Adam KLIMOWICZ
POLITECHNIKA BIAŁOSTOCKA,
ul. Wiejska 45a, 15-351 Białystok

Badania metody minimalizacji nie w pełni określonych automatów skończonych realizowanej w oparciu o sklejanie dwóch stanów

Dr inż. Adam KLIMOWICZ

Ukończył studia magisterskie w Instytucie Informatyki Politechniki Białostockiej. Obronił rozprawę doktorską w 2007 r. na Wydziale Informatyki Politechniki Białostockiej, gdzie zajmuje stanowisko adiunkta. Jego zainteresowania naukowe to projektowanie systemów wbudowanych oraz metody syntezy układów kombinacyjnych i sekwencyjnych na bazie programowalnych układów logicznych CPLD i FPGA.

e-mail: a.klimowicz@pb.edu.pl

Streszczenie

W pracy opisano badania eksperymentalne metody minimalizacji nie w pełni określonych automatów skończonych. Proponowana metoda bazuje na operacji sklejania dwóch stanów. W pracy pokazano warunki konieczne łączenia dwóch stanów oraz przypadek tworzenia się stanów oczekiwania. Opisana metoda pozwala na redukcję liczby stanów średnio 1,16 razy i liczby przejść automatu 1,27 razy. Pozwala także na redukcję liczby przejść w stosunku do programu STAMINA średnio 1,40 razy. Przedstawiono także wyniki implementacji zminimalizowanych automatów w strukturach CPLD i FPGA, które potwierdziły skuteczność metody.

Słowa kluczowe: automat skończony, minimalizacja liczby stanów, synteza logiczna, łączenie stanów.

Experiments on the method of Mealy state machine minimization based on two-states merging

Abstract

This paper presents experiments on a heuristic method for minimization of an incompletely specified finite state machine with unspecified values of output variables. The proposed method is based on two states merging. In addition to reduction of the finite state machine (FSM) states, the method also allows reducing the number of FSM transitions and input variables. In contrast to the previously developed methods, in each step of the algorithm there is considered not only one, but the entire set of all pairs of states for which it is permissible to merge. Then from the set there is selected the pair of states which best matches the criteria of minimizing. In the paper, the conditions of state equivalence are presented. Two FSM states can be merged only if they are equivalent. It should be noted that the wait states can be formed at the merging of FSM states. This method allows reducing the number of internal states of the initial FSM by 1.16 times on the average, and by 2.75 times on occasion. An average reduction of the number of FSM transitions makes up 1.27 times. The comparison of the proposed method with the program STAMINA shows that the offered method does not reduce the number of FSM states, however it allows reducing the number of FSM transitions by 1.40 times on the average. The results of implementation of the minimized FSMs in programmable devices showed that the proposed method allowed building FSMs at lower cost and higher speed than the STAMINA program for CPLD and FPGA devices.

Keywords: finite state machine (FSM), state minimization, logic synthesis, state merging.

1. Wstęp

Automat skończony jest modelem używanym przy projektowaniu różnych struktur, takich jak układy sekwencyjne, cyfrowe układy sterowania, układy sterujące mikroprocesorów, systemy telekomunikacyjne i inne. W niektórych przypadkach przejścia pomiędzy stanami lub wartości zmiennych wyjściowych nie są całkowicie określone. Nie w pełni określonym automatem skończonym można nazwać taki automat, w którym przynajmniej jedno przejście lub jedna wartość z wektora wyjściowego nie jest określona.

Podstawy teorii nie w pełni określonych automatów skończonych opisano w pracy [1]. Standardowym podejściem do minimalizacji jest tworzenie zbiorów stanów zgodnych i znajdowanie minimalnego pokrycia tych zbiorów. Problem ten należy do klasy problemów NP-zupełnych [2] i został opisany w wielu pracach. W pracy [3] zaproponowano proste klasy podobieństwa, a zadanie rozwiązano przy użyciu programowania liniowego. W pracy [4] został opisany program STAMINA, pracujący w trybie dokładnym oraz heurystycznym używający jawnych obliczeń do rozwiązania problemu. Dokładna metoda minimalizacji używająca niejawnych obliczeń do określenia klas podobieństwa została pokazana w pracy [5]. W pracy [6] zaproponowano dokładny algorytm minimalizacji stanów bazujący na mapowaniu nie w pełni określonego automatu w strukturę drzewa automatu. Heurystyczny algorytm *void* pokazano w pracy [7]. Opiera się on na strategii uwzględniania, najpierw warunku zamknięcia, a następnie warunku pokrycia klas. W algorytmie z pracy [8] użyto techniki podziałów i ograniczeń do identyfikacji zbiorów stanów zgodnych.

W nie w pełni określonych automatach skończonych można wyróżnić dwa typy nieokreśloności:

- nieokreślone wartości zmiennych wyjściowych dla różnych wektorów wyjściowych,
- nieokreślone wartości funkcji przejść z różnych stanów dla niektórych wartości zmiennych wejściowych.

Wartości nieokreślone pierwszego typu pojawiają się w sytuacji, gdy wartość pewnej zmiennej wyjściowej w danej chwili w żaden sposób nie wpływa na funkcjonowanie sterowanego obiektu. Drugi typ nieokreśloności pojawia się gdy pewne wartości zmiennych wejściowych nigdy nie pojawiają się na wejściach automatu skończonego.

Klasycznym podejściem przy minimalizacji nie w pełni określonych automatów skończonych jest nadanie konkretnej wartości zamiast wartości nieokreślonej w celu jak najlepszej minimalizacji automatu. Jednak istnieje niebezpieczeństwo, że wartości nieokreślone pojawiły się w rezultacie błędu w opisie automatu, a automat zostanie zminimalizowany w niewłaściwy sposób. W praktyce do zwiększenia niezawodności działania systemów cyfrowych projektanci zwykle określają wartości nieokreślone funkcji przejść przejściami w dodatkowy stan, w którym formowany jest sygnał błędu i następuje powrót do stanu początkowego. Dlatego w przedstawionym podejściu określa się tylko różne wartości zmiennych wyjściowych, a pozostawia bez zmian nieokreślone wartości funkcji przejść.

2. Idea proponowanej metody

W przedstawionym podejściu, minimalizacja liczby stanów wewnętrznych odbywa się drogą kolejnego sklejania dwóch stanów, podobnie jak w pracy [9]. W odróżnieniu od metody pokazanej w [9] w tym podejściu proponuje się na każdym kroku rozpatrywać nie jedna parę stanów, ale zbiór G wszystkich par stanów, które dopuszczają sklejanie. Następnie ze zbioru G wybiera się taką parę stanów, która w największym stopniu odpowiada kryteriom minimalizacji z punktu widzenia kosztu realizacji oraz dalszej minimalizacji liczby stanów wewnętrznych automatu.

Niech $X = \{x_1,...,x_L\}$ – zbiór zmiennych wejściowych, $Y = \{y_1,...,y_N\}$ – zbiór zmiennych wyjściowych, $A = \{a_1,...,a_M\}$ – zbiór stanów wewnętrznych automatu skończonego, $D = \{d_1,\ldots,d_R\}$ – zbiór funkcji wzbudzeń elementów pamięci, gdzie R – liczba elementów pamięci (liczba bitów kodu automatu), $R \in [\text{intlog}_2 M,$

M]. Funkcjonowanie automatu będzie opisane za pomocą listy przejść, która jest tablicą złożoną z czterech kolumn: a_m, - stan bieżący, $X(a_m,a_s)$ – warunki przejścia, a_s – stan następny i $Y(a_m,a_s)$ – wektor zmiennych wyjściowych generowany podczas przejścia.

Zostaną wprowadzone następujące oznaczenia: $Z(a_i)$ – zbiór przejść ze stanu a_i; $C(a_i)$ – zbiór przejść do stanu a_i; $A(a_i)$ – zbiór stanów, do których wykonywane są przejścia ze stanu a_i, $W(a_i)$ – zbiór wektorów zmiennych wyjściowych formowanych na przejściach ze stanu a_i, $a_i \in A$.

Niech z_h – pewne przejście ze stanu a_i, $z_h \in Z(a_i)$, $a_i \in A$. Warunek przejścia jest zapisywany w postaci koniunkcji zmiennych wejściowych, można też go oznaczyć przy pomocy wektora, np. "1-10-0", gdzie '1' – oznacza, że odpowiadająca zmienna wejściowa wchodzi do koniunkcji w postaci prostej, '0' – w zanegowanej, a kreska oznacza, że wartość zmiennej nie wpływa na dane przejście. Z warunku determinizmu funkcjonowania automatu skończonego wynika, że warunki przejścia z każdego stanu automatu powinny być wzajemnie ortogonalne. Dwa warunki przejścia są ortogonalne, jeżeli chociażby na jednej pozycji mają one różne wartości znaczące (0 lub 1).

Dwa stany wewnętrzne automatu skończonego a_i i a_j mogą zostać sklejone, tzn. zastąpione jednym stanem a_{i_j}, jeśli są one równoważne. Równoważność stanów wewnętrznych automatu oznacza, że funkcjonowanie automatu nie zmienia się w rezultacie połączenia tych stanów w jeden. Funkcjonowanie automatu przy sklejeniu stanów będzie identyczne, jeżeli warunki przejść ze stanów a_i i a_j, które prowadzą do różnych stanów będą ortogonalne. Jeżeli istnieją przejścia, które ze stanów a_i i a_j wiodą do tego samego stanu, to warunki przejść dla takich stanów powinny być jednakowe. Oprócz tego wartości zmiennych wyjściowych formowanych na przejściach w jednakowe stany nie powinny być ortogonalne. Można także zauważyć, że przy sklejaniu dwóch stanów mogą pojawić się także stany oczekiwania.

Przy sklejaniu stanów wewnętrznych wektory wyjściowe z wartościami nieokreślonymi mogą łączyć się tylko, jeżeli nie są one ortogonalne. Przy czym określone wartości (0 i 1) wektorów są pozostawiane bez zmian, a wartości nieokreślone przyjmują wartość (0 lub 1) odpowiednio do wartości wektorów. Na przykład, wektory „1-0-0" i „-1010" można zastąpić wektorem „11010".

Główna strategia przedstawionego podejścia polega na znalezieniu zbioru G wszystkich par stanów, dla których są spełnione warunki sklejania. Następnie dla każdej pary stanów ze zbioru G wykonuje się próbne sklejanie stanu. Ostatecznie do sklejania wybiera się parę, która w największym stopniu spełnia kryteria optymalizacji pod względem kosztu realizacji oraz zapewnia największe możliwości sklejania innych par stanów ze zbioru G. Proces ten powtarza się dopóki można skleić chociaż jedną parę stanów automatu skończonego.

3. Sklejanie dwóch stanów

Warunkiem koniecznym i dostatecznym możliwości sklejania dwóch stanów a_i i a_j, a_i, $a_j \in A$, jest identyczność działania automatu przed i po sklejeniu tych stanów.

W sytuacji, gdy $A(a_i) \cap A(a_j) = \emptyset$, wszystkie przejścia ze stanów a_i i a_j wiodą do różnych stanów. W tym przypadku warunkiem możliwości sklejania stanów a_i i a_j jest wzajemna ortogonalność elementów zbiorów $Z(a_i) \cup Z(a_j)$. Warunek ten można zapisać następująco: dla dowolnego $X(a_i,a_h)$, $a_h \in A(a_i)$ i dowolnego $X(a_j,a_t)$, $a_t \in A(a_j)$ spełnione jest:

$$X(a_i,a_h) \perp X(a_j,a_t), \tag{1}$$

gdzie symbol „$\perp$" oznacza ortogonalność warunków.

Można zauważyć, że w tym przypadku wartości wektorów wyjściowych formowanych na przejściach ze stanów a_i i a_j, tzn. wartości zbiorów $W(a_i)$ i $W(a_j)$, nie wpływają na warunki możliwości sklejania.

W przypadku, gdy przejścia ze stanów a_i i a_j wiodą do tego samego zbioru stanów $A(a_i) = A(a_j)$, stany te mogą zostać sklejone, wtedy i tylko wtedy, gdy warunki przejść w te same stany są identyczne, a także wektory wyjściowe formowane na przejściach w jednakowe stany nie są ortogonalne. Warunek ten można zapisać w następujący sposób: dla dowolnego a_h, $a_h \in A(a_i)$ istnieje taki $X(a_j,a_h) \in Z(a_j)$, że:

$$X(a_i,a_h) = X(a_j,a_h) \text{ i } Y(a_i,a_h) \neg\perp Y(a_j,a_h), \tag{2}$$

oraz dla dowolnego a_t, $a_t \in A(a_j)$ istnieje taki $X(a_i,a_t) \in Z(a_i)$, że:

$$X(a_j,a_t) = X(a_i,a_t) \text{ i } Y(a_j,a_t) \neg\perp Y(a_i,a_t), \tag{3}$$

gdzie symbol „$\neg$" oznacza zaprzeczenie.

W sytuacji gdy $A(a_i) \neq A(a_j)$ i $A(a_i) \cap A(a_j) \neq \emptyset$, przejścia ze stanów a_i i a_j mogą prowadzić do zarówno do różnych jak i tych samych stanów. Niech $A' = A(a_i) \cap A(a_j)$; $A'(a_i) = A(a_i) \setminus A'$ i $A'(a_j) = A(a_j) \setminus A'$. W takim przypadku stany a_i i a_j mogą zostać sklejone wtedy i tylko wtedy gdy warunki (2) i (3) są spełnione dla przejść do stanów ze zbioru A', a warunek (1) jest spełniony dla przejść do stanów ze zbiorów $A'(a_i)$ i $A'(a_j)$.

Stanem oczekiwania określa się taki stan wewnętrzny automatu skończonego, z którego pewne przejście (lub kilka przejść) prowadzi do tego właśnie stanu. Wiadomym jest, że jeśli przy sklejaniu dwóch stanów jeden z nich lub oba są stanami oczekiwania, to stan po sklejeniu także będzie stanem oczekiwania. Przy sklejaniu dwóch stanów a_i i a_j pojawi się stan oczekiwania, gdy jeden ze stanów a_i lub a_j (lub oba stany) należą do zbioru $A(a_i) \cup A(a_j)$.

Jeżeli po sklejeniu stanów pojawia się stan oczekiwania, gdzie jednakowe warunki przejść ze stanów a_i i a_j wiodą do różnych stanów, wówczas takie sytuacje należy rozpatrywać oddzielnie. Innymi słowami, jeżeli $X(a_i,a_h) = X(a_j,a_t)$ i $Y(a_i,a_h) \neg\perp Y(a_j,a_t)$, ale $a_h \neq a_t$, to stany a_i i a_j mogą zostać sklejone jeżeli $a_h = a_i$ i $a_t = a_j$ lub $a_h = a_j$ i $a_t = a_i$. Proces sklejania stanów oraz powstawanie stanów oczekiwania dokładniej opisano w pracy [9].

Idea algorytmu budowy zbioru G wszystkich par stanów możliwych do sklejenia polega na kolejnym rozpatrywaniu wszystkich możliwych par stanów (a_i, $a_j \in A$, $a_i \neq a_j$). Jeżeli dla danej pary spełnione są warunki sklejania lub spełnione są warunki formowania stanu oczekiwania to para (a_i, a_j) jest dołączana do zbioru G. Można to zrobić korzystając z poniższego algorytmu.

Algorytm 1 (budowania zbioru par stanów)

1. Podstawia się $G := \emptyset$.
2. Kolejno rozpatruje się wszystkie pary stanów (a_i, a_j) ze zbioru A, $a_i \neq a_j$, a_i, $a_j \in A$. Jeśli rozpatrzono wszystkie pary wykonuje się przejście do punktu 11.
3. Dla stanów a_i, a_j określa się zbiory $Z(a_i)$ i $Z(a_j)$.
4. Dla zbiorów $Z(a_i)$ i $Z(a_j)$ kolejno rozpatruje się pary ($X(a_i,a_h)$, $X(a_j,a_t)$) warunków przejść ($X(a_i,a_h) \in Z(a_i)$, $X(a_j,a_t) \in Z(a_j)$). Jeśli wszystkie pary rozpatrzono, to dla stanów a_i, a_j spełnione są warunki sklejania, podstawia się $G := G \cup (a_i,a_j)$ i wykonuje przejście do punktu 2.
5. Sprawdza się warunki ortogonalności $X(a_i,a_h)$ i $X(a_j,a_t)$. Jeśli warunki przejść są ortogonalne, należy przejść do punktu 4.
6. Jeśli $X(a_i,a_h) \neq X(a_j,a_t)$, to nie są spełnione warunki sklejania, należy przejść do punktu 2.
7. Jeżeli $X(a_i,a_h) = X(a_j,a_t)$, określa się wektory wyjściowe $Y(a_i,a_h)$ i $Y(a_j,a_t)$, formowane na przejściach , które są inicjowane odpowiednio warunkami $X(a_i,a_h)$ i $X(a_j,a_t)$.
8. Jeśli wektory wyjściowe są ortogonalne $Y(a_i,a_h) \perp Y(a_j,a_t)$, wykonuje się przejście do punktu 2.
9. W tej sytuacji $X(a_i,a_h) = X(a_j,a_t)$ i $Y(a_i,a_h) \neg\perp Y(a_j,a_t)$. Jeżeli $a_h = a_t$, należy przejść do punktu 4.
10. W tej sytuacji $X(a_i,a_h) = X(a_j,a_t)$ i $Y(a_i,a_h) \neg\perp Y(a_j,a_t)$, ale $a_h \neq a_t$. Należy sprawdzić warunki powstawania stanu oczekiwania, jeśli są one spełnione – należy przejść do punktu 4. W przeciwnym wypadku dla stanów a_i i a_j nie są spełnione warunki sklejania. Należy przejść do punktu 2.
11. Koniec

4. Ogólny algorytm minimalizacji

Wiadomo, że sklejenie pary stanów wpływa na możliwość sklejania innych par stanów. Z tego powodu, kryterium wyboru pary stanów jest największa możliwość późniejszego sklejania stanów, co zrealizowano w poniższym algorytmie.

Algorytm 2 (szukania pary stanów do sklejenia)

1. Za pomocą algorytmu 1 określa się zbiór G par stanów, które można skleić. Jeżeli G= Ø należy przejść do punktu 5, w przeciwnym wypadku należy podstawić K:=0;
2. Elementy zbioru G są kolejno rozpatrywane.
3. Dla każdej pary stanów (a_s,a_t) $((a_s,a_t) \in G)$ wykonuje się próbne sklejanie.
4. Przy pomocy algorytmu 1 jest określany zbiór G_{st} par stanów możliwych do sklejenia po sklejeniu pary (a_i,a_j).
5. Jeżeli $|G_{st}|>K$, podstawia się K:=$|G_{st}|$, (a_i,a_j):= (a_s,a_t). Punkty 3-5 wykonuje się dla wszystkich elementów zbioru G.
6. Para (a_i,a_j) została wybrana do sklejania.
7. Koniec.

Mając na uwadze powyższe założenia, ogólny algorytm minimalizacji automatu wygląda następująco.

Algorytm 3 (ogólny algorytm minimalizacji)

1. Za pomocą algorytmu 2 wybiera się parę stanów do sklejania, jeśli para została znaleziona przechodzi się do punktu 2, w przeciwnym wypadku – do punktu 3.
2. Wykonuje się łączenie pary stanów (a_i,a_j):
   2.1. Wprowadza się nowy stan a_{i_j}, podstawia się $A(a_{i\_j}) = A(a_i) \cup A(a_j)$.
   2.2. Usuwa się stany a_i, a_j ze zbioru stanów i dołącza się stan a_{i_j}, $A := A \setminus \{a_i\}$, $A := A \setminus \{a_j\}$, and $A := A \cup \{a_{i\_j}\}$.
   2.3. Dokonuje się korekty listy przejść. Jeżeli utworzyły się identyczne przejścia, pozostawia się tylko jedno z nich. W przypadku formowania przejść różniących się w kolumnie Y (a_m,a_s) wartościami nieokreślonymi (np. „1-” i „-0” pozostawia się tylko jedno przejście określając wartości zmiennych wyjściowych (np. „10”).
3. Wykonuje się algorytm minimalizacji liczby przejść [9].
4. Wykonuje się algorytm minimalizacji liczby zmiennych wejściowych automatu zgodnie z [9].
5. Koniec.

Minimalizacja liczby zmiennych wejściowych automatu ma miejsce w sytuacji, gdy istnieją zmienne wejściowe, które nie mają wpływu na warunki przejść.

5. Warunki realizacji metody minimalizacji

Proponowana metoda minimalizacji jest skuteczna tylko wówczas, gdy początkowy opis automatu jest prawidłowy. Wymaga to spełnienia następujących warunków:

- brak powtarzających się przejść (identycznych wierszy) w początkowej liście przejść,
- deterministyczne funkcjonowanie automatu, tzn. dla każdego stanu a_i, warunki przejścia ze stanu a_i (elementy zbioru $Z(a_i)$ powinny być wzajemnie ortogonalne.

Powodem błędnego opisu automatu mogą być błędy projektantów tworzących ręcznie listę przejść lub błędy implementacyjne przy wcześniejszych etapach komputerowo wspomaganego projektowania systemów cyfrowych. Dlatego przed wykonaniem algorytmu minimalizacji należy zweryfikować początkową listę przejść automatu wykorzystując poniższy algorytm.

Algorytm 4 (weryfikacji listy przejść)

1. Kolejno przegląda się wiersze listy przejść. Jeżeli znaleziono grupę identycznych wierszy, pozostawia się tylko jeden z nich, resztę eliminując.
2. Przegląda się kolejno stany automatu. Jeżeli rozpatrzono wszystkie stany – należy przejść do punktu 6, w przeciwnym wypadku rozpatruje się kolejny stan a_i $(a_i \in A)$.
3. Określa się zbiór $Z(a_i)$ warunków przejść ze stanu a_i.
4. Sprawdza się wszystkie możliwe pary elementów zbioru $Z(a_i)$. Jeżeli wszystkie możliwe pary elementów zbioru $Z(a_i)$ zostały sprawdzone, oznacza to, że dla stanu a_i warunek determinizmu funkcjonowania jest spełniony, należy przejść do punktu 2. W przeciwnym wypadku wybierana jest następna para $(X(a_i,a_h), X(a_i,a_t))$ elementów ze zbioru $Z(a_i)$ i wykonuje się przejście do punktu 5.
5. Weryfikuje się ortogonalność warunków przejścia $X(a_i,a_h)$ i $X(a_i,a_t)$. Jeśli są one ortogonalne, należy przejść do punktu 4 w celu sprawdzenia kolejnej pary warunków. W przeciwnym wypadku funkcjonowanie automatu nie jest deterministyczne dla przejść ze stanu a_i. Wykonuje się przejście do punktu 6.
6. Koniec.

6. Badania eksperymentalne

Proponowana metoda minimalizacji została zaimplementowana jako jeden z modułów pakietu ZUBR do wspomagania procesu projektowania systemów cyfrowych. Do zbadania efektywności metody minimalizacji użyto przykładów testowych z MCNC [10].

W tab. 1 pokazano rezultaty badań algorytmu minimalizacji i przyjęto następujące oznaczenia: L_0, N_0, M_0, i P_0 to, odpowiednio, liczba wejść, wyjść, stanów i przejść dla automatu przed minimalizacją; L_1, M_1 i P_1 to liczba wejść, stanów i przejść po zastosowaniu metody minimalizacji.

Tab. 1. Rezultaty badań algorytmu minimalizacji
Tab. 1. The results of experiments on the minimization algorithm

Nazwa	L_0	N_0	M_0	P_0	L_1	M_1	P_1	M_2	P_2	M_0/M_1	P_0/P_1	P_2/P_1
bbara	4	2	10	60	4	7	35	7	42	1,43	1,71	1,20
bbsse	7	7	16	56	7	13	53	13	208	1,23	1,06	3,92
bbtas	2	2	6	24	2	6	19	6	24	1,00	1,26	1,26
dk15	3	5	4	32	3	4	30	4	32	1,00	1,07	1,07
ex1	9	19	18	233	9	18	120	18	233	1,00	1,94	1,94
keyb	7	2	19	170	7	19	167	19	170	1,00	1,02	1,02
lion9	2	1	9	25	2	4	11	4	16	2,25	2,27	1,45
s1488	8	19	48	251	8	48	236	48	251	1,00	1,06	1,06
s1494	8	19	48	250	8	48	236	48	250	1,00	1,06	1,06
s208	11	2	18	153	8	18	122	18	153	1,00	1,25	1,25
s27	4	1	6	34	4	5	25	5	30	1,20	1,36	1,20
s386	7	7	13	64	7	13	56	13	64	1,00	1,14	1,14
s420	19	2	18	137	8	18	122	18	137	1,00	1,12	1,12
s510	19	7	47	77	19	47	75	47	77	1,00	1,03	1,03
sand	11	9	32	184	11	32	125	32	184	1,00	1,47	1,47
sse	7	7	16	56	7	13	53	13	208	1,23	1,06	3,92
styr	9	10	30	166	9	30	156	30	166	1,00	1,06	1,06
tma	7	6	20	44	7	18	44	18	89	1,11	1,00	2,02
train11	2	1	11	25	2	4	12	4	15	2,75	2,08	1,25
śr. geom.										1,16	1,27	1,40

Analiza Tab.1 pokazuje, że proponowana metoda pozwala zredukować liczbę stanów średnio 1,16 razy (s szczególnych przypadkach nawet 2,75 razy). Podobnie średnia redukcja liczby przejść automatu wyniosła 1,27 razy (maksymalnie – 2,27 razy). Oprócz tego zredukowano liczbę zmiennych wejściowych z 11 do 8 w przykładzie s208 i z 19 do 8 w przykładzie s420.

Proponowana metoda została porównana z programem STAMINA [4]. Rezultaty porównania pokazano w tab. 1, gdzie parametry M_2 i P_2 oznaczają liczbę stanów i liczbę przejść po zastosowaniu programu STAMINA.

Porównanie proponowanej metody z programem STAMINA pokazało, że liczba stanów po minimalizacji jest identyczna dla proponowanej metody i programu STAMINA. Jednak opisana metoda pozwala zmniejszyć liczbę przejść średnio 1,40 razy (tab. 1) w stosunku do programu STAMINA (maksymalnie 3,92 razy).

Porównano także koszt realizacji (liczbę zajętych komórek logicznych) oraz szybkość działania (czas przejścia sygnału na krytycznej ścieżce) przy realizacji przykładów testowych w programie MAX+PLUS II dla układów programowalnych typu

CPLD (rodzina MAX3000) i FPGA (rodzina FLEX10K). Przykłady testowe zostały przekształcone do formatu TDF (język AHDL) w formie tablicowej (z użyciem konstrukcji TABLE).

W tab. 2 pokazano wyniki badań dla rodziny układów CPLD MAX3000 (EPM3256AQC208-7), gdzie C_0, C_1 i C_2 – koszt wyrażony liczbą użytych makrokomórek, odpowiednio, dla realizacji pierwotnego automatu, po minimalizacji proponowaną metodą i programem STAMINA; D_0, D_1 i D_2 – maksymalne opóźnienie (w ns) odpowiednio, dla realizacji pierwotnego automatu, po minimalizacji proponowaną metodą i programem STAMINA.

Tab. 2. Wyniki implementacji automatów w strukturze MAX3000
Tab. 2. The results of implementation of FSMs in MAX3000 structure

Nazwa	C_0	D_0	C_1	D_1	C_2	D_2	C_0/C_1	D_0/D_1	C_2/C_1	D_2/D_1
bbara	7	10,2	5	10,2	5	10,2	1,40	1,00	1,00	1,00
bbsse	16	15,7	15	15,6	16	15,6	1,07	1,01	1,07	1,00
bbtas	5	10,2	5	10,2	5	10,2	1,00	1,00	1,00	1,00
dk15	7	13	7	13	7	13	1,00	1,00	1,00	1,00
ex1	32	24,5	32	24,5	32	24,5	1,00	1,00	1,00	1,00
keyb	26	10,8	26	10,8	26	10,8	1,00	1,00	1,00	1,00
lion9	7	13	3	10,2	3	10,2	2,33	1,27	1,00	1,00
s1488	78	28,3	78	28,3	78	28,3	1,00	1,00	1,00	1,00
s1494	74	28,3	73	28,4	74	28,3	1,01	1,00	1,01	1,00
s208	9	11,2	9	11,2	9	11,2	1,00	1,00	1,00	1,00
s27	4	10,2	4	10,2	4	10,2	1,00	1,00	1,00	1,00
s386	17	16,2	17	16,2	17	16,2	1,00	1,00	1,00	1,00
s420	9	11,2	9	11,2	9	11,2	1,00	1,00	1,00	1,00
s510	24	21,6	24	21,6	24	21,6	1,00	1,00	1,00	1,00
sand	62	23,3	58	23,5	62	23,3	1,07	0,99	1,07	0,99
sse	16	15,7	15	15,6	16	15,6	1,07	1,01	1,07	1,00
styr	53	20,2	53	20,2	53	20,2	1,00	1,00	1,00	1,00
tma	24	15,7	22	13,6	33	16,2	1,09	1,15	1,50	1,19
train11	7	10,7	3	10,2	3	10,2	2,33	1,05	1,00	1,00
śr. geom.							**1,13**	**1,02**	**1,03**	**1,01**

Wyniki pokazane w tab. 2 pokazują, że proponowana metoda minimalizacji pozwala tworzyć automaty o mniejszym koszcie w strukturze CPLD z rodziny MAX3000 średnio 1,13 razy, a w stosunku do programu STAMINA – średnio 1,03 razy (maksymalnie 1,50 razy). Wzrost szybkości działania układów wyniósł średnio 1,02 razy, a w stosunku do programu STAMINA – 1,01 razy (maksymalnie 1,19 razy).

W tab. 3 pokazano wyniki badań dla rodziny układów FPGA FLEX10K (EPF10K70RC240-2), gdzie C_3, C_4 i C_5 – koszt wyrażony liczbą użytych komórek logicznych, odpowiednio, dla realizacji pierwotnego automatu, automatu po minimalizacji proponowaną metodą i minimalizacji programem STAMINA; D_3, D_4 i D_5 – maksymalne opóźnienie w nanosekundach odpowiednio, dla realizacji pierwotnego automatu, automatu po minimalizacji proponowaną metodą i minimalizacji programem STAMINA.

Tab. 3. Wyniki implementacji automatów w strukturze FLEX10K
Tab. 3. The results of implementation of FSMs in FLEX10K structure

Nazwa	C_3	D_3	C_4	D_4	C_5	D_5	C_3/C_4	D_3/D_4	C_5/C_4	D_5/D_4
bbara	35	17,8	25	16,6	26	16,7	1,40	1,07	1,04	1,01
bbsse	48	28,6	47	36,6	70	43,9	1,02	0,78	1,49	1,20
bbtas	10	15,2	9	15,9	10	15,2	1,11	0,96	1,11	0,96
dk15	49	25,9	47	27,5	49	25,9	1,04	0,94	1,04	0,94
ex1	175	45,6	168	45,7	175	45,6	1,04	1,00	1,04	1,00
keyb	102	38,4	101	35,6	102	38,4	1,01	1,08	1,01	1,08
lion9	23	21,9	8	15,1	10	20,4	2,88	1,45	1,25	1,35
s1488	363	66,2	359	64,5	363	66,2	1,01	1,03	1,01	1,03
s1494	390	73,2	355	67,4	390	73,2	1,10	1,09	1,10	1,09
s208	127	33,9	111	34,5	127	33,9	1,14	0,98	1,14	0,98
s27	40	29,5	28	24,3	35	26,1	1,43	1,21	1,25	1,07
s386	54	29,4	53	30,1	54	29,4	1,02	0,98	1,02	0,98
s420	121	36	117	38,2	121	36	1,03	0,94	1,03	0,94
s510	91	35,2	88	33,6	91	35,2	1,03	1,05	1,03	1,05
sand	193	42,9	179	47,8	193	42,9	1,08	0,90	1,08	0,90
sse	48	28,6	47	36,6	70	43,9	1,02	0,78	1,49	1,20
styr	171	42,7	177	40,3	171	42,7	0,97	1,06	0,97	1,06
tma	69	37,1	70	41,9	89	50,5	0,99	0,89	1,27	1,21
train11	24	20	8	17,2	9	20,2	3,00	1,16	1,13	1,17
śr. geom.							1,20	1,01	1,12	1,06

Wyniki pokazane w tab. 3 pokazują, że proponowana metoda minimalizacji pozwala tworzyć automaty o mniejszym koszcie w strukturze FPGA z rodziny FLEX10K średnio 1,20 razy, a w stosunku do programu STAMINA – średnio 1,12 razy (maksymalnie 1,49 razy). Wzrost szybkości działania układów wyniósł średnio 1,01 razy, a w stosunku do programu STAMINA – 1,06 razy (maksymalnie 1,35 razy).

7. Podsumowanie

W pracy przedstawiono metodę minimalizacji liczby stanów automatu skończonego w oparciu o sklejanie dwóch stanów. Metoda dorównuje programowi STAMINA jeśli chodzi o liczbę otrzymanych stanów po minimalizacji i przewyższa pod względem liczby przejść automatu. Przekłada się to na wyniki otrzymane przy implementacji przykładów testowych w strukturach CPLD i FPGA zarówno pod względem kosztu realizacji jak i szybkości działania.

Ponieważ w metodzie sklejane są ze sobą tylko dwa stany w jednym etapie minimalizacji, pozwala to na uwzględnianie w tym procesie nie tylko kryterium opartym na możliwości sklejania pozostałych stanów, ale także bezpośrednio kosztu realizacji, szybkości działania oraz pobieranej mocy, co zostanie pokazane w dalszych pracach autora.

Innym możliwym kierunkiem rozwoju przedstawionego podejścia może być uwzględnianie nieokreślonych wartości funkcji przejść jako dodatkowego warunku przy sklejaniu stanów automatu skończonego.

Artykuł opracowano w ramach pracy statutowej S/WI/1/2013 Politechniki Białostockiej.

8. Literatura

[1] Paull M., Unger S.: Minimizing the number of states in incompletely specified state machines, IRE Trans. Electron. Comput., vol. EC-8, pp. 356–367, Sept. 1959.
[2] Pfleeger C. F.: State reduction in incompletely specified finite state machines, IEEE Trans. Comput., vol. C-22, pp. 1099–1102, Dec. 1973.
[3] Grasselli A., Luccio F.: A method for minimizing the number of internal states in incompletely specified sequential networks, IRE Trans. Electron. Comput., vol. EC-14, no. 3, pp. 350–359, June 1965.
[4] Rho J.K., Hachtel G., Somenzi F., Jacoby R.: Exact and heuristic algorithms for the minimization of incompletely specified state machines, IEEE Trans. Computer-Aided Design, vol. 13, pp. 167–177, Feb. 1994.
[5] Kam T., Villa T., Brayton R., Sangiovanni-Vincentelli A.: Synthesis of FSM's: Functional Optimization. Norwell, MA: Kluwer Academic, 1997.
[6] Pena J.M., Oliveira A.L.: A new algorithm for exact reduction of incompletely specified finite state machines, IEEE Trans. Computer-Aided Design, vol. 18, pp. 1619-1632, Nov. 1999.
[7] Ahmad I., Das A.S.: A heuristic algorithm for minimization of incompletely specified finite state machines, Computers and Electrical Engineering, vol. 27, issue 2, pp. 159-172, May 2001.
[8] Gören S., Ferguson F.: On state reduction of incompletely specified finite state machines, Computers and Electrical Engineering, vol. 33, issue 1, pp. 58-69, Jan. 2007.
[9] Klimovich A.S., Solov'ev V.V. Minimization of Mealy finite-state machines by internal states gluing, Journal of Computer and Systems Sciences International, 2012, Vol 51. No. 2, pp. 244-255.
[10] Yang S.: Logic Synthesis and Optimization Benchmarks User Guide; Microelectronics Center of North Carolina, Research Triangle Park, NC, 1991.

otrzymano / received: 06.02.2014
przyjęto do druku / accepted: 01.04.2014 *artykuł recenzowany / revised paper*