Adam MILIK, Dariusz KANIA
POLITECHNIKA ŚLĄSKA, INSTYTUT ELEKTRONIKI

Zastosowanie diagramów BDD w syntezie logicznej dla układów typu PAL

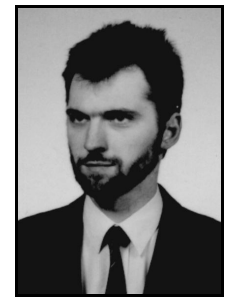
Dr inż. Adam MILIK

Ukończył studia na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej. Pracę doktorską obronił w 2003 r. Jest adiunktem w Instytucie Elektroniki Politechniki Śląskiej. Jego zainteresowania naukowe to układy logiki programowalnej, sterowniki programowalne, modelowanie i synteza złożonych układów sprzętowo-programowych.

e-mail: adam.milik@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 przedstawiono zastosowanie diagramów BDD w procesie syntezy dla układów typu PAL. Diagramy BDD wykorzystywane są w procesie dekompozycji funkcji w celu szybkiego wyszukania możliwych do implementacji w pojedynczej komórce PAL podukładów.

Słowa kluczowe: BDD, PAL, PLD, synteza logiczna, dekompozycja.

Application of BDD in Logic Synthesis for PAL-based Devices

Abstract

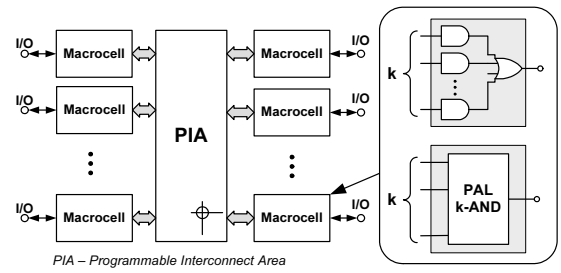
The paper presents the BDD based method of function decomposition for PAL-based devices. A BDD diagram is successfully used for function mapping for LUT based FPGAs [3]. In opposite to LUT-based circuits PAL-based devices are limited in number of products while number of inputs to the block is large (Fig. 1). Before decomposition procedure can be applied, function variables are ordered. Decomposition procedure searches BDD tree for suitable decomposition starting from variables with the largest index (just above terminals 0 and 1). When satisfying function is found its subtree is substituted by node that belong to newly created variable (Fig. 3 a,b,c,d). Procedure is applied iteratively until root node is reached. Decomposition procedure efficiency is proofed with use of ISCAS LG89 benchmarks. Obtained implementation results are compared to classical approach in Tab. 1.

Keywords: BDD, PAL, PLD, logic synthesis, decomposition.

1. Wprowadzenie

Proces implementacji jest nierozerwalnym elementem procesu projektowego z użyciem układów programowalnych [4, 5]. Obok dominujących układów typu FPGA znaczącą grupę układów programowalnych stanowią układy matrycowe CPLD typu PAL (rys. 1). Klasyczna metoda syntezy układów realizowanych w strukturach matrycowych typu PAL obejmuje dwupoziomową minimalizację wykonywaną dla każdej funkcji oddzielnie, po której następuje etap implementacji funkcji w k-iloczynowych blokach logicznych typu PAL. W przypadku realizacji zminimalizowanych funkcji, będących sumą p implikantów, zachodzi potrzeba wykorzystywania większej liczby bloków poprzez wprowadzanie sprzężeń zwrotnych.

Efektywną reprezentację funkcji boolowskich można uzyskać wykorzystując acykliczne grafy skierowane, zwane binarnymi diagramami decyzyjnymi. Pod pojęciem diagramu BDD aktualnie rozumie się wiele odmian tychże diagramów. Niezwykle istotnym w procesie projektowania układów cyfrowych jest zredukowany uporządkowany diagram binarny (ang. ROBDD) [1, 2, 6]. Diagramy ROBDD pozwalają na efektywną reprezentację funkcji boolowskich a także na łatwe prowadzenie procesu dekompozycji poprzez gromadzenie zmiennych lub rozcinanie fragmentów diagramu [3].



Rys. 1. Układy o architekturze typu PAL
Fig. 1. Architecture of a PAL-based device

W celu zbadania własności diagramów BDD pod kątem wykorzystania w procesie dekompozycji dla układów typu PAL, zostaną rozpatrzone zagadnienia związane z porządkowaniem zmiennych oraz dekompozycją iteracyjną grupującą zmienne.

Przedstawione zagadnienia pozwolą dokonać oceny przydatności diagramów decyzyjnych w procesie dekompozycji. Niezwykle istotnym elementem BDD jest stosunkowo niewielka złożoność obliczeniowa algorytmów, co stwarza możliwość implementacji funkcji o znacznej liczbie zmiennych. Do programu, dane wprowadzane są w postaci pliku PLA co odpowiada opisowi funkcji w postaci sumy iloczynów. Przyjmuje się także, że wprowadzone wektory wejściowe odpowiadają iloczynom o minimalnej liczbie literałów.

2. Porządkowanie zmiennych

Proces porządkowania (uszeregowania) zmiennych ma kluczowe znaczenie dla złożoności oraz operacji na zredukowanym diagramie decyzyjnym. Właściwy dobór kolejności zmiennych, pozwoli na szybkie i łatwe uzyskanie wyników w procesie dekompozycji. Ze względu na format danych wejściowych, porządkowanie zmiennych odbywa się na zminimalizowanej funkcji logicznej wyrażonej jako suma iloczynów. Dla funkcji przedstawionej w postaci kanonicznej, trudno wyznaczyć stopień oddziaływania danej zmiennej na wartość funkcji logicznej. Ogólnie znane są dwie heurystyczne zasady odnoszące się do porządkowania zmiennych [6]:

1. Zmienne mające największy wpływ na wartość funkcji logicznej należy umieszczać możliwie najbliżej wierzchołka diagramu
2. Zmienne, które lokalnie są związane z sobą w funkcji, należy umieszczać w bliskim sąsiedztwie.

Obok wymienionych zasad należy dodać kolejną badającą dominację zmiennych. Jeżeli dwie zmienne posiadają statystycznie identyczny wpływ na wartość funkcji, to zmienna występująca tylko w postaci prostej bądź zanegowanej powinna zostać umieszczona wcześniej. Przedstawione obserwacje stanowią podstawy algorytmu szeregowania zmiennych. Należy zwrócić uwagę, że

wyznaczanie uszeregowania zmiennych na drodze statystycznej może prowadzić do uzyskania różnych wyników dla funkcji posiadających wiele możliwych pokryć minimalnych.

3. Algorytm dekompozycji wykorzystujący grupowanie zmiennych

Diagramy BDD bardzo chętnie są stosowane do dekompozycji funkcji w procesie implementacji wykorzystującym blok tablicowy (ang. LUT – Lookup Table). Kryterium podziału funkcji w przypadku układów tablicowych, opiera się na liczbie wejść dostępnych w pojedynczym bloku. Determinuje ona maksymalną liczbę zmiennych jaką może posiadać funkcja implementowana w pojedynczym generatorze tablicowym. Liczbę argumentów funkcji opisanych diagramem BDD można bardzo łatwo określić poprzez wyznaczenie zbioru argumentów, w oparciu o które jest zbudowane drzewo (cecha ROBDD).

Układy typu PAL nakładają ograniczenie na liczbę iloczynów dostępnych w pojedynczym bloku. Zaproponowany algorytm wykorzystujący diagramy BDD przeznaczony jest do dekompozycji z uwzględnieniem najlepszego dopasowania funkcji realizowanych w bloku związanym typu PAL. Określenie liczby iloczynów koniecznych do implementacji danej funkcji wymaga, wyznaczenia zbioru implikantów o minimalnej mocy, na podstawie danego diagramu binarnego.

Podstawowym kryterium próby dekompozycji funkcji, jest minimalna liczba implikantów d_f konieczna do jej realizacji (przed dekompozycją). Znając d_f można wyznaczyć liczbę bloków B_{PAL} koniecznych do implementacji funkcji. Przyjmując, że liczba iloczynów we wszystkich blokach układu jest identyczna i równa k, liczba bloków wynosi:

$$B_{PAL} = \left\lceil \frac{d_f - k}{k-1} \right\rceil + 1 \tag{1}$$

Efektem dekompozycji funkcji jest podział na dwa podukłady (związany i wolny) a zatem można uznać że prowadzenie procesu dekompozycji w przypadku układów PAL jest celowe wtedy gdy:

$$d_f \geq 2k \tag{2}$$

Jeżeli powyższy warunek nie jest spełniony, stosuje się realizację klasyczną funkcji.

3.1. Iteracyjny algorytm dekompozycji

Algorytm dekompozycji zostanie przedstawiony na przykładzie. Niech będzie dana funkcja opisana za pomocą siatki Karnaugha (rys. 2).

	cde							
ab	000	001	011	010	110	111	101	100
00	1	0	1	1	1	0	1	0
01	1	0	1	0	1	1	1	0
11	0	1	0	0	0	1	0	1
10	1	0	1	1	1	0	1	0

Rys. 2. Przykładowa funkcja
Fig. 2. Exemplary function

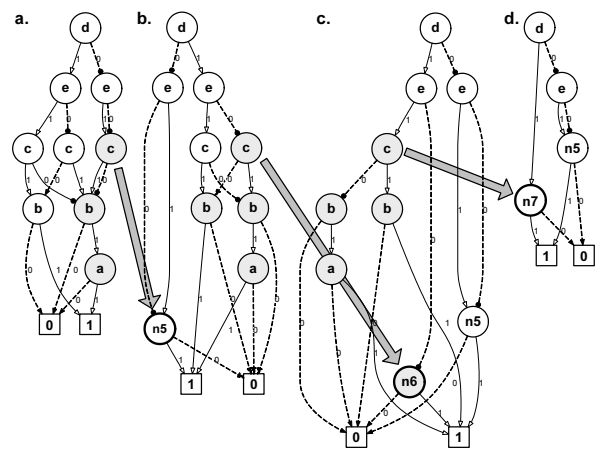
Funkcja przed utworzeniem diagramu BDD zostaje poddana minimalizacji a następnie procedurze rankingowej [8] w celu określenia kolejności zmiennych :

$$V_{ORD} = \{d,\ e,\ c,\ b,\ a\} \tag{3}$$

W następnym etapie postępowania, zostaje zbudowane zredukowane drzewo binarne (rys. 3a). Niech będzie dany układ o blokach PAL zawierających po trzy iloczyny (k=3). Proces dekompozycji polega na wyszukiwaniu najlepiej dopasowanej do zadanego bloku PAL funkcji związanej. Podstawowe kryterium implementacyjności w pojedynczym bloku typu PAL wybranej funkcji, jest zdeterminowane przez liczbę bramek AND w nim dostępnych. Procedura dekompozycji opiera się na takim wyborze funkcji związanej, przez dobieranie kolejnych węzłów drzewa, aby minimalna liczba iloczynów konieczna do jej implementacji nie przekraczała liczby iloczynów zawartych w pojedynczym bloku. Ze względu, że liczba wejść bloku PAL jest znaczna, w algorytmie nie nałożono ograniczeń na liczbę argumentów funkcji związanej. Ze względu na wykorzystanie diagramów BDD niezwykle łatwo uzupełnić procedury podziału o kryterium nakładające ograniczenie na liczbę wejść dekomponowanego bloku.

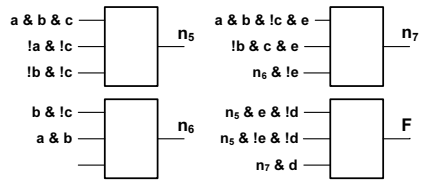
Poszukiwanie funkcji związanej rozpoczyna się od węzłów z najwyższym indeksem (położonych najbliżej liści 0 i 1). Procedura pobiera kolejne węzły drzewa binarnego uszeregowane w kolejności malejących indeksów. Dla każdego z pobranych węzłów należy wyznaczyć minimalny zbiór implikantów, pokrywający funkcję określoną przez drzewo, którego korzeń wskazuje na wybrany węzeł. Liczba implikantów musi zostać wyznaczona dla postaci prostej i zanegowanej funkcji. Mniejsza liczba, spośród liczby implikantów minimalnych dla postaci prostej i zanegowanej jest wybierana jako liczba iloczynów potrzebnych do implementacji funkcji związanej. Obok liczby implikantów określana jest liczba węzłów drzewa odpowiadającego funkcji związanej. Pozwoli to wybrać funkcję związaną opisaną diagramem o największej liczbie węzłów. Jako bieżącą funkcję związaną wybiera się tą, która posiada większą liczbę implikantów. W przypadku identycznej liczby implikantów, wybiera się funkcję o większej liczbie węzłów. Proces poszukiwania funkcji związanej jest kontynuowany do mementu znalezienia funkcji, wymagającej do implementacji większej liczby implikantów aniżeli liczba iloczynów w bloku PAL. Oznacza to, że należy dokonać zastąpienia poddrzewa funkcji związanej przez nowy węzeł. Wiąże się to z koniecznością utworzenia nowej zmiennej dla drzewa BDD. Zmienna ta będzie reprezentowała węzeł wewnętrzny przyjmujący wynik funkcji związanej.

Na kolejnych rysunkach, kolorem szarym oznaczono węzły drzewa reprezentujące wybraną funkcję związaną (rys. 3a, b, c) a strzałka wskazuje na węzeł należący do nowo utworzonej zmiennej oraz jej podstawienie w drzewie binarnym (rys. 3b, c, d).



Rys. 3. Proces dekompozycji prowadzony na diagramie BDD
Fig. 3. Decomposition procedure steps

Po podstawieniu funkcji związanej węzłem reprezentującym wyjście bloku związanego, rozpoczyna się kolejną iterację algorytmu. Proces poszukiwania kolejnych funkcji związanych prowadzony jest do momentu osiągnięcia korzenia drzewa, co jest równoznaczne ze znalezieniem implementacji funkcji w za pomocą bloków PAL o określonej liczbie iloczynów. Ostatecznie reprezentację sprzętową układu po dekompozycji pokazano na rysunku (rys. 4).



Rys. 4. Struktura sprzętowa układu po dekompozycji
Fig. 4. Hardware structure after decomposition

3.2. Uwzględnienie funkcji prostych i zanegowanych

Do tej pory przedstawiono ogólną metodę postępowania w czasie dekompozycji funkcji z wykorzystaniem diagramu BDD. Omówiona metoda, w przypadku zastosowania do klasycznego diagramu BDD, podczas dokonywania podziałów nie jest zdolna do wykrycia funkcji związanych, występujących w postaci prostej i zanegowanej. Może to w niektórych przypadkach prowadzić do niekorzystnej replikacji funkcji związanych lub też nadmiarowego podziału, będącego wynikiem nie odnalezienia funkcji związanej w postaci prostej i zanegowanej. W celu poprawienia algorytmu, należy dokonać dodatkowego sprawdzenia czy w zbiorze funkcji związanych nie występuje identyczna funkcja w postaci zanegowanej. Zaproponowane rozszerzenie reprezentacji diagramu ROBDD o krawędzie posiadające atrybut negacji [6], obok wzrostu efektywności reprezentacji funkcji przez ograniczenie liczby węzłów koniecznych do zapisania funkcji ma istotne znaczenie w procesie dekompozycji. W efekcie, diagram zbudowany w oparciu o idee krawędzi z atrybutem negacji pozwala na usunięcie nadmiarowego reprezentowania funkcji w postaci prostej i zanegowanej. Na przedstawionym przykładzie pokazano drzewo wykorzystujące krawędzie z atrybutem negacji (krawędzie zanegowane zostały zakończone kółkiem). W pierwszym kroku dekompozycji (rys. 3a, rys. 3b) nastąpiło podstawienie funkcji w postaci prostej i zanegowanej poprzez pojedynczy węzeł n5. Dzięki zastosowaniu diagramu z atrybutem negacji przypisanym do krawędzi diagramu można uprościć algorytmu dekompozycji oraz zagwarantować znacząco lepsze rezultaty implementacji.

4. Wyniki eksperymentalne

Przedstawiony algorytm dekompozycji został zaimplementowany za pomocą języka C++. Do operacji na drzewach binarnych wykorzystano pakiet CUDD [7]. Funkcje pakietu umożliwiają dokonanie podstawowych operacji na drzewach binarnych. Na potrzeby algorytmu zostały dodane nietypowe operacje na diagramach BDD konieczne do implementacji. Do oceny jakościowej wykorzystano kilka funkcji z zestawu testów ISCAS LG89. Wyniki przeprowadzonych testów zebrano w (tab. 1). Badania prowadzono dla k od 3 do 8. Dla porównania zestawiono liczbę bloków koniecznych do implementacji za pomocą algorytmu dekomponującego funkcję, oznaczonego jako kolumna BDD oraz metody klasycznej (Kls).

Tab. 1. Rezultaty eksperymentów dla zaproponowanej metody dekompozycji
Tab. 1. Experimental results for proposed decomposition metod

k	3		4		5	
	BDD	Kls	BDD	Kls	BDD	Kls
RD53	10	15	7	11	6	8
RD73	20	70	13	47	12	36
RD84	26	142	18	95	17	72
9SYM	12	43	9	29	9	22
F51M	23	37	19	27	16	22
SAO2	35	36	20	24	17	19
XOR5	4	8	2	5	2	4
Z4M1	18	29	15	19	10	15

k	6		7		8	
	BDD	Kls	BDD	Kls	BDD	Kls
RD53	5	6	5	6	5	6
RD73	13	29	13	24	11	20
RD84	16	49	15	49	15	42
9SYM	8	18	7	15	7	13
F51M	14	18	14	16	13	15
SAO2	14	15	11	14	10	11
XOR5	2	3	2	3	2	3
Z4M1	10	13	8	11	7	7

Należy zwrócić uwagę że najlepsze rezultaty w procesie dekompozycji uzyskuje się dla funkcji o charakterze symetrycznym typu RD53, RD73, RD84 czy 9SYM.

5. Podsumowanie

Diagramy BDD stanowią podstawę szeregu algorytmów dekompozycji funkcji. Zwykle są one wykorzystywane w procesie implementacji funkcji logicznej realizowanych w układach FPGA typu tablicowego. Specyfika tego typu struktur sprawia, że istota dekompozycji sprowadza się do podziału diagramów na części o określonej liczbie zmiennych, realizowane w blokach typu tablicowego (Lookup Table) o niewielkiej liczbie wejść. Istnieje jednak możliwość zastosowania tychże diagramów w procesie dekompozycji ukierunkowanej na implementację funkcji logicznych w układach CPLD typu PAL. Tym razem, istota dekompozycji polega na dopasowaniu fragmentów diagramów do możliwości implementacyjnych bloków typu PAL, zawierających określoną, zwykle niewielką liczbę iloczynów. Diagramy BDD, stanowiąc efektywną reprezentację funkcji pozwalają na znaczną oszczędność pamięci w stosunku do metod dekompozycji, w których funkcje są reprezentowane za pomocą tablic lub wyrażeń będących sumą iloczynów.

W artykule przedstawiono pomysły i związane z nimi algorytmy poszczególnych etapów dekompozycji funkcji, opisanych za pomocą diagramów BDD. Widać, że diagramy te można dzielić, wplatając w proces podziału ograniczenia implementacyjne wykorzystywanych struktur programowalnych oraz zasoby bloków typu PAL, związane na przykład z możliwością realizacji funkcji w postaci prostej lub zanegowanej.

6. Literatura

[1] S.B. Akers “Binary decision diagram”, IEEE Transactions on Computers, Vol.C-27, No.6, str.509-516, czerwiec 1978
[2] R. E. Bryant “Graph Based Algorithms For Boolean Function Manipulation”, IEEE Transactions on Computers, Vol.C-35, No.8, str.677-691, sierpień 1986
[3] T. Sasao “FPGA Design by Generalized Functional Decomposition in Logic Synthesis and Optimization”, Kluwer Academic Publishers, Boston, London, Dotdrecht, 1993
[4] Giovanni De Micheli „Synteza i optymalizacja układów cyfrowych”, Wydawnictwa Naukowo-Techniczne, Warszawa 1998
[5] Srinivas Devadas, Abhiji Ghosh, Kurt Keutzer „Logic Synthesis”, McGraw-Hill, Inc. 1994
[6] Shin-Ichi Minato “Binary Decision Diagrams and Applications For VLSI CAD” Kluwer Academic Publisher 1995
[7] F. Somenzi “CUDD: CU Decision Diagram Package” Department of Electrical and Computer Engineering, University of Colorado at Boulder 2005
[8] A. Milik „Rekonfigurowalny sterownik logiczny”, rozprawa doktorska, Politechnika Śląska, Gliwice 2003

Artykuł recenzowany