Zenon ULMAN, Mirosław PLEBANEK, Marcin OŻAROWSKI
POLITECHNIKA GDAŃSKA, WYDZIAŁ ELEKTROTECHNIKI I AUTOMATYKI, KATEDRA AUTOMATYKI

Arytmetyka zredukowanego systemu binarnego

Dr hab. Inż. Zenon ULMAN

Pracuje w Katedrze Automatyki Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej. Zainteresowania naukowe dotyczą arytmetyki komputerowej i cyfrowego przetwarzania sygnałów.

e-mail: z.ulman@ely.pg.gda.pl

Mgr inż. Mirosław PLEBANEK

Jest doktorantem w katedrze Automatyki, wydziału Elektrotechniki i Automatyki, na Politechnice Gdańskiej. Jego zainteresowania skupiają się na zastosowaniu systemu resztowego w arytmetyce komputerowej, oraz kryptografii.

e-mail: m.plebanek@ely.pg.gda.pl

Mgr inż. Marcin OŻAROWSKI

Doktorant w Katedrze Automatyki Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej. Zainteresowania obejmują technikę mikroprocesorową, arytmetykę komputerową, systemy kryptograficzne i przemysłowe sieci komputerowe.

e-mail: m.ozarowski@ely.pg.gda.pl

Streszczenie

Arytmetyka zredukowanego systemu binarnego umożliwia wykonywanie operacji modulo 2^n+1 w binarnych układach logicznych liczących modulo 2^n. Z tego powodu jest ona chętnie stosowana w algorytmach cyfrowego przetwarzania sygnałów, na przykład do obliczeń transformaty Fouriera modulo liczby Fermata. W literaturze polskiej system ten nie był dotychczas omawiany. Artykuł przedstawia szczegółową definicję zredukowanego systemu binarnego oraz przedstawia zasady wykonywania elementarnych operacji arytmetycznych w układach cyfrowych.

Słowa kluczowe: system liczbowy, arytmetyka binarna, liczby Fermata, arytmetyka modulo.

Diminished-1 arithmetic

Abstract

Diminished-1 arithmetic makes possible performing modulo-2^n+1 operations in binary arithmetic hardware which computes modulo-2^n. For this reason it is willingly used in various digital signal processing applications, for instance in computing modulo-Fermat-number Fourier transforms. In this article the definition of the diminished-1 system is discussed in detail in comparison to the natural binary system. Basic arithmetic diminished-1 operations in binary circuits are also considered.

Keywords: number system, binary arithmetic, Fermat numbers, modulo arithmetic.

1. Wstęp

W cyfrowym przetwarzaniu sygnałów często jest wykorzystywany resztowy system liczbowy (Residue Number System, RNS). Jego szczególną zaletą jest możliwość dekompozycji liczby na niezależne części, na których można wykonywać arytmetyczne dodawanie, odejmowanie i mnożenie, bez konieczności uwzględniania przeniesień. Prowadzi to do bardzo szybkiej realizacji tych operacji w działających jednocześnie, równoległych układach arytmetycznych. Chętnie stosowaną bazą RNS jest zbiór modułów $\{2^n-1, 2^n, 2^n+1\}$, gdyż wówczas operacje arytmetyczne na liczbach wziętych modulo wybrany moduł, można wykonywać w istniejących binarnych układach arytmetycznych, zaś wyniki zapisywać w typowych rejestrach 8, 16, 32, 64-bitowych. Jednak dla modułu 2^n+1 n-bitowy rejestr jest za krótki. Identyczny problem pojawia się podczas obliczania dyskretnej transformaty Fouriera

$$X(n)=\sum_{k=0}^{N-1} x(k)\alpha^{-<nk>} \bmod F_t, \tag{1}$$

gdzie: F_t jest liczbą Fermata w formie $2^{2^t}+1$. Do jednoznacznego przedstawienia wyniku należy tutaj użyć 2^t+1 bitów.

Konieczność użycia dodatkowego bitu komplikuje działania arytmetyczne (dodatkowy bit wymusza kolejne dodawania i przesunięcia podczas mnożenia itp.). W celu ominięcia powyższych trudności L.M. Leibowitz w [2] zaproponował pewną odmianę arytmetyki binarnej (Diminished-1 System) dla transformat modulo F_t. System ten do podobnych celów był wykorzystywany przez kilku innych autorów [1, 3].

W literaturze polskiej system ten przepuszczalnie nie był dotychczas prezentowany i jego nazwa nie ma dotychczas polskiego odpowiednika. Poniżej został on przedstawiony jako zredukowany system binarny (ZB) w odróżnieniu od naturalnego systemu binarnego (NB). Została również przedstawiona arytmetyka ZB. Położono nacisk na wyraźne rozróżnienie między wartością liczby a jej reprezentacją w obu omawianych systemach.

Oznaczenia:

A	– n-bitowa reprezentacja liczby, wektor cyfr,
$\mathcal{A}$	– liczba odpowiadająca reprezentacji A w NB,
$\mathcal{A}_{-1}$	– liczba odpowiadająca reprezentacji A w ZB,
$-\mathcal{A}$	– liczba przeciwna do A,
$\bar{\mathcal{A}}$	– liczba odpowiadająca reprezentacji binarnej powstałej przez zanegowanie wszystkich bitów A,
MSB	– najbardziej znaczący bit,
LSB	– najmniej znaczący bit,
$F:\{A\}\to\{\mathcal{A}\}$	– odwzorowanie zbioru reprezentacji w zbiór liczb,
$F^{-1}:\{\mathcal{A}\}\to\{A\}$	– odwzorowanie zbioru liczb w zbiór reprezentacji, funkcja odwrotna do F,
$\equiv$	– znak kongruencji.

Zależność między A i $\mathcal{A}$ w ZB określa wzór

$$\mathcal{A}_{-1}=F_{\mathrm{ZB}}(A)=F_{\mathrm{NB}}(A)+1=\mathcal{A}+1, \tag{2}$$

gdzie:

$$F_{\mathrm{NB}}(A)=\sum_{i=0}^{n-1} x_i 2^i, \tag{3}$$

gdzie A oznacza tę samą reprezentację w kodzie NB i ZB.

Wynika stąd, że o ile zbiór n-bitowych wektorów cyfr odpowiada w systemie NB zbiorowi liczb od 0 do 2^n-1 to w systemie ZB reprezentuje on zbiór liczb od 1 do 2^n. Oznacza to, że do reprezentacji zera potrzebny jest dodatkowy $n+1$ bit. Przyjmuje się,

że jest on bitem najbardziej znaczącym. Zgodnie z (2) liczba 0 w kodzie ZB powinna odpowiadać -1 w *n*-bitowym systemie uzupełnień (U2), 0 = -1 + 1. W U2 mamy

$$-1 \leftrightarrow \overline{0001}+1 = 1110+0001 = 1111,$$

wobec tego

$$\mathcal{A}_{-1} = 0 = -1+1 \leftrightarrow 1111+0001 = 10000,$$

a zatem MSB = 1 oznacza, że $\mathcal{A}_{-1} = 0$. Jeśli dodatkowy bit $x_n = 0$, pozostałe bity określają wartość rozważanej liczby zgodnie z wyrażeniem

$$\mathcal{A}_{-1} = (1-x_n)\left(\sum_{i=0}^{n-1} x_i 2^i + 1\right). \tag{4}$$

Podejście takie pozwala na pominięcie dodatkowego bitu x_n podczas wykonywania operacji arytmetycznych, co prowadzi do skrócenia czasu ich wykonywania. Relacje pomiędzy wektorami cyfr a liczbami w NB i ZB przedstawiono w tabeli 1.

Tab. 1. Reprezentacje binarne oraz wartości liczb w kodach NB i ZB (na podstawie [2])
Tab. 1. Binary representations and their values in the natural (NB) and diminished-1 (ZB) codes (based on [2])

$\mathcal{A}$	A	$\mathcal{A}_{-1}$
0	00000	1
1	00001	2
2	00010	3
3	00011	4
4	00100	5
5	00101	6
6	00110	7
7	00111	8
8	01000	9 (-8)
9 (-8)	01001	10 (-7)
10 (-7)	01010	11 (-6)
11 (-6)	01011	12 (-5)
12 (-5)	01100	13 (-4)
13 (-4)	01101	14 (-3)
14 (-3)	01110	15 (-2)
15 (-2)	01111	16 (-1)
16 (-1)	10000	0

Przykład 1: Wektorowi cyfr 00110 odpowiada wartość liczbowa 6 w kodzie NB oraz 7 w kodzie ZB

$$6 \xleftarrow{NB} 00110 \xleftarrow{ZB} 7.$$

2. Operacja modulo 2^n+1

Każda operacja modulo określa jednoznacznie zakres reprezentowanych liczb. W przypadku liczb wyrażonych wzorem

$$X = \left|\sum_{i=0}^{m+n-1} x_i \beta^i\right|_{2^n+1}, \tag{5}$$

zakres jest określony przez przedział $[0,\ 2^n+1)$. Symbol β oznacza podstawę wagowego systemu liczbowego. Operację modulo 2^n+1 liczby $(m+n)$-bitowej wykonuje się poprzez odjęcie liczby reprezentowanej przez m najstarszych bitów od liczby reprezentowanej przez n młodszych bitów. Można to wyjaśnić wychodząc z zależności

$$-1 = \left|\beta^S\right|_{\beta^S+1}. \tag{6}$$

Otrzymujemy z (5) i (6)

$$\begin{aligned}|X|_{\beta^n+1} &= \left|\sum_{i=0}^{m+n-1} x_i\beta^i\right|_{\beta^n+1} = \left|\sum_{i=0}^{n-1} x_i\beta^i + \beta^n \sum_{i=0}^{m-1} x_{i+n}\beta^i\right|_{\beta^n+1} \\ &= \left|\sum_{i=0}^{n-1} x_i\beta^i + \left|\beta^n\right|_{\beta^n+1} \sum_{i=0}^{m-1} x_{i+n}\beta^i\right|_{\beta^n+1} \\ &= \left|\sum_{i=0}^{n-1} x_i\beta^i - 1\sum_{i=0}^{m-1} x_{i+n}\beta^i\right|_{\beta^n+1}.\end{aligned} \tag{7}$$

Odejmowanie w (7) można zamienić na dodawanie w U2. Liczbę przeciwną do $\mathcal{A}$ określa się jako odpowiadającą reprezentacji powstałej przez zanegowanie wszystkich bitów i dodanie do wyniku jedności. Wobec tego (7) można przedstawić w postaci:

$$|X|_{\beta^n+1} = \left|\sum_{i=0}^{n-1} x_i\beta^i + \beta^n \sum_{i=0}^{m-1} \bar{x}_{i+n}\beta^i\right|_{\beta^n+1}. \tag{8}$$

Przykład 2:

$$|81|_{17} \equiv 13 = 2^4 \cdot \underset{MSBs}{5} + \underset{LSBs}{1}$$

$$|81|_{17} = |1-5|_{17} = |-4|_{17} \xrightarrow{Tabela\ 1} |13|_{17} = 13$$

Lub według wzoru (8)

$$|81|_{17} = |1-5|_{17} \overset{(Zmiana\ znaku)}{\equiv} |1+12|_{17} = 13$$

Operację mod(2^n+1) wykonuje się wyjątkowo łatwo, co jest jeszcze jedną zaletą ZB.

3. Operacje arytmetyczne w ZB

Arytmetyka ZB wymaga przekształcenia liczb zapisanych jako wektory cyfr w systemie naturalnym do systemu zredukowanego i po wykonaniu obliczeń powrócenia do systemu naturalnego, który jest łatwiejszy do implementacji oraz bardziej popularny. Większość algorytmów i rozwiązań sprzętowych wykorzystuje system naturalny. Przed omówieniem arytmetyki ZB zauważmy, że w systemach wagowych operacje wykonuje się nie na samych liczbach w sensie wielkości abstrakcyjnych, lecz na reprezentujących je wektorach cyfr. Standardowe układy arytmetyczne są przeznaczone do wykonywania operacji na wektorach cyfr reprezentujących liczby w binarnym systemie naturalnym NB. Aby wykorzystać te układy w arytmetyce ZB należy przedstawić liczby z ZB jako wektory cyfr NB. Prowadzi do tego następujące rozumowanie. Jeśli danej reprezentacji A odpowiada w systemie NB liczba $\mathcal{A}$, to zgodnie z (2), w systemie ZB odpowiada jej liczba $\mathcal{A}_{-1} = \mathcal{A} + 1$, co można zapisać:

$$\text{jeśli } A = A_{-1}, \quad \text{to} \quad \mathcal{A}_{-1} = \mathcal{A} + 1.$$

Np.:

$$5 \xleftarrow{NB} 00101 \xleftarrow{ZB} 6 = 5+1$$

Odwracając zagadnienie można stwierdzić, że danej liczbie $\mathcal{A}$ odpowiada w systemie NB reprezentacja A, zaś w systemie ZB odpowiada jej reprezentacja liczby $\mathcal{A} + 1$. Ponieważ $\mathcal{A}$ ma być przedstawiona w NB możemy zapisać, że

$$\text{jeśli } \mathcal{A} = \mathcal{A}_{-1}, \quad \text{to} \quad A_{-1} = F_{\mathrm{NB}}^{-1}(\mathcal{A} - 1).$$

Równoważnie można zapisać

$$A_{-1} = A - 0\ldots001,$$

czyli wektor cyfr odpowiadający w ZB liczbie $\mathcal{A} = \mathcal{A}_{-1}$ otrzymuje się przez odjęcie arytmetyczne wektora cyfr odpowiadającego w kodzie NB liczbie 1 od wektora cyfr odpowiadającego w NB

liczbie $\mathcal{A}$. Prościej, ale formalnie mniej poprawnie można to samo zapisać jako

$$A_{-1} = A - 1.$$

Taki zapis prawej strony w powyższej równości jest powszechnie stosowany przy wykonywaniu działań arytmetycznych.

Np.:

$$00111 \xleftarrow{NB} 7 \xleftarrow{ZB} 00110 = 00111 - 00001$$

4. Dodawanie

Jest to najprostsza operacja w systemie ZB. Dodawanie wektorów cyfr $A_{-1} = A - 1$ oraz $B_{-1} = B - 1$ należy wykonać według reguły

$$\mathcal{A}_{-1} + \mathcal{B}_{-1} \leftrightarrow A_{-1} + B_{-1} = (A-1) + (B-1) = (A + B - 1) - 1. \quad (9)$$

Z powyższej równości wynika, że dla uzyskania poprawnego rezultatu dodawania,

$$S_{-1} \leftrightarrow S_{-1} = S - 1 = (A + B) - 1,$$

do prawej strony (9) należy dodać +1, zgodnie ze wzorem

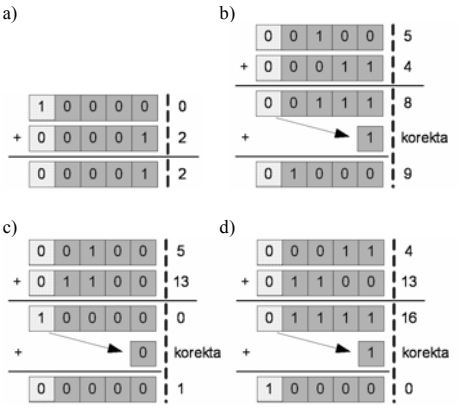
$$(A + B - 1) = [(A-1) + (B-1)] + 1.$$

Jeśli wynik operacji przekracza zakres systemu, co sygnalizuje przeniesienie z bitu MSB, należy na wyniku wykonać operację modulo największa liczba z zakresu liczbowego powiększona o jeden.

Przy dodawaniu dwóch liczb ZB:

- Jeśli bity MSB obu operandów mają wartość 0, zostają one zignorowane i dodawanie odbywa się na pozostałych n bitach. Bit przeniesienia należy zanegować i dodać do powstałej sumy.
- Jeśli bit MSB któregokolwiek składnika jest równy 1, operacja dodawania nie jest wykonywana i jako wynik wpisuje się wartość drugiego składnika.

Przykład 3: Dodawanie dwóch liczb ZB.



5. Wyznaczanie liczby przeciwnej

Dla dowolnej, różnej od zera, n-bitowej liczby $\mathcal{A}_{-1}$ reprezentowanej przez n bitów, element przeciwny jest równy

$$-\mathcal{A}_{-1} \leftrightarrow A_{-1} = 2^n - 1 - (A-1) = \left|2^n + 1 - A - 1\right|_{2^n+1} = \left|-A-1\right|_{2^n+1}. \quad (10)$$

Element przeciwny w ZB otrzymuje się przez negację n bitów jego reprezentacji. Dodatkowy bit (MSB) nie jest negowany. Jeżeli jest on równy 1 liczba zostaje zinterpretowana jako 0 i operacja wyznaczania elementu przeciwnego nie jest wykonywana.

Przykład 4: Znaleźć liczbę przeciwną do $\mathcal{A}_{-1} = -5$ dla $n = 4$

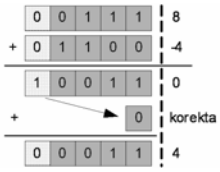
$$-5 = \left|2^4 - 1 - (5-1)\right|_{2^4} = \left|16 - 1 - 5 + 1\right|_{2^4} = \left|11\right|_{2^4}$$

Dla $F_t = 17$ mamy natomiast $-5 \equiv 12 (\mathrm{mod}\, 17)$

6. Odejmowanie

Odejmowanie jakiejkolwiek liczby w systemie ZB jest równe dodaniu liczby przeciwnej (reguła odejmowania jest identyczna jak w przypadku kodu z uzupełnieniem do dwóch). Algorytm polega na dodaniu do odjemnej zanegowanego odjemnika.

Przykład 5:



7. Konwersja liczb

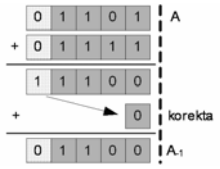
Operacje konwersji opisują poniższe algorytmy.

Konwersja NB → ZB

Aby przekształcić liczbę zapisaną w kodzie NB do systemu ZB, należy do liczby (zapisanej binarnie w NB) dodać binarną reprezentację 2^n-1 w NB.

Po dodaniu liczb wykonywana jest korekcyjna operacja negacji bitu n+1 (dodatkowy bit), który następnie zostaje dodany do wyniku dodawania. Wynik tej operacji jest reprezentacją liczby w systemie ZB.

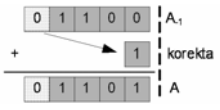
Przykład 6: Konwersja NB → ZB.



Konwersja ZB → NB

Aby liczbę zapisaną w systemie ZB przedstawić w NB, należy zanegować dodatkowy MSB bit tej liczby i dodać go do jej n bitów. Poniższy przykład ilustruje opisany tu mechanizm.

Przykład 7: Konwersja ZB → NB



Tym razem wykonano konwersję otrzymanej w poprzednim przykładzie liczby 13 z systemu ZB do systemu NB. Dodatkowy bit ma wartość 0; po dodaniu jego zanegowanej wartości do liczby ZB otrzymuje się wartość 01101, czyli reprezentację liczby 13 w kodzie NB.

8. Mnożenie liczb przez 2^n

Przed wykonaniem operacji mnożenia dwóch liczb zapisanych w systemie zredukowanym należy sprawdzić dodatkowe bity operandów. Jeżeli jeden z tych bitów ma wartość 1, to wynik operacji zawsze jest równy 0, niezależnie od wartości pozostałych

bitów. W takim przypadku operacja mnożenia nie jest wykonywana. Jeżeli wartość dodatkowych bitów w obu argumentach jest równa 0, to reprezentowane wartości są różne od zera i operacja mnożenia może zostać przeprowadzona.

Operację przeprowadza się za pomocą przesunięcia bitów w kierunku MSB. Należy jednak pamiętać, że liczba zapisana jest w kodzie ZB (czyli pomniejszona o jeden względem reprezentacji pierwotnej) i każde przesunięcie bitów powoduje kolejne pomniejszenie wartości (jedno przesunięcie zmniejsza liczbę o jeden względem reprezentacji pierwotnej). Należy zatem po każdej operacji przesunięcia przeprowadzić korektę wyniku przez dodanie 1.

$$2\cdot\mathcal{A}_{-1} \leftrightarrow 2\cdot A_{-1} = 2\cdot(A-1) = (2A-2) = (2A-1)-1,$$

więc:

$$(2A-1) = 2\cdot(A-1)+1.$$

W przypadku gdy wynik mieści się w zakresie $[0,2^n+1)$, korekta polega na dodaniu wartości 1. Jeżeli wynik wychodzi poza ten zakres, bit dodatkowy (n+1) jest równy 1 i należy go wyzerować. Przykład 8 ilustruje przebieg wykonywania mnożenia przez liczbę 2^3.

Przykład 8:

$$8\times 5 = 2^3\times 5 = 2\times 2\times 2\times 5 = 40 \equiv 6 (\mathrm{mod}\, 17)$$

			$\boldsymbol{A_{-1}}$	$\boldsymbol{\mathcal{A}_{-1}}$
		5	00100	5
2	×	5	01001	10
4	×	5	00010	3
8	×	5	00101	6

Wynik przeprowadzonej symulacji jest zapisany w systemie ZB.

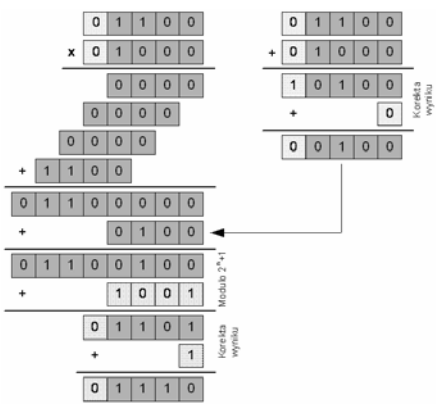
9. Iloczyn dwóch liczb

Operacja mnożenia dwóch liczb A_{-1} i B_{-1} wykonywana jest według wzoru

$$\mathcal{A}_{-1}\cdot\mathcal{B}_{-1} = \mathcal{P}_{-1} \leftrightarrow P_{-1}$$

$$\begin{aligned} |P_{-1}|_{2^n+1} &= |P-1|_{2^n+1} = |A\cdot B-1|_{2^n+1} = \\ &= |(A_{-1}+1)\cdot(B_{-1}+1)-1|_{2^n+1} = \\ &= |A_{-1}\cdot B_{-1} + A_{-1} + B_{-1}|_{2^n+1} \end{aligned} \qquad (11)$$

Na rysunku 1 przedstawiony jest algorytm mnożenia w ZB. W pierwszej kolejności należy sprawdzić dodatkowy bit na pozycji n+1. Jeżeli jest on równy 1 to wynik mnożenia wynosi 0. W przypadku, gdy jeden z argumentów jest równy jedności, wynikiem operacji będzie drugi z argumentów.



Rys. 1. Algorytm mnożenia w ZB (Dane: $\mathcal{A}_{-1}$ = 13, $\mathcal{B}_{-1}$ = 9; Operacja: $\mathcal{A}_{-1}\times\mathcal{B}_{-1}$ = 13 × 9 =117 ≡ 15(mod 17))

Fig. 1. Multiplication algorithm in ZB (Data: $\mathcal{A}_{-1}$ = 13, $\mathcal{B}_{-1}$ = 9; Operation: $\mathcal{A}_{-1}\times\mathcal{B}_{-1}$ = 13 × 9 = 117 ≡ 15(mod 17))

Po dokonaniu analizy danych, algorytm wykonuje obliczenia zgodnie z podanym wzorem (11). Należy zauważyć, iż korekta wykonywana jest tylko dwa razy, przy dodawaniu mnożnej do mnożnika, w celu uzyskania reprezentacji w ZB, i po wykonaniu operacji modulo.

Możliwy jest również inny sposób mnożenia. Metoda ta wymaga, żeby mnożna i mnożnik były różne od zera i było możliwe przejście na reprezentację NB.

W pierwszej kolejności reprezentacje argumentów konwertowane są z ZB do NB. Następnie wykonywanie jest mnożenie binarne, operacja modulo 2^n+1 i korekta wyniku. Wynik operacji przestawiony jest w ZR.

10. Porównanie ZB z NB

Reprezentacja liczb w NB i ZB niewiele się różni. Podstawowa różnica występuje w zapisie liczby 0 i jest ona kluczowa w reprezentacji ZB. Operacje matematyczne w obu systemach przebiegają identycznie.

Gdzie zatem powinno się zastosować reprezentacje ZB? Załóżmy, że wykonujemy obliczenia w systemie RNS ze standardową bazą $B = \{2^4-1, 2^4, 2^4+1\}$. Składniki bazy wyznaczają zakres liczbowy RNS. Dla dwóch pierwszych modułów, reszty można zapisać na 4 bitach, natomiast dla trzeciego modułu 4 bity nie umożliwiają zapisania jednoznacznie wszystkich liczb z danego zakresu, gdyż maksymalną liczbą jest 16. Tak więc operacje matematyczne dla dwóch pierwszych modułów wykonywane są na 4 bitach a dla trzeciego na 5 bitach. To powoduje zwiększenie złożoności układów liczących, gdyż obliczenia dla trzeciego modułu wymagają większej ilości bitów. Jednak można zauważyć, że 5 bitów wykorzystanych jest tylko do przedstawienia wartości 16 (czyli 0 w ZB).

W reprezentacji ZB piąty bit wykorzystany jest tylko do zapisu liczby 0, która ma szczególną reprezentację w tym systemie. Przy takim zapisie, ilość bitów biorących udział w obliczeniach dla każdego z modułów jest taka sama, co w konsekwencji zmniejszy złożoność całego układu. Ceną za posługiwanie się systemem ZB jest konieczność konwersji liczb z NB do ZB i odwrotnie oraz sprawdzanie dodatkowego bitu przed rozpoczęciem wykonywania operacji matematycznych. Gdy jedna z liczb podczas dodawania jest równa 0, to operacja ta nie jest wykonywana, a wynikiem jest druga z liczb. Natomiast przy mnożeniu wynik będzie równy 0. Ze względu na fakt, że bit ten będzie odczytany wyłącznie na początku algorytmu nie jest wymagane pamiętanie go w dalszych krokach obliczeń. W efekcie możliwe jest uproszczenie całości układu i tym samym zwiększenie wydajności obliczeniowej. Zastosowanie systemu ZB pozwala wykorzystać w pełni potencjał systemu o bazie $B = \{2^4-1, 2^4, 2^4+1\}$ bez zbytniego komplikowania układu.

Konwersja liczb z NB, i po wykonaniu obliczeń, do NB jest wykonywana tylko raz i jest bardzo prosta. W konfrontacji z wielokrotnymi operacjami podczas mnożenia, konwersje nie stanowią argumentu przeciwko użyciu ZB w cyfrowym przetwarzaniu sygnałów.

11. Literatura

[1] J. H. McClellan, „Hardware realization of a Fermat number transform", IEEE Trans. Acoust. Speech, Signal Processing, vol. ASSP-24, pp. 216-225, June 1976.

[2] L. M. Leibowitz, „A simplified binary arithmetic for the Fermat number transform", IEEE Trans. Acoust. Speech, Signal Processing, vol. ASSP-24, pp. 356-359, October 1976.

[3] S. Sunder, „Area-efficient diminished-1 multiplier for Fermat number-theoretic transform", IEE PROCEEDINGS-G, vol. 140, No 3, June 1993.

Artykuł recenzowany