Université de Bourgogne

Faculté Mirande

Rapport de projet

Licence Electronique, Signal et Image

Réalisation d'une carte de traitement d'image à base de FPGA

Guillaume Lemaître

Janvier 2009

SOMMAIRE :

SOMMAIRE DES FIGURES :

INTRODUCTION :

Ce manuscrit est la deuxième partie du projet portant sur la réalisation d'une carte de traitement d'image à base de FPGA. Dans le premier manuscrit, nous avons étudié et détaillé le fonctionnement électronique des deux différentes cartes qui composent le projet. Dans ce second rapport, nous détaillerons la partie programmation du FPGA qui permettra d'effectuer le traitement d'image désiré.

Ce manuscrit se décompose en deux parties distinctes. Dans un premier temps, nous ferons un rappel sur la partie électronique de la carte FPGA qui sera utile à l'entière compréhension de la partie programmation.

Dans un second temps, nous étudierons la partie programmation qui permettra le traitement d'image.

1 Rappel sur la partie électronique de la carte FPGA :

1.1 Schéma structurel :

Nous pouvons apercevoir sur la figure 1, cinq blocs différents communiquant avec le FPGA :

- Les données vidéo et les signaux de synchronisation entrant provenant de la carte d'acquisition :
    - Les données numériques de D0 à D7 des broches 1, 3, 5, 7, 9, 11, 13 et 15 du bornier VIDEO IN sur les broches d'entrée 24, 27, 26, 29, 28, 44, 38 et 40 du FPGA. Nous appellerons ces broches du FPGA, bus **IN_FPGA**.
    - Le signal d'horloge est envoyé la broche d'entrée 35 du FPGA. Nous appellerons cette broche du FPGA, broche **Horloge_pixel**.
    - Le signal de synchronisation de ligne est envoyé sur la broche d'entrée 39 du FPGA. Nous appellerons cette broche du FPGA, broche **Synchro_Ligne**.
    - Le signal de synchronisation de trame est envoyé sur la broche d'entrée 46 du FPGA. Nous appellerons cette broche du FPGA, broche **Synchro_Trame**.
- Les données vidéo sortant en direction de la carte d'acquisition :
    - Les données numériques de D0 à D7 sortent des broches de sortie 25, 20, 23, 18, 19, 16, 17 et 15 du FPGA en direction des broches 1, 3, 5, 7, 9, 11, 13 et 15 du bornier VIDEO OUT. Nous appellerons ces broches du FPGA, bus **OUT_FPGA**.
- La liaison entre la RAM et le FPGA :
    - L'adressage d'A0 à A16 des broches de sortie 77, 78, 79, 80, 81, 82, 83, 84, 3, 4, 5, 6, 7, 8, 9, 10 et 13 sur les broches d'adressage 12, 11, 10, 9, 8, 7, 6, 5, 27, 26, 23, 25, 4, 28, 3, 31 et 2 de la RAM. Nous appellerons ces broches du FPGA, bus **ADD_RAM**.
    - Les données bidirectionnelles de D0 à D7 des broches d'entrée/sortie 71, 69, 67, 65, 61, 59, 58 et 56 du FPGA sur les broches d'entrée/sortie 13, 14, 15, 17, 18, 19, 20 et 21 de la RAM. Nous appellerons ces broches du FPGA, bus **DON_RAM**.
    - La sortie 53 du FPGA est reliée à la broche d'entrée d'activation des sorties de la RAM qui est la broche 24. Nous appellerons cette broche du FPGA, broche **OE**.
    - La sortie 72 du FPGA est reliée à la broche d'entrée d'écriture/lecture de la RAM qui est la broche 29. Nous appellerons cette broche du FPGA, broche **WE**.
- La liaison entre les Switch et le FPGA :
    - Du Switch SW0 à SW7 est reliée aux bornes d'entrée 51, 57, 60, 62, 66, 68, 70 et 49 du FPGA. Nous appellerons ces broches du FPGA, bus **SWITCH** quand nous utiliserons tous les Switch ensemble.

- La liaison entre le port parallèle et le FPGA :
    - La broche 1 - DIN du port parallèle connectée à l'entrée DIN qui est la broche 71 du FPGA.
    - La broche 2 - CCLK du port parallèle connectée à l'entrée CCLK qui est la broche 73 du FPGA.
    - La broche 3 – PROG du port parallèle connectée à l'entrée PROG qui est la broche 55 du FPGA.
    - La broche 4 – DP du port parallèle connectée à l'entrée DP qui est la broche 53 du FPGA.
    - La broche 5 du port parallèle connectée à l'entrée TDO qui est la broche 73 du FPGA.

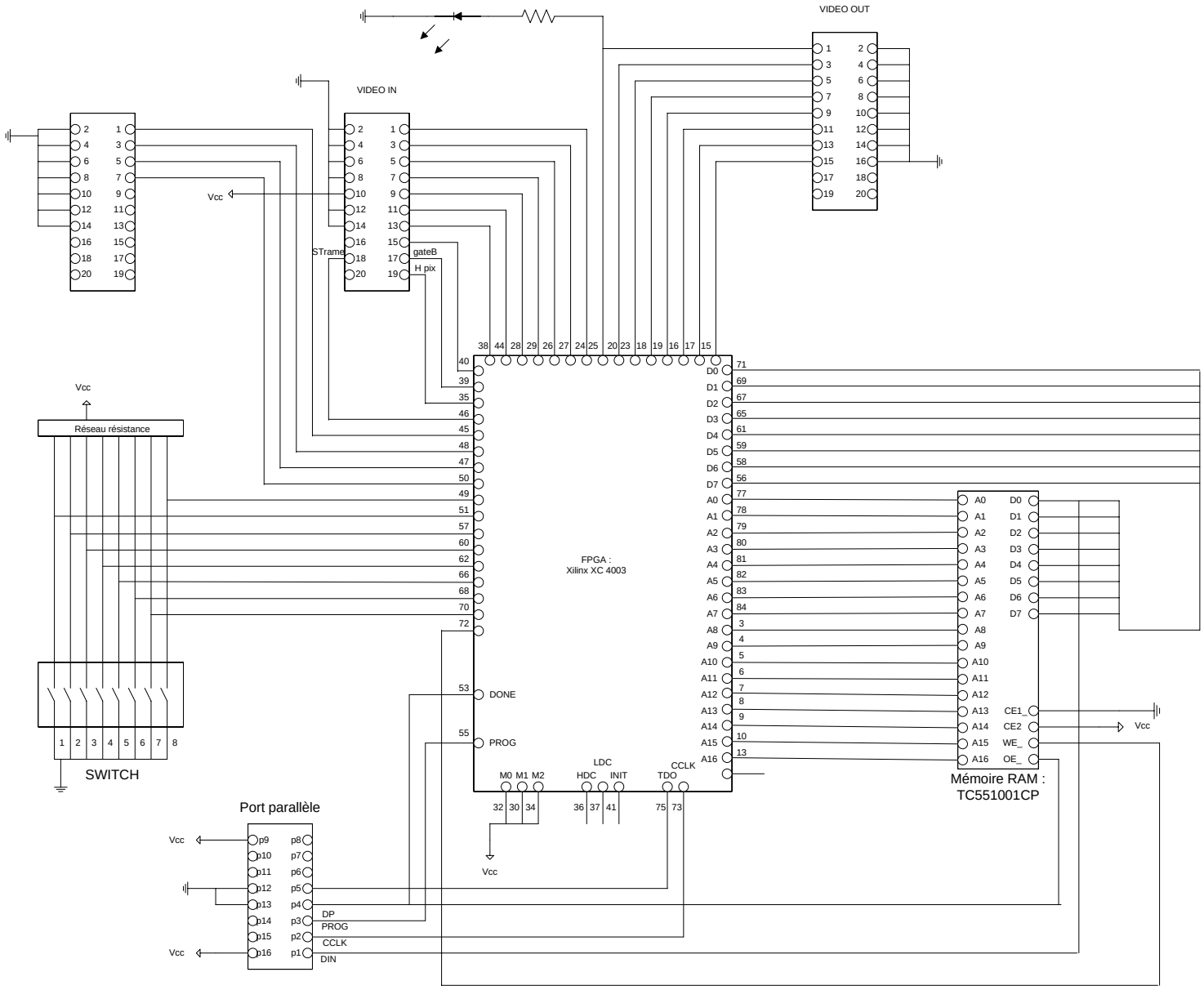

Figure 1 : Schéma structurel du FPGA

1.2 Fonctionnement du FPGA :

Le but du FPGA est de traiter les données numériques d'entrées du bus FPGA_IN. Après traitement, nous renvoyons les données numériques traitées sur le bus FPGA_OUT. Afin de travailler avec deux images à la fois, nous avons mis à disposition une RAM pour stocker une image. Pour cela nous avons un bus bidirectionnel DON_RAM qui sert à envoyer les données numériques dans la RAM ou à recevoir les données numériques de la RAM. Ces données sont stockées et lues grâce à un adressage géré par le FPGA à travers le bus ADD_RAM.

1.3 Fonctionnement de la RAM :

1.3.1 Caractéristiques :

La RAM que nous possédons à une capacité de 128 Ko. Nous utiliserons cette RAM seulement en mode lecture ou en mode écriture.

Pour que la RAM soit utilisée en mode lecture il faudra que le brochage soit le suivant :

WE = 1 ($+V_{cc}$), CE1 = 0 (0V), CE2 = 1 ($+V_{cc}$), OE = 0 (0V).

Pour que la RAM soit utilisée en mode écriture il faudra que le brochage soit le suivant :

WE = 0 (0V), CE1 = 0 (0V), CE2 = 1 ($+V_{cc}$), OE = X.

1.3.2 Détermination de fonctionnement :

Nous ne possédons que de 128 Ko, or pour sauvegarder une image pour le traitement du signal, nous avons besoins de 512 bits, qui correspondent aux 512 pixels, par lignes. D'où :

$$512 \times 625 = 320\ Ko$$

Nous manquons de mémoire. Pour contourner ce problème, lorsque nous aurons besoin de stocker une image, nous garderons alors uniquement un pixel sur deux et seulement une ligne sur deux. Ceci revient à prendre un pixel sur deux sur une des deux trames. Nous aurons alors :

$$\left(\frac{512}{2}\right).\left(\frac{625}{2}\right) = 80\ Ko$$

2 Partie programmation permettant le traitement d'image :

Dans cette partie, nous allons expliquer les différents programmes que nous avons réalisés afin de répondre au traitement demandé. La programmation est réalisée à l'aide du logiciel Xilinx project manager. Nous allons tout d'abord présenter rapidement ce logiciel et dans un second temps nous allons nous détailler les différentes programmations que nous avons réalisées.

2.1 Présentation du logiciel de programmation :

Xilinx project manager permet de programmer en plusieurs langages de programmation. Nous avons choisi, le langage le plus simple, basé sur des compétences déjà acquises dans le passé : la logique séquentielle. Le principe est de manipuler différents éléments logiques, tel que des portes logiques, afin d'obtenir un résultat conforme au cahier des charges.

Nous allons succinctement expliquer les étapes qui nous permettront de programmer le FPGA.

2.1.1 Réalisation d'un schéma logique :

Tout d'abord, nous devons réaliser un schéma logique à base de porte logique et de bascule. Le menu permettant la réalisation du schéma logique est présenté figure 2 :

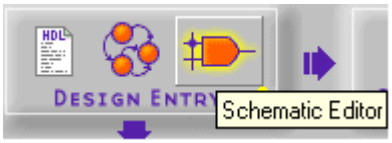

Figure 2 : Menu permettant l'édition du schéma

Les schémas seront de la forme présentés figure 3 :

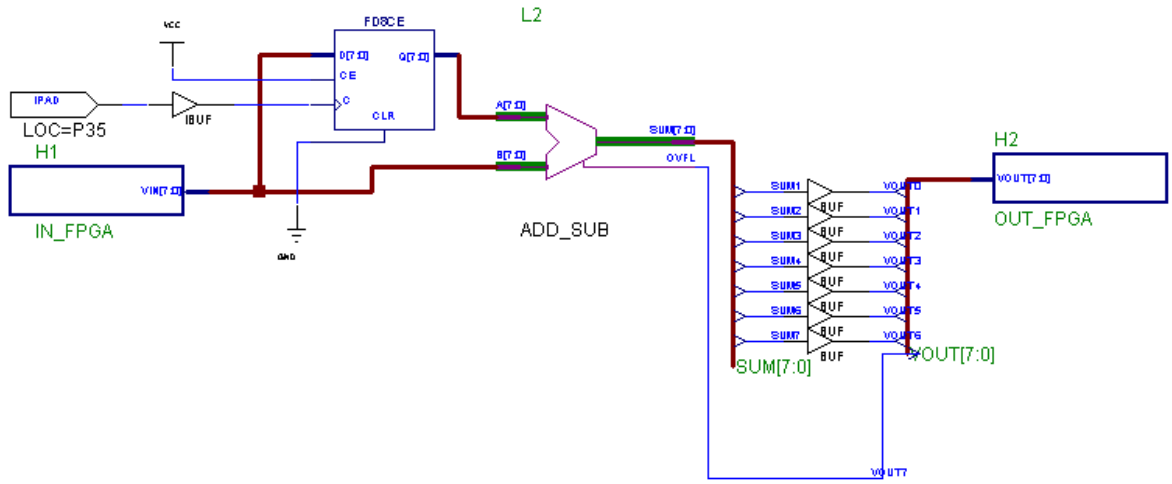

Figure 3 : Exemple de schéma logique

2.1.2 Génération d'un schéma logique :

Après avoir réalisé le schéma logique, il faut générer celui-ci afin de connaître les erreurs s'il y en a.

Le menu permettant la génération est présenté figure 4 :

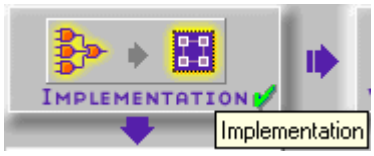

Figure 4 : Menu permettant l'implémentation

Nous avons différentes étapes de génération qui sont présentées figure 5:

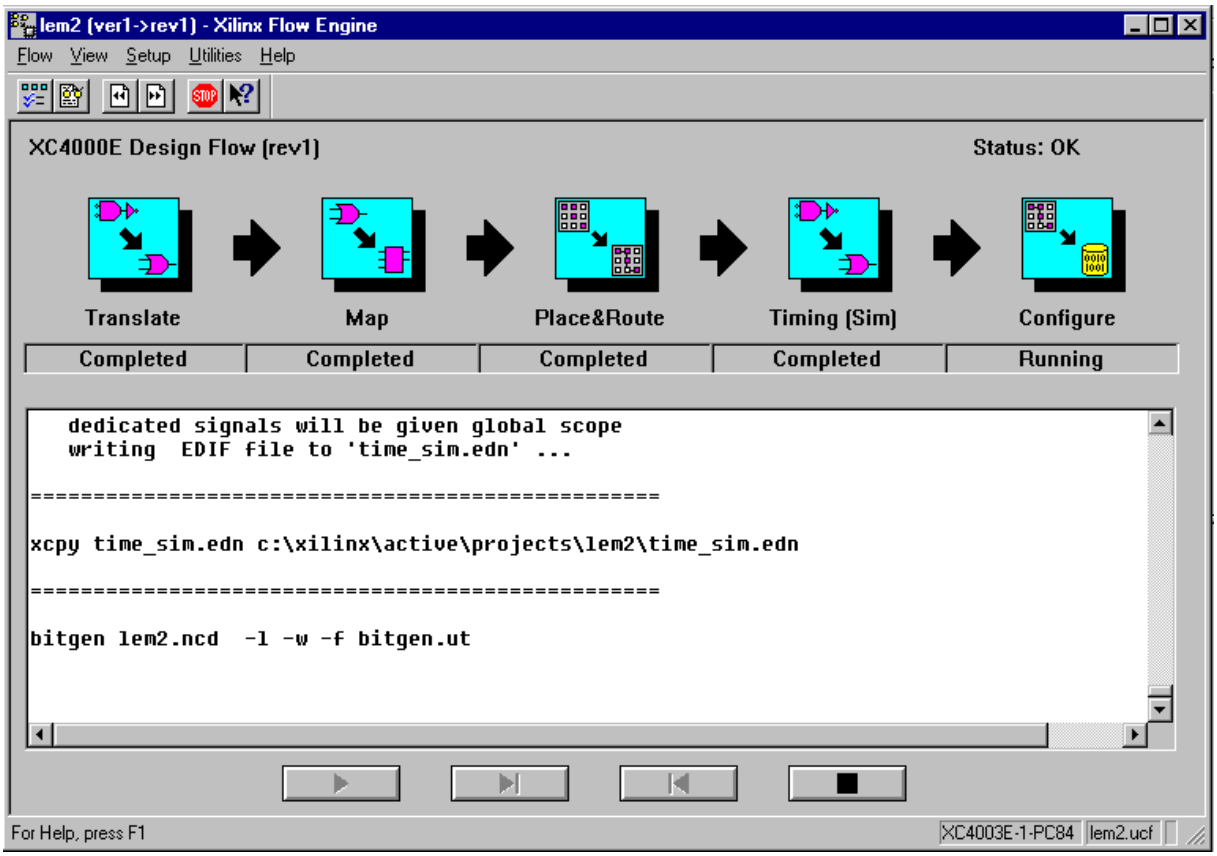

Figure 5 : Etapes lors de la génération

- Translate : traduction du schéma.
- Map : Création du plan.
- Place&Route : placement des éléments et routage.
- Timing : simulation temporelle.
- Configure : génération en binaire.

Lors de la génération, le logiciel créé différents rapports permettant de mettre à disposition des informations importantes à l'utilisateur.

Les différents rapports générés par le logiciel sont présentés figure 6 :

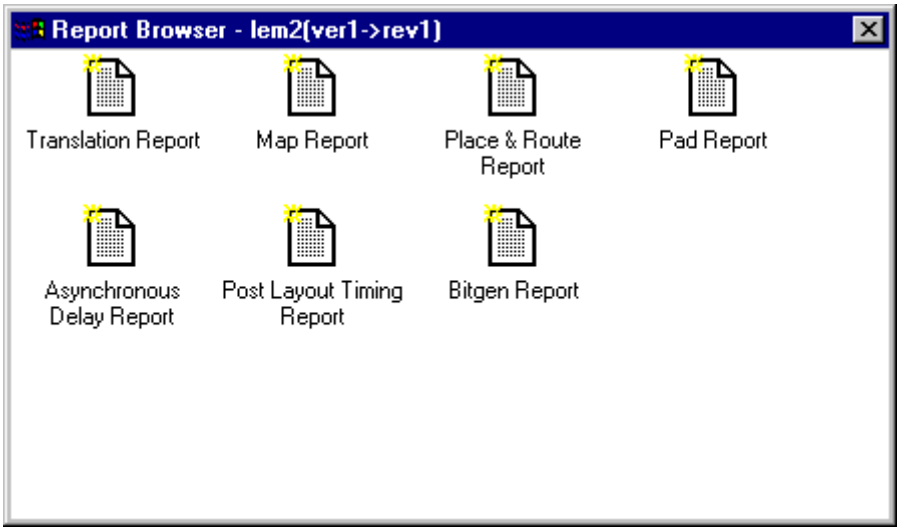

Figure 6 : Rapports créés lors de la génération

- Translation Report.
- Map Report.
- Place&Route Report.
- Pad Report.
- Asynchronous Delay Report.
- Post Layout Timing Report.
- Bitgen Report

Les informations qui nous seront importantes sont :

- Le nombre de broches utilisées.
- Le nombre de CLBs utilisées.
- Le nombre de bascules CLBs utilisées.
- Le temps de propagation d'un signal de l'entrée à la sortie du schéma.

Nous trouvons ces informations dans le rapport Place&Route Report comme présentées figure 7 et 8 :

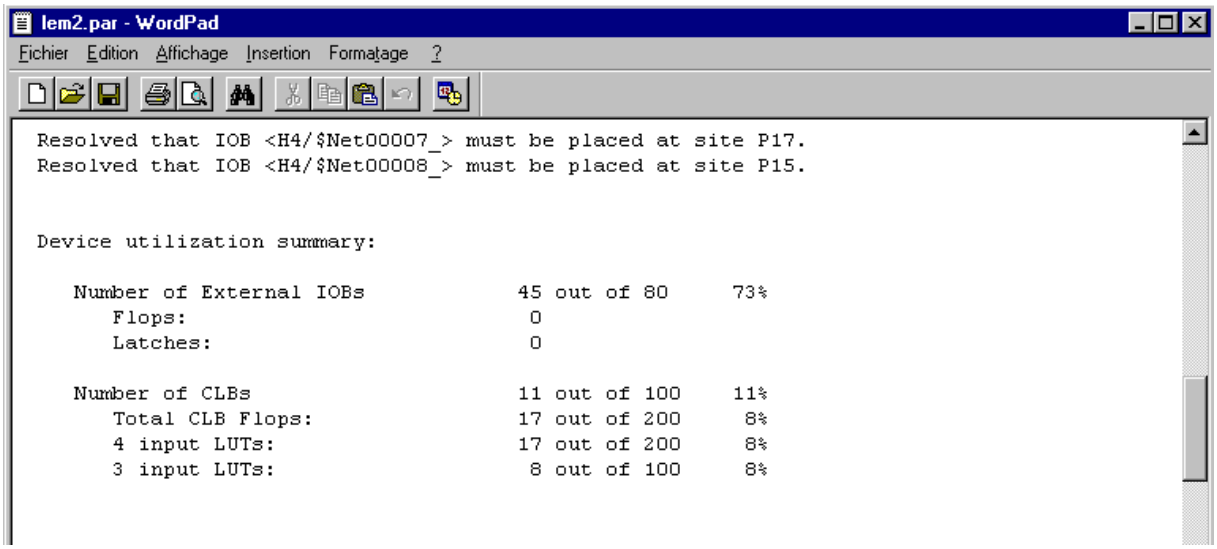

Figure 7 : Informations concernant le nombre d'entrées/sorties utilisées et le nombre de CLBs utilisés

```
The Number of signals not completely routed for this design is: 0

   The Average Connection Delay for this design is:        2.582 ns
   The Maximum Pin Delay is:                               7.089 ns
   The Average Connection Delay on the 10 Worst Nets is:   6.041 ns
```

Figure 8 : Informations concernant le temps de propagation

2.1.3 Programmation dans le FPGA :

Après une génération réussie, nous pouvons alors programmer le FPGA comme présenté figure 9 :

Figure 9 : Menu pour la programmation du FPGA

Nous devons sélectionner le programme Hardware Debugger pour programmer le FPGA :

Figure 10 : Menu de sélection du programme pour la programmation du FPGA

2.2 Programme à l'aide de schéma logique :

Dans cette partie, nous allons présenter les différents programmes répondant aux différents cahiers des charges. Mais pour simplifier les schémas, nous allons commencer par expliquer la notion de boîte noire.

2.2.1 Les boîtes noires :

Le logiciel nous permet de créer des boîtes noires. Une boîte noire dans un schéma est représentée comme figure 11 :

Figure 11 : Exemple d'une boîte noire

Cette boîte noire est le plus généralement utilisée pour simplifier les schémas logiques. Nous remplaçons une fonction complexe composée de beaucoup d'éléments logiques par une boîte noire qui synthétise cette fonction. Dans le cas où nous devons représenter dans le même schéma plusieurs fois cette fonction, il est astucieux d'utiliser uniquement la boîte noire.

Dans notre cas, nous avons créé des boîtes noires synthétisant les entrées et sorties du FPGA par un bus et nous avons utilisé ces boîtes noires de projet en projet, ce qui nous a permis un gain de temps et une meilleur lisibilité des schémas.

Nous allons expliciter dans les paragraphes suivants, les boîtes noires que nous avons créées.

2.2.1.1 IN_FPGA :

La boîte représentant IN_FPGA est présentée figure 12:

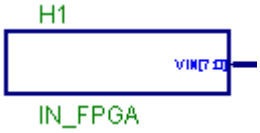

Figure 12 : Boîte noire de IN_FPGA

Cette boîte noire synthétise le schéma présenté figure 13 :

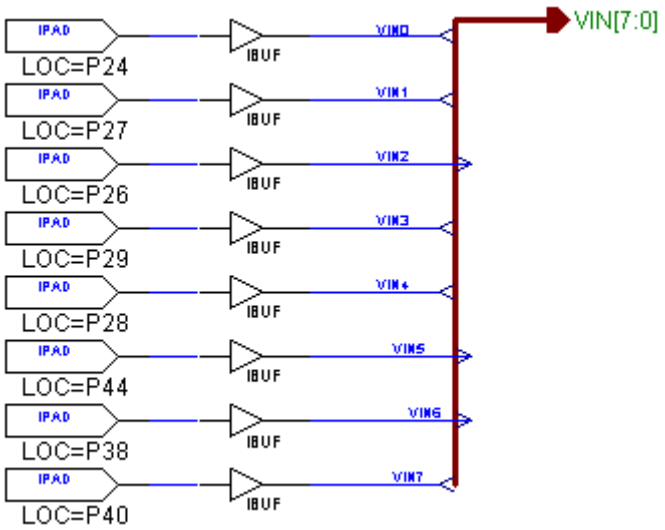

Figure 13 : Broches d'entrée du FPGA (IN_FPGA)

Afin d'alléger le schéma principal, nous avons synthétisé les huit entrées de données provenant de la carte d'acquisition, en un seul bus de 8 bits où VIN0 correspond au bit de poids faible et VIN7 à celui de poids fort.

2.2.1.2 OUT_FPGA :

La boîte représentant OUT_FPGA est présentée figure 14 :

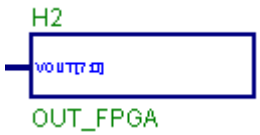

Figure 14 : Boîte noire de OUT_FPGA

Cette boîte noire synthétise le schéma présenté figure 15 :

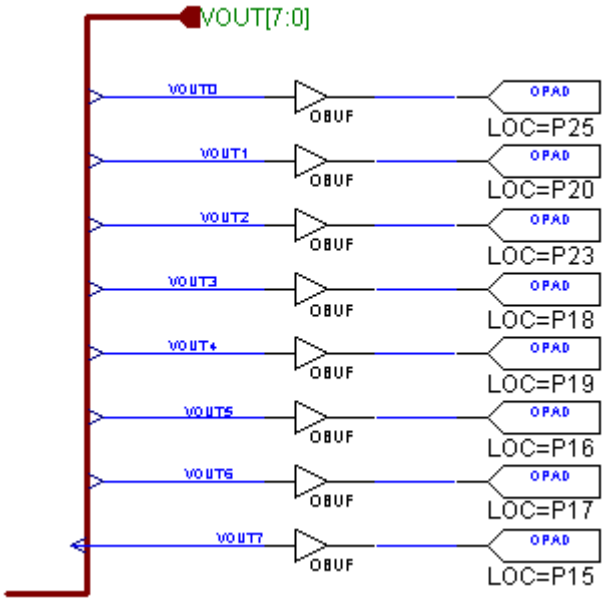

Figure 15 : Broches de sortie du FPGA (OUT_FPGA)

Afin d'alléger le schéma principal, nous avons synthétisé les huit sorties de données destinées à la carte d'acquisition, en un seul bus de 8 bits où VOUT0 correspond au bit de poids faible et VOUT7 à celui de poids fort.

2.2.1.3 ADD_RAM :

La boîte représentant ADD_RAM est présentée figure 16 :

Figure 16 : boîte noire de ADD_RAM

Cette boîte noire synthétise le schéma présenté figure 17 :

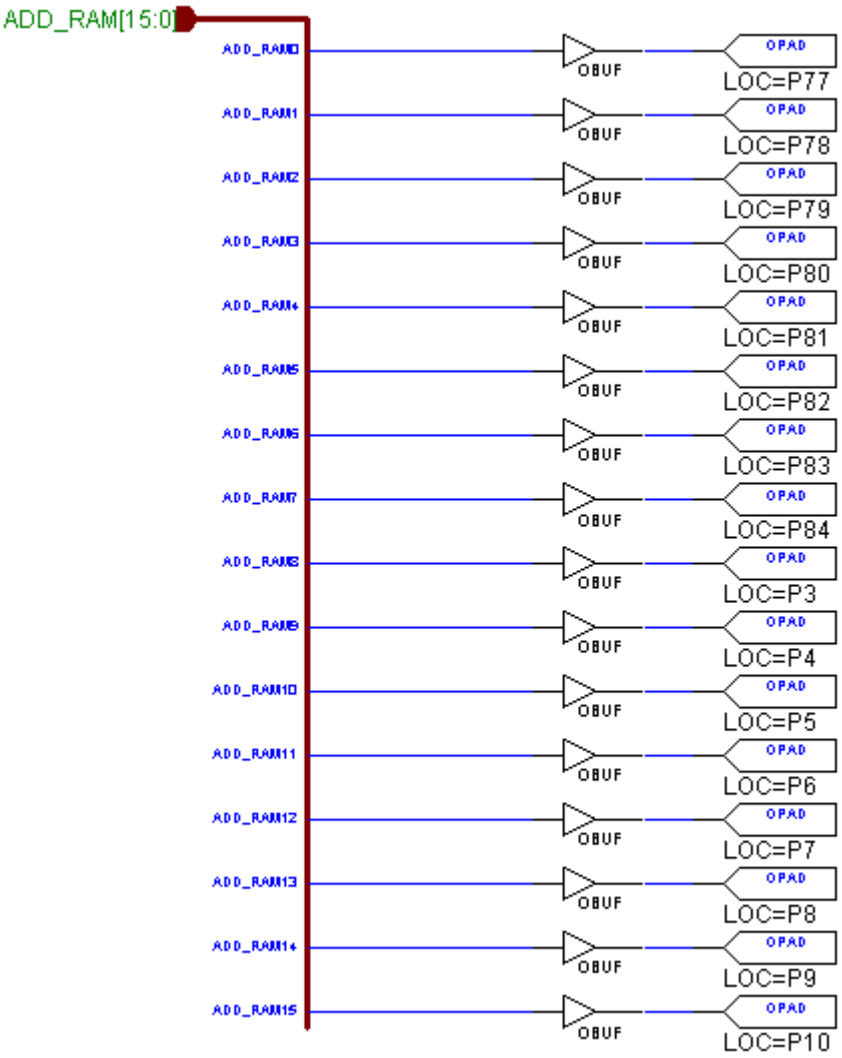

Figure 17 : Broches d'adressage de la RAM (ADD_RAM)

Afin d'alléger le schéma principal, nous avons synthétisé les seize sorties d'adressage destinées à la RAM, en un seul bus de 16 bits où ADD_RAM0 correspond au bit de poids faible et ADD_RAM15 à celui de poids fort.

2.2.1.4 DON_RAM :

La boîte représentant DON_RAM est présentée figure 18 :

Figure 18 : Boîte noire DON_RAM

Cette boîte noire synthétise le schéma présenté figure 19 :

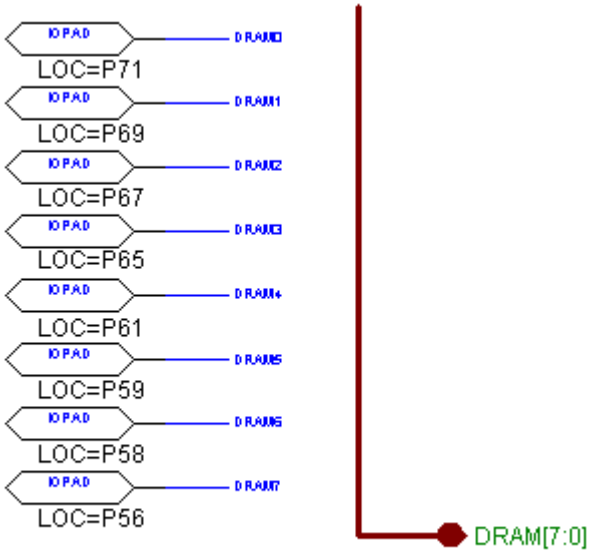

Figure 19 : Broches de données de la RAM (DON_RAM)

Afin d'alléger le schéma principal, nous avons synthétisé les huit entrées/sorties de données destinées ou provenant de la RAM, en un seul bus de 8 bits où DRAM0 correspond au bit de poids faible et DRAM7 à celui de poids fort.

2.2.1.5 SWITCH :

La boîte représentant SWITCH est présentée figure 20 :

Figure 20 : Boîte noire SWITCH

Cette boîte noire synthétise le schéma présenté figure 21 :

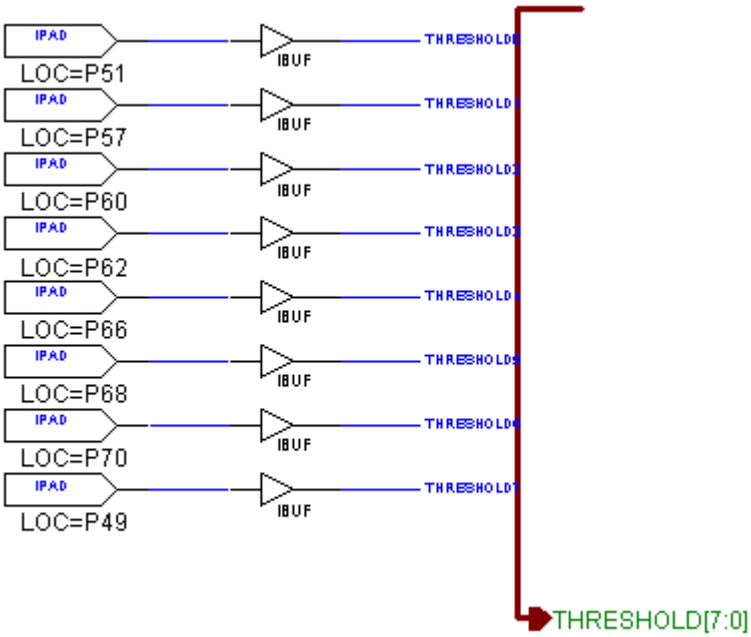

Figure 21 : boîte noire synthétisant les Switch (SWITCH)

Afin d'alléger le schéma principal, nous avons synthétisé les huit entrées Switch, en un seul bus de 8 bits où THRESHOLD0 correspond au bit de poids faible et THRESHOLD7 à celui de poids fort.

2.2.2 Envoi du signal d'entrée en sortie :

Le premier cahier des charges était le suivant : envoyer le signal d'entrée IN_FPGA en sortie OUT_FPGA. Nous allons donc observer en sortie, le signal acquis par la caméra. Ceci permet de tester le bon fonctionnement du matériel. Le schéma logique répondant au cahier des charges est représenté figure 22 :

Figure 22 : Envoi du signal d'entrée directement en sortie

Nous avons donc utilisé les boîtes noires précédemment présentées. Nous pouvons nous apercevoir du gain de clarté et de temps puisque nous n'avons pas besoin de représenter chaque entrées et chaque sorties.

Nous pouvons préciser le nombre de broches utilisées du FPGA, le nombre de bascules utilisées et le temps de propagation d'un signal de l'entrée en sortie. Ces résultats sont calculés lors de la génération et sont disponibles dans le rapport :

- Nombre de broches utilisées : 16 sur 80 soit une occupation de 26 %.
- Nombre de CLBs utilisées : aucune.
- Nombre de bascules CLBs utilisées : aucune.
- Temps de propagation : 2,299 ns.

2.2.3 Réalisation d'un négatif :

Le second cahier des charges était de réaliser un négatif de l'image d'entrée. Pour cela, il nous suffisait d'inverser le signal d'entrée grâce à une porte inverseuse huit bits. Le schéma logique correspondant est représenté figure 23 :

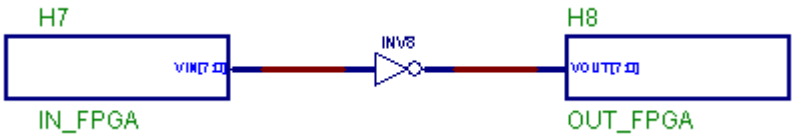

Figure 23 : Réalisation d'un négatif

Nous pouvons préciser le nombre de broches utilisées du FPGA, le nombre de bascule utilisées et le temps de propagation d'un signal de l'entrée en sortie. Ces résultats sont calculés lors de la génération et sont disponibles dans le rapport :

- Nombre de broches utilisées : 16 sur 80 soit une occupation de 26 %.
- Nombre de CLBs utilisées : aucune.
- Nombre de bascules CLBs utilisées : aucune.
- Temps de propagation : 2,299 ns.

2.2.4 Réalisation d'un lisseur :

Le troisième cahier des charges était de réaliser un lisseur qui devait lisser l'image d'entrée. Mathématiquement, la valeur d'un pixel est donnée par :

$$y(x) = \frac{f(x) + f(x-1)}{2}$$

Il faudra additionner le pixel à l'instant *t* et *t-1* et diviser le résultat par deux. Le schéma logique correspondant est représenté figure 24 :

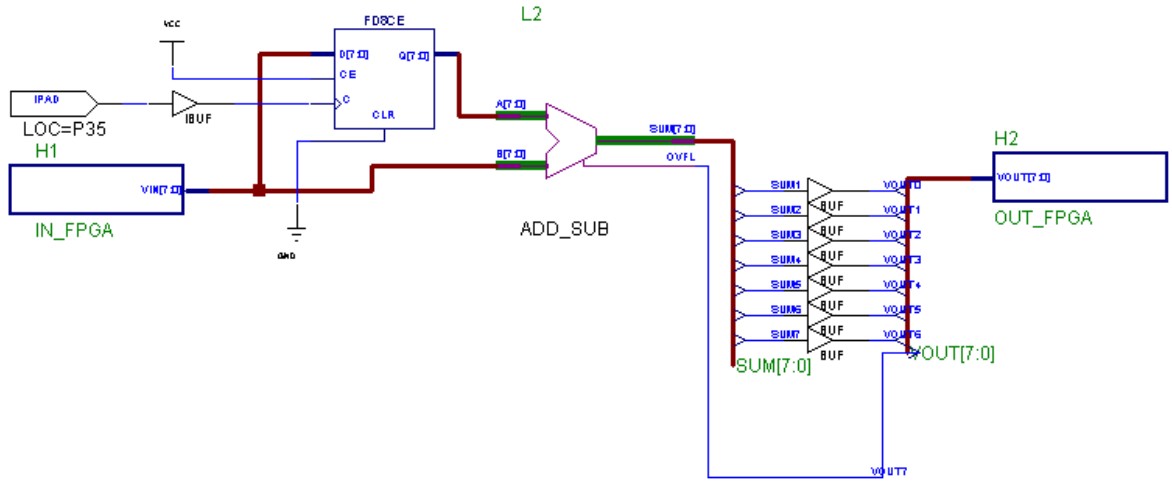

Figure 24 : Schéma représentant un lisseur

Nous allons expliquer partie par partie.

2.2.4.1 *Retardé un pixel :*

Le schéma logique permettant de retardé le signal est représenté figure 25 :

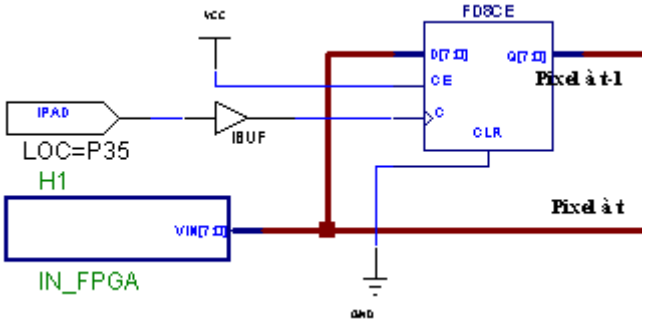

Figure 25 : Retardé un pixel

Nous pouvons donc voir que nous retardons les données d'un pixel grâce à une bascule D huit bits. Celle – ci est cadencée par l'Horloge_pixel, ceci permet d'avoir toujours le pixel à l'instant *t* et *t-1*.

Nous allons réaliser par la suite une addition.

2.2.4.2 *Addition des deux pixels :*

Le schéma logique présentant l'addition est représenté figure 26 :

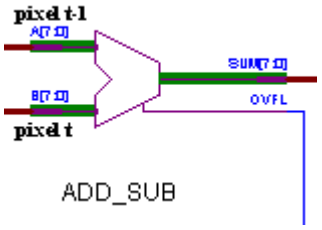

Figure 26 : Addition des deux pixels

Après avoir retardé un pixel, nous additionnons le pixel retardé et le pixel actuel.

Nous allons réaliser par la suite une division par deux.

2.2.4.3 Division de la somme par deux :

Le schéma logique présentant une division par deux est représenté figure 27 :

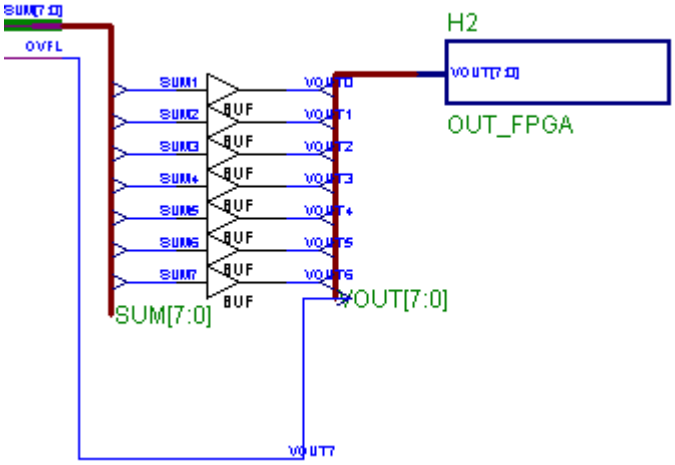

Figure 27 : Division de la somme par deux

Nous réalisons une division par deux en décalant les bits vers la gauche c'est-à-dire $SUM_n = VOUT_{n-1}$.

2.2.4.4 Rapport :

Nous pouvons préciser le nombre de broches utilisées du FPGA, le nombre de bascule utilisées et le temps de propagation d'un signal de l'entrée en sortie. Ces résultats sont calculés lors de la génération et sont disponibles dans le rapport :

- Nombre de broches utilisées : 17 sur 80 soit une occupation de 27 %.
- Nombre de CLBs utilisées : 6 sur 100 soit une occupation de 36 %.
- Nombre de bascules CLBs utilisées : 8 sur 200 soit une occupation de 4 %.
- Temps de propagation : 2,813 ns.

2.2.5 Réalisation d'un gradient :

Le quatrième cahier des charges était de réaliser le gradient de l'image d'entrée. Mathématiquement, la valeur d'un pixel est donnée par :

$$y(x) = f(x) \oplus f(x-1)$$

Il faudra faire passer le pixel à l'instant *t* et *t-1* dans une porte logique XOR. Le schéma logique correspondant est représenté figure 28 :

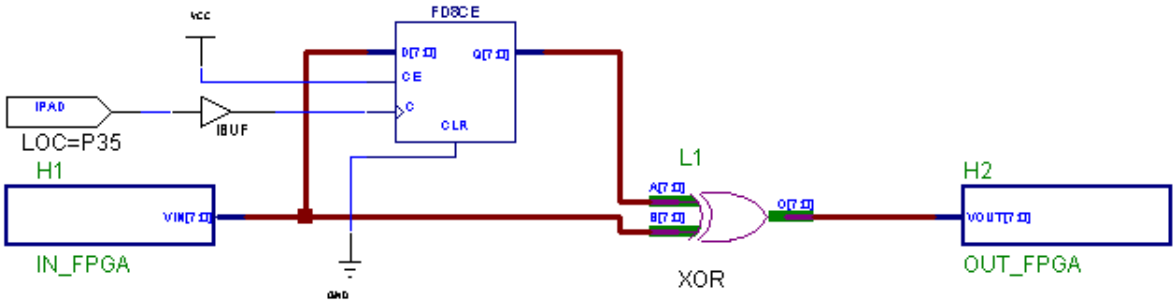

Figure 28 : Schéma logique d'un gradient

Nous allons expliquer partie par partie.

2.2.5.1 Retardé un pixel :

Le schéma logique permettant de retarder le signal est représenté figure 29 :

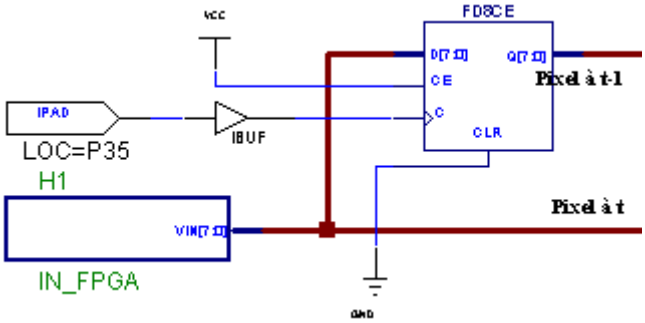

Figure 29 : Retardé un pixel

Nous pouvons donc voir que nous retardons les données d'un pixel grâce à une bascule D huit bits. Celle – ci est cadencée par l'Horloge_pixel, ceci permet d'avoir toujours le pixel à l'instant *t* et *t-1*.

Nous allons passer par une porte logique XOR pour appliquer l'opération logique demandée.

2.2.5.2 Passage par une porte XOR :

Le schéma logique permettant de réaliser l'opération demandé est représenté figure 30 :

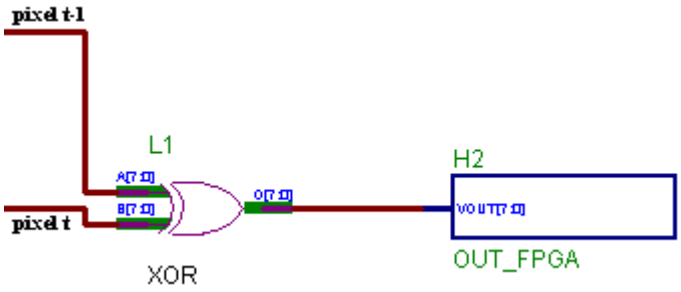

Figure 30 : Passage par une porte XOR

Après avoir retardé un pixel, nous passons par une porte logique XOR huit bits pour effectuer l'opération logique suivante : $y(x) = f(x) \oplus f(x-1)$.

2.2.5.3 Rapport :

Nous pouvons préciser le nombre de broches utilisées du FPGA, le nombre de bascule utilisées et le temps de propagation d'un signal de l'entrée en sortie. Ces résultats sont calculés lors de la génération et sont disponibles dans le rapport :

- Nombre de broches utilisées : 17 sur 80 soit une occupation de 27 %.
- Nombre de CLBs utilisées : 4 sur 100 soit une occupation de 4 %.
- Nombre de bascules CLBs utilisées : 8 sur 200 soit une occupation de 4 %.
- Temps de propagation : 2,400 ns.

2.2.6 Enregistrement en RAM :

Le cinquième cahier des charges était de réaliser un enregistrement d'une image en RAM. Le schéma logique est représenté figure 31 :

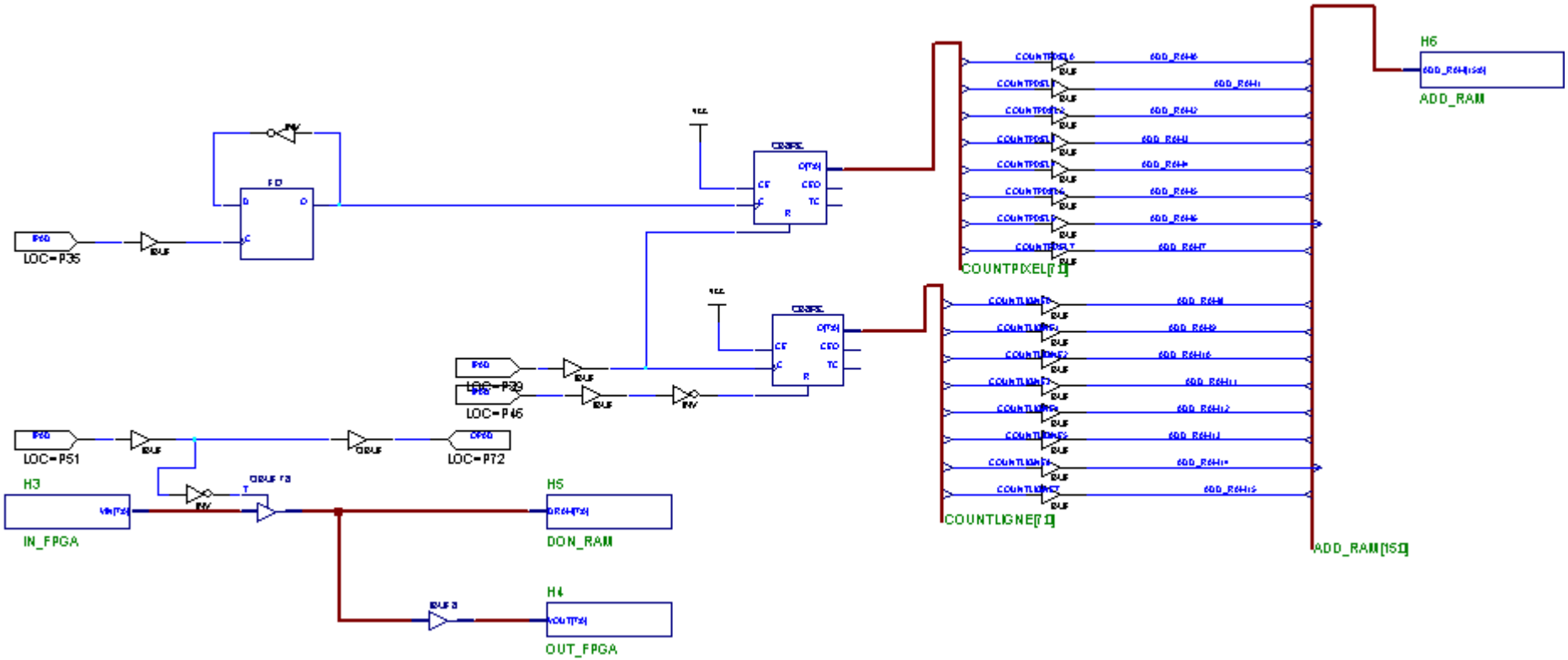

Figure 31 : Schéma logique pour permettre la mise en mémoire RAM d'une image

Nous pouvons décomposer se problème en deux parties : une partie consacrée à la gestion des données et une seconde partie consacrée à la gestion de l'adressage.

2.2.6.1 Partie gestion des données :

En ce qui concerne la gestion des données, nous devons mettre en œuvre deux cas possibles :

- Le premier étant que nous enregistrons une image dans la RAM.
- Le second étant que nous lisons une image dans la RAM.

Le schéma logique correspondant à la gestion des données est représenté figure 32:

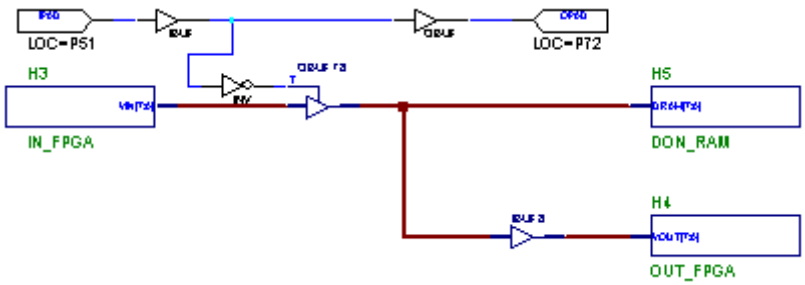

Figure 32 : Schéma logique permettant la gestion des données

Les éléments logiques permettant de commander l'enregistrement ou la lecture de la RAM sont représentés figure 33 :

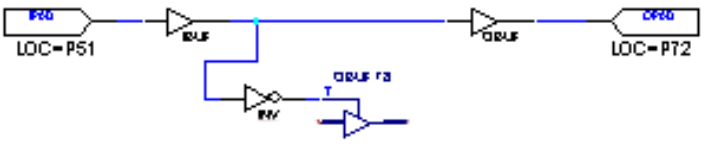

Figure 33 : Partie commande de la partie de gestion des données

En effet la broche 51 étant un Switch, il nous permet de commander WE qui est le contrôle de la lecture et de l'enregistrement dans la RAM. La valeur du Switch est également envoyée à l'OBUFT8 qui est un buffer sélectionneur 8 bits. La particularité de ce buffer est qu'il est passant lorsque la commande (ici la valeur du Switch) vaut 1 et est bloqué lorsque la commande vaut 0.

Nous avons également précisé dans 1.3.1 que lorsque WE = 1 alors la RAM est en lecture. Lorsque WE = 0, la RAM est en écriture. Ainsi nous pouvons expliquer la deuxième partie du schéma présenté figure 34 :

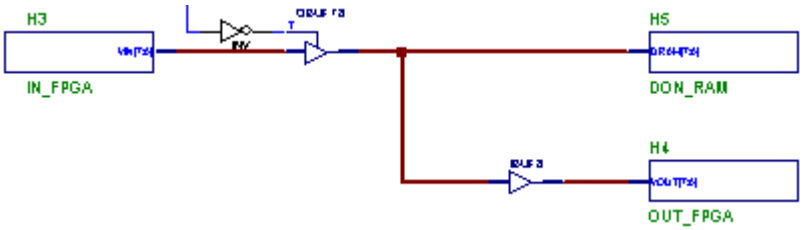

Figure 34 : Partie gérant la directionnement des données

Lorsque la valeur du Switch vaut 1, la RAM sera en lecture. Ainsi nous devons avoir un bouclage des données de la RAM DON_RAM vers la sortie du FPGA OUT_FPGA, c'est-à-dire que le buffer sélectionneur devra être bloqué donc commander par une valeur logique 0. Ceci explique l'intérêt de l'inverseuse devant la commande.

De la même manière, lorsque la valeur du Switch vaut 0, la RAM sera en écriture. Ainsi, nous devons diriger les données d'entrée du FPGA IN_FPGA vers dans la RAM par DON_RAM. Ainsi, le buffer sélectionneur doit être passant et doit être commandé par une valeur logique 1. Ceci explique l'intérêt de l'inverseuse devant la commande.

2.2.6.2 Partie gestion de l'adressage :

Lors de l'enregistrement dans la RAM, le tout n'est pas que d'envoyer mais il faut les enregistrer au bon endroit c'est-à-dire gérer l'adressage. Pour des raisons de capacité de mémoire expliquées au paragraphe 1.3.2, nous avons décidé d'enregistrer un pixel sur deux et une ligne sur deux. Ceci revient à enregistrer un pixel sur deux pendant une trame. Le schéma logique représentant la gestion de l'adressage est présenté figure 35:

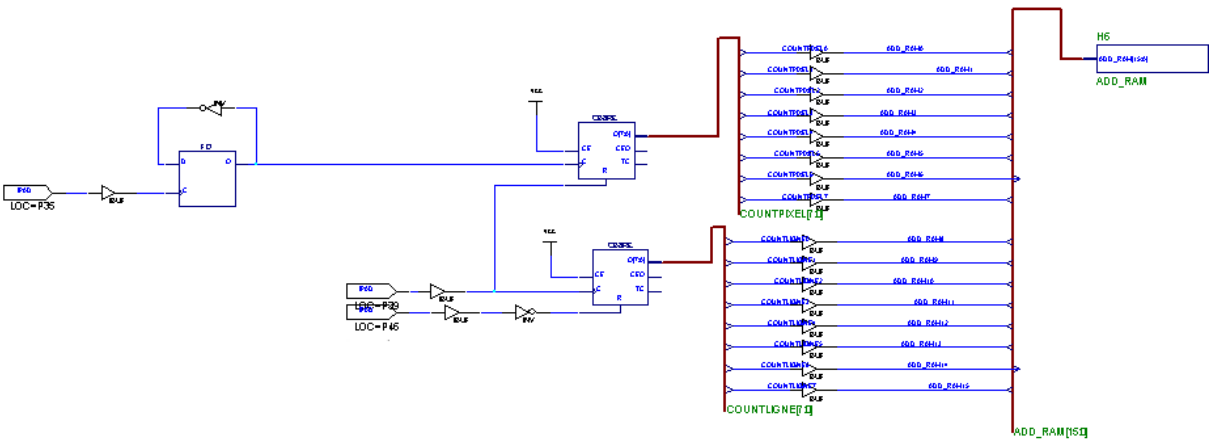

Figure 35 : Schéma logique permettant la gestion de l'adressage

Pour réaliser une lecture correcte, nous avons décidé de gérer un compteur pour la position du pixel sur la ligne et un autre qui donnera la position de la ligne sur la trame. Nous avons donc les huit premiers bits de l'adressage (A_0 à A_7) dédiés à la position du pixel sur la ligne et les 8 derniers bits de

l'adressage (A_8 à A_{15}) dédiés à la position de la ligne sur la trame. Nous allons diviser cette partie en expliquant le schéma éléments par éléments.

2.2.6.2.1 Division de l'Horloge_pixel par deux :

Pour enregistrer seulement un pixel sur deux, nous devons diviser l'Horloge_pixel par deux. Le schéma représentant la division de l'horloge par deux est présenté figure 36:

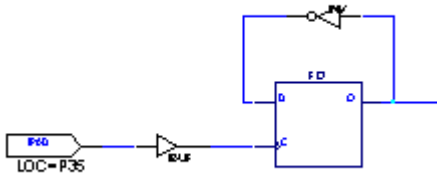

Figure 36 : Schéma logique permettant une division de fréquence par deux

Nous effectuons la division par deux à l'aide d'une bascule D. Le chronogramme relevé est présenté figure 37 :

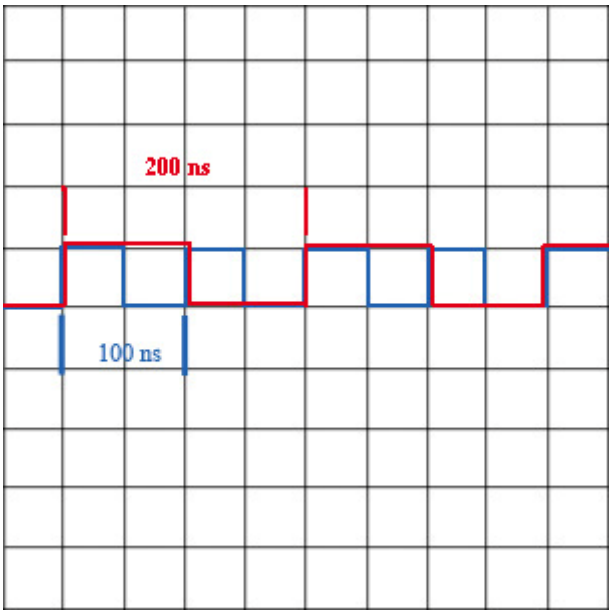

Figure 37 : Bleu : Horloge pixel - Rouge : Horloge pixel divisé par deux

2.2.6.2.2 Détermination de la position du pixel enregistré :

Le schéma logique permettant la détermination du pixel sur la ligne est présenté figure 38:

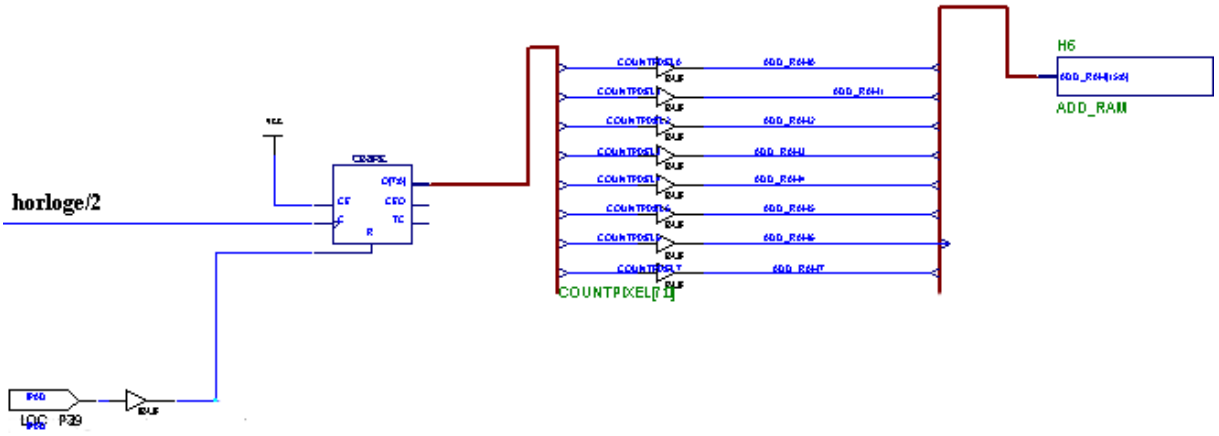

Figure 38 : Shéma logique permettant la détermination du pixel sur la ligne

Le compteur utilisé est un compteur huit bits muni d'un Reset. L'horloge du compteur sera donc l'Horloge_pixel divisé par deux. Ce compteur doit être remis à zéro au début de chaque nouvelle ligne. Ceci est réalisé par la broche Synchro_ligne. Le signal de Synchro_ligne est présenté figure 39 :

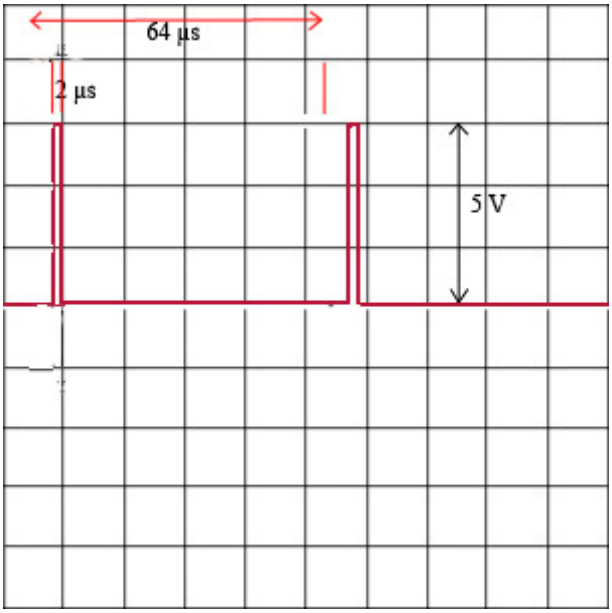

Figure 39 : Signal de synchronisation ligne

Le bus de sortie sera donc renvoyé sur le bus d'entrée d'adressage de la RAM d'A_0 à A_7.

2.2.6.2.3 Détermination du numéro de la ligne :

Le schéma logique pour déterminer le numéro de la ligne entrain d'être enregistré est présenté figure 40 :

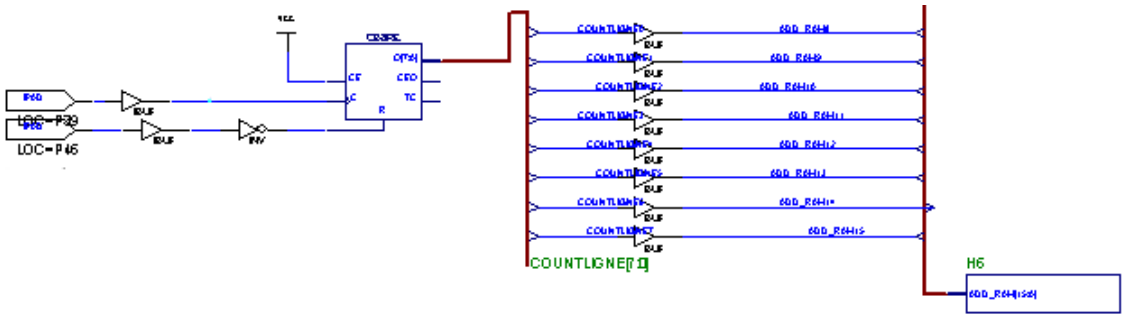

Figure 40 : Schéma logique permettant de déterminé le numéro de la ligne enregistrée

Le compteur utilisé est un compteur huit bits muni d'un Reset. L'horloge du compteur sera donc la Synchro_ligne. Ce compteur doit être remis à zéro au début de chaque nouvelle trame. Le signal de Synchro_trame est présenté figure 41:

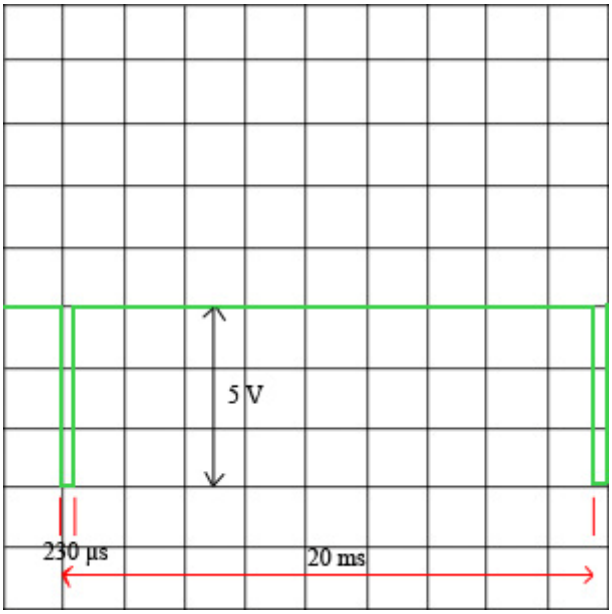

Figure 41 : Signal de synchronisation trame

Nous devons inverser se signal à l'aide d'une inverseuse car la remise à zéro ne s'effectue que sur front montant et nous voulons effectuer la remise à zéro au début de la synchronisation c'est-à-dire sur un front descendant du signal précédent.

Le bus de sortie sera donc renvoyé sur le bus d'entrée d'adressage de la RAM d'A_8 à A_{15}.

2.2.6.3 Rapport :

Nous pouvons préciser le nombre de broches utilisées du FPGA, le nombre de bascule utilisées et le temps de propagation d'un signal de l'entrée en sortie. Ces résultats sont calculés lors de la génération et sont disponibles dans le rapport :

- Nombre de broches utilisées : 45 sur 80 soit une occupation de 73 %.
- Nombre de CLBs utilisées : 11 sur 100 soit une occupation de 11 %.
- Nombre de bascules CLBs utilisées : 17 sur 200 soit une occupation de 4 %.
- Temps de propagation : 6,041 ns.

2.2.7 Calcul du centre de gravité d'une image :

Le cinquième cahier des charges était de calculer le centre de gravité de l'image. Nous devons tout d'abord effectuer une binarisation à l'aide d'une valeur seuil qui sera déterminée avec les Switch. Nous aurons alors une image binaire. Il nous suffira d'accumuler le nombre de pixels à 1 de la partie droite de l'image et d'accumuler le nombre de pixels à 1 de la partie gauche et enfin de comparer les deux valeurs des deux accumulateurs. Le schéma logique représentant le centre de gravité de l'image est présenté figure 42 :

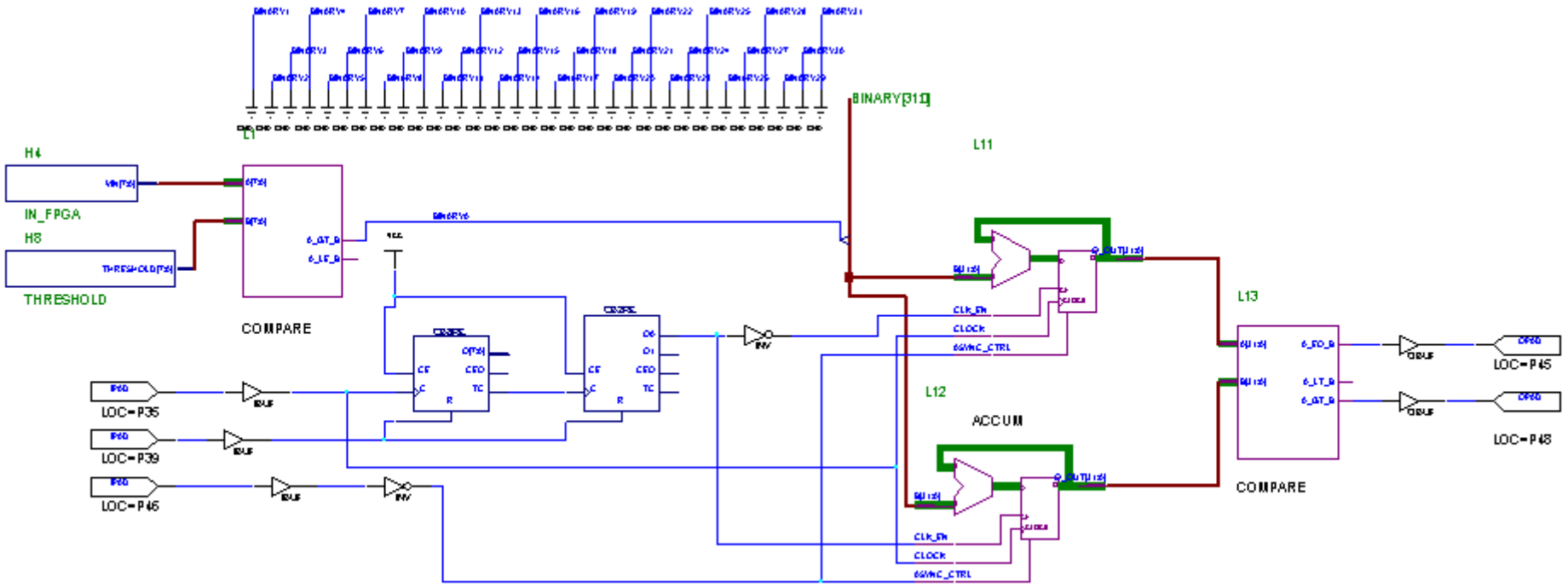

Figure 42 : Schéma logique permettant de calculer le centre de gravité de l'image

Nous allons expliquer étape par étape les opérations réalisées.

2.2.7.1 Binarisation :

Chaque pixel à une valeur comprise entre 0 et 255 codé sur huit bits. La première étape est de binariser l'image, c'est-à-dire de passer chaque pixel à 0 ou à 1, codé sur un seul bit. Pour un affichage éventuel, la valeur 1 correspond à 255 et la valeur 0 vaut 0. Le schéma logique correspondant à cette manipulation est présenté figure 43:

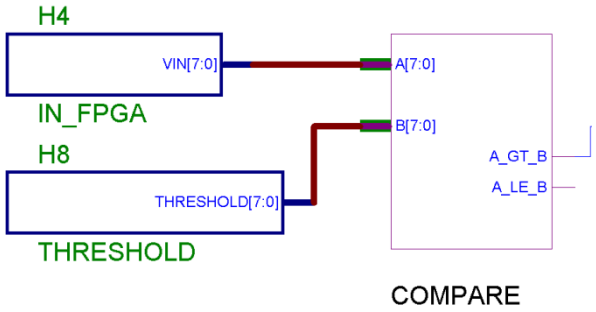

Figure 43 : Schéma logique permettant une binarisation

Nous utilisons un comparateur huit bits qui nous permet de comparer la valeur d'IN_FPGA par rapport à une valeur fixée par l'utilisateur via les Switch. Le résultat escompté est le suivant :

- Si IN_FPGA < SWITCH, alors la valeur du pixel sera de 0.
- Si IN_FPGA > SWITCH, alors la valeur du pixel sera de 1.

2.2.7.2 Détermination de la position sur l'image :

Afin d'accumuler les pixels à 1 de la partie droite de l'image et ceux de la partie gauche, nous devons tout d'abord déterminer de quel côté de l'image nous nous trouvons. Le schéma logique permettant de connaître de quel côté de l'image nous nous trouvons est présenté figure 44 :

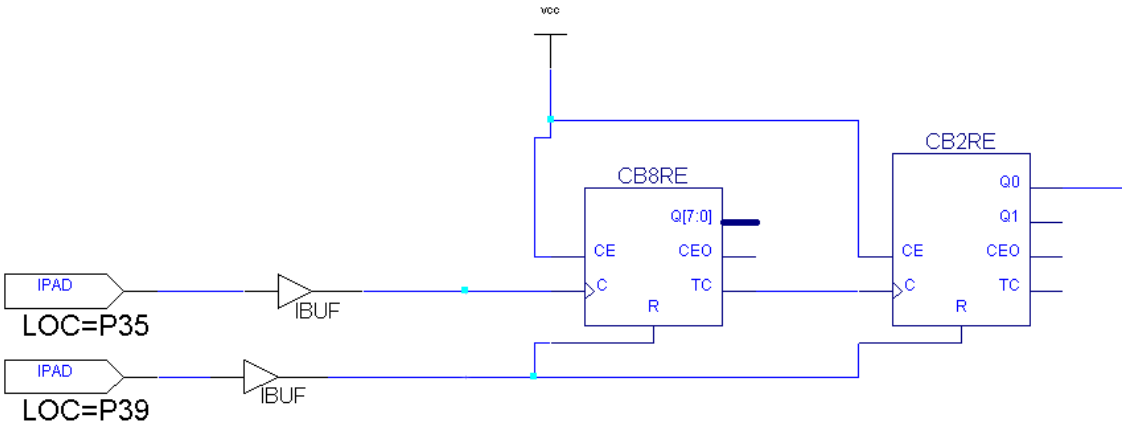

Figure 44 : Schéma logique permettant de déterminer de quel côté de l'image nous nous trouvons

Nous savons que la largeur de l'image est de 512 pixels. Nous devons synthétiser un signal pour lequel la valeur 0 indiquera que nous sommes sur la partie gauche de l'image et la valeur 1 indiquera que nous sommes sur la partie droite de l'image. Pour cela nous utilisons un compteur huit bits avec une commande de reset. La valeur maximale qui peut être prix par le compteur est 255. La sortie TC (Total Count) passera à 1 lorsque nous dépasserons 255. Cependant, la valeur de TC retombera à 0 le coup d'Horloge_pixel suivant. Nous devons donc conserver cette valeur pendant les 255 coups d'horloge suivant. Pour cela nous utilisons un compteur deux bits avec un reset. Nous devons remettre à zéro tous les compteurs lorsque nous recommençons une nouvelle ligne. Pour cela nous utiliserons le signal de Synchro_ligne pour effectuer la remise à zéro. Le signal synthétiser est présenté figure 45 :

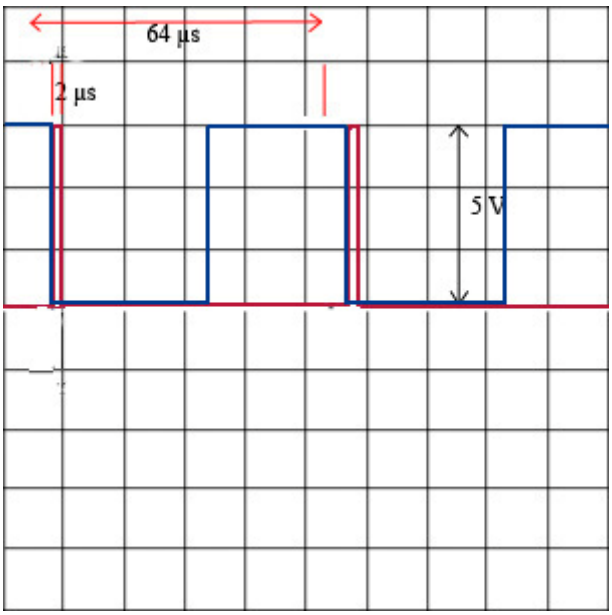

Figure 45 : Rouge : Signal de synchronisation de ligne - Bleu : Signal synthétisé permettant de connaître de quel côté de l'image nous nous trouvons

2.2.7.3 Accumulation des pixels :

Pour chaque partie de l'image, nous devons accumuler les valeurs à 1. Les accumulateurs sont des accumulateurs à trente et un bits, c'est pour cela que nous complétons le bit de la binarisation avec des zéros pour tous les autres bits du bus. Le schéma logique permettant l'accumulation est présenté figure 46 :

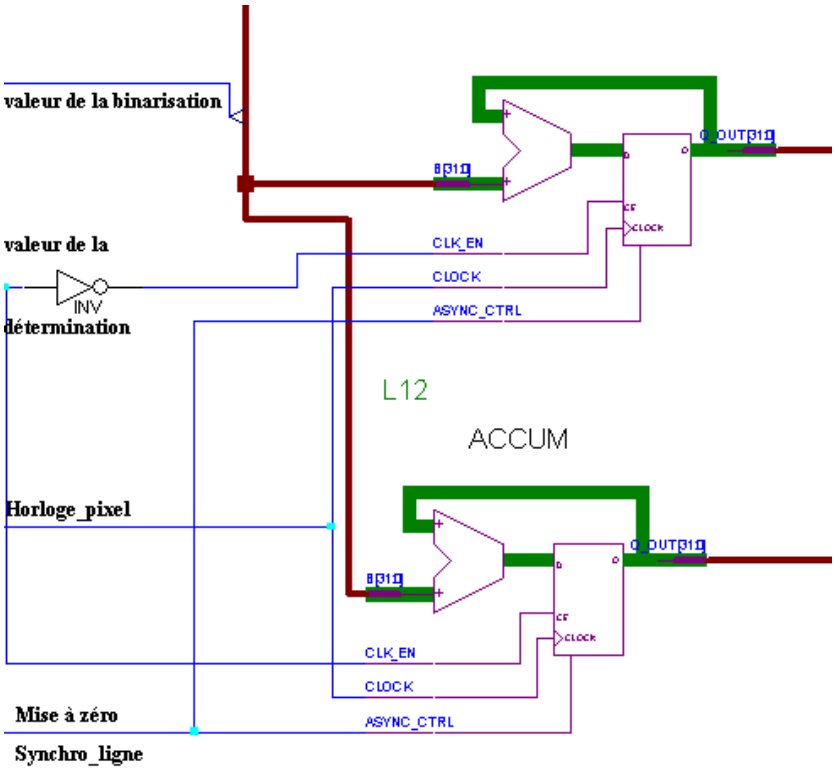

Figure 46 : Schéma logique permettant l'accumulation

Le signal synthétiser dans la partie précédente sélectionne l'accumulateur à sélectionner. Si le signal de la détermination vaut 0, alors l'accumulateur le plus en haut sera activé alors que celui du dessous sera désactivé. Le contraire se produit lorsque la valeur de la détermination vaut 1. Lorsqu'un accumulateur est activé, à chaque coup d'horloge, nous additionnons la valeur de la binarisation avec la valeur précédente de l'accumulateur. A chaque fin de trame, nous devons remettre à zéro les accumulateurs. Nous utilisons pour cela la Synchro_trame inversée pour déclencher la mise à zéro sur un front montant correspondant au début d'une nouvelle trame car le signal de Synchro_trame est présenté figure 47 :

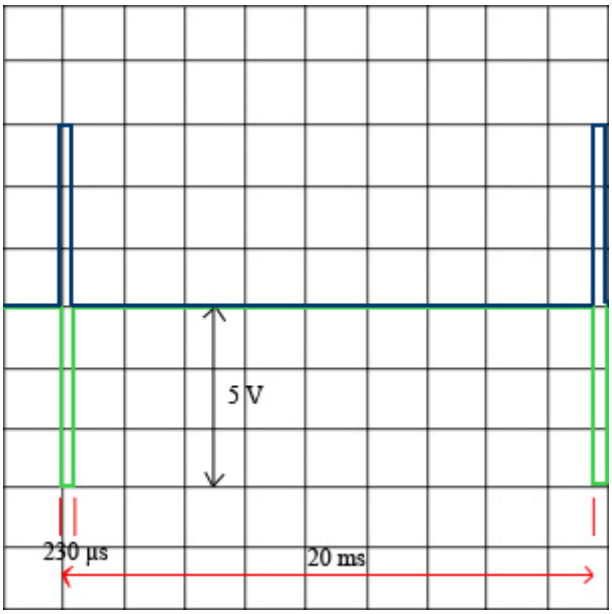

Figure 47 : Vert : Signal de synchronisation de trame - Bleu : Signal de synchronisation de trame inversé

2.2.7.4 *Comparaison des valeurs accumulées :*

Après avoir accumulé toutes les valeurs, nous n'avons plus qu'à comparer la valeur de chaque accumulateur. Le schéma logique correspondant est présenté figure 48:

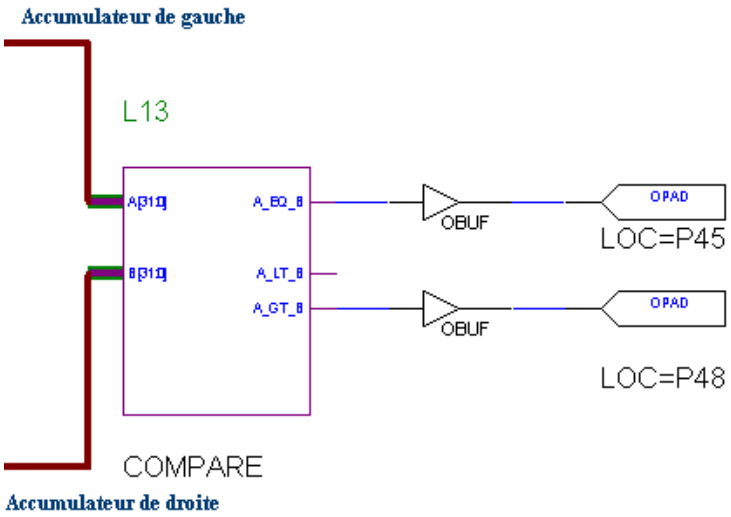

Figure 48 : Schéma logique permettant de déterminer le côté où il y a le plus de valeur logique 1

Si la valeur de l'accumulateur de gauche est supérieure à la valeur de l'accumulateur de droite, alors nous obtenons sur la broche libre 48, la valeur logique 1. Dans le cas contraire, nous obtenons une valeur logique 0.

Lorsque la valeur logique est 0, ceci indique que l'objet se trouve plus sur le côté gauche que sur le côté droit. Dans le cas contraire, l'objet se trouve plus sur le côté droite que sur le côté gauche.

Nous avons également gérer le cas où l'objet se trouve au milieu de l'image c'est-à-dire que les valeurs des accumulateurs de droite et de gauche sont égales. Ce cas est renvoyé sur la broche libre 45.

2.2.7.5 Rapport :

Nous pouvons préciser le nombre de broches utilisées du FPGA, le nombre de bascule utilisées et le temps de propagation d'un signal de l'entrée en sortie. Ces résultats sont calculés lors de la génération et sont disponibles dans le rapport :

- Nombre de broches utilisées : 21 sur 80 soit une occupation de 34 %.
- Nombre de CLBs utilisées : 85 sur 100 soit une occupation de 85 %.
- Nombre de bascules CLBs utilisées : 73 sur 200 soit une occupation de 36 %.
- Temps de propagation : 7,345 ns.

3 Conclusion :

Il a été présenté dans cette étude la carte FPGA et plus précisément la partie programmation du FPGA et cela dédié au développement d'une carte de traitement d'image à base de FPGA.

Nous avons donc rappelé dans un premier temps le fonctionnement et les données électroniques de la carte FPGA.

Dans un second temps, nous avons présenté la partie programmation permettant le traitement d'image. Nous avons présenté tout d'abord le fonctionnement du logiciel Xilinx Project Manager ainsi que l'utilité des boîtes noires. Dans un second temps, nous avons détaillé la programmation de différents traitements réalisés tel que le négatif d'une image, le lissage d'une image, le gradient d'une image. Enfin nous avons fini par aborder, l'enregistrement d'image en RAM ainsi que le calcul du centre de gravité d'une image.