Adam OPARA, Dariusz KANIA
POLITECHNIKA ŚLĄSKA, INSTYTUT INFORMATYKI, INSTYTUT ELEKTRONIKI

Synteza wielowyjściowych układów logicznych prowadząca do wykorzystania wspólnych bloków logicznych

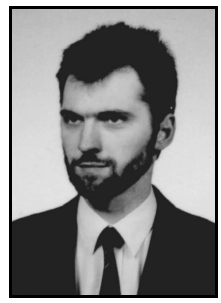
Mgr inż. Adam OPARA

Ukończył studia na Wydziale Automatyki Elektroniki i Informatyki na Politechnice Śląskiej, obronił pracę magisterską w 2002 r. Jest doktorantem w Instytucie Informatyki. Jego zainteresowania naukowe to programowalne układy cyfrowe i układy mikroprocesorowe.

e-mail: Adam.Opara@polsl.pl

Dr hab. inż. Dariusz KANIA

Ukończył studia na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej. Pracę doktorską obronił w 1995, habilitacyjną w 2004r. Jest adiunktem w Instytucie Elektroniki Politechniki Śląskiej. Jego zainteresowania naukowe koncentrują się wokół programowalnych układów i systemów cyfrowych.



e-mail: Dariusz.Kania@polsl.pl

Streszczenie

W artykule przedstawiona jest koncepcja wykorzystania wielokorzeniowych binarnych diagramów decyzyjnych (SBDD) oraz diagramów o wielu liściach (MTBDD) podczas dekompozycji funkcji logicznych. Funkcje te są poddawane dekompozycji, by można je było zaimplementować w typowych strukturach FPGA. W prezentowanym rozwiązaniu operowanie na wielu funkcjach pozwala na współdzielenie bloków związanych w dekompozycji Ashenhursta, a tym samym wymaga mniejszej ilości zasobów struktury programowalnej. Powyższa koncepcja została przedstawiona na przykładzie dekompozycji układu rd84.pla.

Słowa kluczowe: dekompozycja, wielokorzeniowe binarne diagramy decyzyjne (BDD), synteza programowalnych układów logicznych.

Multi-output logic devices synthesis utilizing common logic blocks

Abstract

This paper presents concept of using multi-root (shared) and multi-terminal binary decision diagrams (SBDD and MTBDD) to represent a set of boolean functions. These functions are decomposed to implement them in typical FPGA devices. Most of algorithms based on BDD operates on single function, so many common relations can not be extracted. In presented approach operating on many functions gains better utilization of programmable device's resources. As an example it is shown decomposition of rd84.pla circuit. With best author's knowledge there's no better results for this circuit published in literature.

Keywords: decomposition, shared binary decision diagrams (SBDD), programmable logic devices synthesis.

1. Wstęp

Obecnie układy logiki programowalnej są szeroko stosowane w projektowaniu urządzeń cyfrowych. W obliczu coraz większej złożoności projektów kluczową rolę odgrywają efektywne algorytmy i struktury danych używane w procesie syntezy.

Powszechnymi metodami reprezentacji wyrażeń logicznych są tablice prawdy oraz wyrażenia przedstawione w postaci sumy iloczynów. Tablice prawdy wymagają dla funkcji o n zmiennych użycia 2^n bitów pamięci. Ta wykładnicza zależność sprawia, że tablice prawdy są mało praktyczne dla funkcji boolowskich o wielu zmiennych. Bardziej zwięzłą reprezentacją jest suma iloczynów zwanych czasami kostkami (*cube*). Dla wielu funkcji logicznych reprezentacja w postaci zbioru kostek pozwala uniknąć wykładniczo rosnącej zajętości pamięci, ale operowanie na takich strukturach wymaga użycia czasochłonnych algorytmów redukcji, które są niezbędne by pozbyć się nadmiarowych informacji. Innym efektywnym sposobem reprezentacji funkcji logicznych jest użycie binarnych diagramów decyzyjnych (BDD). Diagramy te zostały wprowadzone przez Akersa [1] i spopularyzowane przez Bryanta [2] i Brace'a [3].

2. Typy BDD

Binarny diagram decyzyjny jest skierowanym, acyklicznym grafem (drzewem) [4]. Każdy węzeł tego grafu skojarzony jest z jedną zmienną funkcji. Z każdego węzła wychodzą dwie krawędzie skojarzone z wartościami zmiennej 0 i 1. Do każdego węzła grafu (oprócz korzenia) dochodzi, co najmniej jedna krawędź z węzłów znajdujących się na wyższym poziomie. Binarny diagram decyzyjny zawiera dwa węzły terminalne (liście), które skojarzone są z wartościami funkcji 0 lub 1. Analiza dróg występujących w binarnym diagramie decyzyjnym pozwala na określenie wartości poszczególnych zmiennych, dla których funkcja przyjmuje wartość 1 lub 0.

W praktyce wykorzystuje się jedynie uporządkowane (Ordered BDD) i zredukowane uporządkowane diagramy decyzyjne (Reduced Ordered BDD). W uporządkowanych diagramach w każdej ścieżce zmienne występują w tej samej kolejności. Na danym poziomie diagramu występują węzły związane z tą samą zmienną. Diagramy zredukowane (ROBDD), zawierające przy danym uporządkowaniu zmiennych minimalną liczbę węzłów, uzyskuje się poprzez sklejanie odpowiednich węzłów i redukcje identycznych podgrafów [2, 4, 5].

Znane są również diagramy, w których w celu zaoszczędzenia czasu obliczeń i pamięci potrzebnej do reprezentacji BDD wprowadzone zostały atrybuty przypisane do krawędzi [5]. Przykładem takiego atrybutu jest negacja. Atrybut taki oznacza, że należy zanegować funkcję, która jest reprezentowana przez podgraf wskazywany przez krawędź z atrybutem.

Zespół funkcji może być reprezentowany przez jeden wielokorzeniowy graf. Funkcje współdzielą wtedy podgrafy, stąd struktura taka nazywana jest diagramem współdzielonym (Shared BDD). Zaletą takiego rozwiązania jest zwięzłość reprezentacji zespołu funkcji oraz ułatwione testowanie równoważności dwóch funkcji.

3. Dekompozycja

Dekompozycja jest procesem podziału zmiennych funkcji. Funkcja n-zmiennych $f(X)$, $X = x_1, x_2, \ldots, x_n$, może być przedstawiona jako $F(g_1(X_b), \ldots, g_m(X_b), X_f)$, gdzie X_b to zbiór zmiennych związanych (*bound set*), a X_f to zbiór wolny (*free set*). Podział zmiennych polega na szukaniu zbiorów X_b oraz X_f. Jeśli część wspólna zbiorów X_b, X_f jest pusta mówimy, że dekompozycja jest rozłączna.

Pierwsze systematyczne podejście do dekompozycji rozłącznej podał Ashenhurst. W prezentowanej przez niego metodzie zmienne najpierw są dzielone na zbiór wolny X_f i związany X_b. Następnie funkcja przedstawiana jest w postaci dwuwymiarowej tablicy. Do opisania kolumn i wierszy tablicy służą odpowiednio zmienne ze zbioru związanego i wolnego. Dekompozycja rozłączna istnieje, jeśli liczba różnych kolumn zwana krotnością kolumn wynosi 2.

Dekompozycja Ashenhursta została udoskonalone przez Rotha i Karpa [6]. W ich metodzie funkcja reprezentowana jest w postaci kostek (*cubes*). Kostki są grupowane w klasy równoważności, przy czym klasy równoważności odpowiadają różnym kolumnom

w dekompozycji Ashenhursta. Poza dużo wydajniejszą implemantacją, obydwa sposoby dekompozycji nie różnią się w istotny sposób.

Podział zmiennych jest równoważny dokonaniu cięcia w binarnym diagramie decyzyjnym, gdzie węzły powyżej cięcia odpowiadają zbiorowi zmiennych związanych, a poniżej cięcia zbiorowi wolnemu. Węzły poniżej cięcia, które są wskazywane przez przecięte krawędzie nazywane są zbiorem węzłów odciętych. Liczba węzłów odciętych odpowiada liczbie klas równoważności w dekompozycji zaproponowanej przez Roth-Karpa [6].

W ogólnym przypadku dla n węzłów odciętych potrzeba do ich rozróżnienia $\lceil \log_2(n) \rceil$ bitów, wtedy mamy kilka funkcji g_i, i=1,…, $\lceil \log_2(n) \rceil$.

Liczba węzłów odciętych zależy od wyboru zmiennych, które tworzą zbiór wolny i związany, czyli od kolejności zmiennych w grafie i od poziomu, na którym nastąpi cięcie. Dla układów opartych o k wejściowe bloki LUT, poziom cięcia przeważnie ustala się na k licząc od korzenia diagramu. Pozostaje jednak problem odpowiedniego uporządkowania zmiennych. Aby szukając dobrego rozwiązywania nie sprawdzać wszystkich możliwości można się posłużyć poniższą zależnością.

Twierdzenie: jeśli poniżej cięcia jest n węzłów odciętych, to zmiana kolejności zmiennych tylko powyżej, albo tylko poniżej cięcia nie zmienia liczby węzłów odciętych.

Dowód: ponieważ graf odpowiada tablicy dekompozycji Ashnehursta, wprost widać, że zamiana kolejności zmiennych zbioru wolnego (lub związanego) nie zmienia krotności kolumn.

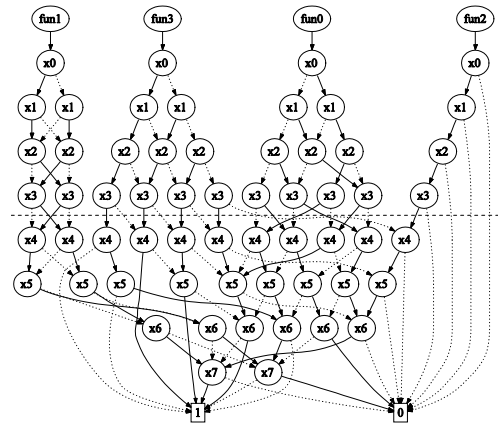
Znane są heurystyczne metody (*divide and conquer* [7]) pozwalające na znalezienie diagramu BDD z możliwie jak najmniejszą liczbą węzłów.

Jak zauważył Vemuri[8] BDD o najmniejszej liczbie węzłów nie zawsze prowadzi do „najlepszej" dekompozycji, dlatego należy szukać uporządkowania zmiennych, dla którego liczność zbioru węzłów odciętych jest najmniejsza. Podczas poszukiwań można kierować się następującą zasadą. Poniżej cięcia wybrać węzeł, który ma największą liczbę incydentnych (wskazujących na niego) krawędzi. Powyżej cięcia wybrać węzeł o najmniejszej liczbie incydentnych krawędzi, następnie zamienić kolejnością wybrane węzły. Powyższą procedurę stosować dopóki nie uzyska się mniejszej liczby węzłów odciętych lub przekroczy się określoną liczbę prób. Węzły o dużej liczbie krawędzi incydentnych przeważnie występują w wielu iloczynach, czyli funkcja silnie zależy od zmiennej przypisanej temu węzłowi, dlatego właśnie te węzły przenoszone są powyżej cięcia.

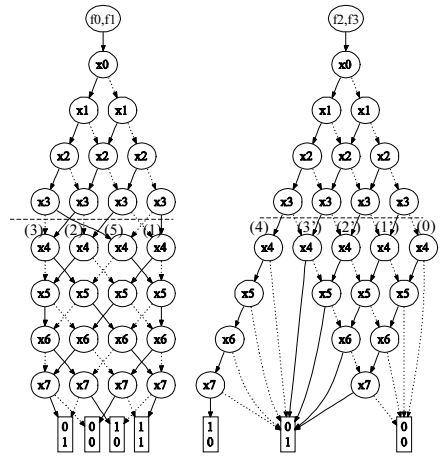
4. Dekompozycja zespołu funkcji

Podczas dekompozycji zespołu funkcji pierwszym etapem jest podział składowych wyjściowych na grupy funkcji, które korzystają z tych samych zmiennych. Kolejnym krokiem jest takie zgrupowanie funkcji by można było uzyskać wspólne funkcje g bloku związanego. Etap ten zostanie przedstawiony na przykładzie dekompozycji układu testowego rd84.pla na bloki LUT 4-we/2wy lub 5we/1wy (bloki układów Xilinx).

Na rys.1 przedstawiony jest wielokorzeniowy binarny diagram decyzyjny (SBDD) z linią cięcia diagramu poniżej węzłów odpowiadających zmiennej x_3. Powyżej linii cięcia są cztery warstwy węzłów, które odpowiadają czterem zmiennym związanym: x_0,x_1,x_2,x_3. Liczba funkcji g bloku związanego zależy od liczby węzłów docelowych poniżej linii cięcia. W celu minimalizacji liczby funkcji g zaproponowano następujący schemat postępowania. Po pierwsze należy utworzyć posortowaną listę funkcji f_0,...,f_3. Funkcje należy posortować wg liczby węzłów docelowych: f_1 – 2 węzły, f_2 – 2, f_0 – 4, f_3 – 5. Lista funkcji jest przeglądana w kolejności rosnącej liczby węzłów. Do każdej funkcji próbuje się dołączyć inną tak by utworzyć diagram MTBDD o liczbie węzłów docelowych nie większej niż przed dołączeniem. Np. do funkcji f_1 nie można dołączyć żadnej by nie zwiększyła się liczba węzłów docelowych. Natomiast do funkcji f_0 można dołączyć f_1 i w powstałym diagramie (rys. 2) poniżej linii cięcia dalej są 4 węzły docelowe, podobnie jest dla f3 i f2.



Rys. 1. Diagram SBDD dla układu rd84.pla
Fig. 1. The SBDD diagram for rd84.pla



Rys. 2. Diagramy MTBDD dla układu rd84.pla
Fig. 2. The MTBDD diagram for rd84.pla

Kolejnym etapem jest przyporządkowanie węzłom docelowym słów kodowych i realizacja funkcji g bloku związanego. Dla funkcji f_0, f_1 (4 węzły docelowe) potrzebne są dwie funkcje kodujące g_0, g_1. Przykładowe kodowanie przedstawione jest w tablicy 1. Dla funkcji f_2, f_3 mamy 5 węzłów docelowych więc potrzebne są trzy funkcje $g_{0'}$, $g_{1'}$, $g_{3'}$.

Aby móc wielokrotnie wykorzystać niektóre bloki związane należy w grafie odszukać węzły, do których prowadzą identyczne ścieżki, ale z różnych korzeni [9]. Na przykład dla funkcji f_0, f_1 do węzła oznaczonego (1) prowadzą ścieżki: $\{1000, 0100, 0010, 0001\}_{(x0,x1,x2,x3)}$. Taki sam zbiór ścieżek dla funkcji f_2, f_3 prowadzi do węzła (1'). Analizując zbiory ścieżek od korzenia do węzłów (1), (2), (3), (5) oraz (1'), (2'), (3'), (4), (0) widać, że pierwsze trzy zbiory są identyczne natomiast (0), (4) zawierają się w (5). W takiej sytuacji możliwe jest wykorzystanie wspólnych funkcji kodujących g_0, g_1, a do rozróżnienia węzłów (0), (4) wykorzystać $g_{3'}$.

Następnym krokiem jest próba wykorzystania dekompozycji nierozłącznej polegająca na sprawdzeniu, czy któraś z funkcji g nie może być postaci: $g_1(X_b)= x$, $x \in X_b$. Aby stwierdzić czy możliwa jest dekompozycja nierozłączna należy przeanalizować górną część diagramu. Np. jeśli chcemy sprawdzić dla funkcji f_2, f_3, czy możliwe jest zakodowanie węzłów (1'), (2'), (3'), (4), (0) za pomocą $g_{0'}$, $g_{1'}$, x_0: ustalamy x_0=0 i sprawdzamy czy liczba węzłów docelowych poniżej linii cięcia nie jest większa niż 4 (bo oprócz x_0 mamy jeszcze $g_{0'}$, $g_{1'}$ a dla dwubitowego kodu da się wygenerować maksymalnie 4 słowa). Podobnie sprawdzamy liczbę węzłów docelowych z ograniczeniem x_0=1. Ponieważ w przytoczonym przykładzie funkcja jest w pełni symetryczna uzyskamy identyczny wynik dla każdej ze zmiennych x_0, x_1, x_2, x_3. W tablicy 1 pokazane jest kodowanie dla $g_{0'}$, $g_{1'}$, x_3.

Tab. 1. Sposób kodowania węzłów
Tab. 1. Node coding patterns

Węzeł	**Ścieżki** (x_0,x_1,x_2,x_3)	**Kodowanie** (g_0g_1 x_3)
(1),(1')	1000, 0100, 0010, 0001	01 -
(2),(2')	1100, 0110, 1001, 1010, 0101, 0011	10 -
(3),(3')	1110, 1101, 1011, 0111	11 -
(5)	1111, 0000	00 -
(4)	1111	00 1
(0)	0000	00 0
Węzeł	**Ścieżki** (x_4,x_5,x_6,x_7)	**Kodowanie** (g_3g_4 x_7)
	1000, 0100, 0010, 0001	01 -
	1100, 0110, 1001, 1010, 0101, 0011	10 -
	1110, 1101, 1011, 0111	11 -
	1111, 0000	00 -
	1111	00 1
	0000	00 0

Tab. 2. Sposób kodowania bloków wyjść f_2f_3
Tab. 2. Coding patterns for f_2f_3 outputs

Węzeł	**Ścieżki** (g_0,g_1,x_3, g_3,g_4)	**Kodowanie** ($g_{10}g_{11}$)	**Kodowanie** ($g_{10}g_{11}$ x_7)	**Wyjścia** (f_2f_3)
(4)	001 00	00	00 1	10
			00 0	01
	01	01	01 -	01
	10	01		
	11	01		
(3')	11- 00	10	10 0	00
			10 1	01
	01	01	01 -	01
	10	01		
	11	01		
(2')	10- 01	11	11 -	00
	00	10	10 0	00
			10 1	01
	10	01	01 -	01
	11	01		
(1')	01- 11	01	01 -	01
	00	10	10 1	01
			10 0	00
	01	11	11 -	00
	10	11		
(0)	000 00	10	10 1	01
			10 1	00
	01	11	11 -	00
	10	11		
	11	11		

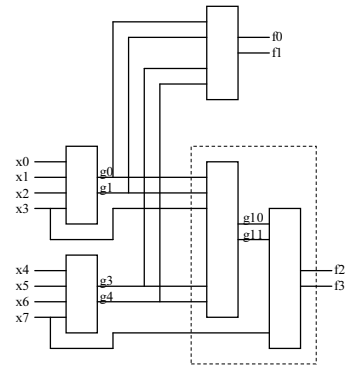
Aby sprawdzić czy możliwe jest kodowanie za pomocą g_0, x_0, x_1 należy dla każdej kombinacji x_0x_1= 00, 01, 10, 11, sprawdzić czy liczba docelowych węzłów poniżej linii cięcia jest nie większa niż 2. Dla podanego przykładu warunek ten nie jest spełniony.

Ostatecznie po pierwszym cięciu grafu otrzymujemy blok związany $x_0x_1x_2x_3$ / g_0g_1 (wejścia/wyjścia) oraz bloki wolne $g_0g_1x_4x_5x_6x_7$ /f_0f_1 i $g_0g_1x_3x_4x_5x_6x_7$ / f_2f_3. Aby móc dokonać kolejnych cięć w grafie trzeba wybrać, które zmienne mają należeć do kolejnego zbioru związanego. Jeśli wśród wybranych zmiennych będą g_0g_1 (np. $g_0g_1x_4x_5$) otrzymamy dekompozycję szeregową. Korzystniejsze jest preferowanie przy wyborze tylko zmiennych x (np. $x_4x_5x_6x_7$) a dopiero gdy nie jest to możliwe zmiennych g – można wtedy otrzymać dekompozycję wielokrotną. W drugim przypadku otrzymujemy mniejszą liczbę warstw a często też mniejszą liczbę bloków. W celu dokonania dekompozycji, w której wyniku zbiór związany tworzą $x_4x_5x_6x_7$ należy zbudować diagram z kolejnością zmiennych od korzenia $x_4x_5x_6x_7g_0g_1$ i poprowadzić linię cięcia poniżej czwartego poziomu węzłów. Wyjścia powstałego bloku związanego zostały opisane za pomocą funkcji g_3g_4.

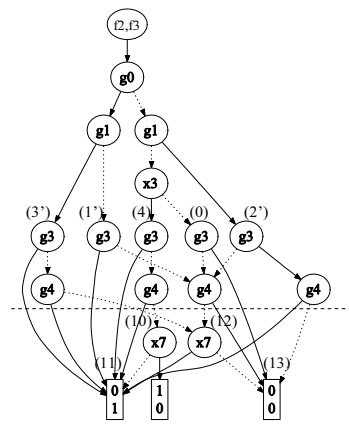
Dalsze postępowanie jest analogiczne do etapu z pierwszym cięciem grafu i zmiennymi $x_0x_1x_2x_3$. Dla funkcji f_2f_3 również korzystny jest wybór $x_4x_5x_6x_7$ jako zbiór związany, bo wtedy można wykorzystać wspólne funkcje g_3g_4. Podsumowując, po drugim cięciu otrzymujemy kolejny blok związany $x_4x_5x_6x_7$ / g_3g_4 oraz bloki wolne $g_0g_1g_3g_4$ / f_0f_1 i $g_0g_1x_3g_3g_4x_7$ / f_2f_3 (rys.3). Jak widać funkcje f_0f_1 nie wymagają dalszej dekompozycji, a diagram f_2f_3 wymaga jeszcze jednego cięcia, które pokazano na rysunku 4.

5. Podsumowanie

Artykuł przedstawia koncepcję wykorzystania wspólnych zależności dla zespołu funkcji. Jako ilustrację prezentowanej idei przedstawiono dekompozycję układu rd84.pla. Do realizacji tego układu potrzebnych było 6 bloków LUT (bloki o 4 wejściach i 2 wyjściach lub 5 wejściach i 1 wyjściu). Dla funkcji f_0, f_1 bloki tworzą strukturę dwuwarstwową, a dla f_2, f_3 trójwarstwową.



Rys. 3. Przykład dekompozycji układu rd84.pla - 4 bloki LUT 4/2 i 2 bloki LUT 5/1
Fig. 3. An example of rd84.pla circuit decomposition – four LUTs 4/2 and two LUTs 5/1



Rys. 4. Ostatnie cięcie grafu funkcji f_2f_3
Fig. 4. The last cut of f_2f_3 graph

Przedstawiony przykład przynosi obiecujące rezultaty, które jednak trzeba zweryfikować na większej liczbie układów testowych. Obecnie prowadzone są prace nad praktyczną implementacją algorytmu, co pozwoli porównać prezentowane podejście z innymi istniejącymi rozwiązaniami opartymi o BDD.

6. Literatura

[1] S. B. Akers: Functional Testing with Binary Decision Diagrams, Materiały konferencyjne: Eighth Annual Conference on Fault-Tolerant Computing, 1978, str. 75–82.
[2] R. E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, Materiały konferencyjne: IEEE Trans. on Computer, 1986, vol. 35, no. 8, str. 677–691.
[3] K. Brace, R. Rudell, R. Bryant: Efficient Implementation of a BDD Package, Materiały konferencyjne: Proc. Design Automation Conference, 1990, str. 40–45.
[4] G. De Micheli: Synthesis and Optimization of Digital Circuits. McGraw-Hill,Hightstown, NJ, 1994.
[5] S. Minato: "Binary Decision Diagrams and Applications for VLSI CAD", Kluwer Academic Publishers, Nov. 1996.
[6] J.P. Roth, R.M. Karp: Minimization Over Boolean Graphs, Materiały konferencyjne: IBM J. Res. Dev., 1962, str. 227-238.
[7] F.-M.YEH, S.-Y.KUO: Variable ordering for ordered binary decision diagrams by a divide-and-conquer approach. Materiały konferencyjne: IEE Proc.-Comput. Digit. Tech., 1997, vol. 144, no. 5, str. 261-266.
[8] N. Vemuri, P. Kalla, R. Tessier: BDD-based Logic Synthesis for LUT-based FPGAs, Materiały konferencyjne: Transactions on Design Automation of Electronic Systems (TODAES) 2000 ACM, 2002, vol. 7, str. 501-525.
[9] A. Opara: Wykorzystanie binarnych diagramów decyzyjnych do syntezy wielopoziomowych układów logicznych. Materiały konferencyjne: Reprogramowalne Układy Cyfrowe, Pomiary, Automatyka, Kontrola, Szczecin 2006, str.115-117.

Artykuł recenzowany