# REMERCIEMENTS :

Avant tout, je tiens à remercier tous ceux qui m'ont apporté leur coopération et leurs suggestions utiles tout au long de ce stage.

Je tiens plus particulièrement à remercier :

M. Yannick Berthoumieu pour m'avoir orienté et suivi tout au long de ce stage.

M. Patrice Nouel pour son soutien et ses connaissances en VHDL.

M. Patrice Kadionik et M. Laurent Dulau pour leur aide et leurs explications diverses.

# TABLE DES MATIERES

# INTRODUCTION

Installé dans le laboratoire Technologie de l'Image et de la Communication de l'ENSERB, j'ai effectué mon stage pendant une période de deux mois.

Sous la responsabilité de M. Berthoumieu, j'ai repris le projet de fin d'étude de deux élèves de troisième année réalisé en 1998 afin de le mettre au point et si possible l'approfondir.

Dans un premier temps, j'ai du prendre connaissance et m'imprégner en détails de ce projet, pour pouvoir commencer concrètement mon travail dans un deuxième temps.

Mais tout d'abord, il est nécessaire de présenter le thème du projet et les objectifs de ce stage.

# 1. PRESENTATION DU PROJET

Lors de leur projet de fin d'étude Dominique BOUVERON et Florent RENAHY ont eu à réaliser une carte d'acquisition vidéo pour l'étude d'algorithmes de traitement d'images.

Cette carte reçoit en entrée un signal vidéo à la norme CCIR 625 lignes, 50 Hz, 1 Vcc, 75 Ω en charge.

L'acquisition est effectuée par un convertisseur analogique - numérique flash 8 bits afin d'obtenir une bonne résolution de l'image.

Les données sont traitées par un composant programmable de type FPGA Xilinx afin d'autoriser le téléchargement de nouveaux algorithmes pour le test et la mise au point d'algorithmes de traitement d'images. L'algorithme de détection de contours de type gradient est implanté dans ce composant programmable.

Le flot d'images traité est alors visualisé sur un moniteur de contrôle. Le but étant d'obtenir un traitement pseudo temps réel de l'image, le temps de traitement d'une image doit être le plus court possible.

La partie la plus importante de leur projet a été consacrée à l'étude et au choix de l'architecture de la carte. Ils ont pu valider les deux circuits programmables au niveau simulation. De plus, le routage et la réalisation de la plaque ont été effectués mais celle-ci n'a pu être testée faute de temps.

Afin que le travail qui a été réalisé ne soit pas mis de côté, et que la carte puisse être utilisée, il m'a été demandé de reprendre le projet, c'est-à-dire de tester la carte en vue de la faire fonctionner. Ces tests consistent principalement à :

- Vérifier le câblage.
- Supprimer les éventuels courts-circuits.
- S'assurer du bon fonctionnement des différents éléments analogiques (étage d'entrée et de sortie)

- Vérifier la justesse du contenu des circuits programmables concernant la gestion des mémoires et le traitement d'images.
- Avant d'envisager de quelconques tests sur la carte, il faut d'abord s'attarder longuement sur l'architecture de celle-ci pour bien en saisir le fonctionnement.

## 2. ARCHITECTURE DE LA CARTE

On peut décomposer le système en cinq modules principaux :

- **Conversion analogique/numérique** – elle est effectuée par un convertisseur ADC flash 8 bits TDA8708A. Le signal vidéo, après filtrage passe-bas, est échantillonné à une fréquence choisie par l'utilisateur. Ici on choisit de prendre 10 MHz pour traiter une image de 512*512 pixels. Cette partie est contrôlée par le composant programmable XC4006e **RAM_CONTROL**.
- **Synchronisation vidéo** – le circuit LM1881 est chargé de prendre, dans le signal vidéo, les différents signaux de synchronisation ( *csync*, *vsync* et *burst* ) et de les fournir au séquenceur.
- **Séquenceur** – intégré dans le composant programmable **RAM_CONTROL**, il devra gérer l'acquisition et la restitution du signal vidéo, ainsi que réguler l'accès aux mémoires entre les convertisseurs et le circuit XC4006e **CALCUL**. L'image est stockée dans la mémoire d'entrée constituée de 2 RAM 32 ko, l'une contenant la trame paire de l'image et l'autre la trame impaire.
- **Traitement de l'image** -- intégré dans le composant programmable **CALCUL**, il est effectué de manière asynchrone par rapport à l'acquisition des données. Contenant uniquement l'algorithme de traitement ainsi qu'une mémoire tampon de 9 pixels, il accède aux données dans la mémoire d'entrée, effectue le calcul pour chaque pixel de l'image et écrit le résultat dans la mémoire de sortie. Comme la mémoire d'entrée, celle-ci est aussi constituée de 2 RAMs 32 ko.
- **Conversion numérique/analogique** -- elle est effectuée par un convertisseur DAC 8 bits TDA 8702. Associé à un étage de mise en forme, il génère un signal vidéo qui sera envoyé sur un moniteur de contrôle afin de visualiser l'image traitée.

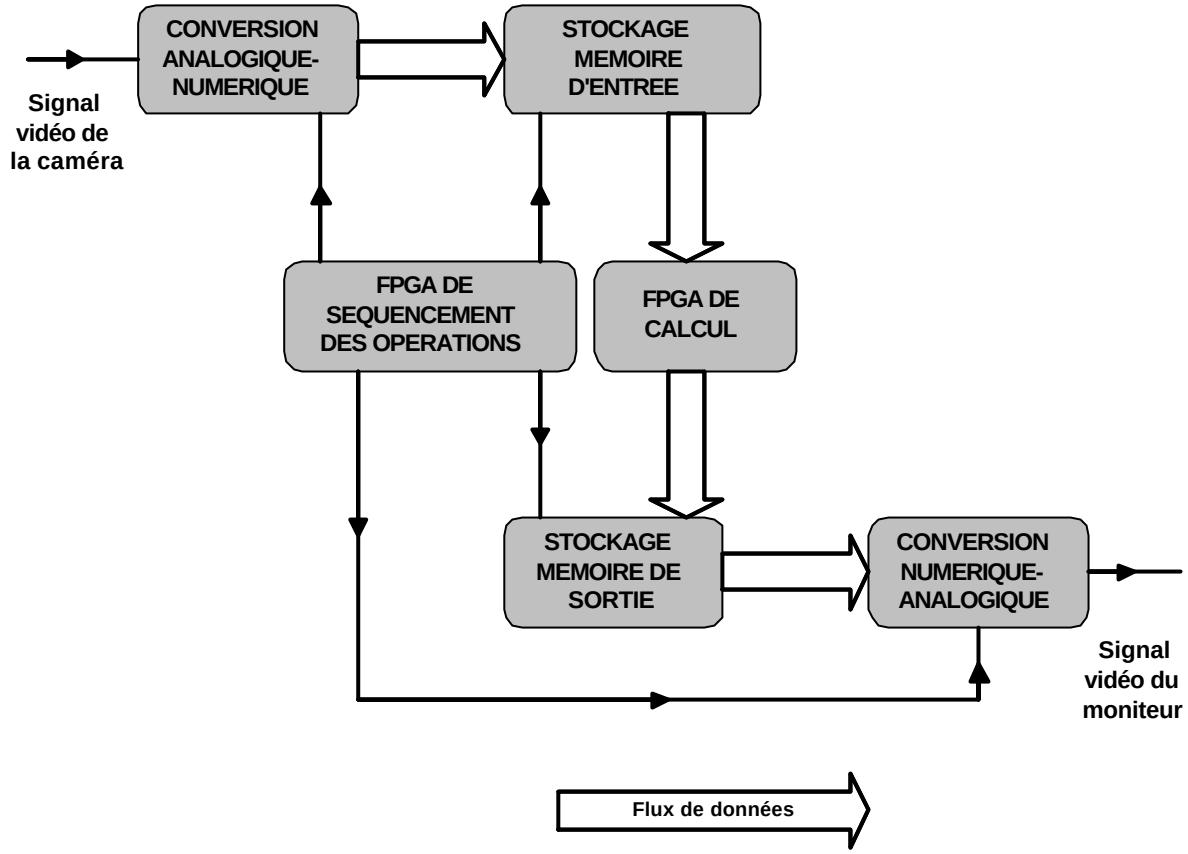



*Figure 1: Synoptique du montage*

## 2.1. PARTIE ANALOGIQUE :

Le circuit LM 1881 extrait, du signal vidéo, les signaux de synchronisation nécessaires au déroulement des opérations. Ces signaux sont :

- signal Composite Synchro appelé Csync qui donne les tops lignes et trames.
- signal Vertical Synchro appelé Vsync qui indique le changement de trames.
- signal Burst qui fournit un top pendant le « blanking », qui est la partie juste avant le signal utile sur une ligne.

Le circuit TDA 8708A est un convertisseur analogique/numérique 8 bits. Il intègre un amplificateur vidéo avec blocage et contrôle automatique de gain. En lui fournissant les signaux appropriés (GATE A et GATE B selon la documentation du composant) il peut calibrer automatiquement son échelle de conversion en fonction du signal vidéo qu'il reçoit.

L'inconvénient avec ce convertisseur est que la plage de conversion (avec le fonctionnement en mode 2 qui est notre cas) va de 64 pour le niveau de noir à 248 au maximum pour le niveau de blanc. Le signal GATE A doit être présent pendant le top de synchronisation ligne afin que le convertisseur sache à quel niveau se trouve celui-ci. Il en fixe le niveau à zéro. Le signal GATE B doit être présent au moment du top Burst qui donne alors le niveau du noir de l'image, qui est codé 64.

Enfin un «Peak level comparator » surveille le niveau maximum du signal vidéo (correspondant au niveau du blanc) et le code à 248. Si le signal vidéo tend à excéder 248 le gain en interne sera limiter pour éviter tout dépassement de capacité du convertisseur.

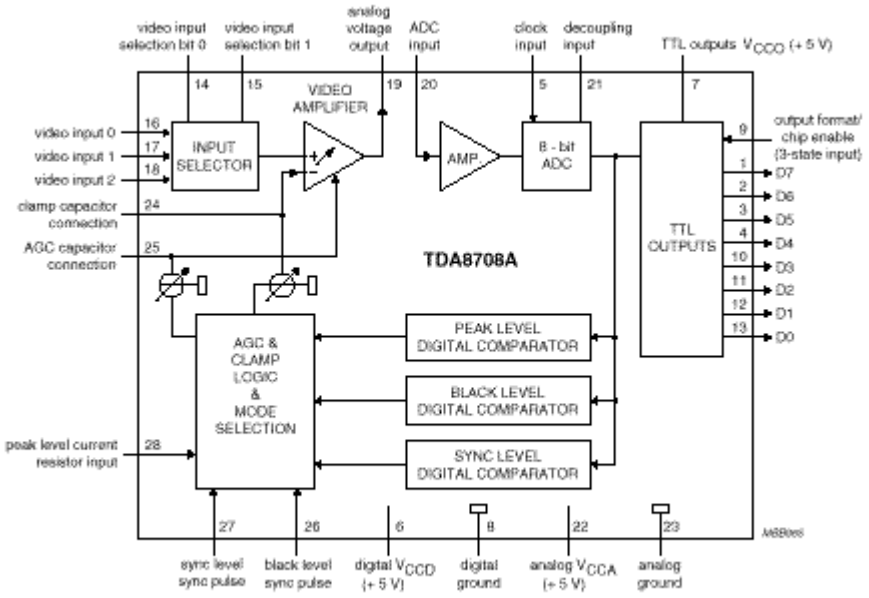



*Figure 2: Schéma interne du TDA8708A*

La documentation du composant suggère de placer un filtre passe bas entre les broches 19 et 20, ce qui a été fait :

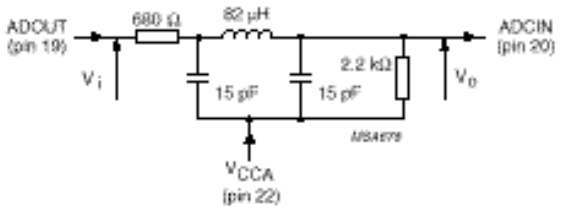



*Figure 3: Filtre passe-bas d'ordre 5 type Chebyshev*

En ce qui concerne l'étage de sortie, le problème est la restitution de l'image sur un moniteur par le convertisseur. En effet, lors de l'échantillonnage, seule l'information utile est conservée ; on perd ainsi les signaux de synchronisation vidéo. Pour recomposer le signal vidéo en sortie, il est nécessaire d'ajouter le signal de synchronisation aux données traitées (qui sont stockées dans la mémoire de sortie). Pour récupérer le signal de synchronisation, deux méthodes sont possibles :

- la première consiste à régénérer artificiellement le signal de synchronisation dans un composant programmable ou par un circuit tel que le générateur de synchronisation Philips SAA1101,
- la seconde consiste à utiliser le signal de synchronisation extrait par le circuit LM1881 du signal vidéo d'entrée. Cette méthode, plus simple, nous permet alors d'utiliser les mêmes compteurs ligne et colonne pour la lecture des mémoires.

Le circuit TDA 8702 est un convertisseur numérique/analogique de type flash. La conversion s'effectue sur le niveau d'horloge à l'état bas. Si l'horloge est en permanence à

l'état bas tous les changements de code en entrée sont répercutés en sortie. Après une conversion, si l'horloge repasse à l'état haut la valeur à convertir est mémorisée et les éventuels changements à l'entrée ne sont pas pris en compte.

Ce convertisseur a une sortie courant, d'où l'utilisation d'un transistor à la broche 15.

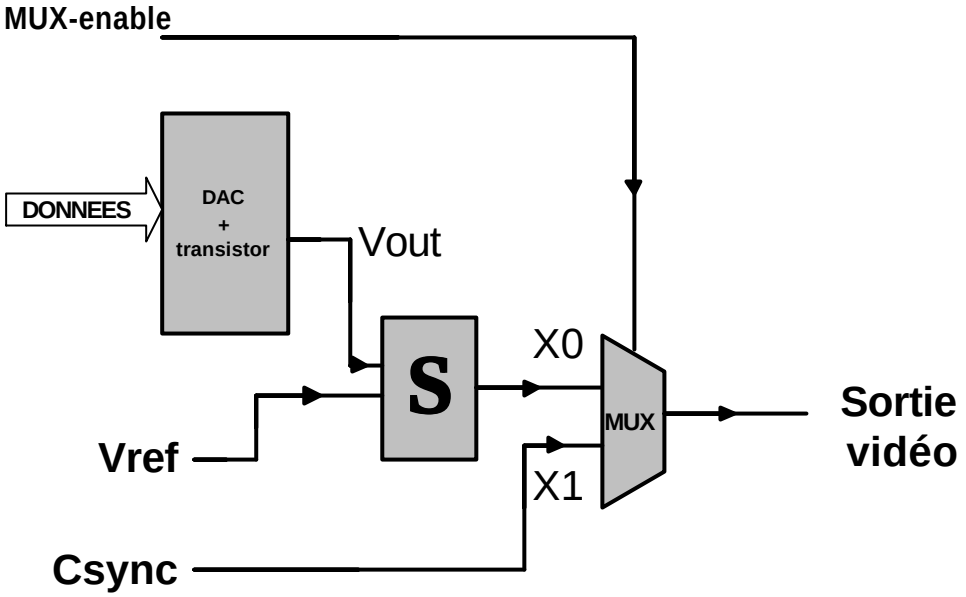



*Figure 4: étage de sortie vers le moniteur de contrôle*

Soient X0 le signal à la sortie de l'amplificateur opérationnel, qui rentre sur la broche X0 du multiplexeur analogique, X1 le signal qui rentre sur la broche X1 du multiplexeur, *Vout* le signal à l'émetteur du transistor (*Vout* rentre dans le sommateur) et *Vref* le deuxième signal rentrant dans le sommateur.

On a alors, avec les notations du schéma :

$$Vref = \frac{-5}{1+\frac{RV5}{R9}}, \quad X1 = \frac{Csync}{1+\frac{R11}{RV4}}, \quad X0 = \left(1+\frac{RV2}{R10}\right)\left(\frac{Vref \times RV3 + Vout \times RV1}{RV3 + RV1}\right)$$

On voit donc bien avec ces formules l'influence des résistances variables sur le signal vidéo de sortie.

On va prendre RV3 = RV1 et RV2 = R10 de manière à avoir $X0 = Vref + Vout$.

Après une première phase de test, nous avons choisi comme valeurs optimales :

$RV1 = RV3 = 10K\Omega$

$RV2 = R10 = 100K\Omega$

RV1,2,3 sont des résistances variables ce qui laisse la possibilité de les ajuster au mieux lors de la phase de mise au point.

Le signal Vout est compris entre 3V et 4,4V (voir schéma) ; le but est de le ramener entre 0,6V et 2V avant de l'envoyer sur le multiplexeur. Donc Vref, le signal négatif que l'on

ajoute à Vout, doit être de 2,4V d'où ( comme R9 = 1,5K) la valeur de RV5 = 1,6K. Nous avons ici aussi prévu une résistance variable de 2,2K.

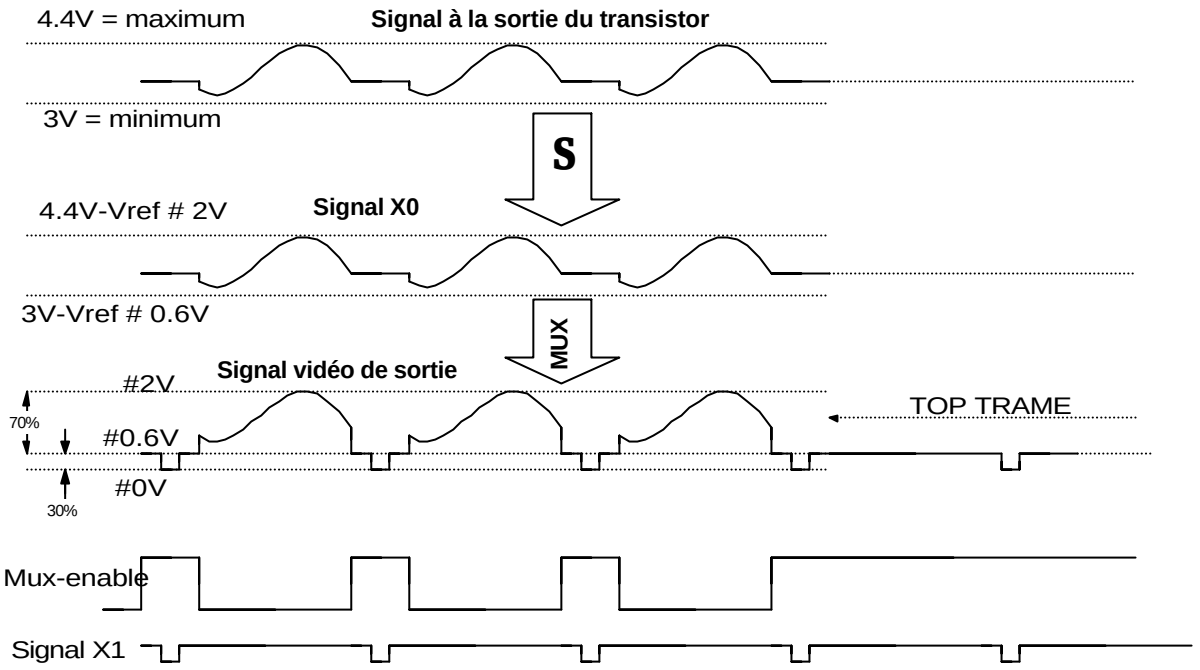



*Figure 5: Restitution du signal vidéo*

Le signal X1 doit être compris entre 0 et 0,6V d'où le choix de R11=10K et RV4 = 1,3K. Ainsi le multiplexage des signaux X0 et X1 donnera bien un signal vidéo compris entre 0 et 2V avec des tops de synchronisations compris entre 0 et 0,6V.

Le signal de multiplexage est fourni par le circuit Xilinx RAM_CONTROL. Il est fait de telle manière qu'il laisse passer le signal X0 (niveau 0) pendant le signal vidéo utile des lignes valides. Sinon, il est à 1 et laisse donc passer le signal de synchronisation X1.

Le signal résultant du multiplexage peut-être envoyé directement sur le moniteur.

## 2.2. PARTIE NUMERIQUE :

Plusieurs méthodes ont été étudiées pour permettre un traitement pseudo temps réel de l'image. Il a été choisi de concevoir la partie traitement asynchrone par rapport à l'acquisition/restitution du signal vidéo. Le seul problème consiste à réguler les accès aux mémoires pour éviter les conflits entre les parties traitement et acquisition, ce qui sera fait dans le composant RAM_CONTROL et en utilisant des buffers unidirectionnels 74HC244. L'interconnexion des deux composants est visualisé est en annexe 4.

Nous allons d'abord étudier le circuit CALCUL chargé du traitement de l'image, puis le circuit RAM_CONTROL chargé de réguler les accès aux mémoires, avant d'aborder l'aspect synthèse et simulation.

### 2.2.1 ETUDE DU CIRCUIT CALCUL :

Ce circuit est composé de 2 composants VHDL : *fifo* et *calcul*. Dans le cas de téléchargement de nouveaux algorithmes sur la carte, seul ce circuit sera modifié.

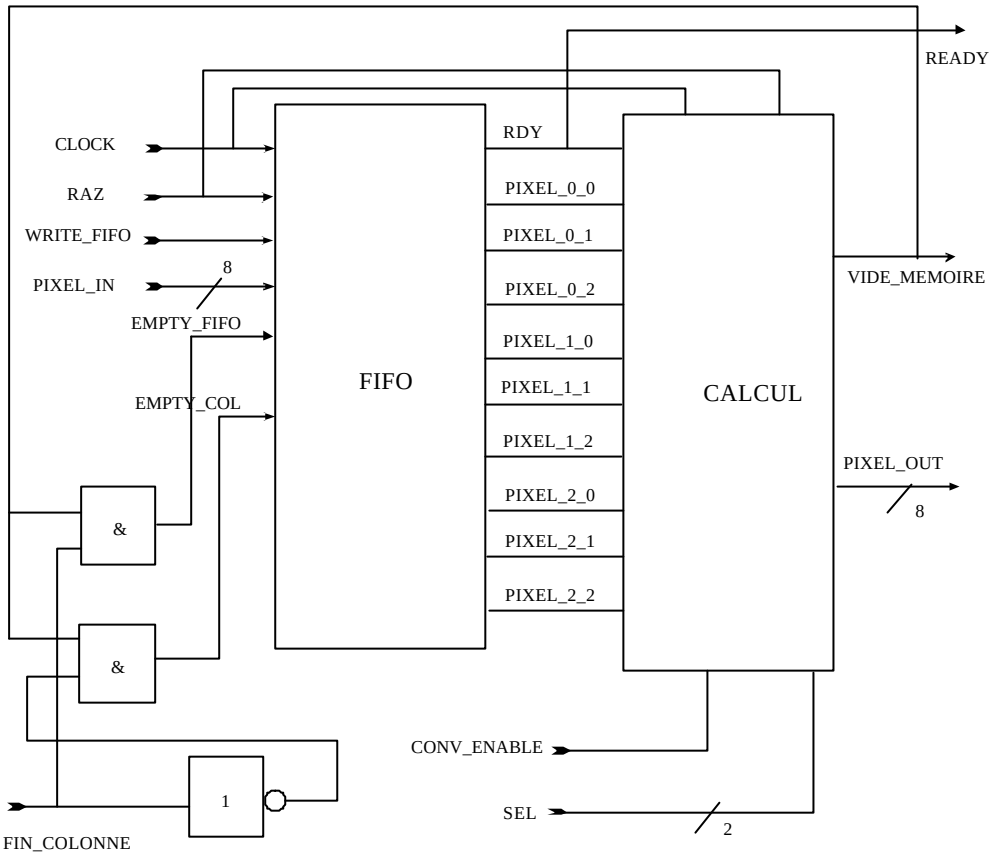



*Figure 6: Représentation graphique du composant CALCUL*

La carte est conçue pour traiter les images avec des algorithmes de bas niveau. Dans ces algorithmes, pour le calcul d'un pixel, 9 pixels sont nécessaires ; c'est pourquoi une mémoire tampon de 9 octets a été implantée dans ce circuit.

Il a été établi de choisir une structure type fifo pour cette mémoire. On évite ainsi d'utiliser un compteur d'adresses supplémentaire. De plus, ce composant ayant été réalisé et testé lors d'un projet de 2ème année, cela a simplifié la conception du circuit. Seules quelques modifications ont été nécessaires pour l'adapter à notre cas.

L'échantillonnage des données *pixel_in[7 :0]* dans la mémoire fifo se fait sur front montant de l'horloge *clock*, alors que le signal *write_fifo* est à l'état haut. Dès que les 9 pixels ont été stockés dans la mémoire, le signal *rdy* passe alors à l'état haut. Les signaux *empty_fifo* et *empty_col* sont utilisés pour effacer soit toute la mémoire au début de chaque colonne de l'image d'entrée, soit les 3 premiers pixels échantillonnés.

Dès que le signal *rdy* passe à l'état haut, le calcul peut commencer. Dans notre application, il est possible de choisir par switch entre 3 algorithmes de calculs différents : recopie de l'image, détection de contours et le filtre *emboss*. La machine d'état implantée, active sur front descendant de l'horloge *clock*, dispose des 3 états suivants :

- Etat ***attente*** : le but est d'attendre que les 9 pixels soient disponibles en entrée avant de commencer le calcul. C'est pendant cet état que le vidage de la mémoire fifo est effectué. Dès que le signal *debut_calcul* passe à l'état haut, la machine d'état passe alors à l'état ***calcul***.
- Etat ***calcul*** : l'algorithme de calcul est choisi avec les bits *sel[1 :0]* via des switchs de sélection.
- L'image résultat B(i, j) est une combinaison linéaire des neufs pixels de départ. Le filtre peut être exprimé par la matrice des coefficients de pondération :

$$F = \begin{bmatrix} a & b & c \\ d & e & f \\ j & g & h \end{bmatrix} = \begin{bmatrix} K(i-1, j-1) & K(i, j-1) & K(i+1, j-1) \\ K(i-1, j) & K(i, j) & K(i+1, j) \\ K(i-1, j+1) & K(i, j+1) & K(i+1, j+1) \end{bmatrix}$$

soit

$$B(i, j) = \sum_{-1}^{1} \sum_{-1}^{1} K(i+k, j+l).A(i+k, j+l)$$

Si le filtre F a tendance à ajouter les voisins au pixel B(i, j) il y a moyennage et F est du type passe-bas. Au contraire, si F a tendance à soustraire les pixels voisins, il y a dérivation et F est du type passe-haut.

Les concepteurs de la carte ont implanté deux algorithmes et gardent la possibilité de recopier l'image. Ces filtres sont :

Le filtre emboss : la matrice F est alors de la forme :

$$F = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

La détection de contours avec le gradient de Prewitt : le principe de l'algorithme est décrit ci-dessous.

Soit un contour d'orientation $\theta$ au point (x, y). Ce contour est détecté par le maximum de la dérivée dans la direction $\phi$ du gradient $\vec{\nabla} f(x, y)$ soit le maximum de la fonction $g(\phi) = \vec{\nabla} f(x, y).\vec{n}$ où $\vec{n}$ est le vecteur unitaire dans la direction du gradient $\vec{n} = (\cos\phi, \sin\phi)$.

$$g(\phi) = \cos\phi.\frac{df}{dx} + \sin\phi.\frac{df}{dy}$$

Dans le cas discret ( qui est notre cas puisque l'on travaille avec des pixels), on approxime les dérivées partielles par de simples différences :

$$\frac{df}{dx} = f(i+1, j) - f(i, j) = \Delta_x f(i, j)$$
$$\frac{df}{dy} = f(i, j+1) - f(i, j) = \Delta_y f(i, j)$$

L'opérateur norme du gradient est donné par

$$\left\|\vec{\nabla}\vec{f}\right\| = \max(\left|\Delta_x f\right|,\left|\Delta_y f\right|)$$

Raisonnons avec des matrices qui représentent les images d'entrée (A) et de sortie (B), où $A, B \in \mathrm{M}([0,511],[0,511])$.

Les dérivées directionnelles horizontale et verticale s'expriment sous la forme :

gradient en x : $A_x(i, j) = H * A(i, j)$ (* = convolution)

gradient en y : $A_y(i, j) = V * A(i, j)$.

On aura alors $B(i, j) = \max(\left|A_x(i, j)\right|,\left|A_y(i, j)\right|)$

avec $H = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ et $V = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

H et V sont appelées masques, avec $H, V \in \mathrm{M}([-1,1],[-1,1])$

Elles dépendent du type de gradient calculé, ici gradient de Prewitt. Développement :

$$A_y(i, j) = \sum_{k=-1}^{1}\sum_{l=-1}^{1} A(i-k, j-l).V(k, l)$$

$$A_x(i, j) = \sum_{k=-1}^{1}\sum_{l=-1}^{1} A(i-k, j-l).H(k, l)$$

On constate que l'on ne pourra pas calculer les pixels des bords de l'image de sortie, correspondant aux indices i={0, 511}et j={0,511}, puisque (i- k) et (j- l) ne doivent pas avoir les valeurs -1 ou 512. Les pixels de sortie effectivement calculés seront donc B(i, j) avec $(i, j) \in [1,510]^2$

On peut remarquer que **H = a + b + c et V = a - c - d** avec :

$$a = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, c = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, d = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

D'où :

$A_x = Ca + Cb + Cc$

$A_y = Ca - Cc - Cd$

$Ca = -A(i+1,j+1) + A(i-1,j-1)$ ; $Cb = -A(i,j+1) + A(i,j-1)$

Cc = -A(i-1,j+1) + A(i+1,j-1) ; Cd = -A(i-1,j) + A(i+1,j)

En résumé , avec **A(i, j)** matrice d'entrée et **B(i, j)** matrice de sortie :

$$B(i,j) = (1/3).\max(|Ca + Cb + Cc|, |Ca - Cc - Cd|) \ , \ (i,j) \in [1,510]^2$$

Ca = -A(i+1,j+1) + A(i-1,j-1) ; Cb = -A(i,j+1) + A(i,j-1) ;
Cc = -A(i-1,j+1) + A(i+1,j-1) ; Cd = -A(i-1,j) + A(i+1,j) ;

Le facteur (1/3) introduit servira à normaliser les coefficients de la matrice. Effectivement : $\max(A_x, A_y) \in [0,3*255]$ puisque les pixels sont codés sur un octet.

.Dans le cas de la recopie de l'image, le pixel de sortie est égal au pixel K(i, j).

Pour les filtres de détection de contours et *emboss*, le calcul s'effectue avec des opérandes type entier avant d'être converti en std_logic_vector. Tous les calculs s'effectuent en un cycle d'horloge. Ensuite, la machine d'état passe à l'état ***fin_calcul***.

- Etat ***fin_calcul*** : si le calcul s'effectue pendant l'échantillonnage d'une ligne vidéo, il est nécessaire d'attendre que le composant CALCUL ait accès à la mémoire de sortie pour pouvoir écrire le résultat, soit lorsque *conv_enable* passe à l'état bas.

### 2.2.2 ETUDE DU CIRCUIT RAM_CONTROL:

Ce circuit est composé des 6 composants vhdl : *adc_control*, *compteur_colonne*, *compteur_ligne*, *compteur_entree*, *compteur_sortie* et *mux*, et réalise leur interconnexion (cf. figure 7 ). Il gère l'acquisition et la restitution du signal vidéo, et régule l'accès aux mémoires entre les convertisseurs et le circuit XC4006e **CALCUL**.

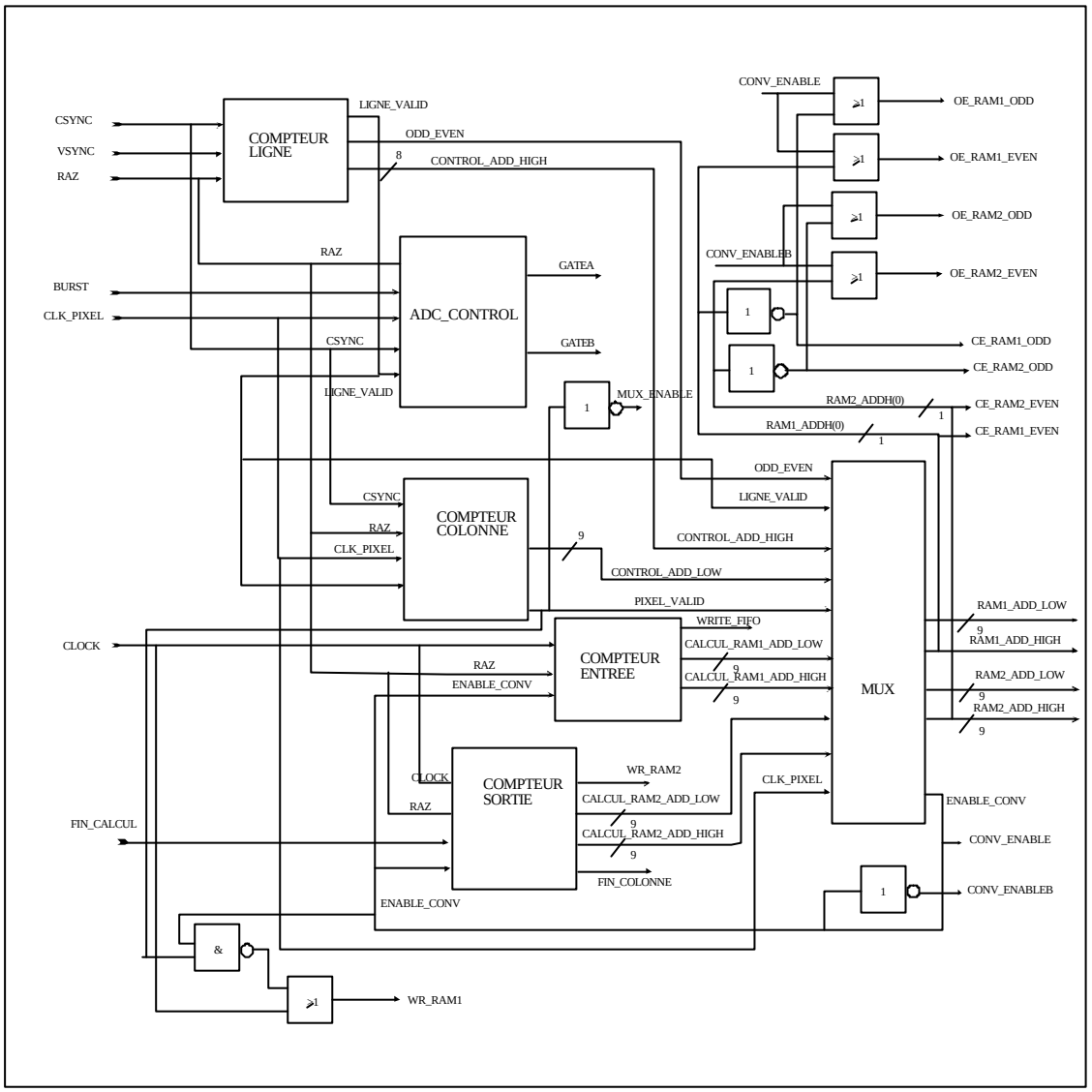



*Figure 7: Représentation graphique des entités de RAM_CONTROL interconnecté*

### 2.2.2.1 ETUDE DU COMPOSANT ADC_CONTROL :

Cette entité est utilisée pour générer les signaux de contrôle du convertisseur ADC TDA8708A. En effet, ce circuit a besoin des deux signaux Gate A (Sync level sync pulse) et Gate B (black level sync pulse) pour sélectionner le mode de fonctionnement.

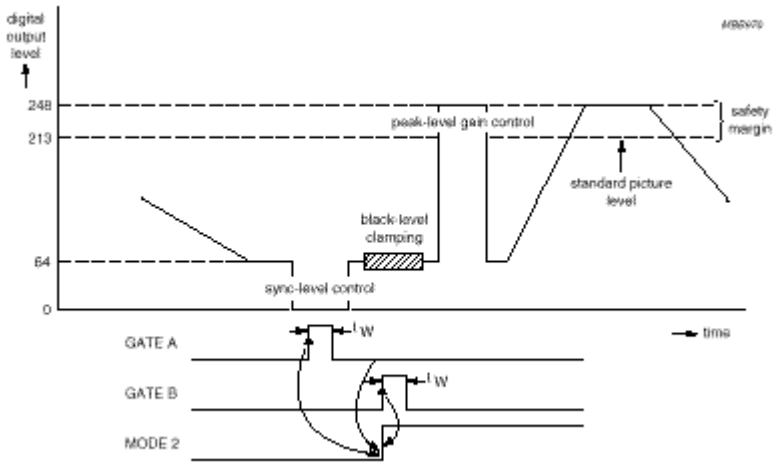



*Figure 8: Mode de contrôle 2*

L'horloge *clk_pixel* cadencée à 10 MHz permet de générer les signaux pendant une durée déterminée, sachant que la durée minimale des impulsions doit être $t_w = 2\ \mu s$.

#### 2.2.2.2 ETUDE DU COMPOSANT *COMPTEUR_COLONNE* :

Cette entité est utilisée pour n'autoriser l'échantillonnage du signal vidéo que pendant sa phase utile. Pendant cette phase, le signal *pixel_valid* est alors mis à l'état haut. Elle intègre le compteur *add_low* qui génère les adresses poids faible des mémoires pendant la phase de conversion. Ce compteur est remis à zéro à chaque impulsion de synchronisation horizontale (csync).

#### 2.2.2.3 ETUDE DU COMPOSANT *COMPTEUR_LIGNE* :

Cette entité intègre deux processus : l'un permet de générer le signal de parité *odd_even* qui change d'état à chaque front descendant de l'impulsion de synchronisation verticale (vsync), l'autre intègre le compteur *add_high* qui génère les adresses poids fort des mémoires pendant la phase de conversion. Seules les lignes 30 à 285 de chaque trame vidéo sont considérées valides pour l'échantillonnage du signal vidéo (le signal *ligne_valid* est alors mis à l'état haut).

#### 2.2.2.4 ETUDE DU COMPOSANT *COMPTEUR_ENTREE* :

Cette entité intègre le compteur d'entrée utilisé pour lire les données dans la mémoire d'entrée pour les stocker dans la mémoire fifo. Ce compteur est scindé en deux compteurs correspondant aux coordonnées du pixel dans la mémoire, la coordonnée (x) générée par le compteur 'entree_colonne', la coordonnée (y) par le compteur 'entree_ligne'.

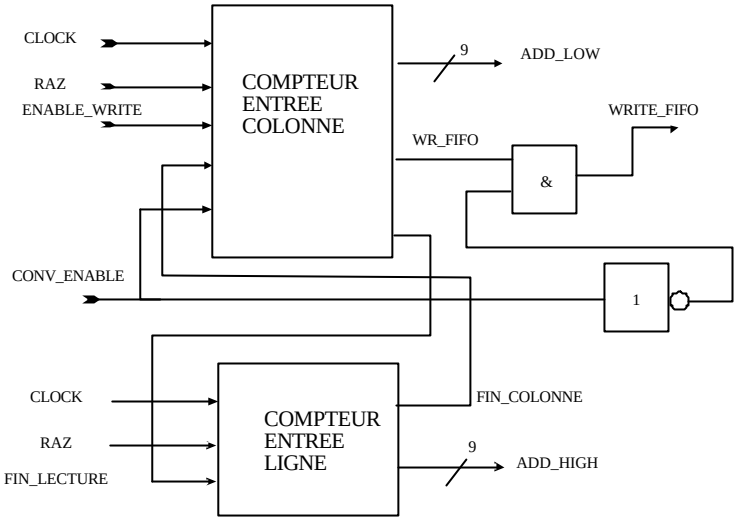


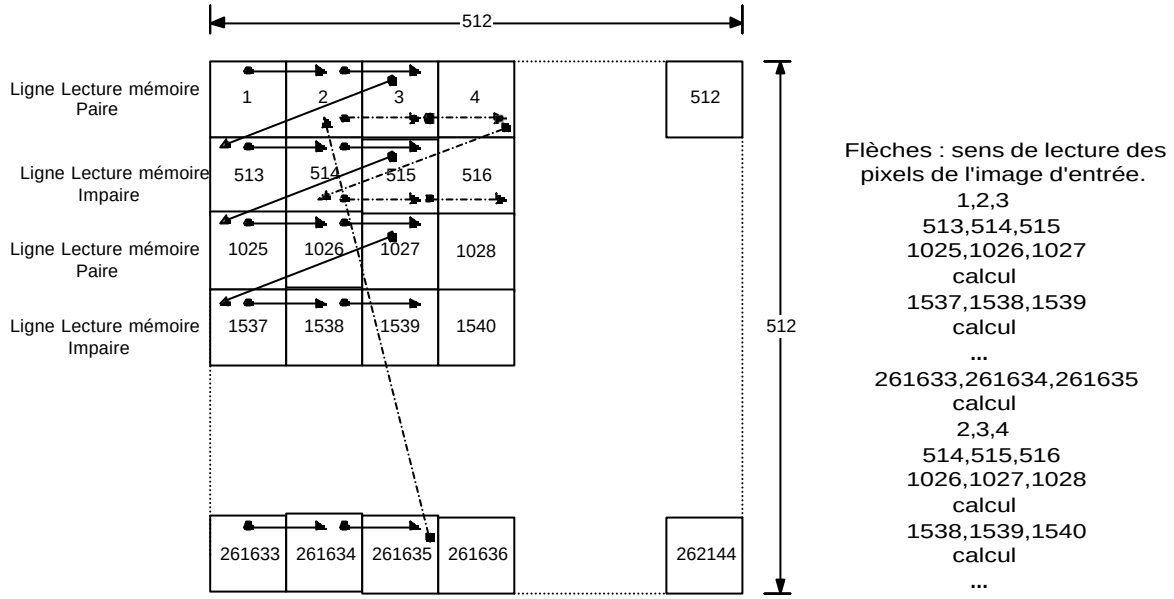

*Figure 9: Représentation graphique des entités interconnectés*

D'après la méthode de lecture de la matrice correspondant à l'image d'entrée, 3 pixels sont d'abord lus sur la ligne courante, puis le compteur ligne est incrémenté. Au début de chaque colonne, il est nécessaire de lire 9 pixels pour commencer le calcul. Pour le calcul suivant, seuls 3 pixels sont lus puisque les 6 derniers pixels lus précédemment sont utilisés également dans ce calcul. Cette méthode permet de réduire considérablement le nombre d'accès à la mémoire et ainsi d'augmenter la rapidité du système puisque chaque cycle de lecture d'un pixel prend un cycle d'horloge. Le nombre d'accès mémoire en entrée est donc de : 510 * 9 + 509 * 510 * 3 accès mémoires. Tout est résumé sur la figure page suivante.

Le compteur colonne est composé de deux compteurs : *compt_pixel* et *compteur*. Le compteur *compt_pixel* est incrémenté à chaque front montant de l'horloge clock si le signal *conv_enable* est à l'état bas. Le signal d'écriture dans la mémoire fifo *wr_fifo* est à l'état actif dès que la machine d'état associée à ce compteur n'est plus dans l'état de repos *etat_fin_lecture*. Dès la fin de la lecture du 3$^{ème}$ pixel, le signal *fin_lecture* est mis à l'état haut, ce qui provoque l'incrémentation du compteur ligne.

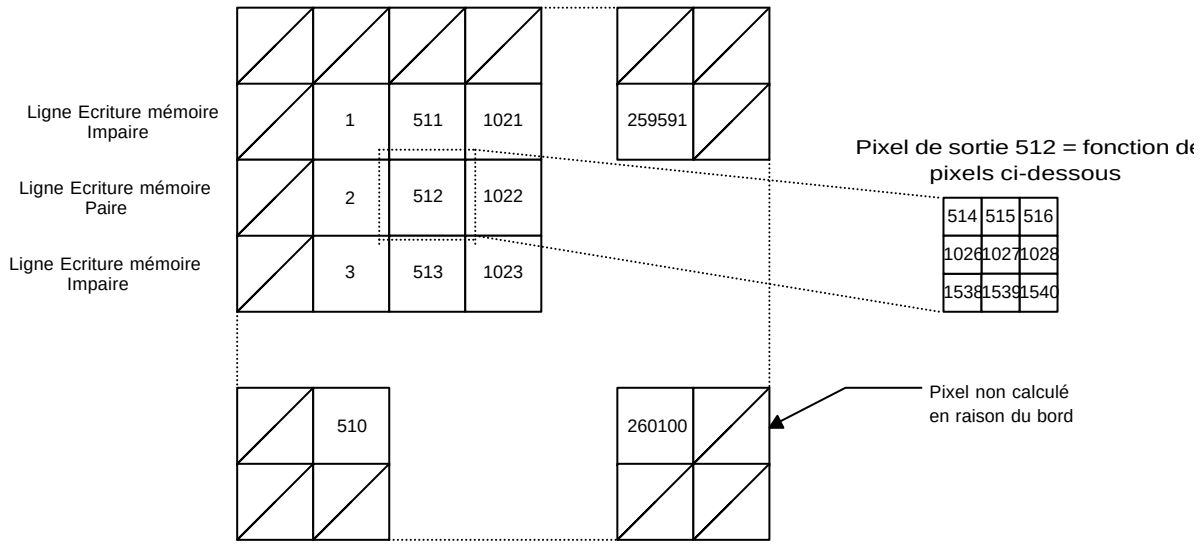
A la fin de chaque colonne, le compteur *compteur* est incrémenté, jusqu'à atteindre la dernière colonne valide de la matrice d'entrée. A ce moment-là, l'image entière a été traitée, et on recommence un cycle complet de traitement.

## LECTURE DE L'IMAGE D'ENTREE




**Nombre d'accès mémoire = 510*9 + 509*510*3**

## ECRITURE DE L'IMAGE DE SORTIE




**Nombre de calculs = nombre d'écriture mémoire = 260100 = 510*510**

Numérotation 1,2,3... ordre de calcul et d'écriture des pixels dans la mémoire de sortie.

*Figure 10: Lecture et écriture dans les mémoires*

#### 2.2.2.5 ETUDE DU COMPOSANT *COMPTEUR_SORTIE* :

Cette entité intègre le compteur de sortie utilisé pour écrire les résultats des calculs dans la mémoire de sortie. Ce compteur est scindé en deux compteurs correspondant aux coordonnées du pixel dans la mémoire, la coordonnée (x) générée par le compteur 'sortie_colonne', la coordonnée (y) par le compteur 'sortie_ligne'.

Dès la fin d'un calcul, le signal *enable_write* ( = *fin_calcul* ) étant mis à l'état haut, la machine d'état passe dans l'état écriture puis le compteur est incrémenté pour stocker le prochain résultat de calcul. A chaque fin de colonne, le compteur colonne est alors incrémenté.

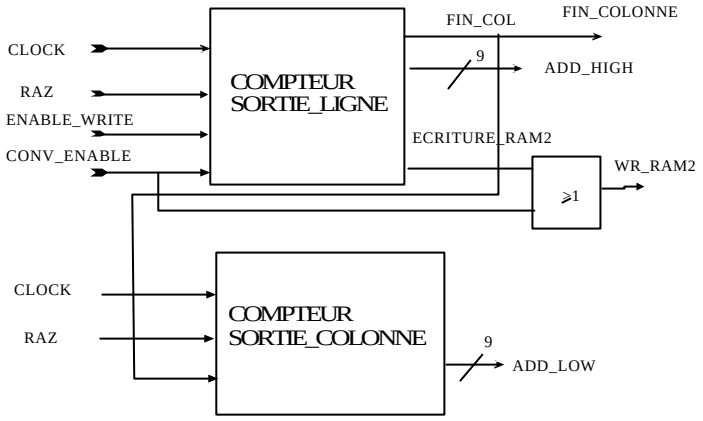



*Figure 11: Représentation graphique des entités interconnectées*

#### 2.2.2.6 ETUDE DU COMPOSANT *MUX* :

Cette entité effectue le multiplexage des bus d'adresses des mémoires. Ces bus sont partagés entre les compteurs associés à l'entité de calcul et les compteurs d'adresses utilisés lors de la phase de conversion. Le signal *conv_enable*, en étant envoyé sur des buffers 74HC244, est utilisé pour gérer l'accès aux bus de données des mémoires entre les convertisseurs (CNA et CAN) et les composants programmables.

### 2.2.3 SYNTHESE DES CIRCUITS PROGRAMMABLES :

Après la simulation comportementale, le fonctionnement des circuits couplés correspondait à ce qui était attendu. Voyons tout d'abord la synthèse des deux circuits avant d'approfondir l'aspect simulation ensuite puisqu'alors interviendront les retards. Etant donné la fréquence d'horloge élevée du signal *clock*, 20 MHz, et le fait que le design utilise les deux fronts de l'horloge, les retards dus aux composants programmables sont alors critiques.

#### 2.2.3.1 SYNTHESE DU CIRCUIT calcul :

La synthèse s'effectue avec Galileo. Au niveau du design d'entrée, on spécifie comme fichier **top.vhd**, au format VHDL. Dans la boîte de dialogue « VHDL Options », on spécifie comme *TOP ENTITY* : **top** et comme *ARCHITECTURE* : **struct**. Au niveau du design de sortie, on choisit le format XNF et comme technologie : XC4000E.

On spécifie également un fichier de contraintes, **top.ctr**, pour forcer le routage au niveau des broches du Xilinx. Voici le résultat de la synthèse par Galileo :

```
Start optimization for design .work.top.struct
Using default wire table: 4025e-3
Pass     Area    Delay     DFFs  PIs   POs --CPU--
         (FGs)    (ns)                    min:sec
 1        224       84      92    15     9   00:29
Info, Added global buffer BUFG for port clock
```

On lance ensuite Design Manager pour obtenir l'occupation du composant dans un circuit X4006e, en choisissant comme format de sortie pour la simulation le VHDL. Le placement - routage nous donne les résultats suivants :

```
Number of External IOBs             23 out of 61     37%
      Flops:                            8
      Latches:                          0
Number of Global Buffer IOBs         1 out of 8      12%
      Flops:                            0
      Latches:                          0

Number of CLBs                     151 out of 256    58%
   Total CLB Flops:                 84 out of 512    16%
   4 input LUTs:                   229 out of 512    44%
   3 input LUTs:                    40 out of 256    15%

Number of PRI-CLKs                   1 out of 4      25%
```

L'important est d'avoir une estimation de la taille en FGs occupée par chaque partie du composant avec Galileo.

| | Nombre de FGs | Pourcentage utilisé |
|---|---|---|
| fifo | 20 | 10% |
| automate de traitement | 5 | 2% |
| algorithme de gradient | 132 | 59% |
| algorithme *'emboss'* | 65 | 29% |

On voit que le circuit n'est occupé qu'à 58% actuellement, et il reste donc de la place pour ajouter de nouveaux algorithmes.

#### 2.2.3.2 SYNTHESE DU CIRCUIT ram_control :

La synthèse s'effectue avec Galileo. Au niveau du design d'entrée, on spécifie comme fichier **ram_control.vhd**, au format VHDL. Dans la boîte de dialogue « VHDL Options », on spécifie comme *TOP ENTITY* : **ram_control** et comme *ARCHITECTURE* : **struct**. Au niveau du design de sortie, on choisit le format XNF et comme technologie : XC4000E.

On spécifie également un fichier de contraintes, **ram_control.ctr**, pour forcer le routage au niveau des broches du Xilinx.

Voici le résultat de la synthèse par Galileo :

```
-- Start optimization for design .work.ram_control.struct
```

```
Using default wire table: 4025e-3

      Pass     Area    Delay     DFFs  PIs   POs --CPU--
               (FGs)    (ns)                    min:sec
      1        184       33        98     7    51   00:52
Info, Added global buffer BUFG for port clk_pixel
Info, Added global buffer BUFG for port clock
Info, Added global buffer BUFG for port csync
```

On lance ensuite Design Manager pour obtenir l'occupation du composant dans un circuit X4006e, en choisissant comme format de sortie pour la simulation le VHDL. Le placement - routage nous donne les résultats suivants :

```
Number of External IOBs            55 out of 61     90%
      Flops:                           2
      Latches:                         0
Number of Global Buffer IOBs        3 out of 8      37%
      Flops:                           0
      Latches:                         0

Number of CLBs                    101 out of 256    39%
   Total CLB Flops:                96 out of 512    18%
   4 input LUTs:                  186 out of 512    36%
   3 input LUTs:                   17 out of 256     6%
Number of PRI-CLKs                 2 out of 4      50%
Number of SEC-CLKs                 1 out of 4      25%
Number of STARTUPs                 1 out of 1     100%
```

Lors du placement-routage, en fixant les options suivantes :

```
Options> Optionnal Targets> Produce Timing Simulation Data
Options>Program option template> Edit template> interface> Format
vhdl
```

On obtient les fichiers time_sim.vhd et time_sim.sdf nécessaires pour simuler les circuits avec les retards. Cette simulation servira notamment dans la mise au point de la carte, pour évaluer l'importance des retards dans le fonctionnement.

# 3. MISE AU POINT DE LA CARTE

## 3.1. PRISE EN MAIN DU PROJET

Une fois l'architecture de la carte bien assimilée, les différents éléments de fonctionnement bien étudiés, il était possible d'envisager une procédure de tests. Etant donné que l'on ne connaît pas les pannes à l'avance, il est impossible d'imaginer une procédure ordonnée et détaillée de mesures. Ce sont les pannes décelées qui engendrent les réparations et qui permettent d'effectuer d'autres tests. La démarche pour effectuer les tests et les mesures a donc été empirique.

Ceux-ci demandent l'utilisation d'outils de mesures. Ceux mis à ma disposition sont principalement :

- un oscilloscope analogique/numérique Metrix OX8627, 2 voies et de définition maximale 0,1 µs en numérique
- un générateur de fonctions 20 MHz.
- une caméra délivrant un signal vidéo norme CCIR.
- Une mire analogique CCIR.
- Un analyseur logique Thurby LA4800, 20 MHz.
- Un écran 625 lignes, CCIR.

Ces outils ont été suffisants pour effectuer des mesures et des tests dans le but de faire fonctionner la carte. Toutefois ils sont limités pour effectuer des mesures plus précises, concernant par exemple les retards dans les cycles d'écritures et de lectures des mémoires. La fréquence d'échantillonnage étant de 5 ou 10 MHz, il serait intéressant d'utiliser notamment un oscilloscope numérique multivoies de précision suffisante pour visualiser de tels signaux (précision supérieure à 0,1 µs).

Ces outils étant répertoriés, on pouvait envisager de commencer la mise au point de la carte. Afin de rester homogène dans la procédure de test , toutes les mesures de fonctionnalité ont été effectuées en échantillonnant les signaux vidéos à **5 MHz**. Ceci permet d'avoir une meilleure précision à l'oscilloscope et diminue les problèmes de retards.

La première étape fut le test des parties analogiques, car le fonctionnement des différents éléments se vérifie très facilement, avec peu de moyens. Une fois cette étape validée il fallait envisager le test des circuits programmables et de leur fonctionnement tant sur le plan logiciel que physiquement sur la carte. A chaque fonction correspond une étape permettant ainsi de scinder le travail en plusieurs parties et d'analyser chaque partie précisément.

## 3.2. AU PREALABLE

Une étude grossière du schematic et des boards a permis de mettre en évidence l'absence de certaines pistes.

Ainsi le signal horloge **clock** n'était généré nulle part dans le schematic et donc absent sur le board. Or une autre horloge existe sur le board. Il s'agit de **clock_pixel** qui est générée par un oscillateur à quartz dont la fréquence est divisée par 2 ou 4 grâce à une bascule.

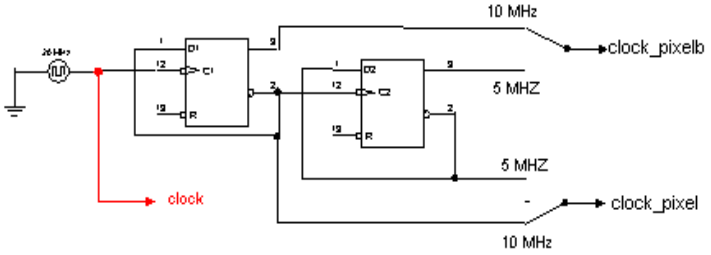



*Figure 12: génération de clock_pixel*

Comme on veut avoir $F_{clock}=2*F_{clock\_pixel}$, on récupère la sortie de l'oscillateur à quartz que l'on relie au entrées **clock** des circuits programmables par un fil.

D'autre part le lay-out de Mentor n'a pas effectué le routage des pistes Chip Enable (CE) des RAMs. Ceci est du à une mauvaise dénomination des signaux dans le schematic. En effet le signal a été nommé CS sur les RAMs et CE sur le circuit RAM_CONTROL, donc le logiciel a considéré 2 signaux différents. Il faut donc réaliser **4 straps** pour connecter les CE des RAMs au Xilinx.

Enfin comme on a décidé d'échantillonner à **5 MHz**, il faut modifier les sources VHDL des Xilinx qui ont été configurées pour 10 MHz.. Le changement de la fréquence implique une modification des compteurs colonnes.

Exemple :

Pour 5 MHz, compteur_colonne va de 32 à 287.

Pour 10 MHz, compteur_colonne va de 64 à 575 .

## 3.3. VALIDATION DES ELEMENTS ANALOGIQUES :

### 3.3.1 ENTREE DU SIGNAL VIDEO :CAN

On effectue une première mesure en court-circuitant le contrôle du gain du CAN commandé par GATEA et GATEB, c'est-à-dire que l'on entre dans le CAN directement en broche 20 au lieu de la broche 16 (cf figure 2 p 6).

On injecte en entrée des niveaux de tensions variants entre 2,59 et 3,59 V. Ces tests ont permis d'avoir une première approche du fonctionnement du CAN, et de vérifier que l'échantillonnage a lieu correctement et que le bus de données ne présente pas de courts-circuits et de micro-coupures.

Une deuxième mesure est appréhendée pour vérifier le fonctionnement des signaux de contrôle du CAN, GATEA et GATEB. Pour la réaliser on envoie un signal vidéo CCIR en entrée. On vérifie dans un premier temps la bonne extraction des signaux de synchronisation du signal vidéo via le LM1881 et leur cheminement jusqu'à RAM_CONTROL.

GATEA et GATEB sont construits à partir de Csync et Burst dans RAM_CONTROL puis envoyés sur le CAN. La visualisation de ces signaux à l'oscilloscope confirme la justesse des algorithmes.

Les données échantillonnées sont difficilement interprétables notamment lorsqu'on envoie en entrée le signal vidéo de la caméra. Le niveau de gris variant à chaque instant , il est impossible de prévoir la valeur des données en sortie. Toutefois **la mire analogique** permet d'obtenir des niveaux constants. Ainsi en injectant un écran noir en entrée on peut interpréter les données échantillonnées : elles sont toutes bien au niveau bas compris entre 0 et 0,2V.

Pour envisager des résultats plus parlants et mieux interprétables, une solution est de réinjecter les données dans le CNA. Ce qui passe par l'étude au préalable de l'étage de sortie.

### 3.3.2 ETAGE DE SORTIE

#### 3.3.2.1 GENERATION DES DONNEES EN SORTIES : MIRE LOGIQUE

Etant donnée l'architecture de la carte, il est possible de gérer des données via le Xilinx CALCUL et les transmettre au CNA.

Pour générer les signaux de synchronisation, on utilise ceux d'un signal vidéo mis en entrée dont on ne tiendra pas compte des données échantillonnées. D'autre part le signal conv_enable, généré par RAM_CONTROL, est récupéré et injecté dans CALCUL . Ce signal comme on peut le voir en figure 13 permet d'effectuer un comptage des colonnes en tenant compte des signaux de synchronisation.

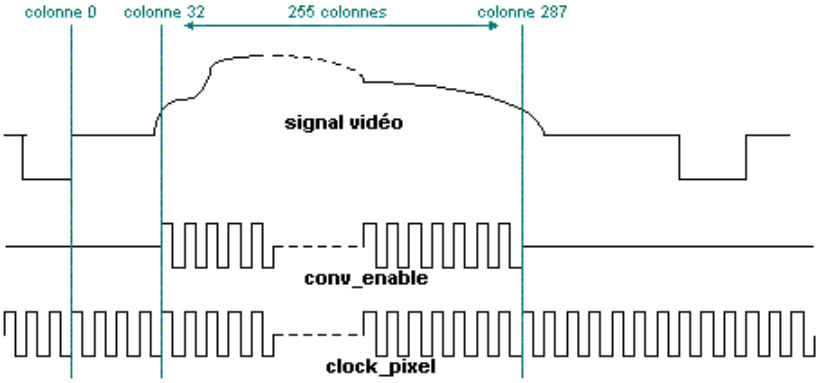



*Figure 13: :allure de conv_enable*

Ceci permet de créer une mire logique pour le circuit CALCUL dont la source VHDL se trouve en annexe 6 dans le programme **mire.vhd** et qui est synthétisé sous le nom **mire.bit**..

Le résultat que l'on veut obtenir à la sortie du CNA sur l'écran est le suivant :

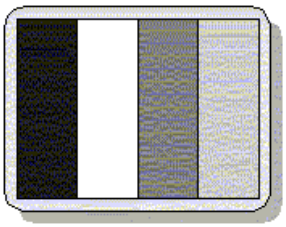


*Figure 14: écran de la mire logique*

Ceci pourra être visualisé sous condition que les autres éléments fonctionnent.

#### 3.3.2.2 SOMMATEUR

Les données de la mire logique envoyées sur le CNA doivent donner en sortie un signal dont l'allure est décrite ci-dessous.

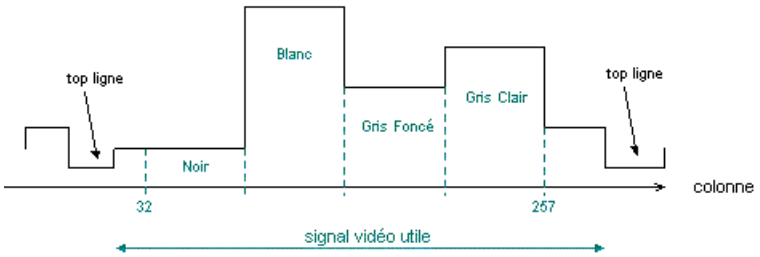



*Figure 15: sortie CNA*

On peut par une analyse des tensions suivre l'évolution de ce signal à partir de la sortie du CNA. Cette analyse a permis de déboucher sur la panne du transistor placé en sortie du CNA et la nécessité de le remplacer. Le sommateur n'a présenté aucune anomalie.

#### 3.3.2.3 MULTIPLEXEUR

Il s'agissait de vérifier que tous les signaux nécessaires à son fonctionnement arrivait bien à lui et en particulier le signal de multiplexage **mux_enable**. Ce qui était le cas.

#### 3.3.2.4 SORTIE ET RESULTAT

Après le multiplexeur la sortie est directement visualisable à l'écran. La mire logique a permis ainsi de déceler un court-circuit sur le bus de données de sorties entre D3 et D7 qui modifiait les niveaux de gris attendus. Une fois ce problème éludé, l'étage de sortie a pu être validé.

#### 3.3.2.5 ENTREE ET SORTIE RELIEES

On peut relier logiciellement le bus de données échantillonnées en entrées aux bus de données de sorties. Il suffit d'effectuer une recopie de pixels dans le Xilinx CALCUL . Le programme se trouve ci-dessous :

```
Graphe : PROCESS(clock, raz)

Begin
     IF (raz ='0') THEN
          pixel_out <= "00000000";
     ELSIF RISING_EDGE(clock) THEN
          pixel_out <= pixel_in;
     END IF;

END PROCESS Graphe;
```

D'autre part il faut court-circuiter les buffers pour relier les convertisseurs au circuit CALCUL en permanence. Pour réaliser ceci il faut laisser les buffers passant en faisant **conv_enable = conv_enableb = 0**. Cette modification intervient dans la source vhdl concernant RAM_CONTROL ; ram_essai.vhd

L'image visualisée sur l'écran est celle attendue mais n'est pas complètement nette. Les contours notamment restent flous.

Ce problème vient de la fréquence d'échantillonnage. En effet la bande passante du signal vidéo CCIR est comprise entre 5 et 6MHz. Le signal est filtré par un filtre passe bas de fréquence de coupure 6,5 MHz. Pour retrouver le signal initial, il faudrait que la fréquence d'échantillonnage $f_e$ soit supérieure à $2f_m$, avec $f_m$ la bande passante du signal.

Or ce n'est pas le cas puisque l'on a pris **$f_e$=5 MHz**. Donc le théorème de Shannon n'est pas respecté et on a un recouvrement spectral du signal vidéo. Ceci se traduit par un flou de l'image notamment lors d'un changement brusque de niveau de gris.

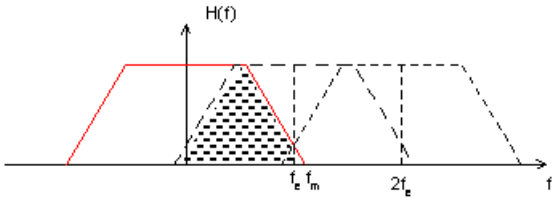



*Figure 16: spectre du signal et échantillonnage*

Ce n'est pas gênant pour valider le fonctionnement de la carte. Mais il est clair que le traitement de l'image ne peut se faire à cette fréquence. En échantillonnant à 10 MHz on devrait respecter Shannon, pouvoir ainsi reconstruire le signal proprement, et ainsi envisager le traitement de l'image.

## 3.4. MIS AU POINT DES CIRCUITS PROGRAMMABLES

La carte comporte deux circuits nommés **RAM_CONTROL** et **CALCUL** :

- - **CALCUL** s'occupe du traitement des données. Celles-ci sont lues dans la mémoire d'entrées, stockées dans une mémoire tampon de type FIFO et traités. A priori ceci ne présente pas de grande difficulté.
- **RAM_CONTROL** est chargé de la gestion des mémoires. Il doit donc gérer les cycles d'écritures, de lectures et le calcul des adresses. C'est le cœur de la carte et il doit être au point pour envisager le traitement des données.

### 3.4.1 GESTION DES MEMOIRES

Le fonctionnement de certaines entités a pu déjà être validé à travers la visualisation de certains signaux :

- GATEA et GATEB valident **adc_control**.
- Mux_enable valide son équation et le comptage des lignes et des colonnes c'est à dire des entites **compteur_ligne** et **compteur_colonne**.

Il reste donc à valider les autres entités.

#### 3.4.1.1 MEMOIRE D'ENTREE : RAM1

Elle est constituée de deux RAMs 32 ko à 1 accès. Une pour stocker la trame paire et l'autre la trame impaire. Les signaux de contrôle de ces RAMs sont :

- CE\ : chip enable, sélection de la mémoire.
- W/R : sélection écriture, lecture

- OE\ :permet d'inhiber la mémoire pendant la lecture.

**TRUTH TABLE**

| Mode | $\overline{WE}$ | $\overline{CE1}$ | CE2 | $\overline{OE}$ | I/O Operation | Vcc Current |
|---|---|---|---|---|---|---|
| Not Selected | X | H | X | X | High-Z | $I_{SB1}$, $I_{SB2}$ |
| (Power-down) | X | X | L | X | High-Z | $I_{SB1}$, $I_{SB2}$ |
| Output Disabled | H | L | H | H | High-Z | $I_{CC1}$, $I_{CC2}$ |
| Read | H | L | H | L | $D_{OUT}$ | $I_{CC1}$, $I_{CC2}$ |
| Write | L | L | H | X | $D_{IN}$ | $I_{CC1}$, $I_{CC2}$ |

*Figure 17: Table de vérité des RAMs*

Ecriture :

L'écriture dans la mémoire passe par la bonne génération de ces signaux et également par la bonne gestion de l'accès au bus de données entre le convertisseur et les circuits programmables via le signal conv_enable et un système de buffer décrit en figure 20. L'équation de ce signal est :

```
Conv_enable = pixel_valid AND ligne_valid AND clock_pixel
```

- 0 : passage ouvert,les données échantillonnées sont disponibles sur le bus.
- 1 :passage fermé, sortie 3 états.

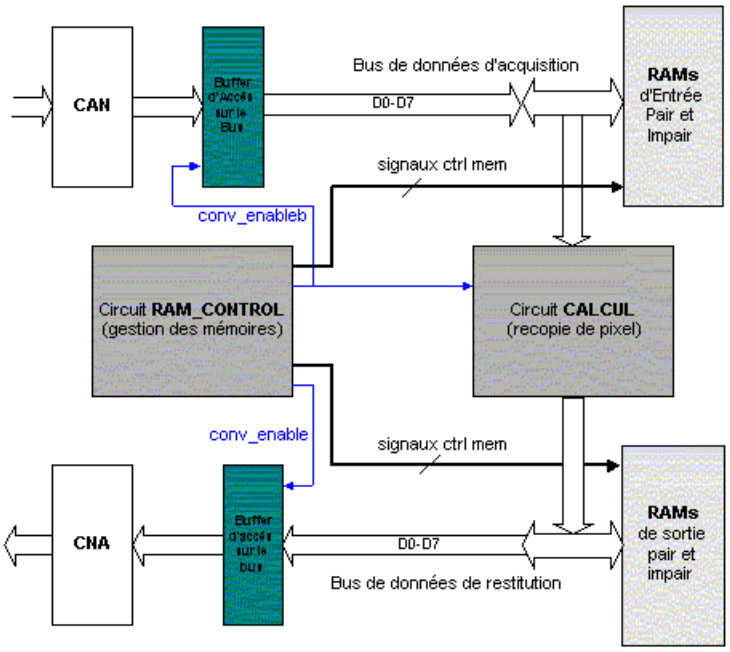



*Figure 18: synoptique de la gestion des mémoires*

On écrit dans la mémoire lorsque les données échantillonnées sont disponibles sur le bus c'est-à-dire lorsque le CS du buffer est à 0.L'équation de **wr_ram1** est la suivante :

```
wr_ram1 = NOT enable_conv AND clock = NOT conv_enable AND clock
```

Les données échantillonnées doivent être disponibles lorsque la mémoire est en écriture c'est-à-dire que CSbuffer doit être égal à 0. Etant donné **wr_ram1** c'est

**conv_enableb** qui doit commander le buffer d'entrée. Le chronogramme de la figure 19 décrit le cycle d'écriture.

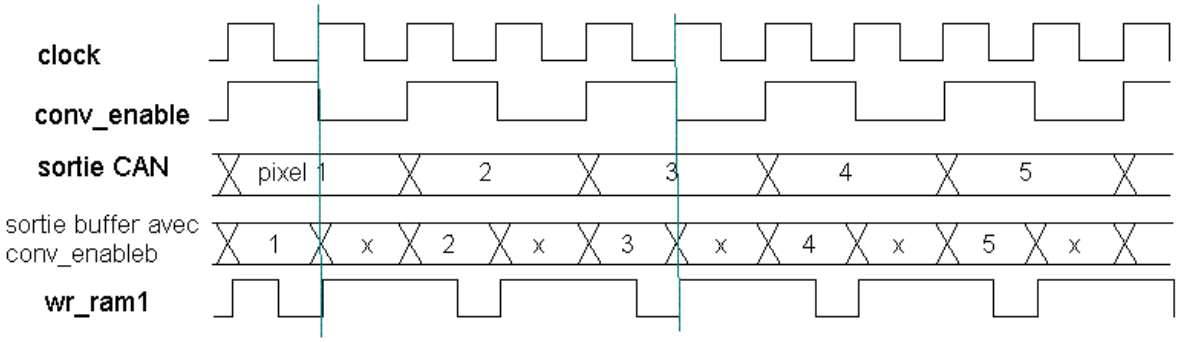



*Figure 19: chronogramme d'écriture dans la mémoire des données*

Concernant **OE** et **CE**, les équations reprennent les bits de poids faibles du bus d'adresse et la sélection se fait correctement.

Lecture :

Pour simplifier le processus et l'analyse et étant donné le programme déjà créé, j'ai choisi de lire les données dans le même ordre d'arrivée que l'écriture. Ainsi on conserve le même calcul d'adresse que lors de l'écriture. Les modifications interviennent dans mux2.vhd.

Les données lues sont transmises au Xilinx CALCUL qui effectue une recopie de pixels, uniquement pendant la lecture, vers le bus de données en sorties.

En court-circuitant le buffer de sortie, celles-ci arrivent directement sur le CNA qui restitue le signal vidéo.

Après multiplexage, on peut visualiser le signal à l'écran. Les corrections apportées donnent un bon résultat et permettent de franchir l'étape suivante qui est la mise au point de la gestion de la mémoire de sortie, RAM2.

#### 3.4.1.2 MEMOIRE DE SORTIE

Pour pouvoir tester cette entité, on utilise le même principe que la mémoire d'entrée.

Les données lues dans la mémoire d'entrée sont disponibles sur le bus de données de sorties. Il suffit de venir les écrire dans RAM2 à la même adresse où elles ont été lues. Pour réaliser ceci on génère **wr_ram2**, selon l'équation :

```
wr_ram2 = enable_conv OR clock
```

Comme pour la RAM d'entrée, les données sont lues après chaque écriture et transmise au CAN.

Le changement de ce calcul d'adresse par rapport au traitement originel intervient dans l'entité mux.vhd. Les modifications sont spécifiés dans le fichier **mux2.vhd** qui se trouve en annexe 4.

On dispose d'une période d'horloge échantillonnage c'est-à-dire 200ns pour effectuer 2 cycles d'écriture-lecture. On peut donc consacrer 50 ns par cycle. Or d'après les datasheets, la durée minimale d'un cycle pour la mémoire est de 20ns. Donc les temps minimaux sont respectés et l'algorithme peut fonctionner. On peut le voir clairement sur le chronogramme de l'annexe 1.

Les modifications apportées ont nécessité de faire une nouvelle synthèse des circuits modifiés. Celle-ci s'effectue comme vu précédemment dans le chapitre 3.2.3 :

Circuit **CALCUL** :

- fichier source : toc.vhd
- fichier brochage :toc.ctr

Circuit **RAM_CONTROL** :

- fichier source : ram_essai.vhd
- fichier brochage :ram_essai.ctr

### 3.4.2 SIMULATION DES CIRCUITS PROGRAMMABLES :

Après le placement - routage avec les options choisies, les fichiers time_sim.vhd et time_sim.sdf ont été créés dans le répertoire associé au design.

Pour le composant CALCUL, puisque le signal *raz* n'est pas routé directement sur le net GSR, il est nécessaire de fixer la durée du GLOBAL RESET qui a été automatiquement inséré par un composant ROC, il faut éditer le fichier time_sim.vhd et mettre en commentaires les 4 lignes depuis :

```
configuration CFG_ROC_V of ROC is  ... end CFG_ROC_V ;
```

et les remplacer par la ligne suivante (Dans l'architecture STRUCTURE OF TOP, après la déclaration de COMPONENT ROC):

```
FOR ALL:ROC USE ENTITY work.roc(roc_v) GENERIC MAP(width=>100 ns);
```

Il faut rajouter la bibliothèque SIMPRIM par la commande :

```
qhmap simprim /net/xilinx/mentor/data/vhdl/simprim
```

Pour tester les composants CALCUL et RAM_CONTROL, on utilise une entité de test en VHDL. Cette entité (dans le fichier *test.vhdl*) effectue le test des deux entités interconnectées comme le seront les deux composants programmables sur la carte. Le process 'test' génère les signaux de synchronisation en respectant les caractéristiques du signal vidéo

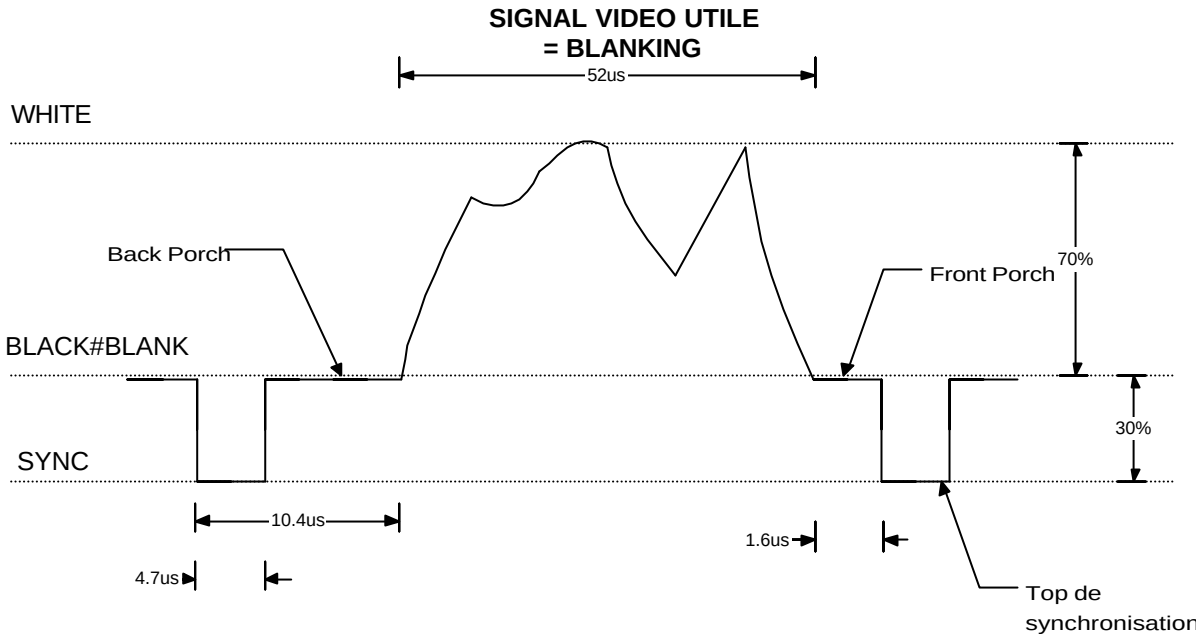



*Figure 20: Caractéristiques du signal vidéo*

On a ainsi défini les temps Tp_csync = 4.7 µs, Tp_vsync = 2 µs, Tp_burst = 2.4 µs et Tp_ligne = 52 µs.

Les données sont générées par un compteur incrémenté à chaque coup d'horloge. La simulation est lancée avec l'option de prise en compte des retards :

```
qhsim -sdftyp /u1=time_sim_top.sdf -sdftyp /u0=time_sim_ram.sdf test_mix
```

Après chargement du design, il faut changer la durée de la simulation dans le menu Options → Properties → Default Run. Le résultat se trouve en annexe 1.

Celui-ci permet de voir qu'en échantillonnant à 5 MHz les retards ne sont pas critiques.

Après téléchargement des fichiers **ram_essai.bit** et **toc.bit** dans les 2 Xilinx on peut tester la carte et visualiser le résultat à l'écran. L'image obtenue est celle attendue donc on peut dire que la gestion des mémoires par le circuit RAM_CONTROL est validée.

Conclusion :

Les modifications et les tests effectués ont permis d'aboutir au fonctionnement de tous les éléments physiques de la carte. Les points qui restent à modifier sont d'ordres logiciels notamment pour échantillonner à 10 MHz et envisager le traitement d'image tel qu'il était prévu originellement.

## 3.5. PASSAGE A 10 MHZ

Les algorithmes ont été faits pour traiter une image de 512*512 pixels, ce qui nécessite d'échantillonner à 10 MHz. De plus ceci devrait permettre de supprimer les flous observés à 5 MHz. Les seules modifications à apporter sont d'ordres logiciels mis à part le changement de

quartz, 20 MHz au lieu de 10 MHz. Il faut modifier tous les compteurs de colonnes en les multipliant par **2±e.**

Une fois ces changements effectués, on peut télécharger les deux Xilnx avec les fichiers **ram_control.bit** et **toc.bit**. On effectue une simple transmission des données échantillonnées vers la sortie sans passer par les mémoires. Le résultat à l'écran n'est pas celui attendu. En effet il y a des problèmes de synchronisation. Certains tops trames semblent disparaître et font sauter l'image à des intervalles irréguliers. Plusieurs hypothèses ont été avancées pour expliquer ces phénomènes :

- pertes des signaux de synchro du au LM1881. Le niveau de csync est peut-être insuffisant et certains tops peuvent échapper lors de la lecture du signal .Mais en échantillonnant à 5 MHz, on devrait retrouver le même problème, ce qui n'est pas le cas .
- erreurs dans les compteurs : fort possible car en plus le signal de norme CCIR contient 312,5 lignes par trame. Plusieurs configurations ont été essayées mais aucune n'a donné de résultats satisfaisants
- bruit quelconque modifiant les tops lignes et tops trames.

Le problème n'a pas été resolu et m'a empêché d'avancer, c'est-à-dire d'envisager le test des algorithmes de traitement d'image.

D'autre part, en effectuant une simulation des deux circuits interconnectés en tenant compte des retards, on remarque que ceux-ci sont critiques lors de l'écriture en mémoire de sortie. On peut le voir en annexe 3 en comparaison avec l'annexe 2 qui est la simulation comportementale.

# 4. AMELIORATIONS ENVISAGEABLES

Afin de palier a certains oublis dans le layout, il a fallu rajouter quelques fils .Les modifications suivantes pourront être apportées dans le schematic sous Mentor :

- - relier les Xilinx à l'horloge clock
- - rajouter les pistes CE des 4 RAMS .
- changer le brochage des connecteurs Xilinx pour le téléchargement afin de simplifier la tâche de l'utilisateur. On peut même imaginer un téléchargement du type Daisy-Chain, afin de charger les deux Xilinx en une opération.

## 4.1. CONFIGURATION DAISY-CHAIN

Les concepteurs de la carte ont prévu des PROMs pour charger les 2 Xilinx 4006E lors de la mise sous tension. Ceci est en effet pratique pour une version définitive de la carte.

Pour la mise au point il a été prévu un connecteur 14 broches pour chaque FPGA. A chaque mise sous tension il faut donc câbler un connecteur, télécharger un Xilinx, puis recommencer l'opération pour le deuxième Xilinx. Ce qui constitue une énorme perte de

temps et également une source de court-circuit. La configuration **Daisy-Chain** permettrait de télécharger les fichiers des deux FPGA en une seule opération.

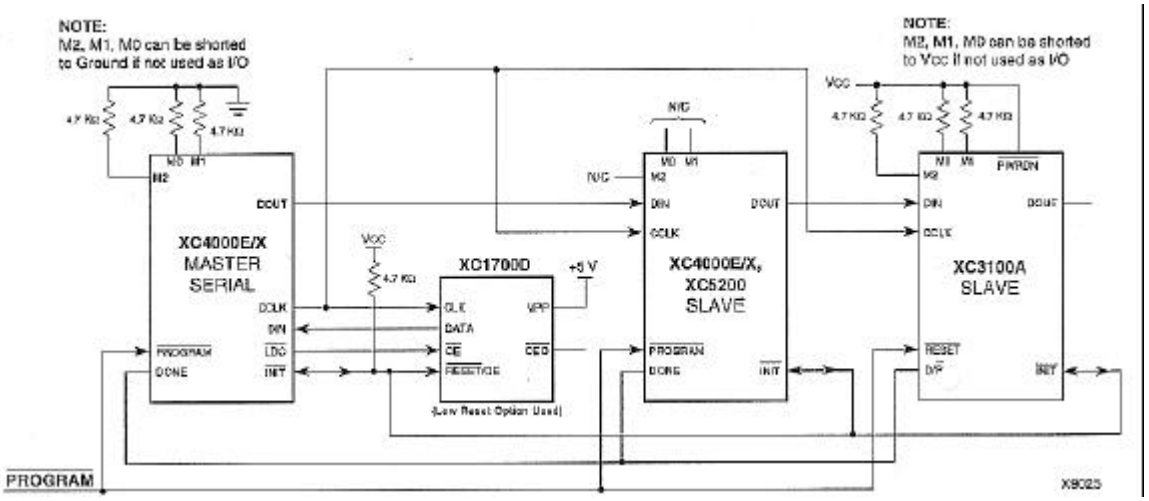



*Figure 21: Exemple de configuration Daisy-Chain*

Il faut tout d'abord créer un unique fichier type chaîne de bit ( .bit) résultant de la concaténation des fichiers synthétisés des deux circuits.

Le **DOUT** du 1er circuit, qui sera le circuit maître doit être relié sur le **DIN** du 2ème appelé circuit esclave. De plus les deux circuits doivent avoir leur pin **CCLK** connectée en parallèle pour avoir un temps commun de référence et leur pin **DONE** connectée ensemble garantissant que les circuits sont actifs uniquement quand les deux sont chargés.

Les deux Xilinx doivent être configurés en mode Slave Serial. On peut alors charger les FPGAs par le câble et Xcheker.

## 4.2. INTERFACE AVEC MICROCONTROLEUR

Une interface avec un ordinateur hôte serait intéressante. Le gestion de l'interface est alors prise en charge par un microcontrôleur type 68HC11. On disposera alors de plusieurs modes de fonctionnement :

- sélection de la fréquence d'échantillonnage : 5 ou 10 MHz, en modifiant la valeur d'un registre interne au composant programmable,
- télécharger dans la mémoire d'entrée une image stockée sur l'ordinateur hôte (pour le test ou le traitement d'une image) ou de la carte vers l'ordinateur (pour récupérer une image acquise),
- télécharger dans la mémoire de sortie une image stockée sur l'ordinateur (pour visualiser sur le moniteur de contrôle) ou de la carte vers l'ordinateur (pour récupérer le résultat du traitement d'une image en mode test par exemple),
- téléchargement des algorithmes de traitement dans la mémoire de configuration du composant programmable...

Afin d'accélérer le temps de traitement, on pourrait prévoir plusieurs unités de traitement en parallèle, ce qui rendrait possible le traitement d'images couleur.

# 5. CONCLUSION

Au terme de mon stage, l'objectif premier n'a pas été entièrement atteint puisque des problèmes subsistent. Bien que la carte soit en bonne voie de fonctionnement, l'échantillonnage à 10 MHz pose problème et m'a empéché d'envisager le traitement d'images à cette fréquence. Si le problème reste insoluble, on pourra toujours se ramener au traitement d'une image 512*256 pixels échantillonnée à 5 MHz où là, la carte est validée.

Une grande partie du stage a été consacrée à l'étude de la carte afin de bien en saisir tous les éléments pour pouvoir la tester. Aussi, j'ai pu me rendre compte de la difficulté à reprendre un travail conçu par une tiers personne et notamment de l'importance de la fiabilité des documents qui sont laissés au terme de celui-ci, car en effet ils ont constitués la base de mon travail.

# REFERENCES BIBLIOGRAPHIQUES :

## LIVRES TECHNIQUES


- R. Besson, *Cours de Télévision Moderne,* Editions Radio
- D. Bouveron et F. Renahy : *Rapport de projet de fin d'études-Carte d'acquisition video pour l'étude d'algorithmes de traitement d'images,* 1998
- T.S. Rzeszewski : *Digital video-concepts and Application accross Industries.* 1995
- Patrice NOUEL, *Implantation Xilinx d'un circuit décrit en VHDL*


## REFERENCES INTERNET


- CircuitWorld Online Services, *http://www.circuitworld.com/*
- Chip Directory, *http://www.chipdir.com*
- WEBLinx, *http://www.xilinx.com*
- Farnell, *http://www.farnell/belgium*

# ANNEXES :

Annexe 1: Simulation avec retard (5 MHz).

Annexe 2: Simulation fonctionnelle (10 MHz).

Annexe 3: Simulation avec retard (10 MHz).

Annexe 4: Schéma généré par Renoir : interconnexion de RAM_CONTROL et TOP.

Annexe 5: Sources VHDL originelles des circuits CALCUL et RAM_CONTROL.

Annexe 6: Sources VHDL modifiées pour tester et échantillonner à 5 MHZ

Annexe 7: Schéma de la carte d'acquisition vidéo

Annexe 8: Plan d'équipement de la carte.