INF8500 Systèmes embarqués : conception et vérification

Laboratoire # 3:

Exploration architecturale et Synthèse haut niveau d'un suiveur de lignes à l'aide de la méthodologie codesign

Automne 2019

1 Objectifs

Lors du développement d'une plateforme embarqué, il n'est pas toujours facile de déterminer initialement l'organisation matérielle/logiciel de celui-ci. Certes, la création de coprocesseurs pour toutes les fonctionnalités du système assurera la meilleure performance, mais qu'en est-il de la consommation et l'utilisation des ressources ? Si le développement d'un système est dicté par une suite de requis, il peut alors nécessiter un long processus d'analyse afin de déterminer l'ordonnancement matériel/logiciel. Lors de la méthode traditionnelle, le développement commencera seulement une fois cette analyse terminer et la séparation matérielle/logiciel claires. Encore là, rien n'indique que cette solution est optimale.

Afin d'augmenter la productivité des développeurs de système et réduire le temps entre la conception et la mise en marché, la méthodologie codesign a vue le jour. Cette approche consiste à l'utilisation de modèle transactionnel de haut niveau d'un système où chaque sous-système est modélisé sous forme de module pouvant être à la fois une définition matérielle ou logiciel.

L'objectif de ce laboratoire est d'introduire le développement de système à l'aide d'une plateforme haut niveau, l'application SpaceStudio de SpaceCodesing, nous permettant d'explorer un espace de solution rapidement et d'en faire une implémentation à l'aide d'outil HLS. La totalité de ce travail pratique s'échelonnera sur deux séances (considérant que vous avez déjà complété les 2 tutoriels[1]).

[1] Vous devriez avoir complété : 1) le tutoriel Vivado HLS sur l'exploration d'un filtre « fir » fournit dans l'archive du cours sous le dossier Vivado_HLS_Tutorial. Réf. « ug871-vivado-high-level-synthesis-tutorial » laboratoires 1 et 2 du chapitre 2 et 2) le tutoriel 1 fourni dans l'archive du cours sous le dossier SpaceStudio Tutorial.

Pour ce laboratoire vous devrez :

1. Vous familiariser avec le concept du Codesign à l'aide du logiciel SpaceStudio de SpaceCodesign
2. Vous familiariser avec le concept de la synthèse haut niveau à l'aide du logiciel Vivado HLS de Xilinx
3. Comprendre les concepts d'accélération (pipelinage, déroulage de boucle, partitionnement mémoire, etc.)
4. Comprendre l'algorithme SystemC d'un sous-système de détection de ligne pour le pilotage automatique d'un « rover » (LineTracker)
5. Explorer la possibilité d'accélérer le logiciel en faisant passer un bloc du logiciel au matériel.

2 Contexte

La création de circuits numérique est un processus dont la complexité ne cesse de grandir. Chaque nouvelle génération, suivant la loi de Moore, permet l'intégration plus de transistors sur une même puce. Or, afin d'utiliser ces nouvelles ressources, les concepteurs doivent produire de plus grand système permettant l'application de logique plus complexe. Pour augmenter la productivité des concepteurs, un nouveau flux de conception s'est tranquillement développé. De ce nouveau procédé sont nés les langages de description matérielle tels que VHDL et VERILOG, les simulateurs afin de valider les fonctionnalités des modèles et les synthétiseurs afin de traduire ces modèles en « netlist » pouvant être intégré sur un support physique (silicium). Par contre, cette méthodologie reste longue et difficile d'atteinte pour les développeurs logiciel qui on l'expertise dans le domaine algorithmique. Les logicielles de type High level synthesis (HLS), aussi appelé synthèse algorithmique, sont alors arrivé afin de faire plus facilement ce lien entre un algorithme et son implémentation matérielle (RTL).

Les logiciels de codesign s'inscrivent alors dans la continuité de ce flux et apporte une solution plus complète et accélère l'exploration architecturale d'un système logiciel et matériel embarqué dans son entièreté. Pourquoi le besoin de tel logiciel? Afin de répondre adéquatement au besoin des utilisateurs, une application suit habituellement un groupe de requis (acquisition 30 images seconde, débit de communication Mbs, etc.). Or, un système embarqué possède lui aussi des requis qui sont de différente nature telle que la performance (la rapidité de traitement), les ressources disponibles (espace, transistors, etc), la consommation (watt), la fiabilité, le coût, etc. En tenant compte des requis de l'application et

de ceux du système embarqué sans l'aide de logiciel de codesign il est difficile de trouver une piste de solution efficace rapidement. Le codesign permet alors de manipuler à haut niveau les divers applications et algorithmes du système afin d'explorer leur implémentation à l'aide de différentes ressources (processeur général (ARM, LEON), coprocesseur, protocole, etc.). À l'aide de ces implémentations, le designer est alors capable d'obtenir une estimation relativement précise des performances du système et être en mesure de sélectionner adéquatement une solution viable qui répond aux requis. Il est alors possible de la raffiner par la suite avec le modèle traditionnel. Bref, le Codesign permet de réduire rapidement l'espace de recherche d'une solution matériel et logiciel.

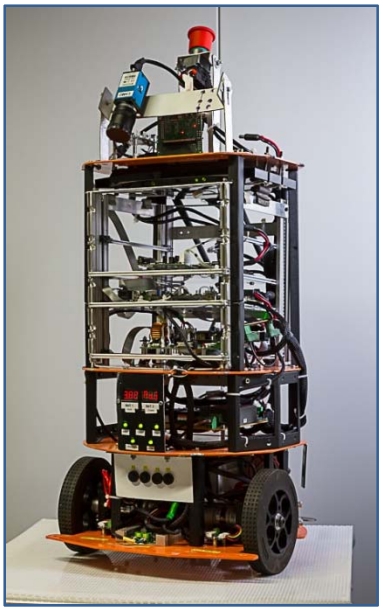
3 Vivado HLS

La synthèse comportementale (High Level Synthesis) aussi appelée « ESL synthesis » permet de créer un système à haut niveau à l'aide de code C/C++/SystemC et produire la logique numérique en format RTL (Verilog/VHDL). De ces sources RTL il est possible de créer une implémentation « gate-level » (porte logique de base) à l'aide de la synthèse logique.

La synthèse à haut niveau est un domaine de recherche très actif et est issue du besoin de palier à l'augmentation de la complexité des systèmes numériques. En permettant de définir un système à un plus haut niveau d'abstraction à l'aide d'un langage comme SystemC, la synthèse permet une plus grande productivité, mais aussi une meilleure optimisation des ressources et des performances de la microarchitecture du module synthétisé. Il existe plusieurs outils HSL notamment des outils commerciaux tels que Vivado HLS de Xilinx, CatapultC de Calypto Design Systems et I++ d'Intel.

Il est aussi connu que les outils HLS d'aujourd'hui procurent de la contrôlabilité (à travers les pragma). Plus l'ingénieur sait comment contrôler, plus optimal est le résultat. Selon les algorithmes analysés et des synthétiseurs utilisés, l'approche HLS peut présentement offrir des performances comparables ou moins bonnes que la conception manuelle. Ceci dit, le bénéfice de la synthèse à haut niveau est le gain en temps de productivité étant donné qu'un logiciel HLS peut donner un module RTL en quelques minutes, voir heures, plutôt qu'en semaines, voir mois, si le module doit être fait manuellement.

4 Space Studio



L'application SpaceStudio de la compagnie Space Codesign permet la conception conjointe de logiciel et matériel de manière efficace à l'aide de plateforme virtuelle. Ce logiciel s'inscrit dans le domaine des CAD c'est-à-dire qu'il assiste les concepteurs de système au partitionnement entre logiciel et matériel et permet de travailler au niveau ESL (Electronic System Level).

SpaceStudio nous permet donc de travailler au niveau algorithmique d'un système. Chaque tâche devant être exécutée par le système est définie à l'aide de module et leurs intercommunications (bus, fifo, etc) sont gérées par l'outil. Ceci offre une abstraction qui facilite les modifications de paramètre (cache, grandeur mémoire, etc.) et d'architecture (bus, type de mémoire,etc.), étant donné que nous ne sommes à un plus haut niveau que le RTL.

Par cette abstraction, le logiciel permet d'accélérer le développement de système embarqué. En effet, il permet la validation tôt dans le cycle de développement sans avoir la plateforme matérielle définie. Il facilite les modifications architecturales et algorithmiques par son approche haut niveau avec le SystemC et il permet d'examiner les performances (e.g. temps d'exécution du programme et consommation d'énergie) d'un grand nombre de solutions rapidement par son approche de configuration et de plateforme virtuelles.

5 Suiveur de ligne

Le suiveur de ligne est une application conjointement développée par SpaceCodesign et l'IRT. Le rover Twirtee est un démonstrateur développé par l'IRT A de Saint-Exupery afin d'évaluer de nouvelles méthodes et de nouveaux outils pour le développement au niveau système. Le rover représente un cas d'utilisation typique des systèmes critiques du domaine du transport, de l'aéronautique et spatial [7].

Le sous-système analysé dans ce laboratoire est un des ceux qui compose le « rover ». Il permet à un appareil de type « rover » d'être en mesure de suivre en temps réel une trajectoire délimitée par un groupe de lignes. Afin d'identifier le chemin, ce système applique des filtres aux images provenant d'une caméra et produit une image simplifier de droites destinées à système d'asservissement.

Pour la suite du laboratoire, on vous guidera sur l'exploration architecturale de ce sous-système.

5.1 Présentation de l'Algorithme

Dans cette section nous allons décrire le fonctionnement du sous-système de détection de ligne.

Figure 1 - TwIRTee [7]

5.1.1 Réception

Le premier module de ce sous-système se nomme « reception ». Ce module agit comme interface entre l'acquisition d'image et le corps de l'algorithme du système. Il se charge de recevoir une image par « socket » et de la localiser en mémoire DDR afin de la rendre disponible à l'algorithme de détection de lignes. Il utilise des librairies système afin d'effectuer cette tâche et nécessite d'être supporté par le système d'exploitation Linux lors d'une implémentation.

Dans le cas d'une réelle implémentation de ce sous-système, ce module serait échangé pour un autre module qui lui ferait l'acquisition d'image par caméra. Cette segmentation permet alors d'interchanger la méthode d'acquisition sans influencer l'algorithme principal.

5.1.2 Détection de lignes

Le second module de ce sous-système se nomme « line_dectection ». Ce module représente le corps de l'algorithme de détection de ligne. Il analyse une image de la mémoire et tante d'identifier les lignes présentes dans l'image. Ces lignes serviront au module d'asservissement afin d'orienter le « rover ». L'algorithme se découpe en 4 sections.

5.1.2.1 Ajustement de l'image

La première section de l'algorithme est effectuée par la fonction « resize ». Cette fonction se charge de réduire la grandeur de l'image afin de travailler avec un nombre restreint de pixels. L'algorithme de détection de ligne ne requiert pas que l'image soit de très haute qualité. La baisse en qualité permet de réduire l'empreinte mémoire et limite les communications avec la mémoire centrale.

L'algorithme calcul la valeur d'un nouveau pixel à l'aide de la moyenne de tous les pixels retirés de l'image. Le tout est limité à une valeur de 8 bits (255).

5.1.2.2 Algorithme de filtrage

La seconde section de l'algorithme est l'exécution d'un filtre de type Gaussien afin de lisser l'image et réduire le bruit de celle-ci. Pour se faire, on doit effectuer une opération de convolution.

Au cœur d'un filtre dans le domaine spatial 2D se trouve l'opération de convolution. La convolution 2D d'une image par un « masque » (filtre spatial) correspond à une transformation glissante basée sur le voisinage des points. Par exemple, avec un masque $M(k,p)$ de taille 3×3 :

$$I_f(i,j) = \sum_k \sum_p I(i-k, j-p) M(k,p)$$

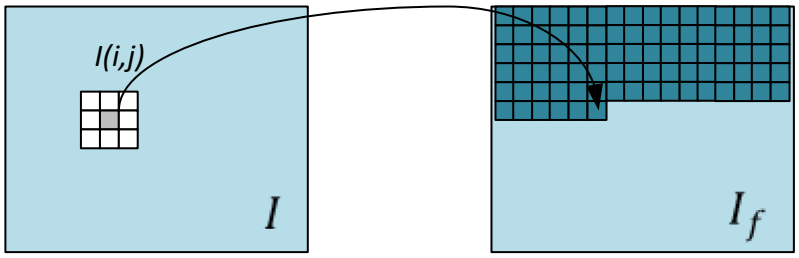


Image originale Image filtré

Par exemple, voici l'application du filtre moyen sur le pixel avec le voisinage suivant :

4	1	2
4	5	6
2	4	8

$$I_f = \begin{bmatrix} 4 & 1 & 2 \\ 4 & 5 & 6 \\ 2 & 4 & 8 \end{bmatrix} * \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \frac{4}{9} & \frac{1}{9} & \frac{2}{9} \\ \frac{4}{9} & \frac{5}{9} & \frac{6}{9} \\ \frac{2}{9} & \frac{4}{9} & \frac{8}{9} \end{bmatrix} = 4$$

Le pixel dans l'image filtré aura la valeur de 4 au lieu de 5 comme l'original.

Point important, on se doit de tenir compte des cas particuliers que sont les pixels de contour. Si on applique le masque du filtre sur le pixel à la position (0,0), une partie du masque se retrouve à l'extérieur de l'image.

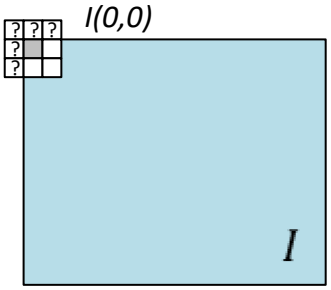


Image originale

Il existe plusieurs solutions pour filtrer les pixels de contour :

- Les ignorer, alors l'image deviendra plus petite que l'originale
- Considérer l'image périodique (répété la valeur du pixel de contour dans les pixels en dehors de l'image)

5	5	5	6
5	5	5	6
5	5	5	6
4	4	4	8

- Considérer l'image symétrisée (miroir aux bords)

8	5	4	8
5	5	5	6
6	5	5	6
8	4	4	8

- Ajouter des valeurs constantes aux bords

Il n'existe aucune solution parfaite.

5.1.2.3 Algorithme de canny

Par la suite l'algorithme effectue la détection des contours à l'aide d'un filtre de Canny effectué par la fonction « convolution_x_y_canny() ». Le filtre Canny est basé principalement sur le filtre de type Sobel. Le filtre de Sobel est un filtre de type gradient. À la base de ce filtre se trouve deux noyaux calculant une approximation du gradient en X appelé G_x et en Y appelé G_y.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \times I \ , G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \times I$$

Le gradient G_x est la variation entre pixels dans l'axe horizontal alors que le gradient G_y est la variation dans l'axe vertical. Le gradient résultant $\overrightarrow{G_{(i,j)}}(G_{x(i,j)}, G_{y(i,j)})$ représente la direction de cette variation. Il est donc possible, avec la combinaison du résultat de ces masques, de trouver le module du vecteur de cette variation afin d'en trouver l'amplitude.

$$G = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$

Cette amplitude peut alors servir pour mettre en relief les grandes variations dans l'image. Il est aussi possible de limiter cette amplitude et d'afficher ou de modifier seulement les pixels contenant une variation plus grande que ce seuil. Ceci permet d'avoir tous les détails ou seulement les grands contours bien définis, car plus l'amplitude est grande plus il y a un bris (variation) entre les pixels de cette région.

Finalement, il est aussi possible de trouver l'orientation de cette variation.

$$\Theta = atan2(G_y, G_x)$$

Le filtre de type Canny ajoute une étape de plus afin d'améliorer le traitement de l'image. Cette étape consiste à éliminer les éléments non maximaux afin de préciser la définition des contours. Ceci est effectué par la fonction keep_edges(). Le filtre Canny est alors plus précis qu'un simple filtre Sobel.

5.1.2.4 La transformée de Hough

La dernière partie effectue une analyse des pixels comportant des contours afin de recréer les lignes passant par ceux-ci. Ceci est effectué par une transformée de Hough. La transformée de Hough permet de prendre plusieurs points, dans notre cas les pixels, et les projeter dans l'espace de Hough sous forme de sinusoïdale. Si les sinusoïdales de différents

points se croisent, ceci indique qu'il existe une droite reliant ces deux points. Le point de croisement dans l'espace de Hough indique les paramètres d'une telle droite.

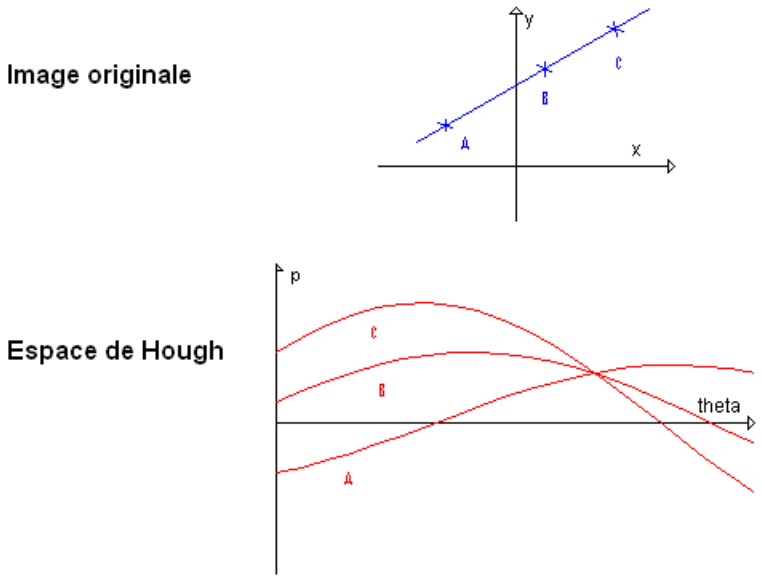


Figure 2 - Représentation de l'espace de Hough[8]

5.1.3 Affichage display

Le troisième module de ce sous-système se nomme « display ». Ce module se charge de communiquer par socket les coordonnées de droites détectées de la mémoire DDR du sous-système. Tout comme le module de réception, il utilise des librairies système afin d'effectuer cette tâche et nécessite d'être supporté par le système d'exploitation Linux lors d'une implémentation.

Dans le cas d'une réelle implémentation de ce sous-système, ce module serait échangé pour un autre module qui lui ferait la communication de ces données par bus local vers un autre sous-système d'asservissement.

5.1.4 Master

Le dernier module de ce sous-système se nomme « master ». Ce module se charge de communiquer aux autres modules les paramètres d'exécution du sous-système. Cette communication est faite par l'entremise de fichier registre. Ceci représente une mémoire locale rapide située au cœur de ce sous-système.

6 Architecture séquentiel de l'algorithme

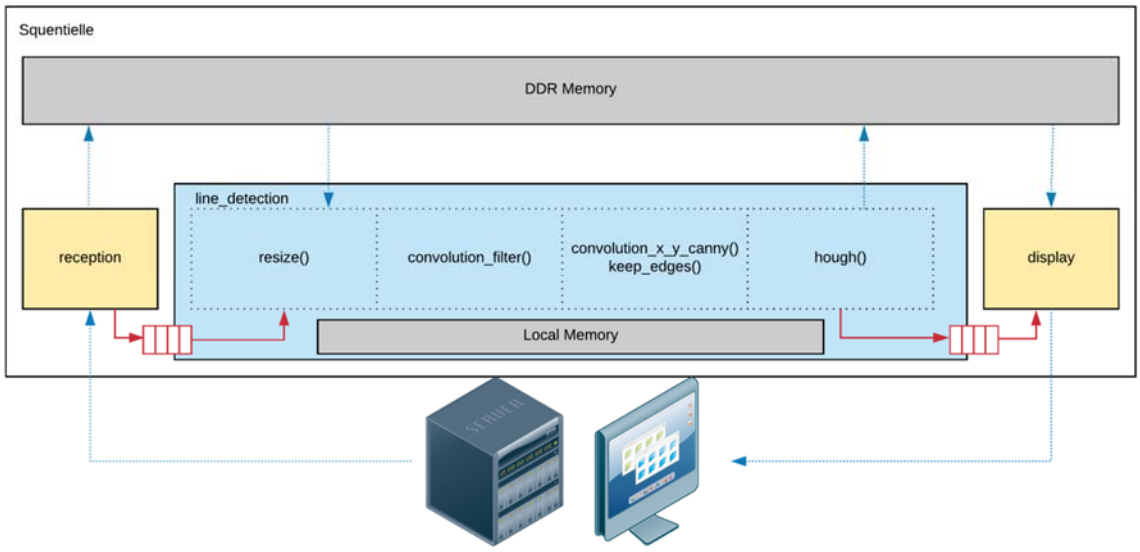


Figure 3 - Algorithme séquentiel

6.1 Architecture Validation et création de l'architecture de référence

Attention :

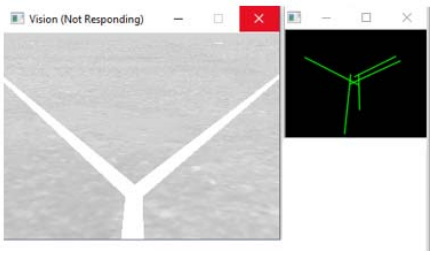
- *Il faut ABSOLUMENT ne pas mettre le code du TP sur le disque X:, il doit absolument être dans C:/TEMP. Pour plus de détails voir la section 9.1.*
- *À tout moment si vous rencontrez des problèmes, consultez la section 9.*

Comme code de départ, nous vous avons fourni une *solution*. Cette solution se nomme *sequentiel*. Cette solution représente la forme traditionnelle de l'algorithme. L'approche séquentielle représente un cas typique d'exploration architecturale d'une application haut-niveau. L'application forme un seul module nommé « line_detection » ou chaque opération effectuée sur une trame est fait une à la suite des autres. Cela veut dire que la fonction « resize() » ne peut commencer une nouvelle opération tant que la fonction « hough() » n'a pas terminé son traitement. De plus, la communication entre ces différentes fonctions se fait de façon implicite à l'aide de la pile d'exécution et de la mémoire locale.

Dans *sequentiel* on retrouve une première architecture du nom de *validation*. Cette dernière indique que tout est matériel (icone HW) et est composé de 4 module (thread SystemC). Cette première architecture est utilisé pour faire la validation fonctionnelle (comme au lab 2). Sauf qu'ici tout est fait automatiquement (e.g. les fichiers main) et vous n'avez pas à travailler au niveau des sc_ports. Vous utilisez l'API de SpaceStudio (ModuleRead, DevideRead, etc.) et tout le détail SystemC d'appels de synchonisation et de communication (en TLM2) se trouve caché derrière.

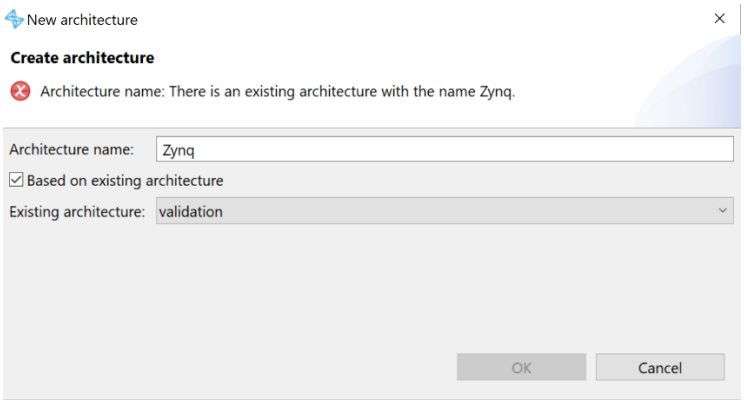
On vous suggère donc de générer, builder et exécuter *validation*, mais avant, assurez-vous que le serveur pour afficher les images fonctionnent. Consultez la section 9.2 i et ii.

Ensuite (après avoir démarré le serveur avec « **start_tcp_server_simulation** »), vous pourrez faire *execute*. Vous devriez voir apparaître de manière périodique un nombre de frames par seconde qui représente le débit avec lequel *line_detection* traite les images (retenez bien le nombre de frame par seconde moyen pour comparaisons futures). Vous devriez aussi voir les fenêtres suivantes :

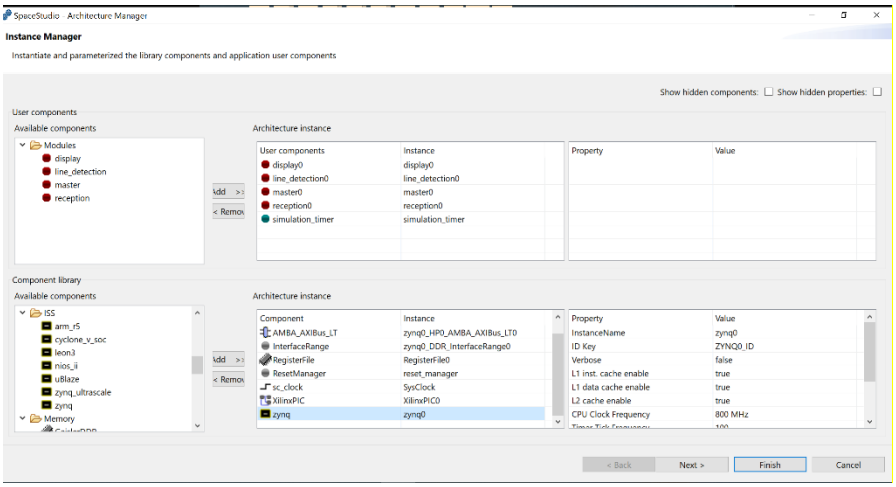


Dans l'architecture validation tout s'exécute sur SystemC. Nous allons maintenant introduire un processeur dans la boucle de simulation SystemC. Plus précisément, nous allons créer une architecture *Zynq* qui contiendra un ARM Cortex A9. Puis nous allons assigner le code *line_detection* sur le ARM.

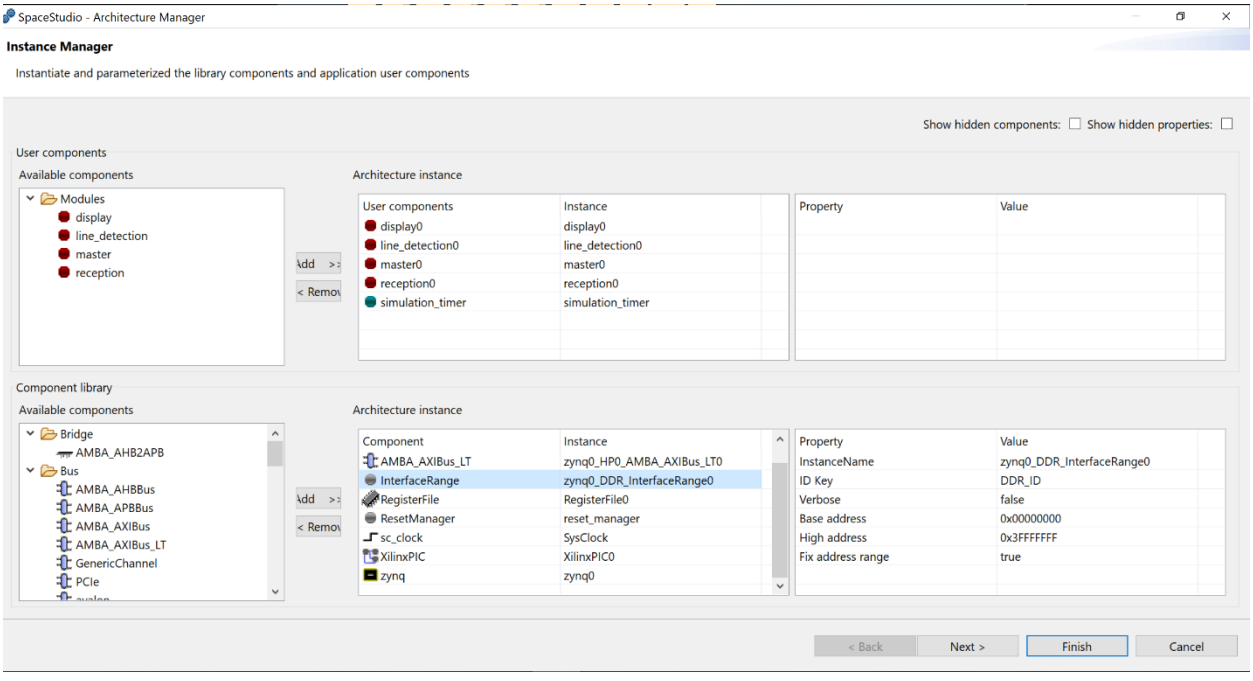
Pour se faire, cliquez à droite sur la solution « **sequential** » et sélectionnez « **New Architecture** ». Dans la fenêtre de création, nommée la « **Zynq** » à partir de *validation* :



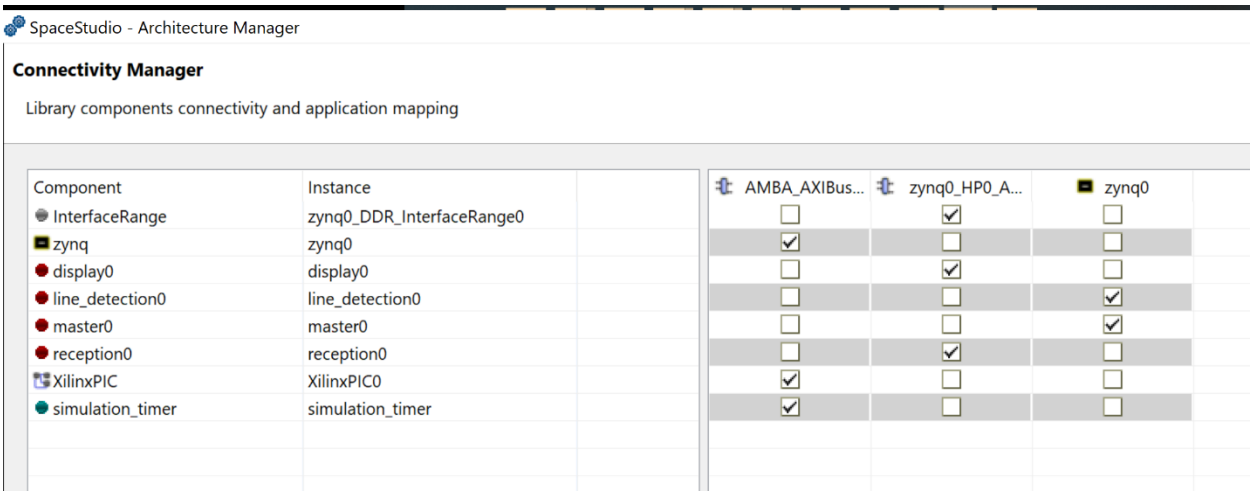
Dans Component library, ajoutez un zynq :



Notez qu'on a pu besoin du XilinxDDR quand on utilise le Zynq puisqu'il a sa propre DDR. À la place, il faut utiliser le InterfaceRange qui vient en même temps que le ADD du Zynq. Par conséquent toujours dans Component library, faites un remove du composant Xilinx DDR et donner à InterfaceRange le ID Key de Xilinx DDR i.e. DDR_ID. Ce qui donne :



Finalement cliquez pour passer à la connexion des modules et assurez-vous d'avoir la l'assignation suivante :

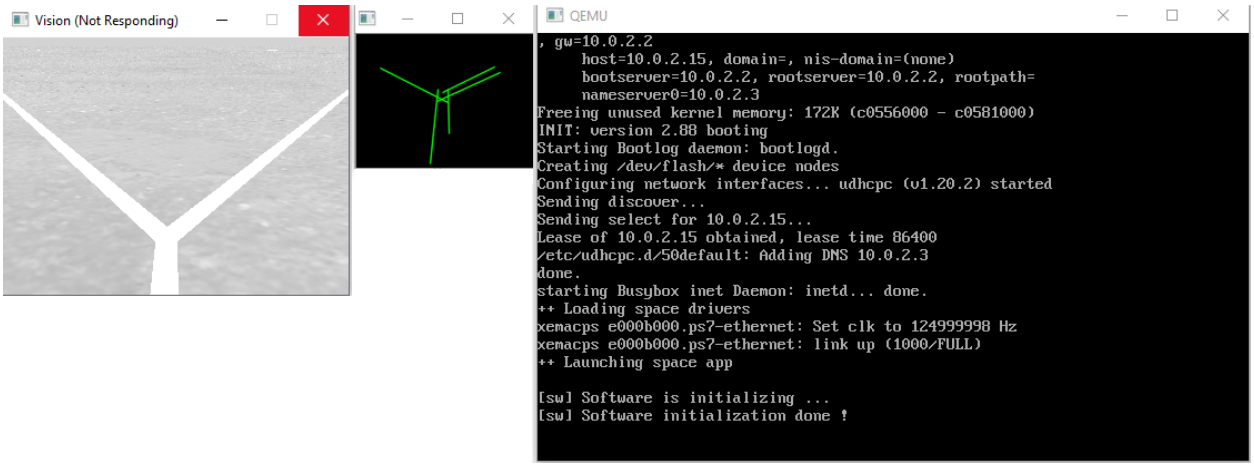


Cette première architecture vous permet d'effectuer la simulation de la détection de lignes. Les modules « **master0** » et « **line_detection0** » sont représentés comme logiciel alors que les modules « **reception0** », « **display0** » et « **simulation_timer** » sont représentés comme matériel.

Lancez la génération des fichiers et assurez-vous de cocher « **Clean generated files** ». Par la suite, « **clean and build** » .

Puis lancer la simulation, en démarrant à nouveau le serveur avec « **start_tcp_server_simulation** ».

Une fois la simulation lancée, le simulateur prend quelques secondes à lancer QEMU et charger Linux (ce qui démontre bien qu'il y a un processeur dans la boucle de simulation SystemC). Une fois les modules en marche, vous devriez voir apparaître l'image envoyée et le résultat de la détection. Encore une fois, la performance du détecteur de ligne sera affichée dans la console de SpaceStudio sous forme d'images par seconde (retenez bien le nombre de frame par seconde moyen pour comparaisons futures).




```
 Space Codesign Systems Inc.
 Copyright 2005-2018. All rights reserved
 http://www.spacecodesign.com
 SpaceStudio 3.1.0
 ------------------------------------------------------------
ip address: localhost
Starting simulation.
DisTLM established a connection on localhost:1024
Warning: received DisTLM request from the past. DisTLM time is 0 and current time is 40000
Warning: received DisTLM request from the past. DisTLM time is 0 and current time is 80000
12.384948 frames/second
25.696033 frames/second
```

Figure 4 - Exemple d'exécution d'une simulation

6.2 Raffinement de votre architecture vers une implémentation

À partir d'ici vous avez deux options : 1) Allez à la prochaine section (section 7) et vous concentrer sur l'exploration architecturale afin d'accélérer l'application ou 2) de poursuivre dans cette section afin d'aller tout de suite faire l'implémentation de la solution Zynq. Bref, l'ordre importe peu, mais vous devrez faire les deux étapes.

Cette section permet d'expérimenter une manière de faire un passage de System (avec ou sans simulateur de processeur dans la boucle). SpaceStudio est (à notre connaissance) un des rares outils permettant ce raffinement. Ce processus de raffinement automatique (en milliers de ligne) se résume aux étapes suivantes :

- Raffinement de la plate-forme virtuelle (architecture) de C++ en RTL en faisant levier sur Vivado HLS
- Génération des interface de communication et de la «glue logic » pour les coprocesseurs matériels (éventuellement filter)
- Pilote pour les threads restés en logiciel
- Bootloader, fichier de configuration et assignation des bus.

Pour réaliser ce raffinement complétez les étapes suivantes[2]. Dans l'écran « **New Architecture** », nommez la « **Zynq_impl** », cochez la case « **based on existing architecture** » et sélectionnez votre architecture de simulation « **Zynq** ». Une fois l'architecture créée, migrez les modules « **display0** » et « **reception0** » vers le processeur Zynq0 en glissant ces modules sur lui dans l'écran « **Project Explorer** ». Faites un clic droit sur le module « **simulation_timer** » et sélectionnez « **Edit Properties** ». Assurez-vous de cocher la case « **Is simulation only** ». Autrement, SpaceStudio ne sera pas en mesure de produire la synthèse de l'architecture. **Pour le reste du laboratoire, ces opérations devront être effectuées pour chaque création d'architecture d'implémentation**.

Avant de lancer la synthèse, changer l'adresse ip de connexion des modules « reception » et « display ». Pour se faire, double cliquez le module et changez l'adresse ip située au haut du fichier entre les conditions de préprocesseur :

```
#if MAPPING == HW_MAPPING
        m_socket = new client_handler(this->name(), "localhost", "5016");
#else
        m_socket = new client_handler(this->name(), "192.168.2.140", "5016");
#endif
```

Il faut que le dernier paramètre soit "5016" pour display.cpp, et "5015" pour reception.cpp.

[2] N.B. Comme le premier raffinement se fait sur une solution complètement logiciel, il n'y aura pas d'appel à HLS Vivado. Cela viendra plus tard avec le passage de filter du logiciel au matériel.

Puis, insérer l'adresse ip de votre poste du laboratoire.

Par la suite, on se doit de spécifier à l'architecture l'adresse IP à utiliser pour ses connexions réseau. Pour ce faire référez-vous à la section 9.5.

Par la suite, on se doit d'ajuster la grandeur de la pile d'exécution alloué aux modules logiciels. Dans le menu de préférence, sélectionnez « Software Configuration File » et changer la grandeur pour « 1024000 ». Voir figure 6.

Cet ajout de script de démarrage ainsi que l'ajustement de la grandeur de la pile d'exécution devront être fait pour toute nouvelle architecture d'implémentation.

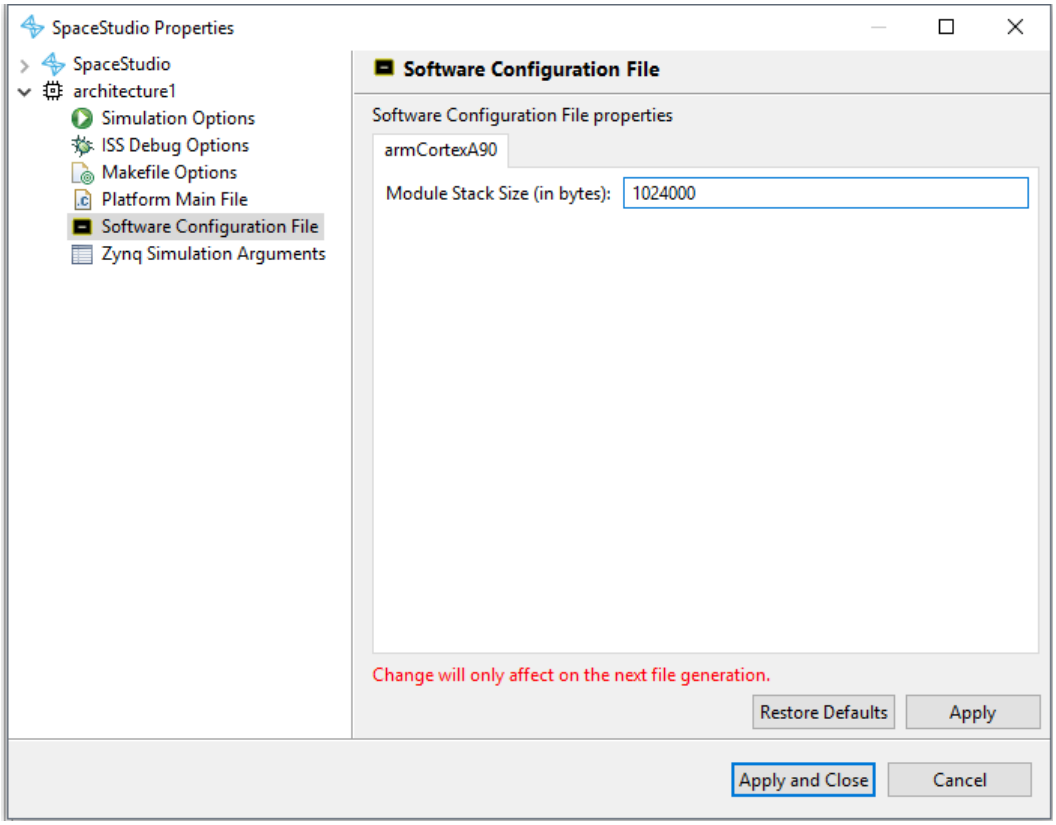


Figure 5 - Grandeur de la pile d'exécution

Finalement, lancez la synthèse en sélectionnant le menu « **Tool** » suivi de « **Architecture Implementation** ». Ceci va automatiquement générer et compiler le code. Dans l'écran d'implémentation, sélection ou créer le dossier « **c :\TEMP\Zynq_Impl** » et assurez-vous de sélectionner la carte « **Zedboard** » du menu déroulant « **Boards** ». Faites « **ok** ». Ignorez l'avertissement et faites « **ok** ». La génération devrait prendre 15 à 20 minutes.

Une fois la génération terminée avec succès, vous trouverez dans le dossier « **LineTracker\implementation\sequential\Zynq_impl\sd_card** » les fichiers à transférer sur la carte SD. Ces fichiers sont au nombre de 4 :

- **BOOT.bin** :le fichier bootloader
- **uImage** : image du noyeau linux
- **device_tree.dtb**: arborescence de périphériques nécessaire à linux pour charger les pilotes spécifiques à la carte actuelle (peut être très long à configurer manuellement…)
- **rootfs.cpio.gz**: le système de fichiers virtuels (ramfs) contenant l'application.

Pour tester l'implémentation, vous devez d'abord lancer le serveur d'image sur votre poste du laboratoire en mode implémentation. Pour se faire, rouler les commandes (référez à la section 9.2.iii) :

export MINGW64_SPACESTUDIO_DIR=<dossier> # Par exemple, avec <dossier> == /h/mingw64 si le ZIP a été extrait sous H:\

export SERVER_IP_ADDRESS=<adresse IP de la machine Windows>

Puis, à chaque fois qu'on roule le programme sur la carte SD, il faut exécuter le script :

./start_tcp_server_implementation.sh

Finalement, branchez la carte SD dans la « **Zebboard** » et allumez la carte.

6.2.1. Connexion par terminal

Afin de vérifier l'exécution de l'application sur la carte FPGA, vous pouvez vous connecter sur celle-ci à l'aide du fil relier au port « UART » et l'application « putty ».
Trouvez tout d'abord le port série auquel est connecté la carte FPGA.

- Ouvrez le « Device Manager »
- Agrandissez la section Ports (COM & LPT)
- Lors de l'activation de la carte FPGA le port « com » de celle-ci devrait y apparaitre.

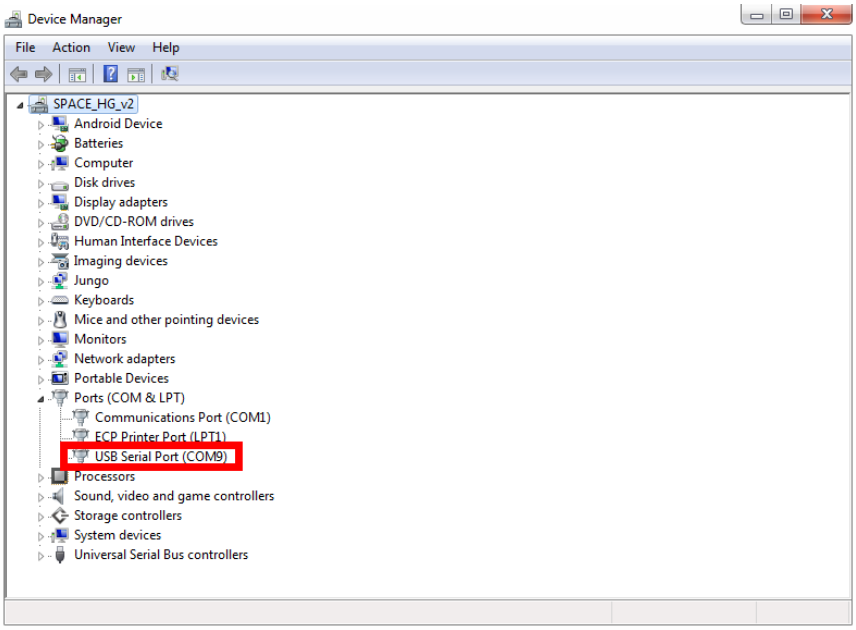


Figure 6 - Device Manager

Par la suite configurez « putty » pour s'y connecter

- Sélectionnez le mode « serial »
- Entrez le port de la connexion. Par exemple « COM3 »
- Entrez le baud rate « 115200 »

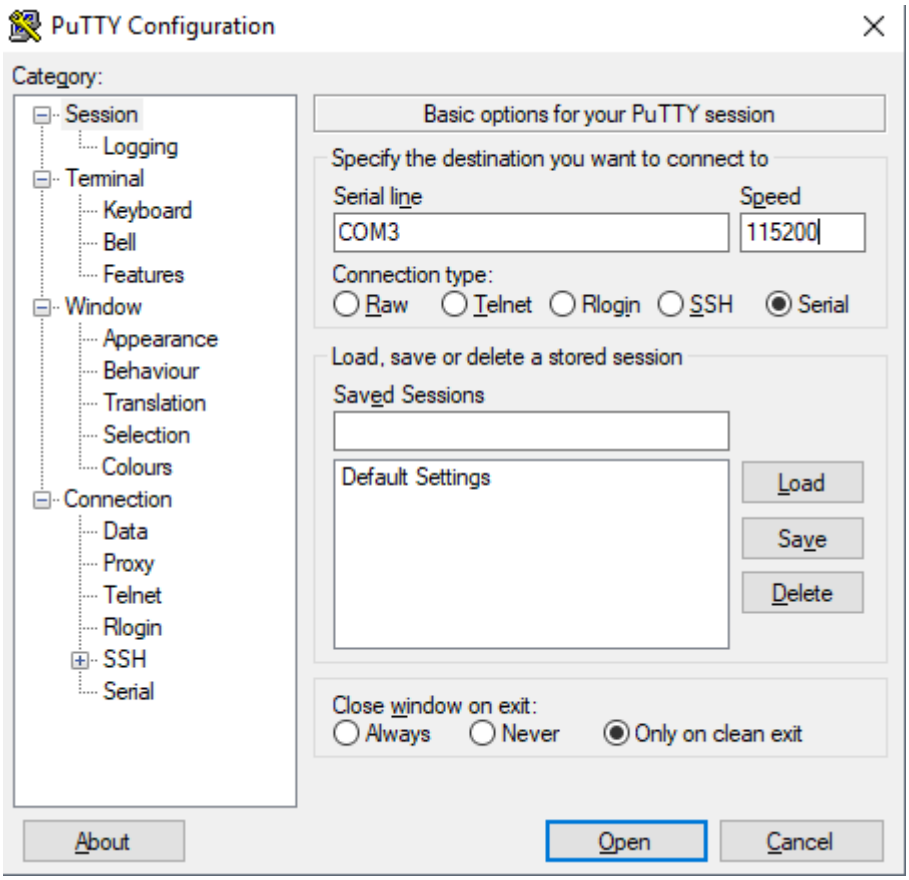


Figure 7 - Connexion à l'aide de putty

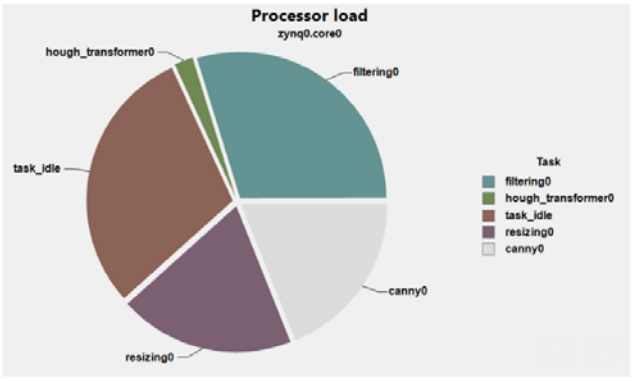
6.1.1 Lancez l'application

Une fois connecter à la carte FPGA et le processus de chargement de linux terminé, vous devriez voir le prompt **root@zynq:~#**. Afin de lancer l'application logiciel, déplacez-vous dans le dossier **/home/root/spaceApp** et lancer l'application **./architecture1_armCortexA91.arm.elf**

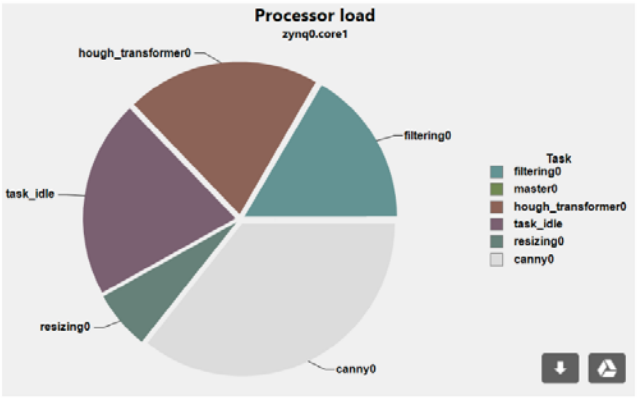
7 Exploration architecturale

L'architecture préalablement développé agira comme base des performances du module de détection de lignes.

La nouvelle version de SpaceStudio en développement (que vous n'avez pas encore) offre un profilage du style *gprof*. Voici le résultat de *line_detection* avec exécution de l'architecture Zynq (Figure 8):



hough: 2.07%
task_idle: 29.81%
resizing: 19.39%
canny: 18.96%
filtering: 29.74%



hough: 20.52%
task idle: 20.88%
resizing: 6.24%
canny: 35.68%
filtering: 16.5%

Figure 8 – Résultat de profilage

On voit donc que de mettre *filter* (il est nommé ici *filtering*) ou *canny* en matériel est un bon choix.

Le but de cette partie de laboratoire est de remplir le tableau suivant (Figure 9) :

CPU-FPGA	Partitioning	Frames per sec.			FPGA resources utilization							
		Estimation	Real execution	Accuracy (%)	Estimation				Real			
					LUT	FF	BRAM	DSP	LUT	FF	BRAM	DSP
Zynq	all_software											
	1x_filter in HW (sans optimisation)											
	1x_filter in HW (avec optimisation)											
	2x_filter in HW (avec optimisation)											

Figure 9 – À compléter

Si vous avez bien conservez les résultats de la dernière séance (tel que demandé), vous devriez être en mesure de compléter la première ligne du tableau (uniquement les performances en frames per sec.).

IMPORTANT : *Ne soyez pas surpris si la performance de filter (surtout un seul filter) en matériel est plus lente que la solution complètement logiciel, il y a plusieurs raisons à cela. D'abord le ARM roule à plus de 600 MHz alors que le FPGA roule à 100MHz. Il faut donc plus qu'un filter en matériel pour paralléliser davantage. Dans ce laboratoire on vous montrera comment en mettre deux mais on ira pas plus loin faute de temps. L'idée est de comprendre comment on peut facilement généraliser à n et ainsi battre la solution complètement logicielle.*

L'objectif est donc de paralléliser *filter* pour obtenir de meilleures performances. Vous devez donc :

1) Créez une nouvelle solution (et non une architecture) basée sur la solution *sequential*. Nommez la *filter_hw*.
2) Dans *filter_hw* on va travailler avec l'architecture *Zynq* (mettre cette architecture active)
3) Créez un nouveau module (par exemple, filter) dans la nouvelle solution (en tapant Solution -> New module)
4) Vous constaterez que *filter.cpp* et *filter.h* est un *sc_thread* SystemC. Vous allez maintenant compléter le code de *filter.cpp* :
    a. Vous devez déplacer le corps de la fonction vers le thread de *filter*
    b. Adapter la communication (entrées et sorties) de la fonction à l'aide de l'API de communication. L'image devra être transférée à partir d'un *DeviceRead()* dans *DDR_ID* et retourner avec *DeviceWrite()* dans *DDR_ID*. Mais auparavant, vous devrez allé chercher les paramètres de l'image (*m_img_width*, *m_img_height*, *m_img_in_addr*, *m_img_out_addr*) avec des ModuleRead.

IMPORTANT : *Pour plus d'information sur les l'API de communication, consultez le pdf sur l'introduction à SpaceStudio sur le web du cours (Chapitre5/Étude de cas d'un outil de conception système: SpaceStudio).*

    c. Déplacer *filter* sur le matériel (zynq0_HP0_AMBA_AXIBus_LT0)
    d. Générer, compiler, exécuter et déboguer!
    e. Faire le raffinement jusqu'à l'implémentation sur la carte
    f. Complétez la 2e ligne la Figure 9.

Une fois que cela fonctionne, voici ce que nous vous proposons:

5) Ajoutez-y des pragmas au module *filter* pour tenter d'optimiser votre solution avec HLS Vivado (comparez vos solutions avec et sans pragma). Je vous suggère de faire 2 solutions d'optimisation[3] :
   a. Pragma HLS unroll sur les 2 boucles intérieures (C : et D :)
   b. Pragma HLS pipeline sur les 2 boucles intérieures (C : et D :)

   Vous devez donc :
   i. Pour justifier rapidement votre choix utiliser le rapport de simulation de Vivado HLS (ne faites pas l'implémentation complète tout de suite). Dans un premier temps arrêtez-vous après avoir complété Architecture Implementation et allez chercher directement les résultats de synthèse dans **:\TEMP\Zynq_Impl_filter_optimization\hls\filter\hls_project\solution1\syn\report**.
   ii. Dans un deuxième temps (quand vous aurez choisi la meilleure des 2 solutions après lecture des rapports de synthèse), faites une synthèse complète jusqu'à l'implémentation.
   iii. Complétez la 3e ligne de la Figure 9.

6) Sur la meilleure solution obtenue en 5), vous allez créer une deuxième instances de *filter* afin de diviser en 2 parties égales le travail. Vous devez :
   i. Cliquez sur le bouton de gauche et sélectionnez le composant *filter* (rond rouge) sous dans *User Components* puis amenez le (*drag and drop*) sous le bus zynq0_HP0_AMBA_AXIBus_LT0. Vous devriez voir apparaître *filter1*.
   ii. Il ne vous reste qu'à diviser l'image en deux parties et faire un appel à *filter0* et *filter1*.
   iii. Complétez la ligne 4 de la Figure 9.

8 Remise et questions

Le laboratoire est à rendre le 8 décembre 2019

Voici le barême pour le laboratoire no 3 :

Barème	
Introduction	/1
Fonctionalité et qualité du code	/10
Réponses aux 2 questions	/7
Conclusions	/2

[3] Au besoin consultez le fichier pragma (exemple du FIR) dans le chapitre 4 afin de voir où placer les pragma.

Dans l'introduction, vous devez résumer en quelques lignes ce que vous faites dans ce laboratoire. Dans la conclusion, faire une synthèse du travail à l'aide des résultats et expliquez le lien que vous voyez entre ce laboratoire et les deux premiers laboratoires. Suggestion : aidez-vous de la figure 1.2 de la question 1 de l'intra…

Question 1 (4 pts)

Présentez d'abord les résultats de la Figure 9.

Ensuite, j'ai mentionné dans la section 7 : *Ne soyez pas surpris si la performance de filter (surtout un seul filter) en matériel est plus lente que la solution complètement logiciel, il y a plusieurs raisons à cela. D'abord le ARM roule à plus de 600 MHz alors que le FPGA roule à 100MHz.*

Déterminez deux autres raisons qui pourrait expliquer cette contre-performance (probablement surtout avec 1 seul). Expliquez clairement.

Question 2 (3 pts)

Expliquez bien la différence entre HLS Vivado et SpaceStudio. Commencez par expliquer le rôle de chaque outil et expliquez comment ceux-ci se complète avec l'aide de la Figure 10.

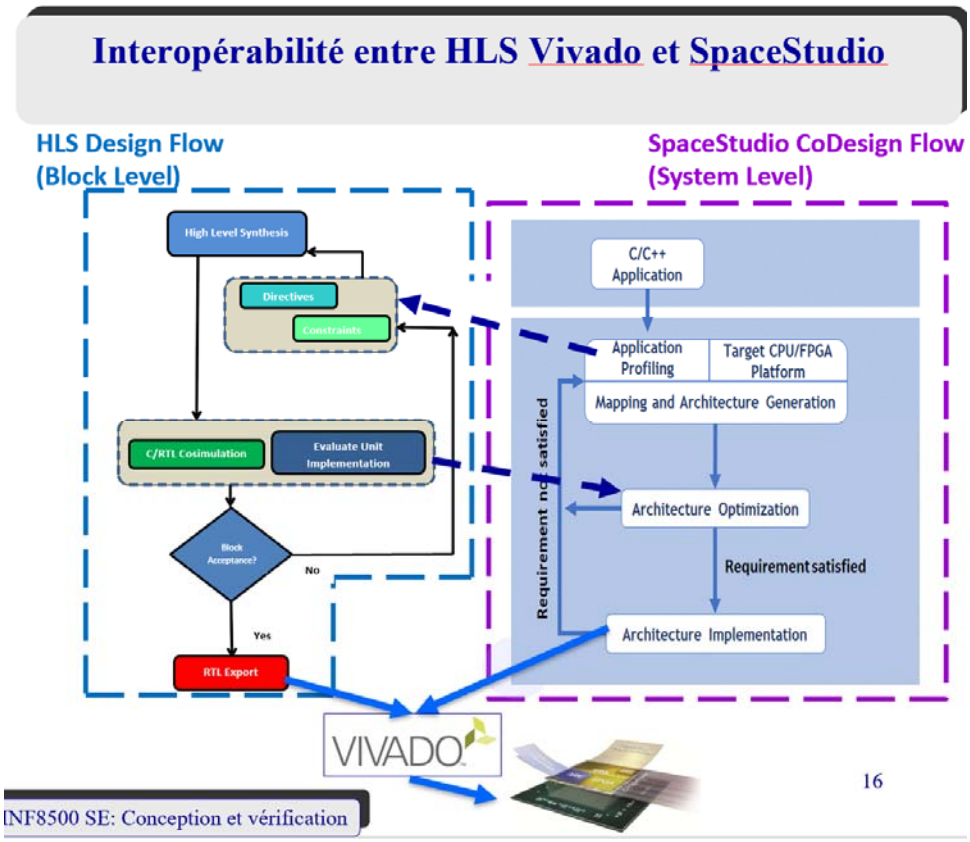


Figure 10

9 Annexe (version sectionné du REAME.md)

9.1 Concernant les répertoires du code de départ

LineTracking

(You can view this file best by opening it in VSCode, then hitting CTRL+SHIFT+V).

This is the multithreaded LineTracking.

IMPORTANT: Put your TP code inside C:/TEMP**. You must NOT put your TP code inside the** X: **drive**, since one of the programs of a toolchain called Cygwin that SpaceStudio uses fails when it is run on a drive that resides at a network location. **Remember to save/push your code before closing your session to avoid losing your work.**

server/ directory

The server is an executable built using MinGW's Posix compatibility libraries and with OpenCV. In itself, it has nothing to do with SpaceStudio ; it just simulates a real server, e.g. a camera on a robot.

spacestudio/ directory

Contains the SpaceStudio's LineTracking project. Open spacestudio/LineTracking.spacestudio with SpaceStudio.

Instructions

SpaceStudio takes on the very difficult task of making Vivado HLS, Vivado, Linux, Linux drivers, UBoot, custom hardware IPs, QEMU, custom CPU simulators, etc. all work together at the push of a button. While this is appealing, it should be expected that some problems arise. Here is a list of things to do/not to do, and of troubleshooting tips.

9.2 Détails concernant le serveur

i. Au départ : installez le *SpaceStudio's MinGW64 toolchain*

Get SpaceStudio's MinGW64 toolchain

Amongst other things, the server needs OpenCV to compile, link and execute. So, before being able to compile the server, we need to download SpaceStudio's MinGW64, available here, with all its packages and libraries. Download the ZIP file and unzip it somewhere, e.g. on a USB stick.

ii. Lorsque vous serez à l'étape de simulation avec SpaceStudio (commander run -> execute)

Simulation
Before running the simulation, **open Git Bash** and set the MINGW64_SPACESTUDIO_DIR variable to the location you extracted the MinGW64 toolchain. E.g.:

```
export MINGW64_SPACESTUDIO_DIR='/h/mingw64' # If you extracted MinGW64 to H:/mingw64
```

Then start the start_tcp_server_simulation.sh script, which is at the root of the TP3 directory:

```
./start_tcp_server_simulation.sh
```

Note that you must restart the server every time you rerun the simulation in SpaceStudio. You can close the server by hitting ENTER while in Git Bash.

iii. Lorsque vous serez à l'étape d'implémentation avec SpaceStudio (commander run -> execute)

Implementation
Before starting the Zedboard, **open Git Bash** set the MINGW64_SPACESTUDIO_DIR variable to the location you extracted the MinGW64 toolchain. E.g.:

```
export MINGW64_SPACESTUDIO_DIR='/h/mingw64' # If you extracted MinGW64 to H:/mingw64
```

You should also set the SERVER_IP_ADDRESS variable. Find out your IP address by running ipconfig in CMD, then run:

```
export SERVER_IP_ADDRESS='12.34.56.78' # If the Windows machine's IP is 12.34.56.78
```

Then start the start_tcp_server_implementation.sh script, which is at the root of the TP3 directory:

```
./start_tcp_server_implementation.sh
```

Note that you must restart the server every time you rerun the implementation on the Zedboard. You can close the server by hitting ENTER while in Git Bash.

9.3 Détails concernant la création d'une solution sous SpaceStudio

i. Lorsque vous créez une solution dans Architecture -> Architecture Manager

In SpaceStudio, when you will create the solution to run your code on a Zynq (Zedboard) virtual platform, make sure you select SMP mode, since the master and display modules require Linux headers to run. This is probably going to save you from insanity :-)

ii. Lorsque vous ferez allocation et l'assignation dans Architecture Manager

If you have activated an architecture that has a Zynq on it, you must always make sure...

1....the zynq0 entity is on the AMBA_AXIBus_LT0 bus.

2....any module you want to run on hardware is on the zynq0_HP0_AMBA_AXIBus_LT0 bus. Of course, modules you want to run on software must be under the zynq0 entity.

iii. Simulation et erreurs pouvant survenir lors de la simulation

Simulation

When you want to simulate the code, make sure...

1....the display0 and reception0 modules are in hardware (they use Linux headers to communicate via TCP to the server, and that cannot be done easily when the module runs on the simulated CPU).

2....there is a simulation_timer on the AMBA_AXIBus_LT0 bus.

Simulation troubleshooting

1.If you encounter the error Module <X> is trying to read from device <Y> with an invalid offset <Z> during the simulation, then right click the zynq0_DDR_InterfaceRange0 entity, and edit its High address property to be at least 300 KB higher than <Z>. Normally, setting 0xBFFFFFFF should do the trick.

2.If you wish to debug, *only run hardware debugging*. You cannot run Software debugging, nor Hardware/Software co-debugging. Move your bugged modules to hardware if you want to debug them.

3.If you built the simulation with hardware debugging, you can't run the simulation without hardware debugging (which you can run by clicking *Run*, then *Debug*, then *Hardware...*). Clicking the green *Play* button will not work, even though SpaceStudio gives you the option.

9.4 Détails concernant la création d'une implémentation sous SpaceStudio

i. Avant de procéder à l'implémentation (e.g., dans Solution -> Architecture Manager)

Module assignment

When you wish to implement, you must make sure...

1....you have a Zynq on the AMBA_AXIBus_LT0 bus.

2....to delete the simulation_timer.

3....that the display0 and reception0 modules are in software, that is, under the zynq0 entity (this is because these modules do Linux networking stuff, which obviously doesn't synthesize in hardware)

4....that your hardware modules are under the zynq0_HP0_AMBA_AXIBus_LT0 bus.

5....that you changed the IP address in reception.cpp and display.cpp to the Windows machine's IP address (inside the #elsearound line 27 for reception.cpp and also inside the #else around line 34 for display.cpp). It takes ~20 minutes for the Architecture Implementation to run, so it's a bit annoying when you realize you forgot to change this and you have to rerun it again ¯_(ツ)_/¯ .

ii. Avant d'exécuter tools -> Architecture Implementation

Running *Architecture Implementation*

1.Before clicking Architecture Implementation, make sure Vivado HLS and Vivado are enabled by going to *Tools*, then *Preferences*, then, under *SpaceStudio* > *EDA* > *Xilinx - Vivado 2018.3*, make sure the *"EDA is enabled"* and *"High-level synthesis is enabled"* boxes are checked, and set the path to *"C:/Logiciels/Xilinx"*.

2.When you click Architecture Implementation, make sure the popup menu shows: *"Xilinx - Vivado 2018.3* for *Electronic Design Automation (EDA) tool"*, *"ZedBoard Zynq Evaluation and Development Kit"* for *"Board"*, *"Vivado HLS"* for *"High-level synthesis"*, and make sure all the modules are checked. You can leave the default value for *"Project directory"*. Click OK.

9.5 Détails concernant la Zedboard

Changing our MAC address and setting a static IP

Once that the *Architecture Implementation* is done, that you've started the server and that you've started the Zedboard, you can connect to the Zedboard by opening the PuTTY program, and setting these configurations:

- Connection type: Serial
- Serial line: either COM3 or COM4 ; go to *"Gestionnaire de périphériques"*, expand *"Ports (COM et LPT)"* and search for *"USB Serial Port"*.
- Speed: 115200

In the window that opens, you will see any output that the Zedboard prints, and you will be able to type commands to the Zedboard with your keyboard. If you have started the Zedboard long before you started the PuTTY window, you might not see any output on the PuTTY window, but that's normal (only *new* ouptut it printed to the PuTTY window once it's open). You can still hit CTRL+C to stop the program that is running when the Zedboard boots.

Decide on a MAC and an IP address

We must assign an IP address manually to the Zedboard (Poly decided that the Zedboard's LAN wouldn't have a DHCP on it). We must also assign a MAC address manually (so that we don't have two Zedboards with the same MAC address in the lab).

Choose an IP address. If you chose a team number during the first session of TP, use 132.207.89.(100 + <team number>), e.g. 132.207.89.110 for team 10. Otherwise, your IP must be in the range 132.207.89.20 to 132.207.89.99 and cannot be the same as the IP another student in the lab room chose. **Be careful, the students of INF3995 ("Projet 3") also use** 132.207.89.[20-99]**addresses.** Please don't make them sad by using the same IP address as they do while they are in the room.

The MAC address you should use is 00:18:3E:00:00:XY, where XY are the last two digits of the IP address you chose. For example, if you chose 132.207.89.56 as IP, then your MAC should be 00:18:3E:00:00:56.

Changing the MAC

In the PuTTY window, run:

```
ifconfig eth0 down
ifconfig eth0 hw ether <MAC address>
ifconfig eth0 up
```

To make sure eth0's MAC was changed, run ip addr.

NOTE: If you restart the Zedboard, you will need to run these commands again.

Changing the IP

To assign the IP address we should edit the /etc/network/interfaces file. You can edit it by running the command:

```
vi /etc/network/interfaces
```

Once in vi, hit ESCAPE then i to go into insert mode, then change the file to look like this:

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
        address <IP address you chose>
        netmask 255.255.255.0
        gateway 132.207.89.1
        network 132.207.89.0
        dns-nameservers 8.8.8.8
```

Once this is done, hit ESCAPE to go out of insert mode, then write :wq to write changes and quit.

We can the apply the new configuration by running:

```
ifdown eth0
ifup eth0
```

You can verify that your IP address was assigned by running ip addr.

NOTE: Files you create on the Zedboard are volatile (or something like that), so you will need to edit /etc/network/interfaces *and run* ifdown eth0; ifup eth0 *again if you reboot the Zedboard.*

Make sure you have internet connection

Run:

ping 8.8.8.8

If you see some lines like 64 bytes from 8.8.8.8: seq=<X> ttl=<Y> time=<Z>, then the Zedboard has internet access. Otherwise, you might have missed a step.

Running the program

Once you've changed both the MAC and the IP, you can run the zynq0.core0.arm_a9.elf executable to observe the results of our SpaceStudio project. Run:

```
cd /root
./zynq0.core0.arm_a9.elf
```