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

Synteza logiczna skończonych automatów stanów z zastosowaniem wielokrotnego kodowania stanów

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. Od roku 2004 pracuje jako asystent oraz jest studentem studiów doktoranckich na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego. W roku 2006 odbył jeden semestr studiów doktoranckich na Universidade Nova de Lisboa w ramach projektu Sokrates-Erasmus.

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 zostaną przedstawione metody syntezy skończonych automatów stanów z wyjściami typu Mealy'ego do struktur programowalnych typu FPGA. Metody te oparte są o wielokrotne kodowanie stanów wewnętrznych układu podzielonych na podzbiory w oparciu o aktualny stan lub aktualnie wykonywaną mikroinstrukcje. Prowadzi to do realizacji układu cyfrowego takiego automatu w strukturze dwupoziomowej, gdzie układ pierwszego poziomu realizuje funkcje kodowania. Zabieg taki umożliwia zmniejszenie liczby funkcji logicznych realizowanych przez kombinacyjną część automatu. Stan wewnętrzny natomiast jest dekodowany w układzie drugiego poziomu na podstawie wielokrotnego kodu stanu wewnętrznego i kodu aktualnego stanu lub kodu aktualnie wykonywanej mikroinstrukcji. Ponieważ system ten posiada regularną strukturę, może on zostać zaimplementowany z wykorzystaniem osadzonych bloków pamięci, jakie występują w nowoczesnych układach FPGA. Rozwiązanie takie prowadzi do zmniejszenia całkowitej liczby bloków logicznych, i ich równomierniejszego wykorzystania, potrzebnych do implementacji układu cyfrowego skończonego automatu stanów.

Abstract

The method of synthesis for FPGAs of the logic circuit of Mealy FSM is proposed. Method is based on the encoding of internal states divided into subsets based on current state or microinstruction. It leads to realization of FSM in double-level structure. It leads to diminish number of functions realized by combinational part of FSM. The state is decoded in the second-level circuit base on multiple code and code of current state or code of microinstruction. This system can be implemented using memory blocks and in summary it leads to minimization of required logic elements for implementation of logic circuit of FSM

Słowa kluczowe: Skończony automat stanów, Układ FPGA, Synteza logiczna.

Keywords: FSM, FPGA, Logic synthesis.

1. Wstęp

Skończone automaty stanów (ang. *Finite state machines*, *FSMs*) z wyjściami typu Mealy'ego [1] są jedną z najpopularniejszych metod projektowania jednostek sterujących układów cyfrowych, przez co mają szerokie zastosowanie w informatyce jaki i również w elektronice czy telekomunikacji. Układy FPGA są bardzo często stosowane do implementacji całego systemu cyfrowego, wraz z jednostką sterującą [6]. Jedną z głównych cech układów FPGA jest występowanie w nich dużej liczby tablic LUT, które mogą realizować dowolną funkcje logiczną [4]. Ograniczeniem jednak jest stosunkowo mała liczba wejść (do 6, typowo 4) jaką posiadają tablice LUT, z drugiej strony funkcje logiczne realizowane przez kombinacyjny układ automatu posiadają znacznie więcej argumentów. Rozbieżność ta prowadzi do konieczności zastosowania dekompozycji funkcji boolowskich opisujących zachowanie automatu skończonego [5]. Negatywnym wynikiem takiej dekompozycji jest zwiększenie liczby poziomów w układzie logicznym realizującym algorytm sterowania, co może prowadzić do zmniejszenia wydajności takiego systemu.

Jedna z metod zmniejszenia liczby funkcji realizowanych przez automat jest zastosowanie implementacji wielopoziomowej [2]. Metody te wymagają zastosowania dodatkowych zmiennych wewnętrznych przez co mogą wymagać zastosowania większej liczby elementów logicznych.

W prezentowanym artykule przedstawione są metody umożliwiająca zmniejszenie liczby funkcji logicznych realizowanych przez automat. Metody te oparte są o 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 elementów logicznych potrzebnych do implementacji automatu.

2. Metody standardowe

2.1. Metoda jednopoziomowa

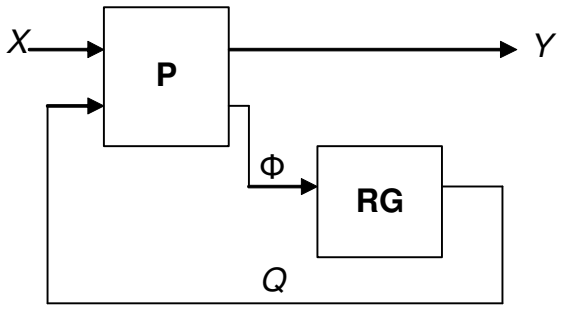
Układ logiczny skończonego automatu stanów z wyjściami typu Mealy'ego opisany jest systemem funkcji Boolowskich:

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

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 (rys. 1) [2]. W strukturze tej układ RG stanowi pamięć automatu i jest zbudowany z przerzutników typu D. Układ P implementuje funkcje Boolowskie (system (1)) automatu i w strukturze FPGA jest zbudowany z tablic LUT.



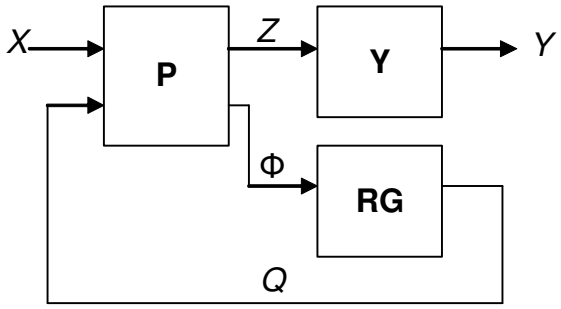
Rys. 1. Schemat blokowy automatu Mealy'ego o strukturze P
Fig. 1. The structural diagram of the P Mealy FSM

Całkowita liczba realizowanych funkcji Boolowskich wynosi:

$$n_P(P) = N + R. \quad (2)$$

2.2. Metoda z kodowaniem mikroinstrukcji

Jedną ze znanych metod zmniejszania liczby realizowanych funkcji logicznych przez układ P jest zastosowanie kodowania mikroinstrukcji [2]. 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 logiczny automatu może zostać zrealizowany z wykorzystaniem dwupoziomowej struktury PY (rys. 2) [2].



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

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

$$Y = Y(Z), \quad (3)$$

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

$$Z = Z(Q, X),$$
$$\Phi = \Phi(Q, X), \quad (4)$$

i liczba realizowanych funkcji Boolowskich zostaje zmniejszona do:

$$n_P(PY) = N + R_1. \quad (5)$$

Jednak wartość ta jest wciąż relatywnie duża i nie gwarantuje zmniejszenia liczby tablic LUT wymaganych do implementacji automatu w strukturze FPGA.

3. Metody z wielokrotnym kodowaniem

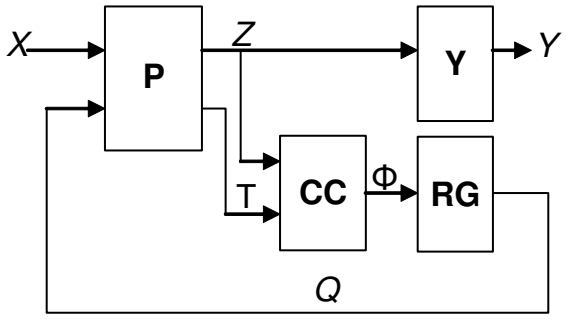
Ideą prezentowanych w tym artykule metod jest zastosowanie jednoczesnego kodowania następnego stanu automatu z wykorzystaniem kodu mikroinstrukcji lub kodu aktualnego stanu jako częściowy kod następnego stanu [3].

3.1. Z podziałem poprzez mikroinstrukcję

Najpierw zostanie omówiona metoda gdzie kod mikroinstrukcji stanowi częściowy kod następnego stanu. 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 $K(Y_t)$:

$$K(a_s) = K_t(a_s) * K(Y_t). \quad (6)$$

Układ logiczny automatu z takim kodowaniem może zostać zrealizowany z wykorzystaniem dwupoziomowej struktury PYY (rys. 3) [3].



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

Układ 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:

$$Z = Z(Q, X),$$
$$\mathrm{T} = \mathrm{T}(Q, X). \quad (7)$$

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

$$\Phi = \Phi(Z, \mathrm{T}). \quad (8)$$

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

Struktura ta pozwala na dalsze zmniejszenie liczby funkcji Boolowskich realizowanych przez automat do:

$$n_P(PYY) = R_2 + R_1. \quad (9)$$

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

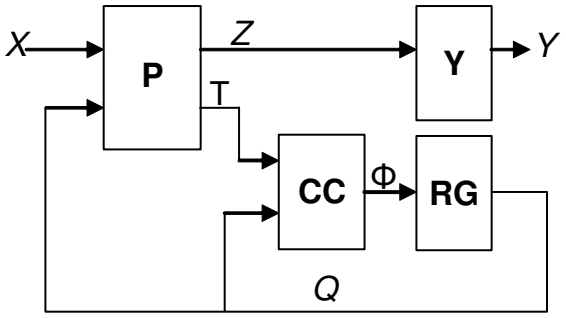
3.2. Z podziałem poprzez stan aktualny

Metoda gdzie kod aktualnego stanu jest wykorzystany jako częściowy kod stanu wewnętrznego jest podobna do metody omówionej powyżej. W tym przypadku zbiór stanów wewnętrznych $a_s \in A = \{a_1, \ldots, a_M\}$ 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 podczas ze stanu a_m. Niech $C_m = |A(a_m)|$ i $C_0 = \max(C_1, \ldots, C_M)$, i zakodujmy 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 $K(a_m)$:

$$K(a_s) = K_m(a_s) * K(a_m). \qquad (10)$$

Układ logiczny automatu z tym kodowaniem może zostać zrealizowany z wykorzystaniem dwupoziomowej struktury PAY (rys. 4) [3].



Rys. 4. Schemat blokowy automatu Mealy'ego o strukturze PAY
Fig. 4. The structural diagram of the PAY Mealy FSM

Układy RG, Y i P mają taką samą budowę i implementują te same systemy funkcji jak w strukturze PYY. Tylko układ CC implementuje inny system:

$$\Phi = \Phi(Q, \mathrm{T}) \qquad (11)$$

w porównaniu ze strukturą PYY.

Struktura ta również pozwala na dalsze zmniejszenie liczby funkcji Boolowskich realizowanych przez automat do:

$$n_{\mathrm{P}}(\mathrm{PAY}) = R_3 + R_1 \qquad (12)$$

w porównaniu ze strukturami P i PY.

Bardzo trudne jest oszacowanie relacji pomiędzy wartościami parametrów n_{P}(PYY) i n_{P}(PAY) ponieważ ich wartości są ściśle powiązane z charakterystyką konkretnie implementowanego algorytmu sterowania. Tak więc dobór struktury PYY lub PAY musi zostać wykonany eksperymentalnie.

4. Metoda syntezy

W rozdziale tym zostanie przedstawiona specjalna metoda syntezy do wcześniej przedstawionych struktur PYY i PAY (rys. 3 i 4). Metoda ta składa się z następujących kroków:

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. 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 niezależnie w każdym z podzbiorów.
4. Utworzenie tablicy przejść-wyjść dla automatu PYY lub PAY. W kroku tym tworzy się przekształconą tablicę przejść-wyjść, która jest podstawą do utworzenia systemu (7). 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 prowadzi do powstania tablicy, która jest podstawą do utworzenia systemu (3). Tablica ta posiada kolumny $K(Y_t)$, Y_t, t. Kolumna Y_t powinna być zapisana w formacie binarnym.
6. Utworzenie tablicy konwertera kodu. W kroku tym tworzy się tablicę, która jest podstawą do utworzenia systemu (8) lub (11). 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 P. Krok ten prowadzi do uzyskania równań Boolowskich opisujących system (7). Tworzone są na podstawie tablicy przejść-wyjść dla automatu PYY lub PAY.
8. Implementacja układu logicznego automatu PYY lub PAY. Krok ten jest ostatnim etapem i prowadzi do uzyskania układu logicznego realizującego algorytm sterowania. W przypadku implementacji z wykorzystaniem układów FPGA układ P i rejestr RG implementowane są z wykorzystaniem programowalnych bloków logicznych – układ P tablic LUT a rejestr RG przerzutników typu D. Układ Y implementowany jest z wykorzystaniem osadzonych bloków pamięci, gdzie $K(Y_t)$ jest adresem a Y_t jest słowem z pod tego adresu. Układ CC również jest implementowany z wykorzystaniem osadzonych bloków pamięci, gdzie adresem jest konkatenacja $K_t(a_s)$ i $K(Y_t)$ lub $K_m(a_s)$ i $K(a_m)$ a wartość słowa zapisanego pod tym adresem stanowi $K(a_s)$.

5. System A♠S

Zaproponowane metody syntezy zostały zaimplementowane w autorskim systemie do logicznej syntezy skończony automatów stanów o nazwie *Automata Synthesis* (A♠S) *System*. Oprogramowanie to zostało wykonane w środowisko *Borland C++ Builder* i pracuje w trybie konsolowym systemu *Windows*.

Wejściem do procesu syntezy, z wykorzystaniem tego oprogramowania, jest automat opisany w formacie *KISS2* [7]. Efektem końcowym syntezy jest zbiór plików opisujących poszczególne moduły wybranej struktury automatu. Pliki wyjściowe zapisane są w języku opisu sprzętu *Verilog*. Układy kombinacyjne opisane są zbiorem równań logicznych z zastosowanie konstrukcji `assign`. Zawartość pamięci opisana jest z wykorzystaniem instrukcji `case` oraz specjalnych dyrektyw umożliwiających implementację z wykorzystaniem osadzonych bloków pamięci. Generowany jest również moduł najwyższego poziomu, który zawiera opis połączeń pomiędzy poszczególnymi blokami. System generuje również plik raportu w którym między innymi podawana jest liczba realizowanych funkcji logicznych oraz wymagany rozmiar pamięci.

Zaimplementowane metody syntezy zostały przetestowane z wykorzystaniem modeli testowych z biblioteki *LGSynth91* [7]. Wyniki syntezy wybranych układów są przedstawione w tabeli 1.

Tab. 1. Wyniki syntezy logicznej
Tab. 1. Results of logic synthesis

FSM	P	PY		PAY		PYY	
	Liczba funkcji Boolowskich	Liczba funkcji Boolowskich	Rozmiar pamięci [bity]	Liczba funkcji Boolowskich	Rozmiar pamięci [bity]	Liczba funkcji Boolowskich	Rozmiar pamięci [bity]
bbsse	11	8	112	7	624	7	624
ex1	24	11	1216	9	2496	8	2496
ex4	13	8	144	5	272	6	400
ex6	11	7	128	7	320	5	224
mc	7	5	40	4	56	4	72
planet	25	12	1216	8	2752	11	13504
s1	11	10	192	9	2752	6	512
s1a	11	6	12	5	2752	6	332
s298	14	11	48	6	16432	10	8240
s386	11	8	112	7	624	7	624
s510	13	10	112	6	1648	9	3184
s820	24	10	608	8	1888	10	5728
sand	14	10	288	9	2848	9	2848
sse	11	8	112	7	624	7	624
styr	15	10	320	8	1600	9	2880

Można zauważyć, że liczba funkcji Boolowskich uzyskana dla proponowanych metod (PYY i PAY) nigdy nie jest większa niż dla metod standardowych (P i PY). Uzyskane wyniki pokazują, że metoda PAY w większości przypadków daje mniejszą liczbę funkcji Boolowskich niż metoda PYY. Jednak w niektórych przypadkach zastosowanie tej drugiej metody daje lepsze rezultaty zarówno pod względem ilości realizowanych funkcji logicznych jak i wymaganego rozmiaru pamięci. Wyniki te uzależnione są od charakterystyki konkretnego algorytmu sterowania i dobór odpowiedniej metody syntezy musi być wykonywane indywidualnie dla każdego układu sterowania.

Zauważyć można, że proponowane metody wymagają zastosowania pamięci o większym rozmiarze niż metoda PY. Należy tu jednak wspomnieć, że rozmiary osadzonych bloków pamięci w nowoczesnych układach FPGA kształtują się od 4Kb do 36Kb i w każdym układzie znajduje się od kilku do kilkunastu takich bloków. Tak więc zwiększenie rozmiaru pamięci wymaganej do implementacji układów Y i CC nie ma negatywnego wpływu na wyniki implementacji.

6. Podsumowanie

Zaproponowane w tym artykule metody syntezy logicznej skończonych automatów stanów z wyjściami typu Mealy'ego z wielokrotnym kodowaniem stanów wewnętrznych w oparciu o aktualnie wykonywaną mikroinstrukcję albo aktualny stan automatu pozwala na zmniejszenie liczby funkcji Boolowskich implementowanych poprzez kombinacyjną część automatu cyfrowego. Prowadzi to do zmniejszenia wymaganej liczby tablic LUT do implementacji układu w strukturze FPGA.

Do weryfikacji opracowanych metod zostało wykonane autorskie oprogramowanie *A♠S System*. Uzyskane wyniki pokazały, że obie metody zmniejszają liczbę funkcji Boolowskich realizowanych przez kombinacyjną część automatu. Przeanalizowane pliki testowe pokazują, że zaproponowane metody są lepsze od standardowych. Średni zysk na liczbie realizowanych funkcji logicznych wynosi 43% w porównaniu ze strukturą P i 17% w porównaniu ze strukturą PY. Wynik ten jednak mocno zależy od parametrów konkretnego algorytmu sterowania i wacha się od 0% do 68% w porównaniu ze strukturą P i od 0% do 60% w porównaniu ze strukturą PY.

7. Literatura

[1] Baranov S.: Logic Synthesis for Control Automat, Kluwer Academic Publisher, Boston, 1994.

[2] Barkalov A., Węgrzyn M.: Design of Control Units with Programmable Logic, University of Zielona Góra Press, Zielona Góra, 2006.

[3] Bukowiec A.: Synteza automatów skończonych z wykorzystaniem metod kodowania wielokrotnego, Materiały II konferencji naukowej Informatyka - sztuka czy rzemiosło KNWS'05, Złotniki Lubańskie, 17-22.

[4] Jenkins J.: Designing with FPGAs and CPLDs, Prentice Hall, New Jersey, 1994.

[5] Łuba T.: Synteza układów logicznych, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2005.

[6] Salcic Z.: VHDL and FPLDs in Digital Systems Design, Prototyping and Customization, Kluwer Academic Publishers, Boston, 1998.

[7] Yang S.: Logic Synthesis and Optimization Benchmarks User Guide. version 3.0, Technical report, Microelectronics Center of North Carolina, North Carolina, 1991.

Title: Logic Synthesis of FSMs with Multiple Encoding of States

Artykuł recenzowany