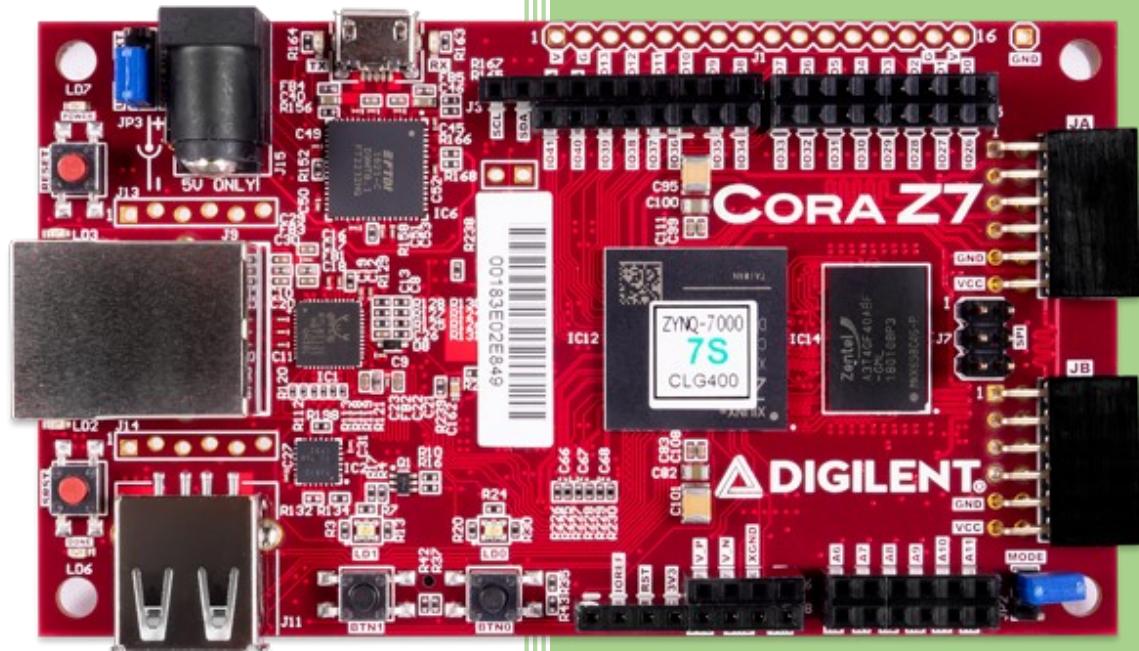


2023

Réalisation d'une IP de traitement d'image sur cible Zynq7020 avec affichage VGA



Sefofo SOKPOR &
Éric GASNIER

AJC

28/06/2023

Table des matières

I.	Introduction	3
II.	Étude de la phase intermédiaire : affichage d'un damier	4
1)	Analyse	4
•	Module PLL	5
•	Module VGA_sync	5
•	Le module Gen_mir	7
2)	Test.....	8
•	Test sur les composantes du module VGA_sync	8
•	Le module Gen_mir	11
•	Le top VGA.....	11
3)	Démonstration.....	14
•	Rappel du matériel à disposition	17
III.	Étude de la phase finale : affichage d'un damier filtré.....	19
1)	Analyse	20
•	Le module PLL.....	20
•	Le module VGA_sync.....	20
•	Le module Gen_mir	20
•	Le module Shift_sync.....	20
•	Le module System_conv	22
2)	Test.....	28
3)	Démonstration.....	31
•	Matrice identité	33
•	Flou Gaussien.....	33
•	Sobel horizontal.....	35
•	Sobel vertical.....	35
•	Detection de Edge	36
IV.	Phase supplémentaire : déplacement d'un carré sur l'écran VGA.....	37
V.	Conclusion.....	39
	ANNEXE	40
•	Convolution module / Autres filtres.....	40
•	Convolution module / Gestion des bords	41
•	Schémas de la phase supplémentaire.....	42
•	Bibliographie.....	43

Date	Affectation	Version	Auteurs
12/06/2023	Création	V0	SAS & EG
16/06/2023	Phase intermédiaire	V1	SAS & EG
28/06/2023	Phase finale	V2	SAS & EG

I. Introduction

Dans le contexte d'un mini-projet dans le cadre de la formation AJC-FPGA pour l'entreprise SAFRAN, il nous a été demandé la réalisation d'une IP de traitement d'image sur cible Zynq7020 et l'affichage par VGA.

L'objectif de ce document est d'étudier dans un premier temps la faisabilité de ce projet, c'est-à-dire évaluer la possibilité de programmer une carte FPGA (CoraZ7) en utilisant le langage VHDL afin de transformer la carte FPGA en un contrôleur VGA capable de produire des images au format 640x480 pixels. Ensuite, les phases d'analyse, de test et de démonstration viendront valider notre étude.

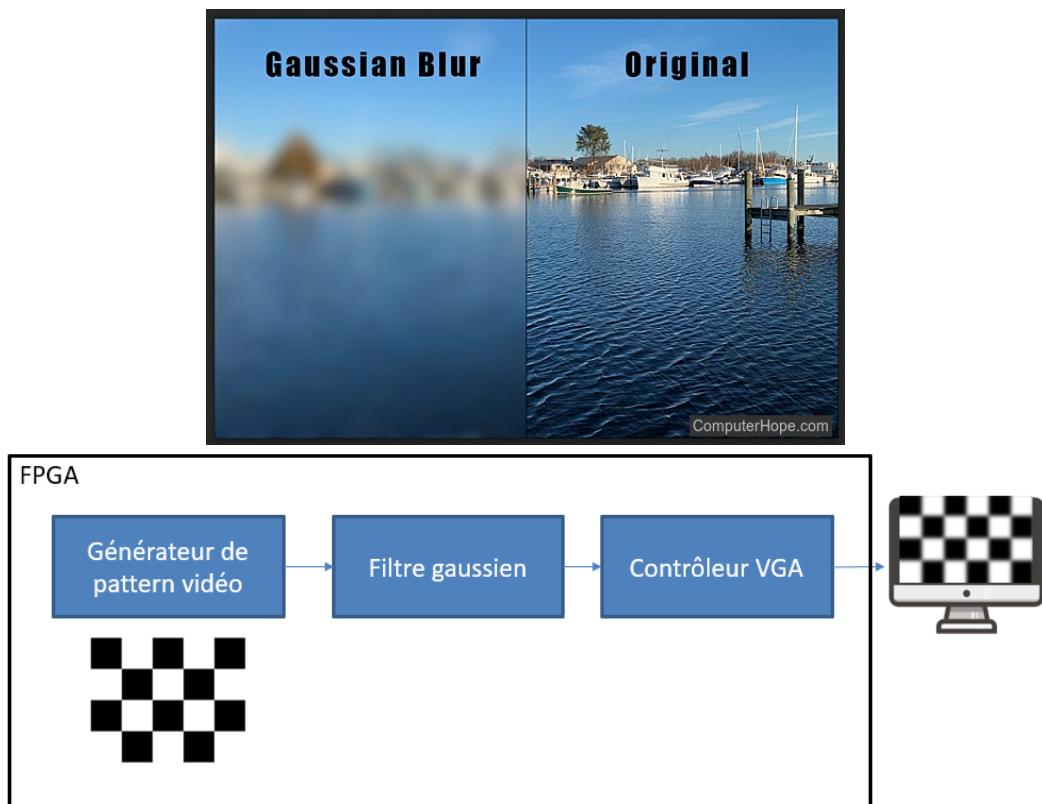


Fig. 1 : Principe de filtrage

Dans l'étude de la faisabilité, nous avons examiné les aspects techniques et les ressources nécessaires pour réaliser ce projet consigné dans le document V0.

La conception logique de notre projet se réalise en suivant les étapes énumérées ci-dessous :

- La phase intermédiaire où nous traiterons l'affichage de l'image sans filtre,
- La phase finale avec l'affichage de l'image filtrée.

II. Étude de la phase intermédiaire : affichage d'un damier

Dans un premier temps, nous allons chercher à afficher sur un écran un damier.

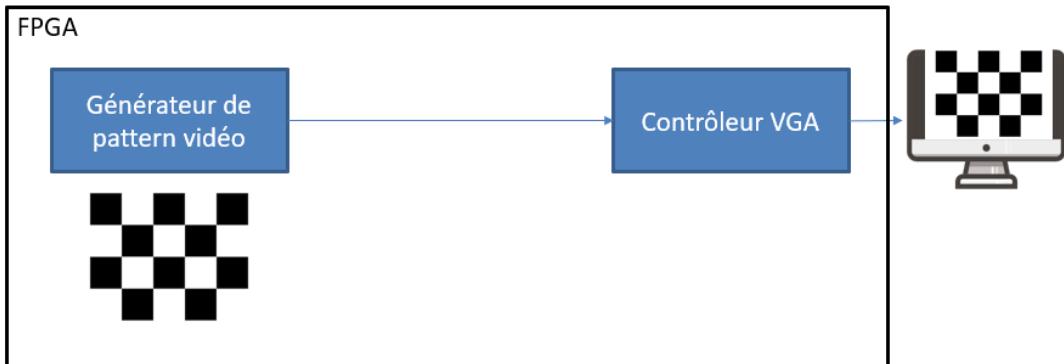


Fig 2 :

Ainsi, nous allons devoir concevoir une architecture sur la carte coraZ7 permettant :

- De générer les signaux de synchronisation (hsync et vsync),
- De générer les signaux RGV.

La carte Pmod VGA sera connectée sur les connecteurs Pmod JA et JB de la carte Xilinx afin de réaliser la conversion numérique/analogique. Enfin, un câble VGA reliera la carte Pmod VGA à un écran.

1) Analyse

Dans cette partie, l'objectif est de générer une image de damier sur un écran avec l'affichage de VGA. Ce sont les résultats obtenus en mettant en œuvre une modélisation et une simulation d'une partie du système.

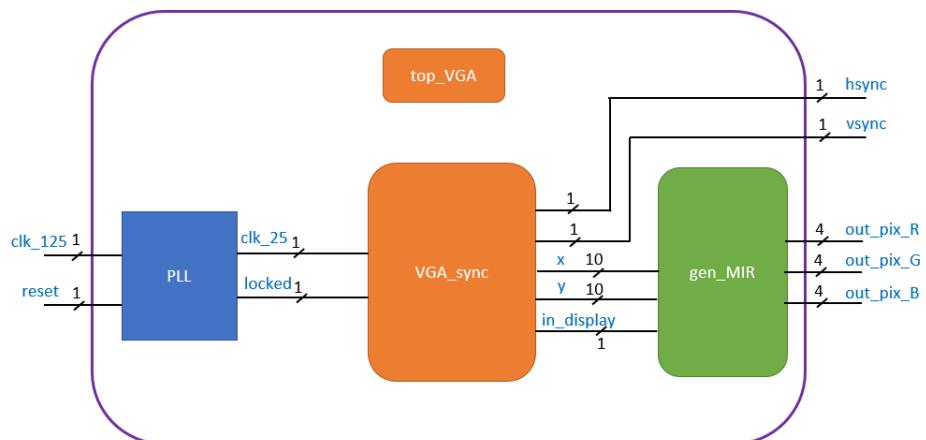
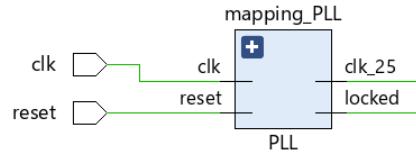


Fig. 3 : synoptique général

Pour afficher l'image à l'écran, nous avons besoin de trois modules.

• Module PLL

Nous débutons par la PLL qui va générer une horloge clk_25 à 25,175 MHz, fréquence de balayage de l'écran pixel par pixel.



Les entrées de ce module sont :

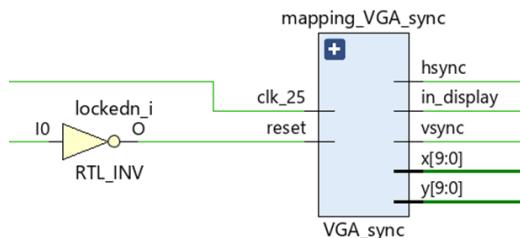
Un signal d'horloge clk à 125 MHz, fréquence générée par un oscillateur présent sur la carte coraZ7. Un signal reset de réinitialisation, actif à l'état haut.

En sortie :

Un signal d'horloge clk_25 à 25,175 MHz, fréquence de balayage de l'écran pixel par pixel et un signal locked de stabilisation de l'horloge, actif à l'état bas.

• Module VGA_sync

Ensuite le module vga_sync qui va générer les signaux nécessaires au balayage de l'écran.

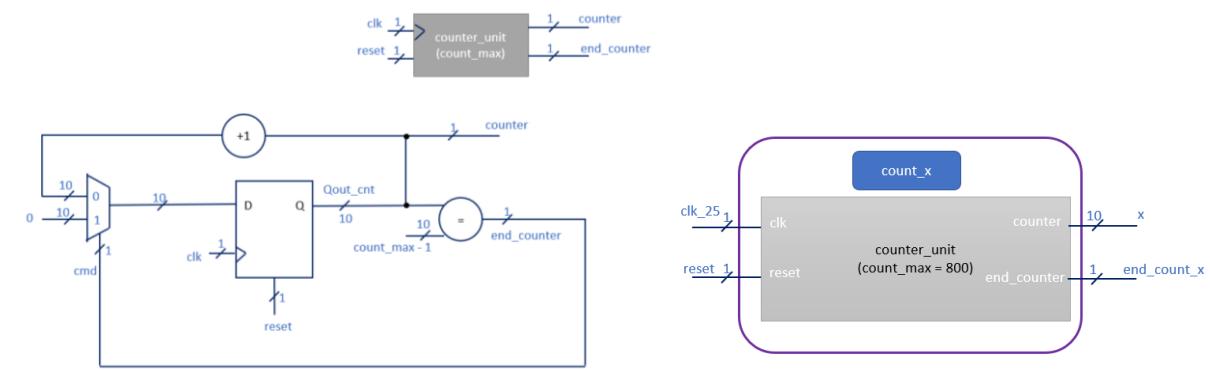


Les entrées de ce module sont : un signal d'horloge clk_25 à 25,175 MHz, fréquence de balayage de l'écran pixel par pixel ; un signal lockedn (l'inverse de reset), signal de stabilisation de l'horloge (à 1 lorsque stable sinon à 0), et qui pourra être activé sur appui d'un bouton-poussoir intégré en surface de la carte.

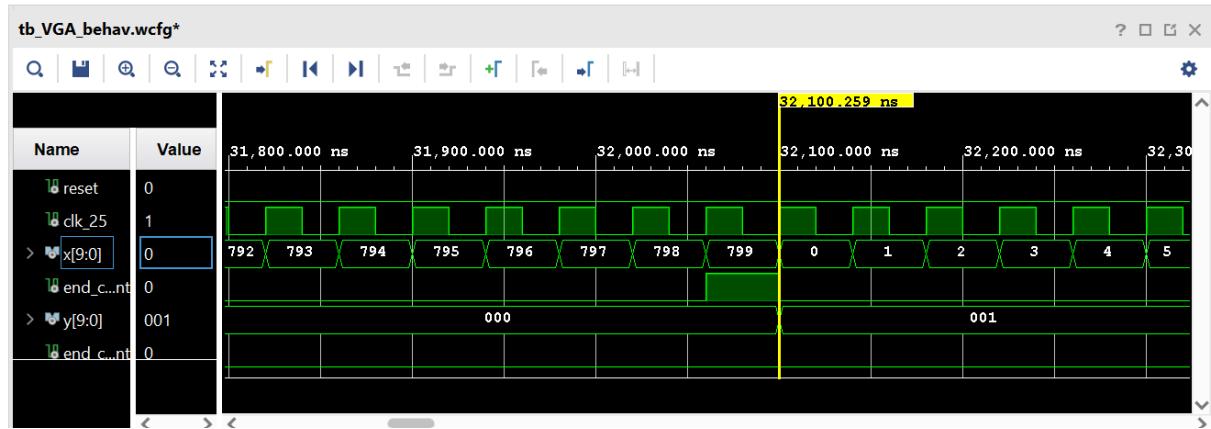
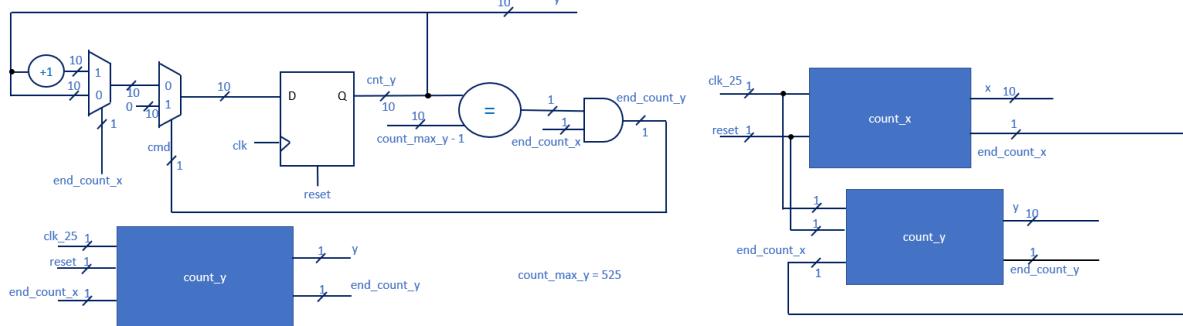
En sortie : les signaux logiques de synchronisation horizontale et verticale hsync et vsync ; les coordonnées du point en cours de la surface balayée x (entre 0 et 799) et y (entre 0 et 524) sur 10 bits évoluant selon le sens de balayage à la fréquence de l'horloge 25,175 MHz ; un signal logique inDisplayArea à l'état haut lorsque les coordonnées du pixel en cours sont dans la zone active d'affichage (x entre 0 et 639, et y entre 0 et 479) et à l'état bas sinon.

Pour générer x et y, nous avons utilisé des compteurs. Les schémas RLT sont les suivants.

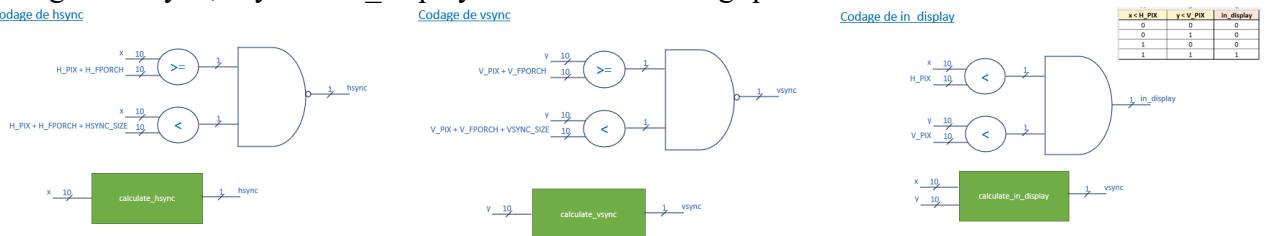
x = position horizontale du pixel dans l'image.



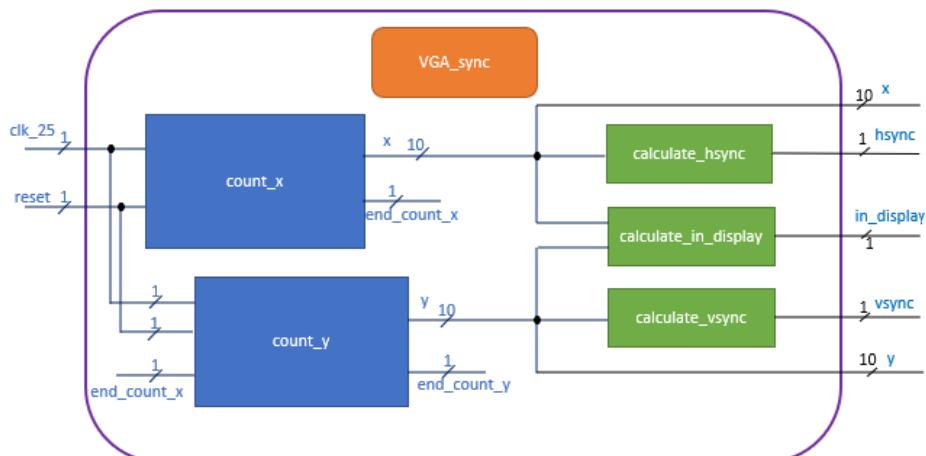
y = position verticale du pixel dans l'image.



Les signaux hsync, vsync et in_display sortent avec des logiques combinatoires.

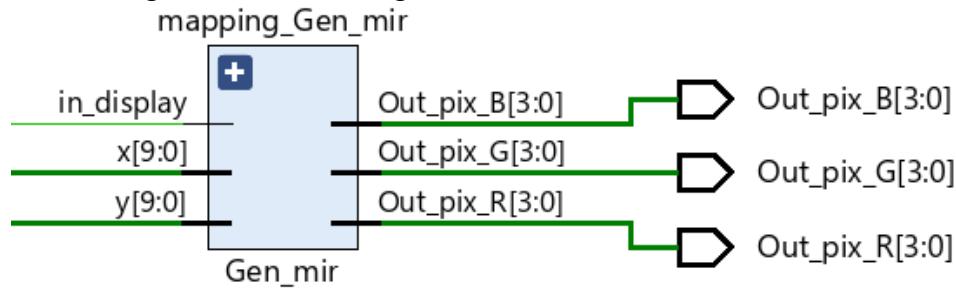


Le schéma RTL donne cet ensemble. Le code vhdl est disponible sur git sous le nom de VGA_sync



- Le module Gen_mir

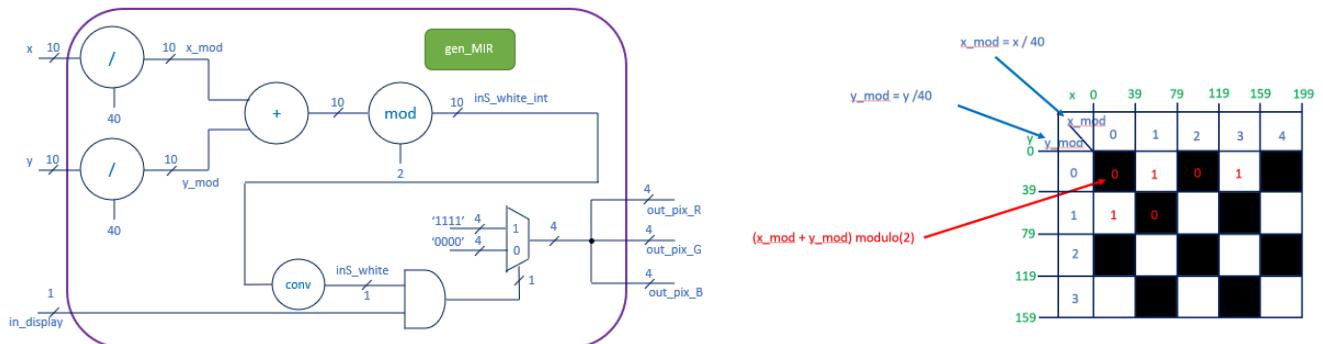
Passons au module générateur de l'image.



En entrée de ce module sont : les signaux logiques des coordonnées du point en cours de la surface balayée x (entre 0 et 799) et y (entre 0 et 524) ; un signal logique in_display.

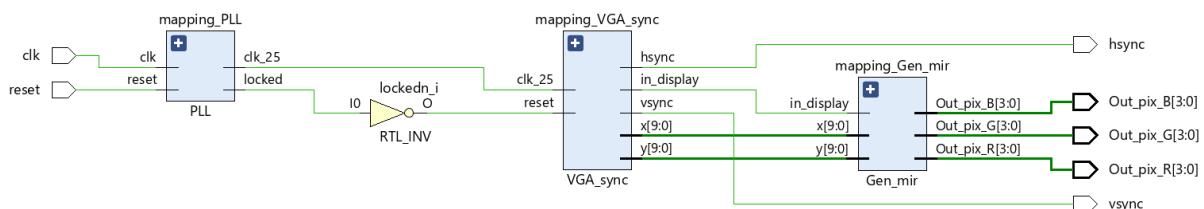
En sortie : les signaux logiques synchronisés des composantes Rouge-Vert-Bleu pendant le balayage des pixels. Out_pix_R, Out_pix_B et Out_pix_G.

Le schéma RTL est donné ci-dessous.



Pour créer l'image, nous avons utilisé les parties entières des divisions de x et y par la taille de du motif (40 pixels par exemple). Ensuite, en additionnant les résultats, nous obtenons des nombres pairs ou impairs suivant la couleur des cases. Ainsi, une opération modulo 2 nous permet d'extraire un 0 ou un 1 suivant la couleur des cases.

La synthèse finale de cette phase intermédiaire est ce schéma ci-dessous (menu Tools → Netlist Viewers → RTL Viewer) :



Le bloc `pll_Clock` est un circuit spécialisé disponible dans la bibliothèque de composants de Xiling (IP Catalogue) : une boucle à verrouillage de phase ou PLL (phase-locked loop) pour asservir la fréquence de sortie sur un multiple de la fréquence d'entrée. L'entrée du bloc est raccordée à l'horloge principale 125 MHz de la carte FPGA (entrée CLOCK_125).

Le bloc du module vga_sync est celui détaillé ci-dessus. En entrée, il est raccordé à l'horloge 25,175 MHz et au signal lockedn généré par l'appui d'un bouton-poussoir en surface de la carte FPGA. Les signaux gérant le mouvement de balayage sont produits sur ses sorties.

Le module gen_mir se charge de générer les signaux synchronisés des composantes Rouge-Vert-Bleu pendant le balayage des pixels. Chaque composante de couleur sera soit allumée, soit éteinte.

Les codes d'implémentations et les testsbenchs sont disponibles sur git.

Intégration des modules : Nous avons intégré les différents modules pour former le système complet avec un testbench et une simulation sur vivado. Nous nous assurons que les interfaces entre les modules sont correctement définies et que les signaux sont transmis correctement entre eux.

2) Test

Dans cette partie, nous affichons les résultats obtenus par simulation et mesure physique (Scope ou ILA) du système soumis à notre pattern d'entrée.

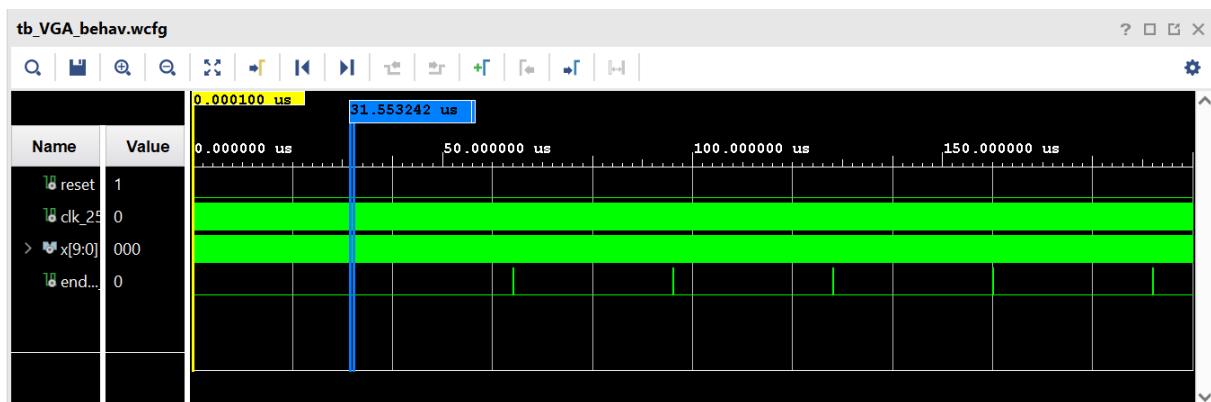
- Simulation : Nous avons effectué des simulations pour vérifier le bon fonctionnement de notre système, et pour tester différentes configurations et scénarios.

- **Test sur les composantes du module VGA_sync**

- X

➤ PRINCIPE DES TESTS

-
- On compte les fronts montants d'horloge.
 - On vérifie que le compteur x s'incrémente dans le même timing.



- PREMIER TEST : INCRÉMENTATION DU COMPTEUR x

- Le signal x s'incrément de 0 à 798
- Le signal end_count_x doit rester à 0 durant ce comptage

- DEUXIÈME TEST : TEST DE LA FIN DU COMPTAGE

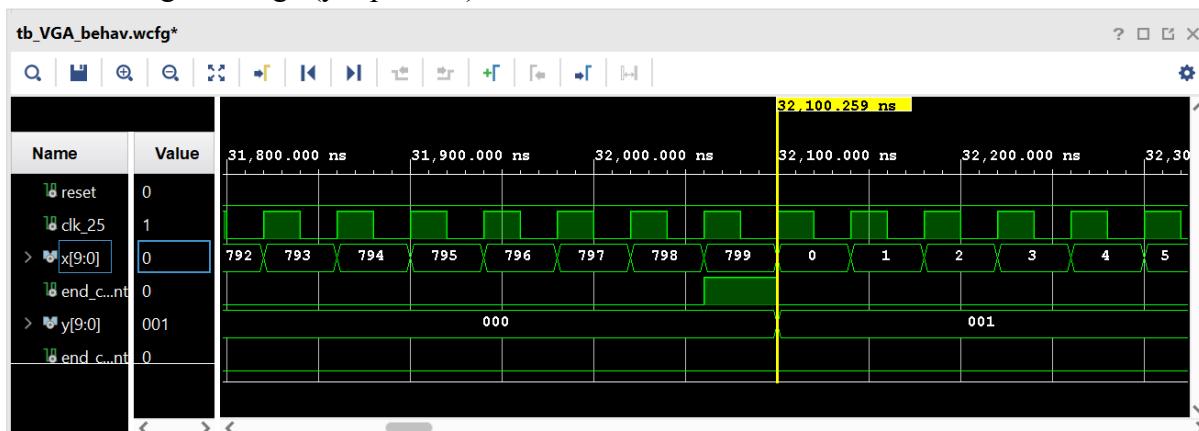
- x passe à 799
- Le signal end_count_x doit prendre la valeur 1



Nous concluons que x marche bien comme nous le désirons.

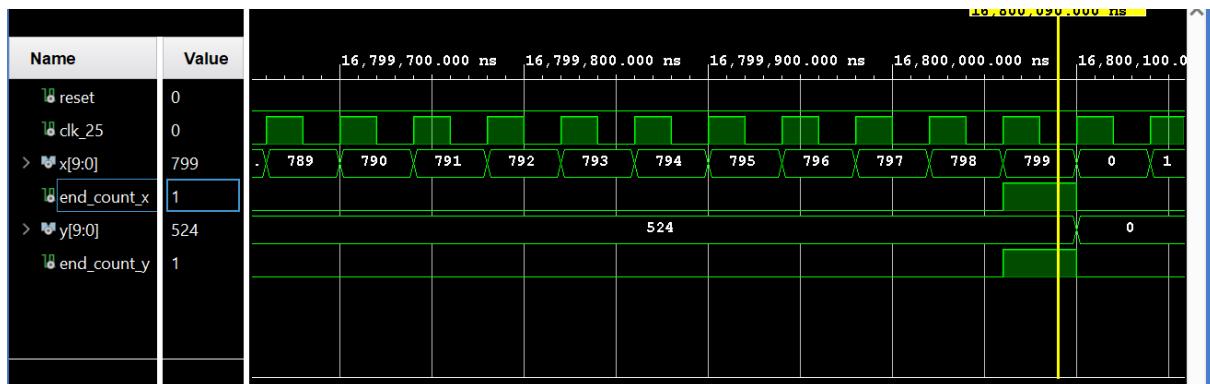
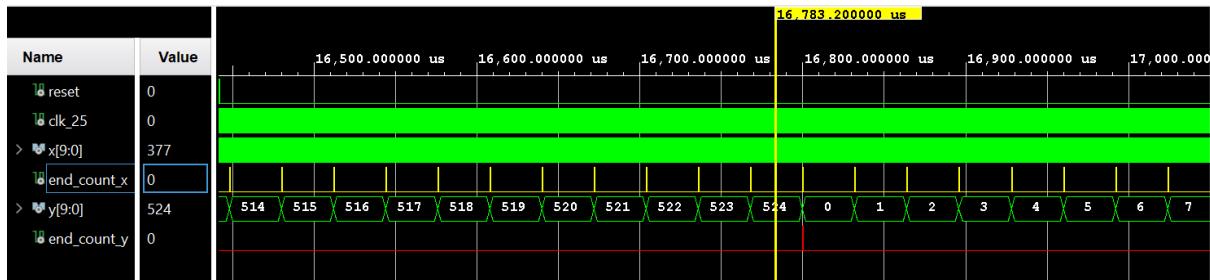
- [y](#)
- PRINCIPE DES TESTS

-
- On compte les fronts montants d'horloge.
 - On vérifie que le compteur x s'incrémente
 - dans le même timing et génère un signal end_count_x.
 -
 - À chaque réception du signal end_count_x,
 - on change de ligne et la valeur de y s'incrémente.
 - Lorsque le nombre max de lignes est atteint,
 - on génère un signal end_count_y
 - et on change d'image (y repart à 0).



-
- PREMIER TEST : INCREMENTATION DU COMPTEUR y
 - Le signal y s'incrémente de 0 à 523
 - Le signal end_count_y doit rester à 0 durant ce comptage

 - DEUXIÈME TEST : CHANGEMENT D'IMAGE
 - y passe à 524 (dernière ligne)
 - Le signal end_count_y passe à 1 à la fin de la dernière ligne

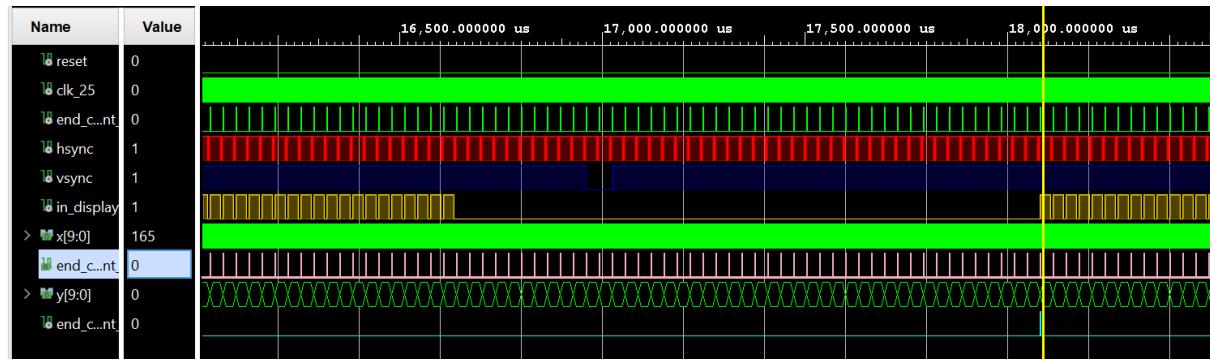


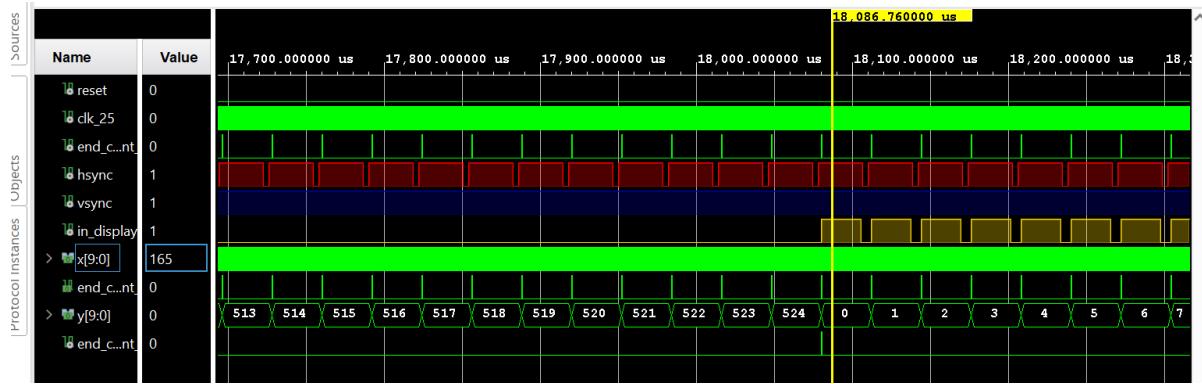
- VGA_sync
- PRINCIPE DES TESTS

— PREMIER TEST : TEST POUR LES VALEURS DE Y DANS L'IMAGE
— ($y < V_{PIX}$)
— BALAYAGE DE y DE 0 A 479
— BALAYAGE DE x DE 0 A 639
— Valeurs des sorties attendues
— hsync = 1 (inactif)
— vsync = 1 (inactif)
— in_display = 1 (dans l'image)
— BALAYAGE DE x DE 640 A 655
— Valeurs des sorties attendues
— hsync = 1 (inactif)
— vsync = 1 (inactif)
— in_display = 0 (dans la zone blanche)
— BALAYAGE DE x DE 752 A 799
— Valeurs des sorties attendues
— hsync = 1 (actif)
— vsync = 1 (inactif)
— in_display = 0 (dans la zone blanche)
— BALAYAGE DE x DE 752 A 799
— Valeurs des sorties attendues
— hsync = 1 (inactif)
— vsync = 1 (inactif)
— in_display = 0 (dans la zone blanche)

— DEUXIÈME TEST : TEST POUR LES VALEURS DE Y DANS LA ZONE BLANCHE
— ($V_{PIX} \leq y < V_{PIX} + V_{FPORCH}$)
— BALAYAGE DE y DE 480 A 489
— BALAYAGE DE x DE 0 A 639
— Valeurs des sorties attendues
— hsync = 1 (inactif)
— vsync = 1 (inactif)
— in_display = 0 (dans la zone blanche)
— BALAYAGE DE x DE 640 A 655
— Valeurs des sorties attendues
— hsync = 1 (inactif)
— vsync = 1 (inactif)
— in_display = 0 (dans la zone blanche)
— BALAYAGE DE x DE 752 A 799
— Valeurs des sorties attendues
— hsync = 0 (actif)
— vsync = 1 (inactif)
— in_display = 0 (dans la zone blanche)
— BALAYAGE DE x DE 752 A 799
— Valeurs des sorties attendues
— hsync = 1 (inactif)
— vsync = 0 (actif)
— in_display = 0 (dans la zone blanche)

— TROISIÈME TEST : TEST POUR LES VALEURS DE Y DANS LA ZONE BLANCHE "VSYNC_SIZE"
— ($V_{PIX} + V_{FPORCH} \leq y < V_{PIX} + V_{FPORCH} + V_{SYNC_SIZE}$)
— BALAYAGE DE y DE 490 A 491
— BALAYAGE DE x DE 0 A 639
— Valeurs des sorties attendues
— hsync = 1 (inactif)
— vsync = 0 (actif)
— in_display = 0 (dans la zone blanche)
— BALAYAGE DE x DE 640 A 655
— Valeurs des sorties attendues
— hsync = 1 (inactif)
— vsync = 0 (actif)
— in_display = 0 (dans la zone blanche)
— BALAYAGE DE x DE 752 A 799
— Valeurs des sorties attendues
— hsync = 0 (actif)
— vsync = 0 (actif)
— in_display = 0 (dans la zone blanche)
— BALAYAGE DE x DE 752 A 799
— Valeurs des sorties attendues
— hsync = 1 (actif)
— vsync = 0 (actif)
— in_display = 0 (dans la zone blanche)





- Le module `Gen_mir`
 - PRINCIPE DES TESTS

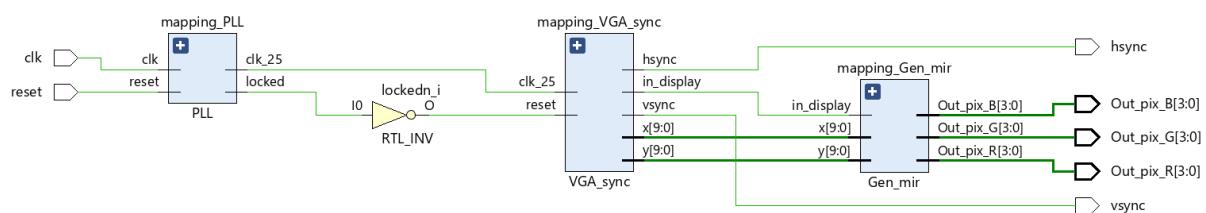
-- Initialisation des signaux d'entrée

in display = 0; x =0; y =0;

in display = 1; x et y = pixel count mod 640 ;



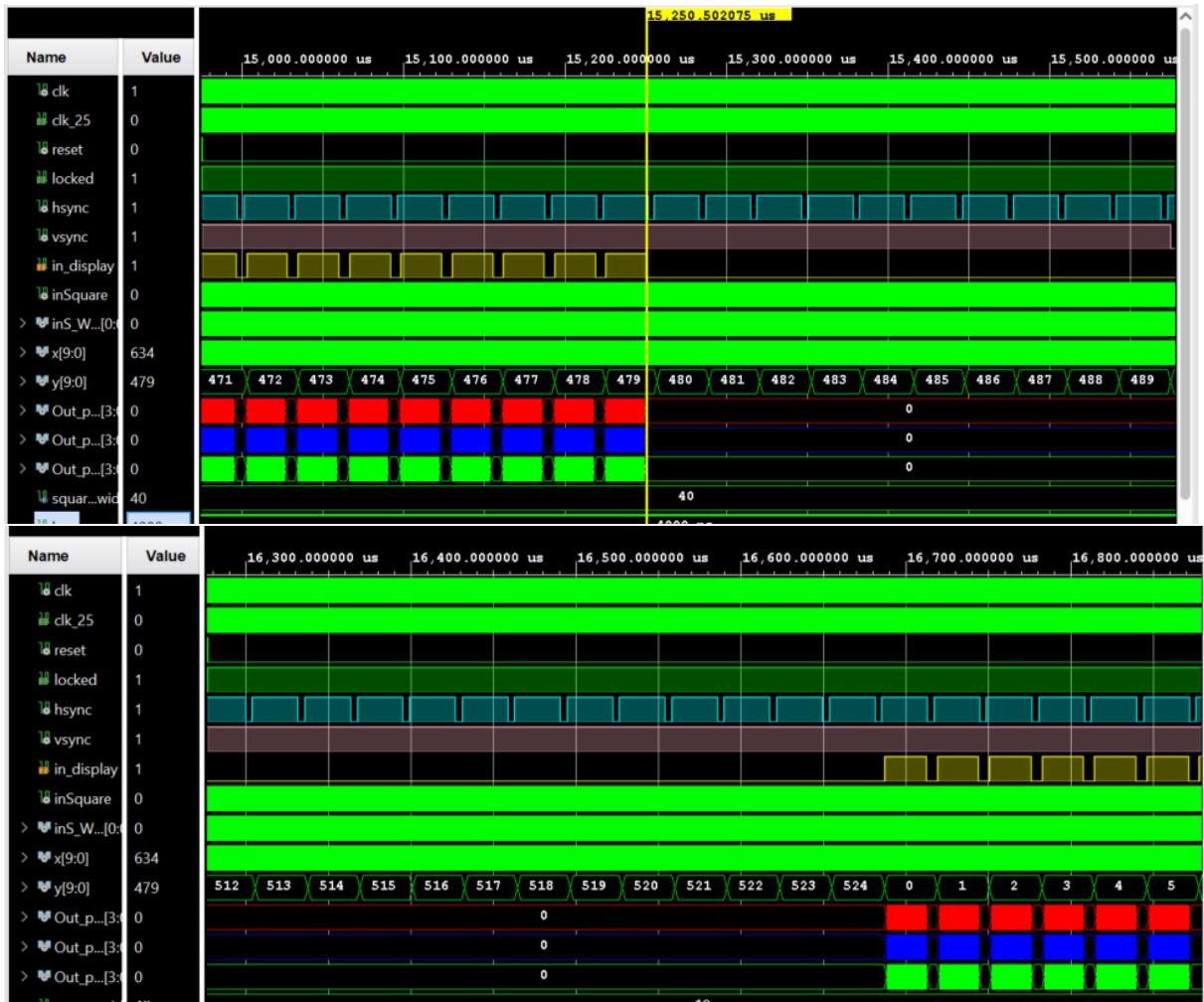
- Le top VGA



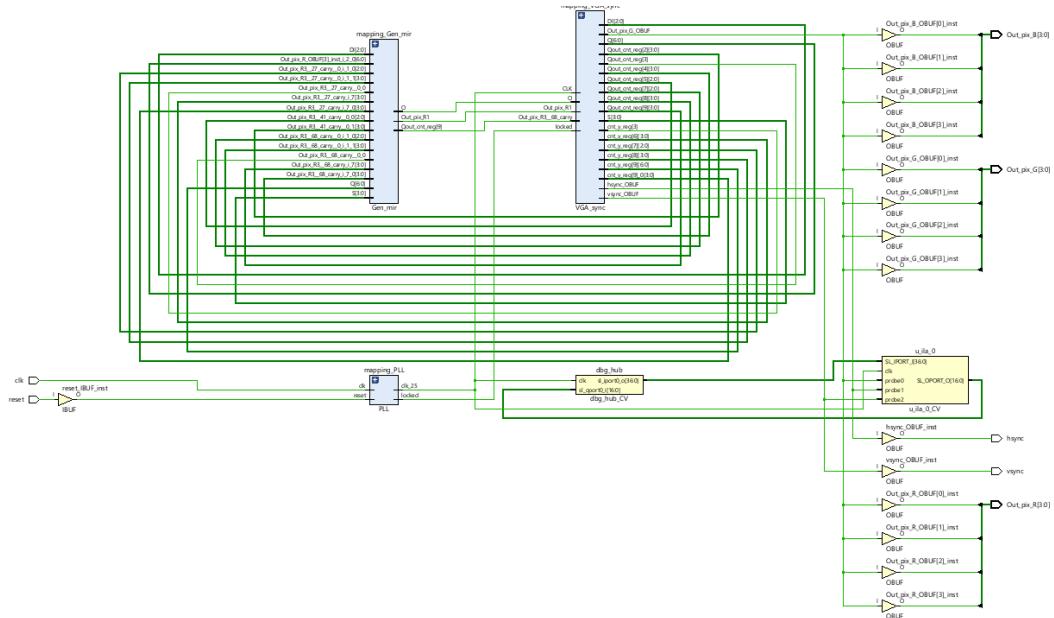
La simulation du schéma donne :

Suivant les valeurs de x et de y reçues, le module `in_display` va déterminer si le pixel est dans l'image visible ou virtuelle





- Synthèse



```

-----+
-----+ Start RTL Component Statistics +-----+
-----+
Detailed RTL Component Info :
+---Adders :
    2 Input   10 Bit      Adders := 2
    2 Input   1 Bit       Adders := 1
+---Registers :
                10 Bit      Registers := 2
+---Muxes :
    2 Input   10 Bit      Muxes := 2
    2 Input   1 Bit       Muxes := 1
-----+
-----+ Finished RTL Component Statistics +-----+
-----+
Report BlackBoxes:
+-----+-----+
| |BlackBox name | Instances |
+-----+-----+
| 1 |PLL           | 1|
+-----+-----+
-----+
Report Cell Usage:
+-----+-----+
| |Cell          | Count |
+-----+-----+
| 1 |PLL_bbox     | 1|
| 2 |CARRY4       | 10|
| 3 |LUT1          | 3|
| 4 |LUT2          | 22|
| 5 |LUT3          | 24|
| 6 |LUT4          | 16|
| 7 |LUT5          | 18|
| 8 |LUT6          | 30|
| 9 |FDCE          | 20|
|10 |IBUF          | 11|
|11 |OBUF          | 14|
+-----+-----+

```

Toutes les ressources utilisées sont répertoriées ci-dessus. Nous avons 2 registres de 10 bits pour x et y (ie 20 registres à 1 bit) ; une boîte pour la PLL.

Nous avons configuré le FPGA avec le fichier de contrainte.

```

5  # PL System Clock
6  set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports clk]
7  create_clock -period 8.000 -name sys_clk_pin -waveform {0.000 4.000} -add [get_ports clk]
8
9
10 # RGB LEDs
11 #set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports out_LED0_B]
12 #set_property -dict {PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports out_LED0_G]
13 #set_property -dict {PACKAGE_PIN N15 IOSTANDARD LVCMOS33} [get_ports out_LED0_R]
14 ##set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33} [get_ports out_LED1_B]
15 ##set_property -dict {PACKAGE_PIN L14 IOSTANDARD LVCMOS33} [get_ports out_LED1_G]
16 #set_property -dict {PACKAGE_PIN M15 IOSTANDARD LVCMOS33} [get_ports out_LED1_R]
17
18 # Buttons
19 set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports reset]
20 #set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports bouton_1]
21
22 ## Pmod Header JA
23 set_property -dict {PACKAGE_PIN Y18 IOSTANDARD LVCMOS33} [get_ports {Out_pix_R[0]}]
24 set_property -dict {PACKAGE_PIN Y19 IOSTANDARD LVCMOS33} [get_ports {Out_pix_R[1]}]
25 set_property -dict {PACKAGE_PIN Y16 IOSTANDARD LVCMOS33} [get_ports {Out_pix_R[2]}]
26 set_property -dict {PACKAGE_PIN Y17 IOSTANDARD LVCMOS33} [get_ports {Out_pix_R[3]}]
27 set_property -dict {PACKAGE_PIN U18 IOSTANDARD LVCMOS33} [get_ports {Out_pix_B[0]}]
28 set_property -dict {PACKAGE_PIN U19 IOSTANDARD LVCMOS33} [get_ports {Out_pix_B[1]}]
29 set_property -dict {PACKAGE_PIN W18 IOSTANDARD LVCMOS33} [get_ports {Out_pix_B[2]}]
30 set_property -dict {PACKAGE_PIN W19 IOSTANDARD LVCMOS33} [get_ports {Out_pix_B[3]}]
31
32 ## Pmod Header JB
33 set_property -dict {PACKAGE_PIN W14 IOSTANDARD LVCMOS33} [get_ports {Out_pix_G[0]}]
34 set_property -dict {PACKAGE_PIN Y14 IOSTANDARD LVCMOS33} [get_ports {Out_pix_G[1]}]
35 set_property -dict {PACKAGE_PIN T11 IOSTANDARD LVCMOS33} [get_ports {Out_pix_G[2]}]
36 set_property -dict {PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports {Out_pix_G[3]}]
37 set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports hsync]
38 set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33} [get_ports vsync]
39 #set_property -dict {PACKAGE_PIN V12 IOSTANDARD LVCMOS33} [get_ports {Out_pix_G[2]}]

```

La fréquence est réglée à 125MHz, celle fournit par la carte coraZ7.

-Implémentation : étude du rapport de timing.

L'horloge : clk

Clock	Waveform(ns)	Period(ns)	Frequency (MHz)
<hr/>			
clk	{0.000 4.000}	8.000	125.000
clk_25_PLL	{0.000 19.862}	39.724	25.174
clkfbout_PLL	{0.000 16.000}	32.000	31.250
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCP	{0.000 16.500}	33.000	30.303
sys_clk_pin	{0.000 4.000}	8.000	125.000
clk_25_PLL_1	{0.000 19.862}	39.724	25.174
clkfbout_PLL_1	{0.000 16.000}	32.000	31.250

Vérification des violations de set up et du hold.

```

4 -----
5 | Design Timing Summary
6 |
7 -----
8
9   WNS(ns)      TNS(ns)    TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)    THS Failing Endpoints  THS Total Endpoints      WFWS(ns)      TPWS(ns)
0   -----
1   26.579       0.000          0                  3639        0.018       0.000          0                  3623        2.000       0.000
2
3
4 All user specified timing constraints are met.

```

Les valeurs dans le THS et TNS sont à 0. Il n'y a pas de violations du set up et du hold, donc pas de métastabilité.

Le chemin critique donne :

```

Max Delay Paths
-----
Slack (MET) : 29.726ns (required time - arrival time)
  Source: <hidden> (rising edge-triggered cell FDRE clocked by clk_25_PLL {rise@0.000ns fall@19.862ns period=39.724ns})
  Destination: <hidden> (rising edge-triggered cell FDRE clocked by clk_25_PLL {rise@0.000ns fall@19.862ns period=39.724ns})
Max Delay Paths
-----
Slack (MET) : 29.726ns (required time - arrival time)
  Source: <hidden> (rising edge-triggered cell FDRE clocked by clk_25_PLL {rise@0.000ns fall@19.862ns period=39.724ns})
  Destination: <hidden> (rising edge-triggered cell FDRE clocked by clk_25_PLL {rise@0.000ns fall@19.862ns period=39.724ns})

```

- Test sur le matériel : Nous avons réalisé le bitstream et testé le système sur le matériel réel. L'affichage de l'échiquier et l'application du filtre fonctionnent correctement sur l'écran VGA.

Nous utilisons l'oscilloscope pour mesurer les signaux de synchronisation, les signaux de couleur (r, g, b) et vérifions leur conformité aux spécifications VGA.

3) Démonstration

Dans cette partie, nous documentons les résultats : compilation des résultats des tests, y compris les captures d'écran ou les enregistrements vidéo de l'affichage de l'image, de l'oscilloscope.

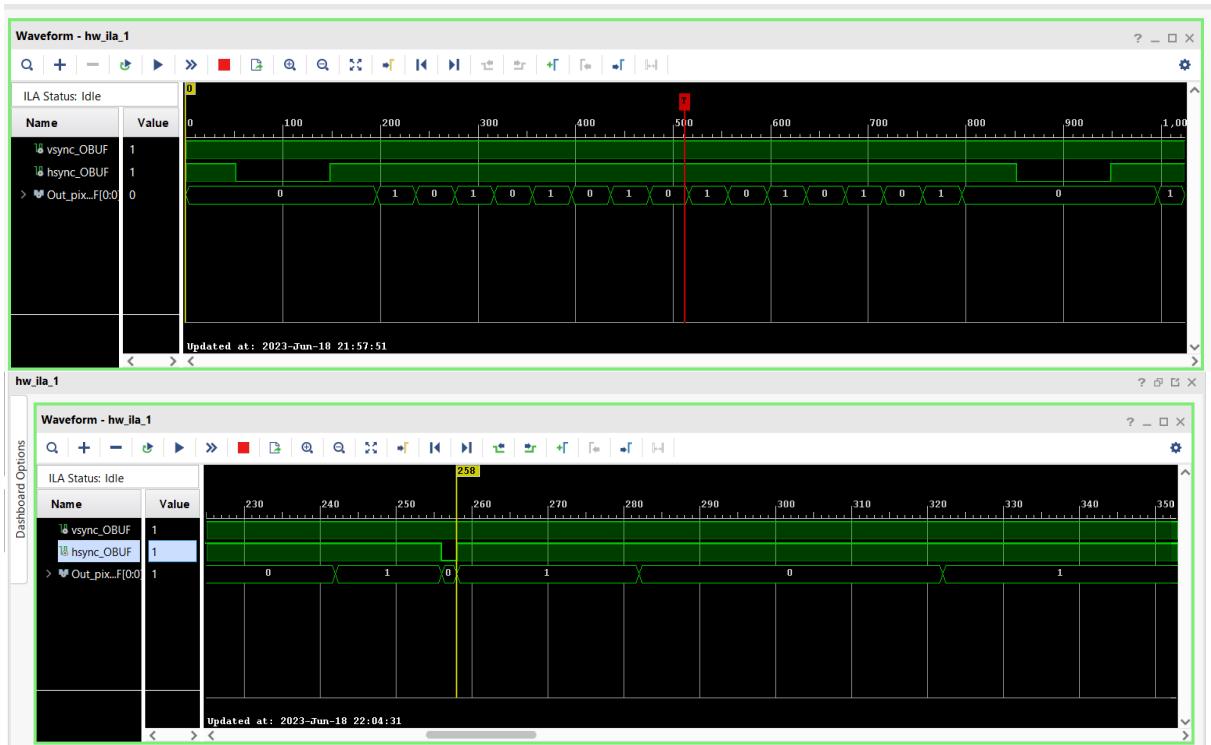
Affichage de L'ILA

Au début du test :

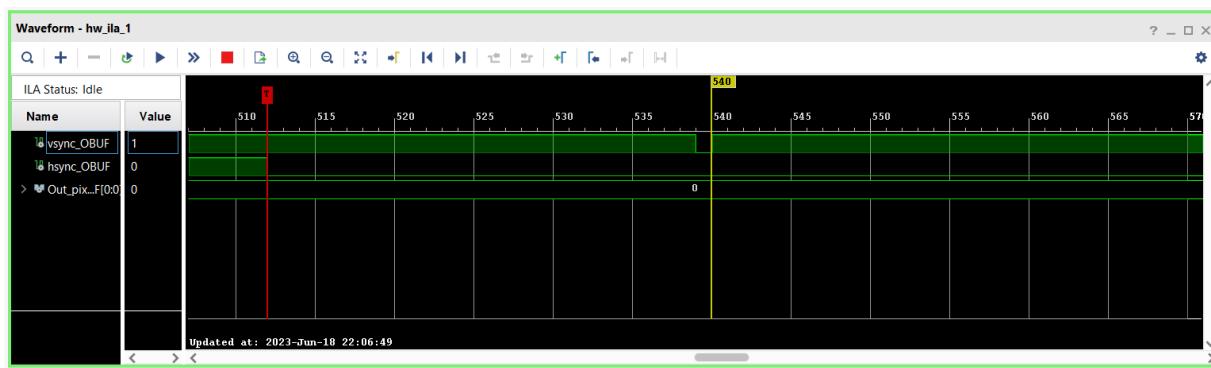
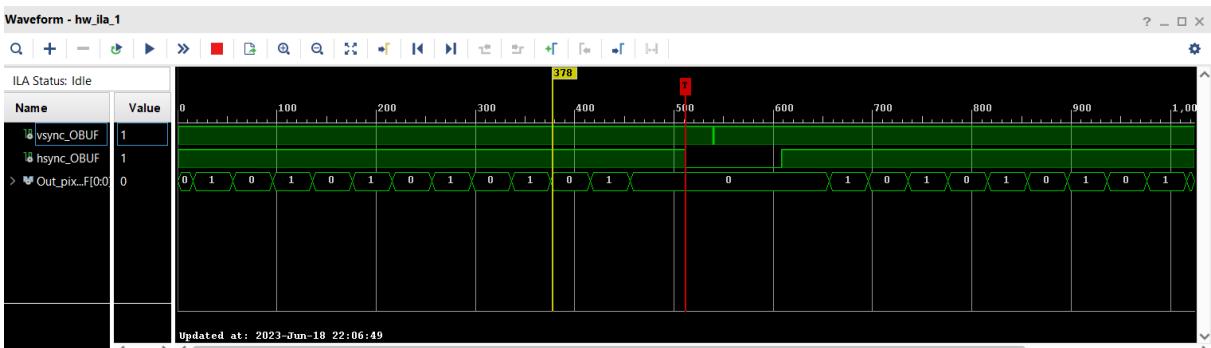
vsync=0 et hsync=0 ou 1.



Le premier envoi de l'Ila nous montre que dans la zone visible de l'image, le signal de sortie est bien blanc ou noir. vsync=1 et hsync=1. Nous n'avons pas d'image lorsque vsync=1 et hsync=0.

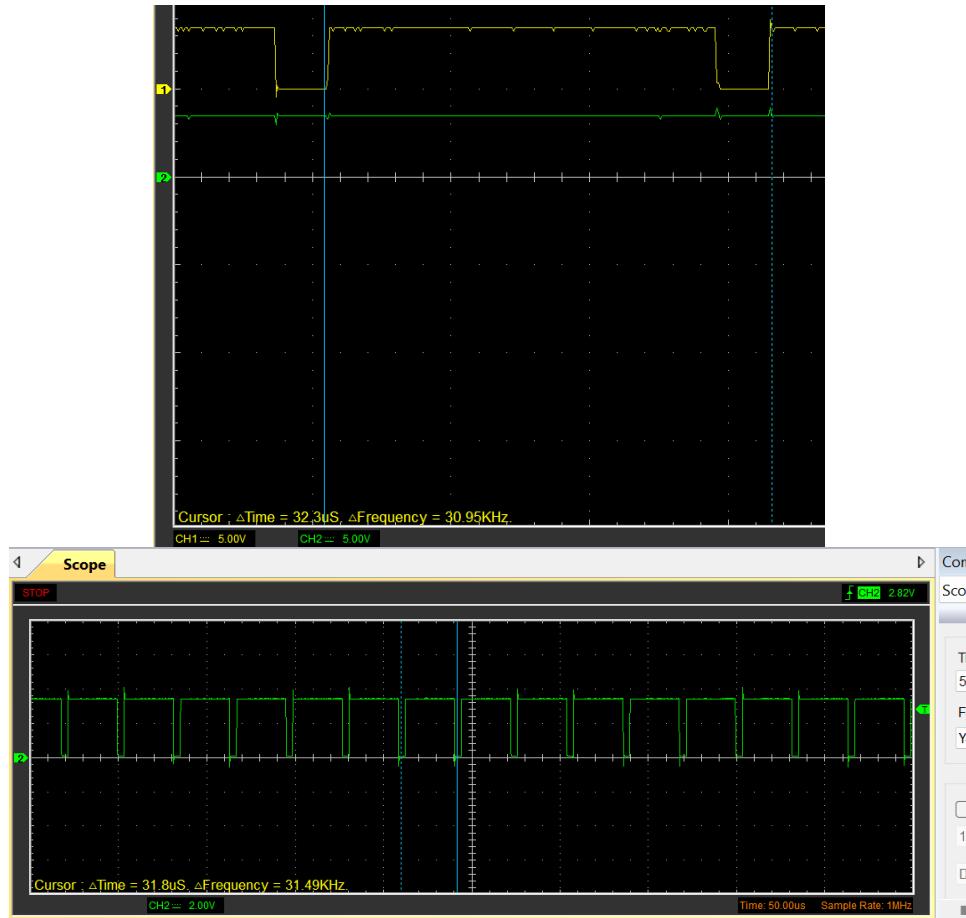


Ici, nous voyons que lorsque vsync et hsync sont à 0, in_display est à 0. Nous sommes donc dans la zone non visible de l'image. Et lorsque nous repassons dans la zone visible, le signal de sortie est bien blanc ou noir.

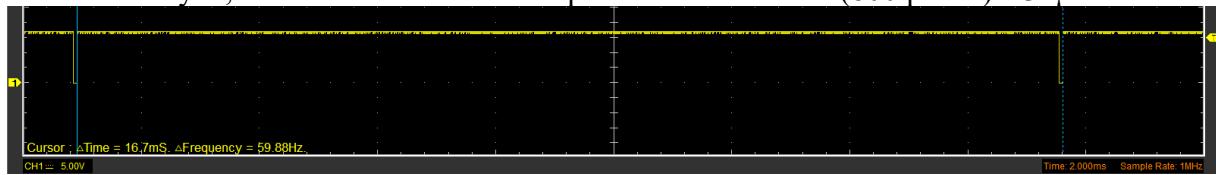


- Utilisons l'oscilloscope pour mesurer les signaux de synchronisation et vérifions leur conformité aux spécifications VGA.

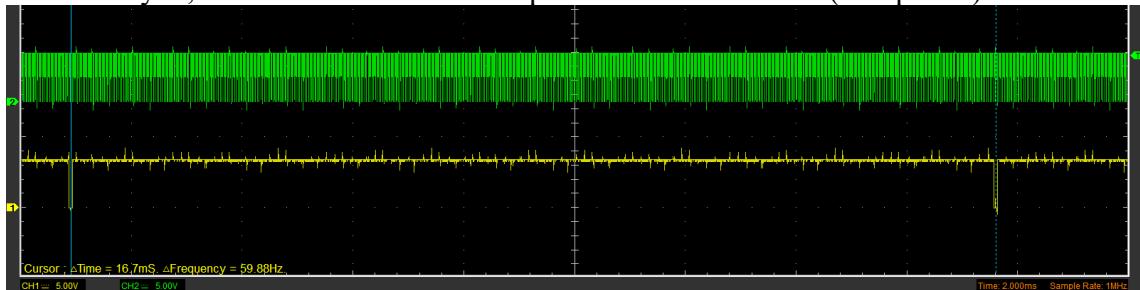
Nous vérifions que les signaux de synchronisation (hsync et vsync) sont générés correctement et respectent les timings VGA.



Mesure de hsync, nous trouvons bien la fréquence du whole line (800 pixels) à 32μs.



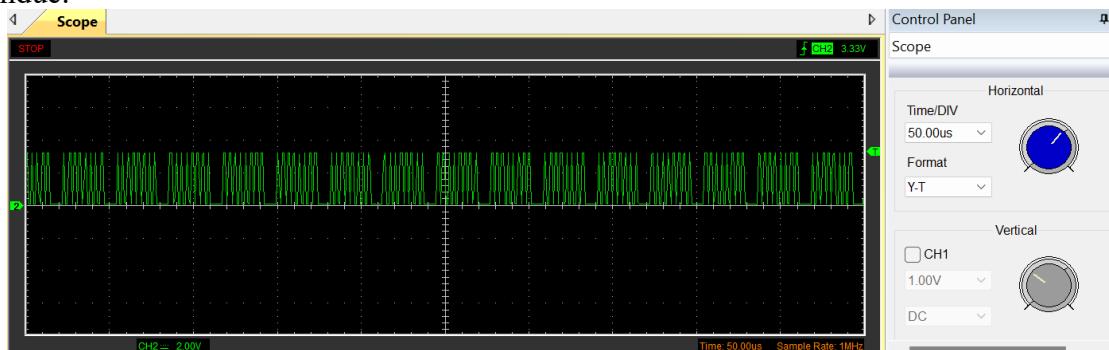
Mesure de vsync, nous trouvons bien la fréquence du whole frame (525 pixels) à 16ms.



Lorsque nous superposons hsync et vsync, nous obtenons les 60Hz correspondant au screen refresh rate.

Nous pouvons donc conclure que les signaux hsync et vsync sont conformes à la norme VGA.

- La mesure des signaux de couleur (r, g, b) nous donne un niveau de tension de 3,3V, celle attendue.



Nous validons ainsi le fonctionnement de notre système.

En connectant le VGA, nous constatons l'affichage du damier sur l'écran de télévision. L'image est stable et ne présente pas de problèmes d'affichage ou de clignotement.

• Rappel du matériel à disposition

Carte Xilinx CoraZ7 avec un câble USB pour connexion à l'ordinateur	Carte Pmod VGA (Digilent)	Câble VGA mâle-mâle
 câble de connexion		
câble de connexion oxilloscope	Téléviseur avec VGA	Ordinateur

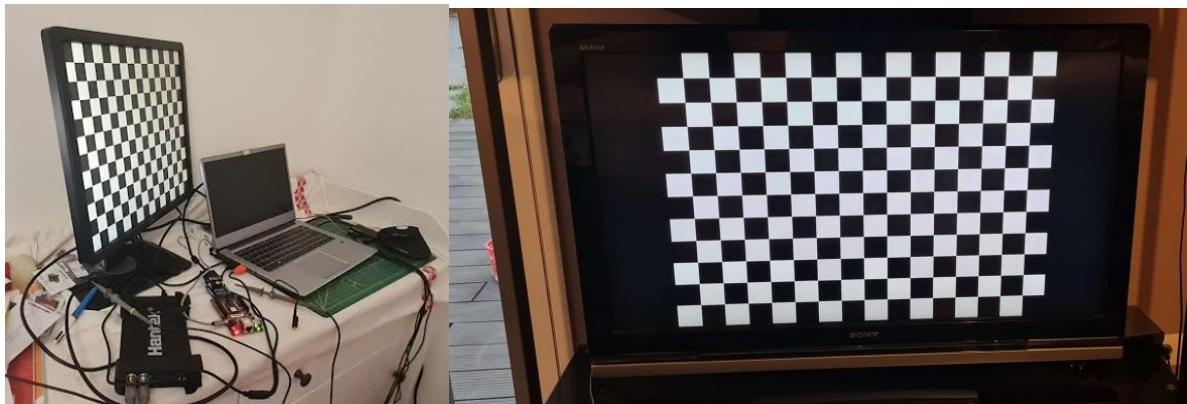
TAB. 1 : Matériels à disposition

La connectique de l'ensemble pour la mesure :



Nous pouvons voir la carte coraZ7 connectée au pmodVGA et au VGA.

Résultat obtenu en mettant en œuvre le système dans les conditions de fonctionnement finales (Vidéo du système en fonctionnement, autre acquisition vidéo).

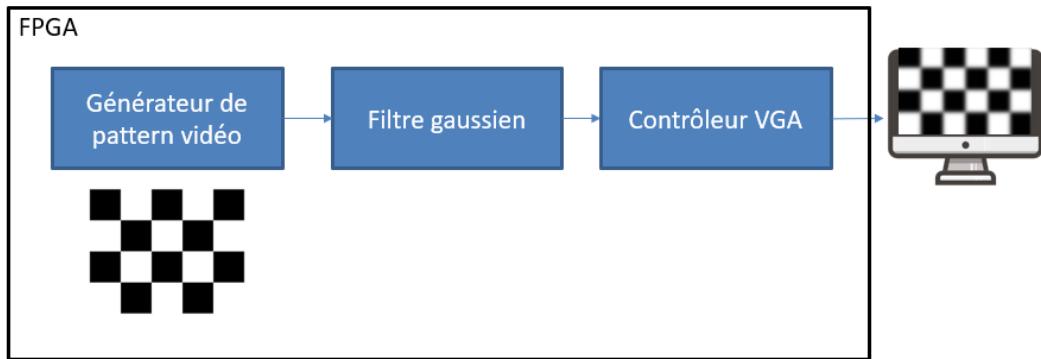


Affichage du damier

III. Étude de la phase finale : affichage d'un damier filtré

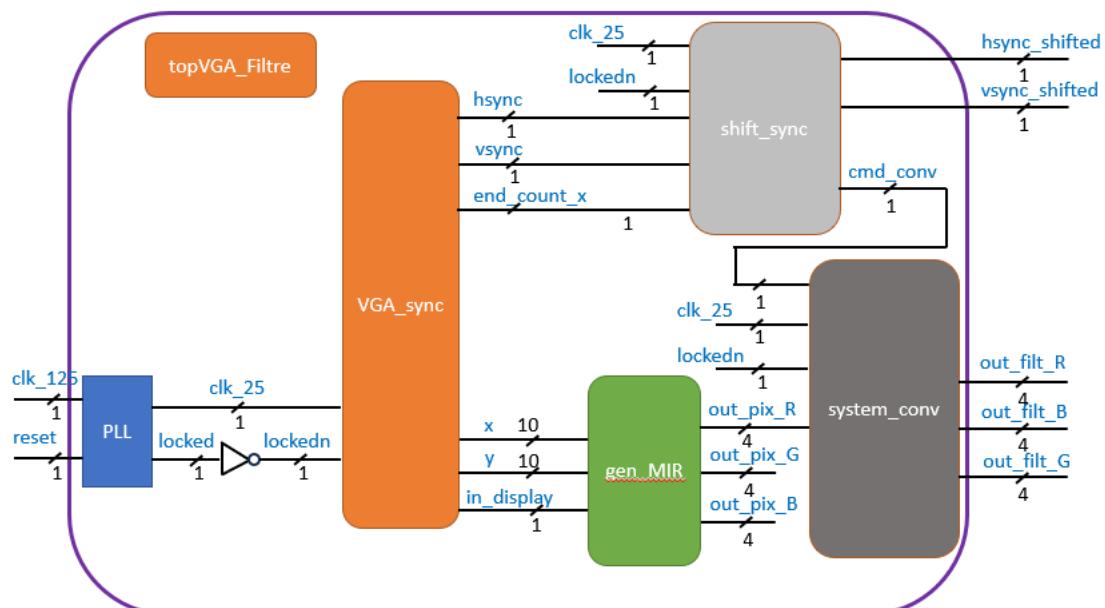
Dans cette phase nous allons chercher à afficher sur un écran un damier après avoir appliqué un flou gaussien. Pour cela, nous allons introduire le filtre de Gauss.

Un filtre gaussien en VHDL est un système qui effectue une convolution de l'image d'entrée avec un noyau de filtrage gaussien. Il est couramment utilisé en traitement d'images pour lisser les contours et réduire le bruit.



Pour réaliser cela, nous créons deux modules : Un module `shift_sync` et un système de convolution, qui vont s'intégrer dans notre architecture.

Pour réaliser le système de convolution, nous utilisons des registres et des composants FIFO. Ainsi, la première donnée de sortie du système de convolution sera décalée dans le temps de 802 périodes. D'où l'intérêt du module `shift_sync` qui va générer les signaux de synchronisation en phase avec les données de sorties.



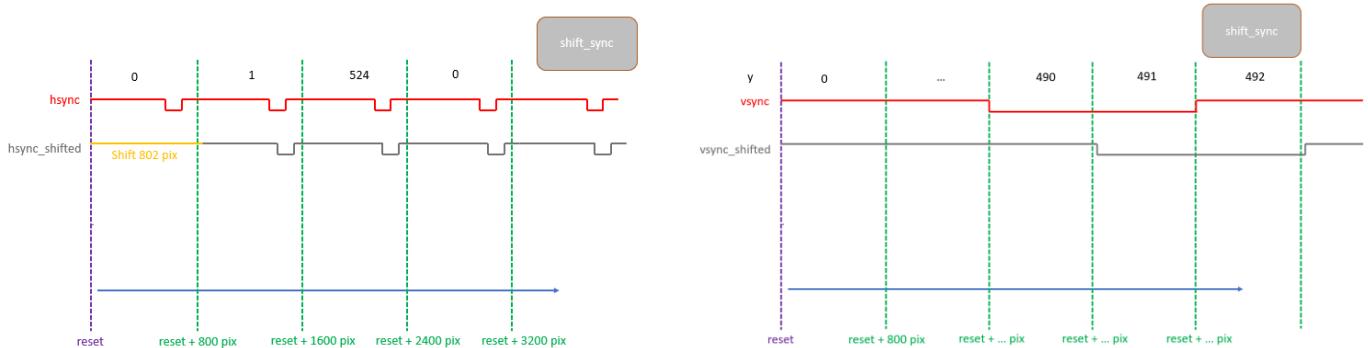
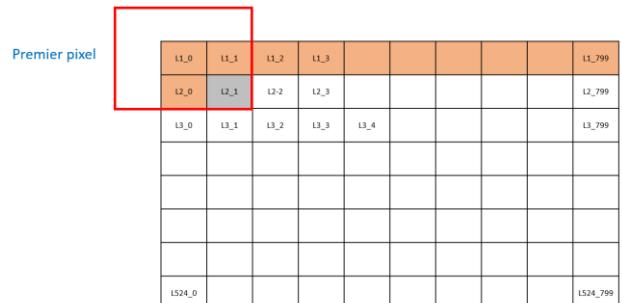
1) Analyse

- Le module PLL
- Le module VGA_sync
- Le module Gen_mir

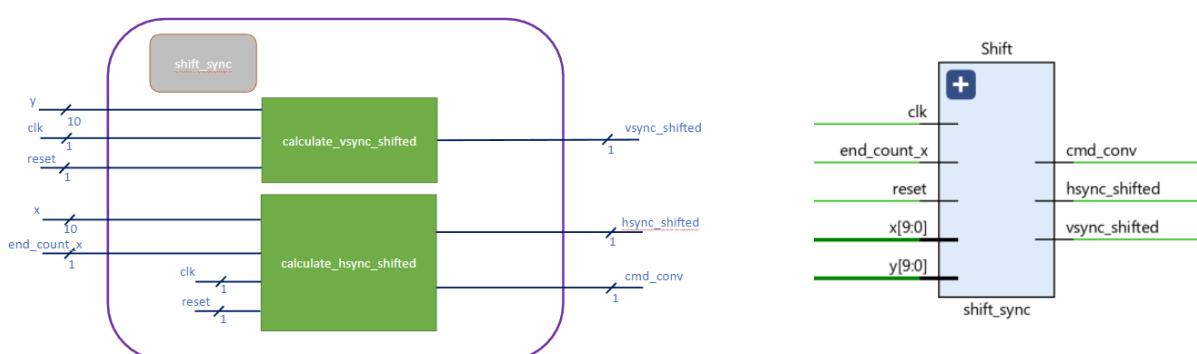
Les trois premiers modules sont décrits dans la phase intermédiaire.

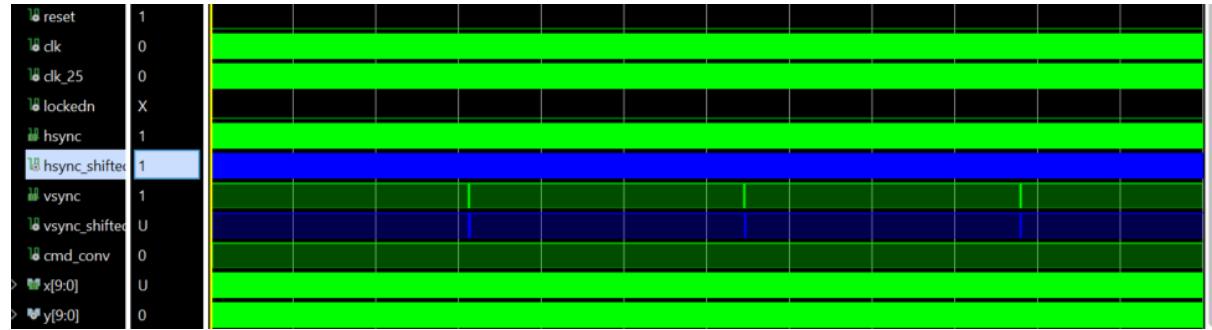
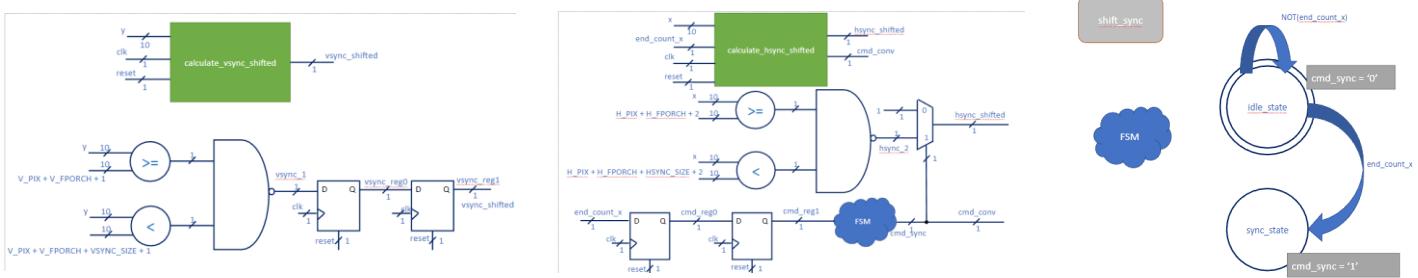
- Le module Shift_sync

Lorsque nous balayons l'image, le premier pixel L1_0 est au cœur du noyau (kernel en anglais) lorsqu'une ligne et deux pixels ont été balayés. Seulement à ce stade, nous pourrons calculer une première sortie de la convolution. D'où la nécessité de décaler les signaux de synchronisation de **802 pixels**.

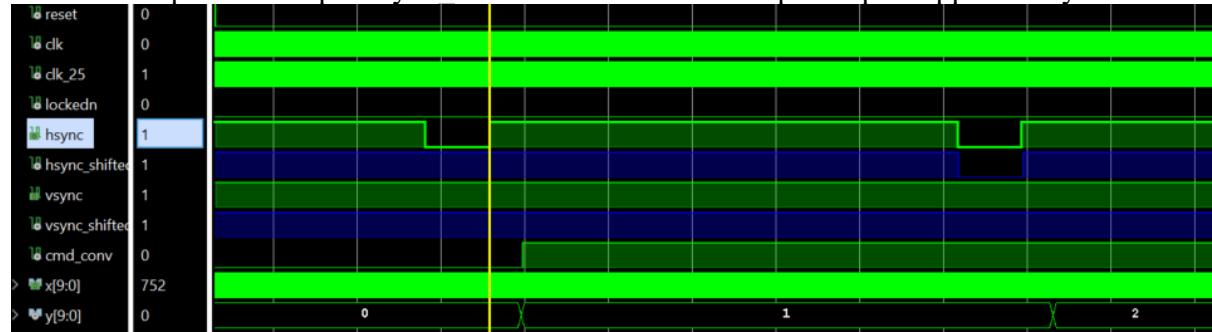


Afin de réaliser ce décalage, nous avons utilisé l'architecture ci-dessous :

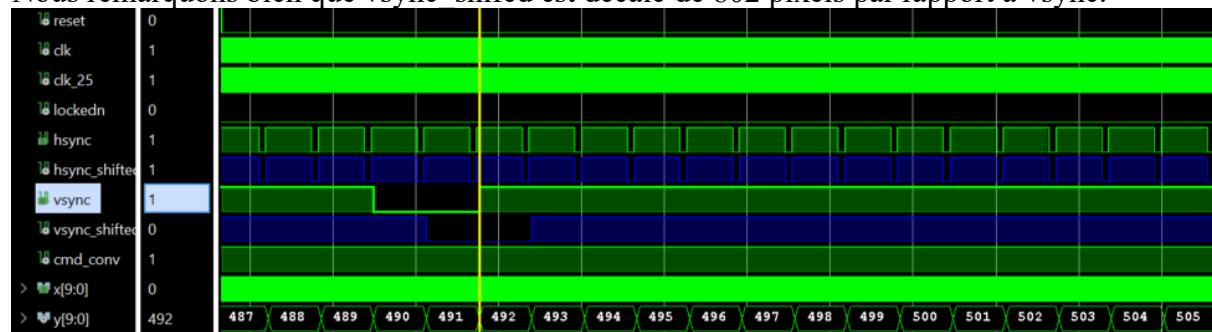




Nous remarquons bien que hsync shifted est décalé de 802 pixels par rapport à hsync.



Nous remarquons bien que vsync shifted est décalé de 802 pixels par rapport à vsync.

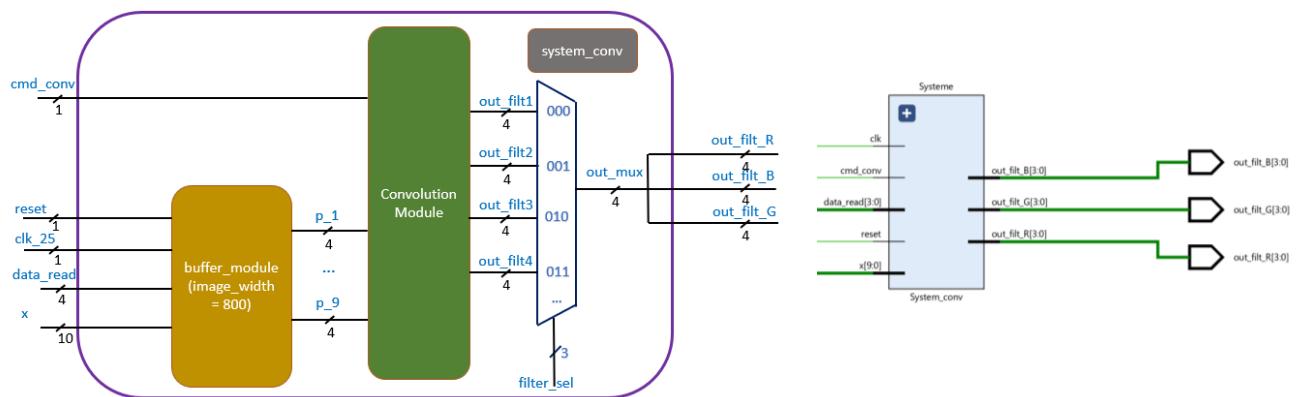


- **Le module System_conv**

Ce module permet d'effectuer la convolution et le padding de l'image. Le système se compose de trois sous-modules : le sélecteur de filtre, le buffer module et la convolution.

Pour rappel, le padding de l'image : processus qui ajoute un padding (généralement des zéros) autour de l'image d'entrée. Le padding est nécessaire pour permettre la convolution du noyau gaussien sur les pixels des bords de l'image. Dans notre architecture, le padding se fera automatiquement car la zone virtuelle de l'image est composée de pixels de couleur noire (intensité nulle). Détails en annexe.

Convolution de l'image : processus pour effectuer la convolution de l'image avec le noyau gaussien. Cela implique de parcourir tous les pixels de l'image, d'appliquer le noyau gaussien à chaque pixel et de calculer la valeur résultante.



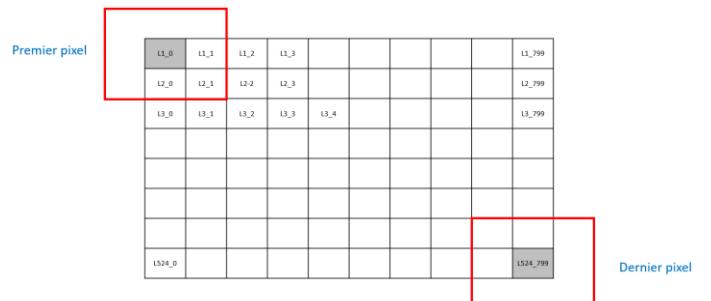
➤ Buffer module

L'objectif du buffer_module est d'extraire de l'image les pixels P1 à P9 que nous allons par la suite filtrer. Pour cela deux approches sont possibles :

- Déplacer le noyau dans l'image,
- Déplacer l'image dans le noyau.

Nous avons opté pour la seconde solution qui ne requiert pas de charger toute l'image en mémoire mais seulement une partie. Ainsi, l'extraction des pixels peut se faire dans le même temps que nous balayons l'image.

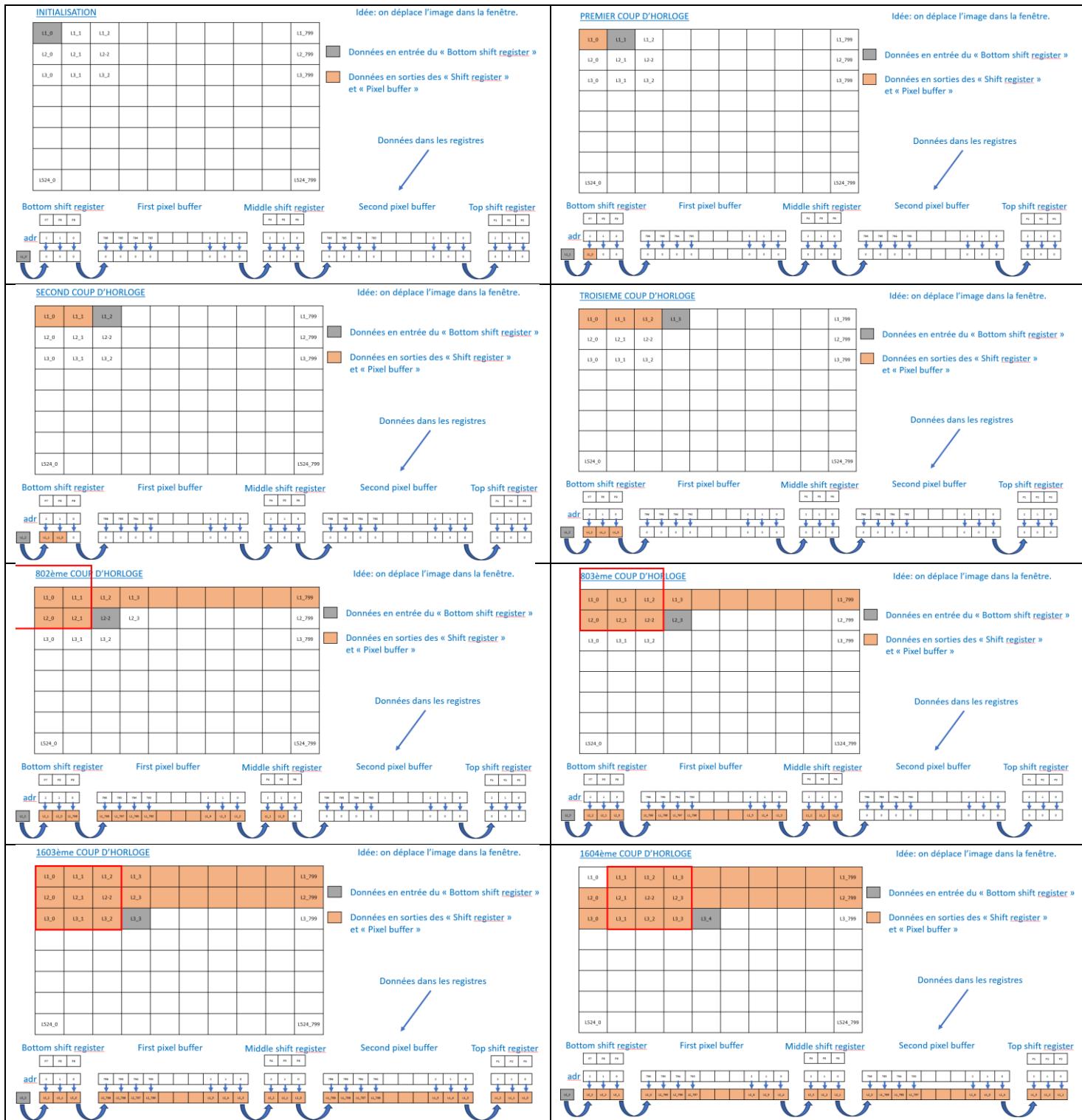
L1_0	L1_1	L1_2	L1_3						L1_799
L2_0	L2_1	L2_2	L2_3						L2_799
L3_0	L3_1	L3_2	L3_3	L3_4					L3_799
				P1	P2	P3			
				P4	P5	P6			
				P7	P8	P9			
									L524_0



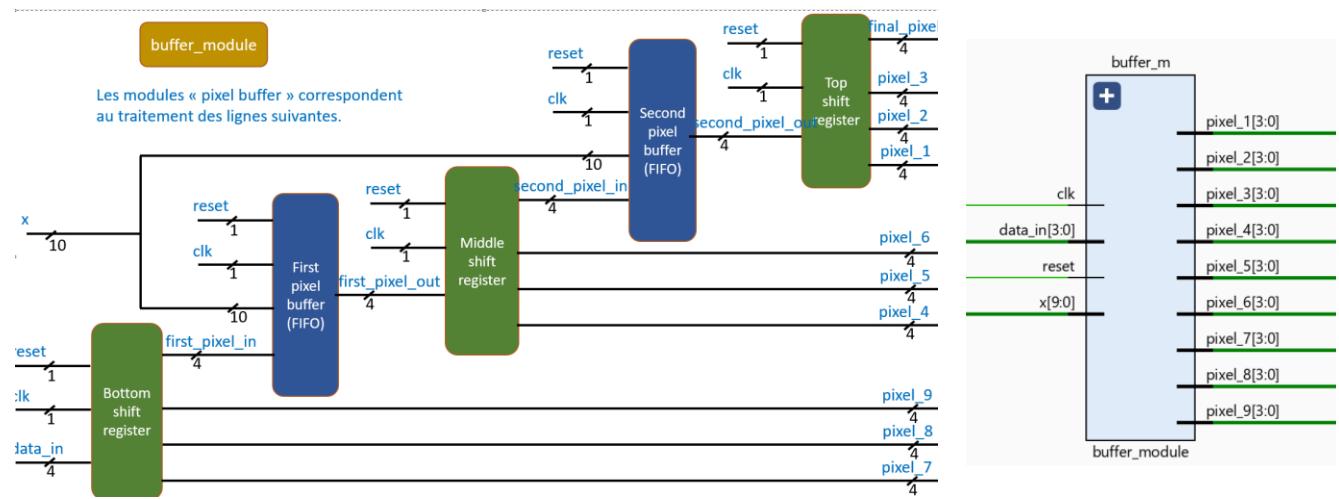
Le principe est le suivant. A chaque coup d'horloge, les pixels que nous balayons sont écrits dans les mémoires de registres à décalage (shift_register) et de FIFO (pixel_buffer).

Les registres à décalage ont une capacité de 3 pixels tandis que nous utilisons les FIFOs avec 797 pixels. A noter, $797 + 3 = 800$. Ce qui correspond au nombre de pixels d'une ligne de l'image.

Au cours du balayage, nous obtenons ainsi le remplissage ci-dessous :

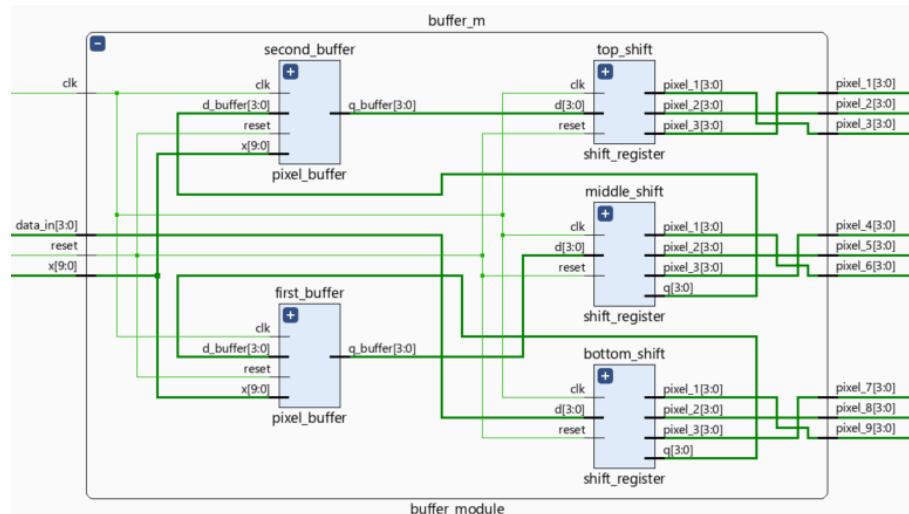


L'architecture mise en place est la suivante :



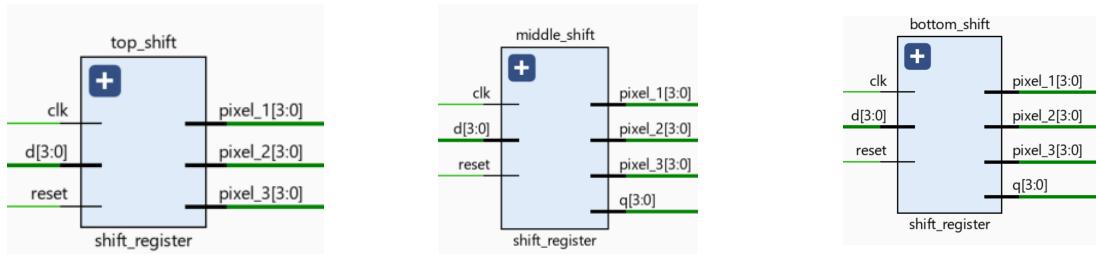
Chaque registre à décalage alimente trois valeurs de pixels vers le module de convolution.

Architecture interne.

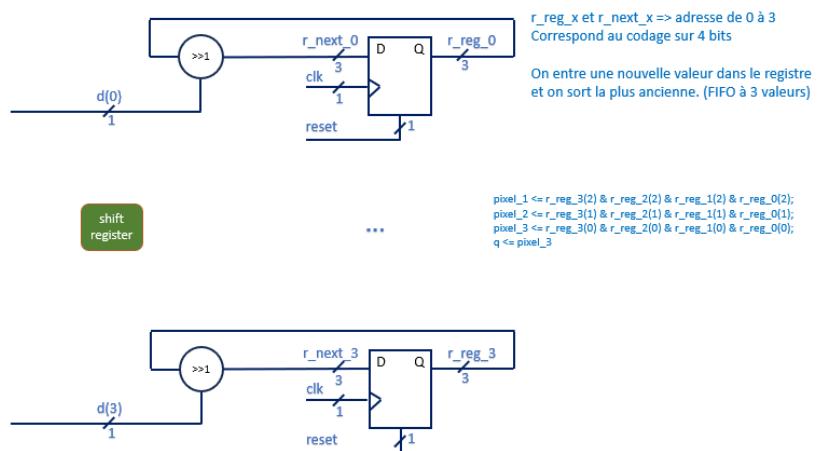


- Shift register

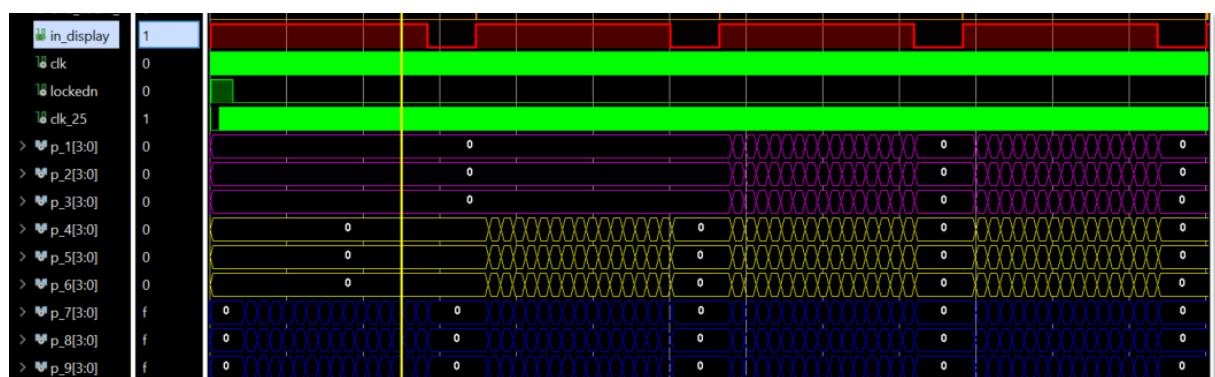
Le registre à décalage permet au noyau du filtre de lire plusieurs fois la même valeur de pixel. Une nouvelle valeur de pixel est décalée dans le registre à décalage inférieur à chaque cycle d'horloge. Lorsqu'une valeur de pixel a traversé l'ensemble du registre à décalage (3 valeurs), elle est écrite dans le pixel buffer. Ce processus est répété pour les registres à décalage du milieu et du haut.



Son architecture interne est la suivante :



Nous obtenons les résultats de simulation ci-dessous :

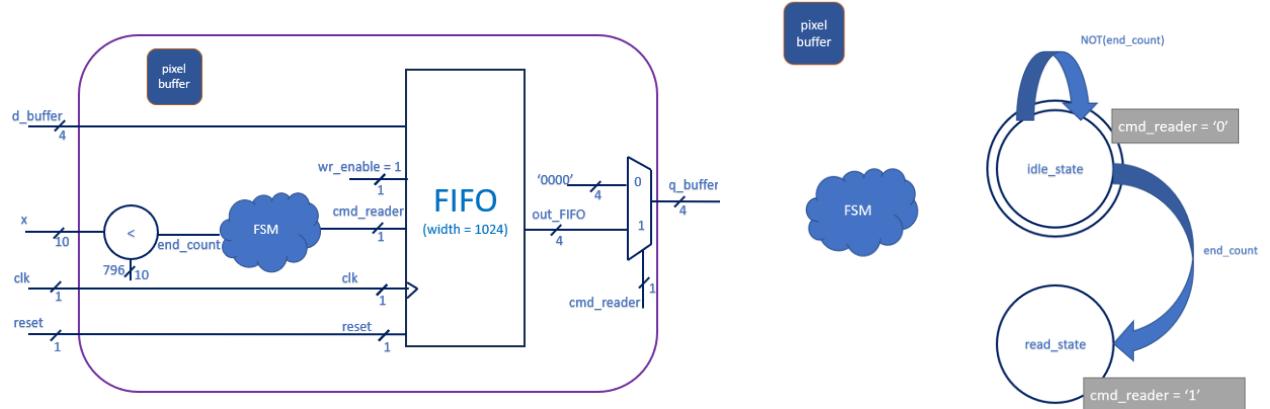


- Pixel buffer

Une nouvelle valeur de pixel est lue à chaque cycle d'horloge, elle entre dans le premier registre à décalage (ligne inférieure du noyau du filtre). Après m cycles d'horloge, cette valeur de pixel est décalée dans le pixel buffer. Le pixel buffer a une largeur de $W - m$ cellules. Dans notre cas, la largeur de l'image est de 800 pixels ($W = 800$) et la largeur du noyau du filtre est de 3 pixels ($m = 3$), soit une largeur de 797 pixels. Lorsque le pixel a traversé le pixel buffer, il passe au registre à décalage suivant, ce processus est répété jusqu'à ce que le noyau du filtre soit rempli de valeurs de pixels.

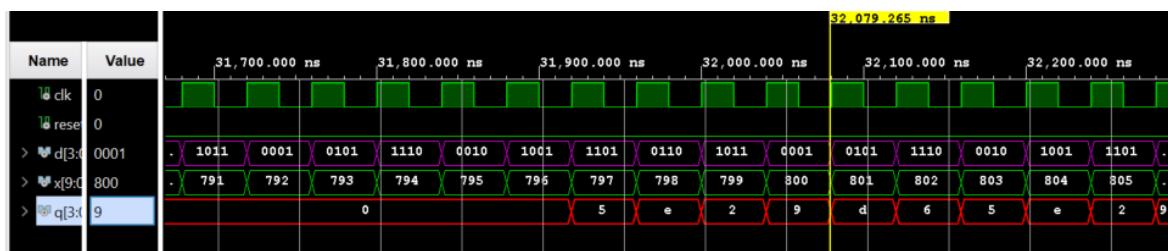


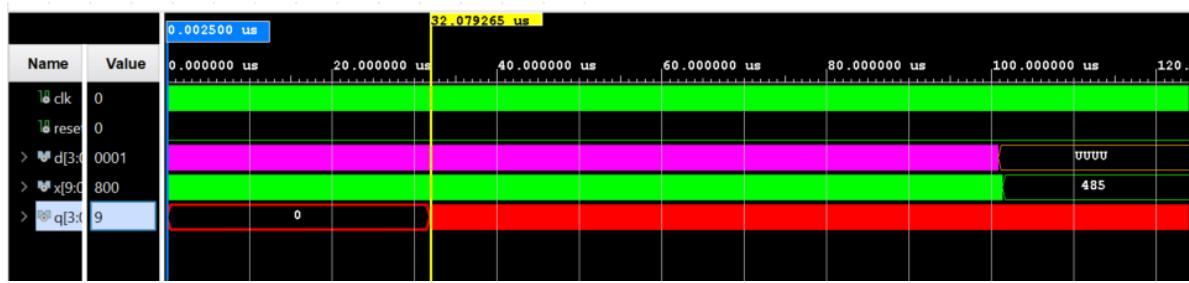
Son architecture interne donne :



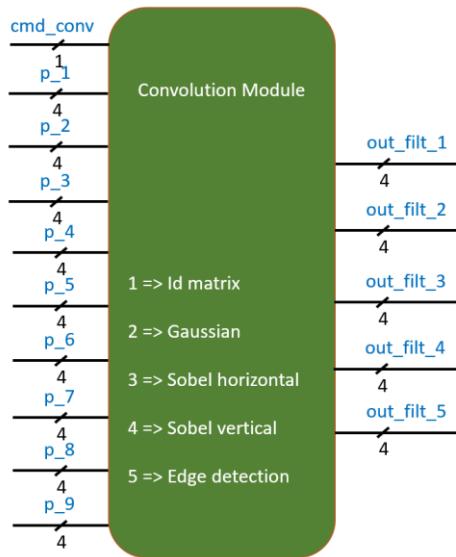
Initialement, la FIFO étant vide, il nous faut la remplir à hauteur de 797 valeurs, comme expliqué précédemment. Le signal d'écriture dans la FIFO est maintenu en permanence à 1 tandis que le signal de lecture dans la FIFO ne commence qu'après 797 coups d'horloge. C'est l'objet de l'entrée x et de la machine à états. A partir de ce moment-là, à chaque coup d'horloge, une nouvelle valeur de pixel est écrite dans la FIFO et la valeur la plus ancienne est lue.

La simulation ci-dessous illustre bien l'apparition des valeurs de pixel à partir de 797 coups d'horloge ($x = 797$).





➤ Convolution module

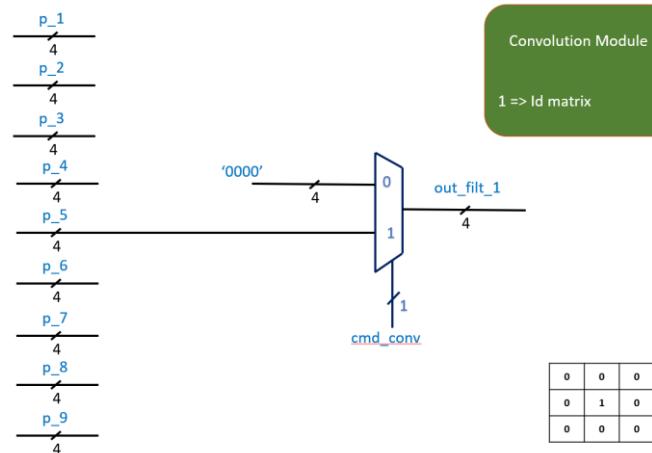


A travers notre étude, nous avons implémenté 5 filtres comme décrits ci-contre.

- Matrice identité (Id matrix)

Nous capturons simplement la valeur du pixel au cœur du noyau. Ceci nous permet de reproduire l'image sans filtre.

Le signal de commande cmd_conv nous sert uniquement pour envoyer les données après 802 coups d'horloge.

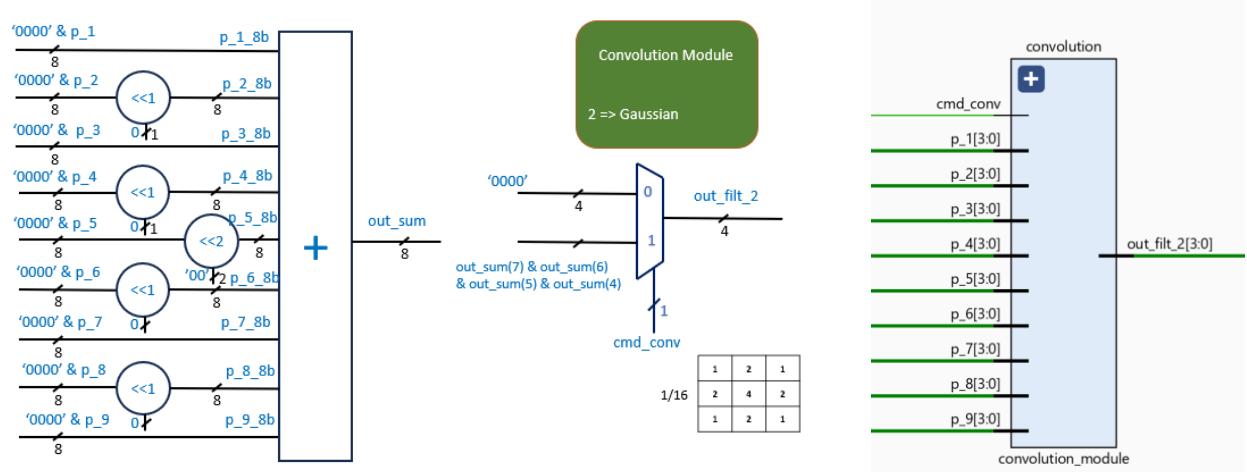


- Filtre gaussien

Le module de convolution a été conçu avec des signaux de décalage à gauche pour effectuer les multiplications. Un décalage d'un bit à gauche nous permet de réaliser une multiplication par 2. Un décalage de 2 bits à gauche nous permet de réaliser une multiplication par 4. Etc...

La formule « out_sum : out_sum(7) & out_sum(6) & out_sum(5) & out_sum(4) » nous permet d'effectuer la division par 16.

En procédant de la sorte, nous obtenons l'architecture ci-dessous :



- Autres filtres

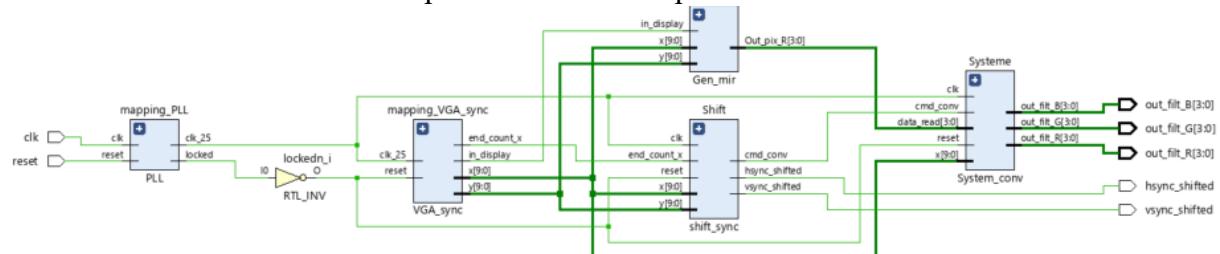
Les architectures des autres filtres sont disponibles en annexe.

➤ Filtre selector

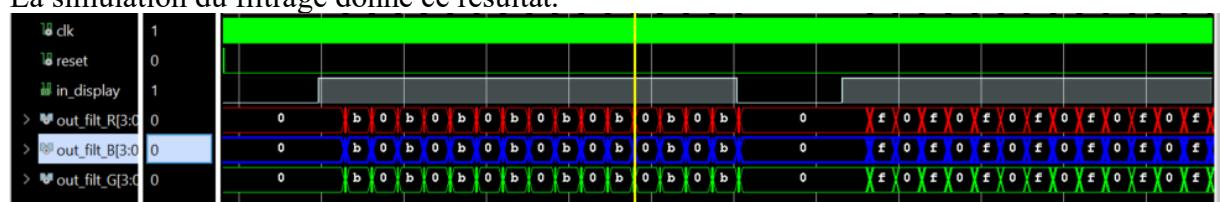
Ce dernier permet la sélection du filtre que nous souhaitons utiliser. C'est un simple multiplexeur.

2) Test

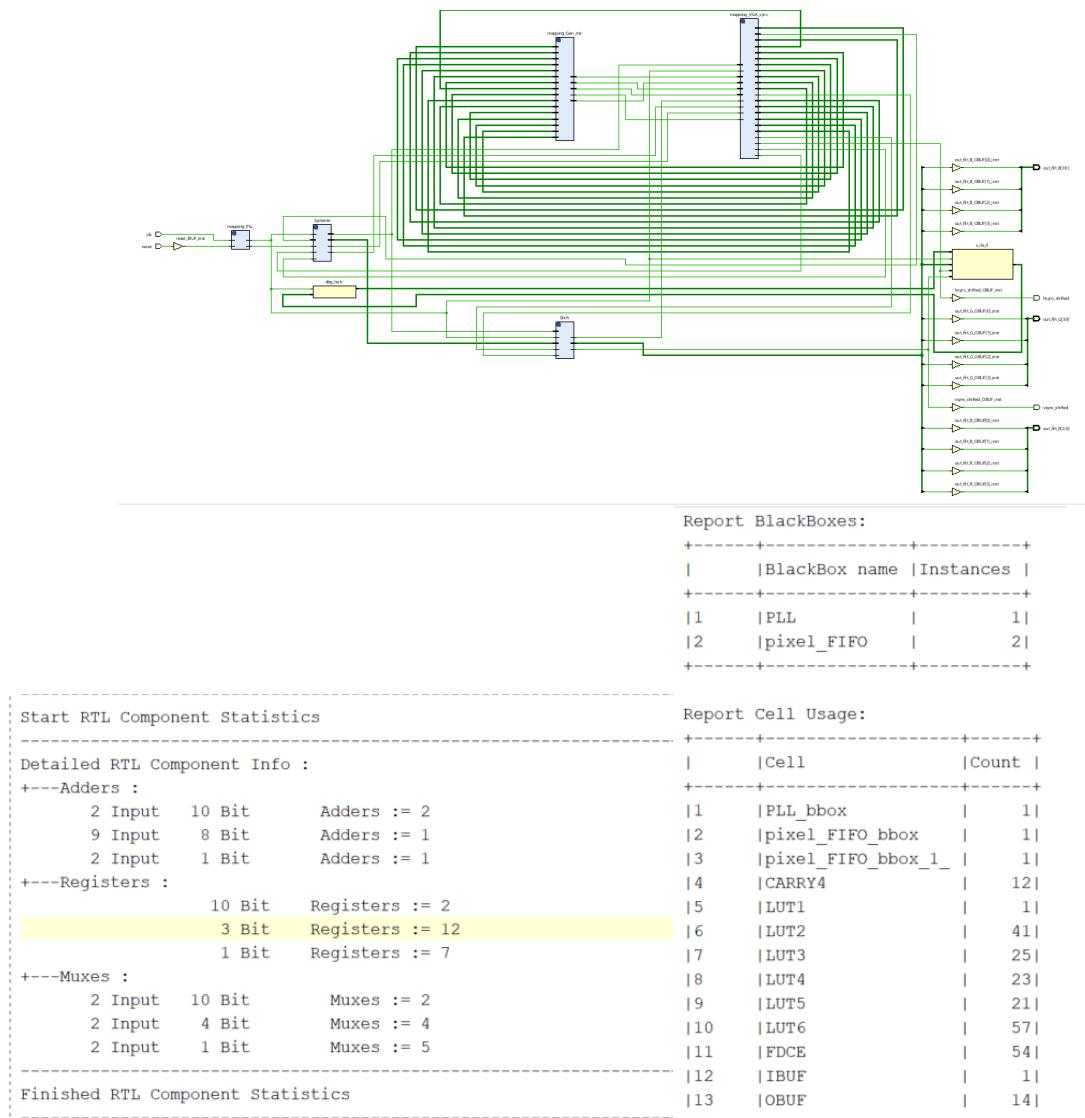
Les tests effectués sont pour s'assurer du bon fonctionnement de chaque module et puis de l'ensemble. Les résultats et interprétations sont compilés dans la suite du document.



La simulation du filtrage donne ce résultat.



- Synthèse



Configuration du FPGA avec le fichier de contrainte.

```

6  # PL System Clock
7  set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports clk]
8  create_clock -period 8.000 -name sys_clk_pin -waveform {0.000 4.000} -add [get_ports clk]
9
10 # RGB LEDs
11 #set_property -dict {PACKAGE_PIN L15 IOSTANDARD LVCMOS33} [get_ports out_LED0_B]
12 #set_property -dict {PACKAGE_PIN G17 IOSTANDARD LVCMOS33} [get_ports out_LED0_G]
13 #set_property -dict {PACKAGE_PIN N15 IOSTANDARD LVCMOS33} [get_ports out_LED0_R]
14 ##set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33} [get_ports out_LED1_B]
15 ##set_property -dict {PACKAGE_PIN L14 IOSTANDARD LVCMOS33} [get_ports out_LED1_G]
16 #set_property -dict {PACKAGE_PIN M15 IOSTANDARD LVCMOS33} [get_ports out_LED1_R]
17
18 # Buttons
19 set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports reset]
20 #set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports bouton_1]
21
22 ## Pmod Header JA
23 set_property -dict {PACKAGE_PIN Y18 IOSTANDARD LVCMOS33} [get_ports {out_filt_R[0]}]
24 set_property -dict {PACKAGE_PIN Y19 IOSTANDARD LVCMOS33} [get_ports {out_filt_R[1]}]
25 set_property -dict {PACKAGE_PIN Y16 IOSTANDARD LVCMOS33} [get_ports {out_filt_R[2]}]
26 set_property -dict {PACKAGE_PIN Y17 IOSTANDARD LVCMOS33} [get_ports {out_filt_R[3]}]
27 set_property -dict {PACKAGE_PIN U18 IOSTANDARD LVCMOS33} [get_ports {out_filt_B[0]}]
28 set_property -dict {PACKAGE_PIN U19 IOSTANDARD LVCMOS33} [get_ports {out_filt_B[1]}]
29 set_property -dict {PACKAGE_PIN W18 IOSTANDARD LVCMOS33} [get_ports {out_filt_B[2]}]
30 set_property -dict {PACKAGE_PIN W19 IOSTANDARD LVCMOS33} [get_ports {out_filt_B[3]}]
31
32 ## Pmod Header JB
33 set_property -dict {PACKAGE_PIN W14 IOSTANDARD LVCMOS33} [get_ports {out_filt_G[0]}]
34 set_property -dict {PACKAGE_PIN Y14 IOSTANDARD LVCMOS33} [get_ports {out_filt_G[1]}]
35 set_property -dict {PACKAGE_PIN T11 IOSTANDARD LVCMOS33} [get_ports {out_filt_G[2]}]
36 set_property -dict {PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports {out_filt_G[3]}]
37 set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports hsync_shifted]
38 set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33} [get_ports vsync_shifted]
39 #set_property -dict {PACKAGE_PIN V12 IOSTANDARD LVCMOS33} [get_ports {Out_pix_G[2]}]
40 #set_property -dict {PACKAGE_PIN W13 IOSTANDARD LVCMOS33} [get_ports {Out_pix_G[3]}]
41

```

La fréquence est réglée à 125MHz, celle de la carte coraZ7.

- Implémentation : étude du rapport de timing.

L'horloge : clk

Clock	Waveform(ns)	Period(ns)
clk	(0.000 4.000)	8.000
clk_25_PLL	(0.000 19.862)	39.724
clkfbout_PLL	(0.000 16.000)	32.000
dbg_hub/in1/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK	(0.000 16.500)	33.000
sys_clk_pin	(0.000 4.000)	8.000
clk_25_PLL_1	(0.000 19.862)	39.724
clkfbout_PLL_1	(0.000 16.000)	32.000

Vérification des violations de set up et du hold.

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints
26.471	0.000	0	3937	0.011	0.000	0	0

Les valeurs dans le THS et TNS sont à 0. Il n'y a pas de violations du set up et du hold. Donc pas de métastabilité.

From Clock:	clk_25_PLL						
To Clock:	clk_25_PLL						
Setup :	0	Failing Endpoints, Worst Slack	31.211ns,	Total Violation	0.000ns		
Hold :	0	Failing Endpoints, Worst Slack	0.159ns,	Total Violation	0.000ns		
PW :	0	Failing Endpoints, Worst Slack	18.612ns,	Total Violation	0.000ns		

Le chemin critique donne

```
Max Delay Paths
-----
Slack (MET) : 31.211ns (required time - arrival time)
  Source: <hidden> (rising edge-triggered cell FDRE clocked by clk_25_PLL {rise@0.000ns fall@19.862ns period=39.724ns})
  Destination: <hidden> (rising edge-triggered cell FDRE clocked by clk_25_PLL {rise@0.000ns fall@19.862ns period=39.724ns})
```

Max Delay Paths

```
Slack (MET) : 31.211ns (required time - arrival time)
  Source: <hidden> (rising edge-triggered cell FDRE clocked by clk_25_PLL {rise@0.000ns fall@19.862ns period=39.724ns})
  Destination: <hidden> (rising edge-triggered cell FDRE clocked by clk_25_PLL {rise@0.000ns fall@19.862ns period=39.724ns})
```

- Test sur le matériel : Réalisation du bitstream et test du système sur le matériel réel. Vérification que l'affichage de l'échiquier et l'application du filtre fonctionnent correctement sur l'écran VGA.

Nous utilisons l'oscilloscope pour mesurer les signaux de synchronisation, les signaux de couleur (r, g, b) et vérifions leur conformité aux spécifications VGA.

3) Démonstration

Dans cette partie, nous documentons les résultats : compilation des résultats des tests, y compris les captures d'écran ou les enregistrements vidéo de l'affichage de l'image, de l'oscilloscope.

Affichage de L'ILA

Au début du test :

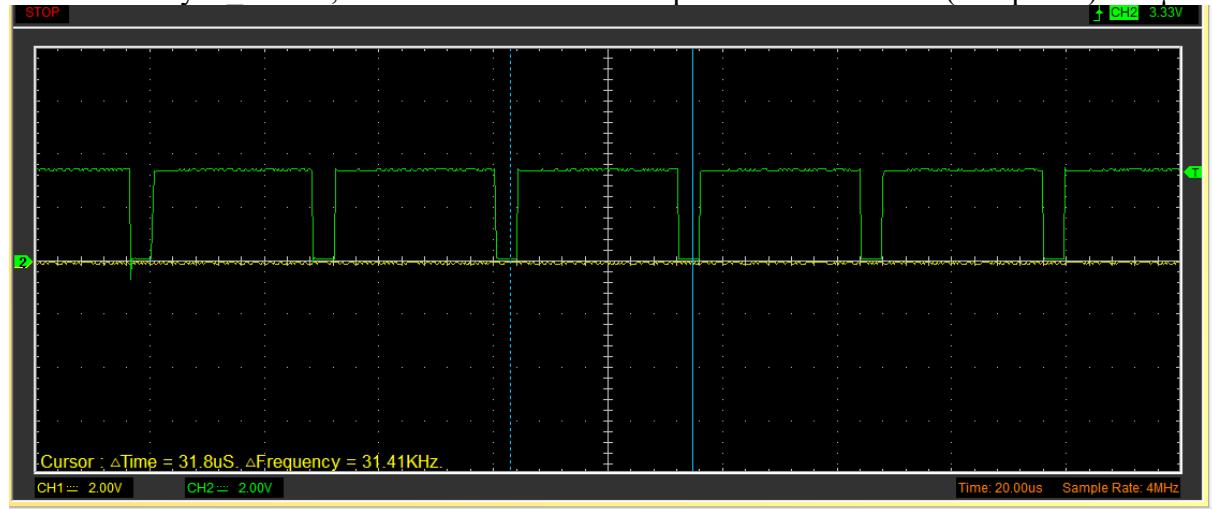
vsync=0 et hsync=0 ou 1.



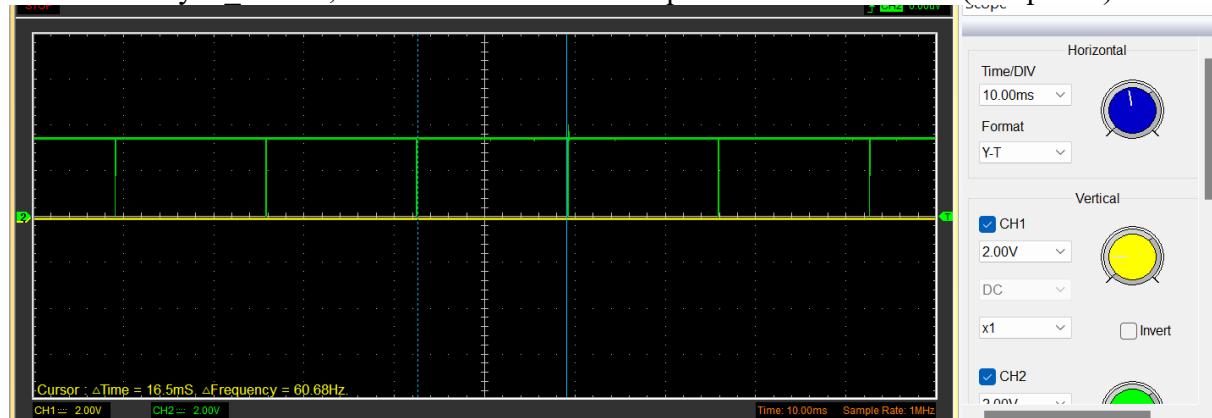
- Utilisons l'oscilloscope pour mesurer les signaux de synchronisation et vérifier leur conformité aux spécifications VGA.

Nous vérifions que les signaux de synchronisation (hsync et vsync) sont générés correctement et respectent les timings VGA.

Mesure de hsync shifted, nous trouvons bien la fréquence du whole line (800 pixels) à 32 μ s.



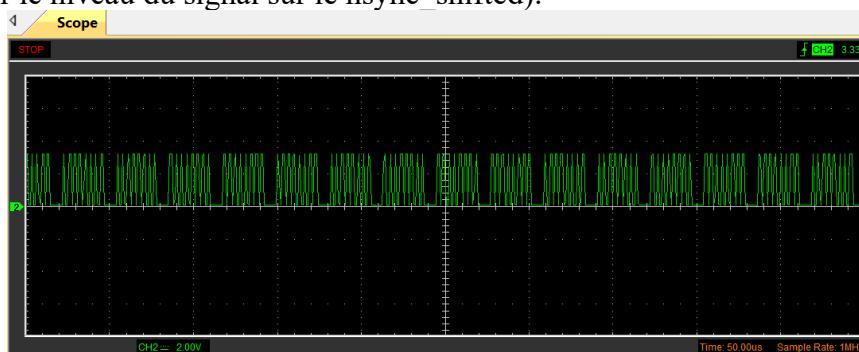
Mesure de vsync shifted, nous trouvons bien la fréquence du whole frame (525 pixels) à 16ms.



Lorsque nous superposons hsync shifted et vsync shifted, nous obtenons les 60Hz correspondant au screen refresh rate.

Nous pouvons donc conclure que les signaux hsync shifted et vsync shifted sont conformes à la norme VGA.

- La mesure des signaux de couleur (r, g, b) nous donne un niveau de tension de 3,3V, celle attendue (voir le niveau du signal sur le hsync shifted).

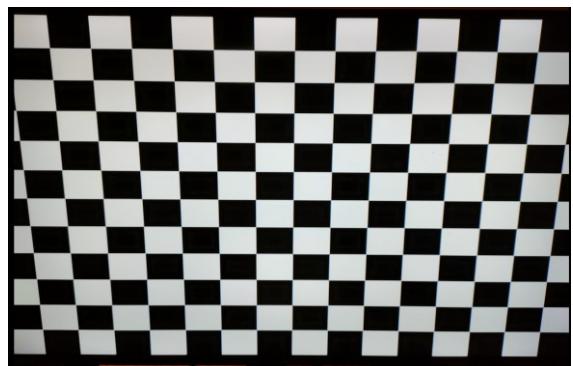


Nous validons ainsi le fonctionnement de notre système.

En connectant le VGA, nous constatons l'affichage du damier sur l'écran de télévision. L'image est stable et ne présente pas de problèmes d'affichage ou de clignotement.

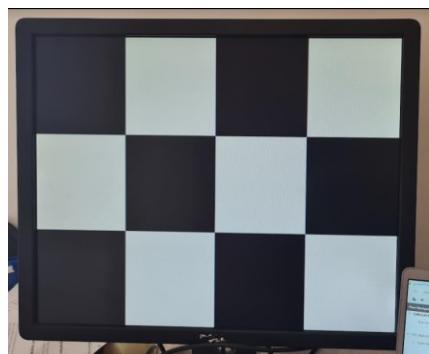
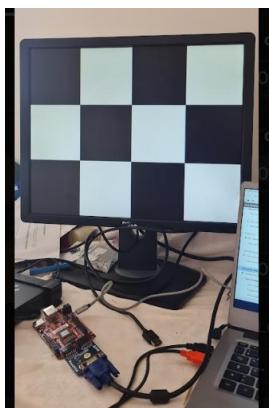
- Matrice identité
(Filter_sel='000')

Carré de largeur 40

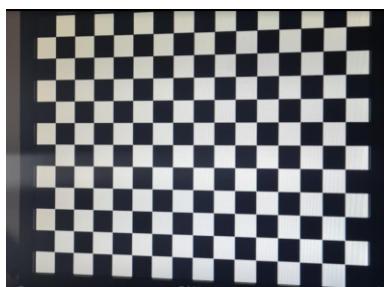


- Flou Gaussien
(Filter_sel='001')

Carré de largeur 160

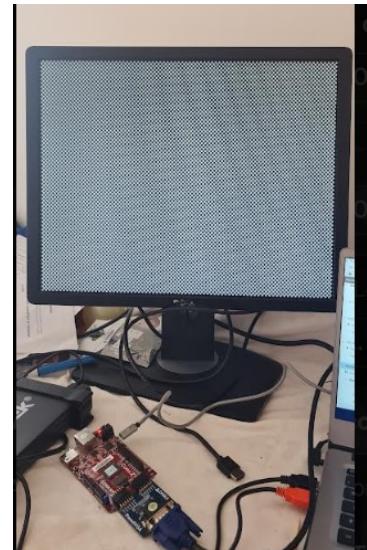
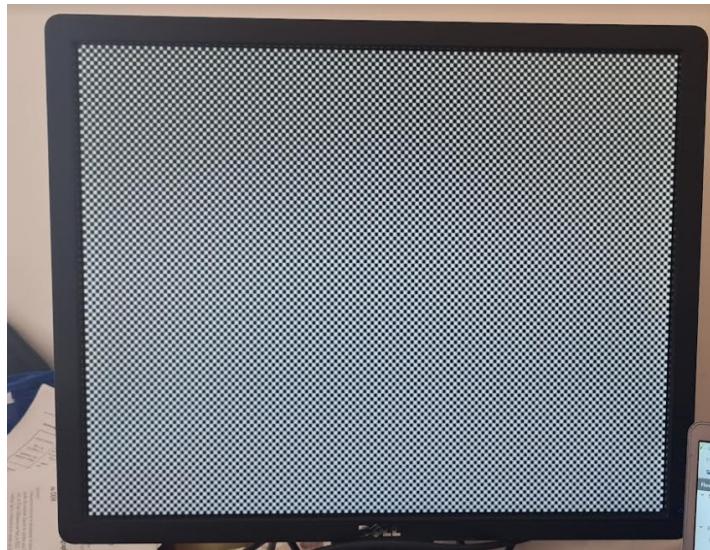


Carré de largeur 40



Carré de largeur 4

Le flou est plus prononcé que sur l'image avec une largeur de 40.



Carré de largeur 2

Les images ci-dessous représentent seulement une partie de l'écran.

Image sans filtre

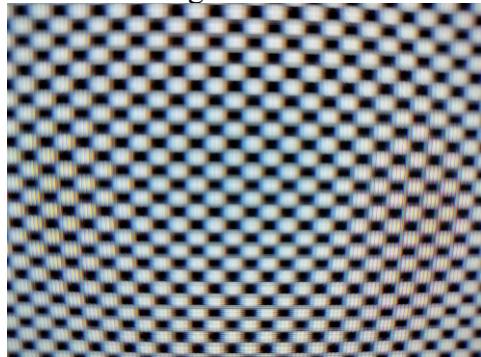
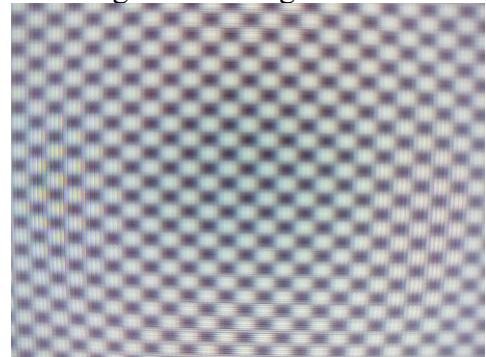


Image avec filtrage de Gauss

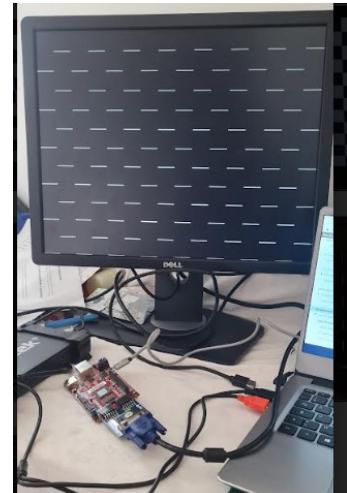


Les noyaux étant de dimensions 3x3, les carrés deviennent gris suite au filtrage de Gauss.

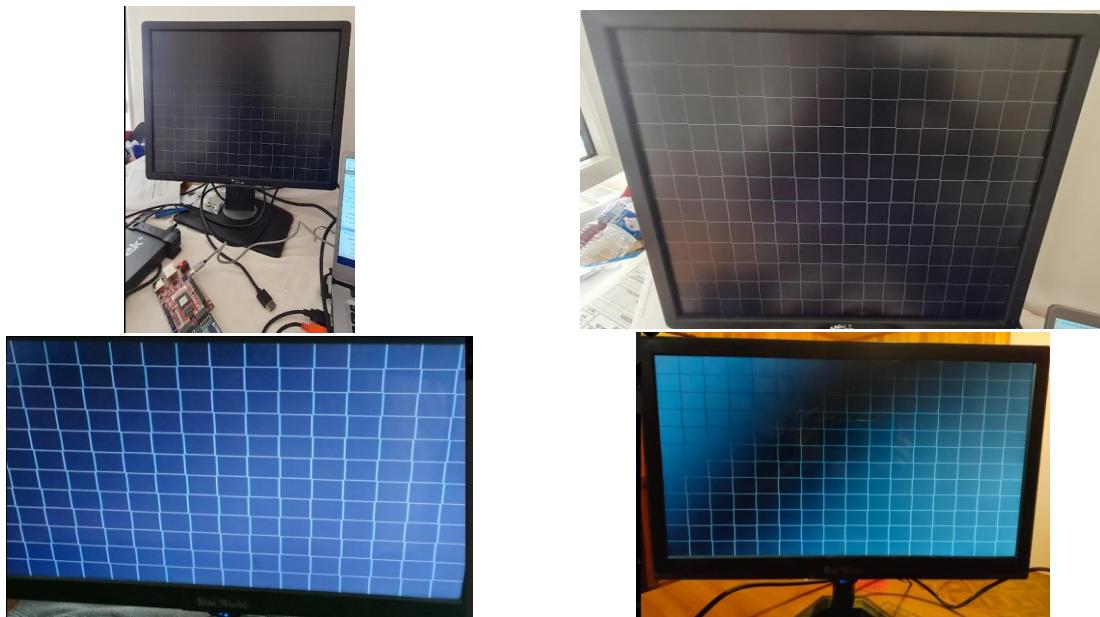
- Sobel horizontal
(Filter_sel='010')



- Sobel vertical
(Filter_sel='011')



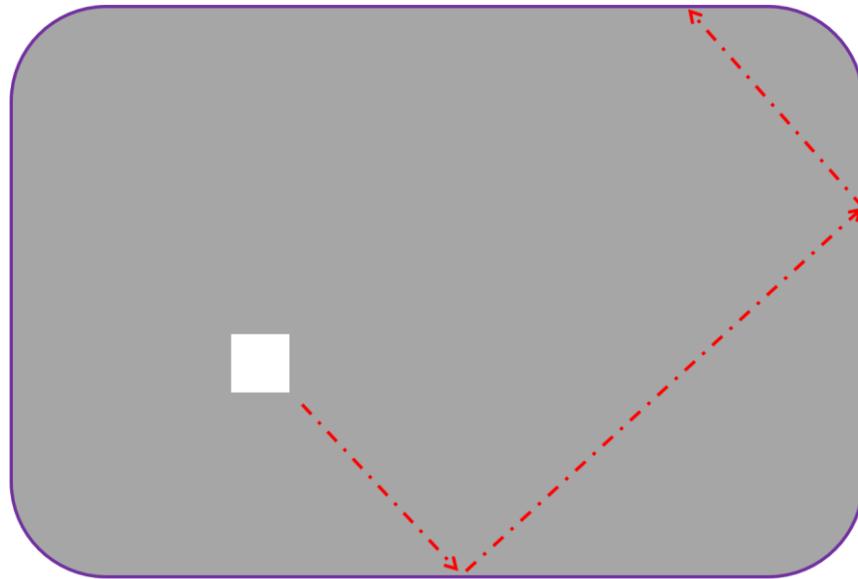
- Détection de Edge
(Filter_sel='100')



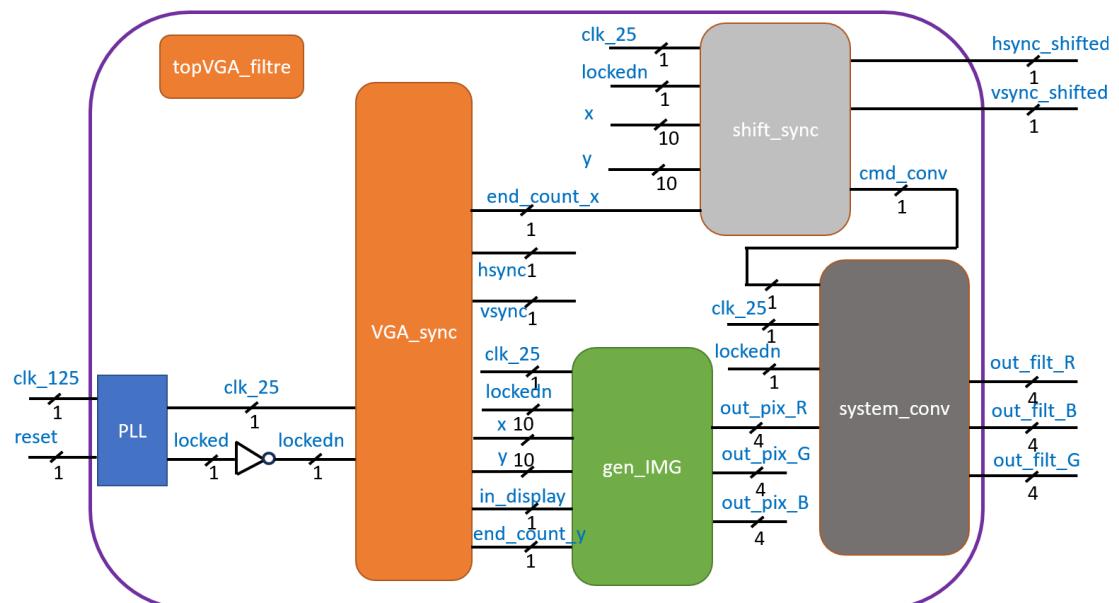
IV. Phase supplémentaire : déplacement d'un carré sur un écran VGA

L'objet de ce chapitre est d'afficher sur un écran un carré qui se déplace et rebondit sur les bords de l'écran.

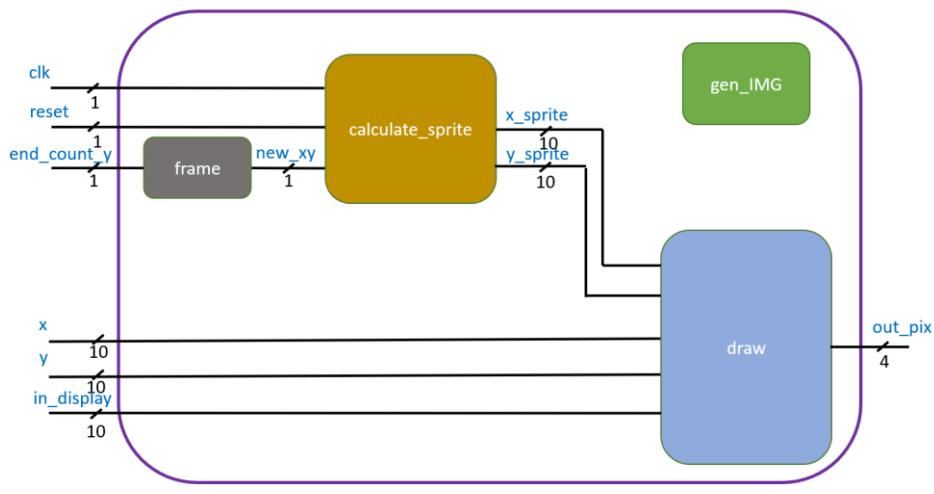
Afin de simplifier l'étude, nous avons opté pour un déplacement du carré en diagonale.



Pour réaliser ce projet, nous avons repris l'architecture de la phase finale en remplaçant le module créant le damier (gen_MIR) par un nouveau module (gen_IMG).



Principe de fonctionnement



- **Frame**

Ce composant permet le déplacement du carré toutes les x images.

- **calculate_sprite**

A chaque réception du signal **new_xy**, les nouvelles coordonnées du carré sont calculées.

- **draw**

Ce composant fournit les niveaux d'intensité des pixels au cours du balayage. Si (x, y) sont dans le carré, affichage des couleurs du carré, sinon affichage du fond d'écran.

V. Conclusion

Le projet soumis à notre étude était de réaliser une IP de traitement d'image sur cible Zynq7020 avec affichage VGA. Les résultats et démonstrations attestent du bon fonctionnement du module mis en place.

A travers notre étude, nous avons mis en place plusieurs filtres (filtre de Gauss, filtre de Sobel, détection de contour). Le code a été conçu pour laisser la possibilité au développeur d'ajouter des filtres.

L'architecture a été ensuite adaptée pour afficher le déplacement d'un carré sur un écran VGA.

Il serait intéressant pour compléter l'étude de tester le code en plaçant en entrée une image quelconque. Ainsi, nous aurions besoin de placer l'image en mémoire. La mémoire sur la partie FPGA de la carte CoraZ7 n'étant pas suffisante, nous devrions utiliser la RAM (DDR3L) présente sur la carte et la piloter par le SOC (System On a Chip) de la partie FPGA.

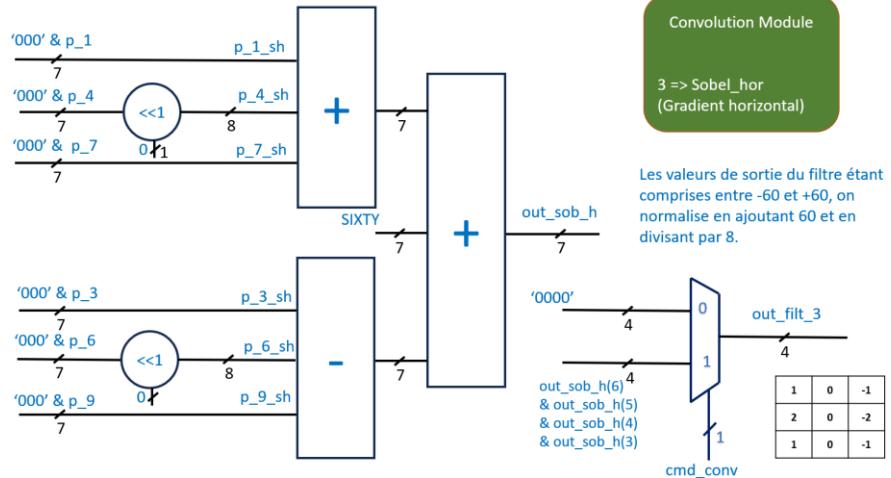
Affichage de l'image avec différents filtres.

Enchainement de filtres, cascader

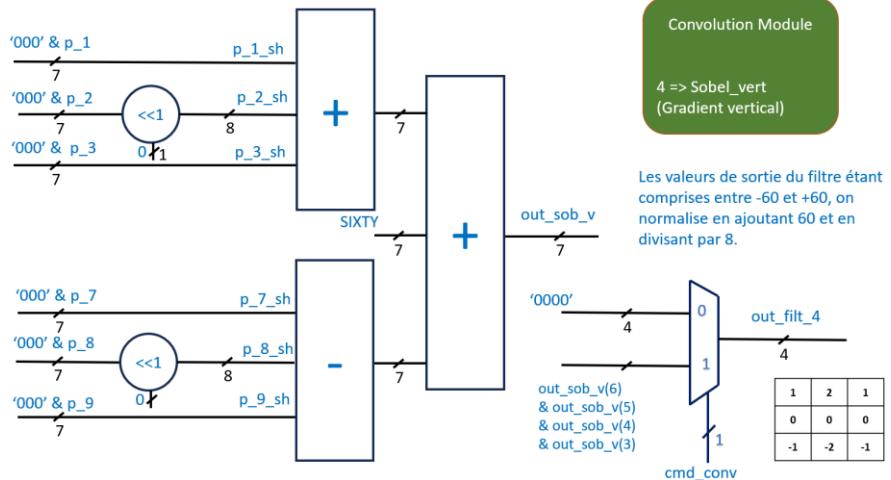
ANNEXE

- Convolution module / Autres filtres

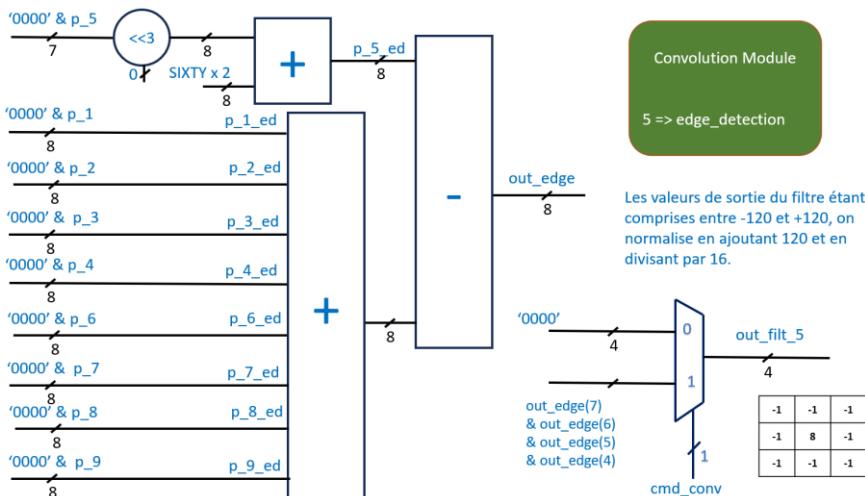
- Sobel horizontal



- Sobel vertical



- Edge detection



- Convolution module / Gestion des bords

Pas besoin de prévoir un traitement des bords de l'image.
Ils se feront automatiquement en raison de la zone blanche.

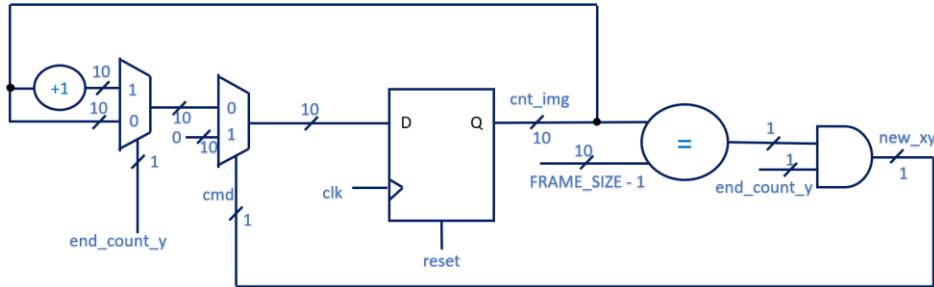
L'image ci-contre illustre bien le fonctionnement.

L’image visible est entourée de « 0000 » provenant de l’image virtuelle.

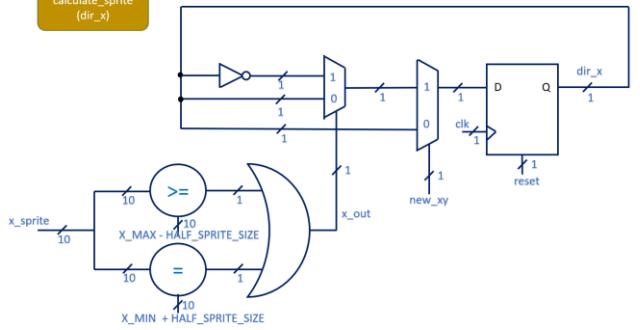
• Schémas de la phase supplémentaire

frame

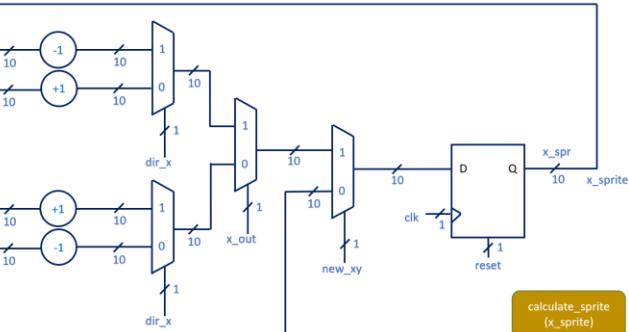
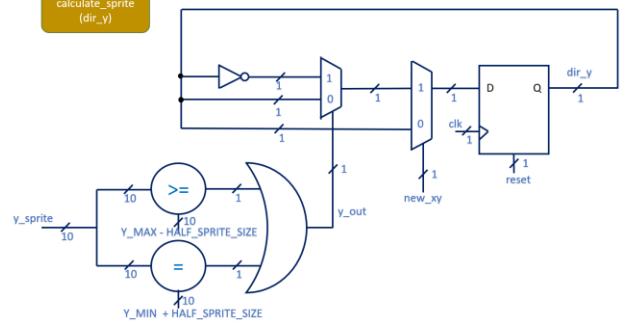
Ce composant permet de compter plusieurs vsync.
Nous ne changeons pas la position du lutin à chaque nouvelle image,
mais toutes les x images.



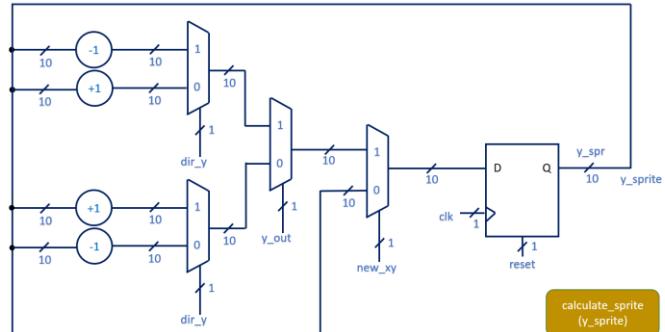
calculate_sprite (dir_x)



calculate_sprite (dir_y)



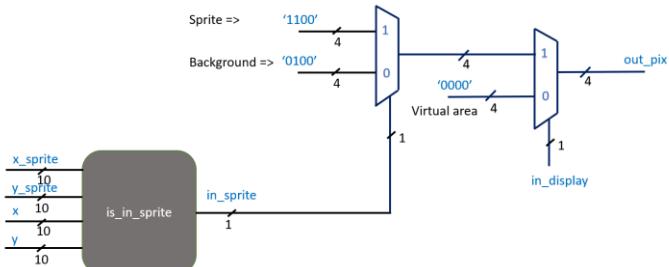
calculate_sprite (x_sprite)



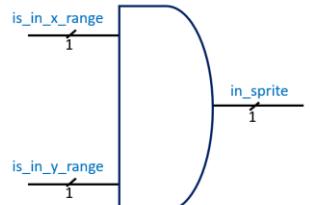
calculate_sprite (y_sprite)

draw

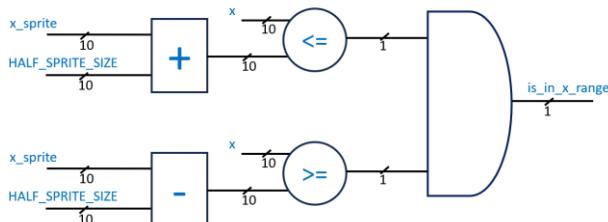
Ce composant permet de dessiner l'image avec le lutin.
Si nous sommes sur la position du lutin, nous dessinons le lutin
sinon nous dessinons le fond d'écran.



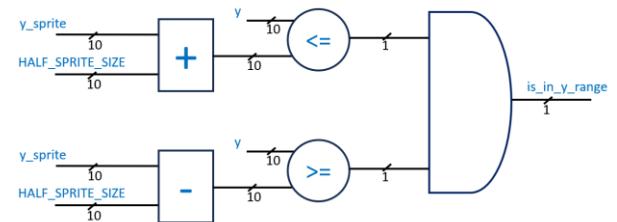
is_in_sprite (traitement final)



is_in_sprite (traitement des x)



is_in_sprite (traitement des y)



- Bibliographie