Dekompozycja skończonych automatów stanów

Arkadiusz Bukowiec

Streszczenie: Popularna metodą realizacji jednostek sterujących są skończone automaty stanów. Następnie bardzo często są one implementowane do reprogramowalnych układów cyfrowych. Jednak mikroprocesory też mogą zostać wykorzystane w celu obniżenia kosztów realizacji. Niestety bardzo często okazują się one zbyt wolne do realizacji jednostek sterujących w systemach osadzonych. Dekompozycja skończonych automatów stanów może stanowić rozwiązanie dla tego problemu. Pozwala ono na równoległe wykonanie sub-automatów w różnych technologiach, zachowując wydajność, koszt i pobór energii na odpowiednim poziomie. W takim podejściu, czasowo krytyczna część jednostki sterującej (powiązana z wydzielonym sub-automatem) może zostać zrealizowana z wykorzystaniem szybkiego układu FPGA, natomiast pozostałe części można wykonać z wykorzystaniem tańszych elementów takich, jak mikroprocesory. Dodatkowo, każdy z sub-automatów może zostać poddany niezależnej syntezie i analizie z wykorzystaniem różnych, odpowiednich metod. Problem i algorytm partycjonowania skończonych automatów stanów jest przedstawiony w tym artykule. Komputerowe narzędzie opracowane przez autora jest również w nim zaprezentowane.

Słowa kluczowe: FPGA, jednostka sterująca, KISS2, dekompozycja, skończony automat stanów

1. WPROWADZENIE

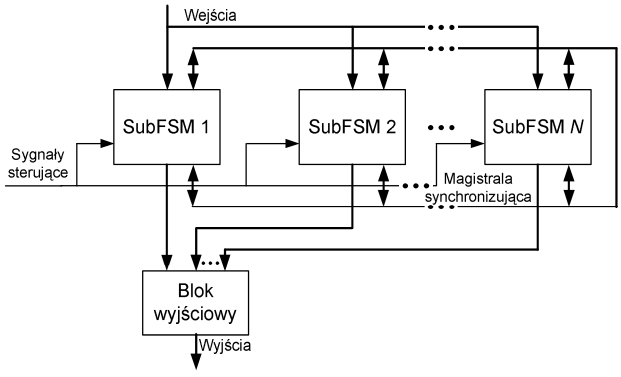
Większość systemów cyfrowych może zostać podzielonych na jednostkę sterującą i ścieżkę danych [6]. Jedne z najpopularniejszych metod specyfikacji i implementacji jednostek sterujących oparte są o zastosowanie skończonych automatów stanów (ang. *finite state machine*, FSM) [1]. Następnie są one bardzo często realizowane z wykorzystaniem programowalnych układów cyfrowych takich, jak: CPLD lub FPGA [10]. Jednak uwzględniając koszty realizacji całego systemu można tu też rozważyć zastosowanie mikrokontrolerów lub mikroprocesorów. Niestety mikroprocesory są często zbyt wolne do realizacji jednostek sterujących krytycznymi częściami systemów cyfrowych. Z drugiej strony, implementacja bardzo dużych automatów stanów do układów FPGA może być bardzo kosztowna, a wydajność może zostać obniżona (w zależności od zastosowanej metody syntezy) na skutek dużej złożoności logicznej automatu cyfrowego.

Dekompozycja skończonego automatu stanów na zbiór powiązanych automatów stanów może stanowić rozwiązanie problemu zbalansowania kosztów i wydajności [7, 9]. Zabieg ten może być również wydajny w przypadku współbieżnego projektowania sprzętu i oprogramowania (ang. *hardware/software co-design*). Umożliwi on mapowanie poszczególnych komponentów do różnych platform w zależności od wymagań projektowych.

Wstępnie przedstawiono ogólny opis proponowanej metody i zdefiniowano stosowaną notację. W następnym rozdziale został przedstawiony szczegółowy opis algorytmu i sposób jego implementacji. Następnie zaprezentowano przykład w celu zilustrowania opisanych procedur oraz przedstawienia uzyskanych wyników. Na końcu, omówiono uwagi, możliwości dalszego rozwoju i podsumowanie.

2. OGÓLNY OPIS METODY

Punktem wejściowym do procesu dekompozycji jest graf stanów automatu Mealy'ego oraz zbiór krawędzi tworzących zbiór podziału [4]. Po usunięciu tych krawędzi zostaną uzyskane niezależne sub-automaty równoległe. Należy tu zaznaczyć, że rozważenie modelu automatu Mealy'ego nie wyklucza możliwości uogólnienia metody na automaty z wyjściami typu Moore'a. Dodatkowo, do każdego sub-automatu dodawany jest stan spoczynkowy oraz zbiór wejść i wyjść służący do synchronizacji pracy z pozostałymi sub-automatami. Ogólny schemat architektury systemu po takiej dekompozycji przedstawiono na rysunku 1.



Rys. 1. Ogólny schemat architektury systemu

Wykorzystując omówioną procedurę partycjonowania, część krytyczna jednostki sterującej może zostać zaimplementowana w szybkim układzie FPGA, natomiast pozostałe komponenty można zrealizować z wykorzystaniem tańszych układów w celu obniżenia kosztów realizacji całego systemu. Z drugiej strony, każdy z sub-automatów można poddać syntezie z wykorzystaniem innych metod [2, 5].

Należy tu zaznaczyć, że wystąpienie różnych czasów wykonania dla poszczególnych komponentów może wymagać zastosowania specjalnych metod synchronizacji [8]. Problem też może zostać znacznie ograniczony poprzez zastosowanie globalnego sygnału zegarowego. Takie właśnie rozwiązanie zastosowane jest w przykładzie omawianym w tym artykule.

Niestety nie jest możliwe w pełni zautomatyzowane uzyskanie zbioru podziału bazując na matematycznych algorytmach grafowych, ponieważ graf stanu nie zawiera żadnych dodatkowych informacji na temat wymaganego czasu wykonania, prawdopodobieństwa wystąpienia przejścia, typów wykonywanych funkcji, itp. W takiej sytuacji zbiór podziału musi zostać wyznaczony przez projektanta algorytmu sterowania a system komputerowy dokona dekompozycji algorytmu na podstawie początkowego grafu stanów i wejściowego zbioru podziału.

3. DEFINICJE I REPREZENTACJA

3.1. SKOŃCZONY AUTOMAT STANÓW

Skończony automat stanów jest to szóstka:

$$S = \langle A, a_1, X, Y, \delta, \omega \rangle \tag{1}$$

gdzie:

- A jest skończonym zbiorem stanów wewnętrznych;
- a_1 jest stanem początkowym automatu;
- X jest skończonym zbiorem wejściowych zmiennych boolowskich;
- Y jest skończonym zbiorem wyjściowych zmiennych boolowskich;
- δ jest funkcją przejść automatu;
- ω jest funkcją wyjść automatu.

3.2. SUB-AUTOMAT

Sub-automat jest to ósemka:

$$SS = \langle A_S, a_1, X_S, X_S^C, Y_S, Y_S^C, \delta, \omega \rangle \tag{2}$$

gdzie:

- A_S jest skończonym zbiorem stanów wewnętrznych sub-automatu, $A_S \subseteq \mathrm{A} \cup a_s$, a_s jest dodatkowym stanem spoczynkowym sub-automatu, wymaganym przez proces synchronizacji;
- a_1 jest stanem początkowym sub-automatu, może on być równy stanowi spoczynkowemu;
- X_S jest skończonym zbiorem wejściowych zmiennych Boolowskich, $X_S \subseteq X$;
- X_S^C jest skończonym zbiorem wejściowych zmiennych Boolowskich służących do synchronizacji, pochodzących z pozostałych sub-automatów;
- Y_S jest skończonym zbiorem wyjściowych zmiennych Boolowskich, $Y_S \subseteq Y$;
- Y_S^C jest skończonym zbiorem wyjściowych zmiennych Boolowskich służących do synchronizacji, przekazywanych do pozostałych sub-automatów;
- δ jest funkcją przejść automatu;
- ω jest funkcją wyjść automatu.

3.3. DIAGRAM STANÓW

Skończony automat stanów może zostać przedstawiony na wiele sposobów. Jedną z najpopularniejszych reprezentacji jest diagram stanów [6]. Diagram stanów jest to siódemka:

$$SD = \langle A, a_1, X, Y, F, FX, FY \rangle \tag{3}$$

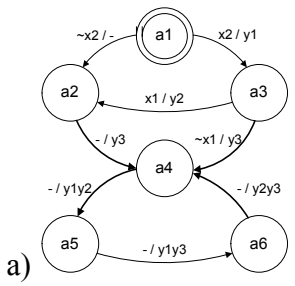
gdzie:

- A jest skończonym zbiorem wierzchołków odpowiadającym stanom wewnętrznym;
- a_1 jest wyróżnionym wierzchołkiem reprezentującym stan początkowy automatu;
- X jest skończonym zbiorem wejściowych zmiennych Boolowskich;
- Y jest skończonym zbiorem wyjściowych zmiennych Boolowskich;
- F jest skończonym zbiorem łuków, odpowiadających przejściom automatu, zdefiniowany jest jako stan źródłowy, przypisany warunek przejścia, stan docelowy oraz przypisane wyrażenie wyjściowe.
- FX jest zbiorem warunków przejść, każdy warunek przejścia zapisany jest jako iloczyn logiczny afirmacji lub negacji pewnych zmiennych wejściowych;
- FY jest zbiorem wyrażeń wyjściowych, każde wyrażenie zapisane jest jako konkatenacja pewnych zmiennych wyjściowych.

Reprezentacja ta jest wygodna dla inżynierów ze względu na czytelny zapis graficzny (rys. 2a) jednak nie jest zbyt wygodna dla komputerowych systemów przetwarzania.

3.4. FORMAT KISS2

Format KISS2 został opracowany na Uniwersytecie Berkeley [11] i jest tekstową reprezentacją jednowymiarowej tablicy przejść-wyjść automatu [10]. Opis taki składa się z nagłówka i tabeli (rys. 2b). Nagłówek zawiera informacje o liczbie wejść (.i), wyjść (.o), linii w tabeli (.p) i stanów (.s). Można w nim zawrzeć również opcjonalną informację o stanie początkowym (.r).




```
.i 2
.o 3
.s 6
.p 8
.r a1
-0 a1 a2 000
-1 a1 a3 100
-- a2 a4 001
1- a3 a2 010
0- a3 a4 001
-- a4 a5 110
-- a5 a6 101
-- a6 a4 011
```

b)

Rys. 2. Przykładowy a) graf stanów, i jego opis w b) formacie KISS2

Tablica opisuje natomiast przejścia automatu. Składa się ona z czterech kolumn: warunku przejścia, stanu źródłowego, stanu docelowego i wyrażenia wyjściowego. Symbol „-” przy zmiennej wejściowej oznacza, że nie wchodzi ona do warunku przejścia, Wartość „0” oznacza, że jej negacja wchodzi do warunku, a wartość „1”, że jej afirmacja wchodzi do warunku. Wartość „0” dla zmiennej wyjściowej oznacza, że nie wchodzi ona do wyrażenia wyjściowego natomiast wartość „1”, że wchodzi.

Dopuszczalny jest również w tym przypadku symbol „-”, który oznacza, że dana zmienna może, ale nie musi, wejść do wyrażenia wyjściowego (tzw. wartość *don't care*).

Reprezentacja ta ze względu na swój prosty zapis jest bardzo wygodna dla systemów komputerowych.

4. IMPLEMENTACJA ALGORYTMU

Celem algorytmu partycjonowania jest podzielenie jednego początkowego automatu stanu, zapisanego w formacie KISS2, na zbiór równoległych sub-automatów. W celu uzyskania tego podziału zostanie wykorzystany zbiór podziału. Zbiór podziału musi spełniać warunek taki, że po usunięciu wszystkich łuków wchodzących w jego skład z modelu bazowego uzyska się przynajmniej dwa komponenty (podgrafy) niepołączone ze sobą żadnymi łukami. Komponenty te będą stanowiły podstawę nowych sub-automatów. Procedurę partycjonowania należy zakończyć poprzez:

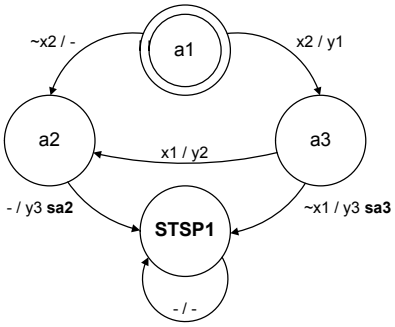
- Dodanie do sub-automatu, zawierającego stan początkowy z modelu bazowego, jednego nowego stanu. Stan ten reprezentuje wszystkie pozostałe komponenty.
- Dodanie do pozostałych sub-automatów jednego nowego stanu, który staje się ich stanem początkowym. Stan ten również reprezentuje pozostałe komponenty.
- Dodanie w każdym sub-automacie przejść pomiędzy nowym stanem a pozostałymi stanami tego automatu w oparciu o usunięte przejścia. Przypisane do nich warunki przejścia uwzględniają wejściowe zmienne boolowskie służące do synchronizacji, pochodzące z pozostałych sub-automatów. Generowane w czasie tych przejść wyrażenie wyjściowe zawierają natomiast zmienne boolowskie służące do synchronizacji, przekazywane do pozostałych sub-automatów.

Omawiana procedura została zaimplementowana w programie CAD zwanym *DivideFSM* [3]. Narzędzie to działa w linii komand i jako parametr wejściowy przyjmuje plik w formacie KISS2 opisujący początkowy automat stanów. Łuki należące do zbioru podziału powinny zostać oznaczone w tym pliku specjalną dyrektywą `#dvtr`.

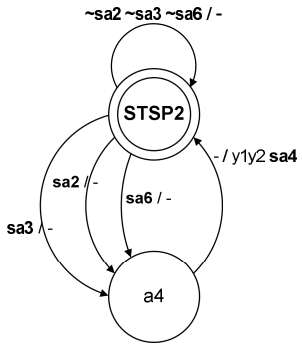
5. PRZYKŁAD

Analiza działania algorytmu zostanie zilustrowana na przykładzie automatu S_1 przedstawionym na rysunku 2a. Na początku należy zdefiniować zbiór podziału. W tym przypadku zbiór ten składa się z 4 przejść (krawędzi) zapisanych w 3, 5, 6 i 8 linii tablicy przejść-wyjść tego automatu. Jako wynik działania algorytmu uzyskuje się 3 sub-automaty SS_1, SS_2 i SS_3. Są one zapisane w formacie KISS2 a ich grafy stanów przedstawiono odpowiednio na rysunkach 3, 4 i 5.

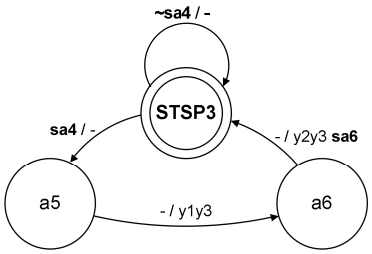
Dodane w trakcie wykonywania algorytmu stany nazywają się STSP*x* i zaznaczone są wytłuszczoną czcionką na rysunkach. Analogicznie są zaznaczone dodane zmienne boolowskie, które nazywają się sa*x*.



Rys. 3. Sub-automat SS_1



Rys. 4. Sub-automat SS_2



Rys. 5. Sub-automat SS_3

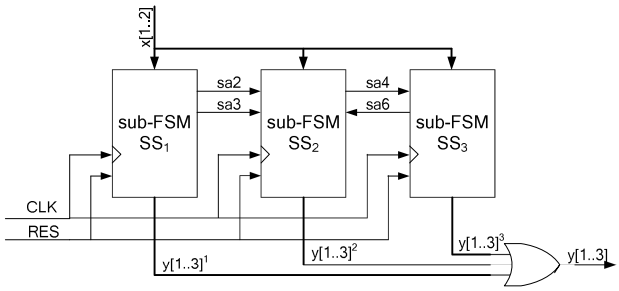
Przykładowo, sygnał sa3 generowany jest w sub-automacie SS_1 w trakcie przejścia ze stanu a3 do stanu spoczynkowego (rys. 3). Jednocześnie jest on wejściem sub-automatu SS_2 i wzbudza go do działania wywołując przejście ze stanu spoczynkowego do stanu a4 (rys. 4). Akcja ta odpowiada przejściu zapisanym w 5. linii tablicy przejść-wyjść bazowego automatu S_1 (rys. 2a).

W celu zapewnienia poprawnego działania całego systemu należy utworzyć moduł nadrzędny, który musi spełniać następujące warunki:

- wszystkie sub-automaty muszą posiadać ten sam sygnał zegarowy i zerujący,
- wszystkie automaty powinny posiadać te same sygnały wejściowe;
- dodatkowe sygnały wyjściowe muszą być połączone z odpowiednimi wejściami pozostałych sub-automatów;
- odpowiadające sobie sygnały wyjściowe ze wszystkich sub-automatów muszą być połączone w globalne sygnały wyjściowe w bloku wyjściowym za pomocą bramki OR.

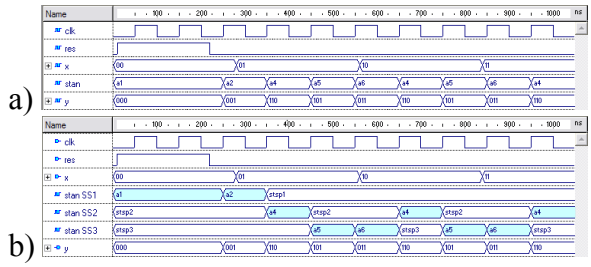
Moduł nadrzędy dla omawianego przykładu pokazano na rysunku 6.

W celu poprawnego działania układ poddano symulacji (przygotowując odpowiedni model w języku



Rys. 6. Moduł nadrzędny automat S_1

VHDL [4]). Przebieg czasowy automatu przed dekompozycją przedstawiono na rysunku 7a, a po dekompozycji na rysunku 7b.



Rys. 7. Symulacja automat S_1 a) przed b) po dekompozycji

Analizując przedstawione przebiegi czasowe widać, że układ po dekompozycji działa poprawnie. Dla ułatwienia aktywne stany poszczególnych sub-automatów wyróżniono wypełnieniem na rysunku 7b.

Uzyskane kody są w pełni syntezowalne. Weryfikacji tego procesu dokonano z wykorzystaniem narzędzi firmy Xilinx i Altera (Tab. 1).

Tabela 1.
Wyniki implementacji układu S_1 po dekompozycji

	Xilinx Virtex4 XC4VLX15 -12SF363			Altera Stratix II EP2S15-F484C3		
	LUT	FF	Slice	LUT	FF	LC
SS1	2	2	1	4	3	2
SS2	2	1	1	2	1	2
SS3	1	2	1	3	3	2
S1 top-level	6	0	4	3	0	4
S1 suma	**11**	**5**	**7**	**12**	**7**	**10**

Analizując wyniki syntezy należy zwrócić uwagę, iż podział zasobów sprzętowych na poszczególne moduły jest symboliczny, gdyż oba narzędzia starają się wykorzystać wspólne zasoby sprzętowe do realizacji logiki różnych modułów wchodzących w skład jednego układu. Wynika to z algorytmów dekompozycji funkcji logicznych [10]. Różnice w liczbie wykorzystanych przerzutników wynikają natomiast z różnych metod kodowania stanów wybranych automatycznie przez narzędzia do syntezy [2, 6, 10].

6. PODSUMOWANIE

Celem tego artykułu było przedstawienie i omówienie algorytmu dekompozycji skończonych automatów stanów i narzędzia CAD, które go implementuje. Algorytm został zilustrowany w pełni działającym przykładem.

W dalszych pracach należałoby się skupić nad opracowanie algorytmów umożliwiających automatyczne uzyskanie zbioru podziału. W tym celu należy jednak przetworzyć dodatkowe informacje takie, jak wymagany czas wykonania, czy prawdopodobieństwo wystąpienia przejścia. Implementacja takiego algorytmu może być bardzo złożona i należy rozważyć zastosowanie algorytmów genetycznych lub systemu wnioskującego do tego celu.

LITERATURA

[1] Baranov S. I., *Logic Synthesis for Control Automat*. Boston: Kluwer, 1994.
[2] Barkalov A., Titarenko L., *Logic Synthesis for FSM-based Control Units*. Berlin: Springer-Verlag, 2009.
[3] Bukowiec A., Gomes L., *DivideFSM*. `http://www.uninova.pt/gres/DivideFSM/`, 2007.
[4] Bukowiec A., Gomes L., "Partitioning of mealy finite state machines", *4th IFAC Workshop Discrete-Event System Design DESDes'09*, Gandia Beach, Hiszpania, 2009, ss. 21-26.
[5] Bukowiec A., *Synthesis of Finite State Machines for FPGA Devices Based on Architecturel Decomposition*. Zielona Góra: Univ. Ziel. Góra Press, 2009.
[6] De Micheli G., *Synthesis and Optimization of Digital Circuits*. NY: McGraw-Hill, 1994.
[7] Gomes L., Costa A., "From use cases to system implementation: Statechart based co-design", *Proc. 1st ACM & IEEE Conf. Formal Methods and Programming Models for Codesign MEMOCODE'03*, Mt St-Michel, Francja, ss. 24-33.
[8] Jantsch A., *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation*. SF: Morgan Kaufmann, 2003.
[9] Jóźwiak L., "General decomposition and its use in digital circuit synthesis", *VLSI Design*, vol. 3, ss. 225-248, 1995.
[10] Łuba T., *Synteza układów logicznych*. Warszawa: OWPW, 2005.
[11] Yang S., *Logic Synthesis and Optimization Benchmarks User Guide. v.3.0*, Raport tech.., NC: MCNC, 1991.

dr inż. Arkadiusz Bukowiec
Uniwersytet Zielonogórski
Wydział Elektrotechniki, Informatyki i Telekomunikacji
Instytut Informatyki i Elektroniki
ul. Podgórna 50
65-246 Zielona Góra

tel.: 68 328 2304
e-mail: a.bukowiec@iie.uz.zgora.pl