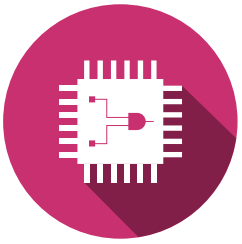


Funktionsgenerator mit Interpolation

Inhaltsverzeichnis

1 Einreichen

In diesem Labor geht es um Operationen mit Zahlen, die anhand der Berechnung der Interpolation von Punkten der Sinusfunktion eines digitalen Funktionsgenerators veranschaulicht werden. Die Berechnung der Interpolation basiert auf der Approximation der Funktion zwischen zwei gegebenen Punkten durch Polynomfunktionen dritter Ordnung. Diese Methode gehört zur großen Familie der **Splines**.

2 Start

Das Schema befindet sich in der Bibliothek **SineInterpolator**. Der Teststand ist unter **SineInterpolator_test**.

*Zur Erinnerung: Das Modellierungsprogramm muss über die Datei **splineInterpolator.bat** gestartet werden.*

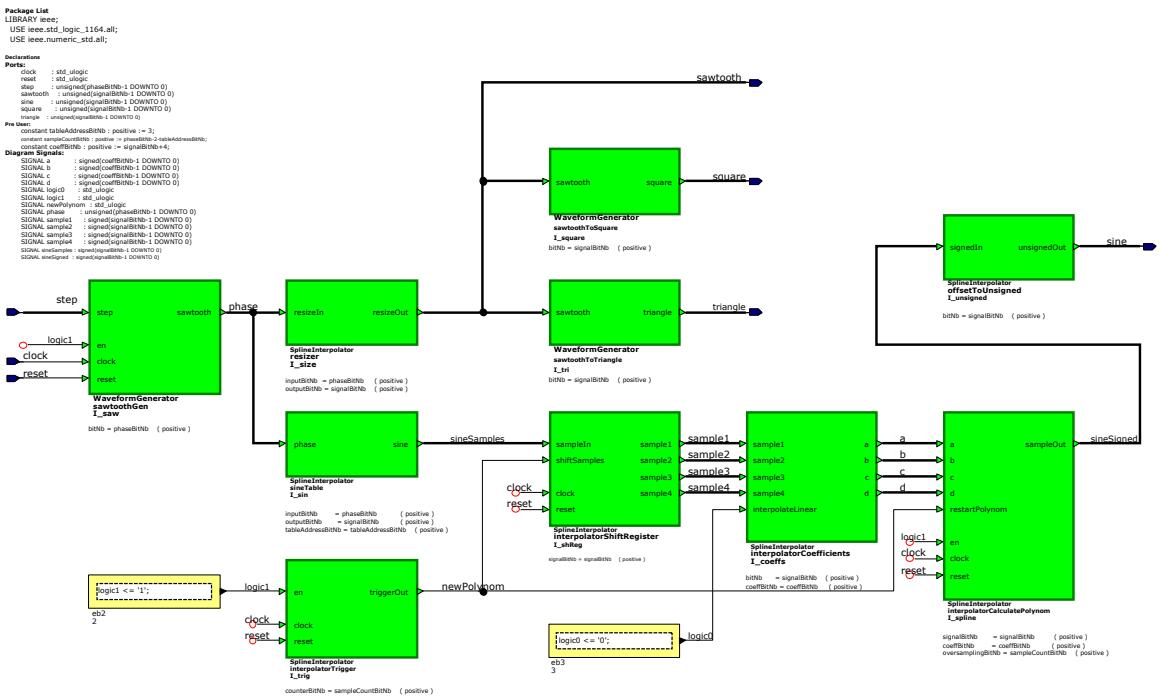


Abbildung 1: Funktionsgenerator mit Interpolation

3 Ziele

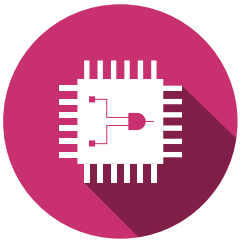
3.1 Sägezahn

In dieser Schaltung ist die Anzahl der Bits des **phase** signal (von **phaseBitNb**) nicht identisch mit der Anzahl der Ausgänge des Funktionsgenerators (von **signalBitNb**). Um die Blöcke zu verwenden, die die Signale Sägezahn, Dreieck und Rechteck liefern, ändert der Block **resizer** die Anzahl der Bits des Phasenzählers. 3 Fälle sind zu erwarten:

- die Anzahl der Bits der Ausgänge ist größer als die der Phase: Daher muss das Phasensignal in die höchstwertigen Bits des Sägezahnsignals kopiert werden
- die Anzahl der Bits der Ausgänge entspricht der Anzahl der Bits der Phase: Sie müssen nur das Phasensignal in das Sägezahnsignal kopieren
- die Anzahl der Bits der Ausgänge ist kleiner als die der Phase: hier müssen die höchstwertigen Bits des Phasensignals erhalten bleiben

3.1.1 Todo

1. Mit Hilfe des Herstellers **if ... generate**, schreiben Sie den VHDL-Code des Blocks **resizer**.
2. Den Block **sineGen_tb** kompilieren und simulieren. Überprüfen Sie, ob der Signalgrößen-modifikator für alle drei Betriebsfälle richtig funktioniert.



3.2 Sinustabelle

Die Interpolation erfolgt aus einer Tabelle, die 8 Sinuswerte für ein Viertel der Periode enthält. Um über die gesamte Periode zu arbeiten, wird das Sägezahnsignal vom Zähler als Phase des Sinus betrachtet und wie folgt verwendet:

- das höchstwertige Bit signalisiert den Vorzeichenwechsel der Sinuswerte
- das nächste Bit signalisiert eine Änderung, die an der Phase vorgenommen werden muss, um die Tabelle in umgekehrter Richtung zu lesen (zweites und viertes Viertel der Periode)
- die folgenden 3 Bits (**tableAddressBitNb = 3**) werden verwendet, um die Werte in der Tabelle zu adressieren
- die anderen Bits werden ignoriert

Die Adresse der Sinustabelle wird aus den drei Bits gezogen, die auf die beiden höchstwertigen Bits der Phase folgen. Sie führt die Sequenz "0, 1, 2, 3, 4, 5, 6, 7, 0, 7, 6, 5, 4, 3, 2, 1, 0, 1, 2, 3, ...äus. Die Sinustabelle enthält die Hexadezimalwerte 0000 oder 7FFF (je nach dem Wert des zweithöchsten Bits), 18F9, 30FB, 471C, 5A82, 6A6D, 7641 und 7D89. Wenn das höchstwertige Bit der Phase den Wert '1' hat, ändert der aus der Tabelle gelesene Wert sein Vorzeichen und ergibt die Ausgabe.

3.2.1 Todo

1. Schreiben Sie die VHDL-Architektur des Sinusgenerators.
2. Den Block **sineGen_tb** kompilieren und simulieren. Überprüfen Sie, ob der Sinusgenerator ordnungsgemäß funktioniert.

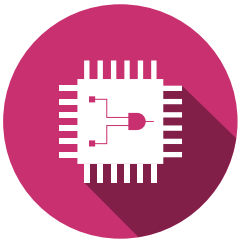
3.3 Interpolation

Die Interpolation erfolgt, indem eine Polynomfunktion der Ordnung 3 zwischen jedem Punkt in der Sinustabelle verknüpft wird: $\mathbf{y = a \cdot k3 + b \cdot k2 + c \cdot k + d}$. Während der Berechnung wird der aktuelle Punkt in der Sinustabelle bei k = 0 definiert, der vorherige Punkt bei k = -1 und der nächste Punkt bei k = 1. Wir werden den Wert des Polynoms für $\mathbf{n = 2m}$ Punkte berechnen, die zwischen 0 und 1 liegen. Um die Kontinuität zwischen dem Polynom, das die Strecke $-1 < k < 0$ verbindet, und dem Polynom, das $0 < k < 1$ verbindet, zu gewährleisten, geben wir die Ableitung im Punkt y(k=0) als gleich der Steigung zwischen den Punkten y(k=-1) und y(k=1) an. Dasselbe gilt für alle Punkte der zu interpolierenden Funktion.

Das Segment $0 < k < 1$ unterliegt also den folgenden 4 Bedingungen:

- y(k=0) = y0: geht das Polynom durch den Punkt (k=0, y=y0)
- y(k=1) = y1: geht das Polynom durch den Punkt (k=1, y=y1)
- y'(k=0) = [y(k=1) - y(k=-1)] / 2: die Ableitung an der Stelle (k=0) ist die Steigung zwischen y(k=-1) und y(k=1)
- y'(k=1) = [y(k=2) - y(k=0)] / 2: die Ableitung an der Stelle (k=1) ist die Steigung zwischen y(k=0) und y(k=2)

Die vier Bedingungen dienen uns dazu, die vier Koeffizienten a, b, c und d des Ploynoms aus den vier Punkten y(k=-1) bis y(k=2) der zu interpolierenden Kurve zu bestimmen. Die Lösung dieses Gleichungssystems liefert uns die folgenden Werte der Koeffizienten:



- 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)

Der Einfachheit halber berechnen wir den doppelten Wert der Koeffizienten und teilen anschließend den Wert des berechneten Polynoms durch 2. Bei jedem neuen Wert des Originalsignals (hier aus der Sinustabelle) müssen wir die Koeffizienten des Polynoms neu berechnen. Dann, in jeder Taktperiode und während wir auf die Ankunft des nächsten Wertes warten, berechnen wir die Polynomfunktion, um den aktuellen Wert des Ausgangsmusters zu bestimmen.

3.4 Generator für Sequenzimpulse

Diese Schaltung liefert bei jedem Polynomwechsel, d. h. für jedes neue Kurvensegment, einen Impuls, der eine Taktperiode lang ist. Dies geschieht jeweils **2n** Taktperioden, wobei **en = '1'**, mit **n = sampleCountBitNb = phaseBitNb - 2 - tableAddressBitNb**. In unserem Laborbeispiel sind **phaseBitNb = 10** und **sampleCountBitNb = 5**.

3.4.1 Todo

1. Schreiben Sie die VHDL-Architektur des Sequenzimpulsgenerators.

3.5 Schieberegister

Bei jedem Synchronisationspuls **shiftSamples** speichert die Schaltung das neue Sample der zu interpolierenden Funktion, **sampleIn**, und speichert es als letzten Samplewert, **sample4**. Gleichzeitig verschiebt sie das Sample **sample4** in **sample3**, usw....

Verwenden Sie ein Array aus vorzeichenbehafteten Zahlen für die interne Beschreibung des Schieberegisters.

3.5.1 Todo

1. Schreibe die VHDL-Architektur des Schieberegisters, das die letzten 4 Punkte der zu interpolierenden Funktion speichert.

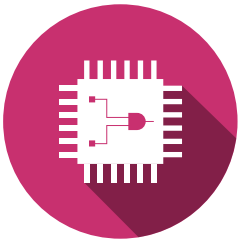
3.6 Berechnung der Koeffizienten

Die Koeffizienten werden wie folgt berechnet:

- 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. Durchsuchen Sie die Bibliothek **ieee.numeric_std** und überprüfen Sie, wie es sich auswirkt, wenn ein Signal vom Typ **signed** mit einer ganzen Zahl multipliziert wird. Überprüfen Sie



auch, ob die Funktion **resize** funktioniert. Sie ist hier sehr nützlich, um die Anzahl der Bits von Signalen zu erzwingen.

2. Schreibe die VHDL-Architektur des Blocks, der die Koeffizienten des Polynoms in Abhängigkeit vom Wert der letzten 4 Punkte der zu interpolierenden Funktion berechnet.
3. Erklären Sie die getroffene Wahl **coeffBitNb := signalBitNb+4**.

3.7 Berechnung des Polynoms

Das Polynom wird iterativ berechnet. So gilt für eine Funktion erster Ordnung **y(x) = a-x + b** der nächste Wert **y(x+eps) = y(x) + a-eps**. Dies wird berechnet, indem **u = a-eps, y(0) = b** initialisiert wird und **y(i+1) = y(i) + u** bei jeder neuen Taktperiode berechnet wird. Für eine Funktion zweiter Ordnung **y(x) = a-x2 + b-x + c**, was **y(x+eps) = y(x) + u(x)** ergibt, wobei **u(x) = 2-a-eps-x + (a-eps2 + b-eps)**. Das Inkrement **u(x)** wird auf die für eine Funktion erster Ordnung definierte Weise berechnet. Dies wird also berechnet, indem **v = 2-a-eps2, u(0) = a-eps2 + b-eps, y(0) = c** initialisiert wird und **y(i+1) = y(i) + u(i) und u(i+1) = u(i) + v** in jeder neuen Taktperiode berechnet wird. Die Berechnung der Funktion dritter Ordnung erfolgt auf ähnliche Weise. Für die numerische Berechnung wird **eps** so gewählt, dass es einer negativen Potenz von 2 entspricht, wodurch sie durch eine Verschiebung berechnet werden kann. Ebenso berechnen wir, um bei ganzen Zahlen zu bleiben, die Funktion **y(i)** multipliziert mit einer Potenz von 2 und eine abschließende Verschiebung liefert uns den resultierenden Wert der Polynomproben.

Unser Algorithmus sieht daher wie folgt aus:

- Wenn eine neue Stichprobe eintrifft, müssen die Anfangswerte berechnet werden, die zur iterativen Entwicklung des Polynoms verwendet werden.
- Zwischen zwei Eingabestichproben erfolgt die Berechnung der Polynomfunktion durch Additionen mithilfe der iterativen Gleichungen.

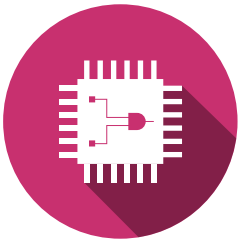
Die Anfangswerte sind wie folgt:

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

Die iterative Berechnung des Polynoms erfolgt wie folgt:

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

Es ist offensichtlich, dass aufgrund der ortextbf2m-Verschiebungen die Anzahl der Bits der internen Signale größer sein muss als die Anzahl der Bits der Eingangs- oder Ausgangsabtastwerte. Schätzen Sie anhand des Werts des Offsets **m = oversamplingBitNb** die Größe, die diese Signale benötigen. Deklarieren Sie sie mit 8 zusätzlichen Bits, um sicherzustellen, dass es nicht zu einem



Überlauf kommt.

3.7.1 Todo

1. Schreibe die VHDL-Architektur des Blocks, der den Wert des Funktionsinterpolationspolynoms bei jeder neuen Taktperiode berechnet.
2. Kompilieren und simulieren Sie den Block **sineGen_tb**. Überprüfen Sie die Form des Sinussignals nach der Interpolation.
3. Reduzieren Sie die Anzahl der Bits in den Signalen des Schaltkreises, wo dies möglich ist.