METRONOME

FPGA-Basys 3

2017

Réalisé par

LEVIEUX Victor

Encadré par M.Kessal

Table des matières

Présentation

Definition

> Appareil servant à marquer la pulsation rythmique d'un morceau de musique. (Il contrôle la régularité des temps et précise la vitesse d'exécution des différents tempos. Son invention par Winkel a été brevetée en 1816 par J. N. Mälzel.)
>
> *Dictionnaire Larousse*

Introduction

Un musicien doit-il utiliser un métronome pour s'exercer ou jouer ? Quels en sont les avantages et les inconvénients ? Le métronome peut-il être nuisible à la musicalité ?
Autant de questions qu'il est légitime de se poser quand on sait que le tempo est la base de la plupart des musiques et qu'il est le premier langage commun partagé par les musiciens.

Très longtemps le métronome a été mécanique, un bras avec un balancier muni d'un poids se déplaçant sur le bras, permettant de varier le tempo .
Même s'il existe encore aujourd'hui il a été remplacé par les métronomes électroniques (matériels ou logiciels) ainsi que les boites à rythmes et autres machines permettant non seulement de jouer un tempo mais aussi de reproduire des rythmes complexes, les décompositions du temps, etc.

Il semble évident à tout le monde qu'un métronome sert à donner le tempo. Or, il existe de multiples façon de l'utiliser et ce dans différents contextes :

- la pratique instrumentale
- le jeu en groupe
- le concert
- l'enregistrement en studio
- le mixage

Cependant, il n'est pas évident de suivre un rythme qui soit seulement sonore ou visuel dans des conditions inadéquates :

- salle bruyante, exemple : salle de répétition
- salle avec de nombreux éclairages : salle de concert

C'est pourquoi un métronome se doit d'être utilisable sous de nombreuses conditions et modifiable à souhait.

Descriptif du projet

Le projet a pour but de créer un métronome lumineux et sonore en adéquation avec le besoin de l'utilisateur.

Le projet est composé d'une vingtaine de module. Chaque module est associé à son TestBench. Le projet a été réalisé sous **ModelSim** et implémenté avec **Vivado**.

Tous les modules VHDL ont été réalisés par Victor LEVIEUX,

sauf le cmpt_raz, qui est un compteur synchrone avec remise à zéro qu'on peut retrouver sur le site suivant : https://fr.wikibooks.org/wiki/TD3_VHDL_Compteurs_et_registres.

Moyens de réglage

Différents moyens de réglages pour les **battements par minutes** (**BPM**) seront mis à dispositions qui devront être faciles d'utilisation.

Pour régler linéairement les BPM, le métronome devra être équipé avec deux boutons : augmenter les BPM et diminuer les BPM.

Pour obtenir un réglage rapide des BPM, le métronome devra être équipé d'un bouton "TAP" qui servira à marquer le rythme.

Pour changer entre les différentes mesures, le métronome disposera d'un autre bouton.

Pour activer ou désactiver le son, un switch on/off sera utilisé.

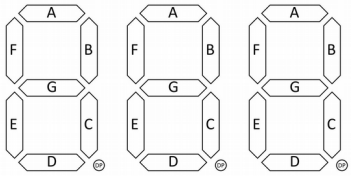
Les entrées :

- 1 Bouton +1
- 1 Bouton -1
- 1 Bouton "Tap"
- 1 Bouton "mesure"
- 1 Switch activation/désactivation son

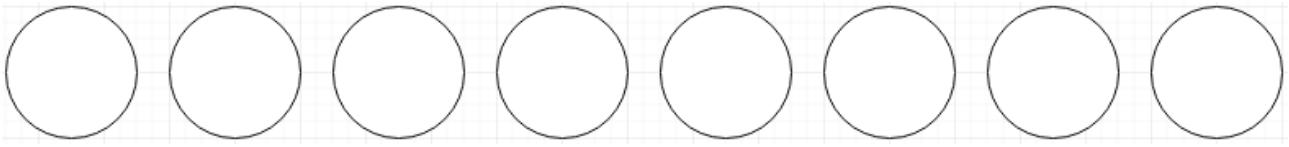
Différentes sorties

Différentes sortie vont être utilisées, mais elles se doivent toutes d'être lisibles rapidement et instinctives.

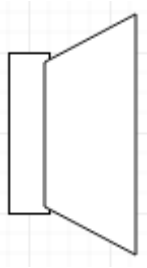
L'utilisateur disposera d'un affichage des BPM et du mode actuel sur 3 afficheurs 7 segments.



Un bandeau de 8 LEDs sera à disposition et effectuera un balayage régulier à la vitesse des BPM, c'est-à-dire, 1 aller correspond à l'intervalle entre 2 battements.



Le métronome sera aussi équipé d'un buzzer qui émeterra un son lorsque les LEDs aux extrémités du bandeau seront allumées et le son sera différent, selon la mesure en cours.



Les sorties :

- 3 afficheurs 7 segments
- 1 bandeau de 8 LEDs
- 1 Buzzer

Fonctionnalités

Fonctionnement générale

Le métronome sera initialisé avec une valeur de 120 BPM au démarrage qui sera affiché sur les afficheurs. Suite à cela, le bandeau de LEDs sera activé automatiquement et entamera son balayage de gauche à droite, puis de droite à gauche. Il proposera une gamme de 50 à environ 500BPM.

Appui sur le bouton "+1" ou le bouton "-1"

Lors de l'appui sur un des deux boutons, le système testera l'état des boutons toutes les 100ms. Ce qui permet une sensibilité normal. C'est-à-dire, assez rapide pour les importants changements de BPM (±10 BPM/s) et assez lente pour avoir un réglage plus fin (1 BPM). La vitesse de balayage des LEDs s'adaptera alors immédiatement au nouveau rythme.

Appui régulier sur le bouton "tap"

L'utilisateur a pour moyen de rentrer une valeur de BPM en appuyant sur le bouton à cette fréquence-ci. Un minimum de 4 appuis sera nécessaire pour obtenir une réponse de la part du système. Dans le cas où la valeur ne convient pas, l'utilisateur peut continuer à appuyer ou alors il peut affiner sa sélection avec les boutons +1/-1. Si l'utilsateur s'arrête d'appuyer ou n'appuie pas pendant un laps de temps trop long. Il faudra reprendre la démarche au début.

Appui sur le bouton "mesure"

Lors du premier appui sur le bouton "mesure", l'afficheur 7 segments affichera le type de mesure en cours d'utilisation. Les appuis suivants, dans un cours laps de temps de l'ordre de la seconde, servirons à changer le type de mesure parmi une liste préselectionnée.

Affichage sur les 3 afficheurs

Les 3 afficheurs afficherons continuellement les BPM ou la mesure en cours (voir appui sur le bouton "mesure").

Balayage des LEDs

Le balayage des LEDs adaptera sa vitesse en temps réel au BPM en cours. Chaque LED s'allumera pendant le même laps de temps par rapport aux autres. L'allumage successif effectuera une infinité d'aller-retours.

Emission du son

Le buzzer, s'il est activé, émettra un son de 440Hz pendant la durée d'activation des LEDs positionnées aux extrémités.

Prototype

Préparation

Après avoir défini précisément chaque fonctionnalité du métronome en fonction des besoins de l'utilisateur. Il était question donc de réaliser le système sur une carte FPGA : la Basys 3.

Pour cela, les séances suivantes ont servi à réaliser un schéma du système dans sa globalité sans rentrer dans les détails de chaque bloc. L'intérêt était d'avoir une idée du travail nécessaire et l'organisation des tâches sur les différentes séances.

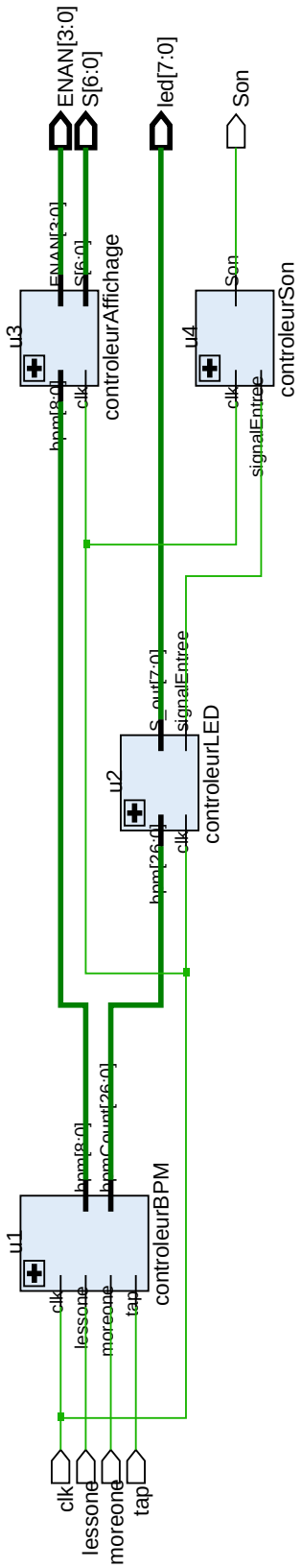
Etude du schéma global

Conformément aux différentes entrées et sorties, le schéma global est composé de :

Nom	Type	Taille	Description
CLK	in STD_LOGIC	1	Permet le fonctionnement du système et la synchronisation de tous les modules
lessone	in STD_LOGIC	1	Bouton "-1"
moreone	in STD_LOGIC	1	Bouton "+1"
tap	in STD_LOGIC	1	Bouton "Tap"
ENAN	out STD_LOGIC_VECTOR	4	Selectionne les afficheurs
S	out STD_LOGIC_VECTOR	7	Active les différents segments d'un afficheur
led	out STD_LOGIC_VECTOR	8	Active le bandeau de LED
Son	Out STD_LOGIG	1	Emet un signal carré positif

Afin de traiter les différents signaux entrants, le système est composé de 4 modules :

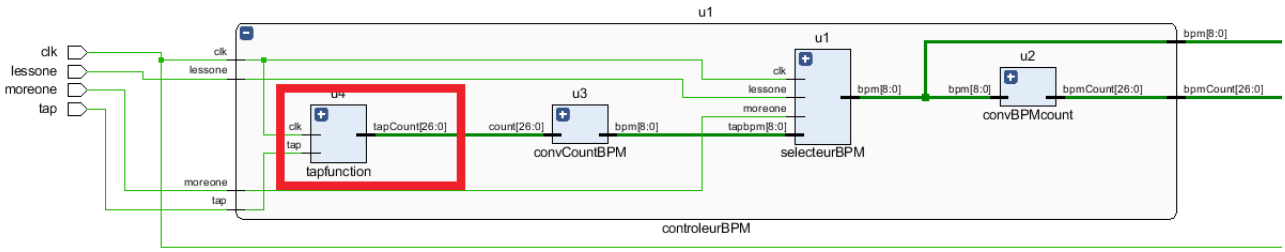
Nom	Fonctionnalité
controleurBPM	Retourne la valeur en BPM
controleurAffichage	Contrôle les afficheurs 7 segments
controleurLED	Effecture le balayage
controleurSon	Emet un son


ENAN[3:0]
S[6:0]
led[7:0]
Son
u3
controleurAffichage
u4
controleurSon
u2
controleurLED
u1
controleurBPM
clk
lessone
moreone
tap

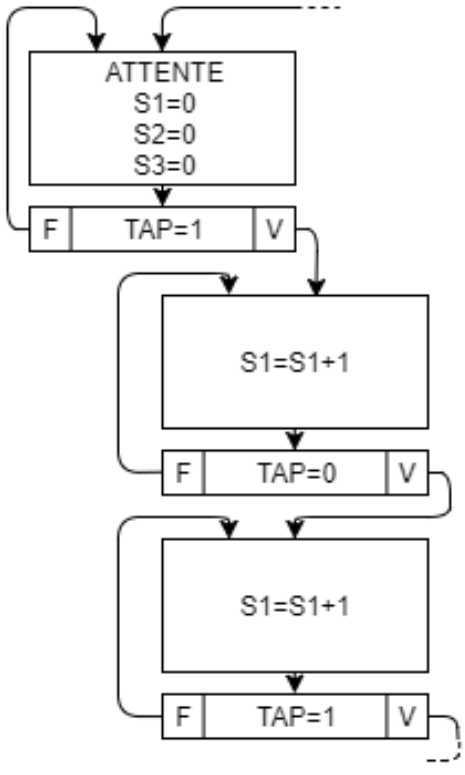
ControleurBPM

Le module controleurBPM permet de contrôler les entrées et d'obtenir une valeur correspondante en BPM. Il est composé de 4 sous modules, qui sont les suivants :

- tapfunction (le plus intéressant)
- convCountBPM
- selecteurBPM
- convBPMcount



tapfunction



Le module "tapfunction" est composé d'un séquenceur et un sous-module : "moyenneur". Il convertit l'appui régulier sur son entrée en une valeur moyenne du temps entre chaque impulsion.

Pour ce qui est du séquenceur, il fonctionne de la manière suivante.

Le module est connécté au bouton "TAP", offre une sortie count STD_LOGIC_VECTOR (27bits) et possède 3 signaux internes S1, S2, et S3, tout trois STD_LOGIC_VECTOR (27bits) qui vont servir à calculer.

Remarque : Xilinx préviligie l'utilisation de signaux internes comme intermédiaires de calcul au lieu des variables types BUFFER.

Description du séquenceur :

En premier lieu, le système est en état "ATTENTE". Il initialise les signaux internes à 0.

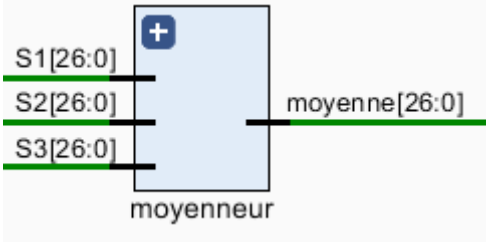
Lorsque le bouton TAP est enclenché, S1 s'incrémente tant que "TAP" est à l'état '1'.

Lorsque le bouton TAP est relâché, S1 continue de s'incrémenter.

Lors du second appui sur le bouton "TAP", S1 n'évolue plus et S2 s'incrémente de la même manière.

Ainsi de suite, jusqu'à l'obtention des valeurs finies S1, S2 et S3.

Ensuite un nouvel état : "MOYEN", acquéris ces 3 valeurs précédentes et en calcul une moyenne avec le module "Moyenneur".



Moyenneur :

Fonctionnalité *: calcul la moyenne des entrées*

Entrée : *S1, S2, S3 STD_LOGIC_VECTOR (27bits)*

Sortie : *count STD_LOGIC_VECTOR (27bits)*

Une fois la moyenne obtenue des trois signaux, elle est envoyée en sortie et le système retourne à l'état d'attente.

Cependant, le séquenceur réel est amélioré de la manière suivante. Si Sx atteint une valeur trop élevé, c'est-à-dire une laps de temps entre les impulsions trop importantes (2 secondes). Le système retourne à l'état "ATTENTE". Ce qui évite les erreurs dans le cas où, Sx deviendrait beaucoup trop grand et entraînerait des érreurs de calcul. Pour une raison qui concerne cette fois-ci l'utilisateur, en cas d'appui malencontreux sur un bouton, il suffit donc d'attendre pour qu'il soit réinitialisé.

ConvCountBPM

Le premier problème qu'engendre le module "tapfunction" est qu'il retourne une valeur sur 27bits. Il retourne le nombre de clock qu'il faut patienter pour avoir la bonne valeur des BPM.

Pour cela, convCountBPM est un convertisseur nombre de clock vers une valeur en BPM codée sur 9 bits.

Pour cela la formule est : $BPM = \frac{60 * nb\,d'échantillon}{nb\,de\,clock}$

Equivalent VHDL : *BPM<=std_logic_vector(RESIZE((second*echant)/UNSIGNED(count), 9));*

avec :

constant second : UNSIGNED (5 downto 0):="111100";

constant echant : UNSIGNED (26 downto 0):="101111101011110000100000000";

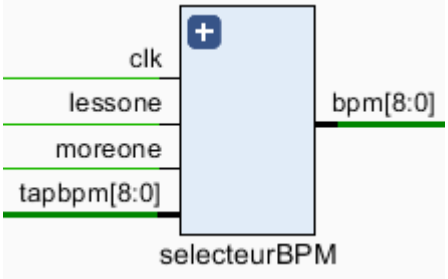
Le principal problème de ce calcul est que c'est un calcul d'inverse. Pour cela, il faut donc utiliser d'autre bibliothèque tel que : numeric_std qui permet le calcul de division. Cependant, il ne traite pas les STD_LOGIC_VECTOR. Il faut donc caster tous les signaux en UNSIGNED pour ce calcul. Puis les recaster sous des STD_LOGIC_VECTOR. Or le calcul peut donner une grande variété de valeur qui n'auront pas besoin du même nombre de bits pour être coder. Il faut donc forcer la taille avec la fonction RESIZE, fourni avec la bibliothèque.

Remarque : La fonction RESIZE n'apporte pas de perte d'information, car la valeur retournée est toujours plus petite ou égale au maximum codée sur 9bits.

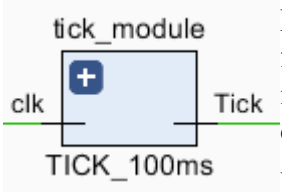
ConvBPMCount

Ce module est l'inverse du module précédent et sera utilisé dans la suite. Il possède la formule et le même comportement sauf que l'entrée et la sortie sont inversées.

SelecteurBPM



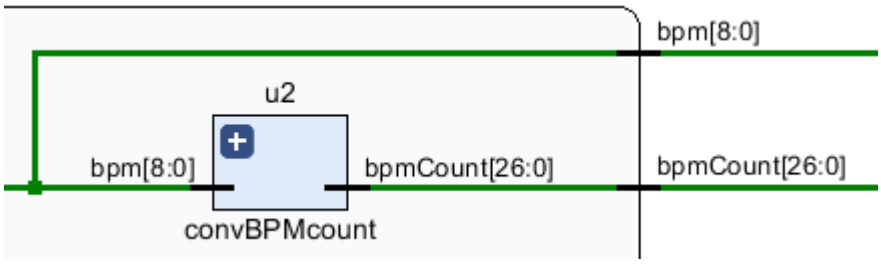
Le module selecteurBPM a pour but de contrôler la valeur des BPM. C'est-à-dire augmenter ou réduire les BPM en fonctions des appuis sur les boutons "+1/-1", ou de mettre en BPM la valeur donnée par le tapfuction si celle-ci est différente de 0.



Pour optimiser et rendre utilisable les différents boutons, il faut réduire la fréquence de travail de ce module. Pour cela, un module "TICK_100ms" est introduis dans le module "selecteurBPM", qui a pour objectif de ralentir la clock à une période de 100ms.

La répercussion principale est la suivante. L'appui sur un bouton est pris en compte toutes les 100ms. Ce qui permet d'être précis et de faire varier les BPM de +1 ou -1 facilement. Mais aussi d'être rapide : l'appui prolongé sur une 1s permet une évolution de +-10BPM.

Sortie du controleurBPM



Pour ce qui est des sorties, le module "controleurBPM" offre 2 sorties possibles : *BPM* sur 9bits (valeur décimales codé en base 2) et *bpmcount* sur 27bits (nombre de clock à attendre).

Il existe deux sorties, car chacune sera utilisé dans des modules différents pour le traitement des sorties.

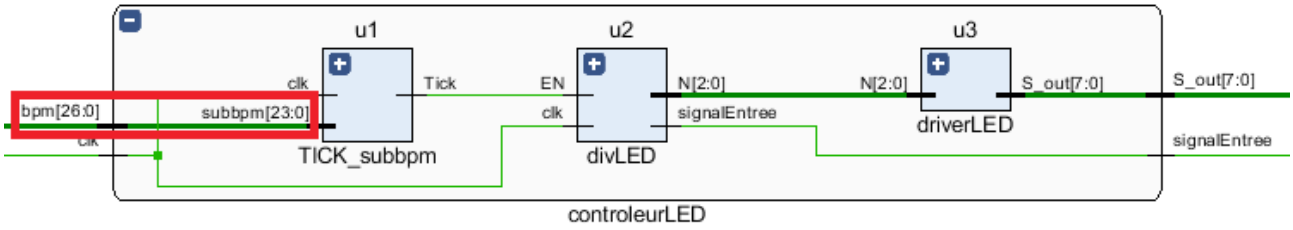
ControleurLED

Le module controleurLED permet de traduire les BPM en un balayge régulier de LED, qui seront allumées succesivement. Il est composé de 4 sous modules, qui sont les suivants :

- tick_subbpm
- divLED
- driverLED

Tout d'abord, pour la première chose à remarquer est la taille des signaux : *bpm* (27 bits) devient *subbpm* (24 bits). Cela s'explique au fait que les modules qui vont suivre : TICK_subbpm, divLED et driverLED, ont besoin d'une fréquence de travail 8fois plus élevée que celle imposée par les BPM.

Le métronome est composé d'un bandeau de 8 LED. Il faut donc subdiviser les BPM en 8 sous periodes afin de pouvoir contrôler les LED une à une pendant la durée d'un battement : augmenter la fréquence de travail revient à diminuer le nombre de clock à compter.

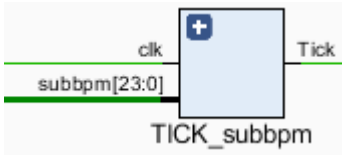


La solution retenue la plus efficace est la méthode du décalage (*shift*), au lieu d'une division normal. Car diviser par 8 revient à décaler vers la droite de 3 bits. Ce calcul peut se faire donc aisément en un seul front d'horloge.

subbpm <= bpm(26 downto 3);

TICK_subbpm

Le premier sous-module est TICK_subbpm. L'objectif principale est un enable à la fréquence subbpm et non celle de l'horloge interne. A l'aide d'un compteur, il mettra donc une sortie *Tick* à l'état haut une fois la valeur subbpm atteinte.



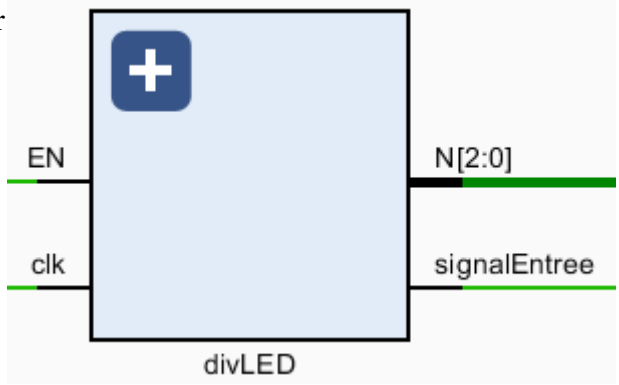
DivLED

Le module divLED à pour fonctionnalité de générer la valeur *N* (comprise entre 0 et 7) à chaque *EN* qui est à l'état "1" tous les subbpm.

N évoluera de la manière suivante :

- De 0 à 8
- Puis de 8 à 0

N=000 : LED N°1; N=001 : LED N°2; ...

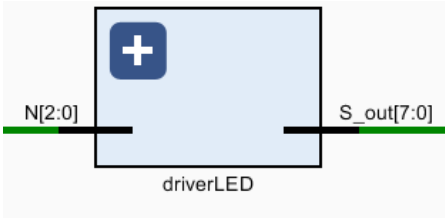


Pour cela, il est équipé d'un signal interne : *retour (*STD_LOGIC*)*. Le signal *retour* retient si *N* doit augmenter ou diminuer, en fonction de sa valeur précédente.

Le module divLED est aussi équipé d'une sortie *signalEntree* (STD_LOGIC) qui permettra la synchonisation entre le module controleurSON et le module controleurLED. La sortie *signalEntree* vaudra la valeur "1" lorsque *N* vaut 000 ou 111. C'est-à-dire, lorsque le balayage se trouve à une extrémité du bandeau de LED.

DriverLED

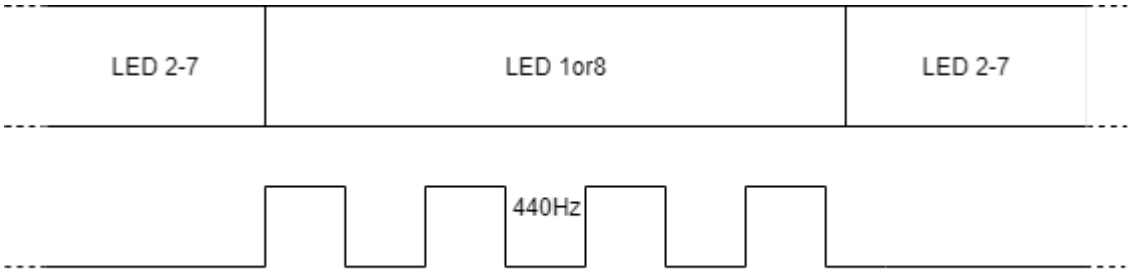
Le module driverLED est un tableau 1x8, qui contient tous les états du bandeau de LED. Il convertit l'entrée *N* (3 bits) en une sortie *S* (8 bits : 1 bit pour chaque LED).



N (états)	S (sortie)	Aspect visuel des LEDs
000	1000000	
001	0100000	
111	0000001	

ControleurSON

Le module controleurSON à pour fonctionnalité d'émettre un son à 440Hz (ou autre) lorsque *signaleEntree* est à l'état haut, donc lorsque la LED N°1 ou N°8 est allumé, de la manière suivante :



Pour cela, le module est équipé de 3 signaux internes :

- *Q* STD_LOGIC : curseur d'un compteur
- *etat* STD_LOGIC : état en cours, état "1" : sortie différente de 0, état "0" sortie nulle.
- *etatprec* STD_LOGIC : état précédent

Fonctionnement

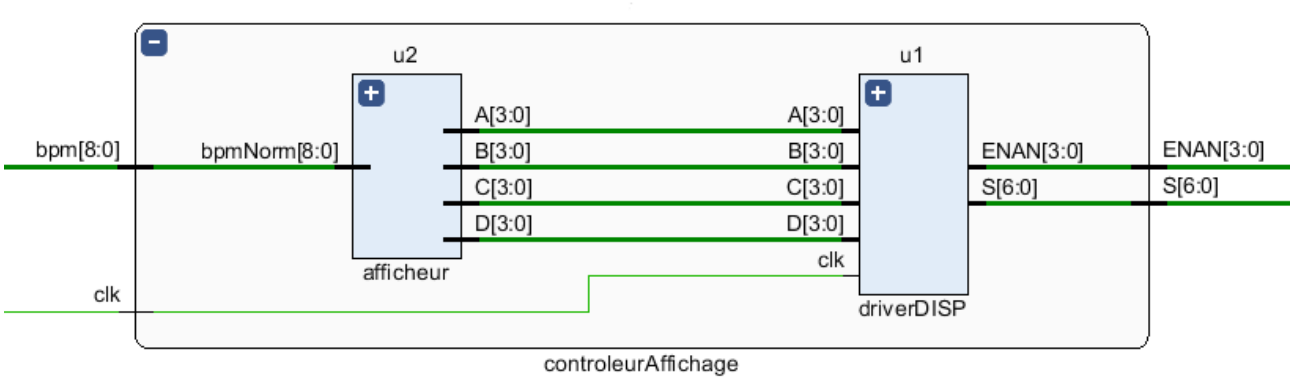
Lorsque le *signalEntrée* est à l'état "1" (LED N°1 ou N°8 activée), le compteur s'enclenche. Ce compteur compte de manière à obtenir du 440Hz. L'état initiale est alors à "1". Lorsque le compteur atteint la valeur max. L'état prend la valeur de l'état précédent, c'est à dire "0". Le compteur recommence à compter. Une fois la valeur maximum atteinte, il reprend la valeur de l'état précédent : "1" et ainsi de suite.

Ce qui génère un signal carré de fréquence 440Hz durant la période où les LED N°1 et N°8 sont allumées.

Remarque : Le signal carré sonore est un peu plus agrésif qu'un signal sinusoïdale car il possède beaucoup d'harmonique aiguë, mais le laps de temps étant assez court, cela n'est pas dérangeant à l'oreille.

ControleurAffichage

Le module controleurAffichage permet d'afficher sur des afficheurs 7 segments la valeur en BPM ou la mesure en cours.



Afficheur

Le module afficheur a pour fonctionnalité de séparer les centaines, dizaines et unités des BPM pour les répartir sur les différents afficheurs.

Pour cela, on utilise la même technique que précédemment : utilisation de la bibliothèque numeric_std et du type UNSIGNED. Nous aurions pu utiliser un compteur BCD pour facilité la tâche. Mais ayant déjà crée le signal BPM. J'ai opté pour la solution du calcul.

A<="0000"; --On ne dépasse pas 500BPM

B<=std_logic_vector(resize(unsigned(bpmNorm)/100, 4));

C<=std_logic_vector(resize((unsigned(bpmNorm) mod 100)/10,4));

D<=std_logic_vector(resize(unsigned(bpmNorm) mod 10,4));

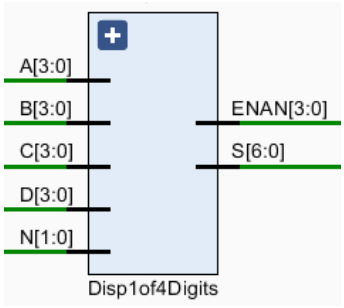
B,C et D prennent respectivement les valeurs des centaines, dizaines, unités. Ils sont codés sur 4 bits, pour afficher l'ensembles des chiffres et des caractères spéciaux qui seront utiliser pour afficher le mot BPM et le type de mesure.

driverDISP

Le module driverDISP est le module qui permet de répartir sur 4 afficheurs 7 segments les signaux A,B,C et D. Il est lui même composé de plusieurs sous-modules :

- TICK1_ms : qui réduit la fréquence
- Compteur : qui compte de 0 à 3 pour sélectionné les différents afficheurs à la fréquence imposée par le module "TICK_1ms"

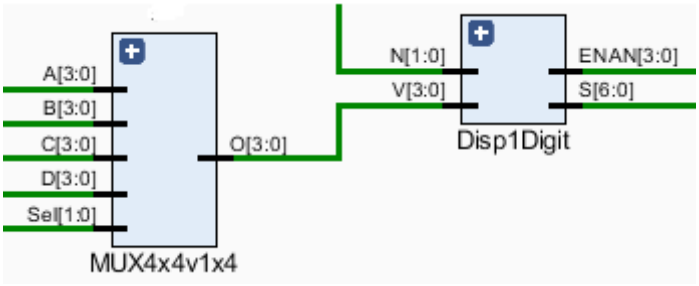
Disp1of4Digits



Le module disp1of4Digits a pour fonctionnalité d'afficher la valeur comprise dans A lorsqu'on sélectionne l'afficheur 1, la valeur comprise dans B lorsqu'on sélectionne l'afficheur 2, de même pour C et D.

Ce même module comprend encore d'autres sous modules :

- MUX4x4v1x4 :sélectionne un signaux de 4 bits parmi 4 signaux entrant
- Disp1Digit : converti le signal entrant en un signal sur 8 bits qui indique les segments à allumer de la même manière que pour les LED, les états possibles des segments sont stockés dans un tableau.



Pour les types de mesures

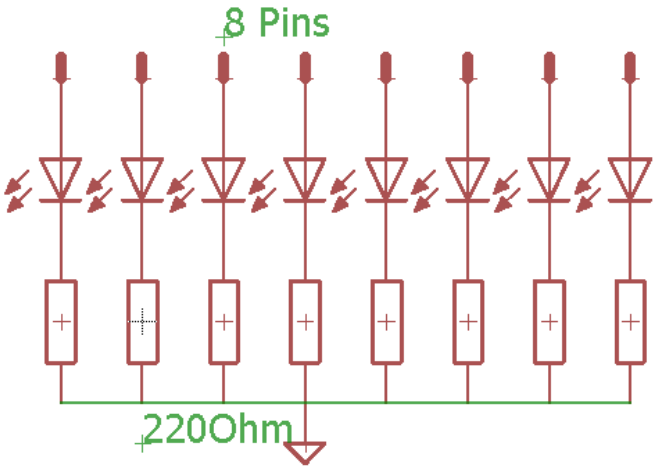
Note : Je n'ai pas eu le temps de créer le module qui gère le type de mesure. Mais il a été pensé et étudié, mais n'a pas eu le temps d'être créé en VHDL. C'est pour cela que dans le tableau pour les afficheurs 7 segments, on retrouve en codé les valeurs "B", "P", "M" et "-" qui avaient déjà été anticipé.

Circuit imprimé et schéma électrique

Le principal objectif est de réaliser un métronome fonctionnel, d'un point de vue pratique. Il faut donc réaliser un circuit électrique pour relier les différentes sorties et entrées.

Les LEDs

Les différentes LEDs sont associées à chacun des pins à 3,3V qui les commandes et sont mises en séries avec des résistances de 220Ω puis reliés à la masse. Par le calcul on trouve qu'elles seront parcourues par un courant de 10mA.



Une idée possible pour réduire le nombre de résistance est d'utiliser une seule résistance pour toutes les LEDs. Effectivement, elles ne sont jamais allumées en même temps.

Le Buzzer

Le modèle retenu est un buzzer de 0,5W et d'une résistance interne de 32Ω. Lui impliquant une tension de 3,3V, le courant passant dans la branche sera de 10mA. La puissance reçu sera donc 0,3W ce qui est en dessous de la puissance nominale et garantira un bon fonctionnement. Il sera directement relier à un pin et relié ensuite à la masse.

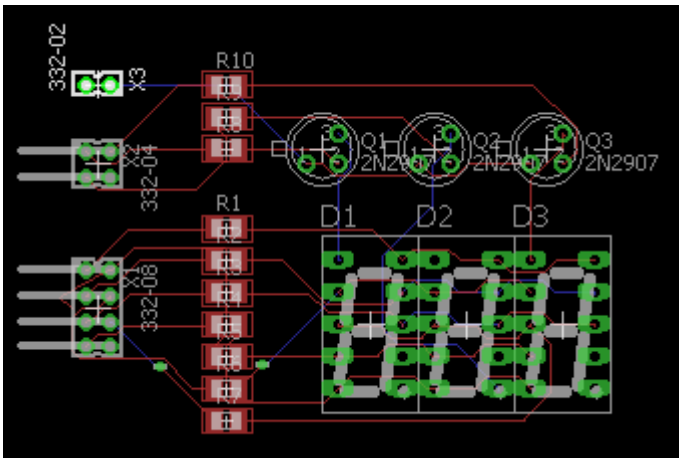
Buzzer : http://www.farnell.com/datasheets/1685143.pdf?_ga=2.87136860.575932278.1497740639-116627149.1496910925

Les afficheurs 7 segments

Le modèle retenu pour les afficheurs sont à anode commune. Etant composé simplement de LED en parallèles, sur les différentes cathodes seront branchées en série des résistances de 220Ω. Ce qui garantira un bon fonctionnement de l'afficheur 7 segments.

Pour ce qui est de la mise en parallèles des afficheurs 7 segments. Des transistors PNP 2N2907A seront positionnés en amont afin de contrôler le courant entrant dans l'afficheur 7 segments. Le collecteur est relié à l'afficheur, la base sera en série avec une résistance de 2kΩ et relié à un pin de commande et l'émetteur à une sortie 3,3V de la puce. Les cathodes identiques pour chaque afficheur seront reliés à la même résistance de 220Ω. Ce qui limitera le nombre de résistance au nombre de 7.

Les pins de commande, reliés à la puce vont contrôler le courant passant dans chaque transistor.

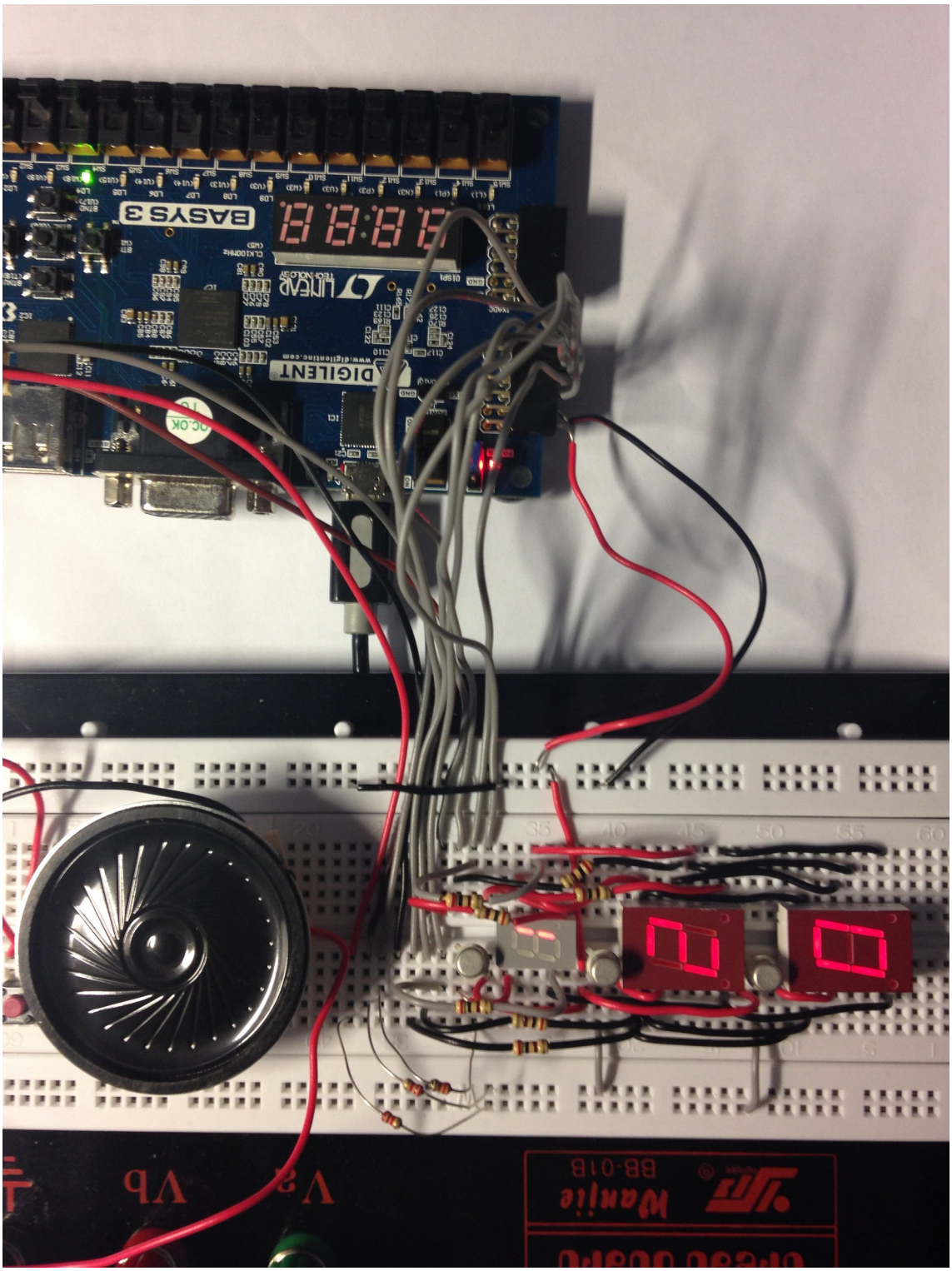


Transistor PNP 2N2907A : http://www.farnell.com/datasheets/1884693.pdf?_ga=2.55593698.890411580.1497259122-116627149.1496910925

Afficheur 7 segments : http://www.farnell.com/datasheets/2096651.pdf?_ga=2.256995138.890411580.1497259122-116627149.1496910925 et http://www.farnell.com/datasheets/2096651.pdf?_ga=2.188530284.575932278.1497740639-116627149.1496910925

La maquette

Une fois tous les schémas électroniques créés, la maquette a pu être réalisée, défini comme précédemment :



Conclusion

Le métronome a été réalisé pratiquement complètement. Il manque encore quelques fonctionnalités qui gèrent le type des mesures et d'autres extras. Cependant, il est complètement fonctionnel. Il affiche correctement les BPM sur les afficheurs et sur les LEDs. Le buzzer fonctionne correctement et le système complet réagi bien aux interractions avec l'utilisateur.
Etant, moi-même musicien, je m'en suis servi à plusieurs reprises et ma première conclusion est qu'il est très fonctionnel et rapide d'utilisation.

J'ai l'ambition d'améliorer le projet, optimiser le code, rajouter les fonctionnalités manquantes, tel qu'un port jack pour des écouteurs et réaliser un prototype complet : circuit imprimé, boîtier et alimentation.

Créer ce métronome a été pour moi un projet intéressant. A la fois dans la réflexion pour répondre au mieux aux besoins, mais aussi de la conception qui a été intéressante : rechercher les solutions techniques les plus favorables pour réaliser ce système.

Le seul inconvénient que je pourrais cité et qui est dû à moi-même. C'est d'avoir travaillé seul. Certes, cela m'a forcé à travailler chaque partie de manière approfondie, mais je n'ai donc pas eu l'occasion de faire un travail d'équipe et de travailler la répartition des tâches comme dans mes anciens projets.