

REPREZENTAREA NUMERELOR SI ARITMETICA LA PROCESOARELE DE SEMNAL

1. Generalități

Fără a avea informații legate de convenția folosită la reprezentarea numerelor binare în calculator nu se poate atribui o valoare exactă unui număr binar, acesta reprezentând o înșiruire de biți. De exemplu, ce valoare zecimală are $10111011B = X$? Este pozitiv sau negativ ? Aceasta depinde de reprezentarea folosită.

La alegerea unui Procesor Digital de Semnal (DSP) pentru o anumită aplicație una dintre caracteristicile importante la luarea deciziei o constituie modul de reprezentare (binar) al datelor de către procesor. Din acest punct de vedere DSP-urile de pe piață se pot clasifica după cum arată fig. 1.

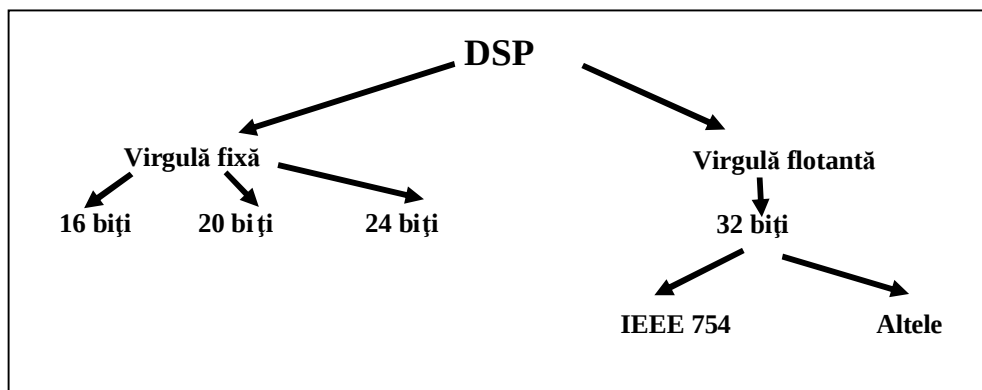


Fig. 1 Reprezentarea numerelor la DSP-urile uzuale

Aritmetica în virgulă fixă a fost folosită de către primele DSP-uri și la ora actuală mai este încă predominantă la multe dintre DSP-urile uzuale.

DSP-urile în virgulă fixă pot reprezenta numerele fie ca:

- întregi (aritmetica întreagă) – fiind folosiți de DSP la operații de control, calcul de adrese sau alte operații care nu implică semnale, sau
- fracționar (aritmetica fracționară) cu valori între -1 și +1 utilă în calculele legate de semnale.

Algoritmii și hard-ul folosit la implementarea aritmeticii fracționare sunt virtual identici cu cei folosiți la aritmetica întreagă. Diferența principală între cele două aritmetici apare la modul de folosire a rezultatelor operațiilor de înmulțire. Majoritatea DSP-urilor în virgulă fixă acceptă cele două aritmetici.

Intrucât sistemele numerice folosesc pentru reprezentarea numerelor un cuvânt de o anumită lungime operațiile aritmetice se vor executa cu o anumită precizie (limitată). Numerele vor fi astfel reprezentate pe un cerc (inel) în loc de axa reală infinită, situație în care poate să apară depășirea aritmetică.

În cazul **DSP-urilor în virgulă mobilă**, valorile sunt reprezentate de o mantisă și de un exponent conform relației $\text{mantisă} \cdot 2^{\text{exponent}}$. Mantisă este în general o fracție în intervalul -1.0 și +1.0, în timp ce exponentul este un întreg care reprezintă numărul de poziții cu care trebuie deplasat stânga punctul binar (termen definit analog cu punctul zecimal) pentru a obține valoarea reprezentată.

Procesoarele în virgulă mobilă sunt mai ușor de programat decât cele în virgulă fixă, dar sunt și mai scumpe. Acest lucru se datorează complexității sporite a circuitului care se traduce într-o suprafață mai mare a chip-ului. Ușurinta programării acestor procesoare este dată și de faptul că programatorul nu mai trebuie să gestioneze cazurile de depășire (overflow) ale acumulatorului ca și în cazul procesoarelor în virgulă fixă (această gestionare înseamnă scalarea periodică a rezultatului în diverse faze ale prelucrării).

Majoritatea aplicațiilor de cost redus utilizează procesoare în virgulă fixă. Programarea în acest caz nu are în general nici o dificultate, deoarece în faza de simulare numerică a algoritmului (care precede faza de dezvoltare) se pot detecta toate situațiile în care este necesară corecția rezultatului pentru a se evita saturarea. Pentru o mai bună înțelegere a acestor fenomene de overflow, vom prezenta două situații care se întâlnesc în cazul procesoarelor DSP sau a celor ce suportă tehnologia MMX, și anume: wraparound (modulo 2^n) și saturarea.

Pentru exemplificare, se prezintă în cele ce urmează o aplicație pe imagini: având imaginea originală pe nivele de gri (Fig. 2a) se poate obține efectul de wraparound (Fig. 2b) și de saturare (Fig. 2c) folosind următoarele prelucrări:

- dacă rezultatul adunării între pixeli și o valoare produce depășire, rezultatul este trunchiat, luându-se în considerare doar biții cei mai puțin semnificativi (efectul wraparound), datorită limitării la n biți;
- dacă rezultatul adunării produce o depășire, apare situația de saturare și astfel rezultatul este limitat la valoarea maximă a domeniului;



Fig2. a) Imaginea originală pe nivele de gri; b) Influența operației modulo 2^8 ; c) Influența saturării.

Numerele mici reprezintă zone cu nivele de gri închise la culoare (negru), în timp ce numerele mari reprezintă arii de culoare deschisă (alb). Folosind 8 biți pentru reprezentarea valorilor pixelilor din imagine, se va obține gama numerelor [0; 255], cu valoarea 0 pentru negru și 255 pentru alb.

Pentru a ilumina imaginea originală (Fig. 2a) se poate aduna un număr pozitiv întreg (de exemplu $64_{10}=40_{16}$) la fiecare pixel din imagine.

În cazul efectului de wraparound (modulo 2^8), dacă există depășire (adică valoarea unui pixel depășește pragul maxim de 255), rezultatul final va fi trunchiat a.î. doar biții cei mai puțin semnificativi vor fi reținuți. De exemplu, dacă adunăm 64_{10} la 250_{10} (aproape alb), vom obține:

250 zecimal	1111.1010 binar	
+ 64 zecimal	+ 0100.0000 binar	
-----	-----	
= 314 zecimal	= 10011.1010 binar	- se produce depasire (overflow)
= 58 zecimal	= 0011.1010 binar	- se retin doar cei mai putin semnificativi 8 biti

Rezultatul este 58_{10} , producând o zonă închisă la culoare (aproape negru) în locul uneia mai deschise la culoare, cum era de așteptat. S-a obținut un efect invers celui dorit, deci o inversare de nuanță, deoarece valorile de nuanță deschisă au devenit zone de nuanță închisă.

În cazul efectului de saturare, prin adăugarea unei valori fiecărui pixel din imaginea originală, zonele deschise la culoare vor deveni zone pur albe. Pragul de saturare în acest caz este valoarea maximă reprezentabilă pe 8 biți, 255.

0x00	0x00	0x57	0x7F	0xAB	0xAB	0xC8	0xFF
+	+	+	+	+	+	+	+
0x40	0x40	0x40	0x40	0x40	0x40	0x40	0x40
=	=	=	=	=	=	=	=
0x40	0x40	0x97	0xBF	0xEB	0xEB	0x03	0x3F

Fig. 3 a) obținerea valorilor pixelilor în situația modulo 2^8

0x00	0x00	0x57	0x7F	0xAB	0xAB	0xC8	0xFF
+	+	+	+	+	+	+	+
0x40	0x40	0x40	0x40	0x40	0x40	0x40	0x40
=	=	=	=	=	=	=	=
0x40	0x40	0x97	0xBF	0xEB	0xEB	0xFF	0xFF

b) obținerea valorilor pixelilor în situația de saturare

2. Reprezentarea întregă

Pe un cuvânt de n biți se pot reprezenta 2^n numere plasate în gama numerelor egal distanțate cu pasul de cuantizare $q=1$.

Reprezentarea binară fără semn, pentru un număr reprezentat pe n biți este: $B_2 = b_{n-1} b_{n-2} \dots b_1 b_0$ cu valoarea zecimală: $B_{10} = b_{n-1} \cdot 2^{n-1} + \dots + b_1 \cdot 2 + b_0$ aflat în gama $[0, 2^n - 1]$. Se poate vedea în Fig. 4 “roata numerelor” pe care se pot verifica proprietățile adunării pentru numerele reprezentate pe un număr finit de biți, în acest caz $n=4$.

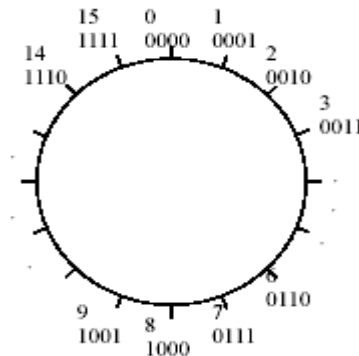


Fig. 4 “Roata numerelor” fără semn reprezentate pe 4 biți

Exemplu. Folosind reprezentarea pe 4 biți fără semn se incrementează cel mai mare număr reprezentabil astfel:

$$\begin{array}{rcl} 1111_2 & = & 15_{10} \\ 1_2 & = & 1_{10} \end{array}$$

$1 | 0000_2 = 0_{10}$, numărul 10000_2 nu mai este pe 4 biți, a apărut depășire!

↖ Bit de transport

Numerele binare fără semn pe n biți reprezintă un sistem modulo 2^n !

Pentru a reprezenta **numerele binare negative** (deci cu semn) se utilizează reprezentarea în complement față de 2. Un număr în complement față de 2 pe n biți este: $B_2 = b_{n-1} b_{n-2} \dots b_1 b_0$, cu valoarea zecimală:

$B_{10} = -b_{n-1} \cdot 2^{n-1} + \dots + b_1 \cdot 2 + b_0$ aflat în gama $[-2^{n-1}, 2^{n-1} - 1]$.

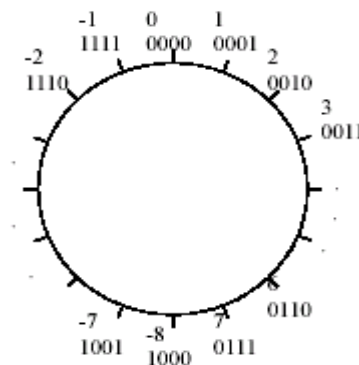


Fig. 5. “Roata numerelor” cu semn reprezentate pe 4 biți

Observație. La reprezentarea zecimală numerele 100_{10} și 0100_{10} sunt identice, dar la reprezentarea în complement față de 2 pentru numerele 1010_2 și 01010_2 nu putem presupune același lucru. De ce? La reprezentarea zecimală se folosește pentru semn un caracter special (-), pe când la reprezentarea binară semnul este dat de bitul de semn. De aceea, valoarea zecimală a numărului 1010_2 depinde de lungimea pe care este reprezentat. Dacă este pe 5 biți are valoarea 10_{10} , iar dacă este pe 4 biți are valoarea -6_{10} .

Proprietăți

- Înmulțirea a două numere binare întregi, în virgulă fixă, reprezentate pe n biți dă un rezultat pe $2n$ biți
- Adunarea a două numere reprezentate pe n biți poate da un rezultat pe $n+1$ biți

Înmulțirea a doi întregi în virgulă fixă duce cu mare probabilitate la depășire.

La N adunări succesive pentru a nu avea depășire sunt necesari $\log_2(N)$ biți suplimentari.

Exemplu. Folosind reprezentarea în virgulă fixă pe 4 biți să se facă înmulțirea: 3_{10} cu 2_{10} .

Observație. Rezultatul este pe 8 biți și a nu se uita extensia semnului la lungimea cuvântului rezultatului.

$$\begin{array}{rcl}
 1101 & = & -3_{10} \\
 0010 & = & 2_{10} \\
 \hline
 00000000 \\
 1111101 \\
 000000 \\
 00000 \\
 \hline
 11111010 & = & -6_{10}
 \end{array}$$

Dacă se revine la reprezentarea pe 4 biți rezultatul va fi $1010_2 = -6_{10}$, deci este corect.

Exemplu. Să se facă înmulțirea în virgulă fixă pe 4 biți a numerelor -3_{10} și 6_{10} .

$$\begin{array}{rcl}
 1101 & = & -3_{10} \\
 0110 & = & 6_{10} \\
 \hline
 00000000 \\
 1111101 \\
 111101 \\
 00000 \\
 \hline
 11101110 & = & -18_{10}
 \end{array}$$

Dacă se revine la reprezentarea pe 4 biți, rezultatul va fi $1110_2 = -2_{10}$, deci nu este corect.

Proprietăți. Orice secvență de operații a cărei rezultat final se poate reprezenta corect în gama dată se poate calcula corect chiar dacă apar depășiri la rezultatele intermediare.

Exemplu. Folosind reprezentarea pe 4 biți cu semn (gama $[-8, 7)$) să se calculeze expresia: $7+6-8 = 5$

$$\begin{array}{rcl}
 0111+ & 7+ \\
 0110 & 6 \\
 \hline
 1101 + & -3+ \text{ (depășire)} \\
 1000 & -8 \\
 \hline
 0101 & 5 \text{ (corect)}
 \end{array}$$

3. Reprezentarea fracționară

Pentru a se evita problema depășirilor în cazul înmulțirii, reprezentarea fracționară în virgulă fixă limitează sau normalizează numerele în gama $[-1.0, 1.0)$, caz în care problema este rezolvată.

Excepție. $(-1)*(-1) = +1$! care nu aparține gamei $[-1, +1)$.

La adunare și scădere depășirea este totuși posibilă!

Observații.

1. Rezultatul înmulțirii este trunchiat pierzând din precizie prin suprimarea biților mai puțin semnificativi.
2. Reprezentarea fracționară pe n biți se obține prin deplasarea punctului cu $n-1$ poziții spre stânga, adică reprezentarea conține bit de semn, punct radix și $n-1$ biți fracționari, deci: $B_{2\text{frac}} = b_0 b_1 \dots b_{n-2} b_{n-1}$, cu valoarea zecimală $B_{10} = -b_0 * 2^0 + b_1 * 2^{-1} + \dots + b_{n-1} * 2^{-(n-1)}$.
3. Pasul de cuantizare este $q = 2^{-(n-1)}$

Reprezentarea fracționară este cunoscută și sub numele de reprezentarea Q_x , unde x este numărul de biți fracționari. Numărul total de biți ai reprezentării fiind $x+1$. Reprezentarea fracționară uzuală la DSP-urile în virgulă fixă (16 biți) este Q_{15} .

Localizarea punctului nu este neapărat precizată aceasta fiind un instrument de programare. Astfel folosind reprezentarea pe 16 biți gama numerelor poate fi:

[0, 65535] - la reprezentarea întreagă fără semn

[-32768, +32767] - la reprezentarea întreagă cu semn

[-1, +0.999969] - la reprezentarea fracționară

Toate aceste variante sunt utilizate la aplicațiile cu DSP.

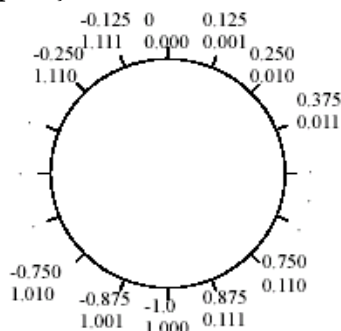


Fig. 6. "Roata numerelor" pentru reprezentarea fracționară pe 4 biți

Observație. La adunarea a două numere binare reprezentarea lor trebuie făcută în aceeași convenție.

Exemple :

Să se reprezinte în format:

a) **Q3, numerele -0.375 și 0.75**

La DSP în virgulă fixă punctul radix (zecimal) nu este disponibil, aceasta cade în sarcina programatorului. La operațiile aritmetice cu date fracționare sau cu semn se folosește aceeași unitate aritmetică și logică (UAL). Reprezentarea fracționară se obține scalând valoarea cu 2^x , unde x reprezintă numărul de biți fracționari. Deci:

$$\begin{aligned} -0.375_{10} &= 1.101_2 * 2^3 = 1101_2 = -3_{10} \\ 0.75_{10} &= 0.110_2 * 2^3 = 0110_2 = 6_{10} \end{aligned}$$

b) **Q15, numărul $X_{10} = -8.9969653E-003$**

Se calculează $X * 2^{15} = -294.8 (\sim -295)$, care în reprezentarea în complement față de 2 este $X_{Q15} = 0FED9h$.

Să se înmulțească folosind reprezentarea fracționară pe 4 biți:

c) **-0.5₁₀ cu 0.75₁₀**

$$\begin{aligned} 1.100 &= -0.5_{10} \\ 0.110 &= 0.75_{10} \end{aligned}$$

```
00000000
11111000
111100
00000
```

$$11.101000_2 = -0.375_{10}$$

↖ Bit de semn suplimentar

Observație. Atenție la extensia de semn.

Rezultatul va avea 6 biți fracționari din 8.

Revenim la reprezentarea pe 4 biți :

$$1|1.101|000_2 = 1.101_2 = -0.375_{10} \text{ care reprezintă un rezultat corect.}$$

d) **-0.5₁₀ cu 0.625₁₀**

$$1.100 * 0.101 = 11.101100_2 = -0.3125_{10}$$

Revenim la reprezentarea pe 4 biți:

$$1|1.101|100_2 = 1.101_2 = -0.375_{10} \text{ care este incorect datorită truncării.}$$

e) – 0.25_{10} cu 0.5_{10} în format Q15.

```

0.100 0000 0000 0000 *
1.110 0000 0000 0000
-----
11.11 1000 0000 0000 0000 0000 0000 0000 format Q30

```

Trecem la format Q15, eliminăm bitul de semn suplimentar și rezultă:

$1.1110\ 0000\ 0000\ 0000_2 = -0.125_{10}$, corect.

Avantaj: Utilizând fracții binare se obține o viteză mai mare în calculele în buclă închisă.

Dezavantaj: Rezultatul poate să nu fie exact. După cum rezultă de mai sus în memorie se obține valoarea $(-4/16)$. Biții cuprinși între 2^{-4} și 2^{-6} au fost trunchiați. Rezultatul corect este $(-3/16)$!

Este de menționat că operația de înmulțire pe 4 biți nu se face la capacitatea reală a C28x care operează pe 32 de biți. În acest caz trunchierea va afecta biții cuprinși între 2^{-32} și 2^{-64} . În cele mai multe cazuri se trunchiază numai zgomotul. Cu toate acestea, în unele aplicații cu reacție (de exemplu filtrele IIR) erorile mici pot adăuga și conduce la un anumit nivel de instabilitate. Este responsabilitatea proiectantului de a recunoaște această potențială sursă de eșec în folosirea fracțiilor binare.

4. Operații cu numere mai mari ca 1

a) Se scalează toți coeficienții din algoritm astfel încât să intre în gama $[-1, 1)$. Ca efect rezultă o atenuare a semnalului la ieșire păstrând răspunsul în frecvență.

b) Se utilizează proprietățile înmulțirii întregi cu 1: $A*B = (A-1)*B + B$

Exemplu. Să se înmulțească eșantionul $X_i = 0.625$ cu coeficientul 1.375 . Se presupune reprezentarea pe 4 biți.

```

1.375*0.625=(0.375+1)*0.625 = 0.375*0.625 + 0.625 =0.859375
0.101 *      0.62510*
0.11         0.37510
-----
00.001111 +      0.23437510 +
0.101000      0.62510
-----
00.110111      0.85937510
0.1102 = 0.7510

```

Revenim la reprezentarea pe 4 biți (Q3) :
Rotunjirea datorată numărului limitat de biți.

c) Utilizarea proprietății înmulțirii întregi cu 2: $A*B = A/2 * B + A/2 * B$

Exemplu. $0.625 * 1.375 = 0.859375$, reprezentați în Q3.

```

1.375/2=0.687510 = 0.10112 care în format Q3 rotunjit este 0.1102 = 0.7510
0.62510 * 0.7510 = 0.4687510
0.4687510 + 0.4687510 = 0.937510 ≈ 0.87510
↑      ↑
0.1012 * 0.1102 = 00.0111102
00.0111102+00.0111102=00.1111002 ≈ 0.1112 (în format Q3).

```

5. Reprezentarea în virgulă mobilă

Nucleul unui procesor în virgulă mobilă e o unitate aritmetică ce suportă operații în virgulă mobilă în conformitate cu standardul IEEE 754/85. Un exemplu tipic pentru această clasă e familia x86 de la Intel începând cu procesoarele 486. Procesoarele în virgulă mobilă sunt foarte eficiente când se operează cu date în virgulă mobilă și permit efectuarea unei game mari de calcule numerice. Aceste procesoare nu sunt așa de eficiente în cazul controlului sarcinilor (manipularea biților, controlul intrărilor și ieșirilor, răspunsul la întreruperi) și în plus sunt destul de scumpe.

Formatul IEEE 754/85 cuprinde numere finite ce pot fi în baza 2 (binare) sau în baza 10 (zecimale). Valoarea numerică a numărului finit va fi dat de formula: $(-1)^s \times f \times b^e$, unde b reprezintă baza numărului. Fiecare număr este descris prin cei 3 parametri: s - semnul (zero sau unu), f – mantisa și e – exponentul. De exemplu, dacă semnul este 1 (un nr negativ), mantisa este 12345, exponentul este -3 și se consideră baza 10, atunci numărul va fi: -12.345.

Toate valorile finite posibile de reprezentare într-un anumit format sunt determinate de: baza de numerație, numărul maxim de digiți din mantisă (ce definesc precizia p) și valoarea maximă a exponentului, $e_{\max}$. Mantisă trebuie să fie un număr întreg în domeniul $[0; b^p-1]$. Exponentul trebuie să fie un întreg a.â. $1-e_{\max} \leq q+p-1 \leq e_{\max}$.

Standardul IEEE 754/85 definește 5 formate de bază: trei formate binare (pot coda numere pe 32, 64 sau 128 biți) și două zecimale (pot coda numere pe 64 sau 128 biți). Formatul binar pe 32 biți pentru precizie simplă în virgulă mobilă descrie numerele în virgulă mobilă astfel:



S= bit de semn

e...e= 8 biți ce reprezintă exponentul

f...f= 23 de biți ce reprezintă mantisa (biți fracționari)

Avantaj: exponentul oferă o gamă dinamică mare de reprezentare a numerelor

Dezavantaj: precizia de reprezentare a numerelor depinde de exponent

Reprezentarea în virgulă mobilă - Obținerea biților:

- Bitul de semn:
Negativ: bitul 31 = 1
Pozitiv : bitul 31 = 0
- Mantisa:

$$M = 1 + m_1 2^{-1} + m_2 2^{-2} + \dots = 1 + \sum_{i=1}^{23} m_i \times 2^{-i}, \text{ unde } 1 \leq M < 2$$

- Exponentul:
O valoare pe 8 biți cu semn, memorată cu offset-ul „+127”

Astfel, calcularea unei valori v se face astfel:

$$\mathbf{z} = (-1)^S \times M \times 2^{E-OFFSET}$$

Example :

Să se scrie în format virgulă mobilă următoarele numere:

1) 0x 3FE0 0000h = 0011 1111 1110 0000 0000 0000 0000 0000 B

$$S = 0$$
$$E = 0111\ 1111 = 127$$
$$M = (1).11000 = 1 + 0.5 + 0.25 = 1.75$$
$$Z = (-1)^0 * 1,75 * 2^{127-127} = 1,75$$

2) $Z = -2.5$

$$S' = 1$$
$$2.5 = 1.25 * 2^1$$
$$1 = E - \text{OFFSET}$$

E = 128

$$M = 1.25 = (1) \cdot 0.25 = 1 + 0.25$$

Binär: 1100 0000 0010 0000 0000 0000 0000 0000 B = 0x C020 0000h

Reprezentarea în virgulă mobilă are și dezavantaje:

Exemplul *:

```
x = 10.0          (0x41200000)
+ y = 0.000000238 (0x347F8CF1)
```

$z = 10.000000238$ Greșit! Nu se poate reprezenta valoarea 10.000000238 cu numere în virgula mobilă în simplă precizie.

0x412000000 = 10.000000000

10.000000238 $\Leftarrow$ nu se poate reprezenta!
0x412000001 = 10.000000950

Astfel, numărul 10.000000238 va fi rotunjit în jos la numărul 10.000000000.

Proprietăți

- Adunarea numerelor reprezentate în virgulă mobilă presupune reprezentarea lor cu același exponent; rezultatul obținut este apoi rotunjit și normalizat dacă este cazul
- Înmulțirea a două numere reprezentate în virgulă mobilă se realizează după regula: exponentul rezultatului se va scrie pe un număr de biți egal cu suma numărului de biți ai exponenților numerelor ce se adună, iar rezultatul obținut este apoi rotunjit și normalizat

6. Formatul „IQ”

Până acum s-a prezentat cazul folosirii numerelor fracționare la care punctul fracționar se afla după MSB. În general acest punct se poate pune oriunde în reprezentarea binară. Astfel se poate obține o rezoluție mai mare.

"I" – reprezintă partea întreagă (integer)

"Q" – reprezintă partea fracționară (quotient)

Avantaj: aceeași precizie pentru toate numerele

Dezavantaj: domeniul dinamic este limitat comparativ cu reprezentarea în virgulă mobilă

$$\begin{array}{c}
 31 \qquad \qquad \qquad 0 \\
 s \text{ } iiii. qqqqqqqqqqqqqqqqqqq \\
 B \text{ } -----> \\
 \qquad \qquad \qquad \text{mantisă pe 23 de biți} \\
 -2^I + 2^{I-1} + \dots + 2^1 + 2^0 \cdot 2^{-1} + 2^{-2} + \dots + 2^{-Q}
 \end{array}$$

Exemplul 1: Formatul I1Q3

3 0

s. qq

Cel mai negativ număr zecimal: -1.0 = 1. 000

Cel mai pozitiv număr zecimal: +0.875 = 0. 111

Cel mai mic număr IQ zecimal negativ: $-1 \cdot 2^{-3}$ = 1. 111

Cel mai mic număr IQ zecimal pozitiv: 2^{-3} = 0. 001

Domeniul: -1.0 ... 0.875

Rezoluția: 2^{-3}

Exemplul 2: Formatul I3Q1

3 0

sii . q

Cel mai negativ număr zecimal: -4.0 = 100. 0

Cel mai pozitiv număr zecimal: +3.5 = 011. 1

Cel mai mic număr IQ zecimal negativ: $-1 \cdot 2^{-1}$ = 111. 1

Cel mai mic număr IQ zecimal pozitiv: 2^{-1} = 000. 1

Domeniul: -4.0 ... +3.5

Rezoluția: 2^{-1}

Exemplul 3: Formatul I1Q31

s. qqqq qqqq qqqq qqqq qqqq qqqq qqqq qqqq

Cel mai negativ număr zecimal: -1.0 = 1.000 0000 0000 0000 0000 0000 0000 0000 B

Cel mai pozitiv număr zecimal: $1 - 1/2^{31}$ = 0.111 1111 1111 1111 1111 1111 1111 1111 B

Cel mai mic număr IQ zecimal negativ: $-1 \cdot 2^{-31}$ = 1.111 1111 1111 1111 1111 1111 1111 1111 B

Cel mai mic număr IQ zecimal pozitiv: 2^{-31} = 0.000 0000 0000 0000 0000 0000 0000 0001 B

Domeniul: [-1.0 +1.0)

Rezoluția: 2^{-31}

Dacă transpunem Exemplul * în formatul IQ, vom avea:

$$\begin{array}{rcl}
 x & = & 10.0 \quad (0x0A000000) \\
 + y & = & 0.000000238 \quad (0x00000004)
 \end{array}$$

$z = 10.000000238 \quad (0x0A000004)$, număr care se poate reprezenta în formatul IQ !

Folosind formatul IQ în locul reprezentării în virgulă mobilă se poate obține o precizie mai mare în reprezentarea datelor.

7. Concluzii

Numerele întregi versus numerele fracționare:

Domeniul:

- numerele întregi au un domeniu maxim determinat de numărul de biți pe care se reprezintă numărul
- numerele fracționare pot aparține doar domeniului $[-1; +1]$

Precizia:

- numerele întregi au o precizie de maxim 1
- numerele fracționare au precizia determinată de numărul de biți

Aritmetica în virgulă fixă versus aritmetica în virgulă mobilă:

Aritmetica în virgulă mobilă este mai flexibilă decât cea în virgulă fixă. Principalul avantaj constă în accesul la o gamă dinamică mai mare a datelor reprezentate.

Exemple de procesoare:

- cu virgulă mobilă: Seria Intel Pentium, Texas Instruments C67xxDSP
- cu virgulă fixă: Motorola HC68x, Infineon C166, Texas Instruments TMS430, TMS320C5000, C2000

Procesoarele în virgulă fixă:

Pot reprezenta numere:

- întregi (aritmetica întregă): pentru control, calcul adrese (nu sunt implicate semnale)
- fracționare: pentru prelucrări cu semnale

Prezintă prețuri scăzute

Programare ușoară: în faza de simulare se pot detecta toate situațiile în care este necesară corecția rezultatului pentru a se evita saturarea.

Procesoarele în virgulă mobilă:

Numerele au mantisă și exponent: $\text{mantisă} * 2^{\text{exponent}}$

Prezintă o tehnologie foarte flexibilă

Au o gamă dinamică largă de reprezentare a numerelor

Programare ușoară: programatorul nu mai trebuie să gestioneze cazurile de umplere (overflow) ale acumulatorului

Dezavantaj: prezintă costuri mari

8. Teme

8.1 Se dau mai jos cei 15 coeficienți ai unui filtru FIR trece sus. Să se convertească coeficienții în format Q15.

($h_1=h_{15}=8.99696E-3$; $h_2=h_{14}=-7.86406E-3$; $h_3=h_{13}=-3.34992E-2$; $h_4=h_{12}=-6.53486E-2$; $h_5=h_{11}=-9.8897E-2$; $h_6=h_{10}=-0.12856$; $h_7=h_9=-0.14897$; $h_8=0.84375$)

8.2 După modelul celor 3 exemple prezentate la formatul IQ, specificați valorile corespunzătoare formatului I8Q24.

8.3 Să se scrie în format virgulă mobilă numărul : **0x BFB0 0000h** pe 32 biți.

8.4 TEME de studiu :

Studiați următoarele exemple pentru adunarea, scăderea și înmulțirea numerelor în virgulă mobilă:

Exemplul 1. Să se efectueze adunarea numerelor 123456.7 și 101.7654

$$\begin{aligned} 123456.7 &= 1.234567 * 10^5 \\ 101.7654 &= 1.017654 * 10^2 = 0.001017654 * 10^5 \end{aligned}$$

În detaliu:

$$\begin{array}{rcl}
 e=5; & z=1.234567 & (123456.7) \\
 + e=2; & z=1.017654 & (101.7654) \\
 \hline
 e=5; & z=1.234567 & \\
 + e=5; & z=0.001017654 & (\text{după shiftare}) \\
 \hline
 e=5; & \text{suma}=1.235584654 & (\text{suma reală este: } 123558.4654)
 \end{array}$$

Dacă se revine la reprezentarea cu 7 digiți rezultatul va fi: $e=5$; $\text{suma}=1.235585$ (adică suma finală va fi: 123558.5), deci rezultatul a fost rotunjit și normalizat. Cei 3 digiți 654 s-au pierdut, aceasta fiind eroarea de rotunjire.

Exemplul 2. Să se efectueze scăderea numerelor 123457.1467 și 123456.659:

$$\begin{array}{rcl}
 123457.1467 & = & 1.234571 * 10^5 \\
 123456.659 & = & 1.234567 * 10^5 = 0.000004 * 10^5
 \end{array}$$

În detaliu:

$$\begin{array}{rcl}
 e=5; & z=1.234571 & (123457.1467) \\
 - e=5; & z=1.234567 & (123456.659) \\
 \hline
 e=5; & d=0.000004 & (\text{diferența reală este: } 0.000004877)
 \end{array}$$

Dacă se revine la reprezentarea cu 7 digiți rezultatul va fi: $e=-1$; $s=4.000000$ (adică rezultatul final va fi: 0.4), deci rezultatul a fost rotunjit și normalizat. Eroarea de rotunjire este în acest caz de aprox. 20%.

Exemplul 3. Să se efectueze înmulțirea numerelor 4734.612 și 541724.2:

$$\begin{array}{rcl}
 e=3; & z=4.734612 & (4734.612) \\
 \times e=5; & z=5.417242 & (541724.2) \\
 \hline
 e=8; & z=1.234567 & \\
 \times e=8; & z=0.001017654 & (\text{după shift-are}) \\
 \hline
 e=8; & \text{prod}=25.648538980104 & (\text{produsul real este: } 2564853898.0104) \\
 e=8; & \text{prod}=25.64854 & (\text{după rotunjire}) \\
 e=9; & \text{prod}=2.564854 & (\text{după normalizare})
 \end{array}$$