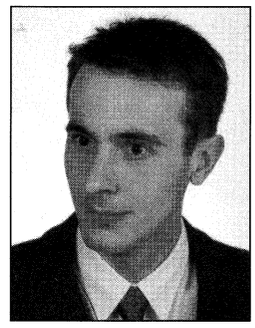
Maciej PETKO
AKADEMIA GÓRNICZO-HUTNICZA
KATEDRA ROBOTYKI I DYNAMIKI MASZYN

Implementacja algorytmów sterowania w układach ASCI/FPGA



Dr inż. Maciej Petko

Urodził się 12 kwietnia 1968 roku w Krakowie. W 1991 roku uzyskuje dyplom magistra inżyniera elektronika na Wydziale Elektrotechniki, Automatyki i Elektroniki Akademii Górniczo-Hutniczej. Od 1992 r. pracuje w Akademii Górniczo-Hutniczej jako asystent, a od 1999 r. jako adiunkt w Katedrze Robotyki i Dynamiki Maszyn. W 1999 r. uzyskuje stopień doktora nauk technicznych w dyscyplinie Automatyka i Robotyka, ze specjalnością Mechatronika. Jego zainteresowania skupiają się na zagadnieniach prototypowania i implementacji algorytmów przetwarzania sygnałów, głównie w sterowaniu i diagnostyce technicznej. Jest autorem 17 publikacji z tego zakresu.

Streszczenie

W artykule przedstawiono problemy związane z prototypowaniem i implementacją algorytmów sterowania, ze szczególnym uwzględnieniem sytuacji, gdy część sprzętowa sterownika oparta jest na układach ASIC/FPGA. Dla takiego przypadku opracowano metodologię implementacji, którą zweryfikowano poprzez zastosowanie do problemu sterowania elastycznym ramieniem robota. Wykazano, że procedura taka realizuje jednocześnie ideę szybkiego prototypowania na docelowej platformie sprzętowej. Przedstawiono szczegóły tej procedury wraz z narzędziami użytymi do jej przeprowadzenia i osiągnięte wyniki.

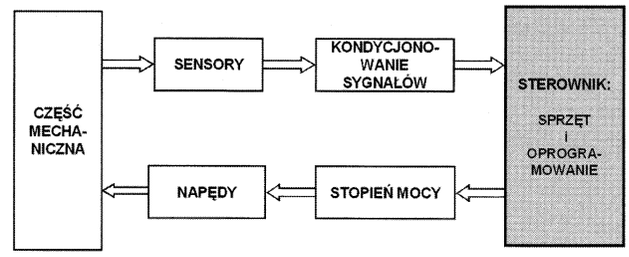
Abstract

In the paper problems with prototyping and implementation stages during development of control algorithms are presented with emphasis placed on ASIC / FPGA based hardware platform for controller. For this case, a methodology of implementation is formulated and validated by practical application to the problem of flexible robot arm control. It is shown, that the same procedure allows for fulfilment of an idea of fast prototyping on target hardware. Details of the procedure are presented along with the tools used and results obtained during its realization.

WPROWADZENIE

Na obecnym etapie rozwoju techniki, sterownik powinien być traktowany jako element systemu mechatronicznego, a opracowywanie sterownika jako proces mechatroniczny. Oznacza to, że podczas opracowywania sterownika powinien zostać osiągnięty synergizm poprzez integrację mechaniki, elektroniki, automatyki oraz informatyki i równorzędne traktowanie tych dziedzin [1].

Struktura typowego systemu mechatronicznego przedstawiona jest na rys. 1. Zawiera ona składa część mechaniczną wymuszaną za pomocą napędów. Sensory umożliwiają obserwację stanu części mechanicznej. Cały system jest nadzorowany przez algorytm sterowania zaimplementowany i wykonywany na pewnej platformie sprzętowej. Tą platformą może być [2]:
– komputer przemysłowy,
– sterownik autonomiczny (ang. embedded controller),
– sterownik oparty na układach ASIC / FPGA.



Rys. 1 Typowa struktura systemu mechatronicznego.

Wybór platformy sprzętowej sterownika zależy generalnie od przewidywanej wielkości produkcji. Dla produkcji jednostkowej stosowane są przeważnie komputery przemysłowe, dla produkcji średnioseryjnej sterowniki autonomiczne, zaś sprzęt oparty na układach specjalizowanych (ASIC) stosowany jest w przypadku produkcji wielkoseryjnej. Wybór taki dokonywany jest ze względu na szereg własności, którymi dla tej ostatniej platformy są: wysoki stopień integracji, niezawodność, duże możliwości optymalizacji, niski jednostkowy koszt wytworzenia, ale równocześnie mała elastyczność i duży koszt implementacji. Cechy komputerów przemysłowych są dokładnie przeciwne, natomiast sterowniki autonomiczne posiadają właściwości pośrednie.

Specjalnym przypadkiem platformy sprzętowej są systemy przeznaczone do szybkiego prototypowania. Aby mogły być wykorzystywane w szerokim zakresie zastosowań, muszą posiadać dużą moc obliczeniową i być elastyczne, co pociąga za sobą wysokie ceny. Ich inną, często spotykaną niedogodnością jest znaczna odmienność od docelowej platformy sprzętowej sterownika, co powoduje problemy podczas implementacji.

Powszechnie występującym etapem podczas opracowywania sterownika jest wykonywane na specjalnym sprzęcie szybkie prototypowanie, które pozwala na znalezienie i usunięcie błędów oraz eksperymentalne, precyzyjne dostrojenie parametrów algorytmu sterowania. Po tym etapie następuje implementacja.

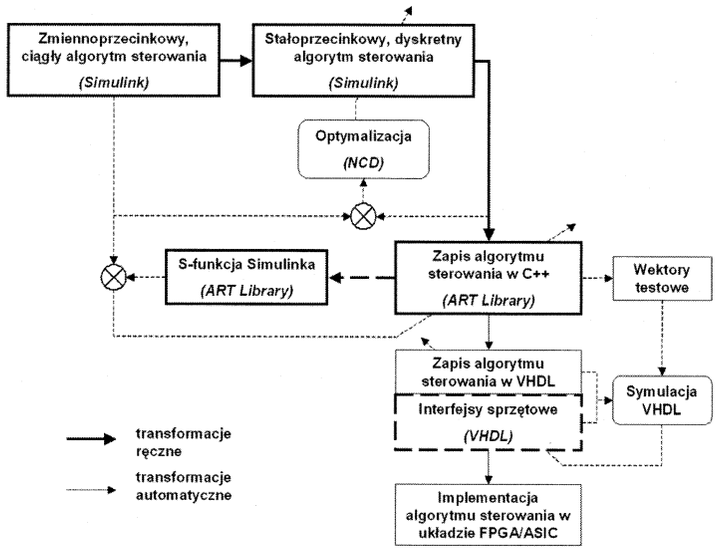
Problemy występujące podczas implementacji doprowadziły do powstania idei szybkiego prototypowania na docelowym sprzęcie, inaczej mówiąc integracji procesu prototypowania i implementacji. Idea ta została zrealizowana na komputerze przemysłowym [3], ale wciąż oczekiwała na urzeczywistnienie na sprzęcie opartym na układach FPGA/ASIC, dla którego oczekiwane korzyści są znacznie większe. Niestety, jak dotąd nie istniała nawet zadowalająca procedura samego procesu implementacji.

OPIS PROBLEMU

Algorytm sterowania, który ma zostać zaimplementowany w układach FPGA/ASIC jest zwykle ciągły w czasie, wykorzystuje ciągłe wartości sygnałów i opisany jest za pomocą równań matematycznych, albo za pomocą schematu blokowego [2]. Opis taki nie nadaje się do bezpośredniego zastosowania.

Przede wszystkim potrzebna jest dyskretyzacja w czasie. Drugą znaczącą transformacją jest dyskretyzacja amplitudy (kwantyzacja). Zastosowanie wyłącznie arytmetyki stałoprzecinkowej pozwala na znaczną redukcję kosztu realizacji algorytmu sterowania.

Programy używane do syntezy układów FPGA i ASIC akceptują ich opis w specjalnych językach (ang. HDL – Hardware Description Language), rzadko znanych inżynierom automatykom. Co więcej, kodowanie w językach HDL jest procesem czasochłonnym i podatnym na błędy [4]. Z używaniem takiej notacji związane są również problemy z testowaniem i weryfikacją poprawności działania i osiągów algorytmu. Istniejące narzędzia pozwalają na dokładną symulację elektroniki na poziomie bramek, wspierają, w pewnym stopniu, symulację na poziomie algorytmu, ale sprawdzenie jakości sterowania, tj. równoczesna symulacja sterownika i sterowanego obiektu jest jak dotąd niemożliwa. Z drugiej strony, programy powszechnie używane podczas projektowania algorytmów sterowania, takie jak Simulink, pozwalają w stosunkowo prosty sposób symulować urządzenie mechaniczne równocześnie ze sterowaniem, opisanym za pomocą schematu blokowego, ale nie umożliwiają symulacji algorytmu zapisanego przy użyciu języków HDL. Dlatego sprawdzenie zgodności opisu sterownika w języku HDL z pierwotną formą w postaci schematu blokowego wydaje się być jednym z najtrudniejszych problemów podczas implementacji sprzętowej.



Rys. 2 Procedura implementacji algorytmu sterowania w układach ASIC/FPGA

Z powyższych okoliczności wynika potrzeba systematycznego podejścia do implementacji sterowania w układach FPGA/ASIC.

IMPLEMENTACJA ALGORYTMÓW STEROWANIA W UKŁADACH FPGA/ASIC

W celu rozwiązania, a przynajmniej zmniejszenia opisanych problemów, w Katedrze Robotyki i Dynamiki Maszyn AGH jest opracowywana procedura implementacji algorytmów sterowania w układach FPGA/ASIC. Poniżej zostanie opisana jej obecna postać wraz z przykładem zastosowania.

Głównymi celami podczas tworzenia tej procedury było ograniczenie ilości i zakresu dokonywanych ręcznie transformacji algorytmu, zastosowanie, tam gdzie to możliwe, komercyjnie dostępnego oprogramowania oraz próba wyraźnego wyodrębnienia kodu HDL nie należącego bezpośrednio do algorytmu sterowania. Pierwszy cel wynika z kosztów przekształceń wykonywanych ręcznie – są one czasochłonne i są źródłem częstych błędów. Drugi cel zmniejsza koszty tworzenia procedury i pozwala na natychmiastowe korzystanie z udogodnień nowych wersji oprogramowania komercyjnego bez dodatkowych nakładów na adaptację. Osiągnięcie ostatniego z wymienionych celów tworzy potencjalne możliwości realizacji szybkiego prototypowania na docelowym sprzęcie.

Procedura, pokazana schematycznie na rys. 2, składa się z kilku etapów. Niektóre z nich są realizowane ręcznie, a pozostałe automatycznie, przez odpowiednie oprogramowanie. Pierwsze przekształcenie do postaci stałoprzecinkowej, dyskretnej w czasie jest realizowane w Simulinku, z wykorzystaniem standardowej biblioteki stałoprzecinkowej Fixed-Point Blockset [5]. Pozwala to na zbadanie dynamiki sygnałów i efektów obliczeń wykonywanych z przyjętą precyzją i zakresem reprezentacji sygnałów oraz charakterystykami nasycenia lub przepełnienia. Transformacja ta musi być wykonana ręcznie, z tym że wartości parametrów mogą być dopasowane z zastosowaniem biblioteki Nonlinear Control Design Toolbox (NCD) [6], starannie dobranej funkcji cel i metody optymalizacji, tak aby uzyskać jakość sterowania jak najbardziej zbliżoną do uzyskiwanej z zastosowaniem ciągłego w czasie sterowania zmiennoprzecinkowego. Środowisko Simulinka umożliwia łatwą ocenę nowej postaci sterownika.

Następny, także realizowany ręcznie etap wykorzystuje oprogramowanie firmy Frontier Design. A|RT Library [7] jest biblioteką dla kilku popularnych kompilatorów C++, dostarczającą klasy stałoprzecinkowe, która umożliwia emulację obliczeń stałoprzecinkowych w programach pisanych w języku C++. A|RT Builder [8] umożliwia automatyczną syntezę kodu w języku VHDL lub Verilog (są to jedne z najbardziej popularnych języków HDL) z programu w C++, pisanego z użyciem biblioteki stałoprzecinkowej. A|RT Library używa takich samych typów co bloki Simulinka z Fixed-Point Blockset, tak więc możliwe jest wzajemnie jednoznaczne odwzorowanie między schematem blokowym a kodem C++, co ułatwia ten etap procedury i redukuje ilość możliwych do popełnienia błędów.

Po dodaniu kodu interfejsu Simulinka do algorytm sterowania w języku C++ można utworzyć tzw. s-funkcję. S-funkcja definiuje działanie nowego bloczka, który może zostać elementem dowolnego schematu blokowego w Simulinku. Równocześnie źródło tej samej s-funkcji może być zsyntetyzowane przez A|RT Builder do kodu VHDL. Umożliwia to symulację tego samego kodu sterownika, który jest używany do syntezy układu FPGA/ASIC, razem z modelem układu sterowanego.

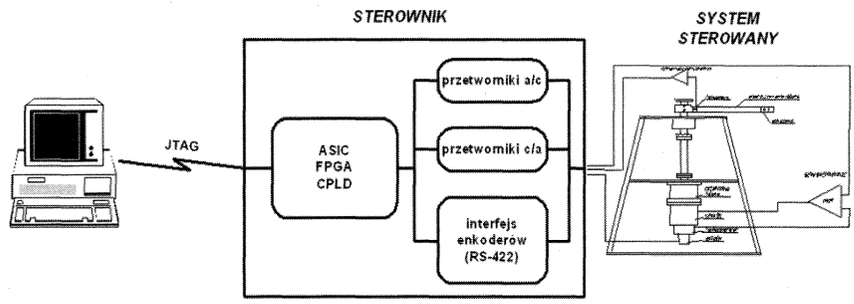
Ważną zaletą A|RT Buildera jest jego „przeźroczystość", która oznacza zależność generowanego kodu VHDL tylko i wyłącznie od stylu programowania w C++. Dzięki wyborowi odpowiedniego stylu, topologia otrzymywanego układu elektronicznego może być optymalizowana pod względem szybkości działania lub rozmiaru poprzez automatyczne syntezowanie bardziej złożonych metod zarządzania przepływem danych, jak przetwarzanie potokowe czy współdzielenie zasobów. Dokładnie przy tym wiadomo jaki kod w VHDL jest generowany dla każdej instrukcji i wyrażenia w C++, zachowywane są także wszystkie identyfikatory z C++. Pozwala to na zachowanie pełnej kontroli nad generowanym kodem w VHDL, uwalniając od konieczności ręcznego kodowania. Co więcej, możliwe jest łączenie otrzymanego zapisu algorytmu sterowania w VHDL z innym kodem w tym języku, np. opisującym interfejsy sprzętowe z urządzeniami peryferyjnymi, takimi jak przetworniki analogowo-cyfrowe i cyfrowo-analogowe czy też synchronizację czasową obliczeń.

Tworzy to możliwości realizacji idei szybkiego prototypowania na docelowej platformie sprzętowej przez stosowanie niezmiennej części organizacyjnej i „peryferyjnej" kodu VHDL i zmianę oraz badanie działania części opisującej algorytm sterowania.

Wygenerowany zapis algorytmu sterowania w VHDL, razem z opisem niezbędnych interfejsów urządzeń peryferyjnych jest następnie kompilowany w celu otrzymania listy połączeń na poziomie bramek, która opisuje architekturę układu scalonego przy użyciu bramek, przerzutników i innych elementów podstawowych dostępnych w danej technologii. Później lista połączeń wymaga dopasowania, pod-

czas którego fizyczne zasoby konkretnego układu scalonego są alokowane dla każdego elementu podstawowego z listy. W trakcie opisywanych badań do kompilacji był używany program Synopsys FPGA Express [9], a do dopasowywania Altera MAX+plus II [10].

W przypadku konieczności dokonania ręcznych modyfikacji kodu w VHDL potrzebna jest metoda sprawdzania jego poprawności. Analiza może być przeprowadzona za pomocą symulatora VHDL. Tworzenie i weryfikacja wektorów testowych dla symulatorów VHDL są wspomagane przez A|RT Library.



Rys. 4 Platforma sprzętowa sterownika

PRZYKŁAD ZASTOSOWANIA – ELASTYCZNE RAMIĘ ROBOTA

Opisana powyżej procedura implementacji została zrealizowana na przykładzie problemu sterowania modelem elastycznego ramienia robota. Schemat i zdjęcie stanowiska laboratoryjnego przedstawia rys. 3.

Składa się ono z elastycznej stalowej belki, zamontowanej na wale napędzanym silnikiem prądu stałego przez przekładnię harmoniczną. Silnik jest sterowany przez serwowzmacniacz, wykorzystujący sygnał tachogeneratora sprzęgniętego z wałem silnika jako sprzężenie zwrotne. Kąt obrotu silnika jest mierzony optycznym przetwornikiem obrotowo-impulsowym. Ugięcie elastycznej belki jest wyznaczane przez pomiar odkształcenia jej powierzchni za pomocą mostka tensometrycznego. Zmiana obciążenia jest emulowana poprzez mocowanie na końcu belki odpowiednich ciężarków.

Platforma sprzętowa (rys. 4), na której implementowano sterowanie składała się z układu scalonego z rodziny FLEX10K Altery [13], programowanego przez standardowy interfejs JTAG, przetworników analogowo-cyfrowych, cyfrowo-analogowych, odbiorników sygnałów przetworników obrotowo-impulsowych, kilku przycisków i diod elektroluminescencyjnych.

Niezbędne interfejsy i generatory podstawy czasu zostały zaprojektowane i napisane w języku VHDL tak, aby można je było połączyć z wygenerowanym kodem algorytmu sterowania.

Wyjściowy algorytm sterowania, został otrzymany przy zastosowaniu metody szybkiego prototypowania i specjalizowanego sprzętu opartego na procesorach sygnałowych [3]. Odpowiadający mu stałoprzecinkowy, dyskretny w czasie sterownik, zoptymalizowany pod względem zakresu, rozdzielczości, ilości operacji i wartości parametrów został sprawdzony pod względem funkcjonalnej zgodności z postacią wyjściową.

W następnym kroku stałoprzecinkowa forma sterowania została zakodowana w języku C++ z użyciem biblioteki A|RT Library. Kod sterownika w C++ został uzupełniony tak, że powstała s-funkcja. To umożliwiło symulację tego kodu w zamkniętej pętli sterowania i sprawdzenie poprawności kodowania. Kolejne operacje: kompilacji źródła w VHDL, otrzymanego po syntezie za pomocą A|RT Buildera, i dopasowania listy połączeń dostarczyły plików do programowania układu FLEX10K, co prawie zakończyło procedurę implementacji. Przeprowadzono jeszcze tylko weryfikację eksperymentalną – odpowiedź układu w zamkniętej pętli sprzężenia zwrotnego na skokową zmianę pozycji zadanej o 30^0 jest pokazana na rys. 4.

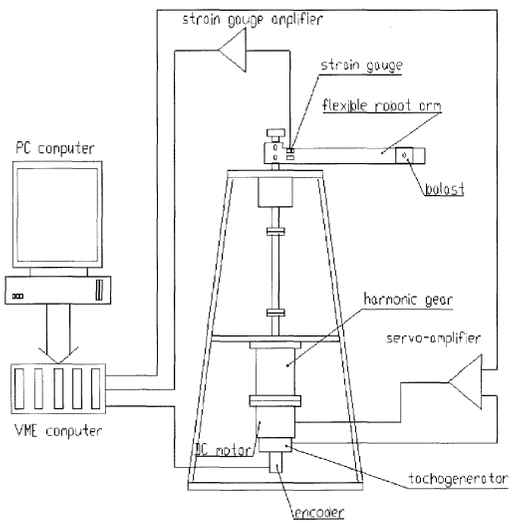
WNIOSKI

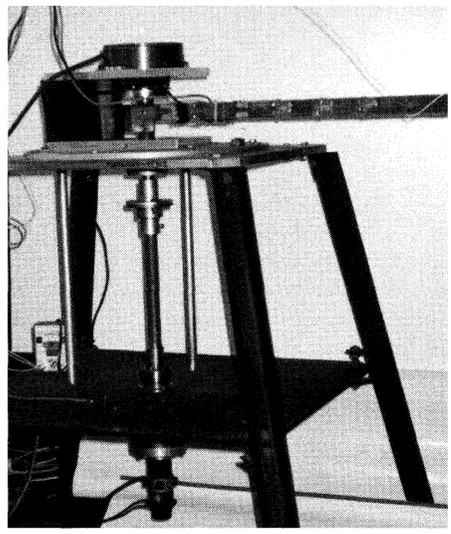
Zaproponowana metodologia implementacji algorytmów sterowania na sprzęcie opartym na układach FPGA/ASIC została w pełni i z powodzeniem zrealizowana. Pozwala ona na automatyzację najbardziej czasochłonnych, sprzyjających powstawaniu błędów i wymagających udziału specjalistów elektroników faz implementacji, pozostawiając pełną kontrolę nad każdą operacją. Przedstawiona procedura może być z łatwością zintegrowana ze standardowymi metodami projektowania układów sterowania.

Przedstawiono i przetestowano nowe możliwości weryfikacji, pozwalające na symulację modelu systemu sterowanego razem z kodem sterownika, który był użyty do syntezy układu FPGA.

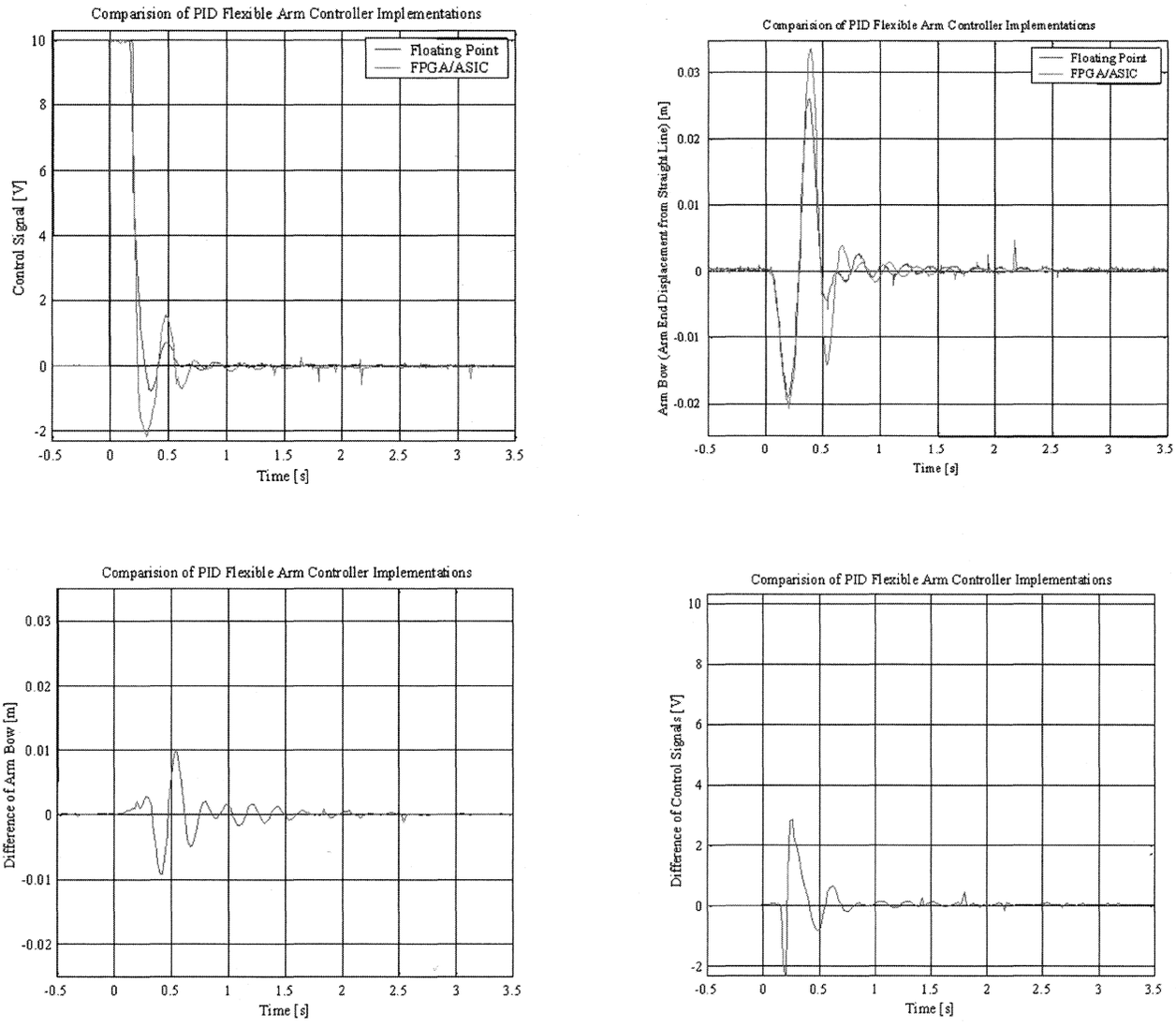
Zaprezentowane podejście może być zaklasyfikowane jako szybkie prototypowanie na docelowej platformie sprzętowej, jednakże widać potrzebę tak dalszej automatyzacji jak i integracji procesów implementacji i prototypowania układów sterowania.

Obecnie w Katedrze Robotyki i Dynamiki Maszyn AGH dobiegają końca prace nad automatycznym generowaniem opisu algorytmu sterowania w języku C++, wykorzystującego stałoprzecin-





Rys. 3 Model elastycznego ramienia robota



Rys. 4 Porównanie implementacji sterownika elastycznego ramienia robota; przebiegi: sygnału sterującego (górny lewy wykres), ugięcia belki (górny prawy wykres), różnicy między sygnałami sterującymi (dolny lewy wykres) oraz różnicy między ugięciami belki (dolny prawy wykres)

kową bibliotekę A|RT Library, na podstawie schematu blokowego Simulinka.

LITERATURA

[1] D. M. Auslander, C. J. Kempf, *Mechatronics*, Prentice Hall, 1996

[2] M. Petko, Realizacja produktów mechatronicznych, W: Projektowanie mechatroniczne (T. Uhl (Ed.)), Wyd. KRiDM AGH, Krakow, 1999

[3] W. Szwabowski, M. Bogacz, T. Uhl, Szybkie prototypowanie sterowników w systemach czasu rzeczywistego, II Krajowa Konferencja: Metody i Systemy Komputerowe, Kraków, Oct. 25-27, 1999

[4] T. Uhl, Z. Mrozek, M. Petko, Rapid control prototyping for flexible arm, 1-st IFAC Conference on Mechatronic Systems, Preprint, vol. 2, pp. 489-494, Darmstadt, 2000

[5] Fixed-Point Blockset User's Guide, The MathWorks Inc., Natic, 1999

[6] Nonlinear Control Design Blockset User's Guide, The MathWorks Inc., Natic, 1997

[7] A|RT Library User's & Reference Manual, Frontier Design Inc., Danville, 2000

[8] A|RT Builder User's and Reference Documentation, Frontier Design Inc., Danville, 2000

[9] FPGA Compiler II/FPGA Express VHDL Reference Manual, Synopsys, Inc., 1999

[10] MAX+PLUS II Getting Started. Altera Corp., San Jose, 1997.

[11] S. Sjoholm, L. Lindh, VHDL for Designers, Prentince Hall, 1997

[12] D. L. Perry, VHDL, McGraw-Hill, 1998.

[13] FLEX 10K Embedded Programmable Logic Family. Altera Corp., 1999