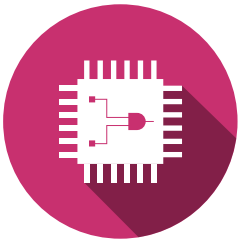


Générateur de fonctions

Table des matières

1 Introduction

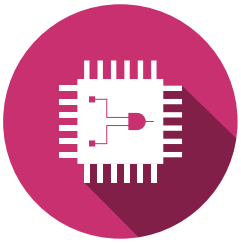
Ce laboratoire illustre l'écriture de code VHDL de blocs combinatoires et de blocs séquentiels synchrones au travers de la réalisation d'un générateur de fonctions numérique. Le générateur délivre :

- un signal en dents de scie (**sawtooth**)
- un signal carré (**square**)
- un signal triangulaire (**triangle**)
- un signal polygonal ((**polygon**))
- un signal sinusoïdal (**sine**)

Les signaux sont codés en tant que nombres (type **positive**), de **bitNb** bits (défini à 16 bits).

2 Lancement

Le circuit se trouve dans la librairie **WaveformGenerator**. Le banc de test est lui disponible sous **WaveformGenerator_test**.



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

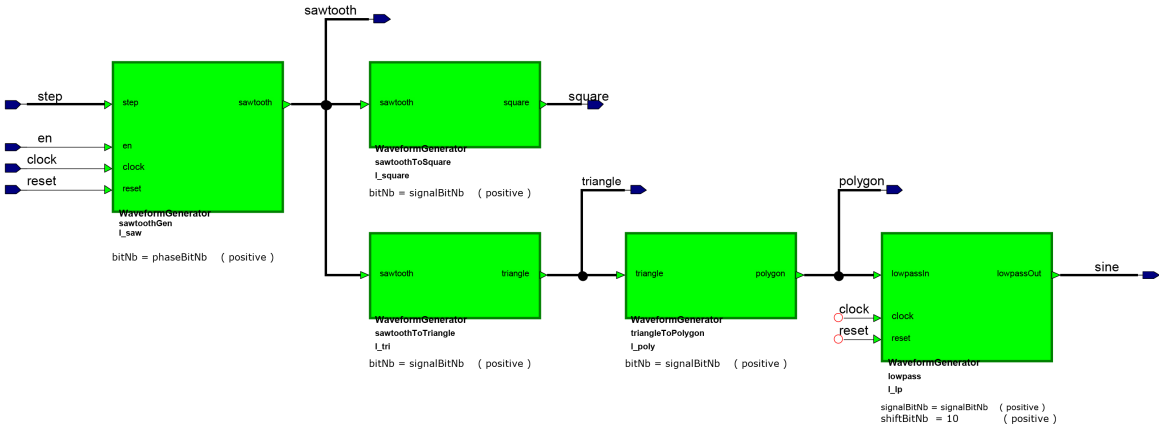


FIGURE 1 – Générateur de fonctions

3 Objectifs

3.1 Dent de scie

Le générateur de signal en dents de scie se fait à l'aide d'un compteur qui tourne en rond. Un nombre en entrée, **step**, permet de commander la fréquence du signal (plus il est grand, plus le compteur déborde rapidement et donc la fréquence est grande). A chaque flanc montant de l'horloge, lorsque **en = '1'**, le compteur ajoute la valeur de **step** à sa valeur courante. Ainsi, plus la valeur d'entrée est grande et plus le compteur tourne rapidement.

Ce bloc contient un paramètre générique (**generic**) : **bitNb**. Ce paramètre est défini aussi bien pour l'entité que pour l'architecture à écrire.

*Le signal de sortie, **sawtooth**, peut uniquement être assigné. Il ne peut être lu, et donc impossible de l'utiliser pour lui additionner la valeur de **step**. Il faut donc définir un signal interne, lequel peut être lu et assigné, et copier ce signal sur le port de sortie de manière combinatoire.*

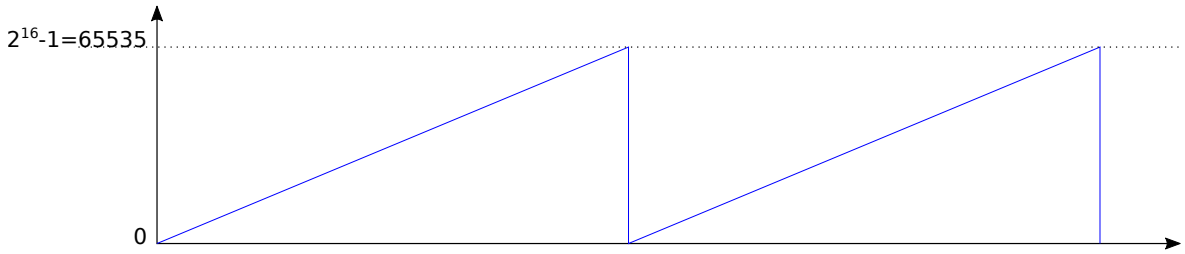
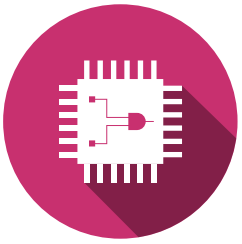


FIGURE 2 – Signal en dent de scie



3.1.1 Todo

1. Ecrire l'architecture VHDL du générateur de signal en dents de scie, sachant que ce circuit est séquentiel et synchrone.
2. Examiner l'entité du générateur de signal en dents de scie pour connaître les types des signaux d'entrée/sortie. Parcourir la définition du paquetage **numeric_std** pour se faire une idée des fonctions à disposition sur les signaux de type **unsigned**.
3. Estimer la période du signal en dents de scie pour un signal d'horloge de 66 MHz, un compteur de 16 bits et une valeur de **step** égale à 8.
4. Donner la formule générale de la fréquence du signal généré en fonction de la fréquence de l'horloge, du nombre de bits du compteur et de la valeur du pas.
5. Compiler et simuler le bloc **waveformGen_tb**. Vérifier le bon fonctionnement du générateur de signal en dents de scie.

3.2 Carré

La transformation du signal en dents de scie en un signal carré se fait en ne considérant que le bit de poids fort du signal en dents de scie. Le signal généré sera lui aussi codé sur 16 bits.

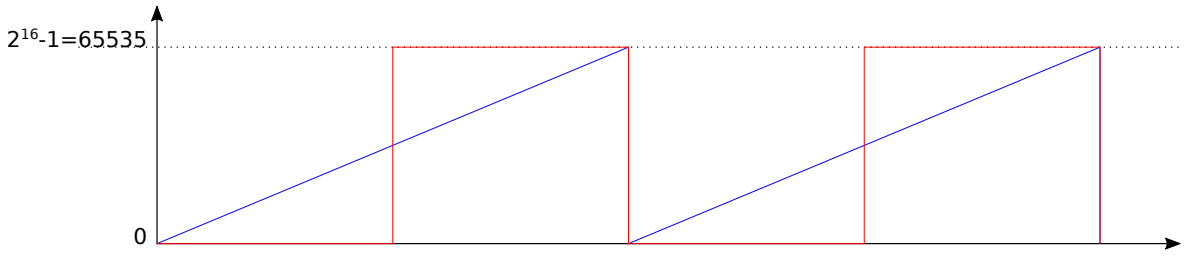


FIGURE 3 – Signal carré

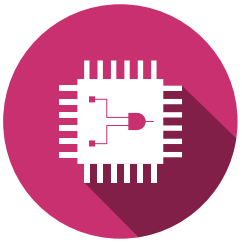
3.2.1 Todo

1. Déterminer la valeur binaire du nombre entier le plus petit, qui correspondra à la valeur inférieure du signal carré. Déterminer aussi la représentation binaire de la valeur supérieure du signal carré.
2. Ecrire l'architecture VHDL du bloc de transformation du signal en dents de scie en un signal carré, sachant que le circuit est combinatoire. La sortie est un **unsigned** de 16 bits.
3. Recompiler et simuler le bloc **waveformGen_tb**. Vérifier la forme du signal carré.

3.3 Triangle

La transformation du signal en dents de scie en un signal triangulaire se fait en "repliant la dent de scie vers le bas" lorsque le bit de poids fort du signal en dents de scie vaut '1'. Ce circuit est un circuit combinatoire. Lorsque le bit de poids fort du signal en dents de scie vaut '1', inverser tous les bits du signal en dent de scie. Comme le signal généré aura ainsi toujours son bit de poids fort à '0', il n'utilise que la moitié de la gamme à disposition. Il faudra donc multiplier le signal obtenu par deux, ce qui correspond à un décalage vers la gauche de la valeur binaire de sortie.

Il est proposé de définir un signal interne résultant de la première opération de repliement et utilisé



pour générer la sortie. Il est aussi plus lisible de définir un autre signal interne : le bit de poids fort du signal d'entrée.

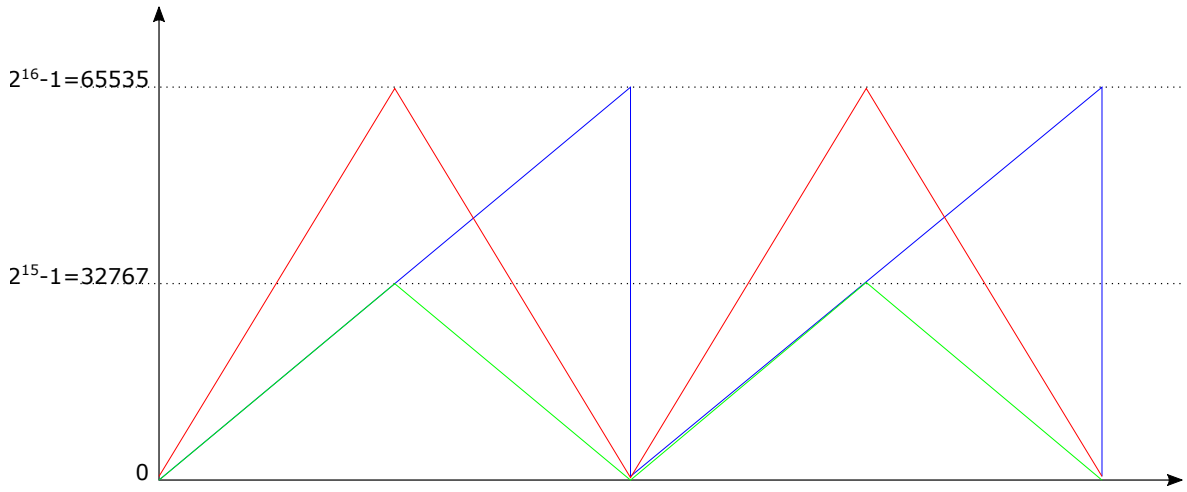


FIGURE 4 – Signal triangle

3.3.1 Todo

1. Ecrire l'architecture VHDL du bloc de transformation du signal en dents de scie en un signal triangulaire.
2. Recompiler et simuler le bloc **waveformGen_tb**. Vérifier la forme du signal triangulaire.

3.4 Polygone

La transformation de triangle à sinus peut se faire par une table de vérité (ROM) qui contient toutes ces valeurs. Cette solution, même si elle est la plus précise, est très coûteuse en matériel.

Ici, le signal triangulaire va être filtré pour ne conserver que sa fondamentale. Pour simplifier la tâche de filtrage, le signal triangulaire est transformé en un signal polygonal, dont la forme est plus proche du sinus désiré :

— La transformation de triangle à polygone commence en multipliant le triangle par 1.5.

— Après cela, il faut remplaçer toutes les valeurs plus petites que 1/8 de la gamme du triangle obtenu par cette valeur de 1/8 : on coupe ce qui est en-dessous. De même, on coupe aussi ce qui dépasse les 5/8.

— Finalement, on peut décaler le signal de 1/8 vers le bas.

La multiplication par 1.5 se fait par un décalage et une addition. Le résultat de la multiplication nécessite un bit de plus que le signal de départ. Après le décalage vers le bas, on peut ignorer le bit de poids fort et revenir au même nombre de bits qu'à l'entrée.

Les valeurs 1/8 et 5/8 sont relatives à la pleine échelle (Full Scale, FS) possible du signal et non à l'amplitude maximale du triangle.

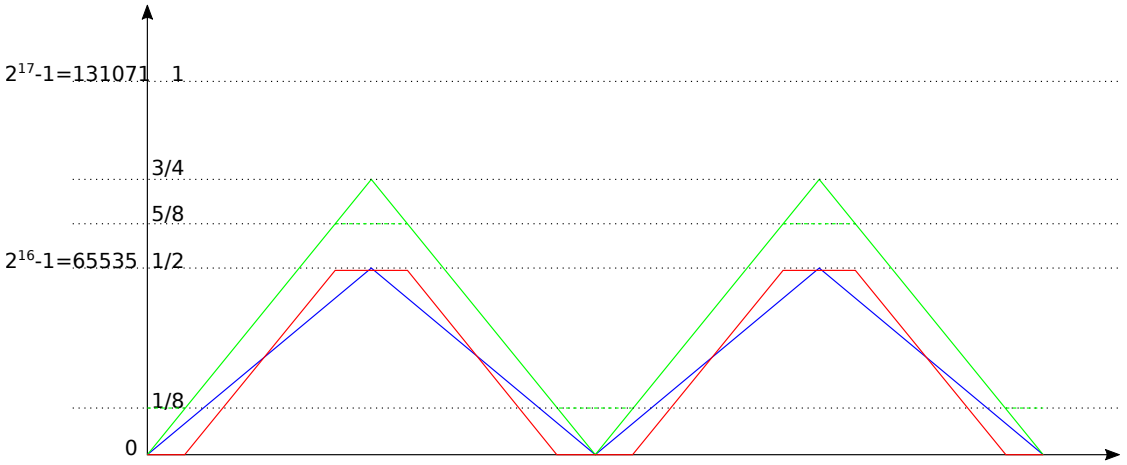


FIGURE 5 – Signal polygone

3.4.1 Todo

1. Ecrire l'architecture VHDL du bloc de transformation du signal triangulaire en un signal polygonal.
2. Recompiler et simuler le bloc **waveformGen_tb**. Vérifier la forme du signal polygonal.

3.5 Sinus

Le filtrage passe-bas se fait à l'aide d'un intégrateur réalisé par un accumulateur, sur le principe d'un filtre RC :

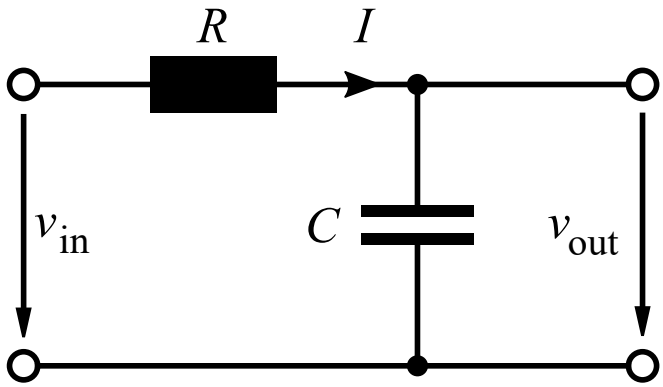


FIGURE 6 – Filtre RC

Les équations suivantes régissent le système :

—

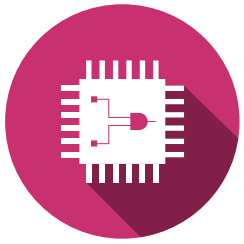
$$Vout = \frac{1}{C} * \int Idt \tag{1}$$

—

$$I = \frac{Vin - Vout}{R} \tag{2}$$

Donnant l'équation finale

$$Vout = \frac{1}{R * C} * \int (Vin - Vout)dt \tag{3}$$



On **intègre/accumule** donc le delta de tension, en lui appliquant un facteur.

Dans le monde numérique, à chaque période d'horloge, on additionne la valeur de l'entrée au contenu de l'accumulateur (intégration), et on leur soustrait la valeur de l'accumulateur décalée d'un certain nombre de bits. Le nombre de bits de décalage donne la fréquence de coupure : à chaque incrémentation de ce nombre, la fréquence de coupure est divisée par 2. L'accumulateur nécessite un nombre de bits égal à celui du signal d'entrée plus le nombre de bits de décalage. En sortie, il suffit de reprendre les bits de poids fort de l'accumulateur.

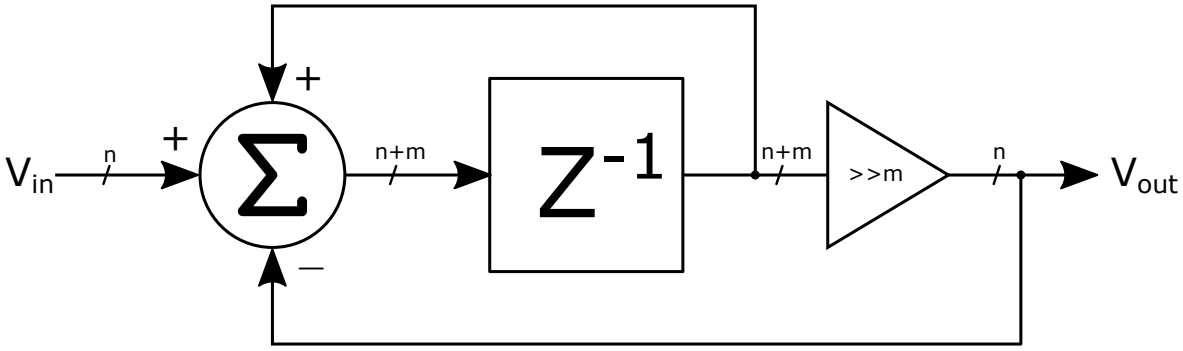


FIGURE 7 – Filtre numérique

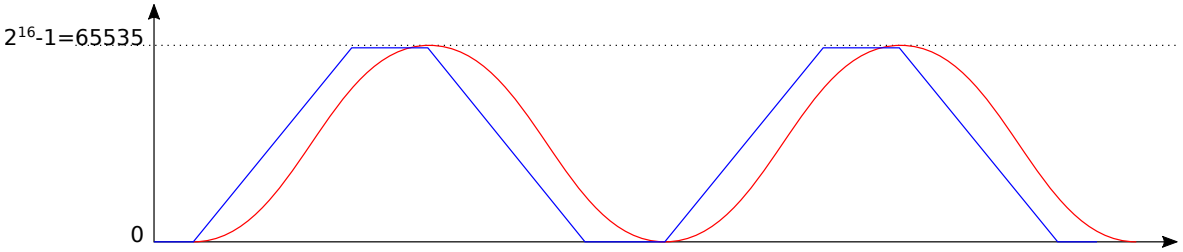


FIGURE 8 – Signal sinus

3.5.1 Todo

1. Ecrire l'architecture VHDL du filtre passe-bas de premier ordre.
2. Recompiler et simuler le bloc **waveformGen_tb**. Vérifier le sinus en sortie.
3. Modifier le nombre de bits de décalage du filtre passe-bas jusqu'à obtenir un signal sinusoïdal "satisfaisant".