

Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki
Instytut Elektroniki

Praca dyplomowa magisterska
„Łączna redukcja szumów i echa akustycznego”
Bartosz Zator

Promotor:
dr inż. Jacek Izydorczyk

Gliwice 2006

Bartosz Zator
„Łączna redukcja szumów i echa akustycznego”

Uwagi, komentarze: braton@interpc.pl

Wygenerowano za pomocą: pdfM_EX v. 1.05

Spis treści

Wstęp.....	ii
Cel i zakres pracy	iii
Notacja.....	iv
I. Przegląd algorytmów eliminacji echa akustycznego oraz redukcji szumów	1
1 Problem echa akustycznego.....	2
2 Filtry adaptacyjne minimalizujące błąd średniokwadratowy	5
3 Algorytmy adaptacyjne działające w oparciu o metodę najmniejszych kwadratów....	14
4 Algorytmy redukcji szumów	23
5 Łączna redukcja szumów i echa akustycznego.....	34
6 Podsumowanie	37
II. Implementacja algorytmu łącznej redukcji szumów i echa akustycznego	
z wykorzystaniem procesora sygnałowego	38
1 Budowa i cechy charakterystyczne procesora sygnałowego	39
2 Zestaw uruchomieniowy procesora sygnałowego	43
3 Środowisko programistyczne procesora sygnałowego	46
4 Wybrany algorytm łącznej redukcji szumów i echa akustycznego	50
5 Implementacja algorytmu łącznej redukcji szumów i echa akustycznego	
w języku asemblera procesora sygnałowego.....	61
6 Wyniki cyfrowego przetwarzania sygnałów przez procesor sygnałowy	81
7 Podsumowanie	86
8 Literatura	88
Spis rysunków.....	91
Spis tabel	93
Spis algorytmów.....	94
Spis procedur	95

Wstęp

Kiedy powstawał pierwszy mikroprocesor, jego konstruktorzy pewnie nie przewidzieli, jak ogromny sukces odniesie on na przestrzeni lat. Coś, co było dostępne jedynie dla wybranych, jest dziś obecne praktycznie w każdym produkowanym urządzeniu użytkowym: samochodach, artykułach gospodarstwa domowego, radiu, telewizji itp. W dzisiejszych czasach nie można sobie wyobrazić życia bez wszechobecnej komputeryzacji. Nowoczesna telekomunikacja, przemysł, nauka – wszystko to opiera się na tym niesamowicie precyzyjnym kawałku krzemu.

Wraz z rozwojem technologii elektronicznej rozwinęła się również powiązana z nią dziedzina: cyfrowe przetwarzanie sygnałów. Jego cechą charakterystyczną jest to, że możliwym staje się przekształcenie sygnału cyfrowego właściwie w dowolny sposób. Za jego pomocą możemy generować muzykę, obrazy trójwymiarowe czy sygnał sztucznej mowy. Kodowanie audio i video, wykrywanie i korekcja błędów w telekomunikacji czy np. w zwyczajnym dysku twardym w komputerze to domena cyfrowego przetwarzania sygnałów. Przetwarzanie sygnałów medycznych pomaga w lokalizacji zmian chorobowych wewnętrz organizmu ludzkiego. Tak znienawidzone przez wszystkich kierowców radary bądź używane w łodziach podwodnych sonary działają na zasadzie niezwykle precyzyjnego pomiaru częstotliwości i wymagają bardzo dużych szybkości przetwarzania. Pomiar danych sejsmologicznych pozwala ostrzec przed zbliżającym się niebezpieczeństwem, jak też jest wykorzystywany do lokalizacji złóż surowców naturalnych.

Powstało wiele dziedzin cyfrowego przetwarzania sygnałów: przetwarzanie mowy, przetwarzanie obrazów, filtracja cyfrowa, statystyczne przetwarzanie sygnałów. Niniejsza praca poświęcona jest zastosowaniu pewnej odmiany filtracji cyfrowej – filtracji adaptacyjnej, oraz wykorzystaniu niektórych algorytmów statystycznych do zmagania się z problemem echa akustycznego oraz zakłóceniami szumowymi obecnymi w rejestrowanych sygnałach. Wykonanie algorytmów cyfrowego przetwarzania sygnałów za pomocą specjalistycznych mikroprocesorów pozwala na znaczne ich zredukowanie, a czasem nawet całkowite usunięcie.

Cel i zakres pracy

Celem pracy było zapoznanie się, przebadanie, a następnie zaimplementowanie i uruchomienie algorytmów cyfrowej eliminacji echa akustycznego i redukcji szumów zakłócających sygnał mowy. Praca podzielona jest na dwie części. W pierwszej z nich zaprezentowany jest przegląd spotykanych w literaturze specjalistycznej i stosowanych w praktyce rozwiązań omawianych problemów. Obejmuje on zagadnienie filtrów adaptacyjnych, najszerzej stosowanych w eliminacji echa akustycznego. Omówione są algorytmy historyczne najwcześniejsze, bazujące na minimalizacji błędu średniokwadratowego, jak też stosunkowo nowe, korzystające z metody najmniejszych kwadratów. Podkreślone zostały zalety poszczególnych rozwiązań, jak też wypunktowane zostały ich wady i wymagania implementacji. Zaprezentowany został liniowy model powstawania fenomenu echa oraz pokazane zostało, jak za pomocą wspomnianych procedur adaptacji można je w znaczny sposób zredukować.

Drugim niekorzystnym zjawiskiem branym pod uwagę w procesie uwydatniania sygnału mowy jest nieuniknione pojawienie się szumu. Jest on jako zjawisko losowe nieodłącznym elementem pracy układów elektronicznych, stałym czynnikiem występującym w naszym otoczeniu, w życiu codziennym, skąd niezwłocznie musi pojawić się również w zmierzonym i poddanym obróbce sygnale mowy. W pierwszej części przedstawione zostały wypracowane na przestrzeni lat w dziedzinie cyfrowego przetwarzania sygnałów metody walki z szumem za pomocą jedynej dostępnej broni – statystyki.

Wszystkie opisane algorytmy zostały zaimplementowane oraz przetestowane w środowisku matematycznym MATLAB. Wyniki symulacji przedstawione są w postaci wykresów czy danych tabelarycznych. Celem tak szerokiego przeglądu było głębokie poznanie poszczególnych przyimiotów istniejących algorytmów, aby wybrać jeden z nich, o możliwie najlepszych parametrach, a także takich wymaganiach sprzętowych aby można je było spełnić za pomocą dostępnego sprzętu.

Mikrokomputerem, którego użyto do realizacji wybranych procedur w czasie rzeczywistym, był procesor sygnałowy ADSP 21061 Sharc firmy Analog Devices. W drugiej części pracy omówiony został w szczegółach zaproponowany algorytm, zaprezentowane zostało środowisko programistyczne procesora sygnałowego, przedstawione zostały wybrane ciekawsze cechy procesora oraz pokazane zostały niektóre interesujące szczegóły implementacji.

Pracę podsumowują wnioski wyciągnięte z wyników pomiarów, jak też z badań teoretycznych.

Notacja

W wielu miejscach pracy zostały zaprezentowane podstawy matematyczne omawianych algorytmów w celu zredukowania do minimum ewentualnych wątpliwości. Aby czytanie było tak wygodne, jak to tylko możliwe, przyjęto omówioną poniżej notację. Sygnał wejściowy filtru adaptacyjnego to skończony ciąg próbek wejściowych zapamiętyany w pamięci filtru zwanej regresorem, oznaczony poprzez u_i lub $u_{X,i}$. Regresor jest zawsze wektorem wierszowym o długości X oraz, jeśli nie jest wyspecyfikowane inaczej, zawiera próbki sygnału wejściowego pochodzące z tak zwanej linii opóźniającej:

$$u_{X,i} = [u(i) \quad u(i-1) \quad \dots \quad u(i-X+1)]$$

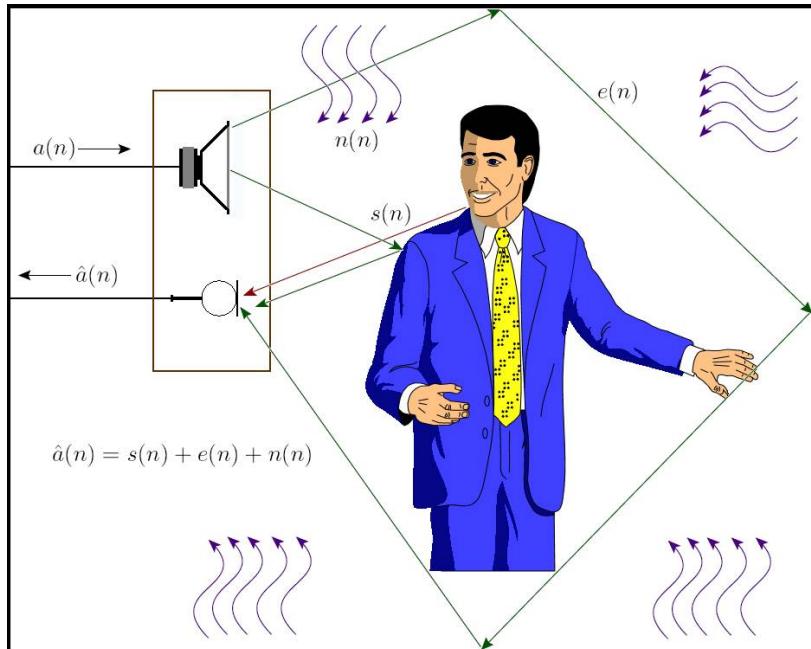
Pominięcie długości wektora w przypadku u_i oznacza, że ma on w omawianej procedurze stałą, założoną początkowo długość. Indeks i odnosi się do aktualnej chwili czasowej. Wszystkie pozostałe używane wektory (notowane podobnie) są wektorami kolumnowymi. Przykładowo, wektor $v_{K,i-4}$ oznacza wektor kolumnowy składający się z K elementów zmierzony w chwili czasu $i-4$. Dużą literę zarezerwowano na notację macierzy (jeśli w tekście nie pisze wyraźnie inaczej). Oznaczenie $A_{NK,i}$ przedstawia macierz o N wierszach zawierających K kolumn z dodatkowym indeksem czasowym. Gdy zastosowano oznaczenie $U_{K,i}$, mamy do czynienia z kolekcją K ostatnich regresorów w kolejnych wierszach macierzy $U_{K,i}$, skąd wiadomo, że liczba jej kolumn wynosi tyle, ile liczba współczynników filtru.



Przegląd algorytmów eliminacji echa akustycznego oraz redukcji szumów

1. Problem echo akustycznego

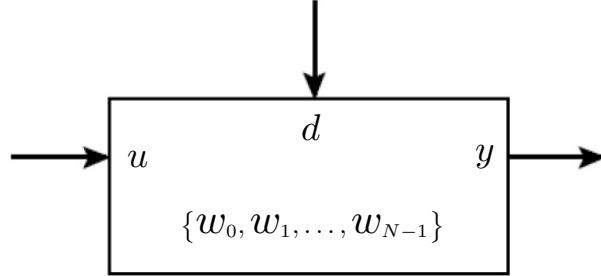
Echo akustyczne powstaje w pomieszczeniach zamkniętych, w których źródło dźwięku – głośnik, sprężone jest ścieżką akustyczną z urządzeniem rejestrującym sygnał dźwiękowy – mikrofonem. Typowym przykładem jest telefon konferencyjny przedstawiony schematycznie na poniższym rysunku.



Jest rzeczą oczywistą, że osoba znajdująca się po drugiej stronie linii telefonicznej chciałaby słyszeć tylko i wyłącznie nasze wiadomości oznaczone kolorem czerwonym i symbolem $s(n)$. Niestety fale wydobywające się z głośnika rozchodzą się we wszystkich kierunkach, ulegając odbiciom od przeszkód słabo pochłaniających dźwięk, takich jak ściany, podłoga czy osoby ludzkie, przecinające ich trajektorie, i docierają z większym lub mniejszym opóźnieniem do mikrofonu. W efekcie ze strony zdalnego rozmówcy wygląda to tak, że wypowiada jakieś zdanie, by po chwili usłyszeć je ponownie mniej lub bardziej stłumione. Te powracające zakłócające fale dźwiękowe to właśnie echo akustyczne oznaczone kolorem zielonym i symbolem $e(n)$. Oczywiście istnieją też pozytywne strony echo akustycznego. Np. w filharmonii czy operze jest ono wręcz pożąданie. Jednak tutaj występuje wyłącznie jako zjawisko negatywne. Odbite fale są w różny sposób postrzegane przez człowieka, w zależności od czasu, po którym zostaną zarejestrowane przez mózg za pomocą aparatu słuchu. Fale o małym opóźnieniu nie są postrzegane jako osobny opóźniony sygnał, ich wpływ ujawnia się w zmianie głośności czy barwy dźwięku, inne jest również przestrzenne postrzeganie dźwięku. Człowiek błędnie lokalizuje jego źródło [27]. Dopiero przy wyższych czasach opóźnienia można oddzielić mentalnie sygnał źródłowy od jego echo. Ważny parametr pomieszczenia, w którym rejestrowane jest echo stanowi tzw. czas rewerberacji T_R . Jest to czas, po którym poziom ciśnienia akustycznego fali ulegającej odbiciu spadnie poniżej 60 dB wartości początkowej. Poziom ciśnienia akustycznego fali dźwiękowej zdefiniowany jest jako [36]:

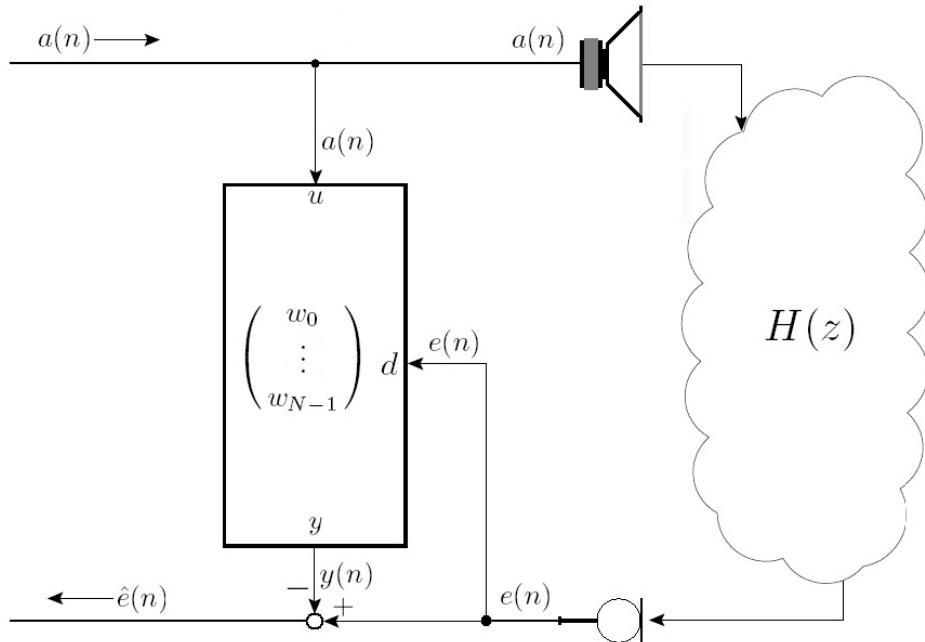
$$(1.1) \quad L_{SPL} = 20 \log_{10} \frac{p}{p_0} [\text{dB SPL}]; \quad p_0 = 20 \mu\text{Pa}; \quad p - \text{ciśnienie fali dźwiękowej.}$$

Jest to standardowa miara intensywności pobudzenia, wartość $L_{SPL} = 0 \text{ dB SPL}$ przyjęta jest jako poziom słyszalności, wartość $L_{SPL} = 140 \text{ dB SPL}$ jako poziom bólu.



RYS. 2 Schemat filtru adaptacyjnego

Jednym z pierwszych rozwiązań problemu echa akustycznego w telefonii było zastosowanie detektorów sygnału mowy [11, 45]. Obie linie abonentów odległego – nadawcza $a(n)$ oraz odbiorcza $\hat{e}(n)$ – posiadały taki detektor, który po wykryciu aktywności w jednej z linii włączał tłumienie w drugiej z nich. Zbudowanie niezawodnego detektora sygnału mowy jest rzeczą trudną, szczególnie w obecności szumu, poza tym problemy pojawiły się, gdy obydwa rozmówcy podjęli chęć komunikacji w tej samej chwili. O wiele doskonalszym rozwiązaniem okazało się zastosowanie filtrów adaptacyjnych.



RYS. 3 Eliminacja echo akustycznego z wykorzystaniem filtru adaptacyjnego

Filtr adaptacyjny jest to właściwie pewien algorytm cyfrowego przetwarzania sygnałów, realizujący filtr cyfrowy o specjalnych właściwościach. Tą specjalną właściwością są zmienne w czasie współczynniki filtru. Filtr adaptacyjny posiada dwa wejścia. Pierwsze z nich to standardowe wejście dla sygnału poddawanego filtracji, zaznaczone na rysunku 2 jako u . Drugie to tzw. wejście sygnału pożdanego d . Filtr adaptacyjny dobiera tak swoje współczynniki, aby sygnał wyjściowy

po filtracji y był możliwie najbliższy sygnałowi pożądanemu. Dodatkowo filtr ten reaguje na zmiany w statystyce sygnału d , zmieniając swoje współczynniki w rytm zmian właściwości statystycznych sygnału pożądanego. Można powiedzieć, że jego współczynniki podlegają ciągłej adaptacji do sygnałów wejściowych. Z tego punktu widzenia eliminacja echa akustycznego z wykorzystaniem filtru adaptacyjnego jest bardzo prosta.

Zwykle przyjmuje się liniowy model środowiska akustycznego. Oznacza to, że sygnał echa $e(n)$ rejestrowany przez mikrofon jest sygnałem z głośnika $a(n)$ po przejściu przez pewien filtr liniowy $H(z)$. Jeśli przyjmiemy to założenie za słuszne, to doprowadzenie sygnału echa na wejście d filtru adaptacyjnego spowoduje dobranie przez niego współczynników możliwie najbliższych współczynnikom filtru otoczenia $H(z)$. Jeśli to nastąpi, na wyjściu filtru adaptacyjnego pojawi się sygnał możliwie najbliższy sygnałowi echa. Po odjęciu przesłany zostanie sygnał możliwie najbliższy zeru.

W kolejnych punktach zostanie zaprezentowana tematyka filtrów adaptacyjnych, zasady ich działania oraz ich właściwości.

2. Filtry adaptacyjne minimalizujące błąd średniokwadratowy

Klasa filtrów adaptacyjnych minimalizujących błąd średniokwadratowy jest najstarszą klasą algorytmów adaptacji. Wynika to wprost z prac Kołmogorowa i Wienera z lat trzydziestych i czterdziestych XX wieku [19, 24, 43]. Zakładamy, że na wejście filtru podawane są zmienne losowe o zewnetrznej wartości średniej u, d . Jak z samej nazwy wynika, współczynniki filtru dobierane są w taki sposób, aby wartość średnia kwadratu sygnału błędu na wyjściu filtru była minimalna.

$$(2.1) \quad \begin{aligned} w : J_{\min}(w) &= E(d - y)^2 \Big|_{\min} = E(d - uw)^2 \Big|_{\min}, \\ w &= [w_0, w_1, \dots, w_{N-1}]^T. \end{aligned}$$

Wartości optymalnych współczynników dane są wzorem:

$$(2.2) \quad w^{\text{opt}} = R_u^{-1} r_{du},$$

gdzie R_u, r_{du} – macierz autokorelacji sygnału wejściowego i wektor krzyżowy korelacji sygnałów wejściowych odpowiednio:

$$(2.3) \quad R_u = E(u^T u), \quad r_{du} = E(d u^T).$$

Aby uniknąć odwracania macierzy autokorelacji można wyznaczyć współczynniki filtru w sposób iteracyjny, zbliżając się do optymalnego rozwiązania w sukcesywnych chwilach czasowych. Osiąga się to, wymuszając, aby wartość błędu średniokwadratowego w chwili czasu i była mniejsza od wartości w poprzedniej chwili czasowej: $J(w_i) < J(w_{i-1})$. Prowadzi to do równania rekurencyjnego:

$$(2.4) \quad w_i = w_{i-1} - \mu B [\nabla_w J(w_{i-1})].$$

B jest dowolną macierzą dodatnio określona, μ to krok iteracji, a $\nabla_w(\cdot)$ oznacza operację gradientu

$$(2.5) \quad \nabla_w = \left[\frac{\partial}{\partial w_0}, \frac{\partial}{\partial w_1}, \dots, \frac{\partial}{\partial w_{N-1}} \right].$$

Wybór różnych macierzy B prowadzi do różnych algorytmów. Najprostszym wyborem jest oczywiście $B = I$. Prowadzi to do znanej metody najszybszego spadku:

$$(2.6) \quad w_i = w_{i-1} + \mu [r_{du} - R_u w_{i-1}].$$

Z kolei wybór $B = [\epsilon I + \nabla_w^2 J(w_{i-1})]^{-1}$ prowadzi do tzw. algorytmu Newtona:

$$(2.7) \quad w_i = w_{i-1} + \mu [\epsilon I + R_u]^{-1} [r_{du} - R_u w_{i-1}].$$

Znamienną cechą algorytmu Newtona jest zbieżność do wartości optymalnej już w pierwszej iteracji. Odbywa się to za cenę odwracania macierzy R_u . Użyteczność algorytmu Newtona zostanie uwypuklona po uniezależnieniu algorytmów od dokładnych wartości macierzy statystycznych. Wartość ϵ to tzw. parametr regulujący – mała wartość dodatnia – umożliwia odwrócenie macierzy w przypadku, gdy macierz R_u jest bliska macierzy osobliwej.

Opisane powyżej procedury iteracyjne mają zwykle znaczenie czysto akademickie i stanowią jedynie dobrą bazę do konstrukcji prawdziwie „adaptacyjnych” algorytmów. Dzieje się tak z dwóch powodów. Po pierwsze: znajomość dokładnych wartości macierzy korelacji jest bardzo

rzadka w przypadku rzeczywistych mierzonych sygnałów. Po drugie: rzeczywiste sygnały przeważnie nie są stacjonarne, co sprawia, że ich macierze korelacji zmieniają się w czasie. Aby algorytm posiadał możliwość śledzenia zmian w adaptowanej odpowiedzi impulsowej otoczenia (co ma miejsce np. przy otwarciu okna w mierzonym pomieszczeniu) musi śledzić również zmiany własności statystycznych sygnałów wejściowych.

W pewne piątkowe popołudnie 1959 roku na Uniwersytecie Stanforda podczas owocnej dyskusji o algorytmie najszybszego spadku narodził się pomysł, aby przybliżyć macierze statystyczne, wykorzystując tylko i wyłącznie najnowsze dane zaobserwowane na wejściu filtru [56]. Innymi słowy: wartości oczekiwane macierzy zastąpiono ich natychmiastowymi wartościami:

$$(2.8) \quad R_u = E(u^T u) \approx u_i^T u_i, \quad r_{du} = E(du^T) \approx d(i)u_i^T,$$

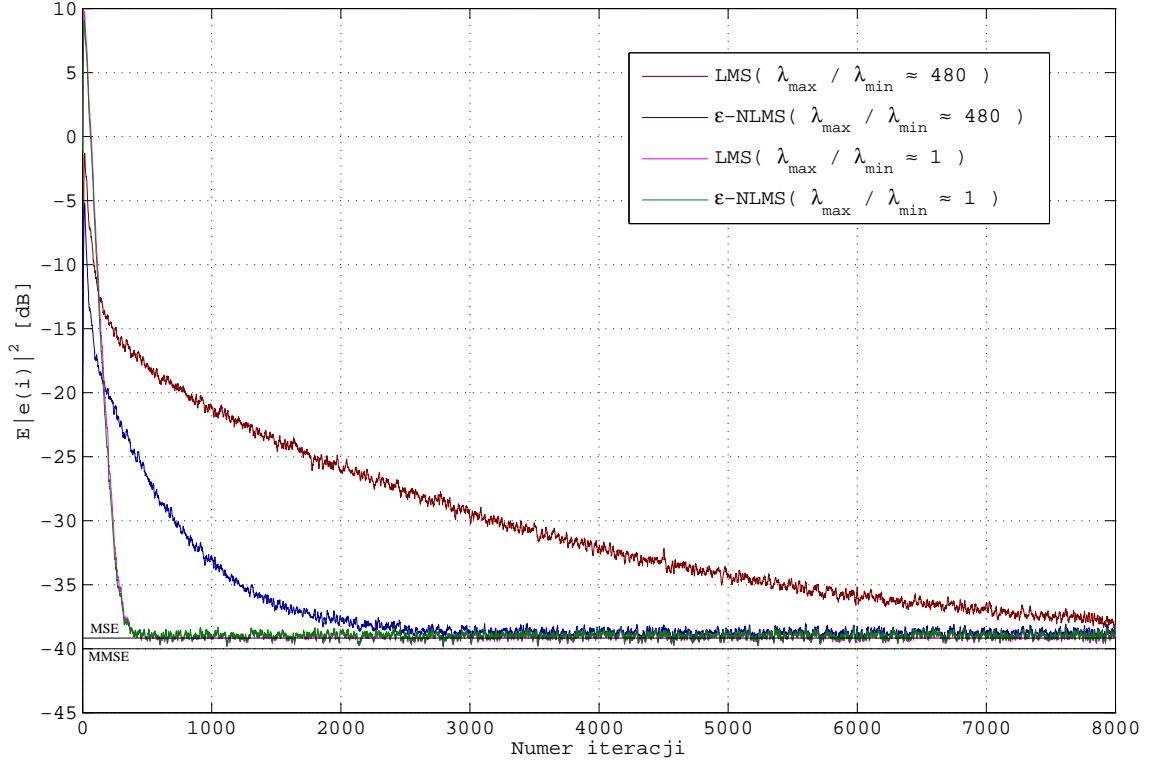
co zaowocowało jednym z najbardziej znanych i chętnie stosowanych przez wiele lat algorytmów cyfrowego przetwarzania sygnałów – algorytmem LMS:

$$(2.9) \quad w_i = w_{i-1} + \mu u_i^T [d(i) - u_i w_{i-1}].$$

Cudowną jego właściwością była jego niesamowita prostota. W swojej standardowej wersji wymagał jedynie $2N$ mnożeń, gdzie N jest liczbą współczynników filtru. Trzeba pamiętać, że w tamtych czasach operacja mnożenia była najbardziej czasochłonną operacją dla ówczesnych arytmometrów. Wykonując podobne podstawienie dla algorytmu Newtona, otrzymujemy tzw. znormalizowany algorytm LMS – ϵ -NLMS:

$$(2.10) \quad w_i = w_{i-1} + \frac{\mu}{\epsilon + \|u_i\|^2} u_i^T [d(i) - u_i w_{i-1}].$$

Podstawową wadą algorytmu LMS jest zależność szybkości zbieżności od zawartości widmowej sygnału wejściowego. Precyzyjnie mówiąc: im większy rozrzut wartości własnych macierzy autokorelacji: $\eta = \frac{\lambda_{\max}}{\lambda_{\min}}$, tym wolniej znajdowane jest optymalne rozwiązanie. Dzięki normalizacji kroku iteracji w algorytmie ϵ -NLMS w stosunku do energii próbek wejściowych zgromadzonych w regresorze szybkość zbieżności została w znacznym stopniu poprawiona. Widać to wyraźnie na rysunku 4. Przedstawia on wykres błędu średniokwadratowego wyznaczony na podstawie 300 niezależnych uśrednień pracy algorytmów LMS i ϵ -NLMS na różnych danych wejściowych. Liczba współczynników filtra N przyjęta została na 16. Sygnałem wejściowym filtra był raz gaussowski szum biały, a następnie szum biały po filtracji dolnoprzepustowej filtrem o transmitancji $H(z) = \frac{\sqrt{1-\alpha^2}}{1-\alpha z^{-1}}$. Funkcja autokorelacji sygnału wejściowego: $r_{uu}(k) = \alpha^{|k|}$, a rozrzut wartości własnych macierzy autokorelacji ≈ 480 (dla $\alpha = 0.95$). Sygnał pożądany wyznaczono przez filtrację sygnału wejściowego przez filtr o 16 stałych współczynnikach, dodatkowo nakładając szum o energii na poziomie -40 dB. Tyle więc wyniosła minimalna wartość błędu możliwa do osiągnięcia przez algorytm – MMSE. Krok iteracji każdego z algorytmów został dobrany tak, by wartość niedopasowania zdefiniowana jako rozrzut wartości błędu średniokwadratowego w stanie ustalonym MSE od jego wartości minimalnej: $M = \frac{\text{MSE-MMSE}}{\text{MMSE}}$, była na poziomie 10%.



RYS. 4 Krzywe błędu algorytmów LMS i ϵ -NLMS dla sygnału „białego” i „kolorowego”

Algorytmy LMS i ϵ -NLMS zostały skonstruowane poprzez użycie bardzo prostych, niemal natychmiastowych przybliżeń funkcji korelacji. Można oczekiwąć, że stosując bardziej wyszukane formy aproksymacji, parametry algorytmów ulegną dalszej poprawie. Tak też jest w istocie. Uogólnieniem algorytmu ϵ -NLMS jest tzw. algorytm ϵ -APA-K (Affine Projection). Powstaje on przez zastąpienie macierzy statystycznych przybliżeniami wyliczonymi na podstawie K ostatnich próbek sygnałów wejściowych:

$$(2.11) \quad R_u = E(u^T u) \approx \frac{1}{K} \sum_{j=i-K+1}^i u_j^T u_j, \quad r_{du} = E(d u^T) \approx \frac{1}{K} \sum_{j=i-K+1}^i d(j) u_j^T.$$

Równanie odświeżania współczynników:

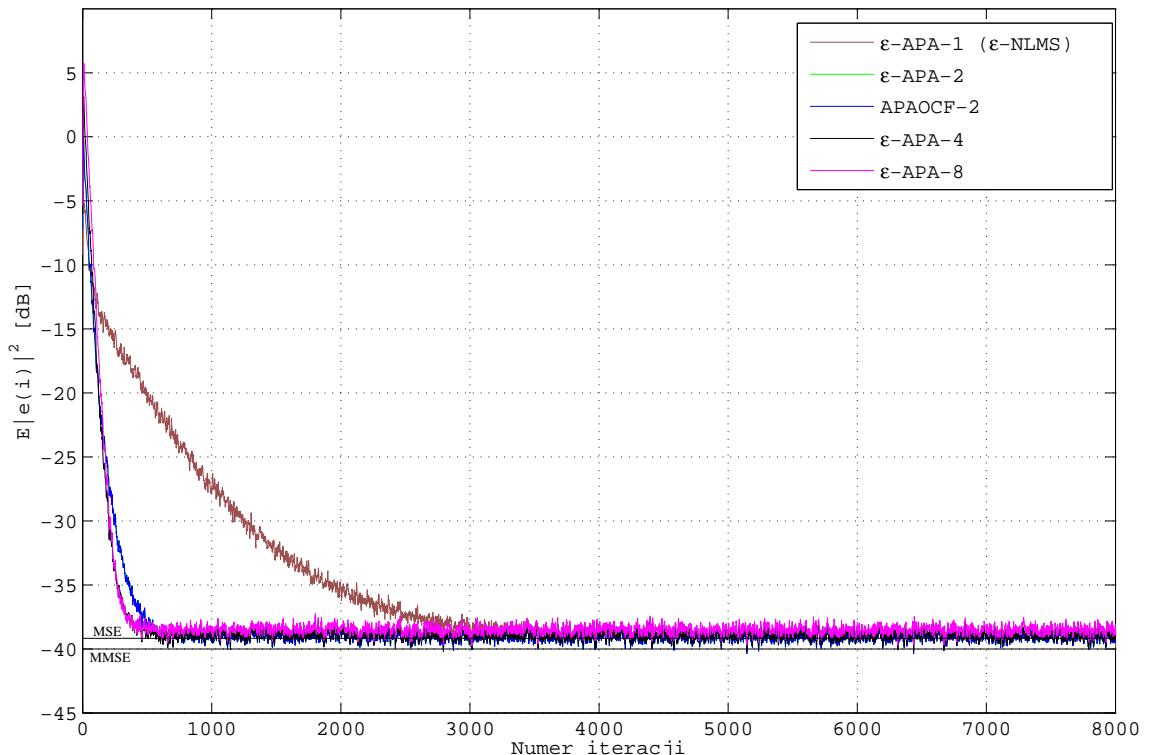
$$(2.12) \quad w_i = w_{i-1} + \mu U_i^T [\epsilon I + U_i U_i^T]^{-1} (d_i - U_i w_{i-1}).$$

Macierz U_i zawiera K ostatnich regresorów, a wektor d_i – K ostatnich próbek sygnału pożądanego:

$$(2.13) \quad U_i = \begin{bmatrix} u_i \\ u_{i-1} \\ \vdots \\ u_{i-K+1} \end{bmatrix} = \begin{bmatrix} u(i) & u(i-1) & \dots & u(i-N+1) \\ u(i-1) & u(i-2) & \dots & u(i-N) \\ \vdots & \vdots & & \vdots \\ u(i-K+1) & u(i-K) & \dots & u(i-K-N+2) \end{bmatrix},$$

$$(2.14) \quad d_i = [d(i), d(i-1), \dots, d(i-K+1)]^T.$$

Symulacje komputerowe wskazują na znaczną poprawę szybkości zbieżności wraz ze wzrostem rzędu projekcji. Pokazują również, że dla dużych K wzrost ten zdaje się nasycić. Niestety wraz ze zwiększeniem wartości K złożoność obliczeniowa również wzrasta drastycznie w porównaniu ze standardowym LMS. Największy nakład obliczeniowy przypada na czynnik $U_i U_i^T$ i jest rzędu $K^2 N$ operacji arytmetycznych. Ponadto odwrócenie macierzy $\epsilon I + U_i U_i^T$ wnosi K^3 operacji do wykonania. Godną uwagi implementacją algorytmu APA jest tzw. APA z prostopadłymi współczynnikami korekcyjnymi (APAOCF)[43]. Idea algorytmu polega na poddaniu zebranych w macierzy U_i regresorów procedurze Grama-Schmidta, otrzymując zbiór regresorów wzajemnie prostopadłych \hat{U}_i . W efekcie macierz $\hat{U}_i \hat{U}_i^T$ jest diagonalna i jej odwrotność jest łatwa do wyznaczenia. Jednakże złożoność w dalszym ciągu pozostaje rzędu $K^2 N$, co może sprawić trudności w realizacji algorytmu w czasie rzeczywistym dla dużych rzędów projekcji K .



RYS. 5 Krzywe błędu algorytmów ϵ -APA różnych rzędów

Jak już zostało wspomniane, algorytm LMS wykazuje najlepsze właściwości, gdy próbki sygnału wejściowego zgromadzone w regresorze są z sobą nieskorelowane. Stąd narodził się pomysł, by postarać się spełnić powyższy warunek nawet dla „kolorowych” sygnałów wejściowych. Idea opiera się na poddaniu wektora próbek wejściowych działaniu pewnej transformacji dekorelującej. Przetworzony regresor przyjmie wtedy postać $\tilde{u}_i = u_i T$, gdzie T jest macierzą transformacji. Zasadzamy, że jest to transformacja unitarna, tj. $TT^* = T^*T = I$. Przyjmując dodatkowo $\tilde{w}_i = T^*w_i$ i mnożąc obie strony równania odświeżania współczynników algorytmu LMS z lewej strony przez T^* , otrzymujemy:

$$(2.15) \quad \tilde{w}_i = \tilde{w}_{i-1} + \mu \tilde{u}_i^* [d(i) - \tilde{u}_i \tilde{w}_{i-1}].$$

Jest to nic innego jak równanie LMS dla przetransformowanych próbek wejściowych. Wprowadzając dodatkowo diagonalną macierz normalizującą D , równanie przyjmie postać:

$$(2.16) \quad \tilde{w}_i = \tilde{w}_{i-1} + \mu D^{-1} \tilde{u}_i^* [d(i) - \tilde{u}_i \tilde{w}_{i-1}].$$

Umieszczona macierz spełnia podobne zadanie jak czynnik $\|u_i\|^2$ w równaniu ϵ -NLMS, z tą różnicą, że teraz każda próbka regresora może mieć inną wartość normalizującą. Przypisując $D^{\frac{1}{2}}$ wartości pierwiastków kwadratowych macierzy D , $\tilde{w}_i = D^{\frac{1}{2}} \tilde{w}_i$ oraz $\tilde{u}_i = u_i D^{\frac{1}{2}}$, a następnie mnożąc obie strony równania przez $D^{\frac{1}{2}}$, otrzymujemy:

$$(2.17) \quad \acute{w}_i = \acute{w}_{i-1} + \mu \acute{u}_i^* [d(i) - \acute{u}_i \acute{w}_{i-1}].$$

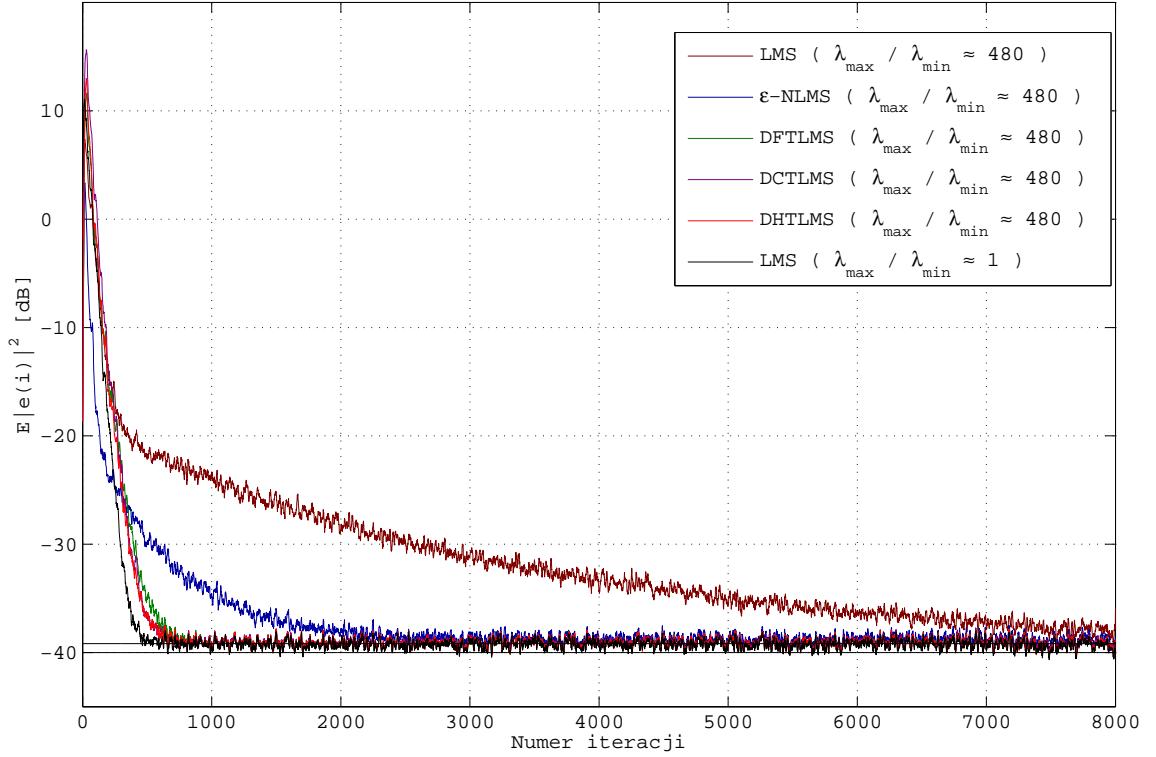
Po raz wtóry pojawia się równanie LMS dla danych wejściowych $\acute{u}_i = u_i T D^{\frac{1}{2}}$. Macierz autokorelacji takiego przebiegu wejściowego jest równa $R_{\acute{u}} = D^{-\frac{1}{2}} T^* R_u T D^{\frac{1}{2}}$. Gdyby teraz dobrać macierze T i D w taki sposób, aby $R_{\acute{u}}$ uczynić macierzą jednostową, osiągnelibyśmy zamierzony cel. Można to zrobić, przyjmując $T = U$, gdzie U składa się z unormowanych do jedności wektorów własnych macierzy R_u . Taka transformacja nosi nazwę przekształcenia Karhunen-Loeve'go i jest znana jako idealna transformacja dekorelująca. Przy takim wyborze $T^* R_u T$ jest równe $U^* R_u U = \Lambda$, gdzie Λ jest macierzą diagonalną wartości własnych R_u . Przypisując $D = \Lambda$, ostatecznie otrzymujemy $R_{\acute{u}} = I$.

W praktycznych algorytmach nie da się zaimplementować transformacji Karhunen-Loeve'go, gdyż wymagałoby to znajomości macierzy autokorelacji. Jednak istnieją inne przekształcenia będące dobrym przybliżeniem idealnej transformacji dekorelującej. Najbardziej znane to oczywiście dyskretna transformacja Fouriera, a także dyskretna transformacja cosinusowa czy dyskretna transformacja Hartley'a [39]. Poszczególne elementy diagonalne macierzy D oblicza się za pomocą równania rekurencyjnego:

$$(2.18) \quad \lambda_k(i) = \beta \lambda_k(i-1) + (1-\beta) |\tilde{u}_i(k)|^2,$$

gdzie k jest indeksem elementu przekątniowego, a β – współczynnikiem wygładzania. Na wykresie 6 przedstawiono wyniki pracy algorytmów dekorelujących w porównaniu z algorytmami standardowymi. Widać wyraźnie, że zbiegają niewiele wolniej niż dla idealnego przypadku, gdy na wejście podano sygnał „biały”.

Rodzina algorytmów LMS charakteryzuje się najmniejszą złożonością obliczeniową w dziedzinie filtracji adaptacyjnej. Najpowszechniej stosowany przedstawiciel tej rodziny ϵ -NLMS wymaga do poprawnej pracy rzędu $6N$ operacji arytmetycznych na próbce sygnału wejściowego. Mimo wszystko nawet tak małe wymagania mogą okazać się nie do spełnienia w pewnych zastosowaniach cyfrowego przetwarzania sygnałów. Doskonałym przykładem jest właśnie eliminacja echa akustycznego. Dla pomieszczeń o dużych czasach reverbacji zastosowanie filtrów o kilku tysiącach współczynników nie jest żadnym ewenementem. Wynika stąd potrzeba dalszej redukcji złożoności obliczeniowej algorytmów. Jednym z takich rozwiązań jest zastosowanie algorytmów blokowych. Zamiast przetwarzać dane próbka po próbce – jak ma to miejsce w standardowych procedurach, narodził się pomysł, by zebrać pewną liczbę kolejnych próbek wejściowych filtra, a następnie dokonać obliczeń, generując blok próbek wyjściowych. Umożliwia to stosowanie najoptimalniejszych algorytmów pracujących na blokach danych, takich jak FFT.



RYS. 6 Krzywe błędu algorytmów adaptacyjnych w dziedzinie transformacji

Cała tajemnica wydajności blokowych algorytmów adaptacyjnych zawarta jest w realizacji szybkiego splotu dwóch sygnałów z wykorzystaniem FFT lub innych przekształceń. Oznaczając przez $u_{B,i}$ blok próbek wejściowych o długości B , przy czym kolekcja ostatniej próbki bloku miała miejsce w chwili czasu i , mamy:

$$(2.19) \quad u_{B,i} = [u(i), u(i-1), \dots, u(i-B+1)].$$

Aby dokonać splotu sygnału wejściowego z wektorem N współczynników filtra:

$$(2.20) \quad w = [w_0, w_1, \dots, w_{N-1}]^T,$$

rozdzielamy zebrane próbki oraz współczynniki filtra następująco:

$$(2.21) \quad U_{\frac{N}{B},i} = \begin{bmatrix} u_{B,i}^T & u_{B,i-B}^T & \dots & u_{B,i-\frac{N}{B}+1}^T \\ u_{B,i-B}^T & u_{B,i-2B}^T & \dots & u_{B,i-\frac{N}{B}}^T \end{bmatrix},$$

$$(2.22) \quad W = \begin{bmatrix} w_0 & w_B & \dots & w_{N-B} \\ w_1 & w_{B+1} & \dots & w_{N-B+1} \\ \vdots & \vdots & & \vdots \\ w_{B-1} & w_{2B-1} & \dots & w_{N-1} \\ 0_B & 0_B & \dots & 0_B \end{bmatrix},$$

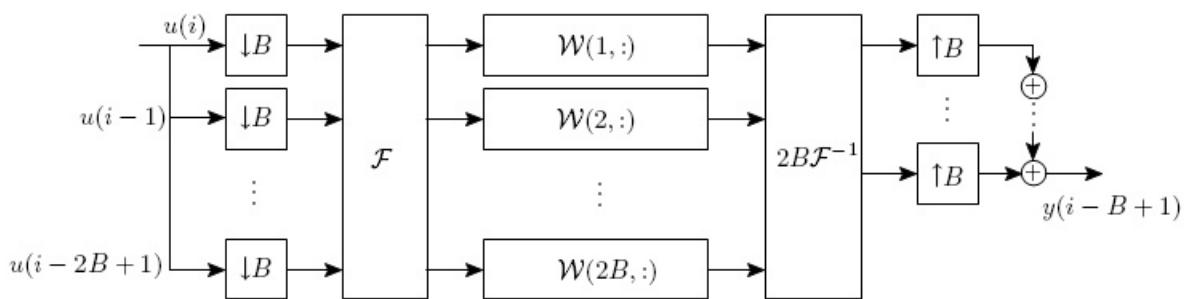
$$(2.23) \quad 0_B = \underbrace{[0, 0, \dots, 0]}_{\text{Brazy}}^{\text{T}}.$$

Następnie przechodzimy w dziedzinę częstotliwości, wykonując dyskretną transformację Fouriera poszczególnych kolumn macierzy źródłowych:

$$(2.24) \quad \mathcal{U}_{\frac{N}{B}, i} = \mathcal{F}\{U_{\frac{N}{B}, i}\}, \quad \mathcal{W} = \mathcal{F}^{-1}\{W\}.$$

Poszczególne wiersze macierzy \mathcal{W} można interpretować jako $2B$ filtrów pasmowych wykonujących operację filtracji w dziedzinie częstotliwości na przetransformowanym, zdecymowanym o czynnik B sygnale wejściowym. Próbki wynikowe oblicza się jako:

$$(2.25) \quad y_{B,i} = B \text{ pierwszych el. wektora } \left(\sum \text{ el. w wierszach macierzy } 2B\mathcal{W} \cdot * \mathcal{U}_{\frac{N}{B}, i} \right).$$



RYS. 7 Schemat blokowy szybkiego splotu w dziedzinie częstotliwości

B iloczynów skalarnych o długości N potrzebnych do wygenerowania B próbek wynikowych splotu zostało zamienionych na $2B$ zespolonych (w przypadku FFT) iloczynów skalarnych o długości $\frac{N}{B}$. Po wliczeniu do końcowego rozrachunku kosztów transformat bilans wypada zdecydowanie na korzyść filtracji blokowej. Wadą tego rozwiązania jest opóźnienie, z jakim pojawiają się przefiltrowane odpowiedzi. Wartość tego opóźnienia wynosi $B - 1$ próbek.

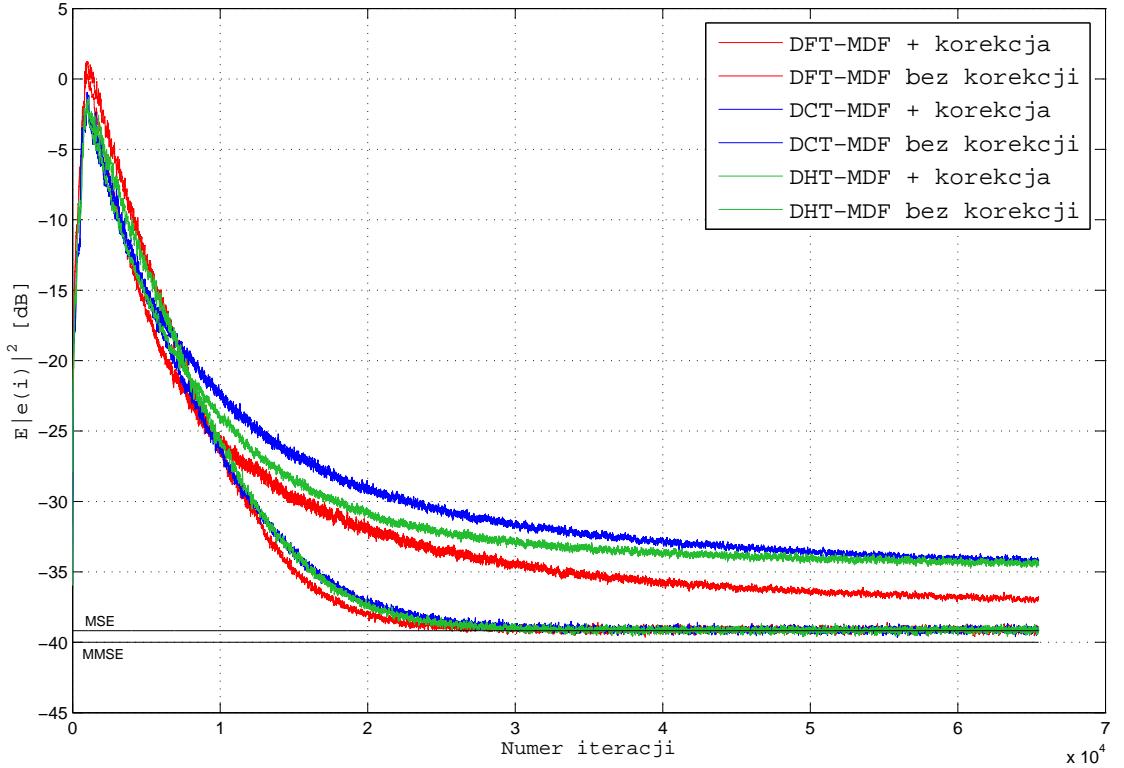
Mając do dyspozycji szybki splot w dziedzinie częstotliwości, pozostaje dodać równanie odświeżania poszczególnych filtrów ukrytych w wierszach macierzy \mathcal{W} . Aby tego dokonać, należy wyznaczyć błąd adaptacji: $\mathcal{E} = \mathcal{F}[d_{B,i} - y_{B,i}, 0_B^T]$, przybliżyć energię sygnału na wejściu poszczególnego z filtrów:

$$(2.26) \quad \Lambda = \beta\Lambda + (1 - \beta)|\mathcal{U}(:, 1)|^2.$$

Wtedy współczynniki filtra można odświeżyć dla $m = 1, \dots, 2B$:

$$(2.27) \quad \mathcal{W}(m, :) = \mathcal{W}(m, :) + \mu \frac{\mathcal{E}(m)}{\Lambda(m)} \mathcal{U}(m, :)^*. \quad \text{---}$$

Powyższy filtr nazywany jest w specjalistycznej literaturze „Multidelay Filter” i oznaczony symbolem MDF [46, 34]. Warto zwrócić uwagę na jeszcze jeden aspekt realizacji filtra MDF. Otóż w miarę odświeżania współczynników \mathcal{W} w dziedzinie częstotliwości przestaje być spełnione wymaganie, ażeby dolne B wierszy macierzy W w dziedzinie czasu było zerowych. Skutkuje to zmniejszeniem szybkości zbieżności filtru oraz osiągnięciem w stanie ustalonym nieoptymalnych wartości zaadaptowanych współczynników. Dlatego potrzebna jest korekcja, najlepiej po każdym przetworzonym bloku. Współczynniki \mathcal{W} są transformowane na powrót w dziedzinę czasu $\mathcal{W} \rightarrow W$, gdzie odpowiednie wartości są zerowane, by znów $W \rightarrow \mathcal{W}$. Dodaje to koszty dwóch transformat do ostatecznego rachunku, ale skutkuje prawidłowymi wartościami adaptowanych współczynników.



RYS. 8 Porównanie algorytmów MDF z korekcją i bez dla implementacji z wykorzystaniem różnych transformacji

Rysunek 8 przedstawia degradację właściwości algorytmu MDF, gdy współczynniki filtru nie są rewidowane. Dodatkowo zaimplementowano filtr MDF z wykorzystaniem dyskretnej transformacji cosinusowej DCT oraz dyskretnej transformacji Hartley'a DHT. Kroki algorytmów zostały tak dobrane, by osiągnąć tę samą rozbieżność od MMSE w stanie ustalonym.

W poniższej tabeli przedstawiono porównanie nakładów obliczeniowych stosowanych przez algorytm MDF dla kilku różnych długości bloku w porównaniu z implementacją próbka po próbce.

Algorytm	Liczba operacji na próbce	Liczba operacji: $N = 1024$			
		B=32	B=64	B=128	B=256
DFT-MDF + korekcja	$32\frac{N}{B} + 20\frac{N}{B}\log_2(2B) + 30\log_2(2B)$	5044	2962	1776	1118
DFT-MDF bez korekcji	$32\frac{N}{B} + 20\frac{N}{B}\log_2(2B)$	4864	2752	1536	848
ϵ -NLMS	$6N + 2$	6146			

TABELA 1. Porównanie nakładów obliczeniowych dla algorytmów MDF w zależności od długości bloku

Podsumowując, MDF jest jedną z lepszych blokowych implementacji filtrów adaptacyjnych. Po pierwsze: działając w dziedzinie szeroko pojętej częstotliwości, przyspieszamy zbieżność dla silnie skorelowanego sygnału wejściowego, z uwagi na właściwości dekorelujące różnych transformacji. Po drugie: możemy sterować złożonością obliczeniową i wnoszonym opóźnieniem poprzez odpowiedni dobór długości bloku. Po trzecie: możemy dobrać długość bloku w zależności od ilości posiadanej pamięci na wykonanie transformacji (dodatkowo transformacja mniejszej ilości próbek oznacza mniej błędów numerycznych). Po czwarте: istnieje sposób na całkowite pozbycie się wprowadzanego opóźnienia kosztem niewielkiego zwiększenia złożoności obliczeniowej, dodając operację bezpośredniej filtracji sygnału wejściowego za pomocą jednego z filtrów pasmowych [43].

3. Algorytmy adaptacyjne działające w oparciu o metodę najmniejszych kwadratów

Początki metody najmniejszych kwadratów datowane są na koniec XVIII wieku, a jej odkrycie przypisywane jest samemu Gaussowi. Dość powiedzieć, że sformułował ją w wieku osiemnastu lat, a pierwszy znaczący sukces odniósła 6 lat później, kiedy poproszono Gaussa o rozstrzygnięcie czy zaobserwowane na niebie ciało niebieskie jest planetą, czy kometą. Gauss, mając dostępne wyniki pomiarów trajektorii obiektu był w stanie przewidzieć położenie ciała niebieskiego w możliwie najlepszy sposób. Dziś po dwustu latach metoda ta w dalszym ciągu odnosi sukcesy jako jedna z najpotężniejszych narzędzi w teorii estymacji [43, 24, 19].

Załóżmy, że zaobserwowałyśmy M różnych wartości sygnału pożądanego filtru $\{d(0), d(1), \dots, d(M-1)\}$ oraz M dowolnych wartości stanu filtru $\{u_0, u_1, \dots, u_{M-1}\}$. Wtedy zakładając ergodyczność sygnału błędu, średnią po różnych realizacjach – jaka jest wymagana w przypadku obliczania błędu średniokwadratowego – można zastąpić średnią po próbkach czasowych:

$$(3.1) \quad E|d - uw|^2 = \frac{1}{N} \sum_{i=0}^{M-1} |d(i) - u_i w|^2.$$

Stąd kryterium najmniejszych kwadratów sprowadza się do znalezienia takich wartości współczynników filtru w , aby:

$$(3.2) \quad w : L_{\min}(w) = \sum_{i=0}^{M-1} |d(i) - u_i w|^2 \Big|_{\min} = \|d - Hw\|^2 \Big|_{\min}.$$

Kryterium optymalizacyjne najmniejszych kwadratów bierze pod uwagę wszystkie aktualne i przeszłe zmierzone próbki sygnałów wejściowych. Próbki M ostatnich pomiarów zostały zebrane do poniższych macierzy:

$$(3.3) \quad d_M = \begin{bmatrix} d(0) \\ d(1) \\ \vdots \\ d(M-1) \end{bmatrix}, \quad H_M = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{M-1} \end{bmatrix}.$$

Wykorzystanie właściwości geometrycznych wektorów prowadzi do wniosku, że wektor \hat{w} minimaлизujący kryterium najmniejszych kwadratów musi spełniać tzw. układ równań normalnych:

$$(3.4) \quad H^* H \hat{w} = H^* d.$$

Po wprowadzeniu do kryterium macierzy regulującej Π , umożliwiającej odwrócenie macierzy $H^* H$ w pewnych szczególnych przypadkach, wyznaczamy współczynniki \hat{w} będące rozwiązaniem przyjętego kryterium:

$$(3.5) \quad \hat{w} = (\Pi + H^* H)^{-1} H^* d.$$

Implementacja filtru adaptacyjnego po zastosowaniu powyższego wzoru jest bardzo prosta. Wystarczy zapamiętywać poszczególne próbki wejściowe filtru, by następnie wykonać operację mnożenia i odwracania macierzy. Niestety jest to rozwiązanie całkowicie niepraktyczne, bowiem wymagałoby mnożenia stale powiększających się w czasie macierzy próbek wejściowych. Dlatego też wymyślono sposób, by odświeżać wektor współczynników rekurencyjnie po skompletowaniu każdej nowej informacji.

Stosując kryterium najmniejszych kwadratów przed skompletowaniem M -tej danej i tuż po tym, optymalne współczynniki są dane wzorami:

$$(3.6) \quad \hat{w}_{M-1} = (\Pi + H_{M-1}^* H_{M-1})^{-1} H_{M-1}^* d_{M-1}, \quad \hat{w}_M = (\Pi + H_M^* H_M)^{-1} H_M^* d_M.$$

Macierz autokorelacji $R_M = \Pi + H_M^* H_M$ spełnia równanie rekurencyjne:

$$(3.7) \quad R_M = R_{M-1} + u_M^* u_M,$$

a jej odwrotność P_M po zastosowaniu twierdzenia Woodbury'ego o odwracaniu macierzy przyjmuje postać:

$$(3.8) \quad P_M = R_M^{-1} = P_{M-1} - \frac{P_{M-1} u_M^* u_M P_{M-1}}{1 + u_M P_{M-1} u_M^*}.$$

Po wstawieniu powyższego wzoru do równania na \hat{w}_M dochodzimy do powiązania optymalnych współczynników po zebraniu M danych wejściowych w zależności od współczynników w_{M-1} w poprzedniej chwili czasu oraz odwrotności macierzy autokorelacji P_{M-1} chwilę wcześniej:

$$(3.9) \quad \hat{w}_M = \hat{w}_{M-1} + \frac{P_{M-1} u_M^*}{1 + u_M P_{M-1} u_M^*} [d(M) - u_M \hat{w}_{M-1}].$$

Powyższy algorytm oznaczany jest symbolem RLS od nazwy „rekurencyjne najmniejsze kwadraty”.

Jak zostało pokazane, oryginalny algorytm RLS korzysta z wszystkich danych, jakie pojawiły się na wejściach filtra. W przypadku nagłych zmian w statystyce sygnałów wejściowych dawno zebrane próbki mogą zafalszowywać informację o prawdziwej naturze obecnie przetwarzanych danych. Stosuje się dwa rozwiązania tego problemu. Jednym z nich jest zastosowanie tzw. filtra RLS z „chodzącym oknem”, który wyznacza optymalne współczynniki, biorąc pod uwagę tylko i wyłącznie założoną ilość przeszłych próbek sygnału. Zagadnienu temu zostanie poświęcone więcej uwagi przy omawianiu zaimplementowanego algorytmu eliminacji echa akustycznego. Drugim rozwiązaniem jest wprowadzenie tzw. współczynnika zapominania α . Sprawia on, że im dana próbka jest starsza, tym mniejszy ma wpływ na wyznaczaną wartość współczynników \hat{w} . Jest to tzw. wykładniczo ważone kryterium najmniejszych kwadratów:

$$(3.10) \quad w : L_{\min}(w) = \sum_{i=0}^M \alpha^{M-i} |d(i) - u_i w|^2 \Big|_{\min} = (d - Hw)^* \Lambda (d - Hw) \Big|_{\min}.$$

Macierz Λ jest macierzą zawierającą na przekątnej wszystkie ważne wartości:

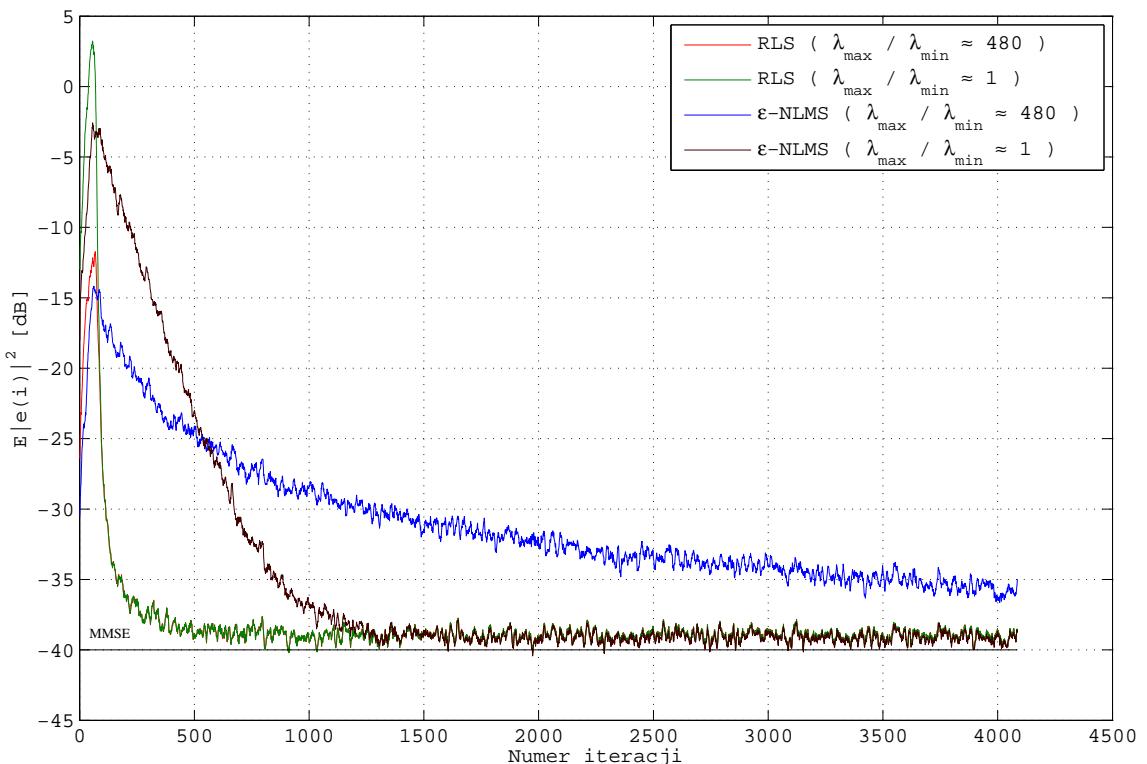
$$(3.11) \quad \Lambda = \text{diag}\{\alpha^M, \alpha^{M-1}, \dots, \alpha, 1\}.$$

Równanie odświeżania współczynników wykładniczo ważonego algorytmu RLS:

$$(3.12) \quad \hat{w}_M = \hat{w}_{M-1} + \frac{\alpha^{-1} P_{M-1} u_M^*}{1 + \alpha^{-1} u_M P_{M-1} u_M^*} [d(M) - u_M \hat{w}_{M-1}].$$

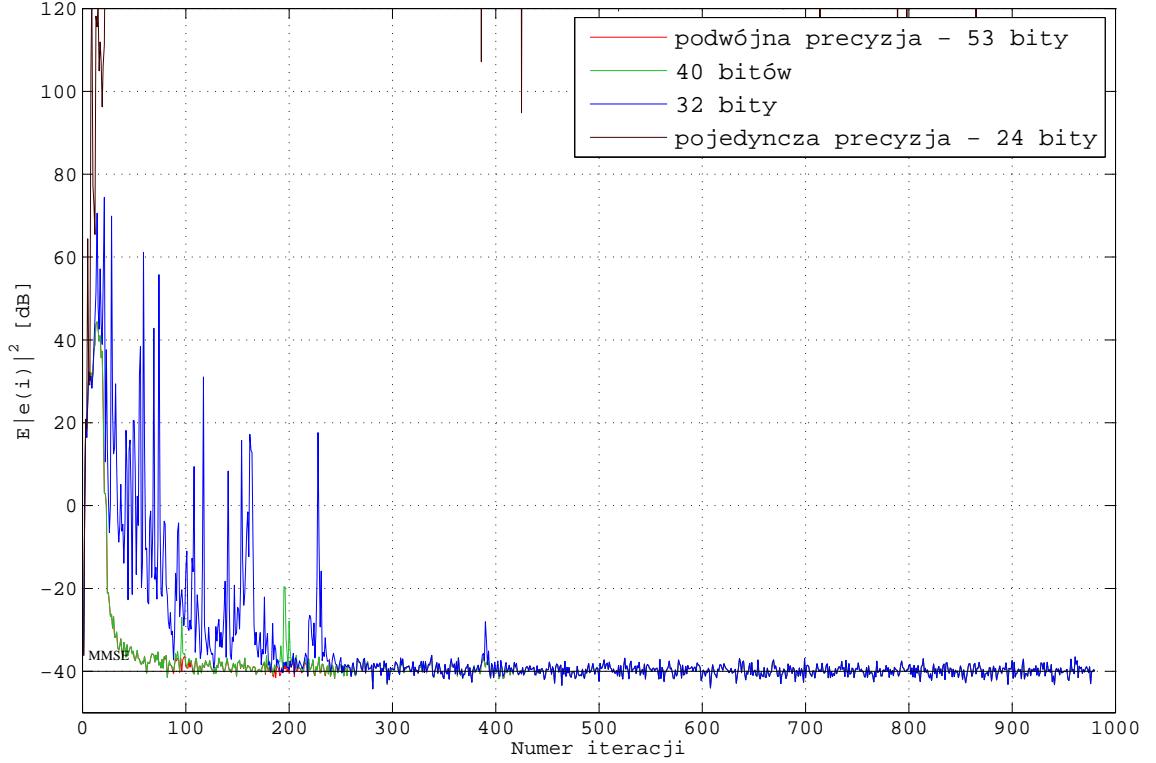
Na rysunku 9 przedstawiono porównanie algorytmów RLS i ϵ -NLMS dla „białego” sygnału wejściowego o jednostkowych wartościach własnych macierzy autokorelacji oraz „kolorowego” o rozrzucie równym $\eta = \frac{\lambda_{\max}}{\lambda_{\min}} \approx 480$. Pierwszym nasuwającym się wnioskiem z przeprowadzonej symulacji jest całkowita niezależność algorytmu RLS od charakterystyki widmowej sygnału wejściowego. Zbieżność następuje natychmiastowo po skompletowaniu próbki o numerze równym długości filtra.

Predystynuje to procedurę RLS do stosowania z wysoce skorelowanymi sygnałami. Niestety implementacje algorytmów RLS napotykają wielkie trudności. A mianowicie algorytmy RLS charakteryzują się bardzo dużą czułością na błędy numeryczne. Jak trudno jest uruchomić taki algorytm, pokazano na rys. 10.



RYS. 9 Porównanie algorytmów RLS i LMS dla sygnałów „białych” i „kolorowych”

Algorytm RLS został uruchomiony w czterech środowiskach obliczeniowych. W pierwszym z nich wszystkie rachunki zostały przeprowadzone we wbudowanej podwójnej precyzyji obliczeniowej programu MATLAB, charakteryzującej się przeznaczeniem 53 bitów na mantysę reprezentowanych liczb zmiennoprzecinkowych. W trzech pozostałych implementacjach mantysa miała odpowiednio: 40 bitów, 32 bity oraz 24 bity. Najlepsze właściwości wykazuje oczywiście implementacja w podwójnej precyzyji. Niestety zdecydowana mniejszość procesorów sygnałowych dostępnych na rynku posiada sprzętowe arytmometry o długości 64 bitów. Króluje standard pojedynczej precyzyji IEEE 754. Oczywiście można podwójną precyzyję symulować programowo, lecz wiąże się to ze znacznym spowolnieniem obliczeń. Poza tym nawet podczas obliczeń w podwójnej precyzyji zdarza się rozbieżność algorytmu RLS. Istnieją dwie główne przyczyny niestabilności. Obie wiążą się ze strukturą odwrotności macierzy autokorelacji P_M . Podczas poprawnej pracy macierz ta jest macierzą symetryczną, dodatnio określona. Kumulacja błędów numerycznych podczas sukcesywnych obliczeń sprawia, że macierz ta traci którąś ze wspomnianych właściwości. Stąd potrzeba bardziej niezawodnych implementacji.



RYS. 10 Algorytm RLS w środowisku obliczeniowym skończonej precyzyji

Implementacja algorytmu RLS w formie macierzowej, zamiast w formie układu równań, okazała się znacznie odporniejsza na efekty skończonej długości rejestrów przechowujących dane liczbowe. Założmy, że istnieje potrzeba wyznaczenia macierzy $\{C, F\}$ mając dane macierze $\{A, B, D, E\}$ spełniające poniższy zestaw równań:

$$(3.13) \quad \begin{aligned} CC^* &= AA^* + BB^* \\ FC^* &= DA^* + EB^*, \end{aligned}$$

przy czym C jest macierzą dolnotrójkątną, tzw. mnożnikiem Cholesky'ego pewnej innej dodatnio określonej macierzy P . Mnożnik Cholesky'ego bywa też czasami nazywany pierwiastkiem z macierzy, gdyż jest to jedyna taka macierz dolnotrójkątna o dodatnich elementach na przekątnej, posiadająca następującą właściwość:

$$(3.14) \quad CC^* = P^{\frac{1}{2}} P^* = P.$$

Zamiast bezpośredniego wyznaczenia szukanych wartości, można je znaleźć w następujący sposób: zebrać dostępne dane w jedną macierz blokową:

$$(3.15) \quad \mathcal{A} = \begin{bmatrix} A & B \\ D & E \end{bmatrix},$$

po czym znaleźć taką transformację unitarną Θ , która sprowadzi ją do postaci dolnotrójkątnej:

$$(3.16) \quad \begin{bmatrix} A & B \\ D & E \end{bmatrix} \Theta = \begin{bmatrix} X & 0 \\ Y & Z \end{bmatrix}.$$

Wtedy szukane C będzie równe X , a szukane F równe Y , które to wartości można odczytać z macierzy wynikowej. Definiując dwa nowe parametry algorytmu RLS:

$$(3.17) \quad \gamma(M) = \frac{1}{1 + \lambda^{-1} u_M P_{M-1} u_M^*}, \quad g_M = \lambda^{-1} \gamma(M) P_{M-1} u_M^*,$$

oraz używając poniższych podstawiń:

$$(3.18) \quad \begin{aligned} C &\leftarrow \gamma(M)^{-\frac{1}{2}} & A &\leftarrow 1 & B &\leftarrow \lambda^{-\frac{1}{2}} u_M P_{M-1}^{\frac{1}{2}}, \\ F &\leftarrow g_M \gamma^{-\frac{1}{2}} & D &\leftarrow 0 & E &\leftarrow \lambda^{-\frac{1}{2}} P_{M-1}^{\frac{1}{2}}, \end{aligned}$$

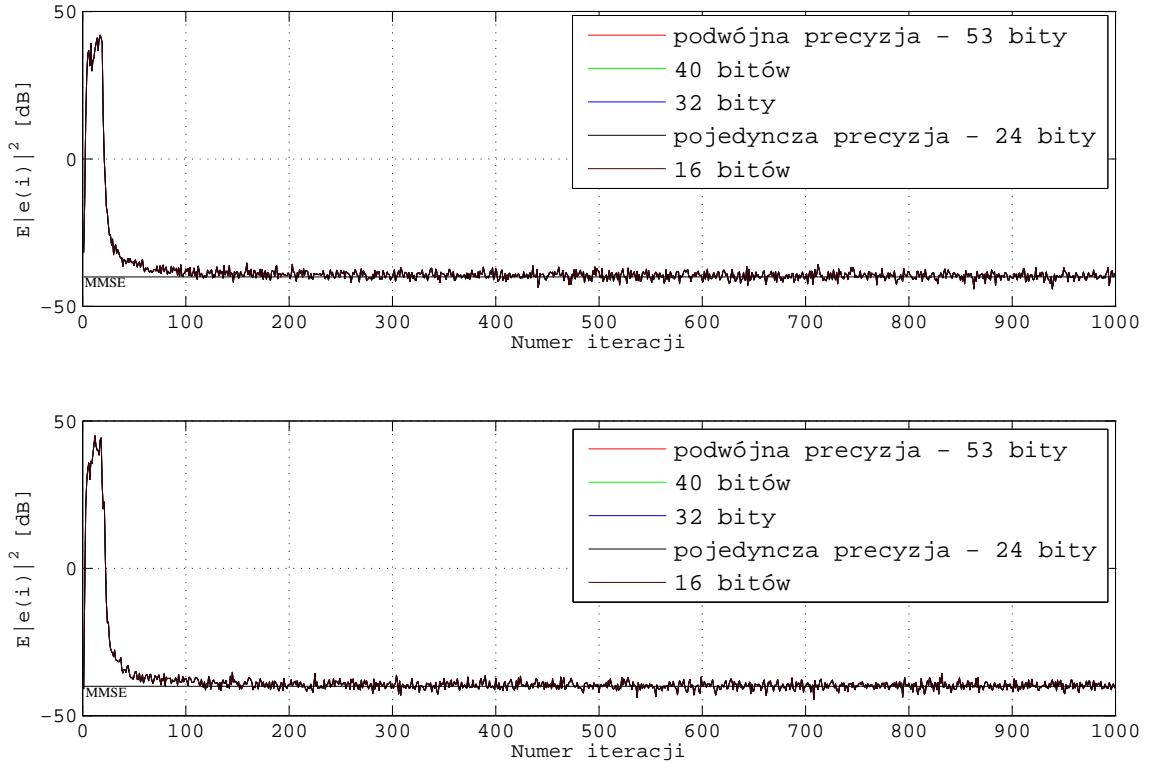
otrzymujemy macierzowy algorytm RLS, tzw. odwrotny QRRLS:

$$(3.19) \quad \begin{bmatrix} 1 & \lambda^{-\frac{1}{2}} u_M P_{M-1}^{\frac{1}{2}} \\ 0 & \lambda^{-\frac{1}{2}} P_{M-1}^{\frac{1}{2}} \end{bmatrix} \Theta_M = \begin{bmatrix} \gamma(M)^{-\frac{1}{2}} & 0 \\ g_M \gamma(M)^{-\frac{1}{2}} & P_M^{\frac{1}{2}} \end{bmatrix},$$

$$(3.20) \quad \hat{w}_M = \hat{w}_{M-1} + \left[g_M \gamma(M)^{-\frac{1}{2}} \right] \left[g_M \gamma(M)^{-\frac{1}{2}} \right]^{-1} [d(M) - u_M \hat{w}_{M-1}].$$

Wszystkie dane potrzebne do odświeżenia współczynników filtru, jak też do sformułowania macierzy wstępnej, potrzebnej do następnej iteracji, pobiera się z macierzy wynikowej. Transformację unitarną można zaimplementować jako serię pojedynczych obrotów, tzw. rotacji Givens'a zerujących jeden element macierzy bądź odbić Householdera, za pomocą których można niwelować całe wiersze [22]. QR w nazwie pochodzi stąd, że równie dobrze transformację do postaci dolnotrójkątnej można przeprowadzić w oparciu o rozkład QR macierzy. Skąd się bierze przewaga macierzowej implementacji nad standardową? Po pierwsze same transformacje unitarne mają bardzo dobre właściwości numeryczne. Po drugie – jak już zostało wspomniane – niestabilność RLS ma swoje główne źródło w złamaniu symetrii i dodatniej określoności odwrotności macierzy autokorelacji P_M . W początkowych implementacjach RLS transmitowano tylko jedną połowę tej macierzy z iteracji do iteracji, co znacznie polepszało zachowanie algorytmu. W algorytmie QRRLS transmitowany jest pierwiastek tej macierzy $P_M^{1/2}$ co z definicji jest już macierzą połówkową. W końcu oparcie się na pierwiastku z macierzy skutkuje znacznie węższą dynamiką wyznaczanych liczb. Rysunek 11 przedstawia ten sam eksperyment, jaki miał miejsce z destabilizującym się algorytmem RLS przy implementacji w formie macierzowej.

Różnica pomiędzy odwrotnym QRRLS a prostym QRRLS polega na tym, że w tym drugim transmitowana jest właściwa macierz autokorelacji $\Phi_M = P_M^{-1}$, a nie jej odwrotność. Obie implementacje są powszechnie uznawane za najlepsze, najbardziej odporne na błędy implementacje algorytmu najmniejszych kwadratów.



[a] [b] RYS. 11 Macierzowa implementacja algorytmu RLS; (a) odwrotny QRRLS; (b) QRRLS

Algorytm RLS pracujący czy w formie macierzowej, czy bezpośrednio w dalszym ciągu zachowuje poważną wadę. Mianowicie dość dużą złożoność obliczeniową, która jest rzędu N^2 . Ogranicza to zastosowanie tego algorytmu tam, gdzie potrzeba dużej ilości współczynników filtru. To właśnie było przyczyną, że pomimo opracowania podstaw teoretycznych algorytm ten przez długi okres czasu nie był stosowany w czasie rzeczywistym. Jednak, jeśli się dobrze przyjrzyć, to wykazuje on pewną nadmiarowość. Otóż przetwarzane przez niego dane wejściowe gromadzone w regresorach u_M mogą przyjmować dowolne wartości. Specyfiką zaś filtru transwersalnego jest to, że dane te zachowują strukturę linii opóźniającej. W kolejnych okresach próbkowania z regresora usuwana jest najstarsza informacja, by ustąpić miejsce nowej. $N - 1$ próbek sygnału wejściowego pozostaje niezmienionych. Uwydatnienie tego aspektu filtru adaptacyjnego prowadzi do znacznych oszczędności. Można zaimplementować algorytm RLS ze złożonością rzędu N , zbliżając się do e-NLMS! A dokonuje się tego za pomocą tzw. szybkich algorytmów RLS [1, 21, 43, 24].

Dla filtru transwersalnego zachodzi poniższa zależność między kolejnymi regresorami w chwilach czasu $i - 1$ oraz i :

$$(3.21) \quad [u(i), u_{i-1}] = [u_i, u(i - N)].$$

Rozpisując równania na przeskalowany przez $\gamma(i)^{-1}$ wektor wzmacnienia algorytmu RLS dla dwóch kolejnych chwil czasu, otrzymujemy:

$$(3.22) \quad \begin{aligned} g_i \gamma(i)^{-1} &= \lambda^{-1} P_{i-1} u_i^*, \\ g_{i-1} \gamma(i-1)^{-1} &= \lambda^{-1} P_{i-2} u_{i-1}^*. \end{aligned}$$

Uzależniając wartość wzmacnienia od poprzedniej chwili czasowej, można przekształcić równania

do postaci:

$$(3.23) \quad \begin{bmatrix} g_i \gamma(i)^{-1} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ g_{i-1} \gamma(i-1)^{-1} \end{bmatrix} + \lambda^{-1} \delta P_{i-1} \begin{bmatrix} u^*(i) \\ u_{i-1}^* \end{bmatrix};$$

δP_{i-1} jest macierzą różnicową zawierającą $N+1$ wierszy i kolumn:

$$(3.24) \quad \delta P_{i-1} = \begin{bmatrix} P_{i-1} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & P_{i-2} \end{bmatrix}.$$

Czyniąc podobny zabieg z równaniami na współczynnik konwersji:

$$(3.25) \quad \begin{aligned} \gamma(i)^{-1} &= 1 + \lambda^{-1} u_i P_{i-1} u_i^*, \\ \gamma(i-1)^{-1} &= 1 + \lambda^{-1} u_{i-1} P_{i-2} u_{i-1}^*, \end{aligned}$$

wyznaczamy zależność rekurencyjną:

$$(3.26) \quad \gamma(i)^{-1} = \gamma(i-1)^{-1} + \lambda^{-1} [u(i), u_{i-1}] \delta P_{i-1} \begin{bmatrix} u_i^* \\ u_{i-1}^* \end{bmatrix}.$$

Jasne już jest, że aby odświeżyć wektor wzmacnienia g_i a tym samym współczynniki filtru, nie trzeba znać dokładnych wartości odwrotności macierzy autokorelacji P_i , a jedynie różnicę dwóch kolejnych z nich. Co więcej, okazuje się, że przy odpowiednim doborze warunków początkowych wspomniana macierz różnicowa zachowuje stale rząd drugi. Jej kolumny są liniowymi kombinacjami dwóch wektorów o specjalnym znaczeniu. Te dwa wektory to wektory współczynników liniowej predykcji w przód $w_{N,i}^f$ oraz w tył $w_{N,i}^b$. Filtr predykcyjny w przód lub w tył to nic innego, jak zwyczajny filtr adaptacyjny, w omawianym przypadku oparty o metodę najmniejszych kwadratów, którego zadaniem jest wyznaczenie najlepszych estymat wszystkich próbek $u(i+1)$ na podstawie N przeszłych wartości $[u(i), u(i-1), \dots, u(i-N+1)]$ bądź najlepszych estymat wszystkich $u(i-N)$ na podstawie przeszłych $[u(i), u(i-1), \dots, u(i-N+1)]$. Dokładniej mówiąc filtr w przód dobiera swoje współczynniki w_N^f w taki sposób, aby suma wszystkich błędów wynikających z porównania wartości sygnału wejściowego w dowolnej chwili czasu z próbką powstała przez filtrację N przeszłych wartości przez owe dobierane współczynniki była minimalna:

$$(3.27) \quad w_N^f : L_{\min}^f(w_N^f) = \sum_{j=0}^i \alpha^{i-j} |u(j) - u_{N,j-1} w_N^f|^2 \Big|_{\min}.$$

Podobnie adaptowane są współczynniki predykcji w tył:

$$(3.28) \quad w_N^b : L_{\min}^b(w_N^b) = \sum_{j=0}^i \alpha^{i-j} |u(j-N) - u_{N,j} w_N^b|^2 \Big|_{\min}.$$

Skoro macierz różnicowa składa się z liniowych kombinacji wektorów $w_{N,i}^f$ oraz $w_{N,i}^b$, to do poprawnej pracy algorytmu wystarczy, by tylko one były propagowane zamiast całej macierzy P_i . Na powyższym wniosku opierają się właśnie szybkie algorytmy RLS.

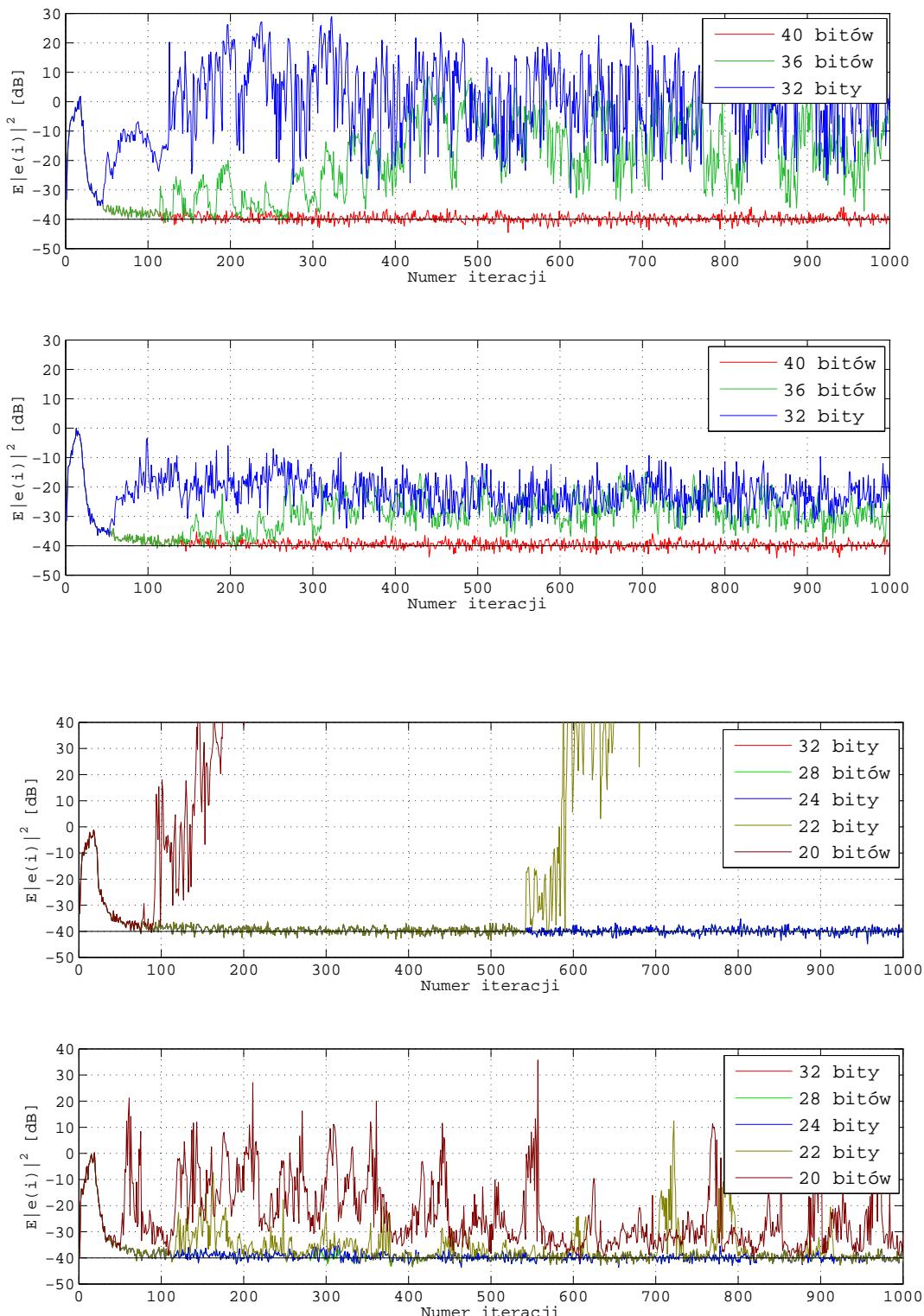
Propagacja tych współczynników odbywa się w różny sposób zależnie od odmiany szybkiego algorytmu. Istnieje forma macierzowa – tzw. szybki macierzowy algorytm RLS (FARLS) – oraz trzy implementacje w postaci zbioru równań (od historycznie najstarszych): „Fast Kalman filter” (FKF), „Fast transversal filters” (FTF) oraz „Fast a posteriori error sequential technique” (FAEST). Algorytm macierzowy zostanie dokładnie zaprezentowany przy opisie implementacji algorytmu eliminacji echa akustycznego. FKF jest właściwie definicyjną implementacją, wszystkie współczynniki

są odświeżane z wykorzystaniem tego samego wektora wzmacnienia $g_{N,i}$. Dwa dalsze algorytmy są oszczędniejsze w wykorzystaniu taktów zegarowych, jednak mają bardziej złożoną postać. Współczynniki filtrów są odświeżane za pomocą znormalizowanego wektora wzmacnienia $g_{N,i}\gamma_N(i)^{-1}$. Nazwa „Fast transversal filters” pochodzi od tego, podobnie zresztą jak w pozostałych odmianach, że w jednym algorytmie są ukryte cztery niezależne filtry transwersalne współpracujące ze sobą. W poniższej tabeli porównano nakłady obliczeniowe czterech odmian szybkiego algorytmu RLS. Najwydajniejsze FTF i FAEST wciąż ulegają pod tym względem standardowemu ϵ -NLMS, ale już prześcigają ϵ -APA-2 o złożoności $16N$, znacznie szybciej osiągając zbieżność od wspomnianej dwójki.

Algorytm	Liczba operacji na próbce
FARLS	$22N+41$
FTF	$14N+21$
FAEST	$14N+20$
FKF	$17N+6$

TABELA 2. Nakład obliczeniowy szybkich algorytmów RLS

Standardowy algorytm RLS wykazywał pewne problemy przy realizacji z wykorzystaniem arytmetyki o skończonej precyzyji. Owe niedogodności dało się jednak wyeliminować, tworząc algorytmy macierzowe. Z szybkimi realizacjami sprawa niestety jest gorsza. Algorytmy te zapisane są w postaci kilkunastu wzajemnie powiązanych równań mających jeszcze szybszą tendencję do załamywania się w prostych (jeśli tak można nazwać 32-bitowe zmiennoprzecinkowe jednostki obliczeniowe) arytmometrach. Na przestrzeni lat zaproponowano wiele idei próbujących wyeliminować tę pięć achillesową szybkich procedur. Pierwszą z nich było wprowadzenie tzw. mechanizmu ratującego. Pomysł polega na obserwacji pewnej zmiennej, która z zasady działania algorytmu zawsze powinna mieć wartość dodatnią. W momencie gdy zmienna ratunkowa zmieni znak na przeciwny, algorytm jest restartowany z zachowaniem zaadaptowanych dotychczas współczynników. Innym rozwiązaniem było wprowadzenie pewnej nadmiarowości do równań. Niektóre z propagowanych parametrów mogą być wyznaczone na różne, mniej lub bardziej efektywne sposoby. Pomysł polegał na przypisaniu końcowej wartości tych parametrów pewnej kombinacji wyliczonych kilkoma sposobami zmiennych. Jeszcze innym rozwiązaniem było wprowadzenie tzw. „przecieków”. Polegało to na przemnożeniu tuż przed odświeżeniem współczynników filtrów transwersalnych przez pewną mniejszą, jednak bliską jedności liczbę. Prowadziło to do pewnej stabilizacji, lecz jako skutek ubocznego powodowało rozbieżności w zaadaptowanych współczynnikach i większy błąd wyliczony na wyjściu filtru. W końcu implementacja macierzowa pozwoliła znacznie ograniczyć problemy niestabilności, ale nie była już tak dobra jak macierzowe wersje oryginalnego RLS. A to z powodu wymaganych do zastosowania rotacji hiperbolicznych [42] o znacznie gorszych właściwościach numerycznych od rotacji cyrkularnych Givensa. Poniższe wykresy prezentują wszystkie cztery odmiany szybkich algorytmów najmniejszych kwadratów w różnych środowiskach liczbowych z zaimplementowanym mechanizmem ratującym. Wnioski są takie, że FTF i FAEST zdecydowanie ulega FKF, jeśli za kryterium przyjąć długość mantysy potrzebnej do prawidłowej pracy, a to tłumaczy się mniejszą ilością równań w FKF niż w pozostałych dwóch. Stosunkowo najlepsze właściwości ma algorytm macierzowy, co nie jest niespodzianką. Niestety zdarzają się sporadyczne „wybuchy” przy krótszych słowach liczbowych. Z zaprezentowanych symulacji można również ocenić przydatność mechanizmu ratującego jako niestety stosunkowo niską. Co prawda, zabezpiecza zmienne algorytmu przed zdążaniem ku nieskończoności, ale prawidłowe wartości współczynników i tym samym minimalna wartość osiągniętego błędu pozostaje w sferze pobożnych życzeń.

[a]
[b]
[c]
[d]

RYS. 12 Szybkie algorytmy RLS w środowisku o skończonej precyzyji; (a) Fast Transversal Filters
 (b) FAEST
 (c) Fast Array RLS (d) Fast Kalman Filter

4. Algorytmy redukcji szumów

Nie ma chyba dziedziny przetwarzania sygnałów, gdzie nie trzeba brać pod uwagę specjalnego, nadmiarowego czynnika wkradającego się do przetwarzanych danych – szumu. Obecny przy przetwarzaniu sygnałów mowy, rozpoznawaniu mowy, przetwarzaniu obrazów i sygnałów medycznych, jest źródłem błędów i niepoprawnie zinterpretowanych danych. Ma swój znaczny wkład w telekomunikacji, głównie w postaci przekłamanych transmitowanych symboli, jest głównym czynnikiem ograniczającym jakość i zastosowania systemów pomiarowych. Historycznie szum narodził się w pracach Einsteina i Smoluchowskiego, zmierzających do wyjaśnienia tzw. ruchów Browna – ciągłych, chaotycznych ruchów pyłków kwiatowych w zawiesinie wodnej pod wpływem atakujących ich ze wszystkich stron o wiele mniejszych cząsteczek H_2O [16]. Wkrótce potem powstała matematyczna teoria szumu, tzw. teoria procesów stochastycznych [37]. Występują różne odmiany szumu, które można dzielić ze względu na ich fizyczną naturę i źródło występowania, np.: szum akustyczny, szum elektromagnetyczny, szum termiczny czy szum cyfrowy, jak również ze względu na zawartość widmową – szum biały, szum kolorowy czy szum impulsowy. Wszechobecność szumu oraz głównie negatywny wpływ na budowane urządzenia techniczne spowodowały, że zjawisko to stało się jednym z głównych obiektów zainteresowań cyfrowego przetwarzania sygnałów, którego zadaniem jest odróżnienie i wydzielenie czynnika zakłócającego od sygnału pożądanego.

W niniejszej pracy główny nacisk położony jest na minimalizację niekorzystnych efektów generowanych przez szum akustyczny zakłócający sygnał mowy ludzkiej. Szum akustyczny może być wynikiem pracy poruszających się, wibrujących urządzeń mechanicznych, takich jak silniki, jak również efektem wszelkich innych czynników wytwarzających dodatkowy sygnał akustyczny: rozmowy ludzkie w tle głównego źródła informacji, wiatr, systemy wentylacyjne itp. Szum akustyczny jest najczęstszym towarzyszem człowieka w jego życiu codziennym. Najpowszechniej stosowanym modelem szumu jest tzw. gaussowski szum biały o płaskiej charakterystyce widmowej gęstości mocy, obejmującej wszystkie możliwe częstotliwości. Niestety do dnia dzisiejszego nikt nie zbudował źródła o nieskończonej energii, potrzebnej do wygenerowania takiego szumu. Praktyczniejszym modelem jest tzw. pasmowy szum biały o równie płaskiej charakterystyce widmowej, obejmującej zakres częstotliwości wyznaczany twierdzeniem Kotelnikowa-Shannona. Inne rodzaje szumu, o niepłaskiej charakterystyce częstotliwościowej, traktujemy jak szum biały po przejściu przez filtr o odpowiadającej charakterystyce. Na rysunku 1 przedstawiającym powstawanie echa akustycznego pojawia się również czynnik zewnętrzny $n(n)$ będący właśnie takim zamodelowanym szumem. Kolejnym założeniem, jakie trzeba przyjąć, które przeważnie jest spełnione, gdy mowa o szumie akustycznym, jest założenie o wolnozmiennym charakterze szumu, aby go można było traktować jako stacjonarny w wystarczająco długim przedziale czasu. Mając na uwadze wspomniane wyżej preliminaria, można – wykorzystując metody statystyczne – podjąć próbę redukcji czynnika losowego z przetwarzanych sygnałów.

Pewna anegdota mówi, że gdy człowiek zajmujący się cyfrowym przetwarzaniem sygnałów nie wie, co zrobić, oblicza szybką transformatę Fouriera. Tak też uczyniono w jednym z pierwszych podejść do odszumiania sygnału mowy. Idea była bardzo prosta i pozostała właściwie niezmieniona po dziś dzień. Pomysł polegał mianowicie na odjęciu od amplitudy widma zabrudzonego sygnału estymaty amplitudy szumu, stąd nazwa widmowego odejmowania amplitudy [26, 35, 9, 10]. Oczywiście obliczanie FFT z całego sygnału mowy jest zabiegem niespecjalnie przemyślanym, gdyż mowa to sygnał silnie niestacjonarny. Z tego powodu oblicza się tzw. krótkoterminową transformatę Fouriera [41]. Sygnał poddawany jest okienkowaniu z nakładaniem, po czym obliczane jest widmo każdego fragmentu. Przyjmując oznaczenie $s(n)$ jako sygnał czystej mowy oraz $n(n)$ za sygnał zakłócający, zaszumiony sygnał mowy dany jest wzorem:

$$(4.1) \quad x(n) = s(n) + n(n).$$

Wtedy widmo wybranej ramki można oznaczyć przez $X(e^{j\omega_k})$. Celem okienkowania jest zmniejszenie przecieku widma, jaki wystąpi po obliczeniu FFT. Trzeba pamiętać, że przykładowo okno

Hamminga posiada listki boczne na poziomie 30 dB niższym od standardowego okna prostokątnego, stąd wartości próbek widma w tych punktach będą po zokienkowaniu znacznie mniejsze. Długość ramki dobiera się tak, aby objęła stacjonarny wycinek sygnału mowy. Przeważnie jest to czas około 20 ms. Zakładając znajomość uśrednionej amplitudy widma mocy szumu $N(e^{j\omega_k})$, estymatę uwydatnionego sygnału mowy oblicza się z zależności:

$$(4.2) \quad \hat{S}(e^{j\omega_k}) = \left(|X(e^{j\omega_k})| - N(e^{j\omega_k}) \right) e^{j\theta_X(\omega_k)}.$$

Do rekonstrukcji przetworzonego sygnału została wykorzystana faza sygnału zaszumionego:

$$(4.3) \quad \theta_X(\omega_k) = \arg\{X(e^{j\omega_k})\}.$$

Można to uczynić, bo – jak zostało wykazane eksperymentalnie przez Wang i Lim [55] – przekłamania fazy nie mają wpływu na jakość odbieranego przez człowieka sygnału.

Problem pojawią się, gdy $|X(e^{j\omega_k})| < N(e^{j\omega_k})$. Wystarczy jednak zerować wszystkie wartości próbek widma, dla których przetworzona amplituda miała wartości ujemne w celu skutecznej eliminacji wykrytych niedogodności. W wyniku odjęcia średniej wartości amplitudy szumu od amplitudy zaszumionego sygnału jako efekt uboczny pojawią się tzw. szum „muzyczny”. A składają się na niego te wartości widma szumu, których amplituda przekraczała pierwotnie wyliczoną wartość średnią. Szum „muzyczny” odbierany jest przez człowieka jako losowo występujące, metaliczne dźwięki o dość denerwującym oddziaływaniu. Nierzadko zdarzało się, że przetworzony sygnał zawierający szum „muzyczny” był mniej tolerowany od sygnału oryginalnego. Boll zaproponował metodę znacznej redukcji „muzycznych” tonów, polegającą na porównywaniu odpowiadających sobie próbek częstotliwościowych sąsiadujących ramek [10, 51]. Wartości do rekonstrukcji są wybierane na podstawie poniższego algorytmu:

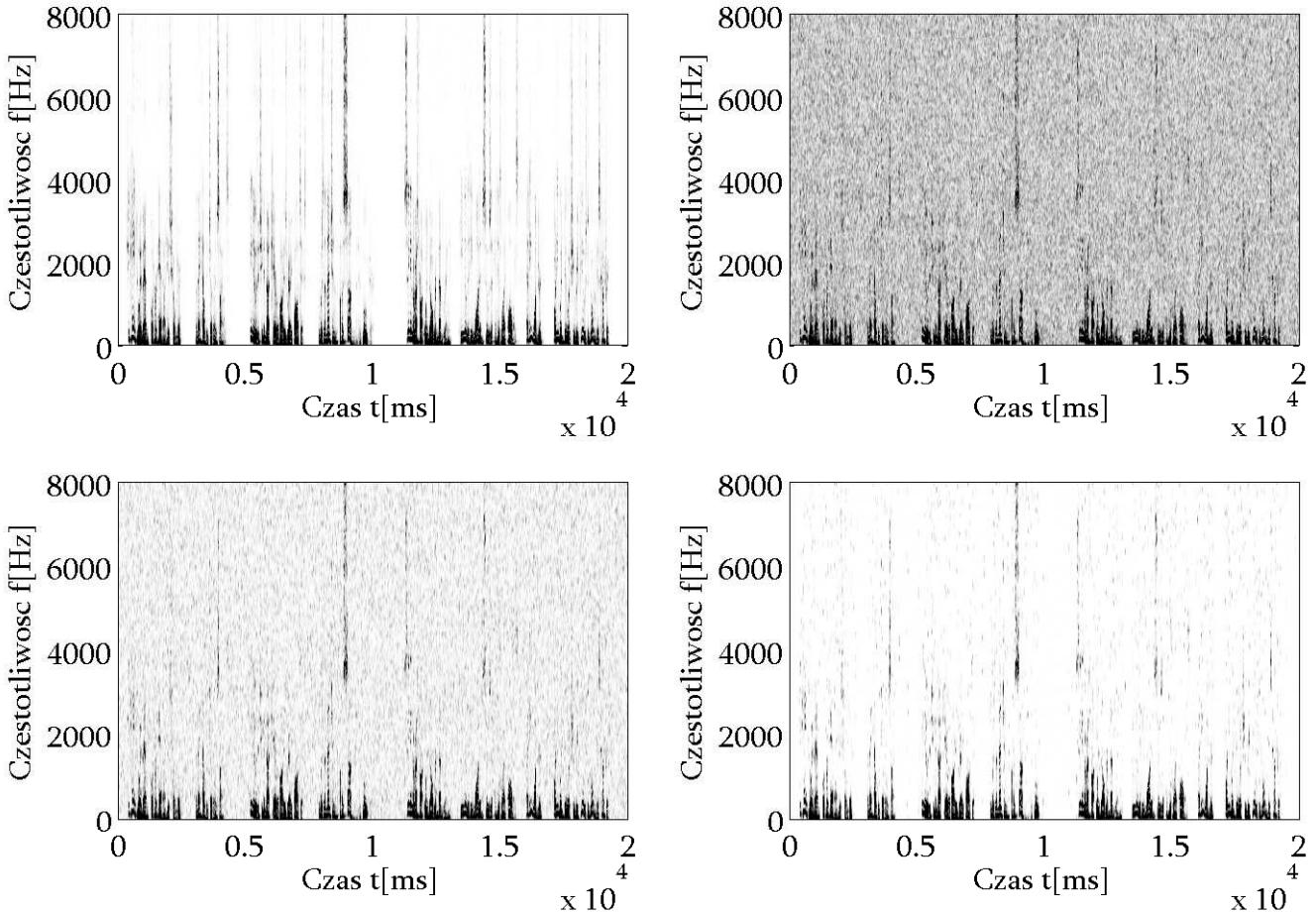
$$(4.4) \quad |\hat{S}_i(e^{j\omega_k})| < \max |N_{\text{res}}(e^{j\omega_k})| \Rightarrow |\hat{S}_i(e^{j\omega_k})| = \min |\hat{S}_j(e^{j\omega_k})|; \quad j = i-1, i, i+1.$$

$|N_{\text{res}}(e^{j\omega_k})|$ jest maksymalną wartością amplitudy szumu, jaki pozostał po odjęciu jego wartości średniej, zmierzona podczas braku sygnału mowy. Zasadę działania procedury można opisać w następujących słowach: gdy zawartość widmowa pewnego pasma częstotliwościowego zmienia się radicalnie na przestrzeni trzech sąsiadujących próbek, to istnieje duża szansa, że widmo dla tejże częstotliwości zależy właśnie od szumu. Przyjęcie minimalnej wartości z dobrym skutkiem go eliminuje. Gdy znów próbki są w miarę stałe, oznacza to, że mamy do czynienia z mową o małej mocy. Pozostawienie ramki bez zmian zachowa informację o sygnale mowy.

Oczywistą wadą tej metody jest zwiększenie wymaganej pamięci oraz trudności implementacyjne. Rysunek 13 przedstawia eksperiment z widmowym odejmowaniem amplitudowym dla 20-sekundowej sentencji z nałożonym, komputerowym białym szumem gaussowskim. Współczynnik nakładania ramek przyjęty został na 50%, długość ramki – 20 ms. Częstotliwość próbkowania – 16 kHz.

Kolejną implementacją algorytmu widmowego odejmowania, w której główny nacisk położono na eliminację szumu „muzycznego”, jest tzw. widmowe odejmowanie mocy sygnałów wraz z wprowadzeniem współczynnika przeestymowania mocy szumu [8]. Jeśli przyjąć za $X(\omega_k)$ amplitudę widma ramki zaszumionego sygnału, a za $N(\omega_k)$ uśrednioną wartość widma mocy szumu, to:

$$(4.5) \quad \hat{S}(\omega_k) = \begin{cases} |X(\omega_k)|^2 - \alpha N(\omega_k), & \text{gdy } \hat{S}(\omega_k) > \beta N(\omega_k); \\ \beta N(\omega_k), & \text{dla pozostałych.} \end{cases}$$



[a] [b] RYS. 13 Widmowe odejmowanie amplitudowe (a) sygnał czystej mowy (b) sygnał mowy zaszumionej (c) sygnał przetworzony z szumem „muzycznym” (d) sygnał po dodatkowej eliminacji szumu „muzycznego”

Wprowadzone zostały dwa zabiegi poprawiające jakość przetworzonego sygnału. Pierwszym z nich jest wprowadzenie współczynnika przeestymowania $\alpha \geq 1$, odjęcie kilkukrotnej wartości średniej szumu powoduje usunięcie większości pików szumu spowodowanych jego niezerową wariancją. Drugim zabiegiem jest wprowadzenie tzw. progu szumu $\beta \ll 1$. Nie pozostawia się zerowych wartości widma w miejscach, gdzie wynik odejmowania jest ujemny, lecz umieszcza się przeskakowaną wartość oryginalnego szumu z zadaniem maskowania pozostałoego szumu „muzycznego”. Sterowanie współczynnikiem β pozwala osiągnąć kompromis pomiędzy tłumieniem szumu szerskopasmowego a minimalizacją nieprzyjemnego efektu szumu „muzycznego”. Zastosowanie dużych wartości współczynnika przeestymowania α skutkuje całkowitym wyeliminowaniem omawianego fenomenu, lecz pogarsza jakość i zrozumiałość przetworzonej mowy. Dlatego uzależnia się wartość tego współczynnika od stosunku zakłóconego sygnału do szumu NSNR dla każdej z ramek:

$$(4.6) \quad \text{NSNR} = 10 \log_{10} \frac{\sum_{k=0}^{N-1} |X(\omega_k)|^2}{\sum_{k=0}^{N-1} N(\omega_k)} [\text{dB}]$$

podług odpowiednio dobranej funkcji $\alpha = f(\text{NSNR})$. Eksperymenty wykazały, że dobrym wyborem jest przyjęcie funkcji, która dla $\text{NSNR} < -5$ dB przypisuje $\alpha = 5$, a dla $\text{NSNR} > 20$ dB wartość $\alpha = 1$

oraz liniowo aproksymuje ten współczynnik w pozostałym przedziale, co prowadzi w efekcie do:

$$(4.7) \quad \alpha = 4.2 - 0.16\text{NSNR}, \quad -5 \text{ dB} \leq \text{NSNR} \leq 20 \text{ dB}.$$

Pod koniec zostanie przedstawione porównanie powyższej metody z innymi algorytmami redukcji szumów.

Naturalnym rozszerzeniem zaproponowanego algorytmu jest użycie innego współczynnika przeestymowania dla każdej próbki częstotliwościowej z osobna. Wymaga to obliczenia stosunku sygnał–szum dla poszczególnych pasm częstotliwościowych. Oznaczając przez $\mathcal{R}(p, \omega_k)$ chwilowy stosunek sygnał–szum dla ramki o numerze p i k -tej próbki częstotliwościowej, jest on dany wzorem:

$$(4.8) \quad \mathcal{R}(p, \omega_k) = \frac{|X(p, \omega_k)|^2}{N(\omega_k)} - 1,$$

gdzie $N(\omega_k)$ jest aktualną estymatą mocy szumu. Wygładzony stosunek sygnał–szum wyznacza się za pomocą średniej ruchomej zapisanej w postaci rekurencyjnej:

$$(4.9) \quad \zeta(p, \omega_k) = \beta \frac{|\hat{S}(p-1, \omega_k)|^2}{N(\omega_k)} + (1-\beta)P[\mathcal{R}(p, \omega_k)].$$

Funkcja $P(x)$ zwraca wartość swojego argumentu, gdy jest on większy od 0, w przeciwnym wypadku zwraca 0. Jeżeli wartość $\mathcal{R}(p, \omega_k)$ jest mała i waha się w okolicach 0 dB, wtedy $\zeta(p, \omega_k)$ jest wygładzoną wersją chwilowego stosunku sygnał–szum o znacznie mniejszej wariancji. Z drugiej strony, dla dużych $\mathcal{R}(p, \omega_k)$, $\zeta(p, \omega_k)$ śledzi jego wartości z opóźnieniem jednej ramki czasowej [13]. Współczynnik przeestymowania widma mocy α uzależnia się od uśrednionego SNR w taki sposób, aby dla bardzo małych wartości tego współczynnika, co ma miejsce w przypadku obecności tylko szumu na danej częstotliwości w sygnale przetwarzanym, odjąć najwięcej. Wygładzenie SNR dla małych wartości dodatkowo redukuje szum „muzyczny” zabezpieczając α przed nagłymi wahaniem. Z kolei dla dużych wartości uśrednionego SNR, gdy jest obecna mowa o dużej mocy, odejmowana jest mniejsza wartość estymaty szumu, by nie wprowadzić dodatkowych zniekształceń sygnału mowy. Współczynnik wygładzania β powinien przyjmować wartości kompromisowe pomiędzy redukcją szumu muzycznego, która jest największa, gdy $\beta \approx 1$, oraz szybką reakcją na nagłe zmiany sygnału.

Implementacją bezpośrednio wykorzystującą uśredniony stosunek sygnał–szum jest tzw. algorytm nieliniowego widmowego odejmowania [52, 38, 30], gdzie estymatę widma mocy oblicza się ze wzoru:

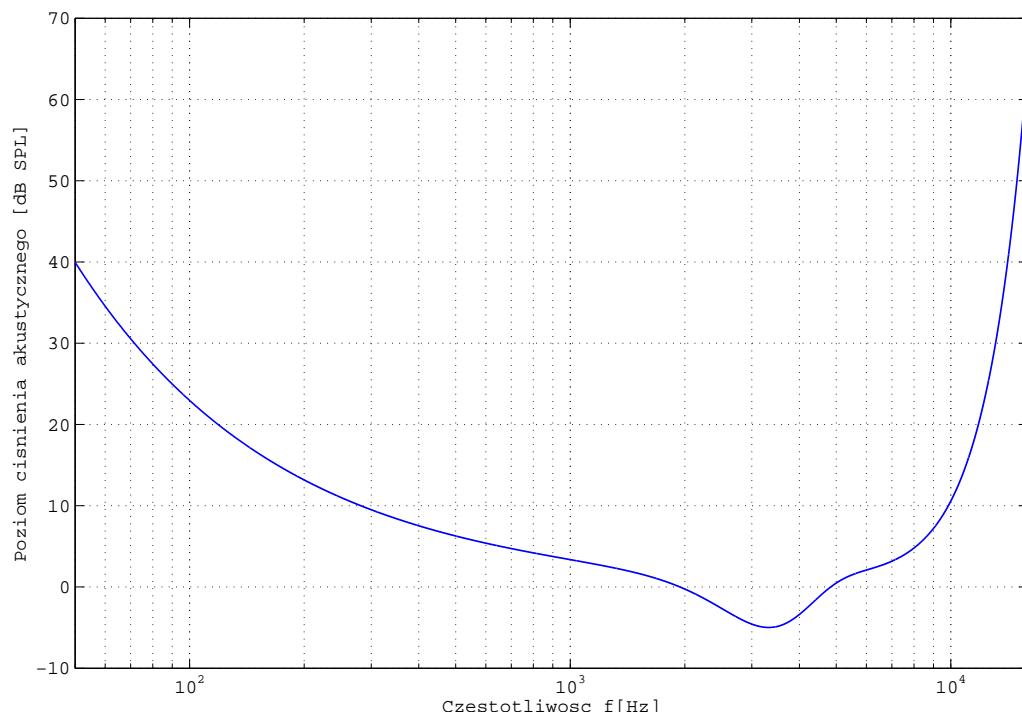
$$(4.10) \quad |\hat{S}(\omega_k)|^2 = |X(\omega_k)|^2 - \Theta(\zeta(\omega_k), N(\omega_k)).$$

Θ jest pewną nieliniową funkcją, której zadaniem jest dobrać współczynnik przeestymowania α w możliwie najlepszy sposób. Lockwood i Boudy [52] zaproponowali następującą nieliniową funkcję:

$$(4.11) \quad \Theta(\zeta(\omega_k), N(\omega_k)) = \frac{\max z M \text{ ramek}(N(\omega_k))}{1 + \gamma \cdot \zeta(\omega_k)}.$$

Współczynnik γ dobiera się eksperymentalnie, aby uzyskać najlepszy efekt. Dla dużego $\zeta(\omega_k)$ funkcja $\Theta \rightarrow 0$, z kolei dla małych wartości $\zeta(\omega_k)$ odejmowana jest maksymalna wartość ostatnio zaobserwowanego szumu.

Dotychczas zaprezentowane algorytmy nie brały pod uwagę właściwości aparatu słuchowego człowieka przy uwydatnianiu sygnału mowy. Podejście takie pojawiło się w literaturze i wykorzystuje model psychoakustyczny narządu słuchu człowieka, zbudowany na potrzeby kodowania sygnałów audio [36, 25]. Pierwszym ograniczeniem jest fakt, iż istota ludzka posiada zróżnicowaną czułość na sygnały akustyczne o różnych częstotliwościach. Poniższy wykres przedstawia krzywą absolutnego poziomu słuchu „uśrednionego” osobnika w zależności od częstotliwości. Wartość poziomu ciśnienia akustycznego oddziałującego na jego ucho, leżąca poniżej wyrysowanej krzywej przy pobudzeniu pojedynczym tonem w środowisku bezszumowym, nie jest rejestrowana przez świadomość.



RYS. 14 Absolutny poziom słuchu człowieka w zależności od częstotliwości

Badając teraz taką krzywą w obecności innego tonu sinusoidalnego, okazałoby się, że znacznie trudniej jest wykryć inny dźwięk w sąsiedztwie generowanego tonu. „Znacznie trudniej” oznacza, że musiałby on posiadać znacznie większą amplitudę niż w przypadku ciszy. Podsumowując, w obecności sygnału maskującego detekowane są tylko impulsy przekraczające jego poziom maskowania. Zjawisko to wykorzystuje się przy kodowaniu sygnałów audio, gdzie pomija się kodowanie impulsów, które z powodu obecności innych, silniejszych sygnałów i tak nie będą rejestrowane przez człowieka. W efekcie końcowym prowadzi to do lepszej kompresji sygnałów dźwiękowych. Przykładem, gdzie stosuje się powyższe zasady, jest np. algorytm kompresji MP3 rozpowszechniony dzięki Internetowi, stosowany do kompresji muzyki czy dźwięku obrazów filmowych.

Drugim aspektem modelu psychoakustycznego jest istnienie tzw. krytycznych pasm, które tłumaczy się opierając się na wzajemnym oddziaływaniu pojedynczego tonu dźwiękowego i szeroko-pasmowego szumu. Zakładając szum o płaskiej charakterystyce częstotliwościowej oraz szerokości pasma B i umieszczając detekowany ton na częstotliwości środkowej szumu, po zwiększeniu B również podnosiłby się poziom detekcji tonu sinusoidalnego. Po przekroczeniu jednak pewnej szerokości

pasma BW_c dalsze jego zwiększanie nie miałoby już wpływu na detekowanie pojedynczego tonu. Można zatem wysnuć wniosek, że wzajemne wpływanie na siebie sygnałów zachodzi w pewnym ograniczonym paśmie zwany właśnie pasmem krytycznym. Pasma te mają charakter krzywych Gaussa oraz są nierównomiernie rozłożone w dziedzinie częstotliwości. Poniższa tabela przedstawia rozkład wyidealizowanych krytycznych pasm rozciągających się na całą przestrzeń audio.

Nr pasma	Szerokość pasma	Nr pasma	Szerokość pasma	Nr pasma	Szerokość pasma
1	– 100	2	100 – 200	3	200 – 300
4	300 – 400	5	400 – 510	6	510 – 630
7	630 – 770	8	770 – 920	9	920 – 1080
10	1080 – 1270	11	1270 – 1480	12	1480 – 1720
13	1720 – 2000	14	2000 – 2320	15	2320 – 2700
16	2700 – 3150	17	3150 – 3700	18	3700 – 4400
19	4400 – 5300	20	5300 – 6400	21	6400 – 7700
22	7700 – 9500	23	9500 – 12000	24	12000 – 15500
25	15500 –				

TABELA 3. Rozkład wyidealizowanych pasm krytycznych rozpiętych w paśmie akustycznym

Proces wyznaczania poziomu maskowania w psychoakustycznym modelu Johnstona [25] przebiega wieloetapowo. Na początku wyznacza się energię w każdym krytycznym paśmie:

$$(4.12) \quad E_r = \sum_{\omega=kl_r}^{kh_r} |S(e^{j\omega_k})|^2,$$

gdzie $S(e^{j\omega_k})$ oznacza widmo maskera, a kl_r i kh_r to najmniejszy i największy numer próbki w r -tym paśmie. Następnie obliczone widmo energii jest splatane z tzw. funkcją rozpraszającą. Jest to funkcja postaci w przybliżeniu trójkątnej o zboczach o nachyleniu +25 oraz -10 dB na każde krytyczne pasmo. Wygodna postać analityczna funkcji rozpraszającej w zależności od numeru krytycznego pasma jest następująca:

$$(4.13) \quad SF_r = 15.81 + 7.5(r + 0.474) - 17.5\sqrt{1 + (r + 0.474)^2} \text{ [dB].}$$

Celem użycia funkcji rozpraszającej jest zasymulowanie maskowania, jakie zachodzi pomiędzy dwoma wyidealizowanymi pasmami.

$$(4.14) \quad C_r = B_r * SF_r.$$

Gdy maskerem jest dźwięk tonowy, poziom maskowania oblicza się ze wzoru:

$$(4.15) \quad TH_T = C_r - 14.5 - r \text{ [dB].}$$

Gdy maskerem jest dźwięk postaci szumowej, poziom maskowania jest przybliżany przez:

$$(4.16) \quad TH_N = C_r - 5.5 \text{ [dB].}$$

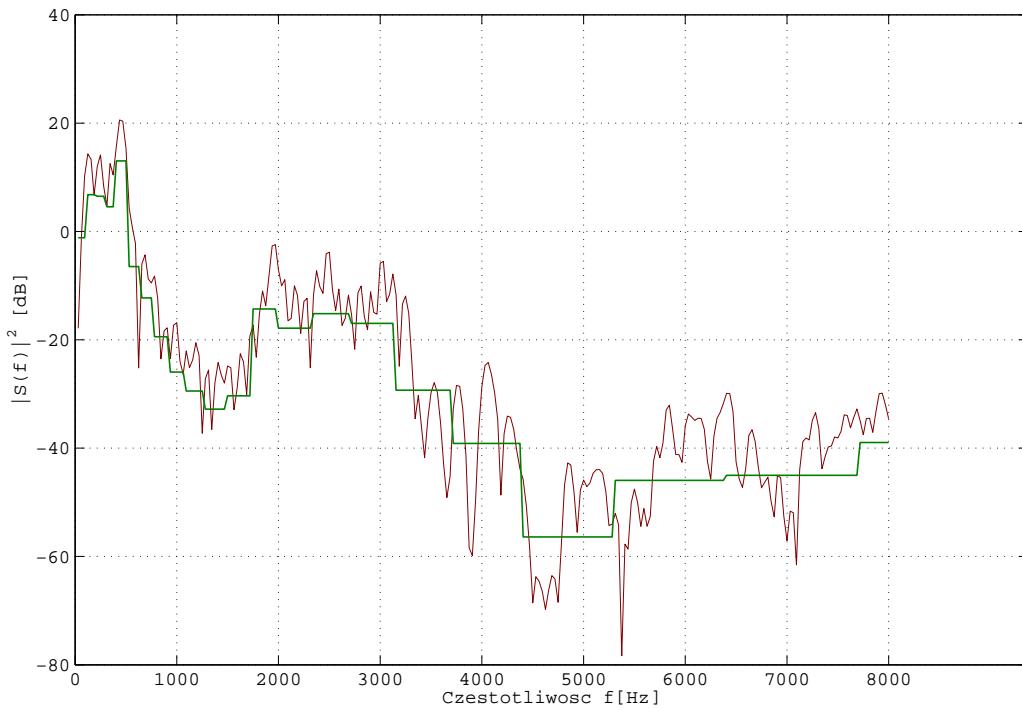
W celu rozstrzygnięcia natury sygnału ukrytego w rozproszonych pasmach używa się tzw. współczynnika płaskości widmowej $SFM = \frac{\mu_g}{\mu_a}$, gdzie licznik i mianownik oznaczają średnią geometryczną i arytmetyczną próbek widma w każdym paśmie odpowiednio. Z uwagi na fakt, że średnia geometryczna jest nie większa od średniej arytmetycznej tych samych liczb, gdy widmo sygnału będzie w miarę płaskie, współczynnik SFM będzie przyjmował wartości bliskie 1, przeciwnie do wartości bliskich 0 dla natury tonowej sygnału. Stąd wartość, o jaką trzeba pomniejszyć C_r w celu wyznaczenia poziomu maskowania, można zapisać jedną zależnością:

$$(4.17) \quad O_r = \alpha(14.5 + r) + (1 - \alpha)5.5 \text{ [dB];} \quad \alpha = \min\left(\frac{SFM[\text{dB}]}{-60}, 1\right).$$

Ostateczny poziom maskowania w r -tym paśmie krytycznym:

$$(4.18) \quad T_r = 10^{\log_{10}(C_r) - (0.1 \cdot O_r)}.$$

Końcowym etapem jest przeskalowanie przez $\frac{C_r}{B_r(kh_r - kl_r)}$ w celu symulacji rozplotu z funkcją rozpraszającą oraz sprawdzenie, czy wyznaczony poziom nie leży przypadkiem poniżej absolutnego poziomu słuchu człowieka. Ponadto przed ostatnią operacją przeprowadza się jeszcze kalibrację w celu wyrównania najniższego punktu charakterystyki słuchu człowieka z energią sygnału odpowiadającą ± 1 bitowi amplitudy sygnału. Wyznaczony poziom maskowania dla przykładowej ramki sygnału mowy przedstawiony jest na poniższym rysunku:



RYS. 15 Widmo mocy przykładowej ramki sygnału mowy z wyznaczonym poziomem maskowania

Algorytm widmowego odejmowania z wykorzystaniem właściwości słuchu ludzkiego dobiera współczynniki przeestymowania α oraz progu szumu β w zależności od wyznaczonego poziomu ma-

skowania, który oblicza się z estymaty widma mocy sygnału z szumem „muzycznym” [53, 54, 50, 49]. Określa się minimalne i maksymalne wartości tych współczynników oraz za pomocą funkcji aproksymujących dobiera się je tak, by dla minimalnej wartości poziomu maskowania współczynniki te były największe i *vice versa*.

$$(4.19) \quad \begin{aligned} \alpha(\omega_k) &= F_\alpha [\alpha_{min}, \alpha_{max}, T(\omega_k)], \\ \beta(\omega_k) &= F_\beta [\beta_{min}, \beta_{max}, T(\omega_k)]. \end{aligned}$$

Najprostszą funkcją aproksymującą jest oczywiście funkcja liniowa. Tak skonstruowany algorytm ma pewną przewagę nad procedurami opartymi na estymacie stosunku sygnał-szum. Gdy SNR jest wyznaczany z zaszumionej ramki, estymata $T(\omega_k)$ jest dokładniejsza z racji wstępnego czyszczenia sygnału. Po drugie: $T(\omega_k)$ ma gładszą ewolucję od samego SNR. Po trzecie: $T(\omega_k)$ jest lepiej skorelowane z percepcją niż SNR.

Jedną z głównych wad algorytmów widmowego odejmowania jest fakt niewykorzystywania podczas przetwarzania zaszumionego sygnału jego właściwości statystycznych. Z drugiej strony charakteryzują się one prostotą, gdyż w praktyce cały algorytm sprowadza się do odjęcia od widma mocy sygnału estymaty szumu. Zupełnie inne podejście zastosowane jest w algorytmie wyznaczającym optymalne amplitudy współczynników rozwinięcia w zespółony szereg Fouriera sygnału mowy w analizowanej ramce w oparciu o kryterium minimalizacji błędu średniokwadratowego. Aby jednak tego dokonać, trzeba znać rozkłady prawdopodobieństwa współczynników rozwinięcia sygnału mowy i szumu. W tym celu przyjęto pewien model statystyczny. Mianowicie założono, że współczynniki rozkładu Fouriera można zamodelować za pomocą niezależnych gaussowskich zmiennych losowych o zerowej średniej i zmiennej w czasie wariancji z racji niestacjonarności sygnału mowy. Motywacją do przyjęcia takiego założenia jest centralne twierdzenie graniczne – poszczególne współczynniki rozwinięcia są sumą ważoną bardzo wielu zmiennych losowych – próbek sygnału. Końcowa postać estymatora dana jest następującą zależnością [18]:

$$(4.20) \quad \hat{S}(\omega_k) = \Gamma(1.5) \frac{\sqrt{v_k}}{\xi_o} e^{-\frac{v_k}{2}} \left[(1 + v_k) I_0 \left(\frac{v_k}{2} \right) + v_k I_1 \left(\frac{v_k}{2} \right) \right] X(\omega_k),$$

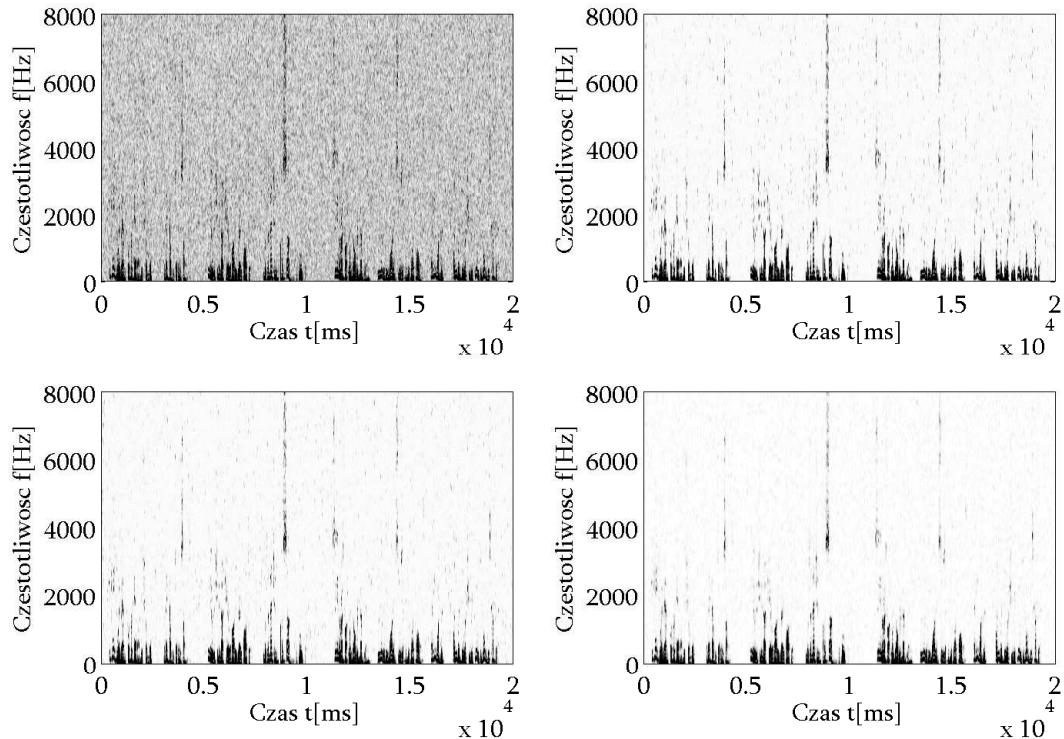
$$(4.21) \quad v_k = \frac{\xi_i}{1 + \xi_i} \xi_o,$$

ξ_i oraz ξ_o oznaczają stosunki sygnał-szum dla sygnału czystego i zaszumionego a $I_n(x)$ zmodyfikowaną funkcję Bessela I rodzaju, n -tego rzędu. Jak się później okazało [17], dokładniejsze wyniki, mniej zniekształceń przetwarzanego sygnału, dostarczył estymator logarytmu widma mocy:

$$(4.22) \quad \hat{S}(\omega_k) = \frac{\xi_i}{1 + \xi_i} \exp \left(\frac{1}{2} \int_{v_k}^{\infty} \frac{e^{-t}}{t} dt \right) X(\omega_k),$$

z wykorzystaniem całki wykładniczej. Powyższy algorytm oznaczany jest w literaturze skrótem LSA.

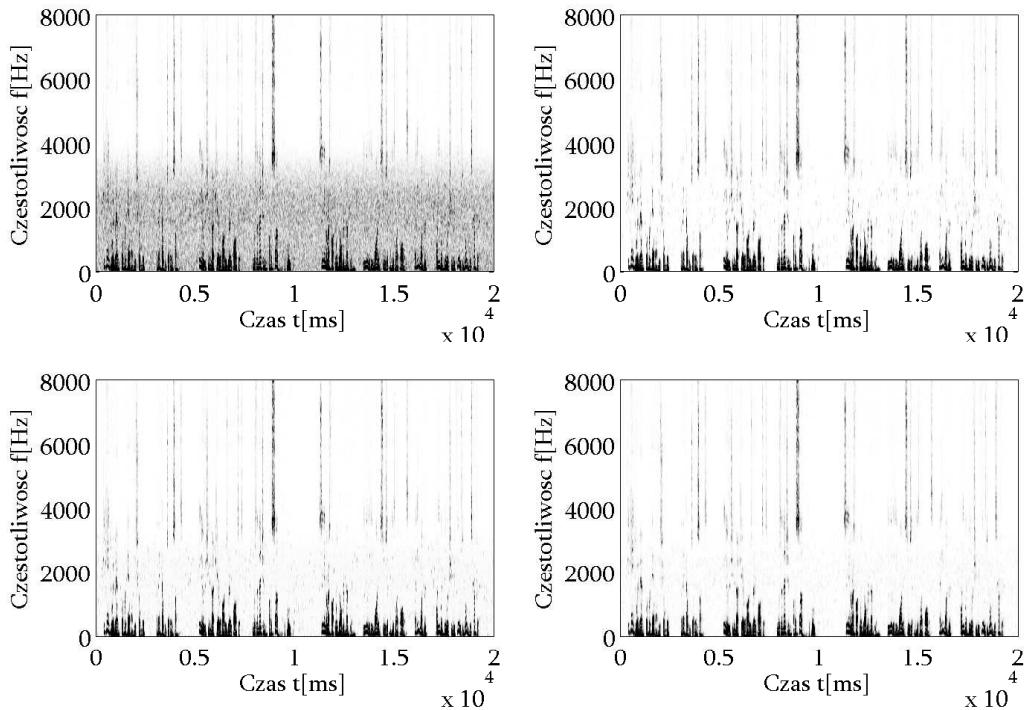
Dalej zostanie przedstawione porównanie właściwości zaproponowanych algorytmów przy odzumianiu 20-sekundowego sygnału mowy z nałożonym komputerowo szumem. W pierwszym doświadczeniu był to szum biały rozciągający się na całe pasmo częstotliwości. W drugim przypadku usuwano szum dolnoprzepustowy, by w końcu odfiltrować szum pasmowy. Na jednym z wykresów przedstawiono spektrogram zaszumionego sygnału dla każdego doświadczenia, by na kolejnych obrazkach przedstawić wynik działania trzech zaimplementowanych algorytmów: nieliniowego widmowego odejmowania, widmowego odejmowania z wykorzystaniem właściwości słuchu człowieka oraz estymatora logarytmu widma mocy. Parametry algorytmów dobrano tak, aby uzyskać możliwie najlepszy efekt.



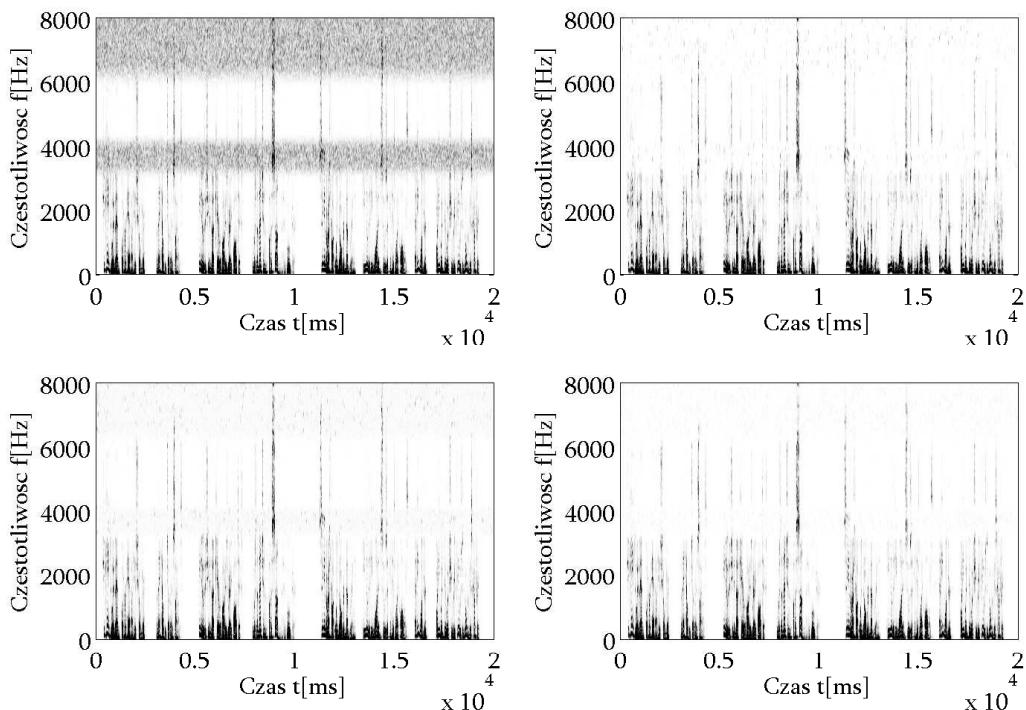
[a] [b] RYS. 16 Uwydatnianie sygnału mowy z szumem białym (a) sygnał zaszumiony (b) nieliniowe widmowe odejmowanie
 [c] [d] (c) widmowe odejmowanie z wykorzystaniem właściwości słuchu człowieka (d) algorytm LSA

Jak wynika z wykresów, stosunkowo najmniej szumu „muzycznego” objawia się w algorytmie LSA. Ponadto subiektywna ocena autora po przesłuchaniu przetworzonych próbek wskazuje na lepszą jakość sygnału mowy przetworzonego algorytmem z wykorzystaniem poziomów maskowania niż procedurą nieliniowego widmowego odejmowania. Autor zdaje sobie sprawę z niezerowej wariancji, jaką przejawia w stosunku do „uśrednionego” słuchacza wzmagana faktorem, iż podczas wielokrotnych odsłuchów tego samego materiału zaczyna się do niego przyzwyczajać, ponadto znając tekst wygłaszanej frazy na pamięć, trudno jest ocenić zrozumiałość mowy po przetworzeniu którymś z algorytmów. Dlatego też ocena działania algorytmów ze zmysłowego punktu widzenia może się różnić w zależności od osoby oceniającej.

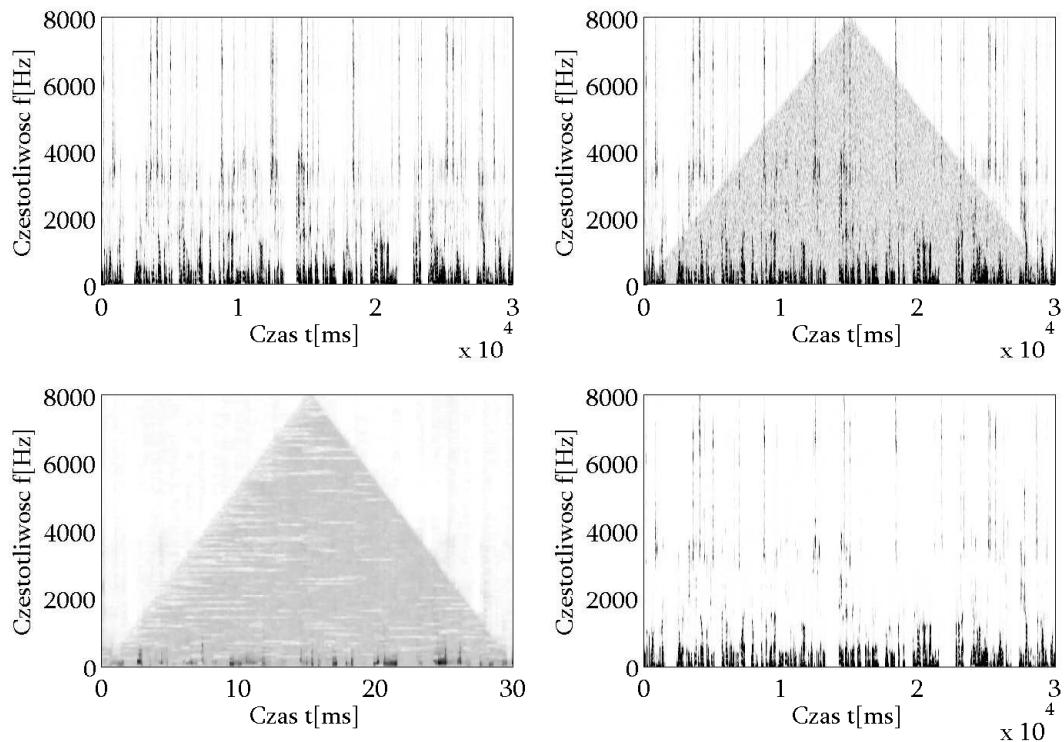
Kluczową sprawą przy implementacji algorytmów redukcji szumów zmiksowanych z sygnałem pożądany w jednym kanale jest poprawna estymata szumu w badanym przedziale czasu. Nawet najbardziej wyszukany algorytm nie zadziała poprawnie bez tej wstępnej informacji. Sygnał mowy ma to do siebie, że regularnie pojawiają się w nim przerwy z zerową zawartością informacji. Najprościej jest właśnie przeprowadzić uśrednianie szumu podczas takich pauz. Niestety wymaga to użycia niezawodnego detektora obecności sygnału mowy, którego zaimplementowanie jest rzeczą trudną. Inne podejście to badanie szumu bez względu na to, czy w danej chwili czasu mowa jest obecna, czy nie. Przykładowym rozwiązaniem jest algorytm szacowania szumu bazujący na metodzie minimów statystycznych [31]. Zasada działania jest prosta. Przeprowadzane jest uśrednianie kolejnych ramek sygnału oraz zapamiętywane są minimalne wartości takich periodogramów. Tak wyznaczona wartość jest znacznie mniejsza od poprawnej estymaty, stąd dodatkowo wprowadza się korekcję. Przykład działania tej metody przedstawiony jest na rys. 19. Tym razem 30-sekundowa próbka mowy zakłócona jest szumem niestacjonarnym, symulującym start samolotu [33]. Kolejny wykres przedstawia estymatę tego szumu. Widoczna jest drobna niedoskonałość – niewielka część widma mowy wnika do obliczonej estymaty. Na ostatnim wykresie zaprezentowany jest sygnał po redukcji z wykorzystaniem algorytmu LSA.



[a] [b] RYS. 17 Uwydatnianie sygnału mowy z szumem dolnoprzepustowym (a) sygnał zaszumiony (b) nieliniowe widmowe odejmowanie (c) widmowe odejmowanie z wykorzystaniem właściwości słuchu człowieka (d) algorytm LSA



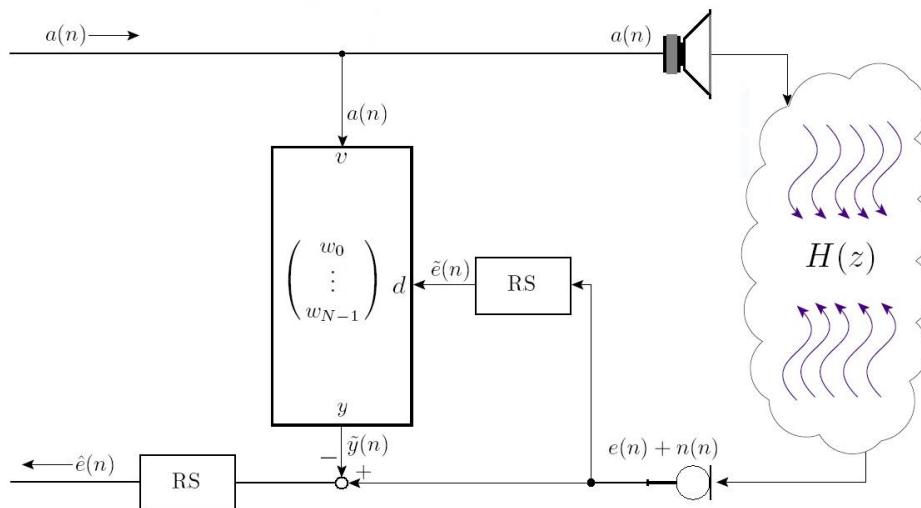
[a] [b] RYS. 18 Uwydatnianie sygnału mowy z szumem pasmowym (a) sygnał zaszumiony (b) nieliniowe widmowe odejmowanie (c) widmowe odejmowanie z wykorzystaniem właściwości słuchu człowieka (d) alg. LSA



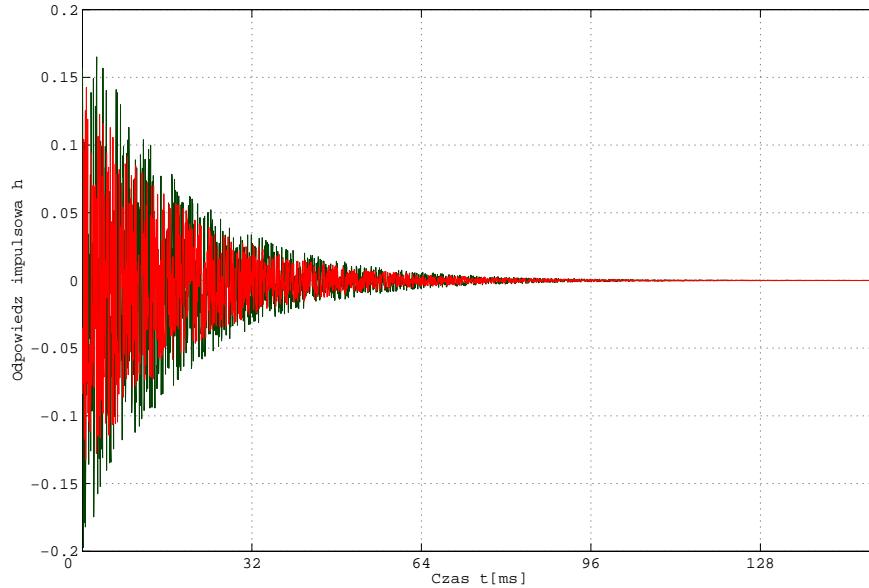
[a] [b] RYS. 19 Uwydatnianie sygnału mowy z szumem niestacjonarnym (a) sygnał czystej mowy (b) sygnał zaszumiony
[c] [d] (c) estymata szumu metodą minimów statystycznych (d) sygnał uwydatniony algorytmem LSA

5. Łączna redukcja szumów i echa akustycznego

Przez długi czas problem echa akustycznego był rozpatrywany w oderwaniu od zwykle obecnych zakłóceń szumowych. Stąd wyewoluowało całe mnóstwo niezwykłe wyszukanych form programistycznych, minimalizujących wspomniane zakłócenia niezależnie od siebie. Ostatnimi czasy pojawiły się propozycje spojrzenia z szerszej perspektywy łącząc obydwa zagadnienia w jedną całość [29]. Najprostszym, wręcz intuicyjnym rozwiązaniem, jest poddanie zaszumionego sygnału wraz z echem akustycznym redukcji szumów, by następnie próbować estymować współczynniki filtra adaptacyjnego. Na tej samej zasadzie można najpierw redukować echo, by później zająć się pozostałym szumem [23, 44]. Oba podejścia nie są jednak równoważne. W pierwszym z nich wstępna redukcja wywołuje zniekształcenia nieliniowe sygnału echa, co zakłóca proces adaptacji. W efekcie współczynniki nie osiągają optymalnych wartości, co skutkuje echem pozostałościowym. Z kolei przeprowadzenie procesu uczenia na zaszumionym sygnale powoduje znaczną rozbieżność współczynników, w przypadku gdy sygnał wejściowy jest bardzo mały, lecz różny od zera. Taka sytuacja występuje stale w sygnale mowy, więc trzeba albo stosować detektory obecności sygnału mowy, by zatrzymywać adaptację, albo stosować duże wartości parametrów regulujących. Dobrą próbą spełnienia postawionych wymagań okazało się zastosowanie struktury z przetwarzaniem wstępny zaszumionego sygnału [28], której schemat blokowy przedstawiony jest na poniższym rysunku.



RYS. 20 Algorytm łącznej redukcji szunów i echa akustycznego; struktura z przetwarzaniem wstępny.



RYS. 21 Zamodelowane odpowiedzi impulsowe otoczenia

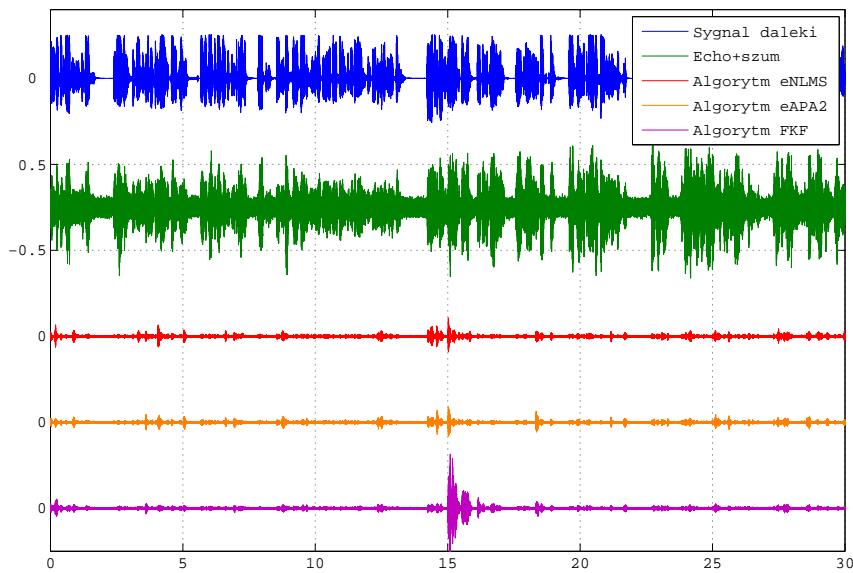
Powracające echo jest wstępnie przetwarzane przed adaptacją, by następnie usunąć pozostałe echo w kolejnej sekcji redukującej.

Zostało zaproponowane doświadczenie w celu potwierdzenia przydatności przedstawionego algorytmu. Najpierw zamodelowano dwie różne odpowiedzi impulsowe otoczenia. Mają one zbliżoną postać do rzeczywistych charakterystyk zamkniętych pomieszczeń. Zanikają z czasem w sposób bliski wykładniczemu. Takie też obwiednie mają przyjęte sygnały, wygenerowane w sposób losowy. Czas rewerberacji przyjęto na 128 ms, co odpowiada niedużym pomieszczeniom, jak małe pokoje czy np. wnętrze samochodu. Częstotliwość próbkowania ustalono na 16 kHz, wyznacza to liczbę współczynników filtru na 2048 w celu eliminacji echa na poziomie 60 dB. Zastosowano filtr o długości 1024 współczynnikiów, co daje w najlepszym wypadku tłumienie echo w granicach 30 dB. Ponadto po 15 sekundach odpowiedź impulsowa otoczenia została zmieniona, a to powinno dać odpowiedź na pytanie o jakość śledzenia algorytmów adaptacji. Wyniki symulacji są zaprezentowane na rysunku 22.

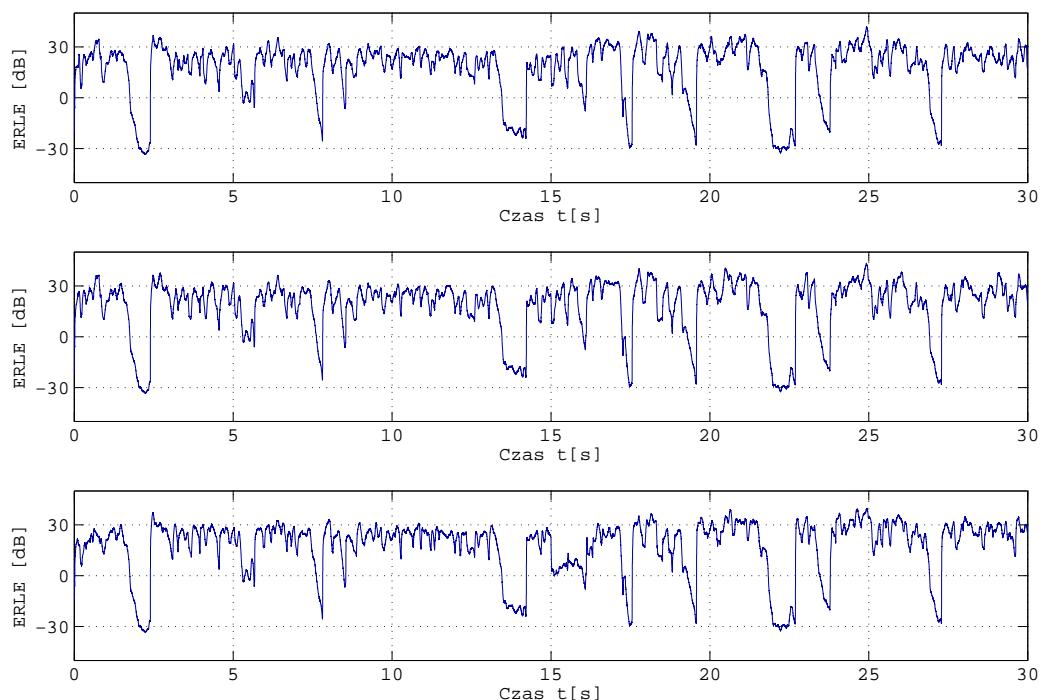
Rys. 23 przedstawia zestawienie współczynnika charakteryzującego wydajność eliminatora echa, tzw. ERLE zdefiniowane jako [19]:

$$(5.1) \quad \text{ERLE} = 10 \log_{10} \frac{E(e^2(n))}{E(\hat{e}^2(n))} [\text{dB}].$$

Przesymulowane algorytmy osiągają wartość tłumienia echo bliską założonej 30 dB. Pozatym algorytm FKF wykazuje wolniejszą zbieżność od implementacji stochastycznych. Tłumaczy się to użyciem współczynnika zapominania bliskiego jedności, gdyż tylko taka jego wartość zapewnia zbieżność algorytmu dla sygnału mowy.



RYS. 22 Wyniki przetwarzania struktury łącznej redukcji szumów i echo akustycznego z przetwarzaniem wstępny



[a] RYS. 23 Współczynnik tłumienia powracającego echo: ERLE (a) ϵ -NLMS (b) ϵ -APA-2
[c] (c) Fast Kalman Filter

6. Podsumowanie

Algorytmy łącznej redukcji szumów i echa akustycznego opierają się na wypracowanych rozwiązańach obu problemów niezależnie od siebie. Zjawisko echa zwykle usuwa się z wykorzystaniem filtrów adaptacyjnych mających za zadanie estymować współczynniki otoczenia odpowiedzialnego za jego powstawanie. W dziedzinie filtracji adaptacyjnej wyróżnia się dwa typy algorytmów: tzw. algorytmy stochastyczne, których zasada działania opiera się na minimalizacji błędu średniokwadratowego, oraz algorytmy deterministyczne, działające w oparciu o metodę najmniejszych kwadratów. Zaletą pierwszej grupy jest prostota i niezawodność, z drugiej jednak strony ich zbieżność jest poważnie ograniczona dla sygnałów, których próbki są mocno z sobą skorelowane, czego doskonałym przykładem są właśnie sygnały mowy. Druga grupa algorytmów nie wykazuje żadnej zależności ich zbieżności od zawartości widmowej przetwarzanych sygnałów, ponadto adaptacja jest znacznie przyspieszona w stosunku do algorytmów stochastycznych. Za wadę uznaje się o wiele większą złożoność obliczeniową. Można ją znacznie zredukować, stosując szybkie implementacje wykorzystujące transwersalną strukturę filtru. Tu jednak ujawnia się najpoważniejsza wada algorytmów deterministycznych, mianowicie niestabilność numeryczna przy uruchomieniu w skończonej precyzji. Wymagane są obliczenia wykorzystujące bardzo dużą liczbę bitów przeznaczonych na zapamiętywanie liczb w pamięci mikrokomputera.

Kolejnym aspektem algorytmów eliminacji echa jest wymagana stosunkowo duża ilość współczynników użytych filtrów adaptacyjnych w celu zapewnienia tłumienia echa na akceptowalnym poziomie. Zwiększa to radykalnie koszt obliczeń i zmusza do szukania nowych rozwiązań. Jednym z nich jest zastosowanie algorytmów blokowych, zmniejszających nakłady obliczeniowe poprzez przetwarzanie całych partii danych z użyciem specjalnie do tego celu przeznaczonych, bardzo optymalnych procedur, takich jak np. szybka transformata Fouriera. Wadą tego rozwiązania jest wprowadzenie opóźnienia pojawiącego się przetworzonego sygnału o wartości proporcjonalnej do długości bloku. Okazuje się dalej, że wspomnianego opóźnienia można też uniknąć zwiększając znów koszty, które i tak pozostają znacznie poniżej kosztów algorytmów standardowych.

Wśród algorytmów redukcji szumów można również wprowadzić pewien podział, mianowicie algorytmy wykorzystujące właściwości statystyczne przetwarzanych sygnałów – zwykle skomplikowane o lepszej jakości – jak też algorytmy niebiorące ich pod uwagę. Te drugie są z reguły o wiele prostsze za cenę gorszych parametrów. Najważniejszą jednak sprawą przesądzającą o właściwościach używanych rozwiązań jest poprawna estymata szumu obecnego w jednym kanale z sygnałem pożądanym. Niektóre z rozwiązań wyznaczają ją podczas nieobecności sygnału mowy. Pociąga to za sobą dodatkową implementację detektorów sygnału mowy. Inne pomysły pozwalają badać taki szum bez zwracania uwagi na to, czy mowa jest w danej chwili obecna, czy nie.

Powiązanie ostatecznie obydwóch dziedzin cyfrowego przetwarzania sygnałów prowadzi do szukanych algorytmów łącznej redukcji szumów i echa akustycznego. W drugiej części pracy w oparciu o wiedzę wynikającą z przeprowadzonego przeglądu teoretycznego, zostanie wybrany, opisany, a następnie uruchomiony i dokładnie przebadany jeden z takich algorytmów.



Implementacja algorytmu łącznej redukcji szumów i echa akustycznego z wykorzysta- niem procesora sygnałowego

1. Budowa i cechy charakterystyczne procesora sygnałowego

Obliczenia prowadzone w czasie rzeczywistym zaimplementowano na procesorze sygnałowym ADSP-21061 Sharc wyprodukowanym przez firmę Analog Devices w roku 1995. Jedenaście lat dla techniki mikroprocesorowej to bardzo długi okres czasu. Jednak – jak zostanie pokazane – procesor ten bardzo dobrze spełnia swoją funkcję przetwarzania sygnałów audio i pomimo swojego dość zaawansowanego wieku wciąż wiele ciekawych aplikacji można na nim uruchomić. Jest to procesor zmiennoprzecinkowy, tzn. posiada zaimplementowaną arytmetykę zmiennoprzecinkową, przetwarzającą 32-bitowe liczby pojedynczej precyzji o 24-bitowej mantysie i 8-bitowej cesze bądź liczby o specjalnym formacie z rozszerzoną precyją mantysy o 8 bitów i ostatecznie zajmujące 40 bitów w wewnętrznych rejestrach układu. Nominalnie procesor jest przeznaczony do pracy z zegarem 50 MHz. Uwzględniając fakt, że w pewnych szczególnych okolicznościach możliwe jest wykonanie w jednym takcie zegarowym trzech instrukcji zmiennoprzecinkowych, tj. mnożenia zawartości dwóch wybranych rejestrów oraz dodawania i odejmowania zawartości dwóch innych rejestrów, jasne staje się, dlaczego we wszystkich biuletynach firmy reklamujących procesor widnieje wydrukowany dużą czcionką napis 150 MFLOPS. W praktyce, przy bardzo wydajnym programowaniu oscyluje się wokół granicy dwóch operacji zmiennoprzecinkowych w jednym takcie zegarowym.

Jednostka numeryczna procesora sygnałowego to trzy połączone równolegle jednostki obliczeniowe: układ mnożący, jednostka arytmetyczno-logiczna oraz przesuwnik. Pierwsza wykonuje mnożenie liczb stałoprzecinkowych w formatach: całkowitym i ułamkowym ze znakiem i bez znaku oraz zmiennoprzecinkowym. ALU dokonuje dodawania, porównywania liczb, całej gamy operacji logicznych, oblicza maksimum, minimum, dokonuje konwersji pomiędzy formatami stało- i zmiennoprzecinkowymi. Dzięki przesuwnikowi można łatwo manipulować bitami operandów. Przesuwanie, ustawianie poszczególnych bitów, wycinanie fragmentów bitów poszczególnych liczb to niektóre z zadań przesuwnika. Trzy wspomniane jednostki wymieniają dane poprzez szesnaście rejestrów ogólnego przeznaczenia. Pracują one niezależnie, problemem jest tylko zorganizowanie ich pracy odpowiednio długim rozkazem.

Active Register File			
R0	3D9C300000	R8	3C3D83BEOO
R1	3D14600000	R9	3C86F2B900
R2	0000006FOO	R10	3B8B89A400
R3	3F80000000	R11	3F90BE9C00
R4	3EBC540000	R12	BBOCC49400
R5	3EBC580000	R13	3A73235000
R6	3E47100000	R14	3C2505B700
R7	3DCF000000	R15	3FOA963500

RYS. 24 Przykładowa zawartość rejestrów danych procesora sygnałowego

Instrukcje procesora sygnałowego mają długość 48 bitów i pobierane są z wewnętrznej pamięci SRAM o pojemności 1Mb. Pamięć ta dzieli się na dwa bloki. W pierwszym z nich można przechowywać instrukcje oraz dane, drugi blok przeznaczony jest tylko na dane liczbowe. Dane te mogą mieć długość 32 lub 40 bitów w przypadku wykorzystywania arytmetyki o rozszerzonej precyji. Konfiguracja długości słowa pamięci odbywa się poprzez wpisanie odpowiedniej wartości do rejestru systemowego procesora. Przemieszanie razem danych i instrukcji w pierwszym bloku pamięci tworzy tzw. superharvardzką architekturę procesora. Ma to na celu zwiększenie do maksimum efektywności przetwarzania danych, można bowiem w jednym takcie zegarowym pobrać dwa

argumenty rozkazu z obu bloków równocześnie. Skąd w takim razie pobrać instrukcję? W jądrze procesora znajduje się pamięć notatnikowa gdzie można przechować do 32 rozkazów przeznaczonych specjalnie na takie okazje.

Kolejną ważną jednostką są tzw. generatorы adresów. Zawierają one szesnaście 32-bitowych rejestrów **I** przechowujących pozycję danej w pamięci do przeczytania lub zapisania. Po osiem takich rejestrów przypada na każdy z bloków pamięci. Oprócz zwykłego wskaźnika rejestyry te pełnią też inne, dodatkowe funkcje. Jedną z nich jest autoinkrementacja wskaźnika po każdym dostępie do pamięci o wartość przechowywaną w specjalnym rejestrze modyfikacyjnym **M**. Można również pobrać daną przesuniętą od wartości rejestrów **I** o wartość **M** bez zmiany faktycznej zawartości rejestrów **I**. Bardzo przydatną właściwością, szczególnie w procedurze FFT jest możliwość wyłączenia dla wybranego rejestrów **I** tzw. adresowania z odwróceniem bitowym. Dzięki specjalnemu rozkazowi można pobrać daną z pamięci spod adresu, który jest odwróceniem bitowym zawartości rejestrów **I**. I tak, gdy w **I0** mamy wartość **0x02468000**, to faktycznie zostanie odczytana komórka pamięci o adresie **0x00016240**. Ostatnią często wykorzystywaną możliwością oferowaną przez generatorы adresów stanowią tzw. bufore cykliczne. Przeznaczone są na to kolejne dwie grupy rejestrów **B** i **L**. W rejestrze **L** przechowywana jest długość bufora kołowego, w rejestrze **B** – adres bazowy. Próba zmodyfikowania zawartości wskaźnika **I** na wartość **0x300**, gdy adresem bazowym jest **0x200**, a długość **L** wynosi **0x80**, skończy się zapisaniem do rejestrów **I** wartości **0x20**. Powyższa funkcja jest bardzo przydatna przy tworzeniu tzw. linii opóźniających z danymi, przydatnych szczególnie przy programowaniu filtrów cyfrowych.

The screenshot displays two windows titled "Active DAG1 (DM)" and "Active DAG2 (PM)". Both windows show a table of memory addresses and their corresponding values.

Active DAG1 (DM)			
I0	00024396	M0	00000000
I1	00024DC1	M1	00000001
I2	00024DD2	M2	00000002
I3	00024B01	M3	00000003
I4	00024DE2	M4	00000001
I5	00024DF4	M5	FFFFFFFFFF
I6	00024E04	M6	FFFFFFFFFF
I7	00024E15	M7	FFFFFFFFFF
		B0	00024600
		B1	00024247
		B2	00024600
		B3	00024A00
		B4	00024A00
		B5	00024600
		B6	00024600
		B7	00000000
		L0	00000000
		L1	00000000
		L2	00000000
		L3	00000200
		L4	00000000
		L5	00000000
		L6	00000000
		L7	00000000

Active DAG2 (PM)			
I8	020C01	M8	000000
I9	020C12	M9	000001
I10	0218BB	M10	000002
I11	0215FE	M11	000003
I12	020C22	M12	FFFEEE
I13	020C34	M13	FFFFFD
I14	020721	M14	FFFFFE
I15	020C46	M15	FFFFFF
		B8	021400
		B9	021200
		B10	021400
		B11	021400
		B12	021601
		B13	021601
		B14	021601
		B15	000000
		L8	000000
		L9	000000
		L10	000000
		L11	000200
		L12	000000
		L13	000000
		L14	000000
		L15	000000

RYS. 25 Przykładowa zawartość generatorów adresów. Zainicjowane po jednym buforze cyklicznym w obu blokach pamięci

Rozkazy opisujące zachowanie procesora w danej chwili pobierane są sekwencyjnie z pamięci. Przetwarzanie jednej instrukcji trwa trzy cykle zegarowe. W pierwszym z nich jest tzw. faza pobrania rozkazu. Następnie zachodzi dekodowanie i w końcu wykonanie. Aby nie marnotrawić zbyt daleko czasu, wprowadzono tzw. przetwarzanie potokowe. Rozkazy pobierane i przetwarzane są w jednym ciągu. Podczas dekodowania jednego rozkazu już pobierany jest następny. Doprowadza to efektywność przetwarzania do jednego rozkazu na jeden cykl zegara. Niestety są sytuacje, kiedy tak przygotowany cykl zostaje zaburzony. Dzieje się tak w przypadku skoków w inne miejsce programu, które to zawiera w sobie zwyczajne skoki, jak też skoki z zapamiętaniem adresu powrotnego w przypadku wywołania podprogramu czy wystąpienia przerwania. Po przystąpieniu do fazy wykonywania takiego rozkazu następny zdekodowany już rozkaz oraz dopiero co pobrany muszą zostać

usunięte z jednostki przetwarzającej i cały cykl zacznie się na nowo w innym miejscu programu. Dobrym remedium na wspomnianą niedoskonałość jest wykorzystanie tzw. skoków z opóźnieniem. Napisanie rozkazu skoku z modyfikatorem (*db*), np.

```
jump label(db);
f8=f0*f3, f15=f8+12;
nop;
```

spowoduje wykonanie dwóch rozkazów (zdekodowanego i pobranego) znajdujących się za rozkazem skoku pod adres ukryty pod etykietą „label” przed faktycznym skokiem. W ten sposób dwa takty zegarowe zostały zaoszczędzone. Gdy jest taka możliwość, można również połączyć rozkaz skoku z adresowaniem przez wskaźnik z operacją arytmetyczną:

```
jump (m8,i8)(db), f0=f1*f2;
```

W ten sposób rozkaz skoku mamy zupełnie za darmo.

Inną przyczyną zakłócenia sekwencyjności wykonywania programu są przerwania. Przerwania są podstawowym mechanizmem w systemach mikroprocesorowych i spełniają wiele przydatnych funkcji. Mamy dostępne przerwania wywoływanie przez czynniki zewnętrzne, mianowicie **RESET** oraz trzy przerwania **IRQ0-2**, jak również całe mnóstwo przerwań uruchamianych na skutek działań odbywających się wewnątrz procesora. Wszystkie przerwania oprócz **RESET** są maskowalne, co oznacza, że w dowolnej chwili czasu możemy czasowo dane przerwanie uczynić nieaktywne. Każde przerwanie ma swój priorytet, w przypadku wystąpienia dwóch przerwań jednocześnie, pierwsze obsługiwane jest przerwanie o większej ważności. Przerwania mogą być zagnieżdżane. Znaczy to, że program obsługuje przerwania o niższym priorytecie, może zostać przerwany przez program obsługi o priorytecie wyższym.

RYS. 26 Przerwania procesora sygnałowego ADSP 21061 (wg priorytetów)

Interrupts Registers	
	IRPTL IMASK
	04C000010 00800503
emu:	0 1
rst:	0 1
sovf:	0 0
tmszh:	1 0
virpt:	0 0
irq2:	0 0
irq1:	0 0
irq0:	0 1
spr0:	0 1
spr1:	0 0
spto:	0 0
spt1:	0 0
ep0:	0 0
ep1:	0 0
cb7:	0 0
cb15:	1 0
tmsl:	1 1
fix:	0 0
f1to:	0 0
f1tu:	1 0
f1ti:	0 0
user0:	0 0
user1:	0 0
user2:	0 0
user3:	0 0

1. Przerwanie emulatora
2. RESET
3. Przepelenie stosu
4. Wyzerowanie licznika taktów zegarowych – opcja z wysokim priorytetem
5. Przerwanie wektorowe w systemie mikroprocesorowym
6. Przerwanie zewnętrzne 2
7. Przerwanie zewnętrzne 1
8. Przerwanie zewnętrzne 0
9. SPORT0 – odbiór
10. SPORT1 – odbiór
11. SPORT0 – wysłanie
12. SPORT1 – wysłanie
13. Zewnętrzny port 0
14. Zewnętrzny port 1
15. Przepelenie buforu cyklicznego 7
16. Przepelenie buforu cyklicznego 15
17. Wyzerowanie licznika taktów zegarowych – opcja z niskim priorytetem
18. Przepelenie stałoprzecinkowe
19. Przepelenie zmennoprzecinkowe
20. Niedobór zmennoprzecinkowy
21. Nieprawidłowa liczba zmennoprzecinkowa
22. Przerwanie programowe 0
23. Przerwanie programowe 1
24. Przerwanie programowe 2
25. Przerwanie programowe 3

Najwyższy priorytet posiada **RESET** (gdy nie korzystamy z emulatora), najniższy – przerwania programowe ogólnego przeznaczenia. W momencie wystąpienia przerwania wykonywanie

programu przeniesione zostaje w specjalne miejsce – program obsługi przerwania, gdzie będzie on kontynuowany. W momencie napotkania instrukcji zakończenia programu obsługi, sterowanie powróci w miejsce, z którego zostało wytrącone. Dzięki przerwaniom można natychmiast zareagować na pewne zdarzenia zewnętrzne, np. naciśnięcie przycisku, czy wewnętrzne, np. nieprawidłowa wartość liczby zmienoprzecinkowej bądź przerwanie z wewnętrznego licznika taktów zegarowych. Pojawienie się przerwania jest sygnaлизowane ustawieniem odpowiadającego mu bitu w rejestrze systemowym **IRPTL** (nawet gdy jest ono zamaskowane). Każde z przerwań może zostać wywołane na drodze programowej, jeśli nie jest ono zamaskowane oraz ustawimy odpowiadający mu bit w rejestrze sterującym przerwaniem **IRPTL**. Przerwania maskuje się, ustawiając odpowiadające im bity w rejestrze **IMASK** na logiczną wartość 0.

Bardzo przydatnym mechanizmem przy obsłudze przerwań, podprogramów czy ogólnie przy tworzeniu algorytmów jest możliwość zmiany kontekstu procesora. Wpisując odpowiednią wartość do rejestru sterującego procesora **MODE1**, aktywny bank rejestrów **r0-r15** może zostać zamieniony na bank tzw. rejestrów cieni. Wszystkie rozkazy arytmetyczne czy odczyty z pamięci będą wykonywane za pomocą rejestrów zawierających inne wartości. Podobnie można uczynić z bankiem generatorów adresów. Tym sposobem, przełączając w podprogramach kontekst procesora, możemy być pewni, że dane przechowywane w rejestrach potrzebne w programie głównym nie zostaną nadpisane.

Innym pomocnym i często wykorzystywanym układem procesora sygnałowego jest licznik taktów zegarowych. Bardzo wygodnie można za jego pomocą odmierzać wybrane odcinki czasu, bazując na oscylatorze kwarcowym generującym takty zegarowe. Zawartość owego licznika, która od strony programowej widziana jest jako rejestr **TCOUNT**, zmniejszana jest w każdym cyklu zegarowym. Po osiągnięciu zera generowane są dwa przerwania: jedno o bardzo wysokim priorytecie, drugie o stosunkowo niskim. Od programisty zależy, które z nich wybierze, bazując na specyfice wykorzystania licznika. Następnie wartość **TCOUNT** jest odnawiana z rejestrów **TPERIOD** i cały cykl zaczyna się od nowa. Znając wartość jednego cyklu zegara procesora można w bardzo łatwy sposób zaprogramować, by np. jakiś specjalny kod był wykonywany dokładnie co 1 sekundę. Licznik można w każdej chwili włączyć bądź zatrzymać za pomocą bitu **TIMEN** w rejestrze sterującym **MODE2**.

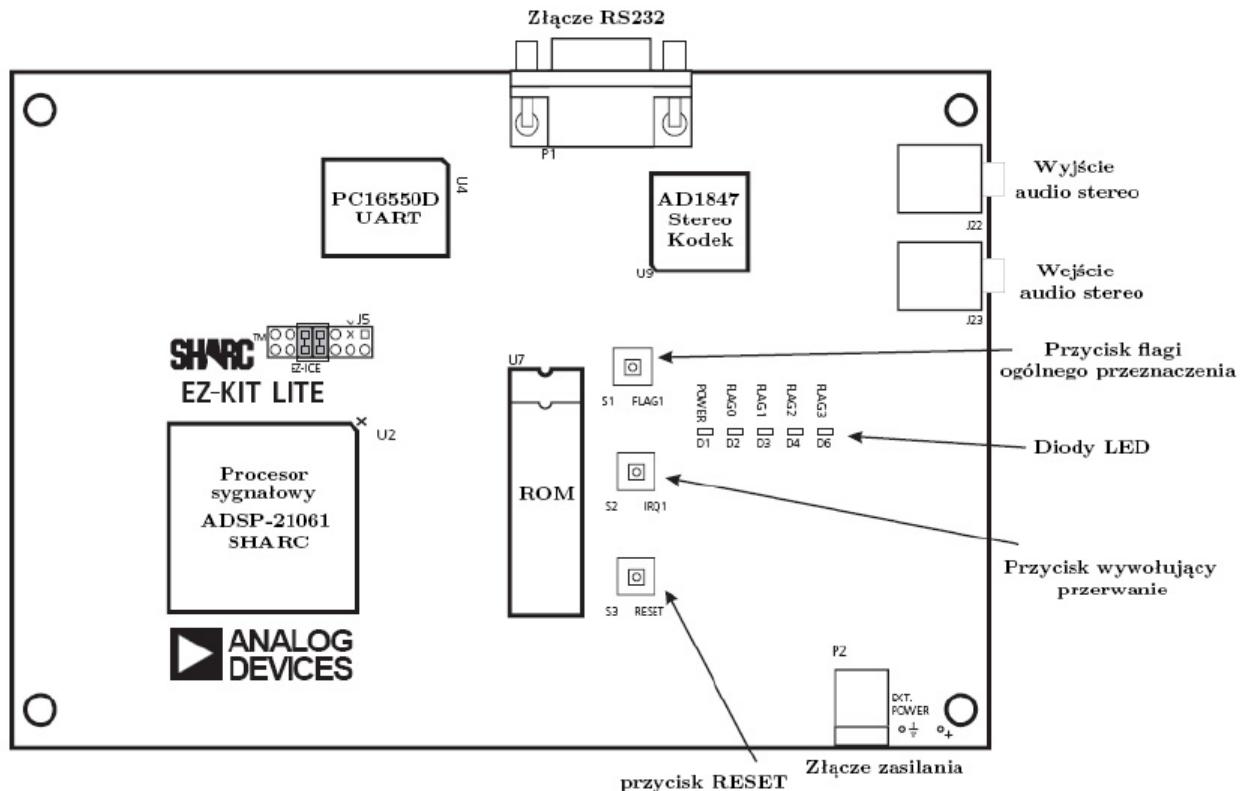
W niniejszym paragrafie zostały przedstawione podstawowe właściwości procesora sygnałowego jako osobnej jednostki sterującej. W pracy został wykorzystany pełny system mikroprocesorowy wyprodukowany przez firmę Analog Devices w postaci zestawu uruchomieniowego EZ-KIT Lite. Zostanie on teraz zaprezentowany.

2. Zestaw uruchomieniowy procesora sygnałowego

Zestaw uruchomieniowy procesora sygnałowego EZ-KIT Lite zawiera w sobie następujące komponenty:

- procesor sygnałowy ADSP-21061 Sharc pracujący z zegarem zdegradowanym do 40 MHz;
- 16-bitowy kodek AD1847 dokonujący konwersji A/C i C/A;
- pamięć ROM przechowującą program startowy płytka, odpowiednik BIOSU w komputerach PC, z możliwością podmiany na inną kość pamięci z dowolnym programem;
- PC16550D UART nadzorujący transmisję szeregową oraz złącze RS-232 do podłączenia do komputera PC w celu komunikacji;
- trzy przyciski do generowania zdarzeń i obsługi płytka: przycisk RESET (pomocny po utracie kontroli nad procesorem), przycisk podłączony na wejście IRQ1 procesora (generujący przerwanie w momencie naciśnięcia) oraz trzeci przycisk oznaczony FLAG1 (do programowego wykorzystania przez użytkownika);
- diodę sygnalizującą podłączenie napięcia zasilania oraz 4 diody kontrolowane przez piny FLAG procesora, których stany logiczne można zmieniać dowolnie programowo;
- wejście i wyjście sygnału audio stereo połączone bezpośrednio z kodekiem;
- gniazdo napięcia zasilania 9V.

Wygląd płyty i rozmieszczenie komponentów przedstawione jest na poniższym rysunku:



RYS. 27 Rozmieszczenie komponentów na płytce EZ-KIT Lite firmy Analog Devices

Po podłączeniu napięcia zasilania do płyty EZ-KIT Lite przesyłany jest program z pamięci ROM do pamięci SRAM procesora i następuje uruchomienie programu monitora. Nadzoruje on komunikację procesora z UART-em i umożliwia przesyłanie i uruchamianie kodu użytkownika, a także emulację i debuggowanie programów. Program monitora zajmuje 1676 słów po 48 bitów każde przeznaczone na instrukcje i dane w pierwszym bloku pamięci oraz 3 słowa 32-bitowe przeznaczone na dane w drugim bloku pamięci. Jak widać, z 1Mb dostępnego obszaru użytkownik musi pogodzić się ze stratą $\approx 78.6\text{kb}$ pamięci, głównie w pierwszym bloku, jeżeli chce korzystać z możliwości oferowanych przez program monitora. Dodatkowo monitor po każdym resecie, zatrzymaniu programu czy restarcie wymusza wartości bitów **IMDW0** i **IMDW1** rejestru **SYS CON** na logiczne 1 i 0 odpowiednio. Sprawia to, że dane odczytywane z pierwszego bloku będą miały długość 48 bitów a dane z drugiego bloku długość 32 bitów. Jeżeli chcielibyśmy korzystać z arytmetyki rozszerzonej precyzji działającej na liczbach o długości 40 bitów, jedyną możliwością ich przechowywania w pamięci danych (blok 2) jest 48-bitowy dostęp do owej pamięci. Stąd trzeba pamiętać o odpowiedniej konfiguracji bitów systemowych po każdej akcji z wykorzystaniem programu monitora.

Po uwzględnieniu wymagań pamięciowych programu monitora i skonfigurowaniu drugiego bloku pamięci na zapis/odczyt 32 bitowy mapa pamięci wygląda następująco:

Segment	Adres początkowy	Adres końcowy	Ilość słów
Blok przerwań	0x20000	0x2007f	128
Kod programu	0x20080	0x209ff	2432
Dane – blok pierwszy	0x20a00	0x21973	3956
Kod i dane monitora	0x21974	0x21fff	1676
Dane – blok 2	0x24000	0x27ffc	16380
Dane monitora	0x27ffd	0x27fff	3

TABELA 4. Mapa pamięci płyty EZ-KIT Lite z ustanowieniem 32-bitowego dostępu do pamięci w bloku drugim

Praca z kodekiem od strony programowej jest nieskomplikowana. Procesor komunikuje się z nim z wykorzystaniem własnego portu szeregowego **SPORT0**, a dane z niego przesyłane są do pamięci za pomocą kontrolera DMA umieszczonego wewnątrz ADSP. Dzięki temu procesor główny zostaje odciążony od zadań przesyłania danych pomiędzy swoją wewnętrzną pamięcią a zewnętrznym źródłem. Jedyną wymaganą akcją jest inicjalizacja kontrolera DMA oraz kodeka za pomocą odpowiednich słów sterujących. Ustawia się wtedy częstotliwość próbkowania, długość słowa danych, wzmacnianie przetwornika C/A itp. Po zakończeniu tego etapu kodek rusza do pracy i generuje przerwanie **SPR0I** po skompletowaniu każdej nowej próbki. Wartości sygnału z lewego i prawego kanału znajdują się wtedy w zdefiniowanym wcześniej miejscu w pamięci. Aby przesłać daną do przetwornika C/A wystarczy ją wpisać do buforu transmisyjnego zajmującego dwa słowa w pamięci.

Jak już zostało wspomniane, procesor sygnałowy posiada cztery zewnętrzne przerwania. Przycisk RESET na płytce EZ-KIT Lite podłączony jest do przerwania **RESET**, skąd naciśnięcie tego przycisku powoduje uruchomienie programu monitora. Przerwanie **IRQ2** o najwyższym priorytecie z przerwań zewnętrznych zaraz po resecie wykorzystywane jest przez monitor do komunikacji z UART'em. **IRQ1** podłączone jest do przycisku, a **IRQ0** nie jest wykorzystywane. Dodatkowo na płytce umieszczone są cztery wspomniane diody LED sterowane przez procesor za pomocą pinów

FLAG. Każde z zewnętrznych wyprowadzeń FLAG procesora może zostać ustawione jako wejście bądź wyjście zależnie od stanu odpowiedniego bitu w rejestrze sterującym **MODE2**. Gdy pin jest ustawiony jako wyjście, możliwa jest zmiana jego stanu poprzez wpisanie wartości do bitu **FLAG** w rejestrze systemowym **ASTAT**. Z kolei w przypadku korzystania z wyprowadzenia jako wejścia, w rejestrze **ASTAT** bity **FLAG** będą miały odpowiednią wartość, taką samą jak wartość na wyprowadzeniu. Wystawienie wartości logicznej 0 na pin FLAG powoduje zaświecenie diody. Naciśnięcie przycisku FLAG1 powoduje ustawienie bitu **FLAG** w rejestrze **ASTAT** w stan logicznego zera.

Po przedstawieniu cech charakterystycznych płyty uruchomieniowej EZ-KIT Lite zostanie teraz zaprezentowane środowisko programistyczne procesora sygnałowego.

3. Środowisko programistyczne procesora sygnałowego

Podstawowym narzędziem udostępnionym przez firmę Analog Devices do programowania ich procesorów sygnałowych jest pakiet VisualDSP++. Wersją programu przeznaczoną do współpracy z płytą EZ-KIT Lite jest wersja 3.0 programu. Praca z tym narzędziem jest bardzo wygodna. VisualDSP++ posiada zintegrowany asembler, kompilator C/C++, linker, splitter, loader, symulator oraz emulator, czyli wszystkie narzędzia potrzebne do efektywnego tworzenia i debbugowania kodu.

Programowanie procesora sygnałowego rozpoczyna się od utworzenia pliku opisu dla linkera. W pliku tym definiuje się segmenty – fragmenty ciągłego obszaru pamięci, gdzie będą przechowywane dane lub program. Przykładowy opis segmentu wygląda następująco:

```
seg_dmda{ TYPE(DM RAM) START(0x24000) END(0x26000) WIDTH(32) }
```

Definiuje on segment w pamięci danych o wpisanym adresie początkowym i końcowym oraz o długości słowa w pamięci równej 32 bitom. Następujący ciąg instrukcji definiuje tablice dziesięciu liczb o adresie początkowym oznaczonym etykietą „data”:

```
.segment /dm seg_dmda;  
.VAR data[10];  
.endseg;
```

Wszystkie dane widniejące pomiędzy dyrektywami **.segment** oraz **.endseg** zostaną umieszczone w pamięci procesora w zdefiniowanym obszarze przypadającym na segment. Za pomocą narzędzia Expert Linker z pakietu VisualDSP++ tworzenie segmentów jest intuicyjne i proste. Po przypisaniu adresów poszczególnych segmentów można przystąpić do tworzenia kodu asemblerowego bądź C/C++ , umieszczając go w odpowiednim miejscu w pamięci przeznaczonym na instrukcje.

Asembler procesorów sygnałowych firmy Analog Devices różni się znacznie od assemblerów innych typów procesorów czy mikrokontrolerów. Mianowicie zrezygnowano całkowicie z mnemoników przy opisie operacji arytmetycznych. I tak, by dodać zawartość dwóch rejestrów **f1** i **f2**, a wynik umieścić w rejestrze **f3**, wystarczy napisać:

```
f3=f1+f2;
```

co w przypadku bardziej złożonych instrukcji znacznie ułatwia programowanie. Co prawda, nowsze procesory sygnałowe innych firm również oferują algebraiczną listę rozkazów, jednak faktem jest, że to procesory Analog Devices jako pierwsze zastosowały tego rodzaju ułatwienie.

Rozkazy mają długość 48 bitów, więc za ich pomocą można zaprogramować wiele działań procesora w jednym takcie zegarowym. I tak jedna grupa rozkazów wykonuje działania arytmetyczne z jednoczesnym sieganiem do pamięci po dane. Przykładowy format rozkazu:

```
IF condition compute, ureg=PM(Ic,Md);
```

Najpierw sprawdzany jest pewien warunek, np. **IF EQ**, co sprawdza, czy w ALU po ostatniej operacji w wyniku otrzymano 0. Gdy taka sytuacja miała miejsce, wykonywana jest operacja arytmetyczna zaprogramowana w polu **compute** oraz do rejestru wyspecyfikowanego w polu **ureg** zostaje wpisana zawartość komórki pamięci programu wskazywana przez wartość w rejestrze **Ic**, po czym zostaje on zwiększy o wartość modyfikatora **Md**. Gwiazdą wśród rozkazów tego typu z pewnością są rozkazy typu:

```
compute, DM(Ia,Mb)=dreg1, PM(Ic,Md)=dreg2;
```

które oprócz obliczeń siegają po dwa argumenty w jednym takcie zegarowym, pod warunkiem przechowania instrukcji w pamięci notatnikowej; **dreg** oznacza jeden z rejestrów **r0...r15,f0...f15**. Rejestry **r0** i **f0** oznaczają faktycznie ten sam rejestr, lecz pierwsza litera oznacza, czy dane w tych rejestrach będą traktowane przez procesor numeryczny jako stało- lub zmiennoprzecinkowe. Jak widać,

w powyższym rozkazie nie ma możliwości sprawdzenia warunku, a zbiór źródłowych/docelowych rejestrów procesora został znaczco ograniczony. Tylko tyle lub aż tyle dało się zapisać w 48-bitowym polu rozkazowym.

Innym ciekawym rozkazem z rodzaju "dużo w jednym taktie" jest rozkaz postaci:

IF condition JUMP(PC,reladdr6), ELSE compute, DM(Ia,Mb)=dreg;

Wykonuje on skok o 6-bitową wartość odstępu od aktualnej zawartości licznika rozkazów, gdy warunek **condition** jest spełniony, gdy nie dokonuje obliczeń i odczytu z pamięci.

Warto zwrócić uwagę na stosunkowo łatwe organizowanie pętli w języku asemblera procesora sygnałowego Analog Devices. Otóż służy do tego jeden rozkaz:

lcntr=data16, do label until lce;

Na początku ładowany jest licznik pętli **LCNTR** wartością wskazującą liczbę obiegów pętli, po czym wykonywane są rozkazy umieszczone pod omawianym rozkazem, przy czym ostatnim z nich jest rozkaz o symbolicznym adresie „label”. Po jego wczytaniu pętla wraca na początek i trwa tak długo, dopóki nie jest spełniony warunek **lce** co oznacza utratę poprawnej zawartości licznika obiegów pętli.

Dzięki przesuwnikowi łatwa jest manipulacja na bitach rejestrów systemowych, np.

bit tgl sreg data32;

powoduje zanegowanie wszystkich bitów rejestru systemowego **sreg**, które to bity mają wartość 1 w 32-bitowym słowie wyspecyfikowanym w rozkazie.

Na koniec wypadałoby zademonstrować, co kryje się pod tajemniczym słowem **compute**. Otóż oznacza ono dowolną operację wykonywaną przez procesor numeryczny – mnożarkę, ALU i przesuwnik. I tak, może to być rozkaz dodawania z przeniesieniem dla liczb stałoprzecinkowych:

Rn=Rx+Ry+CI;

może to być ekstrakcja mantysy liczby zmiennoprzecinkowej:

Rn=mant Fx;

albo mnożenie liczb zmiennoprzecinkowych:

fn=Fx*fy;

w końcu może to być suma logiczna zawartości jednego rejestru z zawartością innego rejestru przesuniętą w lewo o pewną ilość bitów:

Rn=Rn or lshift Rx by data8;

Oczywiście „prawdziwe” rozkazy to rozkazy wielofunkcyjne, możliwe do wykonania również przez procesor numeryczny. Podstawa to zaprzegnięcie do jednoczesnej pracy układu mnożącego i ALU, np. w rozkazie mnożenia z dodawaniem:

Fm=F3-0*F7-4, Fa=F11-8-F15-12;

Widać tu pewne restrykcje co do wyboru operandów, związane ze skońzoną długością 48-bitowego pola rozkazowego. Inny przykład to stałoprzecinkowe mnożenie z akumulacją, połączone ze średnią arytmetyczną:

MRF=MRF+R3-0*R7-4(SSFR), Ra=(R11-8+R15-12)/2;

W powyższym rozkazie wykorzystywany jest 80-bitowy rejestr **MRF** będący w stanie pomieścić do szesnastu wyników takich działań. Modyfikator **SSFR** oznacza, że oba argumenty mnożenia mają format ułamkowych liczb ze znakiem, a wynik zostanie zaokrąglony do możliwie najbliższej zapisowi wartości 32 bitowej.

ALU potrafi wykonywać podwójną operację, – dodawania i odejmowania na tych samych argumentach, co wykorzystywane jest np. w procedurze FFT:

$$Ra = Rx + Ry, \quad Rs = Rx - Ry;$$

Ostateczne połączenie mnożenia, podwójnego dodawania oraz operacji na pamięci prowadzi do prawdziwej bomby rozkazowej:

$$Fm=F3-0*F7-4, \quad Fa=F11-8+F15-12, \quad Fs=F11-8-F15-12, \quad \text{dreg1}=DM(Ia,Mb), \quad \text{dreg2}=PM(Ic,Md);$$

To jest właśnie ów słynny rozkaz wykonujący w jednym taktie trzy operacje zmiennoprzecinkowe oraz dwa odczyty/zapisy do pamięci.

Oprócz asemblacji programów, VisualDSP++ oferuje niezwykle ważną funkcję, mianowicie – symulator. Autor nie wyobraża sobie stworzenia dużego działającego projektu w języku niskiego poziomu bez dokładnego przetestowania każdego ze składających się nań komponentów. A to z tej prostej przyczyny, że pisanie programu w asemblerze z wykorzystaniem instrukcji wielofunkcyjnych jest procesem niezwykle błędogennym. Do takich konstrukcji programistycznych zaliczyć można poprawne zainicjowanie pętli z równoczesnym mnożeniem, dodawaniem oraz odczytami albo zapisami do pamięci, poprawne przeliczanie adresów czy konwersje różnych formatów. Niezwykle mylące są czasem mnemoniki testowanych warunków. Można czasem nie być pewnym do końca, co się stanie po wykonaniu danego rozkazu, a symulator daje na to pytanie szybką i pewną odpowiedź. VisualDSP++ pozwala na pracę krokową programu, ustawianie pułapek, tzw. watchpoints itp.

Program(PM) Memory [Floating Point 32 bit]

Address	Value
[020B40]	-6.1579049E-006
[020B41]	-2.5890768E-006
[020B42]	-5.3979456E-006
[020B43]	-1.1636973E-005
[020B44]	9.3057752E-006
[020B45]	1.3014302E-005
[020B46]	-2.8163195E-006
[020B47]	0.0
[020B48]	1.0713935E-005
[020B49]	-4.9322644E-006
[020B4A]	-2.0861626E-006
[020B4B]	1.8134713E-005
[020B4C]	-1.4021993E-005
[020B4D]	2.3026019E-005
[020B4E]	-3.3378601E-006
[020B4F]	-7.4505806E-007
[020B50]	2.4318695E-005
[020B51]	-8.4042549E-006
[020B52]	-1.052022E-005
[020B53]	-8.2924962E-006

Data(DM) Memory [Floating Point 32 bit]

Address	Value
[024E25]	1.363552
[024E26]	-2.7854347
[024E27]	1.8415021
[024E28]	-0.16326848
[024E29]	-0.37958583
[024E2A]	-0.1011947
[024E2B]	0.67714334
[024E2C]	-0.43192291
[024E2D]	-0.15066011
[024E2E]	0.090265103
[024E2F]	0.10605583
[024E30]	0.00041995663
[024E31]	0.11798807
[024E32]	-0.28141731
[024E33]	0.24214883
[024E34]	-0.099540517
[024E35]	1.3635143
[024E36]	-2.7857964
[024E37]	1.8421906

RYS. 28 Przykładowa zawartość pamięci w bloku pierwszym i drugim w postaci 32-bitowych liczb zmiennoprzecinkowych

Można podejrzeć zawartości pamięci podczas działania programu, nawet więcej: można oglądać dowolny rejestr systemowy począwszy od rejestrów arytmetycznych, poprzez układy czasowe, stos, liczniki pętli, rejestrysty sterujące przerwaniami, generatory adresów podstawowe oraz alternatywne,

pamięć notatnikową, kończąc na rejestrach sterujących procesorem. Możliwe jest skopiowanie zawartości pamięci do pliku w celu dalszej analizy, jak też zapełnienie pamięci podczas działania programu przykładowymi danymi. Można zmieniać odpowiednie bity rejestrów sterujących i badać zachowanie programu. Dla wygody zawartość wszystkich rejestrów i pamięci można przedstawić w dowolnym formacie: heksadecymalnym, oktalnym, binarnym, stałoprzecinkowym ułamkowym i całkowitym ze znakiem i bez, zmiennoprzecinkowym, czy w końcu w formacie kodu maszynowego procesora. W celu pełniejszego wykorzystania możliwości symulacji można zdefiniować tzw. strumienie. Mówiąc prosto, do pewnej komórki pamięci można przypisać plik wejściowy z pewnymi danymi. Po odczytcie z danej komórki kolejna wartość z pliku zostanie przypisana, symulując strumień danych. To samo jest możliwe w drugą stronę: z pamięci do pliku, czy w końcu z pamięci do pamięci. Do pełni szczęścia można zaprogramować cykliczne wystąpienie przerwania zewnętrznego czy też ustawienie flagi, co umożliwia symulowanie przerwań. Dodatkowo istnieje możliwość zdefiniowania kilku różnych szablonów do pracy i szybkie przełączanie między nimi. Oczywiście żaden, nawet najlepszy symulator, nie obejmie swoim zasięgiem wszystkich zdarzeń możliwych do wystąpienia w rzeczywistym układzie. Dodatkowym mankamentem jest niezwykle wolna praca w stosunku do pracy rzeczywistej.

Z pomocą VisualDSP++ 3.0 można asemblerować, ładować, a następnie debuggować programy z wykorzystaniem płyty ewaluacyjnej EZ-KIT Lite. Wciąż istnieje możliwość podglądania wszystkich odczytywalnych rejestrów czy pamięci. Możliwa jest też praca krokowa czy ustawianie pułapek. Z wykorzystaniem tak poteżnego narzędzia został zaprogramowany wybrany algorytm łącznej redukcji szumów i echa akustycznego, a następnie uruchomiony na płytce EZ-KIT Lite. Zostanie on teraz dokładnie przedstawiony od strony matematycznej.

4. Wybrany algorytm łącznej redukcji szumów i echa akustycznego

W poprzedniej części niniejszej pracy zostały zaprezentowane wyniki symulacji całej gamy różnorakich filtrów adaptacyjnych w celu wybrania najodpowiedniejszego rozwiązania do implementacji z wykorzystaniem procesora sygnałowego. Wskazują one, że najprostsze w uruchomieniu są filtry bazujące na minimalizacji błędu średniokwadratowego pomiędzy sygnałem wyjściowym filtra a sygnałem pożdanym. Mimo tej niewątpliwej zalety wadą tego rozwiązania jest dość powolna zbieżność algorytmu do optymalnych wartości współczynników filtru i tym samym powolny zanik błędu mierzony na wyjściu filtra. Z drugiem strony algorytmy bazujące na kryterium najmniejszych kwadratów wykazywały niezwykle szybką zbieżność, lecz wprowadzały niezwykle duży koszt obliczeń oraz potencjalną niestabilność numeryczną. Najlepszym sposobem wybrania docelowego algorytmu byłoby znalezienie złotego środka łączącego zalety obydwóch podejść i minimalizującego ich wady. Takim rozwiązaniem wydaje się być algorytm ϵ -NLMS wyższego rzędu, tzw. algorytm ϵ -APA. Jak wynikało z symulacji, szybkość zbieżności tego algorytmu nieznacznie ustępowała klasycznemu RLS, i choć złożoność obliczeniowa została wielokrotnie zmniejszona, algorytm ten wciąż wymaga zbyt wielu operacji, by go zaimplementować z pozytywnym skutkiem na zaprezentowanej płytcie uruchomieniowej EZ-KIT Lite. W tym celu należało poszukać dalszych redukcji złożoności i jak się okazało jest to możliwe, przy tzw. szybkim algorytmie ϵ -APA, tzw. FAP [20].

Dla przypomnienia: zbierając K ostatnich regresorów w chwili czasu i do macierzy $U_{K,i}$ oraz K ostatnich próbek sygnału pożdanego do wektora d_i , mamy:

$$(4.1) \quad U_{K,i} = \begin{bmatrix} u_{N,i} \\ u_{N,i-1} \\ \vdots \\ u_{N,i-K+1} \end{bmatrix} = \begin{bmatrix} u(i) & u(i-1) & \dots & u(i-N+1) \\ u(i-1) & u(i-2) & \dots & u(i-N) \\ \vdots & \vdots & \vdots & \vdots \\ u(i-K+1) & u(i-K) & \dots & u(i-K-N+2) \end{bmatrix},$$

$$(4.2) \quad d_i = [d(i), d(i-1), \dots, d(i-K+1)]^T.$$

Można wtedy wyznaczyć wektor błędów „a priori”, czyli obliczony przed odświeżeniem współczynników filtru, w następujący sposób:

$$(4.3) \quad e_i = d_i - U_{K,i}^* w_{i-1}.$$

Z odwrotności macierzy autokorelacji dla K ostatnich próbek sygnału otrzymywaliśmy wektor wzmacnienia:

$$(4.4) \quad g_i = (\epsilon I + U_{K,i} U_{K,i}^*)^{-1} e_i.$$

Wtedy współczynniki filtra były odświeżane, jak następuje:

$$(4.5) \quad w_i = w_{i-1} + \mu U_{K,i}^* g_i.$$

Wektor błędu „a posteriori”, czyli wyznaczony po odświeżeniu współczynników, dany jest wzorem:

$$(4.6) \quad \gamma_i = d_i - U_{K,i}^* w_i.$$

Wstawiając równanie odświeżania współczynników w_i do wzoru na γ_i oraz dokonując dekompozycji macierzy $U_{K,i} U_{K,i}^*$ na iloczyn macierzy unitarnych oraz macierzy diagonalnej zawierającej na swej przekątnej wartości własne $U_{K,i} U_{K,i}^*$, można pokazać, że poszczególne elementy wektora γ_i

są w przybliżeniu równe poszczególnym elementom e_i przeskalowanym przez $1 - \mu$, gdy odpowiadająca im wartość własna jest dużo większa od ϵ . W odwrotnym przypadku adaptacja nie zachodzi, gdyż energia sygnału wejściowego jest zdominowana przez współczynnik regulujący, tak więc przeskalowanie przez $1 - \mu$ może jedynie polepszyć sprawę. Stąd z bardzo dobrym przybliżeniem można przyjąć zależność pomiędzy wektorem błędu „a priori” oraz „a posteriori” jako:

$$(4.7) \quad \gamma_i \approx (1 - \mu)e_i.$$

Warto przypatrzeć się dokładniej wektorowi błędu:

$$(4.8) \quad e_i = d_i - U_{K,i}^* w_{i-1} = \begin{bmatrix} d(i) - U_{K,i} w_{i-1} \\ \tilde{d}_{i-1} - \tilde{U}_{K,i-1} w_{i-1} \end{bmatrix};$$

\tilde{d}_{i-1} oznacza $K - 1$ górnych elementów wektora d_{i-1} , a $\tilde{U}_{K,i-1}$ oznacza $K - 1$ pierwszych wierszy macierzy $U_{K,i-1}$. Widać, że $K - 1$ ostatnich elementów wektora e_i jest zarazem $K - 1$ pierwszymi elementami wektora błędu „a posteriori” (czyli wyznaczonego po odświeżeniu współczynników) z poprzedniej iteracji, skąd na podstawie zależności między błędami wektor „a priori” można odświeżyć na podstawie błędu z poprzedniej iteracji:

$$(4.9) \quad e_i \approx \begin{bmatrix} e(i) \\ (1 - \mu)\tilde{e}_{i-1} \end{bmatrix}.$$

Kolejnym krokiem jest przyjrzenie się równaniu odświeżania współczynników w_i :

$$(4.10) \quad w_i = w_{i-1} + \mu U_{K,i}^* g_i = w_0 + \mu [u_{N,i}^* \ u_{N,i-1}^* \ \dots \ u_{N,i-p+1}^*] \begin{bmatrix} g_{i,0} \\ g_{i,1} \\ \vdots \\ g_{i,p} \end{bmatrix} + \mu [u_{N,i-1}^* \ u_{N,i-2}^* \ \dots \ u_{N,i-p}^*] \begin{bmatrix} g_{i-1,0} \\ g_{i-1,1} \\ \vdots \\ g_{i-1,p} \end{bmatrix} + \dots = \mu \underbrace{u_{N,i}^* g_{i,0} + u_{N,i-1}^* (g_{i,1} + g_{i-1,0}) + \dots + u_{N,i-p+1}^* (g_{i,p} + \dots + g_{i-p+1,0})}_{U_{K,i}^* G_i} + \underbrace{u_{N,i-p}^* (g_{i-1,p} + \dots + g_{i-p,0}) + \dots}_{\hat{w}_{i-1}}$$

Z powyższego rozwinięcia wynika, że współczynniki filtra można odświeżyć w oparciu o tzw. zastępczy wektor współczynników \hat{w}_i :

$$(4.11) \quad \begin{cases} w_i = \hat{w}_{i-1} + \mu U_{K,i}^* G_i \\ \hat{w}_i = \hat{w}_{i-1} + \mu u_{N,i-p+1}^* G_{i,p-1} \end{cases} \implies w_i = \hat{w}_i + \mu \tilde{U}_{K,i}^* \tilde{G}_i,$$

gdzie \tilde{X} oznacza $K - 1$ pierwszych wierszy macierzy X . Wektor G_i można łatwo wyznaczyć w sposób rekursywny:

$$(4.12) \quad G_i = \begin{bmatrix} g_{i,0} \\ g_{i,1} + g_{i-1,0} \\ \vdots \\ g_{i,p} + \dots + g_{i-p+1,0} \end{bmatrix},$$

$$(4.13) \quad G_i = g_i + \begin{bmatrix} 0 \\ \tilde{G}_{i-1} \end{bmatrix}.$$

Aby odświeżyć wektor współczynników błędu, trzeba znać błąd „a priori” estymacji w chwili czasu i . Można go również wyznaczyć w oparciu o zastępczy wektor współczynników:

$$(4.14) \quad e(i) = d(i) - u_{N,i}w_{N,i-1} = d(i) - u_{N,i} \left(\hat{w}_{i-1} + \mu \tilde{U}_{K,i-1}^* \tilde{G}_{i-1} \right) = \hat{e}(i) - \underbrace{\mu u_{N,i} \tilde{U}_{K,i-1}^*}_{r_{uu,i}^*} \tilde{G}_{i-1}.$$

Wektor autokorelacji $r_{uu,i}$ można obliczyć iteracyjnie ze znanych próbek wejściowych filtra:

$$(4.15) \quad r_{uu,i} = r_{uu,i-1} + u(i)u_{p-1,i-1}^* - u(i-N)u_{p-1,i-N-1}^*.$$

Z dotychczasowych równań wynika, że aby otrzymać prawidłowy sygnał na wyjściu filtra, nie ma wcale potrzeby odświeżać oryginalnych współczynników w_i , a wystarczy odnowić zastępczy wektor współczynników \hat{w}_i . Ma miejsce znaczna redukcja nakładu obliczeniowego, gdyż do odświeżenia w_i potrzeba dokonać N iloczynów skalarnych o długości K , do odświeżenia \hat{w}_i wystarczy zaś N mnożeń. Wszystkie dotychczasowe dywagacje opierają się na założeniu znajomości wektora wzmacniania g_i . Znalezienie zależności na rekursywne jego obliczenie z iteracji do iteracji jest ostatnim krokiem w skonstruowaniu szybkiego algorytmu FAP.

Definicja wzmacniania g_i jest następująca:

$$(4.16) \quad g_{K,i} = (\epsilon I + U_{KN,i}U_{KN,i}^*)^{-1} e_i.$$

Warto sobie przypomnieć definicję wzmacniania stosowaną w algorytmie RLS:

$$(4.17) \quad g_{N,i} = (\epsilon I + H_{NM,i}H_{NM,i}^*)^{-1} e_i.$$

Z dokonanego porównania wynika, że wektorowi wzmacniania algorytmu FAP o N współczynnikach filtra i rzędowi projekcji równym K odpowiada wektor wzmacniania algorytmu RLS o K współczynnikach filtra wyznaczony na podstawie ostatnich N zmierzonych próbek błędu! Można zatem zaprzegnąć całą maszynerię RLS do iteracyjnego odświeżania wektora $g_{K,i}$. Z teorii najmniejszych kwadratów wynika, że macierz autokorelacji można odświeżyć w czasie w oparciu o współczynniki predykcji w przód i w tył:

$$(4.18) \quad P_{K,i} = \begin{bmatrix} P_{K-1,i} & 0 \\ 0 & 0 \end{bmatrix} + \zeta_{K-1}^{-b}(i) \begin{bmatrix} -w_{K-1,i}^b \\ 1 \end{bmatrix} \begin{bmatrix} -w_{K-1,i}^{b*} & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & P_{K-1,i-1} \end{bmatrix} + \zeta_{K-1}^{-f}(i) \begin{bmatrix} 1 \\ -w_{K-1,i}^f \end{bmatrix} \begin{bmatrix} 1 & -w_{K-1,i}^{f*} \end{bmatrix}.$$

Mnożąc przez wektor błędu „a priori”, otrzymujemy:

$$(4.19) \quad g_{K,i} = \begin{bmatrix} \tilde{g}_{K,i} \\ 0 \end{bmatrix} + B_i = \begin{bmatrix} 0 \\ \tilde{g}_{K,i-1} \end{bmatrix} + F_i,$$

gdzie wektory B_i, F_i są równe:

$$(4.20) \quad B_i = \zeta_{K-1}^{-b}(i) \begin{bmatrix} -w_{K-1,i}^b \\ 1 \end{bmatrix} \begin{bmatrix} -w_{K-1,i}^{b*} & 1 \end{bmatrix} e_i,$$

$$(4.21) \quad F_i = \zeta_{K-1}^{-f}(i) \begin{bmatrix} 1 \\ -w_{K-1,i}^f \end{bmatrix} \begin{bmatrix} 1 & -w_{K-1,i}^{f*} \end{bmatrix} e_i;$$

$\zeta_{K-1}^b(i)$ oraz $\zeta_{K-1}^f(i)$ oznaczają moce błędów predykcji w przód oraz w tył. Wektor wzmacnienia $g_{K,i}$ jest wyznaczany w dwóch krokach. Najpierw jest odświeżany w czasie za pomocą wektora wzmacnienia niższego rzędu z poprzedniej iteracji, następnie jego rzząd jest dekrementowany by mógł posłużyć do odświeżenia w czasie w następnej iteracji. Odnowienie $g_{K,i}$ kończy konstrukcję algorytmu FAP:

$$(4.22) \quad g_{K,i} = \begin{bmatrix} 0 \\ (1-\mu)g_{K-1,i} \end{bmatrix} + F_i,$$

$$(4.23) \quad g_{K-1,i} = g_{K,i}(1 : K-1) - B_i(1 : K-1).$$

Porównując złożoność algorytmu FAP, który gdyby uwzględnić tylko wielokrotności rzędu filtra N , ma złożoność $4N$, z oryginalnym algorytmem ϵ -APA o złożoności $4KN$, widać redukcję K -krotną wymaganych operacji mnożenia. Jak już było omawiane, problem echa akustycznego jest specyficzny z tego powodu, że wymaga bardzo dużej liczby współczynników N . W pierwszej części zostało również zaprezentowane, jak algorytmy blokowe – wprowadzając niewielkie opóźnienie – potrafią zmniejszyć koszty obliczeń o kolejny rzząd wartości. Podobną sztukę można wykonać z algorytmem FAP, implementując go w formie blokowej [48].

Założymy, że długość bloku wynosi L . Aby odświeżyć współczynniki od chwili czasu $i-L$ do aktualnej chwili czasu i , wykorzystujemy równanie algorytmu ϵ -APA. Czytając podobne rozwinięcie jak przy wyprowadzeniu FAP, mamy:

$$(4.24) \quad \begin{aligned} w_i &= w_{i-L} + \mu U_{NK,i-L+1}^* g_{K,i-L+1} + \dots + \mu U_{NK,i}^* g_{K,i} = \\ &= w_{i-L} - \underbrace{\sum_{j=0}^{K-2} u_{N,i-L-j}^* S_{K-1,i-L,j}}_{z_{i-L}} + \sum_{j=0}^{L+K-2} u_{N,i-j}^* S_{K-1,i,j} = \\ &= w_{i-L} - \underbrace{U_{N(K-1),i-L} S_{K-1,i-L}}_{z_{i-L}} + U_{N(L+K-1),i} S_{L+K-1,i}. \end{aligned}$$

Wektor $S_{L+K-1,i}$ zawiera podobne dane jak wektor $G_{K-1,i}$ w przypadku FAP:

$$(4.25) \quad S_{L+K-1,i} = \begin{bmatrix} g_{i,0} \\ g_{i-1,0} + g_{i,1} \\ \vdots \\ g_{i-K+1,0} + \dots + g_{i,K-1} \\ \vdots \\ g_{i-L-K+2,0} + \dots + g_{i-L+1,K-1} \end{bmatrix}.$$

Wektor współczynników z_i spełnia podobną rolę wektora rozszerzonych współczynników filtra, tym razem dla BEFAP. Rozwiązujejąc poniższy układ równań w którym uwzględnia się równanie odświeżania współczynników w_i

$$(4.26) \quad \begin{cases} z_{i-L} = w_{i-L} - U_{N(K-1),i-L} S_{K-1,i-L}, \\ z_i = w_i - U_{N(K-1),i} S_{K-1,i} \end{cases}$$

dostajemy równanie odświeżania rozszerzonych współczynników filtra:

$$(4.27) \quad z_i = z_{i-L} + U_{NL,i-p+1} S_{L,i}.$$

Mnożenie macierzy w powyższym wzorze to nic innego, jak równanie filtarcji sygnału wejściowego $[u(i), u(i-1), \dots]$ przez filtr o stałych współczynnikach $S_{L,i}$. Jak wiadomo, filtrację cyfrową najefektywniej obliczać, korzystając z FFT!

Do wyznaczenia pozostaje jeszcze odpowiedni sygnał na wyjściu filtru. Po skompletowaniu całego bloku danych wejściowych począwszy od chwili $i-L+1$ do i , próbka wyjściowa w chwili $i-L+m$ powinna mieć następującą wartość:

$$(4.28) \quad y(i-L+m) = u_{N,i-L+m} w_{i-L+m-1} = \underbrace{u_{N,i-L+m} w_{i-L}}_{\text{bez korekcji}} + \underbrace{u_{N,i-L+m} U_{N(m+K-2),i} S_{m+K-2,i}}_{\substack{r_{uu,i-L+m} \\ \text{człon korekcyjny}}}.$$

Wektor autokorelacji można wyznaczyć rekurencyjnie podobnie jak przy algorytmie FAP. Ostatecznie blokowy algorytm FAP z korekcją sygnału wyjściowego, tzw. BEFAP, przedstawiony jest jako algorytm 1.

Ostatnim etapem przygotowania algorytmu eliminacji echa akustycznego jest wyznaczenie współczynników predykcji w przód i tył dla odświeżenia wektora wzmacnienia $g_{K,i}$. Można to zrobić w oparciu o algorytm najmniejszych kwadratów. Można zadać sobie pytanie, w jakim celu wykorzystywać algorytm rodziny RLS, jeśli wiadomo, że jego złożoność obliczeniowa jest o rząd większa od podstawowego ϵ -APA. Jest to prawda, gdy wyznaczamy współczynniki predykcji rzędu N , gdzie N oznacza dużą ilość współczynników filtru. Algorytm FAP wymaga jednak wyznaczenia współczynników predykcji rzędu K , co jest zdecydowanie mniejszą liczbą i z reguły nie przekracza 16, gdyż przy większych jego wartościach algorytm ϵ -APA nie wykazuje już przyspieszenia szybkości zbieżności, a powiększa się tylko koszt obliczeń. Drugą istotną sprawą jest wymaganie zastosowania algorytmu RLS z tzw. chodzącym oknem. W przypadku standardowego RLS wprowadza się współczynnik zapominania α . Optymalne współczynniki filtru są wyznaczane w oparciu o wszystkie zmierzonych poprzednio błędy przeskalowane przez kolejne wzrastające potęgi α . W ten sposób coraz starsze próbki błędu będą wnosić coraz mniej informacji do aktualnych współczynników. Z kolei BEFAP wymaga wyznaczenia wsólczynników filtru na podstawie dokładnie N ostatnich próbek błędu. Aby tego dokonać, trzeba oprócz odświeżania współczynników wprowadzić tzw. deaktualizację współczynników [43, 15].

Przypuśćmy, że skompletowaliśmy $L+1$ ostatnich wartości sygnału pożdanego filtru oraz $L+1$ ostatnich regresorów w poniższych macierzach:

$$(4.29) \quad d_{i-L:i} = \begin{bmatrix} d(i-L) \\ d(i-L+1) \\ \vdots \\ d(i) \end{bmatrix},$$

$$(4.30) \quad H_{i-L:i} = \begin{bmatrix} u_{i-L} \\ u_{i-L+1} \\ \vdots \\ u_i \end{bmatrix}.$$

Wtedy optymalne współczynniki minimalizujące kryterium najmniejszych kwadratów bazujące na $L+1$ oraz L ostatnich próbkach błędu, będą następujące:

$$(4.31) \quad w_{i-L:i} = (\Pi + H_{i-L:i}^* H_{i-L:i})^{-1} H_{i-L:i}^* d_{i-L:i},$$

$$(4.32) \quad w_{i-L+1:i} = (\Pi + H_{i-L+1:i}^* H_{i-L+1:i})^{-1} H_{i-L+1:i}^* d_{i-L+1:i}.$$

Inicjalizacja:

$$\begin{aligned} z_0 &= 0_N & r_{uu,0} &= 0_{L+p-2} & S_{L+K-1,0} &= 0_{L+K-1} \\ e_0 &= 0_K & g_{K-1,0} &= 0_{K-1} \end{aligned}$$

Dla wszystkich i :

```

if  $i \bmod L = 0$ :
     $u_f = \mathcal{F}\{[0 \quad \text{fliplr}(u_{N+L-1,i}^T)]\}$ 
     $z_f = \mathcal{F}\{[z_{i-L} \quad 0_L]\}$ 
     $Y_f = u_f \odot z_f$ 
     $Y = \text{flipud}(\mathcal{F}^{-1}\{Y_f\}(N+1:N+L))$ 
    for  $m = i - L + 1 : i$ ;
         $r_{uu,m} = r_{uu,m-1} + u(m)u_{L+K-2,m-1}^* - u(m-N)u_{L+K-2,m-1-N}^*$ 
         $j = m \bmod L + (m \bmod L)\delta(L)$ 
         $\hat{e}(r) = d(r) - Y(L-j+1)$ 
         $e(r) = \hat{e}(r) - S_{L+K-1,m-1}^*(1:j+K-2)r_{uu,m}(1:j+K-2)$ 
         $e_m = \begin{bmatrix} e(m) \\ (1-\mu)\tilde{e}_{m-1} \end{bmatrix}$ 
        SWRLS  $\implies B_i, F_i$ 
         $g_{K,m} = \begin{bmatrix} 0 \\ (1-\mu)g_{K-1,m-1} \end{bmatrix}$ 
         $g_{K-1,m} = g_{K,m}(1:K-1) - B_i(1:K-1)$ 
         $S_{L+K-1,m} = \begin{bmatrix} 0 \\ S_{L+K-2,m-1} \end{bmatrix} + \begin{bmatrix} \mu g_{K,m} \\ 0_{L-1} \end{bmatrix}$ 
    endfor;
     $v_f = \mathcal{F}\{[0 \quad \text{fliplr}(u_{N+L-1,i-K+1}^T)]\}$ 
     $g_f = \mathcal{F}\{[S_{L,i}(K:K+L-1) \quad 0_N]\}$ 
     $h_f = v_f \odot g_f$ 
     $h = \mathcal{F}^{-1}\{h_f\}(L+1:L+N)$ 
     $z_i = z_{i-L} + \text{flipud}(h)$ 
endif

```

fliplr – odbicie lustrzane wartości wektora poziomego;

flipud – odbicie lustrzane wartości wektora pionowego.

ALGORYTM 1. BEFAP – blokowy szybki ϵ -APA z korekcją sygnału wyjściowego

Powtarzając wszystkie kroki jak przy wyprowadzeniu algorytmu RLS, dochodzi się do następującej zależności:

$$(4.33) \quad w_{i-L+1:i} = w_{i-L:i} - \frac{P_{i-L:i} u_{i-L}^*}{1 - u_{i-L} P_{i-L:i} u_{i-L}^*} \left[\underbrace{d(i-L) - u_{i-L} w_{i-L:i}}_{\epsilon(i-L)} \right];$$

$\epsilon(i-L)$ oznacza błąd „a priori” deaktualizacji współczynników, a współczynnik konwersji wynosi:

$$(4.34) \quad \gamma(i-L) = \frac{1}{1 - u_{i-L} P_{i-L:i} u_{i-L}^*}.$$

W pierwszej części niniejszej pracy zostało również powiedziane, że idea szybkich algorytmów RLS opiera się na propagacji w czasie różnicy macierzy autokorelacji. Więcej, różnica ta jest równa przeskalowanym współczynnikom predykcji w przód i w tył sygnału wejściowego. Wykorzystanie tej idei prowadzi do tzw. szybkiego macierzowego algorytmu RLS. Wprowadzając dodatkowo deaktualizację współczynników, otrzymujemy szybki macierzowy algorytm RLS z chodzącym oknem, zaprezentowany jako algorytm 2.

Indeksy przy zapisie $w_{N,i}^{f(L)}$ oznaczają, że mamy do czynienia z wektorem współczynników predykcji w przód, który został wyznaczony na podstawie L ostatnich próbek sygnału błędu, począwszy od czasu i . Przy odświeżaniu bądź deaktualizacji należy dokonać przekształcenia pierwotnej macierzy poprzez macierz transformacji Θ w ten sposób, aby otrzymać 0 na zaznaczonych pozycjach oraz żeby macierz Θ spełniała warunek: $\Theta^* J \Theta = J$. Macierz J to macierz sygnaturowa zaznaczona przy opisie algorytmu.

Niestety zaprezentowany algorytm jest niestabilny numerycznie przy implementacji w pojedynczej precyzyji. Główną tego przyczynę stanowi pętla sprzężenia zwrotnego zawiązana na linii „wyznaczanie błędu – odświeżanie współczynników”. I tak, ażeby np. wyznaczyć błąd predykcji w tył $\beta_N(i)$, musimy wykonać iloczyn skalarny aktualnego regresora $u_{N,i}$ z wektorem $w_{N,i}^{b(L)}$. Wymaga to $N - 1$ dodawań liczb zmienoprzecinkowych, a to właśnie dodawanie jest operacją, która wnosi największą niedokładność do obliczeń. Aby dodać dwie liczby różniące się wykładnikami, muszą one zostać wyrównane. Podczas tej operacji najmniej znaczące bity mantysy mniejszej liczby są tracone bezpowrotnie. Na podstawie tak zaburzonej wartości sygnału błędu są odświeżane współczynniki, co wprowadza dodatkowy błąd. Następnie współczynniki z większą niedokładnością znów są używane do wyznaczenia błędu predykcji i koło się zamyka. Gdyby dało się wyznaczyć $\beta_N(i)$ w sposób niezależny, znacznie wzrosłaby jakość obliczeń. Można to zrobić, stosując tzw. filtry kratowe [43, 58].

Inicjalizacja:

$$M_{u,0} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & \eta^{-\frac{1}{2}} & 0 \\ 0 & 0 & \eta^{-\frac{1}{2}} \end{bmatrix}$$

$$\gamma_N^{-\frac{1}{2}(L-1)}(0) = 1$$

$$g_{N,0}^{L-1} = 0_N$$

Dla wszystkich i :

Odświeżanie współczynników:

$$M_{u,i} = \left[\begin{array}{ccc} \gamma_N^{-\frac{1}{2}L}(i-1) & \alpha_N(i)\zeta_N^{-\frac{f}{2}(L-1)}(i-1) & \beta_N(i)\zeta_N^{-\frac{b}{2}(L-1)}(i-1) \\ \begin{bmatrix} 0 \\ g_{N,i-1}^L \gamma_N^{-\frac{1}{2}L}(i-1) \end{bmatrix} & \zeta_N^{-\frac{f}{2}(L-1)}(i-1) \begin{bmatrix} 1 \\ -w_{N,i-1}^{f(L-1)} \end{bmatrix} & \zeta_N^{-\frac{b}{2}(L-1)}(i-1) \begin{bmatrix} -w_{N,i-1}^{b(L-1)} \\ 1 \end{bmatrix} \end{array} \right] \downarrow \Theta \begin{bmatrix} 1 & & \\ & 1 & \\ & & -1 \end{bmatrix}$$

$$\left[\begin{array}{ccc} \gamma_N^{-\frac{1}{2}L}(i) & 0 & 0 \\ \begin{bmatrix} g_{N,i}^L \gamma_N^{-\frac{1}{2}L}(i) \\ 0 \end{bmatrix} & \zeta_N^{-\frac{f}{2}(L)}(i) \begin{bmatrix} 1 \\ -w_{N,i}^{f(L)} \end{bmatrix} & \zeta_N^{-\frac{b}{2}(L)}(i) \begin{bmatrix} -w_{N,i}^{b(L)} \\ 1 \end{bmatrix} \end{array} \right]$$

Deaktualizacja współczynników:

$$M_{d,i} = \left[\begin{array}{ccc} \gamma_N^{-\frac{1}{2}(L-1)}(i-1) & \acute{\alpha}_N(i-L+1)\zeta_N^{-\frac{f}{2}(L)}(i) & \acute{\beta}_N(i-L+1)\zeta_N^{-\frac{b}{2}(L)}(i) \\ \begin{bmatrix} 0 \\ g_{N,i-1}^{L-1} \gamma_N^{-\frac{1}{2}(L-1)}(i-1) \end{bmatrix} & -\zeta_N^{-\frac{f}{2}(L)}(i) \begin{bmatrix} 1 \\ -w_{N,i}^{fL} \end{bmatrix} & -\zeta_N^{-\frac{b}{2}(L)}(i) \begin{bmatrix} -w_{N,i}^{bL} \\ 1 \end{bmatrix} \end{array} \right] \downarrow \Theta \begin{bmatrix} 1 & & \\ & -1 & \\ & & 1 \end{bmatrix}$$

$$\left[\begin{array}{ccc} \gamma_N^{-\frac{1}{2}(L-1)}(i) & 0 & 0 \\ \begin{bmatrix} g_{N,i}^{L-1} \gamma_N^{-\frac{1}{2}(L-1)}(i) \\ 0 \end{bmatrix} & -\zeta_N^{-\frac{f}{2}(L-1)}(i) \begin{bmatrix} 1 \\ -w_{N,i}^{f(L-1)} \end{bmatrix} & -\zeta_N^{-\frac{b}{2}(L-1)}(i) \begin{bmatrix} -w_{N,i}^{b(L-1)} \\ 1 \end{bmatrix} \end{array} \right]$$

ALGORYTM 2. SWFARLS – szybki macierzowy algorytm RLS z chodzącym oknem

Idea filtrów kratowych polega na tym, że nie propaguje się w ogóle współczynników filtrów, lecz jedynie same błędy. Algorytmy te opierają się na ciągłym powiększaniu rzędu analizowanego problemu, aż do osiągnięcia wymaganej wartości. I tak np. błąd „a posteriori” predykcji w przód wyznaczony dla filtra o N współczynnikach jest dany wzorem:

$$(4.35) \quad f_{N,i} = u(i) - u_{N,i-1} w_{N,i}^f.$$

Przyjmując, że $f_{0,i}$ jest równe $u(i)$, można wyprowadzić zależność na $f_{1,i}$. Tak – propagując wartość f – dochodzimy do wymaganej $f_{N,i}$. Autor pracy zaprogramował i przetestował kilka odmian filtrów kratowych z chodzącym oknem pod kątem stabilności w środowisku MATLAB. Te odmiany to:

- filtr kratowy bazujący na propagacji błędów „a posteriori”;
- filtr kratowy bazujący na propagacji błędów „a priori”;
- filtr kratowy bazujący na propagacji błędów „a posteriori” ze sprzężeniem zwrotnym;
- filtr kratowy bazujący na propagacji błędów „a priori” ze sprzężeniem zwrotnym;
- filtr kratowy bazujący na rozkładzie QRD.

Dbając o to, by objetość niniejszej pracy nie zaczęła rosnąć nieograniczenie, autor zdecydował się pominąć szczegółowe wyprowadzenie i omówienie przetestowanych algorytmów, ograniczając się do sformułowania wniosków mających związek z implementowanym algorytmem BEFAP. Otóż wszystkie algorytmy zostały uruchomione w środowisku obliczeniowym pojedynczej precyzji, gdyż z taką dokładnością pracuje procesor sygnałowy Sharc. Można oczywiście uruchomić algorytm stosując 40-bitowe liczby zmiennoprzecinkowe na wspomnianym procesorze, i taka rzeczywiście była pierwsza próba z algorytmem SWFARLS, jednak po pierwsze: znacznie ograniczona zostaje wtedy pamięć procesora, gdyż trzeba skonfigurować oba bloki pamięci po 48 bitów, więc zapisując każdą liczbę, automatycznie tracimy 8 bitów pamięci. Co więcej, każdy z bloków pamięci podzielony jest na 16-bitowe kolumny, których jest *de facto* osiem. Każda kolumna posiada 4096 słów do wykorzystania. Konfigurując blok pierwszy jako 48-bitowy, zajmujemy sześć kolumn i tracimy automatycznie możliwość użycia pozostałych 128kb pamięci (nie możemy użyć pozostałych dwóch kolumn na słowa 32-bitowe, gdyż muszą one zaczynać się od parzystego numeru kolumny). Widać to w tabeli 4, gdzie przedstawiona została mapa pamięci płyty EZ-KIT. Niestety monitor wymaga skonfigurowania bloku 1 na długość 48 bitów. Używając słów 32-bitowych w bloku drugim, wykorzystujemy po dwie kolumny, co daje razem 16384 słowa, czyli dwa razy więcej, i cały blok 2. Po drugie i ważniejsze, podniesienie dokładności obliczeń do 40 bitów w standardowym SWFARLS wcale nie doprowadziło do stabilności algorytmu. Każdy z algorytmów kratowych uruchamiany był kilkakrotnie z sygnałem wejściowym zawierającym kilka milionów próbek odfiltrowanego dolnoprzepustowo szumu białego. Długość okna ustalono na $N = 896$. Symulacje wykazały, że jedynie algorytm bazujący na rozkładzie QRD nie wykazuje żadnych przejawów niestabilności. Dlatego też został on wybrany do implementacji. Jest on przedstawiony poniżej.

Inicjalizacja:

$$\hat{w}_{K,0}^{f(L-1)} = [\eta^{-\frac{1}{2}} \quad 0 \quad \dots \quad 0] \quad \hat{w}_{K,0}^{b(L-1)} = [0 \quad \dots \quad 0 \quad \eta^{-\frac{1}{2}}] \quad g_{N,0}^L = [0 \quad 0 \quad \dots \quad 0] \quad g_{N,0}^{L-1} = g_{N,0}^L$$

for m=0:K;

$$\zeta_m^{\frac{f}{2}(L-1)}(0) = \zeta_m^{\frac{b}{2}(L-1)}(-1) = \sqrt{\eta} \quad \rho_m^{b(L-1)}(0) = \rho_m^{f(L-1)}(0) = 0$$

$$\acute{b}_m(-1) = b_m(-1) = 0 \quad \acute{\gamma}_m(0) = \gamma_m(0) = 1$$

endfor;

Dla wszystkich $i > 0$:

$$f_0(i) = b_0(i) = u(i) \quad \acute{f}_0(i - L + 1) = \acute{b}_0(i - L + 1) = u(i - L + 1) \quad \gamma_0(i) = \acute{\gamma}_0(i - L + 1) = 1$$

for m=0:K-1;

Predyktor kratowy

$$\zeta_m^{\frac{b}{2}(L)}(i-1) = \sqrt{\left[\zeta_m^{\frac{b}{2}(L-1)}(i-2)\right]^2 + b_m(i-1)^2}$$

$$c_m^b = \frac{\zeta_m^{\frac{b}{2}(L-1)}(i-2)}{\zeta_m^{\frac{b}{2}(L)}(i-1)} \quad s_m^b = \frac{b_m(i-1)}{\zeta_m^{\frac{b}{2}(L)}(i-1)}$$

$$\rho_m^{f(L)}(i) = \rho_m^{f(L-1)}(i-1)c_m^b + f_m(i)s_m^b$$

$$f_{m+1}(i) = f_m(i)c_m^b - \rho_m^{f(L-1)}(i-1)s_m^b$$

$$\zeta_m^{\frac{b}{2}(L-1)}(i-1) = \sqrt{\left[\zeta_m^{\frac{b}{2}(L)}(i-1)\right]^2 - \acute{b}_m(i-L)^2}$$

$$ch_m^b = \frac{\zeta_m^{\frac{b}{2}(L)}(i-1)}{\zeta_m^{\frac{b}{2}(L-1)}(i-1)} \quad sh_m^b = \frac{\acute{b}_m(i-L)}{\zeta_m^{\frac{b}{2}(L-1)}(i-1)}$$

$$\rho_m^{f(L-1)}(i) = \rho_m^{f(L)}(i)ch_m^b - \acute{f}_m(i-L+1)sh_m^b$$

$$\acute{f}_{m+1}(i-L+1) = \acute{f}_m(i-L+1)ch_m^b - \rho_m^{f(L)}(i)sh_m^b$$

$$\zeta_m^{\frac{f}{2}(L)}(i) = \sqrt{\left[\zeta_m^{\frac{f}{2}(L-1)}(i-1)\right]^2 + f_m(i)^2}$$

$$c_m^f = \frac{\zeta_m^{\frac{f}{2}(L-1)}(i-1)}{\zeta_m^{\frac{f}{2}(L)}(i)} \quad s_m^f = \frac{f_m(i)^2}{\zeta_m^{\frac{f}{2}(L)}(i)}$$

$$\rho_m^{b(L)}(i) = \rho_m^{b(L-1)}(i-1)c_m^f + b_m(i-1)s_m^f$$

$$b_{m+1}(i) = b_m(i-1)c_m^f - \rho_m^{b(L-1)}(i-1)s_m^f$$

$$\acute{\gamma}_{m+1}(i) = \acute{\gamma}_m(i-1)c_m^f$$

$$\zeta_m^{\frac{f}{2}(L-1)}(i) = \sqrt{\left[\zeta_m^{\frac{f}{2}(L)}(i)\right]^2 - \acute{f}_m(i-L+1)^2}$$

$$ch_m^f = \frac{\zeta_m^{\frac{f}{2}(L)}(i)}{\zeta_m^{\frac{f}{2}(L-1)}(i)} \quad sh_m^f = \frac{\acute{f}_m(i-L+1)}{\zeta_m^{\frac{f}{2}(L-1)}(i)}$$

$$\rho_m^{b(L-1)}(i) = \rho_m^{b(L)}(i)ch_m^f - \acute{b}_m(i-L)sh_m^f$$

$$\acute{b}_{m+1}(i-L+1) = \acute{b}_m(i-L)ch_m^f - \rho_m^{b(L)}(i)sh_m^f$$

$$\acute{\gamma}_{m+1}(i-L+1) = \acute{\gamma}_m(i-L)ch_m^f$$

endfor;

Wyznaczenie współczynników

$$\begin{aligned}
 \zeta_{K-1}^{\frac{b}{2}(L)}(i) &= \sqrt{\left[\zeta_{K-1}^{\frac{b}{2}(L-1)}(i-1)\right]^2 + b_m(i)^2} \\
 \zeta_{K-1}^{\frac{b}{2}(L-1)}(i) &= \sqrt{\left[\zeta_{K-1}^{\frac{b}{2}(L)}(i)\right]^2 - \dot{b}_m(i-L+1)^2} \\
 \hat{w}_{K,i}^{f(L)} &= \hat{w}_{K,i-1}^{f(L-1)} c_{K-1}^f - \begin{bmatrix} 0 \\ g_{K-1,i-1}^L \end{bmatrix} s_{K-1}^f \\
 g_{K,i}^L &= \begin{bmatrix} 0 \\ g_{K-1,i-1}^L \end{bmatrix} c_{K-1}^f + \hat{w}_{K,i-1}^{f(L-1)} s_{K-1}^f \\
 \hat{w}_{K,i}^{b(L)} &= \hat{w}_{K,i-1}^{b(L-1)} \frac{\zeta_{K-1}^{\frac{b}{2}(L)}(i)}{\zeta_{K-1}^{\frac{b}{2}(L-1)}(i-1)} - g_{K,i}^L \frac{b_{K-1}(i)}{\zeta_{K-1}^{\frac{b}{2}(L-1)}(i-1)} \\
 g_{K-1,i}^L &= g_{K,i}^L (1 : K-1) \frac{\zeta_{K-1}^{\frac{b}{2}(L)}(i)}{\zeta_{K-1}^{\frac{b}{2}(L-1)}(i-1)} - \hat{w}_{K,i-1}^{b(L-1)} \frac{b_{K-1}(i)}{\zeta_{K-1}^{\frac{b}{2}(L-1)}(i-1)} \\
 \hat{w}_{K,i}^{f(L-1)} &= \hat{w}_{K,i}^{f(L)} ch_{K-1}^f + \begin{bmatrix} 0 \\ g_{K-1,i-1}^{L-1} \end{bmatrix} sh_{K-1}^f \\
 g_{K,i}^{L-1} &= \begin{bmatrix} 0 \\ g_{K-1,i-1}^{L-1} \end{bmatrix} ch_{K-1}^f + \hat{w}_{K,i}^{f(L)} sh_{K-1}^f \\
 \hat{w}_{K,i}^{b(L-1)} &= \hat{w}_{K,i}^{b(L)} \frac{\zeta_{K-1}^{\frac{b}{2}(L-1)}(i)}{\zeta_{K-1}^{\frac{b}{2}(L)}(i)} + g_{K,i}^{L-1} \frac{\dot{b}_{K-1}(i-L+1)}{\zeta_{K-1}^{\frac{b}{2}(L)}(i)} \\
 g_{K-1,i}^{L-1} &= g_{K,i}^{L-1} (1 : K-1) \frac{\zeta_{K-1}^{\frac{b}{2}(L-1)}(i)}{\zeta_{K-1}^{\frac{b}{2}(L)}(i)} - \hat{w}_{K,i}^{b(L)} \frac{\dot{b}_{K-1}(i-L+1)}{\zeta_{K-1}^{\frac{b}{2}(L)}(i)}
 \end{aligned}$$

Koniec algorytmu

Wyznaczone współczynniki mają następujące znaczenie:

$$\hat{w}_{K,i}^{f(L)} = \begin{bmatrix} 1 \\ -w_{K-1,i}^{fL} \end{bmatrix} \zeta_{K-1}^{-\frac{f}{2}(L)}(i) \quad \hat{w}_{K,i}^{b(L)} = \begin{bmatrix} -w_{K-1,i}^{bL} \\ 1 \end{bmatrix} \zeta_{K-1}^{-\frac{b}{2}(L)}(i)$$

Dane wymagane przez procedurę BEFAP:

$$B_i = \hat{w}_{K,i}^{bL*} \hat{w}_{K,i}^{bL} e_i \quad F_i = \hat{w}_{K,i}^{fL*} \hat{w}_{K,i}^{fL} e_i$$

ALGORYTM 3. Szybki macierzowy algorytm RLS z chodzącym oknem z błędami predykcji wyznaczonymi za pomocą filtru kratowego z rozkładem QRD

Zaprezentowany algorytm zaimplementowano i wraz z algorytmem redukcji szumów LSA łączono na płycie EZ-KIT Lite, co zostanie teraz dokładnie opisane.

5. Implementacja algorytmu łącznej redukcji szumów i echa akustycznego w języku asemblera procesora sygnałowego

Kod źródłowy zaprogramowanej aplikacji składa się z następujących plików:

Pliki źródłowe:

- „befap.asm” – zawiera kod algorytmu BEFAP; zdefiniowano dwa adresy symboliczne **befap** oraz **befap_init** do wywołania jako podprogramy;
- „cner.asm” – podstawowy plik aplikacji, zawiera program główny zaczynający się od adresu **main**, jak również programy obsługi przerwań;
- „enlms.asm” – zawiera kod algorytmu ϵ -NLMS; zdefiniowano dwa adresy symboliczne **enlms** oraz **enlms_init** do wywołania jako podprogramy;
- „init.asm” – plik inicjalizacyjny procesora sygnałowego; zawiera definicję wszystkich wektorów przerwań;
- „init1847.asm” – plik inicjalizacyjny kodeka **AD1847**;
- „irfftrad2.asm” – kod źródłowy procedury odwrotnej dyskretnej transformaty Fouriera przeznaczonej do pracy z wyjściowymi sygnałami rzeczywistymi;
- „lsa.asm” – zawiera kod algorytmu redukcji szumów LSA; zdefiniowano dwa adresy symboliczne **lsa** oraz **lsa_init** do wywołania jako podprogramy;
- „rfftrad2.asm” – kod źródłowy procedury dyskretnej transformaty Fouriera przeznaczonej do pracy z wejściowymi sygnałami rzeczywistymi;
- „routines.asm” – plik zawierający kod wielu przydatnych funkcji pomocniczych;
- „swflqrd.asm” – kod źródłowy szybkiego macierzowego algorytmu RLS z wykorzystaniem algorytmu kratowego z rozkładem QRD do wyznaczenia błędów adaptacji.

Pliki nagłówkowe:

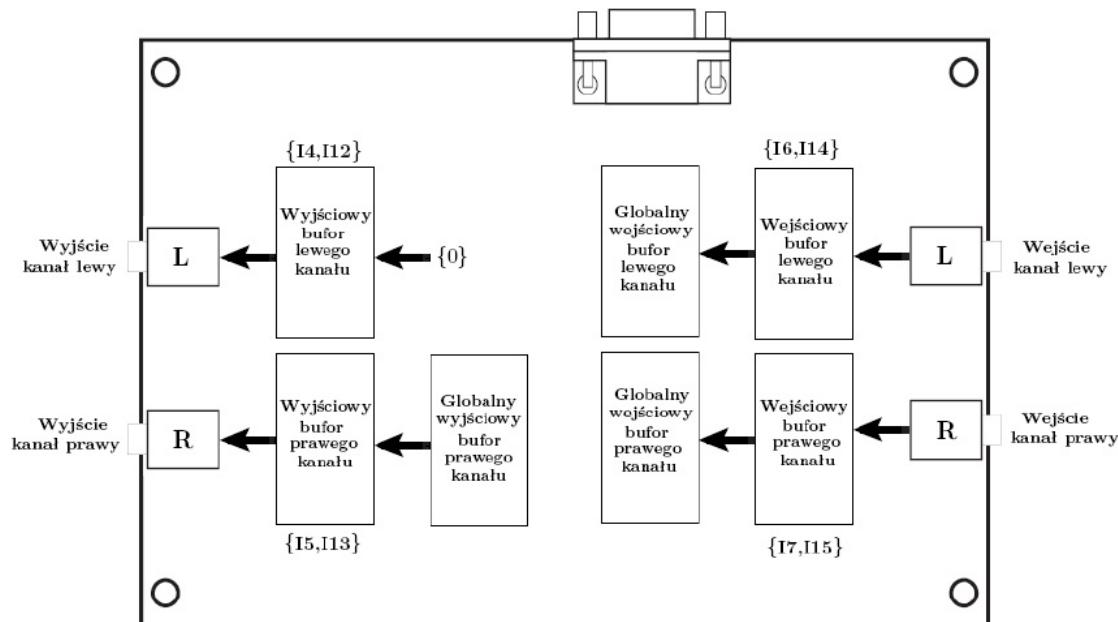
- „macros.h” – definicje przydatnych makr używanych w aplikacji;
- „memdef.h” – definicje zmiennych i tablic w pamięci procesora sygnałowego.

Pliki opisu dla linkera:

- „cner.ldf” – plik opisu dla linkera przypisujący zadeklarowane zmienne do rzeczywistych miejsc w pamięci procesora.

Zbieranie wejściowych danych audio do wykorzystania przez zaimplementowane procedury

Próbki wejściowego sygnału audio po konwersji A/C pojawiają się w pamięci procesora pod symbolicznym adresem **rx_buf** dla próbki z lewego kanału oraz w sukcesywnej komórce pamięci dla próbki z prawego kanału. Próbki sygnału do konwersji C/A są umieszczane do dwóch komórek pamięci, począwszy od adresu **tx_buf** dla obydwóch kanałów. W momencie odebrania danej z przetwornika zgłaszaną jest przerwanie **SPROI**. Przetwarzanie danych zostało zaprojektowane w formie blokowej. W pamięci procesora utworzono siedem buforów służących do jego obsługi. Długość bloku ustalono na 128, co przy częstotliwości próbkowania 8000 Hz daje 16-milisekundowe odcinki czasu. Na poniższym rysunku przedstawiono schematycznie procesy zachodzące podczas zbierania wejściowej informacji.



RYS. 29 Procesy zachodzące podczas zbierania informacji wejściowej

Każdy z buforów podzielony jest na dwie części: parzystą i nieparzystą. Do adresowania wejściowych i wyjściowych buforów lewego i prawnego kanału przeznaczono osiem wskaźników z podstawowego banku rejestrów. Części parzyste buforów wejściowych umieszczone są w pierwszym bloku pamięci, podobnie jak części nieparzyste buforów wyjściowych. Adresują je wskaźniki **I12–I15**. Części nieparzyste buforów wejściowych, podobnie jak części parzyste buforów wyjściowych, umieszczone w drugim bloku pamięci i są one wskazywane przez **I4–I7**. Wskaźniki te mają status „nietykalnych” w dalszej części programu, jedyną procedurą mającą do nich dostęp jest program obsługi przerwania kodeka. Ponadto bufory te zostały ustawione jako bufory cykliczne o długości 64 próbek.

Program obsługi przerwania kodeka po skompletowaniu przychodzącej próbki sygnału wejściowego zamienia ją po pobraniu z pamięci na postać liczby zmiennoprzecinkowej, po czym umieszcza w odpowiednim buforze wejściowym, zależnie od tego, czy mamy do czynienia z próbką parzystą, czy nieparzystą. Następnie wskaźnik jest inkrementowany i wskazuje kolejne miejsce w buforach wejściowych. Równocześnie pobierane są dane z wyjściowych buforów, przeprowadzana jest konwersja na format stałoprzecinkowy, po czym następuje transmisja próbki do przetwornika C/A. Rozkazy przeprowadzające konwersje użytych formatów są następujące:

f0=float **r0 by r1;**

r0=fix **f0 by r1;**

W rejestrze **r1** umieszczana jest wartość skalująca wykładnik wynikowej liczby zmiennoprzecinkowej.

Po skompletowaniu ostatniej próbki i umieszczeniu jej w buforze wejściowym, po inkrementacji, wskaźnik z racji cykliczności zostaje ustawiony na początek buforu oraz generowane jest przerwanie **CB7I**. Na samym starcie programu obsługi **CB7I** zawartość pełnych buforów wejściowych jest kopowana do globalnych buforów wejściowych, a zawartość globalnego buforu wyjściowego jest kopowana do buforu wyjściowego prawnego kanału. Do buforu wyjściowego lewego kanału wpisywane są zera. Dla potrzeb eliminacji echa akustycznego ustalono, że sygnał wejściowy filtru transmitowany jest prawym kanałem wejściowym, zaś sygnał pożądany – lewym. Wyjście filtru wysyłane jest na prawy kanał. Lewy kanał wyjściowy nie jest wykorzystywany. W dalszej części programu obsługi przerwania buforu cyklicznego uruchamiane są odpowiednie procedury w zależności od bieżącego algorytmu. Globalne bufory utworzono po to, by owe procedury miały dostęp

do aktualnych danych, gdyż podczas ich pracy zbierane są już nowe próbki i nadpisywane w wejściowych buforach. Po zakończeniu przetwarzania procedury te umieszczały wyniki do wysłania w globalnym buforze wyjściowym, który zostanie poddany transmisji po skompletowaniu nowego bloku. Jak łatwo policzyć, od momentu pojawienia się próbki sygnału na wejściu płyty EZ-KIT do momentu wystawienia jej przetworzonej na wyjście mija dokładnie 32 milisekundy.

The screenshot displays two windows side-by-side. The top window is titled "Active DAG1 (DM)" and the bottom one is titled "Active DAG2 (PM)". Both windows show tables of memory addresses and their corresponding values. In DAG1, address I0 contains 00000000, M0 contains 00000000, B0 contains 00000000, and L0 contains 00000000. Address I1 contains 00000000, M1 contains 00000001, B1 contains 00000000, and L1 contains 00000000. This pattern continues for I2 through I7. In DAG2, address I8 contains 00000000, M8 contains 00000000, B8 contains 00000000, and L8 contains 00000000. Address I9 contains 00000000, M9 contains 00000001, B9 contains 00000000, and L9 contains 00000000. This pattern continues for I10 through I15. The last row in both tables shows address I7/I15 with value FFFFFFFF and address B7/B15 with value 00024C00, indicating a loop or specific initialization value.

RYS. 30 Wartości wskaźników do buforów audio tuż po inicjalizacji

O jednej rzeczy należy jeszcze pamiętać. Mianowicie zbieranie próbek odbywa się jakby w tle innych działań, więc po przyjęciu przerwania program jego obsługi powinien pozostawić wartość tymczasowo użytych rejestrów w stanie nienaruszonym. Z tej przyczyny przeznaczono na tymczasowy stos sześć komórek pamięci bloku drugiego. Program obsługi przerwania kodeka zajmuje w pamięci programu 48 słów rozkazowych. Rozpoczyna się od adresu *input_samples*. Każdorazowe jej wywołanie pochłania 44 takty zegarowe procesora.

Program organizacyjny

Po uruchomieniu aplikacji przechodzi ona do pierwszego stanu swojej pracy, podczas której przesyła zbierany sygnał wejściowy z lewego kanału na wyjście. Można by powiedzieć, że procesor sygnałowy spełnia funkcję „drutu”. Nie przetwarza w żaden sposób pojawiających się danych. Naciśkając przycisk z przerwaniem IRQ1, przełączamy przesyłany kanał zamiennie między prawym i lewym. Stan aplikacji, w którym na wyjście przenoszony jest kanał prawy, jest oznaczony jako stan 2. Wcisnąć drugi, dostępny użytkownikowi przycisk oznaczony FLAG1, uruchamiamy specjalny program organizacyjny oznaczony jako stan 0. Trwa on tak długo, dopóki nie poluźnimy nacisku z FLAG1. Podczas trwania programu organizacyjnego możemy za pomocą przycisku IRQ1 przełączać się pomiędzy siedmioma możliwymi stanami aplikacji. I tak, trzykrotne naciśnięcie przycisku IRQ1 przełącza aplikację w stan 3, który zostaje ustawiony po puszczeniu FLAG1. Ośmiokrotne naciśnięcie IRQ1 zawija odliczanie począwszy od stanu 1. Dodatkowo, numer aktualnie wybranego stanu jest prezentowany za pomocą diody D4 podłączonej do wyjścia FLAG2 procesora. Dioda zapala się z częstotliwością $3\frac{1}{3}$ Hz tyle razy, ile wynosi stan aplikacji, po czym gaśnie na 1000 milisekund.

Dostępne możliwe stany aplikacji są następujące:

- Stan [0]: program organizacyjny;
- Stan [1]: wejście kanał lewy \Rightarrow wyjście;
- Stan [2]: wejście kanał prawy \Rightarrow wyjście;
- Stan [3]: procedura ϵ -NLMS; na wyjście przesyłany jest sygnał błędu jako różnica sygnału

wyjściowego filtru oraz sygnału pożądanego; przycisk IRQ1 zamiennie włącza bądź wyłącza adaptację;

- Stan [4]: procedura BEFAP; na wyjście przesyłany jest sygnał błędu jako różnica sygnału wyjściowego filtru oraz sygnału pożądanego; przycisk IRQ1 zamiennie włącza bądź wyłącza adaptację;
- Stan [5]: procedura LSA;
- Stan [6]: procedura łącznej redukcji szumów i echa akustycznego z wykorzystaniem algorytmów ϵ -NLMS oraz LSA;
- Stan [7]: procedura łącznej redukcji szumów i echa akustycznego z wykorzystaniem algorytmów BEFAP oraz LSA;
- Stan [8]: na wyjście audio przesyłane są zerowe wartości.

Zaprezentowane menu osiągnięto następująco: po starcie programu inicjalizowane są wszystkie wymagane rejestyry, po czym program zostaje zatrzaśnięty w pętli testowania flagi 1. Aktualny stan jest przechowywany w pamięci w komórce o adresie **Main_Status**. Dokładnie co 16 milisekund występuje jednak przerwanie buforu cyklicznego, więc sterowanie przenosi się do jego programu obsługi o adresie **Fullblock**. Program obsługi po obsłuzeniu buforów wejściowych i wyjściowych sprawdza aktualny stan i podejmuje odpowiednie działania. Podczas zatrzaśnięcia w zamkniętej pętli możliwe jest jednak wywołanie przerwania **IRQ1**, którego program obsługi wykonuje dodatkowe czynności, takie jak opisane włączanie/wyłączanie filtru adaptacyjnego czy przełączanie pomiędzy wysyłanymi na wyjście kanałami audio. Sprawdzanie flagi 1 odbywa się za pomocą rozkazu:

```
if flag1_in instr;
```

flag1_in jest to specjalny warunek języka procesora testujący aktualny bit **FLAG1** w rejesterze **ASTAT**. Po wciśnięciu **FLAG1** wartość pod adresem **Main_Status** zostaje ustawiona na 0 i zawiązywana jest druga pętla, tym razem testująca puszczenie przycisku. Program obsługi **IRQ1** wykrywa stan 0 i za każdym razem zwiększa zawartość komórki pamięci **Set_Status**. Po puszczeniu przycisku FLAG1 operacja **Set_Status** \Rightarrow **Main_Status** wprowadza ostatecznie aplikację w nowy stan. Kod asemblera wykrywający stan aplikacji i odpowiednio nań reagujący jest przedstawiony poniżej:

1)	<i>r0=dm(Main_Status);</i>
2)	<i>r1=0x01;</i>
3)	<i>i0=Alg_address;</i>
4)	<i>comp(r0,r1);</i>
5)	<i>lcntr=8, do Test_Algorithm_Status until lce;</i>
6)	<i>r2=dm(i0,m1);</i>
7)	<i>i8=r2;</i>
8)	<i>if lt jump (m8,i8)(db,la);</i>
9)	<i>r1=r1+1;</i>
10)	<i>Test_Algorithm_Status: comp(r0,r1);</i>

PROCEDURA 1. Badanie stanu aplikacji

Wartość statusu jest wczytywana do rejestru **r0**, po czym jest on sukcesywnie porównywany z zawartością **r1** za pomocą rozkazu 10). Wszystko dzieje się w pętli programowej zawiązanej rozkazem 5). W momencie gdy wciąż inkrementowana wartość w **r1** przekroczy aktualny status, następuje skok do programu obsługującego dany stan. Odpowiedni adres programu obsługi pobierany jest z tablicy w pamięci wskazywanej przez adres **Alg_address**. Do rozkazu skoku zostaje dodany specjalny modyfikator (**la**), co oznacza, że skok jest wykonywany poza obszar objęty pętlą i ma zostać ona unieważniona. Warto wspomnieć jeszcze o zadaniu licznika taktów zegarowych. Jego zadaniem jest odliczanie odcinków czasu 150ms oraz 1000ms. Wykorzystanie licznika jest stosunkowo łatwe. Wystarczy w odpowiednim momencie do **TCOUNT** oraz **TPERIOD** wpisywać wartości **0x005b0d80** oraz **0x02625a00**, po czym odpowiednio obsłużyć przerwanie.

Procedury szybkiej transformacji Fouriera

Szybka transformacja Fouriera FFT to jedna z podstawowych i najczęściej stosowanych operacji w cyfrowym przetwarzaniu sygnałów. Standardowa FFT pobiera ciąg zespolonych próbek wejściowych, dokonując przekształcenia w inny ciąg zespolonych wartości. W przypadku jednak, gdy wartości transformowane są rzeczywiste, procedura ta wykazuje pewną nadmiarowość. Połowa wartości po transformacji będzie ze sobą zespolona sprzężona, z tego powodu nie ma potrzeby ich obliczać. Procedury uwzględniające ten fakt nazywane są rzeczywistymi szybkimi transformacjami Fouriera RFFT [47]. W dalszej części pracy odniesienie do skrótu FFT będzie kryło w sobie milczące założenie, że mowa *de facto* o zastosowanej jego rzeczywistej odmianie. Algorytmy FFT zoptymalizowane pod kątem danego procesora są zwykle dostarczane przez producenta. Nie inaczej jest w tym przypadku. Jednym więc zadaniem jest ich dostosowanie do wymagań aktualnie im stawianych.

Aby wykonać szybką transformację Fouriera rzeczywistego sygnału o długości N , należy zarezerwować w pamięci po $N+1$ słów w każdym z obu bloków. Po $N/2$ słów na blok przeznaczone jest na współczynniki trygonometryczne potrzebne do kalkulacji, które zostały stablicowane w pamięci w celu znacznej redukcji nakładów obliczeniowych. Ponadto w pierwszym bloku trzeba zarezerwować miejsce na $N/2+1$ liczb będących częścią urojoną obliczonej transformaty. Podobnie w drugim bloku do tablicy o długości $N/2+1$ zostanie wpisana część rzeczywista. Rozdzielanie danych po połowie do każdego z bloków sprawia, że możliwe jest wtedy jednoczesne pobieranie dwóch argumentów z pamięci, wykorzystując pamięć notatnikową. Odwrotny FFT ma podobne wymagania pamięciowe, z tą oczywistą różnicą, że tablice wynikowe mają długość $N/2$.

Kolejnym ograniczeniem wprowadzanym przez procedury FFT, związane ze stosowaniem odwracania bitowego adresów danych w tablicach, jest warunek umieszczenia buforu w pamięci tak, by zaczynał się komórką pamięci będącą wielokrotnością $N/2$. I tak, ten warunek przy prostym FFT muszą spełniać obydwie tablice zawierające sygnał do transformowania, w przypadku odwrotnego FFT – tablica części rzeczywistej odwracanej transformaty oraz tablica, w której zostanie umieszczonena część parzysta wyniku. Dla IFFT dodatkowo obie tablice zawierające transformaty są niszczone podczas działania procedury. Furtką obniżającą w pewnym stopniu ilość potrzebnej pamięci jest możliwość użycia tego samego buforu na przechowanie części urojonej transformaty i części nieparzystej wyniku.

Algorytmy FFT wymagają wstępnego zdefiniowania kilku stałych programowych. Dla FFT musi to być odwrócony bitowo 24-bitowy adres wejściowej tablicy części nieparzystej w pierwszym bloku pamięci oraz odwrócony bitowo 32-bitowy adres wejściowej tablicy części parzystej w drugim bloku pamięci. Dla IFFT należy zdefiniować odwrócony bitowo 32-bitowy adres wejściowej tablicy części rzeczywistej w drugim bloku pamięci oraz odwrócony bitowo 32-bitowy adres wyjściowej tablicy części parzystej również w drugim bloku SRAM. Ponadto trzeba zdefiniować długość transformaty, adresy symboliczne tablic z trygonometrycznymi współczynnikami oraz liczbę sekcji algorytmu. Ponieważ procedury FFT będą używane zarówno do algorytmu adaptacyjnego, jak też do procedury redukcji szumów, zostały one zmodyfikowane tak, by możliwa była zmiana ich parametrów podczas pracy programu. W pamięci utworzony został specjalny stos FFT, począwszy od adresu `fft_addr_stack`, gdzie wpisywane są wszystkie wymagane parametry, które zostaną odczytane podczas pracy właściwego algorytmu. Dodatkowo utworzono szablon z argumentami opisującymi FFT dla długości 1024 oraz 256, ponieważ zostały użyte w pracy. Tym sposobem uruchomienie szybkiej transformaty Fouriera sprowadza się do szybkiego skopiowania 42 słów szablonu na stos FFT i wywołanie podprogramu.

Przydatne makra i procedury pomocnicze

Przed przystąpieniem do omawiania głównego programu warto jeszcze przyjrzeć się krótkim, pomocniczym podprogramom często w nim wykorzystywanym. Jednym z nich jest np. stosowana w BEFAP procedura odświeżania wektora autokorelacji. Prezentuje ona wykorzystanie rozkazów wielofunkcyjnych, więc zostanie teraz omówiona.

Zadaniem tego podprogramu jest odświeżenie wektora Y według poniższego wzoru:

$$Y = Y + aX - bZ$$

Odświeżenie każdego jego elementu wymaga dwóch mnożeń, dwóch dodawań i czterech sięgnięć do pamięci po odczyt i zapis argumentów. Zamiast jednak ośmiu taktów zegarowych, dzięki rozkazom wielofunkcyjnym wystarczą tylko dwa. Na efektywność programów asemblerowych można spojrzeć z perspektywy ilości mnożeń potrzebnych do pełnego wykonania danego algorytmu. Każde mnożenie wymaga jednego taktu zegarowego, więc minimalną ilością taktów, jaką trzeba zużyć, by wykonać obliczenia, jest właśnie liczba mnożeń. Gdyby osiągnąć powyższe minimum, można by uznać, że sprawność danej implementacji wynosi 100%. Zobaczmy więc, jak to się ma do zaanonsowanego problemu.

1)			$f5=pm(i9,m15);$
2)	$f8=f0*f5,$		$f7=pm(i10,m15);$
3)	$f12=f1*f7,$	$f4=dm(i1,m7);$	
4)	$f9=f0*f4,$	$f6=dm(i2,m7),$	$f15=pm(i8,m8);$
5)	$f13=f1*f6,$	$f15=f10+f15,$	$f4=dm(i1,m7),$
6)	$f8=f0*f5,$	$f14=f9-f13,$	$f5=pm(i9,m15);$
7)	$f12=f1*f7,$	$f11=f11+f14,$	$f7=pm(i10,m15);$
8)	$f9=f0*f4,$	$f6=dm(i2,m7);$	
9)	$f13=f1*f6,$	$f4=dm(i1,m7),$	$f5=pm(i9,m15);$
10)	lcntr=N/2-2, do cmpt_crr until lce;	$f6=dm(i2,m7),$	$f7=pm(i10,m15);$
11)	$f8=f0*f5,$	$dm(i0,m1)=f11,$	$pm(i8,m9)=f15;$
12)	$f12=f1*f7,$	$f14=f9-f13,$	$f15=pm(i8,m8);$
13)	$f9=f0*f4,$	$f11=dm(i0,m0),$	$f5=pm(i9,m15);$
14) cmpt_crr:	$f13=f1*f6,$	$f4=dm(i1,m7),$	$f7=pm(i10,m15);$
15)		$f6=dm(i2,m7),$	$pm(i8,m9)=f15;$
16)		$f10=f8-f12,$	$f11=dm(i0,m1),$
17)	$rts(db),$	$f14=f9-f13,$	$pm(i8,m8);$
18)		$f15=f10+f15;$	$pm(i8,m9)=f15;$
19)		$f11=f11+f14,$	$dm(i0,m1)=f11;$

PROCEDURA 2. Odświeżanie wektora autokorelacji

Parametry skalujące spoczywają w rejestrach $f0$ i $f1$. Rejestry $I0$ i $I8$ wskazują na część parzystą i nieparzystą wektora Y , rejesty $I1$ i $I9$ – na część parzystą i nieparzystą wektora X , a rejesty $I2$ i $I10$ – podobnie dla wektora Z . W pierwszym i drugim kroku pętli wymnażane są przez współczynniki skalujące odpowiednie nieparzyste wartości tablic X i Z . W trzecim i czwartym kroku wymnażane są wartości parzyste. Obok mnożenia przeprowadzane jest sumowanie wyników mnożeń z poprzedniej iteracji oraz wczytanej wartości tablicy Y . W pierwszym kroku pętli następuje

zapis ostatnio obliczonej wartości Y , w kolejnych trzech są wczytywane parametry z wektorów, które będą użyte przy mnożeniu w kolejnej iteracji. Aby pętla prawidłowo działała, trzeba przygotować liczby do zapisu w jej pierwszym rozkazie. Stąd zapisane są wprost pierwsze dwie iteracje oraz zmniejszono liczbę obiegów pętli o dwa. Poza pętlą znajdują się jeszcze rozkazy wykonujące resztę dodawania i zapisów do pamięci po wykonaniu wszystkich mnożeń.

Dla wektorów o długości N oraz przy założeniu, że żaden wymagany rozkaz początkowo nie będzie znajdował się w pamięci notatnikowej i będzie musiał zostać tam umieszczony, procedura ta zużywa $2N + 22$ takty zegarowe i zajmuje 19 słów rozkazowych. Przykładowo dla $N = 16$ osiągnęlibyśmy sprawność około 59%, co nie jest najlepszym wynikiem, jednak dla $N = 1024$ sprawność sięgnęłaby prawie 99%.

Pozostałe użyte podprogramy to dzielenie liczb zmiennoprzecinkowych, odwrotność pierwiastka, mnożenie z sobą poszczególnych elementów wektorów, wyznaczanie wartości wielomianu za pomocą schematu Hornera, wyznaczanie eksponenty z całki wykładniczej, co zostanie dokładniej omówione przy opisie algorytmu LSA, w końcu wyznaczanie wartości funkcji trygonometrycznych – sin, cos, arctan.

Innym typem ułatwienia oferowanym przez asembler są tzw. makra. Jedno z zastosowanych makr jest używane do dzielenia liczb zmiennoprzecinkowych. Jego forma jest zaprezentowana poniżej.

```
#define DIVIDE(q,n,d,two,tmp)
1) n=RECIPS d,      tmp=n;           \
2) d=n*d;           \
3) tmp=tmp*n,       n=two-d;        \
4) d=n*d;           \
5) tmp=tmp*n,       n=two-d;        \
6) d=n*d;           \
7) tmp=tmp*n,       n=two-d;        \
8) q=tmp*n;
```

PROCEDURA 3. Dzielenie liczb zmiennoprzecinkowych w postaci makra

Makro jest wywoływanie z pięcioma parametrami, którymi są rejestrzy arytmetyczne procesora **r0-r15**. Różnica pomiędzy wywołaniem podprogramu a makrem jest zasadnicza. W przypadku makra, jeszcze przed etapem asemblacji, preprocesor zamienia wszystkie wystąpienia argumentów z definicji poprzez podane w wywołaniu rejestrzy i umieszcza w kodzie programu. Znaczy to, że każdorazowe wywołanie makra generuje tyle kodu, na ile zostało zdefiniowane. Dlatego używanie często bardzo dużych makr zużywa o wiele większą ilość pamięci niż uruchomienie podprogramu, jednak oszczędzamy takty zegarowe potrzebne na wykonanie skoku i powrotu. Gdy skok ze śladem jest umieszczony w pętli programowej przechodzącej przez bardzo dużo cykli, może mieć to ostatecznie znaczenie.

Algorytm RLS w formie kratowej z wykorzystaniem rozkładu QRD

Predyktor kratowy algorytmu 3 został zaimplementowany w asemblerze procesora sygnałowego. Jego początek znajduje się w pamięci programu od adresu *swflqrd*. Algorytm używa większość rejestrów alternatywnego generatora adresowego do wskazywania elementów wykorzystywanych tablic. Rejestry **I1** oraz **I2** adresują wektory mocy błędów predykcyjnych, rejestyry **I4–I7** adresują wektory błędów. Znormalizowane współczynniki odbicia są wskazywane przez rejestyry **I8** oraz **I9**, wektory współczynników konwersji – przez **I12** i **I13**. Rejestr adresowy **I14** posiada adres procedury obliczania odwrotności pierwiastka z liczby zmiennoprzecinkowej, rejestr **I15** adresuje pomocniczy bufor błędów predykcyjnych. Inicjalizacja oraz znaczenie poszczególnych zmiennych przedstawiona jest w tabeli 5.

Procedura dzieli się na cztery bloki wykonyjące podobne obliczenia z użyciem innych danych. Dlatego wystarczy omówić jeden taki blok, gdyż pozostałe trzy działają właściwie tak samo. Bloki logicznie oddzielone są operacją obliczenia odwrotności pierwiastka poprzez wywołanie podprogramu rozkazem *call* bez używania jednak symbolicznego adresu, a za to wskaźnika **I14**, gdyż tylko wtedy można powiązać rozkaz z dodatkową operacją arytmetyczną. Rozpoczniemy więc omawianie od rozkazu 14. Zakładamy, że w rejestrach dotychczas zostały zebrane poprawne parametry. I tak, podczas wywołania podprogramu dokonywane jest jeszcze mnożenie $\rho_m^{f(L-1)}(i-1)s_m^b$ dla danych z poprzedniego bloku. Skok jest opóźniony, więc przed przeniesieniem sterowania zostaną wykonane jeszcze dwa rozkazy. W pierwszym z nich zachodzi operacja:

$$\mathbf{f11} \leftarrow \left[\zeta_m^{\frac{b}{2}(L)}(i-1) \right]^2, \quad \mathbf{f9} \leftarrow f_{m+1}(i),$$

a do **f3** jest wczytywana znowu wartość $f_m(i)$. W rozkazie 16 w **f14** umieszczany jest kwadrat $f_m(i)$ oraz:

$$\mathbf{f0} \leftarrow \left[\zeta_m^{\frac{b}{2}(L)}(i-1) \right]^2 - \acute{b}_m(i-L)^2$$

Wartość w **f14** zostanie zapamiętana i użyta pod pierwiastkiem w następnym bloku, wartość z **f0** jest argumentem operacji odwrotności pierwiastka w aktualnym bloku. Po powrocie z podprogramu w rejestrze **f4** zwracany jest wynik, po czym jest on mnożony przez argument funkcji obliczającej odwrotność pierwiastka, ażeby otrzymać właściwy pierwiastek:

$$\mathbf{f4} \leftarrow \sqrt{\left[\zeta_m^{\frac{b}{2}(L)}(i-1) \right]^2 - \acute{b}_m(i-L)^2} = \zeta_m^{\frac{b}{2}(L-1)}(i-1)$$

Równocześnie do **f0** pobierany jest z pamięci $\acute{b}_m(i-L)$. Rozkaz 18 i 19 to obliczenie parametrów rotacji hiperbolicznej ch_m^b , sh_m^b z równoczesnym wczytaniem do **f7** $\acute{f}_m(i-L+1)$. W rozkazie 19 dodatkowo zapisywane są do pamięci obliczone parametry $\zeta_m^{\frac{b}{2}(L-1)}(i-1)$ oraz $\zeta_m^{-\frac{b}{2}(L-1)}(i-1)$. Pierwszy z nich zostanie wykorzystany w następnej chwili czasu, drugi przy odświeżaniu współczynników predykcyjnych w przód i w tył. Cztery sukcesywne mnożenia w rozkazach 20–23 wyznaczają

$$\rho_m^{f(L)}(i)ch_m^b, \quad \acute{f}_m(i-L+1)sh_m^b, \quad \acute{f}_m(i-L+1)ch_m^b, \quad \rho_m^{f(L)}(i)sh_m^b$$

Korzystające z nich odejmowanie w rozkazie 22 wpisuje do **f9** odświeżoną $\rho_m^{f(L-1)}(i)$. Odejmowanie podnoszące rząd $\acute{f}_{m+1}(i-L+1)$ będzie już miało miejsce w linii opóźnienia kolejnego wywołania procedury odwrotności pierwiastka, podobnie jak to miało miejsce w rozkazie 15 w danym bloku. Operacje na pamięci w rozkazach 20–23 wpisują do rejestrów nowe wartości potrzebne w kolejnym bloku.

Parametr algorytmu	Adres początkowy	Inicjalizacja
$\zeta_m^{\frac{f}{2}(L-1)}(i)$	Pfd	$I1=Pfd$
$\zeta_m^{\frac{b}{2}(L-1)}(i-1)$	$Pbdb$	$I2=Pbdb$
$f_m(i)$	$fposu$	$I4=fposu$
$b_m(i)$	$bposu$	$I5=bposu+1$
$\acute{f}_m(i-L+1)$	$fposd$	$I6=fposd$
$\acute{b}_m(i-L+1)$	$bposd$	$I7=bposd$
$\rho_m^{b(L-1)}(i)$	qbd	$I8=qbd-1$
$\rho_m^{f(L-1)}(i)$	qfd	$I9=qfd$
$\acute{\gamma}_m(i-L+1)$	gd	$I12=gd$
$\gamma_m(i)$	gu	$I13=gu+1$
$[\acute{b}_0(i-L), b_0(i-1), \acute{\gamma}_0(i-L), \gamma_0(i-1)]$	$bpbuff$	$I15=bpbuff+1$

TABELA 5. Inicjalizacja parametrów początkowych predyktora kratowego RLS

Na koniec warto omówić jak wykorzystać pamięć notatnikową w możliwie najoptymalniejszy sposób, gdyż bez ingerencji programisty mogą wystąpić, przy tworzeniu długich pętli tzw. „chybione strzały”. W przypadku wykonywania rozkazu z dostępem do pamięci w pierwszym bloku występuje konflikt, gdyż w tym samym czasie procesor ma za zadanie wczytać drugi z kolej rozkaz za aktualnie wykonywanym. Nie ma innej możliwości, jak tylko opóźnić wykonanie rozkazu o jeden cykl zegarowy. Aby w przyszłości uniknąć podobnego przypadku, po wczytaniu owego rozkazu umieszcza się go w pamięci notatnikowej wewnątrz procesora. Podczas wykonywania kolejnego rozkazu z dostępem do pamięci w bloku pierwszym, zostanie sprawdzone, czy drugi z kolej rozkaz za właśnie wykonywanym nie znajduje się już w pamięci notatnikowej. Gdy tak będzie, wykonywany rozkaz otrzyma swobodny dostęp do pamięci, a kolejną instrukcję pobierze się właśnie z pamięci notatnikowej. Posiada ona miejsce na 32 instrukcje i dzieli się na 16 zbiorów po dwie instrukcje. Każdy zbiór jest adresowany za pomocą czterech najmniej znaczących bitów adresu pobieranego rozkazu. Przykładowo, gdy w kodzie programu znajdują się następujące instrukcje:

```
0x020836:           f2=dm(i2,m0),      f9=pm(i15,m15);
0x020837:   f14=f9*f9,    f11=dm(i7,m0),   f7=pm(i15,m8);
0x020838:           dm(i7,m0)=f7;
```

to podczas wykonywania pierwszej z nich trzeci rozkaz zostanie umieszczony w ósmym zbiorze w pamięci notatnikowej. Gdy teraz w pętli programowej co najmniej trzy rozkazy z dostępem do pamięci w bloku pierwszym zostaną tak rozmieszczone, że wczytywane do pamięci notatnikowej instrukcje będą miały tę samą najmniej znaczącą tetradę adresu, skończy się na tym, że za każdym razem będą tam umieszczane cyklicznie. W wyniku tego efektywność pamięci notatnikowej spadnie do zera i każdy dostęp do pamięci będzie odbywał się poprzez dwa takty zegarowe. Banalnym wręcz na to lekarstwem jest przeniesienie – jeżeli jest to możliwe – części rozkazu z dostępem do pamięci w bloku pierwszym w miejsce o innej najmłodszej tetradzie.

1)		$f2=dm(i2,m0),$	$f9=pm(i15,m15);$
2)	$f14=f9*f9,$	$f11=dm(i7,m0),$	$f7=pm(i15,m8);$
3)		$dm(i7,m0)=f7;$	
4)	<i>lcntr=K, do floop until lce;</i>		
5)	$call(m8,i14)(db), \quad f12=f0*f6,$	$f13=f11-f12;$	
6)	$f11=f2*f2,$	$f0=dm(i7,m0),$	$pm(i12,m9)=f12;$
7)	$f14=f0*f7,$	$f0=f11+f14;$	$pm(i15,m9)=f7;$
8)	$f11=f4*f0,$		$pm(i15,m15)=f9;$
9)	$f7=f9*f4,$		$f4=pm(i9,m8);$
10)	$f3=f2*f4,$	$f5=abs \quad f11,$	
11)	$f12=f3*f4,$	$f2=abs \quad f11;$	
12)	$f9=f6*f7,$		$dm(i7,m1)=f13, \quad pm(i8,m9)=f15;$
13)	$f9=f3*f6,$	$f15=f9+f12;$	
14)	$call(m8,i14)(db), \quad f13=f4*f7;$		
15)	$f11=f2*f5,$	$f9=f9-f13,$	$f3=dm(i4,m1);$
16)	$f14=f3*f6,$	$f0=f11-f14;$	$f0=pm(i15,m9);$
17)	$f13=f4*f0,$		
18)	$f2=f2*f4,$		$f7=dm(i6,m0);$
19)	$f3=f0*f4,$		$dm(i2,m1)=f13, \quad pm(i9,m8)=f4;$
20)	$f11=f2*f15,$		$dm(i4,m0)=f9;$
21)	$f12=f3*f7,$		$f5=dm(i1,m0), \quad f10=pm(i9,m8);$
22)	$f11=f2*f7,$	$f9=f11-f12,$	$f2=dm(i1,m0);$
23)	$call(m8,i14)(db), \quad f12=f3*f15;$		
24)	$f11=f2*f5,$	$f13=f11-f12,$	$f3=dm(i6,m1), \quad f5=pm(i8,m8);$
25)	$f14=f3*f7,$	$f0=f11+f14,$	$pm(i9,m9)=f9;$
26)	$f6=f4*f0,$		$f0=dm(m7,i4);$
27)	$f2=f2*f4,$		$dm(i6,m0)=f13;$
28)	$f3=f0*f4,$		$f4=pm(i15,m9);$
29)	$f11=f2*f5;$		
30)	$f12=f3*f4,$		$dm(i1,m0)=f2;$
31)	$f11=f2*f4,$	$f13=f11+f12,$	$f9=pm(i13,m8);$
32)	$f12=f3*f5,$		$f5=pm(i15,m8);$
33)	$call(m8,i14)(db), \quad f5=f2*f5,$		
34)	$f11=f6*f6,$		$pm(i15,m14)=f9;$
35)	$f14=f0*f4,$	$f0=f11-f14,$	$pm(i8,m8)=f0;$
36)	$f15=f4*f0,$		$pm(i13,m9)=f5;$
37)	$f6=f4*f6,$		$f0=pm(i15,m11);$
38)	$f4=f7*f4;$		
39)	$f11=f6*f13,$		$f9=pm(i8,m8);$
40)	$f12=f0*f4,$		$f2=pm(i12,m8);$
41)	$f11=f0*f6,$	$f15=f11-f12,$	$f7=dm(i7,m0), \quad f0=pm(i15,m8);$
42) floop:	$f12=f4*f13,$		$f2=dm(i2,m0), \quad pm(i15,m13)=f2;$
43)	$f12=f0*f6,$	$f13=f11-f12;$	
44)	$f11=f2*f2,$		$f0=dm(i7,m0), \quad pm(i12,m9)=f12;$
45)			$dm(i7,m1)=f13, \quad pm(i8,m9)=f15;$

Jak widać, pętla została tak zaprojektowana, aby w każdym jej elemencie było przeprowadzane mnożenie. W ten sposób osiągnięto maksimum wydajności. Predyktor kratowy wymaga 38 mnożeń dla każdej iteracji, co zostało zaprogramowane w 38 rozkazach, otrzymując wszystkie dodawania oraz dostęp do pamięci zupełnie za darmo. Oczywiście sytuację psuje wyznaczenie odwrotności pierwiastka, które wymaga 9 mnożeń. Przydałoby się niezmiernie obliczyć go za pomocą sprzętowego arytmometru tak, by trwało to jeden takt zegarowy.

Predyktor kratowy został napisany z uwzględnieniem powyższych zależności, efektywność pamięci notatnikowej jest więc maksymalna. Ponieważ jest 21 rozkazów z dostępem do pamięci w bloku pierwszym, pierwszy obieg pętli potrzebuje 95 taktów zegarowych, zakładającą pustą pamięć notatnikową na początek. Jednak już przy drugim obiegu wystarczą tylko 74 taki. Podsumowując, predyktor kratowy wymaga ostatecznie $74K + 31$ taktów zegarowych. Zakładając $K = 16$, daje to 1215 taktów oraz sprawność 97,5%.

Blokowy szybki algorytm ϵ -APA z korekcją współczynników

Blokowy szybki algorytm ϵ -APA z korekcją współczynników, inaczej BEFAP, to najbardziej złożona procedura w niniejszej pracy. Zawiera aż 521 słów rozkazowych razem z funkcją inicjalizacyjną. Algorytm ten rozpoczyna się od adresu **befap**, procedura inicjalizacyjna od adresu **befap_init**. Przyjęte zostały następujące parametry algorytmu: długość filtru $N = 896$ i długość bloku $L = 128$, co daje w efekcie $N + L = 1024$ dla procedury FFT. Godnym uwagi jest fakt, że to nie ilość zużytych taktów zegarowych, a dostępna pamięć wyznaczyła dolną granicę na długość zastosowanego filtru. Przy częstotliwości próbkowania 8000 Hz i długości bloku 128 oraz uwzględniając zapotrzebowania programu obsługi przerwania kodeka, liczba taktów możliwych do wykorzystania wyniosła ponad 635 000. Algorytm BEFAP wymaga jednak co najmniej $2,5N$ komórek pamięci w bloku pierwszym na same bufory próbek wejściowych oraz FFT. Mając dostępne 3956 słów, wspomniana wartość N okazała się zaporowa. Gdyby jednak uruchomić BEFAP na innym procesorze tej rodziny – ADSP 21060 Sharc – który różni się od 21061 tylko wielkością SRAM równą 4Mb, możliwe stałoby się nawet zwiększenie N .

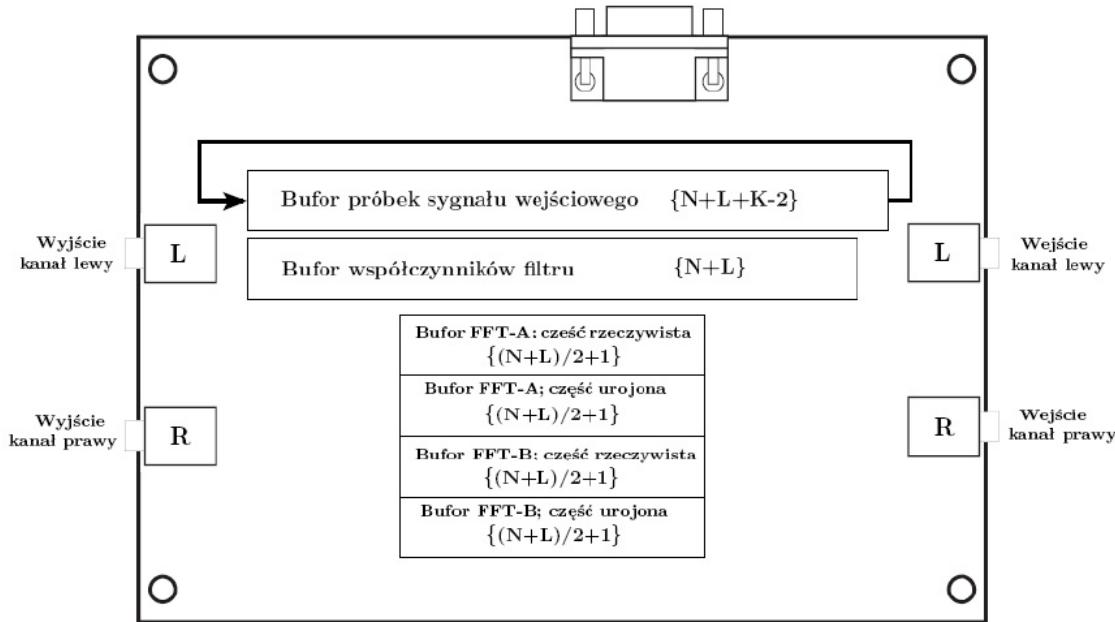
Przed przystąpieniem do adaptacji, trzeba zainicjalizować sześć dużych tablic w pamięci procesora. Pierwsza o długości $N + L + K - 2$ przeznaczona jest na próbki pojawiające się na wejściu filtru adaptacyjnego. Druga o długości $N + L$ ma zadanie przechowywać odświeżany wektor rozszerzonych współczynników filtru z_i . L dodatkowych komórek pamięci należało wypełnić zerami, aby skutecznie przeprowadzić splot z sygnałem wejściowym poprzez FFT. Obie tablice dzielą się na części parzyste i nieparzyste. Dodatkowo bufor próbek wejściowych został zainicjowany jako bufor cykliczny. Na potrzeby FFT przeznaczono dwie tablice o długości $(N + L)/2 + 1$ każda. Adresy początków buforów są następujące:

- bufor próbek wejściowych: **{Mainbuffeven, Mainbuffodd}**;
- bufor rozszerzonych współczynników filtru: **{Tapsbuffeven, Tapsbuffodd}**;
- bufor FFT-A: **{reffta, imffta}**;
- bufor FFT-B: **{refftb, imfftb}**.

Bufor próbek wejściowych w dalszej części pracy zostanie oznaczony skrótem MB, bufor współczynników – skrótem TB, bufory FFT – skrótami FFT-A oraz FFT-B odpowiednio. Bufor próbek wejściowych jest zapełniany wartościami w poniższej kolejności:

$$\{Mainbuffeven\} \Leftarrow [u(i - N - 2L - p + 4), \dots, u(i - 2), u(i)]$$

$$\{Mainbuffodd\} \Leftarrow [u(i - N - 2L - p + 3), \dots, u(i - 3), u(i - 1)]$$



RYS. 31 Bufory pamięciowe procedury BEFAP

Pierwszą czynnością wykonywaną przez algorytm BEFAP jest skopiowanie najnowszego bloku próbek wejściowych filtru z globalnego buforu wejściowego w odpowiednie miejsce w MB. Nie trzeba się martwić o nadpisanie ważnych wartości, gdyż w poprzedzającej iteracji zostały one odpowiednio przesunięte. Do tego celu użyto dolnych czterech rejestrów wskaźnikowych, pamiętając o zarezerwowanych górnych rejestrach dla programu obsługi kodera. Dodatkowo próbka $u(i-N-L+1)$ z MB została schowana do pamięci pod adres *hiddensample*, a na jej miejsce wpisano 0. Nastecną czynnością jest skopiowanie szablonu FFT o długości 1024 na stos FFT i uruchomienie procedury szybkiej transformaty Fouriera, zamieniając w międzyczasie bank generatorów adresowych na alternatywny. Tym sposobem pierwsze z równań algorytmu 1 zostało z powodzeniem przeliczone:

$$u_f = \mathcal{F} \{ [0 \quad \text{fliplr}(u_{N+L-1,i}^T)] \}$$

Wynik znajduje się w tablicach:

$$\mathbf{FFT\text{-}A}_{\text{real}} \Leftarrow \Re\{u_f\}$$

$$\mathbf{FFT\text{-}A}_{\text{imag}} \Leftarrow \Im\{u_f\}$$

Drugą czynnością jest przetransformowanie współczynników filtru uzupełnionych 128 zerami. Osiąga się to po zamianie odwróconych bitowo adresów tablic źródłowych na stosie FFT oraz tablic wynikowych. W rezultacie drugie równanie:

$$z_f = \mathcal{F} \{ [z_{i-L} \quad 0_L] \}$$

zostaje obliczone, a wynik umieszczony w tablicach:

$$\mathbf{FFT\text{-}B}_{\text{real}} \Leftarrow \Re\{z_f\}$$

$$\mathbf{FFT\text{-}B}_{\text{imag}} \Leftarrow \Im\{z_f\}$$

Trzecią czynnością jest wykonanie mnożenia odpowiadających elementów tablic FFT:

$$Y_f = u_f \odot z_f$$

Inicjalizacja:

<i>b0=refftta;</i>	<i>l0=HalfNL+1;</i>	<i>b1=refftbt;</i>	<i>l1=HalfNL+1;</i>
<i>b8=imfftta;</i>	<i>l8=HalfNL+1;</i>	<i>b9=imfftbt;</i>	<i>l9=HalfNL+1;</i>
<i>m0=0;</i>	<i>m1=1;</i>	<i>m7=-1;</i>	
<i>m8=0;</i>	<i>m9=1;</i>	<i>m15=-1;</i>	
1)		<i>f4=dm(i1,m0);</i>	<i>f1=pm(i8,m8);</i>
2)	<i>f8=f1*f4,</i>	<i>f0=dm(i0,m1),</i>	<i>f5=pm(i9,m9);</i>
3)	<i>f12=f0*f5;</i>	<i>f4=dm(i1,m7);</i>	
4)	<i>f8=f0*f4,</i>	<i>f2=f8+f12,</i>	<i>f4=dm(m1,i1);</i>
5)	<i>f12=f1*f5,</i>		<i>f1=pm(m9,i8);</i>
6)	<i>lcntr=HalfNL, do Mulfft1 until lce;</i>		
7)	<i>f8=f1*f4,</i>	<i>f12=f8-f12,</i>	<i>f0=dm(i0,m1),</i>
8)	<i>f12=f0*f5,</i>		<i>dm(i1,m1)=f12,</i>
9)	<i>f8=f0*f4,</i>	<i>f2=f8+f12,</i>	<i>f4=dm(m1,i1);</i>
10) Mulfft1:	<i>f12=f1*f5,</i>	<i>f12=f8-f12,</i>	<i>f1=pm(m9,i8);</i>
11))			<i>pm(i8,m9)=f2;</i>
			<i>pm(i8,m9)=f2;</i>
		<i>dm(i1,m1)=f12;</i>	

PROCEDURA 5. Mnożenie odpowiadających elementów tablic liczb zespolonych

Wynik mnożenia jest umieszczany w tablicach:

$$\text{FFT-B}_{\text{real}} \Leftarrow \Re\{Y_f\}$$

$$\text{FFT-A}_{\text{imag}} \Leftarrow \Im\{Y_f\}$$

Po ostatniej podmianie adresów na stosie FFT i uruchomieniu procedury odwrotnej transformaty Fouriera, zostaje obliczone:

$$Y = \mathcal{F}^{-1}\{Y_f\}$$

Wynik spoczywa w tablicach:

$$\text{FFT-A}_{\text{real}} \Leftarrow \{\text{Część parzysta z } Y\}$$

$$\text{FFT-A}_{\text{imag}} \Leftarrow \{\text{Część nieparzysta z } Y\}$$

Sygnał wyjściowy filtra został wyznaczony. Po odzyskaniu próbki $u(i - N - L + 1)$ można przystąpić do dalszych obliczeń.

Drugim etapem procedury BEFAP jest przetwarzanie sygnałów próbka po próbce. Dla każdej z nich trzeba odświeżyć wektor autokorelacji, wyznaczyć sygnał błędu, odświeżyć wektor błędu oraz wektor wzmacnienia. Aby tego dokonać zawiązano kolejną pętlę programową o L obiegach. Wektor autokorelacji został odnowiony za pomocą omówionej procedury 2. Rolę wektorów X oraz Z spełnia odpowiedni fragment buforu MB, podobnie współczynniki skalujące. Po odjęciu od sygnału pożądanego aktualnej próbki wyjściowej filtra:

$$\hat{e}(i - L + m) = d(i - L + m) - Y(N + m),$$

otrzymujemy sygnał błędu, wyznaczony na podstawie współczynników z chwili czasu $i - L$. Dla kolejnych $d(i - L + m)$ błąd ten będzie różnił się od błędu na wyjściu oryginalnego ϵ -APA. Stąd

potrzeba zaaplikowania korekcji. Wartość korekcyjna zostaje wyznaczona za pomocą iloczynu skalarnego:

$$S_{L+K-1,i-L+m-1}^*(1:m+K-2) r_{uu,m}(1:m+K-2)$$

Rozpisując powyższy wzór, w pierwszej iteracji należy dokonać iloczynu skalarnego pierwszych piętnastu elementów obu wektorów, w kolejnej – pierwszych szesnastu itd. Odpowiednia procedura do tego celu jest następująca:

1-4)	$r14=72;$	$r15=curlcntr;$	$f8=0;$	$f12=0;$
5)	$r3=r15+1;$			
6)	$r3=lshift r3 by -1,$			$f2=pm(i9,m9);$
7)	$r3=r14-r3,$		$f4=dm(i0,m1),$	$f5=pm(i8,m9);$
8)	$lcntr=r3, \text{ do cmpt_dot1 until lce};$			
9)	$f13=f2*f4,$	$f8=f8+f12,$	$f0=dm(i1,m1),$	$f2=pm(i9,m9);$
10) cmpt_dot1:	$f12=f0*f5,$	$f8=f8+f13,$	$f4=dm(i0,m1),$	$f5=pm(i8,m9);$
11)	$btst r15 by 0x00;$			
12)	$\text{if not } sz \text{ } f8=f8+f12;$			

PROCEDURA 6. Wyznaczanie współczynnika korekcyjnego próbki wyjściowej filtra

Rozkazy 8–10 opisują pętlę iloczynu skalarnego. Ponieważ zarówno wektor $S_{L+K-1,i-L+m-1}$, jak też $r_{uu,m}$ podzielone są na części parzystą i nieparzystą, pętla musi zostać uruchomiona na $\lceil m/2 \rceil + 7$ obiegów. Powyższą wartość wyznacza się w oparciu o zawartość licznika **CURLCNTR**. Na początku przetwarzania licznik ten zostaje załadowany wartością $L = 128$. Z każdą iteracją jego wartość zmniejsza się cyklicznie, aż zostanie wyczerpana. Ostatecznie, aby dla pierwszej i drugiej iteracji uruchomić pętlę na osiem obiegów, dla trzeciej i czwartej na dziewięć itd., można się posłużyć następującym wzorem:

$$M = 72 - \left\lceil \frac{\text{CURLCNTR}+1}{2} \right\rceil,$$

co jest obliczone za pomocą rozkazów 1–7. Operację zaokrąglania w góre przeprowadzono z wykorzystaniem przesuwania bitowego w prawo o jedną pozycję. Rozkaz 11 testuje parzystość licznika **CURLCNTR**, aby sprawdzić, czy wynik mnożenia próbek nieparzystych musi zostać dodany do końcowej wartości iloczynu skalarnego. Wskaźniki {10,18} zostają ustawione na początek obydwóch części wektora autokorelacji oraz wskaźniki {11,19} na początek wektora $S_{L+K-1,i-L+m-1}$.

Po korekcji $e(m)$ jest użyte do odświeżenia wektora błędu e_m zgodnie z:

$$e_m = \begin{bmatrix} e(m) \\ (1-\mu)\tilde{e}_{m-1} \end{bmatrix}$$

W tym momencie następuje uruchomienie szybkiego algorytmu RLS z predyktem kratowym w celu wyznaczenia bieżących współczynników predykcji w przód i w tył. Za ich pomocą wyznaczane są wektory B_m, F_m , potem następuje odświeżenie wzmacnienia filtra $g_{K,m}, g_{K-1,m}$ oraz $S_{L+K-1,m}$, co kończy przetwarzanie w danej iteracji. Po zakończeniu wszystkich obiegów pętli głównej zostaje uruchomiona druga faza przetwarzania blokowego z wykorzystaniem FFT.

Inicjalizacja:

<i>f2</i>	$\leftarrow 1 - \mu$		
<i>I0</i>	\leftarrow adres części parzystej z e_m		
<i>I8</i>	\leftarrow adres części nieparzystej z e_m		
1)		<i>f4</i> = $dm(i0,m0);$	
2)	<i>f10=f2*f4,</i>	$dm(i0,m1)=f11,$	<i>f5=pm(i8,m8);</i>
3)	<i>lcntr=K/2-1, do moveerr until lce;</i>		
4)	<i>f9=f2*f5,</i>	<i>f4</i> = $dm(i0,m0),$	$pm(i8,m9)=f10;$
5) moveerr:	<i>f10=f2*f4,</i>	$dm(i0,m1)=f9,$	<i>f5=pm(i8,m8);</i>
6)			$pm(i8,m9)=f10;$

PROCEDURA 7. Odświeżanie wektora błędu procedury BEFAP

Celem drugiej fazy przetwarzania blokowego w dziedzinie częstotliwości jest odświeżenie rozszerzonych współczynników filtra $z_{i-L} \Rightarrow z_i$. Pierwsza transformata Fouriera wyznaczana jest z L wartości wektora $S_{L,i}$, jak następuje:

$$\mathbf{FFT-A}_{\text{real}} \leftarrow \{\text{Część parzysta z } [S_{L,i}(p:p+L-1) \quad 0_N]\}$$

$$\mathbf{FFT-A}_{\text{imag}} \leftarrow \{\text{Część nieparzysta z } [S_{L,i}(p:p+L-1) \quad 0_N]\}$$

Po transformacji Fouriera:

$$\mathbf{FFT-B}_{\text{real}} \leftarrow \Re \{g_f = \mathcal{F} \{\mathbf{FFT-A}_{\text{real}} + j * \mathbf{FFT-A}_{\text{imag}}\}\}$$

$$\mathbf{FFT-B}_{\text{imag}} \leftarrow \Im \{g_f = \mathcal{F} \{\mathbf{FFT-A}_{\text{real}} + j * \mathbf{FFT-A}_{\text{imag}}\}\}$$

Żeby wykonać drugie wymagane FFT, trzeba włożyć jeszcze więcej pracy. Mianowicie najpierw należy przenieść:

$$\mathbf{FFT-A}_{\text{real}} \leftarrow [0 \quad u(i-L-N-p+4) \quad \dots \quad u(i-L-N) \quad \dots \quad u(i-p+2)]$$

$$\mathbf{FFT-A}_{\text{imag}} \leftarrow [u(i-L-N-p+3) \quad \dots \quad u(i-L-N-1) \quad \dots \quad u(i-p+3)]$$

A wszystko dlatego, że FFT można wykonać tylko dla buforu wejściowego o adresie początkowym będącym wielokrotnością $(N+L)/2$. Następnie zawartość buforu FFT-B jest chowana w miejsce dopiero co przelanych wartości z MB, więc można wykonać FFT:

$$\mathbf{FFT-B}_{\text{real}} \leftarrow \Re \{v_f = \mathcal{F} \{[0 \quad \text{fliplr}(u_{N+L-1,i-p+1}^T)]\}\}$$

$$\mathbf{FFT-B}_{\text{imag}} \leftarrow \Im \{v_f = \mathcal{F} \{[0 \quad \text{fliplr}(u_{N+L-1,i-p+1}^T)]\}\}$$

Dokonując mnożenia odpowiadających elementów z FFT-B ze schowanym w MB wartościami g_f i po umieszczeniu wyniku w FFT-B, mamy w końcu:

$$\mathbf{FFT-B}_{\text{real}} \leftarrow \Re \{v_f \odot g_f\}$$

$$\mathbf{FFT-B}_{\text{imag}} \leftarrow \Im \{v_f \odot g_f\}$$

Ostatnie transfery, jakie trzeba wykonać, to:

$$\text{Część parzysta z MB} \Leftarrow \text{Część parzysta z } \mathcal{F}^{-1} \{v_f \odot g_f\}$$

$$\text{FFT-B}_{\text{imag}} \Leftarrow \text{Część nieparzysta z } \mathcal{F}^{-1} \{v_f \odot g_f\}$$

Można teraz odświeżyć współczynniki z_i , a po przesłaniu zachowanych próbek z FFT-A do MB w taki sposób, by zostawić 128 komórek wolnych na najnowsze próbki z następnego bloku, procedura BEFAP zostaje ostatecznie zakończona.

Algorytm redukcji szumów – LSA

Za redukcję szumu z analizowanego sygnału mowy odpowiedzialny jest algorytm bazujący na estymatorze logarytmu widma mocy. Przejewiał on stosunkowo najlepsze właściwości z przebadanych algorytmów, dlatego też został wybrany do uruchomienia. Pierwszą rzeczą, jaką należy uczyć się, przystępując do odszumiania sygnału mowy, jest wybranie długości ramki czasowej. Ponieważ w pracy zostały użyte bufory próbek wejściowych do zapamiętania 128 wartości, uwzględniając nakładanie ramek w czasie w stosunku 50% ramka powinna mieć długość 256 wartości. Stosując częstotliwość próbkowania równą 8000 Hz, otrzymujemy 32-milisekundowe fragmenty przetwarzanej mowy. Powyższa wartość mieści się w obszarze stacjonarności sygnału mowy, tak więc została użyta w algorytmie odszumiającym.

Każdy obrabiany fragment sygnału powinien zostać zokienkowany w celu zmniejszenia skutków przecieku widma. Dobrym rozwiązaniem jest zastosowanie okna Hamminga, którego wartości zostały wstępnie przeliczone, a następnie umieszczone w pamięci mikroprocesora pod adresy **{HannEv,Hannod}**. Dodatkowo zdefiniowano dwa bufory pomocnicze **{Framebuffeven,Framebuffodd}** oraz **{Procbuffeven,Procbuffodd}** w celu zapamiętania połowy poprzedniej ramki zaszumionego oraz przetworzonego sygnału mowy. W pierwszym kroku starsza połowa poprzedniej ramki czasowej jest umieszczana na początku buforu **{refftb,imfta}**, a następnie zostaje dolożonych 128 nowych wartości z prawego kanału globalnego buforu wejściowego. Każdy element tak utworzonej ramki jest mnożony przez odpowiadający mu element funkcji okna, po czym następuje transformacja w dziedzinę częstotliwości:

$$\text{FFT-A}_{\text{real}}(1:129) \Leftarrow \Re \{\mathcal{F} \{\text{FFT-B}_{\text{real}} + j\text{FFT-A}_{\text{imag}}\}\}$$

$$\text{FFT-B}_{\text{imag}}(1:129) \Leftarrow \Im \{\mathcal{F} \{\text{FFT-B}_{\text{real}} + j\text{FFT-A}_{\text{imag}}\}\}$$

Długość FFT wynosi w tym przypadku 256, więc musi ona zostać poprzedzona skopiowaniem odpowiedniego szablonu FFT na stos.

Drugą czynnością jest pobranie i zapamiętanie fazy zaszumionego sygnału w celu późniejszego odtworzenia. Pomocna w tym celu okazuje się funkcja obliczająca arctan z ilorazu dwóch liczb, którą można odnaleźć pod adresem **atan2**. Wynik jest przechowywany w buforze FFT-A:

$$\text{FFT-A}_{\text{imag}}(1:129) \Leftarrow \Theta_k = \Theta \{\text{FFT-A}_{\text{real}} + j\text{FFT-B}_{\text{imag}}\}$$

Mając zapamiętaną fazę sygnału, można z czystym sumieniem wyznaczyć jego amplitudę:

$$\text{FFT-B}_{\text{real}}(1:129) \Leftarrow F(k) = |\{\text{FFT-A}_{\text{real}} + j\text{FFT-B}_{\text{imag}}\}|$$

$$\text{FFT-A}_{\text{imag}}(130:258) \Leftarrow F(k)^2 = |\{\text{FFT-A}_{\text{real}} + j\text{FFT-B}_{\text{imag}}\}|^2$$

Ażeby wyliczyć stosunek (sygnał+szum)–szum, wymagana jest znajomość aktualnej estymaty szumu. Jej odwrotność jest pobierana z tablicy o adresie **Nav**, która jest inicjalizowana wraz ze startem programu. W efekcie zaimplementowany algorytm nadaje się jedynie do uwydatniania sygnału

mowy zanurzonego w szumie stacjonarnym o znanej z góry charakterystyce widma mocy. W ewentualnym późniejszym rozwinięciu algorytmu można dodać funkcję wyznaczającą estymatę szumu na podstawie pomiaru aktualnego sygnału i umieszczającą jej odwrotność w tablicy o podanym adresie. Powyższy parametr jest obliczany następująco:

$$\text{FFT-B}_{\text{real}}(130:258) \Leftarrow \xi_o(k) = \frac{F(k)^2}{N(k)^2}$$

Kolejnym parametrem, jaki trzeba policzyć, jest wygładzony stosunek sygnał–szum aktualnej ramki:

$$\text{FFT-B}_{\text{real}}(259:387) \Leftarrow \xi_i(p, k) = \beta \frac{S(p-1, k)^2}{N(k)} + (1 - \beta) \text{MAX} \{\xi_o(k) - 1, 0\}$$

$S(p-1, k)$ oznacza przetworzoną amplitudę poprzedniej ramki czasowej. Po obliczeniu kolejnych dwóch parametrów:

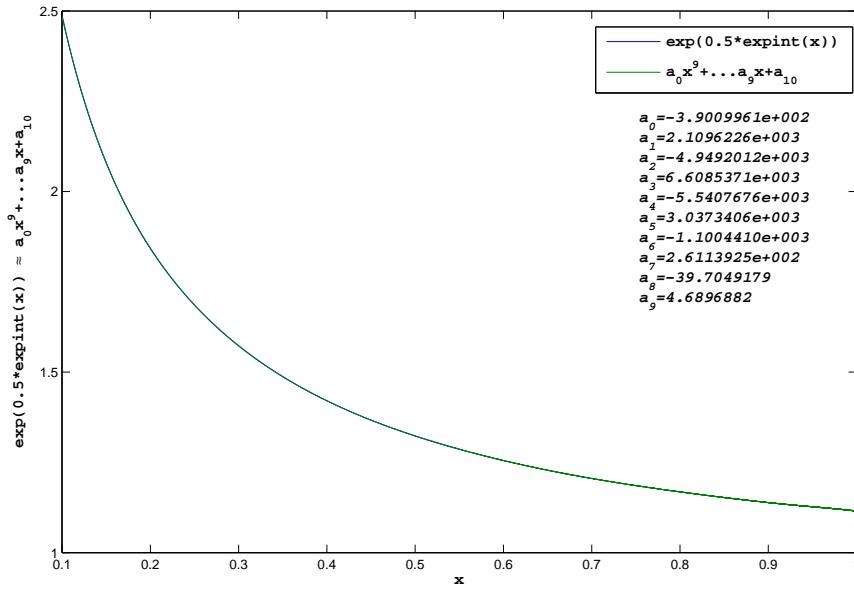
$$\text{FFT-A}_{\text{imag}}(259:387) \Leftarrow \gamma_k = \frac{\xi_i}{1 + \xi_i}$$

$$\text{FFT-B}_{\text{real}}(259:387) \Leftarrow v_k = \frac{\xi_i}{1 + \xi_i} \xi_o$$

można przystąpić do obliczenia wektora wzmacnienia:

$$\text{FFT-A}_{\text{imag}}(130:258) \Leftarrow G(k) = \frac{\xi_i}{1 + \xi_i} \exp \left(\frac{1}{2} \int_{v_k}^{\infty} \frac{e^{-t}}{t} dt \right)$$

Pozostaje jedynie przeliczyć niewinnie wygładzający czynnik eksponenty z całki wykładniczej.



RYS. 32 Przybliżenie eksponenty z całki wykładniczej za pomocą wielomianu 9 rzędu

Powyższą niedogodność złagodzono, aproksymując potrzebną funkcję za pomocą wielomianu dziewięciatego rzędu w każdej z dekad, począwszy od 0.0001 do 10. Rysunek 32 pokazuje właściwą

funkcję $\exp\left(\frac{1}{2} \int_{v_k}^{\infty} \frac{e^{-t}}{t} dt\right)$ wyznaczoną za pomocą rozwinięcia w szereg [40] oraz przybliżenie wielomianowe dla jednej z dekad. Dla argumentów większych od 10 przyjęto zwracaną wartość 1, dla argumentów mniejszych od 0.0001 przyjęto stałą wartość 75. Wartość wielomianu można w bardzo szybki sposób wyznaczyć, stosując tzw. schemat Hornera:

$$a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x^1 + a_n = (\dots ((a_0x + a_1)x + a_2)x + \dots)x + a_n$$

Procedura obliczająca wartość wielomianu potrzebuje $2N+1$ taktów zegarowych, gdzie N oznacza rząd wielomianu. Została ona tak skontruuowana, że wymaga umieszczenia w rejestrze **f1** współczynnika wielomianu stojącego przy najwyższej potędze oraz ustawienia wskaźnika **I0** na drugi współczynnik w tablicy umieszczonej w pamięci. W rejestrze **f0** przekazujemy argument wielomianu. Dodatkowo procedura wymaga ustawienia modyfikatora **m1** na wartość 1. Pobranie dwóch przekazywanych wartości do podprogramu może zostać wpisane do linii opóźnienia skoku. Rezultat zwracany jest poprzez rejestr **f1**.

1)	<i>lcntr=N-2, do polymul until lce;</i>	
2)	<i>f1=f1*f0,</i>	<i>f2=dm(i0,m1);</i>
3) polymul:	<i>f1=f1+f2;</i>	
4)	<i>f1=f1*f0,</i>	<i>f2=dm(i0,m1);</i>
5)	<i>rts(db),</i>	<i>f1=f1+f2;</i>
6)	<i>f1=f1*f0,</i>	<i>f2=dm(i0,m1);</i>
7)	<i>f1=f1+f2;</i>	

PROCEDURA 8. Wyznaczanie wartości wielomianu rzędu N

Przeprowadzając eksperyment z odszumianiem 4-minutowej sekwencji sygnału mowy, błąd średniokwadratowy uwydatnienia sygnału przy aproksymacji przeprowadzonej według opisanego algorytmu wyniósł $6 * 10^{-8}$, co potwierdza skuteczność zaprezentowanej metody.

Po wyznaczeniu wzmacnienia bardzo łatwo wyliczyć estymatę uwydatnionego sygnału:

$$S(p, k) = G(k)F(k)$$

Odtwarzając fazę:

$$\text{FFT-A}_{\text{real}}(1:129) \Leftarrow S(p, k) \cos(\Theta_k)$$

$$\text{FFT-B}_{\text{imag}}(1:129) \Leftarrow S(p, k) \sin(\Theta_k),$$

oraz dokonując odwrotnej dyskretnej transformaty Fouriera, w wyniku dostajemy uwydatnioną ramkę sygnału mowy w dziedzinie czasu.

Finalną czynnością jest nałożenie starszych 128 próbek sygnału świeżej ramki drugą połową próbek poprzedniej przetworzonej ramki, umieszczenie wyniku w globalnym buforze wyjściowym oraz zapamiętanie młodszych 128 próbek aktualnej uwydatnionej ramki czasowej.

Algorytm adaptacyjny ϵ -NLMS

Ostatnim zaimplementowanym algorytmem jest sławny ϵ -NLMS, uruchomiony w celu porównania procedury BEFAP z jednym z podstawowych i najczęściej stosowanych algorytmów adaptacyjnych. Kod programu zaczyna się od adresu ***nlms.alg***, procedura inicjalizacyjna od adresu ***nlms.init***. Algorytm ϵ -NLMS wymaga zarezerwowania w pamięci dwóch buforów o długości równej liczbie próbek filtru $N = 896$. Użyto w tym celu niewykorzystywanego podczas trwania procedury ϵ -NLMS buforu algorytmu BEFAP: ***Mainbuffodd*** w pierwszym bloku pamięci oraz dodatkowego buforu ***dline*** w drugim bloku pamięci. W ***dline*** przechowywane są próbki sygnału wejściowego w formie linii opóźniającej, skąd omawiany bufor jest zainicjalizowany jako cykliczny. W drugim buforze (także cyklicznym) spoczywają współczynniki filtru. Dla przypomnienia, równanie odświeżania współczynników przez algorytm ϵ -NLMS jest następujące:

$$w_i = w_{i-1} + \frac{\mu}{\epsilon + \|u_i\|^2} u_i^T [d(i) - u_i w_{i-1}].$$

Parametr regulujący ϵ umieszczany jest podczas inicjalizacji w rejestrze ***f11***, krok iteracji μ – w rejestrze ***f5***. Obliczenie energii zgromadzonej w wejściowych regresorach $\|u_i\|^2$, a potrzebnej w algorytmie ϵ -NLMS jest wyznaczane w sposób rekurencyjny:

$$\|u_i\|^2 = \|u_{i-1}\|^2 + u(i)^2 - u(i-N)^2.$$

Wejściowa próbka sygnału $u(i)$ przekazywana jest do procedury poprzez rejestr ***f0***, próbka sygnału pożądanego na wyjściu filtru – poprzez rejestr ***f1***. Rejestr ***I0*** wskazuje na początek linii opóźniającej, natomiast rejesty ***I8*** i ***I9*** – na początek wektora współczynników. Modyfikator ***m0*** otrzymuje wartość -1, zaś modyfikator ***m8*** otrzymuje wartość 1.

1)	<i>f14=f0*f0,</i>	<i>dm(i0,m0)=f0,</i>	<i>f4=pm(i8,m8);</i>
2)	<i>f8=f0*f4,</i>	<i>f0=dm(i0,m0),</i>	<i>f4=pm(i8,m8);</i>
3)	<i>f11=f11+f14,</i>		
4)	<i>f12=f0*f4,</i>	<i>f0=dm(i0,m0),</i>	<i>f4=pm(i8,m8);</i>
5)	<i>lcntr=N-3, do macs until lce;</i>		
6)	<i>f12=f0*f4,</i>	<i>f8=f8+f12,</i>	<i>f1=pm(m9,i8);</i>
7)	<i>f8=f1*f4,</i>	<i>f12=f8-f12,</i>	<i>f4=pm(i8,m8);</i>
8)	<i>f12=f0*f4,</i>	<i>f8=f8+f12,</i>	<i>f14=f11;</i>
9)	<i>f6=f1-f2,</i>	<i>f4=dm(i0,m0);</i>	
10)	<i>f7=f6*f5;</i>		
11–18)	<i>DIVIDE(f1,f7,f14,f9,f0);</i>		<i>f12=pm(i8,m8);</i>
19)	<i>f0=f1*f4,</i>		
20)	<i>lcntr=N-1, do upd_w until lce;</i>		
21)		<i>f8=f0+f12,</i>	<i>f4=dm(i0,m0),</i>
22)	<i>upd_w:</i>	<i>f0=f1*f4,</i>	<i>f12=pm(i8,m8);</i>
23)		<i>rts(db);</i>	<i>pm(i9,m8)=f8;</i>
24)		<i>f8=f0+f12,</i>	<i>f0=dm(i0,1);</i>
25)		<i>f13=f4*f4,</i>	<i>pm(i9,m8)=f8;</i>

W pierwszym rozkazie wyznaczana jest druga potęga wartości aktualnej próbki wejściowej oraz jest ona chowana do tablicy symulującej linię opóźniającą. Równocześnie pobierany jest pierwszy współczynnik filtru w_0 . W rozkazie drugim do wartości regresora z poprzedniej iteracji jest dodawana druga potęga aktualnej próbki wejściowej, wykonywane jest pierwsze mnożenie z iloczynu skalarnego $u_i w_{i-1}$ oraz następuje odczyt przeszłej próbki wejściowej filtra i współczynnika w_1 . Trzeci rozkaz kończy odświeżanie energii regresora, która jest przechowana w rejestrze **f11**. Pętla programowa o adresie końcowym **macs** wykonuje resztę operacji potrzebnych do wyznaczenia aktualnej próbki wyjściowej filtra, która finalnie zostaje wpisana do rejestru **f2** w rozkazie ósmym. W rozkazie dziewiątym do rejestru **f6** jest wpisywany błąd adaptacji (który pozostaje w nim do końca obliczeń), a rozkaz dziesiąty przeskala nową wartość regresora przez czynnik μ . Osiem rozkazów począwszy od jedenastego, wykonuje operację dzielenia przeskalowanego błędu adaptacji przez wartość energii aktualnego regresora, zaś zadaniem reszty instrukcji jest odświeżenie współczynników $w_{i-1} \rightarrow w_i$.

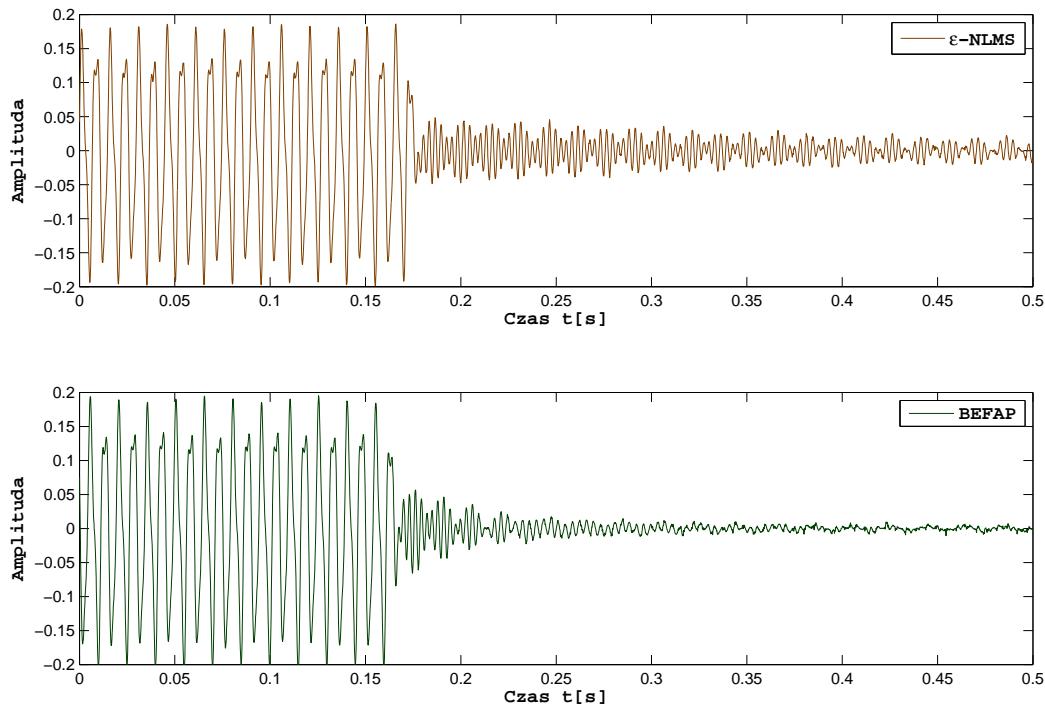
Kody źródłowe używanych procedur

Całość aplikacji z wszystkimi kodami źródłowymi oraz plikami inicjalizacyjnymi jest umieszczona na dołączonej do pracy płycie kompaktowej. Aplikacja jest zapisana jako plik projektu do odczytu w programie VisualDSP++. Pliki zarchiwizowano jednym z popularnych programów do kompresji i umieszczono na płycie pod nazwą **cner.rar**. Dodatkowo umieszczono elektroniczną wersję niniejszej pracy w postaci pliku **cner.pdf**.

6. Wyniki cyfrowego przetwarzania sygnałów przez procesor sygnałowy

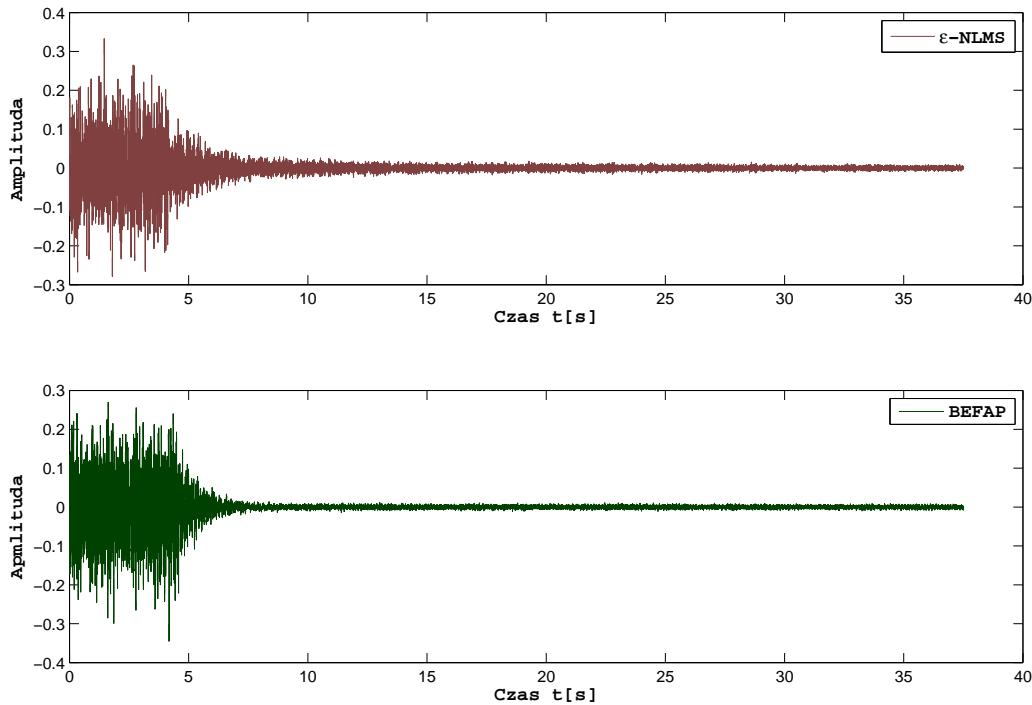
W poniższym paragrafie przedstawione są wyniki uzyskane za pomocą zaprezentowanych algorytmów. Wszystkie sygnały zostały zmierzone na wyjściu audio kodeka przy pobudzeniu ustalonym sygnałem wejściowym. Zostały przeprowadzone następujące doświadczenia:

1. Filtr adaptacyjny został pobudzony sygnałem wejściowym w postaci trzech tonów sinusoidalnych o częstotliwości 500 Hz, 400 Hz oraz 200 Hz. Powyższy sygnał został podany na prawy kanał przetwornika A/C. Ponadto na lewy kanał przetwornika podano trzytonowy sygnał sinusoidalny po przejściu przez filtr o skończonej odpowiedzi impulsowej równej 896 współczynnikom. Długość filtru adaptacyjnego ustalono również na 896, a rzad projekcji algorytmu BEFAP na 16. Rysunek 33 przedstawia wyniki adaptacji dla algorytmu ϵ -NLMS oraz BEFAP. Zaprezentowano wartości sygnału błędu na wyjściu filtru, zmierzone podczas włączania adaptacji.



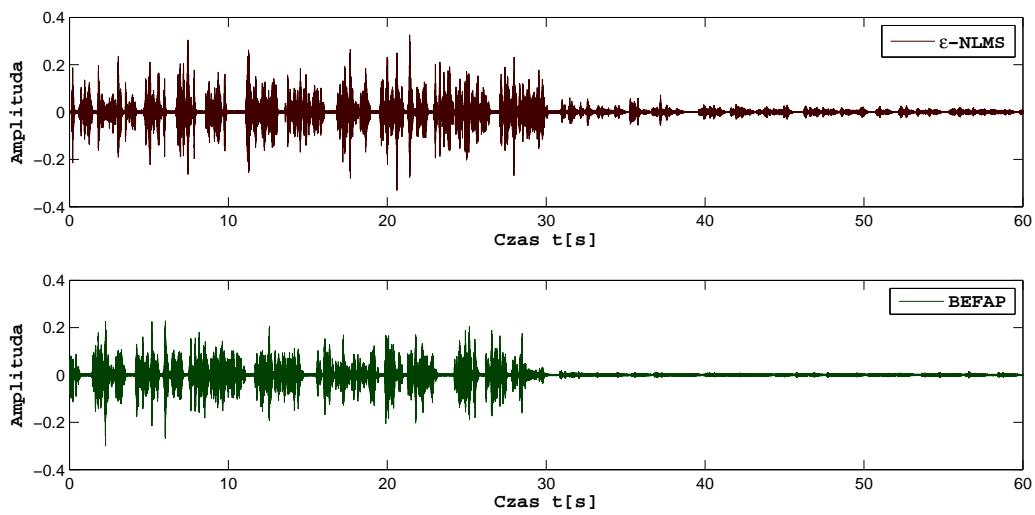
[a] RYS. 33 Eliminacja echo akustycznego w postaci trzytonowego sygnału sinusoidalnego
[b]

2. Filtr adaptacyjny został pobudzony sygnałem wejściowym w postaci szumu białego po filtracji dolnoprzepustowej, czego efektem jest szum o bardzo silnie skorelowanych próbkach i – co się z tym wiąże – dużym rozrzucie wartości własnych macierzy autokorelacji. Na wejście sygnału pożądanego filtru podano sygnał szumowy po przejściu przez filtr jak w punkcie 1. Rysunek 34 przedstawia wynik adaptacji po jej uruchomieniu dla algorytmów ϵ -NLMS oraz BEFAP.

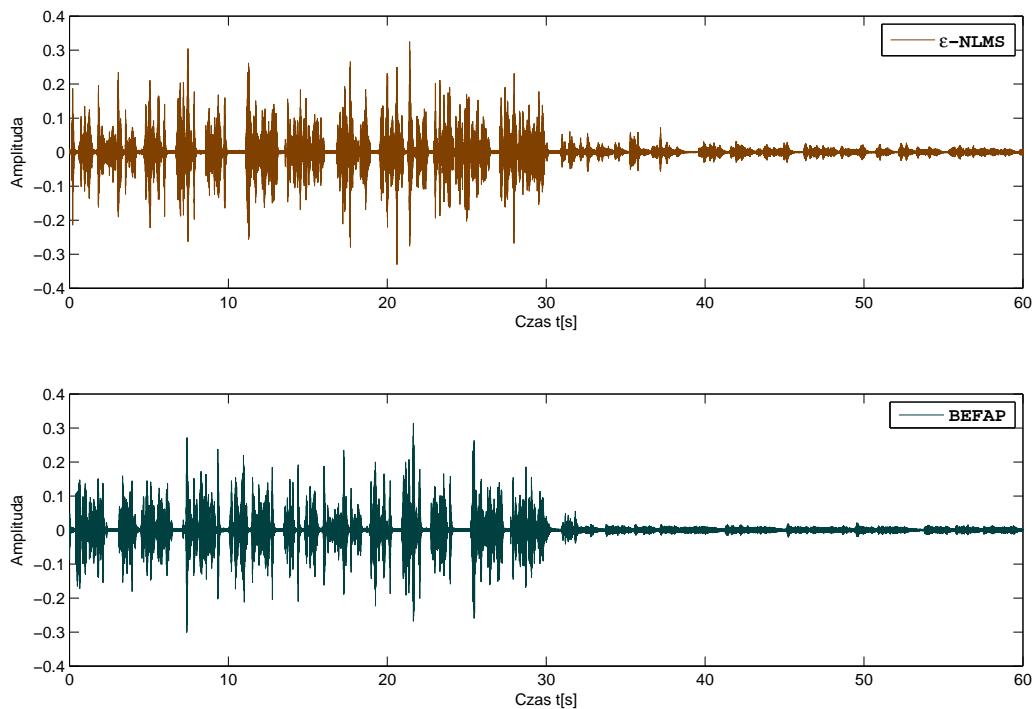


[a] RYS. 34 Eliminacja echo akustycznego w postaci szumu dolnoprzepustowego
 [b]

3. Filtr adaptacyjny został pobudzony sygnałem mowy zarejestrowanym wcześniej przez autora. Sygnał pochodzący na wyjściu filtra otrzymano poprzez filtrację sekwencji sygnału mowy przez filtr o skończonej odpowiedzi impulsowej symulujący ścieżkę echo. Do uzyskania potrzebnego sygnału użyto dwóch filtrów. Jednego o 896 współczynnikach, drugiego o 1792 współczynnikach. Ustalając długość filtru adaptacyjnego na 896, mamy w drugim przypadku możliwość badania zachowania filtru przy niedostatecznej jego długości. Ustalając czas rewerberacji na 224 milisekundy, maksymalne tłumienie echo w drugim przypadku może wynieść 30 dB. Wykresy dla obu długości filtru oraz dwóch algorytmów adaptacji są przedstawione na rysunkach 35 i 36.

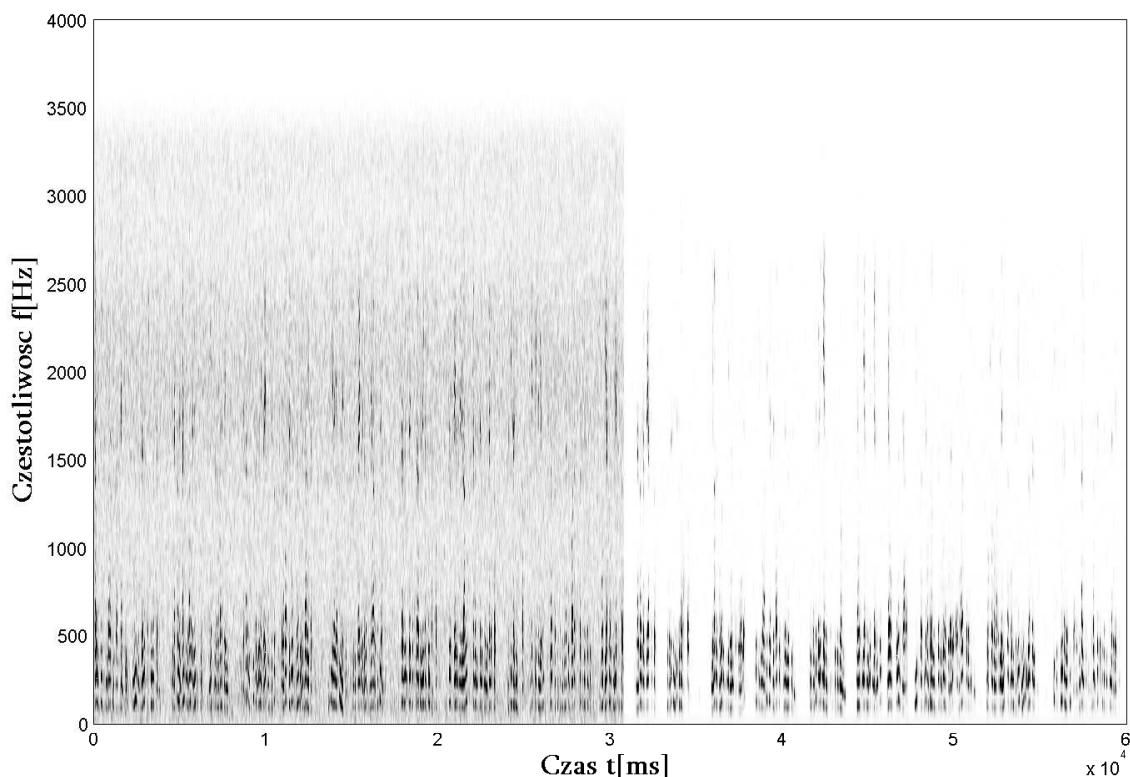


[a] RYS. 35 Eliminacja echo akustycznego w postaci sygnału mowy dla filtru adaptacyjnego o długości równej pełnej ścieżce echo
 [b]



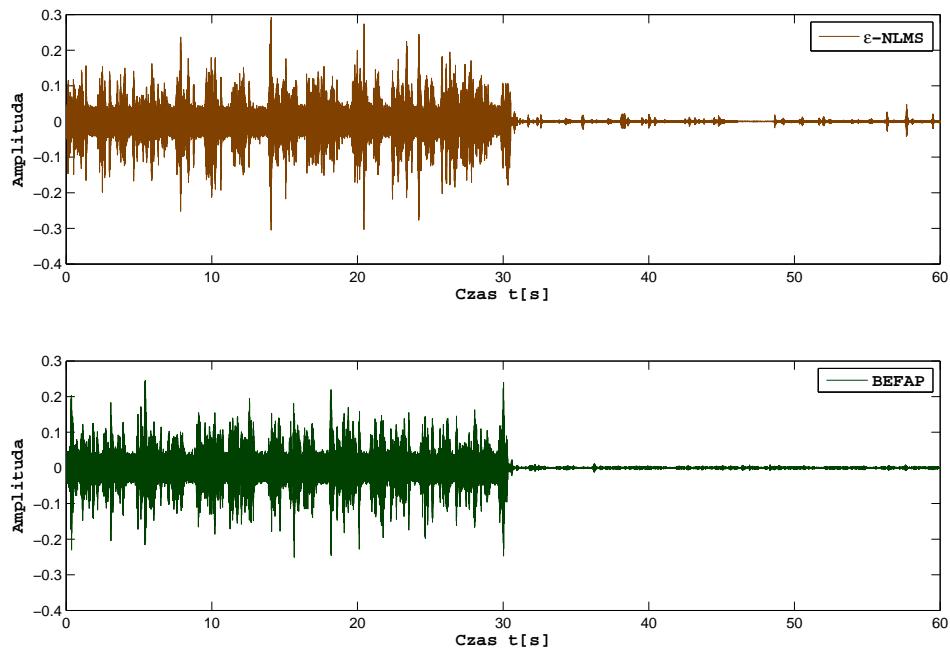
[a] RYS. 36 Eliminacja echo akustycznego w postaci sygnału mowy dla filtru adaptacyjnego o długości równej połowie pełnej ścieżki echo
 [b]

4. Na wejście audio kodeka podano sygnał mowy z nałożonym nań białym szumem o płaskiej charakterystyce widmowej. Tak spreparowany sygnał poddano następnie przetwarzaniu przez algorytm redukcji szumów – LSA. Odszumianiu poddana jest 60-sekundowa sekwencja mowy, przy czym reduktor został załączony po upływie połowy wspomnianego czasu. Spektrogram prezentujący działanie algorytmu zaprezentowany jest na rysunku 37.

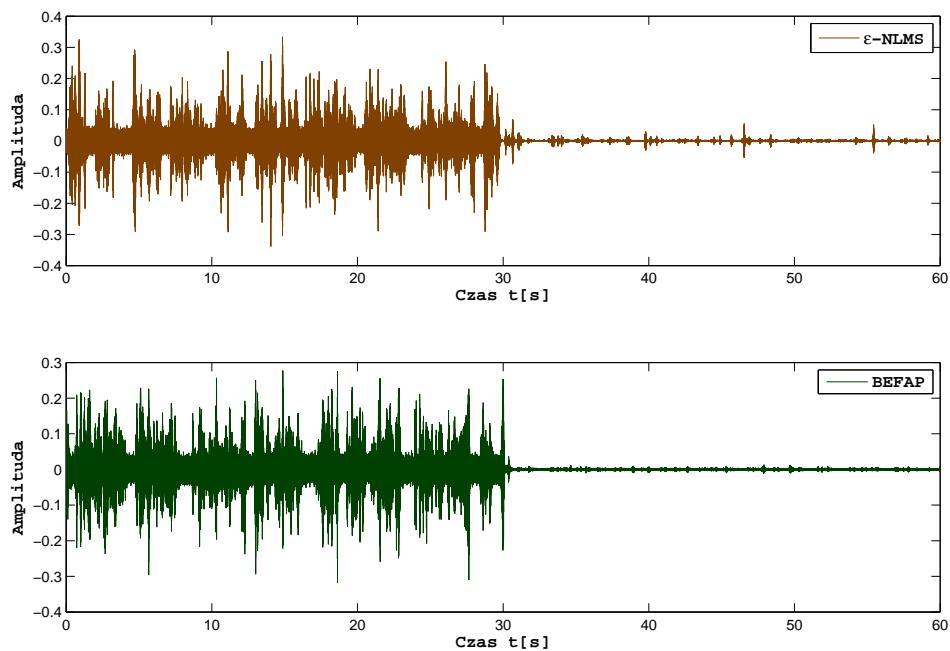


RYS. 37 Redukcja szumu z sygnału mowy za pomocą algorytmu LSA

5. Ostatnim zaproponowanym doświadczeniem było poddanie sekwencji sygnału mowy symulującego echo działaniu algorytmu łącznej redukcji szumów i echa akustycznego. Sygnał echa uzyskano podobnie jak w punkcie 3 zarówno dla pełnej, jak i połówkowej długości filtra. W pierwszym kroku algorytmu przeprowadzana jest eliminacja echa akustycznego za pomocą dwóch omawianych algorytmów. Po przetworzeniu i zredukowaniu w pewnym stopniu echa w zaszumionym sygnale jest on poddawany redukcji szumów z wykorzystaniem LSA. W efekcie takiej kolejności obliczeń powinniśmy otrzymać sygnał bez obu zakłóceń. Kolejne dwa rysunki prezentują wynik działania powyższej procedury.



[a] RYS. 38 Łączna redukcja szumów i echa akustycznego dla filtru adaptacyjnego o długości równej pełnej ścieżce echa
 [b]



[a] RYS. 39 Łączna redukcja szumów i echa akustycznego dla filtru adaptacyjnego o długości równej połowie ścieżki echa
 [b]

7. Podsumowanie

W niniejszej pracy zostały zaprezentowane, a następnie uruchomione algorytmy cyfrowej eliminacji echa akustycznego oraz redukcji szumów. Wśród algorytmów eliminacji echa wyróżnić możemy dwie główne rodziny: algorytmy minimalizujące błąd średniokwadratowy oraz algorytmy pracujące w oparciu o metodę najmniejszych kwadratów. Te pierwsze charakteryzują się stosunkowo niskimi wymaganiami sprzętowymi oraz dużą niezawodnością, wadą ich jest jednak spowolnienie zbieżności dla sygnałów o silnie skorelowanych próbkach, np. dla sygnałów mowy. Druga rodzina nie wykazuje żadnej zależności od statystyki przetwarzanych sygnałów, również adaptacja jest znacznie przyspieszona w porównaniu z algorytmami pierwszej rodziny. Za główną ich wadę uznaje się bardzo dużą złożoność obliczeniową, a wszystkie próby jej zmniejszenia w postaci szybkich algorytmów kończą się niestabilnością algorytmów w implementacjach z ograniczoną długością słowa przeznaczoną na zapamiętanie wyniku. Kompromisem wypracowanym na przestrzeni lat okazały się być tzw. uogólnione algorytmy stochastyczne – AP (Affine Projection). Można by uznać, że lokują się pośrodku, między dwiema rodzinami, oferując znacznie szybszą zbieżność niż standardowe algorytmy pierwszej rodziny kosztem zwiększenia złożoności obliczeniowej, pozostającej jednak ciągle o wiele łagodniejszą niż dla algorytów najmniejszych kwadratów.

Drugim aspektem cyfrowej eliminacji echa akustycznego jest potrzeba stosowania filtrów adaptacyjnych o bardzo dużej liczbie współczynników. Odpowiedią na to wymaganie jest rodzina algorytmów blokowych, oferująca dalsze redukcje wymagań sprzętowych kosztem wprowadzenia opóźnienia do przetwarzania sygnałów. Implementując wybrany algorytm w formie blokowej, otrzymujemy bardzo wydajny algorytm eliminacji echa akustycznego, przeznaczony do pracy z dużą ilością współczynników – BEFAP. Niestety dodatkowe czynności należało tak wykonać aby uczynić go stabilnym w środowisku obliczeniowym pojedynczej precyzji. Należało mianowicie zaimplementować wbudowaną w niego część obliczeniową bazującą na algorytmie najmniejszych kwadratów funkcjonującego w oparciu o tzw. predyktor kratowy. To pozwoliło następnie na uruchomienie powyższego algorytmu z wykorzystaniem procesora sygnałowego.

Algorytmy redukcji szumów są znacznie prostsze i pod względem koncepcyjnym, i złożoności obliczeniowej. Wykorzystują procedury szybkiej transformacji Fouriera w celu wyznaczenia widma przetwarzanego sygnału, następnie operując na nim, usuwają estymatę szumu, zostawiając sygnał pożądany. Również tutaj pojawiło się wiele odmian, m.in. algorytmy wykorzystujące właściwości aparatu słuchu człowieka czy algorytmy korzystające z praw statystyki w celu możliwie naj-optimalniejszej redukcji czynnika losowego. Przykładem jest zaimplementowany algorytm – LSA, czyniący założenie o Gaussowskim rozkładzie prawdopodobieństwa próbek widma sygnału oraz szumu. Kluczową sprawą do poprawnej pracy algorytmów redukcji szumu jest dokładna estymata szumu obecnego w kanale, co można uczynić na podstawie wielostopniowego uśredniania.

Jednostką obliczeniową, której użyto do wykonania wszystkich rachunków, był procesor sygnałowy firmy Analog Devices. Jest to specjalistyczny procesor o odpowiednio dobranej liście rozkazów w celu przetwarzania dużej ilości napływających danych. Stosując długie rozkazy, można uruchamiać równolegle kilka jednostek obliczeniowych, co dodatkowo zwiększa wydajność. Dzięki jego superharvardzkiej architekturze możemy – wykorzystując pamięć notatnikową wewnątrz procesora – sięgać dwa razy do pamięci w jednym taktie zegarowym. Dodatkowo bardzo wydajne i efektywne środowisko programistyczne znacznie ułatwia żmudny proces przygotowania aplikacji do rzeczywistej pracy.

Po zaimplementowaniu, uruchomieniu i przebadaniu omówionych algorytmów potwierdziły się przewidywania teoretyczne. Algorytm BEFAP wykazuje o wiele lepsze właściwości od prostego ϵ -NLMS zarówno pod względem osiąganej szybkości zbieżności, jak też dokładności estymacji echa. O jednej jednak rzeczy należy wspomnieć: otóż algorytm BEFAP wymagał znacznie wyższego parametru regulującego, aby zapewnić stabilność numeryczną, niż algorytm ϵ -NLMS, co już nie przemawia na jego korzyść. Kolejną sprawą do rozważenia jest fakt, że algorytm BEFAP dla bardzo dużej liczby współczynników ma zdecydowanie mniejsze wymagania sprzętowe niż nawet tak prosty algorytm jak ϵ -NLMS. Z drugiej jednak strony wymaga znacznie więcej pamięci, choć ilość pa-

mięci łatwo można zwiększyć, zaś przyspieszyć zegar taktowania jest o wiele trudniej. Z kolei dzięki dokładnej estymacie szumu, jaka została użyta w doświadczeniach, algorytm LSA wykazał również bardzo dobre właściwości, usuwając praktycznie cały szum szerokopasmowy i pozostawiając niewiele znieksztalceń w postaci szumu muzycznego.

Należy podkreślić, że algorytm łącznej redukcji szumów i echa akustycznego również działa bardzo dobrze, usuwa bowiem znaczną większość zakłóceń. Na uwagę zasługuje przypadek zdecydowanie bliższy rzeczywistości: gdy mamy do czynienia z filtrem adaptacyjnym o liczbie współczynników mniejszej niż rzeczywista długość ścieżki echa. Jak się można spodziewać, na wyjściu otrzymujemy większe echo pozostałościowe. Rysuje się już jedna z barier stawiana eliminatorem echa akustycznego, mianowicie niepełne odwzorowanie funkcji otoczenia prowadzące w efekcie do większego błędu na jego wyjściu.

Innym zgoła czynnikiem degradującym zachowanie eliminatorów echa jest specyfika sygnału mowy. Mianowicie obecność sygnału dużej mocy przeplatana jest na przemian odstępami ciszy. Podczas takich przerw adaptacja w najlepszym wypadku zostaje wstrzymana, a co gorsza: pojawiające się zakłócenia szumowe dodatkowo pomniejszają nabycą wiedzę. To jest właśnie główna przyczyna stosowania dużych współczynników regulujących przy przetwarzaniu sygnałów mowy.

Podsumowując, główny cel pracy został osiągnięty. Uruchomiono algorytm łącznej redukcji szumów i echa akustycznego, bazując na dwóch algorytmach adaptacji: ϵ -NLMS i BEFAP, oraz jednym algorytmie odszumiającym – LSA. Zaprezentowane wyniki pomiarów pokazują dobrą pracę algorytmów w warunkach laboratoryjnych.

Na koniec warto w kilku słowach wspomnieć o możliwych ulepszeniach zastosowanego algorytmu. Po pierwsze możnaby dodać procedurę wykrywania stanu tzw. „podwójnej mowy” na wejściu eliminatora echa. Sytuacja taka zachodzi, gdy oprócz echa akustycznego zmieszanego z szumem tła, na wejściu filtra adaptacyjnego pojawia się sygnał bliskiego mówcy. Jest on traktowany przez filtr jako bardzo silny sygnał zakłócający i może w efekcie doprowadzić do rozbieżności algorytmu adaptacyjnego. Dobrym algorytmem w celu wykrycia takiej sytuacji jest algorytm, bazujący na wyliczeniu tzw. znormalizowanego wektora współczynników korelacji krzyżowej pomiędzy sygnałem wejściowym filtra adaptacyjnego a sygnałem echa zmiksowanym z silnym sygnałem zakłócającym [7]. Za pomocą wyliczonego wektora korelacji tworzy się pewną zmienną decyzyjną porównywaną z przyjętym poziomem odniesienia. W razie przekroczenia przyjętego kryterium krok iteracji algorytmu adaptacyjnego jest zerowany w celu zatrzymania adaptacji filtru.

W przypadku algorytmów redukcji szumów istnieje możliwość udoskonalenia algorytmów angażujących metody statystyczne. Jednym z pomysłów jest zastosowanie zmiennej potęgi w estymatorze amplitudy widma, adaptowanej w zależności od aktualnego stosunku sygnał-szum w analizowanej ramce [57]. Skutkuje to zmniejszeniem znieksztalceń przetwarzanej mowy, szczególnie fragmentów o stosunkowo małej mocy, utrzymując wartość usuniętego szumu na założonym poziomie. Inne pomysły, to próby dokładniejszego wyznaczenia właściwości statystycznych sygnału mowy. Zamiast modelować współczynniki części rzeczywistej i urojonej widma za pomocą rozkładu Gaussa wprowadzono dokładniejszy model w postaci rozkładu Laplace'a, a także rozkładu Γ [12, 14, 32]. Na koniec warto wspomnieć o zabiegu, który uczyniłby zaimplementowaną aplikację jeszcze bardziej uniwersalną. Mowa mianowicie o estymacie aktualnego szumu w kanale podczas pracy aplikacji w czasie rzeczywistym. Bardzo dobrze do tego celu nadają się procedury bazujące na uśrednianiu kolejnych periodogramów sygnału zaszumionego i zapamiętujące ich wartości minimalne. Z uwagi na fakt, że tak wyestymowany szum leży zdecydowanie poniżej swojej rzeczywistej wartości średniej, aplikuje się korekcję [31]. Metody te mają tę zaletę, że nie wymagają stosowania detektorów aktywności sygnału mowy, co czyni je układami łatwymi do zaprogramowania i posiadającymi dobre parametry.

8. Literatura

- [1] S.T. ALEXANDER, „Fast Adaptive Filters: A Geometrical Approach”, IEEE ASSP Magazine, October 1986.
- [2] „ADSP-21061 EZ-KIT Lite Evaluation System Manual”, Analog Devices Inc. January 2003.
- [3] „ADSP-2106x SHARC Processor User’s Manual”, Analog Devices Inc. March 2004.
- [4] „VisualDSP++ 4.0 Assembler and Preprocessor Manual”, Analog Devices Inc.
- [5] „VisualDSP++ 4.0 Getting Started Guide”, Analog Devices Inc. January 2005.
- [6] „VisualDSP++ 4.0 Linker and Utilities Manual”, Analog Devices Inc.
- [7] J. BENESTY, D.R. MORGAN, J.H. CHO, „A New Class of Doubletalk Detectors Based on Cross-Correlation”, IEEE Transactions On Speech And Audio Processing, vol. 8, no. 2, March 2000.
- [8] M.BEROUTI, R.SCHWARTZ, J.MAKHOUL, „Enhancement of speech corrupted by acoustic noise”, Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing, 1979, pp. 208-211.
- [9] S.F.BOLL, „A Spectral Subtraction Algorithm for Suppression of Acoustic Noise in Speech”, Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing, 1979, pp. 200-203.
- [10] S.F.BOLL, „Suppression of Acoustic Noise in Speech Using Spectral Subtraction”, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-27, no.2, April 1979.
- [11] C. BREINING, P. DREISEITEL, E. HANSLER, A. MADER, B. NITSCH, H. PUDE, T. SCHERTLER, G. SCHMIDT, J. TILP, „Acoustic Echo Control An Application of Very-High-Order Adaptive Filters”, IEEE Signal Processing Magazine, July 1999.
- [12] C. BREITHAUPT, R. MARTIN, „MMSE Estimation Of Magnitude-Squared DFT Coefficients With Supergaussian Priors”, IEEE Proc. Intern. Conf. on Acoustics, Speech and Signal Processing, vol. I, pp. 896-899, April 2003.
- [13] O.CAPPE, „Elimination of the Musical Noise Phenomenon with the Ephraim and Malah Noise Suppressor”, IEEE Transactions on Speech and Audio Processing, vol. 2, pp. 345-349, April 1994.
- [14] B. CHEN, P. LOIZOU, „Speech Enhancement Using A MMSE Short Time Spectral Amplitude Estimator With Laplacian Speech Modeling”, Proceedings of ICASSP-2005, Philadelphia, PA, May 2005.
- [15] J.M. CIOFFI, T. KAILATH, „Windowed Fast Transversal Filters Adaptive Algorithms with Normalization”, IEEE Transactions On Acoustic, Speech And Signal Processing vol. ASSP-33, no. 3, June 1985.
- [16] L. COHEN, „The History of Noise”, IEEE Signal Processing Magazine, November 2005.
- [17] Y.EPHRAIM, D.MALAH, „Speech Enhancement Using a Minimum Mean-Square Error Log-Spectral Amplitude Estimator”, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-33, no. 2, April 1985.
- [18] Y.EPHRAIM, D.MALAH, „Speech Enhancement Using a Minimum Mean-Square Error Short-Time Spectral Amplitude Estimator”, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-32, no. 6, December 1984.
- [19] B. FARHANG-BOROUJENY, „Adaptive Filters Theory and Applications”, John Wiley & Sons 1998.
- [20] S.L. GAY, S. TAVATHIA, „The Fast Affine Projection Algorithm”, Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP ’95), vol. 5, pp. 3023-3026, Detroit, Mich, USA, May 1995.

- [21] G.O. GLENTIS, K. BERBERIDIS, S. THEODORIDIS, „Efficient Least Squares Adaptive Algorithms For FIR Transversal Filtering”, IEEE Signal Processing Magazine, July 1999.
- [22] G.H. GOLUB, C.F. VAN LOAN, „Matrix Computations 3rd. Ed.”, The Johns Hopkins University Press, 1996.
- [23] Y. GUELOU, A. BENAMAR, P. SCALART, „Analysis Of Two Structures For Combined Acoustic Echo Cancellation And Noise Reduction”, ICASSP-96, IEEE International Conference on Acoustics, Speech, and Signal Processing, 1996.
- [24] S. HAYKIN, „Adaptive Filter Theory 3rd. Ed.”, Prentice Hall 1995.
- [25] J.D. JOHNSTON, „Transform Coding of Audio Signals Using Perceptual Noise Criteria”, IEEE Journal on Selected Areas in Communications, vol. 6, no. 2, February 1988.
- [26] J.S.LIM, A.V.OPPENHEIM, „Enhancement and Bandwidth Compression of Noisy Speech”, Proc. IEEE, vol. 67, no. 12, December 1979.
- [27] M. KAHRS, K. BRANDENBURG, „Applications Of Digital Signal Processing To Audio And Acoustics”, Kluwer Academic Publishers 2002.
- [28] R. LE BOUQUIN JEANNES, G. FAUCON, „How to Reduce the Noise Influence, in a Joint System Developed for Echo and Noise Cancellation”, IEEE Signal Processing Letters, vol. 4, no. 10, October 1997.
- [29] R. LE BOUQUIN JEANNES, P. SCALART, G. FAUCON, C. BEAUGEANT, „Combined Noise and Echo Reduction in Hands-Free Systems: A Survey”, IEEE Transactions On Speech And Audio Processing, vol. 9, no. 8, November 2001.
- [30] M.LORBER, R. HOELDRICH, „A Combined Approach for Broadband Noise Reduction”, Proc. IEEE WASPAA, 1997, Session 8: Noise Reduction.
- [31] R. MARTIN, „Noise Power Spectral Density Estimation Based on Optimal Smoothing and Minimum Statistics”, IEEE Transactions on Speech and Audio Processing, vol. 9, no. 5, July 2001.
- [32] R. MARTIN, „Speech Enhancement Using MMSE Short Time Spectral Estimation With Gamma Distributed Speech Priors”, Proc. IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 253-256, 2002.
- [33] S.M.MCOLASH, R.J. NIEDERJOHN, J.A. Heinen, „A Spectral Subtraction Method for the Enhancement of Speech Corrupted by Non-White, Non-Stationary Noise”, Proceedings of the 1995 IEEE 21st International Conference on Industrial Electronics. Control and Instrumentation, Orlando, Florida, pp. 872-877, November 6-10, 1995.
- [34] E. MOULINES, O.A. AMRANE, Y. GRENIER, „The Generalized Multidelay Adaptive Filter: Structure and Convergence Analysis”, IEEE Transactions On Signal Processing, vol. 43, no. 1, January 1995.
- [35] R.J. NIEDERJOHN, J.A. HEINEN, „Understanding Speech Corrupted By Noise”, Proceedings of the IEEE International Conference on Industrial Technology, Shanghai, China, pp. 1-5, December 2-6, 1996.
- [36] T. PAINTER, A. SPANIAS, „Perceptual Coding of Digital Audio”, Proc. of the IEEE, vol. 88, no. 4, April 2000.
- [37] A. PAPOULIS, „Prawdopodobieństwo, zmienne losowe i procesy stochastyczne”, WNT Warszawa 1972.
- [38] J. PORUBA, „Speech Enhancement Based on Nonlinear Spectral Subtraction”, Fourth IEEE International Caracas Conference on Devices, Circuits and Systems, Aruba April 2002.
- [39] A.D. POULARIKAS, „The Transforms And Applications Handbook, 2nd. Ed.”, CRC Press 2000.

- [40] W.H. PRESS, S.A. TEUKOLSKY, W.T. VETTERLING, B.P. FLANNERY, „Numerical Recipies in C The Art of Scientific Computing 2nd. Ed.”, Cambridge University Press 1992.
- [41] L.R. RABINER, R.W. SCHAFER, „Digital Processing of Speech Signals”, Prentice Hall 1978.
- [42] C.M. RADER, A.O. STEINHARDT, „Hyperbolic Householder Transformations”, IEEE Transactions On Acoustics, Speech And Signal Processing, vol. ASSP-34, no. 6, December 1986.
- [43] A.H. SAYED, „Fundamentals of Adaptive Filtering”, John Wiley & Sons 2003.
- [44] P. SCALART, A. BENAMAR, „A system for speech enhancement in the context of hands-free radio-telephony with combined noise reduction and acoustic echo cancellation”, Speech Communication 20 (1992), pp. 203-214.
- [45] G.U. SCHMIDT, „Acoustic Echo and Noise Control for Low-Cost Processors”, DSP World Spring Desing Conference 2000, April 2000.
- [46] J.S. SOO, K.K. PANG, „Multidelay Block Frequency Domain Adaptive Filter”, IEEE Transactions On Acoustics, Speech And Signal Processing, vol. 38, no. 2, February 1990.
- [47] H.V. SORENSEN, D.L. JONES, M.T. HEIDEMAN, C.S. BURRUS, „Real-Valued Fast Fourier Transform Algorithms”, IEEE Transactions On Acoustics, Speech And Signal Processing, vol. ASSP-35, no. 6, June 1987.
- [48] M. TANAKA, S. MAKINO, J. KOJIMA, „A Block Exact Fast Affine Projection Algorithm”, IEEE Transactions On Speech And Audio Processing, vol. 7, no. 1, January 1999.
- [49] R.M.UDREA, S.CIOCHINA, D.N. VIZIREANU, „Multi-band Bark Scale Spectral Over-Subtraction for Colored Noise Reduction”, International Symposium on Signals, Circuits and Systems, vol. 1, Issue 14-15, pp. 311-314, July 2005.
- [50] R.M.UDREA, S.CIOCHINA, D.N. VIZIREANU, „Reduction of Background Noise from affected Speech using a Spectral Subtraction Algorithm based on Masking Properties of the Human Ear”, 7th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services, pp. 135-138, 2005.
- [51] R.M. UDREA, S.CIOCHINA, „Speech Enhancement Using Spectral Oversubtraction and Residual Noise Reduction”, International Symposium on Signals, Circuits and Systems, Vol. 1, pp. 165-168, 2003.
- [52] S.V.VASEGHI, „Advanced Digital Signal Processing and Noise Reduction”, Second Edition. John Wiley & Sons LTd 2000.
- [53] N.VIRAG, „Single Channel Speech Enhancement Based on Masking Properties of the Human Auditory System”, IEEE Transactions on Speech and Audio Processing, vol. 7, no. 2, March 1999.
- [54] N.VIRAG, „Speech Enhancement Based on Masking Properties of the Auditory System”, Proceedings International Conference on Acoustics, Speech, and Signal Processing, Detroit, USA, May 1995.
- [55] D.L.WANG, J.S.LIM, „The Unimportance of Phase in Speech Enhancement”, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-27, no. 2, pp. 679-681, 1982.
- [56] B. WIDROW, „Thinking About Thinking: The Discovery of the LMS Algorithm”, IEEE Signal Processing Magazine, January 2005.
- [57] C.H. YOU, S.N. NGEE, S. RAHARDJA, „ β -Order MMSE Spectral Amplitude Estimation for Speech Enhancement”, IEEE Transactions On Speech And Audio Processing, vol. 13, no. 4, July 2005.
- [58] K. ZHAO, F. LING, H. LEV-ARI, J.G. PROAKIS, „Sliding Window Order-Recursive Least-Squares Algorithms”, IEEE Transactions On Signal Processing, vol. 42, no. 8, August 1994.

Spis rysunków

1	Mechanizm powstawania echa akustycznego wraz z ingerencją szumu.	2
2	Schemat filtra adaptacyjnego	3
3	Eliminacja echa akustycznego z wykorzystaniem filtru adaptacyjnego.	3
4	Krzywe błędu algorytmów LMS i ϵ -NLMS dla sygnału „białego” i „kolorowego”	7
5	Krzywe błędu algorytmów ϵ -APA różnych rzędów	8
6	Krzywe błędu algorytmów adaptacyjnych w dziedzinie transformacji	10
7	Schemat blokowy szybkiego splotu w dziedzinie częstotliwości.	11
8	Porównanie algorytmów MDF z korekcją i bez dla implementacji z wykorzystaniem różnych transformacji	12
9	Porównanie algorytmów RLS i LMS dla sygnałów „białych” i „kolorowych”.	16
10	Algorytm RLS w środowisku obliczeniowym skończonej precyzji	17
11	Macierzowa implementacja algorytmu RLS.	19
12	Szybkie algorytmy RLS w środowisku o skończonej precyzji	22
13	Widmowe odejmowanie amplitudowe.	25
14	Absolutny poziom słuchu człowieka w zależności od częstotliwości	27
15	Widmo mocy przykładowej ramki sygnału mowy z wyznaczonym poziomem maskowania.	29
16	Uwydatnianie sygnału mowy z szumem białym	31
17	Uwydatnianie sygnału mowy z szumem dolnoprzepustowym.	32
18	Uwydatnianie sygnału mowy z szumem pasmowym	32
19	Uwydatnianie sygnału mowy z szumem niestacjonarnym.	33
20	Algorytm łącznej redukcji szunów i echa akustycznego; struktura z przetwarzaniem wstępny	34
21	Zamodelowane odpowiedzi impulsowe otoczenia.	35
22	Wyniki przetwarzania struktury łącznej redukcji szumów i echa akustycznego z przetwarzaniem wstępny	36
23	Współczynnik tłumienia powracającego echa: ERLE.	36
24	Przykładowa zawartość rejestrów danych procesora sygnałowego	39
25	Przykładowa zawartość generatorów adresów	40
26	Przerwania procesora sygnałowego ADSP-21061	41
27	Rozmieszczenie komponentów na płycie EZ-KIT Lite firmy Analog Devices.	43
28	Przykładowa zawartość pamięci w bloku pierwszym i drugim w postaci 32-bitowych liczb zmiennoprzecinkowych	48
29	Procesy zachodzące podczas zbierania informacji wejściowej.	62
30	Wartości wskaźników do buforów audio tuż po inicjalizacji	63
31	Bufory pamięciowe procedury BEFAP.	72
32	Przybliżenie eksponenty z całki wykładniczej za pomocą wielomianu 9 rzędu	77
33	Eliminacja echa akustycznego w postaci trzytonowego sygnału sinusoidalnego.	81
34	Eliminacja echa akustycznego w postaci szumu dolnoprzepustowego	82

35	Eliminacja echa akustycznego w postaci sygnału mowy dla filtru adaptacyjnego o długości równej pełnej ścieżce echa	83
36	Eliminacja echa akustycznego w postaci sygnału mowy dla filtru adaptacyjnego o długości równej połowie pełnej ścieżki echa	83
37	Redukcja szumu z sygnału mowy za pomocą algorytmu LSA	84
38	Łączna redukcja szumów i echa akustycznego dla filtru adaptacyjnego o długości równej pełnej ścieżce echa	85
39	Łączna redukcja szumów i echa akustycznego dla filtru adaptacyjnego o długości równej połowie pełnej ścieżki echa	85

Spis tabel

1	Porównanie nakładów obliczeniowych dla algorytmów MDF w zależności od długości bloku	12
2	Nakład obliczeniowy szybkich algorytmów RLS	21
3	Rozkład wyidealizowanych pasm krytycznych rozpiętych w paśmie akustycznym	28
4	Mapa pamięci płyty EZ-KIT Lite z ustanowieniem 32-bitowego dostępu do pamięci w bloku drugim	44
5	Inicjalizacja parametrów początkowych predyktora kratowego RLS	69

Spis algorytmów

1	BEFAP – blokowy szybki ϵ -APA z korekcją sygnału wyjściowego	55
2	SWFARLS – szybki macierzowy algorytm RLS z chodzącym oknem	57
3	Szybki macierzowy algorytm RLS z chodzącym oknem z błędami predykcji wyznaczonymi pomocą filtru kratowego z rozkładem QRD	59

Spis procedur

1	Badanie stanu aplikacji	64
2	Odświeżanie wektora autokorelacji	66
3	Dzielenie liczb zmienoprzecinkowych w postaci makra	67
4	Predyktor kratowy RLS w języku asemblera procesora sygnałowego.	70
5	Mnożenie odpowiadających elementów tablic liczb zespolonych	73
6	Wyznaczanie współczynnika korekcyjnego próbki wyjściowej filtru	74
7	Odświeżenie wektora błędu procedury BEFAP	75
8	Wyznaczanie wartości wielomianu rzędu N	78
9	Algorytm ϵ -NLMS	79