

Optimisation matérielle et logicielle d'une application de multiplication de matrices sur carte de développement PYNQ-Z2

2020

EN-319 Conception Conjointe sur FPGA

SEE 2^{ème} année



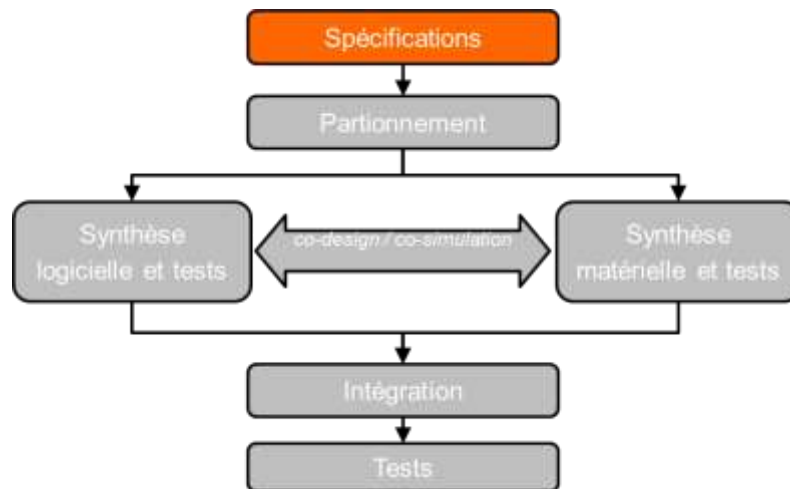
Enseignant : Jérémie Crenne & Benjamin Lux

jcrenne@enseirb-matmeca.fr
belux@enseirb-matmeca.fr

Contenu

0	Introduction.....	4
1	Génération de système 0	6
1.1	Instantiation du matériel.....	6
1.2	Codage du logiciel et tests.....	18
2	Rappels sur la multiplication de matrices	21
2.1	La multiplication de matrices 6x6.....	21
3	Edition du code, créations des fonctions de multiplication	23
3.1	Création de test3	23
3.2	Création de test4	24
3.3	Complétez le compte-rendu.....	25
4	Génération de système 1	26
4.1	Modification du système 0 sous Vivado.....	26
4.2	Exécution sur la cible système 0.....	26

0 Introduction



Problématique

« Ca rame ! Que peut-on y faire ? »

Vous êtes consultant en systèmes électroniques, et votre client, spécialisé dans le traitement vidéo embarqué, vous appelle à la rescousse car, après avoir développé son produit, il se rend compte qu'il n'a pas les performances requises !

Son application passe 80% du temps à réaliser des multiplications de matrices 6x6 pour réaliser un traitement d'images. Il a besoin que vous interveniez pour que cette opération soit au moins 5 fois plus rapide. Si vous arrivez à faire encore mieux, votre bonus n'en sera que meilleur.

Afin d'atteindre vos objectifs, vous doterez le système de ressources matérielles dédiées (multiplieur, puis multiplieur de matrices) qui pourront être exploitées par le logiciel pour accélérer les traitements. La cible est imposée : la carte de développement PYNQ-Z2

Vous devez utiliser la suite Vivado 2018.2. L'appel aux fonctions de multiplication doit être réalisé depuis du code C exécuté sur le processeur ARM. Vous pouvez utiliser toutes les ressources disponibles du FPGA. Vous pourrez intervenir sur le logiciel et sur le matériel. Vous ne pouvez pas modifier les options de compilation du logiciel.

Afin d'éviter des problèmes très coûteux en temps, vous devez être très scrupuleux concernant l'organisation de vos fichiers :

- Nommez le répertoire local de travail « Codesign_projectfiles », des références à ce répertoire seront faites tout au long du projet
- Pas d'espaces dans le chemin de dossier dans lequel vous travaillez (bugs Xilinx)
- **Sauvegardez l'ensemble de vos fichiers à la fin de chaque séance, et restaurez-les au même emplacement en début de séance suivante.** Pour cela, prenez note de votre répertoire de travail

Méthodologie

Le projet suivra les phases suivantes :

- Génération du système 0 :
Cette étape vous permettra de préparer l'environnement du client et de vous familiariser avec les outils.
- Edition du code C, création de fonctions de multiplication :
Vous éditez le logiciel afin d'implémenter la fonction de multiplication de matrices et mesurer son temps d'exécution.
- Génération du système 1 :
Il s'agit d'améliorer le système 0 en augmentant la fréquence du processeur ARM et en optimisant les options du compilateur, afin d'améliorer les performances sur l'opération multiplication de matrice.
- Génération du système 2 :
Cette étape consiste en la création d'une IP dédiée à la multiplication de matrices : un accélérateur pour le processeur ARM que vous concevrez et implémenterez en VHDL. Cette étape est découpée en 2 sous phases :
 - Codage matériel du multiplieur de matrices et simulation de celui-ci.
 - Intégration et test sur cible.

Documentation

Les documents nécessaires au projet sont téléchargeables à l'adresse suivante : www.jeremiecrenne.com/enseirb/EN319. Des documents utiles sont disponibles dans le répertoire « package_doc », et les fichiers sources dont vous aurez besoin sont dans le répertoire « ressources ».

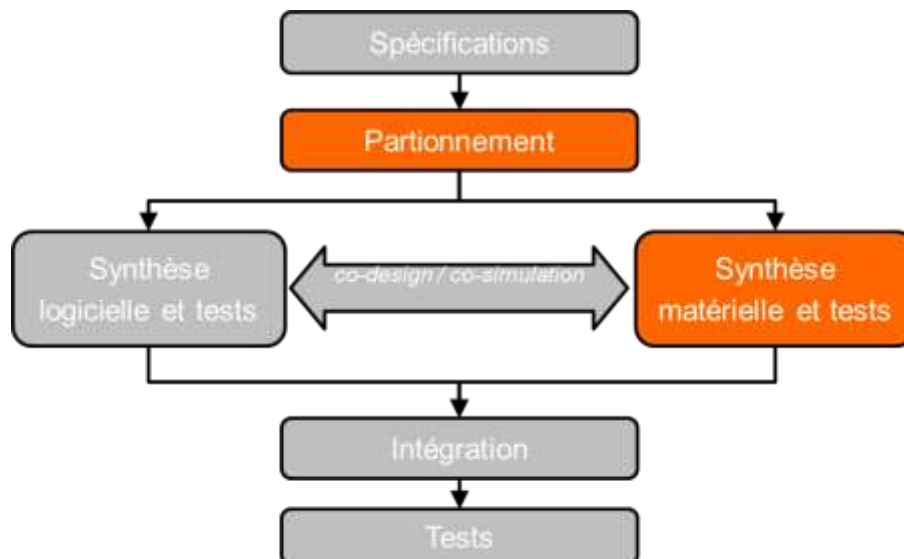
Compte rendu

- A envoyer à jcrenne@enserib-matmeca.fr et belux@enseirb-matmeca.fr
- Au format .pdf (pour le sujet) et fichiers sources à transmettre dans un zip

1 Génération de système 0

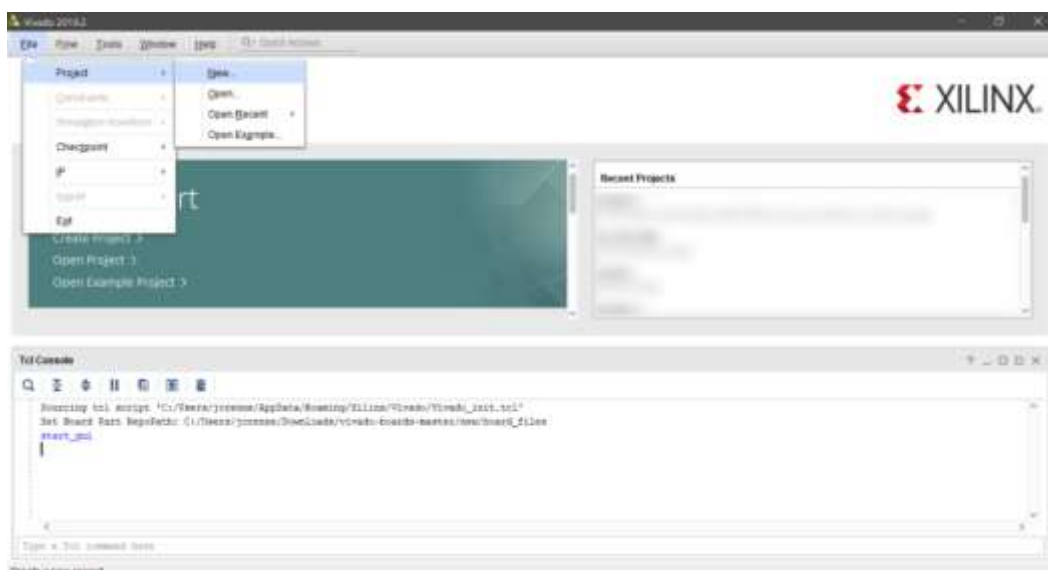
Suivez les étapes de ce tutoriel qui vous permettra de recréer le système à améliorer, il s'agit de l'instantiation du SoC Zynq qui comprend entre autres un processeur ARM double cœur, un timer et une UART. Pour le système 0, le processeur ARM sera sous-cadencé à une fréquence de 50 MHz

1.1 Instantiation du matériel



1.1.1 Création du projet Vivado

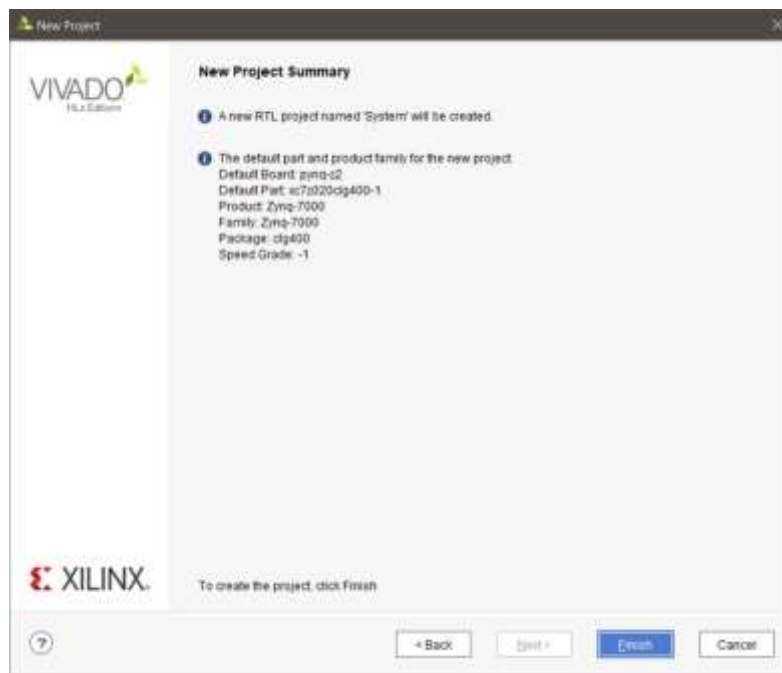
1. Créez un répertoire de travail sur votre session : [...]\Codesign_projectfiles\
2. Démarrez Vivado 2018.2
3. Dans la fenêtre de bienvenue, sélectionnez "Create New Project" :



4. Avancez dans l'assistant en choisissant les paramètres suivants :

Project name	System
Project location	[...]\Codesign_projectfiles
Create project subdirectory	Yes
Project Type	RTL Project "Do not specify sources at this time": check
Default Part	Boards → Pynq-Z2

5. Vérifiez que les paramètres suivants s'affichent à la fin de l'assistant :

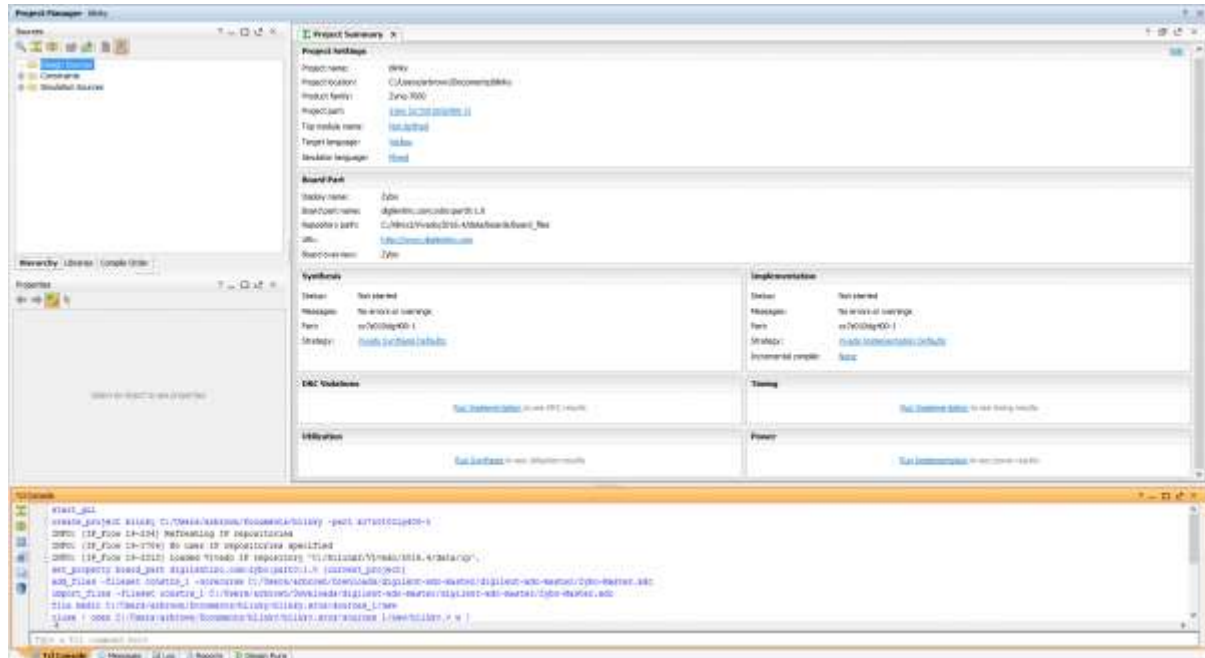


6. Cliquez sur "Finish"
7. Le projet est créé. Analysez chacune des lignes du bandeau de gauche "Flow Navigator" afin de comprendre leur utilité
8. Cliquez sur « Project Settings » dans le panneau « Flow Navigator » et vérifiez que le « Target Language » sélectionné est « VHDL »
9. Dans le « Flow Navigator », cliquez sur « Add Sources » → « Add or create constraints ». Sélectionner le fichier « pynq-z2_v1.0.xdc » du répertoire « ressources/pynq_z2_files/digilent-xdc/ » qui vous a été fourni. Veillez à cocher « Copy constraints files into project »
10. Cliquez sur Finish

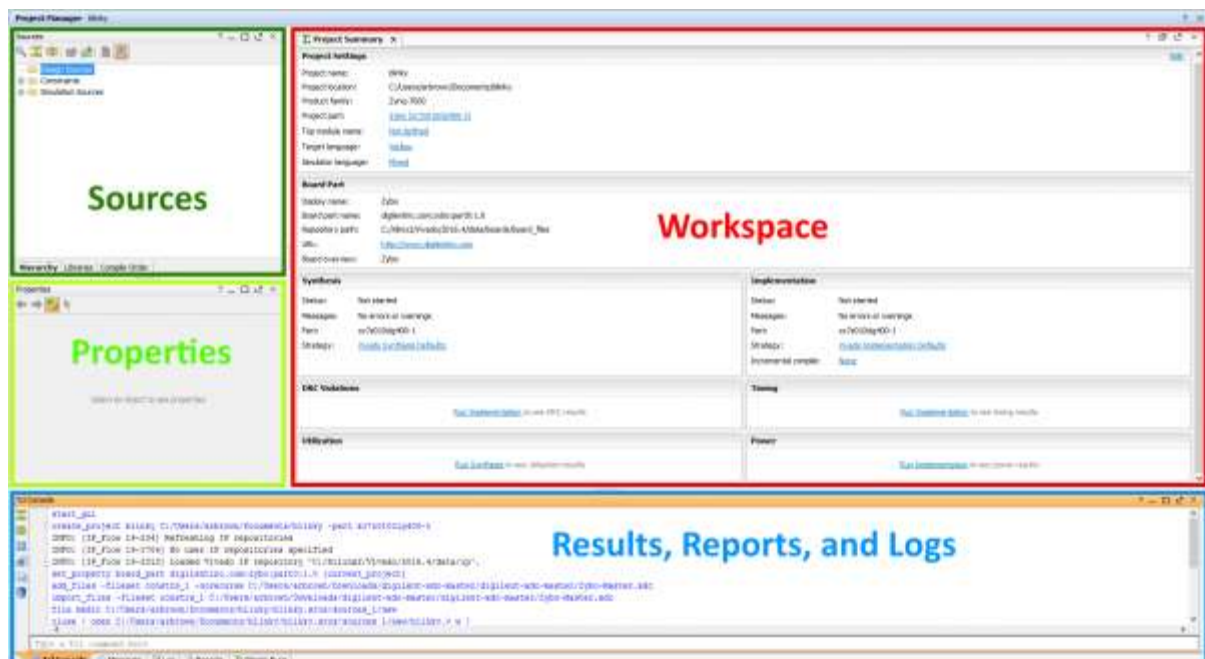
1.1.2 Description de l'environnement de travail VIVADO.

Ce chapitre présente l'organisation de l'environnement de travail VIVADO. Il est en anglais, et est issu de https://reference.digilentinc.com/vivado/getting_started/start


This tool is where most development will occur and is the initial tool open after creating a new project.

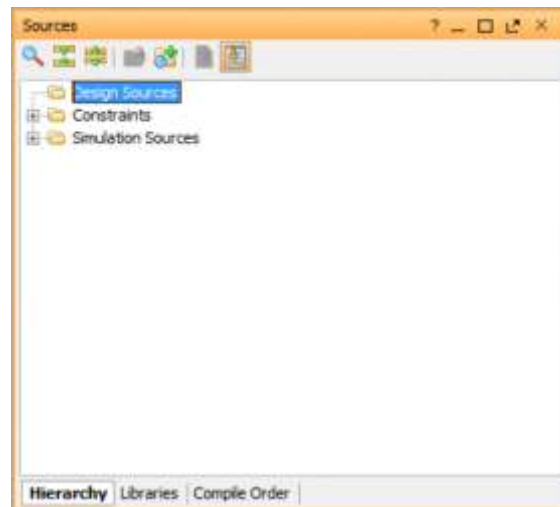


The Project Manager consists of four panes, Sources, Properties, Results, and the Workspace.

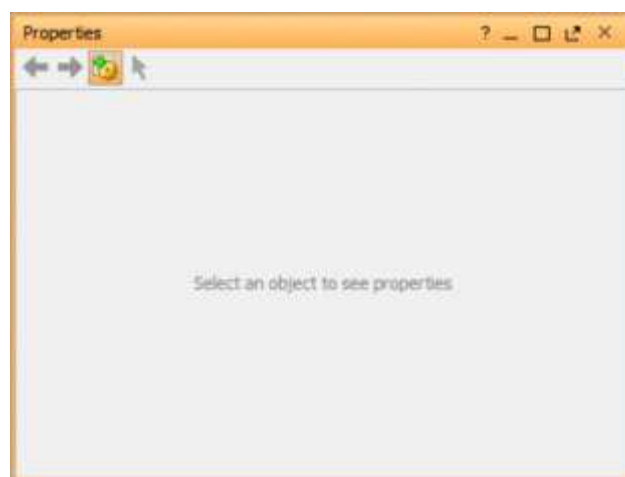


The Sources pane contains the project hierarchy and is used for opening up files. The folder structure is organized such that the HDL files are kept under the *Design Sources* folder, constraints are kept under the *Constraints* folder, and simulation files are kept under the *Simulation Sources* folder. Files can be opened in the Workspace by double-clicking on the corresponding entry in the Sources pane.

Sources can also be added by either right clicking the folder to add the file to and selecting *Add Sources* or by clicking the *Add Sources* button ().



The *Properties* pane allows for viewing and editing of file properties. When a file is selected in the Sources pane its properties are shown in here. This pane can usually be ignored.



The unnamed pane at the bottom of the Project Manager window consists of several different useful tools for debugging a project. The most important one to know is the *Messages* tool. This tool parses the Tcl console for errors, warnings, and other important information and displays it in an informative way. These tools can be accessed by selecting the different tabs at the bottom of this pane.



The *Tcl Console* is a tool that allows for running commands directly without the use of the main user interface. Some messages may link to the Tcl Console to provide more information regarding an error.

```

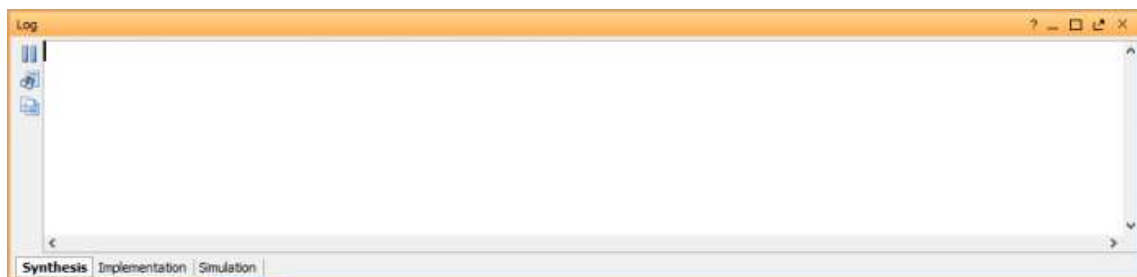
start_gui
create_project blinky C:/Users/arbrown/Documents/blinky -part xc7z010clg400-1
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx2/Vivado/2016.4/data/ip'.
set_property board_part digilentinc.com:zybo:part0:1.0 [current_project]
add_files -fileset constrs_1 -norecurse C:/Users/arbrown/Downloads/digilent-xdc-master/digilent-xdc-master/Zybo-Master.xdc
import_files -fileset constrs_1 C:/Users/arbrown/Downloads/digilent-xdc-master/digilent-xdc-master/Zybo-Master.xdc
file mkdir C:/Users/arbrown/Documents/blinky/blinky.srcs/sources_1/new
close ! open C:/Users/arbrown/Documents/blinky/blinky.srcs/sources_1/new/blinky.v w !
Type a Tcl command here

```

The *Reports* tool is useful for quickly jumping to any one of the many reports that Vivado generates on a design. These reports include power, timing, and utilization just to name a few.



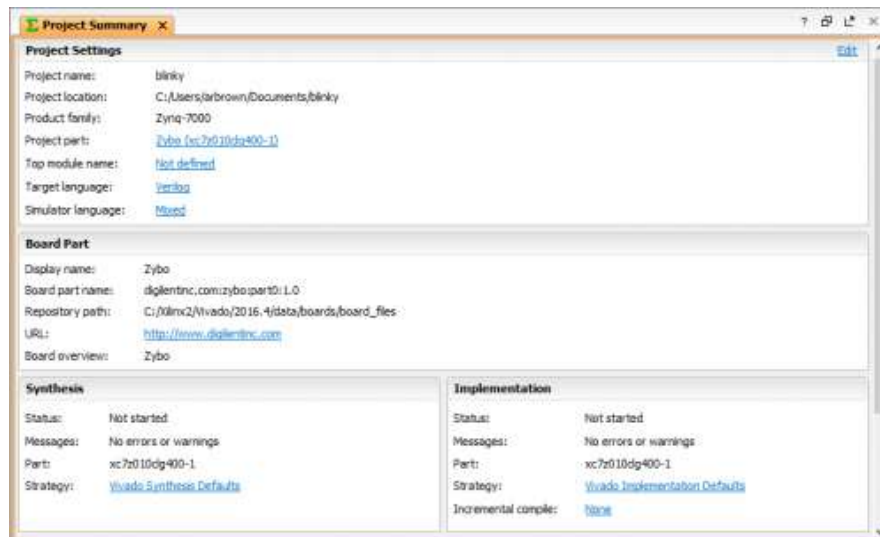
The *Log* displays the output from the latest Synthesis, Implementation, and Simulation runs. Digging into this is usually not necessary as the reports and messages view store the information in the log in a more readable format.



The last tool is the *Design Runs*. Using this tool run settings can be edited and new runs can be created. This tool is useful when targeting multiple devices with the same design.


Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	PCIE %	S
synth_1	constrs_1	Not started													
impl_1	constrs_1	Not started													

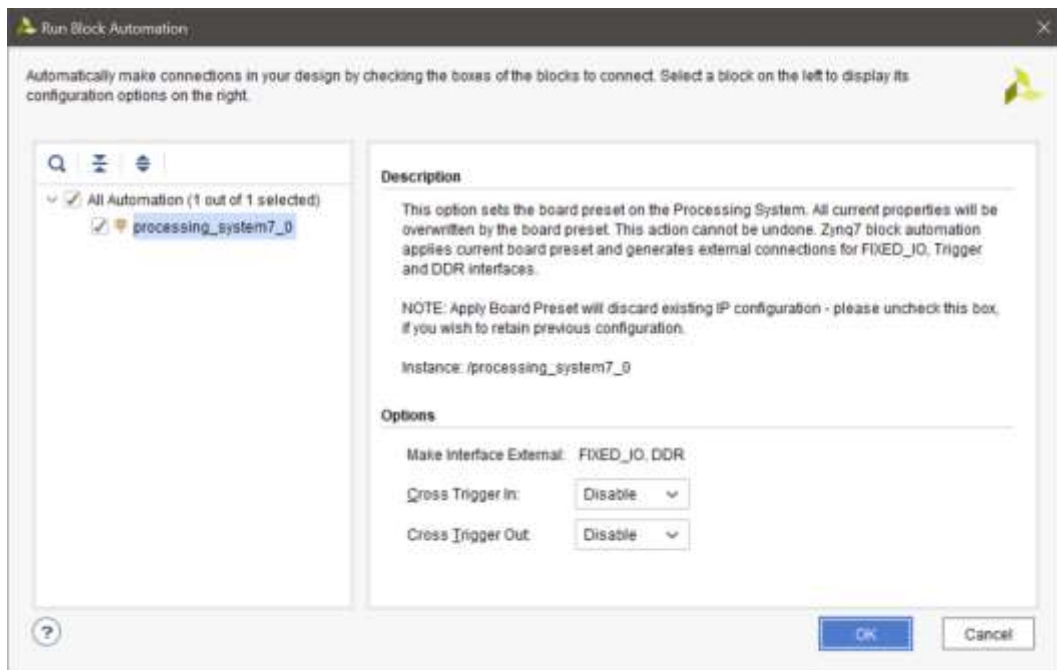
The most important pane in the Project Manager is the *Workspace*. The Workspace is where reports are opened for viewing and HDL/constraints files are opened for editing. Initially the Workspace displays the *Project Summary* which show some basic information from some of the reports.



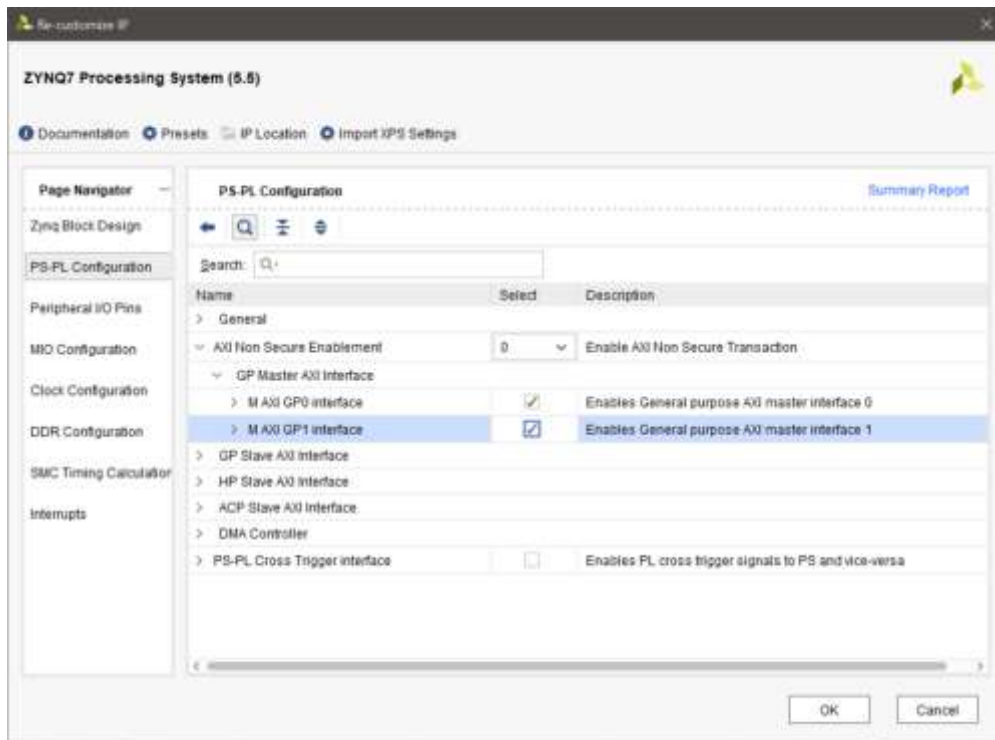
1.1.3 Instantiation du SoC Zynq processing system (PS)

Vivado permet d'instancier des composants matériels sous forme graphique, c'est ce que nous allons faire ici.

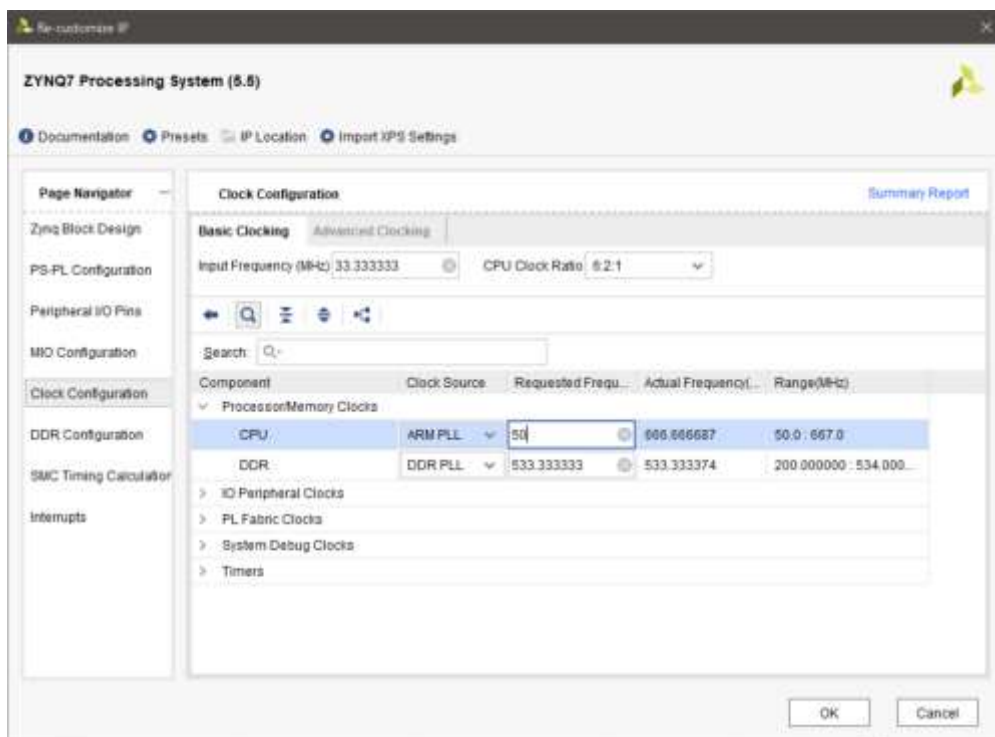
1. Dans le panneau de gauche "Flow Navigator", cliquer sur "Create Block Design". Laissez les champs par défaut, puis cliquer sur OK
2. Dans l'onglet « Diagram » qui vient d'être créé, cliquez sur l'icône  « Add IP ». Cherchez par la suite « ZYNQ7 Processing System » et double cliquez pour ajouter cette IP à votre design
3. Un bandeau vert apparaît en haut du diagramme, sélectionner « Run Block Automation » afin que Vivado propose un câblage du SoC Zynq. Sélectionnez les paramètres suivants :



4. Rebouclez l'horloge « FCLK_CLK0 » en sortie du bloc « ZYNQ7 Processing System » sur l'entrée « M_AXI_GP0_ACLK »
5. Configurez le « ZYNQ7 Processing System » en faisant un clic droit dessus, et en sélectionnant « Customize Block » ou en double cliquant sur le bloc
6. En provision, activez la seconde interface AXI maître « M AXI GP1 Interface »

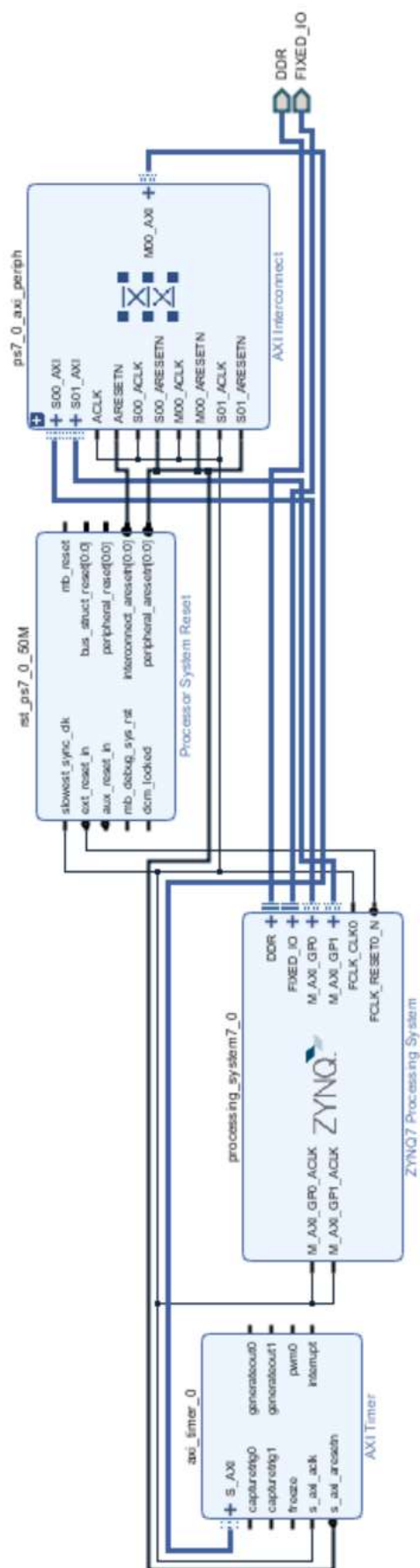



- Assurez-vous de configurer l'horloge du processeur ARM à 50 MHz comme ci-dessous. **Attention : conserver la fréquence d'entrée telle quelle est et de ne pas la changer !**



- Ajoutez maintenant une IP « AXI Timer »
- Cliquez à présent sur le bandeau « Run Connection Automation ». Sélectionnez ensuite toutes les cases disponibles à la coche. **Attention, le bandeau peut apparaître plusieurs fois !**

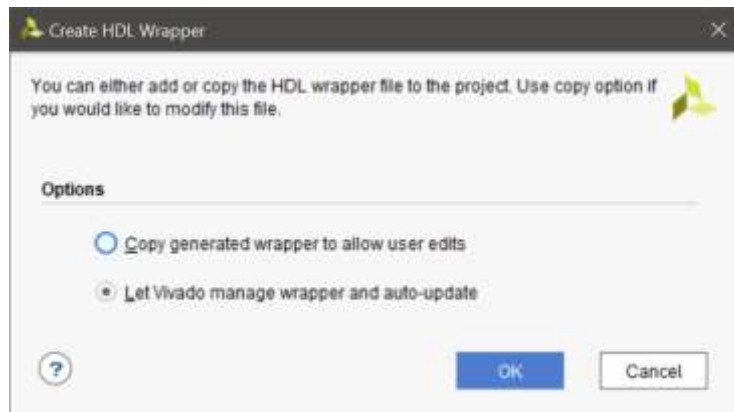
10. Constatez que vous pouvez déplacer les IP afin de présenter le diagramme de manière optimale. A cette étape, les connections de votre design doivent correspondre à cela :



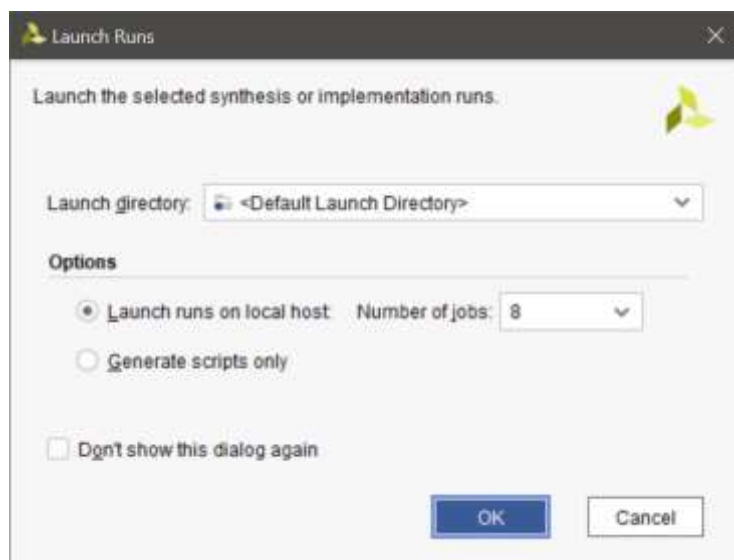
11. Cliquez à présent sur l'icône  « Validate Design » afin de vous assurer qu'il n'y a pas d'incohérence ni de manques dans les connections faites

1.1.4 Génération du bitstream

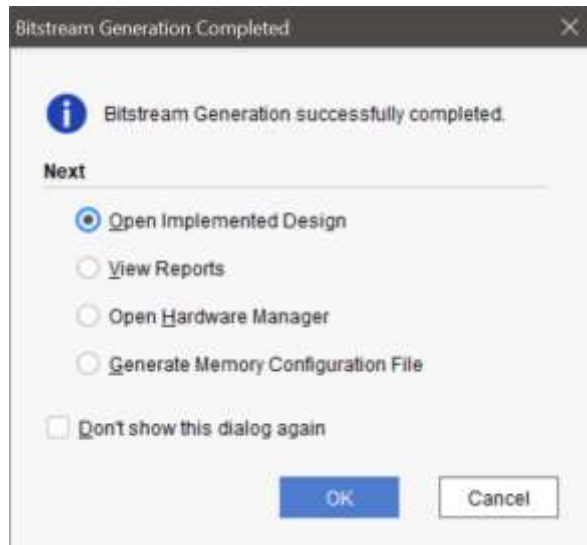
1. Dans le panneau « sources », clic droit sur l'objet « design_1 » et sélectionner « Create HDL wrapper », et choisir l'option « Let Vivado manage wrapper and auto-update » :



2. Vivado crée un fichier « .vhd » qui apparait maintenant dans les « design sources »
3. Vous pouvez à présent lancer les étapes de synthèse de matériel. Pour cela, cliquez directement sur « Generate bitstream » dans le panneau « Flow Navigator » pour dérouler l'ensemble du processus de synthèse



4. Le processus doit se terminer par :



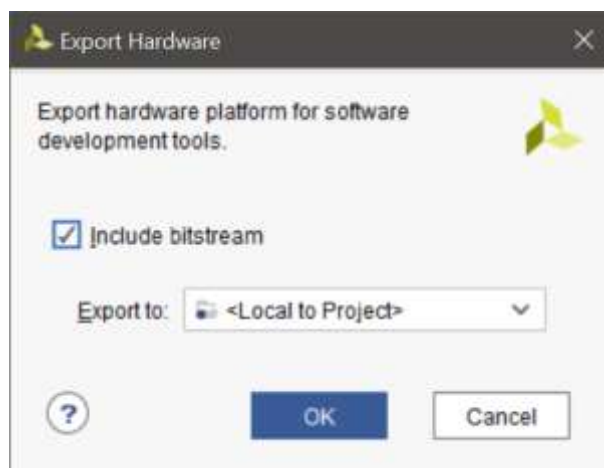
5. Cliquez sur OK pour visualiser le design implémenté

1.1.5 Quelques questions avant de continuer

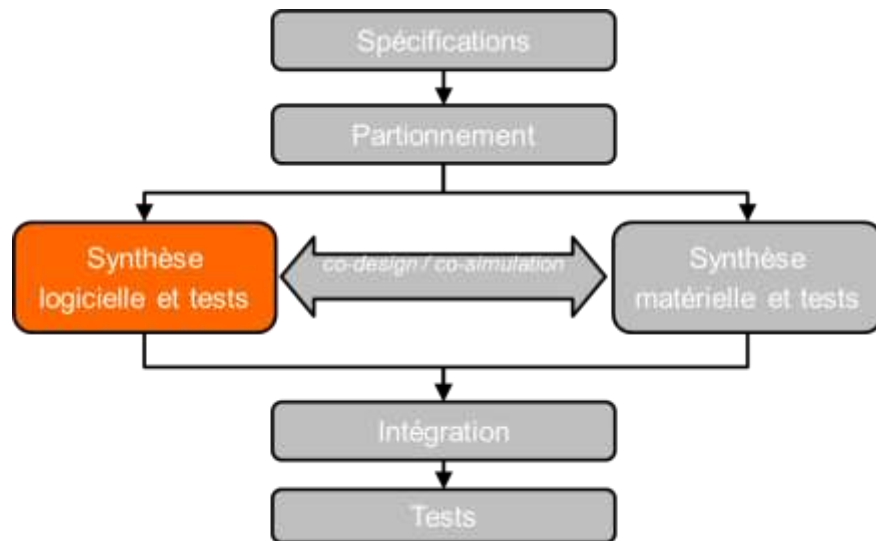
- a) Décrivez succinctement chacune des IP intégrées dans votre design
- b) Dans la fenêtre « Project Summary », affichez la table des ressources utilisées par système. Faites une copie d'écran à joindre dans le rapport
- c) Quelle est la fréquence de fonctionnement du design ? Joignez au compte rendu une copie d'écran prouvant que les contraintes de timing sont respectées

1.1.6 Export du design vers le SDK

1. Afin d'exporter les éléments de design nécessaire au développement et à la compilation du logiciel, cliquez sur File → Export → Export Hardware
2. Veillez à cocher « Include Bitstream »

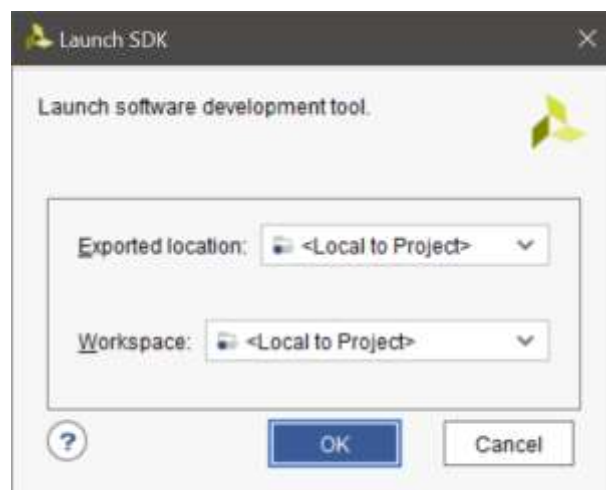


1.2 Codage du logiciel et tests

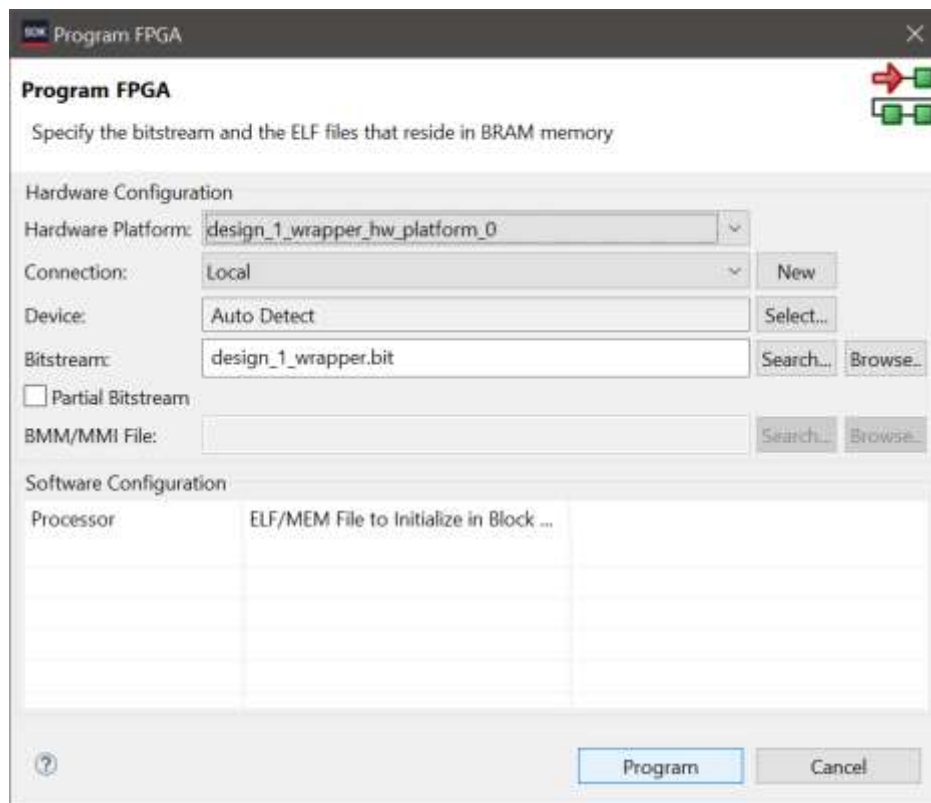



1.2.1 Mise en place de l'environnement de debug

1. Pour lancer le SDK, allez dans « File » > « Launch SDK ». Cliquez ensuite sur OK. L'environnement de développement logiciel « SDK » s'ouvre. Vous pourrez y voir le fichier « system.hdf » qui rassemble les éléments matériels importés. Le SDK est indépendant de Vivado. La conséquence est qu'il est fortement recommandé de fermer le SDK avant de faire des modifications matérielles dans VIVADO. A l'issue de modifications matérielles, il convient de re-faire les opérations de génération et d'export décrites au chapitres 1.1.4 Génération du bitstream et 1.1.6 Export du design vers le SDK



- Allez dans File → New application Project. Choisissez « matrix_m » comme « project name », et laissez les autres champs tels que proposés. Cliquez sur « Next » (attention à ne pas cliquer sur « Finish ») et choisissez le template « Hello World »
- Vous constaterez que 2 répertoires ont été créés, 1) matrix_m qui contient les binaires, les fichiers h, et c, et 2) matrix_m_bsp qui est le BSP (board support package)
- Connectez la carte PYNQ-Z2 au PC via un câble USB et mettez la sous tension.
- Dans « Xilinx Tools » cliquez sur « Program FPGA »



- Cliquez sur « Program »
- La fenêtre de log SDK doit se terminer par « **FPGA configured successfully with bitstream...** »
- Ouvrez un terminal Linux
- Tapez et exécutez la commande « `dmesg | tail` » et vérifiez qu'un message similaire à « `FTDI USB Serial Device converter now attached to ttyUSBx` » s'affiche. Notez x qui est le chiffre correspondant au numéro de l'interface USB série entre le PC et la carte de développement
- Tapez et exécutez la commande « `minicom -D /dev/ttyUSBx` en remplaçant x par le chiffre précédemment relevé
- Revenez dans le SDK et dans le « Project Explorer », faites un clic droit sur « matrix_m », puis sélectionnez « Debug as » → « Debug Configurations »
- Double cliquez ensuite sur « Xilinx C/C++ application (GDB) » pour créer une nouvelle configuration
- Cliquez sur « Apply » puis « Debug »
- Le debugger se lance. Après avoir cliqué sur l'icône , vous devriez voir s'afficher dans le terminal le texte « Hello World » correspondant à l'exécution du code

15. Modifier ce texte dans le fichier c, et relancer une exécution afin de vous assurer que vous maîtriser le processus d'édition, de compilation et de debug

1.2.2 Quelques questions avant de continuer

- d) Localisez le fichier « xparameters.h » qui est appelé depuis le fichier « platform.c ». Décrivez le contenu de ce fichier. Localisez dans ce fichier les informations relatives au processeur ARM et au Timer. Collez-les dans le compte rendu
- e) Localisez et ouvrez le fichier « system.mss » du BSP. Listez dans le compte rendu les drivers de périphériques contenus dans le BSP

2 Rappels sur la multiplication de matrices

Ce chapitre fixe les conventions utilisées par la suite, rappelle quelques éléments théoriques sur la multiplication de matrices. Les questions posées dans ce chapitre ne nécessitent pas l'accès au matériel du TP.

2.1 La multiplication de matrices 6x6

Pendant l'intégralité du projet, on s'intéressera à des multiplications d'entiers positifs. Nous utiliserons donc en C et en VHDL des entiers non signés et limiterons les valeurs d'entrées des opérations afin d'utiliser les ressources du FPGA.

2.1.1 Conventions pour le reste du projet

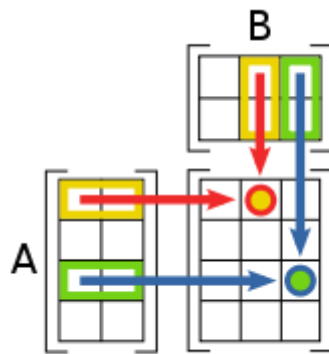
On veillera à utiliser « i » pour indexer les lignes et « j » pour indexer les colonnes dans le code C et VHDL.

On ne s'intéressera qu'aux matrices carrées de taille 6.

Définition du produit matriciel :

Si $A = (a_{ij})$ est une matrice de type (m, n) et $B = (b_{ij})$ est une matrice de type (n, p) , alors leur produit, noté $AB = (c_{ij})$ est une matrice de type (m, p) donnée par :

$$\forall i, j : c_{ij} = \sum_{k=1}^n a_{ik} b_{kj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}$$



On utilisera la convention « entiers non signés » pour le contenu des matrices à multiplier.

Les matrices à multiplier seront A et B. A étant la matrice de gauche de l'opération. Chacun des éléments de A et B devront pouvoir être représentés sur 12 bits. Vous veillerez donc à limiter les valeurs des éléments de ces matrices lorsque vous les initialiserez.

Les éléments de la matrice résultat de AxB seront représentés sur 32 bits.

2.1.2 Répondez aux questions suivantes :

Les questions ci-dessous sont théoriques, elles peuvent être traitées hors séance.

- f) Sur combien de bits est le résultat de la multiplication d'un entier de n bits par un entier de m bits ? Sur combien de bits est le résultat de l'addition d'un entier de n bits par un entier de m bits ?
- g) En consultant la documentation de la Pynq-Z2 déterminez de combien de blocs DSP dispose le FPGA de la carte. Quelle est la taille du multiplieur contenu dans chacun des DSP ?
- h) Représentez le Data Flow Graph du calcul du premier élément d'une matrice 6×6 (produit scalaire). Considérez que vous disposez d'une seule ressource (ALU) pouvant réaliser une multiplication en 3 coups d'horloge, et une addition en un coup d'horloge.
- i) En analysant, la réponse de la question précédente, déterminez-en combien de coups d'horloge le calcul du premier élément de la matrice 6×6 est-il réalisé ?
- j) En combien de coups d'horloge le calcul du premier élément de la matrice 6×6 serait-il réalisé si au lieu d'une ALU vous disposiez d'un multiplieur dédié réalisant une multiplication en 3 coups d'horloge et d'un additionneur réalisant une addition en 2 coups d'horloge ? Justifier.
- k) Dans le cas d'une multiplication de 2 matrices $(6,6)$, combien d'opérations de multiplication sont réalisées ? Combien d'opérations d'additions sont réalisées ?
- l) Dans le cas d'une multiplication de 2 matrices (n,n) , combien d'opérations de multiplication sont réalisées (en fonction de n) ? Combien d'opérations d'additions sont réalisées (en fonction de n) ?
- m) Soit t_{add} le temps de réalisation d'une addition et t_{mult} le temps de réalisation d'une multiplication. Exprimez $T(n)$ le temps de réalisation de la multiplication de 2 matrices (n,n) réalisée de manière séquentielle. Exprimez $T(n)$ sous la forme d'un polynôme

3 Edition du code, créations des fonctions de multiplication

L'objectif de cette phase est d'utiliser le SDK pour écrire le code qui réalise la multiplication des matrices en C. Vous mesurerez le temps d'exécution des opérations élémentaires addition, multiplication puis multiplication de matrice 6x6.

3.1 Création de test3

Dans votre projet créé dans le SDK, importer les fichiers « matrix_operations.c », « matrix_operations.h », et « helloworld.c » du répertoire « source_files » du package fournit.

1. Collez ces fichiers dans le répertoire du fichier « helloworld.c » jusqu'alors utilisé
2. Si nécessaire appuyer sur F5 pour rafraichir l'arborescence du projet

Description du fichier « helloworld.c »

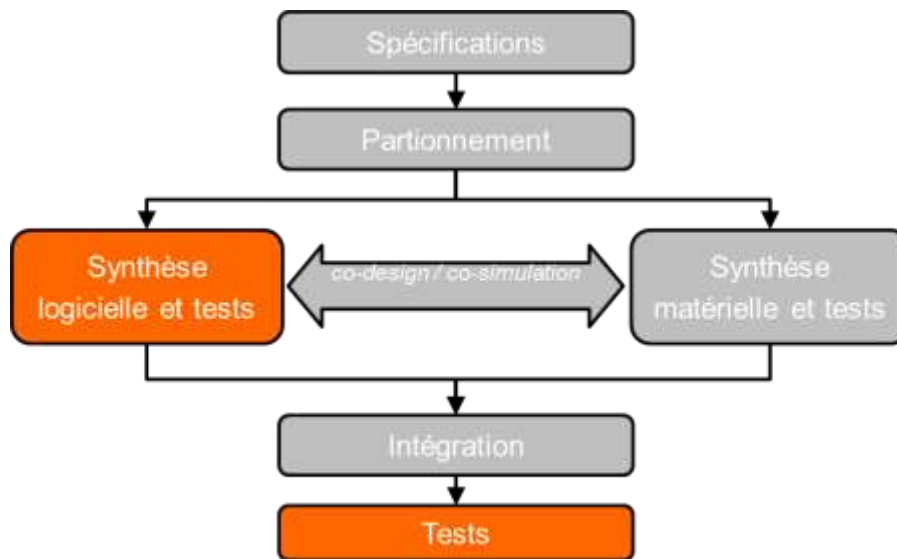
En début d'exécution, le timer est initialisé, vous n'avez pas besoin de toucher à cette partie du code. Analysez le code contenu dans test 1 et test2. Il permet de mesurer le temps entre deux instructions et de l'afficher sur l'écran du PC via l'UART.

3. Créez « test3 » qui doit réaliser une multiplication, afficher son résultat, et le nombre de cycles écoulé lors de son exécution

Conseil

Attention, entre deux exécutions, assurez-vous que la cible est initialisée avec le bon fichier « .elf ». Lorsque vous éditez le code et recompilez, assurez-vous de décharger le débbugger des autres exécutions en cliquant sur « remove all terminated ».

3.2 Création de test4



Complétez les fichiers source pour réaliser « test4 » qui doit mesurer le temps d'exécution de la multiplication de matrice. Faites en sorte que la sortie écran soit conforme aux valeurs numériques suivantes :

```
Hello World of Codelsign MULT 6
Timer rocks
```

```
-----
test 1 : No operation
t1: xxx, t2: xxx, diff:xxx
```

```
-----
test 2 : integer addition
t1: xxx, t2: xxx, diff:xxx
12025 + 11955 = 23980
```

```
-----
test 3 : integer multiplication
t1: xxx, t2: xxx, diff:xxx
12025 * 11955 = 143758875
```

```
Matrix A is
3307    3571    3944    719    785    3125
575     1167    1105    2975   1172   1842
2332    445     3678    369    1733   2127
406     622     536     3173   1946   1504
2034    1546    143     3524    158    863
2323    4086    1336    1241    1034   3258
Matrix B is
858     663     2235    3363    4061   1805
1321    3148    68      592     3851   2369
1105    2889    498     3189    461    1381
3449    2258    3142    1687    1651   1885
```



```

3755      1683      2048      2228      1109      3953
1477      3244      919       46       3419      3747

```

```

-----
test 4 : matrix multiplication
t1: xxx, t2: xxx, diff:xxx

```

```

Matrix Res is
21955948      37910422      16336738      28918572      41741841      36043293
20638251      21912760      15355275      13863207      19847868      22471268
17574566      24222547      13737219      24416567      22682834      25858766
22234605      19094466      16553784      13200629      16830097      22255695
17967670      17651155      16911421      14548311      23223586      18032302
21841965      33373933      15146037      19038852      40119484      34352178

```

```

end of tests

```

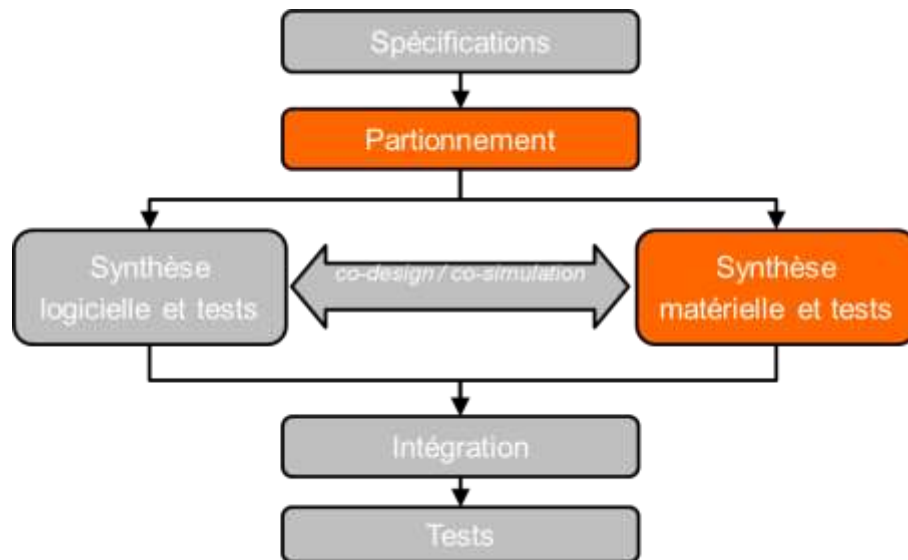
Vérifiez sur quelques exemples que le résultat de multiplication de matrices obtenu est correct. Pour la phase de débogage, n'hésitez pas à utiliser des matrices particulières telles que la matrice Identité afin de faciliter vos interprétations.

3.3 Complétez le compte-rendu

- n) Comment expliquer que le temps de réalisation de test1 ne soit pas nul ?
- o) En nombre de cycles, quel est le temps mesuré d'exécution d'une addition ? En y retranchant le temps de test1, quel est le temps effectif d'exécution d'une addition
- p) En nombre de cycles, quel est le temps effectif d'exécution d'une multiplication ? Remplissez la première colonne du fichier « mat_mult_timing.xlsx » avec les valeurs obtenues
- q) En utilisant les résultats des deux questions précédentes, estimez le nombre de cycles nécessaires à la réalisation d'une opération de multiplication de matrice 6x6
- r) Convertissez en secondes les réponses au 2 questions précédentes en utilisant la valeur de la fréquence d'horloge du processeur ARM. Déduisez-en le nombre théorique maximal d'additions par secondes atteignables sur cette cible. Même question pour le nombre de multiplications par seconde
- s) Utilisez un point d'arrêt en face de l'opération de multiplication de deux entiers. Utilisez Window → Show View → Disassembly afin de regarder les opérations assembleur associées à l'opération de multiplication. Collez dans le compte rendu les instructions associées

A cette étape faites une sauvegarde complète de votre projet, afin d'être capable de le restaurer intégralement dans une phase ultérieure.

4 Génération de système 1



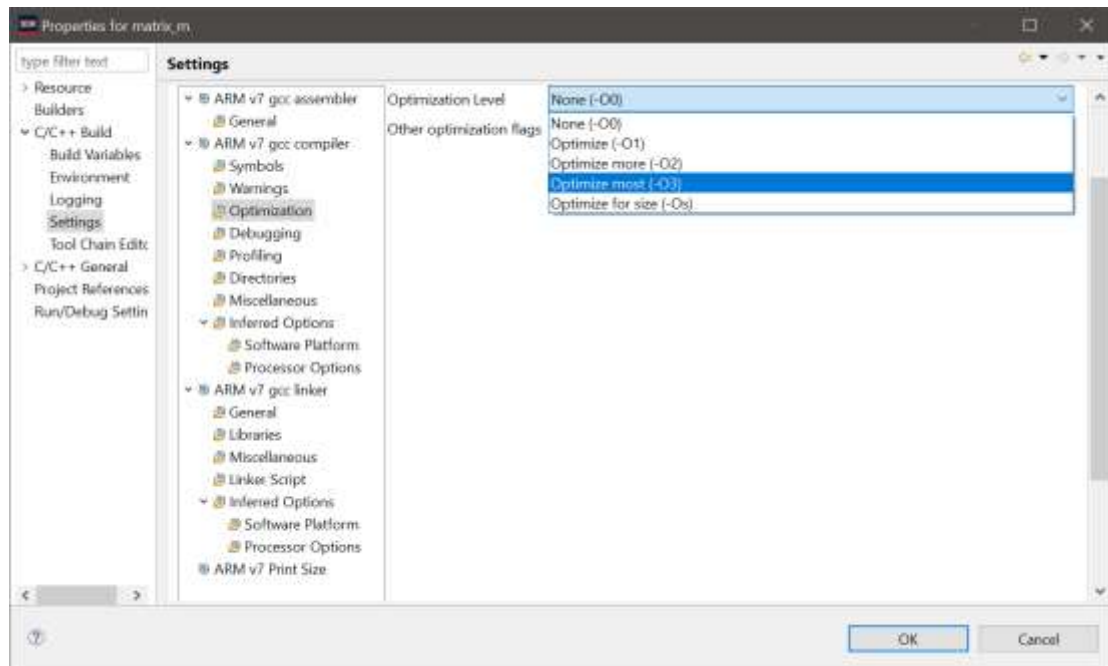
Cette phase consiste à apporter la première évolution au système et à mesurer son impact. Il s'agit de modifier la fréquence d'horloge du processeur grâce à Vivado et de modifier les options du compilateur dans le SDK.

4.1 Modification du système 0 sous Vivado

1. Fermez le SDK et retournez dans le block diagram de Vivado
2. Editer l'IP ZYNQ7 Processing System et changer la fréquence d'horloge du processeur pour la passer à 650 MHz
3. Relancez le processus de synthèse, génération du bitstream et enfin d'export vers le SDK tel que vu précédemment

4.2 Exécution sur la cible système 0

- t) Rendez-vous dans le SDK en faisant File → Launch SDK et complétez le fichier Mat_Mult_timing.xls avec les résultats obtenus pour le système 1. Pourquoi l'exécution est-elle plus rapide avec le système 1 qu'avec le système 0 ?
- u) Changez maintenant les options de compilation pour optimiser le temps d'exécution. Pour cela, modifiez le paramètre « Optimization Level » en le passant du moins agressif « -O0 » vers le plus agressif « -O3 ». Complétez le fichier Mat_Mult_timing.xls



- v) Collez les instructions assembleur associées à la multiplication des deux matrices. Quelle différence y a-t-il avec le système précédent ?
- w) En considérant que le temps de réalisation de la multiplication de 2 matrices (n,n) par les systèmes 0 et 1 s'expriment de la forme $T_{\text{système0}}(n)=K_0.n^3$ et $T_{\text{système1}}(n)=K_1.n^3$. Calculer K_0 et K_1 pour $n=6$ en utilisant le résultat obtenu
- x) Sachant que sur l'application du client, 80% du temps était dédié aux multiplications de matrices 6x6 sur le système 0, donnez le pourcentage du temps est dédié aux multiplications de matrices 6x6 sur le système 1

A partir de maintenant, l'objectif est la création d'un coprocesseur pour le processeur ARM. Il s'agit d'un multiplieur de matrices 6x6 sur le bus AXI-Lite. Cette IP sera synthétisée à partir du code VHDL que vous écrirez. Un tel module directement connecté au processeur permet de réduire le temps d'exécution de l'opération implémentée par rapport à l'exécution sur les ressources natives du processeur. Afin d'y arriver votre travail sera découpé en 2 phases :

1. Vous coderez le fichier vhd de multiplication de matrices
2. Vous intégrerez l'accélérateur matériel codé

Cette approche incrémentale permet d'avancer par petites étapes que l'on valide sur cible afin de découpler les problèmes relatifs à chacun des incréments : l'intégration d'une IP pour le premier, le codage VHDL pour le second.