

2023

Détection de point d'intérêts



**Sefofo SOKPOR &
Éric GASNIER**

AJC

11/07/2023

Table des matières

I.	Introduction	2
II.	Étude de la détection de point d'intérêts	3
	1) Analyse	3
	• Architecture du projet précédent	4
	• Module corner detection.....	5
	• Convolution module.....	6
	• Plan de mise en œuvre.....	7
	2) Test.....	8
	3) Démonstration.....	10
III.	Conclusion.....	15
IV.	Exemples d'application	15

Date	Affectation	Version	Auteurs
17/07/2023	Création	V0	SAS & EG

I. Introduction

Dans le contexte de la suite d'un mini-projet dans le cadre de la formation AJC-FPGA pour l'entreprise safran, il nous a été demandé réalisation une détection de points d'intérêt. L'objectif de ce document est d'étudier dans un premier temps la faisabilité de ce projet (consigné dans un document). Ensuite, les phases d'analyse, de tests et de démonstrations sont déroulées dans celui-ci. La détection de points d'intérêts (ou corner detection) est, au même titre que la détection de contours, une étape préliminaire à de nombreux processus de traitement d'images. Les points d'intérêts, dans une image, correspondent à de doubles discontinuités de la fonction d'intensités. Ce sont par exemple : les coins, les jonctions en T ou les points de fortes variations de texture.



*Différents types de points d'intérêts :
coins, jonction en T et point de fortes variations de texture.*

Fig. 1 : Point d'intérêt

Dans l'étude de la faisabilité, nous avons examiné les aspects techniques et les ressources nécessaires pour réaliser ce projet consigné dans le document V0.

La conception logique de notre projet se réalise en suivant les étapes énumérées ci-dessous. Pour nous tenir, nous proposons une série d'analyse, de tests et de démonstrations.

II. Étude de la détection de point d'intérêts

Plusieurs méthodes existent pour réaliser la détection de points d'intérêt. Les méthodes retenues pour réaliser notre projet sont celles illustrées ci-dessous :

Solution 1 :

Faire une somme simple : FSV + FSH et d'appliquer le seuil.

Si $FSH + FSV > Th$, alors un point d'intérêt est détecté.

Solution 2 :

Comparer FSH et FSV à un seuil. L'opération devient alors :

Si $FSH > Th$ et $FSV > Th$, alors un point d'intérêt est détecté.

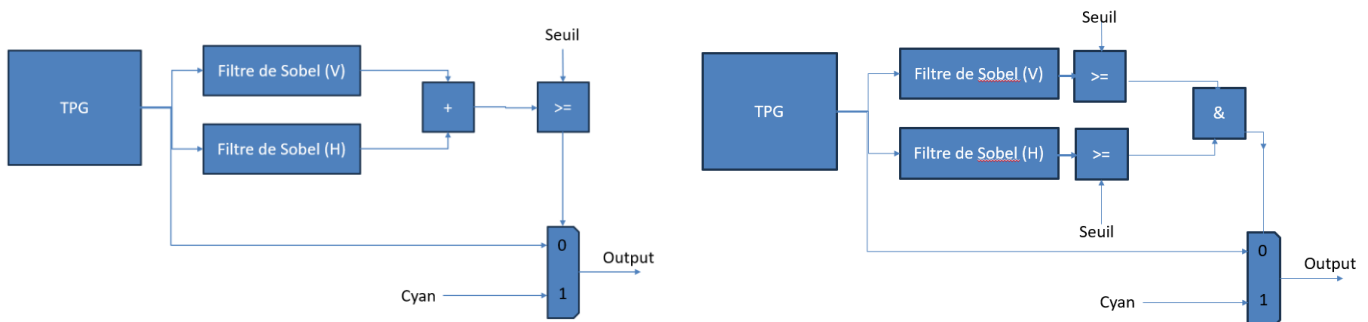


Fig. 2 : Solutions retenues pour la détection de points d'intérêt

Le principe est le suivant :

L'image est générée par le module TPG. Elle est ensuite traitée en parallèle par deux filtres de Sobel, l'un horizontal et l'autre vertical. Ces deux filtres permettent d'effectuer des gradients sur l'image, l'un selon l'axe des abscisses et l'autre selon l'axe des ordonnées.

Ensuite, une comparaison par rapport à un seuil permet d'identifier les points d'intérêt.

Enfin, par le biais d'un multiplexeur, si le seuil est dépassé, nous colorions en cyan ces pixels sinon nous conservons le pixel originel.

1) Analyse

Dans cette partie, l'objectif est de charger une image en entrée de notre module de convolution.

L'image est générée par le module TPG et importer dans notre simulation en extension « .txt ».

Elle est ensuite récupérée en sortie en fichier « .txt » et afficher avec le logiciel ImageJ.

Pour ce faire, nous partons des modules utilisés dans le projet précédent, réalisation d'une IP de traitement d'image sur cible Zynq7020 avec affichage VGA.

- Architecture du projet précédent

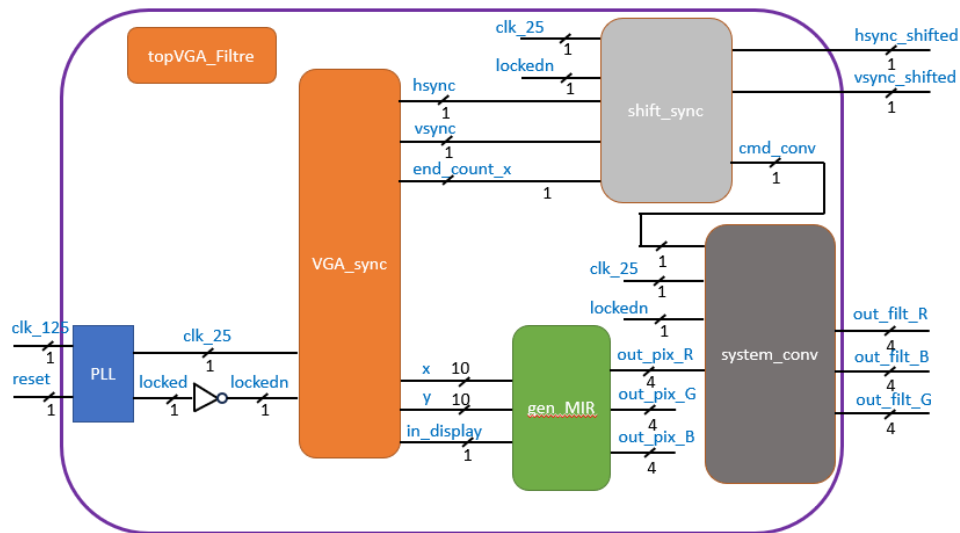


Fig. 3 : synoptique général

Dans ce synoptique, nous avons créé les modules suivants :

- ✓ PLL : génère une horloge clk_25 à 25,175 MHz, fréquence de balayage de l'écran pixel par pixel.
- ✓ VGA_Sync : pour les signaux de synchronisations horizontal et vertical.
- ✓ Gen_mir pour générer l'image à l'écran.
- ✓ System_conv permet d'effectuer la convolution et le padding de l'image.
- ✓ Shift_sync : pour le décalage des signaux de synchronisations.

Notre module system_conv a été conçu pour effectuer plusieurs opérations de filtrage. Il effectue 5 filtrages en parallèle. Pour le besoin de notre projet, nous utiliserons la matrice identité, les filtres de sobel horizontal et vertical, nécessaires à la détection de points d'intérêts.

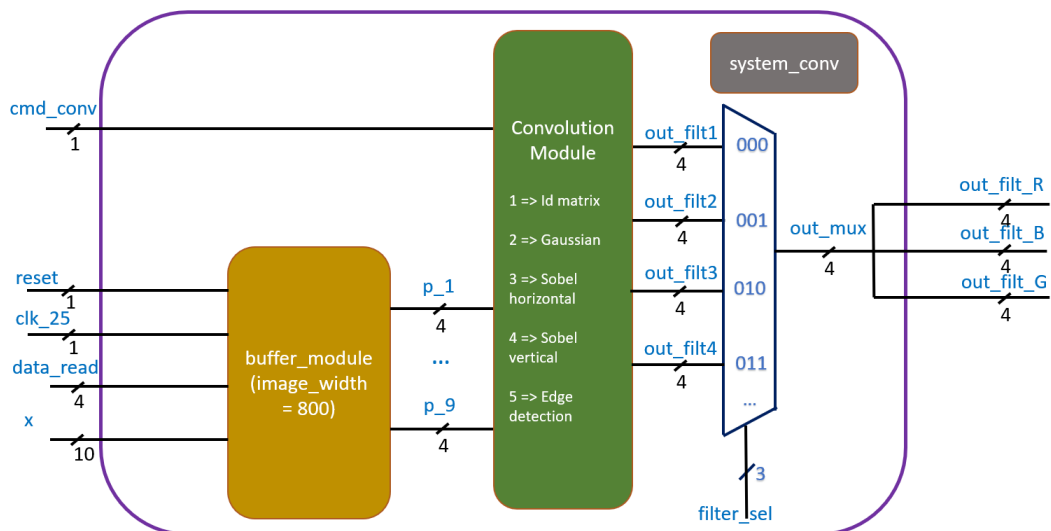
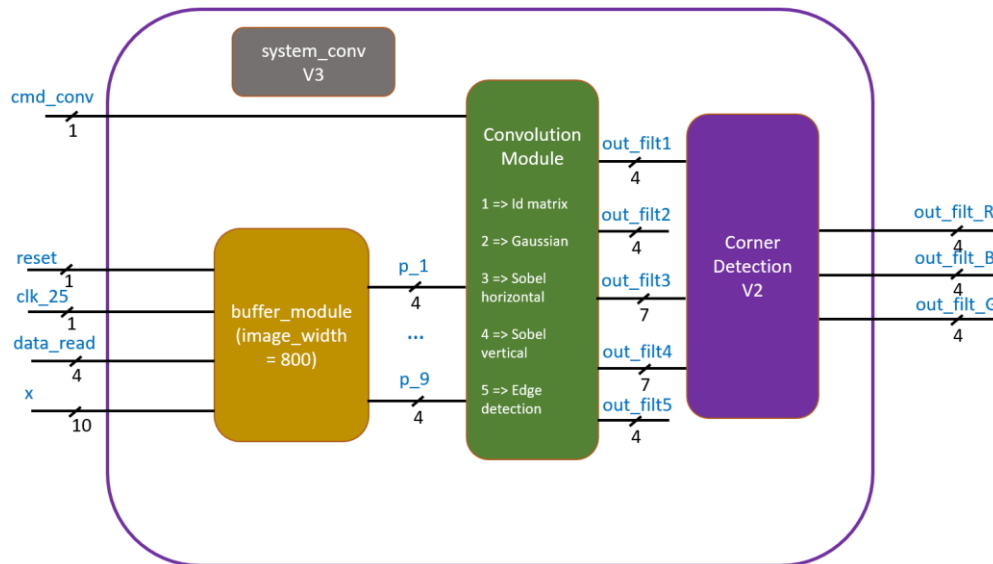


Fig. 4 : Module de convolution

Pour déterminer les points d'intérêts, nous introduisons un nouveau sous-module dans system_conv en remplacement du sélecteur de filtre.

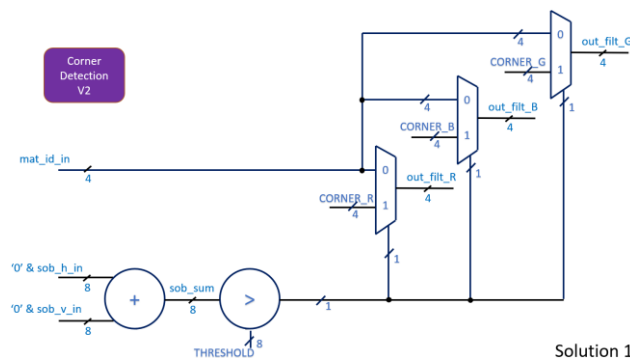
- Module corner detection

Afin de ne pas casser toute notre architecture et surtout au vu du temps imparti, nous avons décidé de ne modifier que system_conv. Pour cela, nous remplaçons le multiplexeur permettant la sélection de filtres par un module réalisant la détection de points d'intérêt. Notre étude propose deux solutions pour la détection des points d'intérêts avec trois objectifs à atteindre.



Solution 1 :

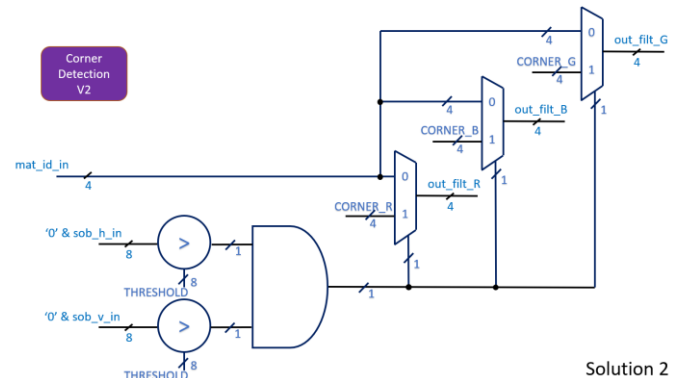
Faire une somme simple : FSV + FSH et d'appliquer le seuil.
Si $FSH + FSV > Th$, alors un point d'intérêt est détecté.



Solution 1

Solution 2 :

Comparer FSH et FSV à un seuil. L'opération devient alors :
Si $FSH > Th$ et $FSV > Th$, alors un point d'intérêt est détecté.



Solution 2

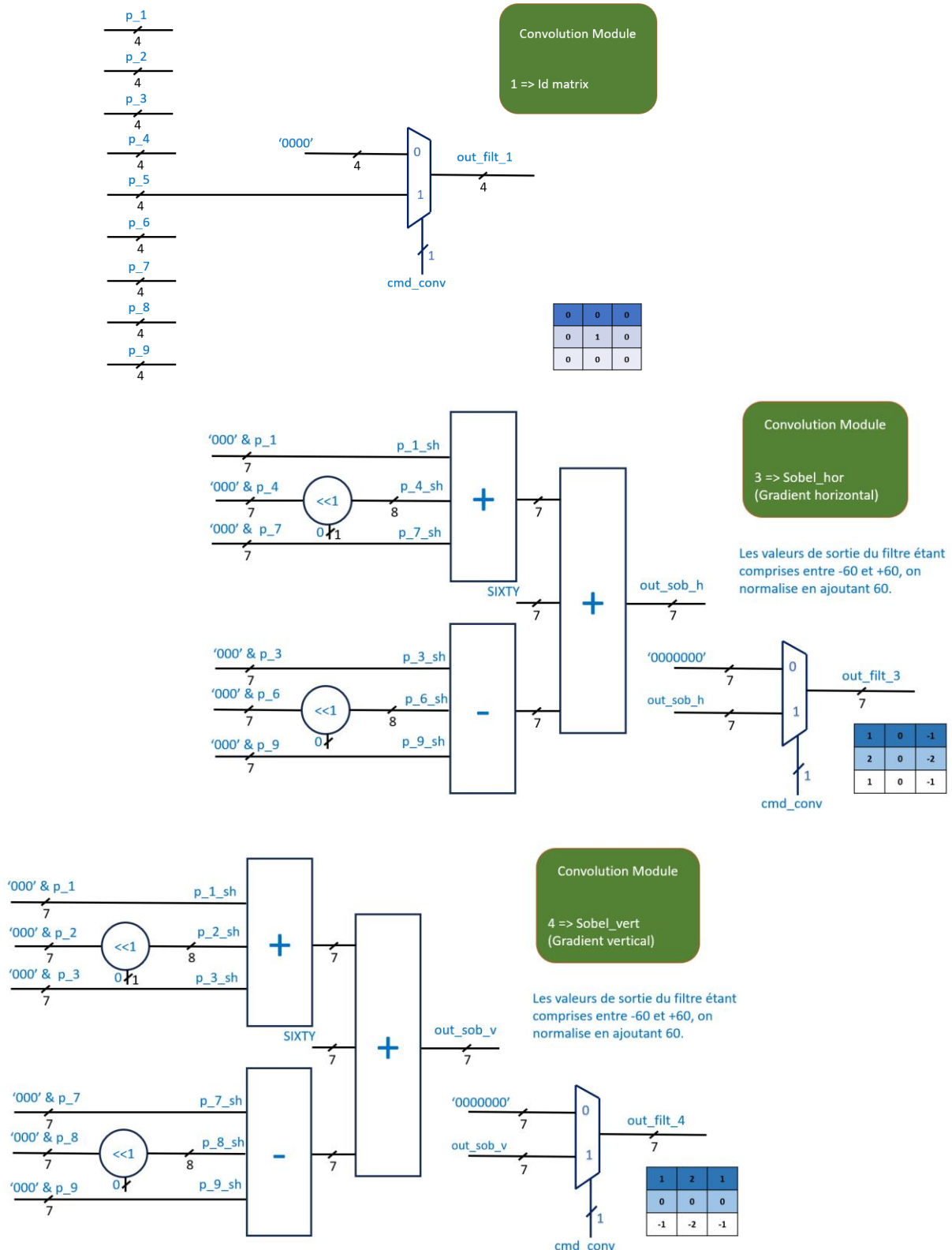
Ce module a à son entrée, la matrice d'identité, le filtre de Sobel vertical (FSV) et le filtre de Sobel horizontal (FSH). En sortie, nous avons le résultat du traitement.

La seconde solution présente l'avantage de prendre en compte les deux gradients de manière indépendante. Ainsi, selon la seconde solution, pour qu'un point soit considéré être un point d'intérêt, il doit varier en intensité fortement en abscisse ET en ordonnée.

- Convolution module

Pour notre nouveau projet, nous avons besoin d'utiliser les 3 sorties ci-dessous du module de convolution, à savoir :

- L'image en sortie de la matrice identité pour récupérer l'image d'origine,
- L'image en sortie du filtre de Sobel horizontal (FSH),
- L'image en sortie du filtre de Sobel vertical (FSV).



- **Plan de mise en œuvre**

Nous avons découpé le projet en 3 phases distinctes afin de simplifier la mise en œuvre de notre solution. A l'issue de ces 3 phases, nous pourrons réaliser la détection de points d'intérêt pour des images au format 640x480 en niveaux de gris, codés sur 4 bits.

L'architecture finale sera alors la suivante :

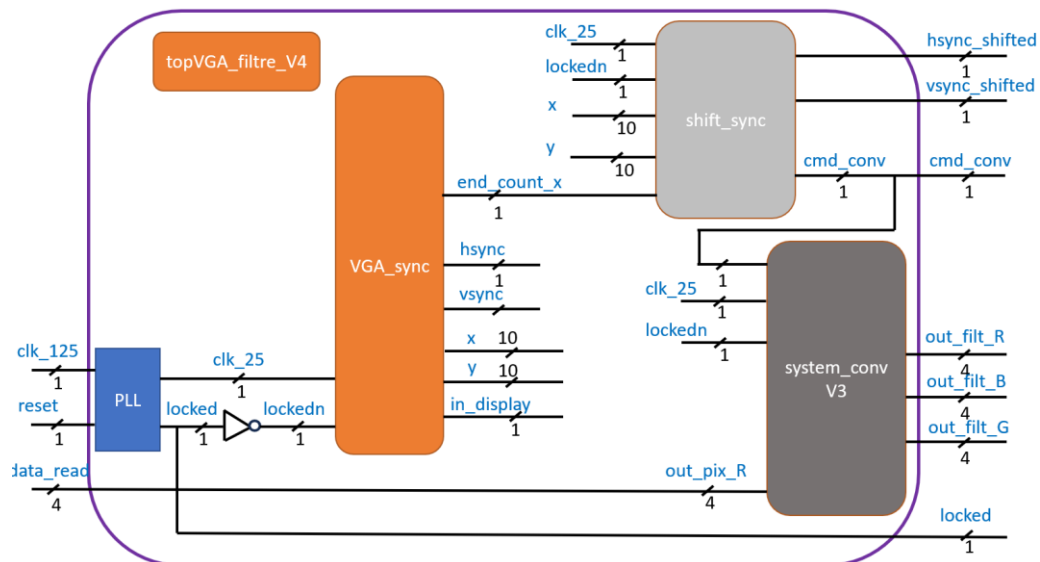


Fig. 5 : Architecture générale avec génération d'image à partir d'une image 640x480 au format .txt

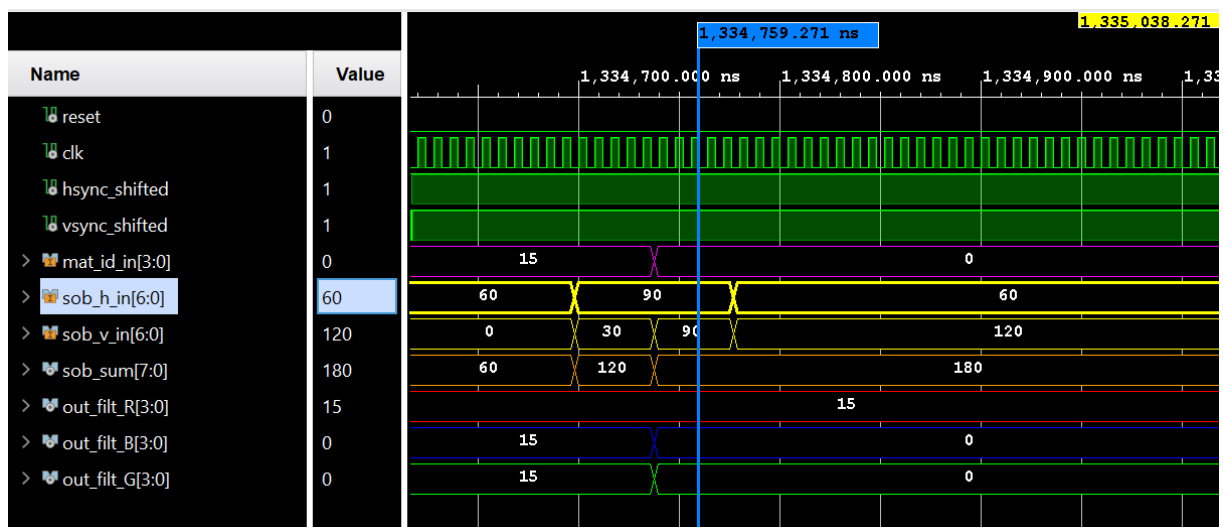
- **Premier objectif :** Afficher l'image du damier avec détection de points d'intérêt sur un écran VGA.
 - ✓ Pour cela, le module gen_mir est conservé.
 - ✚ Lors de cette première étape, seul le code du module system_conv et de ses sous-modules est à modifier.
- **Deuxième objectif :** Générer une image de résolution 640x480 avec détection de points d'intérêt au format .txt à partir d'une image de résolution 640x480 au format .txt. Le module gen_mir n'est plus dans l'architecture.
 - ✓ Pour cela, l'image sera chargée à partir du testbench. L'image 640x480 sera mise au format VGA 800x525 dans un fichier .txt (opération faite avec imageJ).
 - ✓ Ensuite, dans le testbench, l'image de sortie (640x480) sera stockée au format VGA 800x525 dans un fichier .txt.
 - ✓ Entrée : le testbench traite une image au format VGA (800x525), avec la zone virtuelle.
 - ✓ Sortie : le testbench génère une image au format VGA (800x525), avec la zone virtuelle.
 - ✚ Lors de cette seconde étape, le fichier top incluant toute l'architecture est à modifier, ainsi que le testbench.

- Troisième objectif : Générer une image de résolution 640x480 avec détection de points d'intérêt au format .txt à partir d'une image de résolution 640x480 au format .txt.
 - ✓ On réalise la même conversion sur l'image comme à l'étape 2, mais on récupère à la sortie une image au format 640x480.
 - ✓ Entrée : le testbench traite une image au format VGA (640x480), sans la zone virtuelle.
 - ✓ Sortie : le testbench génère une image au format VGA (640x480), sans la zone virtuelle.
- ✚ Lors de cette dernière étape, seul le testbench est à modifier.

2) Test

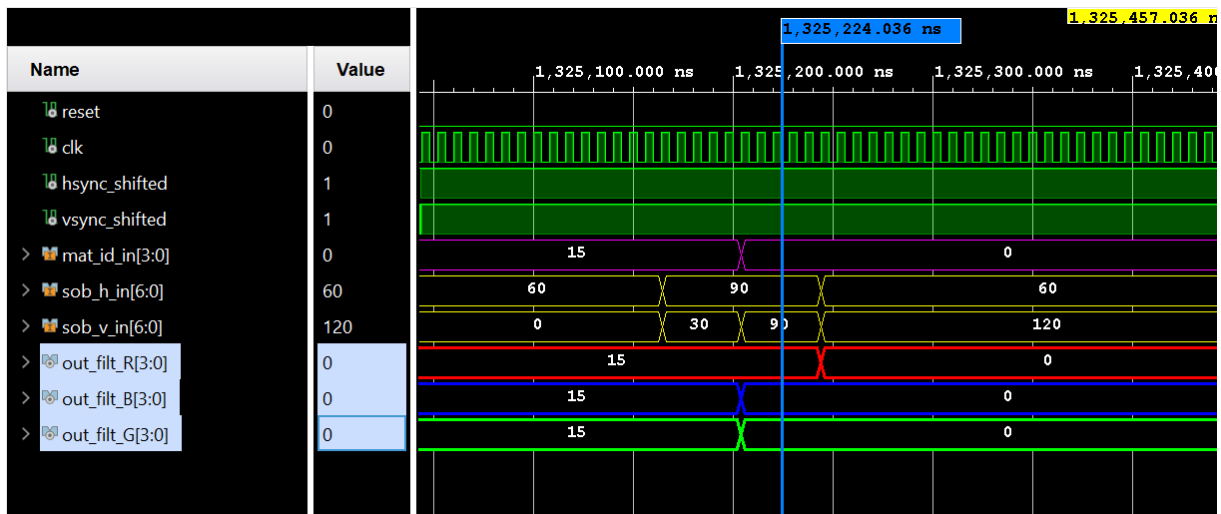
Les tests effectués sont pour s'assurer du bon fonctionnement de chaque module et puis de l'ensemble. Les résultats et interprétations sont compilés dans la suite du document. Nous intégrons l'ensemble.

- Premier objectif : Affichage du damier sur écran VGA.
 - ✓ Solution 1 : Addition des gradients avant seuillage.
 - ✚ La zone à l'emplacement du marqueur correspond à un point d'intérêt. (Sorties B et G à l'état bas & Sortie R à l'état haut).
 - ✚ La somme des données en sortie des filtres de Sobel dépassent effectivement le seuil, fixé à 140 dans cette simulation ($90 + 90 > 140$).
 - ✚ Nous pouvons constater également que le point d'intérêt correspond à la transition de noir à blanc en abscisse (mat_id_in). C'est aussi probablement le cas en ordonnée, mais plus difficile à observer.



✓ Solution 2 : Seuillage individuel des gradients.

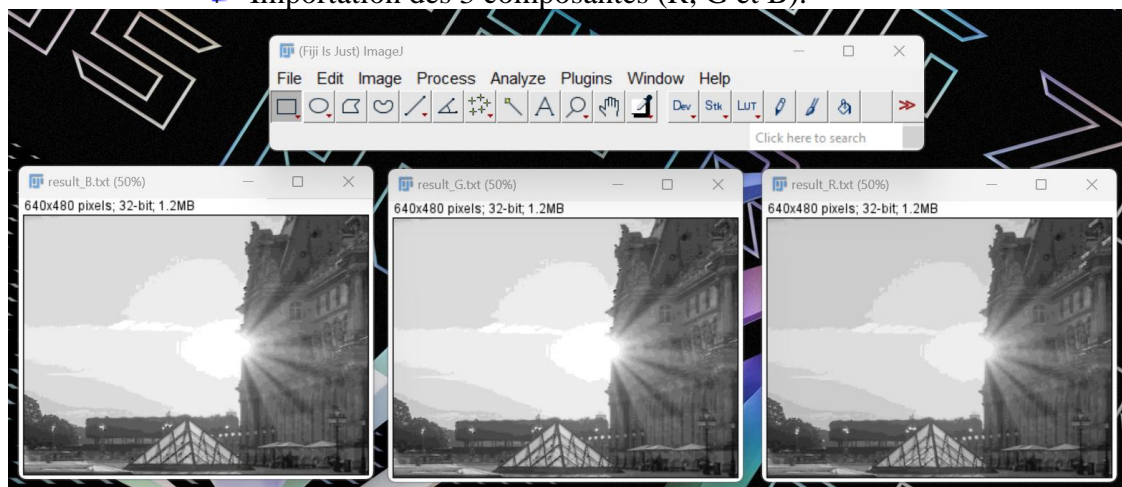
✚ Les données en sortie des filtres de Sobel horizontal et vertical dépassent effectivement le seuil, fixé à 60 dans cette simulation ($90 > 60$).



➤ Deuxième et troisième objectif : Sauvegarde de l'image de sortie avec détection de points d'intérêt dans un fichier au format.txt.

✓ Une fois le premier objectif validé, nous pouvons passer à la sauvegarde dans des fichiers. A ce stade, le test consiste à vérifier le bon format des fichiers de sortie avec le logiciel ImageJ.

✚ Importation des 3 composantes (R, G et B).



✚ Merge des 3 composantes en une seule image RGB.

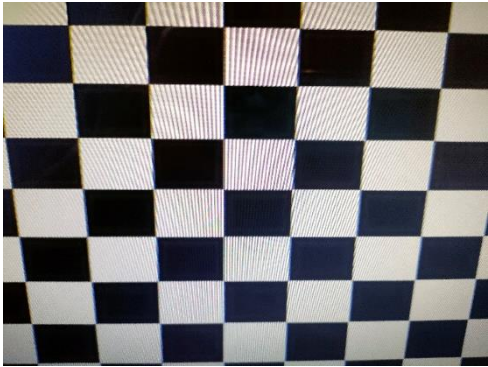
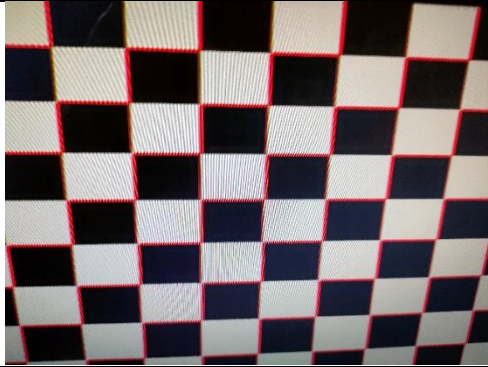
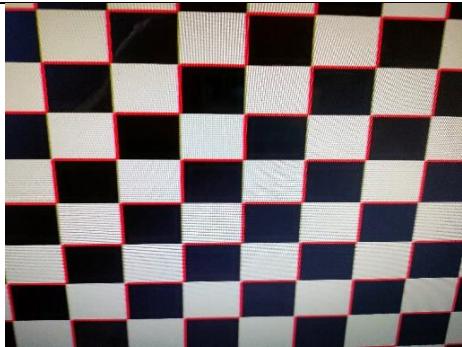


3) Démonstration


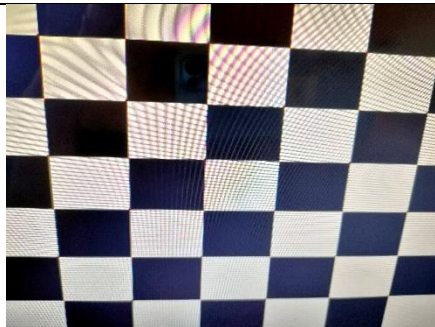
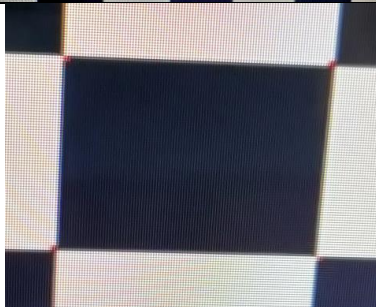
Dans cette partie, nous documentons les résultats : compilation des résultats des tests, y compris les captures d'écran l'affichage de l'image. Pour voir le seuil, il faut l'augmenter ou le réduire.

➤ **Premier objectif : Affichage du damier sur écran VGA.**

✓ Solution 1 : Addition des gradients avant seuillage.


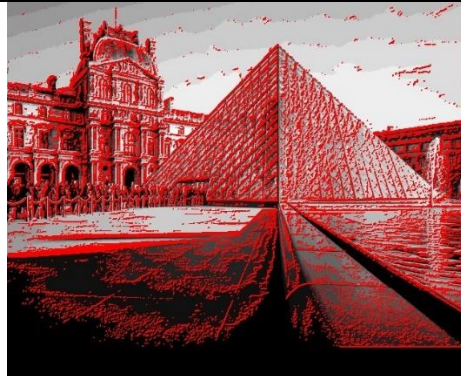
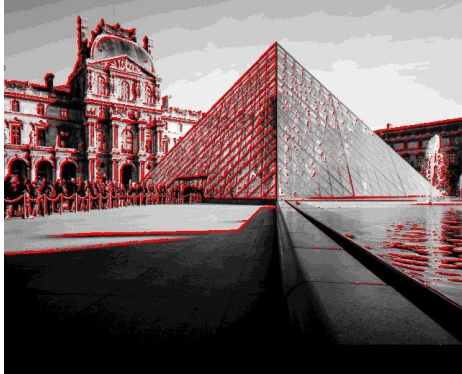

Image d'origine	Seuil 120
	
	Seuil 130 

✓ Solution 2 : Seuillage individuel des gradients.





Seuil 55	Seuil 60
	
En zoomant, nous observons un point rouge à chaque coin du carré noir.	

- Deuxième objectif : Sauvegarde au format 800x525 dans un fichier .txt.
- ✓ Solution 1 : Addition des gradients avant seuillage.

LE LOUVRE

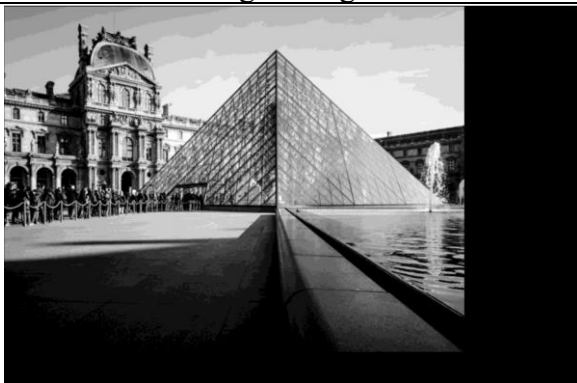
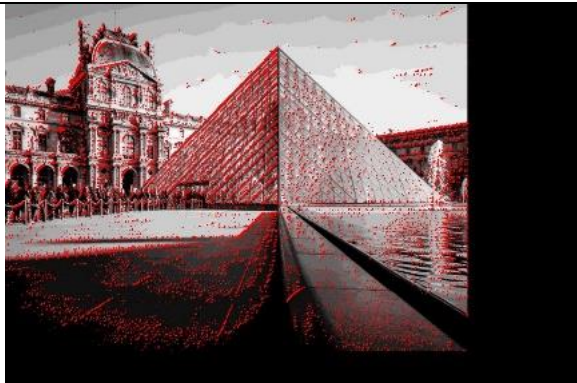
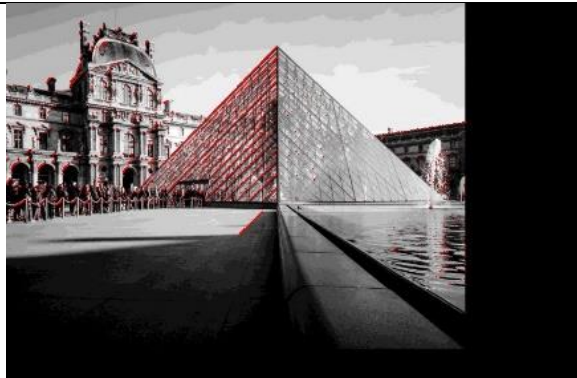
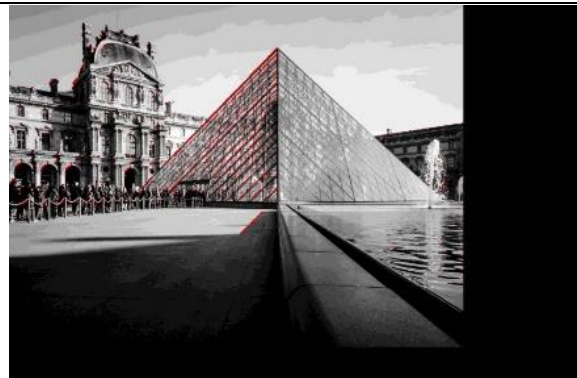
Image d'origine	Seuil 120
	
Seuil 130	Seuil 140
	

BÂTIMENT





Image d'origine	Seuil 120
	
Seuil 130	Seuil 140
	

✓ Solution 2 : Seuillage individuel des gradients.

➤ LE LOUVRE

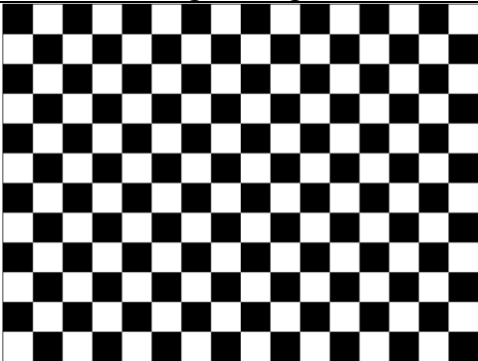
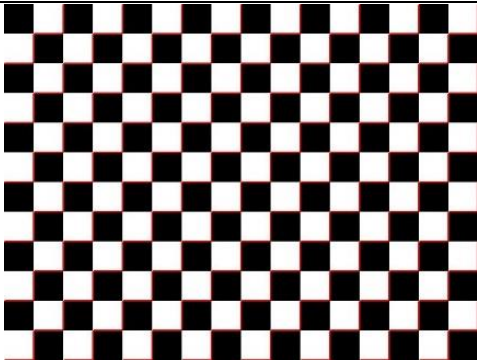
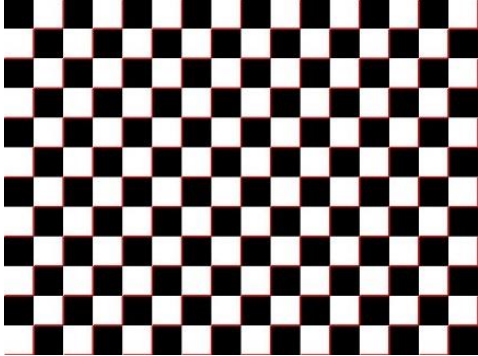
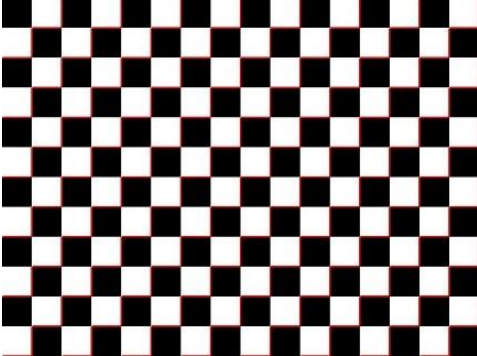
Image d'origine	Seuil 60
	
Seuil 65	Seuil 70
	

BÂTIMENT




Image d'origine	Seuil 60
	
Seuil 65	Seuil 70
	

- Troisième objectif : Sauvegarde au format 640x480 dans un fichier .txt.
 - ✓ Solution 1 : Addition des gradients avant seuillage.

DAMIER

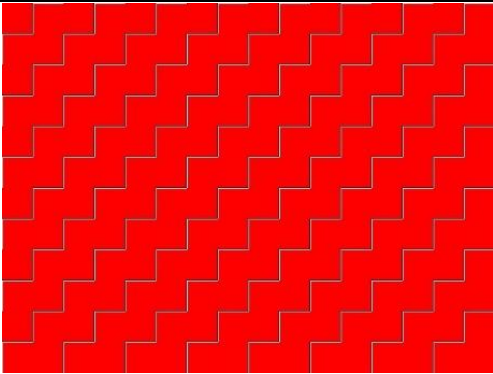
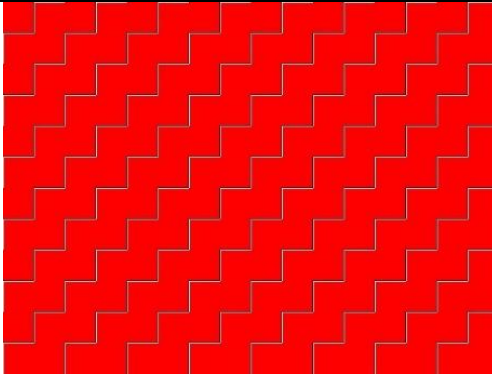
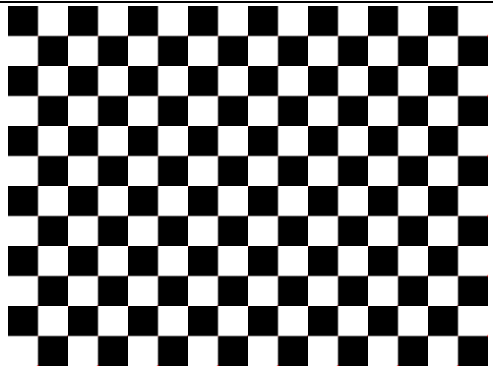
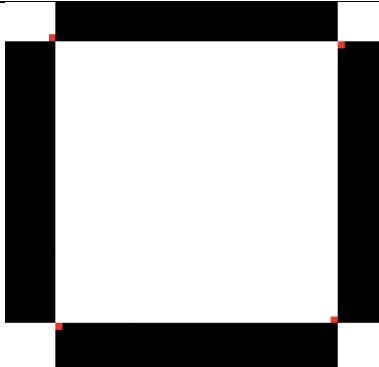
Image d'origine	Seuil 120
	
Seuil 130	Seuil 140
	

FORET





Image d'origine	Seuil 130
	
Seuil 145	
	

- ✓ Solution 2 : Seuillage individuel des gradients.

DAMIER

Seuil 55	Seuil 59
	
Seuil 60	Seuil 60 en zoomant
	

FORET

Seuil 55	Seuil 60
	
Seuil 65	Seuil 70
	

III. Conclusion

A travers ce projet, nous avons testé deux méthodes pour réaliser la détection de points d'intérêt :

- Addition des gradients avant seuillage,
- Seuillage individuel des gradients.

Les résultats et démonstrations attestent du bon fonctionnement du module mis en place.

Nous avons observé que la seconde méthode apportait de meilleurs résultats. Cela se comprend assez facilement car pour la seconde méthode nous devons observer de fortes variations d'intensité dans les deux directions, tandis que pour la première méthode une très forte variation d'intensité dans une direction suffit à détecter un point d'intérêt.

Afin de tester les algorithmes, nous avons chargé une image depuis le testbench. A ce stade, il n'est pas possible de lancer l'implémentation et générer le bitstream. Pour aller plus loin, il serait intéressant d'utiliser la partie SOC de la carte afin de charger l'image dans une RAM (disponible sur la carte CoraZ7). Nous pourrions ainsi afficher l'image avec détection de points d'intérêt sur un écran VGA.

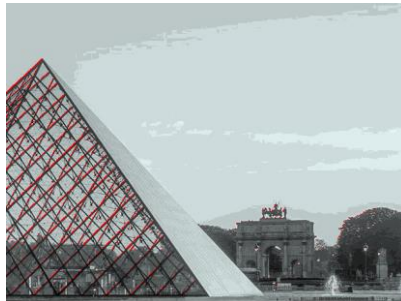
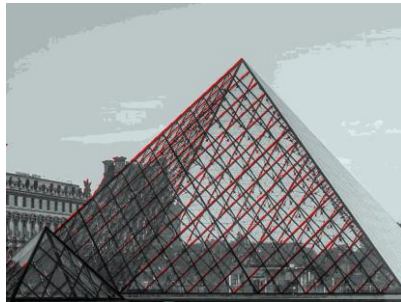
IV. Exemples d'application

Dans les pages suivantes, nous avons illustré des cas d'utilisation de la détection de points d'intérêt :

- ✓ Vue panoramique
 - Nous réalisons une série de photos.
 - Pour les raccorder les unes aux autres, nous repérons les points d'intérêt.
- ✓ Suivi d'objet
 - Sur une vidéo, nous effectuons une détection de points d'intérêt, image par image.
 - Nous pouvons ainsi observer le déplacement de l'objet sur l'image.
 - Avec une caméra pilotable, connaissant la position de l'objet dans l'image. Il est possible d'actionner la caméra pour recentrer l'objet au centre de l'image. Ce qui permet de réaliser un suivi d'objet.



VUE PANORAMIQUE



✓ SUIVI D'UN AVION

- Identification des points d'intérêt de l'avion.
- Suivi d'image en image.



ANNEXES

- Ressources utilisées :

A partir du rapport de synthèse, nous avons extrait les ressources utilisées ci-contre.

```
-----
Start RTL Component Statistics
-----

Detailed RTL Component Info :
+---Adders :
      2 Input    10 Bit      Adders := 2
      9 Input     8 Bit      Adders := 1
     10 Input     8 Bit      Adders := 1
      7 Input     7 Bit      Adders := 2
+---Registers :
                                10 Bit    Registers := 2
                                3 Bit     Registers := 12
                                1 Bit     Registers := 7
+---Muxes :
      2 Input    10 Bit      Muxes := 2
      2 Input     7 Bit      Muxes := 2
      2 Input     4 Bit      Muxes := 6
      2 Input     1 Bit      Muxes := 5
-----
```

Vérifions que ces données sont en phase avec notre architecture.

Component	Detail1	Detail2	Adder (inputs / bits)	Register (bits)	Mux (inputs / bits)
VGA_sync	count_x		1 (2 inputs / 10 bits)	1 (10 bits)	1 (2 inputs / 10 bits)
	count_y		1 (2 inputs / 10 bits)	1 (10 bits)	2 (2 inputs / 10 bits)
	calculate_hsync				
	calculate_in_display				
	calculate_vsync				
shift_sync	calculate_vsync_shifted			2 (1 bit)	
	calculate_hsync_shifted			3 (1 bit)	1 (2 inputs / 1 bit)
system_conv	buffer_module	bottom_shift_register		4 (3 bits)	
		first_pixel_buffer		1 (1 bit)	1 (2 inputs / 4 bits)
		middle_shift_register		4 (3 bits)	
		second_pixel_buffer		1 (1 bit)	1 (2 inputs / 4 bits)
		top_shift_register		4 (3 bits)	
	convolution_module	Filt 1 (Id Mat)			1 (2 inputs / 4 bits)
		Filt 2 (Gauss)	1 (9 inputs / 8 bits)		1 (2 inputs / 4 bits)
		Filt 3, 4 (Sobel)	2 (7 inputs / 7 bits)		2 (2 inputs / 7 bits)
		Filt 5 (Edge detection)	1 (10 inputs / 8 bits)		1 (2 inputs / 4 bits)
	corner_detection				3 (2 inputs / 4 bits)

En analysant nos schémas RTL, nous retrouvons exactement le même nombre de « Adders » et de registres. Nous notons une petite différence concernant le nombre de multiplexeur. Ceci est probablement dû aux optimisations réalisées par le logiciel VIVADO.

Report BlackBoxes:

	BlackBox name	Instances
1	PLL	1
2	pixel_FIFO	2

Report Cell Usage:

	Cell	Count
1	PLL	1
2	pixel_FIFO	1
3	pixel_FIFO_	1
4	CARRY4	4
5	LUT1	4
6	LUT2	18
7	LUT3	15
8	LUT4	14
9	LUT5	19
10	LUT6	45
11	FDCE	61
12	FDPE	2
13	IBUF	5
14	OBUF	16

Dans le rapport de synthèse, nous retrouvons également les données ci-contre :

Nous retrouvons bien dans le bilan la PLL et les 2 FIFOs utilisées dans les modules pixel_buffer.

Les 63 registres à 1 bit FDCE et FDPE correspondent bien à notre architecture RTL. En effet, si nous reprenons les registres de la page précédente, nous obtenons le calcul suivant :

$$2 \times 10 + 12 \times 3 + 7 \times 1 = 63$$

Enfin, les entrées/sorties sont bien celles de notre design :

- ✓ 5 entrées (IBUF) :
 - Signal reset (1 bit),
 - Données de l'image à traiter (4 bits).
- ✓ 16 sorties (OBUF) :
 - Images de sortie, couleur Rouge (4 bits),
 - Images de sortie, couleur Verte (4 bits),
 - Images de sortie, couleur Bleue (4 bits),
 - Signal de synchronisation horizontale (1 bit),
 - Signal de synchronisation verticale (1 bit),
 - Signal locked indiquant la stabilisation de l'horloge à 25 MHz (1 bit),
 - Signal cmd_conv, signal avec retard de 802 pixels (1 bit).