Nombres binaires fractionnaires à virgule fixe: représentation et opérations

Nombres binaires fractionnaires à virgule fixe
Sujets de ce thème

- Systèmes de numération positionnels pour nombres entiers et fractionnaires
- Conversions entre les bases
- Addition de nombres binaires à virgule fixe
- Nombres binaires signés à virgule fixe
- Multiplication de nombres à virgule fixe
- Nombres à virgule fixe en VHDL

Système de numération positionnel: nombres entiers

- Le système de numération le plus usité est le système positionnel.
- Un système de numération positionnel nécessite une base *R*, et il faut *R* chiffres différents.
  - 2 chiffres (0 et 1) en base 2
  - 10 chiffres (0 à 9) en base 10
  - 16 chiffres (0 à 9 et A à F) en base 16
- Un nombre X est représenté par une somme de puissances de la base pondérées avec la valeur des chiffres.

$$X = x_{n-1}x_{n-2}x_{n-3}x_{n-4}\ldots x_1x_0$$
$$= \sum_{i=0}^{n-1} x_i R^i$$

$$36057_{10} = 3\times10000 + 6\times1000 + 0\times100 + 5\times10 + 7$$

$$1011_2 = 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$$
$$= 8 + 2 + 1 = 11_{10}$$

Représentation de nombres fractionnaires en virgule fixe

- Les puissances utilisées peuvent être négatives afin de représenter la partie fractionnaire d'un nombre.
- Une virgule (un point dans les pays anglophones) sépare les parties entière et fractionnaire d'un nombre.
- En utilisant *n* chiffres pour la partie entière et *m* chiffres pour la partie fractionnaire, on a :

$$X = x_{n-1}x_{n-2}x_{n-3}x_{n-4}\ldots x_1x_0, x_{-1}x_{-2}x_{-3}\ldots x_{m-2}x_{m-1}x_m$$
$$= \sum_{i=0}^{n-1} x_i R^i + \sum_{k=1}^{m} x_{-k} R^{-k}$$

$$36{,}057_{10} = 3\times 10 + 6\times 1 + 0\times 0{,}1 + 5\times 0{,}01 + 7\times 0{,}001$$

$$10{,}11_2 = 1\times 2^1 + 0\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2}$$
$$= 2 + 0{,}5 + 0{,}25 = 2{,}75_{10}$$

Conversion de la base 2 à la base 10

- Convertir un nombre fractionnaire de la base 2 à la base 10: appliquer l'équation positionnelle
- Exemples
  - $0,1 = 1 \text{ X } 2^{-1} = 0,5$
  - $0,101 = 1 \text{ X } 2^{-1} + 1 \text{ X } 2^{-3} = 0,625$
  - 11,1101=

Conversion de la base 10 à la base 2
Exprimer un nombre fractionnaire avec un nombre fini de bits

1. Traiter la partie entière
   - Divisions successives en conservant les restes
2. Traiter la partie fractionnaire
   - Multiplication par 2^N avec arrondi

Exemple

- Exprimer 27,75 avec 6 chiffres pour la partie entière et 4 chiffres pour la partie fractionnaire.

Conversion de la base 10 à la base 2
Exprimer un nombre fractionnaire avec un nombre fini de bits

Exemple

- Exprimer π en représentation binaire non signée avec 16 chiffres, dont 8 pour la partie entière.

$$\pi \approx 3.14159265358979 \ldots$$

- La partie entière est 3, soit 00000011 en binaire sur 8 chiffres.
- La partie fractionnaire est 0.14159265358979 …
  On calcule:

$$0.14159265358979 \ldots \times 2^8 = 36.248 \ldots$$

- Donc $\pi \approx 3 + 36/256 = 00000011.00100100$

→ Convertir une fraction décimale en binaire consiste à l'approximer par une fraction dont le dénominateur est une puissance de deux.

Addition de nombres binaires à virgule fixe

- Il faut tout d'abord aligner les virgules.
- Ensuite il faut choisir le nombre de bits à utiliser pour représenter les deux nombres: le nombre maximal de bits des parties entière et fractionnaire des deux opérandes.
- L'addition se fait bit par bit avec propagation des retenues.
- Pour éviter tout débordement, la somme nécessite un bit de plus que les opérandes.

Exemples d'arithmétique non signée.

3,25	11,01	3,0625	11,0001
+ 13,125	1101,001	+ 13,125	1101,001
3,25	00011,010	3,0625	00011,0001
+ 13,125	01101,001	+ 13,125	01101,0010
16,375	10000,011	16,1875	10000,0011

Nombres binaires signés à virgule fixe

- La représentation de nombres binaires signés à virgule fixe suit la représentation de nombres entiers signés à complément à deux.
  - Le bit le plus significatif a un poids négatif.
  - Un 1 au bit le plus significatif indique un nombre négatif, un 0 indique un nombre positif.
  - Pour changer le signe, on inverse tous les bits et on ajoute 1 à la position la moins significative.
- L'extension du signe se fait de la même façon que pour les nombres entiers.
- La soustraction de nombres binaires signés à virgule fixe se fait comme pour les nombres entiers.

1,5	001,1
inv(1,5)	110,0
+ 1×2^{-1}	000,1
-1,5	110,1 = -4+2+0,5

-4,1875	1011,1101
inv(-4,1875)	0100,0010
+ 1×2^{-4}	0000,0001
4,1875	0100,0011

1,125	1,001
inv(1,125)?	0,110
+ 1/8	0,001
0,875	0,111

Erreur: +1,125 ne peut pas être représenté avec seulement 1 bit pour la partie entière en notation signée

1,125	01,001
inv(1,125)	10,110
+ 1/8	00,001
-1,125	10,111 = -2+7/8

Multiplication de nombres en virgule fixe

- La multiplication se fait comme pour les nombres entiers.
- On fait d'abord la multiplication sans tenir compte des virgules, puis à la fin on ajuste la virgule à la bonne place.

1,75	01,11
× 2,5	010,1
	0111
	00000
	011100
	0000000
	0100011
4,375	0100,011

On observe que, si on enlève les virgules, la multiplication revient à faire 7 × 5 = 35.
Le produit doit être décalé de 3 bits vers la droite, soit une division par 8.
On a bien 35 ÷ 8 = 4,375.

Types VHDL à utiliser pour représenter et manipuler des nombres à virgule fixe

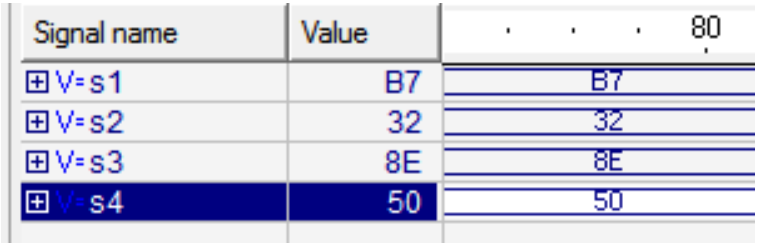
- La norme VHDL 2008 inclut les packages `fixed_pkg`, et `fixed_generic-pkg`.
- Les packages contiennent des définitions de types, des surcharges d'opérateurs et des fonctions pour traiter les nombres fractionnaires signés et non signés.
- Les types `ufixed` et `sfixed` sont utilisés pour des nombres non signés et signés, respectivement.
- La déclaration d'un objet de type `ufixed` et `sfixed` inclut le nombre de bits à utiliser pour les parties entières et fractionnaires. La virgule est située à droite de l'indice 0.

```
library ieee;
use ieee.fixed_pkg.all;

...

signal s1 : sfixed(3 downto -4);
signal u1 : ufixed(4 downto -1);
```

Assignations de valeurs au type sfixed

```
process
variable s1, s2 : sfixed(3 downto -4);
variable s3 : sfixed(3 downto -4) := to_sfixed(-7.125, 3, -4);
variable s4 : sfixed(3 downto -4) := to_sfixed(5, 3, -4);
begin

   s1 := "10110111";
   s2 := to_sfixed(3.1415926, s2);

   report real'image(to_real(s1));
   report real'image(to_real(s2));
   report real'image(to_real(s3));
   report real'image(to_real(s4));

   report integer'image(to_integer(s1));
   report integer'image(to_integer(s2));
   report integer'image(to_integer(s3));
   report integer'image(to_integer(s4));

   wait for 1000 ns;

end process;
```

```
run 100 ns

# EXECUTION:: NOTE  : -4.562500e+000
# EXECUTION:: NOTE  : 3.125000e+000
# EXECUTION:: NOTE  : -7.125000e+000
# EXECUTION:: NOTE  : 5.000000e+000

# EXECUTION:: NOTE  : -5
# EXECUTION:: NOTE  : 3
# EXECUTION:: NOTE  : -7
# EXECUTION:: NOTE  : 5
```



Arithmétique à virgule fixe en VHDL

- Les packages VHDL pour la manipulation des nombres en virgule fixe incluent des définitions des fonctions arithmétiques: +, -, *, /, `mod`, réciproque, `abs`, etc.
- Toutes les opérations ne sont pas synthétisables.
- La taille des résultats suit une convention différente de celle du package `numeric_std` pour les entiers de type `signed` et `unsigned`:
  - Pour `numeric_std`, un débordement est possible
  - Pour `fixed_pkg`, le débordement n'est pas possible: un bit est explicitement ajouté

Opération	Indices du résultat
A + B, A – B	max(A'left, B'left) + 1 downto min(A'right, B'right)
A * B	A'left + B'left + 1 downto A'right + B'right
abs(A), -A	A'left + 1 downto A'right

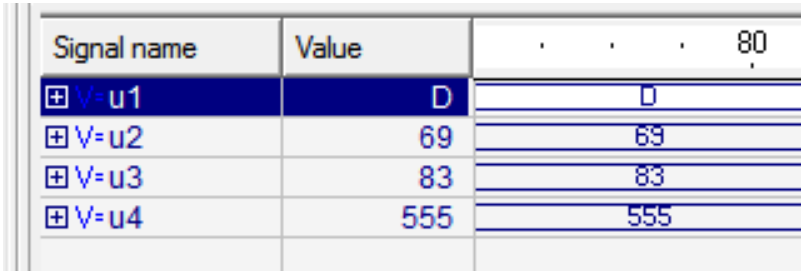
Arithmétique à virgule fixe en VHDL

```
process
variable u1 : ufixed(1 downto -2);
variable u2 : ufixed(3 downto -3);
variable u3 : ufixed(4 downto -3);
variable u4 : ufixed(5 downto -5);
begin

   u1 := to_ufixed(3.25, u1);
   u2 := to_ufixed(13.125, u2);
   u3 := u1 + u2;
   u4 := u1 * u2;

   report real'image(to_real(u1));
   report real'image(to_real(u2));
   report real'image(to_real(u3));
   report real'image(to_real(u4));

   wait for 1000 ns;

end process;
```

3,25	11,01
+ 13,125	1101,001
3,25	00011,010
+ 13,125	01101,001
16,375	10000,011

13,125	1101,001
× 3,25	× 11,01
	1101001
	00000000
	110100100
+	1101001000
(1365 / 2^5)	10101010101
42,65625	101010,10101

```
# EXECUTION:: NOTE   : 3.250000e+000
# EXECUTION:: NOTE   : 1.312500e+001
# EXECUTION:: NOTE   : 1.637500e+001
# EXECUTION:: NOTE   : 4.265625e+001
```



Arithmétique à virgule fixe en VHDL
Le cas spécial d'un accumulateur/compteur

- Comme l'addition produit un résultat qui occupe suffisamment de bits pour éviter un débordement, il faut utiliser une astuce dans le cas de la modélisation d'un accumulateur.
- La fonction `resize()` permet de redimensionner un nombre de type `ufixed` ou `sfixed`.
- Elle prend 4 paramètres:
  - la quantité à redimensionner
  - la dimension finale (ou un objet de cette dimension)
  - le style de débordement (`fixed_wrap` pour débordement normal, `fixed_saturate` pour la saturation à la valeur maximale)
  - le style d'arrondi (`fixed_truncate` pour la fonction plancher, `fixed_round` pour l'arrondi)

```
process(clk, reset)
variable compte : ufixed(1 downto -2);
variable compte2 : ufixed(2 downto -2);
begin
   if reset = '0' then
        compte := to_ufixed(0, compte);
   elsif rising_edge(CLK) then
        compte := resize(
                  compte + 0.25,
                  compte,
                  fixed_wrap,
                  fixed_round);
   end if;
   report real'image(to_real(compte));
end process;
```

Vous devriez maintenant être capable de …

- Décrire et utiliser la représentation de nombres fractionnaires à virgule fixe. (B2, B3)
- Décrire et utiliser l'addition, la soustraction et la multiplication pour ces nombres. (B2, B3)
- Décrire et utiliser les ressources de VHDL pour l'arithmétique en virgule fixe. (B2, B3)

Code	Niveau (http://fr.wikipedia.org/wiki/Taxonomie_de_Bloom)
B1	Connaissance - mémoriser de l'information.
B2	Compréhension – interpréter l'information.
B3	Application – confronter les connaissances à des cas pratiques simples.
B4	Analyse – décomposer un problème, cas pratiques plus complexes.
B5	Synthèse – expression personnelle, cas pratiques plus complexes.