II Konferencja Naukowa KNWS'05
"Informatyka- sztuka czy rzemiosło"
15-18 czerwca 2005, Złotniki Lubańskie

ZASTOSOWANIE TRANSWERSALI HIPERGRAFÓW DO MINIMALIZACJI ROZMIARU PAMIĘCI JEDNOSTEK STERUJĄCYCH

Monika Wiśniewska, Remigiusz Wiśniewski

Instytut Informatyki i Elektroniki, Uniwersytet Zielonogórski
65-246 Zielona Góra, ul. Podgórna 50

e-mail: m.wisniewska@um.zielona-gora.pl
r.wisniewski@iie.uz.zgora.pl

STRESZCZENIE

W artykule zaprezentowany został nowy sposób optymalizacji rozmiaru mikrooperacji, częściowo opierający się na rozwiązaniach klasycznych. Metoda bazuje na wykorzystaniu transwersali hipergrafów do minimalnego wyznaczenia klas kompatybilności mikroinstrukcji. Mikroinstrukcje, które są parami kompatybilne mogą zostać zakodowane z wykorzystaniem mniejszej liczby bitów. Dzięki temu rozmiar pamięci układu mikroprogramowanego będzie w znacznym stopniu zmniejszony.
Idea metody zostanie zilustrowana przykładem. Pokazane zostaną wszystkie kroki, jakie są niezbędne do zaprojektowania zmodyfikowanego układu pamięci. Dokonana zostanie analiza porównawcza układu standardowego oraz zamodelowanego z wykorzystaniem proponowanej metody w strukturach FPGA.

Słowa kluczowe: hipergraf, transwersala, klasy kompatybilności, układ mikroprogramowany, mikrooperacja, mikroinstrukcja, minimalizacja rozmiaru pamięci.

1. WPROWADZENIE

Jednostka sterująca jest ważną częścią projektowanego systemu cyfrowego. Standardowa metoda implementacji jednostki sterującej w postaci skończonego automatu stanów [2, 11] często pochłania zasoby układów programowalnych, które mogą być w efektywniejszy sposób wykorzystane przez inne bloki projektowanego systemu. Coraz częściej spotykanym rozwiązaniem jest ponownie układ mikroprogramowany [4], w którym zastosowano dekompozycję jednostki sterującej na część zarządzającą (adresującą) oraz pamięć, w której przechowywane są mikroinstrukcje kontrolera.

Problemem może być implementacja układu mikroprogramowanego ze względu na ograniczenia rozmiaru słowa pamięci dostępnych w układach programowalnych [10]. Dlatego optymalizacja rozmiaru mikroinstrukcji, generowanych przez układ mikroprogramowany, jest bardzo ważnym etapem projektowania systemów cyfrowych [1].

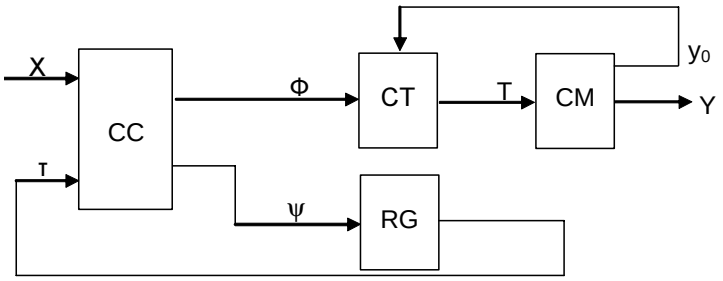
W niniejszym artykule zaproponowana zostanie nowatorska metoda zmniejszenia wielkości mikroinstrukcji przechowywanych w pamięci mikroprogramowanego układu sterującego, oparta na strukturalnej syntezie blokowej automatu cyfrowego. Metoda bazująca na formalnej dekompozycji funkcji logicznych zarysowana została w pracach [12] oraz [6]. Pokazany sposób optymalizacji rozmiaru pamięci jednostki sterującej zostanie zobrazowany przykładem.

2. PODSTAWOWE DEFINICJE

Aby przedstawić algorytm optymalizacji rozmiaru mikroinstrukcji, niezbędne będzie wprowadzenie podstawowych pojęć związanych z mikroprogramowanym układem sterującym oraz hipergrafami.

2.1. Mikroprogramowany układ sterujący

Jednostka sterująca może zostać zrealizowana jako mikroprogramowany układ sterujący [4, 11]. Podstawową cechą takiego systemu jest podział mikrokontrolera na część zarządzającą oraz część przechowującą i generującą mikrooperacje. Mikroprogramowany układ sterujący o strukturze podstawowej przedstawiony został na rys. 1.



Rys. 1. Struktura mikroprogramowanego układu sterującego

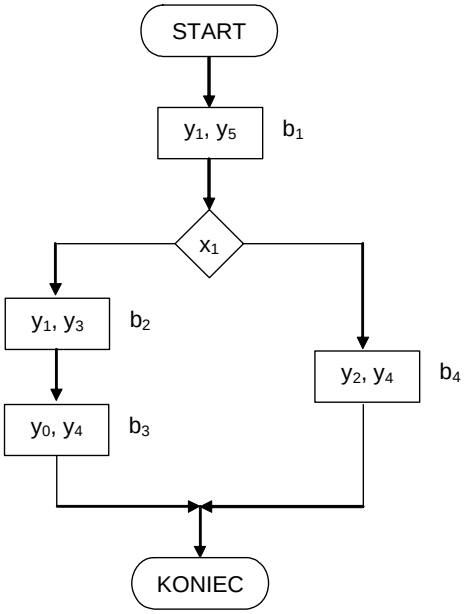
W układzie zilustrowanym poprzez rys.1. układ kombinacyjny CC oraz rejestr RG tworzą uproszczony skończony automat stanów. Automat ten jest odpowiedzialny za wyznaczanie funkcji wzbudzeń dla licznika CT. Licznik oraz pamięć układu CM tworzą blok przechowujący oraz generujący mikroinstrukcje. Struktura oraz sposoby projektowania mikroprogramowanego układu sterującego są szczegółowo opisane w literaturze [5, 13].

Podstawową zaletą tak zaprojektowanego systemu jest możliwość implementacji pamięci mikroprogramowanego układu sterującego w dedykowanych pamięciach struktur FPGA [11]. Pozostałe bloki realizowane są z wykorzystaniem podstawowych elementów logicznych matryc FPGA – bloków LUT (ang. Look-Up Tables). Takie rozwiązanie pozwala znacznie zaoszczędzić ilość wykorzystanych elementów LUT, w porównaniu do klasycznej realizacji

jednostki sterującej [4]. Problemem może być wielkość pamięci układu mikroprogramowanego. Bardzo często rozmiar mikroinstrukcji przekracza rozmiar słowa pamięci dostępnych w układach FPGA. Niezbędna jest wówczas optymalizacja pamięci jednostki sterującej. W niniejszym artykule zaproponowana zostanie metoda zmniejszenia rozmiaru mikroinstrukcji opierająca się na badaniu hipergrafu.

2.2. Organizacja pamięci mikroprogramowanego układu sterującego

Niech dany będzie algorytm sterowania, przedstawiony za pomocą sieci działań Γ [3], składającej się ze zbioru bloków operacyjnych $B=\{b_1, \ldots, b_k\}$, zbioru bloków decyzyjnych $X=\{x_1, \ldots, x_L\}$. Każdy blok operacyjny $b_k \subseteq B$ zawiera mikroinstrukcję, będącą zbiorem mikrooperacji $Y(b_k) \subseteq Y$, gdzie $Y=\{y1, \ldots y_N\}$ jest zbiorem wszystkich mikrooperacji. Każdy blok decyzyjny sieci działań zawiera warunek logiczny X_i, będący elementem zbioru warunków logicznych $X=\{x_1, \ldots, x_L\}$. Przykładowa sieć działań Γ_1 przedstawiona została na rys. 2.



Rys. 2. Przykładowa sieć działań Γ_1

Z powyższej sieci działań można w bardzo łatwy sposób wyznaczyć zawartość mikroprogramowanego układu sterującego. Mikroinstrukcje generowane w kolejnych blokach operacyjnych są adresowane z wykorzystaniem naturalnego kodu binarnego. Przykładowo dla sieci działań przedstawionej na rys. 2 zawartość pamięci będzie wyglądać tak, jak przedstawiono w tabeli 1.

Tab. 1. Zawartość pamięci mikroprogramowanego układu sterującego dla sieci działań Γ_1

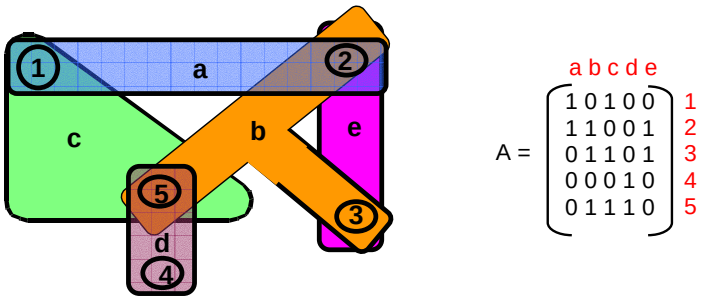
b_k	Adres (b_k)	Mikroinstrukcja						K
		y_0	y_1	y_2	y_3	y_4	y_5	
b_1	00	0	1	0	0	0	1	1
b_2	01	0	1	0	1	0	0	2
b_3	10	1	0	0	0	1	0	3
b_4	11	0	0	1	0	1	0	4

W przedstawionej tabeli poprzez b_k oznaczono kolejne węzły operacyjne sieci działań Γ_1. Adres (b_k) to kod poszczególnych węzłów operacyjnych [5]. Kolejna kolumna tabeli przedstawia rzeczywistą zawartość pamięci mikroporgramowanego układu sterującego. Przedstawiono wartości poszczególnych mikrooperacji w danym węźle operacyjnym. Mikrooperacje tworzą w ten sposób mikroinstrukcje przyszłego układu mikroprogramowanego. Poprzez K oznaczono kolejne wiersze tabeli.

2.3. Podstawowe definicje związane z hipergrafami

Hipergraf jest rozszerzeniem pojęcia grafu. Jego krawędzie mogą być incydentne do dowolnej liczby wierzchołków [9], w odróżnieniu od grafu, w którym krawędzie mogą być incydentne maksymalnie do 2 wierzchołków.

Hipergraf może być reprezentowany przez macierz incydencji, w której wiersze odpowiadają wierzchołkom, a kolumny krawędziom hipergrafu. Jeśli element macierzy jest równy 1 to j-ta krawędź jest incydentna do wierzchołka. W przeciwnym przypadku element ten jest równy 0. Na rys. 3 przedstawiono przykładowy hipergraf oraz jego reprezentację w postaci macierzy incydencji [9].



Rys. 3. Hipergraf oraz jego macierz incydencji

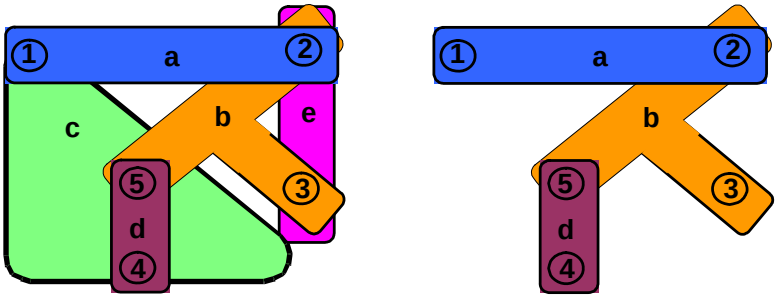
Minimalną transwersalą hipergrafu jest najmniejsza liczba krawędzi incydentnych do wszystkich wierzchołków hipergrafu. Transwersale wyznacza się analizując kolumny i wiersze macierzy incydencji hipergrafu, korzystając z przedstawionych poniżej definicji.

Istotna kolumna – istotna krawędź jest to taka krawędź, dla której istnieje wiersz z pojedynczą jedynką tzn. jedyna krawędź incydentna z wierzchołkiem. Istotne kolumny (krawędzie) muszą być częścią każdego pokrycia.

Kolumna dominuje inną kolumnę, gdy elementy poprzedniej kolumny są większe lub równe elementom następnej. Analogicznie krawędź dominuje inną krawędź, jeżeli jest incydentna do co najmniej wszystkich wierzchołków incydentnych do innej krawędzi. Zdominowane kolumny (krawędzie) można pominąć w dalszych rozważaniach.

Wiersz dominuje inny wiersz, gdy elementy poprzedniego wiersza są większe lub równe odpowiednim elementom następnego Analogicznie wierzchołek dominuje inny wierzchołek, jeżeli jest incydentny do co najmniej wszystkich krawędzi incydentnych do innego wierzchołka. Dominujące wiersze (wierzchołki) można zaniedbać, ponieważ każde pokrycie zbioru zdominowanego jest pokryciem pełnego zbioru.

Przykład minimalnej transwersali hipergrafu przedstawiono na rys. 4.



Rys. 4. Hipergraf oraz jego minimalna transwersala

3. MINIMALIZACJA PAMIĘCI MIKROPROGRAMOWANEGO UKŁADU STERUJĄCEGO Z WYKORZYSTANIEM HIPERGRAFÓW

Rozmiar pamięci mikroprogramowanego układu sterującego bardzo często przekracza zasoby dedykowanych bloków pamięci dostępnych w układach FPGA [8]. W niniejszym rozdziale przedstawiony zostanie algorytm optymalizacji rozmiaru mikroinstrukcji a w efekcie rozmiaru pamięci mikrokontrolera.

Minimalizacja rozmiaru pamięci mikroprogramowanego układu sterującego może zostać podzielona na następujące etapy:

1. Wyznaczenie klas kompatybilności (zgodności) mikrooperacji dla pamięci mikroprogramowanego układu sterującego. W tym punkcie określone zostaną klasy kompatybilności mikroinstrukcji wchodzących w skład pamięci mikroprogramowanego układu sterującego. Mikrooperacje są parami

kompatybilne jeśli nie występują w tych samych mikroinstrukcjach. Formalnie kompatybilność mikrooperacji można zapisać następująco [7]:

$$d_{k_i n_j} = \begin{cases} 1 \Rightarrow kompatybilność \\ 0 \Rightarrow brak\ kompatybilności \end{cases} \quad (1)$$

gdzie ki⊆K oznacza i-tą mikroinstrukcję (słowo) zapisaną w pamięci mikroprogramowanego układu sterującego; nj⊆N– mikrooperacja na j-tym miejscu w i-tej mikroinstrukcji.

Przykładowo dla mikrooperacji zapisanych w pamięci przedstawionej w tabeli 1 można wyznaczyć następujące zbiór C={C_1,...,C_4} czterech klas kompatybilności:

C_1=[y_0, y_1, y_2], C_2=[y_0, y_2, y_3, y_5], C_3=[y_1, y_4], C_4=[y_3, y_4, y_5].

2. Określenie wag klas kompatybilności. Waga klasy to minimalna liczba bitów, jakie są niezbędne do reprezentacji wchodzących w jej skład mikrooperacji. Waga klasy może zostać w bardzo łatwy sposób wyznaczona ze wzoru:

$$L_i =]\log_2 |C_i| + 1[\quad (2)$$

gdzie C_i oznacza i-tą klasę kompatybilności, L_i- wartość wagi klasy C_i.

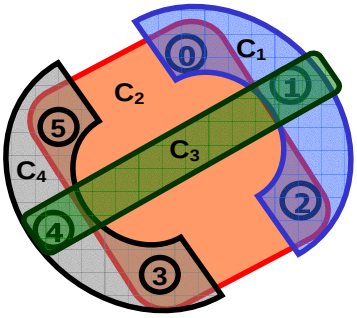
Nadmiarowy bit jest niezbędny do określenia stanu, w którym nie jest wykonywana żadna mikroinstrukcja. W opisywanym przykładzie poszczególne klasy będą miały następujące wagi:
C_1=]$\log_2$3+1[=2; C_2=]$\log_2$4+1[=3; C_3=]$\log_2$2+1[=2; C_4=]$\log_4$2+1[=2.

3. Utworzenie macierzy incydencji hipergrafu dla wyznaczonych klas kompatybilności. Kolejny krok to utworzenie macierzy incydencji hipergrafu H_1. Kolumny macierzy określają klasy kompatybilności, wiersze zaś mikrooperacje. Jeśli na przecięciu kolumny i wiersza pojawia się wartość 1 – oznacza to, że dana mikrooperacja należy do danej klasy kompatybilności. Zawartość macierzy dla rozpatrywanego przykładu przedstawiono za pomocą tabeli 2.

Tab. 2. Macierz incydencji hipergrafu H_1

Mikrooperacje	Klasy kompatybilności			
	C 1	C 2	C 3	C 4
y0	1	1	0	0
y1	1	0	1	0
y2	1	1	0	0
y3	0	1	0	1
y4	0	0	1	1
y5	0	1	0	1

Hipergraf ilustrujący klasy kompatybilności zapisane w tabeli 2 pokazany został na rys. 5. Dla uproszczenia poszczególne mikrooperacje oznaczone zostały cyframi {0,…,5}. Klasy kompatybilności oznaczone są jako krawędzie hipergrafu.



Rys. 5. Hipergraf H1

4. Wyznaczenie minimalnej transwersali hipergrafu. W tym etapie należy wyznaczyć minimalną transwersalę hipergrafu [9]. W przypadku hipergrafu przedstawionego na rys. 5 minimalna transwersala może zostać zrealizowana na dwa sposoby: poprzez klasy C_1 i C_4 lub C_2 i C_3. Aby wybrać optymalne rozwiązanie należy zsumować wagi wchodzące w skład danej transwersali. Suma wag klas C_1 oraz C_4 wynosi 4, natomiast suma wag klas C_2 i C_3 jest równa 5. Do dalszych rozważań należy więc wybrać pokrycie składające się z klas C_1 oraz C_4.

5. Kodowanie klas kompatybilności realizujących minimalną transwersalę hipergrafu. Kolejny krok to zakodowanie klas kompatybilności. Liczba zmiennych Q niezbędnych do realizacji tego zadania wynika bezpośrednio z wag przypisanych tym klasom. Suma wag klas C_1 i C_4 wynosi 4, tak więc w tym przypadku należy wykorzystać 4 zmienne $Q=\{q_1, q_2, q_3, q_4\}$. Ponieważ klasa C_1 ma wagę 2, tak więc do jej zakodowania potrzebne będą dwie zmienne (q_1 oraz q_2). Analogicznie klasa C_4 zakodowana zostanie z wykorzystaniem dwóch

zmiennych q_3 i q_4. Dobór kodu jest dowolny, aczkolwiek przypisanie pierwszego kodu (składającego się wyłącznie z 0) jest zalecany dla tych wierszy, w których nie jest generowana żadna mikrooperacja. W tabeli 3 przedstawiono przykładowe kodowanie klas C_1 i C_4.

Tab. 3. Przykładowe kodowanie klas kompatybilności

Klasa C1			Kod		Klasa C4			Kod		K
y 0	y 1	y 2	q 1	q 2	y 3	y 4	y 5	q 3	q 4	
0	1	0	0	1	0	0	1	0	1	1
0	1	0	0	1	1	0	0	1	0	2
1	0	0	1	0	0	1	0	1	1	3
0	0	1	1	1	0	1	0	1	1	4

6. Wyznaczenie pamięci mikroprogramowanego układu sterującego z zakodowanymi klasami kompatybilności. Zawartość pamięci określana jest poprzez zestawienie wszystkich zmiennych otrzymanych w punkcie 5. Wynika stąd, że rozmiar słowa pamięci (mikroinstrukcji) po optymalizacji jest równy sumie wag wszystkich klas wykorzystanych do minimalnego pokrycia. Dla rozpatrywanego przykładu suma wag C_1 i C_4 jest równa 4, wobec tego rozmiar mikroinstrukcji nowej pamięci będzie równy 4. W tabeli 4 przedstawiono zawartość pamięci układu mikroprogramowanego po optymalizacji.

Tab. 4. Zawartość pamięci mikroprogramowanego układu sterującego po optymalizacji

Bk	Adres (bk)	Mikroinstrukcja				K
		q 1	q 2	q 3	q 4	
b1	00	0	1	0	1	1
b2	01	0	1	1	0	2
b3	10	1	0	1	1	3
b4	11	1	1	1	1	4

Jak widać, wielkość pamięci mikroprogramowanego układu sterującego po optymalizacji jest znacznie mniejszy niż pierwotnie. Rozmiar pamięci przedstawionej w tabeli 2 jest równy 24 bity, natomiast rozmiar pamięci pokazanej w tabeli 4 to 16 bitów. Oznacza to, że wielkość pamięci została zredukowana aż o 33%. W tym miejscu należy dodać, iż pamięć, która została poddana procesowi optymalizacji zawiera zakodowane mikroinstrukcje mikrokontrolera. Niezbędne jest więc zdekodowanie pierwotnych mikrooperacji. Ten problem może zostać rozwiązany na wiele sposobów (na przykład poprzez zastosowanie dekodera) i nie jest przedmiotem niniejszego artykułu.

4. PODSUMOWANIE

W artykule zaproponowano metodę optymalizacji rozmiaru pamięci mikroprogramowanego układu sterującego. Minimalizacja długości mikroinstrukcji przechowywanych w pamięci mikrokontrolera jest realizowana poprzez odpowiednie zakodowanie mikrooperacji wchodzących w skład klas kompatybilności, wyznaczonych z wykorzystaniem hipergrafów.

Badania przeprowadzone przez autorów wykazały, że efektywność proponowanej metody jest ściśle uwarunkowana od stopnia kompatybilności mikrooperacji zapisanych w pamięci mikroprogramowanego układu sterującego. Formalnie można to określić następującym wzorem:

$$t = \left(1 - \frac{\sum_{i=1}^{|S|} L_i}{N} \right) \cdot 100\ \% \qquad (3)$$

gdzie t oznacza zysk (w %) zajętości pomięci w stosunku przed i po optymalizacji; |S|- liczba klas kompatybilności realizujących minimalne pokrycie; Li- waga i-tej klasy wchodzącej w skład minimalnego pokrycia; N- pierwotny rozmiar mikrooperacji.

Testy przeprowadzone na podstawie (3) wykazały, że metoda jest najbardziej przydatna w przypadku, gdy liczba mikrooperacji wchodzących wkład klas realizujących minimalne pokrycie jest jak największa, a osiągnięty zysk może wynosić nawet do 70-80%.

LITERATURA

[1] M. Adamski: *Metodologia projektowania reprogramowalnych sterowników logicznych z wykorzystaniem elementów CPLD i FPGA*, Materiały I KKN RUC, Szczecin: 1998, s. 15-22

[2] I. Ahmed, M. K. Dhodhi: *State assignment of finite state machines*, IEE Proceedings: Computers and Digital Techniques, № 1, 2000 - s. 15-22

[3] S. Baranov: *Logic Synthesis for Control Automata*, Kluwer Academic Publishers, 1994 – s. 312

[4] A. A. Barkalov, A. V. Palagin: *Synthesis of microprogram control units*, Kiev: IC NAS of Ukraine, 1997

[5] A. A. Barkalov: *Synthesis of control units on programmable logic devices*, Donetsk: DNTU, 2002 – s. 262

[6] G. Borowik, M. Rawski: *Effective Implementation of Sequential Circuits using FPGAs with Embedded Memory Blocks*, IFAC Proceedings: Programmable Devices and Systems, Kraków, 2004 – s. 175 – 180

[7] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli: *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Dordrecht, 1984

[8] S. Dasgupta: The *Organization of Microprogram Stores*, Computing Surveys, 1979

[9] G. De Micheli: *Synthesis and Optimization of Digital Circuits*, McGraw Hill: NY, 1994

[10] R. Grushnitsky, A. Mursaev, E. Ugrjumov: *Development of systems on chips with programmable logic*, SPb: BHV – Petersburg, 2002 – s. 608

[11] T. Łuba (Praca zbiorowa pod redakcją prof. Tadeusza Łuby): *Synteza układów cyfrowych*, Warszawa: WKŁ, 2003 – s. 296

[12] T. Łuba, H. Selvaraj: *A General Approach to Boolean Function Decomposition and its Applications in FPGA-based Synthesis*, VLSI Design, Special Issue on Decompositions in VLSI Design, vol. 3. Nos. 3-4, 1995 - s. 289-300

[13] R. Wisniewski: *Design of compositional microprogram control units with sharing of the codes*, Gliwice: Archiwum Konferencji PTETiS , 2004 - Vol. 19, z. 4, s. 217-220