Arkadiusz BUKOWIEC, Alexander BARKALOV
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Implementacja skończonych automatów stanów do struktur FPGA z wielokrotnym kodowaniem stanów

dr inż. Arkadiusz BUKOWIEC

Dr inż. Arkadiusz Bukowiec ukończył studia inżynierskie (2001) o specjalności inżynieria komputerowa na Politechnice Zielonogórskiej a następnie studia magisterskie (2004) o tej samej specjalności na Uniwersytecie Zielonogórskim. W 2008 roku obronił z wyróżnieniem rozprawę doktorską na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego w dziedzinie informatyka. Od roku 2004 pracuje jako asystent w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego.

e-mail: a.bukowiec@iie.uz.zgora.pl

prof. dr hab. inż. Alexander BARKALOV

Prof. Alexander Barkalov ukończył studia (1976) o specjalności komputery na Politechnice Donieckiej. W roku 1983 obronił rozprawę doktorską z zakresu informatyki w Instytucie Mechaniki i Optyki w Leningradzie. W 1995 roku obronił rozprawę habilitacyjną w Instytucie Cybernetyki w Kijowie i uzyskał tytuł doktora habilitowanego ze specjalnością informatyka. W latach 1996-2003 pracował jako profesor w Instytucie Informatyki Narodowej Politechniki Donieckiej. Od 2004 pracuje jako profesor w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego.

e-mail: a.barkalov@iie.uz.zgora.pl

Streszczenie

W artykule została przedstawiona metoda syntezy i implementacji skończonych automatów stanów do struktur FPGA. Metoda ta jest oparta o dekompozycję blokową i wielokrotne kodowanie. Stany automatu zostają podzielone na podzbiory i zakodowane oddzielnie w każdym z nich. Następnie stan jest dekodowany w układzie drugiego poziomu na podstawie jego wielokrotnego kodu i kodu stanu aktualnego lub aktualnie wykonywanej mikroinstrukcji. Prowadzi to do realizacji układu logicznego automatu w strukturze dwupoziomowej, gdzie równocześnie można zastosować tablice LUT i osadzone bloki pamięci. Rozwiązanie takie zapewnia zbalansowane wykorzystanie dostępnych zasobów sprzętowych w nowoczesnych układach FPGA.

Słowa kluczowe: projektowanie układów logicznych, skończony automat stanów, synteza układowa, układ reprogramowalny

Automata Implementation in FPGA devices with Multiple Encoding of States

Abstract

The method of synthesis and implementation into FPGAs (Field Programmable Gate Arrays) of Mealy FSMs (Finite State Machines) is proposed. Synthesis is based on the architectural decomposition and the multiple encoding. A set of states is divided into subsets based on a current state or a currently executed microinstruction. Then, states are encoded separately in each subset. The state is decoded in the second-level circuit based on its multiple code and the code of a current state or the code of a currently executed microinstruction. It leads to implementation of an FSM in double-level structure where utilization of both, LUTs (Look-Up Tables) and embedded memory blocks, is applied. It leads to balanced usage of hardware resources of an FPGA device.

Keywords: circuit synthesis, field programmable gate arrays, finite state machines, logic design

1. Wstęp

Skończone automaty stanów FSM z wyjściami typu Mealy'ego [2] stanowią popularną metodę projektowania jednostek sterujących systemów cyfrowych, przez co mają szerokie zastosowanie w różnych dziedzinach informatyki, elektroniki czy telekomunikacji. Układy FPGA są bardzo często stosowane do implementacji takiego systemu [1]. Układy FPGA zbudowane są z dużej liczby tablic LUT, które mogą realizować dowolną funkcje logiczną [6]. Ograniczeniem jednak jest stosunkowo mała liczba wejść jaką posiadają tablice LUT, ponieważ funkcje logiczne realizowane przez kombinacyjny układ automatu posiadają znacznie więcej argumentów. Rozbieżność ta prowadzi do konieczności zastosowania dekompozycji funkcji boolowskich opisujących zachowanie automatu skończonego [8]. Negatywnym wynikiem takiej dekompozycji jest zwiększenie liczby poziomów w układzie logicznym realizującym algorytm sterowania, co może prowadzić do zmniejszenia wydajności takiego systemu.

Jedną z metod zmniejszenia liczby funkcji realizowanych przez automat jest zastosowanie dekompozycji blokowej [1]. Zaprezentowane metody oparte są o wielokrotne kodowanie stanów wewnętrznych automatu podzielonych na podzbiory w oparciu o aktualny stan lub aktualnie wykonywaną mikroinstrukcję [3]. Kodowanie takie pozwala na zmniejszenie liczby funkcji logicznych realizowanych przez automat. Stan jest dekodowany w układzie drugiego poziomu w oparciu o wielokrotny kod stanu wewnętrznego i kod aktualnego stanu lub kod aktualnie wykonywanej mikroinstrukcji. Ponieważ system ten posiada regularną strukturę może zostać zaimplementowany z wykorzystaniem osadzonych bloków pamięci dostępnych w nowoczesnych układach FPGA. Zabieg taki prowadzi do zmniejszenia liczby standardowych elementów logicznych potrzebnych do implementacji automatu oraz równomiernego wykorzystania różnorodnych zasobów sprzętowych.

2. Standardowe metody syntezy

Układ logiczny skończonego automatu stanów z wyjściami typu Mealy'ego opisany jest systemem funkcji boolowskich [2]:

$$Y = Y(Q, X),$$
$$\Phi = \Phi(Q, X), \qquad (1)$$

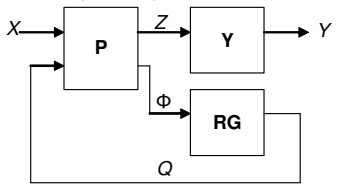
gdzie $X = \{x_1, \ldots, x_L\}$ jest zbiorem warunków logicznych, $Y = \{y_1, \ldots, y_N\}$ jest zbiorem mikrooperacji, $Q = \{q_1, \ldots, q_R\}$ jest zbiorem zmiennych wewnętrznych użytych do kodowania stanów automatu $a_m \in A = \{a_1, \ldots, a_M\}$, gdzie $R = \lceil \log_2 M \rceil$, $\Phi = \{D_1, \ldots, D_R\}$ jest zbiorem funkcji wzbudzeń.

System (1) jest formowany na podstawie tablicy przejść-wyjść automatu, która posiada kolumny: a_m, $K(a_m)$, a_s, $K(a_s)$, X_h, Y_h, Φ_h, h, gdzie $a_m \in A$ jest aktualnym stanem automatu; $K(a_m)$ jest kodem tego stanu; $a_s \in A$ jest następnym stanem automatu; $K(a_s)$ jest kodem stanu a_s; X_h jest koniunkcją afirmacji lub negacji pewnych zmiennych ze zbioru wejściowych warunków logicznych X wymuszając przejście $\langle a_m, a_s \rangle$; $Y_h \subseteq Y$ jest mikroinstrukcją formowaną podczas przejęcia $\langle a_m, a_s \rangle$; $\Phi_h \subseteq \Phi$ jest zbiorem funkcji wzbudzeń, które są równe 1 w celu przełączenia pamięć automatu z kodu $K(a_m)$ na kod $K(a_s)$; h jest numerem linii $h = 1, \ldots, H$.

System (1) jest podstawą do realizacji układu logicznego automatu z wykorzystaniem jednopoziomowej struktury P [1]. W strukturze tej rejestr stanowi pamięć automatu i jest zbudowany z przerzutników typu D, natomiast układ kombinacyjny implementuje system (1) w strukturze FPGA jest zbudowany z tablic LUT.

Jedną ze znanych metod zmniejszania liczby realizowanych funkcji logicznych przez układ kombinacyjny jest zastosowanie kodowania mikroinstrukcji [1]. Niech tablica przejść-wyjść automatu zawiera T różnych mikroinstrukcji $Y_t \subseteq Y$. Zakodujmy każdą taką mikroinstrukcję binarnym kodem $K(Y_t)$ na $R_1 = \lceil \log_2 T \rceil$ bitach ($t = 1, \ldots, T$). Zmienne $z_r \in Z = \{z_1, \ldots, z_{R_1}\}$ zostaną wykorzystane do reprezentacji tego kodu. W takim przypadku układ lo-

giczny automatu może zostać zrealizowany z wykorzystaniem dwupoziomowej struktury PY (rys. 1) [1].



Rys. 1. Schemat blokowy automatu Mealy'ego o strukturze PY
Fig. 1. The structural diagram of the PY Mealy FSM

Rejestr RG ma dokładnie taką samą funkcję i budowę jak w strukturze P. Układ Y implementuje system funkcji boolowskich:

$$Y = Y(Z), \tag{2}$$

czyli dekoduje mikrooperacje. Układ ten w strukturze FPGA może zostać zaimplementowany z wykorzystaniem osadzonych bloków pamięci. W sytuacji takiej układ P implementuje system:

$$Z = Z(Q, X),$$
$$\Phi = \Phi(Q, X), \tag{3}$$

co wpływa na zmniejszenie całkowitej liczby realizowanych funkcji boolowskich. Jednak wartość ta jest wciąż relatywnie duża i nie gwarantuje zmniejszenia liczby tablic LUT wymaganych do implementacji automatu w strukturze FPGA [4].

3. Metoda syntezy z wielokrotnym kodowaniem stanów

Ideą dalszego zmniejszenia liczby realizowanych funkcji boolowskich jest zastosowanie kodowania wewnętrznego stanu automatu z wykorzystaniem kodu mikroinstrukcji lub kodu aktualnego stanu jako częściowego kod tego stanu [3].

Podzielmy zbiór stanów wewnętrznych $a_s \in A = \{a_1, \ldots, a_M\}$ na podzbiory w oparciu o aktualnie realizowaną mikroinstrukcję $Y_t \subseteq Y$. Prowadzi to do powstania T podzbiorów $A(Y_t) \subseteq A$ i stan $a_s \in A(Y_t)$ tylko wtedy gdy jest stanem przejścia podczas realizacji mikroinstrukcji Y_t. Niech $B_t = |A(Y_t)|$ i $B_0 = \max(B_1, \ldots, B_T)$. Zakodujmy każdy stan wewnętrzny $a_s \in A(Y_t)$ binarnym kodem $K_t(a_s)$ na $R_2 = \lceil \log_2 B_0 \rceil$ bitach. Zmienne $\tau_r \in \mathrm{T} = \{\tau_1, \ldots, \tau_{R_2}\}$ zostaną wykorzystane do reprezentacji tego kodu. W tym przypadku kod stanu wewnętrznego $K(a_s)$ jest reprezentowany przez wielokrotny kod stanu wewnętrznego $K_t(a_s)$ i kod mikroinstrukcji $K(Y_t)$:

$$K(a_s) = K_t(a_s) * K(Y_t). \tag{4}$$

Układ logiczny automatu z takim kodowaniem może zostać zrealizowany z wykorzystaniem dwupoziomowej struktury PYY (rys. 2) [3], [4]. Rejestr RG ma dokładnie taką samą funkcję i budowę jak w strukturach P i PY. Układ Y pełni również tą samą rolę co w strukturze PY i realizuje ten sam system (3). Natomiast układ P implementuje system:

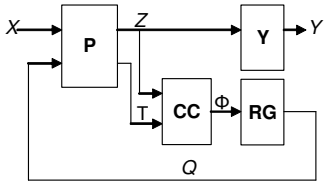
$$Z = Z(Q, X),$$
$$\mathrm{T} = \mathrm{T}(Q, X). \tag{5}$$

W strukturze tej występuje dodatkowy układ CC. Jest on wykorzystywany do dekodowania wewnętrznego stanu automatu i implementuje system:

$$\Phi = \Phi(Z, \mathrm{T}). \tag{6}$$

Ponieważ układ ten posiada regularną strukturę, podobnie jak układ Y, może zostać zaimplementowany z wykorzystaniem osadzonych bloków pamięci w strukturze FPGA.

Dzięki zastosowaniu kodowania obu systemów realizowanych w pierwotnej strukturze P uzyskuje się znaczne zmniejszenie liczby realizowanych funkcji boolowskich, co pozwala na zmniejszenie liczby tablic LUT wymaganych do implementacji automatu w strukturze FPGA.



Rys. 2. Schemat blokowy automatu Mealy'ego o strukturze PYY
Fig. 2. The structural diagram of the PYY Mealy FSM

Metoda, gdzie kod aktualnego stanu jest wykorzystany jako częściowy kod stanu wewnętrznego jest podobna do wcześnie omówionej metody. W tym przypadku zbiór stanów wewnętrznych jest dzielony na podstawie aktualnego stanu a_m. Prowadzi to do powstania M podzbiorów $A(a_m) \subseteq A$ i stan $a_s \in A(a_m)$ tylko wtedy gdy jest stanem przejścia ze stanu a_m. Niech $C_m = |A(a_m)|$ i $C_0 = \max(C_1, \ldots, C_M)$, zakodujmy więc każdy stan wewnętrzny $a_s \in A(a_m)$ binarnym kodem $K_m(a_s)$ na $R_3 = \lceil \log_2 C_0 \rceil$ bitach. Zmienne $\tau_r \in \mathrm{T} = \{\tau_1, \ldots, \tau_{R_3}\}$ zostaną wykorzystane do reprezentacji tego kodu. W tym przypadku kod stanu wewnętrznego $K(a_s)$ jest reprezentowany przez wielokrotny kod stanu wewnętrznego $K_m(a_s)$ i kod aktualnego stanu $K(a_m)$:

$$K(a_s) = K_m(a_s) * K(a_m). \tag{7}$$

Układ logiczny automatu z tym kodowaniem może zostać zrealizowany z wykorzystaniem dwupoziomowej struktury PAY (rys. 3) [3], [4]. Układy RG, Y i P mają taką samą budowę i funkcję jak w strukturze PYY. Tylko układ CC implementuje inny system:

$$\Phi = \Phi(Q, \mathrm{T}). \tag{8}$$

Struktura ta również pozwala na dalsze zmniejszenie liczby funkcji boolowskich realizowanych przez układ kombinacyjny automatu w porównaniu ze strukturami P i PY.

Bardzo trudne jest oszacowanie relacji pomiędzy liczbą funkcji boolowskich realizowanych przez układy kombinacyjne w strukturach PYY i PAY ponieważ wartości te są ściśle powiązane z charakterystyką konkretnie implementowanego algorytmu sterowania. Tak więc dobór struktury PYY lub PAY musi zostać wykonany eksperymentalnie.

4. Proces syntezy i implementacji

Proces syntezy blokowej i implementacji może zostać przeprowadzony w następujący sposób [4]:

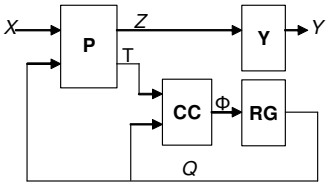
1) Utworzenie i kodowanie mikroinstrukcji. Krok ten opiera się na odczytaniu z tablicy przejść-wyjść automatu wszystkich różnych mikroinstrukcji i zakodowaniu ich binarnym kodem $K(Y_t)$.

2) Podział zbioru stanów wewnętrznych. W kroku tym zbiór stanów wewnętrznych dzielony jest na T lub M podzbiorów, w zależności os stosowanej metody syntezy. Każdy podzbiór zawiera tylko stany które są stanami przejścia podczas wykonywania t-ej mikroinstrukcji lub z m-go stanu.

3) Wielokrotne kodowanie stanów wewnętrznych. Krok ten polega za zakodowaniu binarnym kodem $K_t(a_s)$ lub $K_m(a_s)$ wszystkich stanów wewnętrznych, oddzielnie w każdym z podzbiorów.

4) Utworzenie tablicy przejść-wyjść automatu PYY lub PAY. W kroku tym tworzy się przekształconą tablicę przejść-wyjść, która jest podstawą do utworzenia systemu (5). Jest ona tworzona z oryginalnej tablicy przejść-wyjść poprzez zastąpienie kolumny Y_h kolumną Z_h oraz kolumn $K(a_s)$ i Φ_h kolumnami $K_t(a_s)$ lub $K_m(a_s)$ i T_h. Kolumna Z_h zawiera zmienne $z_r \in Z$, które są równe 1 w kodzie $K(Y_t)$. Kolumna T_h zawiera zmienne $\tau_r \in \mathrm{T}$, które są równe 1 w kodzie $K_t(a_s)$ lub $K_m(a_s)$.

5) Utworzenie tablicy dekodera mikroinstrukcji. Krok ten pro wadzi do powstania tablicy, która jest podstawą do utworzenia systemu (2). Tablica ta posiada kolumny $K(Y_t)$, Y_t, t.



Rys. 3. Schemat blokowy automatu Mealy'ego o strukturze PAY
Fig. 3. The structural diagram of the PAY Mealy FSM

6) Utworzenie tablicy konwertera kodu. W kroku tym tworzy się tablicę, która jest podstawą do utworzenia systemu (6) lub (8). Tablica ta posiada kolumny $K_t(a_s)$ lub $K_m(a_s)$, $K(Y_t)$ lub $K(a_m)$, $K(a_s)$, i.

7) Utworzenie równań logicznych opisujących działanie układu kombinacyjnego. Krok ten prowadzi do uzyskania równań boolowskich opisujących system (5). Tworzone są na podstawie tablicy przejść-wyjść automatu PYY lub PAY.

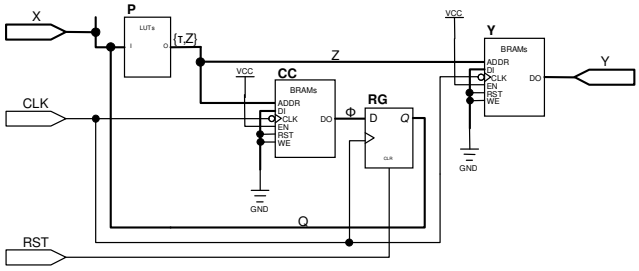
8) Implementacja układu logicznego automatu PYY lub PAY. W przypadku implementacji w układzie FPGA układ kombinacyjny P i rejestr RG implementowane są z wykorzystaniem programowalnych bloków logicznych CLB – układ P z wykorzystaniem tablic LUT a rejestr RG z wykorzystaniem przerzutników typu D. Układ Y implementowany jest z wykorzystaniem osadzonych bloków pamięci, gdzie $K(Y_t)$ jest adresem a Y_t jest słowem zapisanym pod tym adresem. Układ CC również jest implementowany z wykorzystaniem osadzonych bloków pamięci, gdzie adresem jest konkatenacja kodów $K_t(a_s)$ i $K(Y_t)$ lub $K_m(a_s)$ i $K(a_m)$ a wartość słowa zapisanego pod tym adresem stanowi kod $K(a_s)$. Ponieważ osadzone bloki pamięci w strukturach FPGA są elementami synchronicznymi należy zapewnić odpowiednie taktowanie sygnałem zegarowym. Należy do tego celu wykorzystać ten sam sygnał zegarowy co do taktowania rejestru jednak bloki pamięci powinny być aktywowane przeciwny zboczem [4]. Zabieg taki spowoduje, że zdekodowane dane będą gotowe do odczytu w tym samym cyklu zegarowym i nie będzie potrzeby oczekiwania jednego cyklu zegarowego aż będą one stabilne. Sytuacja ta jest szczególne ważna podczas dekodowania stanu wewnętrznego automatu, gdyż w przeciwnym razie mogło by to spowodować spowolnienie działania układu lub nawet jego nieprawidłowo działanie. Schemat logiczny dla automatu PYY przedstawiony jest na rys. 4, a dla automatu PAY na rys. 5 [4].

5. Weryfikacja metod syntezy i implementacji

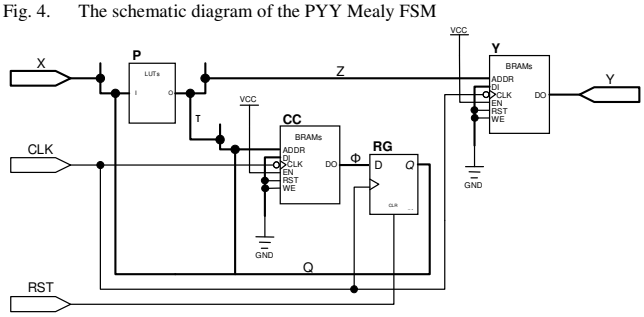
Opracowane metody syntezy zostały zaimplementowane w autorskim, akademickim systemie do syntezy blokowej automatów nazwanym *Automata Synthesis System* [4], [5]. Następnie, uzyskane z tego systemu modele strukturalne [7] mogą zostać poddane implementacji do dowolnego układu FPGA z wykorzystanie komercyjnych narzędzi CAD.

W celu weryfikacji poprawności i efektywności opracowanych metod, takiej syntezie i implementacji poddano moduły testowe z biblioteki *LGSynth91* [9]. Testy zostały przeprowadzone dla układu *Virtex V50* firmy *Xilinx*.

Weryfikacja poprawności polegała na porównaniu wyników symulacji modeli strukturalnych z wynikami symulacji behawioralnej. Uzyskane wyniki potwierdziły poprawne działanie zaproponowanych struktur [4].



Rys. 4. Schemat logiczny automatu Mealy'ego o strukturze PYY
Fig. 4. The schematic diagram of the PYY Mealy FSM



Rys. 5. Schemat logiczny automatu Mealy'ego o strukturze PAY
Fig. 5. The schematic diagram of the PAY Mealy FSM

Weryfikacja efektywności polegała na zestawieniu wyników implementacji uzyskanych dla proponowanych modeli strukturalnych z wynikami implementacji klasycznych modeli (tab. 1). Analiza wyników [4] pokazuje, że zastosowanie struktury PAY redukuje liczbę wymaganych tablic LUT o 33% w porównaniu z strukturą jednopoziomową i o 31% w porównaniu ze strukturą PY. Zysk dla struktury PYY odpowiednio wynosi 19% i 17%.

Tab. 1. Wyniki implementacji wybranych modeli
Tab. 1. Implementation results of selected benchmarks

Moduł	Typ zasobów	Struktura			
		P	PY	PAY	PYY
ex6	Slice	29	19	24	**15**
	LUT	52	34	43	**27**
	przerzutnik	3	3	3	**3**
	BRAM	0	1	2	**2**
planet	Slice	141	90	**64**	75
	LUT	248	155	**113**	131
	przerzutnik	6	6	**6**	6
	BRAM	0	2	**3**	5
s298	Slice	529	628	**238**	398
	LUT	951	1143	**433**	719
	przerzutnik	19	26	**13**	19
	BRAM	0	1	**5**	3
sand	Slice	113	115	89	**84**
	LUT	199	205	156	**148**
	przerzutnik	5	5	5	**5**
	BRAM	0	1	2	**2**
styr	Slice	112	109	**87**	90
	LUT	199	192	**155**	160
	przerzutnik	5	5	**5**	5
	BRAM	0	1	**2**	2
tma	Slice	55	46	**26**	**26**
	LUT	97	80	**45**	**45**
	przerzutnik	5	5	**5**	**5**
	BRAM	0	1	**2**	**2**

6. Podsumowanie

W artykule przedstawiono metody syntezy blokowej i implementacji automatów stanów do struktur FPGA. Zastosowana dekompozycja i kodowanie pozwala na zmniejszenie liczby funkcji boolowskich realizowanych poprzez kombinacyjną część automatu. Prowadzi to do zmniejszenia wymaganej liczby tablic LUT do implementacji układu w strukturze FPGA.

Uzyskane wyniki eksperymentalne pokazały, że obie metody zmniejszają liczbę wymaganych tablic LUT do implementacji układu cyfrowego automatu. W zamian wykorzystane są osadzone bloki pamięci co prowadzi do racjonalnego i równomiernego wykorzystania zasobów sprzętowych dostępnych w nowoczesnych układach FPGA.

7. Literatura

[1] M. Adamski, A. Barkalov: Architectural and Sequential Synthesis of Digital Devices, Univ. of Zielona Góra Press, Zielona Góra, 2006.
[2] S. Baranov: Logic Synthesis for Control Automat, Kluwer, Boston, 1994.
[3] A. Bukowiec, A. Barkalov: Synteza logiczna skończonych automatów stanów z zastosowaniem wielokrotnego kodowania stanów, *Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne*, nr 6, 2008, 766-769.
[4] A. Bukowiec: Synthesis of Finite State Machines for Programmable Devices Based on Multi-Level Implementation, rozprawa doktorska, Uniwersytet Zielonogórski, 2008
[5] A. Bukowiec: Automata Synthesis System Website http://willow.iie.uz.zgora.pl/~abukowie/AS/as.htm
[6] J. Jenkins: Designing with FPGAs and CPLDs, Prentice Hall, NJ, 1994.
[7] J. M. Lee: Verilog QuickStart: A Practical Guide to Simulation and Synthesis in Verilog, Kluwer, Norwell, 1999.
[8] T. Łuba: Synteza układów logicznych, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2005.
[9] S. Yang: Logic Synthesis and Optimization Benchmarks User Guide, raport techniczny, Microelectronics Center of North Carolina, NC, 1991.

Artykuł recenzowany