PROCESORY SYGNAŁOWE DSP (1)

Autor, Piotr Wołowik – doktorant w Instytucie Elektroniki i Telekomunikacji Politechniki Poznańskiej, w cyklu artykułów wprowadzi Czytelników w dziedzinę procesorów sygnałowych DSP, których znaczenie i obszary zastosowań stale wzrastają.

Wprowadzenie

Aktualne kierunki rozwoju rynku układów scalonych wynikają w głównej mierze z potrzeby dostosowywania go do wymogów i zapotrzebowań odbiorców. W ostatnich latach główne tendencje wzrostowe dotyczyły rynku wszelkiego rodzaju procesorów komputerowych, co wynikało z faktu zwiększonego tempa powszechnej informatyzacji, a także rozwoju technik internetowych. O ile postęp w tej dziedzinie będzie trwał nieustannie, to jednak, jeśli chodzi o produkcję układów scalonych, zostanie on niebawem przyćmiony przez rozwój nowej klasy procesorów. Mowa tutaj o tak zwanych procesorach sygnałowych DSP (*Digital Signal Processors*) dostosowanych zwłaszcza do wymogów, jakie niesie ze sobą powszechność i rozwój multimedialnej telefonii komórkowej (generacje: 2,5G; 3G – UMTS, CDMA2000; 4G).

Obecnie największymi producentami procesorów DSP są firmy: Texas Instruments, Motorola, Analog Devices oraz Lucent Technologies i – jak zapowiadają analitycy z tych firm – w najbliższych latach jest spodziewane szybko rosnące zapotrzebowanie na te procesory. Procesory DSP, dedykowane do wszelkiego typu obliczeń wymaganych w aplikacjach i algorytmach cyfrowego przetwarzania sygnałów, w coraz większym stopniu wypierają dotychczasowe analogowe odpowiedniki.

Bardzo duża szybkość obliczeniowa oferowana przez te procesory, umożliwia implementację oprogramowania i algorytmów działających w czasie rzeczywistym (*Real Time Processing*). Dzięki temu układy te już współcześnie znajdują zastosowania w dziedzinach, w których ta właściwość jest szczególnie pożądana. Dziedziny te, to oprócz wspomnianej już telekomunikacji, także technika nawigacyjna i radarowa, przetwarzanie i kompresja sygnału wizyjnego (standardy MPEG), przetwarzanie obrazów i informacji medycznych uzyskiwanych przy pomocy specjalistycznej aparatury (np. techniki magnetycznego rezonansu jądrowego – MRI, tomografii pozytronowej – PET) itp. zastosowania.

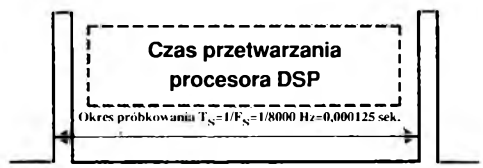
Podsumowując, DSP to specjalistyczna architektura procesora i zbiór instrukcji pomyślanych w taki sposób, aby szybkość działania była bardzo duża (a pobór mocy mały w przypadku urządzeń przenośnych). Konieczne jest dostosowanie do wymogów pracy w trybie czasu rzeczywistego. Dzięki precyzji i opcji reprogramowalności umożliwiają stabilny i szeroki zakres zastosowań, gdzie ograniczeniem wydaje się być tylko ludzka pomysłowość ich zastosowania.

Przykład zastosowania

Przypatrzmy się typowej konfiguracji zastosowania procesora DSP w układach przetwarzających sygnał mowy i pracujących w trybie czasu rzeczywistego, a mających swoje praktyczne odzwierciedlenie w zastosowaniach telekomunikacyjnych.

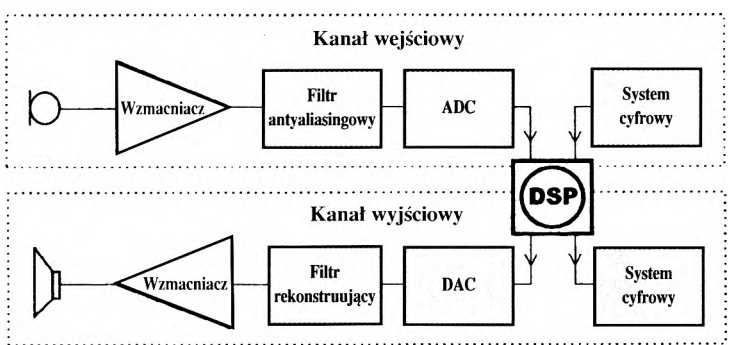
Sygnał w rozważanym na rys. 1 układzie jest pobierany przez mikrofon i następnie wzmacniany do odpowiedniego poziomu, aby w sposób optymalny wysterować przetwornik a/c (ADC). Przed operacją próbkowania sygnał przechodzi przez filtr antyaliasingowy, którego celem jest ograniczenie zakresu częstotliwości składowych w jego widmie, tj. ich eliminacja powyżej górnego progu użytecznych składowych widmowych. Widmo sygnału mowy ma największą zawartość informacyjną w paśmie 300÷3200 Hz, reszta może zostać odfiltrowana – co praktycznie nie wpłynie na obniżenie jakości sygnału mowy. Częstotliwość próbkowania w tym układzie wynosi 8 kHz. Jest to częstotliwość umożliwiająca późniejszą rekonstrukcje sygnału, bowiem zgodnie z kryterium Nyquista powinna być ona co najmniej dwa razy większa od maksymalnej częstotliwości zawartej w próbkowanym sygnale.

Sygnał spróbkowany podlega następnie kwantyzacji 8-bitowej (256 poziomów), dzięki czemu na wyjściu układu ADC pojawia się strumień bitowy 64 kbit/s (modulacja PCM). Strumień tych 8-bitowych próbek danych wpływa do procesora DSP z częstotliwością próbkowania 8 kHz, tj. jedna próbka co 1/8000 Hz=0,000125 s. Jest to czas jakim dysponuje procesor DSP, aby przetworzyć pojedynczą próbkę – ponieważ po tym czasie pojawi się następna, a przetwarzanie ma odbywać się w czasie rzeczywistym, więc nie może być żadnych opóźnień (rys. 2). Czas, jakim dysponuje procesor DSP między kolejnymi pobraniami próbek jest bardzo długi względem



Rys. 2. Czas pomiędzy kolejnymi pobraniami próbek, którym dysponuje procesor DSP dokonujący przetwarzania w trybie czasu rzeczywistego

szybkości przetwarzania danych przez procesor. W tym czasie możliwe są implementacje na DSP bardzo skomplikowanych i złożonych obliczeniowo algorytmów kompresji mowy (np. LPC-10), filtracji adaptacyjnej (np. LMS, RLS) albo w przypadku wysyłania sygnału w kanał telekomunikacyjny – wyszukanych algorytmów modulacji cyfrowej. Możliwe jest także – w tym samym czasie – równoległe przetwarzanie przez procesor innych danych, np. wchodzących do niego ze zwielokrotnionych kanałów lub przechowywanych w pamięci i przetwarzanych w trybie *off-line*. Granicę w tym przypadku wyznacza tylko dostępna częstotliwość pracy procesora z uwzględnieniem podziału jego mo-

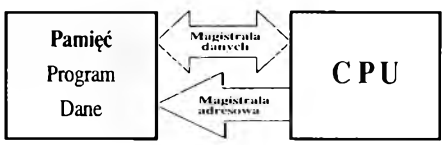


Rys. 1. Typowy układ przetwarzający sygnały akustyczne, gdzie szczególną rolę pełni procesor DSP umożliwiający programową implementację bardzo skomplikowanych i złożonych obliczeniowo algorytmów

cy obliczeniowej na poszczególne procesy. Wracając do omawianego układu, po zakończeniu cyklu pracy procesora, stosownie zmodyfikowane próbki danych (np. odszumione adaptacyjnie) dostępne są na jego wyjściu. Wpływają one do kanału wyjściowego, gdzie zostają następnie odtworzone przez filtr rekonstruujący, stosownie wzmocnione i odsłuchane w głośniku.
Przedstawiony powyżej przykład zostanie wykorzystany w trzeciej części artykułu w celu zobrazowania praktycznego sposobu napisania kodu na procesor sygnałowy realizujący algorytm filtracji adaptacyjnej LMS.

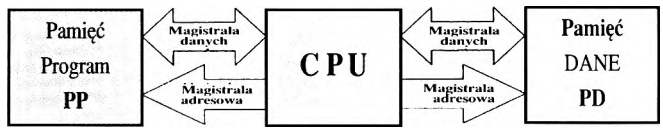
Koncepcja nowej architektury

Pierwsze procesory ogólnego zastosowania jakie pojawiły się na rynku, miały tak zwaną architekturę von Neumanna. Charakteryzowała się ona posiadaniem centralnej jednostki obliczeniowej CPU (*Central Processing Unit*) oraz pamięci, w której był zarówno przechowywany sam program jak i dane, na których on operował (rys. 3). Architektura ta była optymalna jeżeli chodzi o zastosowanie procesorów do prostych operacji manipulacji danymi, takich jak ich przechowywanie, przesunięcia oraz porównywanie. Operacje te najczęściej wykorzystywane były w takich programach jak bazy danych czy edytory tekstu. Dominują w nich przede wszystkim czynności wykonywane

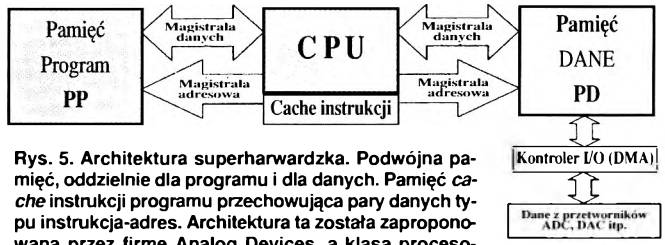


Rys. 3. Architektura von Neumanna. Wspólna pamięć programu i danych

przez CPU polegające na przemieszczaniu danych (zapis na dysk lub drukarkę), ich formatowaniu, selekcji oraz sortowaniu. W programowy sposób większość tych operacji zbudowana była na bazie instrukcji porównujących stosowne dane (operacja IF A>=B) oraz na ich przemieszczaniu (operacja A ->B).
W architekturze von Neumanna instrukcje programu i danych dzielą ten sam obszar pamięci. Z tego powodu korzystają z tych samych magistral danych i adresowych. Działanie takiego procesora jest wolne, ponieważ jednostka obliczeniowa musi pobierać z pamięci operandy instrukcji, jak i samą instrukcję, zanim zacznie ją właściwie wykonywać.
Aby wyeliminować tę niedogodność, a przede wszystkim dzięki temu dostosować procesor do obliczeń matematycznych (szczególnie optymalizacji operacji dodawania i mnożenia), zaproponowano nowatorskie podejście w koncepcji budowy całkowicie nowej architektury. Narodziła się tzw. architektura harwardzka. Polegała ona na rozdzieleniu pamięci danych od pamięci programu, dzięki czemu możliwy stał się jednoczesny, równoległy dostęp do ich stosownych obszarów (rys.4). Takie podejście umożliwiało symultaniczne pobieranie przez jednostkę obliczeniową instrukcji z pamięci programu i stosownych operandów z pamięci danych, a następnie dokonywanie na nich właściwych operacji arytmetycznych. Procesor nie musiał przełączać kontekstu adresując na przemian pamięć danych i pamięć programu – dzięki czemu w znaczny sposób zwiększono szybkość jego działania w porównaniu do rozwiązania zastosowanego w architekturze von Neumanna.
Kierunek rozwoju wyznaczony przez koncepcję architektury harwardzkiej zaowocował nowymi rozwiązaniami opartymi na tej idei w znacznym stopniu zwiększającymi dodatkowo jej wydajność obliczeniową.
Firma Analog Devices wzbogaciła tę architekturę o wewnętrzną pamięć cache instrukcji programu, dzięki czemu raz ściągnięty do niej kod z pamięci programu (PP), umożliwiał nowe, dodatkowe, specyficzne zastosowanie – np. wykorzystanie jej jako dodatkowej pamięci przechowującej drugorzędne dane. Firma Motorola zaproponowała w swoich konstrukcjach dwie pamięci danych X i Y oraz pojedynczą pamięć programu P (rys. 5).
Nowoczesne rozwiązania architektury DSP mają również w układy DMA (*Direct Memory Access*) umożliwiające dostęp do pamięci danych z pominięciem rejestrów procesora – a tym samym znacznie go odciążające obliczeniowo, przenosząc tym samym zysk mocy do wykorzystania na inne zadania.



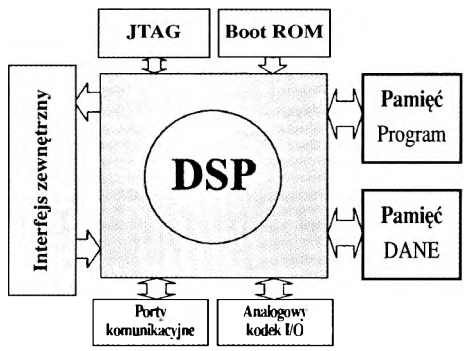
Rys. 4. Architektura harwardzka. Podwójna pamięć oddzielna dla programu i dla danych.



Rys. 5. Architektura superharwardzka. Podwójna pamięć, oddzielnie dla programu i dla danych. Pamięć *cache* instrukcji programu przechowująca pary danych typu instrukcja-adres. Architektura ta została zaproponowana przez firmę Analog Devices, a klasa procesorów zbudowana na tej koncepcji nosi zastrzeżoną nazwę SHARC™ (*Super Harvard ARChitecture*).

Konfiguracja DSP

Oferowane na rynku procesory DSP są wyposażone w wiele elementów dodatkowych, dzięki czemu możliwe są ich różnorodne i szerokie zastosowania (rys.6). Mowa tutaj o różnego typu wyprowadzeniach umożliwiających zestawianie układów DSP z wieloma innymi układami takimi jak: pamięci EPROM, złącza JTAG, magistrale zewnętrzne i podobne układy.
Pełna gama oferowanych przez DSP możliwości dostępna jest w celach edukacyjno-badawczych (umożliwiających konstruowanie prototypów) w sprzedaży w postaci gotowych tzw. starter-kitów.



Rys. 6. Schemat blokowy procesora sygnałowego DSP oraz elementów z nim współpracujących

Cała architektura DSP, mimo iż oferuje bardzo duże możliwości, jest w przypadku niektórych zastosowań całkowicie zbędna. Dodatkowe bogactwo oferowanych przez DSP możliwości jest w konstrukcjach specjalistycznych urządzeń zazwyczaj nie w pełni wykorzystane, a ich obecność wpływa na cenę produktu końcowego. W takim przypadku stosuje się "okrojone" wersje procesorów DSP umożliwiających dokonywanie tylko niezbędnych operacji.
Korzysta się wtedy z układów ASIC (*Application Specific Integrated Circuits*), których użycie znacznie obniża koszty produkcji urządzeń końcowych. Oczywiście ze względu na specyfikę układów ASIC, specjalnych ich projektów oraz linii produkcyjnych – aby całe przedsięwzięcie ich zastosowania było ekonomicznie opłacalne – urządzenia końcowe z nich korzystające muszą być produkowane na dużą skalę przemysłową.

***

W następnej części artykułu będą omówione podstawowe mechanizmy dostępne w procesorach DSP z uwzględnieniem ich zalet i użyteczności w algorytmach cyfrowego przetwarzania sygnałów. ■

Piotr Wołowik

PROCESORY SYGNAŁOWE DSP (2)

Kontynuując rozważania z poprzedniej części artykułu autor omawia właściwości i zalety procesorów sygnałowych DSP pod względem ich użyteczności w specjalistycznych operacjach matematycznych, z jakimi mamy do czynienia w przypadku cyfrowego przetwarzania sygnałów.

Superskalarność

Procesory DSP są pod względem koncepcji działania podobne do procesorów RISC (*Reduced Instruction Set Computers*). Procesory RISC są tak zaprojektowane, aby pracowały w sposób najbardziej efektywny dzięki małemu zbiorowi instrukcji, na jakich operują. Procesory DSP mają duży zestaw dostępnych instrukcji, a podobieństwo polega na efektywności, z jaką one są w nich wykonywane. Instrukcje te zostały pomyślane tak, aby optymalnie odwoływać się do pamięci, wykonywać skomplikowane operacje logiczne i arytmetyczne, których wyniki byłyby przechowywane w odpowiednim zbiorze rejestrów i dzięki temu cały czas dostępne dla jednostki obliczeniowej. Takie rozwiązanie umożliwia eliminację ciągłych odwołań do pamięci w celu dostępu do wyszczególnionych danych – cały czas znajdują się one w jej specjalistycznie pomyślanym zbiorze rejestrów.

Podobnie jak w RISC, w procesorach DSP korzysta się z instrukcji o stałej długości (zazwyczaj jednego słowa). W takiej koncepcji możliwe jest przetwarzanie potokowe, a także minimalizacja odwołań do pamięci w celu poboru instrukcji (każda instrukcja może być pobrana tylko w jednym cyklu).

Superskalarność procesora oznacza, że dzięki specjalnie zaprojektowanemu zbiorowi instrukcji jakimi on operuje, może wiele z nich wykonywać w jednym cyklu zegara równocześnie w sposób równoległy (np. osiem w jednym cyklu zegara). Oznacza to, że zegar procesora może być taktowany z częstotliwością 200 MHz, a sam procesor dysponować szybkością obliczeń 1600 MIPS (*Millions of Instructions Per Second*).

Przykładem takiej instrukcji jest instrukcja *mac* (omówiona w dalszej części artykułu), wykonująca równolegle operacje mnożenia i dodawania, a także dodatkowo operacje indeksowania i pobierania danych.

Operacje cyfrowego przetwarzania sygnałów

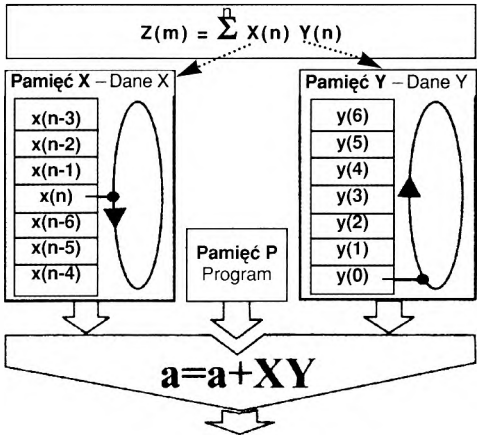
Architektura harwardzka idealnie nadawała się do zastosowań związanych z cyfrowym przetwarzaniem sygnałów. W takim przypadku operacje z jakimi najczęściej mamy do czynienia, to różnego rodzaju transformacje (np. Fouriera), liczenie splotu, korelacji, a także filtracji typu FIR oraz IIR. Wszystkie te obliczenia w postaci zapisu matematycznego mają charakterystyczną postać typu:

$$\sum_{n=1}^{N} x_n y_n = x_1 y_1 + x_2 y_2 + x_3 y_3 + ... + x_N y_N$$

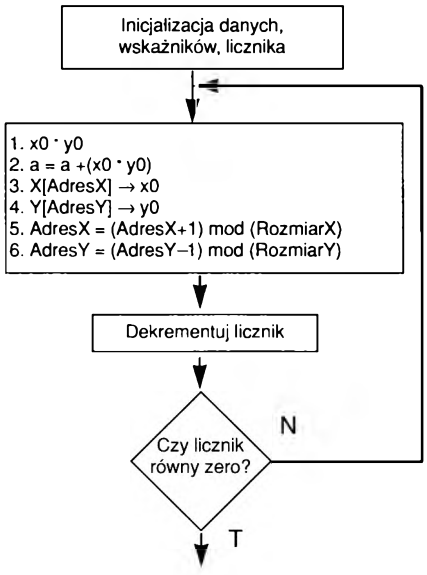
tzn. składają się z mnożeń połączonych z akumulacją, które w procesorach sygnałowych DSP dokonywane są w jednym cyklu jego pracy (operacja MAC omówiona dalej).

Bufory cyrkularne

Operacje MAC (*Multiply and Accumulate*) w procesorach DSP i związane z nimi instrukcje są najistotniejszymi operacjami wpływającymi na szybkość przetwarzania w przypadku cyfrowej obróbki danych. Zasada ich działania opiera się na sprzętowych rozwiązaniach gwarantujących optymalizację wykonywanych przez nie zadań (tzn. mnożenia i jednoczesnej akumulacji). Rozwiązaniami tymi są specjalnie zaprojektowane układy adresujące pamięć DAGs (*Data Address Generators*) oraz odpowiednie towarzyszące im zbiory rejestrów indeksujących. Wykorzystuje się w nich tryb pracy polegający na organizowaniu pamięci w bufory adresowane w sposób cyrkularny (cykliczny, kołowy). Zasada takiego adresowania polega na tym, że najmłodsza pobrana próbka wchodzi na miejsce najstarszej, a wartości wskaźników indeksujących próbki są odpowiednio modyfikowane tak, aby wskazywały odpowiednio ich sekwencyjną kolejność. Na rys. 7 najstarszą wartością w buforze cyrkularnym jest wartość x(n-6). W jej miejsce zapisywana jest aktualna wartość x(n), a dotychczasowa wartość x(n) staje się wartością x(n-1), wartość x(n-1) wartością x(n-2) itd. Wymagana w obliczeniach aktualna wartość próbki x(n), dzięki odpowiedniemu wskazywaniu jej przez stosowne rejestry indeksujące, przemieszcza się w sposób cykliczny po całym adresowanym buforze. Zaletą tego jest fakt, że próbki nie muszą być fizycznie przesuwane tak jak w buforze przesuwnym – "przesunięcie" dokonywane jest tylko poprzez modyfikację i zarządzanie stosownymi rejestrami wskazującymi ich odpowiednią kolejność.



Rys. 7. Zasada działania buforów cyrkularnych oraz bazującej na tej koncepcji operacji MAC



Rys. 8. Algorytm demonstrujący zasadę działania instrukcji *mac* w procesorze firmy Motorola DSP 560xx. Zademonstrowany jest również sprzętowy licznik programu kontrolujący ilość wykonywanych pętli.

Dodatkowo układy DAG mają możliwość

adresowania polegającą na bitowym odwracaniu adresów (*Bit Reverse Addressing*). Ta właściwość jest szczególnie przydatna w przypadku obliczeń, w których korzysta się z algorytmów szybkiej transformacji Fouriera (FFT).

W celu zobrazowania omawianej idei rozważmy fragment kodu procesora firmy Motorola DSP 560xx :

```
rep #liczba–mnożeń
mac x0,y0,a  x(r1)+,x0  y(r5)-,y0
```

Wartości znajdujące się odpowiednio w rejestrach x0 i y0 są wymnażane przez siebie, a wynik tej operacji dodawany jest do stanu aktualnie przechowywanego przez akumulator a (rys. 8). Następnie z pamięci X spod adresu przechowywanego w rejestrze r1 (wartość AdresX) pobierana jest kolejna mnożna do rejestru x0, zaś z pamięci Y spod adresu znajdującego się w rejestrze r5 (wartość AdresY) pobierany jest kolejny mnożnik do rejestru y0. Wartości w rejestrach r1 (AdresX) i r5 (AdresY) są następnie modyfikowane w taki sposób, aby wskazywały na kolejne wartości danych z pamięci, które w następnym cyklu zostaną przez siebie w podobny sposób przemnożone. AdresX jest inkrementowany o 1, natomiast AdresY jest dekrementowany o 1. Wartości w tych rejestrach są adresowane w sposób cykliczny – modulo względem rozmiaru buforów przechowujących kolejne mnożne i mnożniki. Cały cykl operacji instrukcji *mac* wykonywany jest tyle razy – ile wynosi wartość zmiennej liczba_mnożeń.

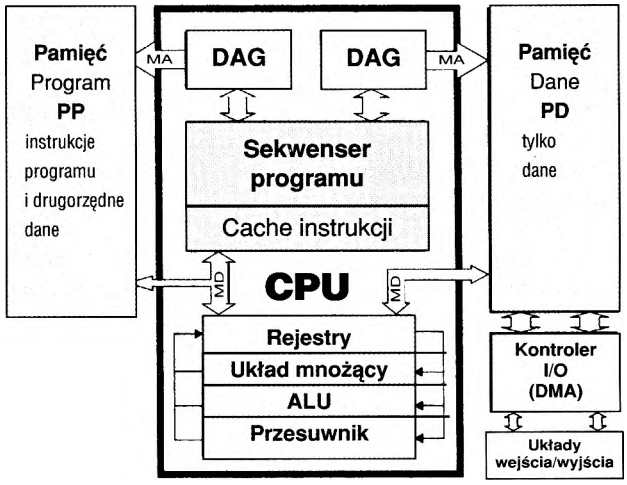
Sama operacja MAC (realizowana w procesorach Motorola instrukcją *mac*, w innych procesorach nazewnictwo jest inne) byłaby nie tak efektywna, gdyby w pamięci przechowywany był programowo zaimplementowany licznik, zliczający ilość dokonanych mnożeń, a następnie go modyfikujący i sprawdzający warunek wyzerowania (rys. 8). W procesorach DSP licznik ten jest zaimplementowany i zarządzany sprzętowo, dzięki czemu w znaczny sposób odciąża właściwą jednostkę obliczeniową.

Na rys. 9 przedstawiono w sposób poglądowy architekturę DSP z wyszczególnieniem jej charakterystycznych elementów, które optymalnie współpracując ze sobą, decydują o jej efektywności w dziedzinie cyfrowego przetwarzania sygnałów.

Przetwarzanie potokowe

Kolejnym mechanizmem jaki wykorzystują procesory sygnałowe jest mechanizm przetwarzania potokowego (*pipelining*). Polega on na wykorzystaniu cykli pracy procesora podobnie do cykli pracy linii montażowej w fabryce (rys. 10).

W jednym cyklu pracy procesor pobiera instrukcję, dekoduje instrukcje pobraną w poprzednim cyklu oraz wykonuje instrukcję pobraną dwa cykle wcześniej. Proces taki trwa w sposób ciągły, a od programisty wymaga się stworzenia takiego kodu niskiego poziomu, aby w pełni ten mechanizm wykorzystać. Na rys. 11 zademonstrowano fragment programu, w którym instrukcja 3 dokonuje skoku do instrukcji 332. Bezpośrednio po tej instrukcji występują instrukcje numer 4 i 5. Zostaną one pobrane i wykonane zgodnie z mechanizmem potoku, zanim zostanie wykonana właściwa instrukcja numer 332. Jeżeli programista nie zadba o optymalne wykorzystanie mechanizmu potoku, wtedy jego przerwanie zaowocuje odpowiednią stratą dwóch cykli procesora.

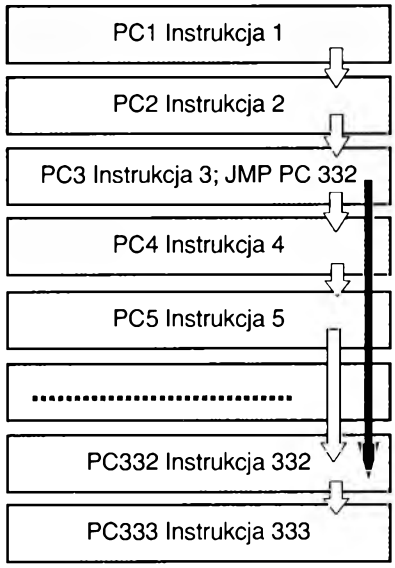


Rys. 9. Schemat blokowy układu DSP, który dzięki odpowiednio zaprojektowanej i zarządzanej architekturze umożliwia sprawne wykonywanie operacji MAC.

Cykle zegarowe	1	2	3	4	5	6	7	8	9
Pobranie	F1			F2			F3		
Dekodowanie		D1			D2			D3	
Wykonanie			E1			E2			E3
	Cykl instrukcji1			Cykl instrukcji 2			Cykl instrukcji 3		

Cykle zegarowe	1	2	3	4	5	6	7	8	9
Pobranie	F1	F2	F3	F4	F5	F6	F7	F8	F9
Dekodowanie		D1	D2	D3	D4	D5	D6	D7	D8
Wykonanie			E1	E2	E3	E4	E5	E6	E7
	Cykl instrukcji1			Cykl instrukcji 4			Cykl instrukcji 7		
		Cykl instrukcji 2			Cykl instrukcji 5				
			Cykl instrukcji 3			Cykl instrukcji 6			

Rys. 10. Zarys zasady działania przetwarzania potokowego. Na górnym rysunku przedstawiono przetwarzanie pozbawione tego mechanizmu



Rys. 11. Kolejność wykonywania instrukcji w potoku. Instrukcje 4 i 5 występują za instrukcją 3 i z tego powodu zostaną uwzględnione w przetwarzaniu potokowym

Dodatkowo, w przypadku gdyby programista nie uwzględnił mechanizmu przetwarzania potokowego, a instrukcje 4 i 5 zawierałyby odwołania konfliktowe względem głównego nurtu programu – to ich niekontrolowane wykonanie mogłoby prowadzić do nieokreślonych błędnych konsekwencji w działaniu całego zaimplementowanego algorytmu.

Przedstawiony tutaj mechanizm potoku jest bardzo ważny, a jego pełne wykorzystanie – szczególnie w algorytmach pracujących w pętlach programowych – odpowiada za ich optymalne i efektywne działanie.

W następnej części artykułu zostanie omówiony przykład demonstrujący praktyczny aspekt implementacji filtra adaptacyjnego LMS na procesorze sygnałowym SHARC 21160 firmy Analog Devices. Przykład ten w praktyczny sposób zobrazuje sposób pisania kodu z uwzględnieniem omówionych w tej części artykułu mechanizmów procesora DSP.

Piotr Wołowik

PROCESORY SYGNAŁOWE DSP (3)

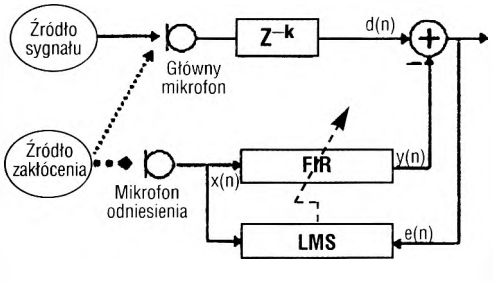
W tej, ostatniej części artykułu Autor podsumuje rozważania dotyczące DSP, przeprowadzone w poprzednich częściach cyklu. Z uwzględnieniem tych rozważań przedstawiony zostanie prosty zarys praktycznego przykładu idei zmiennoprzecinkowej implementacji filtra adaptacyjnego LMS przy pomocy procesora sygnałowego SHARC 21160 firmy Analog Devices.

Algorytm adaptacyjnej filtracji LMS

Filtracja adaptacyjna polega na nieustannym dostrajaniu się układu filtrującego do zmiennych warunków pracy, to jest zmiennego w czasie charakteru zakłócenia, które dany układ ma za zadanie eliminować. Najpopularniejszym algorytmem wykorzystywanym w filtracji adaptacyjnej jest algorytm LMS (*Least Mean Square*).

Rozważmy przykład zastosowania metody filtracji adaptacyjnej, który pomoże dokładnie zobrazować użyteczność tej techniki w praktycznych urządzeniach elektronicznych powszechnego użytku.

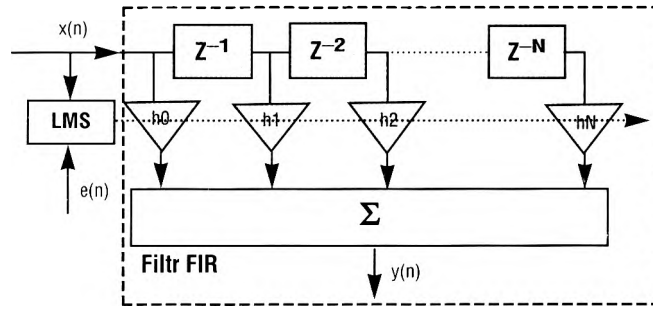
Na rys. 12 przedstawiono schemat układu filtracji adaptacyjnej LMS sygnału mowy w obecności sygnału zakłócającego. Może to być przykład kierowcy rozmawiającego podczas jazdy przez telefon komórkowy w zestawie głośnomówiącym. W takim przypadku mikrofon rejestruje mowę oraz zmieszany z nią sygnał zakłócenia (d(n)). Sygnałem zakłócenia (x(n)) może być na przykład szum otoczenia lub warkot silnika. Jeżeli w samochodzie znajdzie się dodatkowy mikrofon odniesienia rejestrujący tylko hałas zakłócający, to zastosowanie filtracji adaptacyjnej pozwoli usunąć to zakłócenie z sygnału mowy.



Rys. 12. Schemat blokowy układu filtracji adaptacyjnej LMS

Oczywiście zakłócenie rejestrowane przez główny mikrofon i obecne w sygnale mowy oraz zakłócenie rejestrowane przez mikrofon odniesienia różnią się od siebie (jeżeli chodzi o amplitudę i fazę). Wynika to z faktu, że oba mikrofony znajdują się w innych miejscach. Oba sygnały są jednak w pewien sposób ze sobą skorelowane i ta korelacja właśnie jest wykrywana przez algorytm LMS, a następnie wykorzystywana do usunięcia składowej zakłócającej z sygnału mowy kierowcy.

Zakłócenie generowane przez silnik może mieć zmienny charakter, który jednak nie ma wpływu na poprawne działanie układu filtra adaptacyjnego, ponieważ sam filtr nieustannie dostosowuje się do nowych warunków pracy.



Rys. 13. Filtr FIR z przestrajanymi współczynnikami "h" zastosowany w algorytmie filtracji adaptacyjnej LMS

W algorytmie LMS strukturą dokonującą zasadniczej filtracji jest filtr o skończonej odpowiedzi impulsowej SOI (***FIR**-Finite Impulse Response*) – rys. 13.

Zasada jego działania polega na przechowywaniu w rejestrze przesuwającym (w DSP implementowanego jako rejestr cyrkularny) odpowiedniego wektora zarejestrowanych w danym odcinku czasu próbek, które następnie są wymnażane przez stosowne współczynniki "h" i sumowane. Współczynniki filtru FIR nie są stałe, są one zmienne w czasie i dostrajają się tak, aby sygnał y(n) dążył do osiągnięcia stanu, w którym byłby on odpowiednikiem zakłócenia znajdującego się w sygnale d(n). Po operacji odjęcia go od tego sygnału w sygnale błędu e(n) powinna znaleźć się w miarę "czysta" mowa kierowcy (rys. 14).

Współczynniki filtra "h" w zastosowanym algorytmie filtracji adaptacyjnej LMS zmieniają się z każdą nowo zarejestrowaną próbką. Do aktualnych wartości wektora "h" dodawana jest pewna korekta wyliczona przez algorytm LMS. Tak zmodyfikowane wartości stają się w następnym cyklu próbkowania aktualnymi współczynnikami biorącymi udział we właściwej filtracji.

$$H(n+1) = H(n) + \Delta H$$

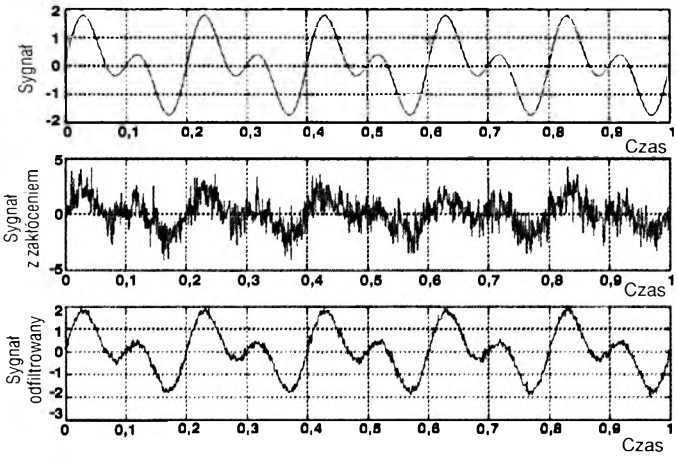
Wektor macierzowy H(n) zawiera zbiór kolejnych wartości współczynników "h" w chwili próbkowania n. Wartość macierzy H(n+1) w następnej chwili próbkowania wyznaczana jest na podstawie aktualnej macierzy H(n) zmodyfikowanej korektą wyliczoną przez adaptacyjny algorytm LMS.

Praktyczna implementacja

Obok podano przykład kodu niskiego poziomu napisany na procesor sygnałowy SHARC 21160 firmy Analog Devices. Przykład ten zawiera tylko algorytm filtracji, sama procedura obliczeń i korekcji nowych wartości współczynników "h" dokonywana przez algorytm filtracji adaptacyjnej LMS – ze względu na ograniczoną objętość artykułu – została pominięta.

W liniach 1÷10 następuje inicjalizacja układów adresujących DAG, wskazujących odpowiednią lokalizację buforów cyrkularnych. Rejestry b0 i b8 zawierają adresy początków tych buforów przechowujących odpowiednio kolejne próbki wejściowe filtra FIR oraz jego współczynniki. W rejestrach l0 i l8 znajdują się rozmiary (długości) tych buforów, a w rejestrach m0 i m8 wartości rozmiaru danych w nich przechowywanych.

W liniach 11÷15 znajduje się kod programu, który powoduje, że procesor pracuje w nieskończonej pętli, oczekując na przerwania z układu próbkującego (w naszym przypadku układu AD1881). Każda nowo pobrana próbka jest sygnalizowana stosownym przerwaniem, dzięki czemu procesor skacze do miejsca wskazywanego przez adres procedury obsługującej dane prze-



Rys. 14. Sygnał oryginalny, który został zakłócony, a następnie odfiltrowany techniką adaptacyjną LMS

```
1. RZAD_FILTRU
2. wej_linia[RZAD_FILTRU]
3. wspolczynniki[RZAD_FILTRU]
4. inicjalizacja_DAG:
5. b0=wej_linia;
6. l0=@wej_linia;
7. m0=1;
8. b8=wspolczynniki;
9. l8=@wspolczynniki;
10. m8=1;
11. wait:
12. idle;
13. jump wait;
14. nop;
15. nop;
16. probka_gotowa_FIR:
17. jmp algorytm_LMS;
18. nop;
19. nop;
20. procedura_filtracji:
21. r0=dm(Mikrofon);
22. r0=lshift r0 by 16;
23. r1=-31;
24. f0=float r0 by r1;
25. dm(i0,m0)=f0;
26. f12=0;
27. f2=dm(i0,m0), f4=pm(i8,m8);
28. f8=f2*f4, f2=dm(i0,m0), f4=pm(i8,m8);
29. lcntr=RZAD_FILTRU-2, do (pc,1) until lce;
30. f8=f2*f4, f12=f8+f12, f2=dm(i0,m0), f4=pm(i8,m8),
31. f8=f2*f4, f12=f8+f12;
32. f12=f8+f12;
33. r1=31;
34. r8=fix f12 by r1;
35. r8=lshift r8 by -16;
36. rti(db);
37. dm(Glosnik)=r8;
38. nop;
39. algorytm_LMS:
40. ...
41. jmp procedura_filtracji;
42. nop;
43. nop;
```

rwanie i zaczyna wykonywać znajdujący się tam kod.

Podwójne instrukcje *nop* znajdujące się po wszystkich instrukcjach *jmp* służą do zademonstrowania właściwości mechanizmu przetwarzania potokowego, jako że zostaną one w nim uwzględnione. Instrukcje te zupełnie nic nie robią, a jedynie zapełniają potok – aby nie został on "zaśmiecony" innymi instrukcjami mogącymi rodzić sytuację konfliktową w programie.

W linii 16 znajduje się właściwa procedura obsługi przerwania zgłaszanego przez układ AD1881. W linii 17 skacze ona do algorytmu LMS (linia 39), gdzie pobierane są próbki z mikrofonu odniesienia, na których podstawie obliczane są stosowne współczynniki filtra. Następnie następuje powrót z tego algorytmu (linia 41) do linii 20. W liniach 21÷25 następuje pobranie próbki z mikrofonu kierowcy, przekształcenie jej z postaci stałoprzecinkowej na zmiennoprzecinkową (układ AD1881 używa stałoprzecinkowego formatu danych) i umieszczenie jej w odpowiednim buforze cyrkularnym. W liniach 26÷35 znajduje się właściwy algorytm filtracji FIR, który po obliczeniu stosownej próbki będącej wynikiem przemnożenia przez stosowne współczynniki filtra i zsumowania, konwertuje ją z postaci zmiennoprzecinkowej na stałoprzecinkową (linie 33÷35). W linii 36 następuje powrót z obsługi przerwania, a za tą instrukcją w liniach 37 i 38 znajduje się zwracana do odpowiedniej zmiennej stosownie przefiltrowana próbka mono. Takie ich umiejscowienie (tych instrukcji) wynika z chęci uzyskania efektywnie działającego mechanizmu potoku, aby nie został on nigdzie niekorzystnie przerwany.

***

Autor ma nadzieję, że przedstawiony w tym cyklu artykułów zarys wiedzy o procesorach DSP pozwolił Czytelnikom zrozumieć podstawowe idee, a także może nawet zainteresować ich na tyle, aby samodzielnie podjęli się trudu wgłębienia w tę dziedzinę. Nie jest ona łatwa, ale warta poznania, gdyż w najbliższej przyszłości można się spodziewać znacznego wzrostu zastosowań systemów i układów z procesorami DSP. ■

Piotr Wołowik