



---

## COMPTE RENDU - PROJET FINAL

---

Formation AJC FPGA Eve CHAR



17 JUILLET 2023  
Eve CHAR

## Table des matières

<b>INTRODUCTION.....</b>	<b>2</b>
<b>I. VUE D'ENSEMBLE DU PROJET 'FILTRE DE SOBEL'.....</b>	<b>3</b>
1) SCHEMA BLOC DU SYSTEME .....	3
2) DECOMPOSITION DE LA SPECIFICATION CLIENT EN EXIGENCES .....	3
3) HIERARCHIE DES DIFFERENTS MODULES .....	4
4) ETUDE DE FAISABILITE ET PLAN DE VALIDATION .....	4
<b>II. DETECTION DE POINTS D'INTERET (ETUDE DU FILTRE SOBEL) .....</b>	<b>5</b>
1) COMPREHENSION GLOBALE DU PROJET .....	5
2) ARCHITECTURE DETAILLEE DU MODULE TOP .....	5
3) LOGIQUE DE CALCUL DU FILTRE DE SOBEL:.....	5
4) LOGIQUE D'APPLICATION DU MASQUE ET BALAYAGE DES PIXELS : .....	7
5) GESTION DES PIXELS (BUFFERISASSIONS) : .....	8
6) GESTION DES BORDS DE L'IMAGE (PADDING): .....	10
7) GESTION DES RETARDS (SYNCHRONISATION):.....	11
<b>III. SYNTHESE ET ANALYSE DES TESTS.....</b>	<b>12</b>
1) SYNTHESE DES TESTS.....	12
2) SCHEMA RTL EN SIMULATION SOUS VIVADO : .....	13
3) ANALYSES DES RESSOURCES ELEMENTAIRES UTILISEES EN SIMULATION VIVADO : .....	13
<b>IV. CONCLUSION .....</b>	<b>14</b>
<b>V. BIBLIOGRAPHIE - WEBOGRAPHIE .....</b>	<b>15</b>
<b>ANNEXE.....</b>	<b>15</b>

## Table des Figures

FIGURE 1: CONTEXTE DU PROJET : IP DE TRAITEMENT D'IMAGE POUR DETECTION DE POINTS D'INTERETS .....	3
FIGURE 2: SYNOPTIQUE DE LA METHODOLOGIE DE VALIDATION.....	4
FIGURE 3: IDENTIFICATION DU MODULE TOP A DEVELOPPER.....	5
FIGURE 4 : SCHEMA SIMPLIFIANT LA COMPREHENSION DE L'ARCHITECTURE GLOBALE .....	5
FIGURE 7 : PRINCIPE DE FONCTIONNEMENT DU BUFFERSATION DES FIFOs.....	9
FIGURE 8 : SCHEMA RTL (DETAILLE) DES FIFOs .....	9
FIGURE 9 : DESCRIPTION DE LA PROBLEMATIQUE DES BORDS DANS LES CALCULS DE CONVOLUTION .....	10
FIGURE 10 : RETARD DE 2 COUPS D'HORLOGES DU AU CALCUL DE CONVOLUTION (FILTRE SOBEL) .....	11
FIGURE 11 : RETARD D'UN COUP D'HORLOGE DU AU FIFOs .....	12
FIGURE 12 : MATRICE RESULTAT DES TESTS PAR EXIGENCE .....	12

## Introduction

Ce projet, lancé sur une courte durée (semaine), a pour objectif la réalisation d'une IP de traitement d'image pour détecter les contours et les points d'intérêts d'une image donnée. Le but aussi est de mettre en pratique la conception en langage VHDL (« Very High Speed Integrated Circuit Hardware Description Language »). Ce projet ne nécessite aucune implémentation dans une composante physique FPGA (Field Programmable Gate Array).

Ce document est divisé en trois chapitres qui présentent les différents aspects ayant rapport à l'élaboration de ce projet. Voici donc un plan du contenu de ce document :

- Le premier chapitre présente une vue d'ensemble du projet avec une description de la spécification client qui a été décomposée en exigences / sous-fonctions. Aussi, ce premier chapitre, décrit l'étude de faisabilité avec l'organisation pour structurer et réaliser ce projet.
- Le deuxième chapitre présente l'étude complète du projet. En commençant par l'identification et la compréhension de l'architecture globale du module à développer. Ensuite le développement de l'architecture RTL détaillée. Ensuite, la logique suivie pour effectuer les calculs de convolution du filtre de Sobel à utiliser durant le projet. Puis une explication de la méthodologie à suivre pour appliquer le filtre avec le masque de KERNEL, et la gestion du flux des pixels lus. Finalement, une description des points d'attentions techniques rencontrés et l'explication des solutions retenues pour y remédier : la problématique de gestion des retards ou bien celle de gestion des calculs des bords de l'image par exemple.
- Le troisième chapitre présente les résultats des tests permettant de valider l'implémentation, avec l'analyse des résultats de tests.

Les codes VHDL des différentes parties du projet seront placés en annexe, suivi par les bancs d'essais (tb fournit par notre formatrice).

Des documents sont également joints : la procédure de test (plan de validation) et les résultats de test (Recette).

## I. Vue d'ensemble du projet 'Filtre de SOBEL'

Le projet consiste à valider une IP de traitement d'image : Détection de points d'intérêts (corner détection).



### 1) Schéma bloc du système

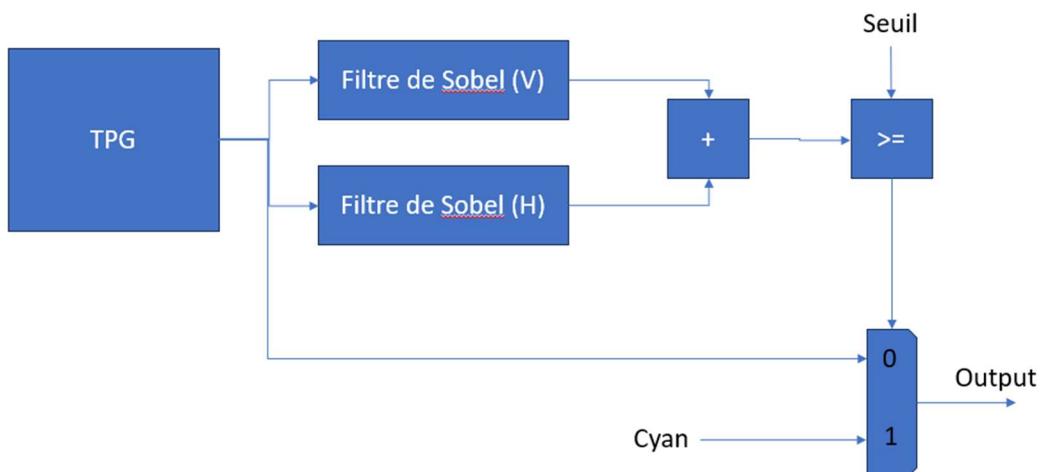


Figure 1: contexte du projet : IP de traitement d'image pour détection de points d'intérêts

### 2) Décomposition de la spécification client en exigences

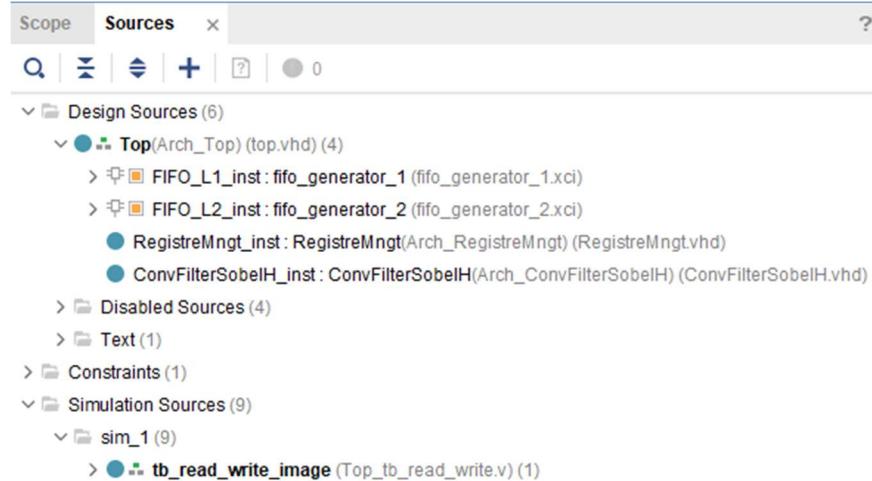
Cette étape consiste à identifier les spécifications clés à satisfaire.

Il est important donc de connaître les fonctions à valider. On identifie la liste suivante :

Fct-lecture-Image – gen01	fonction de lecture d'un fichier de type .txt
Fct-gestion buffers – 02	fonction de gestion des pixels (bufferisation)
Fct- de calcul de convolution -Filtre de Sobel– 03	fonction de calcul de convolution à la méthode de SOBEL
Fct- threshold-comp-04	fonction de vérification par rapport à la valeur seuil
Fct-écriture-image – 05	fonction d'écriture d'un fichier de type .txt
Fct-comparison globale FIJI – 06	fonction comparison globale avec logiciel FIJI

### 3) Hiérarchie des différents modules

Cette section permet de situer rapidement chacun des modules dans la hiérarchie du système en partant du module de plus haut niveau qui est le module top et en descendant vers le bas où se trouvent les modules les plus simples. Voici donc cette arborescence des modules :



### 4) Etude de faisabilité et plan de validation

Ci-dessous des étapes d'étude et de validation de ce projet :

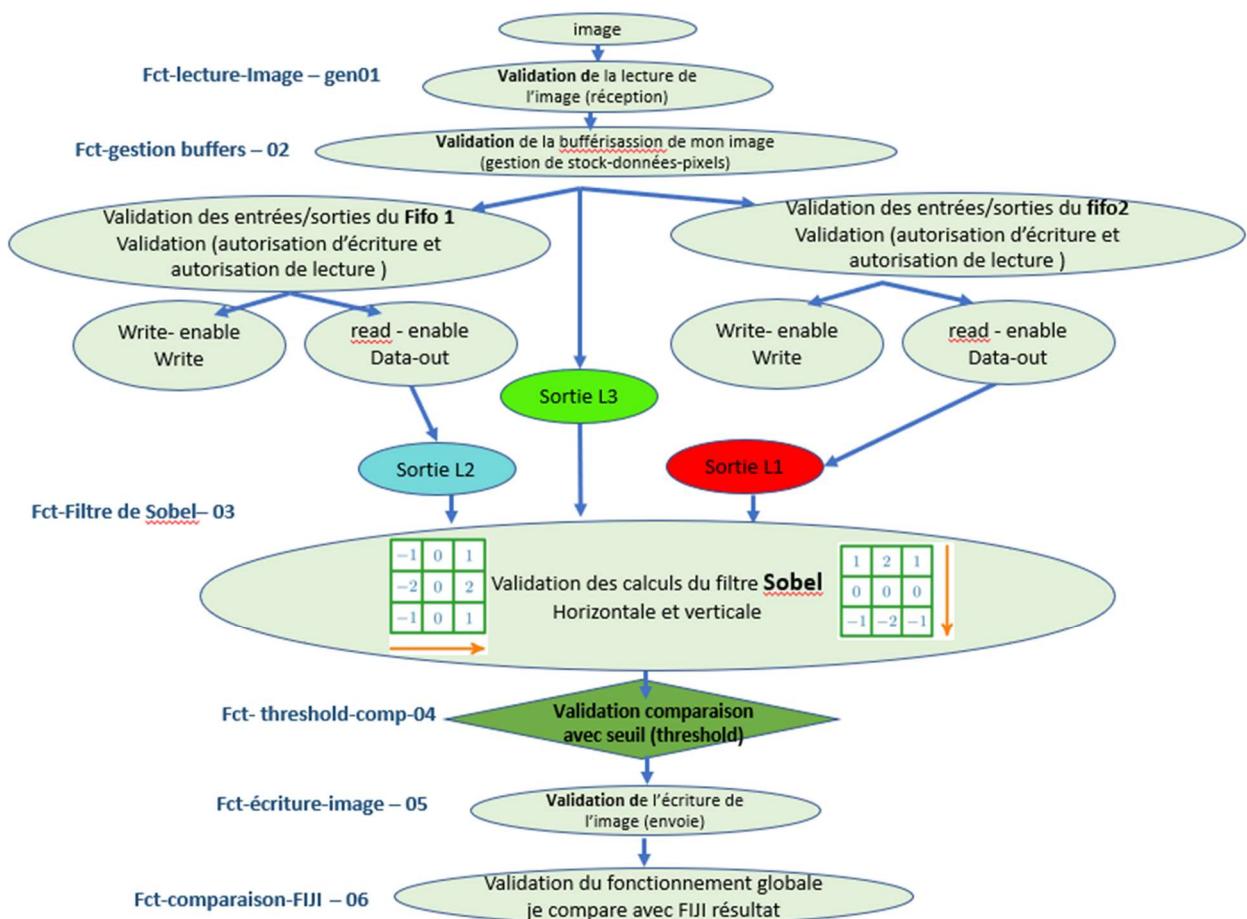


Figure 2: synoptique de la méthodologie de validation

## II. Détection de points d'intérêt (Etude du filtre SOBEL)

### 1) Compréhension globale du projet

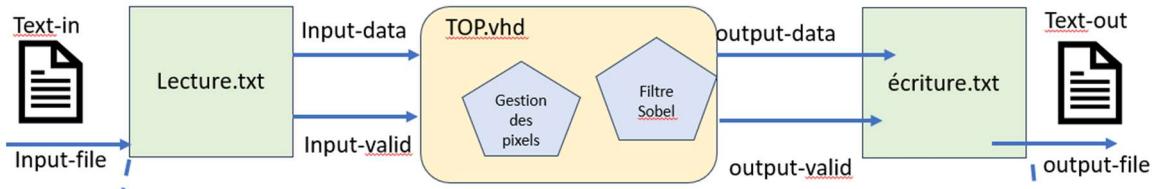


Figure 3: identification du Module Top à développer

### 2) Architecture détaillée du module top

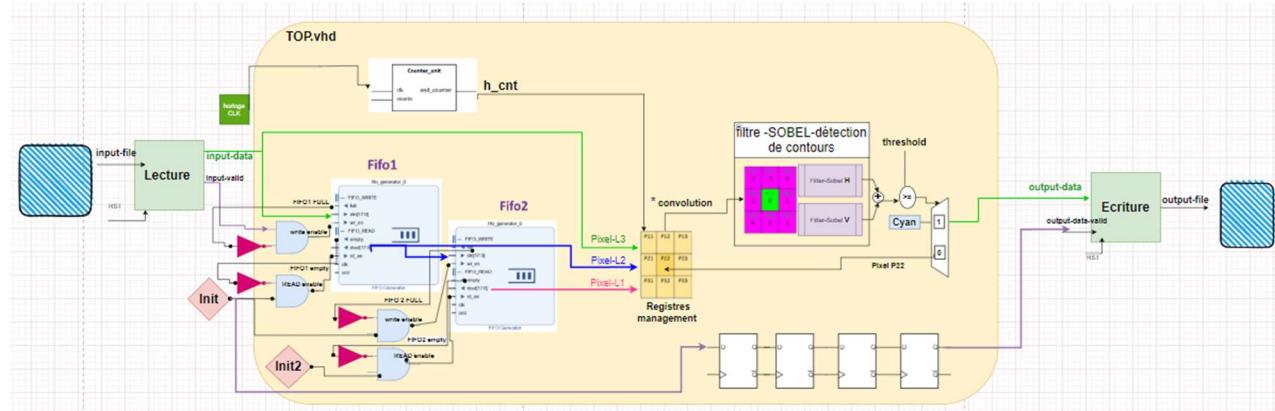


Figure 4 : schéma simplifiant la compréhension de l'architecture globale

### 3) Logique de calcul du filtre de Sobel:

Le filtre de Sobel est un opérateur utilisé en traitement d'image pour la détection de contours. L'opérateur calcule le gradient de l'intensité de chaque pixel. Ceci indique la direction de la plus forte variation du clair au sombre, ainsi que le taux de changement dans cette direction. On connaît alors les points de changement soudain de luminosité, correspondant probablement à des bords, ainsi que l'orientation de ces bords.

En termes mathématiques, le gradient d'une fonction de deux variables (ici l'intensité en fonction des coordonnées de l'image) est un vecteur de dimension 2 dont les coordonnées sont les dérivées selon les directions horizontale et verticale.

(Source : [https://fr.wikipedia.org/wiki/Filtre\\_de\\_Sobel](https://fr.wikipedia.org/wiki/Filtre_de_Sobel))

L'opérateur utilise **des matrices de convolution**. La matrice de taille  $3 \times 3$  subit une convolution avec l'image pour calculer des approximations des dérivées horizontale et verticale.

Le principe est de faire un filtrage par convolution 2D et d'employer une architecture appelée « sliding window » architecture avec deux filtres :

 Matrice horizontale	 Matrice verticale
$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A}$	$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{A}$

Figure 5 : calcul des approximations des dérivées horizontale et verticale.

L'opération de filtrage par convolution est une somme pondérée, et donc c'est une opération linéaire : les valeurs des pixels de l'image filtrée sont des combinaisons linéaires des valeurs des pixels de l'image d'origine. C'est pourquoi les filtres réalisés par convolution sont appelés **filtres linéaires**.

masque kernel utilisé 3x3 :	Masque Sobel	Masque Sobel H + Sobel V :																		
<b>C1 C2 C3</b> <b>L1</b>   K11 K12 K13   <b>L2</b>   K21 K22 K23   <b>L3</b>   K31 K23 K33	<b>(HORIZONTAL) :</b> $\begin{vmatrix} -1 & 0 & 1 \end{vmatrix}$ $\begin{vmatrix} -2 & 0 & 2 \end{vmatrix}$ $\begin{vmatrix} -1 & 0 & 1 \end{vmatrix}$ <b>(VERTICAL) :</b> $\begin{vmatrix} -1 & -2 & -1 \end{vmatrix}$ $\begin{vmatrix} 0 & 0 & 0 \end{vmatrix}$ $\begin{vmatrix} 1 & 2 & 1 \end{vmatrix}$	$\begin{vmatrix} (-1-1) & -2 & (1-1) \end{vmatrix} \quad \begin{vmatrix} -2 & -2 & 0 \end{vmatrix}$ $\begin{vmatrix} (0-2) & 0 & 2 \end{vmatrix} \Rightarrow \begin{vmatrix} -2 & 0 & 2 \end{vmatrix}$ $\begin{vmatrix} 0 & 2 & (1+1) \end{vmatrix} \quad \begin{vmatrix} 0 & 2 & 2 \end{vmatrix}$ <table border="1" style="margin-left: auto; margin-right: auto; text-align: center;"> <tr> <td>P11</td><td>P12</td><td>P13</td></tr> <tr> <td>-2</td><td>-2</td><td>0</td></tr> <tr> <td>P21</td><td>P22</td><td>P23</td></tr> <tr> <td>-2</td><td>0</td><td>2</td></tr> <tr> <td>P31</td><td>P32</td><td>P33</td></tr> <tr> <td>0</td><td>2</td><td>2</td></tr> </table>	P11	P12	P13	-2	-2	0	P21	P22	P23	-2	0	2	P31	P32	P33	0	2	2
P11	P12	P13																		
-2	-2	0																		
P21	P22	P23																		
-2	0	2																		
P31	P32	P33																		
0	2	2																		

J'ai choisi de baser mon étude sur la somme des deux matrices Verticale et horizontale et non pas de les faire séparément.

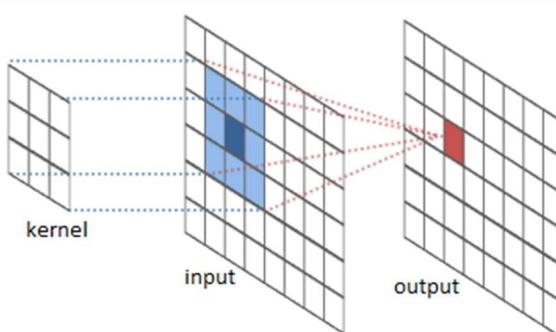
Le calcul ci-après explique le raisonnement pour travailler avec sur une seule matrice somme au lieu de deux matrices.

$\begin{array}{ c c c } \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$	$+$	$\begin{array}{ c c c } \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$																													
$\begin{array}{ c c c } \hline -2 & -2 & 0 \\ \hline -2 & 0 & 2 \\ \hline 0 & 2 & 2 \\ \hline \end{array}$																															
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>sobel H</td><td>-P11</td><td>-2 P21</td><td style="background-color: yellow;">-P31</td><td style="background-color: lightgreen;">+P13</td><td>+2 P23</td><td>+P33</td></tr> <tr> <td>sobel V</td><td>-P11</td><td>-2 P12</td><td style="background-color: lightgreen;">-P13</td><td style="background-color: yellow;">+P31</td><td>+2 P32</td><td>+P33</td></tr> </table>	sobel H	-P11	-2 P21	-P31	+P13	+2 P23	+P33	sobel V	-P11	-2 P12	-P13	+P31	+2 P32	+P33	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="background-color: black; color: white; text-align: center;">SOBEL H+V</td><td>-2 P11</td><td>-2 P12</td><td>+0 P13</td></tr> <tr> <td></td><td>-2 P21</td><td>-0 P22</td><td>+2 P23</td></tr> <tr> <td></td><td>-0 P31</td><td>+2 P32</td><td>+2 P33</td></tr> </table>					SOBEL H+V	-2 P11	-2 P12	+0 P13		-2 P21	-0 P22	+2 P23		-0 P31	+2 P32	+2 P33
sobel H	-P11	-2 P21	-P31	+P13	+2 P23	+P33																									
sobel V	-P11	-2 P12	-P13	+P31	+2 P32	+P33																									
SOBEL H+V	-2 P11	-2 P12	+0 P13																												
	-2 P21	-0 P22	+2 P23																												
	-0 P31	+2 P32	+2 P33																												

Figure 6 : choix de calcul approximative des dérivées horizontale et verticale.

#### 4) Logique d'application du masque et balayage des pixels :

- Les données arrivent dans un premier registre P33 (Pixel\_L3).
- Les pixels se décalent ensuite à chaque coup d'horloge à travers les deux FIFO et les registres, jusqu'au remplissage total des registres (de P11 à P33).
- Le masque se déplace ensuite vers la droite dans la ligne horizontale de l'image à chaque coup d'horloge,
- Etant donnée que le masque est de taille  $3 \times 3$ , nous avons donc 9 registres permettant de stocker les valeurs des 9 pixels du masque.
- La taille des registres est déterminée par la taille d'un pixel : 24 bits ( $3 \times 8$  couleurs)
- On superpose le masque sur le coin gauche de la matrice de l'image, ensuite nous allons multiplier chaque nombre superposé puis additionner le tout.



Balayage de l'image par le masque de convolution : calcul de la valeur de sortie à comparer avec la valeur du threshold, valeur dont les coordonnées correspondent au pixel au milieu du masque

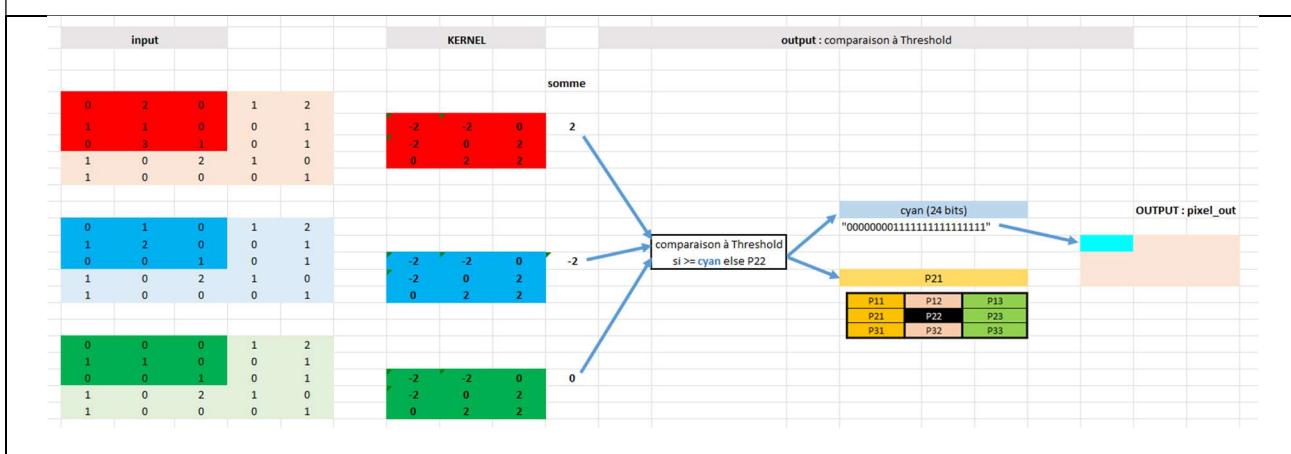


Figure 7 : tableau explicatif de calcul de convolution et application du filtre SOBEL .

- Le balayage se poursuit pixel par pixel puis ligne par ligne sur toute l'image.

Etant donné que tous les coefficients sont des **multiples de 2**, nous allons utiliser les décalages de bits pour les opérations de multiplication.

A chaque coup d'horloge, on décale le masque vers la droite (de 1 pixel), donc les pixels à considérer vont vers la gauche Pixel\_11 <= Pixel\_12, Pixel\_21 <= Pixel\_22, ... Pixel\_31 <= Pixel\_32 <= Pixel\_33. Les valeurs de Pixel\_13, Pixel\_23 et Pixel\_33 sont les nouvelles valeurs à considérer, donc les inputs de la fonction qui sont Pixel\_L1, Pixel\_L2 et Pixel\_L3.

Les pixels utilisés dans le calcul sont nommés

```
-- C1      C2      C3
-- L1 |Pixel_11 Pixel_12 Pixel_13|
-- L2 |Pixel_21 Pixel_22 Pixel_23|
-- L3 |Pixel_31 Pixel_33 Pixel_33|
```

- Pixel\_out\_ConvFilterSobelH sera la valeur utilisée pour comparer à un seuil afin de déterminer s'il s'agit d'un point d'intérêt ou non.
  - Un pixel RGB de 24 bits contient 8bits par composante de couleur
- La somme doit pouvoir contenir des valeurs (somme minimale et somme maximale) entre (8bit x 6) et - (8bits x 6) => somme de 3 multiplications (x (-2)) ou somme de 3 multiplications (x 2).
- [255 x (-6) ; 255 x 6] => [-1530 ; 1530] => Besoin d'une **variable signée** qui va de [-1530 ; 1530]
  - 1530 tient dans 11bits + 1 bit pour le signe => 12 bits

```
architecture Arch_ConvFilterSobelH of ConvFilterSobelH is
  signal Pixel_out_somme_gris : integer;

begin
  process (pxl_clk)
    begin
      if rising_edge(pxl_clk) then
        Pixel_out_somme_gris <= to_integer(signed((Pixel_33(7 downto 0) & '0')
          + (Pixel_32(7 downto 0) & '0')
          + (Pixel_23(7 downto 0) & '0')
          - (Pixel_11(7 downto 0) & '0')
          - (Pixel_21(7 downto 0) & '0')
          - (Pixel_12(7 downto 0) & '0')));
        -- Calcul de la valeur absolue de l'intensité calculée
        -- afin de détecter les pics positifs et négatifs
        if Pixel_out_somme_gris < 0 then -- si négatif => rendre positif
          Pixel_out_ConvFilterSobelH <= -Pixel_out_somme_gris;
        -- transforme la valeur négative en valeur positive pour comparer au threshold
        else
          Pixel_out_ConvFilterSobelH <= Pixel_out_somme_gris; -- si positif, garder la valeur
        end if;
      end if;
    end process;
end Arch_ConvFilterSobelH;
```

Figure 8 : écriture VHDL de la gestion de la calcul des variable signées.

## 5) Gestion des pixels (bufférisassions) :

A la sortie du module « lecture » les pixels passent par les deux fifos pour arriver dans le module de calcul de convolution.

Ces deux FIFOs assurent la gestion d'ordres des pixels et de leurs synchronisations avec le compteur de temporisation (h\_cnt).

- Les pixels successifs sont stockés dans FIFO1
- Une fois la 1<sup>e</sup> ligne entièrement lue, on autorise à lire le contenu de FIFO1 et à écrire dans FIFO2
- Une fois la 2<sup>e</sup> ligne entièrement lue, on autorise à lire le contenu de FIFO2

Les pixels successifs sont stockés dans FIFO1

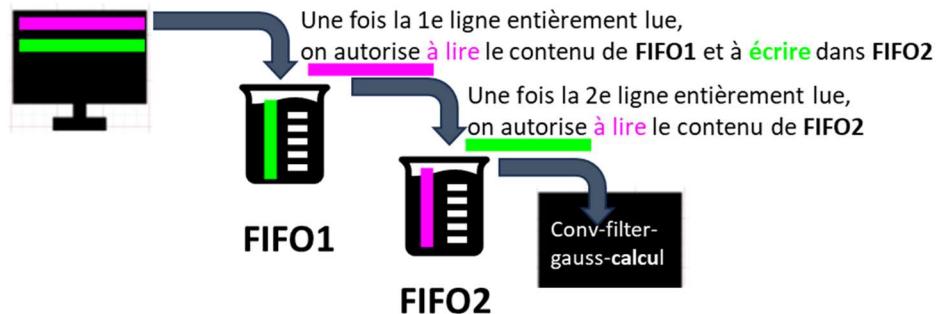


Figure 7 : principe de fonctionnement du buffersation des FIFOs

```

FIFO_1_write_ena <= not(FIFO_1_full) and input_data_valid;
--on écrit dans FIFO_1 dès le début
FIFO_1_read_ena <= not(FIFO_1_empty) and Init;
]--on lit dans FIFO_1 si FIFO_1 n'est pas vide
--et si la première ligne a été entièrement lue
FIFO_2_write_ena <= not(FIFO_2_full) and Init;
]--on écrit dans FIFO_2 si FIFO_1 n'est pas plein
--et si la première ligne a été entièrement lue (
FIFO_2_read_ena <= not(FIFO_2_empty) and Init2;
--on lit dans FIFO_2 si FIFO_2 n'est pas vide et si la 2e ligne a été entièrement lue

signal Init, Init2 : std_logic := '0';-- variable pour reperer le remplissage des fifos

```

- Pixel\_L3 est le pixel qui sort de la lecture « `input data` ». C'est celui qui viendra en bas à droite de notre filtre (P33). C'est aussi celui qui rentre dans FIFO1.
- Pixel\_L2 est le pixel de la ligne au-dessus du pixel courant Pixel\_L3. C'est aussi le pixel qui viendra au milieu à droite de notre filtre (P23) et également en entrée du FIFO2.
- Pixel\_L1 est le pixel de la ligne au-dessus du pixel Pixel\_L2 donc 2 lignes au-dessus du pixel courant Pixel\_L3. C'est aussi le pixel qui viendra en haut à droite de notre filtre (P13).

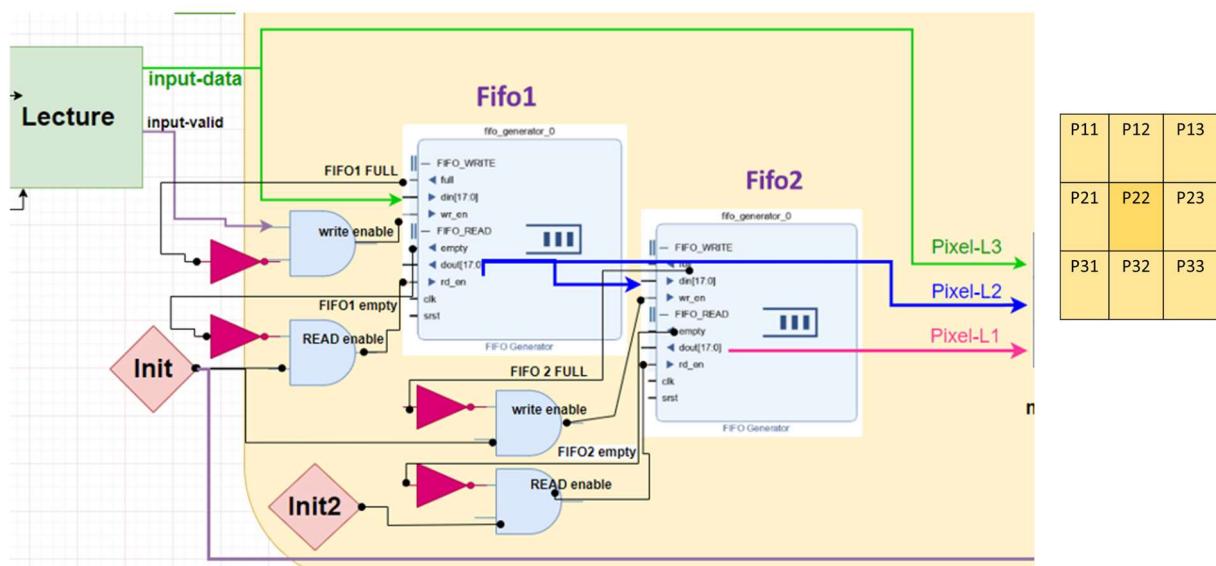


Figure 8 : shema RTL (detaillé) des FIFOs

## 6) Gestion des bords de l'image (padding):

On rappelle que nous traitons le pixel P22 au milieu de la matrice 3X3 :

P11	P12	P13
P21	P22	P23
P31	P32	P33

Quand nous balayons les bords d'une image, notre masque est découpé, à la sortie nous n'obtenons pas les bonnes valeurs calculées attendus.

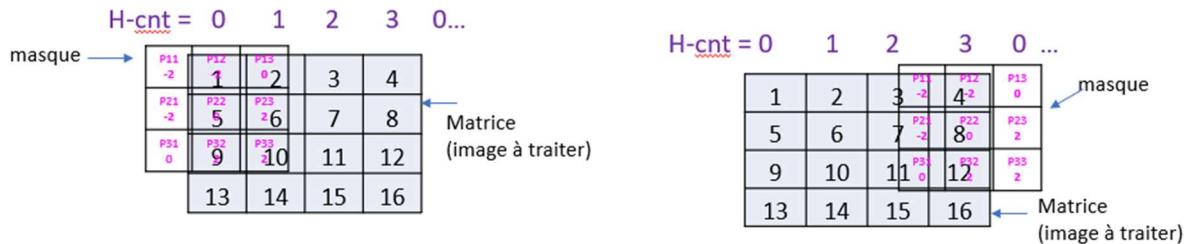


Figure 9 : description de la problématique des bords dans les calculs de convolution

On peut voir que pour les pixels du bord, le masque déborde de l'image. Les valeurs des pixels au-delà du bord sont donc indéfinies. Pour obtenir des calculs plus justes, les pixels en dehors de l'image sont forcés à des valeurs nulles.

### Explication :

=> si h\_cnt vaut 1 alors la matrice est coupée en 2, dans ce cas de figure, on force artificiellement le contenu de la colonne de droite à 0

P13 <= (others =>'0'); -- Pixel en haut à droite => forcé à 0

P23 <= (others =>'0'); -- Pixel du milieu à droite => forcé à 0

P33 <= (others =>'0'); -- Pixel du bas à droite => forcé à 0

Pixel\_13\_svg <= Pixel\_L1; -- sauvegarde de la valeur de la colonne substituée

Pixel\_23\_svg <= Pixel\_L2; -- sauvegarde de la valeur de la colonne substituée

Pixel\_33\_svg <= Pixel\_L3; -- sauvegarde de la valeur de la colonne substituée

=> si h\_cnt vaut 2 alors la matrice est coupée en 2, dans ce cas de figure, on force artificiellement le contenu de la colonne de gauche à 0, il faut également récupérer le vrai contenu de la colonne de droite précédemment substituée

P11 <= (others =>'0'); -- Pixel en haut à gauche => forcé à 0

P21 <= (others =>'0'); -- Pixel du milieu à gauche => forcé à 0

P31 <= (others =>'0'); -- Pixel du bas à gauche => forcé à 0

P12 <= Pixel\_13\_svg; => récupération de la colonne précédemment substituée

P22 <= Pixel\_23\_svg; => récupération de la colonne précédemment substituée

P32 <= Pixel\_33\_svg; => récupération de la colonne précédemment substituée

P13 <= Pixel\_L1;

P23 <= Pixel\_L2;

P33 <= Pixel\_L3;

Ce qui nous donne ce qui suit :

- Eve CHAR - 10/07/2023 – 17/07/2023

```
-- Gestion du bord => si h_cnt vaut 0 alors la matrice est coupée en 2
-- dans ce cas de figure, on force artificiellement le contenu de la colonne de droite à 0
if h_cnt = 0 then
    P11 <- P12;
    P21 <- P22;
    P31 <- P32;
    P12 <- P13;
    P22 <- P23;
    P32 <- P33;
    P13 <- (others=>"0"); -- Pixel en haut à droite => forcé à 0
    P23 <- (others=>"0"); -- Pixel du milieu à droite => forcé à 0
    P33 <- (others=>"0"); -- Pixel du bas à droite => forcé à 0
Pixel_13_svg <- Pixel_L1; -- sauvegarde de la valeur de la colonne substituée
Pixel_23_svg <- Pixel_L2; -- sauvegarde de la valeur de la colonne substituée
Pixel_33_svg <- Pixel_L3; -- sauvegarde de la valeur de la colonne substituée
```

H cnt = 1		
P11 -2	P12 -2	P13 0
P21 0	P22 5	P23 6
P31 2	P32 9	P33 10
13	14	15
16		

H cnt = 2

```
Gestion du bord => si h_cnt vaut 1 alors la matrice est coupée en 2
dans ce cas de figure, on force artificiellement le contenu de la colonne de gauche à 0
il faut également récupérer le vrai contenu de la colonne de droite substituée quand h_cnt vaut 0
```

```
if h_cnt = 1 then
    P11 <- (others=>"0"); -- Pixel en haut à gauche => forcé à 0
    P21 <- (others=>"0"); -- Pixel du milieu à gauche => forcé à 0
    P31 <- (others=>"0"); -- Pixel du bas à gauche => forcé à 0
    P12 <- Pixel_13_svg; -- Pixel de la colonne du milieu => récupération de la colonne précédemment substituée
    P22 <- Pixel_23_svg; -- Pixel de la colonne du milieu => récupération de la colonne précédemment substituée
    P32 <- Pixel_33_svg; -- Pixel de la colonne du milieu => récupération de la colonne précédemment substituée
Pixel_13_ll; -- Pixel de la colonne du milieu
Pixel_23_ll;
Pixel_33_ll;
```

h-cnt = 0			1	2	3	P11 -2	P12 -2	P13 0
P11 -2	P12 1	P13 2		3	4	P21 0	P22 2	P23 2
P21 0	P22 5	P23 6		7	8	P31 2	P32 2	P33 2
P31 2	P32 9	P33 10		11	12			
0	1	2		3	4			
13	14	15		11	12			
16								

Pixel en cours de lecture => en jaune  
Pixel en traitement => en vert

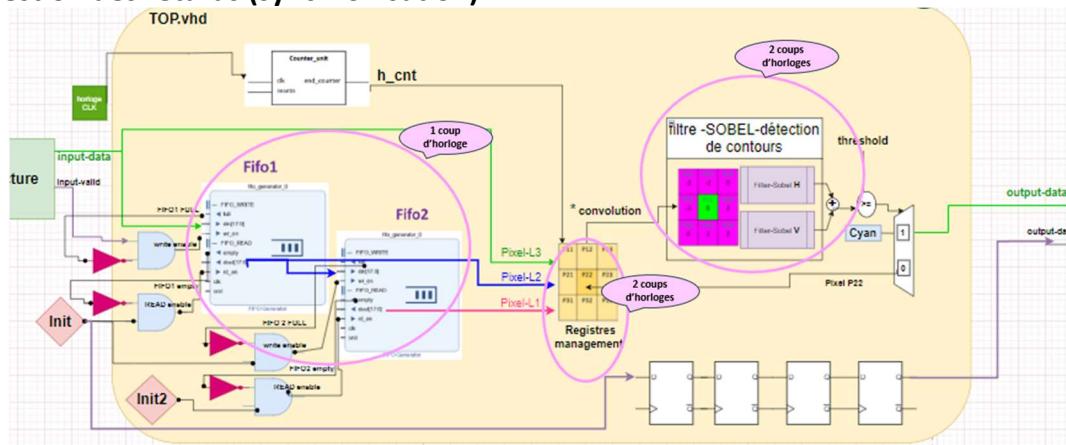
h-cnt = 1			1	2	3	P11 -2	P12 -2	P13 0
P11 -2	P12 1	P13 2		3	4	P21 0	P22 2	P23 2
P21 0	P22 5	P23 6		7	8	P31 2	P32 2	P33 2
P31 2	P32 9	P33 10		11	12			
0	1	2		3	4			
13	14	15		11	12			
16								

Gestion des bords:  
Forçage à 0

h-cnt = 2			1	2	3	P11 -2	P12 -2	P13 0
P11 -2	P12 1	P13 2		3	4	P21 0	P22 2	P23 2
P21 0	P22 5	P23 6		7	8	P31 2	P32 2	P33 2
P31 2	P32 9	P33 10		11	12			
0	1	2		3	4			
13	14	15		11	12			
16								

Formation FPGA-Eve CHAR- 18 juillet 2023

## 7) Gestion des retards (synchronisation):



Après différents ajustements, le constat est le suivant :

- 1 retard d'un coup d'horloge dû au FIFOs
- 1 retard de 2 coups d'horloges dû à la gestion des registres
- 1 retard de 2 coups d'horloges dû au calcul de convolution (filtre SOBEL)

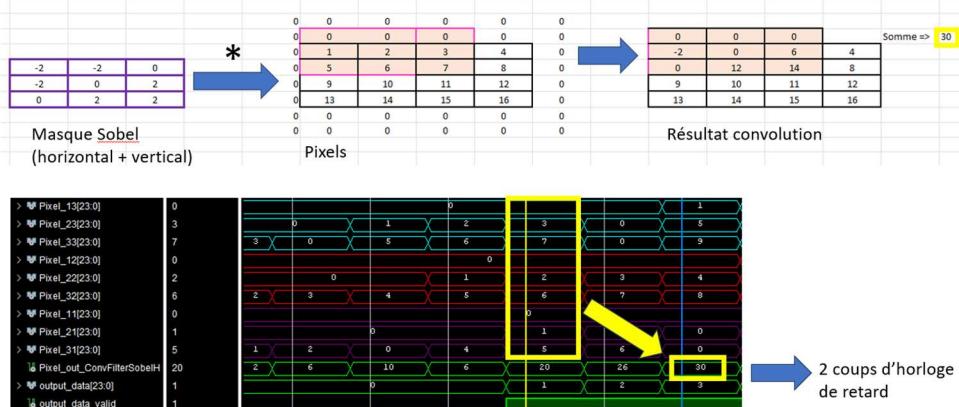


Figure 10 : retard de 2 coups d'horloges dû au calcul de convolution (filtre SOBEL)

- Eve CHAR - 10/07/2023 – 17/07/2023

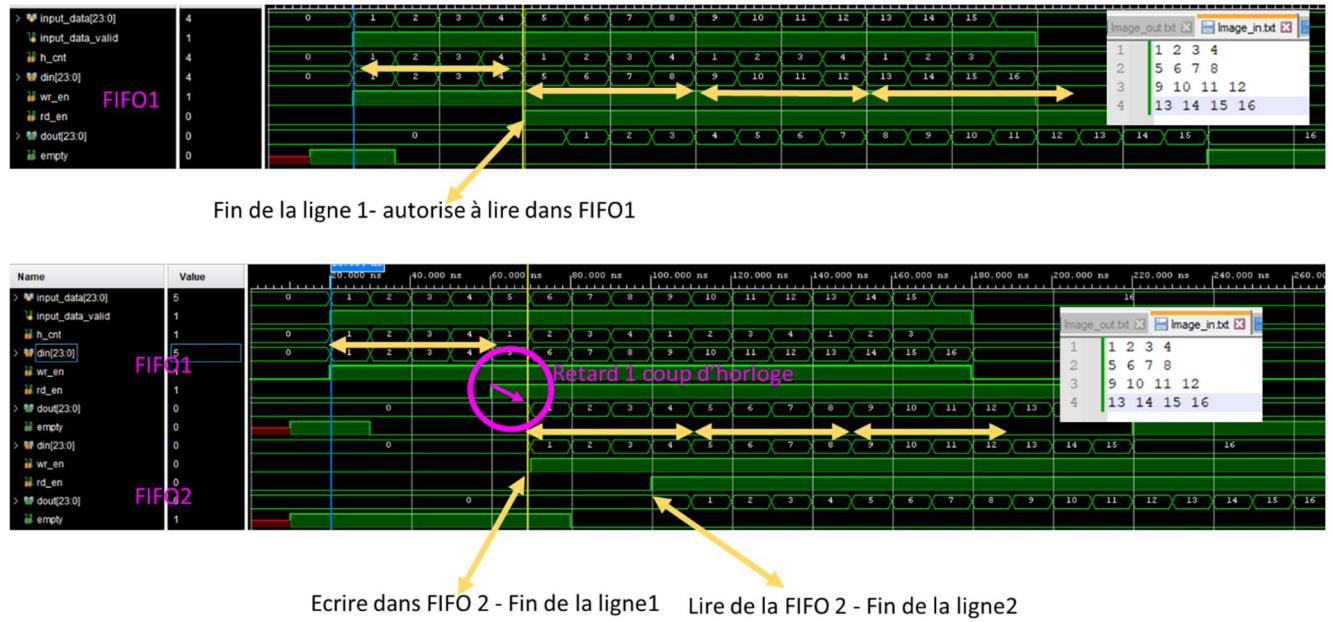


Figure 11 : retard d'un coup d'horloge dû au FIFOs

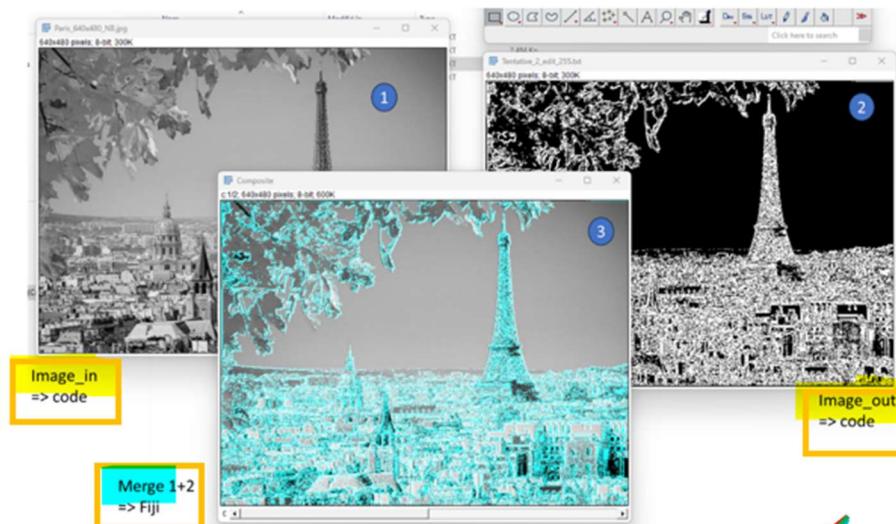
### III. Synthèse et analyse des tests

- Les procédures de test sont dans un document joint (plan de validation)
- Les résultats de ces tests sont présentés dans un autre document (cahier de recette) dont le tableau ci-dessous résume les résultats obtenus.

#### 1) Synthèse des tests

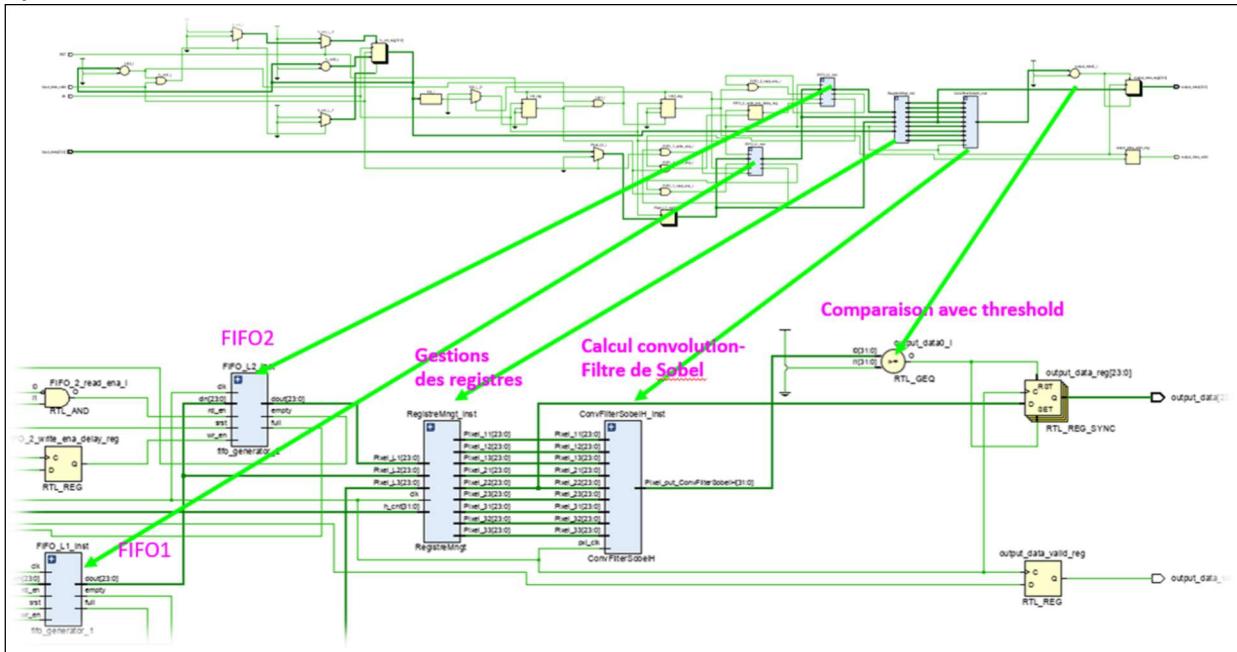
	TEST BENCH
Fct-lecture-Image – gen01	PASS
Fct-gestion buffers – 02	PASS
Fct- de calcul de convolution -Filtre de Sobel– 03	PASS
Fct-threshold-comp-04	PASS
Fct-écriture-image – 05	PASS
Fct-comparaison globale FIJI – 06	PASS

Figure 12 : Matrice résultat des tests par exigence



- Eve CHAR - 10/07/2023 – 17/07/2023

## 2) Schéma RTL en simulation sous VIVADO :



## 3) Analyses des ressources élémentaires utilisés en simulation VIVADO :

```

Start RTL Component Statistics
-----
Detailed RTL Component Info :
+---Adders :
    2 Input   32 Bit      Adders := 2
    7 Input   9 Bit       Adders := 1
+---Registers :
    32 Bit   Registers := 2
    24 Bit   Registers := 23
    9 Bit    Registers := 1
    1 Bit    Registers := 9
+---Muxes :
    2 Input   32 Bit      Muxes := 1
    3 Input   32 Bit      Muxes := 1
    3 Input   24 Bit      Muxes := 3
    2 Input   24 Bit      Muxes := 1
    3 Input   1 Bit       Muxes := 1
    2 Input   1 Bit       Muxes := 1
Finished RTL Component Statistics
-----
```

Report Cell Usage:		
	Cell	Count
1	fifo_generator_1_bbox	1
2	fifo_generator_2_bbox	1
3	BUF	1
4	CARRY4	16
5	LUT1	3
6	LUT2	10
7	LUT3	15
8	LUT4	29
9	LUT5	18
10	LUT6	36
11	SRLL16E	1
12	FDRE	221
13	IBUF	27
14	OBUF	25

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	-	-	-
0	-	-	Set
0	-	-	Reset
0	-	Set	-
0	-	Reset	-
0	Yes	-	-
0	Yes	-	Set
0	Yes	-	Reset
0	Yes	Set	-
0	Yes	Reset	-
221	Yes	Set	-

## IV. Conclusion

Ce projet, nous a permis de s'intéresser particulièrement à la partie simulation et banc de test via l'outil VIVADO de chez XILINX. Le but est de développer une IP en VHDL qui permet de lire un fichier image de type TXT, lui faire appliquer un filtre, ici dans notre cas, il s'agit du filtre de SOBEL, qui détecte les points d'intérêts et les contours. Enfin d'écrire le résultat sous forme d'un fichier image au format texte.

A la l'aide d'un logiciel externe (ici dans notre cas FIJI) nous récupérons le fichier TXT en sortie de notre code VHDL, et nous le fusionnons avec l'image d'origine en composante GRISE sur 8bits, et nous identifions ainsi les contours et les points d'intérêts.

Les livrables pour ce projet sont une conception fonctionnelle (incluant le code) avec son plan de validation associé et ses résultats de test.

La réalisation de ce projet a nécessité d'atteindre différents objectifs :

- Le premier était de maîtriser les outils de conceptions du FPGA et de comprendre l'architecture de ce type de circuit.
- Le deuxième objectif a été d'intégrer un filtre de SOBEL basé sur deux filtres : un horizontal et un vertical avec une matrice de Kernel de taille 3 par 3. Ensuite d'assurer sa mise en œuvre, comprendre son mode de fonctionnement, l'intégration dans le projet et enfin son implémentation (application sur une image).
- Le dernier objectif était de finaliser le plan de validation : expliquer le « quoi » et le « comment » faire pour valider les différents aspects fonctionnels du projet et vérifier en parallèle si nous sommes toujours conformes à la spécification en entrée de ce projet.

## V. Bibliographie - Webographie

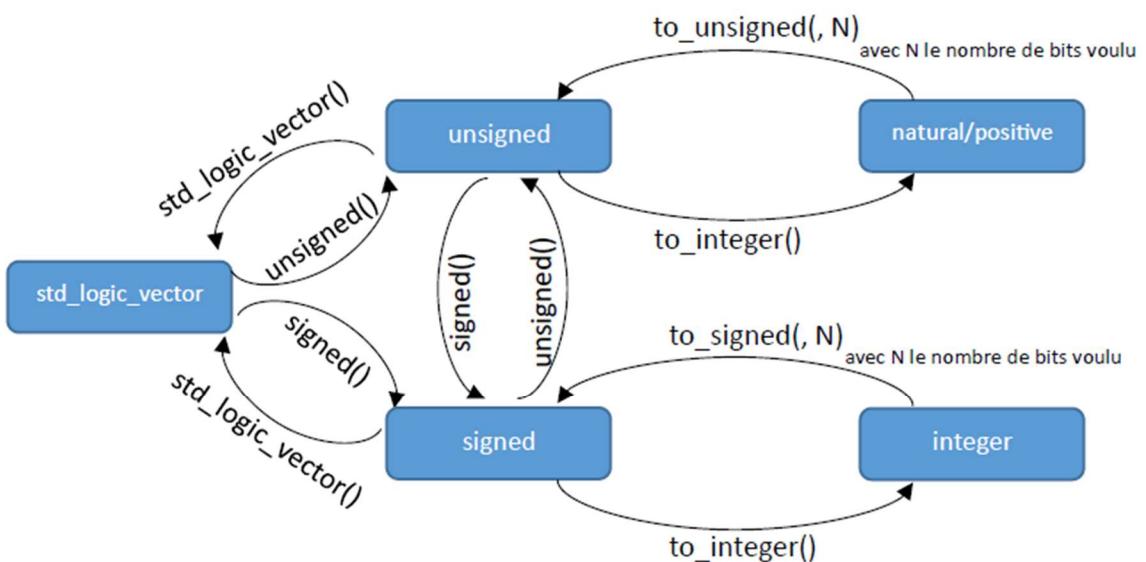
[https://fr.wikipedia.org/wiki/Filtre\\_de\\_Sobel](https://fr.wikipedia.org/wiki/Filtre_de_Sobel)

<https://vhdlwhiz.com/signed-unsigned/>

## ANNEXE

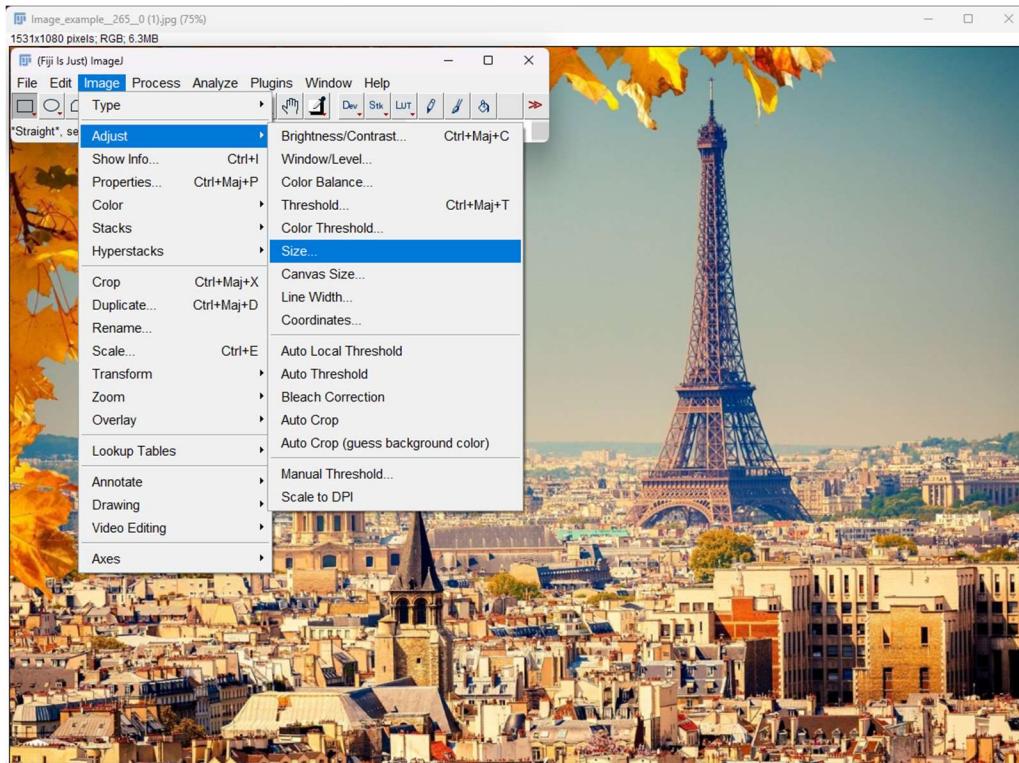
Annexe 1 : Rappel du cours sur les conversions à tenir en compte dans notre code VHDL

### Conversion de types

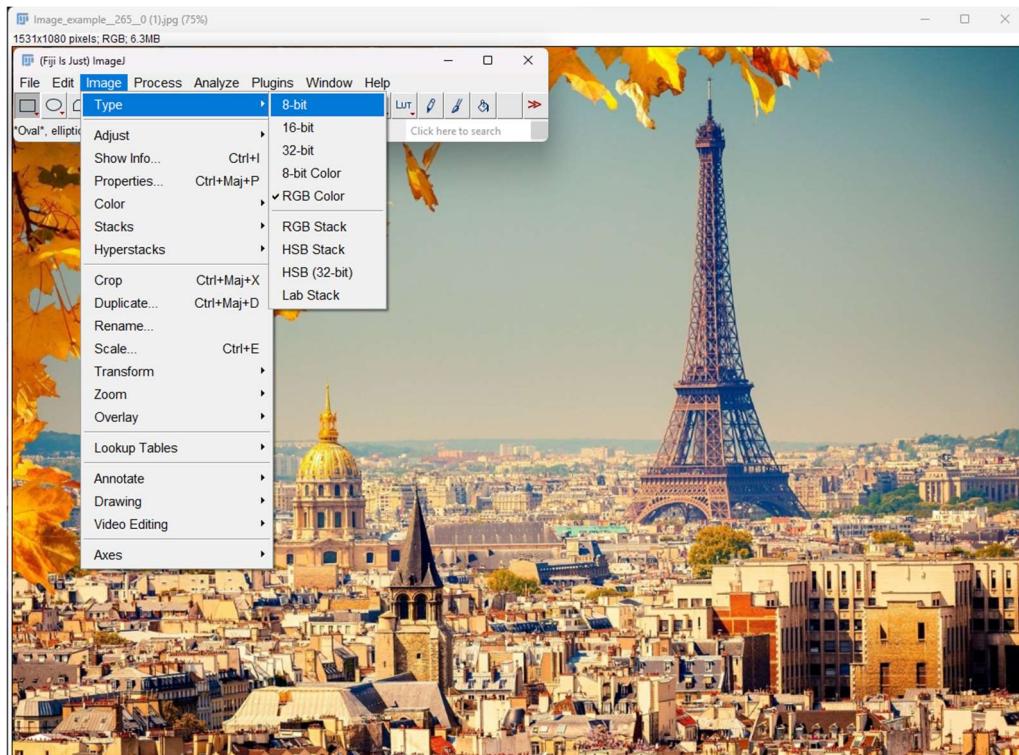


- Eve CHAR - 10/07/2023 – 17/07/2023

## Annexe 2 : Utilisation de Fiji



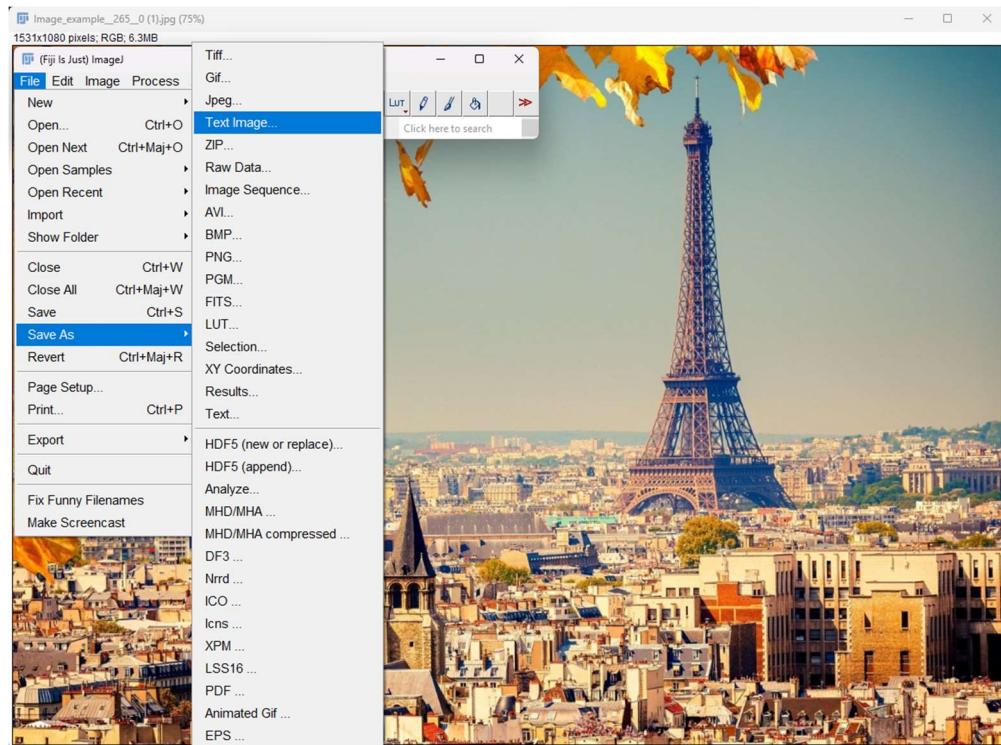
Redimensionner une image (480 x 640)



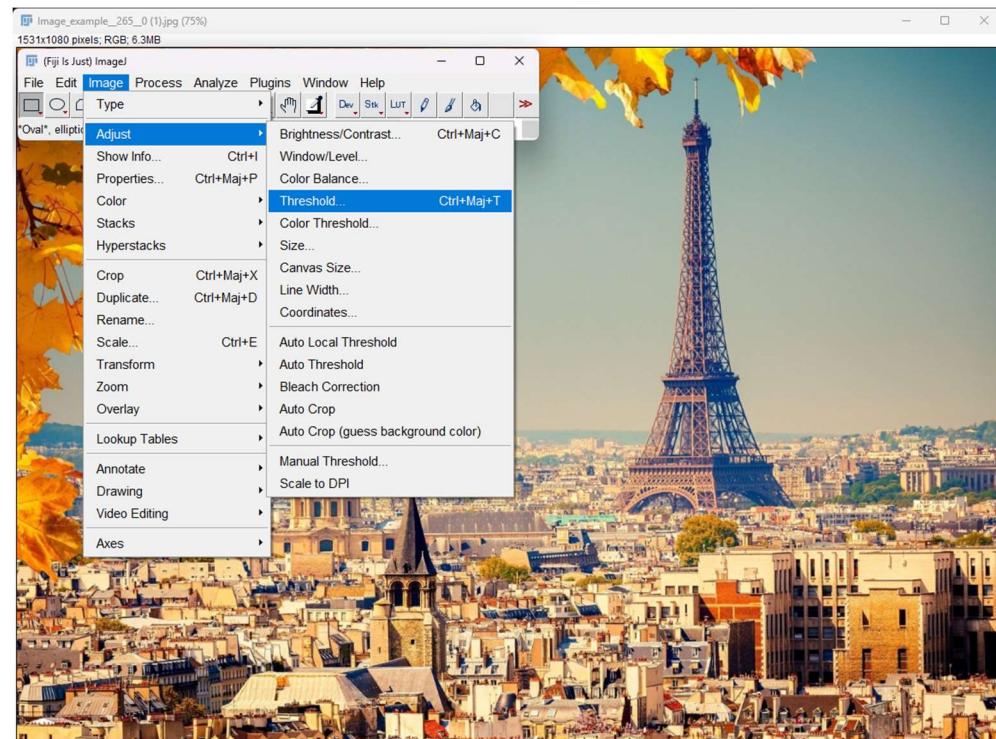
Convertir une image dans un autre format (ici nuance de gris en 8bits)

# COMPTE RENDU DEVELOPPEMENT PROJET FINAL -formation FPGA-AJC

- Eve CHAR - 10/07/2023 – 17/07/2023



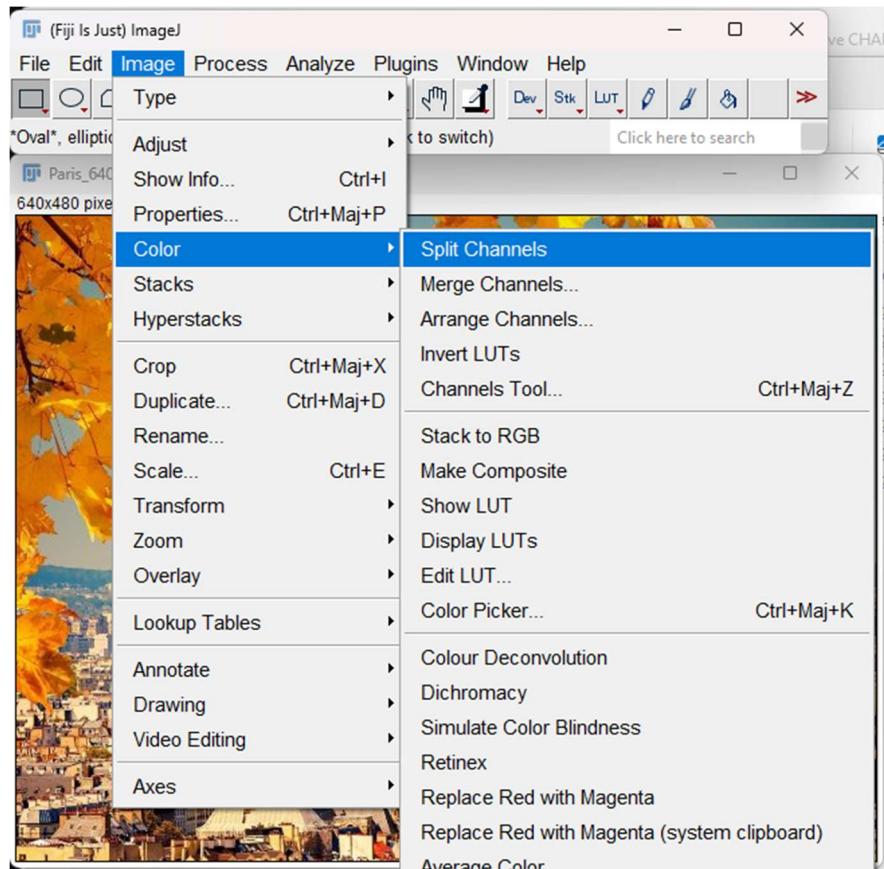
Enregistrer l'image dans une autre extension (ici au format .txt)



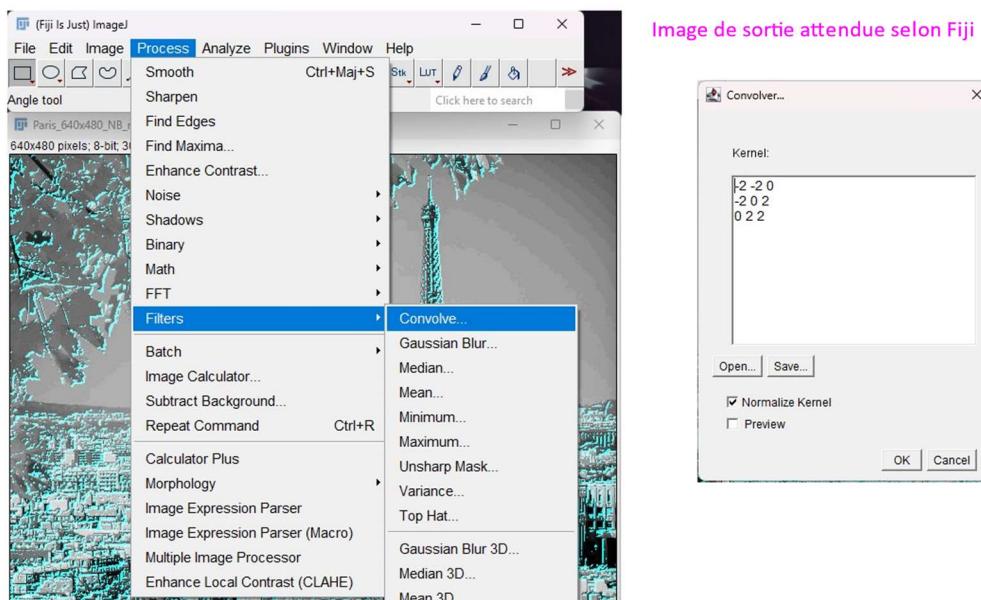
Définir une valeur seuil

## COMPTE RENDU DEVELOPPEMENT PROJET FINAL -formation FPGA-AJC

- Eve CHAR - 10/07/2023 – 17/07/2023



Séparer les plans de couleur de l'image



Appliquer une matrice personnalisée de convolution (ici celui de ce projet)