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

Wykorzystanie pseudo-MTBDD w dekompozycji zespołu funkcji

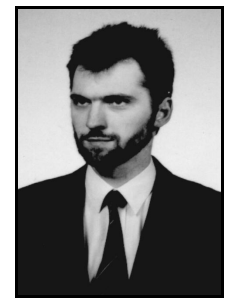
Mgr inż. Adam OPARA

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 2004r. Jest profesorem w Instytucie Elektroniki Politechniki Śląskiej. Jego zainteresowania naukowe koncentrują się wokół programowalnych układów i systemów cyfrowych.



e-mail: dariusz.kania@polsl.pl

Streszczenie

W obliczu coraz większej złożoności projektów kluczową rolę odgrywają efektywne algorytmy i struktury danych używane w procesie syntezy. W artykule przedstawiona jest koncepcja reprezentacji liści diagramów o wielu liściach (MTBDD) za pomocą diagramów BDD z wprowadzonymi dodatkowymi zmiennymi. Zabieg taki upraszcza algorytmy dekompozycji prowadzone dla odpowiednich zespołów funkcji.

Słowa kluczowe: binarne diagramy decyzyjne (BDD), wielokorzeniowe BDD, synteza logiczna, dekompozycja.

Decomposition of multi-output function based on pseudo-MTBDD

Abstract

This paper presents concept of representing multi-terminal binary decision diagram (MTBDD) by BDD diagrams with added special variables. MTBDD represents a set of boolean functions and is decomposed to implement them in typical FPGA devices. Common function relation can be extracted by merging a few single functions into a group represented by MTBDD diagram. There is special approach taken to efficiently perform merging process.

Keywords: binary decision diagrams (BDD), multi-terminal BDD (MTBDD), logic synthesis, decomposition.

1. Wstęp

Kluczowy wpływ na efektywność procesu syntezy ma sposób reprezentacji funkcji logicznych. Tablice prawdy wymagają dla funkcji o n zmiennych użycia 2^n komórek pamięci. Ta wykładnicza zależność sprawia, że ich znaczenie praktyczne jest bardzo niewielkie. Bardziej zwięzłą reprezentacją funkcji jest postać sumy iloczynów zwana czasami zestawem kostek (*cube*). Reprezentacja taka pozwala zapobiec skutkom wykładniczej zależności opisującej zajętość pamięci, jednak wymaga użycia czasochłonnych algorytmów redukcji. Znacznie efektywniej można opisywać funkcje wykorzystując binarne diagramy decyzyjne (BDD). Diagramy te zostały wprowadzone przez Akersa [1] a spopularyzowane przez Bryanta [2] i Brace'a [3].

2. Podstawy teoretyczne

Binarny diagram decyzyjny jest skierowanym, acyklicznym grafem (drzewem) [4]. Węzły grafu skojarzone są z argumentami funkcji. Z każdego węzła wychodzą dwie krawędzie skojarzone z wartościami 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 redukcję identycznych podgrafów [2, 4, 5].

Istnieją również odmiany diagramów pozwalające efektywnie pod względem liczby komórek pamięci przedstawiać zespół funkcji. Może być on reprezentowany przez jeden wielokorzeniowy diagram. Funkcje współdzielą wtedy poddiagramy, stąd struktura taka nazywana jest diagramem współdzielonym (Shared BDD). Zaletą takiego rozwiązania jest zwięzłość reprezentacji oraz ułatwione testowanie równoważności dwóch funkcji. Innym sposobem reprezentacji zespołu funkcji są diagramy o wielu liściach (Multi-Terminal BDD). W diagramach tych występuje jeden korzeń wspólny dla wszystkich m-funkcji, natomiast każdy liść zawiera m-elementowy wektor opisujący wartości poszczególnych funkcji.

Głównym problemem syntezy, oprócz sposobu reprezentacji, jest sposób dekompozycji funkcji pozwalający na dopasowanie projektowanego układu do struktury układu programowalnego. Istota dekompozycji sprowadza się do podziału układu na bloki o zadanej liczbie wejść. Funkcja n-zmiennych $f(X)$, $X = x_1,x_2,...,x_n$, może być przedstawiona jako $h(g_1(X_b),..., g_m(X_b), X_f)$, gdzie X_b to zbiór związany (*bound set*), a X_f to zbiór wolny (*free set*) wtedy, gdy liczba wzorców kolumn matrycy podziału jest mniejsza lub równa 2^m. 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.

Podział zmiennych jest równoważny dokonaniu cięcia w binarnym diagramie decyzyjnym. Węzły powyżej linii cięcia tworzą zbiór związany, a poniżej - zbiór wolny. Węzły poniżej cięcia, które są wskazywane przez przecięte krawędzie nazywane są węzłami odciętymi. Liczba węzłów odciętych odpowiada liczbie wzorców kolumn w dekompozycji Ashenhursta-Curtisa lub liczbie klas równoważności w dekompozycji Roth-Karpa [6]. W ogólnym przypadku dla n węzłów odciętych potrzeba do ich rozróżnienia $\lceil \log_2(n) \rceil$ bitów. Powstaje wtedy 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. Szukając dekompozycji pozwalającej na realizację układu za pomocą k-wejściowych bloków LUT, poziom cięcia przeważnie ustala się na k licząc od korzenia diagramu. Pozostaje jednak problem odpowiedniego uporządkowania zmiennych.

Okazuje się, że zmiana kolejności zmiennych tylko powyżej lub tylko poniżej linii cięcia nie zmienia liczby węzłów odciętych. Powyższe spostrzeżenie wynika bezpośrednio z modelu dekompozycji Curtisa, ponieważ zmiana kolejności zmiennych opisujących wiersze lub kolumny matrycy podziału nie zmienia krotności

kolumnowej tej matrycy. Pozwala to na znaczące ograniczenie obszaru poszukiwań odpowiedniego cięcia diagramu BDD.

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. Poszukiwania można prowadzić zgodnie z następującą zasadą: poniżej cięcia należy wybrać węzeł, który ma największą liczbę incydentnych (wskazujących na niego) krawędzi, a powyżej - węzeł z najmniejszą liczbą incydentnych krawędzi. Wybrane zmienne należy zamienić kolejnością. Powyższą procedurę należy stosować dopóki nie uzyska się mniejszej liczby węzłów odciętych lub nie przekroczy określonej liczby prób. Węzły o dużej liczbie krawędzi incydentnych wskazują zmienne, które występują w wielu iloczynach. Funkcja silnie zależy od takiej zmiennej, stąd są one przenoszone powyżej linii cięcia.

3. Strategia dekompozycji zespołu funkcji

Diagramy decyzyjne MTBDD można wykorzystać w procesie dekompozycji zespołu funkcji. Podczas dekompozycji zespołu funkcji pierwszym etapem powinien być podział zespołu na grupy, które korzystają z tych samych zmiennych. W kolejnym kroku należy poszukiwać grup funkcji, dla których można uzyskać wspólne funkcje g w bloku związanym. Łączenie zbyt wielu funkcji w jeden zespół może prowadzić do nadmiernej ekspansji liczby funkcji g_i a w konsekwencji do gorszych rezultatów niż dla dekompozycji każdej funkcji oddzielnie. Proces doboru funkcji do jednej grupy wymaga odpowiedniej strategii [9].

Liczba funkcji g_i zależy od liczby węzłów odciętych poniżej linii cięcia. W celu minimalizacji liczby funkcji g_i należy postępować według następującego algorytmu. Po pierwsze należy utworzyć posortowaną listę funkcji $f_1,...,f_m$. Funkcje należy posortować wg liczby węzłów odciętych. Lista funkcji powinna być przeglądana w kolejności rosnącej liczby węzłów. Każdą funkcję próbuje się połączyć z inną, tak by utworzyć diagram MTBDD o liczbie węzłów odciętych nie większej niż przed połączeniem. Przykład takiej strategii zamieszczony jest w [9]. Kolejnym etapem jest przyporządkowanie węzłom odciętym słów kodowych i realizacja funkcji g_i w bloku związanym. Dekompozycja zespołu funkcji o znacznej liczności jest mało efektywna w przypadku równoczesnego prowadzenia jej dla całego zespołu. Lepiej jest dobrać tylko kilka funkcji silnie ze sobą powiązanych i prowadzić dekompozycję dla takiej grupy [10]. Liczba odciętych węzłów w diagramie MTBDD zależy nie tylko od wyboru zmiennych, poziomu cięcia, ale również od doboru funkcji. Dobór funkcji, które tworzą jedną grupę wymaga efektywnego tworzenie diagramów MTBDD z diagramów ROBDD i rozszczepianie diagramu MTBDD na podgrupy. Liście diagramu MTBDD mogą być reprezentowane za pomocą poddiagramów ROBDD z dodatkowo wprowadzonymi zmiennymi. Nazwijmy diagramy MTBDD ze zmodyfikowanymi liśćmi diagramami pseudo-MTBDD (PMTBDD). Podejście takie pozwala na łączenie diagramów ROBDD w diagramy PMTBDD za pomocą standardowej procedury bdd_or(). Wymaga ono wprowadzenia w diagramie dodatkowych węzłów skojarzonych z wartościami każdej funkcji zespołu. Podobne pomysły związane z wprowadzaniem dodatkowych węzłów w celu uzyskania określonych właściwości diagramu można znaleźć w [11].

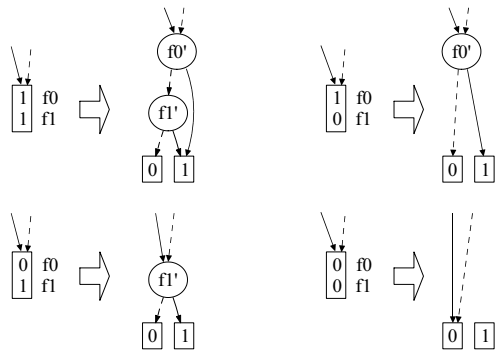
Każdy liść diagramu MTBDD zespołu m funkcji f_i: $\boldsymbol{B}^n \rightarrow \boldsymbol{B}$ gdzie i =0,...,m-1, odpowiada wektorowi wartości funkcji, który jest reprezentowany w PMTBDD poddiagramem opisującym wyrażenie

$$\sum_{i=0}^{m-1} f_i \cdot f_i' , \tag{1}$$

gdzie f_i' reprezentuje dodatkowo wprowadzone zmienne.

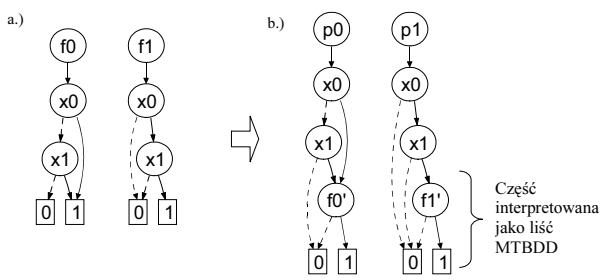
Na rysunku 1 przedstawiona została reprezentacja liści MTBDD dla zespołu dwóch funkcji $f_0 f_1$. Do diagramu wprowadzone zostały dwie dodatkowe zmienne f_0', f_1'. Poszczególne wartości liści są reprezentowane poddiagramami następujących wyrażeń:

$$11_{f_0 f_1} \rightarrow f_0' + f_1', \quad 10_{f_0 f_1} \rightarrow f_0', \quad 01_{f_0 f_1} \rightarrow f_1', \quad 00_{f_0 f_1} \rightarrow 0 \tag{2}$$



Rys. 1. Reprezentacja liścia MTBDD za pomocą poddiagramu ROBDD z dodatkowymi zmiennymi
Fig. 1. MTBDD leaf representation by ROBDD subdiagram with additional variables

Proces łączenia diagramów ROBDD w jeden diagram pseudo-MTBDD zostanie przedstawiony na przykładzie zespołu funkcji $f_0 = x_0 + x_1$, $f_1 = x_0 \cdot x_1$. Mając dane diagramy ROBDD dla f_0 i f_1 (rys. 2a) należy wprowadzić dodatkowe zmienne f_0', f_1' i pomnożyć odpowiednią dodatkową zmienną przez funkcję. Po pomnożeniu powstaną wyrażenia $p_0 = f_0' \cdot (x_0 + x_1)$, $p_1 = f_1' \cdot (x_0 \cdot x_1)$, których diagramy przedstawia rys. 2b. Diagramy dla tych wyrażeń mogą być już interpretowane jako diagramy PMTBDD dla dwóch zespołów funkcji, przy czym w każdym z zespołów jest tylko jedna funkcja. Wszystkie węzły poniżej węzłów związanych z x_1 interpretowane są jako liście MTBDD. Ponieważ liczność zespołu wynosi 1 poniżej poziomu x_1 może występować co najwyżej jeden poziom węzłów związanych z dodatkowo wprowadzanymi zmiennymi f_i'.



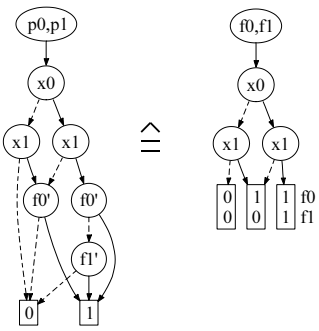
Rys. 2. Proces przekształcania diagramu ROBDD w PMTBDD
Fig. 2. A process of transformation ROBDD into PMTBDD

Diagram PMTBDD dla zespołu dwóch funkcji f_0, f_1 można uzyskać przez wykonanie operacji sumy (bdd_or()) na wcześniej utworzonych diagramach p_0, p_1. Po zsumowaniu otrzymujemy diagram dla następującego wyrażenia:

$$\begin{aligned} p_0 + p_1 &= f_0' \cdot f_0 + f_1' \cdot f_1 \\ &= f_0' \cdot (x_0 + x_1) + f_1' \cdot x_0 \cdot x_1 \\ &= (f_0' + f_1') \cdot x_0 \cdot x_1 + (f_0') \cdot (x_0 \cdot \overline{x_1} + \overline{x_0} \cdot x_1) \end{aligned} \tag{3}$$

Uzyskany diagram PMTBDD wraz z odpowiadającym mu diagramem MTBDD przedstawiono na rys. 3.

Operacją odwrotną do scalania jest rozszczepianie. Aby z diagramu dla zespołu p_0, p_1 utworzyć diagram p_0 wystarczy przypisać zmiennej f_0'=0. Można do tego wykorzystać standardową procedurę bdd_compose() (lub bdd_restrict()) umożliwiającą podstawienie wyrażenia (w tym wypadku stałej) do zmiennej. W ten sposób można uzyskać diagram p_0 z rys. 2b. W analogiczny sposób można usunąć dowolną funkcję z diagramu zespołu funkcji.



Rys. 3. Równoważne postacie PMTBDD i MTBDD
Fig. 3. An equivalent representation of diagrams PMTBDD and MTBDD

Do zalet diagramów PMTBDD można zaliczyć:

– brak konieczności tworzenia oddzielnej tablicy unikalnych identyfikatorów dla liści (w przypadku MTBDD przy tworzeniu nowego liścia trzeba zadbać o jego unikalność by zachować kanoniczność reprezentacji),

– możliwość scalania diagramów za pomocą standardowej operacji sumy wykonywanej na ROBDD (najczęściej wszystkie operacje dwuargumentowe są implementowane przy wykorzystaniu operatora ITE),

– możliwość rozszczepiania przy wykorzystaniu standardowych procedur operujących na ROBDD.

Łączenie diagramów można wykorzystać w procesie dekompozycji zespołu funkcji. Na rys. 5 przedstawiony jest sposób łączenia diagramów PMTBDD dla zespołu funkcji z rys. 4. Dla funkcji p_0 i p_1 liczba węzłów odciętych wynosi 3. Po połączeniu diagramów liczba węzłów odciętych nie zwiększyła się. Taka sytuacja umożliwia wyszukiwanie w procesie dekompozycji wspólnych bloków związanych.

x_0x_1 \ $x_2x_3x_4x_5$	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
00	10	01	10	01	10	00	10	01	10	01	10	01	01	01	10	00
01	11	10	11	10	11	00	11	10	11	10	11	10	10	10	11	00
10	01	10	01	10	01	00	01	10	01	10	01	10	10	10	01	00
11	00	01	00	01	00	00	00	01	00	01	00	01	01	01	00	00
	a	b	a	b	a	c	a	B	a	b	a	b	b	b	a	c

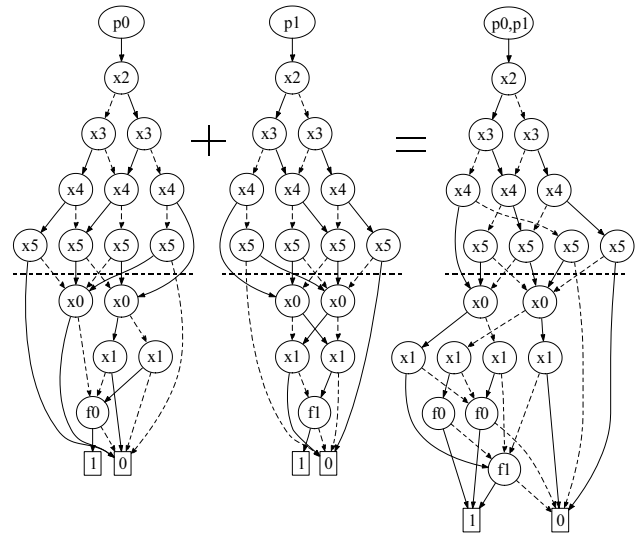
$f_0 f_1$

Rys. 4. Tablica dekompozycji zespołu funkcji
Fig. 4. A decomposition table for group of functions

4. Podsumowanie

Artykuł przedstawia koncepcję sposobu reprezentacji liści diagramów MTBDD umożliwiającej za pomocą standardowej procedury obliczania sumy (bdd_or()) scalanie diagramów w diagramy opisujące zespoły funkcji. Dzięki takiej reprezentacji można również dokonywać rozszczepiania diagramów za pomocą standardowych procedur.

Obecnie prowadzone są prace nad praktyczną implementacją algorytmu wykorzystującego przedstawione idee, co pozwoli porównać prezentowane podejście z innymi istniejącymi rozwiązaniami wykorzystującymi diagramy BDD.



Rys. 5. Diagram PMTBDD dla zespołu funkcji z rys. 4
Fig. 5. PMTBDD diagram for functions from fig. 4

Pracę wykonano w ramach projektu promotorskiego N N515 1805 33 finansowanego przez MNiSW.

5. Literatura

[1] S. B. Akers: Functional Testing with Binary Decision Diagrams, Eighth Annual Conference on Fault-Tolerant Computing, 1978, pp. 75–82.

[2] R. E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. on Computer, 1986, vol. 35, no. 8, pp. 677–691.

[3] K. Brace, R. Rudell, R. Bryant: Efficient Implementation of a BDD Package, Proc. Design Automation Conference, 1990, p. 40–45.

[4] G. De Micheli: Synthesis and Optimization of Digital Circuits. McGraw-Hill, 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, IBM J. Res. Dev., 1962, pp. 227-238.

[7] F.-M.Yeh, S.-Y.Kuo: Variable ordering for ordered binary decision diagrams by a divide-and-conquer approach. IEE Proc.-Comput. Digit. Tech., 1997, vol. 144, no. 5, pp. 261-266.

[8] N. Vemuri, P. Kalla, R. Tessier: BDD-based Logic Synthesis for LUT-based FPGAs, Transactions on Design Automation of Electronic Systems (TODAES) 2000 ACM, 2002, vol. 7, pp. 501-525.

[9] A. Opara, D. Kania: Synteza wielowyjściowych układów logicznych prowadząca do wykorzystania wspólnych bloków logicznych, Pomiary Automatyka Kontrola, Szczecin 2007, ss. 39-42.

[10] M.-T. Lai, K.-R. R. Pan, M. Pedram: OBDD-Based Function Decompolsition: Algorithms and Implementation, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No. 8, 1996, pp. 977-990.

[11] S. Minato: Binary Decision Diagrams and Applications for VLSI CAD, Kluwer Academic Publishers, 1996.

Artykuł recenzowany