Mini-projet

Réalisation d'un filtre FIR (à Réponse Impulsionnelle Finie) en VHDL

Nicolas Sauzède
5GE-SEI

INSA Lyon
2004-2005

Table des matières

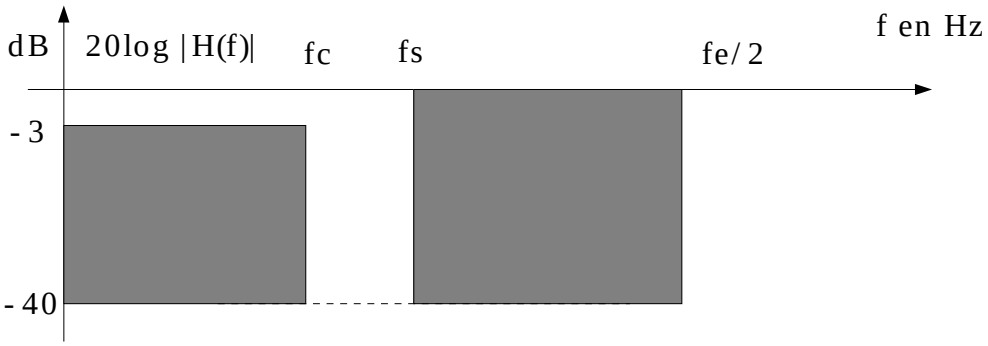
But

Le but de ce mini-projet est de réaliser un Filtre à Réponse Impulsionnelle Finie (FIR) de type passe bas en langage VHDL pour une cible temps-réel de type FPGA.
Ainsi il est possible de tester l'algorithme sur des vrais signaux en temps réel issus par exemple d'applications audio.

Cahier des charges

Les spécifications techniques de ce projet sont :

- fréquence coupure basse fc=20kHz (-3dB)
- fréquence coupure haute fs=60kHz (-40dB)
- fréquence d'échantillonnage fe=1MHz



De plus, la synthèse du filtre nous donne les caractéristiques suivantes :

- 32 coefficients sur 8 bits (codage Q0.8)
- échantillons d'entrée sur 8 bits (codage Q0.8)
- multiplications sur 16 bits (codage Q0.16)
- 32 accumulations sur 21 bits (codage Q5.16)
- échantillons de sortie sur 8 bits (codage Q0.8)

De ces spécifications, il ressort que :

- le FPGA devra acquérir et sortir les données toutes les 1us
- la fréquence de travail du FPGA doit être de 40 MHz minimum
-

La partie implémentation sur cible doit se faire sur une platine de test équipée d'un FPGA modèle FLEX EPF10K10 de chez Altera cadencée à 66MHz.

Réalisation

Ce projet a été réalisé à l'aide des outils suivants :

- partie théorique (synthèse coefs) : **Scilab** + **langage C** pur
- partie VHDL (synthèse code + simulation) : **Simili** de Symphony EDA
- partie implémentation (génération cible FPGA) : Altera Quartus Web

Malheureusement, la partie implémentation n'a pu être aboutie car le code binaire généré par le filtre VHDL dépasse (largement) la capacité du FLEX EPF10K10 en terme de nombre de cellules logiques.

J'ai donc dû me contenter d'une simulation poussée (en C et en VHDL).
La partie en langage C m'a permis de tester intégralement la partie algorithmique pure du calcul de filtrage (c'est à dire sans contrainte de temps/puissance de calcul – calcul linéaire, non parallèle, sur PC).
La partie VHDL simulée avec Simili m'a permis de tester la partie asynchrone, cad dépendante du temps (implémentée sous forme de processes VHDL parallèles).

La comparaison directe des valeurs obtenues par les deux méthodes donne donc une indication très robuste de la validitée de mon travail.

Mise en oeuvre

Pour mettre en oeuvre mon projet, il faut d'abord présenter son arborescence :

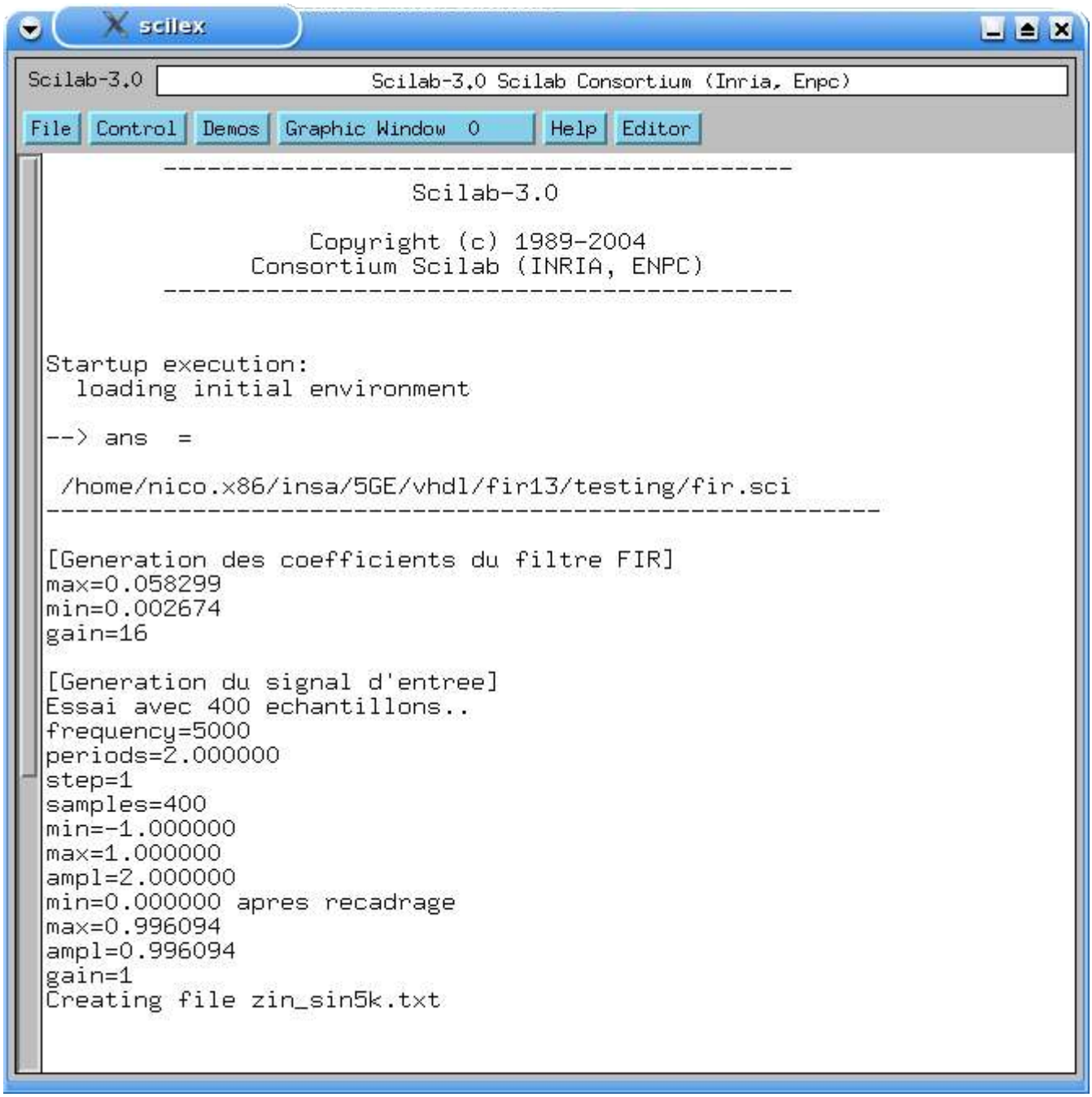
/	-> contient toute la partie VHDL
/testing	-> contient la partie théorique scilab et langage C
/testing/prost	-> débuts de fichiers matlab de Mr Prost (inutilisés)
/report	-> divers documents de documentation/rapport
/fir.sym	-> arborescence générée après compilation VHDL (inutilisée)

Il y a donc trois parties distinctes dans mon projet : une partie (théorique et mathématique) sous Scilab, une partie (algorithmique) en langage C et une partie (implémentation hardware) en VHDL.

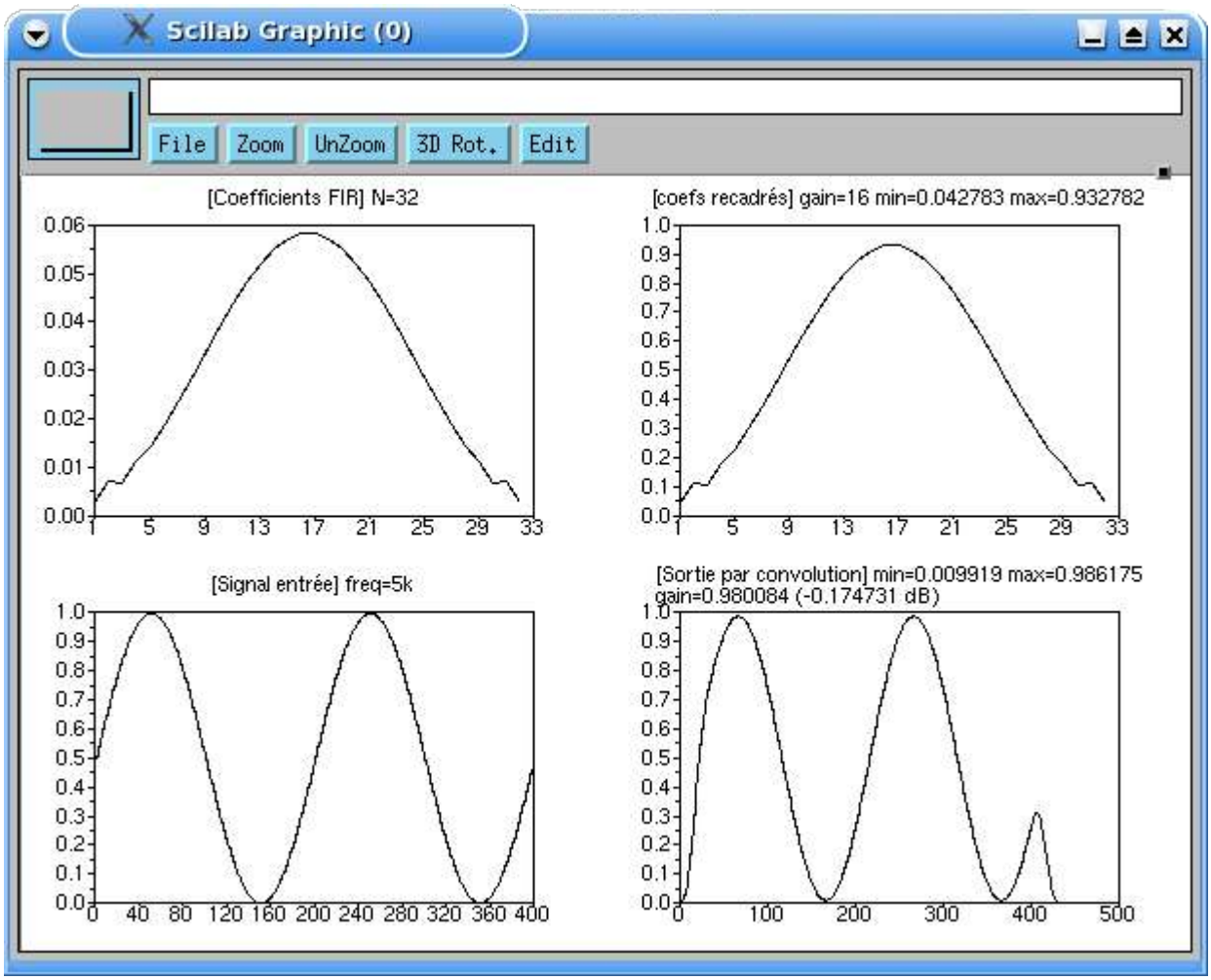
Partie Scilab

- fir.sci . génère les 32 coefficients "sci" (scilab)
  . génère le signal d'entrée (sinus, fréquence réglable)
  . affiche les coefs, le signal d'entrée et de sortie théorique (par convolution)

Résultat dans la console de Scilab après éxecution de fir.sci :

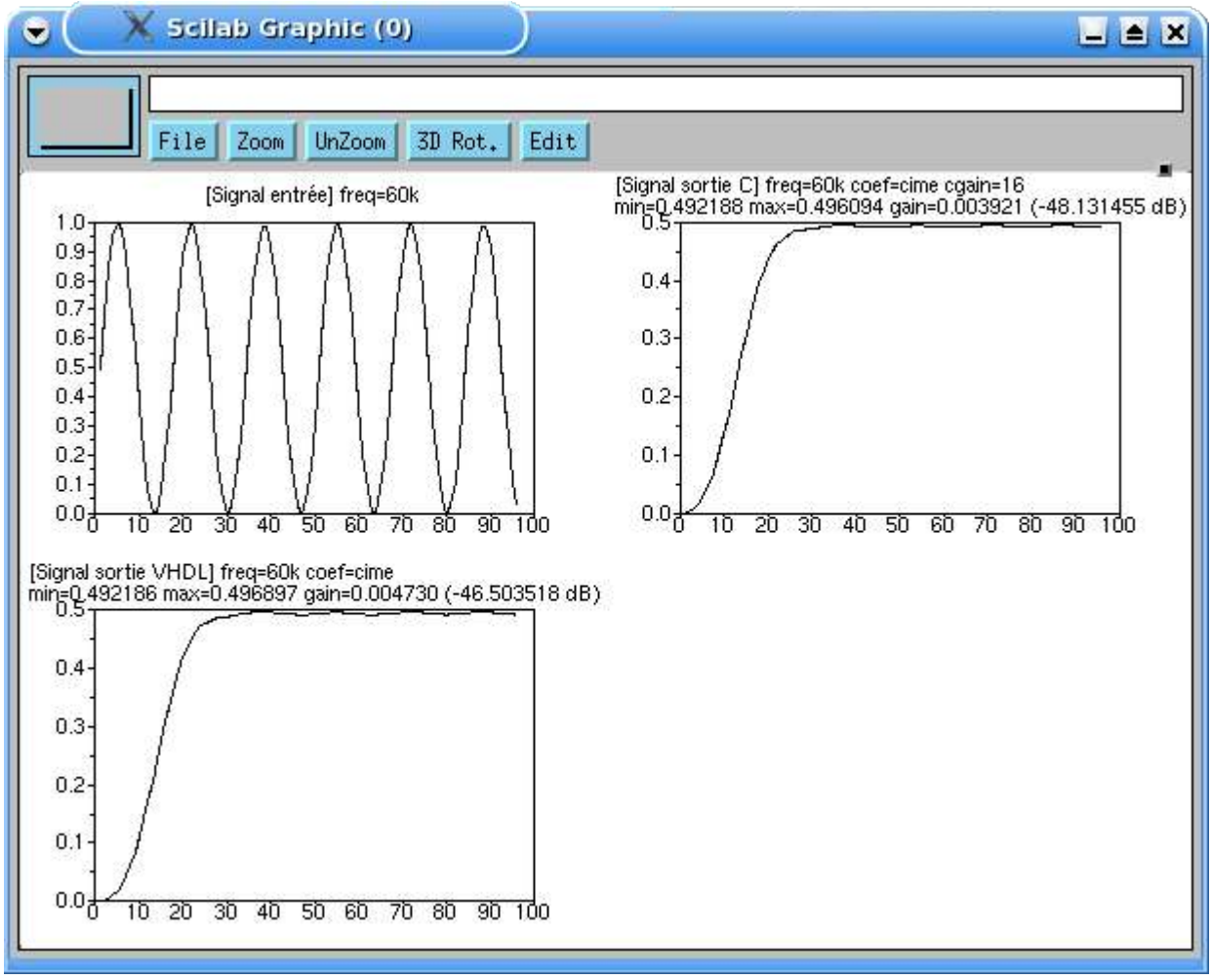


Le script fir.sci affiche également cette fenêtre graphique, qui révèle la forme des coefficients générés (avant et après recadrage), ainsi que le signal d'entrée (sinus) généré également, et enfin le résultat du filtrage théorique équivalent au FIR par une simple convolution.

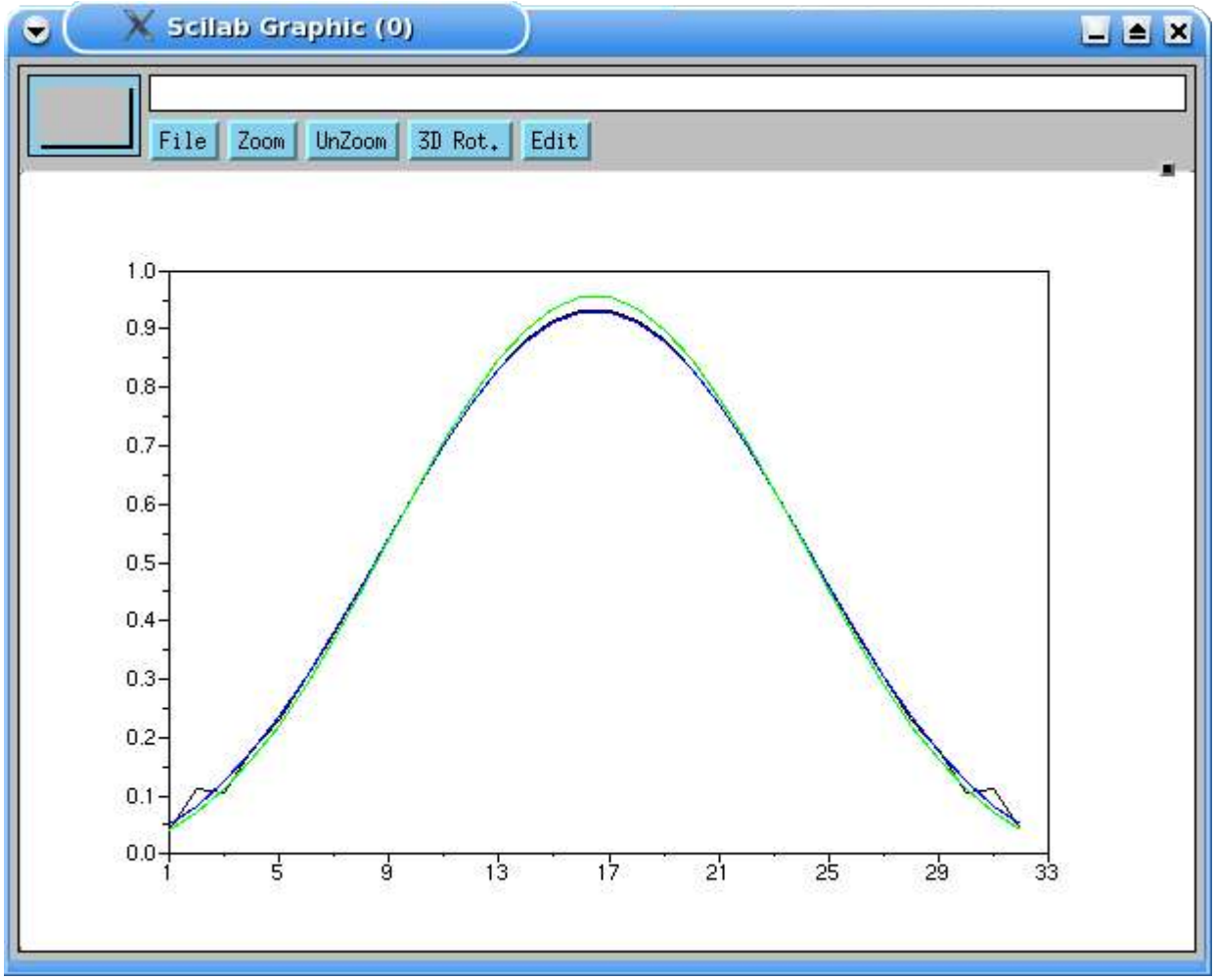


- verif.sci . après exécution du programme C et de la simu VHDL, affiche les résultats (notamment le gain pour la fréquence d'entrée choisie)

Note : la simu VHDL crée un fichier buff.txt, qu'il faut renommer en par example : buff_sci_sin10k.txt selon les coefs/freq choisis (désolé) et ajouté quatre lignes au début du fichier : (désolé²)
sci<retour chariot>
16<retour chariot>
200<retour chariot>
<retour chariot>



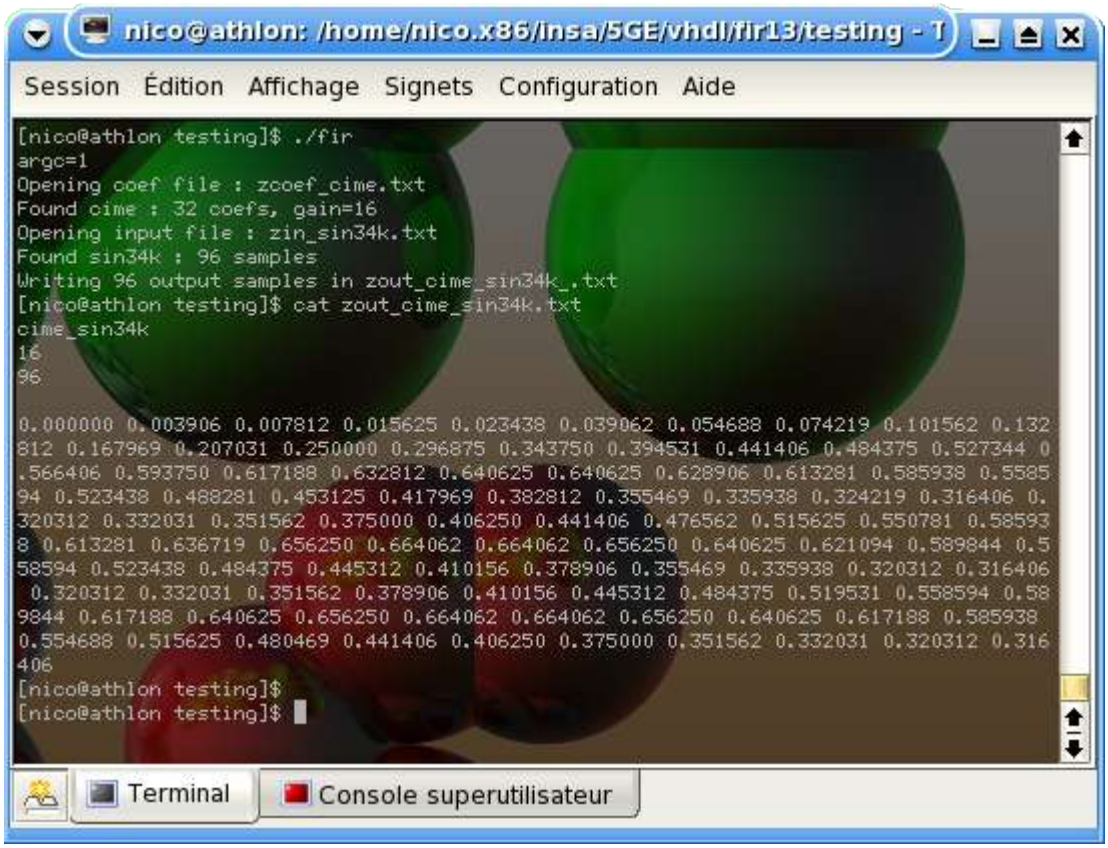
coefs.sci . affiche simultanément les 3 types de coefs : "sci" générés par scilab (cad, par moi :-), "cime" issus du TP CIME (Grenoble), "prost" fournis par Mr Prost



Partie langage C

fir.c . compilé, donne un exécutable : "fir"
. à partir de coefficients (sci, cime, prost) et d'un signal d'entrée (sin10k, sin34k, etc....) génère (par exemple) :
- coefs ROM : zROM_sci.txt
- samples ADC : zADC_sin10k.txt
- échantillons sorties : zout_sci_sin10k.txt

Le fichier de sortie du programme C donne donc le résultat rigoureusement identique (à la précision près) à celui du FIR en VHDL car j'ai programmé l'algo de convolution à la main.
Ceci m'a permis au cours du projet de vérifier petit à petit l'exactitude des différentes parties (théorique, VHDL, ...).



Partie VHDL

Le filtre FIR VHDL est implémenté ainsi :

accu.vhd	code de l'accumulateur 16+16->21
buff.vhd	code du buffer de sortie (qui loggue les échantillons sur disque)
delay.vhd	la ligne à retard
rom.vhd	code de la ROM contenant les coefficients du FIR
mult.vhd	code du multiplieur 8*8->16
fsm.vhd	la machine d'état (versions synthétisable & non-synthétisable)
fir.vhd	entité regroupant tous les modules du FIR (accu,buff,delay,rom,mult,fsm)
adc.vhd	code simulant un ADC de type AD7822 (avec timings réels !!)
tb_fir.vhd	entité+testbench regroupant le FIR+l'ADC, permet de tester des signaux externes (issus de Scilab)

Il n'y a pas grand chose de particulier à dire à propos de mon code VHDL.
Je l'ai écrit en partant de zéro, en faisant des « Google Search » sur des points particuliers lorsque ce fût nécessaire (ex: utilisation de textio, utilisation de Variable, etc..).
Pour ce qui est de l'API implémentée (signaux et bus utilisés), je me suis basé sur le design de Mr Prost et sur celui du CIME (Grenoble).

Le mieux est d'aller jeter un coup d'oeil directement sur mon code, qui est relativement bien commenté, et surtout très modulaire.

Il faut noter qu'il y a beaucoup de traces de code « mort », le plus souvent commenté, et qui correspond à des phases de test et de debug.
Cela permet de voir ma progression en terme de programmation en VHDL, sachant que je me suis lancé dans ce projet comme débutant absolu dans ce langage.

Je ne fournis pas le code en annexe dans ce rapport intentionellement, car d'une part cela représente un nombre de pages hallucinant et d'autre part, la syntaxe VHDL n'apparaitrait pas colorisées, ce qui en annule l'interêt.
En revanche, l'intégralité de l'arborescence de mon projet (fichiers Scilab, C, VHDL, rapport) sur internet à l'adresse suivante :
http://ezusb.free.fr/?page=fir

Le mieux est donc de télécharger les fichiers sources VHDL (*.vhd) et de les ouvrir à l'aide d'un bon logiciel adhoc. (ex : l'excellent Simili de SymphonyEDA téléchargeable ici : http://www.symphonyeda.com/)

Test et validation

Pour tester le filtre de façon correcte, il faut lui soumettre des signaux d'entrées de type sin(t) d'amplitude 1(v) et de fréquences parmi :
- 10kHz,
- 20kHz,
- 34kHz,
- 60kHz,
- 120kHz.

On peut ainsi vérifier les zones suivantes :

- G > -3dB
- G # -3dB
- -40dB < G < -3dB
- G # -40dB
- G < -40dB

Voici trois tableaux résumant l'ensemble des valeurs obtenues pour chaque jeu de coefficients : Scilab (ma méthode), CIME (polycopié de Mr Allard), et la méthode de Mr Prost (matlab).

Chaque tableau donne ensuite les résultats pour chaque approche de filtrage : Scilab (simple convolution coefs/entrée), C (algo) et VHDL

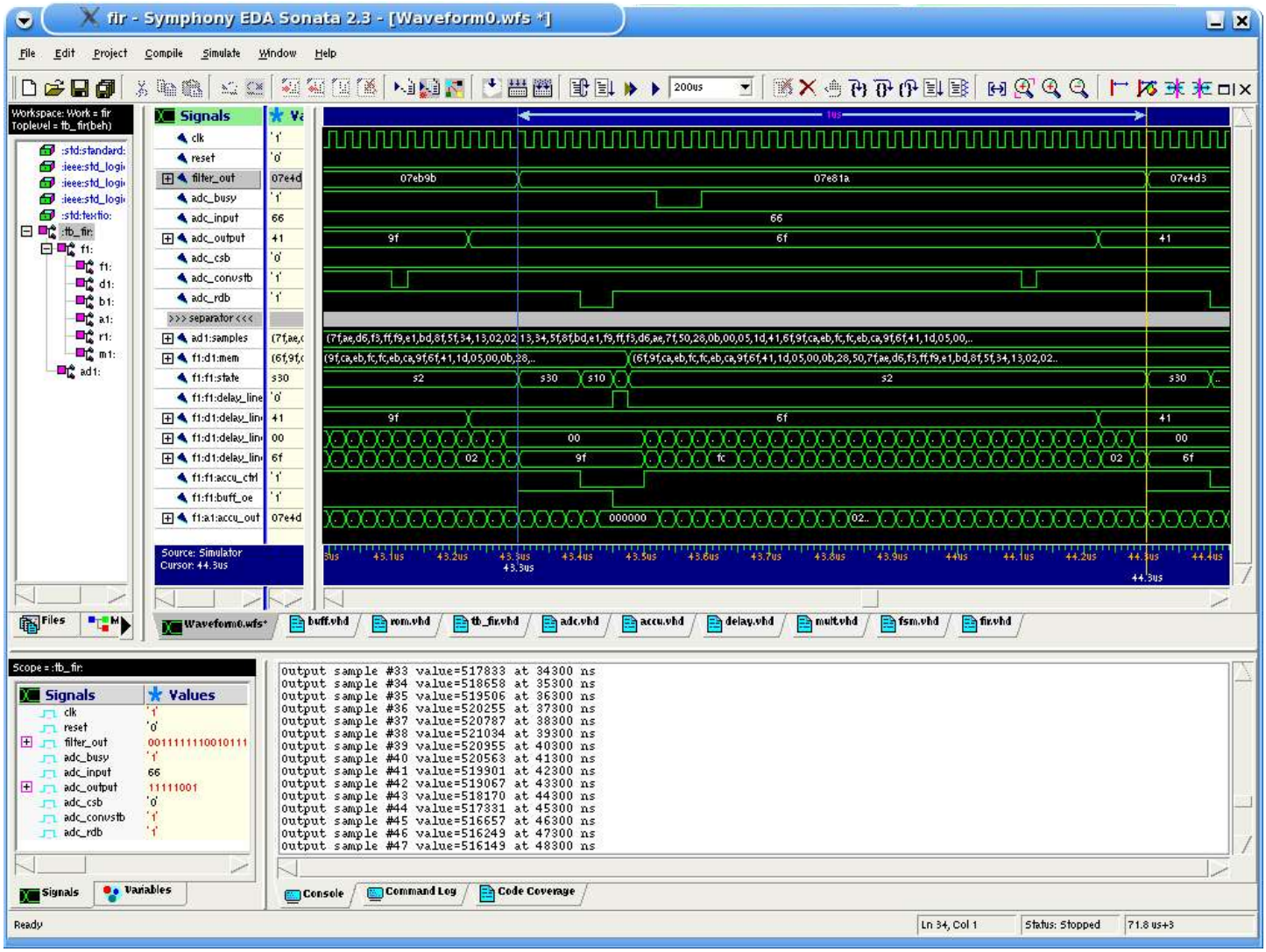
input freq: **[coef : scilab]**	5k	10k	20k	34k	60k	120k
scilab (conv)	-0,17	-0,7	-2,83	-9,16	-43,7	-45,3
C	-0,24	-0,75	-2,83	-9,14	-42,11	-42,11
VHDL	-0,21	-0,73	-2,86	-9,16	-44,79	-45,24

input freq: **[coef : cime]**	5k	10k	20k	34k	60k	120k
scilab (conv)						
C	-0,21	-0,75	-2,83	-9,14	-48,13	-42,11
VHDL			-2,85	-9,16	-46,5	

input freq: **[coef : prost]**	5k	10k	20k	34k	60k	120k
scilab (conv)						
C	-0,21	-0,67	-2,74	-8,67	-36,09	-48,13
VHDL			-2,73	-8,69	-37,65	

Sous Sonata (Symphony EDA Simili) on peut vérifier le fonctionnement au moyen d'une simulation de la manière suivante :
- lancer sonata
- charger le workspace "fir.sws" dans /
- choisir le toplevel "tb_fir(beh)" car on simule le testbench global
- positionner correctement le choix de coef et de fréquence d'entrée dans le fichier tb_fir.vhd (ex : constant ROM_FILE : string := "testing/zROM_cime.txt"; et constant ADC_FILE : string := "testing/zADC_sin60k.txt";)
*Bien sûr, il faut avoir généré ces fichiers à l'aide des fichiers scilab et programme C. (il faut *LIRE* les consignes ci-dessus)*
- charger le fichier 'do' : "tb_fir.do" ce qui ajoute pleins de sigaux utiles pour la simu
- positionner le temps de simu selon la fréquence choisie (ex : pour 10k, il faut calculer 200 échantillons à raison d'une microsec par échant.) encore une fois, regarder la sortie du fichier fir.sci pour savoir le nombre d'échant par example
- lancer la simu — cela crée le fichier buff.txt, qui contient les échantillons de sorties (un par microsec)
- afficher les résultats avec le fichier verif.sci (voir plus haut)

Voici un aperçu de ce que l'on obtient au cours d'une simulation VHDL :
(le calcul se fait en 1us ! et la valeur 21bits hexa correspond à celle du C !)



Conclusion

Le filtre FIR

Au final, si l'on omet la phase de test sur la carte en temps réel, le filtre FIR réalisé fonctionne parfaitement aussi bien au niveau théorique (convolution coefficients sous scilab) qu'au niveau algorithmique (en C) et implémentation VHDL (après simulation sous SymphonyEDA).

Le cahier des charges est respecté haut la main.

Quelque soit la méthode de synthèse des coefficients, mon filtre FIR (version Scilab, C et VHDL) se comporte parfaitement comme un filtre passe-bas du 32ème ordre, pour toutes les fréquences que j'ai pu tester entre 10kHz et 120kHz.

De plus, j'ai réussi à synthétiser le code VHDL de manière à obtenir un code binaire implantable sur une cible FPGA de chez Altera.

Problèmes rencontrés

Les problèmes que j'ai rencontré lors de ce projet sont les suivants :

- être totalement livré à soi-même vis-à-vis d'un nouveau langage
- perte de temps inutile sur le logiciel VSystem que je n'ai pas utilisé au final
- contraintes sévères du cahier des charges (ensemble calculs en moins d'une microsec)
- difficultés lors de la compilation avec l'outil Altera (car mon code VHDL originel était difficilement synthetisable)
- impossibilité de programmer la cible FPGA (car le code binaire est trop volumineux — il faudrait sans doute l'optimiser)

Les conséquences engendrées par ces problèmes :

- auto formation complète sur le langage VHDL et ses diverses astuces
(temps passé sur le projet : facilement supérieur à 80h)

Bilan personnel

Je retire de ce projet l'apprentissage concret d'un nouveau langage : le VHDL et tout l'univers qui gravite autour de sa mise en oeuvre :

- synthèse théorique préalable de l'algo, des paramètres, coefficients..
- développement du code VHDL à la main, sans générateur graphique
- simulation intensive avec création de divers testbenchs
- implémentation physique avec la suite de placement/routage et génération du code binaire du vendeur FPGA (ici Altera)

Ce projet m'a permis également :
- de découvrir un nouveau logiciel de calcul scientifique : scilab (libre), qui m'a permis de me passer avec succès de matlab et son prix exorbitant.
- de découvrir un logiciel de développement et simulation VHDL très performant (en version gratuite limitée) : SymphonyEDA (Simili)

A noter que j'ai créé ce projet de zéro, c'est à dire que toute la partie théorique a été ré écrite avec Scilab, tout le code de vérification de l'algorithme VHDL + influence du codage Q0.8 écrit en langage C, ainsi que tout le code VHDL qui a été fait sans utiliser aucun générateur de code comme Vasco ou autre.

Je suis donc assez satisfait du résultat, d'autant que je suis parti de zéro dans tous les domaines.
Par exemple, j'ai utilisé ma propre méthode pour générer les coefficients du filtre (donc, différents du CIME et de Mr Prost)
J'ai fait une comparaison tabulaire de sa réponse fréquentielle des trois jeux de coef pour les fréquences clés, et les résultats sont très proches, et conformes à la théories. (voir le fichier /report/report.sxc [format OpenOfficeCalc] pour plus d'info)

Enfin, en ce qui concerne la dernière phase de test physique, même mon le code binaire ne tient pas physiquement sur la cible Flex10K livrée sur la plateforme finale (pas assez de cellules logiques), je suis tout de même parvenu à le compiler et à la synthétiser, ce qui n'était pas forcément gagné d'avance, et qui était important pour moi.