Projet de VHDL

Réalisation d'un Fréquencemètre

Jérémie Fourmann (Promo 2013 - Eléctronique)
Maxime Morin (Promo 2013 - Eléctronique)

4 juin 2012

Plan

1 Objectifs

Le but de ce projet est de réaliser un dispositif capable de mesurer avec précision la fréquence d'un signal. Il sera réalisé sur un FPGA de type Spartan-3e implanté sur une carte de développement Nexys2. Nous disposons donc d'une interface comportant des entrées sorties utilisateur comme des boutons poussoirs et des afficheurs 7 segments.

1.1 Rappel du cahier des charges

Notre projet doit répondre aux contraintes suivantes :

Plage de fréquence : de 1 Hz à 10 MHz

Rafraichissement de la mesure : Toutes les secondes, pour un fonctionnement fluide pour l'utilisateur.

Affichage : Sur 4 digits avec affichage du calibre de mesure parmis trois possibles : Hz, kHz, MHz.

Bien qu'en apparence très simple, ce cahier des charges implique d'autres fonctions :

Précision : Il faudra que l'erreur soit de l'ordre de 10^{-3} pour n'affichier que des chiffres significatifs à l'utilisateur.

Méthodes de mesure : Pour garantir une erreur minimale sur toute la plage de fréquence, il faudra avoir recours à deux méthodes différentes. Il faudra aussi que le dispositif soit capable de choisir la méthode la plus adaptée.

Détection des erreurs : Le fréquencemètre devra aussi détecter lorsqu'il sort de sa plage de fonctionnement normal pour le pas afficher des données érronées à l'utilisateur.

1.2 Nos choix

1.2.1 Méthodes de mesures

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

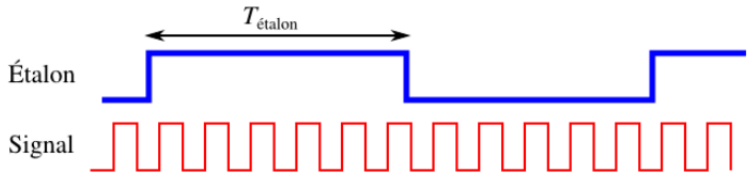

FIGURE 1 – Mesure de fréquence par étalonnage

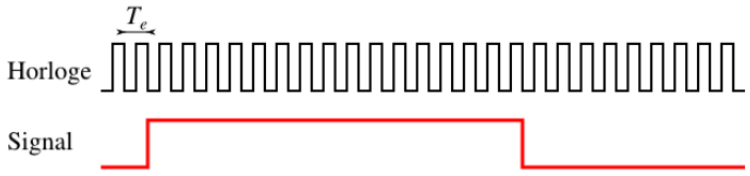

FIGURE 2 – Mesure de fréquence par échantillonnage

Ces deux méthodes ne sont pas meilleures l'une par rapport à l'autre : elles sont complémentaires. En effet pour un signal de fréquence élevée nous utiliserons la méthode étalon, elle permet d'avoir le maximum de précision. En revanche à basse fréquence cette méthode ne peut s'apliquer convenablement, nous utiliserons 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. Nous pouvons évaluer la résolution de chaque de ces deux méthodes par les équations suivantes :

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

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

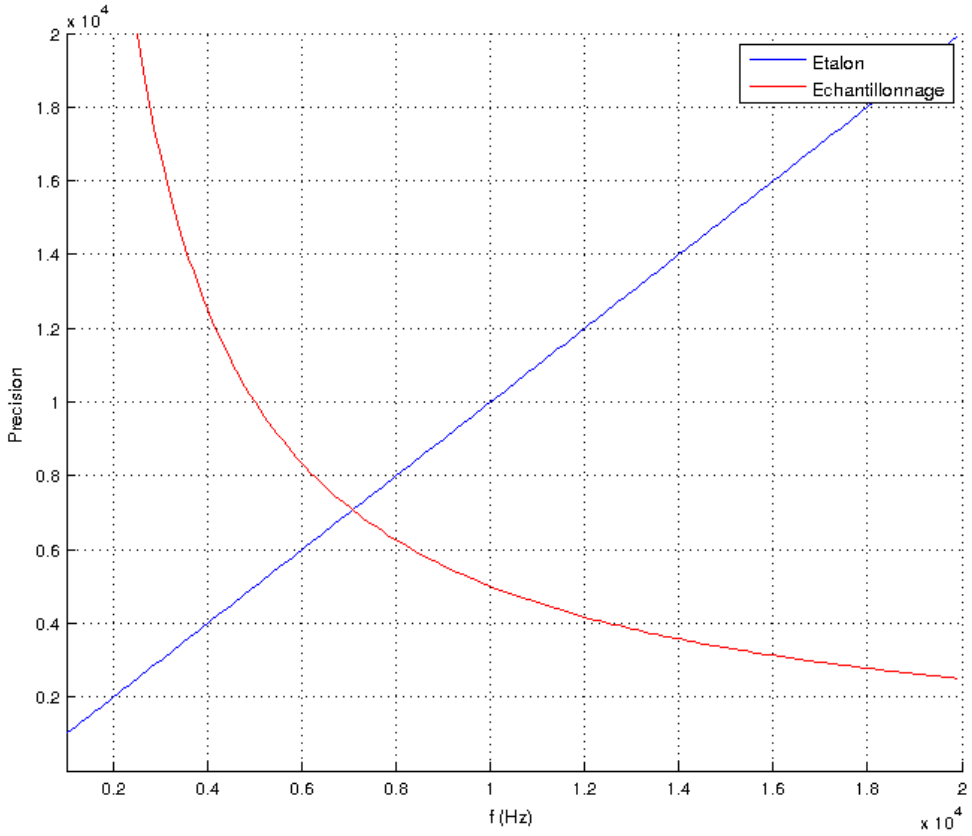

FIGURE 3 – Précision des deux méthodes

On s'apperçoit que pour avoir le maximun 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 séquentiel de notre sytème 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 des résultats après implémentation au plus proche des simulations. Par ailleurs nous avons aussi une meilleur lisibilité de notre programme et la méthodologie de débuggage est simplifiée 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 implémenteons sur le FPGA ressemble à ceci :

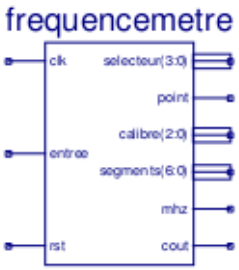

FIGURE 4 – Diagramme A-0

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 :

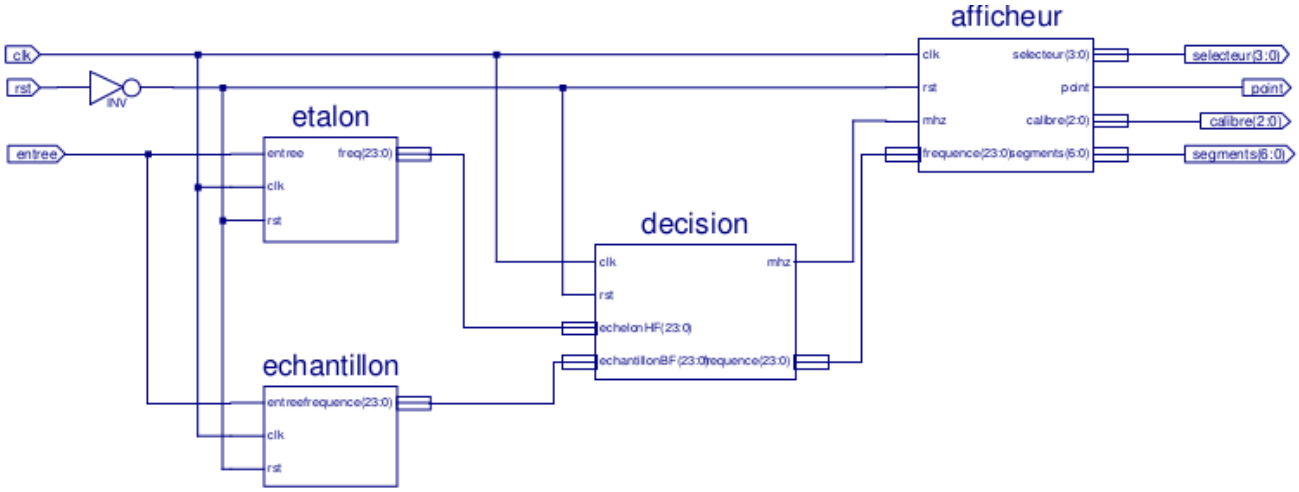

FIGURE 5 – Diagramme A0

Notre fréquencemètre 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 dexus 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 afficher, 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

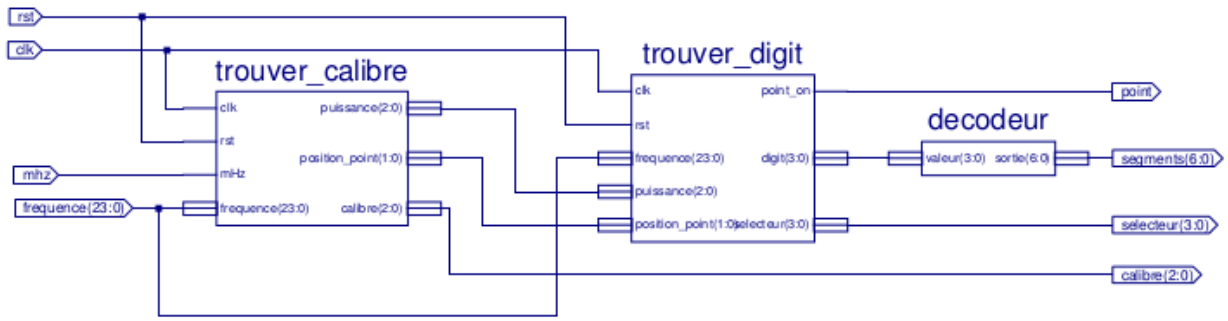

FIGURE 6 – Diagramme A1 : module d'affichage

L'objectif de ce module est de transformer le résultat de la mesure (une succession de zéros et de uns sans unité) en une grandeur physique en décimale dans les grandeurs du système internationnal... 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). Ce bloc est combinatoire.

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 sélecteur.

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

2.2.2 Machine d'état

Ci dessous, le graphe d'état de la seule machine d'état du module d'affichage.

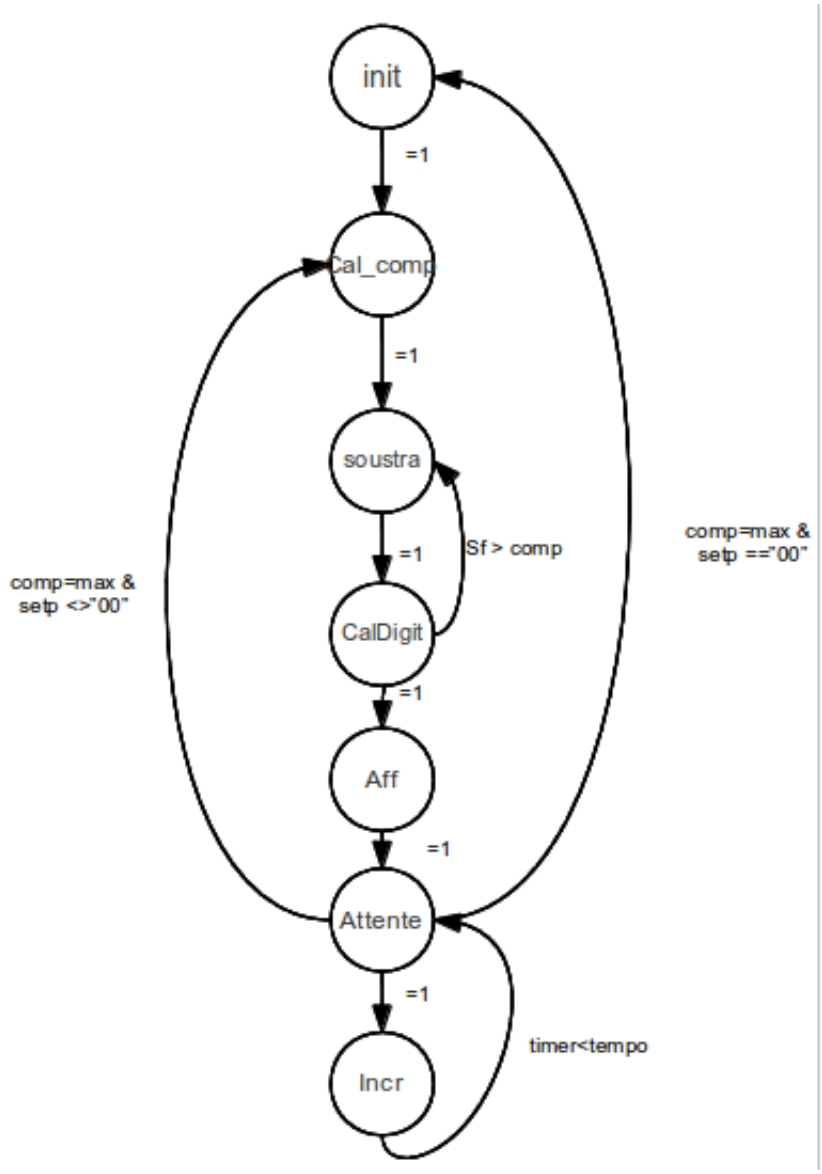

FIGURE 7 – Machine d'état de calcul_digit

Le signal *step* permet de sélectionner le digit de l'afficheur qui restera éclairé pendant 2ms.

2.2.3 Résulats de simulation

2.3 Mesure par étalon

2.3.1 Schéma bloc

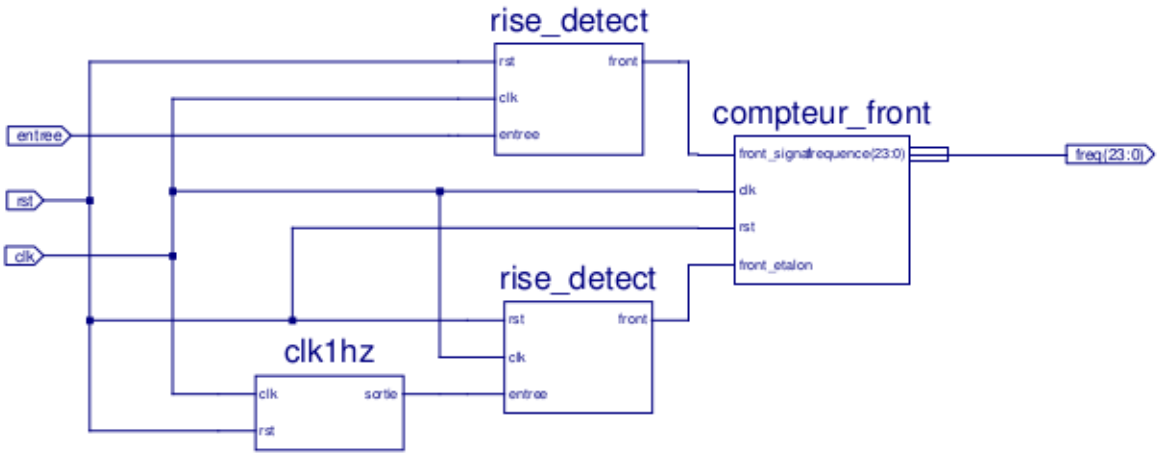

FIGURE 8 – Diagramme A2 : mesure par étalon

L'objectif de ce module est de compter le nombre de front du signal pendant un temps fixe apellé "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 suppérieures 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 chaier des chagres en tremes de précison et en terme de plage de fréquence en n'utilisant que 24bits. Ce module ferta 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

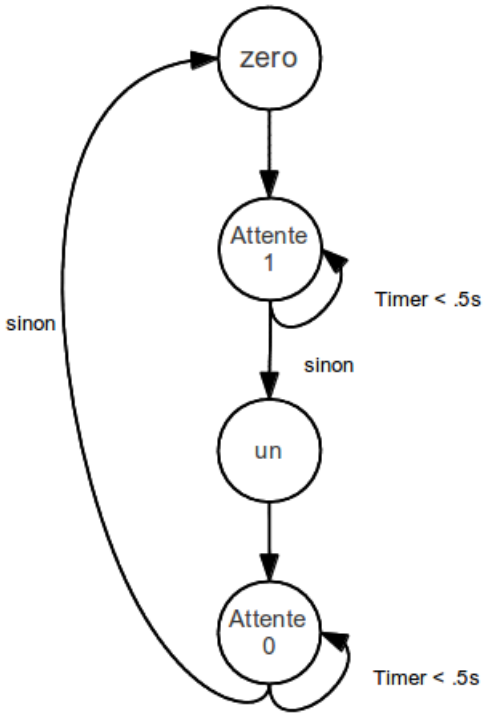

FIGURE 9 – Machine d'état de Clock 1kHz

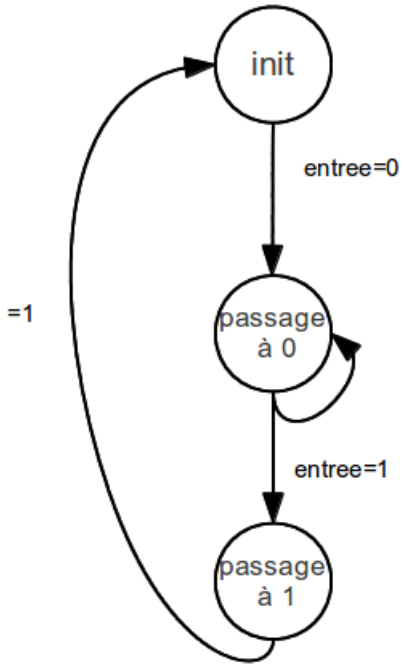

FIGURE 10 – Machine d'état de Rise_detect

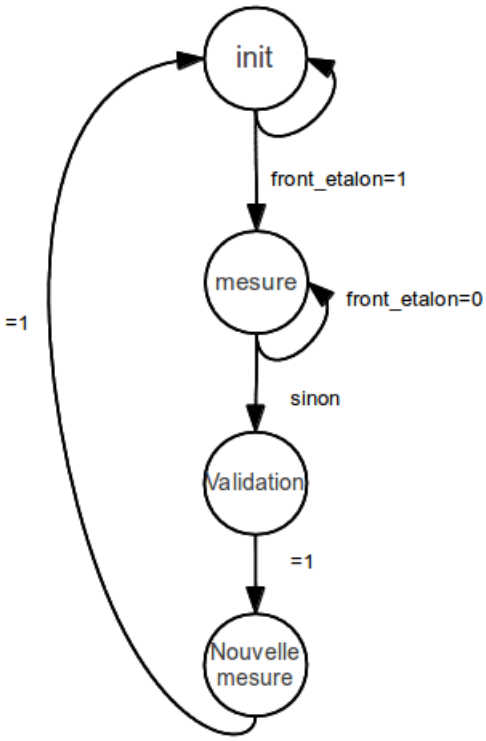

FIGURE 11 – Machine d'état de Compte_etalon

2.3.3 Résulat de simulation

2.4 Mesure par échantillonnage

2.4.1 Schéma bloc

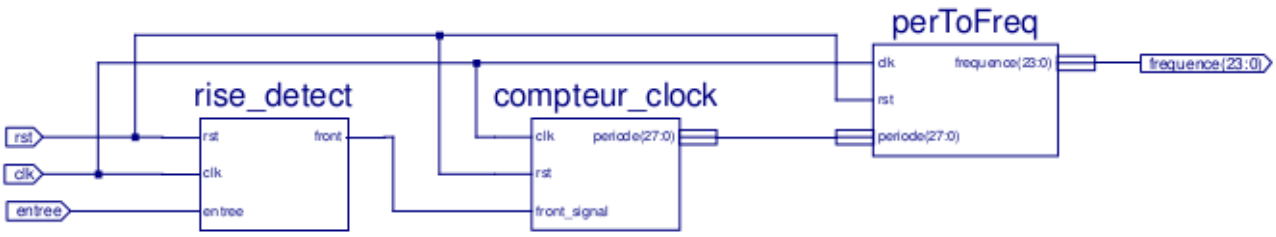

FIGURE 12 – Diagramme A3 : mesure par echantillonnage

L'objectif de ce module est de compter le nombre de front du signal pendant un temps fixe apellé "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 suppérieures 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 chaier des chagres en tremes de précison et en terme de plage de fréquence en n'utilisant que 24bits. Ce module ferta 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

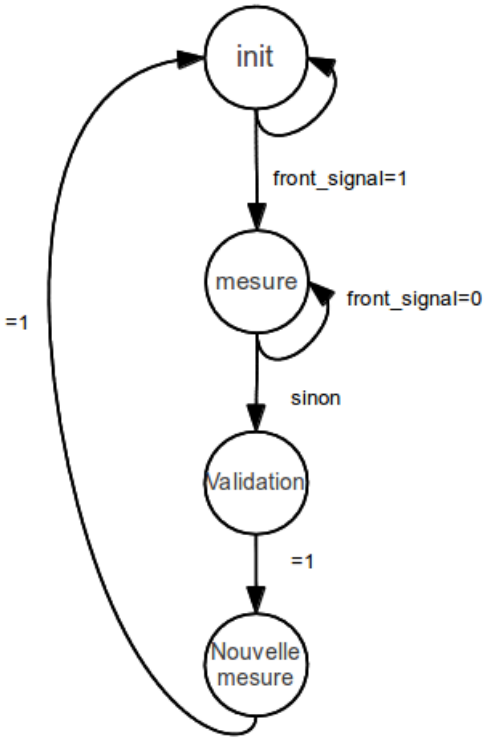

FIGURE 13 – Machine d'état de compteur_clock

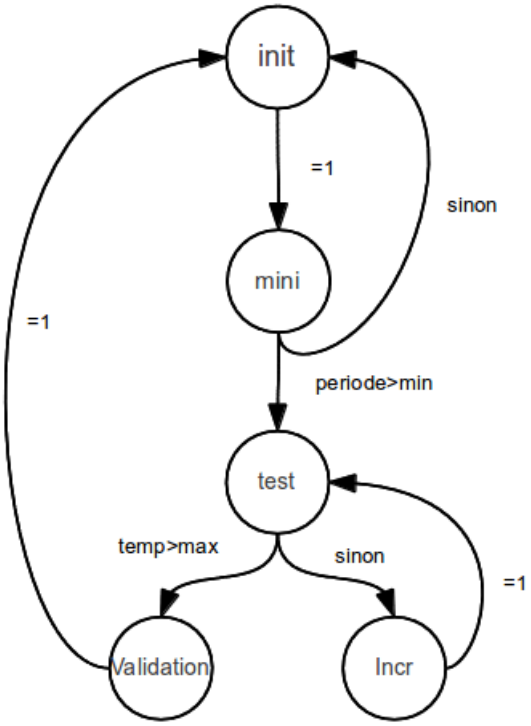

FIGURE 14 – Machine d'état de PerToFreq

2.4.3 Résulat de simulation

2.5 Décision

Nous rapellons la composition final du fréquencemètre :

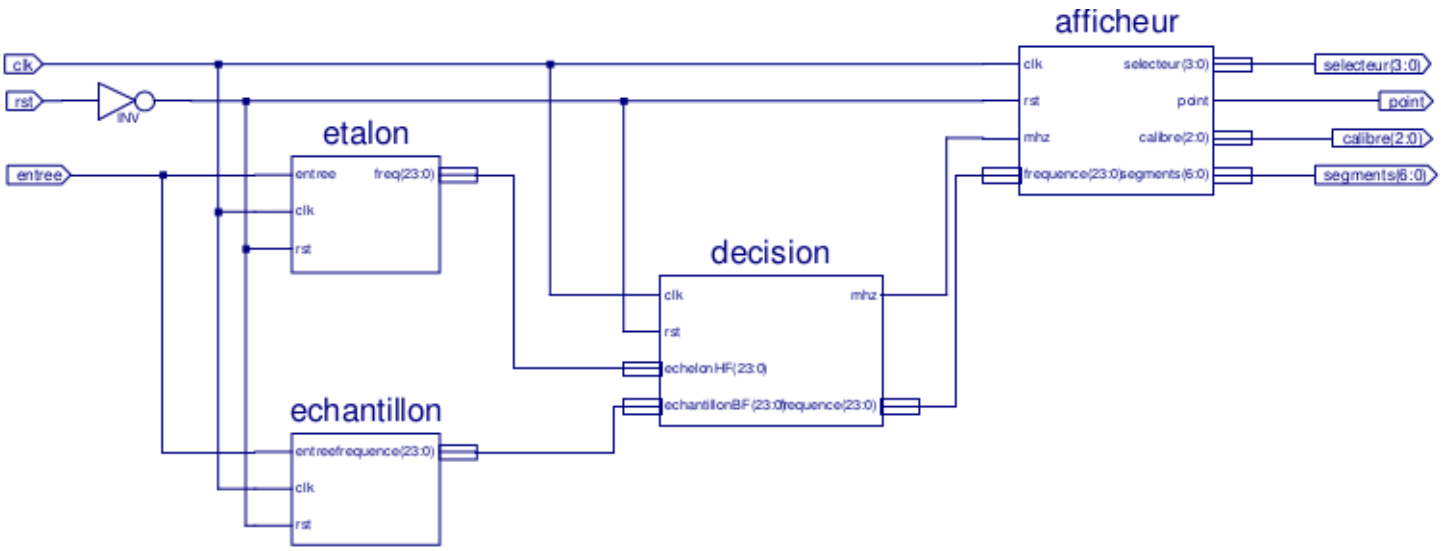

FIGURE 15 – Diagramme A0

Le seul bloc manquant est celui de décision permettant de choisir quelle méthode va être sélectionner pour l'affichage. Voici la machine d'état de ce bloc :

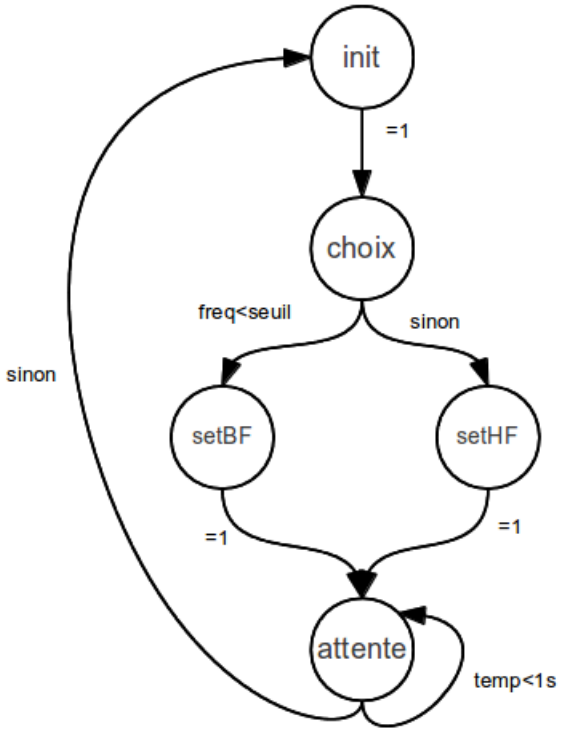

FIGURE 16 – Machine d'état du bloc de décison

2.6 Performances du système

TODO : courbe erreur + pourcentage + rapidite

3 Bilan

A Manuel d'utilisateur

A.1 Introduction

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

A.2 Component VHDL

Notre projet peut se résumer en un seul component suivant :

A.3 Implémentation sur Nexys 2

En chargent le .bit, l'utilisateur pourra utilisé le fréquencemètre.

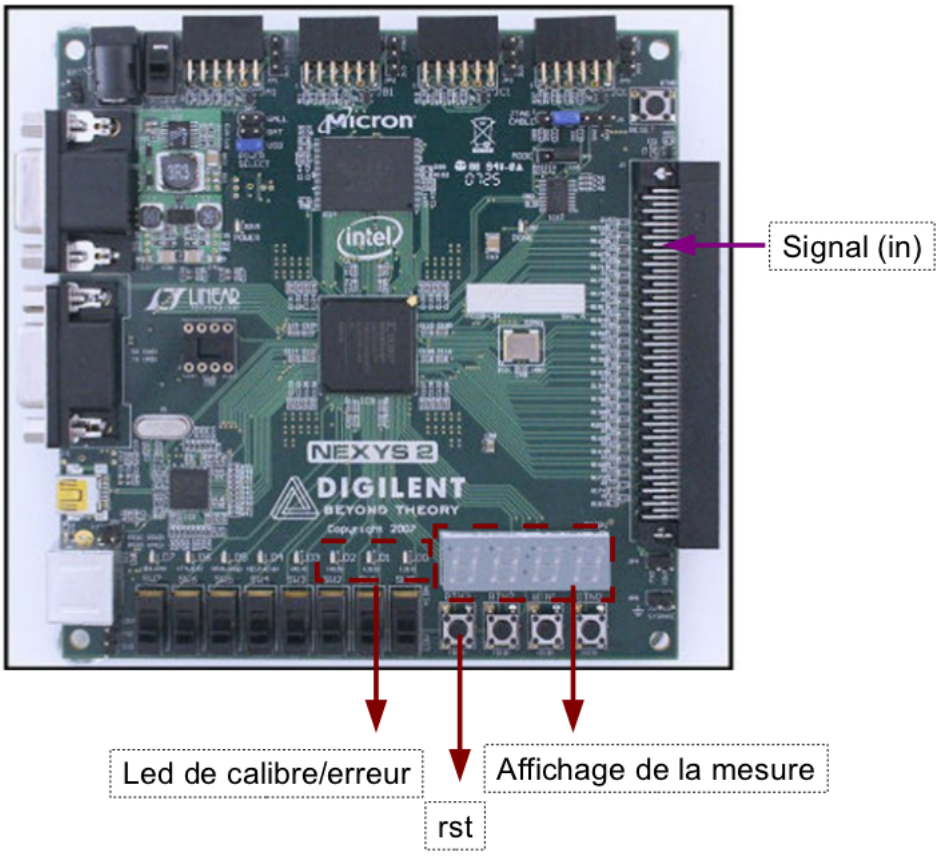

FIGURE 17 – Implémentation sur carte Nexys 2

A.4 Fichier contrainte

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

```
net "rst" loc = "R17";
net "clk" loc = "B8";
net "mhz" loc = "R4";


net "segments[0]" loc = "L18";
net "segments[1]" loc = "F18";
net "segments[2]" loc = "D17";
net "segments[3]" loc = "D16";
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

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    18:18:06 05/06/2012
-- Design Name:
-- Module Name:    nombre_24bits_generateur - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Poue le test de notre module d'affichage
--
-- affiche une puissance de deux par seconde
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity nombre_24bits_generateur is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           nombre : out  STD_LOGIC_VECTOR (23 downto 0));
end nombre_24bits_generateur;

architecture Behavioral of nombre_24bits_generateur is
type etat is (init, calc_nombre, attente);

signal etatf : etat; --etat futur
signal etatp : etat; --etat present

signal timer : STD_LOGIC_VECTOR (23 downto 0);

begin

--Bloc F
process(etatp, rst)
begin
    if rst='0' then etatf <=init ;
    else
        case etatp is
        when init =>

        when others => etatf <= init;

        end case;
    end if;
end process;

--Bloc M
process(clk, rst)
begin
    if (clk'event and clk = '1') then etatp <= etatf ;
    end if;
```

```
end process;

--Bloc G
process(clk)
begin
if(clk'event and clk='1') then

    case etatp is
    when init =>

    when others =>

    end case;

end if;
end process;

--autres process
process(step)
begin

end process;


end Behavioral;
```