



COMPTE RENDU

DEVELOPPEMENT PROJET

INTERFACE VIDEO VGA

Formation AJC FPGA - Eve CHAR



03 JUILLET 2023
EVE CHAR

Table des matières

INTRODUCTION.....	3
I. ETUDE DE FAISABILITE ET DE PLAN DE TRAVAIL.....	4
1) EXPLICATION DU PLAN DE DEVELOPPEMENT UTILISE	4
2) VGA « VIDEO GRAPHICS ARRAY »	5
a. Les signaux de contrôle VGA	5
b. Principe du balayage	5
II. DEVELOPPEMENT DU PROJET-PHASE 1 (ETAPE INTERMEDIAIRE).....	7
1) SCHEMA BLOC DU SYSTEME.....	7
2) HIERARCHIE DES DIFFERENTS MODULES	8
3) GENERATION DE LA « PIXEL CLOCK » - CLOCK_GEN_01	8
a. Explication	8
b. Entrée et sortie de module Clok-gen : (IP catalogue).....	8
4) MODULE GLOBALE DU CONTROLEUR VGA.....	9
a. Architecture globale du Composant VGA_640x480	9
b. Entrées et sorties du module de génération des signaux de synchronisation hsync et vsync:	9
c. Définition des valeurs des différentes constantes pour représenter les paramètres temporels.....	10
d. Calcul du timing horizontal (explication).....	10
e. Calcul du timing vertical (explication)	10
f. Module de génération des compteurs de Position des pixels.....	11
5) MODULE CONVERTISSEUR DAC- PMOD-VGA	11
a. Analyse de la datasheet du composant SN74ALVC245	12
b. Reproduction du CNA sur LtSpice (SN74ALVC245).....	13
6) VUE D'ENSEMBLE DE L'ETAPE INTERMEDIAIRE (PHASE1).....	14
III. DEVELOPPEMENT DU PROJET - PHASE2 (AVEC FILTRE GAUSSIEN)	15
1) SCHEMA BLOC DE CONTROLEUR VGA AVEC UN FILTRE DE GAUSS (FIR) (FINITE IMPULSE RESPONSE)	15
2) HIERARCHIE DES DIFFERENTS MODULES	16
3) PRINCIPE DE FONCTIONNEMENT DE FILTRE DE GAUSS DANS NOTRE PROJET VGA:	16
4) APPLICATION DU FILTRE DE GAUSS.....	18
a. Explication : Fonctionnement des 2 Fifo	19
b. Latence de l'ensemble du système :	20
5) EXPLICATION DU DEGRADE DE NIVEAU DE GRIS DANS LES BORDS	21
IV. SYNTHESE ET ANALYSE DES TESTS.....	23
1) SYNTHESE DES TESTS.....	23
2) ANALYSES DES RESULTATS	24
a. Phase 1 du projet	24
b. Phase 2 du projet	26
c. Leçons retenues et capitalisation :	27
V. CONCLUSION	29
VI. BIBLIOGRAPHIE - WEBOGRAPHIE.....	30
ANNEXE.....	31

Table des Figures

FIGURE 1: ORGANIGRAMME REPRESENTANT LA METHODOLOGIE GLOBALE DU TRAVAIL A EFFECTUER	4
FIGURE 2: ORGANIGRAMME DU PROCESSUS D'ENTREE DU CODE VHDL JUSQU'A L'IMPLANTATION.	4
FIGURE 3: APERÇU DU	4
FIGURE 4: DESCRIPTION DU CONNECTEUR VGA.....	5
FIGURE 5: PRINCIPE DE FONCTIONNEMENT D'UN ECRAN.....	6
FIGURE 6: ETAPE INTERMEDIAIRE -PROJET	7
FIGURE 7: ARCHITECTURE GLOBALE- ETAPE INTERMEDIAIRE -PROJET.....	7
FIGURE 8: MODULE DU COMPOSANT CONTROLEUR VGA-640x480.....	9
FIGURE 9: SCHEMA RTL DU MODULE DU COMPOSANT CONTROLEUR VGA-640x480	9
FIGURE 10: RESUME DES PARAMETRES TEMPORELS DE SYNCHRONISATION	11
FIGURE 11 : COMPOSANT PMOD-VGA	11
FIGURE 12 : ARCHITECTURE GLOBALE- ETAPE INTERMEDIAIRE -PROJET	14
FIGURE 13 : SCHEMA RTL DE L'ARCHITECTURE GLOBALE- ETAPE INTERMEDIAIRE -PROJET	14
FIGURE 15 : SCHEMA SIMPLIFIANT LA COMPREHENSION DE L'ARCHITECTURE GLOBALE- PHASE2 -PROJET	16
FIGURE 16 : MASQUE FILTRE GAUSSIEN 3x3	17
FIGURE 17 : PRINCIPE DE FONCTIONNEMENT DU FILTRE GAUSSIEN 3x3	17
FIGURE 19 : LES PIXELS DU BORD DU FILTRE GAUSSIEN 3x3.....	20
FIGURE 20 : DEGRADE DE GRIS.....	22
FIGURE 21 : MATRICE DES TESTS PAR EXIGENCES	24

Introduction

Ce projet a pour objectif la conception en langage VHDL (« Very High Speed Integrated Circuit Hardware Description Language ») et l'implémentation dans une composante FPGA (Field Programmable Gate Array) d'un afficheur vidéo numérique avec un traitement d'image (filtre gaussien).

Le projet final est un contrôleur d'écran à plusieurs résolutions et fréquences de rafraîchissement.

Ce document est divisé en plusieurs chapitres qui présentent les différents aspects ayant rapport à l'élaboration de ce projet. Voici donc un plan du contenu de ce document et qui permet de suivre le cheminement logique suivi lors de la rédaction de ce document.

- Le premier chapitre a pour buts de montrer l'étude de faisabilité et de structurer le travail à accomplir (méthode et planning)
- Le deuxième chapitre présente la première partie du projet : Etape intermédiaire qui consiste à la mise en place d'un contrôleur VGA simple pour l'affichage d'un pattern (Damier) sur un écran LCD.
- Le troisième chapitre présente la deuxième partie du projet qui consiste à l'ajout d'un filtre gaussien entre le générateur de pattern et le contrôleur VGA. La taille du masque de ce filtre est de 3X3.
- Le quatrième chapitre contient les résultats des tests permettant de valider l'implémentation, avec l'analyse des résultats de tests.

Le code VHDL des différentes parties du projet seront placés en annexe, suivi par les bancs d'essais (test bench), des scripts de compilations et, pour terminer, le fichier de contraintes top.dxc.

Des documents sont également joints : la procédure de test (plan de validation) et les résultats de test (Recette).

I. Etude de faisabilité et de plan de travail

1) Explication du plan de développement utilisé

Voici le plan de développement suivi pour faire avancer le travail. Avant de se lancer dans des explications détaillées, il est préférable de visualiser schématiquement la logistique de chaque tâche sur la Figure 1, la Figure 2, et la Figure 3 qui suivent :

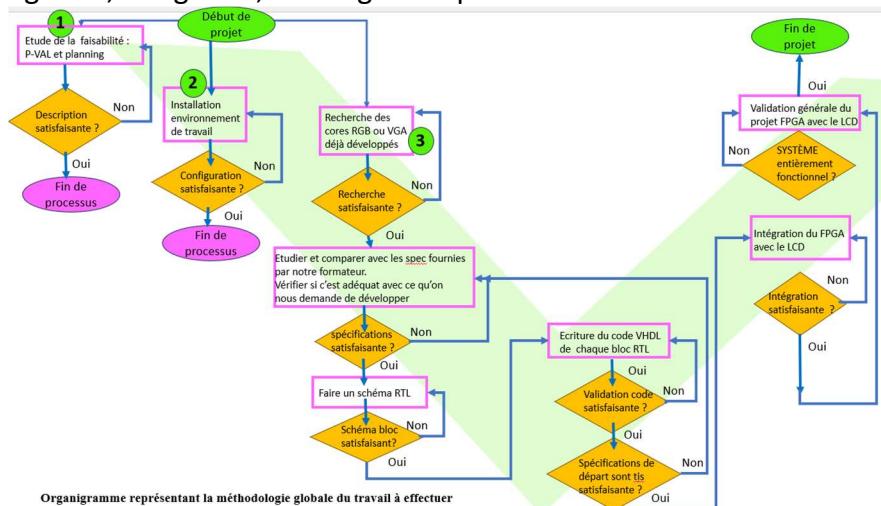


Figure 1: Organigramme représentant la méthodologie globale du travail à effectuer

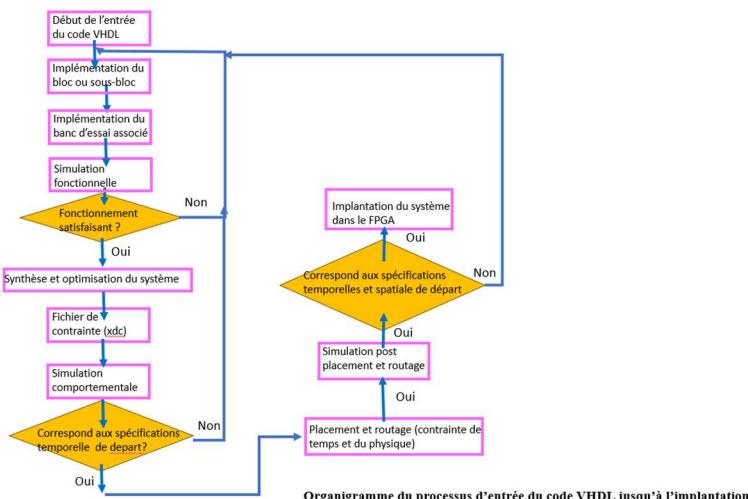


Figure 2: Organigramme du processus d'entrée du code VHDL jusqu'à l'implantation.

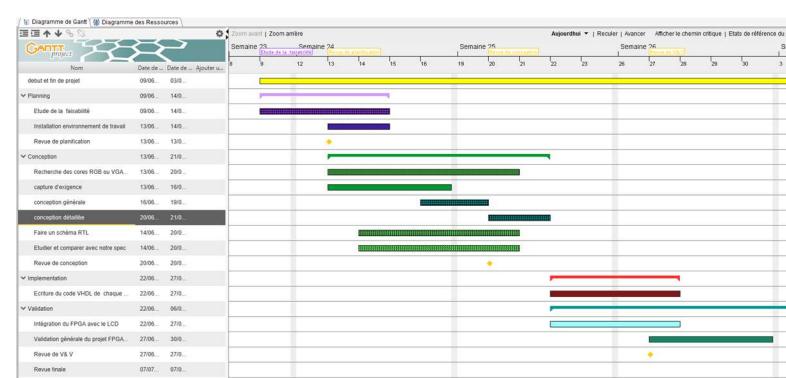


Figure 3: aperçu du .

2) VGA « video graphics array »

VGA désigne à la fois un protocole d'affichage vidéo, un format d'affichage (le format 640x480 pixels) et le fameux connecteur 15 broches à trois rangées que tout le monde connaît.

L'avantage du VGA est sa simplicité : trois broches pour les signaux RGB analogiques et deux broches de synchronisation horizontale et verticale.

Rafraîchir un écran 60 fois par seconde à la résolution standard de 640x480 pixels revient à piloter en parallèle tous ces signaux à une fréquence proche de 25 MHz (voir explications ci-dessous), ce qui justifie l'emploi d'un FPGA comme contrôleur.

a. Les signaux de contrôle VGA

Le rôle d'un contrôleur VGA est de générer les signaux nécessaires qui, à l'époque des écrans et moniteurs à tube cathodique (ou CRT), permettaient de gérer les mouvements de balayage du spot en sortie du canon à électrons ainsi que l'intensité des faisceaux rouge, vert et bleu déterminant la couleur du point. Les écrans et moniteurs modernes n'utilisent évidemment plus cette technologie et sont passés à l'ère du numérique, mais on y trouve encore des ports vidéo VGA pour émuler ce balayage.

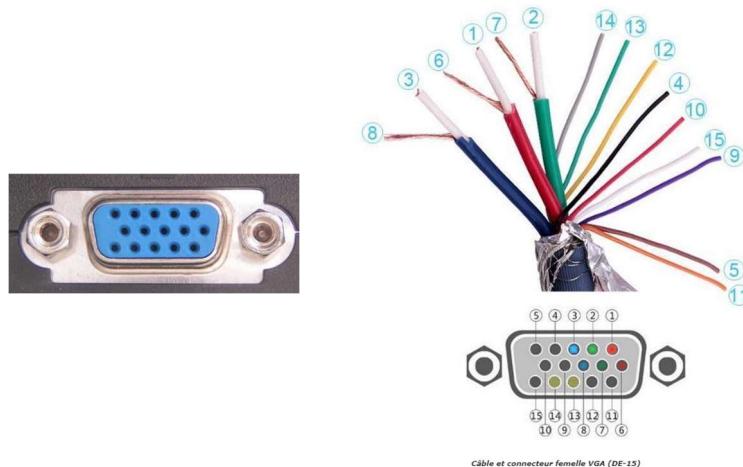


Figure 4: Description du connecteur VGA

b. Principe du balayage

Afficher une image sur un écran consiste à imposer une couleur particulière sur chaque point élémentaire (ou pixel) de l'écran. Afficher une séquence vidéo consiste à enchaîner des images avec un certain taux de rafraîchissement.

Ici donc il s'agit d'afficher des images de résolution 640x480 avec un taux de rafraîchissement de 60Hz.

Pour une image, plutôt que d'afficher tous les points en même temps, il est plus simple de faire un balayage de l'écran à vitesse importante, la persistance des impressions lumineuses sur la rétine de l'œil donnant l'impression d'une image stable.

Le balayage de l'écran sera composé d'un balayage horizontal associé à un balayage vertical. Ainsi, après avoir balayé tous les pixels d'une ligne, de gauche à droite, on passe à la ligne suivante jusqu'à la dernière ligne en bas, puis on remonte à la première en haut.

Pour synchroniser ce balayage, nous aurons donc un signal Hsync de synchronisation horizontale et d'un signal Vsync de synchronisation verticale. Comme indiqué, à l'époque des affichages CRT, il fallait gérer les mouvements de balayage du spot en sortie du canon à électrons. Ce canon nécessitait un certain temps avant d'être de nouveau prêt pour générer la prochaine ligne ou pour remonter et générer la prochaine image. Hsync permettait donc de synchroniser l'envoi d'une nouvelle ligne et Vsync d'une nouvelle image.

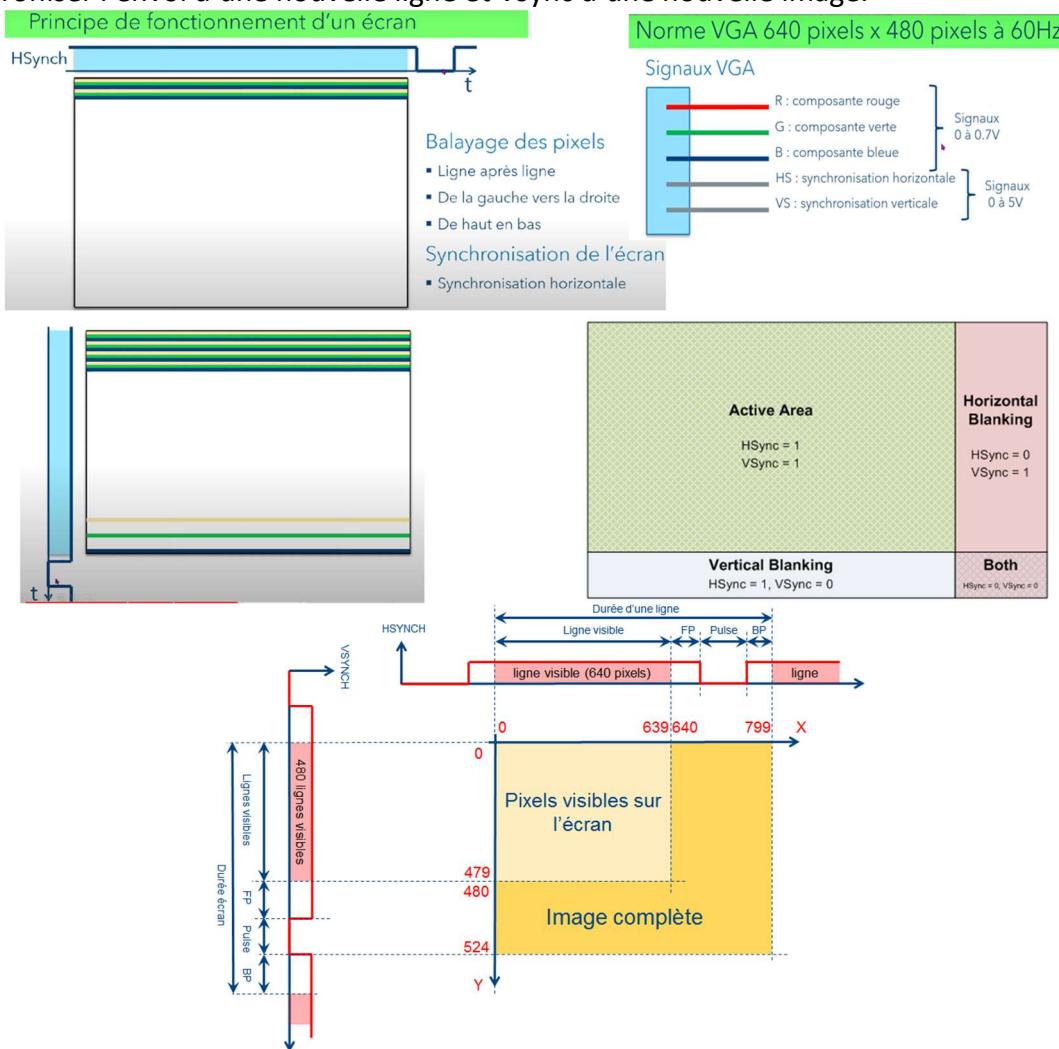


Figure 5: principe de fonctionnement d'un écran.

Pour afficher une image 640x480 et compte tenu des temps définis par la norme pour générer les signaux Hsync et Vsync, cela revient à définir une image virtuelle de 800x525 dont seuls les pixels entre 0 et 639 en horizontal et entre 0 et 479 en vertical seront visibles.

Les pixels entre 640 et 800 en horizontal et entre 480 et 525 en vertical ne servent qu'à positionner correctement notre image via les signaux Hsync et Vsync comme indiqué ci-dessus.

Le pixel (0,0) se trouve en haut à gauche de l'écran. C'est le premier à être envoyé à l'écran.

II. Développement du projet-phase 1 (étape intermédiaire)

Dans cette partie de projet, nous allons faire l'étape intermédiaire qui consiste à la mise en place d'un contrôleur VGA. Ce dernier va interfaçer un générateur de pattern et un écran pour contrôler l'affichage et sa synchronisation.

Ce chapitre expliquera en détail la fonctionnalité des différents modules. Pour chaque module, un descriptif et une explication des entrées et sorties sont donnés.

Par la suite, une explication du fonctionnement de ce module est appuyée par un schéma bloc et un schéma RTL.

1) Schéma bloc du système

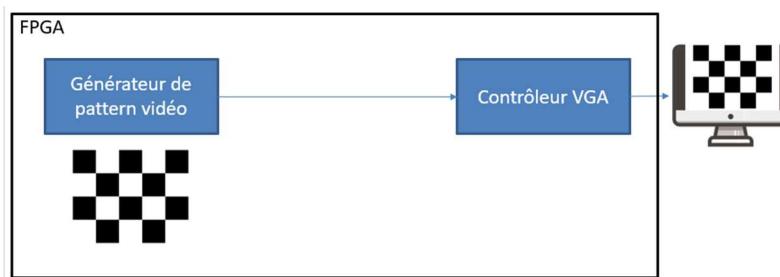


Figure 6: étape intermédiaire -projet.

Dans notre cas d'étude, afin de générer un affichage sur un écran de résolution de 640 x 480 pixels (selon la norme VGA), il faut :

1. Une fréquence pixel de 25,175 MHz
2. Des signaux de synchronisation avec des délais précis (expliqués par la suite (aussi donnés en nombre de pixels))
3. Une fréquence de rafraîchissement d'image de 60 fois par seconde

Nous concluons ainsi, que nous aurons besoin de :

- 1 module pour générer la bonne horloge (fonction `clock_gen_01`)
- 1 module pour générer la bonne synchronisation horizontale et verticale (fonction `horizontal_timing_gen_03` et `vertical_timing_gen_04`)
- 1 module pour identifier la position des pixels dans les deux axes horizontal et vertical (fonction `compteur_H_POS_05` et `compteur_V_POS_06`)
- 1 module pour faire la conversion numérique analogique et en même temps pour faire le rôle d'adaptation avec l'écran d'affichage

On résume ces modules ci- après :

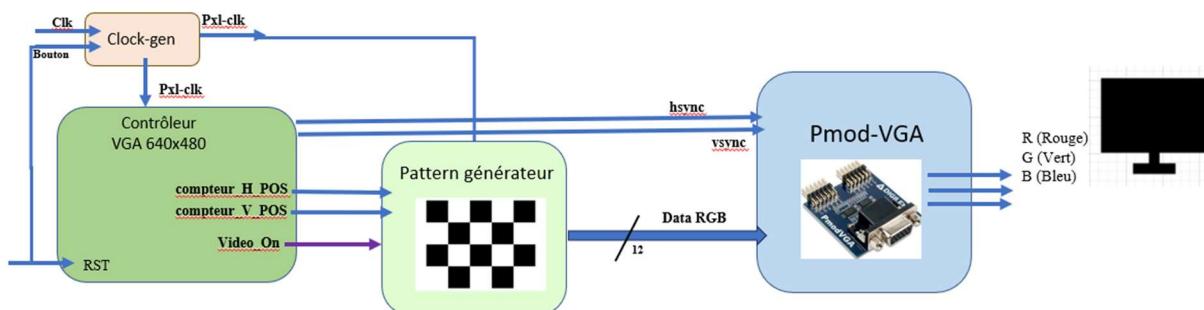


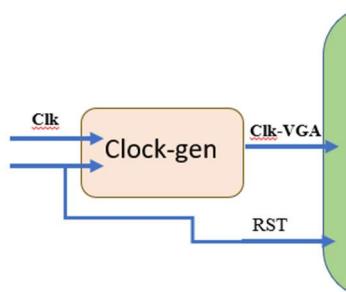
Figure 7: architecture globale- étape intermédiaire -projet.

2) Hiérarchie des différents modules

Cette section permet de situer rapidement chacun des modules dans la hiérarchie du système en partant du module de plus haut niveau qui est le module top et en descendant vers le bas où se trouvent les modules les plus simples. Voici donc cette arborescence des modules :



3) Génération de la « pixel clock » - clock_gen_01



La norme VGA dont la résolution est de 640x480 pixels à 60Hz impose une fréquence d'horloge pour les pixels de 25,2 MHz.

a. Explication

Le taux de rafraîchissement de l'écran est de 60 Hz → c'est-à-dire chaque 1/60ème de seconde. Nous avons ici une surface = 800 x 525 pixels, chaque pixel doit être traité en $1 / (60 \times 800 \times 525) \approx 39,7 \text{ ns}$ (nanosecondes).

→ Alors une horloge de $60 \times 800 \times 525 = 25,2 \text{ MHz}$ est donc nécessaire pour cadencer le processus de balayage pixel par pixel. Il faut donc générer un signal pxi_clk à 25,2 MHz.

b. Entrée et sortie de module Clok-gen : (IP catalogue)

- RST : remise à zéro (entrée)
- clk : horloge maître à 100MHz (entrée)
- pxi-clk (sortie) à 25.2 MHz

4) Module globale du Contrôleur VGA

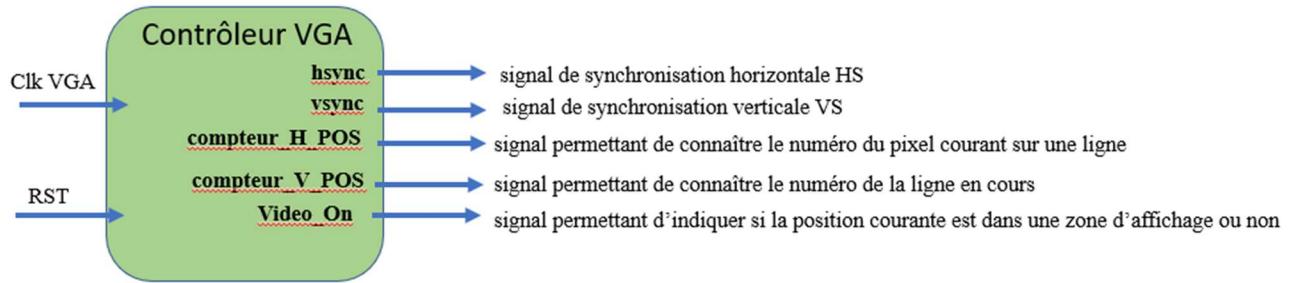


Figure 8: Module du composant contrôleur VGA-640x480

a. Architecture globale du Composant VGA 640x480

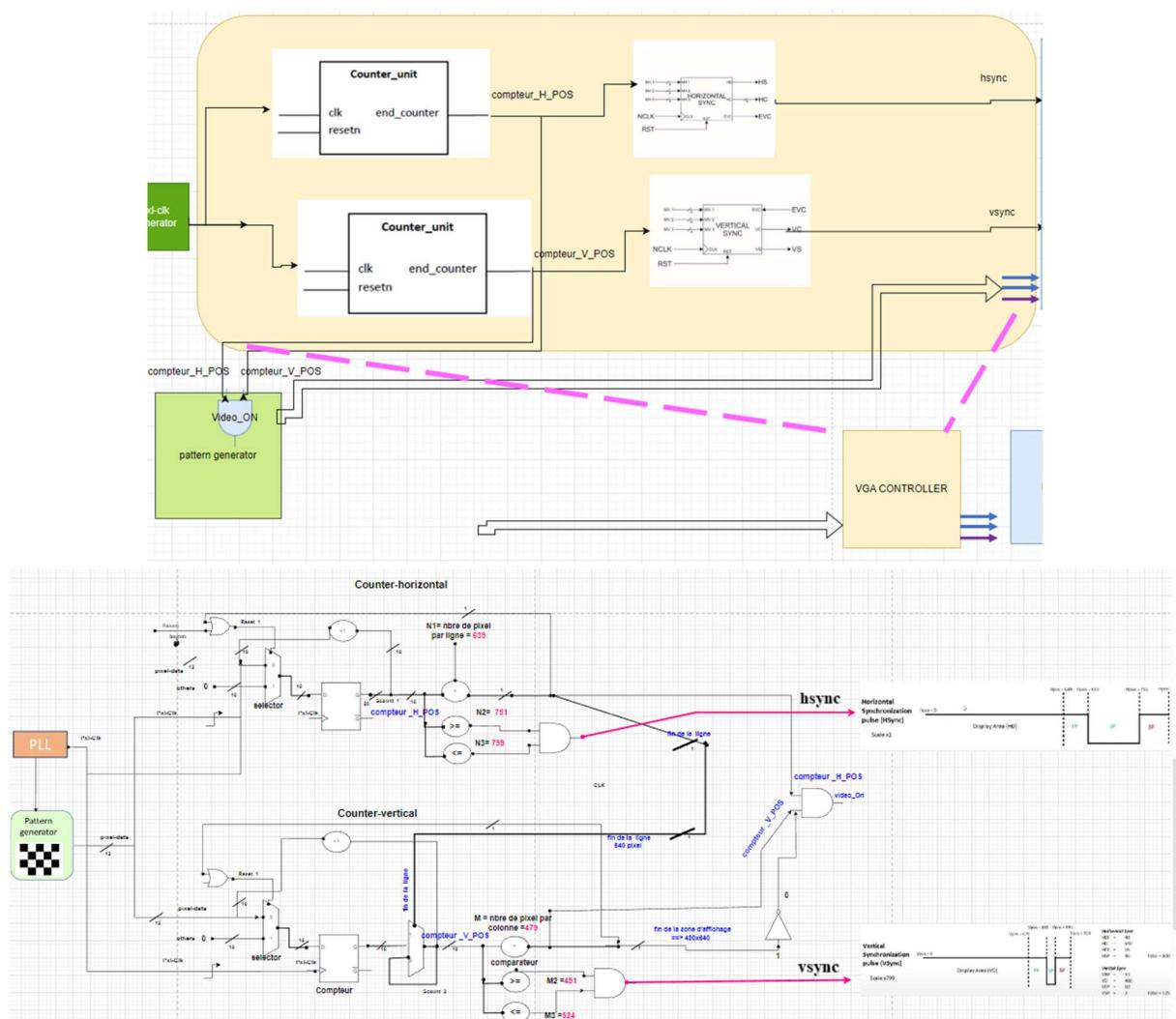


Figure 9: Schema RTL du Module du composant contrôleur VGA-640x480

b. Entrées et sorties du module de génération des signaux de synchronisation hsync et vsync:

L'objectif est de générer les signaux hsync et vsync nécessaires pour le balayage de l'écran.

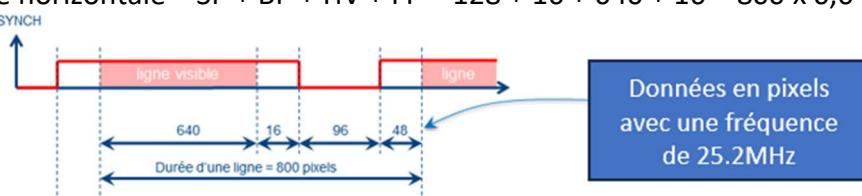
- En entrée :

- un signal d'horloge `pxl_clk` à 25,2 MHz, fréquence de balayage de l'écran pixel par pixel ;
 - un signal `RST` de réinitialisation, qui pourra être activé sur appui d'un bouton-poussoir intégré en surface de notre carte (`btn 0` par exemple).
 - En sortie : les signaux logiques de synchronisation horizontale et verticale `hsync` et `vsync` ;
- c. Définition des valeurs des différentes constantes pour représenter les paramètres temporels**

Constant <code>FRAME_WIDTH</code>	De type natural := 640 → représente la zone d'affichage en longueur (nombre de colonnes)
Constant <code>FRAME_HEIGHT</code>	De type natural := 480 → représente la zone d'affichage en hauteur (nombre de lignes)
Constant <code>H_FP</code>	De type natural := 16; H front porch width (pixels)
Constant <code>H_PW</code>	De type natural := 96 → largeur de pulse de synchronisation H
Constant <code>H_MAX</code>	De type natural := 800; → nombre de pixels (total) sur une ligne
Constant <code>V_FP</code>	De type natural := 10; V front porch width (lignes)
Constant <code>V_PW</code>	De type natural := 2 → largeur de pulse de synchronisation V (lignes)
Constant <code>V_MAX</code>	De type natural := 525 → nombre de lignes (total) par frame

d. Calcul du timing horizontal (explication)

- Pixel clock = 25.2 MHz → Pixel time ≈ 0,04 µs
- Largeur de pulse de synchronisation H = 96 pixels x 0,04 µs = 3,84 µs
- $H_{BP} = 48 \text{ pixels} \times 0,04 \mu\text{s} = 1,92 \mu\text{s}$
- la zone d'affichage horizontale (`FRAME_WIDTH H`) = 640 pixels x 0,04 µs = 25,6 µs
- $H_{FP} = 16 \text{ pixels} \times 0,04 \mu\text{s} = 0,64 \mu\text{s}$
- 1 ligne horizontale = SP + BP + HV + FP = $128 + 16 + 640 + 16 = 800 \times 0,04 \mu\text{s} = 32 \mu\text{s}$



e. Calcul du timing vertical (explication)

- Pixel clock = 25.2 MHz → durée de la ligne horizontale = 32 µs
- Largeur de pulse de synchronisation V = 2 lignes x 32 µs = 64 µs
- $V_{BP} = 29 \text{ lignes} \times 32 \mu\text{s} = 928 \mu\text{s}$
- la zone d'affichage Verticale = 480 lignes x 32 µs = 15,36 ms
- $V_{FP} = 10 \text{ lignes} \times 32 \mu\text{s} = 320 \mu\text{s}$
- Affichage vertical = SP + BP + VV + FP = $521 \times 32 \mu\text{s} = 16,67 \text{ ms} = 1/60 \text{ Hz}$

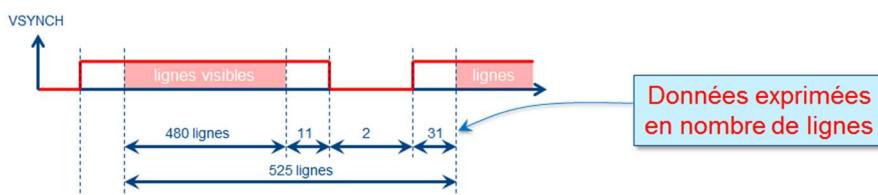


Figure 10: Résumé des paramètres temporels de synchronisation

f. Module de génération des compteurs de Position des pixels

L'objectif est de générer les signaux compteur_H_POS et compteur_V_POS nécessaires pour le balayage de l'écran.

Ces compteurs représentent les coordonnées d'une position sur l'écran (donc un pixel) :

- compteur_H_POS est situé entre (0 et 799)
- compteur_V_POS est situé entre (0 et 524)

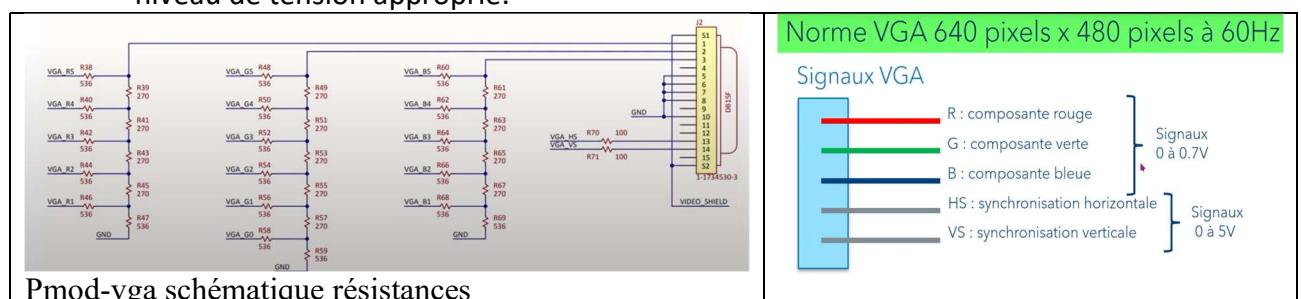
➔ (Il faut 10 bits pour écrire 800 en binaire : 0011 0010 0000 et 12 bits pour écrire 2200 en binaire, pour écran 1920 x 1080)

Un signal logique sera utilisé (Video_On sur les schémas) à l'état haut lorsque les coordonnées du pixel en cours sont dans la zone active d'affichage (compteur_H_POS entre 0 et 639, et compteur_V_POS entre 0 et 479) et à l'état bas sinon.

5) Module convertisseur DAC- Pmod-VGA

Pour interfaçer le contrôleur VGA avec un écran VGA. Il faut un composant supplémentaire pour compléter le système :

- **Pmod-VGA:** il s'agit d'un convertisseur DAC 4 bits qui convertit les données numériques RVB et pilote l'affichage VGA avec des signaux analogiques RVB au niveau de tension approprié.



Ce composant, dont l'interface est représentée ci-dessous, va relier le FPGA avec le moniteur VGA.

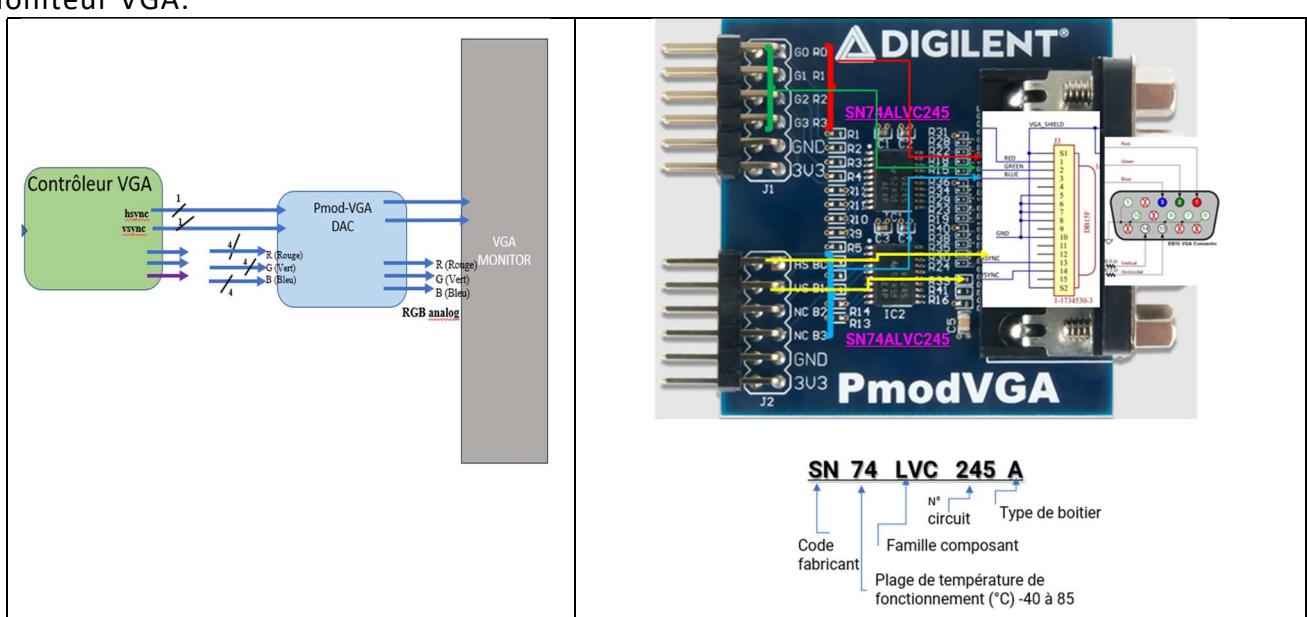


Figure 11 : composant Pmod-VGA

Ce composant est basé sur un processus purement **combinatoire** qui affecte aux sorties RED, GREEN, BLUE une valeur.

Ce module comporte 4 broches numériques par composante de couleur R, G ou B :

- R3, R2, R1 et R0 pour la composante rouge (Red) ;
- G3, G2, G1 et G0 pour la composante verte (Green) ;
- B3, B2, B1 et B0 pour la composante bleue (Blue).

Chaque composante de couleur est donc définie avec 4 bits et le convertisseur numérique-analogique associé (un simple réseau de résistances R-2R gravé en surface du module PmodVGA) pourra générer $2^{(puissance\ 4)} = 16$ niveaux de tension entre 0 et 0,7 V, soit une profondeur de couleurs 12 bits (RGB444).

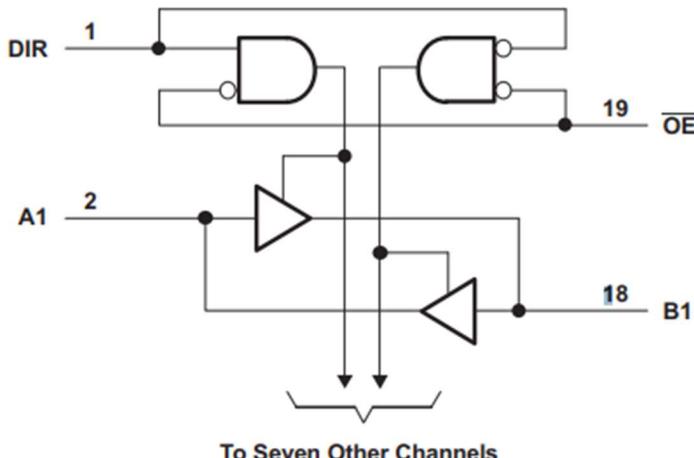
a. Analyse de la datasheet du composant SN74ALVC245

Le composant **SN74ALVC245** est :

- Un émetteur-récepteur de bus, de 8 bits (octal)
- Non inverseur,
- Avec des sorties à 3 états
- Un adaptateur de tension bidirectionnel, avec deux rails d'alimentation séparés :
 - o Le port B, qui est réglé sur 3,3 V
 - o Le port A, qui est réglé sur 5 V. Cela permet la conversion d'un environnement 3,3 V à un environnement 5 V, et vice versa.

Le composant **SN74ALVC245** transmet des données du bus A au bus B ou du bus B au bus A, selon le niveau logique à l'entrée de commande de direction (DIR)

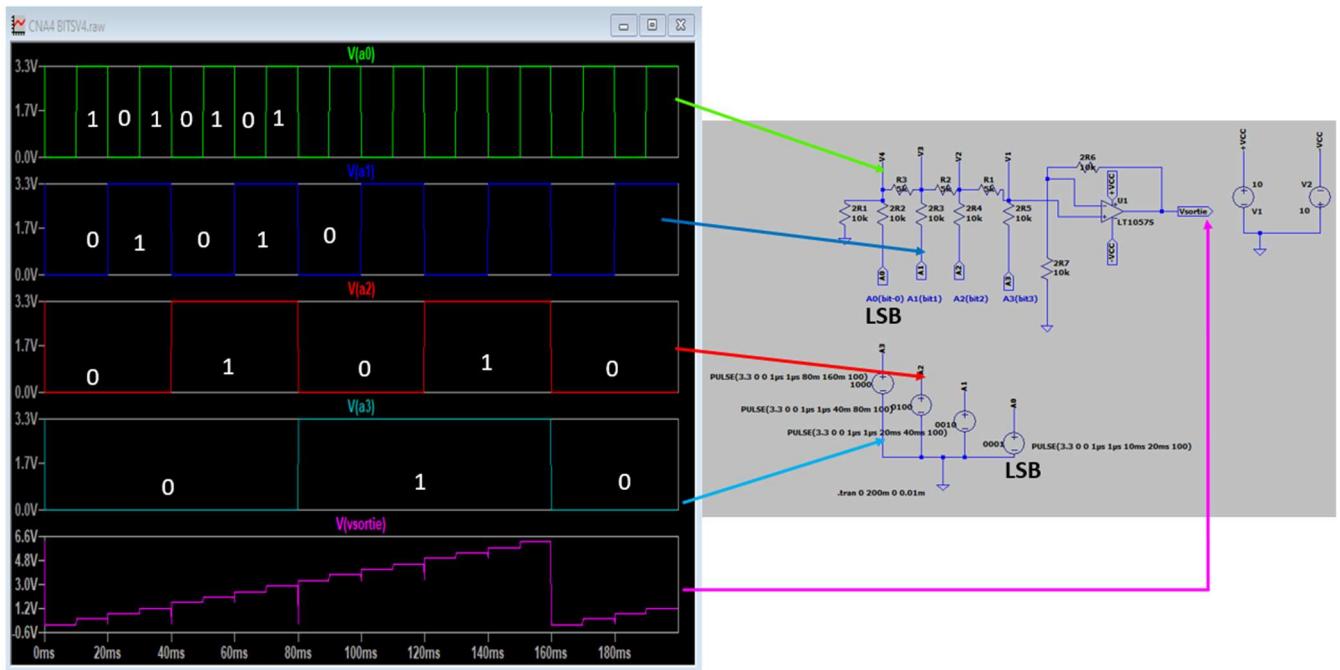
La sortie (not OE) peut être utilisée pour désactiver le composant afin que les bus soient efficacement isolés.



INPUTS		OPERATION
OE	DIR	
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

Voir annexe1 pour plus de détails sur le fonctionnement RTL de cet interface PmodVGA.

b. Reproduction du CNA sur LtSpice (SN74ALVC245)



Mot binaire 1000 → on aura $V_{\text{sortie}} = V_{\text{ref}}/2^1 \rightarrow 3.3/2 = 1.65\text{v}$

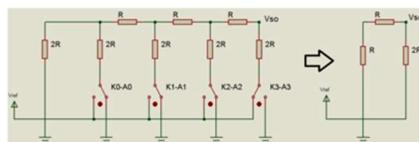
Mot binaire 0100 → on aura $V_{\text{sortie}} = V_{\text{ref}}/2^2 \rightarrow 3.3/2 = 0.825\text{v}$

Mot binaire 0010 → on aura $V_{\text{sortie}} = V_{\text{ref}}/2^3 \rightarrow 3.3/2 = 0.4125\text{v}$

Mot binaire 0001 → on aura $V_{\text{sortie}} = V_{\text{ref}}/2^4 \rightarrow 3.3/2 = 0.20625\text{v}$

Principe de CNA 4 bits à réseau R-2R

- Si le commutateur K3 est seul activé ($a_3 = "1"$) c.à.d. $N=(1000)_2$, le schéma du réseau devient :



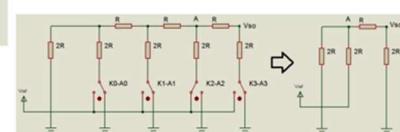
Dans ce cas la relation qui existe entre V_{so} et V_{ref} est :

$$V_{\text{so}} = \left(\frac{2R}{2R+2R} \right) V_{\text{ref}} = \frac{1}{2} V_{\text{ref}}$$

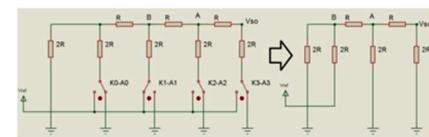
$V_A = R_{\text{eq}} V_{\text{ref}} / (R_{\text{eq}} + 2R)$ avec $R_{\text{eq}} = 6R/5$ donc $V_A = 3V_{\text{ref}}/8$

Alors $V_{\text{so}} = 2V_A/3$ d'où $V_{\text{so}} = V_{\text{ref}}/4$ c.à.d. $\boxed{V_{\text{so}} = V_{\text{ref}}/2^2}$

- Si le commutateur K2 est seul activé ($a_2 = "1"$) c.à.d. $N=(0100)_2$, le schéma du réseau devient :



- Si le commutateur K1 est seul activé ($a_1 = "1"$) c.à.d. $N=(0010)_2$, le schéma du réseau devient :



$V_B = V_{\text{ref}} R_{\text{eq}} / (R_{\text{eq}} + 2R)$ avec $R_{\text{eq}} = 22R/21$ donc $V_B = 11V_{\text{ref}}/32$

$V_A = V_B R_{\text{eq}} / (R_{\text{eq}} + 2R)$ alors $V_A = 6V_B/11$

Comme $V_{\text{so}} = 2V_A/3$, alors $V_{\text{so}} = (2*6)V_B/(3*11)$; $V_{\text{so}} = (2*6)(11V_{\text{ref}}/32)/(3*11)$

$V_{\text{so}} = V_{\text{ref}}/8$ c.à.d. $\boxed{V_{\text{so}} = V_{\text{ref}}/2^3}$

- Si le commutateur K0 est seul activé ($a_0 = "1"$) c.à.d. $N=(0001)_2$, le schéma du réseau peut être trouvé et par analogie on peut trouver facilement $\boxed{V_{\text{so}} = V_{\text{ref}}/2^4}$

6) Vue d'ensemble de l'étape intermédiaire (phase1)

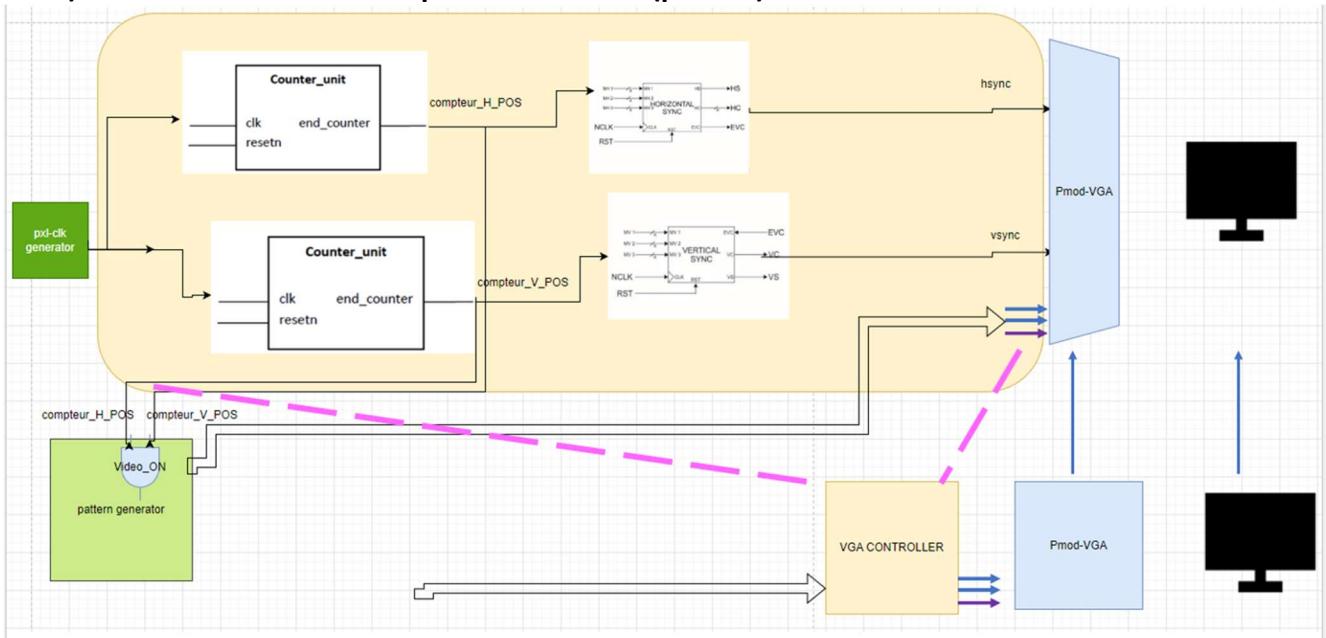


Figure 12 : architecture globale- étape intermédiaire -projet.

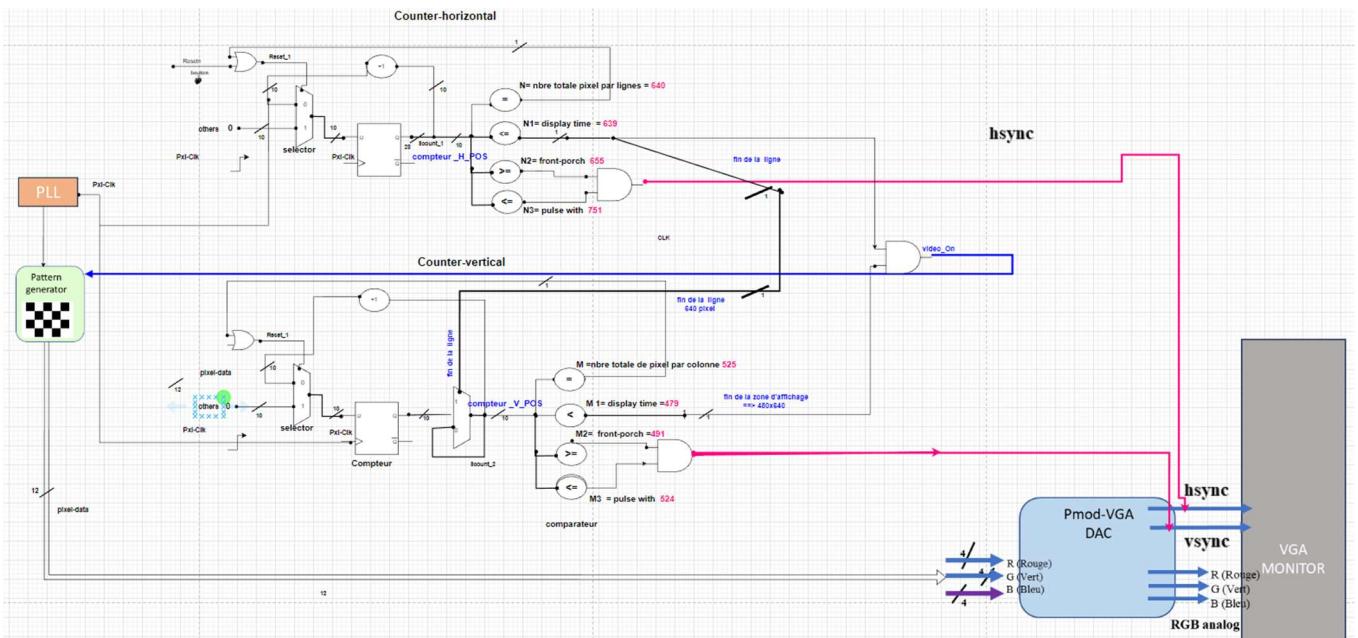


Figure 13 : schéma RTL de l'architecture globale- étape intermédiaire -projet

III. Développement du projet - phase2 (avec filtre Gaussien)

Dans cette partie du projet, nous allons faire la phase 2 qui consiste à ajouter un filtre de gauss. Ce dernier va être mis entre le contrôleur VGA et le générateur de pattern.

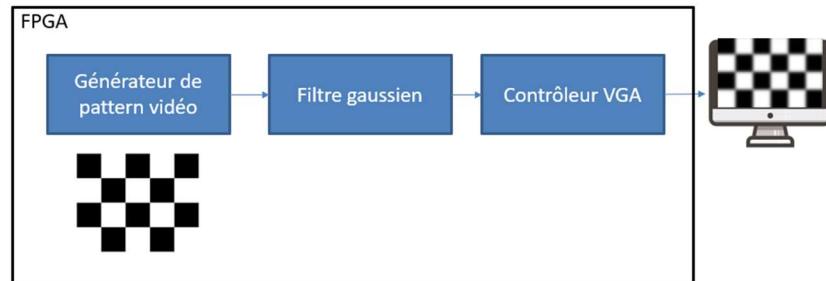
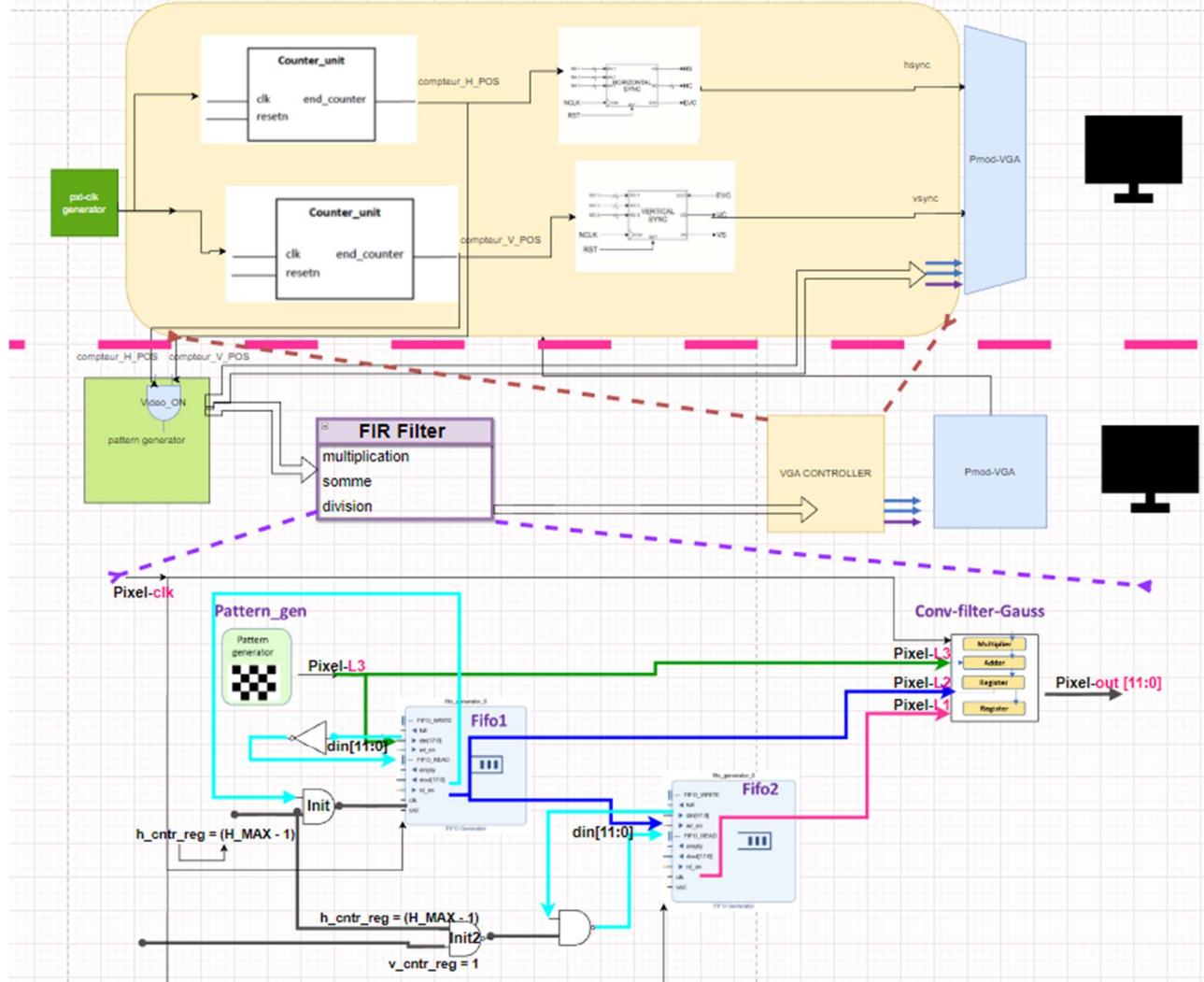


Figure 14: Phase2 -projet

Ce chapitre expliquera en détail la fonctionnalité des différents modules. Pour chaque module, un descriptif et une explication des entrées et sorties sont donnés.

Par la suite, une explication du fonctionnement de ce module est appuyée par un schéma bloc et un schéma RTL.

1) Schéma bloc de contrôleur VGA avec un filtre de gauss (FIR) (Finite Impulse Response)



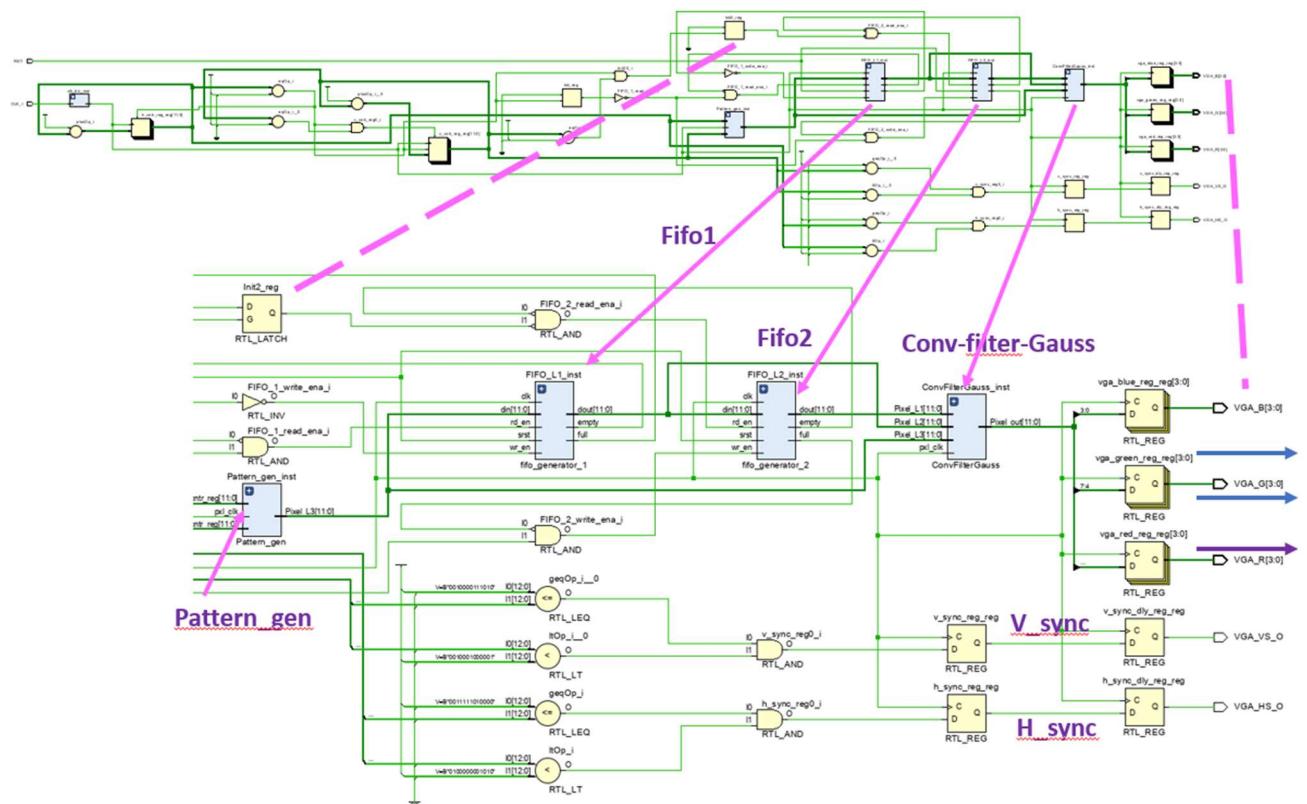
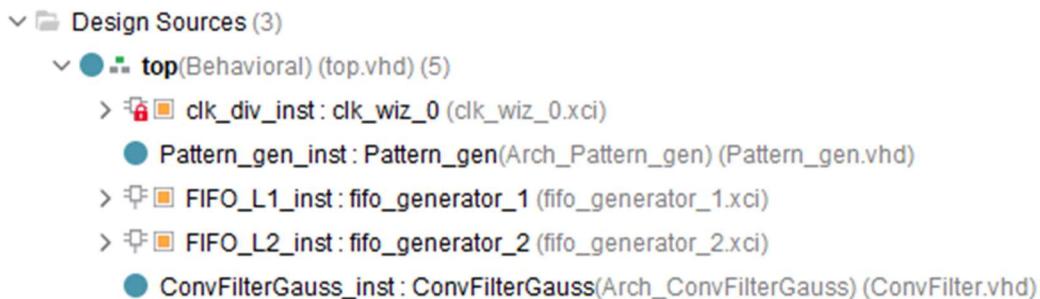


Figure 15 : schéma simplifiant la compréhension de l'architecture globale- phase2 -projet

2) Hiérarchie des différents modules

Cette section permet de situer rapidement chacun des modules dans la hiérarchie du système en partant du module de plus haut niveau qui est le module top et en descendant vers le bas où se trouvent les modules les plus simples. Voici donc cette arborescence des modules :



3) Principe de fonctionnement de filtre de Gauss dans notre projet VGA:

Nous allons faire un filtrage par convolution 2D et nous devrons employer une architecture appelée « sliding window » architecture avec un filtre gaussien spatial :

1	2	1
2	4	2
1	2	1

1/16 x

Figure 16 : Masque filtre gaussien 3x3

On superpose le masque kernel sur le coin gauche de la matrice de l'image, ensuite nous allons multiplier chaque nombre superposé puis additionner le tout et enfin diviser par 16.

- Le filtre a un masque de taille 3×3 .
- Le filtre est composé de 9 registres permettant de stocker les 9 pixels du masque.
- La taille des registres est déterminée par la taille d'un pixel : 12 bits
- Le filtre est composé de 2 FIFO (taille masque-1 soit $3-1=2$ FIFO nécessaires) utilisés pour sauvegarder et sortir dans l'ordre les pixels des lignes précédentes de l'image.
- La taille d'un fifo est calculé ci-après dans le paragraphe suivant.
- Le fonctionnement de cette architecture est présenté ci-après :

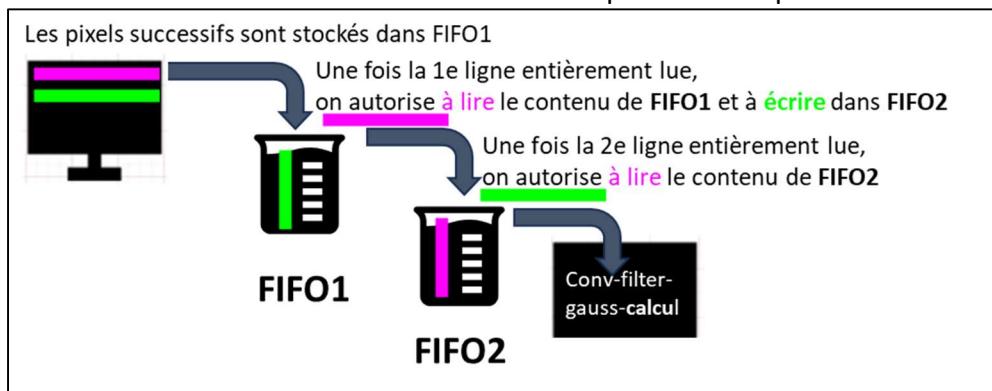
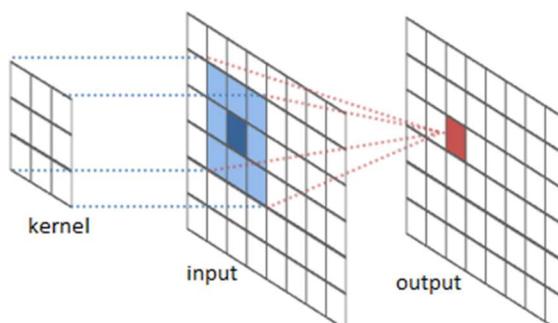


Figure 17 : principe de fonctionnement du filtre gaussien 3x3

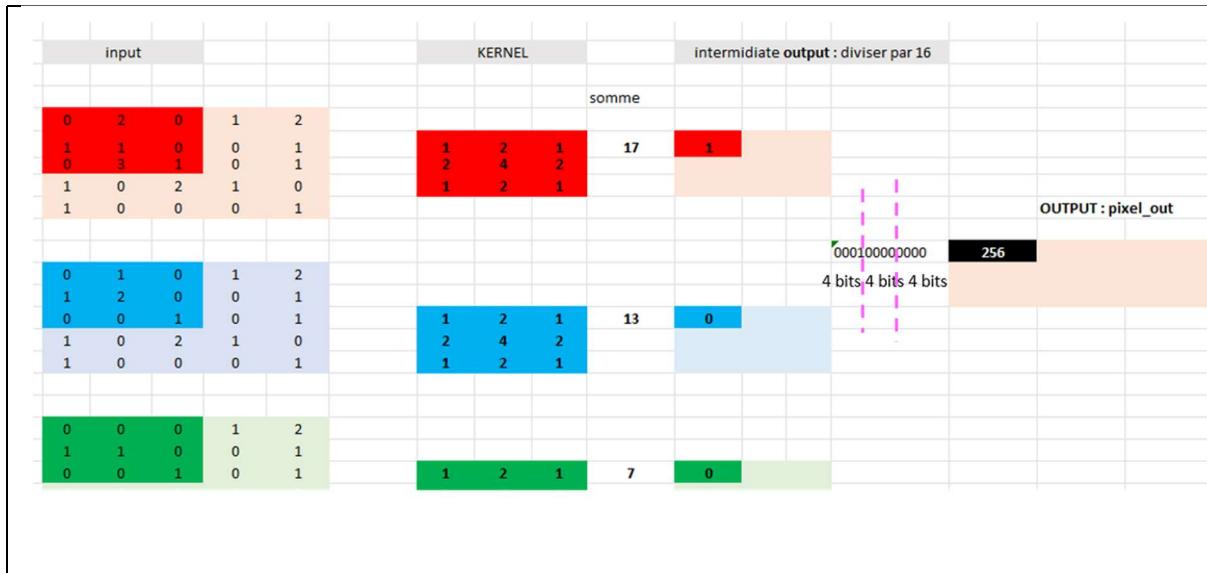
- Les données arrivent dans un premier registre P33 (Pixel_L3).
- Les pixels se décalent ensuite à chaque coup d'horloge à travers les deux FIFO et les registres, jusqu'au remplissage total des registres (de P11 à P33).
- Le masque se déplace ensuite vers la droite dans la ligne horizontale de l'image à chaque coup d'horloge,



Balayage de l'image par le masque de convolution : calcul de la valeur résultat output pour le pixel de sortie (output) dont les coordonnées correspondent au pixel au milieu du masque

COMPTE RENDU DEVELOPPEMENT PROJET INTERFACE VIDEO VGA

Rapport- projet- Eve CHAR - 09/06/2023 – 03/07/2023



Le balayage se poursuit pixel par pixel puis ligne par ligne sur toute l'image.

A noter : l'opération de filtrage par convolution est une somme pondérée, et donc c'est une opération linéaire : les valeurs des pixels de l'image filtrée sont des combinaisons linéaires des valeurs des pixels de l'image d'origine. C'est pourquoi les filtres réalisés par convolution sont appelés filtres linéaires (voir tableau ci-dessous pour comprendre le fonctionnement de notre filtre).

A	B	C	D	E	F	G	H	I	J	K	L
rouge 4 bits	bleu 4 bits	vert 4 bits		Masque Gauss		rouge * gaus	bleu * gauss	vert * gauss	conversion [A & B & C] en decimal	* gauss	
15	15	15		1		15	15	15	4095	4095	
15	15	15		2		30	30	30	4095	8190	
0	0	0		1		0	0	0	0	0	
0	0	0		2		0	0	0	0	0	
15	15	15		4		60	60	60	4095	16380	
15	15	15		2		30	30	30	4095	8190	
0	0	0		1		0	0	0	0	0	
0	0	0		2		0	0	0	0	0	
0	0	0		1		0	0	0	0	0	
				somme puis division par 16 concaténation G&H&I = conversion en 12 bits		8,4375	8,4375	8,4375	somme		36855
1000	1110	0110							2303,4375	somme puis division par 16	2303,4375
									2184		119,4375

La somme G & H & I = concaténation de A &B &C * gauss filter

4) Application du filtre de Gauss

Nous avons un masque kernel 3x3 : K = C1 C2 C3 L1 K11 K12 K13 L2 K21 K22 K23 L3 K31 K32 K33	Nous avons un masque de Gaussien 3x3: 1 2 1 2 4 2 1 2 1	Les pixels utilisés dans le calcul sont nommés C1 C2 C3 L1 P11 P12 P13 L2 P21 P22 P23 L3 P31 P32 P33
--	---	---

Etant donné que tous les coefficients sont des multiples de 2, nous allons utiliser les décalages de bits pour les opérations de multiplication et de division.

A chaque coup d'horloge, on décale de 1 le masque vers la droite, donc les pixels à considérer

vont vers la gauche : P11 <= P12, P21 <= P22, ... P32 <= P33

Les valeurs de P13, P23 et P33 sont les nouvelles valeurs à considérer, donc les inputs de la fonction qui sont Pixel_L1 / Pixel_L2 / Pixel_L3

Pixel_out sera la valeur du pixel à appliquer aux coordonnées du pixel P22 en sortie du filtre.

a. Explication : Fonctionnement des 2 Fifo

- Les pixels successifs sont stockés dans FIFO1
- Une fois la 1e ligne entièrement lue, on autorise à lire le contenu de FIFO1 et à écrire dans FIFO2
- Une fois la 2e ligne entièrement lue, on autorise à lire le contenu de FIFO2

```
Init <= '0' when h_cntr_reg = (H_MAX - 1); --v_cntr_reg = 1;
-- quand la 1e ligne est lue, on autorise à lire dans FIFO1 et écrire dans FIFO2
Init2 <= '0' when (h_cntr_reg = (H_MAX - 1)) and (v_cntr_reg = 1); --v_cntr_reg = 2;
-- quand la 2e ligne est lue, on autorise à lire FIFO2

FIFO_1_write_ena <= not(FIFO_1_full); --on écrit dans FIFO_1 dès le début
FIFO_1_read_ena <= not(FIFO_1_empty) and not(Init);
--on lit dans FIFO_1 si FIFO_1 n'est pas vide et si la première ligne a été entièrement lue (not (Init))
FIFO_2_write_ena <= not(FIFO_2_full) and not(Init);
--on écrit dans FIFO_2 si FIFO_1 n'est pas plein et si la première ligne a été entièrement lue (not (Init))
FIFO_2_read_ena <= not(FIFO_2_empty) and not(Init2);
--on lit dans FIFO_2 si FIFO_2 n'est pas vide et si la 2e ligne a été entièrement lue (not (Init2))
```

- Pixel_L3 est le pixel sortie de pattern_gen. C'est celui qui viendra en bas à droite de notre filtre (P33). C'est aussi celui qui rentre dans FIFO1.
- Pixel_L2 est le pixel de la ligne au-dessus du pixel courant Pixel_L3. C'est aussi le pixel qui viendra au milieu à droite de notre filtre (P23) et également en entrée du FIFO2.
- Pixel_L1 est le pixel de la ligne au-dessus du pixel Pixel_L2 donc 2 lignes au-dessus du pixel courant Pixel_L3. C'est aussi le pixel qui viendra en haut à droite de notre filtre (P13).

```
-- affectation de des pixels dans le masque (on se déplace vers la droite)
P11 <= P12;
P12 <= P13;
P13 <= Pixel_L1;
P21 <= P22;
P22 <= P23;
P23 <= Pixel_L2;
P31 <= P32;
P32 <= P33;
P33 <= Pixel_L3
```

Calcul de la taille nécessaire des registres du filtre Gaussien (sans séparer les couleurs) :

- Nous avons 12 bits par pixel ($4 \times 4 \times 4$)
- Nous souhaitons enregistrer une ligne complète ensuite passer à la ligne suivante.
- Nous avons un poids du masque = 16 et la taille d'un pixel = 12 bits

→ Ainsi la taille minimale du registre à utiliser est = 4 bits + 12 bits = **16 bits** (car 16 en binaire est 2^4)

En séparant les couleurs, nous avons besoin de **3** registres de **8 bits**.

Calcul de la taille nécessaire de la mémoire des fifo du filtre Gaussien :

La mémoire minimale à utiliser dans un FIFO est égale à (largeur image – taille masque) x taille pixel, donc on obtient :

$$\text{Taille mémoire fifo} = (800 - 3) \times 12 \text{ bits} = 9.564 \text{ kbits} = 1.2k \text{ octets}$$

Point d'attention : Gestion des bords

On peut voir que pour les pixels du bord, le masque déborde de l'image. Les valeurs des pixels au-delà du bord sont donc indéfinies, comme l'illustre la figure ci-dessous

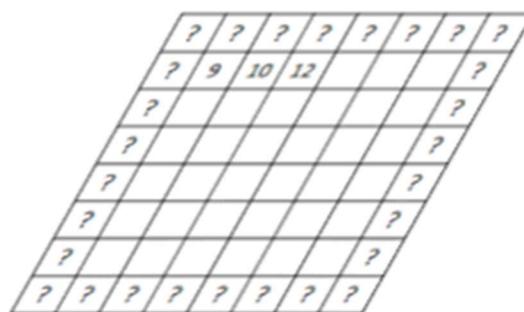


Figure 19 : les pixels du bord du filtre gaussien 3x3

- Pour obtenir des valeurs plausibles, différentes solutions peuvent être adoptées. Nous avons choisi la méthode **Zero-padding** : on suppose que les pixels en dehors de l'image sont de valeur nulle.

b. Latence de l'ensemble du système :

Rappelons que notre pixel à générer a les mêmes coordonnées que le pixel du milieu (P22).

P11	P12	P13
P21	P22	P23
P31	P32	P33

Donc il faut tenir compte que les coordonnées du pixel de sortie sont (h_cntr - 1 ; v_cntr - 1)

Explication :

Exemple, le pixel de l'image du pattern_gen est le (3;3) mais le pixel en sortie du filtre est le (2;2) → il faut décaler v_sync d'une ligne et h_sync d'une colonne. Les signaux h_decal et v_decal sont ajoutés aux calculs de h_sync et v_sync pour assurer ces décalages.

```
-- calcul de h_sync --
-- h_sync = 1 quand (H_FP + FRAME_WIDTH - 1) <= v_cntr_reg < H_FP + FRAME_WIDTH + H_PW - 1
process (pxl_clk)
begin
if (rising_edge(pxl_clk)) then
    if (h_cntr_reg >= (H_FP + FRAME_WIDTH - 1 + h_decal)) and (h_cntr_reg < (H_FP + FRAME_WIDTH + H_PW - 1 + h_decal)) then
        h_sync_reg <= '1';
    else
        h_sync_reg <= '0';
    end if;
end if;
end process;

-- calcul de v_sync --
-- v_sync = 1 quand (V_FP + FRAME_HEIGHT - 1) <= v_cntr_reg < V_FP + FRAME_HEIGHT + V_PW - 1
process (pxl_clk)
begin
if (rising_edge(pxl_clk)) then
    if (v_cntr_reg >= (V_FP + FRAME_HEIGHT - 1 + v_decal)) and (v_cntr_reg < (V_FP + FRAME_HEIGHT + V_PW - 1 + v_decal)) then
        v_sync_reg <= '1';
    else
        v_sync_reg <= '0';
    end if;
end if;
end process;
```

5) Explication du dégradé de niveau de gris dans les bords

Dans notre projet, une couleur est définie par la part qu'elle contient de couleur Rouge, Green (vert) et Bleu. Chaque couleur a donc une valeur pour le rouge, une valeur pour le vert et une valeur pour le bleu (4 bits, 4 bits, 4 bits).

Chacune des trois valeurs est définie par des nombres entre 0 et 15 (4 bits).

- La valeur 0 signifie : aucune part de la couleur fondamentale concernée,
- La valeur 15 signifie : la part maximale de la couleur fondamentale concernée.

En théorie, nous avons une palette de 4096 couleurs au total mais pour notre image (Damier) en noir et blanc, nous avons donc 16 nuances.

Quand R, V et B ont pour valeur 0, la valeur ainsi définie est un noir profond.

Quand les trois parts de couleurs ont pour valeur 15 chacune, c'est un blanc pur qui est défini. Une valeur intermédiaire, par exemple (3,3,3) → 0011, 0011, 0011, définit une couleur grise d'une certaine luminosité.

→ La palette de notre image en niveaux de gris est constituée par les 16 couleurs comprises entre (0,0,0) et (15,15,15) qui y sont sauvegardées - donc **16 nuances de gris.**

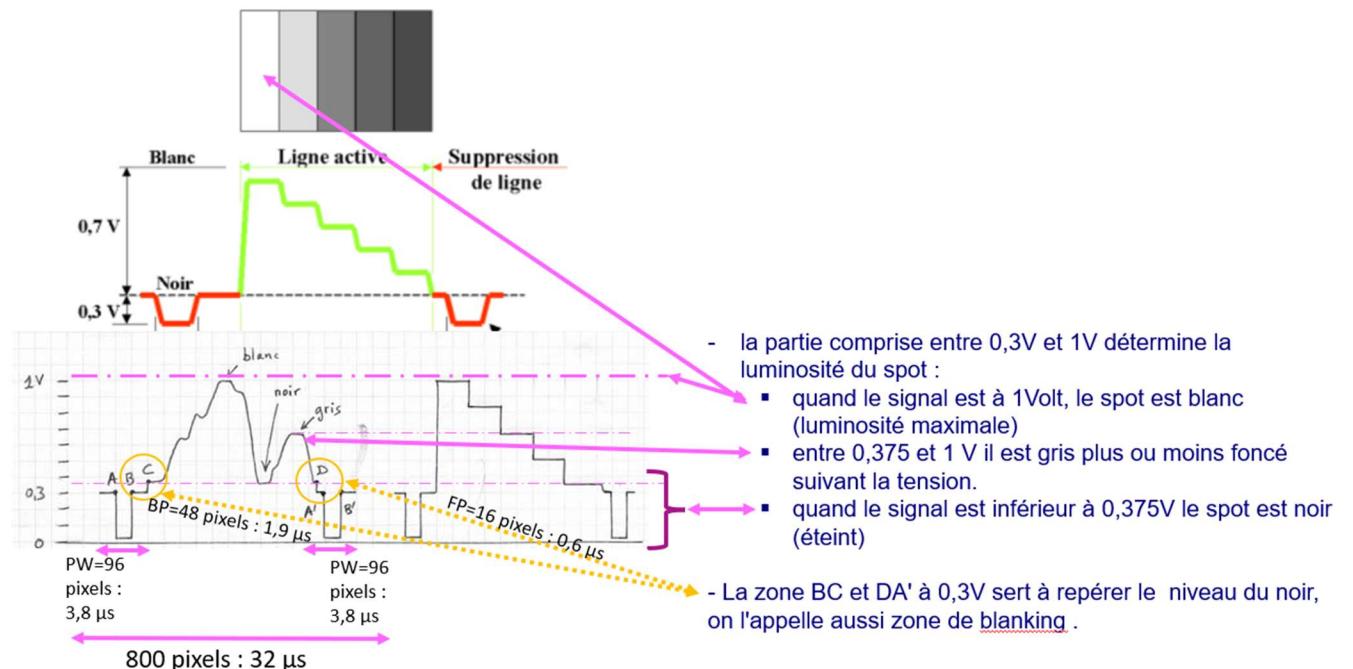


Figure 20 : dégradé de gris

Cette figure représente le signal électrique correspondant à une ligne complète (800 pixels) de l'écran,

- Nous avons deux couleurs noir et blanc qui varient de 0 à 1 Volt
- La partie comprise entre 0,3V et 1V détermine la luminosité du spot :
 - Quand le signal est à 1V, le spot est blanc (luminosité maximale)
 - Entre 0,375 et 1 V il est gris plus ou moins foncé suivant la tension.
 - Quand le signal est inférieur à 0,375V le spot est noir (éteint)
- La zone BC et DA' à 0,3V sert à repérer le niveau du noir, on l'appelle aussi zone de blanking.
- La ligne représentée à droite, dite en escalier, commence par une plage blanche puis par un gris clair puis par une plus foncée, pour arriver jusqu'au noir ; si toutes les lignes de l'image sont identiques à celle-ci, on obtient une mire de barres.

IV. Synthèse et analyse des tests

- Les procédures de test sont dans un document joint (plan de validation)
- Les résultats de ces tests sont présentés dans un autre document (cahier de recette) dont le tableau ci-dessous résume les résultats obtenus.

1) Synthèse des tests

Statut au 27/06/2023 :

	n° test	test	Résultat du test
phase 1 test-bench	1	Vérification de la génération de la « pixel_clock »	pass
	2	Vérification de la synchronisation horizontale	pass
	3	Vérification de la synchronisation verticale	pass
	4	Vérification de la génération des signaux	pass
	5	Vérification de la position pixel sur la ligne horizontale	non effectué
	6	Vérification de la position pixel sur la ligne Verticale	non effectué
	7	Vérification de la zone visible de l'écran	pass
	8	Validation globale de la phase 1 « contrôleur_VGA » sans filtre	pass
phase 2 test-bench	9	test des entrées / sorties du fifo1 (lecture- écriture)	pass
	10	test des entrées / sorties du fifo2 (lecture- écriture)	pass
	11	test des entrées / sorties du Conv-filter-Gauss (calcul)	pass
	12	de la synchronisation du système après l'ajout du filtre	pass
	13	la dégradation de gris dans les bords après l'ajout du filtre	pass
	14	Validation globale de la phase 2 « contrôleur_VGA » avec filtre	pass
ILA	15	sur cible - vérification de la synchronisation horizontale	pass
	16	sur cible - vérification de la synchronisation verticale	pass
	17	sur cible - Vérification de la génération des signaux, sortie de calcul et sorties des FIFOs	pass
	18	sur cible - Vérification des entrées / sorties du Conv-filter-Gauss (calcul)	pass
	19	sur cible - Vérification du dégradée du gris sur ILA	pass
oscilloscope	20	sur cible - oscilloscope - Mesure du Temps entre 2 lignes	pass
	21	sur cible - oscilloscope - Mesure du Temps de H_sync	pass
	22	sur cible - oscilloscope - Mesure pulse H_PW	pass
	23	sur cible - oscilloscope - Mesure de V-SYNC	pass
	24	sur cible - oscilloscope - Mesure de H-SYNC	pass
	25	sur cible - oscilloscope - Mesure de dégradé du gris avec V-SYNC	pass
	26	sur cible - oscilloscope - Mesure de dégradé du gris avec H-SYNC	pass

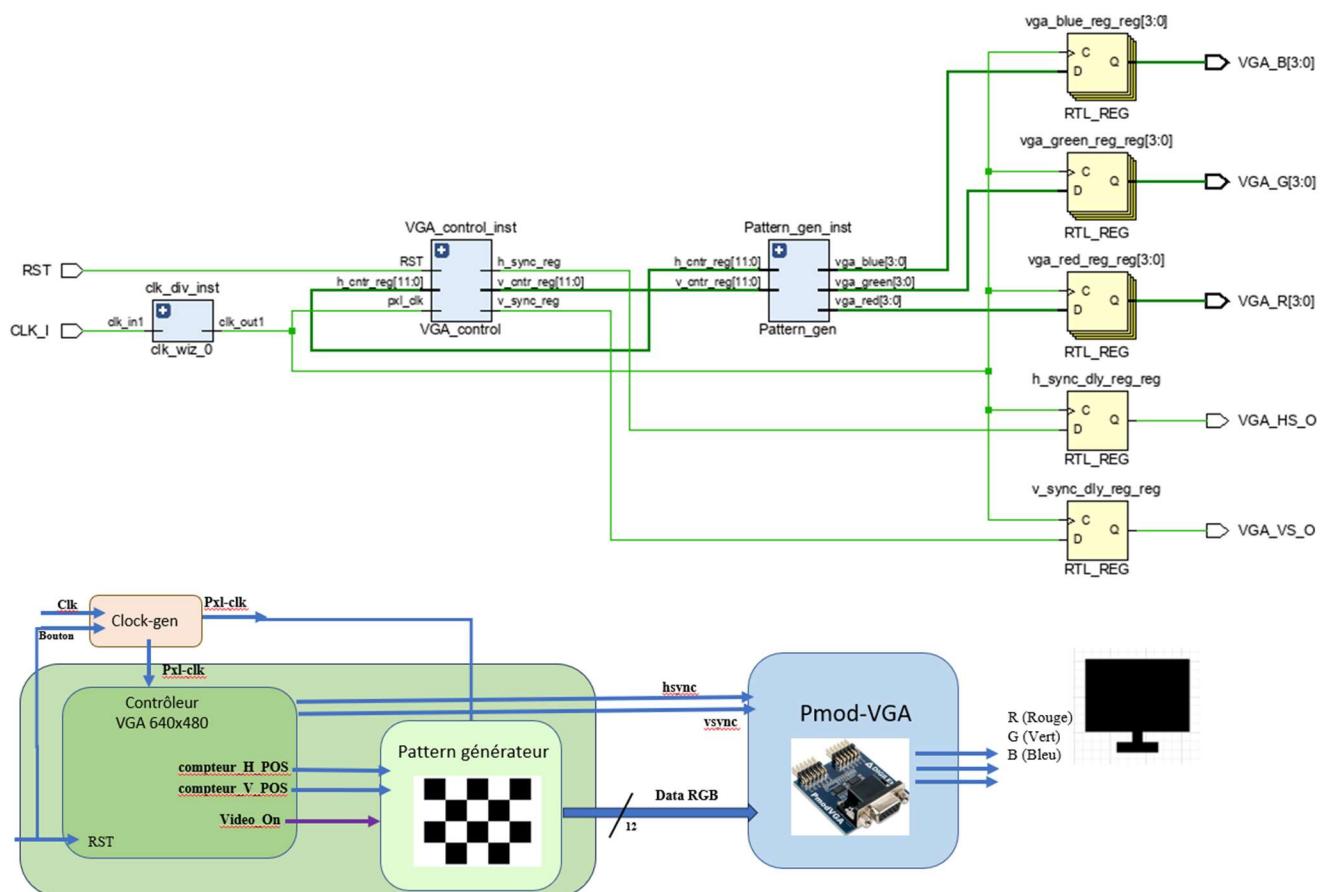
ID exigence	test bench	ILA	oscilloscope	observation écran (TBD)
clock_gen_01	test 1 : pass			test 14: pass test 8 : pass
timing_gen_02	test 4 : pass test 11 : pass	test 17 : pass		test 14: pass test 8 : pass
horizontal_timing_gen_03	test 2 : pass test 12 : pass	test 15: pass	test 24: pass test 26 : pass test 22: pass test 21: pass	test 14: pass test 8 : pass
vertical_timing_gen_04	test 3 : pass test 12 : pass	test 16: pass	test 25 : pass test 23: pass test 20: pass	test 14: pass test 8 : pass
compteur_H_POS_05	test 5 : TBD			
compteur_V_POS_06	test 6 : TBD			
display_area_07	test 7 : pass			test 14: pass test 8 : pass
FIR_Filter_08	test 8 : pass test 9 : pass test 10 : pass	test 16 : pass test 17 : pass test 18 : pass test 19: pass	test 25 : pass test 26: pass	

Figure 21 : Matrice des tests par exigences

2) Analyses des résultats

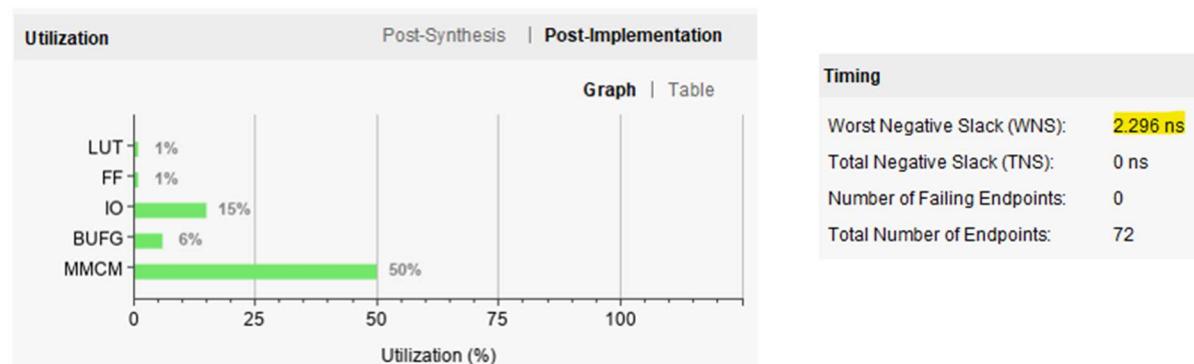
a. Phase 1 du projet

Schéma RTL :



Analyses des ressources élémentaires utilisés :

Report Cell Usage:		
	Cell	Count
Start RTL Component Statistics	1 clk_wiz_0_bbox 1	
Detailed RTL Component Info :	2 CARRY4 16	
+---Registers :	3 LUT1 5	
4 Bit Registers := 3	4 LUT2 41	
1 Bit Registers := 4	5 LUT4 5	
+---Muxes :	6 LUT5 1	
2 Input 4 Bit Muxes := 1	7 LUT6 2	
	8 FDRE 30	
Finished RTL Component Statistics	9 OBUF 14	

Analyses de timing et du Chemin critique :

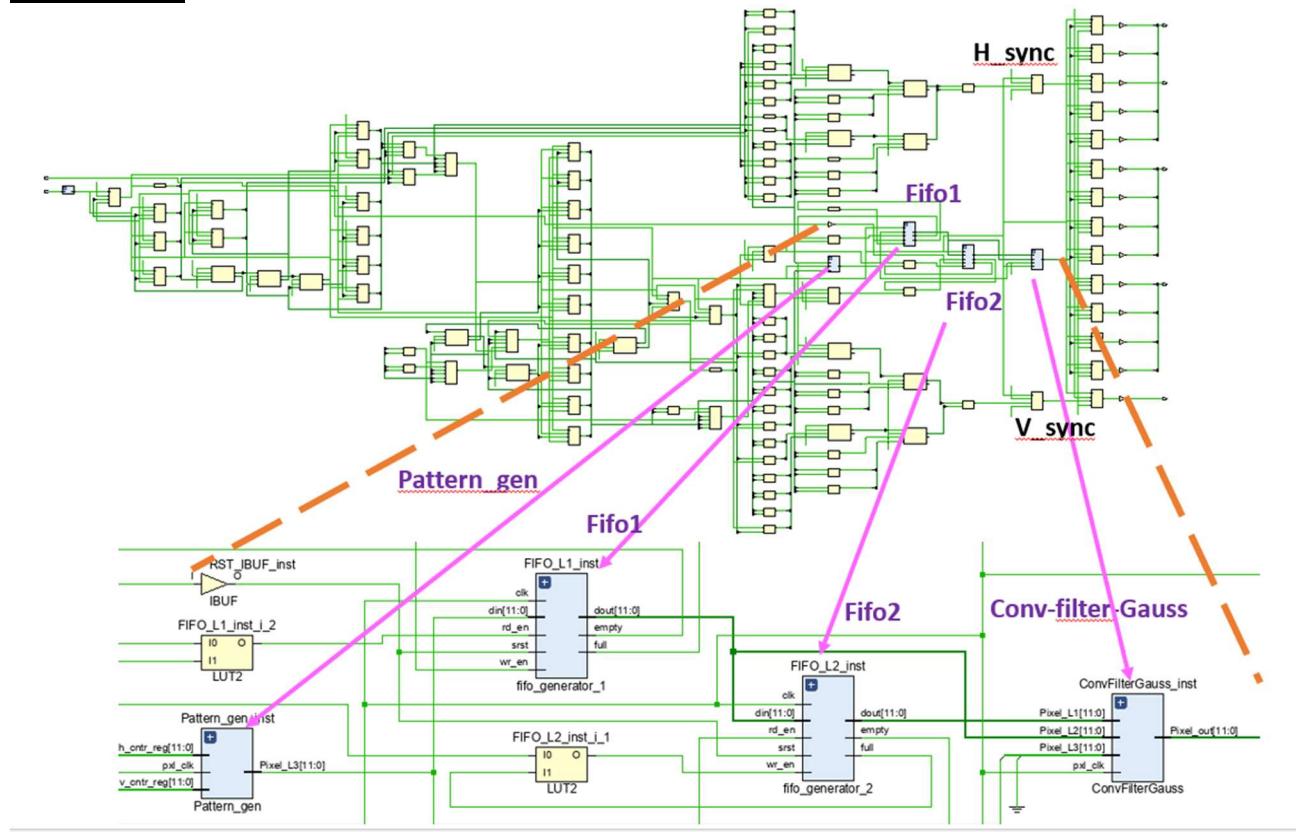
Pas de violation de timing (TNS= 0)

```
| Design Timing Summary
| -----
```

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints
2.296	0.000	0	72	0.115	0.000	0	72

Le chemin le plus long dure : 2.29 ns

```
Max Delay Paths
-----
Slack (MET) : 2.296ns (required time - arrival time)
Source: VGA_control_inst/h_cntr_reg[8]/C
        (rising edge-triggered cell FDRE clocked by clk_outl_clk_wiz_0 (rise@0.000ns fall@3.367ns period=6.734ns))
Destination: vga_green_reg_reg[3]_lopt_replica_2/D
        (rising edge-triggered cell FDRE clocked by clk_outl_clk_wiz_0 (rise@0.000ns fall@3.367ns period=6.734ns))
Path Group: clk_outl_clk_wiz_0
Path Type: Setup (Max at Slow Process Corner)
Requirement: 6.734ns (clk_outl_clk_wiz_0 rise@6.734ns - clk_outl_clk_wiz_0 rise@0.000ns)
Data Path Delay: 4.041ns (logic 1.490ns (36.873%) route 2.551ns (63.127%))
Logic Levels: 3 (CARRY4=1 LUT2=1 LUT4=1)
Clock Path Skew: -0.020ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD): -1.294ns = ( 5.440 - 6.734 )
    Source Clock Delay (SCD): -0.648ns
    Clock Pessimism Removal (CPR): 0.626ns
Clock Uncertainty: 0.114ns ((TSJ^2 + DJ^2)^1/2) / 2 + PE
Total System Jitter (TSJ): 0.071ns
Discrete Jitter (DJ): 0.217ns
Phase Error (PE): 0.000ns
```

b. Phase 2 du projet**Schéma RTL :****Analyses des ressources élémentaires utilisés :**

```

Start RTL Component Statistics
-----
Detailed RTL Component Info :
+-- Adders :
    9 Input   8 Bit      Adders := 3
+-- Registers :
    12 Bit   Registers := 17
    8 Bit    Registers := 3
    4 Bit    Registers := 6
    1 Bit    Registers := 4
-----
Finished RTL Component Statistics

```

Report Cell Usage:

Cell	Count
1 clk_wiz_0_bbox	1
2 fifo_generator_1_bbox	1
3 fifo_generator_2_bbox	1
4 CARRY4	23
5 LUT1	8
6 LUT2	62
7 LUT3	15
8 LUT4	21
9 LUT5	27
10 LUT6	53
11 ISRL16E	1
12 FDRE	152
13 LD	2
14 IBUF	1
15 OBUF	14

Analyses de timing et du Chemin critique :

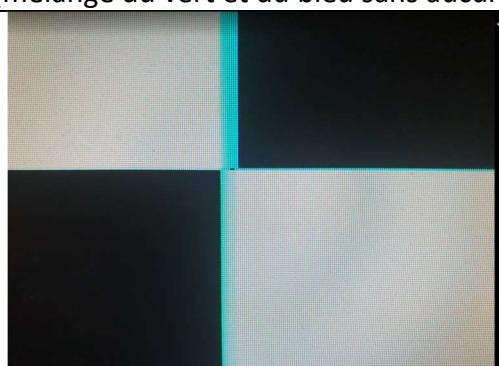
Design Timing Summary						
	WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)
	-0.307	-1.034	11	1222	0.050	0.000
Intra Clock Table						
Clock						
CLK_I						
clk_outl_clk_wiz_0	-0.307	-1.034	11	1222	0.164	0.000
clkfbout_clk_wiz_0						
From Clock: clk_outl_clk_wiz_0						
To Clock: clk_outl_clk_wiz_0						
Setup :	11	Failing Endpoints, Worst Slack	-0.307ns, Total Violation	-1.034ns		
Hold :	0	Failing Endpoints, Worst Slack	0.164ns, Total Violation	0.000ns		
PW :	0	Failing Endpoints, Worst Slack	2.867ns, Total Violation	0.000ns		

Le chemin le plus long dur : -0.307 ns du à l'horloge (148.5 Mhz)

Max Delay Paths	
Slack (VIOLATED) :	-0.307ns (required time - arrival time)
Source:	FIFO_L1_inst/U0/inst_fifo_gen/gconvfifo_rf/grf_rf/gntv_or_sync_fifo.mem/gbm_gbmg_gbmg_ngecc_bmg/inst_blk_mem_gen (rising edge-triggered cell RAMB36E1 clocked by clk_outl_clk_wiz_0 (rise@0.000ns fall@3.367ns period=6.734ns))
Destination:	FIFO_L2_inst/U0/inst_fifo_gen/gconvfifo_rf/grf_rf/gntv_or_sync_fifo.mem/gbm_gbmg_gbmg_ngecc_bmg/inst_blk_mem_gen (rising edge-triggered cell RAMB36E1 clocked by clk_outl_clk_wiz_0 (rise@0.000ns fall@3.367ns period=6.734ns))

c. Leçons retenues et capitalisation :**1) Les bords des carrés du damier après application du filtre**

Une couleur bleu turquoise est apparue au niveau des bords après application du filtre (mélange du vert et du bleu sans aucune composante rouge),



La première méthode de calcul de convolution (**sans séparer** les composantes R G B du pixel) a généré un effet de bord bleu comme le montre la figure à gauche sur l'écran. C'est le résultat des erreurs d'arrondis qui se sont additionnés dans notre calcul. Autrement dit, il s'agit d'harmoniques qui se sont ajoutés au calcul.

Voici la première méthode de calcul de convolution utilisée :

- **Multiplication** par le masque de KERNEL $\rightarrow (2^n = \text{décalage à gauche de } n \text{ bits} = \text{ajouter des bits '0' à droite} = \text{concaténer PXY par '0'})$
- **Somme** des ‘Pixel_out_somme’ $\leq ("0000" \& \mathbf{P11}) + ("000" \& \mathbf{P12} \& '0') + ("0000" \& \mathbf{P13}) + ("000" \& \mathbf{P21} \& '0') + ("00" \& \mathbf{P22} \& "00") + ("000" \& \mathbf{P23} \& '0') + ("0000" \& \mathbf{P31}) + ("000" \& \mathbf{P32} \& '0') + ("0000" \& \mathbf{P33})$
- **Division** par le poids du masque KERNEL (division par 16, donc par 2^4 donc décalage de 4 vers la droite) :

$$\text{Pixel_out} \leq \text{Pixel_out_somme}(15 \text{ downto } 4)$$

En séparant les composantes rouge, vert et bleu (en imaginant 3 matrices successives d'images avec 3 couleurs séparées) et en séparant les étapes du calcul (multiplication, somme et divisons pour chaque matrice de couleur), nous obtenons un dégradé de gris entre le blanc et le noir, ce qui est le résultat attendu.

Voici la deuxième méthode de calcul de convolution en séparant les composantes RGB:

```
-- Application du filtre à chaque composante R G B de manière distincte
-- Implique de prendre la partie rouge puis vert puis bleu
-- Pixel rouge
Pixel_out_somme_rouge <= ("0000" & P11(11 downto 8)) +
("000" & P12(11 downto 8) & "0") + ("0000" & P13(11 downto 8)) +
("000" & P21(11 downto 8) & "0") + ("00" & P22(11 downto 8) & "00") +
("000" & P23(11 downto 8) & "0") + ("0000" & P31(11 downto 8)) +
("000" & P32(11 downto 8) & "0") + ("0000" & P33(11 downto 8));
Pixel_out_rouge <= Pixel_out_somme_rouge(7 downto 4);
-- Pixel vert
Pixel_out_somme_vert <= ("0000" & P11(7 downto 4)) +
("000" & P12(7 downto 4) & "0") + ("0000" & P13(7 downto 4)) +
("000" & P21(7 downto 4) & "0") + ("00" & P22(7 downto 4) & "00") +
("000" & P23(7 downto 4) & "0") + ("0000" & P31(7 downto 4)) +
("000" & P32(7 downto 4) & "0") + ("0000" & P33(7 downto 4));
Pixel_out_vert <= Pixel_out_somme_vert(7 downto 4);
-- Pixel bleu
Pixel_out_somme_bleu <= ("0000" & P11(3 downto 0)) +
("000" & P12(3 downto 0) & "0") + ("0000" & P13(3 downto 0)) +
("000" & P21(3 downto 0) & "0") + ("00" & P22(3 downto 0) & "00") +
("000" & P23(3 downto 0) & "0") + ("0000" & P31(3 downto 0)) +
("000" & P32(3 downto 0) & "0") + ("0000" & P33(3 downto 0));
Pixel_out_bleu <= Pixel_out_somme_bleu(7 downto 4);
```

2) Synchronisation des signaux :

Comme nous sommes dans le cas d'un affichage en temps réel à l'écran LCD. La synchronisation des signaux HSYNC et VSYNC avec le reste des composants dans le système affichage (pattern-gen et/ou filtre...) est très importante dans l'objectif de voir une image nette et non décalée à l'écran. Pour cela il a fallu rajouter des registres h-decal et v-decal pour compenser ce retard de cycles d'horloge. Voir § III 4) b).

3) L'utilisation du reset global « RST »

Nous avons décidé d'ajouter un reset dans notre implémentation, connecté au bouton 0 de la carte. Pour fonctionner correctement, RST doit

- Remettre à 0 les compteurs
- Remettre à 0 les FIFOs
- Remettre à la valeur initiale ('1') les variables « Init » servant à piloter les FIFOs

```
Init <=
'1' when RST = '1' else
'0' when h_cntr_reg = (H_MAX - 1); --v_cntr_reg = 1; -- quand la 1e ligne est lue, on autorise à lire dans FIFO1 et écrire dans FIFO2

Init2 <=
'1' when RST = '1' else
'0' when ((h_cntr_reg = (H_MAX - 1)) and (v_cntr_reg = 1)); --v_cntr_reg = 2; -- quand la 2e ligne est lue, on autorise à lire FIFO2
```

V. Conclusion

Au cours de ce projet, nous nous sommes intéressés à la transmission vidéo via une conversion numérique analogique. Le but est d'afficher une image sur un écran puis d'appliquer un filtre FIR (à Réponse Impulsionnelle Finie) et observer les effets sur l'image suite à l'application du filtre et la conversion numérique analogique.

Les livrables pour ce projet sont une conception fonctionnelle avec son plan de validation associé et ses résultats de test.

La réalisation de ce projet a nécessité d'atteindre différents objectifs :

- Le premier était de maîtriser les outils de conceptions du FPGA et de comprendre l'architecture de ce type de circuit. Il s'agit de la phase 1 : étape intermédiaire du contrôleur VGA simple sans filtre.
- Le deuxième objectif a été d'intégrer un filtre à réponse impulsionnelle finie et de comprendre son mode de fonctionnement. Ensuite d'assurer sa mise en œuvre : architecture, intégration dans le projet contrôleur VGA et par la suite son implémentation. A souligner que, afin d'optimiser cette conception, il a fallu mettre en pratique de nombreuses théories académiques (convolution et TFF...).
- Enfin le dernier objectif était de finaliser le plan de validation : expliquer le « quoi » et le « comment » faire pour valider les différents aspects fonctionnels du projet et vérifier en parallèle si nous sommes toujours conformes à la spécification en entrée de ce projet.

A noter aussi que mes différentes simulations ont soulevés de nombreux dysfonctionnements, ce qui m'a permis de recorriger et réadapter quelques détails (taille pour le filtre en fonction du poids du masque de KERNEL, séparer les calculs des couleurs pour une meilleur troncature des arrondis, la prise en compte des temps de latences dû aux FIFOs et donc de synchroniser les HSYNC et VSYNC pour un affichage correct et non décalé dans la zone active de l'écran...)

VI. Bibliographie - Webographie

VGA Configuration Algorithm using VHDL (1Christian Plaza, 2Olga Ramos, 3Dario Amaya)
Virtual Applications Group-GAV, Nueva Granada Military University –UMNG
Bogotá, Colombia

HAL Id:dumas-00574220

<https://dumas.ccsd.cnrs.fr/dumas-00574220>

Programmation et utilisation du FPGA pour la validation et la vérification de circuits
électroniques

Johanna Mariani

HAL Id:tel-00708233

<https://theses.hal.science/tel-00708233>

THESE Méthodes et algorithmes de dématricage et de filtrage du bruit pour la photographie
numérique

Xilinx, ‘Vivado Design Suite Tutorial : Embedded Processor Hardware Design’, User
Guide UG940, v2014 April 2014,

https://www.xilinx.com/support/documentation/sw_manuals_j/xilinx2014_1/ug940-vivadotutorial-embedded-design.pdf

Xilinx, ‘All Programmable SoC with Hardware and Software Programability’,

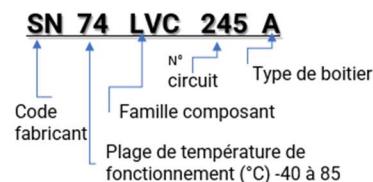
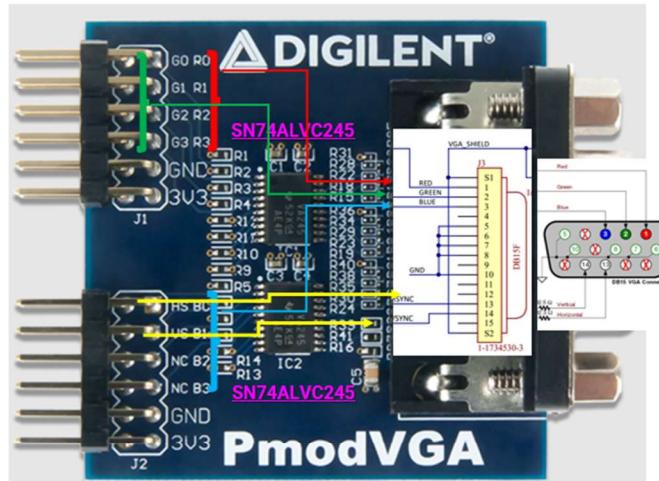
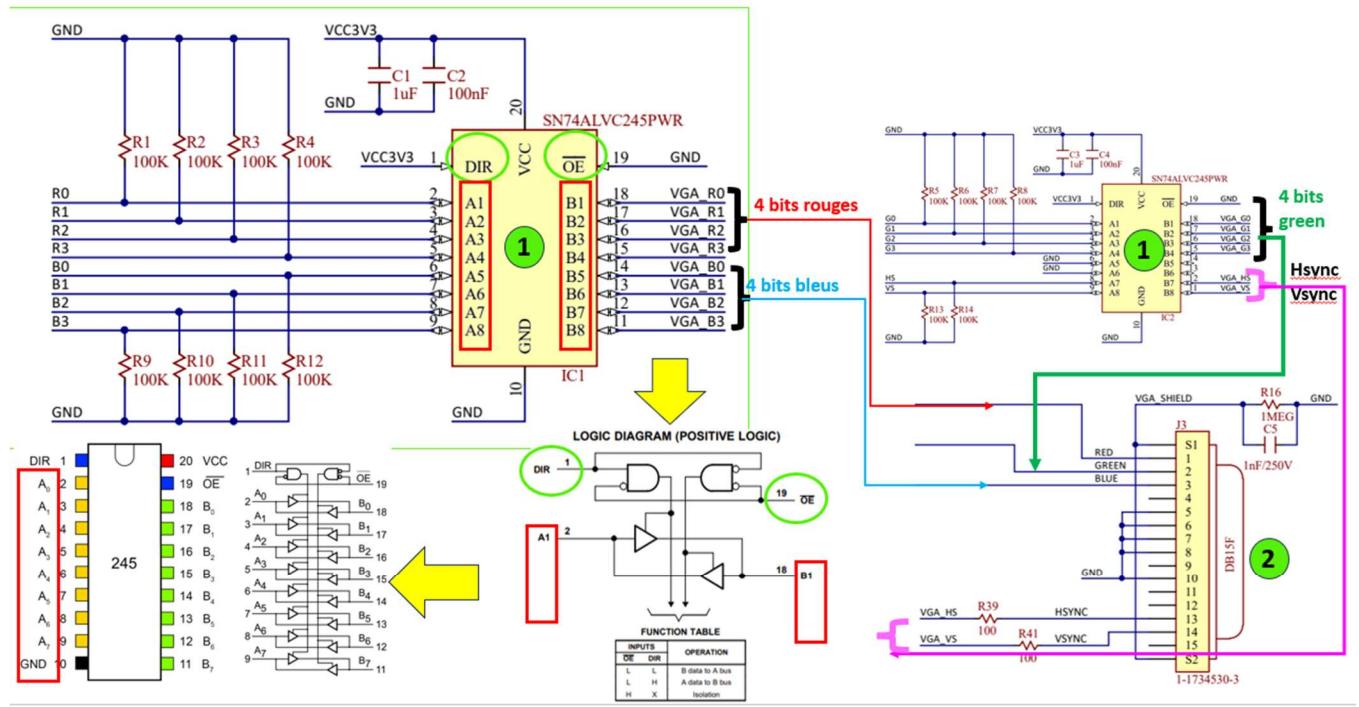
<https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>

http://meteosat.pessac.free.fr/Cd_elect/www.supelec

https://wiki.osdev.org/Video_Signals_And_Timing

ANNEXE

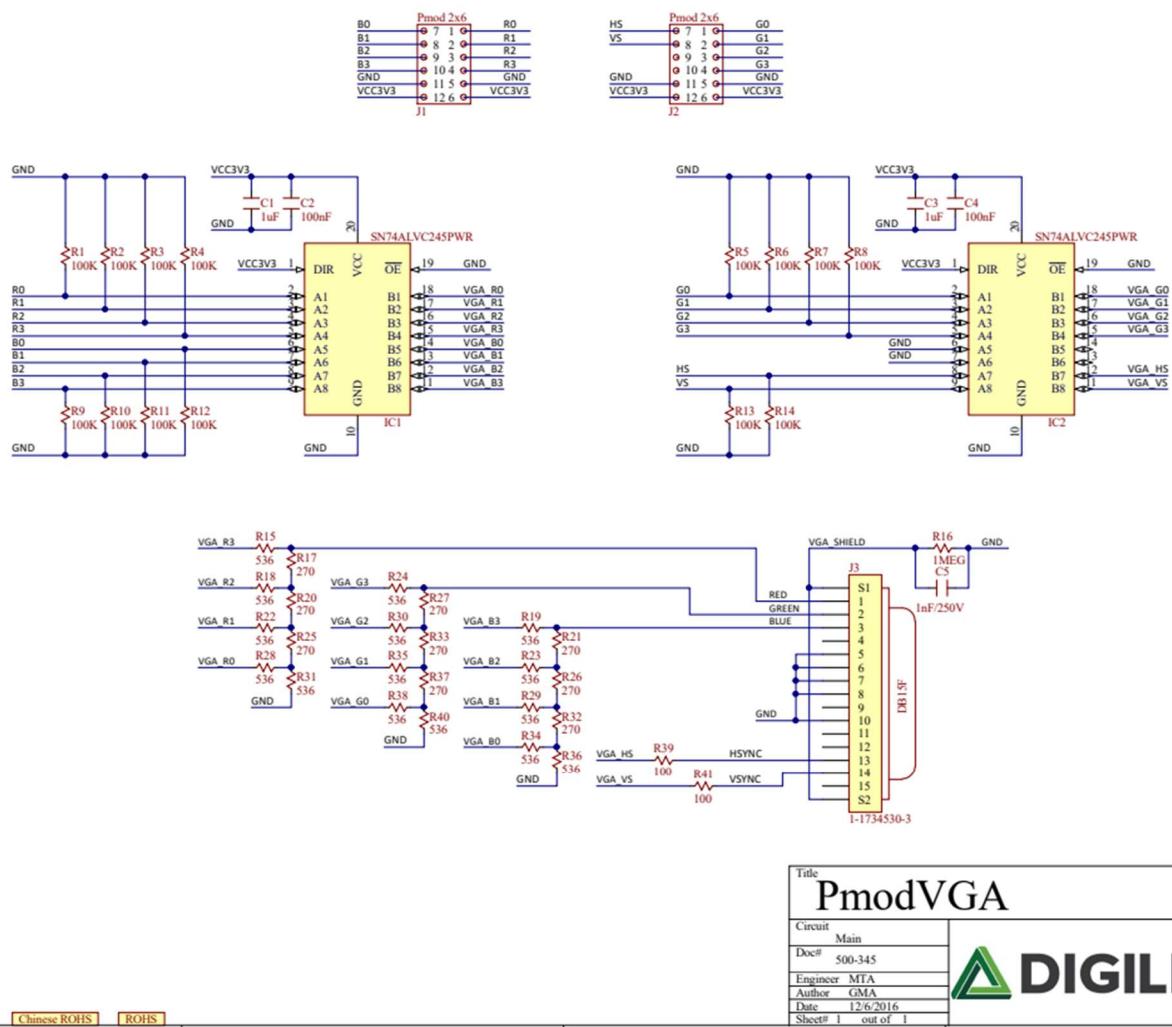
Annexe 1 : Analyse de la datasheet du composant SN74ALVC245 présent sur le PmodVGA



COMPTE RENDU DEVELOPPEMENT PROJET INTERFACE VIDEO VGA

Rapport- projet- Eve CHAR - 09/06/2023 – 03/07/2023

Annexe 2 : Datasheet du PmodVGA



Source :

https://digilent.com/reference/media/reference/pmod/pmodvga/pmodvga_sch.pdf

Annexe 3 : Fonction de transfert dans le domaine fréquentiel du filtre FIR (à Réponse Impulsionnelle Finie)

Soit la fonction de transfert :

$$H(z) = \sum_{k=0}^4 a_k z^{-k}$$

Alors

$$H(z) = a_2 \cdot e^{-2Tp} \cdot \left[\frac{a_0}{a_2} \cdot e^{+2Tp} + \frac{a_1}{a_2} \cdot e^{+Tp} + 1 + \frac{a_3}{a_2} \cdot e^{-2Tp} + \frac{a_4}{a_2} \cdot e^{-2Tp} \right]$$

Par définition:

$$z = e^{Tp} = e^{j\omega T}$$

Avec la symétrie des coefficients on a :

$a_0=a_4$ et $a_1=a_3$ alors

$$H(z) = a_2 \cdot e^{-2Tp} \cdot \left[1 + \frac{2a_0}{a_2} \cdot \cos(2T\omega) + \frac{2a_1}{a_2} \cdot \cos(T\omega) \right]$$

La mise en forme donne

$$H(z) = 20 \cdot \log_{10}(a_2) + 20 \cdot \log_{10} \left[1 + \frac{2a_0}{a_2} \cdot \cos \left(\frac{4\pi f}{F_S} \right) + \frac{2a_1}{a_2} \cdot \cos \left(\frac{2\pi f}{F_S} \right) \right]$$

Avec application numérique, on obtient :

Soit $A=[245 -1952 11620 -1952 245]$ les coefficients

$$H(z) = 81.3dB + 20 \cdot \log_{10} \left[1 + 4.21e^{-2} \cos \left(\frac{4\pi f}{F_S} \right) - 3.36e^{-1} \cdot \cos \left(\frac{2\pi f}{F_S} \right) \right]$$

Annexe 4 : Fonction de transfert dans le domaine fréquentiel du filtre passe bas

Nous avons :

$$H(z) = \sum_{k=0}^{42} b_k z^{-k}$$

Avec $b_{2i+1}=0$ et $b_{2i}=b_{42-2i}$ quelque soit i .

Alors :

$$H(z) = b_{21} \cdot e^{-21Tp} \cdot \left[\frac{b_0}{b_{21}} \cdot e^{+21Tp} + \frac{b_2}{b_{21}} \cdot e^{+19Tp} + \dots + 1 + \frac{b_{22}}{b_{21}} \cdot e^{-Tp} + \dots + \frac{b_{40}}{b_{21}} \cdot e^{-19Tp} + \frac{b_{42}}{b_{21}} \cdot e^{-21Tp} \right]$$

La fonction de transfert est tel que :

$$H(z) = b_{21} \cdot e^{-21Tp} \cdot \left[1 + \frac{2b_{20}}{b_{21}} \cdot \cos(\omega T) + \frac{2b_{18}}{b_{21}} \cdot \cos(3\omega T) + \dots + \frac{2b_{22}}{b_{21}} \cos(19\omega T) + \frac{2b_0}{b_{21}} \cdot \cos(21\omega T) \right]$$

La mise en forme donne :

$$H(z) = 20 \cdot \log_{10}(b_{21}) + 20 \cdot \log_{10} \left[1 + \frac{2b_{20}}{b_{21}} \cdot \cos \left(\frac{2\pi f}{F_S} \right) + \dots + \frac{2b_0}{b_{21}} \cdot \cos \left(21 \frac{2\pi f}{F_S} \right) \right]$$

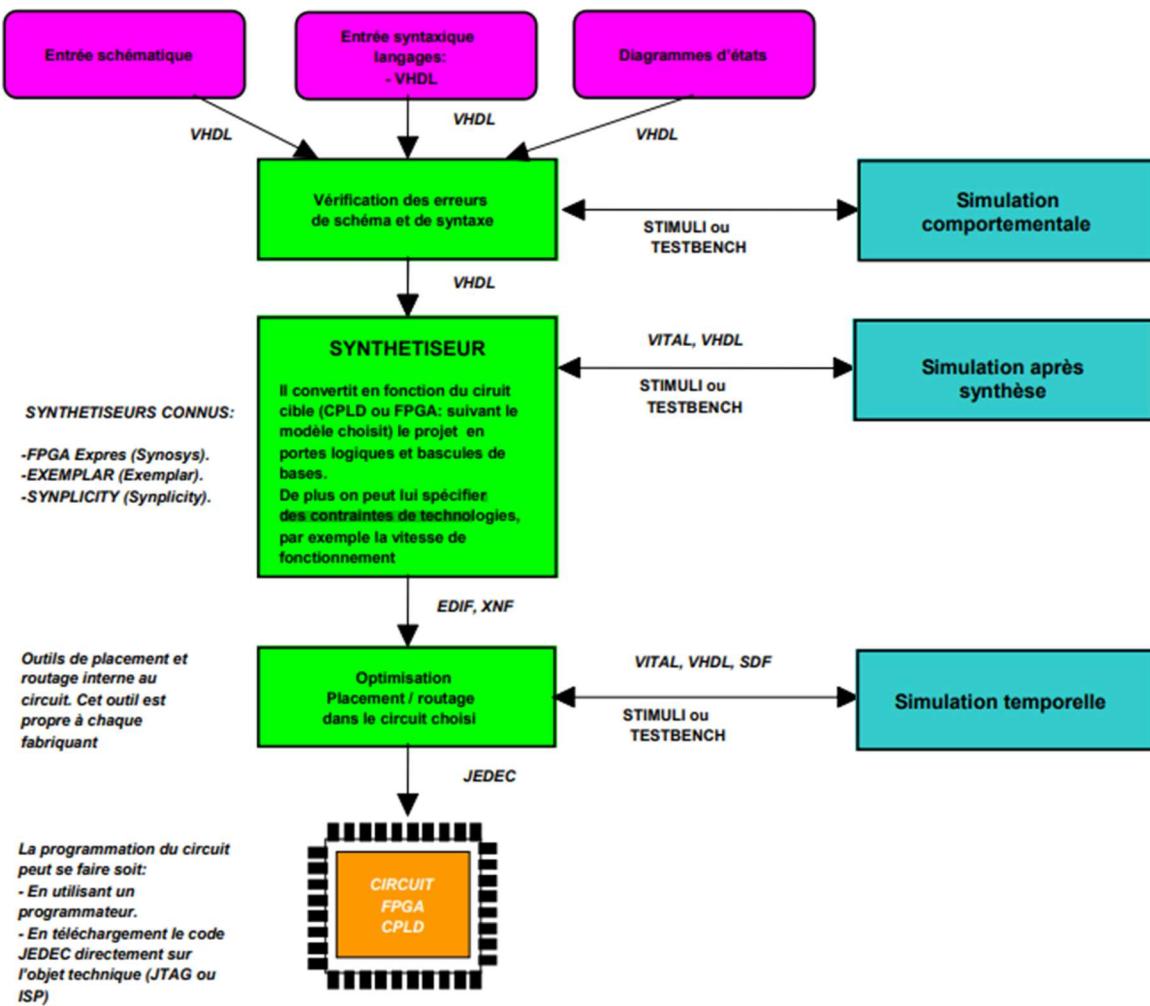
Avec

l'application numérique :

$$H(z) = 72.26dB + 20 \cdot \log_{10} \left[1 + 1.26 \cos \left(\frac{\pi f}{48k} \right) + \dots + 3.42e^{-3} \cos \left(21 \frac{\pi f}{48k} \right) \right]$$

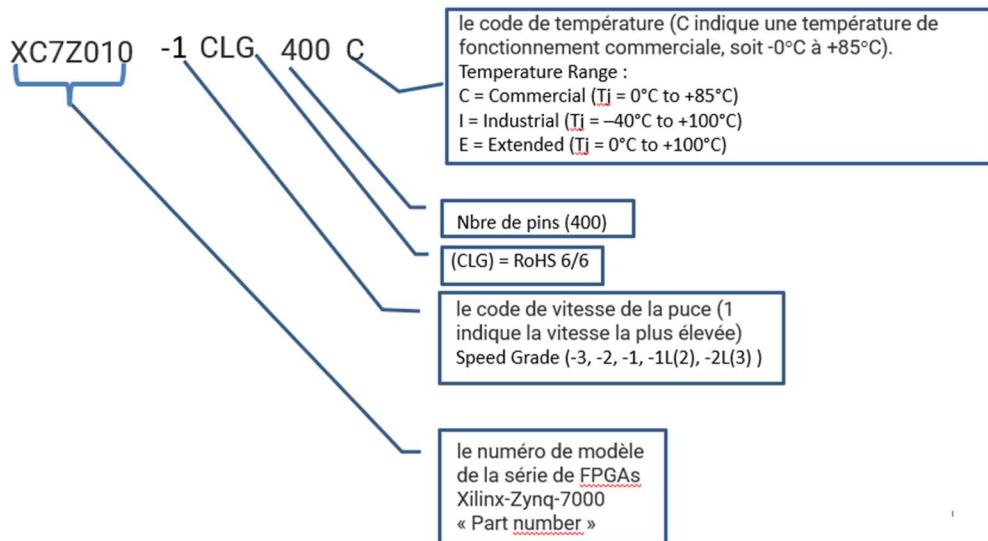
Les 45 coefficients = [7 0 -14 0 27 0 -48 0 78 0 -123 0 188 0 -288 0 458 0 -830 0 2594 4096 2594 0 -830 0 458 0 -288 0 188 0 -123 0 78 0 -48 0 27 0 -14 0 7]

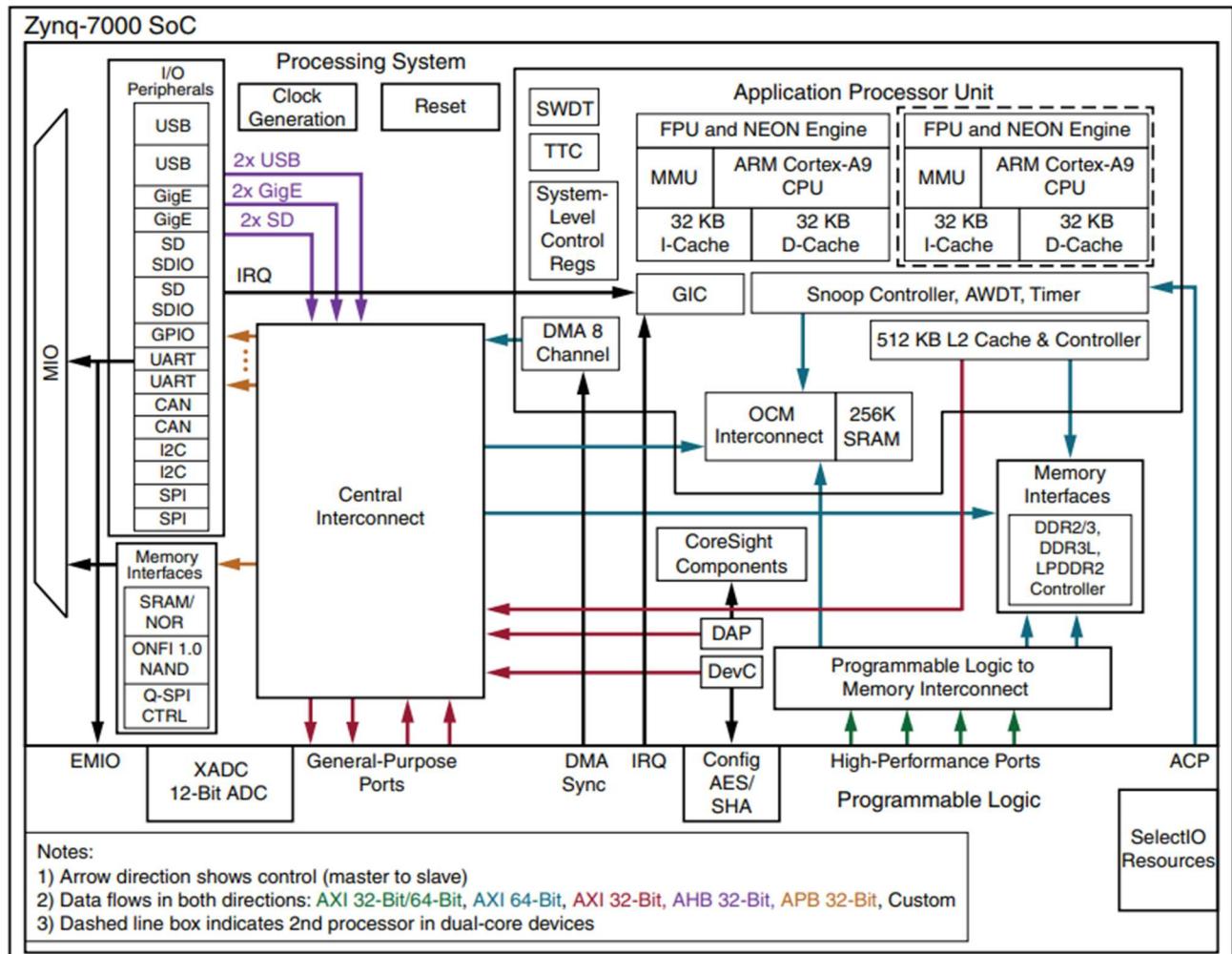
Annexe 5 : Méthodologie de programmation FPGA



Annexe 6 : Description carte Cora Xilinx Zynq 7000

Xc7z010-1CLG400C





Architectural CORA-ZYNQ7000-Overview