FILTRY CYFROWE

na procesorach sygnałowych

Opracowanie: Grzegorz Szwoch

Politechnika Gdańska, Katedra Systemów Multimedialnych

Wprowadzenie

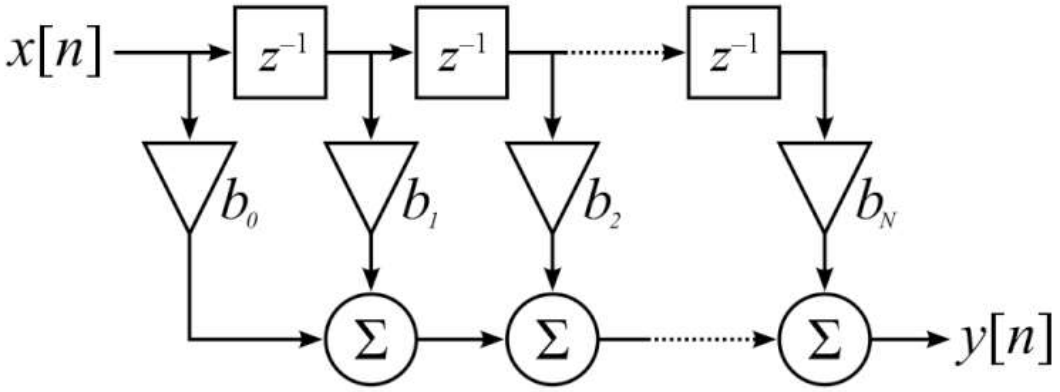
- Filtry cyfrowe należą do podstawowych algorytmów implementowanych na procesorach sygnałowych.
- Stosowane są dwa typy filtrów:
  - o skończonej odpowiedzi impulsowej – FIR („splotowe”),
  - o nieskończonej odp. impulsowej – IIR (rekursywne).
- Oba typy mają praktyczne zastosowania.
- Podstawy teoretyczne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

Filtry FIR

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

- przemnożenie *N* ostatnich próbek sygnału przez współczynniki filtru,
- sumowanie wyników mnożenia – wynik jest wyjściem filtru.

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



Projektowanie filtru FIR

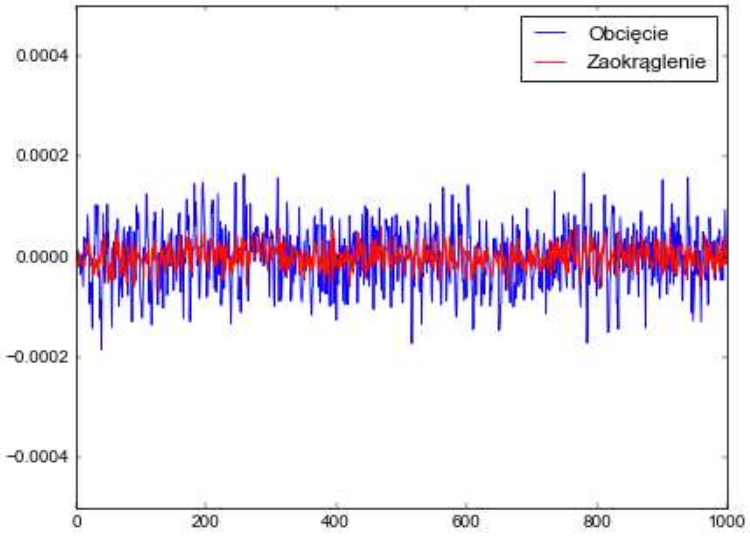
- Zakładamy jak ma wyglądać charakterystyka filtru:
  typ (np. dolnoprzepustowy), częstotliwości graniczne
  (np. 4 kHz), minimalne tłumienie (np. 40 dB),
  szerokość pasma przejściowego (np. 200 Hz).
- Obliczamy współczynniki filtru (zmiennoprzecinkowe)
  za pomocą oprogramowania (np. Matlab, Python/SciPy),
  metody: np. okienkowa lub Parksa-McClellana.
- Sprawdzamy wzmocnienie filtru – powinno być ok. 1
  w paśmie przepustowym. Jeżeli nie jest – normalizujemy.
  Dla filtrów dolnoprzepustowych: podzielenie przez sumę
  współczynników. Dla innych typów: patrz wykład z PDiO.

Współczynniki dla stałoprzecinkowego DSP

- Współczynniki filtru są zapisane jako liczby zmiennoprzecinkowe.
- Jeżeli używamy stałoprzecinkowego DSP (jak w projekcie), musimy przekształcić je do formatu Q15.
- Mnożymy współczynniki przez 32768, po czym zaokrąglamy je do liczby całkowitej (np. *round* w Matlabie).
- Powinniśmy sprawdzić wzmocnienie, czy nie przekracza zakresu. Np. dla filtru DP, suma współczynników nie powinna przekraczać 32767. Jeżeli przekracza, możemy mnożyć współczynniki przez nieco mniejszą liczbę, np. 32760.

Kwantyzacja współczynników

- Przekształcając współczynniki do liczb Q15 przeprowadzamy kwantyzację – zamieniamy współczynniki na najbliższą liczbę mającą reprezentację w formacie Q15.
- Różnica wyniku działania oryginalnego i stałoprzecinkowego filtru ma charakter szumu kwantyzacji.
- Uzyskany filtr jest inny, niż zadany podczas projektowania.
- W zmiennoprzecinkowych DSP efekt kwantyzacji wciąż występuje, ale jest on znacznie mniejszy.



Zapis współczynników w kodzie C

- Współczynniki zapisujemy w postaci stałej tablicy (*const*).
- Tablicę umieszczamy globalnie (poza funkcjami).
- Jeżeli mamy wiele filtrów, dobrą praktyką jest zapisanie ich w osobnym pliku i dołączenie do kodu (*#include*).
- Przykład dla stałoprzecinkowego DSP (dla wygody, deklarujemy liczbę współczynników jako stałą *N*):

```
#define N 30
const int 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. Napiszemy jednak własną implementację filtru FIR w języku C, do celów edukacyjnych.

- Musimy zapamiętać *N* ostatnich próbek sygnału.
- Używamy do tego bufora kołowego.
- Bufor deklarujemy globalnie.

```
int bufor_filtru[N];
```

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

```
int indeks_bufora = 0;
```

Implementacja filtru FIR

- Kompilator stosowany w projekcie ZPS nie zeruje bufora - 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.
- Musimy przejść po całym buforze, wykonać mnożenia i zsumować wyniki.
- Potrzebujemy drugiego indeksu (*poz*) do odczytu z bufora.
- Indeksy przesuwamy funkcją *_circ_incr*.
- Mnożenie i dodanie do sumy wykonujemy funkcją *_smac*.

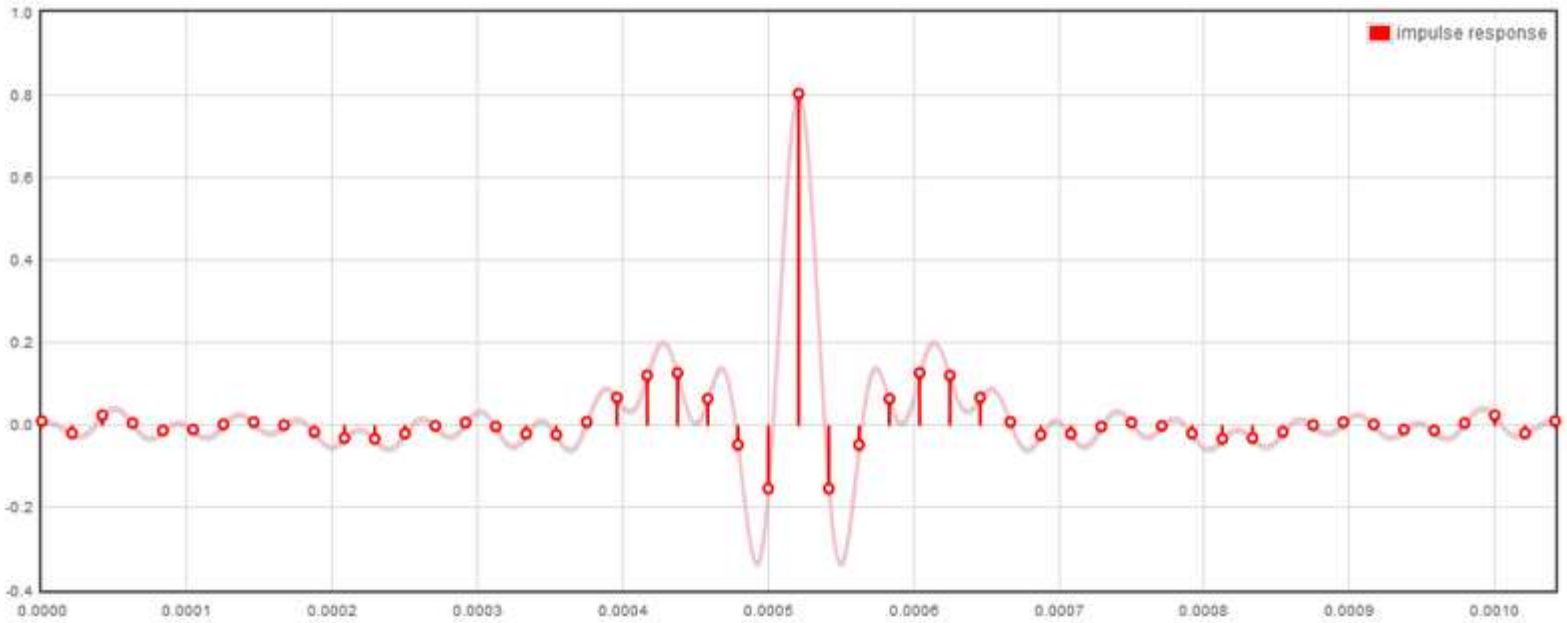
Implementacja filtru FIR

```
// x: próbka odczytana z wejścia (int)
bufor_filtru[indeks_bufora] = x;
int poz = indeks_bufora; // indeks odczytu z bufora
long y = 0;  // y: wynik filtracji

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

// wyjście: próbka wysyłana na wyjście (int)
wyjscie = (int)(_sround(y) >> 16);
// aktualizacja indeksu zapisu bufora kołowego (ib = ib + 1)
indeks_bufora = _circ_incr(indeks_bufora, 1, N);
```

Symetria filtrów FIR

- Najczęściej projektujemy filtry FIR o liniowej fazie – obliczone współczynniki są symetryczne:
  - nieparzyste filtry (typ I) – dwie symetryczne „połówki” plus jeden współczynnik „bez pary”,
  - parzyste filtry (typ II, tylko DP i pasmowo-zaporowe): dwie symetryczne „połówki”.
- Można wykorzystać symetrię do zmniejszenia liczby mnożeń: dodajemy dwie próbki do siebie i wykonujemy mnożenie współczynnika przez obliczoną sumę.
- W tablicy wystarczy zapisać tylko nie powtarzające się współczynniki.

Symetria filtrów FIR




```
const int filtr_dp[] = {15, 58, 109, 135, 68, -151, -493, -796,
-788, -201, 1076, 2884, 4798, 6263, 6812, 6263, 4798, 2884,
1076, -201, -788, -796, -493, -151, 68, 135, 109, 58, 15};
```

```
const int filtr_dp[] = {15, 58, 109, 135, 68, -151, -493, -796,
-788, -201, 1076, 2884, 4798, 6263, 6812};
```

Problem przepełnienia zakresu

- Przy dodawaniu próbek może wystąpić przekroczenie zakresu liczb Q15:

```
int sumapr = _sadd(bufor [poz1], bufor [poz2]);
```

- Aby temu zapobiec, możemy podzielić wartości próbek przez 2 (>>1) przed obliczeniem sumy:

```
int sumapr = _sadd(bufor[poz1]>>1, bufor[poz2]>>1);
```

- Musimy pamiętać o skompensowaniu końcowego wyniku.
- Tracimy w ten sposób 1 bit precyzji zapisu liczb (zyskując na mniejszej liczbie operacji mnożenia – coś za coś).

Implementacja symetrycznego filtru FIR

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

for (i = 0; i < (N>>1); i++) {
     sumapr = _sadd(bufor[poz1]>>1, bufor[poz2]>>1);
     wynik = _smac(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 = _smac(wynik, bufor[poz1]>>1, filtr_dp[i]);

// tutaj kompensujemy dzielenie przez 2:
wyjscie = (int)(_sround(wynik<<1) >> 16);
indeks_bufora = _circ_incr(indeks_bufora, 1, N);
```

Filtry FIR w DSPLIB

- DSPLIB zawiera zoptymalizowane procedury filtracji dla procesorów DSP firmy Texas Instruments. Dla stałoprzecinkowych DSP są one napisane w Asemblerze.
- Najważniejsze funkcje filtracji FIR (DSPLIB dla C55x):
  - *fir* – standardowa implementacja,
  - *fir2* – zoptymalizowana dla szybszego trybu *dual MAC*, wymaga spełnienia warunków co do alokacji buforów,
  - *firs* – implementacja filtrów symetrycznych, niestety, tylko parzystych (typ II), więc filtry DP i PZ.
- Osobno mamy funkcje dla specyficznych typów filtrów FIR (Hilberta, decymacyjne, interpolacyjne i drabinkowe).

Funkcja fir z DSPLIB

- Dokumentacja podaje nagłówek funkcji FIR:

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

- Typ *DATA* jest synonimem *int* (16-bit).
- Gwiazdka (*) oznacza, że potrzebny jest wskaźnik:
  - dla tablic (buforów): nazwa jest wskaźnikiem,
  - dla zmiennych „skalarnych”: wskaźnik trzeba pobrać stawiając znak & przed nazwą zmiennej.
- Bufor kołowy dla filtru należy utworzyć (jako globalną tablicę), wyzerować i podać jako parametr *dbuffer*. Nie należy go modyfikować!

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

- Zwracana wartość *oflag* jest równa 1 przy przepełnieniu.

Funkcja fir z DSPLIB

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

- *x* – próbki do przefiltrowania. Dla pojedynczej próbki: np. `&wejscie` (wskaźnik!)
- *h* – wektor współczynników filtru. Potrzebne rzutowanie typu aby usunąć *const*: `(DATA*)filtr_dp`
- *r* – miejsce do zapisu przefiltrowanych próbek.
  Dla pojedynczej próbki: np. `&wyjście`
- *dbuffer* – bufor kołowy, który utworzyliśmy (rozmiar *nh*+2)
- *nx* – ile próbek przetwarzamy (np. 1)
- *nh* – liczba współczynników filtru.

Funkcja fir z DSPLIB

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

Deklaracja bufora kołowego (globalnie):

```
#define N 30                        // liczba współczynników
DATA bufor_fir[N+2];                // bufor kołowy
const int filtr_dp[] = {...}        // współczynniki filtru
```

Filtracja pojedynczej próbki (ignorujemy zwracaną flagę):

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

Przetwarzanie blokowe w FIR

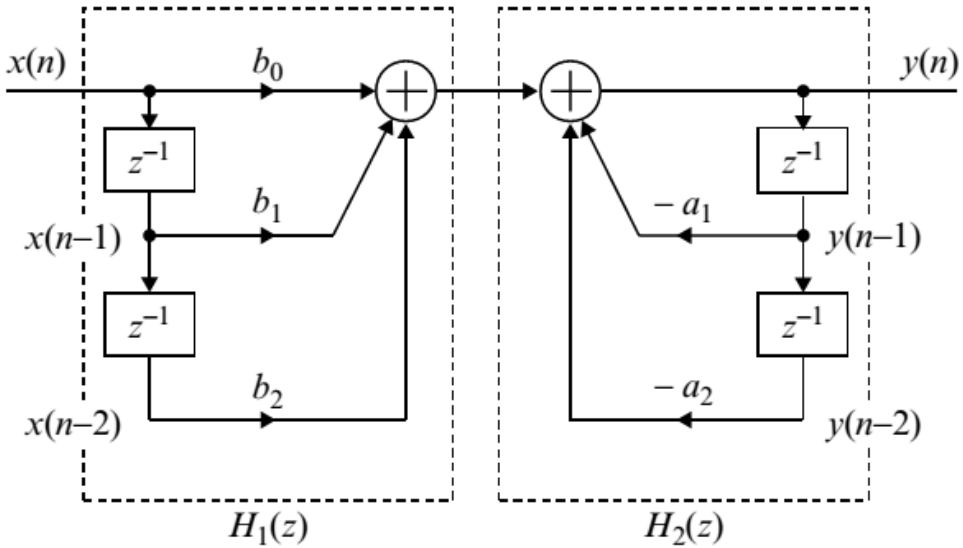
- Czasami korzystniej jest filtrować blok próbek zamiast każdej próbki osobno.
- Można to zrobić za pomocą funkcji *fir* z DSPLIB (w dziedzinie czasu).
- Dla dłuższych filtrów korzystniej jest wykonać filtrację na blokach próbek, za pomocą splotu w dziedzinie widma.
- Do łączenia wyników splotów wykorzystujemy algorytm *overlap-add* (OLA) lub *overlap-save* (OLS).
- Zasadę wykonywania szybkiego splotu przedstawiono na wykładach z PDiO dotyczących filtrów cyfrowych. Wykorzystujemy FFT, jak omówiono na poprzednim wykładzie z ZPS.

Filtry IIR

Drugi typ filtru – IIR (*infinite impulse response*)

- Filtry rekursywne – wymagają *N*+1 ostatnich próbek sygnału oraz *N* poprzednich wyników filtracji.
- Przykład dla filtru 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 oprogramowania.
- Pojawia się pierwszy problem: jeden współczynnik (a_1) ma wartość spoza zakresu [-1, 1). Jak go zapisać w implementacji stałoprzecinkowej?

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

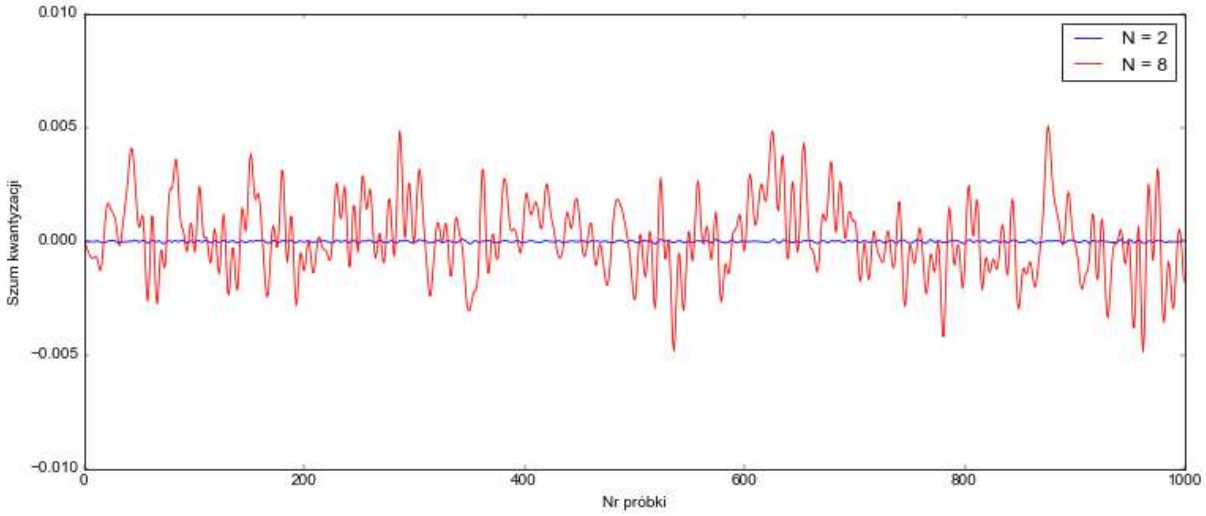
- Nie mamy wyjścia, musimy poświęcić jeden bit precyzji i zapisać współczynniki w formacie Q1.14, mnożąc liczby zmiennoprzecinkowe przez 16384.

```
const int 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, niestabilność filtru.



Struktura kaskadowa

- Aby zmniejszyć problem szumu kwantyzacji w filtrach IIR, stosuje się strukturę kaskadową: podział filtru na sekcje drugiego rzędu (SOS, *biquad*).
- W implementacji zmiennoprzecinkowej, sposób podziału na sekcje jest bez 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.

Co może pójść nie tak?

Mogą wystąpić dwa niepożądane efekty.

- Przepełnienie zakresu (*overflow*)
  - objawia się wystąpieniem na wyjściu szumu szerokopasmowego zamiast spodziewanego wyniku,
  - jest spowodowane zbyt dużym wzmocnieniem filtru w przynajmniej jednej sekcji,
  - rozwiązanie: zmniejszyć wzmocnienie filtru / sekcji.
- Niedopełnienie (*underflow*)
  - objawia się występowaniem samych zer na wyjściu,
  - jest spowodowane zbyt małym wzmocnieniem w przynajmniej jednej sekcji filtru.

Projekt filtru

- Obliczamy współczynniki filtru przy pomocy oprogramowania, w reprezentacji kaskadowej (SOS).
- Jeżeli mamy stałoprzecinkowy DSP, zamieniamy 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 współczynnikami *b*!)
- Sprawdzamy stabilność filtru (po kwantyzacji).
- Zapisujemy współczynniki w kodzie programu.

Filtracja IIR w DSPLIB

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

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

- Implementacja kaskadowa (*cas*) formy bezpośredniej I (1), dla 5 współczynników na sekcję.
- Programy zwykle podają 6 współczynników na sekcję. Pomijamy czwarty współczynnik (a_0), jest on zawsze 1.
- Forma druga (*direct form II*) 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)
```

- *x* – wskaźnik do próbki wejściowej (*&probka*)
  lub do bufora próbek do przetworzenia (*bufor_wej*)
- *h* – wskaźnik do współczynników filtru, w kolejności:
  b0, b1, b2, a1, a2; sekcja po sekcji (bez *a0*!)
- *r* – wskaźnik do próbki wyjściowej (*&wynik*)
  lub do bufora próbek wyjściowych (*bufor_wyj*)
- *dbuffer* – bufor o długości (4 * liczba sekcji + 1),
  który musimy sami utworzyć i wyzerować,
- *nbiq* – liczba sekcji drugiego rzędu,
- *nx* – liczba próbek do przetworzenia (np. 1).

Filtracja IIR w DSPLIB

- Funkcja *iircas51* zakłada format Q15. Dla liczb Q1.14, wynik po każdej sekcji jest 2 razy za mały.
- Można to skompensować na wyjściu, ale ryzykujemy niedopełnienie lub utratę precyzji obliczeń.
- Zmodyfikowany kod (zawarty w plikach do projektu ZPS) – kompensacja po każdej sekcji:

```
238         || RPTBLOCAL OuterLoopEnd-1                 ;outer loop: process a new input
239                 MOV     *AR0+ << #16, AC1           ; HI(AC1) = x(n)
240                 ||RPTBLOCAL     InnerLoopEnd-1      ;inner loop: process a bi-quad
241                         NOP_16                      ; CPU_116: Remark 5682
242                     || MPYM     *AR1+, AC1, AC0     ; AC0 = b0*x(n)
243                        MACM     *AR1+, *(AR6+T0), AC0 ; AC0 += b1*x(n-1)
244                        MACM     *AR1+, *AR6, AC0    ; AC0 += b2*x(n-2)
245                        MOV      HI(AC1), *(AR6+T1)  ; x(n) replaces x(n-2)
246                        MASM     *AR1+, *(AR4+T0), AC0 ; AC0 -= a0*y(n-1)
247                        MASM     *AR1+, *AR4, AC0, AC1 ; AC1 -= a1*y(n-2)
248                        SFTS     AC1, #1            ; AC1 *= 2 (correction for Q14)
249                        MOV      rnd(HI(AC1)), *(AR4+T1) ; y(n) replaces y(n-2)
250     InnerLoopEnd:
251                 AMOV    AR7, AR1                    ;reinitialize coeff pointer
252             || MOV      rnd(HI(AC1)), *AR2+         ;store result to output buffer
```

Filtracja IIR w DSPLIB

Deklaracja współczynników (4 sekcje) i bufora roboczego (UWAGA: pomijamy współczynnik a_0 – „jedynkę”):

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

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

Filtracja próbki *wejście* i zapisanie do zmiennej *wyjście*:

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

Podsumowanie – filtry FIR na DSP

Z punktu widzenia implementacji na procesorach sygnałowych:

- Filtry FIR wymagają znacznie większej liczby operacji mnożenia i dodawania niż filtry IIR, aby uzyskać taki sam efekt.
- Jednak cechy procesorów DSP sprawiają, że filtry FIR liczone są bardzo szybko:
  - instrukcje MAC, dual MAC, FIRLS, itp.,
  - szybkie wykonywanie pętli (bez narzutu),
  - specjalne tryby adresowania (np. kołowego), itp.
- Z tych powodów, filtry FIR są zwykle pierwszym wyborem na współczesnych procesorach DSP.
- Dla „długich” filtrów warto rozpatrzyć implementację w dziedzinie widma (OLA).

Podsumowanie – filtry IIR na DSP

Z punktu widzenia implementacji na procesorach sygnałowych:

- Filtry IIR są „trudniejsze do okiełznania” niż filtry FIR:
  - problem właściwego podziału na sekcje (przepełnienie, niedopełnienie),
  - problem szumu kwantyzacji (dokładność wyników).
- Filtry IIR powinny być rozpatrzone zamiast FIR, gdy:
  - mamy ograniczone zasoby procesora i pamięci (np. uruchamiamy wiele filtrów),
  - bardzo istotne jest zminimalizowanie opóźnień wprowadzanych przez filtry (w FIR są duże).
- Pamiętajmy też, że filtry IIR zniekształcają fazę.