

# Rapport Projet VGA

## Création d'un filtre FIR sur le flux vidéo

Samory DIABY  
Cédrine SOCQUET

# Table des matières

<b>1 Objectifs</b>	<b>2</b>
<b>2 Plan de validation</b>	<b>3</b>
2.1 Analyses . . . . .	3
2.2 Tests . . . . .	3
2.3 Démonstrations . . . . .	3
<b>3 Pilote VGA</b>	<b>4</b>
3.1 Principe de l'affichage VGA . . . . .	4
3.2 Réalisation du pilote VGA . . . . .	7
<b>4 Générateur d'images</b>	<b>8</b>
4.0.1 Principe . . . . .	8
4.0.2 Réalisation du générateur d'images . . . . .	8
<b>5 Filtre gaussien</b>	<b>10</b>
5.1 Principe du filtre gaussien . . . . .	10
5.1.1 Réalisation du filtre gaussien . . . . .	10
5.1.2 Modifications liées au décalage du filtre . . . . .	12
<b>6 Analyse</b>	<b>14</b>
<b>7 Test</b>	<b>17</b>
<b>8 Démonstration</b>	<b>20</b>

# 1 Objectifs

- Afficher une image dotée d'un filtre de flou gaussien et l'afficher sur un écran VGA, via la réalisation d'une IP de traitement d'images.
- Utiliser une mire quadrillée en noir et blanc sous forme de damier, pour afficher le traitement d'image par le filtre.

L'utilité de cette mire est qu'elle permet de mettre en évidence la bonne réalisation du filtre gaussien : en effet, le contraste étant maximal, il est facile de voir les nuances et le pattern permet de mettre facilement en évidence un potentiel décalage au niveau des lignes ou des colonnes.

## 2 Plan de validation

### 2.1 Analyses

- Vérification de la bonne réception des pixels par le contrôleur VGA via la réalisation d'un générateur d'image (pattern generator).
- Vérification de l'obtention des signaux H sync et V sync et de leur timing, via la mise en place du VGA driver.
- Validation de la conformité à la norme VGA via simulation sous Vivado simulator.
- Vérification de l'implémentation du filtre gaussien (entre le pattern generator et le VGA driver), en observant le passage au gris (en passant par la valeur 127) du damier noir et blanc du pattern generator.

### 2.2 Tests

- Mise en place des sondes ILA pour vérifier que les données en sortie de H sync et V sync sont correctement animées.
- Vérification des signaux H sync et V sync du VGA à l'aide d'un oscilloscope, par l'observation de rampes de données.

### 2.3 Démonstrations

- Affichage de bandes tricolores R, V, B sur un écran VGA.
- Affichage d'un quadrillage noir et blanc de type damier sur un écran VGA.
- Affichage du damier bicolore avec le traitement d'image par le filtre de flou gaussien sur un écran VGA.

### 3 Pilote VGA

Afin d'afficher des images sur un port VGA, nous avons utilisé un module *PMOD VGA* qui nous permet d'ajouter une sortie VGA sur notre carte CoraZ7. C'est ce module qui nous sert pour faire l'interface entre la carte et l'écran VGA. Nous nous sommes également renseignés sur la norme VGA afin d'envoyer et de réceptionner correctement les différents signaux nécessaires au fonctionnement du VGA et donc de l'écran d'affichage.

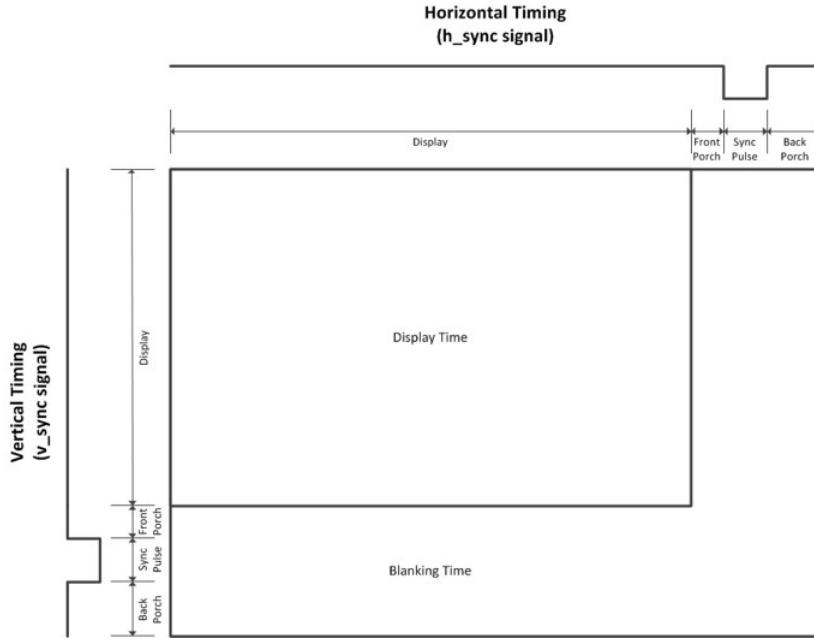
#### 3.1 Principe de l'affichage VGA

Pour afficher nos images à l'aide de la norme VGA, nous devons nous servir de 5 signaux analogiques qui sont :

- **H SYNC** et **V SYNC** : qui sont des signaux de synchronisations permettant d'indiquer à l'écran où nous en sommes dans l'affichage d'une image.
- **RED**, **BLUE** et **GREEN** : qui sont les signaux correspondant à la valeur que prendra chaque sous-pixel de l'écran pour former notre image.

Pour respecter la norme VGA et ainsi afficher correctement nos images à l'écran, nous devons respecter les timings imposés pour *H SYNC* et *V SYNC*. Si ces timings ne sont pas respectés, on obtient un écran noir indiquant que nos signaux ne sont pas reconnus par l'écran.

La figure 1 représente l'organisation de l'affichage pour une image sous la norme VGA. On constate que notre image visible est représentée par le rectangle *display time* et servira à afficher tous les pixels de nos images. Le reste du schéma représente une partie "invisible" des images qui sert de temporisation et qui est obligatoire pour le respect de la norme. On remarque une impulsion des signaux *H SYNC* et *V SYNC* lors de la période *sync pulse*. Nous devrons donc faire en sorte d'obtenir cette impulsion pour nos signaux afin que l'écran détecte que nous utilisons du VGA. Les périodes *front porch* et *back porch* repéresntent des temps morts qui sont utilisés pour la remise en place des systèmes d'affichage utilisés pour d'anciens écrans (notamment les écrans cathodiques).



**FIGURE 1 – Norme VGA**

La figure 2 nous indique plus précisément les temps à respecter pour correspondre à la norme et donc au schéma figure 1. On remarque que pour des images de dimension 640x480 (pixels en largeur x pixels en hauteur) et une fréquence de rafraîchissement de 60 Hz on doit obtenir une impulsion du signal *HSYNC* au bout d'environ **25,06 µs** pendant environ **3,81 µs** et pour *VSYNC* une impulsion au bout de **15,57 ms** pendant **0,06 ms**.

#### General timing

Screen refresh rate	60 Hz
Vertical refresh	31.46875 kHz
Pixel freq.	25.175 MHz

#### Horizontal timing (line)

Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [µs]
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

#### Vertical timing (frame)

Polarity of vertical sync pulse is negative.

Frame part	Lines	Time [ms]
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

**FIGURE 2 – Timings VGA**

Pour ce projet nous allons utiliser la configuration ci-dessus pour nos images, c'est-à-dire des images de dimension 640x480 pixels à une fréquence de rafraîchissement de 60Hz. On doit alors calculer la fréquence d'horloge nécessaire au bon fonctionnement de notre design de la manière suivante :

Nos images auront une dimension de 640 x 480 pixels, or la norme nous oblige à apporter de la latence avec la partie "invisible" de l'image, ce qui augmente la dimension totale des images à 800 x 525 pixels (Cf. figure 1 et figure 2). Nous devons "afficher" 420 000 pixels à l'écran (dont les pixels invisibles compris dans la dimension 800 x 525) 60 fois par seconde (fréquence de 60 Hz), nous avons donc une période d'horloge de :

$$\frac{1}{420000 \times 60} \approx 39,7ns$$

Ce qui correspond à une fréquence d'horloge de :

$$\frac{1}{39,7ns} \approx 25,2MHz$$

### 3.2 Réalisation du pilote VGA

Notre implémentation de la norme VGA est représentée sur le schéma ci-dessous :

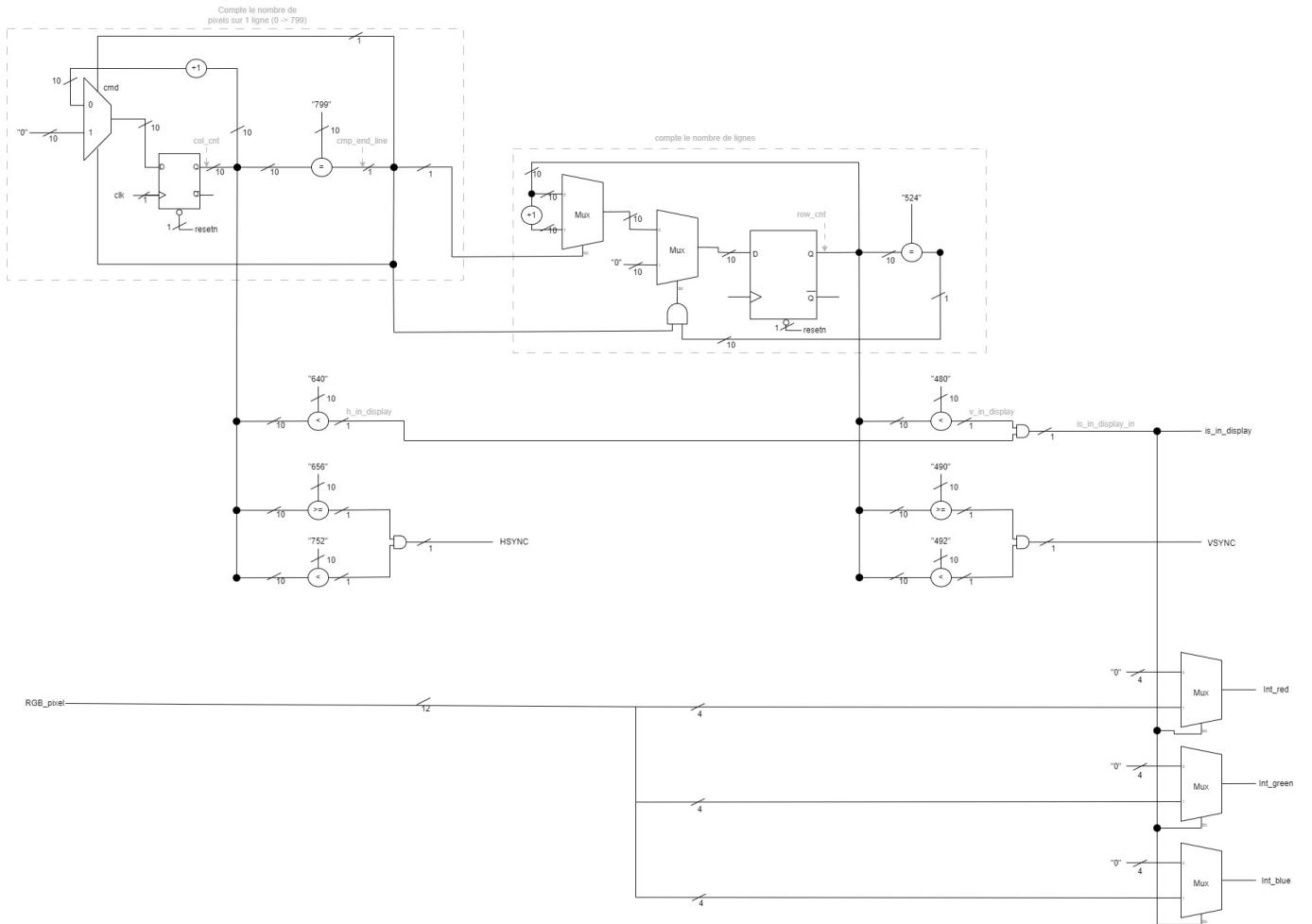


FIGURE 3 – Schéma RTL du pilote vga

On retrouve les 5 signaux nécessaires au respect de la norme (HSYNC, VSYNC, `int_red`, `int_green`, `int_blue`). Les timings sont respectés via 2 compteurs, 1 pour les lignes et 1 pour les colonnes. Chaque incrément du compteur de colonnes nous indique que **39,7ns** sont passées car ce compteur est incrémenté de '1' à chaque front montant d'horloge. On remarque aussi la gestion de l'affichage des pixels qui est effectué uniquement lorsque l'on se trouve dans la zone d'affichage (le signal `is_in_display` vaut '1').

Le signal `is_in_display` est également une sortie du module car il servira d'indicateur au module de génération d'images pour générer correctement ces dernières.

## 4 Générateur d'images

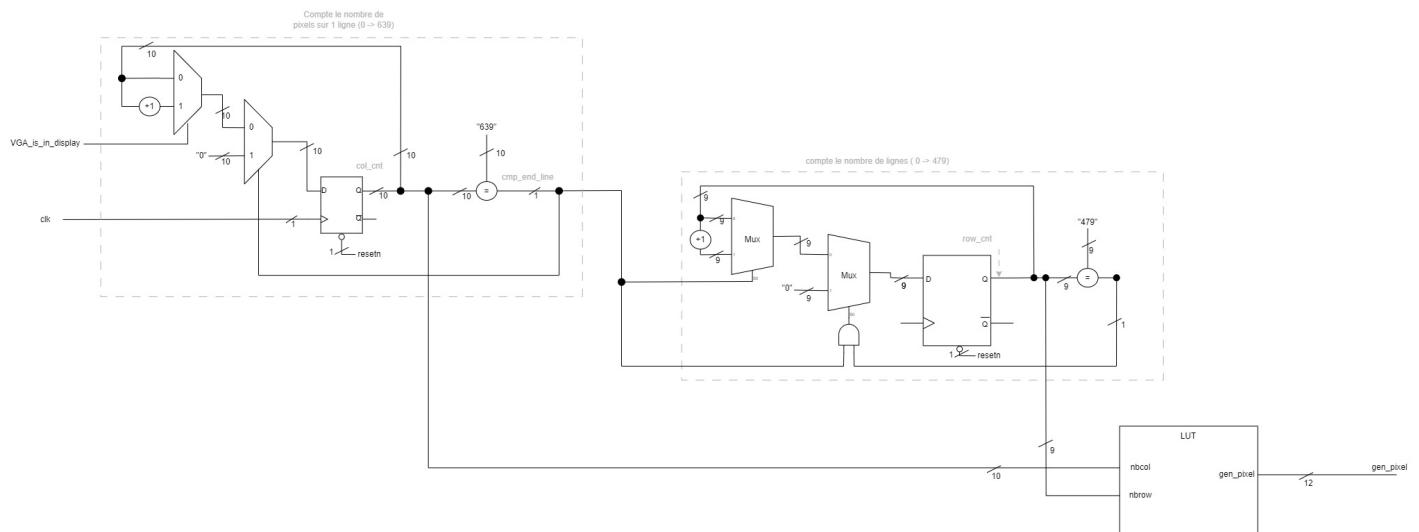
Le générateur d'images est un module qui nous permet de générer des motifs à afficher à l'écran afin de nous assurer d'une part que le pilote VGA fonctionne correctement et d'autre part d'avoir des motifs à passer au futur filtre gaussien.

### 4.0.1 Principe

Ce générateur d'images utilise un compteur interne qui est calé sur le compteur du pilote VGA, et a pour but de fournir des pixels à ce dernier en fonction d'un "motif" défini au préalable. Le motif choisi pour ce projet est un quadrillage en noir et blanc composés de carrés de dimensions 32 x 32 pixels.

### 4.0.2 Réalisation du générateur d'images

Notre implémentation du générateur d'images est représentée sur le schéma ci-dessous :



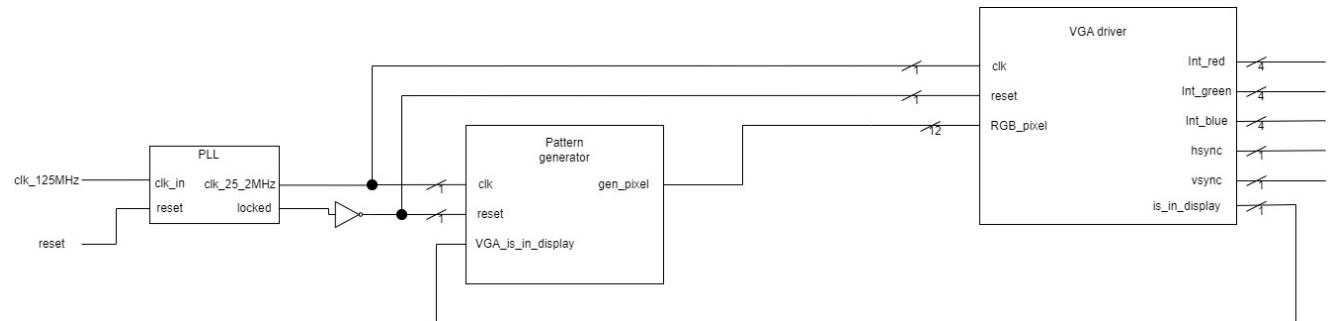
**FIGURE 4** – Schéma RTL du générateur d'images

On retrouve bien nos 2 compteurs (1 pour les colonnes et 1 pour les lignes) qui serviront à générer le pixel en sortie en fonction de la *LUT* qui sert à créer le motif. Dans le cas du quadrillage la *LUT* ressemble à ceci :

```
1 gen_pixel <= (others => '1') when (col_cnt(5) = '0' xor row_cnt(5) = '0')
  else (others => '0');
```

On regarde le 6<sup>eme</sup> bit du compteur de colonnes pour générer 32 pixels noirs quand ce bit vaut '0', et 32 pixels blancs quand il vaut '1'. On continue de même sur chaque ligne jusqu'à un changement sur le 6<sup>eme</sup> bit du compteur de lignes qui nous indique quand alterner de couleur sur les colonnes pour créer une alternance de carrés blancs et noirs, ce qui nous donne donc un quadrillage semblable à un damier, que l'on utilise comme mire.

Avec ces deux éléments (**pilote VGA** et **générateur d'images**) on obtient le synoptique suivant qui représente l'état du projet sans la partie de traitement d'image :



**FIGURE 5 –** Synoptique du projet sans filtre gaussien

## 5 Filtre gaussien

Cette section traite de la partie de traitement d'image, ici le filtre gaussien, que nous avons implémenté dans notre design.

### 5.1 Principe du filtre gaussien

Le filtre gaussien consiste à un calcul de convolution appliquée à chacune des images que nous avons à afficher. Le calcul est le suivant :

$$y[m, n] = x[m, n] * h[m, n] = \sum_{j=-a}^a \sum_{k=-b}^b x[j, k]h[m - j, n - k]$$

**FIGURE 6** – Formule de la convolution

Avec :

- $h[m, n]$  : le noyau de convolution appliqué à l'image d'entrée
- $x[m, n]$  : l'image d'entrée à filtrer
- $y[m, n]$  : l'image résultante
- $m, n$  : la position des pixels

Le noyau de convolution (matrice  $h$  dans la formule ci-dessus) que nous allons utiliser pour réaliser le filtre gaussien est le suivant :

$$\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

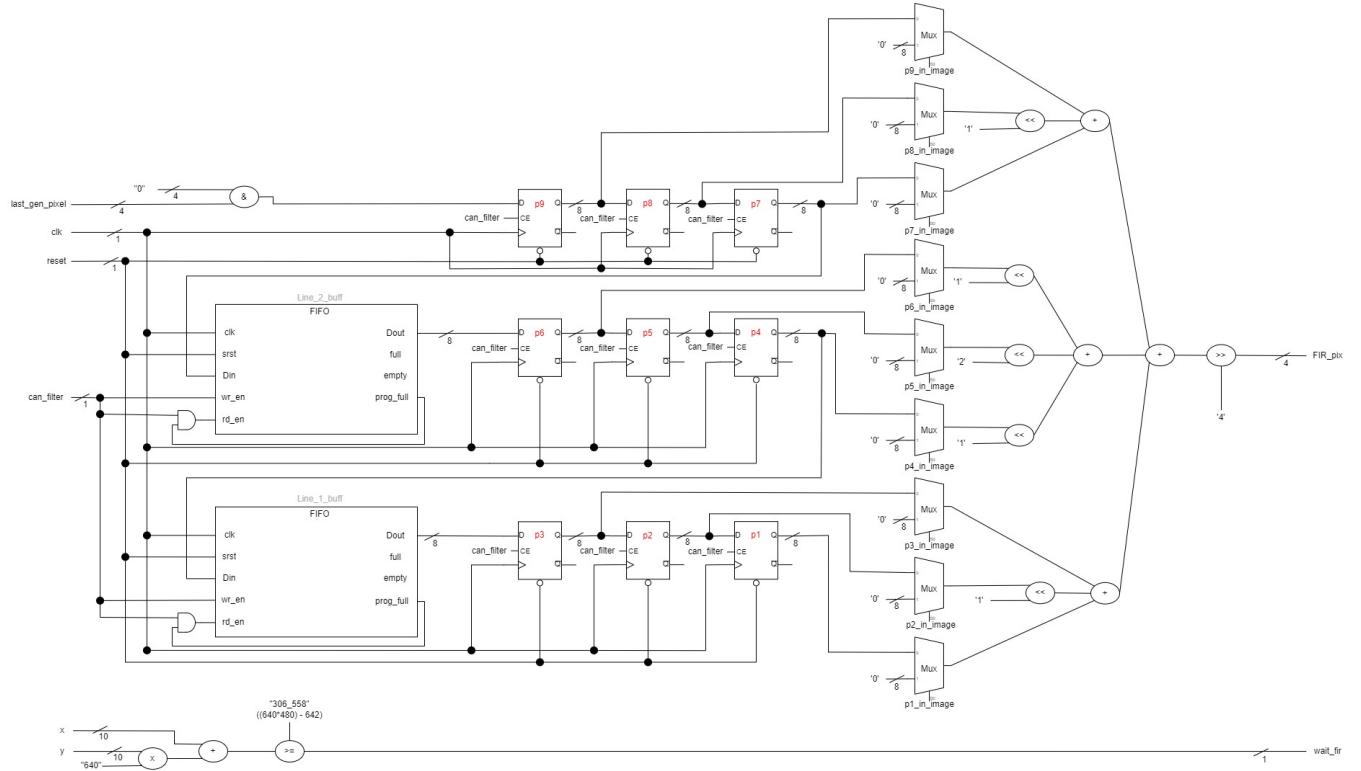
Ce noyau va représenter une "fenêtre glissante" qui va se déplacer sur chaque pixel (de gauche à droite, ligne par ligne.) de nos images pour calculer un nouveau pixel moyen en fonction de ses pixels voisins. Ce nouveau pixel au centre de notre cœur de convolution, représente notre pixel de sortie, puisqu'il aura été traité et donc "filtré", soit "flouté" dans notre cas.

Un point important est qu'il faut appliquer ce traitement d'image (et donc de calcul) à chaque sous-pixel de nos images, soit les canaux rouges, vert et bleus, afin de ne pas dénaturer notre image.

#### 5.1.1 Réalisation du filtre gaussien

Pour réaliser notre filtre gaussien, nous avons utilisé la méthode de cet [article](#) qui montre les différences de performances entre les méthodes de traitement d'images sur FPGA et sur CPU. La méthode pour implémenter ce filtre sur FPGA est d'utiliser 9 registres pour récupérer tous les pixels nécessaires au calcul de la convolution (9 registres, car notre noyau de convolution a une dimension de 3 x 3.) et d'effectuer ce calcul de manière combinatoire. Nous utilisons également des files (FIFOs) pour stocker les 2 plus anciennes lignes nécessaires au calcul.

Voici le schéma RTL du module du filtre gaussien correspondant à notre design :



**FIGURE 7 – Schéma RTL du filtre gaussien pour un canal**

Le schéma ci-dessus représente le filtre pour un des trois canaux rouge, vert, bleu, et est donc réutilisé sur les 2 canaux restants pour compléter le filtre sur le pixel complet. L'inconvénient de cette méthode est qu'il faut que le premier pixel générer par le générateur d'images se retrouve dans le registre **P5** pour que le filtre puisse commencer à calculer. On se rend alors compte que ceci entraîne un décalage entre la partie générateur d'images et *Filtre gaussien / pilote VGA*.

Quelques modifications, qui seront détaillées dans la sous-section suivante, ont donc été apportées dans les modules *VGA\_driver* et *Pattern\_generateur* afin de prendre en compte ces décalages.

On remarque la présence d'un signal *can\_filter*, qui sert à indiquer au filtre s'il peut effectuer son calcul de convolution. Tant que ce signal est à '0', on n'effectue aucune opération de décalage de registre, d'écriture ou de lecture dans les files.

On remarque également le signal *wait\_fir* qui sort du module du filtre, et qui servira à indiquer au générateur d'images que le filtre n'a pas terminer ses calculs afin qu'il puisse retarder la génération de la prochaine image.

### 5.1.2 Modifications liées au décalage du filtre

Comme indiqué précédemment, le filtre est soumis à un décalage dû au fait qu'il ait besoin d'un minimum de pixels pour commencer à calculer. Dans notre cas (pour un noyau de convolution de 3 x 3) il faut que le générateur d'images ait au minimum généré 1 ligne et 2 pixels pour que le filtre ait assez de données pour calculer le filtre gaussien sur le premier pixel de l'image. On doit donc ajouter de la logique au *pilote VGA* et au *générateur d'images* pour compenser ce décalage qui affectera l'affichage en sortie du pilote VGA.

Voici les versions modifiées du pilote VGA et du générateur d'images.

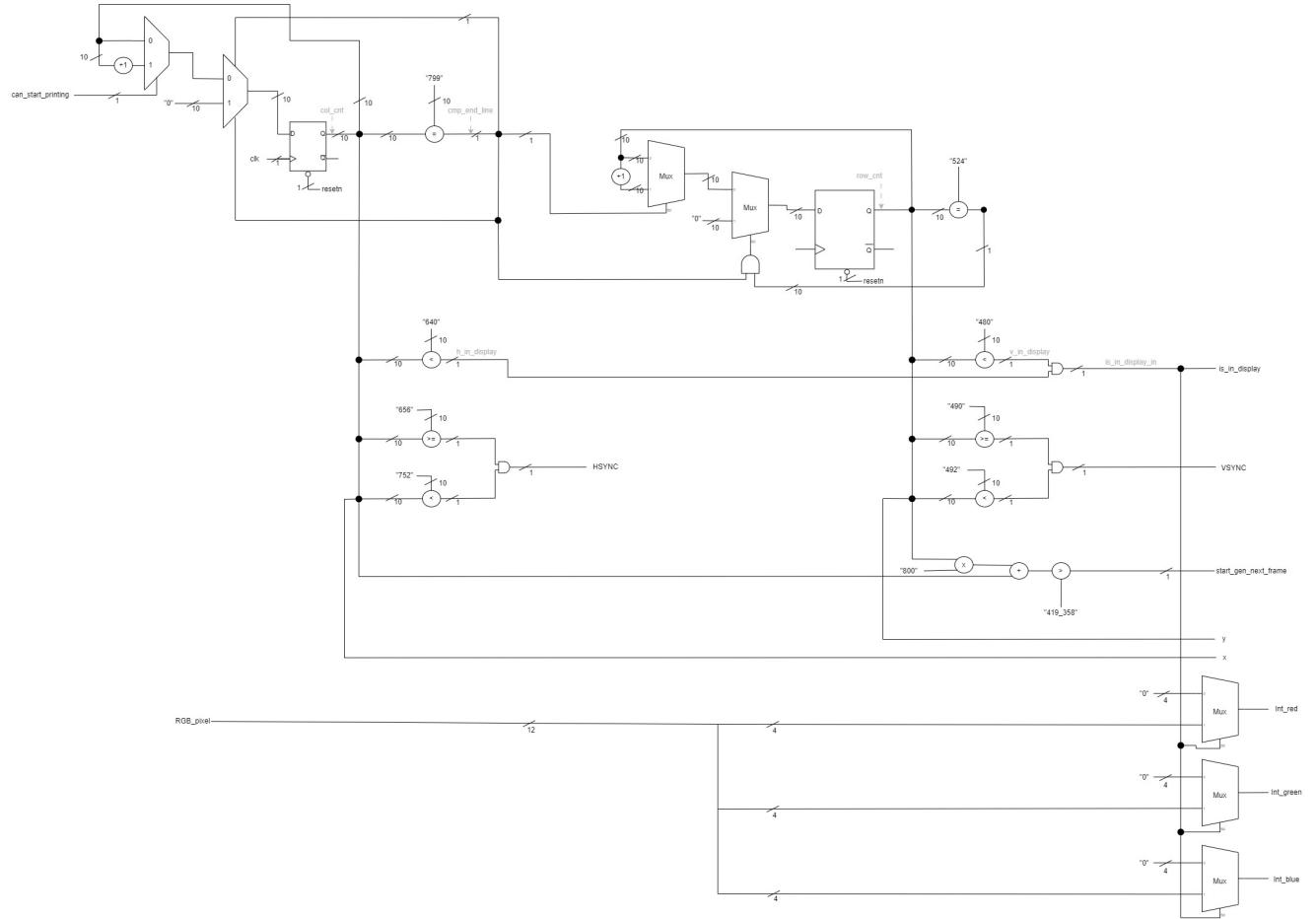
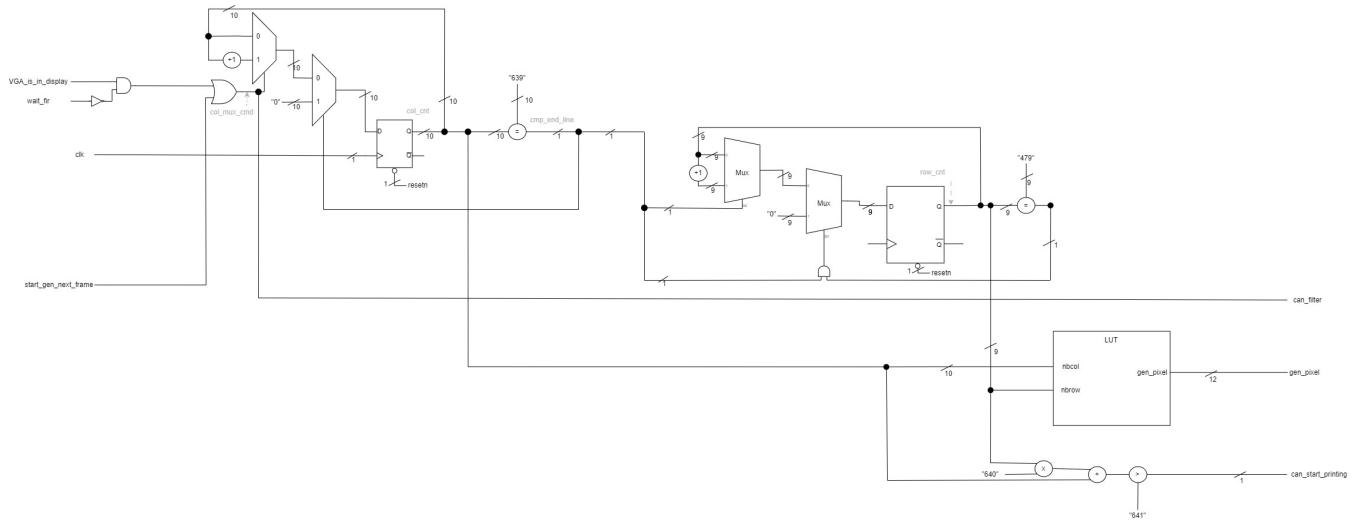


FIGURE 8 – Schéma RTL du pilote VGA modifié

On remarque la présence d'un nouveau signal *start\_gen\_next\_frame* dans le module *pilote VGA* qui nous servira à indiquer au générateur d'images qu'il peut commencer à générer l'image suivante exactement 1 ligne et 2 pixels avant la réinitialisation des compteurs de lignes et colonnes du pilote VGA, ce qui correspond au décalage initialement créé, et nous permet de ne pas modifier la logique liée aux signaux de synchronisations (HSYNC, VSYNC).

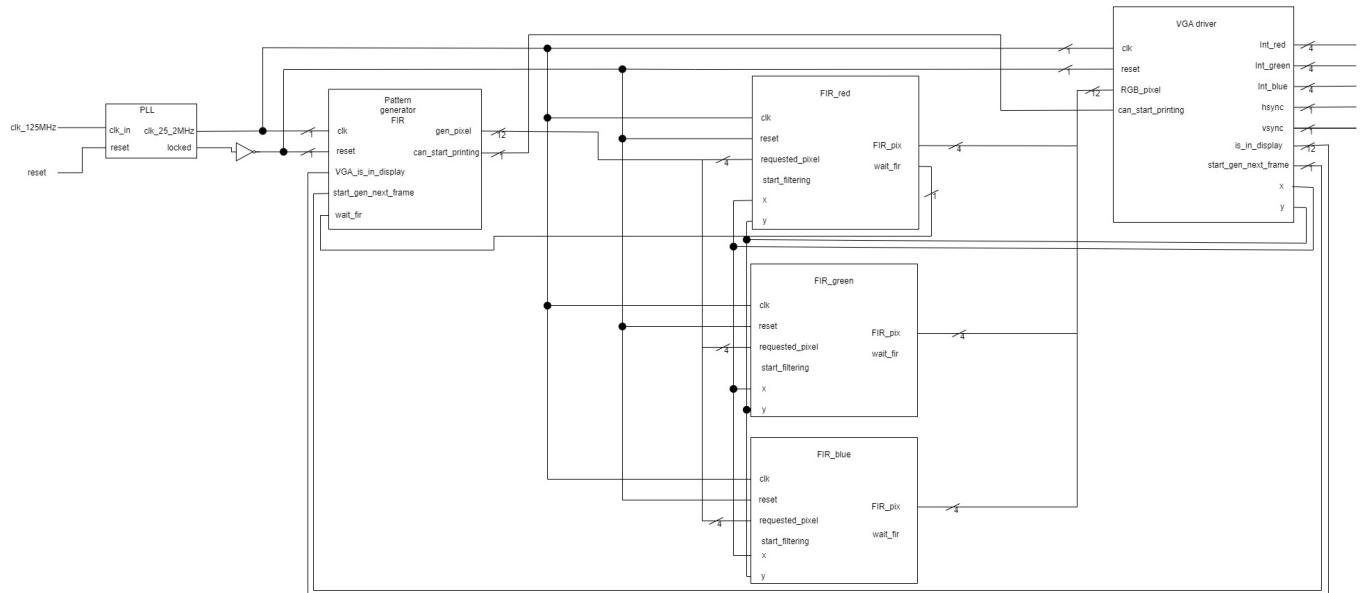
La partie "générateur d'image" a été plus modifiée que la partie du pilote VGA. On observe notamment la modification du compteur de colonnes, qui est maintenant incrémenté



**FIGURE 9 – Schéma RTL du générateur d'images modifiée**

de '1' lorsque l'on se trouve sur la surface affichable déterminée par le pilote et que l'on n'est pas en train d'attendre la fin du traitement du filtre ou que le pilote VGA nous indique de commencer à générer les nouveaux pixels (représentés par le signal `col_mux_cmd`). On émet également un nouveau signal `can_filter` qui sera donné en entrée du module du filtre gaussien afin de lui indiquer s'il peut effectuer les opérations de décalages de pixels dans les registres/FIFOs en accord avec le comportement du pilote VGA, car autrement le générateur aurait envoyé trop de données au module du filtre ce qui aurait causé des décalages au niveau de l'image résultante.

Voici donc le schéma RTL du projet entièrement complété :



**FIGURE 10 – Schéma RTL du projet VGA + filtre gaussien**

## 6 Analyse

Dans cette section, nous allons nous concentrer sur les résultats de simulation obtenue à partir des tests évoqués dans le plan de validation. Nous allons commencer par observer le résultat de simulation dans son entièreté.

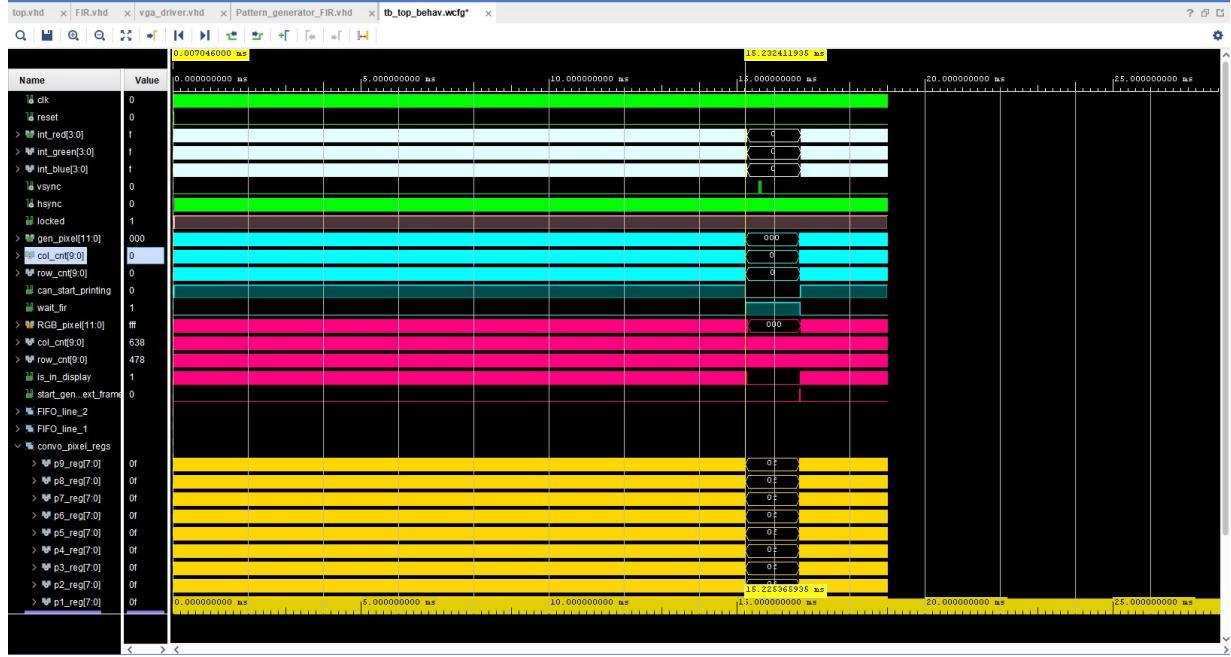
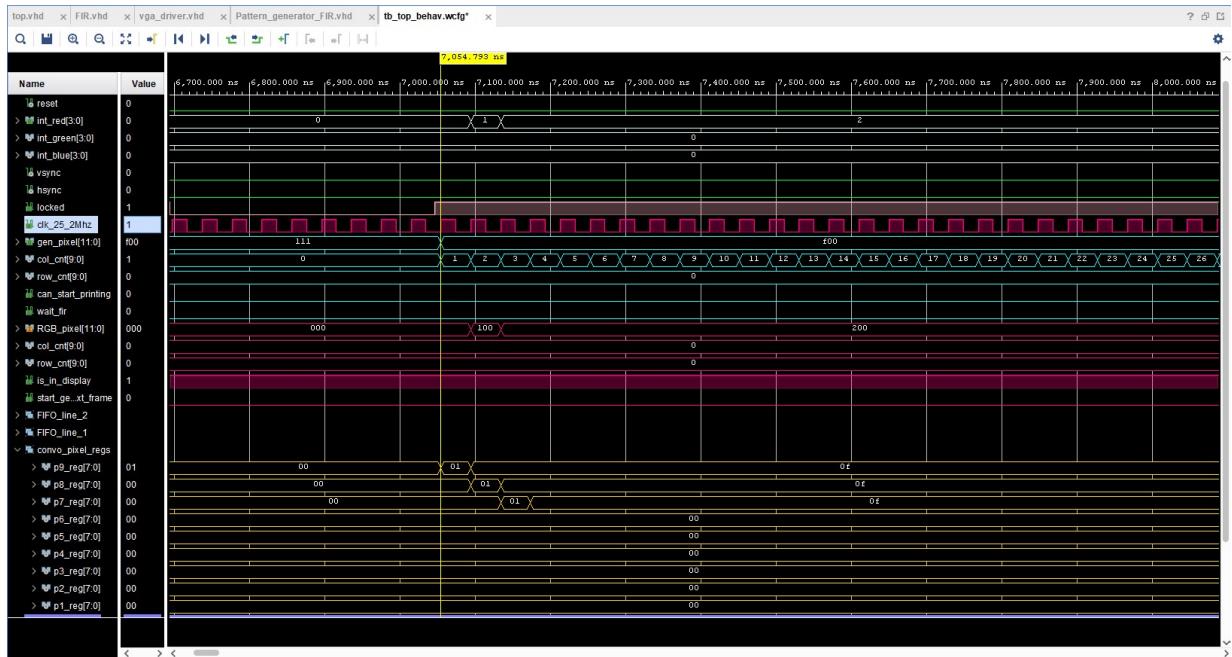


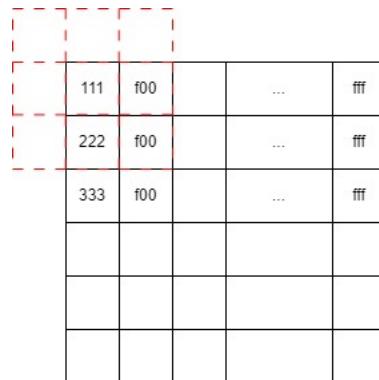
FIGURE 11 – Chronogramme de la simulation - image complète

On peut voir sur l'image ci-dessus une image entièrement générée par le générateur d'image, filtré par le module de filtrage et affiché à l'écran. On profite de cette vue pour vérifier le temps mis par le générateur pour générer une image, afin de s'assurer que la norme VGA est bien respectée. On voit donc sur l'image que ce temps est d'environ 15.23ms, ce qui correspond à ce qui est spécifié par la norme. On observe également le décalage entre la fin de la génération de l'image et la fin de l'affichage des pixels "visibles" par le pilote VGA qui correspond au décalage explicité dans la section précédente.



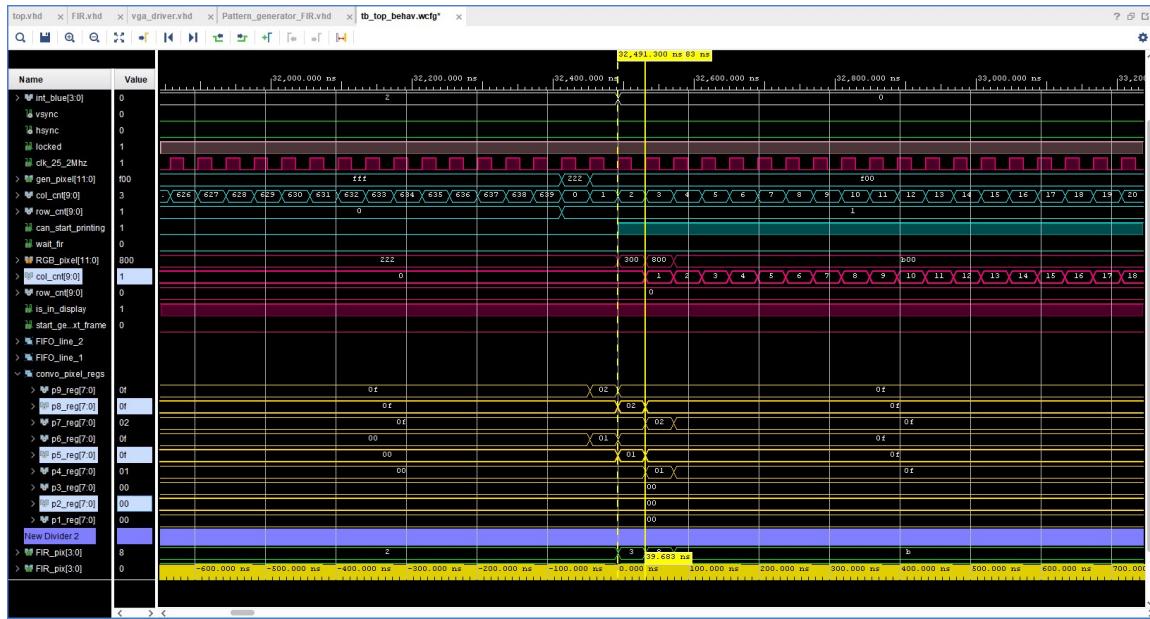
**FIGURE 12** – Chronogramme de la simulation - zoom sur le début de la génération d'image

Sur le chronogramme ci-dessus, on se concentre sur le début de la génération de l'image afin de s'assurer que certains pixels "spéciaux" que nous avons placés sur les trois premiers pixels des trois premières lignes se positionnent correctement dans notre système de registre à décalage servant au calcul du flou gaussien. On essaiera donc de retrouver la configuration du schéma ci-dessous, en prenant en compte que la fenêtre en pointillé rouge représente le noyau de convolution que l'on appliquera à l'image.



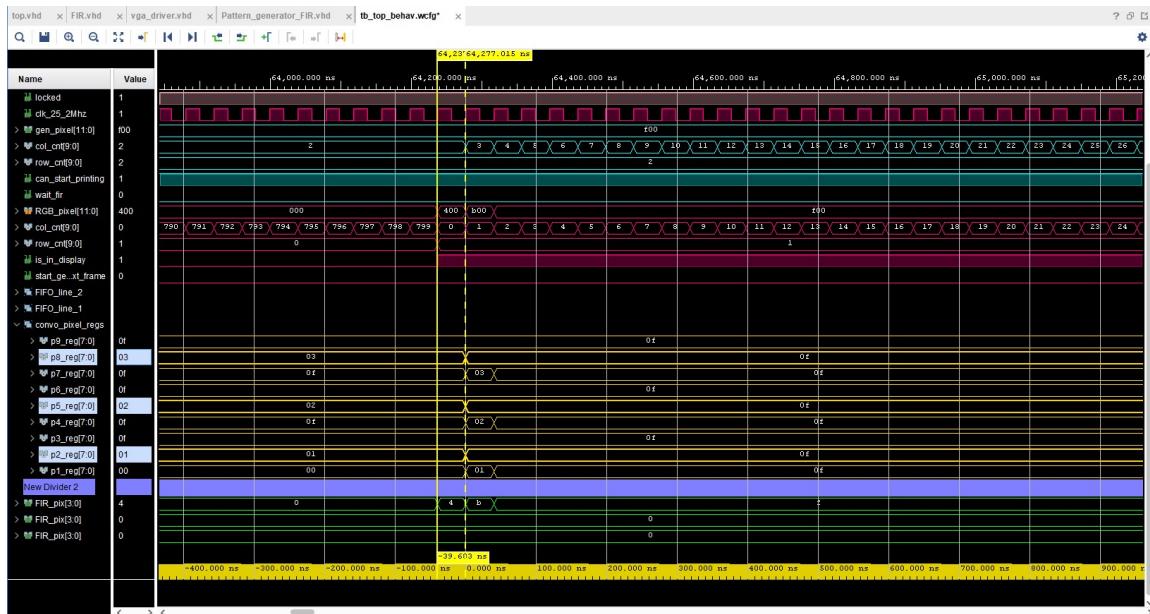
**FIGURE 13** – schéma représentant l'image test pour le calage

On observe ici le pixel de valeur "111" qui passe correctement dans le module de flou gaussien du canal rouge, et qui se décale correctement à chaque incrément du compteur du générateur d'image (registre p9 à p7) avant d'entrer dans la file pour en ressortir à la ligne suivante.



**FIGURE 14** – Chronogramme de la simulation - zoom sur le début du calcul du flou

Sur le chronogramme ci-dessus, on se concentre sur le début du calcul dur filtre (lorsque le générateur a généré 1 ligne et 2 pixels) afin de retrouver notre valeur de sous-pixel spécial pour la première ligne, ici "1", ainsi que celle pour la deuxième ligne qui vaut "2". Nos pixels sont donc correctement "calés" dans nos registres, et les calculs de convolutions seront corrects.



**FIGURE 15** – Chronogramme de la simulation - analyse de la 2<sup>e</sup>me ligne de l'image

Sur ce dernier chronogramme, on observe le même comportement que sur le chronogramme précédent, mais avec cette fois-ci l'apparition du sous-pixel spécial de la troisième ligne.

## 7 Test

Dans cette section, nous allons nous concentrer sur les résultats de mesures effectuées à l'aide d'un oscilloscope. Ces mesures vont nous permettre de nous assurer que les signaux de synchronisations respectent les timings de la norme en sortie de la carte coraZ7.

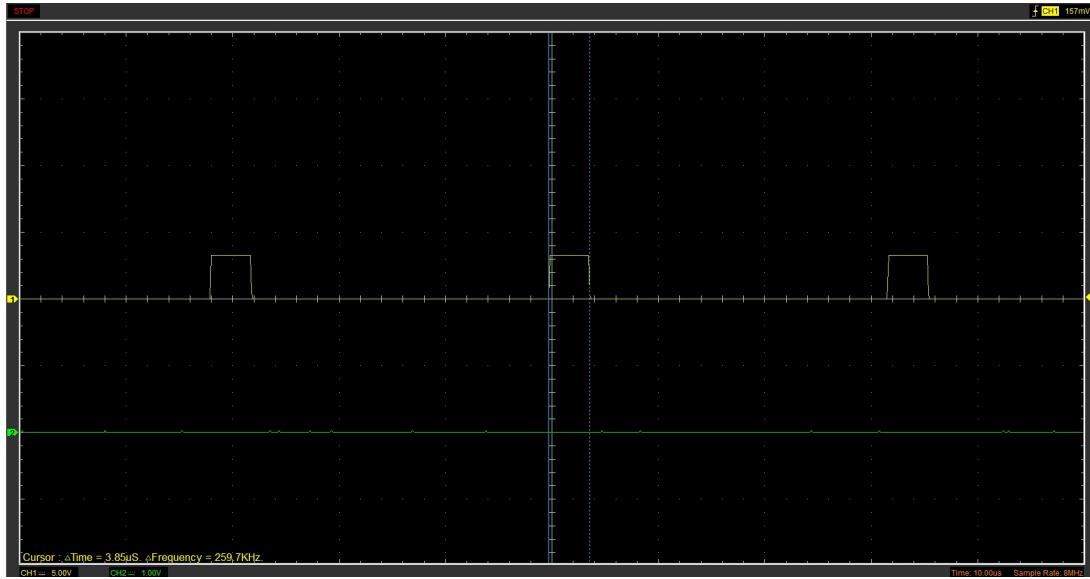
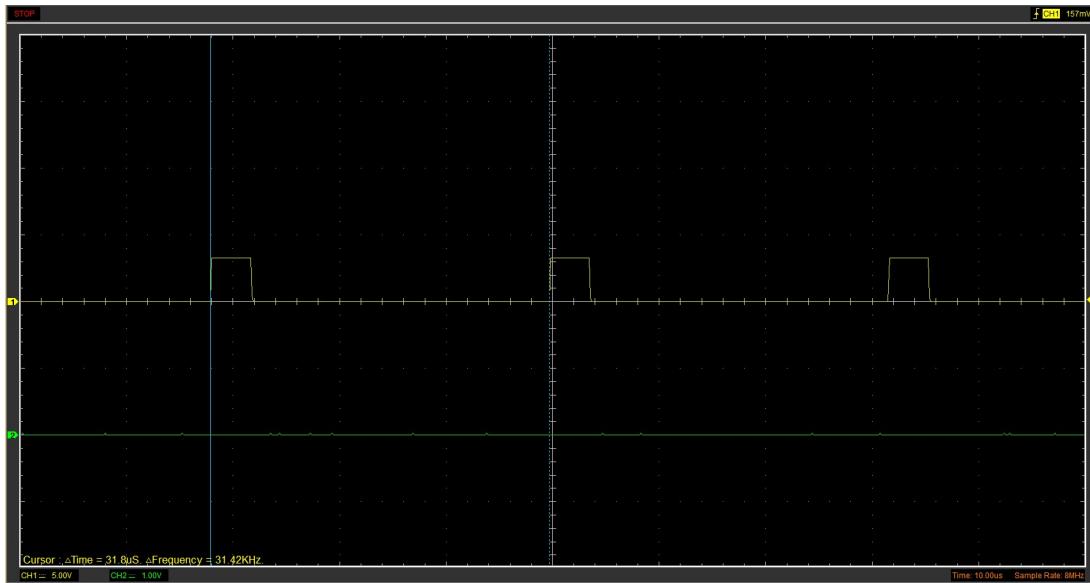


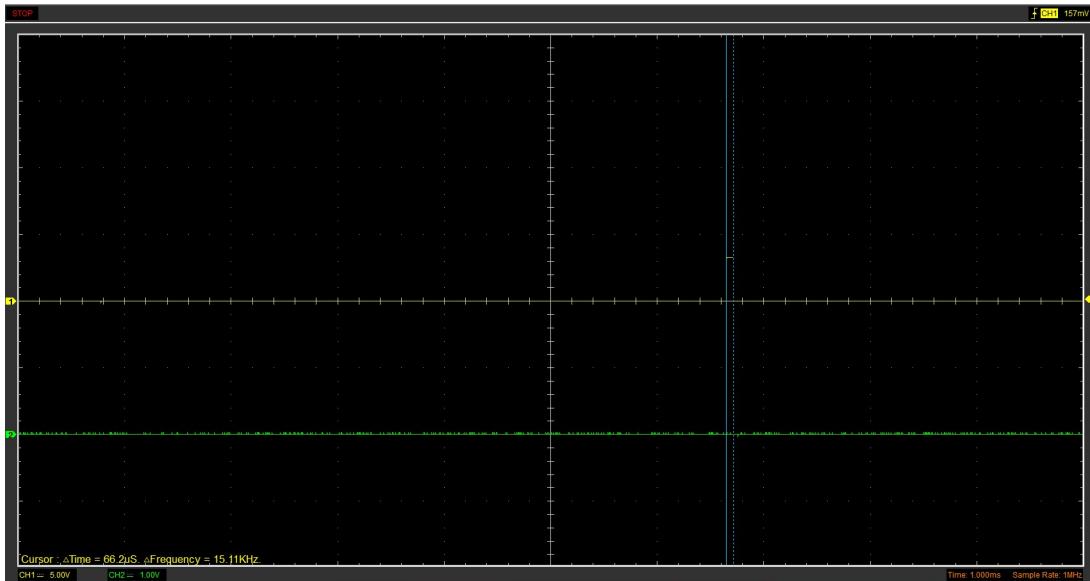
FIGURE 16 – Oscilloscope - impulsion HSYNC

Sur la capture ci-dessus, on peut voir que le temps durant lequel le signal *HSYNC* est à l'état haut est de 3.85μs, ce qui correspond à ce qui est spécifié dans la norme VGA.



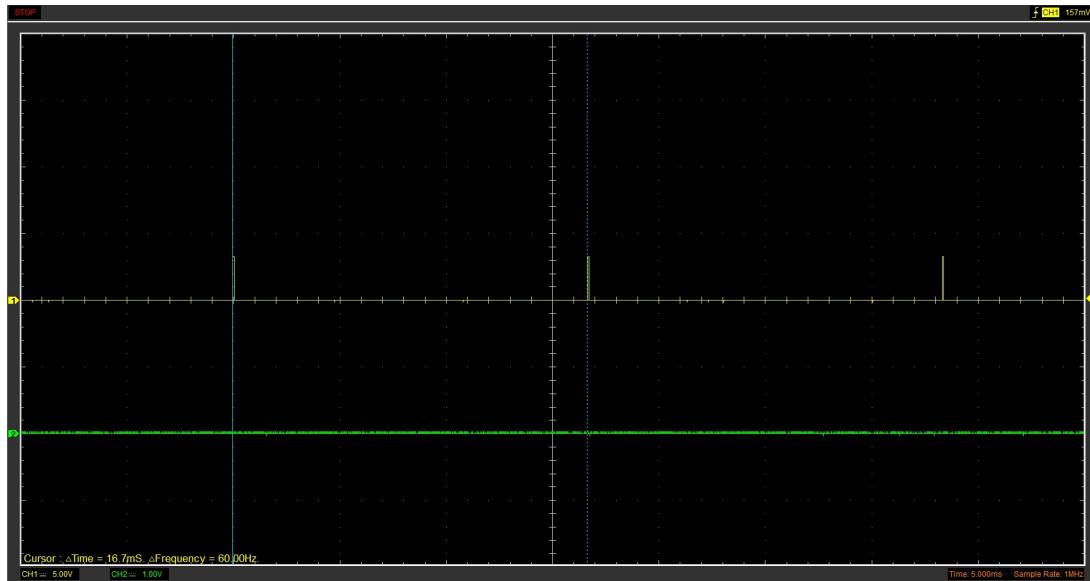
**FIGURE 17 – Oscilloscope - durée d'affichage d'une ligne**

Sur la capture ci-dessus, on peut voir que le temps mis pour afficher une ligne est d'environ 31.8 $\mu$ s, ce qui correspond également à ce qui est spécifié dans la norme VGA.



**FIGURE 18 – Oscilloscope - impulsion VSYNC**

Sur la capture ci-dessus, on peut voir que le temps durant lequel le signal *VSYNC* est à l'état haut est de 66.2 $\mu$ s, ce qui correspond une fois de plus à ce qui est spécifié dans la norme VGA.



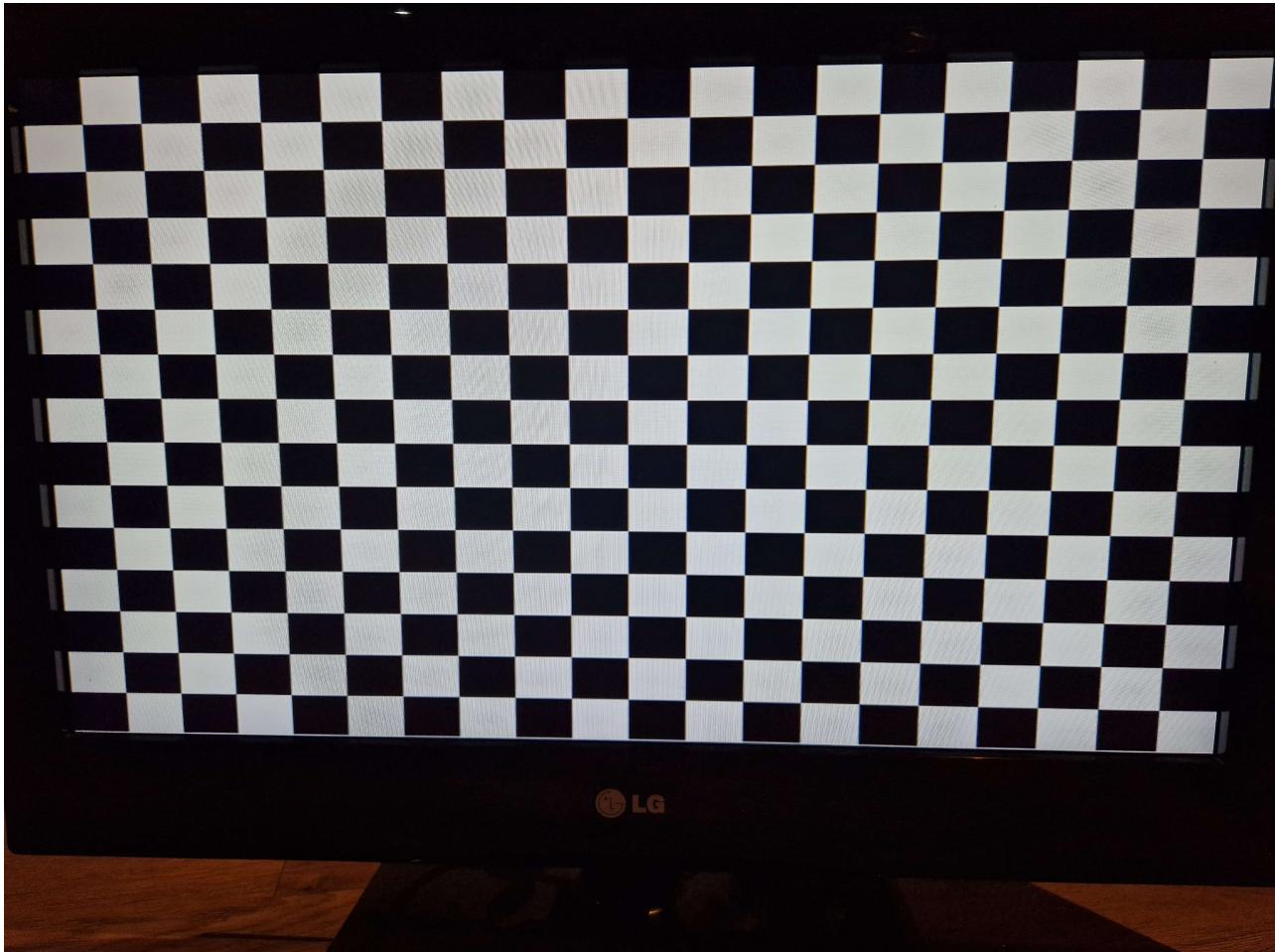
**FIGURE 19** – Oscilloscope - durée d'affichage d'une image

Sur la capture ci-dessus, on peut voir que le temps mis pour afficher une image est d'environ 16.7ms, ce qui correspond à ce qui est spécifié dans la norme VGA.

Au vu de toutes ces mesures, nous pouvons donc affirmer que nos résultats sont valides, car conformes à ce que la norme VGA indique pour ces signaux.

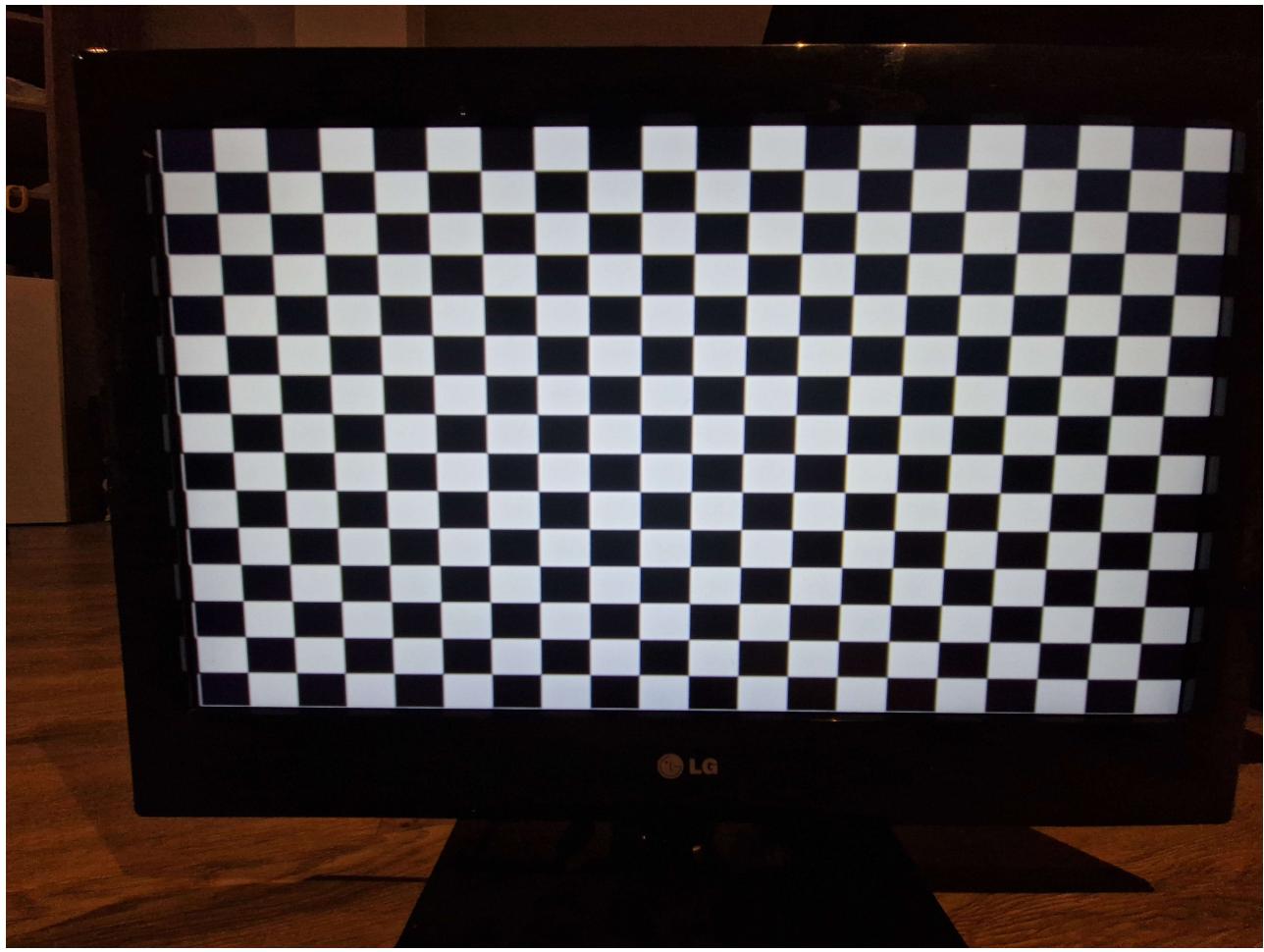
## 8 Démonstration

Dans cette section, on s'intéresse aux résultats obtenus sur l'écran. On va donc comparer les résultats obtenus avant et après le passage du filtre.



**FIGURE 20** – Image avant passage du filtre

Sur l'image ci-dessus, on peut voir notre motif quadrillé sans le passage du filtre gaussien (ce que l'on remarque par la netteté des bords des carrés).



**FIGURE 21** – Image après passage du filtre

Sur l'image ci-dessus, on peut voir notre motif quadrillé après le passage du filtre gaussien. On peut apercevoir sur l'image que les bords des carrés sont moins nets que sur l'image précédente, ce qui nous indique que notre filtre a été correctement calculé.