**Adam ZIĘBIŃSKI, Michał GLINIANOWICZ, Grzegorz LACHOWSKI**
POLITECHNIKA ŚLĄSKA, INSTYTUT INFORMATYKI

# Implementacja regulatora PID w strukturze FPGA

**Dr inż. Adam ZIĘBIŃSKI**

Ukończył studia w Instytucie Informatyki na Wydziale Automatyki Elektroniki i Informatyki Politechniki Śląskiej w 1996 r. Na tym samym wydziale rozpoczął pracę jako asystent w 1996 r. Pracę doktorską obronił w 2002 r. Obecnie pracuje na stanowisku adiunkta w Instytucie Informatyki w Zakładzie Urządzeń Informatyki. Jego zainteresowania to sprzętowo-programowe projektowanie koprocesorów problemowo zorientowanych z wykorzystaniem układów reprogramowalnych VLSI.

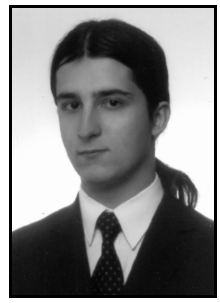
*e-mail: adam.ziebinsk@polsl.pl*

**Michał GLINIANOWICZ**

Student IV roku Informatyki na Wydziale Automatyki Elektroniki i Informatyki w Instytucie Informatyki Politechniki Śląskiej. Jego zainteresowania obejmują zagadnienia z dziedziny tworzenia aplikacji internetowych.

*e-mail: m.glinianowicz@wp.pl*

**Grzegorz LACHOWSKI**



Student IV roku Informatyki na Wydziale Automatyki Elektroniki i Informatyki w Instytucie Informatyki Politechniki Śląskiej (specjalność Oprogramowanie Systemowe). W 2007 studiował Informatykę na Politecnico di Milano (Politechinka Mediolańska) w ramach stypendium Socrates-Erasmus. Obszarem zainteresowań jest m.in. programowanie urządzeń mobilnych oraz języki opisu sprzętu HDL.

*e-mail: g.lachowski@gmail.com*

**Streszczenie**

W pracy przedstawiono sposób realizacji regulatora PID na matrycy FPGA. Omówiono implementację, wyniki symulacji stworzonego teoretycznego projektu oraz sposób jego fizycznej realizacji. Przedmiotem zainteresowania są możliwości wykonania w postaci cyfrowej jednego z typowych układów regulujących procesami, które dostosowują sygnał sterujący urządzeniem na podstawie aktualnej wartości wyjściowej obiektu. Zastosowana matryca jest rekonfigurowalnym układem ogólnego przeznaczenia, który po zaprogramowaniu pozwoli użytkownikowi wykorzystywanie jej jako regulatora PID bez konieczności nabywania specjalizowanego urządzenia.

**Słowa kluczowe**: regulator PID, FPGA.

## The FPGA implementation of the PID controller

**Abstract**

This article contains the method of realization the PID controller using the FPGA array. There were described the implementation, results of the simulation of the theoretical project and the methods of physical realization. The main topic is realizability of the digital version of the typical process controller, which adjusts the command signal basing on the output of the device. The used array is reconfigurable general purpose circuit, which after being programmed one can use as a PID controller without necessity of purchasing the specialistic device.

**Keywords**: PID controller, FPGA, implementation.

## 1. Wstęp

Regulatory PID są od ponad 60 lat jednymi z najpowszechniej stosowanych układów w automatyce przemysłowej [1]. Ich zasada działania opiera się na generowaniu sygnału sterującego o wartości ustalanej na podstawie aktualnego stanu obiektu sterowanego. Urządzenia tego typu stosuje się do regulacji najróżniejszych procesów ciągłych, w szczególności do regulacji temperatury, przepływu i ciśnienia (np. kontrolery przepływu czynników roboczych, sterownie reaktorami chemicznym, węzły ciepłownicze itp.).

## 2. Algorytm regulatora PID

W pracy nad realizacją regulatora zastosowano prosty algorytm PID [2], sumujący wyniki uzyskiwane od poszczególnych modułów: proporcjonalnego, całkującego i różniczkującego.

Wartości *pGain*, *iGain* oraz *dGain* są to nastawy odpowiednio członu proporcjonalnego, całkującego oraz różniczkującego regulatora, dobierane indywidualnie dla każdego sterowanego urządzenia podczas tzw. strojenia regulatora PID.

Wynik otrzymany z modułu proporcjonalnego (zmienna *p_result*) przedstawia poniższy wzór:

$$p\_result = pGain \cdot error, \tag{1}$$

gdzie *error* jest wartością uchybu obliczaną z poniższego wzoru:

$$error = target - current, \tag{2}$$

gdzie: *target* – wartość zadana, *current* – wartość bieżąca.

Wynik z modułu całkującego (zmienna *i_result*) przedstawia poniższy wzór:

$$i\_result = iGain \cdot iState, \tag{3}$$

przy czym *iState* jest skumulowaną wartością dotychczasowych uchybów (*error*), która ulega nasyceniu do określonego przez użytkownika maksimum oraz minimum.

Wynik z modułu różniczkującego (zmienna *d_result*) przedstawiono:

$$d\_result = dGain \cdot (error - lastError), \tag{4}$$

gdzie *lastError* jest wartością uchybu z poprzedniego próbkowania.

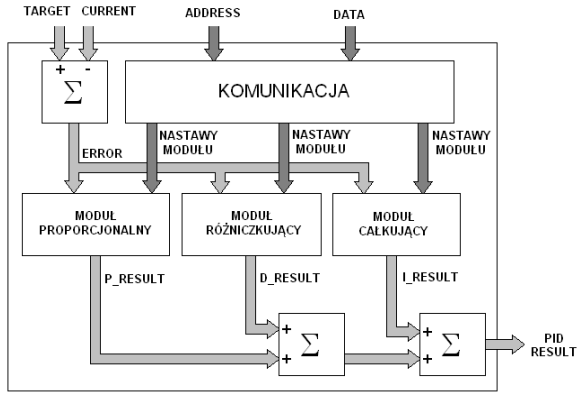
Wynikiem działania zastosowanego algorytmu PID jest wartość:

$$result = p\_result + i\_result + d\_result \tag{5}$$

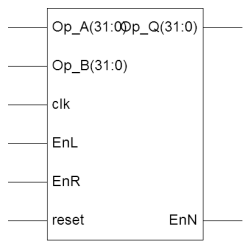
## 3. Projekt regulatora

Regulator PID zbudowano z wykorzystaniem trzech modułów: proporcjonalnego, całkującego oraz różniczkującego. Do każdego z nich wpisywana jest wartość uchybu (error) będąca wynikiem z subtraktora, oraz stosowna nastawa wysłana z modułu komunikacji. Rezultaty otrzymane z trzech głównych modułów są dodawane w dwóch sumatorach dwuwejściowych. Schemat regulatora PID przedstawia rys. 1.

W zaimplementowanym regulatorze PID wykorzystywane są trzy rodzaje modułów jednostki FPU: dodający, odejmujący i mnożący [3]. Wszystkie z nich komunikują się z otoczeniem w jednakowy sposób (rys. 2).




Rys. 1. Schemat modułów regulatora PID
Fig. 1. The schematic of the PID controller modules




Rys. 2. Moduł jednostki zmiennoprzecinkowej
Fig. 2. Typical FPU module

Znaczenie poszczególnych sygnałów jest następujące:

`Op_A, Op_B` – 32-bitowe magistrale wejściowe dla operandów operacji matematycznych,

`Op_Q` – 32-bitowa magistrala wyjściowa zwracająca wynik operacji matematycznej,

`clk` – wejście taktujące,

`EnL, EnR` – sygnały rozpoczynające (gdy oba są w stanie wysokim podczas narastającego zbocza zegara) odczyt operandów z magistral oraz pracę modułu FPU,

`EnN` – wyjście sygnalizujące zakończenie obliczeń jednostki FPU przez wystawienie stanu wysokiego na jeden okres sygnału taktującego,

`reset` – sygnał zerowania asynchronicznego.

Wykorzystywane w całym projekcie 32-bitowe magistrale są przeznaczone do przesyłu liczb zmiennoprzecinkowych o standardowym formacie: 31 bit – znak, 30:23 bit – cecha, 22:0 bit – mantysa.

## 3.1. Interfejs komunikacyjny

Komunikacja regulatora PID (moduł `pid_module`) z urządzeniem zewnętrznym odbywa się za pomocą interfejsu komunikacyjnego przedstawionego na rys. 3. W celu zmniejszenia liczby użytych wejść/wyjść zastosowano wspólną magistralę danych do wczytania wszystkich nastaw regulatora PID oraz progów nasycenia jednostki całkującej.

Interfejs komunikacyjny pozwala na wprowadzenie do regulatora następujących parametrów:

`prog` – sygnał powodujący w stanie wysokim zapis danych z magistrali `data(31:0)` do komórki pamięci określonej przez `address(2:0)`,

`address(2:0)` – adres komórki pamięci, która zostanie zaprogramowana,

`data(31:0)` – dane do zaprogramowania w komórce pamięci wskazywanej przez `address`,

`current(31:0)` – aktualna wartość sterowanego parametru,

`target(31:0)` – wartość docelowa sterowanego parametru,

`clk` – sygnał taktujący,

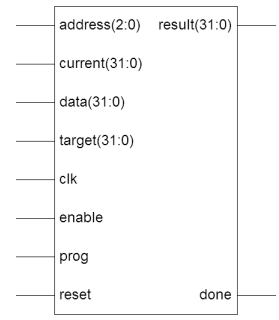
`enable` – sygnał wskazujący gotowość danych wejściowych do próbkowania,

`reset` – zerowanie stanu regulatora i wstrzymanie próbkowania.

Po wykonaniu obliczeń przez regulator PID na wyjściach interfejsu wystawiane są następujące sygnały:

`result (31:0)` – wynik uzyskiwany z regulatora PID,

`done` – sygnał wskazujący, że obliczenia zostały zakończone.




Rys. 3. Linie wejściowe i wyjściowe regulatora
Fig. 3. The IOs of the PID controller

Przygotowując urządzenie do pracy ze sterowanym obiektem, należy w pierwszej kolejności, przy użyciu wejść programujących, wpisać do jego pamięci wszystkie nastawy regulatora PID oraz progi nasycenia jednostki całkującej.

Pamięć wewnętrzna urządzenia ma następującą strukturę:

adres 0 – niewykorzystywany
adres 1 – nastawa członu proporcjonalnego
adres 2 – nastawa członu całkującego
adres 3 – nastawa członu różniczkującego
adres 4 – górny próg nasycenia członu całkującego
adres 5 – dolny próg nasycenia członu całkującego

## 4. Implementacja regulatora PID w języku VHDL

Na podstawie przytoczonych wcześniej założeń projektowych dokonano implementacji regulatora w języku VHDL [4]. Skorzystano z gotowych modułów jednostki zmiennoprzecinkowej: dodającego, odejmującego oraz mnożącego [3] i skoncentrowano się na realizacji algorytmu PID oraz związanych z nim niezbędnych funkcjonalności.

W regulatorze, w pierwszej kolejności obliczana jest wartość uchybu, a następnie przekazywane są niezbędne dane do jednostki proporcjonalnej, całkującej i różniczkującej. Po zakończeniu ich pracy, wyniki są sczytywane, sumowane i przekazywane na wyjście układu.

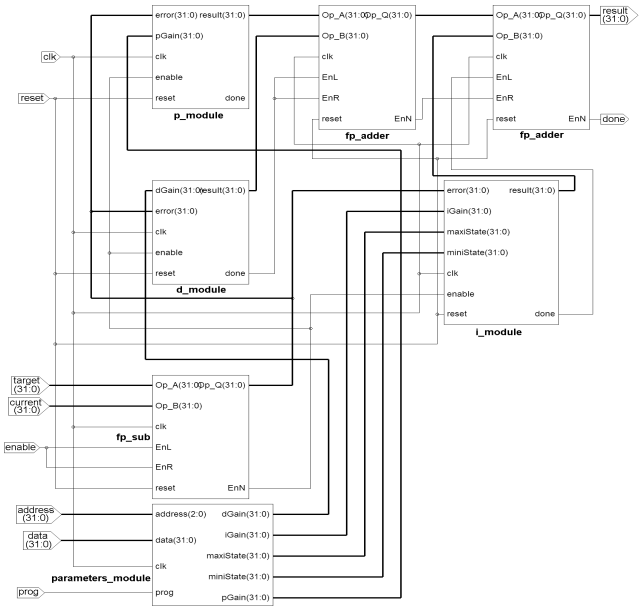
Jednostka modułu proporcjonalnego przeprowadza proste mnożenie wartości uchybu `error(31:0)` przez wartość nastawy `pGain(31:0)`, gdy sygnał `enable` na jeden takt znajdzie się w stanie wysokim.

Przy każdym próbkowaniu moduł całkujący oblicza skumulowaną wartość odczytanych uchybów, która, w przypadku przekroczenia zaprogramowanych wartości krańcowych, zostaje ustawiona na zdefiniowane przez użytkownika minimum lub maksimum.

W jednostce różniczkującej, na podstawie wartości poprzedniego i aktualnego uchybu, obliczana jest wartość różniczki, która po przemnożeniu przez nastawę modułu, jest zwracana jako wynik.

Obok opisanych powyżej modułów wewnątrz regulatora PID znajduje się również wcześniej opisana jednostka pamięci, która umożliwia użytkownikowi wprowadzenie wartości nastaw oraz progów nasycenia jednostki całkującej. Dodatkowo pamięć posiada pięć 32-bitowych linii wyjściowych, które przekazują modułom proporcjonalnemu, całkującemu oraz różniczkującemu, wymagane do prawidłowej pracy, parametry.

Schemat budowy regulatora PID przedstawia rys. 4.



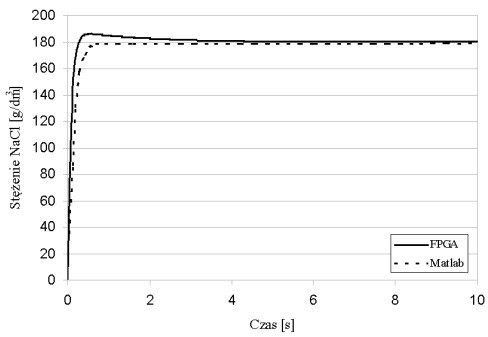

Rys. 4. Schemat zaimplementowanych modułów regulatora PID
Fig. 4. The schematic of the implemented PID controller modules

## 5. Testowanie

W celu sprawdzenia poprawności działania zaprojektowanego układu, zaimplementowano model prostego układu zbiornika z idealnym mieszaniem, a następnie porównano wyniki jego sterowania przy użyciu implementacji regulatora PID w VHDL'u oraz modelu dostępnego w środowisku MatLab/Simulink.

Za sterowany obiekt przyjęto zbiornik o stałej objętości (1 $dm^3$) z idealnym mieszaniem. Przez zbiornik przepływała wodny roztwór NaCl o stałym natężeniu przepływu równym 0,1 $dm^3/s$. Układ reguluje stężenie wpływającej cieczy, aby osiągnąć zadaną wartość stężenia w zbiorniku równą 180 $g/dm^3$. Początkowe stężenie NaCl w zbiorniku wynosi 0.

Parametry wprowadzone do regulatora wynoszą odpowiednio: p_gain = 108,8, i_gain = 1,25, d_gain = 0,31. Na rys. 5 przedstawiono zależność stężenia w zbiorniku od czasu, dla sterowania regulatorem zaimplementowanym w VHDL'u (linia ciągła), oraz dostępnym w pakiecie MatLab/Simulink (linia przerywana).




Rys. 5. Wykres zależności stężenia wodnego roztworu NaCl w zbiorniku od czasu
Fig. 5. Time dependence of NaCl concentration in the tank

Otrzymane wyniki są obiecujące – implementacja regulatora PID w VHDL'u, przy dobrze dobranych parametrach (np. po dostrojeniu metodą Zieglera-Nicholsa [5]).

Regulator wykonany w VHDL'u w pierwszym etapie (0,55s) przekracza wartość zadaną o 3,43% , natomiast regulatorowi z MatLab'a w tym samym czasie brakuje 1,13% do wartości zadanej. Jednakże regulator wykonany w VHDL'u osiąga wartość zadaną z dokładnością ±0,5% w 3,82s, a regulator z MatLab'a dopiero w 44,64s. Wartość zadaną z dokładnością ±0,02% przy regulacji implementacją w VHDL'u osiąga się w 8.84s, natomiast przy regulacji w MatLabie w 374,78s. Wersja w VHDL'u osiąga wartość zadaną z dokładnością 0,00005% w 20,54s, w tym czasie regulator z MatLab'a oscyluje wokół dokładności 0,65%. Regulator z MatLab'a najlepszą badaną dokładność 0,0114% uzyskuje dopiero w 731,31s.

## 6. Wnioski

Regulator PID został zaimplementowany i przetestowany w środowisku Xilinx ISE WebPack (wersja 9.2i) dla układu Xilinx Virtex 4 XC4VLX25, w którym projekt zajął 4457 slice registers oraz 83695 bramek logicznych i wymagał 136 wejść/wyjść. W obecnej formie pełny cykl regulatora (obliczenie pojedynczego wyniku) trwa 41 taktów zegara, co przy określonej przez ISE maksymalnej częstotliwości taktowania ok. 190 MHz odpowiada 215 ns.

Dodatkowo wykonano implementację w układzie Spartan IIe XC2S400E, w którym projekt zajął 4482 slice registers i 83799 bramek logicznych. Maksymalna możliwa częstotliwość pracy regulatora dla tej matrycy to 76 MHz co odpowiada cyklowi regulatora wynoszącemu 539 ns., a więc ponad 2,5 krotnie dłużej w stosunku do implementacji w elemencie Virtex 4.

Przeprowadzone badania pokazują, że możliwe jest w łatwy sposób zrealizowanie w języku VHDL regulatora PID osiągającego dobre wyniki. Zastosowany algorytm jest prosty i przejrzysty, a wymagania odnośnie matrycy nie są zbyt wygórowane (możliwa realizacja np. na wspomnianym Virtex 4 lub Spartan IIe XC2S400E).

Biorąc pod uwagę, że zastosowano gotową jednostkę zmiennoprzecinkową, przypuszcza się, że po zamianie jej na zaprojektowane konkretnie pod regulator moduły dodawania, odejmowania i mnożenia zmiennoprzecinkowego, można by uzyskać mniejszą zajętość matrycy FPGA i większą szybkość działania. Ponadto, w przypadku, gdyby nie była potrzebna założona dokładność (32-bitowa) liczb zmiennoprzecinkowych, można by użyć ich reprezentacji np. 16-bitowej, co z pewnością zaowocowałoby większą szybkością działania regulatora i mniejszą zajętością matrycy. W dalszym etapie prac zamierzamy wykorzystać powstały projekt regulatora PID w prowadzonych pracach badawczych nad implementacją wybranych funkcji sterownika przemysłowego w układzie programowalnym [6, 7, 8, 9, 10].

## 7. Literatura

[1] V. J. Van Doren PID: wciąż najlepszy, Control Engineering, luty 2004 nr 01/02
[2] T. Wescott PID without PhD , Embedded Systems Programming, http://www.embedded.com/2000/0010/0010feat3.htm
[3] Howard University - The RARE Project Floating Point Arithmetic VHDL Models, http://www.imappl.org/~cgloster/rare/vhdl/
[4] M. Zwolinski, Digital System Design with VHDL, Pearson Prentice Hall, Harlow 2004
[5] T. B. Co, CM416 - PID Controller Tuning, Michigan Technological University, http://www.chem.mtu.edu/~tbco/cm416/cm416.html
[6] A. Ziębiński, P. Żaloudik: „Złożony system uruchomieniowy dla układu FPGA", Rozdz. 44 w Systemy Informatyczne z ograniczeniami czasowymi, WKŁ, Warszawa 2006, str. 517-527.
[7] A. Ziębiński, L. Znamirowski: "Challenges in Implementation of FPAA/ FPGA Mixed-signal Technology", International Conference on Engineering Education, ICEE 2005, Conference Proceedings, Volume 1, Current Trends in Engineering Education, July 25-29, Gliwice, Poland. Silesian University of Technology Press, Gliwice 2005, pp. 576-588.
[8] A. Ziębiński, L. Znamirowski, W. Sroka: "Implementacja wybranych funkcji sterownika przemysłowego w układzie programowalnym", Rozdz. 19 w: Praca zbiorowa pod redakcją Z. Huzara i Z. Mazura: Systemy czasu rzeczywistego. Metody i zastosowania, WKŁ, Warszawa 2007, str. 209-218.
[9] A. Ziębiński, L. Znamirowski, Dynamicznie rekonfigurowalny system typu mixed-SIGNAL Systemy czasu Rzeczywistego, Ustroń, 2005
[10] A. Ziębiński, A. Bartkowiak, System uruchomieniowy dla karty HOT II firmy VCC Reprogramowalne Układy Cyfrowe, Szczecin, 2004

*Artykuł recenzowany*