Zastosowania procesorów sygnałowych

FILTRY CYFROWE

na procesorach sygnałowych

Opracowanie: Grzegorz Szwoch

Politechnika Gdańska, Katedra Systemów Multimedialnych

Wprowadzenie

- Filtry cyfrowe należą do podstawowych algorytmów implementowanych na procesorach sygnałowych.
- Typowe praktyczne zastosowania:
  - redukcja (filtracja) zakłóceń,
  - wybór składowych widmowych do analizy.
- Stosowane są dwa typy filtrów:
  - o skończonej odpowiedzi impulsowej – FIR („splotowe”),
  - o nieskończonej odpowiedzi impulsowej – IIR (rekursywne).
- Oba typy mają praktyczne zastosowania.
- Podstawy teoretyczne filtrów FIR i IIR oraz metody ich projektowania zostały zaprezentowane na wykładach z *Przetwarzania dźwięków i obrazów*, więc nie będą tutaj powtarzane.

Projektowanie filtrów cyfrowych

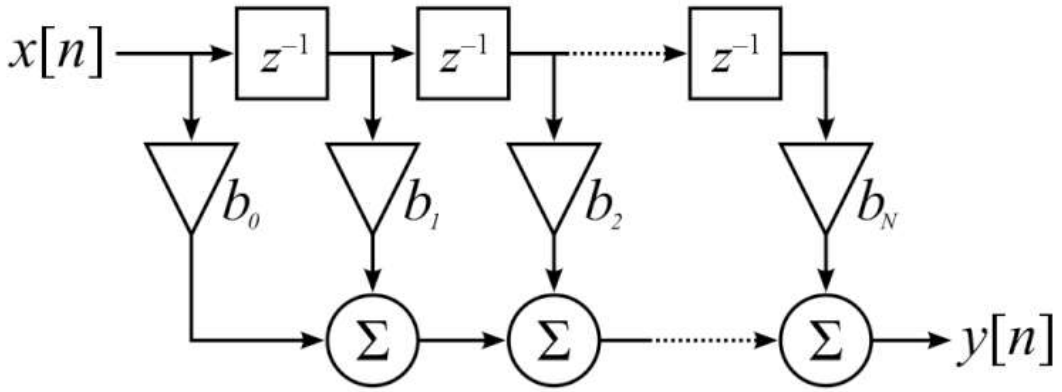
1. Wybór typu filtru: skończona / nieskończona odpowiedź impulsowa (FIR/IIR).
2. Wybór typu charakterystyki, np. filtr dolnoprzepustowy.
3. Wybór częstotliwości granicznej filtracji, np. 3,2 kHz.
4. Wybór długości / rzędu filtru lub szerokości pasma przejściowego.
5. Wybór minimalnego tłumienia i wielkości zafalowań (filtry IIR).
6. Obliczenie współczynników filtru za pomocą programu komputerowego.
7. Sprawdzenie charakterystyk filtru – czy spełniają założenia projektowe.
8. Dla stałoprzecinkowych procesorów: konwersja współczynników na Q15.
9. Zapisanie współczynników w kodzie programu lub w zewnętrznym pliku.
10. Napisanie kodu filtracji lub użycie istniejącej procedury.

Filtry FIR

Filtry o skończonej odpowiedzi impulsowej – FIR (*finite impulse response*):

- zapamiętanie *N* ostatnich próbek sygnału w buforze (*N* jest długością filtru),
- przemnożenie każdej próbki z bufora przez odpowiadający jej współczynniki filtru,
- sumowanie wyników mnożenia – wynik jest wynikiem filtracji dla bieżącej próbki.

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + \ldots + b_N x(n-N)$$




```
y = 0
FOR i = 0 TO N-1:
      y = y + x[i] * b[i]
RETURN y
```

Projektowanie filtru FIR

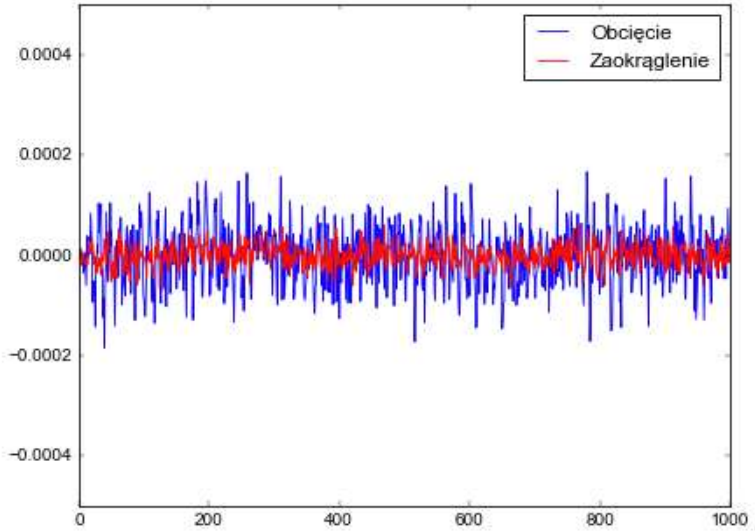
- Zakładamy jak ma wyglądać charakterystyka filtru:
  typ (np. dolnoprzepustowy), częstotliwości graniczne (np. 4 kHz),
  minimalne tłumienie (np. 40 dB), szerokość pasma przejściowego (np. 200 Hz).
- Obliczamy współczynniki filtru (zmiennoprzecinkowe)
  za pomocą oprogramowania (np. Matlab, Python/SciPy),
  metody: np. okienkowa lub Parksa-McClellana.
- Sprawdzamy wzmocnienie filtru – w paśmie przepustowym powinno być równe 1 lub nieco mniejsze. Jeżeli nie jest – normalizujemy.
- Współczynniki filtru zapisujemy w kodzie, zwykle w stałej tablicy.
- Szczegółowe informacje na temat projektowania filtrów: wykład PDiO.

Współczynniki dla stałoprzecinkowego DSP

- Współczynniki filtru obliczone przez program są zapisane jako liczby zmiennoprzecinkowe.
- Jeżeli używamy stałoprzecinkowego DSP, musimy przekształcić je do formatu Q15.
- Mnożymy współczynniki przez 32768, po czym zaokrąglamy je do najbliższej liczby całkowitej (np. instrukcja *round* w Matlabie).
- Powinniśmy sprawdzić wzmocnienie filtru w paśmie przepustowym – nie może ono przekraczać zakresu.
  - Np. dla filtru dolnoprzepustowego: suma współczynników nie powinna przekraczać 32767.
  - Jeżeli przekracza, możemy mnożyć współczynniki przez nieco mniejszą liczbę, np. przez 32760.

Kwantyzacja współczynników

- Przekształcając współczynniki do liczb Q15 przeprowadzamy kwantyzację – zamieniamy rzeczywiste współczynniki na najbliższą liczbę mającą reprezentację w zapisie Q15.
- Przykład: 0,327635 · 32767 = 10735,616045; zaokrąglamy: 10736; różnica 0,38395 stanowi błąd kwantyzacji.
- Różnica wyniku działania oryginalnego i stałoprzecinkowego filtru ma charakter szumu kwantyzacji.
- Uzyskany filtr jest inny, niż zadany podczas projektowania.
- W procesorach zmiennoprzecinkowych efekt kwantyzacji również występuje, ale jest on znacznie mniejszy niż w stałoprzecinkowych.



Zapis współczynników w kodzie C

- Współczynniki zapisujemy w postaci stałej tablicy (*const*).
- Tablicę umieszczamy globalnie (poza funkcjami).
- Jeżeli współczynników jest dużo (wiele filtrów, duża długość), dobrą praktyką jest zapisanie ich w osobnym pliku i dołączenie do kodu przez *#include*.
- Nie należy umieszczać wszystkich współczynników w jednej linii programu (zalecana długość jednej linii kodu: nie więcej niż 120 znaków).
- Przykład dla stałoprzecinkowego DSP (dla wygody deklarujemy liczbę współczynników jako stałą *N*):

```
#define N 30
const short filtr_dp[] = {-4, 39, 97, 149, 133, -23, -337, -701,
-883, -595, 360, 1955, 3883, 5634, 6677, 6677, 5634, 3883, 1955,
360, -595, -883, -701, -337, -23, 133, 149, 97, 39, -4};
```

Implementacja filtru FIR

W praktyce używamy zoptymalizowanych algorytmów FIR dostarczonych przez producenta. Pokażemy własną implementację filtru FIR w języku C w celach demonstracyjnych.

- Musimy zapamiętać *N* ostatnich próbek sygnału.
- Używamy do tego bufora kołowego.
- Bufor deklarujemy globalnie (nie na stosie!).

```
short bufor_filtru[N];
```

- Potrzebujemy indeksu wskazującego miejsce zapisu w buforze (zmienna globalna):

```
unsigned short indeks_bufora = 0;
```

Implementacja filtru FIR

- Nie należy zakładać, że bufor po utworzeniu będzie wypełniony zerami - musimy zrobić to sami (inaczej będą tam „śmieci”):

```
int i;
for (i = 0; i < N; i++)
    bufor_filtru[i] = 0;
```

- Zapisujemy nową próbkę sygnału do bufora kołowego.
- Potrzebujemy drugiego indeksu (*poz*) do odczytu z bufora.
- W pętli przesuwamy się jednocześnie po buforze próbek i po tablicy współczynników filtru (mamy pary: próbka i „jej” współczynnik).
- Dla każdej iteracji pętli wykonujemy mnożenie (próbka · współczynnik) i dodajemy wynik do sumy (akumulatora).
- Indeksy przesuwamy funkcją *_circ_incr*.

Wykorzystanie instrukcji MAC

- Filtracja FIR: każdą próbkę *x* mnożymy przez „jej” współczynnik *b*, a wynik mnożenia dodajemy do sumy *y* (akumulacja):

  y := y + b * x

- Taki sposób obliczania wymaga wykonania najpierw mnożenia (MPY), a potem osobno dodawania (ADD) – dwie instrukcje.
- Możemy wykorzystać specjalną instrukcję MAC – *multiply and accumulate*, wykonującą obie operacje w jednym cyklu procesora.
- Procesor C5535 posiada instrukcje:
  - *_smaci* – wykonuje mnożenie w trybie liczb całkowitych (Q15·Q15 → Q30),
  - *_smac* – wykonuje mnożenie w trybie liczb ułamkowych (Q15·Q15 → Q31).

Własna implementacja filtru FIR w języku C

```
// x: próbka odczytana z wejścia (short, Q15)
bufor_filtru[indeks_bufora] = x;  // zapis do bufora kołowego
// indeks odczytu z bufora
unsigned short poz = indeks_bufora;  // od najnowszej próbki
long y = 0;  // y: wynik filtracji (Q30)

// pętla po współczynnikach i buforze filtru
for (i = 0; i < N; i++) {
    // y = y + x[n]*b[n]
    y = _smaci(y, bufor_filtru[poz], filtr_dp[i]);
    poz = _circ_incr(poz, -1, N); // poz = poz - 1
}

// wyjście: wynik filtracji dla bieżącej próbki (short, Q15)
// zamieniamy Q30 na Q31, zaokrąglamy, obcinamy do 16 bitów
wyjscie = (short)(_sround(y<<1) >> 16);
// przesunięcie indeksu zapisu bufora kołowego (ib = ib + 1)
indeks_bufora = _circ_incr(indeks_bufora, 1, N);
```

Funkcja fir z DSPLIB

- Biblioteka DSPLIB posiada zoptymalizowaną funkcję filtracji FIR w asemblerze:

```
ushort oflag = fir (DATA *x, DATA *h, DATA *r, DATA *dbuffer, ushort nx,
ushort nh)
```

- *x*: wskaźnik do zmiennej, z której odczytywane są wejściowe próbki sygnału.
- *h*: wskaźnik do tablicy współczynników filtru.
- *r*: wskaźnik do zmiennej, do której zapisane zostaną wyniki filtracji.
- *dbuffer*: bufor kołowy dla filtru; należy go utworzyć (jako globalną tablicę) i wyzerować przed użyciem; nie należy go później modyfikować.

```
dbuffer[nh+2]      Pointer to delay buffer of length nh = nh + 2
```

- *nx*: liczba próbek do przetworzenia (długość *x* oraz *r*).
- *nh*: liczba współczynników filtru (długość *h*).
- Zwracana wartość *oflag* jest równa 1 przy przepełnieniu zakresu w obliczeniach.

Funkcja fir z DSPLIB

- Typ *DATA* jest synonimem *short* (16 bitów ze znakiem).
- W globalnej części programu zamieszczamy współczynniki i deklarujemy bufor.

```
#define N 30                        // liczba współczynników
DATA bufor_fir[N+2];                // bufor kołowy
const short filtr_dp[] = {-4, 39, 97, 149, 133, -23, -337, -701,
-883, -595, 360, 1955, 3883, 5634, 6677, 6677, 5634, 3883, 1955,
360, -595, -883, -701, -337, -23, 133, 149, 97, 39, -4};
```

- Bufor filtru należy wyzerować przed pierwszym użyciem. Również wtedy, gdy przełączamy się na filtr FIR z innego trybu pracy. Nie modyfikujemy sami bufora w trakcie gdy filtr przetwarza sygnał.

```
int i;
for (i = 0; i < N + 2; i++)
    bufor_fir[i] = 0;
// (można też użyć instrukcji memset)
```

Funkcja fir z DSPLIB

```
ushort oflag = fir (DATA *x, DATA *h, DATA *r, DATA *dbuffer, ushort nx,
ushort nh)
```

Filtracja jednej próbki sygnału zapisanej w zmiennej *wejscie* (typ *short*)
i zapis wyniku do zmiennej *wyjscie* (*short*):

- należy pobrać wskaźnik ze zmiennej *wejscie* za pomocą znaku &,
- wskaźnik do tablicy współczynników należy rzutować aby pozbyć się *const*.

```
fir(&wejscie, (DATA*)filtr_dp, &wyjscie, bufor_fir, 1, N);
```

Filtracja bloku 512 próbek zapisanych w tablicy *tab_wejscie* (typ *short*)
i zapis wyniku do tablicy *tab_wyjscie* (*short*, musi mieć też 512 elementów):

- nie stawiamy znaku &, nazwa tablicy jest już wskaźnikiem,
- wskaźnik do tablicy współczynników należy rzutować aby pozbyć się *const*.

```
fir(tab_wejscie, (DATA*)filtr_dp, tab_wyjscie, bufor_fir, 512, N);
```

Symetria filtrów FIR

- Najczęściej projektujemy filtry FIR o liniowej fazie – obliczone współczynniki są symetryczne:
  - filtry o nieparzystej długości (typ I): dwie symetryczne „połówki” plus jeden współczynnik „bez pary”, dowolny typ charakterystyki;
  - filtry o parzystej długości (typ II): dwie symetryczne „połówki”, tylko filtry dolnoprzepustowe i pasmowo-zaporowe.
- Przy symetrii współczynników: dwie różne próbki są mnożone przez ten sam współczynnik filtru.
- Można wykorzystać symetrię do zmniejszenia liczby operacji mnożenia: dodajemy dwie próbki do siebie i wykonujemy mnożenie współczynnika przez obliczoną sumę.
- W tablicy wystarczy zapisać tylko nie powtarzające się współczynniki.

```
const short filtr_dp[] = {-4, 39, 97, 149, 133, -23, -337, -701,
-883, -595, 360, 1955, 3883, 5634, 6677, 6677, 5634, 3883, 1955,
360, -595, -883, -701, -337, -23, 133, 149, 97, 39, -4};
```

Implementacja symetrycznego filtru FIR

- Tworzymy dwa indeksy do bufora kołowego z próbkami: jeden wskazuje na najnowszą próbkę, drugi – na najstarszą (jedna pozycja dalej).
- Dodajemy dwie próbki wskazane przez te indeksy do siebie.
- Wykonujemy MAC: współczynnik i suma dwóch próbek.
- Przesuwamy oba indeksy: pierwszy do tyłu, drugi do przodu. Powtarzamy.
- Jeżeli długość filtru jest nieparzysta: musimy osobno potraktować środkową próbkę z bufora i przemnożyć ją przez współczynnik, który nie ma pary.

```
// potrzebujemy dwa wskaźniki odczytu
short poz1 = indeks_bufora;              // najnowsza próbka
short poz2 = _circ_incr(poz1, 1, N);     // najstarsza próbka
short sumapr;
long wynik = 0;

for (i = 0; i < (N>>1); i++) {
    sumapr = bufor[poz1] + bufor[poz2];
    wynik = _smaci(wynik, sumapr, filtr_dp[i]);
    poz1 = _circ_incr(poz1, -1, N);  // jeden wskaźnik wstecz
    poz2 = _circ_incr(poz2,  1, N);  // drugi do przodu
}
// środkowy współczynnik (bez pary)
if (N & 1)                                // nieparzyste N
     wynik = _smaci(wynik, bufor[poz1], filtr_dp[i]);

wyjscie = (short)(_sround(wynik<<1) >> 16);
indeks_bufora = _circ_incr(indeks_bufora, 1, N);
```

Funkcja firs w DSPLIB

- Procesor C5535 posiada specjalną instrukcję *FIRSADD*, która ułatwia obliczanie symetrycznych filtrów FIR o parzystej długości.
- Instrukcja ta wykonuje równolegle (w jednym cyklu procesora) dwie operacje:
  - MAC dla bieżącej pary próbek sygnału,
  - przygotowanie kolejnego cyklu – sumowanie następnych dwóch wartości próbek.
- Znacznie przyspiesza to operację filtracji.
- Instrukcja języka C: *_firsadd*.
- Procedura *firs* z biblioteki *DSPLIB* wykorzystuje instrukcję *FIRSADD* (wystarczy zapisać połowę – *nh2* współczynników filtru w tablicy *h*).

firs	*Symmetric FIR Filter*
Function	ushort oflag = firs (DATA *x, DATA *h, DATA *r, DATA *dbuffer, ushort nx, ushort nh2)

Przetwarzanie blokowe w FIR

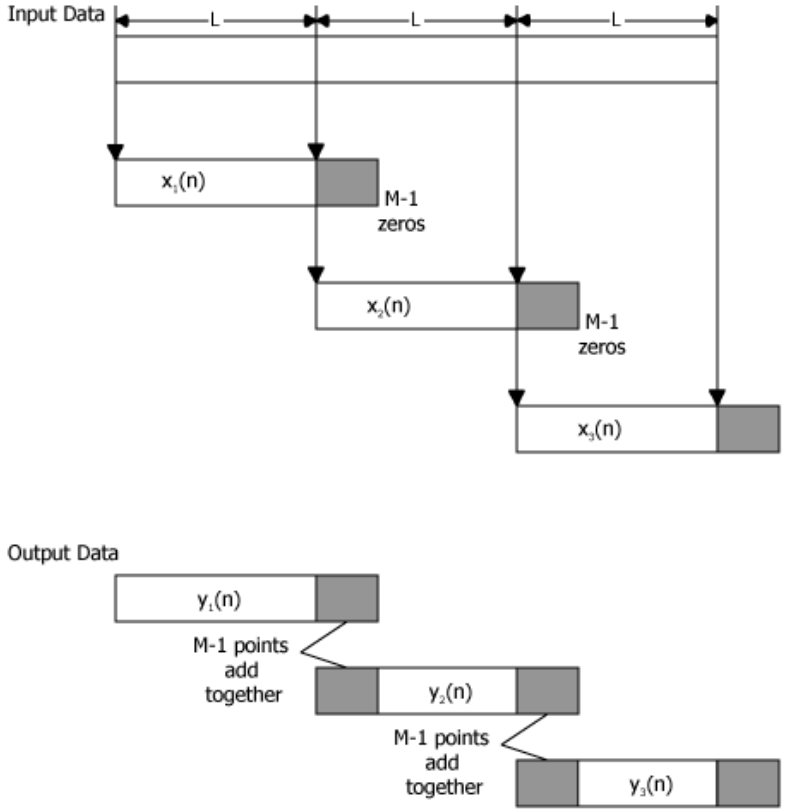
- Procesor zwykle wykonuje filtrację ciągłego sygnału z wejścia.
- Czasami korzystniej jest filtrować blok próbek zamiast każdej próbki osobno.
- Można to zrobić za pomocą funkcji *fir* z DSPLIB (w dziedzinie czasu).
- Filtracja FIR bloku próbek za pomocą zbioru współczynników filtru jest splotem liniowym współczynników z blokiem próbek.
- Dla dłuższych filtrów korzystniej jest wykonać filtrację na blokach próbek, za pomocą splotu w dziedzinie częstotliwości.
- Do łączenia wyników splotów wykorzystujemy algorytm *overlap-add* (OLA) lub *overlap-save* (OLS).

Overlapp-add (OLA)

- Mamy filtr FIR o długości *M*.
- Dzielimy sygnał na bloki o długości *L*.
- Wynik splotu liniowego sygnałów ma długość (*L*+*M*-1). Ze względu na wymagania procedury FFT, liczba ta powinna być potęgą dwójki.
- Załóżmy dla przykładu: $L = 256$, $M = 257$, $(L+M-1) = 512$.
- Do $M = 257$ współczynników filtru dopisujemy 255 zer, mamy 512 wartości.
- Obliczamy FFT dla tych 512 wartości, zapisujemy w pamięci.
- Tworzymy bufor OLA o długości 512 i wypełniamy go zerami.

Overlapp-add (OLA)

- Bierzemy kolejny blok *L* = 256 próbek sygnału.
- Dopisujemy 256 zer i liczymy FFT.
- Mnożymy przez siebie transformaty sygnału i współczynników filtru.
- Liczymy odwrotne FFT (IFFT) – mamy 512 wartości.
- Dodajemy te wartości do bufora OLA.
- Bierzemy pierwsze 256 wartości – wynik filtracji.
- Pozostałe 256 wartości przesuwamy na początek bufora OLA i uzupełniamy zerami (można użyć bufora kołowego aby nie przesuwać wartości w buforze).
- Powtarzamy to dla kolejnych bloków próbek.



Filtry FIR na procesorze C5535

Dlaczego wykonywanie filtracji sygnału za pomocą filtru o skończonej odpowiedzi impulsowej (FIR) na procesorze sygnałowym, takim jak C5535, jest wydajne?

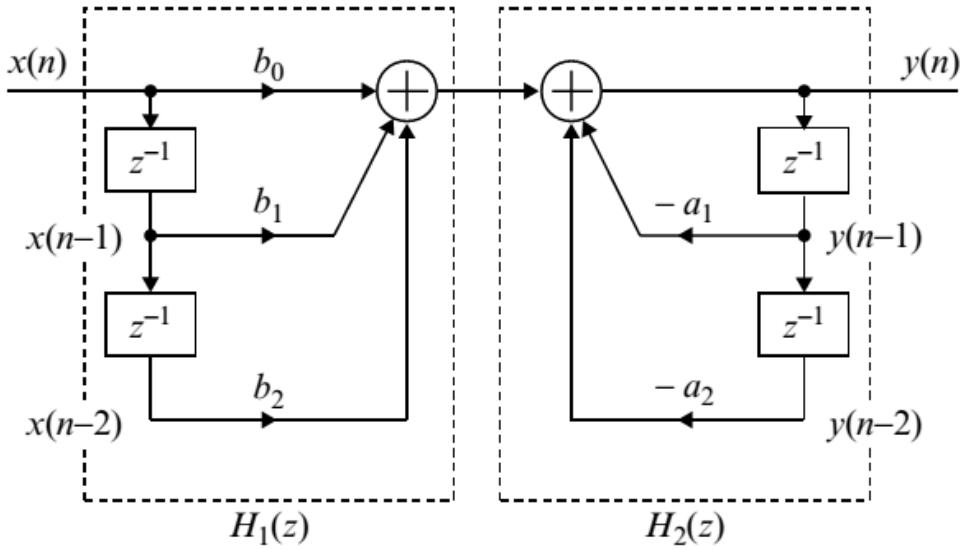
- Instrukcja MAC przyspiesza obliczanie filtru (jednoczesne mnożenie i dodawanie).
- Procesor posiada architekturę *dual MAC* – dwie operacje MAC mogą być wykonane w jednym cyklu procesora (o ile kompilator C na to pozwoli). Dalsze przyspieszenie.
- Dla filtrów o parzystej długości, np. dolnoprzepustowych, można wykorzystać instrukcję *FIRSADD*, która znacznie skraca czas obliczeń.
- Procesor obsługuje adresowanie kołowe, które upraszcza przechodzenie po buforze kołowym (instrukcja *_circ_incr*).
- Pętle mogą być wykonywane bez narzutu (*zero overhead loops*).
- Do filtracji blokowej można wykorzystać szybko obliczane FFT.

Filtry IIR

Drugi typ filtru – o nieskończonej odpowiedzi impulsowej, IIR (*infinite impulse response*).

- Filtry rekursywne – filtr rzędu *N* wymaga *N*+1 ostatnich próbek sygnału (*x*) oraz *N* poprzednich wyników filtracji (*y*).
- Filtr drugiego rzędu (*biquad*, *N*=2):

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2)$$



Kwantyzacja współczynników filtru IIR

- Współczynniki obliczamy za pomocą programu komputerowego.
- Pojawia się problem: jeden współczynnik (a_1) ma wartość spoza zakresu [-1, 1). Jak go zapisać w implementacji stałoprzecinkowej? W Q15 nie da się.

```
0.0039, 0.0078, 0.0039, -1.8153, 0.8310
```

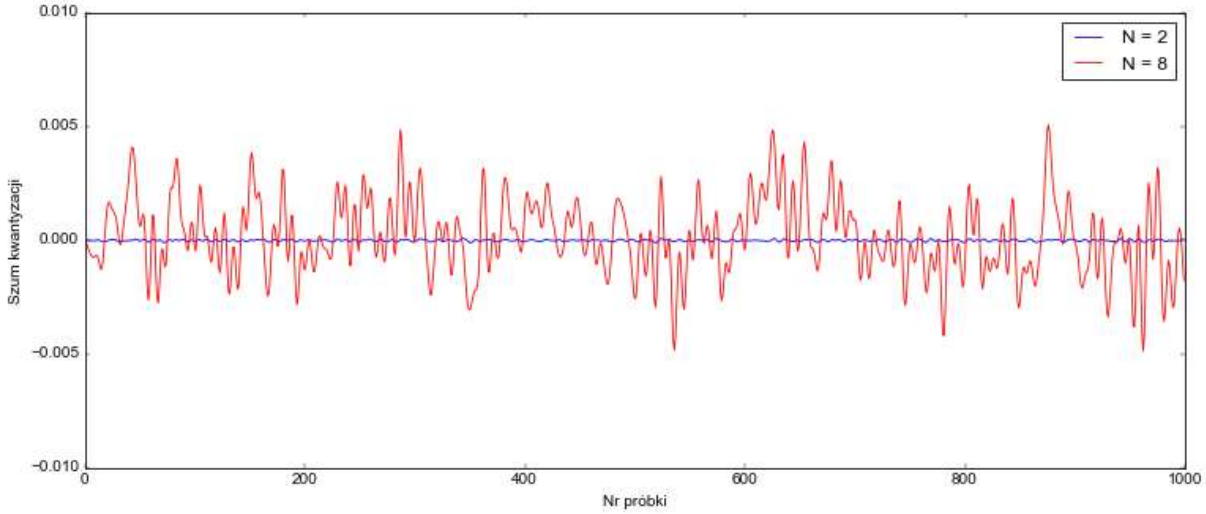
- Musimy poświęcić jeden bit precyzji i zapisać współczynniki w formacie Q1.14, mnożąc liczby zmiennoprzecinkowe przez 16384.

```
const short wsp_iir[] = {64, 128, 64, -29743, 13615};
//                        b0,  b1, b2,     a1,    a2
```

- Powinniśmy sprawdzić wzmocnienie i stabilność filtru.

Kwantyzacja współczynników filtru IIR

- Tak jak w filtrach FIR, tak i tutaj występuje szum kwantyzacji.
- Ale w filtrach IIR wyniki obarczone szumem kwantyzacji są brane do obliczeń dla kolejnych próbek.
- Efekt szumu kwantyzacji kumuluje się. Im większy rząd filtru, tym większy szum kwantyzacji.
- Możliwe problemy: przekroczenie zakresu liczb, utrata stabilności filtru.



Struktura kaskadowa

- Aby zmniejszyć problem szumu kwantyzacji w filtrach IIR, stosuje się strukturę kaskadową: podział filtru na sekcje drugiego rzędu (SOS, *biquad*).
- Na przykład filtr 20. rzędu można zrealizować jako kaskadę 10 filtrów 2. rzędu.
- W implementacji zmiennoprzecinkowej sposób podziału na sekcje jest bez większego znaczenia – mnożenie jest przemienne.
- W implementacji stałoprzecinkowej to już ma znaczenie. Jedna para zer/biegunów może spowodować przepełnienie zakresu, inna nie.
- Zbudowanie optymalnej struktury kaskadowej do implementacji na stałoprzecinkowym DSP nie jest prostym zadaniem i wykracza poza ramy tego wykładu.

Co może pójść nie tak?

Mogą wystąpić dwa niepożądane efekty.

- Przepełnienie zakresu (*overflow*)
  - objawia się wystąpieniem na wyjściu szerokopasmowego szumu zamiast spodziewanego wyniku filtracji,
  - jest spowodowane zbyt dużym wzmocnieniem filtru w co najmniej jednej sekcji,
  - rozwiązanie: zmniejszyć wzmocnienie filtru / sekcji, skalując współczynniki *b*.
- Niedopełnienie (*underflow*)
  - objawia się występowaniem samych zer na wyjściu,
  - jest spowodowane zbyt małym wzmocnieniem w co najmniej jednej sekcji filtru,
  - wymaga korekcji wzmocnienia sekcji – właściwego rozłożenia wzmocnienia na poszczególne sekcje.

Projekt filtru

- Obliczamy współczynniki filtru przy pomocy programu komputerowego, dla założonych parametrów projektowych.
- Dla filtrów rzędu większego niż 6 stosujemy strukturę kaskadową (SOS).
- Całościowe wzmocnienie filtru rozkładamy na sekcje (możliwie równomiernie).
- Jeżeli mamy stałoprzecinkowy DSP, zamieniamy współczynniki na Q1.14.
- Sprawdzamy wzmocnienia każdej sekcji i całego filtru. Nie powinny one przekraczać 1 i powinny być zbliżone do siebie. W razie potrzeby korygujemy wzmocnienia. (UWAGA: wzmocnienie korygujemy tylko współczynnikami *b*!)
- Sprawdzamy stabilność filtru (po kwantyzacji).
- Zapisujemy współczynniki w kodzie programu.
- Implementujemy procedurę filtracji lub używamy gotowej procedury.

Implementacja filtru IIR w języku C

- Zapisujemy współczynniki w stałej tablicy *const*.
- Tworzymy bufor roboczy: 4 wartości na jedną sekcję 2. rzędu.
- Bufor trzeba bezwzględnie wypełnić zerami przed pierwszym użyciem. Inaczej będziemy filtrować „śmieci”.
- W podanym przykładzie stosujemy bufor liniowy, można użyć bufora kołowego.
- W obliczeniach filtru używamy instrukcji _*smac* oraz _*smas* (odejmowanie). Można też zapisać współczynniki *a* z przeciwnym znakiem i użyć pętli z _*smac*.

```
const short wsp_iir[] = {64, 128, 64, -29743, 13615};
short bufor[4];
int i;
for (i = 0; i < 4; i++)
    bufor[i] = 0;
```

Implementacja filtru IIR w języku C

```
// filtracja
long suma = 0;
suma = _smac(suma, wsp_iir[0], wejscie);   //   b0 * x(n)
suma = _smac(suma, wsp_iir[1], bufor[0]);  // + b1 * x(n-1)
suma = _smac(suma, wsp_iir[2], bufor[1]);  // + b2 * x(n-2)
suma = _smas(suma, wsp_iir[3], bufor[2]);  // - a1 * y(n-1)
suma = _smas(suma, wsp_iir[4], bufor[3]);  // - a2 * y(n-2)

short wyjscie = (short)(_sround(suma) >> 16);

// przesuwanie bufora:
bufor[3] = bufor[2];      // y(n-2) - poprzednia y(n-1)
bufor[2] = wyjscie;       // y(n-1) - poprzednia y(n)
bufor[1] = bufor[0];      // x(n-2) - poprzednia x(n-1)
bufor[0] = wejscie;       // x(n-1) - poprzednia x(n)
```

Filtracja IIR w DSPLIB

- Spośród funkcji filtracji IIR dostępnych w DSPLIB, najbardziej przydatna jest funkcja *iircas51*:

iircas51	*Cascaded IIR Direct Form I (5 Coefficients per Biquad)*
Function	ushort oflag = iircas51 (DATA *x, DATA *h, DATA *r, DATA *dbuffer, ushort nbiq, ushort nx)

- Implementacja kaskadowa (*cas*) formy bezpośredniej I (1), 5 współczynników na sekcję.
- Programy komputerowe zwykle podają 6 współczynników na sekcję. Pomijamy czwarty współczynnik (a_0), jest on zawsze 1.
- Forma druga (*direct form II*, funkcja *iircas5*) jest niezalecana dla procesorów stałoprzecinkowych – jest ona bardziej podatna na przepełnienie zakresu.

Filtracja IIR w DSPLIB

```
ushort oflag = iircas51 (DATA *x, DATA *h, DATA *r, DATA *dbuffer, ushort nbiq,
ushort nx)
```

Argumenty funkcji *iircas51*:

- *x* – wskaźnik do próbki wejściowej (*&wejscie*) lub do tablicy próbek do przetworzenia (*tab_wejscie*),
- *h* – wskaźnik do współczynników filtru, sekcja po sekcji, w kolejności: *b0, b1, b2, a1, a2*; (bez *a0*!),
- *r* – wskaźnik do próbki wyjściowej (*&wyjscie*) lub do tablicy próbek wyjściowych (*tab_wyjscie*),
- *dbuffer* – bufor roboczy o długości (4 * liczba sekcji + 1), który musimy sami utworzyć i wypełnić zerami przed użyciem,
- *nbiq* – liczba sekcji drugiego rzędu (rząd filtru / 2),
- *nx* – liczba próbek do przetworzenia (1 lub długość tablicy *x* oraz *r*).

Filtracja IIR w DSPLIB

Przykład filtru IIR 8. rzędu – 4 sekcje. Deklaracja współczynników i bufora roboczego (UWAGA: pomijamy współczynnik a_0 – „jedynkę”):

```
const short wsp_iir[] = {
     62,    124,     62,  -28801,  12665,
     63,    126,     63,  -29307,  13176,
     65,    131,     65,  -30291,  14168,
     68,    137,     68,  -31681,  15570};

short bufor[17]; // pamiętać o wyzerowaniu!
```

Przykłady wywołania funkcji:

```
iircas51(&wejscie, (DATA*)wsp_iir, &wyjscie, bufor, 4, 1);
```

```
iircas51(tab_wejscie, (DATA*)wsp_iir, tab_wyjscie, bufor, 4, 512);
```

Filtry na procesorze sygnałowym – FIR czy IIR?

Filtry o skończonej odpowiedzi impulsowej (FIR) w porównaniu do filtrów o nieskończonej odpowiedzi impulsowej (IIR) na procesorach sygnałowych:

- filtry FIR mogą mieć liniową fazę, przez co opóźnienie filtru jest niezależne od częstotliwości; w filtrach IIR tak nie jest – zniekształcają fazę;
- w filtrach FIR nie ma problemu utraty stabilności; filtry IIR mogą utracić stabilność na skutek błędów kwantyzacji;
- szum kwantyzacji ma w filtrach FIR znacznie mniejsze znaczenie; w filtrach IIR występuje efekt kumulacji błędów kwantyzacji, przez co dynamika sygnału po filtracji jest mniejsza (szczególnie na procesorach stałoprzecinkowych);
- filtry FIR są prostsze do implementacji na stałoprzecinkowych procesorach; w filtrach IIR występują problemy z przepełnieniem i niedopełnieniem zakresu.

Filtry na procesorze sygnałowym – FIR czy IIR?

Filtry o skończonej odpowiedzi impulsowej (FIR) w porównaniu do filtrów o nieskończonej odpowiedzi impulsowej (IIR) na procesorach sygnałowych:

- filtry FIR wymagają dużej liczby współczynników aby osiągnąć założony efekt filtracji; filtry IIR dają ten sam efekt używając mniejszej liczby cykli procesora;
- filtry FIR zużywają więcej pamięci do przechowywania próbek niż filtry IIR;
- opóźnienie wprowadzane przez filtry FIR są znacznie większe (ok. połowa długości filtru) niż w przypadku filtrów IIR.

Ogólnie: filtry IIR są bardziej wydajne obliczeniowo, ale cechy procesora sygnałowego przyspieszające obliczenia dla filtrów FIR zmniejszają znaczenie tych różnic.

Z tych powodów filtry FIR są zwykle pierwszym wyborem na współczesnych procesorach sygnałowych. Dla „długich” filtrów warto rozpatrzyć przetwarzanie w dziedzinie częstotliwości (OLA).