Projekt: Cooley-Tukey FFT algorithm implementation as IP-Core for Zynq7000 System on Chip

Wykonali: Adam Gawlik i Dominik Różycki

1. Opis algorytmu.

Algorytm Cooley-Tukey polega na optymalizacji obliczeń DFT przez dzielenie obliczeń macierzowych na kolejne przemnażanie danych wejściowych w kolejnych coraz to większych blokach o liczbie wejść równej potędze dwójki, przez współczynniki znane właśnie z macierzy DFT.

Celem naszego projektu, jest próba wykorzystania uproszczonego algorytmu, według schematu zamieszczonego poniżej do implementacji DFT na układzie FPGA.

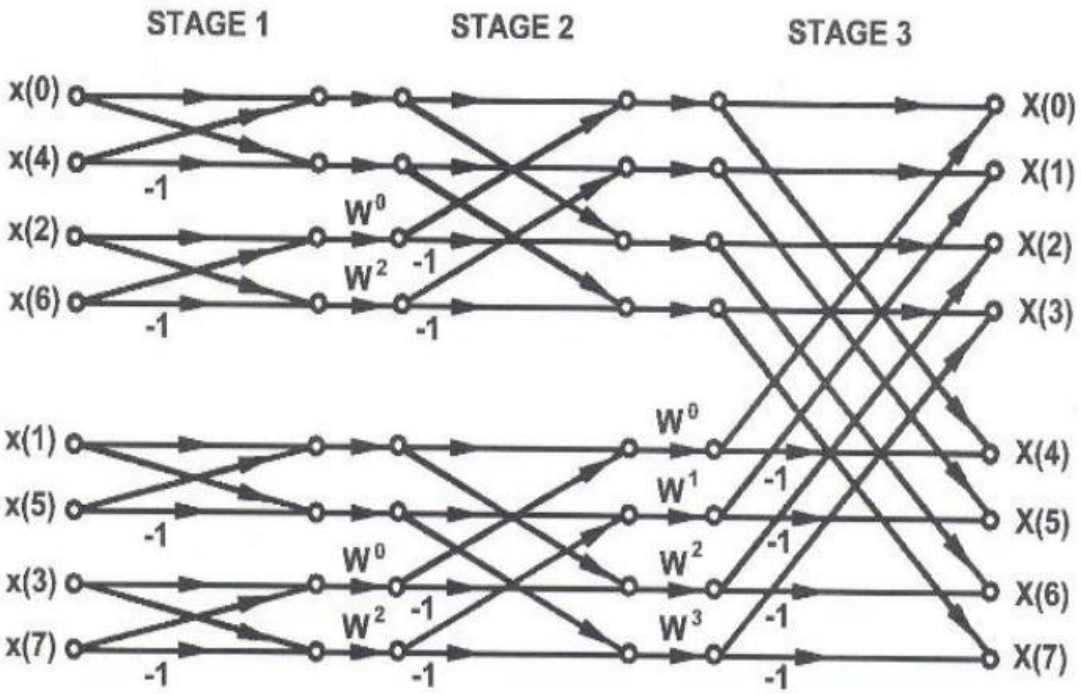


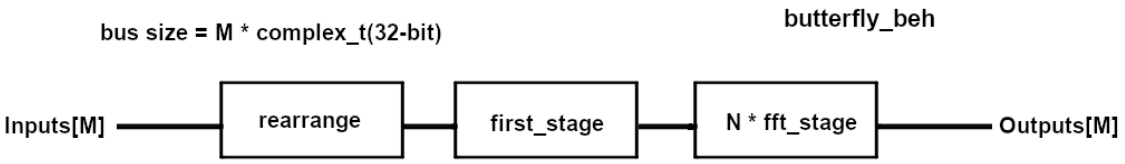
Figure 2. Butterfly diagram for a 8-point DIT FFT

Z poziomu SW, użytkownik powinien móc używać funkcji wysyłających do IP danych za pomocą magistrali AXI-Lite 4, oraz również za jej pomocą odczytywać wynik operacji.

Do projektu wykorzystany zostanie język SystemVerilog znany również jako IEEE 1800.
Jako symulator i narzędzie implementacji posłuży środowisko Xilinx Vivado.

2. Algorytm behawioralny w Verilogu/SV

W celu uproszczenia kodu, stworzona została biblioteka fft_package.sv, w której zawarte zostały kluczowe definicje typów, oraz funkcje.
Wykorzystane zostały one zarówno przez testbenche, model behawioralny jak i przez model RTL.
Między innymi został zdefiniowany typ complex_t o szerokości 32 bitów, składający się z dwóch spakowanych 16 bitowych słów w formacie stałoprzecinkowym(6 bitów integer + 10 bitów fraction).
Model behawioralny posiada równoległe wejścia M próbek, w zależności ile ich zadeklarowano przy tworzeniu instancji. Dodatkowo posiada on w sobie pipelining na wyjściu, który symuluje statyczny czas opóźnienia obliczanych próbek pomiędzy wejściem a wyjściem modułu.
Jeśli chodzi o obliczenia związane z FFT, to główne ich etapy to kolejno: zamiana kolejności próbek, pierwszy poziom FFT, oraz N pozostałych poziomów.
Liczba poziomów w FFT jest uzależniona od zadeklarowanej liczby próbek i genereowane są automatycznie.



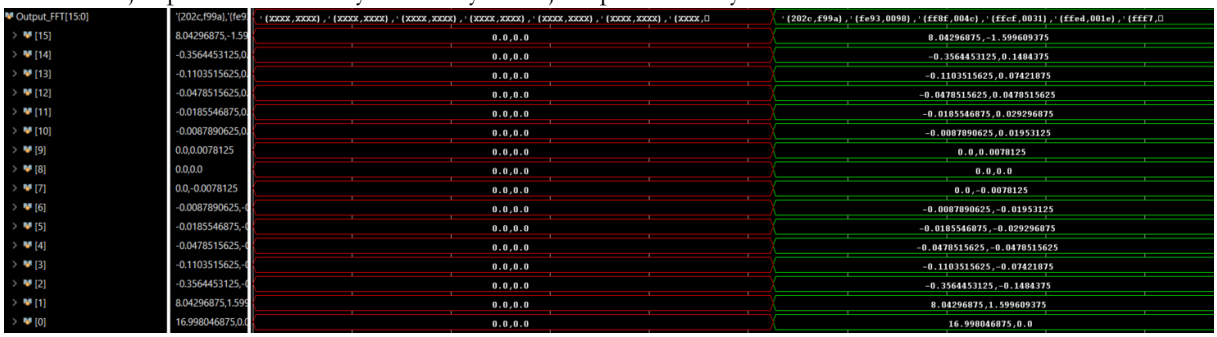
Schemat 1. Diagram blokowy algorytmu behawioralnego.

3. Testbench potwierdzający prawidłowe działanie modułu

Wektory wejściowe zostały wygenerowane przy pomocy skryptu w Matlabie w postaci pliku CSV, które to z kolei w testbenchu zostawały wczytywane w postaci tablicy o długości 16 próbek. Aby sprawdzić całkowitą poprawność algorytmu wygenerowane zostało 5 tablic z charakterystycznymi sygnałami:

- Stały,
- Prostokątny,
- Sinusoidalny,

Poniżej zaprezentowano wyniki dla symulacji z spróbkowanym Cosinusem.



Przebiegi 1. Otrzymane rezultaty z wyjścia algorytmu behawioralnego dla sygnału cosinus.

Dla porównania uzyskanych wyników w Matlabie zaimplementowano skrypt realizujący funkcję FFT na tych samych próbkach wejściowych, które prezentują się następująco:

FFT_Cos

16x1 complex double

	1	2
1	17.0000 + 0.0000i	
2	8.0476 + 1.6008i	
3	-0.3575 - 0.1481i	
4	-0.1126 - 0.0752i	
5	-0.0473 - 0.0473i	
6	-0.0206 - 0.0308i	
7	-0.0078 - 0.0189i	
8	-0.0018 - 0.0090i	
9	0.0000 + 0.0000i	
10	-0.0018 + 0.0090i	
11	-0.0078 + 0.0189i	
12	-0.0206 + 0.0308i	
13	-0.0473 + 0.0473i	
14	-0.1126 + 0.0752i	
15	-0.3575 + 0.1481i	
16	8.0476 - 1.6008i	

Screenshot 1. Otrzymane wyniki ze skryptu Matlaba.

Jak widać powyżej wyniki delikatnie różnią się między sobą, lecz różnice te sięgają co najwyżej tysięcznych części, co pozwala nam uznać wyniki za poprawne. Różnice te głównie wynikają z dokładności algorytmu na które składają się takie czynniki jak:

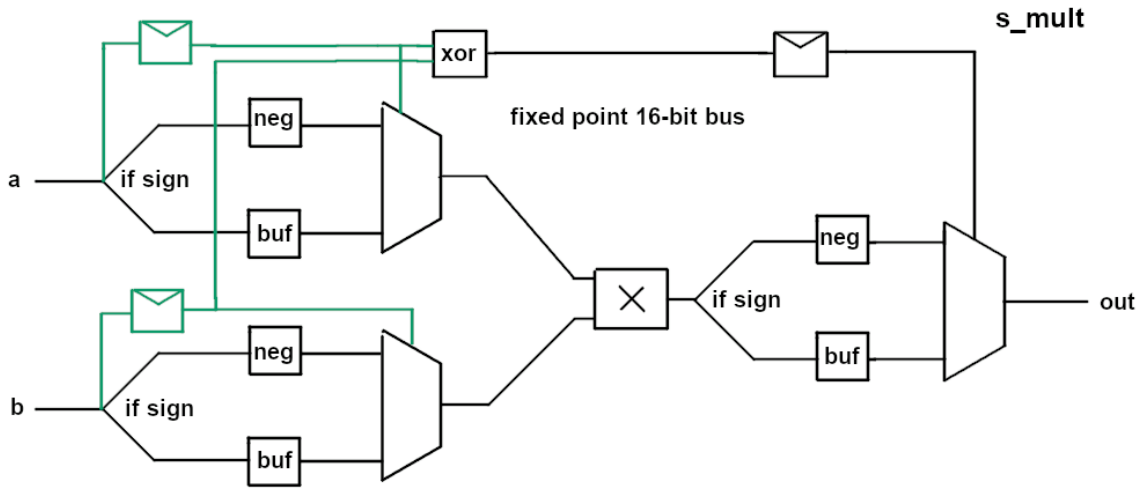
- Długość wartości fixed-point (w projekcie zostały użyte wartości fixed point 5:-10),
- Ograniczenie 32-bitowych rejestrów z wyjścia mnożarek na 16 bitów.

4. Moduł syntezowalny RTL

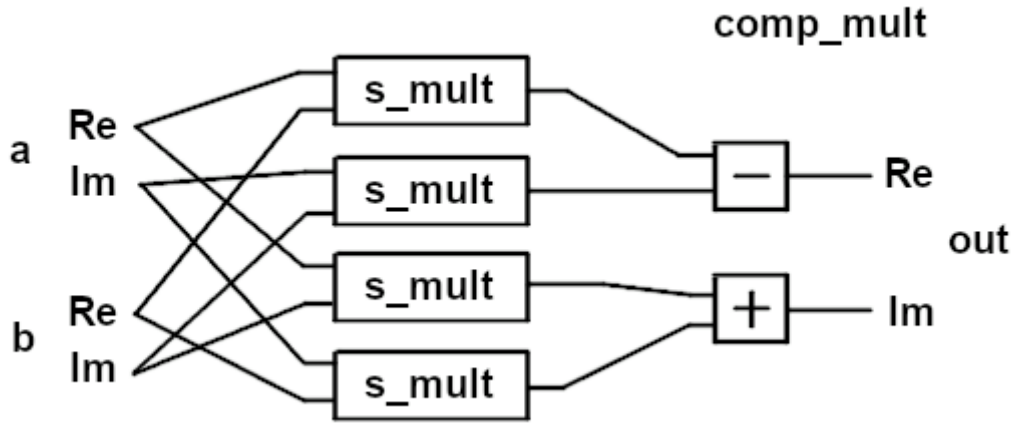
Wersja syntezowalna względem wersji behawioralnej różniła się sposobem dokonywania operacji FFT na próbkach. Po pierwsze duży nacisk postawiono na wersję potokową algorytmu, która by umożliwiła większą przepustowość danych kosztem większej Latency na wyjściu modułu RTL. Po drugie w module RTL wykorzystano dostępną pamięć RAM w dwóch celu przechowywania wartości przemnażanych wag w kolejnych stopniach algorytmu.
Dodatkowo w celu zainicjalizowania pamięci ROM napisano skrypt do generacji wartości rzeczywistych i urojonych wyżej wspomnianych współczynników.

Schemat 2. Poglądowy schemat modułu RTL.

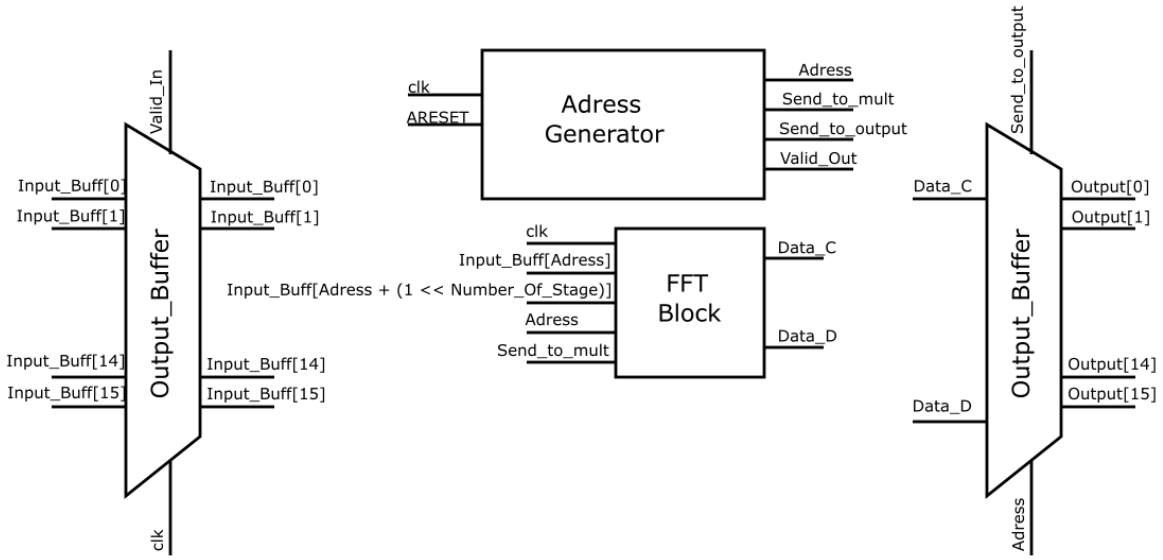
Pierwszy moduł od lewej odpowiada za zamianę kolejności próbek w rejestrze wejściowym co doskonale obrazuje Figure 2. Przykładowo próbka numer 1 zostaje przeniesiona w miejsce próbki 8 i na odwrót. W tym celu moduł wykorzystuje odwracanie bitowe adresów, dzięki czemu moduł wykorzystuje 16 taktów zegara (tyle samo co liczba próbek na wejściu) na odwrócenie kolejności próbek. Dla skończonej transakcji moduł generuje Valid_out sygnalizujący kolejnemu modułowi gotowość do odebrania wyjściowych próbek. Kolejny moduł odpowiada za już za realizację dodawania i odejmowania kolejnych próbek między sobą. To co wyróżnia ten moduł względem kolejnych to fakt, że w pierwszym stopniu można pominąć realizację mnożenia próbek przez odpowiednie wagi/współczynniki (gdyż jest to zwykłe przemnożenie przez -1). Dlatego też dla tego modułu został zaimplementowany dedykowany moduł Simplified_FFT_Block pomijający realizację mnożenia oraz nie wykorzystujący bloków pamięci w celu zaoszczędzenia bloków DSP. Synchronizacja pomiędzy modułami jest realizowana za pomocą sygnałów walidujących, które równoległe względem magistrali danych.



Schemat 3. Pojedynczy moduł s_mult realizujący mnożenie dwóch wartości fixed-point.



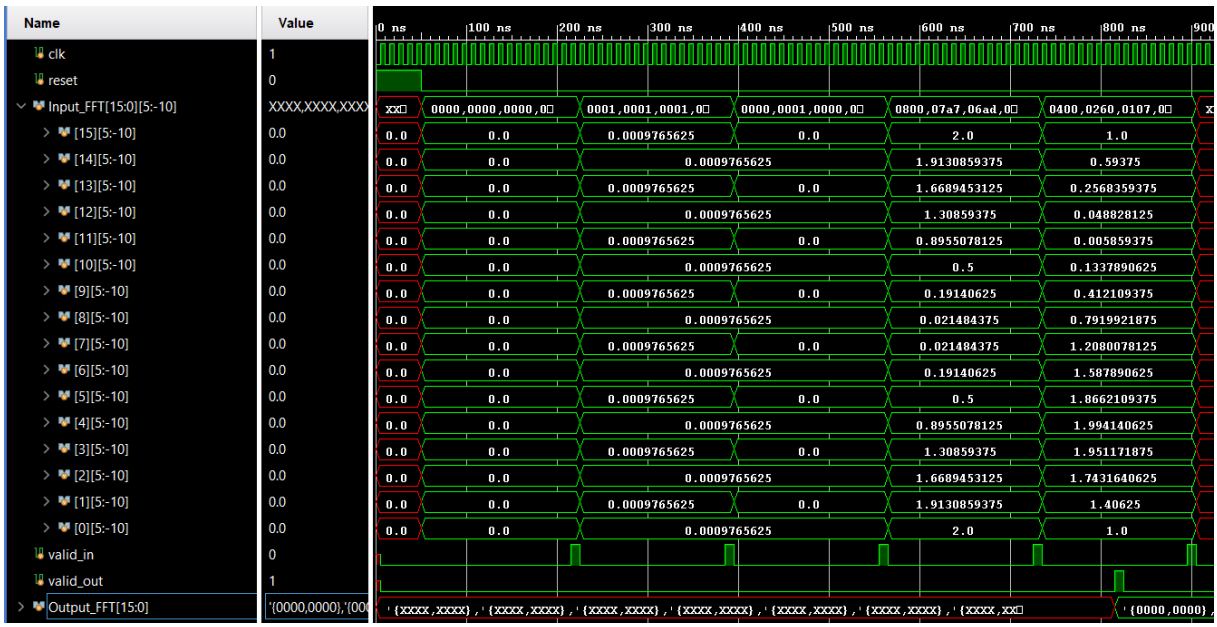
Schemat 4. Pojedynczy blok FFT.



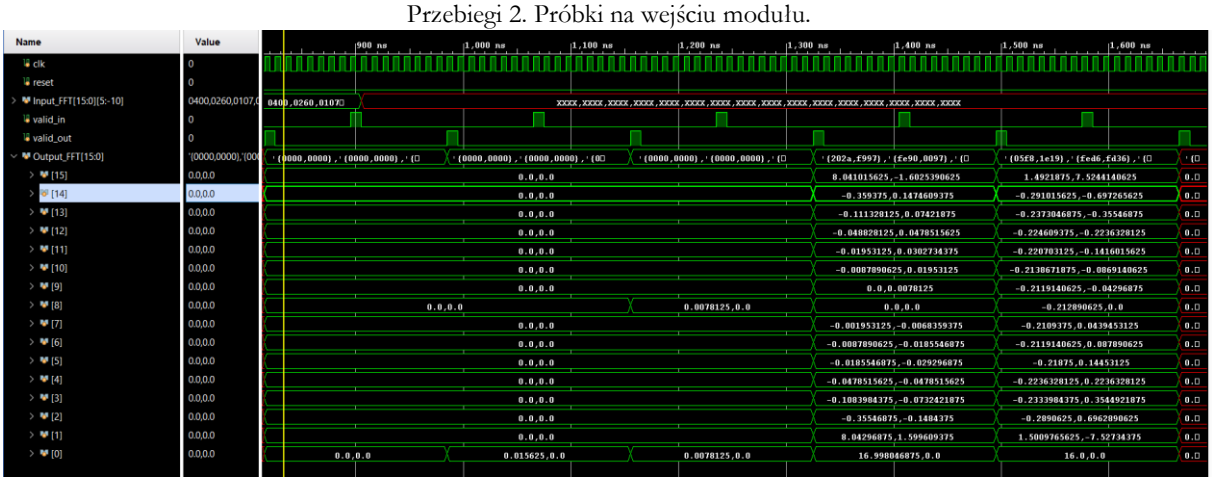
Schemat 5. Moduł Simplified_FFT_Stage.

5. Testbench modułu syntezowalnego RTL

Testbench modułu RTL wygląda podobnie do testbenchu modułu behawioralnego, lecz w celu ukazania realizacji potokowej modułu, co określony czas podawane zostają kolejne tablice z tymi samymi próbkami co w poprzednim testbenchu.



Przebiegi 2. Próbki na wejściu modułu.

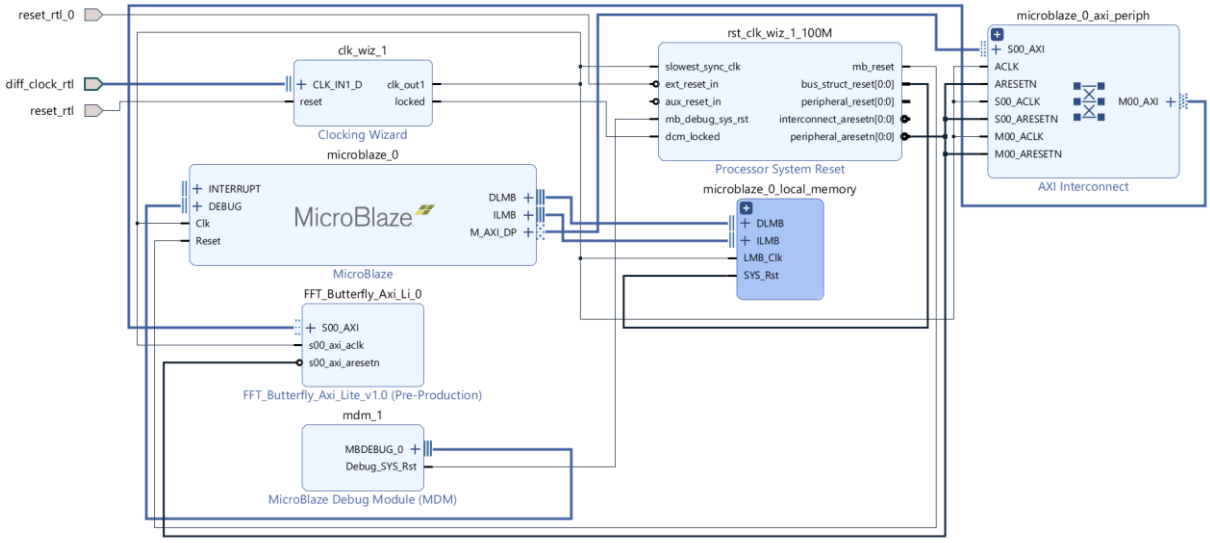


Przebiegi 3. Uzyskane rezultaty na wyjściu modułu.

Widać, że przykładowo dla spróbkowanego Cosinusa (czwarte próbki wyjściowe) uzyskane rezultaty są identyczne z tymi uzyskanymi przy okazji testbenchu algorytmu behawioralnego. Więc można uznać, że dla tego modułu generowane próbki są poprawne. Co należy również zauważyć z przebiegu, to że otrzymujemy próbki co 170ns co pokazuje, że udało się również zrealizować potokowość modułu.

6. Uzbrojenie modułu w AXI(lite) oraz podłączenie do mikroprocesora Microblaze.

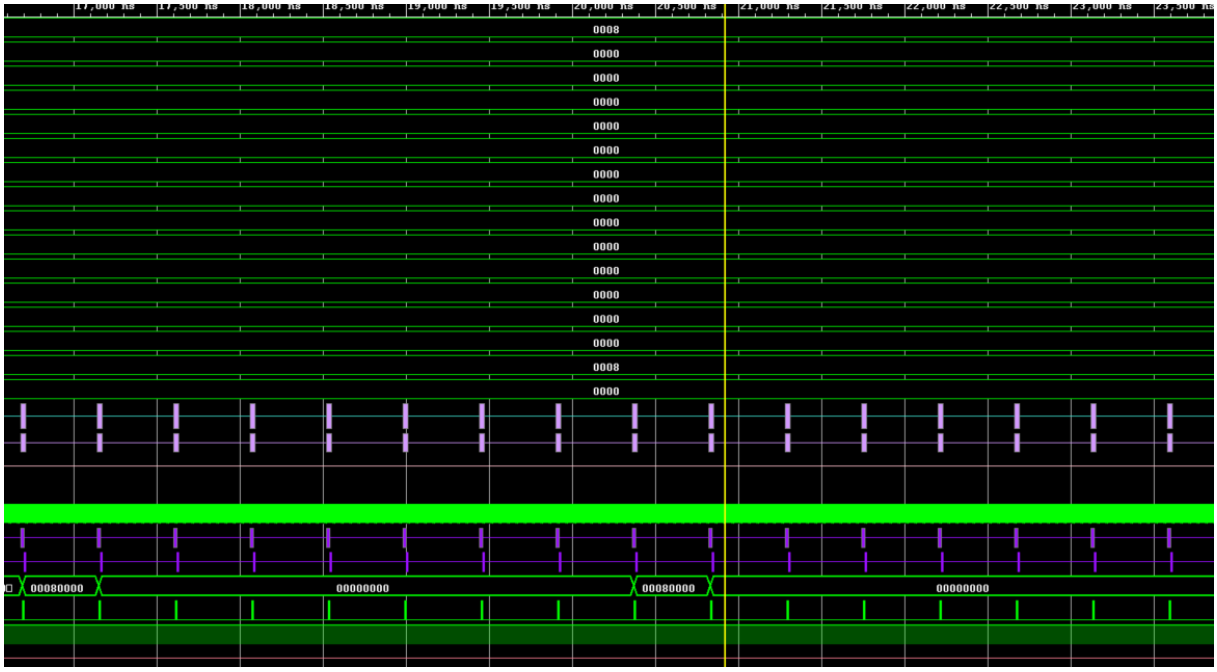
Kolejny etap projektu dotyczył podłączenie modułu RTL do mikroprocesora w celu zobrazowania poprawności działania magistrali AXI lite.



Schemat 6. Diagram blokowy modułu FFT_Butterfly z mikroprocesorem Microblaze.

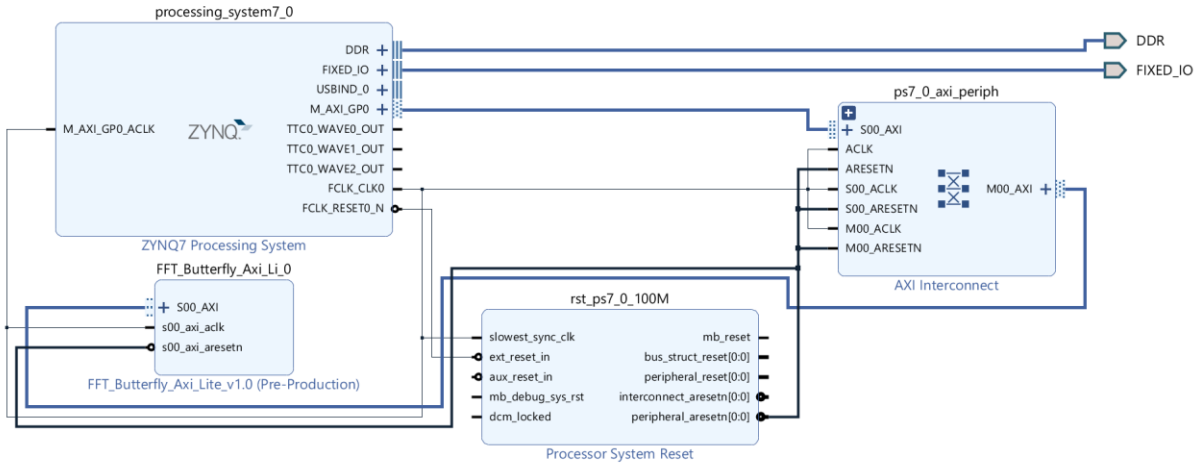
7. Symulacja post-syntezowa Microblaze-a z modułem FFT_Butterfly

Dla tak skonstruowanego modułu napisano prostą aplikację wysyłającą tablicę z 16 próbkami do modułu FFT_Butterfly oraz odbierającą uzyskane wyniki.

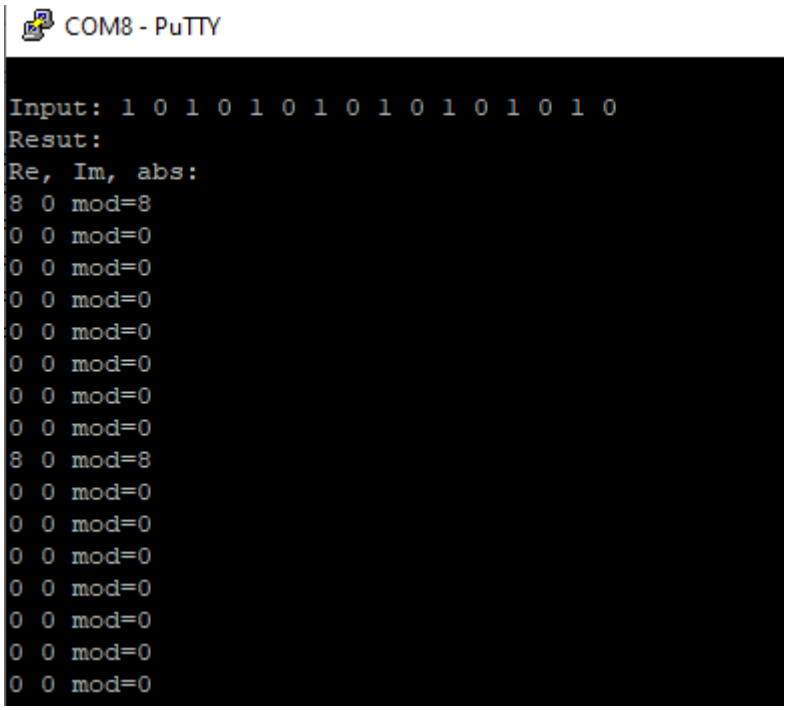


Przebiegi 4. Dane wysyłane przy pomocy magistrali AXI Lite.

8. Podłączenie modułu FFT_Butterfly do procesora ARM.



Schemat 7. Diagram blokowy modułu FFT_Butterfly z procesorem ARM.



Screenshot 2. Otrzymane rezultaty.