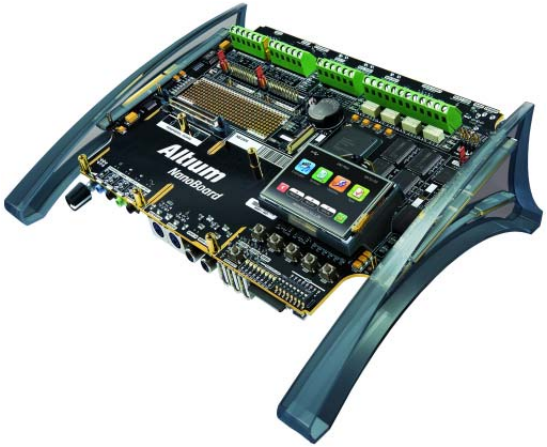
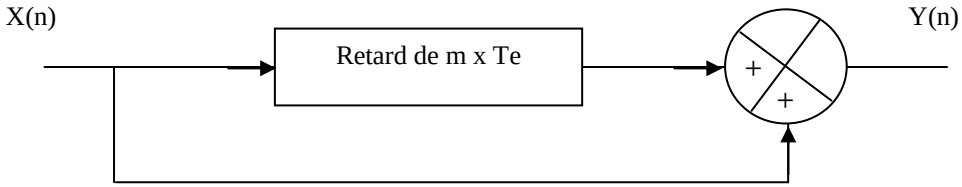
Sommaire :



Annexes :



1 Objectif du projet :

⇒ Réaliser le traitement audio FLANGER de l'ampli MATRIX 100 par un FPGA implanté sur la Nanoboard 3000.

2 Pré requis :

⇒ Savoir écrire un programme en langage C avec passage de paramètres, maîtriser l'algorithmique.
⇒ Savoir écrire un VI sous LABVIEW gérant des entrées sorties.
⇒ Avoir suivi les TP_Processeur_embarque_OPEN_BUS et TP_Compiler_simuler_debugger_un_fichier

3 Nombre d'étudiants menant ce projet :

⇒ 1 étudiant

4 Cahier des charges du mini projet :

⇒ **Mettre en œuvre sur la carte « Nanoboard 3000 » le traitement de l'effet sonore FLANGER.**
- voir la définition de l'effet sonore FLANGER en annexe A1

Vous développerez, dans un premier temps, sous l'environnement ALTIUM la programmation du FPGA afin d'y implanter un microprocesseur **TSK 3000** (mise en œuvre d'**IP**, **I**ntellectual **P**roperties).
Dans un deuxième temps vous programmerez, en langage C, le microprocesseur afin de réaliser l'effet FLANGER.

⇒ Caractéristiques de l'acquisition numérique :
- fe= 48kHz
- résolution 16bits.

⇒ Entrée audio LINE IN
- Niveau d'entrée nominal 6dBV / Niveau d'entrée maximal : 20dBV

⇒ Sortie audio LINE OUT
- Niveau de sortie nominal 6dBV

⇒ Sortie audio sur les HP intégrés à la Nanoboard

5 Outil de test du mini projet :

⇒ **Ecrire sous LABVIEW un VI qui permette de tester le projet à développer.**
⇒ Vous développerez un banc de mesure permettant de tester votre application. Voir le synoptique Annexe 5.
⇒ Ce banc de mesure incorporera un PC équipé de LABVIEW qui permettra de générer les signaux suivants :
- Un signal de $Vm(t) = 1V \sin (2 \pi f t)$ avec $0 < f < 50$ kHz
- Un air de guitare électrique, mode CLEAN sans effet audio. (fichier WAV)

⇒ Voir le site : http://www.hughes-and-kettner.com/products.php5?id=126&prod=Attax 100 Combo
⇒ Ces signaux seront générés par la carte audio du PC
⇒ Vous mettrez en place un grapheur permettant de visualiser les signaux électriques transmis à la Nanoboard.

6 Ressources fournies :

⇒ TD.1 : Analyse structurelle des fonctions de la carte Nanoboard mises en œuvre durant ce projet.
⇒ Le VI «Banc_De_Test_Audio.vi» à compléter.
⇒ Le projet ALTIUM «Projet_Codec_VS=VE_a_Completer» que vous ferez évoluer.
⇒ Les annexes A1 à A8 du présent document.

7 Planification du travail sur les 6 séances de 4h , diagramme de Gantt :

	Séance 1		Séance 2		Séance 3		Séance 4		Séance 5		Séance 6	
Etape 1 : Découverte du sujet et du matériel associé. Lecture du présent document. Organisation de l'espace informatique par l'étudiant.												
Etape 2 : TD1 : Etude structurelle de la Nanoboard 3000 (Voir énoncé du TP8.1) : étude des structures mises en œuvre dans ce projet : FPGA, CODEC, partie audio.												
Etape 3 : Mesure sur le système existant : l'ampli MATTRIX100. relever les oscillogrammes en sortie LINE de l'ampli de guitare avec et sans effet FLANGER.												
Etape 4 : Ecriture du VI générateur du son d'une guitare. Compléter le VI incomplet fourni « Banc_De_Test_Audio.vi » afin de répondre au cahier des charges.												
Etape 5 : Traitement numérique du signal : le filtre Vs=Ve. Compléter le projet incomplet fourni. Guidance ressource voir TP4 formation ALTIUM.												
Etape 6 : Traitement numérique du signal . Faites évoluer le projet de l'étape 5 afin de mettre en œuvre le filtre numérique PB présenté en annexe 4. Relever le Bode du filtre.												
Etape 7 : Traitement numérique du signal . Faites évoluer le projet de l'étape 5 afin de mettre en œuvre l'effet FLANGER.												
Etape 8 : Constituer un dossier contenant le travail à rendre. Présenter votre programme le plus élaboré à l'enseignant. Vérifier par mesure les attentes du cahier des charges.												

8 Travail à rendre :

1 : Le VI «Banc_De_Test_Audio.vi» complété sous LABVIEW, permettant le test des projets développés.
2 : Trois projets ALTIUM sous forme numérique :
- ⇒ Projet ALTIUM 1 : traitement numérique Vs = Ve
- ⇒ Projet ALTIUM 2 : filtre passe bas 1er ordre. Fo = 5000Hz A0=1
- ⇒ Projet ALTIUM 3 : traitement de l'effet sonore FLANGER

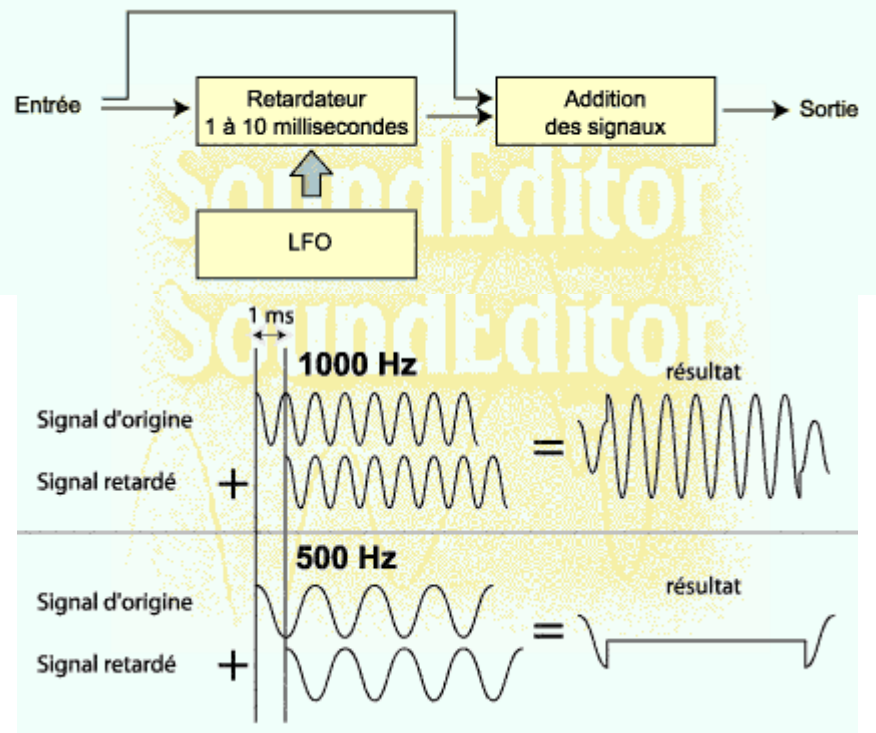
3 Les chronogrammes acquis mettant en évidence les traitements audio. Ces chronogrammes, concaténés dans un fichier WORD, doivent être commentés.
4 Le diagramme de Bode du filtre développé dans le projet 2 sous tableur EXCEL.
5 Vous présenterez à l'enseignant votre programme le plus élaboré lors d'une mise en œuvre du banc de mesure décrit en annexe 5.
6 Le diagramme de Gantt ci-dessus complété.

ANNEXES : INFORMATIONS COMPLEMENTAIRES AU CAHIER DES CHARGES :

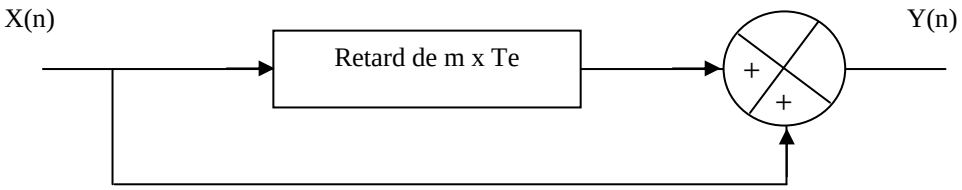
A1 : l'effet Flanger :

Il s'agit d'un effet retard variable de courte durée (de 1 à 10 millisecondes) qui est mélangé au signal d'entrée. On constate que la fréquence de 1000 Hz, multiple du retard de 1ms, choisie pour l'illustration, est amplifiée par ce montage. Les périodes du signal s'additionnent en effet les unes aux autres. La fréquence de 500 Hz est par contre totalement occultée. Bien que cela ne soit pas représenté sur notre graphique, toutes les harmoniques multiples ou sous multiples de 1000 Hz seront affectées (125 Hz, 250 Hz, 500Hz, 1000 Hz, 1500 Hz, 2000 Hz, etc.) Cet effet va donc modifier profondément la répartition des harmoniques en amplifiant certaines fréquences et en en occultant d'autres. Le retard de 1ms étant dans la réalité variable il est piloté par un *Low Frequency Oscillator* - oscillateur de basse fréquence.

⇒ Source : http://www.freesoundeditor.com/flangerphaser.htm



A2 : Synoptique du traitement numérique de l'effet FLANGER :



La fréquence d'échantillonnage est de 48 kHz
Pour un retard variant de 1 à 10ms, pour une période d'échantillonnage de 20,3µs il en découle :

$$1\ ms < m \times Te < 10\ ms$$

donc $48 < m < 480$

Equation aux différences :

$Y(n) = X(n) + X(n-m)$

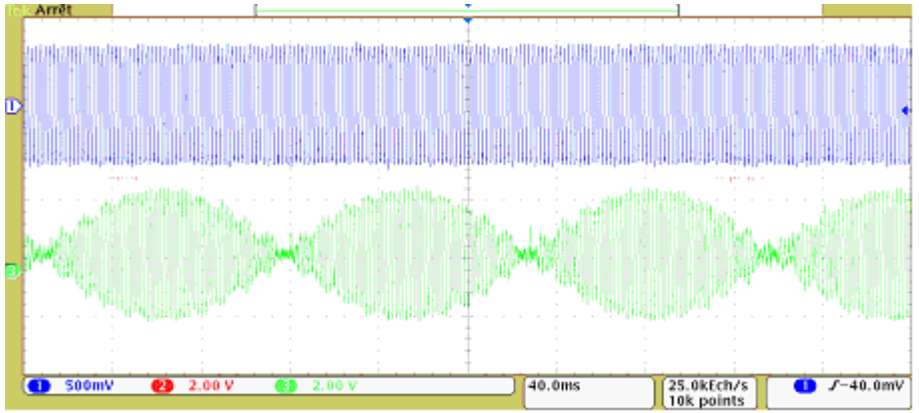
Equation en z :

$Y(z) = X(z)\,[1 + z^{-m}]$

Fonction de transfert en z :

$H(z) = [1 + z^{-m}]$

A3: Chronogrammes de l'effet Flanger attendus.



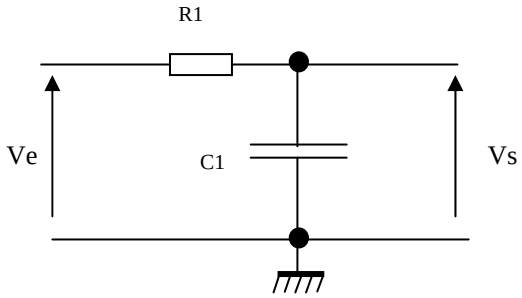
Signal avant traitement
(f= 500Hz)
ve(t) = 1 . sin (2 . Π . 500 . t)

Signal obtenu suite au traitement FLANGER

A4 : filtrage numérique : filtre passe bas numérique. (Rappel des résultats obtenus en physique appliquée).

Filtre passe bas numérique identifié à un filtre passe bas analogique du premier ordre en mettant en oeuvre la transformation bilinéaire, où la fréquence d'échantillonnage est de 48 kHz. Caractéristiques du filtre analogique:Fréquence de coupure fc = 5 kHz , Tmax = 1

Equation du filtre passe bas du premier ordre: $T(p)=\frac{1}{1+\tau\times p}$



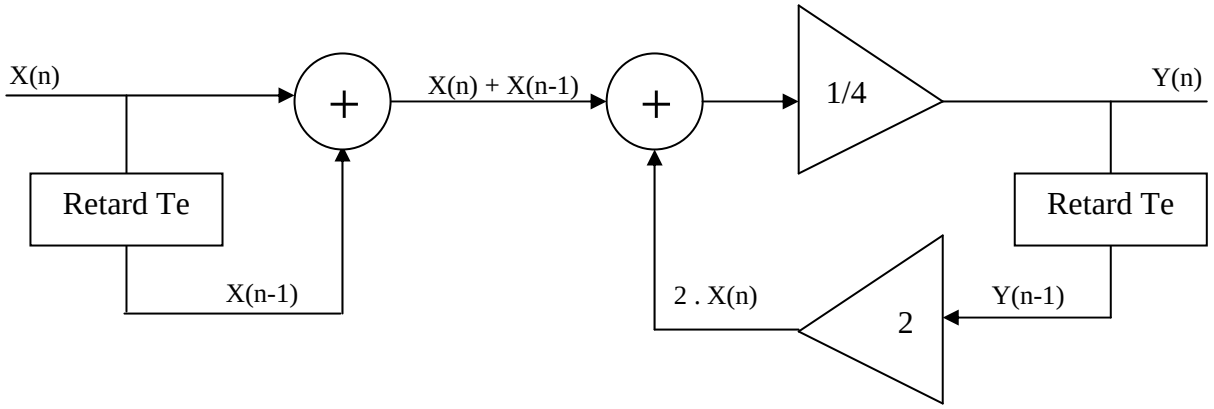
Transformation bilinéaire $p=K\times\frac{1-z^{-1}}{1+z^{-1}}$ avec $K=2\times\pi\times\frac{Fa}{\tan(\pi\times\frac{fd}{fe})}$

⇒ Il est choisi une correspondance des fréquences analogique fa et numérique fd, à la fréquence de coupure du filtre soit : fa = fd = fc = 5kHz et fe = 48Hz

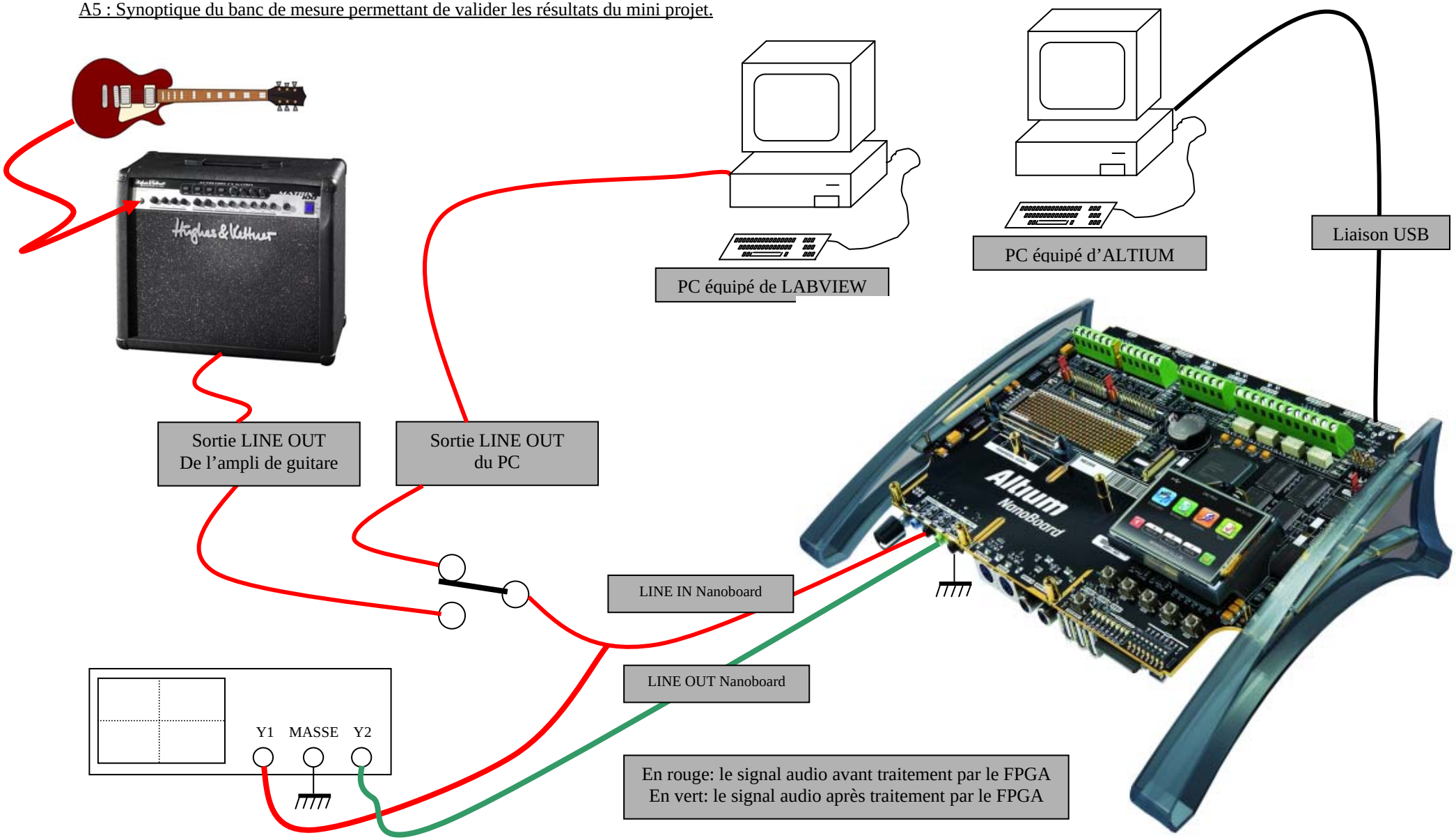
$K=2\times\pi\times\frac{5000}{\tan(\pi\times\frac{5}{48})}=92548$ donc $T(z)=\frac{1+Z^{-1}}{3{,}95-1{,}95\times Z^{-1}}$

d'où l'équation aux différences:

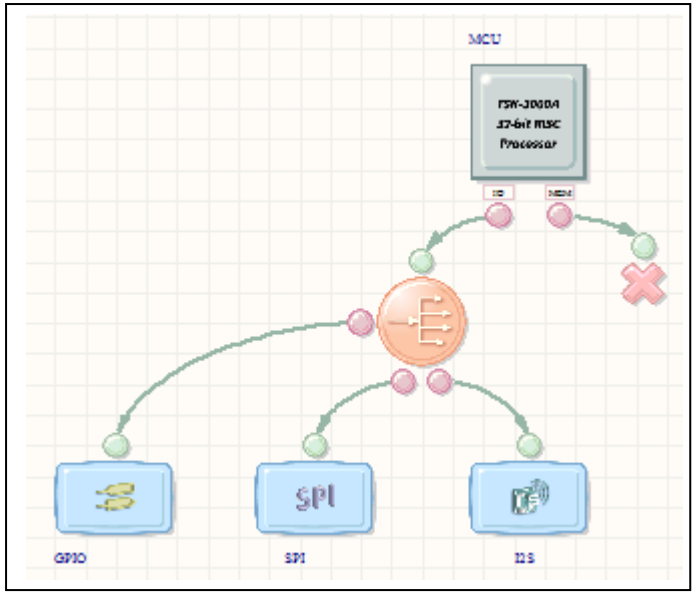
⇒ Y(n) = 0,25 . X(n) + 0,25 . X(n-1) + 0.5 . Y(n-1) ⇒ soit : Y(n) = 1/4 [X(n) + X(n-1) + 2 . Y(n-1)]



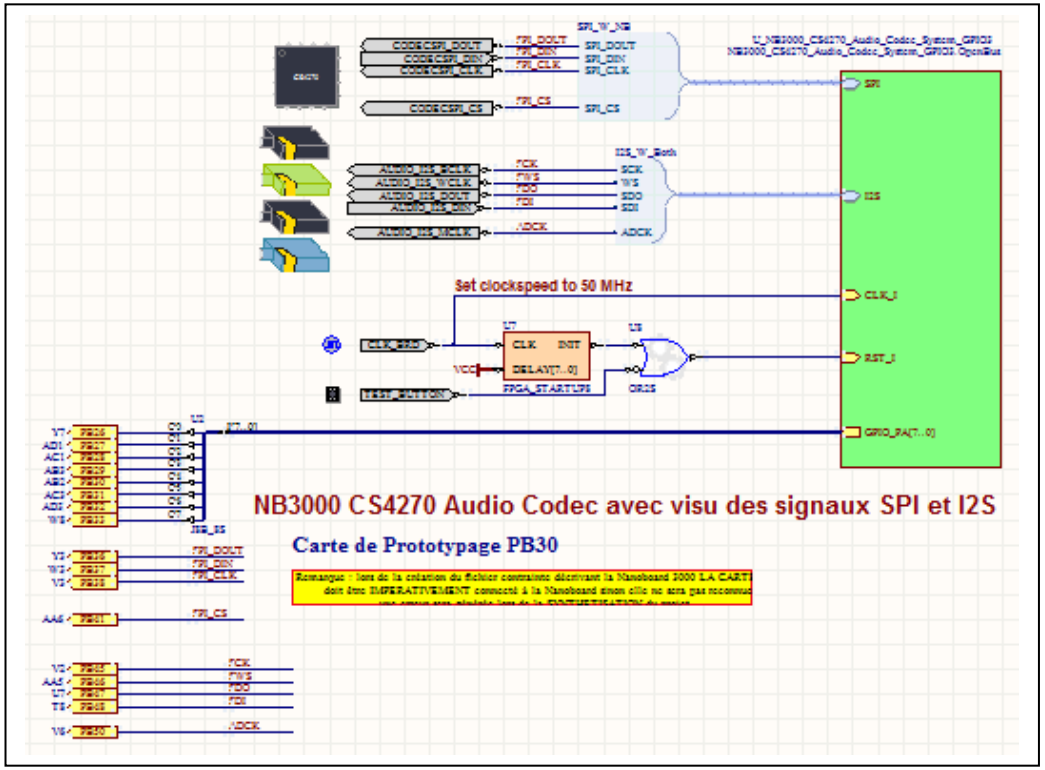
A5 : Synoptique du banc de mesure permettant de valider les résultats du mini projet.



A6 : Schéma « Open Bus » à mettre en place pour le projet.



A7 : Schéma « TOP » à mettre en place dans le projet :



Vous serez amenez à complétez les schémas ci-dessus lors de l'étape 5 du projet. Voir le répertoire « Codec VS=VE avec marqueur a completer. »

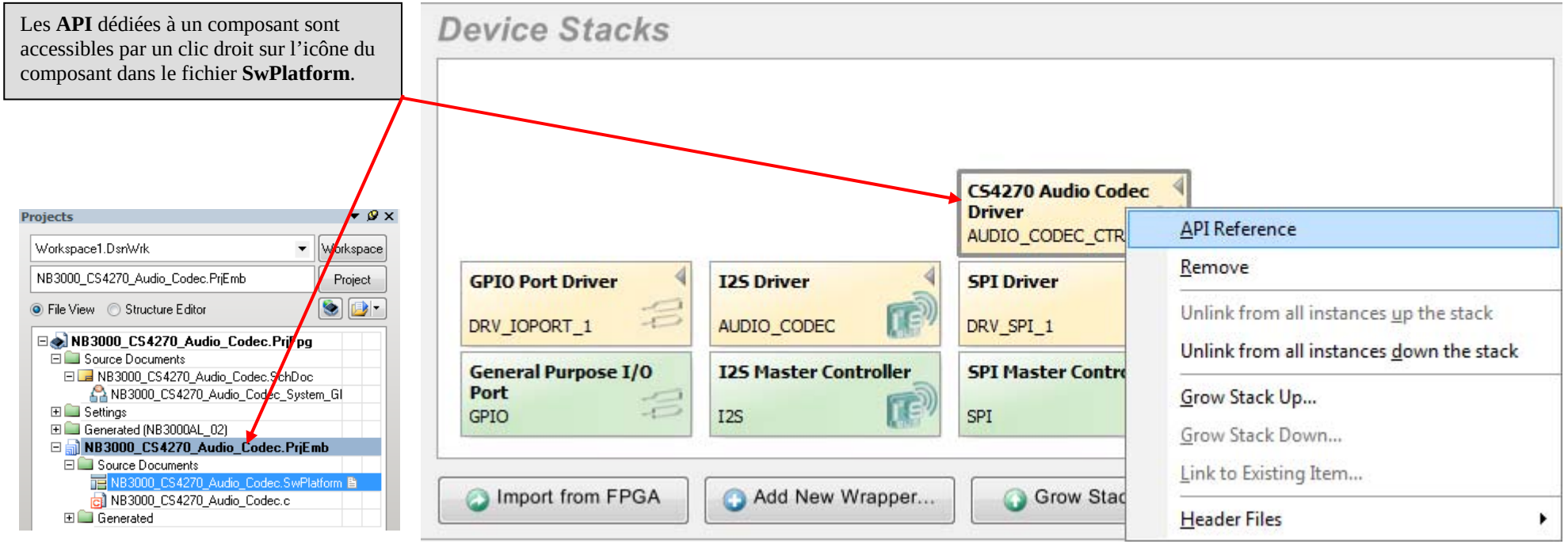
A8 : Présentation des principales fonctions **API** en langage C à mettre en œuvre durant le projet.

API : **A**pplication **P**rogramming **I**nterface
une **API** est un ensemble de fonctions, procédures ou classes mises à disposition par une bibliothèque logicielle qui permettent d'interfacer deux entités.
Dans notre cas les **API** mises à notre disposition nous permettent de dialoguer avec les composants placés sur la carte Nanoboard.
Nous nous intéresserons plus particulièrement aux API permettant de :

- ⇒ mettre en œuvre le bus SPI (pour paramétrer le CODEC)
- ⇒ mettre en œuvre le bus I2S (pour transférer les données numériques images de l'audio entre le CODEC et le FPGA)
- ⇒ lire et écrire sur les entrées sorties port (pour renvoyer des informations sur la carte fille PB30, notamment des marqueurs placés dans le programmes C pour contrôler la durée d'exéction des fonctions)

Ou trouver les API disponibles au sein d'un projet FPGA sous ALTIUM ?

Les **API** dédiées à un composant sont accessibles par un clic droit sur l'icône du composant dans le fichier **SwPlatform**.



Nom de la fonction :	`swplatform_init_stacks`	Composants adressés :	Tous ceux décrit dans le fichier OPEN BUS	**Description des paramètres :** aucun **Paramètre renvoyé :** Aucun
Fichier d'entête :	`#include "swplatform.h"`			
Prototype de la fonction	`void swplatform_init_stacks(void);`			

Exemple :

```
#include "swplatform.h"


void main(void)
{
    swplatform_init_stacks();    // generated initialization code
    while(1);
}
```

Description de la fonction : Cette fonction a été écrite automatiquement lors de la compilation du projet embarqué. Elle initialise les **API** nécessaires au projet en cours.

Voici la fonction dans le cadre de notre projet :

```
 // Initialize all stacks in the Software Platform
void swplatform_init_stacks(void)
{
    audio_codec      = i2s_open(AUDIO_CODEC);
    drv_ioport_1     = ioport_open(DRV_IOPORT_1);
    audio_codec_ctrl = cs4270_open(AUDIO_CODEC_CTRL);
}
```

Nom des fonctions :	`i2s_rx_start / i2s_tx_start`	Composant adressé :	CODEC via le bus I2S	**Description des paramètres :** i2s_drv adresse du composant **Paramètre renvoyé :** Aucun
Fichier d'entête :	`#include <drv_i2s.h>`			
Prototype des fonctions	`void i2s_rx_start(i2s_t * restrict i2s_drv); // void i2s_tx_start(i2s_t * restrict i2s_drv);`			

Exemple :

```
#include "swplatform.h"
#include < drv_i2s.h >


void main(void)
{
    swplatform_init_stacks();  // generated initialization code

    i2s_rx_start(audio_codec);
    i2s_tx_start(audio_codec);
}
```

Description des fonctions :

`i2s_rx_start` : initialise la liaison I2S entre le CODEC et le FPGA.

`i2s_tx_start` : initialise la liaison I2S entre le FPGA et le CODEC.

Nom de la fonction :	i2s_read16	Composant adressé :	CODEC via le bus I2S
Fichier d'entête :	#include <drv_i2s.h>		
Prototype de la fonction	void i2s_read16(i2s_t *restrict i2s_drv, int16_t *buffer, unsigned int n);		

Description des paramètres :
i2s_drv adresse du composant
buffer start address of buffer for samples
n Nombre d'échantillons demandés

Paramètre renvoyé : Aucun

Exemple :

```
#include "swplatform.h"
#include <drv_cs4270.h>
#define AUDIO_BUF_SIZE      2

// Variables globales :
int16_t audio_buf[AUDIO_BUF_SIZE];
uint32_t rx_size=AUDIO_BUF_SIZE;

void main(void)
{
// read samples result in *audio_buf
i2s_read16(audio_codec, audio_buf, rx_size);
}
```

Description de la fonction :

i2s_read16 : permet de lire un tableau d'échantillons acquis par le CODEC et stockés dans une mémoire tampon.
Le nombre d'échantillons à lire est renseigné par « rx_size »
Les échantillons lus seront stockés dans le tableau « audio_buf [ ] »

Nom de la fonction :	i2s_write16	Composant adressé :	CODEC via le bus I2S
Fichier d'entête :	#include <drv_i2s.h>		
Prototype de la fonction	void i2s_read16(i2s_t *restrict i2s_drv, int16_t *buffer, unsigned int n);		

Description des paramètres :
i2s_drv adresse du composant
buffer start address of buffer for samples
n Nombre d'échantillons demandés

Paramètre renvoyé : Aucun

Exemple :

```
#include "swplatform.h"
#include <drv_i2s.h>
#define AUDIO_BUF_SIZE      2

// Variables globales :
int16_t audio_buf[AUDIO_BUF_SIZE];
uint32_t rx_size=AUDIO_BUF_SIZE;

void main(void)
{
// write samples / send *audio_buf
i2s_write16(audio_codec, audio_buf, rx_size);
}
```

Description de la fonction :

i2s_read16 : permet d'écrire, dans le CODEC dans une mémoire tampon. Dans un deuxième temps le CODEC va générer ces échantillons sur la sortie analogique.
Le nombre d'échantillons à lire est renseigné par « rx_size »
Les échantillons transmis au CODEC proviennent du tableau « audio_buf [ ] »

Nom de la fonction :	i2s_rx_avail / i2s_tx_avail	Composant adressé :	CODEC via le bus I2S
Fichier d'entête :	#include <drv_i2s.h>		
Prototype de la fonction			

Description des paramètres :

Paramètre renvoyé :

Exemple :

```
while(1)
{
   while (i2s_rx_avail(audio_codec) < AUDIO_BUF_SIZE / 2)__nop();
   // wait till there are a number of samples available

   while (i2s_tx_avail(audio_codec) < rx_size) __nop();
  // wait till there is enough space available
}
```

Description de la fonction :

`i2s_rx_avail` : Contrôle le nombre d'échantillons en stocke dans le CODEC avant d'effectuer une lecture du buffer.

`i2s_tx_avail :` contrôle l'espace disponible dans le buffer du CODEC avant de faire une écriture du FPGA vers le CODEC.

Nom de la fonction :	ioport_set_value	Composant adressé :	Port de la carte fille PB30
Fichier d'entête :	#include <drv_ioport.h>		
Prototype de la fonction	void ioport_set_value(ioport_t * restrict drv, const uint8_t port, const uint32_t value)		

Description des paramètres :
drv I/O Port driver configuration data
port the port number to write the value to; must be from 0 to (drv->port_count - 1)
value the value to be written to the port
Paramètre renvoyé : Aucun

Exemple :

```
void main(void)
{
 ioport_set_value(drv_ioport_1, GPIO_PA, 0xff);//PORTA ppv 1111 1111
 ioport_set_value(drv_ioport_1, GPIO_PA, 0x00);//PORTA ppv 0000 0000
}
```

Description de la fonction :

`ioport_set_value :` cette fonction permet d'imposer un niveau logique sur les broches PB26 à PB33 de la carte fille PB30.

Dans le cadre du projet cela nous permettra demettre en place des marqueurs afin de controller les temps d'exécution des fonctions.