



COMPTE RENDU-
FPGA
M2 EEA
UFR Sciences et Techniques
Année universitaire
2022 - 2023

Réalisé par :
Benjamin TSHALA TSHIBUMBU
Ayikoe Aymar Junior Atayi

Chargés du cours :
Mr. Hiliwi Leake KIDANE

PARTIE I : Demi-additionneur

A la fin de ce tutoriel, nous devrions être en mesure de savoir comment créer un projet ISE sur Project Navigator. Les étapes sont les suivantes :

1. Pour lancer Xilinx ISE, double-cliquez sur l'icône Xilinx ISE sur le bureau.
2. Dans le navigateur de projet ISE, cliquez sur Fichier> Nouveau projet.
3. Dans la fenêtre de l'assistant Nouveau projet, saisissez le nom du projet souhaité.
4. Sélectionnez HDL pour le champ Type de source de niveau supérieur et cliquez sur Suivant.
5. Entrez les valeurs suivantes dans la fenêtre Assistant Nouveau projet - Paramètres du projet.

- Catégorie de produit : Tous
- Famille : Spartan6
- Appareil : XC6SLX16
- Paquet : CSG324
- Vitesse : -3
- Outil de synthèse : XST (VHDL / Verilog)
- Simulateur : Isim (VHDL / Verilog)
- Langue préférée : VHDL

6. Enfin, cliquez sur Suivant.

7. Enfin, nous cliquons sur Terminer dans la fenêtre Résumé du projet.

Après avoir terminé cette étape, nous devons maintenant ajouter le fichier source (haut niveau HDL) à notre projet. Pour ce faire, nous devons suivre les étapes suivantes :

- Cliquons sur Projet> NOUVELLE Source.
- Sélectionnez le module VHDL dans la fenêtre Project Wizard - Select Source Type.
- Donnez un nom à votre fichier et vérifiez que la case Ajouter au projet est cochée. Cliquons sur Suivant.
- Créons un port d'entrée A, B et une sortie D comme indiqué sur la figure.

Une fois la création du fichier HDL de niveau supérieur terminée, la fenêtre suivante doit s'ouvrir dans l'espace de travail du Navigateur du projet :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity logic_gate is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          C : out STD_LOGIC;
          D : out STD_LOGIC);
end logic_gate;
```

```

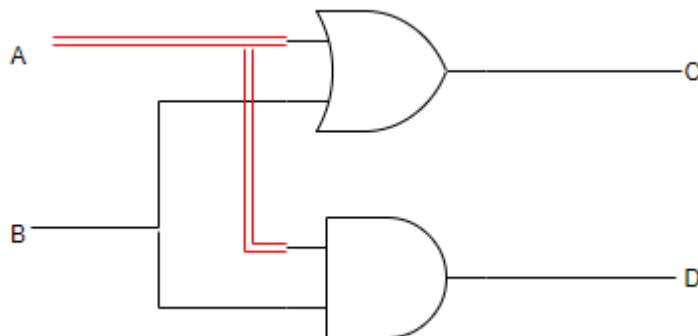
architecture Behavioral of logic_gate is
begin
end Behavioral;

```

Le but de ce TP est de réaliser la porte logique ET qui donne comme résultat 1 quand on multiplie les 1 par le 1 mais on obtient 0 quand on multiplie 0 zéro par 1 ou 0. Sous forme de table de vérité, on aura :

A	B	D	C
1	0	0	1
0	0	0	0
1	1	1	1
0	1	0	1

Les portes logiques qui traduit cette table de vérité est la suivante :



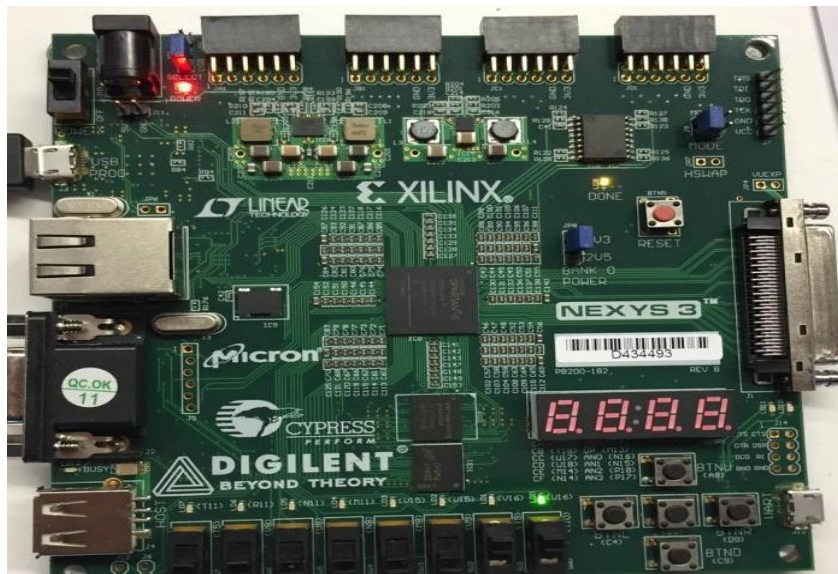
Le code VHDL complet est le suivant :

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity logic_gate is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          C : out STD_LOGIC;
          D : out STD_LOGIC);
end logic_gate;
architecture Behavioral of logic_gate is
begin
    D <= A and B;
    C <= A OR B;
end Behavioral;

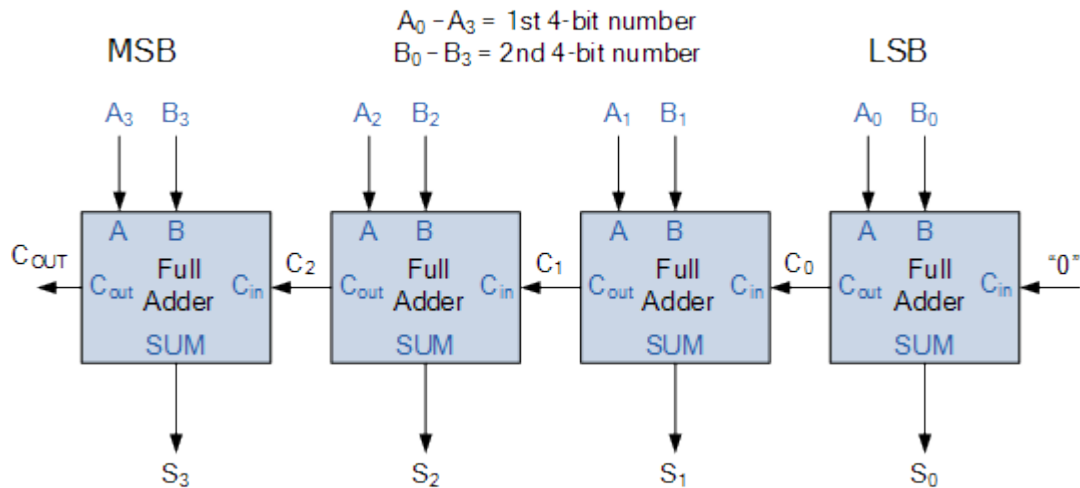
```

Résultats et constats :



Comme vous pouvez le voir, pour l'opération ET (D dans le code), la LED U16 est allumée lorsque l'on appuie sur les interrupteurs T10 et T9.

PARTIE II : Additionneur complet



Le circuit se compose de 4 additionneurs complets puisque nous effectuons une opération sur des nombres à 4 bits. Il peut être représenté comme sur la figure ci-dessus.

En code VHDL, nous avons ceci :

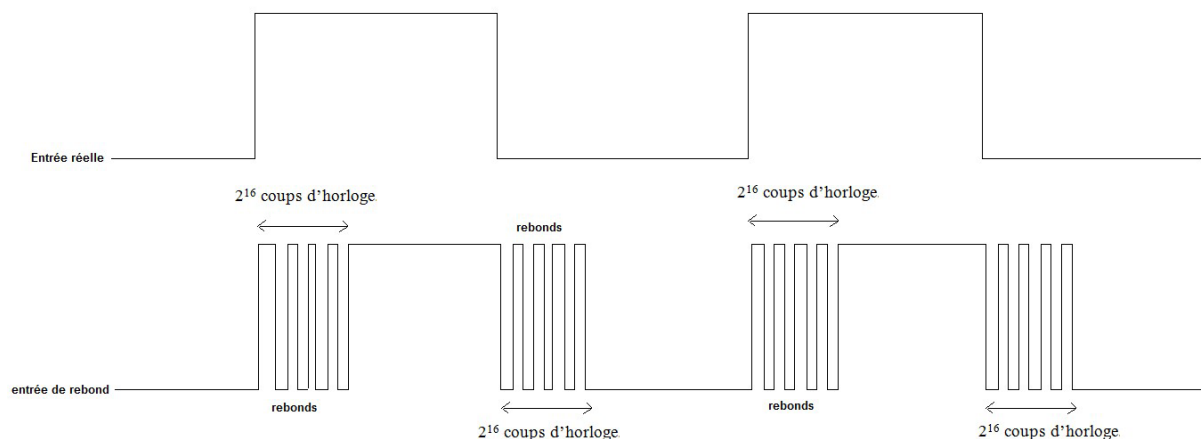
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity AdditionneurComplet is
    Port ( a : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
          b : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
          r : in  STD_LOGIC;
          cin : in  STD_LOGIC;
          cout : out  STD_LOGIC;
          s : out  STD_LOGIC_VECTOR(3 DOWNTO 0));
end AdditionneurComplet;
architecture AdditionneurComplet of AdditionneurComplet is
    signal c : in  STD_LOGIC_VECTOR(4 DOWNTO 0);
begin
    c (0) <= cin;
    adder_gen : for i in 0 to 3 generate

        s (i) <= a(i) xor b(i) xor c (i);
        c (i+1) <= (c(i) and b(i)) or (c(i) and a(i)) or (a(i) and b(i));
    end generate;
    cout <= c(4)
end;
```

end AdditionneurComplet;

PARTIE III Comment utiliser les boutons ?

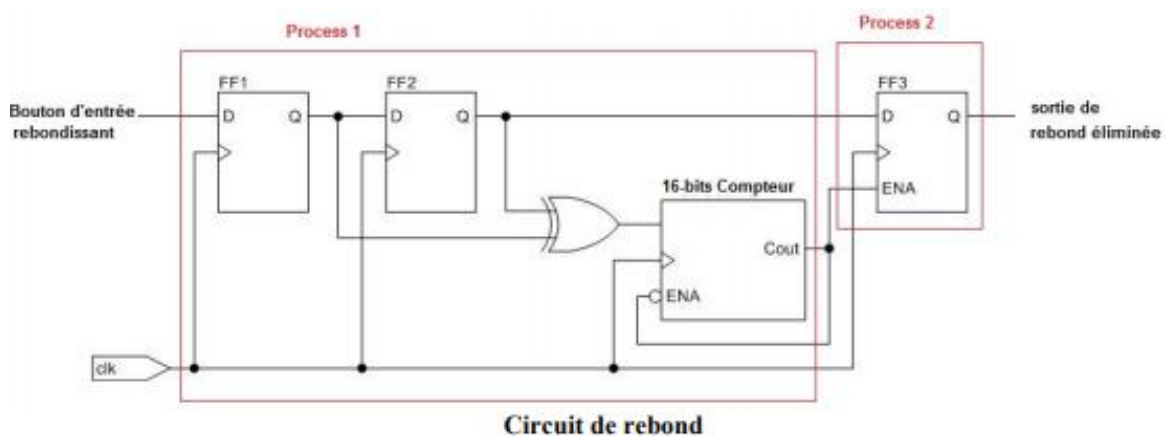
Il y a un phénomène de rebond lors de l'utilisation de boutons mécaniques. En effet, dès qu'on appuie sur un bouton, le contact se produit pour générer des "rebonds" comme le montre l'image ci-dessous. Ces rebonds peuvent être nuisibles si jamais un compteur est connecté à un front montant du signal créé par le bouton.



Ici, on nous demande de créer un programme VHDL avec deux niveaux de hiérarchie.

Tout d'abord, nous devons créer un composant qui éliminera les rebonds sur 5 boutons différents. Ce composant recevra en entrée les états des 5 boutons et l'horloge globale à 100 MHz et sortira les états des 5 boutons après filtrage.

Un état stable est considéré pour un bouton s'il ne change pas pendant 216 coups d'horloge.



On commence pour créer pour chaque bouton, deux process :

- a) Le premier process consiste à incrémenter un compteur lorsque la valeur courante du bouton est la même que celle du bouton mémorisé (si non le compteur est remis à 0).
- b) Le seconde process changera la valeur du bouton mémorisé si le compteur précédent est arrivé à sa valeur maximum.

Ensuite, il faut valider le fonctionnement sur la carte en allumant et éteignant les LED en fonction de l'appui sur les boutons en temps réel.

Code VHDL :

Pour la réalisation du programme VHDL, nous avons réalisé le code VHDL de chaque composant décrit du schéma ci-dessus :

Bascule D

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
entity Bascule_D is
Port ( a : in STD_LOGIC;
raz : in STD_LOGIC;
s : out STD_LOGIC;
clock : in STD_LOGIC);
end Bascule_D;
architecture Behavioral of Bascule_D is
begin
process(clock,raz)

begin
if raz='1' then
s<='0';
elsif clock'event and clock='1' then
s<=a;
end if;
end process;
end Behavioral;
```

Porte XOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
entity port_log_xr is
Port ( a : in STD_LOGIC;
b : in STD_LOGIC;
s : out STD_LOGIC);
end port_log_xr;
architecture Behavioral of port_log_xr is
begin
s<=a XOR b;
end Behavioral;
```

Compteur 16 bits

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
entity Compt16_b is
Port ( a : in STD_LOGIC;
ENA: in STD_LOGIC;
s : out STD_LOGIC_VECTOR (15 downto 0);
clock : in STD_LOGIC);
end Compt16_b;
architecture Behavioral of Compt16_b is
signal s_int: STD_LOGIC_VECTOR ( 15 downto 0):=(others=>'0');
begin
process(clock,a)
begin
if ENA<='0' then
if a = '0' then
s_int <= (others => '0');
elsif clock'event and clock='1' then
s_int <= s_int + 1;
if s_int<="0000000000000001" then
s<= s_int;
end if;
end if;
end if;
end process;
end Behavioral;
```

Bascule D avec Enable

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Bascule_D_enable is
Port ( a : in STD_LOGIC;
ENA:in STD_LOGIC;
raz : in STD_LOGIC;
s : out STD_LOGIC;
clock : in STD_LOGIC);
end Bascule_D_enable;
architecture Behavioral of Bascule_D_enable is
begin
process(clock,raz)
begin
if raz='1' then
s<='0';
elsif clock'event and clock='1' then
if ENA='0' then
s<='0';
elsif ENA='1' then
s<=a;
end if;
end if;
end process;
```


Ensuite, nous avons assemblés tous les codes VHDL dans un *package* ci-dessous :

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package ensemble is
component Bascule_D is
Port ( a : in STD_LOGIC;
raz : in STD_LOGIC;
s : out STD_LOGIC;
clock : in STD_LOGIC);
end component;
component Compt16_b is
Port ( a : in STD_LOGIC;
s : out STD_LOGIC_VECTOR (15 downto 0);
ENA :in STD_LOGIC;
clock : in STD_LOGIC);
end component;
component port_log_xr is
Port ( a : in STD_LOGIC;
b : in STD_LOGIC;
s : out STD_LOGIC);
end component;
component Bascule_D_enable is
Port ( a : in STD_LOGIC;
raz : in STD_LOGIC;
ENA : in STD_LOGIC;
s : out STD_LOGIC;
clock : in STD_LOGIC);
end component;
end ensemble;

A la fin nous avons le code final :
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
use work.ensemble.all;
entity tp1_bouton is
Port ( a : in STD_LOGIC;
s : out STD_LOGIC;
clock : in STD_LOGIC);
end tp1_bouton;
architecture Behavioral of tp1_bouton is
signal s1,s2,s3,s5: std_logic :='0';
signal s4: STD_LOGIC_VECTOR (15 downto 0):=(others=>'0'); TP_FPGA

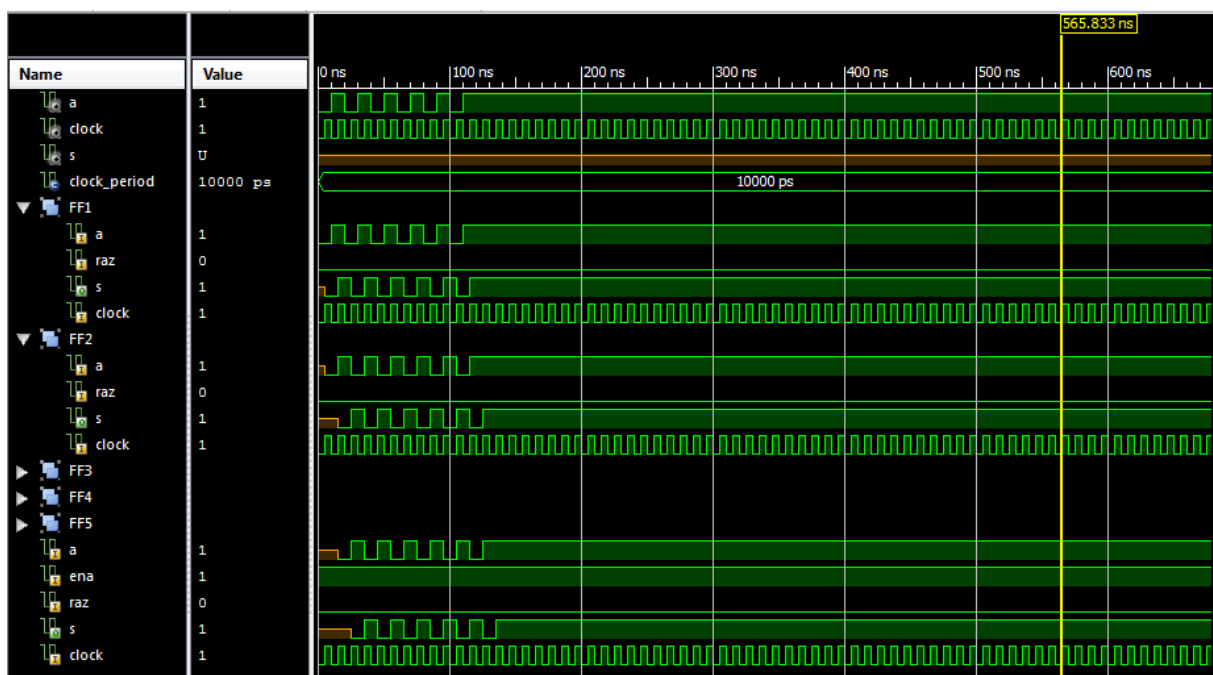
signal ena : std_logic :='1';
begin
inst_FF1: Bascule_D
port map(
a=> a,
clock=> clock,
raz=> '0',
s=> s1
);
```

```

inst_FF2: Bascule_D
port map(
a=> s1,
clock=> clock,
raz=> '0',
s=> s2
);
inst_porte_logique:port_log_xr
port map(
a=> s2,
b=> s1,
s=> s3
);
inst_compteur:compt16_b
port map(
a=> s3,
ENA=> ena,
clock=> clock,
s=> s4
);
inst_process2: Bascule_D_enable
port map(
a=> s2,
ENA=> ena,
clock=> clock,
raz=> '0',
s=> s5
);
end Behavioral;

```

Simulation, Interprétation :



Nous avons bien simulé et nous avons constaté que le signal de sortie (FF5) les rebonds ont été éliminés mais était décalé à cause dans un effet synchro de l'horloge.

PARTIE IV : 7Segment

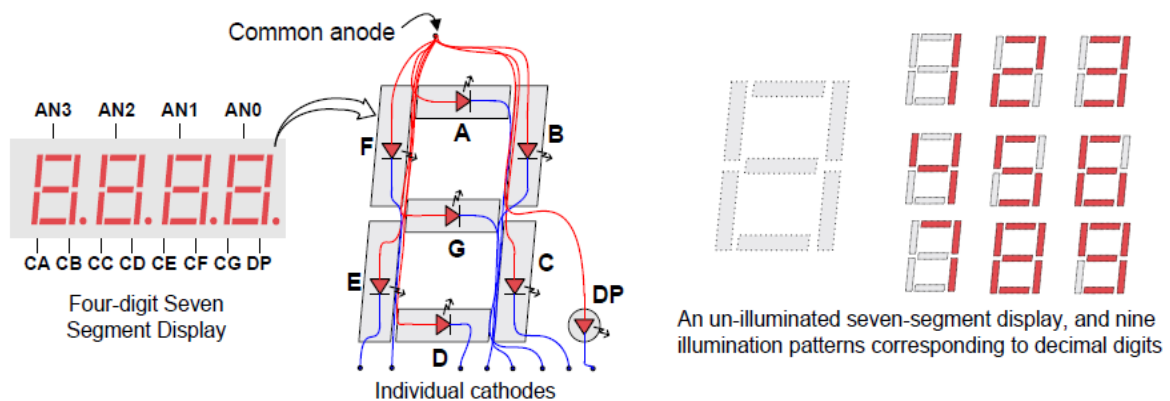
Objectif :

L'objectif c'est d'utiliser les afficheurs 7 segments de la carte FPGA Xilinx Spartan 6.



Pour atteindre ces objectifs, il faut suivre certaines étapes :

- 1) A l'aide de la documentation de la carte Nexys 3, repérez comment afficher une valeur entre 0 et 15 sur un afficheur :

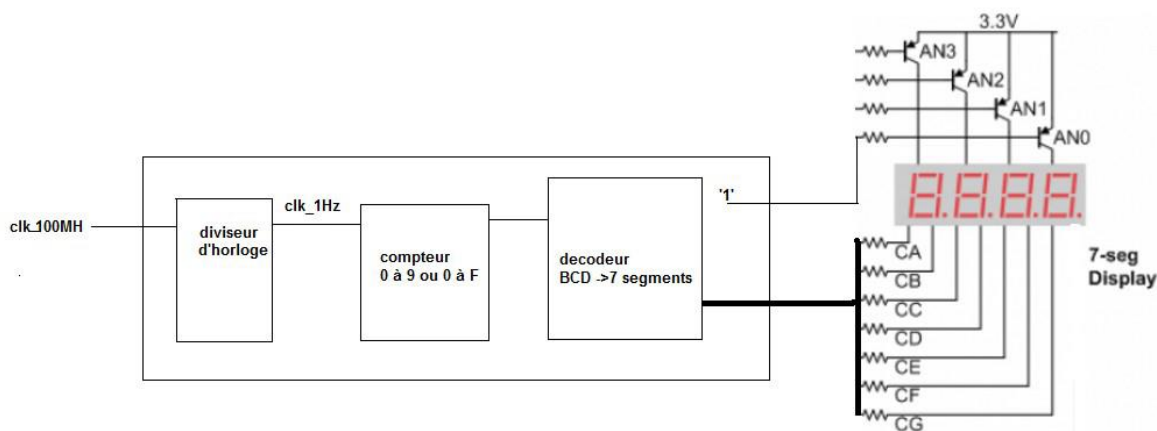


Ensuite, remplir les valeurs des segments sur la table de vérité suivante :

Chiffre	Affichage	a	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
1	1	0	1	1	0	0	0	0
2	2	1	1	0	1	1	0	1
3	3	1	1	1	1	0	0	1
4	4	0	1	1	0	0	1	1
5	5	1	0	1	1	0	1	1
6	6	1	0	1	1	1	1	1

7	7	1	1	1	0	0	0	0
8	8	1	1	1	1	1	1	1
9	9	1	1	1	1	0	1	1
10	A	1	1	1	1	0	1	1
11	B	1	1	1	1	1	1	1
12	C	1	0	0	1	1	1	0
13	D	1	1	1	1	1	1	0
14	E	1	0	0	1	1	1	1
15	F	1	0	0	0	1	1	1

Schéma à suivre :



2) Créer un compteur dont valeur sera affichée de 0 à 15 sur un seul des 4 digits de l’afficheur 7-segments.

- Incréméntation sur le front montant d’un bouton poussoir.
- Décréméntation sur le front descendant d’un autre bouton poussoir.
- Mise à zéro via un autre bouton poussoir.

3) Modifier votre programme afin de pouvoir déplacer le digit actif sur la gauche, et sur la droite en fonction de bouton poussoir (des boutons droite et gauche par exemple).

4) Modifier votre programme afin d’incrémenter toutes les $\frac{1}{2}$ seconde votre compteur.

5) Faire évoluer le programme VHDL de manière que les 4 digits soient utilisés simultanément. L’afficheur devra alors afficheur les nombres de 0 jusqu’à 9999. On pourra commencer par une version hexadécimale (de 0 à ffff).

Code VHDL :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std;
entity a_seg is
GENERIC
(
N : positive :=8
);
PORT (
clk ,IN1, IN2, IN3, IN4 : IN std_logic;
TransEN1 : OUT std_logic;
TransEN2 : OUT std_logic;
TransEN3 : OUT std_logic;
TransEN4 : OUT std_logic;
Seg_value : OUT std_logic_vector(N-1 downto 0);
LED : OUT std_logic_vector(6 downto 0):="1000000"
);
end a_seg;
architecture Behavioral of a_seg is
SIGNAL sel : std_logic_vector(3 downto 0);
SIGNAL Seg_temp : std_logic_vector(N-1 downto 0):=x"FF";
type T_DATA is array (0 to 7) of std_logic_vector(N-1 downto 0);
-- Anode commune : Tableau des valeurs en BCD 7 Segments
constant SEG_7 : T_DATA :=
(x"40",
x"79",
x"24",
x"30",
x"19",
x"12",
x"02",
x"40");

BEGIN
PROCESS (clk,IN1, IN2, IN3, IN4)
BEGIN
IF (clk'EVENT AND clk='1') THEN
CASE sel IS

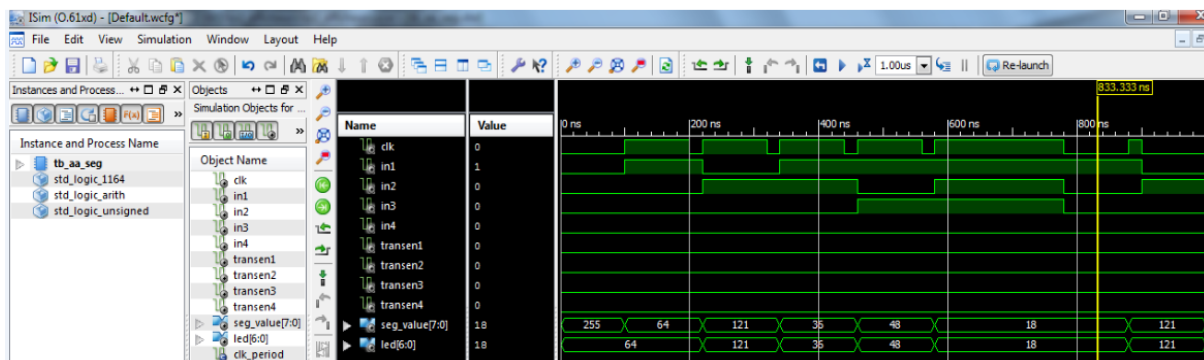
WHEN x"0" => Seg_temp<=SEG_7 (0);      LED <="1000000";
WHEN x"1" => Seg_temp<=SEG_7 (1); LED <="1111001";
WHEN x"2" => Seg_temp<=SEG_7 (2); LED <="0100100";
WHEN x"3" => Seg_temp<=SEG_7 (3); LED <="0110000";
WHEN x"4" => Seg_temp<=SEG_7 (4); LED <="0011001";
WHEN x"5" => Seg_temp<=SEG_7 (5); LED <="0010010";
WHEN x"6" => Seg_temp<=SEG_7 (6); LED <="0000010";
WHEN OTHERS => Seg_temp<=SEG_7 (0); LED <="1000000";
END CASE ;
END IF;
```

```

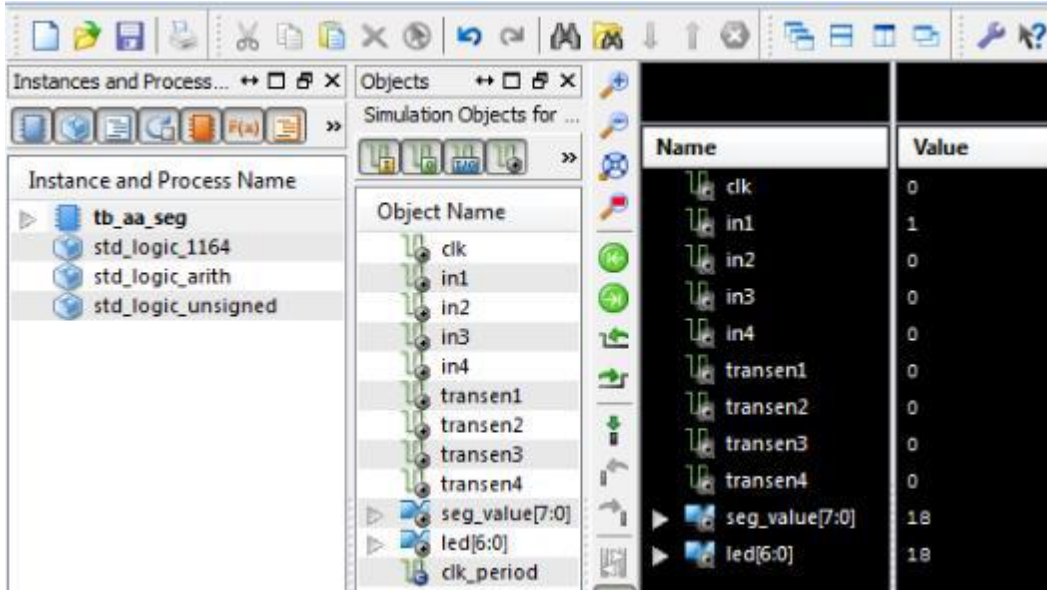
END PROCESS;
Seg_value<=Seg_temp;
-- Le signal sel est la concaténation des 4 entrées
sel <=IN4&IN3&IN2&IN1;
-- Activation des 3 Afficheurs BCD 7 Segments
TransEN1 <= '0';
TransEN2 <= '0';
TransEN3 <= '0';
TransEN4 <= '0';
end Behavioral;

```

Simulation, Interprétation :



Avec plus de précision pour questions de lisibilité et netteté, l'image ci-dessus sera :





PARTIE V : Traitement d'image

Le but de cette partie est d'effectuer des traitements embarqués sur FPGA. Ce traitement d'image aura pour fonction de traiter une image qui a été inversée dans le FPGA.

Inversion d'une image à l'aide d'une table de consultation (LUT)

A partir de la table de correspondance suivante, nous allons générer une mémoire ROM afin de la simuler :

Adresse	Valeur
1	255
2	254
...	...
...	...
254	2
255	1
256	0

1. Rom memory generation

The screenshot shows the 'Block Memory Generator' tool window. The title bar reads 'Block Memory Generator'. The interface is divided into several sections:

- IP Symbol:** A diagram of a purple rectangular block with input and output ports. Inputs on the left include ADDR[2:0], ENA, REGCEA, RSTA, CLKA, INJECTSBITERR, and INJECTDBITERR. Outputs on the right include DOUTA[2:0], SBITERR, DBITERR, and RDADDRECC[2:0].
- Component Name:** A text field containing 'blk_mem_generator'.
- Interface Type:** A dropdown menu with 'Native' selected and 'AXI4' as an alternative option.
- Mode:** A dropdown menu with 'Stand Alone' selected.
- Text Area:** Contains descriptive text about the Native Interface Block Memory Generator (BMG) and its capabilities, including support for Single Port RAM (SP), Simple Dual Port RAM (SDP), True Dual Port RAM (TDP), and Single Port ROM (SP ROM) configurations, as well as features like SoftECC/ECC, Pipeline Stages, and file-based memory initialization.
- Footer:** Includes a 'Datasheet' button, a page indicator 'Page 1 of 6', and navigation buttons '< Back', 'Next >', 'Generate', 'Cancel', and 'Help'.

2. Memory update with .COE file

The screenshot shows the Block Memory Generator tool interface. On the left, the IP Symbol is displayed with inputs: ADDR[2:0], ENA, REGCEA, RSTA, CLKA, INJECTSBITERR, and INJECTDBITERR. On the right, the Optional Output Registers section shows Port A settings. The Memory Initialization section has the Load Init File checkbox checked, and the Coe File is set to .vrom_inv.COE. A COE File Contents dialog is open, showing a Radix of 10 and a COE Vector named memory_initialization_vector. The dialog contains a table with Index and Value columns.

Index	Value
0	7
1	6
2	5
3	4
4	3
5	2
6	1
7	0

Le code VHDL complet pour lire les valeurs LUT stockées dans la ROM est :

The screenshot shows the ISE Project Navigator with the VHDL code for the Traitement_image_top_level entity. The code includes library declarations, port declarations, component declarations, and the main behavioral architecture.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.numeric_std.all;
4
5 entity Traitement_image_top_level is
6
7 port (
8     ADDR : in std_logic_vector (2 downto 0); --input addr
9     an : out std_logic_vector (3 downto 0); -- anode
10    CLK : in std_logic; --system clock
11    CAT : out std_logic_vector (6 downto 0) --cathode hosts 7segment output
12 );
13
14 end Traitement_image_top_level;
15
16 architecture Behavioral of Traitement_image_top_level is
17
18
19 component blk_mem_generator is
20
21 port (
22     ADDR : in std_logic_vector (2 downto 0);-- room input addr
23     CLKA : in std_logic; -- room input clock
24     DOUTA : out std_logic_vector (7 downto 0) -- room output
25 );
26 end component;
27
28 component seven_segment is
29 port (
30     sseg_in : in std_logic_vector(2 downto 0); -- 7 segment input
31
32     sseg: out std_logic_vector (6 downto 0)); -- 7 segment output
33 end component;
34
35
36
37 signal sig : std_logic_vector(2 downto 0); -- signal that links the room output to :
38 begin
39
40 unit1: entity work.blk_mem_generator
41 port map (
42     ADDR => ADDR,
```

Résultats et interprétation :



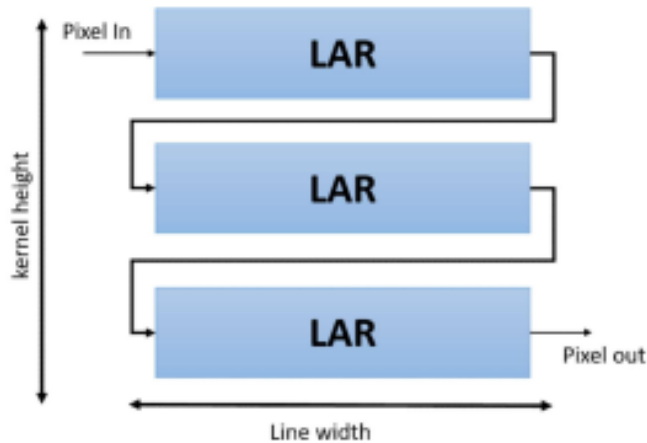
- (1) Les commutateurs hauts représentent les adresses LUT tandis que les LED allumées représentent les valeurs correspondantes lorsque leurs commutateurs sont à l'état haut
- (2) sseg_in recevra en entrée DOUTA la sortie de la mémoire et affichera ces valeurs à l'écran comme indiqué sur les images ci-dessus à droite.

PARTIE VI : Créer un filtre 2D

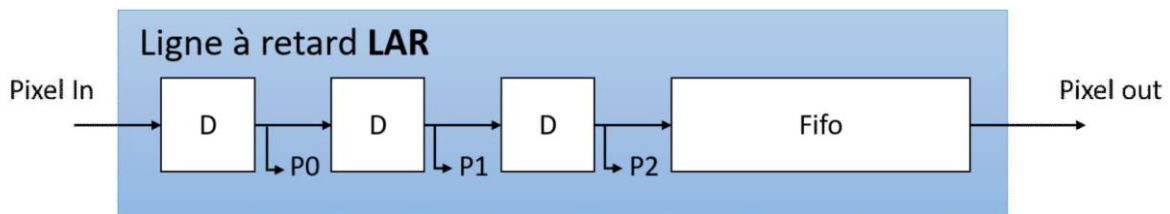
Cette partie vise à créer un filtre 2D 3×3 qui remplira la fonction de notre choix sur l'image d'entrée. Ainsi, il est indispensable de fournir à la partie opérateur les pixels faisant partie du voisinage par rapport auquel les calculs sont effectués. Il faut donc construire un composant GeneV qui nous permettra bien sûr de fournir ce voisinage de 9 pixels.

Ainsi, pour créer ce filtre, nous allons d'abord concevoir une ligne à retard, puis l'instancier 3 fois dans un composant appelé GeneV. De plus, concernant la conception de la ligne Delay, nous devons instancier les bascules 3D et la mémoire FIFO dans un composant appelé LAR.

Le draft GeneV sera représenté comme suit :

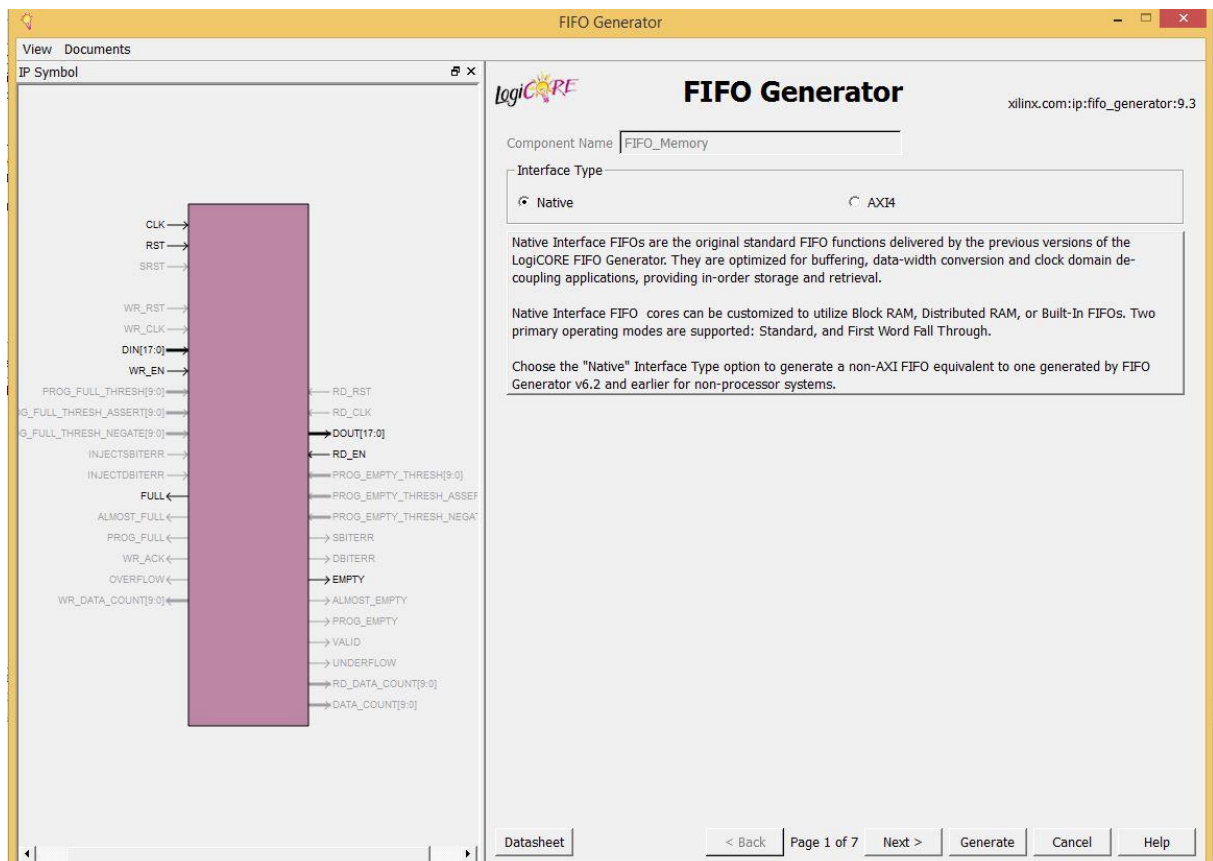


Sur l'image qui suit on peut voir la constitution d'une LAR :



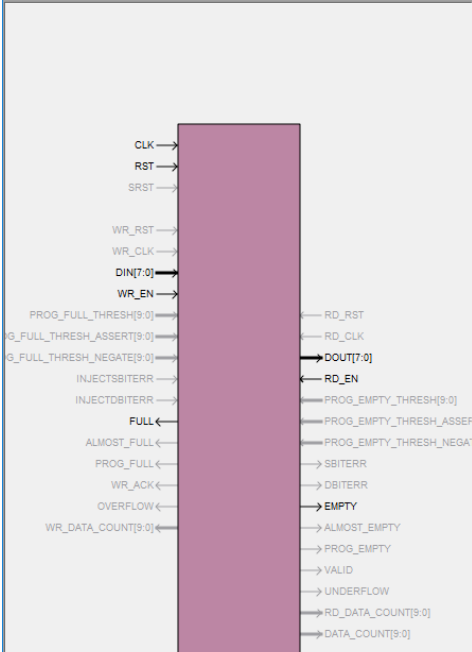
Nous utiliserons 3 bascules D 8 bits, 3 bascules D 1 bit et une mémoire FIFO qui devra être configurée.

FIFO génération :




View Documents

IP Symbol



The diagram shows the FIFO Generator IP symbol with various input and output ports. Inputs on the left include CLK, RST, SRST, WR_RST, WR_CLK, DIN[7:0], WR_EN, PROG_FULL_THRESH[9:0], PROG_FULL_THRESH_ASSERT[9:0], PROG_FULL_THRESH_NEGATE[9:0], INJECTSBITERR, INJECTDBITERR, FULL, ALMOST_FULL, PROG_FULL, WR_ACK, OVERFLOW, and WR_DATA_COUNT[9:0]. Outputs on the right include RD_RST, RD_CLK, DOUT[7:0], RD_EN, PROG_EMPTY_THRESH[9:0], PROG_EMPTY_THRESH_ASSERT, PROG_EMPTY_THRESH_NEGATE, SBITERR, DBITERR, EMPTY, ALMOST_EMPTY, PROG_EMPTY, VALID, UNDERFLOW, RD_DATA_COUNT[9:0], and DATA_COUNT[9:0].

 **FIFO Generator** xilinx.com:ip:fifo_generator:9.3

Read Mode

☒ Standard FIFO
☐ First-Word Fall-Through

Built-in FIFO Options

The frequency relationship of WR_CLK and RD_CLK MUST be specified to generate the correct implementation.

Read Clock Frequency (MHz) Range: 1..1000

Write Clock Frequency (MHz) Range: 1..1000

Data Port Parameters

Write Width Range: 1,2,3..1024

Write Depth Actual Write Depth: 1024

Read Width

Read Depth Actual Read Depth: 1024

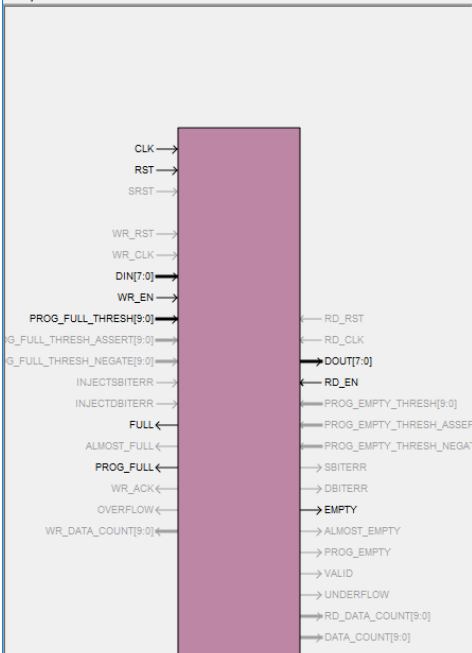
Implementation Options

☐ Enable ECC
☐ Use Embedded Registers in BRAM or FIFO (when possible)


Read Latency (From Rising Edge of Read Clock): 1

View Documents

IP Symbol



The diagram shows the FIFO Generator IP symbol with various input and output ports. Inputs on the left include CLK, RST, SRST, WR_RST, WR_CLK, DIN[7:0], WR_EN, PROG_FULL_THRESH[9:0], PROG_FULL_THRESH_ASSERT[9:0], PROG_FULL_THRESH_NEGATE[9:0], INJECTSBITERR, INJECTDBITERR, FULL, ALMOST_FULL, PROG_FULL, WR_ACK, OVERFLOW, and WR_DATA_COUNT[9:0]. Outputs on the right include RD_RST, RD_CLK, DOUT[7:0], RD_EN, PROG_EMPTY_THRESH[9:0], PROG_EMPTY_THRESH_ASSERT, PROG_EMPTY_THRESH_NEGATE, SBITERR, DBITERR, EMPTY, ALMOST_EMPTY, PROG_EMPTY, VALID, UNDERFLOW, RD_DATA_COUNT[9:0], and DATA_COUNT[9:0].

 **FIFO Generator** xilinx.com:ip:fifo_generator:9.3

Initialization

☒ Reset Pin ☒ Enable Reset Synchronization

Reset Type

☐ Synchronous Reset
☒ Asynchronous Reset

Full Flags Reset Value

☒ Use Dout Reset

Use Dout Reset Value (Hex)

Programmable Flags

Programmable Full Type

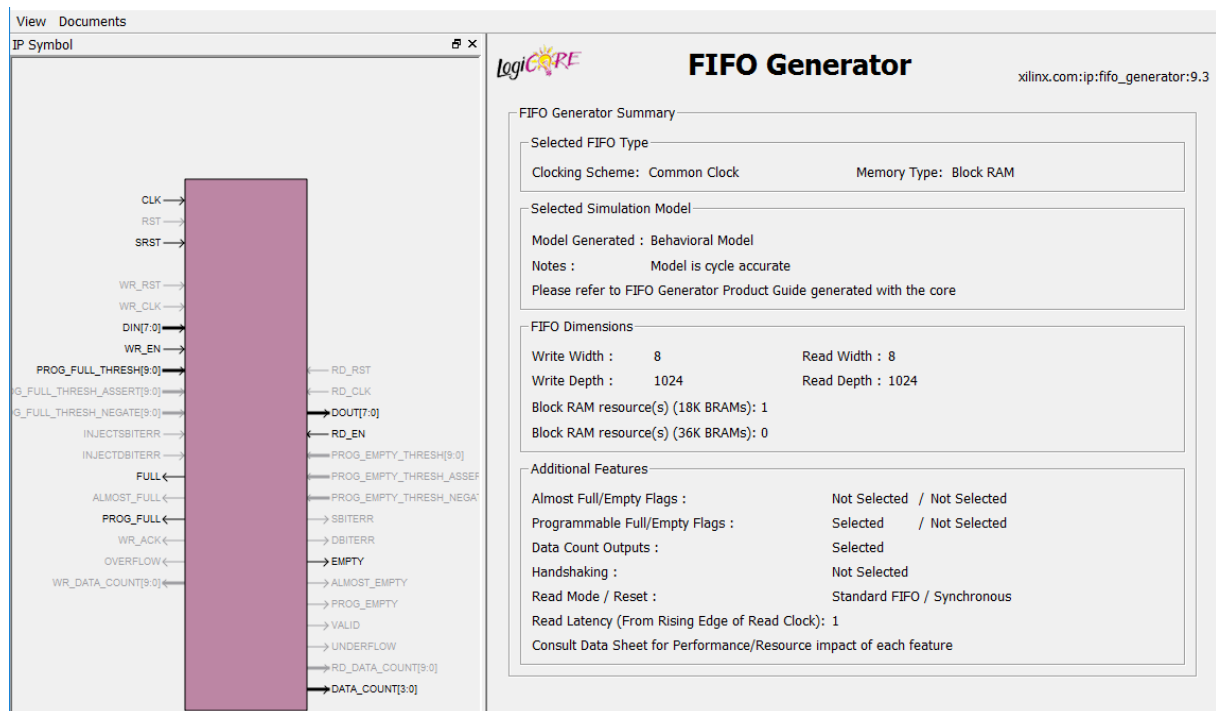
Full Threshold Assert Value Range: 4..1022

Full Threshold Negate Value Range: 2..1021

Programmable Empty Type

Empty Threshold Assert Value Range: 2..1020

Empty Threshold Negate Value Range: 3..1021



À l'aide d'un intégrateur IP, nous avons créé une mémoire FIFO comme indiqué dans l'image ci-dessus.

Bascule D 8 bits :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Bascule_D is
    Port ( D : in STD_LOGIC_VECTOR (7 downto 0);
          clk : in STD_LOGIC;
          clk_ena : in STD_LOGIC;
          rst : in STD_LOGIC;
          Q : out STD_LOGIC_VECTOR (7 downto 0));
end Bascule_D;

architecture Behavioral of Bascule_D is
begin
    process (clk)
    begin
        if rst='1' then Q<=(others=>'0');
        elsif clk_ena='1' then
            if (clk' event and clk='1') then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

Nous pouvons maintenant interconnecter les composants et écrire le code complet :

```

entity LAR is
  Port ( clk : in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        pixel_in : in  STD_LOGIC_VECTOR (7 downto 0);
        pixel_in_enable : in  STD_LOGIC;
        pixel_0 : out  STD_LOGIC_VECTOR (7 downto 0);
        pixel_1 : out  STD_LOGIC_VECTOR (7 downto 0);
        pixel_2 : out  STD_LOGIC_VECTOR (7 downto 0);
        pixel_out  : out  STD_LOGIC_VECTOR (7 downto 0);
        prog_full_threch : in  STD_LOGIC_VECTOR (7 downto 0);
        prog_full : out  STD_LOGIC);
end LAR;

```

Architecture du LAR

```

architecture LAR_arch of LAR is
  signal pix_0, pix_1, pix_2 : std_logic_vector(7 downto 0);
  signal ff1, ff2, ff3, empty, full, read_en : std_logic;
  COMPONENT FIFO_Memory
  PORT (
    clk : IN std_logic;
    rst : IN std_logic ;
    din : IN std_logic_vector(7 downto 0);
    wr_en : IN std_logic ;
    rd_en : IN std_logic ;
    prog_full_thresh : IN std_logic_vector ( 7 downto 0);
    dout : OUT std_logic_vector (7 downto 0);
    full : out std_logic ;
    empty : out std_logic ;
    prog_full : out std_logic

  );
  END COMPONENT;
|

```

Instanciation des bascules 8bits et 1bit


```

54
55 --Instanciation des deux bascules 8 bits et 1 bit
56
57
58 COMPONENT bascule_d_8bits
59 port ( D : in std_logic_vector (7 downto 0);
60       clk,reset : in std_logic ;
61       Q : out std_logic_vector ( 7 downto 0)
62     );
63 END COMPONENT ;
64
65
66 COMPONENT Bascule_d_1bits
67 port ( D : in std_logic;
68       clk,reset : in std_logic ;
69       Q : out std_logic
70     );
71 END COMPONENT ;
72 |
73

```

```

begin
bascule_D_8bit_1: bascule_d_8bits
port map (clk=> clk ,
reset => reset,
D => pixel_in ,
Q => pix_0);

```

```

bascule_D_8bit_2: bascule_d_8bits
port map (clk=> clk ,
reset => reset,
D => pix_0 ,
Q => pix_1);

```

```

bascule_D_8bit_3: bascule_d_8bits
port map (clk=> clk ,
reset => reset,
D => pix_1 ,
Q => pix_2);

```

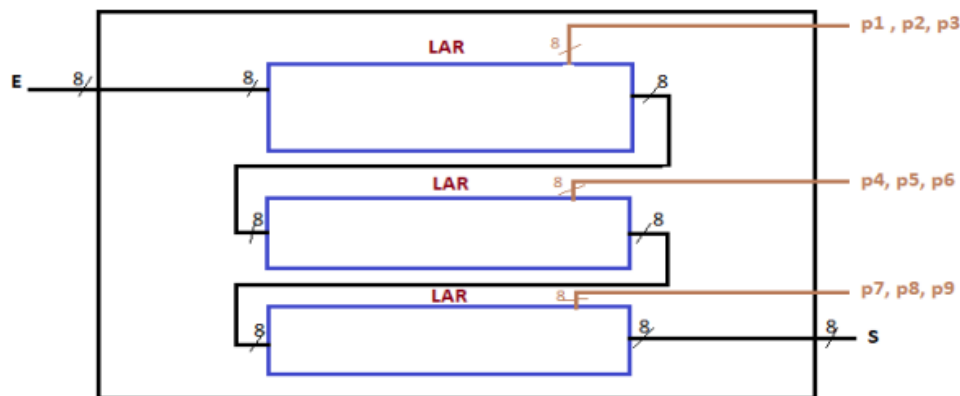
```

123
124 --FIFO
125 FIFO1: FIFO_Memory
126 port map ( clk => clk ,
127 rst => Reset ,
128 din => pix_2,
129 wr_en => ff3,
130 rd_en => read_en ,
131 dout => Pixel_out ,
132 full => full,
133 empty => empty ,
134 prog_full_thresh => prog_full_thre
135 prog_full => read_en
136 );
137
138 prog_full <= read_en;
139
140 pixel_0 <= pix_0;
141 pixel_1 <= pix_1;
142 pixel_2 <= pix_2;
143
144 |
145
146 end LAR_arch;
147
148

```

Nous allons connecter 3 lignes à retard (LAR) entre elles comme le montre l'image ci-dessous

En savoir plus sur ce texte source



Le code VHDL décrivant le système est le suivant :


```

7  -- Module Name:      Noyau_d_voisinage - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity Noyau_d_voisinage is
33     Port ( reset: in STD_LOGIC;
34           clk: in STD_LOGIC;
35           pixel_in : in  STD_LOGIC_VECTOR(7 downto 0);
36           pixel_in_enable : in  STD_LOGIC;
37           prog_full_thresh : in  STD_LOGIC_VECTOR (9 downto 0);
38           pixel_0,pixel_1,pixel_2 : out  STD_LOGIC_VECTOR (7 downto 0);
39           pixel_3,pixel_4,pixel_5 : out  STD_LOGIC_VECTOR (7 downto 0);
40           pixel_6,pixel_7,pixel_8 : out  STD_LOGIC_VECTOR (7 downto 0);
41           pixel_out : out  STD_LOGIC_VECTOR(7 downto 0);
42           prog_full : out  STD_LOGIC);
43 end Noyau_d_voisinage;
44

```

```

28 LIBRARY ieee;
29 USE ieee.std_logic_1164.ALL;
30 USE ieee.numeric_std.ALL;
31 USE ieee.std_logic_unsigned.ALL;
32
33 -- Uncomment the following library declaration if using
34 -- arithmetic functions with Signed or Unsigned values
35 --USE ieee.numeric_std.ALL;
36
37 ENTITY Noyau_voisinage_test_bench IS
38 END Noyau_voisinage_test_bench;
39
40 ARCHITECTURE behavior OF Noyau_voisinage_test_bench IS
41
42     -- Component Declaration for the Unit Under Test (UUT)
43
44     COMPONENT Noyau_d_voisinage
45     PORT(
46         reset : IN std_logic;
47         clk : IN std_logic;
48         pixel_in : IN std_logic_vector(7 downto 0);
49         pixel_in_enable : IN std_logic;
50         prog_full_thresh : IN std_logic_vector(9 downto 0);
51         pixel_0 : OUT std_logic_vector(7 downto 0);
52         pixel_1 : OUT std_logic_vector(7 downto 0);
53         pixel_2 : OUT std_logic_vector(7 downto 0);
54         pixel_3 : OUT std_logic_vector(7 downto 0);
55         pixel_4 : OUT std_logic_vector(7 downto 0);
56         pixel_5 : OUT std_logic_vector(7 downto 0);
57         pixel_6 : OUT std_logic_vector(7 downto 0);
58         pixel_7 : OUT std_logic_vector(7 downto 0);
59         pixel_8 : OUT std_logic_vector(7 downto 0);
60         pixel_out : OUT std_logic_vector(7 downto 0);
61         prog_full : OUT std_logic
62     );
63     END COMPONENT;
64

```

```

79
80 LAR_2: LAR
81 port map( clk => clk ,
82 reset => reset,
83 pixel_in => pix_lar1,
84 pixel_out => pix_lar2,
85 pixel_in_enable => pixel_in_enable_1,
86 pixel_0 => p3,
87 pixel_1 => p4,
88 pixel_2 => p5,
89 prog_full_thresh => prog_full_thresh,
90 prog_full => pixel_in_enable_2
91 );
92
93
94 LAR_3: LAR
95 port map( clk => clk ,
96 reset => reset,
97 pixel_in => pix_lar2,
98 pixel_out => pixel_out,
99 pixel_in_enable => pixel_in_enable_2,
100 pixel_0 => p6,
101 pixel_1 => p7,
102 pixel_2 => p8,
103 prog_full_thresh => prog_full_thresh,
104 prog_full => prog_full
105 );

```

```

106
107 pixel_0 <= p0;
108 pixel_1 <= p1;
109 pixel_2 <= p2;
110 pixel_3 <= p3;
111 pixel_4 <= p4;
112 pixel_5 <= p5;
113 pixel_6 <= p6;
114 pixel_7 <= p7;
115 pixel_8 <= p8;
116
117
118 end Noyau;
119
120

```

Simulation :



En conclusion, Ce projet a été très instructif et très éducatif surtout pour moi car j'ai appris quelques manipulations avancées du VHDL en traitement d'image et c'est un plus pour mes connaissances.