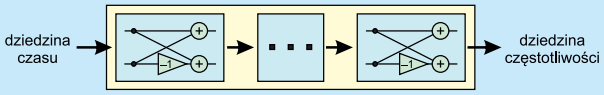
Eksperymenty z FPGA (15)

FFT przepływowa i iteracyjna

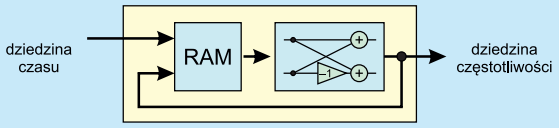
W kilku ostatnich odcinkach zaimplementowaliśmy szybką transformatę Fouriera. Była ona zbudowana w formie „przepływowej" – każdy kolejny krok został zrealizowany w osobnym bloku logiki. Takie podejście zwiększa przepustowość. Jednak gdy dane spływają wolniej, takie podejście prowadzi do marnowania dostępnych zasobów. Podejdziemy więc do problemu inaczej i zobaczymy, jak możemy wykorzystać pojedynczą logikę, kilkukrotnie tworząc coś, co przy programowaniu nazwalibyśmy pętlą.

Nasza poprzednia implementacja FFT została schematycznie pokazana na **rysunku 1**. Do modułu napływają próbki w dziedzinie częstotliwości. Przepływają przez kolejne motylki, które są niemal identyczne. Różnią się długością opóźnienia oraz liczbą współczynników *twiddle factor*. Można jednak przyjąć, że wykonują one tę samą funkcję. Ma to nawet odzwierciedlenie w naszym kodzie, gdzie zostały wstawione za pomocą pętli `for` w bloku `generate`.



Rysunek 1. Przepływowa implementacja FFT

Możemy więc pójść za tą analogią i stworzyć pętlę, która nie zostanie rozwinięta do osobnych bloków sprzętowych, lecz wykorzysta tylko jeden fragment logiki. Takie podejście zostało zaprezentowane na **rysunku 2**. Nowym elementem jest tu pamięć RAM. Posłuży ona do wstępnego zebrania danych, a następnie do przechowywania wyników obliczeń cząstkowych. Potrzebna będzie jeszcze dodatkowa logika sterująca, która umożliwi odczytywanie i zapisywanie danych w odpowiednich miejscach pamięci.



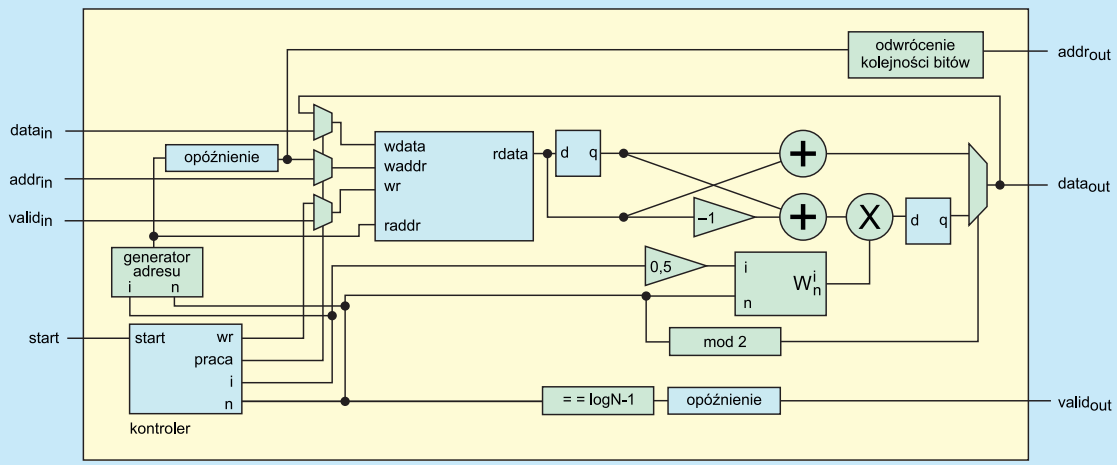
Rysunek 2. Iteracyjna implementacja FFT

Spróbujmy więc (na początku dość zgrubnie) wyobrazić sobie, jak będzie działał taki moduł. Jego uproszczony schemat został pokazany na **rysunku 3**. Mamy cztery wejścia. Pierwsze trzy dotyczą danych wejściowych. Są to kolejno: sama próbka, jej adres oraz sygnał *valid*. Czwarte wejście to sygnał *start*, który rozpocznie przetwarzanie.

Dane wejściowe trafiają na multipleksery. Jeżeli moduł jest w stanie spoczynku, zostaną one zapisane pod wskazany adres w pamięci RAM. Na razie zakładamy, że jest to pamięć idealna: po wpisaniu adresu wynik natychmiast pojawia się na wyjściu. W rzeczywistości wymaga to najczęściej kilku cykli zegara. Jednak na razie nie będziemy się tym przejmować. Tak jak poprzednio przez N oznaczymy długość transformaty. Łatwo więc zaobserwować, że będziemy potrzebować pamięci, która przechowa N zespolonych próbek. Gdy zostanie ona wypełniona danymi, blok FFT zostanie o tym poinformowany poprzez wejście start.

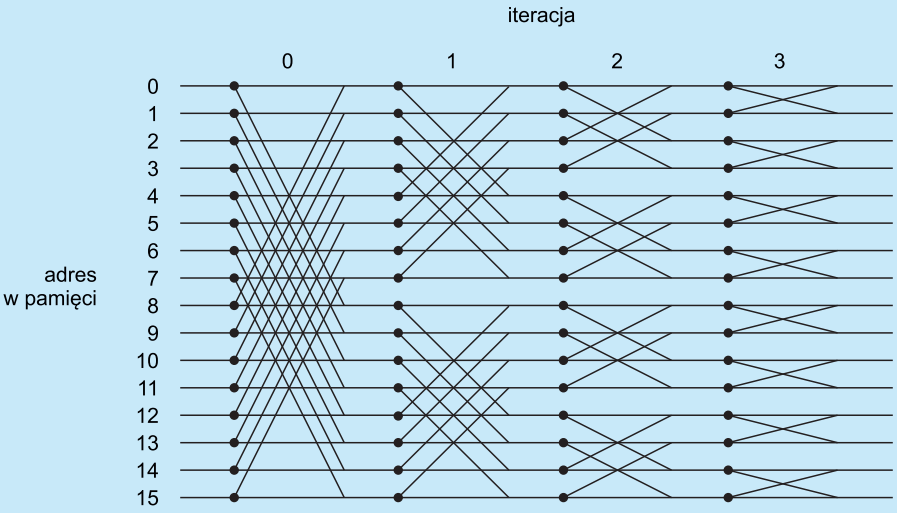
Na początku, żeby nie wchodzić w za dużo detali, blok, który zarządza pracą, nazwiemy kontroler i nie będziemy wnikać w jego implementację. Na razie ważne jest dla nas, że zapewni nam odpowiedni wybór źródła danych i zapis do pamięci. Zawiera on także dwa resetowane przy starcie liczniki. Pierwszy z nich, oznaczony jako i, będzie zliczał przebiegi wewnętrznej pętli. Przyjmuje on wartości od 0 do N. Drugi licznik odpowiada za obiegi zewnętrznej pętli. Oznaczymy go n. Będzie on przebiegał od 0 do log N.

Wyjście z pamięci trafia bezpośrednio na motylka. Jednak tym razem na wejściu i wyjściu ma on tylko po jednym przerzutniku. Jak pamiętamy, poprzednio długość tych opóźnień była zależna od kroku transformaty. Moglibyśmy próbować zrobić podobnie, dodając blok opóźnienia o zmiennej długości sterowanej przez licznik n. Jednak możemy być sprytniejsi i zaoszczędzić dość dużo przerzutników kosztem skomplikowania generatora adresu. Tym razem nie działamy przepływowo, więc wszystkie dane są dostępne od razu. Dlatego możemy po prostu tak dobierać kolejne adresy, żeby na wyjściu z pamięci zawsze mieć dwie kolejne próbki z pojedynczego motylka.



Rysunek 3. Schemat blokowy iteracyjnego FFT

Drugi blok, który musimy zmodyfikować, to generator współczynników *twiddle factor*. Tym razem będzie on musiał przyjmować oba współczynniki, zarówno górny, jak i dolny. Także tu zastosujemy metodę tablicowania. Ale jak się okaże, uzyskana tablica będzie identyczna jak w pierwszym motylku z przepływowego FFT. Po dokonaniu obliczeń dane trafiają z powrotem pod ten sam adres w pamięci, z którego zostały odczytane. Trafiają też na wyjście, jednak sygnał *valid* pojawia się tylko przy ostatnim obiegu pętli. Odpowiada za to porównanie wartości *n* z log *N–1*.



Rysunek 4. Kolejność odczytów z pamięci dla FFT z N=4

Generowanie adresu

Zastanówmy się teraz nad generowaniem kolejnych adresów, spod których musimy odczytywać dane. Pomoże nam w tym **rysunek 4**. Odczytamy z niego, jakie dane są potrzebne do kolejnych obliczeń. Każdy pionowy wiersz odpowiada komórce pamięci. Załóżmy, że na początku dane do pamięci będziemy wpisywać po kolei: próbkę z chwili 0 pod adresem zero, z chwili 1 pod 1 i tak aż do *N–1*. Rozważymy przypadek *N=16*.

Zaczniemy od zerowej iteracji (*n=0*). Pierwszy motylek wykonuje operacje na próbce 0 i 8. Uzyskane wyniki zapisuje z powrotem pod te same adresy. Kolejny motylek wykorzystuje dane spod 1 i 9 i tak aż do 7 i 15. Wszystkie kolejne wartości widzimy w pierwszym wierszu (**tabela 1**).

Tabela 1. Adresy odpowiadające kolejnym wartościom *i*, *n*

n\i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	8	1	9	2	10	3	11	4	12	5	13	6	14	7	15
1	0	4	1	5	2	6	3	7	8	12	9	13	10	14	11	15
2	0	2	1	3	4	6	5	7	8	10	9	11	12	14	13	15
3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Przejdźmy do *n=1*. Tutaj pierwszy motylek odczytuje dane spod adresów 0 i 4, następnie 1 i 5 i aż do 3 i 7. Widzimy, że teraz adresy z „dolnej połowy" nie mieszają się już z adresami z „górnej połowy". Kolejna para to 8, 12 aż do 11, 15. W iteracji *n=2* mamy już cztery grupy motylków, a na samym końcu adresy odczytujemy po kolei od 0 do 15.

Na pierwszy rzut oka kolejne wartości wydają się dość przypadkowe. Ale popatrzmy na **tabelę 2**, gdzie zostały zapisane w systemie dwójkowym. Gdy się im przyjrzymy, zauważymy, że można opracować łatwy przepis na otrzymanie adresu na podstawie współczynników *n*, *i*. Otóż wystarczy przesunąć najmłodszy bit na pozycję *N–1–n*. Został on zaznaczony czerwoną czcionką. Dokładnie tę ideę pokazuje **tabela 3**. Tutaj oznaczenie [], podobnie jak w SystemVerilogu, oznacza numer bitu.

Tabela 2. Fragment tabeli 1 w systemie dwójkowym

n\i	0000	0001	0010	0011	0100	0101
0	0000	1000	0001	1001	0010	1010
1	0000	0100	0001	0101	0010	0110
2	0000	0010	0001	0011	0100	0110
3	0000	0001	0010	0011	0100	0101

Tabela 3. Generowanie adresu na podstawie liczb *n*, *i*

n	adres			
0	i[0]	i[3]	i[2]	i[1]
1	i[3]	i[0]	i[2]	i[1]
2	i[3]	i[2]	i[0]	i[1]
3	i[3]	i[2]	i[1]	i[0]

Twiddle factor

Najpierw przypomnijmy znany nam już wzór na kolejne współczynniki:

$$W_n^i = e^{-j2\pi\frac{i}{n}} \quad (1)$$

Widzimy więc, że możemy skracać *i* oraz *n* jak zwykły ułamek. Jak pamiętamy, dla zerowej iteracji potrzebujemy współczynników od W_{16}^0 do W_{16}^7, natomiast w kolejnej już od W_8^0 do W_8^3. Jednak korzystając z wzoru (1), możemy wszystkie współczynniki sprowadzić do tych z *n=16*. Okazuje się, że kolejny krok używa podzbioru współczynników z poprzedniego, ale wykorzystuje tylko parzyste współczynniki. Dzięki temu możemy użyć tylko jednej, wspólnej tablicy.

Tabela 4. Współczynniki *twiddle factor* w zależności od *i*, *n*

n\i%2	0	1	2	3	4	5	6	7
0	W_{16}^0	W_{16}^1	W_{16}^2	W_{16}^3	W_{16}^4	W_{16}^5	W_{16}^6	W_{16}^7
1	W_{16}^0	W_{16}^2	W_{16}^4	W_{16}^6	W_{16}^0	W_{16}^2	W_{16}^4	W_{16}^6
2	W_{16}^0	W_{16}^4	W_{16}^0	W_{16}^4	W_{16}^0	W_{16}^4	W_{16}^0	W_{16}^4
3	W_{16}^0	W_{16}^0	W_{16}^0	W_{16}^0	W_{16}^0	W_{16}^0	W_{16}^0	W_{16}^0

Wróćmy do naszego szczególnego przypadku *N=16*. W **tabeli 4** pokazano, jakie wartości są potrzebne dla kolejnych współczynników *n*, *i*. Ponieważ każdy motylek potrzebuje dwóch próbek, do wyboru współczynniki, nie używamy najmłodszego bitu zmiennej *i*. Dlatego w tabeli pojawia się *i* modulo 2. Po chwili analizy możemy zapisać potrzebny adres jako:

$$i[\log N - 1 : 1] << n$$

Czyli skreślamy najmłodszy bit współczynnika *i*, a następnie uzyskany wynik przesuwamy o *n* pozycji w prawo.

Podsumowanie

W tym odcinku dowiedzieliśmy się, jak za pomocą pamięci RAM można implementować pętle w układzie FPGA. Przygotowaliśmy także podstawowe bloki iteracyjnego FFT. W następnym odcinku dowiemy się więcej na temat dostępnej w układzie Intel MAX10 blokowej pamięci RAM oraz przetestujemy nową wersję procesora FFT.

Rafał Kozik
rafkozik@gmail.com