PROJET CSI3 : 2012 / 2013

# RAPPORT VHDL : Cardiofréquencemètre de pouls


PERROT Gurvan / RAOULT Corentin

29 Avril 2013

# Sommaire

# Introduction

Avant tout, il est intéressant de savoir qu'un oxymètre de pouls est utilisé depuis des dizaines d'année dans le secteur médical et plus particulièrement par les urgentistes pour la suivie de blessés graves. Aujourd'hui, ce petit gadget est également utilisé par les sportifs afin de déterminer leurs limites. En effet, cet appareil permet de mesurer le taux d'oxygène dans le sang ainsi que d'afficher les pulsations du cœur en battements par minute. Pour prendre des mesures, il est nécessaire de mettre un doigt dans un capteur possédant une LED rouge et infrarouge (sorte de pince à linge).

Durant notre projet de CSI3, nous avons été amenés à réaliser ce type d'appareil. Pour cela, le projet a été divisé en 3 parties différentes : une partie électronique, une de VHDL et enfin une partie informatique. La partie qui va ici nous intéresser est la partie VHDL.

Le VHDL est un langage de description de matériel destiné à représenter le comportement ainsi que l'architecture d'un système électronique numérique. Afin de concevoir le système dans sa totalité, celui-ci a été divisé en plusieurs blocs rattaché entre eux dont nous allons expliquer le fonctionnement dans la partie développement.

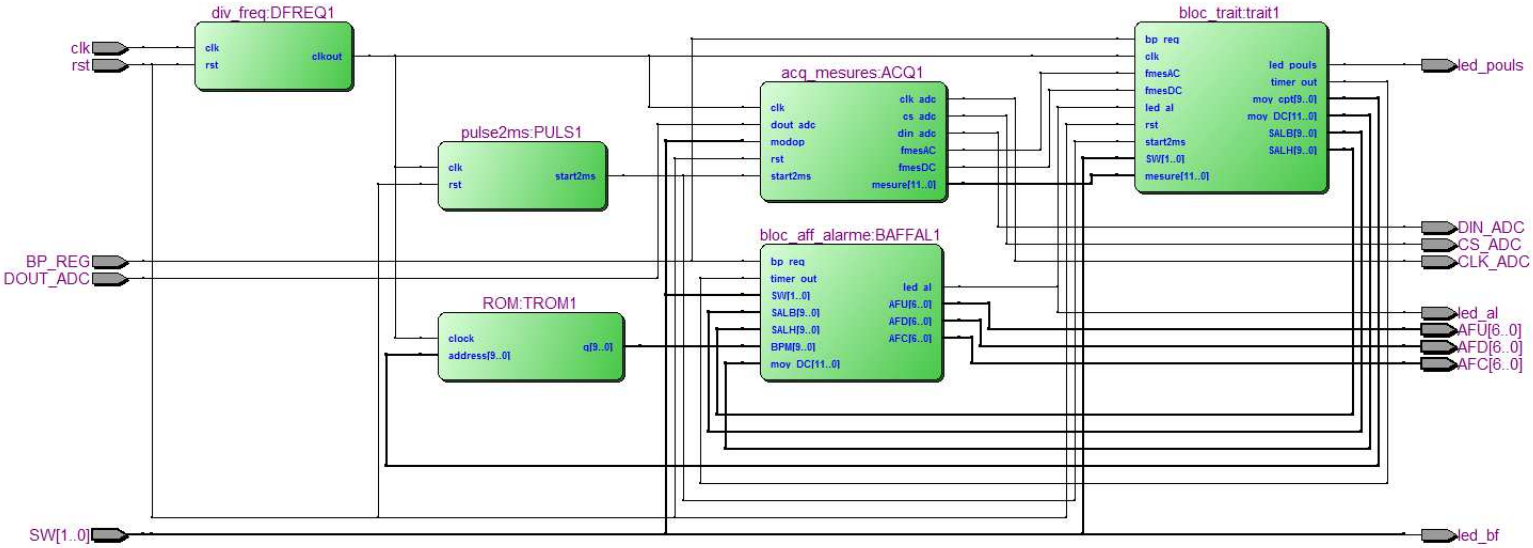
## Partie technique

Vue générale du système (top-level) :

Dans notre système nous avons le diviseur de fréquence qui permet de passer d'une horloge de 20 MHz à une horloge de base de 100 kHz, puis le bloc pulse2ms qui envoie une pulsation au système toutes les 2 ms. Ensuite, on a le bloc d'acquisition mesures qui permet comme son nom l'indique d'acquérir les mesures qui sortent du capteur. Après nous avons le bloc traitement qui est lui-même constitué de 4 blocs différents : le traitement AC, le traitement DC, la gestion des seuils d'alarme et la gestion de la LED pouls. Ce bloc permet de traiter le signal d'entrée. Ensuite, nous avons un bloc ROM qui permet de convertir de l'hexadécimal, en battement par minute. Enfin, nous avons le bloc affichage qui rend possible l'affichage de la pulsation du cœur en battement par minute grâce à un affichage des unités, des dizaines et des centaines.

Comme cela a été expliqué précédemment, ce schéma représente les différents blocs reliés entre eux par des signaux. On peut également y voir les entrées et les sorties du système total. Ici, on a en entrée : l'horloge de 20 MHz, le reset actif à zéro le bouton poussoir (BP_REG), le switch (SW) pour régler les seuils d'alarme et le DOUT_ADC qui produit la donnée sur 12 bits.

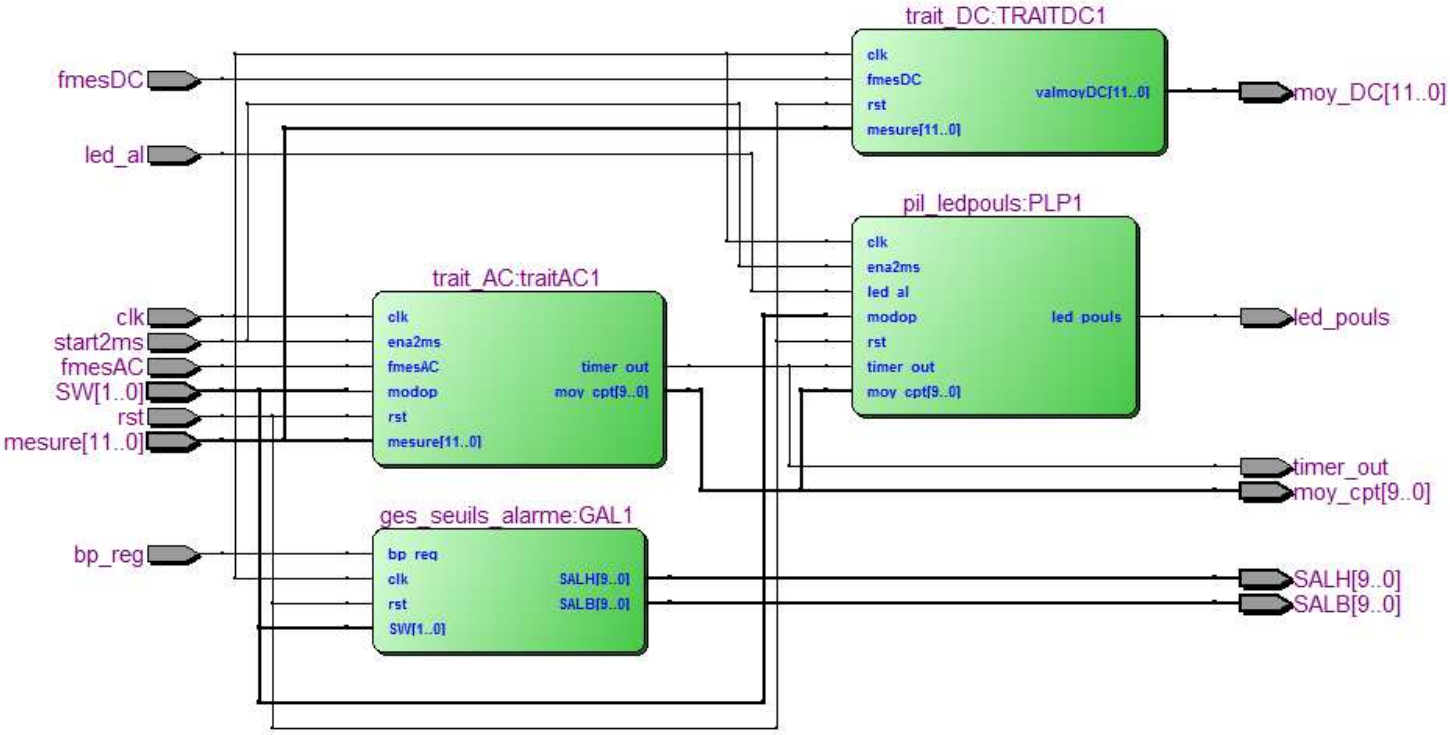
En sortie, on a la LED de bon fonctionnement (led_bf) qui s'allume en vert lors du lancement du système, la LED alarme (led_al) qui s'allume en rouge lorsque le domaine dans lequel le BPM doit être compris est dépassé et la LED pouls (led_pouls) qui clignote en orange au rythme du BPM, plus celui-ci est grand, plus le clignotement s'accélère. Ensuite, nous avons le signal d'activation (chip Select) CS_ADC, le signal d'horloge sur son entrée (CLK_ADC) et le mot de contrôle série sur son entrée DIN_ADC). Pour finir les afficheurs pour les unités, les dizaines et les centaines sont naturellement présent (AFU, AFD, AFC).



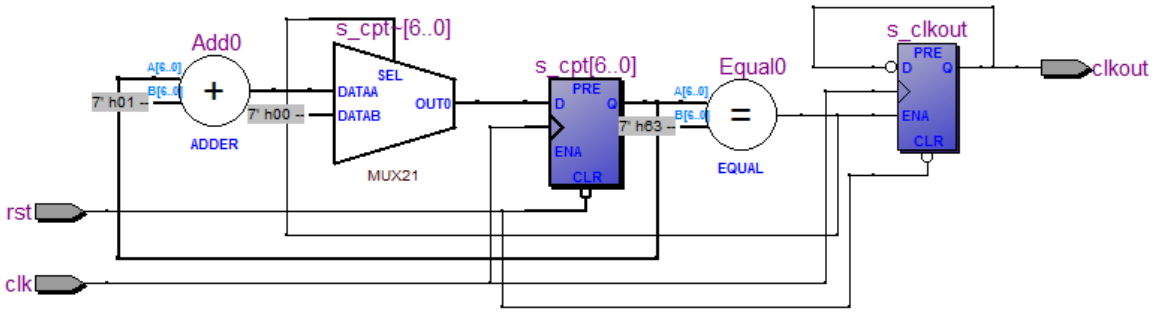

Ce schéma nous permet de visualiser les 4 blocs contenu dans le bloc traitement situé dans le schéma général vu précédemment.

Concernant les entrées de ce système, celui-ci possède naturellement une horloge et un reset. De plus, on peut y voir comme précédemment le bouton poussoir ainsi que le switch qui permet de modifier les seuils des alarmes et la LED alarme. Nous avons un start2ms, qui lors d'une impulsion permet le traitement du signal AC puis du DC. On y retrouve également un signal de fin de message pour AC et pour DC afin de savoir quand le traitement de chacun de ces signaux est terminé. De plus, le bus d'entrée « mesure » de 12 bits venant du bloc acquisition arrive dans le bloc de traitement AC.

Pour ce qui est des sorties, celui-ci sort la moyenne DC (moy_DC): moyenne réalisé sur les 8 dernières mesures de DC et  la moyenne du compteur (moy_cpt) qui correspond à la même chose pour le signal AC. On y retrouve la LED pouls présent en sortie du système précédent ainsi qu'un timer_out qui permet de « sortir » du système lorsqu'il n'y a plus de doigt dans le capteur ou lorsque celui-ci bouge trop. Cela permet donc d'éviter de sortir des valeurs erronées. Enfin, on peut y voir les seuils d'alarme haut et les seuils d'alarme bas (SALH et SALB) en sortie du bloc de gestion des alarmes, leur configuration se fera ensuite grâce à leur affichage sur la carte.



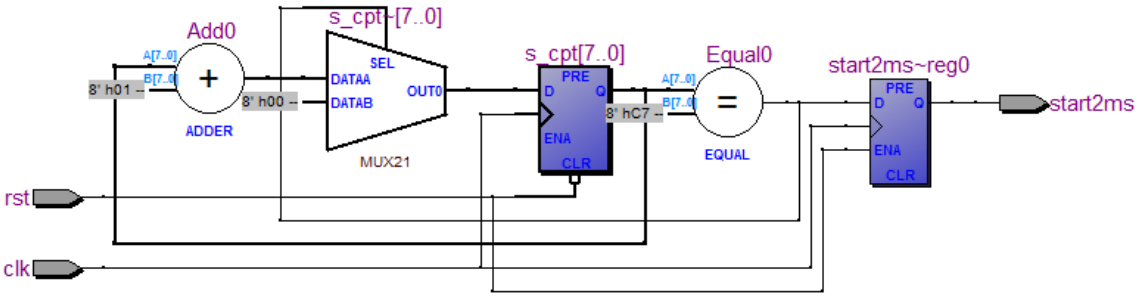

Bloc diviseur de fréquence :




Pour que notre système soit fonctionnel, les 24 fronts d'horloge doivent être contenus dans le 2 ms. On obtient ainsi une fréquence minimum de [illegible] = [illegible] = 50 [illegible]. De plus, la datasheet de l'ADC nous précise que sa fréquence maximale d'utilisation est de 200 kHz. On prendra donc une horloge de base de 100 kHz (valeur se situant entre la valeur maximale et la valeur minimale).

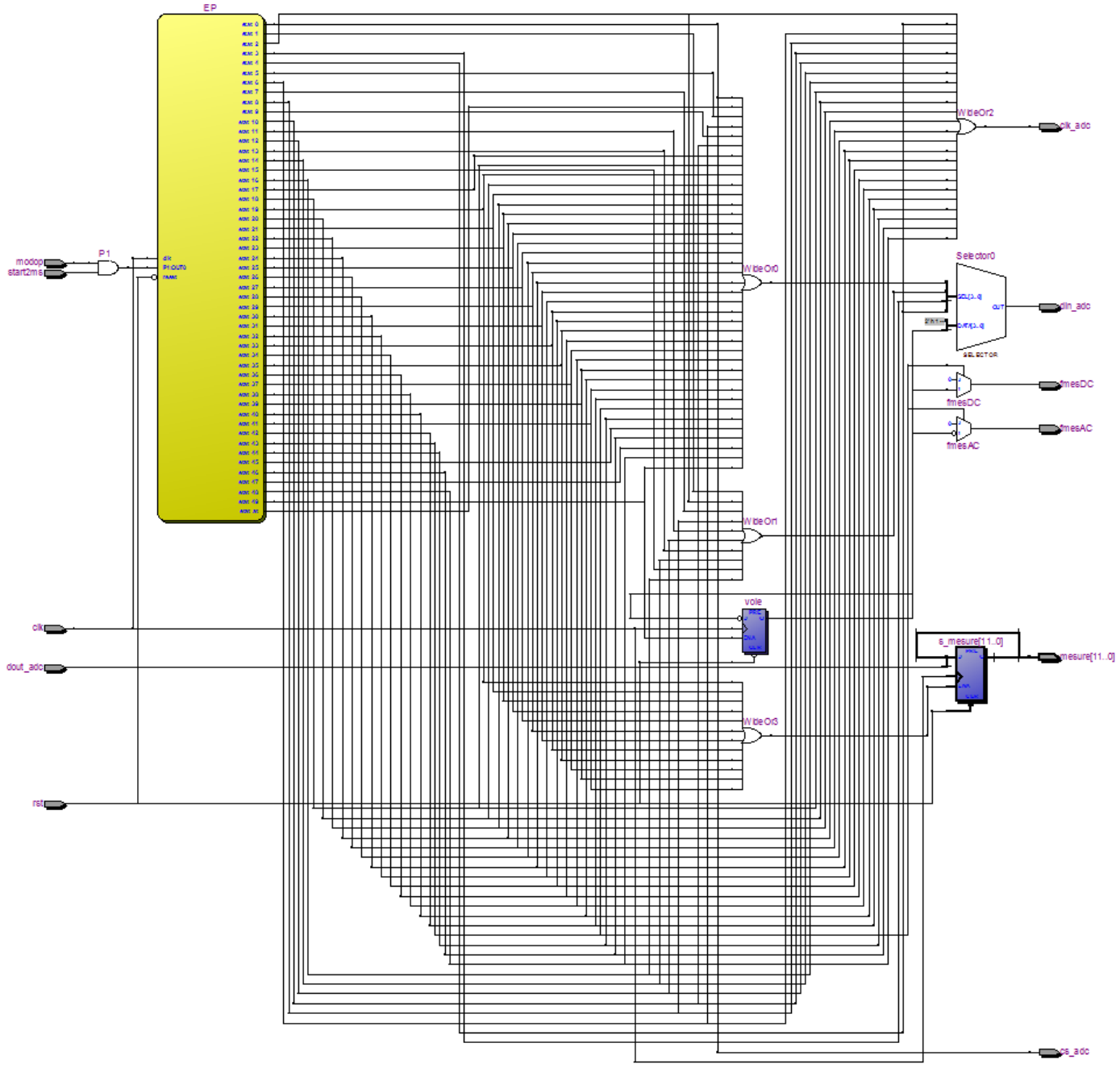
Ce bloc permet de passer d'une horloge de 20 MHz à une horloge de base de 100 kHz qui va ensuite être utilisé pour tout le système. Cette valeur d'horloge a été choisie car elle correspond à la fréquence d'utilisation du bloc traitement AC. De plus, afin d'obtenir une telle fréquence, il est nécessaire d'avoir un compte qui aille jusqu'à 99. En effet, pour passer de 20 MHz à 100 kHz, on a un rapport de 200 que l'on divise ensuite par 2, soit, on obtient 100. Le compte allant de 0 à 99, celui-ci fonctionne bien.

Bloc pulse2ms :



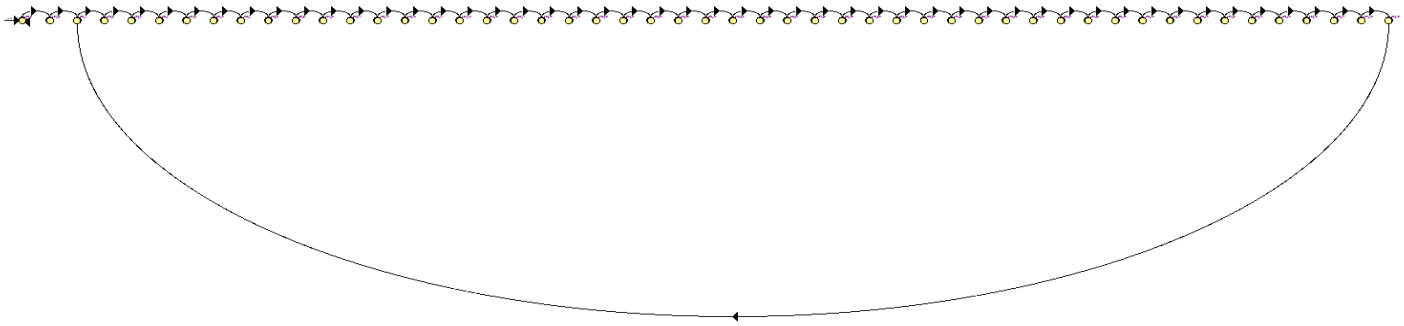

Concernant ce bloc, celui-ci doit, toutes les 2 ms, provoquer une impulsion de durée égale à la période de l'horloge de base à 100KHz. Pour ce faire un compteur allant de zéro jusqu'à 199 a été créé. En effet, on a 200 fronts d'horloge toutes les 2 ms. Le compteur s'incrémente à partir de zéro est avec la sortie start2ms valant zéro, arrivé à 199 la sortie start2ms est mise à 1 et le compteur est remis à zéro, puis on recommence un nouveau cycle de compteur et start2ms est remis à zéro, de cette façon start2ms ne dure qu'une période d'horloge de 100KHz.

Bloc acquisition mesures:

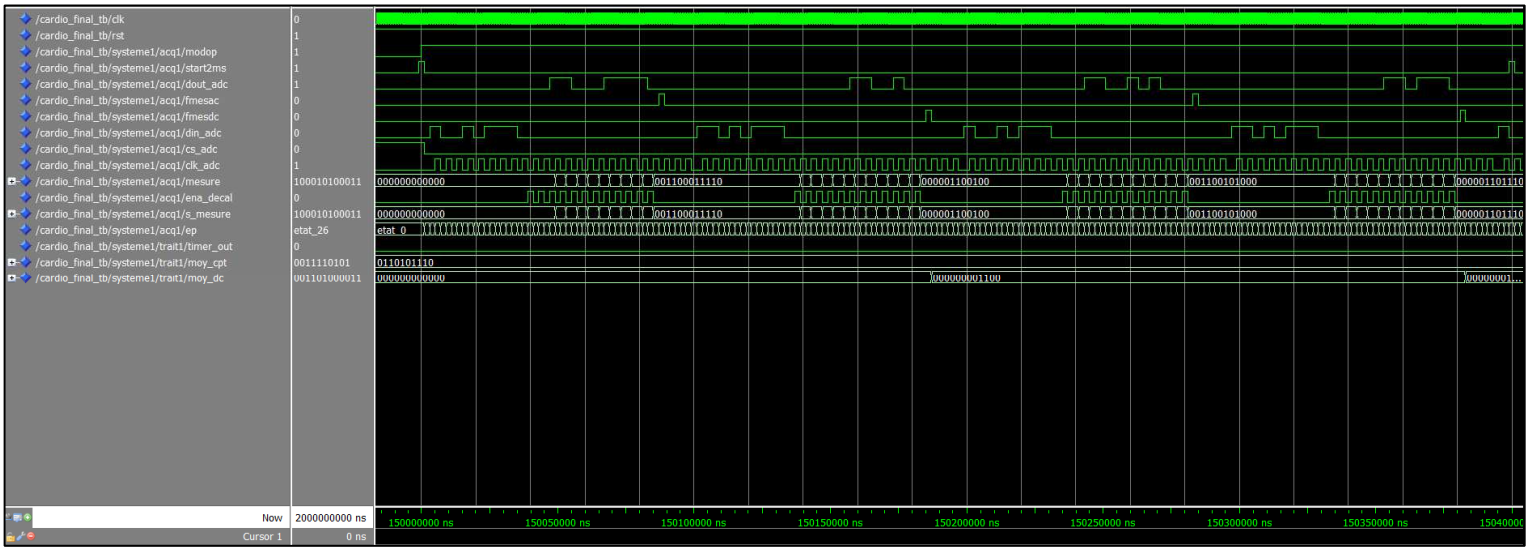



Ce bloc est constitué de beaucoup d'état. Cependant pour diminuer le nombre d'états, une variable temporelle (voie) a été créée pour permettre le traitement de la voie AC ou de la voie DC. Le traitement de ces 2 voies prend 2 ms. En fonction du signal traité, les mots d'entrées vont être différents (10010111 pour la voie AC et 11010111 pour la voie DC), certains bits ne seront évidemment pas les mêmes et la « fmesAc » provoquera le passage à la voie DC.

Machine d'états pour le bloc d'acquisition des mesures :



Chronogramme :

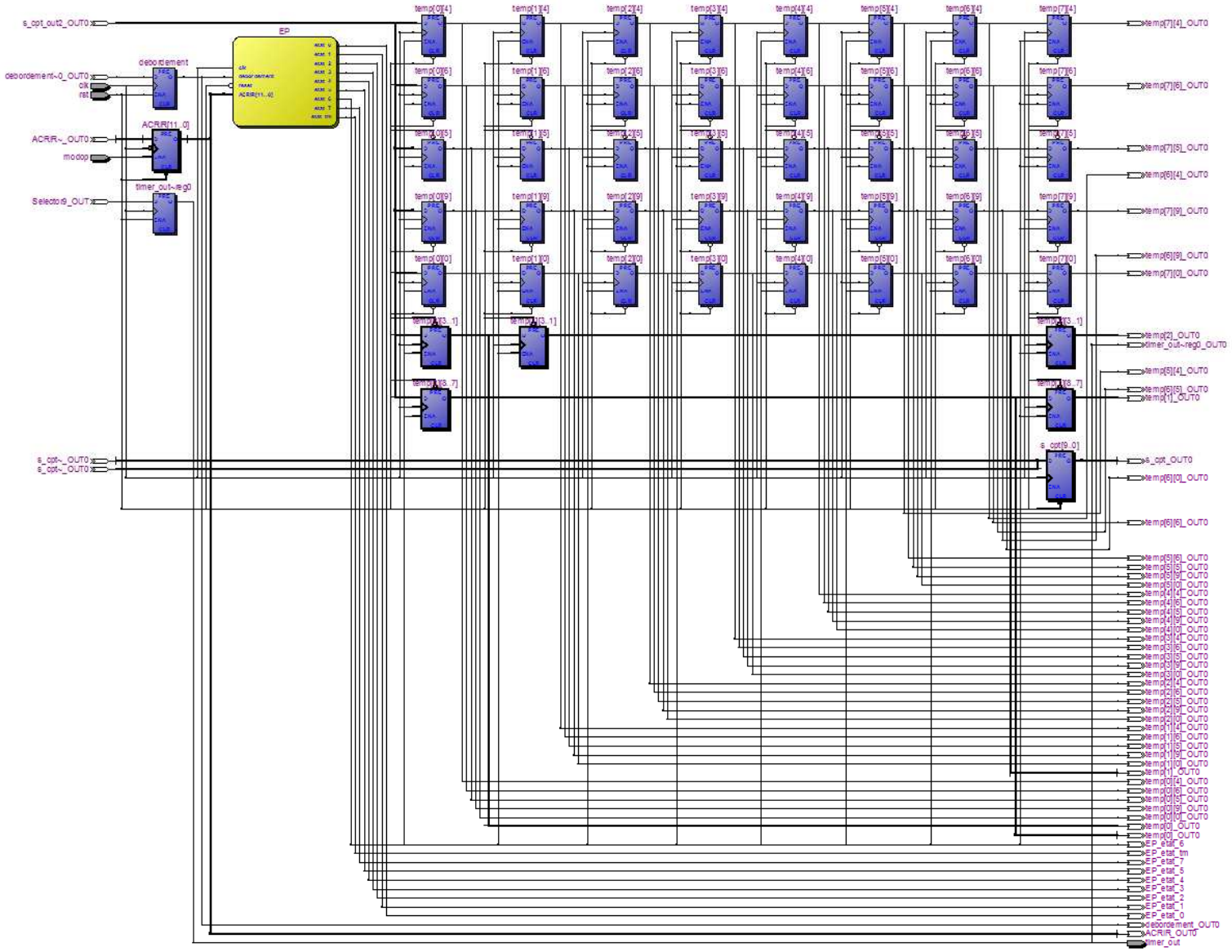


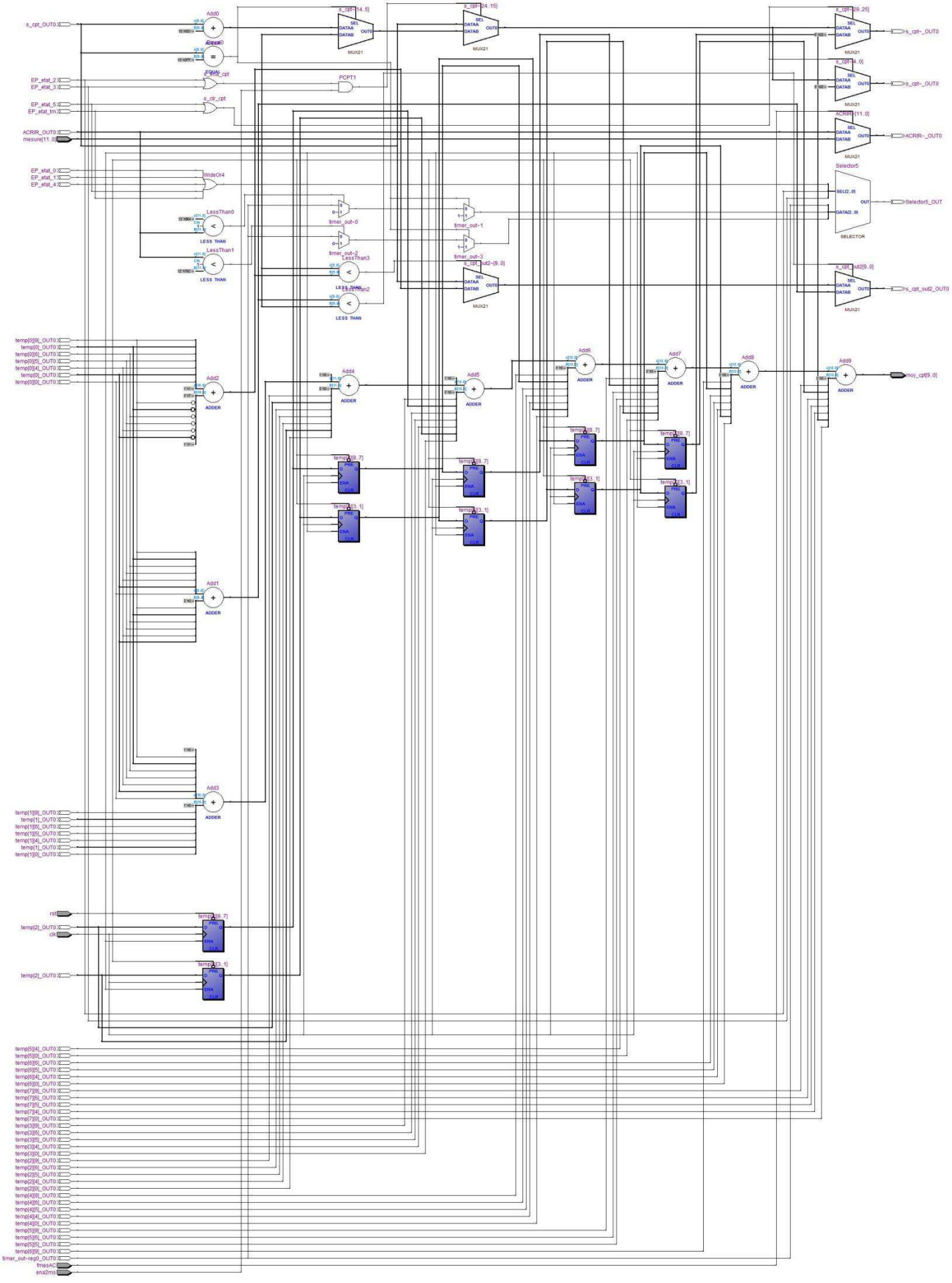

Au final, la création du bloc d'acquisition des mesures s'est en partie réalisée à l'aide du chronogramme que l'on avait réalisé dans le guide d'étude. Le chronogramme correspond à la simulation faite au-dessus sur Modelsim. On peut un voir le signal d'horloge, le reset actif à l'état haut ('1'), le modop actif également à '1' et le start2ms qui permet le démarrage de l'acquisition des mesures. Chaque fin de traitement que ce soit pour la partie AC ou la partie DC entraine un signal de fin de traitement (fmesAC et fmesDC). Pour ce qui est des états, à chaque nouveau front d'horloge (clk_adc) correspond un nouvel état, ainsi grâce à la variable temporelle, le nombre d'état étant diminué de moitié, il est désormais de 50. Ces 50 états peuvent être différents en fonction que ce soit le signal AC ou le DC traité. De plus, il faut savoir qu'avant d'avoir une mesure de prise, la moyenne sur 8 échantillons doit être faite pour le DC.

Ce bloc est également pourvu d'un registre à décalage afin de pouvoir prendre en compte une nouvelle valeur quand celle-ci arrive. Une valeur laisse donc place à une nouvelle valeur (la nouvelle valeur arrive par la droite pendant qu'une ancienne valeur ressort par la gauche). Tout ceci se fait naturellement sur 12 bits, nombre de bits pris par la mesure.

Bloc traitement des mesures :

Bloc traitement d'ACRIR :

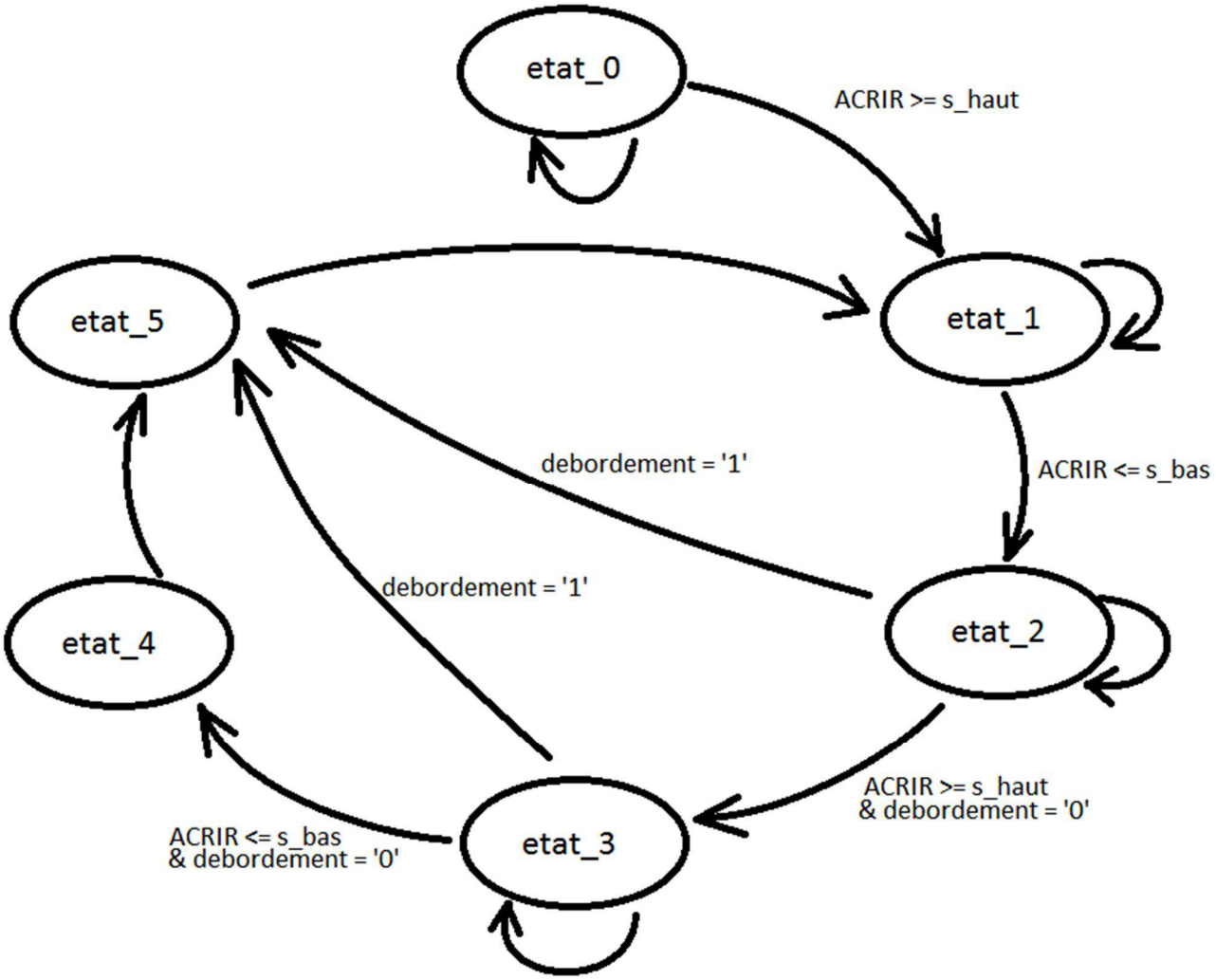



Add0
Add1
Add2
Add3
Add4
Add5
Add6
Add7
Add8
Add9
ADDER
LessThan0
LessThan1
LessThan2
LessThan3
LESS THAN
WideOr4
PCPT1
MUX21
Selector5
SELECTOR
timer_out-0
timer_out-1
timer_out-2
timer_out-3
s_cpt_out2~[9..0]
s_cpt~[14..5]
s_cpt~[24..15]
s_cpt~[29..25]
s_cpt~_OUT0
ACRIR~[11..0]
ACRIR~_OUT0
s_cpt_out2[9..0]
s_cpt_out2_OUT0
Selector5_OUT
mesure[11..0]
ACRIR_OUT0
EP_etat_0
EP_etat_1
EP_etat_2
EP_etat_3
EP_etat_4
EP_etat_5
EP_etat_tm
s_clr_cpt
rst
clk
fmesAC
ena2ms
timer_out-reg0_OUT0
moy_cpt[9..0]

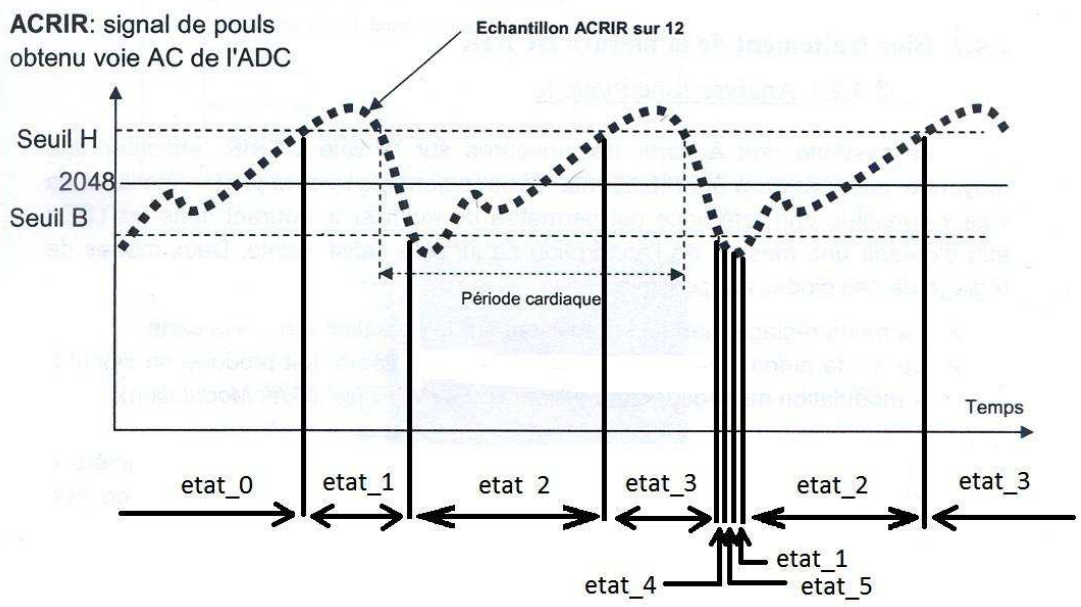
Ce bloc permet de sélectionner le signal arrivant par la voie mesure pour ne prendre que le signal ACRIR (grâce à fmesAC), puis  le mettre en parallèle avec un registre à décalage de 12 bits. Ensuite selon les variations du signal ACRIR et avec une machine de Moore (expliquée ci-dessous), on compte le nombre d'échantillons de 2ms que le bloc reçoit pendant une période cardiaque.  Cette valeur est mémorisée dans un registre de mémorisation qui contient les 8 dernières valeurs du compteur, ces 8 valeurs sont additionnées entre elles puis le résultat est divisé par 8 pour obtenir en fin de compte la moyenne des valeurs contenues dans le registre de mémorisation.

Un système d'écrêtage a aussi été inclus pour limiter les variations entre de valeurs du compteur. Une variable delta est créé et contient la valeur de la sortie du compteur précédente divisée par 8 (décalage de 3 bits), ensuite si la valeur à mémoriser est supérieur à la valeur limite haute qui est la valeur précédente additionnée à delta, alors on remplace la valeur à mémoriser par la valeur limite haute. Idem, si la valeur à mémoriser est inférieur à la valeur limite basse qui est la valeur précédente à laquelle on soustrait delta, alors on remplace la valeur à mémoriser par la valeur limite basse. A noter qu'on aurait dû utiliser cette méthode pour limiter les variations de l'ACRIR.

Machine d'états du traitement AC :



Courbe montrant le passage aux différents états en fonction d'ACRIR :



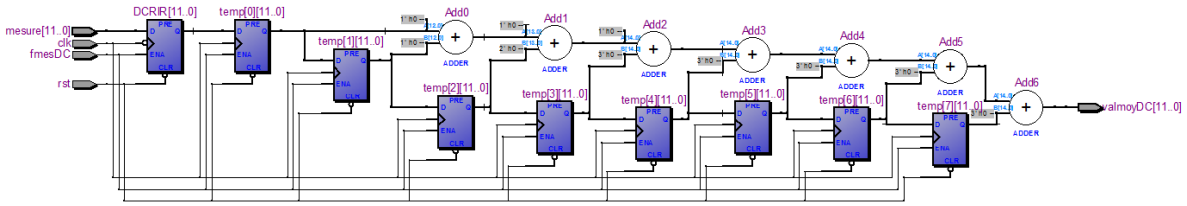

Les sorties de cette machine de Moore sont s_ena_cpt qui permet au compteur de compter, s_clr_cpt qui remet le compteur à zéro et s_ena_reg qui permet au registre de mémorisation d'intégrer la valeur actuelle du compteur et supprime la plus ancienne des 8 valeurs contenues dans sa mémoire.

Lors du reset la machine se met à etat_0 où les trois sorties sont inactives, ensuite si ACRIR est supérieur à s_haut (qui vaut 2148) alors on passe à etat_1, sinon on reste dans etat_0. Dans l'état_1, les sorties sont aussi toutes inactives, si ACRIR et inférieur à s_bas (qui vaut 1948) alors on passe dans l'état_2, sinon on reste dans l'etat_1. Dan l'etat_2 on commence à compter (s_ena_cpt passe à 1 mais les autres sorties restes inactives), si ACRIR devient supérieur à s_haut alors on passe dans l'etat_3 sinon on reste dans l'etat_2. A l'etat_3 on continue de compter (les sorties sont inchangées), si ACRIR devient inférieur à s_bas alors on passe dans l'etat_4 sinon on reste dans l'etat_3.

L'etat_4 est un état de mémorisation, le compteur est arrêté, et on mémorise sa valeur actuelle (s_ena_cpt est mis à zéro, s_ena_reg passe à un et s_clr_cpt reste à zéro), il n'y a pas de conditions pour sortir de cet état donc on passe au front d'horloge suivant directement à l'etat_5. L'etat_5 est l'état de remise à zéro du compteur, le compteur est mis à zéro et on arrête la mémorisation dans le registre, cet état est lui aussi sans conditions de sortie, on passe donc directement au front d'horloge suivant à l'etat_1. A l'etat_1 comme ACRIR est inférieur à s_bas comme on le voit sur la courbe ci-dessus, on passe directement à l'etat_2 après un front d'horloge et un nouveau cycle recommence.

Un débordement de compteur intervient quand le compteur arrive à sa valeur maximale qui est 1023, cela veut dire qu'il n'y a pas de doigt dans le capteur (ACRIR est supérieur à s-haut). Cet événement ne peut se produire que dans l'etat_2 ou l'etat_3 car on ne reste que quelques microsecondes dans etat_1, etat_4 et etat_5. Lors d'un débordement timer_out est mis à 1 et on va directement à l'etat_5 sans passer par etat_4 pour remettre à zéro le compteur sans enregistrer sa valeur actuelle puis on va directement à l'etat_1 (où le compteur est inactif) et on attend qu'ACRIR redevienne inférieur à s_bas. S'il y a un débordement dans l'etat_0 lors celui-ci passe à l'etat_1 car quand il n'y a pas de doigt ACRIR est supérieur à s_haut. Enfin quand on remet un doigt dans le capteur, ACRIR repasse en dessous de s_bas, alors on passe dans l'etat_2 et timer_out est remis à zéro.

Bloc traitement de DCRIR :




Ce bloc permet de sélectionner le signal arrivant par la voie mesure pour ne prendre que le signal DCRIR (grâce à fmesDC), puis de le mettre en parallèle avec un registre à décalage de 12 bits. Ensuite cette valeur est stockée dans un banc de registre à base de bascules D. Celui-ci contient les 8 dernières valeurs de DCRIR qui seront ensuite additionnées entre elles puis divisées par 8 pour en faire la moyenne tout comme les valeurs du compteur dans le bloc trait_AC.

Bloc gestion des seuils d'alarme :

Le fonctionnement de ce bloc reste simple, en fonction du switch choisi, soit 00 ou 01, on modifie respectivement le seuil d'alarme bas et le seuil d'alarme haut. Concernant le seuil d'alarme bas, celui-ci est modifié par pas de 2 jusqu'à 60 BPM puis il retrouve sa valeur initial. Pour le seuil d'alarme haut, celui-ci est décrémenté par pas de 10 jusqu'à une valeur de 100 pour naturellement retrouver sa valeur initiale.

Machine d'états pour la gestion des seuils d'alarme :

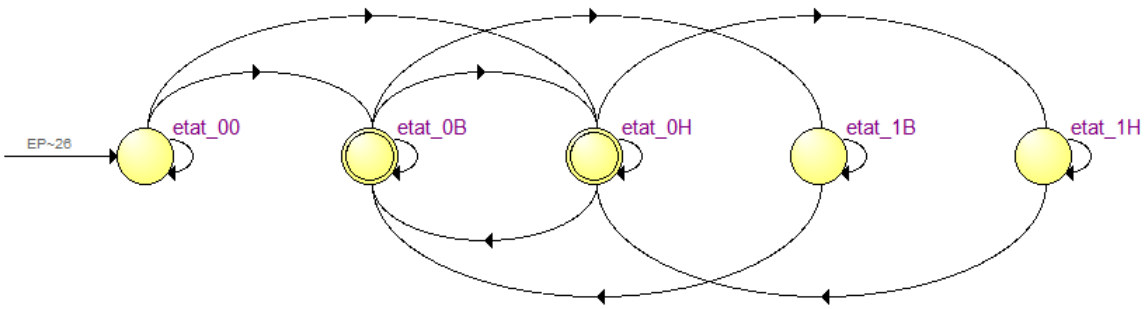
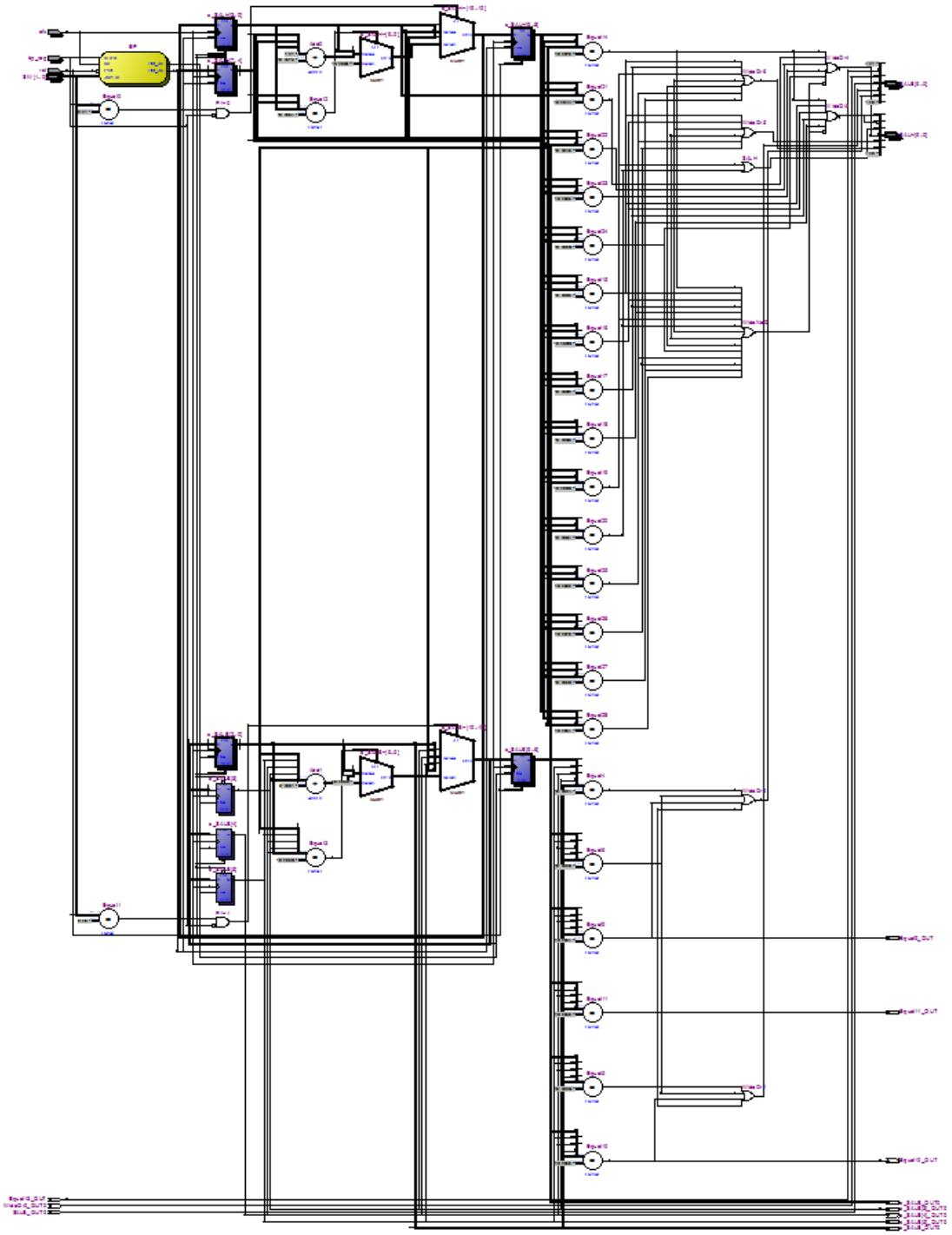


Tableau de transition correspondant à cette machine :

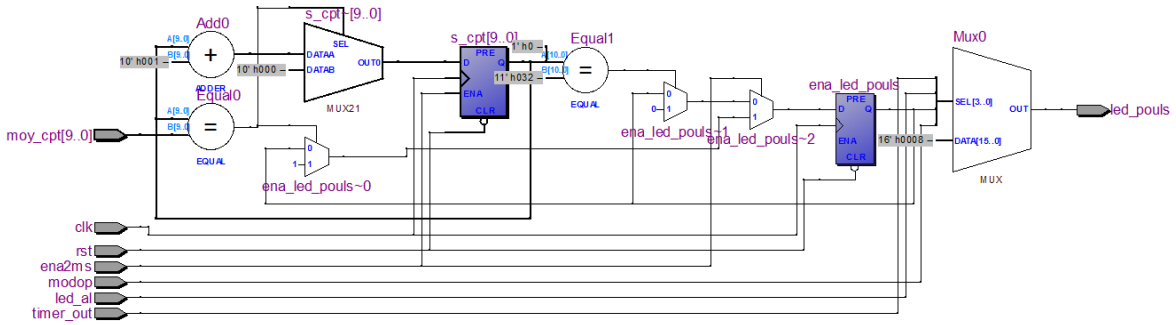
| | Source State | Destination State | Condition |
|---|---|---|---|
| 1 | etat_0B | etat_0B | (!bp_reg).(SW[1]) + (bp_reg).(!SW[0]) + (bp_reg).(SW[0]).(SW[1]) |
| 2 | etat_0B | etat_0H | (SW[0]).(!SW[1]) |
| 3 | etat_0B | etat_1B | (!bp_reg).(!SW[0]).(!SW[1]) |
| 4 | etat_0H | etat_0B | (!SW[0]).(!SW[1]) |
| 5 | etat_0H | etat_0H | (!SW[0]).(SW[1]) + (SW[0]).(!SW[1]).(bp_reg) + (SW[0]).(SW[1]) |
| 6 | etat_0H | etat_1H | (SW[0]).(!SW[1]).(!bp_reg) |
| 7 | etat_00 | etat_0B | (!SW[0]).(!SW[1]) |
| 8 | etat_00 | etat_0H | (SW[0]).(!SW[1]) |
| 9 | etat_00 | etat_00 | (SW[1]) |
| 10 | etat_1B | etat_0B | (bp_reg) |
| 11 | etat_1B | etat_1B | (!bp_reg) |
| 12 | etat_1H | etat_0H | (bp_reg) |
| 13 | etat_1H | etat_1H | (!bp_reg) |

A partir de cette machine d'état, on peut visualiser les différents cas contenu dans le bloc de gestion des alarmes. On a un état_00 (état d'attente après un reset) qui passe à état_0H si le switch = « 01 » ou à état_0H si switch = « 00 ». Ensuite si on est dans état_0H (respectivement état_0B), que l'a un appuie sur le bouton poussoir (bp_reg = '0') et que le switch = « 01 » (respectivement switch = « 00 ») alors on passe à état_1H (respectivement état_0H), si le switch passe à « 00 » (respectivement « 01 ») alors on va à état_0B (respectivement état_0H). A l'état état_1H on décrémente de 10 une seule fois SALH et on attend que le bouton poussoir sot relâcher pour repasser à état_0H. Idem, à l'état état_1B incrémente SALB de 2 une seule fois et on attend que le bouton poussoir soit relâché. Si SALB est supérieur à 60 alors il revient à 40, de la même façon si SALH est inférieur à 100 alors il revient à 240.

Les valeurs obtenues à la sortie de l'automate sont en binaire, il faut donc les convertir en hexadécimal pour que dans le bloc affichage elles soient comparable au BPM qui sortent de la ROM en hexadécimal. Pour cela des multiplexeurs sont utilisé (une ROM aurait aussi pût être utilisée mais au vue du peu de valeurs à convertir cette technique a été préférée). Il suffit alors de faire correspondre à chaque nombre en binaire une valeur en hexadécimal.



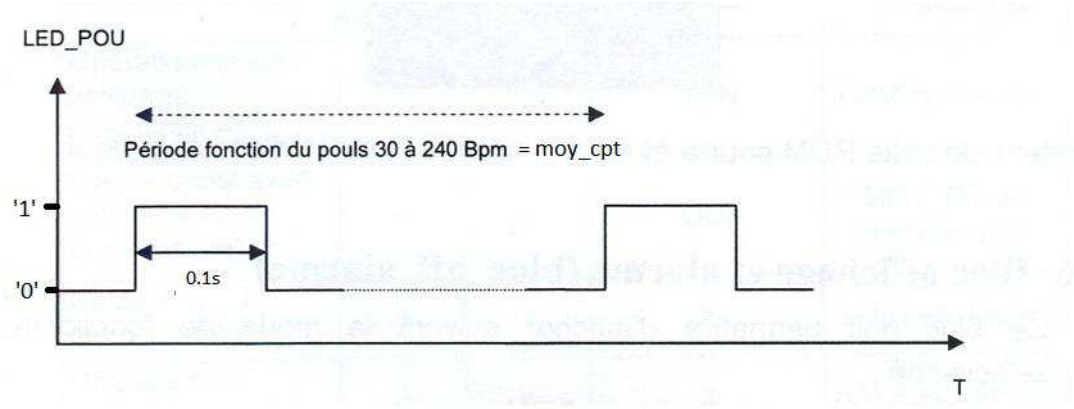
Bloc gestion led_pouls :




Ce bloc permet la gestion de la led_pouls. Plus le rythme cardiaque sera élevé, plus la fréquence d'allumage de la LED pouls sera faible. En opposition, plus le battement du cœur sera lent, plus la fréquence d'allumage de la LED sera grande.

Ce bloc fonctionne de la façon suivante : un compteur allant de 0 à moy_cpt s'incrémente toute les 2 ms, de 0 à 50 il met le signal pour allumer la LED actif, ce qui correspond à 0.1s (50 x 2ms) ou la LED sera allumée, ensuite arrivé à 50, il met le signal d'allumage de la LED inactif. Quand le compteur arrive à la valeur moy_cpt qui correspond à la période cardiaque actuelle, il est remis à zéro et le signal d'allumage de la LED est remis à 1 et on recommence un nouveau cycle.

Le multiplexeur à la fin sert à savoir si on est en mode opérationnel, en effet si led_al est allumée, timer-out est actif ou si le modop est à zéro alors la LED reste éteinte même si le signal d'allumage de la LED est actif.




<u>Bloc ROM</u>:

Il faut savoir que la ROM est placée juste avant le bloc affichage. Celle-ci à des valeurs qui rentre en binaire (sortie du bloc traitement) et à pour sortie des valeurs de battements par minute en décimal.

Justification de la taille des bus de la ROM :

On a la moyenne du compteur qui arrive en entrée de la ROM. Or on mesure au minimum 30 battements par minutes car le domaine de la fréquence cardiaque est compris entre 30 et 255 BPM. D'où 30 BPM correspond à 0,5Hz c'est-à-dire à une période de 2s. Le taux d'échantillonnage étant de 2ms, on a au maximum $\frac{?}{????^{?}?} = 1000$. Or pour code 1000 en binaire, seulement 10 bits sont nécessaire. La valeur maximale du compteur étant codage sur 10 bits, la ROM ne nécessite donc qu'un bus de 10 bits en entrée.

Pour fabriquer la ROM, il a avant tout été nécessaire de créer un tableau sous EXCEL faisant correspondre à chaque case un nombre de battement par minute. Pour ce faire une formule a été utilisée : $\frac{60}{0{,}002 \times n}$ avec n l'indice de la case. Nous avons au numérateur 60 pour convertir les secondes en minutes et au dénominateur 0,002 car une nouvelle mesure arrive toutes les 2 ms (diviser par le nombre de mesure sur une période de temps déterminée). De plus, pour remplir le tableau, la valeur arrondie du résultat de cette formule est choisie. Ainsi nous obtenons en parti :

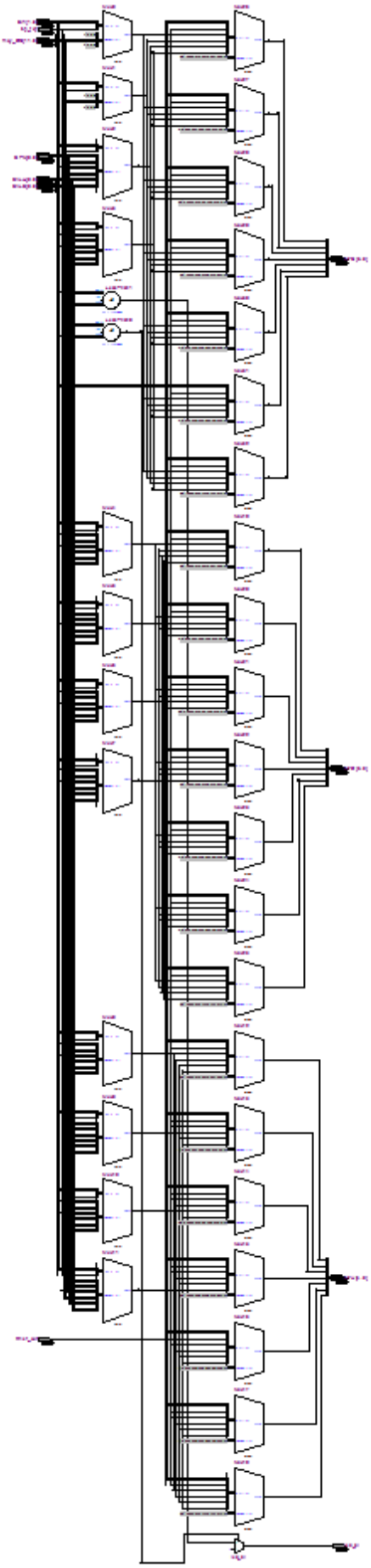
| Cases mémoire (hexadécimal) | BPM (décimal) |
|---:|---:|
| 117 | 0 |
| 118 | 254 |
| 119 | 252 |
| 120 | 250 |
| 121 | 248 |
| 122 | 246 |
| 123 | 244 |
| 124 | 242 |
| 125 | 240 |
| 126 | 238 |
| 127 | 236 |
| 128 | 234 |
| 129 | 233 |
| 130 | 231 |
| 131 | 229 |
| 132 | 227 |
| 133 | 226 |
| 134 | 224 |
| 135 | 222 |
| 136 | 221 |
| 137 | 219 |
| 138 | 217 |
| 139 | 216 |
| 140 | 214 |
| 141 | 213 |
| 142 | 211 |
| 143 | 210 |
| 144 | 208 |
| 145 | 207 |

Comme nous pouvons le voir, les cases mémoires sont en hexadécimal alors que la correspondance des battements par minute se faire en décimal.

Cependant, en réalité, ce tableau fait 1024 cases pour répondre à tous les cas possibles. En effet, la ROM prend en entrée 10 bits (1111111111 = 1023) donc pour prendre en compte toutes les possibilités 1024 cases sont nécessaires : une valeur doit correspond pour chacun des bits, c'est-à-dire pour toutes les possibilités. De plus, le domaine de définition des battements par minute étant compris entre 255 BPM et 30 BPM, tous les nombres n'étant pas compris dans cet encadrement sont remplacé par un zéro.

Suite à cela, la ROM peut enfin être créé à partir du logiciel Quartus en copiant simplement le tableau passé au préalable à l'horizontal d'EXCEL sous Quartus. Cette ROM prend bien en entrée 10 bits codé sur 1024 cases mémoire (words memory en anglais). On fait attention à ce que la sortie ne soit pas reliée sur l'entrée et on fait bien correspondre le fichier .hex au fichier ROM.vhd. Ce bloc permet donc aux afficheurs de retourner le nombre de battement par minute et sa valeur en hexadécimal lors de l'appui sur le bouton poussoir.

Bloc affichage :



Ce bloc est principalement constitué des multiplexeurs car plusieurs signaux de sélections vont être pris en compte. Il en découle l'affichage des unités, des dizaines et des centaines. Ce bloc permet d'afficher suivant le mode de fonctionnement choisi, c'est-à-dire suivant la position du switch et du bouton poussoir soit la fréquence cardiaque en BPM, soit la valeur moyenne DC en hexadécimal. C'est également grâce à cet afficheur que le seuil d'alarme haut et bas peut être modifié.

Dans un premier temps on compare le BPM arrivant de la ROM avec SALB et SALH arrivants du bloc traitement, si le BPM est supérieur à SALH ou inférieur à SALB alors on allume la led_al sinon cette dernière reste éteinte.

Ensuite une concaténation du switch et du bouton poussoir est faite pour créer le signal de sélection du multiplexeur qui correspond au tableau ci-dessous pour décider ce que l'on va afficher.

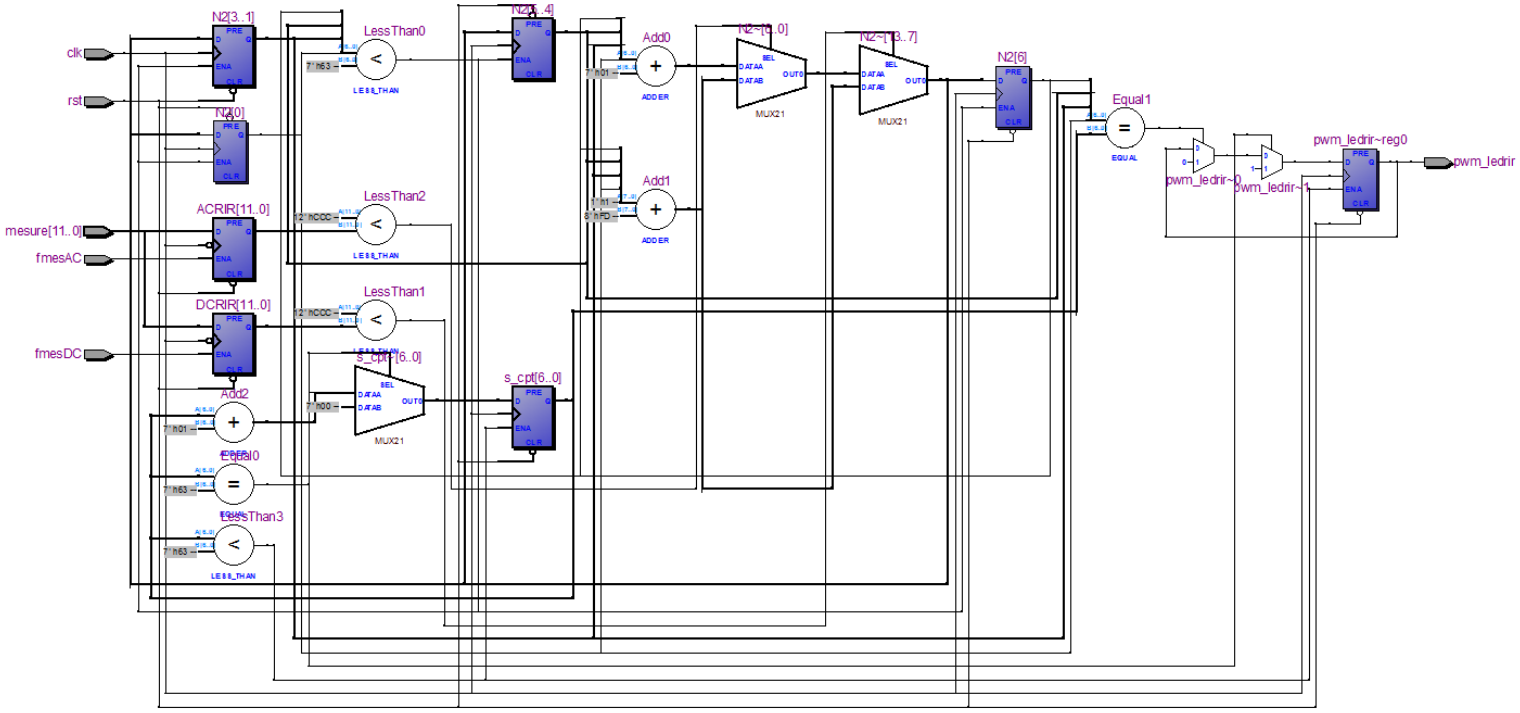
- Affichages souhaités

| mode | SW(1) | SW(0) | Appui BP_REG | affichage |
|---|---|---|---|---|
| Opérationnel avec calibrage manuel | 1 | 1 | NON | BPM en décimal |
| Opérationnel avec calibrage manuel | 1 | 1 | OUI | MOY_DC en hexadécimal |
| Opérationnel avec calibrage automatique* | 1 | 0 | NON | BPM en décimal |
| Opérationnel avec calibrage automatique* | 1 | 0 | OUI | MOY_DC en hexadécimal |
| Réglage seuil alarme haute | 0 | 1 | X | SALH en décimal |
| Réglage seuil alarme basse | 0 | 0 | X | SALB en décimal |
| Débordement compteur | 1 | X | X | - - - |

* option

Le signal sortant de ce multiplexeur (qui est en hexadécimal) est divisé en trois signaux de 4 bits pour obtenir 3 valeurs en binaire, puis chacune de ces trois valeurs est concaténée avec le switch1 et le timer_out. Ensuite ces signaux sont mis en entrée de multiplexeur qui sortiront la valeur en afficheur 7 segments correspondante si le switch_1 est égal à zéro (affichage des seuils d'alarmes) ou si switch_1 est égal à un et timer_out est égal à zéro (affichage de moy_dc ou BPM). Sinon le signal de sortie sera celui pour afficher un trait.

Bloc asservissement :




Ce bloc sert à régler automatiquement l'intensité du courant passant dans les LEDs en envoyant un signal PWM à 100kHz dont le rapport cyclique augmentera ou diminuera en fonction des mesures qu'il reçoit. Ce signal PWM sera ensuite moyenner par un filtre passe-bas. Ce bloc extrait lui-même l'AC et le DC de mesure avec fmesAC et fmesDC pour pouvoir ensuite faire les calculs. L'horloge de base de ce bloc doit être plus rapide que 100kHz ont prend donc celle de 20MHz, en effet il nous faut un compteur pour pouvoir faire varier le rapport cyclique. Ici le compteur va de 0 à 99 car il y a 100 fronts d'horloges de 20MHz dans une période de 100kHz. Après un reset, la valeur par défaut de la durée où le signal PWM de sorti sera à '1' est de 49 (rapport cyclique de 50%). Si ACRIR ou DCRIR est supérieur à 3276 (80% de la dynamique maximale qui est de 4096) alors on décrémente la durée pendant laquelle le signal PWM sera à l'état haut, sinon par défaut ont l'incrémente.

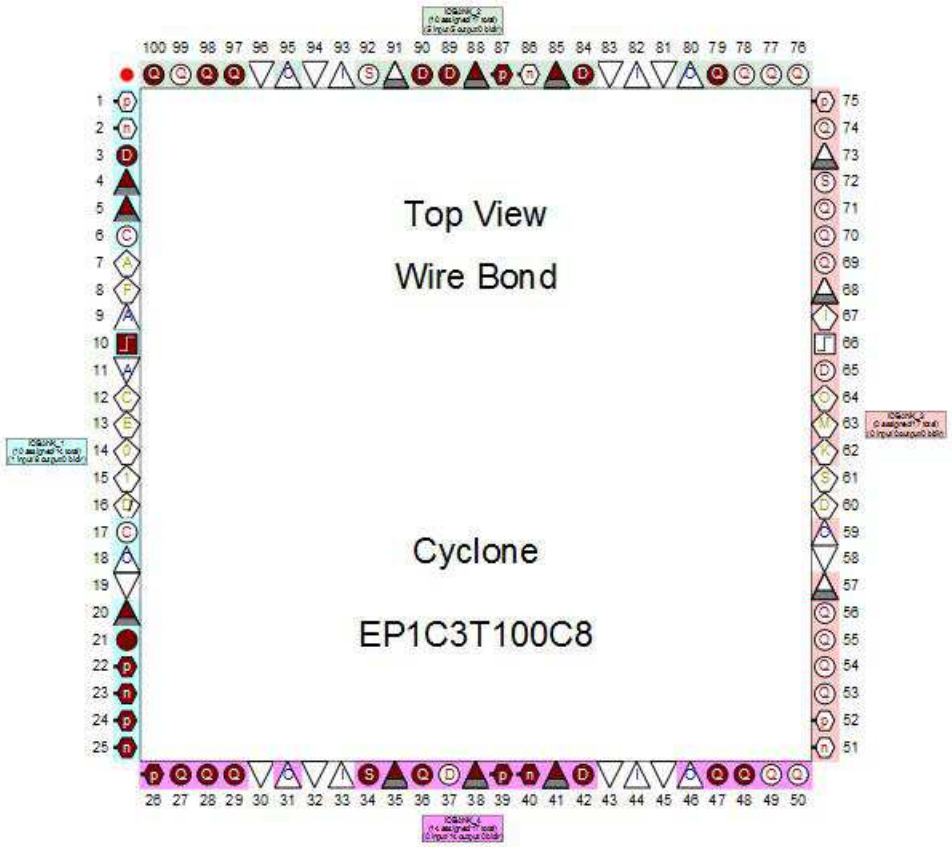
Compilation sur Quartus

Afin de réaliser une simulation au plus proche de ce qui se passera ensuite sur la carte, un bloc ADC est positionné au début du système afin de relier celui-ci aux entrées et sorties du système total.

Une fois la simulation de tous les blocs entre eux effectués, le passage sous Quartus est nécessaire afin de corriger les erreurs et les warnings restant. De plus, un fichier .sdc de contraintes temporelles doit être créé à partir du TimeQuest. Cet outil permet d'analyser tous les chemins de registre à registre et de vérifier si les contraintes temporelles sont respectées. Pour résumer, le TimeQuest vérifie que notre système peut fonctionner à la fréquence de 100kHz.

Ensuite afin d'éliminer totalement tous les « critical warning », l'affectation des pins doit être réalisé à partir du pin planner. Suite à cela l'affectation d'un chemin d'horloge à un signal peut être faite. Cela consiste à aller dans « l'assignment editor » et de sélectionner notre horloge de base clkout pour relier tous les blocs du système à cette horloge.

Finalement, avant le test sur carte le type de FPGA utilisé doit être choisi : ici, on a un EP1C3T100C8.




```
#Projet cardio-frequencemetre
#fichier de contraintes temporelles esploite par l'analyseur de timing de Quartus : TimeQuest
#horloge de nom clk de periode 50 ns (freq=20MHz) entree du systeme numerique
create_clock -name clk -period 50.000 [get_ports {clk}]

#horloge genere par le diviseur de frequence
#frequence de l'horloge generee: clkout est divisee par 200 par rapport a la frequence clk, rapport cyclique de cette horloge: 50%
create_generated_clock -name clkout -source [get_ports {clk}] -divide_by 200 -duty_cycle 50 [get_nets {DFREQ1|s_clkout}]
```

# Glossaire

ACRIR : valeur échantillonnée de la lumière mesurée par le capteur, elle reflète le signal cardiaque.

Automate : machine à états finis

BPM: Battement Par Minute

BP_REG : bouton poussoir

DCRIR : valeur moyenne de la lumière reçue par le capteur, elle dépend de l'absorption des tissus.

FPGA: Field Programmable Gate Array

Modelsim: Outil de simulation

LED: Light Emitted Diode

Quartus: Outil de synthèse logique et de placement routage d'un circuit programmable

ROM:  Read Only Memory

TimeQuest (Quartus): Outil d'analyse de timing statique

VHDL: Very high speed (or scale) integrated circuits Hardware Description Language

# Avis personnel

Ce projet nous a beaucoup apporté car on a ainsi pût voir concrètement ce que nous pourrions être amené à faire dans un futur proche. Ce projet nous permis de  voir sur quoi un ingénieur pouvait être amené à travailler. Par ailleurs, j'ai également découvert que la fabrication d'un circuit intégré n'est pas aussi simple que l'on peut l'imaginer. Il y a un gros travail d'analyse à faire au préalable avant même de commencer à faire la schématique. Pour finir, le faite de faire l'intégration sur la carte réalisée parallèlement en électronique a été très intéressante car cela représentait l'aboutissement de notre projet. On a ainsi, après avoir vérifié la bonne alimentation des composants par la tension qui leur était nécessaire, pût voir l'aboutissement concret de notre projet en visualisant sur l'oscilloscope et sur nos afficheurs nos battements de cœur par minute suite à l'insertion de notre doigt dans le capteur.

# Conclusion

Durant la phase d'intégration, après avoir vérifié que tous les composants étaient bien alimenté par la tension qui leur est nécessaire, nous avons pût finaliser notre projet en combinant la partie VHDL avec la carte faite en parallèle durant la partie électronique. Il a simplement suffi d'implanté la partie VHDL à l'intérieur du FPGA et de mettre notre doigt dans le capteur possédant une LED rouge et infra-rouge et notre pulsation pas minute a pût être lu grâce aux afficheurs 7 segments. Ce projet nous a permet de nos mettre à la place des ingénieurs qui conçoivent les appareils électroniques de demain et de rencontrer des difficultés auxquelles nous serions plus tard confronté.

Grâce à ce projet, nous avons pût voir ce à quoi plus tard nous serions amené à faire. En effet, de nos jours, n'importe quelle entreprise possède des projets à plus ou moins long terme afin que celle-ci puisse sortir de nouvelle innovation et attirer la clientèle. Pour ce qui est du projet de la NASA, c'est de « capturer un astéroïde à l'aide d'une sorte de sac. Tout projet rend une entreprise vivante et lui permet de se démarquer d'une autre

# Annexe

Nous pouvons voir en annexe les codes sources VHDL de chaque bloc réalisé sous Modelsim (fichiers .vhd).