

# Projet Electronique Numérique

Réalisation d'un Fréquencemètre

JÉRÉMIE FOURMANN & MAXIME MORIN  
(Promo 2013 - Électronique)

7 juin 2012



## Plan

<b>1</b>	<b>Objectifs</b>	<b>2</b>
1.1	Rappel du cahier des charges	2
1.2	Nos choix	2
<b>2</b>	<b>Conception du système</b>	<b>4</b>
2.1	Présentation du système	4
2.2	Module d'affichage	5
2.3	Mesure par étalon	7
2.4	Mesure par échantillonnage	9
2.5	Décision	10
2.6	Performances du système	11
<b>3</b>	<b>Bilan</b>	<b>12</b>
<b>A</b>	<b>Manuel d'utilisateur</b>	<b>13</b>
A.1	Introduction	13
A.2	Component VHDL	13
A.3	Implémentation sur Nexys 2	13
A.4	Fichier contrainte	14
<b>B</b>	<b>Extrait code vhd1</b>	<b>15</b>
B.1	Module trouver-digit	15
B.2	Code complet	17

# 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 parmi 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'afficher 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 erronées 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.

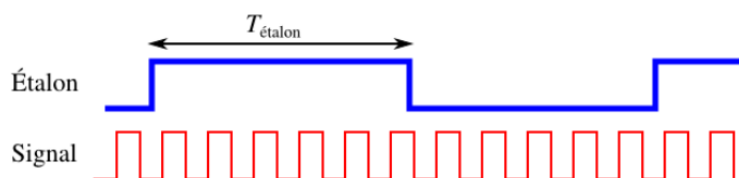


FIGURE 1 – Mesure de fréquence par étalonnage



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'appliquer 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 chacune de ces deux méthodes par les équations suivantes :

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

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

Avec  $F_{clk} = 50MHz$  et  $F_{etalon} = 1Hz$

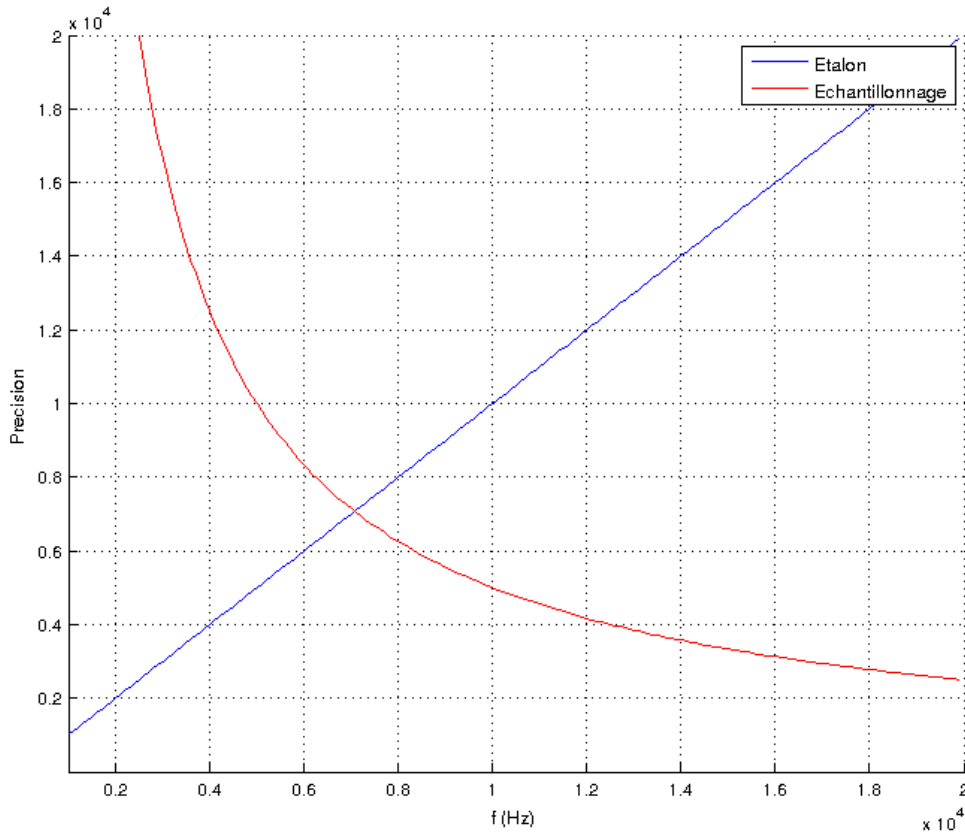


FIGURE 3 – Précision des deux méthodes

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 séquentiel 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 des résultats après implémentation au plus proche des simulations. Par ailleurs nous avons aussi une meilleur lisibilité de notre programme et la méthodologie de débogage 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 implémentons sur le FPGA ressemble à ceci :

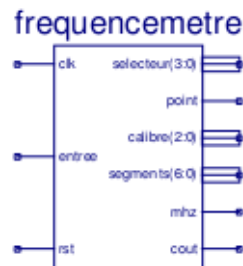


FIGURE 4 – Diagramme A-0

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 :

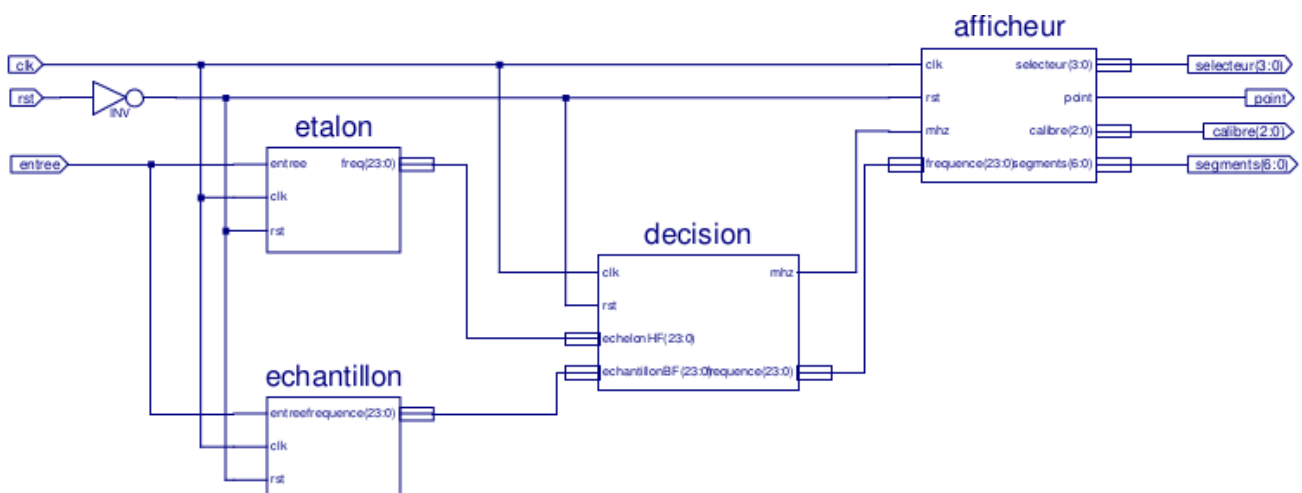


FIGURE 5 – Diagramme A0

Notre 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 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 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

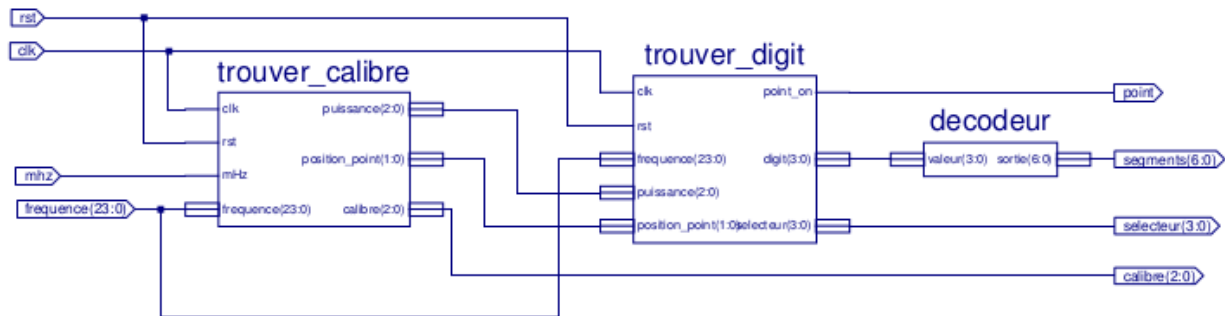


FIGURE 6 – Diagramme A1 : module d'affichage

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 décimal 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). Ce bloc est combinatoire.

**trouver\_digit :** Ce module convertit en binaire codé décimal et affiche un à un les quatres digits significatifs de la fréquence. Il utilise des données déterminées 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.

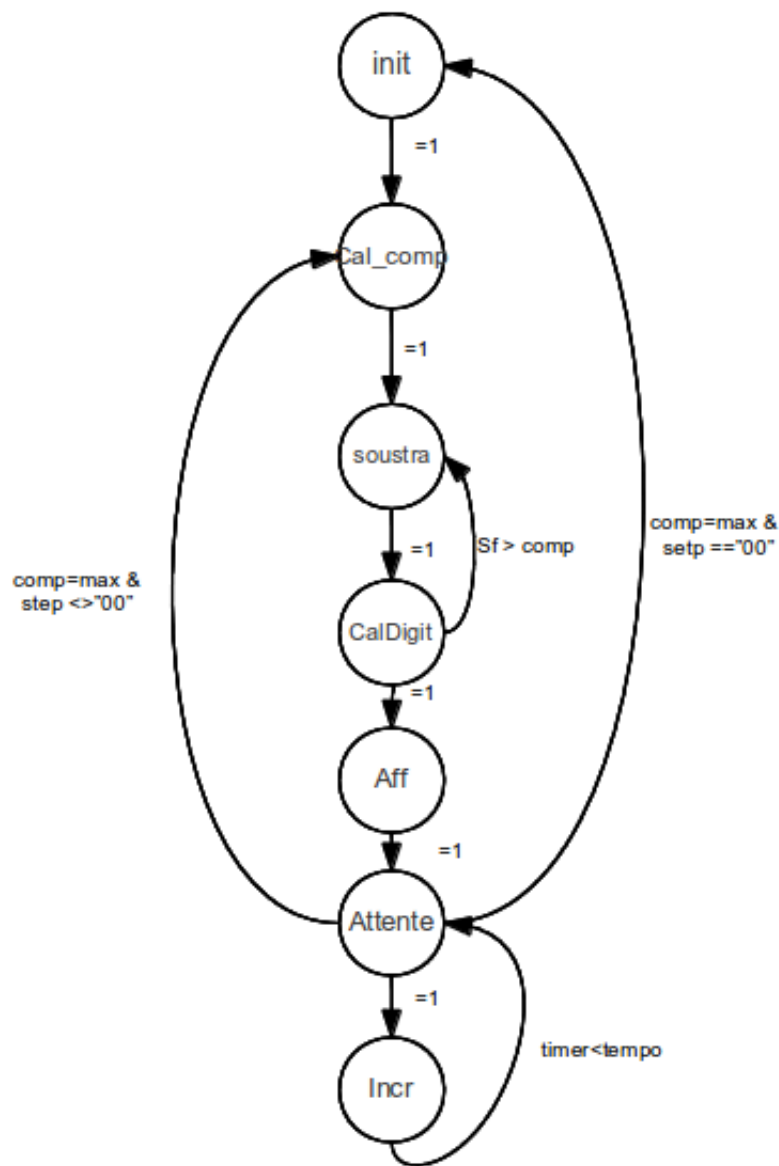


FIGURE 7 – Machine d'état de calcul\_digit

Le signal *step* permet de sélectionner le digit de l'afficheur qui restera éclairé pendant 2ms. Cette machine d'état est d'après nous la plus intéressante de ce projet. Son code source est donné en annexe.

## 2.3 Mesure par étalon

### 2.3.1 Schéma bloc

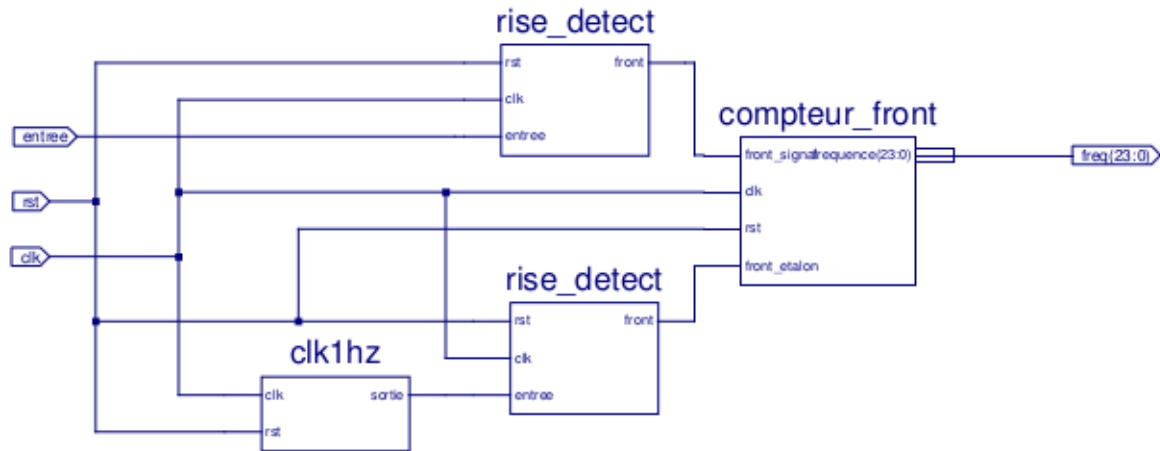


FIGURE 8 – Diagramme A2 : mesure par étalon

L'objectif de ce module est de compter le nombre de front du signal pendant un temps fixe appelé "étalon". La période de cet étalon a été fixé précédemment 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 terme 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

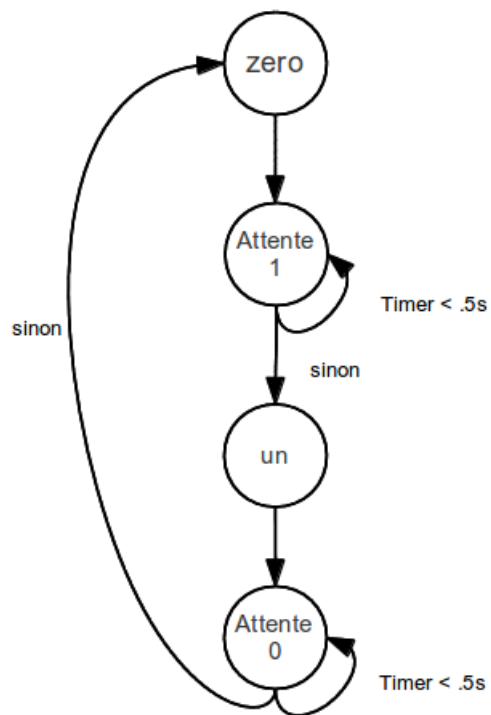


FIGURE 9 – Machine d'état de Clock 1kHz

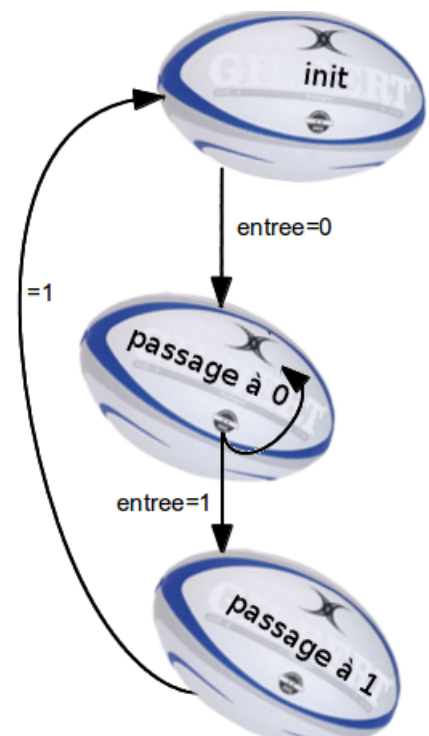


FIGURE 10 – Machine d'état de Rise\_detect

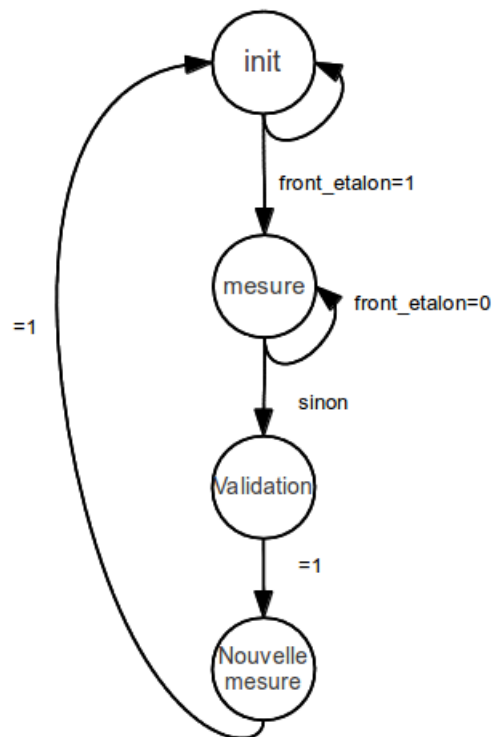


FIGURE 11 – Machine d'état de compteur\_front



## 2.4 Mesure par échantillonnage

### 2.4.1 Schéma bloc

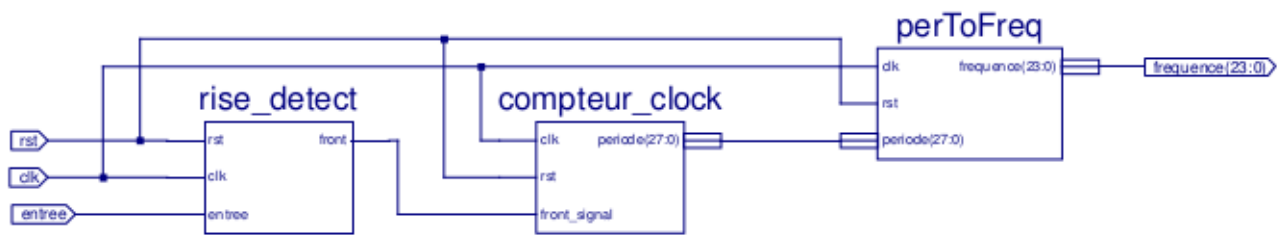


FIGURE 12 – Diagramme A3 : mesure par échantillonnage

L'objectif de ce module est de compter le nombre de front de l'horloge pendant deux front du signal d'entrée. 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.

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

**perToFreq** : Ce bloc permet de calculer la fréquence en effectuant une division.

### 2.4.2 Machines d'états

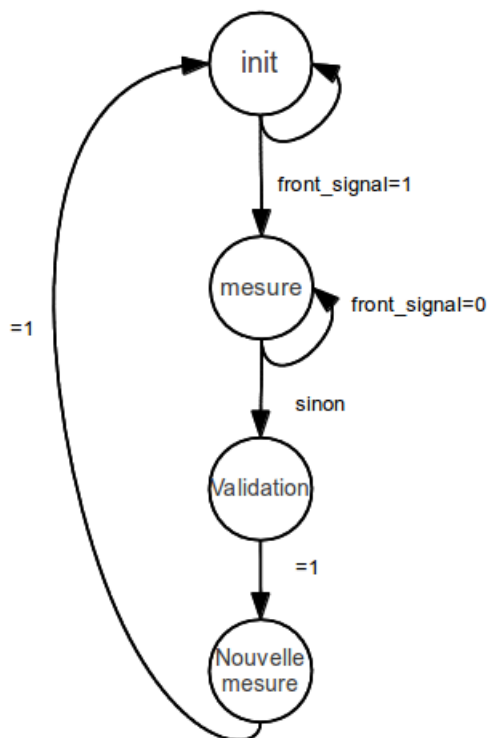


FIGURE 13 – Machine d'état de compteur\_clock

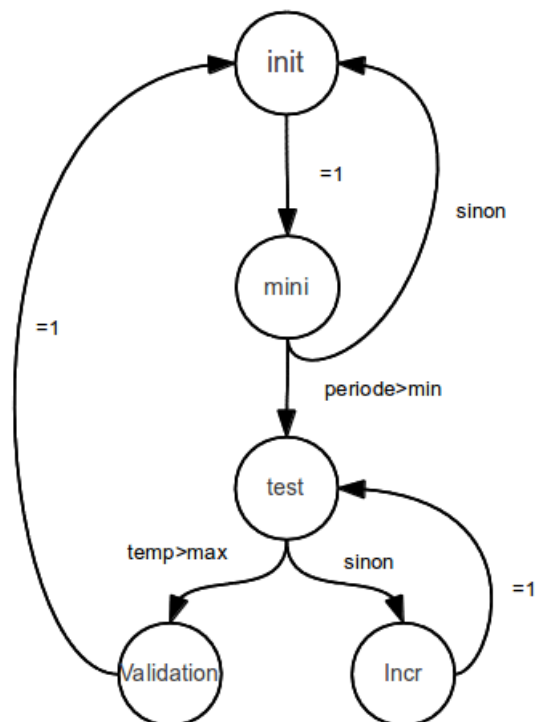


FIGURE 14 – Machine d'état de PerToFreq

## 2.5 Décision

Nous rapelons la composition finale du fréquencemètre :

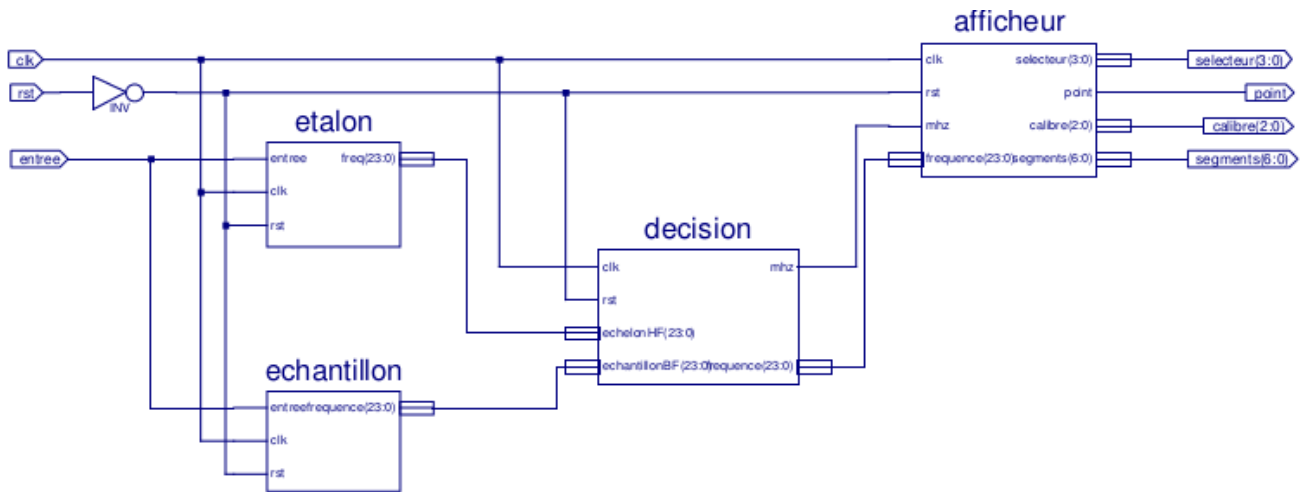


FIGURE 15 – Diagramme A0

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 :

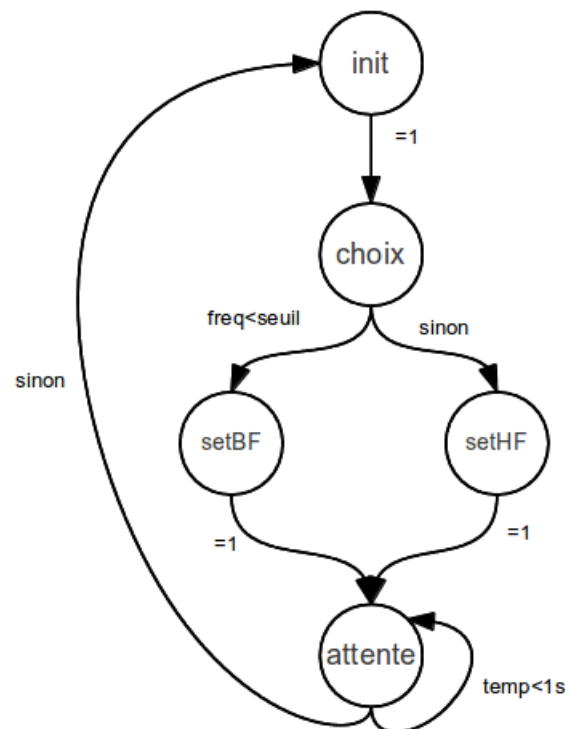


FIGURE 16 – Machine d'état du bloc de décision

## 2.6 Performances du système

### 2.6.1 Erreur de mesure

Nous avons une suite de mesures à différentes fréquences pour regarder l'erreur commise. Une fois le calibrage effectué, on a une erreur maximale de 0.15%, ce qui nous convient parfaitement.

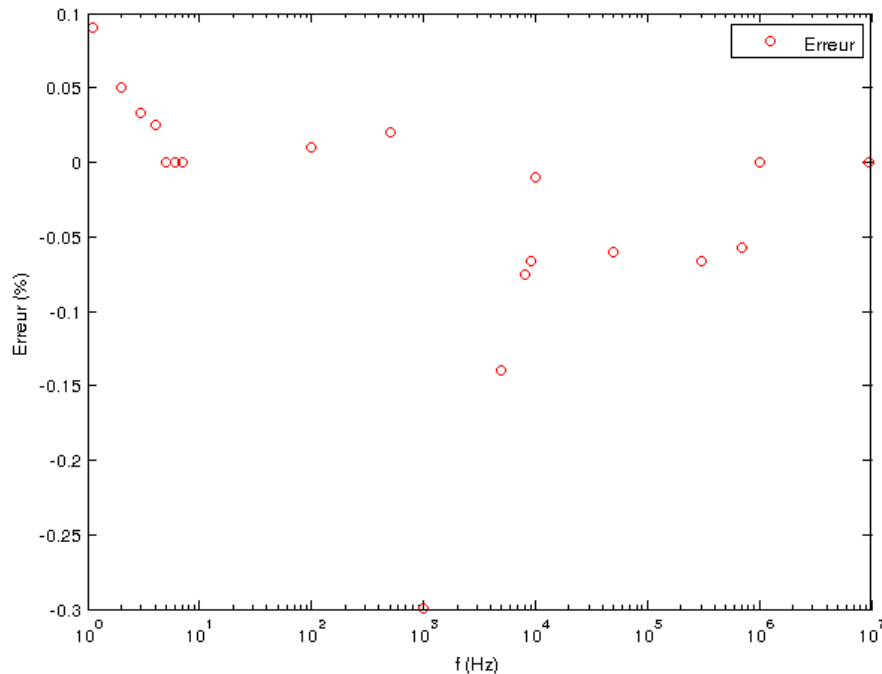


FIGURE 17 – Erreur de mesure de nôtre fréquencemètre

On s'aperçoit que l'on a un pic d'erreur aux alentours de 7KHz, fréquence à laquelle on bascule de méthode pour augmenter la précision.

### 2.6.2 Ressources mobilisées

Dans sa version finale, avec les réglages d'optimisation par défaut, le fréquencemètre utilise les ressources suivantes :


Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	375	9,312	4%	
Number of 4 input LUTs	559	9,312	6%	
Number of occupied Slices	433	4,656	9%	
Number of Slices containing only related logic	433	433	100%	
Number of Slices containing unrelated logic	0	433	0%	
Total Number of 4 input LUTs	742	9,312	7%	
Number used as logic	559			
Number used as a route-thru	183			
Number of bonded <a href="#">IOBs</a>	20	232	8%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.05			

FIGURE 18 – Ressources mobilisées

### 3 Bilan

Le fréquencemètre final répond bien au cahier des charges. Cependant, des améliorations sont possibles : un système de calibration automatisé rendrait le système indépendant de l'horloge de la carte. Il faudrait cependant veiller à garder les mêmes ordres de grandeur en fréquence d'horloge pour ne pas faire déborder les signaux...

Ce projet nous a aussi permis de mieux appréhender les rudiments de la modélisation et la synthèse VHDL. Nous avons appris le synoptique de base qu'il faut suivre pour développer une application électronique numérique en VHDL. La conception entièrement synchrone à l'aide des machines d'état nous a permis d'obtenir des résultats pratiques identiques à la simulation.

## 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 :

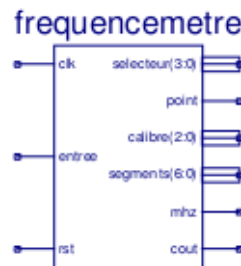


FIGURE 19 – "Composant" du projet fréquence

### A.3 Implémentation sur Nexys 2

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

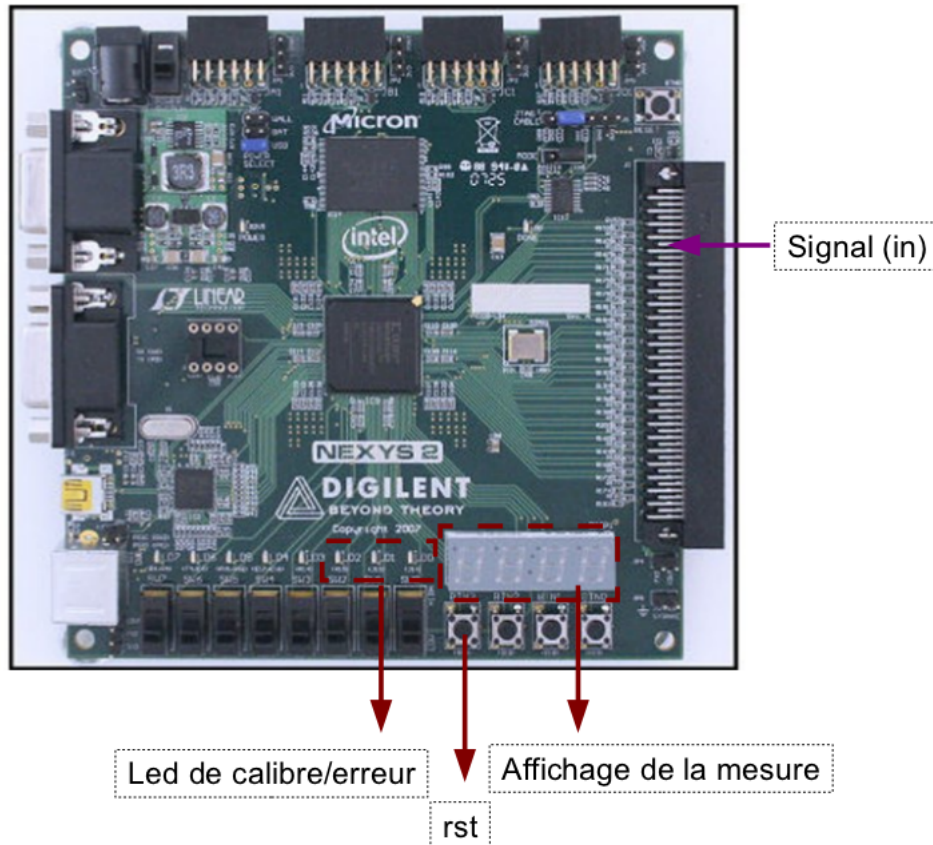


FIGURE 20 – Implémentation sur carte Nexys 2

## A.4 Fichier contrainte

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

```
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 code vhd

### B.1 Module trouver-digit

```
-----
2  -- Jeremie Fourmann & Maxime Morin
3  -- Projet Numerique 2EN 2012 - Realisation d'un frequencemetre
4  --
5  -- Fichier : trouver_digit.vhd
6  -- Description :
7  -----
8  library IEEE;
9  use IEEE.STD_LOGIC_1164.ALL;
10 use IEEE.STD_LOGIC_UNSIGNED.ALL;
11 use IEEE.STD_LOGIC_ARITH.ALL;
12
13
14 entity trouver_digit is
15     Port ( frequence : in  STD_LOGIC_VECTOR (23 downto 0);
16           puissance : in  STD_LOGIC_VECTOR (2 downto 0);
17           position_point : in  STD_LOGIC_VECTOR (1 downto 0);
18           digit : out  STD_LOGIC_VECTOR (3 downto 0);
19           point_on : out  STD_LOGIC;
20           clk : in  STD_LOGIC;
21           rst : in  STD_LOGIC;
22           selecteur : out  STD_LOGIC_VECTOR (3 downto 0));
23 end trouver_digit;
24
25 architecture Behavioral of trouver_digit is
26
27     type etat is (init,calc_digit,attente, incremente,affiche,calc_comp,soustraction);
28     signal etatf : etat; -- etat futur
29     signal etatp : etat; -- etat present
30
31     signal Spuissance : STD_LOGIC_VECTOR (2 downto 0);
32     signal Sfrequent : STD_LOGIC_VECTOR (23 downto 0);
33     signal comp : STD_LOGIC_VECTOR (23 downto 0);
34     signal Sdigit : STD_LOGIC_VECTOR (3 downto 0);
35     signal step : STD_LOGIC_VECTOR (1 downto 0);
36
37 begin
38
39     --Bloc F
40     process(etatp, rst,Sfrequent,comp,step)
41     begin
42         if rst='0' then etatf <=init ;
43         else
44             case etatp is
45                 when init => etatf <= calc_comp;
46
47                 when calc_comp => etatf <= soustraction;
48
49                 when soustraction => etatf <= calc_digit;
50
51                 when calc_digit =>
52                     if(Sfrequent < comp) then etatf <= affiche;
53                     else etatf <= soustraction;
54                     end if;
55
56                 when affiche => etatf<= attente;
57
58                 when attente =>
59                     if(comp = X"00C350") then
60                         if(step = "00") then etatf <= init;
61                         else etatf <= calc_comp;
62                         end if;
```

```

        else etatf <= incremente;
64         end if;

66         when incremente => etatf <= attente;

68         when others => etatf <= init;

70         end case;
        end if;
72     end process;

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

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

86             case etatp is
            when init => step <= "00"; Sfrequence <= frequence; Spuissance<=puissance;

88             when calc_comp => Sdigit <= "0001";
90                 if (Spuissance="110") then comp <= X"0F4240"; --10^6
                    elsif (Spuissance="101") then comp <= X"0186A0"; --10^5
92                     elsif (Spuissance="100") then comp <= X"002710"; --10^4
                    elsif (Spuissance="011") then comp <= X"0003E8"; --10^3
94                     elsif (Spuissance="010") then comp <= X"000064"; --10^2
                    elsif (Spuissance="001") then comp <= X"00000A"; --10^1
96                     elsif (Spuissance="000") then comp <= X"000001"; --10^0
                    else comp <= X"000001"; --10^0
98                     end if;

100             when soustraction => if (Sfrequence >= comp) then Sfrequence <= Sfrequence - comp;
102                                     else Sdigit <= "0000";
                                        end if;

104             when calc_digit =>
106                 if (Sfrequence >= comp) then Sdigit <= Sdigit + 1;
                    end if;

108             when affiche =>
110                 step <= step - 1 ;
                    Spuissance <= Spuissance - 1;
112                 comp <= X"000000"; --Le signal va servir pour le comptage
                    digit <= Sdigit;

114             when incremente =>
116                 comp <= comp + 1; -- c est le bloc F qui verifie si on depasse la valeur

118             when others =>

120             end case;

122         end if;
    end process;

124     --dec_sel
    process(step)
126     begin
128         if (step="00") then selecteur <= "0111";

```



```

130     elsif(step="01") then selecteur <= "1011";
131     elsif(step="10") then selecteur <= "1101";
132     else selecteur <= "1110";
133     end if;
134 end process;
135
136 --point_sel
137 process(position_point, step)
138 begin
139     if(position_point = step) then point_on <='0';
140     else point_on <= '1';
141     end if;
142 end process;
143
144 end Behavioral;

```

## B.2 Code complet

L'intégralité du code source relatif à ce projet est disponible à l'adresse :

<https://github.com/maxn7/Frequence metre>