POZNAN UNIVERSITY OF TECHNOLOGY ACADEMIC JOURNALS
No 87 Electrical Engineering 2016

Robert SMYK*
Maciej CZYŻAK*

REALIZACJA NA POZIOMIE RTL OBLICZANIA PIERWIASTKA KWADRATOWEGO Z UŻYCIEM METODY NIEODTWARZAJĄCEJ

Obliczanie pierwiastka kwadratowego jest jedną z kluczowych operacji cyfrowego przetwarzania sygnałów szczególnie przy obliczaniu modułu sygnałów zespolonych. W pracy przedstawiono algorytm obliczania pierwiastka kwadratowego metodą nieodtwarzającą oraz jego układową realizację. Metoda umożliwia oszczędną realizację układową bazującą na sumatorach i rejestrach. Przeanalizowano wymagania sprzętowe obliczania pierwiastka kwadratowego dla operandów 8-, 16- i 32-bitowych. Przedstawiono implementację w VHDL oraz wynik syntezy układu dla wybranych wariantów w środowisku Altera Quartus II FPGA.

SŁOWA KLUCZOWE: FPGA, moduł sygnałów zespolonych, metoda nieodtwarzająca, pierwiastek kwadratowy

1. WSTĘP

W wielu aplikacjach cyfrowego przetwarzania sygnałów (CPS) istnieje potrzeba zastosowania efektywnego algorytmu obliczania pierwiastka kwadratowego (PK). Typowym przykładem jest obliczanie modułu sygnału zespolonego na wyjściu procesora obliczającego szybką transformatę Fouriera. Z racji tego, że sygnały w rzeczywistych aplikacjach CPS przetwarzane są z zachowaniem restrykcyjnie nałożonych ograniczeń takich jak możliwie małe opóźnienie przy zachowaniu wysokiej częstotliwości potokowania, od układu cyfrowego realizującego opisywany algorytm oczekuje się spełnienia ściśle określonych wymagań dotyczących parametrów czasowych.

W literaturze znane są algorytmy iteracyjne obliczania PK stanowiące rozwinięcie metody Newtona-Raphsona (NR) [1]. W algorytmach tego typu w kolejnych iteracjach obliczane są kolejne przybliżenia PK. Głównym ograniczeniem tej klasy algorytmów jest silna zależność dokładności obliczeń od liczby iteracji, która może być teoretycznie dowolna, oraz od przyjętego punktu

* Politechnika Gdańska.

startowego. Choć spotykane są przykłady implementacji układowych metody NR, jest to algorytm iteracyjny częściej stosowany w postaci implementacji programowych. W przypadku implementacji układowej metody NR można założyć ustaloną liczbę iteracji i zrealizować każdą z nich w postaci pojedynczego stopnia obliczeniowego. Tego typu podejście powoduje jednak niewspółmierny wzrost złożoności układowej w przypadku konieczności uzyskania założonej dokładności obliczeń. Dodatkowo poważnym ograniczeniem w implementacji układowej tej metody jest potrzeba wykonywania mnożenia oraz dzielenia. Podsumowując, metoda NR ze względu na przedstawione właściwości nie jest w stanie spełnić wymagań, takich jak dokładność oraz szybkość obliczeń, stawianych przez współczesne aplikacje CPS.

Inną klasą algorytmów PK opiera się na algorytmie CORDIC [2-4], który powszechnie wykorzystywany jest w technikach CPS do realizacji funkcji trygonometrycznych. Algorytm ten należy do klasy iteracyjnych. W praktyce obliczenie pierwiastka kwadratowego wymaga przeprowadzenia do 10 iteracji. W każdej iteracji wykonywane jest sumowanie oraz mnożenie przez potęgę liczby 2. Zasadniczą cechą algorytmu CORDIC jest to, że przy realizacji układowej nie wymaga on stosowania mnożników, przy doborze współczynników w postaci ujemnych potęg 2 zastępowane są one przez przesunięcia binarne. Za pewną wadę można uznać konieczność dzielenia wyniku operacji przez wartość stałą ze względu na wykonywanie w każdej iteracji tzw. pseudorotacji wektora – wydłużającej jego długość. Warto wspomnieć, że producenci układów FPGA często oferują specjalizowany IP Core realizujący algorytm CORDIC [5]. W praktyce, w przypadku aplikacji CPS implementowanych w FPGA czynnik ten może decydować o wyborze algorytmu PK bazującego na algorytmie CORDIC.

Znane są również nieiteracyjne metody obliczania PK [6]. W realizacji układowej bazującej na wykorzystaniu algorytmu alpha max plus beta min możliwe jest bezpośrednie wyznaczenie modułu $\sqrt{a^2+b^2}$ liczby zespolonej $a+j\cdot b$. Układy wykorzystujące tego typu podejście [7] charakteryzują się, w odróżnieniu od bazujących na metodach iteracyjnych, z góry ustaloną na poziomie algorytmu liczbą stopni obliczeniowych oraz prawie dowolną do ustalenia dokładnością na poziomie poniżej 1%.

W niniejszej pracy przedstawiono algorytm oraz właściwości implementacji układowej na poziomie RTL (ang. Register Transfer Level) metody nieodtwarzającej obliczania PK [8-11]. Algorytm ten wykorzystuje stałą liczbę iteracji, a jego realizację układową można zrealizować tylko przy wykorzystaniu sumatorów oraz rejestrów. W podrozdziale 2 przedstawiono algorytm obliczania PK metoda nieodtwarzającą, a w podrozdziale 3 przedstawiono jej implementację układową.

2. METODA NIEODTWARZAJĄCA OBLICZANIA PIERWIASTKA KWADRATOWEGO

Przy obliczaniu PK metodą nieodtwarzającą wykorzystuje się reprezentację operandów w kodzie uzupełnienia do 2. Algorytm ten w każdej iteracji generuje przybliżoną wartość wyniku (rys. 1). Jako wynik otrzymuje się wartość całkowitą pierwiastka Q oraz resztę $R = X - Q \cdot Q$. W zależności od znaku R w kolejnych iteracjach odtwarza się kolejny bit wartości pierwiastka Q, poprzez dodanie lub odejmowanie 1 na odpowiedniej pozycji. Algorytm ten w każdej iteracji wyznacza wartość pierwiastka bez odtwarzania wspomnianej reszty [11]. Zgodnie z przedstawionym schematem, liczba iteracji głównej pętli obliczeniowej zależna jest od długości reprezentacji binarnej liczby podpierwiastkowej X. Zakładając, że X jest liczbą n-bitową, pierwiastek kwadratowy z X można obliczyć w $n/2$ krokach.

W celu bardziej szczegółowej analizy kroków obliczeniowych rozważanego algorytmu, załóżmy, że liczba podpierwiastkowa X ma reprezentację 16-bitową $X = (x_{15}, x_{14}, ..., x_1, x_0)$. Wartość całkowita Q może być reprezentowana przez liczbę 8-bitową $Q = (q_7, q_6, ..., q_1, q_0)$, a reszta $R = X - Q^2$ jest liczbą co najwyżej 9-bitową o reprezentacji $\mathcal{R} = (r_8, r_7, ..., r_1, r_0)$. W algorytmie przyjmuje się, że na każdą parę bitów liczby podpierwiastkowej przypada jeden bit liczby reprezentującej wartość pierwiastka. Obliczenia w pierwszym kroku prowadzi się dla pary $x_{15}x_{14}$. Najstarszy bit wartości pierwiastka p_7 przyjmuje wartość 0, gdy $(x_{15}, x_{14}) = (0,0)$ lub 1, gdy odpowiednio $(x_{15}, x_{14}) = (0,1)$, $(x_{15}, x_{14}) = (1,0)$, $(x_{15}, x_{14}) = (1,1)$. W ogólności wartość q_7 można obliczyć jako różnicę $x_{15}2^1 + x_{14}2^0 - 1$. Jeżeli różnica jest ujemna, to $q_7 = 0$, a w przeciwnym przypadku $q_7 = 1$. Wartość reszty w kroku pierwszym dla k = 7 oblicza się poprzez wykonanie operacji $R_7 = x_{15}2^1 + x_{14}2^0 - q_7 2^0 \cdot q_7 2^0$, gdzie R_7 oznacza resztę uzyskaną przy wyznaczaniu 7 bitu pierwiastka Q. W efekcie R może przyjąć jedną z trzech wartości 0,1 lub 2.

W kolejnym kroku dla pary bitów (x_{13}, x_{12}) nowa wartość reszty dla k=6 wynosi $R_6 = R_7 2^2 + x_{13}2^1 + x_{12}2^0$ i obliczana jest jako konkatenacja R_7 i bitów $\{x_{13}, x_{12}\}$. Dalej wyznaczany jest szósty bit q_6 pierwiastka Q dla fragmentu liczby podpierwiastkowej X o wartości $\sum_{i=12}^{15} x_i 2^{i-12}$. Ponieważ

$$\sum_{i=12}^{15} x_i 2^{i-12} \geq \left(q_7 2^1 + q_6 2^0\right)^2 \tag{1}$$

można wyprowadzić ogólną formułę wyznaczania kolejnego bitu pierwiastka Q. Obliczając prawą stronę (1) otrzymujemy

$$(2 \cdot q_7 + q_6) \cdot (2 \cdot q_7 + q_6) = 4 \cdot q_7 \cdot q_7 + 4 \cdot q_7 \cdot q_6 + q_6 \cdot q_6 . \quad (2)$$

Korzystając z (1) i (2) uzyskujemy

$$R_7 2^2 + x_{13} 2^1 + x_{12} 2^0 \geq 4 \cdot q_7 \cdot q_6 + q_6 \cdot q_6 . \quad (3)$$

Zależność ta może być zweryfikowana przez podstawienie $R_7 = x_{15} 2^1 + x_{14} 2^0 - q_7 2^0 \cdot q_7 2^0$.

Uogólniając powyższą procedurę dla kolejnych kroków obliczeniowych można uzyskać formułę wyznaczania reszty dla k-1 w postaci

$$R_{k-1} = R_k^{sh2} - Q_k^{neg\ sh2}, \quad (4)$$

gdy $R_{k-1} \geq 0$,

gdzie $R_k^{sh2} \Leftrightarrow \mathcal{R}_k^{sh2} = (\mathcal{R}_k, x_{2k-1}, x_{2k-2})$

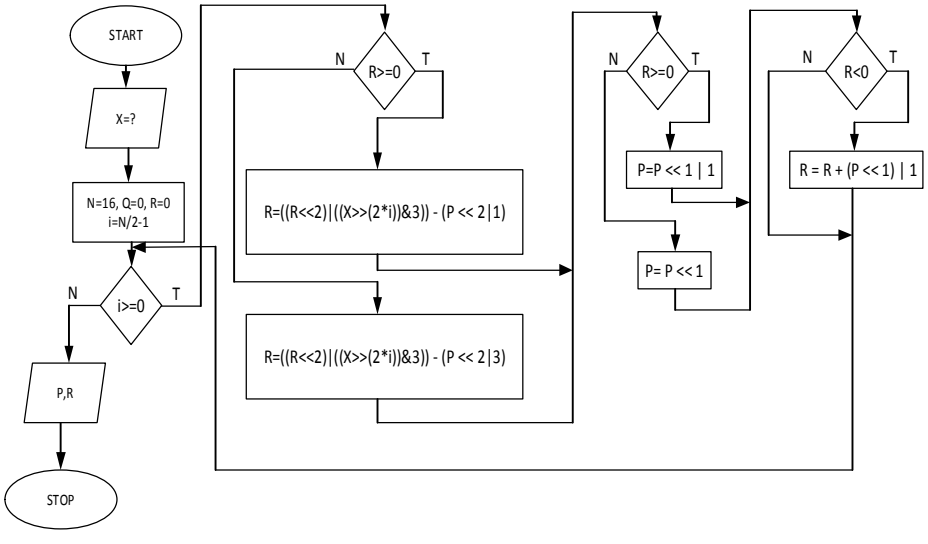
oraz $Q_k^{neg\ sh2} \Leftrightarrow \mathcal{Q}_k^{neg\ sh2} = (Q_k, 0, 1)$

Natomiast dla $R_{k-1} < 0$

$$R_{k-1} = R_k^{sh2} + Q_k^{neg\ sh2}, \quad (5)$$

gdzie $R_k^{sh2} \Leftrightarrow \mathcal{R}_k^{sh2} = (\mathcal{R}_k, x_{2k-1}, x_{2k-2})$

oraz $Q_k^{neg\ sh2} \Leftrightarrow \mathcal{Q}_k^{neg\ sh2} = (Q_k, 1, 1)$.



Rys. 1. Schemat blokowy programowego algorytmu metody nieodtwarzającej obliczania PK z X

W celu analizy właściwości numerycznych przedstawionego algorytmu wykonano jego symulację programową w języku Python. Listing opracowanego programu przedstawiono na rysunku 2. Na rysunku 3 zaprezentowano porównanie przykładowych wyników otrzymanych dla omawianego algorytmu PK z wynikami, które daje funkcja biblioteczna *sqrt*(). Jak widać, wartości reszty dla *X=280* i *X=400* różnią się o 1 od wyników z funkcji bibliotecznej. Może to wynikać z wewnętrznej reprezentacji liczb całkowitych, która jest zawsze stała dla przyjętego typu danych i jest różna od założonej długości słowa *n*.

```
def nonrest(x, N):
    """
    implementacja algorytmu non-restoring
    x - liczba podpierwiastkowa
    q - quotient (pierwiastek)
    r - reszta
    N - zakres wejscia (liczba bitow, 8,16,32,...,2^n)
    """
    q = 0
    r = 0
    for i in range(N/2-1,-1,-1):
        if r>=0:
            r=((r<<2)|((x>>(2*i))&3)) - (q <<2|1)
        else:
            r=((r<<2)|((x>>(2*i))&3)) + (q << 2|3)
        if r>=0:
            q=q << 1 | 1
        else:
            q= q << 1
        if r<0:
            r = r + (q << 1)|1
    return (q,r)
```

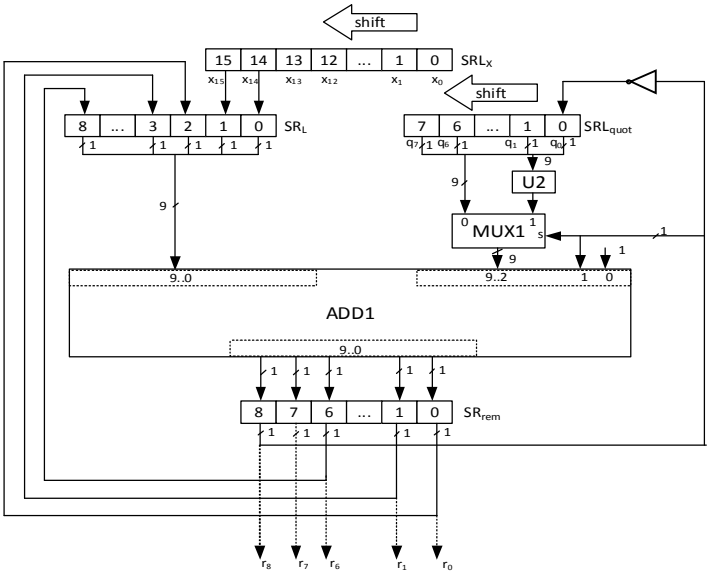
Rys. 2. Implementacja programowa w Python algorytmu metody nieodtwarzającej obliczania PK

podpier X	pierwiastek (non-rest)	reszta (non-rest)	pierwiastek (float->int)	reszta (float->int)
x= 220	q= 14	r= 23	sqrt(x)= 14	rem(x)= 24
x= 240	q= 15	r= 15	sqrt(x)= 15	rem(x)= 15
x= 260	q= 16	r= 3	sqrt(x)= 16	rem(x)= 4
x= 280	q= 16	r= 23	sqrt(x)= 16	rem(x)= 24
x= 300	q= 17	r= 11	sqrt(x)= 17	rem(x)= 11
x= 320	q= 17	r= 31	sqrt(x)= 17	rem(x)= 31
x= 340	q= 18	r= 15	sqrt(x)= 18	rem(x)= 16
x= 360	q= 18	r= 35	sqrt(x)= 18	rem(x)= 36
x= 380	q= 19	r= 19	sqrt(x)= 19	rem(x)= 19
x= 400	q= 20	r= -1	sqrt(x)= 20	rem(x)= 0
x= 420	q= 20	r= 19	sqrt(x)= 20	rem(x)= 20
x= 440	q= 20	r= 39	sqrt(x)= 20	rem(x)= 40

Rys. 3. Przykładowe wyniki zwracane przez program realizujący algorytm metody nieodtwarzającej obliczania PK

3. IMPLEMENTACJA UKŁADOWA METODY NIEODTWARZAJĄCEJ OBLICZANIA PIERWIASTKA KAWADRATOWEGO

Ogólna idea przy realizacji układowej algorytmu metody nieodtwarzającej PK polega na wykorzystaniu jednostki obliczającej z odpowiednimi rejestrami. W przedstawionej strukturze na rysunku 4 dla *n*-bitowego słowa wejściowego *X* wykorzystano *n+2*-bitowy sumator ADD1 realizujący dodawanie lub odejmowanie w kodzie U2. Wartość liczby podpierwiastkowej podawana jest do rejestru SRL_X przesuwającego w lewo o 2 pozycje. Na wejściach sumatora znajdują się rejestry: SR_L – pomocniczy rejestr *n+2*-bitowy, przechowujący bieżąca wartość reszty z poprzedniego kroku uwzględnianą przy sumowaniu oraz SRL_{quot} – rejestr przesuwający w lewo przechowujący bieżącą wartość pierwiastka. Na wyjściu sumatora znajduje się rejestr SR_{rem} przechowujący aktualną wartość reszty. Najstarszy bit bieżącej reszty określa jej znak. Informacja ta jest wykorzystywana do określania znaku bieżącej wartości pierwiastka podczas sumowania w aktualnie realizowanym kroku (MUX1 i U2).



Rys. 4. Schemat blokowy implementacji układowej algorytmu metody nieodtwarzającej obliczania PK dla zakresu n = 16

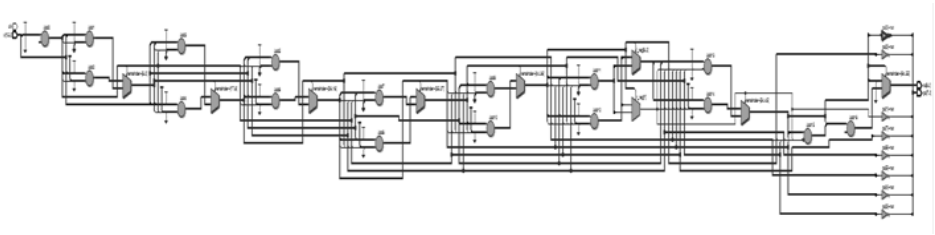
Na rysunku 5 przedstawiono opis w języku VHDL układu realizującgo algorytm obliczania PK metodą nieodtwarzającą. Kod odzwierciedla strukturę układu opisaną wyżej. Przedstawione podejście pozwala na realizację struktur ukła-

dowych dla zakresów *n*-bitowych słowa wejściowego *X*, gdzie $n = 8,16,32,...,2^k$ dla $k = 3,4,5,...$. Zakres ten można podawać bezpośrednio w opisie VHDL poprzez nadanie wartości zmiennej *n* przed wykonaniem syntezy. Dodatkowo poprzez modyfikację parametrów syntezy takiego układu można uzyskać dwie różne implementacje układowe: jedną w postaci struktury drzewiastej (rys. 6), drugą w postaci pojedynczej jednostki przetwarzającej (rys. 7) o konstrukcji zbliżonej do ALU (ang. arithmetic logic unit).

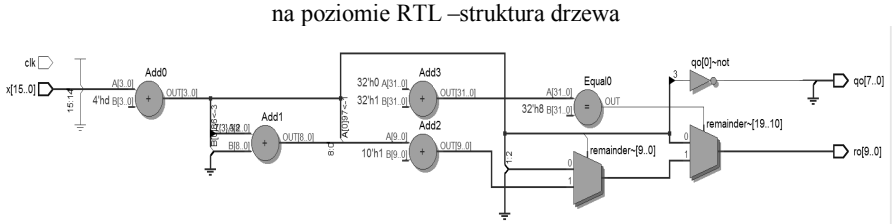
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity nonrest_n_bit is
generic ( n: positive:= 16 );
    Port ( clk : std_logic;
           x : in  STD_LOGIC_VECTOR ((n-1) downto 0);
           qo : out  STD_LOGIC_VECTOR ((n/2-1) downto 0);
           ro : out  STD_LOGIC_VECTOR ((n/2+1) downto 0));
end nonrest_n_bit;
architecture Behavioral of nonrest_n_bit is
begin
process(clk) is
variable x_reg : unsigned((n-1) downto 0) :=(others => '0');  --rejestr reprezentujacy wejscie
variable quotient : unsigned((n/2-1) downto 0):=(others => '0');  --pierwiastek
variable l_reg,r_reg : unsigned((n/2+1) downto 0):=(others => '0');  --rejestry sumatora
variable remainder : signed((n/2+1) downto 0):=(others => '0'); -- reszta
variable i : integer:=0;
begin
   x_reg :=unsigned(x);--rejestr wartosci podpierwiastkowej
   quotient :=(others => '0');
   remainder :=(others => '0');
   l_reg :=(others => '0');
   r_reg :=(others => '0');
   for i in 0 to (n/2-1) loop
      r_reg((n/2+1) downto 0):=quotient&remainder((n/2+1))&'1';
      l_reg((n/2+1) downto 0):=unsigned(remainder((n/2-1) downto 0))&x_reg((n-1) downto (n-2));
      x_reg((n-1) downto 2):=x_reg((n-3) downto 0); --x_reg << 2
      if (remainder((n/2+1))='1') then
         remainder := signed(l_reg + r_reg);
      else
         remainder := signed(l_reg - r_reg);
      end if;
      quotient((n/2-1) downto 0) := quotient((n/2-2) downto 0) &(not remainder((n/2+1)));
   end loop;
   qo <= std_logic_vector(quotient);
   if(remainder((n/2+1))='1') then --korekcja dla reszt ujemnych
      quotient((n/2-1) downto 1) := quotient((n/2-2) downto 0);
      quotient(0) := '0';
      remainder := remainder + signed(quotient) + 1;
   end if;
   ro <= std_logic_vector(remainder);
end process;
end Behavioral;
```

Rys. 5. Opis w VHDL układowej *n*-bitowej implementacji algorytmu metody nieodtwarzającej obliczania PK

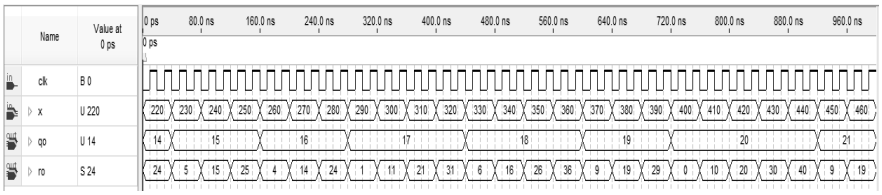
Omawianą realizację układową PK algorytmu metody nieodtwarzającej zasymulowano oraz poddano syntezie w środowisku FPGA Altera Quartus II dla układu serii Cyclone V [12]. Przykładowe wyniki symulacji potwierdzające poprawność pracy układu w wersji 16-bitowej o strukturze drzewiastej przedstawiono na rysunku 8. W tabeli 1 przedstawiono wyniki syntezy układów w konfiguracjach dla zakresów 8-,16- i 32-bitowych.



Rys. 6. Układowa implementacja algorytmu metody nieodtwarzającej obliczania PK na poziomie RTL –struktura drzewa



Rys. 7. Układowa implementacja algorytmu metody nieodtwarzającej obliczania PK na poziomie RTL – struktura z ALU



Rys. 8. Wynik symulacji układowej implementacji algorytmu metody nieodtwarzającej obliczania PK (*x* – liczba podpierwiastkowa, *qo* – wartość całkowita pierwiastka, *ro* – wartość całkowita reszty)

Tabela 1. Wyniki syntezy w Altera Quartus II (układ FPGA Cyclone V) układowej implementacji algorytmu metody nieodtwarzającej obliczania PK dla wariantu 8-, 16 i 32- bitowego

Square rooter input x [no. bits]	8-bit	16-bit	32-bit
Estimate of Logic utilization (ALMs needed)	21	66	246
Combinational ALUT usage for logic	41	132	492
-- 7 input functions	0	0	0
-- 6 input functions	0	0	0
-- 5 input functions	10	20	52
-- 4 input functions	12	21	152
-- <=3 input function	19	91	288
I/O pins	19	35	67
Maximum fan-out	21	33	71
Total fan-out	189	536	2079
Average fan-out	2.39	2.65	3.32

4. PODSUMOWANIE

W pracy zrealizowano algorytm obliczania pierwiastka kwadratowego metodą nieodtwarzającą. Przedstawiono jego podstawowe własności numeryczne oraz wyniki symulacji programowej. Algorytm działa na operandach całkowitych i wykorzystuje stałą liczbę iteracji zależną od długości słowa wejściowego. W wyniku działania algorytmu uzyskuje się wartość pierwiastka oraz wartość reszty.

Przedstawiony algorytm pozwala na realizację układową wykorzystującą tylko jeden sumator w jednym stopniu dla wersji z pojedynczą jednostką przetwarzającą lub *n/2* sumatorów dla struktury drzewiastej. Przedstawiono wyniki implementacji układowej dla przykładowych zakresów n=8-, 16-, 32-bitowych. Opis układu wykonano w VHDL oraz przeprowadzono syntezy w środowisku Altera Quartus II dla trzech wybranych wariantów. Opracowany opis układu w VHDL, przedstawiony w tej pracy, ma charakter ogólny z dowolnie ustalanym zakresem bitowym n.

BIBLIOGRAFIA

[1] 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.

[2] Volder J.E.: The CORDIC Trigonometric Technique: IRE Transactions on Electronic Computers, pp. 330-334, Sept. 1959.

[3] Meher P., K., Vallis J., Tso-Bing Juang, Sridharan K., Maharanta K.: 50 Years of CORDIC: Algorithms, Architectures, and Applications, IEEE Trans. Circuits Syst. Regular Papers, vol. 56, no. 9, pp. 1893 – 1907, Sept. 2009.

[4] 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 Proc. of 2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM), pp. 1– 4, 2010.

[5] Xilinx: LogiCORE IP CORDIC v4.0. Product specification. www.xilinx.com March 2011.

[6] 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.

[7] 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.

[8] 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.

[9] 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.
[10] Li Y., Chu W.: Implementation of Single Precision Floating Point Square Root on FPGAs, Proc. of 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp.227-232, 1997.
[11] Li Y., Chu W.: A New Non-Restoring Square Root Algorithm and Its VLSI Implementations, Proc. on 1996 IEEE International Conference on Computer Design: VLSI in Computers and Processors, ICCD '96, pp. 538 – 544, 1996.
[12] Altera, Overview Cyclone V FPGA & SoC, www.altera.com , 01.2016.

IMPLEMENTATION AT THE RTL LEVEL OF SQUARE ROOTING WITH A USE OF NON-RESTORING METHOD

Computation of square root is the crucial operation in digital signal processing, especially when computing the modulus of complex signals. In this work we present the square rooting algorithm using non-restoring method and its implementation at the RTL level. The method allows for compact realization that uses adders and registers only. The hardware requirements for square rooting for 8-, 16- and 32-bit operand have been analyzed. An VHDL implementation has been presented as well as the results of synthesis for the chosen variants in Altera Quartus II environment.

(Received: 17. 02. 2016, revised: 5. 03. 2016)