4

Szybkie przekształcenie Fouriera

Wprawdzie DFT jest najbardziej bezpośrednią procedurą matematyczną do określania częstotliwościowej zawartości ciągu z dziedziny czasu, jest ona bardzo nieefektywna. Ponieważ liczba punktów w DFT wzrasta do setek lub tysięcy, liczba wymaganych do przetworzenia danych staje się bardzo szybko gigantyczna. W 1965 roku został opublikowany przez Cooleya i Tuckeya artykuł przedstawiający bardzo wydajny algorytm implementujący DFT [1]. Algorytm ten jest obecnie znany jako szybkie przekształcenie Fouriera (ang. *Fast Fourier Transform* — FFT) [1]. Przed nastaniem FFT obliczanie tysiąc punktowych DFT wymagało tak długiego czasu przetwarzania, że jej użycie ograniczało się do większych badawczych i uniwersysteckich centrów komputerowych. Dzięki Cooleyowi, Tuckeyowi oraz przemysłowi półprzewodnikowemu, 1024-punktowa DFT może być obecnie wyznaczona w kilka sekund przy użyciu komputera domowego.

O FFT napisano całe tomy i, jak żadna inna innowacja, rozwój tego algorytmu przetransformował dziedzinę cyfrowego przetwarzania sygnałów dzięki udostępnieniu mocy analizy Fourierowskiej. W niniejszym rozdziale pokażemy, dlaczego najpopularniejszy algorytm FFT (zwany algorytmem FFT *o podstawie 2*) ma przewagę w stosunku do klasycznego algorytmu DFT, przedstawimy szereg zaleceń umożliwiających ulepszenie wykorzystania przez nas FFT w praktyce i zamieścimy zbiór podprogramów FFT w różnych językach programowania. Z myślą o Czytelnikach chcących poznać jego szczegóły wewnętrzne, podsumujemy ten rozdział wyprowadzeniem algorytmu FFT o podstawie 2 i przedstawimy kilka różnych sposobów, na jakie ten algorytm FFT jest implementowany.

---

[1] W rzeczywistości, FFT ma ciekawą historię. Podczas analizowania danych rozpraszania promieni X, kilku fizyków w latach 1940. wykorzystało symetrię sinusów i cosinusów, używając matematycznej metody opartej na technice opublikowanej na początku lat 1900. Co znamienne, musiało minąć ponad 20 lat zanim algorytm FFT został (ponownie) odkryty. W pozycji [2] opowiedziano całą tę historię.

4.1. Związek pomiędzy FFT i DFT

Wprawdzie zaproponowano wiele różnych algorytmów FFT, to w niniejszym punkcie zobaczymy, dlaczego algorytm FFT o podstawie 2 jest tak popularny i dowiemy się, jaka jest relacja pomiędzy nim a klasycznym algorytmem DFT. Algorytm FFT o podstawie 2 jest bardzo efektywną procedurą wyznaczania DFT pod warunkiem, że rozmiar DFT będzie całkowitą potęgą liczby dwa. (To jest, liczba punktów w transformacji wynosi $N = 2^k$, gdzie k jest pewną liczbą naturalną.) Przyjrzyjmy się dlaczego właśnie FFT o podstawie 2 jest szczególnie lubianą techniką analizy, używaną przez praktyków przetwarzania sygnałów.

Przypomnijmy, że nasz przykład 1 DFT z punktu 3.1 ilustrował liczbę nadmiarowych operacji arytmetycznych, wymaganych dla prostej 8-punktowej DFT. (Na przykład, kończyliśmy ją obliczając iloczyn 1,0607 · 0,707 osobno cztery razy.) A więc algorytm FFT o podstawie 2 eliminuje te nadmiarowości i zmniejsza znacząco liczbę niezbędnych operacji arytmetycznych. Aby docenić wydajność FFT, rozważmy liczbę mnożeń zespolonych, niezbędnych dla naszego starego przyjaciela, wyrażenia dla N-punktowej DFT,

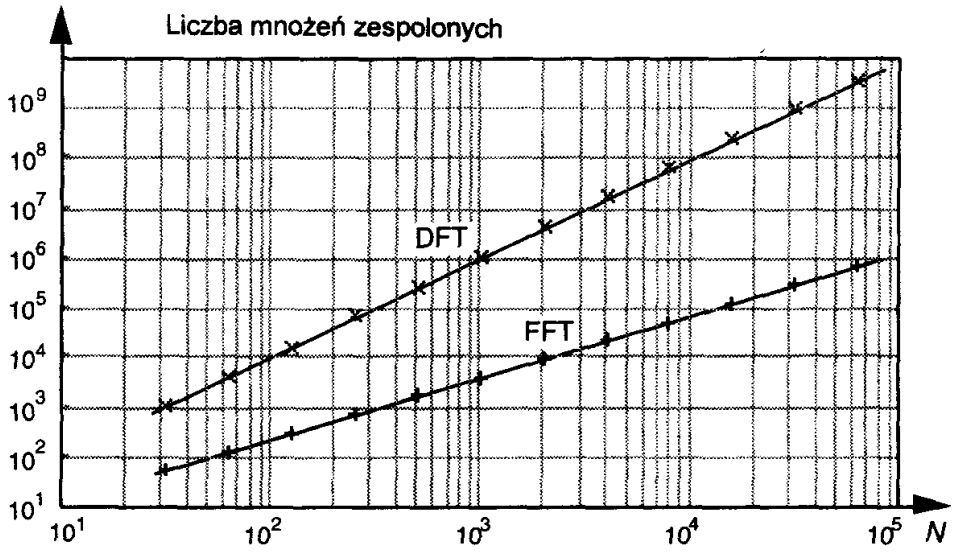
$$X(m) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi nm/N} \tag{4.1}$$

Dla 8-punktowej DFT z równania (4.1) widać, że musielibyśmy przeprowadzić N^2 lub 64 mnożenia zespolone. (Jest tak dlatego, że dla każdej z ośmiu wartości $X(m)$ musimy zsumować osiem iloczynów zespolonych, gdyż n zmienia się od 0 do 7.) Jak sprawdzimy w późniejszych punktach niniejszego rozdziału, liczba mnożeń zespolonych dla N-punktowej FFT wynosi około

$$\frac{N}{2} \cdot \log_2 N \tag{4.2}$$

(Mówimy w przybliżeniu, ponieważ niektóre mnożenia okazują się mnożeniami przez $+1$ lub -1, co sprowadza się jedynie do zmiany znaku.) A więc, ta wartość $(N/2)\log_2 N$ oznacza znaczące, w stosunku do N^2, zmniejszenie liczby mnożeń zespolonych wymaganych przez równanie (4.1), szczególnie dla dużych wartości N. Aby pokazać, jak znacząca jest ta redukcja, na rys. 4.1 porównano liczbę mnożeń zespolonych wymaganych przez operacje DFT i FFT o podstawie 2, jako funkcję liczby N punktów danych wejściowych. Jeśli, dla przykładu, $N = 512$, to DFT wymaga 200 razy więcej mnożeń zespolonych, w stosunku do wymaganych w algorytmie FFT. Jeśli $N = 8192$, to w DFT musimy obliczyć 1000 mnożeń zespolonych na *każde* mnożenie zespolone w FFT!

W tym miejscu należy jasno stwierdzić, że FFT nie jest przybliżeniem DFT. Jest ona równoważna DFT i to *jest* DFT. Ponadto, wszystkie właściwości DFT, przedstawione w poprzednim rozdziale, a więc symetria wartości wyjściowych, liniowość, wartości wyjściowe, przeciek, strata zafalowań itd., dotyczą również zachowania się FFT.



Rys. 4.1. Liczba mnożeń zespolonych w DFT oraz FFT o podstawie 2, jako funkcja *N*

4.2. Wskazówki praktyczne dotyczące algorytmów FFT

Opierając się na tym, jak bardzo użyteczne są algorytmy FFT, dalej przedstawiamy zbiór wskazówek praktycznych, lub też sugestii, dotyczących pozyskiwania próbek danych wejściowych i wykorzystywania algorytmu FFT o podstawie 2 do analizowania sygnałów lub danych rzeczywistych.

4.2.1. Próbkowanie wystarczająco szybkie i wystarczająco długie

Przy cyfryzacji sygnałów ciągłych za pomocą przetwornika A/C wiemy, przykładowo z rozdziału 2, że szybkość próbkowania musi być większa, niż podwójna szerokość pasma ciągłego sygnału wejściowego przetwornika A/C, aby ustrzec się przed aliasingiem w dziedzinie częstotliwości. W zależności od zastosowania, praktycy zazwyczaj próbkują przy szybkości 2,5 do czterech razy większej, niż szerokość pasma sygnału. Jeśli wiemy, że szerokość pasma sygnału ciągłego nie jest zbyt duża w porównaniu z maksymalną szybkością próbkowania stosowanego przetwornika A/C, łatwo można uniknąć aliasingu. Jeśli nie znamy szerokości pasma wejściowego sygnału ciągłego przetwornika A/C, jak możemy stwierdzić, czy występują problemy aliasingu? Cóż, nie powinniśmy ufać tym wszystkim wynikom FFT, które mają znaczące składowe widmowe przy częstotliwościach bliskich połowie szybkości próbkowania. W przypadku idealnym, chcielibyśmy mieć do czynienia z sygnałami, których wartości widmowe maleją ze wzrostem częstotliwości. Powinniśmy podejrzewać powstanie aliasingu, jeśli istnieją jakiekolwiek składowe widmowe, których częstotliwości okazują się zależnymi od szybkości próbkowania. Jeśli podejrzewamy, że pojawia się aliasing lub że sygnał ciągły zawiera szum szerokopasmowy, będziemy musieli użyć analogowego filtru

dolnoprzepustowego przed przetworzeniem A/C. Częstotliwość graniczna filtru dolnoprzepustowego musi, oczywiście, być większa, niż zakres częstotliwości, jaki nas interesuje, ale mniejsza od połowy szybkości próbkowania.

Wprawdzie wiemy, że N-punktowa transformata FFT o podstawie 2 wymaga $N = 2^k$ próbek wejściowych, ale należy zadać pytanie ile próbek musimy zebrać zanim wyznaczymy FFT? Odpowiedź jest taka, że przedział czasu, w którym zbieramy te dane, musi być wystarczająco długi, aby spełnić nasze wymagania dotyczące rozdzielczości częstotliwościowej FFT przy zadanej szybkości próbkowania f_s. Przedział czasu, w którym zbieramy dane, jest odwrotnością wymaganej rozdzielczości FFT i im dłużej próbkujemy z ustaloną szybkością próbkowania f_s, tym lepsza będzie rozdzielność częstotliwościowa; to jest, całkowity przedział czasu, w którym zbieramy dane, wynosi N/f_s sekund, rozdzielczość częstotliwościowa N-punktowej FFT od prążka do prążka zaś, to f_s/N Hz. A więc, na przykład, jeśli wymagana jest rozdzielczość widmowa 5 Hz, to

$$N = \frac{f_s}{\text{wymagana rozdzielczość}} = \frac{f_s}{5} = 0{,}2f_s \tag{4.3}$$

W tym przypadku, jeśli f_s wynosiłaby, powiedzmy, 10 kHz, to wówczas N musiałoby być równe co najmniej 2000, i wówczas wybralibyśmy N równe 2048, ponieważ liczba ta jest potęgą 2.

4.2.2. Przetwarzanie wstępne danych czasowych przed wyznaczeniem FFT

Jeśli nie mamy kontroli nad długością ciągu danych w dziedzinie czasu i ta długość ciągu nie jest całkowitą potęgą dwójki, używając algorytmu FFT o podstawie 2, mamy dwie opcje. Możemy odrzucić tyle próbek danych, aby długość pozostałego ciągu wejściowego FFT była całkowitą potęgą dwójki. Sposób ten nie jest jednak polecany, ponieważ odrzucanie próbek danych degraduje uzyskiwaną rozdzielczość w dziedzinie częstotliwości. (Im większa jest liczba N, tym lepsza jest osiągana rozdzielczość częstotliwościowa, nieprawdaż?) Lepszym podejściem jest dodanie wymaganej liczby próbek o wartościach zerowych do części końcowej ciągu danych czasowych, aby dopasować liczbę jego punktów do kolejnego rozmiaru FFT o podstawie 2. Na przykład, jeśli mamy do przetransformowania 1000 próbek czasowych, wówczas zamiast analizować jedynie 512 spośród nich za pomocą 512-punktowej FFT, powinniśmy dodać 24 kolejne próbki o wartościach zerowych do ciągu oryginalnego i użyć 1024-punktowej FFT. (Ta technika *uzupełniania zerami* jest bardziej szczegółowo omówiona w punkcie 3.11.)

FFT cierpi na takie same negatywne skutki przecieku widmowego, jakie omówiliśmy dla DFT w punkcie 3.8. Możemy przemnażać dane czasowe przez funkcje okien, aby złagodzić ten problem przecieku. Bądźmy jednak przygotowani na degradację rozdzielczości częstotliwościowej, nieuchronnie występującą, jeśli są używane okna. Przy okazji, jeśli dołączenie zer jest niezbędne do rozszerzenia

zbioru danych ciągu czasowego, musimy być pewni, że dołączamy zera *po* przemnożeniu oryginalnego ciągu danych czasowych przez funkcję okna. Zastosowanie funkcji okna do ciągu próbek uzupełnionego próbkami o zerowych wartościach zniekształci wynikowe okno i powiększy przeciek FFT.

Wprawdzie okienkowanie zmniejsza przeciek, to jednak nie eliminuje go całkowicie. Nawet gdy wykorzystuje się okienkowanie, składowe widmowe o wysokim poziomie mogą maskować pobliskie składowe widmowe o poziomie niskim. Jest to szczególnie wyraźnie widoczne, kiedy oryginalne dane czasowe mają niezerową wartość średnią, tj. są przesunięte o składową stałą. Jeśli w takim przypadku wyznacza się FFT, to składowa stała przy częstotliwości 0 Hz o dużej amplitudzie przysłoni jej widmowych sąsiadów. Możemy wyeliminować ten efekt przez obliczenie wartości średniej okienkowanego ciągu czasowego i odjęcie tej wartości średniej od każdej próbki oryginalnego sygnału okienkowanego [2)]. (Alternatywnie, wyliczenie wartości średniej i jej odjęcie może być równie dobrze przeprowadzone przed okienkowaniem.) Ta technika sprawia, że wartość średnia nowego ciągu czasowego staje się równa zeru i eliminuje w wynikach FFT każdą składową wysokiego poziomu przy częstotliwości 0 Hz.

4.2.3. Poprawianie wyników FFT

Jeśli używamy FFT aby wykryć sygnał w obecności szumu i dostępna jest wystarczająca liczba danych w dziedzinie czasu, możemy poprawić czułość przetwarzania przez uśrednienie wielokrotnych procedur FFT. Ta technika, omówiona w punkcie 10.7, może być zaimplementowana bardzo efektywnie, aby wykryć ten sygnał, który w rzeczywistości znajduje się poniżej uśrednionego poziomu szumu; to jest, mając wystarczająco dużo danych w dziedzinie czasu, możemy wykryć składowe sygnału, które charakteryzują się ujemną wartością stosunku sygnał/szum.

Jeśli nasze oryginalne dane w dziedzinie czasu mają jedynie wartości rzeczywiste, to możemy wykorzystać przewagę $2N$-punktowej rzeczywistej techniki FFT z punktu 10.5, aby przyspieszyć przetwarzanie; tj. $2N$-punktowy ciąg rzeczywisty może być przetransformowany za pomocą N-punktowego zespolonego FFT o podstawie 2. Zatem możemy otrzymać rozdzielczość częstotliwościową $2N$-punktowej FFT za obliczeniową cenę przeprowadzenia jedynie standardowej N-punktowej FFT. Innym sposobem poprawienia szybkości FFT jest możliwość użycia techniki okienkowania w dziedzinie częstotliwości, omówionej w punkcie 10.3. Jeśli wymagana jest FFT nieokienkowanych danych z dziedziny czasu i, jednocześnie, chcemy również wyznaczyć FFT tych samych danych czasowych z zastosowaniem funkcji okna, nie musimy przeprowadzać dwóch oddzielnych operacji FFT. Możemy wyznaczyć FFT danych nieokienkowanych, a następnie przeprowadzić okienkowanie w dziedzinie częstotliwości, aby zmniejszyć przeciek widmowy w dowolnym lub we wszystkich prążkach FFT.

[2)] Tę operację nazywa się zazwyczaj *centrowaniem* sygnału (*przyp. tłum.*).

4.2.4. Interpretacja wyników FFT

Pierwszym krokiem w interpretacji wyników FFT jest wyznaczenie w jednostkach bezwzględnych wartości częstotliwości środkowych kolejnych prążków FFT. Podobnie jak w DFT, rozłożenie prążków FFT wynika z ilorazu szybkości próbkowania (f_s) i liczby punktów FFT, czyli f_s/N. Oznaczając przez $X(m)$ wynik FFT, gdzie $m = 0, 1, 2, 3, \ldots, N-1$, bezwzględna częstotliwość środka m-tego prążka wynosi mf_s/N. Jeśli wejściowe próbki czasowe FFT są rzeczywiste, to niezależne są jedynie wartości wyjściowe $X(m)$ od $m = 0$ do $m = N/2$. Zatem, w tym przypadku, wymagane jest wyznaczenie w jednostkach bezwzględnych częstotliwości prążków FFT dla m jedynie w zakresie $0 \leqslant m \leqslant N/2$. Jeśli próbki wejściowe FFT są zespolone, to wszystkie N wartości wyjściowych FFT jest niezależnych i powinniśmy obliczyć bezwzględne częstotliwości prążków FFT dla m w pełnym zakresie $0 \leqslant m \leqslant N-1$.

Jeśli jest to konieczne, możemy wyznaczyć prawdziwą amplitudę sygnałów z dziedziny czasu na podstawie ich wyników widmowych FFT. Aby to uczynić, musimy pamiętać, że jeśli próbki wejściowe są rzeczywiste, to wyniki transformacji FFT o podstawie 2 są zespolone i mają postać

$$X(m) = X_{\text{real}}(m) + jX_{\text{imag}}(m) \tag{4.4}$$

Również wyjściowe próbki amplitudowe FFT

$$X_{\text{imag}} = |X(m)| = \sqrt{X_{\text{real}}(m)^2 + X_{\text{imag}}(m)^2} \tag{4.5}$$

są wszystkie samorzutnie mnożone przez czynnik $N/2$, jak to omówiono w punkcie 3.4 dla przypadku rzeczywistych próbek wejściowych. Jeśli próbki wejściowe FFT są zespolone, to czynnik skalujący jest równy N. Zatem aby wyznaczyć poprawne wartości amplitud składowych sinusoidalnych w dziedzinie czasu, musielibyśmy podzielić amplitudy FFT przez odpowiedni czynnik skalujący, równy $N/2$ dla rzeczywistych sygnałów wejściowych lub równy N dla zespolonych sygnałów wejściowych.

Jeśli do oryginalnych danych z dziedziny czasu została zastosowana funkcja okna, to niektóre próbki ciągu wejściowego procedury FFT zostaną stłumione. Powoduje to zmniejszenie wynikowych amplitud wyjściowych FFT w stosunku do ich prawdziwych nieokienkowych wartości. Aby wyliczyć prawidłowe amplitudy różnych składowych sinusoidalnych w dziedzinie czasu, musielibyśmy wówczas w dalszym ciągu podzielić amplitudy FFT przez odpowiedni współczynnik straty przetwarzania, stowarzyszony z użytą funkcją okna. Współczynniki straty przetwarzania dla najpopularniejszych funkcji okien są zestawione w pozycji [3].

Jeśli chcielibyśmy wyznaczyć wartości widma mocy $X_{\text{PS}}(m)$ wyników FFT, to należałoby wyliczyć wartości kwadratów amplitud za pomocą wyrażenia

$$X_{\text{PS}}(m) = |X(m)|^2 = X_{\text{real}}(m)^2 + X_{\text{imag}}(m)^2 \tag{4.6}$$

Pozwala nam to obliczyć widmo mocy w decybelach jako

$$X_{\text{dB}}(m) = 10 \cdot \log_{10}(|X(m)|^2)\ \text{dB} \tag{4.7}$$

Unormowane widmo mocy w decybelach można obliczyć używając wzoru

$$\text{unormowane } X_{\text{dB}}(m) = 10 \cdot \log_{10}\left(\frac{|X(m)|^2}{(|X(m)|_{\max})^2}\right) \tag{4.8}$$

lub

$$\text{unormowane } X_{\text{dB}}(m) = 20 \cdot \log_{10}\left(\frac{|X(m)|}{(|X(m)|_{\max})}\right) \tag{4.9}$$

W równaniach (4.8) i (4.9) człon $|X(m)|_{\max}$ jest próbką wyjściową FFT o największej wartości. W praktyce okazuje się, że wykreślenie X_{dB} jest znacznie bardziej mówiące z uwagi na polepszoną rozdzielczość w zakresie niskich amplitud, osiągniętą dzięki decybelowej skali logarytmicznej, jak pokazano w dodatku E. W przypadku użycia zarówno równania (4.8), jak i (4.9) nie jest potrzebne przeprowadzenie żadnej kompensacji dla wspomnianego czynnika skalującego N lub $N/2$, ani też okienkowego czynnika straty przetwarzania. Normalizacja na drodze dzielenia przez $(|X(m)|_{\max})^2$ lub $|X(m)|_{\max}$ eliminuje efekt wszystkich czynników skalujących FFT lub okna.

Wiedząc, że kąty fazowe $X_\phi(m)$ poszczególnych wartości wyjściowych FFT są dane jako

$$X_\phi(m) = \text{arctg}\left(\frac{X_{\text{imag}}(m)}{X_{\text{real}}(m)}\right) \tag{4.10}$$

ważne jest, aby zważać na zerowe wartości $X_{\text{real}}(m)$. Wartości te uczyniłyby niesłusznymi obliczenie kątów fazowych z równania (4.10) ze względu na warunek dzielenia przez zero. W praktyce chcemy mieć pewność, że nasze obliczenia (lub kompilatory programowe) wykryją pojawienie się wartości $X_{\text{real}}(m) = 0$ i wstawią za $X_\phi(m)$ wartości: 90° jeśli $X_{\text{imag}}(m)$ jest dodatnie, 0° jeśli $X_{\text{imag}}(m)$ jest równe zeru, −90° jeśli $X_{\text{imag}}(m)$ jest ujemne. Jeśli już jesteśmy przy temacie kątów fazowych wyjściowych wartości FFT, to bądźmy świadomi, że wartości wyjściowe FFT zawierające znaczące składowe szumu mogą powodować duże fluktuacje wyliczonych kątów fazowych $X_\phi(m)$. Oznacza to, że próbki $X_\phi(m)$ są wiarygodne tylko wówczas, gdy odpowiadające wartości $|X(m)|$ leżą wystarczająco powyżej uśrednionego poziomu szumu wyjściowego FFT.

4.3. Oprogramowanie implementujące FFT

Dla Czytelników poszukujących podprogramów implementujących FFT, bez konieczności zakupu kosztownych pakietów programowych z zakresu przetwarzania sygnałów, są łatwo dostępne podprogramy algorytmu FFT o podstawie 2, typu *public domain*. W pozycjach literatury [4 ÷ 7] zamieszczono postaci źródłowe standardowych programów FFT w języku FORTRAN. W publikacji [8] zaprezentowano efektywny program FFT napisany w FORTRANIE dla jedynie rzeczywis-

tych ciągów danych wejściowych. Pozycja [9] oferuje standardowy program FFT, napisany w języku HP BASIC™, publikacja [10] zaś prezentuje podprogram FFT napisany w Applesoft BASIC™. Czytelnicy interesujący się językiem Ada mogą znaleźć podprogramy związane z FFT w pozycji [11].

4.4. Wyprowadzenie algorytmu FFT o podstawie 2

Ten punkt, jak i punkty następne obejmują szczegółową prezentację wewnętrznych struktur danych i operacji algorytmu FFT o podstawie 2 dla tych Czytelników, którzy są zainteresowani wytworzeniem podprogramów FFT lub zaprojektowaniem sprzętowych implementacji algorytmu FFT. Aby dokładnie zobaczyć, jak algorytm FFT wyewoluował z DFT, wróćmy do równania dla N-punktowej DFT

$$X(m) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi nm/N} \tag{4.11}$$

Bezpośrednie wyprowadzenie FFT jest oparte na podziale ciągu $x(n)$ danych wejściowych na dwie części. Gdy ciąg $x(n)$ zostaje rozdzielony na elementy indeksowane parzyście i nieparzyście, możemy podzielić równanie (4.11) na dwie części jako

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)\, e^{-j2\pi(2n)m/N} + \sum_{n=0}^{(N/2)-1} x(2n+1)\, e^{-j2\pi(2n+1)m/N} \tag{4.12}$$

Wyłączając stały kąt fazowy przed drugi znak sumowania, mamy

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)\, e^{-j2\pi(2n)m/N} + e^{-j2\pi m/N} \sum_{n=0}^{(N/2)-1} x(2n+1)\, e^{-j2\pi(2n+1)m/N} \tag{4.13}$$

Cóż, równania te stają się tak długie, że dla uproszczenia użyjemy standardowej notacji. Przyjmiemy oznaczenie $W_N = e^{-j2\pi/N}$ w celu reprezentacji zespolonego czynnika kąta fazowego, który jest stały dla danego N. Wówczas równanie (4.13) przyjmie postać

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)\, W_N^{2nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)\, W_N^{2nm} \tag{4.14}$$

Ponieważ $W_N^2 = e^{-j2\pi 2/(N)} = e^{-j2\pi/(N/2)}$, możemy podstawić $W_{N/2}$ za $W_{N/2}^2$ w równaniu (4.14), otrzymując

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)\, W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)\, W_{N/2}^{nm} \tag{4.15}$$

A więc mamy teraz dwa sumowania o liczbie składników równej $N/2$, których wyniki rozważone łącznie dają nam N-punktową DFT. W równaniu (4.15) zredukowaliśmy niektóre z operacji w porównaniu do równania (4.11), ponieważ

człony typu W w obydwu sumowaniach w równaniu (4.15) są identyczne. Istnieje dalsza korzyść z podziału N-punktowej DFT na dwie części, ponieważ górna połowa wyników DFT jest łatwa do obliczenia. Rozważmy wartość wyjściową $X(m+N/2)$. Jeśli wstawimy $m+N/2$ za m w równaniu (4.15), wówczas

$$X(m+N/2) = \sum_{n=0}^{(N/2)-1} x(2n)\, W_{N/2}^{n(m+N/2)} + \\ + W_N^{(m+N/2)} \sum_{n=0}^{(N/2)-1} x(2n+1)\, W_{N/2}^{n(m+N/2)} \tag{4.16}$$

Wygląda to, jakbyśmy wszystko skomplikowali, nieprawdaż? Zatem zatrzymajmy się na chwilę. Możemy obecnie uprościć człony kątów fazowych wewnątrz sumowań, ponieważ

$$W_{N/2}^{n(m+N/2)} = W_{N/2}^{nm}\, W_{N/2}^{nN/2} = W_{N/2}^{nm}\,(e^{-j2\pi n\, 2N/2N}) = \\ = W_{N/2}^{nm}(1) = W_{N/2}^{nm} \tag{4.17}$$

dla dowolnej liczby całkowitej n. Przyglądając się tak zwanemu *współczynnikowi obrotu* przed znakiem drugiej sumy w równaniu (4.16), możemy go uprościć jako

$$W_{N/2}^{m+N/2} = W_N^m\, W_N^{N/2} = W_N^m\,(e^{-j2\pi N/2N}) = W_N^m\,(-1) = -W_N^m \tag{4.18}$$

Wykorzystując równania (4.17) i (4.18), możemy przedstawić $X(m+N/2)$ z równania (4.16) jako

$$X(m+N/2) = \sum_{n=0}^{(N/2)-1} x(2n)\, W_{N/2}^{nm} - W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)\, W_{N/2}^{nm} \tag{4.19}$$

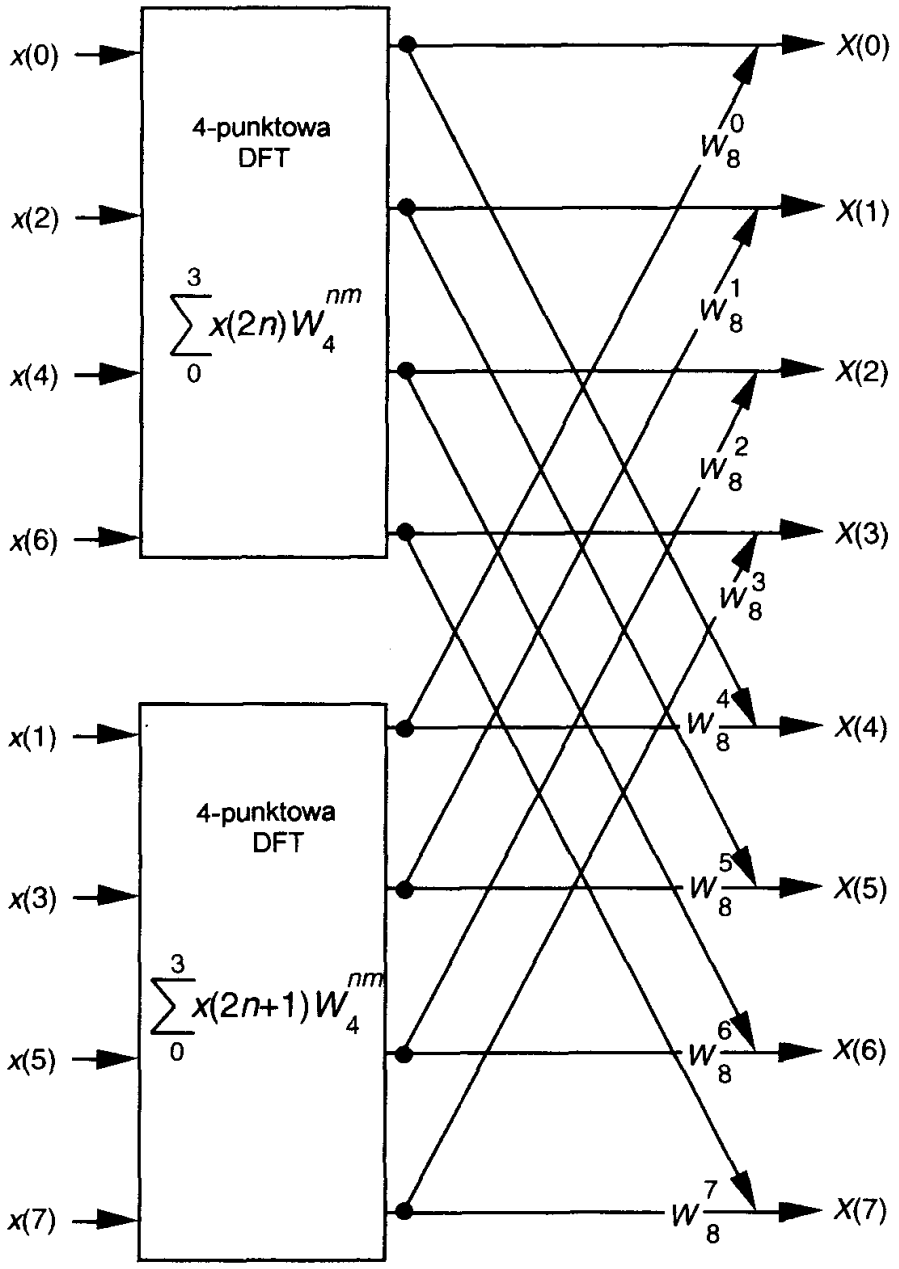
Powtórzmy tutaj równania (4.15) i (4.19), aby zobaczyć podobieństwo wyrażeń

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)\, W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)\, W_{N/2}^{nm} \tag{4.20}$$

i

$$X(m+N/2) = \sum_{n=0}^{(N/2)-1} x(2n)\, W_{N/2}^{nm} - W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)\, W_{N/2}^{nm} \tag{4.20'}$$

A więc dotarliśmy do celu. Nie musimy przeprowadzać żadnych mnożeń sinusowych, ani cosinusowych, aby otrzymać $X(m+N/2)$. Po prostu zmieniamy znak współczynnika obrotu W_N^m i wykorzystujemy wyniki dwóch sumowań z $X(m)$, aby otrzymać $X(m+N/2)$. Oczywiście m zmienia się w równaniu (4.20) od 0 do $(N/2)-1$, co oznacza dla N-punktowej DFT, że przeprowadzamy $N/2$-punktową DFT aby otrzymać pierwsze $N/2$ wartości wyjściowych i używamy tychże, aby otrzymać ostatnie $N/2$ wartości wyjściowych. Dla $N = 8$ równania (4.20) i (4.20') są implementowane w postaci pokazanej na rys. 4.2.



Rys. 4.2. Implementacja 8-punktowej DFT za pomocą FFT z użyciem dwóch 4-punktowych DFT

Jeśli uprościmy równania (4.20) i (4.20′) do postaci

$$X(m) = A(m) + W_N^m B(m) \tag{4.21}$$

i

$$X(m+N/2) = A(m) - W_N^m B(m) \tag{4.21'}$$

możemy pójść dalej i pomyśleć o podpodziale tych dwóch 4-punktowych DFT na cztery 2-punktowe DFT. Zobaczmy, jak możemy podzielić górną 4-punktową DFT na rys. 4.2, której czterema wartościami wyjściowymi są $A(m)$ w równaniach (4.21) i (4.21′). Dokonujemy podziału wartości wejściowych górnej 4-punktowej DFT na ich składowe nieparzyste i parzyste

$$A(m) = \sum_{n=0}^{(N/2)-1} x(2n)\, W_{N/2}^{nm} = \sum_{n=0}^{(N/4)-1} x(4n)\, W_{N/2}^{4nm} + \sum_{n=0}^{(N/4)-1} x(4n+1)\, W_{N/2}^{4nm}$$

lub

$$A(m) = \sum_{2n=0}^{(N/4)-1} x(4n)\, W_{N/4}^{2nm} + W_{N/2}^{m} \sum_{2n=0}^{(N/4)-1} x(4n+1)\, W_{N/4}^{2nm} \tag{4.22}$$

Jeśli przyjmiemy oznaczenie $p = 2n$, to indeks sumowania w równaniu (4.22) może zostać uproszczony, pozwalając wyrazić $A(m)$ w bardziej nam znanej postaci

$$A(m) = \sum_{p=0}^{(N/4)-1} x(2p)\, W_{N/4}^{pm} + W_{N/2}^{m} \sum_{p=0}^{(N/4)-1} x(2p+1)\, W_{N/4}^{pm} \tag{4.23}$$

Zauważmy podobieństwo pomiędzy równaniem (4.23) i równaniem (4.20). Ta zdolność podpodziału $N/2$ DFT na dwie $N/4$-punktowe DFT daje FFT możliwość ogromnego zmniejszenia liczby mnożeń niezbędnych do zaimplementowania DFT. (Mamy zamiar krótko tego dowieść.) Postępując według takich samych kroków, jakich użyliśmy aby otrzymać $A(m)$, możemy pokazać, że $B(m)$ z równania (4.21) wyraża się jako

$$B(m) = \sum_{p=0}^{(N/4)-1} x(2p)\, W_{N/4}^{pm} - W_{N/2}^{m} \sum_{p=0}^{(N/4)-1} x(2p+1)\, W_{N/4}^{pm} \tag{4.24}$$

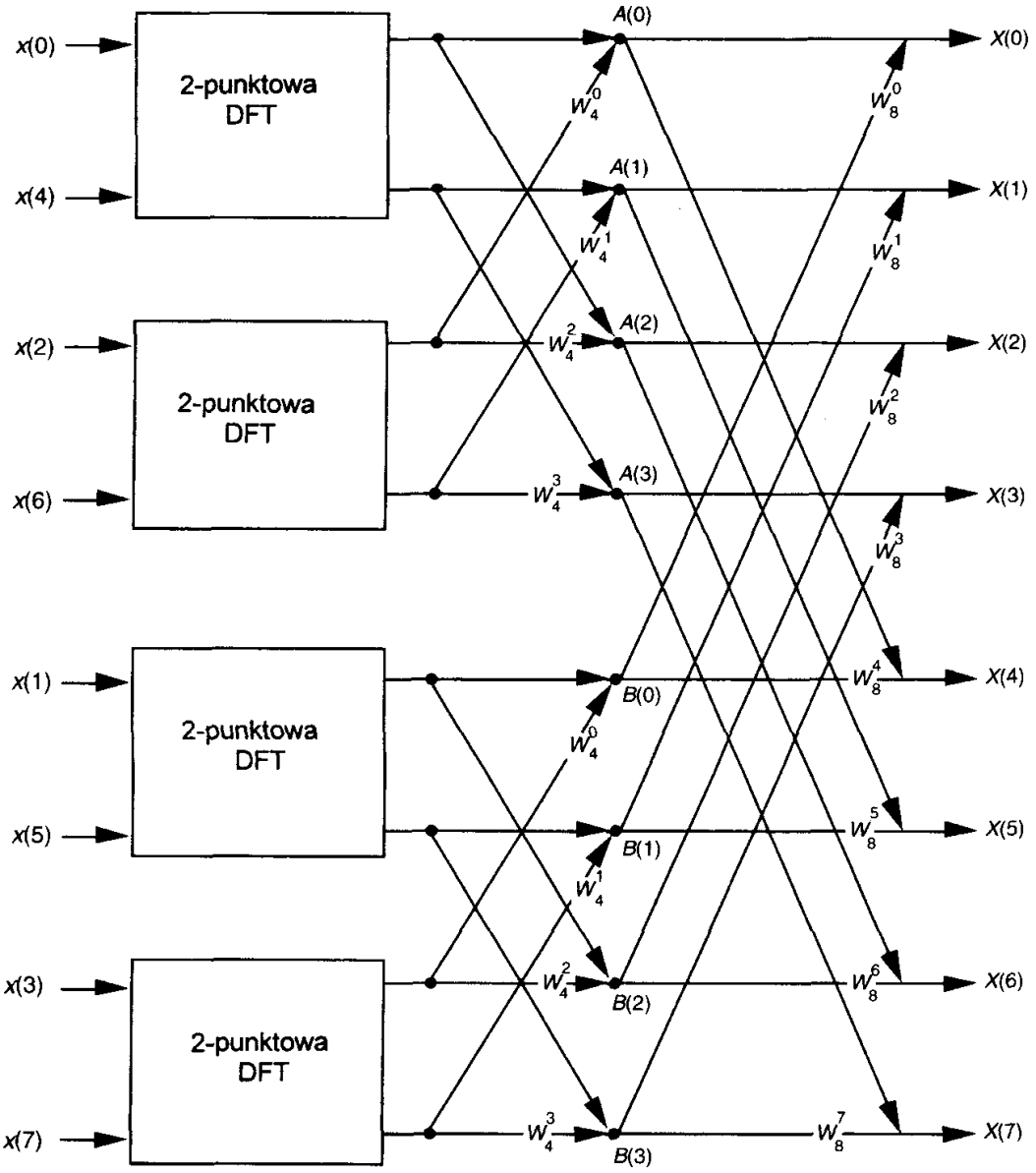
W rozważanym przykładzie dla $N = 8$, równania (4.23) i (4.24) są implementowane w sposób pokazany na rys. 4.3. Oczywisty staje się wzorzec przepływu sygnałów w postaci dobrze znanego *motylka* FFT i możemy zobaczyć dalsze przemieszanie danych wejściowych na rys. 4.3. W przykładzie tym dla $N = 8$ współczynnik obrotu $W_{N/2}^{m}$ zmienia się w zakresie od W_4^0 do W_4^3, ponieważ indeks m dla $A(m)$ i $B(m)$ zmienia się od 0 do 3. Dla dowolnej N-punktowej DFT, możemy rozdzielić każdą z $N/2$-punktowych DFT na dwie $N/4$-punktowe DFT aby dalej zmniejszyć liczbę mnożeń sinusowych i cosinusowych. Ostatecznie docieramy do uszeregowania 2-punktowych DFT, gdzie nie można już osiągnąć dalszych oszczędności obliczeniowych. Oto dlaczego liczba punktów w procedurach DFT jest ograniczona do potęg liczby 2 i dlaczego ten algorytm FFT jest nazywany algorytmem FFT o podstawie 2.

Wykonamy teraz kolejny krok dalej i wówczas zakończymy wyprowadzenie naszej $N = 8$-punktowej FFT. Operacje 2-punktowych DFT, przedstawione na rys. 4.3, nie mogą być już podzielone na mniejsze części – osiągnęliśmy oto koniec naszego procesu redukcji DFT, dochodząc do motylka pojedynczej 2-punktowej DFT, jak pokazano na rys. 4.4.

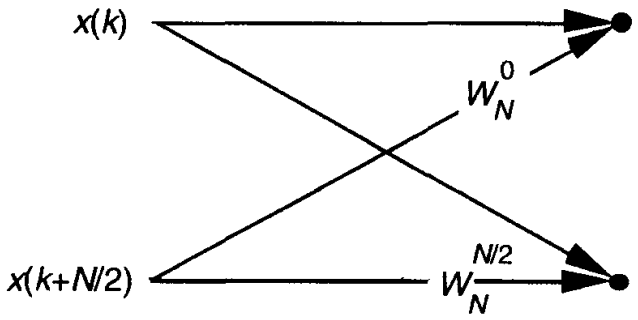
Z definicji W_N mamy $W_N^0 = e^{-j2\pi 0/N}$ i $W_N^{N/2} = e^{-j2\pi N/2N} = e^{-j\pi} = -1$. Zatem bloki 2-punktowej DFT na rys. 4.3 mogą być zamienione na motylki z rys. 4.4, aby dać nam pełną implementację DFT w postaci 8-punktowej FFT, jak pokazno na rys. 4.5.

Przebrnęliśmy tutaj przez niezły bigos albegraiczny. Aby sprawdzić, że wyprowadzenie FFT jest słuszne, możemy poddać 8-punktowy ciąg danych z przykładu 1 w rozdziale 3 procedurze 8-punktowej FFT, przedstawionej na rys. 4.5. Ciąg danych reprezentujących $x(n) = \sin(2\pi \cdot 1000 \cdot nt_s) + 0{,}5\sin(2\pi \cdot 2000 \cdot nt_s + 3\pi/4)$, to

$$\begin{aligned} x(0) &= 0{,}3535 \qquad x(1) = 0{,}3535 \\ x(2) &= 0{,}6464 \qquad x(3) = 1{,}0607 \\ x(4) &= 0{,}3535 \qquad x(5) = -1{,}0607 \\ x(6) &= -1{,}3535 \qquad x(7) = -0{,}3535 \end{aligned} \tag{4.25}$$



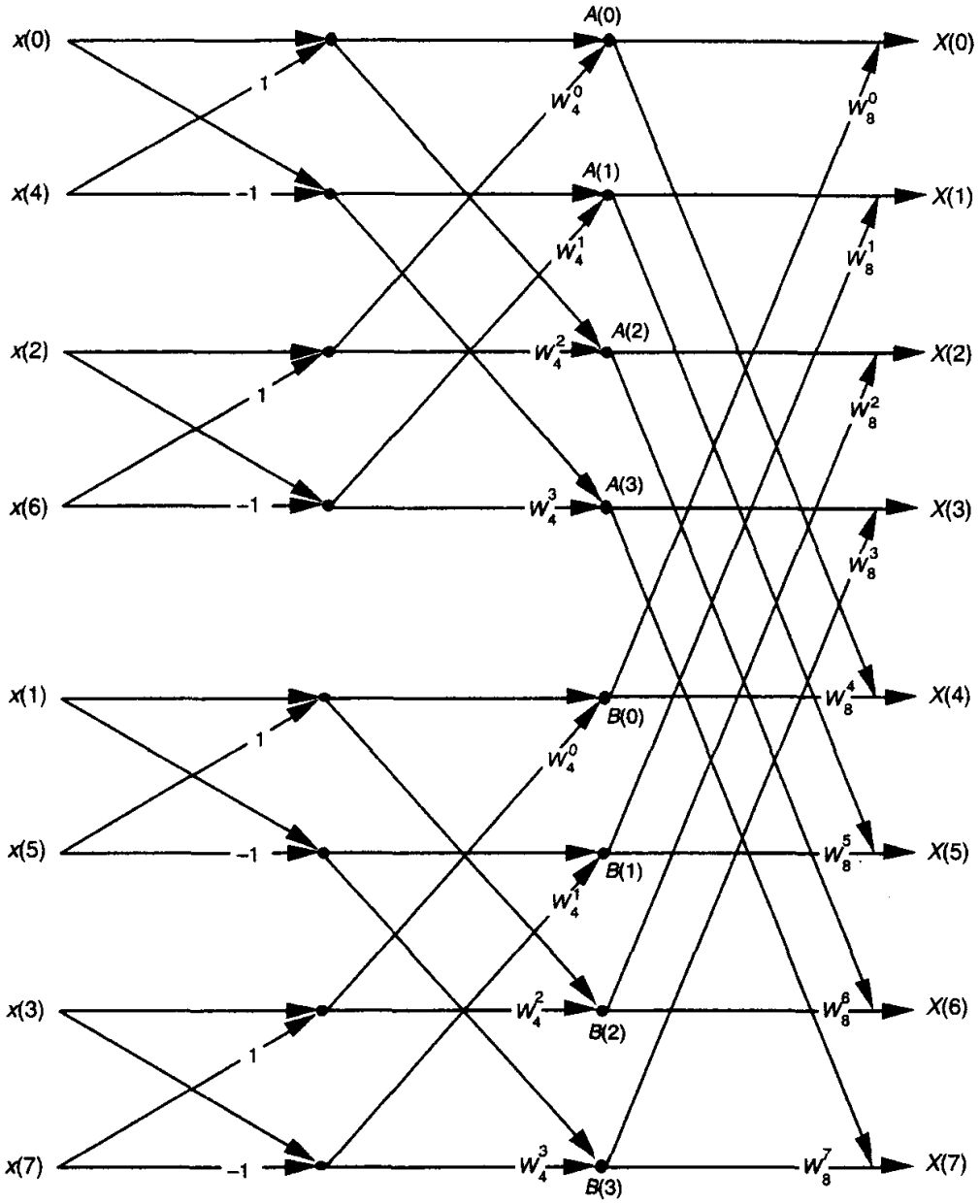
Rys. 4.3. Implementacja 8-punktowej DFT za pomocą FFT jako dwie 4-punktowe DFT i cztery 2-punktowe DFT



Rys. 4.4. Pojedynczy motylek 2-punktowej DFT

Zaczynamy przegryzienie się przez ten przykład od podania wejściowych wartości z równania (4.25) na zaciski wejściowe motylków FFT na rys. 4.5, dającego wartości danych pokazane po lewej stronie rys. 4.6. Wartości wyjściowe drugiego kroku FFT, to

$$
\begin{aligned}
A(0) &= 0{,}707 + W_4^0(-0{,}707) = 0{,}707 + (1+j0)(-0{,}707) = 0+j0\\
A(1) &= 0{,}0 + W_4^1(1{,}999) = 0{,}0 + (0-j1)(1{,}999) = 0-j1{,}999\\
A(2) &= 0{,}707 + W_4^2(-0{,}707) = 0{,}707 + (-1+j0)(-0{,}707) = 1{,}414+j0\\
A(3) &= 0{,}0 + W_4^3(1{,}999) = 0{,}0 + (0+j1)(1{,}999) = 0+j1{,}999\\
B(0) &= -0{,}707 + W_4^0(0{,}707) = -0{,}707 + (1+j0)(0{,}707) = 0+j0\\
B(1) &= 1{,}414 + W_4^1(1{,}414) = 1{,}414 + (0-j1)(1{,}414) = 1{,}414-j1{,}414\\
B(2) &= -0{,}707 + W_4^2(0{,}707) = -0{,}707 + (-1+j0)(0{,}707) = -1{,}414+j0\\
B(3) &= 1{,}414 + W_4^3(1{,}414) = 1{,}414 + (0+j1)(1{,}414) = 1{,}414+j1{,}414
\end{aligned}
$$



Rys. 4.5. Całkowita implementacja 8-punktowej DFT za pomocą FFT z podziałem w czasie

Wyliczając wartości wyjściowe trzeciego kroku FFT dochodzimy do końcowych wyników

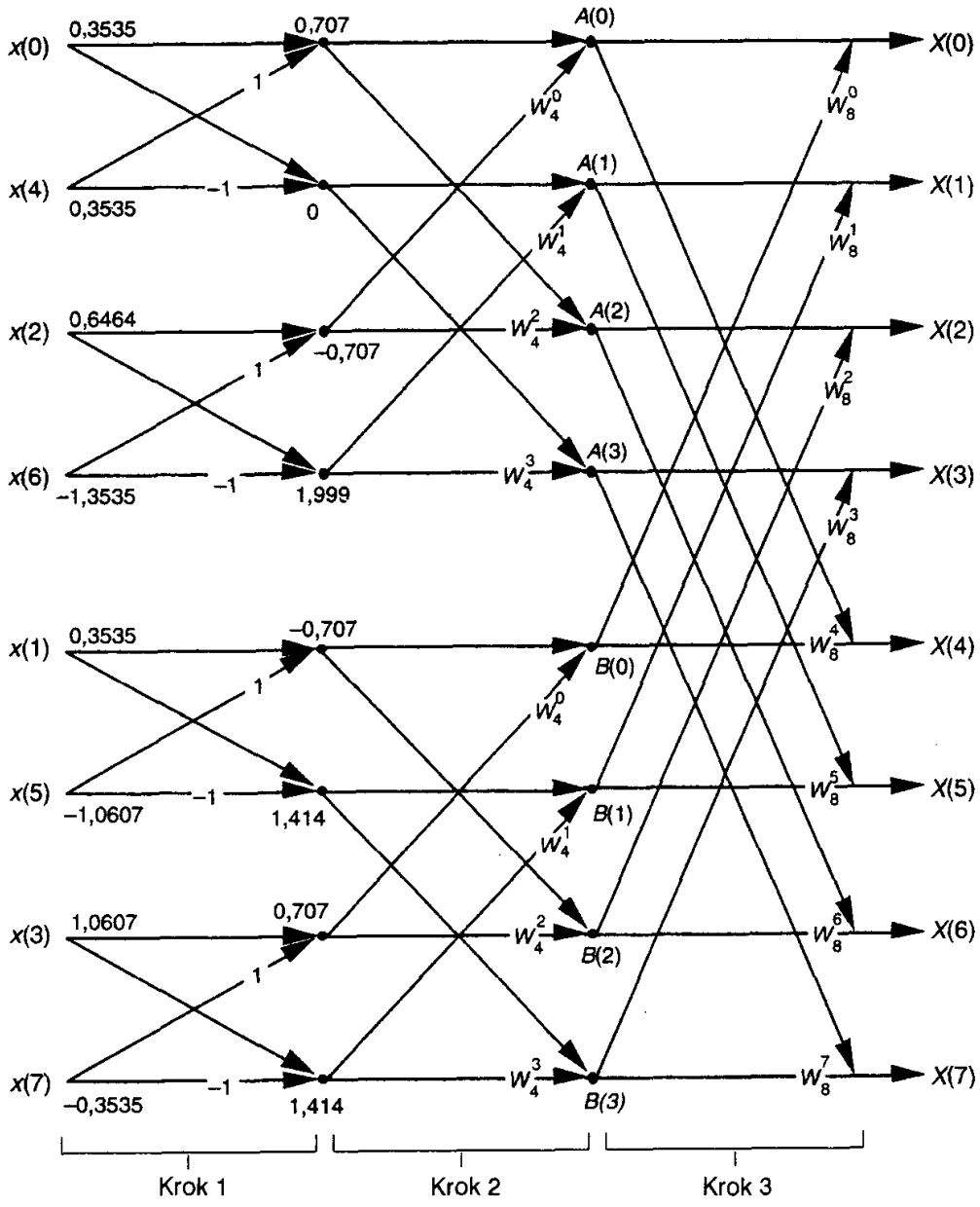
$$X(0) = A(0) + W_8^0 B(0) = 0 + j0 + (1 + j0)(0 + j0) = 0 + j0 + 0 + j0 = 0 \angle 0°$$

$$X(1) = A(1) + W_8^1 B(1) = 0 - j1{,}999 + (0{,}707 - j0{,}707)(1{,}414 - j1{,}414) =$$
$$= 0 - j1{,}999 + 0 - j1{,}999 = 0 - j4 = 4 \angle -90°$$

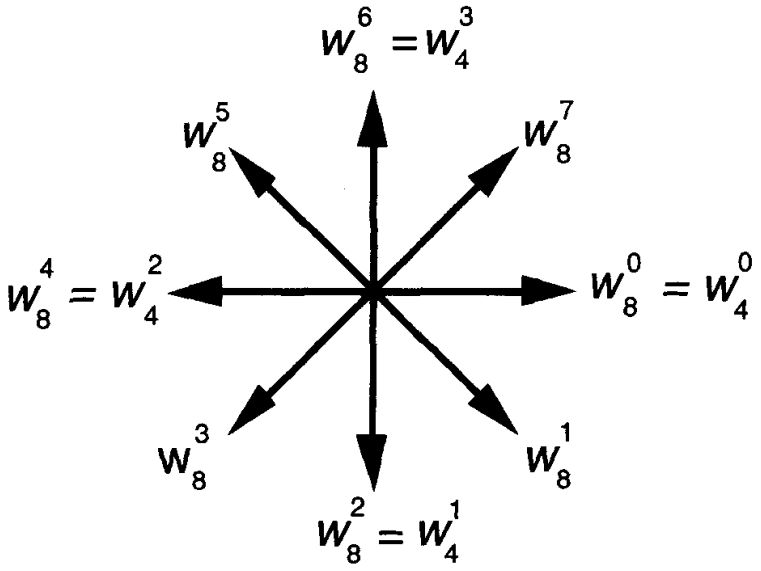
$$X(2) = A(2) + W_8^2 B(2) = 1{,}414 + j0 + (0 - j1)(-1{,}414 + j0) =$$
$$= 1{,}414 + j0 + 0 + j1{,}414 = 1{,}414 + j1{,}414 = 2 \angle 45°$$

$$X(3) = A(3) + W_8^3 B(3) = 0 + j1{,}999 + (-0{,}707 - j0{,}707)(1{,}414 + j1{,}414) =$$
$$= 0 + j1{,}999 + 0 - j1{,}999 = 0 \angle 0°$$

$$X(4) = A(0) + W_8^4 B(0) = 0 + j0 + (-1 + j0)(0 + j0) =$$
$$= 0 + j0 + 0 + j0 = 0 \angle 0°$$



Rys. 4.6. 8-punktowa FFT dla przykładu 1 z paragrafu 3.1



Rys. 4.7. Okresowe nadmiarowości współczynników obrotu 8-punktowej FFT

$$X(5) = A(1) + W_8^5 B(1) = 0 - j1{,}999 + (-0{,}707 + j0{,}707)(1{,}414 - j1{,}414) =$$
$$= 0 - j1{,}999 + 0 + j1{,}999 = 0 \angle 0°$$

$$X(6) = A(2) + W_8^6 B(2) = 1{,}414 + j0 + (0 + j1)(-1{,}414 + j0) =$$
$$= 1{,}414 + j0 + 0 - j1{,}414 = 1{,}414 - j1{,}414 = 2 \angle -45°$$

$$X(7) = A(3) + W_8^7 B(3) = 0 + j1{,}999 + (0{,}707 + j0{,}707)(1{,}414 + j1{,}414) =$$
$$= 0 + j1{,}999 + 0 + j1{,}999 = 0 + j4 = 4 \angle 90°$$

A więc, na szczęście, FFT daje poprawne wyniki i znów przypominamy Czytelnikowi, że FFT nie stanowi przybliżenia DFT; to jest DFT ze zmniejszoną liczbą niezbędnych operacji arytmetycznych. Z powyższego wynika, że przykład 8-punktowej FFT wymaga mniej wysiłku, niż rozważony w punkcie 3.1 w rozdziale 3 przykład 8-punktowej DFT. Niektórzy autorzy lubią wyjaśniać tę redukcję arytmetyczną poprzez wewnętrzne nadmiarowości we współczynniku obrotu W_N^m. Ilustrują to za pomocą wzorca *gwiaździstego*, pokazanego na rys. 4.7, pokazując równoważności niektórych współczynników obrotu w 8-punktowej DFT.

4.5. Odwrócenie bitowe indeksu danych wejściowych/wyjściowych FFT

Spójrzmy na niektóre szczególne właściwości FFT, które są istotne dla twórców oprogramowania FFT i projektantów sprzętu implementującego FFT. Zauważmy, że rys. 4.5 zatytułowano „Pełna implementacja 8-punktowej DFT za pomocą FFT z podziałem w czasie". Wyrażenie *podzial w czasie* odnosi się do tego, w jaki sposób dokonaliśmy podziału wejściowych próbek DFT na część nieparzystą i parzystą przy wyprowadzeniu równań (4.2), (4.23) i (4.24). Ten podział w czasie prowadzi do przemieszanego uporządkowania względem indeksu n danych wejściowych na rys. 4.5. Zasadę tego przemieszanego porządku można zrozumieć

Tablica 4.1. ODWRÓCENIE BITOWE INDEKSU WEJŚCIOWEGO DLA 8-PUNKTOWEJ FFT

Uporządkowanie normalne indeksu n	*Bity dwójkowe indeksu n*	*Odwrócone bity indeksu n*	*Odwrócone bitowo uporządkowanie indeksu n*
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

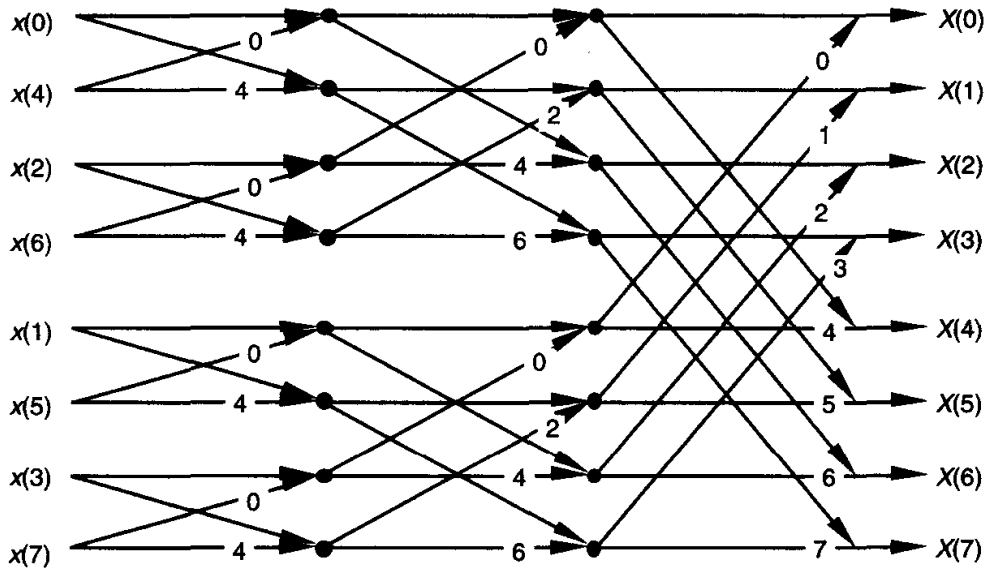
z pomocą tablicy 4.1. Przemieszanie danych wejściowych jest znane jako odwrócenie bitowe, ponieważ takie przemieszane uporządkowanie indeksu danych wejściowych może być otrzymane na drodze odwrócenia bitów dwójkowej reprezentacji normalnego uporządkowania indeksu danych wejściowych. Brzmi to myląco, ale naprawdę nie jest – tablica 4.1 ilustruje odwrócenie bitowe dla naszego przykładu 8-punktowej FFT.

Zauważmy normalne uporządkowanie indeksu w lewej kolumnie tablicy 4.1 i przemieszane uporządkowanie w prawej kolumnie, która odpowiada ostatecznemu uporządkowaniu indeksu wejściowego na rys. 4.5. Przetransponowaliśmy oryginalne bity dwójkowe, reprezentujące normalne uporządkowanie indeksu przez zamianę ich pozycji. Bit najstarszy staje się bitem najmłodszym, bit najmłodszy zaś staje się bitem najstarszym, bit następny względem bitu najstarszego staje się następnym względem bitu najmłodszego, a bit następny względem bitu najmłodszego staje się następnym względem bitu najstarszego itd. [3)].

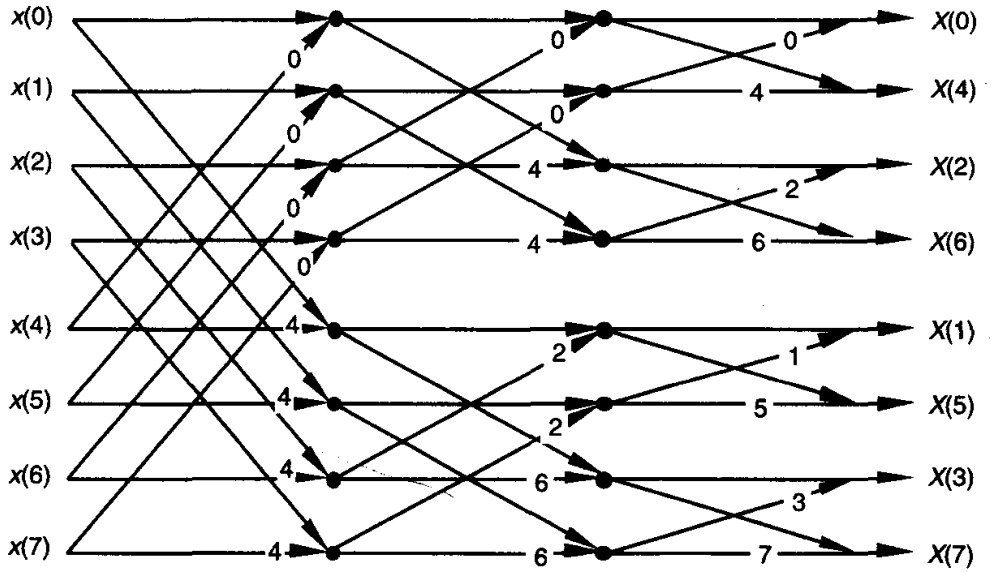
4.6. Struktury motylkowe algorytmu FFT o podstawie 2

Zbadajmy nieco głębiej struktury motylkowe przepływu sygnałów FFT z podziałem w czasie. Aby uprościć oznaczenia przepływów sygnałów, zamieńmy współczynniki obrotu na rys. 4.5 ich równoważnymi wartościami, odniesionymi do W_N^m, gdy $N = 8$. Możemy umieścić tylko wykładniki m współczynników W_N^m, aby otrzymać strukturę FFT pokazaną na rys. 4.8. Innymi słowy, współczynnik W_4^1 z rys. 4.5 jest równy W_8^2 i został przedstawiony jako 2 na rys. 4,8, a współczynnik W_4^2 z rys. 4.5 jest równy W_8^4 i został przedstawiony jako 4 na rys. 4.8 itd. Wartości 1 i -1 w pierwszym kroku algorytmu z rys. 4.5 są zamienione na rys. 4.8 przez,

[3)] Wielu z tych, którzy są pierwszymi będą ostatnimi, a ostatni pierwszymi [Ewangelia Św. Marka 10:31].



Rys. 4.8. 8-punktowa FFT z podziałem w czasie, z odwróconymi bitowo wartościami wejściowymi

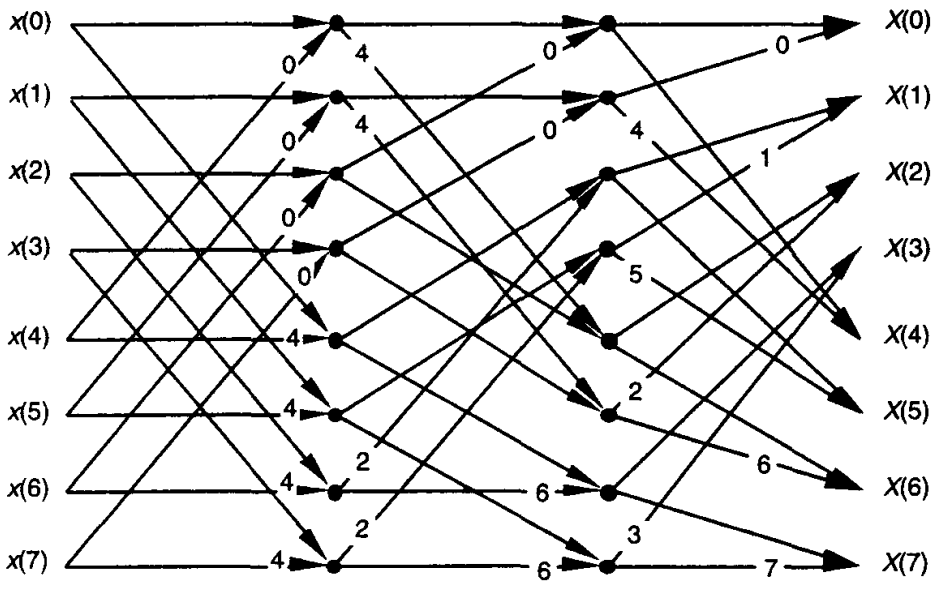


Rys. 4.9. 8-punktowa FFT z podziałem w czasie, z odwróconymi bitowo wartościami wyjściowymi

odpowiednio, wartości 0 i 4. Oprócz innej notacji współczynnika obrotu, rys. 4.8 jest identyczny z rys. 4.5. Możemy cyklicznie przesunąć węzły sygnału na rys. 4.5 i otrzymać 8-punktową FFT z podziałem w czasie, w postaci pokazanej na rys. 4.9. Zauważmy, że dane wejściowe na rys. 4.9 są uporządkowane normalnie, indeksy danych wyjściowych zaś są odwrócone bitowo. W tym przypadku operacja odwrócenia bitowego musi być przeprowadzona na wartościach wyjściowych FFT, aby przywrócić normalne uporządkowanie wyników w dziedzinie częstotliwości.

Na rysunku 4.10 pokazano graf przepływowy sygnałów FFT, który wprawdzie całkowicie unika problemu odwracania bitowego, lecz pełna wdzięku tkanina tradycyjnych motylków FFT zostaje zamieniona na poplątaną wprawdzie, lecz efektywną konfigurację.

Jeszcze nie tak dawno w implementacjach sprzętowych FFT większość czasu (cykli zegarowych) pochłaniało przeprowadzanie mnożeń, proces zaś odwracania



Rys. 4.10. 8-punktowa FFT z podziałem w czasie, z wartościami wejściowymi i wyjściowymi o normalnym uporządkowaniu

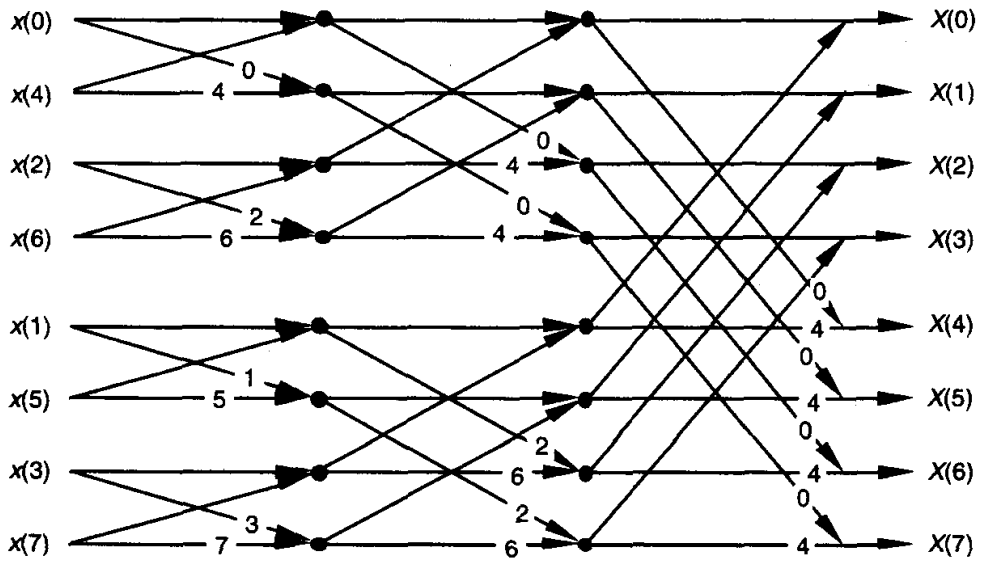
bitowego, niezbędny do uzyskania dostępu do danych w pamięci nie stanowił znaczącej części całego problemu obliczeniowego FFT.

Ponieważ obecnie wysokiej szybkości układy scalone mnożnik/akumulator mogą mnożyć dwie liczby w ciągu jednego cyklu zegara, multipleksowanie danych FFT i adresacja pamięci stały się znacznie ważniejsze. Doprowadziło to do rozwinięcia wydajnych algorytmów przeprowadzania odwracania bitowego [12 ÷ 15].

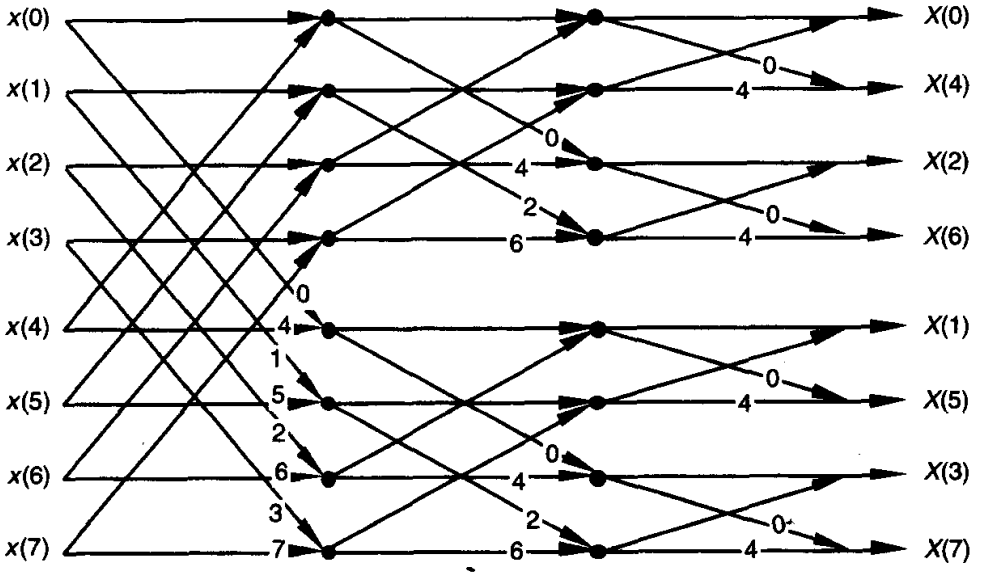
Istnieje inne wyprowadzenie algorytmu FFT, które prowadzi do struktur motylkowych, wyglądających podobnie jak te, które już omówiliśmy, lecz współczynniki obrotu w tych motylkach są różne. Ta alternatywna technika FFT jest znana jako algorytm podziału w częstotliwości. O ile algorytm FFT z podziałem w czasie jest oparty na podpodziale danych wejściowych na ich składowe o indeksach nieparzystych i parzystych, to algorytm FFT z podziałem w częstotliwości jest oparty na wyliczeniu osobno nieparzystych oraz parzystych próbek wyjściowych w dziedzinie częstotliwości. Wyprowadzenie algorytmu z podziałem w częstotliwości jest proste i przedstawione w licznych artykułach przeglądowych i podręcznikach [4, 5, 15, 16], zatem nie będziemy śledzić tego wyprowadzenia tutaj. Jednakowoż, zilustrujemy na rys. od 4.11 do 4.13 struktury motylkowe podziału w częstotliwości (analogicznie do struktur na rys. od 4.8 do 4.10).

A więc dla każdej struktury FFT z podziałem w czasie istnieje równoważna struktura FFT z podziałem w częstotliwości. Ważne jest, aby zauważyć, że liczba mnożeń niezbędnych do implementacji FFT z podziałem w częstotliwości jest taka sama, jak liczba niezbędna dla algorytmów FFT z podziałem w czasie. W literaturze przedstawiono tak wiele różnych struktur motylkowych, że można łatwo pomylić się co do tego, które struktury dotyczą podziału w czasie, a które podziału w częstotliwości.

W zależności od tego, jak ten materiał jest przedstawiany, początkującemu łatwo jest wpaść w pułapkę i uwierzyć, że algorytmy FFT z podziałem w czasie



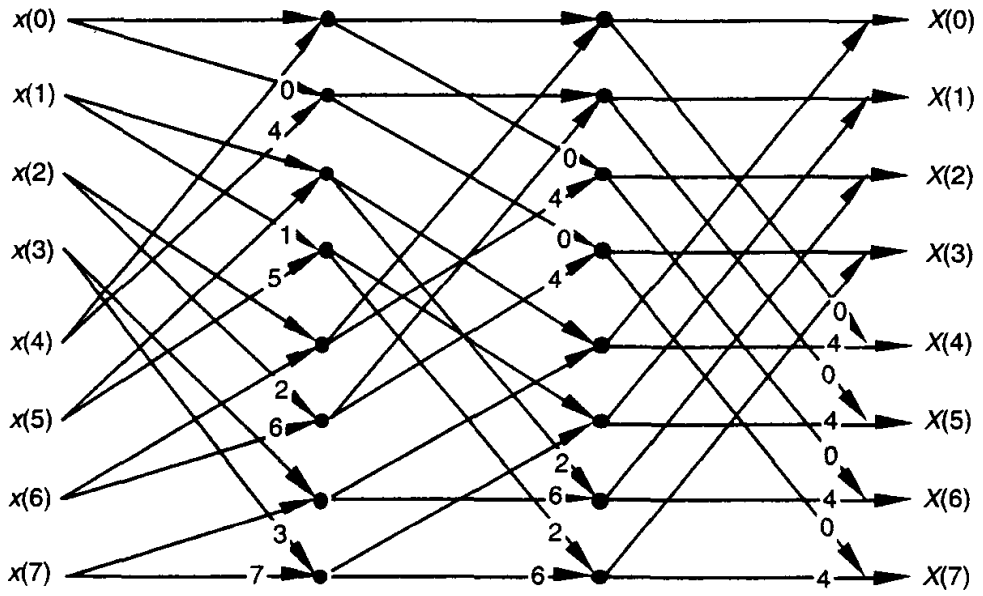
Rys. 4.11. 8-punktowa FFT z podziałem w częstotliwości, z odwróconymi bitowo wartościami wejściowymi



Rys. 4.12. 8-punktowa FFT z podziałem w częstotliwości, z odwróconymi bitowo wartościami wyjściowymi

mają zawsze odwrócone bitowo dane wejściowe, algorytmy FFT z podziałem w częstotliwości mają zaś zawsze odwrócone bitowo wartości wyjściowe. Nie jest to prawdą, jak pokazano na powyższych rysunkach. Podział w czasie lub częstotliwości jest określony przez to, czy dzielone są wartości wejściowe, czy też wartości wyjściowe DFT, kiedy wyprowadza się z równań DFT daną strukturę motylkową FFT.

Przypatrzmy się raz jeszcze pojedynczemu motylkowi. Struktury motylkowe FFT na rys. 4.8, 4.9, 4.11 i 4.12 stanowią bezpośrednie wyniki wyprowadzenia algorytmów z podziałem w czasie i z podziałem w częstotliwości. Wprowadzie na początku nie jest to całkiem oczywiste, ale wykładniki współczynników obrotu, pokazane w tych strukturach, nie mają spójnego wzorca. Zauważmy, że przyjmują



Rys. 4.13. 8-punktowa FFT z podziałem w częstotliwości, z wartościami wejściowymi i wyjściowymi o normalnym uporządkowaniu

one zawsze ogólne postaci pokazane na rys. 4.14(a)[4]. Aby zaimplementować motylek z podziałem w czasie z rys. 4.14(a), musielibyśmy przeprowadzić dwa mnożenia zespolone i dwa zespolone dodawania. Istnieje jednak lepszy sposób. Rozważmy motylek z podziałem w czasie z rys. 4.14(a). Jeśli górną wartość wejściową oznaczyć przez x, dolną zaś wartość wejściową przez y, to górna wartość wyjściowa wyrażałaby się jako

$$x' = x + W_N^k y \tag{4.26}$$

zaś dolna wartość wyjściowa motylka byłaby równa

$$y' = x + W_N^{k+N/2} y \tag{4.27}$$

Na szczęście operacje w równaniach (4.26) i (4.27) mogą być uproszczone, ponieważ te obydwa współczynniki obrotu są związane zależnością

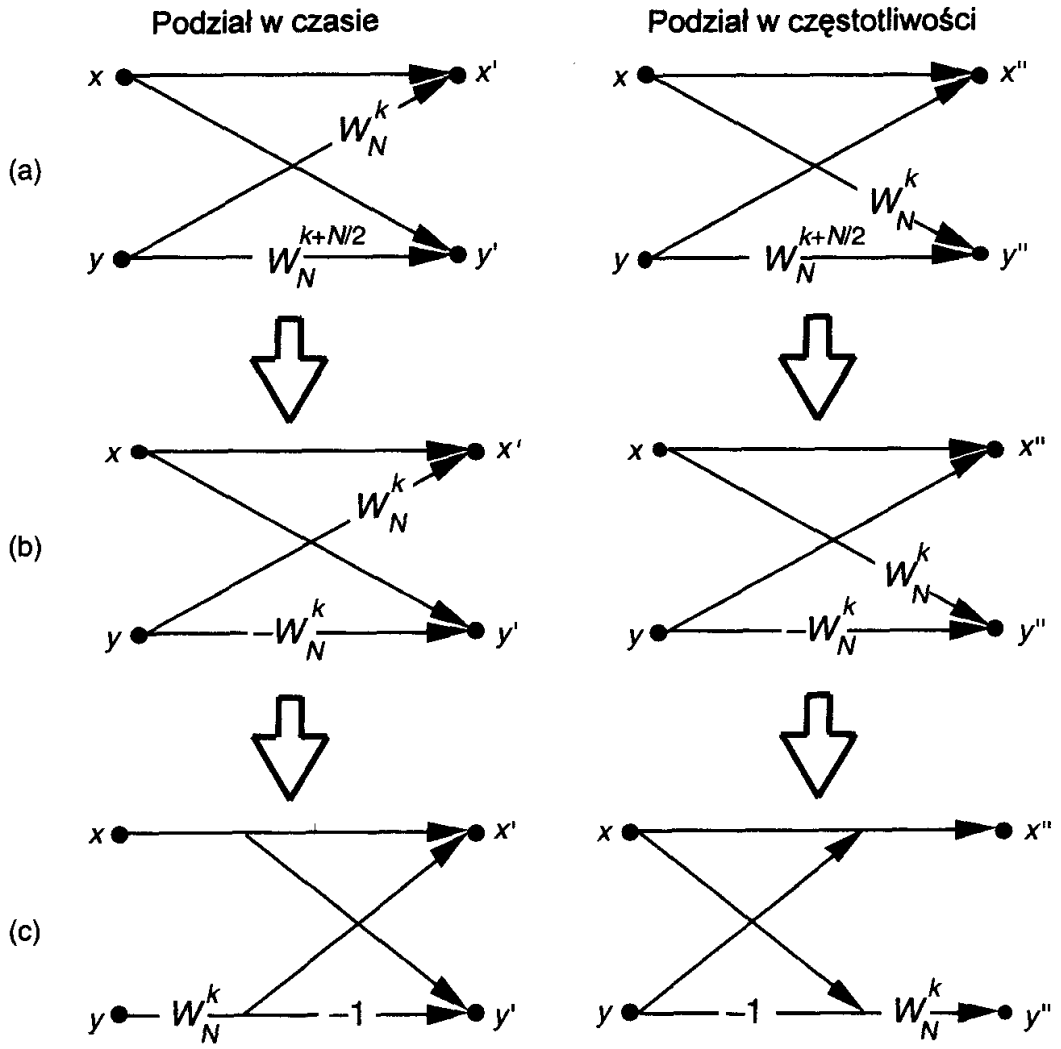
$$W_N^{k+N/2} = W_N^k W_N^{N/2} = W_N^k (e^{-j2\pi N/2N}) = W_N^k(-1) = -W_N^k \tag{4.28}$$

Zatem możemy zamienić współczynniki obrotu $W_N^{k+N/2}$ z rys. 4.14(a) na $-W_N^k$, co prowadzi do uproszczonych motylków pokazanych na rys. 4.14(b). Zauważmy, że współczynniki obrotu na rys. 4.14(b) różnią się jedynie znakami, zatem mogą być użyte zoptymalizowane motylki z rys. 4.14(c). Zauważmy, że te *zoptymalizowane* motylki wymagają dwóch zespolonych dodawań, ale tylko jednego mnożenia zespolonego, a przez to zmniejsza się nasz nakład obliczeniowy[5].

---

[4] Pamiętajmy, że – dla prostoty – struktury motylkowe na rys. od 4.8 do 4.13 pokazują jedynie wykładniki k i $k+N/2$ współczynników obrotu, nie zaś kompletne współczynniki obrotu.

[5] Jest tak dlatego, że w N-punktowym FFT jest $(N/2)\log_2 N$ motylków a, jak powiedzieliśmy, liczba mnożeń zespolonych, przeprowadzanych przez FFT, wynosi $(N/2)\log_2 N$ w równaniu (4.2).

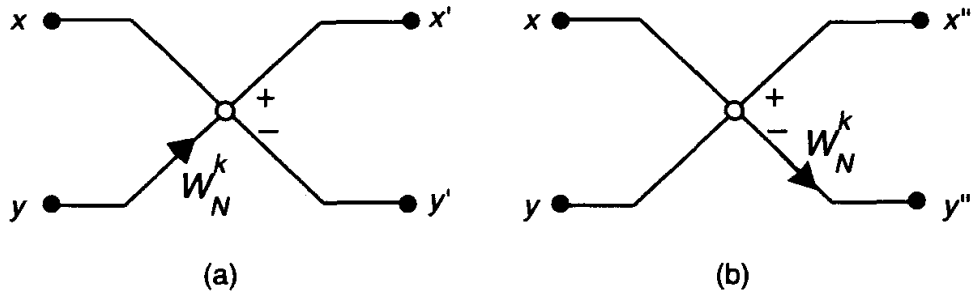


Rys. 4.14. Struktury motylkowe z podziałem w czasie i z podziałem w częstotliwości: (a) postać oryginalna; (b) postać uproszczona; (c) postać zoptymalizowana

Często w literaturze możemy napotkać zoptymalizowane struktury motylkowe z rys. 4.14(c), zamiast tych z rys. 4.14(a). Te zoptymalizowane motylki dają nam łatwy sposób rozpoznania algorytmów z podziałem w czasie i z podziałem w częstotliwości. Gdy dochodzimy do zoptymalizowanych struktur motylkowych z rys. 4.14(c), możemy stwierdzić, że algorytm dotyczy podziału w czasie, jeśli współczynnik obrotu poprzedza –1, w przeciwnym przypadku algorytm dotyczy podziału w częstotliwości, jeśli współczynnik obrotu następuje po -1.

Czasami w literaturze [5, 17] napotykamy struktury FFT, które wykorzystują notację pokazaną na rys. 4.15. Te motylki bez skrzydełek są równoważne strukturom z rys. 4.14(c). Konwencja przepływu sygnałów na rys. 4.15 jest taka, że dodatnia wartość wyjściowa kółka jest sumą dwóch próbek dochodzących do kółka z lewej strony, ujemna zaś wartość wyjściowa kółka jest różnicą próbek dochodzących do kółka. Zatem wartości wyjściowe struktur motylkowych z podziałem w czasie z rys. 4.14(c) i rys. 4.15(a) są dane poprzez wyrażenia

$$x' = x + W_N^k y \quad \text{i} \quad y' = -W_N^k y \tag{4.29}$$



Rys. 4.15. Alternatywne oznaczenie motylków FFT: (a) podział w czasie; (b) podział w częstotliwości

Wartości wyjściowe struktur motylkowych z podziałem w częstotliwości z rys. 4.14(c) i rys. 4.15(b) wyrażają się jako

$$x'' = x+y \quad \text{i} \quad y'' = W_N^k(x-y) = W_N^k x - W_N^k y \tag{4.30}$$

A więc których struktur FFT najlepiej jest używać? Zależy to od zastosowania, implementacji sprzętowej i wygody. Jeśli używamy implementacji programowej algorytmu FFT wykorzystując komputer ogólnego przeznaczenia, zwykle nie mamy zbyt bogatego wyboru. Większość ludzi używa jakichkolwiek istniejących podprogramów FFT, jakie zdarzyło im się znaleźć w wykorzystywanych przez nich komercyjnych pakietach oprogramowania. Ich kod może być zoptymalizowany ze względu na szybkość, ale tego nikt nie wie. Przebadanie kodu oprogramowania może okazać się niezbędne do stwierdzenia, jak procedura FFT jest zaimplementowana. Jeśli *odczuwamy potrzebę szybkości*, powinniśmy sprawdzić, czy oprogramowanie oblicza sinusy i cosinusy za każdym razem, kiedy wymagany jest współczynnik obrotu. Obliczenia trygonometryczne zazwyczaj zajmują wiele cykli maszynowych. Wstępne wyliczenie współczynników obrotu i przechowanie ich w tablicy może umożliwić przyspieszenie działania algorytmu. Dzięki temu ich wartości mogą być *odczytywane*, zamiast obliczane za każdym razem, kiedy są potrzebne w motylku. Jeśli piszemy własny podprogram, sprawdzenie ze względu na przepełnienie danych wyjściowych motylka i staranne skalowanie amplitudy może umożliwić wyznaczanie FFT za pomocą arytmetyki stałoprzecinkowej, która może być szybsza na niektórych maszynach[6]. Należy jednak być ostrożnym z używaniem arytmetyki stałoprzecinkowej; niektóre procesory typu Reduced Instruction Set Computers (RISC) w rzeczywistości dłużej przeprowadzają obliczenia stałoprzecinkowe, ponieważ są one szczególnie dedykowane do operowania na liczbach zmiennoprzecinkowych.

Jeśli używamy komercyjnego sprzętu wieloprocesorowego do naszych obliczeń, kod w tych procesorach jest *zawsze* optymalizowany, ponieważ ich życiowym celem jest szybkość. Producenci systemów wieloprocesorowych zazwyczaj re-

[6] Przepełnienie jest tym, co zdarza się, kiedy wynik operacji arytmetycznej ma zbyt wiele bitów, lub cyfr, aby mogły być reprezentowane w rejestrach sprzętowych, zaprojektowanych do zapamiętywania takiego wyniku. Przepełnienie danych FFT jest przedstawione w punkcie 9.3.

klamują swoje produkty przez specyfikowanie szybkości, z jaką ich maszyny wyznaczają 1024-punktowe FFT. Spójrzmy na niektóre opcje w wyborze szczególnej struktury FFT w przypadku, kiedy projektujemy sprzęt specjalnego przeznaczenia, aby zaimplementować algorytm FFT.

Poprzednio przedyskutowane struktury motylkowe FFT zazwyczaj można zaliczyć do dwóch kategorii: algorytmy FFT w miejscu oraz algorytmy FFT z podwójną pamięcią. Algorytm w miejscu jest przedstawiony na rys. 4.5. Wartości wyjściowe operacji motylkowej mogą być przechowane w tym samym miejscu pamięci sprzętowej, w którym poprzednio były przechowywane dane wejściowe operacji motylkowej. Nie jest tutaj potrzebna żadna pamięć pośrednia. W ten sposób N-punktowej FFT, potrzebne jest jedynie $2N$ adresów pamięci. (Mnożnik 2 wynika z faktu, iż każdy węzeł motylka reprezentuje wartość danej, która ma zarówno część rzeczywistą, jak i część urojoną.) W przypadku algorytmów w miejscu kierowanie danymi i adresacja pamięci jest raczej skomplikowana. Strukturę algorytmu FFT z podwójną pamięcią przedstawiono na rys. 4.10. W przypadku tej struktury konieczna jest pamięć pośrednia, ponieważ nie mamy już tutaj standardowych motylków i są potrzebne $4N$ miejsca w pamięci. Jednakowoż, kierowanie danymi i sterowanie adresacją pamięci jest znacznie prostsze w strukturach FFT z podwójną pamięcią niż w technice w miejscu. Użycie zmiennoprzecinkowych układów scalonych o wysokiej szybkości celem implementacji potokowych architektur FFT ma większą przewagę tych struktur potokowych, gdy jest używany algorytm z podwójną pamięcią [18].

Istnieje jeszcze inna klasa struktur FFT, znana jako algorytmy stałej geometrii, które czynią adresację pamięci zarówno prostą, jak też stałą dla każdego kroku w algorytmie FFT. Takie struktury są przedmiotem zainteresowania tych, którzy budują urządzenia sprzętowe FFT specjalnego przeznaczenia [4, 19]. Z punktu widzenia sprzętu ogólnego przeznaczenia, algorytmy z podziałem w czasie są optymalne dla rzeczywistych ciągów danych wejściowych, podziałem w częstotliwości jest zaś właściwy, kiedy wartości wejściowe są zespolone [8]. Jeśli dane wejściowe FFT są symetryczne w czasie, istnieją specjalne struktury FFT w celu eliminacji niekoniecznych obliczeń. Takie specjalne struktury motylkowe, oparte na symetrii danych wejściowych, są przedstawione w literaturze [20].

Dla dwuwymiarowych zastosowań FFT, takich jak przetwarzanie obrazów fotograficznych, algorytmy z podziałem w częstotliwości okazują się wyborem optymalnym [21]. Wasze zastosowania mogą być takie, że odwracanie bitowe danych wejściowych i wyjściowych nie jest ważnym czynnikiem. Niektóre zastosowania FFT pozwalają na operowanie ciągiem danych wyjściowych FFT odwróconych bitowo w dziedzinie częstotliwości bez konieczności przywracania naturalnego uporządkowania tych danych wyjściowych FFT. Wówczas transformacja odwrotna, która oczekuje wartości wejściowych odwróconych bitowo da wynik wyjściowy w dziedznie czasu, w którym ciąg danych jest poprawny. W tej sytuacji unika się potrzeby przeprowadzania jakiegokolwiek odwracania bitowego w ogóle.

Przykładami takiej możliwości jest mnożenie dwóch wyników wyjściowych FFT celem implementacji splotu lub korelacji [7]. Jak więc możemy zauważyć, znalezienie optymalnego algorytmu FFT i architektury sprzętowej dla FFT jest całkiem złożonym zagadnieniem do rozwiązania. Pozycje literatury [4, 22, 23] stanowią tutaj przewodnik.

Literatura do rozdziału 4

[1] J. Cooley and J. Tuckey. An algorithm for the machine computation of complex Fourier series. *Math. Comput.*, 19(90):297–301, April 1965.

[2] J. Cooley, P. Lewis, and P. Welch. Historical notes on the Fast Fourier Transform. *IEEE Trans. on Audio and Electroacoustics*, AU-15(2), June 1967.

[3] F. J. Harris. On the use of windows for harmonic analysis with the Discrete Fourier Transform. *Proceedings of the IEEE*, 66(1):54, January 1978.

[4] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[5] L. R. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[6] S. Stearns. *Digital Signal Analysis*. Hayden Book Co., Rochelle Park, New Jersey, 1975.

[7] *Programs for Digital Signal Processing*. *IEEE* Press, New York, 1979.

[8] H. V. Sorenson, D. L. Jones, M. T. Heideman, and C. S. Burrus. Real-valued Fast Fourier Transform algorithms. *IEEE Trans. on Acoust. Speech, and Singal Processing*, ASSP-35(6), June 1987.

[9] R. Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill, New York, 2nd revised edition, 1986.

[10] F. Cobb. Use of Fast Fourier Transform programs to simplify, enhance filter anylysis. *EDN*, 8 March 1984.

[11] F. Carlin. Ada and generic FFT routines tailored to your needs. *EDN* , 23, 1992.

[12] D. Evans. An improved digit-reversal permutation algorithms for the Fast Fourier and Hartley Transforms. *IEEE Trans. on Acoust. Speech, and Signal Proc.*, ASSP-35(8), August 1987.

[13] C. S. Burris. Unscrambling for fast DFT alboritms. *IEE Trans. on Acoust. Speech, and Signal Proc.*, 36(7), July 1988.

[14] J. J. Rodriquez. An improved FFT digit-reversal algorithm. *IEEE Trans. on Acoust. Speech, and Signal Proc.*, ASSP-37(8), August 1989.

[15] A. Land. Bit reverser scrambles data for FFT. *EDN*, 2, March 1995.

[16] JG-AE Subcommittee on Measurement Concepts. What the Fast Fourier Transform? *IEEE Trans. on Audio and Electroacoustisc*, AU-15(2), June 1967.

[17] R. Cohen and R. Perlman. 500 kHz single-board FFT system incorporates DSP-optimized chips. *EDN*, 31 October 1984.

[18] J. Eldon and G. E. Winter. Floating-point chips carve out FFT systems. *Electronic Design*, 4 August 1983.

[7] Patrz punkt 10.10, w którym zamieszczono przykład użycia FFT do wyznaczenia splotu.

[19] K. Lamb. CMOS building blocks shrink and speed up FFT systems. *Electronic Desig* 6 August 1987.

[20] J. D. Markel. FFT pruning. *IEEE Trans. on Audio and Electroacoustisc*, AU-19(4), December 197

[21] H. R. Wu and F. J. Paoloni. The structure of vector radix Fast Fourier Transforms. *IEEE Tran on Acoust. Soeech, and Signal Proc.*, ASSP-37(8), August 1989.

[22] Z. M. Ali. High speed FFT processor. *IEEE Trans. on Communications*, COM-26(5), May 197

[23] G. Bergland. Fast Fourier Transform hardware implementations – an overwiev. *IEEE Trans. o Acoust. Speech, and Signal Proc.*, AU-17, June 1969.