Mikroprogramowany układ sterujący z współdzieleniem kodów oraz rozszerzonym formatem mikroinstrukcji

Alexander Barkalov, Larysa Titarenko, Jacek Bieganowski

Streszczenie: W artykule przedstawiona została metoda syntezy umożliwiająca zmniejszenie liczby tablic LUT potrzebnych do realizacji układu mikroprogramowanego z współdzieleniem kodów. Metoda jest przeznaczona dla układów FPGA z osadzonymi blokami pamięci. Część kombinacyjna układu mikroprogramowanego jest realizowana z użyciem tablic LUT, natomiast pamięć sterująca układu jest realizowana z użyciem osadzonych bloków pamięci. Redukcja liczby tablic LUT osiągnięto dzięki wykorzystaniu klas łańcuchów pseudo-równoważnych. W artykule przedstawiono przykład zastosowania proponowanej metody oraz rezultaty eksperymentów.

Słowa kluczowe: mikroprogramowany układ sterujący, współdzielenie kodów, łańcuch bloków operacyjnych, tabela LUT, osadzony blok pamięci

1. WPROWADZENIE

Układ sterujący (US) występuje praktycznie w każdym systemie cyfrowym. Jego zadaniem jest koordynowanie pracy pozostałych elementów systemu [18]. Dobór właściwego modelu układu sterującego zależy dużym stopniu od typu algorytmu sterowania [4]. W przypadku sieci działań, w których liczba wierzchołków operacyjnych jest co najmniej dwa razy większa od liczby wierzchołków warunkowych obok dobrze znanych rozwiązań w logice szytej, bardziej korzystne może być zastosowanie układu mikroprogramowanego. Modele mikroprogramowane kojarzą się przede wszystkim z systemami komputerowymi jednak mogą one z powodzeniem zostać użyte do realizacji układu sterującego w dowolnym systemie cyfrowym.

Obecnie systemy cyfrowe bardzo często realizuje się przy użyciu układów programowalnych takich jak FPGA czy CPLD. W przypadku tych układów nadal aktualnym problemem jest redukcja zasobów sprzętowych potrzebnych do realizacji m. in. jednostek sterujących [16, 13,14, 20]. Zmniejszenie powierzchni układu pozwala zredukować koszt systemu, liczbę układów, zużycie energii i inne [17].

Układy FPGA składają się z regularnej matrycy konfigurowalnych bloków CLB, które zawierają tablice LUT o ograniczonej liczbie wejść (zazwyczaj mniej niż 6). Ograniczona liczba wejść powoduje potrzebę, dekompozycji funkcjonalnej implementowanych funkcji [19] i może prowadzić do wolnych wielopoziomowych struktur. Liczba poziomów może zostać zredukowana przez zastąpienie wielu połączonych ze sobą bloków CLB jednym osadzonym blokiem pamięci (Embedded Memory Block - EMB) [6]. Podejście to może zostać wykorzystane przy projektowaniu mikroprogramowanych układów sterujących [4].

Istotnym aspektem przy projektowaniu układów wykorzystujących automaty skończone jest wykorzystanie odpowiedniego kodowania stanów, które może w znacznym stopniu wpłynąć na rozmiar układu [17, 5, 8, 11]. W przypadku układów mikroprogramowanych możliwość wyboru dowolnego kodowania daje m. in. struktura z współdzieleniem kodów.

2. SIEĆ DZIAŁAŃ

Jednym ze sposobów reprezentacji algorytmu sterowania jest *sieć działań* (ang. Graph Scheme of Algorithm – GSA). Przykładową sieć działań przedstawiono na Rys. 1.

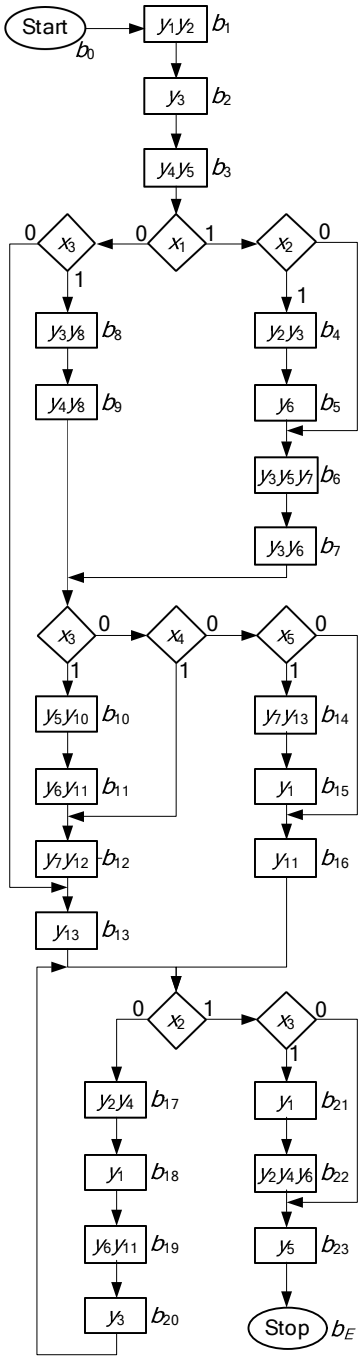
Sieć działań składa się ze zbioru wierzchołków $B=\{b_0,b_E\in E_1\cup E_2$ oraz zbioru krawędzi E. Wierzchołek b_0 jest wierzchołkiem początkowym, b_E jest wierzchołkiem końcowym, E_1 jest zbiorem wierzchołków operacyjnych, E_2 jest zbiorem wierzchołków warunkowych. Wierzchołek operacyjny $b_q\in E_1$ zawiera mikroinstrukcję $Y(b_q)\subseteq Y$ będącą kolekcją mikrooperacji ze zbioru $Y=\{y_1,\ldots, y_N\}$. Wierzchołek warunkowy $b_q\in E_2$ reprezentuje jeden z elementów $x_e\in X$ *ze* zbioru warunków $X=\{x_1,\ldots,x_L\}$.

Sekwencję kolejnych wierzchołków operacyjnych $\langle b_{g1},\ldots,b_{gFg-1}\rangle$ $(i=1,\ldots,F_g)$, w której dla każdej pary wierzchołów istnieje łuk $\langle b_{gi}, b_{gi+1}\rangle\in E$, nazywa się *łańcuchem wierzchołków operacyjnych* (Operational Linear Chain – OLC).

Niech $C=\{\alpha_1,\ldots, \alpha_G\}$ będzie zbiorem łańcuchów wierzchołków operacyjnych w sieci działań.

Dwa łańcuchy nazywa się *pseudorównoważnymi* (pseudoequivalent OLC – POLC) jeżeli ich wyjścia są połączone z wejściem tego samego wierzchołka. Niech $\Pi_C=\{B_1,\ldots,B_I\}$ będzie podziałem zbioru C na klasy POLC, gdzie $B_i\in\Pi_C$, jeżeli wyjście łańcucha OLC nie jest połączone z wierzchołkiem końcowym sieci działań.

Sieć działań nazywa się *liniową,* gdy liczba wierzchołków operacyjnych jest co najmniej dwa razy większa niż liczba łańcuchów operacyjnych.



Rys. 1. Sieć działań

3. MIKROPROGRAMOWANY UKŁAD STERUJĄCY Z WSPÓŁDZIELENIEM KODÓW

W mikroprogramowanym układzie sterującym z współdzieleniem kodów adresowanie pamięci jest realizowane za pomocą rejestru oraz licznika. Rejestr przechowuje starszą część adresu mikroinstrukcji, która jest kodem łańcucha OLC. Licznik generuje młodszą część adresu mikroinstrukcji będącą adresem wierzchołka w ramach łańcucha OLC. Na rysunku 2 starsza część adresu została oznaczona jako τ, młodsza jako T.

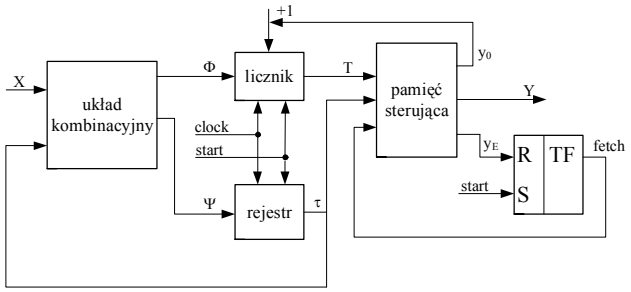
Układ jest sterowany za pomocą sygnału y_0. Jeżeli y_0=1 licznik jest inkrementowany o jeden, co odpowiada bezwarunkowemu przejściu do następnego wierzchołka w ramach łańcucha OLC. Jeżeli y_0=0 zawartość licznika i rejestru jest określana przez blok kombinacyjny, co odpowiada warunkowemu przejściu między łańcuchami OLC. Adres mikroinstrukcji dla wierzchołka b_q można przedstawić jako

$$A(b_q) = K(\alpha_g) * K(b_q) \tag{1}$$

gdzie, $K(\alpha_g)$ jest kodem łańcucha OLC, $K(b_q)$ jest kodem wierzchołka w ramach łańcucha OLC, a symbol '*' jest operatorem konkatenacji. Nowy adres mikroinstrukcji jest wtedy generowany w oparciu o bieżący stan układu oraz zewnętrzne zmienne warunkowe $x_i \in X$. Funkcje przejść układu kombinacyjnego można przedstawić następująco:

$$\begin{aligned} \Phi &= \Phi(\tau, X); \\ \Psi &= \Psi(\tau, X). \end{aligned} \tag{2}$$

Pamięć sterująca przechowuje mikroinstrukcje wystawiane przez układ do ścieżki danych. Schemat blokowy układu mikroprogramowalnego z współdzieleniem kodów przedstawiony został na rysunku 2.



Rys. 2. Struktura mikroprogramowanego układu sterującego z współdzieleniem kodów

Zaadresowanie M mikroinstrukcji wymaga adresu o długości $R = \lceil \log_2 M \rceil$ bitów. W układzie z współdzieleniem kodów adres składa się z dwóch części: kodu łańcucha OLC oraz względnego adresu wierzchołka w łańcuchu. Długość kodu łańcucha OLC, wynosi $R_\tau = \lceil \log_2 G \rceil$ bitów, gdzie G jest liczbą łańcuchów w sieci działań. Długość adresu wierzchołka zależy od liczby wierzchołków w najdłuższym łańcuchu OLC i wynosi $R_T = \lceil \log_2 Q \rceil$ bitów, gdzie Q jest liczbą wierzchołków operacyjnych najdłuższego łańcucha Q = $\max(F_1,\ldots,F_G)$, F_g jest długością łańcucha α_g.

Zastosowanie układu z współdzieleniem kodów ma sens gdy spełniony jest warunek:

$$R = R_\tau + R_T. \tag{3}$$

Spełnienie warunku (3) oznacza, że wprowadzenie adresowania z udziałem licznika i rejestru nie wpłynie na rozmiar pamięci sterującej – nie zwiększy się liczba linii adresowych.

W układzie z współdzieleniem kodów kody łańcuchów $K(\alpha_g)$ nie zależą od kodów wierzchołków operacyjnych. Możliwe jest zatem zastosowanie znanych metod kodowania stanów [7,8,9,12,13,21].

W przypadku liniowych sieci działań model z współdzieleniem kodów zawsze pochłania mniej zasobów sprzętowych w porównaniu z klasycznym automatem Moore'a [15].

4. MODYFIKACJA STRUKTURY Z WSPÓŁDZIELENIEM KODÓW

Zmniejszenie części kombinacyjnej układu z współdzieleniem kodów jest możliwe dzięki wprowadzeniu klas łańcuchów pseudorównoważnych. Liczba klas POLC w sieciach działań o charakterze liniowym jest zawsze mniejsza od liczby łańcuchów. Oznacza to, że funkcja przejść bazująca na klasach POLC składa się z mniejszej liczby przejść (krótsza tablica przejść).

Kody klas POLC można uzyskać przez konwersję adresu mikroinstrukcji (τ) za pomocą konwertera adresów [4]. Jednak blok ten jest układem kombinacyjnym i jego typowa realizacja w układach FPGA wymaga użycia bloków CLB. Innym rozwiązaniem, które nie wykorzystuje bloków CLB, jest umieszczenie kodów klas POLC w wolnym obszarze pamięci sterującej układu mikroprogramowanego.

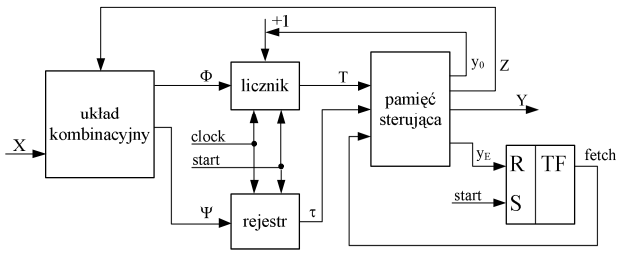
Osadzone bloki pamięci układach FPGA są konfigurowane w ograniczonym zakresie. Na przykład w układach Xilinx Virtex II blok EMB może być skonfigurowany jako: 16K x 1 bit, 8K x 2 bity, 4K x 4bity, 2K x 9bitów, 1K x 18bitów, 512 x 36 bitów [22]. Ograniczone możliwości doboru rozmiaru pamięci pociągają za sobą problemy z pełnym wykorzystaniem pamięci. Mikroprogram często nie zajmuje całej dostępnej pamięci i jej część pozostaje niewykorzystana. Ten wolny obszar pamięci może zostać użyty do przechowywania kodów klas POLC. Jeżeli wolna przestrzeń jest wystarczająca aby pomieścić kody POLC, to zagospodarowanie wolnych obszarów w pamięci sterującej pozwala na zmniejszenie liczby bloków CLB przy zachowaniu tej samej liczby bloków EMB.

Kody klas POLC można umieścić w pamięci sterującej na dwa sposoby. Jeżeli w pamięci pozostało więcej niż G wolnych komórek możliwe jest wstawienie dodatkowych mikroinstrukcji sterujących z kodami klas POLC. Jeżeli w słowie pamięci pozostało $R_{\text{POLC}} = \lceil \log_2 \Pi_C \rceil$ wolnych bitów, to możliwe jest rozszerzenie formatu mikroinstrukcji o pole z kodem klasy POLC. W dalszej części artykułu rozważany będzie tylko drugi przypadek.

Na rysunku 3 została przedstawiona struktura CMCU z współdzieleniem kodów oraz rozszerzonym formatem mikroinstrukcji. Blok kombinacyjny zmodyfikowanego układu realizuje funkcję przejść:

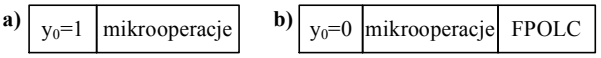
$$\begin{aligned} \Phi &= \Phi(X, Z); \\ \Psi &= \Psi(X, Z). \end{aligned} \tag{4}$$

Przejścia warunkowe między łańcuchami OLC są wykonywane w oparciu o warunki zewnętrzne (X) oraz kody klas POLC (Z). Działanie układu jest analogiczne jak w przypadku podstawowej struktury z współdzieleniem kodów.



Rys. 3. Struktura układu mikroprogramowanego z rozszerzonym formatem mikroinstrukcji

Zmodyfikowany format mikroinstrukcji przedstawiono na rysunku 4b. W porównaniu do podstawowego formatu (Rys. 4a) mikroinstrukcja została rozszerzona o pole FPOLC składające się z R_{POLC} bitów.



Rys. 4. Podstawowy **(a)** i zmodyfikowany **(b)** format mikroinstrukcji

Proponowana metoda syntezy składa się z następujących kroków:

1. Utworzenie zbioru łańcuchów operacyjnych oraz jego podział na klasy POLC.
2. Zakodowania łańcuchów OLC, wierzchołków operacyjnych oraz klas POLC.
3. Przygotowanie zawartości pamięci sterującej.
4. Przygotowanie tabeli przejść.
5. Implementacja układu.

5. PRZYKŁAD

W wyniku analizy sieci działań przedstawionej na rysunku 1 można utworzyć następujące zbiory $\Pi_C=\{B_1, B_2, B_3\}$ gdzie $B_1=\{\alpha_1\}$, $B_2=\{\alpha_2, \alpha_3, \alpha_4\}$, $B_2=\{\alpha_5, \alpha_6, \alpha_7\}$, $B_4=\{\alpha_8\}$, $\alpha_1=\langle b_1, b_2, b_3\rangle$, $\alpha_2=\langle b_4, b_5, b_6, b_7\rangle$, $\alpha_3=\langle b_8, b_9\rangle$, $\alpha_4=\langle b_{10}, b_{11}, b_{12}, b_{13}\rangle$, $\alpha_5=\langle b_{14}, b_{15}, b_{16}\rangle$, $\alpha_6=\langle b_{17}, \ldots, b_{20}\rangle$, $\alpha_8=\langle b_{21}, \ldots, b_{23}\rangle$. Oznacza to, że Q=4, R_T=2, $T=\{T_1, T_2\}$, G=6, R_τ=3, $\tau=\{\tau_1, \tau_2, \tau_3,\}$,$\Psi=\{D_1, D_2, D_3\}$, $\Phi=\{D_4, D_5\}$. Ponieważ M=23, R=5 warunek (3) jest spełniony i korzystne jest zastosowanie współdzielenia kodów.

Niech łańcuchy OLC zostaną zakodowane w następujący sposób $K(\alpha_1) = 000$, …, $K(\alpha_8) = 111$, a poszczególne elementy łańcucha mają kody: 000 dla pierwszego, 001 dla drugiego, 010 dla trzeciego itd. Niech klasy POLC będą zakodowane następująco $K(B_1)$=00, $K(B_2)$=01, $K(B_2)$=10.

Zawartość pamięci sterującej można otrzymać zamieniając każdy wierzchołek $b_q \in B$ na zbiór $Y(B_q) \cup \{y_0\}$, a wierzchołek końcowy b_{21} na zbiór $Y(B_q) \cup \{y_E\}$. Na przykład wierzchołek b_8 zawierać będzie zbiór $Y(b_8)=\{y_1, y_7\}$, natomiast odpowiadająca mu komórka pamięci o adresie $A(b_8)$=01100 będzie zawierała mikroinstrukcję odpowiadającą sygnałom y_0, y_1, y_7. Z wierzchołkiem końcowym b_{21}, będzie związany zbiór $Y(b_{21})=\{y_7, y_8\}$, adres $A(b_{21})$=11111 oraz mikroinstrukcja zawierająca sygnałom y_7, y_8, y_E.

T_1T_2 \ $\tau_1\tau_2\tau_3$	000	001	010	011	100	101	110
00	b_1	b_4	b_8	b_{10}	b_{14}	b_{17}	b_{21}
01	b_2	b_5	b_9	b_{11}	b_{15}	b_{18}	b_{22}
10	b_3	b_6	*	b_{12}	b_{16}	b_{19}	b_{23}
11	*	b_7	*	b_{13}	*	b_{20}	*

Rys. 5. Adresy mikroinstrukcji dla sieci działań z rysunku 1

Przejścia pomiędzy łańcuchami OLC mogą zostać przedstawione następująco:

$$\begin{aligned} b_3 &\to x_1x_2b_4 \vee x_1\bar{x}_2b_6 \vee \bar{x}_1x_3b_8 \vee \bar{x}_1\bar{x}_3b_{13};\\ b_9, b_7 &\to x_3b_{10} \vee \bar{x}_3x_4b_{12} \vee \bar{x}_3\bar{x}_4x_5b_{14} \vee \bar{x}_3\bar{x}_4\bar{x}_5b_{16};\\ b_{13}, b_{16}, b_{20} &\to \bar{x}_2b_{17} \vee x_2x_3b_{21} \vee x_2\bar{x}_3b_{23}. \end{aligned} \quad (5)$$

Na podstawie przejść (5) wyjścia łańcuchów OLC można opisać za pomocą jednej linii. Możliwa jest więc zamiana wyjść łańcuchów OLC na odpowiadające im klasy łańcuchów. W wyniku zamiany otrzymujemy system:

$$\begin{aligned} B_1 &\to x_1x_2b_4 \vee x_1\bar{x}_2b_6 \vee \bar{x}_1x_3b_8 \vee \bar{x}_1\bar{x}_3b_{13};\\ B_2 &\to x_3b_{10} \vee \bar{x}_3x_4b_{12} \vee \bar{x}_3\bar{x}_4x_5b_{14} \vee \bar{x}_3\bar{x}_4\bar{x}_5b_{16}\\ B_3 &\to \bar{x}_2b_{17} \vee x_2x_3b_{21} \vee x_2\bar{x}_3b_{23}. \end{aligned} \quad (6)$$

System (6) można przekształcić na tabelę przejść (Tab. 1). W kolumnach Ψ_h Φ_h umieszczono funkcje wejściowe, które przyjmują wartość '1' w h-tej linii tabeli (h=1, …, H). Poszczególne przejścia są wykonywane gdy prawdziwa jest koniunkcja zmiennych X_h (h=1,…, H).

Tabela 1.
Tabela przejść

B_i	$K(B_i)$ z_1 z_2	b_q	$A(b_q)$	X_h	Ψ_h	Φ_h	h
B_1	00	b_4	00100	x_1x_2	D_3	–	1
		b_6	01010	$x_1\bar{x}_2$	D_2	D_4	2
		b_8	01000	$\bar{x}_1x_3$	D_2	–	3
		b_{13}	01111	$\bar{x}_1\bar{x}_3$	D_2D_3	D_4D_5	4
B_2	01	b_{10}	01100	x_3	D_2D_3	D_4	5
		b_{12}	01110	$\bar{x}_3x_4$	D_2D_3	D_4	6
		b_{14}	10000	$\bar{x}_3\bar{x}_4x_5$	D_1	–	7
		b_{16}	10010	$\bar{x}_3\bar{x}_4\bar{x}_5$	D_1	D_4	8
B_3	10	b_{17}	10100	$\bar{x}_2$	D_1D_3	–	9
		b_{21}	11000	x_2x_3	D_1D_2	–	10
		b_{23}	11010	$x_2\bar{x}_3$	D_1D_2	D_4	11

Na podstawie tabeli przejść możliwe jest utworzenie systemów (4). Na przykład dla tabeli 1:

$$\begin{aligned} D_1 &= \bar{z}_1z_2\bar{x}_3\bar{x}_4 \vee z_1\bar{z}_2;\\ D_2 &= \bar{z}_1\bar{z}_2x_1\bar{x}_2 \vee \bar{z}_1\bar{z}_2\bar{x}_1 \vee \bar{z}_1z_2x_3 \vee \bar{z}_1z_2\bar{x}_3x_4 \vee z_1\bar{z}_2x_2;\\ D_3 &= \bar{z}_1\bar{z}_2x_1x_2 \vee \bar{z}_1z_2x_3 \vee \bar{z}_1z_2\bar{x}_3x_4 \vee z_1\bar{z}_2\bar{x}_2;\\ D_4 &= \bar{z}_1\bar{z}_2x_1\bar{x}_2 \vee \bar{z}_1\bar{z}_2\bar{x}_1\bar{x}_3 \vee \bar{z}_1z_2x_3 \vee \bar{z}_1z_2\bar{x}_3x_4 \vee\\ &= \bar{z}_1z_2\bar{x}_3\bar{x}_4\bar{x}_5 \vee z_1\bar{z}_2x_2\bar{x}_3;\\ D_5 &= \bar{z}_1\bar{z}_2\bar{x}_1\bar{x}_3; \end{aligned} \quad (7)$$

Składniki funkcji D_1 odpowiadają linii 5, 6,7 w tabeli przejść, składniki funkcji D_5 opowiadają linii 9 i 10 itd.

Implementacja jednostki sterującej polega na realizacji systemów (4) za pomocą tablic LUT oraz implementacji pamięci sterującej z użyciem osadzonych bloków pamięci w układach FPGA. Implementację można przeprowadzić z użyciem ogólnie dostępnych pakietów przemysłowych takich jak Xilinx ISE, Quartus II [2, 22].

6. BADANIA EKSPERYMENTALNE

Badania eksperymentalne przeprowadzono syntezując modele testowych jednostek sterujących opisanych w języku VHDL. W badaniach wykorzystane zostały algorytmy sterujące z pracy [15]. Synteza została wykonana w oprogramowaniu Xilinx ISE 10.2 dla układu FPGA Virtex II Pro (xc2vp30-5-ff896) oraz CPLD XC9500. Rezultaty badań przedstawiono w tabeli 2.

Wyniki syntezy pokazują, ze proponowana metoda pozwala zredukować powierzchnię układu sterującego o około 50% przy zachowaniu tej samej liczby przerzutników i bloków EMB w porównaniu do podstawowej struktury jednostki sterującej z współdzieleniem kodów. W układach FPGA użycie mniejszej liczby bloków CLB wiąże się z uproszczeniem sieci połączeń. W rezultacie poprawie ulegają też parametry czasowe. W przypadku badanych sieci działań czas cyklu uległ zmniejszeniu średnio o około 50% (kolumna T_C). Podobne wyniki zostały uzyskane w przypadku układów CPLD, gdzie część kombinacyjna układu jest implementowana za pomocą mikrokomórek PAL, a pamięć sterująca np. z użyciem zewnętrznych układów PROM/RAM.

Tabela 2.
Wyniki syntezy testowych sieci działań

Plik	L	N	M	G	Q	FPGA				CPLD	
						U_1		U_2		U_1	U_2
						S_F	T_C	S_F	T_C	S_C	S_C
mk01	10	7	86	14	15	39	5,11	15	2,62	208	120
mk02	10	7	90	15	10	41	5,37	15	2,13	216	119
mk03	7	8	49	11	8	20	3,46	8	2,15	138	84
mk04	9	8	57	14	10	30	5,84	12	2,12	177	108
mk05	10	9	55	13	9	36	5,82	11	2,11	239	124
mk06	8	11	59	14	7	22	5,14	13	2,08	178	158
mk07	11	8	71	17	8	38	4,32	18	2,12	357	153
mk08	7	7	50	12	9	22	6,05	11	2,15	136	107
mk09	7	7	57	13	9	27	4,35	12	2,15	146	128
mk10	13	11	93	21	7	62	13,7	23	2,15	383	211
mk11	8	8	49	13	6	29	3,81	9	2,13	185	102
mk12	11	8	72	19	7	54	12,6	19	2,11	386	174
mk13	6	7	38	9	6	14	3,92	6	2,13	125	63
mk14	8	7	54	15	6	26	4,50	11	2,12	179	104
mk15	8	10	69	15	8	27	3,77	12	2,13	250	114
mk16	6	10	72	12	11	18	4,36	8	2,15	166	86
mk17	9	7	73	15	8	24	4,45	16	2,11	223	123
mk18	6	11	52	13	8	20	4,44	10	3,77	165	126
mk19	6	7	47	12	6	16	4,46	7	2,15	122	89

L – liczba zmiennych warunkowych, N – długość mikrooperacji, M-liczba łańcuchów operacyjnych, G – liczba łańcuchów, Q – liczba wierzchołków najdłuższego łańcucha w sieci, S_F – liczba elementów slice, S_C – liczba elementów AND2, AND3, INV, OR2, OR3, XOR2 w mikrokomórce PAL, T_C – czas cyklu w nanosekundach.

7. PODSUMOWANIE

Osadzone bloki pamięci w układach mikroprogramowanych mogą być konfigurowane w ograniczonym zakresie. Oznacza to, że w wielu przypadkach część pamięci sterującej pozostaje niewykorzystana. Zaproponowana przez autorów metoda syntezy pozwala wykorzystać niezagospodarowane obszary pamięci do redukcji części kombinacyjnej mikroprogramowanej jednostki sterującej. Rozwiązanie opiera się na umieszczeniu w pamięci sterującej kodów klas łańcuchów pseudorównoważnych, dzięki czemu możliwe jest uproszczenie tabeli przejść układu.

Przeprowadzone eksperymenty dowodzą, że zaproponowana przez autorów metoda pozwala zredukować zasoby sprzętowe potrzebne do implementacji układu mikroprogramowanego z współdzieleniem kodów w przypadku liniowych sieci działań.

LITERATURA

[1] Adamski, M. and Barkalov, A. (2006). Architectural and Sequential Synthesis of Digital Devices, University of Zielona Góra Press.

[2] Altera (2009). Altera Corporation Webpage. http://www. altera.com.

[3] Baranov, S. I. (2008). Logic and System Design of Digital Systems, TUT Press.

[4] Barkalov, A. and Titarenko, L. (2008). Logic synthesis for Compositonal Microprogram Control Units, Springer, Berlin.

[5] Barkalov, A., Titarenko, L. and Wiśniewski, R. (2006). Synthesis of compositional microprogram control units with sharing codes and address decoder, Proceedings of the International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2006, pp. 397 - 400.

[6] Borowik, G., Falkowski, B. and Łuba, T. (2007). Cost-efficient synthesis for sequential circuits implemented using embedded memory blocks of FPGA's, IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, pp. 99-104.

[7] Chattopadhyay, S. (2005). Area conscious state assignment with flip-flop and output polarity selection for finite state machines synthesis - a genetic algorithm, The Computer Journal 48(4): 443-450.

[8] Czerwinski, R. and Kania, D. (2004). State assignment method for high speed FSM, Proceedings ofProgrammable Devices and Systems, pp. 216-221.

[9] Czerwinski, R. and Kania, D. (2005). State assignment for PAL-based CPLDs, Proceedings of 8th Euromicro Symposium on Digital System Design, pp. 127-134.

[10] Deniziak, S. and Sapiecha, K. (1998). An efficient algorithm of perfect state encoding for CPLD based systems, Proceedings of IEEE Workshop on Design and Diagnostic of Electronic Circuits and Systems (DDECS'98), pp. 47-53.

[11] Escherman, B. (1993). State assignment for hardwired VLSI control units, ACM Computing Surveys 25(4): 415-436.

[12] Gupta, B., Narayanan, H. and Desai, M. (1999). A state assignment scheme targeting performance and area, Proceedings of 12th International Conference on VLSI Design, pp. 378¬383.

[13] Kam, T., Villa, T., Brayton, R. and Sangiovanni-Vincentelli, A. (1998). A Synthesis of Finie State Machines: Functional Optimization, Kluwer Academic Publishers, Boston.

[14] Kania, D. (2004). The logic synthesis for the PAL-based complex programmable logic devices, Zeszyty naukowe Politechniki Śląskiej, Gliwice. (in Polish).

[15] Kołopienczyk, M. (2008). Application of address converter for decreasing memory size ofcompositional microprogram control unit with code sharing, University of Zielona Góra Press, Zielona Góra.

[16] Maxfield, C. (2004). The Design Warrior's Guide to FPGAs, Academic Press, Inc., Orlando, FL, USA.

[17] Micheli, G. D. (1994). Synthesis and Optimization of Digital Circuits, McGraw-Hill.

[18] Navabi, Z. (2007). Embedded Core Design with FPGAs, McGraw-Hill.

[19] Scholl, C. (2001). Functional Decomosition with Application of FPGA Synthesis, Kluwer Academic Publishers.

[20] Solovjev, V. V. and Klimowicz, A. (2008). Logic Design for Digital Systems on the Base of Programmable Logic Integrated Circuits, Hot line - Telecom, Moscow. (in Russian).

[21] Xia, Y. and Almani, A. (2002). Genetic algorithm based state assignment for power and area optimization, IEEE Proceedings on Computers and Digital Techniques, Vol. 149, pp. 128-133.

[22] Xilinx (2009). Xilinx Corporation Webpage. http://www. xilinx.com.

[23] Yang, S. (1991). Logic synthesis and optimization benchmarks user guide, Technical report, Microelectronic Center of North Carolina.

prof. dr hab. inż. Alexander Barkalov
Uniwersytet Zielonogórski
Wydział Elektrotechniki, Informatyki i Telekomunikacji
Instytut Informatyki i Elektroniki

ul. Licealna 9
65-417 Zielona Góra
e-mail: A.Barkalov@iie.uz.zgora.pl

dr hab. inż. Larysa Titarenko
Uniwersytet Zielonogórski
Wydział Elektrotechniki, Informatyki i Telekomunikacji
Instytut Informatyki i Elektroniki

ul. Licealna 9
65-417 Zielona Góra
e-mail: L.Titarenko@iie.uz.zgora.pl

mgr inż. Jacek Bieganowski
Uniwersytet Zielonogórski
Wydział Elektrotechniki, Informatyki i Telekomunikacji
Instytut Informatyki i Elektroniki

ul. Licealna 9
65-417 Zielona Góra
e-mail: J.Bieganowski@iie.uz.zgora.pl