

École Polytechnique De Montréal

INF3610 Laboratoire 4

Partie 1 de 2

Conception logicielle/matérielle à partir d'un modèle du Zynq

Frédéric Fortier & Arnaud Desautly
13/03/2016

Table des matières

Introduction.....	2
Objectifs du laboratoire	2
I. Familiarisation avec SpaceStudio	3
1. La librairie SpaceLib.....	3
a. Modules	3
b. API de communication	5
2. L'interface utilisateur : SpaceStudio	6
a. Fenêtre principale.....	6
b. Configurations.....	7
c. Technologies de simulation	9
d. Génération du projet	11
e. Affichage des résultats du monitoring.....	11
II. Travail à effectuer	11
1. Tutoriel du JPEG	11
2. Implémentation de l'algorithme de Sobel	11
Introduction à l'algorithme	11
Implémentation sur SpaceStudio	13
Modules à compléter ou réaliser	13
Références.....	15

Introduction

La conception de système sur puce ou sur FPGA peut être un processus long et coûteux. Dans un contexte où les temps de mise sur le marché sont de plus en plus court et la complexité des systèmes grandit, les méthodes traditionnelles de conception ont beaucoup de mal à suivre le rythme. Il est donc nécessaire de penser à de nouvelles méthodes de conception pour de tels systèmes. La nécessité d'effectuer des simulations à un haut niveau d'abstraction apparaît donc comme une solution intéressante à ce problème. En effet, avec des solutions telles que SystemC, il est possible de valider un design à différents niveaux d'abstraction, en commençant par une solution fonctionnelle, puis en affinant la solution jusqu'à obtenir une solution synthétisable et donc pouvant être implémentée.

À ce principe d'abstraction à haut niveau vient s'ajouter un second problème, celui de l'exploration architecturale. En effet, le design défini plus haut doit être appliqué à une architecture matérielle définie. Cette plateforme peut être un simple FPGA, une carte de développement avec processeurs et FPGA ou bien encore un ASIC. Il convient alors de fragmenter le design haut niveau et de l'appliquer aux ressources matérielles disponibles. Ce mappage est un travail très complexe qui peut nécessiter des quantités importantes de temps et d'expertise, surtout quand des tests doivent être effectués sur un prototype après synthèse du code.

Cette double nécessité de vérification à haut niveau de la fonctionnalité et de l'architecture a conduit à la naissance des outils de co-design logiciel/matériel. Ces outils permettent un flot de conception itératif, de la validation du code à son implémentation logiciel ou matérielle, à l'intérieur d'un seul outil. La simulation à haut niveau du *mapping* à l'aide des simulateurs de jeux d'instructions et de langages de description matérielle permettent d'explorer plus efficacement l'espace architectural.

Objectifs du laboratoire

Au cours de ce laboratoire, nous allons vous présenter un logiciel de co-design: SpaceStudio™. Après un cours tutorial sur l'utilisation de cet outil, il vous sera demandé d'implémenter un algorithme en suivant le flot de conception de SpaceStudio. Puis vous devrez explorer l'espace de solution afin de rendre votre nouvelle fonctionnalité la plus rapide possible.

L'objectif de ce laboratoire consiste à:

- introduire l'étudiant à la conception de SoC à haut niveau d'abstraction,
- initier l'étudiant au fonctionnement d'un logiciel de co-design et à son flot de conception,
- valider un système avec la co-simulation logicielle/matérielle,
- exposer l'étudiant à une brève exploration architecturale visant à améliorer l'efficacité de votre solution¹.

¹ En ce sens, il s'agit d'une petite introduction au cours INF8500 (Systèmes embarqués : Conception et vérification)

I. Familiarisation avec SpaceStudio

Pour cette introduction aux fonctionnalités de SpaceStudio, nous allons nous intéresser au code qui vous est fourni : le projet Sobel, qui sera détaillé plus bas.

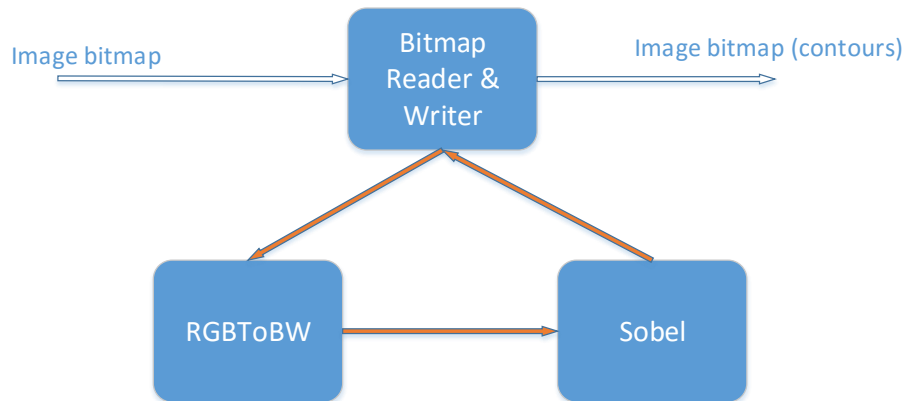


Figure 1: Modèle du projet Sobel

1. La librairie SpaceLib

a. Modules

Les encadrés bleus dans la figure 1 sont des **modules**. Ces modules représentent des parties de la fonctionnalité complète du système. Un module SpaceStudio est une classe C++ qui hérite de la classe *SpaceBaseModule*

```
#ifndef RGBTOBW_H
#define RGBTOBW_H

#include "SpaceBaseModule.h"
#include "systemc"

class RGBToBW: public SpaceBaseModule
```

Code 1 : Déclaration d'un module (RGBToBW.h)

Comme vous pouvez le voir dans le fichier *RGBToBW.h*, le constructeur du module *RGBToBW* appelle le constructeur de *SpaceBaseModule*. Vous pouvez aussi remarquer que nous incluons dans le header *systemC.h*. Cela est dû au fait que la librairie de SpaceStudio (appelée SpaceLib) est basée sur SystemC. Cela est encore plus visible avec la *SC_THREAD(thread)* dans le constructeur ainsi que *SC_HAS_PROCESS(RGBToBW)*.

La méthode *thread* doit décrire l'intégralité du fonctionnement du module. Elle est souvent définie sous la forme : Récupération de données, travail sur ces données, envoi des données.

Ainsi un module SpaceStudio est une classe C++:

- basée sur la classe *SpaceBaseModule*, elle-même basée sur un module SystemC
- possédant un *SC_THREAD* (*uniquement !*)
- Qui récupère des données, travaille dessus et renvoie un résultat

Pour plus d'informations sur les modules, vous pouvez consulter le fichier d'aide de SpaceStudio

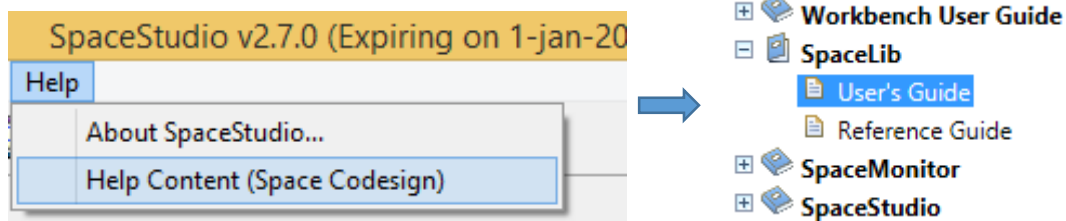



Figure 2 : Accès à la documentation de SpaceLib

b. API de communication

Afin d'envoyer et de recevoir des données entre les modules, SpaceLib définit une API de communication. Les fonctions de cette API qui seront le plus utilisées sont *moduleRead* et *moduleWrite*. Ces fonctions permettent la communication entre deux modules par le biais d'une FIFO. La communication doit donc être effectuée des deux côtés, c'est à dire qu'une écriture doit toujours aller de pair avec une lecture. Lorsque l'écriture et la lecture sont bloquantes, la rencontre des deux se nomme *rendez-vous* et permet l'échange de données et la synchronisation des modules (semblable aux queues en μ C). A noter que chaque module possède son propre ID dans SpaceStudio.

L'API de SpaceStudio permet d'autres types de communications que vous pouvez retrouver dans le manuel utilisateur (cf. Figure 2) mais ceux-ci ne seront pas vus dans le cadre de ce laboratoire.



```
Status = Operation ( Destination ID Tag,
                      Delay,
                      Pointer to message,
                      Number of message elements);
```

Parameter	Description
Operation	ModuleRead or ModuleWrite
Destination ID Tag	ID or tag identifying the destination component (module). This ID should be a constant.
Delay	Delay determines whether a communication is blocking or not: SPACE_BLOCKING or SPACE_NON_BLOCKING
Message	Message buffer address to send or to receive (a pointer)
Number of elements of message	When the message to send is stored into a data array, this parameter identifies the number of elements in that array. The default value is 1 and does not need to be provided to transmit a single value to transmit or data structure.
Status	Returns the status of the current communication operation: SPACE_OK, SPACE_EMPTY, SPACE_FULL, SPACE_ERROR.

Tableau 1 API de communication FIFO (tirée du manuel utilisateur [1])

2. L'interface utilisateur : SpaceStudio

a. Fenêtre principale

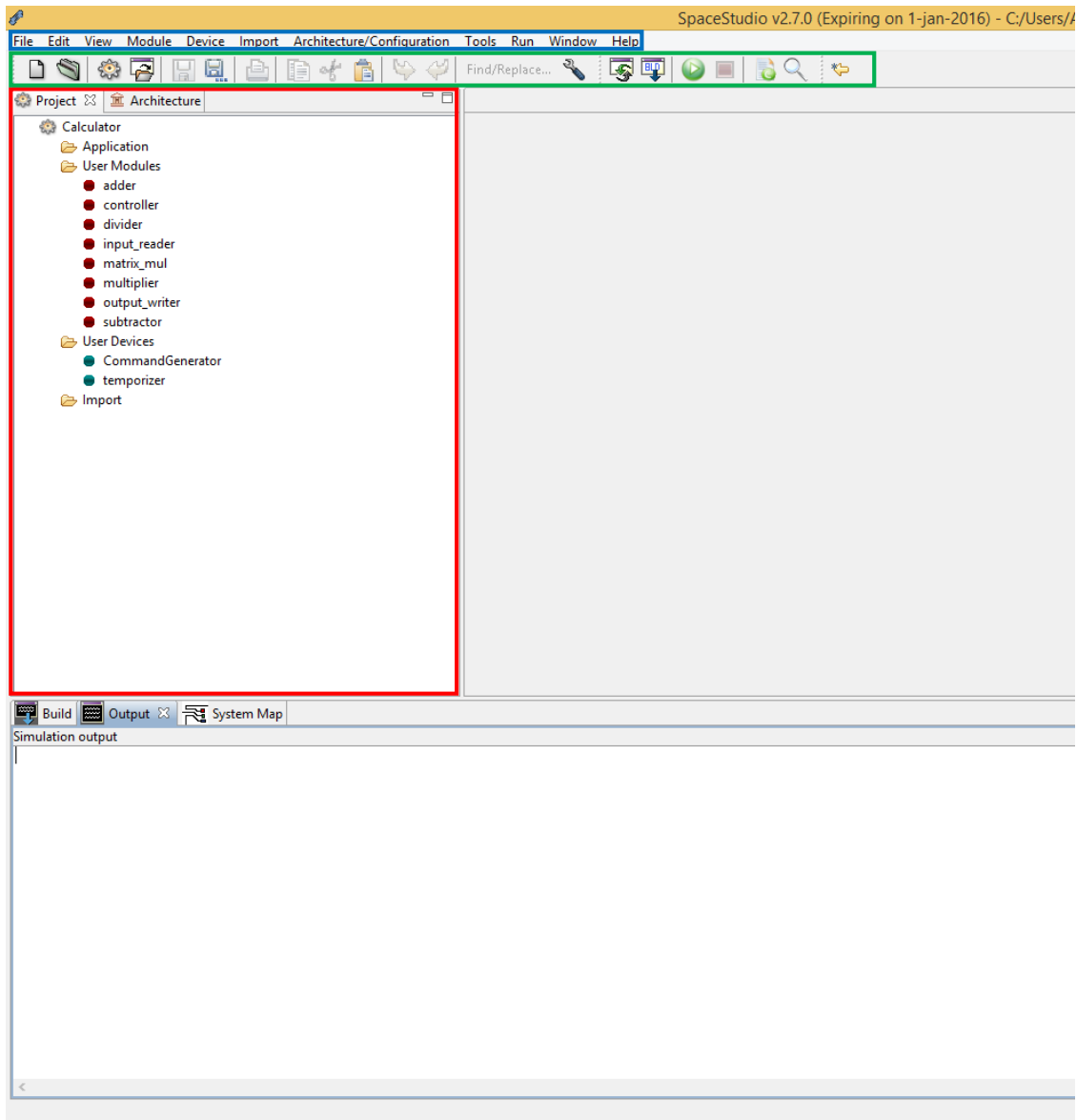




Figure 3 Fenêtre principale de SpaceStudio

La fenêtre principale de SpaceStudio se décompose en plusieurs éléments importants :

1. **La fenêtre projet/architecture** : la perspective *Project* vous permet d'afficher les sources que vous utilisez dans votre code (que celles-ci soient des modules, des *devices* ou du code C++). La perspective architecture devient disponible lorsque vous chargez une configuration et montre le partitionnement de la configuration (c'est-à-dire le partitionnement module/ressources)
2. **La barre d'outils** : vous permet d'effectuer les actions principales d'un IDE classique. Notez l'icône  qui vous permet d'ouvrir le gestionnaire de configuration ainsi que l'icône 

qui vous permet de lancer la simulation pour la configuration en cours (une fois que celle-ci a été générée et compilée).

3. **La barre de menus** : C'est depuis cette barre que vous avez accès au reste des fonctionnalités de SpaceStudio, comme l'ajout de module, de *devices* ou l'importation de sources. Vous pouvez aussi créer et sélectionnez la configuration active. C'est aussi depuis ce menu que vous aurez accès aux outils de génération du code (dans Tools.), aux outils de profilage.

b. Configurations

Pour commencer à travailler dans SpaceStudio, vous devez sélectionner une configuration. Une configuration contient les informations sur la technologie de simulation que vous voulez utiliser, la plateforme matérielle sur laquelle vous voulez tester votre solution ainsi que votre mappage des modules à vos ressources.

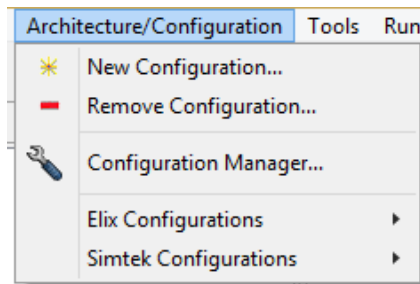


Figure 4 Menu de gestion des configurations

Pour choisir une configuration active, il suffit de dérouler la liste des configurations Elix ou Simtek et de sélectionner une des configurations déjà créées. Pour modifier celle-ci, il faut entrer dans le *configuration manager* :

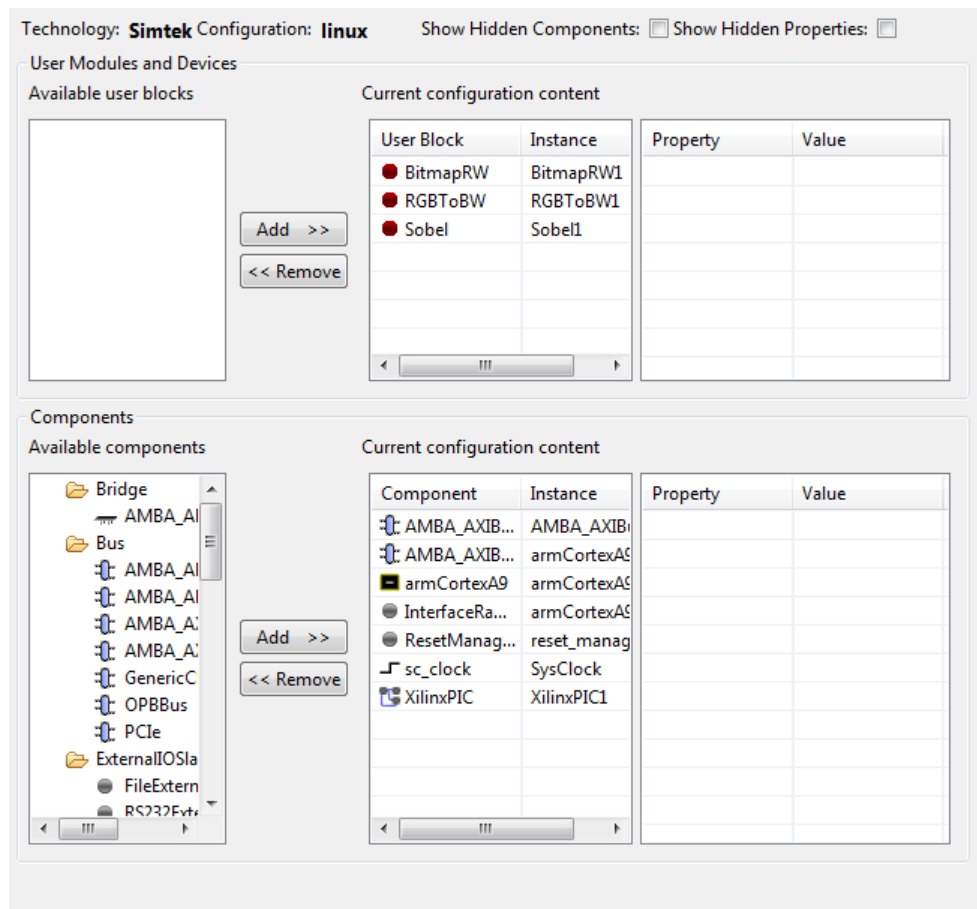


Figure 5 Première fenêtre du Configuration Manager

Le *configuration manager* est séparé en deux parties. *User modules and devices* est la première partie et vous permet de choisir quels modules de votre projet vous voulez introduire dans votre configuration. Seuls ces modules seront actifs pour la simulation.

La deuxième partie *Components* vous permet d'instancier les composants qui composent votre plateforme matérielle. Selon la technologie de simulation utilisée, les composants disponibles ne seront pas les mêmes. Plusieurs technologies sont supportées dont celle du Zynq de Xilinx que vous avez utilisée au laboratoire no 2.

Une fois votre architecture composée, vous pouvez cliquer sur suivant pour accéder au partitionnement *modules/Architecture* :

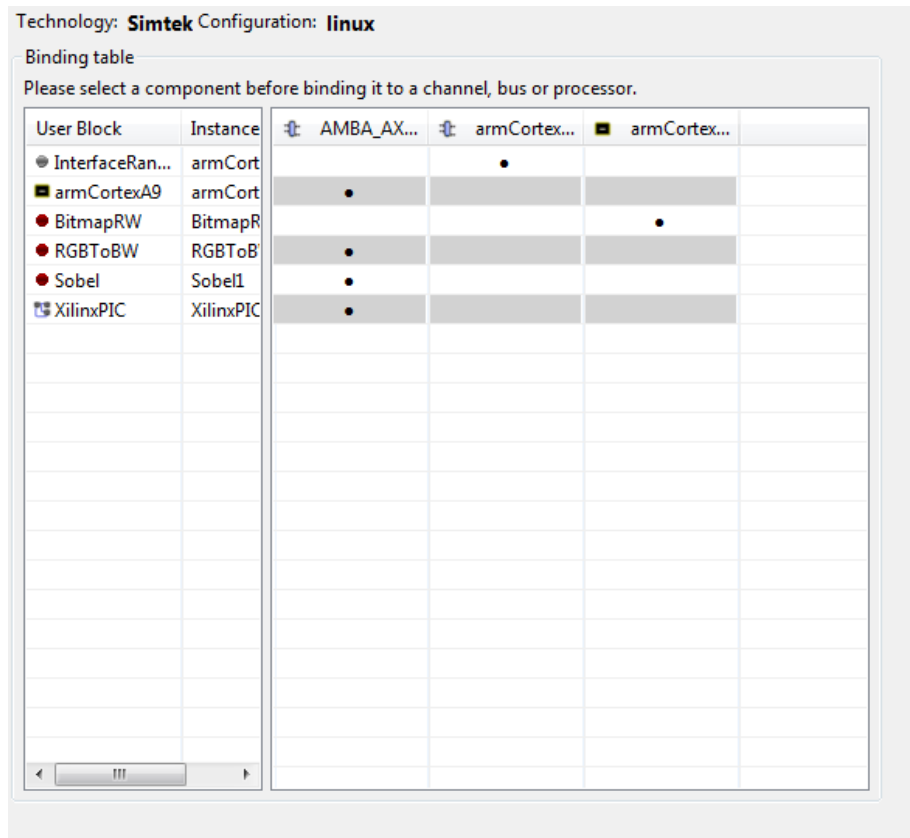


Figure 6 Mappage des modules

Dans cette table, vous pouvez choisir à quelle composante de votre architecture vous voulez connecter vos différents modules. Ainsi, le module contrôleur est assigné au processeur Arm Cortex et tournera donc en logiciel pour la simulation. Le reste des modules est assigné au bus de communication et sont donc par défaut considérés comme des modules matériels.

c. Technologies de simulation

Comme cité dans l'introduction, SpaceStudio possède plusieurs technologies de simulation afin de créer une approche incrémentale lors du design d'un SoC. Ainsi, les deux technologies auxquelles vous aurez à faire sont Elix et SimTek.

Elix permet la vérification fonctionnelle de vos modules et ne requiert pas de plateforme matérielle. Il suffit juste d'un bus de communication qui relie tous les modules. Faire tourner votre code sous Elix vous permet de valider votre design avant de passer à l'exploration architecturale.

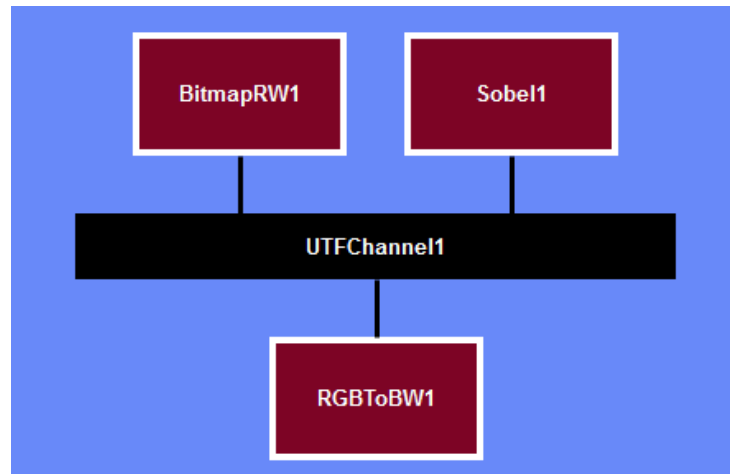


Figure 7 Schéma du système sous Elix

Simtek vous permet, quant à lui, de simuler votre code en rapport avec une architecture matérielle donnée (par exemple une abstraction de la technologie du Zynq). Ainsi, la façon dont vous mapperiez vos modules à votre architecture influencera les performances du programme.

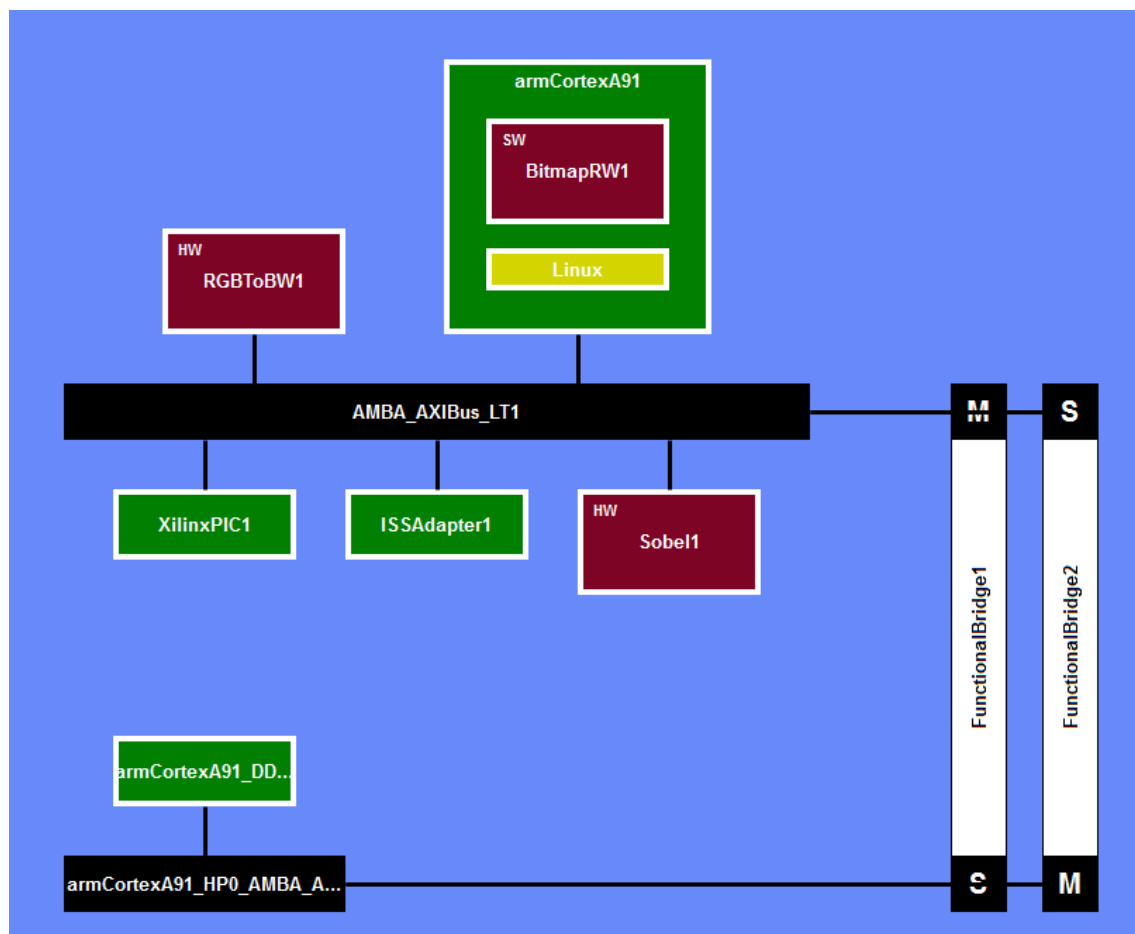


Figure 8 Schéma du système sous Simtek

d. Génération du projet

Pour pouvoir lancer votre configuration, vous avez plusieurs étapes à suivre.

Vous devez tout d'abord générer les fichiers nécessaires à la compilation. Pour ce faire, lancez *generate* depuis le menu Tools. Lors de la génération vous avez plusieurs options :

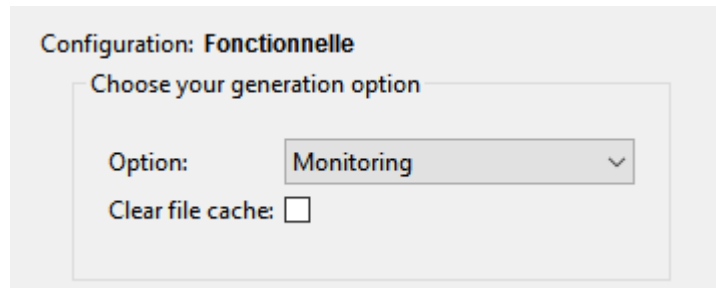


Figure 9 Menu generate

Les options qui nous intéressent sont les options *Debug* et *Monitoring*. *Debug* permet l'utilisation du debugger tandis que *monitoring* va rajouter de l'instrumentation au code afin de pouvoir analyser des métriques comme l'utilisation du processeur. Bien entendu, le monitoring va augmenter le temps nécessaire à la simulation.

Une fois le projet généré avec l'option de votre choix, il faut maintenant compiler. Pour ce faire, il faut cliquer sur Tools -> Build -> All.

Enfin une fois le projet compilé, vous pouvez cliquer sur  pour lancer la simulation.

e. Affichage des résultats du monitoring

Une fois la simulation terminée, vous pouvez récupérer les résultats du monitoring en ouvrant le fichier Excel *MonitoringExplorer*, situé dans `C:\SpaceCodeSign\SpaceStudio-2.8.0\util\MonitoringExplorer` puis en ouvrant la base de données produite par la simulation qui se trouvera dans `<votre projet>\Sobel\Elix\Fonctionnelle\Fonctionnelle\build\monitoring`

II. Travail à effectuer

1. Tutoriel du JPEG

Afin de vous familiariser avec les commandes décrites ci-haut, nous vous recommandons fortement de faire le tutoriel sur le JPEG. Une fois que vous aurez complété le tutoriel, vous pourrez passer à l'étape suivante. Notez que votre chargé de laboratoire ne répondra à aucune question dont la réponse peut être trouvée en faisant le tutoriel.

2. Implémentation de l'algorithme de Sobel

Introduction à l'algorithme

L'algorithme de Sobel est un algorithme couramment utilisé en traitement d'images pour extraire les contours de celle-ci. Il fonctionne en approximant la dérivée de l'image (les contours

étant nécessairement aux endroits où cette dérivée est plus grande) à l'aide de deux tables pré-calculées de coefficients données au tableau 2.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Tableau 2 : Masques de coefficients de l'algorithme de Sobel

L'algorithme consiste donc à attribuer une valeur, pour chaque pixel de l'image, égale à la somme de la valeur des pixels avoisinants multipliés par le coefficient correspondant, ce qui permet de passer d'une image comme celle de la figure 11 à une détection de contours comme celle de la figure 12.

Les étudiants intéressés à avoir plus de détails sur l'algorithme ou le traitement d'image en général sont invités à consulter la page Wikipedia dédiée à celui-ci et/ou à suivre le cours INF4725 : Traitement de signaux et d'images, ce laboratoire ayant plutôt pour but de montrer le flot de conception d'une solution matérielle/logiciel à un problème donné.



Figure 10: Image originale

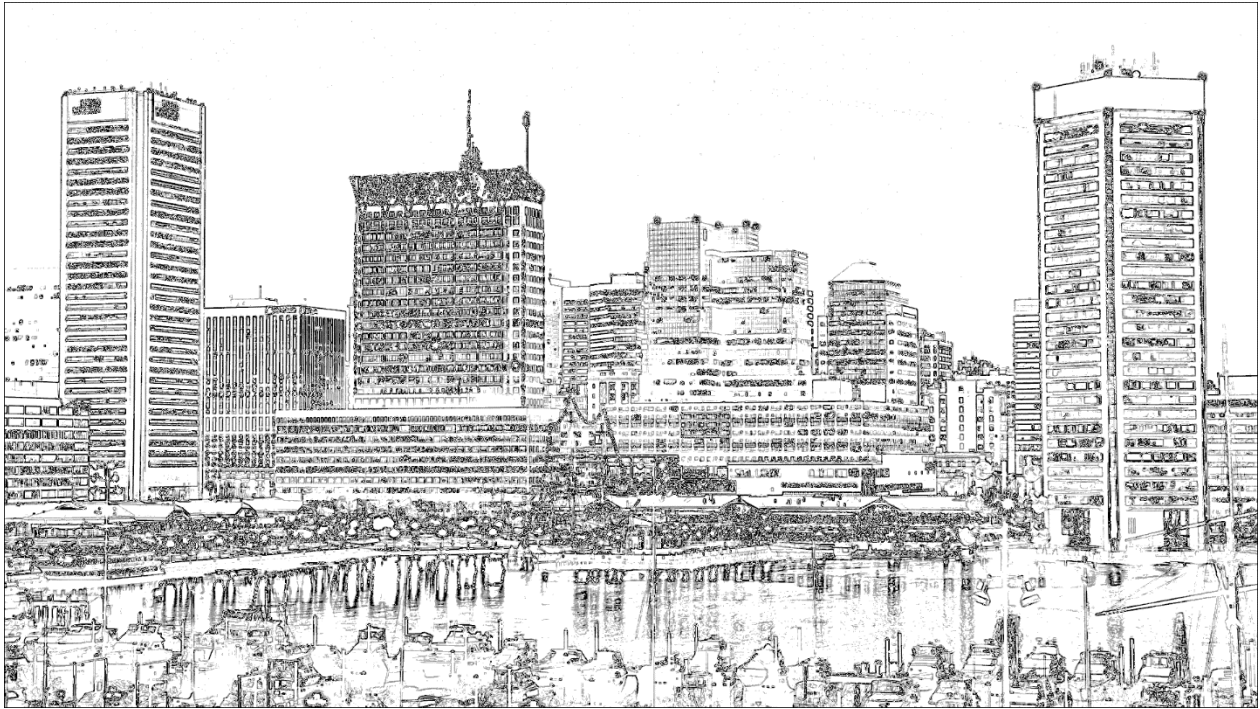


Figure 11: Résultat de l'application de l'algorithme de Sobel sur l'image précédente

Implémentation sur SpaceStudio

Vous devez prendre connaissance du projet *Sobel* qui vous est fourni. Ce projet est fonctionnel et contient deux configurations² : une configuration Elix appelée Fonctionnelle et une configuration SimTek appelée linux. Chacune de ses configurations auront, à terme, 3 modules différents.

Pour la première partie du laboratoire, **vous n'avez qu'à faire fonctionner l'application en simulation fonctionnelle (Elix)**. La simulation temporelle (Simtek), de même que la synthèse, l'extraction de mesures de performances, l'optimisation, etc. seront demandés dans la deuxième partie du laboratoire, dont l'énoncé sortira aux alentours du 20 mars.

Modules à compléter ou réaliser

Note : Référez-vous à la figure 1 pour avoir une compréhension plus visuelle des communications entre chaque module.

BitmapRW

Le rôle du module *BitmapRW* est de lire une image bitmap fournie en entrée, de la convertir sous un format utilisable (tableaux de pixels RGB), d'envoyer ceux-ci au module matériel *RGBToBW* qui convertira les données couleur en un tableau noir et blanc, de récupérer les données de contours du module *Sobel* et d'enregistrer la solution sur le disque.

² La configuration Simtek n'est fournie qu'à titre de référence et variera probablement lorsque la partie 2 du laboratoire sortira. Vous n'en avez pas besoin pour réaliser la première partie du laboratoire.

Le code permettant la lecture/écriture des fichiers bitmaps (de même que la comparaison du résultat avec une référence (*Golden Model*)) est fourni. Vous n'avez qu'à ~~vous~~ implémenter les communications avec les autres modules.

RGBToBW

Ce module assez simple est à concevoir entièrement (via *Module->New Module* puis en l'ajoutant aux configurations via le *Configuration manager*) et sert à convertir l'image RGB (fournie sous forme de tableaux par le module *BitmapRW*) en image en niveaux de gris, puisque c'est sur ce type d'image que l'algorithme de Sobel fonctionne, étant donné que la couleur n'a pas d'importance dans la détermination de contour. Elle envoie ensuite l'image en niveau de gris au module *Sobel*.

Bien que l'on puisse être tenté de convertir l'image couleur en niveaux de gris en ne faisant que la moyenne de ses trois composantes rouge, bleue et verte, nous préférons ici aller chercher la luminance de l'image, très semblable mais corrigeant chaque composante de couleur de manière à donner une image noire et blanc mieux perçue par l'oeil humain (ce principe est entre autres à la base de la représentation *YUV* d'une image). Ces détails n'étant pas particulièrement importants pour la réalisation du laboratoire, l'équation à utiliser (qui fait l'opération par calcul sur des entiers uniquement pour une implémentation matérielle efficace) est fournie ici :

```
// RGB to Y Conversion
// Resulting luminance value used in edge detection
static inline uint8_t rgb2y(uint8_t R, uint8_t G, uint8_t B)
{
    return ((66 * R + 129 * G + 25 * B + 128) >> 8) + 16;
}
```

Sobel

Ce module s'occupe d'exécuter l'algorithme de Sobel sur l'image en niveau de gris fournie, puis de transférer le résultat au module *BitmapRW*. Les paramètres de seuillages sont fournis, vous n'avez qu'à implémenter le noyau de l'algorithme et les communications. Notez que nous supposons que la bordure (les premières et dernières lignes et colonnes) de l'image ne contient pas de contour : nous ignorerons donc ce cas particulier.

Notes importantes

- Dans le constructeur du module RGBToBW que vous créerez, **vous devez** ajouter un appel à la fonction `set_stack_size(0x16000+(IMG_SIZE*n));`, *n* étant le nombre de tableaux d'une couleur de l'image. Assurez-vous aussi de déclarer l'espace pour ces tableaux dans votre fonction *thread* et **non** dans l'en-tête (*RGBToBW.h*) de la classe comme attribut privé.
- La taille des images que l'implémentation supporte est fixe et définie dans le fichier *ApplicationDefinitions.h*. Vous devez modifier cette taille si vous voulez essayer votre implémentation sur une image de taille différente.

Références

1. [XAPP890](#) : *Zynq All Programmable SoC Sobel Filter Implementation using the Vivado HLS Tool*
2. Wikipedia: [Sobel operator](#)
3. Wikipedia: [YUV](#)
4. Tanner Helland: [Seven grayscale conversion algorithms](#)