Maciej PETKO, Grzegorz KARPIEL, Tadeusz UHL
AKADEMIA GÓRNICZO-HUTNICZA, WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI, KATEDRA ROBOTYKI I MECHATRONIKI

Implementacja algorytmów sterowania w układach FPGA na przykładzie robota równoległego

Dr inż. Maciej PETKO

Adiunkt w Katedrze Robotyki i Mechatroniki AGH. Jego zainteresowania skupiają się na mechatronice, robotyce, zagadnieniach prototypowania i implementacji algorytmów przetwarzania sygnałów, głównie w sterowaniu i diagnostyce technicznej.

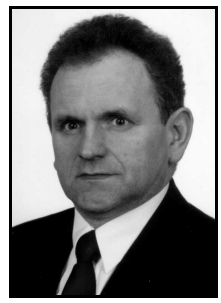
e-mail: petko@agh.edu.pl

Mgr inż. Grzegorz KARPIEL

Ukończył studia na Wydziale Inżynierii Mechanicznej i Robotyki AGH w 2001 roku. Obecnie doktorant w Katedrze Robotyki i Mechatroniki AGH. Główne zainteresowania skupiają się na mechatronice, zagadnieniach dotyczących robotów równoległych oraz projektowaniu sterowników opartych na układach FPGA.

e-mail: gkarpiel@agh.edu.pl

Prof. dr hab. inż. Tadeusz UHL



Jest kierownikiem Katedry Robotyki i Mechatroniki, Akademii Górniczo-Hutniczej w Krakowie. W swoich pracach zajmuje się zagadnieniami dynamiki konstrukcji, a zwłaszcza analizą modalną. Jego zainteresowania obejmują także układy aktywnej redukcji drgań, układy sterowania i szeroko pojętą mechatronikę. Jest autorem 15 książek i kilkuset artykułów dotyczących wspomnianych zagadnień.

e-mail: tuhl@agh.edu.pl

Streszczenie

W artykule przedstawiono problemy związane z implementacją algorytmów sterowania w układach FPGA. Zaproponowane rozwiązanie oparte jest na architekturze „System-on-Programmable-Chip" z wprogramowanym mikroprocesorem, która pozwala na mieszaną, sprzętowo-programową implementację i badanie możliwych realizacji algorytmu. Jako przykład zastosowania przedstawiono sterowanie neuronowe robotem do frezowania o trzech stopniach swobody. Sterownik jest oparty na neuronowym modelu dynamiki odwrotnej manipulatora, uczonym na danych zebranych z użyciem stabilizującego sterownika wykorzystującego strukturalny model analityczny manipulatora. Dla porównania obydwa sterowniki zostały zaimplementowana w systemie o tej samej architekturze.

Implementation of control algorithms in FPGA on an example of a parallel robot

Abstract

The paper presents problems related to implementation of control algorithms in FPGA. Proposed solution is based on System-on-Programmable-Chip architecture with soft-processor that allows for mixed, hardware/software implementation and exploration of possible control algorithm realizations. The case study is a neural controller for 3-DOF parallel robot for milling. The controller is based on neural model of the inverse dynamics of the manipulator, trained on data collected with the use of a computed torque stabilizing controller. For comparison, both controllers were implemented in a system of the same architecture.

1. Wstęp

Sterownik jest istotną częścią systemu mechatronicznego a jego implementacja powinna być uwzględniana w całym procesie projektowania, zwłaszcza kiedy docelowa platforma sprzętowa oparta jest na układach FPGA (ang. Field Programmable Gate Array), co narzuca dodatkowe, złożone ograniczenia na algorytm sterowania [1].

Algorytm sterowana, który ma być implementowany w FPGA jest zazwyczaj ciągły w czasie, wykorzystuje ciągłe wartości sygnałów i jest opisany za pomocą równań algebraicznych lub schematu blokowego. Postać taka nie nadaje się do bezpośredniego zastosowania i wymaga wielu przekształceń, które wpływają nie tylko na notację, ale także na sam algorytm [2].

W przypadku złożonych algorytmów sterowania nie zawsze korzystne jest implementowanie sprzętowe całego algorytmu, a zwłaszcza tych jego fragmentów, które mają nieregularną strukturę, lub są trudne do transformacji do postaci stałoprzecinkowej. Najlepszym rozwiązaniem w takich przypadkach wydaje się być realizacja mieszana, sprzętowo-programowa. Stała się ona możliwa w związku z tym, że nowoczesne układy FPGA pozwalają zawrzeć w sobie cały system z mikroprocesorem (-ami), pamięcią, cyfrowymi układami peryferyjnymi i własnymi blokami przetwarzania sygnałów.

2. Implementacja jako „System-on-a-Programmable-Chip"

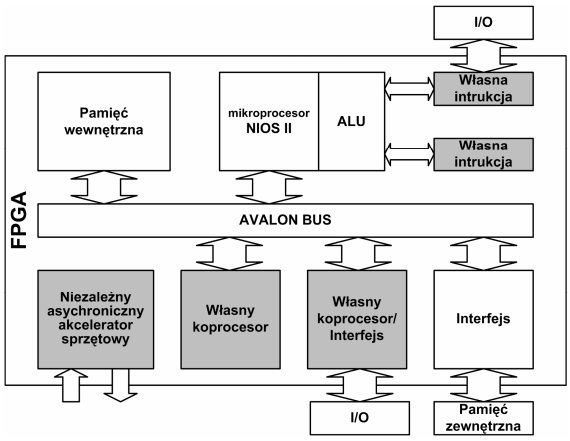
System w układzie (ang. System-on-Chip - SoC) jest pojedynczym układem scalonym zawierającym w sobie cały system: mikroprocesor(y), koprocesory, jednostki przetwarzania sygnałów, układy peryferyjne, pamięci, interfejsy komunikacyjne itd. Możliwość implementacji SoC w układach FPGA, nazywanych wtedy też „System-on-a-Programmable-Chip" (SoPC) daje dodatkową elastyczność w eksperymentowaniu z charakterystykami wydajności, nawet w późnych fazach procesu projektowania, oraz wydłużając cykl życia produktu przez możliwość aktualizowania wersji sprzętu i oprogramowania lub dostosowywania funkcjonalności do specyficznych potrzeb konkretnego użytkownika lub zastosowania, bez modyfikacji płyty drukowanej (fizycznej warstwy sterownika).

W tradycyjnym podejściu do projektowania SoC, projektant musi w sposób ręczny stworzyć opis w języku HDL nadrzędnej warstwy systemu, łączącej w całość poszczególne jego elementy, implementującej wymianę danych, synchronizację i arbitraż pomiędzy nimi. Pochłania to wiele wysiłku i czasu. Pojawiają się w ostatnich latach narzędzia ułatwiające integrację systemu, takie jak. ISE Xilinxa [4] lub SOPC Builder Altery [3]. Użycie ich, uwalnia projektanta od najbardziej pracochłonnych i nużących czynności i pozawala skoncentrować się na architekturze systemu. Dostępne możliwości zostaną bliżej wyjaśnione na przykładzie SoPC opartego na układzie FPGA z rodziny Stratix Altery, wprogramowanego (niewykonanego „na sztywno" w krzemie) 32-bitowego mikroprocesora Nios II typu RISC i oprogramowania SOPC Builder.

Stosując narzędzia SOPC Builder projektant w interfejsie graficznym wybiera elementy systemu, a program automatycznie generuje logikę połączeń między nimi. Uzyskiwane są w efekcie pliki HDL definiujące wszystkie komponenty systemu oraz plik z kodem najwyższego poziomu, definiującym połączenia między tymi komponentami. Celem SOPC Buildera jest wyabstrahowanie

złożoności połączeń i pozwolenie projektantowi na skupienie się na projektowaniu komponentów i architektury systemu.

Typową architekturę przedstawiono na rys. 1. System jest zbudowany ze standardowych komponentów (białe tło) oraz z komponentów stworzonych przez projektanta (szare tło) połączonych magistralą Avalon, zbudowaną ze specjalizowanych zasobów układu.



Rys. 1. Typowa architektura systemu w programowalnym układzie scalonym (SoPC), opatrego na układzie FPGA firmy Altera i wprogramowanym mikroprocesorze Nios II; bloki z szarym tłem przedstawiają niestandardowe, własne komponenty projektanta

W systemie może współistnieć wiele mikroprocesorów Nios II pracujących na wspólnej magistrali lub niezależnie. W układach FPGA o największej pojemności można zaimplementować do kilkuset takich procesorów, które są konfigurowalne i występują w wersjach różniących się szybkością, mocą obliczeniową i ilością zajmowanych zasobów układu. Architektura Niosa II umożliwia dodawanie własnych instrukcji, definiowanych przez użytkownika. Instrukcje te są jedną z metod na zwiększenie osiągów systemu przez rozszerzenie jednostki arytmetyczno-logicznej (ALU) procesora o własne operacje realizowane sprzętowo. Zakres możliwych architektur instrukcji użytkownika rozciąga się od prostych operacji kombinatorycznych do rozległych, wielocyklowych układów sekwencyjnych o zmiennej długości, które mogą komunikować się z urządzeniami zewnętrznymi w stosunku do FPGA, np. w celu akwizycji lub zapisu danych. Z punktu widzenia oprogramowania, własne instrukcje widziane są jako automatycznie generowane makroinstrukcje asemblera lub funkcje języka C. SOPC Builder generuje pliki konfiguracyjne dla kompilatora C/C++ definiujące aktualną architekturę systemu. Dla każdej instrukcji użytkownika zintegrowane środowisko programistyczne procesora Nios II produkuje makroinstrukcję zdefiniowaną w nagłówkowym pliku systemowym. Programista może wywołać taką makroinstrukcję z kodu C/C++ aplikacji tak jak normalną funkcję, bez potrzeby używania asemblera dla osiągnięcia dostępu do instrukcji użytkownika. Środowisko programistyczne jest oparte na kompilatorze GNU C/C++ i standardowym zintegrowanym środowisku Eclipse IDE.

Inną możliwością zwiększenia wydajności systemu jest zastosowanie koprocesorów (akceleratorów sprzętowych). W przeciwieństwie do własnych instrukcji, koprocesory działają jako niezależne moduły logiczne, przyjmujące polecenia z centralnej jednostki sterującej (CPU) procesora przez magistralę Avalon i przetwarzające całe bufory danych i/lub fragmenty algorytmu bez interwencji procesora. Koprocesory mogą osiągnąć wydajność większą o kilka rzędów wielkości w porównaniu z zadaniami wykonywanymi programowo dzięki ich autonomicznej naturze oraz całkowicie sprzętowym wykonywaniu swych funkcji. Mogą one również bezpośrednio wymieniać dane z urządzeniami zewnętrznymi, w tym wykorzystując mechanizm bezpośredniego dostępu do pamięci (DMA).

System w układzie programowalnym może zawierać również niezależne własne komponenty sprzętowe działające asynchronicznie w stosunku do magistrali Avalon nadzorowanej przez CPU. Takie komponenty mogą być odpowiedzialne za zadania wspomagające lub wymagające absolutnej odporności na awarie oprogramowania.

Poniżej poziomu systemowego, akceleratory sprzętowe powinny być projektowane i implementowane w sposób zgodny z procedurami projektowania mechatronicznego [5], co pozwala to na symulację części algorytmu realizowanych sprzętowo razem z pozostałą częścią systemu mechatronicznego w procesie wirtualnego prototypowania. Procedurę projektowania takiego akceleratora sprzętowego przedstawiono w [6].

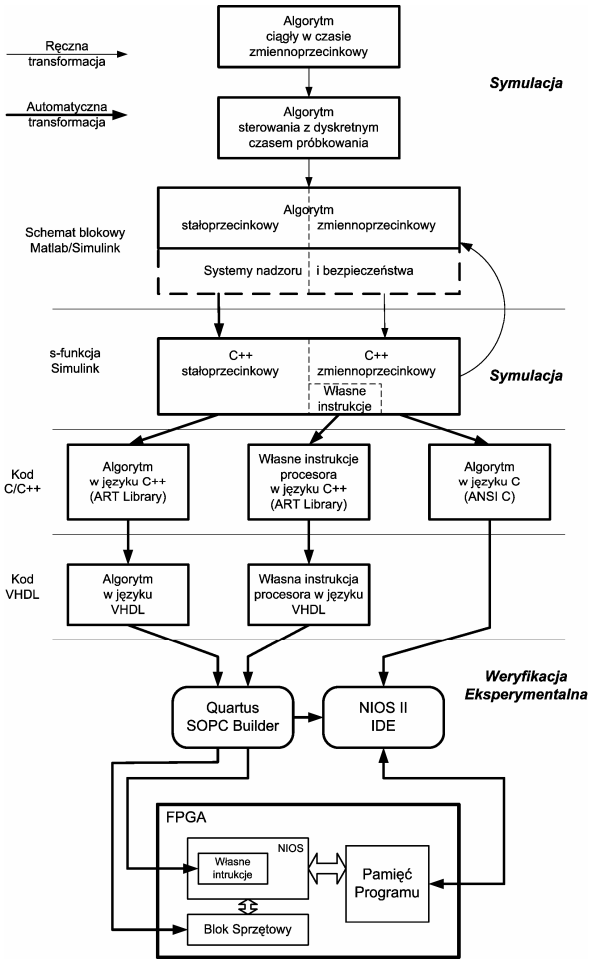
3. Sprzętowo-programowa implementacja algorytmów sterowania

Podział na części realizowane programowo i sprzętowo ściśle zależy od rodzaju algorytmu i musi być rozpatrywane indywidualnie dla każdego przypadku. Powszechnym podejściem jest identyfikacja tych fragmentów algorytmu, które są krytyczne dla wydajności, tj. wymagają najdłuższego czasu przetwarzania i przyspieszanie ich przez implementację sprzętową (akceleratory sprzętowe). Tym niemniej, przed implementacją sprzętową zaleca się określenie możliwego do osiągnięcia wzrostu wydajności, czy jest ona warta zachodu i wykonalna. Mogą wystąpić sytuacje, gdy nieznaczna poprawa wydajności odbywa się za cenę znacznego wzrostu kosztu spowodowanego przeniesieniem funkcji z taniego oprogramowania do drogiego sprzętu [7].

Dobrymi kandydatami do implementacji sprzętowej są te fragmenty algorytmu, które mogą być obliczone przy pomocy operacji stałoprzecinkowych, szybszych i wymagających mniejszej ilości zasobów FPGA. Gdy krytyczne części algorytmu, muszą być obliczane za pomocą arytmetyki zmiennoprzecinkowej, można je przyspieszyć używając własnych specjalizowanych zmiennoprzecinkowych instrukcji mikroprocesora.

Do implementacji sprzętowej nadają się szczególnie regularne algorytmy o prostej i stałej strukturze, operujące na dużych ilościach danych. I przeciwnie, nieregularne algorytmy z wieloma warunkowymi rozgałęzieniami, których wybór zależy od wartości obliczanych podczas wykonywania, wymagają z reguły bardzo dużej ilości zasobów do implementacji sprzętowej i dlatego nadają się bardziej do realizacji programowej na procesorach ogólnego przeznaczenia.

W celu usystematyzowania podejścia do problemu i zautomatyzowania części zadań w Katedrze Robotyki i Mechatroniki AGH opracowano procedurę implementacji sprzętowo-programowej algorytmów sterowania w układach FPGA (rys. 2). Początkowa specyfikacja algorytmu w postaci schematu Simulinka wymaga kilku przekształceń. Pierwszą z nich jest dyskretyzacja w czasie, a drugą dyskretyzacja amplitudy (kwantyzacja). Zastosowanie wyłącznie arytmetyki zmiennoprzecinkowej pozwala na znaczącą redukcję kosztu realizacji algorytmu i czasu jego wykonywania. Tym niemniej należy zachować ostrożność, ponieważ reprezentacja stałoprzecinkowa z trudem radzi sobie z sygnałami o dużej dynamice, których amplitudy zmieniają się w zakresie kilku lub więcej rzędów wielkości. Należy ponadto zwrócić uwagę na wpływ efektów charakterystycznych dla obliczeń stałoprzecinkowych, takich jak przepełnienie i nasycenie. Dlatego po dokonaniu powyższych transformacji działanie algorytmu powinno zostać sprawdzone poprzez symulację. W efekcie tych działań algorytm dzielony jest na dwie części: stało- i zmiennoprzecinkową. Część zmiennoprzecinkowa zostanie zaimplementowana programowo, a część stałoprzecinkowa może zostać zaimplementowana sprzętowo (krytyczne fragmenty) albo programowo (pozostałe fragmenty). W tym miejscu procedury należy wyodrębnić obliczenia niezwiązane bezpośrednio z prawem sterowania, które mogą, lub powinny, np. ze względu na wymaganie odporności na awarie oprogramowania, być wykonywane niezależnie i które zostaną zaimplementowane jako niezależne komponenty sprzętowe, działające niezależnie od magistrali Avalon. W przypadku sterownika dla robota, może to być: system zabezpieczeń.



Rys. 2. Procedura implementacji sprzętowo-programowej algorytmów sterowania w systemie w programowalnym układzie scalonym (SoPC)

Następnie algorytm należy zakodować w języku C/C++, jako tej formie, która będzie później transformowana dalej już całkowicie automatycznie, w przypadku fragmentów przeznaczonych do implementacji programowej przez kompilator dla mikroprocesora Nios II, a w przypadku tych przeznaczonych do implementacji sprzętowej, przez odpowiednie oprogramowanie, do języka opisu sprzętu VHDL (ang. Very high speed integrated circuits Hardware Description Language) i plików do programowania układu FPGA. Kod w języku C/C++ posiada jeszcze trzy zalety. Po pierwsze może, poprzez dodanie odpowiednich interfejsów, zostać jądrem s-funkcji Simulinka, co pozwala symulować fragmenty algorytmu tak, jak inne bloki (transformację do C/C++ można przeprowadzać etapami i łatwo weryfikować jej poprawność). Po drugie, pewne operacje, takie jak manipulacje na bitach, interfejsy urządzeń zewnętrznych i skalowanie danych dużo prościej jest zapisać od razu w języku C/C++ niż składać ze standardowych bloków Simulinka. Po trzecie wreszcie, symulacja kodu pozwala na określenie krytycznych fragmentów algorytmu, czyli tych, które wymagają najdłuższego czasu obliczeń i powinny być zrealizowane sprzętowo. Jeżeli krytyczne fragmenty występują w kodzie zmiennoprzecinkowym, to należy przeprowadzić analizę najczęściej występujących w nich operacji i przeznaczyć je do realizacji sprzętowej za pomocą własnych instrukcji mikroprocesora Nios II. Kod C/C++ dla operacji stałoprzecinkowych jest tworzony automatycznie, z użyciem AR|T Library, ze schematu Simulinka przez opracowany przez autorów generator, produkujący równocześnie gotową s-funkcję z tym kodem [6]. Kod C/C++ części zmiennoprzecinkowej należy utworzyć ręcznie.

Na tym etapie procedury są już podjęte decyzje, które fragmenty algorytmu będą implementowane programowo (zakodowane w ANSI C), a które sprzętowo (zakodowane w C++ z użyciem AR|T Library). Należy teraz dopisać kod C++ opisujący realizację własnych instrukcji mikroprocesora Nios II, przyspieszających wykonywanie kodu zmiennoprzecinkowego.

Następnie należy skompilować kod przeznaczony do implementacji sprzętowej do języka VHDL za pomocą programu AR|T Builder. Końcowym etapem jest połączenie własnych elementów sprzętowych, w języku VHDL z pozostałą częścią systemu (m.in. procesorem) w programie SOPC Builder, w wyniku czego otrzymuje się kompletny opis warstwy sprzętowej sterownika w języku VHDL oraz pliki konfiguracyjne dla kompilatora dla Niosa II. Na podstawie tego kodu VHDL, program Quartus II generuje plik do zaprogramowania układu FPGA. W zaprogramowanym układzie można uruchamiać kod C opisujący część algorytmu realizowaną programowo, skompilowany kompilatorem GNU, a środowisko Eclipse IDE pozwala na kontrolę i uruchamianie programu w czasie rzeczywistym.

4. Przykład zastosowania – sterowanie robotem równoległym

Opisana powyżej procedura została zastosowana do implementacji sterowania równoległym robotem do frezowania [8]. Prawo sterowania, oparte na neuronowym modelu dynamiki odwrotnej manipulatora zostało opisane w pracy [9], wraz ze stabilizującym sterownikiem linearyzującym, opartym na analitycznym modelu strukturalnym manipulatora, zastosowanym w eksperymencie identyfikacyjnym. Oba algorytmy sterowania zostały zaimplementowane na platformie sprzętowej, zbudowanej w standardzie PC104+. Składa się ona z modułu komputera typu PC, wykorzystywanego do programowania układu FPGA, gromadzenia i transmisji danych, oraz głównego modułu z układem FPGA Stratix EP1S30 firmy Altera z peryferiami. Prawie cała elektronika cyfrowa o stosunkowo złożonej strukturze jest zawarta w układzie FPGA, który może być dlatego uważany za system w układzie programowalnym (SoPC).

W pierwszej kolejności algorytmy sterowania zostały zaimplementowane w całości programowo na procesorze Nios II i przeanalizowane w celu zidentyfikowania tych fragmentów, które pochłaniają większość czasu obliczeń oraz określenia ilości operacji zmiennoprzecinkowych, które są wykonywane w trakcie pojedynczego przebiegu algorytmów. Wyniki są pokazane w tabeli 1.

Tab. 1. Ilość poszczególnych operacji zmiennoprzecinkowych w algorytmach sterowania i ich częściach oraz czasy obliczeń

	Sterownik liniearyzujący	Sterownik neuronowy			
		Sieć neuronowa	PID	Kinematyka	Razem
Ilość mnożeń	181	91	30	11	148
Ilość dodawań	59	66	27	7	107
Ilość odwrotności	9	6	0	0	6
Ilość pierwiastków kwadratowych	7	0	0	1	2
Czas wykonywania – realizacja programowa	2.9 ms	1 ms	0.4 ms	0.18 ms	1.8 ms
Czas wykonywania – ze sprzętowymi instrukcjami zmiennoprzecinkowymi	b. d.	25 µs	110 µs	2.3 µs	190 µs
Czas wykonywania – ze sprzętowymi instrukcjami zmiennoprzecinkowymi, kinematyką i PID	27 µs	25 µs	75 ns[a]	182 ns[a]	25 µs

[a] Wykonywane równolegle do procesora Nios II

Następnie fragmenty, które mogą być obliczane z zastosowaniem operacji stałoprzecinkowych (równania kinematyki i algorytm PID) zostały zaimplementowane sprzętowo z wykorzystaniem procedury opisanej w [6] tak, że wykonywane są w jednym cyklu zegara. Dla pozostałej części praw sterowania zostały dodane sprzętowe rozszerzenia ALU, tworząc nowe instrukcje procesora

Nios II umożliwiające szybkie wykonywanie operacji zmiennoprzecinkowych, które najczęściej występują w algorytmach: mnożenia, dodawania, odwrotności i pierwiastka kwadratowego, z zastosowaniem technik opisanych w [10]. Zasoby zużyte do implementacji oraz osiągnięte przyspieszenia czasu obliczeń prezentuje tablica 2.

Tab. 2. Wykorzystane zasoby i przyspieszenie obliczeń dla sprzętowej implementacji operacji zmiennoprzecinkowych, równań kinematyki i PID; LE – ilość elementów logicznych, DSP – ilość bloków DSP

	LE	DSP	Czas wykonania (sprzętowo)	Czas wykonania (programowo)
Mnożenie	41	8	18 ns	2.3 µs
Dodawanie	818	0	55 ns	2.7 µs
Odwrotność	1697	0	200 ns	2.8 µs
Pierwiastek kwadratowy	1149	0	182 ns	12 µs
Kinematyka	1292	78	182 ns[a]	0.18 ms
PID	2150	0	75 ns[a]	0.5 ms

[a] Wykonywane równolegle do procesora Nios II

Efekty rozszerzania ALU procesora Nios II o kolejne sprzętowe instrukcje zmiennoprzecinkowe dla sterowania linearyzującego pokazane są w tablicy 3.

Tab. 3. Czas obliczeń sterowania linearyzującego w FPGA dla różnych realizacji operacji zmiennoprzecinkowych; PID zaimplementowany sprzętowo, na zewnątrz procesora Nios II

Mnożenie	Dodawanie	Odwrotność	Pierwiastek kwadratowy	Czas wykonania (µs)
programowe	programowe	programowa	programowy	2500
programowe	programowe	programowa	sprzętowy	1800
programowe	programowe	sprzętowa	sprzętowy	350
programowe	sprzętowe	sprzętowa	sprzętowy	110
sprzętowe	sprzętowe	sprzętowa	sprzętowy	27

Tablica 4 podaje dane o wykorzystaniu zasobów układu FPGA i czasie wykonywania obu algorytmów sterowania zaimplementowanych wyłącznie programowo, programowo ze sprzętową realizacją operacji zmiennoprzecinkowych oraz dodatkowo z równaniami kinematyki i częścią PID zaimplementowanymi sprzętowo, poza procesorem Nios II.

Tab. 4. Czas wykonania i wykorzystane zasoby dla różnych, sprzętowo-programowych implementacji sterowników w FPGA; LE – ilość elementów logicznych, DSP – ilość bloków DSP

	Sterownik linearyzujący			Sterownik neuronowy		
	Czas wykonania	Zasoby		Czas wykonania	Zasoby	
		LE	DSP		LE	DSP
Realizacja programowa	2.5 ms	9023	10	1.8 ms	9023	10
Ze sprzętowymi instrukcjami zmiennoprzecinkowymi	b.d.	b.d.	b.d.	190 µs	11728	18
Ze sprzętowymi instrukcjami zmiennoprzecinkowymi, kinematyką i PID	27 µs	15170	96	25 µs	15170	96

Cała aplikacja zajmuje ok. 46% elementów logicznych (LE), 100% bloków DSP i ok. 33% pamięci układu EP1S30, pozostawiając wystarczającą ilość niewykorzystanych zasobów dla przyszłych modyfikacji algorytmu sterowania. Pracując z zegarem 55 MHz system przeprowadza obliczenia w 25 µs, co pozwala na maksymalną częstotliwość próbkowania równą 40 kHz. Przy obecnej częstotliwości 1 kHz zapas czasu jest wystarczająco duży dla zaimplementowania dużo bardziej złożonych technik sterowania. Te same algorytmy wymagają porównywalnego czasu obliczeń na mikroprocesorach PowerPC i Celeron z zegarem 400 MHz (tablica 5).

Tab. 5. Czas obliczeń sterowników w ostatecznej postaci w FPGA i na mikroprocesorach ogólnego przeznaczenia

	FPGA Nios II, 55 MHz	PowerPC, 400MHz	Celeron, 400MHz
Sterownik linearyzujący	27 µs	34 µs	34 µs
Sterownik neuronowy	25 µs	29 µs	27 µs

5. Podsumowanie

W artykule zostały przedstawione problemy związane z implementacją algorytmów sterowania w układach FPGA. Proponowanym rozwiązaniem jest zastosowanie systemu w układzie programowalnym (SoPC) z semiautomatyczną implementacją koprocesorów i sprzętowo-programową realizacją algorytmów. Procedura została z powodzeniem zastosowana do problemu implementacji neuronowego sterownika robota do frezowania o trzech stopniach swobody. Podczas akwizycji danych do zbioru uczącego dla sieci neuronowej został zastosowany wstępny sterownik stabilizujący, oparty na analitycznym modelu strukturalnym manipulatora. Artykuł prezentuje implementację obydwu sterowników.

Platforma sprzętowa do implementacji jest oparta na nowoczesnym układzie FPGA. Algorytmy sterowania zostały zaimplementowane w sposób sprzętowo-programowy, tzn. pewna ich część została zrealizowana sprzętowo, a pozostała część za pomocą programu wykonywanego przez mikroprocesor RISC Nios II wprogramowany w układ FPGA. Opracowane sterowniki są przykładami systemu w układzie programowalnym, co pozwala na wysoki stopień integracji i elastyczności - system może być łatwo modyfikowany np. w celu optymalizacji dla innego algorytmu sterowania albo rozszerzenia funkcjonalności, poprzez przeprogramowanie układu FPGA, bez zmian płyty drukowanej. Zastosowana metodologia pozwoliła na obniżenie kosztów skrócenie czasu implementacji.

Prawidłowo zaprojektowany i podzielony system w układzie programowalnym może osiągnąć lepsze osiągi niż mikroprocesor ogólnego przeznaczenia pracujący ze znacznie szybszym zegarem. Zastosowanie specjalizowanych akceleratorów sprzętowych i własnych, specjalizowanych instrukcji CPU skróciło czas obliczeń o dwa rzędy wielkości w porównaniu z realizacją całkowicie programową.

Praca naukowa finansowana ze środków Komitetu Badań Naukowych jako projekt badawczy 4 T07B 077 26

6. Literatura

[1] M. Wegrzyn, M. A. Adamski and J. L. Monteiro, The application of reconfigurable logic to controller design, Control Engineering Practice, Vol. 6, 1998, pp. 879-887
[2] Petko M., Wybrane zagadnienia projektowania mechatronicznego, Rozprawy Monografie, nr 153, Wyd. AGH, Kraków 2005
[3] http://www.altera.com
[4] http://www.xilinx.com
[5] R. Iserman, Mechatronic Systems: Fundamentals, Springer, 2003
[6] M. Petko and G. Karpiel, Semi-Automatic Implementation of Control Algorithms in ASIC/FPGA, in 2003 IEEE Conference on Emerging Technologies and Factory Automation: Proceedings, vol. 1, pp. 427-433
[7] M.D. Edwards, J. Forrest and A.E. Whelan, Acceleration of software algorithms using hardware/software co-design techniques, J. of Systems and Architecture, Vol. 42, 1996/97, pp. 697-707
[8] M. Petko and G. Karpiel, Mechatronic design of a parallel manipulator for milling, in Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 759-764
[9] M. Petko and G. Karpiel, Neural Control of a Parallel Robot – Design and Implementation in FPGA, submitted for 4th IFAC-Symposium on Mechatronic Systems
[10] A. R. Omondi, Computer Arithmetic Systems. Prentice Hall, 1994

Artykuł recenzowany