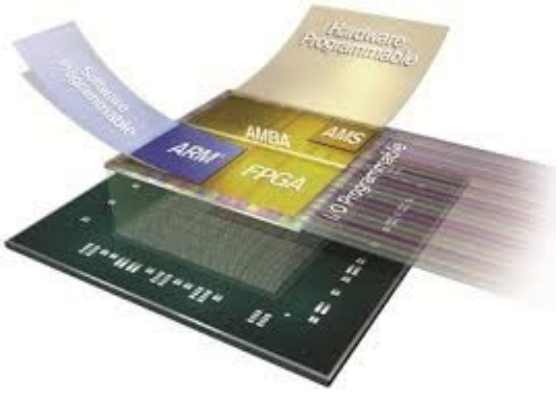
Antonin BROQUET
Michael PERRIN
2017



MINI PROJET System On Chip Rapport

Rédacteurs	Antonin BROQUET [AB], Michael PERRIN [MP]
Date de création	22 septembre 2017
Version	1.0

Antonin BROQUET
Michael PERRIN
2017

SOMMAIRE

Antonin BROQUET
Michael PERRIN
2017

1. Introduction

Ce document a pour but de détailler l'architecture hardware et software du système réalisé ainsi que l'interaction entre les différents modules (logiciels, bus, IP…). La méthode suivie pour la validation du système sera également abordée.

Antonin BROQUET
Michael PERRIN
2017

2. Spécification fonctionnelle

2.1 Introduction

Le SoC conçu fournit aux processeurs un périphérique matériel réalisant un produit de matrice de dimension (4,4).

2.2 Architecture globale

La figure 1 décrit la vue globale de l'architecture du SoC:

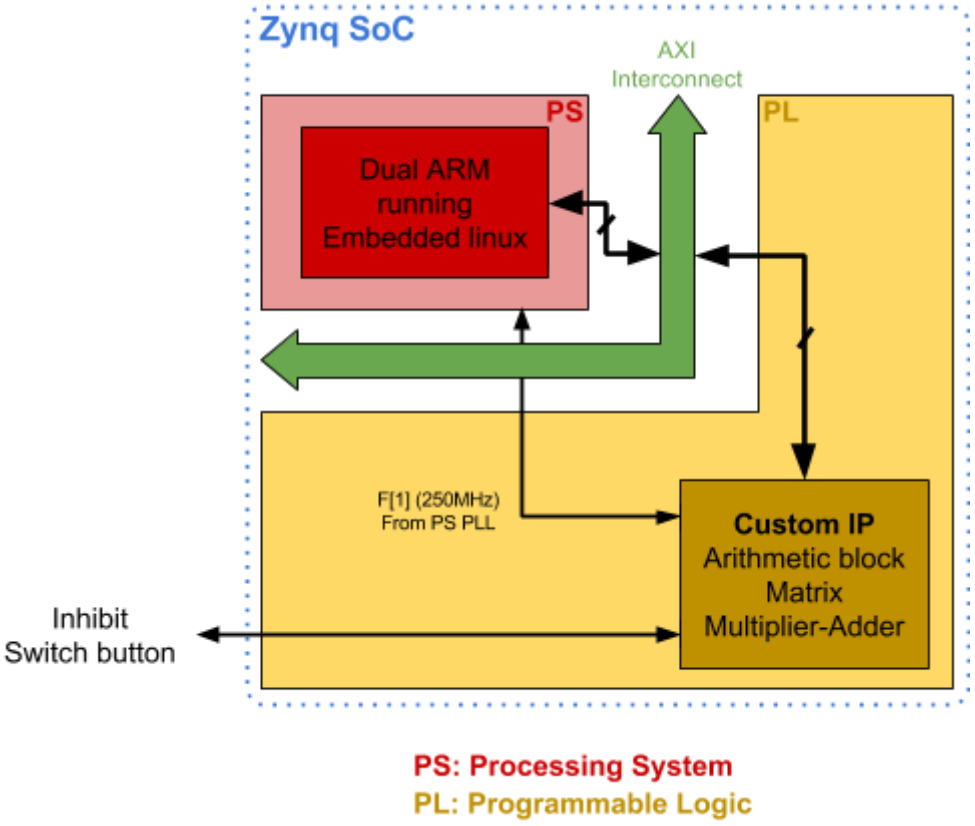


Figure 1: Architecture globale du SoC

Le Zynq est décomposé en 2 parties principales:

- "Processing System (PS)": cette partie contient les processeurs ARM et leur périphériques (MMU, Cache, NEON…).
- "Programmable Logic (PL)": cette partie contient de la logique programmable (FPGA).

Les fonctionnalités logicielles sont implantées dans la PS tandis que le périphérique matériel est implanté dans la PL. Les 2 parties sont interconnectées via un bus d'interconnexion, ici le bus AXI.

2.3 Fonctionnalités logicielles

Un linux embarqué a été choisi pour la partie PS. Le kernel est configuré en mode symétrique, c'est à dire qu'il utilise les 2 processeurs ARM.
Un ou plusieurs programmes peuvent ainsi s'exécuter sur le système et utiliser le périphérique de produit matriciel. Pour cela, un mécanisme de verrouillage du périphérique a été mis en place.
Enfin, un programme de test permet, suivant les options passés en paramètres, de valider l'utilisation du périphérique depuis l'espace utilisateur. Il permet aussi de comparer les performances (temps d'exécution) d'un calcul de produit matriciel qui utilise le périphérique dédié, qui n'utilise pas le périphérique et qui utilise le périphérique NEON présent dans les ARM.

2.4 Fonctionnalités matérielles

Un IP VHDL a été développé afin d'implanter le produit matriciel dans la PL. Le produit matriciel est effectué en 1 cycle d'horloge. L'horloge est à une fréquence de 250MHz et est fournie par la PLL de la PS fournissant aussi l'horloge de 100MHz du bus AXI. La partie PL possède donc 2 domaines d'horloge. Bien que ces 2 fréquences soient issues de la même PLL, elle sont traitées comme des horloges asynchrones, et des mécanismes de synchronisation et de croisement de domaine ont été mise en place.

Le module est interfacé par le logiciel à travers le bus d'interconnexion AXI. Dans cette première version, l'IP est intégré dans un périphérique AXI de type AXI-Lite. En effet, seul un registre de contrôle/statut et les registres contenant les éléments des matrices d'entrées et de la matrice de sortie sont nécessaires. Aucun flux haute performance n'est utilisé (AXI-Stream).

Le registre de contrôle et de statut (CSR), contient un bit permettant de démarrer le calcul (effectué en 1 cycle d'horloge) une fois que les registres contenants les éléments des matrices d'entrées sont valides (ont été écrits). Ce bit est self-clearing (automatiquement remis à 0) et de type monostable.
Il contient aussi un bit de statut permettant d'indiquer que le calcul a été effectué et que les registres, contenants les éléments de la matrice de sortie, sont valides.
Enfin, il contient un bit de verrouillage permettant à plusieurs programmes d'utiliser le périphérique. Il s'agit d'un mécanisme simple qui ne gère pas l'arbitrage.

Un signal extérieur de type bouton "switch" permet d'inhiber le périphérique. C'est à dire que le logiciel ne peux plus démarrer le calcul et les registres contenants les éléments de la matrice de sortie sont figés.

3. Architecture du système

L'architecture globale est donnée dans la figure 1. Cette section traite et détaille l'architecture matérielle et logicielle séparément.

3.1 Architecture matérielle

L'IP est intégrée dans la PL et est destinée à accélérer les produits matriciels de manière à décharger les processeurs.

3.1.1 Description

L'IP effectue le produit de 2 matrices de dimensions (4,4). Soit C la matrice de sortie et A et B les deux matrices d'entrées. On a:

$$C \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} = A \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times B \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

avec:

$$c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31} + a_{14} \times b_{41}$$
$$c_{12} = a_{11} \times b_{12} + a_{12} \times b_{22} + a_{13} \times b_{32} + a_{14} \times b_{42}$$
$$c_{13} = a_{11} \times b_{13} + a_{12} \times b_{23} + a_{13} \times b_{33} + a_{14} \times b_{43}$$
$$c_{14} = a_{11} \times b_{14} + a_{12} \times b_{24} + a_{13} \times b_{34} + a_{14} \times b_{44}$$

$$c_{21} = a_{21} \times b_{11} + a_{22} \times b_{21} + a_{23} \times b_{31} + a_{24} \times b_{41}$$
$$\dots = \dots + \dots + \dots + \dots$$
$$c_{24} = a_{21} \times b_{14} + a_{22} \times b_{24} + a_{23} \times b_{34} + a_{24} \times b_{44}$$
$$\dots$$
$$c_{44} = a_{41} \times b_{14} + a_{42} \times b_{24} + a_{43} \times b_{34} + a_{44} \times b_{44}$$

Le calcul d'un élément de la matrice C correspond à l'accumulation de 4 multiplications. De manière à pouvoir effectuer le calcul en 1 cycle d'horloge, toutes les multiplications sont parallélisées et l'accumulation par ligne est réalisée lorsque le logiciel démarre le calcul via le registre de contrôle.

3.1.2 Description des ports de l'IP

L'objectif était de faire un IP complètement paramétrable au niveau des types des éléments des matrices ainsi qu'au niveau de la dimension des matrices. Dans la première version (livrée ici), seul les matrices de dimension (4,4) - et dont les éléments sont des entiers 8 bits non signés - sont supportées.

Le calcul du nombre de bits N_c nécessaires pour coder les éléments de la matrice de sortie C est donné par:

$$N_c = ceil(\frac{ln(MAX)}{ln(2)}) = ceil(\frac{ln(4*255*255)}{ln(2)}) = 18$$

avec MAX qui correspond à la valeur maximale que peut prendre un élément de la matrice C, c'est à dire l'accumulation de 4 multiplications d'élément ayant une valeur maximale des matrices d'entrées A et B (entiers 8 bits non signés: 255). Les éléments de la matrice de sortie C sont donc des entiers 18 bits non signés.

Un package a été écrit de manière à déclarer des types et les composants. 3 types ont été définis:

- **mat_4x4_8bits**: tableau 2D de dimension (4,4) contenant des 8 bits non signés
- **mat_1x4_8bits**: tableau 1D de dimension (4) contenant des 8 bits non signés
- **mat_1x4_18bits**: tableau 1D de dimension (4) contenant des 18 bits non signés

Spécification des ports:

Signal	Direction	Type	Description
CLK_I	INPUT	STD_LOGIC	Horloge de fonctionnement du composant (validé à 250MHz).
RST_I	INPUT	STD_LOGIC	Reset actif HAUT.
INHIBIT_I	INPUT	STD_LOGIC	Inhibition du composant. L'accumulation n'est pas effectuée. Les registres de la matrice de sortie reste inchangés.
START_I	INPUT	STD_LOGIC	Démarre l'accumulation des sorties des multiplieurs.
MATA_ROWS_I	INPUT	MAT_4X4_8BITS	Eléments de la matrice d'entrée A.
MATB_COLS_I	INPUT	MAT_4X4_8BITS	Eléments de la matrice d'entrée B.
MATC_ROW0_O	OUTPUT	MAT_1X4_18BITS	Eléments de la 1ère ligne de la matrice de sortie C.
MATC_ROW1_O	OUTPUT	MAT_1X4_18BITS	Eléments de la 2ème ligne de la matrice de sortie C.
MATC_ROW2_O	OUTPUT	MAT_1X4_18BITS	Eléments de la 3ème ligne de la matrice de sortie C.
MATC_ROW3_O	OUTPUT	MAT_1X4_18BITS	Eléments de la 4ème ligne de la matrice de sortie C.
DONE_O	OUTPUT	STD_LOGIC	Accumulation terminée. Les registres de la matrice de sortie C sont valides.

3.1.3 Blocs diagramme

La figure 2 détaille l'architecture de l'IP:

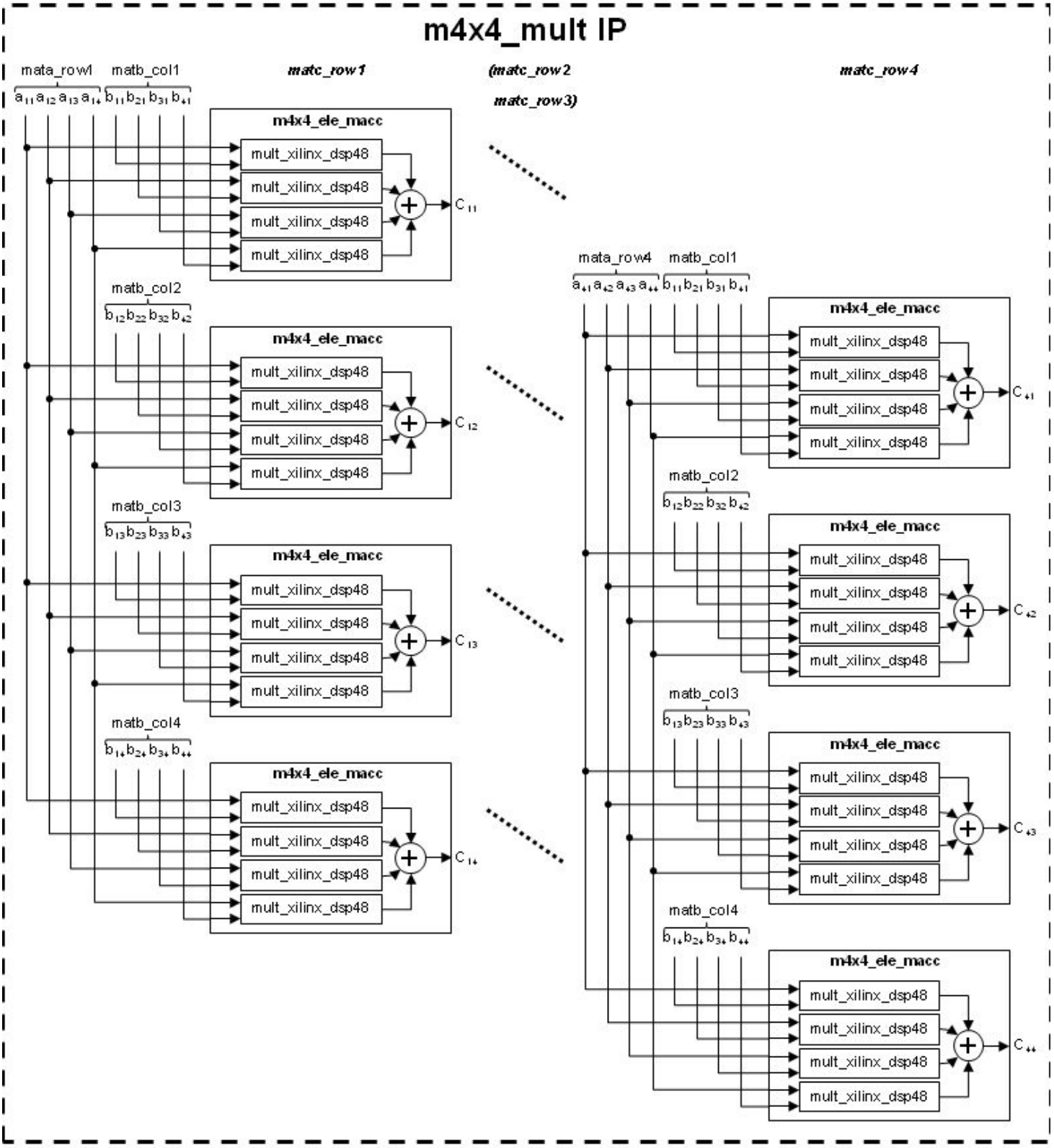


Figure 2: Architecture de l'IP du produit de matrice de dimension (4,4).

L'IP est donc constitué de 3 composants:

- **mult_xilinx_dsp48**: ce composant permet la multiplication de deux signaux de largeur paramétrable (maximum 48 bits). Un paramètre de type "string" permet de forcer l'utilisation (par inférence) des blocs DSP présent dans les FPGA Xilinx, afin d'économiser des ressources. Ce paramètre est exploitable seulement par les outils Xilinx. C'est pourquoi le nom du composant porte le champ "xilinx".
- **m4x4_ele_macc**: ce composant génère les 4 blocs multiplieurs et accumule leur résultat sur une commande de type monostable (non représenté sur le diagramme pour plus de visibilité). Ce composant permet donc le calcul d'un élément de la matrice de sortie.
- **m4x4_mult**: ce composant est le *module top* de la hiérarchie. Il permet de générer les 16 blocs m4x4_ele_macc qui calculent les valeurs des éléments. Il a une machine à états finis (FSM) de type *Mealy* qui contrôle le calcul. Le flowchart de la FSM est donné en figure 3.

Note: Suivant la fréquence de fonctionnement du circuit, le composant *mult_xilinx_dsp48* nécessite des *pipelines* pour tenir les contraintes de timing. Dans le cas d'une fréquence de fonctionnement de 250MHz, un pipeline est nécessaire (sur plateforme Zybo). La multiplication s'effectue donc en 2 coups d'horloge.

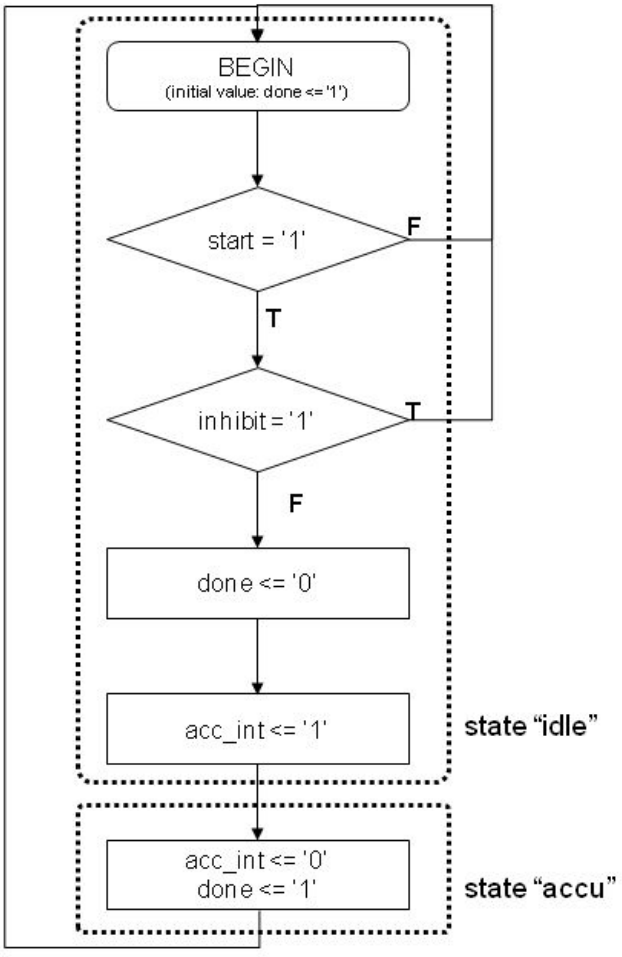


Figure 3: Flowchart de la FSM de l'IP

3.1.4 Intégration système

La figure 4 détaille l'architecture du système et permet de visualiser l'intégration de l'IP :

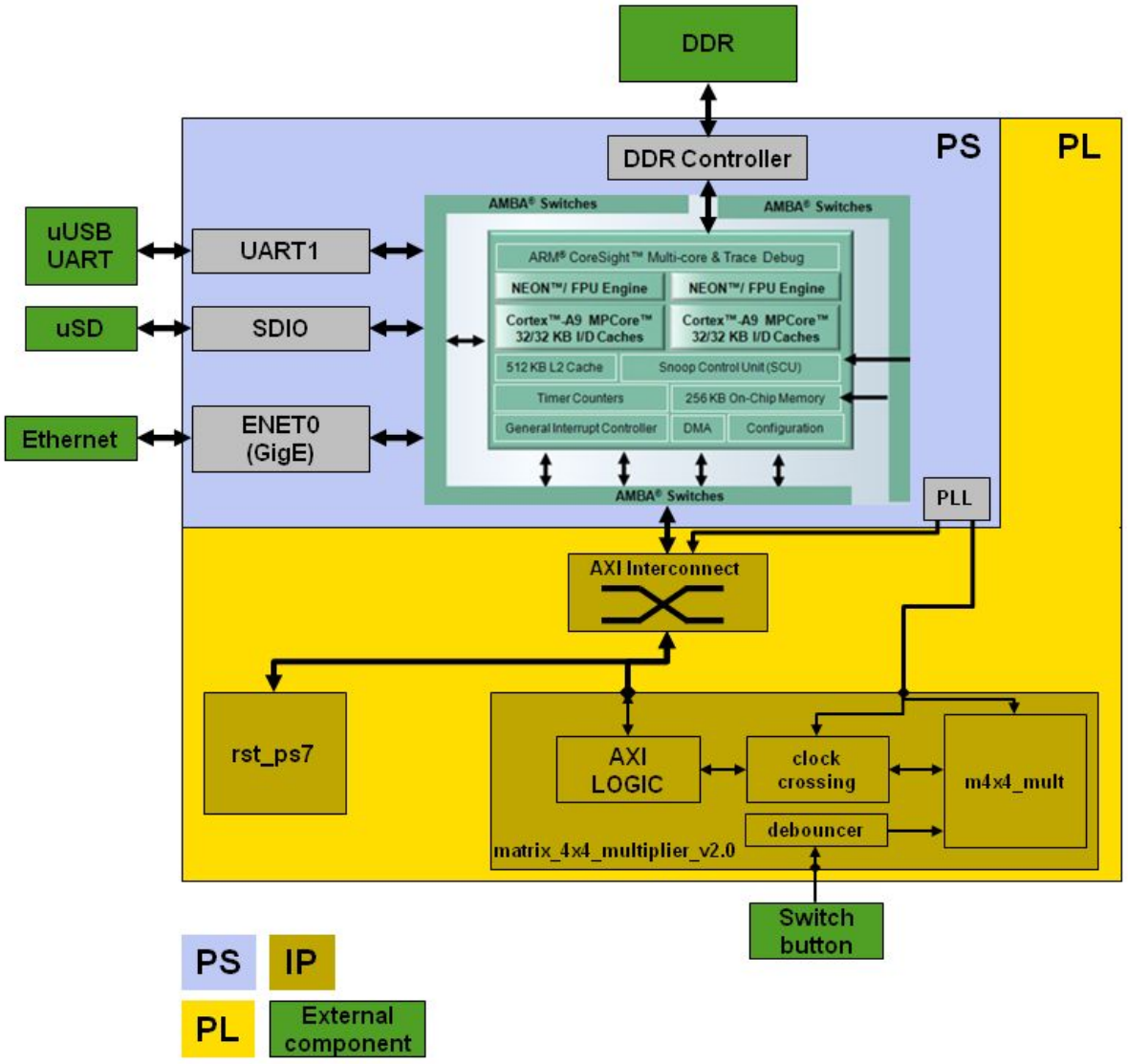


Figure 4: Architecture SoC et intégration du périphérique de produit matriciel (m4x4_mult).

Comme indiqué ci-dessus, le périphérique de produit matriciel est connecté au bus d'interconnexion AXI du système.
Le bus AXI est un bus de type "Master/Slave", c'est-à-dire qu'un "master", ici les processeurs, initie des accès en lecture ou en écriture sur le bus. Les périphériques sont les "slaves" qui *acceptent* ou *répondent* aux requêtes/accès du "master".
Lorsque le "master" initie un accès sur le bus à une adresse spécifique, un "crossbar" (ou "switch") entre le "master" et les "slaves" permet d'aiguiller la requête vers le slave en décodant l'adresse.

Dans la figure 4, on voit aussi un module *debouncer* dans l'IP. Son rôle consiste à éviter les rebonds liés à la mécanique des boutons poussoirs (ou de type switch). A noter aussi le bloc *clock crossing* qui permet de passer du domaine d'horloge du bus AXI (100MHz) au

domaine d'horloge du bloc *m4x4_mult* (250MHz) et vice-versa. Ce bloc contient des synchroniseurs classiques dans les changements de domaine d'horloge.

Enfin, le bloc *AXI LOGIC* contient la logique nécessaire au protocol AXI et permet de recevoir et envoyer les données des registres sur le bus.
La description de la "Register Map" est donnée dans la section 3.2 de l'architecture logicielle.

3.2 Architecture logicielle

3.2.1 Description

Le logiciel est décomposé en 2 parties :

- baremetal : logiciel exécuté directement sur un coeur ARM, sans OS. Cette partie est utilisée pour exécuter des tests unitaires afin de valider les fonctionnalités de l'IP.
- linux : application exécutée sur un linux embarqué exécuté sur les 2 coeurs ARM.

La figure 5 décrit la séquence de boot du Zynq :

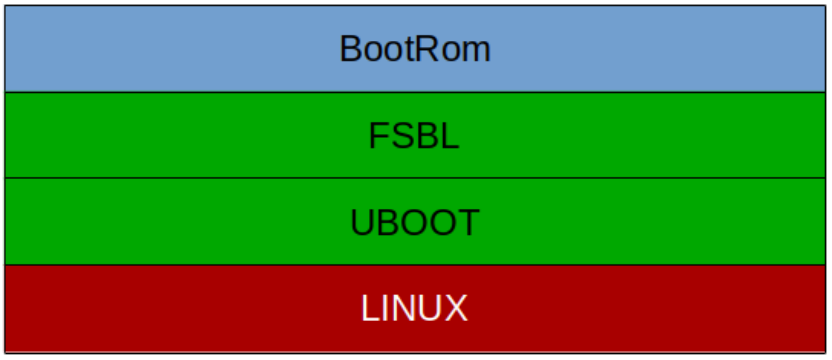


Figure 5: Zynq boot sequence.

Le Zynq a un programme qui s'exécute à la mise sous tension, le BootRom. Ce programme n'est pas modifiable par l'utilisateur. Il permet d'initialiser des périphériques de bases, des horloges et lit (via les pins dédiées, configurées avec des jumpers en général) le périphérique sur lequel booter (QSPI, SD, JTAG). Ensuite, ce programme scanne le périphérique afin de trouver le fichier BOOT.bin. Ce fichier contient:

- le **FSBL**: le **F**irst **S**tage **B**oot **L**oader est propre à l'architecture implantée dans le Zynq. Il dépend de l'application. Son rôle est d'initialiser tous les périphériques instanciés dans la PS du Zynq, ainsi que la DDR, la PLL, …
- le **bitstream** (optionnel): le FSBL peut aussi programmer le FPGA du Zynq (la PL).
- les **applications**: il s'agit des programmes à exécuter en baremetal sur les coeurs ARM. Dans notre cas, il s'agit du boot loader de second niveau, **u-boot**, qui charge Linux.

Dans certain cas, le kernel Linux et le ramdisk peuvent être contenu dans le BOOT.bin. Dans notre cas, ils sont contenu dans la carte SD et u-boot est chargé d'aller les lire.

Pour développer l'application, un BSP (Board Support Package) est généré par le SDK à partir de l'export du design réalisé sous Vivado.
Ce BSP contient, entre autres, les fichiers :

- *xparameters.h* : inclut la register map des périphériques AXI du design.
- matrix_4x4_multiplier.h : inclut l'interface de l'IP (offset des registres, accès mémoire, …)

Le FSBL est généré, comme le BSP, a l'aide du SDK. Il s'agit en quelque sorte d'un BSP spécifique.

3.2.2 Zynq Address Map

La figure 6 montre comment est arrangée l'espace d'adresse du Zynq:

Address Range	CPUs and ACP
0000_0000 to 0003_FFFF(2)	OCM
	DDR
	DDR
0004_0000 to 0007_FFFF	DDR
0008_0000 to 000F_FFFF	DDR
0010_0000 to 3FFF_FFFF	DDR
4000_0000 to 7FFF_FFFF	PL
8000_0000 to BFFF_FFFF	PL
E000_0000 to E02F_FFFF	IOP
E100_0000 to E5FF_FFFF	SMC
F800_0000 to F800_0BFF	SLCR
F800_1000 to F880_FFFF	PS
F890_0000 to F8F0_2FFF	CPU
FC00_0000 to FDFF_FFFF(4)	Quad-SPI
FFFC_0000 to FFFF_FFFF(2)	OCM

Figure 6: Carte de l'espace d'adresse du Zynq.
(Source: ug585-Zynq-7000-TRM.pdf)

On voit ici que le périphérique AXI instancié dans la PL aura une base adresse dans la gamme [0x4000_0000:0x8000_0000]. L'outils Vivado assigne automatiquement une base adresse lors de l'import d'un IP. Dans notre cas, l'adresse 0x43C0_0000 a été assignée, ce qui est bien dans la gamme dédiée au bus AXI Master Port 0 (M_AXI_GP0) gérant les périphériques de la PL.
La section suivante présente la *Register Map* du périphérique.

3.2.3 Register map

Base Address du périphérique: **0x43C00000**.
Le tableau suivant résume les registres utilisés pour interfacer l'IP. Les offsets sont donnés en base hexadécimal et correspondent à l'offset en octet utilisé par le logiciel. En effet, AXI étant un bus 32 bits dans notre cas, un registre correspond à 4 octets.

Offset	Name	Direction	Description
0x0	CSR	RW	Registre de contrôle et de status (Control and Status Register)
0x4	--	--	Reserved
0x8	MATA_E11	RW	Matrice d'entrée A: élément 1,1 (1ère ligne, 1ère colonne)
0xC	MATA_E12	RW	Matrice d'entrée A: élément 1,2 (1ère ligne, 2ème colonne)
0x10	MATA_E13	RW	Matrice d'entrée A: élément 1,3 (1ère ligne, 3ème colonne)
0x14	MATA_E14	RW	Matrice d'entrée A: élément 1,4 (1ere ligne, 4eme colonne)
0x18	MATA_E21	RW	Matrice d'entrée A: élément 2,1 (2eme ligne, 1ere colonne)
0x1C	MATA_E22	RW	Matrice d'entrée A: élément 2,2 (2eme ligne, 2eme colonne)
0x20	MATA_E23	RW	Matrice d'entrée A: élément 2,3 (2eme ligne, 3eme colonne)
0x24	MATA_E24	RW	Matrice d'entrée A: élément 2,4 (2eme ligne, 4eme colonne)
0x28	MATA_E31	RW	Matrice d'entrée A: élément 3,1 (3eme ligne, 1ere colonne)
0x2C	MATA_E32	RW	Matrice d'entrée A: élément 3,2 (3eme ligne, 2eme colonne)
0x30	MATA_E3	RW	Matrice d'entrée A: élément 3,3 (3eme ligne, 3eme colonne)

	3		
0x34	MATA_E34	RW	Matrice d'entrée A: élément 3,4 (3eme ligne, 4eme colonne)
0x38	MATA_E41	RW	Matrice d'entrée A: élément 4,1 (4eme ligne, 1ere colonne)
0x3C	MATA_E42	RW	Matrice d'entrée A: élément 4,2 (4eme ligne, 2eme colonne)
0x40	MATA_E43	RW	Matrice d'entrée A: élément 4,3 (4eme ligne, 3eme colonne)
0x44	MATA_E44	RW	Matrice d'entrée A: élément 4,4 (4eme ligne, 4eme colonne)
0x48	MATB_E11	RW	Matrice d'entrée B: élément 1,1 (1ere ligne, 1ere colonne)
0x4C	MATB_E21	RW	Matrice d'entrée B: élément 2,1 (2ème ligne, 1ere colonne)
0x50	MATB_E31	RW	Matrice d'entrée B: élément 3,1 (3eme ligne, 1ere colonne)
0x54	MATB_E41	RW	Matrice d'entrée B: élément 4,1 (4eme ligne, 1ere colonne)
0x58	MATB_E12	RW	Matrice d'entrée B: élément 1,2 (1ere ligne, 2eme colonne)
0x5C	MATB_E22	RW	Matrice d'entrée B: élément 2,2 (2eme ligne, 2eme colonne)
0x60	MATB_E32	RW	Matrice d'entrée B: élément 3,2 (3eme ligne, 2eme colonne)
0x64	MATB_E42	RW	Matrice d'entrée B: élément 4,2 (4eme ligne, 2eme colonne)
0x68	MATB_E31	RW	Matrice d'entrée B: élément 1,3 (1ere ligne, 3eme colonne)
0x6C	MATB_E32	RW	Matrice d'entrée B: élément 2,3 (2eme ligne, 3eme colonne)
0x70	MATB_E33	RW	Matrice d'entrée B: élément 3,3 (3eme ligne, 3eme colonne)
0x74	MATB_E34	RW	Matrice d'entrée B: élément 4,3 (4eme ligne, 3eme colonne)

0x78	MATB_E41	RW	Matrice d'entrée B: élément 1,4 (1ere ligne, 4eme colonne)
0x7C	MATB_E42	RW	Matrice d'entrée B: élément 2,4 (2eme ligne, 4eme colonne)
0x80	MATB_E43	RW	Matrice d'entrée B: élément 3,4 (3eme ligne, 4eme colonne)
0x84	MATB_E44	RW	Matrice d'entrée B: élément 4,4 (4eme ligne, 4eme colonne)
0x88	MATC_E11	RO	Matrice de sortie C: élément 1,1 (1ere ligne, 1ere colonne)
0x8C	MATC_E12	RO	Matrice de sortie C: élément 1,2 (1ere ligne, 2eme colonne)
0x90	MATC_E13	RO	Matrice de sortie C: élément 1,3 (1ere ligne, 3eme colonne)
0x94	MATC_E14	RO	Matrice de sortie C: élément 1,4 (1ere ligne, 4eme colonne)
0x98	MATC_E21	RO	Matrice de sortie C: élément 2,1 (2eme ligne, 1ere colonne)
0x9C	MATC_E22	RO	Matrice de sortie C: élément 2,2 (2eme ligne, 2eme colonne)
0xA0	MATC_E23	RO	Matrice de sortie C: élément 2,3 (2eme ligne, 3eme colonne)
0xA4	MATC_E24	RO	Matrice de sortie C: élément 2,4 (2eme ligne, 4eme colonne)
0xA8	MATC_E31	RO	Matrice de sortie C: élément 3,1 (3eme ligne, 1ere colonne)
0xAC	MATC_E32	RO	Matrice de sortie C: élément 3,2 (3eme ligne, 2eme colonne)
0xB0	MATC_E33	RO	Matrice de sortie C: élément 3,3 (3eme ligne, 3eme colonne)
0xB4	MATC_E34	RO	Matrice de sortie C: élément 3,4 (3eme ligne, 4eme colonne)
0xB8	MATC_E41	RO	Matrice de sortie C: élément 4,1 (4eme ligne, 1ere colonne)
0xBC	MATC_E42	RO	Matrice de sortie C: élément 4,2 (4eme ligne, 2eme colonne)
0xC0	MATC_E43	RO	Matrice de sortie C: élément 4,3 (4eme ligne, 3eme colonne)
0xC4	MATC_E44	RO	Matrice de sortie C: élément 4,4 (4eme ligne, 4eme colonne)

RW (Read/Write): le logiciel peut lire et écrire dans les registres.
RO (Read Only): le logiciel peut seulement lire dans les registres.

Le registre de contrôle (CSR) est détaillé en Annexe A.

3.2.4 Linux

Pour exécuter Linux sur la plateforme Zybo, 4 entités sont nécessaires :

- **u-boot.elf** : le boot loader permettant de lire la carte SD et charger Linux,
- **devicetree.dtb** : il permet de décrire les ressources hardware au kernel,
- **uImage** : c'est le kernel Linux,
- **uramdisk.image.gz** : il s'agit du *filesystem* racine (system rootfs) compresse et charge en RAM par le kernel. Dans l'embarque, cette méthode est préférable à un *filesystem* sur carte SD qui peut être corrompue en cas de coupure de courant par exemple.

Les fichiers devicetree.dtb et uramdisk.image.gz sont passés au kernel en parametres par le boot loader.
u-boot et le kernel sont cross-compilé pour ARM. Nous avons choisit les sources de Digilent:
u-boot : https://github.com/Digilent/u-boot-Digilent-Dev.git
Linux : https://github.com/DigilentInc/Linux-Digilent-Dev.git

Dans l'application Linux, le device */dev/mem* et la fonction *mmap()* permettent de mapper une adresse physique sur une adresse virtuelle. Ainsi, l'application peut accéder les registres du périphérique dans l'espace utilisateur, sans driver. Un driver serait nécessaire si un mécanisme d'interruption était mis en place dans l'IP.

4. Méthode de Conception et de Validation

Le SoC a été réalisé en co-développement. Après avoir spécifié le SoC et définit l'architecture de chaque partie, nous avons développé le hardware et le software en parallèle. Des itérations ont été nécessaires dues aux modifications et/ou problèmes rencontrés.
Le développement s'est déroulé selon les étapes suivantes:

- Spécification SoC
- Faisabilité linux embarqué
- Conception IP
- Conception logicielle
- Plan de tests
- Développement IP / Logiciel (en parallèle)
    - IP: dev. m4x4_mult ⇒ simulation ⇒ synthèse
    - IP: intégration m4x4_mult ⟷ AXI ⇒ synthèse
    - Logiciel: C baremetal tests unitaires
    - Logiciel: appli linux
- Intégration: Configuration Zynq PS + IP ⇒ synthèse ⇒ implémentation ⇒ bitstream
- Validation

4.1 Répartition des tâches

La répartition des tâches a été définie selon le tableau suivant:

Tâche	Domaine	Développeur	Description
Spécification SoC	SoC	MP, AB	Spécifier l'architecture du SoC et le choix du périphérique matériel.
Conception IP	SoC / HW	MP, AB	Définition de l'architecture de IP, interconnection AXI.
Conception logicielle	SW	MP, AB	Définition de l'architecture logicielle.
Développement IP	HW	AB	Développement VHDL de l'IP.
Linux embarqué	SW	AB	Configuration bootloader, kernel, devicetree.
Développement logiciel	SW	MP	Développement C bare-metal et application linux.
Plan de test	SoC	MP, AB	Définition du plan de tests
Intégration	SoC	MP, AB	Configuration du SoC sous Vivado et connection des IP. Synthèse ⇒ Implémentation ⇒ Contraintes timing OK? ⇒ Bitstream
Validation	SoC	MP, AB	Réalisation du plan de tests

Domaine:

- SoC: concerne le système dans sa globalité (HW+SW).
- HW: concerne le développement de l'IP
- SW: concerne le développement de la partie logiciel

4.2 Plan de tests

Le plan de tests a été définie selon le tableau suivant qui présente seulement les tests "haut niveau":

Numéro de Test	Fonctionnalité	Cible	Description / Mode opératoire
#1	Produit matriciel	IP Simulation	Simuler le comportement de l'IP effectuant le produit matriciel.
#2	Produit matriciel	IP Zybo / C baremetal	Vérifier le produit matriciel sur cible en chargeant le bitstream et en exécutant un test unitaire logiciel.
#3	Bouton inhibition	IP Zybo / C baremetal	Effectuer un produit matriciel. Relever la matrice de sortie. Glisser le bouton switch à l'état haut et effectuer un produit matriciel avec des matrices d'entrées différentes. La valeur de la matrice de sortie ne devrait pas avoir changée.
#4	Linux embarqué	Zybo	Insérer la carte uSD et vérifier que le linux embarqué boot.
#5	Produit matriciel	IP Zybo / linux embarqué	Booter linux et effectuer un produit matriciel.
#6	Application	Zybo	Booter linux et exécuter l'application. Mesurer les performances de produit matriciel: • sans le périphérique (en C dans l'appli) • avec le périphérique • avec NEON

4.3 Simulation - Testbench

Concernant l'IP, plusieurs niveau de simulation ont été réalisés en suivant la hiérarchie de l'IP:

- mult_xilinx_dsp48.vhd → tb_mult_xilinx_dsp48.vhd
- m4x4_ele_macc.vhd → tb_m4x4_ele_macc.vhd
- m4x4_mult.vhd → tb_m4x4_mult.vhd

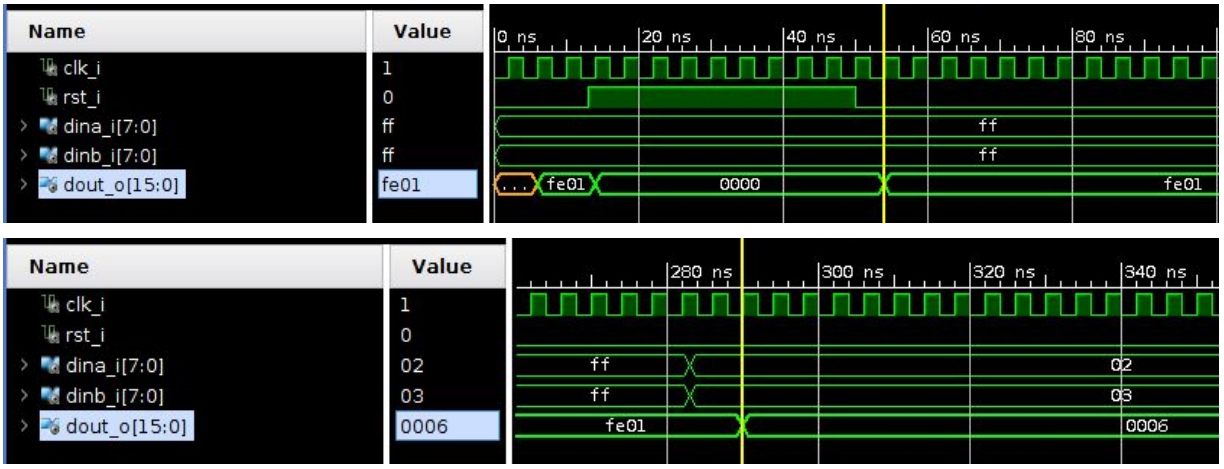
Le testbench du module "top" m4x4_mult utilise un fichier de stimuli contenant les matrices d'entrées A et B ainsi que la matrice de sortie C attendue en appelant les procédures *readline* et *read*. Il fournit les matrices d'entrées au DUT (Design Under Test: m4x4_mult.vhd) et compare la matrice de sortie du DUT à la matrice attendue. La comparaison utilise des ***assertions*** pour valider le DUT:

- assertion de vérification de la fonctionnalité *inhibition*
- assertion pour la vérification de la fonctionnalité *produit matriciel*

Le fichier de stimuli est aussi utilisé pour valider l'IP par l'appli linux. L'appli charge les matrices d'entrées, démarre le calcul, lit la matrice de sortie et compare à la matrice attendue.

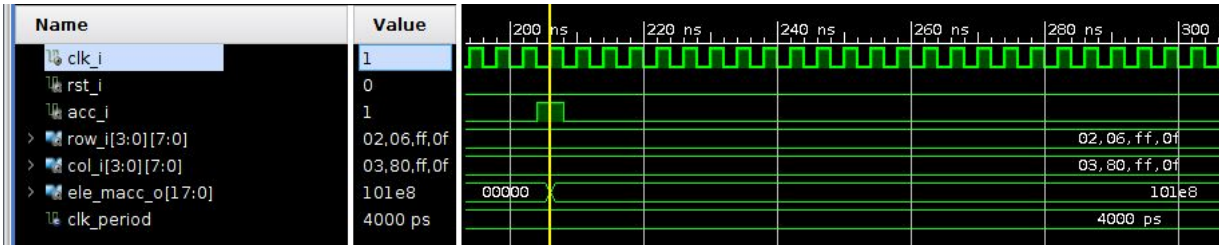
Résultats simulation :

mult_xilinx_dsp48



On observe bien la multiplication réalisée en 2 cycles d'horloge (due au pipeline pour respecter les contraintes de timing a 250MHz). Le module prend bien en charge la multiplication des valeurs maximale (255).

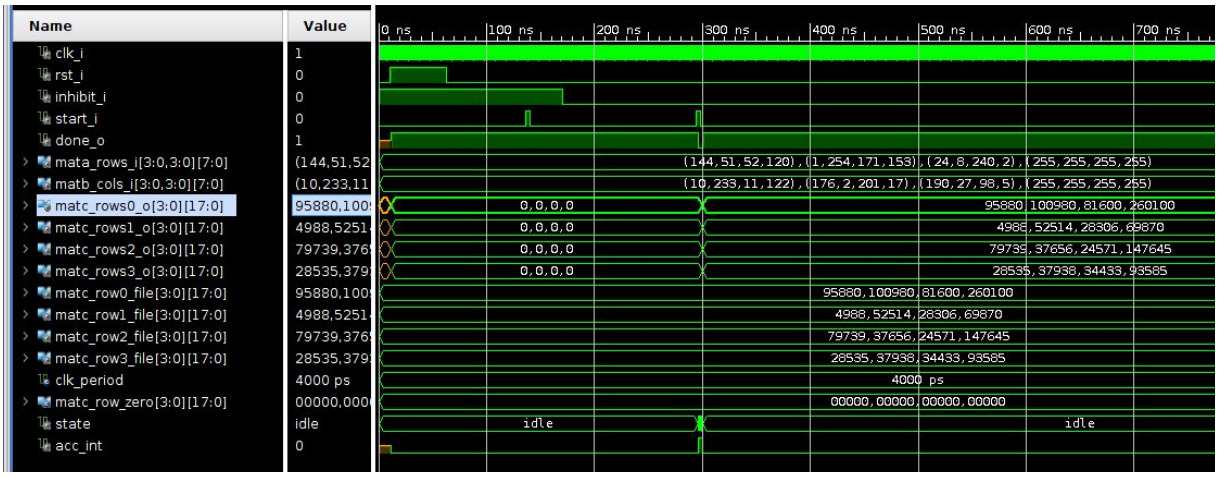
m4x4_ele_macc



$$0xF \times 0xF + 0xFF \times 0xFF + 0x6 \times 0x80 + 0x2 \times 0x3 = 0x101E8$$

OK

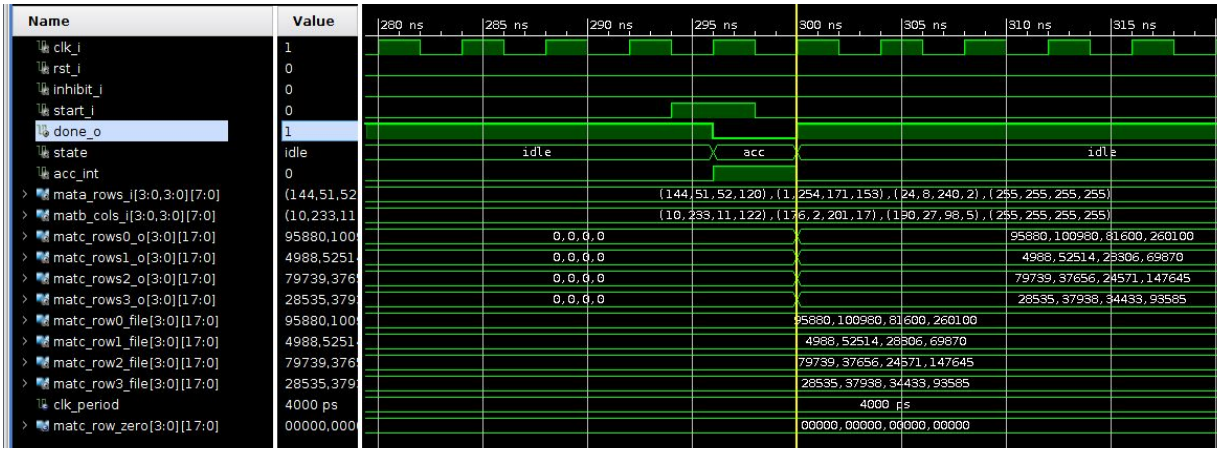
m4x4_mult



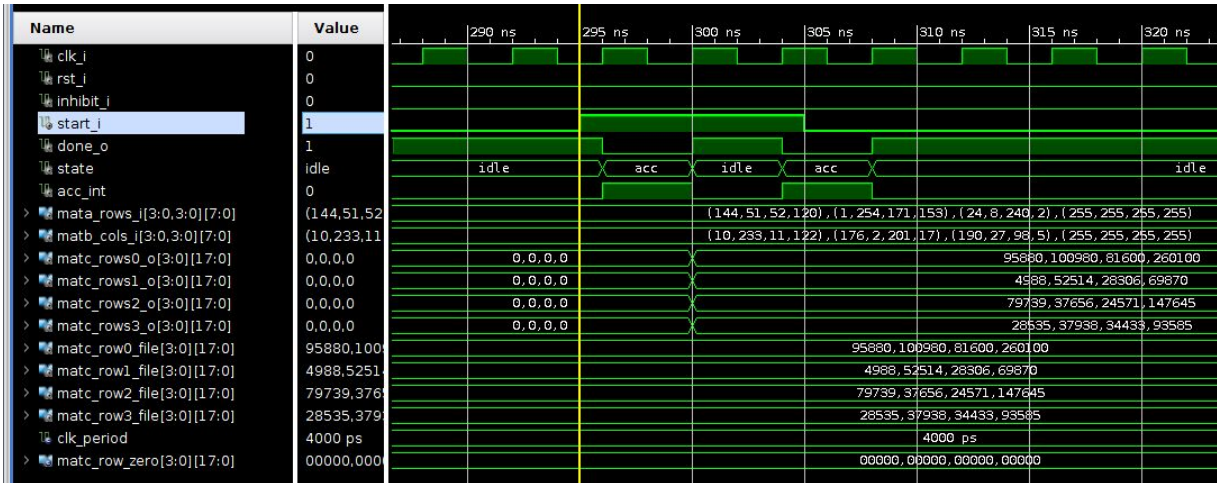
On voit sur cette capture les matrices d'entrées A et B (lues à partir du fichier de stimuli), la matrice de sortie C calculée par l'IP et la matrice de sortie C lue a partir du fichier de stimuli.

On observe bien qu'une requête de calcul (monostable START_I) n'a aucun effet sur la matrice de sortie si le signal INHIBIT_I est actif haut.

La capture suivante focalise sur la machine à état :



Sur la capture suivante, on observe que si le signal START_I dure N période, le produit est active N fois. On voit donc l'importance de générer un signal START_I monostable. Ceci est réalisé dans l'intégration AXI, au niveau du croisement de domaine d'horloge (100MHz vers 250MHz).



4.4 Résultats intégration - Validation

4.4.1 Intégration IP

L'intégration de l'IP s'est réalisée en plusieurs étapes sous Vivado :

- Création d'un périphérique AXI-Lite 50 registres (1 CSR + 1 Reserved + 3 matrices 4,4).
- Intégration AXI : modification logique AXI (template Xilinx à la création de l'IP), ajout croisement domaine d'horloge, mapping registres AXI vers l'IP.
- Synthèse de l'IP seul pour vérification et contrainte de timing.
- Création design SoC : ajout IP Zynq PS7 et IP créé, puis connection AXI automatique Vivado.
- Synthèse, implémentation.
- Vérification contrainte de timing.
- Génération du bitstream.

4.4.2 Rapport ressources et contraintes de timing

Comme indiqué ci-dessus, l'IP développé pour le produit matriciel permet d'utiliser les blocs DSP présents dans le FPGA Xilinx. Il est intéressant de regarder le rapport des ressources utilisées avec et sans les blocs DSP:

Hierarchy

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	LUT Flip Flop Pairs (17600)	DSPs (80)	Bonded IOB (100)	Bonded IOPADs (130)	BUFGCTRL (32)
soc_wrapper	1619	3269	102	51	974	1556	63	560	64	1	130	2
soc_i (soc)	1619	3269	102	51	974	1556	63	560	64	0	0	2

Rapport utilisation ressources en utilisant les blocs DSP

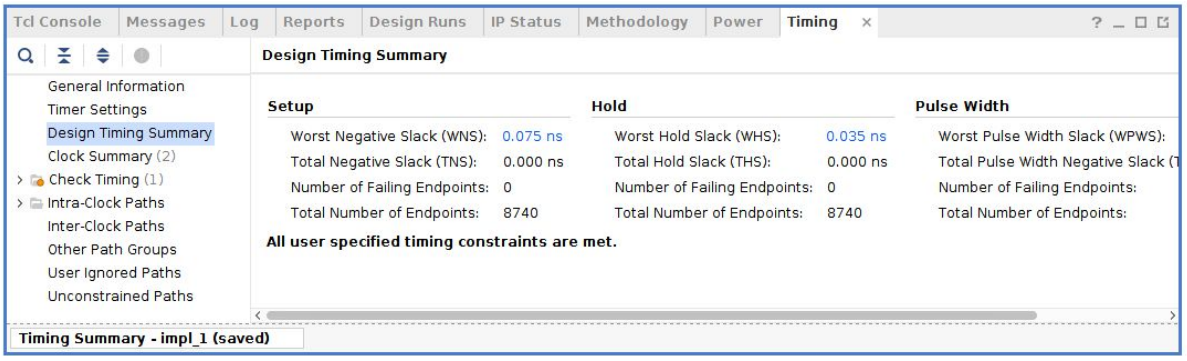
Hierarchy

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	LUT Flip Flop Pairs (17600)	Bonded IOB (100)	Bonded IOPADs (130)	BUFGCTRL (32)
soc_wrapper	6221	5573	102	51	2296	6158	63	1692	1	130	2
soc_i (soc)	6221	5573	102	51	2296	6158	63	1692	0	0	2

Rapport utilisation ressources sans blocs DSP

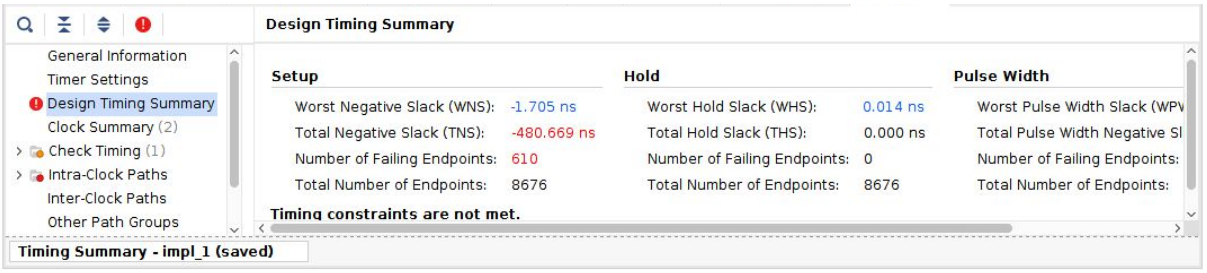
On voit bien l'économie réalisée sur les ressources (facteur ~4 sur les Slice LUTs). Cela serait important si, par exemple, d'autres périphériques devaient être ajoutés.
Note : il est indiqué dans la datasheet du Zynq implanté sur la carte Zybo qu'il y a 80 blocs DSP disponible dans le FPGA. Un produit de 2 matrices (4,4) nécessite 64 multiplications. On remarque que l'on est presque à la limite et que sur des dimensions supérieures, il ne serait plus possible d'utiliser les blocs DSP.

Contraintes de timing :



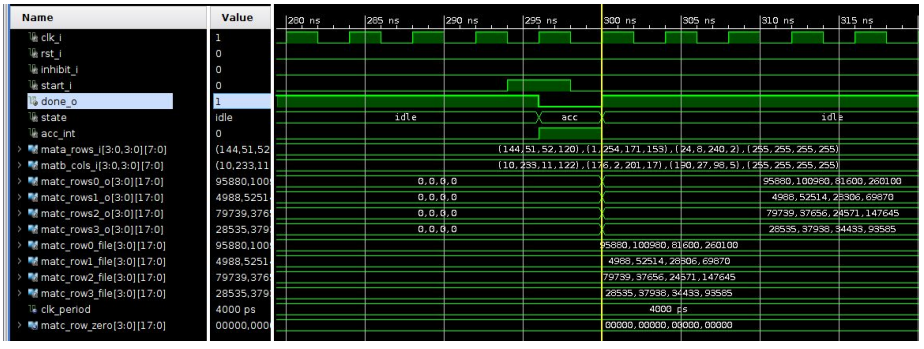
Pour que les contraintes de timing soient respectées sur les 2 domaines d'horloge (AXI 100MHz et IP 250MHz), il faut ajouter une contrainte spécifiant que les 2 horloges sont asynchrones:
set_clock_groups -asynchronous -group [get_clocks clk_fpga_0] -group [get_clocks clk_fpga_1]

On peut remarquer, comme l'indique la capture ci-dessous, que les contraintes de timing ne sont pas respectées si un pipeline n'est pas ajoutée au niveau des multiplieurs:



4.4.3 Validation

La validation a été réalisée en effectuant le plan de tests. Les résultats sont résumés dans le tableau suivant:

Test	Résultat	Commentaires
#1	PASSED	
#2	PASSED	--
#3	PASSED	--
#4	PASSED	--

#5	PASSED	--
#6	PASSED	--

5. Conclusion / Perspectives

Résultats

Le design complet du SoC a été livré. Le programme Linux sur cible Zybo interagit avec l'IP et permet de réaliser des produits de matrices (4,4). L'IP réalisant ce calcul en 1 cycle d'horloge à 250MHz, le logiciel lit directement la matrice de sortie après avoir démarré le calcul. En effet, faire du *polling* sur le bit *done* du registre de contrôle fait perdre plus de temps.
Dans le cadre du TP NEON du module CE514, il était demandé de réaliser un calcul de produit de matrice de dimension (4,4). Nous avons mesuré un temps d'exécution (à partir du Global Timer cadencé à 333MHz) de 150 ticks. On voit donc bien l'intérêt d'utiliser un périphérique matériel pour gagner en temps d'exécution. Par contre, l'IP est dédié à ce calcul alors que NEON peut exécuter d'autres calculs programmés par l'utilisateur. Enfin, le gain de performance est limité, dans notre cas, au temps perdu sur les accès mémoires du bus AXI.

Problèmes rencontrés

Domaine d'horloge : pour pouvoir valider les contraintes de timing, il a été nécessaire d'ajouter une contrainte spécifiant que les 2 horloges sont asynchrones. Ce point nous a fait perdre du temps.
Bit de start monostable : ce monostable était mal géré au début. En effet, il n'était pas remis à zéro (self-clearing) correctement cote IP.

Méthode de conception et développement

Au cours de ce développement, nous avons pu mettre en oeuvre plusieurs compétences:

- Développement VHDL : l'IP a été développé de zéro. Il a fallu réfléchir à l'architecture de l'IP de manière à paralléliser les multiplications.
- Simulation et validation : simuler l'IP et le valider (utilisation direct du cours de spécialité de M.Beroulle) et prévoir un plan de test.
- Intégration système : mise en oeuvre de l'outils Vivado pour l'intégration d'IP et le design d'un SoC.

Nous avons ainsi pu remarquer la rapidité de prototypage de design SoC avec le Zynq et l'efficacité du co-développement. En effet, une fois l'IP simulé et le logiciel développé en parallèle, la mise en oeuvre est relativement rapide. Le point délicat se situe au niveau de l'intégration au bus d'interconnexion AXI.

Perspectives

Quelques points pourraient être améliorés:

- passer sur un périphérique AXI-Stream pour envoyer des "burst" de données et donc être moins limité en performance par les accès mémoires.
- changer le bit *done* dans le registre de contrôle en une interruption. Ainsi le logiciel démarre le calcul et attrape une interruption lorsque le calcul est terminé.

Annexe A - Description détaillée du CSR

Base Address du périphérique: **0x43C00000**

CSR: 0x00

Bits	Name	Description
[31:2]	-	Unused
[4]	-	Reserved
[3]	-	Reserved
[2]	LCK	Bit **LoCK**ed. *Not yet implemented*
[1]	DNE	Bit **DoNE**. Quand ce bit est à 1, le calcul est terminé. Les valeurs contenues dans les registres de la matrice de sortie C sont valides. Ce bit est automatiquement mis à 0 quand le bit STT [0] est mis à 1.
[0]	STT	Bit **STarT**. Quand ce bit est mis à 1, le calcul démarre. Les registres des éléments des matrices d'entrées A et B doivent être valides. Ce bit est self-clearing (automatiquement mis à 0).