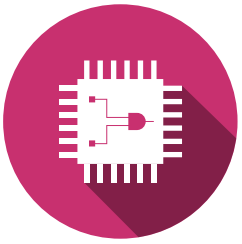


Générateur de fonctions avec interpolation

Table des matières

1 Introduction

Ce laboratoire porte sur des opérations sur des nombres, illustrées par le clacul de l'interpolation de points de la fonction sinus d'un générateur de fonctions numérique. Le calcul de l'interpolation se base sur l'approximation de la fonction entre deux points donnés par des fonctions polynomiales du troisième ordre. Cette méthode appartient à la grande famille des **splines**.

2 Lancement

Le circuit se trouve dans la librairie **SineInterpolator**. Le banc de test dans la librairie **SineInterpolator_test**.

*Rappel : le programme de modélisation doit être lancé à travers le fichier **splineInterpolator.bat**.*

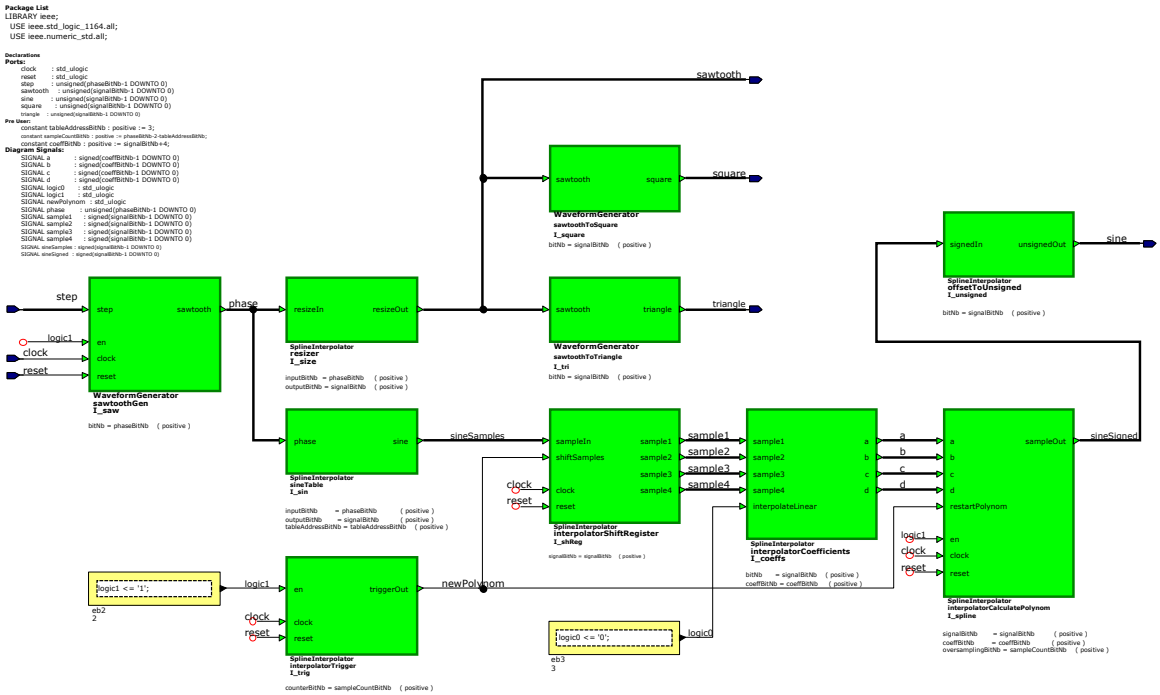


FIGURE 1 – Générateur de fonctions avec interpolation

3 Objectifs

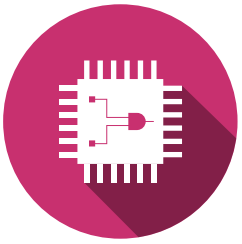
3.1 Dent de scie

Dans ce circuit, le nombre de bits du signal **phase** (de **phaseBitNb**) n'est pas identique à celui des sorties du générateur de fonctions (de **signalBitNb**). Pour utiliser les blocs qui fournissent les signaux en dent de scie, triangulaire et carré, le bloc **resizer** modifie le nombre de bits du compteur de phase. 3 cas sont à prévoir :

- le nombre de bits des sorties est supérieur à celui de la phase : il faut donc copier le signal de phase dans les bits de poids fort du signal en dent de scie
- le nombre de bits des sorties est égal à celui de la phase : il suffit de copier le signal de phase dans celui en dent de scie
- le nombre de bits des sorties est plus petit que celui de la phase : il faut ici conserver les bits de poids fort du signal de phase

3.1.1 Todo

1. Avec l'aide du constructeur **if ... generate**, écrivez le code VHDL du bloc **resizer**
2. Compiler et simuler le bloc **sineGen_tb**. Vérifier le bon fonctionnement du modificateur de taille de signal pour les trois cas de fonctionnement.



3.2 Table de sinus

L'interpolation se fera à partir d'une table contenant 8 valeurs de sinus pour un quart de sa période. Pour fonctionner sur toute la période, le signal en dent de scie provenant du compteur et considéré comme la phase du sinus et est utilisé comme suit :

- le bit de poids fort signale le changement de signe des valeurs du sinus
- le bit suivant signale un changement à apporter à la phase pour lire la table dans le sens inverse (deuxième et quatrième quarts de la période)
- les 3 bits suivants (**tableAddressBitNb = 3**) sont utilisés pour adresser les valeurs de la table
- les autres bits sont ignorés

L'adresse de la table de sinus est tirée des 3 bits suivant les 2 bits de poids fort de la phase. Elle effectue la séquence "0, 1, 2, 3, 4, 5, 6, 7, 0, 7, 6, 5, 4, 3, 2, 1, 0, 1, 2, 3, ...". La table de sinus contient les valeurs hexadécimales 0000 ou 7FFF (selon la valeur du deuxième bit de poids fort), 18F9, 30FB, 471C, 5A82, 6A6D, 7641 et 7D89. Lorsque le bit de poids fort de la phase vaut '1', la valeur lue de la table change de signe pour donner la sortie.

3.2.1 Todo

1. Ecrire l'architecture VHDL du générateur de sinus.
2. Compiler et simuler le bloc **sineGen_tb**. Vérifier le bon fonctionnement du générateur de sinus.

3.3 Interpolation

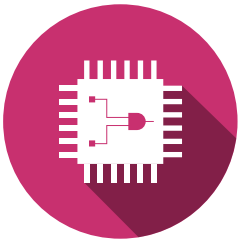
L'interpolation se fait en associant une fonction polynomiale d'ordre 3 entre chaque point de la table de sinus : $\mathbf{y = a \cdot k3 + b \cdot k2 + c \cdot k + d}$. Au cours du calcul, le point courant de la table de sinus est défini en k = 0, le point précédent en k = -1 et le point suivant en k = 1. Nous calculerons la valeur du polynôme pour $\mathbf{n = 2m}$ points situés entre 0 et 1. Pour assurer la continuité entre le polynôme qui relie le segment $-1 < k < 0$ et celui qui relie $0 < k < 1$, nous spécifons la dérivée au point y(k=0) comme étant égale à la pente entre les points y(k=-1) et y(k=1). Il en sera de même pour tous les points de la fonction à interpoler.

Le segment $0 < k < 1$ est donc soumis aux 4 conditions suivantes

- y(k=0) = y0 : le polynôme passe par le point (k=0, y=y0)
- y(k=1) = y1 : le polynôme passe par le point (k=1, y=y1)
- y'(k=0) = [y(k=1) - y(k=-1)] / 2 : la dérivée au point (k=0) est la pente entre y(k=-1) et y(k=1)
- y'(k=1) = [y(k=2) - y(k=0)] / 2 : la dérivée au point (k=1) est la pente entre y(k=0) et y(k=2)

Les 4 conditions nous servent à déterminer les 4 coefficients a, b, c et d du ploynôme à partir des 4 points y(k=-1) à y(k=2) de la courbe à interpoler. La résolution de ce système d'équations nous donne les valeurs suivantes des coefficients :

- a = 1/2 [- y(k=-1) + 3 · y(k=0) - 3 · y(k=1) + y(k=2)]
- b = 1/2 [2 · y(k=-1) - 5 · y(k=0) + 4 · y(k=1) - y(k=2)]
- c = 1/2 [- y(k=-1) + y(k=1)]
- d = y(k=0)



Par souci de simplification, nous calculerons le double de la valeur des coefficients et par la suite nous diviserons la valeur du polynome calculé par 2. A chaque nouvelle valeur du signal d'origine (ici, de la table de sinus), il faut recalculer les coefficients du polynôme. Puis, à chaque période d'horloge et en attendant l'arrivée de la valeur suivante, on calcule la fonction polynomiale pour déterminer la valeur courante de l'échantillon de sortie.

3.4 Générateur d'impulsions de séquencement

Ce circuit fournit une impulsion qui dure une période d'horloge à chaque changement de polynôme, c'est-à-dire pour chaque nouveau segment de courbe. Ceci se fait chaque **2n** périodes d'horloge où **en = '1'**, avec **n = sampleCountBitNb = phaseBitNb - 2 - tableAddressBitNb**. Dans notre exemple de laboratoire, **phaseBitNb = 10** et **sampleCountBitNb = 5**.

3.4.1 Todo

1. Ecrire l'architecture VHDL du générateur d'impulsions de séquencement.

3.5 Registre à décalage

A chaque impulsion de synchronisation **shiftSamples**, le circuit mémorise le nouvel échantillon de la fonction à interpoler, **sampleIn**, et le mémorise comme dernière valeur d'échantillon, **sample4**. En même temps, il décale l'échantillon **sample4** dans **sample3**, etc...

Utiliser un tableau (array) de nombres signés pour la description interne du registre à décalage.

3.5.1 Todo

1. Ecrire l'architecture VHDL du registre à décalage qui mémorise les 4 derniers points de la fonction à interpoler.

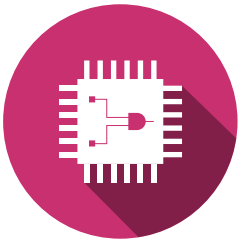
3.6 Calcul des coefficients

Les coefficients se calculent comme suit :

— a = - sample1 +3 · sample2 -3 · sample3 + sample4
— b = 2 · sample1 -5 · sample2 +4 · sample3 - sample4
— c = - sample1 + sample3
— d = sample2

3.6.1 Todo

1. Parcourir la librairie **ieee.numeric_std** et vérifier l'effet de la multiplication d'un signal de type **signed** par un nombre entier. Vérifier également le fonctionnement de la fonction **resize**. Elle est très utile ici pour forcer le nombre de bits des signaux.
2. Ecrire l'architecture VHDL du bloc qui calcule les coefficients du polynôme en fonction de la valeur des 4 derniers points de la fonction à interpoler.
3. Expliquer le choix effectué **coeffBitNb := signalBitNb+4**.



3.7 Calcul du polynôme

Le polynôme se calcule de manière itérative. Ainsi, pour une fonction de premier ordre $\mathbf{y(x) = a \cdot x + b}$, la valeur suivante vaut $\mathbf{y(x+eps) = y(x) + a \cdot eps}$. Ceci se calcule en initialisant $\mathbf{u = a \cdot eps}$, $\mathbf{y(0) = b}$, et en calculant $\mathbf{y(i+1) = y(i) + u}$ à chaque nouvelle période d'horloge. Pour une fonction de deuxième ordre $\mathbf{y(x) = a \cdot x2 + b \cdot x + c}$, ce qui donne $\mathbf{y(x+eps) = y(x) + u(x)}$, avec $\mathbf{u(x) = 2 \cdot a \cdot eps \cdot x + (a \cdot eps2 + b \cdot eps)}$. L'incrément $\mathbf{u(x)}$ se calcule de la manière définie pour une fonction du premier ordre. Ceci se calcule donc en initialisant $\mathbf{v = 2 \cdot a \cdot eps2}$, $\mathbf{u(0) = a \cdot eps2 + b \cdot eps}$, $\mathbf{y(0) = c}$ et en calculant $\mathbf{y(i+1) = y(i) + u(i)}$ et $\mathbf{u(i+1) = u(i) + v}$ à chaque nouvelle période d'horloge. Le calcul de la fonction de troisième ordre se fait de manière similaire. Pour le calcul numérique, **eps** est choisi comme égal à une puissance négative de 2, ce qui permet de la calculer par un décalage. De même, pour rester avec des nombres entiers, on calculera la fonction $\mathbf{y(i)}$ multipliée par une puissance de 2 et un décalage final nous donnera la valeur résultante des échantillons du polynôme.

Notre algorithme est donc le suivant :

- A l'arrivée d'un nouvel échantillon, il faut calculer les valeurs initiales utilisée pour développer le polynôme de manière itérative.
- Entre deux échantillons d'entrée, le calcul de la fonction polynomiale se fait par additions à l'aide des équations itératives.

Les valeurs initiales sont les suivantes :

- $x = 2 \cdot d \cdot 2(3 \cdot m)$
- $u = a + b \cdot 2m + c \cdot 2(2 \cdot m)$
- $v = 6 \cdot a + 2 \cdot b \cdot 2m$
- $w = 6 \cdot a$
- $y = d$

Le calcul itératif du polynôme se fait comme suit :

- $x = x + u$
- $u = u + v$
- $v = v + w$
- $y = x \ / \ (2 \cdot 2(3 \cdot m))$

Il est évident que, du fait des décalages **2m**, le nombre de bits des signaux internes devra être plus élevé que le nombre de bits des échantillons d'entrée ou de sortie. En fonction de la valeur du décalage $\mathbf{m = oversamplingBitNb}$, estimer la taille nécessaire pour ces signaux. Les déclarer avec 8 bits supplémentaires pour s'assurer de ne pas avoir de dépassement de capacité.

3.7.1 Todo

1. Écrire l'architecture VHDL du bloc qui calcule la valeur du polynôme d'interpolation de fonction à chaque nouvelle période d'horloge.
2. Recompiler et simuler le bloc **sineGen_tb**. Vérifier la forme du signal sinusoïdal après interpolation.
3. Réduire le nombre de bits des signaux du circuit là où cela est possible.