



Printemps 2021-2022

Amer Baghdadi

SAR IP : Image Processor

**Conception d'un processeur dédié à la détection de contours :
implémentation matérielle**

Travaux pratiques réalisés par :

Nom

Prénom



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TABLE DES MATIERES

1	Introduction	1
2	Unité opérative du processeur	3
2.1	Projet Vivado	3
2.2	Banc de registres : description VHDL, simulation et synthèse logique	3
2.3	Unités fonctionnelles : description VHDL, simulation et synthèse logique	5
2.4	Registre de sortie	7
2.5	Assemblage des sous-composants de l'unité opérative	7
3	Générateur d'adresses	9
3.1	Architecture et description VHDL du générateur d'adresses	9
3.2	Simulation du générateur d'adresses	10
3.3	Synthèse logique du générateur d'adresses	10
4	Automate ou machine à états finis	11
4.1	Spécification, description VHDL et simulation de l'automate	11
4.2	Synthèse logique de l'automate	13
5	Intégration du processeur et prototypage	14
5.1	Intégration de l'architecture complète du processeur de Sobel	14
5.2	Prototypage et démonstration sur carte	15
6	Analyse et comparaison des performances	17
7	Bonus – pour aller plus loin	20

1 INTRODUCTION

Suite à l'implémentation logicielle développée dans la première séance de travaux pratiques, nous nous intéressons dans la suite à la conception d'une architecture matérielle en vue d'accélérer les performances de la détection de contours en termes de temps d'exécution.

Une proposition incomplète d'une telle architecture a été introduite en cours (**Figure 1**) avec deux composants principaux :

- ▶ **Une unité opérative** qui intègre des registres et des unités fonctionnelles (opérateurs arithmétiques et logiques).
- ▶ **Une unité de contrôle** qui intègre un automate ou machine à états finis et un générateur d'adresses.

En plus de composants qui constituent le cœur du processeur, le système complet de détection de contours intègre des interfaces et des mémoires d'entrée/sortie, dont notamment un contrôleur d'affichage sur écran VGA.

Ainsi, l'objectif de cette série de trois séances de travaux pratiques (en mode projet) est de développer et de mettre en œuvre cette solution d'accélération matérielle. Les différentes étapes du flot de conception associé seront illustrées : (a) modélisation et développement des différents composants de cette solution matérielle, (b) synthèse logique, intégration et prototypage sur carte FPGA, (c) mise en œuvre de test d'intégration, caractérisation et validation des performances.

Le projet doit se terminer par une phase d'analyse des performances et de comparaison des différentes solutions logicielles/matérielles mises en application.

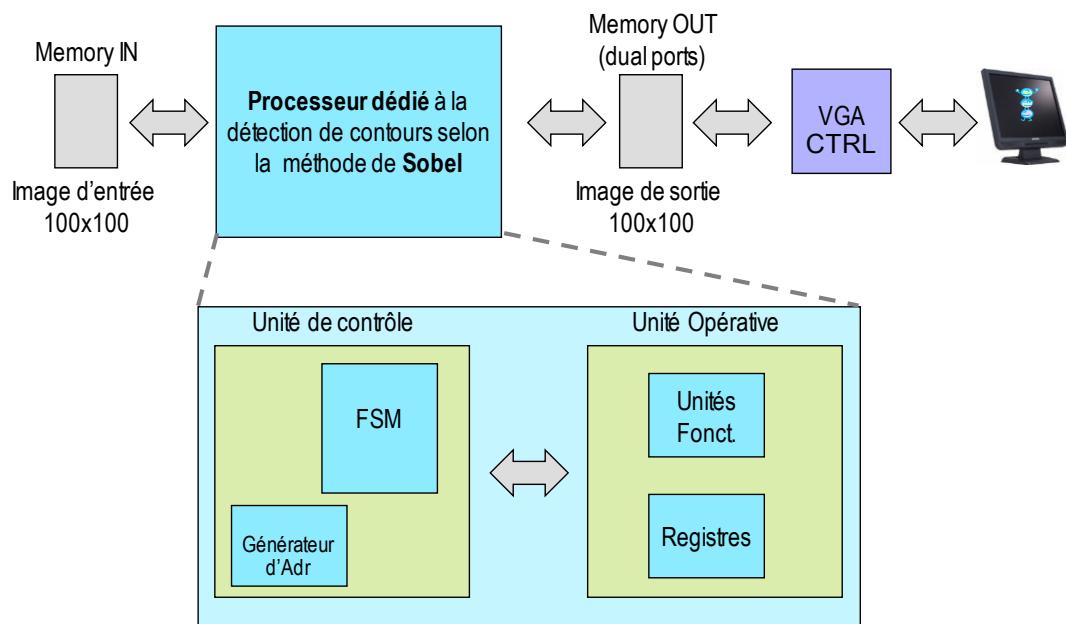


Figure 1 : Processeur dédié à la détection de contours

La plateforme matérielle qui sera utilisée pour le prototypage est basée sur la carte Digilent Nexys 4 représentée en **Figure 2** et comprenant un FPGA Xilinx Artix 7 (référence XC7A100T-CSG324) et de nombreux périphériques et interfaces dont notamment une sortie VGA.

L'environnement de développement associé à cette carte sera la suite Vivado de Xilinx.

Concernant le dimensionnement de l'architecture, les simulations et les expérimentations sur carte, nous allons considérer des images d'une définition de 100x100 codées sur 8 bits en niveau de gris.

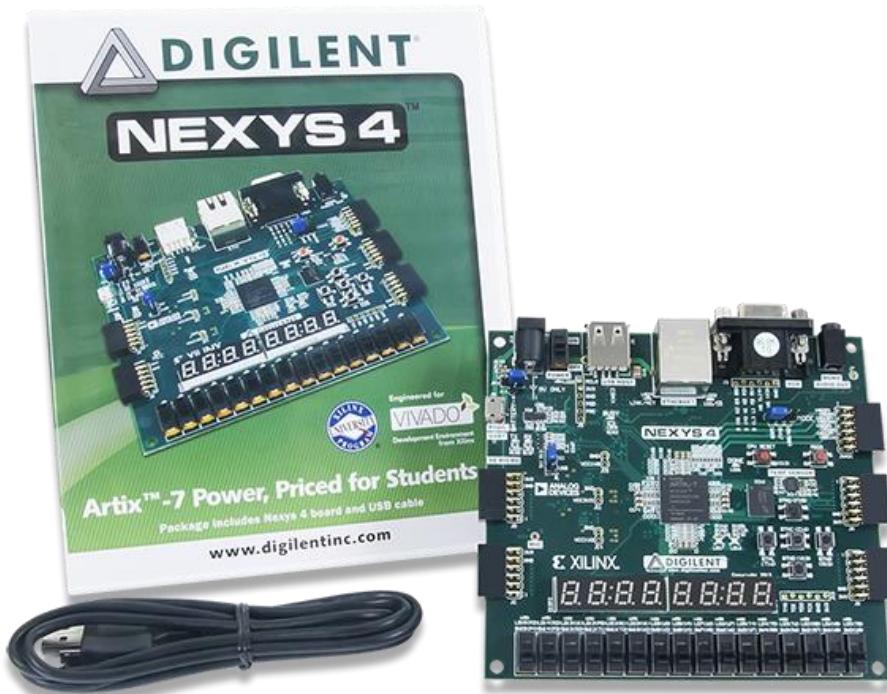


Figure 2 : Carte Digilent Nexys 4

Nous proposons le **planning** suivant des développements à effectuer à chaque séance. Ce planning est prévisionnel et peut être adapté selon l'avancement de chacun. En particulier, quelques idées de développements additionnels sont proposées en bonus pour ceux qui réussissent à prendre de l'avance par rapport à ce planning !

PASS 18 mai (13h45 à 16h30)	Conception et validation de l'unité opérative du processeur
PASS 25 mai (13h45 à 16h30) – Autonomie	Conception et validation du générateur d'adresses
PASS 01 juin (13h45 à 16h30)	Finalisation du générateur d'adresses Conception et validation de l'automate
PASS 08 juin (13h45 à 16h30) – Autonomie	Intégration du processeur, débogage et validation
PASS 15 juin (13h45 à 16h30)	Prototypage sur carte – Démonstration
PASS 22 juin (13h45 à 16h30)	Analyse et comparaison des performances

2 UNITE OPERATIVE DU PROCESSEUR

2.1 PROJET VIVADO

Nous allons débuter le développement du processeur dédié en commençant par l'unité opérative. Dans un premier temps, il faudra lancer l'outil Vivado et ouvrir le projet fourni par l'enseignant sur la page Moodle de la SAR.

Téléchargement de l'archive source

Une archive de projet Vivado contenant plusieurs fichiers sources vous est proposée sur la page Moodle de la SAR. Cette archive contient plusieurs circuits et modules de tests décrits en langage de description matérielle (VHDL), dont certains restent à **compléter**.

Une fois téléchargée, l'archive doit être décompressée dans un répertoire de travail que vous aurez judicieusement créé dans l'arborescence de répertoires de votre compte informatique.

Ouverture de Vivado

Sous Linux, ouvrez un Terminal et lancez les commandes suivantes :

```
> SETUP ELEC_VIVADO20192  
> vivado &
```

L'interface graphique de Vivado démarre. Ouvrez le projet que vous venez de télécharger (*sobel.xpr*) et identifier les différents modules de l'architecture du processeur.

2.2 BANC DE REGISTRES : DESCRIPTION VHDL, SIMULATION ET SYNTHESE LOGIQUE

Pour modéliser l'unité opérative, nous allons opter pour une description structurelle pour une meilleure prise en main du flot de développement. Dans cette approche, nous allons découper l'unité opérative en trois sous-composants comme illustré dans la **Figure 3**. Une fois les sous-composants créés, il faut les assembler. L'utilisation d'un sous-composant dans l'unité opérative est appelée une instanciation, on fait appel à une instance du sous-composant. Une fois les sous-composants instanciés, il faut les connecter aux entrées/sorties/signaux de l'unité opérative.

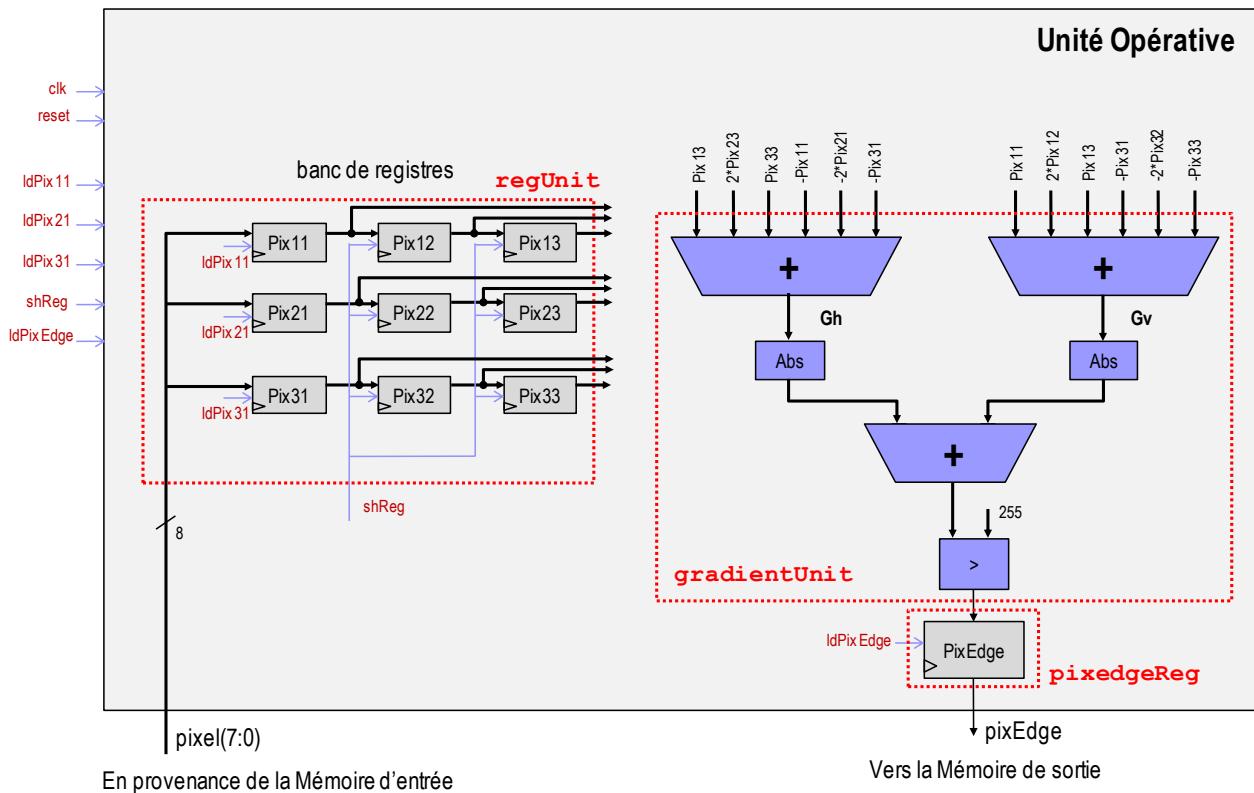


Figure 3 : Architecture de l'unité opérative

Complétez le fichier *regUnit.vhd* pour décrire le banc de registres en respectant les règles pratiquées lors des travaux pratiques en UE Electronique (en première année). Commentez votre code. Un document d'introduction au VHDL incluant de nombreux exemples est disponible sur Moodle.

Validez le fonctionnement du banc de registres par simulation en utilisant le module de test (testbench) fourni *tb_regUnit.vhd*.

Un testbench est un fichier VHDL (ou Verilog, ...) qui spécifie des stimuli (entrées, horloges) à une Unité Sous Test (Unit Under Test, *UUT*, en anglais) afin de valider son comportement pour les différentes combinaisons possibles aux entrées.

Quelle est la nature du processus VHDL qui décrit le banc de registres ? sa liste de sensibilité ? La simulation fonctionnelle a permis de valider le module ? justifiez.

Le processus VHDL qui décrit le banc de registre est séquentielle, sa liste de sensibilité est reset, clk. La simulation fonctionnelle a permis de valider le module. En effet, lorsque I_dIPix11 est sur 1 au front montant de l'horloge, S_Pix11 prend la valeur de I_pixel, de même pour les autres lignes. Lorsque I_shReg est sur '1' au front montant de l'horloge, tous les pixels se déplacent à droite.

Effectuez une synthèse logique et relever les performances obtenues en termes de ressources occupées (Flip-Flops, LUTs) sur le FPGA cible considéré.

Donnez les résultats obtenus.

Les performances obtenues en terme de ressource sont les suivantes :
LUTs : 0, Flip-flops : 72.

2.3 UNITES FONCTIONNELLES : DESCRIPTION VHDL, SIMULATION ET SYNTHESE LOGIQUE

Les mêmes étapes de description VHDL, de simulation et de synthèse logique doivent maintenant être appliquées au deuxième composant de l'unité opérative. Il s'agit du composant qui va effectuer le calcul de Sobel, i.e. les gradients et la comparaison à un seuil pour décider si le pixel appartient ou non à un contour. Ce composant regroupe l'ensemble des unités arithmétiques et logiques dans une seule entité VHDL.

Complétez le fichier `gradientUnit.vhd` pour décrire les unités fonctionnelles du processeur dédié. Commentez votre code.

Donnez le type et la quantification que vous avez utilisé pour les signaux internes. Justifiez.

G_h et G_v sont de type SIGNED en 10 bits afin de pouvoir facilement faire des opérations arithmétiques et relationnels. Dans le pire des cas, le signaux G_h et G_v stockent la valeur 1020 (10 bits). Pour stocker la somme, on utilise une variable Sum de type SIGNED en 11 bits car dans le pire de cas on stocke la valeur 2040.

Validez le fonctionnement de ce deuxième composant par simulation en utilisant le testbench fourni `tb_gradientUnit.vhd`.

Combien de processus sont-ils utilisés et de quelles natures sont-ils ? liste de sensibilité ?

La simulation fonctionnelle a permis de valider le composant ? justifiez.

4 processus sont utilisés :

calculer G_v (I_Pix11,I_Pix12,I_Pix31,I_Pix32,I_Pix33)

calculer G_h (I_Pix,),

faire la somme (G_v,G_h),

affecter une valeur à 0_pixEdge (Sum).

Ils sont tous de nature combinatoire. La simulation fonctionnelle a permis de valider le composant car 0_pixEdge prend la valeur attendu pour chaque entrée testée.

Effectuez une synthèse logique et relever les performances obtenues en termes de ressources occupées (Flip-Flops, LUTs) sur le FPGA cible considéré.

Donnez les résultats obtenus.

Pour la première image il y a un gradient, le résultat 0_pixEdge vaut 1, pour la seconde image, il n'y a pas de gradient et 0_pixEdge vaut 0.

2.4 REGISTRE DE SORTIE

Le troisième composant de l'unité opérative consiste d'une simple bascule D (Flip-Flop, ou registre de taille 1 bit).

Complétez le fichier *pixedgeReg.vhd* pour décrire cette bascule D qui permet de mémoriser le résultat du calcul de Sobel sur chaque pixel avant son transfert dans la mémoire de sortie.

Effectuez une synthèse logique et vérifier que les ressources occupées correspondent bien à un seul Flip-Flop pour ce composant !

2.5 ASSEMBLAGE DES SOUS-COMPOSANTS DE L'UNITE OPERATIVE

Le fichier *operativeUnit.vhd* avec l'entité qui porte le même nom instancie les trois sous-composants qui constituent l'unité opérative. Ce fichier est fourni.

Analysez ce fichier pour comprendre sa structure : déclaration des sous-composants, instantiation, interconnexions.

Concernant les interconnexions, celles qui sont internes à l'unité opérative doivent être déclarées comme signaux internes. Les interconnexions externes sont à réaliser en connectant directement les ports des sous-composants aux ports de l'unité opérative.

Validez le fonctionnement de l'unité opérative par simulation en utilisant le testbench fourni *tb_operativeUnit.vhd*.

La simulation fonctionnelle a permis de valider le composant ? justifiez.

La simulation fonctionnelle a permis de valider le composant. En effet, on voit que le reset fonctionne (tous les variables sont remises sur 0), et le pixel d'entrée est bien répartie sur les pixels 11,21 et 31, puis lorsque la colonne est rempli, le programme shift la colonne pour remplir la suivante.

Effectuez une synthèse logique et relever les performances obtenues en termes de ressources occupées (Flip-Flops, LUTs) sur le FPGA cible considéré.

Donnez et commentez les résultats obtenus.

Les performances obtenues en terme de ressource sont les suivantes :
LUTs : 93, Flip-flops : 73.

3 GENERATEUR D'ADRESSES

3.1 ARCHITECTURE ET DESCRIPTION VHDL DU GENERATEUR D'ADRESSES

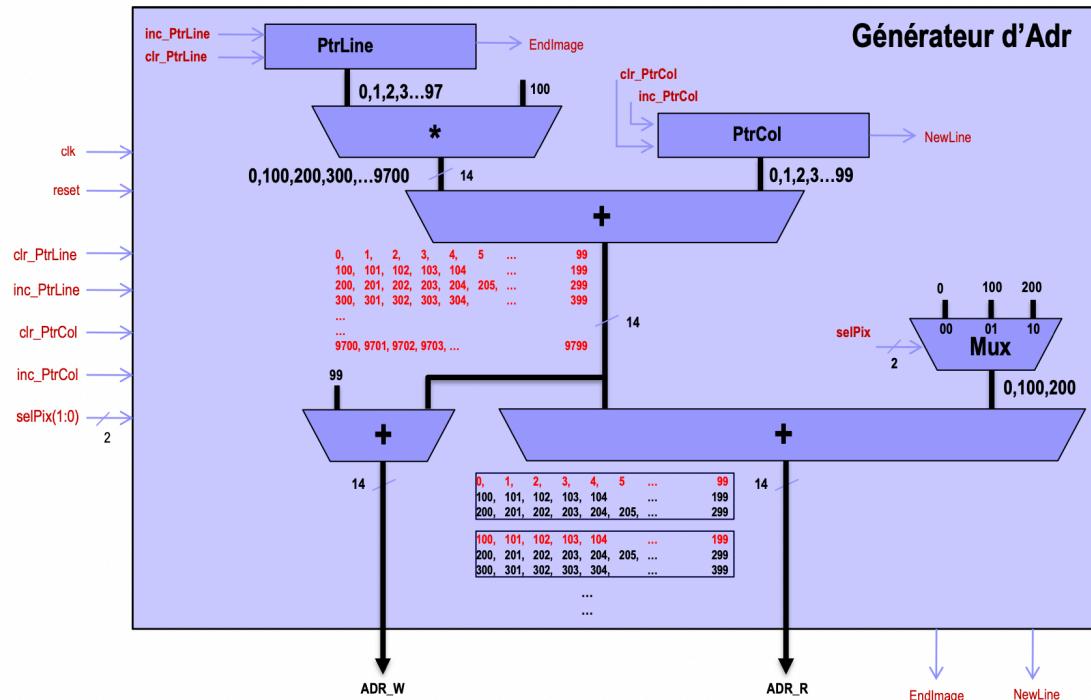
Le générateur d'adresses fait partie de l'unité de contrôle du processeur dédié à la détection de contours. Il permet de générer les bonnes adresses de lecture et d'écriture pour commander la mémoire d'entrée et de sortie respectivement.

Il doit a priori intégrer un ou plusieurs compteurs et des unités arithmétiques et logiques pour générer ces adresses. Plusieurs architectures sont possibles. Une approche de séquencement des opérations et un pseudo code ont été présentés lors du cours d'introduction.

Ainsi, il est demandé de travailler sur la proposition d'une architecture appropriée.

A noter que les signaux de contrôle des différentes unités fonctionnelles du générateur d'adresses doivent être générés par l'automate ou machine à états finis du processeur.

Donnez l'architecture que vous proposez pour le générateur d'adresses



Complétez le fichier `adrgenUnit.vhd` pour décrire cette architecture. Vérifiez si la liste prédefinie des ports d'entrée/sortie de l'entité VHDL correspond à votre architecture, sinon modifiez-la.

Pour cette description, vous allez modéliser les différents sous-composants directement dans ce fichier (i.e. entité) unique sous forme de plusieurs processus. L'approche de description structurelle utilisée dans l'unité opérative ne sera pas adoptée ici.

3.2 SIMULATION DU GENERATEUR D'ADRESSES

Utilisez le testbench fourni (`tb_adrgenUnit.vhd`) pour valider le fonctionnement du générateur d'adresses à travers des simulations comportementales.

La simulation fonctionnelle a permis de valider le composant ? justifiez.

La simulation fonctionnelle a permis de valider le composant. A chaque fois que PtrCol est sur '1', en sortie on a bien les valeurs d'adresse incrémentées de 1.

3.3 SYNTHESE LOGIQUE DU GENERATEUR D'ADRESSES

Effectuez une synthèse logique et relever les performances obtenues en termes de ressources occupées (Flip-Flops, LUTs, ...) sur le FPGA cible considéré.

Donnez les résultats obtenus.

Les performances obtenues en terme de ressource sont les suivantes :
LUTs : 62, Flip-flops : 91.

4 AUTOMATE OU MACHINE A ETATS FINIS

4.1 SPECIFICATION, DESCRIPTION VHDL ET SIMULATION DE L'AUTOMATE

L'automate ou machine à états finis représente le cœur de l'unité de contrôle du processeur dédié à la détection de contours. Elle permet de générer tous les signaux de contrôle : pour l'unité opérative, pour le générateur d'adresses, ainsi que pour les mémoires et les interfaces d'entrée/sortie. Elle reçoit aussi des signaux d'états de ces composants, comme par exemple les résultats des comparateurs.

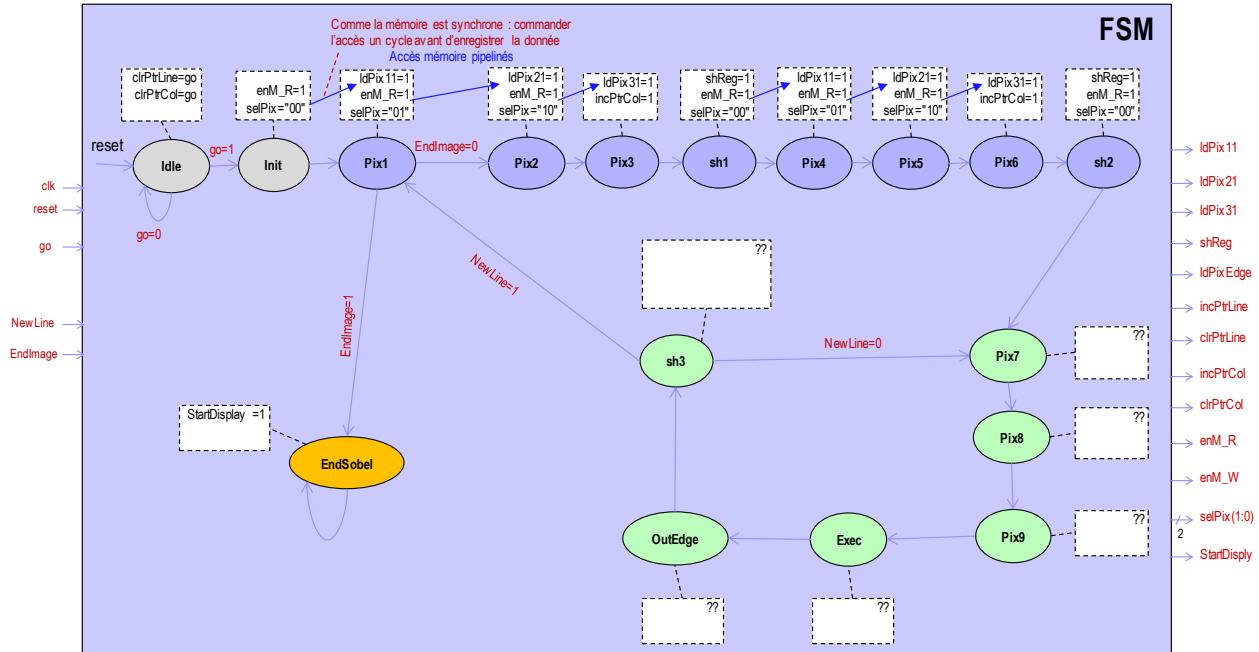
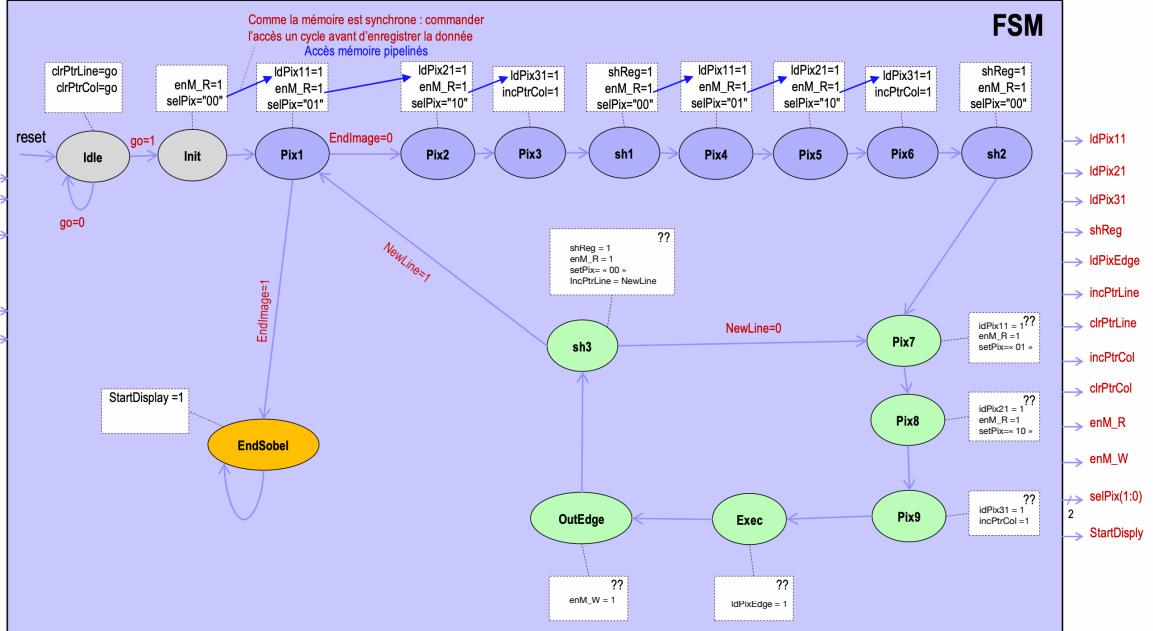


Figure 4 : Architecture (à compléter) de l'automate

Un squelette de cette automate a été présenté en cours d'introduction avec plusieurs signaux déjà complétés (**Figure 4** Erreur ! Source du renvoi introuvable.). Ainsi, il est demandé de travailler sur la finalisation (et éventuellement à l'optimisation) de cette automate.

Donnez la partie que vous avez complété de la spécification de l'automate, ou l'automate complète si vous avez éventuellement modifié/optimisé son architecture.



Complétez le fichier `automate.vhd` pour modéliser cette automate en VHDL.

Utilisez le testbench fourni (`tb_automate.vhd`) pour valider le fonctionnement de l'automate par simulation. Dans la simulation, notez la possibilité de vérifier l'évolution des états de l'automate en affichant le signal de type énuméré `current_state`.

Combien de processus sont-ils utilisés pour décrire l'automate et de quelles natures sont-ils ? liste de sensibilité ?

La simulation fonctionnelle a permis de valider le composant ? justifiez.

Deux processus : un synchrone (sensibilité : `clk`, `reset`) qui change les états à chaque front montant de l'horloge, et un asynchrone (sensibilité: `current_state`, `I_go`, `I_EndImage` et `I_NewLine`) qui détermine l'état suivant et les variables de sortie. La simulation fonctionnelle a permis de valider le composant. En effet, les états changent de la manière décrite par l'automate.

4.2 SYNTHESE LOGIQUE DE L'AUTOMATE

Effectuez une synthèse logique et relever les performances obtenues en termes de ressources occupées (Flip-Flops, LUTs, ...) sur le FPGA cible considéré.

Donnez les résultats obtenus.

Les performances obtenues en terme de ressource sont les suivantes :
LUTs : 9, Flip-flops : 17.

5 INTEGRATION DU PROCESSEUR ET PROTOTYPAGE

5.1 INTEGRATION DE L'ARCHITECTURE COMPLETE DU PROCESSEUR DE SOBEL

Pour assembler les unités constituant le processeur de Sobel, le module VHDL *sobelProc.vhd* a été créé et ajouté à votre projet. Ce module instancie l'unité opérative, le générateur d'adresses et l'automate à états finis.

Utilisez le testbench fourni (*tb_sobelProc.vhd*) pour valider le fonctionnement du processeur par simulation.

La simulation fonctionnelle a permis de valider le processeur ? justifiez.

La simulation fonctionnelle a permis de valider le processeur. En effet, on

Effectuez une synthèse logique et relever les performances obtenues en termes de ressources occupées (Flip-Flops, LUTs, ...) sur le FPGA cible considéré.

Donnez et commentez les résultats obtenus.

Les performances obtenues en terme de ressource sont les suivantes : LUTs : 162, Flip-flop : 181. Il s'agit de la somme des ressources matérielles de chaque composants du sobelProc à savoir : l'unité opérative, le générateur d'adresse et l'automate.

5.2 PROTOTYPAGE ET DEMONSTRATION SUR CARTE

Pour finaliser la validation du processeur conçu, il faut l'intégrer dans un système complet avec une mémoire d'entrée (image de test), une mémoire de sortie et un système d'affichage.

Pour l'image de test, la célèbre photo de Lena est mise à disposition avec une résolution 100x100 et un codage sur 8 bits en niveaux de gris.

Pour le système d'affichage, nous mettons à votre disposition un bloc d'interface codé en VHDL pour utiliser le contrôleur VGA intégré à la carte.

Le module VHDL *sobelSys.vhd* a été créé et ajouté à votre projet. Ce module instancie le processeur de Sobel, les mémoires en entrée et en sortie, ainsi que le contrôleur VGA. Vérifiez sa structure et son contenu.

Effectuez une synthèse logique et relever les performances obtenues en termes de ressources occupées (Flip-Flops, LUTs, mémoires, ...) sur le FPGA cible considéré.

Donnez et commentez les résultats obtenus.

Les performances obtenues en terme de ressource sont les suivantes :
LUTs : 423, Flip-flops : 263. En effet, il faut rajouter les ressources matérielles dédiées à l'affichage de l'image et à l'utilisation de la ram et de la rom.

Pour la compilation matérielle finale et le prototypage sur carte, un fichier de contraints (*Nexys4_Sobel.xdc*) a été ajouté à votre projet pour router les ports d'entrée/sortie aux broches correspondants du circuit FPGA. De plus, une image a été ajoutée pour être chargée dans la mémoire d'entrée. Cette image a été déjà convertie en niveaux de gris codés sur 8 bits et placée dans un fichier texte dans le bon format (*dancing_spider.txt*).

Déroulez le flot de conception jusqu'à la production du bitstream. Il s'agit du fichier de configuration du FPGA qui spécifie comment le FPGA sera utilisé pour fournir le circuit conçu en VHDL et contraint par le fichier qui spécifie les entrées/sorties.

Une fois la génération du bitstream terminée, vous pouvez connecter une carte Nexys 4 et un écran VGA avant de transférer le bitstream sur le FPGA de la carte via le port USB dédié à la programmation. **Dans les conditions actuelles de TP, il faudra déposer le fichier de bitstream**

généré (`sobelSys.bit`) sur Moodle. L'enseignant fera le test sur carte. Le bitstream généré se trouve dans le sous-repertoire `/sobel/sobel.runs/impl_1`

Le prototypage et la démonstration sur carte sont-ils concluants ?

6 ANALYSE ET COMPARAISON DES PERFORMANCES

Cette dernière phase du projet est dédiée à l'analyse et comparaison des performances des solutions logicielles/matérielles mises en application dans cette SAR.

Certains résultats ont déjà été relevé à partir des rapports de synthèse générés par l'outil de conception, d'autres restent à calculer ou à extrapoler analytiquement.

Pour identifier la fréquence d'horloge maximale atteignable sur ce FPGA du système conçu, on peut analyser le rapport généré « Report timing summary » après routage comme indiqué dans cette copie d'écran :

Tcl Console	Messages	Log	Reports	Design Runs
Report	Report Type			
impl_1_phys_opt_report_timing_summary_0	Report timing summary (report_timing_summary)			
Route Design (route_design)				
impl_1_route_report_drc_0	Report on error or violations against a set of design rule checks (report_drc)			
impl_1_route_report_methodology_0	Report on error or violations against a set of methodology checks (report_methodology)			
impl_1_route_report_power_0	Report power analysis details (report_power)			
impl_1_route_report_route_status_0	Report on status of the routing (report_route_status)			
impl_1_route_report_timing_summary_0	Report timing summary (report_timing_summary)			
impl_1_route_report_incremental_reuse_0	Report on achievable incremental reuse for the given design-checkpoint (report_incremental_reuse)			
impl_1_route_report_clock_utilization_0	Report information about clock nets in design (report_clock_utilization)			

Dans ce rapport, cherchez la section « Max Delay Paths » et relevez la valeur du « Data Path Delay ». Cette valeur correspond à la période minimale de l'horloge, i.e. au délai du chemin critique du circuit. Le chemin critique est également indiqué (sa source et sa destination sont identifiées). La fréquence maximale d'horloge est l'inverse de cette période minimale !

impl_1_route_report_timing_summary_0 - impl_1

/homes/abaghdad/TAF_OPE/UE_AMRC/sobel/sobel.runs/impl_1/sobelSys_timing_summary_routed.rpt

Q Next Previous Highlight Match Case Whole Words 4 Match(es)

wns

217

218

219

220 Max Delay Paths

221

222 Slack (MET) : xxxxx ns (required time - arrival time)

223 Source: sobelProc_instl/operativeUnit_1/regUnit_1/S_Pix21_reg[1]/C

224 (rising edge-triggered cell FDCE clocked by clk_outl_clk_wiz_vga_25MHz {rise@0.000ns fall@20.000ns})

225 Destination: sobelProc_instl/operativeUnit_1/pixedgeReg_1/S_pixEdge_reg/D

226 (rising edge-triggered cell FDCE clocked by clk_outl_clk_wiz_vga_25MHz {rise@0.000ns fall@20.000ns})

227 Path Group: clk_outl_clk_wiz_vga_25MHz

228 Path Type: Setup (Max at Slow Process Corner)

229 Requirement: 40.000ns (clk_outl_clk_wiz_vga_25MHz rise@40.000ns - clk_outl_clk_wiz_vga_25MHz rise@0.000ns)

230 Data Path Delay: xxxxx ns (logic 4.688ns (42.372%) route 6.376ns (57.628%))

231 Logic Levels: 14 (CARRY4=6 LUT3=3 LUT4=2 LUT5=2 LUT6=1)

Notez que nous utilisons sur la carte une horloge de fréquence de 25 MHz (i.e. une période de 40 ns), bien inférieure à la fréquence maximale. Le système doit donc s'exécuter correctement avec cette fréquence de fonctionnement.

Identifiez la fréquence d'horloge maximale atteignable sur ce FPGA.

La fréquence maximale d'horloge atteignable sur ce FPGA est de 87MHz

Combien de cycles d'horloge sont nécessaires pour le traitement d'un pixel ?

D'après l'automate, on prend en compte le nombre d'état lors de du cycle de l'automate, on obtient 6 états par cycle. Chaque état change de manière synchrone d'où la nécessité de 6 cycle d'horloge pour le traitement d'un pixel

Combien de cycles d'horloge sont nécessaires pour le traitement d'une image de définition 396x396 ?

En utilisant l'automate qui change d'états à chaque cycle d'horloge on obtient un nombre de cycle valant :
 $(13 + 6 \cdot 394) \cdot 394 = 936\,538$

Combien d'images de définition 396x396 par second ce processeur peut traiter ?

$$87\text{e}6 / 936538 = 92,9 \text{ image par seconde}$$

Comparez et discutez ces performances par rapport à celles de la solution logicielle développée en Python dans la première séance de travaux pratiques.

En python, il fallait environ 1,5 milliards de cycles d'horloge pour traiter une image de définition 396x396. Ici, moins d'un million. On peut imaginer que l'ordinateur qui fait tourner python possède un processeur de cadence typique 3 GHz. Ce qui ferait 2 images par secondes contre 93 images pour la version réalisée en VHDL. Il y a donc un important gain de performance entre la version python et la version VHDL que ce soit en terme de temps d'exécution, de consommation, d'efficacité énergétique ou de matérielle utilisé.

7 BONUS – POUR ALLER PLUS LOIN

Pour ceux qui réussissent à terminer le projet en avance et souhaitent aller plus loin, plusieurs idées peuvent être investiguer.

Concernant l'implémentation logicielle : pour compléter les analyses des performances en terme de temps d'exécution et apprécier/démontrer les vitesses atteignables, on peut étendre l'application du traitement développé sur une vidéo composée de séquence d'images. La performance en nombre d'images par second peut ainsi être illustrée expérimentalement et confrontée à l'étude analytique.

Concernant l'implémentation matérielle : plusieurs idées peuvent être investiguées pour exploiter d'autres niveaux de parallélisme et améliorer la performance en réduisant le temps d'exécution. En plus de la possibilité de dupliquer le processeur pour concevoir une architecture multi-coeur, il est possible d'envisager d'autres alternatives pour l'architecture de l'unité opérative. La **Figure 5** donne deux exemples de variantes architecturales du banc de registres de l'unité opérative. De telles architectures impliquent cependant des changements correspondants dans le reste des composants (unités fonctionnelles, automate, et probablement interfaces mémoires). Vous pouvez éventuellement étudier, au moins analytiquement, les performances atteignables en terme de temps d'exécution avec l'une de ces différentes architectures.

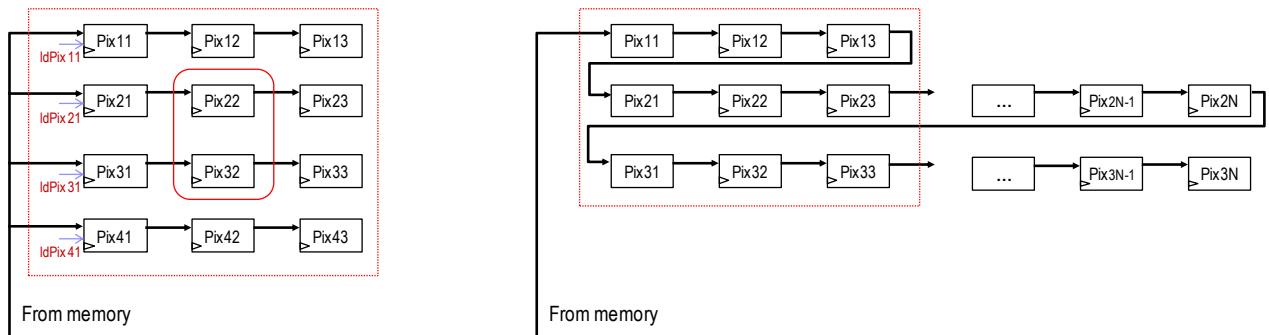


Figure 5 : Exemples de variantes architecturales du banc de registres de l'unité opératives

Et si d'autres idées de développement ou d'optimisation vous viennent à l'esprit, n'hésitez pas à proposer et en discuter avec l'équipe enseignante !

OUR WORLDWIDE PARTNERS UNIVERSITIES - DOUBLE DEGREE AGREEMENTS



3 CAMPUS, 1 SITE



IMT Atlantique Bretagne-Pays de la Loire – <http://www.imt-atlantique.fr/>

Campus de Brest

Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
T +33 (0)2 29 00 11 11
F +33 (0)2 29 00 10 00

Campus de Nantes

4, rue Alfred Kastler
CS 20722
44307 Nantes Cedex 3
France
T +33 (0)2 51 85 81 00
F +33 (0)2 99 12 70 08

Campus de Rennes

2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
France
T +33 (0)2 99 12 70 00
F +33 (0)2 51 85 81 99

Site de Toulouse

10, avenue Édouard Belin
BP 44004
31028 Toulouse Cedex 04
France
T +33 (0)5 61 33 83 65



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom