REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj

Faculté *des Sciences et de la technologie*

Département d'Électronique

Mémoire

Présenté pour obtenir

LE DIPLOME DE MASTER

FILIERE : **ELECTRONIQUE**

Spécialité : Electroniques des Systèmes Embarqués.

Par

- ***SOUKHAL Fairouz.***
- ***NAIDJI Nabila.***

Intitulé

Implémentation des filtres non linéaires 2D sur FPGA avec HDL Coder

Soutenu le : ……

Devant le Jury composé de :

Nom & Prénom	*Grade*	*Qualité*	*Etablissement*
M. SARRA Mustapha	*Pr*	*Président*	*Univ-BBA*
Mme.HAMADACHE Fouzia	*MAA*	*Encadreur*	*Univ-BBA*
M. TALBI Mohamed Lamine	*MCB*	*Examinateur*	*Univ-BBA*

Année Universitaire 2021/2022

Résumé

Le filtre médian 2D est un filtre particulièrement robuste face au bruit impulsionnel, dans ce mémoire, nous nous intéressons à son implémentation sur FPGA Spartan3e (xc3s500e). MATLAB / Simulink combiné avec Xilinx System Generator XSG et HDL coder représentent un outil puissant de conception, test et implémentation des designs sur les systèmes embarqués. Le modèle HDL du filtre médian à base de comparateurs a été modélisé et simulé en utilisant le puissant outil Xilinx System Generator (XSG). L'implémentation du modèle sur FPGA a été réalisée par une compilation pour une Co-Simulation matérielle et par le code VHDL du design généré par HDL Coder.

Mots-clés : Filtre médian, Xilinx System Generator (XSG), FPGA, codeur HDL, Hardware Co-simulation.

Abstract

The 2D median filter is a particularly robust filter for impulse noise, in this brief we are interested in its implementation on FPGA Spartan3e (xc3s500e). MATLAB/ Simulink combined with Xilinx System Generator XSG and HDL coder represent a powerful tool for designing, testing and implementing designs on embedded systems. The HDL model of the median filter based on comparators was modelled and simulated using the powerful Xilinx System Generator (XSG) tool. The implementation of the model on FPGA was carried out by a compilation for a hardware Co-Simulation and by the VHDL code of the design generated by HDL Coder.

Keywords : Median filter, Xilinx System Generator (XSG), FPGA, HDL encoder, Hardware Co-simulation.

ملخص

يعد المرشح المتوسط ثنائي الأبعاد مرشحًا قويًا بشكل خاص للضوضاء الدافعة، وفي هذا الموجز نحن مهتمون بتنفيذه على FPGA Spartan3e (xc3s500e). يمثل MATLAB/Simulink جنبًا إلى جنب مع Xilinx System Generator XSG و HDL coder أداة قوية لتصميم واختبار وتنفيذ التصميمات على الأنظمة المضمنة. تم نمذجة نموذج HDL للمرشح المتوسط القائم على الأسس المقارنة ومحاكاة باستخدام أداة مولد نظام Xilinx القوية (XSG). تم تنفيذ النموذج على FPGA من خلال تجميع لمحاكاة مشتركة للأجهزة ورمز VHDL للتصميم الذي تم إنشاؤه بواسطة HDL Coder.

الكلمات الرئيسية : *مرشح متوسط ، مولد نظام Xilinx (XSG)، FPGA ، مشفر HDL ، محاكاة الأجهزة المشتركة.*

Remerciement

Avant tout nous tenons à remercier dieu qui nous a incités à acquérir le savoir.

Nos remerciements et notre reconnaissance vont à notre encadreur de mémoire, ***madame Hamadache Fouzia****, pour son encadrement exceptionnel, sa grande disponibilité, ses conseils avisés ainsi que pour l'ambiance familiale qu'elle a su créer autour de nous lors de la préparation de ce mémoire.*

Un énorme merci aux membres de jury, vous nous faites un grand honneur en acceptant de juger ce travail.

Comme nous présentons nos remerciements à tous les enseignants du département délectronique.

Merci à toutes les personnes qui ont contribué de près ou de loin à l'élaboration de ce travail,

Merci à tous nos amis et collègues.

Dédicaces

Avant toute chose, nous remercions Allah, notre Dieu qui nous a donné la force et la patience pour accomplir ce travail.

Je dédie ce modeste travail à mon très ***cher père*** *et ma très* ***chère mère*** *pour l'amour qu'ils m'ont toujours donné, leurs encouragements et toute l'aide qu'ils m'ont apportée durant mes études. Aucun mot, aucune dédicace ne pourrait exprimer mon respect, ma considération, et mon amour.*

Qu'ils trouvent ici le témoignage de ma gratitude, laquelle, si grande soit-elle, ne sera jamais égale à leur tendresse et à leur dévouement.

A mon cher frère ***Mohamed***

A ma chère sœur ***Hanane,***

Vous avez toujours cru en moi, et c'est en vous que j'ai puisé la volonté de continuer.

A ma collègue Naidji Nabila et mes amis.

SOUKHAL Fairouz

Dédicaces

Avant tout chose, nous remercions Allah, notre Dieu qui nous a donné la force et la patience pour accomplir ce travail.

Je dédie ce travail à mes chers parents pour leur amour, leurs encouragements, leur soutien indéfectible durant toutes mes années d'étude.

Je dédie également ce travail

À mes frères, mes sœurs, je leur souhaite toute la réussite dans leurs études ainsi que dans leur vies.

A toute ma famille.

A tous mes amis sans oublier ma collègue soukhal Fairouz.

A tous mes amis de classe.

NAIDJI Nabila

Sommaire

Liste des figures

Chapitre I : Débruitage des images par le filtre médian sous Matlab

Chapitre II : Débruitage des images par le filtre médian sous Simulink

Chapitre III : Débruitage des images par le filtre médian sous Xilinx System Generator (XSG)

Liste des tableaux

Chapitre I : Débruitage des images par le filtre médian sous Matlab

Chapitre II : Débruitage des images par le filtre médian sous Simulink

Chapitre III : Débruitage des images par le filtre médian sous Xilinx System Generator (XSG)

Liste des abbreviations

FPGA : ***F****ield* ***P****rogrammable* ***G****ate* ***A****rray*.

VHDL : ***V****ery* ***H****igh-level* ***D****esign* ***L****anguage.*

HDL : ***H****ardware* ***D****escription* ***L****anguage.*

MATLAB *:* ***Mat****rix* ***Lab****oratory.*

MSE *:* ***M****ean* ***S****quare* ***E****rror*.

PSNR *:* ***P****eak* ***S****ignal to* ***N****oise* ***R****atio*.

XSG *:* ***X****ilinx* ***S****ystem* ***G****enerator*.

ISE : ***I****nternational* ***S****ecurities* ***E****xchange.*

IDE : ***I****ntegrated* ***D****esign* ***E****nvironment.*

PLCC : *Plastic Leaded Chip Carrier.*

HIL : ***H****ardware* ***I****n* ***L****oop.*

PC : ***P****ersonal* Computer.

NI : ***N****ational* ***I****nstruments*.

Introduction générale

La suppression du bruit d'une image est très prisée en traitement d'images car il s'agit d'une perturbation qui affecte la qualité de l'image **[1].** Le filtre médian jouit d'une bonne réputation puisqu'il arrive à combiner une suppression du bruit impulsionnel et la conservation des détails importants. L'implémentation de ces filtres nécessite beaucoup de ressources avec l'exigence des résultats en temps réel ; les filtres médians sont désormais mis en œuvre sur une technologie concurrente FPGA (Field Programmable Gate Array). La mise en œuvre de circuits sur FPGA repose habituellement sur des langages de bas niveau dont la flexibilité et la productivité laissent à désirer. MATLAB / Simulink associé au Xilinx System Generator (XSG) constitue un outil puissant pour la conception, le test et l'implémentation de circuits sur des systèmes embarqués. Pour transformer les modèles construits à l'aide de Simulink en matériel, System Generator propose la méthode appelée Hardware/Software Co-simulation qui permet de construire une version matérielle du modèle et d'utiliser l'environnement de simulation flexible de Simulink, pour vérifier la fonctionnalité du système dans le matériel.

Dans le cadre de notre travail, nous implémentons sur FPGA, l'algorithme du filtre médian 3x3 et 5x5 à l'aide de l'outil HDL Coder et l'outil puissant (XSG) sous Simulink. Le présent mémoire est organisé comme suit :

Le premier chapitre aborde le principe du filtrage médian. Le filtre médian (3x3) et (5x5) est testé sur des images bruitées par du bruit poivre et sel à deux densités (D=25% et 3%) et l'image filtrée est examinée en fonction de la métrique d'évaluation PSNR.

Le deuxième chapitre présente l'architecture matérielle du filtre médian qui est à base de comparateurs. Le développement de ce modèle, sa simulation sous l'environnement Simulink et enfin son implémentation sur FPGA à l'aide de l'outil HDL Coder y sont présentés.

Le troisième chapitre est dédié au développement du modèle du filtre médian à l'aide de l'outil XSG, à la simulation du modèle HDL sous l'environnement Simulink et pour finir à son implémentation sur FPGA en utilisant une compilation pour une Co-Simulation matérielle.

Finalement, en conclusion générale, on synthétise le travail et on dégage les pistes de réflexion.

Chapitre I

Débruitage des images par le filtre médian sous Matlab

I.1. Introduction

La suppression du bruit d'une image est très prisée en traitement d'images car il s'agit d'une perturbation qui affecte la qualité de l'image tout en compliquant son interprétation visuelle. Le but du filtrage est de supprimer les traces de ces perturbations tout en préservant les informations vitales de l'image. Des techniques de filtrage sont généralement nécessaires pour améliorer la qualité des images observées **[2]**.

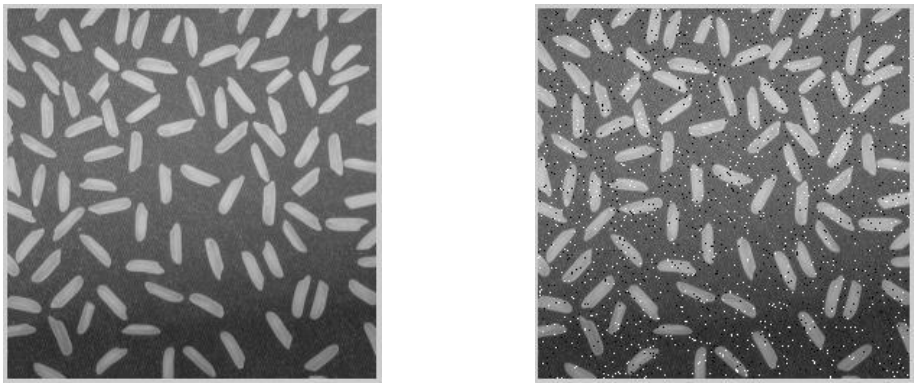
Ce chapitre fournit des détails sur le filtre non linéaire qui est le filtre médian, le bruit additif sel et poivre et fournit également des résultats de simulation du filtrage médian sous Matlab qui seront dûment discutés visuellement et sur la base du métrique PSNR.

I.2. Dégradation d'image par un bruit additif

Les principales sources de bruit dans les images numériques surviennent lors de l'acquisition, la numérisation et la transmission.

I.2.1. Bruit Sel et poivre

Le bruit poivre et sel est un type de bruit impulsionnel où une partie des pixels individuels de l'image numérique sont convertis aléatoirement en deux intensités extrêmes. Dans la majorité des cas, il s'agit des intensités maximale et minimale. Une erreur dans la capture et/ou l'enregistrement de l'image numérique représente la cause la plus fréquente de contamination par le bruit poivre et sel **[3]**.



(a) **(b)**

Figure I. 1.a) Image originale, **b)** Image dégradée par le bruit sel et poivre.

I.2.2. Densité de probabilité du bruit sel et poivre

La densité de probabilité du bruit sel et poivre est donnée comme suit :

$$P(z) = \begin{cases} P_a & \text{pour } z = a \\ P_b & \text{pour } z = b \\ 0 & \text{ailleurs} \end{cases} \quad \text{(I.1)}$$

Si $b > a$, une image contenant du bruit sel et poivre aura le niveau a qui apparait comme un pixel noir et le niveau b qui apparait comme un pixel blanc.

Le tracé de cette densité est montré dans la **Figure I. 2.**

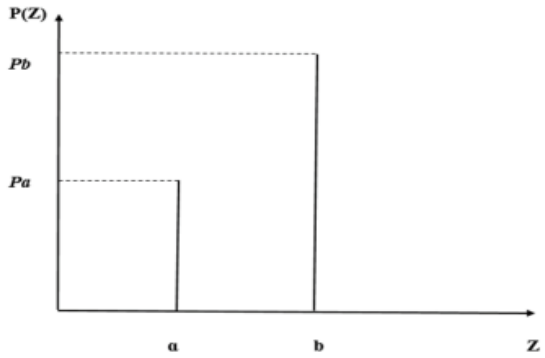


Figure I. 2. Densité de probabilité du bruit Sel et Poivre **[1].**

I.2.3. Modélisation de la dégradation

Si l'on suppose que la fonction de dégradation est le bruitage additif on peut écrire :

$$g(x, y) = f(x, y) + n(x, y) \quad \text{(I.2)}$$

Où $g(x, y)$ est l'image dégradée, $f(x, y)$ est l'image origine et $n(x, y)$ est le bruit additif. L'obtention de l'image dégradée est donnée par la figure suivante :

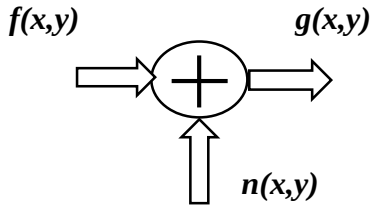


Figure I. 3. Modèle de la dégradation de l'image par un bruit additif.

I.3. Filtrage d'image dans le domaine spatial

Le filtrage dans le domaine spatial permet de réduire les écarts d'intensité dans chaque région de l'image tout en préservant la cohérence globale de l'image.

I.3.1. Traitement local et voisinage

Le filtrage spatial est un traitement local qui correspond à des opérations permettant de modifier chaque pixel de l'image en fonction des valeurs des pixels de son voisinage. L'étendue spatiale d'une image est déterminée par les caractéristiques locales de l'image où se trouve le point germe. En conséquence, une image est représentée comme une collection de régions homogènes plutôt qu'une collection prédéfinie de points ou de points voisins. De ce fait, le voisinage associé sert de fenêtre opérationnelle de transformation d'image à image pour chaque point à traiter **[4]**.

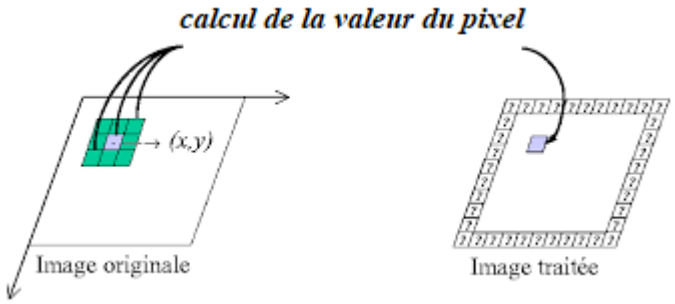


Figure I. 4. Filtrage local des images **[5]**.

En conséquence, les voisins locaux sont identifiés dans l'image à analyser comme des groupes de points connectés. Un voisinage de « 8 » : le pixel a huit pixels voisins : seuls les pixels NW, W, SW, NE, E, SE, N, S du pixel donné sont des voisins dont deux sont des voisins verticaux et deux des voisins horizontaux et quatre voisins de la diagonale et le pixel de la diagonale est aussi considéré **[4]**.

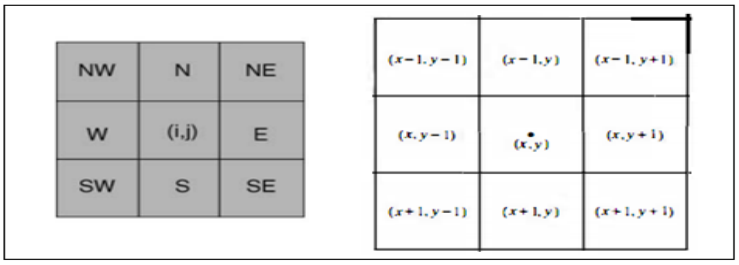


Figure I.5. Notion de voisinage**.**

I.3.2. Filtres non linéaires

Un filtre non linéaire est un opérateur qui remplace la valeur de chaque pixel par une combinaison non linéaire des valeurs des pixels de ses voisins. Le but des filtres non linéaires est de fournir une solution en employant une (ou plusieurs) des stratégies suivantes **[6]** :

- Statistiques d'ordre : Opérateurs basés sur une classification des valeurs de pixels voisins (minimum, maximum, médiane, etc.).
- Moyennes robustes : calcul des moyennes locales pondérées en excluant les valeurs marginales.
- Anisotropie : opérateurs où le support de calcul du filtre effectivement calculé varie localement en fonction du contenu de l'image.

Il existe plusieurs filtres non linéaires qui utilisent une ou plusieurs de ces propriétés. Principalement, nous expliqueront le principe du filtre médian que nous allons mettre en œuvre par la suite.

I.3.3. Filtre médian

On rappelle que le médian d'un ensemble de nombres $\{a_1, \ldots., a_n\}$ est la valeur a_k telle qu'il y ait autant de $a_i \leq a_k$ que de $a_i \geq a_k$. On le trouve facilement en triant les a_i par ordre croissant et en prenant la valeur du milieu **[7]**.

En débruitage d'image, le filtre médian est utilisé pour atténuer des pixels isolés, d'une valeur très différente de leur entourage. Le filtre médian est un filtre d'ordre. Un filtre d'ordre appliqué sur une fenêtre donnée de l'image, contenant la séquence $\{x(i)\}$ de N pixels ($N = 2p + 1$: impair).

I.3.3.1. Principe

- **Filtre médian 1D**

Étant donné un ensemble de valeurs numériques X = (X1, X2,..., XN), les statistiques d'ordre X(1), X(2) et X(N) sont des valeurs numériques définies en triant les valeurs de Xi dans ordre croissant. La valeur médiane est donc donnée par :

$$median(X)=\begin{cases} X_{(K+1)}=X_{(m)},\ for\ N=2K+1 \\ \frac{1}{2}(X_{(K)}+X_{(K+1)}),\ for\ N=2K \end{cases} \quad \text{(I.3)}$$

Où m = 2K + 1 est la plage médiane. La médiane est considérée comme un estimateur fiable du paramètre de localisation d'une distribution, et elle a de nombreuses applications en lissage et débruitage, en particulier pour les signaux contaminés par du bruit impulsif **[8].**

- **Filtre médian 2D**

Pour une image d'entrée en niveaux de gris avec des valeurs d'intensité$x_{i,j}$, le filtre médian bidimensionnel est défini comme :

$$y_{i,j}=median_{(r,s)\in W}(x_{i+r,j+s}) \quad \text{(I.4)}$$

W désigne une fenêtre à laquelle le filtre est appliqué. Pour la suite de ce mémoire, nous utiliserons des fenêtres carrées symétriques de taille M × M avec M = 2L + 1, c'est-à-dire que la plage médiane m est égale à $m=(M^2+1)/2$. C'est aussi la forme la plus couramment utilisée de ce filtre.

Le filtrage médian 2D consiste à balayer l'image par une fenêtre d'analyse (masque) de taille finie $(3\times3),(5\times5),...$ le calcul du nouveau niveau de gris du pixel central est remplacé par la valeur médiane de tous les pixels de la fenêtre d'analyse centrée sur le pixel.

La taille de la fenêtre d'analyse dépend de la fréquence du bruit et la taille des détails significatifs de l'image traitée.

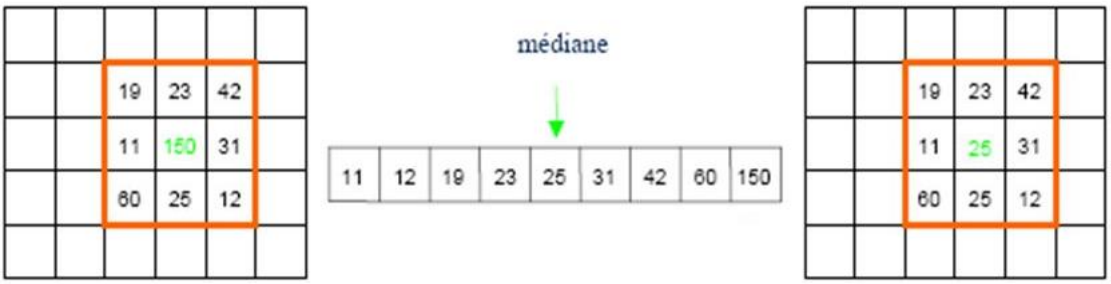


Figure I.6. Mise en œuvre du filtre Médian **[9]**.

En pseudo-code, l'algorithme s'écrit ainsi :

```
procédure tri_insertion(tableau T, entier n)
pour i de 1 à n - 1
x ← T[i]
j ← i
tant que j > 0 et T[j - 1] > x
T[j] ← T[j - 1]
j ← j - 1
Fin tant
T[j] ← x
fin pour
fin procédure
```

I.4. Evaluation de la qualité des images

Le MSE est l'erreur quadratique totale entre les images compressées et originales, tandis que le PSNR est une mesure de l'erreur maximale. Les formules mathématiques pour les deux sont les suivantes :

$$MSE = \frac{1}{M \times N} \sum_{m=1}^{M} \sum_{n=1}^{N} \left[f(m,n) - \hat{f}(m,n) \right]^2, \quad \text{(I.5)}$$

$$PSNR = 10 \log_{10} \frac{\left(peakval^2 \right)}{MSE} \quad \text{(I.6)}$$

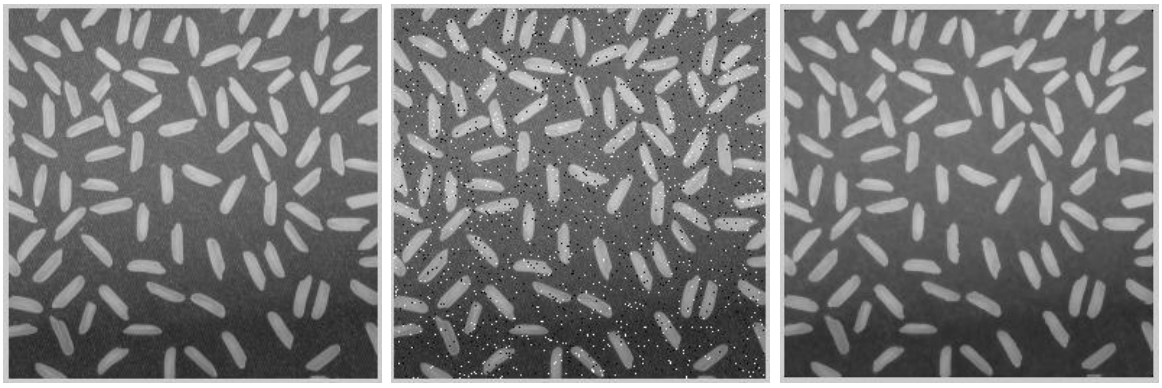
Où (M × N) est la taille de l'image, et f et $\hat{f}$ sont respectivement les images originale et filtrée. Ici, valeur de crête (Peak Value) est la valeur la plus élevée dans les données d'image. S'il s'agit d'un format de données 8 bits non signé, la valeur de crête est de 255. Nous pouvons voir qu'il s'agit d'une représentation de l'erreur absolue en db basée sur l'équation (I.6). Une valeur PSNR plus élevée est souhaitable car elle indique un rapport signal sur bruit plus élevé **[10]**.

I.5. Résultats de simulation sous Matlab

Nous avons appliqué un filtre médian (3x3) et (5x5) sur l'image (Rice.png) affectée du bruit sel et poivre avec deux différentes densités D=3% et D=25%.

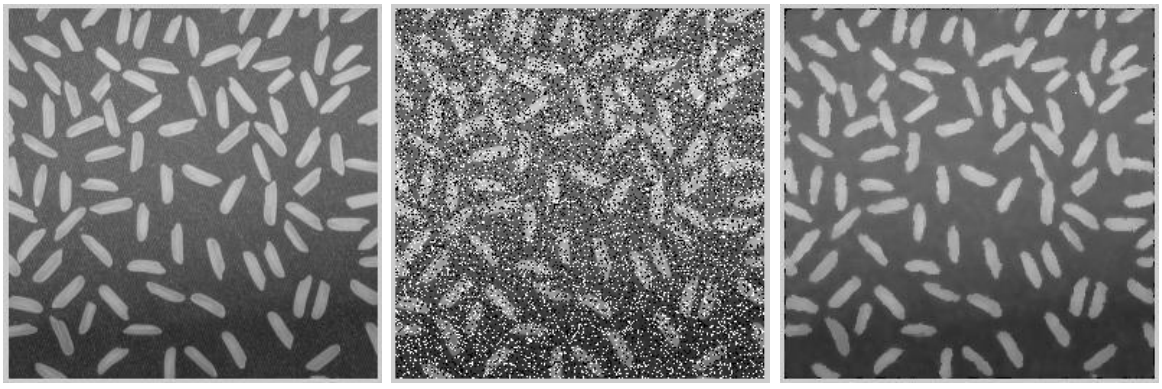
(a) **(b)** PSNR=11.5560 **(c)** PSNR=24.4368

Figure I. 7. (a) Image originale, **(b)** Image bruitée (sel et poivre D=25%), **(c)** Filtrage médian 3*3.



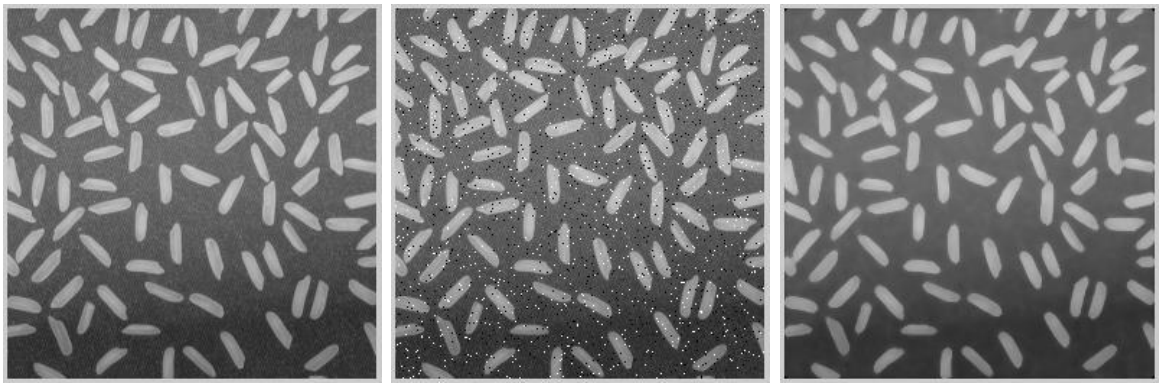
(a) **(b)** PSNR=20.6082 **(c)** PSNR= 31.8920

Figure I. 8. (a) Image originale, **(b)** Image bruitée (sel et poivre d=3%), **(c)** Filtrage médian 3*3.



(a) **(b)** PSNR=11.4937 **(c)** PSNR= 25.7570

Figure I. 9. (a) Image originale, **(b)** Image bruitée (sel et poivre d=25%), **(c)** Filtrage médian 5*5.



(a) **(b)** PSNR=20.7243 **(c)** PSNR=29.5900

Figure I. 10. (a) Image originale, **(b)** Image bruitée (sel et poivre d=3%), **(c)** Filtrage médian 5*5.

- **Interprétations**

Le tableau (I.1) regroupe les différentes valeurs du PSNR entre image originale et images bruitées et filtrées.

Tableau I.1. PSNR du filtrage médian**.**

Masque	**PSNR**			
	D=3%		**D=25%**	
	Image bruitée	Image filtrée	Image bruitée	Image filtrée
Masque (3 × 3)	20.6082	31.8920	11.5560	24.4368
Masque (5 × 5)	20.7243	29.5900	11.4937	25.7570

Les figures suivantes illustrent la variation du PSNR en fonction de la densité du bruit sel et poivre ainsi que la taille du masque utilisé.

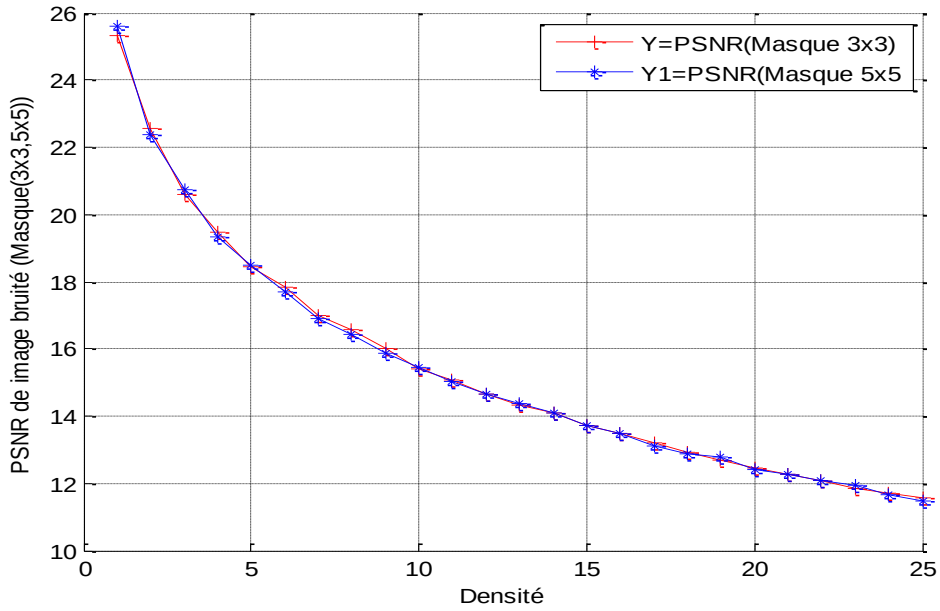


Figure I. 11. Variation du PSNR de l'image originale et l'image bruitée en fonction de la densité du bruit et la taille du masque.

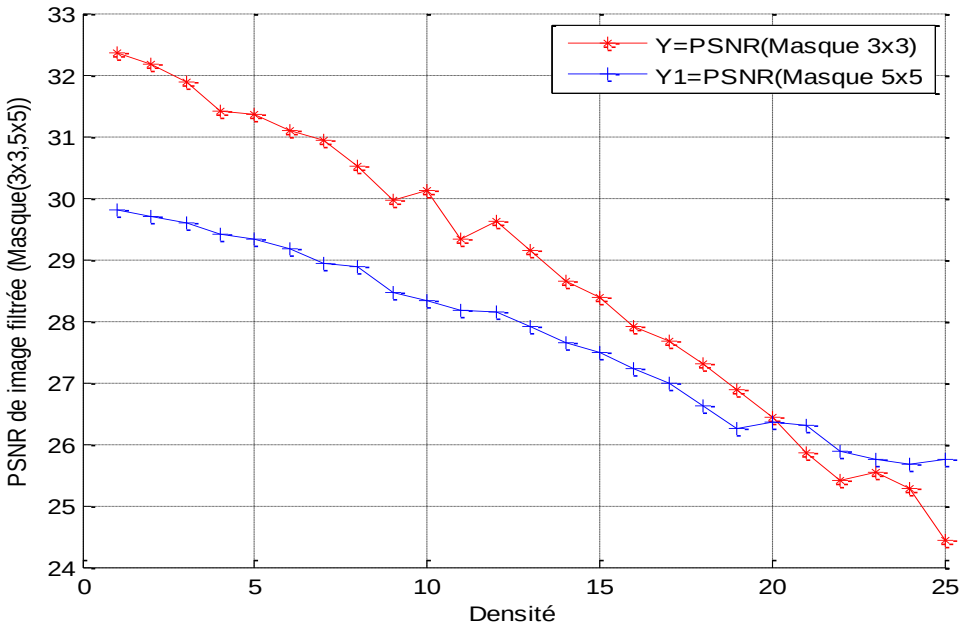


Figure I. 12. Variation du PSNR de l'image originale et l'image filtrée en fonction de la densité du bruit et la taille du masque.

1. Le filtre médian préserve le contraste de l'image (si toutes les valeurs sont multipliées par une constante positive, l'ordre des valeurs n'est pas affecté) et la luminosité de l'image (si une constante est ajoutée, l'ordre n'est pas affecté non plus).
2. Les performances de filtre médian dépendent principalement de la taille du masque utilisé et la densité du bruit impulsionnel.
3. Le filtre médian est bien adapté au filtrage du bruit impulsionnel. Le bruit sel et poivre est éliminé lorsque la densité du bruit est inférieure à 10% et est visiblement réduit dans le cas contraire.
4. On remarquera que le filtre médian de masque 3 × 3 est efficace tant que la densité du bruit est inférieur à 20% par contre le filtre médian de masque 5 × 5 permet d'éliminer le bruit lorsque la densité du bruit est élevée > 20% mais il peut éliminer certains détails qui seront préservés par celui de fenêtre 3 × 3.
5. Le filtre médian aide à réduire une variété de bruits. Les chercheurs ont découvert que ce filtre est plus efficace dans le cas d'un bruit "poivre et sel". De plus, ce filtre est bien connu pour sa préservation des contours, mais il affecte également les angles et les détails fins.

I.6. Conclusion

La nécessité de supprimer le bruit poivre et sel est impératif avant que les tâches de traitement d'image ultérieures telles que la détection des contours ou la segmentation ne soient effectuées. Dans ce chapitre nous avons présenté le principe filtre médian qui est d'une grande importante dans le traitement d'images ainsi que son implémentation sous Matlab. D'éventuelles remarques et interprétations ont été présentées.

Chapitre II

Débruitage des images par le filtre médian sous Simulink

II.1. Introduction

Le filtrage médian est considéré comme une méthode populaire pour supprimer le bruit impulsif des images. Cette technique non linéaire est une bonne alternative au filtrage linéaire car elle peut supprimer efficacement le bruit impulsionnel tout en préservant les informations de bord. Le filtre médian fonctionne pour chaque pixel de l'image avec des algorithmes de tri et assure qu'il correspond aux pixels qui l'entourent. Le tri est le cœur de l'algorithme du filtre médian. Dans l'algorithme de filtrage médian classique, toutes les valeurs de pixels doivent être triées pour que la valeur médiane puisse être obtenue. La qualité de l'algorithme de tri détermine l'efficacité du calcul de la valeur directement, et ainsi il détermine la performance globale du filtre.

Dans ce chapitre, on s'intéresse à la modélisation du filtrage médian sous Simulink et la génération du code VHDL avec l'outil HDL Coder pour pouvoir l'implémenter sur FPGA.

II.2. Environnement de travail

L'un des environnements que nous avons exploité dans notre travail est le logiciel MATLAB R2013a et R2015a qui peut être aussi considéré comme un langage de programmation adapté pour les problèmes scientifiques. Simulink, développé par Mathworks, est un environnement de modélisation fonctionnelle graphique pour la modélisation, la simulation et l'analyse de systèmes dynamiques multi domaines. Son interface est un outil de diagramme de bloc et un ensemble de bibliothèque des blocs. Il prend en charge la simulation, la vérification et la génération automatique de code **[11]**.

Le deuxième logiciel est Xilinx ISE Design Suite 14.7 est un outil logiciel produit par Xilinx pour la synthèse et l'analyse des conceptions HDL, permettant au développeur de synthétiser ("compiler") leurs conceptions, d'effectuer une analyse de synchronisation, d'examiner les diagrammes RTL, simuler la réaction d'un design à différents stimuli, et configurer le périphérique cible avec le programmeur **[12]**.

HDL coder est une fonctionnalité de Simulink qui permet au concepteur de créer un code HDL précis au bit et synthétisable à partir du modèle développé en utilisant les blocs Simulink. Le code HDL obtenu peut être synthétisé et mappé sur la carte FPGA cible en utilisant Xilinx ISE **[13]**.

II.3. Présentation de l'architecteur de filtre médian

Le processus du tri, qui nécessite plusieurs opérations de comparaison, est au cœur de la conception du filtre médian. Le nombre d'opérations de comparaison nécessaires pour la mise en œuvre du filtre médian est directement proportionnel au nombre d'échantillons testés, comme le montre l'équation(II. 1) **[1]**.

Ainsi, le processus de tri de N échantillons nécessite Z opérations de comparaison comme suit :

$$Z = (N-1) + (N-2) + (N-3) + \ldots + 1 \qquad \text{(II. 1)}$$

Le premier échantillon nécessite (N-1) opérations de comparaison pour être le plus petit, ou le plus grand, élément parmi les échantillons triés, puisqu'il doit être comparé à tous ses suiveurs auxquels ils sont égaux (N-1). Après cela, le nombre d'échantillons de repos est maintenant (N-1). Ainsi, le deuxième échantillon a besoin de (N-2) opérations de comparaison pour être dans l'ordre des éléments de tri, puisqu'il doit être comparé à tous ses suiveurs qui comptent (N-2). Ce processus se poursuit jusqu'à atteindre les derniers échantillons qui ne nécessitent aucune opération de comparaison.

Ainsi, le processus de tri de 9 échantillons nécessite 36 opérations de comparaison et le processus de tri de 25 échantillons nécessite 300 opérations de comparaison.

II.4. Implémentation du tri

L'opérateur Mm (pour Max/min) prend deux valeurs en entrée et renvoie les deux valeurs triées (la valeur de gauche est le maximum, la valeur de droite est le minimum) **[12] :**

$$Mm(A,B) = (Max(A,B), Min(A,B)) \qquad \text{(II. 2)}$$

L'opérateur Mm peut aussi être vu comme suit :

$$Mm(A,B) = \begin{cases} (B,A) & si \quad B > A \\ (A,B) & ailleurs \end{cases} \qquad \text{(II. 3)}$$

Chaque nœud de base indiqué sur la **figure II.2** représente un bloc de comparaison représenté sur la **figure II. 1.**

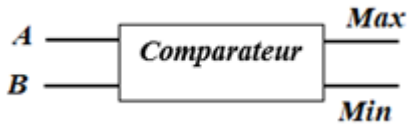


Figure II. 1. Représentation de bloc de comparateur.

II.5. Architecture matérielle du filtre médian à 36 comparateurs

Comme indiqué précédemment, le cœur de la conception du filtre médian est le processus de tri qui exige de nombreuses opérations de comparaison. La mise en œuvre du filtre médian classique est constituée de 36 comparateurs, comme le montre la figure suivante.

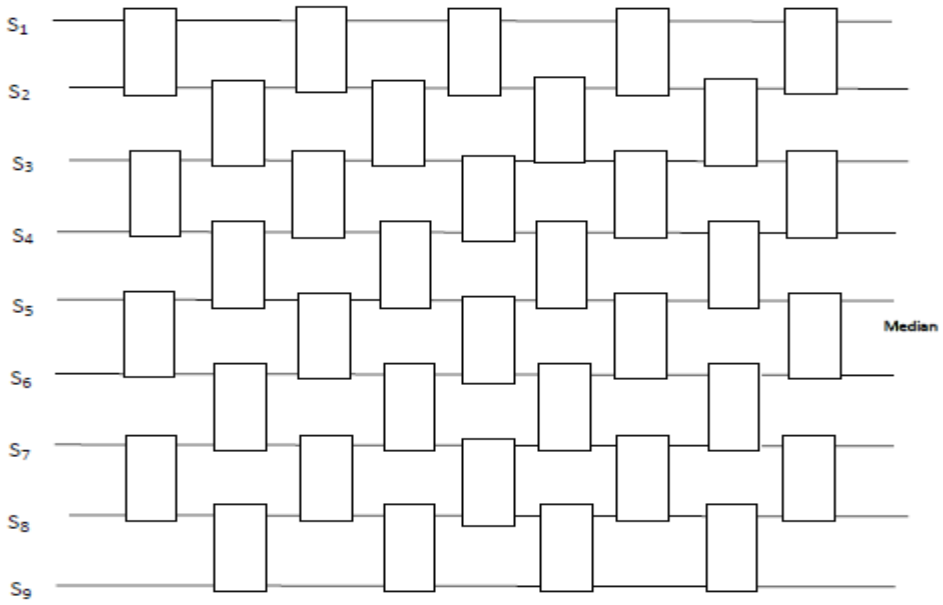


Figure II.2. Schéma bloc du filtre médian 3×3 avec 36 comparateurs.

II.6. Implémentation du filtre médian sous Simulink

II.6.1. Filtre médian 3x3

La figure suivante illustre l'architecture du bloc de filtrage médian 3x3 d'images développées sous Simulink.

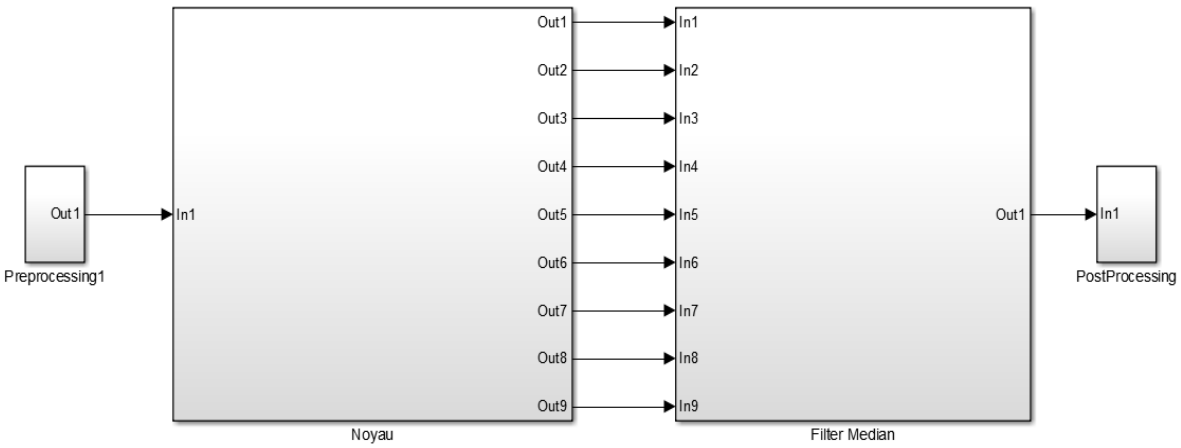


Figure II. 3. Modèle Simulink du filtrage médian 3x3.

II.6.2. Filtre médian 5x5

La figure suivante illustre l'architecture du bloc de filtrage médian 5x5 d'image développée sous Simulink.

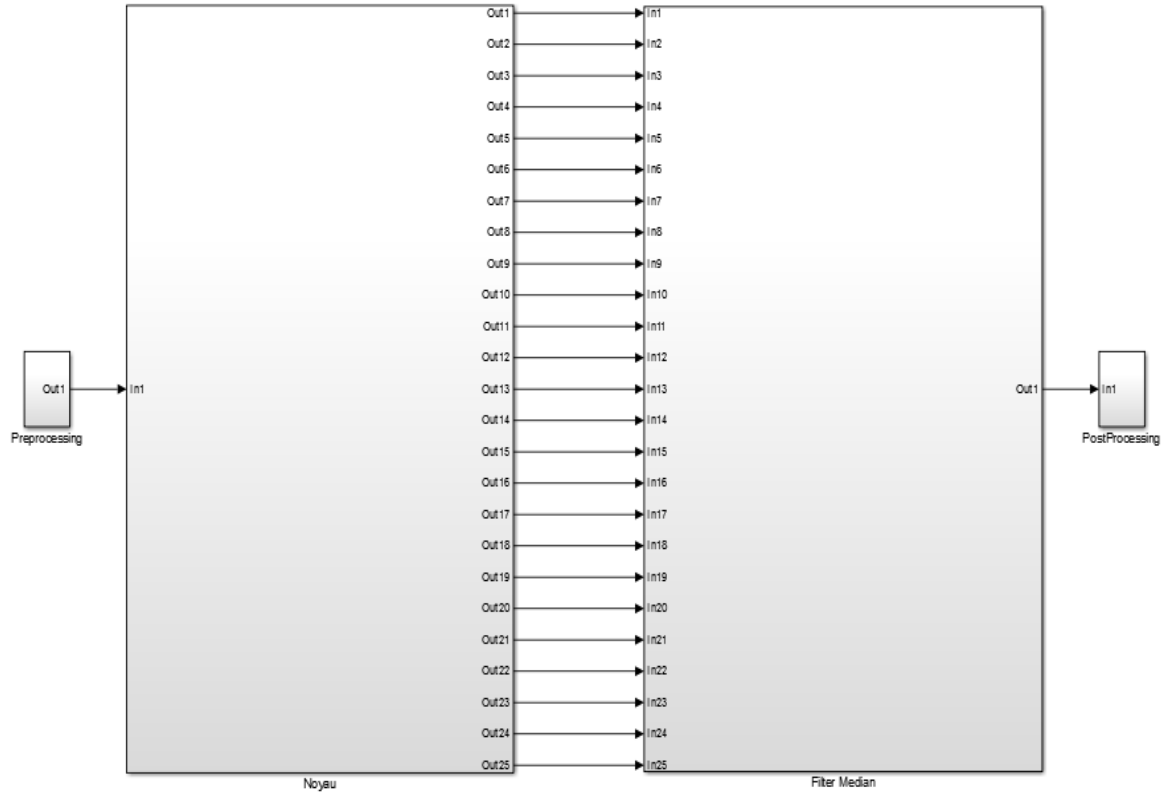


Figure II. 4. Modèle Simulink du filtrage médian 5x5.

II.6.3. Bloc de prétraitement (preprocessing)

Avant de traiter l'image, il faut l'appliquer au bloc de prétraitement qui convertit la matrice d'image en flux pixel par pixel spécifique qui convient à la compilation FPGA Bitstream à l'aide du générateur système **[12]**. La figure II.4 illustre la structure interne de ce bloc.

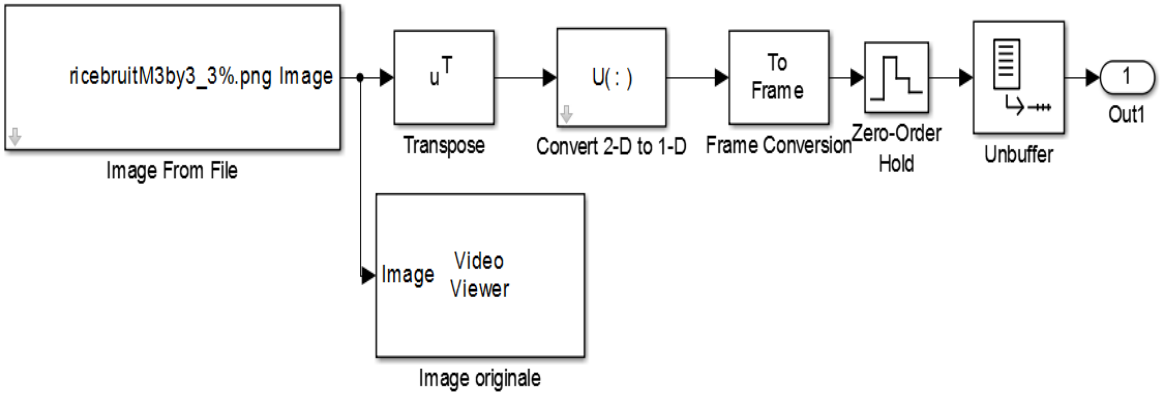


Figure II. 5. Bloc de prétraitement (preprocessing).

- Image from file : Le bloc 'Image From file'' permet d'importer l'image 'Rice.png' bruitée par le bruit sel et poivre de densités 3% et 25%.

- Transpose : définit la transposée de la matrice image.

- Convert 2-D to 1-D : Convertit l'image en un seul tableau de pixels.

- Frame Conversion et Unbuffer: permet de définir le mode d'échantillonnage et la mise en mémoire tampon des données.

II.6.4. Noyau du filtre médian 3x3

Line buffers permet d'obtenir les pixels de l'image bruitée qui participent à la formation de la fenêtre d'analyse. Deux Line buffers suffisent pour séparer trois lignes séquentielles de l'image d'entrée. La taille des Line buffers est sélectionnée selon la résolution de l'image d'entrée. A chaque itération, un noyau 3x3 de pixels est formé à partir du pixel d'entrée des registres à décalage et des sorties des tampons de ligne.

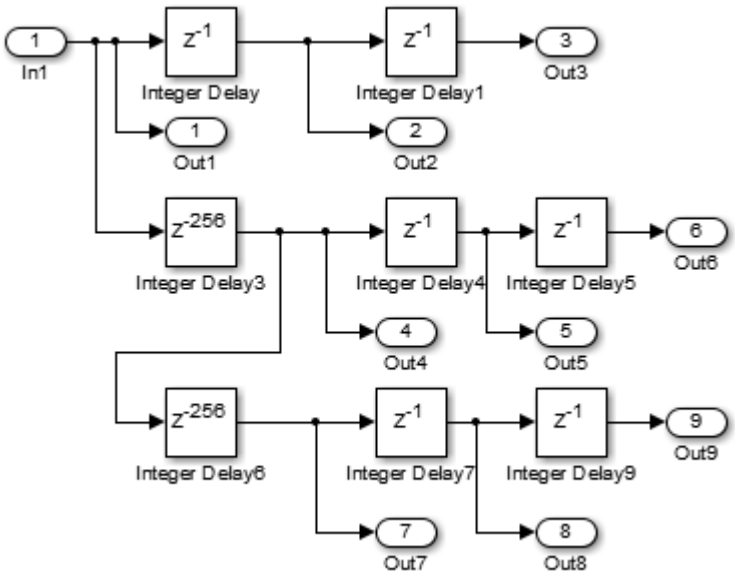


Figure II. 6. Noyau 3x3 du filtre médian.

II.6.5. Filtre médian 36 comparateurs

Plusieurs architectures sont implémentées et testées dans le présent travail pour le filtre médian 3x3. L'implémentation du filtre médian à 36 comparateurs est illustrée comme suit :

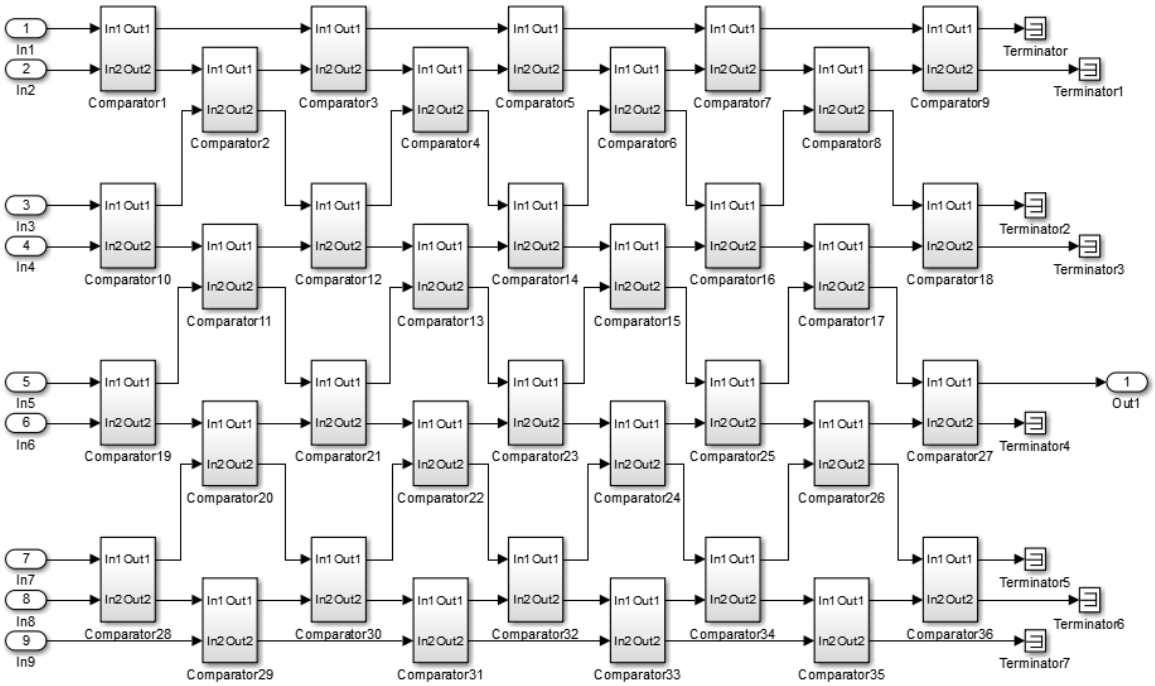


Figure II. 7. Modèle du filtre médian 3x3 avec 36 comparateurs.

II.6.6. Filtre médian 19 comparateurs

Dans la **figure II.8**, les groupes triangulaires à trois nœuds effectuent un tri complet sur trois éléments. Sur cette figure on observe que certains nœuds n'utilisent qu'une seule de leurs deux sorties. Cela permet d'éliminer dans ces nœuds (comparateurs), réduisant ainsi les ressources utilisées. Au total, 8 nœuds n'utilisent qu'une seule sortie, donc peuvent être économisés. L'architecture implémentée commence avec 36 nœuds (**figure II.7**), pour la minimisation du nombre utilisé de comparateurs nécessaires pour obtenir une architecture de 19 comparateurs (**figure II.8**), une étape de suppression de tous les comparateurs qui ne participent pas à l'obtention de la valeur médiane et aussi les composants avec des actions répétées.

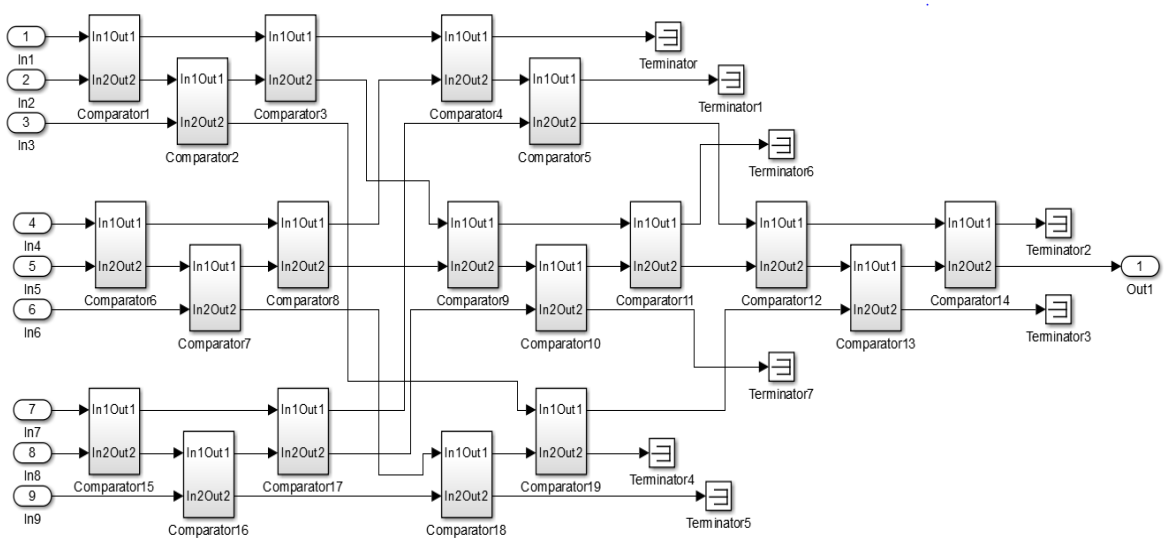


Figure II. 8. Modèle du filtre médian 3x3 avec 19 comparateurs.

II.6.7. Noyau du filtre médian 5x5

Les pixels de l'image bruitée qui participent à la formation de la fenêtre d'analyse sont obtenus par les line buffers. Quatre line buffers sont nécessaires pour séparer cinq lignes séquentielles de l'image d'entrée. La taille des line buffer est choisie en fonction de la résolution de l'image d'entrée. À chaque itération un noyau 5x5 de pixels sont formés à partir du pixel d'entrée des registres à décalage et les sorties des line buffer.

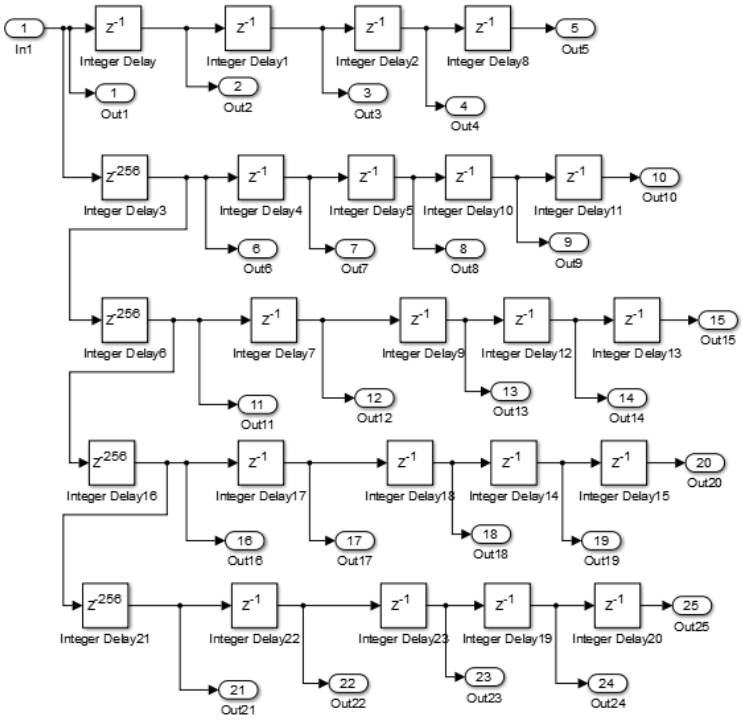


Figure II. 9. Noyau 5x5 du filtre médian.

II.6.8. Filtre médian 300 comparateurs

L'implémentation du filtre médian à 300 comparateurs est illustrée comme suit :

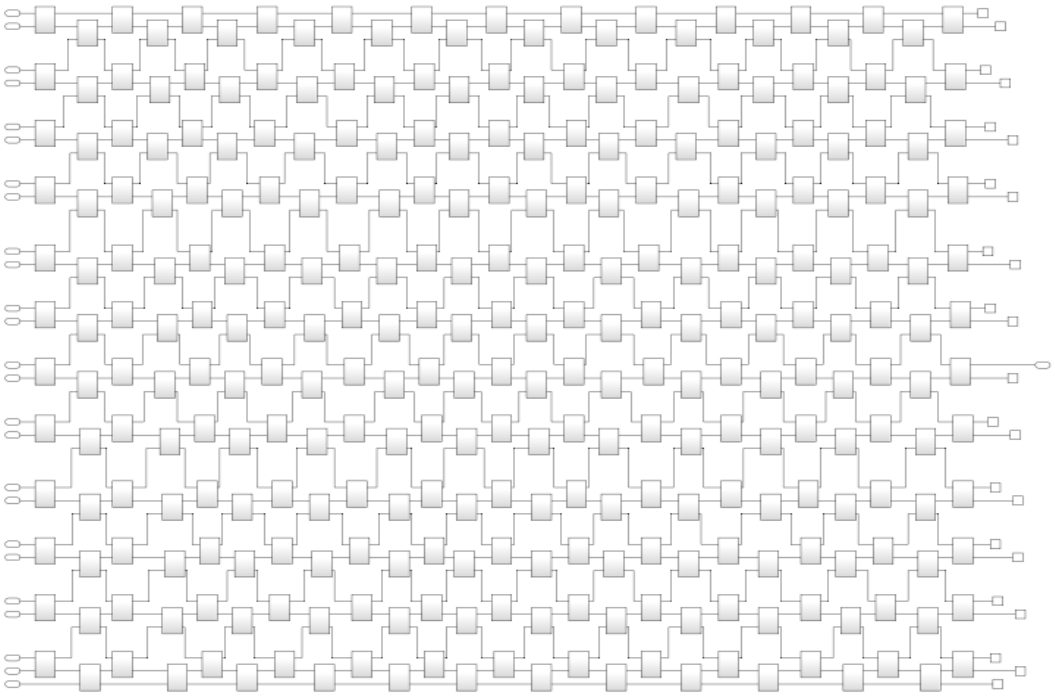


Figure II. 10. Modèle du filtre médian 5x5 à base des 300 comparateurs.

II.6.9. Filtre médian 234 comparateurs

Après l'optimisation faite sur le filtre médian 5x5 de base, nous avons obtenu 234 comparaison nécessaire pour la réalisation du filtre 5x5 comme indique sur la **figure II.11**, dans lequel nous avons adopté la même méthode d'optimisation que utilisée dans le filtre médian 3x3 qui consistait à annulent tous les comparateurs sans importance et aussi les composants avec des actions répétées.

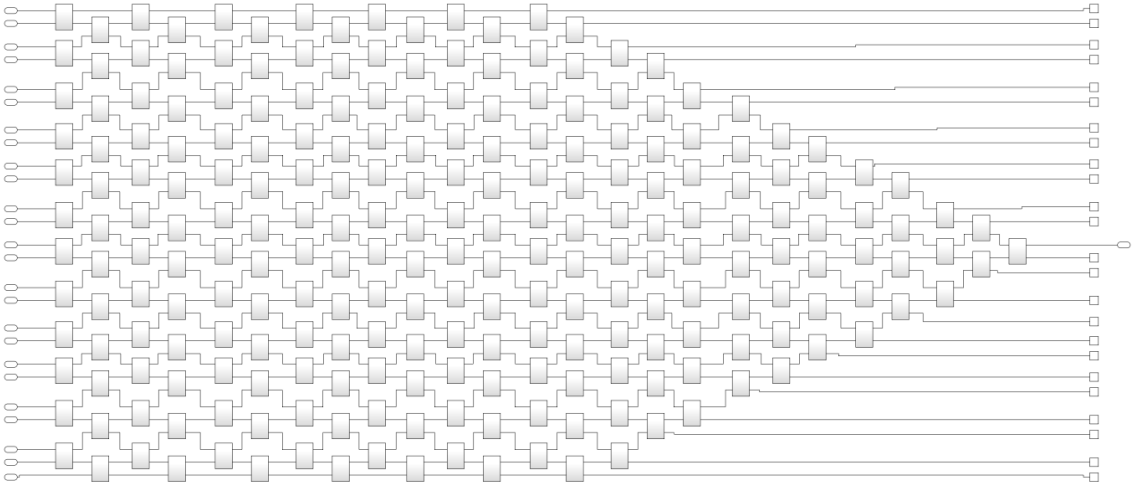


Figure II. 11. Modèle du filtre médian 5x5 à base de 234 comparateurs.

II.6.10. Bloc comparateur

Le filtre médian 3x3 ou 5x5 est basé sur des comparateurs, qui sont utilisées pour trier les pixels sélectionnés par le masque du filtre dans un ordre croissant ou décroissant. Le bloc de comparaison de base sous Simulink est conçu par l'emploi de deux opérateurs relationnels et deux commutateurs. Avec une telle conception, nous obtenons les deux valeurs triées du plus grand au plus petit.

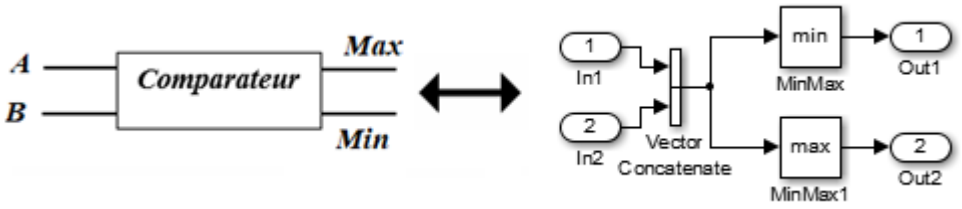


Figure II. 12. Bloc comparateur sous Simulink.

II.6.11. Bloc post-traitement (post-processing)

Après la phase de filtrage, il y a un autre bloc de post-traitement qui fait précisément l'inverse de ce qu'a fait le bloc de pré-traitement, Le calcul du PSNR entre l'image originale et l'image filtrée est assuré par le bloc PSNR.

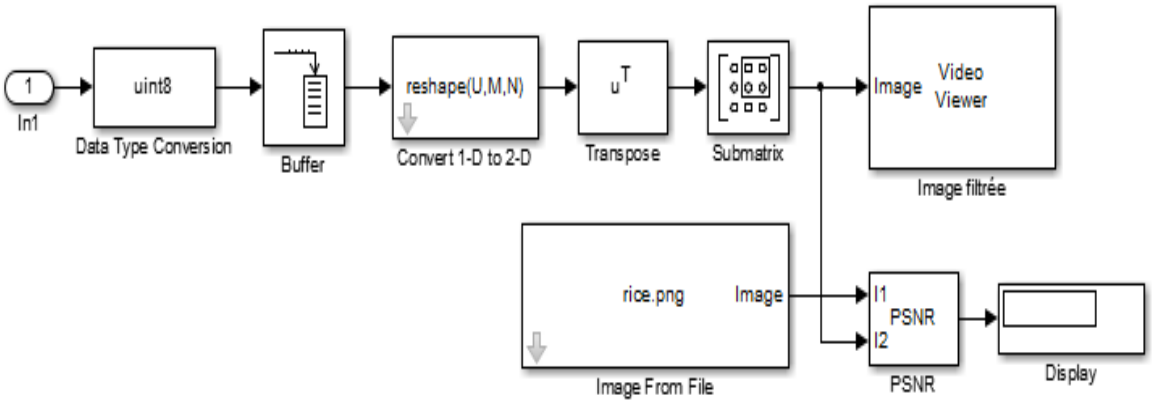
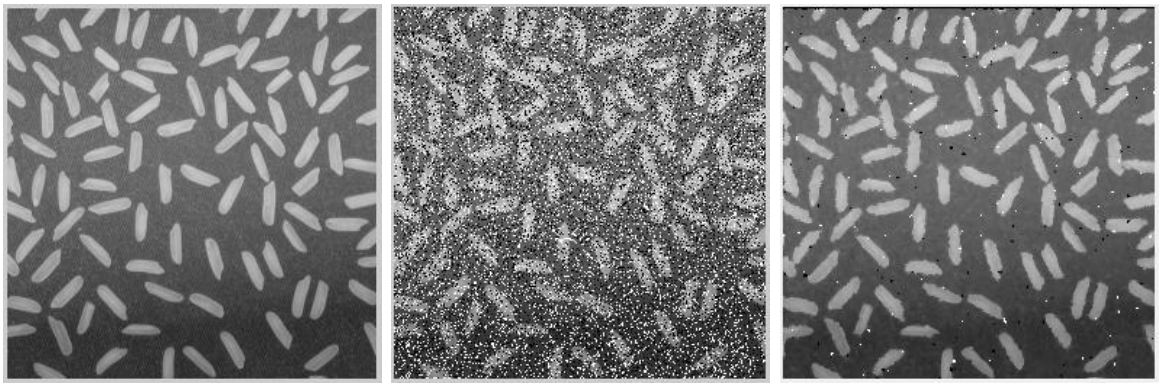


Figure II. 13. Bloc de post-traitement (postprocessing).

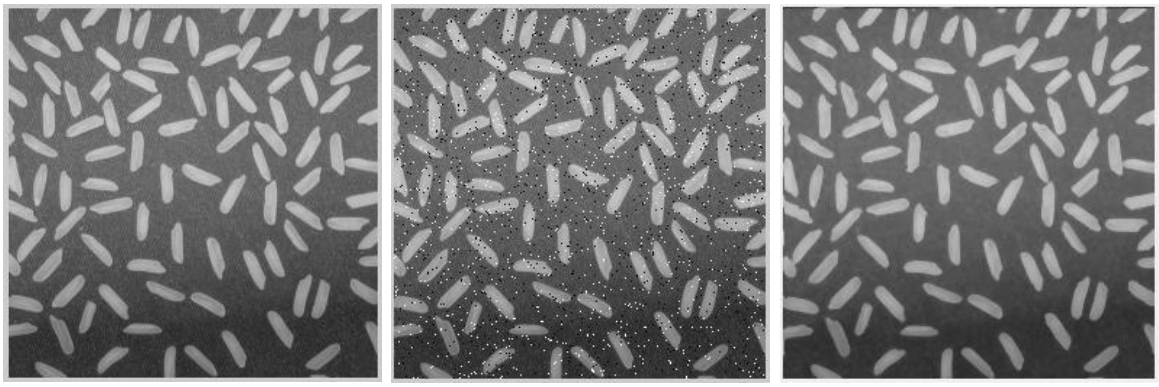
II.7. Résultats simulation sous Simulink

Nous avons appliqué le filtre médian (3x3) et (5x5) sur l'image (Rice.png) affectée du bruit sel et poivre avec deux différentes densités D=3% et D=25%.



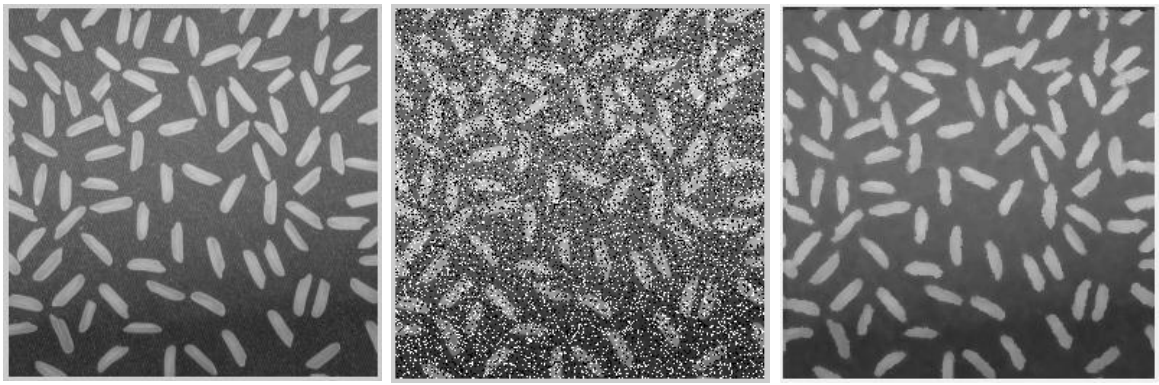
(a) **(b)** **(c)** PSNR=21.58

Figure II. 14. (a) Image originale, **(b)** Image bruitée (sel et poivre D=25%), **(c)** Filtrage médian 3x3.



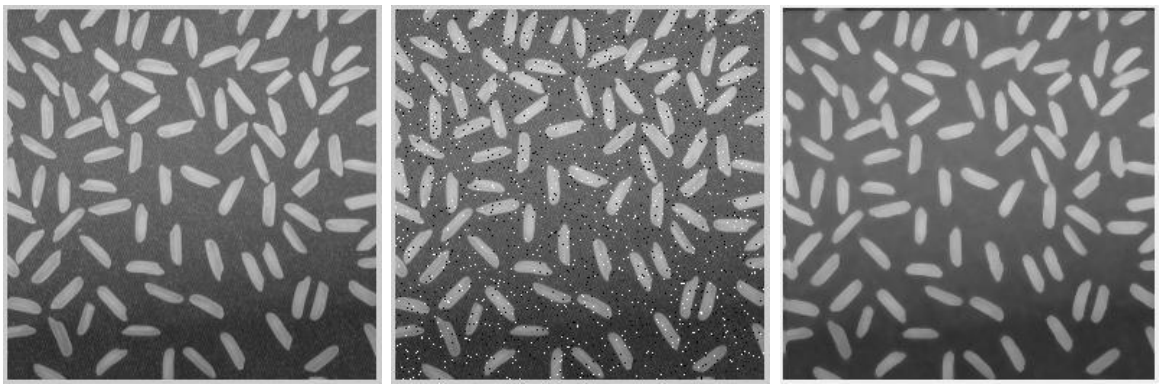
(a) **(b)** **(c)** PSNR= 22.76

Figure II. 15. (a) Image originale, **(b)** Image bruitée (sel et poivre D=3%), **(c)** Filtrage médian 3x3.



(a) **(b)** **(c)** PSNR= 19.21

Figure II. 16. (a) Image originale, **(b)** Image bruitée (sel et poivre d=25%), **(c)** Filtrage médian 5x5.



(a) **(b)** **(c)** PSNR=19.26

Figure II. 17. (a) Image originale, **(b)** Image bruitée (sel et poivre d=3%), **(c)** Filtrage médian 5x5.

II.8. Interprétations

Le tableau ci-dessous montre la comparaison basée sur le PSNR pour l'évaluation de filtre médian 3x3 et 5x5 sur des images bruitées avec deux densités 3% et 25%.

Tableau II.1 : PSNR du filtrage médian**.**

Masque	PSNR	
	D=3%	D=25%
	Image filtrée	Image filtrée
Masque (3 × 3)	22.76	21.58
Masque (5 × 5)	19.26	19.21

On voit bien une bonne performance du filtre médian pour le bruit sel et poivre.

Ce filtre élimine ce type de bruit tout en préservant les détails de l'image pour les deux type de masque 3x3 et 5x5, dans notre cas, pour une densité $D = 3\%$, le PSNR obtenu pour le filtre médian 3x3 est 22,76 et celui du filtre médian 5x5 est 19.26.

Pour des images fortement bruitées, dans notre cas $D = 25\%$, le filtre 3x3 diminue le bruit impulsionnel sans l'éliminer avec un PSNR=21,58 et celui de 5x5 élimine le bruit mais diminue la qualité de l'image (PSNR de 19,21).

II.9. Génération du code VHDL à l'aide de HDL Coder

II.9.1. Configuration de Simulink

On se concentre essentiellement sur deux configurations principales : Le temps de simulation et le type de données pour configurer Simulink.

Pour synchroniser le temps d'échantillonnage avec la période d'horloge du FPGA, il convient d'effectuer les opérations suivantes :

- ✓ Le temps d'échantillonnage doit être un nombre entier.
- ✓ Choix du solveur discret.
- ✓ Le temps d'arrêt est le nombre de pixels de l'image

La figure suivante présente la fenêtre de configuration avec le temps de simulation

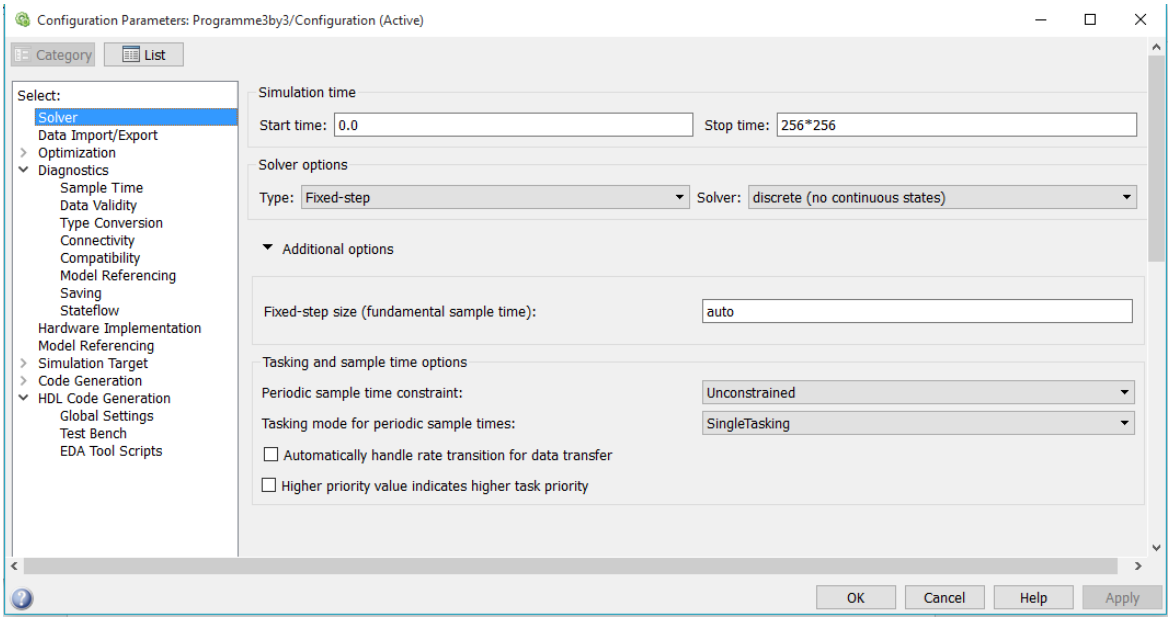


Figure II. 18. Fenêtre de configuration du temps de simulation.

Ces configurations concernant le type de données doivent être effectuées :

- ✓ Erreur dans overflow and downcast.
- ✓ Use local flow for read before write and write and write after read.
- ✓ Erreur dans multitask data store.

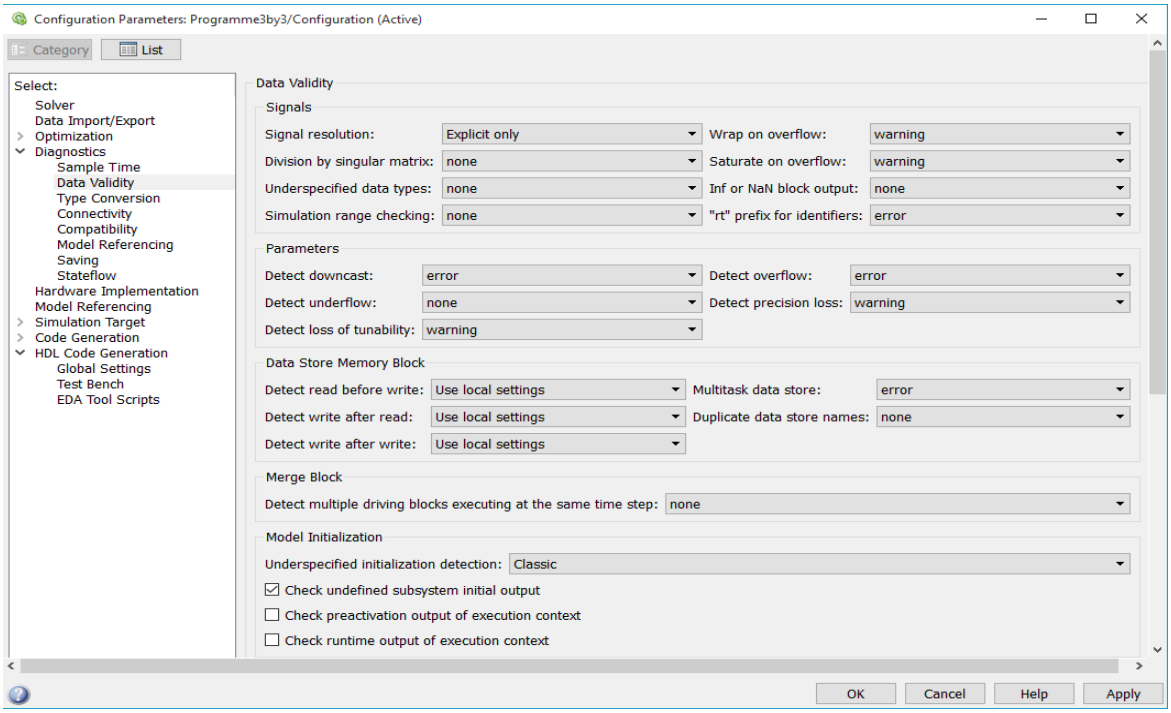


Figure II. 19. Fenêtre de configuration du type de données des signaux.

II.9.2. Génération du code VHDL

Pour générer du code VHDL pour le modèle de Simulink :

- ✓ Sélectionnez le sous-système de l'objet sous test du modèle et assurez-vous que le nom de ce sous-système apparaît dans l'option generate HDL for.
- ✓ pour Language, selectionnez VHDL.
- ✓ Par défaut, HDL Coder génère du code VHDL dans le dossier cible hdlsrc.

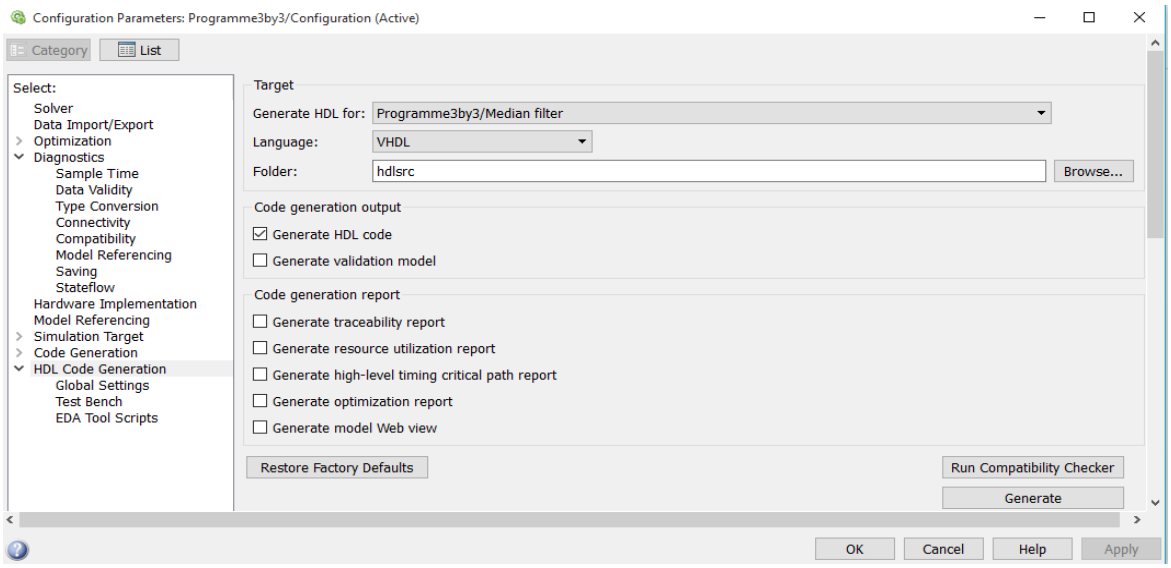


Figure II. 20. Génération du code HDL à partir du modèle Simulink.

HDL Coder compile le modèle avant de générer le code.

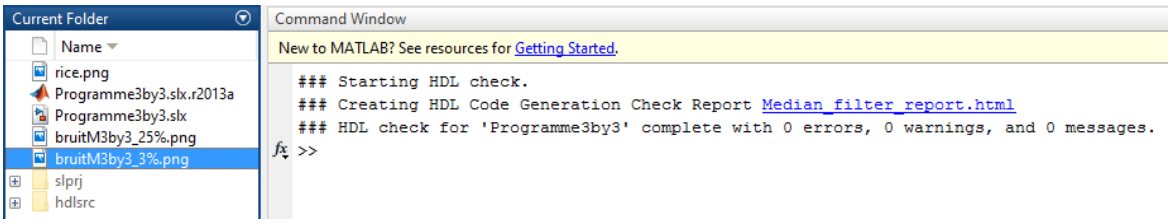


Figure II. 21. Compilation du modèle par HDL Coder.

Création du rapport de contrôle de génération de code HDL "Median_filtre_report.html". La compilation de notre modèle s'est terminée avec 0 warnings et 0 errors.

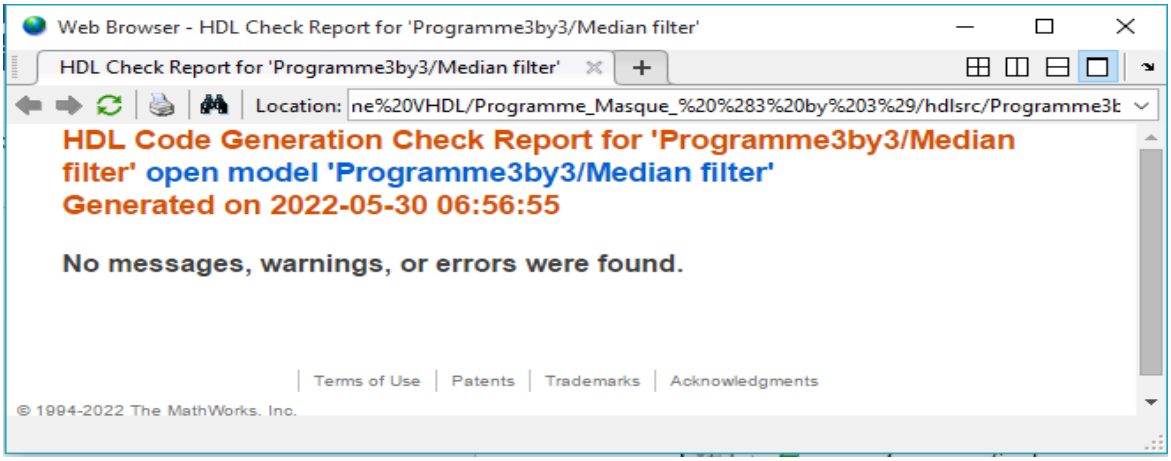


Figure II. 22. Création du rapport de contrôle de génération de code HDL.

Au fur et à mesure de la génération du code, HDL Coder affiche des messages de progression dans la ligne de commande MATLAB avec des liens vers les fichiers générés.

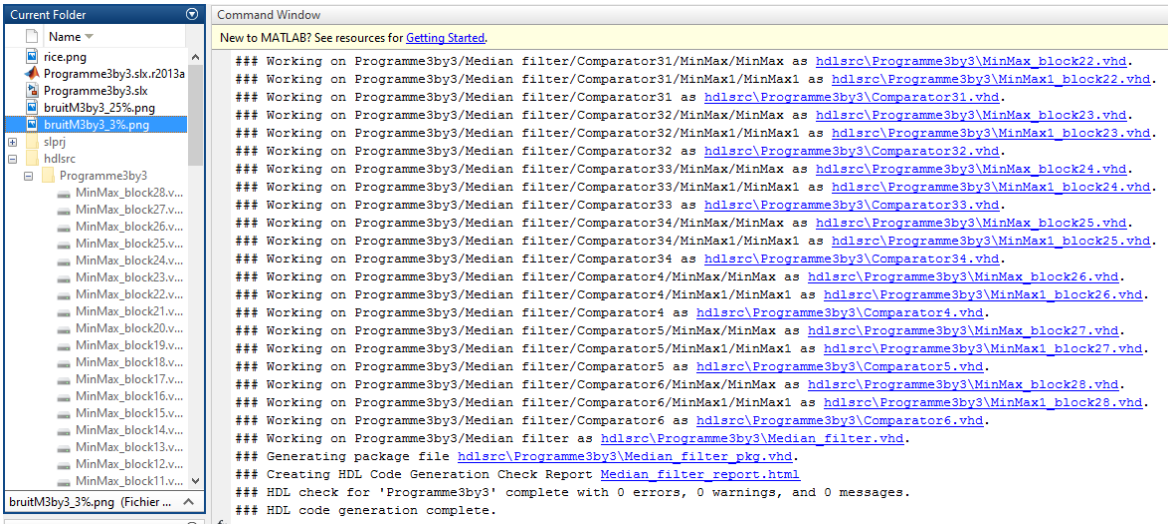


Figure II. 23. Génération du code VHDL.

Le processus est terminé et affiche le message : ### HDL code generation complete.

II.10. Génération du fichier de programmation dans ISE Design Suite

Pour générer le fichier de programmation du code VHDL crée par HDL Coder et son implémentation sur FPGA XC3S500E Xilinx Spartan-3E on procède comme suit :

- ✓ Création d'un nouveau projet sur ISE Design Suite

Le nouveau projet nous invite à définir les les caractéristiques du FPGA utilisé pour l'implémentation.

- Family : Spartan3E.
- Device : XC3S500E.
- Package : FT256.
- Speed Grade : -4.
- Synthesis Tool : XST (VHDL/Verilog).
- Simulator : ISim (VHDL/Verilog).

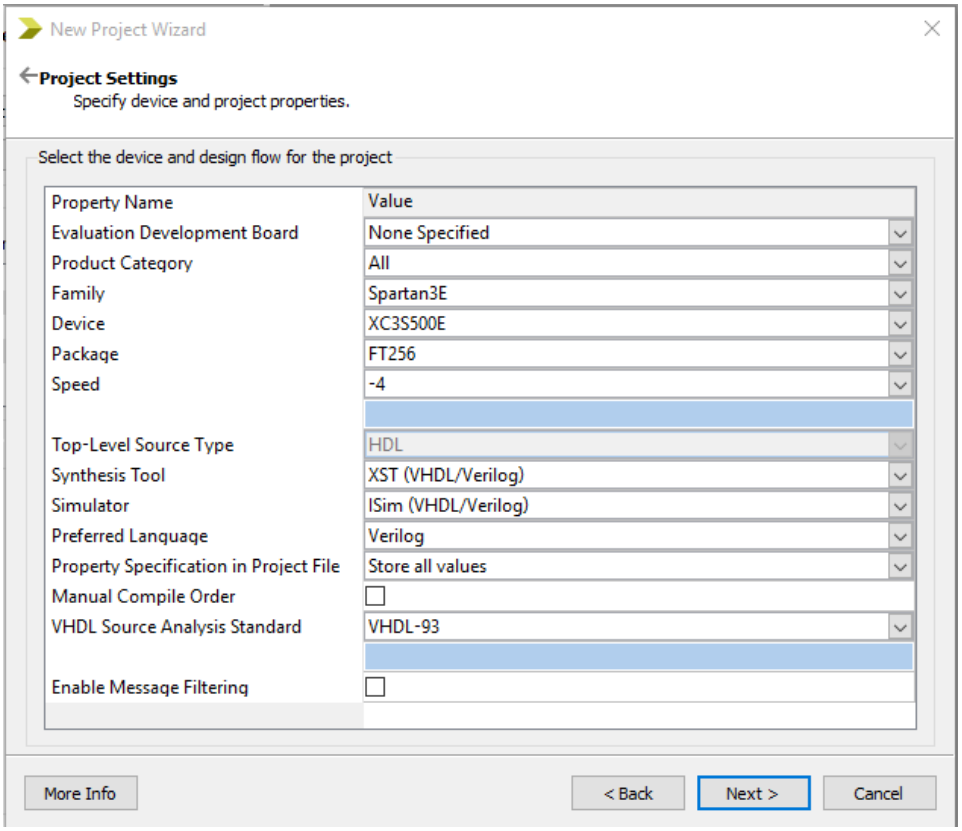


Figure II. 24. Boîte de dialogue New Project Wizard.

✓ Ajouter les fichiers sources VHDL crées par HDL Coder. Cela permettra de voir l'état de tous les fichiers sources qui ont été ajoutés au projet. Il permet également de spécifier l'association de la vue de conception, ainsi que la bibliothèque pour les sources VHDL qui ont été ajoutées avec succès au projet.

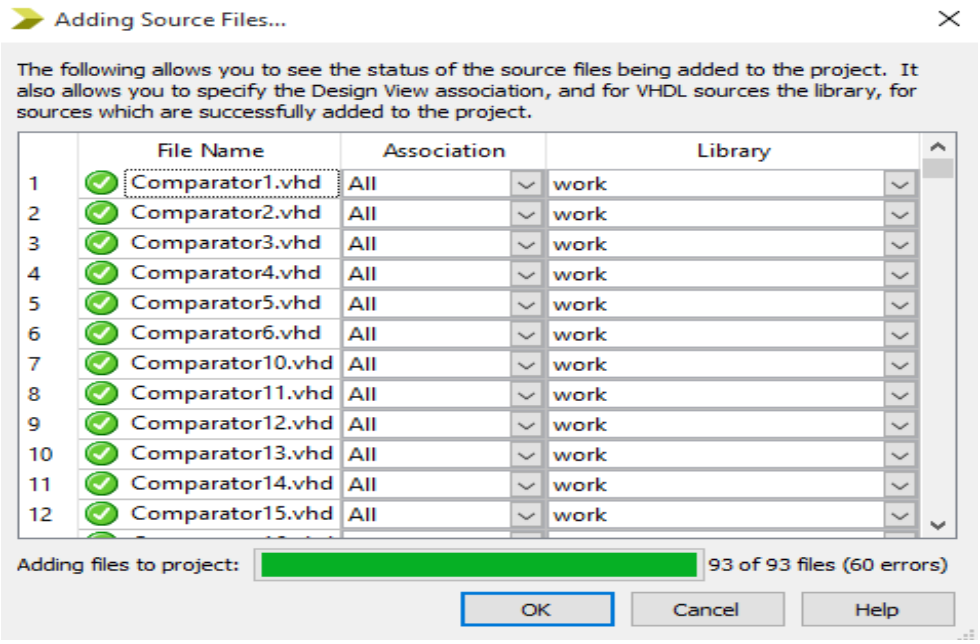


Figure II. 25. Ajout de fichiers source au projet.

L'implémentation de la conception est le processus de traduction, de mappage, de placement, de routage et de génération de fichiers bitstream pour notre conception qui est le fichier de configuration nommé (Median_filtre.bit).

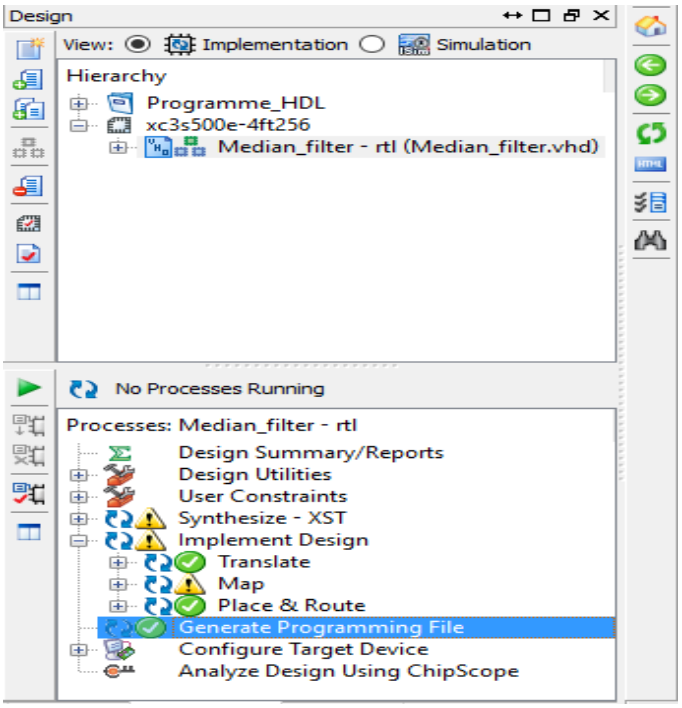


Figure II. 26. Génération du fichier de configuration du modèle généré par HDL Coder.

II.11. Conclusion

Dans ce chapitre nous avons présenté en détail l'implémentation du filtre médian 3x3 et 5x5 ainsi que leur modèles optimisés sous Simulink, donc une attention particulière a été accordée à sa conception et sa mise en œuvre. Les résultats illustrent l'efficacité des architectures de base et optimisés du filtre médian sur la suppression du bruit impulsionnel. L'outil HDL Coder a été utilisé pour générer le code VHDL à partir du modèle Simulink développé pour sur la cible matérielle FPGA.

Chapitre III

Débruitage des images par le filtre médian sous Xilinx System Generator (XSG)

III.1. Introduction

Xilinx System Generator propose un environnement de développement intégré (IDE) pour FPGA sous l'outil Matlab. Cet IDE vise à augmenter le niveau d'abstraction de la conception matérielle et à minimiser l'intervention manuelle de la génération de code HDL. C'est un outil de conception de haut niveau qui permet d'utiliser l'environnement MathWorks Simulink dans la conception de circuits numériques dédiés aux FPGA Xilinx. Il est utilisé pour la génération, la simulation et la validation de systèmes matériels tout au long de la technique de Co-simulation matérielle **[14]**.

Dans ce chapitre, on s'intéresse à la modélisation du filtrage médian à comparateurs avec l'outil XSG (Xilinx System Generator), incorporation du hardware dans le design de Simulink (hardware software Co-Simulation) et simulation avec hardware in the Loop du FPGA Spartan3e (xc3s500e).

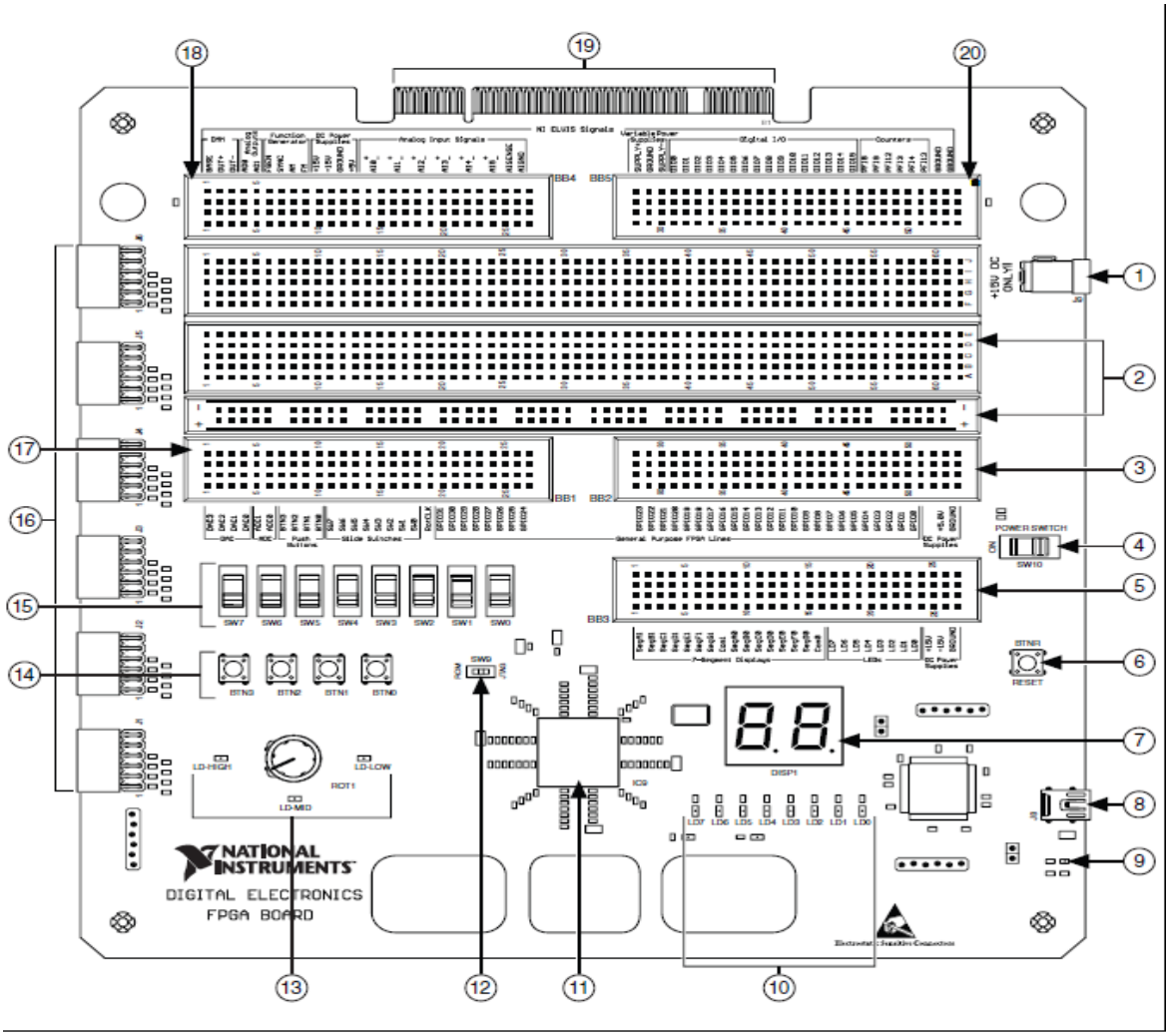
III.2. Environnement de travail

Nous avons notamment exploité la bibliothèque Xilinx dans l'environnement MATLAB/Simulink qui propose une gamme de blocs destinés à modéliser des opérations logiques ou mathématiques pour le traitement du signal et de l'image.

Xilinx System Generator « XSG », qui fait partie de la librairie Xilinx est un outil de conception DSP pour relier ISE et Matlab et permet l'utilisation de MathWorks Model-Based Simulink pour la conception FPGA. Xilinx System Generator simplifie la conception du matériel FPGA, a été le premier à lancer l'idée de compiler un programme FPGA à partir de MATLAB et du modèle Simulink. Il permet la modélisation du système et la génération automatique de code à partir de MATLAB et Simulink. Lors de la conception du modèle Simulink, les blocs générateurs du système peuvent être utilisés comme les autres blocs Simulink. En outre, le concepteur peut également utiliser les deux blocs Simulink et les blocs Xilinx System Generator en même temps. Cet outil permet également la Co-simulation matérielle utilisée pour tester les cœurs créés par l'utilisateur sur le matériel cible simultanément avec le modèle présent dans l'environnement Simulink **[15]**.

III.3. Plateforme matérielle NI Digital Electronics FPGA Board

Ce projet utilise comme plate-forme matérielle la carte FPGA de NI Digital Electronics, une plate-forme de développement de circuits basée sur le FPGA XC3S500E Xilinx Spartan-3E. La figure suivante illustre le schéma de référence de la vue supérieure de la carte FPGA de NI Digital Electronics.



1 Power Connector	7 Seven-Segment Displays	14 Push Buttons
2 General-Purpose Breadboard Banks	8 USB Connector	15 Slide Switches
	9 LD-G LED	16 Digilent Pmod Connectors
3 Signal Breadboard Bank BB2	10 LEDs	17 Signal Breadboard Bank BB1
4 Power Switch	11 FPGA	18 Signal Breadboard Bank BB4
5 Signal Breadboard Bank BB3	12 Switch SW9	19 NI ELVIS Connector
6 Reset Button	13 Rotary Push-Button Knob/LEDs	20 Signal Breadboard Bank BB5

Figure III. 1. Plateforme matérielle NI Digital Electronics FPGA.

III.4. FPGA XC3S500E Xilinx Spartan-3E

Le FPGA XC3S500E Xilinx Spartan-3E est un circuit intégré de type PLCC (Plastic Leaded Chip Carrier) dont le marquage du boitier est comme suit :

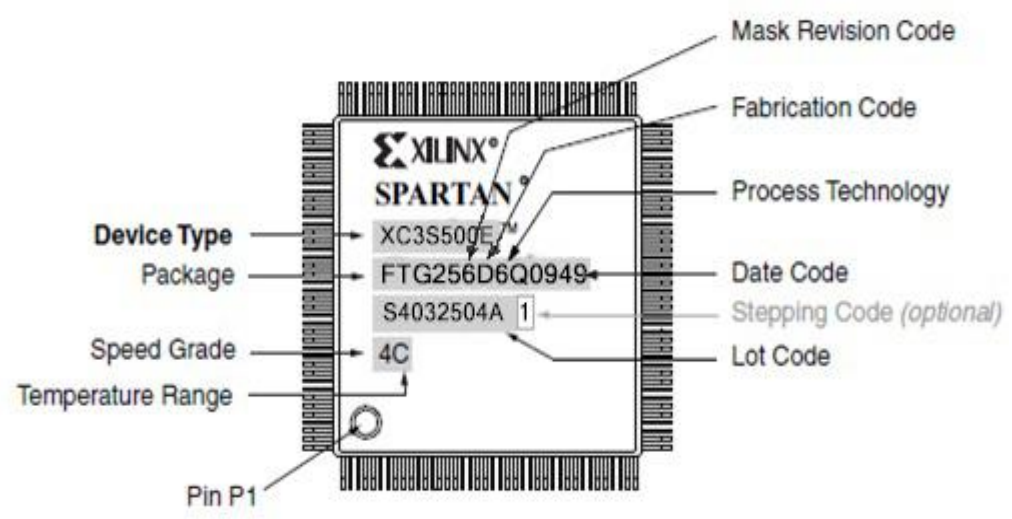


Figure III. 2. Marquage du circuit FPGA XC3S500E.

Les informations du marquage sont :

- ✓ Device Type : XC3S500E.
- ✓ Speed Grade : -4 (Standard Performance).
- ✓ Package Type / Number of Pins : FT256 :256-ball Fine-Pitch Thin Ball Grid Array (FTBGA).
- ✓ Temperature Range (TJ) : C (Commercial (0°C to 85°C)).

III.5. Filtrage médian sous XSG

La bibliothèque XSG comprend une série de blocs fonctionnels essentiels, généralement utilisés dans la programmation de cibles FPGA. La figure suivante illustre l'architecture du bloc de filtrage médian à 19 comparateurs développé sous XSG.

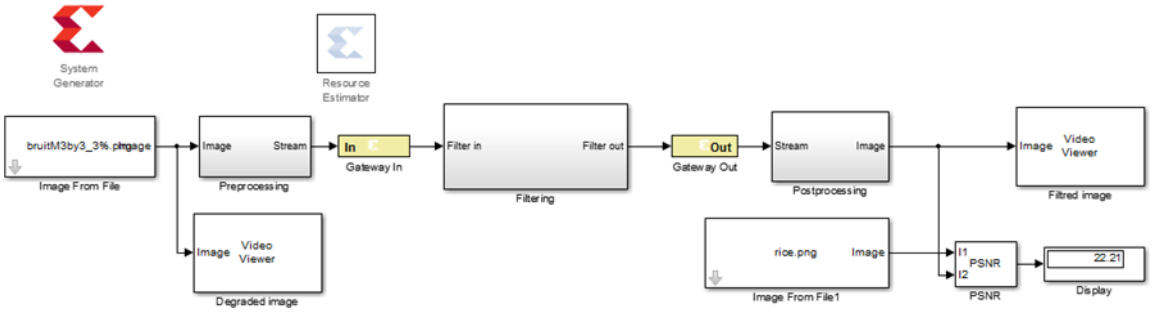


Figure III. 3. Modèle Simulink de conception de filtre médian.

III.5.1. Bloc "System Generator"

Le bloc du générateur de système permet de contrôler les paramètres du système et de simulation, et est utilisé pour exécuter le générateur de code. Chaque modèle Simulink contenant un élément du Blockset Xilinx doit contenir au moins un bloc System Generator. Une fois qu'un bloc System Generator est ajouté à un modèle, il est possible de spécifier comment la génération de code et la simulation doivent être traitées. L'éditeur de propriétés de ce bloc permet de spécifier les périphériques, les objectifs de performance et la période du système **[16]**. Le symbole et la boîte de dialogue du bloc "System Generator" sont illustrés dans la figure suivante :

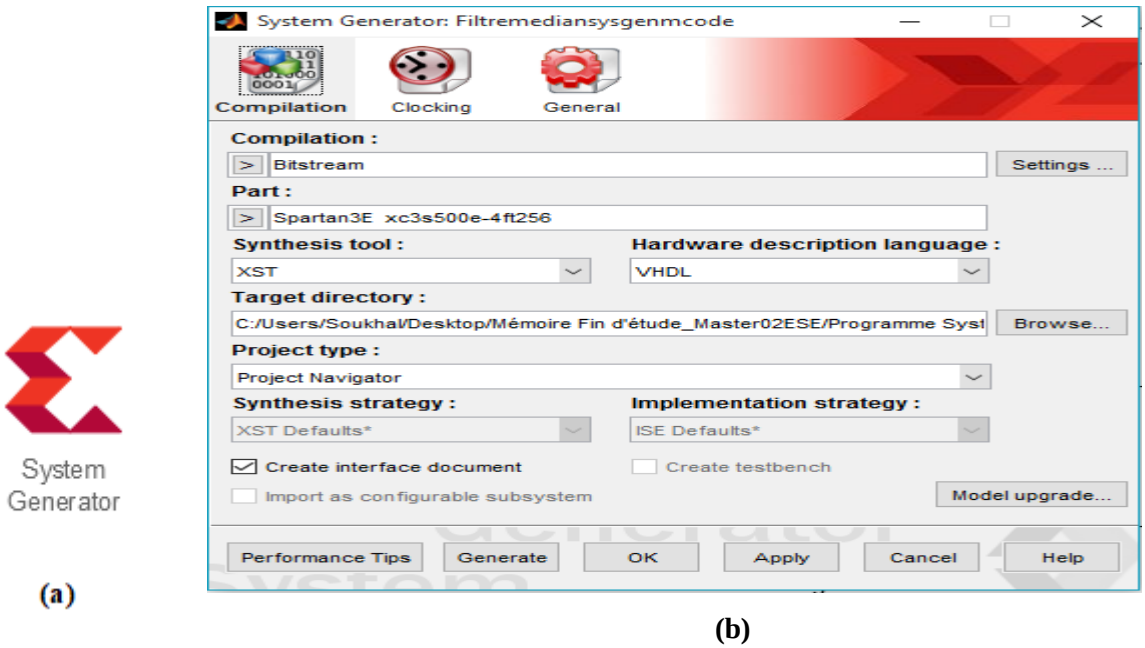


Figure III. 4. (a) Bloc "System Generator" **(b)** Boîte de dialogue du bloc "System Generator".

III.5.2. Blocs Gateway IN & Gateway OUT

Tous les blocs Xilinx doivent être connectés entre les blocs « Gateway In » et « Gateway Out ». Tous les blocs Xilinx fonctionnent dans le domaine en temps discret à virgule fixe, mais les signaux du monde réel en virgule flottante, donc les blocs « Gateway In » et « Gateway Out » servent de traducteurs pour traduire le signal du monde réel dans le format souhaité. Le bloc « Gateway In » convertit les données à virgule flottante en données à virgule fixe lisibles par FPGA et le bloc « Gateway Out » convertit les données à virgule fixe en virgule flottante ou en format visible par MATLAB. La figure 4 montre le bloc « Gateway In » et « Gateway Out » utilisé pour définir la conception basée sur la FPGA.

(a) **(b)**

Figure III. 5. (a) Gateway In et **(b)** Gateway Out, utilisé pour définir les limites de la conception basée sur la FPGA**.**

III.5.3. Implémentation du filtre médian 3x3 à 19 comparateurs

Les pixels de l'image bruitée intervenant dans la formation du masque sont obtenus par les Line buffers. Deux Line buffers sont requises pour séparer trois lignes séquentielles de l'image d'entrée. La taille de ces derniers est choisie en fonction de la résolution de l'image d'entrée. A chaque itération, un noyau 3x3 de pixels est formé à partir du pixel d'entrée des registres à décalage et des sorties des Line buffers.

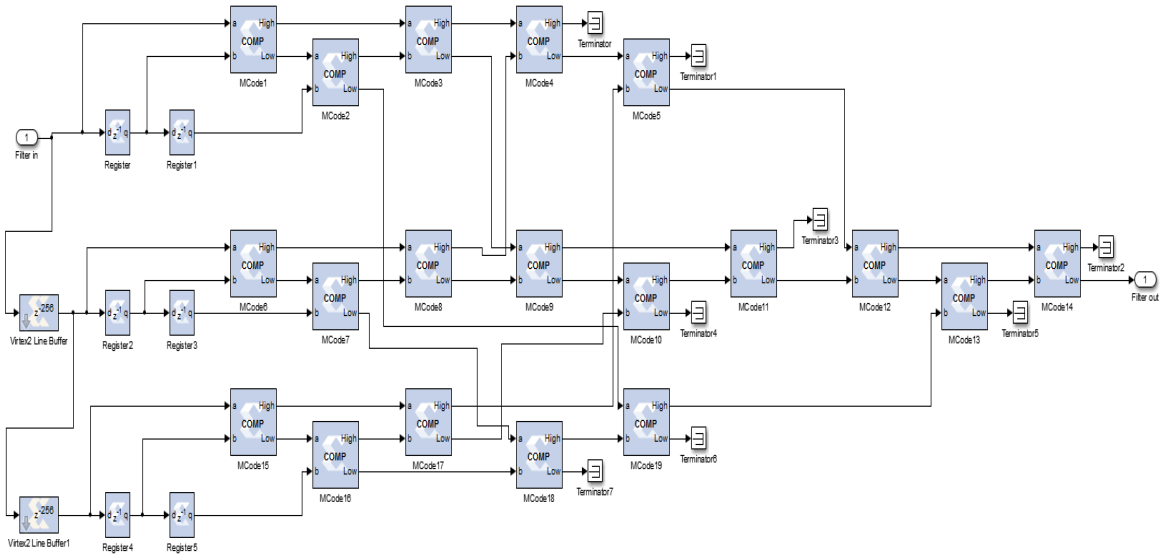


Figure III. 6. Conception XSG du filtre médian 3×3 utilisant un réseau à 19 comparateurs.

III.5.4. Implémentation du filtre médian 5x5 à 273 comparateurs

L'implémentation du filtre médian 5x5 à 279 comparateur utilisant des blocs Xilinx est illustré est illustrée comme à la **figure III. 7** :

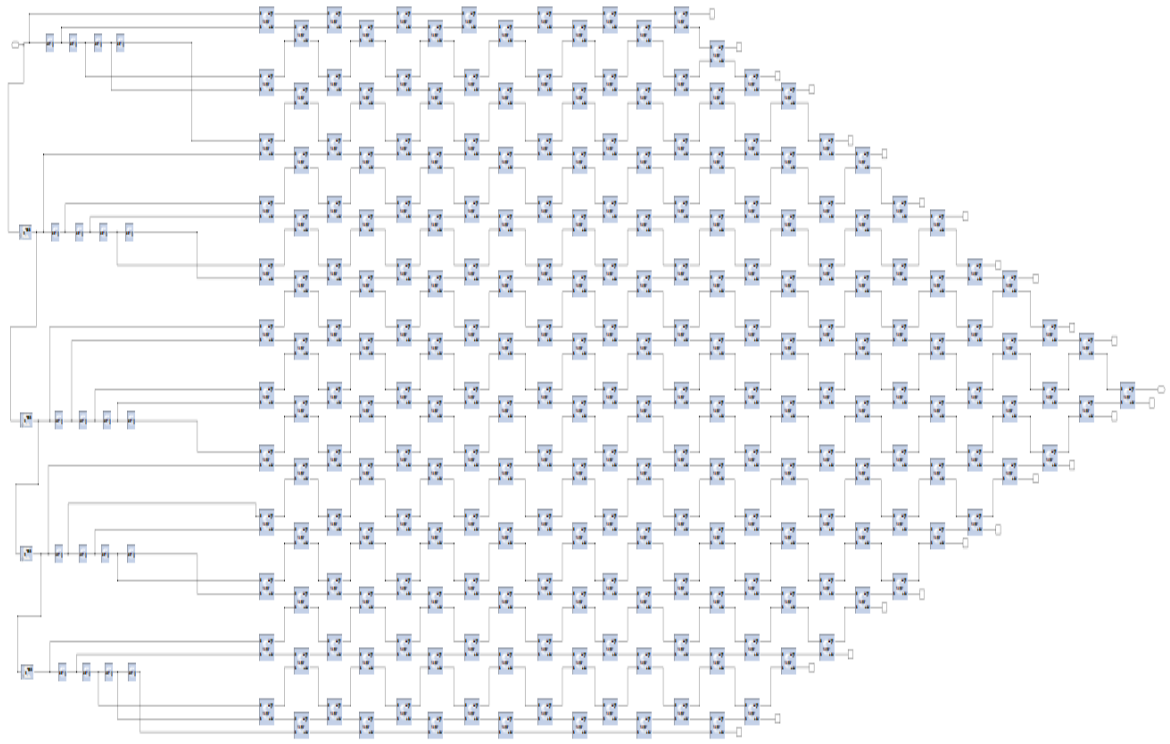


Figure III. 7. Diagramme XSG du Filtre médian de masque 5x5 (273 comparateurs).

III.5.5. Comparateur sous XSG

Le comparateur est le cœur du filtre médian. Toutefois, dans XSG, le comparateur disponible ne peut pas ordonner les entrées, il indique seulement six relations différentes sélectionnées (égal à, supérieur à et inférieur à,etc.). En conséquence, un comparateur particulier est nécessaire pour être utilisé comme comparateur de base dans toutes les conceptions implémentées. Ce comparateur est réalisé à partir du bloc MCode.

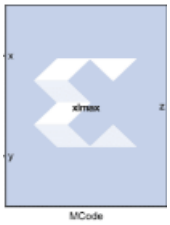


Figure III.8. Bloc MCode.

Le bloc Xilinx MCode est un conteneur pour l'exécution d'une fonction MATLAB fournie par l'utilisateur dans Simulink. Un paramètre sur le bloc spécifie le nom de la fonction M. Le bloc exécute le MCode pour calculer les sorties du bloc lors d'une simulation Simulink. La figure suivante montre le bloc MCode. La structure du bloc MCode ainsi que la fonction Matlab correspondante sont représentés par la **figure III.9.**

```
function  [High,Low]=COMP(a,b)
if a>=b
   High=a;
   Low=b;
else
   High=b;
   Low=a;
end
```

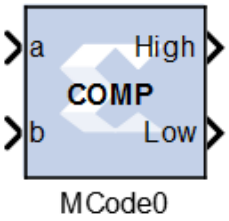
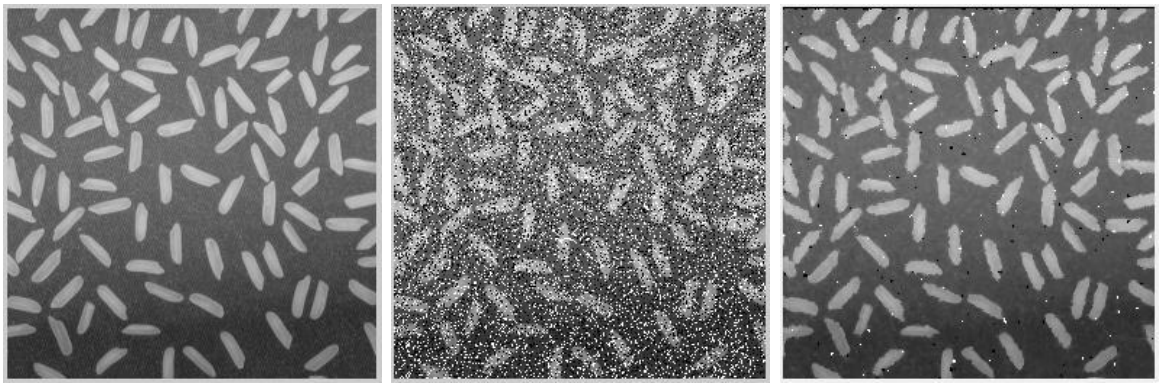


Figure III.9. Bloc MCode et fonction Matalb correspondante de notre conception.

Le bloc MCode ci dessus sera considéré comme un comparateur de base pour toutes les conceptions utilisées.

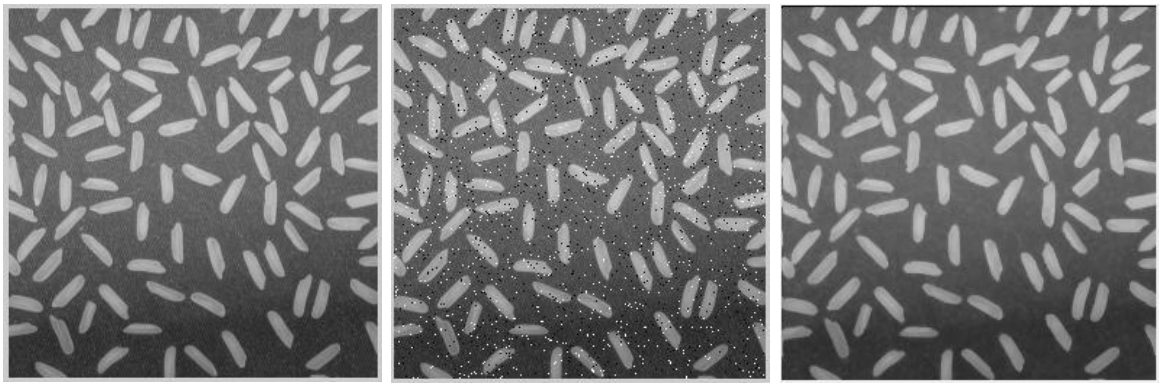
III.6. Résultats de simulation sous XSG

Nous avons appliqué le filtre médian (3x3) et (5x5) sur l'image (Rice.png) affectée du bruit sel et poivre avec deux différentes densités D=3% et D=25%.



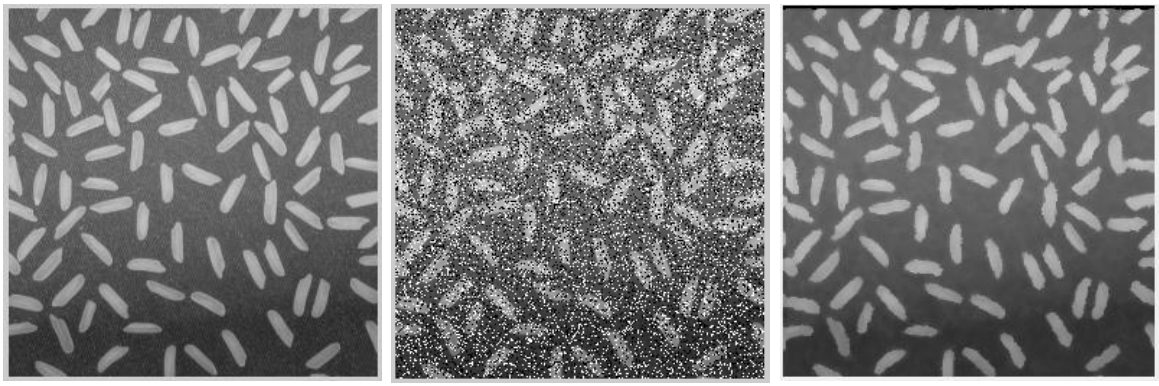
(a) **(b)** **(c)** PSNR=21.05

Figure III. 10. (a) Image originale, **(b)** Image bruitée (sel et poivre D=25%), **(c)** Filtrage médian 3*3 à 19 comparateurs.



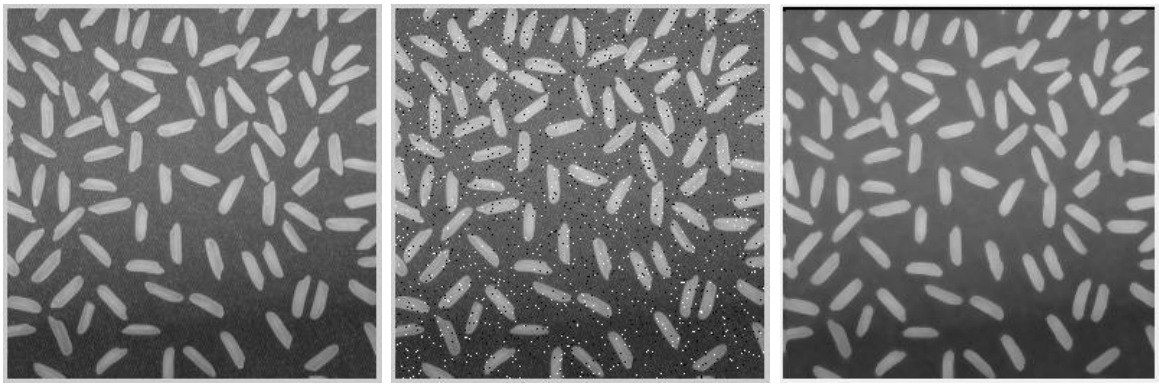
(a) **(b)** **(c)** PSNR= 22.21

Figure III. 11. (a) Image originale, **(b)** Image bruitée (sel et poivre d=3%), **(c)** Filtrage médian 3*3 à 19 comparateurs.



(a) **(b)** **(c)** PSNR= 18.66

Figure III. 12. (a) Image originale, **(b)** Image bruitée (sel et poivre d=25%), **(c)** Filtrage médian 5*5 à 273 comparateurs.



(a) **(b)** **(c)** PSNR=18.76

Figure III. 13. (a) Image originale, **(b)** Image bruitée (sel et poivre d=3%), **(c)** Filtrage médian 5*5 à 273 comparateurs.

III.7. Interprétation

Le tableau III.1 présente les résultats de simulation du filtrage médian 3x3 à 19 comparateurs et 5x5 à 279 comparateurs.

Tableau III.1 : PSNR du filtrage médian.

Masque	PSNR	
	D=3%	D=25%
	Image filtrée	Image filtrée
Masque (3 × 3) à 19 comparateurs	22.21	21.05
Masque (5 × 5) à 279 comparateurs	18.76	18.66

On voit bien une bonne performance du filtre médian pour le bruit sel et poivre.

Ce filtre élimine ce type de bruit tout en préservant les détails de l'image pour les deux type de masque 3x3 et 5x5, dans notre cas, pour une densité $D = 3\%$, le PSNR obtenu pour le filtre médian 3x3 à 19 comparateurs est 22,21 et celui du filtre médian 5x5 à 279 comparateurs est 18.76.

Pour des images fortement bruitées, dans notre cas $D = 25\%$, le filtre 3x3 à 19 comparateurs diminue le bruit impulsionnel sans l'éliminer avec un PSNR=21,05 et celui de 5x5 à 279 comparateurs élimine le bruit mais diminue la qualité de l'image (PSNR de 18,66).

Les valeurs montrent que le PSNR pour le matériel et les logiciels sont équivalents.

III.8. Compilation du modèle pour une Co-Simulation matérielle (Hardware Co-Simulation)

System Generator propose une Co-simulation matérielle, qui permet d'incorporer un design exécuté dans un FPGA directement dans une simulation Simulink. La Co-Simulation matérielle crée le fichier bitstream qui est exécutable par la machine et l'associe à un bloc. Quand le design est simulé dans Simulink, les résultats de la partie compilée sont calculés dans le matériel (FPGA). La Co-simulation matérielle également appelée Hardware in Loop (HIL) nous permet d'utiliser Simulink pour tester la conception sur du matériel réel pour tout code HDL existant. La figure suivante illustre la HIL.

III.8.1. Choix de la plateforme de la Co-simulation

Pour cela, double clic sur Bloc System Generator et dans le menu compilation choisir Hardware Co-Simulation et enfin sélectionner la plateforme Digital Electronics FPGA Board (Digital Electronics FPGA).

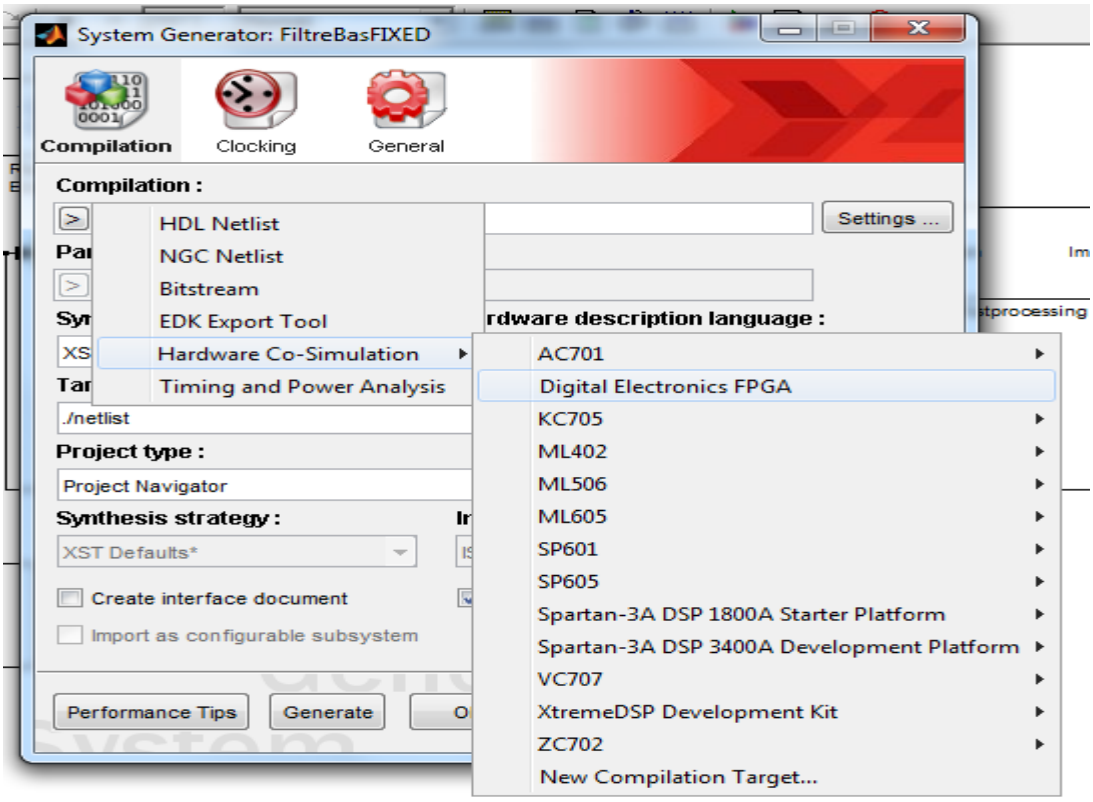


Figure III.14. Choix de la plateforme matérielle.

Lorsqu'une cible de compilation est sélectionnée, les champs de la boîte de dialogue du bloc System Generator sont automatiquement configurés avec les paramètres appropriés pour la cible de compilation sélectionnée. System Generator se souvient des paramètres de la boîte de dialogue pour chaque cible de compilation. Ces paramètres sont enregistrés lorsqu'une nouvelle cible est sélectionnée et restaurés lorsque la cible est rappelée.

III.8.2. Appel du générateur de code

Le générateur de code est lancé en cliquant sur le bouton Generate dans la boîte de dialogue du bloc System Generator.

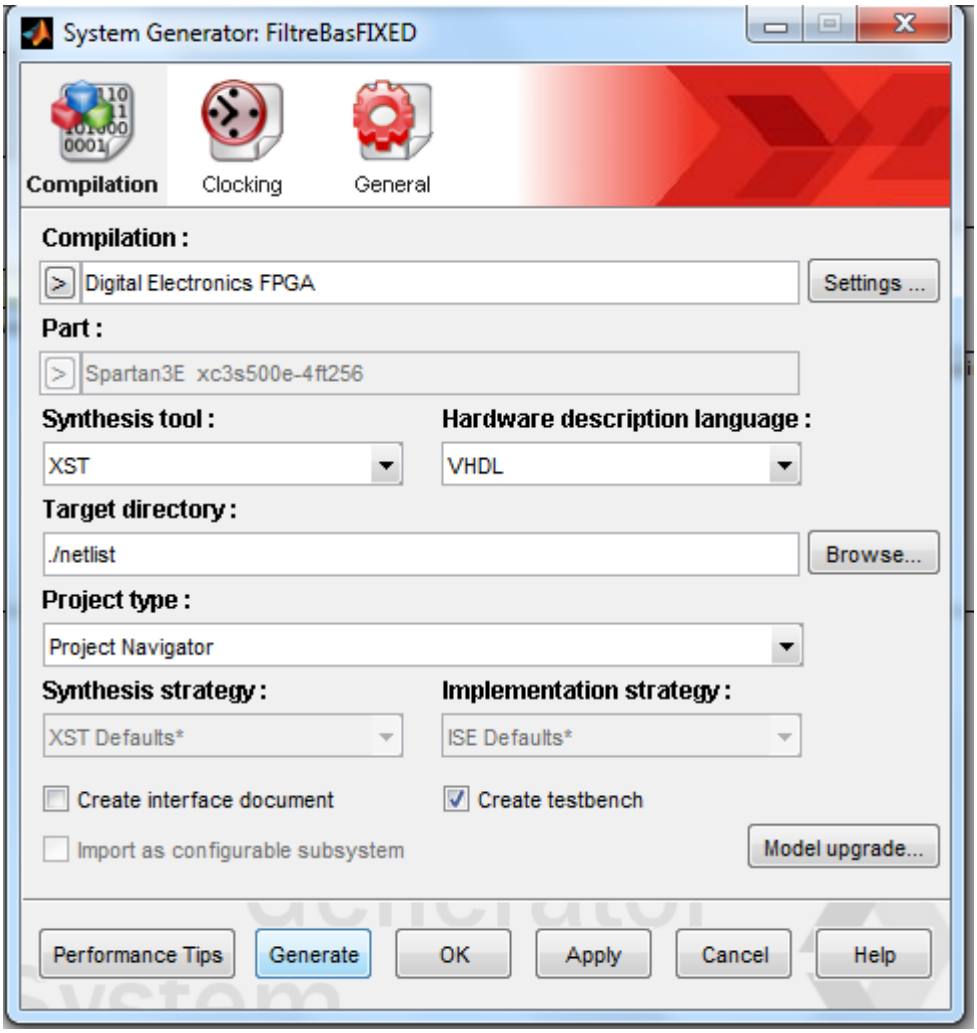


Figure III. 15. Génération de code**.**

Le générateur de code génère le fichier de configuration FPGA (bitstream) pour la conception qui est adapté à la Co-simulation matérielle et qui contient le matériel (Hardware) associé à la conception et également une logique d'interfaçage supplémentaire afin de permettre à System Generator de communiquer avec le matériel en utilisant une interface physique entre la plate-forme et le PC.

III.8.3. Création du bloc Hardware Co-Simulation

Un nouveau bloc de Co-simulation matérielle est automatiquement créé par System Generator une fois que la conception est compilée avec succès dans un fichier de configuration FPGA. Une bibliothèque Simulink est également créée pour stocker le bloc de Co-simulation matériel. L'interface externe du modèle ou du sous-système à partir

duquel il est dérivé est appelée bloc de Co-simulation matériel. Les noms des ports sur le bloc matériel de Co-simulation correspondent aux noms des ports sur le sous-système d'origine. Les types de ports et les taux correspondent également à la conception théorique.

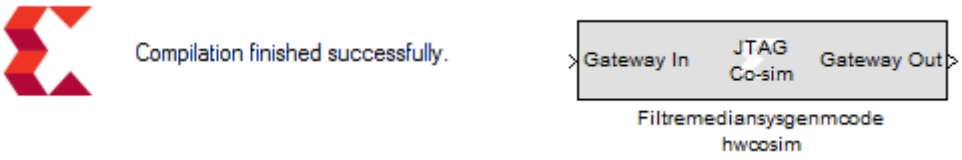


Figure III. 16. Bloc Co-simulation matériel.

III.8.4. Hardware/Software Co-simulation du filtrage median

Pour implémenter la conception dans une carte FPGA, le module entier doit être converti en un format FPGA synthétisable. Pour ce faire, le module principal de filtre médian est converti en une Co-simulation matérielle, réalisée à l'aide du générateur de blocs du système, à savoir son générateur de blocs. Ce bloc est configuré en fonction de la plate-forme cible, et un fichier de flux de bits (*.bit) est créé. Une fois le fichier bit stream généré, la cible matérielle pour la Co-simulation est choisie, et le kit de démarrage Spartan 3E (xc3s500e-4ft256) est utilisé pour tout assembler au niveau de la carte.

Une fois le bloc de Co-simulation est généré, on le copie le bloc en dehors de la bibliothèque et on l'incorpore dans la simulation Simulink du le design du filtre median à 19 comparateur sous XSG. Lorsque la conception est simulée dans Simulink, le résultat de la section compilée est calculé en temps réel FPGA matériel, résultant en un temps de simulation beaucoup plus rapide tout en vérifiant la précision fonctionnelle du matériel.

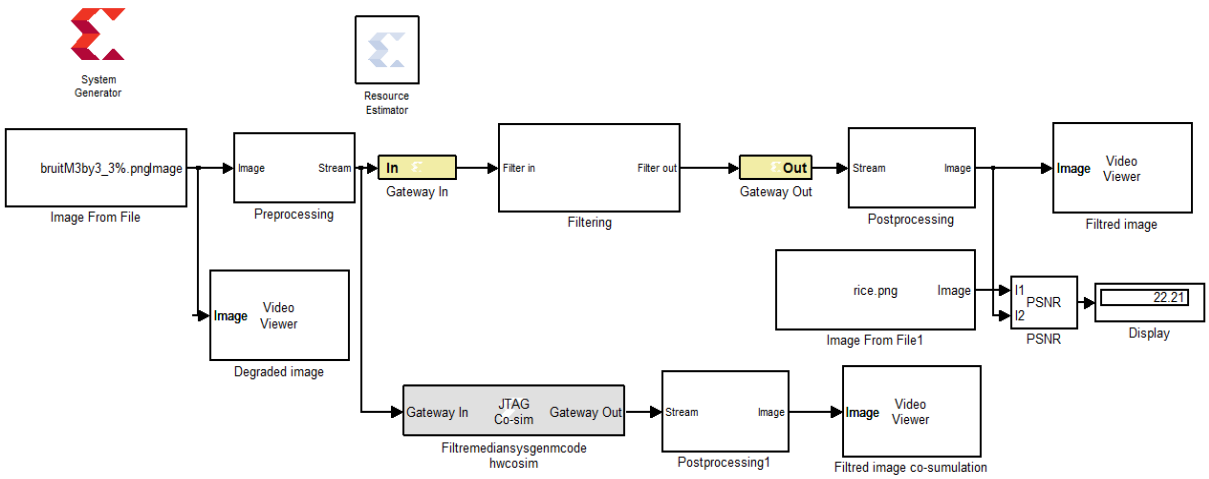
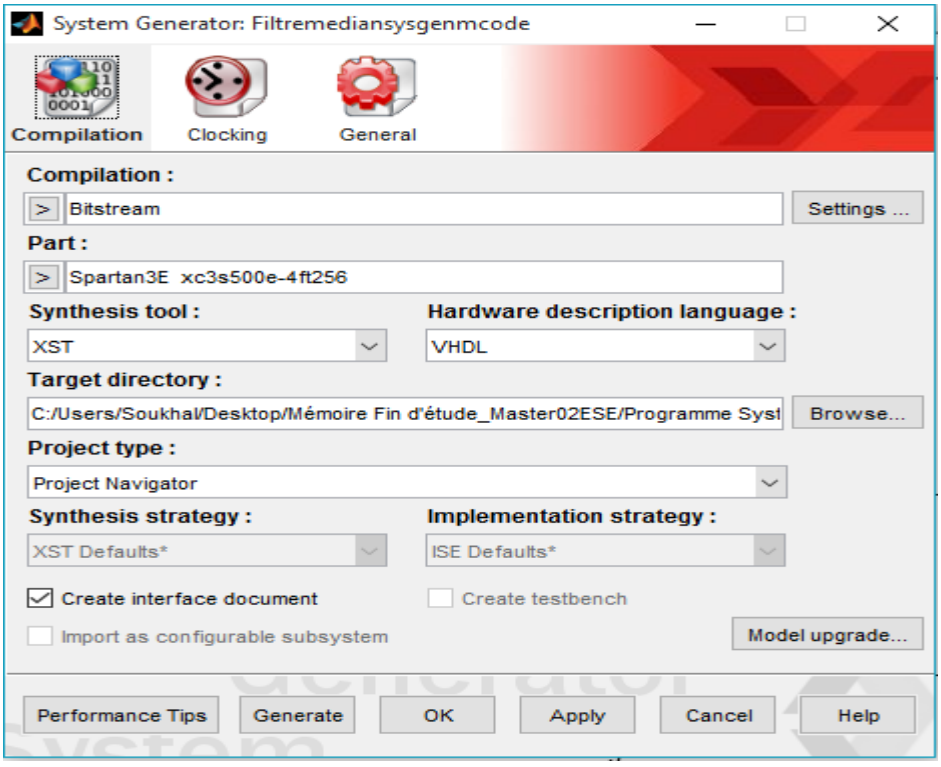


Figure III. 17. Hardware/Software Co-simulation du filtrage median**.**

III.9. Génération du bitstream

Une fois avoir développé le bloc Simulink et simulé le modèle, il faudra paramétrer le bloc System Generator et à lancer la génération du fichier Bitstream (données binaires Bitstream utilisées pour configurer le FPGA) ainsi que le fichier VHDL comme le montre la **figure III. 18 :**



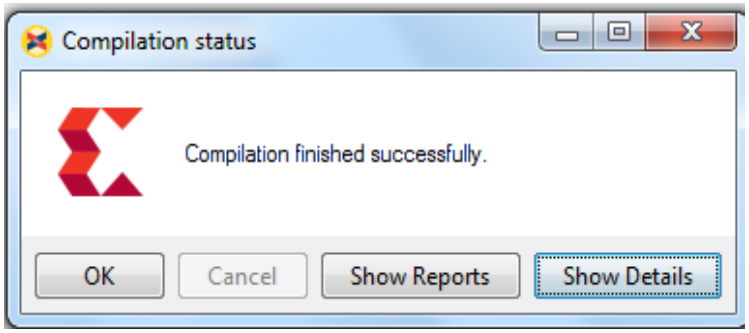


Figure III. 18. Génération du bitstram.

Une fois la génération du Bitstream est réussie, il ne reste qu'à charger ce dernier dans le FPGA avec le logiciel iMPACT.

III.10. Conclusion

Nous pouvons conclure de ce chapitre que Xilinx System Generator est un outil polyvalent pour effectuer des tâches de traitement d'image logiciel et matériel. Il fournit des outils rapides pour mettre en œuvre des techniques de traitement d'image complexes au niveau matériel avec le moins de ressources et de temps.

La Co-simulation simplifie la validation du design du filtre médian en permettant de comparer directement les résultats du code HDL généré et du modèle comportemental du filtre exécuté dans Simulink. Cette intégration permet d'utiliser les capacités d'analyse et de visualisation avancées de MATLAB et Simulink pour tester, débugger et valider l'implémentation HDL du design.

Dans ce chapitre, nous avons implémenté le filtrage médian 3x3 à 19 comparateurs et 6x6 à 273 comparateurs sous XSG puis transférer le design sur FPGA à l'aide de la Co-Simulation matérielle (Hardware Co-Simulation de Xilinx). Enfin, le filtrage a été validé sur la plate-forme de développement DEFB.

Conclusion générale

La présente étude a pour principal objectif l'implémentation du filtre médian à base de comparateurs sur une plate-forme FPGA en utilisant une compilation pour une Co-Simulation matérielle et en se servant du code VHDL généré par l'outil HDL Coder. Ce filtre permet d'éliminer le bruit poivre et sel tout en préservant les détails de l'image à la fois pour les masques 3x3 et 5x5 pour une densité de bruit faible, mais pour les images à fort bruit, le filtre 5x5 est plus approprié.

Le cœur du filtre médian est l'élément de comparaison, donc, un comparateur spécial a été réalisé à partir du bloc MCode. On a développé une fonction Matlab qui est mise en boite par l'outil System Generator et est considéré comme un comparateur de base pour la conception utilisée.

Nous avons implémenté le filtre médian 3x3 avec 36 comparateurs et optimisé à 19 comparateurs ainsi que le filtre 5x5 avec 300 comparateurs et optimisé à 234 sous la plateforme Simulink. L'outil HDL Coder a été utilisé pour générer le code VHDL à partir du modèle Simulink développé pour la cible matérielle FPGA et ensuite la génération du fichier de programmation a été réalisée avec le logiciel ISE Design Suite.

Nous avons modélisé et simulé le filtre médian 3x3 et 5x5 à base de comparateurs utilisant le puissant outil Xilinx System Generator (XSG). Ce dernier fournit une Co-Simulation matérielle qui permet d'incorporer le design fonctionnant dans un FPGA directement dans une simulation Simulink et permet de transférer le design sur FPGA. Lorsque le design est simulé dans Simulink, les résultats de la partie compilée sont calculés dans le matériel (hardware : FPGA).

Nous avons pu ainsi enrichir nos connaissances dans le domaine du traitement d'images et appréhender la méthodologie de conception d'applications sur FPGA avec l'outil Xilinx System Generator XSG et HDL Coder.

Nous concluons de ce travail que le Xilinx System Generator constitue un outil versatile pour réaliser des tâches logicielles et matérielles de traitement d'images. Nous pensons que ce travail est appelé à ne pas rester sans suite.

Bibliographie

[1] Rasheed, A. H. (2017). FPGA-based Optimized Systolic Design for Median Filtering Algorithms. International Journal of Applied Engineering Research, 12(24), 16100-16113.

[2] N. Diffellah, "Débruitage et simplification d'images," Ph.D. thesis, Dept. G Elect., Biskra, Algérie, 2021.

[3] Toh, K. K. V., & Isa, N. A. M. (2009). Noise adaptive fuzzy switching median filter for salt-and-pepper noise reduction. IEEE signal processing letters, 17(3), 281-284.

[4] Debayle, J., & Pinoli, J. C. (2006). General adaptive neighborhood image processing. Journal of Mathematical Imaging and Vision, 25(2), 245-266.

[5] https://www.iro.umontreal.ca/~mignotte/IFT6150/Chapitre5_1_IFT6150.pdf

[6] Moussa, S. (2020). Image denoising: étude comparative des méthodes de filtrage d'image.

[7] Saâdi, N. (2015). Etude comparative entre le débruitage d'images par des méthodes classiques et par le filtre bilatéral.

[8] Kirchner, M., & Fridrich, J. (2010, January). On detection of median filtering in digital images. In Media forensics and security II (Vol. 7541, p. 754110). International Society for Optics and Photonics.

[9] https://slideplayer.fr/slide/1153263/3/images/25/FILTRE+MEDIAN.jpg

[10] Qidwai, U., & Chen, C. H. (2009). Digital image processing: an algorithmic approach with MATLAB. Chapman and Hall/CRC.

[11] Zhao, J. (2015). Video/Image Processing on FPGA (Doctoral dissertation, Worcester Polytechnic Institute).

[12] DIVYA, P., & HARSHITHA, M. IMPLEMENTATION OF HOME AUTOMATION SYSTEM DESIGN ON FPGA USING VHDL.

[13] Abdullah, H. N., & Hadi, H. A. (2009). Design and Implemetation of FPGA based Software Defined Radio Using Simulink HDL Coder.

[14] Ayoub, B. E. N. S. A. I. D. (2021). Hardware Software Co-Simulation For image filtering and blurring Using Xilinx system generator (Doctoral dissertation, Faculté des Sciences ET Technologies).

[15] Raut, N. P., & Gokhale, A. V. (2013). FPGA implementation for image processing algorithms using xilinx system generator. IOSR Journal of VLSI and Signal Processing (IOSR-JVSP), 2(4), 26-36.

[16] System Generator for DSP Reference Guide UG638 (v14.5) March 20, 2013.