AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Systemy Dedykowane w Układach Programowalnych

Algorytm Cooleya-Tukeya – FFT

Autor: Jakub Lasak

Kierunek studiów: Elektronika i telekomunikacja

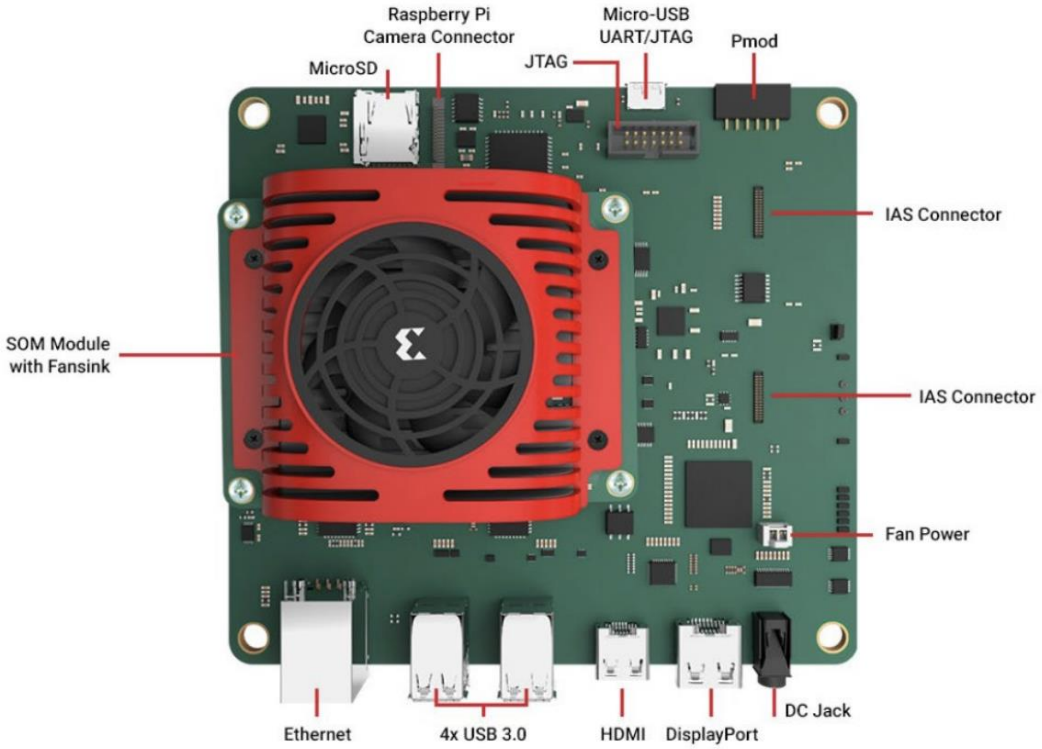
Kraków, 2025

1. Wstęp

Celem projektu było zaprojektowanie oraz implementacja algorytmu szybkiej transformacji Fouriera (FFT) w środowisku układów programowalnych. Transformata Fouriera jest jednym z kluczowych narzędzi analizy sygnałów, wykorzystywanym do przekształcenia sygnału z dziedziny czasu do dziedziny częstotliwości. Zastosowanie algorytmu FFT pozwala na znaczące przyspieszenie obliczeń w porównaniu z klasyczną dyskretną transformatą Fouriera (DFT), dzięki czemu metoda ta jest powszechnie używana w systemach cyfrowego przetwarzania sygnałów (DSP), komunikacji, obrazowaniu oraz wielu innych dziedzinach inżynierii.

W ramach projektu wykonano:

- Modelowanie i weryfikację algorytmu FFT w środowisku MATLAB – umożliwiło to przeprowadzenie symulacji oraz weryfikację poprawności wyników przed implementacją sprzętową.
- Projekt architektury w języku Verilog – moduły algorytmu FFT zostały zaimplementowane jako komponenty sprzętowe, zoptymalizowane pod kątem równoległego przetwarzania danych.
- Syntezę i symulację projektu w środowisku Xilinx Vivado 2018.3 – pozwoliło to na analizę funkcjonalną i czasową implementacji.
- Implementację na platformie sprzętowej AMD-Xilinx Kria KV260 Vision AI Starter Kit – umożliwiającej przetestowanie algorytmu w rzeczywistym układzie programowalnym FPGA.
- Projekt stanowi kompleksowy przykład przejścia od modelu matematycznego, poprzez projekt logiczny, aż do sprzętowej realizacji algorytmu FFT na układzie FPGA. Szczególny nacisk położono na modularność kodu, możliwość ponownego wykorzystania bloków oraz optymalizację zasobów sprzętowych.



Rys. 1 Płytka rozwojowa Kria KV260

2. Opis teoretyczny algorytmu

2.1. Transformacja Fouriera

Transformata Fouriera jest jednym z kluczowych narzędzi analizy sygnałów, umożliwiającym przejście z reprezentacji czasowej sygnału do reprezentacji częstotliwościowej. W przypadku sygnałów dyskretnych, próbkowanych w równych odstępach czasu, stosuje się Dyskretną Transformatę Fouriera (DFT). Dla sygnału x[n] o długości *N* próbek, DFT definiuje się wzorem:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}kn}, k = 0, 1, \dots, N-1$$

Gdzie:

$x[n]$ – próbka sygnału wejściowego w dziedzinie czasu,

$X[k]$ – współczynnik transformaty dla częstotliwości dyskretnej k,

N – liczba próbek sygnału,

j – jednostka urojona

DFT jest podstawowym narzędziem do analizy widma sygnału, jednak bezpośrednie obliczenie N^2 wymaga operacji mnożenia i dodawania, co przy dużych wartościach *N* jest obliczeniowo kosztowne. Z tego powodu w praktyce stosuje się Szybką Transformatę Fouriera (FFT) – zoptymalizowany algorytm obliczania DFT.

2.2. Szybka transformacja Fouriera – FFT

FFT (Fast Fourier Transform) to zoptymalizowany algorytm umożliwiający efektywne wyznaczanie współczynników DFT, redukujący złożoność obliczeniową z $O(N^2)$ do $O(N \log_2 N)$ Najczęściej stosowana jest metoda Cooley'a–Tukeya**,** która polega na rekurencyjnym dzieleniu obliczeń na mniejsze pod problemy i wykorzystywaniu symetrii oraz okresowości tzw. współczynników twiddle factors. Współczynniki twiddle factors definiuje się jako:

$$W_N^k = e^{-j\frac{2\pi}{N}kn}, k = 0, 1, \dots, N-1$$

Algorytm FFT działa etapowo, dzieląc obliczenia na kolejne etapy(stages), w których wykonywane są podstawowe operacje zwane butterfly. Każda operacja butterfly łączy dwie próbki sygnału i dokonuje transformacji:

$$X_{górny} = x_{górny} + W \cdot x_{dolny}$$

$$X_{dolny} = x_{górny} - W \cdot x_{dolny}$$

Gdzie:

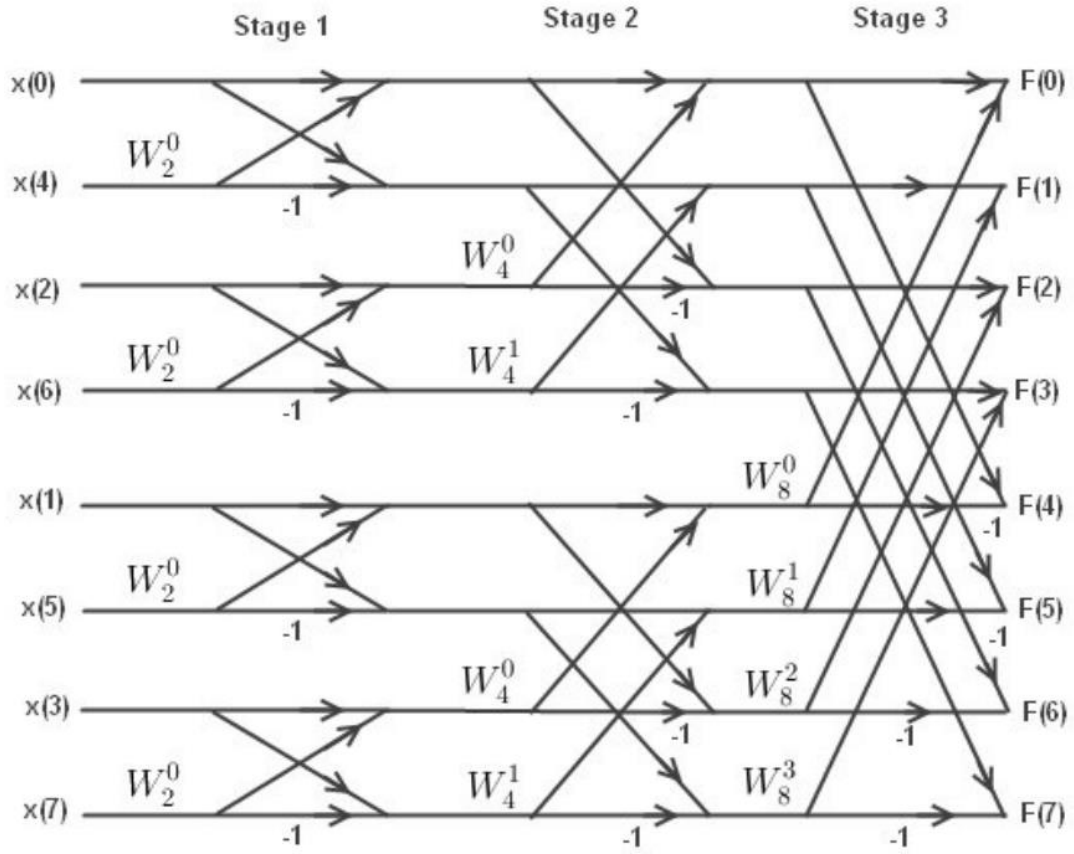
$x_{górny}$ i x_{dolny} to próbki wejściowe dla danego węzła,

W – odpowiedni współczynnik twiddle factor.

Dla sygnału o długości $N = 2^m$ algorytm składa się z $\mathrm{m} = \log_2 N$ etapów. Kolejne operacje wykonywane są na przestawionych próbkach wejściowych, uporządkowanych zgodnie z odwrotnością bitową indeksów (bit reversal), co umożliwia efektywne równoległe przetwarzanie danych w architekturze sprzętowej.

Dzięki tym właściwościom FFT stało się standardowym narzędziem w aplikacjach czasu rzeczywistego, takich jak:

- cyfrowe przetwarzanie sygnałów (DSP),
- systemy komunikacyjne,
- analiza i filtracja widma sygnałów,
- systemy radarowe, obrazowanie medyczne (MRI) i wiele innych.



Rys. 2 Diagram motylkowy dla FFT o długości 8

3. Model behawioralny

Prezentowany model behawioralny w MATLAB implementuje FFT dla sygnału o długości 8 próbek. Na wejściu generowany jest sygnał kosinusoidalny, którego próbki tworzą wektor input. Następnie obliczane są współczynniki twiddle $W_N^k = e^{-j\frac{2\pi}{N}kn}$, wykorzystywane w kolejnych etapach motylkowych (butterfly). Wektory wejściowe są przestawiane według odwrotności bitów indeksów (bitrevorder) w celu przygotowania danych do etapów FFT. Transformacja odbywa się iteracyjnie w etapach, w których każda para próbek jest przetwarzana zgodnie ze wzorem motylka: górna gałąź jest sumą oryginalnej wartości i dolnej gałęzi pomnożonej przez współczynnik twiddle, natomiast dolna gałąź jest różnicą oryginalnej górnej gałęzi i zaktualizowanej dolnej gałęzi. Na końcu model prezentuje widmo amplitudowe sygnału przy użyciu funkcji *abs*. Model ten pozwala na zrozumienie działania algorytmu FFT, w tym roli współczynników twiddle, permutacji bitowej i operacji butterfly, oraz umożliwia weryfikację wyników w dziedzinie częstotliwości.

4. Model strukturalny

Przedstawiony projekt implementuje szybką transformatę Fouriera dla ośmiu próbek sygnału w postaci strukturalnego modelu w języku Verilog. Głównym modułem systemu jest *fft8_top*, który integruje wszystkie etapy transformacji FFT, w tym przestawienie bitów (bitrev),

operacje motylkowe (butterfly) oraz mnożenie zespolone (cplx_mul) z wykorzystaniem współczynników twiddle przechowywanych w pamięci ROM (twiddle_rom). Moduł twiddle_rom zawiera stałe współczynniki zapisane w formacie Q1.15, pozwalające na wykonywanie mnożeń zespolonych w kolejnych etapach FFT. Moduł *bitrev* realizuje przestawienie próbek wejściowych zgodnie z odwrotnością bitów indeksów, co przygotowuje dane do sekwencyjnych etapów motylkowych, minimalizując konieczność przesunięć w trakcie obliczeń. Moduł butterfly wykonuje podstawową operację FFT dla par próbek: górna gałąź jest sumą wartości próbki górnej i dolnej po przemnożeniu przez odpowiedni współczynnik twiddle, natomiast dolna gałąź stanowi różnicę tych wartości, zapewniając prawidłowy przepływ informacji w drzewie motylkowym.

Moduł *fft8_top* implementuje trzy etapy transformacji, odpowiadające kolejnym poziomom diagramu motylkowego dla ośmiu próbek. Pierwszy etap obejmuje cztery motylki na parach sąsiednich próbek, korzystając z twiddle W_8^0. Drugi etap przetwarza pary oddalone o dwa miejsca, stosując współczynniki twiddle W_8^0 i W_8^2. Trzeci etap łączy pary odległe o cztery miejsca z odpowiednimi twiddle W_8^0, W_8^1, W_8^2, W_8^3 generując końcowe wyjścia FFT. W module testbench tb_fft8 definiowane są przykładowe dane wejściowe w formacie Q1.15 odpowiadające sygnałowi kosinusoidalnemu oraz wszystkie części zespolone równe zero, co pozwala na weryfikację poprawności działania układu. Wyniki symulacji są prezentowane zarówno w formie całkowitej, jak i przeskalowanej do wartości zmiennoprzecinkowej, umożliwiając łatwe porównanie z oczekiwanym przebiegiem częstotliwościowym.

5. Wyniki

Dane wejściowe to próbki sygnału:

$$x[n] = \cos\left(\frac{2\pi n}{8}\right), n = 0..7$$

Wartości wejściowe w formacie Q1.15 to:

1.0000, 0.7071, 0, -0.7071, -1.0, -0.7071, 0, 0.7071

Dla sygnału kosinusoidalnego $\cos\left(\frac{2\pi n}{8}\right), n = 0..7$:

- FFT powinna mieć niezerowe składowe w k=1 i k=7, bo $\cos\left(\frac{2\pi n}{8}\right) = \frac{1}{2}(e^{j\frac{2\pi}{8}n} + e^{-j\frac{2\pi}{8}n})$
- Pozostałe współczynniki (k=0,2,3,4,5,6) powinny być bliskie zeru

Skala Q1.15

- Dane wejściowe Q1.15: wartość maksymalna 32767 odpowiada 1.0.
- Skala wyników FFT: przy 8-punktowej FFT, suma wejść może dać maksymalnie 8*32767 ≈ 262136.
- Twoje wyjścia: k=1 → 131065 ≈ 32767 * 4 = 131068.

- To jest dokładnie połowa maksymalnej wartości, co jest poprawne, ponieważ $\cos\left(\frac{2\pi n}{8}\right)$ daje amplitudę 0.5 w k=1 po FFT (bo suma połówek wykładników).

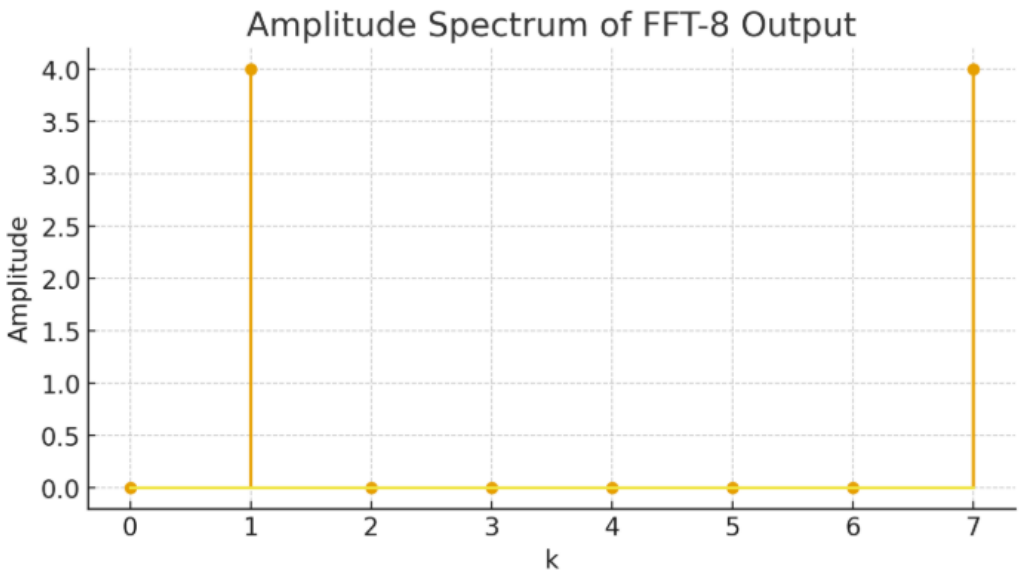
Wynika z tego że współczynniki k=1 i k=7 mają wartości bliskie oczekiwanym (re≈4 w float po normalizacji przez 32767). Pozostałe współczynniki mają wartości bardzo bliskie zeru (~±0.00009), co jest typowe przy ograniczeniach arytmetyki stałoprzecinkowej.

```
// Input: x[n] = cos(2*pi*n/8)   (Q1.15)
// Precomputed Q1.15 values:
in_re0 = 16'h7fff; //  1.0000
in_re1 = 16'h5a82; //  0.7071
in_re2 = 16'h0000; // ~0
in_re3 = 16'ha57e; // -0.7071
in_re4 = 16'h8001; // -1.0000 (-32767)
in_re5 = 16'ha57e; // -0.7071
in_re6 = 16'h0000; // ~0
in_re7 = 16'h5a82; //  0.7071
```

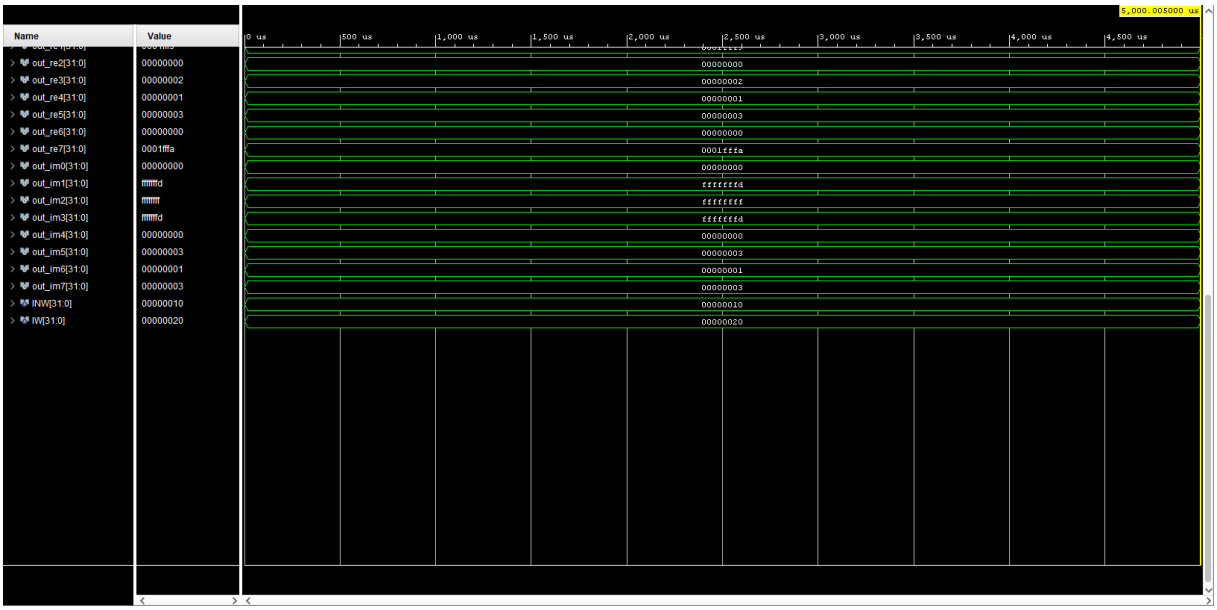
Rys. 3 Dane wejściowe do FFT

```
FFT-8 outputs (integer Q-format scaled by 2^15):
k |  re (int)    imag (int)    |  re (float)        imag (float)
-----------------------------------------------------------------
0 |       -1        0 |    -0.000031     0.000000
1 |   131065       -3 |     3.999908    -0.000092
2 |        0       -1 |     0.000000    -0.000031
3 |        2       -3 |     0.000061    -0.000092
4 |        1        0 |     0.000031     0.000000
5 |        3        3 |     0.000092     0.000092
6 |        0        1 |     0.000000     0.000031
7 |   131066        3 |     3.999939     0.000092
```

Rys. 4 Dane wyjściowe po operacji FFT



Rys. 5 Wykres amplitudy dla widma FFT-8 dal podanych danych wejściowych



Rys. 6 Dane wyjściowe po operacji FFT w formie przebiegów

6. Bibliografia

Fast Fourier Transform, Wikipedia: https://en.wikipedia.org/wiki/Fast_Fourier_transform
Github: https://github.com/BezStresu/Cooley-Tukey_FFT_FPGA