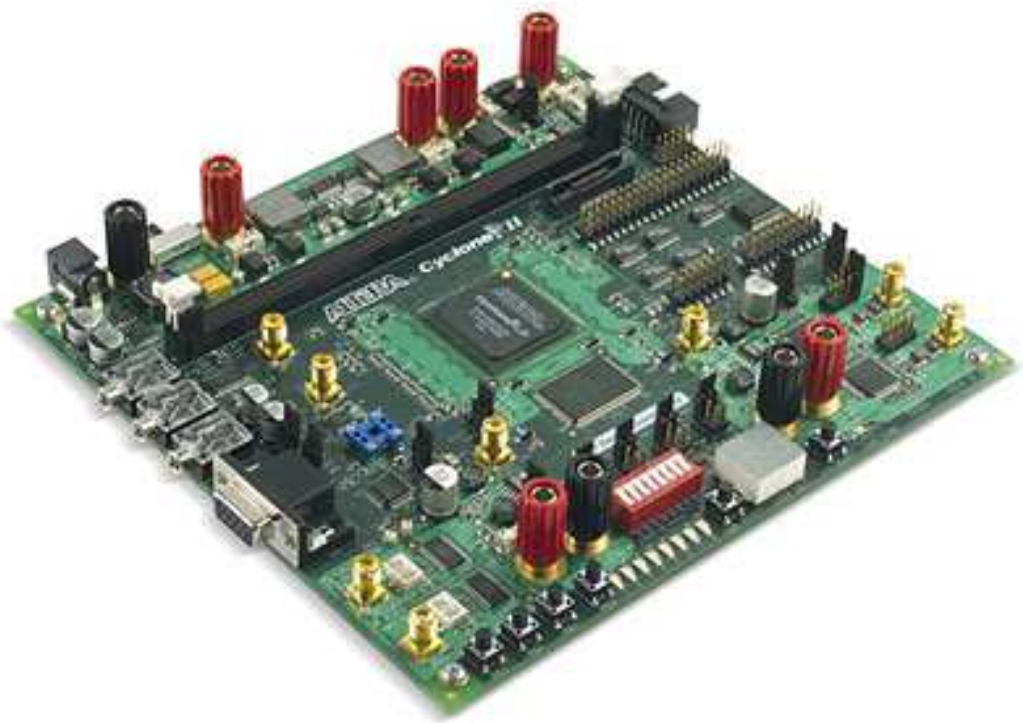
Enseignant: Abraham Suissa

Spécialité: Architecture et Conception des Systèmes Intégrés

MS223: SYSTEMES INTEGRES PROGRAMMABLES

Projet - Mars/Avril 2008

Oscilloscope numérique sur une carte de développement DSP d'Altera



BASSET Sarah
CHAMPION Cédric
IDBAHMANE Soufiane

BRAKNI Yahia
HONNET Cédric
LEROY Cédric

0) Introduction

L'objectif du module Systèmes Intégrés Programmables est tout d'abord de nous introduire au Codesign. Nous aurons l'occasion de nous perfectionner dans l'utilisation des langages de programmation tels que le VHDL et le C.

Enfin, nous approfondirons nos connaissances théoriques sur les principes de simulation en nous familiarisant avec la plateforme de développement Quartus d'Altera pour les implantations de nos propres IP sur FPGA.

Le projet oscilloscope numérique nous permet donc d'aborder tous ces sujets, de développer nos esprits d'initiative et d'équipe ainsi que nos capacités d'organisation. Comme nous travaillons à six sur le projet nous avons sous divisé l'équipe en binôme ou trinômes variables, sur des parties séparées, mais dépendantes, puis nous avons recoupé le fruit de nos recherches et de notre travail.

Dans ce projet, nous allons concevoir, à l'aide d'une carte de développement DSP Altera, un système réalisant un oscilloscope numérique ainsi qu'un générateur de signaux (ou NCO pour Numerically-Controlled Oscillator) le tout commandé par un microprocesseur Nios II et relié grâce au bus Avalon.

La carte de développement possède entre autres ses propres convertisseurs analogique-numérique et numérique-analogique que ce soit pour l'acquisition du signal à afficher (pour l'oscilloscope), les couleurs à afficher sur l'écran VGA ou pour la création de signaux analogiques (pour le NCO).

Le principe de l'oscilloscope numérique à réaliser est d'afficher en temps réel sur un écran VGA un signal analogique acquis grâce au convertisseur analogique-numérique.

La fonction NCO génère un signal numérique dont l'allure est stockée dans une mémoire ROM. Le signal numérique est ensuite converti grâce à un DAC (Digital-to-Analog Converter) en un signal analogique.

Le rôle du microprocesseur Nios II, pour la partie oscilloscope numérique, est de gérer l'amplitude du signal à afficher, la base de temps, et le mode d'affichage (simple ou double entrée). Pour la partie NCO, le Nios II commande la forme du signal ("ground", sinus, carré ou triangle) ainsi que sa fréquence. Le Nios II a donc la charge de l'Interface Homme-Machine (IHM) qui permet l'utilisation de boutons et de l'affichage 7 segments de la carte de développement.

La conception de l'oscilloscope numérique se décompose donc en quatre étapes:

Dans un premier temps, à l'aide de SOPC builder, nous développerons un microprocesseur sur FPGA, pour permettre le contrôle désiré et surtout aborder les techniques de Codesign. Le Nios doit pouvoir faire l'acquisition des ordres donnés par les boutons poussoirs de la carte et répercuter les commandes appropriées à l'oscilloscope ou au NCO.

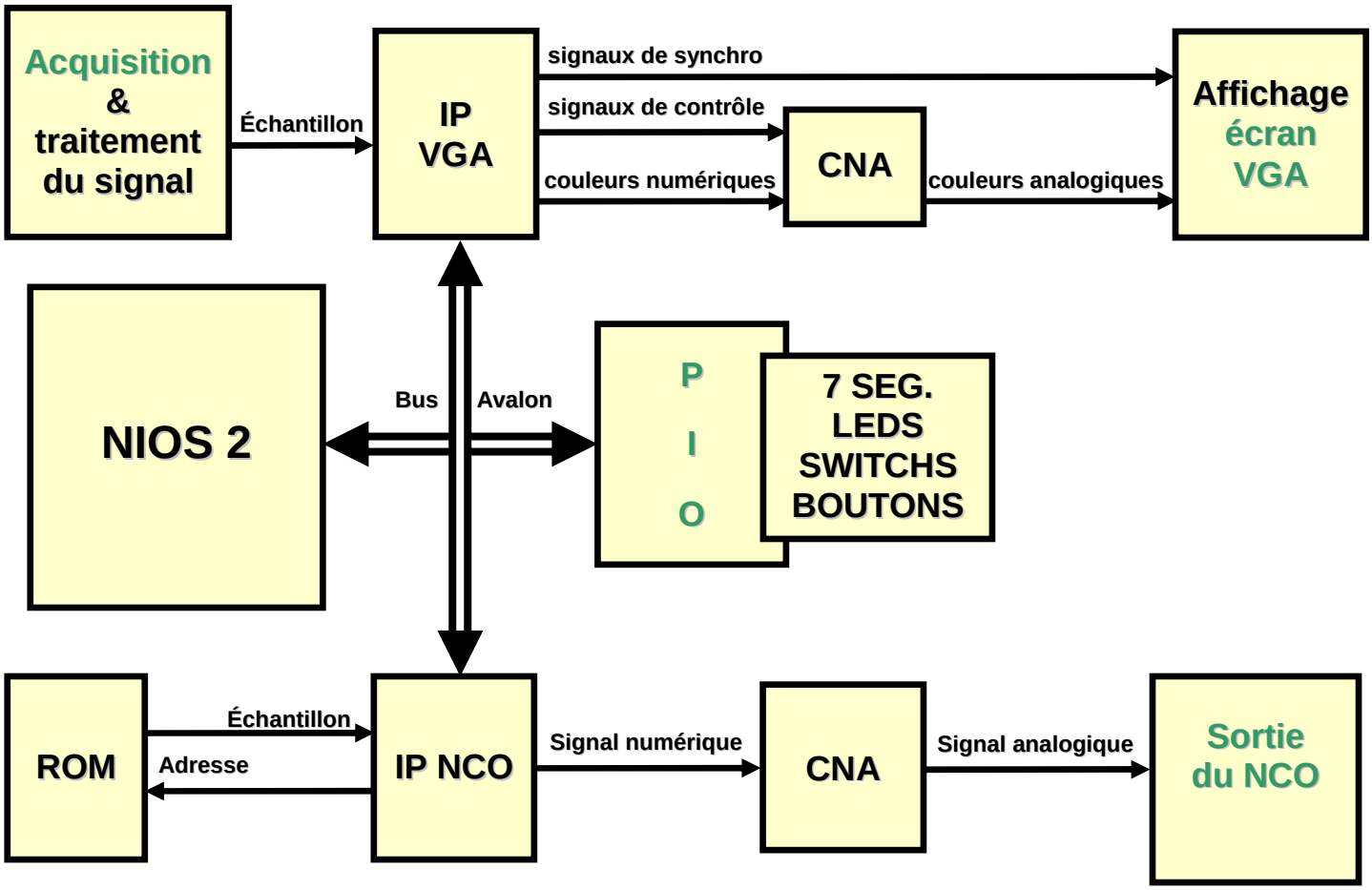
Dans un deuxième temps, nous concevrons une IP de contrôle VGA chargée d'utiliser le triple convertisseur intégré dans la carte pour afficher une image de résolution 640×480 sur un moniteur.

Après cela, nous nous pencherons sur l'acquisition et le traitement d'un signal analogique pour permettre son affichage sur un écran à l'aide du contrôleur VGA précédemment développé.

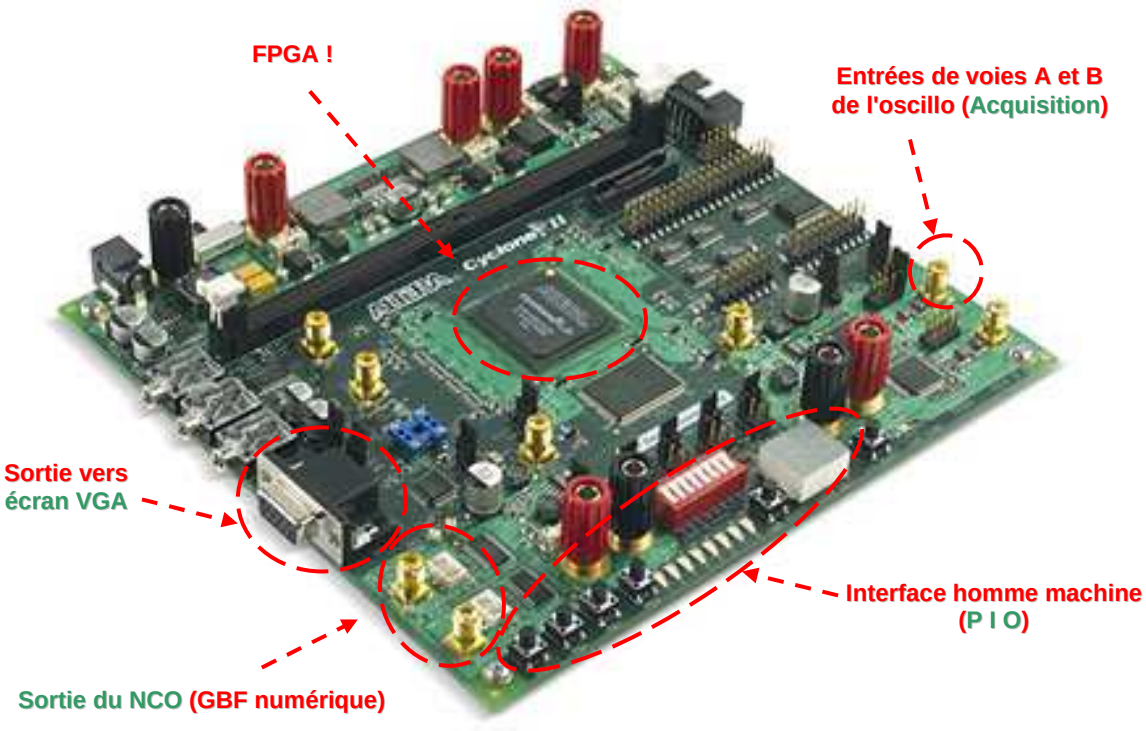
Nous aborderons ensuite la partie NCO dont le rôle sera de lire en boucle une mémoire ROM où l'on aura préalablement stocké les amplitudes des signaux évoqués.

Enfin, nous assemblerons le tout à l'aide du bus Avalon, qui permettra une communication synchrone et ordonnancée de la globalité des fonctionnalités conçues.

Le schéma ci-dessous résume un peu plus précisément le projet :

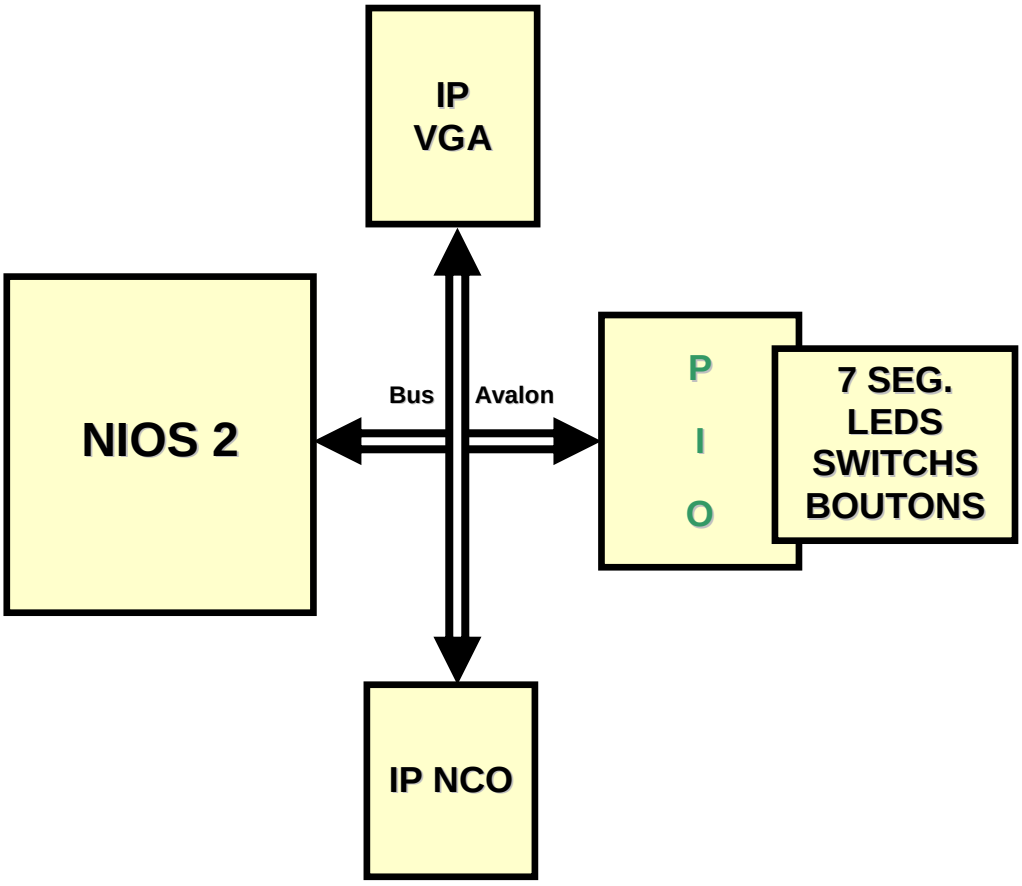


Ce qui se concrétise sur la carte de la façon suivante :



I) Tache 1 : Système NIOS II

Le système NIOS II doit donc permettre une communication entre les différents composants du projet. On représente ses interactions par le schéma suivant:



Le développement du NIOS II se déroule en 2 parties, une partie matérielle et une partie logicielle:

- *Partie matérielle*

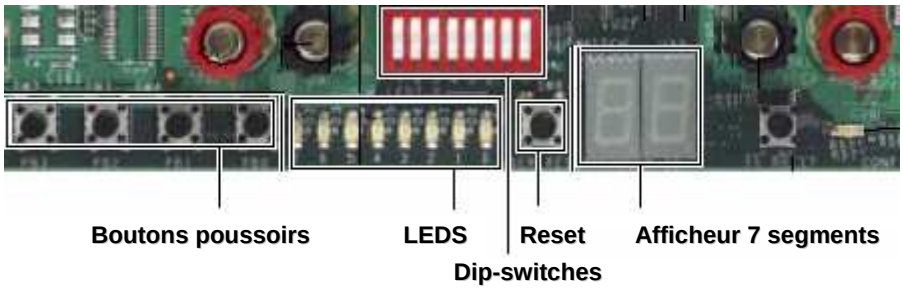
Nous avons utilisé l'exemple d'une IP de microprocesseur Nios II fourni par Altera que nous avons modifié pour répondre au cahier des charges. A l'aide de l'outil SOPC Builder nous avons intégré dans l'architecture initiale les ressources et les entrées-sorties nécessaires pour commander l'oscilloscope et le NCO.

Dans un premier temps, sans l'interface de communication Avalon, nous avons ajouté d'une part les commandes pour piloter la fréquence et la forme du signal issu du NCO et d'autre part celles modifiant l'amplitude et la base de temps du signal affiché par l'oscilloscope.

Dans un deuxième temps, lorsque nous utiliserons le bus Avalon, ces commandes seront modifiées. On écrira alors directement sur le bus qui se chargera de véhiculer les informations.

L'exemple fournit par Altera étant assez complet, nous n'avons pas rencontré de réelles difficultés à concevoir cette IP, une fois l'outil SOPC Builder pris en main.

Interface homme machine devant être gérée par les parties matérielles et logicielles:



- *Partie logicielle*

Le programme pilote_nios écrit en langage C est utilisé par le NIOS afin de piloter des fonctions du NCO (la fréquence et la forme du signal délivré) ainsi que des fonctions de l'oscilloscope (la base de temps et l'amplitude de l'affichage ainsi que mode simple ou double entrée). Un des switchs de la carte permet de choisir si l'on veut agir sur le NCO ou sur l'oscilloscope, un autre permet de choisir le mode simple ou double, puis on modifie les différent paramètres précités en utilisant les boutons poussoirs. Les valeurs correspondantes à ces paramètres sont visibles sur l'afficheur sept segments.

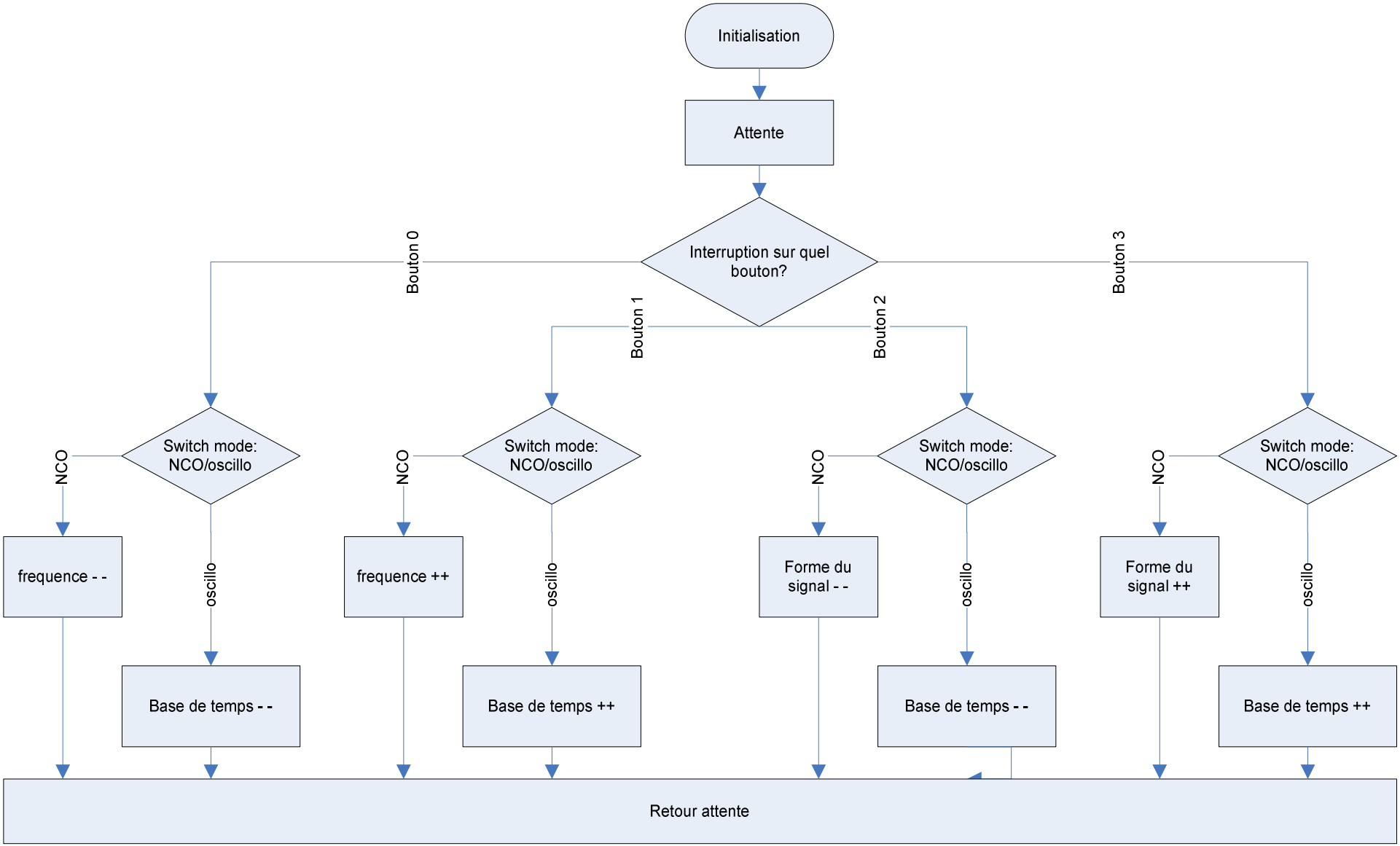
Le bus Avalon est utilisé pour lire les informations des boutons poussoirs et des switchs mais aussi pour transmettre ces informations aux IP concernées (nous détailleront cet aspect en tache 4).

Nous avons utilisé pour écrire ce programme des exemples développés par Altera. Il a donc fallu dans un premier temps comprendre puis apprendre à utiliser et modifier les différentes fonctions :

- Lire les boutons-poussoirs et les switchs
- Ecrire sur le 7 segments et les leds

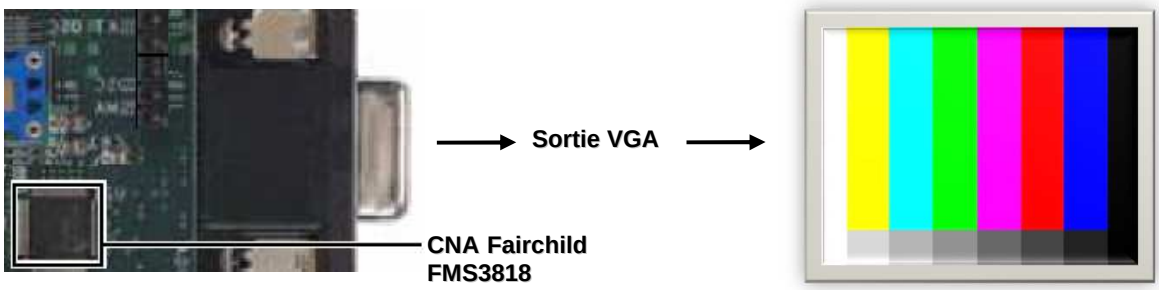
Pour lire les informations des boutons-poussoirs il faut utiliser des interruptions, la difficulté résidait donc dans la gestion de celles-ci. Pour cela nous avons utilisé une fonction développée par Altera que nous avons adapté à nos contraintes. Après chaque modification des commandes, il nous fallait remettre à zéro le drapeau de capture de front du bouton-poussoir, sans cela on ne sortait pas du programme d'interruption. Et par conséquent les commandes étaient modifiées continuellement.

Note: Vous trouverez le descriptif graphique du système logiciel à la page suivante.

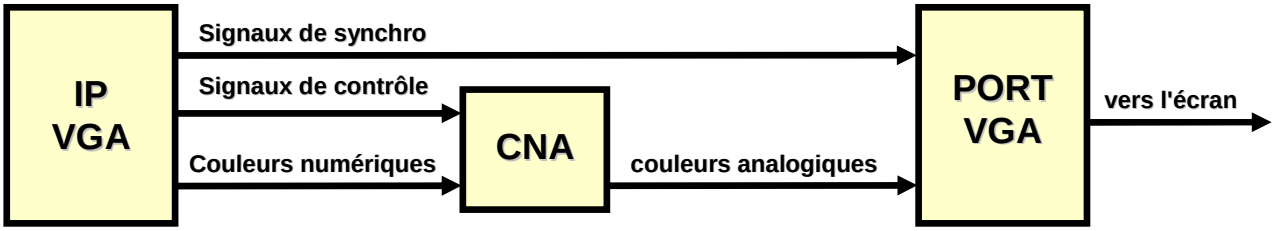

Initialisation
Attente
Interruption sur quel bouton?
Bouton 0
Bouton 1
Bouton 2
Bouton 3
Switch mode: NCO/oscillo
NCO
oscillo
frequence - -
Base de temps - -
frequence ++
Base de temps ++
Forme du signal - -
Base de temps - -
Forme du signal ++
Base de temps ++
Retour attente

II) Tache 2: IP VGA - GENERATEUR DE MIRE

Dans cette partie, nous allons nous intéresser à la conception d'une IP de contrôleur VGA. La carte de développement dont nous disposons est munie d'un Convertisseur Numérique Analogique (Fairchild FMS3818) relié à un connecteur DB15 pour brancher l'écran VGA:



L'objectif de est donc de concevoir un élément capable d'utiliser le convertisseur intégré et de générer les signaux de contrôle conformes à la "norme VGA" pour afficher une mire en résolution 640x480 sur un moniteur:



Les signaux de contrôle générés par le FPGA sont majoritairement reliés directement aux broches du CAN et au connecteur DB15 pour brancher l'écran VGA. Pour construire ces signaux nous avons d'abord mis en place une structure, qui permet de considérer un écran comme une matrice de 800 colonnes par 525 lignes. Seules les 640 premières colonnes et les 480 premières lignes sont affichées, le reste permet virtuellement au pointeur d'avoir le temps de retourner à la ligne suivante ou en haut de l'écran quand il a été entièrement parcouru (contrainte imposée initialement pour les tubes cathodiques).

La norme VGA exige 3 signaux analogiques pour les couleurs (RGB) et nous utiliserons 2 signaux de synchronisation (un signal de synchronisation horizontale et un signal de synchronisation verticale) mais il est possible de n'utiliser qu'un seul signal de synchronisation composite. Nous utiliserons aussi des signaux de contrôle pour désactiver le convertisseur pendant le retour du pointeur.

Nous nous sommes donc attachés dans un premier temps à construire ces signaux de synchronisation. La norme VGA n'étant en réalité pas une norme très fixe, nous avons donc élaboré le plus intelligemment possible les timings adaptés à notre application.

La majorité des signaux générés par le FPGA sont reliés directement aux broches du CAN mais la synchronisation, aussi générée par le FPGA, est directement envoyée à l'écran VGA par les broches du connecteur DB15.

Quelques détails sur la construction de la structure évoquée précédemment:

1) timing **horizontal**: (1 pixel dure 40ns @ 25MHz)

-Signal de *désactivation* horizontale (BLANK_H):

-Signal de *synchronisation* horizontale (VGA_HSYNC):

2) timing **vertical**: (1 ligne dure 800 pixels et 1 écran dure 525 lignes)

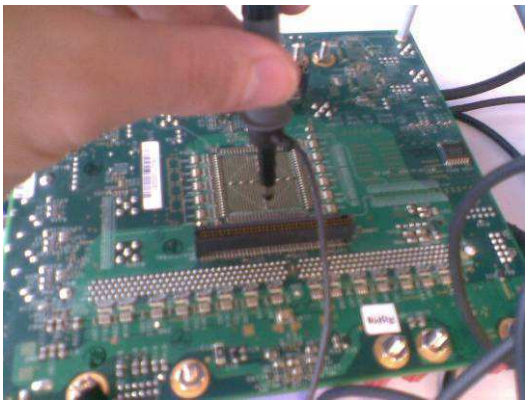
-Signal de *désactivation* verticale (BLANK_V):

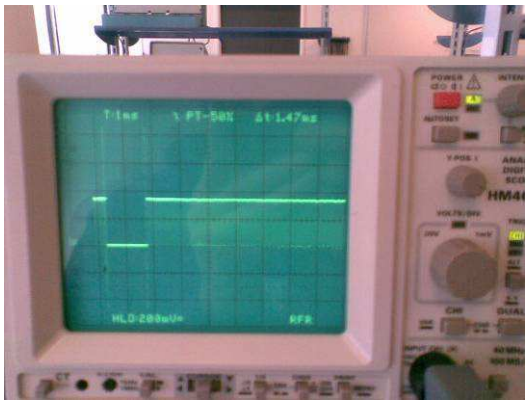
-Signal de *synchronisation* verticale (VGA_VSYNC):

Note: 525 lignes × 800 pixels × 40 ns/pixel = 16.8 ms par image ≈ 60 image par secondes (=60Hz)

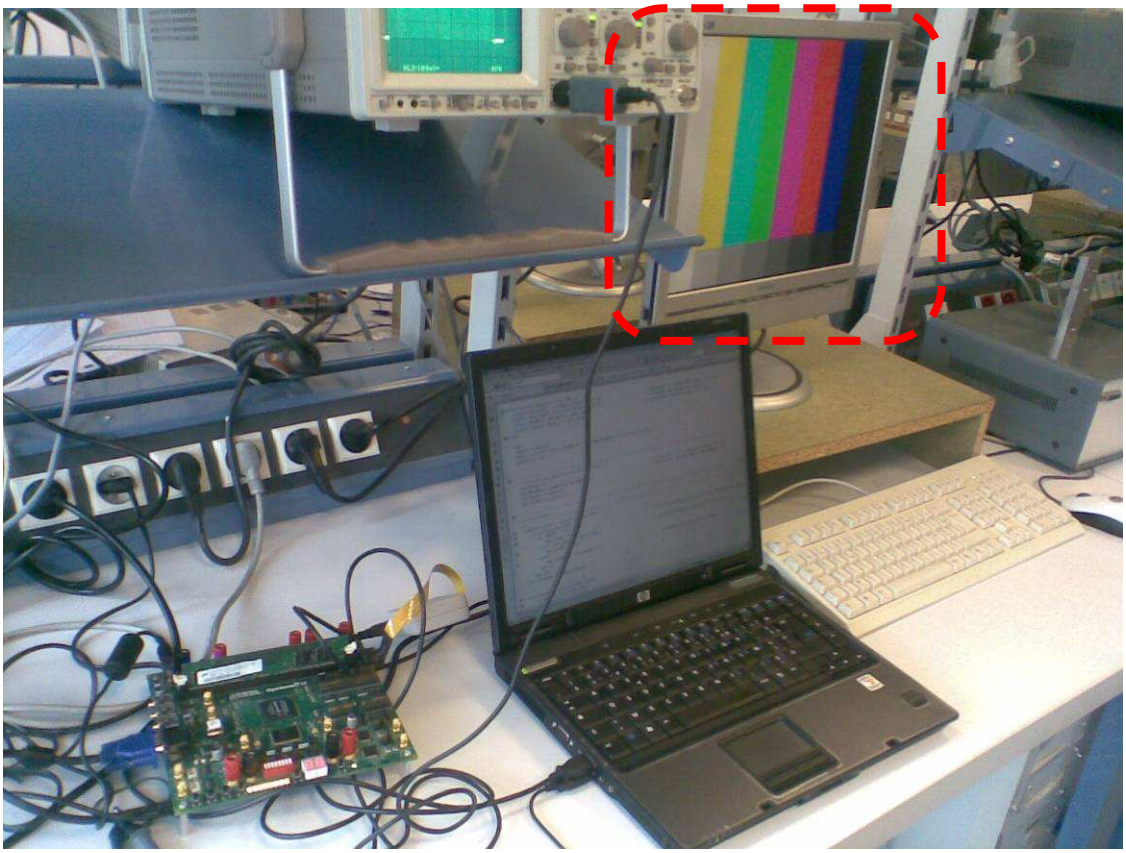
Nous avons rencontrés quelques difficultés dues à l'assignement d'un signal (parmi plus d'une trentaine que nous n'avons pas détaillés car non significatifs). À cause de cette affreuse erreur d'inattention, l'affichage ne pouvait fonctionner.

Vu que les chronogrammes de simulation étaient irréprochables nous avons décidé de sonder et mesurer chaque signal de sortie concerné au dos du FPGA:



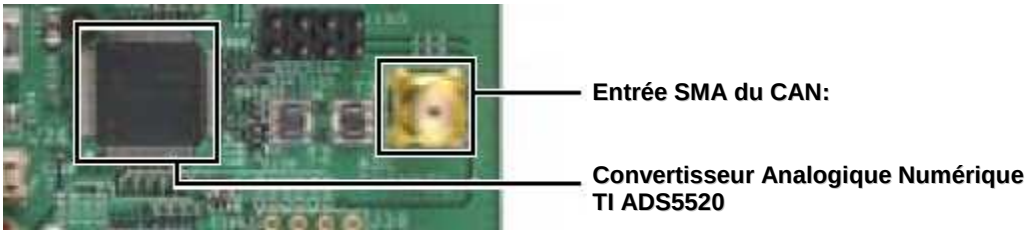


Nous avons donc fini par débusquer le problème, mais il est certain que même si c'est entièrement de ma faute, avec des accès libres aux cartes de développements pendant les périodes de TP, nous aurions certainement évité de perdre au moins 2 séances.



III) Tache 3: Oscilloscope numérique

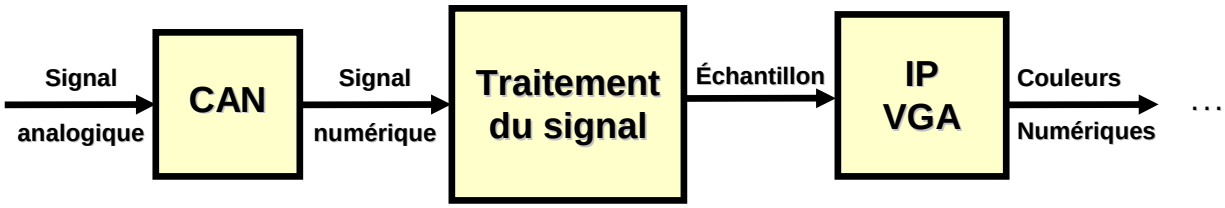
Cette IP est le coeur du projet, elle réunit les IP Acquisition et VGA modifiée pour permettre de faire l'acquisition et le traitement numérique d'un signal analogique (issu du NCO ou non) puis de l'afficher. L'acquisition se fait donc à l'aide du convertisseur Texas Instrument suivant:



Après échantillonnage par le CAN, l'IP Acquisition stocke les échantillons du signal d'entrée dans une mémoire RAM, que l'IP VGA lit puis affiche à l'écran.

Nous avons utilisé une mémoire RAM qui accepte deux horloges différentes, une pour l'écriture des valeurs du signal échantillonné et une plus lente pour leur lecture par l'IP VGA, car la "norme" VGA nous impose une fréquence de 25 Mhz.

On note que le CAN échantillonne à une fréquence constante, mais que la mémoire RAM, elle, mémorise les échantillons à une fréquence variable. C'est cette mémorisation à vitesse variable qui, indirectement, permet la variation de la base de temps de l'affichage.



Le principe de traitement du signal numérique récupéré est de comparer l'amplitude qu'il représente à l'abscisse instantanée de la position du pointeur.

Ce pointeur balayant l'écran pixel par pixel (horizontalement), on effectue cette comparaison à chaque pixel et ce pour toutes les lignes.

L'écran ayant une définition de 640 pixels de large, nous avons choisi une RAM de la même taille pour y stocker nos échantillons (représentés sur 12 bits chacun).

Si les coordonnée du point correspondent bien à l'amplitude et à la position temporelle à représenter, on affiche un point vert (pour la voie A et bleu pour la voie B) et le reste de l'écran en noir (sauf pour les axes de l'écran qui sont rouges).

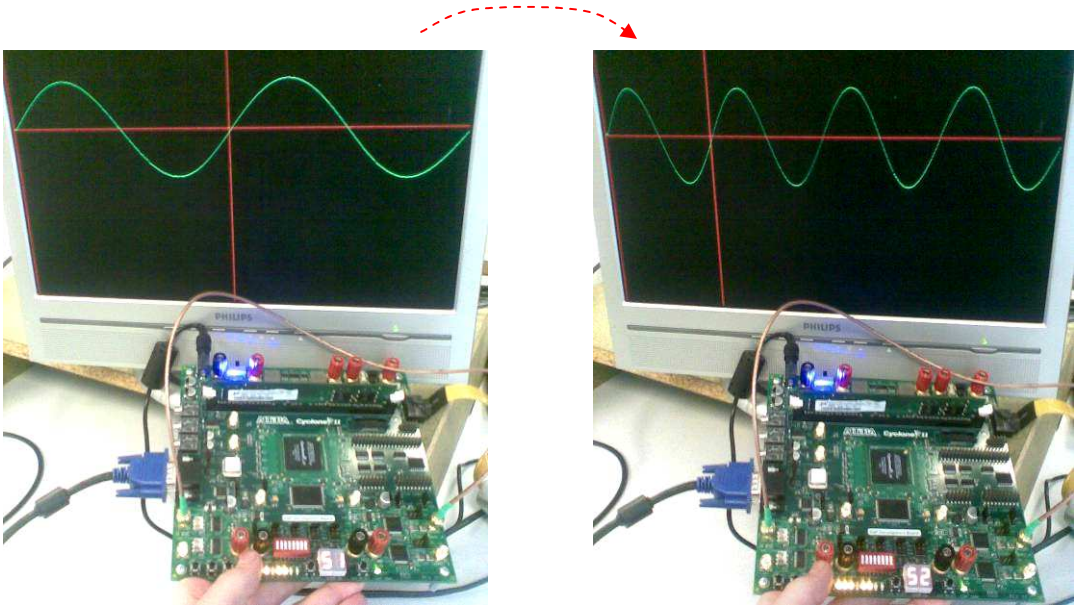
Nous avons jugé intéressant de représenter notre signal dans un repère temporel dynamique comme dans Matlab pour la représentation des zooms avant et arrière (tout comme dans une carte routière consultable en ligne par exemple).

Le principe est de considérer que la base de temps limitante est celle à laquelle la "fréquence d'échantillonnage" est maximum.

Si l'on "échantillonne" à une fréquence plus faible, l'effet visuel ressenti correspond à un zoom arrière. Le cadre temporel représenté par un axe vertical rouge va donc se resserrer pour marquer cet effet de zoom et bien sûr aussi pour permettre de mesurer la fréquence de notre signal.

En effet, cette largeur variable correspond à une durée fixe malgré les changements de base de temps de l'affichage:

Division par 2 de la fréquence d'échantillonnage
(ou base de temps ×2)



Pour faciliter les mesures de temps sur nos signaux nous avons conçu le tableau suivant, dans lequel les calculs ont été automatisés à l'aide d'un tableur (d'où les valeurs précises):

selec_freq (lisible sur le 7 segment)	**F_e**= fréquence d'échantillonnage (MHz)	**T_e** (ns)	**T_{640}** = durée de 640 samples (µs)	**F_{640}** = fréquence d'une sinusoïde entièrement affichable pendant T_{640} (kHz)	**N_p** = nombre de pixels par divisions de l'écran (taille)
000 = 0	100	10	6,4	156,25	640
001 = 1	50	20	12,8	78,125	320
010 = 2	25	40	25,6	39,0625	160
011 = 3	12,5	80	51,2	19,53125	80
100 = 4	6,25	160	102,4	9,765625	40
101 = 5	3,125	320	204,8	4,8828125	20
110 = 6	1,5625	640	409,6	2,44140625	10
111 = 7	0,78125	1280	819,2	1,220703125	5

$F_e = 100\ MHz / 2^{freq\_selec}$
$T_e = 1 / F_e$
$T_{640} = 640\ T_e$ (temps correspondant à la durée des 640 samples affichés = l'écran entier)
$F_{640} = 1 / T_{640}$
$N_p = 640 / 2^{freq\_selec}$

Comme on a pu le voir, l'aspect fractionnaire de ces valeurs n'aide pas à la mesure de nos signaux, nous avons donc aussi conçu une version plus "lente", consistant à utiliser une PLL à 80MHz au lieu de 100MHz.
L'intérêt des 80MHz est évident lorsque l'on compare le tableau précédent avec celui ci:

selec_freq	F_e= fréquence d'échantillonnage (MHz)	T_e (ns)	T_{640} = durée de 640 samples (µs)	F_{640} = fréquence d'une sinusoïde entièrement affichable pendant T_{640} (kHz)	N_p	durée d'une division d'écran par 8 (µs)
000 = 0	80	12,5	8	125	640	1
001 = 1	40	25	16	62,5	320	2
010 = 2	20	50	32	31,25	160	4
011 = 3	10	100	64	15,625	80	8
100 = 4	5	200	128	7,8125	40	16
101 = 5	2,5	400	256	3,90625	20	32
110 = 6	1,25	800	512	1,953125	10	64
111 = 7	0,625	1600	1024	0,9765625	5	128

Nous aurions pu en faire encore plus avec les PLL mais nous avons choisi de faire 8 bases de temps différentes (les 4 PLL disponibles sont de toute façon monopolisées par le Nios). Le problème lorsque l'on atteint des fréquences d'échantillonnage trop basses, on peut finir d'écrire les derniers échantillons dans la RAM alors qu'on est en train de les lire pendant l'affichage (ce qui donne un résultat visuel complètement indigeste).
Nous avons donc utilisé le signal de désactivation d'affichage de fin d'écran pour autoriser un éventuel déclenchement de trigger.

L'allure du signal analogique est mise à jour seulement toutes les 16.8 ms ≈ 60 fois par secondes (dû au taux de rafraîchissement de 60Hz expliqué en tache 2.)

Ce n'est pas un problème dans le sens où l'on ne veut qu'observer un signal en temps réel, mais si on voulait l'enregistrer pour une écoute audio par exemple, on aurait eu besoin d'une RAM pour l'écriture permanente et une autre qu'on aurait rechargé à 60Hz pour la représentation visuelle.

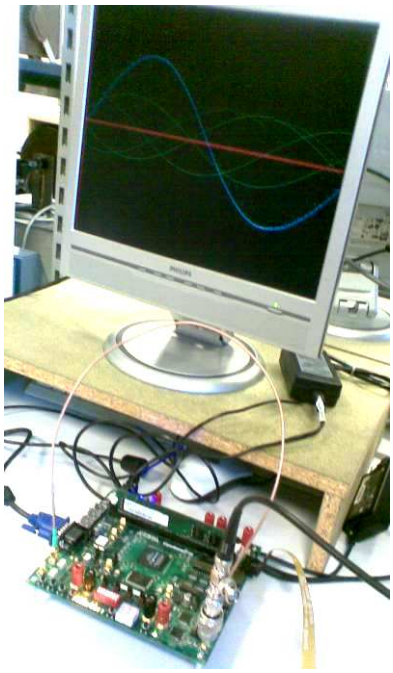
Ce signal d'autorisation d'écriture nous a donc été indispensable pour permettre un affichage stable. En effet, nous avons dû faire un trigger pour "immobiliser" l'affichage, le principe de détection de front montant est assez simple mais il fallait penser au glitchs.

On mémorise en permanence 3 échantillons successifs, si les 2 plus anciens sont négatifs et le dernier est positif, on a de grandes chances d'avoir détecté un front montant.

Nous avons aussi pensé à effectuer un filtrage "moyenneur" sur ces 3 échantillons mais selon l'amplitude du glitch, le résultat n'était pas convainquant.

Le problème le plus agaçant de cette tache fut encore une erreur d'inattention. La variable stockant la valeur de l'échantillon (normalisé) à comparer à l'abscisse instantanée du pointeur était un entier.

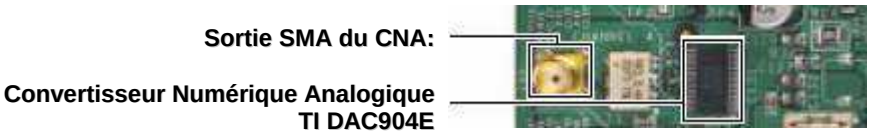
La déclaration de cet entier était bornée entre des valeurs bien trop petites (0 à 480 pour la hauteur de l'écran alors que les amplitudes peuvent atteindre 4095) ce qui faisait un banal overflow mais incompréhensible sur le moment.



Cas avec une double entrée
(Les 2 signaux sont déphasés)

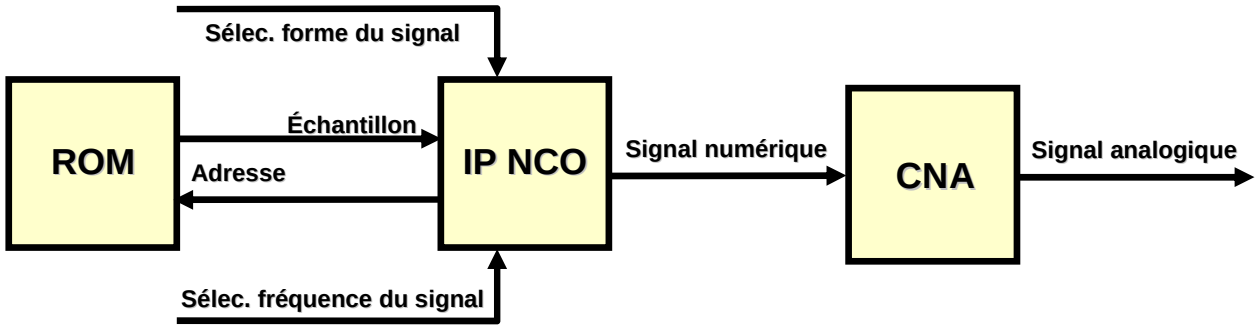
IV) Tache 4: NCO

Dans cette partie nous allons nous intéresser à la conception d'un "GBF numérique" qui va donc pouvoir générer, selon notre choix, des signaux périodiques de forme sinusoïdale, carrée ou triangulaire à des fréquences sélectionnables (on va aussi pouvoir générer un "ground" mais on n'insistera pas sur ce détail).



Le NCO (Numerically Controlled Oscillator) va utiliser des données stockées dans une mémoire ROM, qui ne sont rien d'autre que des valeurs d'amplitudes numériques (ou échantillons).

Ces valeurs, lues à une cadence suffisante, représentent notre signal numérique qui sera converti grâce au CAN selon le schéma suivant :



Principe de fonctionnement de l'IP NCO :

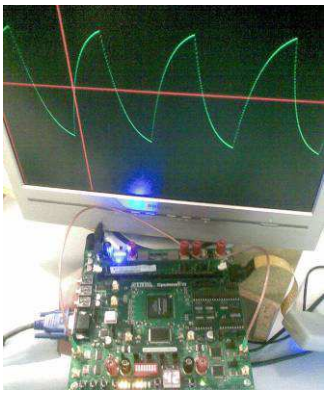
On va stocker 640 échantillons, représentant une période, par forme de signal dans la ROM puisque la définition de notre affichage est de 640 pixels de large.

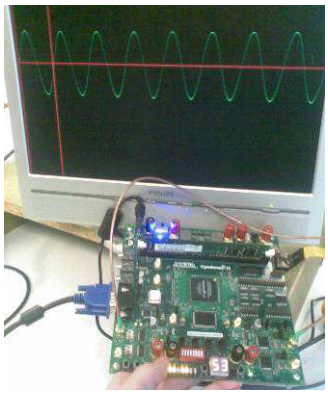
On utilise donc un compteur allant de 0 à 640 qui va demander à la ROM les données successives nécessaires. A chaque front montant de l'horloge sélectionnée, on incrémente ce compteur qui désignera l'adresse de l'échantillon désiré.

Pour choisir la forme sinusoïdale, carrée ou triangulaire on utilisera un offset permettant de balayer la mémoire dans la zone dédiée au signal sélectionné.

Pour charger les échantillons dans la mémoire on a généré les valeurs des signaux sous Matlab qui nous ont permis de faire un fichier d'initialisation de la ROM.

Exemple avec un signal triangulaire





Exemple avec un signal sinusoïdal

Ce "GBF numérique" possède certainement un problème majeur: l'adaptation d'impédance. En effet, selon la fréquence du signal généré, son allure est modifiée par des effets capacitifs, inductifs et résistifs parasite (certainement dus conections). Le peu de temps nous étant accordé pour effectuer des tests sur carte ne nous a pas permis de trouver de solution à ce problème.

V) Tache 5: Bus Avalon

Le bus Avalon est un bus conçu pour permettre, à des périphériques (ou dispositifs logiques) développés sur des SOPC (systèmes sur puces programmables), d'assurer une communication entre IP.

A travers une structure d'interconnections de systèmes propriétaire, ce dernier se charge de transférer les informations aux éléments concernés.

Ce bus a la particularité de mettre en liaison des périphériques de types master et slave sans connaître à priori la structure interne de ces derniers.

De plus en fonction du type de transferts que l'on souhaite réaliser, ce bus est d'une certaine façon configurable en se limitant à certains signaux types prédéfinis tels que « *ADDRESS* », « *DATA* », « *WRITE* »...

En effet, on peut remarquer que documentation fournie par Altera sur l'interface du bus Avalon définie:

- ✓ Un jeu de signaux types
- ✓ Leurs comportements (chronogrammes)
- ✓ Les types de transferts tolérés par ces signaux

On est alors libre d'utiliser ou non ce signal à condition qu'il réponde à nos attentes.

Dans cette section nous devons donc concevoir une IP de contrôle qui assurera la liaison entre le processeur NIOS II et les différentes IP du projet, IP NCO et IP OSC, et ce à partir des différents signaux issue du bus Avalon. Afin de réaliser cette tâche nous nous sommes posés divers questions notamment le rôle de cette IP qui serait simplement de transmettre les données issues du NIOS et de les distribuer aux modules concernés, on est donc dans une opération d'écriture de données tel qu'on aurait dans le cas d'une écriture dans une mémoire.

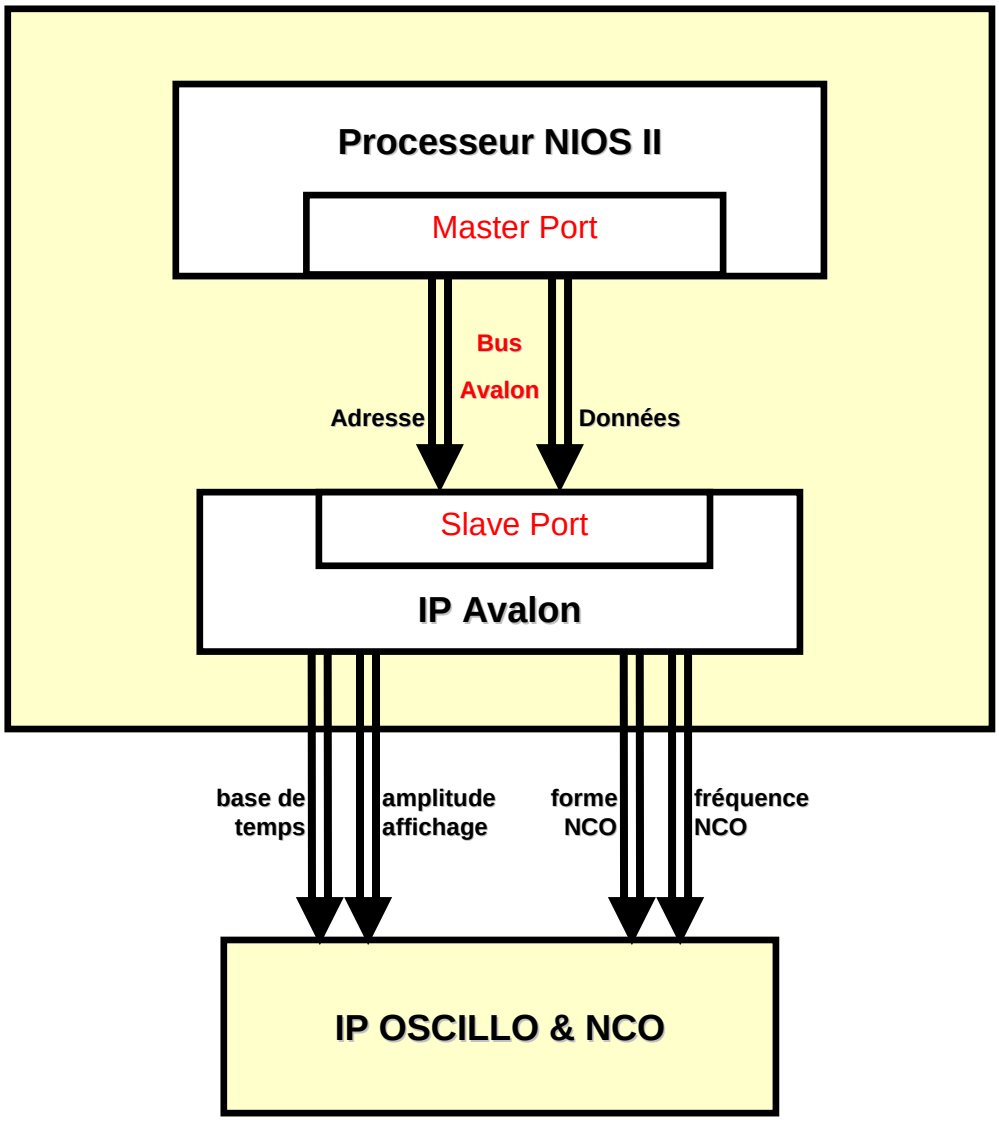
Nous avons donc recensé les informations à transmettre du NIOS II aux autres IP :

- ✓ Forme du signal délivré par le NCO
- ✓ Fréquence du signal délivré par le NCO
- ✓ Amplitude d'affichage de l'oscilloscope.
- ✓ Base de temps de l'oscilloscope.

Puis nous avons alors sélectionné les signaux utiles à ces transferts de données :

- ✓ « *ADRESS* » : signal contenant les adresses des différentes entrées de nos IP de notre projet.
- ✓ « *WRITEDATA* » : signal contenant les différentes données à transmettre décrites ci-dessus.
- ✓ « *WRITE* » : signal autorisant ou non l'écriture de données (synchrone)
- ✓ « *CLK* » : signal d'horloge

On peut alors résumer les explications fournies ci-dessus par le schéma suivant :



Après description comportementale de notre IP_Avalon en langage VHDL nous avons, grâce au SPOC builder, ajouté cette IP au système NIOS II en tant que nouveau composant.

La subtilité réside dans l'utilisation de ce composant, en effet, on lui assigne des valeurs comme on écrirait dans une mémoire. Ce composant ayant sa propre adresse, on utilise un offset permettant d'adresser les consignes désirées aux entrées appropriées des autres IP (NCO et oscillo).

Note: On peut constater que l'on ne représente pas tous les signaux de contrôle comme par exemple `CLK` dans les schémas.

VI) Conclusion

En menant à bien la conception d'un oscilloscope numérique, notre équipe a acquis une première expérience instructive au niveau de la gestion, l'organisation et la réalisation de projet en groupe. Nous avons appris à travailler au sein d'une équipe, à nous repartir les différentes tâches demandées tout en restant à l'écoute de nos coéquipiers pour pouvoir accorder les parties dépendantes entre elles et s'entraider en cas de problème. Comme elle nous a permis d'utiliser des logiciels et langages divers (Matlab, Quartus, VHDL, C), cette approche pratique des travaux de concepteurs de systèmes intégrés nous sera bénéfique à l'avenir, que ce soit dans un premier temps pour nos stages ou bien lors de nos premières expériences professionnelles.

VIII) Bibliographie et documentation:

Carte de développement DSP cyclone II d'Altera :
http://www.altera.com/literature/lit-cyc2.jsp

Microprocesseur NIOS II d'Altera et bus Avalon :
http://www.altera.com/literature/lit-nio2.jsp

Convertisseur VGA : Fairchild FMS3818
http://www.fairchildsemi.com/ds/FM/FMS3818.pdf

Timings VGA :
http://www.epanorama.net/documents/pc/vga_timing.html

DAC : TI DAC904E
http://focus.ti.com/lit/ds/symlink/dac904.pdf

ADC : TI ADS5520
http://focus.ti.com/lit/ds/symlink/ads5520.pdf