1 But du T.P.

Le but de ce TP est d'implémenter un système temps réel à l'aide d'un FPGA. Ici, l'application étudiée portera sur le domaine audio. La carte FPGA devra effectuer les tâches suivantes :

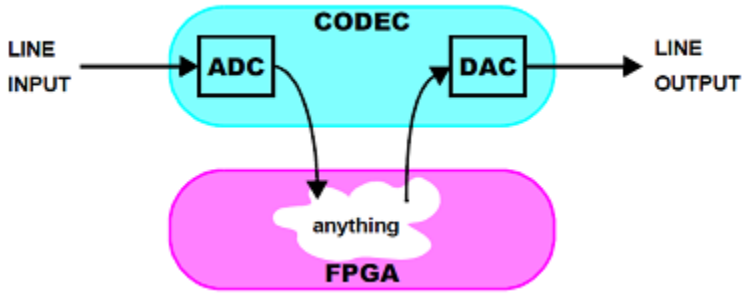
- Implémenter un CODEC pour émettre et recevoir un signal audio
- Ajuster le niveau sonore
- Implémenter un bargraph
- Implémenter un filtre numérique (passe-bas, passe-bande, etc)

Vous utiliserez la carte de développement **DE2-115** qui comprend un FPGA de la famille **Cyclone IV** de chez **ALTERA**. La référence exacte du FPGA est **EP4CE115F29C7**, il possède 114480 éléments logiques et 532 multiplieurs hardware définis sur 9 bits.

2 Présentation générale

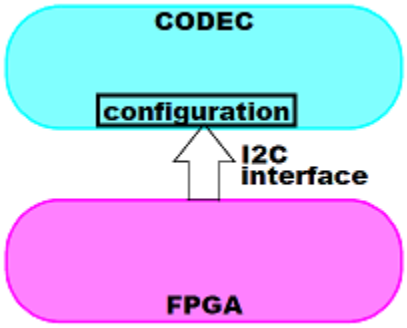
La carte de prototypage utilisée pour ce TP (DE2 ou DE2-115) possède un CODEC audio.

1. Ce composant numérisera un signal audio appliqué à l'entrée LINE-IN
2. Le signal numérisé sera envoyé au FPGA à travers un bus série dédié
3. Tout le traitement numérique sera effectué par le FPGA
4. Le signal traité sera renvoyé au CODEC
5. Ce dernier convertit le signal numérique reçu en signal analogique. Ce signal sera alors transmit à la sortie LINE-OUT



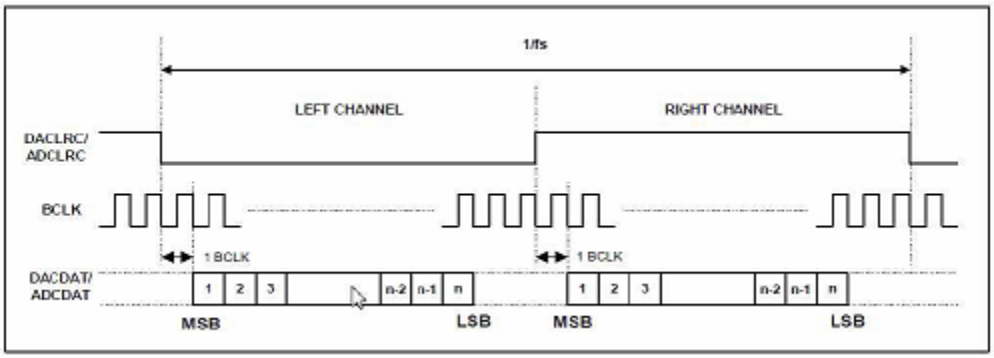
Le CODEC peut être configuré grâce à divers paramètres comme la fréquence d'échantillonnage, la résolution (nombre de bits du signal numérique), le volume, la sélection de l'entrée (MIC ou LINE-IN), format audio,etc...

La configuration du CODEC doit être transmise au maitre : dans ce cas, le FPGA envoie les informations nécessaires à travers une liaison I2C.



Une fois la configuration effectuée, le convertisseur analogique-numérique (ADC) peut commencer à numériser le signal audio analogique. Ce signal est alors transmis au FPGA via une liaison série. En contrepartie, le signal numérique est envoyé au convertisseur numérique-analogique (DAC) via cette même liaison série. Il s'agit d'une liaison I2S, elle est composée des signaux suivants :

- ADCDAT : Flux de données allant de l'ADC vers le FPGA
- DACDAT : Flux de données allant du FPGA vers le DAC
- BCLK : Horloge de synchronisation des signaux ADCDAT et DACDAT
- DACLRC : Horloge d'alignement montrant la présence du canal gauche ou droit sur DACDAT (CODEC vers FPGA)
- ADCLRC : Horloge d'alignement montrant la présence du canal gauche ou droit sur ADCDAT (CODEC vers FPGA)



Le CODEC reçoit l'horloge principale (MCLK) du FPGA, via la voie AUD_XCK. Cette horloge est générée grâce à un oscillateur 50 MHz présent sur la carte de développement, dont la fréquence sera divisée par 4. De ce fait, on obtient un signal MCLK de fréquence 12,5 MHz.

Le CODEC est également configuré pour échnatillonner le signal d'entrée à $\frac{MCLK}{256}$, d'où une fréquence d'échantillonnage de $\frac{12,5MHz}{256} = 48,828kHz$.

Durant une période d'échantillonnage, le signal BCLK génère 64 périodes. Sa fréquence sera alors de $48,828kHz \times 64 = 3,125MHz$.

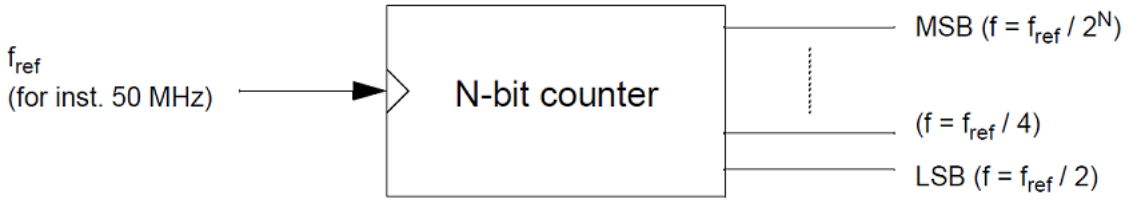
Par exemple, si le signal audio est numérisé sur 16 bits, durant une période d'horloge, le flux de donnée sur DACDAT ou ADCDAT sera composé des 16 bits du canal gauche, puis de 16 bits sans intérêt ou bits de remplissage, puis des 16 bits du canal droit, et enfin 16 bits de remplissage.

3 Clignotement d'une LED

Un problème courant dans les systèmes numériques est la synchronisation de plusieurs programmes utilisant une seule et même horloge.

L'objectif de cet partie est de faire clignoter une LED à la fréquence de 1 Hz en utilisant l'oscillateur 50 MHz présent sur la carte de développement.

Un compteur est alors nécessaire pour diviser la fréquence de l'oscillateur. La sortie d'un compteur binaire décrit une séquence fixe sur N bits allant du bit de poids fort (MSB) au bit de poids faible (LSB). La fréquence d'horloge est divisée par deux à mesure que le poids du bit augmente.



1. Dans un premier temps, vous allez concevoir un compteur 26 bits ayant les caractéristiques suivantes :
   - L'entrée CLOCK_50 est reliée à l'oscillateur 50 Mhz
   - Les quatre sorties représentant les quatre bits de poids fort du compteur seront reliés à quatre LEDs (LEDR17 à LEDR14) situées en bas de la carte de développement
   - La sortie LED sera reliée à la LED (LEDG0). Pour l'instant, cette sortie désignera le MSB

   (a) Ouvrir le logiciel QUARTUS.

   (b) Créer un nouveau projet et le nommer TP4_CODEC_audio.

   (c) Écrire la description VHDL correspondante.

   (d) Générer le module graphique correspondant à ce composant.

   (e) Créer un fichier TP4_CODEC_audio.bdf et réaliser le compteur 26 bits.

   (f) Programmer le système sur la carte de test afin de vérifier le fonctionnement. Commenter.

2. Dans l'exemple précédent, la fréquence du MSB est de $\frac{50MHz}{2^{26}} = 0,75Hz$, ce qui n'est pas suffisant pour atteindre notre objectif, à savoir une fréquence de 1 Hz. Par conséquent, nous avons besoin de diminuer le cycle du compteur en le remettant à zéro avant qu'il atteigne sa valeur maximale $2^{26} - 1$.

   Maintenant, nous voulons que la LED clignote exactement à la fréquence de 1 Hz (la période est de 1 sec) : l'état de la LED doit changer tous les 0,5 sec. Cette durée peut être détecté à condition que le compteur aille de 0 à 24999999. Une fois ce nombre atteint, le compteur est remis à zéro et la LED change d'état.

   (a) Modifier la description VHDL précédente afin de respecter le cahier des charges.

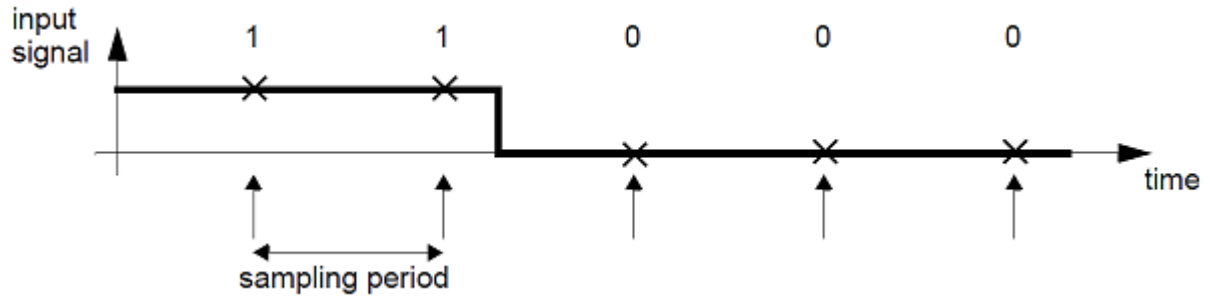
   (b) Programmer le système sur la carte de test afin de vérifier le fonctionnement. Commenter.

3. Nous voulons maintenant ajouter un afficheur 7 segments qui compte de 0 à 9 à la fréquence 1 Hz.
   (a) Écrire la description VHDL correspondante.
   (b) Générer le module graphique correspondant à ce composant.
   (c) Insérer le module dans le fichier TP4_CODEC_audio.bdf.
   (d) Programmer le système sur la carte de test afin de vérifier le fonctionnement. Commenter.

4 Détection d'un appui sur le bouton poussoir

Dans cet exercice, nous voulons arrêter puis redémarrer le compteur après chaque appui sur le bouton poussoir. Sur la carte de développement, chaque bouton poussoir (KEY0 à KEY3) est relié à une entrée du FPGA, dont l'état par défaut est 1. Dès que le bouton poussoir est pressé, l'entrée passe à l'état 0.

Pour détecter un front descendant, l'entrée du FPGA doit être échantillonnée à une fréquence assez faible, un registre à décalage servant à stocker les échantillons successifs. Nous admettrons qu'un front descendant se produit quand la séquence "1-1-0-0" est détectée, comme montré dans le chronogramme ci-dessous :



Vous allez créer un composant "key_sampler" possédant une sortie qui change d'état dès qu'un front descendant est détecté à l'entrée du circuit. Ce composant sera utilisé pour stopper ou non le compteur. Il comportera les éléments suivant :

- Un compteur sur 10 bits fournissant une horloge de fréquence 50 kHz pour échantillonner le signal provenant du bouton poussoir
- Un registre à décalage sur 4 bits pour stocker les 4 derniers échantillons
- Un circuit combinatoire pour détecter la séquence "1-1-0-0"

1. Écrire la description VHDL correspondante.
2. Générer le module graphique correspondant à ce composant.
3. Modifier la description VHDL du compteur 26 bits de façon à intégrer une entrée "enable" qui arrête le compteur à l'état bas.
4. Générer le module graphique correspondant à ce composant.
5. Insérer les modules dans le fichier TP4_CODEC_audio.bdf.
6. Programmer le système sur la carte de test afin de vérifier le fonctionnement. Commenter.

5 Configuration du CODEC audio

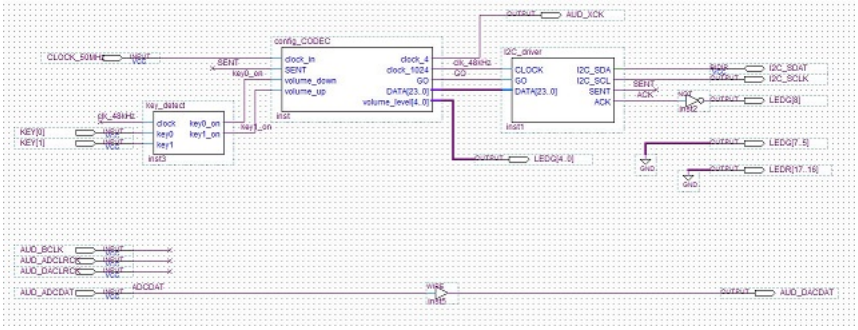
La carte de développement comprend un CODEC audio dont le fonctionnement est décrit au paragraphe 2. Ce CODEC permet notamment de numérisé un signal audio arrivant sur l'entrée LINE-IN, de le traiter et de le renvoyer directement sur la sortie LINE-OUT.

Pour cela, plusieurs fichiers vous sont fournis afin de compléter votre circuit :
— Le fichier **config_CODEC** contient les informations qui doivent être chargées dans les registres de configuration du CODEC
— Le fichier **I2C_driver** est le périphérique qui implémente la liaison I2C entre le FPGA et le CODEC

Le CODEC est configuré avec les paramètres suivants :
— Fréquence d'échantillonnage = 48.8 kHz
— Taille des données audio = 16 bits
— Format audio = I2S (transmission des bits provenant du canal gauche puis du canal droit)
— Entrée LINE-IN vers convertisseur analogique numérique
— Etc...

La configuration du CODEC doit être envoyé après chaque appui sur les touches KEY0 ou KEY1. De plus, le contrôle du volume du casque modifie le niveau sonore par pas de 1 dB. Dans le fichier **config_CODEC**, vous vérifierez que le volume est stocké sur un registre de 5 bits appelé *vol*. Sa valeur (comprise entre 1 et 31) est affiché en binaire à l'aide des LEDs LEDG4 à LEDG0 (en bas à droite sur la carte de développement). Le bouton poussoir KEY1 permet d'augmenter le volume et KEY0 de le diminuer.

1. Écrire la description VHDL correspondant au module **key_detect** qui fournit une pulsation courte après appui sur le bouton poussoir KEY0 ou KEY1. Pour cela, inspirez-vous du programme écrit au paragraphe 4, l'idée étant que le bouton poussoir doit envoyer un zéro si un front descendant est détecté.
2. Générer le module graphique correspondant à ce composant.
3. Ajouter les fichiers **config_CODEC** et **I2C_driver** au projet, puis générer les modules graphiques correspondants.
4. Insérer les modules graphiques dans le fichier TP4_CODEC_audio.bdf en respectant le schéma ci-dessous :

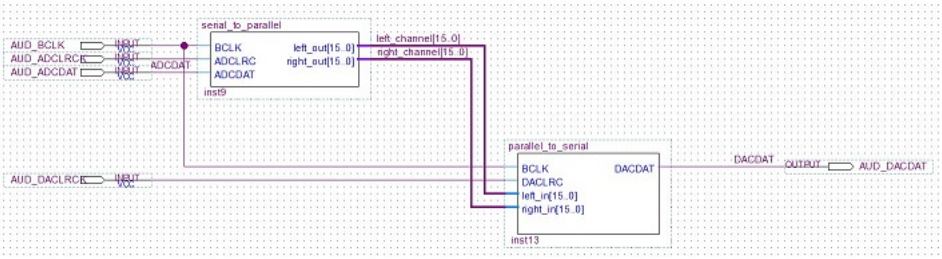


5. Écrire la description VHDL qui permet d'afficher la valeur de la sortie *volume_level* sur les afficheurs 7 segments HEX0 et HEX1.
6. Générer le module graphique correspondant à ce composant.
7. Programmer le système sur la carte de test afin de vérifier le fonctionnement.

6 Implémentation d'un bar graph audio

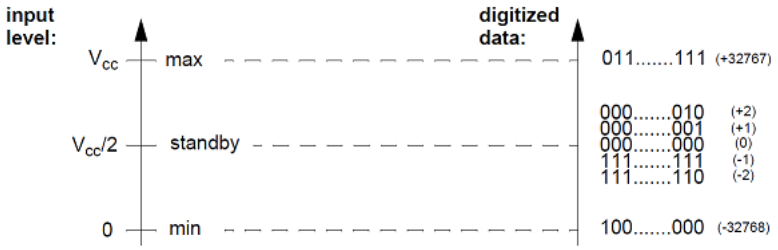
Dans cet exercice, le signal audio (reçu par le CODEC via l'interface série I2S) est converti en un mot de 16 bits par le composant **serial_to_parallel** dans le FPGA. Le mot de 16 bits est alors renvoyé au CODEC via le composant **parallel_to_serial**. Le mot reçu est affiché sur les LEDs LEDR15 à LEDR0.

1. Ajouter les fichiers **serial_to_parallel** et **parallel_to_serial** au projet, puis générer les modules graphiques correspondants.
2. Insérer les modules graphiques dans le fichier TP4_CODEC_audio.bdf en respectant le schéma ci-dessous :



3. Brancher les LEDs LEDR15 à LEDR0 à la sortie *left_channel[15..0]* du composant **serial_to_parallel**.
4. Programmer le système sur la carte de test afin de vérifier le fonctionnement.

   Vous constaterez alors que toutes les LEDs sont allumées (et ce, même s'il n'y a pas de signal présent sur LINE_IN). Cela est dû au fait que le CODEC fournit des informations codées en code complément à deux. De ce fait, tous les bits sont à l'état haut même si le niveau sonore est faible.



   Autour du niveau *standby*, une variation de $\pm 1 LSB$ peut modifier l'état de toutes les LEDs, ce qui n'a aucun sens du point de vue audio. Au lieu d'afficher la donnée brute sur 16 bits, il serait plus intéressant d'afficher sa valeur absolue. Pour cela, vous insérerez le composant **absolute_value** juste avant les LEDs.

5. Modifier le fichier **absolute_value** de façon à ce qu'il affiche la valeur absolue d'un nombre binaire.
6. Générer le module graphique correspondant.
7. Insérer ce module dans le fichier TP4_CODEC_audio.bdf.
8. Programmer de nouveau le système sur la carte de test afin de vérifier le fonctionnement.

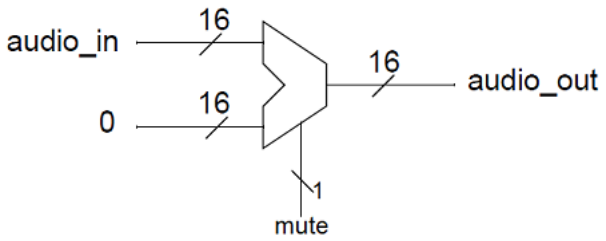
7 Traitement du signal audio

Une fois que le signal audio est disponible sur le FPGA, il peut être traité de différentes façons, par exemple en ajoutant une fonction "muet" ou alors en filtrant numériquement le signal.

7.1 Mise en place de la fonction "muet"

La fonction "muet" peut être mise en place simplement : selon la position d'un switch, le signal audio est transmis tel quel au CODEC ou modifié en une valeur fixe, ici zéro.

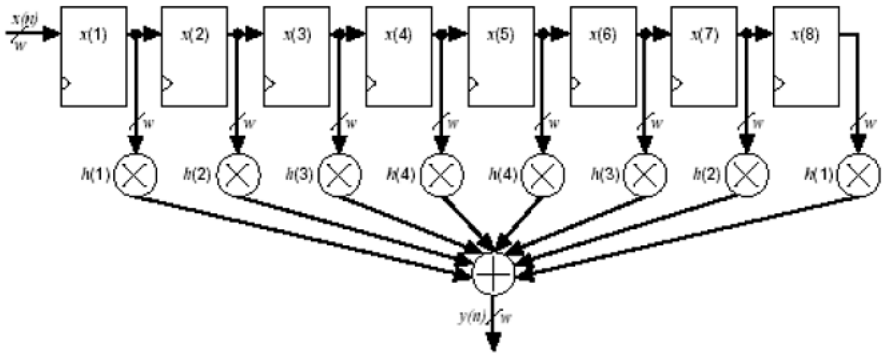
Vous utiliserez le switch SW1 pour mettre en sourdine le canal gauche et SW0 pour le canal droit.



1. Écrire la description VHDL correspondante.
2. Générer le module graphique correspondant à ce composant.
3. Insérer les modules graphiques dans le fichier TP4_CODEC_audio.bdf.
4. Programmer le système sur la carte de test afin de vérifier le fonctionnement.

7.2 Mise en place d'un filtre numérique

Un FPGA permet également d'implémenter un filtrage numérique. Par exemple, un filtre FIR (*Finite Impulse Response*) a la structure suivante :



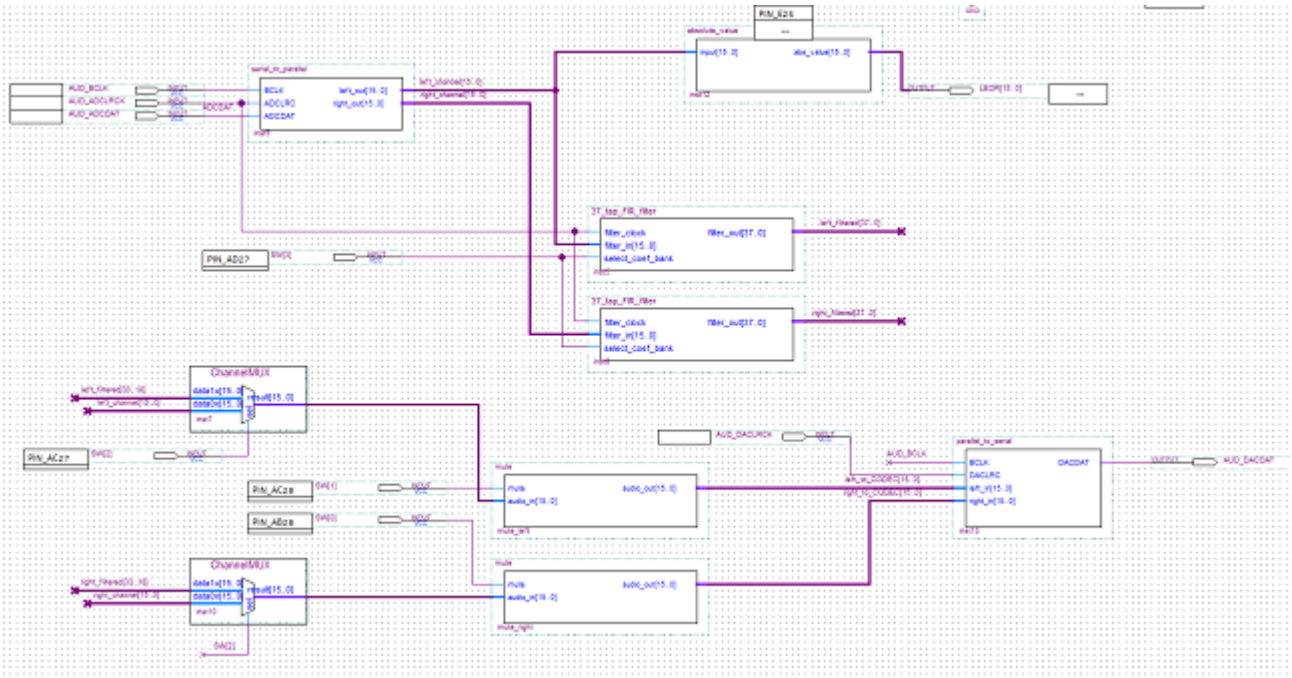
Ce composant utilise des registres, des additionneurs et des multiplieurs, tous disponibles sur FPGA.

Le fichier **FIR_filters** décrit un filtre FIR d'ordre 36 avec 37 coefficients symétriques définis sur 16 bits (soit 19 coefficients différents). Ces coefficients sont stockés dans le fichier **coef_table**. Deux jeux de coefficients sont disponibles et peuvent être sélectionnés via le switch SW3, SW2 permettant de démarrer le filtrage.

Les filtres utilisés sont :
— Un filtre passe-bas de fréquence de coupure 1 kHz
— Un filtre passe-bande de bande passante 1 kHz - 5 kHz

Pour information, le système complet (Interface de configuration du CODEC + interfaces série-parallèle et parallèle-série + filtres FIR + composants logiques additionnels) utilise environ 1500 éléments logiques, soit à peu près 1% du FPGA EP4CE115F29C7 qui en comprend 114 480.

1. Ajouter le dossier **FIR_filters** au projet.
2. Insérer le module graphique **37_tap_FIR_filter.bdf** dans le fichier TP4_CODEC_audio.bdf en respectant le schéma ci-dessous :



3. Programmer le système sur la carte de test afin de vérifier le fonctionnement.
4. Si vous avez le temps, n'hésitez pas à regarder comment le filtre FIR est implémenté dans le FPGA.