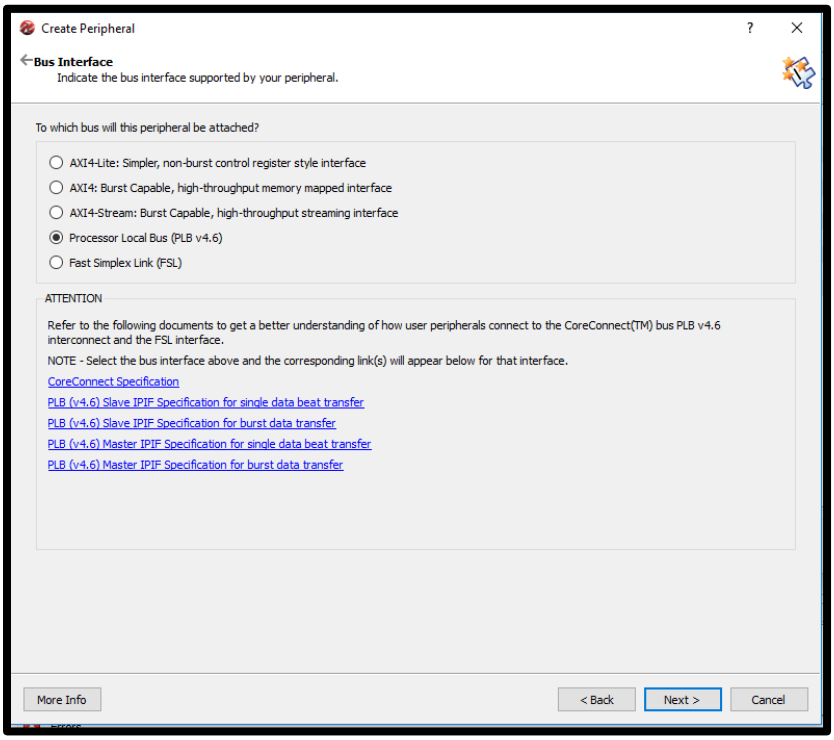


Figure III.15*Choix du type de bus*

Dans la fenêtre suivante, on appuie seulement sur **"Next".**

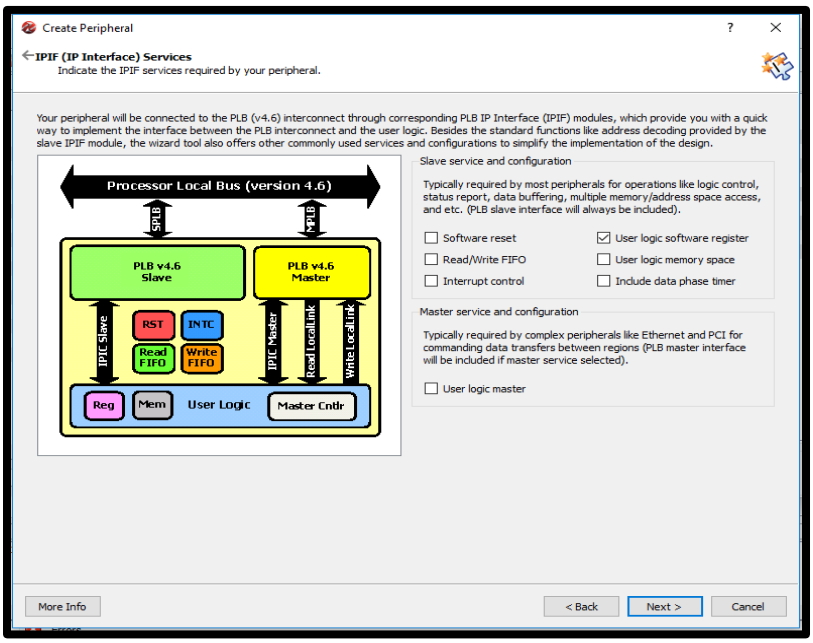


Figure III.16 *Services de l'IP core*

Dans la fenêtre de la figure III.17, nous avons la possibilité d'ajouter le support de la mémoire cache ainsi que de changer la largeur de bus de données. Nous n'avons pas fait des changements. On appuie sur **"Next"** pour continuer.

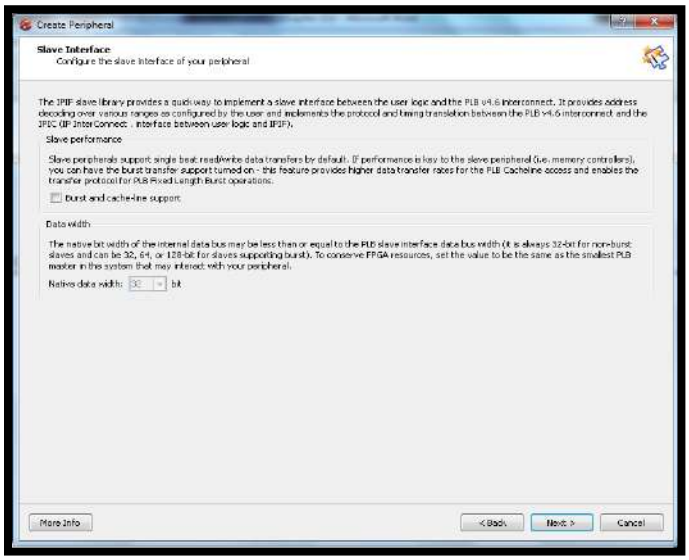


Figure III.17 *Configuration du bus de l'interface*

L'assistant va créer la logique nécessaire à l'intérieur de l'IP core pour pouvoir être accessible en lecture/écriture. Dans cette étape, on spécifie le nombre de registres dans notre conception

comme on le voit dans la figure III.18. On sélectionne 11 registres de 32 bits : 1 registre pour les signaux d'entrée : start_i, ce_i, di_wr_i, bytes_i et end_i, 1registre pour le mots de 32 bits du message d'entrée, 8 registres pour le message haché, un registre pour les signaux de sortie : erreur_o ,di_req_o et do_valid_o. Puis on appuie sur **"Next"** pour continuer.

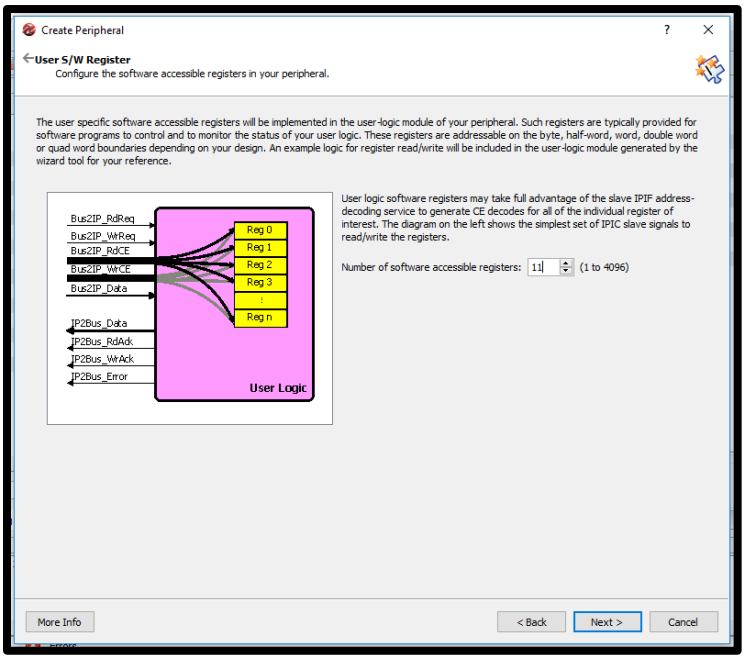


Figure III.18 *Configuration du nombre de registres de l'IP core*

Sur la fenêtre suivante (figure III.19), on ne change rien et on appuie sur **"Next"** pour continuer.

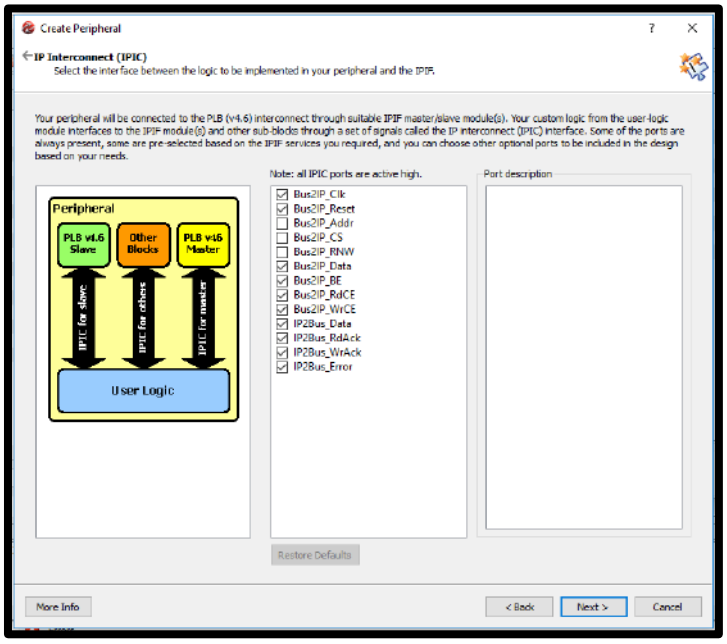


Figure III.19 *Interconnexion de l'IP core*

L'IP core peut être simulé pour la vérification fonctionnelle et le timing (Figure III.20). Afin d'effectuer la simulation, des fichiers supplémentaires sont nécessaires et sont ajoutés à la simulation des composants de bus.Dans notre projet, nous avons opté pour la vérification pratique, donc, nous n'avons pas généré ces fichiers. On appuie sur **"Next"** pour continuer.

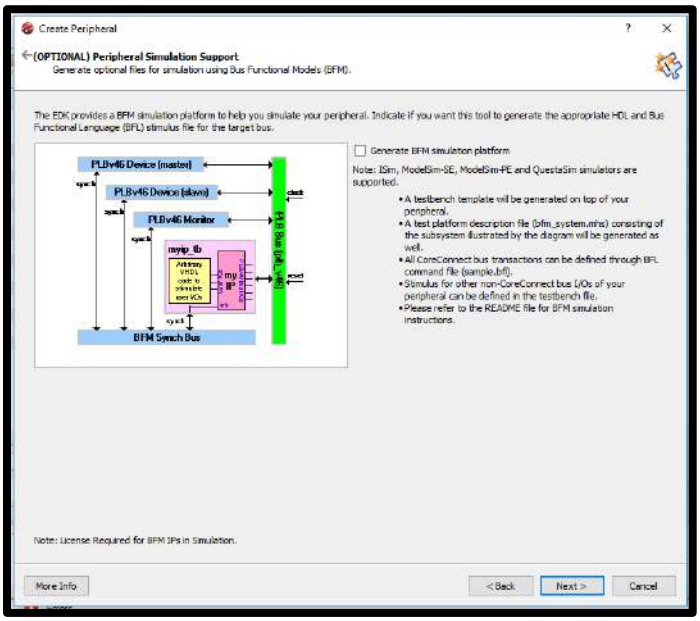


Figure III.20 *Génération les fichiers de simulation de l'IP core*

La dernière étape de l'assistant contient trois options :

1. L'option pour générer le module user_logic en Verilogou en VHDL.
2. La possibilité de générer un projet ISE pour aider à la mise en œuvre de l'IP core.Le projet ISE généré peut être utilisé pour vérifier le core avant l'intégration dans l'EDK.
3. La dernière option est de créer un pilote software pour le core qui peut être utilisé au cours du développement dusoftware.

On effectue la sélection comme il est indiqué sur la Figure III.21, et on appuie sur **"Next"** pour continue.

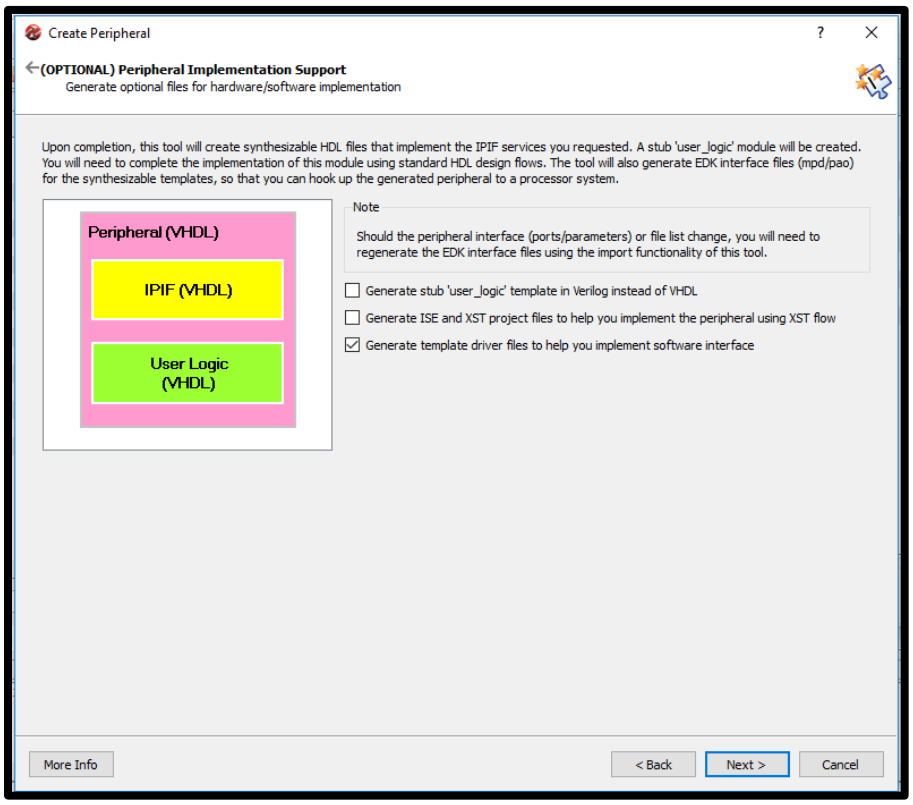


Figure III.21 *Sélection du support d'implémentation de l'IP core*

4.2 Etape N°2 : Finalisation del'IPcore SHA256

Dans la partie précédente, l'assistant de l'XPSa créé tous les fichiers nécessaires pour intégrer le core dans une conception EDK. Ces fichiers sont situés dans le répertoire local de notre projet comme le montre la Figure III.22.

Les fichiers créés sont stockés dans le répertoire principal**"pcores"** .Dans cerépertoire, on trouve un autre répertoire pour notre IP core avec le nomde**"sha256_ip_v1_00_a ".**

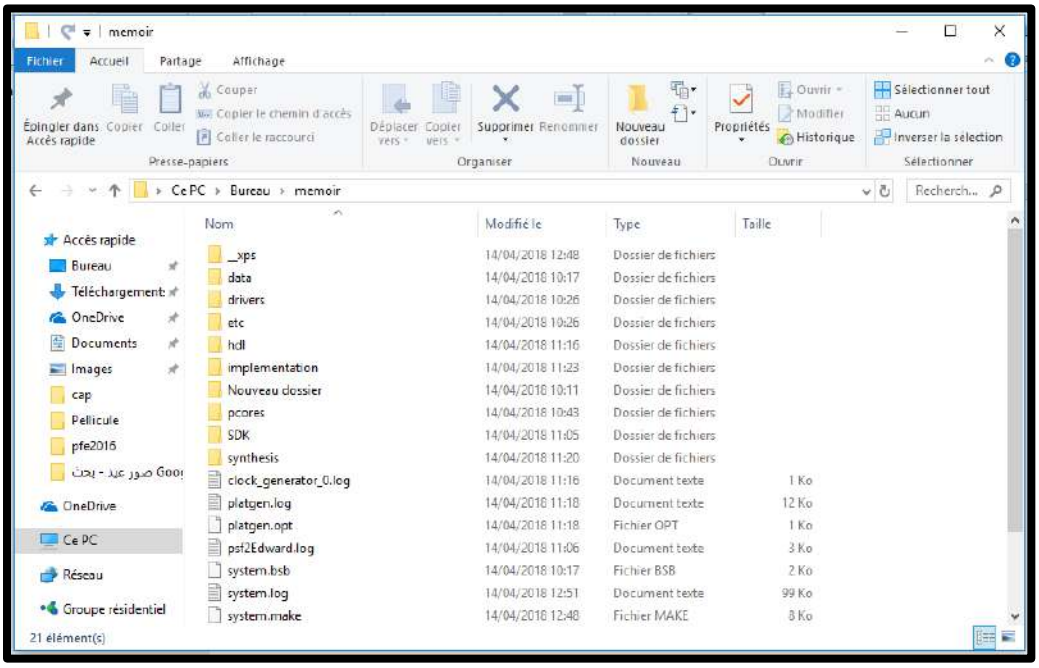


Figure III.22 *Liste des répertoires du projet.*

Le dossier **"pcores->sha256_ip_v1_00_a "**(figure III.23) contient trois répertoires :

- **"data"** : ce répertoire contient les fichiers nécessaires pour définir le IPcore dans l'EDK.LeEDK lit les fichiers dans ce répertoire pour identifier les connexions de bus, et le lesfichiers du matériel requis pour construire le core.
- **"devl"** : ce répertoire contient les fichiers pour le projet ISE.
- **"hdl"**: ce répertoire contient les fichiers de code source HDLde l'IP core.

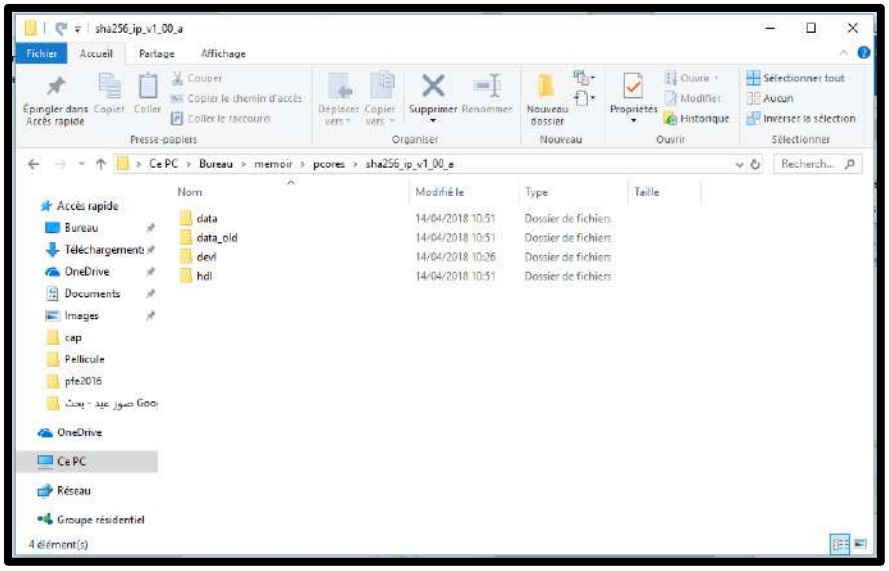


Figure III.23 *Contenu du dossier "pcores ->sha256_ip_v1_00_a ".*

Le dossier**"pcores->sha256_ip_v1_00_a ->hdl->vhdl"**contient lefichier : **user_logic.**On ajoute à ce fichier les fichiers codes sources du module de hachage SHA256 décrit précédemment (voir Figure III.24).

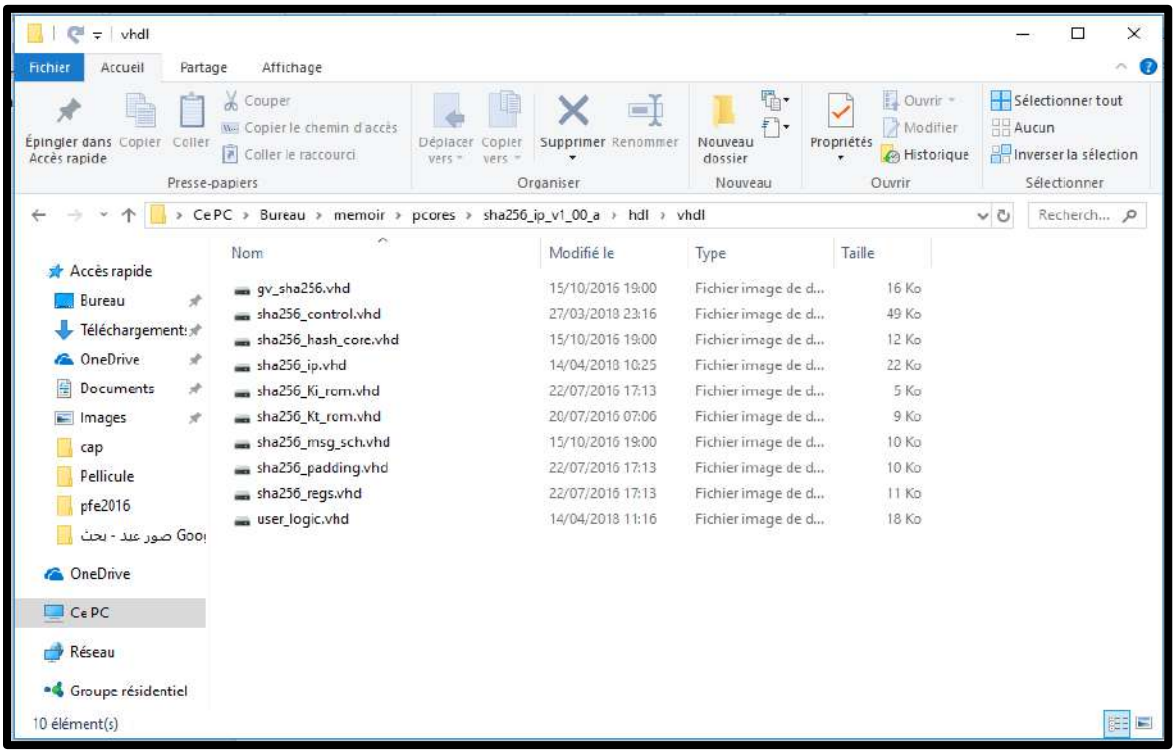


Figure III.24 *Dossier "pcores->sha256_ip_v1_00_a->hdl->vhdl"*

Ensuite, on ouvre le fichier **"user_logic.vhd"**. La figure III.25 montre les parties ajoutées et modifiées à ce fichier dans le cas du module sha256.

```
126 ------------------------------------------------------------------------------
127 -- Architecture section
128 ------------------------------------------------------------------------------
129 architecture IMP of user_logic is
130
131   --USER signal declarations added here, as needed for user logic
132
133   component gv_sha256
134     port (
135         -- clock and core enable
136         clk_i : in std_logic := 'U';                                    -- system clock
137         ce_i : in std_logic := 'U';                                     -- core clock enable
138         -- input data
139         di_i : in std_logic_vector (31 downto 0);    -- big endian input message words
140         bytes_i : in std_logic_vector (1 downto 0);  -- valid bytes in input word
141         -- start/end commands
142         start_i : in std_logic := 'U';                                  -- reset the engine and start a new
143         end_i : in std_logic := 'U';                                    -- marks end of last block data inp
144         -- handshake
145         di_req_o : out std_logic;                                       -- requests data input for next wor
146         di_wr_i : in std_logic := 'U';                                  -- high for di_i valid, low for hol
147         error_o : out std_logic;                                        -- signalizes error. output data is
148         do_valid_o : out std_logic;                                     -- when high, the output is valid
149         -- 256bit output registers
150         H0_o : out std_logic_vector (31 downto 0);
151         H1_o : out std_logic_vector (31 downto 0);
152         H2_o : out std_logic_vector (31 downto 0);
```

```
152         H2_o : out std_logic_vector (31 downto 0);
153         H3_o : out std_logic_vector (31 downto 0);
154         H4_o : out std_logic_vector (31 downto 0);
155         H5_o : out std_logic_vector (31 downto 0);
156         H6_o : out std_logic_vector (31 downto 0);
157         H7_o : out std_logic_vector (31 downto 0) );
158 end component;
159
```

```
182   --USER logic implementation added here
183
184 gv_sha256_0 : gv_sha256
185       port map(          clk_i         => Bus2IP_Clk,
186                          ce_i          => slv_reg0(0),
187                          bytes_i       => slv_reg0(8 to 9),
188                          start_i       => slv_reg0(16),
189                          end_i         => slv_reg0(24),
190                          di_wr_i       => slv_reg0(31),
191
192                          di_req_o      => slv_reg1(0),
193                          error_o      => slv_reg1(8),
194                          do_valid_o     => slv_reg1(16),
195
196                          di_i(31 downto 0)          => slv_reg2(0 to 31),
197
198                          H0_o(31 downto 0)             => slv_reg3(0 to 31),
199                          H1_o(31 downto 0)             => slv_reg4(0 to 31),
200                          H2_o(31 downto 0)             => slv_reg5(0 to 31),
201                          H3_o(31 downto 0)             => slv_reg6(0 to 31),
202                          H4_o(31 downto 0)             => slv_reg7(0 to 31),
203                          H5_o(31 downto 0)             => slv_reg8(0 to 31),
204                          H6_o(31 downto 0)             => slv_reg9(0 to 31),
205                          H7_o(31 downto 0)             => slv_reg10(0 to 31)
206                          );
```

```
226   ------------------------------------------
227   slv_reg_write_sel <= Bus2IP_WrCE(0 to 10);
228   slv_reg_read_sel  <= Bus2IP_RdCE(0 to 10);
229   slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or Bus2IP_WrCE(3) or Bus2IP_WrCE
230   slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or Bus2IP_RdCE(3) or Bus2IP_RdCE
231
232   -- implement slave model software accessible register(s)
233   SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
234   begin
235
236     if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
237       if Bus2IP_Reset = '1' then
238         slv_reg0 <= (others => '0');
239         --slv_reg1 <= (others => '0');
240         slv_reg2 <= (others => '0');
241         --slv_reg3 <= (others => '0');
242         --slv_reg4 <= (others => '0');
243         --slv_reg5 <= (others => '0');
244         --slv_reg6 <= (others => '0');
245         --slv_reg7 <= (others => '0');
246         --slv_reg8 <= (others => '0');
247         --slv_reg9 <= (others => '0');
248         --slv_reg10 <= (others => '0');
249       else
250         case slv_reg_write_sel is
251           when "10000000000" =>
252             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
```

```
253               if ( Bus2IP_BE(byte_index) = '1' ) then
254                 slv_reg0(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
255               end if;
256             end loop;
257           when "01000000000" =>
258             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
259               if ( Bus2IP_BE(byte_index) = '1' ) then
260                 --slv_reg1(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
261               end if;
262             end loop;
263           when "00100000000" =>
264             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
265               if ( Bus2IP_BE(byte_index) = '1' ) then
266                 slv_reg2(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
267               end if;
268             end loop;
269           when "00010000000" =>
270             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
271               if ( Bus2IP_BE(byte_index) = '1' ) then
272                 --slv_reg3(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
273               end if;
274             end loop;
275           when "00001000000" =>
276             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
277               if ( Bus2IP_BE(byte_index) = '1' ) then
278                 --slv_reg4(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
279               end if;
```

```
280             end loop;
281           when "00000100000" =>
282             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
283               if ( Bus2IP_BE(byte_index) = '1' ) then
284                 --slv_reg5(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
285               end if;
286             end loop;
287           when "00000010000" =>
288             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
289               if ( Bus2IP_BE(byte_index) = '1' ) then
290                 --slv_reg6(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
291               end if;
292             end loop;
293           when "00000001000" =>
294             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
295               if ( Bus2IP_BE(byte_index) = '1' ) then
296                 --slv_reg7(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
297               end if;
298             end loop;
299           when "00000000100" =>
300             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
301               if ( Bus2IP_BE(byte_index) = '1' ) then
302                 --slv_reg8(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
303               end if;
304             end loop;
305           when "00000000010" =>
306             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
```

```
307               if ( Bus2IP_BE(byte_index) = '1' ) then
308                -- slv_reg9(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
309               end if;
310             end loop;
311           when "00000000001" =>
312             for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
313               if ( Bus2IP_BE(byte_index) = '1' ) then
314              --   slv_reg10(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
315               end if;
316             end loop;
317           when others => null;
318         end case;
319       end if;
320     end if;
321
322   end process SLAVE_REG_WRITE_PROC;
323
324   -- implement slave model software accessible register(s) read mux
325   SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1, slv_reg2, slv_reg3, slv_reg4, slv_re
326   begin
327
328     case slv_reg_read_sel is
329       when "10000000000" => slv_ip2bus_data <= slv_reg0;
330       when "01000000000" => slv_ip2bus_data <= slv_reg1;
331       when "00100000000" => slv_ip2bus_data <= slv_reg2;
332       when "00010000000" => slv_ip2bus_data <= slv_reg3;
333       when "00001000000" => slv_ip2bus_data <= slv_reg4;
```

Figure III.25 *Modification apportée au code source "user_logic"*

4.3 Etape N°3 : Intégration de l'IP core dans le système embarqué

On commence par la modification du fichier PAO comme il est indiqué sur la figure III.26 dans le cas du module sha256.

```
1  ##############################################################################
2  ## Filename:          C:\Users\abrar\Desktop\memoir\pcores/sha256_ip_v1_00_a/data/sha256_ip_v2_1_0.pao
3  ## Description:       Peripheral Analysis Order
4  ## Date:              Sat Apr 14 10:51:02 2018 (by Create and Import Peripheral Wizard)
5  ##############################################################################
6
7  lib proc_common_v3_00_a all vhdl
8  lib plbv46_slave_single_v1_01_a all vhdl
9  lib sha256_ip_v1_00_a user_logic vhdl
10 lib sha256_ip_v1_00_a sha256_ip vhdl
11 lib sha256_ip_v1_00_a gv_sha256 vhdl
12 lib sha256_ip_v1_00_a sha256_control vhdl
13 lib sha256_ip_v1_00_a sha256_hash_core vhdl
14 lib sha256_ip_v1_00_a sha256_Ki_rom vhdl
15 lib sha256_ip_v1_00_a sha256_Kt_rom vhdl
16 lib sha256_ip_v1_00_a sha256_msg_sch vhdl
17 lib sha256_ip_v1_00_a sha256_padding vhdl
18 lib sha256_ip_v1_00_a sha256_regs vhdl
19
```

Figure III.26*Fichier sha256_ip_v2_1_0.pao*

Après, on importe l'IP core déjà créé dans l'étape 1 et modifier dans l'étape 2. On sélectionne dans la fenêtre de l'XPS, l'élément de menu **'Hardware->Create or Import Peripheral..... ->Next->import existingperipheral'** ,puis on clique sur **"Next"** pour continuer (Figure III.27).

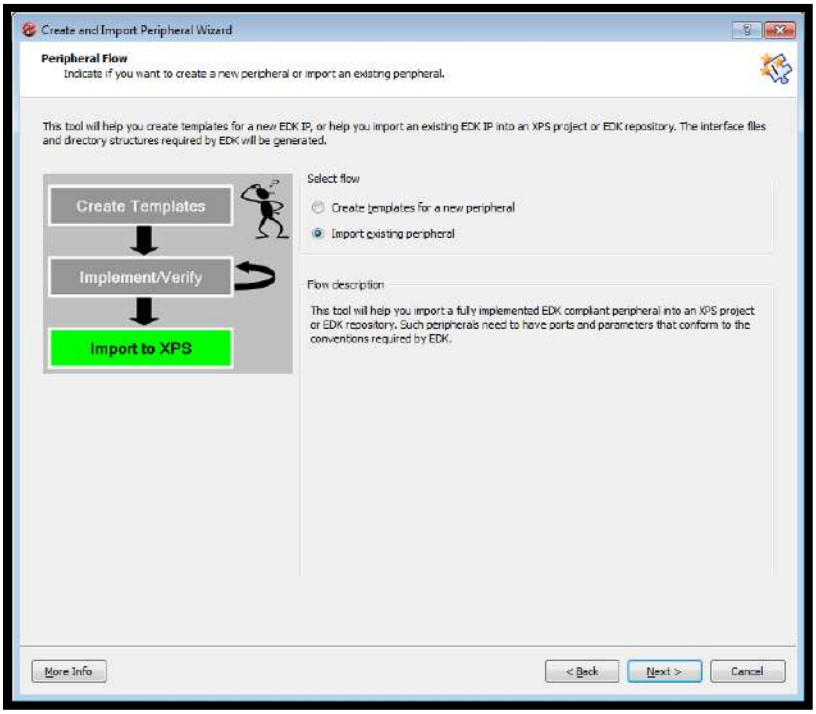


Figure III.27 *Assistant de création ou chargement de l'IP core*

Cliquer sur **"yes"** et **"Next"** pour aller à la fenêtre suivante.

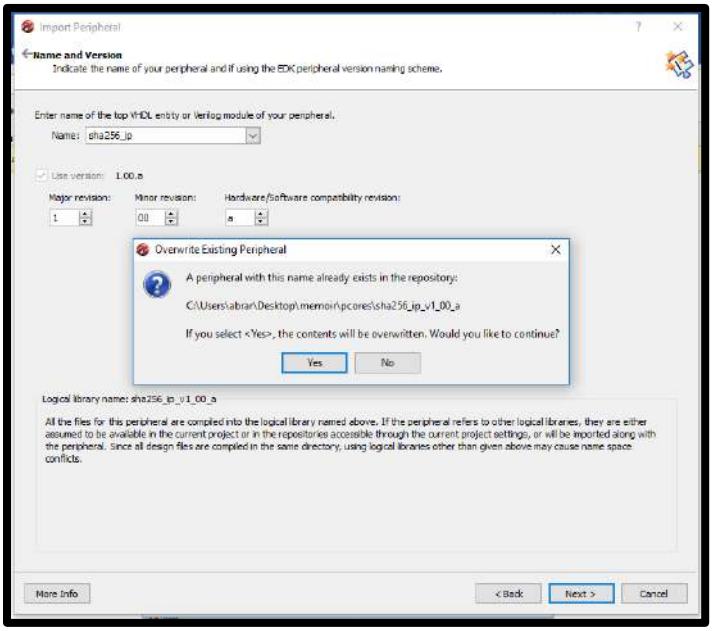


Figure III.28 *Import de l'IP core*

Dans la figure III.29, on a choisie**"HDL source file" ->"Next".**

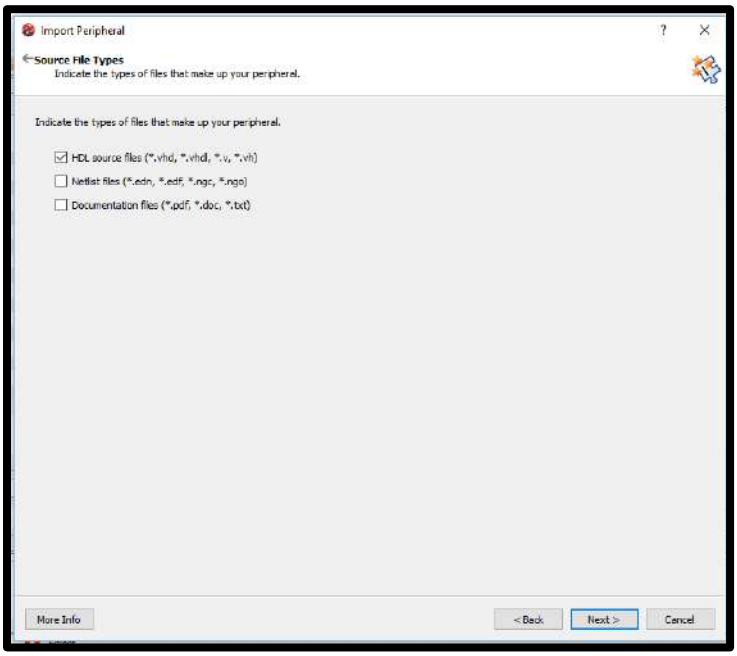


Figure III.29 *Types de fichiers sources de l'IP core*

Dans l'étape suivante (figure III.30), onchoisit**" use existingperipheralanalysisorder file (*.pao) "** et on indique au logiciel le chemin du répertoire du **" sha256_ip_v2_1_0.pao"**etensuite on appuie sur **"Next".**

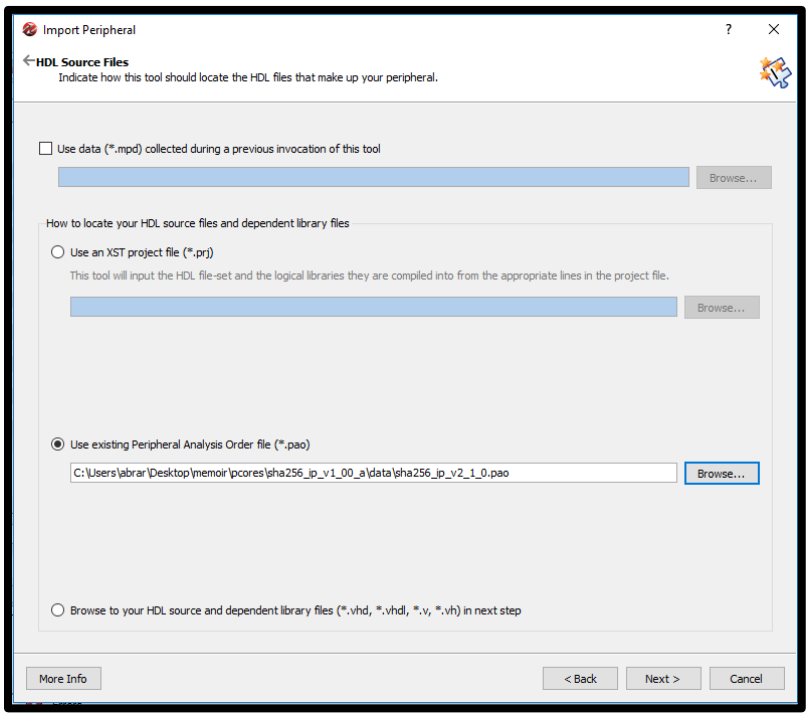


Figure III.30 *Indication du chemin du fichier pao*

La fenêtre suivante de l'assistant est illustrée à la figure III.31, on appuie sur **"Next".**

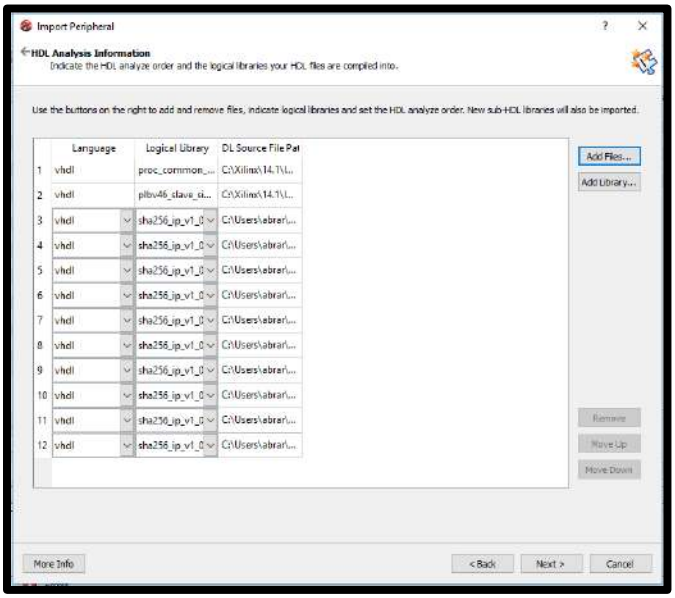


Figure III.31 *Analyse des fichiers du projet*

Ensuite, on choisit le bus **" PLBV46 slave(SPLB)"** et appuyez **"Next".**

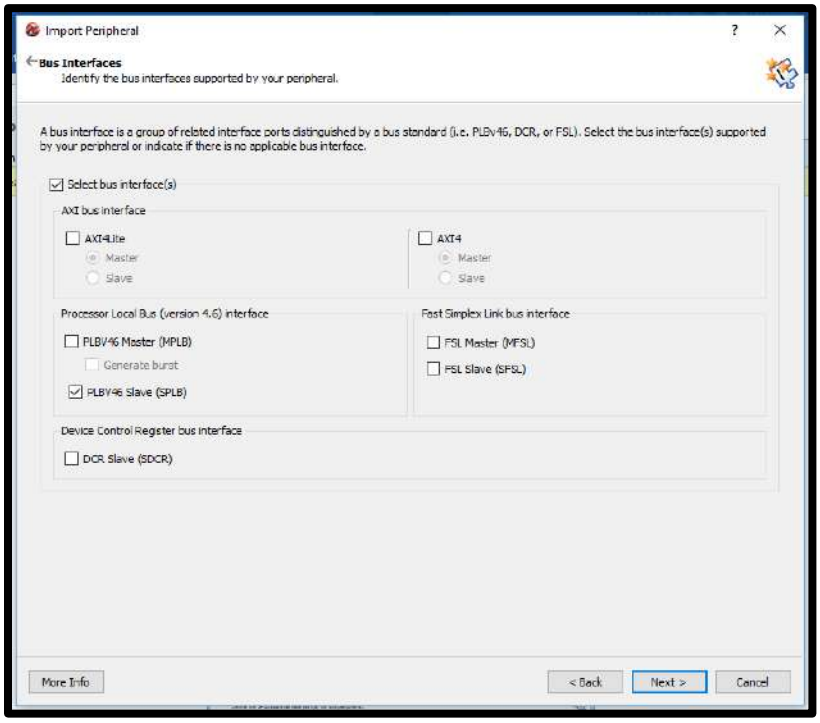


Figure III.32 *Choix du bus PLB*

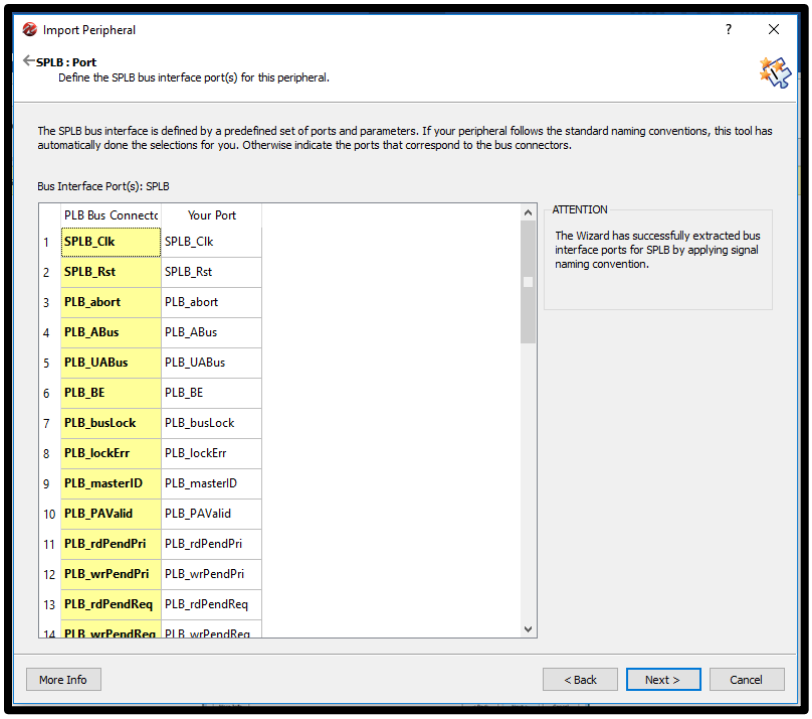


Figure III.33 *Configuration bus SPLB de l'IP core*

La figure III.34montre les paramètres choisis du bus SPLB.

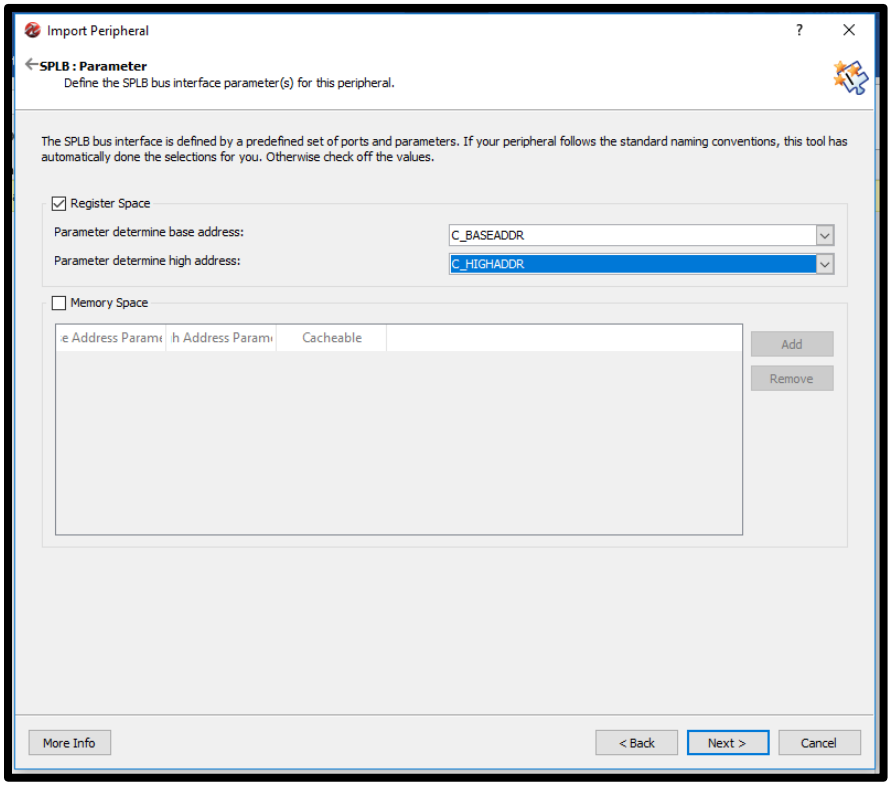


Figure III.34 *Configuration paramètres bus SPLB*

Dans la fenêtre suivante (figure III.35), on décoche la case " select and configure interrupt (s)". C'est-à-dire on n'utilise pas les interruptions.

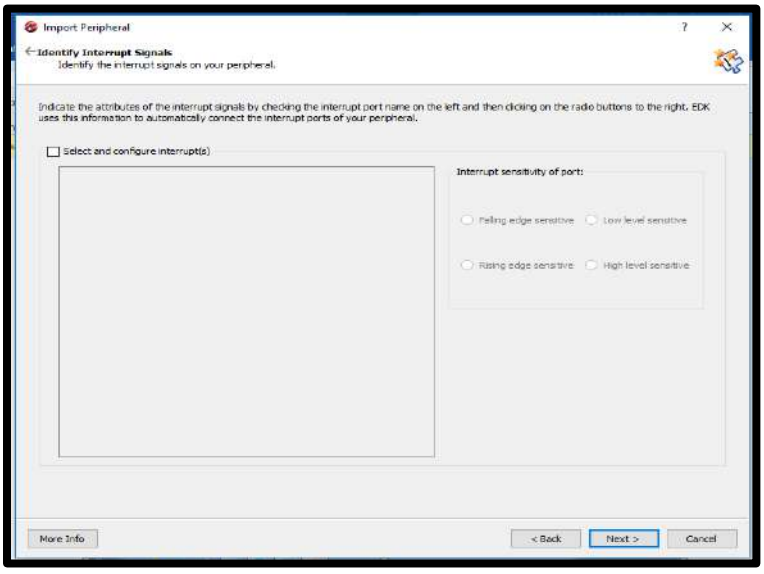


Figure III.35 *Sélection mode des interruptions*

Et pour les étapes qui suivent, on clique toujours sur **"Next".** A la fin, on appuie sur **finich**.

Dans l'étape suivante, on ajoute l'IP coreausystème déjà crée, basé sur un Microblaze à 50 MHz, une mémoire de données (dlmb) de 16kB, une mémoire d'instructions (ilmb) de 16kB et un UART pour la communication avec le PC.

On sélectionne le bouton "IP Catalog" dans la partie gauche de la fenêtre principale comme le montre la Figure III.36

On double-clique sur le core qui va diriger l'outil XPS à ajouter une instance de ce composant de sha256.

D'abord, il nous demandera si on veut ajouter l'IP core de **"**sha256_ip**"** à notre conception. On appuie sur "**yes**". Ensuite, il affiche une fenêtre comme le montre la Figure III.36.

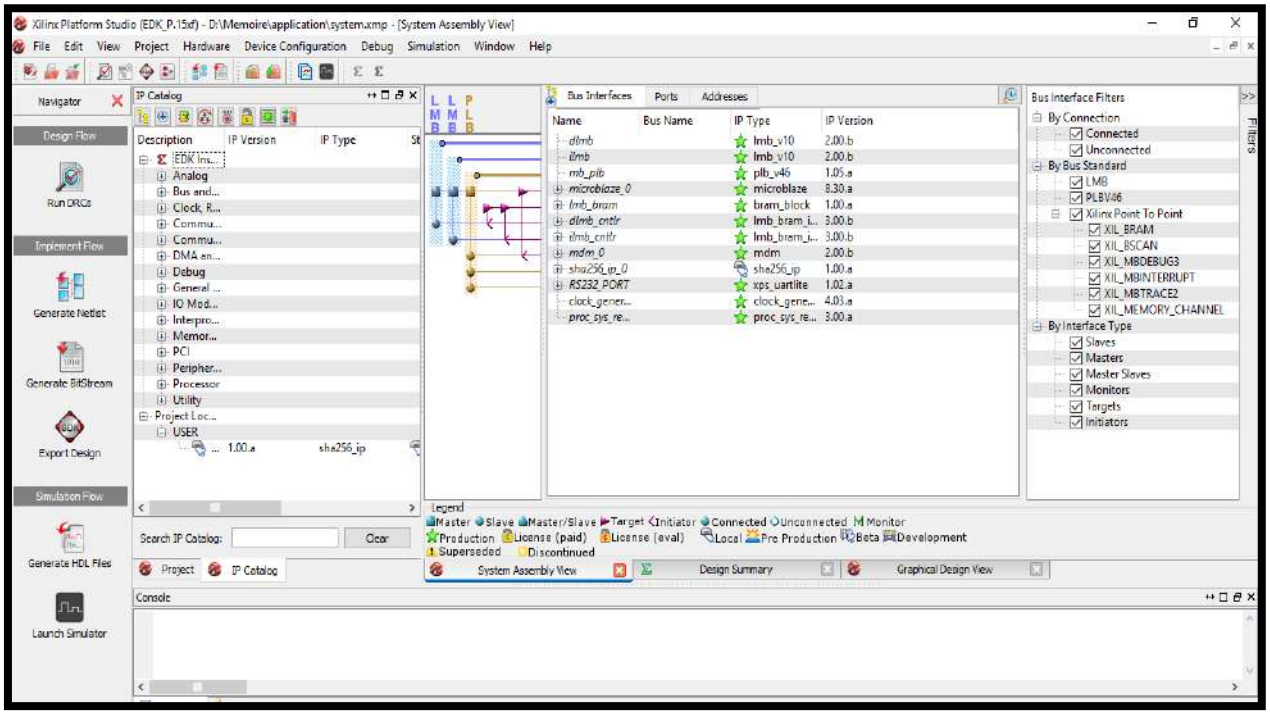


Figure III.36 *Visualisation dans « IP Catalog de IP core*

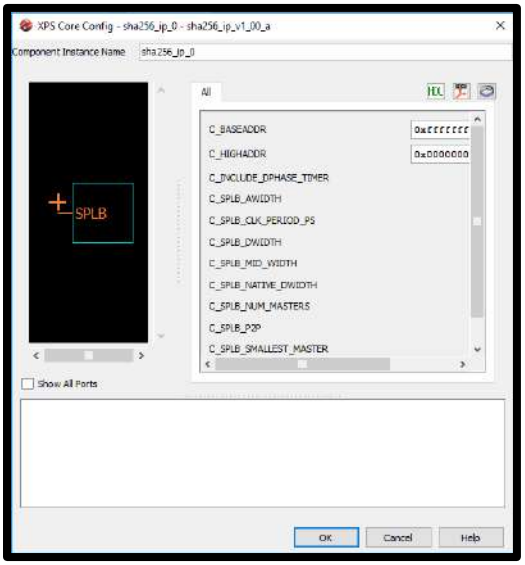


Figure III.37 *Configuration de l'IP core*

On appuie sur **"OK"** pour on ajoute le composant à notre conception comme il est indiqué à la figure III.38. Ce nouveau composant est prêt à être connecté au bus PLB commun du MicroBlaze. On clique sur le signe "+" à côté du nom du core (**"sha256_ip_0").** On remarque une connexion nommée **"SPLB"**. Cela indique simplement que cette composante peut être reliée au bus PLB. On clique sur la liste **"No Connection"** et on choisit**"mb_plb"** de la liste.

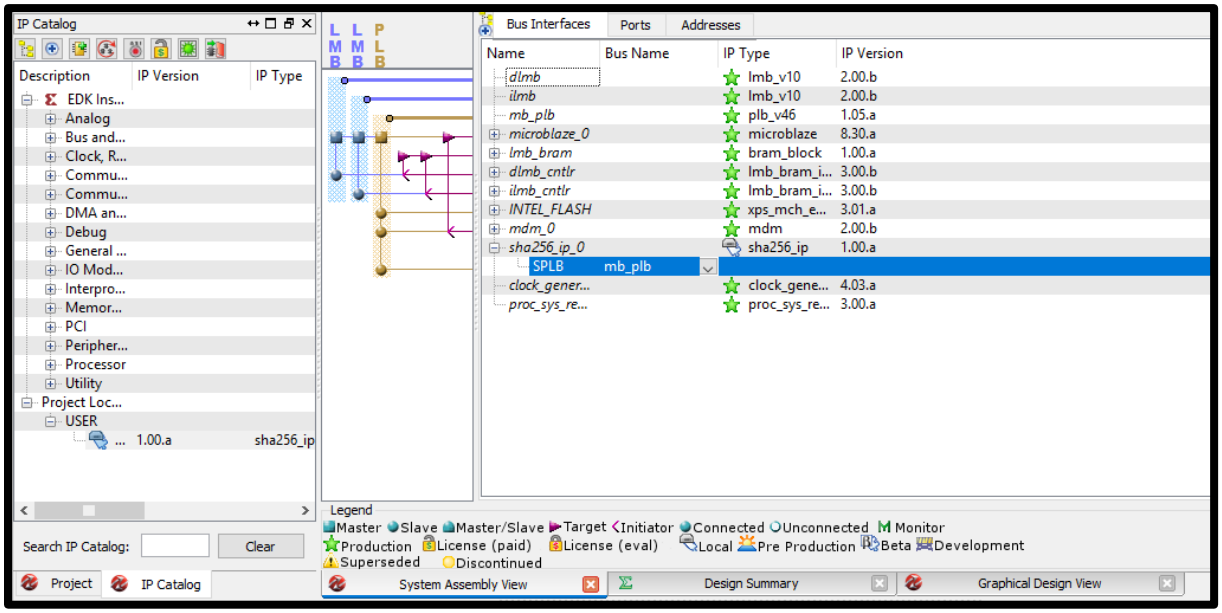


Figure III.38 *Nouveau composant ajouté au système*

Enfin, on ajoute les adresses dumodule ajouté comme le représente la figure III.39.

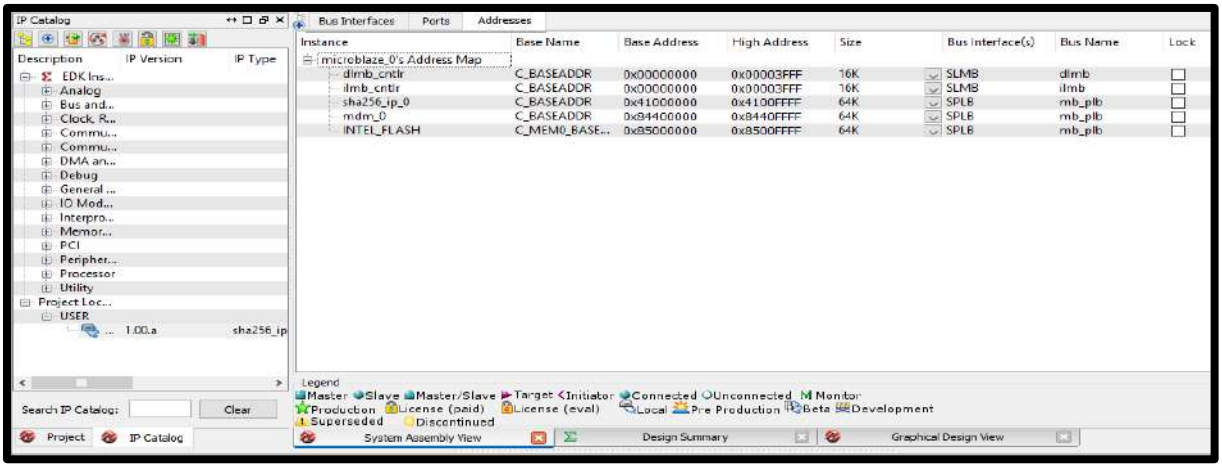


Figure III.39 *Configuration des zones d'adresses de module sha256*

4.4 Etape N°4 : Conception de la partie logicielle

Après génération du fichier bitstream de l'architecture matérielle, c'est-à-dire, que notre système à base du microblaze ainsi que l'IP core sha256 sont implémentés. On passe à la conception logicielle en langage C.On clique sur l'icône Export Design, l'environnement SDK démarre. On crée un nouveau projet « **sha256 ».**La figure III.40 montre l'environnement SDK avec notre projet développé.

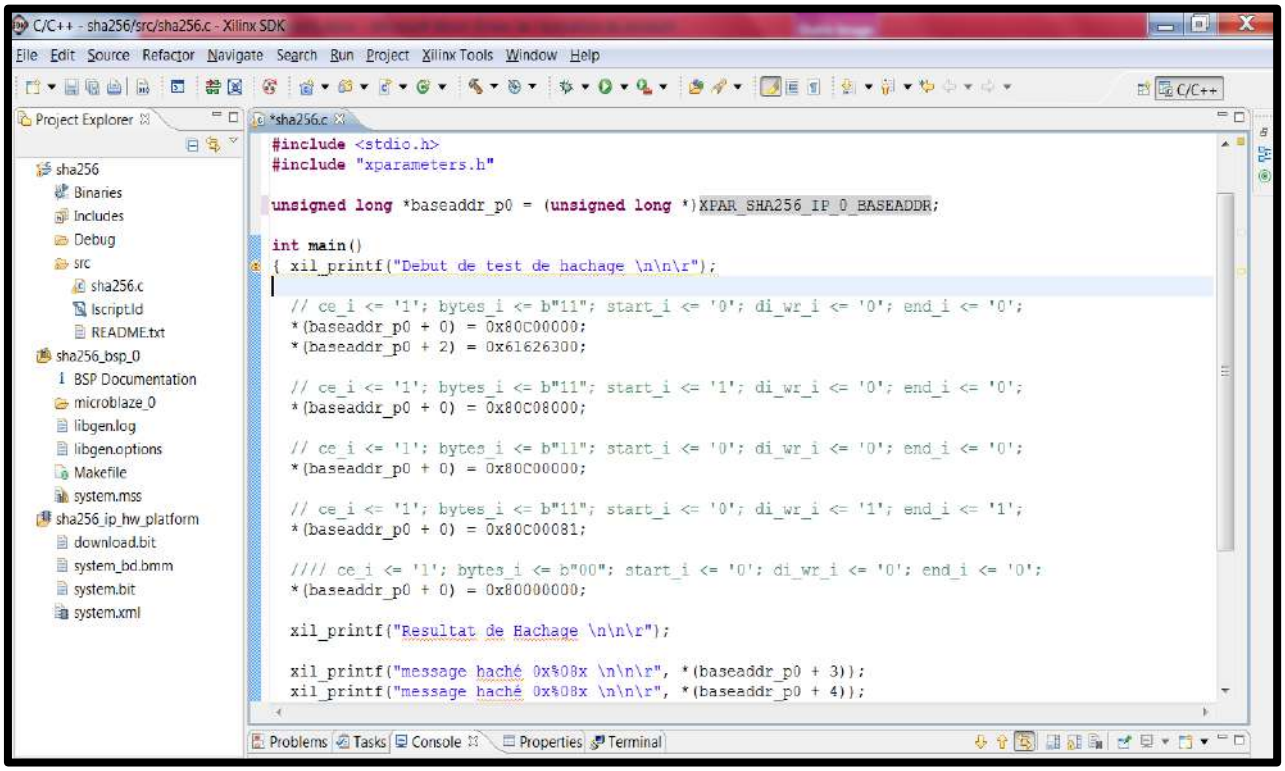


Figure III.40*Environnement SDK avec le projet développé*

5 Résultats des tests pratiques

Pour faire des tests pratiques du système développé sur un circuit FPGA, on a utilisé les outils suivants :

- La carte FPGADigilentNexys 2-500C.
- Un ordinateur portable.
- Un câble adaptateur liaison série RS232 / USB.
- Un hyper Terminal HERCULES 3.2.5 pour l'envoi et la réception des données en hexadécimal via le port USB émulé port série RS232.

Afin d'assurer la synchronisation entre l'ordinateur et la carte, on a utilisé la configuration suivante : le débit de transfert = 9600 bps ; la taille de la donnée = 8 bits ; pas de bits de parité ; 1 bit stop.

Pour valider le fonctionnement de l'IP corequi implémente la fonction de hachage SHA256, nous avons réalisé les mêmes tests de simulations présentés auparavant**.**Les figures III.41, III.42 et III.43 représentes les résultats obtenus. On remarque que ces résultats reçus de la carte FPGA et affichés sur l'HyperTerminal sur le PC sont corrects.

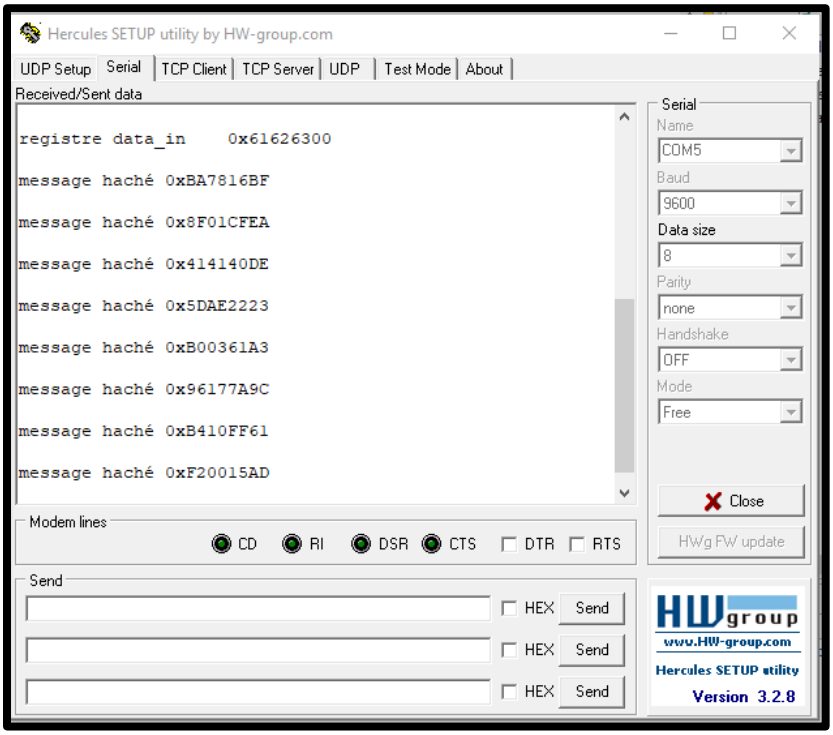


Figure III.41 *Résultat du 1^er^ test pratique.*

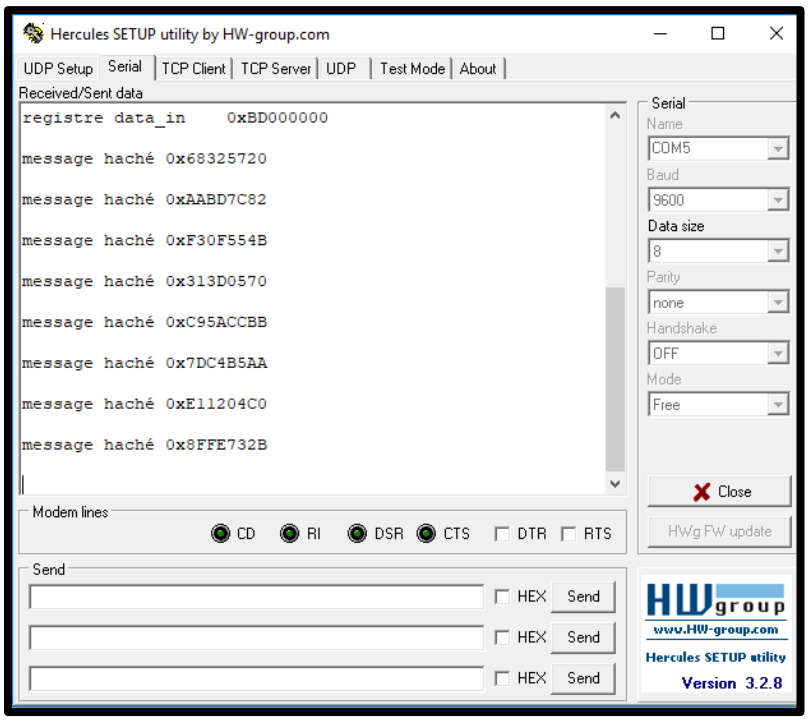


Figure III.42 *Résultat du 2ème test pratique.*

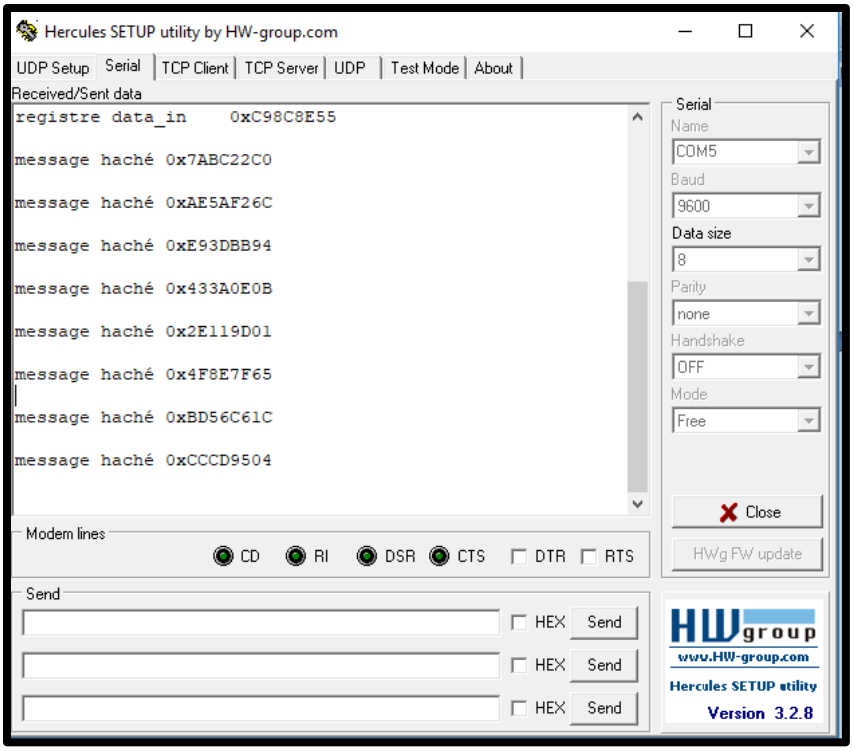


Figure III.43*Résultat du 3ème test pratique.*

6 Conclusion

Dans ce chapitre, nous avons exposé en premier lieu un circuit conçu en VHDL de l'algorithme de hachage cryptographique SHA256. Ensuite nous avons validé le bon fonctionnement de ce circuit par des simulations. Après, ce circuit a été transformé en un IP core par une conception conjointe logiciel/matériel. Enfin, le système embarqué intégrant l'IP coredéveloppé a été examiné pratiquement. Les résultats du test sont entièrement validés en utilisant une carte DigilentNexys 2-500C.

Conclusion générale

Les techniques classiques de conception des systèmes embarqués séparent le hardware du software dans le cycle de conception. Ainsi, les concepteurs de hardware spécifient leurs produits sans apprécier exactement les besoins, en ressources matérielles, des logiciels qui s'exécuteront dessus. De leur côté, les concepteurs de logiciels ne profitent pas convenablement des ressources matérielles de la plate-forme hardware sur laquelle seront implantés leurs produits.

Pour soulever ces limitations, nous avons utilisé une conception conjointe logiciel/matériel (Codesign), dans le but de réaliser un système embarqué basé sur un softcore de type Microblaze avec un IP core qui implémente l'algorithme de hachage SHA256.

L'utilisation des circuitsFPGAconsidérés comme des systèmes reconfigurables est une solution qui permet la possibilité de concevoir des IP core d'une manière hardware, ainsi que le développement et la validation rapide de nouveaux principes de sécurité. Ces composants peuvent être considérés comme des accélérateurs matériels du système embarqué.

Nous avons utilisé l'outil de développement XPS de Xilinx pour créer la partie matérielle. Ensuite nous avons conçu par l'outil SDK la partie logicielle exécutée par le Microblaze, qui joue un rôle d'un testbench de l'IP core. Des tests pratiquesavec les exemples du document officiel, ont été validés sur une carte de type DiglilentNexys 2 Spartan 3E 500.

Bibliographie

[1] : https://fr.wikipedia.org/wiki/Fonction_de_hachage, consulté le 02/04/2018.

[2] : Les Fonctions de Hachage. http://bruno.maitresdumonde.com/optinfo/ads-scie/2008.pdf consulté le 04/04/2018.

[3] : http://www.wikiwand.com/fr/Attaque_de_collisions, consulté le 02/04/2018.

[4] : Thomas Fuhr, Conception, preuves et analyse de fonctions de hachage cryptographiques, thèse de doctorat, université Télécom ParisTech.

[5] : Anas Abou Elkalam, cours sur la cryptographie, Ecole nationale supérieure d'électrotechnique, d'électronique, d'informatique, d'hydraulique et des télécommunications.

[6] : https://fr.wikipedia.org/wiki/SHA-2, consulté le 20/04/2018.

[7] : https://fr.wikipedia.org/wiki/SHA-2#Bourrage, consulté le 20/04/2018.

[8] : Francois Rodier, cours sur l'utilisation de la cryptographie par bloc foctions de hachage, université de Marseille, http://iml.univ-mrs.fr/~kohel/tch/M2-CryptoSymetrique/CM/FnsHachage.pdf consulté Le 19/04/2018.

[9] : Christina Boura, Analyse de Fonctions de Hachage Cryptographiques, thèse de doctorat, Université Pierre et Marie Curie, France, 2012.

[10] :Fabrice Caignet, Les circuits logiques programmables, LesFPGA, https://homepages.laas.fr/fcaignet/Cours/presentation_FPGA.pdf

[11] : Jean-Luc Danger, cours sur les circuits logiques programmables FPGAs, 2010. UniversitéTélécom ParisTech.

[12] : GuellalAmar, Les circuits FPGA description et applications, Bulletin N°24, Centre de recherche CDER, 2012.

[13] : El Gemayel Charbel, Approche comportementale pour la validation et le test système des systèmes embarqués; application aux dispositifs médicaux embarqués, thèse de doctorat, INSA Lyon et Université Libanaise, 2014.

[14] : Frédéric Leger. Architectures Système sur FPGAXilinx,http://triplea.fr/alchimie2009/documents/PresentationFredericLeger.pdf ,consulté le 25/04/2018.

[15]:AhceneYoucefBenabdallah , RachidBoudour. Méthodologie de conception à base d'IPs, Département d'informatique, Université Badji-Mokhtar Annaba.

[16] : Guillaume Savaton, Emmanuel Casseau, Eric Martin, Catherine Lambert-Nebout, Composants Virtuels Comportementaux pour Applications de Compression d'Images.Université de Bretagne Sud, Centre de Recherche, consulté le 06/05/2018

[17] : K.Messaoudi, E. Bourennane S. Toumi M. Touiza A. Yahi, Etude et Implantation des Systèmes de Communication Dans les Systèmes sur Puce (SoCs et SoPCs), International Conference on embedded systems in telecommunications and Instrumentation, november 2012, Annaba Algeria

[18] : MicroBlaze processor reference guide, consulté le 02/05/2018.

[19] :Messaoudi, K. Etude et implementation des systeme de communication dans les systeme sur puce(SOC et SOPC).

[20] :https://www.xilinx.com/support/documentation/ip_documentation/plb_arbiter.pdf,

consulté le 29/04/2018.

[21] : On-Chip Peripheral Bus, April 2001, http://www.cs.columbia.edu/~sedwards/

classes/2005/emsys-summer/opb_ibm_spec.pdf

[22] : Kamel Messaoudi, Traitement des signaux et images en temps réel : implantation de H.264 sur MPSoC, thèse de doctorat, Université de Badji Mokhtar Annaba, 2012.

[23] : https://opencores.org/project/sha256_hash_core, consulté le 06/05/2018

[24] : FIPS PUB 180-2, Federal Information Processing Standards Publication, 2002.