2. Arytmetyka procesorów 16-bitowych stałoprzecinkowych

Liczby stałoprzecinkowe

Podstawowym zastosowaniem procesora sygnałowego jest przetwarzanie, w czasie rzeczywistym, ciągu próbek wejściowych w ciąg próbek wyjściowych. Próbki wejściowe i wyjściowe są najczęściej 16-bitowe tak jak w procesorze TMS320C5515. Procesor TMX320C5515 jest procesorem stałoprzecinkowym 16-bitowym, co oznacza, że jest zaprojektowany do przetwarzania 16-bitowych operandów. W trakcie przetwarzania pośrednie wyniki mogą być liczbami innych formatów, ale wynik (ciąg próbek wyjściowych) musi być 16-bitowy. Kompilator języka C pozwala na posługiwanie się liczbami i działaniami zmiennoprzecinkowymi, jednak te działania złożone z wielu 16-bitowych działań stałoprzecinkowych wymagają wielu taktów zegara i szybko okazuje się, że procesor nie nadąża z ich wykonywaniem w przerwach między kolejnymi próbkami. W takiej sytuacji można zmniejszyć częstotliwość próbkowania, ale lepiej w przetwarzaniu sygnału ograniczyć się do działań stałoprzecinkowych. Arytmetykę zmiennoprzecinkową można wykorzystać na przykład do projektowania badanego układu – przetwarzanie sygnału jest zatrzymywane, procesor oblicza nowe współczynniki filtrów po czym przetwarzanie sygnału jest wznawiane z nowymi współczynnikami.

Działania jednostek obliczeniowych procesora TMX320C5515 są wykonywane na 16-bitowych liczbach typu całkowitego, ze znakiem lub bez. Liczbom całkowitym można przyporządkować różne formaty stałoprzecinkowe, w zależności od przyjętej pozycji przecinka dziesiętnego. Format stałoprzecinkowy zapisuje się w postaci Q*a.b*, gdzie *a* oznacza liczbę cyfr przed przecinkiem, *b* oznacza liczbę cyfr po przecinku. Liczby całkowite 16-bitowe ze znakiem i bez

znaku zapisywane są w formacie Q16.0. Wagi bitów w formacie Q16.0 podane są w tabeli 1.

Tabela 1. Wagi bitów liczby w formacie Q16.0, a) liczba bez znaku, b) liczba ze znakiem

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

a)

-2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

b)

Wartość liczby bez znaku zapisanej w formacie Q16.0 oblicza się wg wzoru

$$x_{160} = b_{15}2^{15} + b_{14}2^{14} + \ldots + b_0 2^0, \tag{2}$$

a wartość liczby ze znakiem zapisanej w formacie 16.0 oblicza się wg wzoru

$$x_{160} = -b_{15}2^{15} + b_{14}2^{14} + \ldots + b_0 2^0. \tag{3}$$

W formacie 16.0 można zapisać liczby całkowite ze znakiem z zakresu od –32768 do 32767.

Oprócz formatu 16.0 służącego do zapisu liczb całkowitych istnieje wiele formatów stałoprzecinkowych pozwalających zapisywać liczby wymierne, jednak praktyczne znaczenie ma głównie format Q1.15. Wagi bitów w formacie Q1.15 podane są w tabeli 2.

Tabela 2. Wagi bitów liczby w formacie 1.15, a) liczba bez znaku, b) liczba ze znakiem

2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

a)

-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

b)

Wartość liczby bez znaku zapisanej w formacie Q1.15 oblicza się wg wzoru

$$x_{Q1.15} = b_{15}2^0 + b_{14}2^{-1} + \ldots + b_0 2^{-15}, \tag{4}$$

a wartość liczby ze znakiem zapisanej w formacie Q1.15 oblicza się wg wzoru

$$x_{Q1.15} = -b_{15}2^0 + b_{14}2^{-1} + \ldots + b_0 2^{-15}. \tag{5}$$

W formacie Q1.15 można zapisać liczby wymierne ze znakiem z zakresu od –1.0 do 0.99997 z rozdzielczością 0.0000305.

Dodawanie i odejmowanie w formacie stałoprzecinkowym jest proste, pod warunkiem, że oba argumenty są w tym samym formacie. Jeśli oba argumenty są w tym samym formacie, wynik działania zachowuje format argumentów, a działanie jest wykonywane identycznie jak na liczbach całkowitych. Położenie przecinka dziesiętnego w żaden sposób nie wpływa na przebieg działania. Jedynym problemem przy dodawaniu i odejmowaniu jest potencjalne przepełnienie rejestru wyjściowego, które jest wykrywane w trakcie działania i może być uwzględnione w kolejnej instrukcji programu.

Stałoprzecinkowe mnożenie nie wymaga zgodności formatu czynników, ale zwykle przyjmuje się, że czynniki mają zgodny format. Mnożenie dwóch 16-bitowych liczb całkowitych w formacie Q16.0 daje 32-bitowy wynik w formacie Q32.0, którego nie można zamienić na format czynników Q16.0 bez utraty najbardziej znaczących bitów.

Mnożenie dwóch liczb w formacie Q1.15 daje 32-bitowy wynik w formacie Q2.30, którego konwersję do formatu Q1.15 ułatwia fakt, że liczby ze znakiem

w formacie 1.15 mają zakres od –1.0 do 0.99997 i wynik mnożenia należy do tego zakresu – za wyjątkiem mnożenia $(-1)\cdot(-1) = 1$. Wynik w formacie Q2.30 można zmienić na format Q1.15 poprzez przesunięcie go o jeden bit w prawo (dokonując tym samym konwersji wyniku z formatu Q2.30 na Q1.31), następnie pobierając 16 najbardziej znaczących bitów jako wynik mnożenia w formacie Q1.15.

W działaniach stałoprzecinkowych może wystąpić przepełnienie, które w pewnych sytuacjach można wykorzystać, ale które częściej jest problemem z którym trzeba sobie jakoś radzić. W zależności od potrzeb w przetwarzaniu sygnałów można stosować dwie arytmetyki modularną i nasyceniową.

Arytmetyka modularna

W arytmetyce modularnej wynik wykraczający poza zakres liczb danego formatu „zawija się" z powrotem do tego zakresu, tak jakby liczby tworzyły pierścień. Przykładem może być dodawanie liczb w formacie Q1.15 ze znakiem oznaczone symbolem $\oplus_2$ dane wzorem

$$a \oplus_2 b = \begin{cases} a + b, & \text{gdy } a + b < 1, \\ a + b - 2, & \text{gdy } a + b \geq 1. \end{cases}$$

Działania w arytmetyce modularnej są łatwe w realizacji przez procesor, gdyż bity wychodzące poza zakres są po prostu pomijane. Arytmetyka modularna w cyfrowym przetwarzaniu sygnałów stosowana jest do generacji sygnałów okresowych.

Arytmetyka nasyceniowa

Przy przetwarzaniu sygnałów arytmetyka modularna może być przyczyną poważnych błędów. Na przykład, gdy suma dwóch próbek sygnału będzie równa 1,1 to w arytmetyce modularnej wynik zostanie zawinięty do 1,1 – 2 = – 0.9 i błąd wartości próbki będzie równy – 2, co zmieni charakter sygnału. Najmniejszy błąd wartości próbki wykraczającej poza zakres uzyska się przyjmując za jej wartość najbliższą granicę zakresu. Jeśli przez *x* oznaczyć wartość próbki, która może

wykroczyć poza zakres liczb formatu Q1.15, to w arytmetyce nasyceniowej jej wartość *y* zostanie sprowadzona do zakresu liczb Q1.15według wzoru

$$y = \begin{cases} 1-2^{-15}, \text{ gdy } x \geq 1-2^{-15} \\ x, \quad \text{gdy } -1 \leq x < 1, \\ -1, \quad \text{gdy } x < -1. \end{cases} \tag{4}$$

Programy cyfrowego przetwarzania sygnału projektuje się tak, aby nie występowało w nich niepożądane przepełnienie, ale nie zawsze udaje się całkowicie go uniknąć, wtedy „mniejszym złem" jest nasycenie niż zawinięcie wartości próbki. Przepełnienia można uniknąć zmniejszając poziom sygnału, ale niski pozom sygnału zmniejsza stosunek sygnału do szumu.

Normalizacja

Przy posługiwaniu się liczbami w formacie Q1.15 używane wartości należy unormować do zakresu [–1, 1), dotyczy to próbek sygnału, współczynników filtrów, częstotliwości i innych parametrów. Programy do projektowania filtrów w tym MATLAB dostarczają wyniki w formacie Q1.15, w algorytmach generacji sygnałów występują unormowane częstotliwość i faza, funkcje biblioteki TMS320C55xx DSPLIB mają unormowane argumenty i wyniki.