

Projet de VHDL

Réalisation d'un Fréquencemètre

JÉRÉMIE FOURMANN (Promo 2013 - Électronique)

MAXIME MORIN (Promo 2013 - Électronique)

29 mai 2012



Plan

1	Objectifs	3
1.1	Rappel du cahier des charges	3
1.2	Nos choix	3
1.2.1	Choix de la validité des méthodes de mesures	3
1.2.2	Choix de la conception en machine d'état	4
2	Conception du système	5
2.1	Présentation du système	5
2.2	Module d'affichage	6
2.2.1	Schéma bloc	6
2.2.2	Machines d'états	7
2.2.3	Résultats de simulation	7
2.3	Mesure par étalon	7
2.3.1	Schéma bloc	7
2.3.2	Machines d'états	8
2.3.3	Résultat de simulation	9
2.4	Mesure par échantillonnage	9
2.4.1	Schéma bloc	9
2.4.2	Machines d'états	9
2.4.3	Résultat de simulation	9
2.5	Performances du système	9
3	Bilan	10
A	Manuel d'utilisateur	11
A.1	Introduction	11
A.2	Component VHDL	11
A.3	Implémentation sur Nexys 2	11
A.4	Fichier contrainte	11

1 Objectifs

Le but de ce projet est de mesurer la fréquence d'un signal, le résultat de cette mesure sera affichée sur quatre afficheurs 7 segments.

1.1 Rappel du cahier des charges

Notre projet doit répondre aux contraintes suivantes :

Changement de gamme automatique : Pour garder le plus de précision possible, il faudra changer de méthode de mesure quand la fréquence dépassera une certaine valeur (voir partie nos choix).

Rafraîchissement de la mesure de 1s : Permet de garder une bonne fluidité lorsque la fréquence est amenée à changer.

Affichage sur 4 digits : Permet d'avoir une précision correcte

Plage de fréquence de 1Hz à 10 MHz

Affichage des calibres

RQ : Par ailleurs notre système affichera un message d'alerte quand le signal mesuré dépasse la plage de fréquence admise.

1.2 Nos choix

1.2.1 Choix de la validité des méthodes de mesures

Nous avons deux méthodes de mesures à notre disposition : méthode étalon et méthode échantillonnage.

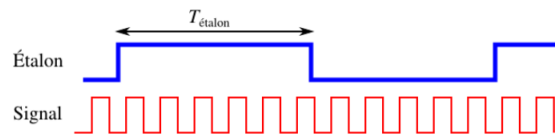


FIGURE 1 – Méthode étalon

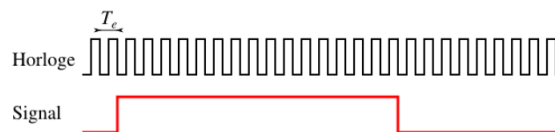


FIGURE 2 – Méthode échantillonnage

Ces deux méthodes ne sont pas meilleur l'une que l'autre, elles sont complémentaires. En effet pour un signal de fréquence élevé nous utiliserons la méthode étalon, elle permet d'avoir le maximum de précision.

En revanche à basse fréquence cette méthode ne peut s'appliquer convenablement, nous utilisons alors la méthode échantillonnage.

Il faut donc trouver la valeur pour laquelle on change de méthode de mesure. Nous allons tracer la précision en fonction de la fréquence pour les deux méthodes

On a :

$$Precision_{BF} = \frac{F_{clk}}{F_{signal}}$$

$$Precision_{HF} = \frac{F_{signal}}{F_{talon}}$$

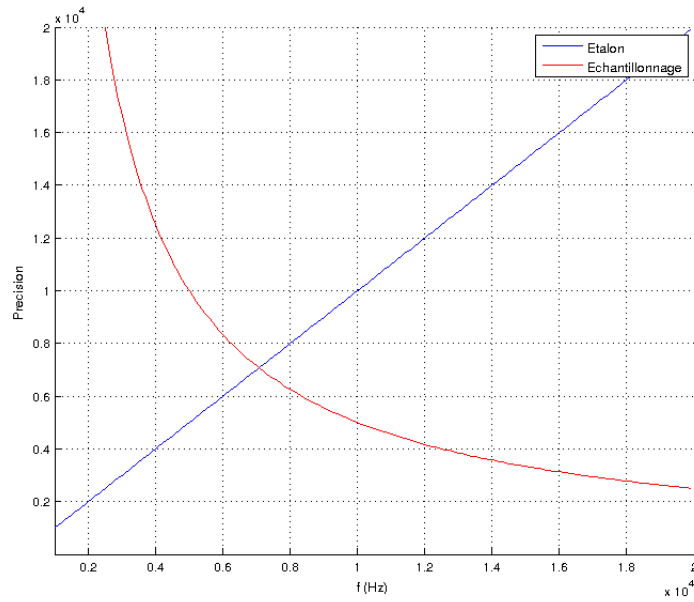


FIGURE 3 – Précision des deux méthodes

On s'aperçoit que pour avoir le maximum de précision il faut changer de méthode aux alentours de 7 KHz.

1.2.2 Choix de la conception en machine d'état

Notre conception étant entièrement synchrone, chaque sous module de notre système sera une machine d'état avec le bloc M et G synchronisé sur la clock du FPGA.

Cette conception en machine d'état nous assure une implémentation adapté aux fonctionnement du FPGA. Par ailleurs nous avons aussi une meilleur lisibilité de notre programme et une meilleur facon de débbugger car nous avons accès à l'état de notre machine lors des simulations.

2 Conception du système

2.1 Présentation du système

Le système final que nous implémentons sur le FPGA ressemble à ceci :

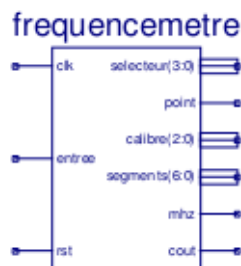


FIGURE 4 – Diagramme A-0

L'utilisateur n'aura qu'à envoyer le signal dont il veut mesurer la fréquence. Pour une approche plus technique, décomposons ce bloc en sous modules :

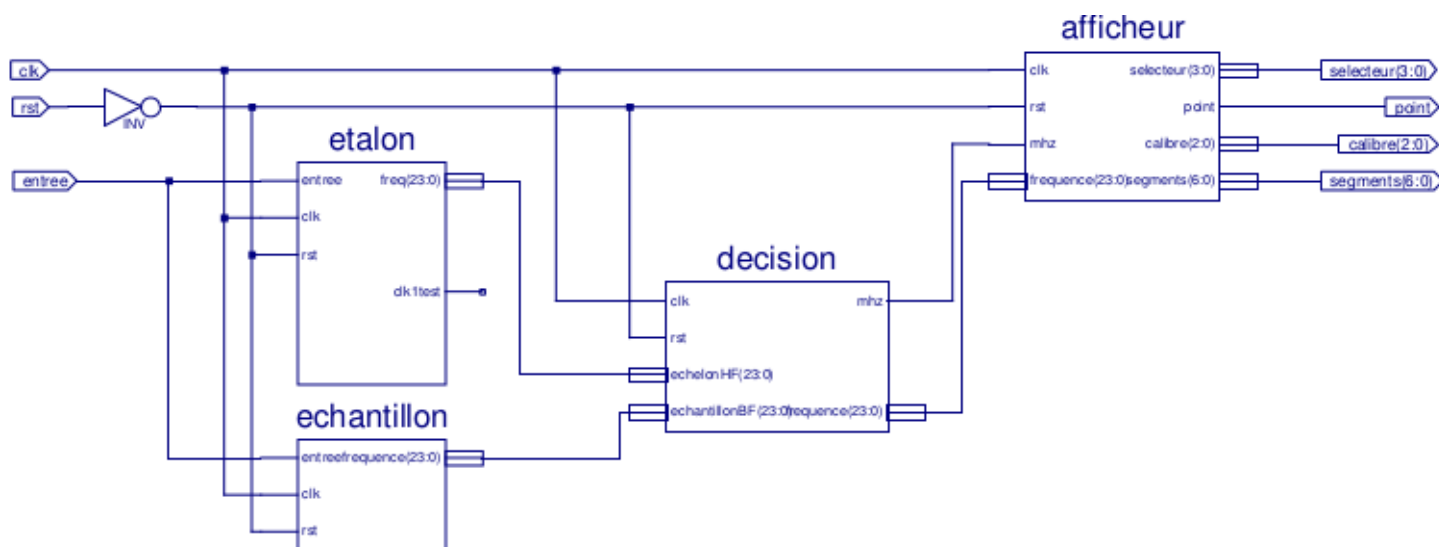


FIGURE 5 – Diagramme A0

Notre fréquencesmètre se décompose en 4 blocs principaux que nous développerons par la suite :

La mesure par étalon : Nous mesurons le nombre de fronts du signal pendant une période étalon de 1 seconde. Ce nombre de fronts est proportionnel à la fréquence.

La mesure par échantillonnage Nous comptons le nombre de fronts d'horloge entre deux fronts du signal. Ce nombre est inversement proportionnel à la fréquence, il faudra l'inverser...

La décision Nous choisissons quelle mesure est la plus précise pour être affichée, selon la fréquence de travail...

L'affichage Nous affichons la valeur envoyée par le module de décision.

Par la suite, nous allons détailler ces modules dans le même ordre que nous les avons conçus.

2.2 Module d'affichage

2.2.1 Schéma bloc

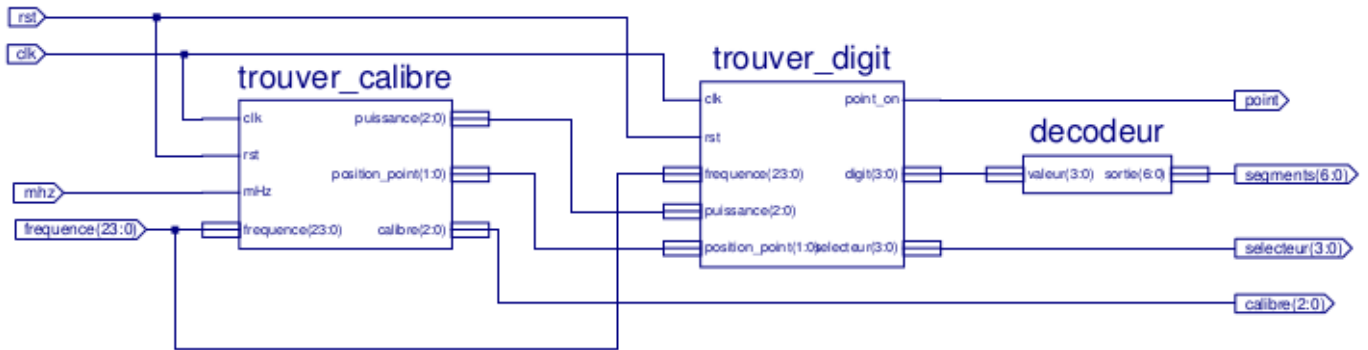


FIGURE 6 – Diagramme A1 : module d'affichage

L'objectif de ce module est de transformer le résultat de la mesure (une succession de zéros et de uns sans unités) en une grandeur physique en décimale dans les grandeurs du système international... Nous avons choisi de faire un bloc très générique : il recevra un nombre binaire codé sur 24 bits et une information sur son unité (Hertz ou milli-Hertz), il devra ensuite le formater et l'afficher sur les 4 afficheurs de la carte, en plaçant le point et déterminant le calibre.

trouver_calibre : Ce bloc détermine le calibre que nous allons afficher à l'utilisateur, c'est à dire la position du point, la led à allumer, et la puissance de 10 du nombre à afficher (utile pour la conversion dans le bloc suivant)

trouver_digit : Ce module converti en binaire codé décimal et affiche un à un les quatres digits significatifs de la fréquence. Il utilise des données déteminées par le bloc précédent. Il gère aussi le balayage en insérant un reatrd de 2ms entre deux digits et en commandant le sélectionneur.

decodeur : Ce bloc se contente de convertir le BCD pour commander l'afficheur.

2.2.2 Machines d'états

2.2.3 Résultats de simulation

2.3 Mesure par étalon

2.3.1 Schéma bloc

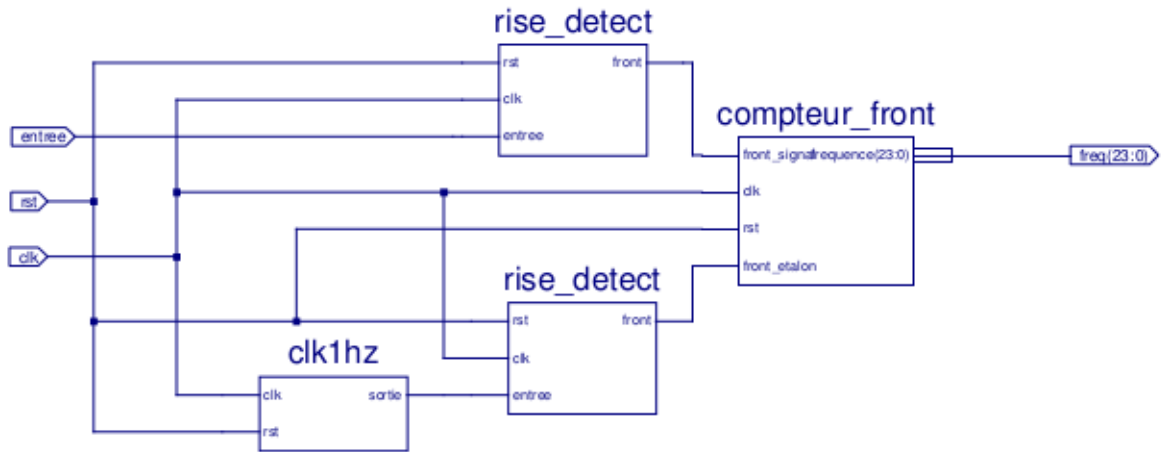


FIGURE 7 – Diagramme A2 : mesure par étalon

L'objectif de ce module est de compter le nombre de front du signal pendant un temps fixe appelé "étalon". La période de cet étalon a été fixé précédemment (TODO) à une seconde. Etant donné que cette méthode ne sera utilisée que pour les fréquences supérieures au KHz (le seuil étant à 7kHz), nous ferons en sorte que la grandeur de sortie soit en Hz, ce qui nous permet de rentrer dans le cahier des charges en termes de précision et en terme de plage de fréquence en n'utilisant que 24bits. Ce module fera appel aux sous modules suivants :

rise_detect : Ce bloc permet de générer une impulsion durant exactement 1 cycle d'horloge lorsqu'un front montant est détecté sur son entrée.

clk1hz : Module de division de fréquence permettant de générer une fréquence de 1 Hertz pour cadencer les échelons.

compteur_fronts : Ce bloc compte le nombre de fronts du signal entre deux fronts du signal d'étalon, après chaque mesure, il met la valeur sur le bus de sortie et relance une nouvelle mesure.

2.3.2 Machines d'états

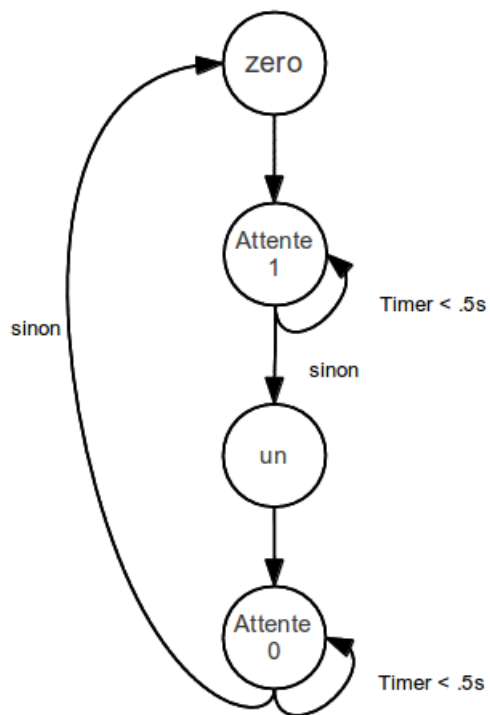


FIGURE 8 – Machine d'état de Clock 1kHz

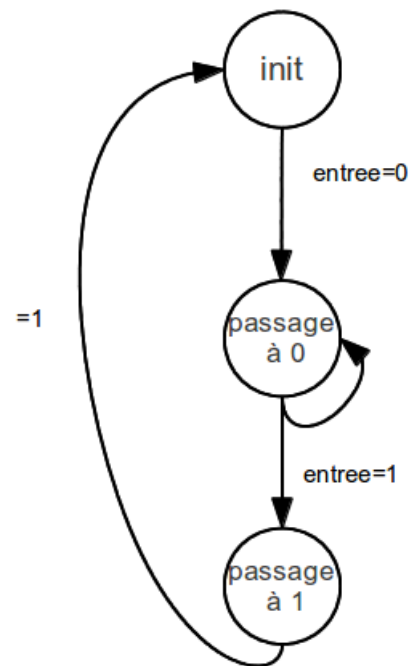


FIGURE 9 – Machine d'état de Rise_detect

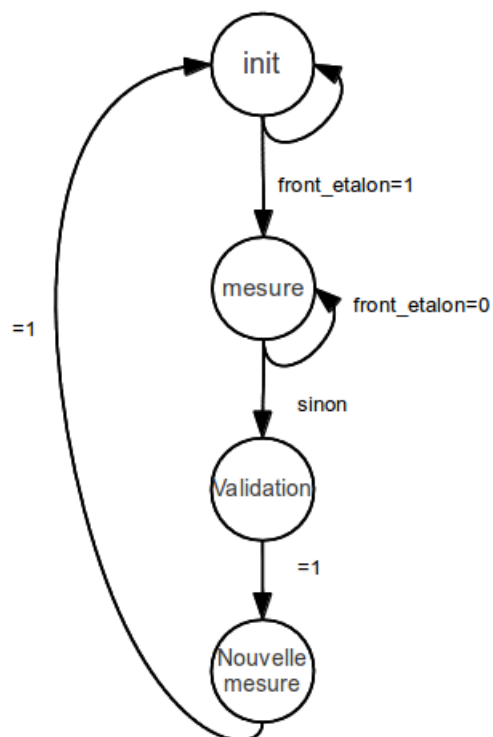


FIGURE 10 – Machine d'état de Compte_etalon

2.3.3 Résultat de simulation

2.4 Mesure par échantillonnage

2.4.1 Schéma bloc

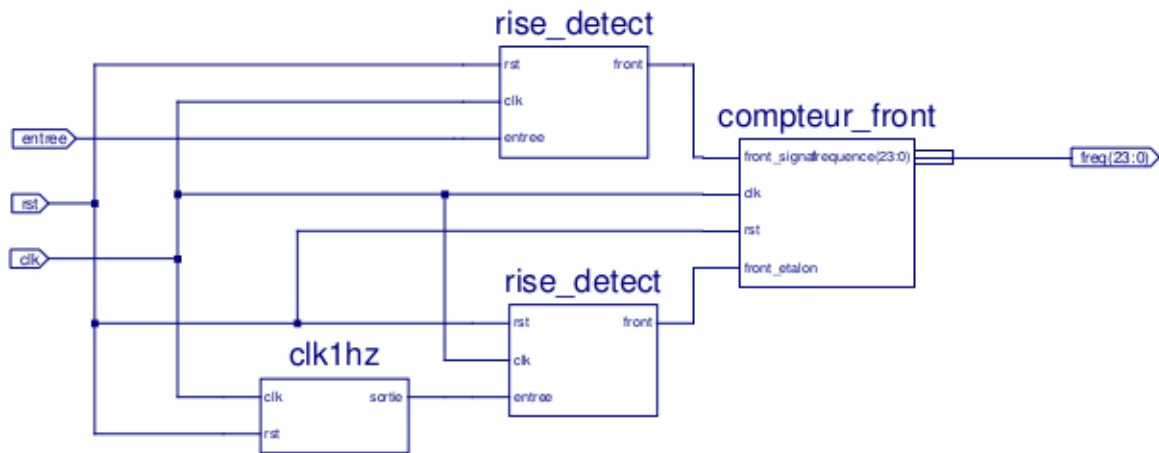


FIGURE 11 – Diagramme A3 : mesure par étalon

L'objectif de ce module est de compter le nombre de front du signal pendant un temps fixe appelé "étalon". La période de cet étalon a été fixé précédemment (TODO) à une seconde. Etant donné que cette méthode ne sera utilisée que pour les fréquences supérieures au KHz (le seuil étant à 7kHz), nous ferons en sorte que la grandeur de sortie soit en Hz, ce qui nous permet de rentrer dans le cahier des charges en termes de précision et en terme de plage de fréquence en n'utilisant que 24bits. Ce module fera appel aux sous modules suivants :

rise_detect : Ce bloc permet de générer une impulsion durant exactement 1 cycle d'horloge lorsqu'un front montant est détecté sur son entrée.

clk1hz : Module de division de fréquence permettant de générer une fréquence de 1 Hertz pour cadencer les échelons.

compteur_fronts : Ce bloc compte le nombre de fronts du signal entre deux fronts du signal d'étalon, après chaque mesure, il met la valeur sur le bus de sortie et relance une nouvelle mesure.

2.4.2 Machines d'états

2.4.3 Résultat de simulation

2.5 Performances du système

3 Bilan

A Manuel d'utilisateur

A.1 Introduction

L'utilisateur pourra utiliser le projet soit en utilisant le composant Fréquencemètre pour l'inclure dans un autre projet soit l'utiliser directement sur la Nexys 2.

A.2 Composant VHDL

Notre projet peut se résumer en un seul composant suivant :

A.3 Implémentation sur Nexys 2

En chargeant le .bit, l'utilisateur pourra utiliser le fréquencemètre.

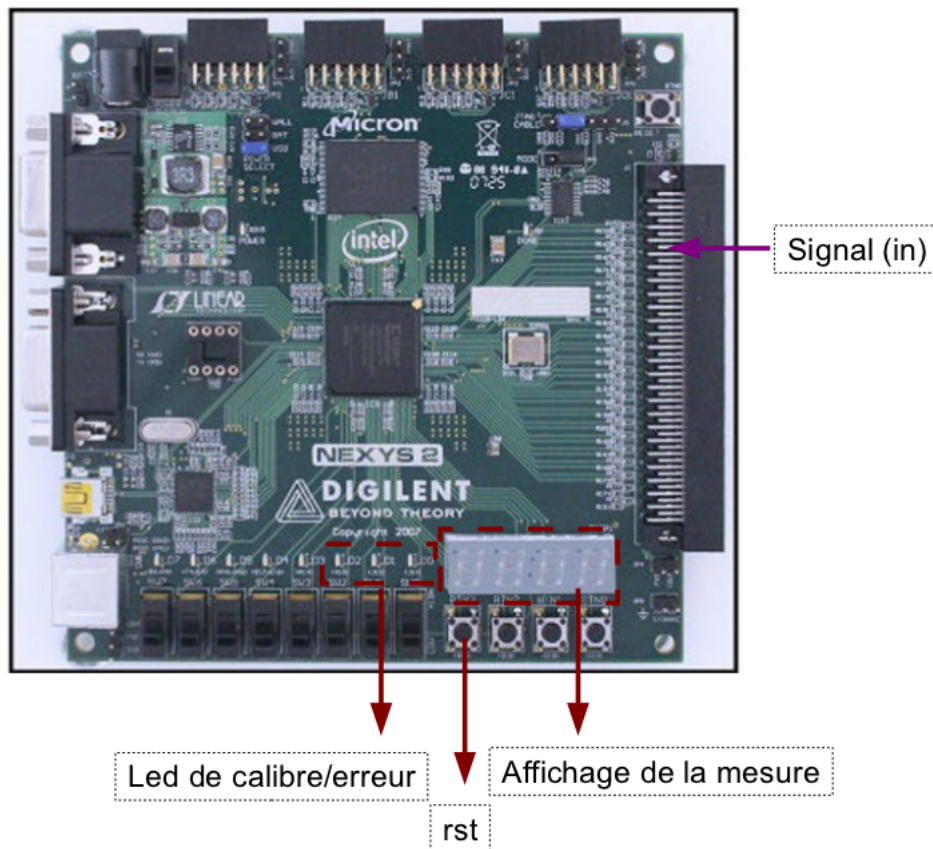


FIGURE 12 – Implémentation sur carte Nexys 2

A.4 Fichier contrainte

L'utilisateur pourra éditer le fichier de contraintes selon ses besoins et ressources disponible.

```
1 net "rst" loc = "R17";  
  net "clk" loc = "B8";  
3 net "mhz" loc = "R4";  
  
5  
  
7 net "segments[0]" loc = "L18";  
  net "segments[1]" loc = "F18";  
9 net "segments[2]" loc = "D17";  
  net "segments[3]" loc = "D16";  
11 net "segments[4]" loc = "G14";  
   net "segments[5]" loc = "J17";
```

```
13 net "segments[6]" loc = "H14";  
    net "point" loc = "C17";  
15  
    net "selecteur[0]" loc = "F15";  
17 net "selecteur[1]" loc = "C18";  
    net "selecteur[2]" loc = "H17";  
19 net "selecteur[3]" loc = "F17";  
  
21 net "calibre[0]" loc = "J14";  
    net "calibre[1]" loc = "J15";  
23 net "calibre[2]" loc = "K15";  
  
25 net "entree" loc = "B4";
```

B Extrait de code VHDL

```
-----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:      18:18:06 05/06/2012
6  -- Design Name:
7  -- Module Name:      nombre_24bits_generateur - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Poue le test de notre module d'affichage
14 --
15 -- affiche une puissance de deux par seconde
16 --
-----
18 library IEEE;
19 use IEEE.STD_LOGIC_1164.ALL;
20
21 -- Uncomment the following library declaration if using
22 -- arithmetic functions with Signed or Unsigned values
23 --use IEEE.NUMERIC_STD.ALL;
24
25 -- Uncomment the following library declaration if instantiating
26 -- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity nombre_24bits_generateur is
31     Port ( rst : in  STD_LOGIC;
32           clk : in  STD_LOGIC;
33           nombre : out STD_LOGIC_VECTOR (23 downto 0));
34 end nombre_24bits_generateur;
35
36 architecture Behavioral of nombre_24bits_generateur is
37     type etat is (init, calc_nombre, attente);
38
39     signal etatf : etat; --etat futur
40     signal etatp : etat; --etat present
41
42     signal timer : STD_LOGIC_VECTOR (23 downto 0);
43
44 begin
45
46     --Bloc F
47     process(etatp, rst)
48     begin
49         if rst='0' then etatf <=init ;
50         else
51             case etatp is
52             when init =>
53
54                 when others => etatf <= init;
55
56             end case;
57         end if;
58     end process;
59
60     --Bloc M
61     process(clk, rst)
62     begin
63         if (clk'event and clk = '1') then etatp <= etatf ;
64         end if;
```

```

        end process;
66
    --Bloc G
68    process(clk)
    begin
70        if(clk'event and clk='1') then

72            case etatp is
            when init =>

74                when others =>

76                end case;

78            end if;
80        end process;

82    --autres process
    process(step)
84    begin

86    end process;

88    end Behavioral;

```