INFORMATYKA
CZASOPISMO TECHNICZNE
TECHNICAL TRANSACTION
COMPUTER SCIENCE AND INFORMATION SYSTEMS
WYDAWNICTWO POLITECHNIKI KRAKOWSKIEJ
1-I/2011
ZESZYT 9
ROK 108
ISSUE 9
YEAR 108

MIROSŁAWA NOWICKA, DANUTA OJRZEŃSKA-WÓJTER *

SYNTEZA UKŁADÓW KOMBINACYJNYCH W STRUKTURACH FPGA Z WBUDOWANYMI BLOKAMI PAMIĘCI

SYNTHESIS OF COMBINATIONAL CIRCUITS IN FPGAs WITH EMBEDDED MEMORY BLOCKS

Streszczenie

Artykuł porusza problem realizacji układów kombinacyjnych w strukturach FPGA z wbudowanymi blokami pamięci ROM. Zaprezentowano system uniwersytecki, implementujący zaawansowane procedury syntezy logicznej, który umożliwia rozkład układów kombinacyjnych na pamięci M512 i M4K. Przedstawiono wyniki eksperymentów potwierdzające wpływ zastosowania zaprezentowanego oprogramowania na efektywność wykorzystania zasobów nowoczesnych struktur FPGA.

Słowa kluczowe: synteza logiczna, układy FPGA, dekompozycja szeregowa, dekompozycja równoległa, redukcja argumentów

Abstract

The paper discusses the problem of implementation of combinational circuits in FPGA structures with embedded memory blocks. University software tool is presented, implementing advanced synthesis procedures, that allows decomposition of combinational circuits into M512 and M4K memory blocks. Results of experiments are presented that prove the influence of presented software on the efficiency of FPGA resources utilization.

Keywords: logic synthesis, FPGA devices, serial decomposition, parallel decomposition, argument reduction

* Dr Mirosława Nowicka, mgr inż. Danuta Ojrzeńska-Wójter, Instytut Telekomunikacji, Wydział Elektroniki i Technik Informacyjnych, Politechnika Warszawska.

Artykuł złożony do Wydawnictwa: 10.07.2010

1. Wstęp

We współczesnych systemach cyfrowych coraz powszechniej stosuje się układy programowalne FPGA z kilku powodów: po pierwsze z racji ich elastyczności (możliwość rekonfiguracji w działającym sprzęcie); po drugie ze względu na znaczne zasoby funkcjonalne i możliwości (rozbudowane architektury logiczne, wyposażenie w standardowe macierze komórek od kilku tysięcy do stu kilku tysięcy, pamięci wbudowane, specjalizowane bloki funkcjonalne np. DSP, interfejsy komunikacyjne np. do obsługi pamięci zewnętrznej, interfejsy linii transmisyjnych); po trzecie ze względu na udostępniane narzędzia do projektowania (środowiska programowe, platformy uruchomieniowe, narzędzia do weryfikacji) [1].

Konstrukcja logiczna układów FPGA bezpośrednio wpływa na: koszt produktu, osiągi oraz maksymalne „upakowanie". Wraz z pojawieniem się najnowszych układów programowalnych zawierających wbudowane bloki pamięci ROM (Read Only Memory) powstała potrzeba realizacji układów kombinacyjnych w strukturach pamięci ROM. Zazwyczaj układy kombinacyjne są realizowane w strukturach FPGA opartych na 4 wejściowych LUT. Często jednak struktura sieci komórek opisujących funkcje boolowskie jest dość skomplikowana i to nie tylko ze względu na liczbę otrzymanych komórek z rozkładu, ale również ze względu na liczbę poziomów logicznych sieci komórek i połączeń między nimi. Stopień złożoności i wielopoziomowość struktury wpływa na niezawodność i szybkość działania układu. Realizacja funkcji boolowskiej w pamięciach ROM posiada o wiele prostszą strukturę logiczną dzięki temu, że pamięci ROM typu M512 i M4K można dowolnie konfigurować odpowiednio do 512 i 4K bitów (tabela 1).

Tabela 1

Możliwe konfiguracje pamięci *M512* i *M4K*

M512 ROM Block (32 × 18 Bits)	M4K ROM Block (128 × 36 Bits)
512 × 1	4K × 1
256 × 2	2K × 2
128 × 4	1K × 4
64 × 8	512 × 8
64 × 9	512 × 9
32 × 16	256 × 16
32 × 18	256 × 18
	128 × 32
	128 × 36

Niemniej komercyjne środowiska projektowe nie wykorzystują efektywnie bloków wbudowanych, a w swoich rozwiązaniach nie uwzględniają niestandardowego wykorzystania tych pamięci jako tablicowego modelu realizacji funkcji boolowskich. Natomiast nowe technologie stały się od pewnego czasu wyzwaniem dla akademickich systemów do projektowania i optymalizacji. Jednym z nich jest system Demain_ROM[1] opracowany w Insty-

[1] System Decomposer-Demain-ROM został częściowo wykonany w ramach pracy naukowej Projektowanie układów cyfrowych do zastosowań w systemach i sieciach telekomunikacyjnych

tucie Telekomunikacji Politechniki Warszawskiej. Program ten wspiera syntezę układów kombinacyjnych w strukturach FPGA z wbudowanymi blokami pamięci.

2. Narzędzia do projektowania

Prezentowany system opiera się na podstawowych metodach syntezy logicznej takich jak: redukcja argumentów, szeregowa dekompozycja rozłączna, szeregowa dekompozycja nierozłączna oraz dekompozycja równoległa.

2.1. Redukcja argumentów

Zadaniem algorytmu redukcji argumentów jest wyznaczenie takich argumentów funkcji, które nie wpływają na jej zachowanie. Są one zbędne, można je więc usunąć, przy czym tablica prawdy będzie nie sprzeczna. W tym celu należy wyznaczyć wszystkie minimalno-argumentowe reprezentacje funkcji. Takich zbiorów będących minimalno-argumentowymi reprezentacjami funkcji może być wiele. Mogą mieć również różne liczności. Wybiera się zwykle jedną z najmniejszych, kierując się heurystyczną przesłanką redukcji złożoności kolejnych etapów. Jest wiele sposobów wyznaczania minimalno-argumentowych reprezentacji funkcji. Tutaj posłużono się metodą wyznaczania minimalnego pokrycia kolumnowego. Przykładową funkcję przedstawia tabela 2a.

Tabela 2

Zapis funkcji przed i po redukcji argumentów

a)

\x	x_1	x_2	x_3	x_4	x_5	x_6	f
1	1	0	0	0	1	1	0
2	1	1	0	0	0	1	0
3	0	1	0	1	0	0	0
4	1	1	0	1	1	1	0
5	1	1	0	1	1	0	1
6	1	0	1	0	0	1	1
7	1	0	0	0	0	1	1
8	1	1	1	1	0	1	1

b)

\x	x_2	x_3	x_5	x_6	f
1	0	0	1	1	0
2	1	0	0	1	0
3	1	0	0	0	0
4	1	0	1	1	0
5	1	0	1	0	1
6	0	1	0	1	1
7	0	0	0	1	1
8	1	1	0	1	1

Funkcja posiada jedno wyjście i sześć zmiennych wejściowych. Zbiór argumentów funkcji to $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$. Aby wyznaczyć tablicę dla kolumnowego pokrycia należy dokonać porównania wszystkich par wierszy o różnych wyjściach. Jeśli oba wektory wejściowe mają na tej samej pozycji identyczne wartości to w wektorze porównania wpisana jest na tej pozycji wartość 0, w przeciwnym wypadku wartość 1. Wyniki kolejnych porównań należy wpisywać do tablicy pokrycia pamiętając o tym, aby redukować takie

o wysokiej wiarygodności działania, realizowanych przy użyciu struktur programowalnych FPGA/CPLD finansowanej ze środków na naukę w latach 2007–2010 jako projekt badawczy nr N517 003 32/0583.

zbiory wektorów $\mathbf{C}_i$, dla których istnieją $\mathbf{C}_j$: $\mathbf{C}_j \subseteq \mathbf{C}_i$. Funkcja zapisana w tablicy 2 ma 8 wierszy.

Pierwsze cztery wiersze mają na wyjściu wartość 0, a cztery pozostałe wartość 1. Każdy wiersz od 1 do 4 należy porównać z każdym wierszem o 5 do 8. Dla przykładu wypisano porównania wiersza 1 z wierszami od 5 do 8.

C1 :	0	1	0	1	0	1	ponieważ C2 ⊃ C3 i C4 ⊃ C3 otrzymano C1 : 0 1 0 1 0 1
C2 :	0	0	1	0	1	0	
C3 :	0	0	0	0	1	0	
C4 :	0	1	1	1	1	0	

Po wykonaniu pozostałych porównań, końcowa tablica dla pokrycia jest postaci

x_1	x_2	x_3	x_4	x_5	x_6
0	0	0	0	1	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	0	0	1

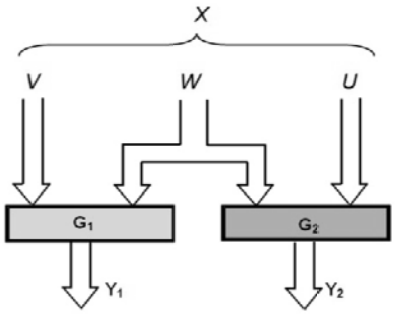
Rozwiązaniem są dwa zbiory: $X_1 = \{x_2, x_3, x_5, x_6\}$, $X_2 = \{x_2, x_4, x_5, x_6\}$. Oznacza to, że istnieją dwa zbiory minimalno-argumentowe. Argumenty powtarzające się w każdym rozwiązaniu (pogrubione) są zmiennymi niezbędnymi (w tablicy pokrycia jest to pozycja w wierszu posiadającym jedną jedynkę) i tych zmiennych nie wolno usunąć. Po usunięciu ich tablica prawdy byłaby sprzeczna. Można natomiast usunąć wszystkie zmienne nie należące do zbioru $X_1 \cup X_2$ czyli tu tylko x_1 (nie należy do żadnego rozwiązania). W zależności od rozpatrywanego rozwiązania można usunąć albo x_3 albo x_4. Oznacza to, że rozwiązanie minimalno-argumentowe jest sumą dwóch zbiorów: X_N – zbioru zmiennych niezbędnych i X_P – zbioru pozostałych zmiennych potrzebnych do realizacji funkcji. Przy czym $X_N \cap X_P = \emptyset$ (nie posiadają wspólnych elementów), $X_N \cup X_P \neq \emptyset$ (oba nie mogą być puste). Często wyznaczanie minimalno-argumentowej reprezentacji funkcji sprowadza się właśnie do dwóch etapów: najpierw do wyznaczenia zbioru X_N, a później do wyznaczenia X_P. Metoda ta szczegółowo opisana jest [2]. W tabeli 2b zapisana została funkcja zredukowana z 6 do 4 zmiennych. Do reprezentacji jej użyto rozwiązania X_1.

2.2. Dekompozycja równoległa

Dekompozycja równoległa polega na podziale zbioru wyjść funkcji na dwa rozłączne podzbiory i realizacji każdego z nich oddzielnie (rys. 1). Przy założeniu, że X zbiór argumentów funkcji pierwotnej, Y zbiór jej wyjść, dekompozycja spełnia następujące zależności

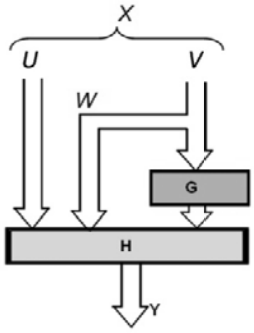
$$Y_1 \subset Y, \quad Y_2 \subset Y, \quad Y_1 \cap Y_2 = \emptyset, \quad Y_1 \cup Y_2 = Y$$

$V \cup U \cup W \subseteq X$ (zbiór wejść funkcji pierwotnej) gdzie W jest zbiorem zmiennych od których zależą wszystkie wyjścia funkcji pierwotnej, natomiast zbiory V i U to zbiory zmiennych od których zależą tylko wyjścia odpowiednio bloków G_1 i G_2.



Rys. 1. Dekompozycja równoległa

Fig. 1. Parallel decomposition



Rys. 2. Dekompozycja szeregowa

Fig. 2. Serial decomposition

Funkcja pierwotna zastępowana jest dwoma funkcjami o mniejszej złożoności w sensie liczby wyjść i/lub liczby zmiennych wejściowych. Przy czym, jeśli nie zastosuje się do każdego z bloków redukcji argumentów to zbiory U i V są puste, natomiast zbiór W to zbiór X wszystkich argumentów funkcji. Zastosowanie redukcji argumentów w powiązaniu z dekompozycją równoległą tam, gdzie jest to możliwe powoduje zmniejszenie liczby argumentów w jednym lub dwóch blokach. Ma to istotny wpływ na jakość struktury wielopoziomowej. Na blokach funkcji o mniejszej złożoności łatwiej wykonywać dalsze doświadczenia i szybciej można uzyskać oczekiwany rezultat.

2.3. Dekompozycja szeregowa

Dla dekompozycji szeregowej (rys. 2) wprowadzono oznaczenia: X – zbiór argumentów, Y – zbiór wyjść. Zbiory V i U są podzbiorami rozłącznymi zbioru X takimi, że $V \cup U = X$, natomiast zbiór $W \subset V$ (W jest podzbiorem zbioru V). Jeśli $W = \emptyset$ to dekompozycja szeregowa jest rozłączna. Dla dekompozycji szeregowej nierozłącznej $W \neq \emptyset$.

Rozłączna dekompozycja szeregowa polega na zastąpieniu bloku o zadanej liczbie wejść i wyjść dwoma blokami (na rysunku 2 oznaczone G i H) o mniejszej złożoności. Dzieli ona zmienne wejściowe na dwa rozłączne podzbiory. Dekompozycja szeregowa istnieje wtedy, gdy liczba wyjść z bloku G jest mniejsza od liczności zbioru V. Wtedy sygnały wejściowe należące do zbioru U i wyjścia bloku G tworzą wspólnie zbiór wejść do bloku H.

Dekompozycja nierozłączna ma miejsce wtedy, gdy można zmniejszyć liczbę wyjść bloku G dekompozycji rozłącznej kosztem dodania do wejść bloku H podzbioru W zbioru V. Aby taka dekompozycja miała sens, liczność zbioru W musi być mniejsza lub równa liczbie zredukowanych wyjść bloku G. Zazwyczaj dekompozycja nierozłączna zmniejsza liczbę wyjść bloku G o jeden i dodaje jedną zmienną ze zbioru V do wejść bloku H. Złożoność bloku H pozostaje wtedy bez zmian, zaś złożoność funkcji G maleje ze względu na liczbę wyjść.

3. Realizacja funkcji boolowskich w układach FPGA z wbudowanymi blokami pamięci

Można zadać pytanie, czy cały ten zaprezentowany aparat do syntezy logicznej jest niezbędny do realizacji funkcji boolowskich w układach FPGA na pamięciach wbudowanych? Wydawałoby się, że banalne są realizacje w strukturach ROM typu M512 i M4K dla układów kombinacyjnych o liczbie wejść mniejszej od 13. Oczywiście twierdzenie to jest prawdziwe pod warunkiem, że nie chodzi o koszt realizowanego układu, a co za tym idzie o minimalizację wykorzystywanych zasobów realizowanego układu.

Prześledźmy prosty przykład z funkcją f51m o 8 wejściach i 8 wyjściach.

Rozwiązanie 1: M4K. Pamięć M4K można skonfigurować do bloku o 8 wejściach i 16 wyjściach. Funkcja zajmuje wówczas tylko połowę pamięci ROM.

Rozwiązanie 2: 4 × M512. Rozpatrywaną funkcję można podzielić na 4 bloki o rozmiarach (8, 2) i każdy z tych bloków może być zrealizowany w pamięci ROM typu M512 skonfigurowanej do 8 wejść i 2 wyjść.

Rozwiązanie 3: 2 × M512. W tym przypadku posłużymy się syntezą logiczną i skorzystamy z podglądu tablicy funkcji otrzymanej po przeprowadzeniu redukcji argumentów każdego z wyjść. Ma on postać

f00 :	1	1	1	1	1	1	1	1
f01:	0	1	1	1	1	1	1	1
f02:	0	0	1	1	1	1	1	1
f03:	0	0	0	1	1	1	1	1
f04:	0	0	0	0	1	1	1	1
f05 :	0	0	0	0	0	1	1	1
f06 :	0	0	0	0	0	0	1	1
f07 :	0	0	0	0	0	0	0	1

Kolejne wiersze odpowiadają numerom wyjść, a kolumny numerom wejść. Jeśli w wierszu na pewnej pozycji występuje 0 to oznacza, że argument (wejście) odpowiadający tej pozycji jest niepotrzebny dla tego wyjścia i gdyby rozpatrywać to wyjście jako samodzielną funkcję, można a nawet trzeba by go zredukować. Natomiast pozycja z wartością 1 oznacza, że argument odpowiadający tej pozycji jest potrzebny. Nie można więc go usunąć. Tablica zależności argumentów wejściowych od wszystkich wyjść pokazuje, że tylko pierwsze wyjście jest zależne od wszystkich wejść. Pozostałe w kolejności od 7, 6, 5, 4, 3, 2 i 1 argumentów. Łatwo zauważyć więc, jak należy łączyć wyjścia w bloki, aby liczba pamięci wbudowanych była jak najmniejsza. Podział wyjść na dwa bloki o numerach wyjść 0, 1 oraz 2, 3, 4, 5, 6, 7 tworzy bloki o parametrach (8, 2) oraz (6, 6). Pierwszy blok jest pamięcią M512 skonfigurowaną do 8 wejść i 2 wyjść, natomiast drugi blok może być zrealizowany w pamięci M512 skonfigurowanej jako 6 wejść i 8 wyjść.

Analiza trzech rozwiązań pokazuje, że najlepszym rozwiązaniem jest **rozwiązanie 3**. Pojemność użytych pamięci wbudowanych wynosi 1024 bity i jest czterokrotnie mniejsza w stosunku do **rozwiązania 1** i dwukrotnie w stosunku do **rozwiązania 2**. Ale uzyskanie

takiego rozwiązania wiąże się z opracowaniem i zaimplementowaniem skomplikowanych procedur syntezy logicznej.

Nawet jeśli w niektórych przypadkach można poradzić sobie z realizacją układu kombinacyjnego w strukturach ROM typu M512 i M4K bez znajomości zagadnień z syntezy logicznej (uzyskując często o wiele gorsze rozwiązanie), to dla funkcji boolowskiej o liczbie argumentów większej od 12 jest już niemożliwe. Pokazane zostało to na przykładzie benchmarku z funkcją e_7 o 16 wejściach i 5 wyjściach w punkcie 5.1.1 i 5.1.2. Tu bez dekompozycji szeregowej nie jest możliwa realizacja funkcji o 16 argumentach, ponieważ największą liczbę wejść ma pamięć M4K skonfigurowana jako blok o 12 wejściach i 1 wyjściu.

4. Opis systemu Decomposer-Demain-ROM

System Decomposer-Demain-ROM jest pakietem oprogramowania służącym do rozkładu złożonych układów kombinacyjnych realizowanych w strukturach programowalnych typu FPGA zarówno dla pamięci typu LUT o zadanych parametrach (określonej liczbie wejść i wyjść) jak i dla standardowych wbudowanych bloków pamięci typu ROM lub kombinację struktur typu LUT i ROM. Program przystosowany jest do dwóch typów pamięci ROM: M512 o pojemności 512 bitów oraz M4K o pojemności 4K bitów. Układy do dekompozycji są opisywane:

– tablicami prawdy zapisanymi w standardzie Berkeleya,

lub

– wektorem współczynników dla arytmetyki rozproszonej AD (opis w p.4.2).

Decomposer-Demain-ROM posiada możliwość rozkładu układu początkowego na dowolną liczbę bloków (a nie tylko na dwa jak przy dekompozycji równoległej) o zadanych przez użytkownika parametrach.

System Demain_ROM jest programem interakcyjnym, w którym użytkownik sam buduje strukturę układu. Użytkownik steruje procesem syntezy i to on podejmuje decyzję o wyborze strategii. Jest to zaletą tego systemu, ponieważ dzięki temu można dla każdego układu otrzymać wiele rozwiązań i po analizie wybrać to, które będzie spełniało kryteria szukanego rozwiązania.

System opiera się na podstawowych metodach syntezy logicznej:

– redukcji argumentów, pozwalającej na redukcję zbędnych argumentów funkcji. Zbędne argumenty pojawiają się zazwyczaj w pierwotnej jednowyjściowej funkcji, w wielowyjściowych blokach budowanych za pomocą arytmetyki rozproszonej i po rozkładzie wielowyjściowego bloku wykonanego za pomocą dekompozycji równoległej. Dekompozycja równoległa ma wbudowaną redukcję argumentów, co oznacza, że otrzymane bloki będące wynikiem jej rozkładu nie posiadają już zbędnych argumentów.
– dekompozycji funkcjonalnej, łączącej dwie jej procedury: szeregową (rozłączną i nierozłączną) oraz równoległą. Umożliwia to przekształcenie układu wejściowego w wielopoziomową sieć układów o zadanych parametrach (LUTy o określonej liczbie wejść i wyjść lub standardowe bloki typu ROM lub kombinację struktur typu LUT i ROM).

Praca z programem Decomposer-Demain-ROM polega na każdorazowym wyborze strategii przez użytkownika na każdym kroku rozkładu. Wybór strategii ułatwiają narzę-

dzia, które zostały w tym celu wprowadzone. Taki interakcyjny sposób działania daje więc bogate możliwości w realizacji struktur wielopoziomowych. A przeprowadzane badania wykazują, że zarówno zmiana kolejności występowania poszczególnych typów dekompozycji, jak też parametry (liczba wejść i wyjść) bloku *G* w dekompozycji szeregowej, silnie wpływają na końcową strukturę układu, którą w efekcie można tak dobrać po kilku przeprowadzonych doświadczeniach, aby uzyskać polepszenie parametrów końcowej struktury układu [4, 6].

Aby ułatwić użytkownikowi wybór odpowiedniej strategii przy pomocy przemyślanej decyzji, wprowadzono takie udogodnienia jak: podgląd na dekompozycję równoległą oraz podgląd na dekompozycję szeregową wykonywaną, ale nie zrealizowaną. Oba te pojęcia będą omówione przy opisie przykładowych rozwiązań przykładów (pkt. 5). Niemniej zasadnicze znaczenie dla użytkownika ma „budowa" funkcji. Wprowadzono więc dodatkową opcję: Włączenie **podglądu tablicy prawdy** przed rozpoczęciem rozkładu.

Program bada zależność wszystkich wyjść od argumentów wejściowych (na podstawie wyznaczonych realizacji minimalno-argumentowych dla poszczególnych wyjść). Na podstawie wyświetlanego podglądu, użytkownik może wykonać własny rozkład na bloki lub nie. Jeśli wykonywany jest własny rozkład na bloki, to należy podawać kolejno liczbę wyjść danego bloku, a następnie numery wyjść należących do danego bloku. Można tworzyć dowolnie dużo bloków. Wprowadzenie 0 jako parametr bloku spowoduje zakończenie wprowadzania. Jeśli pozostaną jakieś numery wyjść nie wprowadzone, to zostaną one przypisane do jednego, następnego bloku. Przykładowy wydruk:

The H table under consideration has 11 inputs and 6 outputs
inputs: i0 i1 i2 i3 i4 i5 i6 i7 i8 i9 i10
outputs: o0 o1 o2 o3 o4 o5

f00 :	0	1	1	1	1	1	1	1	1	1	0
f01:	1	1	1	1	1	1	1	1	1	1	1
f02:	1	1	1	1	1	1	1	1	1	1	1
f03:	1	1	1	1	1	0	1	1	1	1	1
f04:	1	1	1	1	1	0	1	1	1	1	1
f05 :	1	1	1	1	1	0	1	1	1	1	1

oznacza, że funkcja posiada 11 wejść i 6 wyjść. Pierwsze wyjście f00 zależy od dziewięciu argumentów, wyjścia f01 i f02 od 11 argumentów, a pozostałe trzy od f03, f04, f05 od dziesięciu, tych samych argumentów. Po wyświetleniu podglądu należy podjąć decyzję, czy użytkownik sam dokona podziału, czy też wykona się on automatycznie.

MAKE DIVIDING INTO BLOCKS ? (*y/n*)

Po wprowadzeniu znaku *n*, program wyświetli pogląd na dekompozycję równoległą i będzie już możliwy tylko rozkład na dwa bloki. Po wprowadzeniu znaku *y*, należy wprowadzać dane zgodnie z podpowiedziami wyświetlanymi na ekranie.

4.1. Strategie dekompozycji

Jedną z najważniejszych czynności w realizacji programu, od której zależą wyniki rozwiązania, jest wybór strategii postępowania. Są cztery opcje wyboru strategii: *p* dla dekompozycji równoległej, dwie: *s* i *c* dla dekompozycji szeregowej oraz *r* do zapamiętywania bloku w pamięci ROM. Pojawiają się one na ekranie w postaci:

Do you prefer parallel decomposition ? : *p*
OR continue serial decomposition as suggested ? : *s*
OR continue serial decomposition with changed parameters ? : *c*
OR save *xxxxx* ? : *r*

Dla dekompozycji szeregowej mamy do wyboru dwie opcje: *s* i *c*.

Opcja *s* – dekompozycja szeregowa z parametrami sugerowanymi przez program. Dla każdej funkcji poddawanej dekompozycji, na początku są sugerowane parametry odpowiadające komórce (parametry jej wprowadzone są przed uruchomieniem programu), a jeśli dla takich parametrów nie istnieje rozwiązanie, sugerowane są inne parametry które doprowadzą do końcowego rozwiązania (nawet użytkownika bez „większej" wiedzy na temat układów logicznych), ale nie musi to być rozwiązanie optymalne (i często nie jest). Można zawsze zmienić parametry sugerowane za pomocą opcji *c*.

Opcja *c* – dekompozycja szeregowa z parametrami wprowadzonymi z klawiatury przez użytkownika. Pozwala to na zmianę parametrów sugerowanych. Po wprowadzeniu znaku *c* należy wpisać liczbę wejść i wyjść dla funkcji *G*. Jeśli chcemy wykonać dekompozycję z podglądu na dekompozycję szeregową, to po wprowadzeniu znaku **c** i wciśnięciu klawisz [Enter], jako liczbę wejść do bloku należy podać wartość pierwszej kolumny odpowiedniego wiersza, a do liczby wyjść należy wpisać wartość z drugiej kolumny tego samego wiersza.

Opcja *p* – dekompozycja równoległa. Ta strategia pojawia się tylko wtedy gdy blok funkcji posiada więcej niż jedno wyście. Wykonuje podział wyjść na dwa rozłączne podzbiory o parametrach podanych w podglądzie na dekompozycję równoległą. Sugerowane parametry bloków można zmienić przez podanie liczby wyjść bloku, którego wyjścia mają być zależne od największej lub najmniejszej liczby argumentów. Drugi blok będzie zawierał pozostałe wyjścia.

Na ekranie po wprowadzeniu znaku *p* w strategiach dekompozycji pojawia się napis:

Change block size ? (*y/n*) :

Po wprowadzeniu znaku *n* jeden z bloków będzie aktualnym blokiem do dalszego rozkładu, natomiast drugi zapamiętywany jest w pomocniczym pliku, do którego program powróci w odpowiednim momencie.
Po wprowadzeniu znaku *y* pojawia się wydruk:

Block of inputs with min. no. of arguments ?: *m*
Block of inputs with max. no. of arguments ?: *l*

Bez względu na to jaki wprowadzamy znak ***m*** lub ***l*** pojawia się wydruk:

number of outputs (<liczba_wyjsc):

Należy teraz wprowadzić ustaloną liczbę wyjść. Dopiero teraz program utworzy jeden blok zależny od najmniejszej lub największej liczby argumentów, a do drugiego przypisze pozostałe wyjścia.

Opcja *r* – zapamiętanie aktualnego bloku w pamięci M512 lub M4K. Występuje tylko wtedy, gdy aktualnie rozpatrywany blok może być umieszczony w pamięci ROM M512 lub M4K. Wtedy przed znakiem ? w miejsce *xxxx* podany jest typ pamięci ROM oraz jej rozmiar. Użytkownik ma wtedy świadomość, czy opłacalne jest umieszczanie bloku w pamięci ROM. Np. jeśli blok ma rozmiar: 7 wejść i 1 wyjście, to wśród możliwych strategii pojawi się:

OR save ROM – M512(7x4) ? : *r*

i wtedy użytkownik musi zadecydować, czy jest to opłacalne. Oczywiście zależy to od kryteriów, jakie nałożone są na dany rozkład.

W programie przyjęto następujące oznaczenia dla różnych konfiguracji pamięci M512 i M4K, i zapisano je odpowiednio w postaci M4K($n \times m$) lub M512($n \times m$) gdzie n to liczba wejść do bloku a m liczba wyjść z bloku. Największą liczbę wejść mają bloki M4K skonfigurowane do M(12 × 1). Oznacza to, że dla funkcji o liczbie argumentów większej od 12 należy zastosować specjalne metody, aby można je realizować w blokach ROM.

4.2. Wektor współczynników dla arytmetyki rozproszonej (AD)

Dla celów testowych wprowadzono opcję tworzenia tablicy prawdy za pomocą reguł arytmetyki rozproszonej [3, 5]. Aby utworzyć tablicę prawdy całkowicie określoną o n argumentach wejściowych należy podać wektor o długości n złożony z liczb całkowitych (liczby dodatnie, ujemne, dodatnie i ujemne) $[c_{n-1}, c_{n-2}, ...c_1, c_0]$. Zbudowana tablica prawdy jest postaci

x_0	x_1	x_2	...	x_{n-2}	x_{n-1}	y
0	0	0	...	0	0	y_0
0	0	0	...	0	1	y_1
0	0	0	...	1	0	y_2
1	1	1	...	1	0	y^n_{2-2}
1	1	1	...	1	1	y^n_{2-1}

gdzie wiersze tablicy reprezentują kolejne liczby dziesiętne zapisane w kodzie binarnym na n pozycjach, rozpoczynając od zera. Natomiast wartości wyjść y_i są zapisem binarnym liczby dziesiętnej

$$y_j = \sum_{i=0}^{n-1} c_i \cdot x_{ji}$$

gdzie:

j – j-ty wiersz w tablicy,
i – numer zmiennej wyjściowej.

W przypadku różnych długości wektorów wyjściowych (y_i), wybierana jest maksymalna. Pozostałe wektory o mniejszej długości, uzupełniane są z lewej strony zerami w przypadku liczby dodatniej, lub jedynkami dla liczb ujemnych. Jeśli maksymalna liczba dodatnia i minimalna ujemna kodowane są na jednakowej liczbie bitów to długość wektora zwiększana jest o jeden.

Np. jeśli w pliku ala.txt jest zapisany wektor [3, –18, 5, 15], to program wygeneruje plik ala.pla z tablicą prawdy o 4 wejściach i 6 wyjściach postaci:

```
.type fr
.i 4
.o 6
.ilb i0 i1 i2 i3
.ob o0 o1 o2 o3 o4 o5
0000 000000
0001 000011
0010 101110
0011 110001
0100 000101
0101 001000
0110 110011
0111 110110
1000 001111
1001 010010
1010 111101
1011 000000
1100 010100
1101 010111
1110 000010
1111 000101
.end
```

Dla każdego wiersza tablicy wyjścia wyznaczane są: $i_0 * c_0 + i_1 * c_1 + i_2 * c_2 + i_3 * c_3$. Dla wektora [3, –18, 5, 15]: $c_0 = 15$, $c_1 = 5$, $c_2 = -18$, $c_3 = 3$. Maksymalną wartością dodatnią utworzoną ze współczynników c_i jest $c_3 + c_2 + c_0$ i wynosi 23 (binarnie kodowane na pięciu pozycjach), najmniejszą ujemną jest wartość –18 dla c_2 (binarnie kodowana na sześciu pozycjach). Max(5,6) wynosi 6, więc tablica prawdy ma sześć wyjść.

W przypadku, gdy funkcja dana jest za pomocą wektora współczynników DA, rozkład zależności argumentów dla poszczególnych wyjść tworzony jest bez budowy tablicy prawdy. Przeprowadzana analiza oparta jest tylko na własnościach arytmetyki rozproszonej ponieważ jest o wiele szybsza niż analogiczna przeprowadzana na tablicy prawdy. Oczywiście tablica prawdy jest również tworzona ponieważ na niej przeprowadzany jest właściwy rozkład.

5. Przykłady rozwiązań za pomocą programu Decomposer-Demain-ROM

5.1. Rozkład funkcji e7.pla

Funkcja z benchmarku e_7 posiada 16 wejść i 5 wyjść. Po uruchomieniu programu pojawia się podgląd na budowę funkcji w postaci wydruku rozkładu zależności poszczególnych wyjść od argumentów. Jest on postaci

f00 :	1	0	0	0	1	1	1	1	1	0	0	0	1	0	0	0
f01:	1	1	0	0	1	1	1	1	1	1	0	0	1	1	0	0
f02:	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0
f03:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f04:	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1

Następnie program pyta, czy będzie wykonany samodzielny podział na bloki.

MAKE DIVIDING INTO BLOCKS ? (y/n)

Wybór opcji i sposób podziału zależy tylko od użytkownika. W kolejnych rozdziałach przedstawiono dwie drogi postępowania w zależności od wykorzystywania zależności wyjść od argumentów.

5.1.1. Rozkład funkcji e7.pla wykorzystujący tablicę zależności wyjść od argumentów

Przed wykonaniem samodzielnego podziału na bloki należy dokonać wstępnej analizy. Wyjścia o numerach 0 i 1 zależą wspólnie od 10 argumentów mogą być zrealizowane w pamięci M4K skonfigurowanej jako 10 × 4 (10 wejść, 4 wyjścia). Wyjście o numerze 2 zależne jest od 13 argumentów, o numerze 3 od 16, a ostatnie o numerze 4 od 13 argumentów. Gdyby każde z wyjść o numerach 2, 3 i 4 rozkładać indywidualnie, to każdy z nich musiałby być zrealizowany na co najmniej dwóch pamięciach ROM lub jednej pamięci ROM plus bloki LUT(4,1), a więc wszystkie łącznie na od 3 do 6 blokach pamięci ROM plus ewentualnie dodatkowe LUT(4,1). Z drugiej strony, łączenie wyjść o numerach 2 i 4 nie zmniejszyłoby liczby argumentów w tym bloku, taki blok byłby zależny od 16 wejść. Warto więc je dołączyć do bloku wyjść o numerze 3 – liczba argumentów tego bloku pozostanie bez zmian. Teraz można odpowiedzieć na pytanie:

MAKE DIVIDING INTO BLOCKS ? (y/n)

Po wprowadzeniu znaku *y* zdefiniowano podział w sposób:

Input the size and output number for each block(0-end of inputting)
Size of the block (number of outputs)? : 2
Numbers of outputs : 0
1
Size of the block(number of outputs)? : 0 Size of the block ? : 0
to oznacza, że ostatni blok będzie
zawierał pozostałe numery wyjść (2, 3, 4).

Teraz program przechodzi kolejno do rozkładu zapamiętanych w plikach bloków. Blok o wyjściach o0 i o1 po dekompozycji równoległej ma zredukowane argumenty do 10 wejść i za pomocą opcji *r* może być zapamiętany w bloku M4K.

Dla drugiego bloku funkcji o rozmiarze (16, 3) istnieją dekompozycje szeregowe rozłączne z funkcjami $G(8, 3)$, $G(9, 4)$, $G(10, 5)$ oraz mogą być zrealizowane jako nierozłączne z funkcją $G(8, 2)$, $G(9, 3)$ i $G(10, 4)$. Każda z nich zmniejsza o 5 liczbę wejść do następnego bloku H, który będzie miał 11 wejść i 3 wyjścia. Wybrano $G(8, 2)$, ponieważ może być zrealizowana w jednym M512. Funkcja $H(11, 3)$ nie może być jeszcze zrealizowana w pamięci ROM. Niemniej jedno z jej wyjść zależy od 7 argumentów, a dwa pozostałe od 10. Po wykonaniu dekompozycji równoległej na dwa bloki i po redukcji argumentów można je zrealizować w pamięciach ROM: jeden z nich w pamięci M512, a drugi w pamięci M4k. **Otrzymane rozwiązanie to: 2 × M512 + 2 × M4K**.

5.1.2. Rozkład funkcji e7.pla nie wykorzystujący tablicy zależności wyjść od argumentów

W przypadku tym nie jest wykonywany wstępny, samodzielny rozkład na bloki. Jeśli po wyświetleniu tablicy rozkładu wyjść od argumentów na zapytanie:

MAKE DIVIDING INTO BLOCKS ? (y/n)

zostanie użyta opcja *n*, nie ma już możliwości samodzielnego łączenia wyjść w bloki. Należy już raczej kierować się tylko podpowiedziami programu.

The H table under consideration has 11 inputs and 3 outputs
inputs: i0 i1 i2 i3 i4 i5 i6 i7 i8 i9 i10 i11 i12 i13 i14 i15
outputs: o0 o1 o2 o3 o4
decomposition parallel block1 (16,2), block2(13,3)
Serial decomposition is suggested with G function parameters
assumed to be equal to the declared cell parameters
Do you prefer parallel decomposition ?: p
OR continue serial decomposition as suggested ?: s
OR continue serial decomposition with changed parameters ?: c

Z podglądu na dekompozycję równoległą można tylko wnioskować, że dwa wyjścia są zależne od 16 argumentów. Natomiast nie wiadomo nic o liczbie argumentów każdego z nich. Podobnie z podanego podglądu wiadomo tylko, że trzy wyjścia mają liczbę argumentów mniejszą lub równą 13. Wykonana dekompozycja równoległa za pomocą opcji *p* bez zmiany sugerowanych rozmiarów bloków. Pierwszy wzięto blok o rozmiarze (16, 2) i otrzymano wydruk:

The H table under consideration has 16 inputs and 2 outputs
inputs: i0 i1 i2 i3 i4 i5 i6 i7 i8 i9 i10 i11 i12 i13 i14 i15
outputs: o3 o4
decomposition parallel block1 (16,1), block2 (13,1)
Serial decomposition is suggested with G function parameters
assumed to be equal to the declared cell parameters
Do you prefer parallel decomposition ?: *p*

OR continue serial decomposition as suggested ?: *s*
OR continue serial decomposition with changed parameters ?: *c*

Z podanego podglądu na dekompozycję równoległą wynika, że rozkładany blok ma 13 wspólnych argumentów. Gdyby istniała dekompozycja szeregowa z funkcją $G(10, 4)$, to funkcja H byłaby blokiem o 10 wejściach i 2 wyjściach (pamięć M4K).
Dla wykonania tej dekompozycji należy użyć opcji ***c***:

no. of block inputs=10
no. of block inputs=4.

Wykona się dekompozycja $G(10, 3)$ ponieważ program szuka zadanej dekompozycji, ale po napotkaniu na dekompozycję lepszą (w sensie liczby wyjść) wykonuje ją automatycznie bez pytania o zgodę. Otrzymana funkcja H to blok o parametrach (9, 2), co stanowi dwie pamięci M512 (należy tylko blok (9, 2) rozłożyć przedtem na dwa bloki jednowyjściowe, aby nie został zapamiętany w pamięci M4K skonfigurowanej jako 9 wejść i 8 wyjść). Do bloku (10, 3) program wróci w odpowiednim momencie i należy go zapamiętać w pamięci M4K za pomocą opcji *r*.

Następuje teraz powrót do rozkład drugiego bloku o parametrach (13, 3).

The H table under consideration has 13 inputs and 3 outputs
inputs: i0 i1 i2 i4 i5 i6 i7 i8 i9 i10 i12 i13 i14
outputs: o0 o1 o2
decomposition parallel block1 (13,1), block2(10,2)
Serial decomposition is suggested with G function parameters
assumed to be equal to the declared cell parameters
OR continue serial decomposition with changed parameters ?: *c*
assumed to be equal to the declared cell parameters
Do you prefer parallel decomposition ?: *p*
OR continue serial decomposition as suggested ?: *s*

Z podanego opisu nie można wywnioskować liczby wspólnych argumentów więc bezpieczniej wykonać dekompozycję równoległą. Opcja *p* zadaje pytanie:

Change block size (*y/n*) ? : *n*

i w odpowiedzi *n* oznacza, że rozmiary bloków nie będą zmieniane. Jeden z otrzymanych bloków o wyjściach $o0$ i $o1$ to blok o parametrach (10, 2) i może być umieszczony w pamięci M4K skonfigurowanej jako 10 wejść i 4 wyjść.

Zostaje jeszcze blok o rozmiarze (13, 1) dla wyjścia $o2$. Ponieważ blok ma 13 argumentów to, aby nie wydłużać procesu obliczeń wykonujemy dla niego dekompozycję szeregową z funkcją np. $G(8, 4)$ za pomocą opcji ***c*** wprowadzając:

no. of block inputs=8
no. of block inputs=4.

i w wyniku wykonanej otrzymujemy blok (9, 1). Taka dekompozycja szeregowa istnieje i blok (8, 4) stanowi dwa bloki M512, a blok (9, 1) jest również pamięcią M512. W rezultacie otrzymujemy**:**

***Rozwiązanie1*: 2 x M4K + 5M512.**

Rozkład ostatniego bloku dla wyjścia *o2* o parametrach (13,1) można zmienić wykonując w dwóch kolejnych krokach dekompozycję szeregową z funkcją G(4,1). W pierwszym kroku wykona się dekompozycja G(4,1). Zmniejszy ona parametry nowego bloku H do 10 wejść i jednego wyjścia. Następna dekompozycja szeregowa z funkcją G(4,1) nie istnieje. Natomiast podgląd na dekompozycję szeregową pokaże, że istnieje dekompozycja z funkcją G(4,2). Można ją wykonać za pomocą opcji ***c***. Po jej wykonaniu pozostanie blok (8,1), który można zapamiętać w pamięci M512 skonfigurowanej jako 8 wejść i 2 wyjścia. Rezultatem jest rozwiązanie:

***Rozwiązanie2*: 2xM4K + 3xM512 + 3xLUT(4,1).**

5.2. Przykład rozkładu funkcji z jednym wyjściem i liczbą argumentów większą od 12

Funkcje o liczbie wejść większej od 12, mogą sprawiać dużą trudność. Nie można podzielić ich na takie bloki, aby w całości mogły być realizowane w pamięciach ROM. Musi być zawsze w takim przypadku wykonana przynajmniej jedna dekompozycja szeregowa. Jako przykład pokazano rozkład jednego bloku, wyselekcjonowanego z filtra sym14, którego wektor współczynników jest w postaci:

[165,64,63802,63283,4644,12353,8626,280,64739,1092,491,65333,65520,43]

Tablica prawdy posiada 14 wejść i 19 wyjść. Przy czym 12 wyjść jest zależnych od wszystkich argumentów, pozostałe od mniejsze liczby argumentów. Wyselekcjonowano blok o 1 wyjściu (o numerze 1) i 14 wejściach. Aby zrealizować go przynajmniej w pamięci M4K(12×1), należy wykonać dekompozycję szeregową taką, aby powstała funkcja *H* miała rozmiar: 12 wejść, 1 wyjście. Wybór strategii dekompozycji jest skromny. Nie ma opcji *r*, ponieważ blok nie może być pamięcią ROM. Nie ma też opcji *p*, ponieważ jest tylko jedno wyście. Wybrano opcję *s*, wykonującą dekompozycję szeregową dla komórek o rozmiarze (4, 1). Wprawdzie nie ma dekompozycji szeregowej z funkcją *G*(4, 1), ale podgląd na dekompozycję pokazuje, że jest z funkcją *G*(4, 2). Wykonujemy ją za pomocą opcji *c*. Po wykonaniu tej dekompozycji, wyświetlany jest opis „nowej" funkcji *H* wraz ze spisem strategii do wykonania.

The H table under consideration has 12 inputs and 1 outputs
inputs: i1 i2 i4 i5 i6 i9 i10 i11 i12 i13 g0_0 g0_1
outputs: o1
Serial decomposition is suggested with G function parameters
assumed to be equal to the declared cell parameters
OR continue serial decomposition as suggested ?: *s*
OR continue serial decomposition with changed parameters ?: *c*
OR save ROM-M4K(12x1) ? : *r*

Opcja *r* zapamiętuje funkcję *H* w pamięci M4K. Otrzymany rozkład jest na dwóch poziomach logicznych bloków, a jego rozwiązanie to:

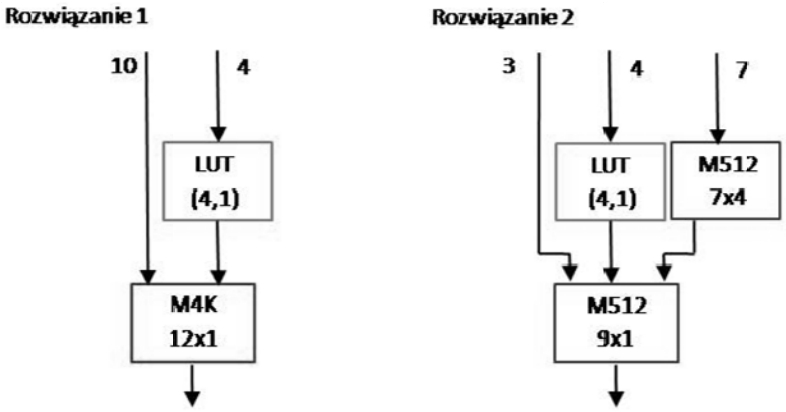
Rozwiazanie1: M4K + 2xLc.

Jeśli nie zatwierdzimy tego rozkładu w pamięci ROM to można sprawdzić, czy bez podwyższania istniejących już dwóch poziomów można zmniejszyć liczbę argumentów funkcji *H*(12, 1). W tym celu wykorzystujemy opcję *c* i wykonujemy kolejno dekompozycje z funkcjami *G*(4, 1), *G*(6, 1), *G*(7, 1), *G*(8, 1). Dla żadnych z nich nie istnieje z zadanymi parametrami, ale podgląd na dekompozycję pokazuje inne możliwości (najlepsze otrzymane rozwiązania). Trzeba tylko je przeanalizować.

1 0 0 0
2 0 0 0 0
3 0 0 0 0 0
4 3 8 T 2 5 6 7
5 0 0 0 0 0 0 0
6 4 10 T 0 1 2 4 8 9 Nd 7
7 4 10 T 0 1 2 3 4 8 9 Nd 8
8 4 9 T 0 1 2 3 4 7 8 9 Nd 8

Łatwo zauważyć, że funkcja *H* ma 10 wejść bezpośrednich i dwa będące wyjściami z poprzednich dekompozycji (w opisie funkcji węzły g0_0 i g0_1). W pierwszej kolejności przeszukiwane są tylko kombinacje wejść bezpośrednich. Jeśli istnieje jakakolwiek dekompozycja o zadanej liczbie wejść i liczbie wyjść większej od ustalonej, to program zapamiętuje ją i nie przeszukuje już kombinacji z wejściami będącymi węzłami pośrednimi. Ma to na celu nie podwyższanie liczby poziomów, nawet kosztem zwiększenia liczby pamięci ROM. Skraca to również czas wykonywania dekompozycji. Np. w tym przypadku dla funkcji *G* z 6 argumentami skraca czas przeszło czterokrotnie.
Dekompozycja z *G*(4, 3) zmniejszy nam liczbę wejść do 11 i **wynik:M4K + 5xLc.**
Dekompozycja z *G*(6, 4) zmniejszy liczbę wejść do 10. Funkcja H nadal będzie pamięcią M4K i dojdzie dodatkowa pamięć M512. **Wynik: M4K + M512 + 2xLc**.
Dekompozycja z *G*(7, 4) zmniejszy liczę wejść do 9. Funkcja H stanie się pamięcią M512. **Wynik: 2xM512 + 2xLc.**
Dekompozycja z *G*(8, 4) zmniejszy liczbę wejść do 8. Funkcja H stanie się pamięcią M512, ale funkcja *G* to dwie pamięci M512. **Wynik: 3xM512 + 2xLc.**
Spośród nich najkorzystniejszy jest wybór dekompozycji z funkcją *G*(7, 4). Przyjmujemy zatem jako **Rozwiązanie2: 2xM512 + 2xLc**.

Analiza dwóch rozwiązań pokazuje, że pojemność pamięci ROM użyta do drugiego rozwiązania jest czterokrotnie mniejsza niż analogiczna dla rozwiązania pierwszego (rys. 3).



Rys. 3. Rozkład funkcji dla wybranych najlepszych rozwiązań

Fig. 3. Decomposition of function for the selected best solutions

6. Wnioski

Eksperymenty przeprowadzono dla kilku benchmarków pod kątem stosowania różnych technik dla pamięci ROM. W rozwiązaniach stosowano: jeden typ pamięci ROM i ewentualnie bloki LUT(1,4), albo techniki mieszane (oba typy pamięci ROM i ewentualnie bloki LUT(1,4)).

Tabela 3

Wyniki dekompozycji uzyskane za pomocą Decomposer-Demain-ROM

	L. wejść	L. wyjść	M512	M4K	M512+M4K	LUT
Alu1	12	8	4/2	8/1		36/7
Alu4	14	8	25/3	10/2	(10+4)/2	535/25
Br1	12	8	4/2	8/1		54/5
F51m	8	8	2/1	1/1		20/4
5xp1	7	10	2/1	1/1		25/3
newapla	12	10	(2xLUT+4)/2	4/1	(2+2)/1	25/4

W tabeli 3 umieszczono rozwiązania, dla których przyjęto następujące kryterium optymalizacji: w ierwszej kolejności minimalizację liczby poziomów bloków logicznych (wyniki umieszczone po /), a dopiero w drugiej kolejności minimalizację zasobów pamięci. W związku z tym, niektóre pozycje przedostatniej kolumny pozostały puste, gdyż rozwiązania te podwyższyłyby liczbę poziomów kosztem zmniejszenia zasobów pamięci lub przy zachowaniu optymalnego poziomu zwiększyłyby zasoby pamięci. Np. dla funkcji br1 otrzymano rozwiązania:

r1=4xM512/2,
r2=(5xLc +1x4K)/2,
r3=(1xM512 + 1x4K)/2,
r4=8x4K/1.

Dla rozwiązania r1 realizowanego na dwóch poziomach logicznych pojemność pamięci jest prawie o połowę mniejsza niż dla analogicznych rozwiązań r3 i r4. Nie istnieje żadne inne lepsze rozwiązanie od r1, realizowane na dwóch poziomach bloków logicznych. Nie istnieje oprócz rozwiązania r4 żadne inne rozwiązanie realizowane na jednym poziomie logicznych bloków. Ostatnia kolumna dla funkcji br1 jest więc pusta. Natomiast dla funkcji newapla ostatnia kolumna zawiera najlepsze rozwiązanie realizowane na jednym poziomie logicznym, do którego użyto techniki mieszanej.

Przeprowadzone eksperymenty pokazują, że dla realizacji układów kombinacyjnych w strukturach FPGA z wbudowanymi blokami pamięci najodpowiedniejsza wydaje się taka strategia, która pozwala na w miarę optymalne wykorzystanie zasobów bloków. Wybór pamięci typu ROM powinien być dobierany do rozmiaru aktualnego bloku. Jeśli spełnia on parametry dla odpowiednio skonfigurowanej pamięci do bloku M512 to powinien być w niej realizowany. Nie powinno się na „siłę" realizować bloku w pamięci 4K jeśli można go swobodnie zrealizować w pamięci M512. Tym bardziej, że blok pamięci 4K ma pojemność 8 razy większą. To samo dotyczy Lutów. Jeśli po całkowitym wypełnieniu bloków pamięci (M4K lub/i M512) zostaje blok (4,1), to powinien pozostać jako LUT (4,1).

W tabeli 3 umieszczono również wyniki rozkładu funkcji na komórki LUT(4,1)[2]. W większości przypadków (np. dla alu4 to 535 komórek LUT i 25 poziomów logicznych) struktury sieci komórek opisujących funkcje są dość skomplikowane i to nie tylko ze względu na liczbę otrzymanych komórek z rozkładu, ale również ze względu na liczbę poziomów logicznych komórek i połączeń między nimi. Wpływa to negatywnie na niezawodność i szybkość takiego układu. Natomiast realizacja funkcji boolowskich w pamięciach ROM posiada o wiele prostszą strukturę logiczną. I właśnie pojawienie się najnowszych układów programowalnych zawierających wbudowane bloki pamięci ROM dają duże możliwości projektowe i powinny być w pełni wykorzystane przy realizacjach złożonych układów kombinacyjnych.

Artykuł nie ocenia skuteczności proponowanych metod poprzez porównanie ich z innymi implementacjami realizowanymi w różnych typach LUT i różnych typach układów. Nie podawane są więc zarówno parametry czasowe i wykorzystanie zasobów. Artykuł prezentuje jedynie metody umożliwiające realizacje funkcji w strukturach programowalnych z wbudowanymi blokami pamięci (rozdz. 2, 3). Omawiany system Decomposer--Demain-ROM pokazuje strategie dające różne rozwiązania. Nie podaje złotego środka prowadzącego do optymalnego rozwiązania. Każdy projekt może mieć inne wymagania i inne kryteria optymalizacji. Przy użyciu systemu Demain-ROM można uzyskać dla każdej funkcji wiele rozwiązań. Użytkownik wybiera rozwiązanie najbardziej zbliżone do swoich potrzeb.

Aktualnie w ogólnodostępnej literaturze nie są prezentowane rozwiązania realizacji funkcji boolowskich w układach FPGA z wykorzystaniem pamięci wbudowanych ROM.

[2] W tabeli 3 umieszczono na końcu także rozwiązania dla komórek LUT uzyskanych z tego samego systemu Demain-ROM drogą optymalizacji liczby poziomów, aby zwrócić uwagę na złożoność otrzymywanych struktur z czego użytkownik często nie zdaje sobie sprawy.

Literatura

[1] Ojrzeńska-Wójter D., Jasiński K., *Układy FPGA. Możliwości powszechnego zastosowania*, Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne, 2-3, 2008.
[2] Łuba T., *Synteza układów logicznych*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2005.
[3] Nowicka M., Tomaszewicz P., Zbierzchowski B., *Arytmetyka rozproszona w syntezie filtrów cyfrowych*, Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne, 1, 2006.
[4] Nowicka M., Łuba T., Rawski M., *FPGA-Based Decomposition of Boolean Functions. Algorithms and Implementation*, Sixth International Conference on Advanced Computer Systems, Szczecin 1999.
[5] Tomaszewicz P., Nowicka M., Falkowski B. J., Łuba T.: *Logic Synthesis Importance in FPGA-Base Designing of Image Signal Processing Systems*, [in:] proceedings of the 14th International Conference on Mixed Design of Integrated circuits and Systems (MIXDES 2007), Ciechocinek, Poland 2007, 141-146.
[6] Łuba T., Selvaraj H., Nowicka M., Kraśniewski A., *Balanced multilevel decomposition and its applications in FPGA-based syntehesis*, [in:] Saucier G., Mignotte A. (ed.), Logic and Architecture Synthesis, Chapman&Hall, 1995.