Piotr SZOTKOWSKI, Mariusz RAWSKI
POLITECHNIKA WARSZAWSKA, INSTYTUT TELEKOMUNIKACJI

Algorytm funkcjonalnej dekompozycji symbolicznej automatów skończonych dla celów implementacji w strukturach FPGA

Mgr inż. Piotr SZOTKOWSKI

Otrzymał tytuł magistra inżyniera na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej w 2004 roku. W 2005 roku rozpoczął na tym wydziale studia III stopnia i obecnie jest doktorantem w Zakładzie Podstaw Telekomunikacji w Instytucie Telekomunikacji PW; tematem doktoratu jest symboliczna dekompozycja funkcjonalna w syntezie automatów. Jest zainteresowania naukowe obejmują też szerszą syntezę logiczną układów cyfrowych; jest autorem kilku publikacji z tej dziedziny.

e-mail: p.szotkowski@tele.pw.edu.pl

Dr inż. Mariusz RAWSKI

Otrzymał tytuł magistra inżyniera na Wydziale Elektroniki Politechniki Warszawskiej w 1995 roku. Stopień doktora otrzymał na tym samym wydziale w 2000 roku. Obecnie jest adiunktem na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Jego zainteresowania naukowe obejmują syntezę logiczną układów cyfrowych, narzędzia CAD dla syntezy i optymalizacji logicznej, projektowanie systemów cyfrowych z wykorzystaniem struktur programowalnych PLD.

e-mail: rawski@tele.pw.edu.pl

Streszczenie

Dotychczasowe podejście do implementacji automatów skończonych w układach FPGA składa sie z dwóch etapów: kodowania stanów automatu oraz mapowania powstałych funkcji w strukturze układu. W przypadku mapowania za pomocą dekompozycji funkcjonalnej trudno jest znaleźć „dobrą" metodę kodowania stanów, szczególnie przy zastosowaniu procesu syntezy wielopoziomowej. Artykuł opisuje algorytm funkcjonalnej dekompozycji symbolicznej, który zamiast osobnego etapu kodowania wprowadza sukcesywne kodowanie stanów gwarantujące dobrą jakość dekompozycji, oraz przedstawia działanie tego algorytmu na przykładzie wybranego automatu skończonego.

Słowa kluczowe: funkcjonalna dekompozycja symboliczna, automat skończony, FSM, FPGA.

Symbolic Functional Decomposition Algorithm for FSM Implementation in FPGA Structures

Abstract

The following paper presents an algorithm of symbolic functional decomposition for implementation of finite state machines in FPGA circuits. The idea of symbolic functional decomposition does not require a separate step of encoding the FSM's states. This method uses a description of the FSM that maintains a symbolic representation of the machine's states and introduces their encoding gradually, during each of the iterations of the decomopsition process; such approach guarantees high quality of the final decomposition. This paper presents an algorithm of symbolic functional decomposition based on blanket algebra. Each of the algorithm's steps is described in detail, as well as presented on an example FSM.

Keywords: symbolic functional decomposition, finite state machine, FSM, FPGA.

1. Wprowadzenie

Poniższy artykuł przedstawia algorytm funkcjonalnej dekompozycji symbolicznej automatów skończonych dla celów implementacji w strukturach FPGA. Dotychczasowe podejścia do tego zagadnienia opierały się na dwóch krokach: zakodowaniu stanów automatu w celu stworzenia binarnej tablicy przejść/wyjść „łatwej" do zamapowania, oraz samym mapowaniu otrzymanej funkcji w strukturze docelowej (komórkach LUT). W przypadku układów FPGA w procesie mapowania wykorzystywany jest algorytm dekompozycji funkcjonalnej; niestety jego złożoność powoduje, że trudno jest znaleźć algorytm kodowania dający dobre rezultaty, szczególnie w przypadku syntezy wielopoziomowej [1, 2, 3].

Metoda funkcjonalnej dekompozycji symbolicznej, zaproponowana w [4] i [5], eliminuje ten problem i nie wymaga osobnego etapu kodowania stanów. Zastosowanie tej metody do wielopoziomowej implementacji automatów skończonych w układach FPGA zostało opisane w [6]. Metoda ta opiera się na reprezentacji automatu utrzymującej przez cały proces dekompozycji symboliczną reprezentację jego stanów, wprowadzając kodowanie stopniowo na każdym z kolejnych iteracji procesu dekompozycji. Takie podejście gwarantuje wysoką jakość ostatecznej dekompozycji automatu.

Poniższy artykuł przedstawia algorytm funkcjonalnej dekompozycji symbolicznej oparty na rachunku nakryć; jego działanie zostało także przedstawione na przykładowym automacie.

2. Funkcjonalna dekompozycja symboliczna i przykładowy automat skończony

2.1. Funkcjonalna dekompozycja symboliczna

Niech X oznacza zbiór wejść automatu skończonego określonego tablicą przejść/wyjść, a Y – zbiór jego wyjść. Niech Q i Q' będą wielowartościowymi zmiennymi zawierającymi aktualny i następny stan automatu. Niech U i V oznaczają dwa podzbiory X takie, że $U * V = X$. Niech Q_V i Q_U oznaczają zmienne wielowartościowe kodujące zmienną Q. Niech β_V i β_U oznaczają nakrycia generowane przez zbiory zmiennych wejściowych V i U. Niech β_{Qv} i β_{Qu} oznaczają nakrycia generowane przez wielowartościowe zmienne Q_V i Q_U. Niech β_Y i $\beta_{Q'}$ oznaczają nakrycia generowane przez zbiór zmiennych wyjściowych i wielowartościową zmienną Q'.

Twierdzenie o istnieniu funkcjonalnej dekompozycji symbolicznej.

Automat skończony ma funkcjonalną dekompozycję symboliczną względem U, Q_U, Q_V i V jeśli istnieje nakrycie β_G takie, że $\beta_V \bullet \beta_{Qv} \le \beta_G$ oraz $\beta_U \bullet \beta_{Qu} \bullet \beta_G \le \beta_F$, gdzie $\beta_F = \beta_Y \bullet \beta_{Q'}$.

Algorytm funkcjonalnej dekompozycji symbolicznej zakłada automat skończony o n zmiennych wejściowych $X = \{x_1, x_2, ..., x_n\}$ i m zmiennych wyjściowych $Y = \{y_1, y_2, ..., y_m\}$. Algorytm zakłada też, że podział zmiennych wejściowych na zbiory U i V (bezpośrednie i pośrednie części zbioru X) został zdefiniowany, oraz że została określona architektura bloku pośredniego G (w szczególności – liczba jego wyjść).

2.2. Przykładowy automat skończony

Algorytm został zilustrowany na przykładzie automatu skończonego z tabeli 1a.

Tab. 1a. Tablica przejść/wyjść automatu; b. Symboliczne kodowanie zmiennej Q; c. Blok H końcowej dekompozycji; d. Blok G końcowej dekompozycji
Tab. 1a. State transition table of an FSM; b. Symbolic encoding of the Q variable; c. The H block of the final decomposition; d. The G block of the final decomposition

a.

	x_1	x_2	x_3	x_4	Q	Q'	y_1	y_2
1	-	-	0	0	init0	init1	0	0
2	0	1	0	0	init1	init1	0	0
3	-	-	1	-	init1	init2	1	0
4	1	-	1	0	init2	init4	1	0
5	-	1	1	1	init4	init4	1	0
6	-	-	0	1	init4	IOWait	0	1
7	0	0	0	-	IOWait	IOWait	0	1
8	1	0	0	-	IOWait	init1	0	1
9	0	1	1	0	IOWait	read0	0	0
10	1	1	0	0	IOWait	write0	1	1
11	0	1	1	1	IOWait	RMACK	1	1
12	1	1	0	1	IOWait	WMACK	0	0
13	-	0	1	-	IOWait	init2	0	1
14	0	0	1	0	RMACK	RMACK	1	1
15	0	1	1	1	RMACK	read0	0	0
16	1	1	0	0	WMACK	WMACK	0	0
17	1	0	0	1	WMACK	write0	0	1
18	0	0	0	1	read0	read1	1	1
19	0	0	1	0	read1	IOWait	0	1
20	0	1	0	0	write0	IOWait	0	1

b.

	Q_U	Q_V
1	u1	v1
2	u2	v1
3	u2	v1
4	u2	v2
5	u3	v4
6	u3	v4
7	u4	v3
8	u4	v3
9	u4	v3
10	u4	v3
11	u4	v3
12	u4	v3
13	u4	v3
14	u1	v4
15	u1	v4
16	u1	v2
17	u1	v2
18	u2	v4
19	u3	v2
20	u3	v1

c.

	x_1	x_2	g_1	g_2	Q_U	Q'_U	Q'_V	y_1	y_2
1	-	-	0	0	u1	u2	v1	0	0
2	0	1	0	0	u2	u2	v1	0	0
3	-	-	0	1	u2	u2	v2	1	0
4	1	-	0	0	u2	u3	v4	1	0
5	-	1	1	1	u3	u3	v4	1	0
6	-	-	1	0	u3	u4	v3	0	1
7	0	0	0	-	u4	u4	v3	0	1
8	1	0	0	-	u4	u2	v1	0	1
9	0	1	1	0	u4	u2	v4	0	0
10	1	1	0	0	u4	u3	v1	1	1
11	0	1	1	1	u4	u1	v4	1	1
12	1	1	0	1	u4	u1	v2	0	0
13	-	0	1	-	u4	u2	v2	0	1
14	0	0	1	1	u1	u1	v4	1	1
15	0	1	1	1	u1	u2	v4	0	0
16	1	1	0	1	u1	u1	v2	0	0
17	1	0	0	1	u1	u3	v1	0	1
18	0	0	1	0	u2	u3	v2	1	1
19	0	0	0	0	u3	u4	v3	0	1
20	0	1	0	0	u3	u4	v3	0	1

d.

	x_3	x_4	Q_V	g_1	g_2
1	0	0	v1	0	0
2	1	-	v1	0	1
3	0	-	v2	0	1
4	1	0	v2	0	0
5	0	0	v3	0	0
6	0	1	v3	0	1
7	1	0	v3	1	0
8	1	1	v3	1	1
9	0	1	v4	1	0
10	1	-	v4	1	1

3. Algorytm funkcjonalnej dekompozycji symbolicznej automatów skończonych

3.1. Zarys algorytmu

Algorytm implementacji automatów skończonych do układów FPGA, opisany szczegółowo poniżej, operuje na nakryciach generowanych przez zmienne wejściowe, wyjściowe i pośrednie przedstawione na rysunku 1.

Algorytm składa się z pięciu powtarzających się iteracyjnie etapów:

1. Wyznaczenie nakryć generowanych przez definicję automatu (w szczególności β_Q i β_F).
2. Wyznaczenie nakryć generowanych przez bezpośrednie i pośrednie podzbiory zmiennych wyjściowych (β_U i β_V).
3. Konstrukcja nakrycia generowanego przez bezpośredni podzbiór zmiennej stanów automatu (β_{Qu}) na podstawie r-przydatności.
4. Konstrukcja nakrycia generowanego przez blok G (β_G) na podstawie kolorowania grafu.
5. Konstrukcja nakrycia generowanego przez pośredni podzbiór zmiennej stanów automatu (β_{Qv}) na podstawie kolorowania grafu.

3.2. Nakrycia generowane przez definicję automatu

Algorytm rozpoczyna działanie od wyznaczenia nakryć generowanych przez zmienne wejściowe i wyjściowe automatu (β_{xi} i β_Y) oraz nakrycia generowane przez stany automatu (β_Q i $\beta_{Q'}$). Nakrycie generowane przez funkcję wyjściową jest równe $\beta_F = \beta_{Q'} \bullet \beta_Y$.

Dla przykładowego automatu z tabeli 1a, nakrycia β_{xi}, β_Q, $\beta_{Q'}$, β_Y i β_F są równe

β_{x1} = {1,2,3,5,6,7,9,11,13,14,15,18,19,20; 1,3,4,5,6,8,10,12,13,16,17},
β_{x2} = {1,3,4,6,7,8,13,14,17,18,19; 1,2,3,4,5,6,9,10,11,12,15,16,20},
β_{x3} = {1,2,6,7,8,10,12,16,17,18,20; 3,4,5,9,11,13,14,15,19},
β_{x4} = {1,2,3,4,7,8,9,10,13,14,16,19,20; 3,5,6,7,8,11,12,13,15,17,18},
β_Q = {1; 2,3; 4; 5,6; 7,8,9,10,11,12,13; 14,15; 16,17; 18; 19; 20},
$\beta_{Q'}$ = {1,2,8; 3,13; 4,5; 6,7,19,20; 9,15; 10,17; 11,14; 12,16; 18},
β_Y = {1,2,9,12,15,16; 6,7,8,13,17,19,20; 3,4,5; 10,11,14,18},
$\beta_F = \beta_{Q'} \bullet \beta_Y$ = {1,2; 3; 4,5; 6,7,19,20; 8; 9,15; 10; 11,14; 12,16; 13; 17; 18}.

3.3. Nakrycia generowane przez podzbiory zmiennych wejściowych

W drugim etapie dekompozycji algorytm wyznacza nakrycia β_U i β_V generowane przez podzbiory zmiennych wejściowych X – podzbiór U oraz podzbiór V.

W rozważanym przykładzie podzbiór U zawiera zmienne x_1 i x_2, a podzbiór V zmienne x_3 i x_4. Oznacza to, że podziały β_U i β_V są równe $\beta_U = \beta_{x1} \bullet \beta_{x2}$ i $\beta_V = \beta_{x3} \bullet \beta_{x4}$:

β_U = {1,2,3,5,6,9,11,15,20; 1,3,4,5,6,10,12,16; 1,3,4,6,8,13,17; 1,3,6,7,13,14,18,19},
β_V = {1,2,7,8,10,16,20; 3,4,9,13,14,19; 3,5,11,13,15; 6,7,8,12,17,18}.

3.4. Nakrycie generowane przez kodowanie bezpośredniej zmiennej stanów automatu

Po wyznaczeniu nakryć β_F i β_U możliwa jest konstrukcja nakrycia β_{Qu}, generowanego przez kodowanie zmiennej stanów automatu wprowadzanej bezpośrednio na blok H.

Konstruowane na tym etapie nakrycie β_{Qu} musi spełniać warunek $\beta_{Qu} < \beta_Q$ (nakrycie to nie może wprowadzać rozdzieleń, których nie dostarcza β_Q). Konstrukcja tego nakrycia może zostać oparta na pojęciu r-przydatności: znane są już nakrycia β_U i β_F, a cała dekompozycja musi spełniać warunek $\beta_U \bullet \beta_{Qu} \bullet \beta_G \le \beta_F$. Nakrycie β_{Qu} decyduje zatem ile rozdzieleń (nie dostarczanych przez $\beta_U \bullet \beta_{Qu}$, a wymaganych przez β_F) będzie musiało dostarczyć nakrycie β_G. Z drugiej strony, nakrycie β_G musi być realizowalne w zadanej architekturze bloku G; jeśli blok G jest elementem r-wyjściowym, nakrycie β_G może mieć co najwyżej 2^r bloków.

Nakrycie β_{Qu} jest konstruowane przez łączenie bloków nakrycia β_Q (gwarantuje to spełnienie warunku $\beta_{Qu} \le \beta_Q$). Bazując na przykładowym automacie i wyznaczonych wcześniej nakryciach β_Q i β_U możliwa jest konstrukcja tabeli, której rzędy są wyznaczone przez bloki nakrycia β_Q, kolumny są wyznaczone przez bloki nakrycia β_U, a komórki zawierają iloczyny odpowiednich bloków; w przypadku przykładowego automatu tabelę taką przedstawia tabela 2. Pary nawiasów w komórkach tabeli rozdzielają wektory, których rozdzielenia wymaga nakrycie β_F; te rozdzielenia musi dostarczyć nakrycie β_G.

Tab. 2. Tabela konstrukcji nakrycia β_{Qu}
Tab. 2. The β_{Qu} blanket construction table

bloki β_Q	bloki β_U: 1,2,3,5,6,9,11,15,20	1,3,4,5,6,10,12,16	1,3,4,6,8,13,17	1,3,6,7,13,14,18,19
1	(1)	(1)	(1)	(1)
2,3	(2)(3)	(3)	(3)	(3)
4		(4)	(4)	
5,6	(5)(6)	(5)(6)	(6)	(6)
7,8,9, 10,11, 12,13	(9)(11)	(10)(12)	(8)(13)	(7)(13)
14,15	(15)			(14)
16,17		(16)	(17)	
18				(18)
19				(19)
20	(20)			

Łączenie rzędów tej tabeli jest odpowiednikiem kodowania zmiennej Q za pomocą zmiennej symbolicznej Q_U; łączenie należy przeprowadzić tak, by utrzymywać możliwie niską ilość par nawiasów w każdej z komórek – w szczególności ilość par nawiasów nie może być większa niż 2^r, gdzie r oznacza ilość wyjść bloku G. W przypadku dekompozycji przykładowego automatu, złączenie pierwszego, szóstego i siódmego rzędu, złączenie drugiego, trzeciego i ósmego rzędu oraz złączenie

czwartego, dziewiątego i dziesiątego rzędu doprowadzi do powstania następującego czteroblokowego nakrycia β_{Qu}:

β_{Qu} = {1,14,15,16,17; 2,3,4,18; 5,6,19,20; 7,8,9,10,11,12,13}.

Zmienną Q_U można zatem zakodować za pomocą symboli u_1, u_2, u_3 i u_4 (patrz tabela 1b).

3.5. Nakrycie generowane przez blok G

W czwartym kroku algorytm konstruuje nakrycie generowane przez wyjścia bloku G (nakrycie β_G). Nakrycie to musi spełniać warunek $\beta_U \bullet \beta_{Qu} \bullet \beta_G \leq \beta_F$ tak, by razem z nakryciami generowanymi przez zmienne bezpośrednio wchodzące do bloku H dostarczało wszystkich rozdzieleń wymaganych przez funkcję F.

W tym kroku algorytm wyznacza rozdzielenia wymagane przez nakrycie β_F, usuwa ze zbioru tych rozdzieleń rozdzielenia dostarczane przez nakrycia β_U i β_{Qu}, a następnie tworzy graf niezgodności. Wierzchołki tego grafu stanowią bloki nakrycia $\beta_V \bullet \beta_Q$ – nakrycia reprezentującego wejścia do bloku G; krawędzie grafu przedstawiają rozdzielenia, które blok G musi dostarczyć blokowi H, by cały system mógł poprawnie zaimplementować funkcję F.

Nakrycie β_G powstaje z połączenia bloków podziału $\beta_V \bullet \beta_Q$ (wierzchołki grafu niezgodności) w taki sposób, by blok G miał maksimum 2^r bloków, a równocześnie dostarczał brakujących rozdzieleń. W tym celu stosowane jest kolorowanie grafu niezgodności.

W przypadku dekompozycji przykładowego automatu, nakrycia $\beta_V \bullet \beta_Q$ *i* $\beta_U \bullet \beta_{Qu}$ są równe

$\beta_V \bullet \beta_Q$ = {1; 2; 3; 4; 5; 6; 7,8,10; 7,8,12; 9,13; 11,13; 14; 15; 16; 17; 18; 19; 20},
$\beta_U \bullet \beta_{Qu}$ = {1,14; 1,15; 1,16; 1,17; 2,3; 3,4; 3,18; 5,6,20; 6,19; 7,13; 8,13; 9,11; 10,12}.

W przypadku dekompozycji tego automatu nakrycie β_F wymaga 179 rozdzieleń, z czego nakrycie $\beta_U \bullet \beta_{Qu}$ dostarcza 166. Następujące, brakujące rozdzielenia muszą być dostarczone przez nakrycie β_G: 1|14, 1|15, 1|16, 1|17, 2|3, 3|4, 3|18, 5|6, 5|20, 7|13, 8|13, 9|11 i 10|12.

Pokolorowanie grafu niezgodności pozwala skonstruować następujące nakrycie β_G:

β_G = {1,2,4,7,8,10,19,20; 3,7,8,12,16,17; 6,9,13,18; 5,11,13,14,15}.

3.6. Nakrycie generowane przez kodowanie pośredniej zmiennej stanów automatu

Ostatnim etapem algorytmu jest zakodowanie zmiennej stanów automatu wprowadzanej na blok G. Podobnie jak w przypadku zmiennej Q_U, nie wszystkie rozdzielenia dostarczane przez zmienną Q są wymagane do poprawnego działania bloku G. Oczywiście wynikowe nakrycie β_{Qv} musi spełniać warunki $\beta_{Qv} \leq \beta_Q$ i $\beta_{Qv} \bullet \beta_V \leq \beta_G$.

Konstrukcja nakrycia β_{Qv} opiera się na sklejaniu bloków podziału β_Q. Nakrycie β_{Qv} musi dostarczyć rozdzieleń wymaganych przez nakrycie β_G, a nie dostarczanych przez nakrycie β_V. W praktyce konstrukcja ta polega na wyznaczeniu rozdzieleń wymaganych przez nakrycie β_G, usunięciu z nich rozdzieleń dostarczanych przez nakrycie β_V, a następnie skonstruowaniu grafu niezgodności, którego wierzchołkami są bloki nakrycia β_Q, a krawędzie przedstawiają pozostałe, wymagane przez nakrycie β_G rozdzielenia.

W rozważanym przykładzie nakrycie β_G wymaga 132 rozdzieleń; nakrycie β_V dostarcza 105 z nich. Pozostałe rozdzielenia to 1|16, 2|16, 3|4, 3|5, 3|9, 3|11, 3|13, 3|14, 3|15, 3|19, 4|9, 4|13, 4|14, 6|7, 6|8, 6|12, 6|17, 7|18, 8|18, 9|14, 9|19, 10|16, 12|18, 13|19, 14|19, 16|20 i 17|18.

W wyniku pokolorowania grafu niezgodności, nakrycie β_{Qv} jest równe

β_{Qv} = {1,2,3,20; 4,16,17,19; 7,8,9,10,11,12,13; 5,6,14,15,18}.

3.7. Końcowa dekompozycja

Bloki końcowej dekompozycja przykładowego automatu przedstawionego w tabeli 1a znajdują się w tabelach 1c i 1d. Jak widać, dekompozycja ta koduje stany automatu na dwóch zmiennych symbolicznych (o czterech symbolach każda), a skonstruowany blok G jest możliwy do zrealizowania w komórce o czterech wejściach i dwóch wyjściach. Do bloku H należy zastosować kolejną iterację procesu dekompozycji (ponownie korzystając z powyższego algorytmu) w celu oderwania od niego kolejnego bloku 4/2.

4. Podsumowanie

Jak wspomniano we wprowadzeniu, dotychczasowe metody implementacji automatów skończonych w układach FPGA składają się z dwóch etapów: kodowania stanów automatu i mapowania zakodowanego automatu w strukturze docelowej. Algorytmy kodowania, takie jak opisane w [7, 8, 9] Nova, Jedi, hot-one itp., zostały zaprojektowane w celu uproszczenia etapu mapowania; niestety, żaden z nich nie daje optymalnych wyników w przypadku dekompozycji funkcjonalnej. Szczególnie trudno opracować jest algorytm kodowania dający dobre rezultaty w przypadku dekompozycji wielopoziomowej.

Zaprezentowany algorytm opisuje nowy sposób implementacji automatów skończonych usuwający problem nieoptymalności etapu kodowania. Dzięki zastosowanie zmiennych symbolicznych kodowanie stanów automatu jest wprowadzane stopniowo, w sposób optymalny dla kolejnych iteracji procesu dekompozycji. Przedstawione w [6] wyniki sugerują, że podejście to jest lepsze niż podejścia oparte na dwuetapowym procesie niezależnego kodowania stanów automatu i mapowania zakodowanej funkcji.

Praca naukowa finansowana ze środków na naukę w latach 2007-2010 jako projekt badawczy nr N517 003 32/0583.

5. Literatura

[1] J. A. Brzozowski i T. Łuba: Decomposition of Boolean Functions Specified by Cubes, [w:] Journal of Multiple-Valued Logic and Soft Computing, Vol. 9, Old City Publishing, Inc., Philadelphia, 2003, s. 377-417.

[2] M. Rawski, L. Jóźwiak i T. Łuba: Functional Decomposition with an Efficient Input Support Selection for Sub-functions Based on Information Relationship Measures, [w:] Journal of Systems Architecture, 47, 2001, s. 137-155.

[3] C. Scholl: Functional Decomposition with Application to FPGA Synthesis, Kluwer Academic Publishers, 2001.

[4] M. Rawski: The Novel Approach to FSM Synthesis Targeted FPGA Architectures, [w:] Proceedings of IFAC Workshop on Programmable Devices and Systems, PDS 2004, Kraków, s. 169-174.

[5] M. Rawski, H. Selvaraj, T. Łuba i P. Szotkowski: Application of Symbolic Functional Decomposition Concept in FSM Implementation targeting FPGA devices, [w:] Sixth International Conference on Computational Intelligence and Multimedia Applications, ICCIMA 2005, Las Vegas, s. 153-158.

[6] M. Rawski, H. Selvaraj, T. Łuba i P. Szotkowski: Multilevel Synthesis of Finite State Machines Based on Symbolic Functional Decomposition, [w:] International Journal of Computational Intelligence and Applications, w druku.

[7] G. De Micheli, R. K. Brayton i A. Sangiovanni-Vincentelli: Optimal state assignment for finite state machines, [w:] IEEE Trans. on CAD, 1985, s. 269-284.

[8] T. Villa i A. Sangiovanni-Vincentelli: Nova: state assignment of finite state machines for optimal two-level logic implementation, [w:] IEEE Trans. on CAD, 1990, s. 905-924.

[9] B. Lin i A. R. Newton: Synthesis of multiple level logic from symbolic high-level description languages, [w:] Proc. of IFIP Int. Conf. on VLSI, s. 187-196.

Artykuł recenzowany