01/06/2023

Interface VGA

Rapport de validation (Intermédiaire)

Atmani Hicham & Kamal Kherchouch
FORMATION SAFRAN CHEZ AJC

Date	Version	Remarque	Auteur
15/06/2023	A0	Rapport de validation en accord avec le plan de validation version A2 du 15/06/2023	AHI & KKH

Table des matières

1. Création et simulation du module de synchronisation 3
   1.1. Code VHDL du module synchronisation 3
   1.2. Code VHDL du test Bench pour le module de synchronisation 3
   1.3. Résultat de simulation du module synchronisation 4
      1.3.1. Validation du fonctionnement du reset 4
      1.3.2. Validation de la création de l'horloge 4
      1.3.3. Validation du signal H_SYNCH 5
      1.3.4. Validation du signal V_SYNCH 6
2. Ajout et simulation du module PLL au projet 7
   2.1. Code VHDL du module PLL 7
   2.2. Code VHDL du test Bench pour le module PLL 7
   2.3. Résultat de simulation du module PLL au fichier top du projet 8
      2.3.1. Validation de la création de l'horloge 8
      2.3.2. Validation de l'horloge VGA 8
3. Ajout et simulation du module Pattern_VGA au projet 9
   3.1. Code VHDL du module Pattern_VGA 9
   3.2. Code VHDL du test Bench pour la validation du module Pattern_VGA 10
   3.3. Résultat de simulation du module Pattern_VGA au fichier top du projet 10
4. Test de la solution intermédiaire 12
   4.1. Validation de la synthèse 12
   4.2. Validation de l'implémentation 13
   4.3. Génération du bitstream et test à l'oscilloscope 14
      4.3.1 Validation du signal H_SYNCH 14
      4.3.2. Validation du signal V_SYNCH 15
      4.3.3. Représentation de ligne à deux et trois parties blanche 16
   4.4. Démonstration de la partie intermédiaire 17

1. Création et simulation du module de synchronisation

1.1. Code VHDL du module synchronisation

La première étape lors que la création de notre module de synchronisation est de rédiger l'entité de notre boite noire en accord avec la proposition présenté dans le plan de validation.

On obtient l'entité suivante :

```
entity Synchro is
    port (
        clk                 : in std_logic;
        resetn              : in std_logic;
        H_SYNCH             : out std_logic;
        V_SYNCH             : out std_logic;
        Position_Horizontal : out integer;
        Position_Vertical   : out integer
    );
end Synchro;
```

Ensuite on passe à la rédaction de l'architecture de notre module de synchronisation.
Dans cette architecture, nous créons deux signaux internes :

- S_Position_Horizontal : entier allant de 0 à 799 (soit 800)
- S_Position_Vertical : entier allant de 0 à 524 (soit 525)

Ces deux variables correspondront à la dimension de notre image complète (visible et invisible) et sont affectées aux sorties respectives Position_Horizontal et Position_Vertical.

On créer un processus qui sera sensible aux signaux reset et horloge. Lorsque le reset sera activé, on remet les signaux H_SYNCH, V_SYNCH, S_Position_Horizontal et S_Position_Vertical à 0.
Sinon lorsque le reset ne sera pas activé, on réalise les incrémentations de nos variables S_Position_Horizontal et S_Position_Vertical à chaque front montant de notre horloge. Des que nos variables atteignent nos valeurs max, ils reviennent à 0. Ainsi on définit la taille de notre image complète.

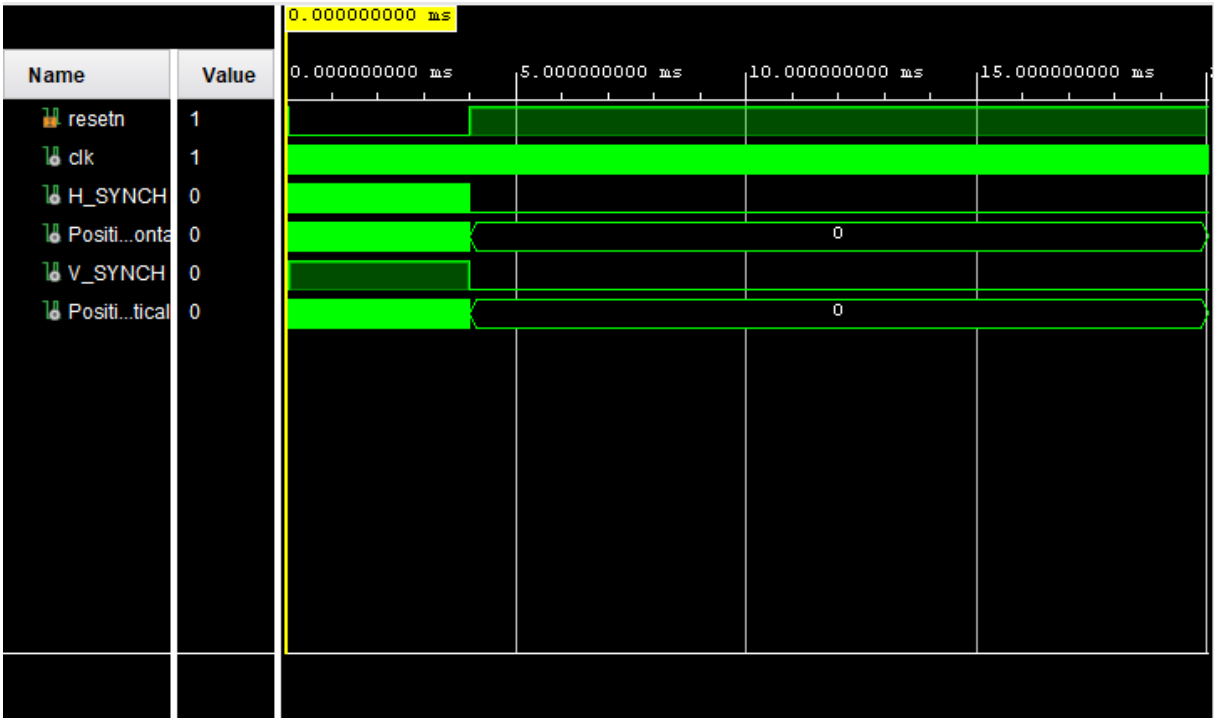
Pour continuer, on génère les mises à 0 de nos signaux H_SYNCH et V_SYNCH, ces derniers devront respectivement intervenir entre [655 : 751] et entre [489 : 491].

1.2. Code VHDL du test Bench pour le module de synchronisation

On réalise le test Bench pour valider notre module synchronisation en simulation. Pour ce faire, on réalise une horloge cadencée à 25.175 MHz. Ce qui donne une période de 39.72ns et un temps à l'état haut hp de 19.86ns.
Dans notre test bench on retrouvera deux processus, un pour l'horloge et un pour la gestion de notre reset.
Le reste sera désactivé pour permettre de visualiser deux séquences de V_SYNCH.

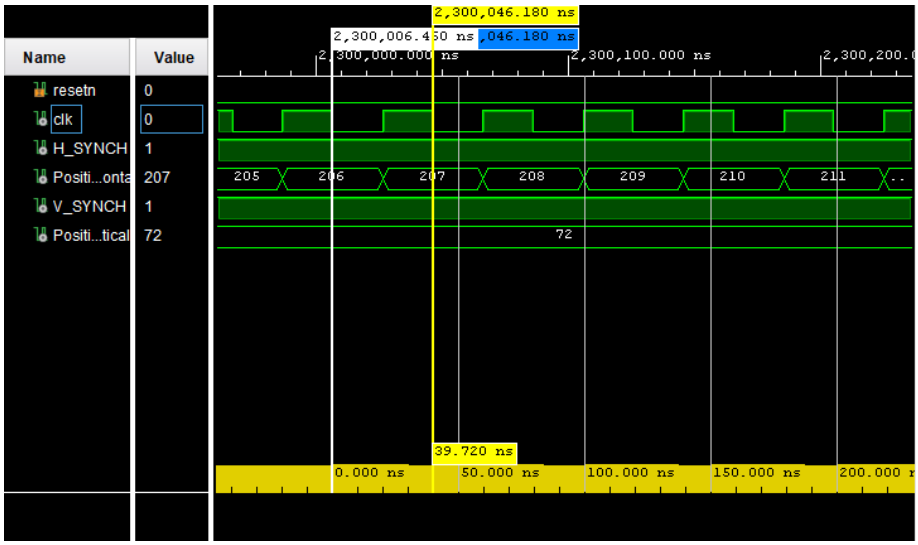
1.3. Résultat de simulation du module synchronisation

1.3.1. Validation du fonctionnement du reset



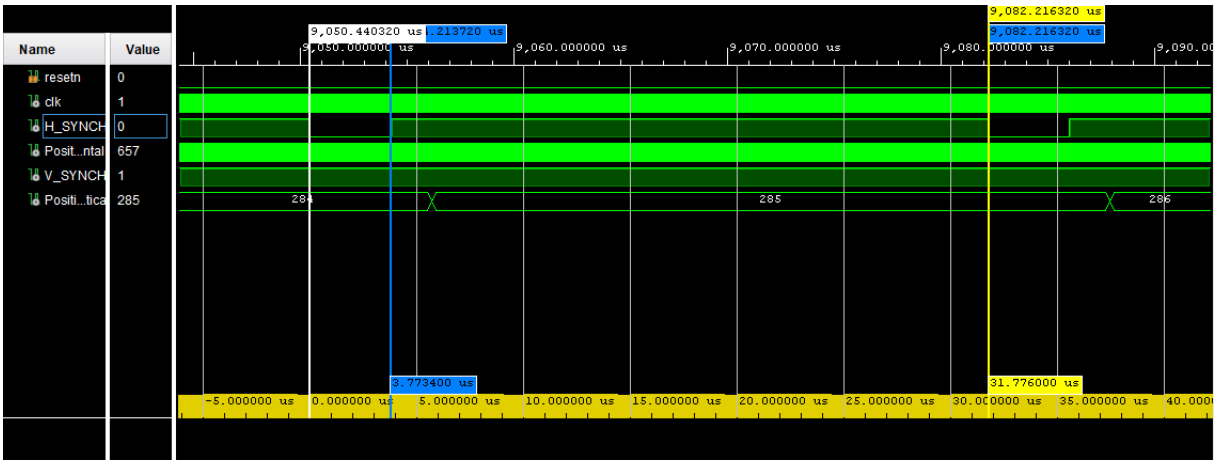
Le fonctionnement du reset est correct, car lorsque le reset passe à 1, les signaux H_SYNCH et V_SYNCH passe directement à 0 et les compteurs Position_Horizontal et Position_Vertical passe également à 0.

1.3.2. Validation de la création de l'horloge

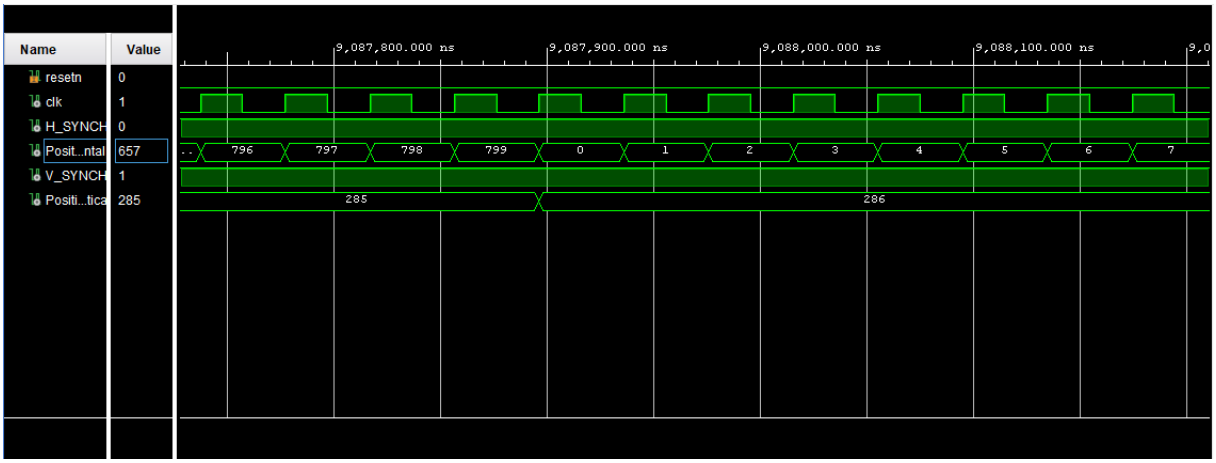


Notre période est de 39.72ns avec un rapport cyclique à 50%, ce qui correspond bien à une horloge cadencée à 25.175MHz comme souhaité.

1.3.3. Validation du signal H_SYNCH

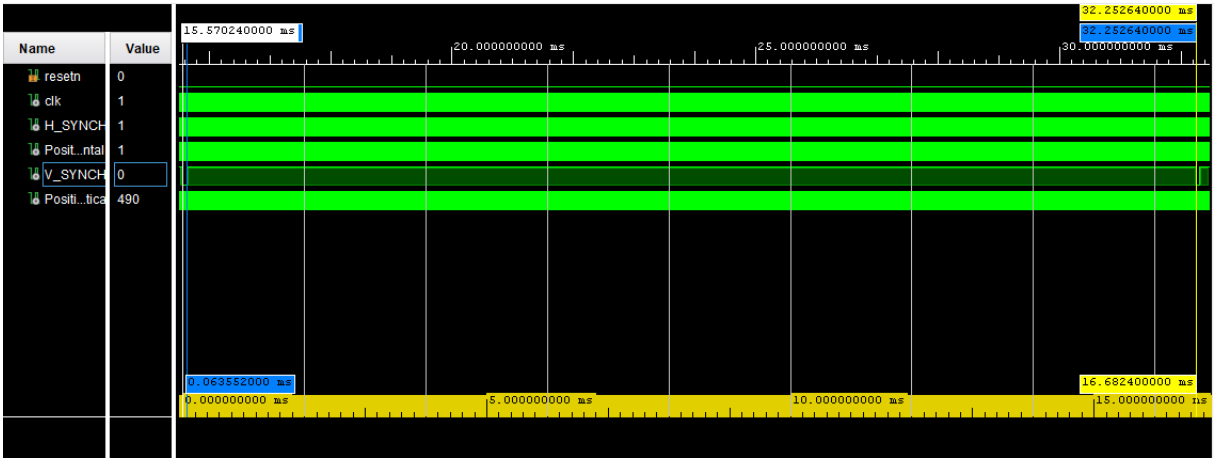


Ici on voit bien que notre « Pulse » sur le signal H_SYNCH dure 3.7734µs ce qui corresponds à notre 3.8µs attendu comme décrit dans la partie **Erreur ! Source du renvoi introuvable.**.
On voit également que le signal complet de H_SYNCH dure 31.7351µs ce qui correspond également à notre 31.778µs attendu comme décrit dans la partie **Erreur ! Source du renvoi introuvable.**.

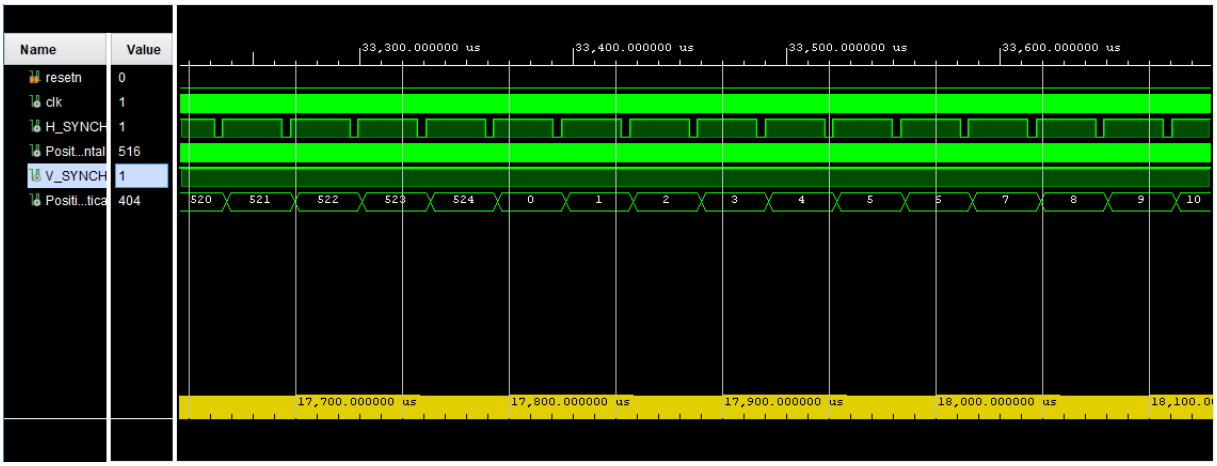


Le compteur compte bien de 0 à 799 comme demandé.

1.3.4. Validation du signal V_SYNCH



Ici on voit bien que notre « Pulse » sur le signal V_SYNCH dure 0.063552ms ce qui corresponds à notre 64ms attendu comme décrit dans la partie **Erreur ! Source du renvoi introuvable.**.
On voit également que le signal complet de V_SYNCH dure 16.6824ms ce qui correspond également à notre 16.68ms attendu comme décrit dans la partie **Erreur ! Source du renvoi introuvable.**.



Le compteur compte bien de 0 à 524 comme demandé.

2. Ajout et simulation du module PLL au projet

2.1. Code VHDL du module PLL

Pour la création du module PLL nous utilisons le composent « Clocking Wizard » présent dans le catalogue IP proposé par Vivado.

Comme expliqué dans le plan de validation nous prenons une horloge d'entrée de **clk = 125 MHz** (horloge fournit par la carte Cora Z7).

Une fois notre module PLL créé, il nous faut l'implémenter dans notre code.
Pour ce faire on ajoute le composant dans l'architecture de notre système, comme suit :

```
-- Composant PLL
component clock_VGA
        port (
            clk_125MHz  : in std_logic;
            reset       : in std_logic;
            clk_25MHz   : out std_logic;
            locked      : out std_logic
        );
 end component;
```

Ensuite, il nous faut affecter les signaux de notre module PLL aux signaux de notre projet :

```
--Affectation des signaux Pour le module Synchro
clock_VGA0 : clock_VGA
    port map (
        clk_125MHz => clk,
        reset     => resetn,
        clk_25MHz => clk_Vga,
        locked    => Locked_PLL_resetAuto
    );
```

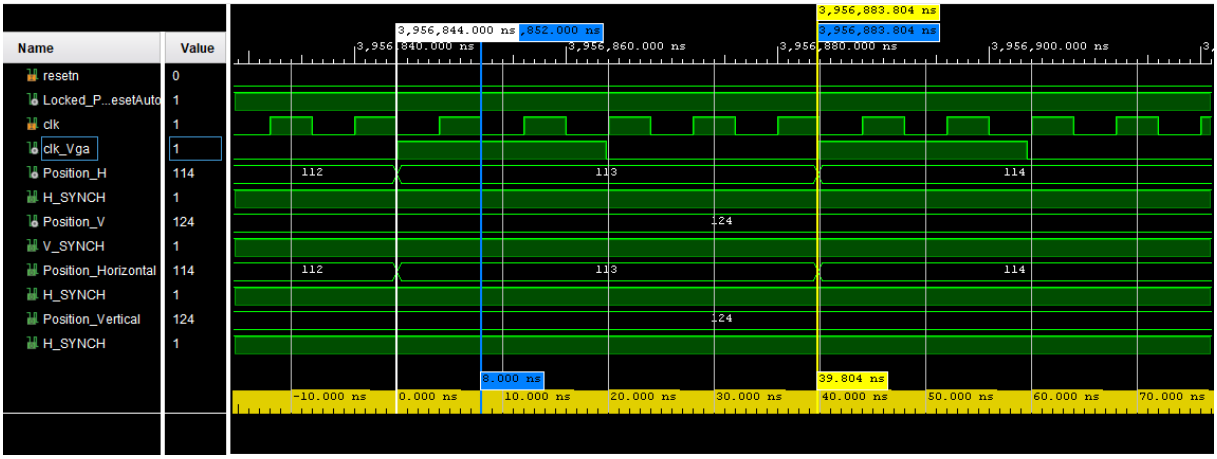
2.2. Code VHDL du test Bench pour le module PLL

On réalise le test Bench pour valider notre module PLL en simulation. Pour ce faire, on réalise une horloge cadencée à 125 MHz. Ce qui donne une période de 8 ns et un temps à l'état haut hp de 4 ns.

2.3. Résultat de simulation du module PLL au fichier top du projet

2.3.1. Validation de la création de l'horloge

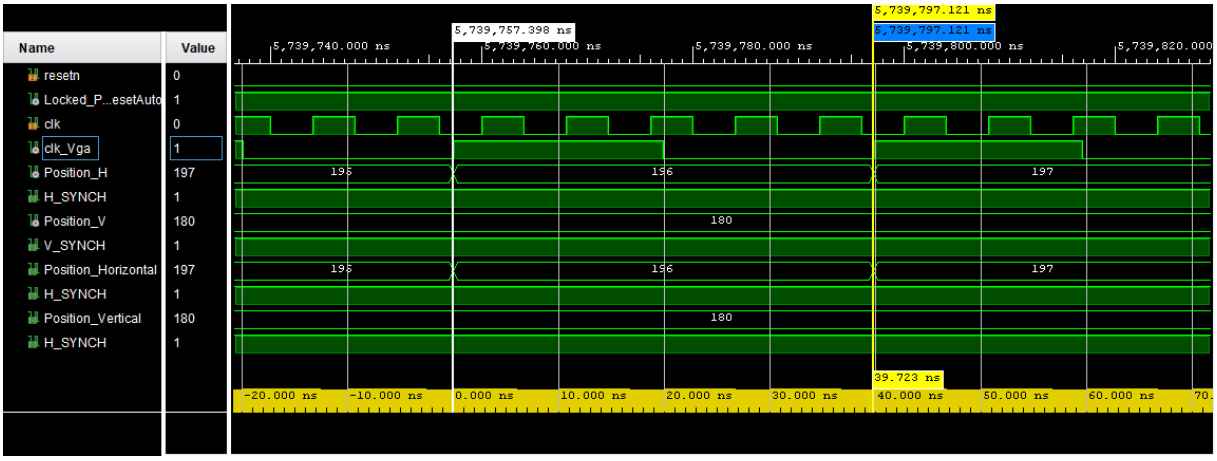
Dans un premier temps, nous allons valider que nous créons une horloge clk cadencé à 125 MHz.



Ici, on voit bien que notre période est de 8 ns soit une fréquence de 125 MHz. La création de notre horloge est correcte.

2.3.2. Validation de l'horloge VGA

Dans un second temps, nous allons valider que la PLL nous délivre bien un signal d'horloge cadencé à 25.175MHz.



Ici, on voit bien que notre période est de 39.723 ns soit une fréquence de 25.175 MHz. La création de notre horloge VGA est correcte.

3. Ajout et simulation du module Pattern_VGA au projet

3.1. Code VHDL du module Pattern_VGA

Pour réaliser le module Pattern_VGA, on créer déjà la boite noire en y retrouvant :

- En entrée :
  - L'Horloge : cadencé à l'horloge de fonctionnement du VGA
  - Le reset : ce qui permettra de réinitialiser le système
  - La localisation de pixel en horizontale et verticale
- En sortie :
  - Nos intensités de couleurs pour le rouge, le vert et le bleu

Le code est donc le suivant :

```
entity Pattern_VGA is
    port (
        clk             : in std_logic;
        resetn          : in std_logic;
        Position_H      : in integer;
        Position_V      : in integer;
        Rouge_Intensite : out std_logic_vector(3 downto 0);
        Vert_Intensite  : out std_logic_vector(3 downto 0);
        Bleu_Intensite  : out std_logic_vector(3 downto 0)
     );
end Pattern_VGA;
```

Ensuite on passe à la rédaction de l'architecture de notre module Pattern_VGA.
Dans cette architecture, nous créons deux signaux internes :

- Intensite_faible : qui est un std_logic_vector de 4 bits
- Intensite_haute : qui est un std_logic_vector de 4 bits

Ces deux variables correspondent à la représentation d'une intensité minimum (avec le signal égale à « 0000 ») et d'une intensité maximum (avec le signal égale à « 1111 »).

On créer un processus qui sera sensible aux signaux reset et horloge. Lorsque le reset sera activé, on remet les signaux de couleur rouge, vert et bleu à une intensité faible.
Sinon lorsque le reset ne sera pas activé, on réalise la création de notre damier sur la partie visible de l'image (640 x 480). Le reste des pixels (partie non visible de l'image) sont fixées à une intensité faible.

On ajoute le module à notre fichier top du projet pour ce faire, on ajoute le nouveau composant :

```
-- Composant Pattern_VGA
component Pattern_VGA
port (
    clk             : in std_logic;
    resetn          : in std_logic;
    Position_H      : in integer;
    Position_V      : in integer;
    Rouge_Intensite : out std_logic_vector(3 downto 0);
    Vert_Intensite  : out std_logic_vector(3 downto 0);
    Bleu_Intensite  : out std_logic_vector(3 downto 0)
 );
end component;
```

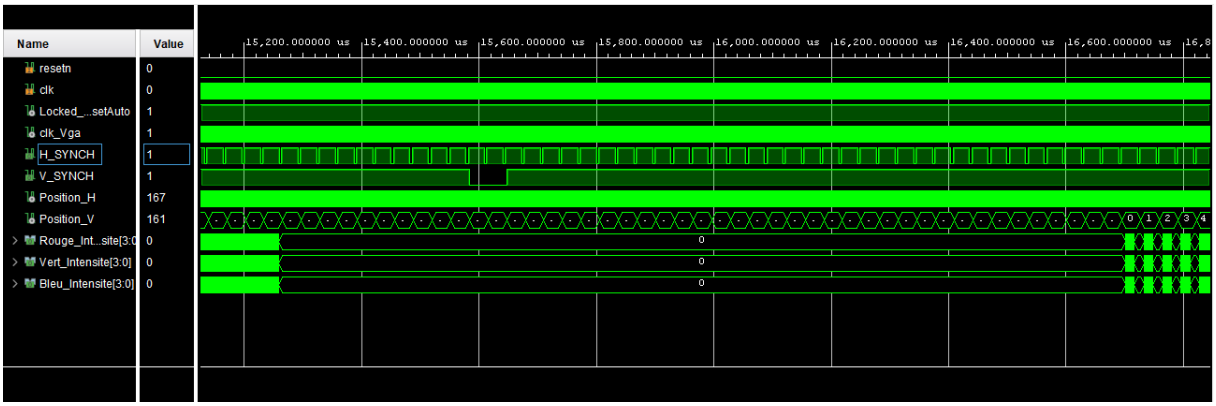
Puis on affecte nos signaux internes du module aux différents signaux de notre projet :

```
--Affectation des signaux Pour le module Synchro
Pattern_Damier : Pattern_VGA
    port map (
        clk             => clk_Vga,
        resetn          => activation_synchro,
        Position_H      => Position_H,
        Position_V      => Position_V,
        Rouge_Intensite => Rouge_Intensite,
        Vert_Intensite  => Vert_Intensite,
        Bleu_Intensite  => Bleu_Intensite
    );
```

3.2. Code VHDL du test Bench pour la validation du module Pattern_VGA

On réalise le test Bench pour valider notre module Pattern_VGA en simulation. Pour ce faire, on réalise une horloge cadencée à 125 MHz. Ce qui donne une période de 8 ns et un temps à l'état haut hp de 4 ns.

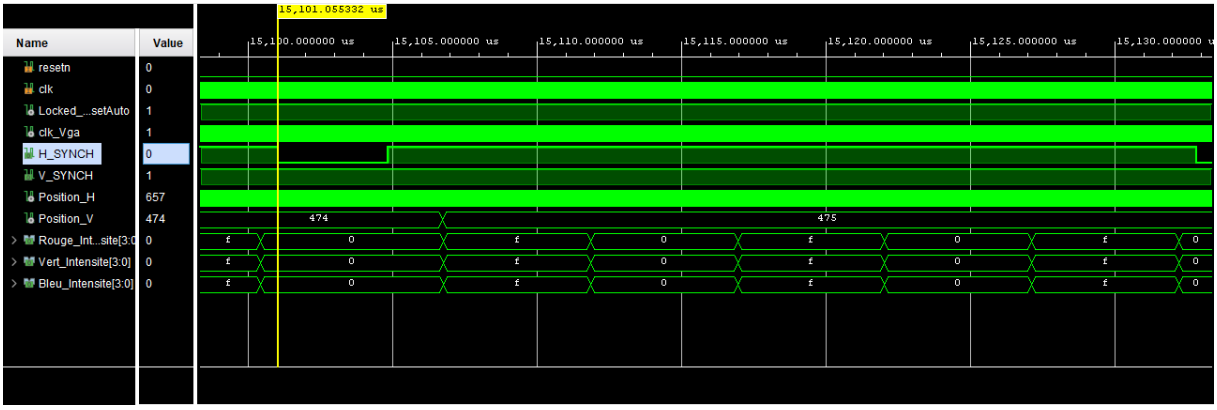
3.3. Résultat de simulation du module Pattern_VGA au fichier top du projet



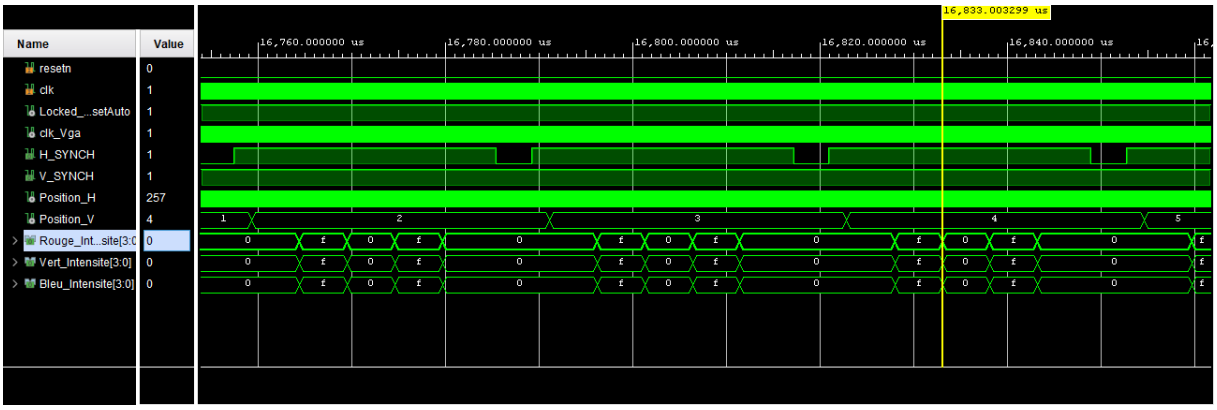
Sur cette simulation, on voit bien que la norme VGA est bien respectée. En effet les couleurs rouges, vert et bleu passe sont forcé à 0 lorsque l'on se trouve dans les plages non visible de notre image, soit :

- 10 pixels verticaux pour le front porch (10 cycles sur H_SYNCH)
- 2 pixels pour la largeur de syncronisation (moment ou V_SYNCH est à 0) (2 Cycle de H_SYNCH)
- 33 pixels verticaux pour le Back porch (33 cycles sur H_SYNCH)

Représentation d’une ligne avec 3 parties blanches :



Représentation d’une ligne avec 2 parties blanches :

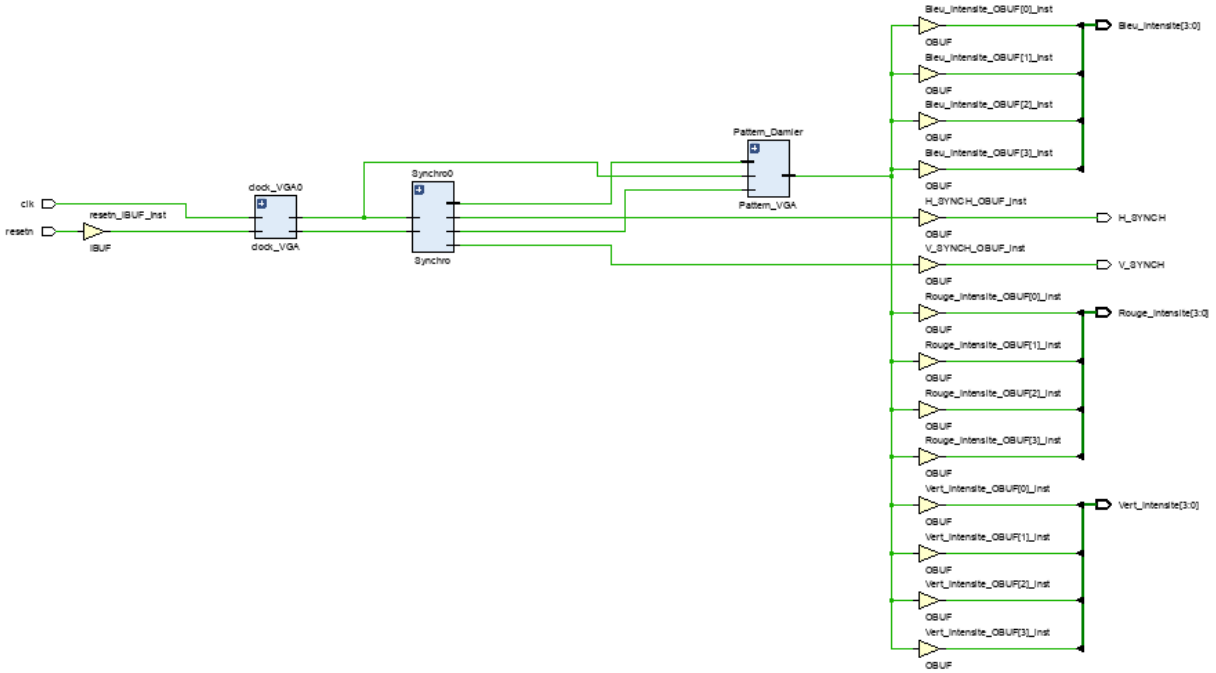


4. Test de la solution intermédiaire

La validation de notre solution étant terminée, nous pouvons passer au test de notre solution. Pour ce faire nous allons réaliser la programmation de notre carte Cora et valider les différents signaux via un oscilloscope.

4.1. Validation de la synthèse

La première étape, pour générer un Bitstream, consiste à lancer une implémentation. L'implémentation nous permet également de voir le schéma de notre projet.



Ici, on voit bien que l'on a :

- Deux signaux d'entrée
  - Clk (1 bit)
  - Resetn (1 bit)
- Cinq signaux de sortie
  - H_Synch (1 bit)
  - V_SYNCH (1 bit)
  - Rouge_intensite (4 bits)
  - Vert_intensite (4 bits)
  - Bleu_intensite (4 bits)
- Trois modules
  - Module PLL
  - Module Synchro
  - Module Pattern_VGA

On peut donc en déduire que notre solution respect les attentes demandées.

4.2. Validation de l'implémentation

L'implémentation nous permet de valider que nous n'avons pas de violation de timing.

Design Timing Summary

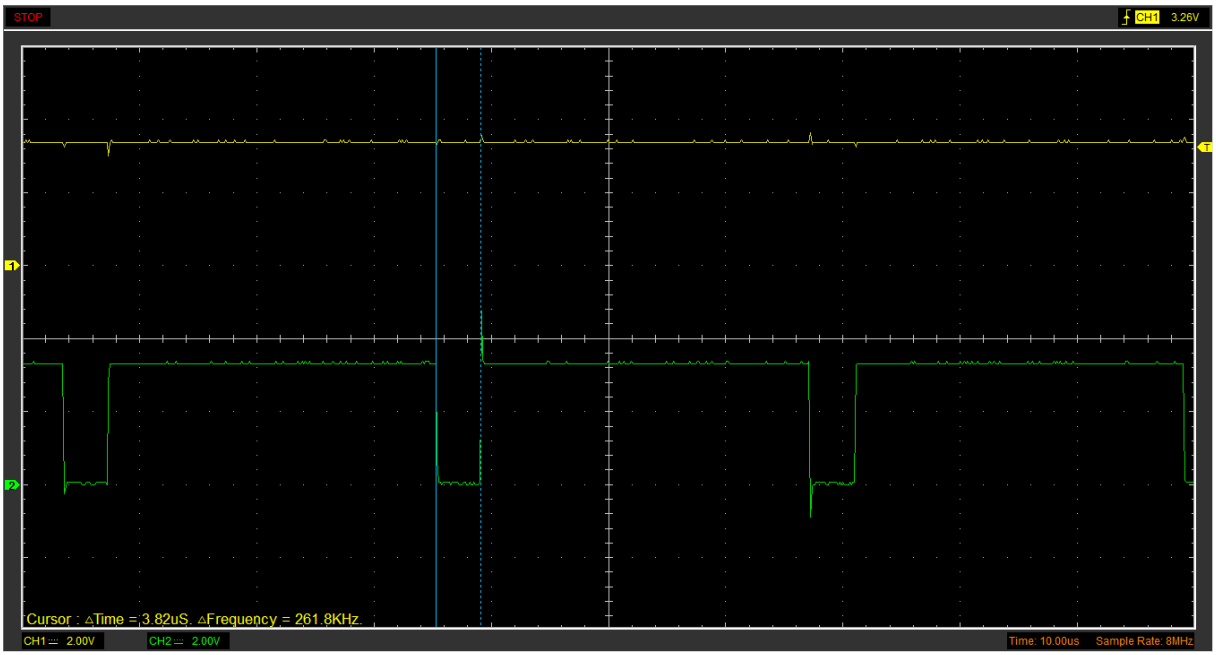
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	36,282 ns	Worst Hold Slack (WHS):	0,092 ns	Worst Pulse Width Slack (WPWS):	2,000 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	44	Total Number of Endpoints:	44	Total Number of Endpoints:	40

All user specified timing constraints are met.

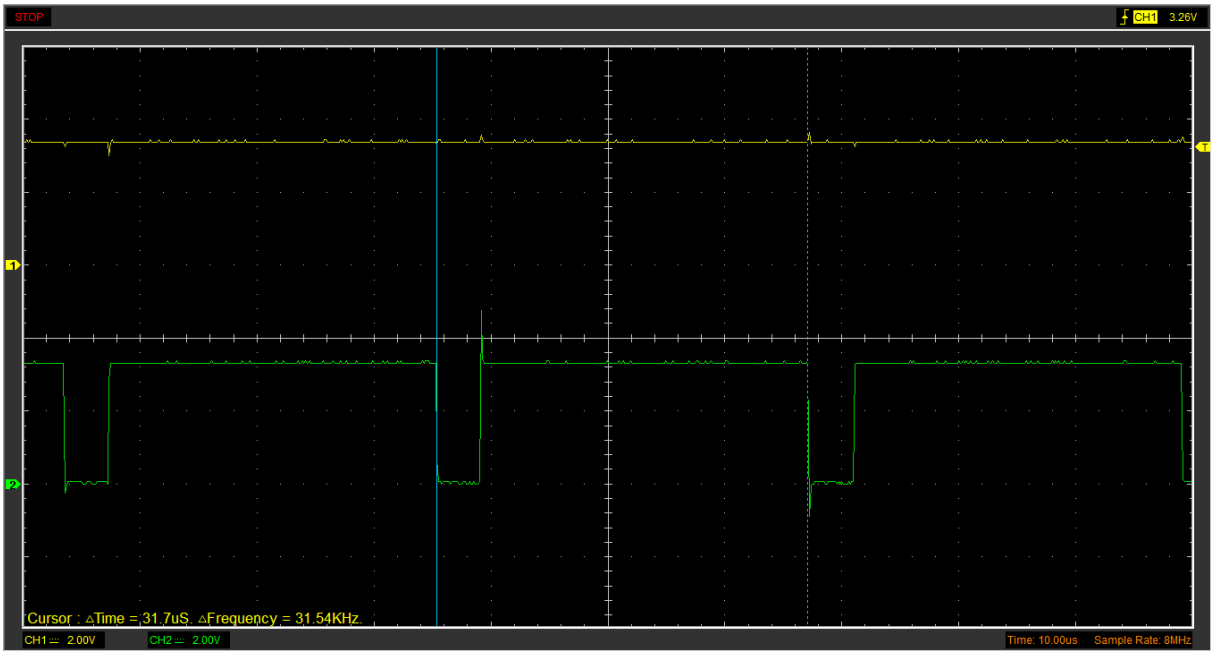
Dans notre cas, tout est correct puisque nous n'avons pas de stack.

4.3. Génération du bitstream et test à l'oscilloscope

4.3.1 Validation du signal H_SYNCH



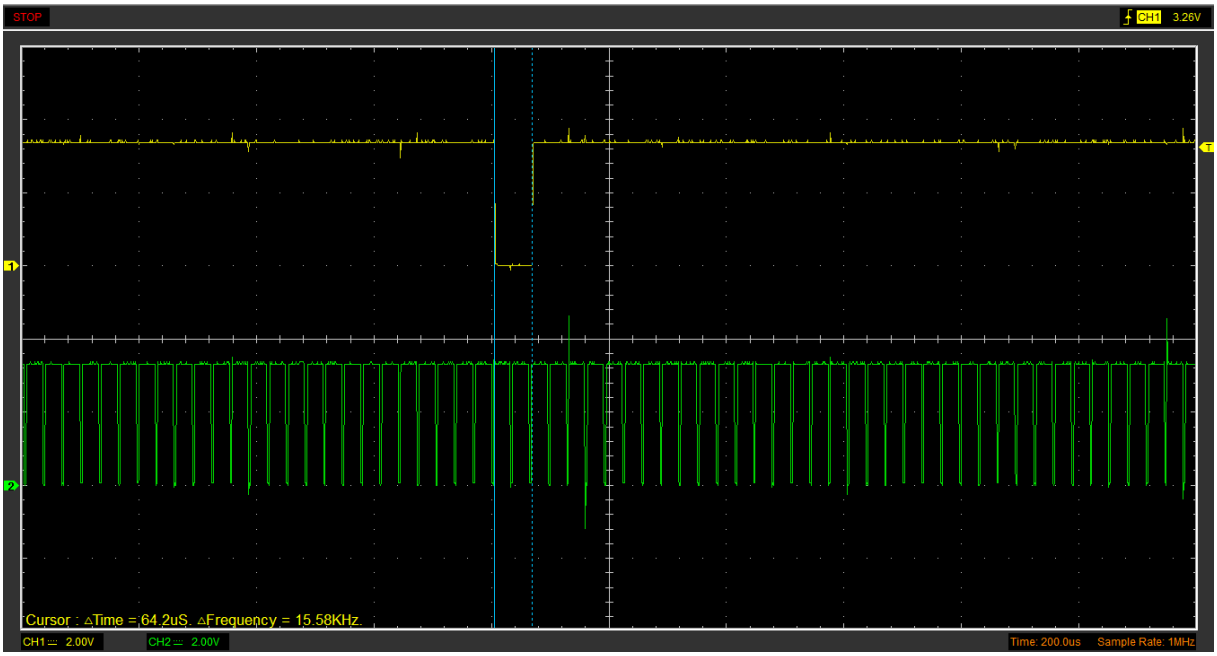
Ici, on voit bien que l'impulsion du signal H_SYNCH est de 3.82µs ce qui correspond à notre attente.



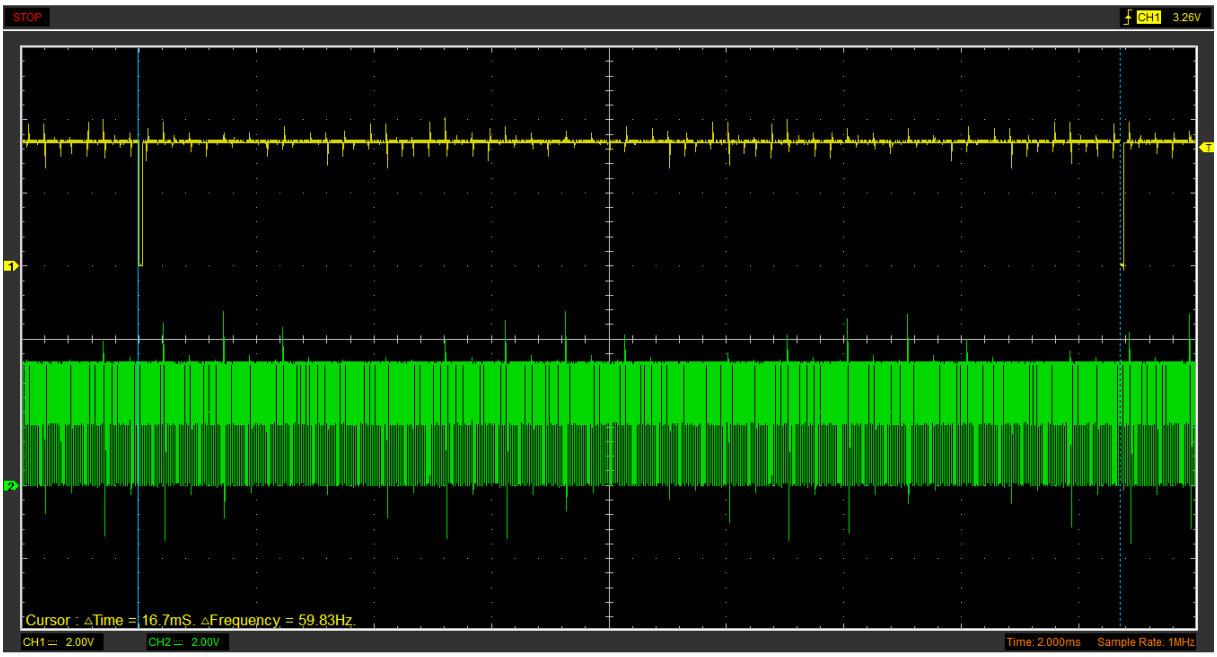
Et que la période du signal H_SYNCH est de 31.7µs ce qui est conforme également.

Le signal H_SYNCH est donc correct à notre attente.

4.3.2. Validation du signal V_SYNCH



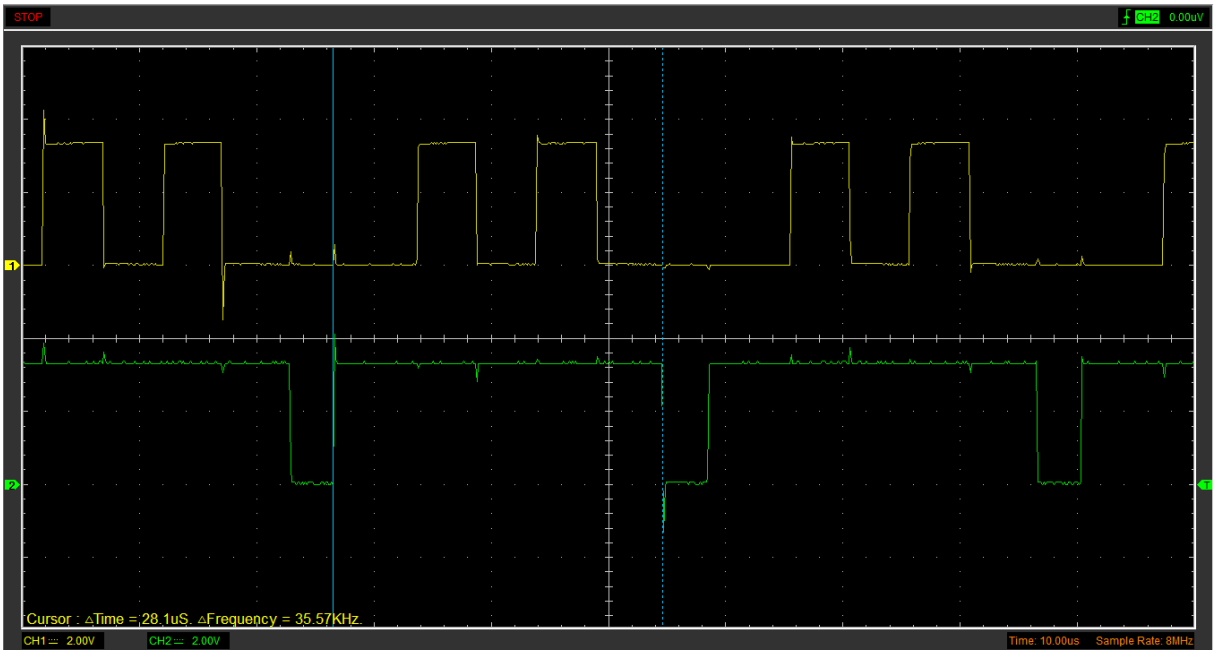
Ici, on voit bien que l'impulsion du signal V_SYNCH est de 64.2µs ce qui correspond à notre attente.



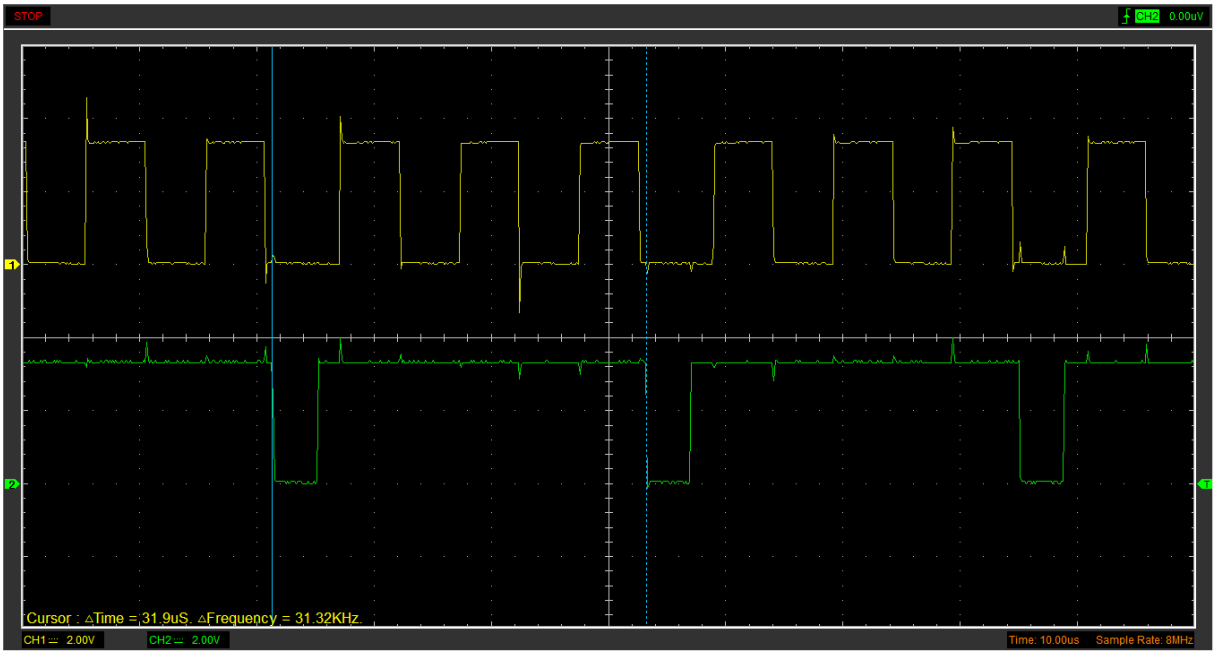
Et que la période du signal V_SYNCH est de 16.7ms ce qui est conforme également.

Le signal V_SYNCH est donc correct à notre attente.

4.3.3. Représentation de ligne à deux et trois parties blanche

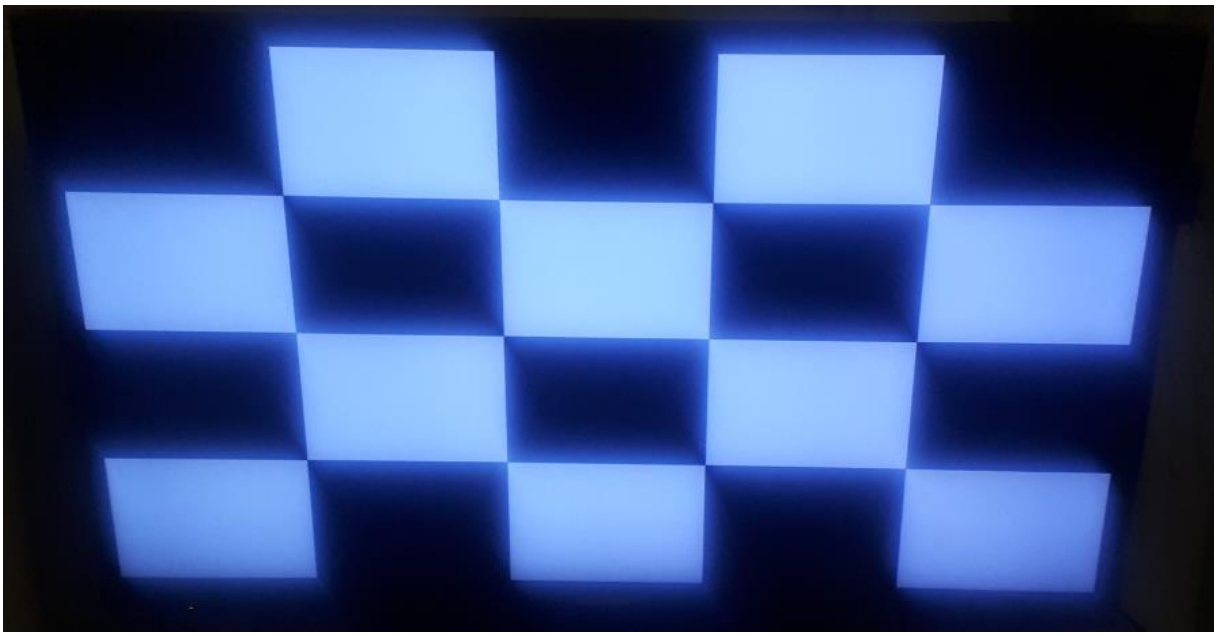


Ici, on voit bien la représentation d'une ligne à deux parties blanches.



Ici, on voit bien la représentation d'une ligne à trois parties blanches.

4.4. Démonstration de la partie intermédiaire



La projection de notre pattern sur l'afficheur nous permet de valider notre solution intermédiaire car on voit bien que :

- Le damier est correctement configurer avec cinq colonnes et quatre lignes avec des lignes à 2 parties blanches et des lignes à 3 parties blanches
- Qu'aucune ligne n'est décalée car les carrés blanc sont bien placé au bonne endroit
- Notre damier reste bien dans la partie visible de notre afficheur.

De plus, lorsque l'on appuie sur le bouton 0 de notre carte cora, le reset s'active, et l'afficheur s'éteint.