VHDL

Réalisation d'un filtre passe bas

Pour la réalisation de ce filtre nous utiliserons la carte ASX PLD.

ASX PLD a été conçue pour permettre à l'utilisateur de réaliser des applications de traitement du signal ou de télécommunication à travers l'utilisation de CPLD et FPGA intégrés au système.

Aujourd'hui, la complexité des composants CPLD et FPGA nécessite l'utilisation de VHDL, langage descriptif de haut niveau .

Afin de faciliter l'apprentissage de ce langage, nous fournissons avec la carte ASX PLD une librairie de plusieurs programmes permettant a l'utilisateur de découvrir graduellement le langage VHDL . Les premiers programmes lui permettront d'utiliser les 8 leds pour la réalisation d'un chenillard ou d'utiliser les 2 afficheurs 7 segments pour réaliser un compteur /décompteur.

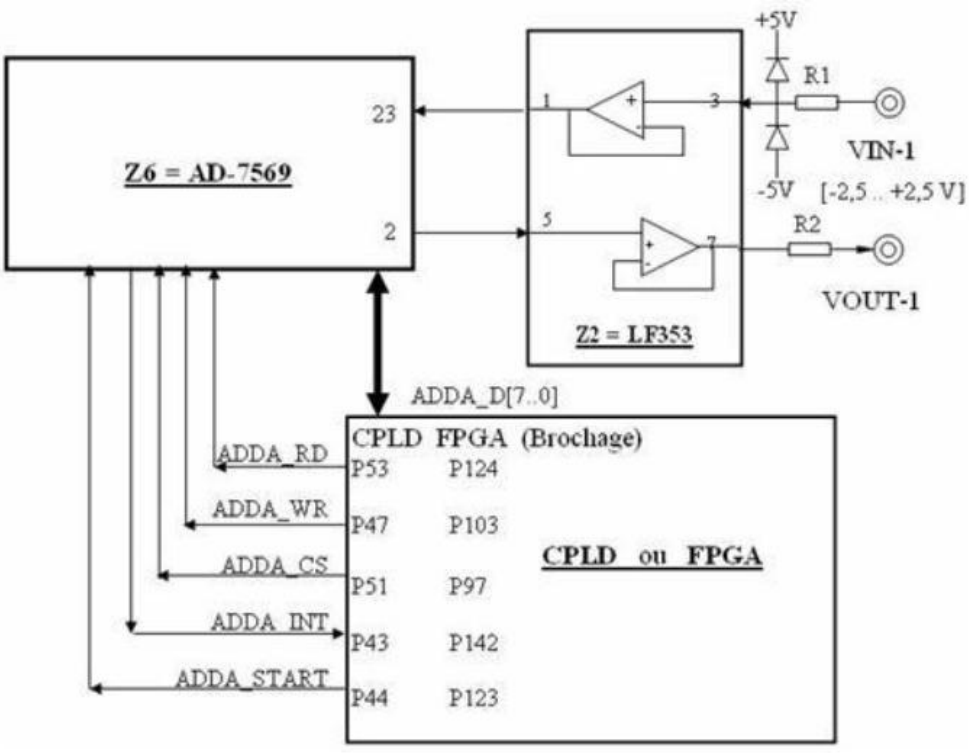
Le TP que nous proposons ci après nécessite d'avoir une connaissance préalable du langage VHDL .

Nous utiliserons le FPGA SPARTAN II de Xilinx pour implanter le gabarit du filtre sélectionné. Ce composant intègre 100,000 portes. Sa capacité interne permet la réalisation d'applications complexes.

Description de matériel

Les entrées/sorties analogiques **<VIN-1>** et **<VOUT-1>** sont reliées au convertisseur **AD7569 (Z6)** sur la carte ASX_PLD. C'est un convertisseur A/D et D/A à bus parallèle sur 8 bits qui peut travailler en signé ou non signé, avec une dynamique de 5 volts ou de 2,5 volts. Les signaux déterminant la configuration ont été fixés au niveau du circuit imprimé et on se limite donc à la configuration suivante :

- **conversion signée et dynamique de 5 volts (de –2,5 à +2,5).**
- **Le temps de conversion est de 2 µs ; soit une fréquence d'utilisation maximale de 500 Khz.**



Le convertisseur **AD7569** possède un bus de données bidirectionnel sur 8 bits et des signaux de contrôle

permettant l'acquisition ou la restitution des données.

Signaux en entrée du convertisseur :

/ADDA_CS : sélection du convertisseur pour une lecture ou une écriture
/ADDA_READ : lecture d'une donnée (conversion AD)
/ADDA_WRITE : écriture d'une donnée (conversion DA)
/ADDA_START : débuter une conversion AD.

Signal en sortie du convertisseur :
/ADDA_INT : indique la fin de la conversion AD

Tous ces signaux sont actifs au niveau bas. Activer un signal signifie le mettre à 0.

Réalisation d'un filtre passe-bas du premier ordre

```
entity SX_FPBAND1 is port (
        RESET               : in  std_logic;          -- reset global
        CLOCK               : in  STD_LOGIC;          -- horloge de la carte (40 MHz)
        ADDA_CS             : out STD_LOGIC;          -- sélection du convertisseur
        ADDA_ST             : out STD_LOGIC;          -- ordre début d'échantillonnage
        COM_RD              : out STD_LOGIC;          -- ordre de lecture
        COM_WR              : out STD_LOGIC;          -- ordre d'écriture
        B_DATA              : inout STD_LOGIC_VECTOR (7 downto 0)  -- data bus
        );
end SX_FPBAND1;
```

Lors de la réalisation du redresseur mono alternance, nous avions vu que la structure du programme contenait un process dans lequel on pouvait implanter n'importe quel traitement sur le signal d'entrée sans modifier le reste du programme. C'est ce qui est fait ici en implantant l'équation d'un filtre du premier ordre à la place du redressement. On utilise directement les opérations arithmétiques fournies dans les librairies. Ceci a l'avantage d'être très compact au niveau du code mais présente l'inconvénient de générer une architecture assez conséquente lors du routage. Nous verrons par la suite que pour réaliser des filtres d'ordre plus élevé, il nous faudra utiliser une stratégie différente.

Equation d'un filtre du premier ordre :

$y(n) = x(n) + coeff * y(n-1)$

La valeur du coefficient est 0.5 ce qui donne codé en binaire format Q8 la valeur : 01000000.

Principe du filtre :

```
                    y(n - 1)          . . . . . .        (8 bits)
                    coeff.            . . . . . .        (8 bits)
                  ---------------------------------
                  résultat            . . . . . .        (16 bits)

          x(n) étendu     +    x(7)&x(n)&0000000   (16 bits)
                  ---------------------------------
                                Résultat_final       (16 bits)

                  y(n) <= Résultat_final( bits 14 à 7)     (8 bits)
```

Télécharger le source du programme, cliquer ici.