Arkadiusz BUKOWIEC
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Synteza skończonych automatów stanów z zastosowaniem szeregowego przekształcenia mikroinstrukcji

Mgr inż. Arkadiusz BUKOWIEC

Mgr 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 roku 2000 odbył przemysłową praktykę studencką w firmie Aldec Inc. w Stanach Zjednoczonych. Od roku 2004 pracuje jako asystent na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego.

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

Streszczenie

W artykule została przedstawiona metoda zmniejszenia wymaganych zasobów sprzętowych do implementacji skończonych automatów stanów z wyjściami typu Mealy'ego w matrycowym układzie programowalnym. Zaproponowana metoda oparta jest na szeregowym przekształceniu mikroinstrukcji. W rezultacie takiego zabiegu wszystkie mikrooperacje w nowo powstałej tablicy przejść-wyjść automatu będą kompatybilne. Umożliwia to zakodowanie każdej mikrooperacji za pomocą binarnego kodu na możliwie minimalnej liczbie bitów. W sytuacji takiej do implementacji systemu mikrooperacji potrzebny jest tylko jeden dekoder. Metoda ta pozwala na zmniejszenie liczby wyjść części kombinacyjnej automatu Mealy'ego w porównaniu z realizacją jako automat Mealy'ego z kodowaniem klas kompatybilnych mikrooperacji.

Słowa kluczowe: Automat stanów, jednostka sterująca, układy programowalne.

Synthesis of Finite State Machines with Verticalization of Microinstructions

Abstract

The method of decreasing of logic amount in FPGA device implementing the logic circuit of finite state machine (FSM) is proposed. Method is based on verticalization of microinstructions. As a result of verticalization all microoperations of direct structural table (DST) are compatible ones. It permits to encode each microoperation by code with minimal possible number of bits. In this case only one decoder is used for implementation of the microoperations system. This method permits to minimize an amount of outputs of the combinational part of Mealy FSM in comparison with the same characteristic of Mealy FSM with encoding of fields of compatible microoperations.

Keywords: FSM, Control unit, FPGA.

1. Wstęp

Jednym ze sposobów projektowania jednostek sterujących układem cyfrowym lub obiektem przemysłowym jest zastosowanie skończonych automatów stanów z wyjściami typu Mealy'ego [1]. Programowalne układy FPGA są bardzo często stosowane do implementacji takiego automatu [7]. Główną cechą układów FPGA jest występowanie w nich dużej liczby n-wejściowych tablic LUT, które mogą zrealizować każdą funkcje logiczną n zmiennych [6, 7]. Ograniczeniem jednak jest stosunkowo mała liczba n wejść jaką posiadają tablice LUT (typowo 4), z drugiej strony funkcje logiczne realizowane przez kombinacyjny układ automatu posiadają znacznie więcej argumentów (do 200). Rozbieżność ta prowadzi do blokowej dekompozycji funkcji boolowskich opisujących zachowanie automatu skończonego [6]. Negatywnym wynikiem takiej dekompozycji jest zwiększenie liczby poziomów w układzie logicznym realizującym algorytm sterowania, co może prowadzić do zmniejszenia wydajności takiego systemu.

W prezentowanym referacie przedstawiona jest metoda umożliwiająca zmniejszenie liczby funkcji logicznych realizowanych przez kombinacyjną część automatu, zależnych od wejściowych warunków logicznych i zmiennych wewnętrznych reprezentujących aktualny stan automatu. Prowadzi to do zmniejszenia liczby poziomów układu cyfrowego realizującego automat i do zmniejszenia wymaganej liczby tablic LUT oraz zwiększenia wydajności jednostki sterującej, w porównaniu ze znanymi metodami projektowania [2, 3].

Głównym założeniem prezentowanej metody jest przeprowadzenie szeregowego przekształcenia mikroinstrukcji występujących w tablicy przejść-wyjść automatu opisującej jego algorytm. Metoda ta jest wzorowana na liniowym przekształceniu sieci działań [4] opisującej algorytm sterowania, jednak stosowana jest w późniejszym etapie syntezy.

2. Analiza metody syntezy z kodowaniem klas kompatybilnych mikrooperacji

Najpopularniejszym sposobem opisu automatu skończonego z wyjściami typu Mealy'ego jest tablica przejść-wyjść [1, 6]. Może ona zostać uzyskana z sieci działań [1] lub grafu stanów automatu [6]. Jednak ze względu na swoja prostotę i możliwość czytelnego zapisu z wykorzystaniem tekstowych formatów [6] jest ona podstawą do procesu syntezy (utworzenia systemu funkcji boolowskich) automatu. Tablica przejść-wyjść posiada kolumny: a_m, $K(a_m)$, a_s, $K(a_s)$, X_h, Y_h, Φ_h, h, gdzie a_m jest początkowym stanem automatu, $a_m \in A$ gdzie $A = \{a_1, ..., a_M\}$ jest zbiorem stanów automatu; $K(a_m)$ jest kodem stanu $a_m \in A$, zapisanym na $R =]\log_2 M[$ bitach; a_s jest następnym stanem automatu; $K(a_s)$ jest kodem stanu $a_s \in A$; X_h jest koniunkcją pewnych zmiennych ze zbioru wejściowych warunków logicznych $X = \{x_1, ..., x_L\}$ wymuszając przejście $\langle a_m, a_s \rangle$; Y_h jest mikroinstrukcją (μI) formowaną podczas przejęcia $\langle a_m, a_s \rangle$, $Y_h \subseteq Y$ gdzie $Y = \{y_1, ..., y_N\}$ jest zbiorem wszystkich mikrooperacji (μO); Φ_h jest zbiorem funkcji wzbudzeń, które są równe 1 w celu przełączenia pamięć automatu z kodu $K(a_m)$ na kod $K(a_s)$, $\Phi_h \subseteq \Phi$, gdzie $\Phi = \{D_1, \ldots, D_R\}$; h jest numerem przejścia automatu ($h = 1, ..., H$). Wewnętrzne zmienne $Q_r \in Q$ wykorzystane są do zakodowania stanów automatu, gdzie $Q = \{Q_1, ..., Q_R\}$. Na tej podstawie wyprowadzany jest system wzbudzeń przerzutników i mikrooperacji:

$$\Phi = \Phi(Q, X), \tag{1}$$

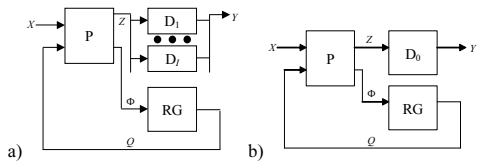
$$Y = Y(Q, X). \tag{2}$$

Systemy te realizowane są przez kombinacyjną część automatu, natomiast aktualny stan zapamiętywany jest w zbiorze rejestrów. W przypadku implementacji z wykorzystaniem układów FPGA do jego budowy stosuje się przerzutniki typu D [6, 7]. W takiej sytuacji mamy doczynienia z jedno-poziomową strukturą automatu [2], jednak ze względu na ograniczenia tablic LUT dekompozycja funkcji logicznych doprowadzi do struktury wielopoziomowej.

Jednym ze sposobów optymalizacji wymaganych zasobów sprzętowych do realizacji automatu jest zmniejszenie liczby funkcji zależnych od warunków logicznych i zmiennych wewnętrznych. Jedną z umożliwiających to metod jest kodowanie klas kompatybilnych mikrooperacji [3]. Mikrooperacje $y_i, y_j \in Y$ są kompatybilne, jeżeli nigdy nie występują razem w tej samej mikroinstrukcji Y_h. Niech zbiór $\Pi_Y = \{B_1, ..., B_I\}$ będzie podziałem zbioru Y na klasy kompatybilnych mikrooperacji. Niech każda mikrooperacja $y_n \in B_i$ zostanie zakodowana za pomocą kodu $K(y_n)$ na r_i bitach, gdzie $r_i =]\log_2(|B_i|+1)[$ ($i = 1, ..., I$). W takim przy-

padku automat Mealy'ego zostanie zaimplementowany w dwupoziomowej strukturze PD (rys. 1 a).

W strukturze takiej układ D implementowany jest za pomocą I dekoderów, gdzie każdy z nich implementuje mikrooperacje $y_n \in B_i$ z jednej klasy kompatybilności B_i ($i = 1, ..., I$).



Rys. 1. Schemat blokowy a) automatu PD i b) automatu PD_0
Fig. 1. Block diagram of a) the PD automaton and b) the PD_0 automaton

Mikrooperacje $y_n \in Y$ są dekodowane na podstawie zmiennych ze zbioru $Z = \{z_1,\ldots,z_{R_1}\}$ gdzie $R_1 = \sum_{i=1}^{I} r_i$. W sytuacji takiej układ P realizuje funkcje (1) i

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

Prowadzi to do zmniejszenia liczby funkcji zależnych od warunków logicznych i zmiennych wewnętrznych realizowanych przez kombinacyjną część automatu.

Wadą takiego rozwiązania jest wciąż relatywnie duża liczba funkcji Z oraz konieczność implementacji I dekoderów. Najlepszym rozwiązaniem problemu zmniejszenia wyjść układu P bez zmniejszania wydajności jednostki sterującej jest sytuacja, gdy wszystkie mikrooperacje realizowane przez automat są wzajemnie do siebie kompatybilne. Przypadek taki odzwierciedlony jest przez liniową sieć działań [3, 4] gdzie każdy blok operacyjny zawiera tylko jedną mikrooperację. W tablicy przejść-wyjść sytuacje taką odwzorowuje występowanie mikroinstrukcji zawierających tylko jedną mikrooperację. W takim przypadku każda mikrooperacja $y_n \in Y$ jest kodowana przez kod $K(y_n)$ na $R_3 =]\log_2(N+1)[$ bitach. Układ taki jest implementowany za pomocą automatu PD_0 (rys. 1 b), gdzie układ D_0 realizowany jest za pomocą jednego dekodera o R_3 wejściach i N wyjściach.

Głównym problem sieci działań opisujących rzeczywiste układy sterowania jest fakt, iż nie są one liniowe. W prezentowanym artykule zaproponowane zostanie szeregowe przekształcenie mikroinstrukcji wzorowane na liniowym przekształceniu sieci działań [3, 4]. Zabieg ten doprowadzi do uzyskania kompatybilności pomiędzy wszystkimi mikrooperacjami.

3. Szeregowe przekształcenie mikroinstrukcji

Niech tablica przejść-wyjść automatu zawiera H mikroinstrukcji $Y_h \subseteq Y$ tworzących zbiór $\beta = \{Y_1, ..., Y_H\}$. Niech mikroinstrukcja $Y_h \in \beta$ zawiera N_h mikrooperacji, $Y_h = \{y_n^h \mid y_n \in Y_h\}$. Procedurę szeregowego przekształcenia mikroinstrukcji należy zastosować do każdej mikroinstrukcji $Y_h \in \beta$ (każdego wiersza tablicy przejść-wyjść automatu). W takim przypadku każdy wiersz opisujący przejście $\langle a_m, a_s\rangle$ pod warunkiem X_h podczas, którego realizowana jest mikroinstrukcja Y_h zostanie zastąpiony N_h nowymi wierszami. Pierwszy z nowopowstałych wierszy opisywał będzie przejście $\langle a_m, a_s^i\rangle$ (gdzie $i = 1, ..., N_h$ i jest numerem tworzonego wiersza) pod warunkiem X_h podczas, którego realizowana jest unikalna mikrooperacja $y_n^h \in Y_h$. Wiersze od 2 do N_h-1 opisywały będą bezwarunkowe przejścia $\langle a_s^{i-1}, a_s^i\rangle$ podczas, których realizowane są kolejne unikalne mikrooperacje $y_n^h \in Y_h$. Ostatni wiersz opisywać będzie przejście $\langle a_s^{i-1}, a_s\rangle$ podczas, którego realizowana jest ostatnia unikalna mikrooperacja $y_n^h \in Y_h$. Przykład tablicy przejść-wyjść przed i po przekształceniu został pokazany odpowiednio w tabeli 1 i 2. Podczas przekształcenia powstanie Δ nowych stanów, które należy zakodować. Zabieg ten może doprowadzić do konieczności powiększenia zbioru Q.

Tab. 1. Fragment tablicy przejść-wyjść automatu
Tab. 1. A part of DST table of automaton

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Y_h	Φ_h	h
a_2	001	a_3	010	1	$y_1\ y_2$	D_2	1
a_3	010	a_4	011	x_1	y_3	$D_2\ D_3$	2
		a_5	100	$\sim x_1$	$y_4\ y_5$	D_1	3

Tab. 2. Fragment tablicy przejść-wyjść po szeregowym przekształceniu μI
Tab. 2. A part of DST table after verticalization of μI

a_m	$K(a_m)$	a_s	$K(a_s)$	$X_{h\,i}$	$Y_{h\,i}$	$\Phi_{h\,i}$	h	i
a_2	001	a_3^1	110	1	y_1	$D_1\ D_2$	1	1
a_3^1	110	a_3	010	1	$y_2\ y_0$	D_2	1	2
a_3	010	a_4	011	x_1	$y_3\ y_0$	$D_2\ D_3$	2	1
		a_5^1	111	$\sim x_1$	y_4	$D_1\ D_2\ D_3$	3	1
a_5^1	111	a_5	100	1	$y_5\ y_0$	D_1	3	2

Należy jednak tu wspomnieć, że rozwiązanie takie posiada następujące wady:

1. Liczba stanów w automacie PD_0 jest większa niż w automacie PD realizującym ten sam algorytm o:

$$\Delta = \sum_{t=1}^{T} (N_t - 1), \tag{4}$$

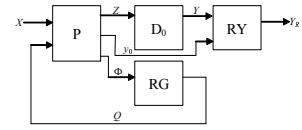
gdzie T jest liczbą różnych mikroinstrukcji w tablicy przejść-wyjść zawierającą N_t mikrooperacji. W przypadku gdy zachodzi warunek:

$$]\log_2(M + \Delta)[>]\log_2 M[, \tag{5}$$

następuje zwiększenie liczby zmiennych w zbiorze Q potrzebnych do zakodowania stanów z R do $R_0 =]\log_2(M+\Delta)[$. Prowadzi to do zwiększenia liczby przerzutników potrzebnych do zapamiętania kodu stanu oraz tablic LUT potrzebnych do implementacji systemu (1). Jednocześnie jednak zmniejszona zostaje liczba tablic LUT potrzebnych do implementacji systemu (3).

2. Liczba cykli potrzebnych do wykonania mikroinstrukcji Y_h w automacie PD_0 wzrasta N_h razy w porównaniu z automatem PD. Powoduje to spadek wydajności układu cyfrowego realizującego algorytm sterowania. Gdy sposób wykonywania mikrooperacji $y_n \in Y_h$ ma wpływ na ścieżkę danych [3] takie przekształcenie nie może zostać zastosowane.

W celu wyeliminowania drugiej wady w zmodyfikowanej tablicy przejść-wyjść można dodać specjalny sygnał y_0 do synchronizacji ścieżki danych w każdym ostatnim wierszu opisującym przejście do stanu a_s. W takim przypadku ścieżka danych rozpoczyna działanie tylko wtedy, gdy wszystkie mikrooperacje $y_n \in Y_h$ zostaną wygenerowane i zapamiętane w specjalnym rejestrze wyjściowym RY. Podejście takie prowadzi do powstania automatu o strukturze PD_V (rys. 2). Należy tu wspomnieć, że rejestr wyjściowy RY stosowany jest również w strukturach PD i PD_0 w celu ich stabilnego funkcjonowania [2].



Rys. 2. Schemat blokowy automatu PD_V
Fig. 2. Block diagram of the PD_V automaton

Układ cyfrowy automatu PD_V działa w następujący sposób: niech w monetach czasu t ($t = 1, 2, ...$) automat znajduje się w stanie $a_m \in A$. Jeżeli $N_k > 1$ to $y_0 = 0$ blokuje synchronizację ze ścieżką danych. Mikrooperacja $y_n \in Y_h$ jest zapisywana w n-tym przerzutniku rejestru RY. Następnie automat przechodzi przez kolejne stany a_s^i i zapisuje kolejne mikrooperacje w rejestrze RY. Kiedy zostanie osiągnięty stan a_s to y_0 przyjmuje wartość 1 i mikrooperacje $y_n \in Y_h$ z wyjść Y_R rejestru RY zostają wykonane

równolegle przez ścieżkę danych. Po wykonaniu mikroinstrukcji Y_h rejestr RY jest zerowany i operacja powtarzana jest dla kolejnych stanów. Proponowane podejście zapewnia zwiększenie wydajności układu cyfrowego a jego rezultat zależy od relacji między liczbą cykli wykonywanych przez jednostkę sterującą a ścieżką danych.

W celu wyeliminowania pierwszej wady można wprowadzić modyfikacje struktury PD_V poprzez wprowadzenie dodatkowego licznika do zliczania sekwencji stanów a_s^i [5]. Rozwiązanie to wymaga dodatkowych zasobów do realizacji układu licznika jedynka nie wpływa na zwiększenie liczby tablic LUT potrzebnych do implementacji systemu (1). Opis modyfikacji tej jednak nie jest tematem tego artykułu.

4. Metoda syntezy automatu PD_V

W tym rozdziale zostaną omówione kroki metody syntezy automatu do struktury PD_V w oparciu o szeregowe przekształcenie mikroinstrukcji.

Punktem wejścia do procesu syntezy jest tablica przejść-wyjść automatu.

Proponowana metoda syntezy składa się z następujących kroków:

1. Przekształcenie początkowej tablicy przejść-wyjść poprzez szeregowe przekształcenie wszystkich mikroinstrukcji. Algorytm ten został omówiony w rozdziale 3.
2. Zakodowanie nowo powstałych stanów. Jeżeli nie zachodzi warunek (6) kodowanie dodatkowych stanów należy rozpocząć od kodu $K(a_M)$+1. W przeciwnym razie należy rozszerzyć zbiór zmiennych Q do R_0 bitów. Dla stanów od a_1 do a_M dodatkowe bity należy wypełnić wartościami 0. Następnie kodowanie należy również rozpocząć od kodu $K(a_M)$+1.
   Jeżeli nie jest konieczne zachowanie kodów z przed procesu przekształcenia dla stanów od a_1 do a_M w obu przypadkach korzystne może okazać się ponowne zakodowanie stanów na R_0 bitach w sekwencji a_1, a_2^1, ..., a_2^{Nh-1}, a_2, ..., a_M^1, ..., a_M^{Nh-1}, a_M, a_1^1, ..., a_1^{Nh-1}.
3. Wprowadzenie sygnału y_0 do przekształconej tablicy przejść-wyjść automatu. Należy postępować zgodnie z regułą opisaną w rozdziale 3.
   Dla przykładu z tabeli 1. efektem wykonania kroków od 1 do 3 będzie uzyskanie przekształconej tablicy przejść-wyjść pokazanej w tabeli 2.
4. Maksymalne kodowanie [2] mikrooperacji $y_n \in Y$ za pomocą binarnego kodu $K(y_n)$ na R_3 bitach i utworzenie zbioru Z, z którego zmienne zostaną wykorzystane do reprezentacji tego kodu.
   Ponieważ po szeregowym przekształceniu wszystkie mikrooperacje są do siebie kompatybilne wystarczy zastosować maksymalne kodowanie mikrooperacji oraz jeden dekoder do zdekodowania sytemu Y na podstawie sytemu Z.
5. Utworzenie tablicy przejść-wyjść dla automatu o strukturze PD_V poprzez zastąpienie kolumny $Y_{h\cdot i}$ przez kolumny $K(y_n)$, $Z_{h\cdot i}$ i y_0. Kolumna $Z_{h\cdot i}$ zawiera zmienne $z_r \in Z$, które są równe 1 w kodzie $K(y_n)$ dla $h\cdot i$-tego wiersza tablicy przejść-wyjść automatu PD_V.
   Tablica przejść-wyjść dla automatu o strukturze PD_V dla rozważanego przykładu przedstawiona jest w tabeli 3.

Tab. 3. Fragment tablicy przejść-wyjść automatu PD_V
Tab. 3. A part of DST table of PD_V automaton

a_m	$K(a_m)$	a_s	$K(a_s)$	$X_{h\cdot i}$	$K(y_n)$	$Z_{h\cdot i}$	y_0	$\Phi_{h\cdot i}$	h	i
a_2	001	a_3^1	110	1	001	z_3	0	$D_1 D_2$	1	1
a_3^1	110	a_3	010	1	010	z_2	1	D_2	1	2
a_3	010	a_4	011	x_1	011	$z_2 z_3$	1	$D_2 D_3$	2	1
		a_5^1	111	$\sim x_1$	100	z_1	0	$D_1 D_2 D_3$	3	1
a_5^1	111	a_5	100	1	101	$z_1 z_3$	1	D_1	3	2

6. Utworzenie tablicy dekodera do uformowania systemu Y. Tablica ta posiada kolumny $K(y_n)$ i y_1, ...,y_N.
   Tablica dekodera dla rozważanego przykładu przedstawiona jest w tabeli 4.

   W układzie FPGA taki dekoder może zostać zaimplementowany za pomocą bloku pamięci pracującego w trybie pamięci ROM o N bitowych słowach i R_3 bitowym adresie lub za pomocą tablic LUT poprzez implementację N funkcji logicznych zależnych od R_3 zmiennych.

Tab. 4. Tablica dekodera automatu PD_V
Tab. 4. Decoder table of PD_V automaton

$K(y_n)$							
z_1	z_2	z_3	y_1	y_2	y_3	y_4	y_5
0	0	1	1	0	0	0	0
0	1	0	0	1	0	0	0
0	1	1	0	0	1	0	0
1	0	0	0	0	0	1	0
1	0	1	0	0	0	0	1

7. Utworzenie systemów (1), (3) i

$$y_0 = y(Q, X) \tag{6}$$

   na podstawie tablicy przejść-wyjść automatu o strukturze PD_V. Funkcje te następnie mogą zostać zaimplementowane w układzie FPGA z wykorzystaniem tablic LUT. W celu optymalizacji tej części układu można zastosować funkcjonalną dekompozycję [6] do uzyskanych funkcji boolowskich.

5. Podsumowanie

Zaprezentowana metoda projektowania dwupoziomowego automatu skończonego z wyjściami typu Mealy'ego z zastosowaniem szeregowego przekształcenia mikroinstrukcji pozwala na zmniejszenie liczby funkcji zależnych od warunków logicznych oraz zmiennych wewnętrznych automatu realizowanych przez układ kombinacyjny. Szeregowe przekształcenie mikroinstrukcji zapewnia konieczność użycia tylko jednego dekodera do implementacji systemu mikrooperacji. Przeprowadzone badania analityczne pokazały, że proponowana metoda pozwala na zmniejszenie wymaganych do implementacji układu tablic LUT.

Metoda szeregowego przekształcenia mikroinstrukcji daje takie same efekty jak metoda liniowego przekształcenia sieci działań, jednak ponieważ operuje ona na tablicy przejść-wyjść automatu będzie ona znacznie łatwiejsza w programowej implementacji podczas realizacji sytemu do automatycznej syntezy automatów. Tablica przejść-wyjść jest obecnie również bardziej popularnym formatem do reprezentacji algorytmu automatu tak więc w przypadku zastosowania liniowego przekształcenia sieci działań konieczne było by przejście z tablicy do sieci i z powrotem.

Ubocznym efektem szeregowego przekształcenia mikroinstrukcji jest zmniejszenie wydajności systemu cyfrowego ze strukturą PD_V. Efekt ten jednak może zostać ograniczony przez zastosowanie omówionej metody synchronizacji.

6. Literatura

[1] Baranov S.: Logic Synthesis for Control Automata, Kluwer Academic Publishers, 1994
[2] Barkalov A. A.: Development of Formal Methods of Structural Synthesis of Compositional Automata, DonTSY, Donieck, 1994 (po rosyjsku).
[3] Barkalov A. A.: Synthesis of Operational Units, DonNTU, Donieck, 2004 (po rosyjsku).
[4] Barkalov A. A., Bukowiec A.: Synthesis of Mealy FSM on Verticalized Flow-chart, Radioelektronika i Informatika, 2005, no 2, s. 49-52
[5] Barkalov A. A., Bukowiec A.: Synthesis of Mealy Finite States Machines for Interpretation of Verticalized Flow-charts, Informatyka Teoretyczna i Stosowana, 2005, R. 5, nr 8, s. 39-51
[6] Łuba T. (ed.): Synteza układów cyfrowych, WKŁ, Warszawa 2003.
[7] Salcic Z.: VHDL and FPLDs in Digital Systems Design, Prototyping and Customization, Kluwer Academic Publishers, 1998.

Artykuł recenzowany