Bordeaux INP – ENSEIRB-MATMECA

Filière Systèmes Électroniques Embarqués

PR209 – Projet expérimental de conception de circuit numérique (S8)

Module assuré par Christophe JEGO

Rapport d'activité

Par

Maxime Dabrowski

Remis le 02 avril 2021

Répertoire GIT : https://github.com/Neopsis-SO/PR209-Lecteur_MP3

SOMMAIRE

1. Cahier des charges

Ce projet a pour but de designer et d'implémenter un lecteur MP3 sur une carte FPGA NEXYS A7.

Il sera décomposé en 4 parties :

- Une interface utilisateur du lecteur de fichier mp3 à l'aide de la carte FPGA NEXYS A7 ;
- Une architecture de lecture, de modulation de type PWM et de transmission sur la sortie audio (jack) de la carte FPGA NEXYS A7 ;
- Une liaison UART permettant le transfert de fichier MP3 d'un ordinateur vers la mémoire de la carte FPGA NEXYS A7 ;
- Une association de tous ces blocs pour créer notre lecteur MP3 sur la carte FPGA NEXYS A7.

En plus de la fonction de lecteur MP3 et du retour de l'interface utilisateur, une production personnelle permettra d'effectuer la lecture plus rapide ou plus lente du fichier audio.

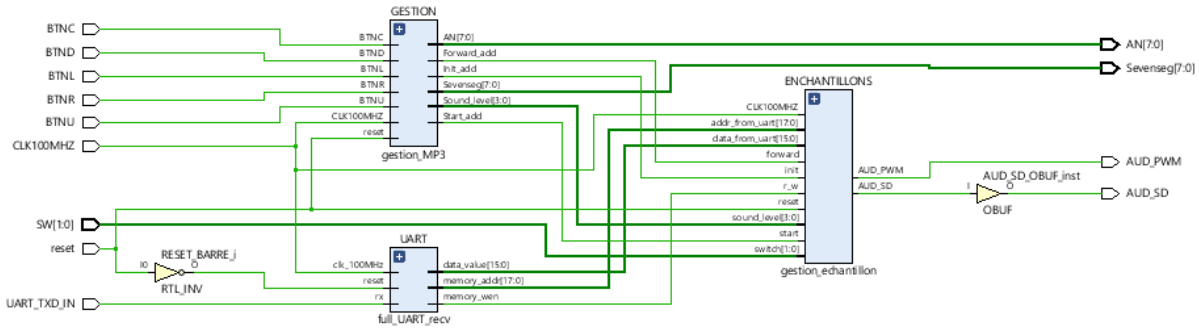


Figure 1 : Schéma RTL de l'architecture attendu du MP3

Voici les blocs visibles sur ce schéma :

- [GESTION] : correspond à l'interface utilisateur qui gère la FSM, le contrôle du volume, du sens de lecture et l'affichage des informations sur les 8 afficheurs 7 segments de la cible ;
- [ECHANTILLONS] : correspond à l'architecture de lecture, de modulation, de stockage du fichier audio. Comprend également la production personnelle ;
- [UART] : permet la liaison série avec l'ordinateur pour stocker sur la carte le fichier MP3 en binaire.

2. Module d'affichage et de contrôle

Comme précisé dans le cahier des charges, ce projet se découpe en 3 sous parties. Ici je présenterai la partie concernant le contrôle des états du lecteur MP3 ainsi que l'affichage sur les afficheurs 7 segments.

Voici tout d'abord un schéma RTL des blocs :

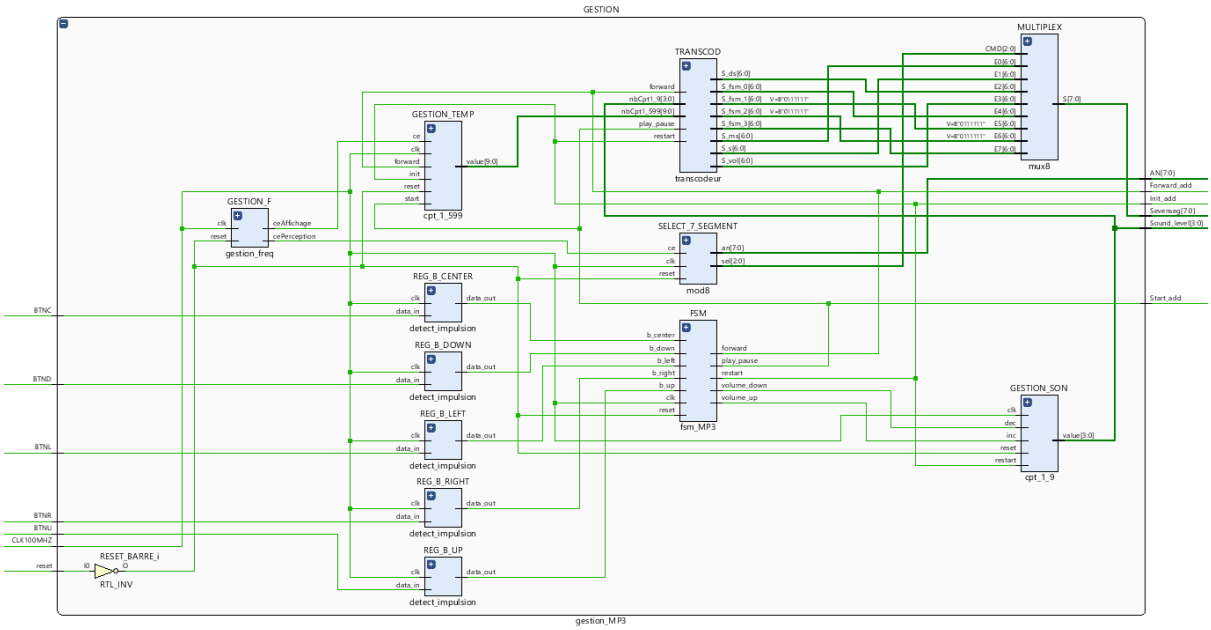


Figure 2 : Schéma RTL du module de « Gestion » du MP3

Nous avons sur cette partie 12 blocs avec une conception de 8 blocs :

- [GESTION_F] : un synthétiseur de fréquence permettant de générer plusieurs « clock enable » ;
- [REG_B_…] : un détecteur d'impulsion permettant de simuler un front montant lors de l'appui sur un des boutons ;
- [GESTION_TEMP] : un compteur allant de 1 à 599 pour compter le nombre de milliseconde qui s'écoule ;
- [GESTION_SON] : un compteur allant de 1 à 9 permettant d'ajuster le niveau sonore en sortie ;
- [FSM] : une machine à état permettant de contrôler les différentes phases de fonctionnement ;
- [TRANSCOD] : un transcodeur prenant les différentes informations de cette partie pour les remettre en forme afin de les afficher sur les 7 segments ;
- [SELECT_7_SEGMENT] : un sélecteur permettant de sélectionner un des 8 afficheurs 7 segments ;
- [MULTIPLEX] : un multiplexeur pour sélectionner la donnée à afficher sur les 7 segments.

2.1. Détecteur d'impulsion

Voici tout d'abord le schéma RTL de ce bloc :

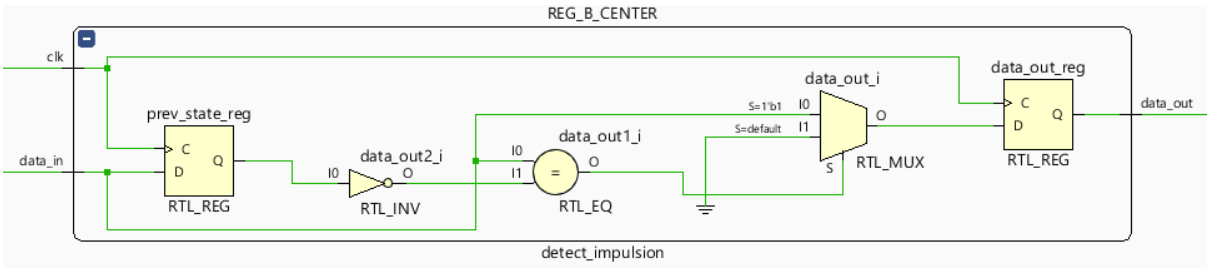


Figure 3 : Schéma RTL du « détecteur d'impulsion »

Ce bloc a pour but de détecter le front montant d'un des boutons poussoir. La bascule d'entrée sauvegarde la donnée lors du front montant de l'horloge. Si la valeur de cette bascule est différente de l'entrée, alors nous appliquons l'inverse de cette valeur sur la sortie du bloc, sinon nous appliquons un 0 logique. Cela permet de n'avoir la sortie à 1 que sur une période d'horloge. Cela permet de simplifier la FSM puisque sans ces blocs, nous devrions ajouter des étapes intermédiaires pour simuler la détection de front montant ou descendant.

2.2. Gestion des fréquences

Voici tout d'abord le schéma RTL de ce bloc :

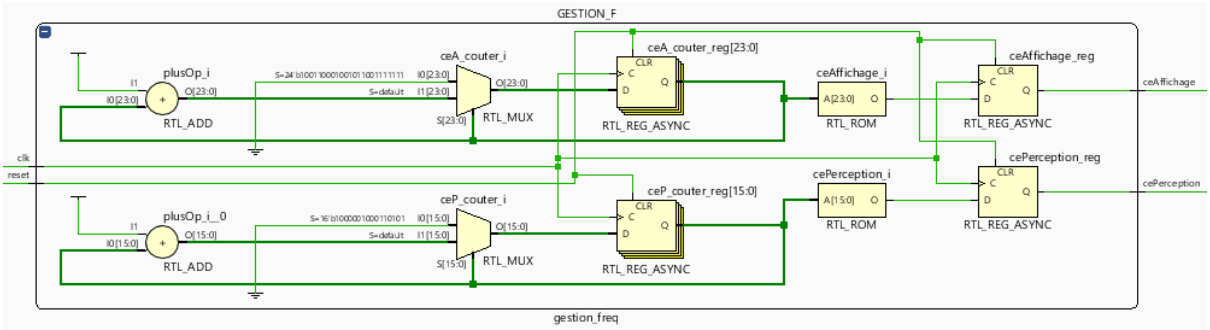


Figure 4 : Schéma RTL de la gestion de fréquence

Notre carte cible fonctionne à une fréquence de 100MHz. Ce bloc génère des « clock enable » qui permettent de baisser fictivement la fréquence de fonctionnement de certains blocs. Ici, le bloc comptera le nombre de front de période d'horloge écoulé en partant de 0 jusqu'à atteindre un nombre permettant de générer une autre fréquence. Nous avons donc ici 2 « clock enable » permettant de baisser la vitesse à :

- CePerception : 3kHz qui sera utilisé pour la persistance rétienne des afficheurs 7 segments ;
- CeAffichage : 10Hz qui permettra d'avoir une précision à la dixième de seconde pour l'écoulement du temps de la musique.

Nous avons donc des compteurs allant jusqu'à :

- CePerception : 100 000 000 / 3 000 – 1 = 33 334 ;
- CeAffichage : 100 000 000 / 10 – 1 = 9 999 999

2.3. Compteur temporel

Voici tout d'abord le schéma RTL de ce bloc :

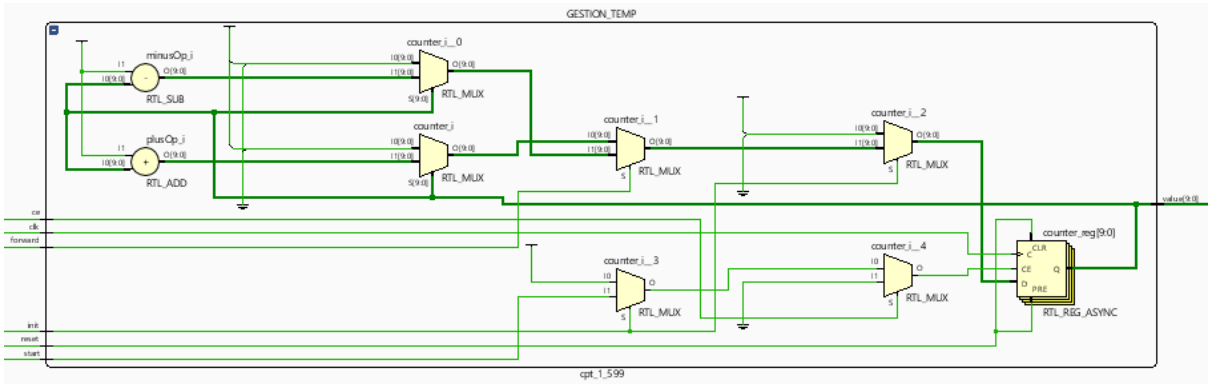


Figure 5 : Schéma RTL du « compteur de temps »

Ce compteur est un compteur permettant d'aller de 1 à 599 pour compter le temps qui s'écoule, soit 60 secondes dans notre cas. Lorsqu'il arrive à sa valeur minimale, le compteur recommence à 599 et il recommence à 1 s'il arrive à sa valeur maximale en fonction du sens de comptage. Ce compteur est cadencé par l'horloge et par le CE affichage fonctionnant à 10Hz. Il fonctionne également selon certaines entrées :

- « forward » indique le sens de comptage ;
- « start » indique que le compteur doit compter à la fréquence de 10Hz dans le sens indiqué par « forward » ;
- « restart » indique au compteur qu'il doit se réinitialiser à 1 et rester dans cette configuration.

Ces entrées viennent toutes de la machine à état décrite plus loin dans ce rapport.

La sortie de ce compteur est codée sur 10 bits en « STD_LOGIC_VECTOR » pour coder la valeur maximale du compteur soit 599.

2.4. Compteur pour le volume sonore

Voici tout d'abord le schéma RTL de ce bloc :

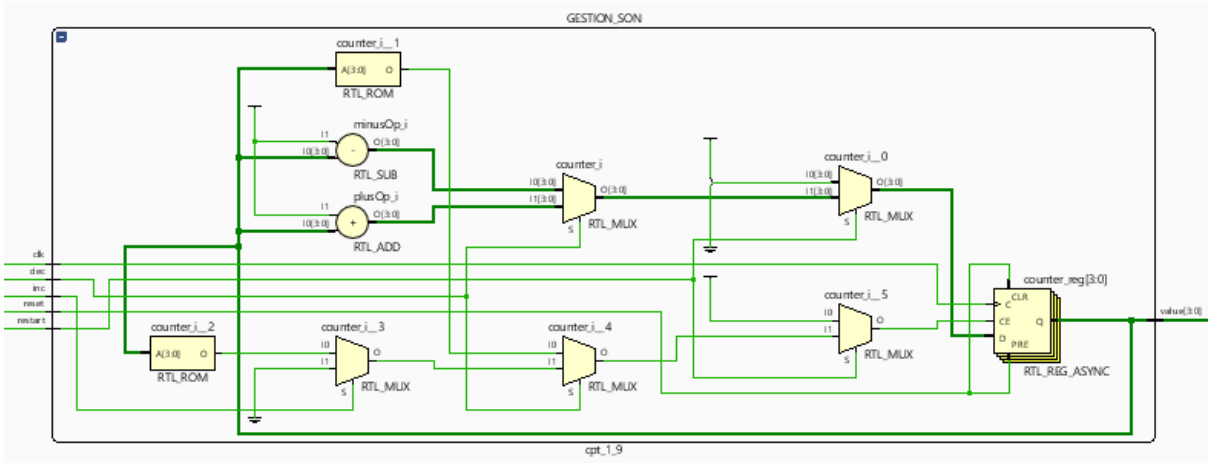


Figure 6 : Schéma RTL du « compteur pour le volume sonore »

Ce compteur est un compteur permettant d'aller de 1 à 9. Ce bloc est cadencé à la vitesse de l'horloge mais fonctionne selon les entrées :

- « inc » pour incrémenter la valeur de 1 ;
- « dec » pour décrémenter la valeur de 1 ;
- « restart » indique au compteur qu'il doit se réinitialiser à 1 et rester dans cette configuration.

L'entrée « restart » vient de la machine à état. Les 2 autres entrées (« inc » et « dec ») viennent de la machine à état. Les boutons poussoir doivent forcément passer par ces blocs car le compteur fonctionnant à 100MHz, il est impossible pour un humain de n'appuyer sur un bouton poussoir que 1ns.

La sortie de ce compteur est codée sur 4 bits en « STD_LOGIC_VECTOR » pour coder la valeur maximale du compteur soit 9.

2.5. Machine à état fini

Voici tout d'abord le schéma RTL de ce bloc :

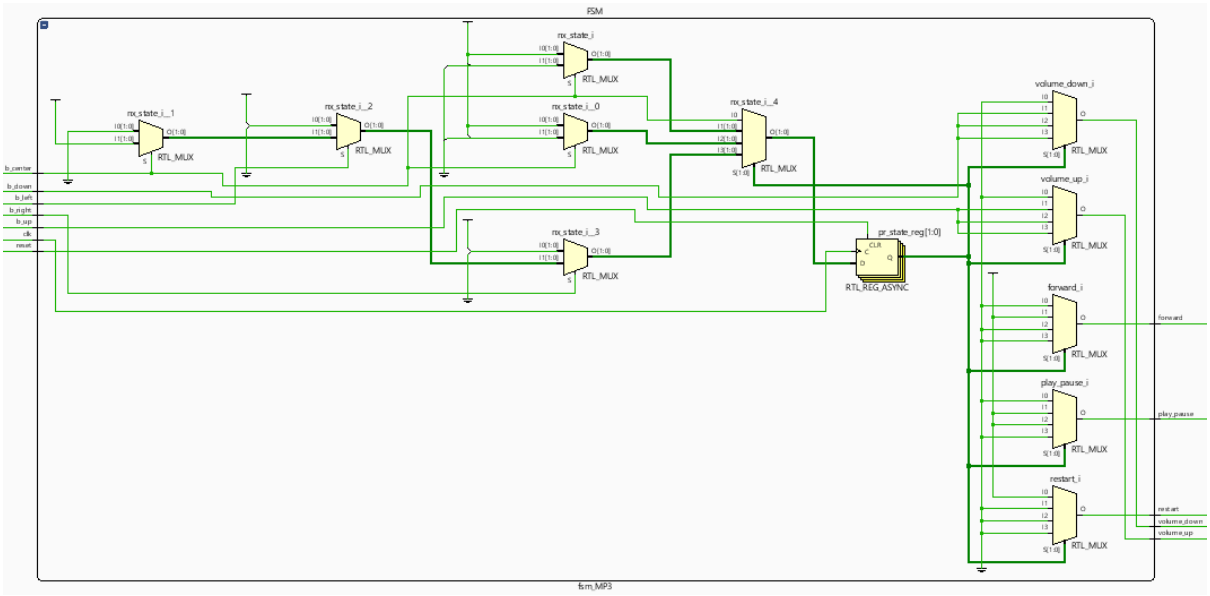


Figure 7 : Schéma RTL de la « Machine à état »

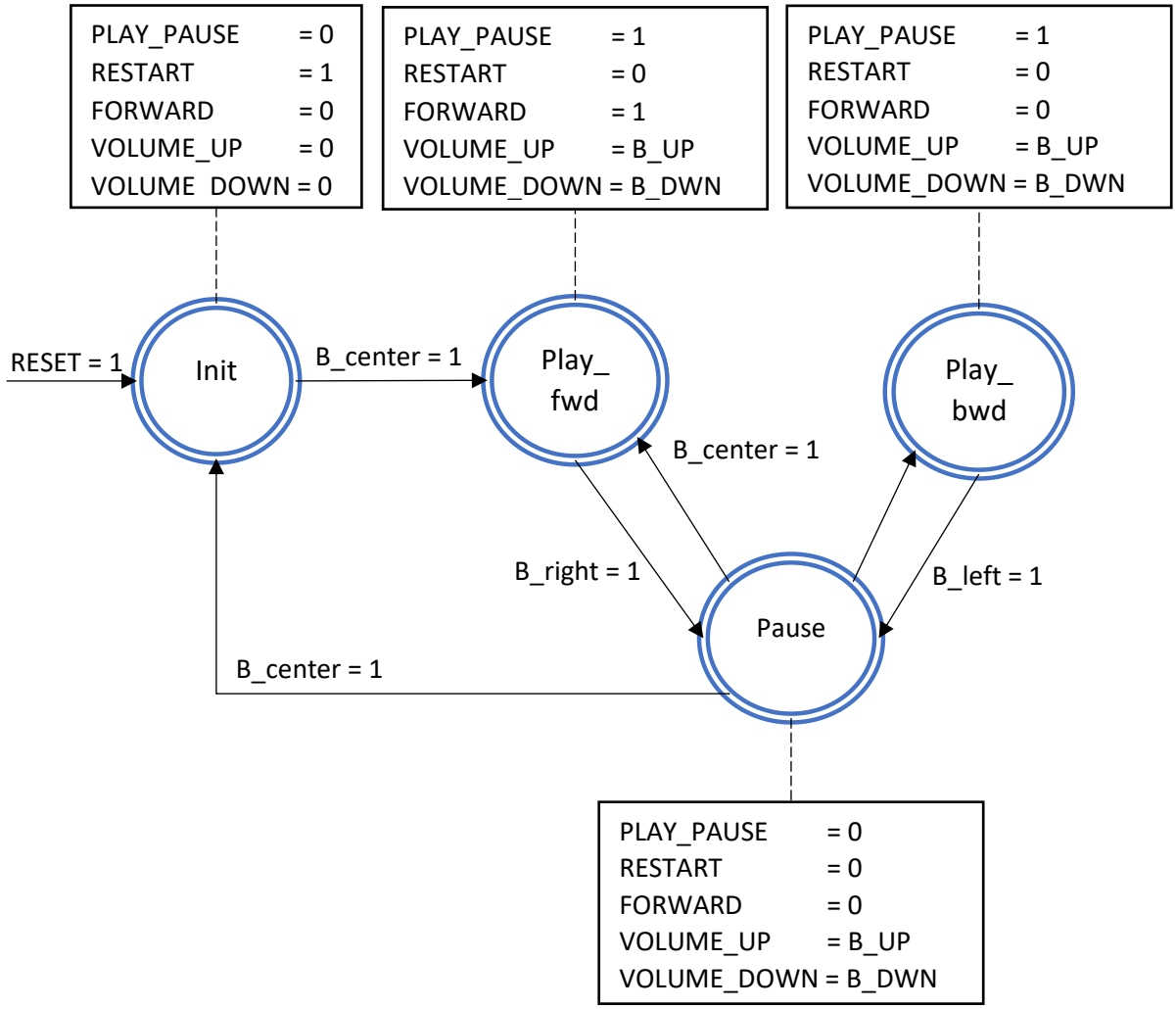


Figure 8 : Diagramme d'état de la FSM

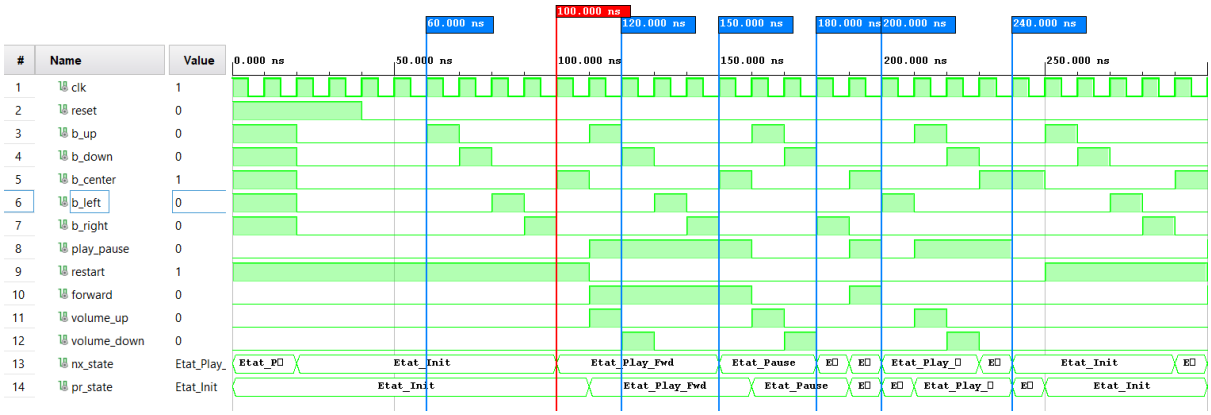


Figure 9 : Simulation de la Machine à état fini

D'après le diagramme d'état de la FSM et la simulation, nous pouvons confirmer le fonctionnement de cette dernière.

- 60ns : nous sommes dans un état d'initialisation et peut importe l'appui sur des boutons autres que b_center, aucune sortie ne passe à 1 et nous restons dans l'état init ;

- 100ns : l'appui sur b_center permet de passer l'état suivant à play_fwd ;
- 120ns : les sorties correspondent à ce qui est attendu dans l'état play_fwd ;
- 150ns : un nouvel appui sur b_center permet de passer à l'état pause ;
- 180ns : un appui sur b_right permet de revenir dans l'état play_fwd lorsque nous sommes dans l'état pause ;
- 200ns : dans l'état pause, un appui sur b_left permet d'aller dans l'état play_bwd avec les sorties correspondantes au diagramme d'état ;
- 240ns : dans l'état pause, un nouvel appui sur b_center permet de revenir dans l'état init.

Cette simulation permet donc bien de valider le fonctionnement de notre machine à état puisque les différents états et sorties correspondent à ce qui est décrit par le diagramme de la FSM.

Nous pouvons également noter l'utilité du détecteur d'impulsion puisqu'à 240ns le b_center a été simulé en maintien et la FSM a passé 2 états. Sans ce bloc, notre FSM serait instable en l'état et nous devrions la redesigner afin de prendre en compte des états intermédiaires.

2.6. Transcodeur

Voici tout d'abord le schéma RTL de ce bloc :

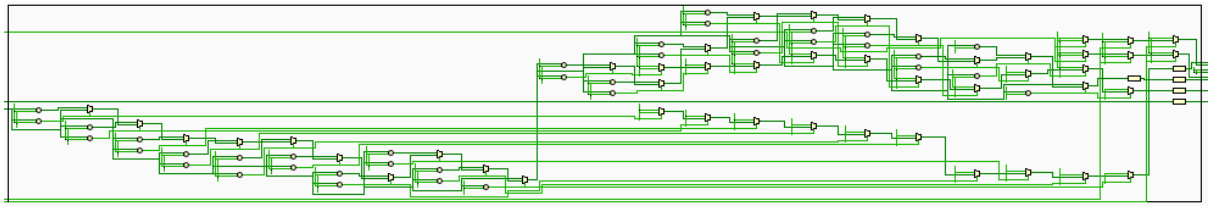


Figure 10 : Schéma RTL du « Transcodeur »

Ce bloc est combinatoire et permet de préparer les données en entrées de manière à les afficher sur les 7 segments.

Voici les entrées :

- « play_pause » : indique si le lecteur est en pause (à 0) ou en lecture (à 1) ;
- « restart » : indique si le lecteur est à l'état init ;
- « forward » : indique le sens de lecture ;
- « nbCpt1_9 » : indique le niveau sonore ;
- « nbCpt1_599 » : indique le temps écoulé depuis le début du morceau.

Les 3 premières entrées sont utilisées pour coder les 4 afficheurs 7 segments sur la gauche de la carte, le compteur de 1 à 9 est affiché sur 1 seul afficheur 7 segments et le compteur 1 à 599 est sur les 3 afficheurs restant. Il est à noté que l'affichage du niveau sonore et du temps doivent être séparés par un point sur les afficheurs.

Puisque nous avons 8 afficheurs 7 segments avec en supplément un point, nous aurons alors 8 sorties sur 8 bits correspondant chacune à un afficheur.

2.7. Sélecteur

Voici tout d'abord le schéma RTL de ce bloc :

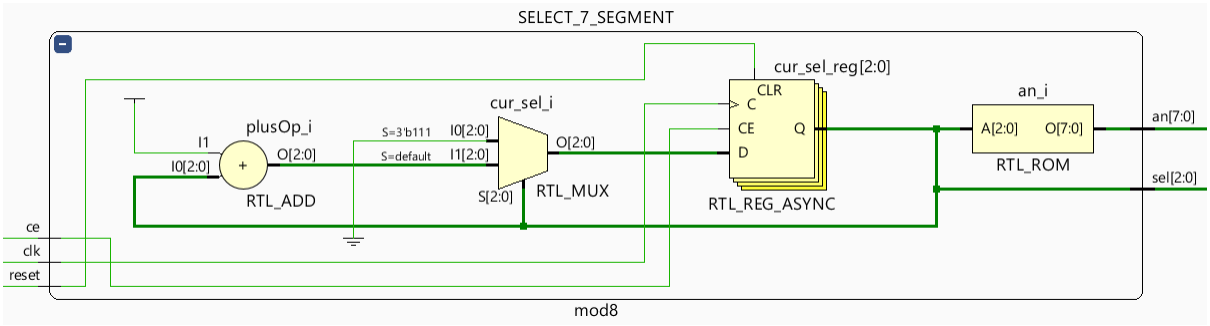


Figure 11 : Schéma RTL du « Sélecteur de 7 segments »

Ce bloque permet de sélectionner les afficheurs 7 segments en propageant un 0 (actif à l'état bas). Nous avons une sortie sur 8 bits pour commander 1 à 1 les 8 afficheurs 7 segments et une sortie « sel » qui permet de transmettre l'information de l'afficheur sélectionné au bloc multiplexeur 8 vers 1 présenté juste après.

Ce bloc fonctionne à 3kHz avec le Ce relié au « CePersistance ». En fonctionnant à cette fréquence, nous pouvons changer d'afficheurs à 3kHz, ce qui permet de bien générer la persistance générique.

2.8. Multiplexeur 8 vers 1

Voici le schéma RTL de ce bloc :

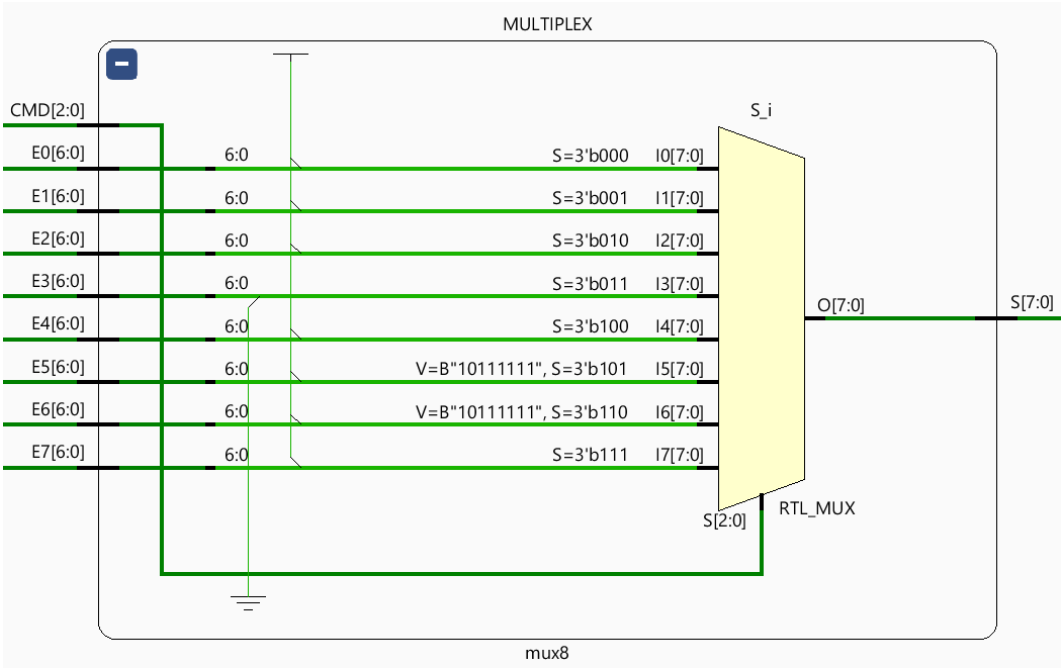


Figure 12 : Schéma RTL du « multiplexeur 8 vers 1 »

La sélection vient du bloc sélecteur présenté précédemment et permet d'afficher la bonne entrée sur le bon afficheur 7 segments.

3. Contrôle de l'audio et production personnelle

Comme précisé dans le cahier des charges, ce projet se découpe en 3 sous parties. Ici je présenterai la partie concernant le stockage et la modulation des échantillons audio.

Voici tout d'abord un schéma RTL des blocs :

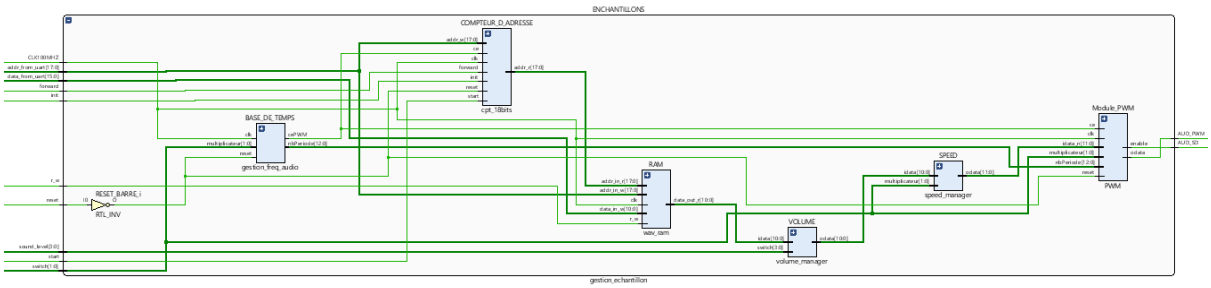


Figure 13 : Schéma RTL du module "Echantillonneur"

Ce bloc est séparé en 6 modules :

- [BASE_DE_TEMPS] : permet de générer un clock enable en fonction de la vitesse de lecture souhaitée ;
- [COMPTEUR_D_ADRESSE] : permet de parcourir la mémoire de type RAM ;
- [RAM] : mémoire qui stocke les échantillons audios. Elle est remplie à partir d'une liaison UART fournie par l'enseignant ;
- [VOLUME] : permet de modifier la valeur de l'échantillon de la RAM pour modifier le volume sonore en sortie ;
- [SPEED] : permet de modifier la valeur de l'échantillon afin de l'adapter au nombre de période disponible pour moduler le signal ;
- [Module_PWM] : permet de moduler les échantillons sur la sortie jack de la carte.

3.1. Mémoire RAM

Voici tout d'abord le schéma RTL de cette RAM :

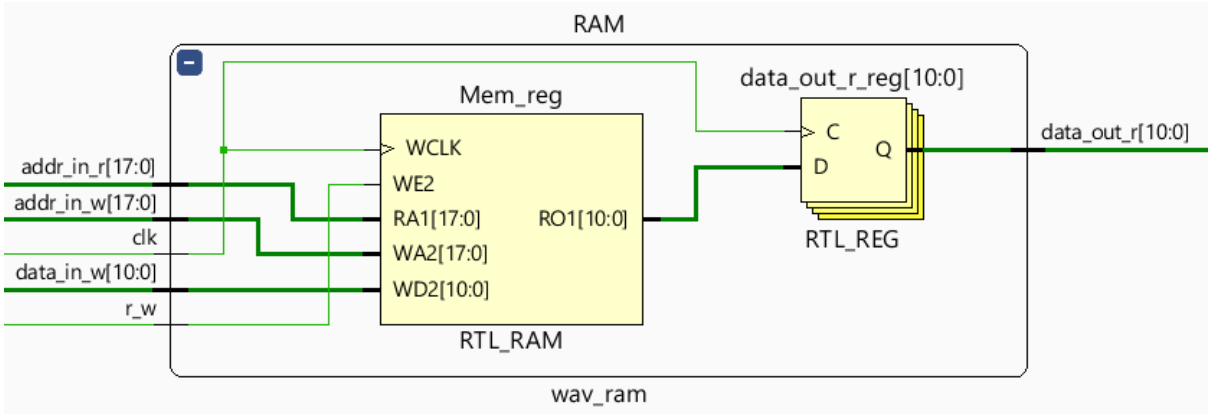


Figure 14 : Schéma RTL de la « RAM »

Cette RAM possède des emplacements mémoire de 11 bits sur 2^18 bits d'adressages soit 262 144 valeurs ce qui représente un total de 88 blocs BRAM sur la carte NEXYS A7. Les valeurs stockées sont signées mais sont transmises aux différents blocs sous forme de « STD_LOGIC_VECTOR ». Cette mémoire est remplie à partir d'une liaison série. Elle a été conçue sous forme de mémoire à double accès, elle peut donc être lue et rempli en même temps. Nous avons donc pour son architecture, une adresse de lecture et une adresse d'écriture en entrée du bloc. Nous avons également une entrée permettant de mettre à disposition les données devant être recopiées dans la mémoire lorsqu'un signal est à 1 (« RW »). Ce dernier indique à la mémoire lorsqu'une donnée est prête à être copiée (mode écriture) ou si la mémoire doit lire les différentes valeurs (mode lecture).

La mémoire est à accès séquentiel (cadencé à 100MHz) et ne possède pas de « clock enable » pour accélérer son accès par rapport au reste du système. Elle est également dite à lecture prioritaire puisque peu importe l'état de « RW », la mémoire recopiera à chaque front d'horloge sa valeur stockée à l'adresse de lecture sur sa sortie.

3.2. Contrôle du son

Voici tout d'abord le schéma RTL de ce bloc :

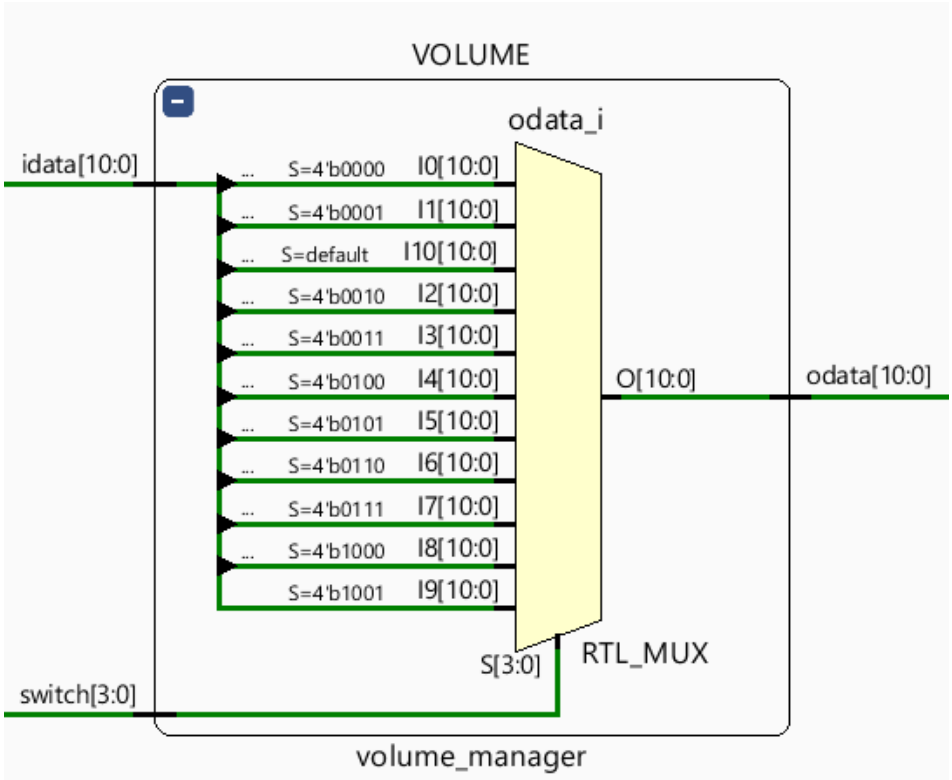


Figure 15 : Schéma RTL de « Contrôleur de volume »

Nous voyons que d'après le schéma RTL, il ne s'agirait que d'un multiplexeur mais son fonctionnement est quelque peu plus compliqué. Cette partie récupère la valeur du compteur présente dans la partie contrôle du lecteur MP3 (valeur allant de 1 à 9) sur son entrée « switch ». En fonction de cette valeur, nous effectuons une division sur les valeurs reçues de la RAM afin de diminuer l'amplitude du signal et donc de diminuer le son en sortie de la prise jack.

Les valeurs reçues sur « idata » sont de type STD_LOGIC_VECTOR mais représentent en réalité des nombres signés sur 11 bits. Il faut donc prendre en compte le bit de signe lors de la division.

Puisque nous avons 9 niveaux de son et 11 bits pour coder l'information (bit de signe compris), nous pouvons, au lieu d'effectuer une division gourmande en ressource, réaliser un décalage de bit.

Pour se faire, il suffit de garder tous les bits pour la valeur 9 (qui représente le niveau de son maximum) et de retirer 1 bit de poids faible à chaque diminution de la valeur pour diviser par 2 successivement. Etant donné qu'il s'agit de nombre signé, nous ne pouvons pas juste mettre un 0 à la place des bits de poids fort mais devons propager le bit de signe pour prendre en compte les nombres négatifs.

```
with switch select
odata <=
    idata(10)&idata(10)& idata(10 downto 2)  when "0111",
    idata(10)& idata(10 downto 1)    when "1000",
    idata   when "1001",
    idata(10 downto 4)&idata(10)&idata(10)&idata(10)&idata(10) when OTHERS;
```

Nous avons ci-dessus une partie du code utilisé. Pour la valeur maximale (« 1001 » soit 9), nous affectons à la valeur de sortie toute la plage d'entrée. Cependant, lorsque le compteur vaut 8 (« 1000 »), nous ne prenons que les 10 bits de fort en supprimant le bit de faible. Pour avoir toujours une valeur sur 11 bits et toujours signé, nous remettons un bit de poids fort du signe du nombre d'origine devant la nouvelle valeur.

Le même calcul est effectué pour les autres valeurs mais en diminuant toujours le nombre de bit de poids fort pris sur la valeur de départ. Voici une simulation présentant le fonctionnement :

La simulation concernant cette partie est présentée en partie 3.7 pour valider le bloc au complet.

3.3. Principe de l'augmentation et diminution de la vitesse de lecture

Cette partie concerne l'ajout personnel à ce projet. Le but ici est de pouvoir lire 2 fois plus rapidement ou 2 fois plus lentement le même morceau audio. Il ne s'agit pas uniquement d'un bloc mais d'une fonction agissant sur plusieurs blocs du contrôle audio. Voici les tâches à effectuer en fonction de la vitesse de lecture souhaitée :

- Modification de la fréquence d'échantillonnage ;
- Modification du nombre de période nécessaire pour la PWM ;
- Modification de la valeur des échantillons pour s'adapter au nombre de période.

Ces 3 blocs seront décris dans les prochaines parties.

Nous souhaitions effectuer une augmentation de la vitesse jusqu'à 4 fois la vitesse de base mais la carte possède un filtre passe bas nous limitant à une vitesse 2 fois plus élevée. Ce filtre sera présenté dans le bloc de PWM.

3.4. Gestion de la fréquence d'échantillonnage

Voici tout d'abord le schéma RTL de ce bloc :

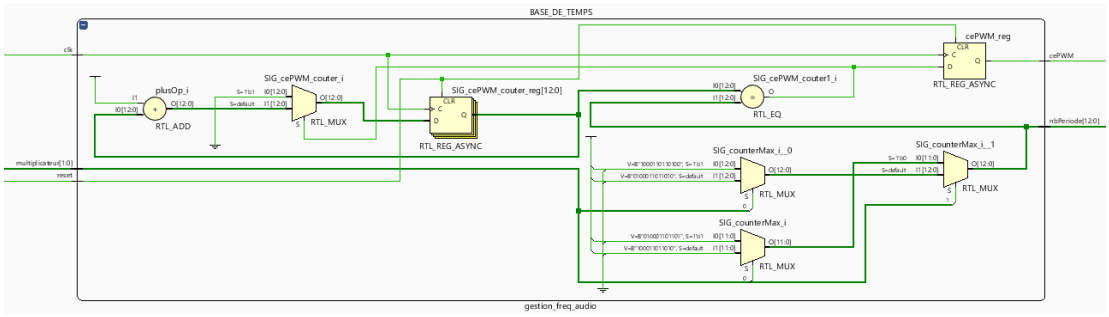


Figure 16 : Schéma RTL de « gestion de fréquence audio »

La fréquence d'échantillonnage est de 44 100Hz avec un système fonctionnant à 100MHz. Nous cherchons donc à générer une horloge de 44 100Hz afin de pouvoir moduler nos échantillons.

$$nb\ de\ période = \frac{100\ 000\ 000}{44\ 100} = 2\ 267$$

Lors de la multiplication ou de la division, nous multiplions ou divisons par 2 le nombre de période de base puisque nous parcourons plus rapidement la mémoire et avons donc plus ou moins de temps pour moduler l'information dans la PWM.

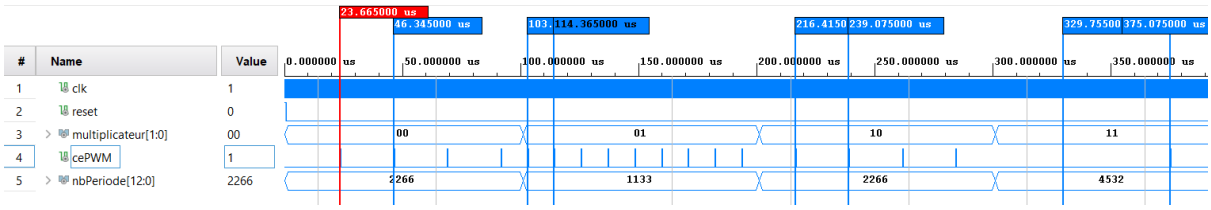


Figure 17 : Simulation de la synthèse de fréquence

Cette simulation montre qu'en fonction du multiplicateur en entrée, le nombre de période change et les « clock enable » sont plus ou moins espacés par un nombre de coup d'horloge égale au nombre de période.

3.5. Prise en compte de la vitesse sur la valeur des échantillons

Voici tout d'abord le schéma RTL de ce bloc :

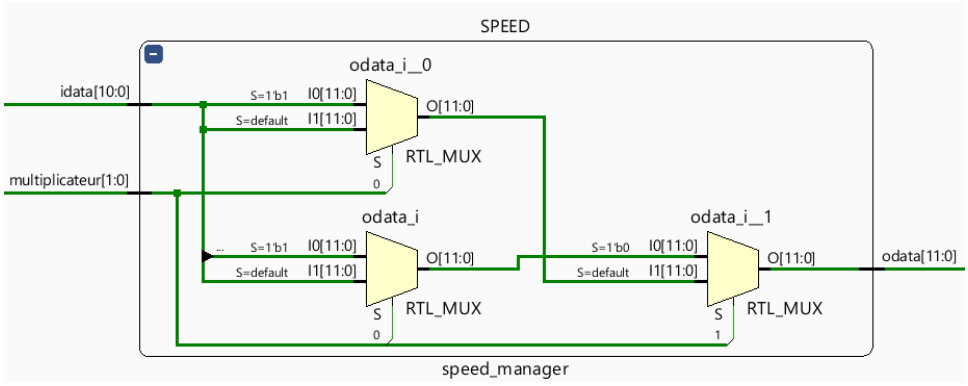


Figure 18 : Schéma RTL du bloc « speed manager »

Ce bloc est construit de la même manière que pour le contrôle du niveau sonore. En fonction de l'entrée multiplicateur sur 2 bits nous augmentons la vitesse ou la diminuons. Nous divisons par 2 la valeur si nous accélérons car la fréquence augmente, nous avons donc pour la PWM moins de période disponible et la valeur de l'échantillon doit alors être adaptée au nombre de période possible. A l'inverse, nous augmentons la valeur de l'échantillon si nous souhaitons diminuer la vitesse car nous aurons plus de période disponible pour coder l'information dans le bloc PWM. La méthode utilisée ici est la même que pour le contrôle du son. En effet, la division et la multiplication par 2 correspond à un décalage de bit.

La description de multiplexeur permet de réaliser cette partie tout en prenant en compte la propagation du bit de signe.

3.6. Modulateur numérique

Voici tout d'abord le schéma RTL de ce bloc :

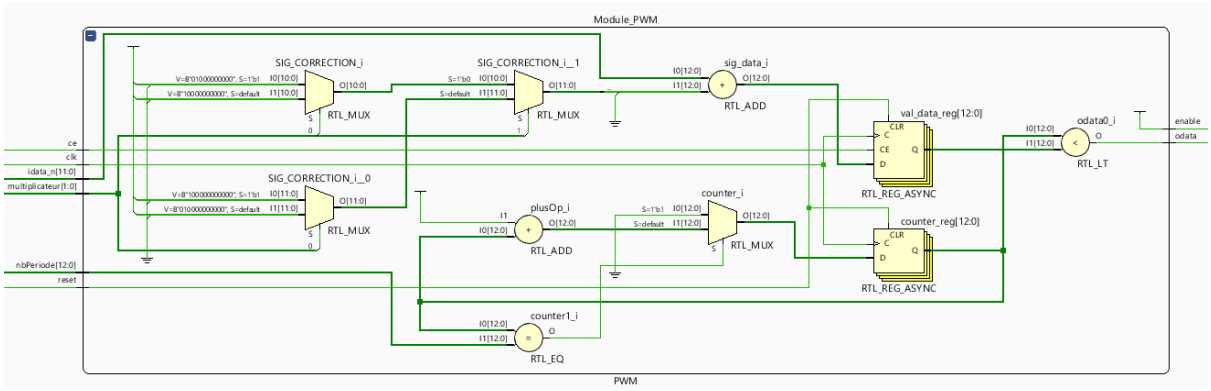


Figure 19 : Schéma RTL du bloc « PWM »

Voici également comment fonctionne la PWM :

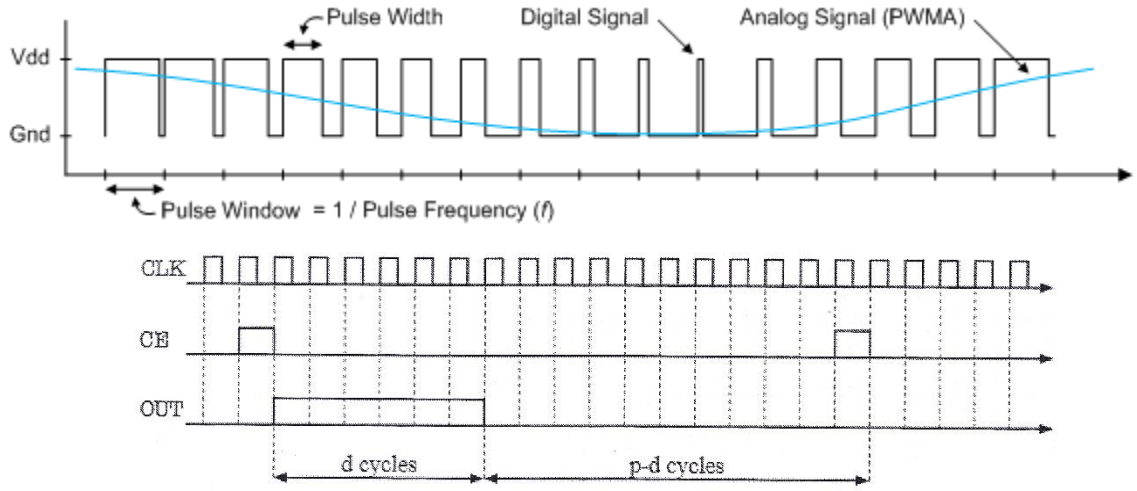


Figure 20 : Fonctionnement de la PWM

Le nombre de cycle p est calculé en fonction de la période d'échantillonnage du signal. Ce calcul a déjà été réalisé dans la partie générant la fréquence d'échantillonnage et est transmise en entrée de ce bloc PWM.

Les données stockées dans la RAM sont des nombres signés entre -1024 et + 1023. Pour pouvoir avoir un nombre positif, nous ajoutons 1024 ce qui permet de ramener les valeurs entre 0 et 2047. La fréquence d'échantillonnage de base permet d'avoir 2 266 périodes. Nous pouvons alors faire un comparateur qui vérifie que le nombre de période est inférieur à la valeur. Si c'est le cas, la sortie est à 1 sinon elle est à 0. Cela permet de bien moduler la largeur d'impulsion.

Une subtilité est à prendre en compte en cas de modification de la vitesse. Si nous allons 2 fois plus vite, la valeur en entrée sera divisée par 2 par le bloc précédent ce qui ramène la valeur entre -512 et +511. La valeur de correction sera alors de +512 dans le cas de la vitesse fois 2 et de +2048 lorsque nous allons 2 fois moins vite.

La simulation de la totalité du bloc est disponible en partie 3.7.

En sortie de ce bloc, un filtre est présent et bloque les hautes fréquences. Voici un diagramme de Bode présentant la réponse du filtre :

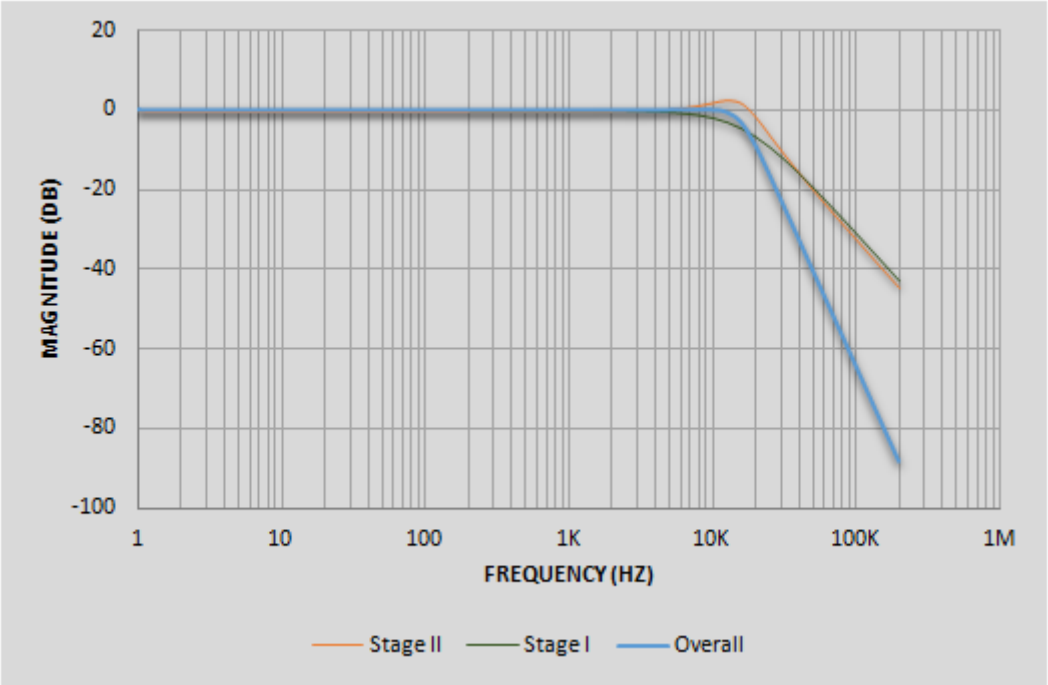


Figure 21 : Réponse du filtre de Butterworth

Avec un fois 4, nous n'avons que 566 coups d'horloge pour coder l'information ce qui est trop rapide par rapport au filtre puisque la fréquence des signaux audio sera trop élevée (en considérant un échantillonnage à 176 400Hz). Dans le même sens, un ralentissement de 4 fois la vitesse de base donne trop de période et provoque un suréchantillonnage dans le bloc précédent.

3.7. Validation

Pour vérifier le fonctionnement de ces blocs, nous n'utiliserons pas la RAM ou le compteur d'adresse sur 18 bits mais une ROM fournis par l'enseignant et un compteur d'adresse sur 16 bits (44 100 échantillons). Cette ROM contient un sinus qui nous permettra de vérifier la capacité d'augmenter ou de diminuer la vitesse de lecture ainsi que la prise en compte du niveau sonore.

Voici la simulation réalisée :

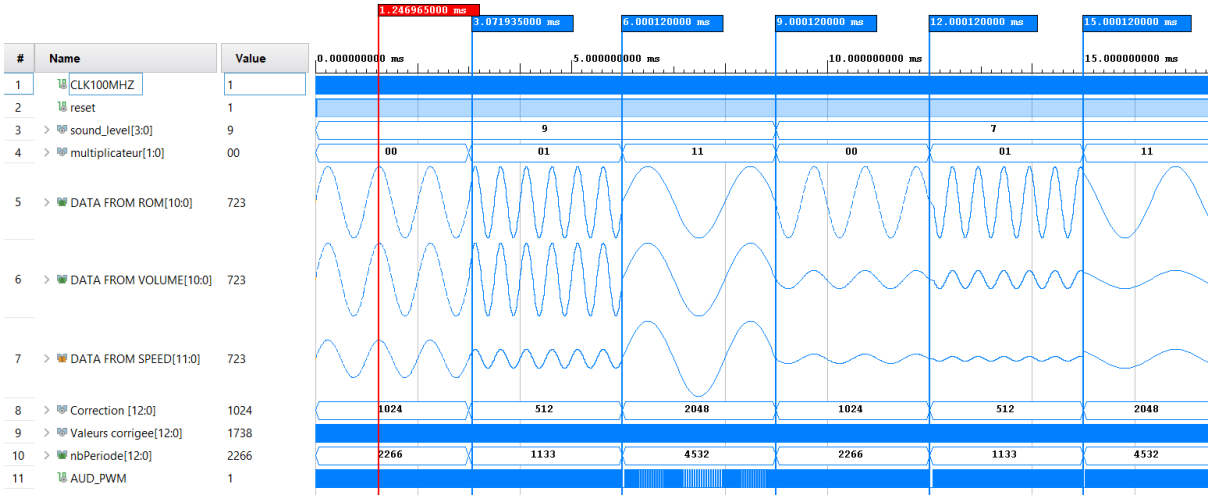


Figure 22 : Simulation du contrôle audio et de la production personnelle

La ligne 5 correspond aux données en sortie de la mémoire ROM, la ligne 6 correspond à la sortie du bloc de volume et à l'entrée du bloc « speed_manager ». La ligne 7 correspond à l'entrée du bloc PWM après être passé par le bloc de son puis de vitesse.

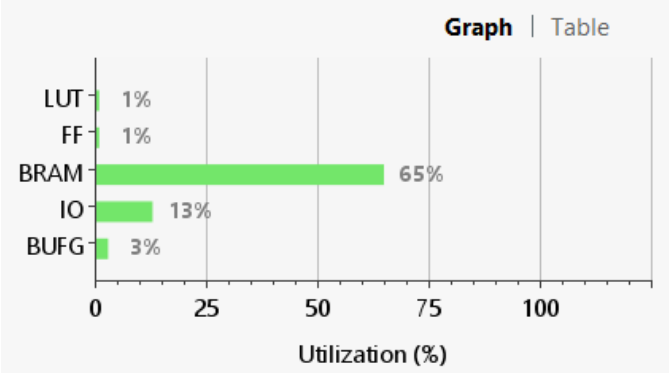
La valeur corrigée ligne 9 correspond à la somme de la ligne 7 avec la ligne 8 pour ramener la valeur sur le nombre d'échantillons ligne 10.

- Entre 0 et 3ms
  - Vitesse normal (switch « 00 ») ;
  - Son à 100% (sound_level à 9) ;
  - Valeur maximale à 723 en sortie de la ROM ainsi qu'en entrée du bloc PWM ce qui montre que le bloc de « volume_manager » et « speed_manager » laisse passer la totalité du signal jusqu'au module PWM ;
  - Ce n'est pas visible ici mais en zoomant sur cette partie en particulier, le modulateur numérique module sur toute sa plage de valeur comme attendu.
- Entre 3 et 6ms
  - Vitesse fois 2 (switch « 01 ») ;
  - Son à 100% (sound_level à 9) ;
  - Valeur maximale à 723 en sortie de la ROM mais de seulement 361 en entrée du bloc PWM soit 723/2. C'est le bloc « speed_manager » qui vient ici diviser le signal par 2 puisqu'en allant 2 fois plus vite nous avons 2 fois moins de période d'échantillonnage (visible sur la ligne 10) qu'à vitesse constante.
  - Puisque la donnée a été divisée dans le bloc speed_manager, nous notons une diminution de l'amplitude en entrée du bloc PWM.
- Entre 6 et 9ms
  - Vitesse diviser par 2 (switch « 11 ») ;

    - Son à 100% (sound_level à 9) ;
    - A l'inverse de la situation précédente, la vitesse étant réduite nous avons plus de période pour échantillonner le signal. Nous avons donc en entrée de la PWM une valeur de 1448 qui s'ajoute à 2048 pour ramener cette donnée en unsigned.
- Entre 9 et 12ms
    - Vitesse normal (switch « 00 ») ;
    - Son à 100% (sound_level à 7) ;
    - Ici nous notons une diminution de l'amplitude en entrée du bloc speed_manager ce qui correspond bien aux opérations effectuées dans le bloc de volume_manager.
- Entre 12 et 15ms
    - Vitesse fois 2 (switch « 01 ») ;
    - Son à 100% (sound_level à 7) ;
    - En plus d'une diminution de l'amplitude en entrée du bloc speed_manager, nous notons une diminution en entrée du bloc PWM ce qui montre la double action des 2 blocs.
- A partir de 15ms
    - Vitesse diviser par 2 (switch « 11 ») ;
    - Son à 100% (sound_level à 7) ;
    - Même constat que précédemment mais le bloc speed_manager réaugmente l'amplitude puisque nous avons plus de période disponible pour moduler le signal.

4. Ressources utilisées

La consommation de ressources sur la NEXYS A7 – 100T est limitée.



Resource	Utilization	Available	Utilization ...
LUT	843	63400	1.33
FF	501	126800	0.40
BRAM	88	135	65.19
IO	28	210	13.33
BUFG	1	32	3.13

Nous n'utilisons quasiment aucune ressource disponible à par la mémoire. Nous avons dû prendre la 100T car cette carte possède plus de blocs BRAM et la 50T n'en avait pas assez. La mémoire de 18 bits d'adressage est le maximum puisque le passage à 19 bits nous fera consommer plus de blocs que disponibles sur la carte.