POZNAN UNIVERSITY OF TECHNOLOGY ACADEMIC JOURNALS
No 84 Electrical Engineering 2015

Maciej CZYŻAK*
Robert SMYK*

OBLICZANIE MODUŁU LICZBY ZESPOLONEJ W FPGA Z UŻYCIEM ALGORYTMU CORDIC

W pracy przedstawiono obliczanie modułu liczb zespolonych z użyciem zmodyfikowanej wersji algorytmu CORDIC przy zastosowaniu pięciu stopni iteracyjnych. Zaprezentowano zależność wielkości błędu od liczby stopni algorytmu CORDIC dla arytmetyki zmiennoprzecinkowej jak również zbadano wpływ użycia arytmetyki całkowitej. Zaproponowana modyfikacja algorytmu CORDIC dla arytmetyki całkowitej polega na wprowadzeniu korekcji po zakończeniu podstawowych obliczeń w celu zmniejszenia maksymalnego błędu. Wartość korekcji jest ustalana na podstawie stosunku współrzędnych uzyskanych po piątym stopniu iteracyjnym. Korekcja pozwala na około dwukrotną redukcję błędu maksymalnego. W pracy pokazano też przykładową architekturę układu realizującego zmodyfikowany algorytm w układzie FPGA.

SŁOWA KLUCZOWE: moduł liczby zespolonej, CORDIC, FPGA

1. WSTĘP

Obliczanie modułu liczby zespolonej jest konieczne w cyfrowym przetwarzaniu sygnałów (CPS) w tych algorytmach, w których na wyjściu uzyskuje się wynik w postaci zespolonej. Do najważniejszych z nich należą szybka transformacja Fouriera (ang. FFT - Fast Fourier Transform) oraz zespolone filtry o skończonej długości odpowiedzi impulsowej (SOI; ang. FIR - Finite Impulse Response). Obliczanie modułu próbki transformaty na wyjściu procesora FFT wymaga obliczenia sumy kwadratów części rzeczywistej i części urojonej i następnie pierwiastka kwadratowego z sumy. Generalnie pierwiastek kwadratowy należy do tzw. arytmetycznych funkcji standardowych i do jego obliczania w komputerach ogólnego przeznaczenia stosowane są procedury z bibliotek numerycznych lub też częściej koprocesory arytmetyczne przechwytujące z kodu wygenerowanego przez kompilator rozkazy związane w wykonaniem operacji arytmetycznych. Jednak w przypadku procesorów FFT czy też filtrów FIR obliczanie wartości modułu próbki sygnału wyjściowego przy zastosowaniu podejść standardowych nie pozwala spełnić wymagań co do szybkości działania, ze względu na wymagany czas obliczeń, jak też nie daje możliwości potokowania z częstotliwościami rzędu setek MHz wymaganymi w

* Politechnika Gdańska.

pewnych zastosowaniach przy pracy przepływowej. Istnieje też problem dopuszczalnej złożoności sprzętowej. Czynnikiem ułatwiającym zadanie jest stosunkowo niewielki zakres liczbowy, ograniczony zwykle do 12-14 bitów. W pracy zaproponowano obliczanie modułu liczby zespolonej z zastosowaniem zmodyfikowanego algorytmu CORDIC oraz przykładową architekturę układu do jego implementacji. Poniżej w poszczególnych podrozdziałach przedstawiono kolejno przegląd wybranych metod obliczania pierwiastka kwadratowego, algorytm CORDIC, wyniki badań symulacyjnych oraz architekturę układu w FPGA i wyniki syntezy w środowisku Xilinx.

2. PRZEGLĄD METOD OBLICZANIA MODUŁU LICZBY ZESPOLONEJ

Do obliczania modułu liczby zespolonej można wykorzystać jeden z odpowiednio zmodyfikowanych standardowych algorytmów obliczania pierwiastka kwadratowego (PK). Istnieje szereg ogólnych algorytmów obliczania PK jak te oparte na rozwinięciu w szereg Taylora [1], babiloński [2] czy też zgrubnej estymacji [3], jednak ze względu na liczbę koniecznych do wykonania operacji arytmetycznych raczej nie są używane w implementacjach sprzętowych.

Przegląd algorytmy obliczania PK stosowanych w implementacjach sprzętowych przedstawiono w [4]. Zwykle w implementacjach takich stosuje się zmodyfikowane warianty technik nieodtwarzających [5, 6, 7]. Algorytmy tej grupy mogą być implementowane w postaci potokowej, jednak ich realizacja w tej formie może wprowadzać znaczne opóźnienie ze względu na wymaganą liczbę stopni obliczeniowych. Wprowadzane opóźnienie może być większe od dopuszczalnego czasu obliczeń. Dla implementacji sprzętowych rezultat z zadaną dokładnością może być uzyskany w ustalonym czasie (w ustalonej liczbie taktów zegarowych) w przeciwieństwie do metod iteracyjnych. Metody te mogą wykorzystywać algorytm Newtona-Raphsona (NR) [8]. Algorytm ten może zostać przekształcony do postaci beziteracyjnej poprzez realizację wybranej liczby iteracji w kolejnych stopniach układu, z których każdy wykonuje obliczenia dla jednej iteracji. Istotną przeszkodą w realizacji sprzętowej tego algorytmu jest konieczność wykonywania mnożenia i dzielenia, których wykonanie nie tylko spowalnia realizację algorytmu, ale też istotnie zwiększa złożoność sprzętową układu. Dodatkowo, błąd obliczeń dla ustalonej liczby stopni układu zależy od obranego punktu startowego punktu startowego algorytmu. Jeśli liczba podpierwiastkowa jest sumą kwadratów, jak to ma miejsce przy obliczaniu modułu liczb zespolonych, można zastosować algorytm alfa max plus beta min [9]. Algorytm ten w swojej oryginalnej wersji używa jednego lub dwóch regionów aproksymacji i pozwala obliczać moduł liczby zespolonej bez dzielenia i bez iteracji. W wersji z jednym regionem

aproksymacji pozwala obliczyć moduł liczby zespolonej z błędem nie przekraczającym 3.95% w wersji z jednym regionie oraz 1% z dwoma regionami. Wersję rozszerzoną tego algorytmu, która umożliwia dowolne zwiększanie liczby regionów aproksymacji i zmniejszanie błędu aproksymacji zaprezentowano w [10].

Inną klasą rozwiązań iteracyjnych są te oparte na algorytmie CORDIC (Coordinate Rotation Digital Computer) [11, 12], który może efektywnie być stosowany do obliczania pierwiastka kwadratowego [13]. Algorytm był szeroko stosowany do obliczania różnych funkcji arytmetycznych. Jego podstawowa zaletą jest, przy odpowiednim doborze współczynników, możliwość uniknięcia wykonywania mnożenia poprzez zastąpienie ich przesunięciami binarnymi. Podsumowanie rozwoju samego algorytmu i architektur do jego realizacji przedstawiono w [14]. Typowo jeden stopień sprzętowy przy implementacji algorytmu CORDIC wymaga cztery do ośmiu sumatorów i tej samej liczby rejestrów przesuwnych. W formie bezpośredniej CORDIC wymaga minimalnie ośmiu stopni obliczeniowych. Po wykonaniu algorytmu konieczne jest dzielenie przez stałą związaną z zwiększeniem długości wektora, wynikającą ze stosowania pseudorotacji wektora zamiast rotacji. Istotnym zagadnieniem jest też rodzaj stosowanej arytmetyki. Dla układów FPGA dostępne są tzw. rdzenie CORDIC (ang. CORDIC Cores), umożliwiające implementację algorytmu wraz z wyborem realizowanej funkcji, rodzaju arytmetyki, zakresu liczbowego i typu architektury. Przykładowo rdzeń Xilinx LogiCORE IP CORDIC v6.0 [15] może realizować rotację, translację, sinus, cosinus, sinus i cosinus hiperboliczny, arctg i pierwiastek kwadratowy, Dla pierwiastka kwadratowego rdzeń Xilinx stosuje uproszczoną formę algorytmu CORDIC, gdyż liczba pierwiastkowana jest nieujemna. Sygnały wejściowy X_{in} i wyjściowy X_{out} są nieujemne i obydwa są przedstawiane jako binarne liczby ułamkowe bez znaku lub jako liczby całkowite. Jeśli stosowany jest format ułamkowy (ang. unsigned fraction) wejściowy zakres liczbowy ograniczony jest do przedziału $0 <= X_{in} < 2$. Dla formatu liczb całkowitych bez znaku (ang. unsigned integer), X_{in} do przedziału: $0 <= X_{in} < 2^l$. gdzie l jest długością X_{in} w bitach, X_{out} jest ustawiana automatycznie na podstawie długości X_{in}.

3. ALGORYTM CORDIC

Celem opracowania algorytmu CORDIC było uzyskanie formy obliczeń, jak zaznaczono powyżej, nie wymagającej stosowania mnożenia i dzielenia przy obliczaniu standardowych funkcji arytmetycznych. Mnożenie i dzielenie są operacjami kosztownymi z punktu widzenia złożoności obliczeniowej zarówno dla realizacji programowej jak i sprzętowej. CORDIC umożliwia ich zastąpienie mnożeniem przez liczby ułamkowe będące potęgą 2, co przy realizacji binarnej wymaga tylko przesunięć w prawo. Algorytm CORDIC wykonuje rotację

wektora na płaszczyźnie, co przy jej odpowiednim sformułowaniu, pozwala uzyskać realizację pożądanej funkcji. Załóżmy, że mamy wektor $X = Re^{j\varphi}$ i chcemy dokonać obrotu tego wektora o kąt δ zgodnie z kierunkiem wskazówek zegara by uzyskać wektor $X' = Re^{j\varphi+\delta}$, uzyskujemy wtedy następującą zależność między współrzędnymi tych wektorów:

$$x' = x\cos\delta + y\sin\delta, \tag{1a}$$

$$y' = -x\sin\delta + y\cos\delta. \tag{1b}$$

Dla małych kątów δ można przyjąć $\sin\delta \approx \delta$ oraz $\cos\delta = 1$, uzyskujemy wtedy zmodyfikowane rotacje o następującej postaci:

$$x' = x + y\delta, \tag{2a}$$

$$y' = y - x\delta. \tag{2b}$$

Transformacja ta powoduje jednak zmianę długości wektora R i kąta obrotu. Otrzymujemy nowy wektor R' o długości:

$$R' = R\sqrt{1+\delta^2}. \tag{3}$$

Współrzędne nowego wektora $W = (x_w, y_w)$ można zapisać następująco:

$$x_w = \frac{x}{\sqrt{1+\delta^2}} + \frac{y\delta}{\sqrt{1+\delta^2}}, \tag{4a}$$

$$y_w = \frac{y}{\sqrt{1+\delta^2}} + \frac{x\delta}{\sqrt{1+\delta^2}}. \tag{4b}$$

Można teraz wyznaczyć kąt α, o który został obrócony wektor X w wyniku transformacji (2), mamy:

$$\sin\alpha = \frac{\delta}{\sqrt{1+\delta^2}}, \tag{5a}$$

i

$$\cos\alpha = \frac{1}{\sqrt{1+\delta^2}} \tag{5b}$$

czyli $tg\alpha = \delta$ i $\alpha = arctg\,\delta$.

Transformacje (2) mogą zostać wykorzystane do obliczenia $\sqrt{x^2+y^2}$. Może się to odbywać poprzez realizację rotacji w taki sposób, aby po pewnej liczbie kroków sprowadzić y do zera. Możemy w tym celu użyć następujących zależności:

$$x_{i+1} = x_i + y_i\delta_i, \tag{6a}$$

$$y_{i+1} = y_i - x_i\delta_i, \tag{6b}$$

oraz $\delta_i = \pm 0.5^i$, $i = 0, 1, 2, 3...,n$.

Dobór δ_i w postaci ujemnej potęgi 2 umożliwia realizację mnożenia w (6) jako binarnego przesunięcia w prawo.

Aby uzyskać w każdym kroku zmniejszenie wartości bezwzględnej y_i, znak musi być tak dobierany, aby wyrażenie $-y_i \delta_i$ miało przeciwny znak w stosunku do y_i, czyli jeśli $y_i < 0$, $\delta_i = 0.5^i$, a dla $y_i > 0$ $\delta_i = -0.5^i$.

4. WYNIKI BADAŃ EKSPERYMENTALNYCH ALGORYTMU CORDIC

Jak zaznaczono powyżej, celem prezentowanej pracy była sprzętowa realizacja algorytmu CORDIC w układzie FPGA. Opracowanie takiej realizacji wymaga przyjęcia szeregu założeń co do rodzaju arytmetyki, długości słowa wejściowego, maksymalnego błędu bezwzględnego i związanej z nim liczby iteracji algorytmu jak też formy układu. Dla rozważanej realizacji liczba iteracji przekłada się na liczbę stopni układu, która powinna być możliwie mała, co skutkuje mniejszym opóźnieniem. Przyjęto 12-bitową reprezentację ze znakiem dla sygnału wejściowego i 14-bitową ze znakiem do realizacji obliczeń. Przyjęte długości reprezentacji wynikają z przewidywanego zastosowania układu do obliczania modułu liczby zespolonej na wyjściu przepływowego procesora FFT.

Liczba koniecznych iteracji w algorytmie CORDIC wynika z dopuszczalnego błędu maksymalnego obliczenia modułu liczby zespolonej. Generalnie jako błąd dla celów praktycznych wystarczy określić jako różnicę między wartością otrzymywaną dla arytmetycznej funkcji standardowej(zwykle *sqrt*), której wartość jest obliczana przy użyciu arytmetyki zmiennoprzecinkowej i 64-bitowego typu *double* i posiada 15-16 cyfr znaczących, a wartością otrzymaną dla założonych n iteracji algorytmu CORDIC.

Maksymalny błąd bezwzględny można określić w sposób następujący:

$$e_{max} = max \left| R_f(x,y) - R(x,y)_{\mathrm{CORDIC}(n)} \right| , \quad x,y \in [1, 2^{m-1}-1] \tag{7}$$

gdzie e_{max} - wartość bezwzględna błędu maksymalnego, $R_f(x,y)$ - wartość modułu dla arytmetyki zmiennoprzecinkowej, a $R_{\mathrm{CORDIC}(n)}$ - wartość modułu otrzymana dla n iteracji algorytmu CORDIC. Określimy także błąd względny

$$e_{Relmax} = max \left| (R_f(x,y) - R(x,y)_{\mathrm{CORDIC}(n)})/R_f(x,y) \right| , \quad x,y \in [1, 2^{m-1} - 1] \tag{8}$$

Wstępnie założono, że liczba iteracji nie powinna przekraczać $n = 5$, jednak przy bezpośrednim podejściu bezwzględny błąd maksymalny, wynoszący 5.16, jest zbyt duży.

Ponieważ błąd dla pewnych regionów aproksymacji miał zbyt dużą wartość, podjęto próbę wprowadzenia korekcji wyniku w końcowej fazie obliczeń algorytmu. Podamy najpierw przykładowy sposób korekcji dla arytmetyki zmiennoprzecinkowej, a następnie rozważymy realizację algorytmu dla liczb całkowitych.

Tabela 1. Wyniki analizy błędu obliczeń modułu liczby zespolonej w arytmetyce zmiennoprzecinkowej dla algorytmu CORDIC z liczbą iteracji $n = 4,...,8$. dla wszystkich par argumentów z zakresu liczbowego $[0, 2^{11} - 1]$

n	4	5	6	7	8
e_{max}	18,73	5,16	1,36	0,35	0,09
e_{Relmax}	0,77%	0,19%	0,05%	0,01%	0,0031%
x_{maz}	1302	1692	1912	2044	2044
y_{max}	2046	2045	2038	2047	2047

(x_{maz}, y_{max}) oznacza parę argumentów, dla której wystąpił błąd maksymalny. Wyniki przedstawione w kolejnych tabelach wynikają z analizy błędu dla wszystkich możliwych par (x, y) z podanego zakresu liczbowego.

A. Korekcja dla arytmetyki zmiennoprzecinkowej

Stwierdzono na podstawie badań symulacyjnych, że wartość korekcji zależy od wartości y_5 i wartości obliczonego modułu. Z formy algorytmu CORDIC wynika, że błąd wyniku ogólnie zależy od wartości $|y_i|$, ale też od wartości modułu. Stąd przyjęte korekcje miały następującą formę: jeśli $R_e = R^{(5)} = \sqrt{x_5^2 + y_5^2} > 256$ wtedy $R_{e1}=R_e+1$ oraz jeżeli $|y_5|>128$, wtedy $R_{e2}=R_{e1}+2$. Rezultaty otrzymane w wyniki zastosowania tych korekcji przedstawiono w Tabeli 2. Wprowadzone korekcje są odpowiednio dobrane i zmniejszają błąd dla $n = 5$ i mogą być zastosowane dla obliczeń w arytmetyce zmiennoprzecinkowej.

Tabela 2. Wyniki analizy błędu obliczeń modułu liczby zespolonej w arytmetyce zmiennoprzecinkowej dla algorytmu CORDIC z liczbą iteracji $n = 5$ z wprowadzonymi korekcjami

n	5
e_{max}	2,16
e_{Relmax}	0,39%
x_{maz}	1693
y_{max}	2047

B. Korekcja dla arytmetyki całkowitej

Ponieważ projektowany układ, ze względu na wymagania odnośnie szybkości, ma działać z użyciem sygnałów kodowanych w arytmetyce z uzupełnieniem do 2, konieczna jest modyfikacja zależności (6) do formy następującej

$$x_{i+1} = x_i + \lfloor y_i 2^{-i} \rfloor, \tag{9a}$$

$$y_{i+1} = y_i - \lfloor x_i 2^{-i} \rfloor, \tag{9b}$$

Binarne przesunięcia w prawo są dzieleniem z obcięciem części ułamkowej wyniku i mogą wprowadzać istotne błędy. Wyniki obliczeń pokazano w Tabeli 3.

Tabela 3. Wyniki analizy błędu obliczeń modułu liczby zespolonej w arytmetyce liczb całkowitych dla algorytmu CORDIC z liczbą iteracji $n = 5$

n	5
e_{max}	6.42
e_{Relmax}	112.13 % ($x = 1, y = 1$)
x_{maz}	2073
y_{max}	1691

Bardzo duży błąd względny pojawia się dla małych wartości x i y natomiast błąd bezwzględny dla tych wartości nie przekracza 2. Aby zredukować bezwzględny błąd maksymalny, wprowadzono korekcję polegającą na zgrubnym szacowaniu kąta obrotu α_{cor} i dodatkowym obrocie wektora R o ten kąt. Kąt ten może być obliczony jako arctg(y_5/x_5). Przybliżone obliczenie tego kąta, jak pokazano poniżej, może być wykonane poprzez odpowiednie zaprogramowanie pamięci ROM i jej odczyt przy użyciu argumentów o zredukowanej długości. Badania symulacyjne wskazały, iż korekcja powinna zostać zmniejszona, z czego wynika jej finalna postać

$$x_{corr}=x_5 + y_5\, \alpha_{corr}/2, \tag{10}$$

Jak widać z Tabeli 4, możliwe jest uzyskanie podobnych rezultatów jak dla arytmetyki zmiennoprzecinkowej z korekcją.

Tabela 4. Wyniki analizy błędu obliczeń modułu liczby zespolonej w arytmetyce liczb całkowitych dla algorytmu CORDIC z liczbą iteracji n=5 i wprowadzoną korekcją

n	5
e_{max}	2,48
e_{Relmax}	112,13% ($x = 1, y = 1$)
x_{maz}	1279
y_{max}	1059

5. IMPLEMENTACJA UKŁADU OBLICZANIA MODUŁU LICZBY ZESPOLONEJ

Układ obliczający moduł liczby zespolonej zrealizowano w oparciu o przedstawiony powyżej zmodyfikowany algorytm CORDIC.

W pierwszym stopniu układu, ponieważ $\delta_0 = 1$, wykonywane są działania:

$$x_1= x_0 + y_0\,, \tag{11a}$$

$$y_1 = y_0 - x_0. \tag{11b}$$

Dodawanie w (11a) wykonywane jest na 11-bitowych reprezentacjach bez znaku, a w (11b) w kodzie U2. Reprezentacja $-x_0$ w U2 jest tworzona poprzez negację x_0 i wprowadzenie 1 jako przeniesienia do najmłodszej pozycji sumatora BA1.

W drugim stopniu wykonywane są następujące działania:

$$x_2 = x_1 + y_1 >> 1\,, \tag{12a}$$

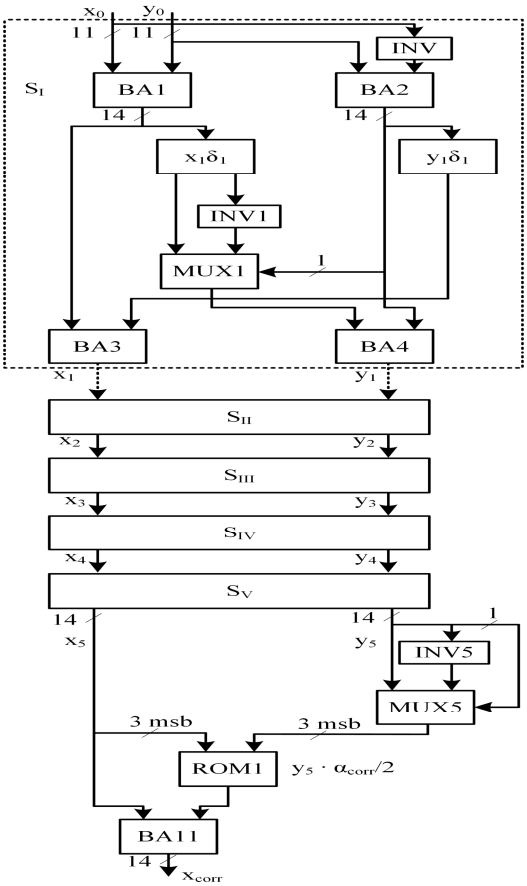
oraz jeśli $y_1 >= 0$

$$y_2 = y_1 - x_1 >> 1, \tag{12b}$$

lub dla $y_1 < 0$

$$y_2 = y_1 + x_1 >> 1, \tag{12c}$$

gdzie >> 1 oznacza binarne przesunięcie w prawo o 1 bit. Przesunięcie to jest realizowane poprzez odpowiednie przekierowanie bitów z odrzuceniem najmłodszego bitu. Analogiczne działania są wykonywane w kolejnych stopniach układu.



Rys. 1. Architektura układu obliczania modułu liczby zespolonej w oparciu o algorytm CORDIC

Na rys. 1 pokazano architekturę układu. W bloku S_I sumator BA1 wykonuje dodawanie $x_0 + y_0$, a sumator BA2 odejmowanie w kodzie U2, blok INV1 wykonuje negację, a dodawanie 1 jest realizowane poprzez wprowadzenie 1 jako przeniesienia do najmłodszej pozycji sumatora. Przesunięcia w prawo o 1 bit dla x_1 i y_1 symbolizują bloki x_i δ_i. Bloki S_{II},S_{III},S_{IV} i S_V mają tę samą strukturę jak S_I. Obliczane są zarówno $x_i\delta_i$ jak i $-x_i\delta_i$ w kodzie U2. Wartości te są multipleksowane przy użyciu multipleksera MUX1 sterowanego znakiem y_i, tak aby dla każdego znaku y_i, uzyskać redukcję wartości absolutnej $|y_i,|$. Finalna korekcja jest realizowana przy użyciu ROM1 adresowanego trzema bitami msb x_5 i y_5. Użycie tylko trzech bitów pozwala na zastosowanie pamięci o 6-bitowym adresie. Dodatkowa rotacja x_5 wymaga obliczenia $\sin(\alpha_{cor})$, $\alpha_{corr}=(\text{arctg}(\lfloor y_5/2^3 \rfloor 2^3\}/(\lfloor x_5/2^8 \rfloor 2^8))$, jednak ze względu na małą wartość kąta można przyjąć $\sin(\alpha_{corr}) = \alpha_{corr}$. Poniżej przedstawiono wyniki syntezy architektury z rys. 1 przy zastosowaniu układu XilinxVirtex-6 FPGA 6vlx240tff784-2w[16].

```
=========================
Top Level Output File Name : cordic1.ngc
Primitive and Black Box Usage:
------------------------------
# BELS : 547
# GND : 1
# INV : 15
# LUT1 : 18
# LUT2 : 6
# LUT3 : 146
# LUT5 : 1
# LUT6 : 17
# MUXCY : 169
# XORCY : 174
#      IO Buffers : 37
# IBUF : 24
# OBUF : 13
Device utilization summary:
---------------------------
Selected Device : 6vlx240tff784-2

Slice Logic Utilization:
Number of Slice LUTs:                  203 out of 150720 0%
Number used as Logic:                  203 out of 150720 0%

Slice Logic Distribution:
     Number of LUT Flip Flop pairs used:       203
     Number with an unused Flip Flop:          203 out of 203 100%
Number with an unused LUT:             0 out of 203 0%
 Number of fully used LUT-FF pairs:          0 out of 203 0%
 Number of unique control sets:              0
IO Utilization:
 Number of IOs:                        39
 Number of bonded IOBs:           37 out of 400 9%

Total delay 10.035ns (4.055 ns logic, 5.980 ns route)
```

Rys. 2. Wyniki syntezy układu do obliczania modułu liczby zespolonej w środowisku Xilinx

6. PODSUMOWANIE

W pracy przedstawiono obliczanie modułu liczby zespolonej w układzie FPGA przy użyciu zmodyfikowanej formy algorytmu CORDIC dla argumentów 11-bitowych. Głównym celem pracy była redukcja liczby stopni algorytmu do 5, co umożliwia redukcję złożoności sprzętowej jak i opóźnienia. Ze względu na fakt, iż zmniejszanie liczby iteracji a algorytmie CORDIC zwykle powoduje wzrost błędu maksymalnego, zaproponowano korekcję na wyjściu umożliwiającą redukcję błędu maksymalnego do 2.48. Błąd ten występuje dla względnie dużych wartości argumentów powyżej 1000. Dla małych wartości argumentów błąd bezwzględny nie przekracza 2. Opracowany algorytm może być stosowany dla argumentów dłuższych niż 11-bitowe, lecz wymaga to użycia dodatkowych stopni w układzie.

LITERATURA

[1] Kwon T., Sondeen J., Draper J.: Floating-point division and square root using a Taylor-series expansion algorithm. In 50th Midwest Symposium on Circuits and Systems, MWSCAS 2007, pp. 305 – 308, 2007.

[2] Kosheleva O.: Babylonian method of computing the square root: Justifications based on fuzzy techniques and on computational complexity. In Fuzzy Information Processing Society, NAFIPS 2009, pp. 1– 6, 2009.

[3] Ercegovac M., D.: On Digit-by-Digit Methods for Computing Certain Functions. In Conference Record of the 41th Asilomar Conference on Signals, Systems and Computers, ACSSC 2007, pp. 338 – 342, 2007.

[4] Montuschi P., Mezzalama M.: Survey of square rooting algorithms. Comput. Digit. Tech. IEE Proc. E, vol. 137, no. 1, pp. 31– 40, Jan. 1990.

[5] Sutikno T.: An efficient implementation of the nonrestoring square root algorithm in gate level. Int. Journal Comput. Theory Eng., vol. 3, no. 1, pp. 46 – 51, 2011.

[6] Sutikno T., Jidin Z.: Simplified VHDL Coding of modified nonrestoring square root calculator. Int. J. Reconfigurable Embed. Syst., vol. 1, no. 1, pp. 37– 42, 2012.

[7] Sajid Ahmed M., Ziavras S. G.: Pipelined implementation of fixed point square root in FPGA using modified non-restoring algorithm. In 2010 2nd International Conference on Computer and Automation Engineering (ICCAE), vol. 3, pp. 226–230. 2010.

[8] Kabuo H., Taniguchi T., Miyoshi A., Yamashita H., Urano M., Edamatsu H., Kuninobu S.: Accurate rounding scheme for the Newton-Raphson method using redundant binary representation. IEEE Trans. Comput., vol. 43, no. 1, pp. 43–51, Jan. 1994.

[9] Filip A. E: Linear approximations to sqrt(x^2+y^2) having equiripple error characteristics. IEEE Trans. Audio Electroacoustics, vol. 21, no. 6, pp. 554–556, Dec. 1973.

[10] Czyżak M., Smyk R.: FPGA realization of an improved alpha max plus beta min algorithm. Poznan University of Technology Academic Journals Electrical Engineering, vol. 80, pp.151 – 160, 2014.
[11] Volder J.E.: The CORDIC Trigonometric Technique: IRE Transactions on Electronic Computers, pp. 330-334, Sept. 1959.
[12] Walther J.S.: A unified algorithm for elementary functions. In Proc. of Sprint Joint Computer Conference, pp. 379–385, May 1971.
[13] Ye M., Liu T., Ye Y., Xu G., Xu T.: FPGA Implementation of CORDIC-Based Square Root Operation for Parameter Extraction of Digital Pre-Distortion for Power Amplifiers. In 2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM), pp. 1– 4, 2010.
[14] Meher P., K., Vallis J., Tso-Bing Juang, Sridharan K., Maharanta K.: 50 Years of CORDIC: Algorithms, Architectures, and Applications. IEEE Trans. Circuits Syst. Regul. Pap., vol. 56, no. 9, pp. 1893 – 1907, Sept. 2009.
[15] Xilinx: LogiCORE IP CORDIC v4.0. Product specification.. www.xilinx.com March 2011.
[16] Xilinx:Virtex-6. www.xilinx.com/products/silicon-devices/fpga/virtex-6.html, Feb. 2015.

COMPUTATION OF MAGNITUDE OF COMPLEX NUMBER IN FPGA USING CORDIC

The work presents computation of the magnitude of complex numbers with a modified version of the CORDIC algorithm using five iteration steps. A relationship between the error and the number of CORDIC iterations for floating point arithmetic was examined as well as the impact of using the integer arithmetic. The proposed modification of the algorithm for integer arithmetic relies upon the introduction of a correction after performing the assumed number CORDIC iterations The correction value is established upon the approximate quotient of coordinates obtained after the fifth iteration step. Such correction allows to reduce the maximum error approximately by half. The architecture implementing the algorithm in the FPGA is also shown.