Zbigniew HAJDUK
POLITECHNIKA RZESZOWSKA, KATEDRA INFORMATYKI I AUTOMATYKI,
ul. W. Pola 2, 35-959 Rzeszów

Zmiennoprzecinkowa jednostka arytmetyczna dla sprzętowej maszyny wirtualnej

Dr inż. Zbigniew HAJDUK

Absolwent Wydziału Elektrycznego Politechniki Rzeszowskiej (1998, specjalność aparatura elektroniczna). Od 2006 jest adiunktem w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Zainteresowania badawcze obejmują projektowanie aplikacji z układami programowalnymi, języki opisu sprzętu - w tym szczególnie język Verilog oraz systemy mikroprocesorowe.

e-mail: zhajduk@prz-rzeszow.pl

Streszczenie

W artykule omówiono, opracowaną dla struktur FPGA, implementację układów realizujących podstawowe operacje arytmetyki zmiennoprzecinkowej. Implementacja charakteryzuje się pewnym kompromisem pomiędzy zapotrzebowaniem na zasoby logiczne układu programowalnego a szybkością realizacji operacji arytmetycznych określoną przez liczbę taktów zegara niezbędną do wykonania operacji. Wspomniane układy zostały wykorzystane jako zasadnicze komponenty zmiennoprzecinkowej jednostki arytmetycznej przeznaczonej dla sprzętowej maszyny wirtualnej. Maszyna ta, implementowana w układach FPGA, jest specjalizowanym mikrokontrolerem wykonującym pośredni kod wykonywalny generowany przez kompilator środowiska inżynierskiego CPDev, przeznaczonego do projektowania oprogramowania sterowników przemysłowych. Wykonane testy wydajności maszyny sprzętowej wyposażonej w zmiennoprzecinkową jednostkę arytmetyczną wskazują, że jest ona średnio kilkadziesiąt razy szybsza od dotychczas istniejących realizacji programowych, wykorzystujących popularne mikrokontrolery AVR i ARM.

Słowa kluczowe: układy FPGA, arytmetyka zmiennoprzecinkowa.

A floating point unit for the hardware virtual machine

Abstract

Under the CPDev (*Control Program Developer*) engineering environment, programs written in one of the languages defined in the IEC 61131-3 standard are compiled into the universal intermediate code executed on the side of programmable controllers by the virtual machines [9]. There are software implemented virtual machines, dedicated for the platform with popular AVR and ARM microcontrollers, and also there is a recently developed hardware virtual machine implemented using FPGA devices [2]. The hardware virtual machine, which in fact is a specialized microcontroller described in the Verilog Hardware Description Language [3], is several dozen times faster then its software counterparts [2]. But the main drawback of the existing hardware virtual machine is a lack of the ability of executing the floating point computations. The paper presents an architecture of the floating point arithmetic unit accomplishing basic floating point operation, designed for the hardware virtual machine. There are quite a lot of publications concerning FPGA implementation of the floating point arithmetic, for instance [6, 7, 8, 10, 11]. In this paper the realization of basic floating point operation, balanced between logic resources requirements and speed of computing (defined by the number of clock cycles necessary to end up a floating point operation), is presented. Figs. 1 and 2 show a simplified micro-architecture of the single precision (according to IEEE 754-1985 standard [5]) floating point multiplier and adder. A floating point divider has roughly the same structure as the multiplier – it differs in states functions performed by some blocks. A few different realizations of the multiplier and adder unit were designed – the details are presented in Tabs. 1 and 3. The general trend is as follows: a shorter clock cycle necessary to execute the operation needs more logic resources of FPGA. A floating point unit for the hardware virtual machine was designed based on the floating point multiplier, divider and adder blocks. Apart from the mentioned above basic floating point operation, the floating point unit also performs operations like: comparison and relation (equals, not equals, more than, more than or equal etc.), absolute value, negation, integer value to floating point value conversion, floating point to integer conversion (rounding, truncating) and some functions fetched from IEC 61131-3 standard like MIN, MAX, LIMIT. To compare performance of the hardware virtual machine equipped with the floating point unit and its software counterparts, the Whetstone based benchmark [1] was written in ST language. The test results are given in Tab. 4. The hardware virtual machine (implemented using *Xilinx* Spartan 3-AN FPGA XC3S1400AN-4FGG676) is several times faster than the software one implemented on AVR and ARM microcontrollers, and even a little bit faster than the PC based virtual machine (under .NET environment).

Keywords: field programmable gate array, floating point arithmetic.

1. Wstęp

W środowisku programistyczno-uruchomieniowym CPDev (*Control Program Developer*) [9], przeznaczonym do projektowania oprogramowania sterowników programowalnych zgodnie z normą IEC 61131-3, programy kompilowane są do uniwersalnego kodu pośredniego, który po stronie sterownika wykonywany jest przez maszynę wirtualną. Oprócz programowych implementacji maszyny wirtualnej dedykowanych dla platform sprzętowych z mikrokontrolerami AVR oraz ARM, opracowana została również całkowicie sprzętowa realizacja tej maszyny [2], wykorzystująca układy programowalne FPGA. Sprzętowa realizacja maszyny wirtualnej, nazwana w skrócie maszyną sprzętową, charakteryzuje się znacznie większą szybkością wykonywania kodu pośredniego. Jak wykazały odpowiednie testy, maszyna ta jest średnio kilkadziesiąt razy szybsza od jej programowego odpowiednika pracującego z użyciem typowych, w tym również 32-bitowych, mikrokontrolerów [2]. Jednak wadą dotychczasowej implementacji maszyny sprzętowej jest brak możliwości realizacji obliczeń w oparciu o arytmetykę zmiennoprzecinkową. W artykule opisano stosunkowo prostą zmiennoprzecinkową jednostkę arytmetyczną realizującą podstawowe operacje arytmetyczne, przeznaczoną do integracji z maszyną sprzętową.

Podczas projektowania modułu zmiennoprzecinkowej jednostki arytmetycznej przyjęto założenie, że podobnie jak w przypadku wirtualnego komponentu maszyny sprzętowej (maszyna sprzętowa istnieje w postaci tzw. wirtualnego komponentu – *IP core* – modułu opisanego w języku opisu sprzętu, który w każdej chwili może być potencjalnie zaimplementowany z wykorzystaniem układów FPGA lub ASIC), specyfikacja tego modułu w języku opisu sprzętu będzie niezależna od docelowej architektury, producenta czy też rodziny układów FPGA i będzie możliwa do implementacji również w układach ASIC. Przyjęcie takiego założenia wyklucza możliwość wykorzystania udostępnianych przez producentów układów programowalnych tzw. generatorów rdzeni – gotowych, ale w pewnym zakresie konfigurowalnych, komponentów realizujących określoną funkcjonalność, np. dodawanie czy mnożenie zmiennoprzecinkowe, dedykowanych dla konkretnych rodzin układów programowalnych. Jednocześnie pociąga to za sobą konieczność zaprojektowania od podstaw elementarnych komponentów odpowiedzialnych za realizację działań w oparciu o arytmetykę zmiennoprzecinkową.

Struktury programowalne FPGA zoptymalizowane są pod kątem realizacji obliczeń stałoprzecinkowych [11]. Implementacja nawet podstawowych operacji zmiennoprzecinkowych pochłania dużą liczbę zasobów układu FPGA i dodatkowo wiąże się ze stosunkowo niską wydajnością (niską częstotliwością taktowania) całego systemu ze zmiennoprzecinkową arytmetyką [11, 8]. Dlatego też wiele opracowań naukowych koncentruje się wokół problemów efektywnej implementacji arytmetyki zmiennoprzecinkowej w układach FPGA. Dla przykładu, w pracy Malika i Ho [6] autorzy analizują różne implementacje zmiennoprzecinkowego sumatora w układach Virtex p2 (*Xilinx*). Rozważane są trzy różne

algorytmy dodawania: standardowy, oparty na predykcji wiodącej jedynki (*Leading One Predictor*) oraz algorytm z podwójnym układem operacyjnym zoptymalizowanym dla przypadków, gdy istnieje lub nie, potrzeba użycia detektora lub predykcji wiodącej jedynki (*Far and Close Data-path algorithm*). Thakkar i Ejnioui [11] opisują sekwencyjną i potokową realizację modułu zmiennoprzecinkowego dzielenia podwójnej precyzji oraz wyliczania pierwiastka kwadratowego. Autorzy dokonali implementacji w układach Virtex 5 (*Xilinx*) wykorzystując język opisu sprzętu VHDL. Ho, Yu, Luk, Leong, Wilton [4] zaproponowali specjalną architekturę rekonfigurowanych układów FPGA zoptymalizowaną dla aplikacji zmiennoprzecinkowych. Dla wybranych testów architektura ta charakteryzuje się 4 krotnym wzrostem szybkości i 25 krotnym zmniejszeniem zapotrzebowania na zasoby logiczne, w stosunku do typowych, komercyjnych układów FPGA.

W niniejszym artykule uwagę skoncentrowano na implementacji podstawowych operacji arytmetyki zmiennoprzecinkowej pojedynczej precyzji, opartej na standardzie IEEE 754-1985 [5], odpowiedniej do zastosowania jako jeden z komponentów wspomnianej wcześniej maszyny sprzętowej. Implementacja ta charakteryzuje się m. in. pewnym kompromisem pomiędzy zapotrzebowaniem na zasoby logiczne układu programowalnego i czasem obliczeń określonym jako liczba taktów zegara niezbędna do wykonania danej operacji. Opracowana zmiennoprzecinkowa jednostka arytmetyczna ma charakter wirtualnego komponentu opisanego w języku Verilog [3] i podobnie jak maszyna sprzętowa może być zaimplementowana w dowolnym układzie programowalnym o odpowiednio dużych zasobach logicznych.

W dalszej części artykułu przedstawione zostaną architektury układów zmiennoprzecinkowego mnożenia, dzielenia, dodawania oraz odejmowania, wchodzące w skład zmiennoprzecinkowej jednostki arytmetycznej, a także pokazane zostaną wyniki testów wydajności maszyny sprzętowej wyposażonej w prezentowaną jednostkę w odniesieniu do programowych maszyn wirtualnych.

2. Mnożenie zmiennoprzecinkowe

Mnożenie jest najprostszym działaniem spośród podstawowych operacji zmiennoprzecinkowych. W ogólnym przypadku sprowadza się ono do pomnożenia mantys i dodania eksponent dwóch mnożonych liczb zmiennoprzecinkowych [6]:

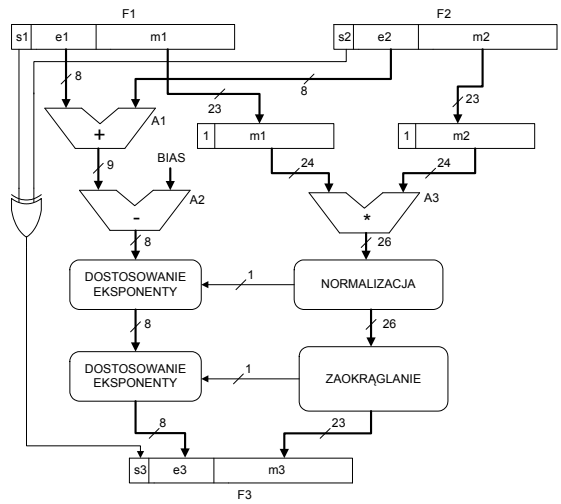
$$F_1 \times F_2 = (-1)^{s_1}(1.m_1 2^{e_1-bias}) \times (-1)^{s_2}(1.m_2 2^{e_2-bias}) = \\ = (-1)^{s_3} 1.m_3 2^{e_3-bias}, \quad (1)$$

gdzie:

$$m_3 = 1.m_1 \times 1.m_2,$$
$$e_3 = e_1 + e_2 - bias,$$
$$bias = 127 \text{ dla liczb pojedynczej precyzji}.$$

Na rys. 1 pokazano uproszczony schemat architektury układu mnożenia zmiennoprzecinkowego, którego działanie opisuje równanie (1). Blok normalizacji jest tutaj zwykłym rejestrem przesuwającym warunkowo wynik mnożenia mantys o jeden bit w prawo (jednocześnie w takim przypadku istnieje potrzeba zwiększenia eksponenty o 1). Blok zaokrąglania realizuje zaokrąglanie w trybie do najbliższej parzystej (*round to nearest even*), zgodnie ze standardem IEEE 754 [5]. Do tego celu potrzebne są dodatkowe bity: ochronny (*guard bit*), zaokrąglenia (*round bit*) oraz tzw. lepki bit (*sticky bit*) wyliczane w module mnożącym A3 (rys. 1).

W tab. 1 przedstawiono wyniki syntezy modułu mnożenia zmiennoprzecinkowego w układach Spartan 3AN (*Xilinx*) o klasie szybkości -4 (w prototypie sterownika PLC opartego na maszynie sprzętowej wykorzystano układ XC3S1400AN-4FGG676 [2]) dla różnych odmian układu stałoprzecinkowego mnożenia mantys (blok A3). W wersji A zastosowano szeregowy układ mnożący szczegółowo opisany w [3]. W wersji B wykorzystano cztery równoległe układy mnożące (*fast array multiplier*) o szerokości słów wejściowych równej 12 bitów i dwa sumatory tworzące razem 3 stopniowy, potokowy układ mnożenia 24-bitowego. Dla porównania, w wersji C jako bloku A3 użyto wbudowane w docelowym układzie FPGA 18-bitowe sprzętowe bloki mnożące według idei podanej w [13].



Rys. 1. Schemat mikroarchitektury zmiennoprzecinkowego układu mnożącego
Fig. 1. Micro-architecture of the floating point multiplication unit

Tab. 1. Porównanie wyników syntezy różnych wersji układu mnożącego
Tab. 1. Comparison of the synthesis results of different multiplier versions

Wersja	Liczba bloków logicznych *slice*	Maksymalna częstotliwość taktowania [MHz]	Liczba taktów zegara
A	162	161.9	27
B	462	77	5
C	160	151	5

Jak widać na podstawie danych zawartych w tab. 1, zastosowanie prostego szeregowego algorytmu mnożenia mantys skutkuje najmniejszym zapotrzebowaniem na zasoby logiczne, ale jednocześnie wymaga dużej liczby taktów zegara niezbędnej do realizacji pojedynczej operacji mnożenia. Zastosowanie szybkiego równoległego układu mnożenia mantys zwiększa blisko 3 krotnie zapotrzebowanie na zasoby, ale powoduje również 5 krotne skrócenie czasu obliczeń określonego liczbą taktów zegara, konieczną do wykonania zmiennoprzecinkowego mnożenia. Dodatkowo w tym przypadku występuje również znaczące zmniejszenie maksymalnej częstotliwości taktowania. W układach FPGA zawierających dedykowane sprzętowe multiplikatorami, ich wykorzystanie w bloku mnożenia mantys pozwala uzyskać najlepsze wskaźniki zapotrzebowania na zasoby i szybkości obliczeń.

3. Dzielenie zmiennoprzecinkowe

Operacja zmiennoprzecinkowego dzielenia jest analogiczna do operacji mnożenia i sprowadza się do podzielenia mantys i odjęcia eksponent:

$$F_1 / F_2 = (-1)^{s_1}(1.m_1 2^{e_1-bias}) / (-1)^{s_2}(1.m_2 2^{e_2-bias}) = \\ = (-1)^{s_3} 1.m_3 2^{e_3+bias}, \quad (2)$$

gdzie:

$$m_3 = 1.m_1 / 1.m_2,$$
$$e_3 = e_1 - e_2 - bias.$$

Ogólny schemat mikroarchitektury układu dzielenia zmiennoprzecinkowego – po dokonaniu zamiany typu operacji realizowanych w następujących blokach: A1 z dodawania na odejmowanie, A2 z odejmowania na dodawanie i A3 z mnożenia na dzielenie – jest identyczny do przedstawionego na rys. 1. Blok normalizacji realizuje tym razem operację warunkowego przesunięcia o jeden bit w lewo, a blok zaokrąglania jest zasadniczo taki sam jak dla układu zmiennoprzecinkowego mnożenia.

W tab. 2 pokazano, analogiczne do zawartych w tab. 1, wyniki syntezy układu dzielenia zmiennoprzecinkowego. W tym przypadku zaprojektowano tylko jedną wersję układu wykorzystując, w ramach bloku A3, szeregowy algorytm dzielenia stałoprzecinkowego opisany w [11].

Tab. 2. Wynik syntezy układu dzielenia zmiennoprzecinkowego
Tab. 2. Synthesis result of the floating point divider

Liczba bloków logicznych *slice*	Maksymalna częstotliwość taktowania [MHz]	Liczba taktów zegara
123	137.2	26

Szybkość (liczba taktów zegara potrzebna na wykonanie operacji dzielenia) oraz zapotrzebowanie na zasoby dla układu zmiennoprzecinkowego dzielenia jest zdeterminowane algorytmem zastosowanym w bloku dzielenia mantys. Najmniej wymagające pod względem ilości zasobów logicznych i jednocześnie najprostsze w realizacji są algorytmy szeregowe takie jak algorytm dzielenia restytucyjnego, szczegółowo omówiony w [3], czy też bazujący na nim algorytm przedstawiony w [11]. Jednak są to algorytmy bardzo wolne (dla słowa wejściowego szerokości N bitów, potrzeba N taktów zegara na wyliczenie ilorazu). Szybsze algorytmy dzielenia, takie jak np. dzielenie poprzez mnożenie przez odwrotność charakteryzują się dużo większą złożonością w realizacji sprzętowej, w tym szczególnie zapotrzebowaniem na zasoby logiczne układu FPGA [12] oraz wymagają np. dodatkowych zabiegów eliminujących akumulujące się błędy kwantyzacji [7]. Dlatego też, biorąc pod uwagę założenia projektowe dla jednostki zmiennoprzecinkowej, zorientowane na minimalizację ilości zasobów logicznych niezbędnych do implementacji, w module dzielenia mantys zastosowano prosty algorytm szeregowy.

4. Dodawanie zmiennoprzecinkowe

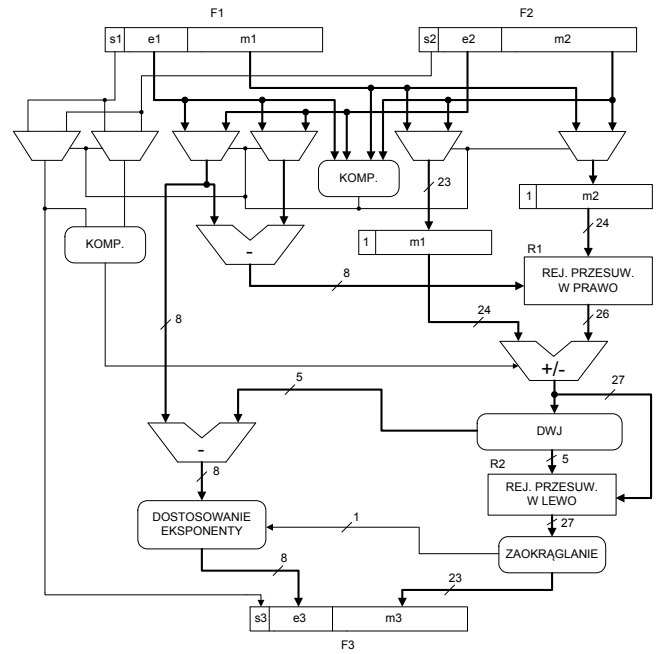
Operacja zmiennoprzecinkowego dodawania jest nieco bardziej złożona niż np. operacja mnożenia. W ogólnym przypadku możemy zapisać:

$$F_1 + F_2 = (a_1 2^{e_1}) \pm (a_2 2^{e_2}) = (a_1 2^{e_1}) \pm ((a_2 / 2^{e_1 - e_2}) 2^{e_1}) = \\ = (a_1 \pm (a_2 / 2^{e_1 - e_2})) 2^{e_1}. \quad (3)$$

Sumowanie odpowiednich składników w równaniu (3) następuje w przypadku gdy znaki obydwu liczb F_1 i F_2 są identyczne. W przeciwnym razie składniki są odejmowane. Końcowa postać równania, celem spełnienia wymagań standardu IEEE 754, wymaga jeszcze procedury normalizacji. Dodatkowo przyjęto założenie, że wartość bezwzględna pierwszej dodawanej liczby jest większa od wartości bezwzględnej drugiej liczby.

Na rys. 2 pokazano uproszczony schemat mikroarchitektury układu zmiennoprzecinkowego sumatora według standardowego algorytmu dodawania, implementującego równanie (3). Według autorów pracy [6] standardowy algorytm dodawania charakteryzuje się najmniejszym zapotrzebowaniem na zasoby logiczne docelowego układu programowalnego. Zespół multiplekserów widoczny u góry rysunku wraz z blokami komparatorów oznaczonymi jako „KOMP.” zapewnia spełnienie założenia o odpowiedniej relacji wartości bezwzględnych dodawanych liczb.

W tablicy 3 przedstawiono wyniki syntezy układu o ogólnej strukturze podanej na rys. 2, dla różnych rozwiązań konstrukcyjnych niektórych bloków składowych. W wersji A zastosowano szeregowy rejestr przesuwający R1 i szeregowy układ normalizacji składający się z detektora wiodącej jedynki (DWJ) i rejestru przesuwającego R2. Wersja B różni się od wersji A budową rejestru R1, który w tym przypadku w pojedynczym takcie zegara przesuwa swoją zawartość o określoną liczbę bitów w prawo. W wersji C zastosowano rejestr R1 taki jak dla wersji B i dodatkowo przyśpieszono działanie układu normalizacji, który również wykonuje operację normalizacji w pojedynczym takcie zegara.



Rys. 2. Schemat mikroarchitektury zmiennoprzecinkowego układu sumatora
Fig. 2. Micro-architecture of the floating point adder

Tab. 3. Porównanie wyników syntezy różnych wersji układu sumatora
Tab. 3. Comparison of the synthesis results of different adder versions

Wersja	Liczba bloków logicznych *slice*	Maksymalna częstotliwość taktowania [MHz]	Liczba taktów zegara
A	261	113.6	5…52
B	299	131.6	5…29
C	415	88	5

Jak można było się spodziewać wersja zmiennoprzecinkowego sumatora, która wymaga najmniejszej liczby taktów zegara (wersja C) zajmuje jednocześnie najwięcej zasobów logicznych i charakteryzuje się stosunkowo niską maksymalną częstotliwością taktowania. Wersje o mniejszym zapotrzebowaniu na zasoby logiczne dodatkowo wykazują się zmienną liczbą taktów zegara potrzebną do wykonania operacji dodawania, zależną od wartości dodawanych liczb.

5. Zmiennoprzecinkowa jednostka arytmetyczna

Podstawowymi komponentami składowymi zmiennoprzecinkowej jednostki arytmetycznej są przedstawione powyżej układy zmiennoprzecinkowego dodawania, mnożenia i dzielenia. Oprócz operacji arytmetycznych realizowanych przez te układy jednostka wykonuje również operacje relacji (mniejsze, większe itp.), porównania (równy, różny), podaje wartość bezwzględną, realizuje zmianę znaku, konwersję liczby całkowitej do postaci zmiennoprzecinkowej, konwersję liczby zmiennoprzecinkowej do liczby całkowitej (zaokrąglanie i obcięcie części ułamkowej) oraz niektóre operacje przewidziane w normie IEC 61131-3, takie jak mini-

mum oraz maksimum z dwóch liczb oraz funkcję ogranicznika pomiędzy dwoma skrajnymi wartościami (LIMIT).

Pod względem układowym, jednostka arytmetyczna jest rodzajem automatu sekwencyjnego, który w celu realizacji jednej z wymienionych wyżej operacji odpowiednio aktywuje układy zmiennoprzecinkowego dodawania, mnożenia oraz dzielenia. Podstawową wersję zmiennoprzecinkowej jednostki arytmetycznej zbudowano w oparciu o wersją A układu mnożenia i wersję B układu dodawania. W takiej konfiguracji jednostka zaimplementowana w układzie FPGA *Xilinx* z rodziny Spartan-3AN (XC3S1400AN-4FGG676) zajmuje 985 bloków logicznych typu *slice*, co stanowi około 8% wszystkich bloków *slice* dostępnych w tym układzie. Jednocześnie może być taktowana zegarem o maksymalnej częstotliwości 89.6MHz.

6. Testy porównawcze

Aby określić wydajność maszyny sprzętowej wyposażonej w zmiennoprzecinkową jednostkę arytmetyczną, w odniesieniu do realizacji programowych, wykonano pewne testy porównawcze. Testy te oparto na przeniesionym z komputerów osobistych algorytmie Whetstone'a [1] wykorzystującym arytmetykę stało i zmiennoprzecinkową. Algorytm ten zapisano w języku ST (zasadniczy język programowania sterowników przemysłowych w środowisku CPDev [9]) implementując 6 z 10 standardowo stosowanych modułów. Nie zaimplementowano np. modułów realizujących przesyłanie tablic jako parametrów wywołania procedur (w języku ST systemu CPDev odpowiednikiem procedur są bloki funkcjonalne, do których jednak nie można przekazywać tablic) oraz modułu wykonującego działania z użyciem funkcji trygonometrycznych.

W tab. 4 zestawiono czasy obliczeń testu Whetstone'a dla programowych maszyn wirtualnych pracujących na platformach z mikrokontrolerami AVR (ATMega128) i ARM oraz na komputerze PC. Maszyna sprzętowa wyposażona w opisaną zmiennoprzecinkową jednostkę arytmetyczną, taktowana zegarem 40MHz, okazała się szybsza odpowiednio: 115 razy od realizacji programowej za pomocą mikrokontrolera AVR ATMega128 (maszyna ta pracuje w sterowniku SMC produkowanym przez zakłady LUMEL z Zielonej Góry), 13.8 razy od maszyny programowej pracującej na mikrokontrolerze ARM i 1.8 razy od maszyny zaimplementowanej na komputerze PC. Szczególnie ten ostatni wynik jest dość zaskakujący, ale należy wziąć pod uwagę, że programowa maszyna wirtualna pracowała tu w środowisku .NET, które dodatkowo wprowadza spore opóźnienie.

Tab. 4. Czas obliczeń testu Whetstone'a podany w milisekundach
Tab. 4. Computation time of the Whetstone benchmark (in milliseconds)

AVR ATMega128 (14,7456 MHz)	ARM AT91SAM7S (18.432 MHz)	PC Core Duo 1,83GHz (.NET)	Maszyna sprzętowa FPGA (40MHz)
3 407 980	407 719	52 141	29 560

Dla porównania w przypadku stałoprzecinkowych testów opisanych w [2] (liczby pierwsze, liczby doskonałe, konwersje binarno-dziesiętne) maszyna sprzętowa taktowana zegarem 50MHz okazała się szybsza średnio 187 razy od wersji programowej z mikrokontrolerem AVR i 57 razy od wersji z mikrokontrolerem ARM. Z kolei 4.75 razy wolniejsza od wersji na komputerze PC. Można więc powiedzieć, że wynik testu Whetstone'a dla maszyny sprzętowej, w odniesieniu do programowych realizacji maszyn wirtualnych na platformach z mikrokontrolerami AVR i ARM, wypadł nieco poniżej oczekiwań. Z kolei maszyna sprzętowa okazała się nieznacznie szybsza od maszyny wirtualnej zaimplementowanej na komputerze PC w wybranej konfiguracji – co jest wynikiem raczej niespodziewanym.

Przeprowadzono również test Whetstone'a dla szybszej odmiany zmiennoprzecinkowej jednostki arytmetycznej wyposażonej w układ mnożący w wersji B (tab. 1) oraz sumator w wersji C (tab. 3) a także zmodyfikowany - przyśpieszony stałoprzecinkowy układ mnożący w jednostce ALU maszyny sprzętowej. W tym przypadku czas obliczeń zmniejszył się tylko o nieco ponad 3 sekundy i wyniósł 26083ms. Zasadniczo więc zmiana ta nie wpłynęła na ogólny wynik testu.

7. Podsumowanie

Wyposażenie maszyny sprzętowej w zmiennoprzecinkową jednostkę arytmetyczną pozwala na znaczne zwiększenie funkcjonalności oraz potencjalnego obszaru zastosowań tej maszyny jako szybkiego sterownika programowalnego. Przeprowadzone testy wykazały, że maszyna sprzętowa jest średnio kilkadziesiąt razy szybsza od dotychczas istniejących implementacji programowych przeznaczonych na platformy z mikrokontrolerami AVR i ARM.

Zaprojektowane wirtualne komponenty zmiennoprzecinkowych układów arytmetycznych mogą znaleźć zastosowanie również w innych przypadkach, np. jako podstawowe składniki szybkich sprzętowych bloków funkcjonalnych realizujących algorytmy cyfrowego przetwarzania sygnałów w oparciu o arytmetykę zmiennoprzecinkową itp.

Praca realizowana w ramach projektu MNiSW nr N N514 412736.

8. Literatura

[1] Curnow H. J., Wichmann B. A.: A synthetic benchmark. Computer Journal, Vol. 19, No 1, pp. 43-49, 1976.
[2] Hajduk Z., Trybus B., Sadolewski J.: Sprzętowa implementacja maszyny wirtualnej dla sterowników programowalnych. Metody Wytwarzania i Zastosowania Systemów Czasu Rzeczywistego, Praca zbiorowa pod redakcją L. Trybusa i S. Samoleja, WKŁ, str. 333-343, Warszawa, 2010.
[3] Hajduk Z.: Wprowadzenie do języka Verilog. Wydawnictwo BTC, Legionowo 2009.
[4] Ho Ch. H., Yu Ch. W., Luk W., Wilton S. J. E.: Floating-Point FPGA: Architecture and Modeling. IEEE Trans. on VLSI Systems, Vol. 17, No. 12, pp. 1709-1718, Dec. 2009.
[5] IEEE Standard Board and ANSI: IEEE Standard for Binary Floating-Point Arithmetic. IEEE Std 754-1985.
[6] Malik A., Ko S. B.: A Study on the Floating-Point Adder in FPGAS. IEEE Canadian Conf. on Electrical and Computer Engineering, CCECE '06, pp. 86-89, Ottawa, May 2006.
[7] Meyer-Baese U.: Digital Signal Processing with Field Programmable Gate Arrays. Third Edition, Springer Berlin Heidelberg, 2007.
[8] Parker M.: High-performance floating-point implementation using FPGAs. IEEE Military Communications Conference, MILCOM 2009, pp. 1-5, 2009.
[9] Rzońca D., Sadolewski J., Trybus B.: Prototype environment for controller programming in the IEC 61131-3 ST language. Computer Science and Information Systems, Vol. 4, No. 2, Dec. 2007.
[10] Sahin S., Kavak A., Becerikli Y., Demiray H. E.: Implementation of floating point arithmetics using an FPGA. Mathematical Methods in Engineering, Part 5, Pages 445-453, Springer, 2007.
[11] Thakkar A. J., Ejnioui A.: Design and implementation of double precision floating point division and square root on FPGAs. IEEE Aerospace Conference, 2006.
[12] Wang X., Nelson B. E.: Tradeoffs of designing floating-point division and square root on Virtex FPGAs. Proceedings of the 11th Annual IEEE Symp. on Field-Programmable Custom Computing Machines, FCCM 2003, pp. 195-203, 2003.
[13] Xilinx Application Note: Using Embedded Multipliers in Spartan-3 FPGAs. XAPP 467, 2003.

otrzymano / received: 16.10.2010
przyjęto do druku / accepted: 01.12.2010 *artykuł recenzowany*