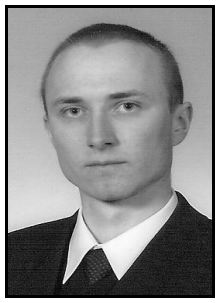
Maciej WIELGOSZ, Ernest JAMRO, Kazimierz WIATR
AKADEMIA GÓRNICZO-HUTNICZA, ACK - CYFRONET
AKADEMIA GÓRNICZO-HUTNICZA, KATEDRA ELEKTRONIKI

Moduł obliczający funkcję eksponenty implementowanej w układach FPGA

Mgr inż. Maciej WIELGOSZ



Ukończył studia na AGH (2005), na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki na kierunku Elektronika i Telekomunikacja. Obecnie jest doktorantem w Katedrze Elektroniki AGH i bierze czynny udział w pracach badawczych realizowanych w zespole rekonfigurowalnych systemów obliczeniowych. Jego zainteresowania naukowe dotyczą sprzętowej akceleracji obliczeń, kompresji obrazu i sieci neuronowych.

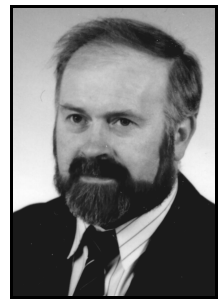
e-mail: wielgosz@agh.edu.pl

Dr inż. Ernest JAMRO

Ukończył studia na AGH na kierunku Elektronika oraz na University of Huddersfield (UK) na kierunku Elektronika i Telekomunikacja. Obronił pracę doktorską w 2001 roku na AGH na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki. Aktualnie jest adiunktem w Katedrze Elektroniki na AGH. Jego zainteresowania naukowe to sprzętowa akceleracja obliczeń, niskopoziomowe przetwarzanie obrazów, sieci neuronowe.

e-mail: jamro@agh.edu.pl

Prof. dr hab. inż. Kazimierz WIATR



Studia AGH Kraków (1980), dr nauk technicznych (1987), dr habilitowany (1999) i profesor (2002). Profesor zwyczajny na Akademii Górniczo-Hutniczej oraz Dyrektor Akademickiego Centrum Komputerowego Cyfronet AGH. Prowadzone prace badawcze dotyczą komputerowego sterowania procesami, systemów wizyjnych, systemów wieloprocesorowych, układów programowalnych, rekonfigurowalnych systemów obliczeniowych i sprzętowych metod akceleracji obliczeń.

e-mail: wiatr@agh.edu.pl

Streszczenie

Niniejszy artykuł prezentuje implementację operacji obliczania eksponenty o podwójnej precyzji obliczeń w układach FPGA. Zaproponowano metodę tablicowo – aproksymacyjną, dla której wykorzystano 3 niezależne tablice 512×64-bity do obliczenia 27 najstarszych bitów mantysy oraz aproksymacje wielomianową ex≈1+x dla pozostałych bitów mantysy. Wyniki implementacji pokazują że proponowany moduł zajmuje około 7.5% układu Virtex-4 LX200.

Słowa kluczowe: HPC, exp(), FPGA.

FPGA Implementation of Exponent Function

Abstract

This paper presents FPGA implementation of exponent operation in double precision format. A mixture of Look-Up Table (LUT) and approximation methods was employed. Twenty seven most significant bits of input mantissa are calculated employing 3 independent LUTs, the rest input bits are calculated by approximation: ex≈1+x. Implementation results in roughly 7.5% occupation of Virtex-4 LX-200.

Keywords: HPC, exp(), FPGA.

1. Wstęp

Numeryczna analiza skomplikowanych modeli fizycznych oraz chemicznych realizowana np. w pakiecie Gaussian wymaga dużej mocy obliczeniowej. Na szczególną uwagę zasługuje tutaj operacja eksponenty, która jest powszechnie wykonywana natomiast nie jest bezpośrednio wspierana przez procesory ogólnego przeznaczania. Dlatego kluczowym celem, jaki postawili sobie autorzy niniejszej pracy było wyraźne przyspieszenie realizacji operacji eksponenty dzięki dedykowanej a przez to wydajnej architekturze sprzętowej realizowanej w układach programowalnych FPGA.

Operacje zmiennoprzecinkowe o podwójnej precyzji wymagają znacznych zasobów układu FPGA i dlatego do niedawna układy FPGA nie były wykorzystywane do wspomagania obliczeń numerycznych pełnej precyzji. Niestety kryterium odgrywającym ogromną rolę we współczesnych systemach HPC jest dokładność obliczeń i duża precyzja danych tworzących modele obliczeniowe. Kierując się wyżej wspomnianymi potrzebami w ramach pracy zaprojektowano moduł obliczeniowy realizujący operacje exp() w standardzie IEEE-754 o podwójnej precyzji.

2. Funkcja eksponenty

Techniki obliczania elementarnych funkcji mogą zostać zakwalifikowane do dwóch grup: iteracyjne oraz nieiteracyjne. Do implementacji została wybrana metoda nieiteracyjna, która może być realizowana następującymi sposobami:

- metody tablicowe - Look-Up Table (LUT),
- metody aproksymacyjne,
- metody mieszane (tablicowo-aproksymacyjne).

Metody tablicowe są najprostsze koncepcyjnie i zarazem wyniki ich implementacji skutkują bardzo szybką pracą układu wynikowego. Zasadniczą wadą tego rozwiązania jest jednak spora ilość zasobów zajmowanych przez tablice LUT przechowujące wartości funkcji, która wzrasta wykładniczo wraz z precyzją danej wejściowej. W konsekwencji wielkość pamięci staje się niedopuszczalna w przypadku danej wejściowej szerszej niż około 16-bitów.

Istnieje sporo implementacji metod tablicowych obliczania funkcji exp(), lecz projekt zrealizowany w Ames Laboratory [1] zwraca szczególną uwagę ze względu na zbliżony profil prac badawczych oraz stosowaną 64-bitową precyzje obliczeń. We wspomnianej pracy zostały wykorzystane bardzo dobrze znane zależności:

$$e^x = 2^{x \cdot log2(e)} = 2^{xc} \cdot e^{x-xc \cdot ln(2)} \quad (1)$$

oraz

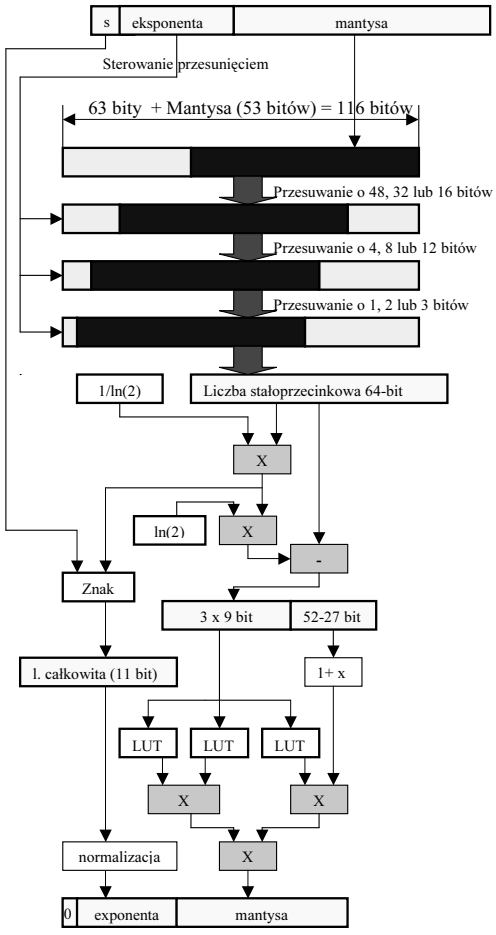
$$e^{x1+x2} = e^{x1} \cdot e^{x2}. \quad (2)$$

Zastosowanie podstawy liczby *2* zamiast *e* zdecydowanie upraszcza obliczenie wartości 2^x dla x_c całkowitych, ponieważ wartość x_c bezpośrednio przekłada się na wartość eksponenty wyniku. W konsekwencji wartość x jest dzielona na część całkowitą x_c i ułamkową x_u. Część całkowita x_c jest zapisywana bezpośrednio do eksponenty wyniku. Część ułamkowa x_c jest natomiast wykorzystywana dalej do obliczenia mantysy wyniku.

Przykład [1] oblicza funkcję *exp()* podwójnej precyzji z wykorzystaniem równania (2). Można w nim zaobserwować jak dużą część zajmowanych zasobów stanowią układy mnożące zmiennoprzecinkowe. Powyższa implementacja absorbuje niestety prawie wszystkie zasoby jednego z większych układów: Virtex-II Pro (55,616 slice'ów).

Powszechnie stosowanymi metodami są również aproksymacje wielomianowe [2]. Rozwiązania te wykazują znaczące zalety w porównaniu ze wspomnianymi powyżej implementacjami tablicowymi [3], należą do nich relatywnie niewielka zajętość zasobów oraz spora szybkość pracy przy właściwie dobranej funkcji aproksymującej. Rozważania te jednak dotyczyły obliczeń pojedynczej precyzji. Zastosowanie tych metod dla liczb o większej precyzji (większej szerokości argumentu realizowanej funkcji) narzuca jednak konieczność stosowania wielomianów wyższego stopnia, co z kolej związane jest z większą liczbą mnożeń.

Zauważyć, więc można, że dla większych precyzji danych wejściowych (np. 64 bit) metoda aproksymacji wielomianowej traci swoje podstawowe zalety. Rozwiązanie tablicowe jest również obarczone istotnymi mankamentami w postaci bardzo dużej zajętości zasobów.



Rys. 1. Architektura proponowanego modułu obliczania funkcji *exp()*
Fig. 1. Architecture of the proposed *exp()* function module

Wnioski płynące z powyższych obserwacji w naturalny sposób skłaniają do zastosowania metody alternatywnej, łączącej zalety dwóch wymienionych rozwiązań. Są to rozwiązania mieszane tablicowo-aproksymacyjne. W przeszłości zrealizowano wiele implementacji metod mieszanych obliczania funkcji elementarnych [4]. Dotyczyły one jednak w znakomitej większości operacji na danych zmiennoprzecinkowych, co najwyżej 32-bitowej precyzji [5]. Jako że metody mieszane obliczania funkcji elementarnych w standardzie liczb zmiennoprzecinkowych zostały najpierw zaimplementowane dla procesorów ogólnego przeznaczenia [6], pierwsze rozwiązania sprzętowe bardzo wyrazie czerpią z algorytmów softwarowych. Takie podejście nie wykorzystuje potencjału sprzętowego środowiska, tkwiącego w równoległości pracy i większych dostępnych zasobach logicznych. Dlatego obserwuje się obecnie prace mające na celu stworzenie szybkiej implementacji metody mieszanej obliczania funkcji *exp()* [7] oraz innych funkcji elementarnych. Jest to uzasadnione ogromnym potencjałem nowoczesnych układów programowalnych FPGA.

3. Proponowany moduł

Architekturę [1] można w prosty sposób ulepszyć poprzez zastosowanie większej pamięci LUT. Na przykład zamiast stosowania pamięci LUT o szerokości magistrali adresowej 1 można zastosować pamięć LUT o szerokości 4 (powszechnie stosowane w układach FPGA) i większej. Analiza zajmowanych zasobów przez pamięć LUT oraz układów mnożących doprowadziła do wniosku, że najlepszym rozwiązaniem będzie użycie dużych pamięci blokowych Block RAM (BRAM) znajdujących się w układach FPGA. Dla układów Virtex 4 mamy do dyspozycji pamięci *512x32*-bity (9-bitowa magistrala adresowa). Dla 54-bitowej mantysy wymagane jest wiec użycie 6-pamięci LUT (czyli w sumie 12 pamięci BRAM) oraz 5 układów mnożących.

Proponowany moduł obliczeniowy składa się z następujących elementów (schemat blokowy przedstawiony został na rys. 1):

- logiki sprawdzania stanów wyjątkowych (*inf*, *NaN*) oraz konwersji danych wejściowych do wewnętrznego stałoprzecinkowego standardu zapisu liczb,
- logiki separującej część całkowitą od ułamkowej liczby, łącznie z migracją znaku do części całkowej (wyrażonej jako pole eksponenta w standardzie IEEE-754),
- tablic LUT (9-cio wejściowych) przechowujących cząstkowe wartości *exp(x)*,
- logiki normalizacji wyniku końcowego do standardu IEEE-754.

4. Wyniki implementacji

Wyniki zamieszczone poniżej nie uwzględniają mechanizmu potokowości, który będzie zaimplementowany w finalnej wersji modułu sprzętowego, gdy zakończony zostanie etap badań nad numerycznymi aspektami architektury modułu (oczekiwana dokładność obliczeń, wielkości tablic LUT, rząd wielomianu aproksymującego, wielkość zajmowanych zasobów).

Tab. 1. Wyniki implementacji modułu *exp()* bez mechanizmu potokowości oraz procentowa zajętość układu Xilinx Virtex-4 LX200. Mnożarka sprzętowa przed modyfikacja
Tab. 1. Implementation results (area and % of available resources for Xilinx Virtex-4 LX200) for *exp()* module without pipelining, standard multiplier

Rodzaj	# 4-wej LUT	# przerzutników	# 18-Kb BRAM	# DSP48
Bez logiki DSP48	13375 (7.5%)	105 (0.06%)	6 (1.8%)	0
Z logiką DSP48	1293 (0.73%)	105 (0.06%)	6 (1.8%)	71 (74%)

W naszych badaniach korzystamy z komputera o dużej mocy obliczeniowej Altix 4700 firmy SGI. Niniejszy projekt zostanie docelowo zaimplementowany na karcie SGI RASC RC100 Blade jako element większego systemu obliczeniowego. SGI RASC RC100 Blade został wyposażony w dwa układy typu Xilinx Virtex-4 LX200, dlatego zamieszono poniżej wyniki implementacji dotyczą właśnie tego układu FPGA.

Wykorzystanie bloków DSP48 wiąże się z użyciem wbudowanych mnożarek 18×18 będących ich integralna częścią. Takie podejście zmniejsza wyraźnie liczbę zaangażowanych pamięci 4-wejściowych LUT, co zostało przedstawione w Tab. 1. Jednakże liczba wbudowanych modułów mnożących jest stosunkowo niewielka i tylko jeden moduł *exp()* może być zaimplementowany z użyciem tych układów. Dlatego w przypadku użycia większej liczby modułów *exp()* (równoległej pracy paru modułów w celu przyspieszenia sumarycznej szybkości działania) konieczne jest użycie logiki ogólnego przeznaczenia (pamięci LUT) w ramach układu mnożącego.

Można zauważyć, że dane przechowywane w pamięci LUT (BRAM) mają pewną charakterystyczną strukturę. Cześć komórek pamięci zapełniona jest w pewnym stopniu bitami o wartości 0. Korzystając z tej własności możliwe jest wprowadzenie pewnych modyfikacji do układów mnożących. Mianowicie mnożenie jest wykonywane tylko dla młodszych bitów, nie obejmuje starszych bitów będących zawsze zero. Natomiast mnożenie przez wiodącą jedynkę jest wykonywane jako operacja dodawania.

Takie rozwiązanie układu mnożącego wyraźnie wpłynęło na zmniejszenie zajętości zasobów. Zamieszczone poniżej Tab. 2 i Tab. 3 obrazują zarówno wpływ zastosowanej mnożarki sprzętowej jaki i liczby *guard_bits* na zajętość zasobów układu FPGA.

Tab. 2. Wyniki implementacji modułu *exp()* bez mechanizmu potokowości oraz procentowa zajętość układu Xilinx Virtex-4 LX200. Zmodyfikowana mnożarka sprzętowa. Guard_bits = 8

Tab. 2. Implementation results for *exp()* module (area and % of available resources for Xilinx Virtex-4 LX200), no pipelining, improved multiplier structure, number of guard bits is 8

Rodzaj	# 4-wej LUT	# przerzutników	# 18-Kb BRAM	# DSP48
Bez logiki DSP48	10710 (6 %)	105 (0.06%)	6 (1.8%)	0
Z logiką DSP48	1398 (0.73%)	105 (0.06%)	6 (1.8%)	54 (56%)

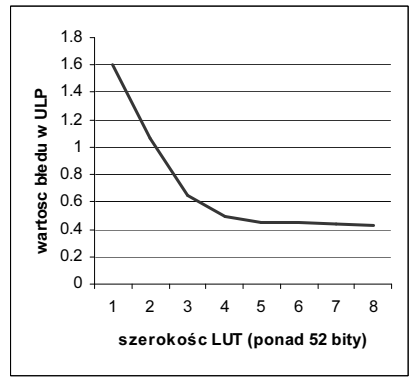
Zmniejszenie ilości bitów ochronnych (guard_bits) skutkuje zmniejszeniem szerokości rejestrów wewnętrznych a tym samym zajętości zasobów (zob. Tab. 2 i Tab. 3).

Tab. 3. Wyniki implementacji modułu exp() bez mechanizmu potokowości oraz procentowa zajętość układu Xilinx Virtex-4 LX200. Zmodyfikowana mnożarka sprzętowa. Guard_bits = 1

Tab. 3. Implementation results for *exp()* module (area and % of available resources for Xilinx Virtex-4 LX200), no pipelining, improved multiplier structure, number of guard bits is 1

Rodzaj	# 4-wej LUT	# przerzutników	# 18-Kb BRAM	# DSP48
Bez logiki DSP48	9039 (5 %)	105 (0.06%)	6 (1.8%)	0
Z logiką DSP48	1300 (0.72%)	105 (0.06%)	6 (1.8%)	54 (56%)

Modyfikacja szerokości rejestrów wewnętrznych poprzez zmianę ilości guard_bit wpływa jednak na dokładność obliczeń co pokazuje Rys. 2. Na podstawie Rys. 2 można zauważyć, że optymalna ze względu na szerokość rejestrów wewnętrznych ilości *guard_bits* wynosi 2-4. Dla większych wartości *guard_bits* nie obserwuję się znaczącej poprawy dokładności, wzrasta natomiast zajętość zasobów.



Rys. 2. Wartości błędu średniego (moduł) w ULP (Unit Last Place) dla różnych szerokości guard_bit

Fig. 2. Mean error (ABS) in ULP for different guard-bit widths

5. Podsumowanie

W niniejszym artykule przedstawiono architekturę sprzętowego modułu obliczania funkcji *exp()* o podwójnej precyzji. Należy podkreślić, że większość implementacji funkcji *exp()* ograniczała się do pojedynczej precyzji. Zwiększenie precyzji obliczeń wymagało nie tylko zwiększenia dokładności poszczególnych operacji składowych, ale również rozważenia różnych architektur implementacji. Wybrano algorytm mieszany: 27-bitów najstarszych mantysy wejściowej jest obliczana za pomocą metody tablicowej. Natomiast mniej znaczące bity są obliczane za pomocą rozwinięcia w szereg Taylora. Ponieważ dana wejściowa dla tego szeregu jest bardzo mała (mniejsza niż 2^{-27}) szereg jest bardzo szybko zbieżny, wymaga tylko pierwszego rozwinięcia $e^x = 1 + x$.

Wyniki implementacji pokazują, że zasoby współczesnych układów FPGA są wystarczające do wykonywania operacji eksponenty o podwójnej precyzji obliczeń, co więcej możliwe jest umieszczenie w jednym układzie FPGA więcej niż dziesięć tego typu modułów obliczeniowych.

6. Literatura

[1] Stanek S., Benjegerdes T., Reconfigurable Computing for High Perform-ance Technical Computing, Scalable Computing Lab, Ames Laboratory, Ames, IA 50010

[2] Jeremie Detrey, Florent de Dinechin, Second Order Function Approxima-tion Using a Single Multiplication on FPGAs, FPL 2004 : Field-Programmable Logic and Applications Antwerp, 30 Aug. - 1 Sep. 2004, pp. 221-230

[3] Detrey J., de Dinechin F, Table-based polynomials for fast hardware func-tion evaluation, 16th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'05), Samos, Greece, July 2005, pp. 328-333.

[4] Doss C.C., Riley R.L., Jr., FPGA-Based Implementation of a Robust IEEE-754 Exponential Unit, 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04), pp. 229- 238

[5] Bui H.T., Tahar S., Design and Synthesis of an IEEE-754 Exponential Function, IEEE Canadian Conference on Electrical and Computer Engineering Shaw Conference Center, Edmonton, Alberta, Canada May 9-12 1999, pp. 450-455 vol.1

[6] Tang P., Table-Driven Implementation of the Exponential Function in IEEE Floating-Point Arithmetic, ACM Transactions on Mathematical Software (TOMS), Volume 15 , Issue 2 (June 1989), pp. 144 - 157

[7] Detrey J., de Dinechin F., A parameterized foating-point exponential function for FPGAs, IEEE International Conference on Field-Programmable Technology (FPT'05), Singapore, December 2005, pp. 27-34.

Artykuł recenzowany