Dominik RZEPKA, Ernest JAMRO, Kazimierz WIATR
KATEDRA ELEKTRONIKI, AKADEMIA GÓRNICZO-HUTNICZA
ACK CYFRONET, AKADEMIA GÓRNICZO-HUTNICZA

Implementacja szybkiej transformacji Fouriera o parametryzowanym rozmiarze w układach FPGA

Dominik RZEPKA

Student V roku Elektroniki i stażysta w Katedrze Elektroniki na Wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki AGH. Jego zainteresowania obejmują radiowe techniki transmisji danych, przetwarzanie sygnałów cyfrowych i sprzętową implementację algorytmów.

e-mail: dominik.rzepka@gmail.com

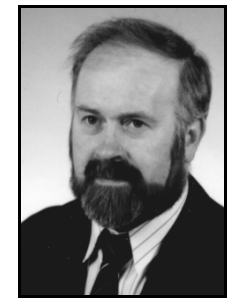
Dr inż. Ernest JAMRO

Ukończył studia na AGH na kierunku Elektronika oraz na University of Huddersfield (UK) na kierunku Elektronika i Telekomunikacja. Obronił pracę doktorską w 2001 roku na AGH na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki. Aktualnie jest adiunktem w Katedrze Elektroniki na AGH. Jego zainteresowania naukowe to sprzętowa akceleracja obliczeń, niskopoziomowe przetwarzanie obrazów, sieci neuronowe.

e-mail: jamro@agh.edu.pl

Prof. dr hab. inż. Kazimierz WIATR

Studia AGH Kraków (1980), dr nauk technicznych (1987), dr habilitowany (1999) i profesor (2002). Profesor zwyczajny na AGH w Krakowie oraz Dyrektor Akademickiego Centrum Komputerowego Cyfronet AGH. Prowadzone prace badawcze dotyczą systemów wizyjnych, systemów wieloprocesorowych, rekonfigurowanych systemów obliczeniowych i sprzętowych metod akceleracji obliczeń. Jest autorem trzech monografii, w tym najnowsza Akceleracja obliczeń w systemach wizyjnych wydana przez WNT w roku 2003.



e-mail: wiatr@agh.edu.pl

Streszczenie

W artykule przedstawiono przykład implementacji szybkiej transformacji Fouriera w układach FPGA. Operacja obrotu liczby zespolonej o dany kąt wykonywana podczas obliczeń FFT jest realizowana za pomocą modułu CORDIC. Dokonano analizy błędów zaokrągleń dla algorytmu CORDIC i mnożenia zespolonego, wykorzystywanych przy rotacji wektorów zespolonych. Główną motywacją niniejszej implementacji było współdzielenie zasobów pamięci BRAM pomiędzy różne zadania (nie tylko FFT) w ramach całego systemu zbudowanego w pakiecie EDK firmy Xilinx.

Słowa kluczowe: FFT, FPGA, CORDIC.

Implementation of fast Fourier transform of configurable size in FPGA circuits

Abstract

The paper presents hardware implementation of the Fast Fourier Transform (FFT) implemented in FPGAs. The FFT module is based on the CORDIC [4], therefore there is no need to store sin(α) coefficients. The main idea besides designing this FFT module was to share FPGA internal memory resources between different modules, e.g. FFT, Procedure of Linear Decimation [8]. This is a very important issue as FFT operation is one of many computation tasks performed by the embedded system [8], and internal memory resources are critical. Apart from it, for large FFT size (2^{16}), the external memory must be used. Therefore a special control and address counters were designed in order to allow internal and external memory transfers. The proposed FFT module calculates one butterfly operation per clock cycle (assuming internal memory transfers), therefore it is not speed optimized, nevertheless it is still much quicker than only MicroBlaze based implementation and it satisfies the system requirements. This paper presents also the computation error analysis.

Keywords: FFT, FPGA, CORDIC.

1. Wstęp

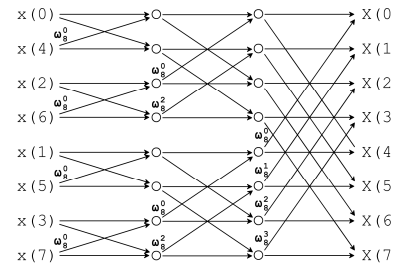
Istnieje wiele różnych implementacji szybkiej transformacji Fouriera (FFT) w układach FPGA np. [5, 6]. Ich podstawową wadą są duże wymagania odnośnie zajmowanej pamięci BRAM - wewnętrznej pamięci blokowej układu FPGA. Podstawowym założeniem niniejszej publikacji było zaprojektowanie sprzętowego wspomagania FFT w systemie osadzonym opartym na układzie FPGA [8], dla którego obliczanie FFT jest tylko jednym z wielu zadań. Głównym ograniczeniem w systemach osadzonych są z reguły niewystarczające zasoby pamięci BRAM. Zasoby te są zużywane np. na pamięć programu i danych procesora. Użycie modułu [5] wymaga aby wszystkie dane wejściowe i wyjściowe podczas wykonywania operacji FFT były w pamięci wewnętrznej, co powoduje użycie dużej liczby pamięci BRAM. Niestety pamięć ta nie jest dostępna dla innych zadań w momencie kiedy moduł FFT nie jest wykorzystywany. Po drugie, przy dużym rozmiarze FFT rzędu 2^{14} – 2^{16} wykorzystywanym w [8] dostępna pamięć BRAM dla układu XC3S1500 staje się niewystarczająca nawet do celów samego FFT. Dlatego głównym założeniem niniejszej pracy była możliwość stworzenia własnego modułu FFT który umożliwiałby współdzielenie zasobów pamięci BRAM pomiędzy różne moduły sprzętowe. Dodatkowym założeniem była możliwość korzystania z pamięci zewnętrznej w przypadku obliczania FFT dla dużej liczby punktów.

2. Szybka transformacja Fouriera

N–punktową dyskretną transformację Fouriera (DFT) można opisać jako sumę zespolonych próbek poddanych rotacji:

$$X(m)=\sum_{n=0}^{N-1}x(n)\exp\left(-j\frac{2\pi nm}{N}\right) \tag{1}$$

Wykorzystując właściwości przekształcenia DFT opracowano algorytm szybkiej transformacji Fouriera (FFT), który umożliwia zmniejszenie złożoności obliczeniowej z $O(N^2)$ do $O(N\log_2 N)$ [1]. Rys. 1 przedstawia strukturę przekształceń FFT dla rozmiaru transformaty N = 8 i podstawie 2.



Rys. 1. Schemat przetwarzania FFT
Fig. 1. Block diagram of FFT

Przekształcenie składa się z serii operacji dodawania (odejmowania) dwóch liczb zespolonych powiązanym z obrotem jednej z nich o zadany kąt, nazywanych często operacjami motylkowymi:

$$\begin{aligned} S_A &= s_A + s_B \cdot \omega_N^\theta \\ S_B &= s_A - s_B \cdot \omega_N^\theta \end{aligned} \quad \text{gdzie} \quad \omega_N^\theta = \exp\left(-j\frac{2\pi\theta}{N}\right) \tag{2}$$

Najbardziej skomplikowaną obliczeniowo operacją jest obrócenie liczby zespolonej o zadany kąt, której poświęcony jest następny punkt niniejszego opracowania.

3. Mnożenie zespolone i algorytm CORDIC

Zmiana argumentu liczby zespolonej $\overline{A}$ jest często realizowana za pomocą mnożenia zespolonego:

$$\begin{aligned} \overline{A}\cdot e^{j\varphi} &= \left(\mathrm{Re}\left[\overline{A}\right] + j\,\mathrm{Im}\left[\overline{A}\right]\right)\cdot\left(\cos\varphi + j\sin\varphi\right) \\ \mathrm{Re}\left[\overline{A}\cdot e^{j\varphi}\right] &= \mathrm{Re}\left[\overline{A}\right]\cos\varphi - \mathrm{Im}\left[\overline{A}\right]\sin\varphi \\ \mathrm{Im}\left[\overline{A}\cdot e^{j\varphi}\right] &= \mathrm{Re}\left[\overline{A}\right]\sin\varphi + \mathrm{Im}\left[\overline{A}\right]\cos\varphi \end{aligned} \tag{3}$$

Z powyższej zależności wynika, że wymaga to zastosowania 4 mnożarek i 2 sumatorów. Istnieje również możliwość zoptymalizowania takiego działania tak, aby zredukować ilość mnożeń do 3, natomiast konieczne jest wtedy użycie 5 sumatorów [2].

Do wykonania rotacji potrzebne są wartości funkcji trygonometrycznych. Powszechnie stosuje się rozwiązanie polegające na przechowywaniu w pamięci stałej wartości sin(α) dla $\alpha \in [0, \pi/2]$. Na ich podstawie można łatwo otrzymać wartość sinusa i cosinusa z dowolnego przedziału.

Na podstawie minimalnej wartości kąta rotacji potrzebnej do przeprowadzenia transformacji FFT o rozmiarze N (2) można zauważyć, że konieczne jest posiadanie P próbek sinusa, gdzie:

$$P = \frac{N}{4} + 1 \tag{4}$$

Alternatywnym dla mnożenia zespolonego rozwiązaniem jest zastosowanie algorytmu CORDIC (ang. *COordinate Rotation DIgital Computer*) [3, 4]. Zastosowanie algorytmu CORDIC dla obliczania FFT zostało opisane np. [9]. Algorytm CORDIC wykorzystuje tożsamość matematyczną:

$$\begin{aligned} \mathrm{Re}\left[\overline{A}\cdot e^{j\varphi}\right] &= \cos\varphi\left(\mathrm{Re}\left[\overline{A}\right] - \mathrm{Im}\left[\overline{A}\right]\tan\varphi\right) \\ \mathrm{Im}\left[\overline{A}\cdot e^{j\varphi}\right] &= \cos\varphi\left(\mathrm{Im}\left[\overline{A}\right] + \mathrm{Re}\left[\overline{A}\right]\tan\varphi\right) \end{aligned} \tag{5}$$

Jeżeli ograniczy się możliwe rotacje do wartości takich, że $tg(\varphi) = \pm 2^{-m}$, to mnożenie przez tangens będzie można zastąpić przez operację przesunięcia bitowego. Aby osiągnąć rotację o dowolny kąt należy wykonać szereg rotacji spełniających warunek na $tg(\varphi) = \pm 2^{-m}$ tak, aby w każdej kolejnej sumaryczny kąt dążył do oczekiwanego kąta. Mnożenie przez $cos(\varphi)$, wykonywane w każdej z M iteracji można zastąpić przez jednokrotne mnożenie zmiennej zespolonej przez stałą, przed pierwszą lub po ostatniej iteracji, wykorzystując fakt, że:

$$\prod_{m=0}^{M-1}\cos(\arctan(\pm 2^{-m})) = const \tag{6}$$

Do kontrolowania sumarycznej rotacji potrzebna jest znajomość kątów odpowiadających warunkowi $tg(\varphi) = \pm 2^{-m}$. Aby sposób zapisu kąta sterującego był efektywny, powinien być on wyrażony w wielokrotnościach $2\cdot\pi/N$ (kąty wykorzystywane w FFT), oraz przeskalowany tak, aby można było w nim zapisać wykorzystywany zakres wartości $[-\pi/2, \pi/2]$.

Dokładność obliczeń w algorytmie CORDIC zależy między innymi od z liczby iteracji. Wraz ze zmniejszeniem liczby iteracji zmniejszamy zajmowane zasoby sprzętowe kosztem większego błędu obliczeń. Na podstawie symulacji w Matlabie otrzymano zależność przybliżoną:

$$\begin{aligned} &\max_{\alpha\in R}\left(\left|\sin(\alpha) - \sin(\alpha_{CORDIC})\right|\right) = 2^{-M+0{,}86} \\ &\text{gdzie } \alpha_{CORDIC} = \sum_{m=0}^{M-1}\arctan\left(\pm 2^{-m}\right) \end{aligned} \tag{7}$$

Z zależności (7) wynika, że aby otrzymać wymaganą dokładność M bitów, konieczne jest wykonanie nie mniej niż M+1 iteracji (po zaokrągleniu 0.86 do jedynki) algorytmu CORDIC.

Warto podkreślić, że aby osiągnąć wymaganą dokładność obliczeń algorytmu CORDIC zalecane jest dodanie 1 – 4 dodatkowych najmłodszych bitów, tzw. *guard*, tak aby zapobiec kumulacji błędów kolejnych obliczeń. Wyniki symulacyjne pokazały, że dodanie więcej niż jednego bitu *guard* nie wpływa znacząco na wynik końcowy całej transformacji FFT.

Ilość bitów w słowie danych wyjściowych FFT W_{OUT} (a więc także w każdym elemencie uczestniczącym w obliczaniu FFT) powinna być odpowiednio większa od ilości bitów W_{IN} na wejściu modułu FFT ze względu na możliwość wystąpienia przepełnień. Zależność między tymi wartościami może być łatwo wyprowadzona ze wzoru (1), poprzez podstawienie jako sygnału wejściowego składowej stałej o maksymalnej wartości:

$$W_{OUT} = W_{IN} + \log_2 N \tag{8}$$

4. Generacja adresów wewnętrznych i kątów rotacji

Koncepcja opisywanej implementacji FFT zakłada, że do wykonywania transformacji wykorzystywana jest jedna struktura *motylka*. Konieczne jest więc zastosowanie algorytmu, który będzie generował adresy danych wejściowych oraz kąt obrotu dla każdej operacji *motylka*. W każdym z (log_2N) etapów FFT trzeba przetworzyć $N/2$ par zmiennych zespolonych. Przyjęto założenie, że przetwarzane dane znajdujące się w pamięci BRAM zostały ułożone w kolejności charakterystycznej dla FFT z podziałem w czasie (rys. 1), czyli po adresowaniu rewersyjno-bitowym.

Ze względu na duży czas pobierania danych z pamięci zewnętrznej dla dużego rozmiaru FFT, możliwe jest przeprowadzenie optymalizacji polegającej na podzieleniu transformacji FFT na fragmenty, wymagające mniejszej ilości wykorzystywanych danych. Dzięki temu dane te można ulokować w szybszej pamięci BRAM. Jest tak dlatego, że po dokonaniu adresowania rewersyjno-bitowego, obliczenia na początkowych warstwach FFT nie zależą od rozmiaru FFT (zob. rys. 1). Dlatego dla dużych rozmiarów FFT możliwe jest przeprowadzanie wielu niezależnych obliczeń dla początkowych warstw na danych, które znajdują się w pamięci wewnętrznej BRAM, co zdecydowanie przyśpieszy dostęp do danych i przez to zwiększy szybkość obliczeń. Dopiero na poziomie warstw końcowych wymagane jest scalenie poszczególnych wyników pośrednich, i dopiero na tym etapie konieczne jest korzystanie z wolniejszej pamięci zewnętrznej.

Sterowanie generatorem adresów odbywa się za pośrednictwem licznika CT_{FFT}, którego $log_2(N/2)$ najmłodszych bitów stanowi zmienną CT_{PAIR} a kolejne $log_2(log_2N)$ bitów – zmienną CT_{STAGE}. Generacja adresów zmiennych zespolonych dla pary wejść A i B struktury motylkowej przeprowadzana jest według łatwego do implementacji wzoru:

$$\begin{aligned} ADDR_A &= CT_{PAIR} \bmod 2^{CT_{STAGE}} + \\ &\quad + CT_{PAIR}\cdot 2 - \left(CT_{PAIR}\cdot 2\right) \bmod 2^{CT_{STAGE}} \\ ADDR_B &= ADDR_A + 2^{CT_{STAGE}} \end{aligned} \tag{9}$$

Drugą część układu to generator kątów obrotu dla poszczególnych operacji motylkowych. Obliczanie kąta obrotu z zakresu $[-\pi, 0]$ w postaci dogodnej dla algorytmu CORDIC odbywa się według wzoru:

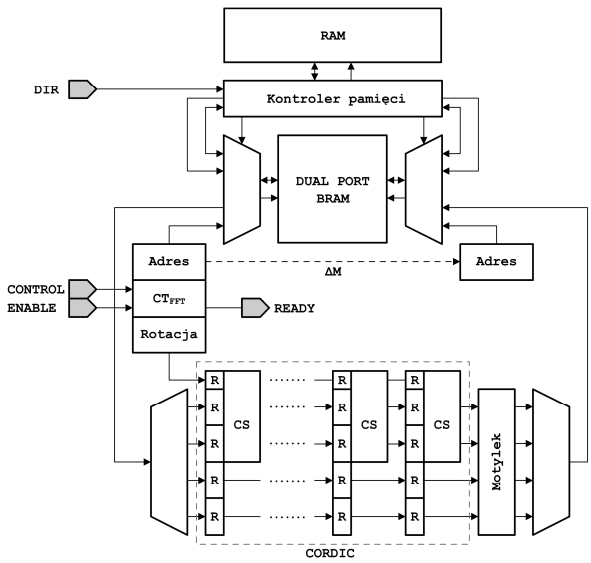
$$\alpha = \left(CT_{FFT} \bmod 2^{CT_{STAGE}} \right) \cdot 2^{\log_2 N - CT_{STAGE} + 2} \qquad (10)$$

W tym przypadku implementacja wymaga zastosowania układu barrel-shifter dla maksymalnej obsługiwanej szerokości danych $log_2(N/2)$.

Konfiguracja układu generatora sprowadza się do podania wartości początkowej licznika CT_{FFT}, rozmiaru transformacji N i opcjonalnie ograniczenia górnej wartości licznika, które jest przydatne przy obliczaniu transformat częściowych.

5. Implementacja

Schemat blokowy przedstawiający opisywaną implementację znajduje się na rys. 2. W pierwszym etapie kontroler pamięci przeprowadza transfer danych z adresowaniem rewersyjno-bitowym z pamięci zewnętrznej do wewnętrznej dwuportowej pamięci BRAM. Następnie transformacja jest sterowana poprzez generator adresów i kątów rotacji, który dostarcza dane do modułu CORDIC. Mimo, że algorytm CORDIC wymaga do przeprowadzenia obrotu danej zespolonej kilkunastu etapów, dzięki implementacji potokowej pojedyncza operacja motylkowa jest wykonywana co takt zegara. Operacja *motylka* wymaga odczytu dwóch danych wejściowych (zespolonych) i zapisu dwóch danych wyjściowych.



Rys. 2. Schemat blokowy opisywanej realizacji FFT
Fig. 2. Block diagram of the described FFT implementation

Tabela 1 przedstawia zasoby sprzętowe układu Spartan 3 XC3S1500 firmy Xilinx zajmowane przez wybrane moduły. Podczas implementacji założono, że dane wejściowe do tego modułu są 32-bitowe a liczba etapów wykonywania operacji CORDIC wynosi 17. W konsekwencji moduł taki może służyć do wykonywania transformacji FFT dla maksymalnego rozmiaru $N= 2^{16}$ oraz danych wejściowych 16-bitowych - należy pamiętać, że wymagana dokładność reprezentacji danych wzrasta wraz rozmiarem operacji FFT (8). Założono że operacja CORDIC wymaga tylko jednego dodatkowego bitu *guard*. Wyniki implementacji podano dla następujących modułów:

- CORDIC - moduł służący do obrotu danej zespolonej o zadany kąt.
- Motylek – moduł wykonujący operację według równania (2), składa się on z modułu CORDIC oraz z dodatkowych sumatorów.
- Licznik FFT – moduł służący do generacji adresów danych wejściowych i wyjściowych oraz generacji kątów obrotu.
- Licznik rewersyjno-bitowy – licznik ten jest potrzebny podczas odczytu danych z zewnętrznej (potrzebny do pobrania danych wejściowych dla pierwszej warstwy FFT.

Tab. 1. Wyniki implementacji
Tab. 1. Implementation results

Moduł	LUT	MULT18×18
CORDIC	1,759	8
Motylek	1,820	8
Licznik FFT	270	0
Licznik rewersyjno-bitowy	41	0

Analizując wyniki implementacji można zauważyć, że największe wymagania sprzętowe ma moduł CORDIC. Obliczenia są w nim dokonywane w 17 etapach, w każdym wykonywana jest operacja dodawania na dwóch liczbach 33-bitowych (32-bity + 1 *guard*), szacowane zasoby LUT takiej operacji są więc następujące: 17×2×33= 1122 (Spartan 3). Zasoby te można zmniejszyć poprzez zmniejszenie ilości etapów (co zmniejszy dokładność rotacji, a więc także maksymalny możliwy rozmiar FFT) lub też precyzji danych, np. poprzez skalowanie próbek po każdym etapie motylka. Dodatkowe zasoby są potrzebne między innymi na akumulację kąta. Mnożenie przez stałą (6) wykonano przy użyciu wbudowanych układów mnożących *mult18×18*.. W przypadku użycia układów FPGA nie posiadających takich zasobów, implementację mnożenia przez wartość stałą można zdecydowanie uprościć [7].

6. Wnioski

Zaprezentowany algorytm umożliwia obliczanie transformaty Fouriera o parametryzowanym rozmiarze. Zastosowanie algorytmu CORDIC eliminuje potrzebę wykorzystania pamięci do przechowywania współczynników $sin(\alpha)$, co ma szczególne znaczenia w przypadku dużego rozmiaru FFT. Dzięki wykorzystaniu zewnętrznego kontrolera pamięci, oraz programowanego generatora adresów wewnętrznych istnieje możliwość współdzielenia pamięci BRAM używanej przez moduł FFT również przez inne moduły, np. Procedurę Liniowej Decymacji [8]. Możliwe stało się również wykonywanie FFT dla dużych rozmiarów, dla których konieczne jest korzystanie z pamięci zewnętrznej.

Publikacja finansowana ze środków na naukę MNiSW w roku 2009.

7. Literatura

[1] Richard G. Lyons: Wprowadzenie do cyfrowego przetwarzania sygnałów, Wydawnictwa Komunikacji i Łączności, Warszawa 2000.
[2] Xilinx, Complex Multiplier v2.1, DS291 April 28, 2005.
[3] Ray Andraka: A survey of CORDIC algorithms for FPGA based computers, Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, p.191-200, February 22-25, 1998, Monterey, California, United States.
[4] Despain Alvin: Fourier transform computers using CORDIC iterations, IEEE Transactions on Computers, v.23 n.10, p.993-1001, October 1974.
[5] Xilinx, Fast Fourier Transform v. 3.2, Core Generator, DS260, 11 Jan 2006.
[6] Opencores.org CF_FFT, www.opencores.org, 8 May 2008.
[7] K. Wiatr, E. Jamro: Constant Coefficient Multiplication in FPGA Structures, Proc. of the IEEE Int. Conf. Euromicro, Maastricht, The Netherlands, Sep. 5-7, 2000, Vol. I, pp. 252-259.
[8] J. Adamczyk, P. Krzyworzeka, W. Cioch, E. Jamro: Monitoring of Nonstationary States in Rotating Machitery, WITE Państwowy Instytut Badawczy - Radom, Kraków 2006.
[9] B. Heyne, J. Gotze: A pure cordic based FFT for reconfigurable digital signal processing, EUSIPCO, Vienna , AUTRICHE 6-10 Sep. 2004.

Artykuł recenzowany