École Centrale de Lille

Architecture des systèmes embarqués pour la commande et la supervision

# Fibonacci

## Implémentation en VHDL


| **Nom** | **Prénom** |
|---|---|
| BASTOS SANTOS | Vitória |
| DOS SANTOS SILVA | Vitor |


Villeneuve-d'Ascq
2024

École Centrale de Lille

# Fibonacci

## Implémentation en VHDL

Compte-rendu du tutoriel de l'électif Architecture des systèmes embarqués pour la commande et la supervisionde l'École Centrale de Lille.

Villeneuve-d'Ascq
2024

# Table des matières

# Table des figures

# 1. Diviseur de fréquence

## 1.1 Description

Ce code définit un diviseur de fréquence en VHDL. Le composant freq_divider prend un signal d'horloge (clock) et un signal de remise à zéro (raz) en entrée et produit un signal de sortie (Cout). La fréquence de sortie est divisée par le facteur N défini par un paramètre générique.

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


entity freq_divider is
  generic(
  N          :      integer := 10
  );

  port(
  clock, raz   :    in std_logic;
  Cout       :    out std_logic
  );

end freq_divider;

architecture behavior of freq_divider is
signal aux : std_logic;
signal counter : integer:= 0;

begin
    process(clock, raz)
    begin
    if (raz = '1') then
      aux<='0';
    elsif rising_edge(clock) then
      counter <= counter + 1;
      aux<='0';

      if (counter = N) then
        aux<= '1';
        counter<=0;
      end if;
    end if;
    end process;
    Cout<=aux;

end behavior;
```

## 1.2 Simulation Fonctionnelle

La simulation fonctionnelle (figure 1.1) a été réalisée avec un paramètre N = 10. Le comportement observé montre que le signal de sortie Cout bascule après 10 cycles d'horloge, confirmant le bon fonctionnement du diviseur de fréquence.

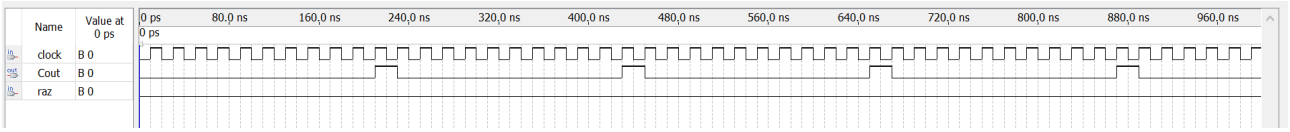


FIGURE 1.1 – Simulation fonctionnelle du diviseur de fréquence pour N = 10

# 2. Générateur de Fibonacci

## 2.1 Description

Ce code implémente un générateur de séquence de Fibonacci en VHDL. Le composant gen_fibonacci génère les nombres de Fibonacci jusqu'à une valeur maximale de 999999. Il utilise des signaux d'horloge (clock), d'activation (activation), et de remise à zéro (raz).

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.ALL;
4
5  entity gen_fibonacci is
6    port(
7      clock, activation, raz  : in std_logic;             -- Signal d'horloge,
       d'activation et de reset
8      Fout               : out natural range 0 to 999999 -- Sequence de
     Fibonacci a la sortie jusqu'a 999999
9    );
10 end gen_fibonacci;
11
12 architecture gen_fibonacci_arch of gen_fibonacci is
13   signal F0 : integer range 0 to 999999:=0; -- Fn de la sequence
14   signal F1 : integer range 0 to 999999:=1; -- Fn+1 de la sequence
15   signal F2 : integer range 0 to 999999:=0; -- Fn+2 de la sequence
16   signal aux: integer range 0 to 30    :=0; -- Signal auxiliaire pour garder
       le n de la sequence
17 begin
18   process(clock, raz)
19   begin
20     if (raz='1') then -- Si le reset est active, on remets les valeurs
     initiales
21       aux <= 0;
22       F2 <= 0;
23       F1 <= 1;
24       F0 <= 0;
25     elsif (clock'event and clock='1' and activation='1') then -- Si il y a
     un event de clock et le composant est active
26       if (aux=0) then -- Valeurs initiales de la sequence
27         F2 <= 1;
28         aux <= aux + 1;
29       elsif (aux=1) then
30         F2 <= 1;
31         F1 <= 1;
32         aux <= aux + 1;
33       elsif (aux<30) then -- Si on n'est pas encore arrive a la fin
34         F2<=F2+F1; -- on calcule Fn+2
35         F0<=F1; -- et on deplace Fn+1
36         F1<=F2;
```

```
37          aux <= aux + 1;
38        else -- Si on est arrive a la fin
39          aux <= 0; -- on se remet a 0
40          F2 <= 0;
41          F1 <= 1;
42          F0 <= 0;
43        end if;
44      end if;
45    end process;
46    Fout <= F2;
47  end architecture;
```

## 2.2 Simulation fonctionelle

La simulation fonctionnelle, ilustré par la figure 2.1, montre que le générateur produit correctement la séquence de Fibonacci, en commençant par 0, 1, 1, 2, 3, etc., à chaque cycle d'horloge lorsque le signal d'activation est actif.

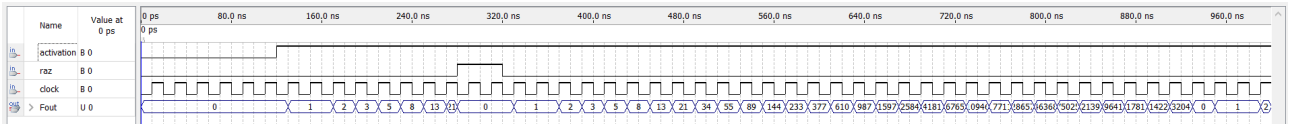


FIGURE 2.1 – Simulation fonctionelle du générateur de Fibonacci

## 2.3 Simulation temporelle

La simulation temporelle (figure 2.2) vérifie que le générateur de Fibonacci fonctionne correctement dans le temps, en confirmant que les valeurs de la séquence sont calculées et mises à jour à chaque cycle d'horloge.

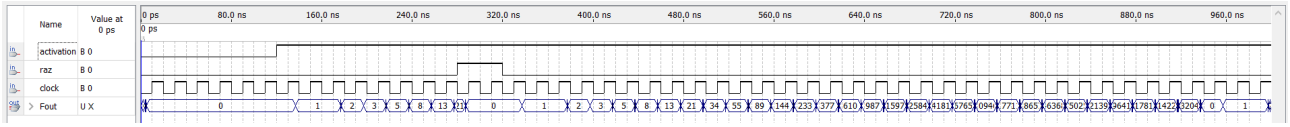


FIGURE 2.2 – Simulation temporelle du générateur de Fibonacci

# 3. Décodeur binaire

## 3.1 Description

Ce code implémente un décodeur binaire en VHDL. Le composant dec_binaire prend un entier en entrée et décompose cet entier en ses chiffres constitutifs (unités, dizaines, centaines, etc.) sous forme de vecteurs unsigned.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.math_real.all;

entity dec_binaire is
  port(
    entier             :   in natural range 0 to 999999;
    cent_milier          :   out unsigned(3 downto 0);
    diz_milier           :   out unsigned(3 downto 0);
    milier             :   out unsigned(3 downto 0);
    centaines          :   out unsigned(3 downto 0);
    dizaines             :   out unsigned(3 downto 0);
    unites             :   out unsigned(3 downto 0)
  );
end dec_binaire;

architecture dec_binaire_arch of dec_binaire is
begin
  unites <= to_unsigned(entier mod 10,unites'length); -- Pour chaque chiffre
   , on divise par 10^n et on prend le reste de la division par 10
  dizaines <= to_unsigned(entier/10 mod 10,dizaines'length); -- n=1
  centaines <= to_unsigned(entier/100 mod 10,centaines'length); -- n=2
  milier <= to_unsigned(entier/1000 mod 10,milier'length); -- n=3
  diz_milier <= to_unsigned(entier/10000 mod 10,diz_milier'length); -- n=4
  cent_milier <= to_unsigned(entier/100000 mod 10,cent_milier'length); --n=5
end architecture;
```

## 3.2 Simulation fonctionelle

La simulation fonctionnelle a vérifié que chaque chiffre de l'entier d'entrée est correctement décomposé et affiché dans les sorties respectives (cent_milier, diz_milier, milier, etc.).

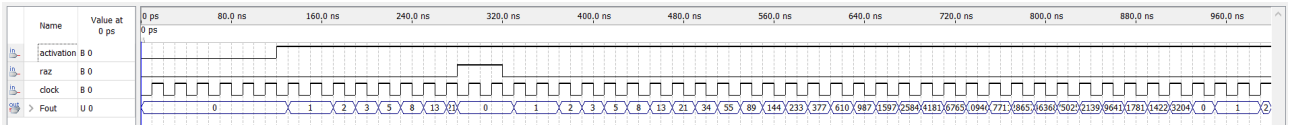


FIGURE 3.1 – Simulation fonctionelle du décodeur binaire

# 4. Décodeur 7 segments

## 4.1 Description

Ce code définit un décodeur 7 segments en VHDL. Le composant DecodeurBCD convertit un nombre en entrée (N) en un vecteur de segments (S) qui peut être utilisé pour afficher le nombre sur un afficheur 7 segments.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.ALL;
4
5  entity DecodeurBCD is
6  port(
7    N  :    in natural range 0 to 15; -- in std_logic_vector(3 downto 0)
8    S  :    out std_logic_vector(7 downto 0)
9      );
10 end DecodeurBCD;
11
12 architecture decodeur7seg of DecodeurBCD is
13 begin
14   with N select
15     S<= "00111111"  when 0,    -- 0000
16         "00000110"  when 1,    -- 0001
17         "01011011"  when 2, -- 0010
18         "01001111"  when 3,    -- 0011
19         "01100110"  when 4,    -- 0100
20         "01101101"  when 5,    -- 0101
21         "01111101"  when 6,    -- 0110
22         "00000111"  when 7,    -- 0111
23         "01111111"  when 8,    -- 1000
24         "01101111"  when 9,    -- 1001
25         "01110111"  when 10,   -- 1010
26         "01111100"  when 11,   -- 1011
27         "00111001"  when 12,   -- 1100
28         "01011110"  when 13,   -- 1101
29         "01111001"  when 14,   -- 1110
30         "01110001"  when 15;   -- 1111
31 end decodeur7seg;
32
33
```

## 4.2 Simulation fonctionelle

La simulation fonctionnelle (figure 4.1) a montré que le décodeur convertit correctement les nombres de 0 à 15 en leur représentation binaire correspondante pour un affichage sur un afficheur 7 segments.

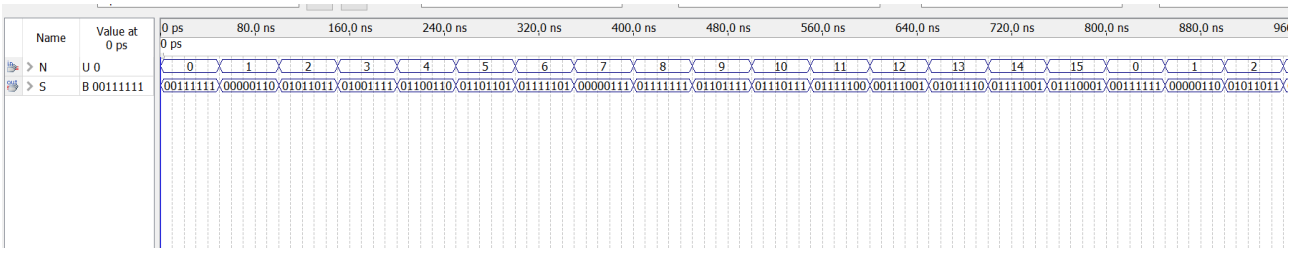


FIGURE 4.1 – Simulation fonctionnelle du décodeur

# 5. Système complet

Le système complet, representé par la figure 5.1, combine les modules précédents pour créer un système intégré. Chaque module (diviseur de fréquence, générateur de Fibonacci, décodeur binaire, décodeur 7 segments) fonctionne ensemble pour accomplir les tâches définies.

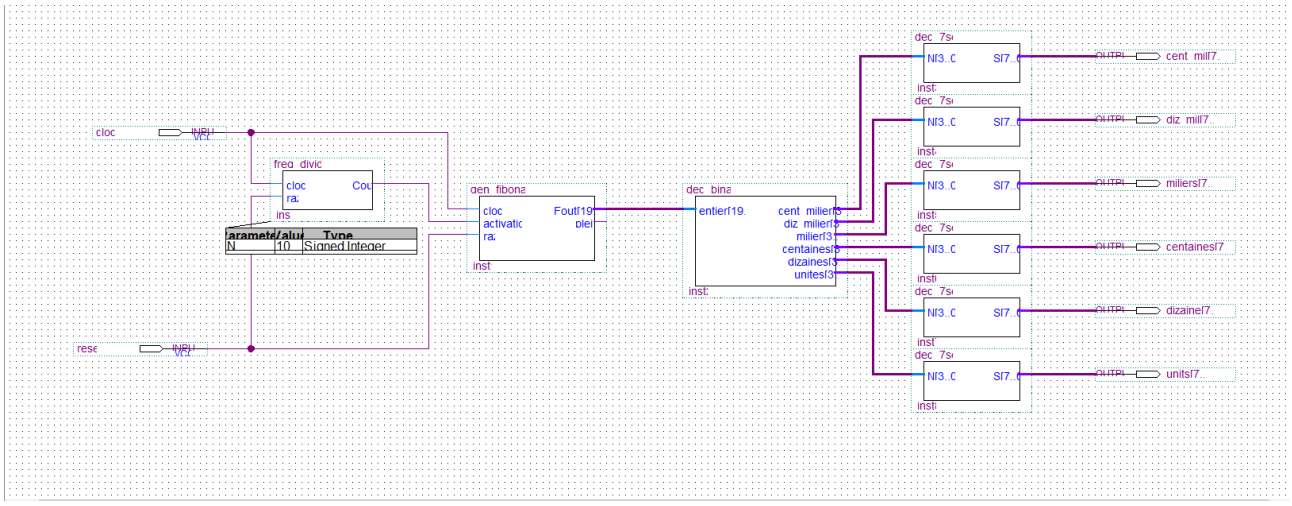


FIGURE 5.1 – Système Complète

## 5.1 Simulation Fonctionnelle

Les simulations montrent que chaque composant fonctionne correctement à la fois individuellement et en tant que partie intégrée du système complet comme on peut observer par la figure 5.2.

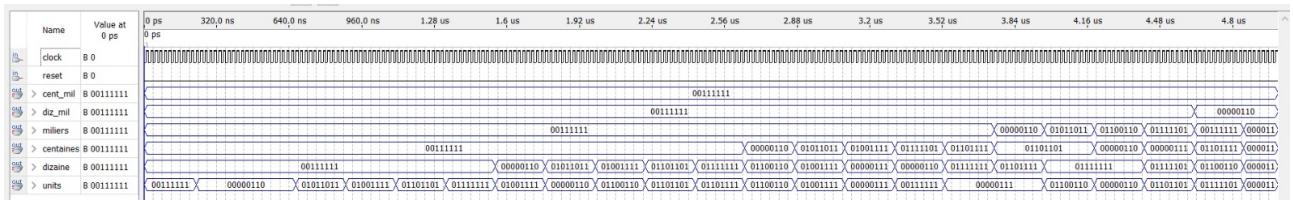


FIGURE 5.2 – Simulation Fonctionnelle du système complète