Sprzętowa implementacja algorytmu Jacobiego do diagonalizacji macierzy

Piotr Radecki, Aleksander Strzeboński

I. Podstawy teoretyczne

A. Metoda Jacobiego

Algorytm Jacobiego jest algorytmem iteracyjnym do rozkładu macierzy na wartości własne. Pozwala z macierzy wejściowej symetrycznej i rzeczywistej $\boldsymbol{A}$ o rozmiarach $\mathrm{N} \times \mathrm{N}$ uzyskać macierz zdiagonalizowaną $\boldsymbol{W}$ oraz macierz wektorów własnych $\boldsymbol{V}$.
W każdej iteracji macierz $\boldsymbol{W}$ jest aktualizowana:

$$\boldsymbol{W}^{(k+1)} = \mathbf{G}^{\mathrm{T}}(\theta)\boldsymbol{W}^{(k)}\mathbf{G}(\theta) \tag{1}$$

gdzie macierz $\mathbf{W}^{(0)} = \mathbf{A}$ a macierz $\mathbf{G}$ jest nazywana macierzą rotacji i dana jako:

$$\begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} & & \mathrm{i} & & \mathrm{j} & & \end{array} \\ \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & \mathrm{c} & \dots & \mathrm{s} & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & \text{-s} & \dots & \mathrm{c} & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix} \end{array}\!\!\begin{array}{c} \\ \\ \mathrm{i} \\ \\ \mathrm{j} \\ \\ \end{array}$$

gdzie $s = \sin(\theta)$, $c = \cos(\theta)$ a kąt $\theta(i, j)$ może być obliczony jako:

$$\tan 2\theta = \frac{2W_{ij}}{W_{jj} - W_{ii}} \tag{2}$$

Można udowodnić [1], że taka operacja zmniejsza zawartość elementów pozadiagonalnych. W rezultacie przy odpowiednio dużej liczbie iteracji K dla par liczb i, j możliwe jest uzyskanie niemal diagonalnej macierzy.
Macierz wektorów własnych to iloczyn kolejnych macierzy rotacji:

$$\boldsymbol{V}^{(K)} = \prod^{K} \mathbf{G}(\theta^{(k)}), \tag{3}$$

Mnożenie macierzy z (1) można też zapisać jako:

$$\begin{cases} W_{ii}^{(k+1)} = c(cW_{ii}^{(k)} - sW_{ij}^{(k)}) - s(cW_{ij}^{(k)} - sW_{jj}^{(k)}) \\ W_{jj}^{(k+1)} = s(sW_{ii}^{(k)} + cW_{ij}^{(k)}) + c(sW_{ij}^{(k)} + cW_{jj}^{(k)}) \\ W_{ij}^{(k+1)} = W_{ji}^{(k+1)} = 0 \\ W_{in}^{(k+1)} = W_{ni}^{(k+1)} = cW_{in}^{(k)} - sW_{jn}^{(k)} \; n \neq i, j \\ W_{jn}^{(k+1)} = W_{nj}^{(k+1)} = sW_{in}^{(k)} + cW_{jn}^{(k)} \; n \neq i, j \\ W_{ln}^{(k+1)} = W_{nl}^{(k+1)} = W_{ln}^{(k)} \; n, l \neq i, j. \end{cases} \tag{4}$$

Równanie (3) można równoważnie zapisać jako:

$$\begin{cases} V_{ni}^{(k+1)} = cV_{ni}^{(k)} - sV_{nj}^{(k)} \\ V_{nj}^{(k+1)} = sV_{ni}^{(k)} + cV_{nj}^{(k)}, \end{cases} \tag{5}$$

Te zapisy będą przydatne podczas sprzętowej implementacji algorytmu.

B. Algorytm Cordic

Algorytm CORDIC to prosty i lekki algorytm do obliczania funkcji trygonometrycznych, który może być zaimplementowany w trybie "rotation" i "vectoring". Istnieje kilka podobnych implementacji procesora. Poniższy opis dotyczy szczególnej implementacji użytej w tej pracy. W trybie "rotation" algorytm przyjmuje następujące wejścia:

$$\begin{cases} \mathrm{x}_{in} = x \\ \mathrm{y}_{in} = y \\ \mathrm{z}_{in} = \alpha, \end{cases} \tag{6}$$

oraz zwraca następujące wyjścia:

$$\begin{cases} \mathrm{x}_{out} = x\cos(\alpha) - y\sin(\alpha) \\ \mathrm{y}_{out} = x\sin(\alpha) + y\cos(\alpha) \\ \mathrm{z}_{out} = 0, \end{cases} \tag{7}$$

W trybie "vectoring" wejścia to:

$$\begin{cases} \mathrm{x}_{in} = x \\ \mathrm{y}_{in} = y \\ \mathrm{z}_{in} = 0, \end{cases} \tag{8}$$

zaś wyjścia to :

$$\begin{cases} \mathrm{x}_{out} = \sqrt{x^2 + y^2} \\ \mathrm{y}_{out} = 0 \\ \mathrm{z}_{out} = \arctan(\frac{y}{x}). \end{cases} \tag{9}$$

Szczegółowe uzasadnienia mogą być znalezione w [2].
W tej pracy CORDIC będzie używany do wyliczania funkcji trygonometrycznych.

II. Implementacja

A. Użycie CORDIC

Dzięki użyciu układu FPGA możliwe jest bardzo efektywna implementacja systemu do obliczania wartości i wektorów własnych. Algorytm CORDIC został wspomniany w poprzednim rozdziale ponieważ pozwala stworzyć logikę

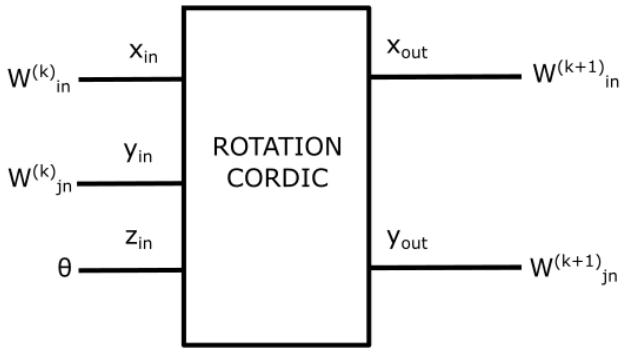

Fig. 1. Obliczanie elementów pozadiagonalnych $\boldsymbol{W}$ i elementów $\boldsymbol{V}$

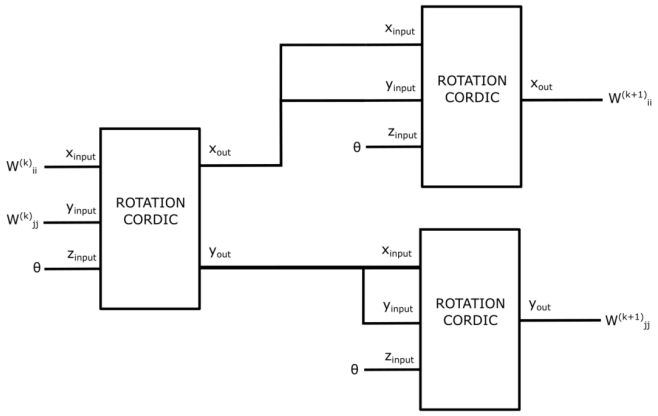

Fig. 2. Obliczanie elementów diagonalnych

konieczną do metody Jacobiego **bez użycia układów mnożnących**.

Po porównaniu równań (4) i (5) z równaniem (7) okazuje się, że obliczenie nowych wartości elementów diagonalnych $\boldsymbol{W}$ może być zrealizowane za pomocą podwójnego użycia CORDICA, natomiast do obliczenia pozostałych elementów $\boldsymbol{W}$ i nowych elementów $\boldsymbol{V}$ wystarczający będzie pojednyncze użycie CORDICA. W tym przypadku CORDIC będzie pracował w trybie "rotation". Procesor do obliczania rotacji dla elementów diagonalnych znajduje się na Fig. 2 natomiast dla pozadiagonalnych na Fig. 1. Należy jednak zauważyć, że jest to jedynie rysunek poglądowy. W właściwym układzie używa się tylko jednego CORDICA w wersji "rotation" w trybie potokowym.
Co więcej, kąt θ można obliczyć za pomocą CORDICA w trybie vectoring. Tutaj uzasadnieniem może być porównanie równań (2) i (9). Co ważne CORDIC pracuje dobrze dla kątów z przedziału $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Aby uporać się z tym problemem do obliczeń używamy algorytmu pokazanego na schemacie fig. 3.

Można zatem krótko podsumować, że dla wybranych i, j moduł wykonuje następujące kroki:

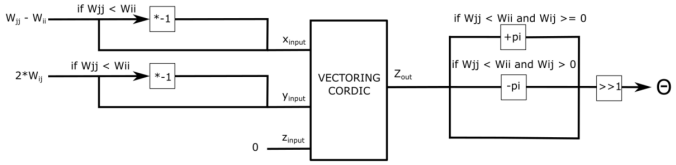

Fig. 3. Obliczanie kąta.

- Znalezienie θ za pomocą CORDIC "vectoring".
- Pomnożenie macierzy za pomocą CORDIC "rotation"
- Aktualizacja elementów macierzy w pamięci RAM.

B. Dobór iteracji

Ważnym elementem jest rzecz do tej pory nie omawiana- jak dobrać kolejne wartości i, j. Algorytm składa się z kilku kolejno następujących epok zwanych dalej sweepami. W każdej takiej epoce przeszukiwany jest tak zwany górny trójkąt indeksów macierzy czyli wszystkie możliwe dwuelementowe podzbiory.

Sweep składa się z rund. W każdej rundzie dobierane jest $N/2$ par liczb i, j pokrywających cały zakres indeksów (na przykład dla $N = 8 : 0-3, 4-5, 2-6, 1-7$). Kolejne rundy wewnątrz sweepa są realizowane za pomocą algorytmu każdy z każdym zaimplementowanego metodą kołową [3]. Sweep się kończy w momencie gdy każdy indeks zostanie sparowany z każdym

Dzięki takiej organizacji jest możliwe podczas każdej rundy równoległe policzenie θ dla wszystkich par liczb z uwagi na to, że rotacja dla i, j zmienia tylko kolummy i rzędy macierzy $\boldsymbol{W}$ o indeksach i, j . W przypadku tej implementacji kąt zostanie obliczony przy pomocy potokowej architektury z Fig. 3. Wymaga to zatem zaimplementowania algorytmu CORDIC w wersji potokowej.

Po obliczeniu kątów kolejne rotacje wewnątrz rundy obliczane są szeregowo. Po wykonaniu wszystkich rotacji w danej rundzie następuje wygenerowanie nowych par i kolejna runda lub koniec sweepa.
Istnieją implementacje, które wewnątrz jednej rundy potrafią równolegle liczyć rotację dla wszystkich par liczb i, j wygenerowanyc na potrzeby tejże rundy. Takie podejście wymaga jednak użycia $N/2$ modułów "Rotation" CORDIC i "Vectoring" CORDIC a także znacznie zwiększają skomplikowanie handshaków. Zaproponowana architektura jest nieco wolniejsza ale pozwala ograniczyć do minimum zużycie zasobów sprzętowych.

C. Architektura

Architektura układu jest pokazana na fig. 4. Poniżej opisany jest sposób działania poszczególnych składowych układu:

- **Main Controller** - Zarządza kolejnymi fazami algorytmu (sweepy, rundy, liczby i, j). Komunikuje się z blokiem RAM i mikrokontrolerem. W każdej rundzie najpierw

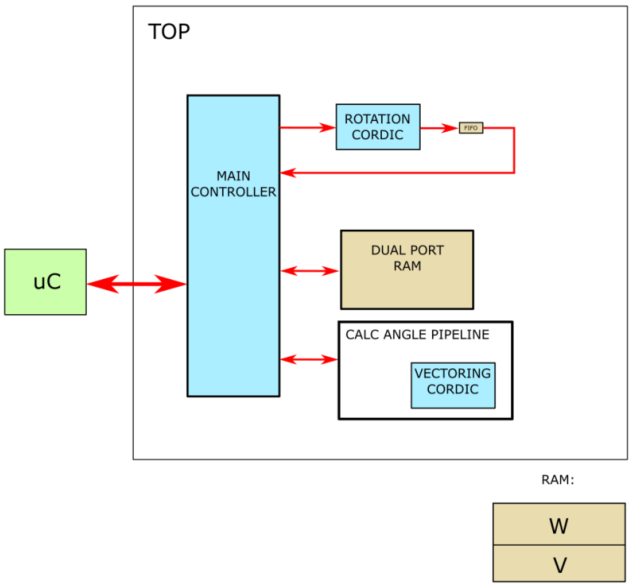

Fig. 4. Architectura

zleca obliczenie kątów a później obsługuje obliczenie rotacji. Aby obliczyć rotację podaje na CORDICA kolejne elementy macierzy z pamięci. Następnie elementy diagonalne są wrzucane na drugą operację do CORDICA i elementy są kolejno czytane z FIFO i zapisywane do pamięci.

- **Calc Angle Pipeline** - Moduł zawierający vectoring cordic i obliczający kąt dla danych indeksów.
- **Rotation Cordic** - Instancja modułu rotation cordic służąca do rotacji macierzy $\boldsymbol{W}$.
- **RAM** - Pamięć przechowująca obecny stan macierzy $\boldsymbol{W}$ i $\boldsymbol{V}$. Pamięć jest dwuportowa. Podczas czytania elementów wejściowych do CORDICA oba porty wykonują operację odczytu a podczas aktualizacji macierzy oba porty wykonują operację zapisu.

Architektura używa formatu liczb $Q(1.4.15)$ natomiast dane wejściowe są w formacie $Q(1.0.15)$. Maksymalne elementy macierzy W mogą osiągnąć wartość N, które w tym wypadku zostało przyjęte jako 8. Oznacza to, że diagonalizowana macierz będzie miała rozmiar 8×8. Wszystkie transakcje pomiędzy modułami odbywają się za pomocą AXI4-S.

Macierz $\boldsymbol{W}$ jest reprezentowana jako górny trójkąt (fig. 5). Ponieważ $\boldsymbol{W}_{nl} = \boldsymbol{W}_{ln}$, gdzie $n < l$ można przekazywać tylko ten pierwszy element. Podczas transakcji po AXIS-4 przekazywane są szeregowo kolejne wektory horyzontalne macierzy. Podobnie jest z reprezentacją w pamięci.

Macierz $\boldsymbol{V}$ jest reprezentowana w pełni ze względu na brak symetrii. Podobnie jak w przypadku $\boldsymbol{W}$, jest przechowywana i przekazywana jako kolejne wektory poziome.

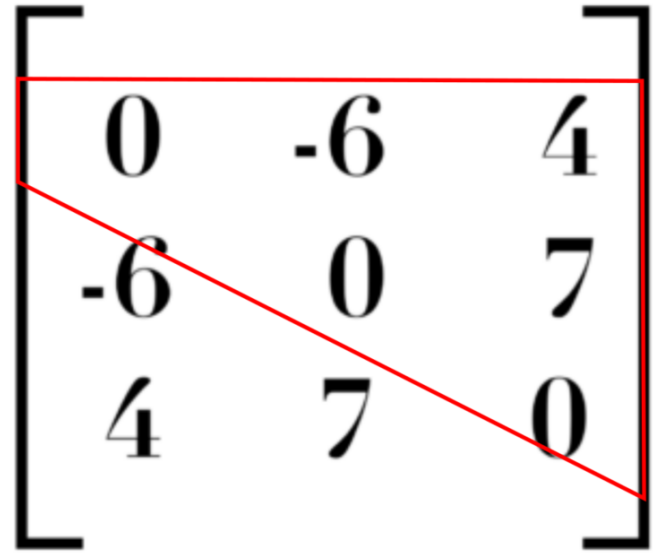

Fig. 5. Reprezentacja symetrycznej macierzy.

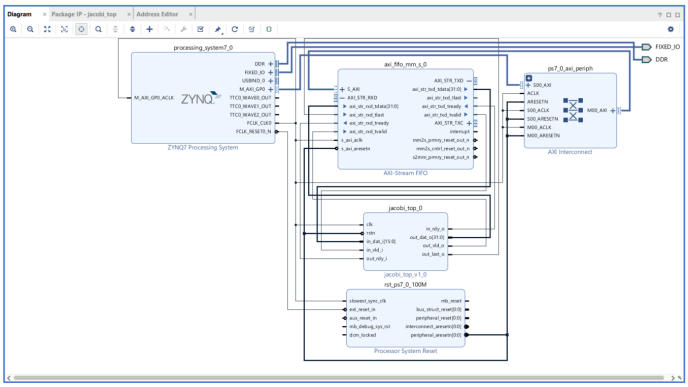

Fig. 6. System

D. Integracja

System został zaimplementowany na platformie Zynq. Został uruchomiony jako peryferium mikrokontrolera ARM. Moduł jest skomunikowany z procesorem za pomocą IP-Core od firmy Xilinx "AXI Stream FIFO". Pełny system jest przedstawiony na rysunku fig. 6. Moduł został uruchomiony na częstotliwości zegara 100MHz.

III. Weryfikacja

A. Model Referencyjny

Model referencyjny algorytmu został stworzony w języku python za pomocą biblioteki fxp-math. Zawiera behawioralny opis algorytmu w fixed point. Model będzie używany do generowania wektorów testowych do weryfikacji układu. Ideą projektu jest to aby w Pythonie wygenerować wejścia i wyjścia algorytmu. Następnie podczas testowania wyjścia układu muszą dokładnie pokrywać się z wyjściami modelu.

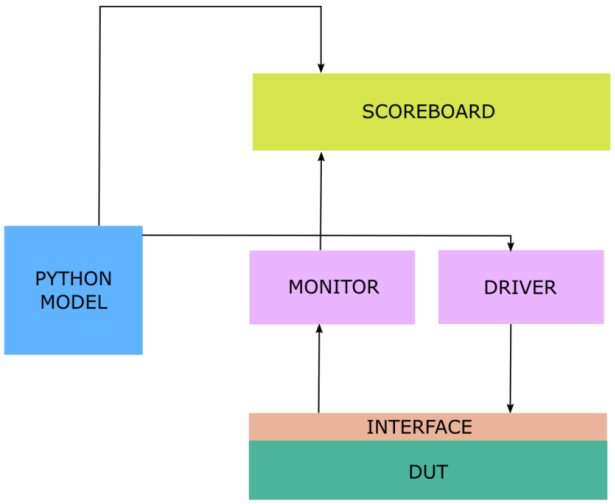

Fig. 7. Diagram UVM dla testbencha.

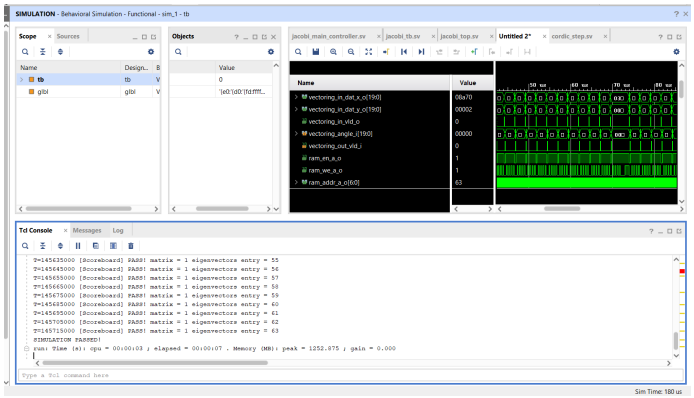

Fig. 8. Symulacja w Vivado.

```
Let's calculate eigenvalues
Input matrix:
[0.7379 -0.1585 0.3854 0.4633 0.5201 0.0140 0.6165 0.2286]
[-0.1585 -0.2907 0.5947 0.2359 0.1153 -0.2915 0.4894 -0.3591]
[0.3854 0.5947 0.7668 -0.0934 0.1857 0.4251 0.8674 -0.1826]
[0.4633 0.2359 -0.0934 0.0214 -0.1231 -0.0447 0.4969 0.1309]
[0.5201 0.1153 0.1857 -0.1231 -0.6306 0.1897 -0.2693 0.6904]
[0.0140 -0.2915 0.4251 -0.0447 0.1897 -0.6011 0.1729 0.2832]
[0.6165 0.4894 0.8674 0.4969 -0.2693 0.1729 -0.7814 0.4625]
[0.2286 -0.3591 -0.1826 0.1309 0.6904 0.2832 0.4625 0.9559]
Eigenvalues matrix:
[-1.8099 0.0000 0.0000 -0.0000 0.0000 -0.0000 0.0000 -0.0000]
[0.0000 -1.1254 -0.0000 -0.0000 0.0000 0.0000 -0.0000 -0.0000]
[0.0000 -0.0000 -0.7579 0.0000 0.0000 -0.0000 0.0000 -0.0000]
[-0.0000 -0.0000 0.0000 -0.3600 0.0000 -0.0000 -0.0000 0.0000]
[0.0000 0.0000 0.0000 0.0000 0.1752 -0.0000 0.0000 -0.0000]
[-0.0000 0.0000 -0.0000 -0.0000 -0.0000 0.5867 0.0000 0.0000]
[0.0000 -0.0000 0.0000 -0.0000 0.0000 0.0000 1.4484 -0.0000]
[-0.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000 -0.0000 2.0202]
Eigenvectors matrix:
[0.2300 0.1569 -0.1461 0.0964 0.4872 0.5765 -0.0993 0.5577]
[0.2994 0.4613 0.1318 0.6066 -0.3970 -0.0502 0.3617 0.1399]
[0.1697 -0.4526 -0.0235 -0.1217 0.1181 -0.4471 0.5134 0.5214]
[0.0572 -0.3185 0.4278 -0.2684 -0.5871 0.4921 -0.0092 0.2262]
[-0.4830 -0.1644 0.5977 0.4646 0.2407 -0.1263 -0.2196 0.2034]
[0.1297 0.5738 0.4959 -0.5378 0.1445 -0.2659 -0.0554 0.1470]
[-0.7132 0.3094 -0.3547 -0.1487 -0.2736 0.0042 0.1217 0.3964]
[0.2548 -0.0494 -0.2146 0.0827 -0.3025 -0.3664 -0.7267 0.3534]
```

Fig. 9. Wynik działania programu.

```
Wartości własne:
[ 2.02407345  1.45048993 -1.81231179  0.58723672  0.175933   -1.12699334
 -0.36096326 -0.75930309]
Wektory własne:
[[ 0.55781286 -0.0989291   0.23002323  0.57669614  0.48811064  0.15718433
   0.09607513 -0.14621691]
 [ 0.13978648  0.36204537  0.29984492 -0.05035257 -0.39694476  0.46191321
   0.60715116  0.13182962]
 [ 0.5223792   0.51397929  0.17000323 -0.44744528  0.11818542 -0.45232418
  -0.12129656 -0.02364457]
 [ 0.22681653 -0.00959394  0.05777946  0.49302631 -0.58694962 -0.31872007
  -0.26850782  0.42878903]
 [ 0.20355252 -0.22020679 -0.48371543 -0.1269877   0.24121417 -0.16446607
   0.46574293  0.59817975]
 [ 0.14712399 -0.05589779  0.13003407 -0.26647457  0.1437567   0.57472652
  -0.53800388  0.49687794]
 [ 0.39643053  0.12127923 -0.71348409  0.00458517 -0.27430891  0.30971848
  -0.14882973 -0.35458765]
 [ 0.35318907 -0.72701113  0.25493871 -0.36667202 -0.30267074 -0.04964301
   0.08253971 -0.21536799]]
```

Fig. 10. Wyniki referencyjne.

B. Testbench

Testbench dla projektu został napisany w języku SystemVerilog. Weryfikacja obejmuje zarówno implementacje modułów CORDIC jak i algorytm Jacobiego jako całość. Testbench pobiera macierze z przygotowanego w Pythonie pliku, ładuje je do DUT i czyta wyjścia. Wyjścia są porównywane z referencyjnymi wynikami z Pythona. Test jest napisany obiektowo zgodnie z diagramem przedstawionym na rysunku fig. 7.
Udało się przesymulować IP zgodnie z założeniami, tj. model referencyjne całkowicie pokrył się z rtl. Wynik symulacji jest przedstawiony na fig. 8.

C. Uruchomienie Układu

Układ został zaimplementowany zgodnie z fig. 6. Za pomocą softwaru wysyłana jest na układ Jacobiego macierz wejściowa do diagonalizacji a następnie odbierane są dane wyjściowe. Komunikacja odbywa się za pomocą AXI-LITE. Screenshot z działania programu znajduje się na rysunku 9
Ten wynik może być porównany z funkcją do obliczania wartości własnych w bibliotece numpy (fig. 10). Podczas porównywania należy zwrócić uwagę na inną kolejnosc wartości własnych a co za tym idzie wektorów własnych w kolumnach. Można jednak zauważyć, że otrzymane wartości są bardzo podobne.
Na koniec tego raportu warto krótko podsumować działanie układu. Diagonalizacja macierzy zajmuje 70us. Jest to nieco dłużej niż w wykonanie funkcji np.linalg.ein() na komputerze osobistym z procesorem intel i5 2.5GHz 2 rdzenie. Tutaj czas to około 40us. Należy wziąć jednak pod uwagę 25-krotną różnicę taktowania (a biorąc pod uwagę oba rdzenie 50-krotną).

REFERENCES

[1] J. Lambers, *Lecture 7 notes*. [Online]. Available: https://web.stanford.edu/class/cme335/lecture7.pdf.
[2] X. Jin, *Implementation of the music algorithm in cash*.
[3] *Round robin tournament*. [Online]. Available: https://en.wikipedia.org/wiki/Round-robin_tournament.