ANALYSEUR DE SPECTRE

sur Zedboard (Zynq-7000)

PROJET CO-DESIGN

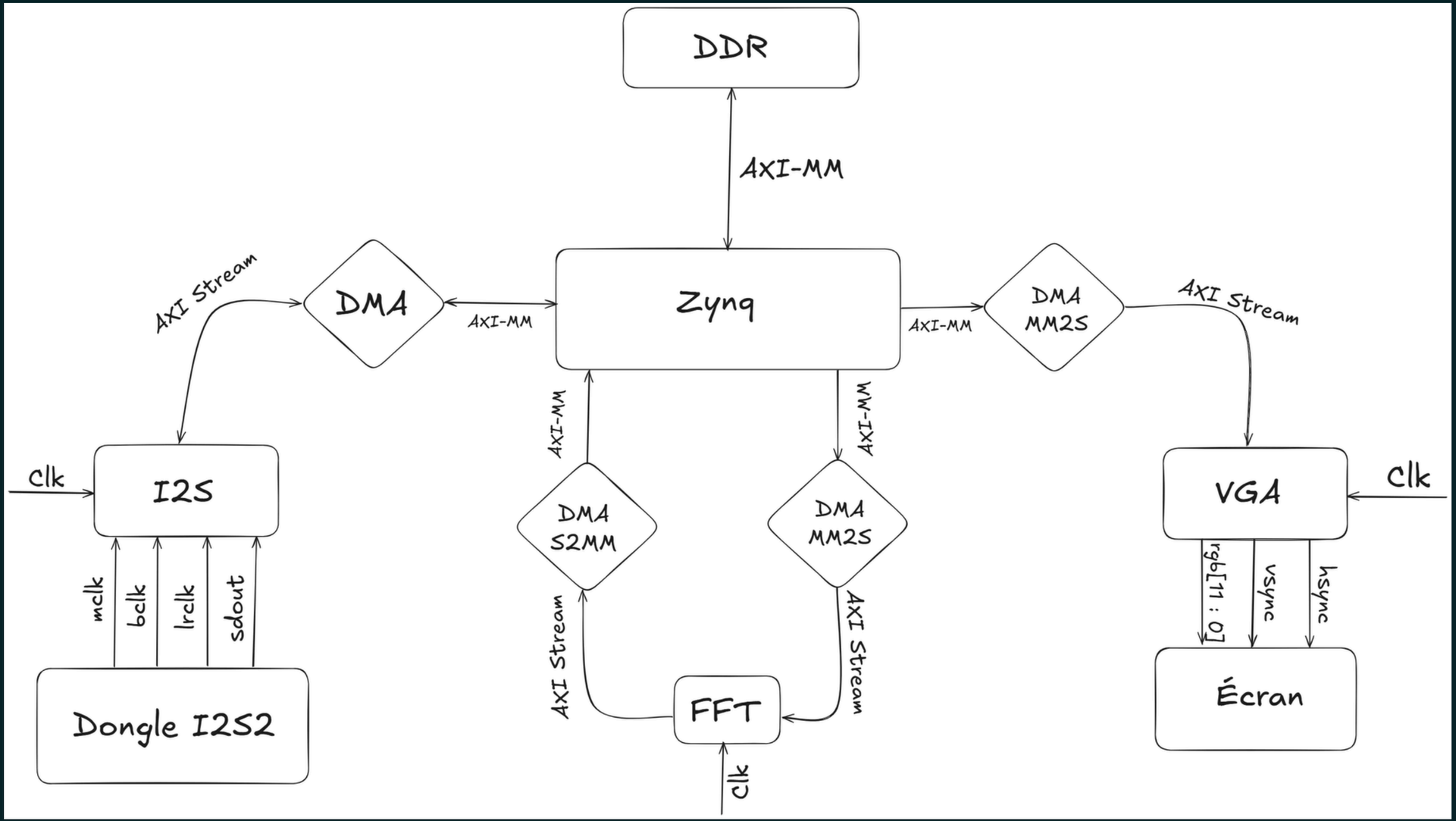
Étudiants :

Asdjad BAKARY
Ayoub LADJICI
Elsa RAKOTO
Ioana CACAU
Zimeng JIANG

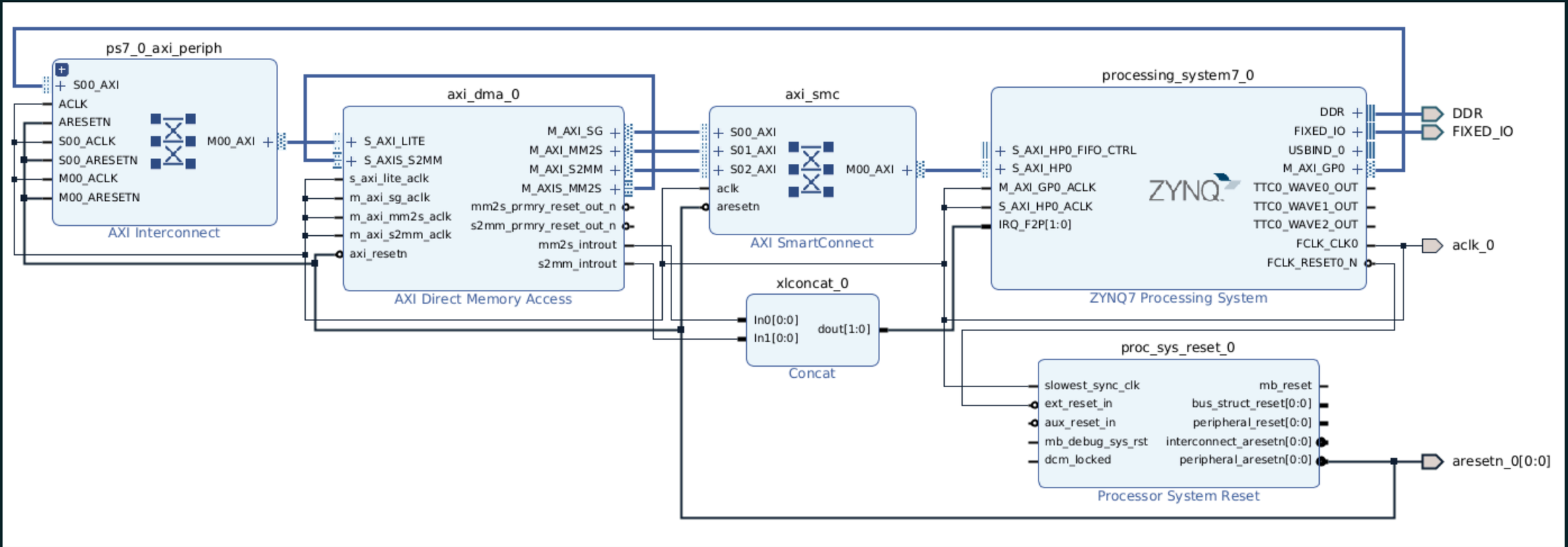
Enseignants référents :

Maxime LEBRETON
Raphaël BRESSON

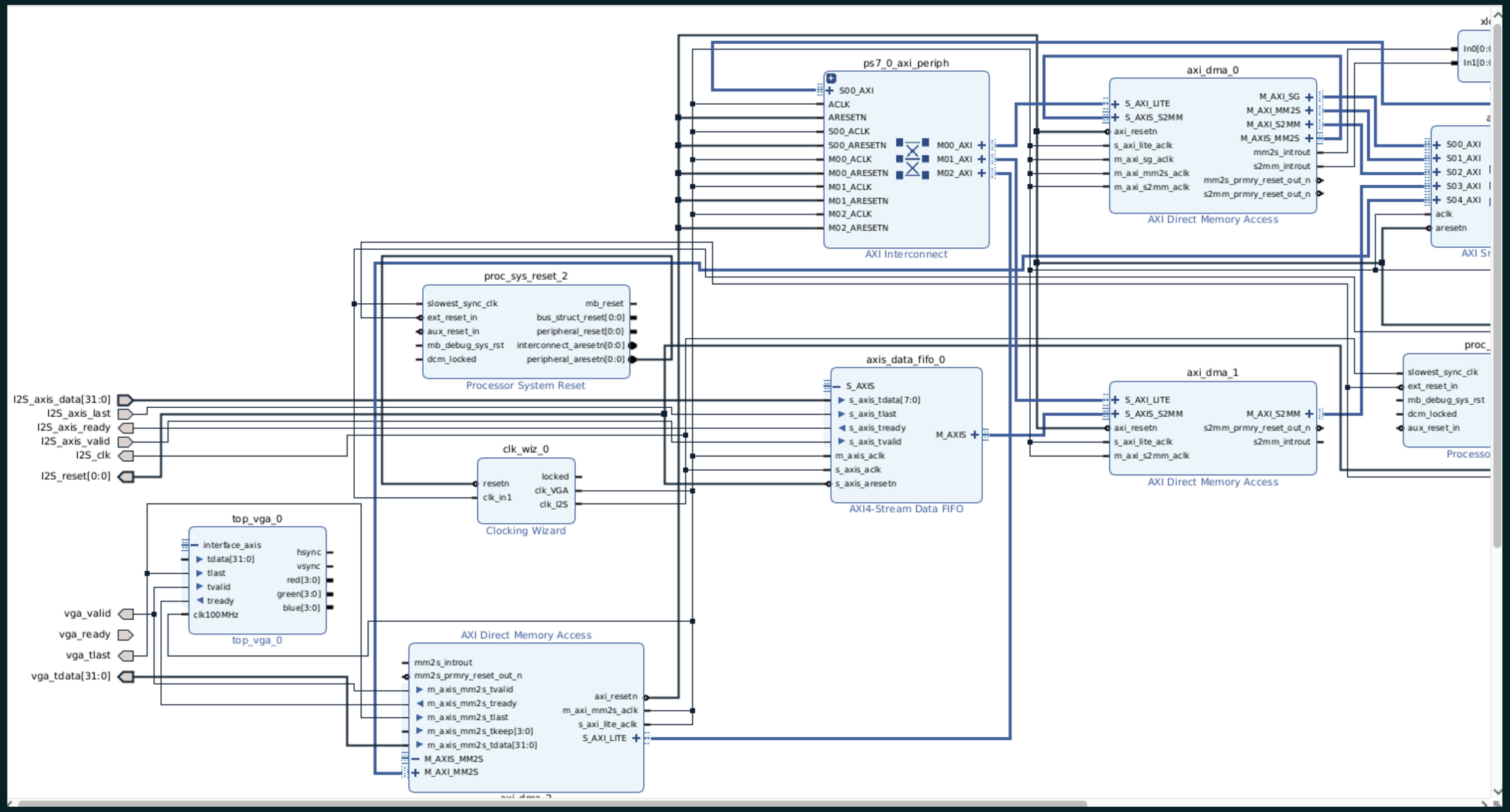
Architecture simplifiée



Architecture globale



Architecture globale



Choix des fréquences

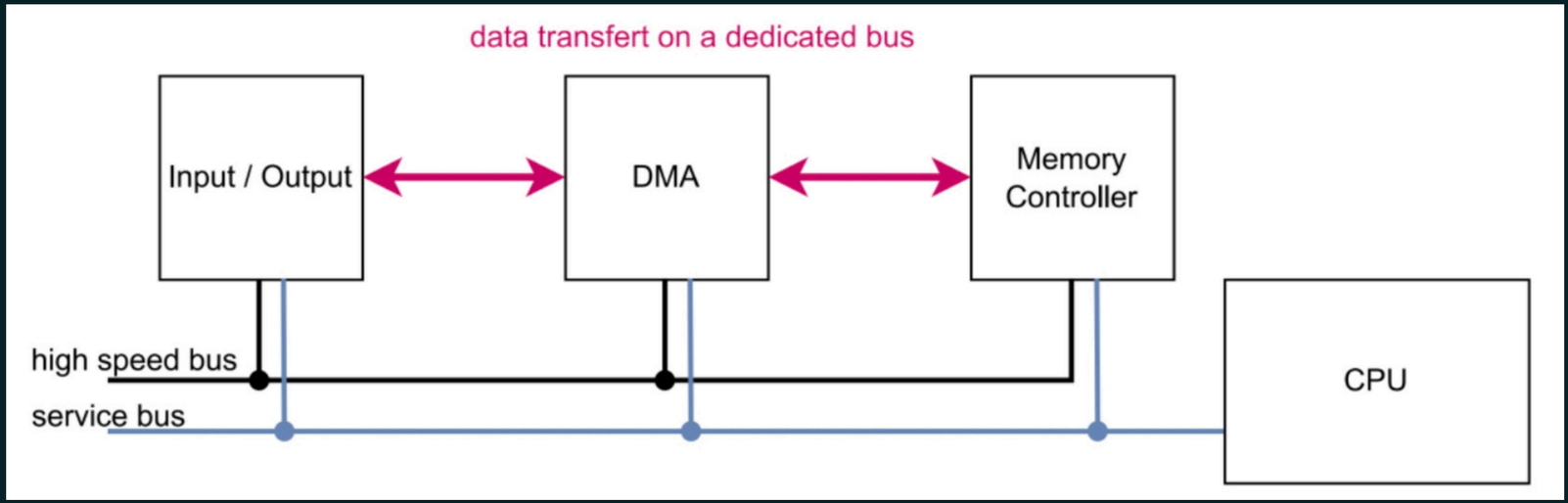
Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36

LRCK (kHz)	MCLK (MHz)									
	64x	96x	128x	192x	256x	384x	512x	768x	1024x	1152x
32	-	-	-	-	8.1920	12.2880	-	-	32.7680	36.8640
44.1	-	-	-	-	11.2896	16.9344	22.5792	33.8680	45.1580	-
48	-	-	-	-	12.2880	18.4320	24.5760	36.8640	49.1520	-
64	-	-	**8.1920**	**12.2880**	-	-	32.7680	49.1520	-	-
88.2	-	-	**11.2896**	**16.9344**	**22.5792**	**33.8680**	-	-	-	-
96	-	-	**12.2880**	**18.4320**	**24.5760**	**36.8640**	-	-	-	-
128	8.1920	12.2880	-	-	**32.7680**	**49.1520**	-	-	-	-
176.4	11.2896	16.9344	22.5792	33.8680	-	-	-	-	-	-
192	12.2880	18.4320	24.5760	36.8640	-	-	-	-	-	-
Mode	QSM				DSM		SSM			

Speed Mode	MCLK/LRCK Ratio	SCLK/LRCK Ratio	Input Sample Rate Range (kHz)
Single-Speed Mode	256x	64	4-24, 43-54
	512x	64	43-54
	384x	64	4-24, 43-54
	784x	64	43-54

Direct Memory Access (DMA)

Le DMA est un composant matériel qui transfère des données entre des périphériques (ex: I2S, FFT, VGA) et la mémoire (DDR) sans intervention continue du CPU.



Les deux modes principaux du DMA Xilinx

Mode simple

- Déclenche un seul transfert à la fois
- Configuration manuelle via registres(baremetal)
- Le CPU intervient pour chaque transfert
- Le CPU peut être bloqué (polling)
- Facile à implémenter (pas de strucutres mémoire)

Mode Scatter-Gather

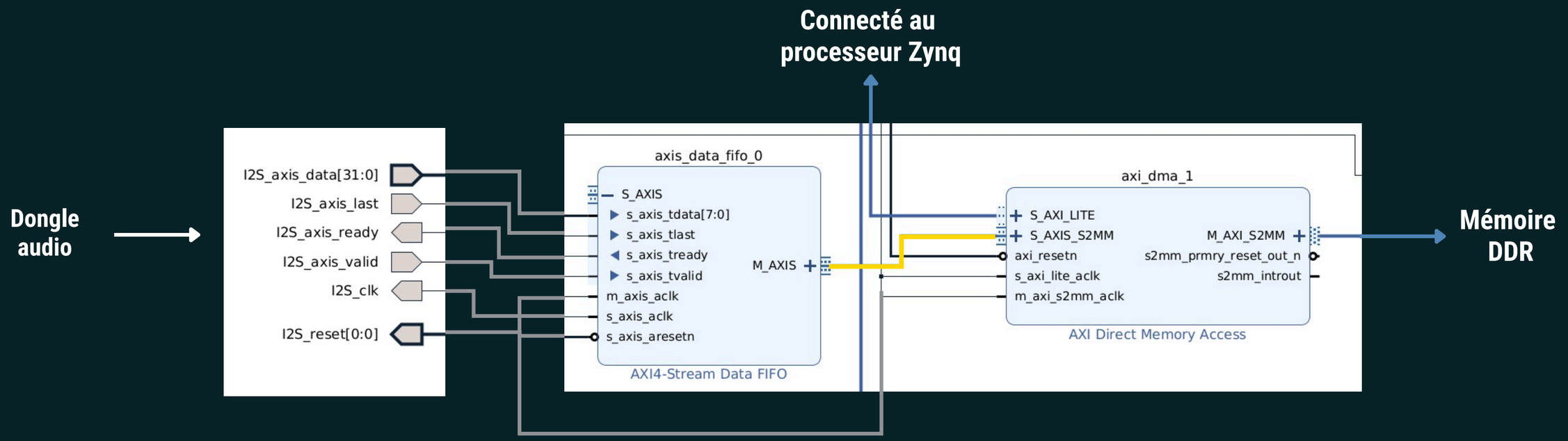
- Liste de descripteurs en mémoire
- Transferts enchaînés automatiquement
- Le CPU intervient une seule fois au début
- Plus complexe:
  → structures précises à créer

- Mais plus puissant pour les traitements continus

Rôle du DMA

Câblage autour du DMA

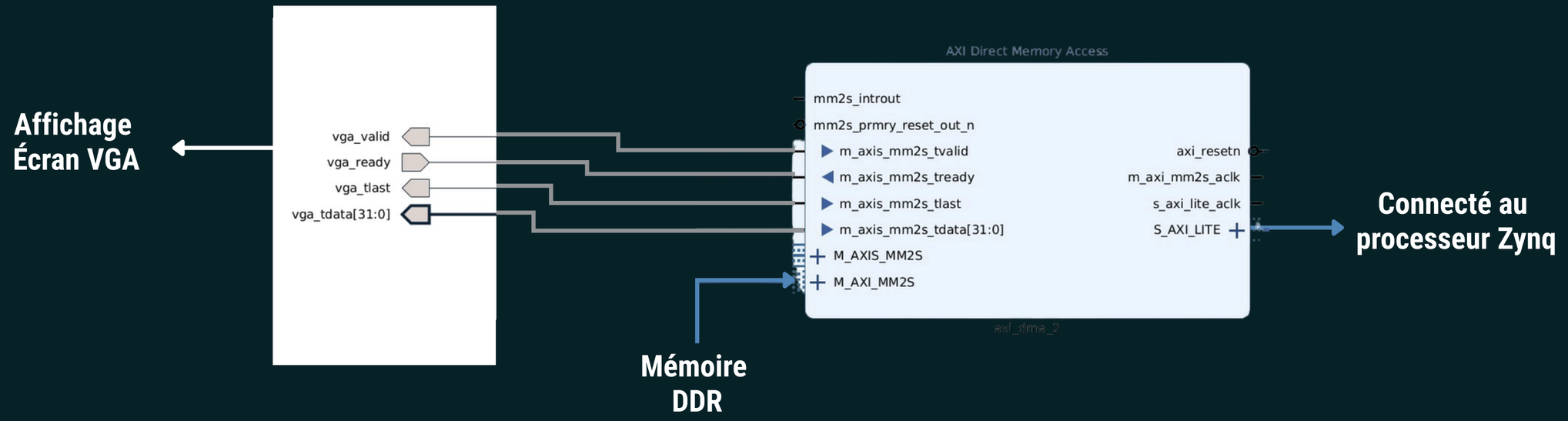
DMA1 configuré en mode écriture (S2MM)



Rôle du DMA

Câblage autour du DMA

DMA2 configuré en mode lecture (MM2S)



Les registres essentiels :

- *Control Register*
- *Status Register*
- *Souce Address Registre sur 64 bits*
- *Length Registre*

symetrique dans deux sens →

| MM2S 0x00 - 0x28

| S2MM 0x30 - 0x58

Approches :

```
Xil_Out(ADDR_BASE_DMA + Offset , data)
XAxiDma_SimpleTransfer(&axiDma, buffer_addr, buffer_size, direction);
```

Code Baremetal

PS (CPU) —AXI4-Lite / xaxidma.h→ **PL (DMA)**

Code Baremetal

```
// ---------------------------------------------------------
// Étape 1 : Transfert I2S -> DDR (S2MM)
// ---------------------------------------------------------
// 1. Écrire l'adresse de destination (DDR)
Xil_Out32(0x40400000 + 0x48, 0x00100000);
// 2. Écrire la taille du transfert
Xil_Out32(0x40400000 + 0x58, 1024);
// 3. Activer le DMA (bit 0 = start, bit 1 = interruption)
Xil_Out32(0x40400000 + 0x30, 0x01);
// 4. Attendre la fin du transfert (polling du bit IDLE)
while (!(Xil_In32(0x40400000 + 0x30) & 0x02));
```

```
// ---------------------------------------------------------
// Étape 4 : Transfert DDR -> VGA (MM2S)
// ---------------------------------------------------------
XAxiDma_SimpleTransfer(&axiDma, (u32)vgaBuffer, BUFFER_SIZE * sizeof(u32), XAXIDMA_DMA_TO_DEVICE);
while (!XAxiDma_IntrGetIrq(&axiDma, XAXIDMA_DMA_TO_DEVICE));
```

I2S

OUT

MCLK	22.5792 MHz
SCLK	2.8224 MHz
LRCLK	44.1 KHz

/8

/12

IN

SDIN	Ajouté à chaque front descendant de SCLK

CÔTÉ DONGLE

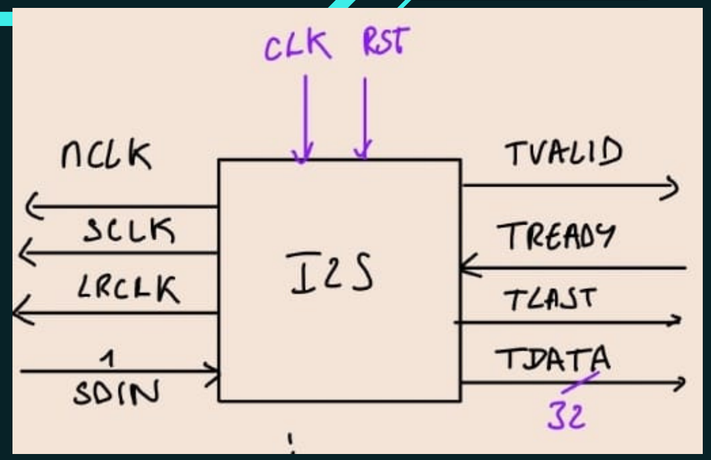
Rythme d'envoi des horloges :

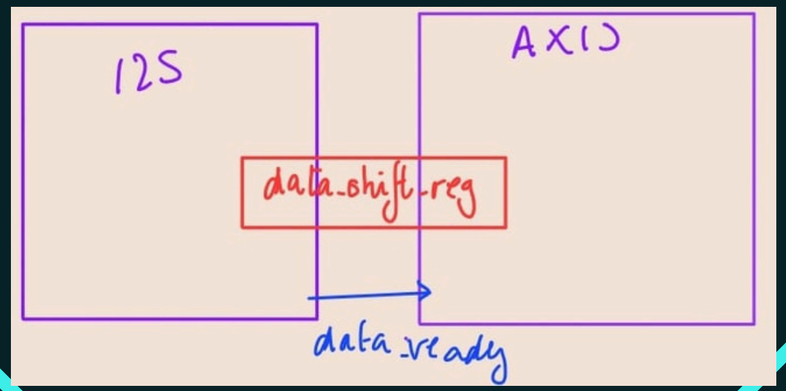
Process : clk_gen

```
elsif rising_edge(I2S_clk) then
    -- Incrémenter le compteur
    counter <= counter + 1;

    -- Génération de SCLK (div par 8)
    sclk_i <= counter(2);

    -- Génération de LRCLK (div par 512)
    lrclk_i <= counter(8);
end if;
```

I2S

CÔTÉ DONGLE





Réceptionner et stocker les 24 bit de canal gauche ou droit

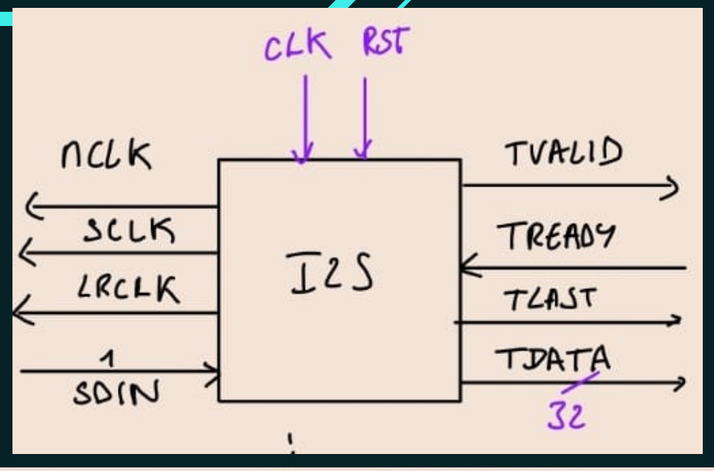
Process : i2s_receive

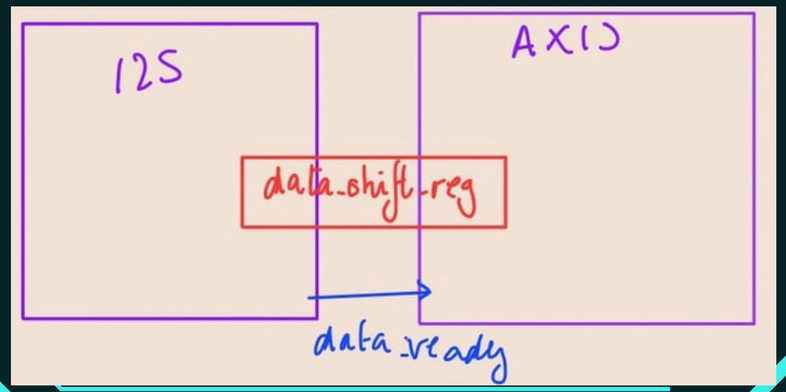
```
-- Réinitialiser data_ready après prise en compte
if data_ready = '1' and tvalid_reg = '0' then
    data_ready <= '0';
end if;

-- Traiter le changement de canal en priorité
if lrclk_i /= lrclk_prev then
    bit_counter <= 0;  -- Réinitialiser le compteur pour le nouveau canal
    collecting_bits <= '1';  -- Commencer à collecter des bits
end if;

-- Échantillonnage des données sur le front descendant de SCLK
if sclk_prev = '1' and sclk_i = '0' then
    -- Toujours décaler les bits dans le registre
    data_shift_reg <= data_shift_reg(22 downto 0) & sdin;
```

Fin Process : bit_counter = 24 → data_ready <= '1';

I2S

CÔTÉ DONGLE





Envoi des données dans un vecteur 32 bits

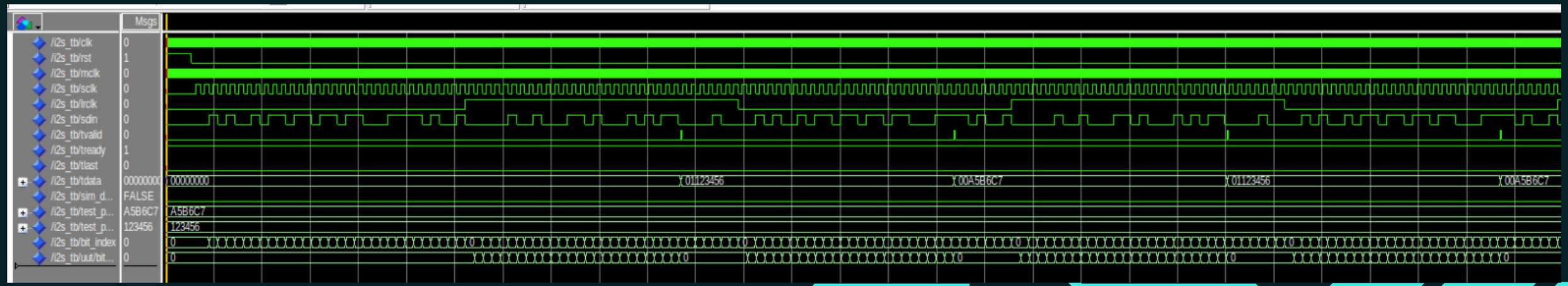
Process : axi_stream

```
tlast_reg <= '0';

-- Si des données sont prêtes et qu'on n'est pas déjà e
if data_ready = '1' and tvalid_reg = '0' then
    -- Format: 7 bits à '0' + 1 bit de canal + 24 bits
    tdata_reg <= "0000000" & lrclk_i & data_shift_reg;
    tvalid_reg <= '1';
```

Gestion du nombre de paquet pour tlast

I2S

Simulation

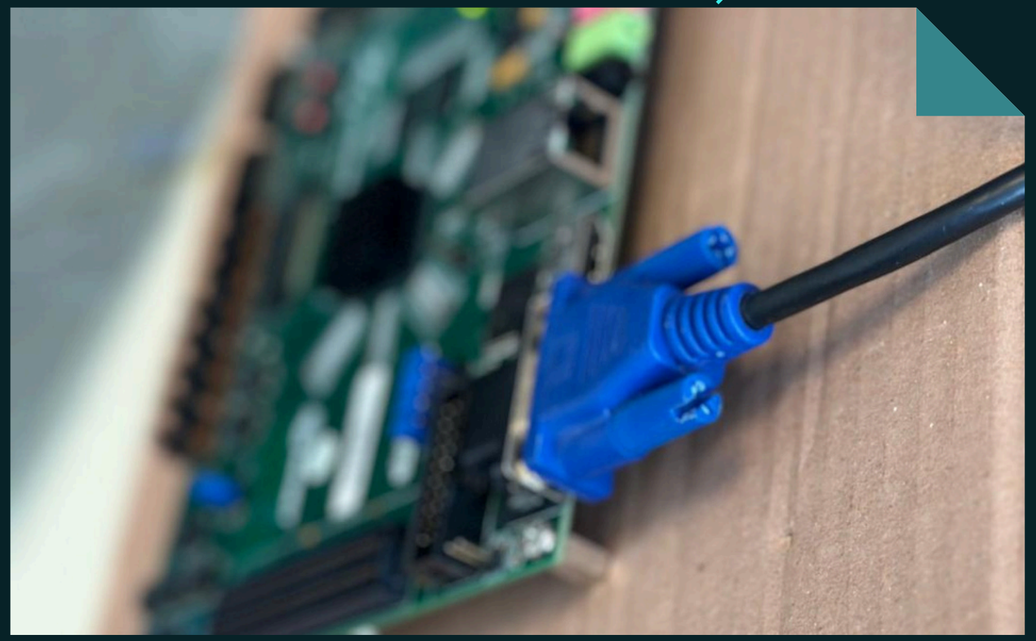



```
else
    -- Sélectionner le bon motif selon le canal actif
    if lrclk = '0' then
        -- Canal gauche (LRCLK = 0)
        sdin <= test_pattern_left(23 - bit_index);
    else
        -- Canal droit (LRCLK = 1)
        sdin <= test_pattern_right(23 - bit_index);
    end if;
```

lrclk
Padding
Canal Gauche et droit
Envoi à tvalid

Présentation VGA

Affichage final, pour matérialiser visuellement les données issues du système audio



- *Source de données*

Flux I2S → données numériques

↓ *Stockage*

DDR (Double Data Rate) via un DMA

↓ *Lecture*

2nd DMA vers le module VGA

↓

Affichage dynamique sur l'écran

VGA

Générer un affichage conforme à la norme VGA 1024×768 à 60 Hz.

Paramètres de synchronisation utilisés:

Format	**Pixel Clock 1**	H.Front Porch	H.Sync Pulse	H.Back Porch	**H.Total**	V.Front Porch	V.Synch Pulse	V.Back Porch	**V.Total**
1024x768, 60Hz	65MHz	24 px	136 px	160 px	**1344 px**	3 lignes	6 lignes	29 lignes	**806 lignes**
Zone visible	Fréquence de pixels	Pause après ligne visible	Durée du signal hsync à 0	Pause avant nouvelle ligne	1024 + 24 + 136 + 160	Pause après l'image visible	Durée du signal vsync à 0	Pause avant nouvelle image	768 + 3 + 6 + 29
Attendues	15.384 ns	369 ns	2.09 us	2.46 us	20.67 us	768 - 769 - 770	771-772-773-774-775-776	776 --- 805	0 --- 805
	1 / 65MHZ	24 * 15.384ns	136 * 15.384ns	160 * 15.385 ns	1344 * 15.385 ns	vsync → 0	vsync → 1	videon → 1	reinitialisation
						62.03 us	124..06 us	599.65 us	16.66 ms

VGA Sans AXIStream

Simulation

Structure du projet

vga_controller.vhd

Synchronise parfaitement l'affichage en indiquant à l'écran quand commencer une ligne, une image, et quels pixels sont valides à afficher.

```
architecture Behavioral of vga_controller is
    --Synchronisation horizontale
    constant H_DISPLAY : integer := 1024;
    constant H_FRONT_PORCH : integer := 24;
    constant H_SYNC_PULSE : integer := 136;
    constant H_BACK_PORCH : integer := 160;
    constant H_TOTAL : integer := H_DISPLAY + H_FRONT_PORCH + H_SYNC_PULSE + H_BACK_PORCH;

    --Synchronisation verticale
    constant V_DISPLAY : integer := 768;
    constant V_FRONT_PORCH : integer := 3;
    constant V_SYNC_PULSE : integer := 6;
    constant V_BACK_PORCH : integer := 29;
    constant V_TOTAL : integer := V_DISPLAY + V_FRONT_PORCH + V_SYNC_PULSE + V_BACK_PORCH;

    signal h_count : integer range 0 to H_TOTAL - 1 := 0;
    signal v_count : integer range 0 to V_TOTAL - 1 := 0;
```

VGA Sans AXIStream

Simulation

Structure du projet

vga_pattern.vhd

Génère une image en damier, où chaque carré de 64×64 pixels est rouge ou bleu selon sa position.

- 16 colonnes
- 12 lignes

```
architecture Behavioral of vga_pattern is
begin
    process(pixel_x, pixel_y, video_on)
        variable x : integer := 0;
        variable y : integer := 0;
    begin
        if video_on = '1' then  -- on affiche une couleur que si on est dans la zone visible
            x := to_integer(unsigned(pixel_x));   --n colonne
            y := to_integer(unsigned(pixel_y));   -- n ligne

            -- chaque case de 64x64 pixels est alternativement colorée
            if ((x / 64) mod 2 = (y / 64) mod 2) then
                red   <= "1111";  --case paire rouge
                green <= "0000";
                blue  <= "0000";
            else
                red   <= "0000";
                green <= "0000";
                blue  <= "1111";  --case impaire bleu
            end if;
        else  -- en dehors de cette zone fond noir
            red   <= (others => '0');
            green <= (others => '0');
            blue  <= (others => '0');
        end if;
    end process;
end Behavioral;
```

VGA Sans AXIStream

Simulation

Structure du projet

top_vga_design

- Regroupe les deux sous modules
- Connecte les sig,aux entre eux
- Gère les entrées (horloge reset) et le sosrties (hsync vsync rgb)

```vhdl
entity top_vga is
    Port (
                --entrées vga
        clk       : in  STD_LOGIC;  -- 65 MHz
        reset     : in  STD_LOGIC;
                -- sorties vga
        hsync     : out STD_LOGIC;
        vsync     : out STD_LOGIC;
        red       : out STD_LOGIC_VECTOR(3 downto 0);
        green     : out STD_LOGIC_VECTOR(3 downto 0);
        blue      : out STD_LOGIC_VECTOR(3 downto 0);
        pixel_y   : out STD_LOGIC_VECTOR(9 downto 0)  -- observ position verti simu
    );
end top_vga;

architecture Structural of top_vga is

    signal pixel_x     : STD_LOGIC_VECTOR(10 downto 0);
    signal pixel_y_int: STD_LOGIC_VECTOR(9 downto 0);
    signal video_on    : STD_LOGIC;   --indicateur zone visible
```

VGA Sans AXIStream

Simulation

Structure du projet

tb_top_vga.vhd

Banc de test (testbench), utilisé pour simuler le comportement du circuit VGA dans Vivado sans utiliser de carte réelle.

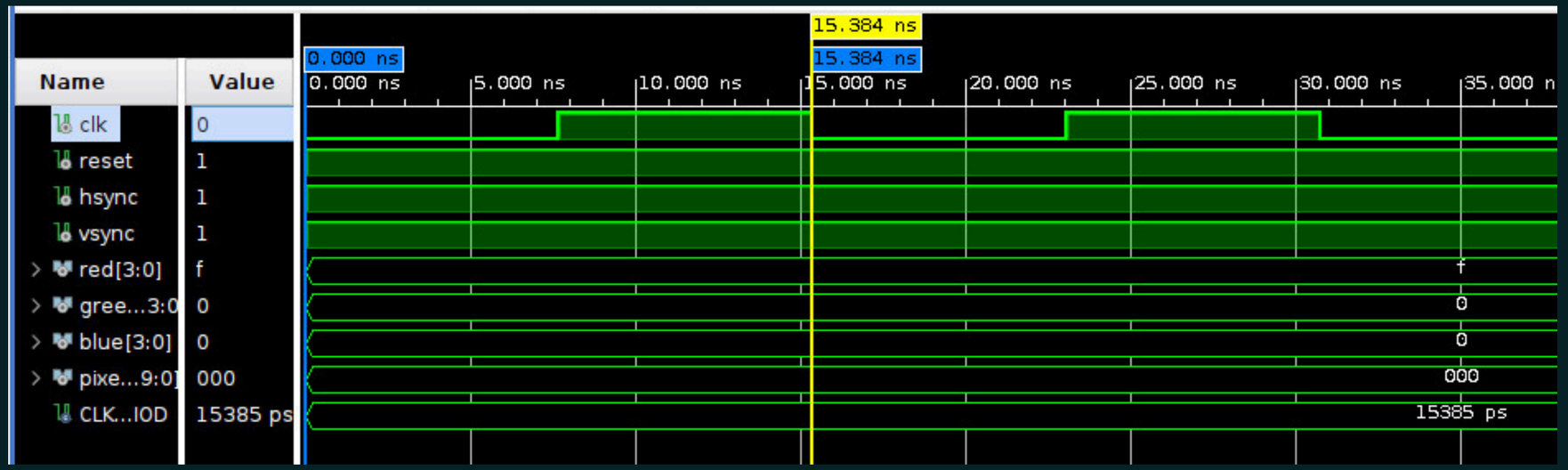
```
architecture Behavioral of tb_top_vga is

    -- Composant à tester
    component top_vga
        Port (
            clk     : in  STD_LOGIC;
            reset   : in  STD_LOGIC;
            hsync   : out STD_LOGIC;
            vsync   : out STD_LOGIC;
            red     : out STD_LOGIC_VECTOR(3 downto 0);
            green   : out STD_LOGIC_VECTOR(3 downto 0);
            blue    : out STD_LOGIC_VECTOR(3 downto 0);
            pixel_y : out STD_LOGIC_VECTOR(9 downto 0)
        );
    end component;

    -- Signaux de test
    signal clk       : STD_LOGIC := '0';
    signal reset     : STD_LOGIC := '1';
    signal hsync     : STD_LOGIC;
    signal vsync     : STD_LOGIC;
    signal red       : STD_LOGIC_VECTOR(3 downto 0);
    signal green     : STD_LOGIC_VECTOR(3 downto 0);
    signal blue      : STD_LOGIC_VECTOR(3 downto 0);
    signal pixel_y   : STD_LOGIC_VECTOR(9 downto 0);

    constant CLK_PERIOD : time := 15.385 ns;  -- 65 MHz frequence utilisé ici
```

VGA Sans AXIStream

Simulation

Observation Periode

Format	Pixel Clock 1
1024x768, 60Hz	65MHz
Zone visible	Fréquence de pixels
Attendues	15.384 ns
	1 / 65MHZ

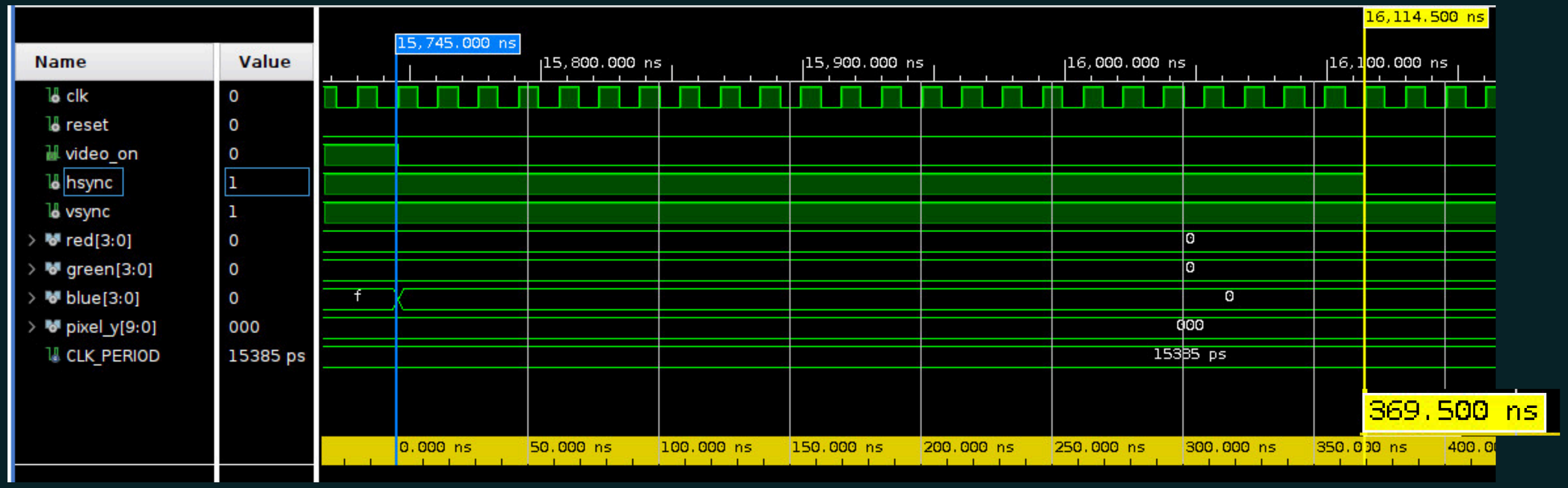


VGA Sans AXIStream

Simulation

Observation H.Front Porch

H.Front Porch
24 px
Pause après ligne visible
369 ns
24 * 15.384ns

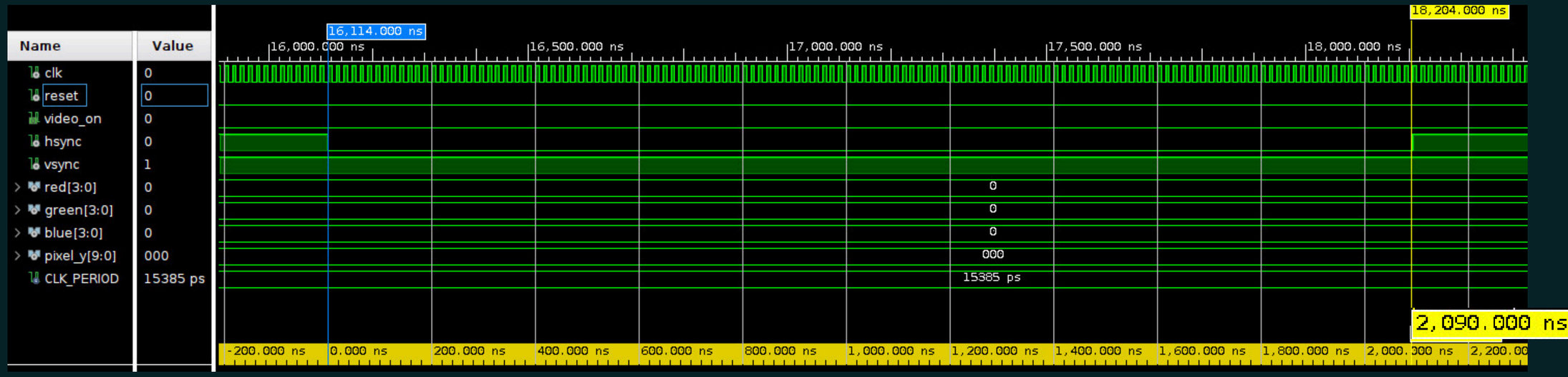


VGA Sans AXIStream

Simulation

Observation H.Sync Pulse

H.Sync Pulse
136 px
Durée du signal hsync à 0
2.09 us
136 * 15.384ns

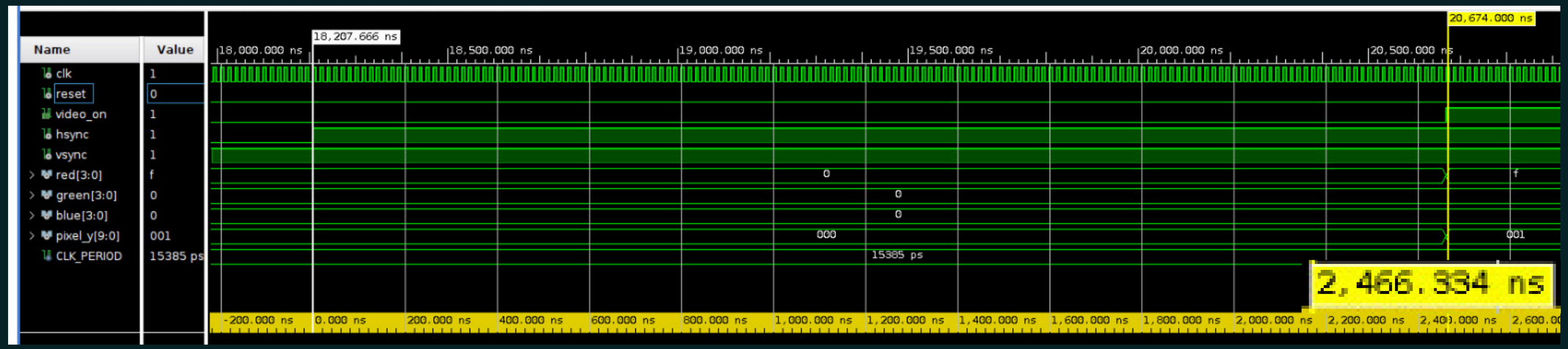


VGA Sans AXIStream

Simulation

Observation H.Back Porch

H.Back Porch
160 px
Pause avant nouvelle ligne
2.46 us
160 * 15.385 ns

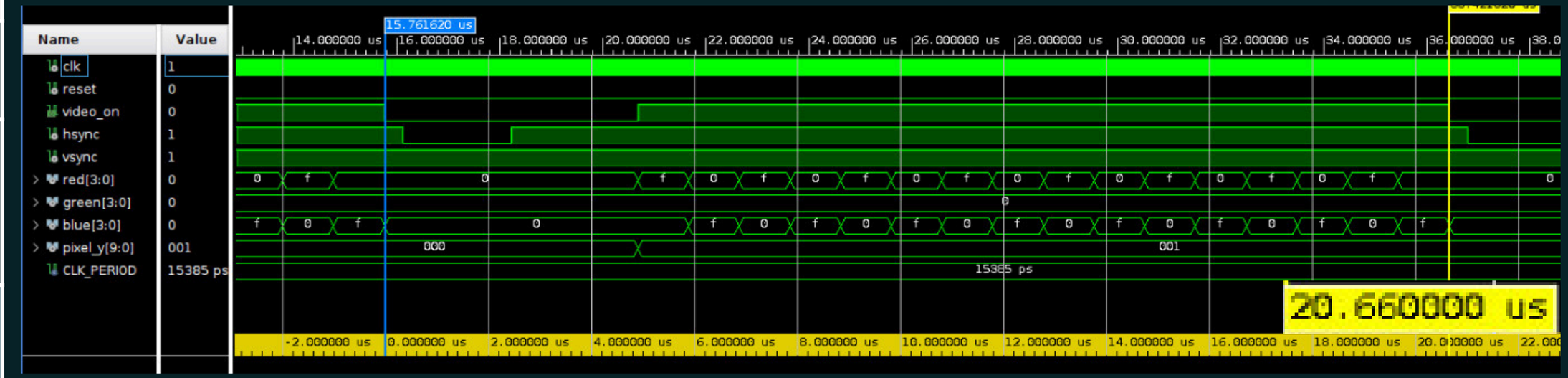


VGA Sans AXIStream

Simulation

Observation H.Total

H.Total
1344 px
1024 + 24 + 136 + 160
20.67 us
1344 * 15.385 ns

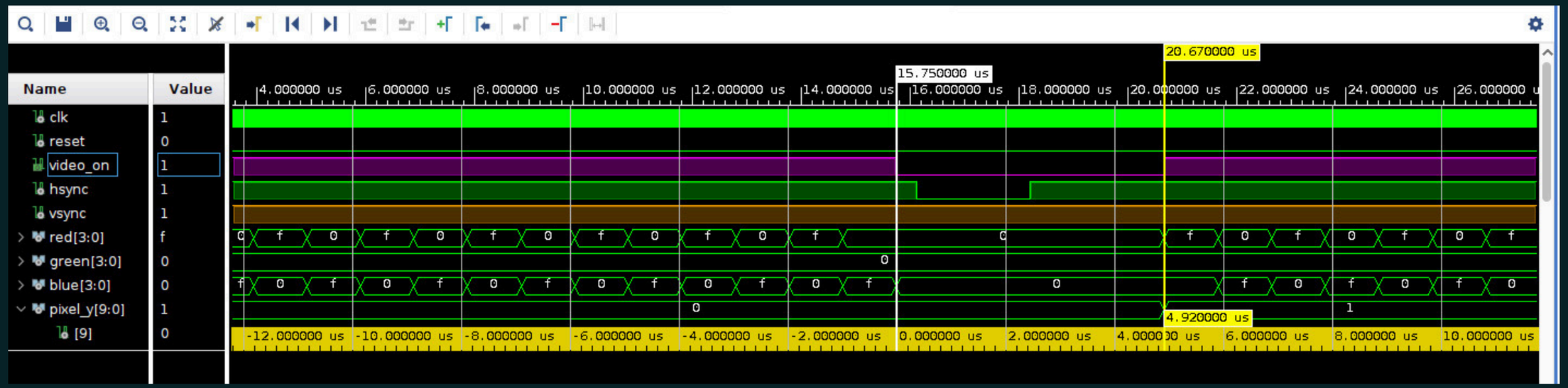


VGA Sans AXIStream

Simulation

Observation RGB

```
        -- chaque case de 64x64 pixels est alternativement colorée
        if ((x / 64) mod 2 = (y / 64) mod 2) then
            red   <= "1111";  --case paire rouge
            green <= "0000";
            blue  <= "0000";
        else
            red   <= "0000";
            green <= "0000";
            blue  <= "1111";  --case impaire bleu
        end if;
    else  -- en dehors de cette zone fond noir
        red   <= (others => '0');
        green <= (others => '0');
        blue  <= (others => '0');
    end if;
```

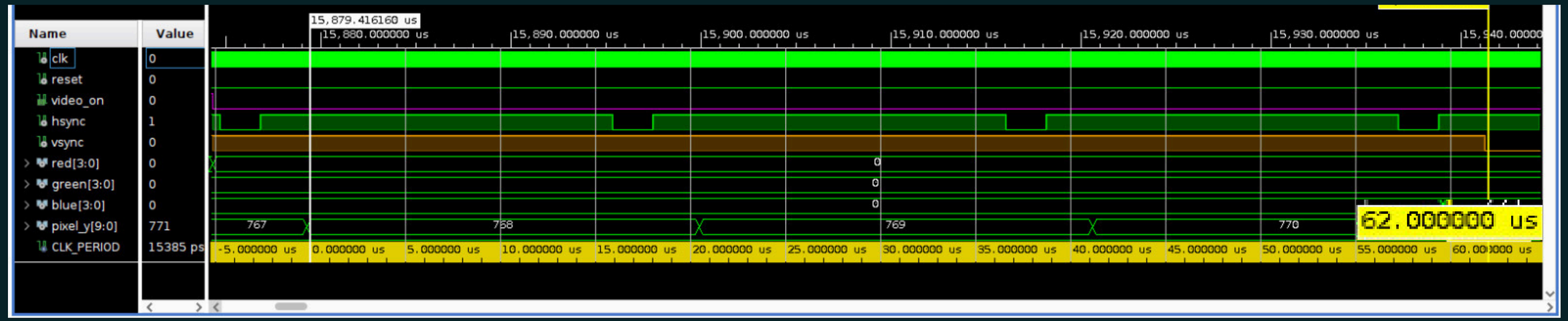


VGA Sans AXIStream

Simulation

Observation V.Front Porch

V.Front Porch
3 lignes
Pause après l'image visible
768 – 769 – 770
vsync → 0
62.03 us

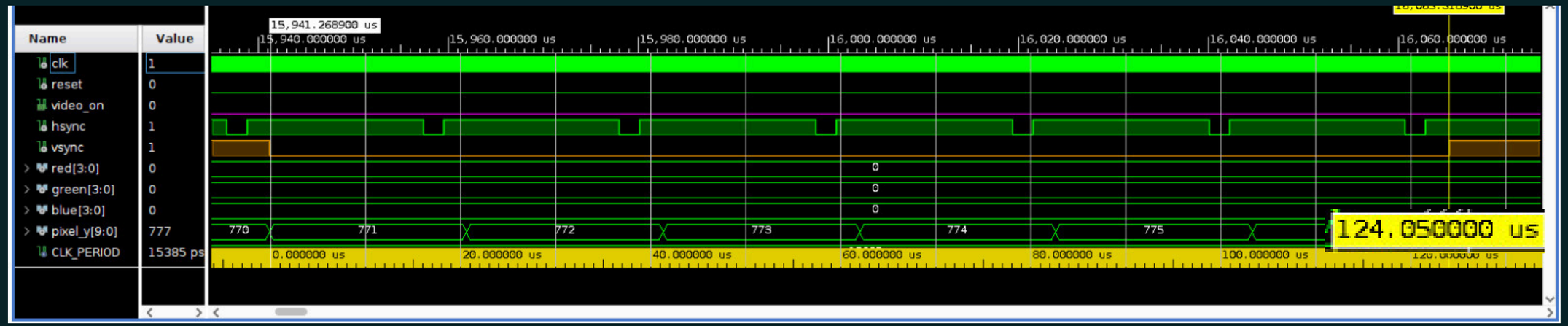


VGA Sans AXIStream

Simulation

Observation V.Sync Pulse

V.Synch Pulse
6 lignes
Durée du signal vsync à 0
771-772-773-774-775-776
vsync → 1
124..06 us

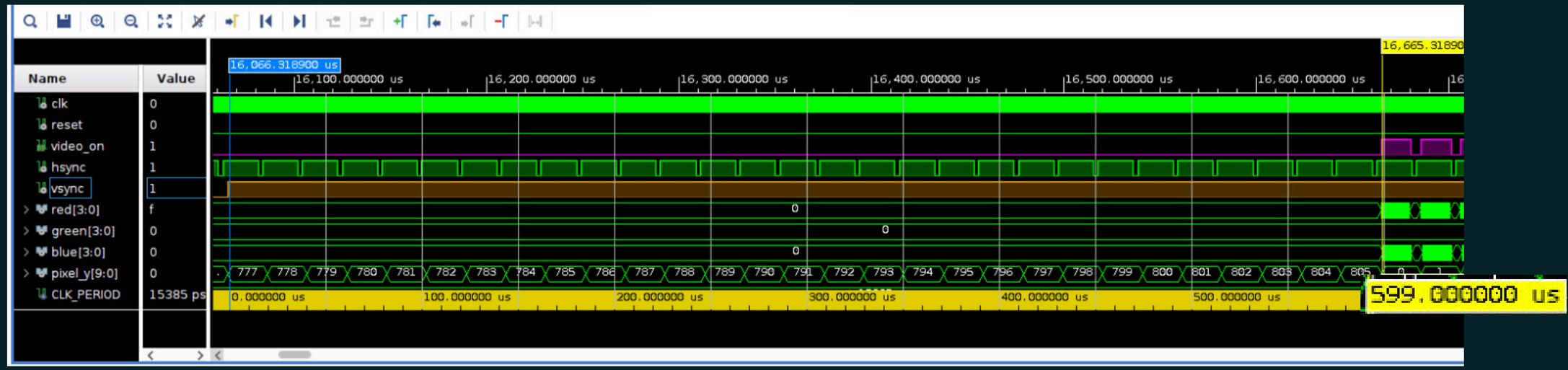


VGA Sans AXIStream

Simulation

Observation V.Sync Pulse

V.Back Porch
29 lignes
Pause avant nouvelle image
776 --- 805
videon → 1
599.65 us

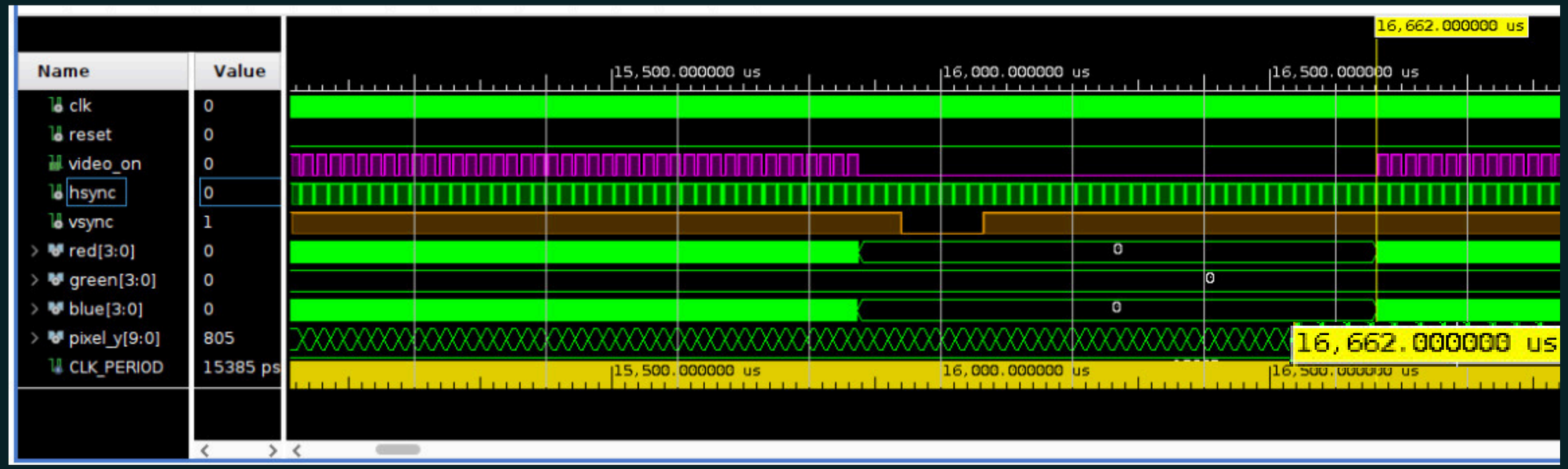


VGA Sans AXIStream

Simulation

Observation V.Total

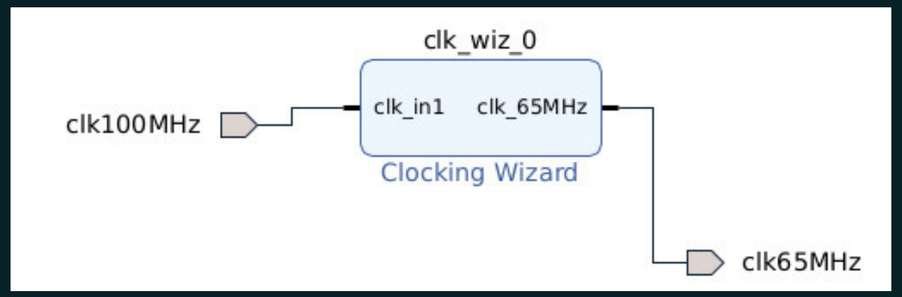
V.Total
806 lignes
768 + 3 + 6 + 29
0 --- 805
reinitialisation
16.66 ms



VGA Sans AXIStream

Implémentation

Structure du projet



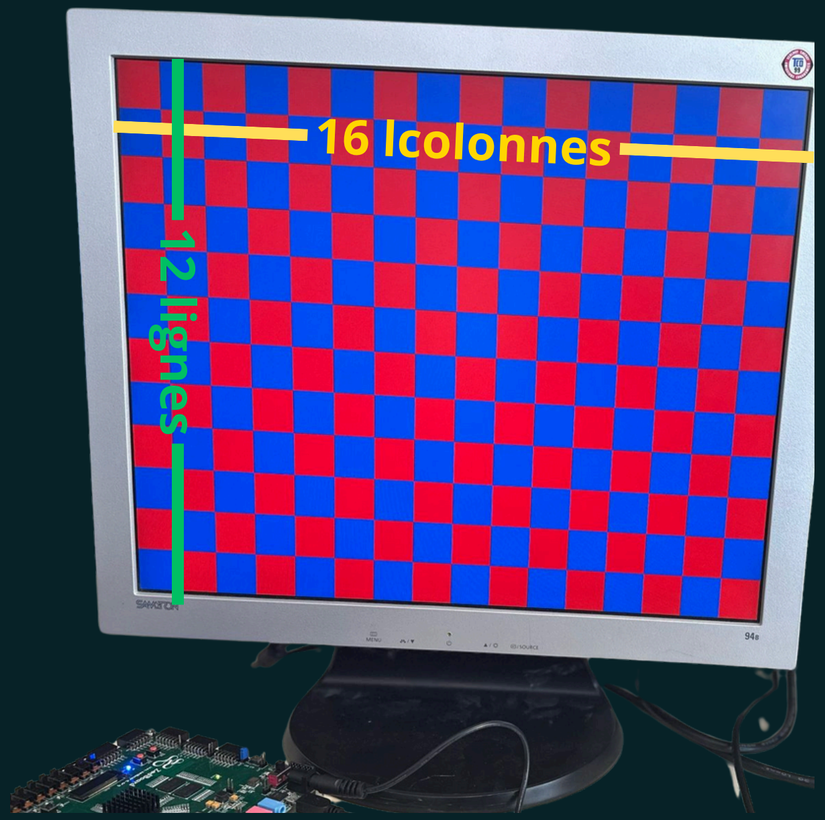
Ajout dun Clocking Wizard pour transformer le signal d'horloge 100 Mhz de la carte en 65 MHz pour notre VGA

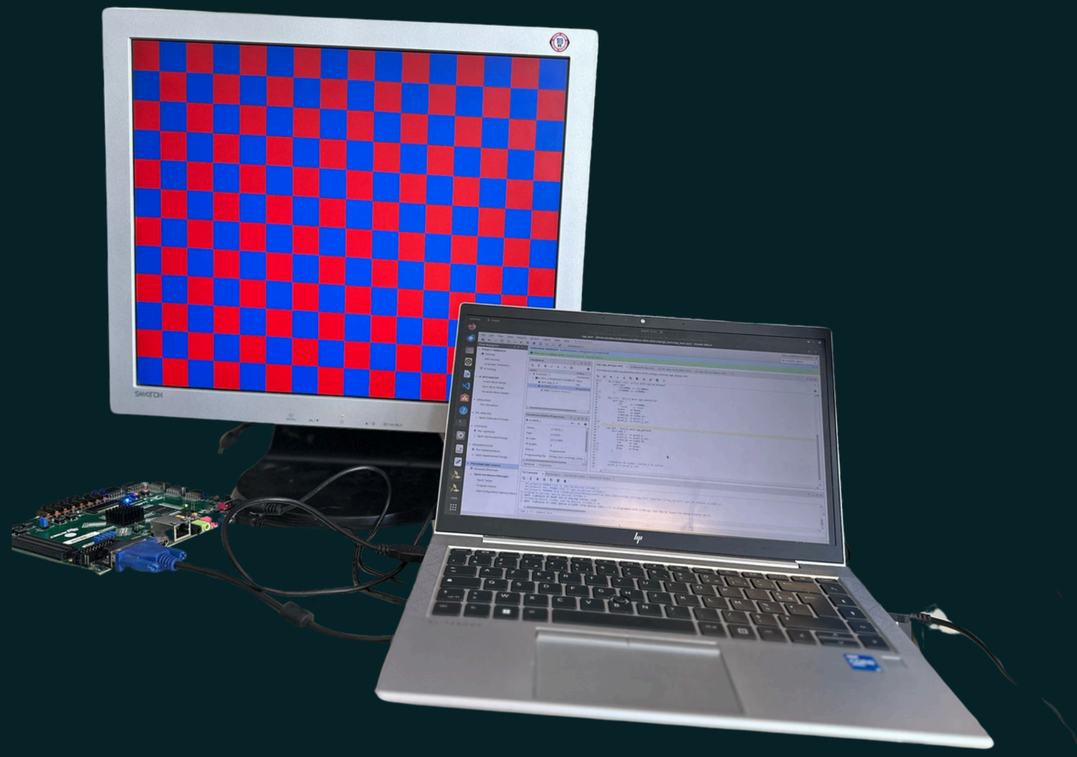
```
begin
    -- Instance du bloc qui convertit 100 MHz en 65 MHz
    bd_wrapper_inst: entity work.vga_bd_wrapper
        port map(
            clk100MHz => clk100MHz,
            clk65MHz => clk65MHz
        );
    -- Contrôleur VGA : fournit hsync, vsync, video_on, pixel_x/y
    vga_ctrl : entity work.vga_controller
        port map (
            clk      => clk65MHz,
            --reset    => reset,
            hsync    => hsync,
            vsync    => vsync,
            video_on => video_on,
            pixel_x  => pixel_x,
            pixel_y  => pixel_y_int
        );

    -- Générateur de couleurs VGA (reçoit AXI + coordonnées)
    vga_gen : entity work.vga_pattern
        port map (
            pixel_x  => pixel_x,
            pixel_y  => pixel_y_int,
            video_on => video_on,
            red      => red,
            green    => green,
            blue     => blue,
```

VGA Sans AXIStream

Implémentation

Observation damier





VGA Avec AXI Stream

Implémentation

Test AXI Stream simulé

Valider le bon fonctionnement de la lecture et de l'affichage via le protocole AXI Stream, sans DMA réel, uniquement avec des valeurs simulées dans le top_vga

```
if video_on = '1' then
    if tvalid = '1' then
        -- Couleur via AXI ( violet avec tdata = x"000(
        red   <= tdata(11 downto 8);
        green <= tdata(7 downto 4);
        blue  <= tdata(3 downto 0);
    else
        -- Fond : damier noir et blanc (64x64)
        if ((to_integer(unsigned(pixel_x)) / 64) mod 2)
            red   <= "1111";
            green <= "1111";
            blue  <= "1111";  -- Blanc
        else
            red   <= "0000";
            green <= "0000";
            blue  <= "0000";  -- Noir
        end if;
    end if;
```

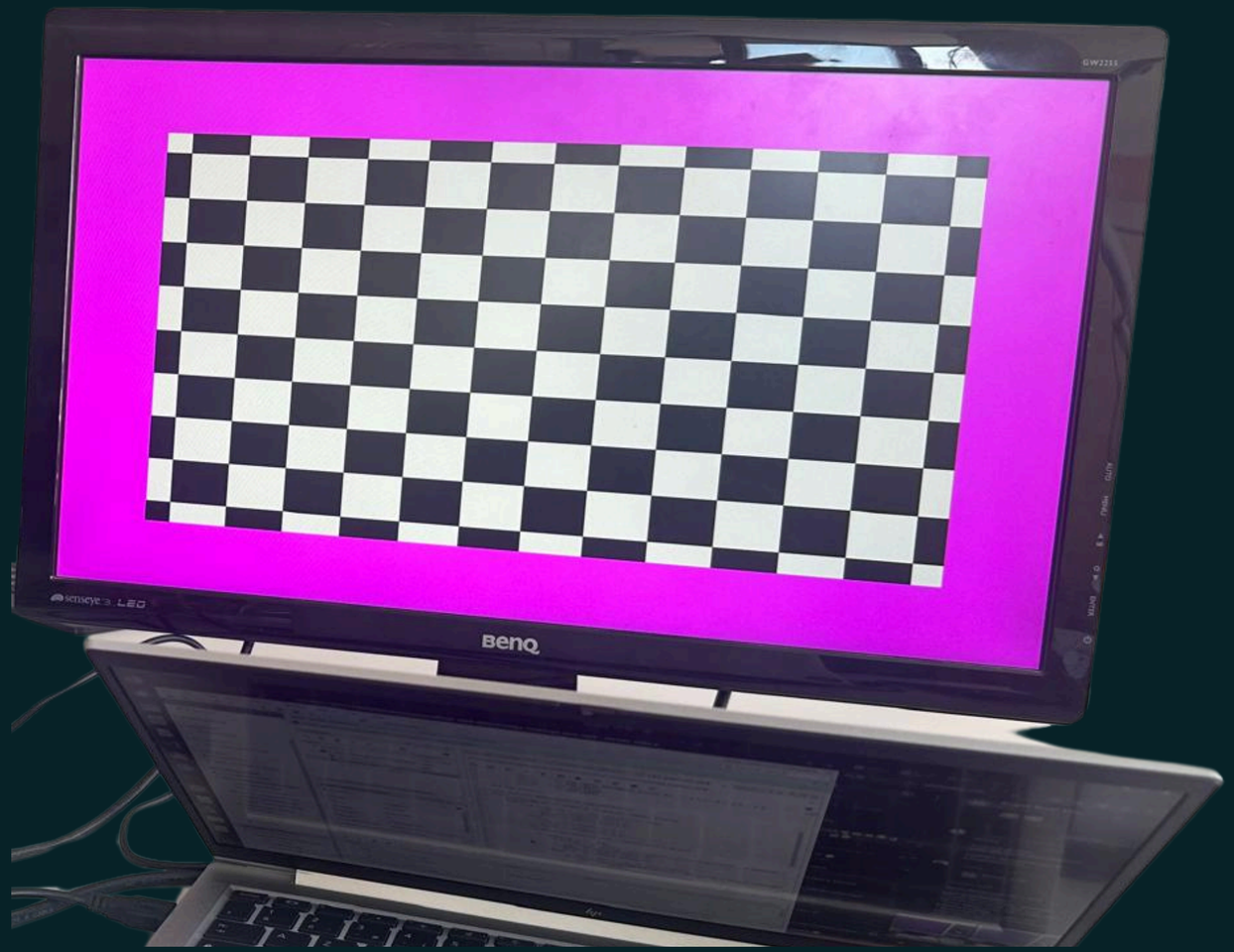
```
begin
    if to_integer(unsigned(pixel_x)) < 100 or
       to_integer(unsigned(pixel_x)) > 924 or
       to_integer(unsigned(pixel_y_int)) < 100 or
       to_integer(unsigned(pixel_y_int)) > 668 then
        tvalid_test <= '1'; -- Cadre
    else
        tvalid_test <= '0'; -- Centre
    end if;
```

force tvalid = 1 (donc active le flux AXI → violet) pour toutes les zones EN DEHORS du rectangle [100,924] x [100,668]

VGA Avec AXI Stream

Implémentation

Test AXI Stream simulé



CONCLUSION