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 : **électronique**

Spécialité : électronique industrielle

Par

- **SOMRANI NADER**
- **ZAIDI HICHAM**

Intitulé

Conception matérielle de blocs de construction destinés à des algorithmes de tatouage d'image

Soutenu le : ……

Devant le Jury composé de :

Nom & Prénom	*Grade*	*Qualité*	*Etablissement*
M. Youssfi A.		***Président***	***Univ-BBA***
Mme Meguellati S.		***Encadreur***	***Univ-BBA***
M. Abed T.		***Co. Encadreur***	***Univ-BBA***
M. Laouamri A.	*….*	***Examinateur***	***Univ-BBA4***

Année Universitaire 2021/2022

Résumé:
On présente une implémentation à base de description VHDl d'un algorithme de dissimulation de donnée réversible. Destiné à un support image en niveaux de gris, l'algorithme choisi est à base de tri de valeur de pixel et d'expansion de l'erreur de prédiction (PVO-PEE). La nature du traitement réalisée sur des valeurs de pixels favorise une implémentation matérielle sur support configurable FPGA.
Nous avons implémenté les fonctions clés de cet algorithme sous forme d'une bibliothèque de blocs de construction permettant le codage (intégration) et le décodage (récupération) de donnée dissimulées. Les simulation réalisées dans l'outil ALDEC Active HDL prouvent le bon fonctionnement.

Abstract :
We present a VHDL based implementation of a reversible data-hiding algorithm. Using binary grey-level images as host support, the chosen algorithm is based on pixel value ordering and prediction error expansion techniques. The pixel-based operations suggest configurable hardware implementation on FPGA as an attractive solution.
We have implemented key functions of this algorithm as a building blocks library that could be used for the coding (hiding) and the decoding (restoration) of hidden data. The simulations are done in ALDEC Active HDL tools and prove a good operation.

Sommaire

Liste des figures

Dédicaces

Je dédie ce travail A mes parents : Au-delà des mots et des phrases, aucune parole ni aucune dédicace ne saurait exprimer mon profond amour, mon éternel attachement, ma perpétuelle affection et l'infinie gratitude que je vous dois. Vous êtes tous pour moi et je vous dois tout..

Aussi, je souhaite remercier mes deux frères ***Oussama*** *et* ***Toufik****, ma sœur* ***Tassnime,***

Mes Collègues et amis ***Mohamed Amine, Islam, Abdelkader, Zakaria Mohamed Saïd, Abdenacer, Chemsou*** *pour leur soutien constant et leur sympathie.*

Enfin, je dédie ce projet de fin d'étude à toute ma famille et à tous mes amis pour les peines et joies partagées ensemble tout au long de notre parcours universitaire.

SOMRANI Nader

Dédicaces

Je dédie ce travail A mes parents ***Abd elhafid et Messaouda*** *: Au-delà des mots et des phrases, aucune parole ni aucune dédicace ne saurait exprimer mon profond amour, mon éternel attachement, ma perpétuelle affection et l'infinie gratitude que je vous dois. Vous êtes tous pour moi et je vous dois tout..*

Aussi, je souhaite remercier mes frères ***Mustapha****,* ***Rabah****,* ***Djalal, Azzedine*** *et mes sœurs* ***Meriem et Souad,*** *Mes Collègues et amis* ***Brahim, Imad, Nasro, Riyad,*** *pour leur soutien constant et leur sympathie.*

Enfin, je dédie ce projet de fin d'étude à toute ma famille et à tous mes amis pour les peines et joies partagées ensemble tout au long de notre parcours universitaire.

ZAIDI Hichem

Remerciements

Le grand remerciement revient à Dieu, qui m'a donné la force de pouvoir terminer ce travail. Nous tenons tout d'abord à exprimer notre plus sincère gratitude à notre encadrante ***Mme. MEGUELLATI Sabrina****, pour sa patience, sa disponibilité, pour la confiance qu'elle nous a accordée, pour nous avoir guidé, suivi et aidé tout au long de ce travail, mais surtout pour son apport scientifique sans lequel ce mémoire n'aura jamais vu le jour.*

Un grand merci à ***M.ABED Tarek*** *pour nous aider à faire ce mémoire.*

Un très grand merci aux membres du jury pour bien vouloir évaluer notre travail.

Je tiens aussi à remercier toute l'équipe administrative du département d'électronique Industrielles pour tous leurs efforts qui ont rendus cette formation possible.

Un grand merci pour toute personne qui a contribué de près ou de loin à la réalisation de ce mémoire.

Introduction générale

L'utilisation d'internet dans plusieurs secteurs d'activité, tels que : la télé-éducation, la santé, les activités militaires et bien d'autres et maintenant un fait. La quantité de données numériques échangées est donc immense, facilitée par l'efficacité des réseaux de communication et l'augmentation constante des débits de transfert de données. Cependant, cette même facilité ainsi que la gratuité d'accès aux données a banalisé le piratage des œuvres multimédias, grâce au téléchargement direct permettant le partage sans contrôle des fichiers audio et vidéo. Ceci a ramené le développement de plusieurs techniques de dissimulation de données (DH : data hiding) dans les œuvres. Plus particulièrement, la dissimulation de donnée réversible (RDH : reversible DH) représente un domaine de recherche très actif. Les méthode RDH permettent un transfert de données sécurisé tout en assurant un recouvrement fidèle du média hôte. En général, les techniques RDH actuelles visent l'amélioration de la capacité d'intégration en conservant la qualité.

Dans ce travail, on essaye d'ajouter une contribution dans le développement de la sécurité des images à travers le tatouage réversible en proposant une implémentation matérielle. On a choisi un algorithme à base d'expansion d'erreur et de tri de valeur de pixels (PEE/PVO) . Utilisant le langage de description universel VHDL, les codes écrits forment une bibliothèque qui servira à une implémentation complète du système dans une FPGA. Notre mémoire est organisé en quatre chapitres : le premier passe en revue les techniques de dissimulation dans le domaine spatial reconnues actuellement. Le chapitre 2 présente l'essentiel de la technologie reconfigurable FPGA ainsi que quelques contributions existantes dans la littérature pour implémenter des algorithmes de tatouage d'images. Dans le chapitre 3 on détaille l'algorithme qu'on a choisi tandis que la conception et l'implémentation des blocs de construction sont détaillées dans le chapitre 4. On finalise le mémoire par une conclusion et une perspective de continuité de ce travail.

Chapitre I : Dissimulation de données réversible

Dans ce chapitre, on présente quelques notions générales sur la dissimulation de données ainsi qu'une classification de base des techniques piliers dans ce domaine. On passe en revue les technique RDH dans le domaine simple ainsi que dans le domaine crypté.

I.1 Généralités et Définitions :

Dans certains environnements hostiles, cacher l'existence d'une communication est nécessaire pour éviter un certain nombre d'attaque de la part d'adversaires malveillants. Ceci a fait que la dissimulation de l'information existe depuis plusieurs milliers d'années : par exemple, la sténographie qui fait partie de la dissimulation de l'information existait déjà en Grèce Antique.

La dissimulation de l'information a survécu le temps jusqu'à nos jours et connait même une grande révolution avec l'avènement d'Internet et des supports numériques : des centaines d'articles et de communication traitant des différentes techniques d'utilisation de la dissimulation de l'information [1] sont publiés chaque année. Elle est devenue l'un des outils les plus efficaces dans la lutte contre le piratage multimédia et dans l'encadrement de la diffusion des données numériques.

Le data-hiding (dissimulation de l'information) compte plusieurs classifications et partitionnements. Par exemple, selon l'application ou encore selon le modèle de communication adoptée par le système du data-hiding :

a) **Stéganographie :** elle consiste en l'altération d'un contenu d'une manière indétectable dans le but d'insérer un message [2]. Elle est la plus ancienne application du data-hiding. Elle sert généralement à communiquer secrètement à travers des réseaux publics. Ainsi, elle est souvent utilisée dans le domaine militaire ou le contre-espionnage.

b) **Tatouage numérique :** C'est l'application la plus connue du data-hiding. D'ailleurs, le tatouage numérique est souvent utilisé comme le terme générique qui désigne tous les systèmes de data hiding. Il est défini comme l'ensemble des modifications robustes et invisibles dans un document hôte. Ces modifications permettent d'insérer une information

qui concerne directement le document tatoué, copyright par exemple, ou indirectement, tel que les droits de l'utilisateur sur le contenu [3] [4] [5].

c) **Estampillage (Fingerprinting) :** [6] C'est une technologie donnant la possibilité à la justice d'identifier et punir les personnes responsables de la diffusion des copies non-autorisées Pour pouvoir identifier l'utilisateur qui est à l'origine d'une distribution illicite d'une vidéo, une marque spécifique est insérée pour chaque utilisateur.

Les systèmes de data hiding sont généralement composés d'encodeur et de décodeur, l'information est transmise en utilisant un support hôte qui joue le rôle d'un canal de transmission. Ainsi, un système de data hiding est bien un système de communication [5]. C'est la raison pour laquelle il existe une deuxième classification selon le modèle de communication adopté.

d) **Système de data-hiding sans information adjacente ou non-informé :** Lorsqu'un système de data-hiding insert une information sans tenir compte des caractéristiques du signal hôte, le système est dit non-informé. Ce type de système a été initialement adopté par les premiers schémas de data hiding.

e) **Système de data-hiding avec information adjacente (systèmes informés) :** D'après les travaux de Costa [7] dans le domaine des télécommunications, le signal hôte est considéré comme un moyen d'adapter l'encodage au canal de transmission. De cette manière, le recouvrement de la marque est plus efficace et le système devient plus robuste.

Une autre manière de classification concerne l'aspect conservateur des données : les techniques de data-hiding peuvent être à pertes ou sans pertes. Le data-hiding à pertes [8] vise une grande capacité d'incorporation (embedding) sans exiger un recouvrement fidèle du média hôte. Or, dans plusieurs applications (militaire, médical ,authentification [9, 10], codage d'image stereo..etc.) la distorsion du médium hôte est inacceptable. Ceci a fait naitre les méthodes de data-hiding sans pertes, appelées encore reversible data-hiding (RDH), où l'on tient compte et exige une recouvrement du média hôte aussi complet que possible en plus de la grande capacité d'incorporation.

I.2 Classification des techniques RD

La dissimulation de donnée réversible peut être classifiée principalement en : domaine simple (plain) et domaine encrypté. La figure I.1 résume cette classification proposée par [11]. Le média hôte peut être : un texte, une image, une vidéo ou une audio.

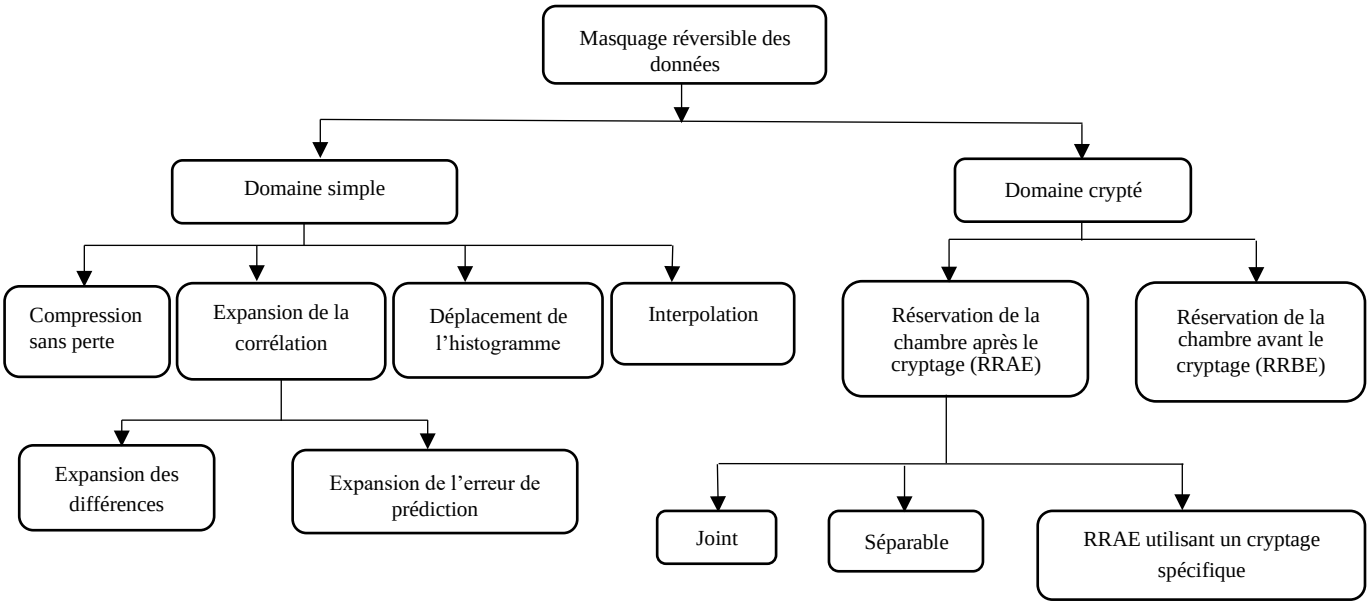


Figure I.4:classification des méthodes RDH

I.2.1 Méthodes RHD du domaine encrypté:

Le média hôte est crypté dans le but de protéger l'intimité et la sécurité. Afin de décrypter le media tatoué, le récepteur au choix entre différentes clés.

Le domaine encrypté fait implique trois intervenants : le propriétaire, le dissimulateur de données et le récepteur. (Figure I.2)

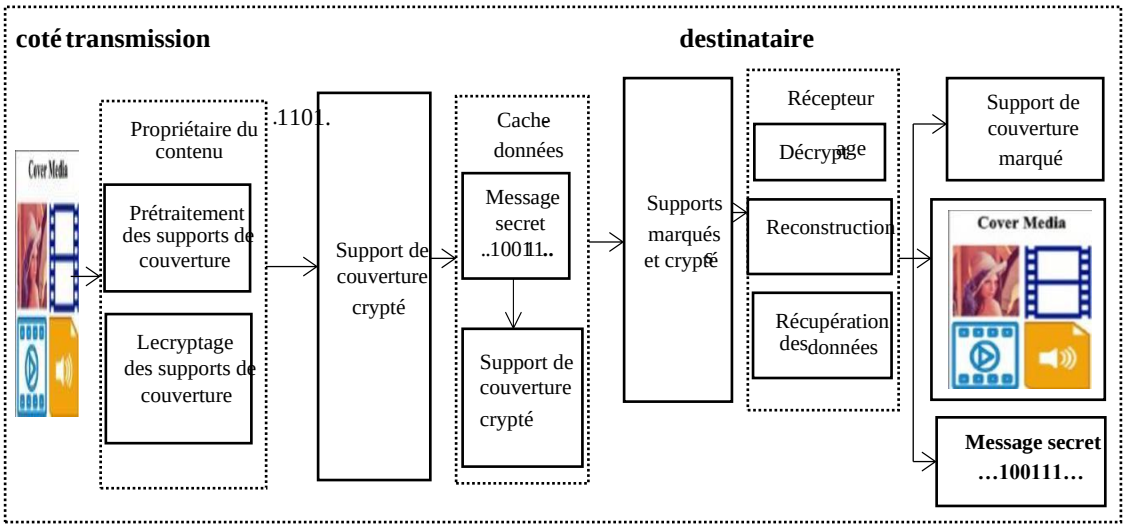


Figure I.5:dissimulation de donnée dans le domaine crypté

Généralement, le media hôte est soit prétraité soit directement crypté utilisant une clé.

L'information secrète est alors intégrée dans le media original par le dissimulateur de données. Le média tatoué est envoyé vers un récepteur qui va extraire les données secrètes.

Afin d'assurer la sécurité, les techniques RRBE (reserving room before encryption: réservation d'espace avant cryptage)[12,13] et RRAE (reserving room after Encryption : réservation d'espace après cryptage) [14,15] ont été élaborées.

RRAE : la corrélation spatiale est perdue à cause du cryptage du media hôte. Un algorithme d'extraction sans pertes permet au récepteur le décryptage du media tatoué , utilisant une clé, afin de récupérer la donnée secrète et le media original.

Les technique RRAE non-séparables, le media tatoué est d'abord décrypté ensuite l'information secrète est extraite utilisant une clé de dissimulation et une clé de décryptage.

Dans la RRAE séparable, il est possible d'extraire l'information secrète et/ou décrypter le media , selon la clé.

RRAUSE: le media original est crypté en utilisant des méthodes spécifiques qui préservent une certaine corrélation entre les pixels.

RRBE : le prétraitement permet de créer un espace dans le media original qui est ensuite crypté utilisant une clé. Le media crypté et tatoué est envoyé vers un récepteur où l'information secrète et le media original sont récupérés.

I.2.2 Méthodes RDH du domaine simple (plain) :

Ces méthodes se concentrent sur la réalisation d'une grande capacité d'intégration des donnée tout en minimisant la distorsion. La donnée secrète est incorporée dans le média hôte sans cryptage.

Le média hôte est prétraité afin de libérer un espace suffisant pour incorporer des données. En plus, des informations secrètes sont aussi incluses dans le média hôte en utilisant un algorithme d'intégration sans pertes. Le résultat, un média hôte tatoué, est alors envoyé à travers un canal de communication. Le récipient récupère l'information secrète. Le média hôte original est regagné avec fidélité usant un algorithme d'extraction de données.

Dans la figure I.3 on trouve les techniques du domaine simple catégorisées principalement en : compression sans pertes[16,17,18], l'expansion de corrélation[21–22], décalage d'histogramme [23,24,25] et l'interpolation[26,27].

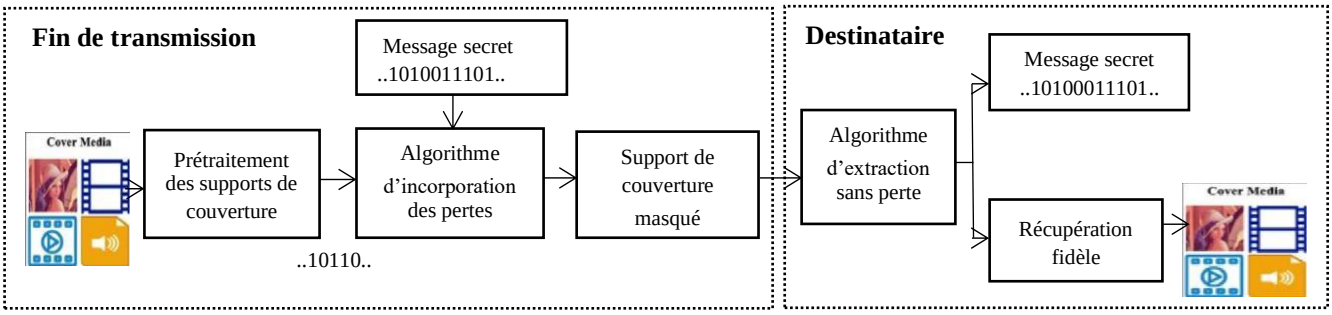


Figure I.6: Dissimulation de donnée dans le domaine simple

I.2.2.a RDH à base de compression sans perte :

Une partie du media original est compressée pour permettre l'intégration des données. Ces méthodes exigent des calculs plus puissants et intègrent moins de données (capacité d'intégration plus basse). Elles sont donc utilisées pour des fins d'authentification. Il n'y a pas de perte de qualité mais la taille du media original est réduite à cause de la compression. L'utilisation de ces méthodes est limitée à cause du temps et de la complexité du décodage.

I.2.2.b RDH à base d'interpolation : l'information secrète est intégrée dans des pixels interpolés sans changements dans le media original.

I.2.2.c RDH à base d'expansion de la corrélation : On en trouve : l'expansion de la différence (DE) [20–22] et expansion de l'erreur de prédiction (PEE) [28, 29,30].

L'expansion de la différence utilise la corrélation entre deux pixels voisins, tandis que l'expansion de l'erreur de prédiction considère la corrélation locale d'un voisinage de pixels plus grand, afin d'intégrer la données secrète.

RHD à base d'expansion de la différence (DE) : Dans [19, 20] , Tian a proposé une technique promettante a base de DE. Une transformée ondelette de Haar entière est d'abord appliquée à l'image originale afin de former les valeurs de différences , ensuite , ces valeurs sont étendues afin de créer des espaces pour intégrer des données.

Pour une paire de pixels (x0,x1), on définit la moyenne et la différence (entières) comme $l = \lfloor (x_0 + x_1)/2 \rfloor$ et h=x0-x1 , respectivement. Afin d'intégrer un bit $m \in [0,1]$, la différence est étendue en h*=2h+m tout en gardant la moyenne l fixe. La paire de pixels tatouée est calculée en se basant sur la nouvelle valeur de différence h* et la moyenne originale l. c a d , la paire de pixels tatouée (y0 ;y1) est calculée comme :

$$\begin{cases} y_0 = l - \lfloor h^*/2 \rfloor \\ y_1 = l + \lfloor (h^* + 1)/2 \rfloor \end{cases} === \begin{cases} y_0 = 2x_0 - \lfloor (x_0 + x_1/2 \rfloor \\ y_1 = 2x_1 - \lfloor (x_0 + x_1/2 \rfloor + m \end{cases} \quad (I.1)$$

Le décodeur peut déterminer le bit intégré m comme étant le bit LSB de y1-y0, et la paire de pixels originaux est alors récupérée comme :

$$\begin{cases} x_0 = l' - \lfloor h'/2 \rfloor \\ x_1 = l' - \lfloor h'/2 \rfloor \end{cases} \quad (I.2)$$

Où $l' = \lfloor (y_0 + y_1)/2 \rfloor$ et $h' = \lfloor (y_1 - y_0)/2 \rfloor$ sont calculés à partir de la paire tatoué (y0 ;y1).

Par l'expansion de la différence, il est possible d'intégrer un bit dans une paire de pixels ; ceci permet un grand taux d'intégration (jusqu'à 0.5bit par pixel).

Comparée les méthodes précédentes (à base de compression sans perte), la méthode DE est meilleure en fournissant une grande capacité d'intégration et faible distorsion.

Tian a employé une carte d'emplacement (location map : LM) pour garder la trace des emplacements expansibles sélectionnés (les paire de pixels ayant une faible différence). La

LM est adoptée dans plusieurs autres algorithmes, et la technique DE est devenue fondamentale comme technique RDH, à la base d'autres technique tel que transformation entier-entier (IT), PEE et intégration adaptive.

RHD à base de la transformation entier-entier (IT integer-to-integer transformation):

Alattar [22] a proposé une amélioration de la méthode de Tian en généralisant la paire de pixels en blocs de pixels de taille arbitraire. Ici, la méthode d'intégration est la suivante :

$$\begin{cases} y_0 = 2x_0 - \left\lfloor \frac{2\sum_{i=0}^{n} x_i + \sum_{i=1}^{n} m_i}{n+1} \right\rfloor + \left\lfloor \frac{\sum_{i=0}^{n} x_i}{n+1} \right\rfloor \\ y_1 = 2x_1 - \left\lfloor \frac{2\sum_{i=0}^{n} x_i + \sum_{i=1}^{n} m_i}{n+1} \right\rfloor + \left\lfloor \frac{\sum_{i=0}^{n} x_i}{n+1} \right\rfloor + m_1 \\ y_n = 2x_n - \left\lfloor \frac{2\sum_{i=0}^{n} x_i + \sum_{i=1}^{n} m_i}{n+1} \right\rfloor + \left\lfloor \frac{\sum_{i=0}^{n} x_i}{n+1} \right\rfloor + m_1 \end{cases} \quad \text{(I.3)}$$

Où $(x_0, x_1, \dots\dots, x_n) \in \mathbb{Z}^{n+1}$, $(y_0, y_1, \dots\dots, y_n) \in \mathbb{Z}^{n+1}$ et $(m_1, \dots\dots, m_n) \in (\mathbb{Z}_2)^n$ représentent les pixels du block hôte de taille n+1 , les pixels tatoués et la donnée à intégrer, respectivement. Avec cette transformation nbits sont intégrés dans n +1 pixels avec la possibilité d'augmenter le taux d'intégration à 1 pixel par bit pour n assez grand.

En général, les méthodes à base de la méthode IT commencent par la division de l'image en blocs, ensuite d'intégrer les données dans chaque bloc. L'avantage est la réduction de l'impact de la carte d'emplacement LM sur la performance. L'inconvénient étant l'utilisation une prédiction moins efficace (la valeur moyenne est utilisée pour prédiction de chaque pixel dans un bloc). Cependant, les méthodes IT ne peuvent pas contrôler la modification maximale des pixels originaux. Ceci fait que la IT soit efficace pour les grandes quantités de données à intégrer.

RHD à base d'expansion de l'erreur de prédiction (PEE)

Parmi les maintes extensions de DE, PEE attire une attention considérable car elle permet une meilleure exploitation de la redondance spatiale dans une image. Ici, la corrélation locale d'un grand nombre de pixels voisins (par opposition à deux pixels dans DE) est considérée, ce qui aboutit à une meilleur performance. Cette technique a été proposée

pour la première fois par Thodi et Rodriguez [28,29], et est devenue à la base de plusieurs nouvelles RDH [31,32].

Dans PEE, un prédicteur de pixels est utilisé comme opérateur de corrélation au lieu de la différence utilisée dans DE. Considérant un détecteur de contour médian, par exemple, les proches voisins droits, gauches et diagonaux d'un pixel x sont utilisés pour donner une prédiction de sa valeur $\hat{x}$. L'erreur de prédiction est calculée comme$e = x - \hat{x}$. L'erreur est ensuite étendue à $e^* = 2e + m$ afin d'intégrer un bit $m \in [0,1]$. Finalement, le pixel tatoué est $e^* + \hat{x}$. Pour l'extraction, le décodeur la valeur de prédiction $\hat{x}$, qui doit être la même que celle obtenue dans l'encodeur. Ensuite, l'erreur de prédiction tatouée est calculée comme $e^* = y + \hat{x}$. Finalement, la donnée intégrée est extraite : c'est le bit LSB de e* et le pixel original est récupéré par : $y - \lceil e^*/2 \rceil$

Le point commun entre la DE et la PEE est l'intégration à base d'expansion, dans laquelle la valeur différence et l'erreur de prédiction sont respectivement étendues dans DE et PEE. Dans PEE 1 bit peut être intégré dans un pixel ce qui donne un taux d'intégration de 1bpp (contre 0.5 bpp dans DE). La conséquence est la capacité d'intégration est améliorer. Le PEE permet aussi d'améliorer la performance de la RHD à base de décalage d'histogramme HS; ceci est discuté dans la section suivante.

I.2.2.d Décalage d'histogramme (Histogramme shifting HS):

La procédure commence par la génération d'un histogramme de l'image originale, ensuite, l'intégration de donnée réversible est réalisée par modification de cet histogramme. Cette méthode a été proposée la première fois par [33 ,34] : pour un entier a donné, le message secret est intégré dans l'image I de la manière suivante :

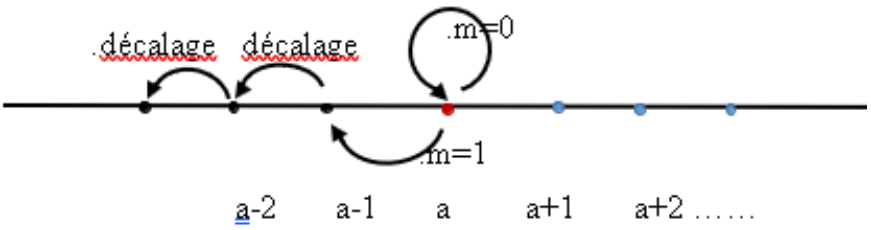
$$J_{i,}\{\begin{matrix} I_{i,}-1 & si\ I_{i,j} < a \\ I_{i,j}-m & si\ I_{i,j} = a \\ I_{i,j} & si\ I_{i,j} > a \end{matrix} \qquad \text{(I.4)}$$

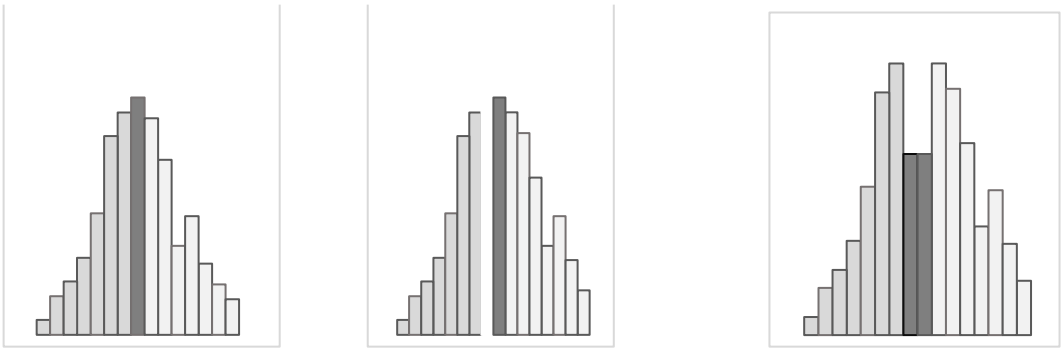
Où (i,j) représente les coordonnées du pixel dans l'image originale et $m \in [0,1]$ est le bit à intégrer. Chaque pixel est modifié au moins par 1. Le PSNR de l'image tatouée /originale est au moins 48.13 dB. Une grande qualité visuelle est donc garantie. L'entier a peut être pris comme le pic de l'histogramme pour maximiser la capacité.

De son côté, le décodeur peut extraire la donnée intégrée et restaurer l'image originale par simple lecture des valeurs des pixels ; c à d pour chaque pixel $Ji;j$:

- Si $Ji;j < a$ -1, il n'y a pas de donnée dissimulé dans le pixel et sa valeur original est $Ji;j$ +1
- Si $Ji;j \in \{a - 1; a\}$, le pixel est utilisé pour transporter une donnée secrète, sa valeur original étant a. le bit intégré est m=a- $Ji;j$.
- Si $Ji;j > a$, le pixel est inchangé dans l'intégration de donnée, sa valeur est $Ji;j$.

(a) règle de mappage des bins d'histogramme





(b)Histogramme avant (gauche) et après (droite) intégration des données

Figure I.4 : Dissimulation de donnée à base de décalage d'histogramme

Pour l'histogramme(b) , les bins inférieurs à a sont décalés à gauche par un 1 pour créer un espace vacant utilisable pour l'intégration. (a) loi de mappage des bins de l'histogramme. (b) histogramme avant (gauche) et après (droite) intégration de données Dans [33], Lee et al. Ont proposé une nouvelle méthode utilisant l'histogramme de l'image différence : l'histogramme de l'image différence est meilleur que celui de l'image intensité de pixel.

L'histogramme de l'image contient des points pics plus hauts, ce qui permet une meilleure exploitation de la corrélation spatiale.

[34] propose une construction générale pour la conception de RDH à base de HS. L'image est partitionnée en blocs non chevauchés de n bits. L'histogramme n-dimensionnel est généré en comptant la fréquence de chaque bloc et l'intégration se fait par modification de l'histogramme n-dimensionnel. Le bloc de pixels est divisé en deux ensembles disjoints : l'un porte la donnée dissimulée par expansion, en se basant sur une fonction d'intégration prédéfinie, l'autre est seulement décalé afin de créer un espace vacant pour assurer la réversibilité.

I.2.2.e RDH à base d'intégration de données adaptative

Le premier schéma RDH adaptatif basé sur DE et le tri, est celui proposé par Kamstra et Heijmans[35]. Le travail de Tian est amélioré en triant les pixels selon la variance locale avant l'intégration de donnée. Dans DE, la valeur moyenne de deux pixels dans une paire est invariante ; seule la différence est altérée. Ces invariants à l'intégration peuvent donc être utilisés par l'encodeur et le décodeur pour calculer la variance locale. Si la variance locale d'une paire est faible, la paire se situe dans une région lisse de l'image et peut probablement être étendu par une petite valeur de différence. Donc, le tri permet de réduire la taille de la carte d'emplacement qui est alors plus compressible, comparée à celle de DE.

Des travaux récents [36,37,38,39] ont montré que la combinaison d'une intégration adaptative, telle que le tri ou sélection de pixel, avec d'autres techniques telle que PEE améliore la performance considérablement. L'idée de base derrière l'intégration adaptative est d'utiliser les pixels lisse d'une image pour construire un meilleur media hôte ensuite d'implémenter l'intégration réversible en modifiant le media hôte seulement, qui ne représente qu'une partie de l'image originale. Par exemple, en ne considérant que les pixels situés dans les régions lisses d'une image et ignorant les régions bruitées, la prédiction d'erreur serait plus précise et son distribution d'histogramme plus pointue. Avec un tel histogramme (généré adaptative ment), la performance de PEE peut être améliorée [40,41].

I.3 Conclusion

À travers ce chapitre, nous avons résumé quelques techniques de dissimulation réversible de données (RDH) dans le domaine spatiale. Les techniques innovatives à la base de plusieurs autres extensions sont présentées, notamment : les RDH à base d'expansion de différence

(DE), à base d'expansion de l'erreur de prédiction (PEE) et à base de décalage d'histogramme. Les techniques mentionnées dans ce chapitre ne sont pas les seules ; plusieurs extensions et améliorations ont été, et sont toujours proposées, mais elles sont représentatives des solutions de base les plus sollicitées actuellement.

Chapitre II : FPGA et implémentations matérielles des techniques RDH

Les techniques de RDH présentées dans le chapitre précédent sont implémentées sous forme logicielle. C à d sous forme d'un code écrit en un langage de haut niveau (C, C++,….) exécuté par une CPU. Une exécution sur CPU à usage général est souvent moins rapide que celle sur un matériel dédié qui, en plus de rapidité, a l'avantage de consommation plus faible. Un ASIC est le meilleur matériel qu'on puisse réaliser mais qui a l'inconvénient imbattable du très grand cout. Une solution matérielle moins couteuse, offrant autant d'avantages que le VLSI en plus d'un avantage très sollicitant qu'est la reconfigurabilté, est l'FPGA. Dans ce chapitre on va d'abord présenter des notions de base communes aux technologies FPGA actuelles, ensuite on passe en revu quelques implémentations de techniques RDH sur matériel reconfigurable FPGA.

II.1 Définition :

FPGA (de *Field-programmable gate array*), qui peut être traduit en français par « réseau de portes programmables sur site » est un circuit intégré fait pour être (re)programmé (configuré) par l'utilisateur après sa fabrication en utilisant un langage informatique spécifique, donc sans modifier le matériel. Un FPGA se distingue ainsi d'un ASIC (*Application specific integrated circuit)*, un circuit intégré spécialisé dont le fonctionnement est figé une fois construit.

Les FPGA sont performants, rapides et reprogrammables, ce qui permet de faire des économies de temps et d'argent. Contrairement aux microprocesseurs traditionnels, un FPGA peut exécuter plusieurs opérations en parallèle. Le premier dispositif reprogrammable fut construit en 1984 par Altera. L'année suivante, les cofondateurs de Xilinx l'logiciel inventèrent le premier FPGA. Depuis les années 1990, les FPGA sont de plus en plus utilisés, notamment par de grandes compagnies comme Microsoft. Récemment Altera et Xilinx ont été respectivement rachetées par Intel et AMD.

II.2 Utilité d'un FPGA

Étant reprogrammable, un FPGA a l'utilité qu'on lui donne. Les FPGA sont notamment utilisés pour concevoir des prototypes d'ASIC, dans des systèmes en temps réel où le temps de réponse joue un rôle crucial et dans des projets où la configuration du matériel est susceptible d'être modifiée. On les retrouve dans des domaines et industries tels que les télécommunications, l'industrie automobile, l'aérospatial, la médecine, l'audio, la finance, l'informatique, la vidéo ou encore la sécurité. Il est par exemple possible de reprogrammer et de mettre à jour la caméra de recul de toute une série de voitures à distance et sans avoir à modifier physiquement les véhicules.

II.3 Architecture d'un FPGA

Chaque constructeur adopte une architecture qui lui est spécifique, mais il est possible de décrire une architecture générale qui englobe les fonctionnalités communes entre toutes les technologies FPGA actuelles.

D'un manière générale, un FPGA consiste en trois types de modules : les blocs d'E/S configurables (I/OB blocks), interconnexions et matrices d'interconnexions configurables (interconnecte Switch matrix) et les blocs logiques configurables (CLB). Les CLB sont organisés en un réseau bidimensionnel (lignes et colonnes) , donnant à l'utilisateur la possibilité d'arranger les interconnexions entres les blocs logiques.

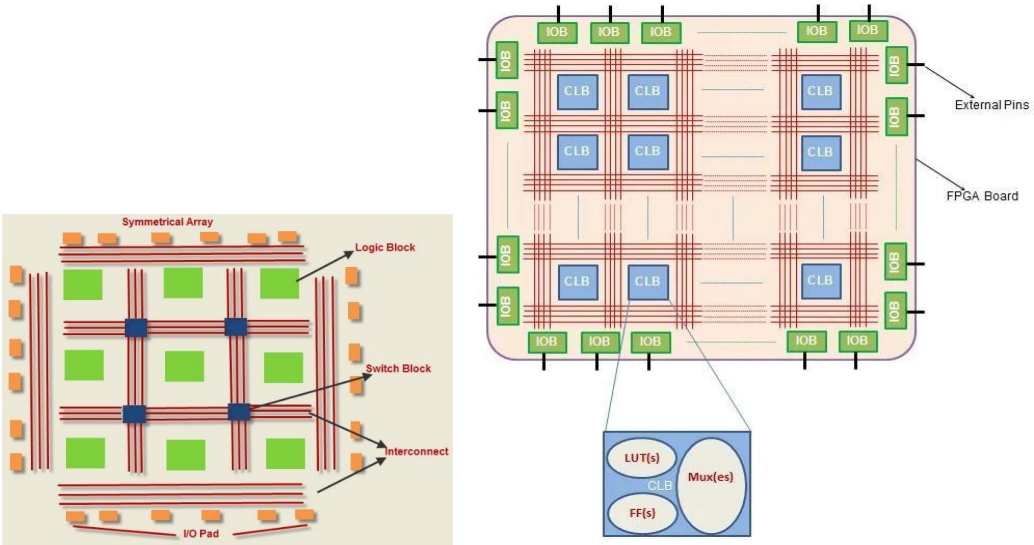


Figure II.1: Architecture générale d'un FPGA

II.3.1 Les blocs logiques configurable CLB:

C'est la structure configurable de base qui permet la configuration de toute fonction logique combinatoire ou séquentielle, avec un nombre de variables dépendant de la famille et la série du composant. Cette structure se répète sous forme de lignes et de colonnes sur la surface de l'FPGA. La figure II.2 représente un schéma général simplifié d'une CLB (non propre à un constructeur particulier)

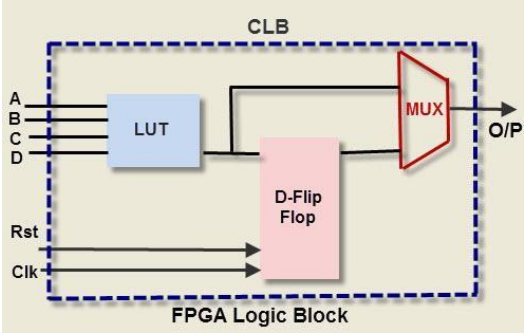


Figure II.2: schéma général simplifié d'un bloc CLB

Ces constituants principaux sont : un Multiplexeur, une bascule D et une table de transcodage LUT (Look-up table).

La table de transcodage LUT :

La LUT est une petite mémoire qui permet l'implémentation d'une fonction combinatoire. Le nombre de LUT dans un bloc logique ainsi que le nombre d'entrées dans une LUT font la différence entre les différentes familles d'FPGA. Par exemple, une CLB de l'FPGA Spartan 6 de Xilinx contient quatre LUT . Le nombre d'entrées d'une LUT limite le nombre de variables de la fonction qu'elle peut implémenter. Ce nombre varie entre e, 4, 6 et même 8 entrées.

Une LUT est une cellule mémoire SRAM 1-bit, qui peut être chargée par un 0 ou un 1 logiques.

L'une des valeurs contenues dans cette SRAM figure au niveau de la sortie selon les valeurs fournies aux lignes de contrôle , aux multiplexeurs et aux entrées qui fonctionnent comme lignes d'adresse. Le nombre d'entrées disponibles pour une LUT définit sa taille (profondeur de la SRAM) : une LUT à n-entrées est une mémoire SRAM à 2^n cellules 1-bit, suivie d'un multiplexeur 2^n à 1. La figure II.3 représente un exemple d'une LUT à 2 entrées, constituée d'une SRAM (4 cases)

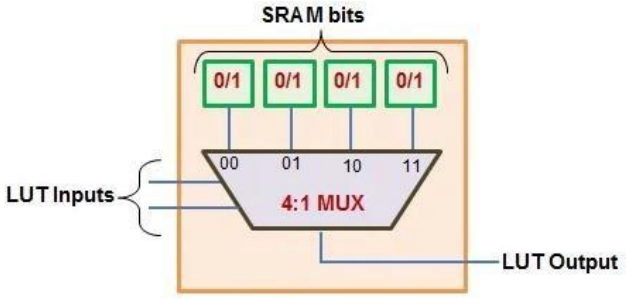


Figure II.3: Une LUT à deux entrées (4 cases mémoire)

Les bascules D permettent l'implémentation d'une logique séquentielle.

II.3.2 Les blocs E/S configurable:

C'est une unité configurable qui représente une interface de communication entre l'FPGA et les circuits externes. Elle permet d'assurer la compatibilité entre les différentes normes électriques des signaux d'entrée/sortie.

Un bloc IOB est constitué de plusieurs banques permettant une diversité de standards électriques, définis par la tension d'interface VCCO. Une banque peut avoir une seule VCOO, chaque banque peut avoir une VCOO différente. Il n'est possible de connecter que les signaux ayant le même standard électrique : la même VCOO est la condition de base.

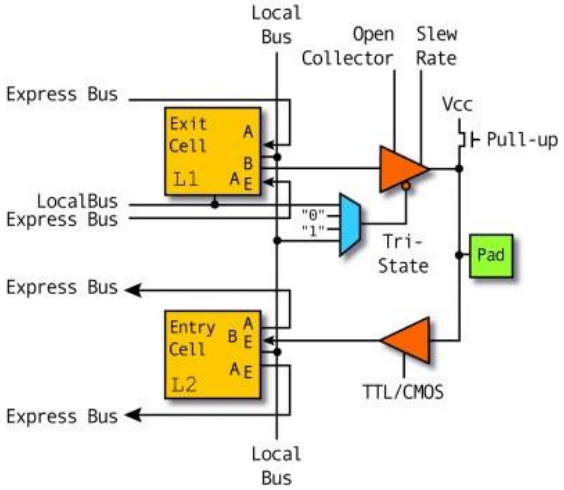


Figure II.4: un bloc IOB de Xilinx

La figure II.4 représente un bloc de base de l'architecture Xilinx. On y voit des buffers d'entrée et de sortie avec des contrôles de sortie trois-états et collecteur ouvert. Typiquement,

des résistances pull_up et pull_down sont placées au niveaux de sorties pour éviter l'utilisation des composants externes.
La polarité d'une sortie peut être programmée comme active haut ou bas et la dynamique de sortie est aussi programmable pour des temps de montée et de descente rapides ou lents.

Les bascules aux niveau des sorties permettent aux signaux de sortir directement aux pins sans retard. Aux niveau des entrées, les bascules réduisent le retard d'un signal avant d'atteindre une bascule.

II.3.3 Interconnexions programmable:

Dans la figure II.5 on peut voir une hiérarchie de ressources d'interconnexions. Les lignes longues sont utilisées pour connecter les CLBs distants en minimisant le retard de propagation.
Les lignes courtes servent à connecter les CLB voisins ou proches. Il est possible d'étendre une interconnexion en connectant deux lignes ensembles ; ceci se fait à travers des points d'interconnexion programmables et des matrices de commutations. Des buffers trois-états permettent de créer un bus en connectant un CLB et une longue ligne. Il y aussi des lignes dédiées aux signaux d'horloge, qui assurent un faible retard .

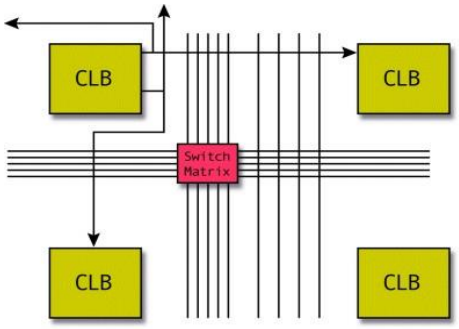


Figure II.5: interconnexions et matrices d'interruptions configurables

II.4 Les architectures matérielles pour les schémas de dissimulation de données

Généralement, les deux types de matériel les plus utilisés pour implémenter les techniques de DH sont les DSP (digital signal processors) et les FPGA. Des implémentations DSP ont été reportées dans [42,43]. Les stratégies de programmations de ces plateformes font que le parallélisme possible de l'algorithme n'est pas exploité. bien qu'il existe des

solutions pour implémenter le parallélisme, le traitement multi-canal reste une tache indirecte très exigeante.

L'implémentation sur FPGA semble une option intéressante, le parallélisme inhérent dans l'FPGA facilite la tâche. Dans [34 44] un système DH pour la parole est proposé. Le système utilise les techniques DH pour transmettre des composantes parole hautes fréquences et utilise un logiciel d'application associé à une FFT implémentée sur une plateforme matérielle.

L'utilisation du logiciel d'application limite la performance en termes de vitesse. Dans [35 45], on propose un système DH utilisant des image comme media hôte. Les résultats de performance sont présentés de manière superficielle, mais l'auteur réclame l'utilisation du système en temps réel. Une implémentation FPGA d'un algorithme de tatouage de vidéo est présenté dans [36 46] dont la performance est comparée à celle d'une DSP ; les résultats indiquent la supériorité de l'FPGA en termes de vitesse, de consommation et de cout. Dans [37 47], des implémentations matérielles de techniques stéganographiques pour document, images et vidéos sont présentées. Selon les auteurs, les résultats montrent qu'un fonctionnement en temps réel est garanti. [38 48] propose une architecture matérielle pour un système DH audio 160 fois plus rapide que son implémentation logicielle. Dans [49] on propose une implémentation FPGA d'un RHD visant les images médicales qui assure une grande capacité d'intégration et un minimum de distorsion. Dans [50] implémente un RDH à base de transformé DCT et étalement de spectre. Les auteurs réclament une Vitesse allant à 79.7025 Mega byte/s et 88.46% moins de consommation que les travaux similaires. Une RDH utilisant une génération de clé chaotique est présentée dans [51] ou le débit réseau est amélioré de 99%, le retard est diminué de 60%.

II.5 Conclusion

Dans ce chapitre, nous avons présenté des définitions de base concernant les FPGAs ainsi qu'un sondage court sur les implémentations matérielles sur FPGA des techniques RDH. Bien que les littératures rapportent peu de détails et de travaux, l'utilité d'une implémentation FPGA dans l'amélioration de la vitesse et évidente .

Chapitre III: implémentation du système RDH à base de classification de pixel et d'expansion d'erreur de prédiction

Les techniques RDH à base de DE, de PEE et de HS ont prouvé leur efficacité et flexibilité pour la dissimulation des données. Dans le chapitre I on a mentionné que plusieurs extensions réussies, combinant des idées des trois méthodes, sont proposées. Dans ce chapitre on présente un algorithme de RDH à base de classification de pixels (PVO : pixel value ordering) et d'expansion de l'erreur de prédiction(PEE). C'est l'algorithme qu'on va implémenter utilisant le langage VHDL et les outils de conception Aldec ACTIVE HDL.

III.1 L'algorithme :

III.1.1 prédiction de pixel à base de PVO et intégration de données réversible :

Dans la technique RDH à base de PEE, l'histogramme de l'erreur de prédiction est partagé en deux régions appelés région interne et région externe. Les erreurs de prédiction dans la région interne sont étendues afin d'intégrer des données, tandis que celles dans la région externe sont décalées de manière à garder les régions interne et externe séparées l'une de l'autre après l'intégration de donnée. Si un ou deux bins seulement existent dans la région interne, c-à-d la modification maximale de la valeur d'un pixel dans l'intégration est de 1, alors, on peut dériver la valeur attendue de la modification (en norme L2) de l'image hôte comme étant 0.5Nc+Ns, où Nc est la capacité d'intégration (EC) et Ns est le nombre de bits décalés [19]. Il est clair que pour un une capacité Nc donnée, il y aura moins de modification pour moins de Ns. Par conséquent, la proportion des bits décalés

$$P_{décalé} = \frac{\#\{pixel\ décalé\}}{\#\{pixel\ agrandis\ ou\ décalé\}} = \frac{N_s}{N_c+N_s} \qquad \text{(III.1)}$$

Est une mesure du RDH à base de PEE pour évaluer la performance de l'intégration. # représente le nombre cardinal d'un ensemble. Plus petit Pshifted implique moins de modification de l'image originale et une meilleure performance.

III.1.2 Le nouveau prédicteur pour PEE:

L'avantage par rapport à quelques prédicateurs existants peut être démontré utilisant la mesure P_{shif} . D'abord, l'image hôte est divisée en blocs non-chevauchés de tailles identiques. Pour un bloc X, contenant n pixels, trier ses valeurs (X_1, ... , X_n) en ordre ascendant pour obtenir ($X_{\sigma(1)}$, ... , $X_{\sigma(n)}$) ou σ:{1,…..,n} → {1,……,} est une correspondance un-à-un tel que : $X_{\sigma(1)} \leq \cdots$ ... . . $\leq X_{(n)}$ $et\ \sigma(i) < \sigma(j)\ si\ X_{\sigma(i)} = X_{\sigma(j)}$ et i<j. Alors on utilise la deuxième valeur plus grande, $X_{(n-1)}$ pour prédire le maximum $X_{\sigma(n)}$. L'erreur de prédiction correspondante est : $PE_{max} = X_{(n)} - X_{(n-1)}$

Pour un bloc de taille 2x2, l'histogramme de PE_{max} de l'image lena est donné par la figure III.1.

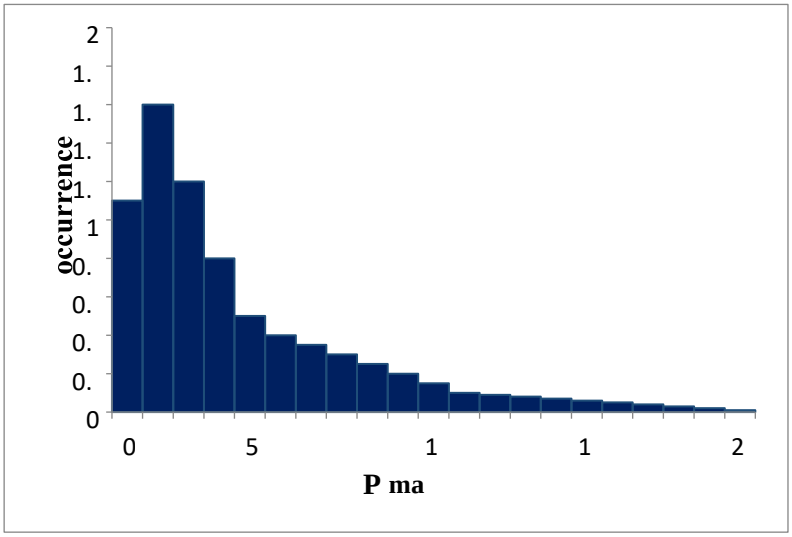


Figure III.1 : histogramme de l'erreur PEmax pour l'image lena 256x256 niveaux de gris

En se basant sur se prédicteur et la PEE, l'intégration et l'extraction de données réversible peut être décrite comme suit (figure III.2) :

Comme le premier bin 1 (bin correspondant à PEmax=1) est souvent le pic de l'histogramme, on considère ce bin comme région interne et les bins supérieurs à 1 comme région externe. Dans cette situation, afin d'intégrer les données par PEE, l'erreur PEmax est modifiée comme :

$$\widetilde{PE}_{max} = \begin{cases} PE_{max} & si \;\; PE_{max} = 0 \\ PE_{max} + b & si \;\; PE_{max} = 1 \\ PE_{max} + 1 & si \;\; PE_{max} > 1 \end{cases} \qquad \text{(III.2)}$$

Où $b \in \{0,1\}$ est le bit de donnée à intégrer, par conséquent, $X_{\sigma(n)}$ est modifié par :

$$\tilde{X} = X_{\sigma(n-1)} + \widetilde{PE}_{max} = \begin{cases} X_{\sigma(n)} & si \;\; PE_{max} = 0, \\ X_{\sigma(n)} + b & si \;\; PE_{max} = 1 \\ X_{\sigma(n)} + 1 & si \;\; PE_{max} > 1, \end{cases} \qquad \text{(III.3)}$$

En gardant les autres bits $X_{\sigma(1)}, \dots \dots, X_{\sigma(n-1)}$ inchangés. Donc, les valeurs X tatouées sont $(Y_1;\dots, \; Y_n)$ où $Y_{\sigma(n)} = \tilde{X} \;\; et \;\; Y_i = X_i \; pour \; chaque \; i \neq \sigma(n)$

Dans la procédure ci-haut, comme $X_{\sigma(n)}$ est soit incrémenté soit inchangé, l'ordre de la valeur du pixel (c-à-d le mapping sigma) reste inchangé. Donc, pour un bloc tatoué de valeurs $(Y_1;\dots, Y_n)$, le calcul de l'erreur $\widetilde{PE}_{max} = Y_{\sigma(n)} - Y_{\sigma(n-1)}$, nous permet l'extraction de la donnée intégrée ainsi que l'image hôte à travers la procédure suivante :

- Si $\widetilde{PE}_{max} = 0$ alors, le bloc est inchangé lors de l'intégration de donnée et sa valeur original est $(Y_1;\dots, Y_n)$
- Si $\widetilde{PE}_{max} \in \{1,2\}$ le bloc a été étendu pour recevoir une donnée cachée lors de l'intégration. Le bit de donnée intégré est $b = \widetilde{PE}_{max} - 1$ et sa valeur originale est $(X_1;\dots, X_n)$ où :
  $Y_{\sigma(n)} = X_{\sigma(n)} \;\; et \;\; Y_i = X_i \; pour \; chaque \; i \neq \sigma(n)$
- Si $\widetilde{PE}_{max} > 2$, le bloc a été décalé lors de l'intégration de donnée et sa valeur originale est $(X_1;\dots, X_n)$ où
  $X_{\sigma(n)} = Y_{\sigma(n)} - 1 \;\; et \;\; Y_i = X_i \; pour \; chaque \; i \neq \sigma(n)$

Particulièrement, pour l'intégration à base de PVO ci-haut, la quantité Pshifted définie en (eq III.1) peut être reformulée comme : $P_{décalé} = \frac{\#\{PE_{max}>1\}}{\#\{PE_{max}\geq 1\}}$ où PE_{max} est l'erreur de prédiction de l'image originale défine en (eq III.2).

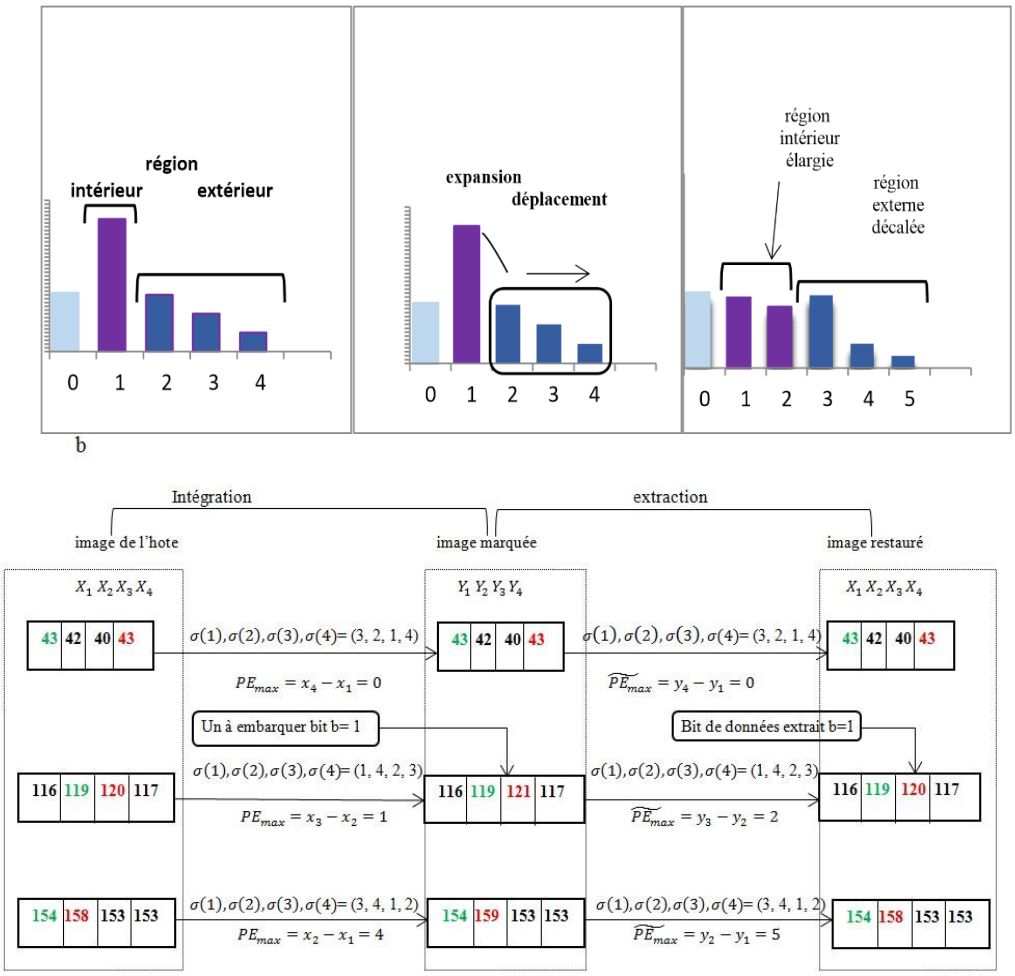


Figure III.2 : intégration PEE à base de PVO (modification du max)

(a) bin 0 n'est pas considéré lors de l'intégration, la région interne étant le bin 1, la région externe formée par le reste des bins. (b) le maximum et deuxième maximum sont colorés en rouge et vert respectivement. Lors de l'intégration de donnée, le bloc haut est inchangé, le bloc de milieu est étendu pour porter b-1, le bloc de bas est décalé.

Cette intégration réalise une meilleure performance en utilisant des blocs de taille plus grande ; c'est une observation raisonnable sachant qu'un bloc de taille plus grande implique une meilleure exploitation de la redondance spatiale dans l'image naturelle.

Il est aussi possible d'intégrer une donnée par modification du minimum du bloc trié. Plus spécifiquement, le deuxième plus petit élément $X_{\sigma(2)}$ peut être utilisé pour prédire le minimum

$X_{\sigma(1)}$ l'intégration PEE est alors implémentée par modification de l'erreur correspondante $PE_{min} = X_{\sigma(1)} - X_{\sigma(2)}$ (figure III.3). Dans ce cas l'erreur est modifiée à :

$$\widetilde{PE}_{min} = \begin{cases} PE_{min} & si \quad PE_{min} = 0, \\ PE_{min} - b & si \quad PE_{min} = -1, \\ PE_{min} - 1 & si \quad PE_{min} < -1, \end{cases} \qquad \text{(III.4)}$$

Où $b \in \{0; 1\}$ est le bit de donnée à inclure.

En prenant en compte le maximum et le minimum, deux bit au plus peuvent être inclus dans un seul bloc, ce qui augmente la capacité efficacement. Par exemple, quand les deux conditions $X_{(n)} - X_{\sigma(n-1)} = 1 \quad et \quad X_{\sigma(1)} - X_{\sigma(2)} = -1$ sont satisfaites, on peut inclure deux bits dans un bloc X. (le bloc en milieu dans la figure III.2). Dans la section suivante on détaille la procédure d'intégration de donnée.

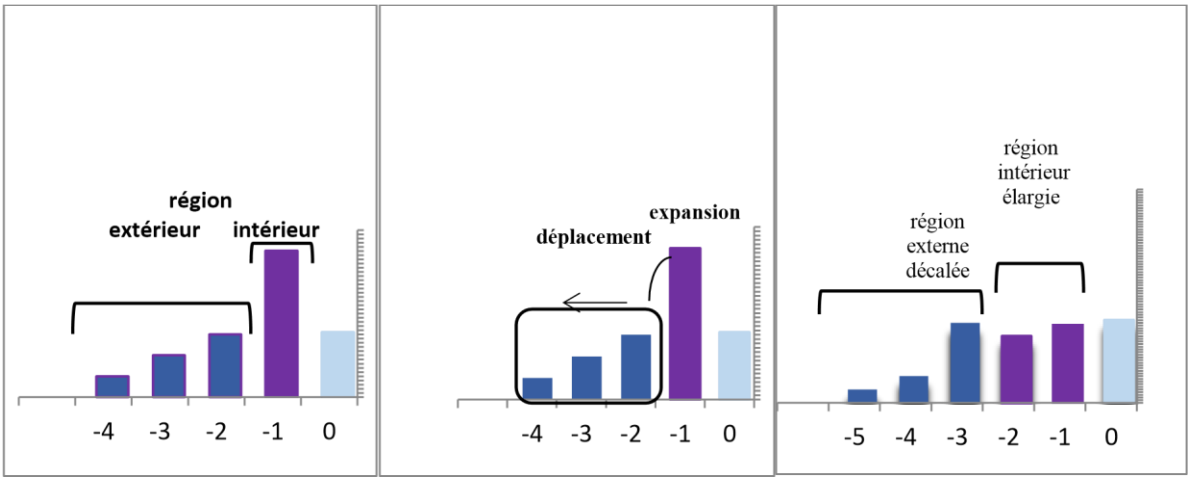


Figure III.3 : intégration de donnée PEE à base de PVO (modification du minimum)

L'utilisation des blocs lisses est plus favorable pour la RDH [19]. Les blocs lisses sont donc utilisés pour intégrer une donnée, laissant les blocs texturés inchangés. La complexité du bloc X est mesurée par la différence$\boldsymbol{X_{(n-1)} - X_{\sigma(2)}}$. Un bloc est déclaré lisse si sa complexité est inférieure à un seuil prédéterminé T. par exemple, dans la figure III.2.b, les complexités sont : $\boldsymbol{X_1}$-$\boldsymbol{X_2}$=43-42=1, $\boldsymbol{X_2}$-$\boldsymbol{X_4}$=119-117 et $\boldsymbol{X_1}$-$\boldsymbol{X_4}$=154-153=2 respectivement. On note que la quantité $\boldsymbol{X_{(n-1)} - X_{\sigma(2)}}$ .est invariante après intégration de donnée. L'encodeur et le décodeur peuvent trouver les mêmes blocs lisses. Ceci garantit la réversibilité de l'intégration PEE à base de PVO.

III.1.2.a Procédure d'intégration des données :

Étape 1 (partitionnement de l'image et tri) : diviser l'image hôte en k blocs non-chevauchés $\{\boldsymbol{X_1}, \ldots. \boldsymbol{X_k}\}$ tel que chaque Xi contient n pixels. Trier les valeurs de pixels de Xi par ordre croissant pour obtenir $(\boldsymbol{X_1}, \ldots. \boldsymbol{X_n})$ (Cette notation indique maintenant les valeurs triées au lieu de valeurs originales).

Étape 2 (construction de la carte d'emplacements et compression) :

La carte d'emplacement du débordement (positif et/ou négatif) est construite. Pour un bloc $\boldsymbol{X}$, si l'une des deux situations a lieu :

$$\boldsymbol{X_n - X_{n-1} \geq 1 \; et \; X_n = 255}$$

$$\boldsymbol{X_1 - X_2 \leq -1 \; et \; X_1 = 0}$$

C'est un bloc exceptionnel qui pourrait générer un débordement après expansion ou décalage en PEE. Dans ce cas on met : LM(i)=1, sinon LM(i) =0. LM est une séquence binaire de longueur k. la séquence LM est compressée (sans pertes) pour donner une séquence de longueur l_{clm}. On remarque que, souvent, les blocs exceptionnels sont très peu nombreux, ce qui donne une séquence avec quelques 1 seulement. La compression peut donc réduire la taille de LM efficacement.

Étape 3 (intégration de données) :

Intégrer les bits de données dans l'image. Pour $i \in \{1, \ldots, k\}$

- Si LM(i)=1, risque de débordement: ne rien faire à $\boldsymbol{X_i}$
- Si LM(i)=0 et $\boldsymbol{X_{n-1} - X_2 \geq T\ X_i}$ est un bloc texturé et là aussi on ne fait rien avec$\boldsymbol{X_i}$
- Si LM(i)=0 et $\boldsymbol{X_{n-1} - X_2 \leq T}$: pas de débordement et $\boldsymbol{X_i}$ est lisse. Dans ce cas le maximum et le minimum vont être décalés ou étendus selon les cas suivants :

 Pour le maximum :

- Si $\boldsymbol{X_n - X_{n-1} > 1, X}$, est décalé à $\boldsymbol{X_{n+1}}$
- Si $\boldsymbol{X_n - X_{n-1} =}$, $\boldsymbol{X_n}$ est étendu à Xn+b, où $b \in \{0;\ 1\}$ est le bit de donnée à intégrer.
- Si $\boldsymbol{X_n - X_{n-1} =}$, $\boldsymbol{X_n}$ ne serait pas modifié.

 Pour le minimum

- Si $\boldsymbol{X_1 - X_2 < -, X_1}$ est décalé à $\boldsymbol{X_1 - 1}$
- Si $\boldsymbol{X_1 - X_2 = -, X_1}$ est étendu à $\boldsymbol{X_1 - b'}$ où $b' \in \{0;\ 1\}$ est autre bit de donnée à intégrer.
- Si $\boldsymbol{X_1 - X_2 =}$, $\boldsymbol{X_1}$ ne serait pas changé.

Cette étape s'arrete si tous les bits de donnée sont intégrés, dans ce cas on note kend l'indice du dernier bloc tatoué.

Étape 4 (intégration des informations auxilliaire et de la LM) : on construit une séquence binaire S_{LSB} formée à partir des bits LSB des premiers $16+2\lceil log_2 N \rceil + l_{clm}$ pixels de l'image qu'on va sauvegarder. Avec N=nk et $\lceil . \rceil$ est la fonction plafond. Ensuite remplacer ces bits par l'information auxilliaire suivantes et la séquence LM compressée :

- La taille du bloc n1 (4 bits) et n2 (4 bits) avec n = n1 x n2
- Le seuil de complexité du bloc T (8 bits),
- La position du dernier bloc tatoué ($\lceil log_2 N \rceil\ bits$)
- La longueur de la carte d'emplacement comressée l_{clm} ($\lceil log_2 N \rceil\ bits$)

Finalement, intégrer la séquence S_{LSB} dans les bloc non-tatoué restants $\{\boldsymbol{X_{kfin+1}}, \ldots\ \boldsymbol{X_k}\}$ selon l'étape 3.

Dans la procédure ci-haut, à part quelques blocs portant l'information auxilliaire et la carte d'emplacement, seule le maximum et le minimum sont changés à 1 dans un bloc, le reste de pixels ne change pas. Lorsque la taille du bloc est $n \geq 4$, le PSNR de l'image tatouée conntre l'image originale est au moins $10log_{10}\frac{255^2 n}{2} \geq 10log_{10}(2.255^2) = 51.14dB$

Avec cette méthode, la qualité visuelle de l'image est bien garantie.
Le seuil T est pris comme la plus petite valeur positive permettant d'intégrer les données. Il peut être déterminé de manière itérative.

III.1.2.b Procédure d'extraction de données :

L'extraction des données intégrés se fait de la manière suivante :

Étape 1 (extraction des informations auxilliaires et LM) : on extrait les bits LSB des premiers

16+2$\lceil log_2 N \rceil$ pixels de l'image tatouée : c'est l'information auxilliaire (n1,n2,T,kend et lclm).ensuite lire lclm pixels suivants : c'est la LM compressée. On décompresse cette séquence pour reformer la carte d'emplacement.

Étape 2 (extraction de la séquence S_{LSB}) : de la même manière faite dans la procédure d'intégration, on divise l'image tatouée en k blocs non-chevauchés $\{\boldsymbol{X_1}, \ldots. \boldsymbol{X_k}\}$ tel que chaque bloc Xi contient n pixels. À partir des blocs $\{\boldsymbol{X_{kfin+1}}, \ldots. \boldsymbol{X_k}\}$ on extrait SLSB définie dans l'étape d'intégration 4 en même temps on restore l'image. Pour un bloc Xi (i> $\boldsymbol{k_{fin}}$) dont les valeurs triées par ordre ascendant sont (Y_1 ;…, Y_n) :

- Si $si\ L(i) = 0\ et\ y_{n-1} - y_2 < T$ ○

  Pour le maximum yn :

  - Si $y_n - y_{n-1} > 2$, il n'y a pas de donnée dissimulée et la valeur originale de y_n est y_{n-1}
  - Si $y_n - y_{n-1} \in \{1,2\}$ la donnée cachée est $b = y_n - y_{n-1} - 1$ et la valeur originale de y_n est y_n – b.

- Si $y_n - y_{n-1} = 0$ il n'y a pas de donnée cachée et la valeur originale de y_n est y_n.

  ○ Pour le minimum y_1 :

- Si $y_1 - y_2 < -2$ il n'y a pas de donnée dissimulée et la valeur originale de y_n est y_{n+1}
- Si $y_1 - y_2 \in \{-1, -2\}$; la donnée cachée est $b' = y_2 - y_1 - 1$ et la valeur originale de y_n est – b'.
- Si $y_1 - y_2 = 0$, il n'y a pas de donnée cachée et la valeur originale de y_1 est y_1 .

 Sinon, il n'y a pas de donnée cachée et les valeurs originales de y_n. , y_1 sont y_n. et y_1.

De plus, dans les cas, pour chaque $j \in \{2, \ldots, n-1\}$, la valeur originale de y_i est elle-même car elle n'a pas subi de changement lors de l'intégration de donnée. Cette étape s'arrête si SLSB est extraite.

Étape 3 (extraction de donnée et récupération de l'image) : remplacer les bits LSB des premiers $16+2\lceil log_2 N \rceil + l_{clm}$ pixels de l'image par la séquence SLSB extraite dans l'étape 2, ensuite utiliser la même méthode étape 2 pour extraire la donnée dissimulée dans les blocs $\{\boldsymbol{X_1}, \ldots, \boldsymbol{X_{kfin}}\}$. Finalement, la donnée intégrée est extraite et l'image originale est récupérée.

III.2 Conclusion :

Grace à la technique de masquage réversible de données (RDH) a base de PEE et PVO l'excellente propriété selon laquelle l'image d'origine peut être récupérée sans perte après l'extraction des données intégrée, tout en protégeant la confidentialité de contenu de l'image. C'est une technique très promettante pour une implémentation matérielle.

Chapitre IV : Conception et Implémentation VHDL de l'algorithme d'intégration PEE à base de PVO

Dans ce dernier chapitre, nous présentons l'aspect conception et programmation VHDL, pour l'implémentation de l'algorithme choisi (chapitre III). On présente le raisonnement suivi pour aboutir à des schématiques généraux ainsi que les descriptions VHDL des blocs de base et la simulation dans l'outil ALDEC Active HDL.

IV.1 Format des données :

La conception commence par définir le format de codage pour les données. L'algorithme vise des images en niveaux de gris 8_bits. Le chemin de données principal est donc choisi comme des entiers non signés 8bits et le type VHDL correspondant est unsigned(7 downto 0).

IV.2 Conception :

La première étape à faire pour implémenter l'algorithme est l'identification des fonctions de base et le partitionnement en blocs de constructions. Cette étape doit aboutir à un schématique dans lequel chaque bloc réalise au moins une fonction ou bien des fonctions homogènes ou étroitement liées. Pour la procédure d'intégration, le schéma déduit est illustré par la figure IV.1.

IV.2.1 Codage :

La figure IV.1 montre la partie codage (intégration réversible de données) du système de tatouage.

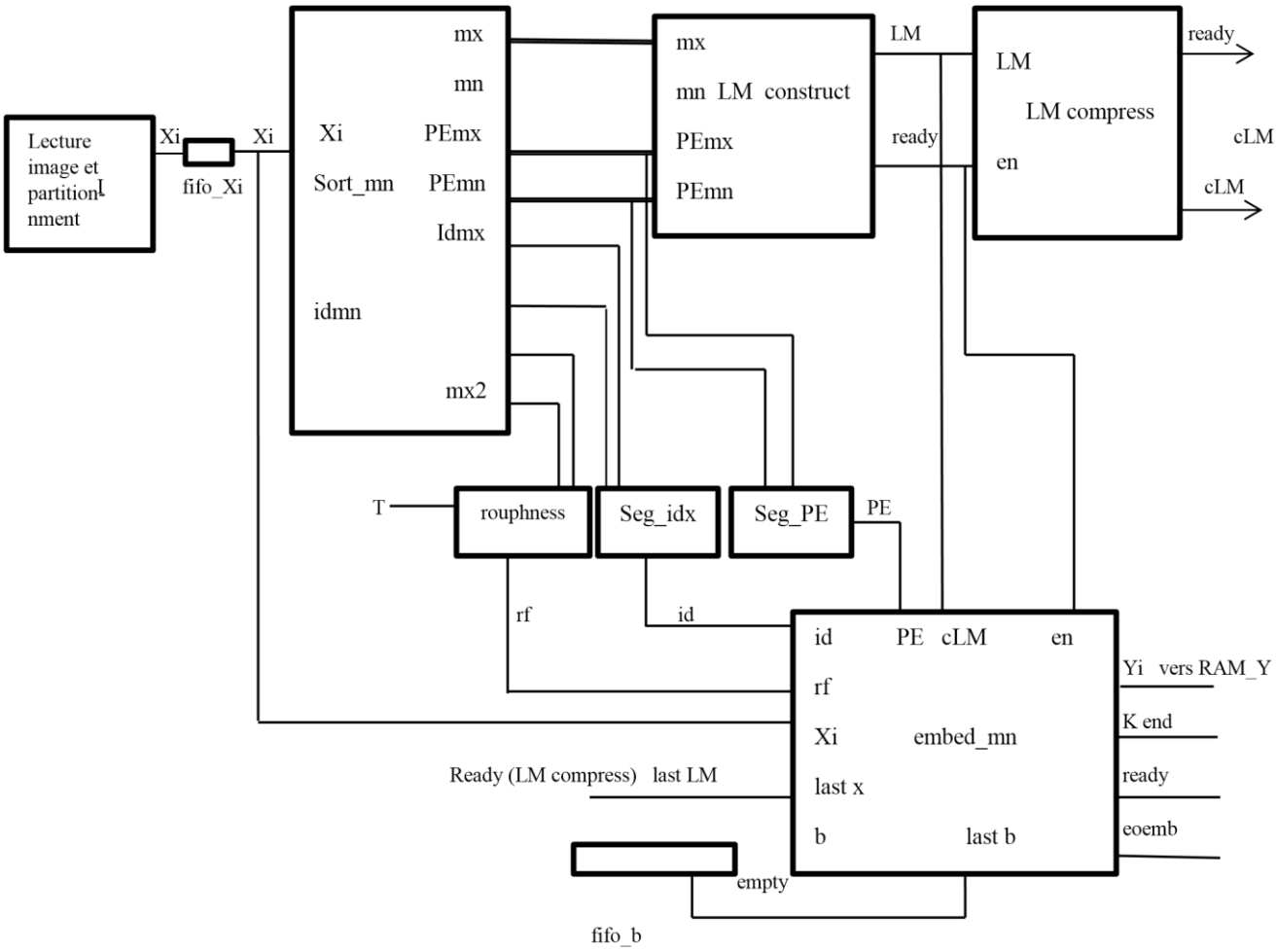


Figure IV.1 : Schéma bloc de la partie codage PVO

Les blocs principaux identifiés sont : Les blocs Im_read , Im_blocin, Im_blocout et Im_write sont les blocs responsables de la lecture de l'image , le partitionnement en blocs de 4 x 4,

reconstruction de l'image à partir des bloc 4x4 et enfin l'écriture de l'image dans la mémoire, respectivement.

Les autres blocs : sort_mn, LM_construct, LM_compress, Xi_rough, Embed_mn, Seq_idx et Seq_PE. On va détailler la fonction, la conception et la simulation de chaque bloc dans les sections suivantes.

Bloc Sort_mn :

Ce bloc implémente le tri ascendant des 4x4 valeurs de pixels dans le bloc Xi. Plusieurs architectures sont possibles pour concevoir le tri mais on a choisi le tri bitonique, qu'on va détailler ci-après.

Tri bitonique :

Comme tout réseau de tri, le tri bitonique de Batcher prend en entrée une suite (a1,…..an) et produit en sortie la suite (b1,…..bn) qui est la même suite, mais triée. L'algorithme de tri bitonique est un algorithme récursif basé sur un réseau particulier appelé *fusionneur*. Un *fusionneur* à *n* entrées a la propriété que si les entrées sont formées de deux suites triées a1,….an/2) et an/2 +1,…….an) , alors la sortie (b1,…..bn) est une suite triée.

Un réseau de tri bitonique à *n* entrées est composé de deux réseaux bitoniques à *n*/2 entrées opérant en parallèle, chacun sur la moitié des entrées, suivi d'un fusionneur. En développant cette définition récursive, on peut voir un réseau de tri bitonique comme composé uniquement de fusionneurs, chacun ayant pour tâche de fusionner deux suite triées de taille moitié en une seule suite triée. On suppose ici et dans la suite que *n* est une puissance de 2. Il est utile de numéroter les fils de 1 à *n*. Un comparateur entre le fil *i* et le fil *j* est noté [i,j]. Par exemple, les huit premiers comparateurs du réseau ci-dessus sont [i :i+1] pour i=1,3,5,7,9,11,15.

Un *fusionneur* est composé de deux parties. La première est un *réseau de comparateurs*, et la deuxième un ensemble de réseaux appelés *séparateurs*. Un réseau du premier type peut d'ailleurs être vu comme un séparateur où on a renversé les fils d'entrée de la deuxième moitié.

Un *réseau de comparateurs* à *n* entrées est formé des comparateurs

[1 :n], [2 :n-1],……,[i :n-i+1],…….,[n/2 :n/2 :n]

Un *séparateur* à *n* entrées est formé des comparateurs

[1 :n/2+1], [2 :n/2+2],……,[i :n/2+i],…….,[n/2 :n]

Le fusionneur d'ordre *n* est composé d'un comparateur d'ordre *n*, puis d'une suite de groupes de séparateurs, d'abord deux d'ordre *n/2*, puis quatre d'ordre *n/4* etc. L'entrée du fusionneur est une suite dont les deux moitiés sont triées, la sortie est composée de deux suites bitoniques dont la première est plus petite que la seconde. Chacun de ces suites bitoniques entre dans une cascade de séparateurs dont le résultat est la suite triée. La concaténation de ces deux suites est elle-même triée.

La figure IV.2 illustre un réseau de tri bitonique. Les s0,…s9 représentent les étages (constitués de séparateurs et de fusionneurs), sb0….sb9 sont les signaux d'interconnexion entres les étages.

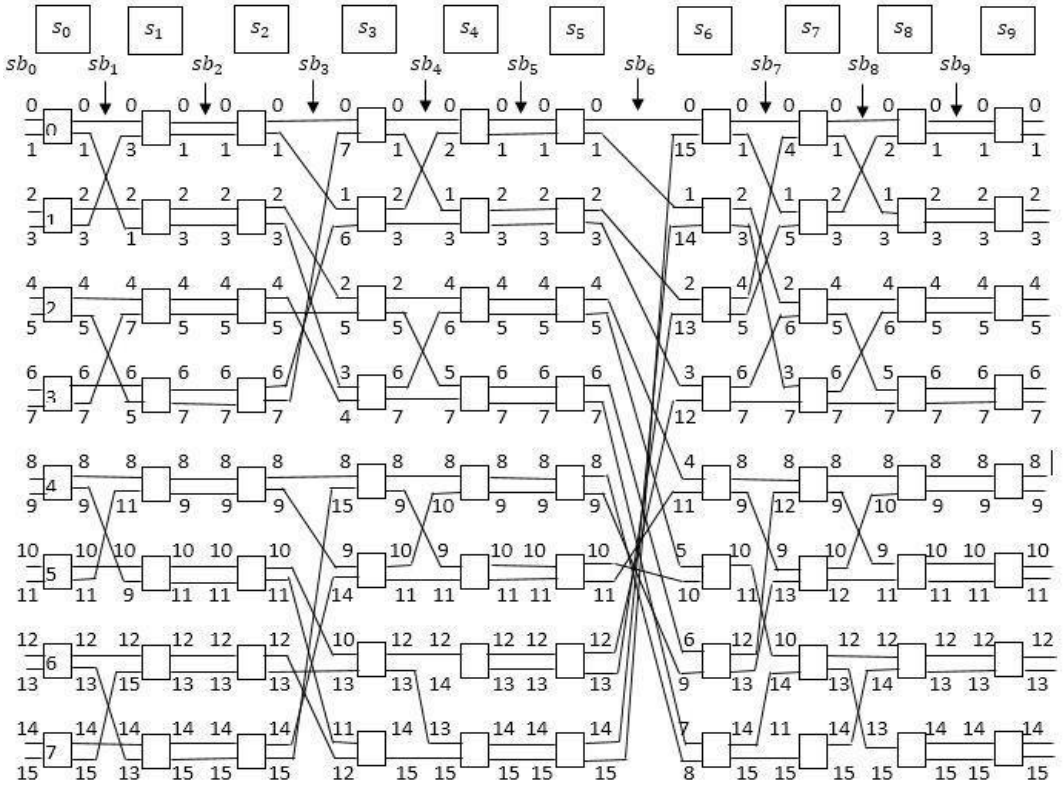
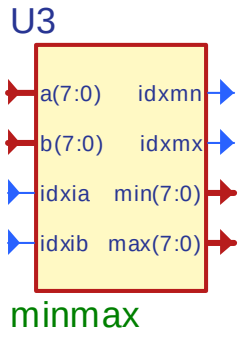


Figure IV.2 :schéma de tri bitonique

Bloc minmax

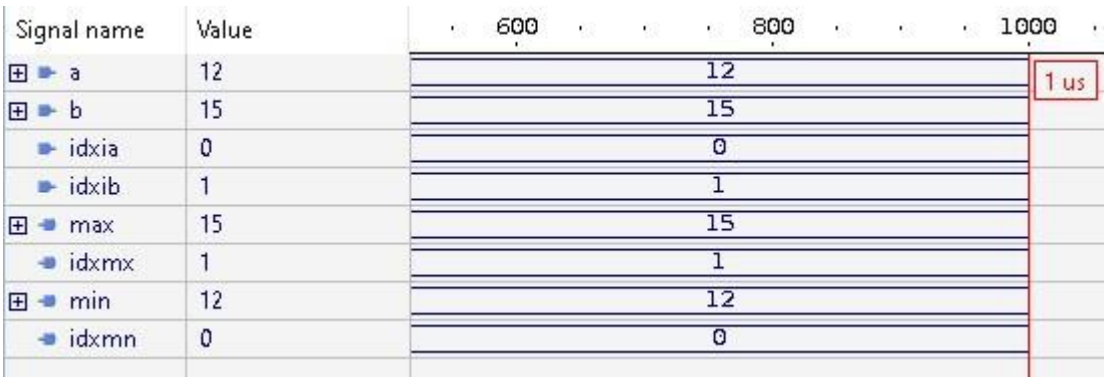
Le tri bitonique adopté est construit autour d'un bloc de base (minmax) qui réalise le tri ascendant entre deux entrées : pour deux entrées a et b, les sorties max = a et min=b si b <a, sinon max = b et min=a. Les fractions de code ci-après montrent le process et la description d'entité VHDL correspondants :

```
entity maxmin is
   Port ( a :       in  unsigned(7 downto 0);
b :       in  unsigned(7 downto 0);
idxia:   in  integer;
idxib:   in  integer;
idxmx: out integer;
idxmn: out integer;
            min:    out unsigned (7 downto 0
            max :     out unsigned (7 downto 0));
end maxmin;
```




```
process(a,b)
begin
if (a> b) then
min<=      b;
max <=     a;
 idxmn <= idxib;
 idxmx <= idxia;
else
 min <=    a;
 max<=     b;
 idxmn <= idxia;
 idxmx <= idxib;
end if;
end process;
```

Les entrées idxia et idxib de type integer, permettent d'associer un indice à chaque entrée. Les sorties idxmx et idxmn de type integer, fournissent les indices des arguments maximum et minimum, respectivement. Un exemple de simulation est donné ci-après

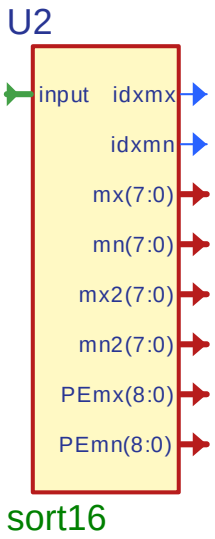


Bloc sort16

On utilise le bloc minmax comme composant (component) dans une hiérarchie plus haute pour réaliser le tri d'une liste d'entrée Xi constituée de 16 éléments (16 pixels d'entrée, type unsigned(7 downto 0) chacun). La sortie finale n'est pas la liste triée mais plutôt : les valeurs maximale (mx), minimale (mn), les deuxième valeurs maximale et minimale (mx2,mn2), les indices associés aux valeurs mx et mn (idxmx, idxmn) et les sorties différences PEmx et PEmn.

L'algorithme de tri bitonique représenté par le schématique de la figure (IV.2) est implémenté sous forme d'une architecture de type structural, où le composant minmax est déclaré et instancié.

```
entity sort16 is
port(input : in bus_array;
idxmx: out integer;--int_array(0 to 1);
idxmn: out integer;--int_array(0 to 1);
mx,mn: out unsigned(7 downto 0);
mx2,mn2: out unsigned(7 downto 0);
PEmx,PEmn: out signed(8 downto 0));
end sort16;
```



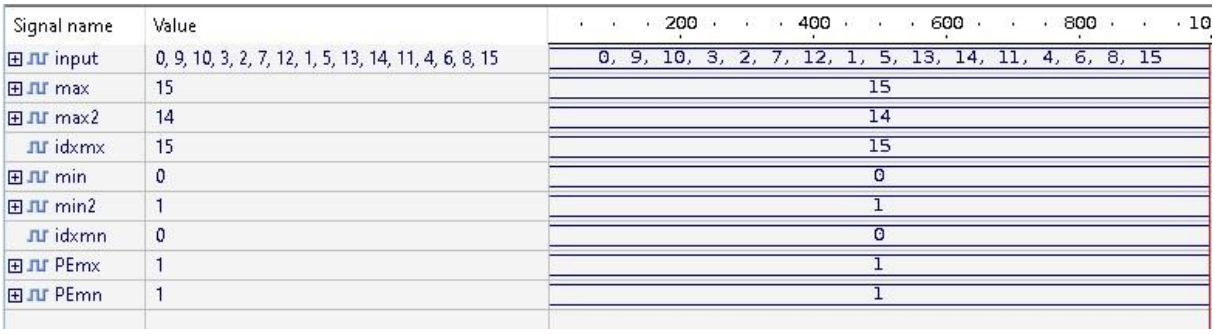
Le circuit de tri est constitué de 9 étages. Chaque étage est représenté comme un ensemble de 16 blocs minmax organisés en une colonne, prend ses entrées à partir de l'étage précédent et donne ses sorties comme entrées à l'étage suivant.

Pour l'implémentation VHDL, l'instruction for..generate est utilisée pour générer toutes les instances du bloc minmax avec les indices nécessaires pour représenter la figure III.*. La partie de code suivante représente la boucle *for…generate* correspondante au dernier étage ainsi que les lignes des sorties :

```
gen_reg9: for i in 0 to 7 generate   u16:
minmax port map(stage9_op(2*i),
  stage9_op((2*i)+1),idx9_op((2*i)),idx9_op((2*i)+1),
idxo_op((2*i)),idxo_op((2*i)+1),  stageo_op(2*i),  stageo_op((2*i)+1));
  end generate ;
mx <=stageo_op(15); idxmx <=  idxo_op(15);
mx2 <=stageo_op(14);

mn <=stageo_op(0); idxmn <=  idxo_op(0);    mn2 <=stageo_op(1);

PEmx <=conv_signed(stageo_op(15)-stageo_op(14),9);
PEmn <=conv_signed(stageo_op(1)-stageo_op(0),9);
```

Les sorties PEmx et PEmn sont de type signé car on a besoins de leur signe.

Une simulation est réalisée pour une entrée de test input =(0,9,10,3,2,7,12,4,5,13,14,11,4,6,8,15)

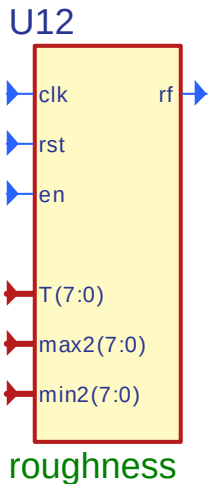


Bloc roughness :

Ce bloc décide de la complexité du bloc Xi trié. Si max2-min2<T alors le blocs Xi est déclaré lisse avec rf=1, sinon rf=0. Le seuil T est prédéterminé.

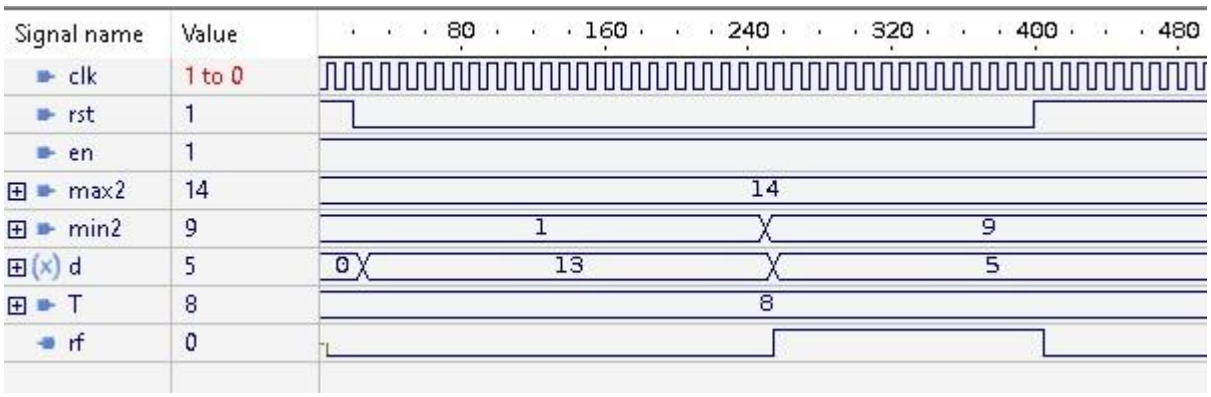
L'entité et le process VHDL correspondant sont les suivants :

```
entity roughness is
port (
clk,rst,en:in std_logic;
max2,min2: in unsigned(7 downto 0);
T: in unsigned(7 downto 0);
rf:out std_logic);
end roughness;
```



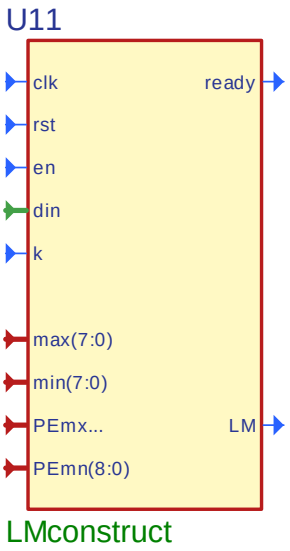

```
rfp:process (clk,rst,en)
variable d: unsigned(7 downto 0):="00000000";
begin
if clk'event and clk='1' then
        if rst='1' then
                rf <='0';
        elsif en='1' then
                d   :=max2-min2;
                if d >= T then
                        rf <= '0';
                else
                        rf <= '1'; --lisse
                end if;
        end if;
end if;
end process;
```

Pour tester ce bloc, on a pris comme stimuli deux situations : max2-min2 > T et max2-min2 < T. la simulation montre la sortie rf correspondante.



Bloc LM_construct :

C'est le circuit qui implémente la construction de la carte d'emplacements LM. Ce bloc prend comme entrées les valeurs k, max, min, PEmx et PEmn (nombre total de blocs 4x4, maximum, minimum, différence max-max2 et différence min-min2, respectivement) issues du bloc sort16. Selon les conditions définies par l'algorithme (chapitre III) ; ce bloc donne une sortie LM=0 pour un blocs d'image dans lequel il y a possibilité d'intégration de données, LM=1 dans le cas contraire. Le process VHDL écrit ainsi que l'entité sont données :



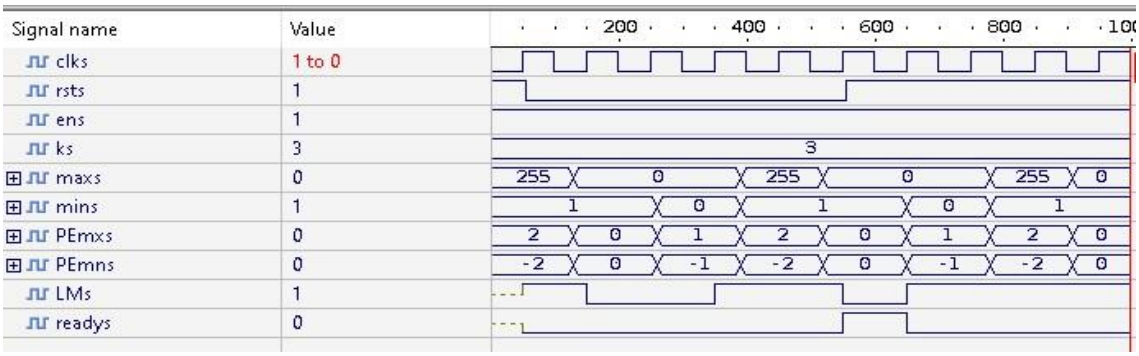

```
entity LMconstruct is
port (     clk,rst;en :  in std_logic;
din: in bus_array;
max,min: in unsigned(7 downto 0);
PEmx,PEmn:  in signed(8 downto 0);
k: in integer;
ready: out std_logic;
LM:out std_logic);
end LMconstruct;
```

```
lmap:process (clk,rst,en)
begin
if clk'event and clk='1' then
if rst='1' then LM <='1';  ready<= '0'; bcnt<=0;
elsif en='1' then
if (PEmx >= pone and max = two55 ) or (PEmn <= mone and min= zero
)   then
LM <= '1';   bcnt<= bcnt+1;
else
LM <= '0';  bcnt<= bcnt+1;
end if;
if  bcnt  >= (k+1) then
ready<= '1'; bcnt<=0;
end if; end if;
end if;
end process;
```

On a testé le bloc à travers les cas suivants:

```
maxs<="11111111"; wait for 130 ns;
maxs<="00000000";      wait      for      260      ns;
mins<="00000001";   wait for 260 ns;
mins<="00000000";   wait for 130 ns;
PEmxs  <=   "000000010" ;          wait for 130 ns;
PEmxs  <=   "000000000" ;          wait for 130 ns;
PEmxs  <=   "000000001" ;          wait for 130 ns;
PEmns  <=   "111111110" ;          wait for 130 ns;
PEmns  <=   "000000000" ;          wait for 130 ns;
PEmns  <=   "111111111" ;          wait for 130 ns;
```

La simulation montre que le bloc fonctionne comme attendu



Bloc LM_compress :

Ce bloc réalise la compression sans perte de la carte LM. LM devrait être une séquence de 0 et de 1. L'algorithme de RDH PEE à PVO prédit un grand nombre de zéros (blocs intégrables) contre quelques occurrences seulement d'uns. LM est donc constituée de longues séquences de zéros. La compression de ces zéros apporte une réduction considérable de la taille de LM.

On a choisi d'implémenter la compression à codage par plages (run length). C'est un algorithme qui repose sur l'idée de comprimer des plages de valeurs identiques en signalant le nombre de fois qu'une valeur donnée devrait être répétée.

L'idée est implémentée à travers une machine d'états séquentielle (Figure IV.3): on définit deux états : zeros et ones. Dans l'état zeros, la machine compte le nombre de zéros arrivants à l'entrée, une fois un 1 détecté à l'entrée la machine devance le nombre de zéros qu'elle vient de calculer, en plaçant devant '0', qui indique que c'est le nombre de zéros. En même

temps, la machine passe à l'état « ones » où la même procédure précédente se fait mais en comptant les 1 et en mettant un 1 en tête du nombre d'occurrence à la sortie.

L'état start est l'état de départ et de repos, où les sorties et les compteurs sont initialisés. Le signal ready est mis à 1 lorsque le comptage d'une plage (de 0 ou de 1) est terminé.

Le meilleur cas de compression a lieu lorsque tous les bits LM (k) (avec k nombre total du bloc 4x4 égal à 16384 pour une image de 512x512) sont des 0 et la taille de la séquence compressée est alors $1+\log_2(k)$. Ce qui donne 15 bits comme codage minimal pour une plage (qui code le nombre de 0 ou de 1); 14 bits pour le codage du nombre d'occurrence plus un bit pour 0/1 en tête. La sortie cLM est donc codée sur 14 bits. Cette sortie représente une plage (de 0 ou de 1) compressée. La carte finale est une séquence de plusieurs cLM, selon l'occurrence des 1 et des 0, qui va être sauvegardée dans une RAM_cLM (voir section blocs supplémentaires).

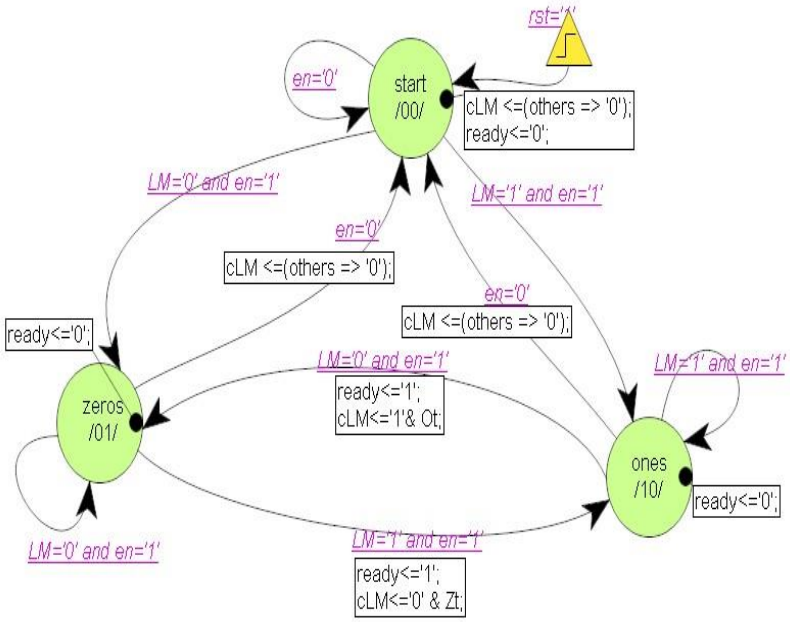
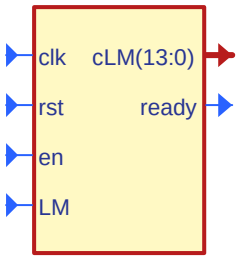


Figure IV.3: schéma de machine d'état séquentielle

La partie de code suivante représente l'entité VHDL ainsi que le symbole.

```
entity LMcompress is
port (
        clk: in std_logic;        rst:
in std_logic;            en:        in
std_logic;     LM: in std_logic;
cLM: out std_logic_vector (13 downto 0);
ready: out std_logic);
end LMcompress;
```



La description VHDL d'une machine d'état nécessite la définition d'un type personnalisé qui va représenter l'état. Pour notre implémentation ceci est fait par les lignes de code suivants :

attribute ENUM_ENCODING: string;
type Sreg0_type is (start, zeros, ones);
attribute ENUM_ENCODING of Sreg0_type: type is "00 " & "01 " &"10" ;

Le comptage des 0 et des 1 se fait par deux processus, qui ne sont exécutés que si la machine est dans l'état correspondant (c-à-d l'état ones pour compter les 1 et zeros pour compter les 0).

Les codes VHDL correspondants sont les suivants :

```
cnt1s:process(clk)
begin
if clk'event and clk='1' then
 if Sreg0= ones then
          Ot <=Ot+1;
          else
          Ot <=(others => '0');
 end if;
end    if;
end
process;
```

```
cnt0s:process(clk)
begin
 if clk'event and clk='1' then

if Sreg0= zeros then

Zt <=Zt+1;
else
 Zt <=(others => '0');
end if;
   end if;
end process;
```

Le code suivant montre les grandes lignes du processus de définitions des états et le passage entre elles :

```
Sreg0_NextState:  process(en,  LM,  Sreg0)
begin
 case Sreg0 is
        when   start   =>--initialisation
        when zeros => --tester LM en en
        when ones =>
                if LM='0' and en='1' then
        NextState_Sreg0 <= zeros;
        ready<='1';
                        cLM<='1'& Ot;
                elsif LM='1' and en='1' then
                        NextState_Sreg0 <= ones;
        elsif en='0' then
                        NextState_Sreg0 <= start;
                        cLM  <=(others  =>  '0');
                end if;                 when
others => null;
end case;
end process;
```

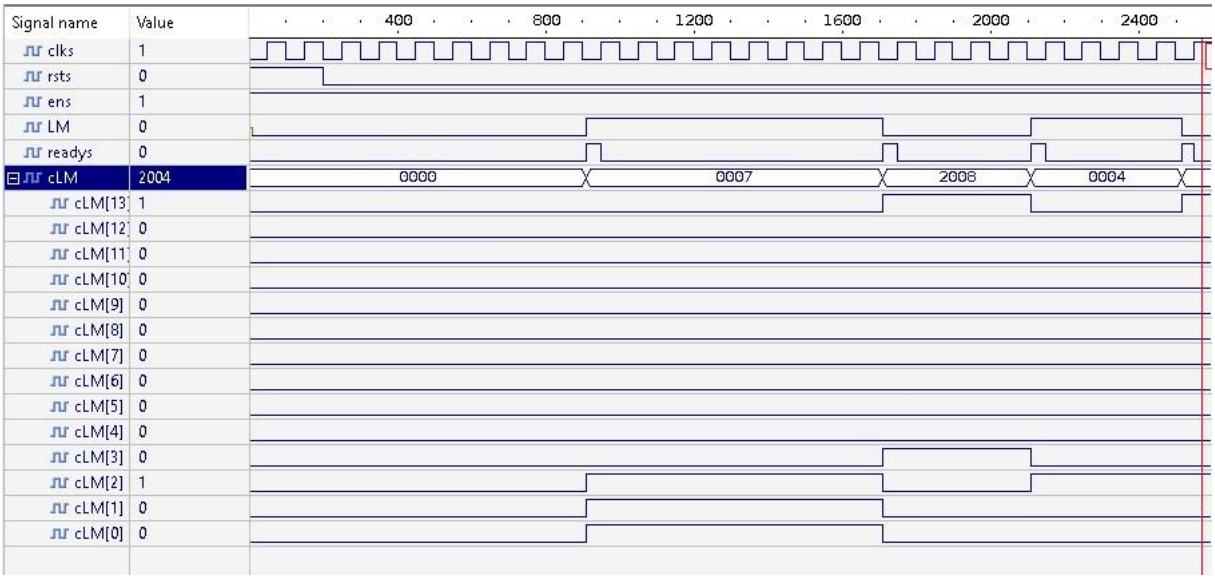
Une simulation est réalisée pour les stimuli d'entrée suivants :

```
        rsts  <=  '1';  wait  for  200  ns;
       rsts  <=  '0';  wait  for  2500  ns;
        clks <= '0'; wait for 50 ns;
clks <= '1'; wait for 50 ns;     LM   <=
'0'; wait for 900 ns;
         LM <= '1'; wait for 800 ns;
         LM <= '0'; wait for 400 ns;
         LM <= '1'; wait for 400 ns;
```

En prenant en compte le retard initial introduit par l'activation de remise à zéro et la période de l'horloge, les stimuli appliqués sont équivalents à :

{0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1}

On s'attend à une séquence compressée :{0(7),1(8),0(4),1(4)}. Ce résultat est confirmé par les chronogrammes suivant, où la sortie est représentée en radix 16 (hexadécimal).



Bloc embed_mxn :

C'est le bloc responsable de l'intégration de données (maximum deux bits dans un seul bloc 4x4 d'image). Selon l'algorithme, les pixels de valeurs maximale ou minimale sont admissibles à recevoir un bit à intégrer si quelques conditions sont rencontrées : le bit LM correspondant au bloc doit indiquer l'admissibilité à l'intégration (LM=0), le bloc doit être lisse (rf=1), la prédiction d'erreur PE >=1 pour pixel max, et PE <=1 pour le pixel min.

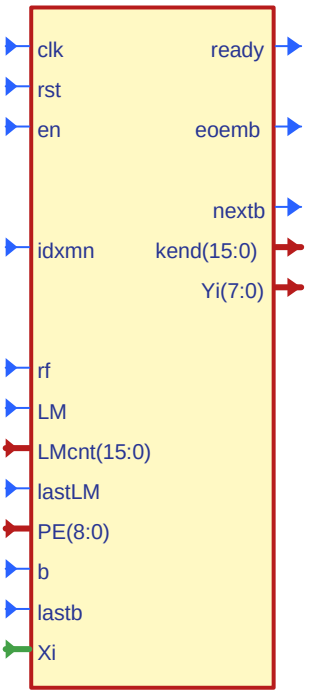
On a choisi d'implémenter ce bloc de la manière suivante : le bloc prend comme entrées : Xi (bloc d'image 4x4), LM (issu du bloc fifo_LM) , PE (issu du bloc seq_PE), idxmn (indice du pixel considéré, max ou min), rf (issu du bloc roughness) b (le bit de payload) en plus des signaux de contrôle : lastb (issu de fifo_b, indique que c'est le dernier bit de payload à intégrer), lastLM (issu de LMcomress, indique que c'est le dernier bit dans la séquence LM), LMcnt (issu de fifo_Xs, indique l'ordre du bit LM dans la fifo, donc l'indice du dernier bloc intégrable).

Pour chaque Xi, le bloc embed_mn s'exécute deux fois : d'abord on test les condition pour le pixel max et on incorpore un bit b (ou non) et une deuxième fois pour le pixel min, avec un nouveau bit b s'il y avait eu intégration dans le pixel max, sinon le même bit b précédent. Ceci se fait en activant la ligne nextb.

L'entité VHDL et une partie du process d'intégration correspondants sont les suivants :

U8

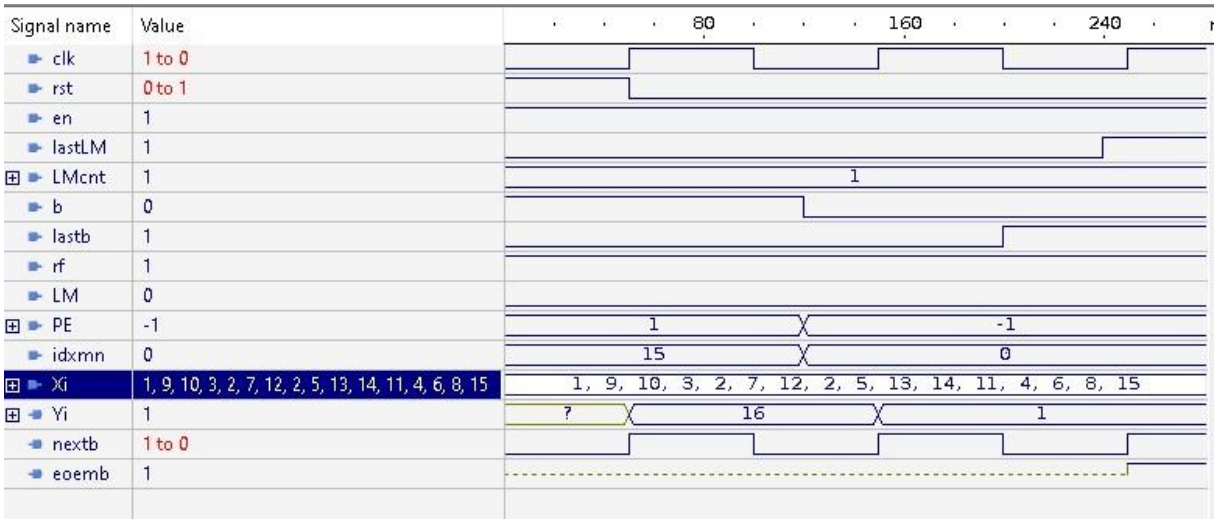
```
entity embedmxn is port(
clk : in STD_LOGIC;
        rst : in STD_LOGIC;
        en : in STD_LOGIC;
        rf : in STD_LOGIC;                    LM
: in STD_LOGIC;
                LMcnt: in std_logic_vector(15 downto
0); lastLM : in STD_LOGIC;     PE : in signed(8
downto 0);     b : in std_logic;              lastb: in
STD_LOGIC;   nextb: out STD_LOGIC;    idxmn :
in integer;    Xi : in bus_array;     ready : out
STD_LOGIC;                                    kend: out
std_logic_vector(15 downto 0);
        eoemb :    out std_logic;
        Yi : out unsigned(7 downto 0)     );
end embedmxn;
```




```
if PE>one and rf ='1' and LM='0' then

Yi     <= Xi(idxmn)+ 1;
elsif PE=one and rf ='1' and LM='0' then

 Yi   <=   Xi(idxmn)+ conv_unsigned(b,8);

 nextb<='1';
elsif  PE<mone and rf ='1' and LM='0' then

Yi     <= Xi(idxmn)- 1;

elsif PE=mone and rf ='1' and LM='0' then
```

La simulation suivante est réalisée pour une séquence de pixels :

(1,9,10,3,2,7,12,2,5,13,14,11,4,6,8,15). Et b=(1,0). Dans ce cas PEmx=1, PEmn=-1 LM=0. En supposant le bloc lisse on s'attend à une intégration dans deux pixels max (b=1) et min (b=0). Se fonctionnement est confirmé par les résultats suivants où on voit que le pixel max (à l'origine 15) est devenu 16 pour ainsi intégrer b=1, tandis que le pixel min (à l'origine =1) reste à 1 pour intégrer b=0:



Bloc S_construct :

Ce bloc permet de construire l'information auxiliaire Saux qui doit remplacer les S_{LSB} premiers bits de l'image après un premier tour d'intégration. SLSB est sauvegardée dans une **RAM_Slsb**. **Saux** est construite de $16+2\lceil log_2 N \rceil + l_{clm}$ bits et est codée selon le format : n1&n2&T&kend&lclm. l_{clm} est 2*$\log_2$N bits, où N=512x512, et $\log_2$N = 18, T est 8 bits n1 et n2 4 bits et k_{cnd} $\log_2$N bits. Il suffit de concaténer ses paramètres, passés au bloc S_construct comme entrées, pour former la séquence Saux les bits correspondant à chaque paramètre ; **Les blocs supplémentaires :**

D'autres blocs, qui ne font pas partie de notre projet, mais qui sont prévus pour une implémentation complètes ultérieure : les RAM **RAM_cLM, RAM_Y, RAM_Slsb ,** les files de données **fifo_b**, et **fifo_LM**, les blocs de conversion parallèle-série **seq_LM** et **seq_idx**.

Les files de données sont en général instanciés à partir de la bibliothèque de l'FPGA cible (Xilinx, Altera,etc…). Les fifo de tous les constructeurs offrent en général les mêmes options et possibilités d'implémentation. En plus des entrée et sortie série, il est possible de configurer un nombre de signaux de contrôle facilitant la bonne synchronisation du circuit : par exemple les signaux **empty** et **full** indiquent si la fifo est vide ou pleine, **almost_empty** et **almost_full** indiquent que la fifo est presque (à 25% pour xilinx) vide ou pleine, **fifo_cnt**

indique la quantité de donnée restante dans la file, **wen** et **ren** activent l'écriture ou la lecture, etc…..

Le bloc fifo_Xs :

C'est une banque de 16 fifo à partir du quelle, à chaque front d'horloge, sont lus 16 pixels constituants un bloc Xi. En plus de Xi, on doit avoir comme sorties : empty (fifo vides), full (fifo pleines), almost_empty (presque vides), almost_full (presque pleines) et Xcnt (nombre de données restantes).

Le bloc fifo_b :

Une file fifo_b est prévue pour garder la séquence du payload. En plus du bit à intégrer b, ce bloc doit fournir comme sortie un signal **empty** qui va indiquer la fin de la séquence b. Son entrée horloge doit être contrôlée par le signal **nextb** issu du bloc embed_mxn.

Le bloc fifo_LM :

Cette file reçoit le flux de bit généré par le bloc LMcompress. À sa sortie en doit avoir LMcnt (nombre de bits restants), empty (fifo vide).

Le bloc RAM_cLM :

La séquence LM originale a une longueur l_{LM}= N (nombre de blocs 4x4). Pour une image 512x512, on aura N=(512/4)x(512/4) = 128*128=16384.

(de taille N=nk, n=n1xn2, N=262,144 pour 4x4 et k=16384) est compressée selon le format : a&na où 'a '=0/1 indique la nature de la séquence compressée et na est le nombre d'occurrences du bit 'a' codé en binaire naturel.

Pour LM= '000…..0' ou LM= '111…..1' (séquence formée que de 0 ou que de 1), le nombre d'occurrence de a est na= Dans le cas de compression pour un seul type de plages (toute la séquence LM et formée de 0 ou bien de 1), le nombre d'occurrence est codé comme na=ceil(log2(N)) =14bits. Le format de cLM est donc 'axxxxxxxxxxxxxx' (14 bits pour na) qui va représenter la séquence totale.

La longueur de LM compressée totale dépend donc du nombre et de la longueur des plages compressés (runs)

IV.2.2 Décodage :

La figure IV.donne le schéma proposé pour la partie décodage. Le schéma est identique à celui de la partie décodage avec un bloc **S_extract** (au lieu de S_construct), **dembed_mxn** (au lieu de **embed_mxn)** et **LMdecompress** (au lieu de **LM compress)**. Le bloc d'image d'entrée Yi est un bloc de 4x4 pixels provenant de l'image tatouée Y. Les blocs **sort_mn**, et **roughness** sont les mêmes utilisés dans la partie codage. L'image tatouée Y est chargée dans un ensemble de files de données fifo_IM à partir d'une mémoire RAM. La sortie peut être écrite directement dans une RAM en adoptant un schéma d'adressage adéquat.

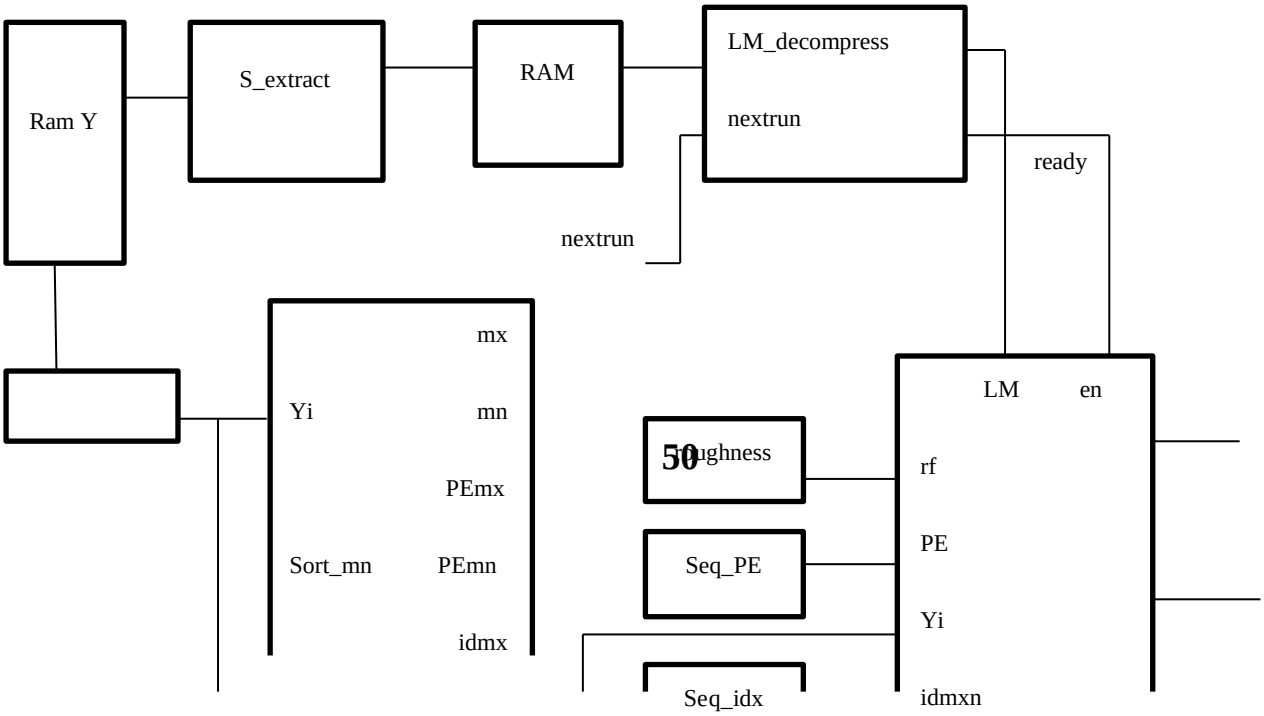


Figure IV.4: schéma de partie décodage Bloc

S_extract :

Ce bloc permet d'extraire les informations supplémentaires S intégrées lors du codage dans les

S_{LSB} premiers bits de l'image tatouée. S est construite de 16+2ceil(log2N)+lclm bits et est codée selon le format : n1&n2&2*$\log_2$N&$\log_2$N, où N=512x512, et $\log_2$N = 18. Il suffit d'extraire les bits correspondant à chaque paramètre ;

Bloc LM_decompress :

Ce bloc a pour tâche la décompression de la séquence cLM (LM compressée issue de LMcompress). Sachant que cette séquence a été compressée en plage sans perte, selon le format : a&na où 'a '=0 /1 indique la nature de la séquence compressée et na (na=xxxxxxxxxxxxxx 14 bits) est le nombre d'occurrences du bit 'a' codé en binaire naturel, on propose une conception qui permet de décompresser cLM en LM.

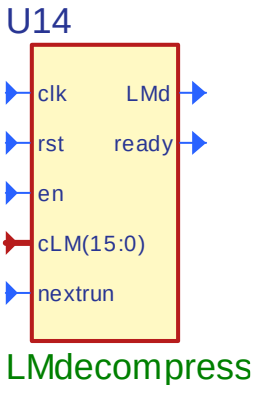
La séquence compressée cLM est extraite à partir de l'information supplémentaire intégrée dans l'image tatouée par **S_extract** et est sauvegardée temporairement dans une **RAM_cLM** selon le format **0a&na**. RAM_cLM est donc une RAM de taille 16 bits et une profondeur maximale égale à 2*(512/4)*(512/4) (2*nombre de bloc Yi) : c'est le cas de compression worst case où la séquence LM est constituée de 0 et de 1 consécutifs.

La procédure de décompression se déroule comme suis :

1. Lire la première case de RAM_cLM et en décoder les valeurs a et na
2. Générer sur la sortie dLM na fois le bit a (a=0 ou 1)
3. Si contenu de la case suivante et différent de zéro refaire 1) et 2)
4. Si contenu de la case suivante égale à 0 ou fin de la RAM_cLM, activer la ligne ready.

L'entité VHDL et une partie de l'architecture correspondante sont les suivants :

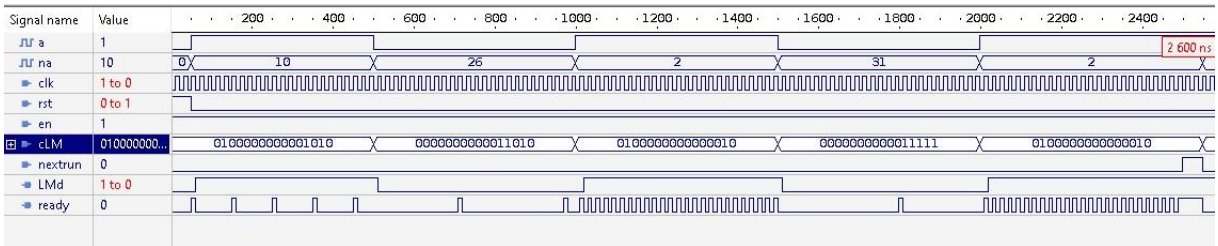
```
entity LMdecompress is
port (clk, rst, en: in std_logic;
cLM: in std_logic_vector(15 downto 0);
nextrun: in std_logic;
LMd: out std_logic;
ready: out std_logic);
end LMdecompress;
```



La simulation suivante est faite pour les stimuli :

```
cLMp:process
begin
cLM <="0100000000001010"; wait for 500 ns;
cLM <="0000000000011010"; wait for 500 ns;
cLM <="0100000000000010"; wait for 500 ns;
cLM <="0000000000011111"; wait for 500 ns;
cLM <="0100000000000010"; wait for 550 ns;
end process;
```

Les résultats de simulation montrent que la sortie LMd est une séquence décompressée de cLM

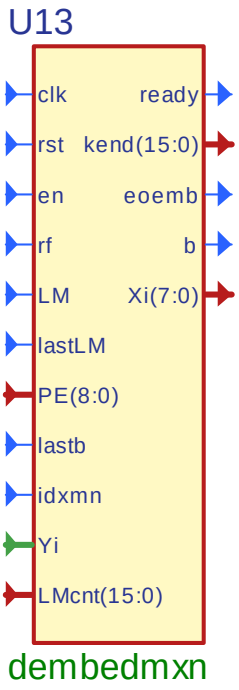


Bloc dembed_mxn :

Pour chaque bloc Yi (4x4 pixels), le décodage se fait en testant les valeurs PEmx, PEmn (issus du bloc sort_mxn), le bit LM et rf. En sortie, on trouve le bloc de pixels originaux Xi ainsi que le bit de donnée intégré b' (bit du payload). L'algorithme implémenté défini les conditions pour extraire un bit de payload comme (chapitre III) : (PEmx >2 et rf ='1' et LM='0') pour le pixel max et (PEmn<-2 et rf ='1' et LM='0'). Les conditions : ((PEmx =1 ou PEmx=2) et rf ='1' et LM='0') et ((PEmn =-1 ou PEmn=-2) et rf ='1' et LM='0') correspondent au décalage. Dans les autres cas, le pixel original est le même que celui dans le bloc Yi.

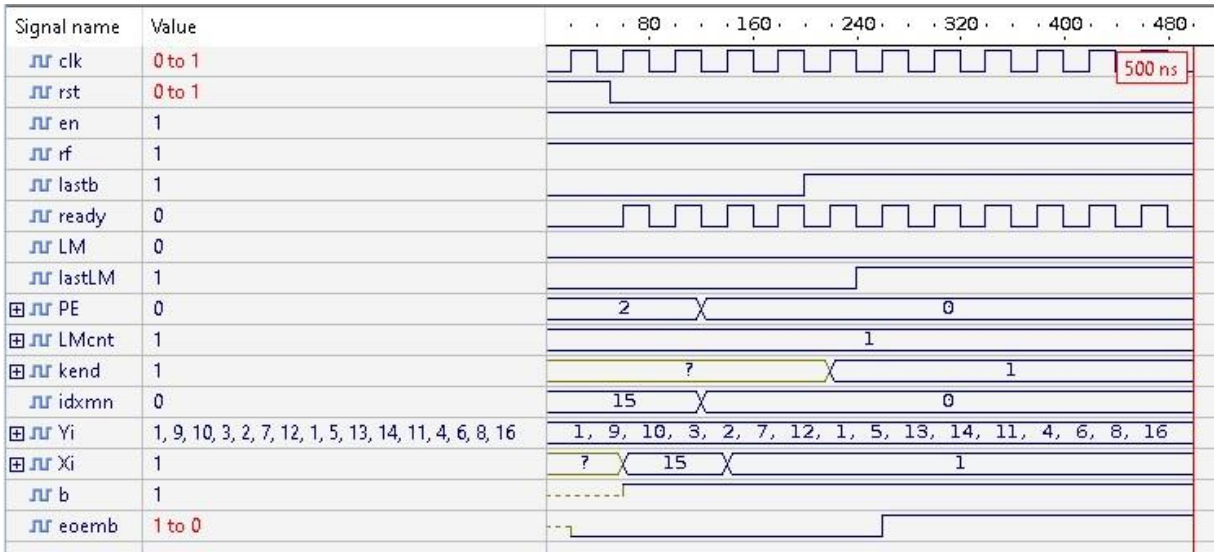
Les portions de codes suivants montrent l'entité VHDL et une partie du process d'extraction de données :

```
entity dembedmxn is
port(     clk : in STD_LOGIC;
rst : in STD_LOGIC;
en : in STD_LOGIC;
rf : in STD_LOGIC;
LM : in STD_LOGIC;
lastLM          : in STD_LOGIC;
PE : in signed(8 downto 0);
lastb:  in STD_LOGIC;
idxmn : in integer;
Yi : in  bus_array;
LMcnt: in  std_logic_vector(15 downto 0);
ready : out STD_LOGIC;
kend: out std_logic_vector(15 downto 0);   eoemb
:     out std_logic; --end of embedding
        b : out std_logic;--dembeded payload
Xi : out unsigned(7 downto 0));
end dembedmxn;
```




```
if PE>ptwo and   rf ='1' and LM='0' then                                    -
Xi      <= Yi(idxmn)- 1;
elsif (PE=pone or PE=ptwo) and rf ='1' and LM='0' then
bt      := PE-pone;
Xi      <=   Yi(idxmn)- conv_unsigned(bt,8);
b       <=       bt(0);
elsif  PE<mtwo and rf ='1' and LM='0' then
Xi      <= Yi(idxmn)+ 1;
elsif  (PE=mtwo or PE=mone) and rf ='1' and LM='0' then
bt      := -PE-pone;
Xi      <=   Yi(idxmn)+ conv_unsigned(bt,8);
b       <=       bt(0);
else
Xi      <=   Yi(idxmn);
end if;
```

La simulation suivante est réalisée pour une séquence de pixels : (1,9,10,3,2,7,12,1,5,13,14,11,4,6,8,15). Et b=(1,0). Dans ce cas PEmx=1, PEmn=0, LM=0 pour le pixel max et LM=1 pour le pixel min. En supposant le bloc lisse on s'attend à extraire un bit de payload b' à partir du pixel max (b'=1). Le pixel min est le même que l'original car PEmn=0 donc il n'y a pas eu d'intégration dans ce pixel. Se fonctionnement est confirmé par les résultats suivants où on voit que le pixel tatoué max (à l'origine 16) est devenu 15 pour ainsi extraire b'=1, tandis que le pixel min (à l'origine =1) reste à 1:



IV.3 Conclusion :

Dans ce chapitre, nous avons présenté la conception et l'implémentation l'algorithme d'intégration PEE à base des blocs de traitement des données soit dans le sens de codage ou bien le décodage et la description d'entité VHDL correspondants . La conception a commencé par l'identification des fonctions réalisées ce qui a abouti à des schémas d'implémentation partitionnés pour les parties codage et décodage. Les simulations faites prouvent le bon fonctionnement de chaque bloc.

Conclusion générale

À partir d'une description fonctionnel de l'algorithme de dissimulation de données à base de tri de valeur de pixels (PVO), on a construit une bibliothèque de blocs de construction permettant l'implémentation finale de cet algorithme. Nous avons commencé par l'étude et la compréhension de l'algorithme : ceci nous a permis d'identifier les fonctions clés constituants les procédures de codage (tatouage de l'image) et de décodage (récupération de la donnée intégrée ainsi que l'image support originale). Ainsi , nous avons construit deux schéma principaux pour ces deux procédures. Dans ces schéma nous avons écrit les descriptions VHDL de chaque bloc, et nous avons vérifié son fonctionnement par des simulations. Les codes VHDL décrivent des fonction combinatoires et séquentielles, selon le bloc. On a aussi utilisé des description de machines d'états.

Nous estimons que l'objectif principal de notre travail est complètement atteint, ce qui nous ramène vers une proposition de perspective de continuation pour ce travail : les fonctions clés

de l'algorithme étant conçues, il ne reste que de mettre le système entier en une seule conception en y ajoutant les différents blocs de mémoire et les files fifo. Une grande partie du succès du système repose sur une étude de synchronisation approfondie, vue la nature séquentielle de l'algorithme ; ceci représente une perspective possible pour ce travail.

Bibliographie

[1] G. Doerr. Security Issue and Collusion Attacks in Video Watermarking. phd thesis, Universitée de Nice-Sophia Antipois, 2005.

[2] I. J. Cox, M. L. Miller, J. A. Bloom, J. Fridrich, and T. Kalker. Digital water- marking and steganography. Morgan Kaufmann, 2008.

[3] F.Y Shih. Digital watermarking and steganography. CRC Press, Taylor & Francis Group, 2008.

[4] M. Barni and F. Bartolini. Watermarking systems engineering. Signal processing and communications series, 2004.

[5] B. Furht and D. Kirovski. Multimedia watermarking techniques and applica- tions. Auberbach publications, Taylor & Francis Group, 2006.

[6].J.R. Liu, W. Trappe, Z.J. Wang, M. Wu, and H. Zhao. Multimedia Fin- gerprinting Forensics for Traitor Tracing. EURASIP Book Series on Signal Processing and Communications, Hindawi Publishing Co., 2005.

[7] M. H. M. Costa. Writing on dirty paper. IEEE Trans. on Information Theory, 29 :439{441, 1983.

[8] Zhang X, Qian Z, Feng G, Ren Y (2014) Efficient reversible data hiding in encrypted images. J Visual Commun Image Represent 25(2):322–328.

[9] F, Deng RH, Ooi BC, Yang Y (2005) Tailored reversible watermarking schemes for authentication of electronic clinical atlas. IEEE Trans Inf Technol Biomed 9(4):554–563. https://doi.org/10.1109/TITB.2005.855556

[10] Coatrieux G, Guillou CL, Cauvin J, Roux C (2009) Reversible watermarking for knowledge digest embedding and reliability control in medical images. IEEE Trans Inf Technol Biomed13(2):158–165. https://doi.org/10.1109/TITB.2008.2007199

[11] Sanjay Kumar1 · Anjana Gupta1 · Gurjit Singh Walia, Reversible data hiding: A contemporary survey of state-of-the-art, opportunities and challenges. Applied Intelligence (2022) 52:7373–7406 https://doi.org/10.1007/s10489-021-02789-2

[12] Mathew T, Wilscy M (2014) Reversible data hiding in encrypted images by active block exchange and room reservation. In: 2014 International Conference on Contemporary Computing and Informatics (IC3I), pp 839–844. https://doi.org/10.1109/IC3I.2014.7019628

[13] Zhang W, MA K, Yu N (2013) Reversibility improved

[14]Abhinav A, Manikandan VM, AAB (2020) An improved reversible data hiding on encrypted images by selective pixel flipping technique. 2020 5*th* International Conference

[15]Barton JM (1997) Method and apparatus for embedding authentication information within digital data. U.S. Patent,5,646,997
[16]Fridrich J, Goljan M, Du R (2001) Invertible authentication. Proc SPIE Security and Watermarking of Multimedia Contents 4314:197–208
[17] Honsinger CW, Jones P, Rabbani M, Stffel JC (2001) Lossless recovery of an original image containing embedded data. U.S.Patent,6,278,79

[18] Fridrich J, Goljan M, Du R (2002) Lossless data Embedding-

[19]J. Tian, ``Wavelet-based reversible watermarking for authentication," *Proc. SPIE*, vol. 4675, pp. 679_690, Apr. 2002.
[20] J. Tian, ``Reversible data embedding using a difference expansion," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 8, pp. 890_896, Aug. 2003.

[21]Tian J (2002) Wavelet-based reversible watermarking for authentication. Proceedding SPIE 4675:679–690. https://doi.org/ 10.1117/12.465329
[22] A. M. Alattar, ``Reversible watermark using the difference expansion of a generalized integer transform," *IEEE Trans. Image Process.*, vol. 13, no. 8, pp. 1147_1156, Aug. 2004.

[23]Ni N, Shi YQ, Ansari N, Su W (2006) Reversible data hiding. IEEE Trans Circ Syst Vid Technol 16(3):354–362. https://doi.org/10.1109/TCSVT.2006.869964

[24]Lee S, Suh Y, Ho Y (2006) Reversible image authentication based on watermarking. In: 2006 IEEE International Conference on Multimedia and Expo, pp 1321–1324. https://doi.org/10.1109/ICME.2006.262782

[25] Vleeschouwer CD, Delaigle JE, Macq B (2001) Circular interpretation of histogram for reversible watermarking. 2001

[26] Loua DC, Choub CL, Weia HY, Huang HF (2013) Active steganalysis for interpolation-error based reversible data hiding.

[27] Wahed MA, Nyeem H (2017) Efficient LSB substitution for interpolation based reversible data hiding scheme. In: 2017 20*th* International Conference of Computer and Information Technology (ICCIT), pp 1–6. https://doi.org/10.1109/ICCITECHN. 2017.8281771

[28] Thodi DM, Rodriguez JJ (2004) Prediction-error based reversible watermarking. International Conference on Image Processing, 2004 ICIP '04 3:1549–1552. https://doi.org/10.1109/ICIP.2004. 1421361
[29] Thodi DM, Rodriguez JJ (2007) Expansion embedding techniques for reversible watermarking. IEEE Trans Image Process 16(3):721–730. https://doi.org/10.1109/TIP.2006.891046

[30] Hu Y, Lee H, Li J (2009) DE-based reversible data hiding with improved overflow location map. IEEE Trans Circ Syst Vid Technol 19(2):250–260. https://doi.org/10.1109/TCSVT.2008. 2009252

[31] M. Fallahpour, ``Reversible image data hiding based on gradient adjusted prediction," *IEICE Electron. Exp.*, vol. 5, no. 20, pp. 870_876, Oct. 2008.
[32] S. Xiang and Y. Wang, ``Non-integer expansion embedding techniques for reversible image watermarking," *EURASIP J. Adv. Signal Process.*, vol. 2015, p. 56, 2015.

[33] S.-K. Lee, Y.-H. Suh, and Y.-S. Ho, ``Reversiblee image authentication based on watermarking," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2006, pp. 1321_1324.

[34] X. Li, B. Li, B. Yang, and T. Zeng, ``General framework to histogramshifting- based reversible data hiding," *IEEE Trans. Image Process.*, vol. 22, no. 6, pp. 2181_2191, Jun. 2013.

[35] L. Kamstra and H. J. A. M. Heijmans, ``Reversible data embedding into images using wavelet techniques and sorting," *IEEE Trans. Image Process.*, vol. 14, no. 12, pp. 2082_2090, Dec. 2005.

[36] D. Coltuc, ``Improved embedding for prediction-based reversible watermarking," *IEEE Trans. Inf. Forensics Security*, vol. 6, no. 3, pp. 873_882, Sep. 2011.
[37] X. Li, B. Yang, and T. Zeng, ``Ef_cient reversible watermarking based on adaptive prediction-error expansion and pixel selection," *IEEE Trans. Image Process.*, vol. 20, no. 12, pp. 3524_3533, Dec. 2011.

[38] G. Coatrieux, W. Pan, N. Cuppens-Boulahia, F. Cuppens, and C. Roux, ``Reversible watermarking based on invariant image classi_cation and dynamic histogram shifting," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 1, pp. 111_120, Jan. 2013.

[39] C. Qin, C.-C. Chang, Y.-H. Huang, and L.-T. Liao, ``An inpaintingassisted reversible steganographic scheme using a histogram shifting mechanism," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 7, pp. 1109_1118, Jul. 2013.

[40] W. Hong, ``An efficient prediction-and-shifting embedding technique for high quality reversible data hiding," *EURASIP J. Adv. Signal Process.*, vol. 2010, Feb. 2010, article ID 104835.

[41] W. Hong, ``Adaptive reversible data hiding method based on error energy control and histogram shifting," *Opt. Commun.*, vol. 285, no. 2, pp. 101_108, Jan. 2012.

[42]J. J. Garcia-Hernandez, M. Nakano, and H. Perez, “Real time implementation of low complexity audio watermarking algorithm,” in Proceedings of the Third International Workshop on Random Fields and Processing in Inhomogeneous Media, Guanajuato, Mexico, October 2005.

[43] “Real time mclt audio watermarking and comparison of several whitening methods in receptor side,” in Proceedings of the Eighth IEEE International Symposium on Multimedia, San Diego, Cal, USA, 2006, pp. 991–997

[44]F. Wu, S. Chen, and H. Leung, “Data hiding for speech bandwidth extension and its hardware implementation,” in 2006 IEEE International Conference on Multimedia and Expo, 2006, pp. 1277–1280

[45] S. Mainty, A. Banerjee, and M. Kundu, “An image-in-image communication scheme and vlsi implementation using fpga,” in IEEE Indian annual conference (INDICON 2004), 2004, pp. 6–11.

[46]W. Irizarry-Cruz, “Fpga implementation of a video watermarking algorithm,” Master’s thesis, University of Puerto Rico, Mayaguez Campus, 2006

[47] H. Y. Leung, L. M. Cheng, L. L. Cheng, and C. Chan, “Hardware realization of steganographic techniques,” in Third International Conference on International Information Hiding and Multimedia Signal Processing (IIH-MSP 2007), vol. 1, 2007, pp. 279–282.

[48] J. J. Garcia-Hernandez, C. Feregrino-Uribe, R. Cumplido, and C. Reta, “On the implementation of a hardware architecture for an audio data hiding system,” Accepted in Journal of Signal Processing Systems ISSN: 1939-8018 DOI: 10.1007/s11265-010-0503-8, 2010

[49] A Reversible Data Hiding Algorithm for Radiological Medical Images and Its Hardware Implementation

[50] FPGA based implementation of reversible data hiding scheme for content verification and quality access control of image

[51] FPGA Realization of a Reversible Data Hiding Scheme for 5G MIMO-OFDM System by Chaotic Key Generation-Based Paillier Cryptography Along with LDPC and Its Side Channel Estimation Using Machine Learning Technique