ZESZYTY
NAUKOWE
POLITECHNIKI
ŚLĄSKIEJ

P.4474/04

Dariusz KANIA

SYNTEZA LOGICZNA PRZEZNACZONA DLA MATRYCOWYCH STRUKTUR PROGRAMOWALNYCH TYPU PAL

Gliwice 2004

POLITECHNIKA ŚLĄSKA
ZESZYTY NAUKOWE
Nr 1619

P. 4474/04

Dariusz KANIA

SYNTEZA LOGICZNA PRZEZNACZONA DLA MATRYCOWYCH STRUKTUR PROGRAMOWALNYCH TYPU PAL

Gliwice 2004

Opiniodawcy

Prof. dr hab. inż. Marian ADAMSKI
Prof. dr hab. Tadeusz ŁUBA

Kolegium redakcyjne

REDAKTOR NACZELNY — Prof. dr hab. inż. Andrzej BUCHACZ
REDAKTOR DZIAŁU — Doc. dr inż. Zdzisław POGODA
SEKRETARZ REDAKCJI — Mgr Elżbieta LEŚKO

Redakcja

Mgr Kazimiera SZAFIR

Redakcja techniczna

Alicja NOWACKA

PL ISSN 1231-1596

© Copyright by
Dariusz KANIA
Gliwice 2004

Od autora

Pierwsza część pracy stanowi podsumowanie zadań realizowanych w ramach projektu badawczego nr 4T11B 060 22 pt.: „Synteza logiczna układów cyfrowych dedykowana dla macierzowych struktur programowalnych". W kolejnej części przedstawiono rozszerzenie teorii dekompozycji funkcjonalnej. Podsumowaniem całości są, zawarte w końcowej części pracy, złożone strategie syntezy.

Recenzentom, prof. dr. hab. inż. Marianowi Adamskiemu oraz prof. dr. hab. Tadeuszowi Łubie składam serdeczne podziękowania za wiele cennych uwag, które wpłynęły na ostateczną postać niniejszej pracy.

Prof. dr. hab. inż. Edwardowi Hrynkiewiczowi dziękuję za słowa zachęty, stworzenie możliwości realizacji tej pracy oraz szereg cennych uwag.

Żonie Krystynie oraz wszystkim bliskim dziękuję za pomoc i wyrozumiałość.

Autor

Gliwice, marzec 2004

SPIS TREŚCI

Wykaz ważniejszych oznaczeń . . . 9
Wykaz ważniejszych skrótów . . . 13
Wykaz skrótów określających metody oraz strategie syntezy . . . 14

1. Wprowadzenie . . . 15

1.1. Elementy syntezy logicznej układów cyfrowych przeznaczone dla struktur programowalnych . 18
1.2. Tematyka pracy . . . 22
1.3. Stosowana terminologia . . . 23

2. Metody realizacji wielowyjściowych funkcji w strukturach typu PAL wykorzystujące grafy wyjść . . . 24

2.1. Klasyczna metoda realizacji wielowyjściowych funkcji w strukturach typu PAL . . . 24
2.2. Reprezentacja wielowyjściowej funkcji w postaci skierowanego grafu wyjść . . . 28
2.3. Idea implementacji funkcji wielowyjściowej wykorzystująca grafy wyjść . . . 31
2.4. Twierdzenie o wyborze wierzchołków grafu wyjść . . . 33
2.5. Algorytmy implementacji wielowyjściowych funkcji w strukturach typu PAL . . . 38
2.5.1. Algorytm implementacji wielowyjściowych funkcji w strukturach typu PAL (optymalizacja liczby bloków logicznych); metoda W_LB . . . 38
2.5.2. Algorytm implementacji wielowyjściowych funkcji w strukturach typu PAL (optymalizacja liczby warstw logicznych); metoda W_LW . . . 45
2.6. Wyniki eksperymentów . . . 48

3. Metody realizacji funkcji logicznych w strukturach typu PAL z trójstanowymi buforami wyjściowymi . . . 58

3.1. Idea wykorzystania trójstanowych buforów wyjściowych . . . 58
3.2. Reprezentacja sposobu implementacji funkcji w postaci grafu podziału . . . 60
3.3. Minimalizacja z rozłączaniem implikantów . . . 67
3.4. Koncepcja implementacji funkcji w strukturach typu PAL z trójstanowymi buforami wyjściowymi . . . 70
3.4.1. Metoda wyszukiwania zmiennej podziału, wykorzystywana w *p*-warstwowej implementacji funkcji w strukturach typu PAL z trójstanowymi buforami wyjściowymi . . . 71
3.4.2. Algorytm *p*-warstwowej implementacji funkcji na blokach logicznych typu PAL z trójstanowymi buforami wyjściowymi o zadanej liczbie iloczynów (metoda B_PW) . . . 75
3.4.3. Metoda wyszukiwania kostek podziału, wykorzystywana w jednowarstwowej implementacji funkcji w strukturach typu PAL z trójstanowymi buforami wyjściowymi . . . 76
3.4.4. Algorytm jednowarstwowej implementacji funkcji na blokach logicznych typu PAL z trójstanowymi buforami wyjściowymi o zadanej liczbie iloczynów (metoda B_JW) . . . 78
3.5. Wyniki eksperymentów . . . 81

4. Metody realizacji funkcji w strukturach typu PAL bazujące na dekompozycji . . .90

4.1. Znaczenie dekompozycji w syntezie przeznaczonej dla układów programowalnych 90
4.2. Istota dekompozycji przeznaczonej dla struktur typu PAL 93
4.3. Kodowanie równomierne 96
4.4. Dekompozycja kolumnowa zorientowana na realizację w strukturach typu PAL 105
4.4.1. Metody wyznaczania złożoności kolumnowej zorientowane na realizację układów w strukturach typu PAL 105
4.4.2. Metoda kodowania wzorców kolumn zorientowana na realizację układów w strukturach typu PAL 109
4.4.3. Algorytm implementacji funkcji jednowyjściowych w strukturach typu PAL oparty na dekompozycji kolumnowej 117
4.4.4. Dekompozycja kolumnowa funkcji wielowyjściowych 121
4.4.5. Wyniki eksperymentów. 127
4.5. Dekompozycja wierszowa zorientowana na realizację w strukturach typu PAL 130
4.5.1. Metoda wyznaczania złożoności wierszowej 134
4.5.2. Zagadnienia związane w wyborem dekompozycji wierszowej 137
4.5.3. Algorytm implementacji funkcji w strukturach typu PAL oparty na dekompozycji wierszowej 141
4.5.4. Wyniki eksperymentów 146
4.6. Porównanie zaproponowanych modeli dekompozycji zorientowanych na realizację funkcji w strukturach typu PAL 149

5. Złożone strategie syntezy funkcji w strukturach typu PAL 155

5.1. Dobór elementów syntezy pod kątem optymalizacji liczby bloków logicznych lub liczby warstw logicznych. 155
5.2. Strategia syntezy dla struktur typu PAL bez trójstanowych buforów wyjściowych, (optymalizacja liczby bloków logicznych); strategia W_DK-W_LB 156
5.3. Strategia syntezy dla struktur typu PAL z trójstanowymi buforami wyjściowymi, (optymalizacja liczby bloków logicznych); strategia W_DK-W_LB-B_JPW 158
5.4. Strategia syntezy dla struktur typu PAL bez trójstanowych buforów wyjściowych, (optymalizacja liczby warstw logicznych; strategia J_DW-W_LW 159
5.5. Strategia syntezy dla struktur typu PAL z trójstanowymi buforami wyjściowymi, (optymalizacja liczby warstw logicznych); strategia B_JPW 160
5.6. Porównanie zaproponowanych strategii syntezy z innymi metodami 161
5.6.1. Proste układy typu PAL (ABEL, MACHxl, ispDesignEXPERT, Synplify, ASYL) . .162
5.6.2. Układy Classic firmy Altera (MAX+PLUS) 167
5.6.3. Układy MAX firmy Altera (MAX+PLUS) 167
5.6.4. Układy MACH (Synplify+MACHxl) 176
5.6.5. Układy FLASH firmy Cypress (WARP) 178
5.6.6. Systemy akademickie przeznaczone dla matryc PLA 180
5.6.7. Analiza końcowa wyników eksperymentów 181

6. Podsumowanie 184

Literatura 187
Streszczenie 201
Dodatki 203

CONTENTS

List of more important symbols 9
List of more important abbreviations 13
List of abbreviations used for defining methods and strategies of the synthesis 14

1. Introduction 15

1.1. Components on logic synthesis of digital circuits applicable to programmable structures 18
1.2. Scope of work 22
1.3. Applied terminology 23

2. Methods for realization of multi-output functions within PAL-based structures based on graphs of outputs 24

2.1. Classical method for realization of multi-output function within PAL-based structures 24
2.2. Representation of a multi-output function in the form of a directed graph of outputs 28
2.3. The idea related to implementation of a multi-output function based on graph of outputs 31
2.4. The theorem on choosing a node of a graph of outputs 33
2.5. Algorithms for implementation of multi-output functions within PAL-based structures 38
2.5.1. The algorithm for implementation of multi-output functions within PAL-based structures (optimization of logic blocks); the W_LB method 38
2.5.2. The algorithm for implementation of multi-output functions within PAL-based structures (optimization of logic stages); the W_LW method 45
2.6. Results of experiments 48

3. Methods for realization of multi-output functions within PAL-based structures, which contain three-state output buffers 58

3.1. The concept of taking advantage of three-state output buffers 58
3.2. Presentation of the concept how to implement a function by means of the graph of divisions 60
3.3. Minimization that involves splitting of implicants 67
3.4. The concept related to implementation of a Boolean function within PAL-based structures, which contain three-state output buffers 70
3.4.1. The method involving searching for the partitioning variable, applicable for the *p*-stage implementation of a function within PAL-based structures, which contain three-state output buffers 71
3.4.2. The algorithm for the *p*-stage implementation of a Boolean function within PAL-based logic blocks that contain three-state output buffers and constrained by certain number of terms (the B-PW method) 75
3.4.3. The method involving searching for the partitioning cube, applicable for the single-stage implementation of a function within PAL-based structures, which contain three-state output buffers 76
3.4.4. The algorithm for the single-stage implementation of a Boolean function within PAL-based logic blocks that contain three-state output buffers and constrained by certain number of terms (the B-JW method) 80
3.5. Results of experiments 81

4. Methods for realization of a Boolean function within PAL-based structures, involving the approach of decomposition 90

4.1. Importance of decomposition for synthesis methods designed for programmable circuits 90
4.2. Essence of decomposition approach dedicated to PAL-based structures 93
4.3. Even coding 96
4.4. The column decomposition approach oriented towards realization within PAL-based structures 105
4.4.1. Methods of determining the column complexity oriented towards realization of digital circuits within PAL-based structures 105
4.4.2. The method for coding patterns of columns, oriented towards realization of digital circuits within PAL-based structures 109
4.4.3. The algorithms for implementation of a single-output Boolean function within PAL-based structures that involves column decomposition approach 117
4.4.4. The column decomposition of a multi-output Boolean function 121
4.4.5. Results of experiments 127
4.5. The row decomposition approach oriented towards realization within PAL-based structures 130
4.5.1. Methods of determining the row complexity 134
4.5.2. Problems connected with choice of the row decomposition 137
4.5.3. The algorithms for implementation of a Boolean function within PAL-based structures that involves row decomposition approach 141
4.5.4. Results of experiments 146
4.6. Comparison of the proposed decomposition models, which are oriented towards realization of a Boolean function within PAL-based structures 149

5. Complex strategies for synthesis a Boolean function within PAL-based structures 155

5.1. Selection of components of the synthesis procedure from the point of optimization of either number of logic blocks or number of logic stages 155
5.2. The strategy of synthesis applicable to PAL-based structures without three-state output buffers; (optimization from the point of number of logic blocks); (the W_DK-W_LB strategy) 156
5.3. The strategy of synthesis applicable to PAL-based structures with three-state output buffers, (optimization from the point of number of logic blocks); the W_DK-W_LB-B-JWP strategy 158
5.4. The strategy of synthesis applicable to PAL-based structures without three-state output buffers, (optimization from the point of number of logic stages); the J_DW-W_LW strategy 159
5.5. The strategy of synthesis applicable to PAL-based structures with three-state output buffers, (optimization from the point of number of logic stages); the B-JWP strategy 160
5.6. Comparison of proposed strategies of synthesis with other methods 161
5.6.1. Simple PAL circuits (ABEL, MACHxl, ispDesignEXPERT, Synplify, ASYL) 162
5.6.2. The Classic circuits from Altera (MAX+PLUS) 167
5.6.3. The MAX circuits from Altera (MAX+PLUS) 167
5.6.4. The MACH circuits (Synplify+MACHxl) 176
5.6.5. The FLASH circuits from Cypress (WARP) 178
5.6.6. The university-developed systems designed for PLA matrices 180
5.6.7. Final analysis of experimental results 181

6. Conclusions 184

References 187
Abstract 201
Supplements 203

WYKAZ WAŻNIEJSZYCH OZNACZEŃ

k - liczba iloczynów zawartych w bloku logicznym typu PAL
n - liczba argumentów funkcji; liczba wejść układu logicznego

m - liczba elementów wyjściowych wielowyjściowej funkcji; liczba wyjść układu logicznego

δ_{f_i} - liczba bloków logicznych typu PAL potrzebnych do realizacji funkcji $f_i : B^n \to B$

δ_f - liczba bloków logicznych typu PAL potrzebnych do realizacji funkcji $f: B^n \to B^m$

δ_f^1 - liczba bloków logicznych typu PAL potrzebnych do realizacji funkcji $f: B^n \to B^m$ metodą klasyczną; $\delta_f^1 = \sum_{i=1}^{m} \delta_{f_i}$

${}^i\delta_f$ - wartość parametru δ_f w i-tym kroku

ξ_{f_i} - liczba warstw logicznych układu realizującego funkcję $f_i : B^n \to B^1$

ξ_f^1 - liczba warstw logicznych układu realizującego funkcję $f : B^n \to B^m$ metodą klasyczną; $\xi_f^1 = \max(\xi_{f_1}, \xi_{f_2}, \ldots, \xi_{f_m})$

y - m-elementowy wektor wyjściowy, odpowiadający części wyjściowej wielowyjściowego implikantu; np.: jeżeli m=4, to przykładowy wektor wyjściowy y=0110

Δ_{f_i} - liczba implikantów funkcji $f_i : B^n \to B^1$

Δ_y - liczba wektorów y występujących w zbiorze wielowyjściowych implikantów funkcji $f: B^n \to B^m$ skojarzona z wierzchołkiem grafu wyjść, np.: Δ_{1110}

Δ_j^m - suma wyróżników $\Delta_{\_\ \_\ \_\ 1\ \_\ \_\ \_}$ wierzchołków grafu wyjść pokrywanych przez drogi rozpoczynające się w wierzchołku $\Delta_{0\ \_\ 0\ 1\ 0\ \ldots 0\,0}$, a kończące się w wierzchołkach wyższych rzędów

${}^i\Delta_j^m$ - wartość parametru Δ_j^m w i-tym kroku

${}^i\Delta_y$ - wybrany w i-tym kroku wierzchołek grafu wyjść skojarzony z wyróżnikiem Δ_y

Δ_{FB} - wyróżnik reprezentujący sprzężenia zwrotne

$\mu(\Delta_y)$ - rząd wyróżnika Δ_y ; liczba elementów {1} zawartych w wektorze y

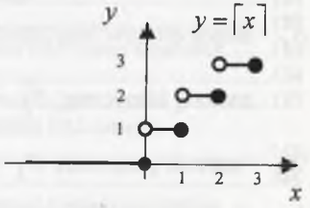
μ - skrót oznaczenia $\mu(\Delta_y)$ stosowany na rysunkach

r_j^m - reszta z dzielenia $\Delta_j^m - 1$ przez $(k-1)$; $\Delta_j^m - 1 \equiv r_j^m \,(\mathrm{mod}(k-1))$

${}^i r_j^m$ - wartość parametru r_j^m w i-tym kroku

${}^i\gamma$ - liczba bloków logicznych typu PAL, wykorzystanych do realizacji implikantów w i-tym kroku; ${}^i\gamma = \left\lceil \frac{{}^i\Delta_y - k}{k-1} \right\rceil + 1$

$\lceil x \rceil$ - najmniejsza wartość naturalna (z włączeniem zera) nie mniejsza od x



$\mathbf{R}$ - zbiór reszt r_j^m ; $\mathbf{R} = \{r_j^m ;\ j \in <1, m>\}$

$r_a^{\mu({}^i\Delta_y)}$, $r_b^{\mu({}^i\Delta_y)}$, $r_x^{\mu({}^i\Delta_y)}$ - reszta należąca do zbioru reszt

$$\mathbf{R} = \{r_j^{\mu({}^i\Delta_y)} ;\ j \in <1, \mu({}^i\Delta_y)>\}$$

$I=(i_n,...,i_2,i_1)$ - wektor wejściowy

I_j - stan wejściowy; n-elementowy ciąg wartości 0 lub 1 odpowiadających zmiennym wejściowym $i_n,...,i_2,i_1$, $(i=0,1,...,2^n\text{-}1)$

$A_y = (a_n,...,a_2,a_1)$ - kostka, dla której funkcja y przyjmuje wartość 1; część wejściowa jednowyjściowych implikantów; $a_i \in \mathbf{A}_{0,1,-} = \{0,1,-\}$, $(i=1,2,...,n)$

$\mathbf{A}_f$ - zbiór części wejściowych jednowyjściowych implikantów opisujących funkcję f; zbiór kostek, dla których funkcja przyjmuje wartość 1

$f_{\tilde{i}_j}$ - element rozwinięcia Shannona funkcji $f(i_n,...,i_{j+1},i_j,i_{j-1},...,i_1,i_0)$ równy

$f(i_n,...,i_{j+1},0,i_{j-1},...,i_1,i_0)$ dla $\tilde{i}_j = \bar{i}_j$ lub $f(i_n,...,i_{j+1},1,i_{j-1},...,i_1,i_0)$ dla $\tilde{i}_j = i_j$

$\tilde{x}$ - literał zmiennej x; $\tilde{x} = x$ lub $\bar{x}$

f^x - funkcja opisująca zachowanie wyjścia bloku logicznego typu PAL z trójstanowym buforem wyjściowym sterowanym przez zmienną x

$$f^x = \begin{cases} f_x & dla\ x = 1 \\ stan\ wysokiej\ impedancji & dla\ x = 0 \end{cases}$$

$X = (x_p,...,x_2,x_1)$ - kostka podziału, $x_i \in \mathbf{X}_{0,1} = \{0,1\}$, $(i=1,2,...,p)$

$< a_i, x_i >$ - para uporządkowana elementów $a_i \in \mathbf{A}_{0,1,-} = \{0,1,-\}$ oraz $x_i \in \mathbf{X}_{0,1} = \{0,1\}$

$\mathbf{A}_{f_X}$ - zbiór kostek niesprzecznych z kostką podziału X (*definicja* 3.1)

$\mathbf{A}_{f_{\tilde{x}}}$ - zbiór kostek niesprzecznych z kostką podziału zredukową do pojedynczej zmiennej x (*definicja* 3.1)

${}^*\mathbf{A}_{f_X}$ - zbiór kostek nierównoważnych z kostką podziału X (*definicja* 3.2)

${}^*\mathbf{A}_{f_{\tilde{x}}}$ - zbiór kostek nierównoważnych z kostką podziału zredukowaną do pojedynczej zmiennej x (*definicja* 3.2)

$\overline{\overline{\mathbf{A}}}$ - moc zbioru $\mathbf{A}$; liczba elementów zbioru $\mathbf{A}$

Δ_{f_X} - liczba dziesiętna równa mocy zbioru $\mathbf{A}_{f_X}$; $\Delta_{f_X} = \overline{\overline{\mathbf{A}_{f_X}}}$

${}^*\Delta_{f_X}$ - liczba dziesiętna równa mocy zbioru ${}^*\mathbf{A}_{f_X}$; ${}^*\Delta_{f_X} = \overline{\overline{{}^*\mathbf{A}_{f_X}}}$

$\mathbf{I}_{A_y}$ - zbiór stanów wejściowych I_j, pokrytych przez kostkę $A_y = (a_n,...,a_2,a_1)$

$a_i \in \mathbf{A}_{0,1,-} = \{0,1,-\}$; $i=1,2,...,n$; $(\overline{\overline{\mathbf{I}_{A_y}}} = 2^r)$, gdzie r jest liczbą elementów a_i= -

$\delta_{f_{\tilde{x}}}$ - liczba bloków logicznych typu PAL potrzebnych do realizacji funkcji $f_{\tilde{x}}$

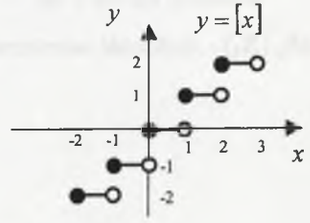
$\xi_{f_{\tilde{x}}}$ - liczba warstw logicznych układu realizującego funkcję $f_{\tilde{x}}$

$\Delta_{f_{\tilde{x}}}$ - liczba dziesiętna równa mocy zbioru $\mathbf{A}_{f_{\tilde{x}}}$

δ_{f_X} - liczba bloków logicznych typu PAL potrzebnych do realizacji implikantów (kostek) należących do zbioru $\mathbf{A}_{f_X}$

x==x+1 – x staje się x+1

$\lfloor x \rfloor$ - największa wartość całkowita nie większa od x; (część całkowita liczby x)



$\mathbf{X}_1$ - zbiór związany

$\mathbf{X}_2$ - zbiór wolny

$\nu(\mathbf{X}_2 \mid \mathbf{X}_1)$ - złożoność (krotność) kolumnowa siatki Karnaugha (matrycy podziałów)

$f(\mathbf{X}_2,\mathbf{X}_1)$ - funkcja opisana siatką Karnaugha, w której $\mathbf{X}_2$ jest zbiorem zmiennych skojarzonych z wierszami, a $\mathbf{X}_1$ – zbiorem zmiennych skojarzonych z kolumnami siatki

$w_k(i_{S-1},..., i_{r+1}, i_r)$ - współczynnik komplikacji; liczba różnych słów kodowych utworzonych przez zmienne $i_{S-1},..., i_{r+1}, i_r$.

n_t - liczba wejść transkodera

m_t - liczba wyjść transkodera

k_t - liczba możliwych do utworzenia w programowalnym transkoderze wyjściowych słów kodowych

$\gamma(G)$- liczba chromatyczna

η_k - kolumnowy współczynnik kodowy

η_A - współczynnik pokrycia wzorca A

η_{AB} - współczynnik pokrycia pary wzorców (A,B)

η_s - współczynnik sąsiedztwa wzorców

H_A - liczba jedynek występujących w słowie kodowym przyporządkowanym wzorcowi A

z - współczynnik złożoności bloku związanego (*definicja* 4.4)

w - współczynnik złożoności bloku wolnego (*definicja* 4.5)

δ_z - liczba bloków logicznych typu PAL potrzebnych do realizacji bloku związanego

δ_w - liczba bloków logicznych typu PAL potrzebnych do realizacji bloku wolnego

δ_{zw} - liczba bloków logicznych typu PAL potrzebnych do realizacji bloku związanego i wolnego; $\delta_{zw}=\delta_z + \delta_w$

$\mu(\mathbf{X}_2 \mid \mathbf{X}_1)$ - złożoność wierszowa siatki Karnaugha (matrycy podziałów)

WYKAZ WAŻNIEJSZYCH SKRÓTÓW

AHDL - Altera Hardware Description Language

ASIC - Application Specific Integrated Circuit

BDD - Binary Decision Diagram

CLB - Configurable Logic Block

CPLD - Complex Programmable Logic Device

FPGA - Field Programmable Gate Array

FPLA - Field Programmable Logic Array

GAL - Generic Array Logic

HDL - Hardware Description Language

MACH - Macro Array CMOS High-density

MAPL - Multiple Array Programmable Logic

MAX - Multiple Array MatriX

pLSI - Lattice programmable Large Scale Integration

PAL - Programmable Array Logic

PIA - Programmable Interconnect Area

PLA - Programmable Logic Array

PLD - Programmable Logic Device

PLE - Programmable Logic Element

PTP - Pierwotna Tablica Przepaleń

PROM - Programmable Read Only Memory

SPLD - Simple Programmable Logic Device

VHDL - Very high speed integrated circuit Hardware Description Language

WYKAZ SKRÓTÓW OKREŚLAJĄCYCH METODY ORAZ STRATEGIE SYNTEZY

metoda W_LW - metoda realizacji funkcji Wielowyjściowej wykorzystująca grafy wyjść (optymalizacja Liczby Warstw logicznych)

metoda W_LB - metoda realizacji funkcji Wielowyjściowej wykorzystująca grafy wyjść (optymalizacja Liczby Bloków logicznych)

metoda B_PW - metoda *P* - Warstwowej realizacji funkcji wykorzystująca trójstanowe Bufory wyjściowe

metoda B_JW - metoda JednoWarstwowej realizacji funkcji wykorzystująca trójstanowe Bufory wyjściowe

metoda K_LW - Klasyczna metoda realizacji funkcji w strukturach typu PAL (optymalizacja Liczby Warstw logicznych)

metoda J_DW - metoda realizacji funkcji Jednowyjściowej w strukturach typu PAL wykorzystująca Dekompozycję Wierszową

metoda J_DK - metoda realizacji Jednowyjściowej funkcji w strukturach typu PAL wykorzystująca Dekompozycję Kolumnową

metoda W_DK - metoda realizacji Wielowyjściowej funkcji w strukturach typu PAL wykorzystująca Dekompozycję Kolumnową

strategia B_JPW (optymalizacja liczby warstw logicznych)
strategia syntezy przeznaczona dla struktur typu PAL z trójstanowymi buforami wyjściowymi; (metoda B_JW; w przypadku braku rozwiązania metoda B_PW zapewniająca minimalną liczbę warstw)

strategia W_DK-W_LB (optymalizacja liczby bloków logicznych)
strategia syntezy przeznaczona dla struktur typu PAL bez trójstanowych buforów wyjściowych, oparta na realizacji Wielowyjściowej funkcji wykorzystującej Dekompozycję Kolumnową i metodzie **W_LB**, użytej do realizacji bloku związanego i wolnego

strategia W_DK-W_LB-B_JPW (optymalizacja liczby bloków logicznych)
strategia syntezy przeznaczona dla struktur typu PAL z trójstanowymi buforami wyjściowymi, oparta na realizacji Wielowyjściowej funkcji wykorzystującej Dekompozycję Kolumnową i metodzie **W_LW**, użytej do realizacji bloku związanego i wolnego, uzupełniona strategią **B_JPW**, prowadzącą do poprawy właściwości dynamicznych uzyskiwanych układów

strategia J_DW-W_LW (optymalizacja liczby warstw logicznych)
strategia syntezy przeznaczona dla struktur typu PAL bez trójstanowych buforów wyjściowych, oparta na realizacji funkcji Jednowyjściowej wykorzystującej Dekompozycję Wierszową i metodzie **W_LW**, użytej do realizacji bloku związanego i wolnego

1. WPROWADZENIE

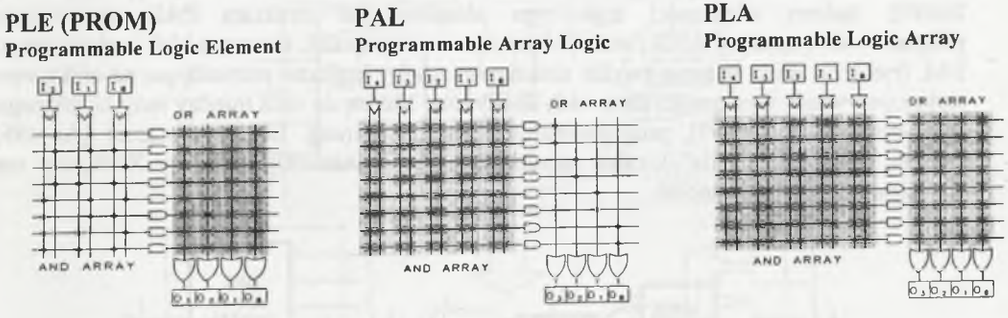
Rozwój technologiczny w sposób do niedawna niewyobrażalny zmienia w ostatnim czasie oblicze przemysłu elektronicznego. Urządzenia elektroniczne są o wiele bardziej złożone, mają coraz większe funkcjonalne możliwości, a równocześnie stają się coraz mniejsze. Rozwój ten jest możliwy dzięki opracowaniu i ciągłemu doskonaleniu komputerowych narzędzi wspomagających projektowanie.

W ostatnich latach najdynamiczniej rozwijającą się formą implementacji układów cyfrowych, zmieniających oblicze urządzeń elektronicznych, stały się układy specjalizowane (ang. ASIC - Application Specific Integrated Circuit) projektowane lub dostosowywane do indywidualnych potrzeb użytkownika. Jedną z głównych grup tych układów są struktury programowalne (**PLD** - Programmable Logic Devices). Układy te, produkowane seryjnie, są dostosowywane do potrzeb projektanta poprzez ich odpowiednie zaprogramowanie.

Śledząc rozwój układów programowalnych oraz prognozy ich rozwoju w przyszłości, można z całą pewnością stwierdzić, że najbliższe lata będą erą układów programowalnych. Rosnąca dostępność coraz lepszych, jednocześnie tańszych komputerowych systemów wspomagających projektowanie powoduje upowszechnienie tej formy implementacji układów cyfrowych dla szerokiego kręgu projektantów. Struktury programowalne dzielą się na trzy podstawowe grupy [Shar98]:

1. Proste układy PLD (ang. SPLD - Simple PLD)
2. Złożone układy matrycowe (ang. CPLD – Complex PLD)
3. Układy FPGA (ang. Field Programmable Gate Array), do których należą między innymi:
   - Struktury tablicowe (ang. Look up Table)
   - Struktury multiplekserowe [Acte95].

Większość struktur programowalnych wykorzystuje jedną z trzech konfiguracji matryc AND-OR przedstawionych na rys. 1.1.



Rys. 1.1. Podstawowe konfiguracje matryc AND-OR
Fig. 1.1. Basic configurations of AND-OR matrices

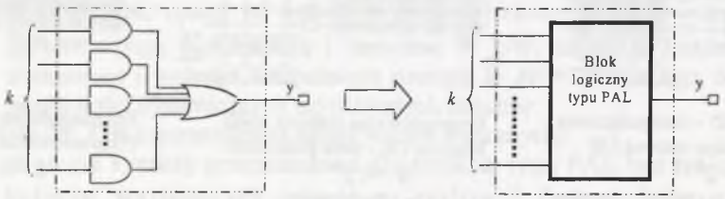
Struktury przedstawione na rys. 1.1 stanowią podstawę większości prostych układów PLD. Największą popularność zdobyły struktury PAL, składające się z odrębnych bloków logicznych typu PAL (rys. 1.2), wyposażonych w dodatkowe elementy, między innymi takie jak [Adva96, Latt94, Sign92]:

- sterowane trójstanowe bufory wyjściowe - wyprowadzenia wejścia/wyjścia,
- mechanizm programowania aktywnego poziomu wyjściowego,
- przerzutniki (najczęściej typu D),
- mechanizm ustawiania i/lub zerowania przerzutników,
- bramki XOR.

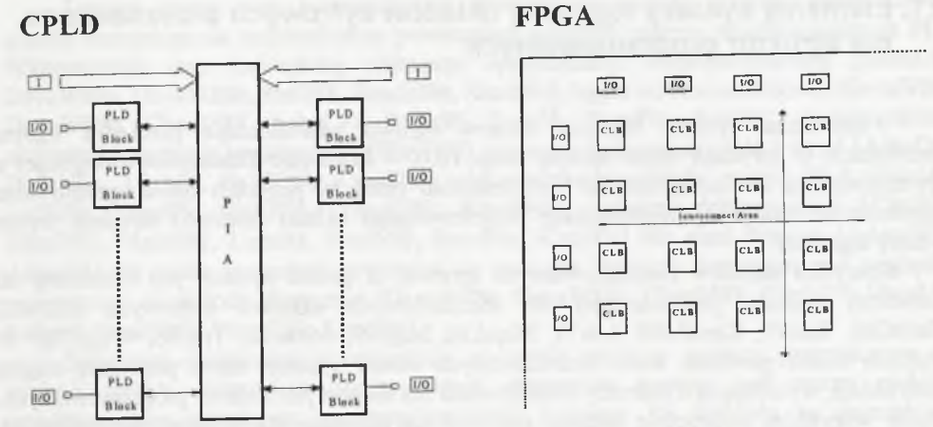
Rozbudowane układy programowalne (CPLD - *Complex Programmable Logic Devices*, FPGA - *Field Programmable Gate Array*) składają się z programowalnych komórek (bloków) logicznych oraz dodatkowych zasobów logicznych do ich łączenia i wymiany informacji z otoczeniem [Alte99, Shar98, Xili99]. Struktury CPLD, często nazywane strukturami matrycowymi [ŁubaZ02], zbudowane są z bloków logicznych podobnych do prostych układów PLD, bloków wejścia/wyjścia oraz matrycy połączeń, umożliwiającej wykonywanie dowolnych połączeń pomiędzy blokami logicznymi oraz blokami wejścia/wyjścia. Cechą charakterystyczną tego typu struktur, odróżniających je od układów FPGA, jest stały czas propagacji sygnału (niezależny od "drogi połączenia") wnoszony przez wewnętrzne połączenia [Alte99].

Drugą rodziną najbardziej rozbudowanych układów programowalnych są struktury FPGA typu tablicowego. Zawierają one konfigurowalne bloki logiczne (CLB - Configurable Logic Block), bloki wejścia/wyjścia oraz przestrzeń pomiędzy nimi, wykorzystywaną do wykonywania połączeń. Konfigurowalne bloki logiczne umożliwiają realizację każdej funkcji logicznej o określonej, najczęściej niewielkiej (3-8) liczbie argumentów. Struktura logiczna tych bloków jest podobna do struktury układów PLE. Program konfigurujący, odpowiadający za sposób wykonania połączeń wewnątrz układu, najczęściej zawarty jest w komórkach pamięci typu RAM [Xili99]. Uproszczoną strukturę układów CPLD oraz FPGA przedstawia rys. 1.3.

Struktury CPLD znajdują znaczące miejsce w szerokiej gamie układów programowalnych, dostarczanych na rynek przez wiele różnych firm, na czele z firmami Altera, AMD, Atmel, Cypress, Xilinx, Lattice [Alte99, Atme00, Adva95, Cypr96, Latt94, Xili99]. Jądrem większości tego typu układów jest struktura PAL, zawierająca programowalną matrycę AND i stałe połączenia w matrycy OR, tworząca blok logiczny typu PAL (rys.1.2). Bloki te mają zwykle dodatkowe zasoby logiczne pozwalające na efektywne wykorzystywanie występujących w nich iloczynów. Należą do nich między innymi: różnego typu ekspandery [Alte99], programowalne rozdzielacze (ang. Logic Allocator) [Adva95, Cypr96, Xili99], „szybkie" lokalne sprzężenia zwrotne [Atme00], elementy XOR, czy też wyjściowe bufory trójstanowe.

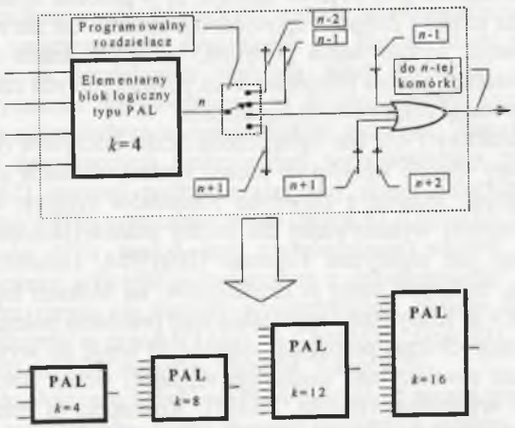


Rys. 1.2. Struktura bloku logicznego typu PAL, zawierającego *k* iloczynów
Fig. 1.2. Structure of PAL-based logic block consisting of *k* terms



Rys. 1.3. Architektura układów CPLD i FPGA
Fig. 1.3. CPLD and FPGA architectures

Dodatkowe, sprzętowe zasoby układów programowalnych typu PAL najczęściej umożliwiają nierównomierny rozdział iloczynów pomiędzy poszczególne komórki wyjściowe (logic allocator w układach MACH [Adva95] oraz FLASH [Cypr96], mixed mode w układach pLSI [Latt94] itp.). Mimo swoich niewątpliwych zalet nie zapewniają jednak realizacji każdej funkcji w jednym bloku. W tej sytuacji konieczna jest dodatkowa ekspansja liczby termów – składników sum. Ekspansja taka może wykorzystywać wewnętrzne sprzężenia zwrotne z wyjść na matrycę AND lub dodatkowe zasoby logiczne do tworzenia bloków logicznych typu PAL, zawierających dostosowaną do potrzeb liczbę iloczynów. Przykładem może być jedno z rozwiązań zawarte w opracowanych przez firmę AMD strukturach MACH, którego istota przedstawiona jest na rys. 1.4.



Rys. 1.4. Konfiguracje bloków logicznych typu PAL w układach MACH
Fig. 1.4. PAL-based logic blocks configurations applied to MACH devices

1.1. Elementy syntezy logicznej układów cyfrowych przeznaczone dla struktur programowalnych

Współczesna synteza logiczna stanowi ogniwo pośredniczące pomiędzy ogólną specyfikacją w językach opisu sprzętu (ang. HDL – Hardware Description Language) a odwzorowaniem technologicznym. Uniwersalność opisu w językach opisu sprzętu oraz różnorodność docelowej implementacji projektowanego układu znacząco utrudnia proces syntezy logicznej.

Specyfika układów programowalnych sprawia, iż proces syntezy jest odmienny od klasycznej syntezy, przeznaczonej dla standardowych układów cyfrowych SSI/MSI [GarbG02, Kali91, KamiM00, Lee76, MajeŁ86, Maje99, Mowl76, Trac86, Wagn78]. W ostatnich latach powstało wiele rozbudowanych metod syntezy, które podlegają ciągłej modyfikacji, wynikającej z potrzeby dostosowania ich do nowych struktur programowalnych. Przede wszystkim w procesie syntezy pojawiły się nieznane dla struktur standardowych elementy, takie jak na przykład:

- podział projektowanego układu na poszczególne części (ang. *partitioning*) realizowane przez odrębne proste układy programowalne lub bloki logiczne zawarte wewnątrz układów CPLD lub FPGA,
- rozmieszczenie poszczególnych bloków powstałych po podziale w odpowiednich prostych układach programowalnych lub wewnątrz struktury programowalnej (ang. *placement*),
- wykonywanie połączeń pomiędzy poszczególnymi blokami logicznymi (ang. *routing*).

W przypadku najprostszych struktur programowalnych (PAL, GAL, PLA) dominującą rolę odgrywa dwupoziomowa minimalizacja [Mich94]. Ma ona bezpośredni wpływ na redukcję liczby wykorzystywanych iloczynów w tego typu strukturach. Powszechnie uważa się, że jest ona wystarczająco skutecznym elementem syntezy logicznej dla prostych układów PLD.

Niewielka złożoność tych struktur sprawia, że w procesie syntezy pojawia się problem podziału projektu na części i doboru odpowiednich struktur do ich realizacji [BabbC92]. Ze względu na możliwość wykonywania połączeń, łączących każde dwa wybrane elementy struktury programowalnej, proces rozmieszczenia poszczególnych części w układzie oraz ich połączenie nie wprowadza dodatkowych ograniczeń.

Specyfika struktury PAL, tzn. ograniczona liczba iloczynów (k) dołączonych do sumy wyjściowej, w istotny sposób wpływa na proces syntezy układów cyfrowych przeznaczony dla tego typu układów. Jednym z głównych elementów syntezy, oprócz dwupoziomowej minimalizacji, najczęściej wykonywanej dla każdej jednowyjściowej funkcji oddzielnie za pomocą klasycznego już algorytmu Espresso [BrayH84, Łuba00, Mich94], jest sposób realizacji funkcji y_i, będących sumą p_i implikantów, na blokach logicznych zawierających k-iloczynów, gdy $p_i > k$. Klasyczne rozwiązanie tego problemu polega na tworzeniu sprzężeń zwrotnych zwiększających czas propagacji sygnału od wejść do wyjść [Bolt90, Skah96]. W literaturze znana jest również inna koncepcja ekspansji liczby iloczynów, wykorzystująca trójstanowe bufory wyjściowe [Bolt90, Phil85]. Koncepcja ta stanowi podwaliny metod zaprezentowanych między innymi w pracach [Kani99a, Kani99b].

Istnieje szereg prac prezentujących alternatywne metody syntezy, ukierunkowane przede wszystkim na optymalizację powierzchni prostych matryc programowalnych PLA. Wykorzystują one najczęściej elementy optymalizacji wielopoziomowej [BabbC92, DevaW88a, DevaW88b, Puch98, SaucS90b, Kani99c], logiki wielowartościowej [DevaW88a, DevaW88b, ChenM88, ŁubaL94, Łuba95, Sasa88, Sasa89a, Sasa91], metody doboru aktywności poziomu wyjściowego [WeyC90], elementy kodowania wejść i wyjść [AshaD92, DevaN91, Mich94, SaldK88, Kani02a], odpowiedniego podziału matryc PLA [HsuL91, LiuW94, Gree86, MaliH91, YangC89, Kani99d], elementy dekompozycji [CiesY92, JóźwV95, MaliH91, Łuba94, ProuN95, Sasa89b, Kani00a] lub sieci Petriego [AdamW97, Adam98]. W wielu sytuacjach prowadzą do znacznie lepszych rozwiązań pod względem powierzchni od metody klasycznej [DevaW88a, DevaW88b, ChenM88, CiesY92, MajeŁ92, Łuba94, SaucS90b, YangC89, Kani99c].

Najwięcej elementów nowatorskich zawierają strategie syntezy przeznaczone dla struktur FPGA. Jednym z podstawowych elementów syntezy jest proces podziału projektowanego układu na poszczególne bloki logiczne. Ze względu na segmentowy charakter połączeń dużego znaczenia nabiera sposób rozmieszczenia bloków powstających po podziale wewnątrz struktury FPGA (*ang. placement*) i ściśle z nim związany proces prowadzenia połączeń (*ang. routing*) [Acte95, Xili99]. Dodatkowo dwa końcowe etapy syntezy (*placement*, *routing*) odgrywają dużą rolę ze względu na ograniczone zasoby połączeń występujące w strukturach FPGA.

Układy FPGA przyczyniły się do odrodzenia teorii dekompozycji wprowadzonej przez Ashenhursta [Ashe57] i rozszerzonej przez Curtisa [Curt62, Curt63], pierwotnie wykorzystywanej do minimalizacji układów realizowanych na bramkach logicznych [Piec72a, Piec72b, Piec74, ShenM70, ShenM71]. Teoria dekompozycji stanowi teoretyczne podstawy podziału projektowanego układu na podukłady, które mogą być realizowane w blokach logicznych występujących w strukturach FPGA.

Pierwsze algorytmy syntezy (MIS-PGA [BrayR87, BrowF93], Chortle [FrancR91, BrowF93], Asyl [BabbC92, SicaC91]) przeznaczone dla struktur bramkowych (*ang. gate array*) zostały wykorzystane w procesie syntezy dla układów FPGA. Algorytmy te wykorzystują między innymi dekompozycję Roth-Karpa [RothK62], technikę wyłączania rdzeni (*ang. kernel extraction*) [BrayM82, Mich94], faktoryzację wyrażeń boolowskich wspomaganą procedurami grupowania [AbouS90], metodę leksykograficznego porządkowania zmiennych [AbouB93], programowanie dynamiczne itd. Bardzo często poszczególne etapy dekompozycji polegają na iteracyjnym podziale sieci logicznej, odpowiadającej w początkowym etapie wyrażeniu booleowskiemu po dwupoziomowej minimalizacji. Podział sieci logicznej polega na odpowiednim wyborze węzłów, gałęzi itd. [BrouF93, Mich94].

Klasyczny model dekompozycji funkcjonalnej zaproponowany przez Ashenhursta i Curtisa [Ashe57, Curt62] stanowi podstawę szerokiej grupy algorytmów dekompozycji opracowanych dla struktur FPGA typu tablicowego [BrzoŁ97, LeglW95, NowiŁ97, WanP92, Kani00b]. Jedną z zalet tych metod jest uwzględnianie stanów nieokreśloności w początkowym etapie syntezy, najczęściej związanym z dekompozycją. W ostatnich latach dużym zainteresowaniem cieszą się metody wykorzystujące binarne diagramy decyzyjne przedstawione między innymi w pracach [Aker78, ChanM96, DzikH01, DzikH03, Deva93, FujiF93, Laso99, MiliH01, Mili03, Sasa93b, StanS94, YangC02].

Szybki rozwój technologiczny układów FPGA sprawia, iż proces syntezy logicznej staje przed coraz większymi wyzwaniami. Mimo iż wiele elementów syntezy nie zostało jeszcze zadowalająco rozwiązanych, pojawiają się coraz to trudniejsze problemy, związane na

przykład z syntezą wysokiego poziomu, współbieżnym projektowaniem programu i sprzętu [CarpM98, GajsD92, GajsD98, OrteL98], syntezą układów wielokontekstowych [Atme00] itp.

Głównym etapem syntezy logicznej przeznaczonej dla rozbudowanych struktur matrycowych (CPLD) jest podział projektu na znajdujące się wewnątrz bloki logiczne. W większości przypadków struktury CPLD składają się z bloków logicznych typu PAL, zawierających ograniczoną liczbę iloczynów (k=3,4,5,8). Pierwszoplanowym celem syntezy jest efektywne wykorzystywanie bloków logicznych typu PAL. Synteza logiczna dla struktur matrycowych najczęściej stanowi rozszerzenie syntezy przeznaczonej dla prostych struktur programowalnych. Polega ona najczęściej na dwupoziomowej minimalizacji z dodatkowymi elementami związanymi z odpowiednim odwzorowaniem technologicznym, umożliwiającym wykorzystanie specyficznych cech poszczególnych rodzin układów. W przypadku układów CPLD znajdują również zastosowanie elementy optymalizacji wielopoziomowej [BrayH90, Mich94], syntezy wykorzystującej elementy XOR [Sasa93a, BranS93, SasaB90, SasaP93, Puch96] i różnego typu dekompozycji. W przypadku matrycowych struktur typu PAL mogą być wykorzystywane zarówno algorytmy przeznaczone do realizacji układów w strukturach bramkowych, jak i strategie klasycznej dekompozycji funkcjonalnej. Zdaniem autora elementy dekompozycji nie zostały jeszcze wprowadzone do procesu syntezy przeznaczonego dla struktur matrycowych. Okazuje się, że mogą one doprowadzić do znaczących oszczędności układowych [Kani00a]. Warto zwrócić uwagę, że w początkowym okresie, teoria dekompozycji (lata 1960-1970) wykorzystywana była do minimalizacji układów logicznych realizowanych na bramkach, stanowiąc konkurencję do minimalizacji Quine'a-McCluskeya [Quin52, McCl56, Bolt90, Kali91]. Wydaje się, że można by tą ideę wykorzystać również w procesie minimalizacji, przeznaczonej dla struktur matrycowych typu PAL.

Proces syntezy, zaimplementowany w znanych autorowi narzędziach wspomagających projektowanie układów cyfrowych dla struktur typu PAL (ABEL [Abel90], SNAP [Snap91], MACHXL [Mach95], MAX+PLUS [Alte96, Alte97] itp.), najczęściej rozpoczyna dwupoziomowa minimalizacja jednowyjściowych funkcji, po której następuje etap realizacji poszczególnych funkcji na występujących w strukturze blokach logicznych. Takie podejście jest logicznie związane z wewnętrzną budową struktury typu PAL. Często rozbudowywane jest o elementy umożliwiające wykorzystanie dodatkowych zasobów logicznych, występujących w elementarnym bloku logicznym (np. element XOR, wyjściowy bufor trójstanowy itp.) lub w większej jednostce strukturalnej, wyposażonej we wspólne lokalne zasoby (np. „szybkie" lokalne sprzężenia zwrotne, ekspandery itp.).

Różnorodność form implementacji układów cyfrowych doprowadziła do powstania uniwersalnych narzędzi projektowych, przeznaczonych dla szerokiej gamy układów programowalnych [Abel90, Symp00a, Symp00b]. Zaletą takich narzędzi wspomagających projektowanie jest możliwość przenoszenia opisu projektu pomiędzy różnymi firmowymi narzędziami. Najczęściej, ze względu na nieuwzględnienie w początkowym etapie syntezy właściwości docelowej struktury, nie zapewniają one jednak efektywnego wykorzystywania zasobów układu programowalnego.

Drugą grupę narzędzi wspomagających projektowanie stanowią narzędzia specjalizowane, wchodzące w skład firmowych pakietów oprogramowania dostarczanego przez poszczególnych producentów układów PLD [Snap91, Mach95, Alte97]. Pozwalają one stosunkowo efektywnie wykorzystać specyficzne elementy struktur, lecz mogą być wykorzystane do ograniczonej grupy układów programowalnych.

Najczęściej jednak w pierwszych etapach synteza logiczna wykonywana jest w oderwaniu od końcowego sposobu implementacji. Takie podejście, oprócz niewątpliwych zalet, do których należy przede wszystkim łatwa przenoszalność, prowadzi często do nieefektywnego wykorzystywania iloczynów zawartych w strukturach typu PAL. Pojawia się zatem oczywisty wniosek, iż trzeba poszukiwać metod syntezy, które umożliwiają implementację sieci logicznej w postaci elementów dostępnych w danej grupie układów programowalnych, wprowadzając charakterystyczne cechy tych układów w początkowe etapy syntezy. Tylko takie podejście zapewnia efektywne wykorzystanie zasobów struktur programowalnych.

Inspiracją do poszukiwania metod syntezy spełniających powyższe warunki były doświadczenia autora zdobyte podczas projektowania różnorodnych układów cyfrowych w strukturach programowalnych [HrynB96, KaniS92b, KaniS93b, HrynK96]. W pierwszym okresie zainteresowania autora ukierunkowane były przede wszystkim na opracowanie różnorodnych metod podziału projektowanego układu. Podział projektowanego układu ściśle związany był z metodami ekspansji liczby wejść, wyjść i termów [HrynK95, Kani99e]. Projektowany układ opisywany był tzw. Pierwotną Tablicą Przepaleń (PTP). Podział PTP na tzw. Charakterystyczne Tablice Przepaleń, odpowiadające konkretnym układom PAL, FPLA, stanowił podstawę podziału projektowanego układu na poszczególne proste struktury PLD [KaniS92a, KaniS93a, Kani95]. Drugi obszar zainteresowań związany był z metodami dekompozycji przeznaczonymi dla układów FPGA typu tablicowego [Kani96, Kani98a, KaniH96, Kani01a]. Prace te zostały zakończone rozprawą doktorską [Kani95] i opracowaniem prototypowego systemu syntezy Decomp [Kani00b]. Algorytmy dekompozycji zaimplementowane w tym systemie były ciągle udoskonalane [HrynK03, Kani03a], lecz w późniejszym okresie zainteresowania autora zostały skierowane przede wszystkim na struktury matrycowe typu PAL.

Okazało się, że dostosowanie poszczególnych elementów syntezy, pierwotnie opracowanych dla struktur typu FPGA, do właściwości struktur typu PAL (np. sposobu kodowania wzorców kolumn matrycy podziału, sposobu kodowania słów uwzględniającego typ aktywności wyjścia) doprowadziło do zaskakująco lepszych wyników od tych, które uzyskiwano za pomocą klasycznej metody syntezy [CzerK02, CzerK03, HrynP97, Kani98b, KaniH98, Kani99f, Kani99h, Kani99i, Kani00d, Kani00e, Kani01b]. Spowodowało to głębsze zainteresowanie się strukturami typu PAL. Opracowano dla nich szereg różnorodnych algorytmów syntezy [Kani99g, Kani00c, Kani00f, Kani00g, Kani01c, Kani02b, Kani02c, Kani02d, Kani03b, Kani03c, Kani03d]. Wszystkie pomysły były weryfikowane za pomocą tworzonych do tego celu prototypowych modułów programowych, często wykorzystujących moduły systemu Decomp, stanowiących ostatecznie poszczególne elementy systemu PALDec.

1.2. Tematyka pracy

Metody syntezy będące przedmiotem niniejszej pracy wykorzystują specyficzne cechy struktur matrycowych, scharakteryzowane przez właściwości bloku logicznego typu PAL. Koncepcja podziału całego układu matrycowego na bloki logiczne typu PAL umożliwia opracowanie ogólnych metod syntezy dla powszechnie występujących zasobów struktury matrycowej. Podejście to nie ogranicza jednak prezentowanych metod syntezy do konkretnej rodziny układów programowalnych. Dopiero w końcowym etapie syntezy istnieje możliwość optymalizacji układu, dopasowująca go do specyficznych właściwości wykorzystywanej struktury programowalnej.

Projektowany układ cyfrowy najczęściej optymalizowany jest pod kątem powierzchni (minimalizacja liczby wykorzystywanych bloków logicznych typu PAL) lub szybkości (minimalizacja warstw logicznych). Zaproponowane strategie syntezy obejmują obydwa obszary optymalizacyjne. Istotne nowatorstwo opracowanych strategii syntezy, którym poświęcona jest niniejsza praca, polega na dopasowaniu poszczególnych etapów syntezy projektowanego układu do zasobów logicznych struktur typu PAL.

Takie podejście w wielu przypadkach prowadzi do istotnego ograniczenia wykorzystywanych zasobów struktury programowalnej, często zapewniając realizację w znacznie prostszym, mniejszym, tańszym układzie niż struktura wymagana przez strategię klasyczną.

W kolejnych częściach pracy zostaną przedstawione między innymi:

- Metody realizacji wielowyjściowych funkcji w strukturach typu PAL wykorzystujące grafy wyjść (rozdział 2)
- Metody realizacji funkcji wykorzystujące wyjściowe bufory trójstanowe, prowadzące do znacznego zmniejszenia czasu propagacji sygnałów od wejścia do wyjścia (rozdział 3) (dwupoziomowa realizacja układów cyfrowych w strukturach typu PAL)
- Metody dekompozycji przeznaczone dla struktur typu PAL (rozdział 4)
- Strategie syntezy logicznej układów kombinacyjnych w strukturach typu PAL, wykorzystujące poszczególne, uprzednio przedstawione metody syntezy (rozdział 5).

Metodyka badań sprowadza się do zaimplementowania algorytmów syntezy i ich praktycznej weryfikacji. Efekty syntezy układów testowych będą porównane w pierwszej kolejności z metodą klasyczną, a potem z innymi metodami, zaimplementowanymi w dostępnych narzędziach programowych.

Każda część pracy kończy się przedstawieniem wyników eksperymentów, uzyskanych za pomocą opracowanych prototypowych programów, wykorzystanych do syntezy układów testowych (ang. benchmark) [Bench] w różnorodnych strukturach typu PAL (zawierających różnej wielkości bloki logiczne typu PAL).

1.3. Stosowana terminologia

Intencją autora było stosowanie ogólnie przyjętego nazewnictwa. Niestety polskie nazewnictwo dotyczące układów programowalnych nie jest dostatecznie ugruntowane i wystarczająco precyzyjne, dlatego zdaniem autora konieczne jest uzasadnienie występujących w pracy pojęć.

Stosowane w literaturze pojęcie ***układy matrycowe*** (czasami macierzowe) dotyczy szerokiej rodziny układów CPLD (ang. Complex Programmable Logic Devices). Tak się składa, że w obecnej chwili prawie wszystkie układy matrycowe zawierają matryce AND-OR, z programowalnymi połączeniami w matrycy AND i stałymi połączeniami w matrycy OR. Tego typu konfiguracja matryc AND-OR występuje w układach PAL. W niniejszej pracy zaproponowano określenie ***układy matrycowe typu PAL***, odnoszące się do układów CPLD, zbudowanych z matryc typu PAL. Drugą grupą układów matrycowych są ***układy matrycowe typu PLA***, tzn. układy CPLD zbudowane z matryc typu PLA. Ta grupa układów matrycowych była i jest mało popularna. Przedstawicielem tej grupy układów były opracowane przez firmę National Semiconductor układy MAPL (ang. Multiple Array Programmable Logic). Tak więc, mówiąc o ***strukturach typu PAL*** autor myśli o zanikających już prostych układach PAL oraz złożonych strukturach matrycowych typu PAL.

Proste układy PAL (np. PAL22V10), jak i matrycowe układy typu PAL składają się z podobnych elementów logicznych, zawierających określoną liczbę iloczynów dołączonych na stałe do sumy logicznej, nazywanych w pracy ***blokami logicznymi typu PAL***. Blok logiczny typu PAL, zawierający *k*-iloczynów, nazywany będzie skrótowo ***k*-iloczynowym blokiem PAL**.

Ograniczona liczba iloczynów zawartych w bloku logicznym typu PAL sprawia, że często trzeba zawracać sygnały z wyjść na matrycę AND. Ta powszechnie stosowana technika, wykorzystująca do ekspansji liczby iloczynów sprzężenia zwrotne, została nazwana ***realizacją klasyczną*** funkcji w strukturach typu PAL.

Wszystkie oznaczenia, pojęcia, skróty wprowadzone przez autora zostały przedstawione w tekście pracy oraz ujęte w odpowiednich wykazach znajdujących się na początku pracy.

2. METODY REALIZACJI WIELOWYJŚCIOWYCH FUNKCJI W STRUKTURACH TYPU PAL WYKORZYSTUJĄCE GRAFY WYJŚĆ

Większość układów programowalnych wykorzystuje jedną z trzech konfiguracji matryc AND-OR (rys.1.1). Taka dwupoziomowa struktura układów programowalnych niejako naturalnie wymusza w pierwszym etapie syntezy wykonanie minimalizacji dwupoziomowej. W późniejszym etapie występuje wpasowanie zminimalizowanych funkcji do wybranej struktury programowalnej.

Typ struktury układu programowalnego wpływa na sposób wykonywania minimalizacji dwupoziomowej. Struktura PLA umożliwia dołączenie każdego iloczynu do kilku sum równocześnie. W tej sytuacji w pierwszym etapie syntezy wskazane jest wykonanie minimalizacji dwupoziomowej wielowyjściowej funkcji opisującej zachowanie układu. W strukturach typu PAL możliwe jest dołączenie wybranego iloczynu tylko do jednej sumy, stąd najczęściej minimalizacja jest wykonywana dla każdego wyjścia oddzielnie.

2.1. Klasyczna metoda realizacji wielowyjściowych funkcji w strukturach typu PAL

Niech f będzie n-wejściową i m-wyjściową funkcją logiczną odwzorowującą zbiór B^n w zbiór B^m, tzn. $f: B^n \to B^m$, gdzie $B=\{0,1\}$. Klasyczna metoda realizacji funkcji $f: B^n \to B^m$ w strukturach typu PAL związana jest z realizacją zminimalizowanych funkcji $f_i: B^n \to B$ (i=1,2,...,m) w sieciach zbudowanych z k-iloczynowych bloków PAL. Każda funkcja $f_i: B^n \to B$ (i=1,2,...,m) może być opisana przez zbiór implikantów będących zbiorem kostek o wymiarach n oraz 1 nazywanych odpowiednio częścią wejściową oraz częścią wyjściową [Mich94]. Część wejściowa składająca się z elementów 0,-,1 reprezentuje iloczyn literałów, dla którego funkcja przyjmuje wartość 1, stanowiącą część wyjściową jednowyjściowego implikantu.

Przykład 2.1

Rozważmy funkcję $f: B^3 \to B^2$, którą po minimalizacji funkcji $f_i: B^3 \to B^1$ (i=1,2) można przedstawić w postaci $f_1 = \bar{a}\bar{c} + bc$, $f_2 = bc + ab$.

Zbiory kostek określających każdą z funkcji zawierają po 2 elementy, tzn.:

f_1:

a	b	c
0	-	0
-	1	1

f_2:

a	b	c
-	1	1
1	1	-

Załóżmy, że dla uproszczenia będą rozpatrywane struktury typu PAL z wyjściem aktywnym w stanie wysokim, stąd 1 w części wyjściowej jednowyjściowego implikantu odpowiada wykorzystaniu jednego iloczynu w bloku logicznym typu PAL.

Niech wyróżnik Δ_{f_i} będzie liczbą implikantów, dla których funkcja $f_i: B^n \to B$ przyjmuje wartość 1.

Niech δ_{f_i} oznacza liczbę bloków potrzebnych do realizacji i-tej funkcji. Rozważmy przypadek realizacji funkcji f_i za pomocą k-iloczynowych bloków PAL. W sytuacji, gdy $\Delta_{f_i} > k$, realizując funkcję f_i konieczne jest zawracanie sygnałów z wyjść na matrycę połączeń, określane mianem sprzężeń zwrotnych. Prowadzi to do wykorzystania $\delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1$ k-iloczynowych bloków PAL, gdzie $\lceil x \rceil$ oznacza najmniejszą liczbę naturalną nie mniejszą od x. Do realizacji m-funkcji (każda funkcja $f_i: B^n \to B$ (i=1,2,...,m) zminimalizowana oddzielnie) używa się δ_f^1 bloków logicznych typu PAL, gdzie:

$$\delta_f^1 = \sum_{i=1}^{m} \delta_{f_i} \tag{2.1}$$

Przykład 2.2

Rozważmy funkcję $f: B^5 \to B^4$, która po dwupoziomowej minimalizacji jednowyjściowych funkcji $f_i: B^5 \to B$ (i=1,2,3,4) za pomocą programu Espresso (Espresso-Dso) przedstawiona jest w postaci plików utworzonych w formacie berkeleyowskim *yi.pla* (rys.2.2).

Z wyniku minimalizacji określane są wartości $\Delta_{f_4} = 8$, $\Delta_{f_3} = 6$, $\Delta_{f_2} = 8$ i $\Delta_{f_1} = 3$. Zakładając realizację funkcji z wykorzystaniem bloków logicznych typu PAL, zawierających trzy iloczyny (k=3), należy użyć:

$$f_4: \quad \delta_{f_4} = \left\lceil \frac{\Delta_{f_4} - k}{k-1} \right\rceil + 1 = 4$$

$$f_3: \quad \delta_{f_3} = \left\lceil \frac{\Delta_{f_3} - k}{k-1} \right\rceil + 1 = 3$$

$$f_2: \quad \delta_{f_2} = \left\lceil \frac{\Delta_{f_2} - k}{k-1} \right\rceil + 1 = 4$$

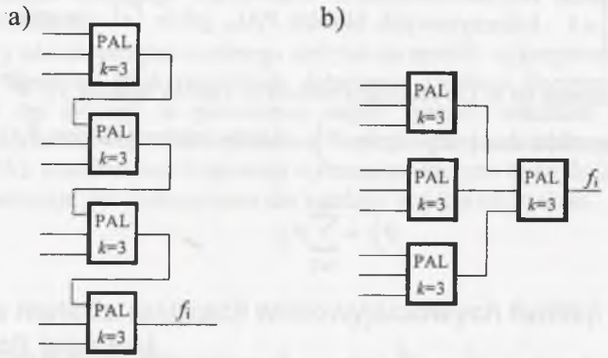
$$f_1: \quad \delta_{f_1} = \left\lceil \frac{\Delta_{f_1} - k}{k-1} \right\rceil + 1 = 1$$

wybranych bloków.

Sumując wyznaczone wartości okazuje się, że do klasycznej realizacji funkcji $f: B^5 \to B^4$ należy użyć $\delta_f^1 = \sum_{i=1}^{4}\left(\left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1\right) = \sum_{i=1}^{4} \delta_{f_i} = 12$ bloków logicznych typu PAL, zawierających trzy iloczyny.

Jeżeli $\Delta_{f_i} > k$, gdzie k jest liczbą iloczynów zawartych w pojedynczym bloku logicznym typu PAL, to realizacja implikantów wymaga użycia więcej niż jednego bloku. Konsekwencją tego jest wprowadzanie wewnętrznych sprzężeń zwrotnych, umożliwiających ekspansję liczby iloczynów.

Istnieją dwa sposoby ekspansji liczby iloczynów wykorzystujące sprzężenia zwrotne (rys.2.1).



Rys. 2.1. Ekspansja liczby iloczynów wykorzystująca sprzężenia zwrotne
Fig. 2.1. Product term expansion using feedback

Strategia ekspansji przedstawiona na rys. 2.1b wykorzystuje identyczną liczbę bloków co strategia z rys. 2.1a, wprowadzając jednocześnie mniejszą liczbę warstw logicznych.

Klasyczna realizacja przykładowej funkcji wraz z opisem poszczególnych elementów, wykorzystująca pierwszy sposób ekspansji (rys. 2.1a), przedstawiona jest na rys. 2.2.

W ogólnym przypadku realizacja funkcji $f_i: B^n \to B$ będącej sumą Δ_{f_i} implikantów za pomocą bloków logicznych zawierających k iloczynów, przedstawiona na rys. 2.1b, prowadzi do uzyskania układu w postaci struktury ξ_{f_i} warstwowej, gdzie ξ_{f_i} jest minimalną liczbą naturalną spełniającą nierówność $\Delta_{f_i} \le k^{\xi_{f_i}}$, tzn. $\xi_{f_i} = \lceil \lg_k \Delta_{f_i} \rceil$.

Wielowyjściowa funkcja $f: B^n \to B^m$ realizowana jest metodą klasyczną z minimalizacją liczby warstw logicznych (metoda K_LW) w postaci struktury ξ_f^1 warstwowej, gdzie $\xi_f^1 = \max(\xi_{f_1}, \xi_{f_2}, \ldots, \xi_{f_m})$.

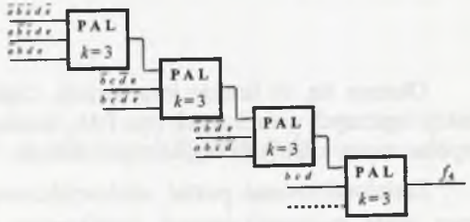
$f: B^5 \to B^4$

```
.i 5
.o 4
.ilb a b c d e
.ob f4 f3 f2 f1
.p 25
00010 1000
10011 1000
01-11 1000
-0101 1000
-1000 1000
00-01 1000
1100- 1000
-111- 1000
111-0 0100
01011 0100
00010 0100
00-01 0100
-1000 0100
-111- 0100
10011 0010
00010 0010
01-11 0010
0010- 0010
-1000 0010
00-01 0010
1100- 0010
-111- 0010
01000 0001
01011 0001
00-01 0001
.e
```

$f_4: B^5 \to B$

```
.i 5
.o 1
.ilb a b c d e
.ob f4
.p 8
00010 1
10011 1
01-11 1
-0101 1
-1000 1
00-01 1
1100- 1
-111- 1
.e
```

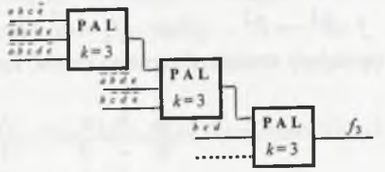
$\Delta_{f_4} = 8 \qquad \delta_{f_4} = \left\lceil \frac{\Delta_{f_4} - k}{k-1} \right\rceil + 1 = 4$



$f_3: B^5 \to B$

```
.i 5
.o 1
.ilb a b c d e
.ob f3
.p 6
111-0 1
01011 1
00010 1
00-01 1
-1000 1
-111- 1
.e
```

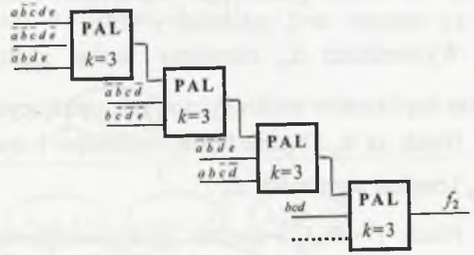
$\Delta_{f_3} = 6 \qquad \delta_{f_3} = \left\lceil \frac{\Delta_{f_3} - k}{k-1} \right\rceil + 1 = 3$



$f_2: B^5 \to B$

```
.i 5
.o 1
.ilb a b c d e
.ob f2
.p 8
10011 1
00010 1
01-11 1
0010- 1
-1000 1
00-01 1
1100- 1
-111- 1
.e
```

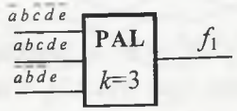
$\Delta_{f_2} = 8 \qquad \delta_{f_2} = \left\lceil \frac{\Delta_{f_2} - k}{k-1} \right\rceil + 1 = 4$



$f_1: B^5 \to B$

```
.i 5
.o 1
.ilb a b c d e
.ob f1
.p 3
01000 1
01011 1
00-01 1
.e
```

$\Delta_{f_1} = 3 \qquad \delta_{f_1} = \left\lceil \frac{\Delta_{f_1} - k}{k-1} \right\rceil + 1 = 1$



Rys. 2.2. Klasyczna realizacja funkcji $f: B^5 \to B^4$ wykorzystująca bloki logiczne typu PAL zawierające trzy iloczyny logiczne (linią przerywaną oznaczono nie wykorzystane iloczyny)

Fig. 2.2. Classical implementation of the function $f: B^5 \to B^4$ based on PAL-based logic blocks including three terms (unused terms marked with a dashed line)

2.2. Reprezentacja wielowyjściowej funkcji w postaci skierowanego grafu wyjść

Okazuje się, że istnieje inny sposób, często efektywniejszej realizacji wielowyjściowych funkcji logicznych w strukturach typu PAL, bazujący na wyszukiwaniu implikantów, stanowiących wspólne zasoby kilku jednowyjściowych funkcji.

Zminimalizowana postać wielowyjściowej funkcji $f: B^n \to B^m$ jest przedstawiona przez zbiór wielowyjściowych implikantów zawierających część wejściową złożoną z elementów 0,-,1 oraz część wyjściową złożoną z elementów 0,1 [Mich94].

Przykład 2.3

Niech $f: B^3 \to B^2$, gdzie $f_1 = \bar{a}\bar{c} + bc$, $f_2 = bc + ab$. Zbiór wielowyjściowych implikantów opisujący zminimalizowaną postać funkcji f zawiera 3 elementy, tzn.:

a	b	c	f_1	f_2
0	-	0	1	0
-	1	1	1	1
1	1	-	0	1

Niech y będzie m-elementowym wektorem wyjściowym odpowiadającym części wyjściowej wielowyjściowego implikantu (przykładowo dla pierwszego implikantu `0-0 10` wartość m=2, natomiast y=10).

Wyróżnikiem Δ_y nazwijmy liczbę identycznych wektorów y występujących w zbiorze implikantów wielowyjściowych, opisujących funkcję $f: B^n \to B^m$.

Niech $\mu(\Delta_y)$ będzie liczbą elementów 1 zawartych w wektorze y. Nazwijmy wartość $\mu(\Delta_y)$ rzędem wyróżnika Δ_y.

Niech $G < \mathbf{Y}, \vec{\mathbf{U}} >$ będzie grafem skierowanym, gdzie $\mathbf{Y}$ jest zbiorem wszystkich wierzchołków Δ_y, natomiast $\vec{\mathbf{U}}$ zbiorem krawędzi łączących wierzchołki Δ_{y_s}, Δ_{y_r} takie, że odległość kodowa wektorów y_s, y_r wynosi 1 oraz $\mu(\Delta_{y_s}) + 1 = \mu(\Delta_{y_r})$.

Przykład 2.4

Rozważmy funkcję $f: B^5 \to B^4$ z przykładu 2.2, która po minimalizacji przyjmuje postać przedstawioną w pliku *f.pla* (rys.2.3). Wartości wyróżników Δ_y dla poszczególnych rzędów wyróżników $\mu(\Delta_y)$=1,2,3,4 wierzchołków grafu, krawędzie skierowane wraz z pierwotnym grafem wyjść, odpowiadającym przykładowej funkcji $f: B^5 \to B^4$ przedstawione są na rys. 2.3.

$f: B^5 \to B^4$

(f.pla)

```
.i 5
.o 4
.ilb a b c d e
.ob f4 f3 f2 f1
.p 10
11-00 0100
10011 1010
00010 1110
01000 1111
01011 1111
-0101 1000
0010- 0010
00-01 1111
1100- 1010
-111- 1110
.e
```

$\mu(\Delta_y)$=4 $\quad \Delta_{1111} = 3$

$\mu(\Delta_y)$=3 $\quad \Delta_{1110} = 2; \Delta_{1101} = 0; \Delta_{1011} = 0; \Delta_{0111} = 0$

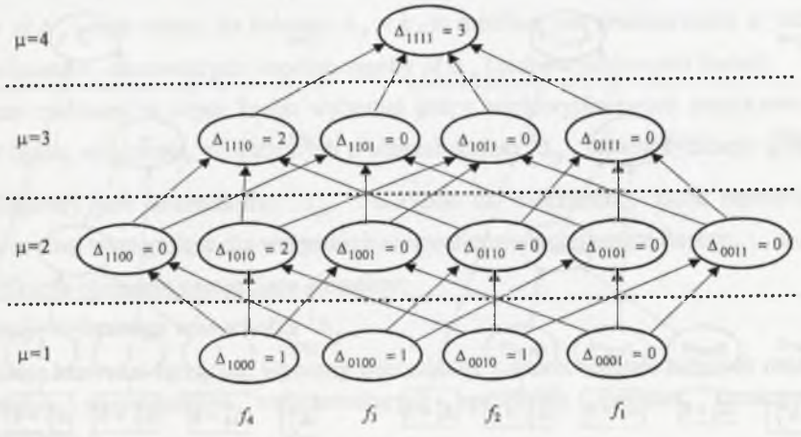
$\mu(\Delta_y)$=2

$\Delta_{1100} = 0; \Delta_{1010} = 2; \Delta_{1001} = 0; \Delta_{0110} = 0; \Delta_{0101} = 0; \Delta_{0011} = 0$

$\mu(\Delta_y)$=1 $\quad \Delta_{1000} = 1; \Delta_{0100} = 1; \Delta_{0010} = 1; \Delta_{0001} = 0$

$\mathbf{Y}=\{\Delta_{1111},\Delta_{1110},\Delta_{1101},\Delta_{1011},\Delta_{0111},\Delta_{1100},\Delta_{1010},\Delta_{1001},\Delta_{0110},\Delta_{0101},\Delta_{0011},\Delta_{1000},\Delta_{0100},\Delta_{0010},\Delta_{0001}\}$

$\vec{\mathbf{U}} = \{(\Delta_{1000},\Delta_{1100}); (\Delta_{1000},\Delta_{1010}); (\Delta_{1000},\Delta_{1001}); (\Delta_{0100},\Delta_{1100}); (\Delta_{0100},\Delta_{0110}); (\Delta_{0100},\Delta_{0101});$
$(\Delta_{0010},\Delta_{1010}); (\Delta_{0010},\Delta_{0110}); (\Delta_{0010},\Delta_{0011}); (\Delta_{0001},\Delta_{1001}); (\Delta_{0001},\Delta_{0101}); (\Delta_{0001},\Delta_{0011});$
$(\Delta_{1100},\Delta_{1110}); (\Delta_{1010},\Delta_{1110}); (\Delta_{0110},\Delta_{1110}); (\Delta_{1100},\Delta_{1101}); (\Delta_{1001},\Delta_{1101}); (\Delta_{0101},\Delta_{1101});$
$(\Delta_{1010},\Delta_{1011}); (\Delta_{1001},\Delta_{1011}); (\Delta_{0011},\Delta_{1011}); (\Delta_{0110},\Delta_{0111}); (\Delta_{0101},\Delta_{0111}); (\Delta_{0011},\Delta_{0111});$
$(\Delta_{1110},\Delta_{1111}); (\Delta_{1101},\Delta_{1111}); (\Delta_{1011},\Delta_{1111}); (\Delta_{0111},\Delta_{1111})\}$



Rys. 2.3. Przykład pierwotnego grafu wyjść

Fig. 2.3. Example of a primary graph of outputs

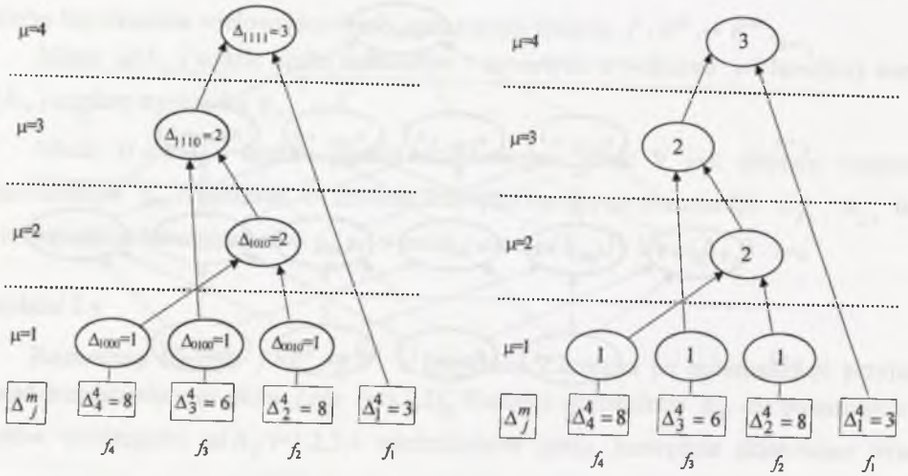
Rozpatrując graf z rys. 2.3 każdemu wierzchołkowi pierwszego rzędu związanemu z implikantami j-tego (j=1,2,3,4) wyjścia m-wyjściowej (m=4) funkcji przyporządkowuje się wartość dziesiętną Δ_j^m równą sumie wyróżników należących do dróg skierowanych, których początkiem jest wybrany wierzchołek 1 rzędu, a końcem wierzchołek 4 rzędu. Przykładowo dla wierzchołka Δ_{0100} istnieje sześć takich dróg:

$\Delta_{0100} \to \Delta_{1100} \to \Delta_{1110} \to \Delta_{1111}$
$\Delta_{0100} \to \Delta_{1100} \to \Delta_{1101} \to \Delta_{1111}$
$\Delta_{0100} \to \Delta_{0110} \to \Delta_{1110} \to \Delta_{1111}$
$\Delta_{0100} \to \Delta_{0110} \to \Delta_{0111} \to \Delta_{1111}$
$\Delta_{0100} \to \Delta_{0101} \to \Delta_{1101} \to \Delta_{1111}$
$\Delta_{0100} \to \Delta_{0101} \to \Delta_{0111} \to \Delta_{1111}$

stąd

$$\Delta_3^4 = \Delta_{0100} + \Delta_{1100} + \Delta_{0110} + \Delta_{0101} + \Delta_{1110} + \Delta_{1101} + \Delta_{0111} + \Delta_{1111} = 6.$$

Uogólniając rozważanie stwierdza się, że każdemu wierzchołkowi pierwszego rzędu $\Delta_{\substack{0\ \dots\ 0\ 1\ 0\ \dots 0\ 0 \\ m,\dots,j+1,j,j-1,\dots 2,1}}$ należy przyporządkować liczbę Δ_j^m równą sumie wyróżników $\Delta_{\substack{-\ -\ -\ 1\ -\ \dots\ -\ - \\ m,\dots,j+1,j,j-1,\dots 2,1}}$ zawartych w wierzchołkach pokrytych przez wszystkie drogi wychodzące z tego wierzchołka, a kończące się w wierzchołkach wyższych rzędów. Eliminując z pierwotnego grafu wyjść wierzchołki, dla których $\Delta_y = 0$ i przyporządkowując w tej sytuacji wyróżniki Δ_j^m wierzchołkom najniższych rzędów, związanych z realizacją j-tej funkcji, otrzymujemy zredukowany graf wyjść przedstawiony na rys. 2.4. Rysunek 2.4 przedstawia dwie równoważne formy zredukowanego grafu wyjść funkcji z przykładu 2.4. Uproszczona postać z prawej strony rysunku zawiera w wierzchołkach grafu wyjść tylko wartości poszczególnych wyróżników.



Rys. 2.4. Przykład zredukowanego grafu wyjść z wartościami wyróżników Δ_j^m

Fig. 2.4. Example of a reduced graph of outputs with associated values of discriminants Δ_j^m

Na podstawie wartości wyróżników Δ_j^m określa się liczbę k-iloczynowych bloków PAL, potrzebnych do realizacji wielowyjściowej funkcji. Liczba ta wynosi:

$$\delta_f = \sum_{j=1}^{m} \left(\left\lceil \frac{\Delta_j^m - k}{k-1} \right\rceil + 1 \right) \qquad (2.2)$$

Najczęściej jest ona większa od wartości współczynnika δ_f^1, wyznaczonego z zależności (2.1). Należy podkreślić, że wartość współczynnika δ_f określa liczbę bloków logicznych typu PAL, potrzebnych do realizacji wielowyjściowej funkcji $f : B^n \to B^m$, przy czym każda jednowyjściowa funkcja $f_i : B^n \to B$, (i=1,2,...,m) realizowana jest oddzielnie, a proces realizacji wykonany jest po minimalizacji funkcji $f : B^n \to B^m$.

2.3. Idea implementacji funkcji wielowyjściowej wykorzystująca grafy wyjść

Załóżmy, że chcemy zrealizować wielowyjściową funkcję $f : B^n \to B^m$ opisaną grafem wyjść $G < \mathbf{Y}, \vec{\mathbf{U}} >$ na k-iloczynowych blokach PAL.

Wierzchołki grafu wyjść odpowiadają poszczególnym grupom wielowyjściowych implikantów z identyczną częścią wyjściową. Jeżeli przykładowo na grafie znajduje się wierzchołek $\mu(\Delta_y)$-tego rzędu, dla którego $\Delta_y = k$, to możliwe jest zrealizowanie w jednym bloku k-implikantów, stanowiących wspólne zasoby $\mu(\Delta_y)$ jednowyjściowych funkcji.

Efektem realizacji w i-tym kroku wybranej grupy wielowyjściowych implikantów, z identyczną częścią wyjściową, skojarzonych z wierzchołkiem ${}^i\Delta_y$, jest modyfikacja grafu. Z grafu eliminowany jest wierzchołek ${}^i\Delta_y$. Tworzone są wierzchołki, które reprezentują sprzężenia zwrotne, występujące dla wszystkich niezrealizowanych jeszcze funkcji.

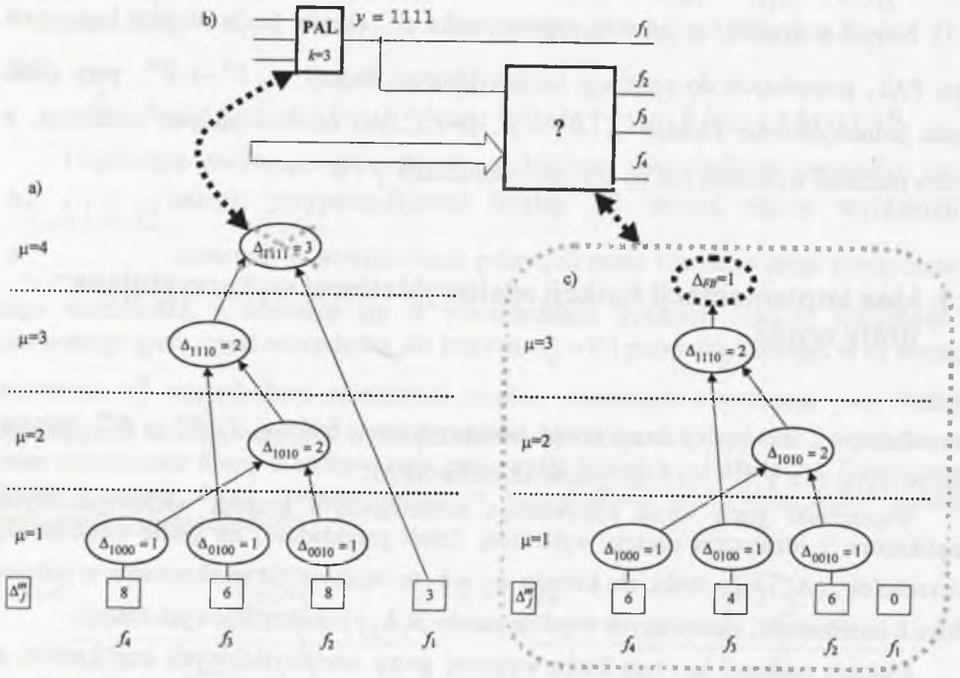
Modyfikacja obejmuje następujące elementy:

- eliminację wybranego wierzchołka ${}^i\Delta_y$,
- eliminację krawędzi łączących wybrany wierzchołek z wierzchołkami niższych rzędów,
- utworzenie wierzchołków reprezentujących sprzężenia zwrotne, oznaczonych wyróżnikami Δ_{FB},
- utworzenie krawędzi łączących wierzchołki reprezentujące sprzężenia zwrotne z odpowiednimi wierzchołkami, niezrealizowanych jeszcze jednowyjściowych funkcji.

Wierzchołki reprezentujące sprzężenia zwrotne tworzone są dla każdej niezrealizowanej jeszcze jednowyjściowej funkcji, wybranej spośród $\mu({}^i\Delta_y)$-funkcji, dla których w części

wyjściowej y wystąpiła 1. Jeżeli dla jakiejś funkcji występował wierzchołek reprezentujący sprzężenia zwrotne, to modyfikacja polega na inkrementacji wartości wyróżnika.

Konsekwencją przekształcenia grafu jest odpowiednie zmniejszenie współczynników Δ_j^m. Rysunek 2.5 przedstawia przykład modyfikacji grafu wyjść po realizacji implikantów odpowiadających wierzchołkowi czwartego rzędu Δ_{1111}. Wprowadzenie sprzężeń zwrotnych zobrazowane jest przez dodatkowy wierzchołek z wyróżnikiem Δ_{FB}, wyróżniony linią przerywaną.



Rys. 2.5. a) Zredukowany graf wyjść; b) Realizacja implikantów opisanych przez wierzchołek 4 rzędu za pomocą 3-iloczynowego bloku logicznego typu PAL; c) Graf wyjść po redukcji związanej z realizacją implikantów opisanych przez wierzchołek 4 rzędu

Fig. 2.5. a) Reduced graph of outputs; b) Implementation of implicants defined by fourth-range node by PAL-based logic block consisting of three terms; c) Graph of outputs after reduction related to implementation of implicants defined by fourth-range node

2.4. Twierdzenie o wyborze wierzchołków grafu wyjść

Lemat:

Realizacja w i-tym kroku grupy implikantów, odpowiadających wybranemu wierzchołkowi ${}^i\Delta_y$, prowadzi do minimalizacji liczby wykorzystywanych k-iloczynowych bloków PAL wtedy, gdy spełniony jest warunek ${}^i\delta_f - {}^{i+1}\delta_f > \left\lceil \frac{{}^i\Delta_y - k}{k-1} \right\rceil + 1$.

Dowód:

Realizacja grupy ${}^i\Delta_y$ implikantów, odpowiadających wybranemu wierzchołkowi ${}^i\Delta_y$, wymaga użycia ${}^i\gamma = \left\lceil \frac{{}^i\Delta_y - k}{k-1} \right\rceil + 1$ k-iloczynowych bloków PAL. Wybór w i-tym kroku wierzchołka ${}^i\Delta_y$, prowadzi do modyfikacji grafu wyjść, wynikiem której jest zmiana wartości współczynnika δ_f, z wartości ${}^i\delta_f$ do wartości ${}^{i+1}\delta_f$. Jeżeli wartość różnicy ${}^i\delta_f - {}^{i+1}\delta_f$ jest większa od wartości ${}^i\gamma = \left\lceil \frac{{}^i\Delta_y - k}{k-1} \right\rceil + 1$, to realizacja w i-tym kroku grupy implikantów, odpowiadających wybranemu wierzchołkowi ${}^i\Delta_y$, prowadzi do minimalizacji liczby wykorzystywanych k-iloczynowych bloków PAL.

cbdo.

Ponieważ wybór wierzchołka ${}^i\Delta_y$ wpływa na $\mu({}^i\Delta_y)$ wyróżników ${}^i\Delta_j^m$, stąd (po uporządkowaniu wyróżników w taki sposób, że wpływa on na kolejnych ${}^i\Delta_j^{\mu({}^i\Delta_y)}$ wyróżników) warunek minimalizacji liczby bloków logicznych typu PAL w i-tym kroku można przedstawić w postaci:

$$\sum_{j=1}^{\mu({}^i\Delta_y)} \left(\left\lceil \frac{{}^i\Delta_j^{\mu({}^i\Delta_y)} - k}{k-1} \right\rceil + 1 \right) - \sum_{j=1}^{\mu({}^i\Delta_y)} \left(\left\lceil \frac{{}^{i+1}\Delta_j^{\mu({}^i\Delta_y)} - k}{k-1} \right\rceil + 1 \right) > \left\lceil \frac{{}^i\Delta_y - k}{k-1} \right\rceil + 1 \quad (2.3)$$

Niech $r_j^{\mu({}^i\Delta_y)}$ będą liczbami wyznaczonymi z kongruencji:

$$ {}^i\Delta_j^{\mu({}^i\Delta_y)} - 1 \equiv r_j^{\mu({}^i\Delta_y)} (\mathrm{mod}(k-1)), \tag{2.4}$$

gdzie j=1,2,..., $\mu({}^i\Delta_y)$.

Twierdzenie *(o wyborze wierzchołka grafu wyjść)*

Jeżeli istnieje wierzchołek grafu wyjść (wyróżnik) ${}^i\Delta_y$

1. rzędu $\mu({}^i\Delta_y) > 2$, dla którego ${}^i\Delta_y \geq k$

lub

2. rzędu $\mu({}^i\Delta_y) \geq 2$, dla którego w zbiorze reszt $\mathbf{R} = \{r_j^{\mu({}^i\Delta_y)};\ j \in <1, \mu({}^i\Delta_y) > \}$

istnieją co najmniej 2 reszty, takie że $0 < r_a^{\mu({}^i\Delta_y)} < {}^i\Delta_y < k$ i $0 < r_b^{\mu({}^i\Delta_y)} < {}^i\Delta_y < k$

lub

3. rzędu $\mu({}^i\Delta_y) = 2$, dla którego ${}^i\Delta_y = k$ lub ${}^i\Delta_y \geq 2k-1$

lub

4. rzędu $\mu({}^i\Delta_y) = 2$, dla którego $k < {}^i\Delta_y < 2k-1$ i w zbiorze reszt

$\mathbf{R} = \{r_j^{\mu({}^i\Delta_y)};\ j \in <1, \mu({}^i\Delta_y) > \}$ istnieje co najmniej jedna reszta, taka że

$0 < r_a^{\mu({}^i\Delta_y)} < {}^i\Delta_y - (k-1)$,

to realizacja implikantów odpowiadających takiemu wyróżnikowi prowadzi do minimalizacji liczby k-iloczynowych bloków PAL, potrzebnych do realizacji wielowyjściowej funkcji.

Dowód twierdzenia zamieszczony jest w dodatku.

Istota przedstawionego twierdzenia polega na wskazaniu, jakie warunki powinna spełniać wartość wyróżnika, aby realizacja skojarzonych z nim wielowyjściowych implikantów prowadziła do minimalizacji wykorzystywanych bloków logicznych typu PAL. Twierdzenie o wyborze wierzchołka grafu wyjść stanowi podstawę, na której można budować algorytm efektywnej realizacji wielowyjściowych funkcji w strukturach typu PAL.

Ponieważ liczba bloków $\delta_f = \sum_{j=1}^{m} \left(\left\lceil \frac{\Delta_j^m - k}{k-1} \right\rceil + 1 \right)$ (zależność 2.2) jest z reguły większa od wartości $\delta_f^1 = \sum_{i=1}^{m} \left(\left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 \right)$ (zależność 2.1), stąd istota realizacji wielowyjściowej

funkcji polega na analizie grafu wyjść i wyszukiwaniu wierzchołków skojarzonych z możliwie największymi grupami wspólnych implikantów.

Załóżmy, że graf wyjść opisuje wielowyjściowe implikanty realizowalne na ${}^i\delta_f$ k-iloczynowych blokach PAL. Wybór w i-tym kroku wierzchołka ${}^i\Delta_y$ pociąga za sobą wykorzystanie ${}^i\gamma = \left\lceil \frac{{}^i\Delta_y - k}{k-1} \right\rceil + 1$ bloków logicznych typu PAL. Prowadzi to do redukcji grafu wyjść, który po redukcji opisuje implikanty realizowalne na ${}^{i+1}\delta_f$ blokach logicznych typu PAL. Wybór wierzchołka jest tym lepszy, im większa jest wartość wyrażenia ${}^i\delta_f - ({}^{i+1}\delta_f + {}^i\gamma)$.

Jak wybierać wierzchołki grafu wyjść, dla których ${}^i\delta_f - ({}^{i+1}\delta_f + {}^i\gamma) > 0$ i ${}^i\delta_f - ({}^{i+1}\delta_f + {}^i\gamma) = \max$?

Zasady wyboru wierzchołka, wynikające pośrednio z twierdzenia o wyborze wierzchołka grafu wyjść, można przedstawić następująco:

1. Najlepiej w pierwszej kolejności wybierać wierzchołek ${}^i\Delta_y$, dla którego $\mu({}^i\Delta_y) = \max$ (wybrany wierzchołek musi spełniać warunki twierdzenia).
2. Spośród wierzchołków tego samego rzędu należy wybierać, w zależności od wartości wyróżników:

2a. *Jeżeli istnieją wierzchołki, dla których* ${}^i\Delta_y \geq k$

- wierzchołek, dla którego wyróżnik ${}^i\Delta_y = \max$

2b. *Jeżeli wartości wszystkich wyróżników są mniejsze od k*

- wierzchołek, dla którego w zbiorze reszt $\mathbf{R} = \{r_j^{\mu({}^i\Delta_y)};\ j \in <1, \mu({}^i\Delta_y) > \}$ istnieje maksymalna liczba reszt $r_x^{\mu({}^i\Delta_y)}$ spełniających warunek $0 < r_x^{\mu({}^i\Delta_y)} < {}^i\Delta_y < k$.

Twierdzenie o wyborze wierzchołka grafu wyjść precyzuje warunki minimalizacji liczby bloków typu PAL potrzebnych do realizacji wielowyjściowych funkcji w i-tym kroku. W pewnych sytuacjach (brak wierzchołków spełniających warunki twierdzenia; brak wierzchołków, dla których ${}^i\delta_f - ({}^{i+1}\delta_f + {}^i\gamma) > 0$) uzasadniony jest wybór wierzchołków, dla których ${}^i\delta_f - ({}^{i+1}\delta_f + {}^i\gamma) = 0$.

Okazuje się, że jeżeli dla wierzchołków drugiego rzędu, dla których $k < {}^i\Delta_y < 2k-1$, nie istnieje żadna reszta $0 < r_x^{\mu({}^i\Delta_y)} < {}^i\Delta_y - (k-1)$ lub w zbiorze reszt $\mathbf{R} = \{r_j^{\mu({}^i\Delta_y)};\ j \in <1, \mu({}^i\Delta_y) > \}$ istnieje tylko jedna reszta $r_x^{\mu({}^i\Delta_y)}$ taka, że $0 < r_x^{\mu({}^i\Delta_y)} < {}^i\Delta_y < k$, to realizacja implikantów

skojarzonych z wierzchołkiem ${}^{i}\Delta_y$ nie zmienia liczby wykorzystywanych bloków logicznych typu PAL, tzn. ${}^{i}\delta_f = {}^{i+1}\delta_f + {}^{i}\gamma$.

Wybór takich wierzchołków stwarza nadzieję na pojawienie się po redukcji grafu wyjść wierzchołka ${}^{i+1}\Delta_y$, dla którego ${}^{i+1}\delta_f - ({}^{i+2}\delta_f + {}^{i+1}\gamma) > 0$ (minimalizacja liczby bloków typu PAL w kolejnym $(i+1)$-szym kroku).

Przykład 2.5

Rozważmy graf wyjść funkcji $f: B^n \to B^4$, który przed wyborem wierzchołka przyjmuje postać przedstawioną na rys. 2.6a. Załóżmy, że wykorzystujemy bloki logiczne typu PAL, zawierające 3 iloczyny. Wierzchołkom 1 rzędu można przyporządkować wartości Δ_j^m oraz r_j^m.

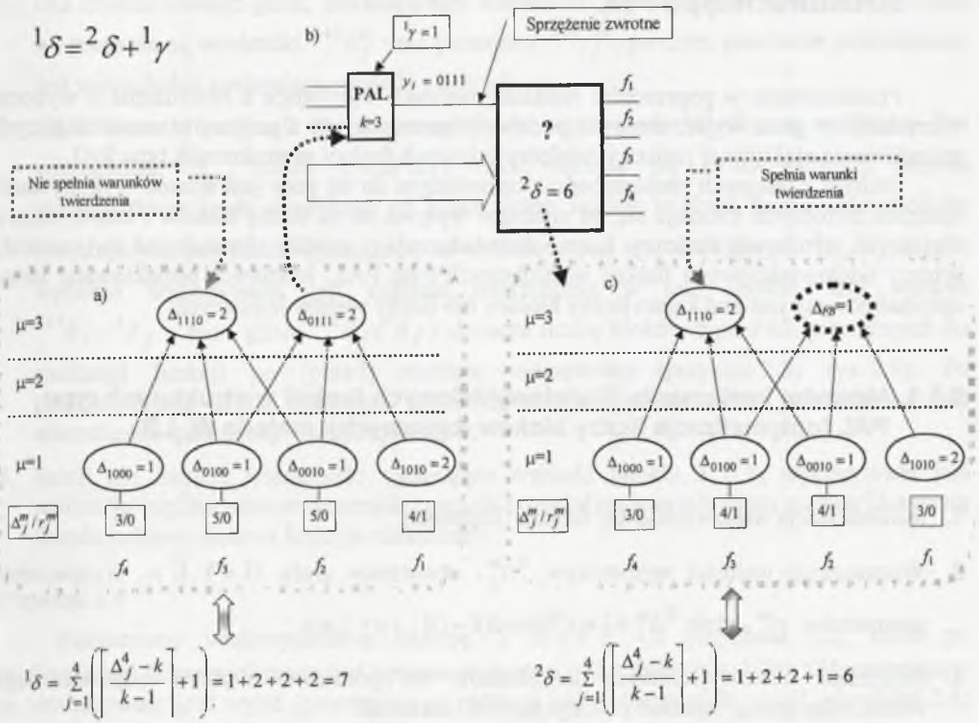
Rozważmy wybór wierzchołków 3 rzędu:

1. $\Delta_{1110} \Rightarrow \mathbf{R} = \{r_4^4, r_3^4, r_2^4\} \Rightarrow$ w zbiorze reszt nie istnieje reszta r_x^4, dla której $0 < r_x^4 < \Delta_{1110} < k$.

   Wniosek: wybór wierzchołka Δ_{1110} (realizacja implikantów skojarzonych z tym wierzchołkiem) doprowadzi do zwiększenia liczby bloków logicznych typu PAL.

2. $\Delta_{0111} \Rightarrow \mathbf{R} = \{r_3^4, r_2^4, r_1^4\} \Rightarrow$ w zbiorze reszt istnieje jedna reszta $r_1^4 = 1$, dla której $0 < r_1^4 < \Delta_{0111} < k$.

   Wniosek: wybór wierzchołka Δ_{0111} nie zwiększa liczby bloków logicznych typu PAL, dodatkowo powoduje, że po redukcji grafu wierzchołek Δ_{1110} spełnia warunki minimalizacji (rys. 2.6.c).



Rys. 2.6. a) Przykład zredukowanego grafu wyjść; b) Realizacja implikantów opisanych przez wierzchołek 3 rzędu Δ_{0111} za pomocą 3-iloczynowego bloku logicznego typu PAL; c) Graf po redukcji związanej z realizacja implikantów opisanych przez wierzchołek Δ_{0111}

Fig. 2.6. a) Example of a reduced graph of outputs, b) Implementation of implicants defined by third-range node Δ_{0111} by PAL-based logic block consisting of three terms; c) Graph of outputs after reduction related to implementation of implicants defined by node Δ_{0111}

2.5. Algorytmy implementacji wielowyjściowych funkcji w strukturach typu PAL

Przedstawione w poprzednim rozdziale wnioski, wynikające z twierdzenia o wyborze wierzchołków grafu wyjść, stanowią podstawę opracowanych algorytmów umożliwiających poszukiwanie efektywnej realizacji wielowyjściowych funkcji w strukturach typu PAL.

Jednym z istotnych problemów nie omawianym do tej pory jest sposób wykonywania sprzężeń zwrotnych. Okazuje się, że znacząco wpływa on na liczbę bloków i liczbę warstw logicznych wynikowej struktury. Liczne doświadczenia pozwoliły sformułować dwie metody syntezy wielowyjściowych funkcji w strukturach typu PAL, w których projektowany układ optymalizowany jest pod kątem liczby bloków lub liczby warstw logicznych.

2.5.1. Algorytm implementacji wielowyjściowych funkcji w strukturach typu PAL (optymalizacja liczby bloków logicznych); metoda W_LB

1. Minimalizacja wielowyjściowej funkcji (Espresso)
2. Wyznaczenie wartości wyróżników ${}^0\Delta_j^m$, utworzenie grafu $G<\mathbf{Y},\vec{\mathbf{U}}>$, wyznaczenie parametrów r_j^m, gdzie ${}^0\Delta_j^m - 1 \equiv r_j^m (\mathrm{mod}(k-1))$; $j \in <1,m>$
3. Realizacja wielowyjściowych implikantów na podstawie wyboru odpowiedniego wierzchołka grafu, zgodnie z następującymi zasadami:
   a) rozpatrywane są wierzchołki, dla których spełniona jest zależność $2 \le \mu({}^i\Delta_y) = \max$,
   b) wybierany jest wierzchołek, dla którego ${}^i\Delta_y \ge k$,
   c) jeżeli nie istnieje wierzchołek spełniający warunek b, to wybierany jest wierzchołek, dla którego $0 < r_a^{\mu({}^i\Delta_y)} < {}^i\Delta_y$ i $0 < r_b^{\mu({}^i\Delta_y)} < {}^i\Delta_y$,
   d) jeżeli nie istnieje wierzchołek spełniający warunek c, to wybierany jest wierzchołek ${}^i\Delta_y$, dla którego w zbiorze reszt $\mathbf{R} = \{r_j^{\mu({}^i\Delta_y)};\ j \in <1, \mu({}^i\Delta_y)>\}$ istnieje tylko jedna reszta $r_x^{\mu({}^i\Delta_y)}$ spełniająca warunek $0 < r_x^{\mu({}^i\Delta_y)} < {}^i\Delta_y$; (nie zmienia to liczby potrzebnych bloków logicznych typu PAL, lecz może doprowadzić do minimalizacji w kolejnym kroku - przykład 2.5; rys. 2.6).
4. Jeżeli warunki punktu 3 doprowadziły do wyboru wierzchołka, to realizowane są implikanty skojarzone z wybranym wierzchołkiem, po czym następuje redukcja grafu. Wprowadzane są wierzchołki, reprezentujące sprzężenia zwrotne, w gałęzie łączące wybrany wierzchołek z wierzchołkami niższych rzędów. Sprzężenia te wprowadzane są w taki sposób, aby w miarę możliwości wykorzystywały wolne iloczyny w blokach

logicznych typu PAL, realizujących implikanty skojarzone z wierzchołkami niższych rzędów, połączone z wybranym wierzchołkiem (przykład 2.7; rys. 2.8).

5. Dla zredukowanego grafu, zawierającego wierzchołki reprezentujące sprzężenia zwrotne, wyznaczane są wyróżniki ${}^{i+1}\Delta_j^m$ oraz parametry ${}^{i+1}r_j^m$, po czym ponownie poszukiwany jest wierzchołek spełniający warunki punktu 3.
6. Jeżeli warunki punktu 3 nie doprowadziły do wyboru wierzchołka, to wybierany jest wierzchołek ${}^i\Delta_y$, rzędu $3 \le \mu({}^i\Delta_y) = \max$, łączący się z co najmniej jednym wierzchołkiem rzędu większego od jeden. Jeżeli istnieje większa liczba wierzchołków spełniających powyższe wymagania, to wybierany jest ten, dla którego ${}^i\Delta_y = \min$. Po wyborze wykonywane jest rozbicie wierzchołka w taki sposób, aby wartość ${}^{i+1}\delta_f - {}^i\delta_f = \min$, gdzie ${}^{i+1}\delta_f({}^i\delta_f)$ oznacza liczbę bloków typu PAL potrzebnych do realizacji funkcji po (przed) rozbiciu wierzchołka (przykład 2.8; rys. 2.9). Po przekształceniu grafu, będącego wynikiem rozbicia wierzchołka, poszukiwany jest wierzchołek spełniający warunki punktu 3.
7. Jeżeli nie istnieją wierzchołki spełniające warunki punktu 3 i 6, wykonywana jest realizacja implikantów przyporządkowanych pozostałym wierzchołkom metodą klasyczną (każda jednowyjściowa funkcja oddzielnie).

Przykład 2.6

Rozpatrzmy wielowyjściową funkcję $f: B^5 \to B^4$ z przykładu 2.2, która po minimalizacji (Espresso) może być opisana w postaci pliku *f.pla* (rys. 2.7a). Odpowiadający jej zredukowany graf wyjść (powstający po redukcji pierwotnego grafu wyjść - przykład 2.4; rys. 2.3) przedstawiony jest na rys. 2.7.b.

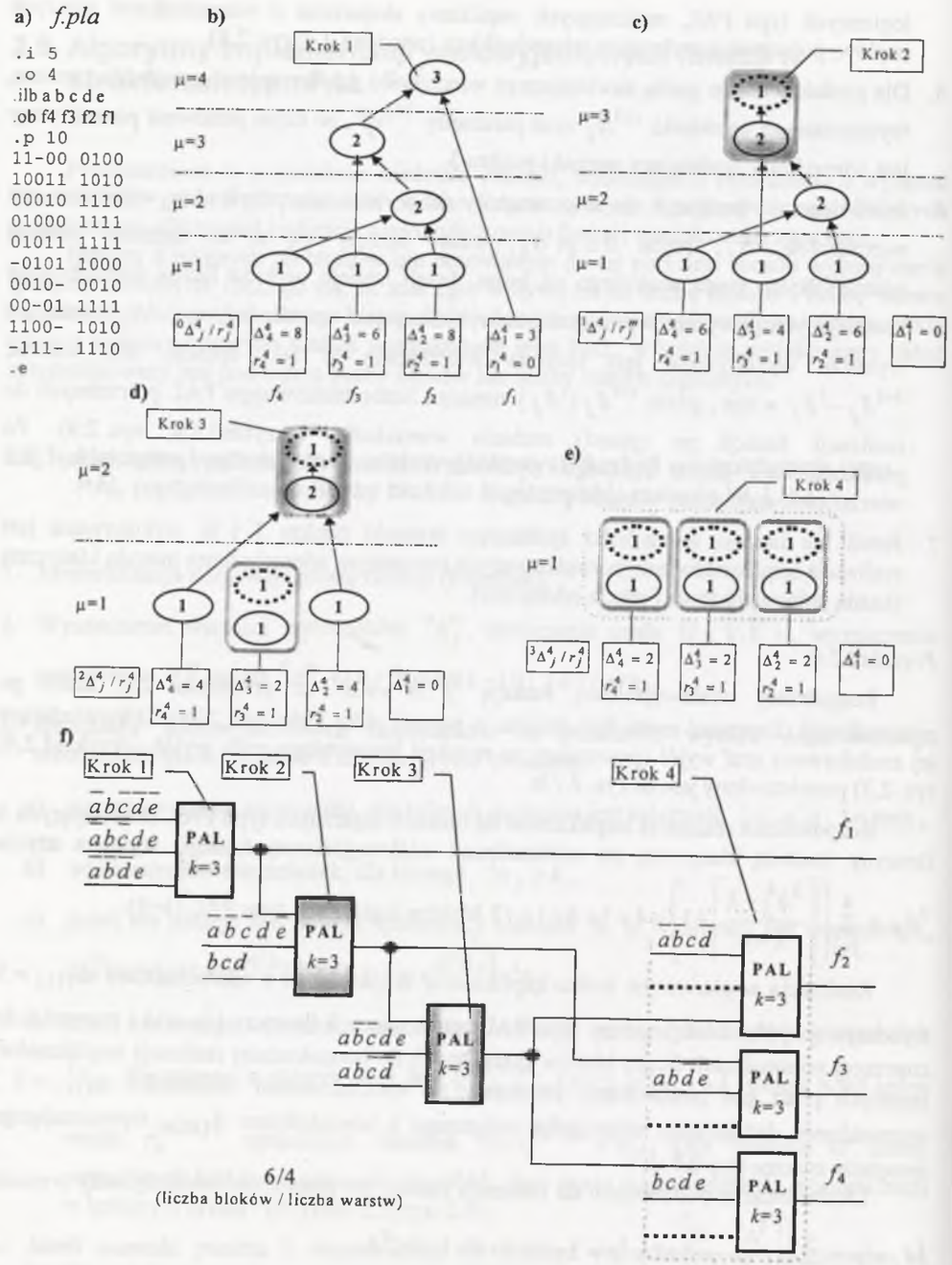
Bezpośrednia realizacja implikantów na blokach logicznych typu PAL zawierających 3 iloczyny (metodą klasyczną; po minimalizacji wielowyjściowej funkcji) wymaga użycia

$${}^0\delta_f = \sum_{j=1}^{4}\left(\left\lceil \frac{{}^0\Delta_j^4 - k}{k-1} \right\rceil + 1\right) = 4+3+4+1 = 12$$ bloków logicznych typu PAL (k=3).

Realizacja w pierwszym kroku implikantów skojarzonych z wierzchołkiem $\Delta_{1111} = 3$ wykorzystuje jeden blok logiczny typu PAL, zawierający 3 iloczyny (${}^1\gamma = 1$) i prowadzi do znaczącej minimalizacji liczby bloków potrzebnych do bezpośredniej realizacji implikantów, opisanych przez graf zredukowany, powstający po wyeliminowaniu wierzchołka $\Delta_{1111} = 3$ i wprowadzeniu dodatkowego wierzchołka związanego z wierzchołkiem Δ_{1110}, reprezentującego sprzężenie zwrotne (rys. 2.7.c).

Liczba bloków potrzebnych do realizacji funkcji po pierwszym kroku syntezy wynosi:

$${}^1\delta_f + {}^1\gamma = \sum_{j=1}^{4}\left(\left\lceil \frac{{}^1\Delta_j^4 - k}{k-1} \right\rceil + 1\right) + {}^1\gamma = 3+2+3+0+1 = 9 \le {}^0\delta_f$$



Rys. 2.7. Przykład realizacji funkcji (metoda W_LB)

Fig. 2.7. Example of the function implementation (the W_LB method)

W kolejnych krokach syntezy realizowane są implikanty odpowiadające wierzchołkom Δ_{1110}, Δ_{1010} (rys.2.7.c,d). Realizacja tych implikantów prowadzi do dalszej redukcji niezbędnych bloków logicznych typu PAL i wymaga użycia liczby bloków wyznaczonych z poniższych zależności:

$$ {}^2\delta_f + {}^2\gamma = \sum_{j=1}^{4}\left(\left\lceil \frac{{}^2\Delta_j^m - k}{k-1} \right\rceil + 1\right) + {}^2\gamma = 2+1+2+0+1 = 6 \le {}^1\delta_f , $$

$$ {}^3\delta_f + {}^3\gamma = \sum_{j=1}^{4}\left(\left\lceil \frac{{}^3\Delta_j^m - k}{k-1} \right\rceil + 1\right) + {}^3\gamma = 1+1+1+0+1 = 4 \le {}^1\delta_f $$

W ostatnim kroku syntezy realizowane są wierzchołki 1 rzędu (rys. 2.7e). Ostateczna realizacja wykorzystująca $\delta_f = {}^3\delta_f + \sum_{i=1}^{3} {}^i\gamma = 6 < \delta_f^1 = 12$ bloków logicznych typu PAL zawierających 3 iloczyny przedstawiona jest na rys. 2.7.f.

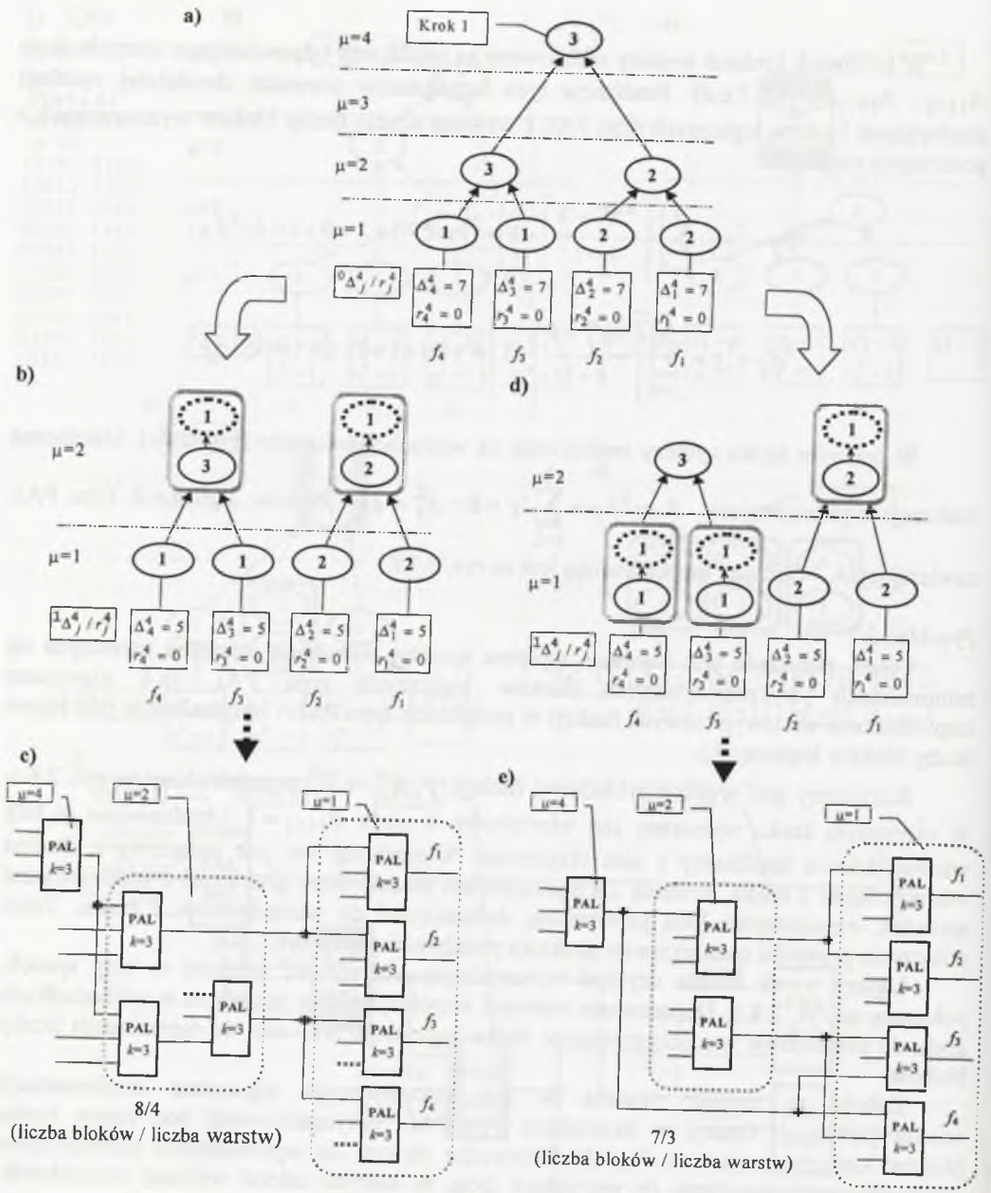
Przykład 2.7

Celem przykładu jest ilustracja wpływu sposobu wykonania sprzężeń zwrotnych na minimalizację wykorzystywanych bloków logicznych typu PAL (p.4 algorytmu implementacji wielowyjściowych funkcji w strukturach typu PAL - optymalizacja pod kątem liczby bloków logicznych).

Rozważmy graf wyjść przykładowej funkcji $f: B^n \to B^4$ przedstawiony na rys. 2.8.a. W pierwszym kroku wybierany jest wierzchołek 4 rzędu $\Delta_{1111} = 3$ i realizowane są trzy wielowyjściowe implikanty z nim skojarzone. Wierzchołek ten jest połączony z dwoma wierzchołkami 2 rzędu. Rysunek 2.8 b przedstawia zredukowany graf wyjść z wierzchołkami sprzężeń, oznaczonymi linią przerywaną, dołączonymi do wierzchołków 2 rzędu. Takie połączenie prowadzi ostatecznie do struktury przedstawionej na rys. 2.8 c.

Lepszy wynik można uzyskać wprowadzając wierzchołki sprzężeń w inny sposób, pokazany na rys. 2.8 d. Dopasowanie wartości współczynników zawartych w wierzchołkach grafu do parametrów wykorzystywanego bloku logicznego prowadzi do ograniczenia liczby bloków.

Zasada ta została zawarta w p.4 opracowanego algorytmu implementacji wielowyjściowych funkcji w strukturach typu PAL, optymalizowanej pod kątem liczby bloków logicznych (metoda W_LB). Sprowadza się ona do wprowadzania wierzchołków reprezentujących sprzężenia do wszystkich dróg, do których należał wybrany wierzchołek. Wierzchołki reprezentujące sprzężenia umieszczane są w taki sposób, aby w miarę możliwości nie zwiększać liczby bloków potrzebnych do realizacji implikantów skojarzonych z wierzchołkiem, do którego został dołączony wierzchołek reprezentujący sprzężenie. Ostateczna realizacja grafu z rys. 2.8 d, wykorzystująca tylko 7 bloków logicznych typu PAL zawierających 3 iloczyny, została przedstawiona na rys. 2.8 e.



Rys. 2.8. Wpływ opisu sprzężeń zwrotnych na ostateczną realizację

Fig. 2.8. Impact of feedback description on final implementation

Przykład 2.8

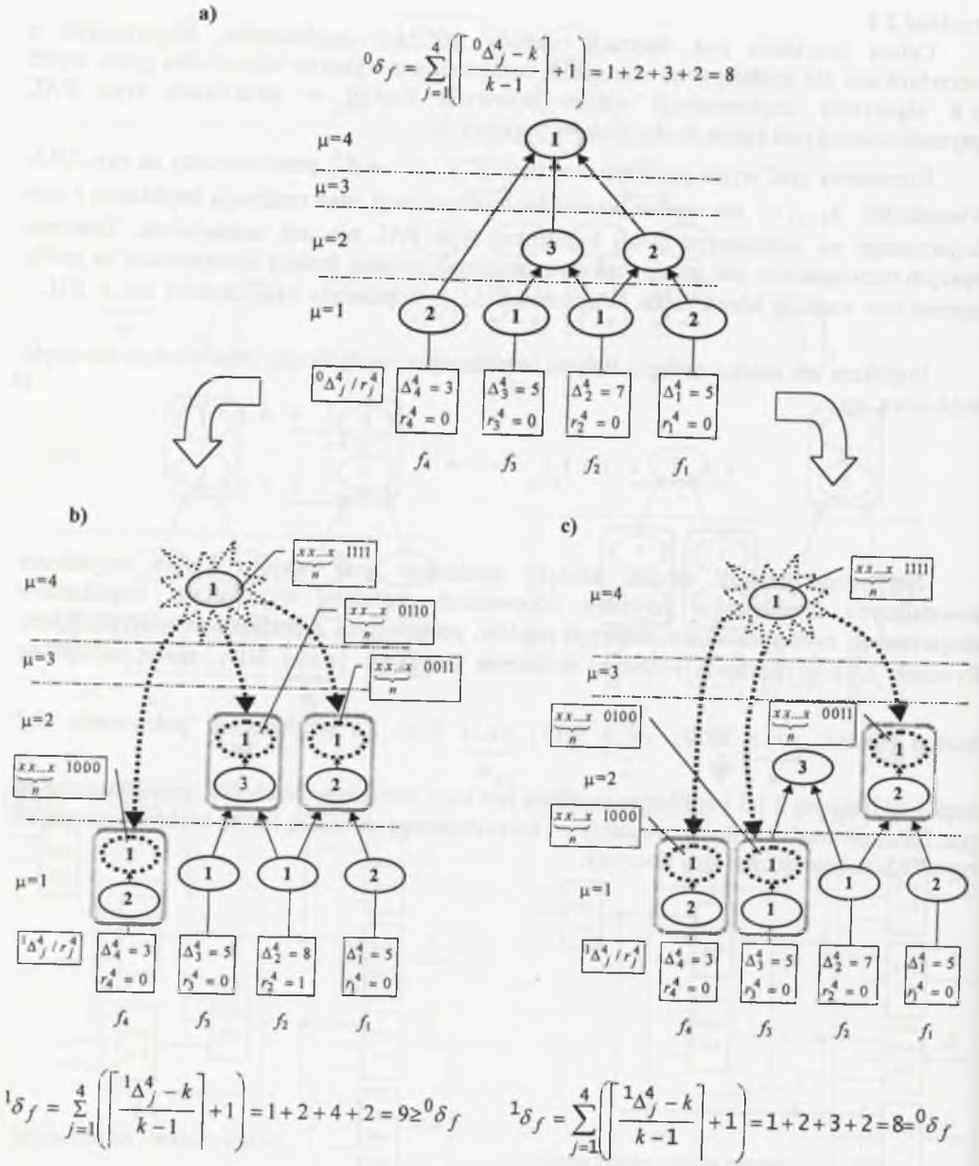
Celem przykładu jest ilustracja metody realizacji implikantów, skojarzonych z wierzchołkiem nie spełniającym warunków twierdzenia o wyborze wierzchołka grafu wyjść (p. 6 algorytmu implementacji wielowyjściowych funkcji w strukturach typu PAL optymalizowanej pod kątem liczby bloków logicznych).

Rozważmy graf wyjść przykładowej funkcji $f: B^n \rightarrow B^4$ przedstawiony na rys. 2.9.a. Wierzchołek $\Delta_{1111}=1$ nie spełnia warunków minimalizacji, stąd realizacja implikantu z nim skojarzonego na oddzielnym bloku logicznym typu PAL nie jest uzasadniona. Znacznie lepszym rozwiązaniem jest przekształcenie zminimalizowanej funkcji zobrazowane na grafie poprzez tzw. rozbicie wierzchołka. Wierzchołek Δ_{1111} odpowiada implikantowi $\underbrace{xx...x}_{n}$ 1111.

Implikant ten można zastąpić innymi implikantami modyfikując odpowiednio ich część wyjściową, np.:

$$\underbrace{x\,x\,.\,.\,.\,x}_{n}\ 1\ 1\ 1\ 1 \longrightarrow \begin{cases} \underbrace{x\,x\,.\,.\,.\,x}_{n}\ 0\ 0\ 1\ 1 \\ \underbrace{x\,x\,.\,.\,.\,x}_{n}\ 1\ 1\ 0\ 0 \end{cases}$$

Najlepsze rozbicie można znaleźć analizując graf wyjść. Części wyjściowe powstających implikantów powinny odpowiadać częściom wyjściowym implikantów skojarzonych z wierzchołkami niższych rzędów, połączonych z rozbijanym wierzchołkiem. Rysunek 2.9.b przedstawia rozbicie, w którym implikant $\underbrace{xx...x}_{n}$ 1111 został zastąpiony trzema innymi: $\underbrace{xx...x}_{n}$ 1000; $\underbrace{xx...x}_{n}$ 0110; $\underbrace{xx...x}_{n}$ 0011. Ze względu na "pokrywanie się" części wyjściowej 2 i 3 implikantu możliwe jest inne rozbicie wierzchołka, przedstawione na rys. 2.9.c, prowadzące do rozwiązania wykorzystującego mniejszą liczbę bloków logicznych typu PAL zawierających trzy iloczyny.



Rys. 2.9. Metody rozbicia wierzchołka

Fig. 2.9. The methods where a node is split

2.5.2. Algorytm implementacji wielowyjściowych funkcji w strukturach typu PAL (optymalizacja liczby warstw logicznych); metoda W_LW

1. Minimalizacja wielowyjściowej funkcji (Espresso)
2. Wyznaczenie wartości wyróżników $^0\Delta_j^m$, utworzenie grafu $G<\mathbf{Y},\vec{\mathbf{U}}>$, wyznaczenie parametrów r_j^m, gdzie $^0\Delta_j^m - 1 \equiv r_j^m \pmod{(k-1)}$; $j \in <1,m>$
3. Realizacja wielowyjściowych implikantów na podstawie wyboru odpowiedniego wierzchołka grafu zgodnie z następującymi zasadami:
   a) rozpatrywane są wierzchołki, dla których spełniona jest zależność $2 \leq \mu(^i\Delta_y)$, począwszy od wierzchołków, dla których $\mu(^i\Delta_y) = \max$,
   b) wybierany jest wierzchołek, dla którego $^i\Delta_y \geq k$,
   c) jeżeli nie istnieje wierzchołek spełniający warunek b, to wybierany jest wierzchołek, dla którego $0 < r_a^{\mu(^i\Delta_y)} < {}^i\Delta_y$ i $0 < r_b^{\mu(^i\Delta_y)} < {}^i\Delta_y$,
   d) jeżeli nie istnieje wierzchołek spełniający warunek c, to wybierany jest wierzchołek $^i\Delta_y$, dla którego w zbiorze reszt $\mathbf{R} = \{r_j^{\mu(^i\Delta_y)};\ j \in <1, \mu(^i\Delta_y)>\}$ istnieje tylko jedna reszta $r_x^{\mu(^i\Delta_y)}$ spełniająca warunek $0 < r_x^{\mu(^i\Delta_y)} < {}^i\Delta_y$; (nie zmienia to liczby potrzebnych bloków logicznych typu PAL, lecz może doprowadzić do minimalizacji w kolejnym kroku - przykład 2.5; rys. 2.6)
4. Jeżeli warunki punktu 3 doprowadziły do wyboru wierzchołka, to realizowane są implikanty skojarzone z wybranym wierzchołkiem, po czym następuje redukcja grafu. Wprowadzane są wierzchołki zerowego rzędu, reprezentujące sprzężenia zwrotne, w gałęzie łączące wybrany wierzchołek z wierzchołkami niższych rzędów (przykład 2.9, rys. 2.10).
5. Dla zredukowanego grafu, zawierającego wierzchołki reprezentujące sprzężenia zwrotne, wyznaczane są wyróżniki $^{i+1}\Delta_j^m$ oraz parametry $^{i+1}r_j^m$, po czym ponownie poszukiwany jest wierzchołek spełniający warunki punktu 3.
6. Jeżeli nie istnieją wierzchołki spełniające warunki punktu 3, wykonywana jest realizacja implikantów przyporządkowanych pozostałym wierzchołkom metodą klasyczną (każda funkcja oddzielnie).

Przykład 2.9

Rozpatrzmy wielowyjściową funkcję $f: B^5 \rightarrow B^4$ analizowaną w przykładzie 2.6 (metoda W_LB), opisaną w postaci pliku *f.pla* (rys. 2.10a). Odpowiadający jej zredukowany graf wyjść przedstawiony jest na rys. 2.10b.

Realizacja w pierwszym kroku implikantów skojarzonych z wierzchołkiem $\Delta_{1111}=3$ wykorzystuje jeden blok logiczny typu PAL, zawierający 3 iloczyny ($^1\gamma=1$) i prowadzi do znaczącej minimalizacji liczby bloków potrzebnych do bezpośredniej realizacji implikantów, opisanych przez graf zredukowany powstający po wyeliminowaniu wierzchołka $\Delta_{1111}=3$ i wprowadzeniu dodatkowych wierzchołków zerowego rzędu, reprezentujących sprzężenia zwrotne (rys. 2.10c).

Liczba bloków potrzebnych do realizacji implikantów skojarzonych z wybranym wierzchołkiem ($^1\gamma$), w sumie z liczbą bloków ($^1\delta_f$) potrzebnych do realizacji funkcji po 1 kroku syntezy, opisanej za pomocą zmodyfikowanego grafu wyjść (rys.2.10.c), wynosi:

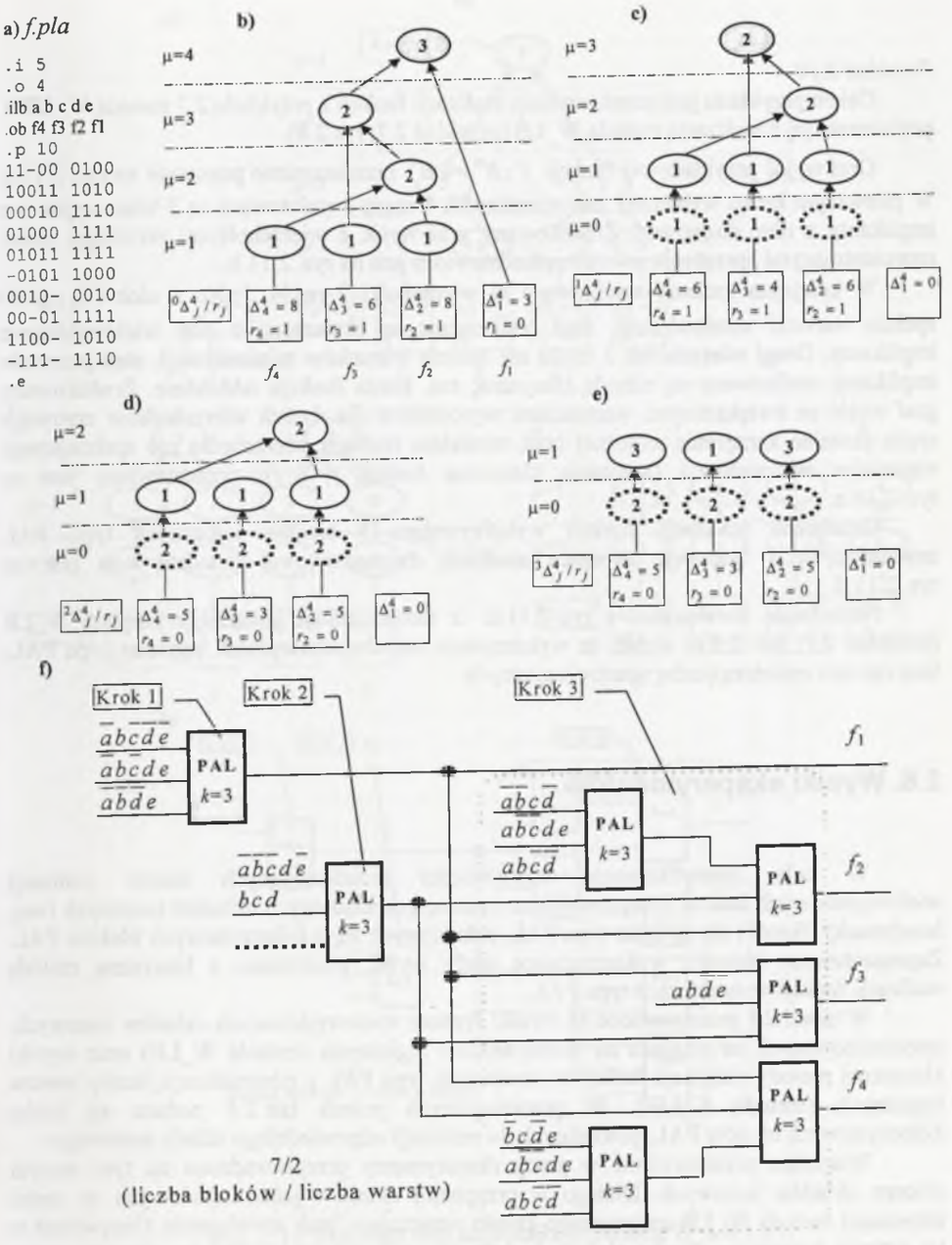
$$^1\delta_f + {}^1\gamma = \sum_{j=1}^{4}\left(\left\lceil \frac{^1\Delta_j^4 - k}{k-1}\right\rceil + 1\right) + {}^1\gamma = 3+2+3+0+1 = 9 \le {}^0\delta_f = \sum_{j=1}^{4}\left(\left\lceil \frac{^0\Delta_j^4 - k}{k-1}\right\rceil + 1\right) = 12$$

W kolejnym etapie syntezy realizowane są implikanty odpowiadające wierzchołkowi Δ_{1110} (rys.2.10c) i wprowadzane są sprzężenia zwrotne objawiające się na zredukowanym grafie w postaci inkrementacji wartości wyróżników wierzchołków zerowego rzędu (rys.2.10d). Realizacja implikantów skojarzonych z wierzchołkiem Δ_{1110} prowadzi do dalszej redukcji niezbędnych bloków logicznych typu PAL.

$$^2\delta_f + {}^2\gamma = \sum_{j=1}^{4}\left(\left\lceil \frac{^2\Delta_j^m - k}{k-1}\right\rceil + 1\right) + {}^2\gamma = 2+1+2+0+1 = 6 \le {}^1\delta_f$$

Ponieważ wierzchołek Δ_{1010} nie spełnia warunków minimalizacji (twierdzenie o wyborze wierzchołka grafu wyjść; p.3 algorytmu implementacji wielowyjściowych funkcji w strukturach typu PAL optymalizowanej pod kątem liczby warstw logicznych - metoda W_LW), pozostałe implikanty są realizowane metodą klasyczną, tzn. każda funkcja oddzielnie (rys. 2.10e).

Ostateczna realizacja wykorzystująca $\delta_f = {}^2\delta_f + \sum_{i=1}^{2} {}^i\gamma = 7$ bloków logicznych typu PAL zawierających 3 iloczyny, będąca równocześnie strukturą dwuwarstwową, przedstawiona jest na rys. 2.10f.



Rys. 2.10. Przykład realizacji funkcji (metoda W_LW)

Fig. 2.10. Example of the function implementation (the W_LW method)

Przykład 2.10

Celem przykładu jest przedstawienie realizacji funkcji z przykładu 2.7 metodą W_LW i porównanie jej z realizacją metodą W_LB (przykład 2.7; rys.2.8).

Graf wyjść przykładowej funkcji $f: B^n \rightarrow B^4$ przedstawiono ponownie na rys. 2.11.a. W pierwszym kroku wybierany jest wierzchołek 4 rzędu i realizowane są 3 wielowyjściowe implikanty z nim skojarzone. Zredukowany graf wyjść z wierzchołkami zerowego rzędu, reprezentującymi sprzężenia zwrotne, przedstawiony jest na rys. 2.11.b.

W kolejnym krokach analizowane są wierzchołki 2 rzędu. Jeden z nich (Δ_{1100}=3) spełnia warunki minimalizacji, stąd realizowane są skojarzone z nim wielowyjściowe implikanty. Drugi wierzchołek 2 rzędu nie spełnia warunków minimalizacji, stąd pozostałe implikanty realizowane są metodą klasyczną, tzn. każda funkcja oddzielnie. Zredukowany graf wyjść ze zwiększonymi wartościami wyróżników dla dwóch wierzchołków zerowego rzędu (kolejne sprzężenia zwrotne) oraz wynikiem rozbicia wierzchołka nie spełniającego warunków minimalizacji (realizacja klasyczna funkcji f_1 i f_2) przedstawiony jest na rys. 2.11.c.

Ostateczna realizacja funkcji wykorzystująca 8 bloków logicznych typu PAL zawierających 3 iloczyny, tworząca strukturę dwuwarstwową, przedstawiona jest na rys. 2.11 d.

Porównując rozwiązanie z rys. 2.11.d z rozwiązaniem uzyskanym metodą W_LB (przykład 2.7; rys. 2.8.e) widać, że wykorzystuje ono dodatkowy blok logiczny typu PAL, lecz zawiera mniejszą liczbę warstw logicznych.

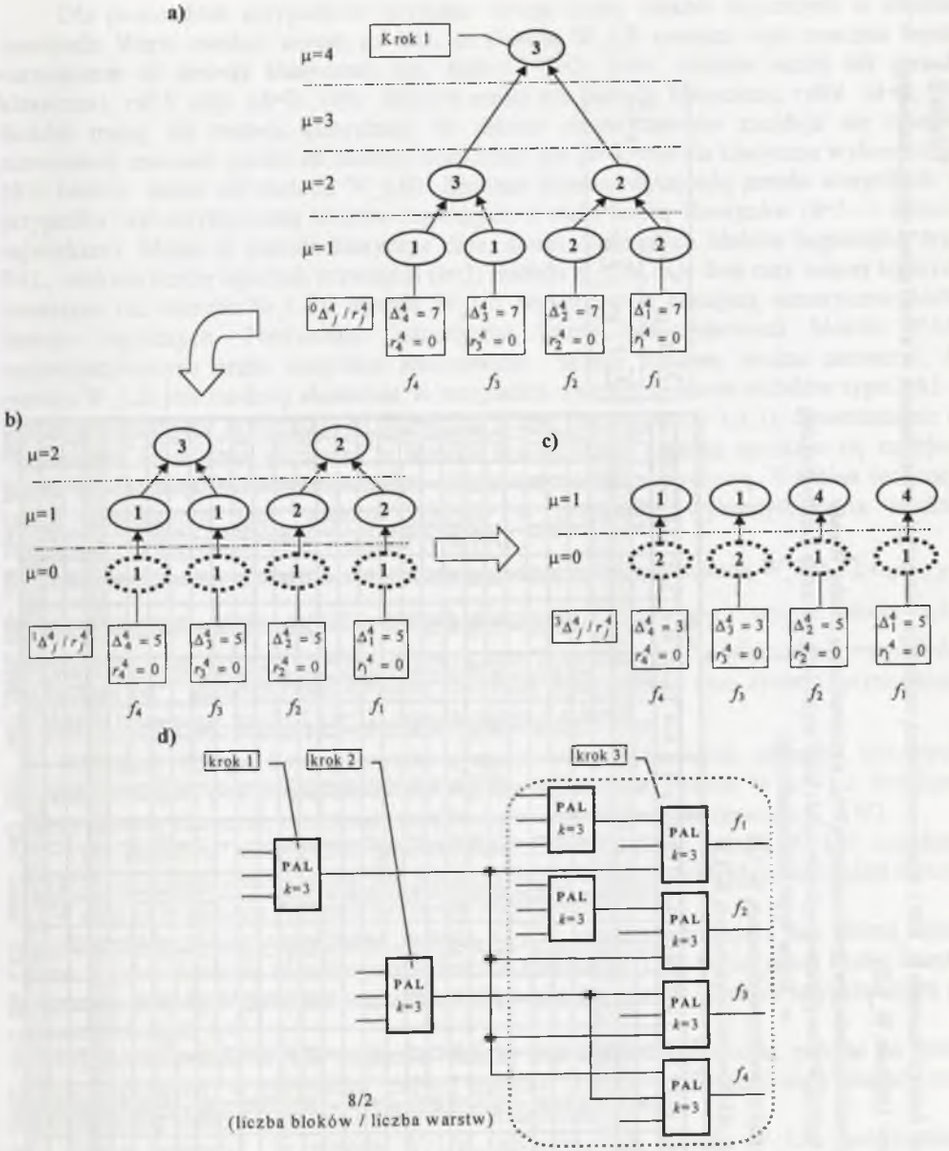
2.6. Wyniki eksperymentów

W celu zweryfikowania efektywności przedstawionych metod realizacji wielowyjściowych funkcji przeprowadzono syntezę przykładowych układów testowych (ang. benchmark) [Bench] dla struktur typu PAL, składających się z *k*-iloczynowych bloków PAL. Zaprezentowane metody, wykorzystujące grafy wyjść, porównano z klasyczną metodą realizacji funkcji w strukturach typu PAL.

W tabeli 2.1 przedstawione są wyniki syntezy wielowyjściowych układów testowych, optymalizowanych ze względu na liczbę bloków logicznych (metoda W_LB) oraz wyniki klasycznej metody realizacji funkcji w strukturach typu PAL z minimalizacją liczby warstw logicznych (metoda K_LW). W poszczególnych polach tab. 2.1 podane są liczby *k*-iloczynowych bloków PAL, potrzebnych do realizacji odpowiedniego układu testowego.

Wszystkie przedstawione w pracy eksperymenty przeprowadzono na tym samym zbiorze układów testowych. Dlatego w przypadku układów jednowyjściowych w części dotyczącej metody W_LB umieszczono kropki oznaczające brak rozwiązania. Oczywiście w tej sytuacji sumaryczna liczba bloków logicznych umieszczona w ostatnim wierszu tablicy 2.1 nie obejmuje układów jednowyjściowych zarówno dla metody W_LB, jak i K_LW.

W zbiorze 208 porównywanych przypadków metoda W_LB doprowadziła do 40 (19%) rozwiązań wykorzystujących mniejszą liczbę bloków logicznych niż metoda K_LW. Metodą K_LW uzyskano 74 (35%) lepsze rozwiązania.



Rys. 2.11. Przykład realizacji funkcji (metoda W_LW)

Fig. 2.11. Example of the function implementation (the W_LW method)

Tabela 2.1

Porównanie klasycznej realizacji funkcji (metoda K_LW) z zaproponowaną realizacją wielowyjściowej funkcji, w której układ optymalizowany jest pod względem liczby bloków (metoda W_LB)

metoda K_LW - Klasyczna metoda realizacji funkcji w strukturach typu PAL z minimalizacją **L**iczby **W**arstw
metoda W_LB - metoda realizacji funkcji **W**ielowyjściowej wykorzystująca grafy wyjść (optymalizacja **L**iczby **B**loków)
i - liczba wejść układu testowego; o - liczba wyjść układu testowego
$\sum t_{(W\_LB)}$ - suma czasów syntezy układów testowych metodą W_LB
$\sum t_{(K\_LW)}$ - suma czasów syntezy układów testowych metodą K_LW

	Układ testowy	i	o	K_LW k=3	k=4	k=5	k=6	k=7	k=8	k=12	k=16	W_LB k=3	k=4	k=5	k=6	k=7	k=8	k=12	k=16
1	5xp1	7	10	35	26	22	18	16	15	12	11	36	28	22	20	18	16	14	11
2	9sym	9	1	43	29	22	18	15	13	8	6	.	.	.	.	.	.	.	.
3	9symml	9	1	42	28	21	17	14	12	8	6	.	.	.	.	.	.	.	.
4	b12	15	9	24	16	14	13	10	10	9	9	.	.	.	.	.	.	.	.
5	bw	5	28	49	38	30	28	28	28	28	28	26	21	15	15	11	11	9	9
6	clip	9	5	73	49	38	30	26	22	14	12	116	83	66	56	48	45	33	33
7	con1	7	2	4	3	2	2	2	2	2	2	66	45	37	29	26	22	16	11
8	duke2	22	29	99	72	61	49	44	41	35	31	4	3	2	2	2	2	2	2
9	ex1010	10	10	242	165	124	101	85	74	49	38	103	80	66	59	51	47	35	32
10	f51m	8	8	37	27	22	18	16	15	11	10	243	165	126	100	86	75	50	37
11	inc	7	9	23	17	16	13	11	11	9	9	37	27	22	18	16	15	11	10
12	ldd	9	19	34	25	25	21	19	19	19	19	24	20	15	16	13	12	9	9
13	misex1	8	7	14	12	8	7	7	7	7	7	37	29	25	24	23	20	19	19
14	misex2	25	18	19	19	18	18	18	18	18	18	15	10	9	8	9	8	7	7
15	pcle	19	9	10	9	9	9	9	9	9	9	19	19	18	18	18	18	18	18
16	rd53	5	3	15	11	8	6	6	6	4	3	10	9	9	9	9	9	9	9
17	rd73	7	3	70	47	36	29	24	20	14	11	15	11	8	7	6	6	5	3
18	rd84	8	4	142	95	72	58	49	42	28	21	63	44	32	26	23	19	13	10
19	root	8	5	35	24	18	16	13	11	9	7	129	87	66	54	45	39	28	20
20	sao2	10	4	36	24	19	15	14	11	7	7	33	22	20	17	15	14	9	7
21	sqn	7	3	21	15	11	10	8	7	5	4	29	20	16	13	11	10	7	6
22	sqr6	6	12	31	24	20	16	16	15	12	11	21	15	12	10	8	7	5	4
23	squar5	5	8	14	11	9	9	9	8	8	8	33	26	21	20	16	17	12	11
24	table3	14	14	264	181	135	110	94	80	53	40	16	11	10	9	9	8	8	8
25	vg2	25	8	53	37	28	22	20	19	13	11	162	114	93	82	79	69	56	44
26	x2	10	7	13	10	10	7	7	7	7	7	53	37	28	22	20	19	13	11
27	xor5	5	1	8	5	4	3	3	3	2	1	14	11	9	8	8	7	7	7
28	z4ml	7	4	29	19	15	13	11	9	7	5	.	.	.	.	.	.	.	.
29	z5xp1	7	10	37	27	23	19	17	15	12	11	42	31	27	21	19	16	12	11
30	z9sym	9	1	42	28	21	17	14	12	8	6	.	.	.	.	.	.	.	.
	Sumaryczna liczba bloków:			1423	1003	793	657	579	521	401	349	1375	987	789	676	600	540	414	354

$$\frac{\sum t_{(W\_LB)}}{\sum t_{(K\_LW)}} \approx 1{,}25$$

Dla pozostałych przypadków uzyskano równą liczbę bloków logicznych w obydwu metodach. Warto zwrócić uwagę na fakt, że metoda W_LB czasami daje znacznie lepsze rozwiązania od metody klasycznej, np.: *table3* (k=3; 39% bloków mniej niż metoda klasyczna), *rd73, clip* (k=3; 10% bloków mniej niż metoda klasyczna), *rd84* (k=3; 9% bloków mniej niż metoda klasyczna). W zbiorze eksperymentów znajduje się również rozwiązanie znacznie gorsze od metody klasycznej: *bw* (k=3; metoda klasyczna wykorzystuje 58% bloków mniej niż metoda W_LB). Znaczne różnice występują przede wszystkich w przypadku wykorzystywania bloków logicznych z małą liczbą iloczynów (k=3 - - różnice największe). Mimo iż metoda klasyczna daje, nawet dla małych bloków logicznych typu PAL, większą liczbę lepszych rozwiązań (k=3; metoda K_LW daje dwa razy więcej lepszych rozwiązań niż metoda W_LB), metoda W_LB wykorzystuje mniejszą sumaryczną liczbę bloków logicznych. Porównując sumaryczną liczbę k-iloczynowych bloków PAL, wykorzystywanych przez wszystkie analizowane układy testowe, można zauważyć, że metoda W_LB jest bardziej skuteczna w przypadku wykorzystywania układów typu PAL z blokami logicznymi zawierającymi niewielką liczbę iloczynów (k=3,4,5). Spostrzeżenie to wyczuwalne intuicyjnie, ponieważ w wyniku minimalizacji częściej uzyskuje się mniejsze grupy implikantów wielowyjściowych z identyczną częścią wyjściową. Wspólna realizacja takich implikantów daje większe korzyści w przypadku wykorzystywania struktur zawierających niewielkie bloki logiczne typu PAL.

Stosunek sumarycznego czasu syntezy układów testowych metodą W_LB ($\sum t_{(W\_LB)}$) do sumarycznego czasu syntezy metodą klasyczną ($\sum t_{(K\_LW)}$) wynosi około 1,25. Bezwzględny czas syntezy (z wyłączeniem czasu minimalizacji) w większości przypadków jest krótszy od 1 sekundy (Intel Celeron 333MHz). Maksymalny czas syntezy, występujący dla układu testowego *ex1010* (k=3), wynosi około 4 sekund.

W tabeli 2.2 porównano wyniki syntezy wielowyjściowych układów testowych, optymalizowanych pod względem liczby warstw logicznych (metoda W_LW) z wynikami syntezy metodą klasyczną z minimalizacją liczby warstw logicznych (metoda K_LW).

Tym razem w zbiorze 208 porównywanych przypadków metodą W_LW uzyskano 39 (19%) rozwiązań wykorzystujących mniejszą liczbę bloków logicznych, natomiast metoda K_LW dała aż w połowie przypadków lepsze rozwiązania.

Wszystkie układy uzyskiwane metodą K_LW zawierały mniejszą lub równą liczbę warstw w odniesieniu do układów uzyskiwanych metodą W_LW. Sumaryczna liczba warstw po syntezie metodą W_LW jest większa o 18% od liczby warstw układów uzyskiwanych po syntezie metodą K_LW.

Mimo iż metoda W_LW znacznie rzadziej daje lepsze rozwiązania, patrząc na liczbę bloków logicznych, w przypadku małych bloków (k=3, k=4) wykorzystuje sumarycznie mniejszą liczbę bloków logicznych niż metoda klasyczna.

Warto zauważyć, że metoda W_LW, podobnie jak metoda W_LB, jest bardziej skuteczna w przypadku wykorzystywania układów typu PAL zawierających niewielkie bloki logiczne (k=3,4).

Metoda W_LW jest nieznacznie szybsza od metody W_LB. Stosunek sumarycznego czasu syntezy układów testowych metodą W_LW ($\sum t_{(W\_LW)}$) do sumarycznego czasu syntezy metodą klasyczną ($\sum t_{(K\_LW)}$) wynosi około 1,15. Maksymalny, bezwzględny czas syntezy jest tak jak poprzednio na poziomie pojedynczych sekund.

Tabela 2.2

Porównanie klasycznej realizacji funkcji (metoda K_LW) z zaproponowaną realizacją wielowyjściowej funkcji, w której układ optymalizowany jest pod względem liczby warstw logicznych (metoda W_LW)

metoda K_LW - **K**lasyczna metoda realizacji funkcji w strukturach typu PAL z minimalizacją **L**iczby **W**arstw

metoda W_LW - metoda realizacji funkcji **W**ielowyjściowej wykorzystująca grafy wyjść (optymalizacja **L**iczby **W**arstw logicznych)

i - liczba wejść układu testowego; o - liczba wyjść układu testowego; B - liczba bloków; W - liczba warstw

$\Sigma t_{(W\_LW)}$ - suma czasów syntezy układów testowych metodą W_LW

$\Sigma t_{(K\_LW)}$ - suma czasów syntezy układów testowych metodą K_LW

$$\frac{\sum t_{(W\_LW)}}{\sum t_{(K\_LW)}} \approx 1{,}15$$

			K_LW																W_LW											
			k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16		k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16	
	Układ testowy	i	o	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W
1	5xp1	7	10	35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2	38	5	28	3	23	3	20	2	18	2	15	2	12	2	11	2
2	9sym	9	1	43	5	29	4	22	3	18	3	15	3	13	3	8	2	6	2																
3	9symml	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2																
4	b12	15	9	24	3	16	2	14	2	13	2	10	2	10	2	9	1	9	1	26	3	19	2	15	2	14	2	11	2	9	1	9	1	9	1
5	bw	5	28	49	2	38	2	30	2	28	1	28	1	28	1	28	1	28	1	118	3	85	3	67	2	57	2	49	2	46	2	33	2	33	2
6	clip	9	5	73	4	49	3	38	3	30	3	26	2	22	2	14	2	12	2	67	6	46	5	37	4	29	4	26	3	21	3	16	2	11	2
7	con1	7	2	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1
8	duke2	22	29	99	3	72	3	61	2	49	2	44	2	41	2	35	2	31	2	117	4	82	3	70	2	57	2	49	2	46	2	37	2	32	2
9	ex1010	10	10	242	4	165	3	124	3	101	3	85	3	74	2	49	2	38	2	243	5	165	4	126	4	100	3	86	3	74	2	50	2	37	2
10	f51m	8	8	37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2	37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2
11	inc	7	9	23	3	17	2	16	2	13	2	11	2	11	2	9	1	9	1	27	3	20	2	16	2	14	2	13	2	11	2	9	1	9	1
12	ldd	9	19	34	2	25	2	25	2	21	2	19	1	19	1	19	1	19	1	38	2	29	2	25	2	24	2	23	2	20	2	19	1	19	1
13	misex1	8	7	14	2	12	2	8	2	7	1	7	1	7	1	7	1	7	1	18	3	14	2	11	3	10	2	9	2	9	2	7	1	7	1
14	misex2	25	18	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1
15	pcle	19	9	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1
16	rd53	5	3	15	3	11	2	8	2	6	2	6	2	6	2	4	2	3	1	15	4	11	2	8	2	7	2	6	2	6	2	5	2	3	1
17	rd73	7	3	70	4	47	3	36	3	29	3	24	3	20	2	14	2	11	2	63	7	44	5	32	5	26	4	23	3	19	3	13	3	10	3
18	rd84	8	4	142	5	95	4	72	4	58	3	49	3	42	3	28	2	21	2	130	8	87	6	67	5	54	4	45	4	40	4	27	3	20	3
19	root	8	5	35	4	24	3	18	3	16	2	13	2	11	2	9	2	7	2	37	5	26	4	20	3	17	2	14	2	14	2	10	2	8	2
20	sao2	10	4	36	3	24	3	19	2	15	2	14	2	11	2	7	2	7	2	29	4	20	3	16	3	13	3	11	3	10	3	7	3	6	2
21	sqn	7	3	21	3	15	3	11	2	10	2	8	2	7	2	5	2	4	2	20	4	14	3	11	2	10	2	8	2	7	2	5	2	4	2
22	sqr6	6	12	31	3	24	2	20	2	16	2	16	2	15	2	12	2	11	1	34	3	25	2	21	2	18	2	16	2	16	2	12	2	11	1
23	squar5	5	8	14	2	11	2	9	2	9	2	9	2	8	1	8	1	8	1	16	2	11	2	10	2	9	2	9	2	8	1	8	1	8	1
24	table3	14	14	264	4	181	4	135	3	110	3	94	3	80	3	53	2	40	2	172	6	118	5	98	5	89	5	86	4	77	4	59	4	45	3
25	vg2	25	8	53	4	37	3	28	3	22	3	20	2	19	2	13	2	11	2	53	4	37	3	28	3	22	3	20	2	19	2	13	2	11	2
26	x2	10	7	13	2	10	2	10	2	7	1	7	1	7	1	7	1	7	1	16	2	11	2	10	2	8	2	8	2	7	1	7	1	7	1
27	xor5	5	1	8	2	5	2	4	2	3	2	3	2	3	2	2	2	1	1																
28	z4ml	7	4	29	4	19	3	15	3	13	2	11	2	9	2	7	2	5	2	29	4	19	3	15	3	13	2	11	2	9	2	7	2	5	2
29	z5xp1	7	10	37	3	27	3	23	2	19	2	17	2	15	2	12	2	11	2	43	5	31	3	26	3	21	3	18	2	16	2	12	2	11	2
30	z9sym	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2																
Sumaryczna liczba bloków:				1423		1003		793		657		579		521		401		349		1419		1000		803		679		604		545		417		356	
Sumaryczna liczba warstw logicznych:				79		67		58		52		49		46		42		40		101		77		69		62		57		54		48		44	

Z porównania dwóch zaproponowanych metod syntezy funkcji wielowyjściowych, wykorzystujących grafy wyjść (tab. 2.3) wynika, że zarówno jedną, jak i drugą metodą dają lepsze rozwiązania od metody klasycznej, porównywane pod względem liczby bloków logicznych. W przypadku metody W_LB liczba lepszych rozwiązań jest jednak prawie dwukrotnie większa. Rozpatrując sumaryczną liczbę *k*-loczynowych bloków PAL, potrzebnych do realizacji analizowanego zbioru układów testowych, metoda W_LB wykorzystuje od ułamków do 3% mniej bloków logicznych. Większe oszczędności uzyskuje się dla struktur zawierających niewielkie bloki logiczne typu PAL (*k*=3).

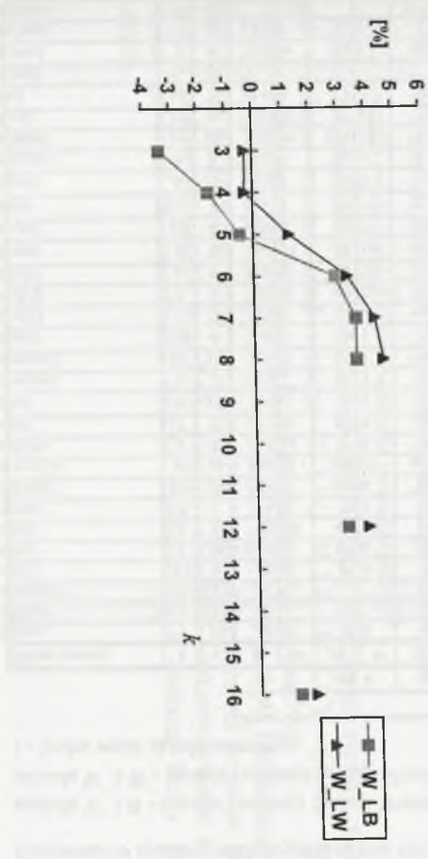
Metodą W_LW uzyskuje się jednak zawsze nie większą liczbę warstw logicznych od metody W_LB. Metoda W_LB prowadzi w wielu przypadkach do układów zawierających znacznie większą liczbę warstw, ponieważ w celu przyśpieszenia syntezy do realizacji grup implikantów zawierających identyczną część wyjściową zastosowano szybką klasyczną metodę realizacji funkcji w strukturach typu PAL, bez minimalizacji liczby warstw. Oczywiście istnieje możliwość wplecenia w metodę W_LB elementów minimalizujących liczbę warstw, lecz wydłużyłoby to czas syntezy doprowadzając mimo wszystko do rozwiązań gorszych pod względem liczby warstw od metody W_LW.

Podsumowaniem powyższych rozważań mogą być wykresy przedstawiające porównanie trzech metod: metody K_LW, W_LB i W_LW. Metody te porównano pod względem liczby bloków oraz liczby warstw. Sumaryczną liczbę bloków (warstw), uzyskiwaną dla różnych wielkości bloków logicznych typu PAL (*k*), odniesiono do sumarycznej liczby bloków (warstw) uzyskiwanej w metodzie klasycznej.

Wartości przedstawione na wykresie 2.1 wyznaczone zostały z zależności

$$\frac{x - \sum bloków_{(K\_LW)}}{\sum bloków_{(K\_LW)}} 100\%$$

, poprzez wstawienie za wartości x sumy liczby bloków, wykorzystywanych przez rozpatrywany zbiór układów testowych w metodzie W_LB ($\Sigma bloków_{(W\_LB)}$) oraz metodzie W_LW ($\Sigma bloków_{(W\_LW)}$), dla poszczególnych wartości *k*.



Wykres 2.1. Porównanie metod syntezy W_LB, W_LW z metodą K_LW pod względem liczby bloków (opis w tekście)

Tabela 2.3

Porównanie syntezy wielowyjściowych funkcji metodą W_LB oraz W_LW

metoda W_LB - metoda realizacji funkcji **W**ielowyjściowej wykorzystująca grafy wyjść (optymalizacja **L**iczby **B**loków)
metoda W_LW - metoda realizacji funkcji **W**ielowyjściowej wykorzystująca grafy wyjść (optymalizacja **L**iczby **W**arstw logicznych)
i - liczba wejść układu testowego; o - liczba wyjść układu testowego; B - liczba bloków; W - liczba warstw

			W_LB																W_LW													
			k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16		k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16	
	Układ testowy	i	o	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W
1	5xp1	7	10	36	8	28	6	22	5	20	4	18	3	16	3	14	3	11	2	38	5	28	3	23	3	20	2	18	2	15	2	12	2	11	2
2	9sym	9	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	9symml	9	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	b12	15	9	26	5	21	4	15	3	15	3	11	2	11	2	9	1	9	1	26	3	19	2	15	2	14	2	11	2	11	2	9	1	9	1
5	bw	5	28	116	9	83	6	66	5	56	4	48	3	45	3	33	2	33	2	118	3	85	3	67	2	57	2	49	2	46	2	33	2	33	2
6	clip	9	5	66	19	45	13	37	10	29	9	26	7	22	7	16	4	11	3	67	6	46	5	37	4	29	4	26	3	21	3	16	2	11	2
7	con1	7	2	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1
8	duke2	22	29	103	8	80	6	66	5	59	4	51	4	47	4	35	3	32	2	117	4	82	3	70	2	57	2	49	2	46	2	37	2	32	2
9	ex1010	10	10	243	29	165	20	126	15	100	12	86	10	75	9	50	6	37	4	243	5	165	4	126	4	100	3	86	3	74	2	50	2	37	2
10	f51m	8	8	37	11	27	8	22	6	18	5	16	4	15	4	11	2	10	2	37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2
11	inc	7	9	24	4	20	4	15	3	16	3	13	3	12	3	9	1	9	1	27	3	20	2	16	2	14	2	13	2	11	2	9	1	9	1
12	ldd	9	19	37	4	29	3	25	2	24	2	23	2	20	2	19	1	19	1	38	2	29	2	25	2	24	2	23	2	20	2	19	1	19	1
13	misex1	8	7	15	3	10	3	9	4	8	2	9	2	8	2	7	1	7	1	18	3	14	2	11	3	10	2	9	2	9	2	7	1	7	1
14	misex2	25	18	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1
15	pcle	19	9	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1
16	rd53	5	3	15	8	11	5	8	4	7	4	6	3	6	3	5	3	3	1	15	4	11	2	8	2	7	2	6	2	6	2	5	2	3	1
17	rd73	7	3	63	32	44	22	32	16	26	13	23	12	19	10	13	6	10	5	63	7	44	5	32	5	26	4	23	3	19	3	13	3	10	3
18	rd84	8	4	129	47	87	31	66	23	54	19	45	16	39	14	28	9	20	7	130	8	87	6	67	5	54	4	45	4	40	4	27	3	20	3
19	root	8	5	33	14	22	10	20	9	17	8	15	6	14	5	9	2	7	2	37	5	26	4	20	3	17	2	14	2	14	2	10	2	8	2
20	sao2	10	4	29	11	20	7	16	6	13	5	11	4	10	4	7	3	6	2	29	4	20	3	16	3	13	3	11	3	10	3	7	3	6	2
21	sqn	7	3	21	9	15	7	12	5	10	5	8	3	7	3	5	2	4	2	20	4	14	3	11	2	10	2	8	2	7	2	5	2	4	2
22	sqr6	6	12	33	6	26	5	21	4	20	3	16	3	17	3	12	2	11	1	34	3	25	2	21	2	18	2	16	2	16	2	12	2	11	1
23	squar5	5	8	16	4	11	3	10	2	9	2	9	2	8	1	8	1	8	1	16	2	11	2	10	2	9	2	9	2	8	1	8	1	8	1
24	table3	14	14	162	14	114	12	93	10	82	8	79	8	69	7	56	5	44	5	172	14	118	12	98	10	89	8	86	8	77	4	59	4	45	3
25	vg2	25	8	53	20	37	13	28	10	22	8	20	7	19	6	13	4	11	3	53	4	37	3	28	3	22	3	20	2	19	2	13	2	11	2
26	x2	10	7	14	3	11	2	9	2	8	2	8	2	7	1	7	1	7	1	16	2	11	2	10	2	8	2	8	2	7	1	7	1	7	1
27	xor5	5	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
28	z4ml	7	4	29	14	19	9	15	7	13	6	11	5	9	4	7	3	5	2	29	4	19	3	15	3	13	2	11	2	9	2	7	2	5	2
29	z5xp1	7	10	42	8	31	6	27	5	21	4	19	4	16	3	12	1	11	2	43	5	31	3	26	3	21	3	18	2	16	2	12	2	11	2
30	z9sym	9	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	Sumaryczna liczba bloków:			1375		967		789		676		600		540		414		354		1419		1000		803		679		604		545		417		356	
	Sumaryczna liczba warstw logicznych:			296		210		164		138		118		106		69		56		109		84		74		65		61		54		48		44	

Podobnie wyznaczone zostały wartości przedstawione na wykresie 2.2. Wyznaczono je na podstawie zależności $\frac{x-\sum warstw_{(K\_LW)}}{\sum warstw_{(K\_LW)}}$ 100%, gdzie za wartości x podstawiono sumy liczby warstw układów uzyskiwanych poszczególnymi metodami syntezy: W_LB (Σ *warstw* $_{(W\_LB)}$), W_LW (Σ *warstw* $_{(W\_LW)}$), dla poszczególnych wartości k.

Wykres 2.2. Porównanie metod syntezy W_LB, W_LW z metodą K_LW pod względem liczby warstw logicznych (opis w tekście)

Z powyższych rozważań wynika, że metoda W_LB, jak i W_LW jest pewną alternatywą dla metody klasycznej, szczególnie w przypadku wykorzystywania układów z niewielkimi blokami logicznymi typu PAL. Mogą one stanowić cenne uzupełnienie metody klasycznej, doprowadzając do redukcji liczby bloków logicznych, prowadząc niestety do wydłużenia czasu propagacji sygnału od wejść do wyjść w uzyskiwanym układzie. Ponieważ przedstawione algorytmy są stosunkowo szybkie, można śmiało myśleć o połączeniu metody W_LB, W_LW z metodą klasyczną i wybraniu najlepszego rozwiązania.

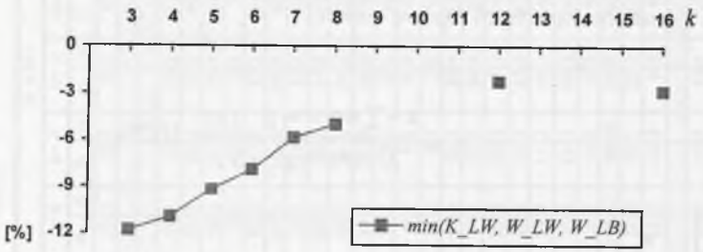
Jakie korzyści może dać tak skonstruowana metoda syntezy?

Tabela 2.4 przedstawia porównanie metody klasycznej ze strategią syntezy, w której spośród wyników trzech metod syntezy (K_LW, W_LB, W_LW) wybrane zostało rozwiązanie, wykorzystujące najmniejszą liczbę k-iloczynowych bloków PAL.

Wybierając najlepsze rozwiązanie spośród metod K_LW, W_LB, W_LW można liczyć na oszczędność na poziomie kilku % (wykres 2.3). Największe korzyści uzyskuje się dla niewielkich bloków logicznych typu PAL. Niestety oszczędne wykorzystywanie bloków logicznych prowadzi do uzyskiwania struktur zbudowanych z większej liczby warstw logicznych.

Wykres 2.3. Porównanie sumarycznej liczby bloków, uzyskiwanej po wyborze najlepszych rozwiązań min(K_LW,W_LW,W_LB) z sumaryczną liczbą bloków uzyskiwaną metodą klasyczną (opis w tekście)

$$\frac{\sum bloków_{\min(K\_LW,W\_LW,W\_LB)} - \sum bloków_{(K\_LW)}}{\sum bloków_{(K\_LW)}} 100\%$$



Stosunek sumarycznego czasu syntezy układów testowych metodą wyboru najlepszego rozwiązania spośród metod W_LW, W_LB, K_LW ($\sum t_{\min(W\_LW,W\_LB,K\_LW)}$) do sumarycznego czasu syntezy metodą klasyczną ($\sum t_{(K\_LW)}$) wynosi około 3,45.

Tabela 2.4

Porównanie klasycznej realizacji funkcji (metoda K_LW) ze strategią syntezy z wyborem najlepszego rozwiązania, spośród rozwiązań uzyskiwanych metodami W_LW, W_LB oraz K_LW

metoda K_LW - **K**lasyczna metoda realizacji funkcji w strukturach typu PAL z minimalizacją **L**iczby **W**arstw
metoda W_LW - metoda realizacji funkcji **W**ielowyjściowej wykorzystująca grafy wyjść (optymalizacja **L**iczby **W**arstw logicznych)
metoda W_LB - metoda realizacji funkcji **W**ielowyjściowej wykorzystująca grafy wyjść (optymalizacja **L**iczby **B**loków)
i - liczba wejść układu testowego; o - liczba wyjść układu testowego; B - liczba bloków; W - liczba warstw

	Układ testowy	i	o	K_LW														min(W_LW; W_LB; K_LW)																	
				k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16		k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16	
				B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W
1	5xp1	7	10	35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2	35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2
2	9sym	9	1	43	5	29	4	22	3	18	3	15	3	13	3	8	2	6	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	9symml	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	b12	15	9	24	3	16	2	14	2	13	2	10	2	10	2	9	1	9	1	24	3	16	2	14	2	13	2	10	2	10	2	9	1	9	1
5	bw	5	28	49	2	38	2	30	2	28	1	28	1	28	1	28	1	28	1	49	2	38	2	30	2	28	1	28	1	28	1	28	1	28	1
6	clip	9	5	73	4	49	3	38	3	30	3	26	2	22	2	14	2	12	2	66	19	45	13	37	4	29	4	26	2	21	3	14	2	11	2
7	con1	7	2	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1
8	duke2	22	29	99	3	72	3	61	2	49	2	44	2	41	2	35	2	31	2	99	3	72	3	61	2	49	2	44	2	41	2	35	2	31	2
9	ex1010	10	10	242	4	165	3	124	3	101	3	85	3	74	2	49	2	38	2	242	4	165	3	124	3	100	3	85	3	74	2	49	2	37	2
10	f51m	8	8	37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2	37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2
11	inc	7	9	23	3	17	2	16	2	13	2	11	2	11	2	9	1	9	1	23	3	17	2	15	3	13	2	11	2	11	2	9	1	9	1
12	ldd	9	19	34	2	25	2	25	2	21	2	19	1	19	1	19	1	19	1	34	2	25	2	25	2	21	2	19	1	19	1	19	1	19	1
13	misex1	8	7	14	2	12	2	8	2	7	1	7	1	7	1	7	1	7	1	14	2	10	3	8	2	7	1	7	1	7	1	7	1	7	1
14	misex2	25	18	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1
15	pcle	19	9	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1
16	rd53	5	3	15	3	11	2	8	2	6	2	6	2	6	2	4	2	3	1	15	3	11	2	8	2	6	2	6	2	6	2	4	2	3	1
17	rd73	7	3	70	4	47	3	36	3	29	3	24	3	20	2	14	2	11	2	63	7	44	5	32	5	26	4	23	3	19	3	13	3	10	3
18	rd84	8	4	142	5	95	4	72	4	58	3	49	3	42	3	28	2	21	2	129	47	87	6	66	5	54	4	45	4	39	4	27	3	20	3
19	root	8	5	35	4	24	3	18	3	16	2	13	2	11	2	9	2	7	2	33	14	22	10	18	3	16	2	13	2	11	2	9	2	7	2
20	sao2	10	4	36	3	24	3	19	2	15	2	14	2	11	2	7	2	5	2	29	4	20	3	16	3	13	3	11	3	10	3	7	3	6	2
21	sqn	7	3	21	3	15	3	11	2	10	2	8	2	7	2	5	2	4	2	20	4	14	3	11	2	10	2	8	2	7	2	5	2	4	2
22	sqr6	6	12	31	3	24	2	20	2	16	2	16	2	15	2	12	2	11	1	31	3	24	2	20	2	16	2	16	2	15	2	12	2	11	1
23	squar5	5	8	14	2	11	2	9	2	9	2	9	2	8	1	8	1	8	1	14	2	11	2	9	2	9	2	9	2	8	1	8	1	8	1
24	table3	14	14	264	4	181	4	135	3	110	3	94	3	80	3	53	2	40	2	162	14	114	12	93	10	82	8	79	8	69	7	53	2	40	2
25	vg2	25	8	53	4	37	3	28	3	22	3	20	2	19	2	13	2	11	2	53	4	37	3	28	3	22	3	20	2	19	2	13	2	11	2
26	x2	10	7	13	2	10	2	10	2	7	1	7	1	7	1	7	1	7	1	13	2	10	2	9	2	7	1	7	1	7	1	7	1	7	1
27	xor5	5	1	8	2	5	2	4	2	3	2	3	2	3	2	2	2	1	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
28	z4ml	7	4	29	4	19	3	15	3	13	2	11	2	9	2	7	2	5	2	29	4	19	3	15	3	13	2	11	2	9	2	7	2	5	2
29	z5xp1	7	10	37	3	27	3	23	2	19	2	17	2	15	2	12	2	11	2	37	3	27	3	23	2	19	2	17	2	15	2	12	2	11	2
30	z9sym	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	Sumaryczna liczba bloków:			1423		1003		793		657		579		521		401		349		1255		893		720		605		545		495		392		339	
	Sumaryczna liczba warstw logicznych:			79		67		58		52		49		46		42		40		157		94		68		59		54		52		43		40	

3. METODY REALIZACJI FUNKCJI LOGICZNYCH W STRUKTURACH TYPU PAL Z TRÓJSTANOWYMI BUFORAMI WYJŚCIOWYMI

Sposób efektywnego wykorzystania iloczynów stanowi jeden z kluczowych problemów syntezy logicznej, przeznaczonej dla układów CPLD typu PAL. Zaprezentowane w rozdziale drugim metody syntezy w wielu przypadkach ograniczają liczbę użytych bloków logicznych typu PAL poprzez wyszukiwanie odpowiednich grup wielowyjściowych implikantów. Uzyskana tymi metodami redukcja liczby bloków często okupiona jest jednak ekspansją liczby warstw logicznych.

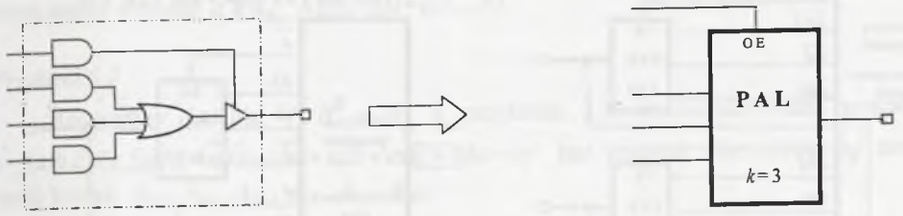
Problem zwiększania liczby warstw logicznych towarzyszy również metodzie klasycznej. Występuje on w przypadku konieczności realizacji funkcji będącej sumą większej liczby implikantów niż liczba iloczynów zawartych w bloku logicznym typu PAL. Problem ten został zauważony przez producentów układów programowalnych, którzy do struktur CPLD typu PAL wprowadzili sprzętowe mechanizmy umożliwiające nierównomierny rozdział iloczynów pomiędzy komórki wyjściowe. Należą do nich między innymi programowalne rozdzielacze (ang. product term allocator - struktury MACH opracowane przez firmę AMD a obecnie rozwijane przez firmę Lattice, struktury Flash firmy Cypress, struktury rodziny XC9500 firmy Xilinx) lub różnego typu ekspandery (ang. shareable expanders, parallel expanders – struktury rodziny MAX5000, MAX7000, MAX9000 itp. firmy Altera). Sprzętowe zasoby układów programowalnych, umożliwiające nierównomierny rozdział iloczynów, prowadzący do tworzenia bloków logicznych typu PAL zawierających różną liczbę iloczynów, mimo swoich niewątpliwych zalet nie zapewniają realizacji każdej funkcji w jednym bloku logicznym typu PAL.

Okazuje się, że problem ekspansji liczby warstw logicznych można rozwiązać wykorzystując występujące w strukturach programowalnych trójstanowe bufory wyjściowe.

3.1. Idea wykorzystania trójstanowych buforów wyjściowych

Elementarne bloki logiczne typu PAL, składające się z iloczynów dołączonych do sumy, często zawierają dodatkowe elementy. W większości struktur programowalnych występują trójstanowe bufory wyjściowe [Atme00, Adva95, Adva96, Alte99, Latt94, Cypr96, Shar98, Xili99]. Wprowadzenie do struktury programowalnej tych elementów przede wszystkim wprowadza możliwość elastycznego konfigurowania wyprowadzeń układu jako wyprowadzenia wejściowe, wyjściowe lub dwukierunkowe.

Struktura bloku logicznego typu PAL wyposażonego w sterowany trójstanowy bufor wyjściowy oraz jego schemat blokowy przedstawione są na rys. 3.1.



Rys. 3.1. Struktura bloku logicznego typu PAL, zawierającego trójstanowy bufor wyjściowy

Fig. 3.1. Structure of PAL-based logic block consisting of three-state output buffer

Ze względu na wspólny proces technologiczny wszystkie elementy struktur programowalnych mają zbliżone parametry dynamiczne. Stwarza to możliwość nietypowego wykorzystania trójstanowych buforów wyjściowych, z uwzględnieniem ich obecności w procesie syntezy logicznej.

Przykład 3.1

Rozważmy funkcję $f: B^8 \to B$, którą po minimalizacji można przedstawić w postaci sumy implikantów $y=f(a,b,c,d,e,f,g,h) = \bar{a}d\bar{h} + \bar{a}\bar{d} + \bar{a}de + a\bar{b}g + abc + af$. Klasyczna metoda realizacji funkcji, przedstawiona na rys. 3.2a, zajmuje 3 bloki logiczne typu PAL, zawierające 3 iloczyny. Możliwa jest inna realizacja przykładowej funkcji, wykorzystująca trójstanowe bufory wyjściowe. Zmienna a dzieli zbiór implikantów funkcji $f: B^8 \to B$ na dwa podzbiory: jeden, zawierający implikanty, w których występuje literał $\bar{a}$ oraz drugi, zawierający implikanty z literałem a. Zmienna a, zwana zmienną podziału implikantów, może być wykorzystana do sterowania odpowiedniego trójstanowego bufora wyjściowego.

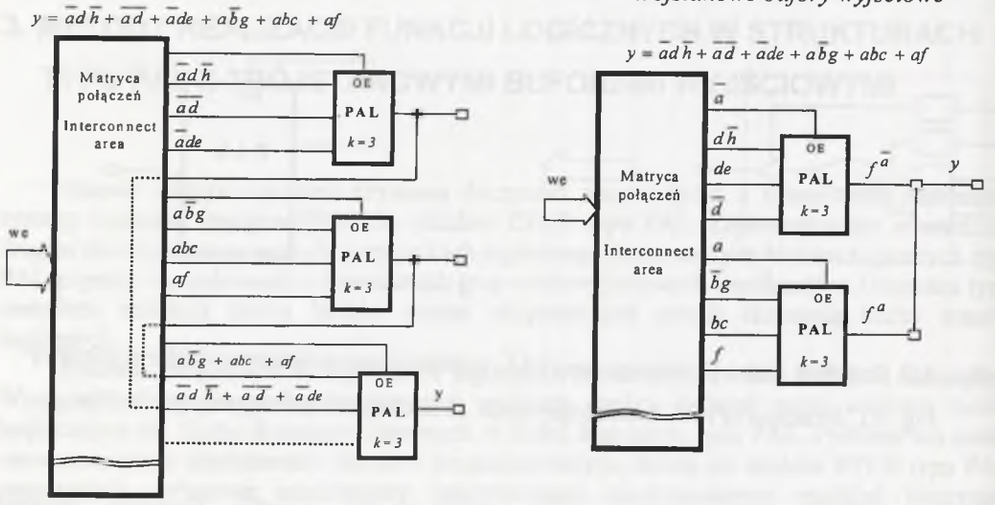
Realizacja funkcji wykorzystująca trójstanowe bufory wyjściowe, sterowane przez zmienną podziału implikantów a, przedstawiona jest na rys. 3.2b.

Przykład 3.1 pokazuje, że wykorzystanie trójstanowych buforów wyjściowych prowadzi przede wszystkim do zmniejszenia liczby warstw logicznych, lecz może też zmniejszać liczbę użytych bloków logicznych typu PAL.

Głównym problemem syntezy logicznej, wykorzystującej trójstanowe bufory wyjściowe, jest sposób doboru zmiennych, w ogólnym przypadku kostek podziału implikantów.

a) *Realizacja klasyczna*

b) *Realizacja wykorzystująca trójstanowe bufory wyjściowe*



Rys. 3.2. Realizacje funkcji $f: B^8 \to B$

Fig. 3.2. Implementations of the function $f: B^8 \to B$

3.2. Reprezentacja sposobu implementacji funkcji w postaci grafu podziału

Niech $f: B^n \to B$, gdzie $B=\{0,1\}$. Funkcja f może być również przedstawiona w postaci $y=f(i_n,...,i_2,i_1)$ przez przyporządkowanie każdemu wektorowi wejściowemu $I=(i_n,...,i_2,i_1)$ wartości y równej 0 lub 1. Nadając zmiennym $i_n,...,i_2,i_1$ wartości zero lub jeden, otrzymujemy stany wejściowe $I_j \subset B^n$; $(j=0,1,...,2^n-1)$.

Zminimalizowana postać funkcji może być opisana przez zbiór jednowyjściowych implikantów będących zbiorem kostek o wymiarach n oraz 1 nazywanych odpowiednio częścią wejściową oraz częścią wyjściową [Mich94]. Część wejściowa składa się z elementów 0,1,- i reprezentuje iloczyn literałów, dla którego funkcja przyjmuje wartość 1, stanowiącą część wyjściową jednowyjściowego implikantu.

Niech $\mathbf{A}_f = \{A_{y(l-1)},...,A_{y1},A_{y0}\}$ będzie zbiorem części wejściowych jednowyjściowych implikantów, zwanych dalej zbiorem kostek. Każda kostka $A_{y_l} = (a_n,...,a_2,a_1)$, należąca do zbioru $\mathbf{A}_f$, składa się z n-elementów, odpowiadających zmiennym wejściowym $i_n,...,i_2,i_1$, przy czym każdy element $a_i \in \mathbf{A}_{0,1,-} = \{0,1,-\}$, $(i=1,2,...,n)$.

Przykład 3.2

Rozważmy funkcję $f: B^8 \to B$ z przykładu 3.1. Zminimalizowana postać funkcji $y=f(a,b,c,d,e,f,g,h) = \bar{a}\bar{d}\bar{h} + \bar{a}\bar{d} + \bar{a}de + a\bar{b}g + abc + af$ jest opisana przez zbiór $\mathbf{A}_f$ zawierający sześć kostek A_{y5}, A_{y4}, A_{y3}, A_{y2}, A_{y1}, A_{y0}.

$$\mathbf{A}_f = \begin{Bmatrix} (0--1---0) \\ (0--0----) \\ (0--11---) \\ (10----1-) \\ (111-----) \\ (1----1--) \end{Bmatrix}_{abcdefgh}$$

Każdą funkcję $f(i_n,...,i_2,i_1)$ można rozwinąć względem każdej zmiennej (rozwinięcie Shannona [AlbiI86, MichL93]), na przykład:

$$f(i_n,...,i_2,i_1) = i_1 f(i_n,...,i_2,1) + \bar{i}_1 f(i_n,...,i_2,0) = i_1 f_{i_1} + \bar{i}_1 f_{\bar{i}_1}$$

Kontynuując rozwijanie otrzymuje się:

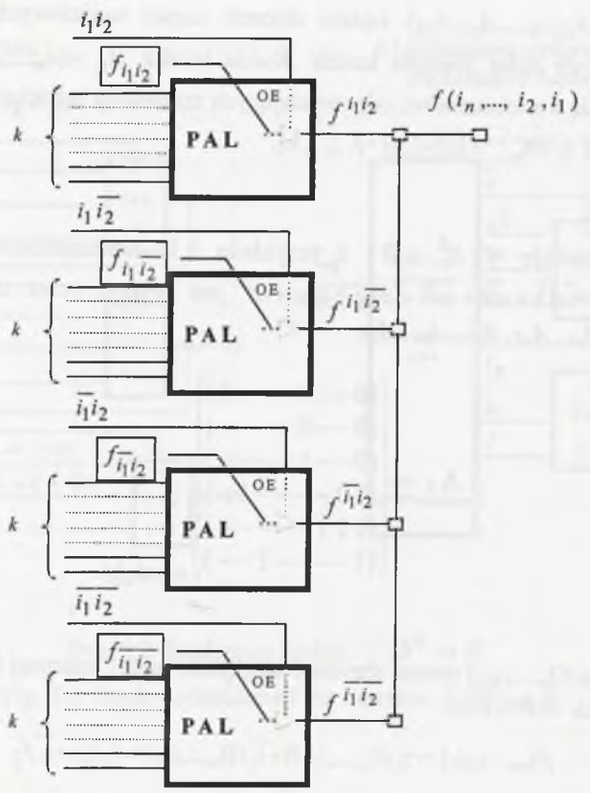
$$f(i_n,...,i_2,i_1) = i_1 i_2 f_{i_1 i_2} + i_1 \bar{i}_2 f_{i_1 \bar{i}_2} + \bar{i}_1 i_2 f_{\bar{i}_1 i_2} + \bar{i}_1 \bar{i}_2 f_{\bar{i}_1 \bar{i}_2} \qquad (3.1)$$

Niech funkcja f^x opisuje zachowanie wyjścia bloku logicznego typu PAL z trójstanowym buforem wyjściowym, sterowanym przez zmienną x.

Ponieważ

$$f^x = \begin{cases} f_x & dla\ x=1 \\ stan\ wysokiej\ impedancji & dla\ x=0 \end{cases},$$

stąd funkcję $f(i_n,...,i_2,i_1)$ przedstawioną w postaci (3.1) można zrealizować, łącząc wyjścia bloków logicznych z trójstanowymi buforami wyjściowymi (rys. 3.3), których zachowanie wyjść opisują funkcje: $f^{i_1 i_2}$, $f^{i_1 \bar{i}_2}$, $f^{\bar{i}_1 i_2}$, $f^{\bar{i}_1 \bar{i}_2}$.



Rys. 3.3. Ekspansja liczby iloczynów wykorzystująca bufory trójstanowe

Fig. 3.3. Product term expansion with three-state buffers

Przedstawioną zasadę rozwinięć można wykorzystać do podziału zbioru $\mathbf{A}_f$ i realizacji uzyskanych w ten sposób grup implikantów na blokach logicznych o zadanej liczbie iloczynów.

Niech $X=(x_p,\ldots,x_2,x_1)$ będzie p-elementową ($p<n$) kostką, w której poszczególne elementy odpowiadają wybranym elementom kostek $A_y=(a_n,\ldots,a_2,a_1)$, przy czym każdy element $x_i \in \mathbf{X}_{0,1}=\{0,1\}$, $(i=1,2,\ldots,p)$.

Po odpowiednim uporządkowaniu elementów kostek dowolny i-ty element kostki $A_y=(a_n,\ldots,a_2,a_1)$ i $X=(x_p,\ldots,x_2,x_1)$, gdzie $(i=1,2,\ldots,p)$, odpowiada tej samej zmiennej wejściowej. Uporządkowane pary $<a_i,x_i>$ należą do zbioru par będącego iloczynem kartezjańskim zbiorów $\mathbf{A}_{0,1,-}$ i $\mathbf{X}_{0,1}$; $(<a_i,x_i> \in \mathbf{A}_{0,1,-} \times \mathbf{X}_{0,1})$.

Definicja 3.1

Dwie kostki $A_y=(a_n,\ldots,a_2,a_1)$ i $X=(x_p,\ldots,x_2,x_1)$ takie, że $p \le n$ i $a_i \in \mathbf{A}_{0,1,-}=\{0,1,-\}$, $x_i \in \mathbf{X}_{0,1}=\{0,1\}$ nazywamy ***kostkami niesprzecznymi*** wtedy i tylko wtedy, gdy zbiór uporządkowanych p-par $<a_i,x_i>$ nie zawiera elementów <0,1> i <1,0>.

Np.: kostki $A_y=(10-1-)_{abcde}$ i $X=(011)_{bcd}$ są kostkami niesprzecznymi.

Definicja 3.2

Dwie kostki $A_y=(a_n,\ldots,a_2,a_1)$ i $X=(x_p,\ldots,x_2,x_1)$ takie, że $p \le n$ i $a_i \in \mathbf{A}_{0,1,-}=\{0,1,-\}$, $x_i \in \mathbf{X}_{0,1}=\{0,1\}$ nazywamy ***kostkami nierównoważnymi*** wtedy i tylko wtedy, gdy w zbiorze uporządkowanych p-par $<a_i,x_i>$ istnieje para nie należąca do zbioru $\{<0,0>,<1,1>\}$.

Np.: kostki $A_y=(10-1-)_{abcde}$ i $X=(100)_{bcd}$ są kostkami nierównoważnymi.

Dowolna kostka $X=(x_p,\ldots,x_2,x_1)$, nazywana ***kostką podziału***, implikuje podział zbioru kostek $\mathbf{A}_f$, dla których funkcja przyjmuje wartość 1 na dwa podzbiory $\mathbf{A}_{fX}$ oraz ${}^*\mathbf{A}_{fX}$. Pierwszy z nich zawiera kostki niesprzeczne, a drugi kostki nierównoważne z wybraną kostką podziału.

W sytuacji, gdy kostka podziału zredukowana jest do pojedynczej zmiennej x, zachodzą następujące zależności: $\mathbf{A}_{f_x}={}^*\mathbf{A}_{f_{\bar{x}}}$, $\mathbf{A}_{f_{\bar{x}}}={}^*\mathbf{A}_{f_x}$.

Przykład 3.3

Rozważmy funkcję $f: B^8 \to B$ z przykładu 3.1. Zbiór kostek, dla których funkcja przyjmuje wartość 1, zawiera sześć elementów

$$\mathbf{A}_f=\begin{Bmatrix}(0--1---0)\\(0--0----)\\(0--11---)\\(10----1-)\\(111-----)\\(1----1--)\end{Bmatrix}_{abcdefgh}$$

Niech kostka podziału $X=(10)_{ab}$. Zbiory kostek niesprzecznych oraz nierównoważnych z wybraną kostką podziału przedstawione są poniżej:

Kostki niesprzeczne z kostką $X=(10)_{ab}$

$$\mathbf{A}_{fX} = \begin{Bmatrix} (10----1-) \\ (1----1--) \end{Bmatrix}_{abcdefgh}$$

Kostki nierównoważne z kostką $X=(10)_{ab}$

$${}^{*}\mathbf{A}_{fX} = \begin{Bmatrix} (0--1---0) \\ (0--0----) \\ (0--11---) \\ (111-----) \\ (1----1--) \end{Bmatrix}_{abcdefgh}$$

Przedstawiony podział zbioru $\mathbf{A}_f$ odpowiada następującemu rozwinięciu funkcji

$$f(a,b,c,d,e,f,g,h) = a\bar{b}f_{a\bar{b}} + \bar{a}\bar{b}f_{\bar{a}\bar{b}},$$

gdzie

$$\bar{a}\bar{b}f_{\bar{a}\bar{b}} = abf_{ab} + \bar{a}bf_{\bar{a}b} + \bar{a}\bar{b}f_{\bar{a}\bar{b}}$$

Niech Δ_{f_X} będzie wyróżnikiem równym mocy zbioru $\mathbf{A}_{fX}$ ($\Delta_{f_X} = \overline{\overline{\mathbf{A}_{fX}}}$), natomiast ${}^{*}\Delta_{f_X}$ wyróżnikiem sprzężonym równym $\overline{{}^{*}\mathbf{A}_{fX}}$. W przypadku, gdy kostka podziału zredukowana jest do pojedynczej zmiennej x, zachodzą następujące równości: $\Delta_{f_x} = {}^{*}\Delta_{f_{\bar{x}}}$ oraz $\Delta_{f_{\bar{x}}} = {}^{*}\Delta_{f_x}$.

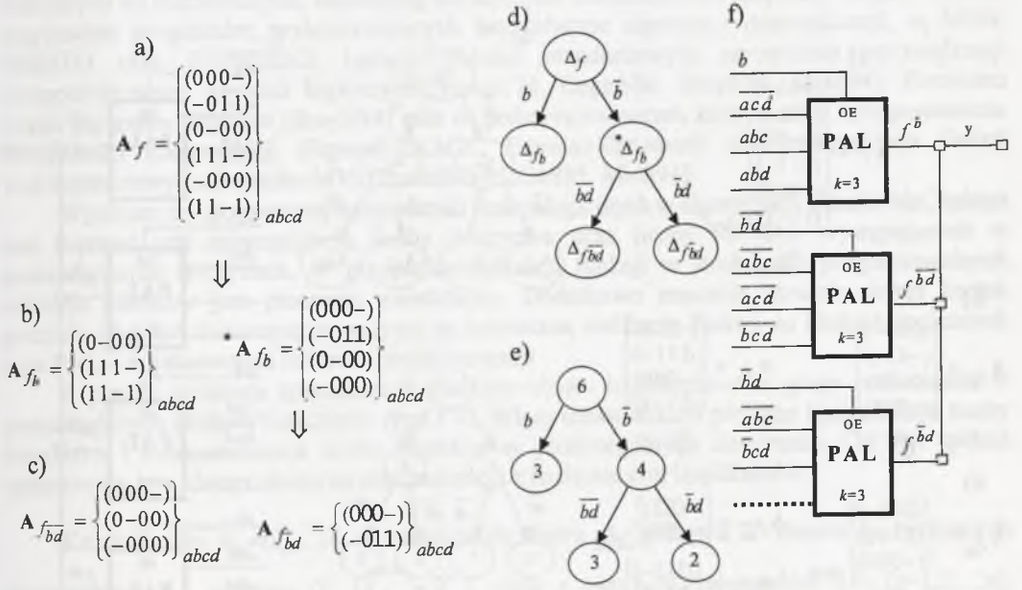
Realizacja funkcji logicznej z wykorzystaniem bloków logicznych typu PAL z trójstanowymi buforami wyjściowymi jest ściśle związana z odpowiednim podziałem zbioru kostek (implikantów) w taki sposób, aby poszczególne części były realizowalne na k-iloczynowych blokach PAL. Proces podziału zbioru $\mathbf{A}_f$ bezpośrednio łączy się z doborem odpowiednich kostek podziału. Metody doboru kostek podziału prowadzące do różnorodnych rozwiązań zostaną przedstawione w dalszej części pracy.

Sposób podziału implikantów ściśle związany z realizacją funkcji z wykorzystaniem bloków logicznych typu PAL może być opisany za pomocą grafu podziału.

Przykład 3.4

Rozważmy funkcję $f: B^4 \to B$, która jest opisana przez zbiór kostek $\mathbf{A}_f$, przedstawiony jest na rys. 3.4a. Załóżmy, że poszukujemy realizacji funkcji na blokach logicznych typu PAL, zawierających trzy iloczyny oraz trójstanowe bufory wyjściowe. Zmienna podziału b implikuje podział zbioru $\mathbf{A}_f$ na dwa podzbiory $\mathbf{A}_{fb}$ oraz ${}^{*}\mathbf{A}_{fb} = \mathbf{A}_{f_{\bar{b}}}$ przedstawione na rys. 3.4b. Ponieważ

$\overline{\overline{{}^{*}\mathbf{A}_{fb}}} = {}^{*}\Delta_{f_b} > k$, to konieczny jest dalszy podział zbioru ${}^{*}\mathbf{A}_{fb}$. Wynik podziału implikowanego przez zmienną d przedstawiony jest na rys. 3.4c. Proces podziału zbioru $\mathbf{A}_f$ o mocy Δ_f można przedstawić za pomocą grafu podziału (rys. 3.4d, 3.4e). Ostateczna realizacja przykładowej funkcji pokazana jest na rys. 3.4f.



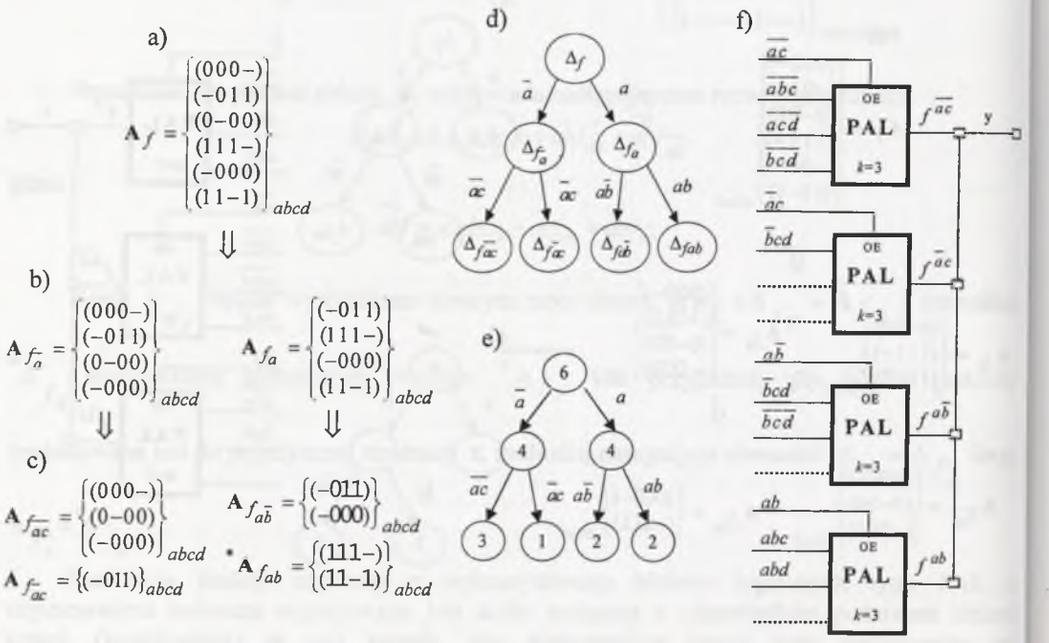
Rys. 3.4. Realizacja funkcji wykorzystująca trójstanowe bufory

Fig. 3.4. Implementation of the function using three-state buffers

Wybór zmiennej (kostki) podziału znacząco wpływa na liczbę wykorzystywanych bloków logicznych typu PAL z trójstanowymi buforami wyjściowymi.

Przykład 3.5

Rozważmy ponownie funkcję z przykładu 3.4 szukając innej realizacji z wykorzystaniem bloków logicznych typu PAL, zawierających 3 iloczyny oraz trójstanowy bufor wyjściowy. Jako zmienną podziału wybierzmy tym razem zmienną *a*. Ponieważ zmienna ta implikuje podział zbioru $\mathbf{A}_f$ (rys. 3.5a) na dwa podzbiory o mocy większej od *k* (rys. 3.5b), stąd konieczny jest dalszy podział obydwu części (rys. 3.5c). Poszczególne etapy podziału zbioru $\mathbf{A}_f$, graf podziału oraz strukturę układu, tym razem wykorzystującą 4 bloki, przedstawiono na rys. 3.5 a,b,c,d,e,f.



Rys. 3.5. Realizacja funkcji wykorzystująca trójstanowe bufory

Fig. 3.5. Implementation of the function using three-state buffers

Pierwsze koncepcje doboru zmiennych podziału zostały przedstawione w pracach [HrynK95, Kani95]. Nie zawsze prowadziły one do zadowalających rezultatów. W późniejszym okresie okazało się, że nie tylko dobór zmiennych (kostek) podziału wpływa na efektywność wykorzystywania bloków logicznych typu PAL. Niezwykle istotne znaczenie ma również sposób wykonania minimalizacji [Kani99a, Kani 99b].

3.3. Minimalizacja z rozłączaniem implikantów

Zwykle celem minimalizacji funkcji logicznych jest zredukowanie wyrażeń będących sumą iloczynów lub iloczynem sum. W przypadku struktur programowalnych z matrycami AND-OR istotna jest optymalizacja wyrażeń będących sumami iloczynów. Opracowane przez Quine'a i McCluskeya dokładne algorytmy minimalizacji zapewniają minimalizację wyrażeń dwupoziomowych. W późniejszym okresie zaproponowano kilka heurystycznych algorytmów bazujących na różnorodnych, najczęściej iteracyjnych mechanizmach ulepszeń. Między innymi, przykładem programów, wykorzystujących heurystyczne algorytmy minimalizacji, są MINI, PRESTO oraz ESPRESSO, będący obecnie standardowym narzędziem optymalizacji dwupoziomowych wyrażeń logicznych [HongC74, DageA86, BrayH84, Mich94]. Pierwotna wersja programu Espresso [BrayH84] stała się podstawą rozszerzeń, których celem było poprawienie efektywności minimalizacji (Espresso-EXACT, Espresso-Signatures) oraz minimalizacja funkcji wielowartościowych (Espresso-MV) [RudeS87, McGeS93, Mich94].

Wynikiem dwupoziomowej minimalizacji funkcji logicznych w algorytmach Quine'a-McCluskeya oraz Espresso jest minimalizacja liczby iloczynów oraz liczby literałów występujących w poszczególnych iloczynach. W przypadku realizacji funkcji w strukturach programowalnych redukcja literałów jest procesem nieistotnym. Dodatkowo znacznie utrudnia dobór kostek podziału i bardzo niekorzystnie wpływa na ostateczną realizację funkcji na blokach logicznych typu PAL z trójstanowymi buforami wyjściowymi.

Z punktu widzenia optymalizacji podziału zbioru implikantów na grupy realizowane w poszczególnych blokach logicznych typu PAL celem minimalizacji powinna być redukcja liczby iloczynów i maksymalizacja liczby literałów w poszczególnych iloczynach. Cel ten spełnia opracowana, tzw. dwupoziomowa minimalizacja z rozłączaniem implikantów.

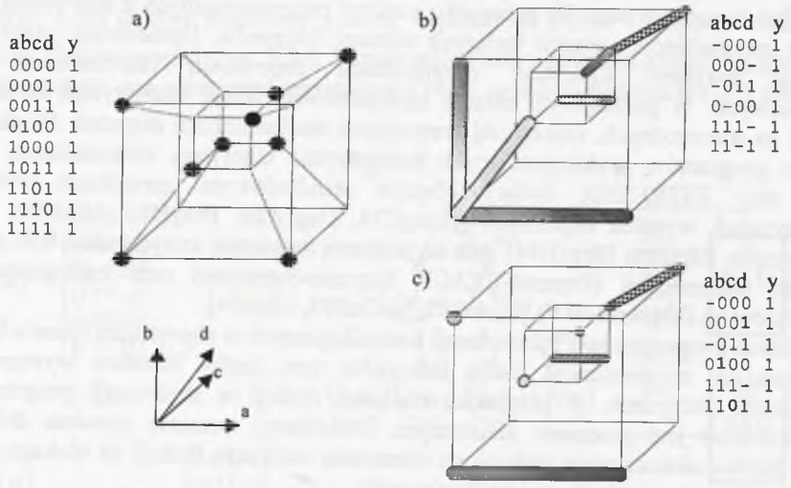
Każda kostka $A_y=(a_n,...,a_2,a_1)$ należąca do zbioru $\mathbf{A}_f$ pokrywa 2^r stanów I_j, tworzących zbiór stanów $\mathbf{I}_{A_y}$ o mocy 2^r ($\overline{\overline{\mathbf{I}_{A_y}}}=2^r$), gdzie r jest liczbą elementów a_i= - (i=1,2,...,n), zwaną wymiarem kostki.

Definicja 3.3

Dwie kostki $A_{ys}=(a_{ns},...,a_{2s},a_{1s})$ i $A_{yt}=(a_{nt},...,a_{2t},a_{1t})$, których elementy $a_{is},a_{it}\in\mathbf{A}_{0,1,-}=\{0,1,-\}$ takie, że A_{ys} pokrywa 2^{r_s} stanów tworzących zbiór stanów $\mathbf{I}_{A_{ys}}$, a A_{yt} pokrywa 2^{r_t} stanów tworzących zbiór stanów $\mathbf{I}_{A_{yt}}$, nazywamy kostkami półpokrywającymi wtedy i tylko wtedy, gdy $\overline{\overline{\mathbf{I}_{A_{ys}}\cap\mathbf{I}_{A_{yt}}}}=2^{r_s-1}$ lub 2^{r_t-1}.

Przykładowo kostki $A_{ys}=(-\ 0\ 0\ -\ 1)_{abcde}$ oraz $A_{yt}=(1\ 0\ 0\ 1\ -)_{abcde}$ są kostkami półpokrywającymi.

Dwupoziomowa minimalizacja z rozłączaniem implikantów składa się dwóch elementów: klasycznej dwupoziomowej minimalizacji (ESPRESSO) oraz procedury rozłączania implikantów. Idea rozłączania dla funkcji analizowanej w przykładach 3.4 oraz 3.5 przedstawiona jest na rys. 3.6.



Rys. 3.6. Idea minimalizacji dwupoziomowej z rozłączaniem implikantów;
a) przedstawienie funkcji przed minimalizacją; b) wynik minimalizacji dwupoziomowej;
c) wynik minimalizacji dwupoziomowej z rozłączaniem implikantów
Fig. 3.6. Two-level splitting minimization concept, a) representation of the function before minimization, b) the result of two-level minimization, c) the result of two-level splitting minimization

Dane wejściowe procedury rozłączania implikantów, wykonywanej po klasycznej minimalizacji dwupoziomowej, stanowi zbiór kostek $\mathbf{A}_f = \{A_{y(l-1)},...,A_{y1},A_{y0}\}$, dla których funkcja y przyjmuje wartość 1.

Niech kostka $A_{ys} \in \mathbf{A}_f$ równa $A_{ys} = (a_{ns},...,a_{2s},a_{1s})$ pokrywa 2^{r_s} stanów tworzących zbiór stanów $\mathbf{I}_{A_{ys}}$, natomiast kostka $A_{yt} \in \mathbf{A}_f$ równa $A_{yt} = (a_{nt},...,a_{2t},a_{1t})$ pokrywa 2^{r_t} stanów tworzących zbiór stanów $\mathbf{I}_{A_{yt}}$. Jeżeli dwie kostki $A_{ys} = (a_{ns},...,a_{2s},a_{1s})$ i $A_{yt} = (a_{nt},...,a_{2t},a_{1t})$ są kostkami półpokrywającymi, to okazuje się, że można zmodyfikować jedną kostkę bez zmiany zbioru $\mathbf{I}_{A_{ys} \cup A_{yt}} = \mathbf{I}_{A_{ys}} \cup \mathbf{I}_{A_{yt}}$. Wyszukiwanie par kostek półpokrywających polega na analizie uporządkowanych par $< a_{is}, a_{it} >$, gdzie i=1,2,...,n. Jeżeli założymy, że kostka A_{ys} zawiera nie mniejszą liczbę elementów $a_{is} = -$ niż kostka A_{yt}, to kostki A_{ys} i A_{yt} są kostkami półpokrywającymi wtedy i tylko wtedy, gdy wśród elementów zbioru uporządkowanych par

występują pary należące do zbioru {<0,0>,<1,1>,<-,->,<-,0>,<-,1>} i tylko jedna para $< a^*_{is}, a^*_{it} >$, należąca do zbioru {<1,->,<0,->}. Modyfikacja, nie zmieniająca zbioru stanów $\mathbf{I}_{A_{ys} \cup A_{yt}} = \mathbf{I}_{A_{ys}} \cup \mathbf{I}_{A_{yt}}$ pokrytych przez kostki A_{ys} i A_{yt}, polega na zamianie elementu a^*_{it} zgodnie z zasadą: jeżeli $a^*_{is} = 1$, to $a^*_{it} = 0$, natomiast jeżeli $a^*_{is} = 0$, to $a^*_{it} = 1$. Jeżeli przykładowo $\mathbf{A}_f = \left\{\begin{matrix}(0---)\\(--00)\end{matrix}\right\}_{abcd}$, to po modyfikacji $\mathbf{A}_f = \left\{\begin{matrix}(0---)\\(1-00)\end{matrix}\right\}_{abcd}$.

Przedstawiona zasada jest podstawą procedury rozłączania implikantów, która poszukuje wszystkich par kostek półpokrywających i przekształca jedną kostkę należącą do tej pary. W efekcie tego procesu modyfikowane są wszystkie pary kostek półpokrywających, co prowadzi do uzyskania zbioru kostek $\mathbf{A}_f$, w którym zminimalizowano liczbę elementów a_i = -.

Przykład 3.6

Algorytm rozłączania kostek zostanie przedstawiony dla funkcji y_9 należącej do układu testowego 5xp1 [Bench]. Po dwupoziomowej minimalizacji uzyskujemy zbiór:

$$\mathbf{A}_{f9} = \left\{\begin{matrix}(----10-)\\(00--10)\\(1---011)\\(-1--011)\\(0--01-0)\\(0-0-1-0)\\(--11011)\end{matrix}\right\}_{abcdefg}$$

Poszczególne kroki rozłączania kostek przedstawione są poniżej:

$$\begin{bmatrix}----10-\\00--1-0\\1---011\\-1--011\\0--01-0\\0-0-1-0\\--11011\end{bmatrix} \mapsto \begin{bmatrix}----10-\\00--110\\1---011\\-1--011\\0--0110\\0-0-110\\--11011\end{bmatrix} \mapsto \begin{bmatrix}----10-\\00--110\\1---011\\-1--011\\01--110\\01-0110\\--11011\end{bmatrix} \mapsto \begin{bmatrix}----10-\\00--110\\1---011\\01--011\\01-0110\\01-0110\\--11011\end{bmatrix} \mapsto \begin{bmatrix}----10-\\00--110\\1---011\\01--011\\01-0110\\01-0110\\001101-\end{bmatrix} \mapsto \begin{bmatrix}----10-\\00--110\\1---011\\01--011\\01-0110\\01-0110\\001101-\end{bmatrix}$$

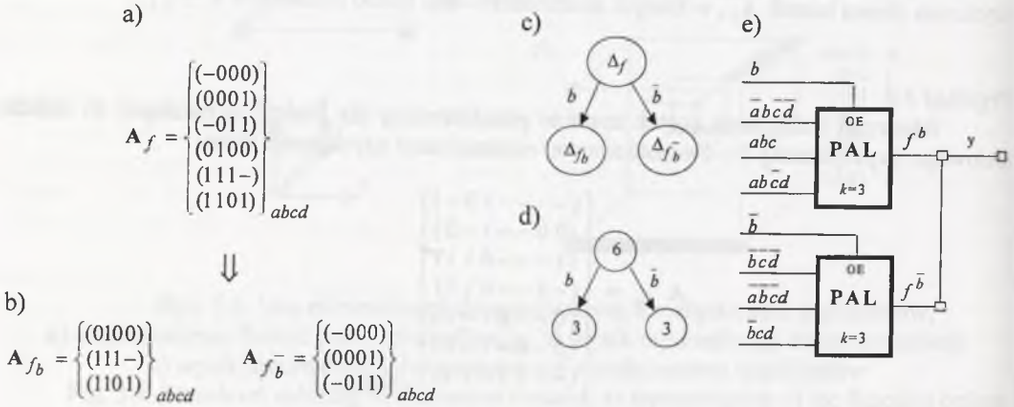
Ostatecznie, po procedurze rozłączania kostek uzyskujemy zbiór kostek:

$$\mathbf{A}_{f9} = \left\{\begin{matrix}(----10-)\\(00--110)\\(1---011)\\(01--011)\\(01-0110)\\(0101110)\\(0011011)\end{matrix}\right\}_{abcdefg}$$

Rozłączanie implikantów znacząco wpływa na efektywność podziału zbioru kostek $\mathbf{A}_f$, dla których funkcja przyjmuje wartość 1.

Przykład 3.7

Rozważmy ponownie funkcję analizowaną w przykładach 3.4 i 3.5. W wyniku dwupoziomowej minimalizacji z rozłączaniem implikantów (rys.3.6) pojawiła się możliwość doboru zmiennej podziału *b*, zapewniającej realizację funkcji tylko na dwóch blokach logicznych typu PAL, zawierających 3 iloczyny oraz trójstanowy bufor wyjściowy. Poszczególne etapy syntezy, graf podziału oraz ostateczna realizacja przedstawione są na rys. 3.7a,b,c,d,e.



Rys. 3.7. Realizacja funkcji wykorzystująca trójstanowe bufory

Fig. 3.7. Implementation of the function using three-state buffers

3.4. Koncepcja implementacji funkcji w strukturach typu PAL z trójstanowymi buforami wyjściowymi

Synteza logiczna prowadząca do realizacji funkcji $f: B^n \to B^m$ w strukturach typu PAL z trójstanowymi buforami wyjściowymi składa się z kilku elementów. Proces syntezy rozpoczyna klasyczna dwupoziomowa minimalizacja, wykorzystująca algorytm Espresso [BrayH84]. Minimalizacja wykonywana jest dla każdej funkcji $f_i: B^n \to B$ (i=1,2,...,m) oddzielnie (espresso-Dso). Następnie kolejno rozpatrywane są poszczególne, zminimalizowane funkcje $f_i: B^n \to B$, (i=1,2,...,m). Dla każdej z nich wykonana jest procedura rozłączania implikantów.

Celem następnego etapu jest podział zbiorów jednowyjściowych implikantów opisujących poszczególne funkcje, na podzbiory realizowalne w blokach logicznych typu PAL o zadanej liczbie iloczynów. Procedura podziału opiera się na odpowiednim doborze zmiennych (kostek) podziału. Sposób wyboru zmiennych (kostek) podziału ściśle związany jest z ostateczną implementacją funkcji w strukturach typu PAL z trójstanowymi buforami wyjściowymi.

3.4.1. Metoda wyszukiwania zmiennej podziału, wykorzystywana w *p*-warstwowej implementacji funkcji w strukturach typu PAL z trójstanowymi buforami wyjściowymi

Klasyczna realizacja funkcji $f_i: B^n \to B$ będącej sumą Δ_{f_i} - implikantów wymaga użycia $\delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1$ k-iloczynowych bloków PAL i prowadzi do struktury co najmniej $\xi_{f_i} = \lceil \lg_k \Delta_{f_i} \rceil = \left\lceil \frac{\lg(\Delta_{f_i})}{\lg k} \right\rceil$ - warstwowej (realizacja klasyczna z minimalizacją liczby warstw). Wykorzystanie trójstanowych buforów wyjściowych stwarza możliwość ograniczenia liczby warstw. Zmienna podziału x dzieli zbiór kostek $\mathbf{A}_{fi}$ na dwa podzbiory $\mathbf{A}_{f_{i_x}}$ i $\mathbf{A}_{fi_{\bar{x}}}$ o mocy $\Delta_{f_{i_x}}$ oraz $\Delta_{f_{i_{\bar{x}}}}$. Jeżeli wartości wyróżników $\Delta_{f_{i_x}}$ oraz $\Delta_{f_{i_{\bar{x}}}}$ są większe od liczby iloczynów zawartych w blokach logicznych typu PAL, zachodzi konieczność wprowadzania sprzężeń zwrotnych (klasyczna realizacja kostek należących do zbiorów $\mathbf{A}_{f_{i_x}}$ oraz $\mathbf{A}_{fi_{\bar{x}}}$). Prowadzi to do wykorzystania $\delta_{f_i} = \delta_{f_{i_x}} + \delta_{f_{i_{\bar{x}}}} = \left(\left\lceil \frac{\Delta_{fi_x} - k}{k-1} \right\rceil + 1 \right) + \left(\left\lceil \frac{\Delta_{fi_{\bar{x}}} - k}{k-1} \right\rceil + 1 \right)$ k-iloczynowych bloków PAL i uzyskania struktury $\xi_{f_i} = \max(\xi_{f_{i_x}}, \xi_{f_{i_{\bar{x}}}}) = \max\left(\left\lceil \frac{\lg(\Delta_{f_{i_x}})}{\lg k} \right\rceil, \left\lceil \frac{\lg(\Delta_{f_{i_{\bar{x}}}})}{\lg k} \right\rceil \right)$ - warstwowej.

Dla każdej funkcji $f_i: B^n \to B$ można wybrać n-zmiennych podziału. Najważniejszą zaletą metod syntezy wykorzystujących trójstanowe bufory wyjściowe jest redukcja liczby warstw w stosunku do metody klasycznej. Fakt ten sprawia, że głównym kryterium wyboru zmiennej podziału powinna być przede wszystkim minimalizacja liczby warstw logicznych. W sytuacji, gdy dla kilku zmiennych uzyskuje się identyczną liczbę warstw, wybierana powinna być zmienna, która zapewnia minimalizację liczby użytych bloków logicznych typu PAL.

Przykład 3.8

Rozważmy funkcję $f:B^7 \to B$, która po dwupoziomowej minimalizacji z rozłączaniem implikantów przyjmuje postać przedstawioną w pliku *f.pla* (rys.3.8a). Załóżmy, że poszukujemy realizacji funkcji na blokach logicznych typu PAL, zawierających 3 iloczyny oraz trójstanowy bufor wyjściowy. W tabeli przedstawionej na rys. 3.8b zawarte są wartości wyróżników oraz parametry charakteryzujące podział dla wszystkich możliwych zmiennych podziałów.

a) *f.pla*

```
.i 7
.o 1
.ilb abcdegh
.ob y
.p 12
----101 1
----010 1
0-0-110 1
1-1-001 1
0-10110 1
110-001 1
0011110 1
000-011 1
11-1100 1
0010011 1
1110100 1
0101000 1
.e
```

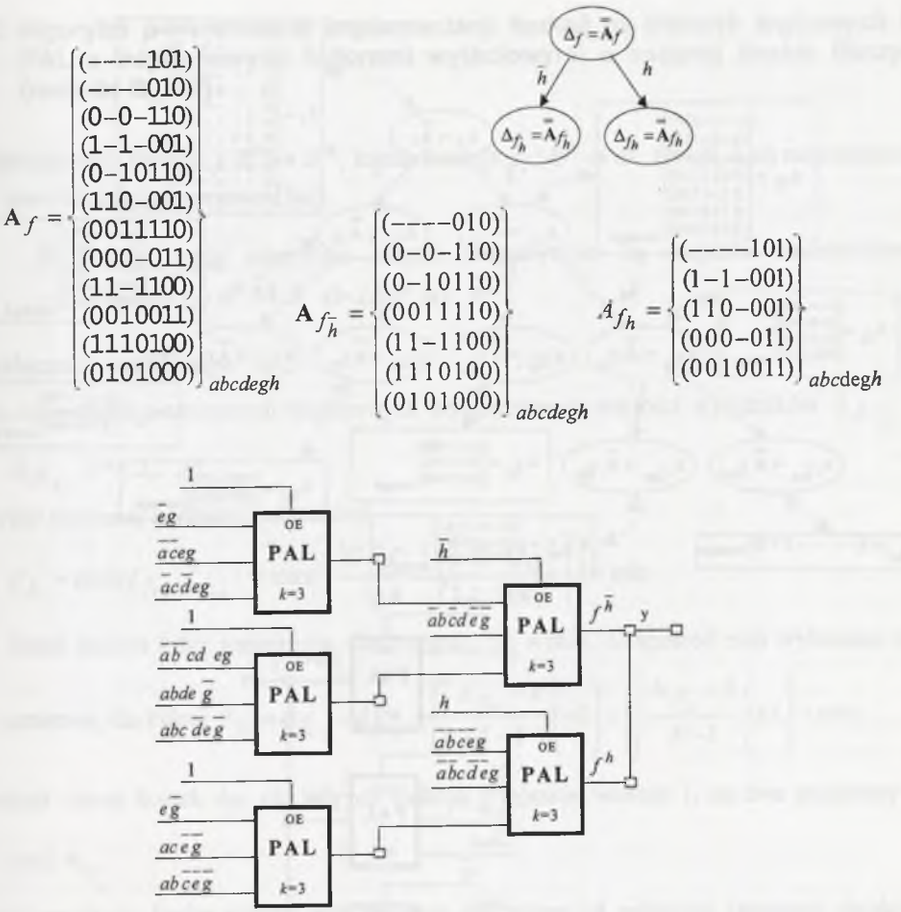
b)

	A	b	c	d	e	g	**h**
	-	-	-	-	1	0	1
	-	-	-	-	0	1	0
	0	-	0	-	1	1	0
	1	-	1	-	0	0	1
	0	-	1	0	1	1	0
	1	1	0	-	0	0	1
	0	0	1	1	1	1	0
	0	0	0	-	0	1	1
	1	1	-	1	1	0	0
	0	0	1	0	0	1	1
	1	1	1	0	1	0	0
	0	1	0	1	0	0	0
	$\Delta_{f_{\bar{a}}}=8$	$\Delta_{f_{\bar{b}}}=8$	$\Delta_{f_{\bar{c}}}=7$	$\Delta_{f_{\bar{d}}}=9$	$\Delta_{f_{\bar{e}}}=6$	$\Delta_{f_{\bar{g}}}=6$	$\Delta_{f_{\bar{h}}}=7$
	$\Delta_{f_a}=6$	$\Delta_{f_b}=9$	$\Delta_{f_c}=8$	$\Delta_{f_d}=9$	$\Delta_{f_e}=6$	$\Delta_{f_g}=6$	$\Delta_{f_h}=5$
	$\xi_{f_{\bar{a}}}=2$	$\xi_{f_{\bar{b}}}=2$	$\xi_{f_{\bar{c}}}=2$	$\xi_{f_{\bar{d}}}=2$	$\xi_{f_{\bar{e}}}=2$	$\xi_{f_{\bar{g}}}=2$	$\xi_{f_{\bar{h}}}=2$
	$\xi_{f_a}=2$	$\xi_{f_b}=2$	$\xi_{f_c}=2$	$\xi_{f_d}=2$	$\xi_{f_e}=2$	$\xi_{f_g}=2$	$\xi_{f_h}=2$
	$\delta_{f_{\bar{a}}}=4$	$\delta_{f_{\bar{b}}}=4$	$\delta_{f_{\bar{c}}}=3$	$\delta_{f_{\bar{d}}}=4$	$\delta_{f_{\bar{e}}}=3$	$\delta_{f_{\bar{g}}}=3$	$\delta_{f_{\bar{h}}}=3$
	$\delta_{f_a}=3$	$\delta_{f_b}=4$	$\delta_{f_c}=4$	$\delta_{f_d}=4$	$\delta_{f_e}=3$	$\delta_{f_g}=3$	$\delta_{f_h}=2$
ξ_f	2	2	2	2	2	2	2
δ_f	7	8	7	8	6	6	**5**

Rys. 3.8. Metoda wyszukiwania zmiennej podziału

Fig. 3.8. The method of searching for the partitioning variable

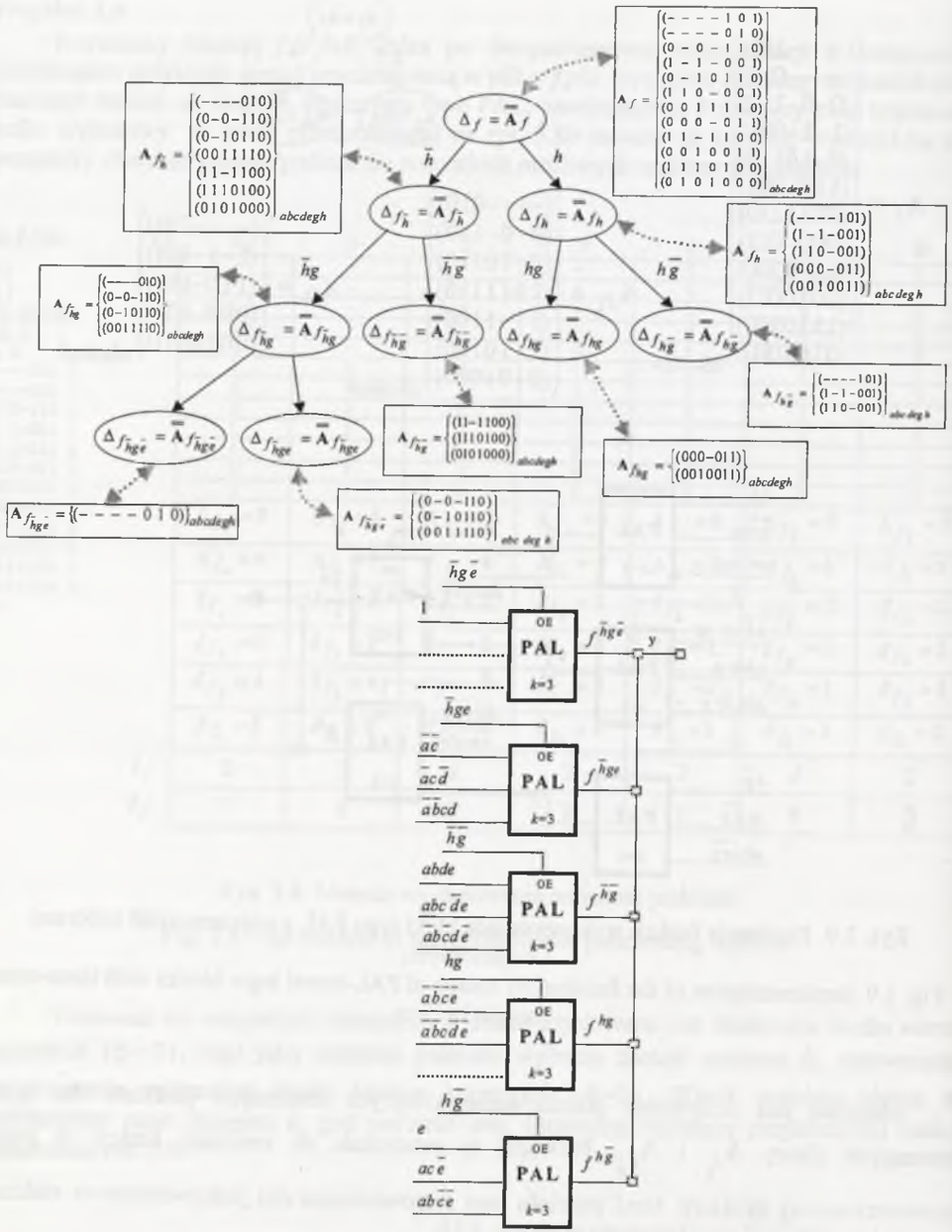
Ponieważ dla wszystkich zmiennych podziału uzyskiwana jest identyczna liczba warstw logicznych ($\xi_f=2$), stąd jako zmienna podziału wybrana zostaje zmienna *h*, zapewniająca wykorzystanie minimalnej liczby bloków logicznych ($\delta_f=5$). Wynik podziału zbioru $\mathbf{A}_f$, implikowany przez zmienną *h*, graf podziału oraz ostateczną realizację przykładowej funkcji przedstawia rys. 3.9.



Rys. 3.9. Realizacja funkcji wykorzystująca bloki typu PAL z trójstanowymi buforami wyjściowymi

Fig. 3.9. Implementation of the function by means of PAL-based logic blocks with three-state output buffers

Możliwe jest oczywiście poszukiwanie kolejnych zmiennych podziału dla kostek tworzących zbiory $\mathbf{A}_{f_{\bar{h}}}$ i $\mathbf{A}_{f_{\bar{h}}}$. Prowadzi to ostatecznie do realizacji funkcji w postaci jednowarstwowej struktury. Graf podziału oraz odpowiadająca mu jednowarstwowa realizacja przykładowej funkcji przedstawione są na rys. 3.10.



Rys.3.10. Jednowarstwowa realizacja funkcji
Fig. 3.10. Single-stage implementation of the function

3.4.2. Algorytm *p*-warstwowej implementacji funkcji na blokach logicznych typu PAL z trójstanowymi buforami wyjściowymi o zadanej liczbie iloczynów (metoda B_PW)

1. Minimalizacja funkcji $f: B^n \to B^m$; każda funkcja $f_i : B^n \to B$ (i=1,2,...,m) minimalizowana jest oddzielnie (espresso -Dso).

   W dalszej części algorytmu kolejno rozpatrywane są wszystkie zminimalizowane funkcje $f_i : B^n \to B$ *(i=1,2,...,m)*

2. Rozłączanie implikantów

3. Dla wszystkich n-zmiennych wejściowych wyznaczane są wartości wyróżników $\Delta_{fi\,\bar{x}_j}$ oraz $\Delta_{fi\,x_j}$ (j=1,2,...,n).

4. Wybór zmiennej podziału x, dla której

$$\xi_{f_i} = \max(\xi_{f_{i\,x}}, \xi_{f_{i\,\bar{x}}}) = \max\left(\left\lceil \frac{\lg(\Delta_{f_{i\,x}})}{\lg k} \right\rceil, \left\lceil \frac{\lg(\Delta_{f_{i\,\bar{x}}})}{\lg k} \right\rceil\right) = \min .$$

   Jeżeli istnieje kilka zmiennych, dla których $\xi_{f_i} = \min$, to spośród nich wybierana jest ta zmienna, dla której $\delta_{f_i} = \delta_{f_{i\,x}} + \delta_{f_{i\,\bar{x}}} = \left(\left\lceil \frac{\Delta_{fi_x} - k}{k-1} \right\rceil + 1\right) + \left(\left\lceil \frac{\Delta_{fi_{\bar{x}}} - k}{k-1} \right\rceil + 1\right) = \min .$

5. Podział zbioru kostek A_{fi}, dla których funkcja przyjmuje wartość 1, na dwa podzbiory $\mathbf{A}_{fi_x}$ oraz $\mathbf{A}_{fi_{\bar{x}}}$.

6. Jeżeli uzyskana liczba warstw jest mniejsza lub równa od założonej (parametr wejściowy) wartości p ($\xi_{f_i} \le p$), to realizacja funkcji, opisanych przez zbiory kostek, powstających po podziale, tzn. $\mathbf{A}_{fi_x}$ oraz $\mathbf{A}_{fi_{\bar{x}}}$. W przypadku konieczności wprowadzania sprzężeń zwrotnych wykorzystywana jest strategia klasyczna z minimalizacją liczby warstw.

7. Jeżeli uzyskana liczba warstw jest większa od założonej wartości p ($\xi_{f_i} > p$), to poszukiwane są zmienne podziałów dla uzyskanych podzbiorów $\mathbf{A}_{fi_x}$ oraz $\mathbf{A}_{fi_{\bar{x}}}$ zgodnie z zasadami zawartymi w punkcie 4. Zmienne podziału poszukiwane są dla odpowiednich (wymagających podziału) podzbiorów, aż do spełnienia warunku punktu 6, tzn. $\xi_{f_i} \le p$.

3.4.3. Metoda wyszukiwania kostek podziału, wykorzystywana w jednowarstwowej implementacji funkcji w strukturach typu PAL z trójstanowymi buforami wyjściowymi

Metoda wyszukiwania zmiennych podziału przedstawiona w rozdziale 3.4.1 pozwala na iteracyjny podział implikantów funkcji, stwarzając możliwość poszukiwania rozwiązań o zadanych własnościach dynamicznych. Stanowi ona rozszerzenie szybkiej strategii podziału zaprezentowanej w pracy [Kani00g]. Szczególnie cenną właściwością metod syntezy wykorzystujących trójstanowe bufory wyjściowe jest znaczna (w stosunku do metody klasycznej) minimalizacja liczby warstw logicznych. Powoduje to, że na szczególną uwagę zasługuje metoda umożliwiająca jednowarstwową implementację funkcji. Próby poszukiwania efektywnej jednowarstwowej metody realizacji funkcji w strukturach typu PAL zawierających trójstanowe bufory wyjściowe zostały przedstawione w pracach [Kani99a, Kani99b]. Okazuje się, że można zwiększyć efektywność metod syntezy zaprezentowanych w przytoczonych pracach poprzez lepszy dobór kostek podziału, będący wynikiem analizy parametrów charakteryzujących uzyskiwane podziały.

Iteracyjny dobór zmiennych podziału, zaprezentowany w rozdziale 3.4.2, upraszcza podział implikantów w *i*-tym kroku, lecz w pewnych sytuacjach prowadzi do nieefektywnego wykorzystania bloków logicznych typu PAL z trójstanowymi buforami wyjściowymi. Dobór zmiennej podziału w *i*-tym kroku uzależniony jest od parametrów określających liczbę warstw logicznych oraz liczbę bloków struktury powstającej po podziale. Dojście do realizacji jednowarstwowej wymaga wielokrotnego poszukiwania zmiennych podziału.

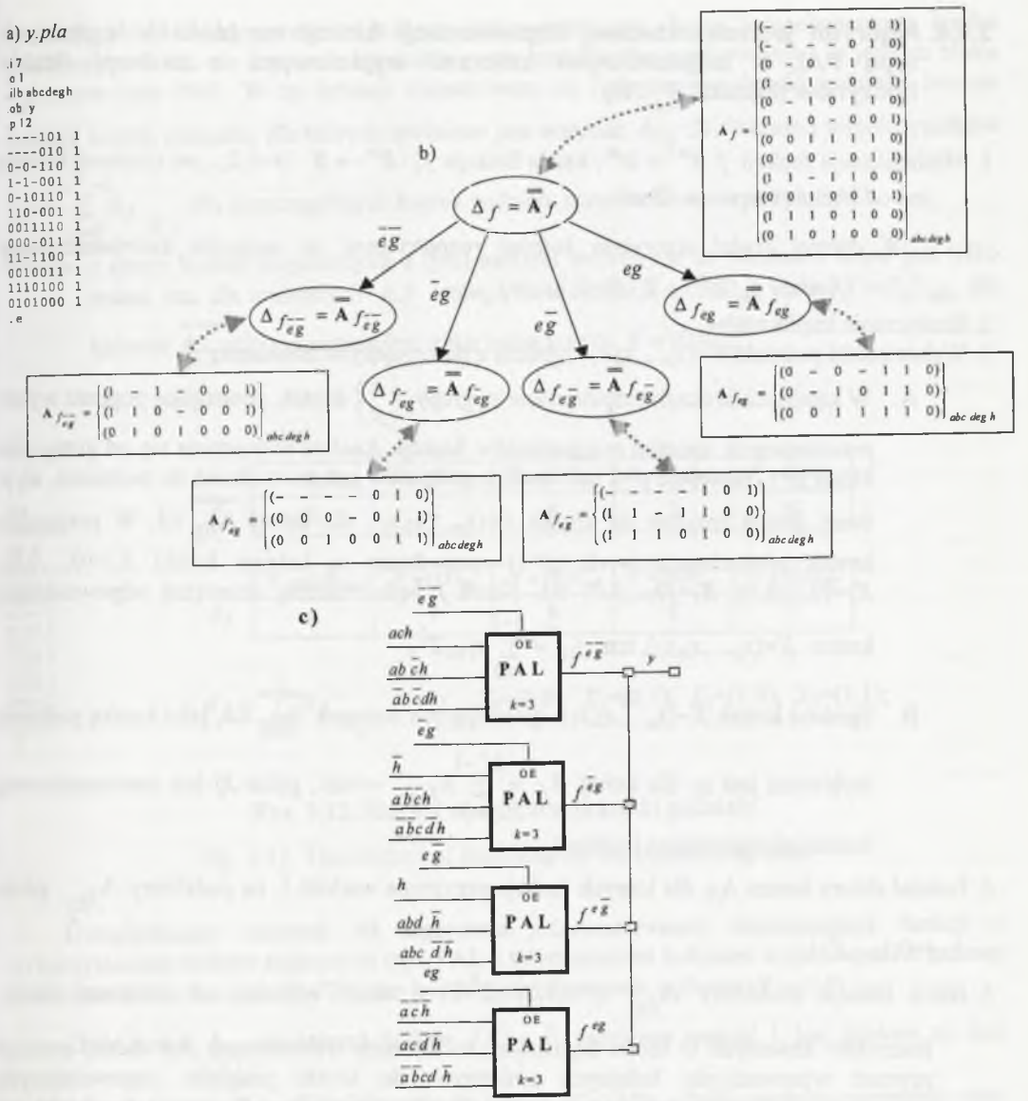
W wielu sytuacjach lepsze rozwiązania uzyskuje się poszukując od razu kilku elementowych kostek podziału, uzależniając ich wybór od parametrów układu jednowarstwowego, powstającego po podziale.

Przykład 3.9

Rozważmy ponownie funkcję $f: B^7 \to B$ analizowaną w przykładzie 3.8, która po dwupoziomowej minimalizacji z rozłączaniem implikantów przyjmuje postać przedstawioną w pliku *y.pla* (rys.3.11a). Załóżmy, że poszukujemy realizacji funkcji na blokach logicznych typu PAL, zawierających 3 iloczyny oraz trójstanowy bufor wyjściowy.

Dobór dwuelementowej kostki podziału $X=(0,0)_{eg}$ pozwala uzyskać lepsze rozwiązanie (rys. 3.11) od uzyskiwanego w metodzie B_PW (rys. 3.10).

Dobór w pierwszym kroku dwuelementowej kostki podziału, dla której $\mathbf{A}_{f_X} \leq k$, wprowadza ograniczenie w wyborze kolejnych kostek podziału. Przyjęto, że kolejne kostki podziału przyjmują postać $X'=(0,1)_{eg}$; $X''=(1,0)_{eg}$; $X'''=(1,1)_{eg}$. Jeżeli $\mathbf{A}_{f_{X'}} > k$ lub $\mathbf{A}_{f_{X''}} > k$ lub $\mathbf{A}_{f_{X'''}} > k$, to odpowiednia kostka X', X'', X''' jest „rozszerzana" o kolejne zmienne tak, aby zapewnić realizowalność w postaci struktury jednowarstwowej. Sytuacja taka nie występuje w rozważanym przykładzie. Ostateczne rozwiązanie przedstawione jest na rys. 3.11.c.



Rys. 3.11. Jednowarstwowa realizacja funkcji

Fig. 3.11. Single-stage implementation of the function

3.4.4. Algorytm jednowarstwowej implementacji funkcji na blokach logicznych typu PAL z trójstanowymi buforami wyjściowymi o zadanej liczbie iloczynów (metoda B_JW)

1. Minimalizacja funkcji $f: B^n \rightarrow B^m$; każda funkcja $f_i : B^n \rightarrow B$ (i=1,2,...,m) minimalizowana jest oddzielnie (espresso -Dso).

   W dalszej części algorytmu kolejno rozpatrywane są wszystkie zminimalizowane funkcje $f_i : B^n \rightarrow B$ *(i=1,2,...,m).*

2. Rozłączanie implikantów
3. Wybór kostki podziału $X=(x_p,...,x_2,x_1)$ zgodnie z następującymi zasadami:

   A. W kolejnych krokach rozpatrywane są grupy $\binom{n}{p}$ kostek, powstające poprzez wybór p-zmniennych spośród n-argumentów funkcji. Analiza rozpoczyna się od grupy, dla której p=1, następnie p=2 itd. Analiza podziałów jest prowadzona do momentu, aż w danej grupie znajdzie się kostka $X=(x_p,...,x_2,x_1)$, dla której $\overline{\overline{\mathbf{A}_{f_i X}}} \leq k$. W przypadku kostek wieloelementowych (p>1) sprawdzane są kolejno kostki X_1=(0,...,0,0), X_2=(0,...,0,1), X_3=(0,...,1,0) itd. Niech j będzie liczbą dziesiętną odpowiadającą kostce $X=(x_p,...,x_2,x_1)$, tzn. $j_{10} = \sum_{l=0}^{p-1} x_{l+1} 2^l$.

   B. Spośród kostek $X=(x_p,...,x_2,x_1)$, spełniających warunek $\overline{\overline{\mathbf{A}_{f_i X}}} \leq k$, jako kostkę podziału wybierana jest ta, dla której $\delta_{f_i} = \sum_{j=0}^{2^p-1} \delta_{f_i X_j} = \min$, gdzie Xj jest zerowymiarową kostką odpowiadającą liczbie j.

4. Podział zbioru kostek $\mathbf{A}_{f_i}$, dla których funkcja przyjmuje wartość 1, na podzbiory $\mathbf{A}_{f_i X_j}$, gdzie j=0,1,...,2^p-1.
5. Jeżeli istnieją podzbiory $\mathbf{A}_{f_i X_j}$ (j=0,1,2,...,2^p-1) o mocy większej od założonej liczby iloczynów zawartych w bloku logicznym, to dla nich wykonywany jest dalszy podział poprzez wprowadzenie kolejnych zmiennych do kostki podziału, zapewniających ostatecznie realizację jednowarstwową. Zasady "rozszerzania" kostki podziału są identyczne z warunkami podanymi w punkcie 3.

Przykład 3.10

Rozważmy funkcję $f : B^5 \rightarrow B$ należącą do układu testowego rd53. Postać funkcji po dwupoziomowej minimalizacji z rozłączaniem implikantów przedstawiona jest w pliku *y.pla* (rys.3.12a). Załóżmy, że poszukujemy realizacji funkcji na blokach logicznych typu PAL,

zawierających 3 iloczyny oraz trójstanowy bufor wyjściowy. Żadna jednoelementowa kostka (zmienna) podziału nie zapewnia uzyskania grupy implikantów realizowalnych w jednym bloku logicznym typu PAL. W tej sytuacji rozpatrywane są 2-elementowe kostki podziału. Istnieje dziesięć kostek podziału, dla których spełniony jest warunek $\overline{\overline{\mathbf{A}_{f_X}}} \leq k$. Wartości współczynników $\delta_f = \sum_{j=0}^{2^p-1} \delta_{f\,X_j}$ dla poszczególnych kostek podziału przedstawione są na rys. 3.12b.

Uwaga: z grupy kostek skojarzonych z tymi samymi zmiennymi do rozważań brana jest tylko jedna, tzn. dla zmiennych a,b istnieją dwie kostki $X=(0,0)_{ab}$ oraz $X=(1,1)_{ab}$, dla których $\overline{\overline{\mathbf{A}_{f_X}}} \leq k$; rozważana jest tylko jedna kostka $X=(0,0)_{ab}$.

a) *y.pla*

```
.i 5
.o 1
.ilb abcde
.ob. y
.p 11
1100- 1
0011- 1
011-0 1
01-01 1
1001- 1
1-100 1
-1010 1
10110 1
10001 1
-0101 1
0-011 1
.e
```

b)

	$X=(0,0)_{ab}$	$X=(0,0)_{ac}$	$X=(0,0)_{ad}$	$X=(0,0)_{ae}$	$X=(0,0)_{bc}$
δ_f	6	6	6	7	6

	$X=(0,0)_{bd}$	$X=(1,1)_{be}$	$X=(0,0)_{cd}$	$X=(0,0)_{ce}$	$X=(0,0)_{de}$
δ_f	6	7	**5**	6	7

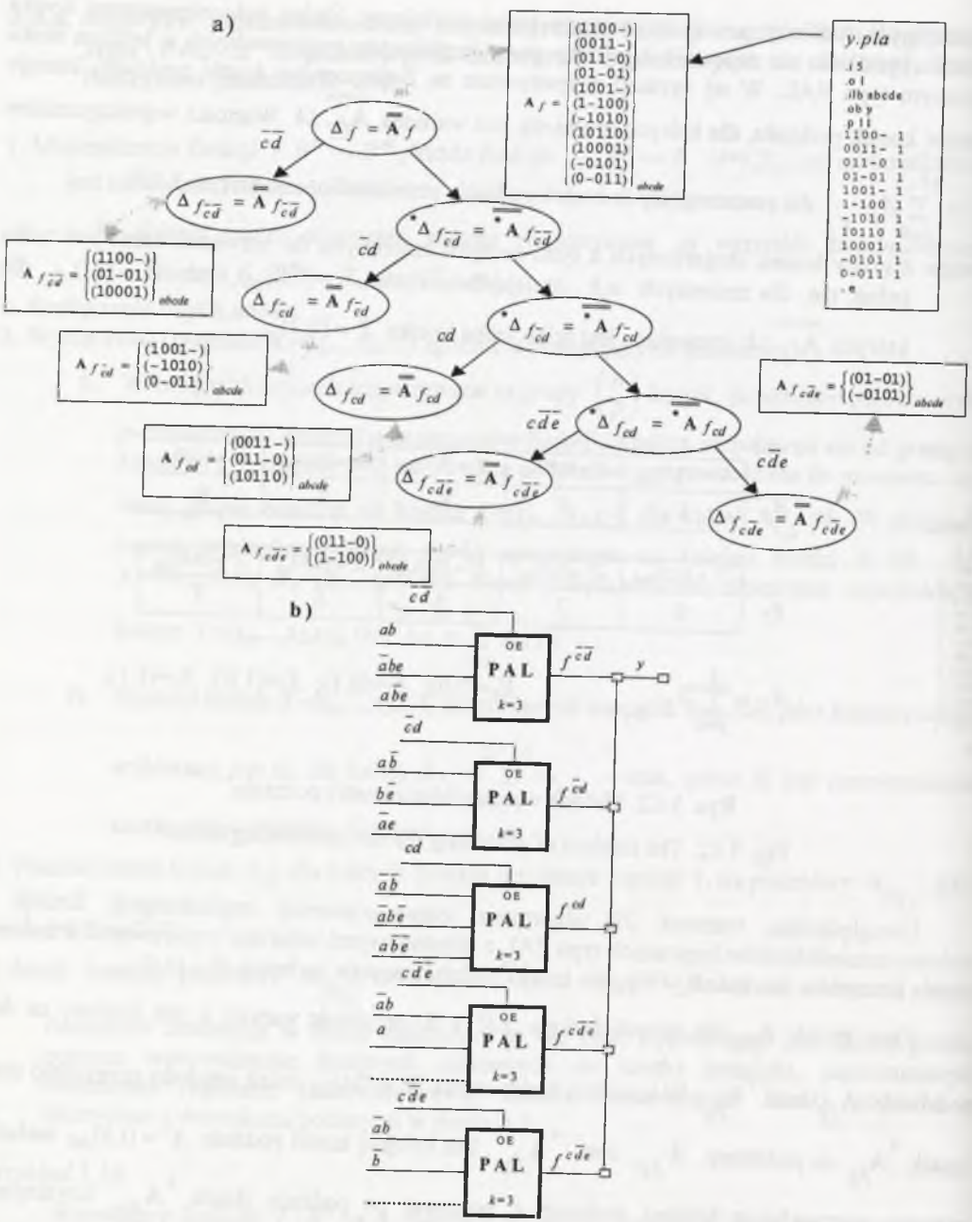
$\delta_f = \sum_{j=0}^{3} \delta_{f\,X_j}$ $\quad X_0$=(0,0); X_1=(0,1); X_2=(1,0); X_3=(1,1);

Rys. 3.12. Metoda wyszukiwania kostki podziału

Fig. 3.12. The method of searching for the partitioning cube

Uwzględniając warunek 3B algorytmu jednowarstwowej implementacji funkcji z wykorzystaniem bloków logicznych typu PAL z trójstanowymi buforami wyjściowymi o zadanej liczbie iloczynów (metoda B_JW) jako kostka podziału zostaje wybrana $X=(0,0)_{cd}$.

Zbiór kostek $\mathbf{A}_f$, dla których funkcja $f: B^5 \rightarrow B$ przyjmuje wartość 1, jest dzielony na dwa podzbiory: $\mathbf{A}_{f_X}$ oraz ${}^*\mathbf{A}_{f_X}$. W kolejnym kroku kostka $X'=(0,1)_{cd}$ dzieli uzyskany poprzednio zbiór kostek ${}^*\mathbf{A}_{f_X}$ na podzbiory $\mathbf{A}_{f_{X'}}$ oraz ${}^*\mathbf{A}_{f_{X'}}$. Dla kolejnej kostki podziału $X''=(1,0)_{cd}$ zachodzi potrzeba wprowadzenia kolejnej zmiennej e, ponieważ po podziale zbioru ${}^*\mathbf{A}_{f_{X'}}$ uzyskujemy podzbiór, dla którego $\overline{\overline{\mathbf{A}_{f_{X''}}}} > 3$. Graf podziału oraz końcowe jednowarstwowe rozwiązanie przedstawione są na rys. 3.13.



Rys. 3.13. Jednowarstwowa realizacja funkcji (metoda B_JW)

Fig. 3.13. Single-stage implementation of the function (the B_JW method)

3.5. Wyniki eksperymentów

Przedstawione w poprzednich podrozdziałach metody syntezy są szczególnie atrakcyjne ze względu na uzyskiwaną liczbę warstw. Prowadząc eksperymenty zwrócono również uwagę na liczbę wykorzystywanych bloków logicznych.

W pierwszej kolejności porównano zaproponowaną *p*-warstwową realizację funkcji na blokach typu PAL z trójstanowymi buforami wyjściowymi o zadanej liczbie iloczynów (metoda B_PW), z klasyczną realizacją funkcji z minimalizacją liczby warstw (metoda K_LW).

W pierwszej części tab. 3.1 przedstawione są wyniki syntezy układów testowych metodą klasyczną z minimalizacją liczby warstw (metoda K_LW). W kolumnach, w których występuje litera „B", podano liczbę wykorzystywanych *k*-iloczynowych bloków PAL, natomiast w kolumnach z literą „W" - liczbę warstw wynikowego układu. W kolejnych częściach tabeli zawarte są wyniki syntezy zaproponowaną *p*-warstwową (p=1,...,5) realizacją funkcji wykorzystującą wyjściowe bufory trójstanowe. Ze względu na założoną (parametr wejściowy) liczbę warstw w tej części tabeli zawarte są tylko liczby wykorzystywanych *k*-iloczynowych bloków PAL. Symbole „x" oznaczają brak rozwiązania. Konstrukcja algorytmu syntezy wykorzystywanego w metodzie B_PW zapewnia znalezienie rozwiązania o nie większej liczbie warstw niż liczba warstw uzyskiwana metodą klasyczną z minimalizacją liczby warstw (K_LW). W tej sytuacji puste kratki odpowiadają sytuacjom, w których uzyskiwano mniejszą liczbę warstw od zadanej.

Analizując uzyskane wyniki okazuje się, że w większości przypadków (85%) metodą B_PW można znaleźć rozwiązanie, wykorzystujące mniejszą liczbę warstw niż metodą klasyczną z minimalizacją liczby warstw (K_LW). Przykładowo układ testowy *rd84* metodą K_LW można zrealizować wykorzystując 142 bloki logiczne zawierające 3-iloczyny w postaci struktury 5-warstwowej. Metoda B_PW umożliwia realizację tego samego układu testowego w postaci struktury 2-warstwowej, używając jednak 162 bloki logiczne.

Istnieją przypadki (8%), w których wraz z oczywistym zmniejszeniem liczby warstw w stosunku do metody klasycznej dochodzi również do zmniejszenia liczby wykorzystanych bloków logicznych. Przykładowo, tak dzieje się dla układu testowego *5xp1*. Metodą K_LW można zrealizować ten układ testowy, wykorzystując 35 bloków logicznych zawierających 3-iloczyny w postaci struktury 3-warstwowej. Metoda B_PW umożliwia realizację tego samego układu testowego w postaci struktury jednowarstwowej, używając tylko 32 bloki logiczne. Dlaczego tak się dzieje? Rozważmy prosty przykład. Załóżmy, że chcemy zrealizować funkcję będącą sumą 6 implikantów. Metoda klasyczna z minimalizacją liczby warstw pozwala na realizację dwuwarstwową, wykorzystującą trzy bloki logiczne typu PAL, zawierające 3 iloczyny. Jeżeli istnieje zmienna podziału implikująca podział zbioru implikantów na dwa podzbiory zawierające po trzy elementy, to możliwa jest realizacja jednowarstwowa, wykorzystująca tylko 2 bloki logiczne typu PAL (patrz przykład 3.1). Powyższy prosty przykład pokazuje istotę minimalizacji liczby wykorzystywanych bloków w metodzie B_PW oraz wskazuje na warunki, jakie powinna spełniać funkcja, aby jej realizacja metodą B_PW doprowadziła do minimalizacji liczby bloków logicznych. Warunki te spełnione są w kilku przypadkach dla układów testowych takich jak: *5xp1*, *bw*, *clip*, *f51m*, *squar5*, *xor5* i *z5xp1*.

Tabela 3.1

Porównanie klasycznej realizacji funkcji z minimalizacją liczby warstw (metoda K_LW) z zaproponowaną *p*-warstwową realizacją funkcji na blokach logicznych typu PAL z trójstanowymi buforami wyjściowymi o zadanej liczbie iloczynów; (metoda B_PW)

metoda K_LW - **K**lasyczna metoda realizacji funkcji w strukturach typu PAL z minimalizacją **L**iczby **W**arstw

metoda B_PW - metoda *p* - **W**arstwowej realizacji funkcji, wykorzystująca trójstanowe **B**ufory wyjściowe

B - liczba bloków; W - liczba warstw uzyskanych w metodzie K_LW; p - zadana liczba warstw w metodzie B_PW

i - liczba wejść układu testowego; o - liczba wyjść układu testowego

Układ	i	o	K_LW						B_PW p=1						B_PW p=2						B_PW p=3						B_PW p=4			B_PW p=5		
			k=3	k=4	k=5	k=6	k=7	k=8	k=3	k=4	k=5	k=6	k=7	k=8	k=3	k=4	k=5	k=6	k=7	k=8	k=3	k=4	k=5	k=6	k=7	k=8	k=3	k=4	k=5	k=3	k=4	k=5
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Liczba przypadków, w których nie znaleziono rozwiązań: 11 5 3 2 1

x - brak rozwiązania

"pusta kratka" - uzyskano mniejszą liczbę warstw od zadanej wartości *p*

Niestety, dla założonej wartości liczby warstw metoda B_PW nie zawsze znajduje rozwiązanie. W przypadku realizacji jednowarstwowej metoda B_PW nie doprowadziła do rozwiązania w 21 przypadkach (9%). Najczęściej nie znajdywano rozwiązań (37% przypadków) dla małych bloków logicznych typu PAL (k=3). Nie jest to jednak wielkim problemem, ponieważ ostateczna metoda syntezy może być skomponowana w taki sposób, że w przypadku braku rozwiązania w postaci układu i-warstwowego można rozpocząć poszukiwania struktury (i+1)-warstwowej. Taka strategia syntezy w większości przypadków (85%) prowadzi do rozwiązania wykorzystującego mniejszą liczbę warstw niż metoda klasyczna.

Szczególnie cenną właściwością metod syntezy wykorzystujących trójstanowe bufory wyjściowe jest znaczna (w stosunku do metody klasycznej) minimalizacja liczby warstw logicznych. Powoduje to, że na szczególną uwagę zasługuje metoda umożliwiająca jednowarstwową implementację funkcji.

W tabeli 3.2 przedstawiono porównanie klasycznej metody syntezy z minimalizacją liczby warstw (metoda K_LW) z jednowarstwową realizacją wykorzystującą wyjściowe bufory trójstanowe (metoda B_JW). Niestety, podobnie jak metoda B_PW, metoda B_JW nie zawsze znajduje rozwiązanie. Jeżeli rozwiązanie zostanie znalezione, to uzyskuje się jednak istotne ograniczenie liczby warstw przy wykorzystywaniu znacznie większej liczby bloków logicznych. Analizując sumaryczną liczbę bloków oraz sumaryczną liczbę warstw (pomijając pola zaznaczone na szaro) widać, że znaczne ograniczenie sumarycznej liczby warstw występujące w metodzie B_JW (k=7 zmniejszenie sumarycznej liczby warstw o 100%) pociąga za sobą procentowo mniejsze zwiększenie sumarycznej liczby wykorzystywanych bloków logicznych (k=7 zwiększenie sumarycznej liczby bloków o 54%). Podobnie wygląda sytuacja dla innych wartości k, przy czym większe korzyści uzyskuje się dla mniejszych bloków logicznych (dla bloków logicznych z 3 iloczynami metodą B_JW uzyskuje się 2,6 razy mniejszą sumaryczną liczbę warstw logicznych przy porównywalnej sumarycznej liczbie wykorzystanych bloków logicznych). Cóż z tego, skoro w wielu przypadkach nie udaje się znaleźć rozwiązania (dla k=3 brak 11 rozwiązań, tzn. 37 %).

Tabela 3.3 zawiera porównanie dwóch zaproponowanych metod syntezy wykorzystujących wyjściowe bufory trójstanowe. Porównano metody prowadzące do rozwiązań jednowarstwowych, tzn. metodę B_JW z metodą B_PW realizowaną dla wartości p=1. Obie metody znajdują podobną liczbę rozwiązań. Metoda B_JW wykorzystuje nieznacznie mniejszą liczbę bloków logicznych (1-2%). Dodatkowo metoda B_JW jest szybsza (dla rozpatrywanego zbioru układów testowych o około 5%) od metody B_PW (p=1).

Przedstawione eksperymenty wskazują na korzyści wynikające z metody B_JW w poszukiwaniu rozwiązań jednowarstwowych i możliwość poszukiwania rozwiązań p-warstwowych (p>1) metodą B_PW w przypadku braku rozwiązania metodą B_JW. Wyniki tak skomponowanej strategii syntezy, oznaczanej jako B_JPW, oraz porównanie jej z metodą klasyczną zawarte są w tab. 3.4. Metodą B_PW szukano oczywiście rozwiązań zawierających minimalną liczbę warstw logicznych.

Tabela 3.2

Porównanie klasycznej realizacji funkcji z minimalizacją liczby warstw (metoda K_LW) z zaproponowaną jednowarstwową realizacją funkcji na blokach logicznych typu PAL z trójstanowymi buforami wyjściowymi (metoda B_JW)

metoda K_LW - **K**lasyczna metoda realizacji funkcji w strukturach typu PAL z minimalizacją **L**iczby **W**arstw
metoda B_JW - metoda **J**edno**W**arstowej realizacji funkcji, wykorzystująca trójstanowe **B**ufory wyjściowe
B - liczba bloków; W - liczba warstw uzyskanych w metodzie K_LW * - bez wyróżnionych pół

				K_LW														B_JW							
				k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16		k=	k=	k=	k=	k=	k=	k=	k=
	Układ testowy	i	o	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	3	4	5	6	7	8	12	16
1	5xp1	7	10	35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2	31	25	20	18	17	15	12	11
2	9sym	9	1	43	5	29	4	22	3	18	3	15	3	13	3	8	2	6	2	x	101	83	66	56	41	25	17
3	9symml	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	x	103	79	57	48	38	25	17
4	b12	15	9	24	3	16	2	14	2	13	2	10	2	10	2	9	1	9	1	30	21	16	13	10	10	9	9
5	bw	5	28	49	2	38	2	30	2	28	1	28	1	28	1	28	1	28	1	48	38	30	28	28	28	28	28
6	clip	9	5	73	4	49	3	38	3	30	3	26	2	22	2	14	2	12	2	x	67	56	44	33	30	17	14
7	con1	7	2	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1	4	3	2	2	2	2	2	2
8	duke2	22	29	99	3	72	3	61	2	49	2	44	2	41	2	35	2	31	2	x	x	71	56	47	46	36	31
9	ex1010	10	10	242	4	165	3	124	3	101	3	85	3	74	2	49	2	38	2	x	463	354	285	233	198	114	77
10	f51m	8	8	37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2	34	25	22	17	16	15	11	10
11	inc	7	9	23	3	17	2	16	2	13	2	11	2	11	2	9	1	9	1	24	18	16	14	11	11	9	9
12	ldd	9	19	34	2	25	2	25	2	21	2	19	1	19	1	19	1	19	1	38	29	25	21	19	19	19	19
13	misex1	8	7	14	2	12	2	8	2	7	1	7	1	7	1	7	1	7	1	13	12	8	7	7	7	7	7
14	misex2	25	18	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1	19	19	18	18	18	18	18	18
15	pcle	19	9	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1	10	9	9	9	9	9	9	9
16	rd53	5	3	15	3	11	2	8	2	6	2	6	2	6	2	4	2	3	1	16	10	9	7	7	5	4	3
17	rd73	7	3	70	4	47	3	36	3	29	3	24	3	20	2	14	2	11	2	x	x	x	x	35	24	17	11
18	rd84	8	4	142	5	95	4	72	4	58	3	49	3	42	3	28	2	21	2	x	x	92	81	76	56	39	25
19	root	8	5	35	4	24	3	18	3	16	2	13	2	11	2	9	2	7	2	37	26	21	18	15	13	9	8
20	sao2	10	4	36	3	24	3	19	2	15	2	14	2	11	2	7	2	7	2	x	x	x	47	30	25	11	8
21	sqn	7	3	21	3	15	3	11	2	10	2	8	2	7	2	5	2	4	2	25	17	12	10	8	7	5	4
22	sqr6	6	12	31	3	24	2	20	2	16	2	16	2	15	2	12	2	11	1	31	22	20	17	16	15	12	11
23	squar5	5	8	14	2	11	2	9	2	9	2	9	2	8	1	8	1	8	1	13	10	9	9	9	8	8	8
24	table3	14	14	264	4	181	4	135	3	110	3	94	3	80	3	53	2	40	2	x	x	137	113	96	84	56	47
25	vg2	25	8	53	4	37	3	28	3	22	3	20	2	19	2	13	2	11	2	x	67	53	36	30	26	16	14
26	x2	10	7	13	2	10	2	10	2	7	1	7	1	7	1	7	1	7	1	14	11	10	7	7	7	7	7
27	xor5	5	1	8	2	5	2	4	2	3	2	3	2	3	2	2	2	1	1	8	4	4	4	4	2	2	1
28	z4m1	7	4	29	4	19	3	15	3	13	2	11	2	9	2	7	2	5	2	25	20	19	13	10	9	8	5
29	z5xp1	7	10	37	3	27	3	23	2	19	2	17	2	15	2	12	2	11	2	34	27	22	19	17	16	12	12
30	z9sym	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	x	107	82	62	47	42	27	19
	sumaryczna liczba bloków*			452		674		745		634		625		561		427		368		454	1254	1299	1098	961	826	574	461
	sumaryczna liczba warstw*			50		64		62		58		60		57		50		47		19	25	28	29	30	30	30	30
	brak rozwiązań																			11	5	2	1	0	0	0	0

* - sumaryczne liczby bloków oraz warstw obliczone zostały z wyłączeniem wartości zawartych w wyróżnionych polach x - brak rozwiązania

Tabela 3.3

Porównanie syntezy funkcji na blokach logicznych typu PAL z trójstanowymi buforami wyjściowymi metodą B_PW oraz B_JW

metoda B_JW - metoda **J**edno**W**arstowej realizacji funkcji, wykorzystująca trójstanowe **B**ufory wyjściowe
metoda B_PW - metoda ***p*** - **W**arstwowej realizacji funkcji, wykorzystująca trójstanowe **B**ufory wyjściowe
i - liczba wejść układu testowego; o - liczba wyjść układu testowego; p - zadana liczba warstw

				B_JW								B_PW *p*=1							
	Układ testowy	i	o	*k*=3	*k*=4	*k*=5	*k*=6	*k*=7	*k*=8	*k*=12	*k*=16	*k*=3	*k*=4	*k*=5	*k*=6	*k*=7	*k*=8	*k*=12	*k*=16
1	5xp1	7	10	31	25	20	18	17	15	12	11	32	25	20	18	17	15	12	11
2	9sym	9	1	x	101	83	66	56	41	25	17	x	101	83	66	56	41	25	17
3	9symml	9	1	x	103	79	57	48	38	25	17	x	103	79	57	48	38	25	17
4	b12	15	9	30	21	16	13	10	10	9	9	30	21	16	13	10	10	9	9
5	bw	5	28	48	38	30	28	28	28	28	28	48	38	30	28	28	28	28	28
6	clip	9	5	x	67	56	44	33	30	17	14	x	67	56	44	34	31	17	15
7	con1	7	2	4	3	2	2	2	2	2	2	4	3	2	2	2	2	2	2
8	duke2	22	29	x	x	71	56	47	46	36	31	x	x	x	x	51	50	36	31
9	ex1010	10	10	x	463	354	285	233	198	114	77	x	467	358	290	235	202	115	79
10	f51m	8	8	34	25	22	17	16	15	11	10	34	26	22	17	16	15	11	10
11	inc	7	9	24	18	16	14	11	11	9	9	24	18	16	14	11	11	9	9
12	ldd	9	19	38	29	25	21	19	19	19	19	38	29	25	21	19	19	19	19
13	misex1	8	7	13	12	8	7	7	7	7	7	13	12	8	7	7	7	7	7
14	misex2	25	18	19	19	18	18	18	18	18	18	19	19	18	18	18	18	18	18
15	pcle	19	9	10	9	9	9	9	9	9	9	10	9	9	9	9	9	9	9
16	rd53	5	3	16	10	9	7	7	5	4	3	18	10	9	7	7	5	4	3
17	rd73	7	3	x	x	x	x	35	24	17	11	x	x	x	x	37	28	19	13
18	rd84	8	4	x	x	92	81	76	56	39	25	x	x	96	81	79	58	38	26
19	root	8	5	37	26	21	18	15	13	9	8	38	26	22	19	15	13	10	8
20	sao2	10	4	x	x	x	47	30	25	11	8	x	x	x	47	30	25	11	8
21	sqn	7	3	25	17	12	10	8	7	5	4	25	17	13	12	10	8	5	4
22	sqr6	6	12	31	22	20	17	16	15	12	11	33	25	20	17	16	15	12	11
23	squar5	5	8	13	10	9	9	9	8	8	8	13	10	9	9	9	8	8	8
24	table3	14	14	x	x	137	113	96	84	56	47	x	x	140	116	98	85	59	47
25	vg2	25	8	x	67	53	36	30	26	16	14	x	67	53	36	30	26	16	15
26	x2	10	7	14	11	10	7	7	7	7	7	14	11	10	7	7	7	7	7
27	xor5	5	1	8	4	4	4	4	2	2	1	8	4	4	4	4	2	2	1
28	z4m1	7	4	25	20	19	13	10	9	8	5	27	18	17	15	10	9	8	5
29	z5xp1	7	10	34	27	22	19	17	16	12	12	34	27	22	19	16	16	12	12
30	z9sym	9	1	x	107	82	62	47	42	27	19	x	107	82	62	47	42	27	19
	Sumaryczna liczba bloków*			454	1254	1228	1042	961	826	574	461	462	1260	1239	1055	976	843	580	468
	Brak rozwiązań			11	5	2	1	0	0	0	0	11	5	3	2	0	0	0	0

* - sumaryczne liczby bloków oraz warstw obliczone zostały z wyłączeniem wartości zawartych w wyróżnionych polach x - brak rozwiązania

Tabela 3.4

Porównanie klasycznej realizacji funkcji z minimalizacją liczby warstw (metoda K_LW) ze strategią syntezy B_JPW wykorzystującą trójstanowe bufory wyjściowe (metoda B_JW; w przypadku braku rozwiązania metoda B_PW zapewniająca minimalną liczbę warstw)

i - liczba wejść układu testowego o - liczba wyjść układu testowego B - liczba bloków W - liczba warstw

	Układ testowy	i	o	K_LW k=3 B	K_LW k=3 W	K_LW k=4 B	K_LW k=4 W	K_LW k=5 B	K_LW k=5 W	K_LW k=6 B	K_LW k=6 W	K_LW k=7 B	K_LW k=7 W	K_LW k=8 B	K_LW k=8 W	K_LW k=12 B	K_LW k=12 W	K_LW k=16 B	K_LW k=16 W	B_JPW k=3 B	B_JPW k=3 W	B_JPW k=4 B	B_JPW k=4 W	B_JPW k=5 B	B_JPW k=5 W	B_JPW k=6 B	B_JPW k=6 W	B_JPW k=7 B	B_JPW k=7 W	B_JPW k=8 B	B_JPW k=8 W	B_JPW k=12 B	B_JPW k=12 W	B_JPW k=16 B	B_JPW k=16 W
1	5xp1	7	10	35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2	31	1	25	1	20	1	18	1	17	1	15	1	12	1	11	1
2	9sym	9	1	43	5	29	4	22	3	18	3	15	3	13	3	8	2	6	2	112	2	101	1	83	1	66	1	56	1	41	1	25	1	17	1
3	9symml	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	109	2	103	1	79	1	57	1	48	1	38	1	25	1	17	1
4	b12	15	9	24	3	16	2	14	2	13	2	10	2	10	2	9	1	9	1	30	1	21	1	16	1	13	1	10	1	10	1	9	1	9	1
5	bw	5	28	49	2	38	2	30	2	28	1	28	1	28	1	28	1	28	1	48	1	38	1	30	1	28	1	28	1	28	1	28	1	28	1
6	clip	9	5	73	4	49	3	38	3	30	3	26	2	22	2	14	2	12	2	77	2	67	1	56	1	44	1	33	1	30	1	17	1	14	1
7	con1	7	2	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1	4	1	3	1	2	1	2	1	2	1	2	1	2	1	2	1
8	duke2	22	29	99	3	72	3	61	2	49	2	44	2	41	2	35	2	31	2	99	3	72	1	71	1	56	1	47	1	46	1	36	1	31	1
9	ex1010	10	10	242	4	165	3	124	3	101	3	85	3	74	2	49	2	38	2	511	2	463	1	354	1	285	1	233	1	198	1	114	1	77	1
10	f51m	8	8	37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2	34	1	25	1	22	1	17	1	16	1	15	1	11	1	10	1
11	inc	7	9	23	3	17	2	16	2	13	2	11	2	11	2	9	1	9	1	24	1	18	1	16	1	14	1	11	1	11	1	9	1	9	1
12	ldd	9	19	34	2	25	2	25	2	21	2	19	1	19	1	19	1	19	1	38	1	29	1	25	1	21	1	19	1	19	1	19	1	19	1
13	misex1	8	7	14	2	12	2	8	2	7	1	7	1	7	1	7	1	7	1	13	1	12	1	8	1	7	1	7	1	7	1	7	1	7	1
14	misex2	25	18	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1	19	1	19	1	18	1	18	1	18	1	18	1	18	1	18	1
15	pcle	19	9	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1	10	1	9	1	9	1	9	1	9	1	9	1	9	1	9	1
16	rd53	5	3	15	3	11	2	8	2	6	2	6	2	6	2	4	2	3	1	16	1	10	1	9	1	9	1	9	1	9	1	4	1	3	1
17	rd73	7	3	70	4	47	3	36	3	29	3	24	3	20	2	14	2	11	2	76	2	49	2	37	2	30	2	35	1	24	1	17	1	11	1
18	rd84	8	4	142	5	95	4	72	4	58	3	49	3	42	3	28	2	21	2	162	2	103	2	92	1	81	1	76	1	56	1	39	1	25	1
19	root	8	5	35	4	24	3	18	3	16	2	13	2	11	2	9	2	7	2	37	1	26	1	21	1	18	1	15	1	13	1	9	1	8	1
20	sao2	10	4	36	3	24	3	19	2	15	2	14	2	11	2	7	2	7	2	61	2	29	2	19	2	47	1	30	1	25	1	11	1	8	1
21	sqn	7	3	21	3	15	3	11	2	10	2	8	2	7	2	5	2	4	2	25	1	17	1	12	1	10	1	8	1	7	1	5	1	4	1
22	sqr6	6	12	31	3	24	2	20	2	16	2	15	2	12	2	11	1	11	1	31	1	22	1	20	1	17	1	16	1	15	1	12	1	11	1
23	squar5	5	8	14	2	11	2	9	2	9	2	9	2	8	1	8	1	8	1	13	1	10	1	9	1	9	1	9	1	8	1	8	1	8	1
24	table3	14	14	264	4	181	4	135	3	110	3	94	3	80	3	53	2	40	2	247	2	179	2	137	1	113	1	96	1	84	1	56	1	47	1
25	vg2	25	8	53	4	37	3	28	3	22	3	20	2	19	2	13	2	11	2	70	2	67	1	53	1	36	1	30	1	26	1	16	1	14	1
26	x2	10	7	13	2	10	2	10	2	7	1	7	1	7	1	7	1	7	1	14	1	11	1	10	1	7	1	7	1	7	1	7	1	7	1
27	xor5	5	1	8	2	5	2	4	2	3	2	3	2	3	2	2	2	1	1	8	1	4	1	4	1	4	1	4	1	2	1	2	1	1	1
28	z4ml	7	4	29	4	19	3	15	3	13	2	11	2	9	2	7	2	5	2	25	1	20	1	19	1	13	1	10	1	9	1	8	1	5	1
29	z5xp1	7	10	37	3	27	3	23	2	19	2	17	2	15	2	12	2	11	2	34	1	27	1	22	1	19	1	17	1	16	1	12	1	12	1
30	z9sym	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	107	2	82	1	62	1	47	1	42	1	27	1	19	1		
	liczba bloków			1558	96	1093	81	861	69	712	63	625	60	561	57	427	50	368	47	2096	42	1686	34	1355	32	1128	31	961	30	826	30	574	30	461	30
	liczba warstw logicznych																																		

Podsumowaniem przedstawionych rozważań mogą być wykresy uwypuklające mocną i słabą stronę zaproponowanej strategii syntezy wykorzystującej wyjściowe bufory trójstanowe. Wartości przedstawione na wykresie 3.1 wyznaczone zostały z zależności $\frac{\sum bloków_{(B\_JPW)} - \sum bloków_{(K\_LW)}}{\sum bloków_{(K\_LW)}}$ 100%, gdzie $\sum bloków_{(K\_LW)}$ i $\sum bloków_{(B\_JPW)}$ oznaczają sumaryczne liczby bloków, wykorzystywanych odpowiednimi metodami syntezy, przedstawione w tab. 3.4.

Podobnie wyznaczono wartości przedstawione na wykresie 3.2 na podstawie zależności $\frac{\sum warstw_{(K\_LW)} - \sum warstw_{(B\_JPW)}}{\sum warstw_{(B\_JPW)}}$ 100%, gdzie $\sum warstw_{(K\_LW)}$ i $\sum warstw_{(B\_JPW)}$ oznaczają sumaryczne liczby warstw logicznych dla poszczególnych wartości *k*.

Wykres 3.1. Porównanie strategii syntezy B_JPW z metodą K_LW pod względem liczby bloków (opis w tekście)

Wykres 3.2. Porównanie strategii syntezy B_JPW z metodą K_LW pod względem liczby warstw logicznych (opis w tekście)

Tabela 3.5

Wpływ procedury rozłączania implikantów na efektywność syntezy funkcji logicznych w strukturach typu PAL z trójstanowymi buforami wyjściowymi

metoda B_JW - metodą **Jedno**Warstowej realizacji funkcji, wykorzystująca trójstanowe **B**ufory wyjściowe
i - liczba wejść układu testowego; o - liczba wyjść układu testowego

	Układ testowy	i	o	B_JW k=3	k=4	k=5	k=6	k=7	k=8	k=12	k=16	B_JW - bez rozłączania implikantów k=3	k=4	k=5	k=6	k=7	k=8	k=12	k=16
1	5xp1	7	10	31	25	20	18	17	15	12	11	x	x	x	18	17	15	12	11
2	9sym	9	1	x	101	83	66	56	41	25	17	x	116	92	75	56	45	28	18
3	9symml	9	1	x	103	79	57	48	38	25	17	x	113	85	61	53	41	26	17
4	b12	15	9	30	21	16	13	10	10	9	9	x	x	x	13	10	10	9	9
5	bw	5	28	48	38	30	28	28	28	28	28	x	38	30	28	28	28	28	28
6	clip	9	5	x	67	56	44	33	30	17	14	x	x	x	x	x	x	x	14
7	con1	7	2	4	3	2	2	2	2	2	2	4	3	2	2	2	2	2	2
8	duke2	22	29	x	x	71	56	47	46	36	31	x	x	x	x	48	48	36	31
9	ex1010	10	10	x	469	361	294	241	206	117	81	x	x	x	321	256	224	120	85
10	f51m	8	8	34	27	22	17	16	15	11	10	x	x	x	17	17	15	11	10
11	inc	7	9	24	18	16	14	11	11	9	9	x	19	16	15	11	11	9	9
12	ldd	9	19	38	29	25	21	19	19	19	19	x	x	26	21	19	19	19	19
13	misex1	8	7	13	12	8	7	7	7	7	7	13	12	8	7	7	7	7	7
14	misex2	25	18	19	19	18	18	18	18	18	18	19	19	18	18	18	18	18	18
15	pcle	19	9	10	9	9	9	9	9	9	9	10	9	9	9	9	9	9	9
16	rd53	5	3	16	10	9	7	7	5	4	3	x	x	9	7	7	5	4	3
17	rd73	7	3	x	x	x	x	39	28	19	11	x	x	x	x	x	x	x	x
18	rd84	8	4	x	x	97	84	81	60	41	26	x	x	x	x	x	x	x	x
19	root	8	5	37	26	21	18	15	13	9	8	x	x	x	24	18	16	10	8
20	sao2	10	4	x	x	x	47	30	25	11	8	x	x	x	x	x	x	20	8
21	sqn	7	3	25	17	13	12	10	8	5	4	x	x	x	15	12	11	6	4
22	sqr6	6	12	31	22	20	17	16	15	12	11	x	22	20	18	16	15	12	11
23	squar5	5	8	13	10	9	9	9	8	8	8	16	11	9	9	9	8	8	8
24	table3	14	14	x	x	142	127	107	93	65	47	x	x	x	x	121	103	69	49
25	vg2	25	8	x	67	53	36	30	26	16	14	x	x	x	x	x	x	x	x
26	x2	10	7	14	11	10	7	7	7	7	7	x	x	x	x	7	7	7	7
27	xor5	5	1	8	4	4	4	4	2	2	1	8	4	4	4	4	2	2	1
28	z4ml	7	4	25	20	19	13	10	9	8	5	x	x	x	x	x	x	x	5
29	z5xp1	7	10	34	27	22	19	17	16	12	12	x	x	x	20	18	17	12	12
30	z9sym	9	1	x	107	82	62	47	42	27	19	x	109	85	60	48	42	27	19
	Sumaryczna liczba bloków (bez wyróżnionych pól)			67	446	407	719	768	674	489	415	70	475	434	769	811	718	511	422

Z powyższych rozważań wynika, że wykorzystując wyjściowe bufory trójstanowe uzyskuje się układy pracujące około dwa razy szybciej niż układy uzyskiwane metodą klasyczną. Niestety, okupione jest to wykorzystaniem większej (około 50%) liczby bloków logicznych typu PAL. Średni czas syntezy analizowanych układów testowych metodą B_JPW jest około pięć razy dłuższy od czasu syntezy metodą klasyczną z minimalizacją liczby warstw logicznych (metoda K_LW). Najwięcej czasu zajmuje procedura rozłączania implikantów. Maksymalny, sumaryczny czas syntezy, bez uwzględnienia czasu minimalizacji (Espresso-Dso), dla układu testowego *ex1010* wynosi około 9s (Intel Celeron 433MHz).

W celu podkreślenia ważności procedury rozłączania implikantów przedstawiono porównanie zaproponowanej metody B_JW z metodą B_JW bez procedury rozłączania implikantów. Wyniki syntezy zawarto w tab. 3.5. Metoda syntezy bez procedury rozłączania implikantów znacznie częściej nie znajduje rozwiązań, natomiast znajdując je, najczęściej wykorzystuje większą liczbę bloków logicznych od metody zawierającej procedurę rozłączania.

4. METODY REALIZACJI FUNKCJI W STRUKTURACH TYPU PAL BAZUJĄCE NA DEKOMPOZYCJI

W początkowym okresie dekompozycja była wykorzystywana do optymalizacji sieci logicznej, realizowanej w postaci bramek logicznych [Ashe57,Curt62,Piec72a,Piec72b, Piec74,ShenM71]. Ogromne zainteresowanie dekompozycją obserwuje się od momentu wprowadzenia na rynek struktur FPGA typu tablicowego. Specyfika tych struktur wymusiła poszukiwanie efektywnych metod dekompozycji, stanowiących teoretyczne podstawy podziału projektowanego układu na części realizowalne w poszczególnych konfigurowalnych blokach logicznych (*ang.* Configurable Logic Block), znajdujących się wewnątrz struktury programowalnej [Xili99]. Poszukiwania szły w wielu różnorodnych kierunkach: adaptacji istniejących algorytmów przeznaczonych dla struktur bramkowych, poszukiwania „wygodnych" form reprezentacji funkcji, wykorzystania klasycznego modelu dekompozycji zaproponowanego przez Ashenhursta i Curtisa itp. W literaturze można znaleźć wiele różnorodnych metod dekompozycji, wykorzystywanych w procesie syntezy, przeznaczonej nie tylko dla układów FPGA typu tablicowego lub multiplekserowego [CiesY92, DevaW88a, KrishT02, MaliH91, PerkM97, RzecJ99, SchaP93, SongP96, SteiL99, SchaP93, Yan01a, Yan01b].

4.1. Znaczenie dekompozycji w syntezie przeznaczonej dla układów programowalnych

Jednym z podstawowych problemów syntezy układów cyfrowych implementowanych w strukturach programowalnych stał się sposób podziału projektu na odpowiednie części, które realizowane są za pomocą odrębnych układów PLD lub bloków logicznych, występujących w strukturach CPLD lub FPGA [Fawc95, Łuba99, RawsJ01, Shar98, Kani00h].

Proste programowalne układy logiczne należą do najstarszych układów PLD. Zawierają one niewielką liczbę elementów logicznych, lecz nie wprowadzają żadnych ograniczeń, związanych z połączeniami wewnątrz struktury. Ze względu na niewielką złożoność tych struktur często tworzy się sieci prostych układów PLD. Głównym problemem syntezy w tym przypadku jest dobór odpowiednich układów PAL lub PLA, dopasowanie projektowanego układu do struktury układu programowalnego oraz problem podziału projektu na bloki, realizowane w poszczególnych układach [AndeB98, BabbC92, PuchH94, Puch96a, Puch96b, Puch00, Sauc90a, Kani95, Kani99d].

W przypadku rozbudowanych struktur matrycowych (CPLD) najczęściej cały projekt realizowany jest w jednym układzie. W tym przypadku podstawowym problemem syntezy jest podział projektu na poszczególne bloki logiczne, zawarte wewnątrz struktury CPLD. Bloki logiczne zawarte w strukturach CPLD najczęściej wykorzystują strukturę typu PAL, stąd równie ważnym problemem syntezy staje się problem efektywnego wykorzystywania iloczynów.

Odmienną architekturę wewnętrzną mają układy FPGA typu tablicowego (*ang.* Look-up Table). Składają się one z matrycy konfigurowalnych bloków logicznych otoczonych na obrzeżach blokami wejścia/wyjścia. Ponieważ zasoby logiczne konfigurowalnych bloków logicznych umożliwiają realizację dowolnej funkcji o określonej liczbie argumentów, stąd głównym problemem syntezy jest dekompozycja, stanowiąca teoretyczne podstawy podziału całego projektu na poszczególne części o zadanej liczbie wejść i wyjść.

Z przedstawionych rozważań wynika, że proces projektowania układów cyfrowych wykorzystujących struktury programowalne wiąże się bezpośrednio z podziałem całego układu na fragmenty realizowane w odrębnych strukturach lub blokach logicznych, zawartych wewnątrz struktury programowalnej. Podział, będący strategicznym etapem całej nowoczesnej syntezy logicznej, znajduje swoje podstawy w dekompozycji funkcji logicznych.

Głównym elementem wielu systemów (przede wszystkim uniwersyteckich) wspomagających projektowanie układów cyfrowych w strukturach programowalnych jest dekompozycja, najczęściej ukierunkowana na wielopoziomową realizację układów w strukturach bramkowych lub strukturach typu tablicowego. Pierwsze algorytmy syntezy (MIS-PGA [BrayR87, BrowF93], Chortle [FrancR91, BrowF93], Asyl [AbouB93, BabbC92, SaucS90b, SicaC91]) przeznaczone dla struktur bramkowych (*ang.* gate array) zostały również wykorzystane w procesie syntezy, przeznaczonej dla układów FPGA typu tablicowego. Algorytmy te wykorzystują między innymi dekompozycję Roth-Karpa [RothK62], technikę wyłączania tzw. rdzeni (*ang.* kernel extraction) [BrayM82, Mich94, MurgM90, SaucS90b, SentS92], faktoryzację wyrażeń boolowskich [AbouS90b]. Wspomagane są one różnorodnymi procedurami grupowania, leksykograficznego porządkowania zmiennych [AbouB93], programowaniem dynamicznym itp. Bardzo często poszczególne kroki dekompozycji polegają na iteracyjnym podziale sieci logicznej, odpowiadającej wyrażeniu booleowskiemu, powstającemu w wyniku dwupoziomowej minimalizacji. Podział sieci logicznej związany jest z odpowiednim wyborem węzłów, gałęzi itd. [BrouF93, Mich94, Murg93]. W kilku kolejnych latach powstało szereg rozwinięć pierwotnych koncepcji, przeznaczonych przede wszystkim dla struktur FPGA typu tablicowego lub multiplekserowego. Należą do nich między innymi następujące algorytmy syntezy: Xmap [Karp91a], Amap [Karp91b], HYDRA [FiloY91], VISMAP [Woo91a], mis-fpga(*new*) [MurgS91], SIS [SentS92], KGPMAP [ChatR94].

Równolegle powstawały i rozwijały się metody syntezy, których podstawą jest klasyczny model dekompozycji funkcjonalnej, zaproponowany przez Ashenhursta [Ashe57], a rozszerzony przez Curtisa [Curt62, Curt63]. Model ten stanowi fundament szeregu algorytmów dekompozycji [BurnP97, BrzoŁ97, Kani99g, Kani00b, LeglW95, Łuba94, PanP96, Volf97, WanP92]. Algorytmy te wykorzystują często różnorodne strategie kolorowania grafów [Kani00a, MalvP98, PerkM99, SelvŁ97, WanP92], transformacji funkcji [WanP92], łączenia klasycznej dekompozycji z procedurami podziału wyjść [Kani00b, NowiŁ97] itd. Wydaje się, że jedną z zalet tych metod syntezy jest uwzględnianie stanów nieokreśloności w początkowym etapie syntezy, najczęściej związanym z dekompozycją.

W ostatnich latach dużym zainteresowaniem cieszą się metody syntezy wykorzystujące binarne diagramy decyzyjne (*ang.* Binary Decision Diagram) [Aker78]. W literaturze pojawiło się szereg prac dotyczących różnorodnych odmian binarnych diagramów decyzyjnych. Związane są one przede wszystkim z efektywnością reprezentacji funkcji, porządkowaniem zmiennych, faktoryzacją wyrażeń booleowskich oraz elementami dekompozycji, bazującymi na odpowiednich „cięciach" różnych form binarnych diagramów decyzyjnych [ChanM96, Drec99, Dzik99, Dzik03, IgucS00, IgucS03, JacoT92, Kapo95, MailM93, MinaI01, LaiP94a, LaiP94b, LaiP96, LiuW01, Sasa93b, SaraH93, StanS94, ThorD01, YangS99].

Ciągły rozwój różnorodnych modeli dekompozycji prowadzi do powstawania coraz to lepszych algorytmów syntezy, przedstawionych między innymi w pracach: [BurnP98, ChanM96, CongH01, CongD94, CongW98, JózwC99, Koło03, LaiP96, LaiP94, RawsJ99a, RawsJ99b, Sasa93b, SawaJ98, Scho98, YangC02]. Mimo burzliwego i wielotorowego rozwoju tematyki związanej z dekompozycją optymalny sposób podziału układu cyfrowego na bloki logiczne zawarte w strukturach programowalnych na obecnym poziomie wiedzy ciągle nie jest jednak znany.

Z poprzednich rozważań wynika, że dekompozycja odgrywa niezwykle istotną rolę w nowoczesnej syntezie logicznej. Jest ona przede wszystkim wykorzystywana, dając zarazem najlepsze wyniki, w syntezie przeznaczonej dla struktur programowalnych typu tablicowego. Znacznie rzadziej jest wykorzystywana w syntezie logicznej, przeznaczonej dla innych typów układów programowalnych. Znane są algorytmy syntezy wykorzystujące dekompozycję, przeznaczone dla matryc PLA [CiesY92, DevaW88a, DevaW88b, LeeW98, MaliH91, Sasa89b, WangA02, YangC89]. W niektórych metodach syntezy bezpośrednio zaadaptowano algorytmy dekompozycji, opracowane dla układów FPGA typu tablicowego [AndeB98, ChenH02]. Cechą charakterystyczną tego typu algorytmów jest występujący w nich proces odpowiedniego kodowania wyjść oraz wejść, znacząco wpływający na minimalizację liczby iloczynów wykorzystywanych w blokach uzyskiwanych w wyniku dekompozycji [DevaN91, SaldK88, Sasa01, YangC91]. Problemy odpowiedniego kodowania wejść i wyjść są szeroko dyskutowane w powiązaniu z zagadnieniami kodowania stanów wewnętrznych automatów sekwencyjnych [AshaD92, CzerwK02, DeniS94, MitrA99, Mich94, ShiB93]. Między innymi związane są one bezpośrednio z symbolicznym kodowaniem stanów wewnętrznych, teorią dychotomii, minimalizacją funkcji wielowartościowych, analizą relacji dominacji wyjść itp. [AshaD92, Mich94, Łuba00]. Znane są również prace bezpośrednio wiążące proces kodowania wejść i wyjść z problemami dekompozycji [PerkB96].

Widząc duże znaczenie dekompozycji w nowoczesnej syntezie logicznej nasuwa się pytanie: *Czy nie można wykorzystać jej w procesie syntezy, przeznaczonej dla struktur programowalnych typu PAL?*

Przeprowadzone wstępne eksperymenty z narzędziami opracowanymi dla układów FPGA typu tablicowego (Decomp) [Kani96, Kani00b], wykorzystanymi do syntezy logicznej dla układów typu PAL, były niezwykle zachęcające [Kani99g, Kani00a, Kani00f]. Tym bardziej, że uzyskując stosunkowo dobre rezultaty, widać było możliwość optymalizacji wielu elementów pod kątem docelowej struktury. Można było wykorzystać tzw. kodowanie równomierne [Kani98b, Kani99f, Kani00d, Kani01b] oraz doświadczenia związane z podziałem projektowanego układu na poszczególne proste układy programowalne [Kani95, Kani98b].

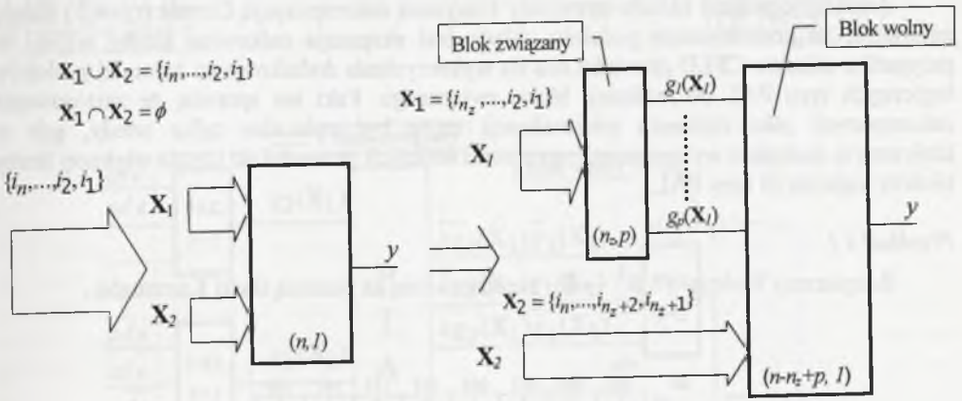
4.2. Istota dekompozycji przeznaczonej dla struktur typu PAL

Funkcja $y = f(i_n,\ldots,i_2,i_1) = f(\mathbf{X}_2,\mathbf{X}_1)$ podlega dekompozycji prostej, tzn. $f(\mathbf{X}_2,\mathbf{X}_1)=F[g(\mathbf{X}_1),\mathbf{X}_2]$ wtedy i tylko wtedy, gdy złożoność[1)] kolumnowa matrycy podziałów $\nu(\mathbf{X}_2|\mathbf{X}_1)\leq 2$ [Ashe57, Curt62]. Zbiory $\mathbf{X}_1$ i $\mathbf{X}_2$ nazywane są odpowiednio zbiorem związanym i wolnym, przy czym $\mathbf{X}_1\cup\mathbf{X}_2=\{i_n,\ldots,i_2,i_1\}$ oraz $\mathbf{X}_1\cap\mathbf{X}_2=\phi$.

Twierdzenie o dekompozycji prostej zostało rozszerzone [Curt63], stanowiąc podstawę dekompozycji funkcjonalnej

$$\nu(\mathbf{X}_2|\mathbf{X}_1)\leq 2^p \Leftrightarrow f(\mathbf{X}_2,\mathbf{X}_1)=F[g_1(\mathbf{X}_1),g_2(\mathbf{X}_1),\ldots,g_p(\mathbf{X}_1),\mathbf{X}_2] \tag{4.1}$$

Wykorzystanie takiego modelu dekompozycji implikuje podział bloku scharakteryzowanego uporządkowaną parą liczb (liczba wejść, liczba wyjść) wynoszącą (n,l), na dwa inne o parametrach (n_z, p) i $(n-n_z+p, 1)$. Podział taki przedstawiony jest na rys. 4.1.



Rys. 4.1. Rozłączna dekompozycja Curtisa

Fig. 4.1. Disjoint Curtis decomposition

Zwykle istota dekompozycji funkcji sprowadza się do odpowiedniego podziału projektowanego układu logicznego na podukłady o zadanej liczbie wejść i wyjść (realizacja układu cyfrowego w strukturach FPGA typu tablicowego zawierającego konfigurowalne bloki logiczne). Dekompozycja może być również wykorzystywana w innych przypadkach, np. do podziału projektu na części, prowadząc do realizacji całości w postaci sieci prostych układów PLD.

Bloki logiczne typu PAL zawarte w strukturach CPLD mają znacznie większą liczbę wejść niż konfigurowalne bloki logiczne zawarte w strukturach FPGA lub proste układy PLD. Zwykle do każdego iloczynu wchodzącego w skład bloku logicznego typu PAL można dołączyć sygnał z dowolnego wejścia lub sprzężenia zwrotnego, co ogranicza znaczenie

[1] W literaturze występuje również pojęcie krotności kolumnowej.

dekompozycji jako narzędzia stanowiącego podstawę podziału projektu na poszczególne bloki logiczne. Dodatkowo projektując układ cyfrowy implementowany w strukturach programowalnych dąży się zawsze do odpowiedniego doboru struktury, aby zapewnić realizowalność całego projektu w jednym układzie. Szeroka oferta struktur CPLD w większości przypadków umożliwia dobór jednej struktury do konkretnego zadania. Powoduje to dalsze ograniczenie znaczenia dekompozycji jako podstawy podziału projektu na poszczególne układy.

Głównym ograniczeniem bloków logicznych typu PAL jest liczba wielowejściowych iloczynów zawartych w jednym bloku. Fakt ten sprawia, że istota dekompozycji przeznaczonej dla struktur typu PAL sprowadza się do dwóch głównych zadań:

1. Dekompozycja powinna stanowić element minimalizacji liczby wykorzystywanych iloczynów, pośrednio prowadząc do minimalizacji liczby użytych bloków logicznych typu PAL;
2. Dekompozycja powinna stanowić element dopasowania projektowanego układu do struktury bloków logicznych typu PAL.

Analizując podział układu otrzymany klasyczną dekompozycją Curtisa (rys.4.1) należy zauważyć, że konsekwencją podziału układu jest ekspansja całkowitej liczby wyjść. W przypadku układów CPLD prowadzi ona do wykorzystania dodatkowo co najmniej *p*-bloków logicznych typu PAL do realizacji bloku związanego. Fakt ten sprawia, że zastosowanie dekompozycji jako elementu minimalizacji może być opłacalne tylko wtedy, gdy w klasycznym podejściu wykorzystanie sprzężeń zwrotnych prowadzi do użycia większej liczby bloków logicznych typu PAL.

Przykład 4.1

Rozpatrzmy funkcję $f: B^5 \to B$ przedstawioną za pomocą siatki Karnaugha .

ab \ cde	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

$y=f(a,b,c,d,e)$

Klasyczna realizacja funkcji $y=f(a,b,c,d,e)$ wymaga użycia 11 iloczynów. Wykorzystując 3-iloczynowe bloki typu PAL uzyskujemy strukturę 3-warstwową składającą się z 5 bloków ($\delta_f = \left\lceil \frac{\Delta_f - k}{k-1} \right\rceil + 1 = 5$).

Przedstawiona siatka Karnaugha zawiera cztery rodzaje kolumn ($\nu(\mathbf{X}_2|\mathbf{X}_1)=4$), stąd korzystając z twierdzenia Curtisa (zależność (4.1)) uzyskujemy realizację funkcji wykorzystującą mniejszą liczbę 3-iloczynowych bloków typu PAL (rys. 4.2).

ab \ cde	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
	A	B	A	C	A	D	A	B

$\mathbf{X}_2=\{a,b\};\ \mathbf{X}_1=\{c,d,e\}$

$\nu(\mathbf{X}_2|\mathbf{X}_1)=4 \Rightarrow f(\mathbf{X}_2,\mathbf{X}_1)=F[g_1(\mathbf{X}_1),g_2(\mathbf{X}_1),\mathbf{X}_2]$

Blok wolny

ab \ cde	101	100	010	111
00	1	0	1	0
01	1	0	0	1
11	0	1	0	1
10	1	0	1	0
	A	B	C	D

Blok związany

c \ de	00	01	11	10
0	A	B	A	C
1	B	A	D	A

Kodowanie
A - 00
B - 01
C - 11
D - 10

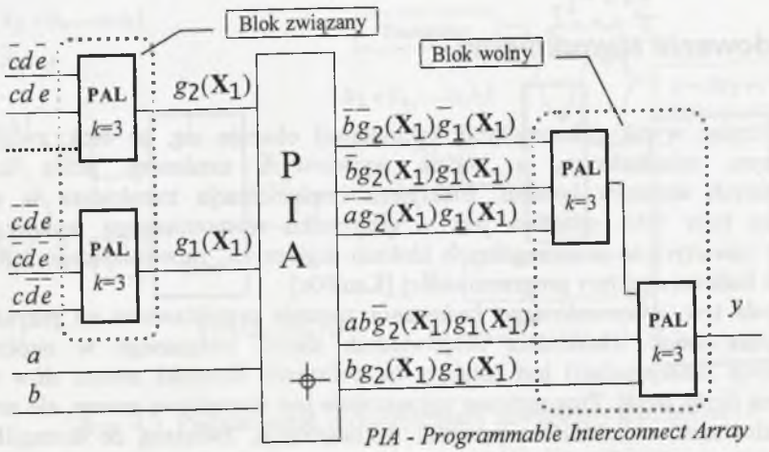
⇒

Blok wolny

ab \ $g_2(\mathbf{X}_1)\,g_1(\mathbf{X}_1)$	00	01	11	10
00	1	0	1	0
01	1	0	0	1
11	0	1	0	1
10	1	0	1	0
	A	B	C	D

Blok związany

c \ de	00	01	11	10
0	00	01	00	11
1	01	00	10	00
	$g_2(\mathbf{X}_1)\,g_1(\mathbf{X}_1)$			



Rys.4.2. Realizacja funkcji bazująca na dekompozycji funkcjonalnej

Fig. 4.2. Implementation of the function based on functional decomposition

Istotny wpływ na efektywność wykorzystania bloków logicznych typu PAL, o zadanej liczbie iloczynów, do realizacji bloku związanego i wolnego, ma sposób kodowania wzorców kolumn. Dekompozycja Curtisa, nazwijmy ją ***dekompozycją kolumnową***, nie daje bezpośrednio odpowiedzi na pytanie, ile bloków logicznych typu PAL o zadanej liczbie iloczynów jest potrzebnych do realizacji bloku związanego i wolnego. Dopiero po kodowaniu

wzorców kolumn i minimalizacji otrzymujemy odpowiedź na to pytanie. Wobec powyższego dekompozycja kolumnowa wraz z elementami kodowania wzorców kolumn może stanowić tylko element istotnie wpływający na minimalizację liczby wykorzystywanych iloczynów, pośrednio wpływając na minimalizację liczby użytych bloków logicznych typu PAL. Korzyści wynikające z minimalizacji liczby iloczynów mogą być tracone ze względu na złe dopasowanie uzyskiwanych po podziale podukładów do struktury bloku logicznego.

Złożoność algorytmów poszukiwania dekompozycji, przede wszystkim liczba podziałów argumentów funkcji, które trzeba rozpatrzyć, powoduje, że trudno wyobrazić sobie sytuację, w której sprawdzane byłyby wszystkie rozwiązania i wybierane to, które wymaga użycia minimalnej liczby bloków logicznych.

Okazuje się, że można w etap dekompozycji wpleść element dopasowania uzyskiwanych podukładów do struktury bloku logicznego. Efekt dopasowania bloku wolnego do struktury bloku logicznego uzyskuje się w tzw. ***dekompozycji wierszowej*** (rozdział 4.5).

Podsumowując powyższe rozważania należy przypuszczać, że dekompozycja może stanowić niezwykle cenny element syntezy, związany z minimalizacją sumarycznej liczby wykorzystywanych bloków (w stosunku do metody klasycznej) lub dopasowaniem realizowanych funkcji do struktury bloków logicznych znajdujących się wewnątrz układu. Prowadzi to ostatecznie do efektywnej realizacji funkcji w strukturach typu PAL. Takie nietypowe zastosowanie teorii dekompozycji jest przedmiotem rozważań stanowiących główny wątek dalszej części niniejszego rozdziału.

4.3. Kodowanie równomierne

Analizując wynik dekompozycji kolumnowej okazuje się, że blok związany jest specyficznym transkoderem, o kodzie wyjściowym ustalonym przez kodowanie poszczególnych wzorców kolumn. Efektywna implementacja transkodera w strukturze matrycowej typu PAL możliwa jest w przypadku równomiernego wykorzystywania iloczynów zawartych w poszczególnych blokach logicznych, zapewniającego maksymalną pojemność kodową struktury programowalnej [Kani00e].

Metoda tzw. ***równomiernego kodowania*** zostanie przedstawiona na przykładzie, w którym zbiór wejść transkodera (odpowiednik zbioru związanego w ogólnej teorii dekompozycji funkcjonalnej) jest ustalany na podstawie liczności zbioru słów układu z ograniczoną liczbą wejść. Proponowane uproszczenie jest stosunkowo mocne, ale ze względu na dużą złożoność obliczeniową procedur dekompozycji, związaną ze skomplikowanym zadaniem wyznaczenia zbioru wejść bloku związanego, w pewnych sytuacjach (np. realizacja dekoderów adresu) może być z powodzeniem stosowane. Wyszukiwanie zbioru wejść transkodera związane jest z problemem projektowania układów modyfikacji adresu [Łuba00, MajeŁ96, Mols86, Trac86].

W celu przedstawienia sposobu wyszukiwania zmiennych tworzących zbiór sygnałów wejściowych transkodera wprowadźmy pojęcie współczynnika komplikacji.

Definicja 4.1

Współczynnikiem komplikacji $w_k(i_{S-1},..., i_{r+1}, i_r)$ nazywamy liczbę różnych słów, utworzonych przez zmienne $i_{S-1},..., i_{r+1}, i_r$.

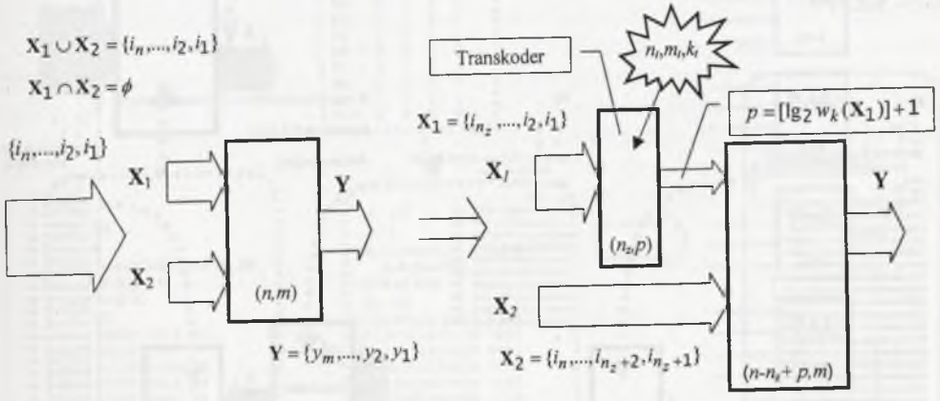
Przykład 4.2

Rozważmy funkcję $f: B^5 \to B$ opisaną w pliku *f.pla*, dla której wyznaczone są wybrane współczynniki komplikacji.

```
.i 5
.o 1
.ilb a b c d e
.ob f
.p 5
10000 1
10011 1
00010 1
01000 1
01011 1
.e
```

$w_k(a,b,c) = 3$
$w_k(b,c,d) = 4$
$w_k(a,d,e) = 5$

Okazuje się, że jeżeli istnieje zbiór $\mathbf{X}_1$, taki że $w_k(\mathbf{X}_1) < 2^{(\overline{\mathbf{X}_1}-1)}$, to można ograniczyć liczbę wejść podukładu realizującego wielowyjściową funkcję $f: B^n \to B^m$ poprzez wykorzystanie zewnętrznego transkodera. Idea podziału wejść przedstawiona jest na rys. 4.3.



Rys. 4.3. Podział wejść wykorzystujący zewnętrzny transkoder

Fig. 4.3. Input partitioning using external transcoder

Programowalny transkoder jest scharakteryzowany przez trzy parametry:
n_t - liczbę wejść, m_t - liczbę wyjść oraz współczynnik k_t równy liczbie możliwych do utworzenia wyjściowych słów kodowych. Współczynnik k_t bezpośrednio zależy od struktury układu programowalnego oraz sposobu kodowania.

Poszukiwania zmiennych tworzących zbiór $\mathbf{X}_1$ są ograniczane do przypadków, dla których $\overline{\mathbf{X}_1} \le n_t$. Zwykle najbardziej interesujące są przypadki, dla których wyrażenie $\overline{\mathbf{X}_1} - ([\lg_2(w_k(\mathbf{X}_1))]+1)$ przyjmuje wartość maksymalną. W przypadku gdy istnieje kilka

podziałów argumentów funkcji umożliwiających wykorzystanie transkodera o zadanej liczbie wejść, wybierane jest rozwiązanie, dla którego $\overline{\overline{\mathbf{X}2} + ([\lg_2(w_k(\mathbf{X}_1))]+1)} = \min$.

Dokładny opis algorytmu podziału wejść za pomocą zewnętrznego transkodera jest zawarty w pracach [Kani95,HrynP97].

Przykład 4.3

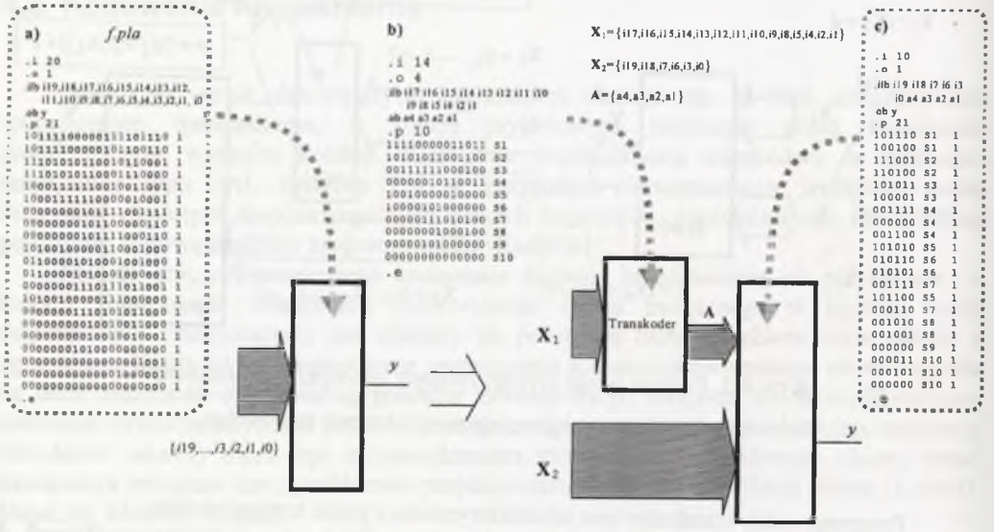
Spróbujmy poszukać podziału argumentów funkcji $f : B^{20} \to B$ (rys.4.4.a), zakładając wykorzystanie transkodera o liczbie wejść n_t=14.

Ze względu na założoną liczbę wejść poszukiwania rozpoczynamy od podzbiorów zmiennych wejściowych takich, dla których $\overline{\overline{\mathbf{X}_1}}$=14.

Okazuje się, że istnieje zbiór $\mathbf{X}_1$={*i*17,*i*16,*i*15,*i*14,*i*13,*i*12,*i*11,*i*10,*i*9,*i*8,*i*5,*i*4,*i*2,*i*1} spełniający wszystkie warunki, prowadzący do rozwiązania przedstawionego na rys. 4.4.

Opis układu pierwotnego zawarty jest w postaci pliku *f.pla* (rys.4.4a). Pogrubione znaki (0 i 1) w pliku *f.pla*, odpowiadające zmiennym należącym do zbioru $\mathbf{X}_1$, tworzą $w_k(\mathbf{X}_1)$=10 różnych wzorców słów.

Symboliczny opis układów powstających po podziale zawiera zamiast 4-bitowych kodów symbole S1-S10 skojarzone z występującymi w plikach różnymi wzorcami słów (rys. 4.4 b;c).



Rys. 4.4. Realizacja funkcji $f : B^{20} \to B$ wykorzystująca transkoder

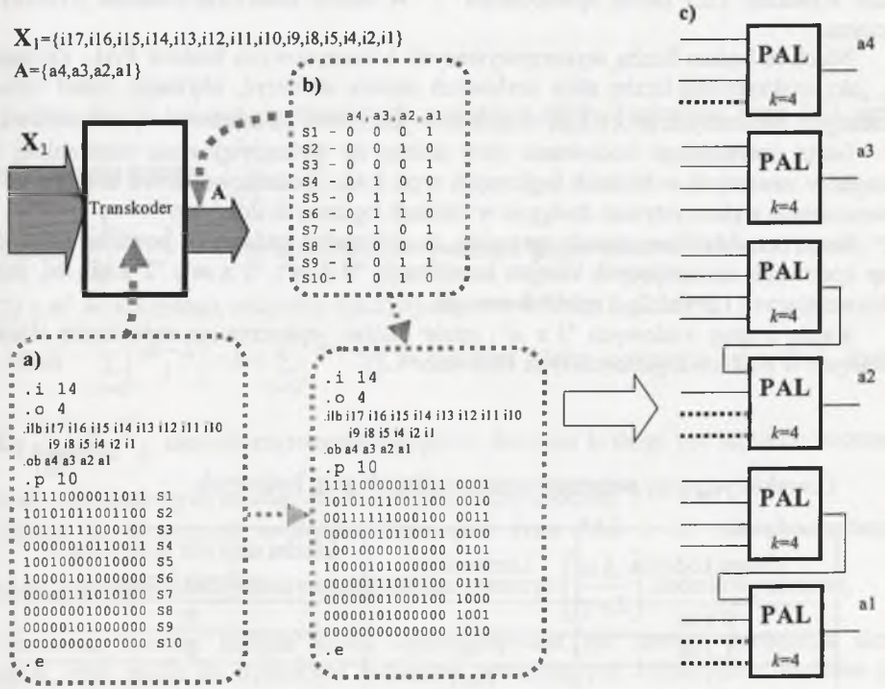
Fig. 4.4. Implementation of the function using transcoder

Specyfika struktury typu PAL, tzn. ściśle określona liczba iloczynów zawartych w blokach logicznych typu PAL sprawia, że niezwykłej wagi nabiera sposób kodowania wyjść, bezpośrednio wpływający na liczbę wykorzystywanych bloków logicznych.

Przykład 4.4

Spróbujmy zrealizować transkoder z przykładu 4.3 (rys.4.5a) wykorzystując 4-iloczynowe bloki logiczne typu PAL.

Rozróżniając wzorce słów za pomocą kodów binarnych (rys.4.5b), ze względu na nierównomierne wykorzystanie iloczynów można zakodować (bez sprzężeń zwrotnych) tylko 2*k*+1=9 różnych słów. Ekspansja liczby iloczynów poprzez zastosowanie sprzężeń zwrotnych (rys.4.5c) prowadzi do zwiększenia liczby słów kodowych kosztem użycia dodatkowych bloków logicznych typu PAL.



Rys. 4.5. Realizacja transkodera za pomocą bloków logicznych typu PAL zawierających cztery iloczyny (linią przerywaną oznaczono niewykorzystane iloczyny)

Fig. 4.5. Implementation of a transcoder by means of PAL-based logic blocks consisting of four terms (unused terms marked with the dashed line)

Czy można lepiej zakodować wzorce słów?

Wadą zaprezentowanego w przykładzie 4.4 kodowania binarnego jest nierównomierne wykorzystywanie iloczynów. Znacznie lepszym rozwiązaniem byłoby stosowanie innego kodowania, które równomiernie wykorzystywałoby iloczyny zawarte w blokach logicznych typu PAL.

Jak kodować równomiernie wykorzystując iloczyny?

Ile różnych słów można utworzyć wykorzystując m-bloków logicznych typu PAL zawierających k-iloczynów?

Odpowiedź na drugie pytanie wiąże się bezpośrednio z wartością parametru k_t (rys.4.3). W celu uproszczenia rozważań załóżmy, że przedstawiona w dalszej części analiza przeprowadzona zostanie tylko dla bloków logicznych typu PAL z wyjściami aktywnymi w stanie wysokim. Przy takim uproszczeniu "1" w słowie kodowym oznacza wykorzystanie iloczynu.

Niech m będzie liczbą wykorzystywanych k-iloczynowych bloków PAL. Zastanówmy się, jaką maksymalną liczbę słów kodowych można utworzyć, używając układ typu PAL o zadanych parametrach (m,k). Liczba tych słów jest równa parametrowi k_t transkodera.

Istota optymalnego kodowania słów polega na wykorzystywaniu minimalnej liczby iloczynów zawartych w blokach logicznych typu PAL. Dodatkowo słowa kodowe powinny równomiernie wykorzystywać dostępne w blokach logicznych iloczyny.

Dwie przedstawione zasady sprawiają, że optymalne kodowanie powinno składać się z grup kodowych zawierających kolejno kombinacje "0 z m"; "1 z m"; "2 z m" itd. jedynek, uaktywniających i (i=0,1,2...) spośród m-wyjść.

Każda z grup kodowych "i z m", gdzie $0 \le i \le m$, wykorzystuje stałą liczbę iloczynów zawartych w blokach logicznych typu PAL (tab. 4.1).

Tabela 4.1

Charakterystyczne parametry poszczególnych grup kodowych

Grupa kodowa	Liczba słów	Liczba użytych iloczynów w jednym bloku
0 z m	1	0
1 z m	m	1
2 z m	$\binom{m}{2}$	m-1
3 z m	$\binom{m}{3}$	$\binom{m-1}{2}$
.	.	.
.	.	.
m z m	$\binom{m}{m}$	$\binom{m-1}{m-1}$

Twierdzenie 4.1

Wykorzystując m-bloków logicznych typu PAL zawierających k-iloczynów można utworzyć

$$\sum_{i=0}^{j}\binom{m}{i}+\left[\frac{m\,k_r}{j+1}\right] \tag{4.2}$$

różnych słów kodowych, gdzie j jest liczbą spełniającą nierówność

$$\sum_{i=0}^{j-1}\binom{m-1}{i} \le k < \sum_{i=0}^{j}\binom{m-1}{i}, \tag{4.3}$$

natomiast $$k_r = k - \sum_{i=0}^{j-1}\binom{m-1}{i}$$

Dowód:

Liczba iloczynów wykorzystywanych w każdym bloku logicznym typu PAL przez j-grup kodowych wynosi $\sum_{i=0}^{j-1}\binom{m-1}{i}$.

Jeżeli $\sum_{i=0}^{j-1}\binom{m-1}{i} = k$, to słowa kodowe tworzące grupy kodowe "0 z m"; "1 z m"; "2 z m";...;"j z m" wykorzystują wszystkie iloczyny zawarte w bloku logicznym.

Jeżeli $\sum_{i=0}^{j-1}\binom{m-1}{i} < k < \sum_{i=0}^{j}\binom{m-1}{i}$, w każdym bloku logicznym typu PAL zostaje $k_r = k - \sum_{i=0}^{j-1}\binom{m-1}{i}$ niewykorzystanych iloczynów. Iloczyny te mogą być użyte do tworzenia kolejnych słów kodowych należących do kolejnej grupy kodowej "j+1 z m".

Mając do dyspozycji m-bloków logicznych typu PAL i k_r niewykorzystanych iloczynów w każdym bloku, maksymalnie można utworzyć $\left[\frac{m\,k_r}{j+1}\right]$ dodatkowych słów.

Ponieważ tworząc kolejne słowa wykorzystywana jest zawsze minimalna liczba iloczynów, stąd, mając do dyspozycji m-bloków zawierających k-iloczynów, możliwe jest utworzenie maksymalnie $\sum_{i=0}^{j}\binom{m}{i}+\left[\frac{m\,k_r}{j+1}\right]$ różnych słów.

cbdo.

Twierdzenie 4.1 wskazuje na zależność współczynnika k_t, charakteryzującego transkoder typu PAL, od parametrów m i k, tzn. $k_t = f(m,k)$. Wartości współczynnika k_t dla optymalnego kodowania uzależnione od parametrów m i k zawarte są w tab. 4.2. Dla porównania, dla kodowania binarnego bez wykorzystania sprzężeń zwrotnych można utworzyć tylko $2k$+1 słów kodowych.

Tabela 4.2

Wartości współczynnika $k_t = f(m,k)$

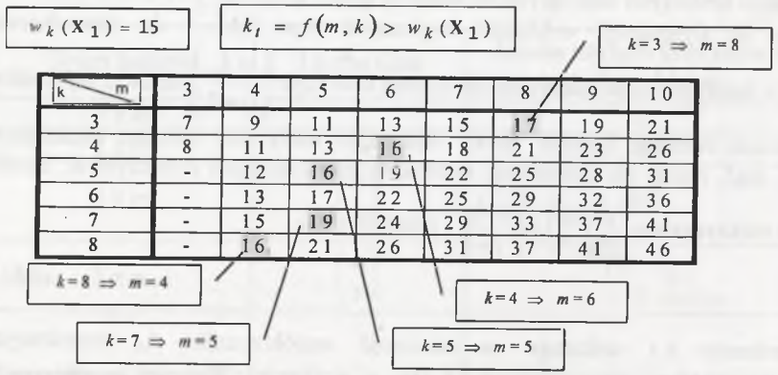
k \ m	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	7	9	11	13	15	17	19	21	23	25	27	29	31	33
4	8	11	13	16	18	21	23	26	28	31	33	36	38	41
5	-	12	16	19	22	25	28	31	34	37	40	43	46	49
6	-	13	17	22	25	29	32	36	39	43	46	50	53	57
7	-	15	19	24	29	33	37	41	45	49	53	57	61	65
8	-	16	21	26	31	37	41	46	50	55	59	64	68	73

Analizując wartości zawarte w tab. 4.2 wyciągnięto praktyczne wnioski. Jeżeli dla wybranego podzbioru argumentów $\mathbf{X}_1$ współczynnik komplikacji wynosi $w_k(\mathbf{X}_1)$, to znając liczbę iloczynów zawartych w bloku logicznym typu PAL (k) można wyznaczyć liczbę bloków potrzebnych do zakodowania $w_k(\mathbf{X}_1)$ wzorców słów. Liczba ta jest najmniejszą wartością m, dla której, przy założonej wartości k, zachodzi nierówność $w_k(\mathbf{X}_1) < k_t$ (musi zachodzić nierówność ostra, ponieważ słowo kodowe 0...00 zarezerwowane jest dla wszystkich pozostałych wzorców nie wchodzących w skład $w_k(\mathbf{X}_1)$-wzorców słów).

Przykład 4.5

Spróbujmy dobrać liczbę bloków typu PAL, w przypadku gdy $w_k(\mathbf{X}_1) = 15$. Ze względu na konieczność zarezerwowania jednego słowa kodowego dla pozostałych wzorców słów nie wchodzących do liczby $w_k(\mathbf{X}_1)$ współczynnik k_t musi być większy od współczynnika komplikacji, tzn.: $k_t > w_k(\mathbf{X}_1)$.

Dobór minimalnej liczby bloków potrzebnych do rozróżnienia $w_k(\mathbf{X}_1)$ wzorców słów, dla przykładowych wartości k, przedstawiony jest na rys. 4.6.



Rys. 4.6. Zasada doboru liczby bloków typu PAL uzależniona od $w_k(\mathbf{X}_1)$ oraz k

Fig. 4.6. The rule for selecting the number of PAL-based blocks depending on $w_k(\mathbf{X}_1)$ and k

Nazwijmy zaproponowaną metodę tworzenia słów kodowych ***kodowaniem równomiernym*** [Kani01b].

Najczęściej parametrami początkowymi doboru transkodera są wielkości $w_k(\mathbf{X}_1)$ oraz k. Pozwala to na wyznaczenie liczby bloków logicznych typu PAL (m) potrzebnych do zakodowania $w_k(\mathbf{X}_1)$ wzorców słów ($k_t = f(m,k) > w_k(\mathbf{X}_1)$). W kolejnym etapie możliwe jest utworzenie słów kodowych.

Jeżeli istnieje takie j, dla którego $\sum_{i=0}^{j-1}\binom{m-1}{i} = k$, to zasada kodowania jest stosunkowo prosta. Polega ona na wyborze dowolnego zbioru słów o mocy $w_k(\mathbf{X}_1)+1$, należących do pierwszych $(j+1)$–grup kodowych (kombinacja 0...00 przeznaczona jest dla wszystkich pozostałych słów nie wchodzących do liczby $w_k(\mathbf{X}_1)$).

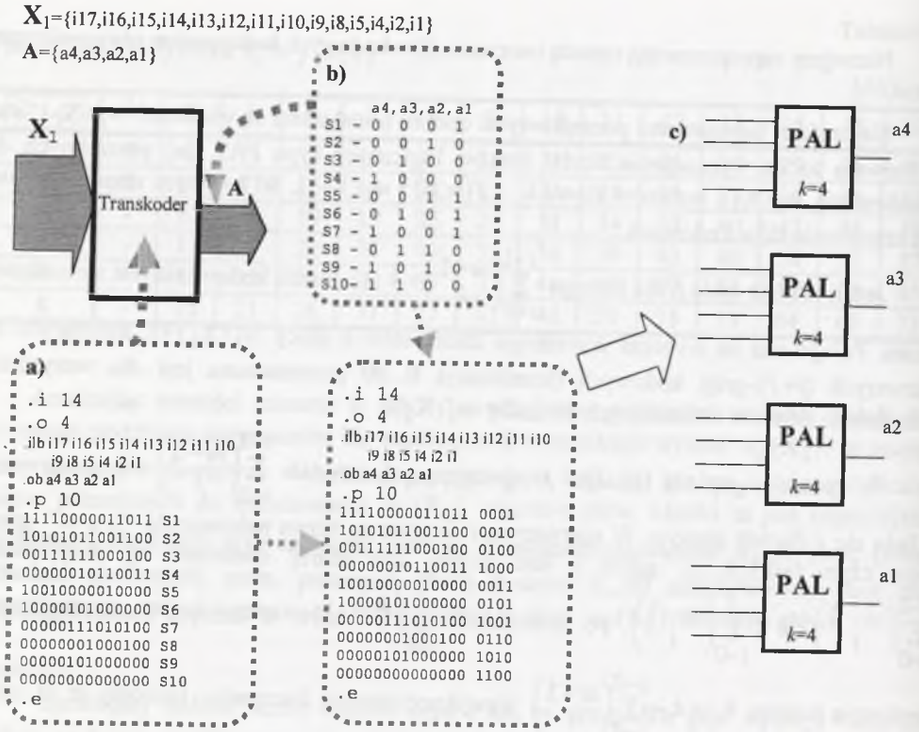
W sytuacji, gdy nie istnieje j spełniające zależność $\sum_{i=0}^{j-1}\binom{m-1}{i} = k$, kodowanie składa się z dwóch etapów. W pierwszym tworzone są słowa należące do "i z m" grup, przy czym i=0,1,2,...,j, gdzie j jest liczbą, dla której spełnione są nierówności $\sum_{i=0}^{j-1}\binom{m-1}{i} < k < \sum_{i=0}^{j}\binom{m-1}{i}$. Po utworzeniu $\sum_{i=0}^{j}\binom{m}{i}$ słów w każdym bloku logicznym występuje jeszcze $k_r = k - \sum_{i=0}^{j-1}\binom{m-1}{i}$ niewykorzystanych iloczynów. Iloczyny te są użyte do realizacji następnych słów, należących do grupy "j+1 z m".

Dokładny opis algorytmów kodowania równomiernego, opracowanych dla struktur typu PAL ze stałą liczbą iloczynów w bloku logicznym, można znaleźć w pracach [HrynP97, Kani98b]. Istnieją struktury programowalne, w których bloki logiczne typu PAL zawierają lub mogą zawierać różną liczbą iloczynów, np. 22V10, 26V12, układy rodziny MACH, Flash370 itp. [Adva95, Adva96, Cypr96]. Rozszerzone algorytmy kodowania równomiernego, opracowane dla tego typu układów programowalnych, zaprezentowane są w pracach [KaniH98, Kani99f].

Przykład 4.6

Wróćmy do transkodera z przykładu 4.3. Analiza przedstawiona w przykładzie 4.4 pokazała, że korzystając z kodowania binarnego do realizacji transkodera (rys.4.7a) należy użyć 6 bloków logicznych typu PAL, zawierających 4 iloczyny.

Zastosowanie kodowania równomiernego (rys.4.7b) prowadzi do realizacji transkodera z wykorzystaniem tylko czterech bloków (rys. 4.7c).



Rys. 4.7. Realizacja transkodera wykorzystująca bloki logiczne zawierające 4 iloczyny

Fig. 4.7. Implementation of the transcoder using PAL-based logic block with 4 terms

Zaproponowana metoda podziału argumentów, polegająca na sprawdzaniu liczności zbioru słów układu z ograniczoną liczbą wejść, mimo iż wprowadza stosunkowo mocne ograniczenia, dla pewnej klasy układów może być z powodzeniem stosowana. Zakres jej stosowania praktycznie ograniczony jest do syntezy układów ze stosunkowo dużą liczbą wejść, opisanych jednak za pomocą małej liczby implikantów, np. dekodery adresu itp.

Okazuje się jednak, że zaprezentowana na najprostszym modelu dekompozycji metoda kodowania równomiernego może być z powodzeniem stosowana do kodowania wyjść bloku związanego w klasycznej dekompozycji Curtisa, prowadząc w wielu przypadkach do znaczącej minimalizacji liczby wykorzystywanych iloczynów.

4.4. Dekompozycja kolumnowa zorientowana na realizację w strukturach typu PAL

Dekompozycja kolumnowa (Curtisa) zorientowana na realizację układów w strukturach typu PAL może istotnie wpływać na efektywność wykorzystania zasobów struktur programowalnych. Jednym z elementów, występujących w początkowym etapie poszukiwania odpowiedniej dekompozycji kolumnowej, jest sposób wyznaczenia złożoności kolumnowej matrycy podziałów (siatki Karnaugha). Złożoność kolumnowa siatki Karnaugha bezpośrednio wpływa na liczbę wyjść bloku związanego zgodnie z zależnością $p = \lceil \lg_2 \nu(\mathbf{X}_2|\mathbf{X}_1) \rceil$, gdzie p jest liczbą wyjść tego bloku (rys.4.1).

4.4.1. Metody wyznaczania złożoności kolumnowej zorientowane na realizację układów w strukturach typu PAL

Istnieje wiele różnorodnych metod wyznaczania złożoności kolumnowej, najczęściej opartych na analizie grafów zgodności lub niezgodności kolumn, przedstawionych między innymi w pracach: [MalvP98, PerkM98, RawsN96, SelvL97, WanP92, Kani00a]. Metody wykorzystujące grafy zgodności kolumn bazują na różnorodnych, najczęściej heurystycznych algorytmach wyznaczania klik, różnorodnych algorytmach związanych z problemem pokrycia wierzchołków itp. Komplementarne podejście oparte na analizie grafów niezgodności kolumn jest związane z problemem kolorowania wierzchołków grafu, prowadzącym do wyznaczenia liczby chromatycznej, równiej złożoności kolumnowej.

Parę komórek (i, j) należących do tego samego wiersza siatki Karnaugha, nazywamy niezgodną, jeżeli wartości funkcji w nich zawarte wynoszą (1,0) lub (0,1). Jeżeli w zbiorze par komórek należących do dwóch różnych kolumn wystąpi przynajmniej jedna para komórek niezgodnych, to takie kolumny nazywamy niezgodnymi. Grafem niezgodności kolumn nazywamy graf $G(\mathbf{Y},\mathbf{U})$, gdzie $\mathbf{Y}$ jest zbiorem wierzchołków odpowiadających kolumnom, natomiast $\mathbf{U}$-zbiorem krawędzi łączących wierzchołki odpowiadające kolumnom niezgodnym.

Przykład 4.7

Wyznaczmy graf niezgodności kolumn dla funkcji $f:B^5 \rightarrow B$ analizowanej w przykładzie 4.1. Siatka Karnaugha i odpowiadający jej graf niezgodności kolumn przedstawione są na rys. 4.8.

$\mathbf{X}_1=\{c,d,e\}$, $\mathbf{X}_2=\{a,b\}$

ab \ cde	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. 4.8. Siatka Karnaugha i graf niezgodności kolumn

Fig. 4.8. The Karnaugh map and the graph of column incompatibility

Kolorowanie grafu niezgodności kolumn pozwala na wyznaczenie złożoności kolumnowej $\nu(\mathbf{X}_2 | \mathbf{X}_1)$, która jest równa liczbie chromatycznej grafu $\gamma(G)$. Istota kolorowania polega na przypisaniu wierzchołkom grafu kolorów w taki sposób, aby każde dwa wierzchołki połączone krawędzią miały inny kolor. W literaturze można znaleźć wiele różnych dokładnych i heurystycznych algorytmów kolorowania wierzchołków grafu [Brya93, Chri75, MalvP98, PerkM98, WanP92, Mich94].

Dobór algorytmu kolorowania wierzchołków grafu jest niezwykle istotny ze względu na konieczność powtarzania tego procesu wielokrotnie dla różnych podziałów argumentów funkcji. Podstawą wykorzystywanej metody kolorowania jest szybki algorytm kolorowania przedstawiony w pracy [WanP92]. Przeprowadzone eksperymenty wskazały na możliwość jego uproszczenia, prowadzącą do przyśpieszenia procesu kolorowania wierzchołków bez istotnego pogorszenia efektywności dekompozycji kolumnowej, wykorzystywanej w procesie syntezy, przeznaczonej dla struktur typu PAL.

Algorytm kolorowania wierzchołków polega na sekwencyjnym wyborze wierzchołków. Wybranemu w *i*-tym kroku wierzchołkowi przypisywany jest tzw. kolor dozwolony, oznaczany dużą literą. W miarę możliwości wybierany jest kolor z grupy kolorów dozwolonych przyporządkowanych wierzchołkom w poprzednich krokach. Po przypisaniu w *i*-tym kroku dozwolonego koloru wybranemu wierzchołkowi, przypisywane są tzw. kolory zabronione (oznaczane małą literą) wszystkim wierzchołkom, połączonym krawędziami z wybranym wierzchołkiem.

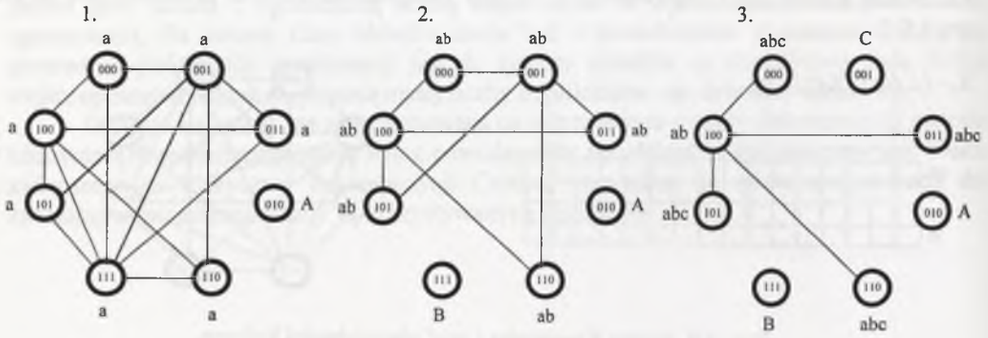
Wybór *i*-tego wierzchołka prowadzony jest zgodnie z następującymi zasadami:

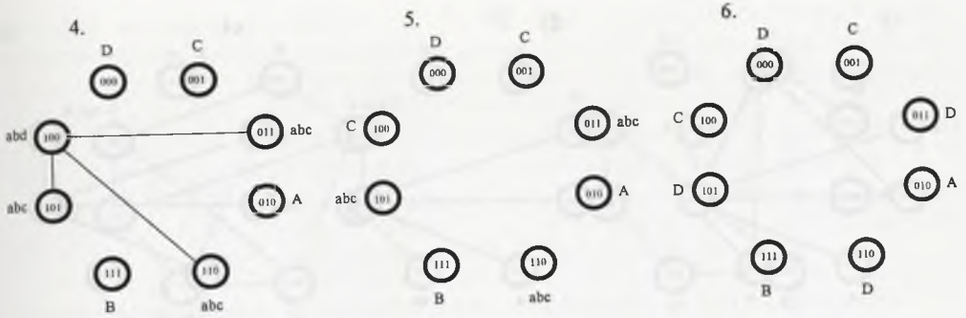
- wybierany jest wierzchołek z maksymalną liczbą kolorów zabronionych,
- wybierany jest wierzchołek, do którego dochodzi maksymalna liczba krawędzi.

Po wybraniu wierzchołka, przypisaniu odpowiednim wierzchołkom kolorów dozwolonych i zabronionych, wykonywana jest redukcja grafu, polegająca na eliminacji krawędzi łączących wybrany wierzchołek z innymi wierzchołkami grafu. Po eliminacji krawędzi wybierany jest kolejny (i+1) wierzchołek, tym razem po analizie grafu zredukowanego.

Przykład 4.8

Zastosujmy powyższy algorytm kolorowania wierzchołków grafu dla grafu z rys. 4.8. Kolejne etapy kolorowania wierzchołków (1-6) przedstawione są na rys. 4.9.





Rys.4.9. Kolejne etapy kolorowania grafu
Fig. 4.9. The successive steps of the colouring process

W przypadku dekompozycji funkcji słabo określonych możliwe jest włączenie do procesu kolorowania dodatkowego elementu, prowadzącego do minimalizacji bloku związanego.

Celem zorientowanego na realizację funkcji w strukturach typu PAL kolorowania grafu jest przyporządkowanie kolorów w taki sposób, aby proces kolorowania wpływał na minimalizację wykorzystywanych iloczynów. Algorytm kolorowania w tym przypadku składa się z dwóch etapów.

Przykład 4.9

Rozpatrzmy pierwszy etap kolorowania wierzchołków grafu niezgodności kolumn funkcji słabo określonej $f: B^5 \rightarrow B$ opisanej siatką Karnaugha (rys. 4.10a). Graf niezgodności kolumn przedstawiony jest na rys. 4.10.b. Poszczególne kroki kolorowania zobrazowano na rys. 4.10.c - 4.10.f.

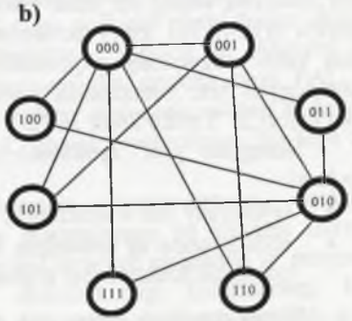
W ostatnim kroku pierwszego etapu (takiego samego jak w przypadku funkcji w pełni określonych) przypisywane są wierzchołkom, które mają kolory zabronione (rys.4.10.f), wszystkie występujące na grafie kolory dozwolone (rys. 4.10.g).

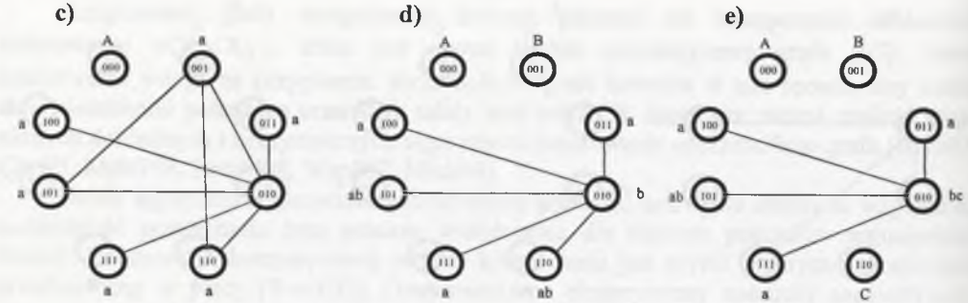
a)

$\mathbf{X}_1=\{c,d,e\}$, $\mathbf{X}_2=\{a,b\}$

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

b)



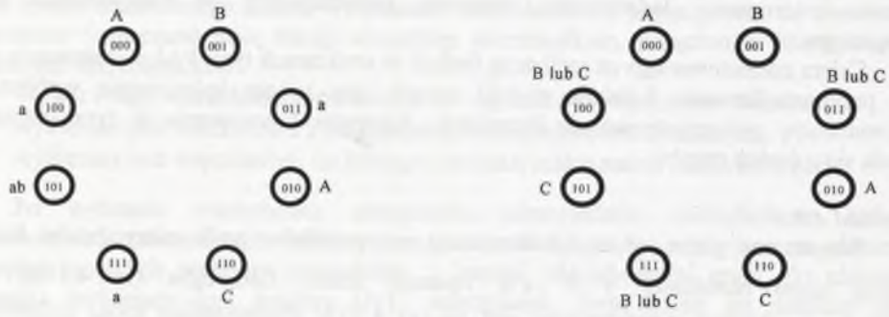


f)

Wynik 1 etapu kolorowania przed przypisaniem kolorów dozwolonych wierzchołkom z kolorami zabronionymi

g)

Wynik 1 etapu kolorowania po przypisaniu wszystkim wierzchołkom kolorów dozwolonych



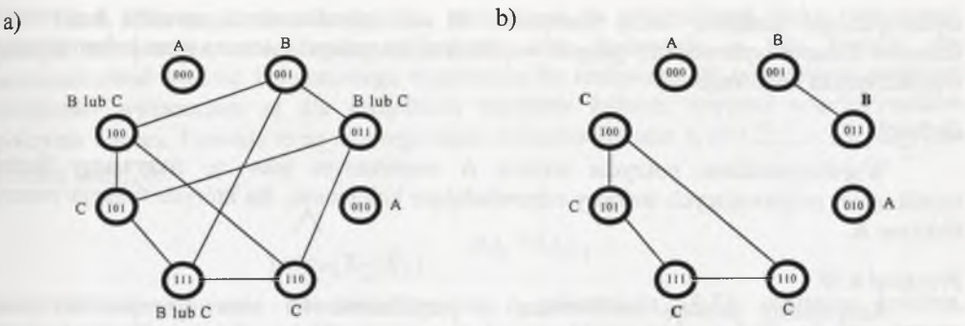
Rys. 4.10. Pierwszy etap kolorowania grafu
Fig. 4.10. The first step of the colouring process

Celem drugiego etapu kolorowania grafu jest przypisanie jednego koloru wierzchołkom, które zawierają więcej niż jeden kolor dozwolony. W tym celu tworzony jest graf zgodności kolorów (rys.4.11a), który powstaje przez połączenie wierzchołków mających więcej niż jeden kolor dozwolony z pozostałymi wierzchołkami z identycznym, jednym kolorem dozwolonym (tzn. wierzchołek z kolorem „B lub C" łączy się z wierzchołkami mającymi tylko kolor „B" i tylko kolor „C").

Następnie graf zgodności kolorów podlega redukcji zgodnie z następującymi regułami:

- wybierany jest wierzchołek, który ma więcej niż jeden kolor dozwolony,
- eliminowane są krawędzie łączące wybrany wierzchołek z wierzchołkami, których odległość kodowa jest większa od 1 od kodu wybranego wierzchołka.

Wynikiem tego etapu jest zredukowany graf zgodności kolorów (rys.4.11b), na podstawie którego przypisywane są kolory wierzchołkom, które po pierwszym etapie miały większą liczbę kolorów dozwolonych.



Rys. 4.11. Drugi etap kolorowania grafu
Fig. 4.11. The second step of colouring process

Zaproponowana metoda analizy grafu zgodności kolorów wplata w proces kolorowania element wpływający na minimalizację liczby iloczynów wykorzystywanych w bloku związanym, powstającym w wyniku dekompozycji kolumnowej. W ten sposób w jeden z pierwszych etapów dekompozycji wpleciony został element heurystycznej minimalizacji liczby użytych iloczynów.

4.4.2. Metoda kodowania wzorców kolumn zorientowana na realizację układów w strukturach typu PAL

Złożoność kolumnowa matrycy podziałów $\nu(\mathbf{X}_2 \mid \mathbf{X}_1)$ jest równa liczbie chromatycznej grafu niezgodności kolumn $\gamma(G)$, stąd liczba bitów potrzebna do rozróżnienia wzorców kolumn wynosi $\lceil \lg_2 \gamma(G) \rceil$. Blok związany, będący wynikiem dekompozycji kolumnowej, jest specyficznym $\lceil \lg_2 \gamma(G) \rceil$-wyjściowym transkoderem, o kodzie wyjściowym ustalonym przez kodowanie wzorców kolumn. Dobór kodowania wzorców kolumn istotnie wpływa na minimalizację funkcji logicznych realizowanych w bloku związanym i wolnym. W literaturze można znaleźć różne metody kodowania wyjść oraz wejść [AshaD92, DevaN91, ShiB93, YangC91], związane najczęściej z bliźniaczym procesem kodowania stanów wewnętrznych automatów sekwencyjnych. Rozpatrywane są również metody minimalizacji liczby wykorzystywanych iloczynów w strukturach PLA po podziale pierwotnego układu i odpowiednim kodowaniu wyjściowo-wejściowym [PerkB96]. Metody te mogłyby być bezpośrednio przeniesione w proces syntezy przeznaczonej dla struktur typu PAL, nie gwarantując jednak zadowalających rezultatów. Cenne byłoby wplecenie w etap kodowania wzorców kolumn specyfiki bloku logicznego typu PAL, charakterystycznego dla tego typu układów.

Zaproponowana w rozdziale 4.3 idea kodowania równomiernego, doskonale sprawdzająca się w przypadku realizacji transkodera powstającego po analizie liczności zbioru słów układu z ograniczoną liczbą wejść, może być również przydatna w przypadku realizacji bloku związanego, będącego wynikiem dekompozycji kolumnowej. Istota zaproponowanego kodowania wzorców kolumn polega na tworzeniu słów kodowych w taki sposób, żeby wykorzystywały one minimalną liczbę iloczynów. Dodatkowo kolumnom, które zawierają wzorce kolumn występujące częściej, przyporządkowywane są kody

wykorzystujące mniejszą liczbę iloczynów. W celu przedstawienia sposobu kodowania wzorców kolumn wprowadźmy pojęcie współczynnika pokrycia wzorca oraz kolumnowego współczynnika kodowego.

Definicja 4.2

Współczynnikiem pokrycia wzorca A oznaczanym jako η_A nazywamy liczbę implikantów pokrywających wektory odpowiadające kolumnom, dla których funkcja tworzy wzorzec A.

Przykład 4.10

Rozpatrzmy funkcję analizowaną w przykładach 4.1 oraz 4.7, przedstawioną ponownie na rys. 4.12.a. Algorytm kolorowania wierzchołków grafu niezgodności kolumn (przykład 4.8) doprowadził do rozróżnienia wzorców kolumn (rys. 4.12.b) i wyznaczenia złożoności kolumnowej $\nu(\mathbf{X}_2 \mid \mathbf{X}_1)=4$. Kolejnym etapem syntezy jest kodowanie wzorców kolumn. Dla wzorców kolumn A, B, C, D wyznaczone zostały współczynniki pokrycia wzorca η_A, η_B, η_C, η_D.

a)

$\mathbf{X}_1=\{c,d,e\}$, $\mathbf{X}_2=\{a,b\}$

ab \ cde	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
	D	C	D	A	D	B	D	C

$$A \Leftrightarrow \{010\}_{cde} \Rightarrow \eta_A = 1$$

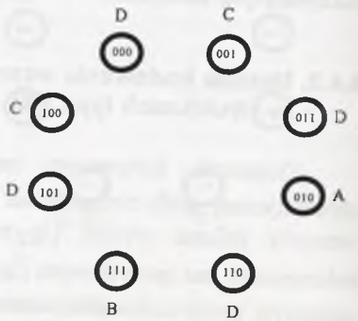
$$B \Leftrightarrow \{111\}_{cde} \Rightarrow \eta_B = 1$$

$$C \Leftrightarrow \left\{\begin{matrix}001\\100\end{matrix}\right\}_{cde} \Rightarrow \eta_C = 2$$

$$D \Leftrightarrow \left\{\begin{matrix}000\\011\\110\\101\end{matrix}\right\}_{cde} \Rightarrow \eta_D = 4$$

b)

Wynik kolorowania wierzchołków grafu niezgodności kolumn



$$\gamma(G) = \nu(\mathbf{X}_2 \mid \mathbf{X}_1) = 4$$

Rys. 4.12. Wyznaczanie współczynników pokrycia wzorca; a) siatka Karnaugha; b) wynik kolorowania grafu

Fig. 4.12. Determination of covering pattern coefficients; a) Karnaugh map, b) result of the colouring process

Definicja 4.3

Kolumnowym współczynnikiem kodowym nazywamy liczbę η_k, wyznaczoną z zależności

$$\eta_k = \sum_{i=1}^{\nu(\mathbf{X}_2|\mathbf{X}_1)} \eta_{A_i} H_{A_i}, \qquad (4.4)$$

gdzie η_{A_i} jest współczynnikiem pokrycia wzorca kolumny A_i, natomiast H_{A_i} jest liczbą jedynek występujących w słowie kodowym, przyporządkowanym wzorcowi kolumny A_i.

Istota kodowania wzorców kolumn, prowadząca do minimalizacji bloku związanego, polega na przypisywaniu wzorcom kolumn słów kodowych w taki sposób, aby minimalizować wartość kolumnowego współczynnika kodowego. W tym celu w pierwszej kolejności wyznaczane są dla wszystkich wzorców kolumn wartości współczynników pokrycia wzorca. Pozwala to na uszeregowanie wzorców kolumn A_i (i=1,2,...,$\nu(\mathbf{X}_2 \mid \mathbf{X}_1)$) do postaci takiej, że:

$$\bigwedge_{1 \le i \le \nu(\mathbf{X}_2|\mathbf{X}_1)} \eta_{A_i} \ge \eta_{A_{i+1}} \qquad (4.5)$$

Po uszeregowaniu wzorców zgodnie z zależnością (4.5), wzorcom kolumn, poczynając od A_1, przyporządkowywane są kolejne słowa kodowe kodu równomiernego [Kani01b].

Rozpatrzmy kodowanie wzorców kolumn funkcji z przykładu 4.10. Uszeregowanie wzorców kolumn zgodnie z wartościami współczynników pokrycia wzorca umożliwia przypisanie słów kodowych kodu równomiernego poszczególnym wzorcom.

Ponieważ $\eta_D > \eta_C > \eta_B = \eta_A$, stąd istnieją dwa rozwiązania, dla których kolumnowy współczynnik kodowy przyjmuje wartość minimalną.

1)

D - 00 $\Rightarrow H_D$=0
C - 01 $\Rightarrow H_C$=1
B - 10 $\Rightarrow H_B$=1
A - 11 $\Rightarrow H_A$=2

$$\eta_k = \eta_D H_D + \eta_C H_C + \eta_B H_B + \eta_A H_A = 4*0+2*1+1*1+1*2 = 5$$

2)

D - 00 $\Rightarrow H_D$=0
C - 01 $\Rightarrow H_C$=1
A - 10 $\Rightarrow H_A$=1
B - 11 $\Rightarrow H_B$=2

$$\eta_k = \eta_D H_D + \eta_C H_C + \eta_A H_A + \eta_B H_B = 4*0+2*1+1*1+1*2 = 5$$

W celu oceny złożoności układu po dekompozycji wprowadźmy dwa współczynniki.

Definicja 4.4

Współczynnikiem złożoności bloku związanego z nazywać będziemy liczbę iloczynów potrzebnych do realizacji funkcji implementowanych w bloku związanym. Współczynnik złożoności wyznacza się sumując liczby iloczynów potrzebnych do realizacji wszystkich p-funkcji realizowanych w tym bloku, tzn. $z = \sum_{i=1}^{p} z_i$.

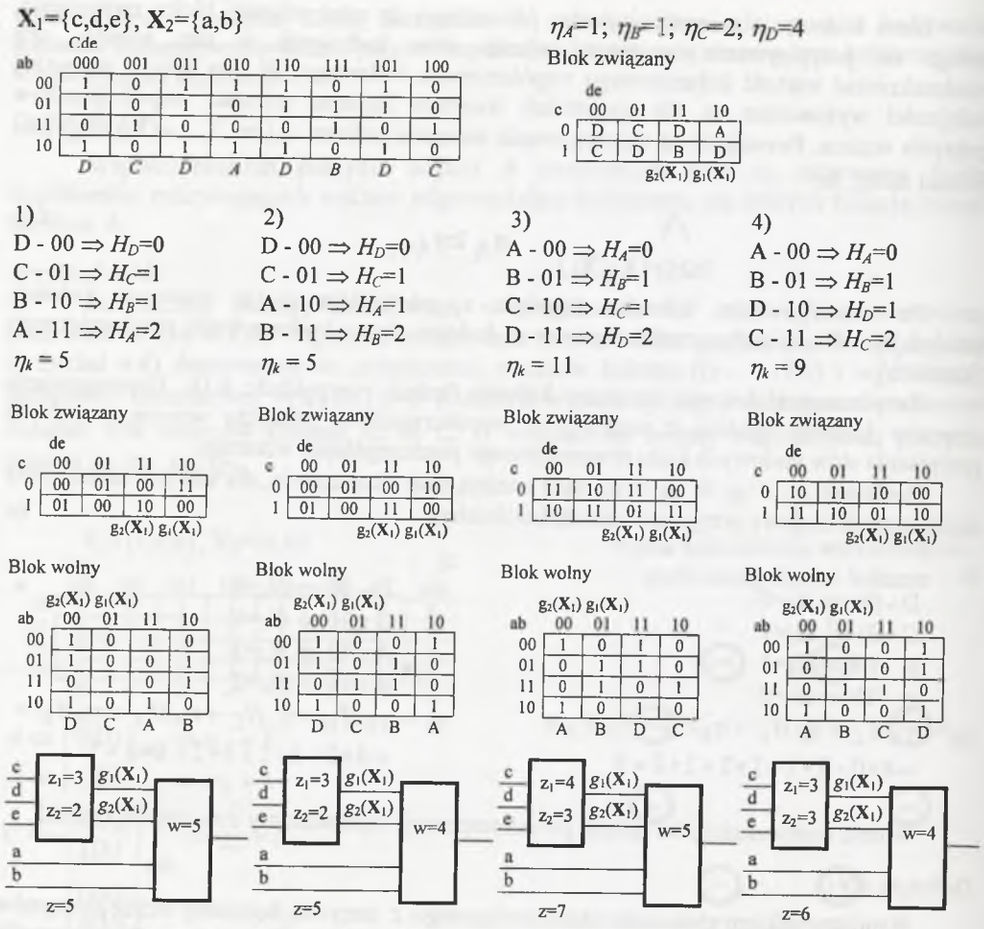
Definicja 4.5

Współczynnikiem złożoności bloku wolnego w nazywać będziemy liczbę iloczynów potrzebnych do realizacji funkcji implementowanych w bloku wolnym. W przypadku realizacji funkcji m-wyjściowej wartość współczynnika złożoności wyznacza się z zależności

$$w = \sum_{i=1}^{m} w_i .$$

Przykład 4.11

Rozpatrzmy cztery przykładowe kodowania wzorców kolumn dla funkcji rozpatrywanej w przykładzie 4.10. Wyniki kodowania przedstawione są w kolumnach z oznaczeniami 1-4.



Analizując przykładowe wyniki, można zauważyć zależność współczynnika złożoności bloku związanego (z) od kolumnowego współczynnika kodowego (η_k). Wyniki przeprowadzonych eksperymentów potwierdziły hipotezę, że dla mniejszych wartości kolumnowego współczynnika kodowego η_k uzyskuje się mniejszą złożoność bloku związanego. Heurystyczna zasada doboru kodowania wzorców kolumn, oparta na analizie współczynników pokrycia wzorca, wpływa tylko na minimalizację bloku związanego. Istotną cechą kodowania powinna być również minimalizacja liczby iloczynów wchodzących w skład bloku wolnego. Widać więc potrzebę rozszerzenia idei kodowania wzorców kolumn o element wpływający na minimalizację bloku wolnego.

Weźmy pod uwagę dwa przykładowe wzorce kolumn A i C. Wprowadźmy zmienną x przyjmującą wartość $x = 0$ dla wzorca A i $x = 1$ dla wzorca C. Rozważmy funkcję f_{AC}, której argumentami są zmienne zbioru wolnego $\mathbf{X}_2$ oraz zmienna x.

Definicja 4.6

Współczynnikiem pokrycia pary wzorców (A,C) oznaczanym jako η_{AC} nazywamy minimalną liczbę implikantów potrzebnych do pokrycia funkcji $f_{AC} : B^{\overline{X_2}+1} \to B$.

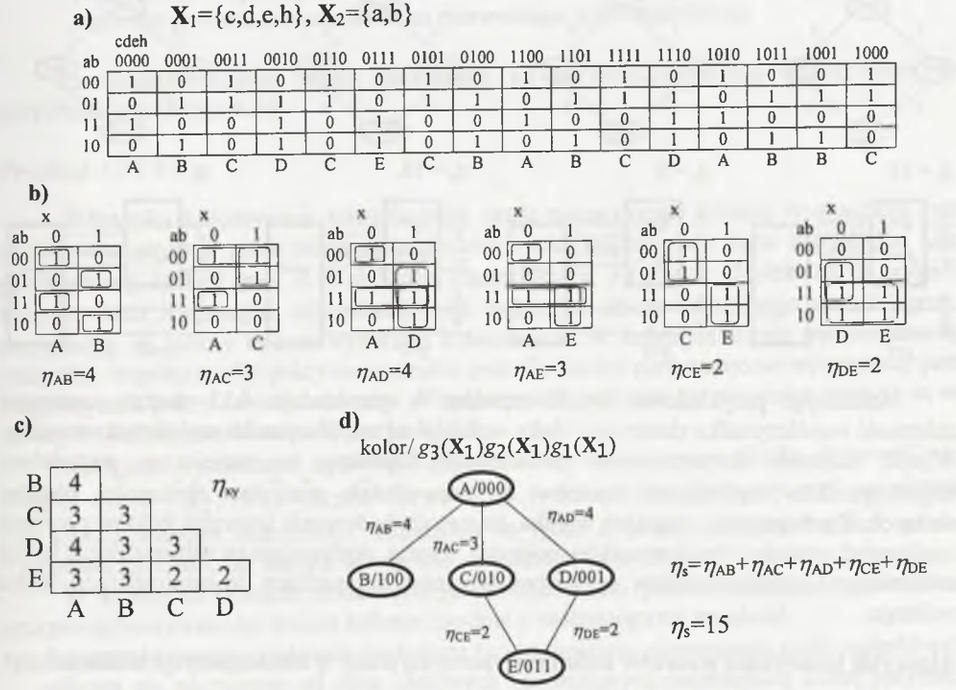
Przyporządkowując poszczególnym wzorcom kolumn odpowiednie słowa kodowe można utworzyć ***graf sąsiedztwa wzorców***. Wierzchołki tego grafu skojarzone są z wzorcami kolumn siatki Karnaugha, natomiast krawędziom łączącym wierzchołki, których odległość kodowa jest równa jeden, przyporządkowane są odpowiednie wartości współczynników pokrycia par wzorców.

Definicja 4.7

Współczynnikiem sąsiedztwa wzorców, oznaczanym jako η_s, nazywamy sumę współczynników pokrycia par wzorców, skojarzonych z krawędziami grafu sąsiedztwa wzorców.

Przykład 4.12

Dla przykładowej funkcji $f: B^5 \to B$ (rys.4.1.a) wyznaczmy przykładowe współczynniki pokrycia par wzorców (rys.4.13.b,c). Przypisując poszczególnym wzorcom kolumn kolejne słowa kodowe kodu równomiernego można utworzyć graf sąsiedztwa wzorców (rys.4.13.d) i wyznaczyć współczynnik sąsiedztwa wzorców.



Rys. 4.13.Elementy kodowania wykorzystywane do minimalizacji bloku wolnego; a) siatka Karnaugha, b) metoda wyznaczania współczynników pokrycia par wzorców; c) tablica trójkątna zawierająca wszystkie współczynniki pokrycia par wzorców; d) graf sąsiedztwa wzorców
Fig. 4.13. Coding elements used for minimization of free logic blocks; a) Karnaugh map; b) method for determination of coefficients of pattern pairs covering; c) the triangle table that comprises all coefficients of pattern pairs covering; d) graph of pattern neighbourhood

Przykład 4.13

Wróćmy do funkcji z przykładu 4.11 i rozpatrzmy analizowane tam cztery przykładowe sposoby kodowania wzorców kolumn, obserwując wartości współczynników sąsiedztwa wzorców η_s. Wyniki kodowania przedstawione są w kolumnach z oznaczeniami 1-4.

$\mathbf{X}_1$={c,d,e}, $\mathbf{X}_2$={a,b}

ab \ cde	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
	D	*C*	*D*	*A*	*D*	*B*	*D*	*C*

Tablica trójkątna zawierająca wszystkie współczynniki pokrycia par wzorców

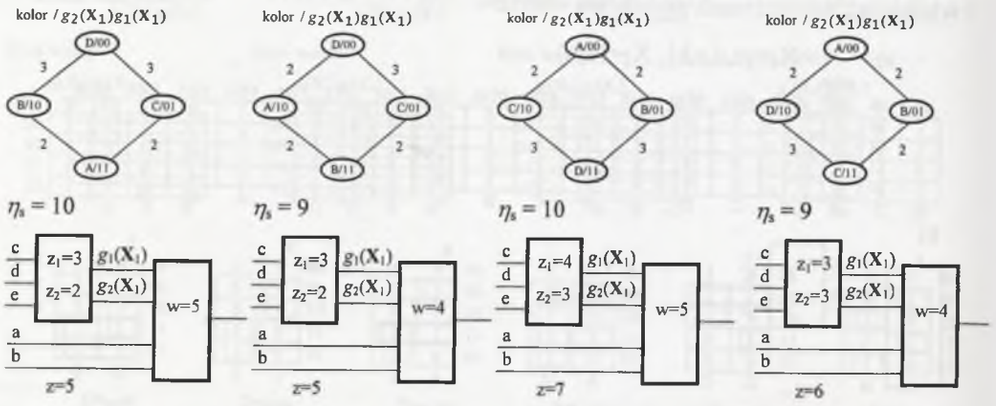
	A	B	C
B	2		
C	2	2	
D	2	3	3

1)
D - 00
C - 01
B - 10
A - 11
$\eta_k = 5$

2)
D - 00
C - 01
A - 10
B - 11
$\eta_k = 5$

3)
A - 00
B - 01
C - 10
D - 11
$\eta_k = 11$

4)
A - 00
B - 01
D - 10
C - 11
$\eta_k = 9$



Analizując przykładowe wyniki zawarte w przykładzie 4.13 można zauważyć zależność współczynnika złożoności bloku wolnego od współczynnika sąsiedztwa wzorców. Wyniki licznych eksperymentów potwierdzają hipotezę, że mniejszym wartościom współczynników sąsiedztwa wzorców η_s odpowiadają mniejsze złożoności bloków wolnych. Z powyższych rozważań wynika, że metoda kodowania wzorców kolumn powinna analizować wartości współczynników pokrycia wzorca, wpływające na minimalizację bloku związanego i współczynników sąsiedztwa wzorców, prowadzące do minimalizacji bloku wolnego.

Algorytm kodowania wzorców kolumn wykorzystywany w dekompozycji kolumnowej:

1. Wyznaczenie współczynników pokrycia wzorców;
2. Wyznaczenie słów kodowych kodu równomiernego:
   liczba słów = $\gamma(G)$; długość słowa = $\lceil \lg_2 \gamma(G) \rceil$;
3. Utworzenie pierwotnego grafu sąsiedztwa (graf przed przyporządkowaniem konkretnych wzorców kolumn do wierzchołków opisanych przez poszczególne słowa kodowe kodu równomiernego);

4. Uszeregowanie wzorców kolumn na podstawie współczynników pokrycia wzorców do postaci takiej, że

$$\bigwedge_{1 \le i \le \gamma(G)} \eta_{A_i} \ge \eta_{A_{i+1}},$$

   gdzie A_i (i=1,2,..., $\gamma(G)$) oznacza wzorzec kolumny;

5. Przyporządkowanie wzorców kolumn do poszczególnych wierzchołków pierwotnego grafu sąsiedztwa:

   - wybierane są wzorce kolumn z maksymalną wartością współczynnika pokrycia wzorca; rozpatrywane są wszystkie przyporządkowania jednego z tych wzorców do słowa kodowego składającego się z samych "0";
   - do słów kodowych zawierających jedną jedynkę (H_i=1) przyporządkowywane są wzorce kolumn, które zapewniają minimalną wartość składników tworzących współczynnik sąsiedztwa wzorców;
   - do słów kodowych zawierających dwie jedynki (H_i=2) przyporządkowywane są wzorce kolumn, które zapewniają minimalną wartość kolejnych składników tworzących współczynnik sąsiedztwa wzorców itd. do skojarzenia wszystkich wzorców kolumn z wierzchołkami pierwotnego grafu sąsiedztwa.

Prześledźmy kolejne etapy kodowania na przykładzie funkcji analizowanej w przykładach 4.11 oraz 4.13.

Przykład 4.14

W wyniku kolorowania wierzchołków grafu niezgodności kolumn wyznaczana jest liczba chromatyczna grafu odpowiadająca złożoności kolumnowej siatki Karnaugha. Dla przykładowej funkcji (rys. 4.14.a) liczba chromatyczna $\gamma(G) = 4$. Umożliwia to podział kolumn siatki Karnaugha, odpowiadających wektorom zbioru związanego na $\gamma(G)$-grup, przypisując im kolory będące wynikiem kolorowania. W kolejnym etapie wyznaczane są wszystkie współczynniki pokrycia wzorców oraz dla każdej pary wzorców wyznaczany jest współczynnik pokrycia pary wzorców. Współczynniki pokrycia par wzorców zawarte są w tablicy trójkątnej przedstawionej na rys.4.14.b.

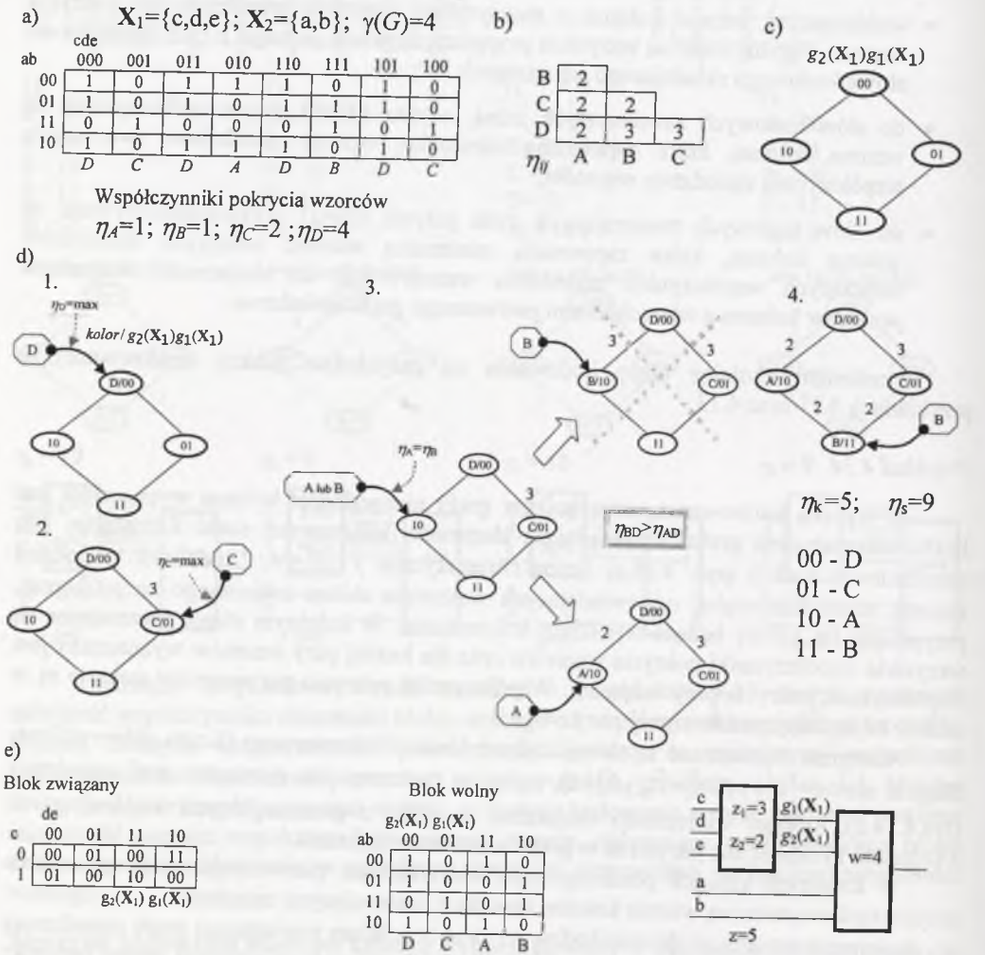
Następnie wyznaczane są słowa kodowe kodu równomiernego (liczba słów=$\gamma(G)$=4; długość słowa=$\lceil \lg_2 \gamma(G) \rceil$=$\lceil \lg_2 4 \rceil$=2), po czym tworzony jest pierwotny graf sąsiedztwa (rys.4.14.c), którego wierzchołki skojarzone są tylko z poszczególnymi kodami, a nie z kolorami i kodami, tak jak jest to w grafie sąsiedztwa wzorców.

W kolejnych krokach poszczególnym wierzchołkom pierwotnego grafu sąsiedztwa przyporządkowywane są wzorce kolumn zgodnie z następującymi zasadami:

- skojarzenie wzorca ze słowem kodowym (wierzchołkiem pierwotnego grafu sąsiedztwa) odbywa się poczynając od słów kodowych zawierających najmniejszą liczbę jedynek; kolejno rozpatrywane są słowa, dla których H_i=0,1,...itd.;
- wzorce kolumn z większymi współczynnikami pokrycia wzorców, przyporządkowywane są wierzchołkom z mniejszą liczbą jedynek w słowach kodowych (minimalizacja kolumnowego współczynnika kodowego);
- w przypadku istnienia kilku wzorców, z identycznymi współczynnikami pokrycia wzorców, wybierany jest wzorzec, dla którego współczynnik pokrycia pary wzorców

(wzorzec analizowany; wzorzec łączący się z analizowanym wierzchołkiem skojarzony z kodem zawierającym mniejszą liczbę jedynek) ma wartość najmniejszą (minimalizacja współczynnika sąsiedztwa wzorców).

Kolejne kroki przyporządkowania wzorców do wierzchołków pierwotnego grafu sąsiedztwa przedstawia rys.4.14.d. Uzyskane kodowanie prowadzi do rozwiązania przedstawionego na rys. 4.14.e.



Rys. 4.14. Przykład kodowania wzorców kolumn, a) siatka Karnaugha; b) tablica trójkątna zawierająca współczynniki pokrycia par wzorców; c) pierwotny graf sąsiedztwa; d) kolejne etapy przyporządkowania wzorców do wierzchołków pierwotnego grafu sąsiedztwa; e) wynik kodowania

Fig. 4.14. The example of the method for encoding the column patterns; a) Karnaugh map; c) the triangle table with coefficients of pattern pairs covering; d) the primary graph of pattern neighbourhood; d) successive steps of assigning the patterns to the nodes of the primary graph of pattern neighborhood; e) result of encoding

4.4.3. Algorytm implementacji funkcji jednowyjściowych w strukturach typu PAL oparty na dekompozycji kolumnowej

Konsekwencją dekompozycji kolumnowej jest podział układu na dwa podukłady: związany i wolny. Podział ten prowadzi do ekspansji liczby wyjść. Każde dodatkowe wyjście pociąga za sobą użycie kolejnego bloku logicznego typu PAL. W wyniku podziału wykorzystuje się co najmniej p-bloków logicznych typu PAL do realizacji bloku związanego. Wartość p jest zależna od złożoności kolumnowej $\nu(\mathbf{X}_2 \mid \mathbf{X}_1)$ zgodnie z zależnością $2^p \geq \nu(\mathbf{X}_2 \mid \mathbf{X}_1)$. Fakt ten sprawia, że zastosowanie dekompozycji kolumnowej jako elementu minimalizacji może być opłacalne tylko wtedy, gdy w klasycznym podejściu wykorzystanie sprzężeń zwrotnych prowadziłoby do użycia większej liczby bloków logicznych typu PAL.

Minimalna liczba wyjść, równocześnie minimalna liczba bloków logicznych typu PAL, potrzebnych do realizacji układu po podziale, będącego wynikiem dekompozycji kolumnowej, wynosi p+1 (co najmniej p bloków logicznych typu PAL potrzebnych jest do realizacji bloku związanego i co najmniej jeden blok do realizacji bloku wolnego).

Rozpatrzmy realizację funkcji $f_i : B^n \to B$ na k-iloczynowych blokach PAL. Zminimalizowana postać funkcji $f_i : B^n \to B$, przedstawiona w postaci zbioru Δ_{f_i}-implikantów, może być zrealizowana metodą klasyczną za pomocą δ_{f_i}-bloków logicznych typu PAL, zawierających k-iloczynów, przy czym $\delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1$.

Zastosowanie dekompozycji kolumnowej może prowadzić do ograniczenia liczby bloków logicznych typu PAL tylko w sytuacji, gdy:

$$p+1 < \delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 \qquad (4.6)$$

Zależność (4.6) stanowi warunek konieczny ograniczenia liczby bloków logicznych typu PAL, który może eliminować pewne podziały z dalszych rozważań lub wręcz wskazywać na niemożliwość uzyskania lepszych rozwiązań od metody klasycznej. Jeżeli na przykład zminimalizowana postać funkcji $f_i : B^n \to B$ opisana jest przez zbiór mniej niż $2k$-implikantów, to stosowanie dekompozycji kolumnowej jest bezcelowe, gdyż na pewno nie ograniczy liczby bloków logicznych wykorzystywanych w metodzie klasycznej.

W poszukiwaniu odpowiedniego podziału argumentów funkcji występuje problem doboru odpowiedniego podziału stwarzającego najlepsze warunki minimalizacji. Wyniki przeprowadzonych eksperymentów wskazały, że w przypadku struktur typu PAL najlepsze rezultaty można uzyskać wykorzystując podziały prowadzące do podziału układu na podukłady o zbliżonej złożoności. W przypadku realizacji funkcji jednowyjściowej złożoność bloku ocenia się na podstawie liczby wejść. Przeprowadzone eksperymenty wykazały, że przyjęte kryterium oceny złożoności bloku jest wystarczające. W tej sytuacji dekompozycja kolumnowa funkcji jednowyjściowej wykorzystywana jest do podziału układu na bloki o zbliżonej liczbie wejść (równej lub różniącej się o jeden).

Zaprezentowane w poprzednich rozdziałach sposoby wyznaczania złożoności kolumnowej, kodowania wzorców kolumn oraz rezultaty przedstawionych rozważań

umożliwiają przedstawienie heurystycznego algorytmu implementacji funkcji jednowyjściowej w strukturach typu PAL.

Algorytm dekompozycji kolumnowej funkcji jednowyjściowej, prowadzący do efektywnego podziału układu na blok związany i wolny, realizowane za pomocą *k*-iloczynowych bloków PAL:

1. Minimalizacja funkcji $f_i : B^n \to B$ (Espresso); wyznaczanie parametru $\delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1$

2. Jeżeli $\Delta_{f_i} < 2k$, to realizacja funkcji $f_i : B^n \to B$ metodą klasyczną; w przeciwnym przypadku analiza podziałów argumentów funkcji i poszukiwanie odpowiedniej dekompozycji kolumnowej, tzn.:
   A. Rozpatrywane są podziały zbioru $\mathbf{I} = \{i_n, ..., i_2, i_1\}$ na podzbiory $\mathbf{X}_1 = \{i_{n_z}, ..., i_2, i_1\}$ oraz $\mathbf{X}_2 = \{i_n, ..., i_{n_z+2}, i_{n_z+1}\}$ takie, że:
      jeżeli n jest liczbą nieparzystą, to $\overline{\mathbf{X}}_1 = \overline{\mathbf{X}}_2 + 1$;
      jeżeli n jest liczbą parzystą, to $\overline{\mathbf{X}}_1 = \overline{\mathbf{X}}_2 + 2$.
   B. Dla poszczególnych podziałów wyznaczane są złożoności kolumnowe, wykorzystując algorytm kolorowania grafu niezgodności kolumn, zorientowany na realizację układów w strukturach typu PAL;
   C. Spośród rozpatrywanych podziałów wybierany jest ten, dla którego $p+1 < \delta_{f_i}$, gdzie $p = \lceil \lg_2 \nu(\mathbf{X}_2 \mid \mathbf{X}_1) \rceil = \lceil \lg_2 \gamma(G) \rceil$ oraz liczba chromatyczna grafu niezgodności kolumn $\gamma(G)$ przyjmuje wartość minimalną; jeżeli w grupie analizowanych podziałów podział taki nie istnieje, to analizowana funkcja realizowana jest metodą klasyczną;
   D. Jeżeli dla wybranego podziału zachodzi zależność $\overline{\mathbf{X}}_2 + p > \overline{\mathbf{X}}_1 + 1$, to poszukiwany jest kolejny podział z grupy podziałów, dla której $\overline{\mathbf{X}}_1 == \overline{\mathbf{X}}_1 + 1$ i $\overline{\mathbf{X}}_2 == \overline{\mathbf{X}}_2 - 1$ itd. p. 2B;
   E. Spośród wybranych w kolejnych krokach podziałów, wybierany jest jeden podział, dla którego $\max(\overline{\mathbf{X}}_1; \overline{\mathbf{X}}_2 + p) = \min$; jeżeli istnieje więcej podziałów spełniających powyższy warunek, to wybierany jest spośród nich ten, dla którego $\overline{\mathbf{X}}_1 p + \overline{\mathbf{X}}_2 = \min$.
3. Dla wybranego podziału wykonywane jest kodowanie wzorców kolumn metodą przedstawioną w rozdziale 4.4.2.
4. Minimalizacja funkcji realizowanych w bloku związanym $g : B^{n_z} \to B^p$ (Espresso-Dso) oraz w bloku wolnym $f_w : B^{n-n_z+p} \to B$ (Espresso); wyznaczanie liczby bloków logicznych typu PAL, potrzebnych do realizacji układu po podziale; liczba ta oznaczana przez symbol δ_{zw} jest sumą liczby bloków potrzebnych do realizacji bloku związanego (δ_z) i wolnego (δ_w):

$$\delta_{zw} = \delta_z + \delta_w = \sum_{i=1}^{p} \left(\left\lceil \frac{z_i - k}{k-1} \right\rceil + 1 \right) + \left\lceil \frac{w-k}{k-1} \right\rceil + 1$$

5. Jeżeli $\delta_{f_i} \le \delta_{zw}$, to realizacja funkcji metodą klasyczną; w przeciwnym przypadku kolejny etap dekompozycji kolumnowej - podział bloku związanego i wolnego (p.1).

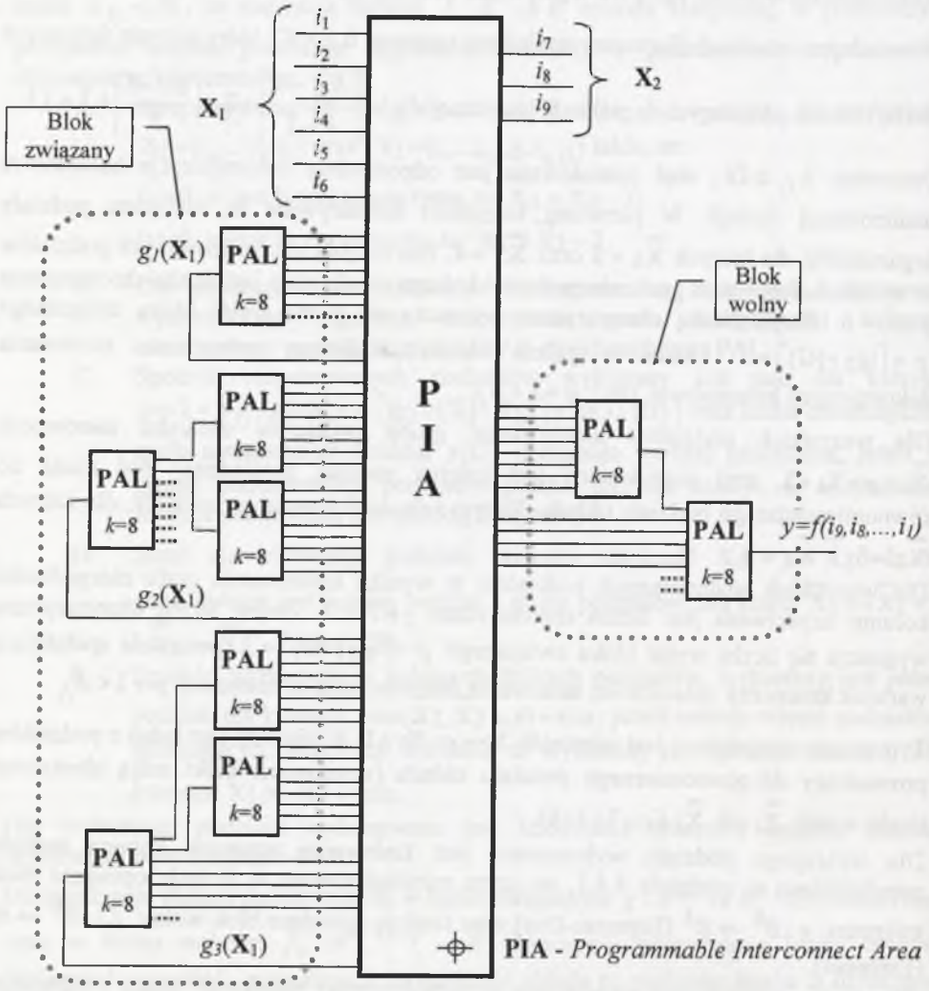
Przykład 4.15

Rozpatrzmy realizację funkcji testowej *9sym* ($f_i : B^9 \to B$) korzystając z bloków logicznych typu PAL, zawierających 8 iloczynów (poniżej przedstawione są kolejne etapy poszukiwania rozwiązania zgodne z algorytmem dekompozycji kolumnowej funkcji).

- W wyniku minimalizacji (Espresso) uzyskujemy parametr $\Delta_{f_i} = 87$, który pozwala wyznaczyć liczbę bloków, potrzebnych do realizacji klasycznej $\delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 = \left\lceil \frac{87-8}{7} \right\rceil + 1 = 13$.
- Ponieważ $\Delta_{f_i} \ge 2k$, stąd poszukiwana jest odpowiednia dekompozycja kolumnowa analizowanej funkcji. W pierwszej kolejności rozpatrywane są rozłączne podziały argumentów, dla których $\overline{\mathbf{X}}_1 = 5$ oraz $\overline{\mathbf{X}}_2 = 4$. Dla wszystkich analizowanych podziałów w wyniku kolorowania grafu niezgodności kolumn uzyskiwana jest liczba chromatyczna $\gamma(G) = 6$. Znając liczbę chromatyczną wyznacza się liczbę wyjść bloku związanego $p = \lceil \lg_2 \gamma(G) \rceil = 3$. Liczba ta spełnia warunek konieczny opłacalności stosowania dekompozycji kolumnowej, tzn.: $p + 1 < \delta_{f_i}$.
- Dla wszystkich podziałów analizowanej grupy podziałów zachodzi nierówność $\overline{\mathbf{X}}_2 + p > \overline{\mathbf{X}}_1 + 1$, stąd poszukiwany jest kolejny podział, prowadzący być może do równomierniejszego podziału układu. W tym celu analizowane są podziały, dla których $\overline{\mathbf{X}}_1 = 6$ i $\overline{\mathbf{X}}_2 = 3$.
- Dla wszystkich analizowanych podziałów w wyniku kolorowania grafu niezgodności kolumn uzyskiwana jest liczba chromatyczna $\gamma(G) = 7$. Znając liczbę chromatyczną wyznacza się liczbę wyjść bloku związanego $p = \lceil \lg_2 \gamma(G) \rceil = 3$, ponownie spełniającą warunek konieczny opłacalności stosowania dekompozycji kolumnowej $p + 1 < \delta_{f_i}$.
- Tym razem niespełniona jest zależność $\overline{\mathbf{X}}_2 + p > \overline{\mathbf{X}}_1 + 1$. Wybierany jest jeden z podziałów prowadzący do równomiernego podziału układu (uzyskiwane bloki mają identyczną liczbę wejść: $\overline{\mathbf{X}}_1 = 6$; $\overline{\mathbf{X}}_2 + p = 3 + 3 = 6$).
- Dla wybranego podziału wykonywane jest kodowanie wzorców kolumn metodą przedstawioną w rozdziale 4.4.2, po czym minimalizowane są funkcje opisujące blok związany $g : B^6 \to B^3$ (Espresso-Dso) oraz funkcja opisująca blok wolny $f_w : B^6 \to B$ (Espresso).
- W wyniku minimalizacji uzyskiwane są współczynniki złożoności bloku związanego z_1=7, z_2=16, z_3=21 oraz bloku wolnego w=13, co pozwala wyznaczyć liczbę bloków logicznych typu PAL, potrzebnych do realizacji układu po podziale:

$$\delta_{zw} = \delta_z + \delta_w = \sum_{i=1}^{3}\left(\left\lceil \frac{z_i - k}{k-1} \right\rceil + 1\right) + \left\lceil \frac{w-k}{k-1} \right\rceil + 1 = 9$$

- Wyszukiwany podział jest akceptowany, ponieważ $\delta_{f_i} > \delta_{zw}$.
- Poszukiwana jest dekompozycja kolumnowa dla dwóch funkcji realizowanych w bloku związanym, dla których współczynniki złożoności wynoszą odpowiednio z_2=16, z_3=21. Poszukiwania te nie kończą się jednak sukcesem. Ostateczne rozwiązanie przedstawione na rys. 4.15 wykorzystuje tylko 9 bloków logicznych typu PAL, zawierających 8 iloczynów (4 bloki mniej niż metoda klasyczna)



Rys. 4.15. Wynik dekompozycji kolumnowej układu testowego *9sym*
Fig. 4.15. Results of column decomposition for the *9sym* benchmark

4.4.4. Dekompozycja kolumnowa funkcji wielowyjściowych

Wszystkie przedstawione w poprzednich rozdziałach algorytmy związane z dekompozycją kolumnową funkcji jednowyjściowej mogą być uogólnione i wykorzystywane dla funkcji wielowyjściowych. W celu przedstawienia dokładnego algorytmu zachodzi jednak potrzeba uogólnienia pewnych pojęć.

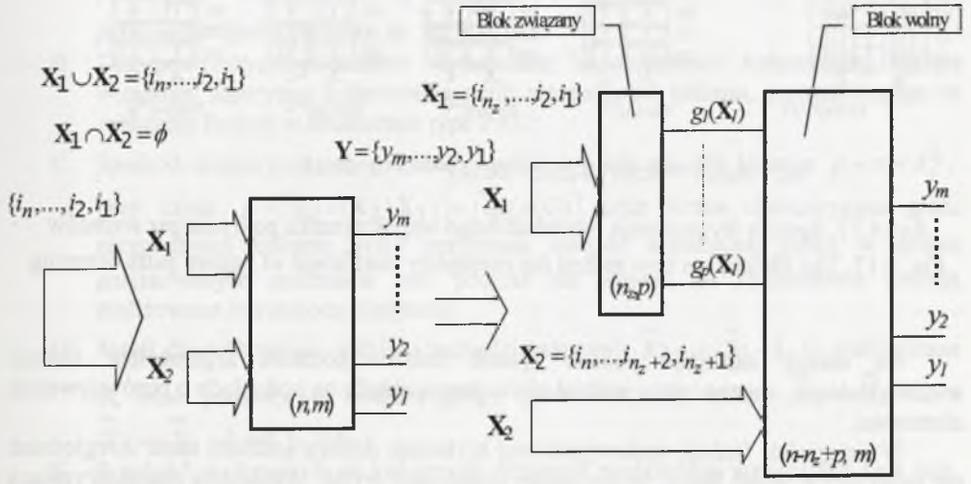
W przypadku funkcji wielowyjściowej $f: B^n \to B^m$ można równocześnie rozpatrywać matrycę podziałów (siatkę Karnaugha) wszystkich m-funkcji $f_i: B^n \to B$, i=1,2,...,m. Oznaczmy przez $\mathbf{Y} = \{y_m,...,y_2,y_1\}$ zbiór m-funkcji $y_i = f_i(i_n,...,i_2,i_1)$.

Funkcja $f: B^n \to B^m$ podlega dekompozycji wtedy i tylko wtedy, gdy złożoność kolumnowa matrycy podziałów $\nu(\mathbf{X}_2 \mid \mathbf{X}_1) \le 2^p$, tzn.

$$\nu(\mathbf{X}_2 \mid \mathbf{X}_1) \le 2^p \Leftrightarrow f(\mathbf{X}_2, \mathbf{X}_1) = F[g_1(\mathbf{X}_1), g_2(\mathbf{X}_1),..., g_p(\mathbf{X}_1), \mathbf{X}_2], \tag{4.7}$$

przy czym $\mathbf{X}_1 \cup \mathbf{X}_2 = \{i_n,...,i_2,i_1\}$ oraz $\mathbf{X}_1 \cap \mathbf{X}_2 = \phi$ [Ashe57, Curt62, Curt63].

Wykorzystanie takiego modelu dekompozycji prowadzi do podziału wielowyjściowego bloku scharakteryzowanego uporządkowaną parą liczb (liczba wejść, liczba wyjść), wynoszącą (n, m), na dwa inne o parametrach (n_z, p) i $(n-n_z+p, m)$. Podział układu odpowiadający dekompozycji funkcji wielowyjściowej przedstawiono na rys. 4.16.



Rys. 4.16. Rozłączna dekompozycja funkcji wielowyjściowej

Fig. 4.16. Disjoint decomposition of a multi-output function

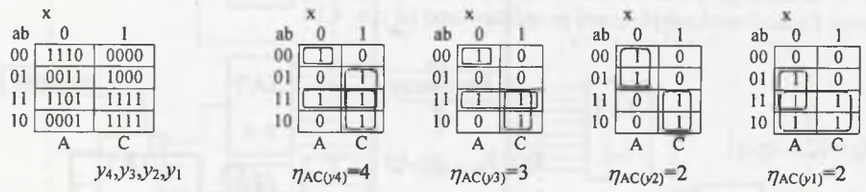
W przypadku funkcji wielowyjściowych komórki siatki Karnaugha zawierają m-bitowe wektory wyjściowe. Parę komórek (i, j) należących do tego samego wiersza siatki Karnaugha nazywamy niezgodną, jeżeli zawarte w tych komórkach m-bitowe wektory wyjściowe zawierają przynajmniej na jednej pozycji wartości (1,0) lub (0,1). Jeżeli w zbiorze par komórek należących do dwóch różnych kolumn wystąpi przynajmniej jedna para komórek niezgodnych, to takie kolumny nazywamy niezgodnymi.

Przedstawione w rozdziale 4.4.2 elementy kodowania wzorców kolumn mogę zostać bezpośrednio przeniesione do syntezy funkcji wielowyjściowych. Wyjaśnienia wymaga tylko sposób wyznaczania współczynnika pokrycia pary wzorców.

Przykład 4.16

Weźmy pod uwagę dwa przykładowe wzorce kolumn A i C oraz zmienną x przyjmującą wartość $x = 0$ dla wzorca A i $x = 1$ dla wzorca C.

Współczynnikiem pokrycia pary wzorców (A,C), oznaczanym jako η_{AC}, nazywamy sumę implikantów potrzebnych do pokrycia jednowyjściowych funkcji $f_{iAC} : B^{\overline{\overline{X_2}}+1} \to B$, gdzie ($i$=1,2,...,m), której argumentami są zmienne zbioru wolnego $\mathbf{X}_2$ oraz zmienna x (rys. 4.17).



$$\eta_{AC}=\eta_{AC(y4)}+\eta_{AC(y3)}+\eta_{AC(y2)}+\eta_{AC(y1)}=4+3+2+2=11$$

Rys.4.17. Sposób wyznaczenia przykładowego współczynnika pokrycia par wzorców
Fig. 4.17. The illustration how to find the exemplary coefficient of pattern pairs covering

Na uwagę zasługuje jeszcze sposób doboru podziału argumentów funkcji wielowyjściowej, zapewniający podział pierwotnego układu na podukłady o porównywalnej złożoności.

W przypadku funkcji wielowyjściowej kryterium doboru podziału musi uwzględniać nie tylko liczbę wejść bloku, lecz również liczbę jego wyjść. Wielkością charakteryzującą złożoność bloku może być iloczyn dwóch wartości: liczby wejść oraz liczby wyjść. Wyniki eksperymentów pokazały, że najlepiej złożoność układu po podziale określać na podstawie wyrażenia $\overline{\overline{X_1}}p+(\overline{\overline{X_2}}+p)m$.

Uwzględniając powyższe rozważania, istota dekompozycji kolumnowej funkcji wielowyjściowej sprowadza się do poszukiwania podziału argumentów funkcji, zapewniającego podział pierwotnego układu na podukłady o zbliżonej liczbie wejść, dla którego współczynnik złożoności $\overline{\overline{X_1}}p+(\overline{\overline{X_2}}+p)m$ przyjmuje wartość minimalną.

Algorytm dekompozycji kolumnowej funkcji wielowyjściowej, prowadzący do efektywnego podziału układu na blok związany i wolny, realizowane za pomocą k-iloczynowych bloków PAL

1. Minimalizacja funkcji $f : B^n \to B^m$ (Espresso-Dso); jeżeli w zbiorze m-funkcji istnieją funkcje $f_i : B^n \to B$, dla których $\Delta_{f_i} < 2k$, to są one realizowane metodą klasyczną; (dla uproszczenia dalszy ciąg algorytmu zostanie przedstawiony dla przypadku, gdy wszystkie wartości Δ_{f_i} ($i \in <1,m>$) są większe lub równe $2k$)
2. Wyznaczenie liczby k-iloczynowych bloków PAL potrzebnych do realizacji funkcji $f : B^n \to B^m$ metodą klasyczną $\delta_f^1 = \sum_{i=1}^{m} \delta_{f_i} = \sum_{i=1}^{m} \left(\left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 \right)$
3. Poszukiwanie odpowiedniej dekompozycji kolumnowej, tzn.:
   A. Rozpatrywane są podziały zbioru $\mathbf{I} = \{i_n,...,i_2,i_1\}$ na podzbiory $\mathbf{X}_1 = \{i_{n_z},...,i_2,i_1\}$ oraz $\mathbf{X}_2 = \{i_n,...,i_{n_z+2},i_{n_z+1}\}$ takie, że:
   jeżeli n jest liczbą nieparzystą, to $\overline{\overline{X_1}} = \overline{\overline{X_2}} + 1$;
   jeżeli n jest liczbą parzystą, to $\overline{\overline{X_1}} = \overline{\overline{X_2}} + 2$.
   B. Dla powyższych podziałów wyznaczane są złożoności kolumnowe, będące wynikiem algorytmu kolorowania grafu niezgodności kolumn, zorientowanego na realizację funkcji w strukturach typu PAL;
   C. Spośród rozpatrywanych podziałów wybierany jest ten, dla którego $p + m < \delta_f^1$, przy czym $p = \lceil \lg_2 \nu(\mathbf{X}_2 \mid \mathbf{X}_1) \rceil = \lceil \lg_2 \gamma(G) \rceil$ oraz liczba chromatyczna grafu niezgodności kolumn $\gamma(G)$ przyjmują wartość minimalną; jeżeli w grupie analizowanych podziałów taki podział nie istnieje, to analizowana funkcja realizowana jest metodą klasyczną;
   D. Jeżeli dla wybranego podziału zachodzi zależność $\overline{\overline{X_2}} + p > \overline{\overline{X_1}} + 1$, to analizowane są inne podziały z kolejnej grupy podziałów, dla której $\overline{\overline{X_1}} == \overline{\overline{X_1}} + 1$ i $\overline{\overline{X_2}} == \overline{\overline{X_2}} - 1$ itd. p. 3B;
   E. Spośród wybranych w kolejnych krokach podziałów wybierany jest ten, dla którego $\overline{\overline{X_1}}p + (\overline{\overline{X_2}} + p)m = \min$; jeżeli istnieje więcej podziałów, dla których $\overline{\overline{X_1}}p + (\overline{\overline{X_2}} + p)m = \min$, to wybierany jest ten podział, dla którego $\max(\overline{\overline{X_1}}; \overline{\overline{X_2}} + p) = \min$;
4. Dla wybranego podziału wykonywane jest kodowanie wzorców kolumn metodą przedstawioną w rozdziale 4.4.2 z uwzględnieniem rozszerzonego znaczenia współczynnika pokrycia pary wzorców przedstawionego w przykładzie 4.16;

5. Minimalizacja funkcji realizowanych w bloku związanym $g: B^{n_z} \to B^p$ (Espresso-Dso) oraz w bloku wolnym $f_w: B^{n-n_z+p} \to B^m$ (Espresso-Dso); wyznaczanie liczby bloków logicznych typu PAL, potrzebnych do realizacji układu po podziale; liczba ta oznaczana przez symbol δ_{zw} jest sumą liczby bloków potrzebnych do realizacji bloku związanego (δ_z) i wolnego (δ_w);

$$\delta_{zw} = \delta_z + \delta_w = \sum_{i=1}^{p}\left(\left\lceil \frac{z_i - k}{k-1} \right\rceil + 1\right) + \sum_{i=1}^{m}\left(\left\lceil \frac{w_i - k}{k-1} \right\rceil + 1\right)$$

6. Jeżeli $\delta_{zw} < \delta_f^1$, zatem kolejny etap dekompozycji kolumnowej to podział bloku związanego i wolnego;
   w przeciwnym przypadku:
   jeżeli m=1, to realizacja funkcji metodą klasyczną,
   jeżeli $m \geq 2$, to prosty podział wyjść [Kani99d] na dwie części m_1 i m_2 takie, że m_1=m_2 lub m_1=m_2+1; dobierany jest prosty podział wyjść, minimalizujący liczbę wejść uzyskiwanych podukładów (heurystyczny algorytm);
   następnie przeskok do p.1, tzn. analiza funkcji uzyskanych po podziale wyjść.

Przykład 4.17

Rozpatrzmy realizację układu testowego *rd73* [Bench] ($f: B^7 \to B^3$) korzystając z bloków logicznych typu PAL, zawierających 8 iloczynów (poniżej przedstawione są kolejne etapy poszukiwania rozwiązania).

- W wyniku minimalizacji $f: B^7 \to B^3$ (Espresso-Dso) uzyskiwane są wartości parametrów Δ_{f_1}=42; Δ_{f_2}=64; Δ_{f_3}=35.
- Wyznaczana jest liczba bloków logicznych typu PAL, zawierających 8-iloczynów, potrzebnych do realizacji funkcji $f: B^7 \to B^3$ metodą klasyczną:

$$\delta_f^1 = \sum_{i=1}^{3} \delta_{f_i} = \sum_{i=1}^{3}\left(\left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1\right) = 6+9+5 = 20$$

- Poszukiwana jest dekompozycja kolumnowa funkcji 3-wyjściowej, ponieważ wszystkie wartości Δ_{f_i} (i=1,2,3) są większe od 2k. W pierwszej kolejności rozpatrywane są rozłączne podziały argumentów, dla których $\overline{\overline{X}}_1 = 4$ oraz $\overline{\overline{X}}_2 = 3$. Dla wszystkich analizowanych podziałów w wyniku kolorowania grafu niezgodności kolumn uzyskiwana jest liczba chromatyczna $\gamma(G) = 5$. Znając liczbę chromatyczną wyznacza się liczbę wyjść bloku związanego $p = \lceil \lg_2 \gamma(G) \rceil = 3$. Liczba ta spełnia warunek konieczny opłacalności stosowania dekompozycji kolumnowej, tzn.: $p + m < \delta_f^1$.
- Dla wszystkich podziałów analizowanej grupy podziałów zachodzi nierówność $\overline{\overline{X}}_2 + p > \overline{\overline{X}}_1 + 1$, stąd poszukiwany jest kolejny podział prowadzący być może do podziału układu na podukłady o mniejszej złożoności. W tym celu analizowane są kolejne podziały, dla których $\overline{\overline{X}}_1 = 5$ i $\overline{\overline{X}}_2 = 2$. Ponownie dla wszystkich

analizowanych podziałów w wyniku kolorowania grafu niezgodności kolumn uzyskano identyczną liczbę chromatyczną $\gamma(G)$= 6. Liczba wyjść bloku związanego ($p = \lceil \lg_2 \gamma(G) \rceil$) ponownie wynosi 3 i spełnia warunek konieczny opłacalności stosowania dekompozycji kolumnowej, tzn.: $p + m < \delta_f^1$.

- Dla żadnego z podziałów analizowanej grupy podziałów nie zachodzi nierówność $\overline{\overline{X}}_2 + p > \overline{\overline{X}}_1 + 1$, stąd poszukiwania podziałów zostają zakończone.
- W wyniku poszukiwań wybrano dwa podziały, po jednym z każdej grupy, dla których:
  $\overline{\overline{X}}_1 = 4$; $\overline{\overline{X}}_2 = 3$; $\gamma(G)$= 5; p=3; $\overline{\overline{X}}_1 p + (\overline{\overline{X}}_2 + p)m$=12+18=30
  oraz
  $\overline{\overline{X}}_1 = 5$; $\overline{\overline{X}}_2 = 2$; $\gamma(G)$= 6; p=3; $\overline{\overline{X}}_1 p + (\overline{\overline{X}}_2 + p)m$=15+15=30
- Wartości wyrażeń $\overline{\overline{X}}_1 p + (\overline{\overline{X}}_2 + p)m$, charakteryzujących złożoność układu po podziale, dla dwóch wybranych podziałów są równe. Wybrany zostaje podział, dla którego $\overline{\overline{X}}_1 = 5$ oraz $\overline{\overline{X}}_2 = 2$, ponieważ $\max(\overline{\overline{X}}_1; \overline{\overline{X}}_2 + p)$ przyjmuje wartość minimalną.
- Dla wybranego podziału $X_2 = \{i_1, i_2\}$; $X_1 = \{i_3, i_4, i_5, i_6, i_7\}$ tworzącego 6 wzorców kolumn wyznaczane są współczynniki pokrycia wzorców oraz współczynniki pokrycia pary wzorców. Pozwala to przypisać słowa kodowe poszczególnym wzorcom. Wzorce kolumn, współczynniki pokrycia wzorców, tablica trójkątna zawierająca współczynniki pokrycia par wzorców oraz graf sąsiedztwa wzorców przedstawione są na rys. 4.18.

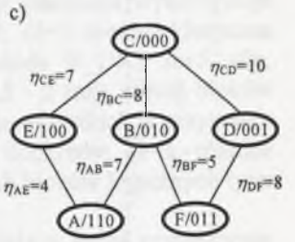
a)

X_2=$\{i_1, i_2\}$	X_1=$\{i_3, i_4, i_5, i_6, i_7\}$					
00	000	010	100	110	001	011
01	010	100	110	001	011	101
11	100	110	001	011	101	111
10	010	100	110	001	011	101
	A	B	C	D	E	F
	η_A=1	η_B=5	η_C=10	η_D=10	η_E=5	η_F=1

b)

B	7				
C	6	8			
D	8	7	10		
E	4	8	7	9	
F	8	5	9	8	8
	A	B	C	D	E

c)



Rys. 4.18. Elementy kodowania wzorców kolumn; a) wzorce kolumn z wyznaczonymi współczynnikami pokrycia wzorców; b) tablica trójkątna zawierająca współczynniki pokrycia par wzorców; c) graf sąsiedztwa wzorców

Fig. 4.18. Components of encoding process for the column patterns; a) column patterns with determined coefficients of pattern pairs covering; b) the triangle table with the coefficients of pattern pairs covering; c) the graph of pattern neighbourhood

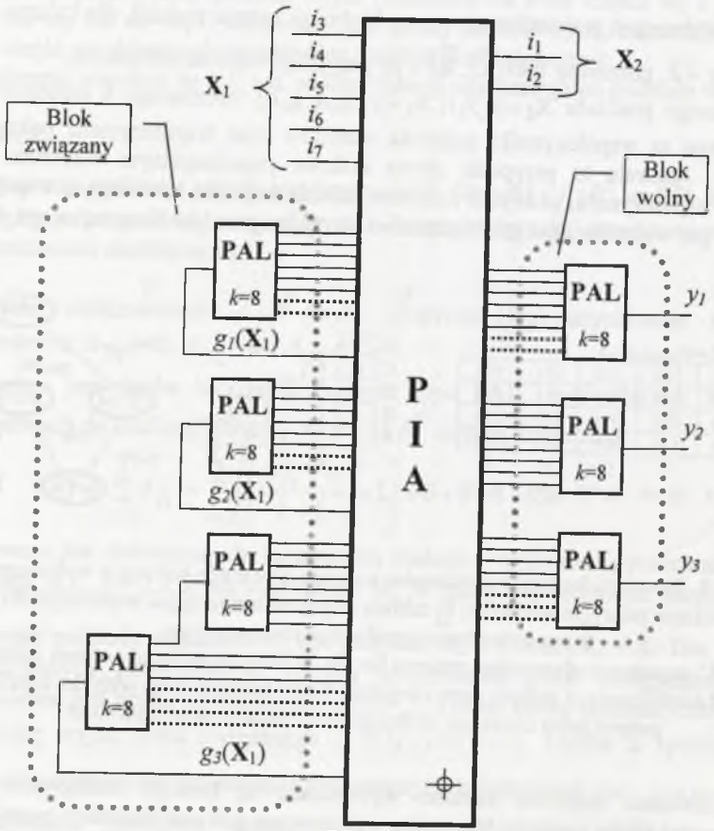
- Po zakodowaniu wzorców kolumn wyznaczane są funkcje realizowane w bloku związanym i bloku wolnym. Następnie wykonywana jest minimalizacja poszczególnych funkcji, tzn.: funkcji realizowanej w bloku związanym $g: B^5 \to B^3$ (Espresso-Dso) oraz funkcji realizowanej w bloku wolnym $f_w: B^5 \to B^3$ (Espresso-Dso).

- W wyniku minimalizacji uzyskiwane są współczynniki złożoności bloku związanego z_1=6, z_2=6, z_3=11 oraz bloku wolnego w_1=6, w_2=8, w_3=5, co pozwala wyznaczyć liczbę bloków logicznych typu PAL, potrzebnych do realizacji układu po podziale:

$$\delta_{zw} = \delta_z + \delta_w = \sum_{i=1}^{3}\left(\left\lceil \frac{z_i - k}{k-1} \right\rceil + 1\right) + \sum_{i=1}^{3}\left(\left\lceil \frac{w_i - k}{k-1} \right\rceil + 1\right) = 7$$

- Ponieważ $\delta_{zw} < \delta_f^1$ oraz dla wszystkich funkcji realizowanych w bloku wolnym i związanym $\Delta_{f_i} < 2k$, synteza układu testowego *rd73* zostaje zakończona.

Ostateczne rozwiązanie, wykorzystujące tylko 7 bloków logicznych typu PAL, zawierających 8 iloczynów, (13 bloków mniej niż realizacja metodą klasyczną), przedstawione jest na rys. 4.19.



PIA - *Programmable Interconnect Area*

Rys. 4.19. Wynik dekompozycji kolumnowej układu testowego *rd73*
Fig. 4.19. Results of column decomposition for the *rd73* benchmark

4.4.5. Wyniki eksperymentów

W celu zweryfikowania efektywności metody syntezy wykorzystującej dekompozycję kolumnową przeprowadzono syntezę analizowanych w pracy układów testowych (ang. benchmark) [Bench] dla struktur typu PAL, składających się z *k*-iloczynowych bloków PAL. Przedstawioną w poprzednim podrozdziale metodę realizacji wielowyjściowych funkcji w strukturach typu PAL (metoda W_DK) porównano pod względem liczby wykorzystywanych bloków logicznych oraz liczby warstw z metodą klasyczną z minimalizacją liczby warstw (metoda K_LW).

W pierwszej części tab. 4.3 przedstawione są wyniki syntezy układów testowych metodą klasyczną z minimalizacją liczby warstw (metoda K_LW). W kolumnach, w których występuje litera „B", podano liczbę wykorzystywanych *k*-iloczynowych bloków PAL, natomiast w kolumnach z literą „W" - liczbę warstw wynikowego układu. W drugiej części tabeli zawarte są wyniki syntezy uzyskiwane metodą W_DK. Symbole „?" oznaczają brak rozwiązania wynikający z długiego czasu syntezy (powyżej 15 minut; Celleron 900MHz). Algorytm syntezy wykorzystywany w metodzie W_DK w poszczególnych etapach poszukiwania odpowiedniej dekompozycji kolumnowej zawiera elementy porównania z metodą klasyczną. Zapewnia to znalezienie rozwiązania wykorzystującego nie większą liczbę bloków logicznych niż liczba bloków uzyskiwana metodą klasyczną z minimalizacją liczby warstw. Element porównania zawarty w metodzie W_DK wydłuża czas syntezy o około 2-3%, eliminując bardzo nieefektywne rozwiązania.

W zbiorze 226 porównywanych przypadków metoda W_DK doprowadziła do 90 (40%) rozwiązań wykorzystujących mniejszą liczbę bloków logicznych niż metoda K_LW. Dla pewnych układów testowych uzyskano znaczne zmniejszenie liczby wykorzystywanych bloków logicznych, np.: *rd84* (*k*=3; metoda klasyczna wykorzystuje około 6 razy więcej bloków logicznych niż metoda W_DK), *rd73*, (*k*=3; metoda klasyczna wykorzystuje około 4,5 razy więcej bloków logicznych niż metoda W_DK), *xor5, clip, rd53, 5xp1, f51m* (*k*=3; metoda klasyczna wykorzystuje około 1,5 - 2 razy więcej bloków logicznych niż metoda W_DK). Znaczne różnice występują przede wszystkich w przypadku wykorzystywania bloków logicznych zawierających niewiele iloczynów (*k*=3 - różnice największe). Nieco mniejsze korzyści uzyskiwane są dla większych bloków logicznych typu PAL.

Niestety ze zmniejszeniem liczby bloków logicznych nie idzie w parze zmniejszenie liczby warstw. Wśród układów testowych, dla których w stosunku do metody klasycznej uzyskano zmniejszenie liczby wykorzystywanych bloków, w około 66% przypadków rozwiązania uzyskiwane metodą W_DK zawierają większą liczbę warstw logicznych. Okazuje się jednak, że redukcja liczby bloków logicznych nie zawsze pociąga za sobą ekspansję liczby warstw logicznych, np.: *rd84*, *rd73* (*k*>4; znaczne zmniejszenie liczby bloków logicznych, przy nie zmienionej liczbie warstw).

Podstawową wadą metody W_DK jest złożoność zaproponowanych algorytmów syntezy, co praktycznie ogranicza jej wykorzystanie do układów zawierających maksymalnie kilkanaście wejść.

Podsumowaniem powyższych rozważań mogą być wykresy uwypuklające mocną i słabą stronę zaproponowanej metody syntezy wykorzystującej dekompozycję kolumnową. Wartości przedstawione na wykresie 4.1 wyznaczone zostały z zależności

Tabela 4.3

Porównanie klasycznej realizacji funkcji z minimalizacją liczby warstw (metoda K_LW) z zaproponowaną metodą opartą na dekompozycji kolumnowej funkcji wielowyjściowej (metoda W_DK)

metoda W_DK - metoda realizacji **W**ielowyjściowej funkcji w strukturach typu PAL wykorzystująca **D**ekompozycję **K**olumnową
metoda K_LW - **K**lasyczna metoda realizacji funkcji w strukturach typu PAL z minimalizacją **L**iczby **W**arstw
i - liczba wejść układu testowego; o - liczba wyjść układu testowego; B - liczba bloków; W - liczba warstw

K_LW

	Układ testowy	i	o	k=3 B	k=3 W	k=4 B	k=4 W	k=5 B	k=5 W	k=6 B	k=6 W	k=7 B	k=7 W	k=8 B	k=8 W	k=12 B	k=12 W	k=16 B	k=16 W
1	5xp1	7	10	35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2
2	9sym	9	1	43	5	29	4	22	3	18	3	15	3	13	3	8	2	6	2
3	9symml	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2
4	b12	15	9	24	3	16	2	14	2	13	2	10	2	10	2	9	1	9	1
5	bw	5	28	49	2	38	2	30	2	28	1	28	1	28	1	28	1	28	1
6	clip	9	5	73	4	49	3	38	3	30	3	26	2	22	2	14	2	12	2
7	con1	7	2	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1
8	duke2	22	29	99	3	72	3	61	2	49	2	44	2	41	2	35	2	31	2
9	ex1010	10	10	242	4	165	3	124	3	101	3	85	3	74	2	49	2	38	2
10	f51m	8	8	37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2
11	inc	7	9	23	3	17	2	16	2	13	2	11	2	11	2	9	1	9	1
12	ldd	9	19	34	2	25	2	25	2	21	2	19	1	19	1	19	1	19	1
13	misex1	8	7	14	2	12	2	8	2	7	1	7	1	7	1	7	1	7	1
14	misex2	25	18	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1
15	pcle	19	9	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1
16	rd53	5	3	15	3	11	2	8	2	6	2	6	2	6	2	4	2	3	1
17	rd73	7	3	70	4	47	3	36	3	29	3	24	3	20	2	14	2	11	2
18	rd84	8	4	142	5	95	4	72	4	58	3	49	3	42	3	28	2	21	2
19	root	8	5	35	4	24	3	18	3	16	2	13	2	11	2	9	2	7	2
20	sao2	10	4	36	3	24	3	19	2	15	2	14	2	11	2	7	2	7	2
21	sqn	7	3	21	3	15	3	11	2	10	2	8	2	7	2	5	2	4	2
22	sqr6	6	12	31	3	24	2	20	2	16	2	16	2	15	2	12	2	11	1
23	squar5	5	8	14	2	11	2	9	2	9	2	9	2	8	1	8	1	8	1
24	table3	14	14	264	4	181	4	135	3	110	3	94	3	80	3	53	2	40	2
25	vg2	25	8	53	4	37	3	28	3	22	3	20	2	19	2	13	2	11	2
26	x2	10	7	13	2	10	2	10	2	7	1	7	1	7	1	7	1	7	1
27	xor5	5	1	8	2	5	2	4	2	3	2	3	2	3	2	2	2	1	1
28	z4m1	7	4	29	4	19	3	15	3	13	2	11	2	9	2	7	2	5	2
29	z5xp1	7	10	37	3	27	3	23	2	19	2	17	2	15	2	12	2	11	2
30	z9sym	9	1	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2
	Suma bloków/warstw			1406	89	984	71	772	64	641	58	561	56	501	53	414	48	357	45

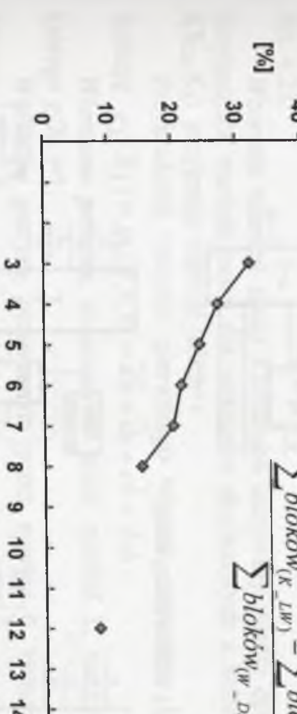
W_DK

	Układ testowy	k=3 B	k=3 W	k=4 B	k=4 W	k=5 B	k=5 W	k=6 B	k=6 W	k=7 B	k=7 W	k=8 B	k=8 W	k=12 B	k=12 W	k=16 B	k=16 W
1	5xp1	24	5	19	3	18	3	15	2	14	2	14	2	12	2	11	2
2	9sym	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3
3	9symml	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3
4	b12	24	3	16	2	14	2	13	2	10	2	10	2	9	1	9	1
5	bw	49	3	38	2	30	2	28	1	28	1	28	1	28	1	28	1
6	clip	47	4	32	4	28	4	24	4	19	3	18	3	13	4	10	3
7	con1	4	2	3	2	2	1	2	1	2	1	2	1	2	1	2	1
8	duke2	?	?	?	?	?	?	?	?	?	?	?	?	35	2	31	2
9	ex1010	242	4	165	3	124	3	101	3	85	3	74	2	49	2	38	2
10	f51m	24	5	20	4	16	4	13	3	13	3	14	3	11	2	10	2
11	inc	23	3	17	2	16	2	13	2	11	2	11	2	9	1	9	1
12	ldd	34	2	25	2	25	2	21	2	19	1	19	1	19	1	19	1
13	misex1	14	2	12	2	8	2	7	1	7	1	7	1	7	1	7	1
14	misex2	19	2	19	2	18	1	18	1	18	1	18	1	18	1	18	1
15	pcle	10	2	9	1	9	1	9	1	9	1	9	1	9	1	9	1
16	rd53	9	4	7	4	5	3	5	3	5	3	5	3	4	2	3	1
17	rd73	16	4	12	4	10	3	9	3	8	3	7	3	6	2	6	2
18	rd84	23	6	17	5	13	4	12	3	11	3	11	3	9	2	7	2
19	root	35	4	24	3	18	3	16	2	13	2	11	2	9	2	7	2
20	sao2	30	5	23	4	19	4	11	3	10	3	10	3	7	2	7	2
21	sqn	21	3	15	3	11	2	10	2	8	2	7	2	5	2	4	2
22	sqr6	31	3	24	2	20	2	16	2	16	2	15	2	12	2	11	1
23	squar5	14	3	11	2	9	2	9	2	9	2	8	1	8	1	8	1
24	table3	264	4	181	4	135	3	110	3	94	3	80	3	53	2	40	2
25	vg2	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
26	x2	13	2	10	2	10	2	7	1	7	1	7	1	7	1	7	1
27	xor5	4	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1
28	z4m1	12	6	11	4	10	4	9	4	7	2	7	2	6	2	5	2
29	z5xp1	24	5	19	3	18	3	15	2	15	2	14	2	12	2	11	2
30	z9sym	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3
	Suma bloków/warstw	1064	115	773	86	621	79	528	70	467	66	435	61	382	53	333	49

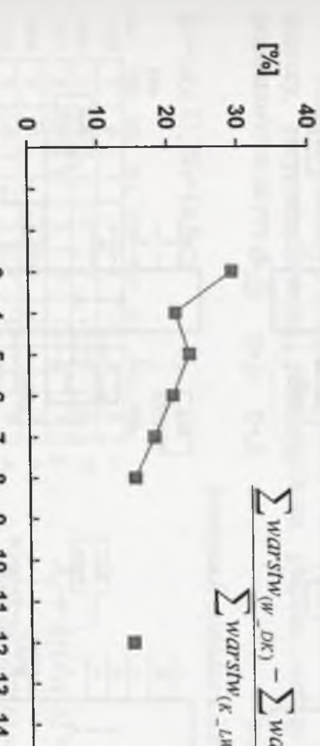
$\frac{\sum bloków_{(K_LW)} - \sum bloków_{(W_DK)}}{\sum bloków_{(W_DK)}}$ 100%, gdzie $\sum bloków_{(K_LW)}$ i $\sum bloków_{(W_DK)}$ oznaczają odpowiednią sumę liczby bloków, wykorzystywanych poszczególnymi metodami syntezy, przedstawioną w tab. 4.3.

Podobnie wyznaczone zostały wartości przedstawione na wykresie 4.2 na podstawie zależności $\frac{\sum warstw_{(W_DK)} - \sum warstw_{(K_LW)}}{\sum warstw_{(K_LW)}}$ 100%, gdzie $\sum warstw_{(K_LW)}$ i $\sum warstw_{(W_DK)}$ oznaczają odpowiednie sumy liczby warstw logicznych, dla poszczególnych wartości *k*.

Wykres 4.1. Porównanie metody syntezy W_DK z metodą K_LW pod względem liczby bloków (opis w tekście)



Wykres 4.2. Porównanie metody syntezy W_DK z metodą K_LW pod względem liczby warstw logicznych (opis w tekście)



Analiza wyników pozwala stwierdzić, że uzyskana metodą W_DK redukcja liczby bloków pociąga za sobą procentowo podobną ekspansję liczby warstw logicznych. Metoda W_DK jest skuteczniejsza dla struktur zbudowanych z niewielkich bloków logicznych typu PAL. Dla większych bloków uzyskuje się procentowo mniejszą redukcję liczby bloków, ale również procentowo mniejszą ekspansję liczby warstw logicznych.

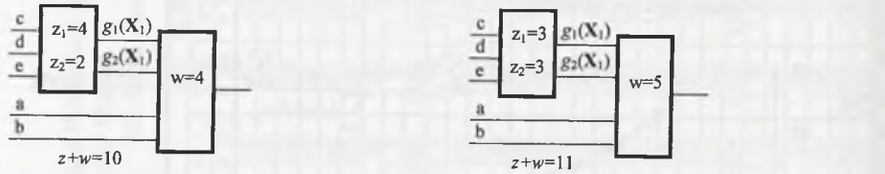
4.5. Dekompozycja wierszowa zorientowana na realizację w strukturach typu PAL

Istota dekompozycji kolumnowej zorientowanej na realizację układów w strukturach typu PAL tkwi w minimalizacji sumarycznej liczby wykorzystywanych iloczynów. Specyfika bloku logicznego typu PAL sprawia, że minimalizacja sumarycznej liczby iloczynów nie zawsze idzie w parze z minimalizacją liczby bloków logicznych typu PAL.

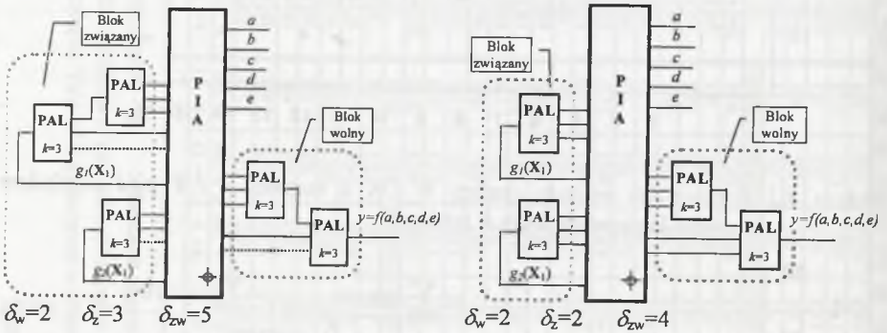
Przykład 4.18

Rozpatrzmy realizację dwóch układów uzyskiwanych w wyniku dekompozycji kolumnowej, przedstawionych na rys. 4.20.a.

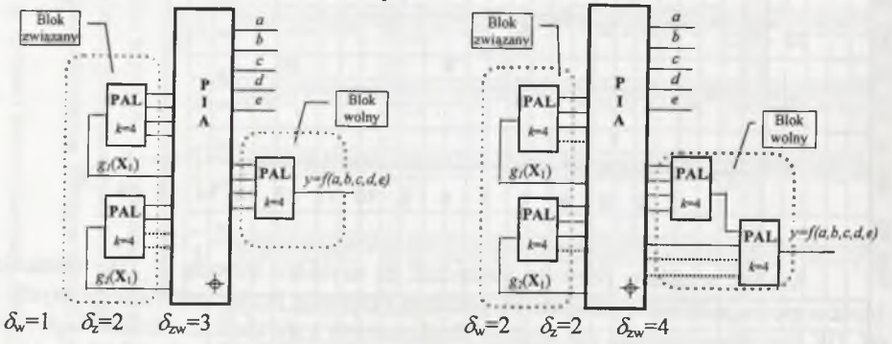
a)



b) bloki logiczne typu PAL z trzema iloczynami



c) bloki logiczne typu PAL z czterema iloczynami



Rys. 4.20. Realizacja układów za pomocą bloków logicznych typu PAL

Fig. 4.20. Implementation of digital circuits by means of PAL-based logic blocks

Realizacja układu pierwszego (z lewej strony), który wymaga użycia mniejszej liczby iloczynów, wykorzystuje większą liczbę bloków logicznych typu PAL, zawierających 3 iloczyny (rys.4.20.b). W przypadku bloków zawierających 4 iloczyny sytuacja jest odwrotna (rys.4.20.c).

Przedstawiony przykład jednoznacznie pokazuje, że uzyskiwana w dekompozycji kolumnowej minimalizacja sumarycznej liczby iloczynów nie zawsze idzie w parze z minimalizacją liczby bloków. Cenne byłoby wplecenie w etap dekompozycji elementu dopasowania realizowanych funkcji do struktury bloków logicznych typu PAL. Takie zadanie spełnia dekompozycja wierszowa, będąca przedmiotem rozważań dalszej części niniejszego rozdziału. Wprowadźmy, korzystając z prostego przykładu, niezbędne określenia.

Przykład 4.19

Rozpatrzmy funkcję $f: B^6 \rightarrow B$, której siatka Karnaugha przedstawiona jest na rys. 4.21.

Wzorcem wiersza siatki Karnaugha nazywany wiersz opisany przez funkcję $g(\mathbf{X}_1)$, przyjmującą wartość jeden dla wektorów skojarzonych z kolumnami, dla których funkcja $f(\mathbf{X}_2, \mathbf{X}_1)$ przyjmuje wartość jeden.

Przykładowo wzorzec pierwszego wiersza oznaczony literą A opisany jest przez funkcję $g_1(\mathbf{X}_1) = g_1(d,e,h) = \overline{d}\overline{h} + \overline{d}e + e\overline{h} + d\overline{e}h$.

Wierszem pełnym, oznaczanym przez symbol **1**, nazywamy wzorzec wiersza, dla którego $g(\mathbf{X}_1)$=1.

Wierszem pustym, oznaczanym przez symbol **0**, nazywamy wzorzec wiersza, dla którego $g(\mathbf{X}_1)$=0.

Dopełnieniem wzorca wiersza A, oznaczanym jako $\overline{A}$, nazywamy wzorzec wiersza opisany przez funkcję $\overline{g}(\mathbf{X}_1)$.

Analiza wierszy przykładowej siatki Karnaugha prowadzi do wyznaczenia wzorców wierszy, przyporządkowując poszczególnym grupom wierszy odpowiednie symbole, przedstawione na rys. 4.21.

$\mathbf{X}_1=\{d,e,h\}$, $\mathbf{X}_2=\{a,b,c\}$

abc \ *Deh*	000	001	011	010	110	111	101	100	
000	1	0	1	1	1	0	1	0	A
001	1	0	1	0	1	1	1	0	B
011	0	1	0	0	0	1	0	1	$\overline{A}$
010	1	0	1	1	1	0	1	0	A
110	1	0	1	1	1	0	1	0	A
111	1	0	1	0	1	1	1	0	B
101	0	0	0	0	0	0	0	0	**0**
100	1	1	1	1	1	1	1	1	**1**

$A \rightarrow g_1(d,e,h) = \overline{d}\overline{h} + \overline{d}e + e\overline{h} + d\overline{e}h$

$B \rightarrow g_2(d,e,h) = eh + de + dh + \overline{d}\overline{e}\overline{h}$

Rys. 4.21. Siatka Karnaugha z symbolami rozróżniającymi poszczególne wzorce wierszy

Fig. 4.21. The Karnaugh map with symbols, which distinguish individual row patterns

W ogólnym przypadku zbiór wierszy można podzielić na kilka grup:

- wiersze puste,
- wiersze pełne,
- wiersze skojarzone z identycznym wzorcem wiersza lub jego dopełnieniem (np. A lub $\overline{A}$).

Złożonością wierszową matrycy podziałów $\mu(\mathbf{X}_2|\mathbf{X}_1)$ nazywamy liczbę różnych grup wierszy, z wyłączeniem grup zawierających wiersze puste i wiersze pełne. Przykładowo dla funkcji przedstawionej na rys. 4.21 złożoność wierszowa $\mu(\mathbf{X}_2|\mathbf{X}_1)=2$.

Twierdzenie o dekompozycji wierszowej

Jeżeli złożoność wierszowa matrycy podziałów $\mu(\mathbf{X}_2|\mathbf{X}_1)=p$, to funkcja $f(\mathbf{X}_2, \mathbf{X}_1)$ może być przedstawiona w postaci:

$H_0(\mathbf{X}_2)+H_1(\mathbf{X}_2)g_1(\mathbf{X}_1)+H_1'(\mathbf{X}_2)\overline{g}_1(\mathbf{X}_1)+\ldots+H_p(\mathbf{X}_2)g_p(\mathbf{X}_1)+H_p'(\mathbf{X}_2)\overline{g}_p(\mathbf{X}_1)$ tzn.:

$$\mu(\mathbf{X}_2|\mathbf{X}_1)=p \Rightarrow f(\mathbf{X}_2,\mathbf{X}_1)=H_0(\mathbf{X}_2)+\sum_{i=1}^{p}\left[H_i(\mathbf{X}_2)g_i(\mathbf{X}_1)+H_i'(\mathbf{X}_2)\overline{g}_i(\mathbf{X}_1)\right] \qquad (4.8)$$

Dowód:

Jeżeli $\mu(\mathbf{X}_2|\mathbf{X}_1)=p$, to zbiór wierszy można podzielić na p+2-grup:

- grupę zawierającą wiersze puste,
- grupę zawierającą wiersze pełne,
- p-grup zawierających wiersze skojarzone z jednym wzorcem wiersza lub jego dopełnieniem.

Każdej grupie wierszy można przyporządkować funkcję, której argumentami są zmienne zbioru wolnego. Niech

- $H_0(\mathbf{X}_2)=\begin{cases}1, \text{ dla wierszy pełnych}\\ 0, \text{ dla pozostałych wierszy}\end{cases}$
- $H_i(\mathbf{X}_2)=\begin{cases}1, \text{ dla wierszy, dla których wzorzec wiersza jest opisany przez funkcję } g_i(\mathbf{X}_1)\\ 0, \text{ dla pozostałych wierszy}\end{cases}$

  gdzie i=1,2,...,p
- $H_i'(\mathbf{X}_2)=\begin{cases}1, \text{ dla wierszy, dla których wzorzec wiersza jest opisany przez funkcję } \overline{g}_i(\mathbf{X}_1)\\ 0, \text{ dla pozostałych wierszy}\end{cases}$

  gdzie i=1,2,...,p

Zgodnie z powyższymi oznaczeniami funkcja $f(\mathbf{X}_2, \mathbf{X}_1)$ może być przedstawiona w postaci:

$$f(\mathbf{X}_2,\mathbf{X}_1)=H_0(\mathbf{X}_2)+\sum_{i=1}^{p}\left[H_i(\mathbf{X}_2)g_i(\mathbf{X}_1)+H_i'(\mathbf{X}_2)\overline{g}_i(\mathbf{X}_1)\right]$$

cbdo.

Przykład 4.20

Rozpatrzmy ponownie funkcję z przykładu 4.19. Wyznaczmy dla rozważanego podziału zmiennych ($\mathbf{X}_1=\{d,e,h\}$, $\mathbf{X}_2=\{a,b,c\}$) funkcje $H_i(\mathbf{X}_2)$. Funkcje te przyjmują następujące postacie:

- $H_0(\mathbf{X}_2)=\overline{abc}$
- $H_1(\mathbf{X}_2)=\overline{ac}+b\overline{c}$; $\quad H_1'(\mathbf{X}_2)=\overline{abc}$
- $H_2(\mathbf{X}_2)=\overline{ab}c+abc$; $\quad H_2'(\mathbf{X}_2)=0$

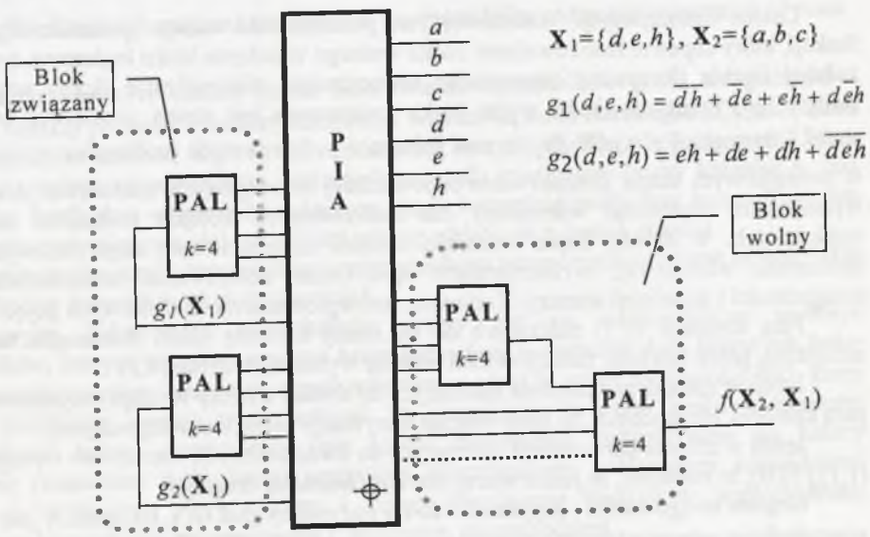
Znając funkcje $H_i(\mathbf{X}_2)$ oraz funkcje $g_i(\mathbf{X}_1)$ (rys. 4.21), możliwe jest, korzystając z zależności (4.8), przedstawienie funkcji $f(\mathbf{X}_2, \mathbf{X}_1)$ w następującej postaci:

$$f(\mathbf{X}_2,\mathbf{X}_1)=H_0(\mathbf{X}_2)+H_1(\mathbf{X}_2)g_1(\mathbf{X}_1)+H_1'(\mathbf{X}_2)\overline{g}_1(\mathbf{X}_1)+H_2(\mathbf{X}_2)g_2(\mathbf{X}_1)+H_2'(\mathbf{X}_2)\overline{g}_2(\mathbf{X}_1)=$$
$$=\overline{abc}+\left(\overline{ac}+b\overline{c}\right)\left(\overline{dh}+\overline{de}+e\overline{h}+d\overline{eh}\right)+\overline{abc}\left(\overline{dh}+\overline{de}+e\overline{h}+d\overline{eh}\right)+\left(\overline{ab}c+abc\right)\left(eh+de+dh+\overline{deh}\right),$$

tzn.:

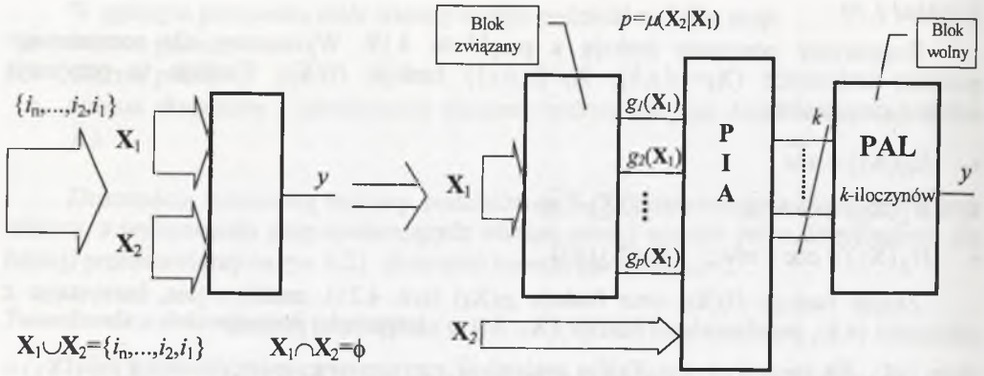
$$f(\mathbf{X}_2,\mathbf{X}_1)=\overline{abc}+\left(\overline{ac}+b\overline{c}\right)g_1(\mathbf{X}_1)+\overline{abc}\,\overline{g}_1(\mathbf{X}_1)+\left(\overline{ab}c+abc\right)g_2(\mathbf{X}_1)$$

Realizacja funkcji wykorzystująca bloki logiczne typu PAL przedstawiona jest na rys. 4.22.



Rys. 4.22. Realizacja funkcji oparta na dekompozycji wierszowej
Fig. 4.22. Implementation of the function based on the row decomposition

Istota dekompozycji wierszowej tkwi w poszukiwaniu takiego podziału, który zapewni realizowalność bloku wolnego w jednym bloku logicznym typu PAL o zadanej liczbie iloczynów. Podział układu będący wynikiem takiej dekompozycji wierszowej przedstawia rys. 4.23.



Rys.4.23. Idea podziału układu bazująca na dekompozycji wierszowej
Fig. 4.23. Circuit partitioning concept based on the row decomposition

4.5.1. Metoda wyznaczania złożoności wierszowej

Celem dekompozycji wierszowej jest poszukiwanie takiego podziału argumentów funkcji, który zapewni realizowalność bloku wolnego w jednym bloku logicznym typu PAL o zadanej liczbie iloczynów, zapewniając równocześnie minimalizację liczby wyjść bloku związanego. Ponieważ liczba wyjść bloku związanego jest równa złożoności wierszowej siatki Karnaugha ($p = \mu(\mathbf{X}_2|\mathbf{X}_1)$), stąd jednym z podstawowych problemów występujących w początkowym etapie poszukiwania odpowiedniej dekompozycji wierszowej jest problem wyznaczenia złożoności wierszowej dla analizowanych kolejno podziałów zmiennych wejściowych. W dalszej części rozdziału zostanie zaprezentowany algorytm wyznaczania złożoności wierszowej, wykorzystujący specyficzne kolorowanie wierzchołków grafu niezgodności i dopełnień wierszy. Konieczne jest wprowadzenie dodatkowych pojęć.

Parę komórek (i, j) należących do tej samej kolumny siatki Karnaugha nazywamy niezgodną, jeżeli wartości funkcji w nich zawarte wynoszą (1,0) lub (0,1).

Jeżeli w zbiorze par komórek należących do dwóch wierszy wystąpi przynajmniej jedna para komórek niezgodnych, to takie wiersze nazywamy ***wierszami niezgodnymi***.

Jeżeli w zbiorze par komórek należących do dwóch różnych wierszy nie występują pary (1,1) i (0,0), to mówimy, że jeden wiersz jest ***dopełnieniem*** drugiego.

Grafem niezgodności i dopełnień wierszy nazywamy graf G(**Y**,**U**), gdzie **Y** jest zbiorem wierzchołków odpowiadających wierszom, natomiast $\mathbf{U}=\mathbf{U}_n\cup\mathbf{U}_d$ zbiorem krawędzi, gdzie:

$\mathbf{U}_n$ - jest zbiorem krawędzi łączących wierzchołki odpowiadające wierszom niezgodnym z wyłączeniem wierszy, w których jeden jest dopełnieniem drugiego oraz wierszy pustych i pełnych,

$\mathbf{U}_d$ - jest zbiorem krawędzi łączących wierzchołki odpowiadające wierszom, w których jeden jest dopełnieniem drugiego z wyłączeniem wierszy pustych i pełnych.

Krawędzie należące do zbioru $\mathbf{U}_n$ będą oznaczane linią ciągłą, natomiast krawędzie należące do zbioru $\mathbf{U}_d$ linią przerywaną.

Przykład 4.21

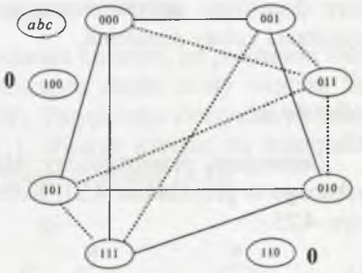
Wyznaczmy graf niezgodności i dopełnień wierszy dla funkcji $f: B^6 \rightarrow B$ przedstawionej na rys. 4.24.a. W pierwszej kolejności wykrywane są wiersze puste i pełne, które są eliminowane z dalszych rozważań. W przypadku analizowanej funkcji występują tylko dwa wiersze puste, oznaczone na rys. 4.24a) obok wiersza symbolem **0**.

Analiza poszczególnych par wierszy pozwala utworzyć graf niezgodności i dopełnień wierszy przedstawiony na rys. 4.24b).

a) $\mathbf{X}_1=\{d,e,h\}$, $\mathbf{X}_2=\{a,b,c\}$

abc \ deh	000	001	011	010	110	111	101	100	
000	0	1	0	1	0	-	1	0	
001	1	-	1	0	-	0	1	0	
011	-	0	-	-	-	1	0	1	
010	0	1	0	1	0	-	-	0	
110	-	0	-	0	-	0	-	0	0
111	0	-	-	1	0	1	0	1	
101	-	1	1	0	1	0	-	-	
100	-	0	-	-	-	-	0	-	0

b)



Rys. 4.24. Siatka Karnaugha i graf niezgodności i dopełnień wierszy

Fig.4.24. The Karnaugh map and the graph of incompatibility and complementation of rows

Zaproponowane w dalszej części kolorowanie wierzchołków grafu niezgodności i dopełnień wierszy pozwala wyznaczyć złożoność wierszową siatki Karnaugha $\mu(\mathbf{X}_2|\mathbf{X}_1)$, która jest równa liczbie kolorów użytych do rozróżnienia wszystkich wierzchołków grafu. Istota kolorowania polega na przypisaniu wierzchołkom grafu minimalnej liczby kolorów w taki sposób, aby każde dwa wierzchołki połączone ciągłą krawędzią miały inny kolor. Algorytm kolorowania grafu niezgodności i dopełnień wierszy składa się z dwóch etapów.

W pierwszym etapie poszczególnym wierzchołkom przyporządkowywane są wszystkie możliwe kolory dozwolone i kolory dopełnień.

Algorytm kolorowania wierzchołków oparty jest na sekwencyjnym wyborze wierzchołków, którym przypisywany jest ***kolor dozwolony*** (oznaczany dużą literą) lub ***kolor dopełnienia*** (oznaczany dużą literą z podkreśleniem u góry) w miarę możliwości taki, który jest już przypisany innemu wierzchołkowi. Po przypisaniu w i-tym kroku wybranemu wierzchołkowi koloru dozwolonego lub koloru dopełnienia, przypisywane są kolory zabronione (oznaczane małą literą) wszystkim wierzchołkom, połączonym krawędziami ciągłymi z wybranym wierzchołkiem i kolory dopełnienia wszystkim wierzchołkom, połączonym krawędziami przerywanymi z wybranym wierzchołkiem.

Wybór i-tego wierzchołka prowadzony jest zgodnie z następującymi zasadami:

- wybierany jest wierzchołek z maksymalną liczbą kolorów zabronionych; przyporządkowywany jest mu kolor dozwolony (jeżeli jest to możliwe, to kolor ze zbioru już użytych kolorów);
- spośród wierzchołków z identyczną liczbą kolorów zabronionych wybierany jest ten, do którego dochodzi maksymalna liczba krawędzi;
- spośród wierzchołków z identyczną liczbą kolorów zabronionych i maksymalną liczbą krawędzi wybierany jest ten, który ma dodatkowo maksymalną liczbę kolorów dopełnień (jeżeli jest to możliwe, przyporządkowywany jest mu kolor dopełnienia);

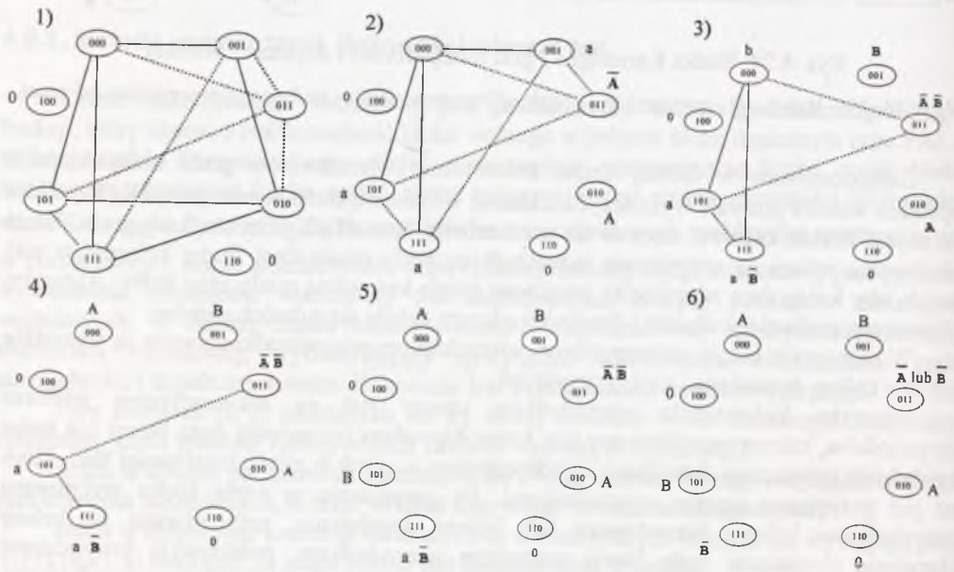
- spośród wierzchołków z identyczną liczbą kolorów zabronionych, kolorów dopełnień i krawędzi wybierany jest ten, do którego dochodzi maksymalna liczba krawędzi ciągłych.

Po wybraniu wierzchołka i przypisaniu odpowiednim wierzchołkom kolorów dozwolonych, kolorów dopełnień i kolorów zabronionych wykonywana jest redukcja grafu, polegająca na eliminacji krawędzi łączących wybrany wierzchołek z innymi wierzchołkami grafu, po czym wybierany jest kolejny (i+1) wierzchołek, tym razem po analizie grafu zredukowanego.

W ostatnim kroku pierwszego etapu wierzchołkom, które mają kolory zabronione i kolory dopełnień, przypisywane są wszystkie możliwe, występujące na grafie kolory dozwolone i kolory dopełnień.

Przykład 4.22

Zastosujmy przedstawiony algorytm kolorowania wierzchołków grafu dla grafu uzyskanego w przykładzie 4.21. Kolejne etapy kolorowania wierzchołków przedstawione są na rys. 4.25.



Rys.4.25. Kolejne etapy kolorowania wierzchołków grafu niezgodności i dopełnień wierszy

Fig.4.25. Step-by-step colouring of the incompatibility and complementation of graph

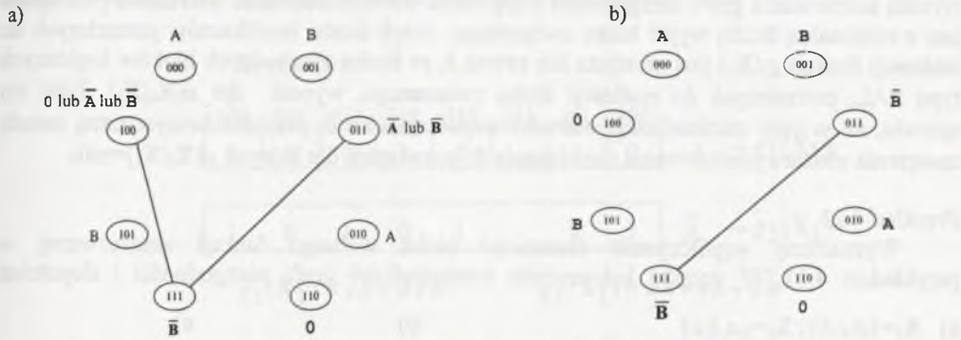
Celem drugiego etapu kolorowania grafu jest przypisanie jednego koloru (dozwolonego lub dopełnienia) wierzchołkom, które mają więcej niż jeden kolor - dozwolony lub dopełnienia. W tym celu tworzony jest graf zgodności kolorów, który powstaje przez połączenie wierzchołków mających więcej niż jeden kolor (dozwolony lub dopełnienia) z pozostałymi wierzchołkami z identycznym, jednym kolorem dozwolonym lub dopełnienia (tzn. wierzchołek z kolorami „B i C" łączy się z wierzchołkami mającymi tylko kolor „B" i

tylko kolor „C"; wierzchołek z kolorami dopełnienia „$\overline{B}$ i $\overline{C}$" łączy się z wierzchołkami mającymi tylko kolor dopełnienia „$\overline{B}$" i tylko kolor „$\overline{C}$"). Dodatkowo rozpatrywane są wszystkie wiersze pełne i puste, którym przypisywane są wszystkie możliwe kolory dozwolone i kolory dopełnienia.

Po utworzeniu grafu zgodności kolorów podlega on redukcji zgodnie z następującymi regułami:

- wybierany jest wierzchołek, który ma więcej niż jeden kolor (dozwolony lub dopełnienia),
- eliminowane są krawędzie łączące wybrany wierzchołek z wierzchołkami, których odległość kodowa jest większa od 1 od kodu wybranego wierzchołka.

Wynikiem tego etapu jest zredukowany graf zgodności kolorów, na podstawie którego przypisywane są kolory wierzchołkom, które po pierwszym etapie miały większą liczbę kolorów dozwolonych i kolorów dopełnienia (rys. 4.26). Przypisanie identycznych kolorów wierzchołkom, dla których odległość kodowa wynosi 1, stwarza warunki do minimalizacji liczby iloczynów wykorzystywanych w bloku wolnym (patrz zależność (4.8)).



Rys. 4.26. Drugi etap kolorowania; a) graf zgodności kolorów, b) zredukowany graf zgodności kolorów z przypisanymi wierzchołkom kolorami

Fig. 4.26. The second step of the colouring process; a) graph of colour compatibility; b) reduced graph of colour compatibility with fixed colours assigned to the nodes

Zaproponowana metoda analizy grafu zgodności kolorów wplata w proces kolorowania element wpływający na minimalizację liczby iloczynów wykorzystywanych w bloku wolnym, powstającym w wyniku dekompozycji wierszowej. W ten sposób w jeden z pierwszych etapów dekompozycji wierszowej wpleciony został element heurystycznej minimalizacji liczby użytych iloczynów.

4.5.2. Zagadnienia związane w wyborem dekompozycji wierszowej

Istota dekompozycji wierszowej polega na takim podziale układu, który zapewnia realizowalność bloku wolnego w jednym bloku logicznym typu PAL o zadanej liczbie iloczynów. Nasuwa się pytanie, jak znaleźć odpowiednią dekompozycję wierszową, zakładając realizację na blokach logicznych o zadanej liczbie iloczynów.

Efektywny podział układu pierwotnego na blok związany i wolny, będący wynikiem dekompozycji wierszowej, powinien spełniać dwa warunki:

- współczynnik złożoności bloku wolnego powinien być mniejszy lub równy zadanej liczbie iloczynów zawartych w bloku logicznym typu PAL ($w \le k$),
- blok związany powinien być realizowany na minimalnej liczbie bloków logicznych typu PAL o zadanej liczbie iloczynów (δ_z=min).

O ile pierwszy warunek ($w \le k$) można stosunkowo łatwo sprawdzić, poszukiwanie rozwiązania, dla którego δ_z=min, jest trudne i w ogólnym przypadku nierozwiązywalne.

Dla danego podziału argumentów funkcja $f(\mathbf{X}_2,\mathbf{X}_1)$ może być przedstawiona w postaci: $f(\mathbf{X}_2,\mathbf{X}_1)=H_0(\mathbf{X}_2)+\sum_{i=1}^{\mu(\mathbf{X}_2|\mathbf{X}_1)}\left[H_i(\mathbf{X}_2)g_i(\mathbf{X}_1)+H_i'(\mathbf{X}_2)\bar{g}_i(\mathbf{X}_1)\right]$. Współczynnik złożoności bloku wolnego w jest równy liczbie implikantów tak przedstawionej funkcji $f(\mathbf{X}_2, \mathbf{X}_1)$.

Optymalizacja bloku związanego wymaga przyjęcia pewnych uproszczeń, umożliwiających szybki wybór odpowiedniej dekompozycji wierszowej. Wyznaczana w wyniku kolorowania grafu niezgodności i dopełnień wierszy złożoność wierszowa powiązana jest z minimalną liczbą wyjść bloku związanego. Jeżeli liczba implikantów potrzebnych do realizacji funkcji $g_i(\mathbf{X}_1)$ jest mniejsza lub równa k, to liczba niezbędnych bloków logicznych typu PAL, potrzebnych do realizacji bloku związanego, wynosi $\delta_z=\mu(\mathbf{X}_2|\mathbf{X}_1)$. Fakt ten sprawia, że w celu minimalizacji wartości współczynnika δ_z przyjęto heurystyczną zasadę zawężenia obszaru poszukiwania rozwiązania do rozwiązań, dla których $\mu(\mathbf{X}_2|\mathbf{X}_1)$=min.

Przykład 4.23

Wyznaczmy współczynnik złożoności bloku wolnego funkcji analizowanej w przykładzie 4.21. W wyniku kolorowania wierzchołków grafu niezgodności i dopełnień

a) $\mathbf{X}_1=\{d,e,h\}$, $\mathbf{X}_2=\{a,b,c\}$

abc \ def	000	001	011	010	110	111	101	100	
000	0	1	0	1	0	-	1	0	A $\rightarrow g_1(X_1)$
001	1	1	1	0	1	0	1	0	B $\rightarrow g_2(X_1)$
011	0	0	0	1	0	1	0	1	$\bar{\mathbf{B}}$ $\rightarrow \bar{g}_2(X_1)$
010	0	1	0	1	0	-	1	0	A $\rightarrow g_1(X_1)$
110	0	0	0	0	0	0	0	0	0 0
111	0	0	0	1	0	1	0	1	$\bar{\mathbf{B}}$ $\rightarrow \bar{g}_2(X_1)$
101	1	1	1	0	1	0	1	0	B $\rightarrow g_2(X_1)$
100	0	0	0	0	0	0	0	0	0 0

$f(\mathbf{X}_2,\mathbf{X}_1)$

b)

```
.i 5
.o 1
.ilb a b c g1(X1) g2(X1)
.ob f(X2,X1)
.p 6
0001- 1
001-1 1
011-0 1
0101- 1
111-0 1
101-1 1
.e
```

c)

```
.i 5
.o 1
.ilb a b c g1(X1) g2(X1)
.ob f(X2,X1)
.p 3
0-01- 1
-01-1 1
-11-0 1
.e
```

w=3

Rys. 4.27. Sposób wyznaczania współczynnika złożoności bloku wolnego; a) wynik kolorowania; b) pierwotny opis funkcji realizowanej w bloku wolnym; c) wynik minimalizacji funkcji

Fig. 4.27. The method of determining the coefficient of the free block complexity; a) results of the colouring process; b) primary description of the function, which is implemented within the free block; c) results of the minimization process

wierszy (*przykład* 4.22) poszczególnym wierszom zostały przydzielone kolory. Doprowadziło to do określenia wartości funkcji dla wektorów, dla których przyjmowała ona wartości nieokreślone. Wynik tego etapu przedstawiony jest na rys. 4.27.a). W kolejnych krokach wyznacza się funkcję opisującą blok wolny (rys. 4.27.b)) i po jej minimalizacji współczynnik złożoności bloku wolnego (rys. 4.27.c)).

Ponieważ relacja dopełnienia wierszy jest relacją symetryczną (tzn., jeżeli wiersz X jest dopełnieniem wiersza Y, to wiersz Y jest dopełnieniem wiersza X), to istnieje możliwość takiego przyporządkowania wierszom kolorów dozwolonych, które prowadzi do minimalizacji współczynnika złożoności bloku związanego. Należy podkreślić fakt, że zamiana kolorów dozwolonych z kolorami dopełnień nie wpływa na wartość współczynnika złożoności bloku wolnego.

Przykład 4.24

Rozpatrzmy wzorce wierszy z przykładu 4.23. Ponieważ funkcja $g_1(\mathbf{X}_1)$ jest sumą mniejszej liczby implikantów niż funkcja $\bar{g}_1(\mathbf{X}_1)$, stąd właściwe jest przyporządkowanie koloru A do wzorców wierszy opisanych przez funkcję $g_1(\mathbf{X}_1)$.

$\mathbf{X}_1=\{d,e,h\}$

deh	000	001	011	010	110	111	101	100	
	0	1	0	1	0	-	1	0	A $\rightarrow g_1(\mathbf{X}_1)$
	1	0	1	0	1	-	0	1	$\bar{\mathrm{A}}$ $\rightarrow \bar{g}_1(\mathbf{X}_1)$

$g_1(\mathbf{X}_1)=\bar{e}h+\bar{d}e\bar{h}$ $\qquad$ $\bar{g}_1(\mathbf{X}_1)=\bar{e}\bar{h}+eh+d\bar{h}$

Inaczej wygląda sytuacja w przypadku wzorca B. Funkcja $g_2(\mathbf{X}_1)$ jest sumą większej liczby implikantów niż funkcja $\bar{g}_2(\mathbf{X}_1)$. W tej sytuacji korzystna jest zamiana przyporządkowania koloru dozwolonego z kolorem dopełnienia. Wpłynie to na minimalizację wykorzystywanych w bloku związanym iloczynów, nie wpływając na złożoność bloku wolnego.

$\mathbf{X}_1=\{d,e,h\}$

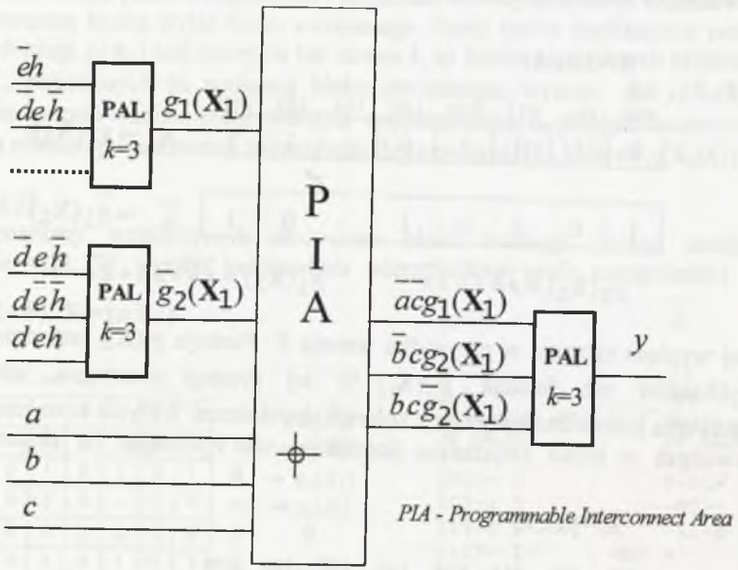
deh	000	001	011	010	110	111	101	100	
	1	1	1	0	1	0	1	0	B $\rightarrow g_2(\mathbf{X}_1)$
	0	0	0	1	0	1	0	1	$\bar{\mathrm{B}}$ $\rightarrow \bar{g}_2(\mathbf{X}_1)$

$g_2(\mathbf{X}_1)=\bar{d}\bar{e}+\bar{d}h+\bar{e}h+de\bar{h}$ $\qquad$ $\bar{g}_2(\mathbf{X}_1)=\bar{d}\bar{e}\bar{h}+deh+d\bar{e}\bar{h}$

Uwaga: Sposób przyporządkowania kolorów dozwolonych i kolorów dopełnień jest istotny tylko w przypadku wykorzystywania bloków logicznych typu PAL bez możliwości ustawiania aktywnego stanu wyjściowego.

Efektem rozważań przedstawionych w przykładach 4.21 - 4.24 jest realizacja przykładowej funkcji na blokach logicznych typu PAL, zawierających trzy iloczyny, przedstawiona na rys. 4.28.

abc \ deh	000	001	011	010	110	111	101	100	
000	0	1	0	1	0	-	1	0	A → $g_1(X_1)$
001	1	1	1	0	1	0	1	0	$\bar{B}$ → $g_2(X_1)$
011	0	0	0	1	0	1	0	1	B → $\bar{g}_2(X_1)$
010	0	1	0	1	0	-	1	0	A → $g_1(X_1)$
110	0	0	0	0	0	0	0	0	0 0
111	0	0	0	1	0	1	0	1	B → $\bar{g}_2(X_1)$
101	1	1	1	0	1	0	1	0	$\bar{B}$ → $g_2(X_1)$
100	0	0	0	0	0	0	0	0	0 0



Rys. 4.28. Realizacja funkcji za pomocą 3-iloczynowych bloków typu PAL

Fig. 4.28. Implementation of the function by means of three-terms PAL-based blocks

4.5.3. Algorytm implementacji funkcji w strukturach typu PAL oparty na dekompozycji wierszowej

Konsekwencją dekompozycji wierszowej jest podział układu na dwa podukłady: związany i wolny. Podział ten prowadzi do ekspansji sumarycznej liczby wyjść. Każde dodatkowe wyjście pociąga za sobą użycie kolejnego bloku logicznego typu PAL. W wyniku podziału należy wykorzystać co najmniej $\mu(\mathbf{X}_2 \mid \mathbf{X}_1)$ bloków logicznych typu PAL do realizacji bloku związanego. Fakt ten sprawia, że zastosowanie dekompozycji wierszowej, podobnie jak kolumnowej, jako elementu minimalizacji może być opłacalne tylko wtedy, gdy w klasycznym podejściu wykorzystanie sprzężeń zwrotnych prowadzi do wykorzystania większej liczby bloków logicznych typu PAL.

Minimalna liczba wyjść, równocześnie minimalna liczba bloków logicznych typu PAL, potrzebnych do realizacji układu po podziale, wynosi $\mu(\mathbf{X}_2 \mid \mathbf{X}_1)+1$ (co najmniej $\mu(\mathbf{X}_2 \mid \mathbf{X}_1)$ bloków logicznych typu PAL potrzebnych jest do realizacji bloku związanego i dokładnie jeden blok do realizacji bloku wolnego).

Rozpatrzmy realizację funkcji $f_i : B^n \to B$ na k-iloczynowych blokach PAL. Zminimalizowana postać funkcji $f_i : B^n \to B$, przedstawiona w postaci zbioru Δ_{f_i} implikantów, może być zrealizowana metodą klasyczną za pomocą δ_{f_i} bloków logicznych typu PAL, zawierających k-iloczynów, przy czym $\delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1$.

Zastosowanie dekompozycji wierszowej może prowadzić do ograniczenia liczby bloków logicznych typu PAL tylko wtedy, gdy:

$$\mu(\mathbf{X}_2 \mid \mathbf{X}_1)+1 < \delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 \qquad (4.9)$$

Zależność (4.9) stanowi warunek konieczny ograniczenia liczby bloków logicznych typu PAL, który może eliminować pewne podziały z dalszych rozważań lub wręcz wskazywać na niemożliwość uzyskania lepszych rozwiązań od metody klasycznej. Jeżeli na przykład zminimalizowana postać funkcji $f_i : B^n \to B$ opisana jest przez zbiór mniej niż $2k$-implikantów, to stosowanie dekompozycji wierszowej jest bezcelowe, gdyż na pewno nie ograniczy liczby bloków logicznych wykorzystywanych w metodzie klasycznej.

W procesie poszukiwania efektywnej dekompozycji wierszowej jednym z głównych elementów jest problem doboru odpowiedniego podziału argumentów funkcji. Powinien on zapewnić realizowalność bloku wolnego w bloku logicznym typu PAL o zadanej liczbie iloczynów oraz minimalizację uzyskiwanego bloku związanego.

Rozpatrzmy podział argumentów funkcji na rozłączne podzbiory $\mathbf{X}_1$ i $\mathbf{X}_2$. Realizowalność bloku wolnego w k-iloczynowym bloku PAL jest zapewniona dla każdego podziału, dla którego spełniona jest nierówność $2^{\overline{\overline{\mathbf{X}_2}}} \le k$, tzn. $\overline{\overline{\mathbf{X}_2}} \le [\lg_2 k]$. Fakt ten sprawia, że poszukiwania podziału argumentów zapewniającego realizowalność bloku wolnego w bloku logicznym typu PAL o zadanej liczbie iloczynów są rozpoczynane od

podziałów, dla których moc zbioru $\mathbf{X}_2$ spełnia warunek $\overline{\overline{\mathbf{X}_2}} = [\lg_2 k] + 1$. Przytoczony warunek doboru podziałów sprawia, że w pierwszym kroku analizowane są podziały, dla których może (nie musi) istnieć dekompozycja wierszowa prowadząca do realizacji bloku wolnego na k-iloczynowym bloku PAL. W zależności od uzyskanego wyniku pierwszego kroku moc zbioru wolnego jest zmniejszana o jeden (zawsze istnieje rozwiązanie) lub zwiększana o jeden (poszukiwanie lepszego rozwiązania).

Algorytm dekompozycji wierszowej funkcji jednowyjściowej, prowadzący do efektywnego podziału układu na blok związany i wolny, realizowane na k-iloczynowych blokach PAL

1. Minimalizacja funkcji $f_i : B^n \to B^1$ (Espresso); wyznaczanie parametru $\delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1$

2. Jeżeli $\Delta_{f_i} < 2k$, to realizacja funkcji $f_i : B^n \to B^1$ metodą klasyczną; w przeciwnym przypadku analiza podziałów argumentów funkcji i poszukiwanie odpowiedniej dekompozycji wierszowej, tzn.:

   A. Rozpatrywane są podziały zbioru $\mathbf{I} = \{i_n, \ldots, i_2, i_1\}$ na podzbiory $\mathbf{X}_1 = \{i_{n_z}, \ldots, i_2, i_1\}$ oraz $\mathbf{X}_2 = \{i_n, \ldots, i_{n_z+2}, i_{n_z+1}\}$ takie, że $\overline{\overline{\mathbf{X}_2}} = [\lg_2 k] + 1$, gdzie k jest liczbą iloczynów zawartych w bloku logicznym typu PAL;

   B. Dla poszczególnych podziałów wyznaczane są złożoności wierszowe (algorytm kolorowania grafu niezgodności i dopełnień wierszy) oraz współczynniki złożoności bloku wolnego;

   C. Spośród analizowanych podziałów, dla których współczynniki złożoności bloku wolnego są mniejsze lub równe k, jako rozwiązanie wybierany jest jeden podział, dla którego współczynnik złożoności wierszowej $\mu(\mathbf{X}_2 \mid \mathbf{X}_1)$ przyjmuje wartość minimalną. Jeżeli znaleziono taki podział, to jest on zapamiętywany, po czym następuje przejście do punktu 2D. Jeżeli do tego kroku nie wybrano żadnego podziału, to następuje przejście do punktu 2E. Jeżeli nie wybrano żadnego podziału w aktualnym kroku, a wybrano podział w poprzednich krokach, to ostatnio wybrany podział jest poszukiwanym podziałem, po czym następuje przejście do punktu 2F;

   D. Poszukiwany jest „lepszy podział", zapewniający realizowalność bloku wolnego w bloku logicznym typu PAL o zadanej liczbie iloczynów k, w kolejnej grupie podziałów, dla których $\overline{\overline{\mathbf{X}_2}} == \overline{\overline{\mathbf{X}_2}} + 1$. Analizie poddawane są tylko wybrane podziały, dla których $\overline{\overline{\mathbf{X}_2}} == \overline{\overline{\mathbf{X}_2}} + 1$, przy czym dodatkowa zmienna dodawana jest do zbiorów wolnych, dla których w poprzednim kroku (2C) złożoność bloku wolnego $w \le k$.

   E. Jeżeli nie znaleziono żadnego podziału spełniającego warunek realizowalności w bloku wolnym, zawierającym k-iloczynów, to rozwiązanie wybierane jest w zbiorze podziałów, dla których $\overline{\overline{\mathbf{X}_2}} = [\lg_2 k]$. Wybierane jest rozwiązanie, dla którego $\mu(\mathbf{X}_2 \mid \mathbf{X}_1)$ przyjmuje wartość minimalną.

   F. Jeżeli dla wybranego podziału zachodzi zależność $\mu(\mathbf{X}_2 \mid \mathbf{X}_1) + 1 > \delta_{f_i}$, to analizowana funkcja realizowana jest metodą klasyczną.

3. Dla wybranego podziału poszukiwane jest efektywne przyporządkowanie kolorów dozwolonych i dopełnień, prowadzące do minimalizacji współczynnika złożoności bloku związanego (przykład 4.24); przyporządkowanie jest poszukiwane za pomocą programu do minimalizacji (Espresso - Dso_both).

4. Minimalizacja funkcji realizowanych w bloku związanym $g : B^{n_z} \to B^p$ (Espresso-Dso); wyznaczanie liczby bloków logicznych typu PAL, potrzebnych do realizacji układu po podziale; liczba ta oznaczana przez symbol δ_{zw} jest sumą liczby bloków potrzebnych do realizacji bloku związanego (δ_z) i wolnego (δ_w=1).

$$\delta_{zw} = \delta_z + 1 = \sum_{i=1}^{p} \left(\left\lceil \frac{z_i - k}{k-1} \right\rceil + 1 \right) + 1$$

5. Jeżeli $\delta_{f_i} \le \delta_{zw}$, to realizacja funkcji metodą klasyczną; w przeciwnym przypadku kolejny etap dekompozycji wierszowej - podział bloku związanego (p.2), realizowany dla każdej funkcji $g_i : B^{n_z} \to B^1$, gdzie i=1,2,..., $\mu(\mathbf{X}_2 \mid \mathbf{X}_1)$.

Przykład 4.25

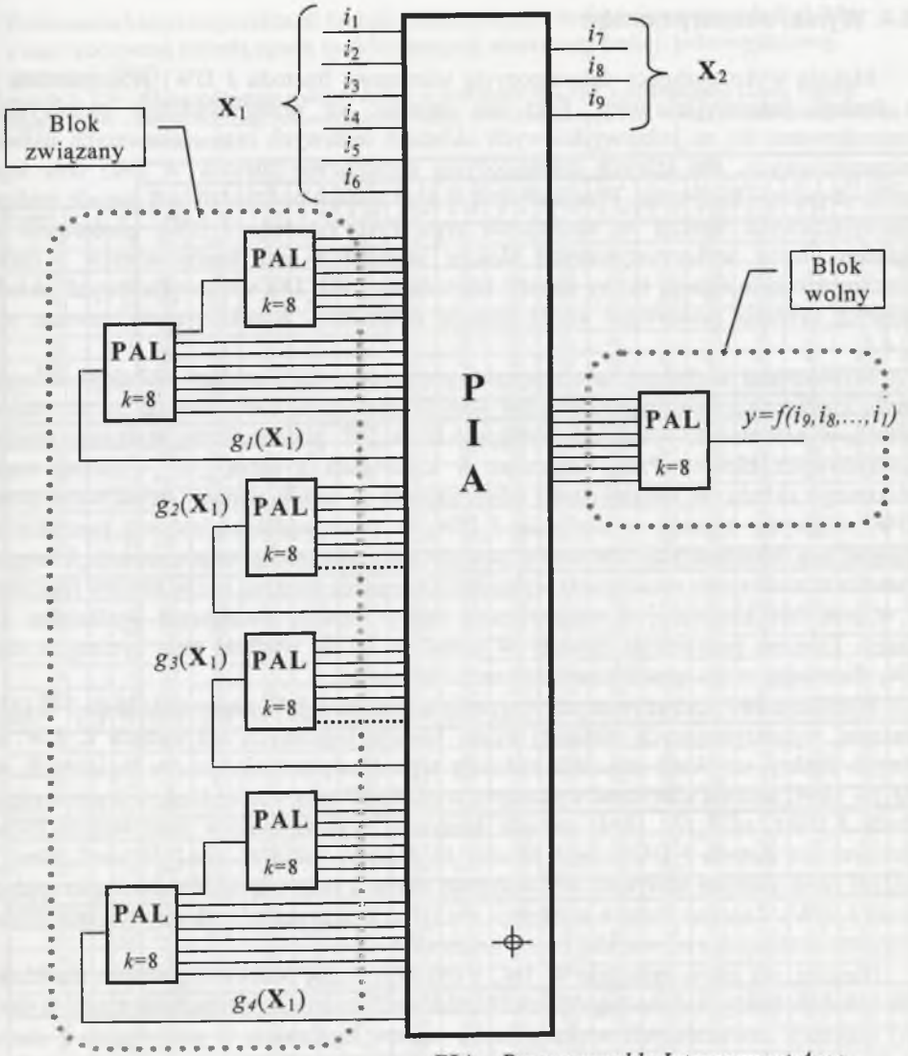
Rozpatrzmy ponownie realizację funkcji testowej *9sym* ($f_i : B^9 \to B^1$) korzystając z bloków logicznych typu PAL, zawierających 8 iloczynów (k=8). W dalszej części przykładu przedstawione są kolejne etapy poszukiwania rozwiązania zgodne z algorytmem dekompozycji wierszowej funkcji.

- W wyniku minimalizacji (Espresso) uzyskujemy parametr $\Delta_{f_i} = 87$, który pozwala wyznaczyć liczbę bloków potrzebnych do realizacji klasycznej $\delta_{f_i} = \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 = \left\lceil \frac{87-8}{7} \right\rceil + 1 = 13$.
- Ponieważ $\Delta_{f_i} > 2k$, stąd poszukiwana jest odpowiednia dekompozycja wierszowa analizowanej funkcji. W pierwszej kolejności rozpatrywane są rozłączne podziały argumentów, dla których $\overline{\overline{\mathbf{X}_2}} = [\lg_2 k] + 1 = [\lg_2 8] + 1 = 4$ oraz $\overline{\overline{\mathbf{X}_1}} = 5$.
- Dla wszystkich analizowanych podziałów w wyniku kolorowania grafu niezgodności i dopełnień wierszy uzyskiwana jest złożoność wierszowa $\mu(\mathbf{X}_2 \mid \mathbf{X}_1)$=4.

- Dla wszystkich podziałów analizowanej grupy podziałów uzyskiwany w wyniku podziału blok wolny nie jest realizowalny w bloku logicznym typu PAL, zawierającym osiem iloczynów.
- W kolejnym kroku rozpatrywane są podziały, dla których $\overline{\overline{\mathbf{X}_2}} = [\lg_2 k] = [\lg_2 8] = 3$ oraz $\overline{\overline{\mathbf{X}_1}} = 6$.
- Dla wszystkich analizowanych podziałów w wyniku kolorowania grafu niezgodności i dopełnień wierszy uzyskiwana jest złożoność wierszowa $\mu(\mathbf{X}_2 \mid \mathbf{X}_1)=4$.
- Dla wszystkich podziałów analizowanej grupy podziałów uzyskiwany w wyniku podziału blok wolny jest realizowalny w bloku logicznym typu PAL, zawierającym osiem iloczynów.
- Nie występują żadne kolory dopełnień; minimalizacja funkcji realizowanych w bloku związanym (Espresso-Dso) wskazała, że lepiej wzorce wierszy skojarzyć z kolorami dopełnień (minimalizacja bloku związanego).
- W wyniku minimalizacji uzyskiwane są współczynniki złożoności bloku związanego z_1=15, z_2=7, z_3=7, z_4=15, co pozwala wyznaczyć liczbę bloków logicznych typu PAL, potrzebnych do realizacji układu po podziale:

$$\delta_{zw} = \delta_z + \delta_w = \sum_{i=1}^{4} \left(\left\lceil \frac{z_i - k}{k-1} \right\rceil + 1 \right) + 1 = 7$$

- Wyszukiwany podział jest akceptowany, ponieważ $\delta_{zw} < \delta_{f_i}$.
- Ponieważ współczynniki złożoności bloku związanego z_1 do z_4 są mniejsze od wartości $2k = 16$, stąd poszukiwania zostają zakończone. Ostateczne rozwiązanie, przedstawione na rys. 4.29 wykorzystuje tylko 7 bloków logicznych typu PAL, zawierających 8 iloczynów (4 bloki mniej niż metoda klasyczna, 2 bloki mniej niż rozwiązanie uzyskane za pomocą dekompozycji kolumnowej).



PIA - *Programmable Interconnect Area*

linią przerywaną oznaczono niewykorzystane iloczyny

Rys. 4.29. Wynik dekompozycji wierszowej układu testowego *9sym*
Fig. 4.29. Results of row decomposition for the *9sym* benchmark

4.5.4. Wyniki eksperymentów

Metoda wykorzystująca dekompozycję wierszową (metoda J_DW) przeznaczona jest dla funkcji jednowyjściowych. Fakt ten sprawia, że przeprowadzając eksperymenty skoncentrowano się na jednowyjściowych układach testowych oraz ciekawszych układach wielowyjściowych, dla których dekompozycja kolumnowa (metoda W_DK) dała lepsze wyniki od metody klasycznej. Przedstawioną w poprzednich podrozdziałach metodę realizacji jednowyjściowych funkcji w strukturach typu PAL (metoda J_DW) porównano pod względem liczby wykorzystywanych bloków logicznych oraz liczby warstw z metodą klasyczną z minimalizacją liczby warstw (metoda K_LW). Dla wielowyjściowych układów testowych syntezie poddawano każdą funkcję oddzielnie. Wyniki syntezy zawarte są w tab. 4.4.

W pierwszej części tej tabeli przedstawione są wyniki syntezy układów testowych metodą klasyczną z minimalizacją liczby warstw (metoda K_LW). Tak jak w poprzednich tabelach, w kolumnach, w których występuje litera „B", podano liczbę wykorzystywanych *k*-iloczynowych bloków PAL, natomiast w kolumnach z literą „W" - liczbę warstw wynikowego układu. W drugiej części tabeli zawarte są wyniki syntezy uzyskiwane metodą J_DW. Algorytm syntezy w metodzie J_DW, w poszczególnych etapach poszukiwania odpowiedniej dekompozycji wierszowej zawiera elementy porównania z metodą klasyczną. Zapewnia to znalezienie rozwiązania wykorzystującego nie większą liczbę bloków logicznych niż w metodzie klasycznej z minimalizacją liczby warstw, nieznacznie wydłużając czas syntezy. Element porównania zawarty w metodzie J_DW wydłuża czas syntezy o około 2-3%, eliminując w ten sposób nieefektywne rozwiązania.

W zbiorze 440 porównywanych przypadków metoda J_DW doprowadziła do 141 (32%) rozwiązań wykorzystujących mniejszą liczbę bloków logicznych niż metoda K_LW. Dla pewnych funkcji uzyskano znaczącą redukcję wykorzystywanych bloków logicznych, np.: *rd84_f1* (*k*=4; metoda klasyczna wykorzystuje około 10 razy więcej bloków logicznych niż metoda J_DW), *rd73_f1* (*k*=4; metoda klasyczna wykorzystuje 7 razy więcej bloków logicznych niż metoda J_DW), *5xp1_f2, clip_f3, f51m_f1, rd53_f1, sao2_f3, xor5, z4ml_f2, z5xp1_f3* (*k*=4; metoda klasyczna wykorzystuje około 2 razy więcej bloków logicznych niż metoda J_DW). Znaczne różnice występują nie tylko w przypadku wykorzystywania bloków logicznych złożonych z niewielkiej liczby iloczynów.

Niestety, tak jak w metodzie W_DK, z redukcją liczby bloków logicznych nie idzie w parze redukcja liczby warstw logicznych. Wśród analizowanych przypadków znalazło się 37 (8%) realizacji zawierających większą liczbę warstw logicznych w porównaniu z metodą K_LW i tylko 9 (2%) rozwiązań zawierających mniejszą liczbę warstw logicznych. W pozostałych przypadkach uzyskiwano identyczną liczbę warstw logicznych.

Podstawową wadą metody J_DW jest złożoność przedstawionych algorytmów syntezy, co praktycznie ogranicza jej wykorzystanie do układów zawierających maksymalnie kilkanaście wejść.

Podsumowaniem powyższych rozważań mogą być wykresy 4.3 oraz 4.4. Wartości przedstawione na wykresie 4.3 wyznaczone zostały z zależności $\frac{\sum bloków_{(K\_LW)} - \sum bloków_{(J\_DW)}}{\sum bloków_{(J\_DW)}}$ 100%, gdzie $\sum bloków_{(K\_LW)}$ i $\sum bloków_{(J\_DW)}$ oznaczają odpowiednie sumy liczby bloków, wykorzystywanych w poszczególnych metodach syntezy, przedstawione w tab. 4.4.

Tabela 4.4

Porównanie klasycznej realizacji funkcji z minimalizacją liczby warstw (metoda K_LW) z zaproponowaną metodą opartą na dekompozycji wierszowej funkcji jednowyjściowej (metoda J_DW)

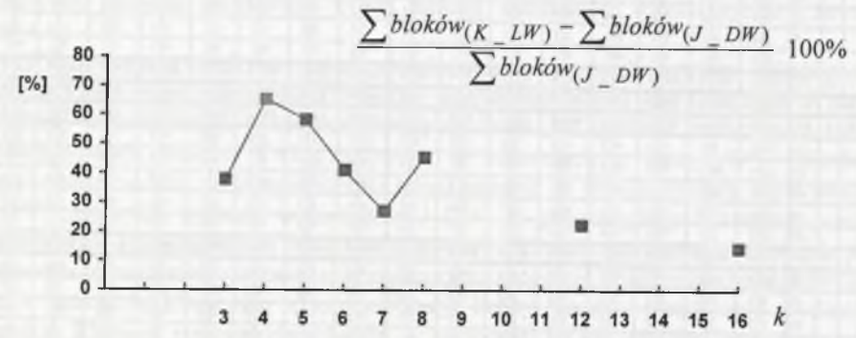
metoda K_LW - Klasyczna metoda realizacji funkcji w strukturach typu PAL z minimalizacją Liczby Warstw
metoda J_DW - metoda realizacji Jednowyjściowej funkcji w strukturach typu PAL wykorzystująca Dekompozycję Wierszową

				K_LW												J_DW											
				k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16		k= 3		k= 4		k= 5		k= 6		k= 7	
Przykład	i	o	p	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W
f0			7	3	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	3	2	2	2	2	2	2	1	1	1
f1			11	5	3	4	2	3	2	2	2	2	2	1	1	1	1			5	3	4	3	3	2	2	2	2	2
f2			18	9	3	6	3	5	2	4	2	3	2	3	2	2	2	2	2	5	3	3	2	3	2	3	2	2	2
f3			14	7	3	5	2	4	2	3	2	3	2	2	2	2	2	1	1	4	3	3	2	3	2	3	2	2	2
f4			10	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	3	3	3	3	2	2	2	2	2	2
f5			5	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1
f6-f9				4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1
5xp1	7	10	75	35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2	26	3	21	3	18	2	17	2	16	2
9sym	9	1	87	43	5	29	4	22	3	18	3	15	3	13	3	8	2	6	2	42	5	21	4	15	4	14	4	14	4
9symml	9	1	87	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	42	5	21	4	15	4	14	4	14	4
f0			21	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	10	3	7	3	5	2	4	2	4	2
f1			31	15	4	10	3	8	3	6	3	5	2	5	2	3	2	2	2	15	4	10	3	8	3	6	3	5	3
f2			42	21	4	14	3	11	3	9	3	7	2	6	2	4	2	3	2	21	4	14	3	11	3	9	3	7	2
f3			34	17	4	11	3	9	3	7	3	6	2	5	2	3	2	2	2	9	4	6	3	5	3	5	3	4	2
f4			20	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	10	3	7	3	5	2	4	2	4	2
f51m	9	5	167	73	4	49	3	38	3	30	3	26	2	22	2	14	2	11	2	65	4	44	3	34	3	28	3	24	3
f0			23	11	3	8	3	6	2	5	2	4	2	4	2	2	2	2	2	6	4	5	3	5	3	3	2	3	2
f1			18	9	3	6	3	5	2	4	2	3	2	3	2	2	2	2	2	5	3	3	2	3	2	3	2	3	2
f2			14	7	3	5	2	4	2	3	2	3	2	2	2	2	2	1	1	4	2	3	2	2	2	2	2	2	2
f3			10	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	3	3	3	3	2	2	2	2	2	2
f4			5	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1
f5-f7			1..3	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1
rd53	5	3	32	15	3	11	2	8	2	6	2	6	2	6	2	4	2	3	1	11	3	7	2	6	2	5	2	5	2
f0			42	21	4	14	3	11	3	9	3	7	2	6	2	4	2	3	2	21	4	7	3	7	3	7	3	7	3
f1			64	32	4	21	3	16	3	13	3	11	3	9	2	6	2	5	2	6	3	3	2	3	2	3	2	3	2
f2			35	17	4	12	3	9	3	7	2	6	2	5	2	4	2	3	2	15	4	7	3	5	3	5	3	5	3
rd73	7	3	141	70	4	47	3	36	3	29	3	24	3	20	2	14	2	11	2	42	5	17	3	15	3	15	3	15	3
f0			84	42	4	28	3	21	3	17	2	14	2	12	2	8	2	6	2	40	4	11	4	7	3	7	3	7	3
f1			128	64	5	43	4	32	4	26	3	22	3	19	3	12	2	9	2	7	3	4	3	4	3	4	3	4	3
f2			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f3			70	35	4	23	3	18	3	14	3	12	3	10	3	7	2	5	2	28	4	11	3	11	3	11	3	10	3
rd84	8	4	256	142	5	95	4	72	4	58	3	49	3	42	3	28	2	21	2	76	4	27	4	23	3	23	3	22	3
f0			10	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	5	3	3	2	3	2	2	2	2	2
f1			20	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	10	3	7	3	5	2	4	2	4	2
f2			22	11	3	7	3	6	2	5	2	4	2	3	2	2	2	2	2	8	3	7	3	6	3	4	2	4	2
f3			21	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	5	3	4	3	4	3	3	2	3	2
sao2	10	4	58	36	3	24	3	19	2	15	2	14	2	11	2	7	2	7	2	28	3	21	3	18	3	13	2	13	2
xor5	5	1	16	8	2	5	2	4	2	3	2	3	2	3	2	2	2	1	1	4	2	2	2	2	2	2	2	2	2
f0			15	7	3	5	2	4	2	3	2	3	2	2	2	2	2	1	1	4	3	3	3	3	2	3	2	2	2
f1			28	14	3	9	3	7	3	6	2	5	2	4	2	3	2	2	2	5	4	3	3	3	3	3	3	2	2
f2			12	6	3	4	2	3	2	3	2	2	2	2	2	1	1	1	1	4	3	2	2	2	2	2	2	1	1
f3			4	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1
z4ml	7	4	59	29	3	19	3	15	3	13	2	11	2	9	2	7	2	5	2	15	4	9	3	9	3	9	3	8	2
f1			7	3	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	3	2	2	2	2	2	2	2	1	1
f2			11	5	3	4	2	3	2	2	2	2	2	2	2	1	1	1	1	5	3	4	2	3	2	3	2	2	2
f3			22	11	3	7	3	6	2	5	2	4	2	3	2	2	2	2	2	5	3	3	2	3	2	3	2	2	2
f4			14	7	3	5	2	4	2	3	2	3	2	2	2	2	2	1	1	4	3	3	2	3	2	3	2	2	2
f5			10	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	3	3	3	3	2	2	2	2	2	2
f6			5	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1
f0;f7-f9				4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1
z5xp1	7	10	128	37	3	27	3	23	2	19	2	17	2	15	2	12	2	11	2	26	3	21	3	18	2	17	2	16	2
z9sym	9	1	420	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	42	5	21	4	15	4	14	4	14	4
Suma bloków				609		415		323		261		225		195		135		109		442		251		204		185		177	
Suma warstw				48		41		34		32		31		30		26		24		50		41		38		36		35	

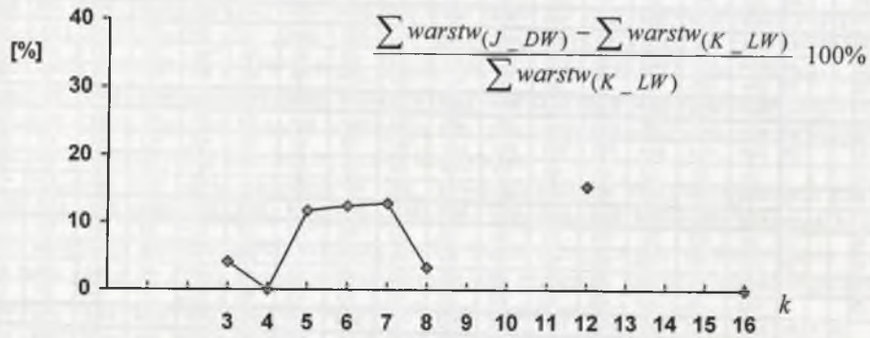
				J_DW											
				k= 8		k= 12		k= 16							
Przykład	i	o	p	B	W	B	W	B	W						
f0			7	1	1	1	1	1	1						
f1			11	2	2	2	2	2	2						
f2			18	2	2	2	2	1	1						
f3			14	2	2	2	2	1	1						
f4			10	2	2	1	1	1	1						
f5			5	1	1	1	1	1	1						
f6-f9				4	1	4	1	4	1						
5xp1	7	10	75	16	2	14	2	12	2	11	2				
9sym	9	1	87	14	4	7	3	7	3	5	2				
9symml	9	1	87	14	4	7	3	7	3	5	2				
f0			21	4	2	3	2	2	2						
f1			31	6	3	5	3	5	3						
f2			42	7	2	6	2	4	2	3	2				
f3			34	5	3	5	3	4	2	4	2				
f4			20	4	2	3	2	2	2						
f51m	9	5	167	34	3	28	3	24	3	21	2	14	2	12	2
f0			23	5	3	3	2	3	2	3	2				
f1			18	3	2	3	2	3	2	3	2				
f2			14	2	2	2	2	2	2	2	2				
f3			10	2	2	2	2	2	2	2	2				
f4			5	1	1	1	1	1	1	1	1				
f5-f7			1..3	3	1	3	1	3	1	3	1				
rd53	5	3	32	10	3	9	2	8	2						
f0			42	7	3	7	3	7	3						
f1			64	6	3	5	3	5	3	4	2				
f2			35	5	3	5	3	5	3	4	2				
rd73	7	3	141	15	3	15	3	10	3	9	2	8	2		
f0			84	7	3	7	3	5	3	3	2				
f1			128	4	3	4	3	3	3	3	2				
f2			1	1	1	1	1	1	1	1	1				
f3			70	11	3	11	3	10	3	6	3	5	2	4	2
rd84	8	4	256	22	3	15	3	12	3	10	2				
f0			10	3	2	2	2	2	2	1	1				
f1			20	5	2	4	2	4	2	3	2				
f2			22	4	2	4	2	3	2	2	2				
f3			21	3	2	3	2	3	2	2	2				
sao2	10	4	58	13	2	13	2	11	2	7	2	7	2		
xor5	5	1	16	2	2	2	2	2	2	2	2	1	1		
f0			15	3	2	3	2	2	2	2	2				
f1			28	3	3	3	3	2	2	2	2				
f2			12	2	2	2	2	1	1	1	1				
f3			4	1	1	1	1	1	1	1	1				
z4ml	7	4	59	9	3	9	3	8	2	7	2	6	2	5	2
f1			7	2	2	2	2	1	1	1	1				
f2			11	3	2	2	2	2	2	2	2				
f3			22	3	2	3	2	2	2	2	2				
f4			14	3	2	3	2	2	2	2	2				
f5			10	3	3	2	2	2	2	1	1				
f6			5	2	2	1	1	1	1	1	1				
f0;f7-f9				4	1	4	1	4	1	4	1				
z5xp1	7	10	128	19	2	17	2	16	2	14	2	12	2	11	2
z9sym	9	1	420	14	4	7	3	7	3	5	2				
Suma bloków				134		110		93							
Suma warstw				35		31		30		24					

Podobnie wyznaczone zostały wartości przedstawione na wykresie 4.4 na podstawie zależności $\frac{\sum warstw_{(J\_DW)} - \sum warstw_{(K\_LW)}}{\sum warstw_{(K\_LW)}} 100\%$, gdzie $\sum warstw_{(K\_LW)}$ i $\sum warstw_{(J\_DW)}$ oznaczają odpowiednie sumy liczby warstw logicznych dla poszczególnych wartości *k*.

Wykres 4.3. Porównanie syntezy metodą J_DW z metodą K_LW pod względem liczby bloków (opis w tekście)



Wykres 4.4. Porównanie syntezy metodą J_DW z metodą K_LW pod względem liczby warstw logicznych (opis w tekście)



Analiza wyników pozwala stwierdzić, że uzyskana metodą J_DW redukcja liczby bloków pociąga za sobą (w większości przypadków) ekspansję liczby warstw logicznych. Metoda J_DW jest szczególnie skuteczna w sytuacji, gdy k=4 lub k=8. Występuje wtedy znaczne zmniejszenie liczby bloków przy porównywalnej liczbie warstw logicznych. Powyższy wniosek jest intuicyjnie wyczuwalny. Rozpatrywane podziały argumentów funkcji tworzą zawsze 2^l wierszy. Dla funkcji „słabo dekomponowalnych" często występują sytuacje, w których złożoność wierszowa jest równa liczbie wierszy siatki Karnaugha. Sytuacje takie są szczególnie korzystne w przypadku wykorzystywania 2^l iloczynów bloków logicznych typu PAL.

4.6. Porównanie zaproponowanych modeli dekompozycji zorientowanych na realizację funkcji w strukturach typu PAL

Ponieważ dekompozycja wierszowa (metoda J_DW) przeznaczona jest dla funkcji jednowyjściowej, stąd w pierwszej kolejności porównano ją z metodą dekompozycji kolumnowej funkcji jednowyjściowej (metoda J_DK). Do porównania wybrano układy testowe, dla których obie metody dekompozycji dały lepsze rozwiązania od metody klasycznej.

Wyniki eksperymentów zawarte w tab. 4.5 pokazują, że metoda J_DW w 49 przypadkach (11%) daje lepsze rozwiązanie pod względem liczby wykorzystanych bloków logicznych typu PAL. Metoda J_DK jest skuteczniejsza w 44 przypadkach (10%).

Analizując liczby warstw logicznych wynikowych układów okazuje się, że metodą J_DW znaleziono 61 rozwiązań (14%) z mniejszą liczbą warstw logicznych od liczby warstw uzyskiwanych metodą J_DK. Metoda J_DK pod tym względem była skuteczniejsza tylko w 13 przypadkach (3%).

Powyższe liczby wskazują, iż metoda J_DW, zawierająca elementy dopasowania dekompozycji do struktury bloku logicznego typu PAL, jest skuteczniejsza od metody J_DK pod względem liczby warstw logicznych. Jest również nieznacznie lepsza pod względem wykorzystywanych bloków logicznych typu PAL.

Głębsza analiza uzyskanych wyników pozwala zauważyć, że metoda J_DW jest szczególnie skuteczna dla bloków, dla których k=4 lub k=8 (k=4 - prawie dwa razy więcej lepszych rozwiązań niż metoda J_DK; k=8 – trzy razy więcej lepszych rozwiązań niż metoda J_DK).

Mimo iż metodą J_DW uzyskiwano większą liczbę lepszych rozwiązań (patrząc na liczby bloków logicznych) okazało się, że dla analizowanego zbioru układów testowych wykorzystano większą liczbę bloków logicznych. Redukcja liczby warstw logicznych uzyskiwana metodą J_DW występuje tylko i wyłącznie dla niewielkich bloków logicznych typu PAL (k=3,4,5,6,7,8). Dla większych bloków logicznych (k=12,16) uzyskano porównywalną sumaryczną liczbę warstw logicznych.

Podsumowaniem powyższych rozważań niech będą wykresy 4.5 i 4.6. Wartości przedstawione na wykresie 4.5 wyznaczone zostały z zależności $\frac{\sum bloków_{(J\_DW)} - \sum bloków_{(J\_DK)}}{\sum bloków_{(J\_DK)}} 100\%$, gdzie $\sum bloków_{(J\_DW)}$ i $\sum bloków_{(J\_DK)}$ oznaczają odpowiednie sumy liczby bloków, wykorzystywanych w poszczególnych metodach syntezy, przedstawione w tab. 4.5.

Podobnie wyznaczone zostały wartości przedstawione na wykresie 4.6 na podstawie zależności $\frac{\sum warstw_{(J\_DK)} - \sum warstw_{(J\_DW)}}{\sum warstw_{(J\_DW)}} 100\%$, gdzie $\sum warstw_{(J\_DK)}$ i $\sum warstw_{(J\_DW)}$ oznaczają odpowiednie sumy liczby warstw logicznych dla poszczególnych wartości *k*.

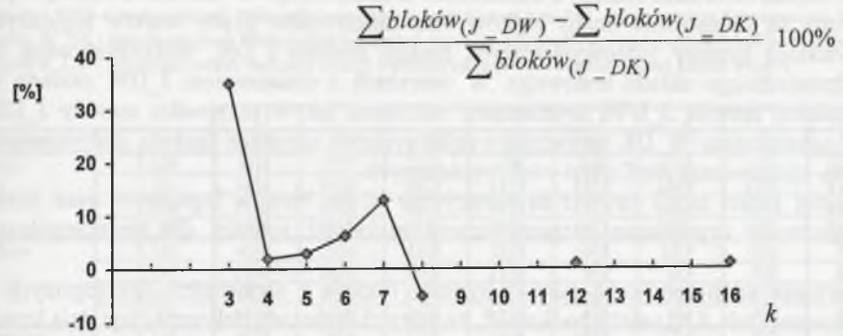
Tabela 4.5

Porównanie zaproponowanych modeli dekompozycji funkcji jednowyjściowych

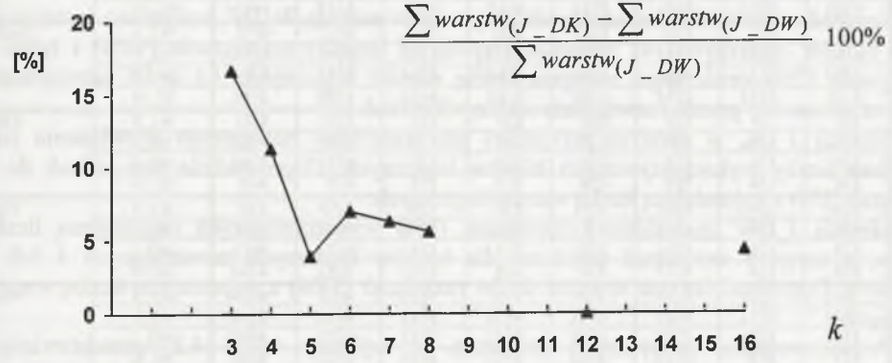
metoda J_DK - metoda realizacji Jednowyjściowej funkcji w strukturach typu PAL wykorzystująca Dekompozycję Kolumnową
metoda J_DW - metoda realizacji Jednowyjściowej funkcji w strukturach typu PAL wykorzystująca Dekompozycję Wierszową

	J_DK														J_DW											
	k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16		k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16	
Przykład	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W
f0	3	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	3	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1
f1	5	3	4	2	3	2	2	2	2	2	2	2	1	1	1	1	6	3	4	3	3	2	2	2	2	2	2	2	1	1	1	1
f2	5	3	5	3	3	2	3	2	3	2	3	2	2	2	2	2	5	3	3	2	3	2	3	2	3	2	2	2	2	2	2	2
f3	4	3	4	3	3	2	3	2	3	2	2	2	2	2	1	1	4	3	3	2	3	2	3	2	3	2	2	2	2	2	1	1
f4	3	3	3	3	2	2	2	2	2	2	2	2	1	1	1	1	3	3	3	3	2	2	2	2	2	2	2	2	1	1	1	1
f5	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
f6-f9	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1
5xp1	26	3	24	3	18	2	17	2	16	2	15	2	12	2	11	2	27	3	21	3	18	2	17	2	16	2	14	2	12	2	11	2
9sym	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3	42	5	21	4	15	4	14	4	14	4	14	4	7	3	7	3	5	2
9symml	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3	42	5	21	4	15	4	14	4	14	4	14	4	7	3	7	3	5	2
f0	9	5	6	3	5	2	4	2	4	2	3	2	2	2	2	2	16	3	7	3	5	2	4	2	4	2	4	2	3	2	2	2	2	2
f1	15	4	10	3	8	3	6	3	5	2	5	2	3	2	2	2	15	4	10	3	8	3	6	3	5	3	5	2	3	2	2	2	2	2
f2	14	5	10	4	8	4	6	4	6	4	5	3	4	2	3	2	21	4	14	3	11	3	9	3	7	2	6	2	4	2	3	2	2	2
f3	8	4	6	4	5	3	5	3	3	2	3	2	3	2	2	2	9	4	6	3	5	3	5	3	4	2	4	2	3	2	3	2	2	2
f4	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	10	3	7	3	5	2	4	2	4	2	4	2	3	2	2	2	2	2
clip	56	5	39	4	31	4	25	4	22	4	19	3	14	2	11	2	65	4	44	3	34	3	28	3	24	3	21	2	14	2	12	2
f0	10	5	7	4	6	2	5	2	4	2	4	2	2	2	2	2	6	4	5	3	5	3	3	2	3	2	3	2	2	2	2	2	2	2
f1	6	4	5	3	4	3	3	2	3	2	3	2	2	2	2	2	5	3	3	2	3	2	3	2	3	2	3	2	2	2	2	2	2	2
f2	6	4	4	3	4	2	3	2	3	2	2	2	2	2	1	1	4	2	3	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1
f3	3	2	3	2	2	2	2	2	2	2	2	2	1	1	1	1	3	3	3	3	2	2	2	2	2	2	2	2	1	1	1	1	1	1
f4	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f5-f7	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1
f51m	30	5	24	4	20	3	17	2	16	2	15	2	11	2	10	2	23	4	19	3	16	3	14	2	14	2	14	2	11	2	10	2
f0	2	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f1	4	4	2	2	2	2	2	2	2	2	2	2	2	2	1	1	4	3	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1
f2	5	3	4	2	3	2	2	2	2	2	2	2	1	1	1	1	5	3	3	2	3	2	2	2	2	2	2	2	2	2	1	1	1	1
rd53	11	4	8	2	6	2	5	2	5	2	5	2	4	2	3	1	11	3	7	2	6	2	5	2	5	2	5	2	4	2	3	1
f0	8	4	7	4	4	2	4	2	4	2	3	2	3	2	3	2	21	4	7	3	7	3	7	3	7	3	3	2	3	2	3	2	3	2
f1	6	3	4	2	4	2	4	2	4	2	2	2	2	2	2	2	6	3	3	2	3	2	3	2	3	2	3	2	2	2	2	2	2	2
f2	16	5	11	4	9	4	7	4	6	4	5	3	4	2	3	2	15	4	7	3	5	3	5	3	5	3	5	3	4	2	3	2	3	2
rd73	30	5	22	4	17	4	15	4	14	4	10	3	9	2	8	2	42	5	17	3	15	3	15	3	15	3	10	3	9	2	8	2	8	2
f0	14	5	10	4	8	4	6	4	6	4	5	3	3	2	3	2	40	4	11	4	7	3	7	3	7	3	5	3	3	2	3	2	3	2
f1	7	3	4	3	4	3	4	3	4	3	3	3	3	3	2	2	7	3	4	3	4	3	4	3	4	3	3	3	3	3	2	2	2	2
f2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f3	14	5	11	4	8	4	6	4	6	4	6	4	4	2	4	2	28	4	11	3	11	3	11	3	10	3	6	3	5	2	4	2	4	2
rd84	36	5	26	4	21	4	17	4	17	4	15	4	11	3	10	2	76	4	27	4	23	3	23	3	22	3	15	3	12	3	10	2	10	2
f0	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	1	1
f1	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	2	2
f2	10	4	7	3	6	2	5	2	4	2	3	2	2	2	2	2	8	3	7	3	6	3	4	2	4	2	3	2	2	2	2	2	2	2
f3	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	6	3	4	3	4	3	3	2	3	2	3	2	2	2	2	2	2	2
sao2	35	4	24	3	19	2	15	2	14	2	11	2	7	2	7	2	29	3	21	3	18	3	13	2	13	2	11	2	7	2	7	2	6	2
xor5	4	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	4	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1
f0	4	3	3	3	3	3	3	3	2	2	2	2	2	2	1	1	4	3	3	3	3	3	3	2	3	2	2	2	2	2	2	2	1	1
f1	4	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	5	4	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2
f2	3	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	4	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1
f3	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
z4m1	13	3	9	3	9	3	9	3	7	2	7	2	6	2	5	2	15	4	9	3	9	3	9	3	8	2	7	2	6	2	5	2	5	2
f1	3	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	3	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1
f2	5	3	4	2	3	2	2	2	2	2	2	2	1	1	1	1	5	3	4	2	3	2	2	2	2	2	2	2	1	1	1	1	1	1
f3	10	4	6	4	6	2	5	2	4	2	3	2	2	2	2	2	5	3	3	2	3	2	3	2	3	2	2	2	2	2	2	2	2	2
f4	6	4	4	3	4	2	3	2	3	2	2	2	2	2	1	1	4	3	3	2	3	2	3	2	3	2	2	2	2	2	1	1	1	1
f5	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	3	3	3	3	2	2	2	2	2	2	2	2	1	1	1	1	1	1
f6	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f0;f7-f9	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1
z5xp1	35	4	25	4	23	3	19	2	17	2	15	2	12	2	11	2	26	3	21	3	18	2	17	2	16	2	14	2	12	2	11	2	11	2
z9sym	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3	42	5	21	4	15	4	14	4	14	4	7	3	7	3	5	2	5	2
suma bloków	330		245		199		174		157		141		109		92		444		251		204		185		177		134		110		93	
suma warstw	161		128		108		107		101		95		79		72		138		115		104		100		95		90		79		69	

Wykres 4.5. Porównanie metody J_DW z metodą J_DK pod względem liczby bloków (opis w tekście)



Wykres 4.6. Porównanie metody J_DW z metodą J_DK pod względem liczby warstw logicznych (opis w tekście)



Zależność przedstawiona na wykresie 4.6 potwierdza mocną stronę metody J_DW, jaką jest redukcja liczby warstw logicznych w stosunku do metody J_DK. Rozwiązania uzyskiwane metodą J_DK zawierały średnio od kilku do kilkunastu procent (k=3,4) więcej warstw logicznych.

Metoda J_DW jest szczególnie atrakcyjna dla struktur typu PAL, złożonych z bloków logicznych, zawierających 4 lub 8 iloczynów. Dla bloków logicznych zawierających 4 iloczyny metoda J_DW wykorzystała nieznacznie większą liczbę bloków logicznych (2%) równocześnie znacznie ograniczając sumaryczną liczbę warstw logicznych (11%). Jeszcze lepsze rezultaty uzyskano dla bloków zawierających 8 iloczynów. Oprócz typowego dla metody J_DW ograniczenia liczby warstw logicznych (5,6%) uzyskano również redukcję liczby wykorzystanych bloków (5%).

Niestety element dopasowania syntezy do struktury bloków logicznych, zawarty w metodzie J_DW, nie sprawdza się w przypadku funkcji wielowyjściowych. W takim przypadku metoda J_DW powinna być stosowana dla każdej jednowyjściowej funkcji oddzielnie. Niezwykle korzystną cechą dekompozycji kolumnowej jest możliwość bezpośredniego rozszerzenia metody J_DK na funkcje wielowyjściowe (metoda W_DK).

W tabeli 4.6 zawarte są wyniki syntezy (B-liczba bloków; W-liczba warstw) uzyskane metodą klasyczną z minimalizacją liczby warstw (metoda K_LW) oraz trzema zaproponowanymi metodami syntezy z elementami dekompozycji. W wierszach oznaczonych J_DK zawarte są sumaryczne liczby bloków oraz maksymalne liczby warstw logicznych będące wynikiem syntezy jednowyjściowych funkcji metodą J_DK, wykorzystywaną do syntezy odpowiedniego układu testowego. W wierszach z oznaczeniem J_DW zawarte są wyniki uzyskane metodą J_DW, wyznaczone tak samo jak w przypadku metody J_DK. Wiersze z oznaczeniem W_DK zawierają wyniki syntezy, uzyskane metodą dekompozycji kolumnowej, opracowanej dla funkcji wielowyjściowych.

W dolnej części tabeli zawarte są sumaryczne liczby bloków logicznych oraz liczby warstw logicznych uzyskiwane rozpatrywanymi metodami syntezy dla poszczególnych wartości k.

Porównując zaproponowane metody syntezy funkcji z elementami dekompozycji z metodą klasyczną (tab. 4.6) należy podkreślić, że wszystkie metody dekompozycji dają lepsze rezultaty od metody klasycznej, patrząc na liczbę wykorzystywanych bloków logicznych. Uzyskiwane układy zawierają jednak większą liczbę warstw logicznych. Najlepsze wyniki pod względem liczby bloków logicznych uzyskano metodą W_DK. Niestety metoda ta dla wszystkich analizowanych wielkości bloków logicznych dała gorsze wyniki, patrząc na sumaryczną liczbę warstw logicznych.

Spośród wszystkich analizowanych przypadków metodą W_DK znaleziono (patrząc na liczbę bloków logicznych) 42 najlepsze (lepsze od innych) rozwiązania (40%) i tylko 2 rozwiązania (2%) zawierające mniejszą liczbę warstw logicznych. Aż w 96 przypadkach (92%) uzyskano nie gorsze rozwiązania od innych metod.

Metodą J_DK w żadnym przypadku nie znaleziono najlepszego rozwiązania pod względem liczby wykorzystywanych bloków logicznych. Doprowadziła ona jednak do 2 rozwiązań (2%) z najmniejszą liczbą warstw logicznych.

Metodą J_DW znaleziono 8 rozwiązań (8%) wykorzystujących najmniejszą liczbę bloków, z czego 5 rozwiązań uzyskano dla bloków logicznych zawierających 4 lub 8 iloczynów. Doprowadziła ona również do 34 rozwiązań (33%) z najmniejszą liczbą warstw logicznych.

Podsumowaniem powyższych rozważań są wykresy 4.7 i 4.8, przedstawiające porównanie trzech metod dekompozycji: J_DK, J_DW i W_DK. Metody te porównano pod względem liczby bloków oraz liczby warstw logicznych, odnosząc uzyskane rezultaty do wyników uzyskanych metodą klasyczną z minimalizacją liczby warstw. Sumaryczną liczbę bloków (warstw) uzyskiwaną dla różnych wartości k odniesiono do sumarycznej liczby bloków (warstw) uzyskiwanej w metodzie klasycznej.

Wartości przedstawione na wykresie 4.7 wyznaczone zostały z zależności $\frac{x - \sum bloków_{(K\_LW)}}{\sum bloków_{(K\_LW)}}$ 100% poprzez wstawienie za wartość x sumy liczby bloków wykorzystywanych przez rozpatrywany zbiór układów testowych w metodzie J_DK, J_DW i W_DK dla poszczególnych wartości k.

Podobnie wyznaczone zostały wartości przedstawione na wykresie 4.8 na podstawie zależności $\frac{x - \sum warstw_{(K\_LW)}}{\sum warstw_{(K\_LW)}}$ 100%, gdzie za wartości x podstawiono sumy liczby warstw układów uzyskiwanych poszczególnymi metodami syntezy J_DK, J_DW i W_DK dla poszczególnych wartości k.

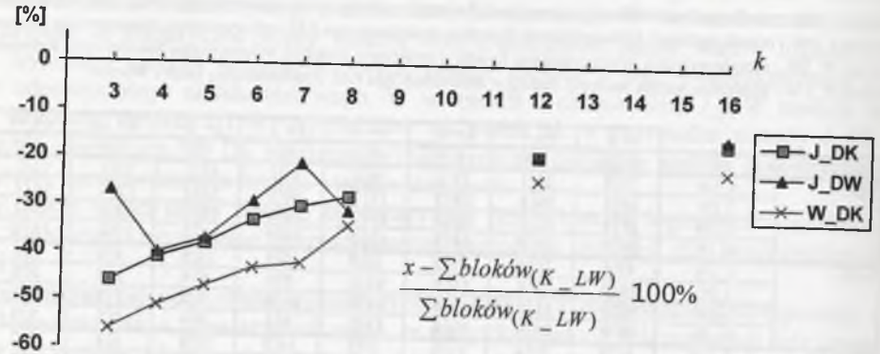
Tabela 4.6

Porównanie zaproponowanych modeli dekompozycji funkcji (opis w tekście)

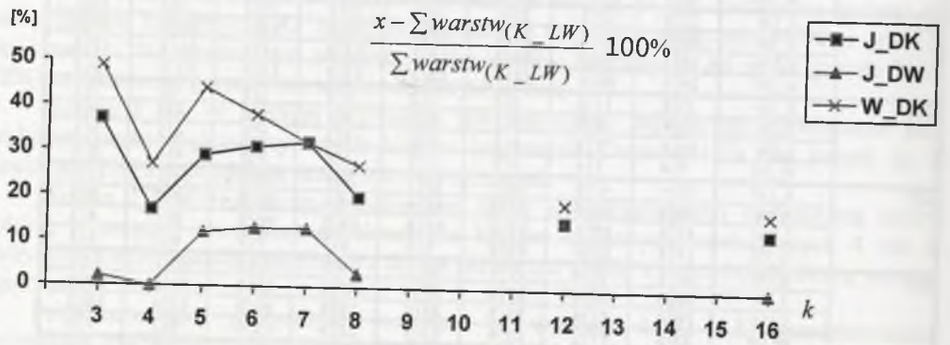
metoda J_DK - metoda realizacji Jednowyjściowej funkcji w strukturach typu PAL wykorzystująca Dekompozycję Kolumnc
metoda J_DW - metoda realizacji Jednowyjściowej funkcji w strukturach typu PAL wykorzystująca Dekompozycję Wierszc
metoda W_DK - metoda realizacji Wielowyjściowej funkcji w strukturach typu PAL wykorzystująca Dekompozycję Kolumnov
metoda K_LW - Klasyczna metoda realizacji funkcji w strukturach typu PAL z minimalizacją Liczby Warstw

		k= 3		k= 4		k= 5		k= 6		k= 7		k= 8		k= 12		k= 16	
Przykład		B	W	B	W	B	W	B	W	B	W	B	W	B	W	B	W
5xp1	K_LW	35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2
	J_DK	26	3	24	3	18	2	17	2	16	2	15	2	12	2	11	2
	J_DW	27	3	21	3	18	2	17	2	16	2	15	2	12	2	11	2
	W_DK	24	5	19	3	18	3	15	2	14	2	14	2	12	2	11	2
9sym	K_LW	43	5	29	4	22	3	18	3	15	3	13	3	8	2	6	2
	J_DK	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3
	J_DW	42	5	21	4	15	4	14	4	14	4	7	3	7	3	5	2
	W_DK	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3
9symml	K_LW	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2
	J_DK	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3
	J_DW	42	5	21	4	15	4	14	4	14	4	7	3	7	3	5	2
	W_DK	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3
clip	K_LW	73	4	49	3	38	3	30	3	26	2	22	2	14	2	12	2
	J_DK	56	5	39	4	31	4	25	4	22	4	19	3	14	2	11	2
	J_DW	65	4	44	3	34	3	28	3	24	3	21	2	14	2	12	2
	W_DK	47	4	32	4	28	4	24	4	19	3	18	3	13	4	10	3
f51m	K_LW	37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2
	J_DK	30	5	24	4	20	3	17	2	16	2	15	2	11	2	10	2
	J_DW	23	4	19	3	16	3	14	2	14	2	14	2	11	2	10	2
	W_DK	24	5	20	4	16	4	13	3	13	3	14	3	11	2	10	2
rd53	K_LW	15	3	11	2	8	2	6	2	6	2	6	2	4	2	3	1
	J_DK	11	4	8	2	6	2	5	2	5	2	5	2	4	2	3	1
	J_DW	11	3	7	2	6	2	5	2	5	2	5	2	4	2	3	1
	W_DK	9	4	7	4	5	3	5	3	5	3	5	3	4	2	3	1
rd73	K_LW	70	4	47	3	36	3	29	3	24	3	20	2	14	2	11	2
	J_DK	30	5	22	4	17	4	15	4	14	4	10	3	9	2	8	2
	J_DW	42	5	17	3	15	3	15	3	15	3	10	3	9	2	8	2
	W_DK	16	4	12	4	10	3	9	3	8	3	7	3	6	2	6	2
rd84	K_LW	142	5	95	4	72	4	58	3	49	3	42	3	28	2	21	2
	J_DK	36	5	26	4	21	4	17	4	17	4	15	4	11	3	10	2
	J_DW	76	4	27	4	23	3	23	3	22	3	15	3	12	3	10	2
	W_DK	23	6	17	5	13	4	12	3	11	3	11	3	9	2	7	2
sao2	K_LW	36	3	24	3	19	2	15	2	14	2	11	2	7	2	7	2
	J_DK	35	4	24	3	19	2	15	2	14	2	11	2	7	2	7	2
	J_DW	29	3	21	3	18	3	13	2	13	2	11	2	7	2	7	2
	W_DK	30	5	23	4	19	4	11	3	10	3	10	3	7	2	7	2
xor5	K_LW	8	2	5	2	4	2	3	2	3	2	3	2	2	2	1	1
	J_DK	4	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1
	J_DW	4	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1
	W_DK	4	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1
z4m1	K_LW	29	4	19	3	15	3	13	2	11	2	9	2	7	2	5	2
	J_DK	13	3	9	3	9	3	9	3	7	2	7	2	6	2	5	2
	J_DW	15	4	9	3	9	3	9	3	8	2	7	2	6	2	5	2
	W_DK	12	6	11	4	10	4	9	4	7	2	7	2	6	2	5	2
z5xp1	K_LW	37	3	27	3	23	2	19	2	17	2	15	2	12	2	11	2
	J_DK	35	4	25	4	23	3	19	2	17	2	15	2	12	2	11	2
	J_DW	26	3	21	3	18	2	17	2	16	2	14	2	12	2	11	2
	W_DK	24	5	19	3	18	3	15	2	15	2	14	2	12	2	11	2
z9sym	K_LW	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2
	J_DK	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3
	J_DW	42	5	21	4	15	4	14	4	14	4	7	3	7	3	5	2
	W_DK	18	9	14	5	11	5	11	5	9	5	9	4	7	3	5	3
	K_LW	609	49	415	41	323	34	261	32	225	31	195	30	135	26	110	24
	J_DK	330	67	245	48	199	44	174	42	157	41	141	36	109	30	92	27
	J_DW	444	50	251	41	204	38	185	36	177	35	135	31	110	30	93	24
	W_DK	267	73	204	52	172	49	148	44	131	41	129	38	103	31	86	28

Wykres 4.7. Porównanie metod syntezy wykorzystujących dekompozycję J_DK, J_DW i W_DK pod względem liczby bloków (opis w tekście)



Wykres 4.8. Porównanie metod syntezy wykorzystujących dekompozycję J_DK, J_DW i W_DK pod względem liczby warstw (opis w tekście)



Podsumowując wyniki eksperymentów należy jeszcze raz podkreślić, że:

- metody bazujące na dekompozycji dają zdecydowanie lepsze rezultaty od metody klasycznej pod względem liczby wykorzystywanych bloków logicznych; w przypadku niewielkich bloków logicznych typu PAL (k=3,4) redukcja sięga około 50 %,
- występująca w metodach bazujących na dekompozycji redukcja liczby bloków okupiona jest dość znacznym przyrostem liczby warstw logicznych; najlepszą metodą, patrząc na liczbę warstw, jest metoda J_DW, która jednak jest gorsza od metody klasycznej z minimalizacją liczby warstw, szczególnie dla bloków zawierających różną od 2^i liczbę iloczynów,
- dekompozycja wielowyjściowych funkcji (metoda W_DK) prowadzi w większości przypadków do najlepszych rozwiązań pod względem liczby bloków logicznych,
- metoda J_DW jest szczególnie skuteczna dla struktur złożonych z bloków logicznych typu PAL, zawierających 2^i iloczynów w przypadku chęci ograniczenia liczby warstw logicznych.

5. ZŁOŻONE STRATEGIE SYNTEZY FUNKCJI W STRUKTURACH TYPU PAL

Przedstawione w poprzednich rozdziałach metody syntezy mogą być wykorzystywane osobno lub mogą stanowić elementy złożonych strategii syntezy, przeznaczonych dla struktur typu PAL. Najczęściej strategie syntezy ukierunkowane są na minimalizację dwóch parametrów: liczby wykorzystywanych bloków logicznych lub liczby warstw logicznych. Istotna jest również złożoność algorytmów i związany z nią czas syntezy.

5.1. Dobór elementów syntezy pod kątem optymalizacji liczby bloków logicznych lub liczby warstw logicznych

Przed uzasadnieniem doboru elementów strategii syntezy, prowadzących do minimalizacji liczby bloków lub liczby warstw logicznych, warto podsumować wyniki przedstawione w poprzednich rozdziałach pracy.

Nie ulega najmniejszej wątpliwości, że najlepsze, pod względem liczby warstw logicznych, są metody syntezy, wykorzystujące wyjściowe bufory trójstanowe. Najlepsza pod tym względem jest metoda jednowarstwowej realizacji funkcji, wykorzystująca trójstanowe bufory wyjściowe (metoda B_JW). Niestety, nie zawsze prowadzi ona do rozwiązania. Nic nie stoi jednak na przeszkodzie, by w tych przypadkach użyć metody p-warstwowej realizacji funkcji, wykorzystującej trójstanowe bufory wyjściowe (metoda B_PW), za pomocą której można znaleźć rozwiązanie zawierające najmniejszą liczbę warstw logicznych. Takie podejście, zawarte w strategii B_JPW przedstawionej w rozdziale 3, prowadzi do rozwiązań zawierających minimalną liczbę warstw logicznych. Wykorzystując strategię B_JPW trzeba się jednak liczyć ze znacznym wzrostem liczby wykorzystywanych bloków logicznych w stosunku do innych metod. Strategia B_JPW wykorzystuje średnio około 1,5 razy więcej bloków logicznych niż metoda klasyczna.

Warto zwrócić uwagę, że unikalną cechą zaproponowanej metody p-warstwowej realizacji funkcji, wykorzystującej trójstanowe bufory wyjściowe (B_PW), jest możliwość poszukiwania rozwiązań z zadaną liczbą warstw logicznych. Rozwiązania z większą liczbą warstw logicznych znajdowane są w krótszym czasie i wykorzystują mniejszą liczbę bloków logicznych.

Prostota opracowanych algorytmów (B_JW, B_PW) sprawia, że mogą one być z powodzeniem wykorzystywane jako elementy złożonych strategii syntezy, służące do minimalizacji liczby warstw logicznych dla wszystkich układów typu PAL z trójstanowymi buforami wyjściowymi.

Najcenniejszą cechą metod realizacji funkcji wielowyjściowych wykorzystujących grafy wyjść (metoda W_LB oraz W_LW) jest prostota algorytmów i związany z nią krótki czas syntezy, nieznacznie dłuższy od czasu syntezy metodą klasyczną z minimalizacją liczby warstw logicznych (metoda K_LW). Metody te nie prowadzą do rewelacyjnych wyników, lecz z powodzeniem można je połączyć np. z metodą klasyczną, wybierając ostatecznie najlepsze rozwiązanie. Takie postępowanie jest powszechnie wykorzystywane w wielu

komercyjnych systemach syntezy do wyszukiwania najlepszych rozwiązań, np. analizując realizacje funkcji z warunków działania i niedziałania. Łącząc metodę realizacji funkcji wielowyjściowych uzyskiwaną po analizie grafu wyjść z realizacją klasyczną (rozdział 2) uzyskuje się kilkuprocentową redukcję liczby bloków logicznych w stosunku do metody klasycznej. Redukcja ta niestety jest okupiona kilkunastoprocentowym wzrostem liczby warstw logicznych.

Warto podkreślić, że metody realizacji funkcji wielowyjściowych wykorzystujące grafy wyjść są szczególnie atrakcyjne dla struktur z niewielkimi blokami logicznymi typu PAL (k=3,4,5). W takich przypadkach występuje redukcja liczby bloków logicznych w stosunku do metody klasycznej na poziomie około 10%.

W większości analizowanych przypadków, niezależnie od wielkości bloków logicznych typu PAL, zastosowanie metod realizacji funkcji wielowyjściowych, wykorzystujących grafy wyjść, prowadzi do ekspansji liczby warstw logicznych w stosunku do metody klasycznej. Okazuje się jednak, że mankament ten można wyeliminować w układach, w których występują trójstanowe bufory wyjściowe. W metodach realizacji funkcji wielowyjściowych, przedstawionych w rozdziale 2 (W_LB, W_LW), wyszukiwane w kolejnych krokach grupy wielowyjściowych implikantów o identycznych częściach wyjściowych są realizowane metodą klasyczną. Nic nie stoi jednak na przeszkodzie, żeby do realizacji tych grup implikantów użyć metody syntezy, wykorzystującej trójstanowe bufory wyjściowe. Taka strategia doprowadziłaby do znacznej redukcji liczby warstw logicznych wynikowych układów.

Najefektywniejszymi metodami syntezy pod względem liczby bloków logicznych są metody wykorzystujące elementy dekompozycji. Niestety duża złożoność zaproponowanych algorytmów ogranicza ich wykorzystanie do układów zawierających co najwyżej kilkanaście wejść. Najlepsze rezultaty daje metoda dekompozycji funkcji wielowyjściowych oparta na dekompozycji kolumnowej (W_DK). Metodą tą uzyskuje się redukcję liczby bloków w stosunku do metody klasycznej na poziomie 20-30%.

Elementy dopasowania dekompozycji do struktury bloków logicznych typu PAL zawarte są w metodzie J_DW. Metoda ta prowadzi do nieznacznie gorszych rozwiązań od metody W_DK pod względem liczby bloków logicznych. Uzyskuje się jednak rozwiązania lepsze pod względem liczby warstw logicznych.

Analiza wyników licznych eksperymentów, częściowo przedstawionych w poprzednich rozdziałach oraz między innymi w pracach [Kani99g, Kani00a, Kani00f, Kani00i, Kani00g, Kani01c, Kani02c], pozwala na sformułowanie strategii syntezy, w których projektowany układ optymalizowany jest pod kątem liczby bloków logicznych lub liczby warstw logicznych, przeznaczonych dla struktur typu PAL z i bez trójstanowych buforów wyjściowych.

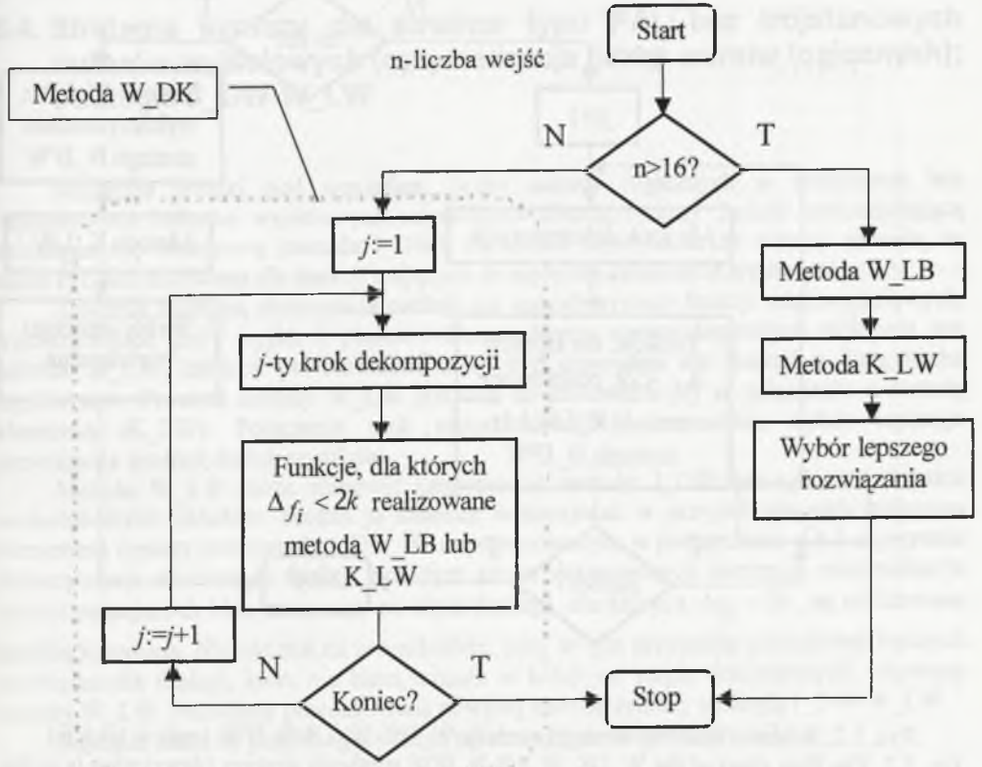
5.2. Strategia syntezy dla struktur typu PAL bez trójstanowych buforów wyjściowych (optymalizacja liczby bloków logicznych); strategia W_DK-W_LB

Najlepsze wyniki pod względem liczby bloków logicznych uzyskano stosując metodę syntezy funkcji wielowyjściowych wykorzystującą dekompozycję kolumnową (metoda W_DK). Złożoność algorytmów sprawia jednak, że metoda ta może być stosowana dla funkcji mających co najwyżej kilkanaście argumentów.

Znacznie mniejszą złożoność mają metody syntezy funkcji wielowyjściowych wykorzystujące grafy wyjść. Z punktu widzenia liczby bloków logicznych najlepsza jest metoda W_LB, która z powodzeniem może być stosowana dla funkcji z dużą liczbą argumentów. Prostota metody W_LB sprawia, że może być ona stosowana łącznie z metodą klasyczną. Efektem tego połączenia może być wybór lepszego rozwiązania spośród dwóch wyników uzyskiwanych metodą W_LB i K_LW.

Metoda W_LB może stanowić uzupełnienie metody W_DK nie tylko w obszarze rozbudowanych układów. Można ją również wykorzystać w przypadkach, gdy głównym elementem syntezy jest metoda W_DK. W zaproponowanym w podrozdziale 4.4.4 algorytmie dekompozycji kolumnowej funkcji wielowyjściowej, po j-tym kroku dekompozycji następuje minimalizacja funkcji opisujących blok związany i wolny, po czym funkcje, dla których $\Delta_{f_i} < 2k$, są realizowane metodą klasyczną. Nic nie stoi na przeszkodzie, żeby w tej sytuacji szukać lepszych rozwiązań dla funkcji, które nie biorą udziału w kolejnym etapie dekompozycji, używając metody W_LB.

Nazwijmy przedstawioną powyżej strategię syntezy strategią W_DK-W_LB. Schemat blokowy przybliżający istotę tej strategii przedstawia rys. 5.1.

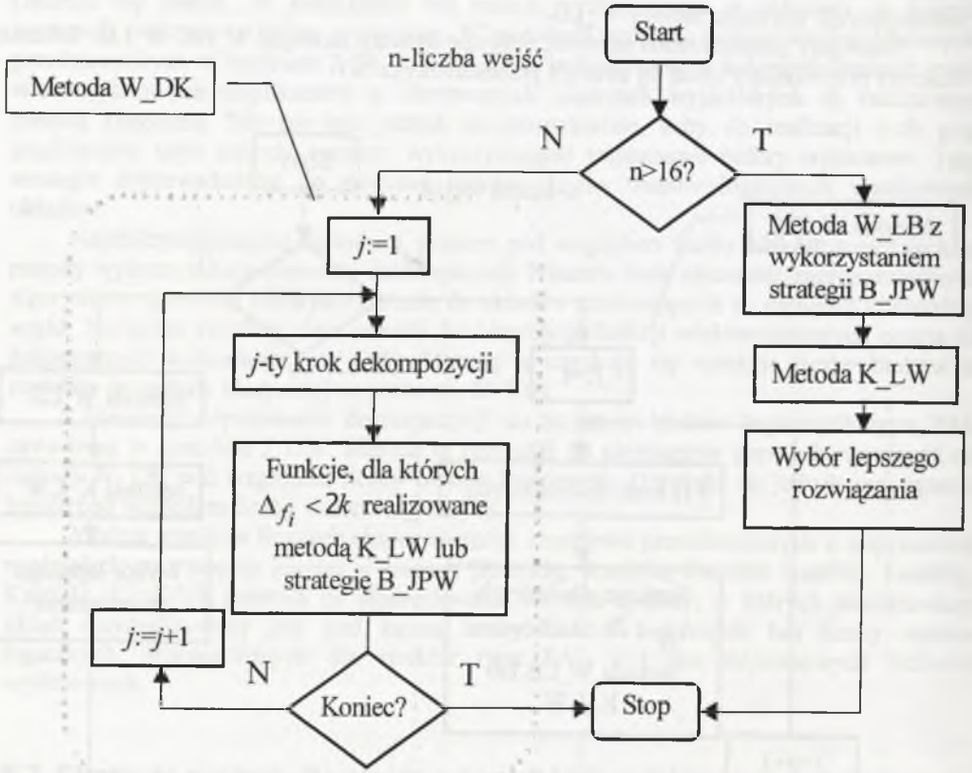


Rys. 5.1. Schemat blokowy strategii syntezy W_DK-W_LB (opis w tekście)
Fig. 5.1. The flow chart of the W_DK-W_LB synthesis strategy (description is in the text)

5.3. Strategia syntezy dla struktur typu PAL z trójstanowymi buforami wyjściowymi (optymalizacja liczby bloków logicznych); strategia W_DK-W_LB -B_JPW

W przypadku struktur z trójstanowymi buforami wyjściowymi można ograniczyć liczbę warstw logicznych uzyskiwanych układów, rozbudowując strategię W_DK-W_LB o element umożliwiający wykorzystanie trójstanowych buforów wyjściowych, tzn. strategię B_JPW. Nazwijmy taką metodę syntezy strategią W_DK-W_LB-B_JPW.

Kiedy warto użyć trójstanowych buforów wyjściowych po to, żeby zmniejszyć liczbę warstw logicznych?



Rys. 5.2. Schemat blokowy strategii syntezy W_DK-W_LB-B_JPW (opis w tekście)
Fig. 5.2. The flow chart of the W_DK-W_LB-B_JPW synthesis strategy (description is in the text)

Strategia B_JPW może stanowić uzupełnienie metody W_DK w przypadku funkcji, które nie są brane pod uwagę w kolejnych krokach poszukiwania dekompozycji kolumnowej, tzn. takich, dla których $\Delta_{f_i} < 2k$. Metodę klasyczną, wykorzystywaną w tych przypadkach (podrozdział 4.4.4 - algorytm dekompozycji kolumnowej funkcji wielowyjściowej), można z powodzeniem zastąpić strategią B_JPW. Nic nie stoi na przeszkodzie by poszukać rozwiązania metodą K_LW i B_JPW, po czym wybrać lepsze rozwiązanie, w pierwszej kolejności patrząc na liczbę bloków, a potem liczbę warstw logicznych.

Strategia B_JPW stanowi ewentualne uzupełnienie metody W_LB. Wyszukiwane w kolejnych krokach grupy wielowyjściowych implikantów o identycznych częściach wyjściowych mogą być realizowane strategią B_JPW. Uzyskane rozwiązanie można porównać z rozwiązaniem otrzymanym metodą klasyczną. Wybierane jest oczywiście rozwiązanie wykorzystujące mniejszą liczbę bloków logicznych. Jeżeli liczby bloków logicznych są równe, wybierane jest rozwiązanie z mniejszą liczbą warstw logicznych. Takie podejście w wielu przypadkach prowadzi do znacznej redukcji liczby warstw logicznych.

Schemat blokowy strategii W_DK-W_LB-B_JPW przedstawia rys. 5.2.

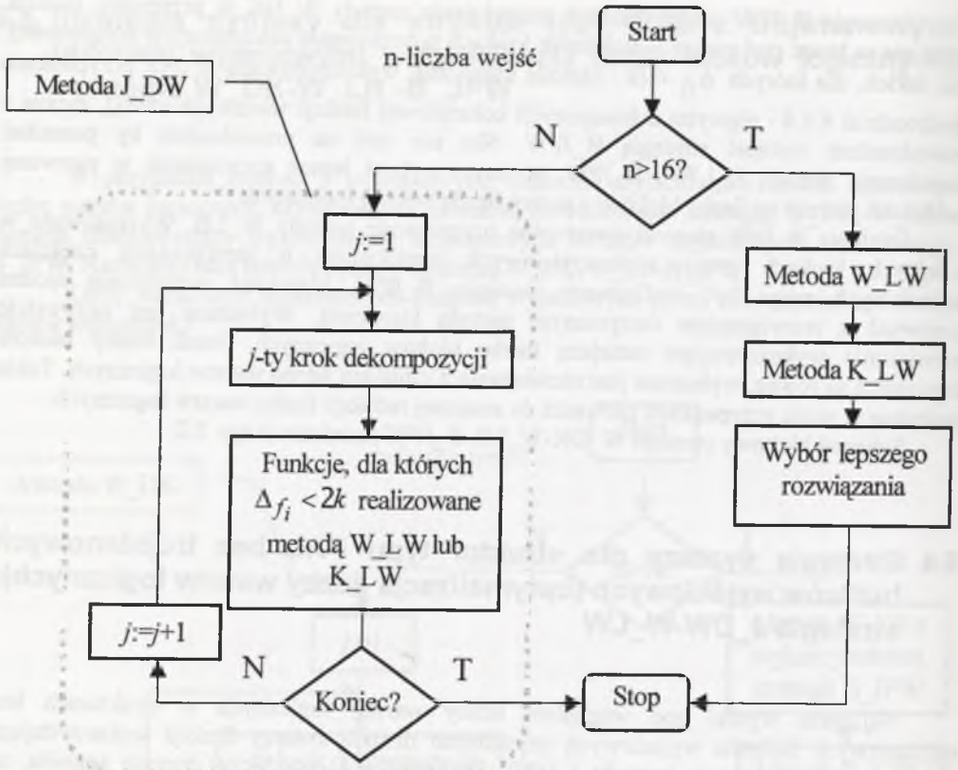
5.4. Strategia syntezy dla struktur typu PAL bez trójstanowych buforów wyjściowych (optymalizacja liczby warstw logicznych); strategia J_DW-W_LW

Najlepsze wyniki pod względem liczby warstw logicznych w strukturach bez trójstanowych buforów wyjściowych uzyskiwano metodą syntezy funkcji wykorzystującą dekompozycję wierszową (metoda J_DW). Złożoność algorytmów tej metody sprawia, że może być ona stosowana dla funkcji mających co najwyżej kilkanaście argumentów.

Znacznie mniejszą złożonością cechują się metody syntezy funkcji wielowyjściowych, wykorzystujące grafy wyjść. Z punktu widzenia liczby warstw logicznych najlepsza jest metoda W_LW, która z powodzeniem może być stosowana dla funkcji z dużą liczbą argumentów. Prostota metody W_LW pozwala na stosowanie jej w połączeniu z metodą klasyczną (K_LW). Połączenie tych metod umożliwia ostatecznie wybór lepszego rozwiązania spośród dwóch wyników.

Metoda W_LW może stanowić uzupełnienie metody J_DW nie tylko w obszarze rozbudowanych układów. Można ją również wykorzystać w przypadkach, gdy głównym elementem syntezy jest metoda J_DW. W zaproponowanym w podrozdziale 4.5.3 algorytmie dekompozycji wierszowej funkcji po j-tym kroku dekompozycji następuje minimalizacja funkcji opisujących blok związany, po czym funkcje, dla których $\Delta_{f_i} < 2k$, są realizowane metodą klasyczną. Nic nie stoi na przeszkodzie, żeby w tym przypadku poszukiwać lepszych rozwiązań dla funkcji, które nie biorą udziału w kolejnym etapie dekompozycji, używając metody W_LW. Nazwijmy przedstawioną powyżej metodę syntezy strategią J_DW-W_LW.

Schemat blokowy przybliżający istotę tej strategii przedstawia rys. 5.3.

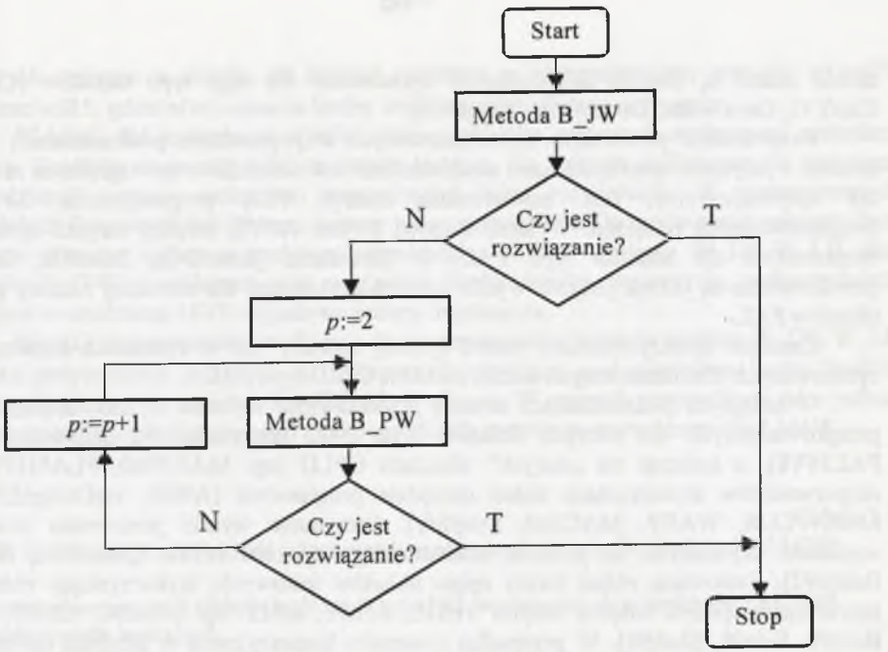


Rys. 5.3. Schemat blokowy strategii syntezy J_DW-W_LW (opis w tekście)
Fig. 5.3. The flow chart of the J_DW-W_LW synthesis strategy (description is in the text)

5.5. Strategia syntezy dla struktur typu PAL z trójstanowymi buforami wyjściowymi (optymalizacja liczby warstw logicznych); strategia B_JPW

Bezkonkurencyjną strategią pod względem liczby warstw logicznych, dla struktur z trójstanowymi buforami wyjściowymi, jest strategia B_JPW. Głównym elementem tej strategii jest metoda jednowarstwowej realizacji funkcji, wykorzystująca trójstanowe bufory wyjściowe (metoda B_JW). W przypadku gdy metoda ta nie prowadzi do rozwiązania, wykorzystywana jest metoda p-warstwowej realizacji funkcji (metoda B_PW).

Schemat blokowy przybliżający istotę strategii B_JPW przedstawia rys. 5.4.



Rys. 5.4. Schemat blokowy strategii syntezy B_JPW (opis w tekście)
Fig. 5.4. The flow chart of the B_JPW synthesis strategy (description is in the text)

5.6. Porównanie zaproponowanych strategii syntezy z innymi metodami

Zaproponowane w poprzednich podrozdziałach strategie syntezy użyto do licznych eksperymentów. Próbowano porównać je z innymi komercyjnymi oraz akademickimi systemami syntezy.

Systemy akademickie ukierunkowane są przede wszystkim na układy FPGA typu tablicowego lub multiplekserowego [AbouB93, BrayR87, ChanM96, CongD94, CongW98, FrancR91, HuanJ00, JózwC99, Karp91a, Karp91b, LaiP96, LeglW95, Łuba94, MurgS91, NowiŁ97, Sasa93b, SentS92, SicaC91, Volf97, WanP92, YangC02]. Znane są również metody syntezy, opracowane bezpośrednio dla struktur CPLD [AndeB98, ChenH02, KimB99, KimK00, KimK01]. Wykorzystano w nich jednak elementy syntezy opracowane dla układów FPGA typu tablicowego, wplatając w poszczególne etapy ograniczenia związane z zasobami układów CPLD.

Bardzo ciekawe podejście zaprezentowane jest w pracach [AndeB98, ChenH02]. W procesie syntezy skorzystano z metod opracowanych dla układów typu tablicowego, zastępując bloki tablicowe specyficznymi komórkami typu PLA. Optymalną strukturę tych komórek dobrano na podstawie eksperymentów przedstawionych w pracach [KoulG91, KoulG92]. Okazało się, że ze względu na nieefektywne wykorzystywanie zasobów logicznych bloków typu tablicowego korzystniejsze pod względem powierzchni byłoby konstruowanie bloków logicznych w postaci odpowiednich matryc PLA. Tym bardziej, że od

dawna znane są metody dekompozycji opracowane dla tego typu układów [ChieM88, CiesY92, DevaW88a, DevaW88b, YangC89].

Bezpośrednie porównanie zaprezentowanych w poprzednich podrozdziałach strategii syntezy z przytoczonymi systemami akademickimi jest niemożliwe ze względu na nieco inny cel optymalizacyjny, tzn. powierzchnię matryc PLA proporcjonalną do liczby programowalnych połączeń. Wyjątek stanowi system ASYL, między innymi opracowany bezpośrednio dla struktur typu PAL. W literaturze [SaucS90a, Sauc90b, Babbc92] przedstawione są jednak jedynie wyniki syntezy wykonanej dla szerokiej rodziny prostych układów PAL.

Znacznie szerszy wachlarz metod syntezy zawarty jest w systemach komercyjnych, opracowanych dla różnorodnych rodzin układów CPLD typu PAL.

W kolejnych podrozdziałach zostaną przedstawione wybrane wyniki eksperymentów, przeprowadzonych dla różnych układów typu PAL, poczynając od najprostszych (np. PAL16V8), a kończąc na „dużych" układach CPLD (np. MAX7000, FLASH370). Do eksperymentów wykorzystano różne narzędzia programowe (ABEL, ispDesignEXPERT, MAX+PLUS, WARP, MACHxl, Synplify). Otrzymane wyniki porównano również z wynikami uzyskanymi za pomocą akademickiego systemu ASYL [SaucS90a, Sauc90b, Babbc92]. Testowano różne formy opisu układów testowych, wykorzystując różnorodne języki opisu sprzętu, między innymi VHDL, AHDL, ABEL itp. [Abel90, Alte95, Alte96, Bout98, Salc98, Skah96]. W przypadku systemów komercyjnych w tabelach umieszczono zawsze najlepsze rozwiązania, mając na uwadze założony cel optymalizacyjny. Porównano również opracowane strategie syntezy z metodami akademickimi, przeznaczonymi dla matryc PLA. W tym celu wyznaczono parametry, charakteryzujące powierzchnię zajmowaną przez uzyskiwane układy.

5.6.1. Proste układy typu PAL (ABEL, MACHxl, ispDesignEXPERT, Synplify, ASYL)

Mimo iż proste układy PAL (16V8, 22V10, 26V12) wychodzą z użycia, autor zdecydował się przeprowadzić eksperymenty dla tych struktur, chcąc porównać opracowane strategie syntezy z istniejącymi na rynku starszymi i nowszymi narzędziami syntezy [Abel90, Mach95, SaucS90a, SaucS90b, Synp00a]. Najczęściej synteza logiczna zaimplementowana w tych narzędziach wykonywana jest dla wybranej w początkowym etapie struktury. Czasami typ struktury dobiera się w końcowym etapie syntezy spośród układów zawartych w bibliotece elementów. Możliwe jest również wykonanie syntezy opisując projektowany układ za pomocą języków opisu sprzętu (VHDL, Verilog). W tym przypadku w pierwszym etapie wykonywana jest synteza w oderwaniu od konkretnej struktury programowalnej. Najczęściej po początkowym etapie syntezy związanym z kompilacją opisu używa się firmowych narzędzi programowych, umożliwiających wykonanie ostatnich etapów syntezy, tzn. dopasowania projektu do wybranej struktury programowalnej i utworzenia zbiorów konfiguracyjnych.

W większości przypadków, gdy projektowany układ nie mieści się w wybranej strukturze, system syntezy nie potrafi podzielić go na mniejsze części, realizowane w poszczególnych, prostych układach PLD. W takiej sytuacji zachodzi konieczność wyboru większej struktury, zapewniającej realizowalność projektowanego układu.

W pierwszej kolejności wykonano eksperymenty dla najprostszych struktur PAL16V8. Ponieważ układ PAL 16V8 ma 8 wejść, 8 wyprowadzeń dwukierunkowych oraz 8 bloków logicznych typu PAL, zawierających 8 iloczynów, z analizowanego zbioru układów

testowych wybrano te układy, dla których spełnione są następujące trzy warunki: $n+m \leq 16$, $m \leq 8$ oraz $n \leq 15$, gdzie $n(m)$ oznacza liczbę wejść(wyjść) układu testowego.

W tabeli 5.1 zawarte są wyniki syntezy układów testowych wybranymi metodami syntezy. Znajdują się w niej tylko te układy testowe, dla których analizowanymi metodami (narzędziami) syntezy znaleziono przynajmniej jedno rozwiązanie. W rozpatrywanych narzędziach firmowych uzyskiwano zawsze jedno rozwiązanie. Do porównania wybrano dwie strategie syntezy: najlepszą pod względem liczby bloków strategię W_DK-W_LB oraz strategię B_JPW – najlepszą pod względem liczby warstw logicznych, wykorzystującą istniejące w strukturze 16V8 wyjściowe bufory trójstanowe.

Wyniki eksperymentów wskazują, że zaproponowana strategia syntezy W_DK-W_LB w kilku przypadkach prowadzi do najlepszych rozwiązań pod względem liczby bloków logicznych (*9sym, 9symml, rd53, rd73, z4ml, z9sym*). W czterech przypadkach jako jedyna metoda umożliwiła realizację odpowiedniego układu testowego w strukturze PAL16V8.

Tabela 5.1

Wyniki syntezy logicznej układów testowych przeprowadzonych dla struktur PAL16V8

B - liczba wykorzystanych bloków logicznych typu PAL występujących w strukturze PAL16V8
W - liczba warstw logicznych
i - liczba wejść układu testowego
o - liczba wyjść układu testowego

W_DK-W_LB - strategia realizacji **W**ielowyjściowej funkcji w strukturach typu PAL, wykorzystująca **D**ekompozycję **K**olumnową i metodę realizacji **W**ielowyjściowej funkcji, bazująca na analizie grafów wyjść, w której układ optymalizowany jest pod kątem **L**iczby **B**loków logicznych, użytą do realizacji bloku związanego i wolnego (k=8);

B_JPW – strategia syntezy wykorzystująca trójstanowe bufory wyjściowe (metoda B_JW; w przypadku braku rozwiązania metoda B_PW zapewniająca minimalną liczbę warstw) (k=7)

			W_DK-W_LB		B_JPW		ABEL		MACHxl		ispDesignEXPERT	
Układ testowy	i	o	B	W	B	W	B	W	B	W	B	W
9sym	9	1	8	4	-	-	-	-	-	-	-	-
9symml	9	1	8	4	-	-	-	-	-	-	-	-
con1	7	2	2	1	2	1	2	1	2	1	2	1
misex1	8	7	7	1	7	1	7	1	7	1	7	1
rd53	5	3	5	2	7	1	6	2	-	-	6	2
rd73	7	3	7	3	-	-	-	-	-	-	-	-
Sqn	7	3	7	2	8	1	7	2	-	-	-	-
squar5	5	8	8	1	-	-	8	1	8	1	8	1
xor5	5	1	2	2	4	1	3	2	-	-	2	2
z4ml	7	4	7	2	-	-	-	-	-	-	8	2
z9sym	9	1	8	4	-	-	-	-	-	-	-	-

Strategia B_JPW w sześciu przypadkach nie doprowadziła do rozwiązania. Wynika to z faktu wykorzystywania większej liczby bloków logicznych niż strategia W_DK-W_LB. Dodatkowo w strukturze PAL16V8 wykorzystanie trójstanowego bufora wyjściowego wiąże się z użyciem jednego iloczynu, który w strategii W_DK-W_LB wchodził w skład bloku logicznego typu PAL [Adva96]. Mimo tego strategia B_JPW prowadzi w trzech przypadkach (*rd53, sqn, xor5*) do najlepszych rozwiązań pod względem szybkości.

W zasadzie po wykonaniu syntezy dowolną metodą, wykonaną w oderwaniu od konkretnej struktury, możliwy jest w końcowym etapie dobór odpowiedniego układu typu PAL. Narzędzia syntezy np. ABEL, MACHxl itp. [Abel90, Mach95] mają opracowane moduły doboru układu spośród wskazanej grupy układów.

Zaprezentowane w pracy strategie syntezy zostały opracowane dla układów typu PAL z identyczną liczbą iloczynów w poszczególnych blokach. Nic nie stoi jednak na przeszkodzie, aby rozszerzyć zaproponowane metody na grupę układów z nierównomiernym rozdziałem iloczynów pomiędzy poszczególne komórki (np. 22V10). Pewne zagadnienia związane z syntezą logiczną przeznaczoną dla takich układów zostały przedstawione w pracach [Kani99f, Kani01c]. Ogólna koncepcja syntezy logicznej dla układów typu PAL z nierównomiernym rozdziałem iloczynów polega na takim doborze bloków, aby minimalizować liczbę niewykorzystanych iloczynów. W tej sytuacji można porównać tak skomponowaną strategię syntezy z narzędziami firmowymi. W zaproponowanych strategiach syntezy (W_DK-W_LB, B_JPW) interakcyjne przyporządkowywanie poszczególnych bloków typu PAL do realizacji uzyskiwanych funkcji logicznych wykonano po kolejnych krokach syntezy. Kierowano się oczywiście zasadą minimalizacji liczby nie wykorzystywanych iloczynów i próbowano dobrać jak najprostsze układy spośród pięciu najpopularniejszych struktur: 16V8, 20V8, 18V10, 22V10, 26V12. Wyniki najciekawszych eksperymentów przedstawiono w tab. 5.2.

Porównując uzyskane wyniki należy zauważyć, że strategia syntezy W_DK-W_LB w 7 przypadkach doprowadziła do najlepszych rozwiązań. W kilku sytuacjach jest jedyną metodą umożliwiającą realizację odpowiedniego układu testowego w wybranej strukturze typu PAL (np. realizacja układu testowego *rd84* w strukturze 22V10). Tak jak poprzednio istnieją sytuacje, w których strategia B_JPW prowadzi do ograniczenia liczby warstw logicznych (np. *rd53*, *z4ml*). Skutkiem ubocznym jest czasami konieczność wyboru większej struktury (*squar5*). Dzieje się tak, ponieważ w tego typu strukturach wykorzystanie buforów trójstanowych wiąże się ze zmniejszeniem o jeden liczby iloczynów, znajdujących się w bloku logicznym typu PAL.

Podobne podejście, umożliwiające dobór odpowiedniej struktury typu PAL, zaimplementowano w systemie ASYL [AbouB93, BabbC92, SaucS90a, SaucS90b,]. W pracy [BabbC92] zostały przedstawione eksperymenty dla szerokiej gamy układów typu PAL firmy AMD, wśród których najbardziej rozbudowaną strukturą jest układ PAL22V10.

Porównanie dwóch strategii syntezy: W_DK-W_LB oraz B_JPW z systemem ASYL przedstawiono w tab. 5.3.

Tabela 5.2

Wyniki syntezy logicznej z doborem struktury spośród pięciu układów: PAL16V8, PAL20V8, PAL18V10, PAL22V10, PAL26V12 (opis w tekście)

U - typ układu wykorzystanego do realizacji odpowiedniego układu testowego
W - liczba warstw logicznych
i - liczba wejść układu testowego
o - liczba wyjść układu testowego

W_DK-W_LB - strategia realizacji **W**ielowyjściowej funkcji w strukturach typu PAL wykorzystująca **D**ekompozycję **K**olumnową i metodę realizacji **W**ielowyjściowej funkcji bazującą na analizie grafu wyjść, w której układ optymalizowany jest pod kątem **L**iczby **B**loków logicznych, użytą do realizacji bloku związanego i wolnego (k=8);

B_JPW - strategia syntezy wykorzystująca trójstanowe bufory wyjściowe (metoda B_JW; w przypadku braku rozwiązania metoda B_PW zapewniająca minimalną liczbę warstw, (k=7))

			W_DK-W_LB		B_JPW		Synplify + ispDesignEXPERT		Synplify + MACHxl	
			16V8, 20V8, 18V10, 22V10, 26V12		16V8, 20V8, 18V10, 22V10, 26V12		16V8, 20V8, 18V10, 22V10, 26V12		16V8, 20V8, 22V10, 26V12	
Układ testowy	i	o	U	W	U	W	U	W	U	W
9sym	9	1	16V8	4	-	-	-	-	22V10	4
9symml	9	1	16V8	4	-	-	-	-	22V10	4
Con1	7	2	16V8	1	16V8	1	16V8	1	16V8	1
Inc	7	9	22V10	1	22V10	1	26V12	2	22V10	1
Misex1	8	7	16V8	1	16V8	1	16V8	1	16V8	1
Rd53	5	3	16V8	2	16V8	1	16V8	2	22V10	1
Rd73	7	3	16V8	3	-	-	-	-	22V10	2
Rd84	8	4	22V10	3	-	-	-	-	-	-
Root	8	5	22V10	2	-	-	-	-	22V10	3
Sao2	10	4	18V10	3	-	-	-	-	22V10	2
Sqn	7	3	16V8	2	16V8	1	26V12	3	22V10	1
Squar5	5	8	16V8	1	18V10	1	16V8	1	16V8	1
Xor5	5	1	16V8	2	16V8	1	16V8	2	22V10	1
z4ml	7	4	16V8	2	18V10	1	16V8	2	22V10	2
z9sym	9	1	16V8	4	-	-	-	-	22V10	4

Dla analizowanego zbioru układów testowych sumaryczna liczba układów użytych w strategii W_DK-W_LB jest o 25% mniejsza od sumarycznej liczby układów wykorzystanych w syntezie systemem ASYL. Redukcja liczby układów okupiona jest wzrostem sumarycznej liczby warstw logicznych. W strategii W_DK-W_LB uzyskano o 44 % większą liczbę warstw logicznych niż w systemie ASYL. Odwrotna sytuacja występuje w strategii B_JPW. Strategia ta (w stosunku do systemu ASYL) prowadzi do zmniejszenia sumarycznej liczby warstw (39%), wykorzystując o 79% więcej układów.

Tabela 5.3

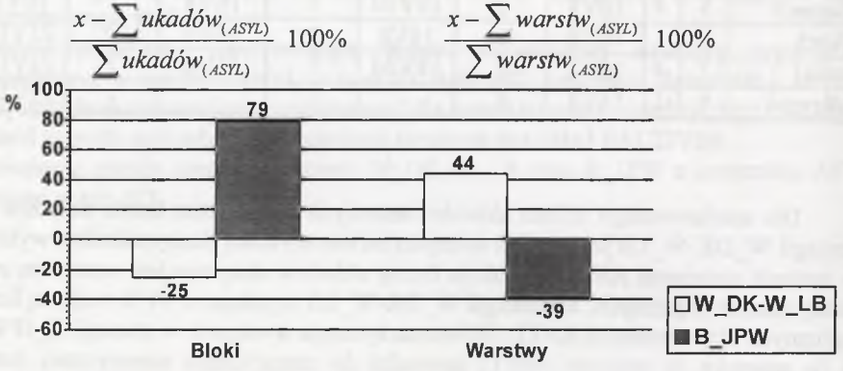
Porównanie wybranych strategii syntezy z systemem ASYL

U-liczba układów programowalnych, wybranych z rodziny prostych układów PAL, wśród których najbardziej rozbudowany był układ 22V10
W-liczba warstw logicznych; i(o) – liczba wejść (wyjść) układu testowego

			W_DK-W_LB		B_JPW		ASYL [BabbC92]	
Układ testowy	i	o	U	W	U	W	U	W
5xp1	7	10	2	2	2	1	2	2
9sym	9	1	1	4	6	1	1	2
bw	5	28	3	1	3	1	3	1
clip	9	5	2	3	3	1	3	2
f51m	8	8	2	3	2	1	2	2
misex1	8	7	1	1	7	1	1	1
misex2	25	18	3	1	3	1	3	1
rd53	5	3	1	2	1	1	1	1
rd73	7	3	1	3	4	1	3	2
rd84	8	4	1	3	8	1	4	2
sao2	10	4	1	3	4	1	1	2
		Σ	18	26	43	11	24	18

Przedstawione wyniki zobrazowane są na wykresie 5.1. Wartości przedstawione na nim wyznaczone zostały z zależności $\frac{x-\sum ukadów_{(ASYL)}}{\sum ukadów_{(ASYL)}}$ 100% oraz $\frac{x-\sum warstw_{(ASYL)}}{\sum warstw_{(ASYL)}}$ 100%, gdzie x oznacza odpowiednią sumę liczby układów lub warstw w systemie ASYL, strategii W_LB_W_LB oraz strategii B_JPW.

Wykres 5.1. Porównanie wyników syntezy uzyskanych w systemie ASYL z wynikami uzyskanymi za pomocą strategii W_DK-W_LB oraz B_JPW pod względem liczby układów oraz liczby warstw (opis w tekście)



5.6.2. Układy Classic firmy Altera (MAX+PLUS)

Znacznie bardziej złożonymi układami typu PAL są struktury produkowane przez firmę Altera, należące do układów rodziny Classic. Złożoność tych układów jest kilkakrotnie większa od złożoności układu 22V10. Najbardziej rozbudowanym przedstawicielem tej rodziny jest układ EP1810 o złożoności równej 1800 bramek (złożoność 4,5 razy większa od złożoności struktury 22V10). Układ ten zawiera 48 bloków logicznych typu PAL z ośmioma iloczynami. Z każdym blokiem związany jest dodatkowo jeden iloczyn, który jest przeznaczony do sterowania trójstanowym buforem wyjściowym [Alte99]. Firma Altera oferuje system syntezy MAX+PLUS, który wykorzystano do porównania z zaproponowanymi strategiami syntezy (W_DK-W_LB; B_JPW). Opis układów testowych był wprowadzany w postaci tablicy prawdy, przedstawionej w formacie AHDL (Altera Hardware Description Language) [Alte95]. Taki opis układów prowadził w systemie MAX+PLUS do najlepszych wyników. Wyniki eksperymentów przedstawiono w tab. 5.4.

System MAX+PLUS dla rodziny Classic umożliwia wykonanie syntezy, w której optymalizowana jest powierzchnia lub szybkość pracy układu. Wykonane eksperymenty pokazały jednak, że dla badanego zbioru układów testowych uzyskuje się identyczne rozwiązania dla dwóch celów optymalizacyjnych. Dlatego w tab. 5.4 wyniki umieszczono w jednej kolumnie z nagłówkiem MAX+PLUS (Area, Speed). Strategia W_DK-W_LB doprowadziła w 20 przypadkach (67%) do najlepszych rozwiązań pod względem liczby bloków. Uzyskano 4 rozwiązania wykorzystujące najmniejszą liczbę układów EP1810. Jeżeli celem optymalizacyjnym są właściwości dynamiczne układu, to można wykorzystać strategię B_JPW. Sumaryczna liczba warstw logicznych dla tej strategii jest około 2 razy mniejsza od sumarycznej liczby warstw uzyskanej za pomocą systemu MAX+PLUS. Strategia B_JPW wykorzystuje jednak o 67% więcej bloków logicznych niż strategia W_DK-W_LB. Należy jednak podkreślić, że zaproponowane strategie syntezy działają wolniej od strategii zaimplementowanych w systemie MAX+PLUS. Tak dzieje się w większości przypadków, przede wszystkim wtedy, gdy system MAX+PLUS znajduje rozwiązanie mieszczące się w jednym układzie. Jeżeli zachodzi potrzeba podziału projektowanego układu na wiele części, realizowanych w poszczególnych strukturach programowalnych, to synteza w systemie MAX+PLUS jest bardzo nieefektywna i dość czasochłonna. Przykładowo dla układu testowego *ex1010* czas syntezy wynosi około 4min (Celleron 900MHz) i jest około 6-krotnie dłuższy od czasu syntezy strategią W_DK-W_LB. Dodatkowo uzyskane rozwiązanie wykorzystuje przeszło 10 razy więcej bloków logicznych typu PAL niż rozwiązanie uzyskane za pomocą strategii W_DK-W_LB.

5.6.3. Układy MAX firmy Altera (MAX+PLUS)

Zaprezentowane strategie syntezy zostały opracowane przede wszystkim z myślą o strukturach CPLD. Układy tego typu, oprócz elementarnych bloków logicznych typu PAL, często są wyposażone w różnorodne dodatkowe zasoby sprzętowe, znacznie zwiększające ich możliwości. Zasoby te są charakterystyczne dla poszczególnych producentów i rodzin układów programowalnych. Należą do nich między innymi: programowalne rozdzielacze, różnorodne ekspandery, elementy XOR itp. Systemy komercyjne umożliwiają wykonanie syntezy prowadzącej do wykorzystania różnorodnych elementów zawartych w strukturze programowalnej.

Tabela 5.4

Porównanie metod syntezy dla układów EP1810 (opis w tekście)

B - liczba użytych bloków logicznych typu PAL, występujących w strukturze PAL16V8
W - liczba warstw logicznych
U - liczba układów wykorzystanych do realizacji odpowiedniego układu testowego
i - liczba wejść układu testowego, o – liczba wyjść układu testowego
W_DK-W_LB - strategia realizacji Wielowyjściowej funkcji w strukturach typu PAL wykorzystująca Dekompozycję Kolumnową i metodę realizacji Wielowyjściowej funkcji bazującą na analizie grafu wyjść, w której układ optymalizowany jest pod kątem Liczby Bloków logicznych, użytą do realizacji bloku związanego i wolnego (k=8);
B_JPW - strategia syntezy wykorzystująca trójstanowe bufory wyjściowe (metoda B_JW; w przypadku braku rozwiązania metoda B_PW zapewniająca minimalną liczbę warstw) (k=8)

EP1810			MAX+PLU S II (Area,speed)			W_DK-W_LB (Area)			B_JPW (Speed)		
Układ testowy	i	o	B	W	U	B	W	U	B	W	U
5xp1	7	10	21	2	1	14	2	1	15	1	1
9sym	9	1	27	3	1	8	4	1	41	1	1
9symml	9	1	11	3	1	8	4	1	38	1	1
b12	15	9	9	1	1	10	2	1	10	1	1
bw	5	28	28	1	1	28	1	1	28	1	1
clip	9	5	30	2	1	17	3	1	30	1	1
con1	7	2	2	1	1	2	1	1	2	1	1
duke2	22	29	51	2	2	41	2	1	46	1	2
ex1010	10	10	770	20	17	74	2	2	198	1	5
f51m	8	8	18	2	1	14	3	1	15	1	1
inc	7	9	14	2	1	11	2	1	11	1	1
ldd	9	19	19	1	1	19	1	1	19	1	1
misex1	8	7	7	1	1	7	1	1	7	1	1
misex2	25	18	18	2	1	18	1	1	18	1	1
pcle	19	9	9	1	1	9	1	1	9	1	1
rd53	5	3	9	2	1	5	2	1	5	1	1
rd73	7	3	22	2	1	7	3	1	24	1	1
rd84	8	4	113	8	4	11	3	1	56	1	2
root	8	5	15	2	1	11	2	1	13	1	1
sao2	10	4	14	2	1	10	3	1	25	1	1
sqn	7	3	11	2	1	7	2	1	7	1	1
sqr6	6	12	18	2	1	15	2	1	15	1	1
squar5	5	8	8	1	1	8	1	1	8	1	1
table3	14	14	104	3	4	80	3	3	84	1	3
vg2	25	8	18	2	1	19	2	1	19	2	1
x2	10	7	7	1	1	7	1	1	7	1	1
xor5	5	1	3	2	1	2	2	1	2	1	1
z4m1	7	4	12	2	1	7	2	1	9	1	1
z5xp1	7	10	21	2	1	14	2	1	16	1	1
z9sym	9	1	11	3	1	8	4	1	42	1	1
			Σ 1420	80	53	491	64	33	819	31	38

W najprostszych układach CPLD firmy Altera blok logiczny typu PAL wyposażony został w dodatkowy element XOR. W układach tej rodziny występują również tzw. ekspandery, tzn. dodatkowe iloczyny zawracające sygnały z wyjść komórek PAL na matrycę programowalnych połączeń. Są one wykorzystywane w sytuacjach, gdy liczba iloczynów zawartych w bloku logicznym typu PAL jest niewystarczająca.

W tej sytuacji bezpośrednie porównanie zaproponowanych strategii syntezy z systemem MAX+PLUS staje się niemożliwe. Analizując strukturę komórki logicznej typu PAL, stanowiącej podstawowy element układów rodziny MAX5000, okazuje się, że funkcja realizowana w tym bloku może być przedstawiona w postaci $y = a \oplus (b+c+d)$, gdzie a,b,c,d są wyrażeniami iloczynowymi. Jeżeli $a(b+c+d)=0$, to zachodzi równość $a\oplus(b+c+d)=a+(b+c+d)$. Sprawia to, że istnieje możliwość uproszczonego porównania wyników syntezy w systemie MAX+PLUS z zaproponowanymi strategiami, w których syntezę wykonano dla bloków logicznych typu PAL, zawierających 4 iloczyny. W zaproponowanych strategiach syntezy trzeba dodatkowo w uzyskanych grupach 4 implikantów wyszukiwać jeden implikant rozłączny z pozostałymi. Implikant rozłączny może być realizowany wykorzystując iloczyn bezpośrednio dołączony do elementu XOR. Jeżeli taki implikant nie istnieje, to zachodzi potrzeba użycia dwóch komórek typu PAL, występujących w układach rodziny MAX5000. Sytuacja taka zdarza się niezwykle rzadko, szczególnie wtedy, gdy poszukiwania rozłącznych implikantów wykonywane są po minimalizacji z rozłączaniem implikantów, przedstawionej w rozdziale 3.3. Oczywisty jest fakt, że takie założenie, dostosowujące przedstawione strategie syntezy do struktur rodziny MAX5000, stawia w uprzywilejowanej pozycji system MAX+PLUS. Okazuje się jednak, że mimo tego proponowane strategie syntezy prowadzą do konkurencyjnych rozwiązań.

Wyniki eksperymentów dla strategii, których układ optymalizowany jest pod kątem powierzchni, zawarte są w tab. 5.5. W systemie MAX+PLUS wykonano syntezę logiczną trzema metodami. W pierwszej części tabeli (metoda A) zawarto wyniki syntezy, w której nie wykorzystywano ekspanderów i elementów XOR. W części z opisem (metoda B) zawarto wyniki syntezy, w której wykorzystano elementy XOR. W trzeciej części (metoda C) umieszczono wyniki syntezy, w której wykorzystano elementy XOR oraz ekspandery.

Wyniki przeprowadzonych eksperymentów są zaskakująco korzystne dla zaproponowanej strategii W_DK-W_LB. Porównując ją z metodą A okazuje się, że strategia W_DK-W_LB w większości przypadków prowadzi do lepszych rozwiązań niż system MAX+PLUS. Wykorzystuje dla badanej grupy układów testowych o 33% mniej bloków logicznych. Często prowadzi również do lepszych rozwiązań pod względem liczby warstw logicznych. Sumaryczna liczba warstw logicznych jest o 25% mniejsza od liczby warstw uzyskanych po syntezie w systemie MAX+PLUS (metoda A).

Wyniki licznych eksperymentów wykonanych w systemie MAX+PLUS pokazały, że potrafi on bardzo efektywnie wykorzystywać elementy XOR. Mimo tego okazało się, że dodatkowe możliwości wprowadzane przez element XOR nie wystarczyły do tego, aby w systemie MAX+PLUS uzyskać lepsze rozwiązania. Dopiero użycie ekspanderów spowodowało, że system MAX+PLUS prowadził do rozwiązań, wykorzystujących mniejszą liczbę komórek logicznych niż strategia W_DK-W_LB. Oczywiście takie porównanie obarczone jest dużym błędem, ponieważ system MAX+PLUS wykorzystywał dodatkowe elementy, których nie wykorzystywała strategia W_DK-W_LB. Bardziej obiektywne byłoby porównanie liczby wykorzystywanych iloczynów, traktując ekspander jako dodatkowy iloczyn. Oczywiście należy wziąć pod uwagę również niemożliwe do wykorzystania iloczyny, pozostające w użytych komórkach logicznych. Wartości wyznaczone zgodnie z powyższą zasadą zostały przedstawione w kolumnach z nagłówkiem AND. Dla syntezy wykonanej w systemie MAX+PLUS metodą C wartość w odpowiedniej komórce została wyznaczona z zależności 4*(liczba bloków)+liczba ekspanderów. Dla pozostałych metod oraz strategii W_DK-W_LB zawarto wartości równe 4*(liczba bloków). Okazuje się, że strategia W_DK-W_LB wykorzystuje o 15% mniej iloczynów niż system MAX+PLUS metodą C.

Tabela 5.5

Porównanie strategii W_DK-W_LB z systemem MAX+PLUS (optymalizacja powierzchni)
B - liczba wykorzystanych komórek typu PAL występujących w układach rodziny MAX5000
W - liczba warstw logicznych
U - najmniejszy układ rodziny MAX5000, w którym można zrealizować wybrany układ testowy
E - liczba wykorzystanych ekspanderów
AND - liczba wykorzystanych iloczynów

- dla metody A, B oraz strategii W_DK-W_LB: AND=4*(liczba bloków)
- dla metody C: AND=4*(liczba bloków)+liczba ekspanderów

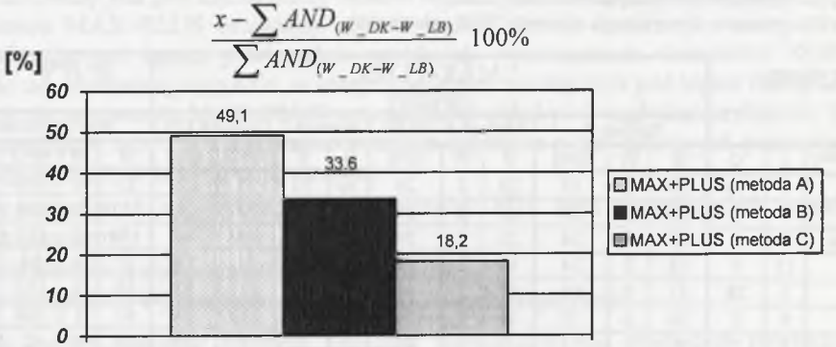
Metoda A - synteza niewykorzystująca elementów XOR i ekspanderów
Metoda B - synteza wykorzystująca elementy XOR
Metoda C - synteza wykorzystująca elementy XOR i ekspandery

MAX5000	MAX+PLUS (AREA)												W_DK-W_LB			
	Metoda A				Metoda B (XOR)				Metoda C (XOR+EXP)				k=4 (rozłączanie)			
Układ testowy	B	W	AND	U	B	W	AND	U	B	E	AND	U	B	W	AND	U
5xp1	33	3	132	5064	27	3	108	5032	19	12	88	5032	19	3	76	5032
9sym	42	5	168	5064	42	5	168	5064	31	32	156	5032	14	5	56	5032
9symml	25	3	100	5032	24	3	96	5032	13	32	84	5032	14	5	56	5032
b12	22	2	88	5032	15	2	60	5032	9	6	42	5032	16	2	64	5032
bw	51	3	204	5064	37	2	148	5064	29	10	126	5064	38	2	152	5064
clip	56	6	224	5064	37	4	148	5064	32	11	139	5032	31	4	124	5032
con1	4	2	16	5032	3	2	12	5032	2	1	9	5032	3	2	12	5032
duke2	101	9	404	5128	91	8	364	5128	76	23	327	5128	72	3	288	5128
ex1010	x	x	x	x	x	x	x	x	265	11	1071	2*5192	165	3	660	5128
f51m	32	3	128	5032	23	3	92	5032	19	5	81	5032	20	4	80	5032
inc	24	3	96	5032	20	3	80	5032	17	5	73	5032	17	2	68	5032
ldd	31	3	124	5064	28	4	112	5064	19	10	86	5064	25	2	100	5064
misex1	14	2	56	5032	13	2	52	5032	9	4	40	5032	12	2	48	5032
misex2	20	2	80	5032	19	2	76	5032	19	0	76	5032	19	2	76	5032
pcle	10	2	40	5032	9	1	36	5032	9	0	36	5032	9	1	36	5032
rd53	10	3	40	5032	9	2	36	5064	6	4	28	5032	7	3	28	5032
rd73	29	4	116	5032	30	3	120	5032	21	10	94	5032	12	4	48	5032
rd84	53	6	212	5064	51	5	204	5064	46	10	194	5064	17	5	68	5032
root	31	3	124	5032	27	3	108	5032	22	7	95	5032	22	5	88	5032
sao2	23	5	92	5032	23	4	92	5032	20	7	87	5032	23	4	92	5032
sqn	18	3	72	5032	16	3	64	5032	13	3	55	5032	15	3	60	5032
sqr6	30	4	120	5032	26	3	104	5064	22	7	95	5032	24	2	96	5032
squar5	15	4	60	5032	11	3	44	5032	9	4	40	5032	11	2	44	5032
table3	270	19	1080	2*5192	267	19	1068	5128+5192	227	70	978	5128+5192	181	4	724	5128
vg2	31	7	124	5064	23	6	92	5064	20	3	83	5064	37	3	148	5064
x2	12	2	48	5032	12	2	48	5032	8	4	36	5032	10	2	40	5032
xor5	2	2	8	5032	2	2	8	5032	2	0	8	5032	2	2	8	5032
z4ml	18	4	72	5032	17	3	68	5032	8	12	44	5032	11	4	44	5032
z5xp1	33	3	132	5064	27	3	108	5064	20	10	90	5032	19	3	76	5032
z9sym	25	3	100	5032	25	3	100	5032	14	30	86	5032	14	5	56	5032
	Σ 1065	120	4260		954	108	3816		761	332	3376		714	90	2856	

Należy jednak podkreślić (wyniki nie ujęte w tab. 5.5), że użycie ekspanderów doprowadziło do znacząco lepszych rozwiązań pod względem właściwości dynamicznych.

Podsumowaniem powyższych rozważań niech będzie wykres 5.2. Wartości przedstawione na mim wyznaczone zostały z zależności $\frac{x-\sum AND_{(W\_DK-W\_LB)}}{\sum AND_{(W\_DK-W\_LB)}}$ 100%, gdzie x oznacza odpowiednie sumy liczby iloczynów wykorzystanych przez system MAX+PLUS, realizujący syntezę metodą A,B lub C.

Wykres 5.2. Porównanie wyników syntezy uzyskanych w strategii W_DK-W_LB z wynikami uzyskanymi za pomocą systemu MAX+PLUS (opis w tekście)



Podobne eksperymenty wykonano dla strategii, w których układ optymalizowany jest pod względem szybkości. Ponieważ w układach rodziny MAX5000 występują sterowane za pomocą dodatkowych iloczynów trójstanowe bufory wyjściowe, do porównania wykorzystano strategię B_JPW.

Wyniki przeprowadzonych eksperymentów wyraźnie wskazują na niezwykłą efektywność strategii B_JPW. Oczywiście wynika to przede wszystkim z faktu wykorzystania dodatkowych iloczynów występujących w strukturach MAX5000, użytych do sterowania trójstanowych buforów wyjściowych. Porównując strategię B_JPW z metodą A okazuje się, że dla wszystkich przypadków prowadzi ona do lepszych rozwiązań pod kątem szybkości. Sumaryczna liczba warstw logicznych dla strategii B_JPW jest około 2,4 razy mniejsza niż dla metody A. Niestety redukcja liczby warstw logicznych okupiona jest znacznym wzrostem (23%) liczby wykorzystanych bloków.

Wykorzystanie w strukturach MAX5000 właściwości elementów XOR (metoda B) poprawiło uzyskane wyniki w stosunku do metody A. Nie doprowadziło jednak do lepszych rozwiązań (pod względem szybkości) od rozwiązań uzyskiwanych za pomocą strategii B_JPW. Dla rozpatrywanego zbioru układów testowych sumaryczna liczba warstw logicznych dla metody B jest 2,1 razy większa od sumarycznej liczby warstw uzyskanej w strategii B_JPW. Metoda B doprowadziła do redukcji liczby bloków logicznych, sumarycznie wykorzystując o 32% bloków mniej niż strategia B_JPW. Wykorzystanie ekspanderów doprowadziło do dalszej poprawy właściwości dynamicznych rozwiązań uzyskiwanych w systemie MAX+PLUS. Wliczając ekspandery jako dodatkowe iloczyny wykonano przybliżone porównanie metody C ze strategią B_JPW. Strategia B_JPW wykorzystuje około 1,7 razy więcej iloczynów, lecz średni czas propagacji dla analizowanego zbioru układów testowych jest około 1,9 razy krótszy.

Tabela 5.6

Porównanie strategii B_JPW z systemem MAX+PLUS (optymalizacja szybkości)

B - liczba wykorzystanych bloków (komórek) typu PAL rodziny MAX5000
W - liczba warstw logicznych
t - czas propagacji sygnału od wejść do wyjść
E - liczba wykorzystanych ekspanderów
AND - liczba iloczynów

- dla strategii zaimplementowanej w systemie MAX+PLUS: AND=4*(liczba bloków)+liczba ekspanderów
- dla strategii W_DK-W_LB: AND=4*(liczba bloków)

Metoda A - synteza niewykorzystująca elementów XOR i ekspanderów
Metoda B - synteza wykorzystująca elementy XOR
Metoda C - synteza wykorzystująca elementy XOR i ekspandery

MAX5000			MAX+PLUS (SPEED)										B_JPW			
			Metoda A			Metoda B (XOR)			Metoda C (XOR+EXP)				k=4 (rozłączanie)			
Układ testowy	i	O	B	W	t[ns]	B	W	t[ns]	B	E	AND	t[ns]	B	W	AND	t[ns]
5xp1	7	10	33	3	55	23	2	24	10	31	71	23	25	1	100	15
9sym	9	1	25	2	24	25	2	24	25	0	100	24	101	1	404	15
9symml	9	1	25	2	24	25	2	24	25	0	100	24	103	1	412	15
b12	15	9	18	2	24	13	2	24	9	5	41	23	21	1	84	15
bw	5	28	55	3	55	37	2	41	28	15	127	37	38	1	152	25
clip	9	5	56	4	70	41	4	70	24	47	143	24	67	1	268	15
con1	7	2	5	2	24	3	2	24	2	1	9	23	3	1	12	15
duke2	22	29	92	4	45	87	4	45	39	99	255	45	72	1	288	15
ex1010	10	10	286	7	146	276	6	116	143	286	858	100	463	1	1852	15
f51m	8	8	33	2	41	19	2	24	8	27	59	23	25	1	100	15
inc	7	9	27	2	24	23	2	24	9	30	66	23	18	1	72	15
ldd	9	19	33	3	57	26	3	56	19	9	85	52	29	1	116	15
misex1	8	7	18	2	24	13	2	24	7	11	39	23	12	1	48	15
misex2	25	18	20	2	24	19	2	24	19	0	76	24	19	1	76	15
pcle	19	9	10	2	24	9	1	15	9	0	36	15	9	1	36	15
rd53	5	3	15	2	24	11	2	24	3	17	29	23	10	1	40	15
rd73	7	3	51	3	55	37	2	41	22	44	132	24	49	2	196	41
rd84	8	4	147	6	175	86	3	55	54	93	309	71	103	2	412	24
root	8	5	30	2	24	27	2	24	14	32	88	24	26	1	104	15
sao2	10	4	22	2	24	20	2	24	10	32	72	24	29	2	116	24
sqn	7	3	18	2	24	15	2	24	4	30	46	24	17	1	68	15
sqr6	6	12	28	2	24	28	2	24	12	32	80	23	22	1	88	15
squar5	5	8	14	2	24	11	2	24	8	7	39	23	10	1	40	15
table3	14	14	159	4	70	159	4	70	78	331	643	70	179	2	716	24
vg2	25	8	48	3	56	41	4	70	23	46	138	56	67	1	268	15
x2	10	7	12	2	24	12	2	24	7	9	37	23	11	1	44	15
xor5	5	1	7	2	24	4	2	24	1	8	12	23	4	1	16	15
z4ml	7	4	25	2	24	15	2	24	4	32	48	23	20	1	80	15
z5xp1	7	10	33	3	55	22	2	24	10	27	67	23	27	1	108	15
z9sym	9	1	25	2	24	25	2	24	25	0	100	24	107	1	428	15
Σ			1370	81	1312	1152	73	1059	651	1301	3905	961	1686	34	6744	513
t średni[ns]					43,7			35,3				32,0				17,1

Podobne eksperymenty przeprowadzono dla bardziej złożonej rodziny układów CPLD, tzn. układów serii MAX7000 firmy Altera [Alte99]. Podstawowa konfiguracja komórki PAL

zawiera pięć iloczynów, z których jeden może być dołączony do elementu XOR. Elastyczność struktur tego typu zdecydowanie zwiększają dwa rodzaje ekspanderów. Pierwsze z nich, tzn. ekspandery wspólne (ang. shareable expanders), dostępne są w obrębie jednej matrycy bloków logicznych (ang. LAB – Lagic Array Block) umożliwiając wykonywanie sprzężeń zwrotnych z wyjść komórek typu PAL na matryce połączeń. Drugie, tzn. ekspandery równoległe (ang. parallel expanders), pozwalają tworzyć większe bloki logiczne typu PAL (do 20 iloczynów dołączonych do sumy), przeznaczone do realizacji złożonych funkcji. Ekspansja iloczynów w bloku logicznym typu PAL wprowadza bardzo nieznaczne opóźnienia [Alte99]. Ze względu na obecność tych elementów, wykonując eksperymenty koncentrowano się przede wszystkim na porównaniu strategii, w których układ optymalizowany jest pod kątem liczby bloków logicznych. Wykonano szereg eksperymentów w systemie MAX+PLUS ustawiając różne warunki syntezy (realizacja wielopoziomowa, optymalizacja pod kątem powierzchni, szybkości, wykorzystanie elementów XOR itp.). Okazało się, że istnieją przypadki, w których najlepsze rozwiązania pod kątem liczby bloków, uzyskuje się w systemie MAX+PLUS optymalizując układ pod względem szybkości. Wyniki przedstawione w tab. 5.7 umieszczono w trzech częściach odpowiadających poszczególnym metodom syntezy, w których odpowiednio:

- w metodzie A nie wykorzystano elementów XOR oraz występujących w układach ekspanderów,
- w metodzie B wykorzystano elementy XOR,
- w metodzie C wykorzystano elementy XOR oraz ekspandery.

W każdej metodzie wybrano najlepsze rozwiązanie pod względem powierzchni z szeregu wyników syntezy, wśród których rozpatrywano również rozwiązania optymalizowane pod względem szybkości. Najlepsze wyniki uzyskiwano opisując układ testowy w postaci wyrażeń logicznych, będących sumą iloczynów (dodatek).

Podobnie jak dla układów rodziny MAX 5000 wyniki przeprowadzonych eksperymentów są zaskakująco korzystne dla zaproponowanej strategii W_DK-W_LB. Porównując ją z metodą A okazuje się, że strategia W_DK-W_LB w 17 przypadkach (56%) prowadzi do lepszych rozwiązań niż system MAX+PLUS. Tylko w 3 przypadkach (10%) prowadzi do rozwiązań gorszych pod względem liczby bloków logicznych. Wykorzystuje, dla badanej grupy układów testowych, o 13% mniej bloków logicznych. W pięciu przypadkach (*duke2, ex1010, sao2, sqn, sqr6*) prowadzi do nieznacznie lepszych rozwiązań pod względem liczby warstw logicznych. W sześciu przypadkach (*9sym, 9symml, clip, f51m, z4ml, z9sym*) system MAX+PLUS znajduje lepsze rozwiązania pod względem szybkości. Rozwiązania te są jednak znacznie lepsze niż w strategii W_DK-W_LB (np. *9sym*), co sprawia, że sumaryczna liczba warstw logicznych jest w metodzie A o 6% mniejsza. Wykorzystanie w systemie MAX+PLUS elementów XOR (metoda B) prowadzi, w stosunku do metody A, do lepszych rozwiązań zarówno pod względem powierzchni, jak i szybkości. Ciągle jednak strategia W_DK-W_LB wykorzystuje, dla badanej grupy układów testowych, o 11% bloków mniej od metody B. Użycie występujących w układach MAX7000 ekspanderów spowodowało, że w systemie MAX+PLUS uzyskiwano rozwiązania, wykorzystujące mniejszą liczbę komórek logicznych typu PAL. Bardziej obiektywne jest jednak porównanie liczby wykorzystywanych iloczynów, traktując ekspandery jako dodatkowe iloczyny. Oczywiście należy wziąć pod uwagę również niemożliwe do wykorzystania iloczyny, pozostające w użytych komórkach logicznych. Wartości wyznaczone zgodnie z powyższą zasadą zostały przedstawione w kolumnach z nagłówkiem AND. Dla systemu MAX+PLUS wartość w odpowiedniej komórce została wyznaczona z zależności 5*(liczba bloków)+liczba ekspanderów. Dla strategii W_DK-W_LB zawarto wartości równe 5*(liczba bloków). Przy powyższych

uproszczeniach, ciągle stawiających w lepszej sytuacji system MAX+PLUS, strategia W_DK-W_LB wykorzystuje tylko nieznacznie więcej (2%) iloczynów niż system MAX+PLUS, wykonujący syntezę metodą C.

Tabela 5.7

Porównanie strategii W_DK-W_LB z systemem MAX+PLUS dla układów CPLD rodziny MAX7000 (najlepsze rozwiązania pod względem powierzchni)

B - liczba wykorzystanych bloków (komórek) typu PAL rodziny MAX5000
W - liczba warstw logicznych
U - najmniejszy układ z rodziny MAX7000, w którym można zrealizować układ testowy
E - liczba wykorzystanych ekspanderów
AND - liczba iloczynów

- dla strategii zaimplementowanej w systemie MAX+PLUS: AND=5*(liczba bloków)+liczba ekspanderów
- dla strategii W_DK-W_LB: AND=5*(liczba bloków)

Metoda A - synteza niewykorzystująca elementów XOR i ekspanderów
Metoda B - synteza wykorzystująca elementy XOR
Metoda C - synteza wykorzystująca elementy XOR i ekspandery

MAX7000			MAX+PLUS (AREA)											W_DK-W_LB			
			Metoda A			Metoda B (XOR)			Metoda C (XOR+EXP)					*k*=5			
Układ testowy	i	o	B	W	U	B	W	U	B	E	AND	U		B	W	AND	U
5xp1	7	10	23	3	7032	20	2	7032	14	8	78	7032		18	3	90	7032
9sym	9	1	16	2	7032	17	3	7032	15	2	77	7032		11	5	55	7032
9symml	9	1	16	2	7032	16	3	7032	15	2	77	7032		11	5	55	7032
b12	15	9	11	2	7032	10	2	7032	9	2	47	7032		14	2	70	7032
bw	5	28	30	2	7032	28	1	7032	28	0	140	7032		30	2	150	7032
clip	9	5	37	3	7064	25	3	7032	22	3	113	7032		28	4	140	7032
con1	7	2	2	1	7032	2	1	7032	2	0	10	7032		2	1	10	7032
duke2	22	29	64	4	7064	65	5	7096	46	45	275	7064		61	2	305	7064
ex1010	10	10	175	5	2*7096	173	5	2*7096	161	12	817	2*7096		124	3	620	2*7096
f51m	8	8	22	3	7032	20	3	7032	10	13	63	7032		16	4	80	7032
inc	7	9	18	2	7032	20	3	7032	11	9	64	7032		16	2	80	7032
ldd	9	19	26	2	7032	24	2	7032	19	7	102	7032		25	2	125	7032
misex1	8	7	8	2	7032	8	2	7032	7	3	38	7032		8	2	40	7032
misex2	25	18	18	1	7032	18	1	7032	18	0	90	7032		18	1	90	7032
pcle	19	9	9	1	7032	9	1	7032	9	0	45	7032		9	1	45	7032
rd53	5	3	6	2	7032	7	2	7032	3	8	23	7032		5	2	25	7032
rd73	7	3	21	3	7032	24	2	7032	19	6	101	7032		10	3	50	7032
rd84	8	4	37	4	7064	42	5	7064	31	11	166	7032		13	4	65	7032
root	8	5	19	3	7032	19	2	7032	16	5	85	7032		18	3	90	7032
sao2	10	4	15	5	7032	13	2	7032	11	7	62	7032		19	4	95	7032
sqn	7	3	11	3	7032	10	2	7032	7	9	44	7032		11	2	55	7032
sqr6	6	12	20	3	7032	21	2	7032	15	8	83	7032		20	2	100	7032
squar5	5	8	10	2	7032	11	2	7032	8	3	43	7032		9	2	45	7032
table3	14	14	120	3	2*7096	120	3	2*7096	85	99	524	2*7096		135	3	675	7032
vg2	25	8	18	3	7032	16	3	7032	16	0	80	7032		28	3	140	7032
x2	10	7	9	2	7032	9	2	7032	7	3	38	7032		9	2	45	7032
xor5	5	1	2	2	7032	2	2	7032	1	2	7	7032		2	2	10	7032
z4ml	7	4	13	3	7032	12	2	7032	6	9	39	7032		10	4	50	7032
z5xp1	7	10	23	3	7032	20	2	7032	12	12	72	7032		18	3	90	7032
z9sym	9	1	16	2	7032	16	3	7032	15	1	76	7032		11	5	55	7032
			815	78		797	73		638	289	3479			709	83	3545	

Użycie ekspanderów prowadzi do znacząco lepszych rozwiązań pod względem właściwości dynamicznych, wynikających przede wszystkim z krótkiego czasu propagacji wnoszonego przez każdy ekspander (wyniki nie ujęte w tab. 5.7). Na przykład, dla analizowanego zbioru układów testowych wykorzystanie ekspanderów w metodzie C, w której układ optymalizowany jest pod kątem szybkości, doprowadziło do skrócenia średniego czasu propagacji sygnałów od wejść do wyjść o 34%.

Żadna z zaprezentowanych w pracy strategii syntezy nie wykorzystuje specyficznych właściwości poszczególnych rodzin układów CPLD. Stawia to system MAX+PLUS, wykorzystujący ekspandery, na bardzo uprzywilejowanej pozycji. W tej sytuacji trudno porównać zaproponowane strategie syntezy z narzędziami firmowymi, które wykorzystują dodatkowe elementy, wprowadzone do struktury programowalnej, w celu poprawy właściwości dynamicznych. Ponieważ w układach rodziny MAX7000 nie ma możliwości sterowania trójstanowymi buforami wyjściowymi dla każdego wyjścia oddzielnie, dlatego do porównania użyto strategii J_DW-W_LW. Wyniki eksperymentów dla kilkunastu wybranych układów testowych zawarto w tab. 5.8. Analiza uzyskanych wyników wskazuje jednoznacznie, że dla przedstawionych powyżej uproszczeń system MAX+PLUS prowadzi do znacznie lepszych rozwiązań pod względem szybkości.

Tabela 5.8

Porównanie strategii J_DW-W_LW z systemem MAX+PLUS dla układów CPLD rodziny MAX7000 (optymalizacja szybkości)

B - liczba wykorzystanych bloków (komórek) typu PAL rodziny MAX7000
W - liczba warstw logicznych
t - czas propagacji sygnału od wejść do wyjść
E - liczba wykorzystanych ekspanderów
AND - liczba iloczynów

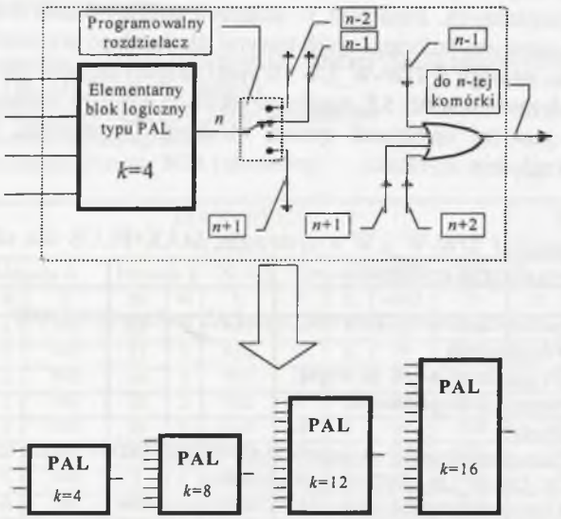
- dla strategii zaimplementowanej w systemie MAX+PLUS: AND=5*(liczba bloków)+liczba ekspanderów
- dla strategii W_DK-W_LB: AND=5*(liczba bloków)

Metoda A - synteza niewykorzystująca elementów XOR i ekspanderów
Metoda B - synteza wykorzystująca elementy XOR
Metoda C - synteza wykorzystująca elementy XOR i ekspandery

MAX7000			MAX+PLUS (SPEED)										J_DW-W_LW			
			Metoda A			Metoda B (XOR)			Metoda C (XOR+EXP)				*k*=5			
Układ testowy	i	o	B	W	t[ns]	B	W	t[ns]	B	E	AND	t[ns]	B	W	AND	t[ns]
5xp1	7	10	24	2	9,6	20	2	9,6	14	8	78	9,6	18	2	90	9,6
9sym	9	1	33	3	17,5	33	3	17,5	29	16	161	13,2	15	4	75	16,8
9symml	9	1	16	3	13,2	16	3	13,2	15	1	76	13,2	15	4	75	16,8
clip	9	5	37	2	12,5	30	2	9,6	26	5	135	9,6	34	3	170	17,5
f51m	8	8	22	2	9,6	17	2	9,6	10	13	63	9,6	16	3	80	13,2
rd53	5	3	9	2	9,6	7	2	9,6	3	14	29	9,5	6	2	30	9,6
rd73	7	3	32	2	9,6	24	2	9,6	23	2	117	9,6	15	3	75	13,2
rd84	8	4	126	7	36	60	3	17,5	56	16	296	17,5	23	3	115	13,2
sao2	10	4	15	2	9,6	13	2	9,6	11	7	62	9,6	18	3	90	13,2
xor5	5	1	5	2	9,6	3	2	9,6	1	8	13	9,5	2	2	10	9,6
z4m1	7	4	16	2	9,6	12	2	9,6	6	9	39	9,6	9	3	45	13,2
z5xp1	7	10	24	2	9,6	20	2	9,6	12	12	72	9,6	18	2	90	9,6
z9sym	9	1	16	3	13,2	16	3	13,2	15	1	76	13,2	15	4	75	16,8
Σ			375	34	169,2	271	30	148,8	221	112	1217	143,3	204	38	1020	172,3
t średni[ns]					13			11,5				11				13,3

5.6.4. Układy MACH (Synplify+MACHxl)

Alternatywnym rozwiązaniem w stosunku do różnego typu ekspanderów są programowalne rozdzielacze (ang. Logic Allocator), występujące w układach MACH, opracowanych przez firmę AMD. Układy MACH składają się z kilku (2, 4, 8, 16) dużych bloków logicznych, tzw. MACHxxx PAL bloków (xxx - typ układu), których częściami składowymi są elementarne bloki logiczne typu PAL, zawierające 4 iloczyny. Występujące w strukturach MACH programowalne rozdzielacze zwiększają efektywność wykorzystania iloczynów, umożliwiając, w zależności od potrzeb, odpowiedni rozdział iloczynów pomiędzy poszczególne komórki wyjściowe (rys. 5.1).



Rys. 5.5. Konfiguracja bloków typu PAL w układach MACH

Fig. 5.5. PAL-based logic blocks configurations applied to MACH devices

Mechanizm ten umożliwia realizację bardziej złożonych funkcji w jednym bloku typu PAL, zawierającym większą liczbę iloczynów, bez konieczności wykorzystywania sprzężeń zwrotnych. MACHxxx PAL blok zawiera określoną liczbę iloczynów, komórek wyjściowych oraz zasoby do konfigurowania bloków logicznych typu PAL. Przykładowo najprostszy układ MACH110 zawiera dwa MACH110 PAL bloki, w których znajduje się 16 komórek wyjściowych oraz 64 iloczyny wstępnie podzielone na 16 grup po 4 iloczyny dołączone do sumy. Programowalne rozdzielacze umożliwiają między innymi utworzenie 16 elementarnych bloków logicznych typu PAL, zawierających 4 iloczyny lub 6 bloków, zawierających odpowiednio 12, 12, 12, 12, 8, 8 iloczynów [Adva95].

Tabela 5.9 zawiera wyniki syntezy wybranych układów testowych wykonanej dla rodziny układów MACH2. Najlepsze rezultaty dla wybranej rodziny układów CPLD uzyskano wykonując syntezę logiczną w systemie Synplify [Synp00a, Synp00b], który dodatkowo wykonał podział układu testowego na poszczególne bloki (map logic to makrocell), występujące w układach MACH. Plik wynikowy w formacie *.*src* wczytywany

był następnie do systemu MACHxl, który wykonał ostatni etap syntezy i pozwolił na prosty odczyt parametrów charakteryzujących jakość syntezy, takich jak liczba wykorzystanych iloczynów (AND_W) oraz liczba pozostałych iloczynów (AND_P), które mogą być jeszcze wykorzystane.

Alternatywą do tak skomponowanej metody syntezy jest strategia W_DK-W_LB, w której wykonano syntezę dla bloków logicznych typu PAL, zawierających 4 iloczyny. Oczywiście takie założenie stawia strategię W_DK-W_LB w znacznie gorszej sytuacji. Po to, aby porównać powyższe dwie metody syntezy (Synplify+MACHxl; W_DK-W_LB), należy określić wspólny parametr charakteryzujący uzyskiwane rozwiązania. Może nim być procent wykorzystania układu lub liczba zajętych iloczynów (AND_Z), tzn. takich iloczynów, które są wykorzystane do realizacji odpowiedniego układu testowego w sumie z tymi, których nie można wykorzystać (wolne iloczyny, znajdujące się w wykorzystanych blokach typu PAL).

W przypadku strategii W_DK-W_LB liczba zajętych iloczynów (AND_Z) jest równa iloczynowi liczby bloków i liczby iloczynów zawartych w pojedynczym bloku.

Tabela 5.9

Porównanie strategii W_DK-W_LB z syntezą realizowaną za pomocą systemów Synplify oraz MACHxl dla układów rodziny MACH2 (optymalizacja powierzchni)

U - typ układu wykorzystanego do realizacji odpowiedniego układu testowego
AND_W - liczba iloczynów wykorzystanych do realizacji odpowiedniego układu testowego
AND_P - liczba iloczynów pozostałych w strukturze
AND_Z - liczba iloczynów zajętych, tzn. takich, których nie można wykorzystać
% - procent wykorzystania układu
B - liczba bloków logicznych typu PAL (k=4) potrzebnych do realizacji odpowiedniego układu testowego

MACH 2		Synplify+MACHxl				W_DK-W_LB		
Układ testowy	U	AND_W	AND_P	AND_Z	%	B(*k*=4)	AND_Z	%
5xp1	MACH210	66	172	84	32	19	76	29
9sym	MACH210	65	180	76	29	14	56	21
9symml	MACH210	64	184	72	28	14	56	21
Clip	MACH210	134	72	184	71	31	124	48
con1	MACH210	9	244	12	4	3	12	4
duke2	MACH230	204	136	376	73	72	288	56
f51m	MACH210	58	180	76	29	20	80	31
Inc	MACH210	51	192	64	25	17	68	26
Ldd	MACH210	56	152	104	40	25	100	39
rd53	MACH210	32	216	40	15	7	28	10
rd73	MACH210	71	176	80	31	12	48	18
rd84	MACH210	89	156	100	39	17	68	26
Root	MACH210	77	164	92	35	22	88	34
sao2	MACH210	57	184	72	28	23	92	35
Sqn	MACH210	44	204	52	20	15	60	23
sqr6	MACH210	60	168	88	34	24	96	37
squar5	MACH210	29	212	44	17	11	44	17
vg2	MACH210	112	88	168	65	37	148	57
x2	MACH210	23	204	52	20	10	40	15
xor5	MACH210	16	240	16	6	2	8	3
z4ml	MACH210	42	208	48	18	11	44	17
z5xp1	MACH210	66	172	84	32	19	76	29
z9sym	MACH210	64	184	72	28	14	56	21
Σ				2056			1756	

Mimo iż strategia W_DK-W_LB w ograniczonym stopniu wykorzystuje możliwości układów MACH, w 16 przypadkach (70%) doprowadziła do lepszych rozwiązań pod względem powierzchni. Doprowadziło to, dla analizowanego zbioru układów testowych, do 15-procentowego ograniczenia sumarycznej liczby zajętych iloczynów.

Ze względu na przyjęte uproszczenia porównanie uzyskiwanych rozwiązań pod względem szybkości jest bezcelowe. Istnieje oczywiście możliwość opracowania strategii W_DK-W_LB dla poszczególnych układów rodziny MACH. Doprowadziłoby to do dalszego poprawienia uzyskiwanych wyników ze względu na lepsze dopasowania bloków związanych i wolnych do struktury bloków typu PAL, dostępnych w poszczególnych układach rodziny MACH. W takiej sytuacji istniałaby również możliwość porównania wyników pod względem szybkości. Elementy podobnego dopasowania projektowanego układu do struktury MACH, opracowane dla syntezy wielowyjściowych funkcji, przedstawione są w pracy [Kani01c]

5.6.5. Układy FLASH firmy Cypress (WARP)

Znacznie bardziej elastyczne są struktury CPLD firmy Cypress rodziny FLASH370. Układy te składają się z kilku (2, 4, 8) bloków logicznych, zawierających między innymi 80 iloczynów, programowalne rozdzielacze oraz 16 makrokomórek. Nazwijmy te bloki blokami logicznymi typu FLASH. Programowalny rozdzielacz zawarty w tych strukturach pozwala na dołączenie do jednej sumy od 0 do 16 iloczynów. Tak więc blok logiczny typu PAL w przypadku tych układów może zawierać dowolną (2..16) liczbę iloczynów. Istnieją jednak ograniczenia wynikające z sumarycznej liczby iloczynów (80) przypadających na jeden blok logiczny typu FLASH, zawierający 16 sum. Tak więc przykładowo w jednym bloku logicznym typu FLASH można utworzyć tylko 5 (80/16) bloków logicznych typu PAL, zawierających 16 iloczynów. Ze względu na dodatkowe ograniczenia występujące dla dwóch krańcowych sum w bloku logicznym typu FLASH można miedzy innymi zrealizować 16 bloków logicznych typu PAL zawierających 4 iloczyny, 15 bloków typu PAL zawierających 5 iloczynów, 11 bloków logicznych typu PAL zawierających 7 iloczynów, 10 bloków logicznych typu PAL zawierających 8 iloczynów itd. Dokładne struktury oraz właściwości poszczególnych układów rodziny FLASH370 przedstawiono w pracy [Skah96].

Firma Cypress wraz z układami programowalnymi oferuje narzędzie do syntezy pod firmową nazwą WARP. Przeprowadzono szereg eksperymentów wykorzystując ten system. Okazało się, że dla analizowanego zestawu układów testowych nie stwierdzono różnicy w rozwiązaniach, dla których w procesie syntezy optymalizowano powierzchnię lub szybkość docelowego rozwiązania. Testowano również różne sposoby opisu układów testowych w języku VHDL. Najbardziej efektywna okazała się metoda opisu za pomocą tablicy prawdy przedstawionej w języku VHDL (dodatek).

Do porównania użyto dwie strategie syntezy: W_DK-W_LB oraz J_DW-W_LW. Wykonywano syntezę dla różnych wielkości bloków logicznych typu PAL. Wyniki eksperymentów przedstawiono w tab. 5.10. Użycie strategii J_DW-W_LW wynikało z ograniczonych możliwości sterowania trójstanowymi buforami wyjściowymi. W tabeli 5.10 zawarto liczby iloczynów wykorzystanych do realizacji odpowiedniego układu testowego (kolumny z oznaczeniem AND) oraz liczby warstw logicznych (kolumny z oznaczeniem W). W przypadku strategii W_DK-W_LB oraz J_DW-W_LW liczba umieszczona w kolumnie AND jest iloczynem liczby bloków i liczby iloczynów w bloku logicznym typu PAL.

Tabela 5.10
Porównanie zaproponowanych strategii z systemem WARP dla układów rodziny FLASH370 (opis w tekście)

AND - liczba iloczynów wykorzystanych do realizacji odpowiedniego układu testowego
W - liczba warstw logicznych uzyskiwanego rozwiązania
B - liczba bloków logicznych wykorzystanych do realizacji odpowiedniego układu testowego
k - liczba iloczynów zawartych w bloku logicznych typu PAL

Flash370	WARP		W_DK-W_LB																	J_DW-W_LW					
			k=4			k=5			k=7			k=8			Dobór k				k=4			k=5		
Przykład	AND	W	B	AND	W	B	AND	W	B	AND	W	B	AND	W	B	k	AND	W	B	AND	W	B	AND	W
5xp1	71	2	19	76	3	18	90	3	14	98	2	14	112	2	19	4	76	3	21	84	3	18	90	2
9sym	73	2	14	56	5	11	55	5	9	63	5	8	64	4	11	5	55	5	21	84	4	15	75	4
9symml	73	2	14	56	5	11	55	5	9	63	5	8	64	4	11	5	55	5	21	84	4	15	75	4
clip	147	2	31	124	4	28	140	4	19	133	3	17	136	3	32	4	128	4	44	176	3	34	170	3
duke2	124	2	72	288	3	61	305	2	44	308	2	41	328	2	72	4	288	3	72	288	3	61	305	2
f51m	78	2	20	80	4	16	80	4	13	91	3	14	112	3	20	4	80	4	19	76	3	16	80	3
rd53	31	1	7	28	3	5	25	2	5	35	2	5	40	2	5	5	25	2	7	28	2	6	30	2
rd73	138	2	12	48	4	10	50	3	8	56	3	7	56	3	12	4	48	4	17	68	3	15	75	3
rd84	264	2	17	68	5	13	65	4	11	77	3	11	88	3	13	5	65	4	27	108	4	23	115	3
sao2	54	2	23	92	4	19	95	4	10	70	3	10	80	3	10	7	70	3	21	84	3	18	90	3
vg2	112	2	37	148	3	28	140	3	20	140	2	19	152	2	20	7	140	2	37	148	3	28	140	3
xor5	16	1	2	8	2	2	10	2	2	14	2	2	16	2	2	4	8	2	2	8	2	2	10	2
z4ml	59	2	11	44	4	10	50	4	7	49	2	7	56	2	11	4	44	4	9	36	3	9	45	3
z5xp1	71	2	19	76	3	18	90	3	15	105	2	14	112	2	19	4	76	3	21	84	3	18	90	2
z9sym	73	2	14	56	5	11	55	5	9	63	5	8	64	4	11	5	55	5	21	84	4	15	75	4
Σ	1384	28		1248	57		1305	53		1365	44		1480	41			1213	53		1440	47		1465	43

Okazuje się, że mimo uproszczeń w strategiach W_DK-W_LB oraz J_DW-W_LW uzyskano wiele lepszych rozwiązań, przede wszystkim pod względem powierzchni, od rozwiązań uzyskanych za pomocą systemu WARP.

Najlepsza pod względem powierzchni jest strategia W_DK-W_LB realizowana dla bloków logicznych typu PAL zawierających 4 iloczyny. Doprowadziła ona do 9 lepszych rozwiązań od systemu WARP pod względem powierzchni. Niestety jest ona najgorsza pod względem szybkości. Tworzenie większych bloków logicznych typu PAL prowadziło do coraz gorszych rozwiązań pod względem powierzchni i lepszych pod względem szybkości. Istniały jednak przypadki, w których znajdowano najlepsze rozwiązania pod względem powierzchni dla różnych wielkości bloków logicznych typu PAL (np. dla *k*=5 - *9sym, rd53 rd84*, dla *k*=7 - *sao2, vg2*). W części tabeli z nagłówkiem „dobór k" zawarto najlepsze rozwiązania pod względem powierzchni spośród rozwiązań uzyskanych dla analizowanych wielkości bloków logicznych typu PAL.

Zmniejszenie liczby warstw logicznych uzyskuje się stosując strategię J_DW-W_LW. Niestety rozwiązania te są gorsze pod względem szybkości od rozwiązań uzyskanych w systemie WARP i dodatkowo wykorzystują większą liczbę iloczynów. Fakt ten wynika z większej elastyczności sytemu WARP, związanej z nierównomiernym rozdziałem iloczynów pomiędzy poszczególne sumy.

Wyniki eksperymentów pośrednio wskazują, że nierównomierny rozdział iloczynów pomiędzy poszczególne sumy istotnie wpływa na poprawę właściwości dynamicznych uzyskiwanych rozwiązań.

5.6.6. Systemy akademickie przeznaczone dla matryc PLA

Różnorodne metody optymalizacji układów, realizowanych w postaci matryc PLA, przedstawione są w pracach: [ChenM88, DevaN88a, DevaN88b, CiesY92].

Pierwsza z nich [ChenM88] polega na tworzeniu dekoderów na wejściu matrycy AND. Istota metody tkwi w odpowiednim wyborze zmiennych wejściowych poszczególnych dwu- lub *k*-wejściowych dekoderów, realizowanym przez algorytm ASSIGN.

Metody przedstawione w pracach [DevaW88a, DevaW88b] związane są z wielopoziomową optymalizacją układu, realizowanego w postaci matryc PLA. Oparte są one na elementach faktoryzacji i dekompozycji funkcji logicznych. Wykorzystują elementy minimalizacji funkcji wielowartościowych w procesie poszukiwania tzw. silnych podzielników (*ang.* strong divisors), służących do minimalizacji liczby literałów i powierzchni wielopoziomowego układu.

Istota metod przedstawionych w pracy [CiesY92] tkwi w poszukiwaniu odpowiedniej dwupoziomowej dekompozycji matrycy PLA. Pierwotna matryca PLA dzielona jest na kilka kaskadowo połączonych mniejszych matryc, wśród których występują: przypadkowa liczba matryc na pierwszym poziomie i jedna matryca na poziomie drugim. Poszukiwany jest podział, który zapewnia, że sumaryczna powierzchnia matryc po podziale jest mniejsza od powierzchni matrycy pierwotnej. W pracy przedstawione są różne metody poszukiwania takiego podziału, bazujące na programowaniu całkowitoliczbowym (tab. 5.11 kolumny z opisem IP - Integer Programming, IP/GA - Integer Programming/ Greedy Approach) oraz elementach teorii podziałów (tab. 5.11 kolumna z opisem GP - - Graph Partitioning).

We wszystkich powyższych metodach syntezy powierzchnię układu wyznaczono z zależności $S = (2i + o)p$, gdzie i oznacza liczbę wejść matrycy PLA, o - liczbę wyjść, natomiast p - liczbę wykorzystanych iloczynów.

Zaproponowane strategie syntezy zostały opracowane dla układów CPLD typu PAL, stąd głównym celem optymalizacyjnym była liczba bloków logicznych typu PAL. W celu porównania najlepszych pod względem powierzchni strategii W_DK-W_LB dla funkcji wielowyjściowych i J_DW-W_LB dla funkcji jednowyjściowej z przytoczonymi powyżej metodami konieczne stało się wyznaczenie powierzchni zajmowanej przez wykorzystywane bloki logiczne typu PAL. Powierzchnia bloku została wyznaczona z zależności $S_k = (2i)k$, gdzie i oznacza liczbę wejść, natomiast k - liczbę iloczynów zawartych w bloku logicznym typu PAL. Ponieważ w przypadku systemów akademickich powierzchnię określano biorąc pod uwagę tylko wykorzystane iloczyny, stąd określono również powierzchnię tylko wykorzystanych części bloków logicznych typu PAL, tzn. $S_p = (2i)p$, gdzie p oznacza liczby wykorzystanych iloczynów w poszczególnych blokach.

Wyniki porównania zostały zawarte w tab. 5.11. W czterech najniższych wierszach zawarte są odpowiednie sumaryczne wartości powierzchni zajmowanej przez odpowiedni zestaw układów testowych. Pogrubioną czcionką wyróżniono najlepsze pod względem powierzchni rozwiązania, wśród których znalazły się rozwiązania uzyskiwane zaproponowaną strategią syntezy W_DK-W_LB (*f51m*, *rd53*, *rd84*, *sao2*).

Tabela 5.11
Porównanie zaproponowanej strategii W_DK-W_LB z innymi akademicki algorytmami przeznaczonymi do minimalizacji powierzchni matryc PLA (*opis w tekście*)

S_{pocz} – powierzchnia pierwotna (przed podziałem)
$S = (2i + o) * p$ – zależność opisująca powierzchnię
i – liczba wejść,
o – liczba wyjść,
p – liczba implikantów układu testowego

					ASSIGN			PLADE [CiesY92]			W_DK-W_LB (J_DW-W_LB)	
Układ	i	o	p	S_{pocz}	[ChenM88]	[DevaW88a]	[DevaN88b]	IP	IP/GA	GP	S_k=2ik	S_p=2ip
5xp1	7	10	65	1560		1139	1019	**492**	864	872	1032	770
9sym	9	1	87	1653		940	735	**545**	**545**	526	800 (688)	728 (664)
duke2	22	29	86	6278		6504		**5653**	**5653**	5984	8240	6182
f51m	8	8	76	1824	848	1482		848	848	896	1040	**682**
rd53	5	3	31	403		219	219				336	**218**
rd73	7	3	127	2159	**358**	683	613				560	430
rd84	8	4	255	5100		1040	1040				944	**602**
root	8	5	57	1197	**824**	966	966	829	829	891	1408	1248
sao2	10	4	58	1392		1312	1320	968	898	1008	1216	**850**
sqn	7	3	38	646	**439**						784	658
vg2	25	8	110	6380		5082	3652	**1502**	2796	4978	5088	3700
z4ml	7	4	59	1062		**440**	399				560	472
z5xp1	7	10	63	1512	950	1190	1076	968	968	**864**	1296	946
z4	7	4	59	1062	240			**200**	262	211	560	472
		Σ		8400	**3659**						5648	4436
		Σ		30520		20997					22520 (22408)	**16828 (16764)**
		Σ		22418			11039				13240 (13128)	**9964 (9900)**
		Σ		22858				**12005**	13663	16230	20680 (20568)	15578 (15514)

5.6.7. Analiza końcowa wyników eksperymentów

Autor pracy daleki jest od twierdzenia, że zaproponowane strategie syntezy nadają się bezpośrednio do wykorzystania w systemach komercyjnych. Jednoznacznie wskazują jednak, gdzie należy poszukiwać znaczącej poprawy efektywności procesu syntezy. Patrząc na optymalizację powierzchni, znacznej poprawy można oczekiwać wplatając w proces syntezy elementy dekompozycji, dopasowane do zasobów logicznych struktur matrycowych. Niestety proces syntezy wykorzystujący strategie bazujące na dekompozycji (np. W_DK-W_LB) jest bardzo złożony i czasochłonny. Należałoby poszukiwać szybszych algorytmów dekompozycji wykorzystujących np. diagramy BBD.

Jeżeli głównym celem optymalizacji jest szybkość układu, to należy zastosować metody syntezy wykorzystujące, powszechnie występujące w strukturach matrycowych, trójstanowe bufory wyjściowe. Tym bardziej, że opracowane metody syntezy (np. B_JPW) są stosunkowo proste.

Podsumowując wyniki eksperymentów należy podkreślić, że:

1. Nieznane są dokładne algorytmy syntezy zaimplementowane w komercyjnych narzędziach wspomagających projektowanie, przeznaczone dla struktur programowalnych (np. MAX+PLUS, Synplify itp.
2. W wielu przypadkach, przede wszystkim dla rozbudowanych struktur CPLD (MAX7000, MACH, FLASH370), zaprezentowane porównanie jest tylko orientacyjne.
3. Trudno dokładnie porównać czasy syntezy. W większości przypadków systemy komercyjne działają szybciej.

Ad 1

Większość komercyjnych algorytmów syntezy chroniona jest szeregiem patentów. Opisy zawarte w dokumentacji są bardzo lakoniczne, a próby dotarcia do dokładnych informacji zawsze kończyły się fiaskiem. W tej sytuacji jedyną drogą, która pozwoliła na głębsze poznanie metod syntezy zaimplementowanych w firmowych narzędziach programowych, była analiza uzyskiwanych wyników. Analiza ta pozwala przypuszczać, że w większości przypadków dominującą rolę w syntezie prowadzonej dla struktur CPLD typu PAL odgrywa proces dwupoziomowej minimalizacji. Dominująca rola minimalizacji, wykonywanej w oderwaniu od zasobów docelowej struktury, często uniemożliwia znalezienie efektywnych rozwiązań (np. *rd84* – patrz tab. 5.2, 5.4, 5.10).

Należy przypuszczać, że efektywność zaproponowanych strategii syntezy wynika przede wszystkim z zastosowania elementów syntezy, dopasowujących projektowany układ do specyficznych cech wykorzystywanej struktury programowalnej.

Ad 2

Uniwersalność zaproponowanych elementów syntezy wynika ze sposobu scharakteryzowania struktur CPLD poprzez blok logiczny typu PAL. Pozwala to na uniezależnienie procesu syntezy od konkretnej rodziny układów, a równocześnie dopasowanie go do właściwości struktury programowalnej. Rozbudowane układy CPLD mają szereg dodatkowych elementów, które przede wszystkim umożliwiają nierównomierny rozdział iloczynów do poszczególnych sum lub realizację różnego typu „szybkich" sprzężeń zwrotnych. W tej sytuacji należałoby rozszerzyć proponowane strategie syntezy wplatając w nie dodatkowe informacje, związane z różnorodnymi konfiguracjami bloków logicznych. Opis takich możliwości zawarty jest między innymi w pracach [Kani00d,Kani01c]. Opracowanie kompletnych, automatycznych narzędzi do syntezy dla różnych rodzin układów CPLD, choć możliwe, nie było celem autora.

W tej sytuacji zaprezentowane porównania, wspierane często różnymi uproszczeniami i interakcyjnymi decyzjami, wskazują przede wszystkim na drzemiące w zaproponowanym podejściu możliwości znacznej poprawy komercyjnych metod syntezy. Szczególnie kłopotliwe jest porównanie właściwości dynamicznych, ponieważ szereg elementów (np. ekspandery, programowalne rozdzielacze itp.) zostało wprowadzonych do struktury programowalnej w celu zwiększenia „szybkości" pracy, realizowanego w strukturze programowalnej układu. W tej sytuacji tym bardziej cenne są wyniki, w których mimo niewykorzystywania ekspanderów udało się uzyskać lepsze rozwiązania pod względem dynamicznym (tab. 5.6), na dodatek w nieznacznie dłuższym czasie.

Ad 3

Najbardziej kłopotliwe jest porównanie czasów syntezy. Przedstawione algorytmy syntezy były implementowane w ciągu wielu lat w jednym z języków Pascal lub C. W pierwszej części pracy porównano opracowane metody syntezy z odpowiednią implementacją (Pascal, C) metody klasycznej. Identyczny zestaw danych wejściowych (opis w formacie *.pla), danych wyjściowych (raport syntezy) oraz język implementacji algorytmu (Pascal lub C) pozwolił na dokładne porównania zaproponowanych metod z metodą klasyczną. Nie ulega wątpliwości, że najbardziej czasochłonne są metody bazujące na dekompozycji.

Porównanie złożonych strategii syntezy, przede wszystkim dla rozbudowanych układów CPLD (MAX5000, MAX7000, MACH, FLASH), jest niezwykle kłopotliwe, ponieważ:

- wyniki eksperymentów dla złożonych struktur CPLD uzyskano stosując metody interakcyjne;
- występowały znaczne różnice w zbiorach danych wejściowych i wyjściowych;

  *W przypadku narzędzi firmowych dane wejściowe stanowił opis układów testowych w postaci różnorodnych konstrukcji językowych (AHDL, VHDL) w odróżnieniu od zaproponowanych metod, w których dane zawsze były w postaci pliku w formacie *.pla. Większe różnice występowały dla danych wyjściowych. Systemy firmowe tworzyły opis połączeń w postaci różnorodnych plików. Zaproponowane metody tworzyły raport syntezy (liczba bloków, liczba iloczynów, liczba warstw) i opis uzyskiwanej struktury, najczęściej w języku VHDL. W wielu sytuacjach zrealizowanie syntezy do końca jest niemożliwe ze względu na brak informacji o sposobie opisu połączeń.*
- wykorzystano gotowy moduł do minimalizacji (Espresso).

6. PODSUMOWANIE

Rozwój technologiczny umożliwił wytwarzanie układów cyfrowych, które mogą być dostosowywane do indywidualnych potrzeb użytkownika (ASIC). Okazało się, że ta forma implementacji układów cyfrowych wymaga stosowania nowych metod syntezy. Etap syntezy logicznej w wielu przypadkach wywiera decydujący wpływ na ostateczną realizację układów. Dynamiczny rozwój syntezy logicznej trwa do chwili obecnej. Mimo swego burzliwego charakteru ciągle pozostają nierozwiązane problemy a dodatkowo w ostatnim czasie pojawiają się nowe wyzwania, związane na przykład z reprogramowalnością układów.

Niniejsza praca poświęcona jest syntezie logicznej, przeznaczonej dla jednej z grup układów PLD (podgrupa układów ASIC, w której dostosowywanie do potrzeb użytkownika odbywa się poprzez programowanie). Możliwości układów PLD są niezwykle atrakcyjne w Polsce. Ograniczenia technologiczne uniemożliwiają swobodne korzystanie z innych grup układów scalonych produkowanych na zamówienie (ASIC). W tej sytuacji struktury programowalne (CPLD, FPGA) umożliwiają projektowanie i wykonywanie złożonych układów cyfrowych bez wykorzystywania niedostępnych w Polsce linii technologicznych.

Przedmiotem zainteresowania są struktury CPLD, których jądro stanowi struktura typu PAL, wyposażone w programowalną matrycę AND i stałe połączenia w matrycy OR. Podstawowym elementem tego typu układów jest tzw. blok logiczny typu PAL, składający się z kilku iloczynów dołączonych do komórki wyjściowej. Blok logiczny typu PAL zawiera często dodatkowe elementy charakterystyczne dla wąskiej grupy układów jednego z producentów. Należą do nich między innymi wyjściowe bufory trójstanowe, bramki XOR, programowalne rozdzielacze, różnego typu ekspandery itp. Kluczowym problemem syntezy przeznaczonej dla tego typu struktur jest efektywne wykorzystanie zawartych w nich iloczynów.

Klasyczna synteza logiczna oparta jest na dwupoziomowej minimalizacji poszczególnych funkcji, po której występuje etap dopasowania technologicznego funkcji do wykorzystywanej struktury. Strategie syntezy zaimplementowane w firmowych narzędziach wspomagających projektowanie dla tego typu układów, obwarowane szeregiem patentów, przeznaczone są najczęściej dla wąskiej grupy układów konkretnego producenta. Mimo tych istotnych uproszczeń często nie prowadzą do efektywnych rozwiązań.

Strategie syntezy zaimplementowane w kompilatorach języków opisu sprzętu (VHDL, Verilog) wykorzystują różne techniki odwzorowujące sieć logiczną na postać możliwą do realizacji w zadanej strukturze programowalnej. Metody te bazują na wielopoziomowej reprezentacji funkcji, a proces syntezy sprowadza się do dopasowania topologicznego drzewa wielopoziomowej reprezentacji funkcji do wzorców znajdujących się w bibliotece technologicznej. Metody te, oprócz niewątpliwych zalet, do których należy przede wszystkim łatwe przenoszenie projektów pomiędzy różnymi technologiami, wprowadzają często duże zaburzenia, prowadzące ostatecznie do nieefektywnego wykorzystywania iloczynów.

Pojawia się zatem oczywisty wniosek, iż celowe jest stosowanie takich metod syntezy, które umożliwiają implementację sieci logicznej w postaci elementów dostępnych w danej

grupie układów programowalnych, wprowadzając charakterystyczne cechy wybranych układów w początkowe etapy syntezy. Tylko takie podejście zapewnia efektywne wykorzystanie zasobów struktur programowalnych. Badania w tym kierunku prowadzone są w różnych ośrodkach na całym świecie. Wystarczy wspomnieć o poszukiwaniu od wielu lat efektywnej strategii dekompozycji funkcji, przeznaczonej dla struktur FPGA typu tablicowego.

Metody syntezy będące przedmiotem niniejszej pracy wykorzystują specyficzne właściwości struktur matrycowych, zawarte w zaproponowanej strukturze bloku logicznego typu PAL. Zaproponowanie podziału całego układu matrycowego na bloki logiczne typu PAL umożliwia opracowanie ogólnych metod syntezy dla powszechnie występujących zasobów struktury matrycowej. Podejście to nie ogranicza jednak końcowej optymalizacji, uwzględniającej specyficzne właściwości konkretnego układu. Projektowany układ cyfrowy najczęściej optymalizowany jest pod kątem wykorzystania zasobów struktur programowalnych (minimalizacja powierzchni; liczby wykorzystywanych bloków logicznych typu PAL) lub czasu propagacji sygnałów (minimalizacja liczby warstw logicznych). Zaproponowane metody syntezy logicznej obejmują te dwa obszary optymalizacyjne.

Minimalizację liczby wykorzystywanych bloków logicznych uzyskuje się za pomocą wyszukiwania wspólnych grup wielowyjściowych implikantów (rozdział 2). Implikanty te mogą być realizowane we wspólnych dla kilku funkcji blokach logicznych. Zaproponowane metody stanowią pewną odmianę klasycznej syntezy wielopoziomowej, w której wykorzystano minimalizację dwupoziomową oraz zasady podziału implikantów dostosowane do struktury programowalnej.

Prawie we wszystkich układach matrycowych występują wyjściowe bufory trójstanowe, które wplecione w etap syntezy mogą wydatnie wpłynąć na minimalizację warstw logicznych uzyskiwanych rozwiązań (rozdział 3). Istota syntezy sprowadza się do odpowiedniego wyboru zmiennych (wektorów wejściowych) sterujących trójstanowymi buforami wyjściowymi, umożliwiając łączenie wyjść trójstanowych bloków logicznych typu PAL. W większości przypadków uzyskuje się tą metodą syntezy struktury jednowarstwowe.

Niezwykle cenny element syntezy, wpływający przede wszystkim na znaczną minimalizację liczby bloków logicznych, stanowi dekompozycja. Takie nietypowe zastosowanie teorii klasycznej dekompozycji jest przedmiotem rozważań zawartych w rozdziale 4. Opracowana metoda dekompozycji wierszowej funkcji jednowyjściowej pozwala na dopasowanie uzyskiwanych w efekcie dekompozycji podukładów do struktury bloku logicznego zawartego w układzie programowalnym (rozdział 4.5).

Doświadczenia praktyczne zdobyte podczas projektowania układów cyfrowych w strukturach programowalnych z użyciem różnorodnych narzędzi programowych (Synopsys, Warp, Synplify, ABEL, CUPL, SNAP, MAX+PLUS, Foundation itp.) stanowiły podstawę naukowego warsztatu i jednoznacznie wskazały na celowość opracowania przedstawionych nowatorskich metod syntezy. Niebagatelne były również doświadczenia zdobyte podczas tworzenia systemu DECOMP, przeznaczonego dla układów FPGA typu tablicowego.

Przedstawione w pracy metody syntezy zostały zaimplementowane w postaci prototypowych programów komputerowych, stwarzających możliwość natychmiastowego weryfikowania, modyfikowania i udoskonalania poszczególnych etapów syntezy.

Przeprowadzone eksperymenty [Bench] wskazują jednoznacznie na możliwość poprawy efektywności komercyjnych metod syntezy zarówno pod względem powierzchni, jak i szybkości. Autor pracy daleki jest jednak od stwierdzenia, że wszystkie przedstawione algorytmy nadają się do bezpośredniego zastosowania w systemach komercyjnych. Jednakże

zdaniem autora istnieją elementy, między innymi opracowane „szybkie" algorytmy syntezy, wykorzystujące trójstanowe bufory wyjściowe, które można natychmiast włączyć do komercyjnych systemów syntezy. W praktyce inżynierskiej pozwoliły one wielokrotnie zrealizować projektowany „szybki" układ w dostępnej, zwykle niewielkiej strukturze programowalnej.

Zdaniem autora do najważniejszych osiągnięć pracy należy zaliczyć:

- opracowanie podstaw teoretycznych i wynikających z nich algorytmów syntezy, pozwalających na wyszukiwanie odpowiednich grup wielowyjściowych implikantów,
- opracowanie metod syntezy umożliwiających wykorzystanie powszechnie występujących w strukturach typu PAL trójstanowych buforów wyjściowych,
- opracowanie metody minimalizacji z rozłączaniem implikantów,
- opracowanie metody jednowarstwowej realizacji funkcji w strukturach typu PAL,
- rozszerzenie teorii dekompozycji funkcjonalnej (dekompozycja kolumnowa, dekompozycja wierszowa),
- opracowanie poszczególnych elementów dekompozycji kolumnowej pod kątem efektywnego wykorzystania zasobów logicznych struktur typu PAL,
- wprowadzenie i wykorzystanie w procesie syntezy kodowania równomiernego,
- opracowanie podstaw teoretycznych dekompozycji wierszowej,
- opracowanie strategii syntezy przeznaczonych dla różnorodnych układów programowalnych typu PAL,
- przedstawienie sposobu specyfikacji w języku VHDL umożliwiającego dalszą efektywną syntezę w komercyjnych narzędziach wspomagających projektowanie,
- opracowanie prototypowego oprogramowania implementującego przedstawione algorytmy syntezy, umożliwiającego współpracę z systemami komercyjnymi.

Wieloletnie zainteresowania autora problemami syntezy logicznej wskazują, że zawsze przy rozwiązywaniu jednego problemu syntezy równocześnie pojawia się szereg innych, równie interesujących zagadnień. Przedstawione w niniejszej pracy problemy mogą stanowić podstawę dalszych prac, wśród których zdaniem autora do najciekawszych można zaliczyć:

- uszczegółowienie przedstawionych metod syntezy dla różnych rodzin układów CPLD,
- wykorzystanie w procesie dekompozycji diagramów BBD,
- zastosowanie w procesie syntezy elementów minimalizacji symbolicznej.

Autor ma również nadzieję, że niektóre elementy syntezy przedstawione w tej pracy można będzie wykorzystać dla innych rodzin układów programowalnych, być może również tych, które pojawią się na rynku w przyszłości.

LITERATURA

[Abel90] DataI/O, ABEL. Design software - user manual, 1990

[AbouB93] Abouzeid P., Babba B., Crastes M., Saucier G., Input-Driven Partitioning Methods and Application to Synthesis on Table-Lookup-based FPGAs, IEEE Transactions on Computer-Aided Design, Vol. 12, No. 7, 1993, pp. 913-925

[AbouS90] Abouzeid P., Sakoni K., Saucier G., Poirot F., Multilevel Synthesis Minimizing the Routing Factor, Proceedings of Design Automation Conference, DAC'90, 1990, pp.365-368

[Acte95] Actel, FPGA Data Book and Design Guide, 1995

[Adam98] Adamski M., Metodologia projektowania reprogramowalnych sterowników logicznych z wykorzystaniem elementów CPLD i FPGA, Materiały I Krajowej Konferencji Naukowej Reprogramowalne Układy Cyfrowe, RUC'98, Szczecin, 12-13 marca 1998, ss. 15-22

[AdamW97] Adamski M., Węgrzyn M., Wykorzystanie standardowych kompilatorów PLD do syntezy sterowników logicznych, Kwartalnik Elektroniki i Telekomunikacji Polskiej Akademii Nauk, 1997, t. 43, z. 3, pp.335-354

[Adva95] AMD, MACH 1,2,3, and 4 Family Data Book, 1995

[Adva96] AMD, PAL Devices Data Book and Design Guide, 1996

[Aker78] Akers S.B., Binary Decision Diagrams, IEEE Transactions on Computers, Vol. C-27, No.6, June 1978, pp.509-516

[Alte95] Altera, MAX+PLUS II AHDL, 1995

[Alte96] Altera, MAX+PLUS II VHDL, 1996

[Alte97] Altera, MAX+PLUS II, Getting started, 1997

[Alte99] Altera, Device Data Book, 1999

[AndeB98] Anderson J.H., Brown S.D., Technology mapping for large complex PLDs, Proceedings of Design Automation Conference, DAC'98, 15-19 January, 1998, pp.698 -703

[AshaD92] Ashar P., Devadas S., Newton A.R., Sequential Logic Synthesis, Kluwer Academic Publisher, Boston/London/Dotdrecht, 1992

[Ashe57] Ashenhurst R.L., The decomposition of switching functions, Proceedings of an International Symposium on the Theory of Switching, April 1957, (przedruk w dodatku do Curt62)

[Atme00] Atmel, Programmable Logic and System-Level ICs, 2000

[BabbC92] Babba B., Crastes M., Saucier G., Input driven synthesis on PLDs and PGAs, The European Conference on Design Automation, Brussels, Belgium, March 1992, pp.48-52

[Bench] Collaborative Benchmarking Laboratory, Department of Computer Science at North Carolina State University, http://www.cbl.ncsu/edu/

[Bolt90] Bolton M., Digital Systems Design with Programmable Logic, Addison-Wesley Publishing Company, 1990

[Bout98] Bout D.V., The Practical Xilinx Designer Lab Book, Prentice Hall, 1998

[BranS93] Brand D., Sasao T., Minimization of AND-EXOR Expressions using Rewrite Rules, IEEE Transaction on Computers, Vol. 42, No. 5, May 1993

[BrayH84] Brayton R.K., Hachtel G.D., McMullen C. and Sangiovanni-Vincentelli A.L., Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Boston 1984

[BrayH90] Brayton R.K., Hachtel G.D., Sangiovanni-Vincentelli A.L., Multilevel logic synthesis, Proceedings of the IEEE, Vol.78, No.2, 1990, pp. 264-300,

[BrayM82] Brayton R.K., McMullen C., The Decomposition and Factorization of Boolean Expression, Proceedings of International Symposium on Circuits and Systems, 1982, pp.49-54

[BrayR87] Brayton R.K., Rudell R., Sangiovanni-Vincentelli A., Wang A., MIS: a Multiple-Level Logic Optimization System, IEEE Transactions on Computer-Aided Design, Vol. CAD-6. No.6, November 1987, pp.1062-1081

[BrowF93] Brown S.D., Francis R.J., Rose J., Vranesic Z.G., Field Programmable Gate Arrays, Kluwer Academic Publishers, Boston/London/Dotdrecht, 1993

[Brya93] Bryant V., Aspects of Combinatorics, Cambridge University Press, 1993

[BrzoŁ97] Brzozowski J.A., Łuba T., Decomposition of Boolean Functions Specified by Cubes, University of Waterloo Computer Science Department, CS-97-01, January 1997 (ftp://cs-archive.uwaterloo.ca/cs-archive/CS-97-01/CS-97-01

[BurnP98] Burns M., Perkowski M., Jóźwiak L., An Efficient Approach to Decomposition of Multi-Output Boolean Functions with Large Sets of Bound Variables, Proceedings of Euromicro Conference, Vasteras, Sweden, August 25-27, 1998, pp. 16-23

[CarpM98] Carpenter A., Messer N., The Use of VHDL+ in The Specification Level Modeling of An Embedded System, Proceedings of the Forum on Design Languages, Luasanne, Switzerland, Sept. 6-11 1998, pp. 83-92.

[ChanM96] Chang S., Marek-Sadowska M., Hwang T., Technology Mapping for TLU FPGA's Based on Decomposition of Binary Decision Diagrams, IEEE Transactions on Computer-Aided Design, Vol.15, No.10, October 1996, pp. 1226-1235

[ChatR94] Chattopadhyay S., Roy S., Chaudhuri P., KGPMAP: library-based technology-mapping technique for antifuse based FPGAs, IEE Proceedings-Computers and Digital Techniques, Vol. 141, November 1994, pp. 361-368

[ChenH02] Chen S. L., Hwang T. T., Liu C.L., A technology mapping algorithm for CPLD architectures, IEEE International Conference on Field-Programmable Technology, Hong Kong, December 16-18, 2002, pp. 204-210

[ChenM88] Chen K. C., Muroga S., Input assignment algorithm for decoded-PLAs with multi-input decoders, IEEE International Conference on Computer-Aided Design, ICCAD'88, Digest of Technical Papers, 7-10 November 1988, pp. 474-477

[Chri75] Christofides N., Graph Theory - An Algorithmic Approach, Academic Press, New York, London, San Francisco, 1975

[CiesY92] Ciesielski M. J., Yang S., PLADE: A two-stage PLA decomposition, IEEE Transactions on Computer-Aided Design, Vol.11, No.8, August 1992, pp. 943-954

[CongD94] Cong J., Ding Y., FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Design, IEEE Transactions on Computer-Aided Design, Vol.13, No.1, 1994, pp. 1-12

[CongH01] Cong J., Hwang Y-Y., Boolean Matching for LUT-Based Logic Blocks With Applications to Architecture Evaluation and Technology Mapping, IEEE Transactions on Computer-Aided Design, Vol.20, No.9, 2001, pp.1077-1090

[CongW98] Cong J., Wu C., An Efficient Algorithm for Performance-Optimal FPGA Technology Mapping with Retiming, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No.9, 1998, pp. 738-748

[Curt62] Curtis H.A., The Design of switching Circuits, D.van Nostrand Company, Inc., Princeton, New Jersey, Toronto, New York, 1962

[Curt63] Curtis H.A., Generalized tree circuit - the basic building block of an extended decomposition theory, J.ACM, Vol.10, 1963, pp.562-581

[Cypr96] Cypress, Data Book, 1996

[CzerK02] Czerwiński R., Kania D., Nocuń P., Kodowanie stanów dedykowane dla struktur typu PAL, Krajowe Sympozjum Telekomunikacji, KST2002, vol. A, Bydgoszcz, wrzesień 2002, ss.65-71

[CzerK03] Czerwiński R., Kania D., Metody kodowania stanów automatów sekwencyjnych oparte na wyborze aktywności wyjść, RUC 2003, Szczecin, 8-9 maja, 2003, ss. 9-16

[DageA86] Dagenais M., Agarwal V., Rumin N., McBOOLE: A New Procedure for Exact Logic Minimization, IEEE Transactions on CAD/ICAS, Vol. CAD-4, No.3, 1986, pp.269-284

[DeniS94] Deniziak S., Sapiecha K., CUPLAND-a behavioral level description compiler for designing of PLD/EPLD-based systems, IEEE International Symposium on Circuits and Systems, ISCAS '94, Vol. 1, 30 May-2 Jun 1994, pp. 201-204

[Deva93] Devadas S., Comparing Two-Level and Ordered Binary Decision Diagram Representations of Logic Functions, IEEE Transactions on Computer-Aided Design, Vol.12, No.5, May 1993

[DevaN91] Devadas S., Newton A.R., Exact Algorithms for Output Encoding, State Assignment, and Four-Level Boolean Minimization, IEEE Transactions on Computer-Aided Design, Vol. 10, No. 1, January 1991

[DevaW88a] Devadas S., Wang A.R., Newton A.R., A. Sangiovanni-Vincentelli, Boolean Decomposition of Programmable Logic Arrays, IEEE Custom Integrated Circuits Conference, May 1988, pp. 2.5.1 -2.5.5

[DevaW88b] Devadas S., Wang A.R., Newton A.R., Sangiovanni-Vincentelli A., Boolean decomposition in multi-level logic optimization, Digest of Technical Papers, IEEE International Conference on Computer-Aided Design, ICCAD-88, 7-10 Nov 1988, pp.290 -293

[Drec99] Drechsler R., Checking Integrity during Dynamic Reordering in Decision Diagrams, Proceedings of the 25th EUROMICRO Conference, IEEE Computer Society, Milan, Italy, September 8-10, 1999, pp. 360-367

[DzikH01] Dzikowski A., Hrynkiewicz E., Metody dekompozycji zespołu funkcji z wykorzystaniem diagramów ROBDD, IV Krajowa Konferencja Naukowa, Reprogramowalne Układy Cyfrowe, RUC'2001, Szczecin, maj 2001, ss.19-28

[DzikH03] Dzikowski A., Hrynkiewicz E., Złożona dekompozycja obszarowa zespołu funkcji logicznych z wykorzystaniem diagramów ROBDD, II Krajowa Konferencja Elektroniki, KKE'03 Kołobrzeg, czerwiec 2003, tom II, ss.393-398

[DzikH99] Dzikowski A., Hrynkiewicz E., Sposób wyszukiwania wspólnych grup węzłów grafu ROBDD przy dekompozycji wielowyjściowych funkcji logicznych, KST'99 Bydgoszcz, 1999, tom B-1.30, ss. 259-266

[Fawc95] Fawcett B.K., Map, place and route: the key to high-density PLD implementation, Microelectronics Communications Technology Producing Quality Products Mobile and Portable Power Emerging Technologies, WESCON/'95, 7-9 Nov 1995, pp. 292-297

[FiloY91] Filo D., Yang J.C., Mailhot F., De Micheli G., Technology Mapping for Two-Output RAM-based Field Programmable Gate Array, Proceedings of European Design Automation Conference, February, 1991, pp.534-538

[FranR91] Francis R.J., Rose J., Vranesic Z., Chortle-Crf: Fast Technology Mapping for Lookup Table-based FPGAs, Proceedings of the Design Automation Conference, 1991, pp. 227-233

[FujiF93] Fujita M., Fujisawa H., Matsunaga Y., Variable Ordering Algorithms for Ordered Binary Decision Diagrams and Their Evaluation, IEEE Transactions on CAD, Vol.12, No.1, January 1993

[GajsD92] Gajski D., Dutt N., Wu A., Lin Y., High-Level Synthesis: Introduction to Chip and System Design, Kluwer Academic Publishers, 1992

[GajsD98] Gajski D., Doemer R., Zhu J., IP-centric Methodology and Design with the SpecC Language, Materiały szkoleniowe szkoły letniej zorganizowanej przez NATO Advance Study Institute na temat "System Level Synthesis", Barga, Włochy, 11-20 sierpnia 1998

[GarbG02] Garbolino T., Gucwa K., Hławiczka A., Kania D., Kardaszewicz J., Kulisz J., A.Morawiec, Laboratorium podstaw techniki cyfrowej, Wydawnictwo Politechniki Śląskiej, Gliwice 2002

[Gree86] Green D., Modern logic design, Addison-Wesley Publishing, 1986

[HuanJ00] Huang J-D., Jou J-Y, Shen W-Z, ALTO: An Iterative Area/Performance Tradeoff Algorithm for LUT-Based FPGA Technology Mapping, IEEE Transactions on Very Large Integration (VLSI) Systems, Vol. 8, No.4, August 2000, pp. 392-400

[HongC74] Hong S., Cain R., D.Ostapko, MINI: A heuristic approach for logic minimization, IBM Journal of Research and Development, Vol.18, 1974, pp.443-458

[HrynK03] Hrynkiewicz E., Kania D., Impact of decomposition direction on synthesis effectiveness, Programmable Devices and System, PDS'03, February 11-13, Ostrava, 2003, pp.144-149

[HrynK95] Hrynkiewicz E., Kania D., Dekompozycja układów kombinacyjnych pod kątem realizacji na strukturach PLD o niewystarczającej liczbie termów, KST'95, Bydgoszcz, 6-8 września 1995, ss. 366-373

[HrynK96] Hrynkiewicz E., Kania D., Niemczyk A., Logic Function Set Minimization Using the Method of Usefulness Anticipation if the Generated Implicants, XIXth National Conference Circuits Theory and Electronic Networks Kraków-Krynica, Poland, October 23-26, 1996, pp.127-132

[HrynP97] Hrynkiewicz E., Pucher K., Kania D., The input partitioning and coding problem in PAL-based CPLDs, XXth National Conference CT&EN, Kołobrzeg, Poland, October 21-24, 1997, ss.145-152

[HsuL91] Hsu Y.Ch., Lin Y.L., Hsied H.Ch., Chao T.H., Combining Logic Minimization and Folding for PLA's, IEEE Transactions on Computers, Vol. 40, No. 6, June 1991

[IgucS00] Iguchi Y., Sasao T., Matsuura M., Implementation of multiple-output functions using PQMDDs, Proceedings 30th IEEE International Symposium on Multiple-Valued Logic, ISMVL'00, 2000, pp. 199 -205

[IgucS03] Iguchi Y., Sasao T., M.Matsuura, Evaluation of multiple-output logic functions using decision diagrams, Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC'03, 2003, pp. 312 –315

[JacoT92] Jacobi R.P., Trullemans A.M., Generating Prime and Irredundant Covers for Binary Decision Diagrams, The European Conference on Design Automation, Brussels, Belgium, March 1992, pp.104-108

[JozwC99] Jóźwiak L., Chojnacki A., Functional Decomposition based on Information Relationship Measures Extremely, Proceedings of the 25th EUROMICRO Conference, IEEE Computer Society, Milan, Italy, September 8-10, 1999, pp.150-160

[JozwV95] Jóźwiak L., Volf F., Efficient decomposition of assigned sequential machines and Boolean functions for PLD implementations, Proceedings of Electronic Technology Directions to the Year 2000, 23-25 May 1995, pp. 258 –266

[Kali91] Kalisz J., Podstawy elektroniki cyfrowej, WKŁ, 1991

[KamiM00] Kamionka-Mikuła H., Małysiak H., Pochopień B., Układy cyfrowe. Teoria i przykłady, Wydawnictwo Jacka Skalmierskiego, Gliwice 2000

[Kani00a] Kania D., Decomposition-based synthesis and its application in PAL-oriented technology mapping, Proceedings of 26-th Euromicro Conference, IEEE Computer Society Press, Maastricht, 2000, pp. 138-145

[Kani00b] Kania D., Heurystyczna metoda dekompozycji zespołu funkcji boolowskich wykorzystująca dekompozycje złożone, przeznaczona dla układów FPGA typu tablicowego, Kwartalnik Elektroniki i Telekomunikacji, 2000, 46, z.2, ss.195-210

[Kani00c] Kania D., A technology mapping algorithm for PAL-based devices using multi-output function graphs, Proceedings of 26-th Euromicro Conference, IEEE Computer Society Press, Maastricht, 2000, pp.146-153

[Kani00d] Kania D., Coding capacity of PAL-based logic blocks included in CPLDs and FPGAs, IFAC Workshop on Programmable Devices and Systems, PDS 2000, Ostrava, February 8-9, Published for the IFAC by PERGAMON, An Imprint of Elsevier Science, 2000, pp. 164-169

[Kani00e] Kania D., Sposób kodowania stanów wewnętrznych automatów sekwencyjnych realizowanych w strukturach programowalnych ze stałym (H, L) lub programowalnym poziomem aktywności wyjścia, zgłoszenie patentowe P 344029, 2000

[Kani00f] Kania D., Logic Decomposition for CPLD Synthesis, IFAC Workshop on Programmable Devices and Systems, PDS 2000, Ostrava, February 8-9, Published for the IFAC by PERGAMON, An Imprint of Elsevier Science, 2000, pp. 49-52

[Kani00g] Kania D., Synteza logiczna wielopoziomowych układów w strukturach typu PAL z trójstanowymi buforami wyjściowymi, Kwartalnik Elektroniki i Telekomunikacji, 2000, 46, z.1, ss.81-90

[Kani00h] Kania D., Struktury programowalne - strategia syntezy logicznej, Elektronizacja, nr 10, 2000, ss.17-19

[Kani00i] Kania D., Wybór sposobu realizacji wielowyjściowych funkcji logicznych w strukturach CPLD typu PAL, RUC'2000, Szczecin, ss. 88-96

[Kani01a] Kania D., Metody dekompozycji funkcji przeznaczone dla układów FPGA typu tablicowego. Krajowe Sympozjum Telekomunikacji, KST2001, Bydgoszcz, wrzesień 2001, pp.48-56

[Kani01b] Kania D., Sposób kodowania równomiernie wykorzystujący iloczyny zawarte w strukturach programowalnych typu PAL, zgłoszenie patentowe, P 345784, 30.01.2001

[Kani01c] Kania D., Realizacja układów kombinacyjnych w strukturach MACH, Kwartalnik Elektroniki i Telekomunikacji, 2001, 47, z. 1, ss.65-74

[Kani02a] Kania D., An Efficient Algorithm for Output Coding in PAL-based CPLDs, International Journal of Engineering, Vol.15, No.4, November 2002, pp.325-328

[Kani02b] Kania D., Synteza logiczna dla struktur typu PAL wykorzystująca bufory wyjściowe, Kwartalnik Elektroniki i Telekomunikacji, 2002, 48, z.1, ss. 53-66

[Kani02c] Kania D., Logic Synthesis of Multi-Output Functions for PAL-based CPLDs, IEEE International Conference on Field-Programmable Technology, Hong Kong, December 16-18, 2002, pp. 429-432

[Kani02d] Kania D., Improved Technology Mapping for PAL-based Devices Using a New Approach to Multi-Output Boolean Functions, DATE 02, Paris, March 2002, p.1087

[Kani03a] Kania D., Dekompozycja wielokrotna w syntezie logicznej dla struktur FPGA typu tablicowego, Elektronika, nr 2-3, 2002, ss. 43-46

[Kani03b] Kania D., An Efficient Approach to Synthesis of Multi-Output Boolean Functions on PAL-based Devices, IEE Proceedings - Computer and Digital Techniques, Vol. 150, No. 3, May 2003, pp.143-149

[Kani03c] Kania D., Synteza logiczna funkcji w strukturach typu PAL z trójstanowymi buforami wyjściowymi, RUC 2003, Szczecin, 8-9 maja, 2003, ss. 21-28

[Kani03d] Kania D., Synteza logiczna wielowyjściowych funkcji w strukturach typu PAL, RUC 2003, Szczecin, 8-9 maja, 2003, ss. 29-36

[Kani95] Kania D., Tablicowe metody dekompozycji układów kombinacyjnych. Realizacja tych układów na wybranych strukturach programowalnych, rozprawa doktorska, Politechnika Śląska, Gliwice 1995

[Kani96] Kania D., Complex decomposition of multiple-output functions, International Conference on Programmable Devices and Systems, PDS'96, Ostrava, Czech Republic, November 26-28 1996, pp. 86-91

[Kani98a] Kania D., An algorithm of functional decomposition with free set variables coding, Kwartalnik Elektroniki i Telekomunikacji, 1998, 44, z.3, pp.317-324

[Kani98b] Kania D., Coding Capacity of Programmable Transcoder, Electronics and Telecommunications Quarterly, 1998, 44, z.2, pp. 193-204

[Kani99a] Kania D., Two-level logic synthesis on PAL-based CPLD and FPGA using decomposition, Proceedings of 25-th Euromicro Conference, IEEE Computer Society Press, Milan, 1999, pp. 278-281

[Kani99b] Kania D., Two-level logic synthesis on PALs, Electronics Letters, 1999, Vol.35, No. 11, pp. 879-880

[Kani99c] Kania D., Efektywna metoda realizacji zespołu funkcji w strukturach typu PAL, Kwartalnik Elektroniki i Telekomunikacji, 1999,45, z. 3-4, ss. 433-444

[Kani99d] Kania D., Algorytmy podziału wyjść umożliwiające realizację układów cyfrowych w strukturach PLD, Kwartalnik Elektroniki i Telekomunikacji, 1999, 45, z.2, pp. 189-202;

[Kani99e] Kania D., Metody ekspansji liczby wyjść układów PLD. Krajowe Sympozjum Telekomunikacji, KST'99, 8-10 wrzesień, Bydgoszcz, 1999, ss. 170-177

[Kani99f] Kania D., Coding Capacity of PAL-based Programmable Transcoder with Uneven Number Terms per Output, Kwartalnik Elektroniki i Telekomunikacji, 1999, 45, z.1, pp.73-84

[Kani99g] Kania D., Synteza logiczna dla układów CPLD typu PAL wykorzystująca dekompozycję, Kwartalnik Elektroniki i Telekomunikacji, 1999,45, z. 3-4, ss. 445-454

[Kani99h] Kania D., Impact of logic minimization on term partitioning effectiveness, Proceedings of the XXIInd National Conference on Circuits Theory and Electronic Networks, Warszawa-Stare Jabłonki, October 22-24, 1999, pp. 131-136

[Kani99i] Kania D., Podział termów umożliwiający realizację układów cyfrowych w strukturach CPLD i FPGA typu PAL, RUC'99, Szczecin, ss. 241-248

[KaniH96] Kania D., Hrynkiewicz E., Variable partitioning method based on subdecomposition, XIXth National Conference Circuits Theory and Electronic Networks, Kraków-Krynica, Poland, October 23-26, 1996, pp. 121-126

[KaniH98] Kania D., Hrynkiewicz E., Pucher K., Coding capacity of PAL-based devices with different number terms per output, Proceedings of the XXIst National Conference on Circuits Theory and Electronic Networks, Poznań-Kiekrz, October 22-24, 1998, pp. 203-208

[KaniS92a] Kania D., Szreter J., Programowalne układy logiczne FPLA, FPLS, Elektronizacja nr 7, 1992

[KaniS92b] Kania D., Szreter J., Projektowanie układów cyfrowych z wykorzystaniem układów PLD, Elektronizacja nr 12, 1992

[KaniS93a] Kania D., Szreter J., Metody opisu urządzeń cyfrowych z układami PLD, Elektronizacja nr 3, 1993

[KaniS93b] Kania D., Szreter J., Języki opisu w projektowaniu urządzeń z PLD, Elektronizacja nr 1, 1993

[Kapo95] Kapoor B., Improved Technology Mapping Using A New Approach to Boolean Matching, Proceedings of ED&TC'95, March 6-9, Paris, France, 1995, pp.86-90

[Karp91a] Karplus K., Xmap: a Technology Mapper for Table-lookup Field-Programmable Gate Arrays, Proceedings of 28th DAC, June 1991, pp. 240-243

[Karp91b] Karplus K., Amap: a Technology Mapper for Selector-based Field-Programmable Gate Arrays, Proceedings of 28th DAC, June 1991, pp. 244-247

[KimB99] Kim J., Byun S., Kim H., Development of technology mapping algorithm for CPLD under time constraint, 6th International Conference on VLSI and CAD, ICVC '99, 1999, pp.411-414

[KimK00] Kim H-S., Kim J-J., Lin Ch-H., An efficient CPLD technology mapping under the time constraint, Proceedings of the 12th International Conference on Microelectronics, ICM 2000, 2000, pp.265 -268

[KimK01] Kim J-J., Kim H-S., Lin Ch-H., A new technology mapping for CPLD under the time constraint, Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC'01, 2001, pp.235 -238

[Koło03] Kołodziński S., Dekompozycja funkcji logicznych metodami spektralnymi zorientowana na realizację tych funkcji w układach FPGA typu tablicowego, rozprawa doktorska, Gliwice 2003

[KoulG91] Kouloheris J.L., Gamal El, FPGA performance versus cell granularity, Proceedings of the IEEE Custom Integrated Circuits Conference, 12-15 May 1991, pp. 6.2/1 -6.2/4

[KoulG92] Kouloheris J.L., Gamal A.El, PLA-based FPGA Area Versus Cell C+ Granularity, Proceedings of the IEEE Custom Integrated Circuits Conference, 3-6 May 1992, pp.4.3.1-4.3.4

[KrisT02] Krishnamoorthy S., Tessier R., Technology Mapping Algorithms for Hybrid FPGAs Containing Lookup-Tables and PLAs, IEEE Transactions on CAD of Integrated Circuits and Systems, Vol.22, No.11, 2002, pp.545-559

[LaiP94a] Lai Y., Pedram M., Vrudhula S., EVBDD-Based Algorithms for Integer Linear Programming, Spectral Transformation, and Function Decomposition, IEEE Transactions on Computer-Aided Design, Vol.13, No.8, August 1994, pp. 959-975

[LaiP94b] Lai Y., Pan K., Pedram M., FPGA synthesis using function decomposition, Proceedings of the IEEE International Conference on Computer Design, Cambridge, 1994, pp. 30-35

[LaiP96] Lai Y., Pan K. R., Pedram M., OBDD-Based Function Decomposition: Algorithms and Implementation, IEEE Transactions on Computer-Aided Design, Vol.15, No.8, August 1996, pp. 977-990

[Laso99] Lasocki R., Dekompozycja zależności funkcjonalnych określonych w zbiorach skończonych, rozprawa doktorska, Politechnika Warszawska, Warszawa 1999

[Latt94] Lattice, Data Book, 1994

[LeeW98] Lee K.K., Wong D.F., Using PLAs to design universal logic modules in FPGAs, IEEE International Symposium on Circuits and Systems, Vol. 6, 31 May-3 Jun 1998, pp. 421-425

[Lee76] Lee S. C., Digital Circuits and Logic Design, Prentice-Hall, Inc. Englewood Cliffs, New Jersey 1976

[LeglW95] Legl Ch., Wurth B., Eckl K., An Implicit Algorithm for Support Minimization during Functional Decomposition, ED&TC, Paris, 1995, pp. 412-417,

[LiuW01] Liu Y., Wang K., Hwang T., Liu C., Binary Decision Diagrams Using Minimum Expected Path Length, Date01, Munich, Germany, 13-16 March, 2001, pp.708-712

[LiuW94] Liu B.D., Wei K.Ch., An Efficient Algorithm for Selecting Bipartite Row or Column Folding of Programmable Logic Arrays, IEEE Transactions on Circuits and Systems, Vol.41, No.7, July 1994, pp.494-498

[Łuba01] Łuba T., Synteza układów logicznych, Wyższa Szkoła Informatyki Stosowanej i Zarządzania, wyd. 2, Warszawa, 2001

[Łuba94] Łuba T., Multi-level logic synthesis based on decomposition, Microprocessors and Microsystems, Vol.18, No. 8, October 1994

[Łuba95] Łuba T., Decomposition of multiple-valued functions, Proceedings 25th International Symposium on Multiple-Valued Logic, 23-25 May 1995, pp. 256–261

[Łuba99] Łuba T., Rola i znaczenie syntezy logicznej w projektowaniu układów cyfrowych. Materiały z konferencji: Reprogramowalne Układy Cyfrowe, RUC'99, Szczecin 1999, ss. 75-84

[ŁubaL94] Łuba T., Lasocki R., Decomposition of Multiple-Valued Boolean Functions, Appl. Math. and Comp. Sci., Vol.4, No.1, 1994, pp.125-138

[ŁubaZ02] Łuba T., Zbierzchowski B., Zbysiński P., Układy reprogramowalne dla potrzeb telekomunikacji cyfrowej, Przegląd Telekomunikacyjny nr 5, 2002

[Mach95] AMD, MACHXL software User's Guide, 1995

[MailM93] Mailhot F., De Micheli G., Algorithms for Technology Mapping Based on Binary Decision Diagrams and on Boolean Operations, IEEE Transactions on CAD, Vol.12, No.5, May 1993

[Maje99] Majewski W., Układy logiczne, WNT, wydanie 6, Warszawa 1999

[MajeŁ86] Albicki A., Ignaczak A., Jasiński K., Kostka A., Łuba T., Majewski W., Zbierzchowski B., Cyfrowe układy telekomunikacyjne. Podstawy teoretyczne i zasady syntezy. Praca zbiorowa pod red. W.Majewskiego i T.Łuby, WKŁ, Warszawa 1986

[MajeŁ92] Majewski W., Łuba T., Jasiński K., Zbierzchowski B., Programowalne moduły logiczne w syntezie układów cyfrowych, WKŁ, Warszawa 1992

[MaliH91] Malik A., Harrison D., Brayton R.K., Three-level decomposition with application to PLDs, Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processors, 14-16 October, 1991, pp. 628 –633

[MalvP98] Malvi R., Perkowski M., Jóźwiak L., Exact Graph Coloring for Functional Decomposition: Do we Need it?, Proceedings of 3rd International Workshop on Boolean Problems, Freiberg University of Mining and Technology, Institute of Computer Science, September 17-18, 1998, pp. 1-10

[McCl56] McCluscey E., Minimization of Boolean Functions, The Bell System Technical Journal, Vol.35, November 1956, pp. 1417-1444

[McGeS93] McGeer P., Sanghavi J., Brayton R. K., Sangiovanni-Vincentelli A., ESPRESSO-SIGNATURES: A New Exact Minimizer for Logic Functions, Proceedings of the Design Automation Conference, 1993, pp. 618-621

[Mich94] Micheli G., Synthesis and optimization of digital circuits, McGrew-Hill, Inc., 1994 (tł. Synteza i optymalizacja układów cyfrowych. WNT, Warszawa 1998

[MichL93] Michel P., Lauther U., Duzy P., The Synthesis Approach to Digital System Design, Kluwer Academic Publishers, Boston/London/Dotdrecht, 1993

[MiliH01] A. Milik, Hrynkiewicz E., Reconfigurable Logic Controller Architecture, Programming, Implementation, International Conference on Programmable Devices and Systems, PDS'01, Gliwice, November 22-23, 2001, pp. 171-176

[Mili03] Milik A., Rekonfigurowalny sterownik logiczny, rozprawa doktorska, Politechnika Śląska, Gliwice 2003

[MinaI01] Minato S., Ishihara S., Streaming BDD Manipulation for Large-Scale Combinatorial Problems, Date01, Munich, Germany, 13-16 March, 2001, pp.702-707

[MitrA99] Mitra S., Avra L.J., McCluskey E.J., An output Encoding Problem and a Solution Technique, IEEE Transactions on Computer-Aided Design, Vol.18, No.6, June 1999, pp.761-768

[Mols86] Molski M., Modułowe i mikroprogramowalne układy cyfrowe. WKŁ, Warszawa 1986

[Mowl76] Mowle F. J., A Systematic Approach to Digital Logic Design, Addisin-Wesley, 1976

[Murg93] Murgai R., Logic Synthesis for Field-Programmable Gate Array, PhD Thesis, University of California at Berkeley, 1993

[MurgN90] Murgai R., Nishizaki Y., Shenay N., Brayton R.K., Sangiovanni-Vincentelli A., Logic Synthesis for Programmable Gate Array, Proceedings 27th DAC, June 1990, pp. 620-625

[MurgS91] Murgai R., Shenoy N., Brayton R.K., Sangiovanni-Vincentelli A., Improved Logic Synthesis for Table Look Up Programmable Gate Arrays, Proceedings of the International Conference on Computer-Aided Design, 1991, pp.564-567

[NowiŁ97] Nowicka M., Łuba T., Selvaraj H., Multilevel Decomposition Strategies in Decomposition-Based Algorithms and Tools, International Workshop on Logic and Architecture Synthesis, Grenoble, 1997, pp. 129-136

[OrteL98] Ortega R. B., Lavagno L., Borriello G., Models and Methods for HW/SW Intellectual Property Interfacing, Materiały szkoleniowe szkoły letniej zorganizowanej przez NATO Advance Study Institute na temat "System Level Synthesis", Barga, Włochy, 11-20 sierpnia 1998

[PanP96] Pan K. R., Pedram M., FPGA Synthesis for Minimum Area, Delay and Power, ED&TC, Paris, 1996, p. 603

[PerkB96] Perkowski M., Burns M., Almeria R., Iliev N., Approaches to the Input-Output Encoding Problem in Boolean Decomposition,' Portland State University, Electrical Engineering Department, Report, January 9, 1996.

[PerkM97] Perkowski M., Marek-Sadowska M., Jóźwiak L., Łuba T., Grygiel S., Nowicka M., Malvi R., Wang Z., Zhang J.S., Decomposition of Multiple-Valued Relations, Proc. ISMVL'97, Halifax, Nova Scotia, Canada, May 1997, pp. 13-18

[PerkM99] Perkowski M., Malvi R., Grygiel S., Burns M., Mishchenko A., Graph coloring Algorithms for Fast Evaluation of Curtis Decomposition, The 36-th ACM/IEEE DAC'99, New Orleans, USA, June 22-25, 1999

[Phil85] Philips, Integrated Fuse Logic, Handbook, 1985

[Piec72a] Piecha J., Dekompozycyjna metoda minimalizacji sieci logicznych, rozprawa doktorska, Politechnika Śląska, Gliwice 1972

[Piec72b] Piecha J., Algorytm dekompozycji pewnej klasy funkcji przełączających. Zeszyty Naukowe Politechniki Śląskiej, s. Automatyka, z. 21, Gliwice 1972

[Piec74] Piecha J., Test prostej alternatywnej dekompozycji funkcji logicznych, Archiwum Automatyki i Telemechaniki, Tom XIX, Zeszyt 2, 1974

[ProuN95] Proudfoot J.T., Ngwira S.M., Efficient method for the identification of optimum disjunctive decompositions of complex logic functions, IEE Proceedings-Computers and Digital Techniques, Vol. 142, No.4, Jul 1995, pp. 249 –254

[Puch00] Pucher K., Efektywne wykorzystanie cyfrowych struktur programowalnych, rozprawa doktorska, Gliwice 2000

[Puch96a] Pucher K., The use of an XOR element in PLD resources, XIX National Conference Circuit Theory and Electronic Networks, Krynica, 23-26 październik 1996

[Puch96b] Pucher K., Improved term partitioning efficiency in the implementation of a single function in PAL structures, International Conference Programmable Devices and Systems, Ostrawa, Czechy (PDS'96), 1996, pp.69-75

[Puch98] Pucher K., Effective use of digital complementary line in PLD circuits, International Conference on Programmable Devices and Systems, PDS'98, Gliwice, February 1998, pp.149-158

[PuchH94] Pucher K., Hrynkiewicz E., About utilization of the PLD resources, XVII-th National Conference Circuit Theory and Electronic Networks Wrocław-Polanica Zdrój, Poland, October 19-21, 1994

[Quin52] Quine W., The problem of Simplifying truth functions, American Mathematical Monthly, Vol.59, 1952, pp. 521-531

[RawsJ01] Rawski M., Jachna Z., Chmielewski J., Praktyczne znaczenie dekompozycji funkcjonalnej w projektowaniu układów cyfrowych, RUC'2001, Szczecin, 7-8 maja, 2001, ss. 29-38

[RawsJ99a] Rawski M., Jóźwiak L., Łuba T., The Influence of the Number of Values in Sub-Functions on the Effectiveness and Efficiency of the Functional Decomposition, Proceedings of the 25th EUROMICRO Conference, IEEE Computer Society, Milan, Italy, September 8-10, 1999, pp. 86-93

[RawsJ99b] Rawski M., Jóźwiak L., Łuba T., Efficient Input Support Selection for Sub-Functions in Functional Decomposition based on Information Relationship Measures, Proceedings of the 25th EUROMICRO Conference, IEEE Computer Society, Milan, Italy, September 8-10, 1999, pp. 94-101

[RawsN96] Rawski M., Nowicka M., Tomaszewicz P., Łuba T., Decomposition-based logic Synthesis and its application in FPGA-oriented technology mapping, International Conference on Programmable Devices and Systems, PDS'96, Ostrava, Czech Republic, November 26-28, 1996, pp.47-54

[RothK62] Roth P.J., Karp R.M., Minimization over Boolean Graphs, IBM Journal of Research and Development, Vol.6, 1962, pp. 227-238

[RudeS87] Rudell R., Sangiovanni-Vincentelli A., Multiple-valued Minimization for PLA Optimization, IEEE Transactions on CAD/ICAS, Vol.CAD-6, No. 5, September 1987, pp.727-750

[RzecJ99] Rzechowski R., Jóźwiak L., Łuba T., Technology Driven Multilevel Logic Synthesis based on Functional Decomposition into Gates, Proceedings of the 25th EUROMICRO Conference, IEEE Computer Society, Milan, Italy, September 8-10, 1999, pp. 368-375

[SalcS98] Salcic Z., Smailagic A., Digital systems design and prototyping using field programmable logic, Kluwer Academic Publishers, 1998

[SaldK88] Saldanha A., Katz R.H., PLA optimization using output encoding, IEEE International Conference on Computer-Aided Design, ICCAD-88, 7-10 November 1988, pp. 478–481

[SaraH93] Sarabi A., Ho P. F., Iravani K., Daasch W. R., Perkowski M. A., Minimal Multi-Level Realization of Switching Functions Based on Kronecker Functional Decision Diagrams, Proceedings of IEEE International Workshop on Logic Synthesis, IWLS '93, Tahoe City, CA, May 1993, pp. 364-369

[Sasa01] Sasao T., Compact SOP representations for multiple-output functions-an encoding method using multiple-valued logic, Proceedings 31st IEEE International Symposium on Multiple-Valued Logic, 2001, pp.207 -212

[Sasa88] Sasao T., T, Multiple-valued logic and optimization of programmable logic arrays, Computer, Vol. 21 No.4, April 1988, pp. 71 -80

[Sasa89a] Sasao T., On the Optimal Design of Multiple-Valued PLA's, IEEE Transactions on Computers, Vol. 38, No. 4, April 1989, pp.582-592

[Sasa89b] Sasao T., Application of multiple-valued logic to a serial decomposition of PLAs, Proceedings Nineteenth International Symposium on Multiple-Valued Logic, 29-31 May 1989, pp. 264 –271

[Sasa91] Sasao T., Bounds on the Average Number of Products in the Minimum Sum-of-Products Expressions for Multiple-Valued Input Two-Valued Output Functions, IEEE Transactions on Computers, Vol. 40, No. 5, May 1991, pp.645-651

[Sasa93a] Sasao T., Logic Synthesis with EXOR Gates, Logic Synthesis and Optimization, Kluwer Academic Publishers, Boston/London/Dordrecht, 1993

[Sasa93b] Sasao T., FPGA Design by Generalized Functional Decomposition in Logic Synthesis and Optimization, Kluwer Academic Publishers, Boston/London/Dotdrecht, 1993

[SasaB90] Sasao T., Besslich P., On the complexity of MOD-2 sum PLA's, IEEE Transaction on Computers, Vol.32, No.2, 1990, pp.262-266

[SasaP93] Sasao T., Perkowski M., EXOR Logic Synthesis, Kluwer Academic Publisher, Boston/London/Dotdrecht, 1993

[SaucS90a] Saucier G., Sicard P., Bouchet L., Multi-level synthesis on PAL's, Proc. European Design Automation Conference, Glasgow, March 1990, pp.542-546

[SaucS90b] Saucier G., Sicard P., Bouchet L., Multi-level synthesis on programmable devices in the ASYL system, Euro ASIC '90, 29 May-1 Jun 1990, pp. 136-141

[SawaY98] Sawada H., Yamashita S., Nagoya A., Restructuring Logic Representations with Easily Detectable Simple Disjunctive Decompositions, Date98, February 23-26, Paris, 1998, pp. 755-761

[SchaP93] Schafer I., Perkowski M.A., Synthesis of Multilevel Multiplexer Circuits for Incompletely Specified Multioutput Boolean Functions with Mapping to Multiplexer Based FPGA's, IEEE Transactions on Computer-Aided Design, Vol.12, No. 11, November 1993, pp.1655-1663

[Scho98] Scholl Ch., Multi-output Functional Decomposition with Exploitation of Don't Cares, Date98, February 23-26, Paris, 1998, pp.743-748

[SelvŁ97] Selveraj H., Łuba T., Nowicka M., Bignall B., Multiple-valued decomposition and its applications in data compression and technology mapping, Proceedings of ICCIMA'97, Gold Coast, Australia, 1997, pp.42-48

[SentS92] ESentovich. M., Singh K.J., Lavagno L., Moon Ch., Murgai R., Saldanha A., Savoj H., Stephan P. R., Brayton R. K., Sangiovanni-Vincentelli A., SIS: A System for Sequential Circuit Synthesis, Electronics Research Laboratory, Memorandum No. UCB/ERL M92/41, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720, 4 May 1992

[Shar98] Sharma K., Programmable Logic Handbook, PLDs, CPLDs, & FPGAs, McGraw-Hill, 1998

[ShenM70] Shen V.Y., McKellar A.C., An Algorithm for the Disjunctive Decomposition of Switching Functions, IEEE Transactions on Computers, Vol. c-19, No.3, 1970

[ShenM71] Shen V.Y-S., McKellar A.C., Weiner P., A Fast Algorithm for the Disjunctive Decomposition of Switching Functions, IEEE Transactions on Computer, Vol. C-20, No.3, 1971

[ShiB93] Shi Ch.J., Brzozowski J.A., An Efficient Algorithm for Constrained Encoding and its Applications, IEEE Transactions on Computer-Aided Design, Vol. 12, No.12, December 1993, pp.1813-1826

[SicaC91] Sicard P., Crates M., Sakouti K., Saucier G., Automatic Synthesis of Boolean Functions on Xilinx and Actel Programmable Devices, Euro ASIC'91, 1991, pp.142-145

[Sign92] Signetics Philips, Programmable Logic Devices, 1992

[Skah96] Skahill K., VHDL for programmable logic, Addison-Wesley Publ. 1996 (tł. "Język VHDL. Projektowanie programowalnych układów logicznych". WNT, Warszawa 2001

[Snap91] Signetics, SNAP. Software for Programmable Logic Devices, 1991

[SongP98] Song N., Perkowski M., A New Approach to AND/OR/EXOR Factorization for Regular Arrays, Proc. 1998 Euromicro, Vasteras, Sweden, August 25-27, 1998, pp. 269-276

[StanS94] Stanion T., Sechen C., Boolean Division and Factorization Using Binary Decision Diagrams, IEEE Transactions on CAD, Vol.13, No.9, September 1994, pp. 1179-1184

[SteiL99] Steinbach B., Lang Ch., Perkowski M., Bi-Decomposition of Discrete Function Sets, Proceedings of 4th International Workshop on Applications of the Reed-Müller Expansion in Circuit Design (Reed-Müller 99), University of Victoria, Victoria B.C., Canada, August 20-21, 1999, pp. 233-251

[Synp00a] Synplify, Synplicity Synthesis Reference Manual, 2000

[Synp00b] Synplify, User Guide and Tutorial, 2000

[ThorD01] Thornton M., Drechsler R., Spectral Decision Diagrams Using Graph Transformations, Date01, Munich, Germany, 13-16 March, 2001, pp.713-719

[Trac86] Traczyk W., Układy cyfrowe. Podstawy teoretyczne i metody syntezy, WNT, Warszawa, 1986

[Volf97] Volf F., A bottom-up approach to multiple-level synthesis for look-up table based FPGAs, CIP-Data Library, Eindhoven 1997

[Wagn78] Wagner F., Projektowanie urządzeń cyfrowych. WNT, Warszawa 1978

[WangA02] Wang L., Almaini A.E.A., Optimisation of Reed-Muller PLA implementations Circuits, IEE Proceedings-Devices and Systems, Vol. 149, April 2002, pp.119-128

[WanP92] Wan W., Perkowski M.A., A new Approach to the Decomposition of Incompletely Specified Multi-Output Functions Based on Graph Coloring and Local Transformations and Its Applications to FPGA Mapping, Proc. of EDAC'92, 1992, pp. 230-235

[WeyC90] Wey Ch.L., Chang T.Y., An Efficient Output Phase Assignment for PLA Minimalization, IEEE Transactions on Computer-Aided Design, Vol. 9, No.1, January 1990

[Woo91a] Woo N-S., A Heuristic Method for FPGA Technology Mapping Based on Edge Visibility, Proceedings of 28th Design Automation Conference, June 1991, pp.248-251

[Xili99] Xilinx, The Programmable Logic Data Book, 1999

[Yan01a] Yan K., Logic synthesis for CPLDs and FPGAs with PLA-style logic blocks, Fourteenth International Conference on VLSI Design, 2001, pp. 291 –297

[Yan01b] Yan K., Practical logic synthesis for CPLDs and FPGAs with PLA-style logic blocks, Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC 2001, 2001, pp.231 –234

[YangC02] Yang C., Ciesielski M., BDS: A BBD-Based Logic Optimization System, IEEE Transactions on CAD of Integrated circuits and systems, Vol.21, No.7, July 2002, pp. 866-876

[YangC89] Yang C., Ciesielski M.J., PLA decomposition with generalized decoders, IEEE International Conference on Computer-Aided Design, ICCAD-89, 5-9 Nov 1989, pp. 312 –315

[YangC91] Yang S., Ciesielski M., Optimum and Suboptimum Algorithms for Input Encoding and its Relationship to Logic Minimization, IEEE Trans. on CAD, Vol.1, January 1991, pp.4-12

[YangS99] Yang C., Singhal V., Ciesielski M., BDD Decomposition for Efficient Logic Synthesis, International Conference on Computer Design, ICCD'99, 1999, pp.626-631

Synteza logiczna przeznaczona dla matrycowych struktur programowalnych typu PAL

Streszczenie

W pracy przedstawiono metody syntezy przeznaczone dla matrycowych struktur programowalnych. Jądrem układów matrycowych (CPLD) jest struktura typu PAL, zawierająca programowalną matrycę AND i stałe połączenia w matrycy OR, tworząca bloki logiczne typu PAL. Celem pracy jest zaprezentowanie metod syntezy, które umożliwiają realizację funkcji za pomocą bloków logicznych typu PAL, zawierających określoną liczbę iloczynów.

Podstawę pierwszej metody stanowi oryginalny sposób przedstawienia zminimalizowanej postaci wielowyjściowej funkcji. Istota tej metody polega na wyszukiwaniu wspólnych wielowyjściowych implikantów na podstawie analizy wierzchołków grafu wyjść, reprezentujących wektory wyjściowe. Zaproponowane algorytmy syntezy osadzone są na twierdzeniu o wyborze wierzchołka grafu wyjść. Twierdzenie to stanowi podstawę teoretyczną doboru grup implikantów realizowanych we wspólnych blokach typu PAL. W przypadku niespełnienia przez wierzchołki grafu wyjść warunków twierdzenia dobór grupy implikantów dokonywany jest metodą heurystyczną. W części tej można znaleźć również oryginalną metodę opisu sprzężeń zwrotnych, sprowadzającą się do odpowiedniej modyfikacji wierzchołków grafu wyjść.

Bloki logiczne spotykane w układach matrycowych często zawierają dodatkowe zasoby logiczne. Należą do nich między innymi trójstanowe bufory wyjściowe, których obecność może być uwzględniona w procesie syntezy. Proces syntezy rozpoczyna dwupoziomowa minimalizacja z rozłączaniem implikantów. Następnie wykonywany jest podział poszczególnych grup implikantów. Wynikiem tych dwóch procedur są zbiory implikantów o mocy mniejszej lub równej liczbie iloczynów znajdujących się w bloku logicznym typu PAL.

Niezwykle cennym elementem syntezy jest dekompozycja. Wpływa ona przede wszystkim na minimalizację liczby bloków logicznych. Takie nietypowe zastosowanie teorii klasycznej dekompozycji jest przedmiotem rozważań zawartych w kolejnej części pracy. Opracowane algorytmy dekompozycji oparto na klasycznym modelu dekompozycji funkcjonalnej, wprowadzonym przez Curtisa. Poszczególne etapy dekompozycji opracowane są pod kątem zasobów logicznych struktur CPLD typu PAL. Opracowana metoda dekompozycji wierszowej funkcji jednowyjściowej pozwala na dopasowania uzyskiwanych podukładów do struktury bloku logicznego typu PAL.

Rozważania teoretyczne stanowią podstawę zawartych w pracy algorytmów syntezy. W końcowej części pracy przedstawiono złożone strategie syntezy przeznaczone dla różnych układów CPLD (z/bez wyjściowych buforów trójstanowych). Strategie syntezy opracowane są pod kątem minimalizacji liczby bloków logicznych lub liczby warstw. Elementami tych strategii są przedstawione poprzednio poszczególne metody syntezy. W pracy zawarto również szereg wyników eksperymentów. Przeprowadzono je na popularnych układach testowych dla różnych rodzin układów programowalnych. Opracowane strategie syntezy porównano z akademickimi i komercyjnymi narzędziami syntezy.

The Logic Synthesis for the PAL-based Complex Programmable Logic Devices

Abstract

This work presents the synthesis methods for CPLDs. The core of CPLDs is a PAL-based structure which consists of the programmable AND matrix and fixed connections within the OR matrix. These matrixes form the PAL-based logic blocks. The aim of the work is to present the synthesis method of which enables implementation of a Boolean function by the means of the PAL-based logic blocks containing a definite number of terms.

The first method is based on a unique representation of the multi-output Boolean function. The essence of the method concentrates on the process of searching for the common multi-output implicants based on the analysis of graph's nodes representing the output vectors. The suggested algorithms of synthesis have their foundation in the theorem on choosing a node of the graph outputs. That theorem serves as a theoretical background for the selection of those implicants' groups that are realized by the means of the shared PAL-based logic blocks. In a case of not meeting the conditions of the theorem by the nodes of the graph outputs to choose the specific node, a selection of the implicants' group is carried out according to the heuristic rules. Moreover, in this part of work, a new method for the description of feedback can be found, which leads to the appropriate modification of those nodes constituting the graph of outputs.

The logic blocks that occur within the CPLD structures include frequently the additional logic resources. There are, among other things, such resources as the three-state output buffers, which can be taken into account in the process of logic synthesis. First, the process of synthesis starts with the two-level splitting minimization procedure. Then, a partition of the individual implicants' groups takes place. As a result of the two procedures mentioned above, the initial set of the Boolean function's implicants is divided into subsets with the higher or equal cardinality to the number of terms within the certain PAL-based logic blocks.

Decomposition is an extremely valuable component of synthesis. First of all, it influences directly the number of logical blocks. The unusual application of the classical theory on the decomposition is discussed in the second part of this work. The algorithms developed are derived from the classical model of the functional decomposition that was introduced by Curtis. The individual phases of the decomposition are prepared for the PAL-based CPLD structures. The designed method for the row decomposition of the single-output Boolean function enables the adjustment of the obtained sub-circuits with the certain structure of the PAL-based logic block.

The theoretical considerations serve as a base for the synthesis of algorithms included in this work. The final part of the work presents the complex strategies of synthesis, aimed at the various CPLD circuits (with/without the three-state output buffers). The strategies of synthesis are designed for the optimization following either the number of logical blocks or the number of levels. The individual synthesis methods, presented in the previous chapters, serve as the components of those strategies. The work presents also a number of results obtained from experiments. The experiments were carried out for the commonly used benchmarks, applying various families of programmable devices. The designed strategies were compared with the university-developed and the commercially available tools of the synthesis.

Dodatek A

Twierdzenie o wyborze wierzchołka grafu wyjść

Jeżeli istnieje wierzchołek grafu wyjść (wyróżnik) ${}^{i}\Delta_y$

1. rzędu $\mu({}^{i}\Delta_y) > 2$, dla którego ${}^{i}\Delta_y \geq k$

lub

2. rzędu $\mu({}^{i}\Delta_y) \geq 2$, dla którego w zbiorze reszt $\mathbf{R} = \{r_j^{\mu({}^{i}\Delta_y)};\ j \in< 1, \mu({}^{i}\Delta_y) > \}$

istnieją co najmniej 2 reszty takie, że $0 < r_a^{\mu({}^{i}\Delta_y)} <^{i}\Delta_y < k$ i $0 < r_b^{\mu({}^{i}\Delta_y)} <^{i}\Delta_y < k$

lub

3. rzędu $\mu({}^{i}\Delta_y) = 2$, dla którego ${}^{i}\Delta_y = k$ lub ${}^{i}\Delta_y \geq 2k-1$

lub

4. rzędu $\mu({}^{i}\Delta_y) = 2$, dla którego $k <^{i}\Delta_y < 2k-1$ i w zbiorze reszt $\mathbf{R} = \{r_j^{\mu({}^{i}\Delta_y)};\ j \in< 1, \mu({}^{i}\Delta_y) > \}$ istnieje co najmniej jedna reszta, taka że $0 < r_a^{\mu({}^{i}\Delta_y)} <^{i}\Delta_y - (k-1)$,

to realizacja implikantów odpowiadających takiemu wyróżnikowi prowadzi do minimalizacji liczby *k*-iloczynowych bloków PAL potrzebnych do realizacji wielowyjściowej funkcji.

Dowód:

W najgorszym przypadku, jeżeli wybrany wierzchołek ${}^{i}\Delta_y$ połączony jest z $\mu({}^{i}\Delta_y)$ wierzchołkami niższych rzędów, ponieważ

$$\bigwedge_{1 \leq j \leq \mu({}^{i}\Delta_y)} {}^{i+1}\Delta_j^{\mu({}^{i}\Delta_y)} = \Delta_j^{\mu({}^{i}\Delta_y) -{}^{i}\Delta_y + 1},$$

to warunek minimalizacji (2.3) można przedstawić w następującej postaci:

$$\sum_{j=1}^{\mu({}^{i}\Delta_y)} \left(\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil + 1 \right) - \sum_{j=1}^{\mu({}^{i}\Delta_y)} \left(\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} -{}^{i}\Delta_y + 1 - k}{k-1} \right\rceil + 1 \right) > \left\lceil \frac{{}^{i}\Delta_y - k}{k-1} \right\rceil + 1 \quad \text{(d1)}$$

1°. ${}^{i}\Delta_y \geq k$

Niech ${}^{i}\Delta_y \in < p(k-1)+1,(p+1)(k-1) >$, gdzie $p \in N$.

Ponieważ dla każdego $p \in N$

$$\sum_{j=1}^{\mu({}^{i}\Delta_y)} \left(\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k - {}^{i}\Delta_y + 1}{k-1} \right\rceil \right) \geq$$

$$\geq \sum_{j=1}^{\mu({}^{i}\Delta_y)} \left(\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k - [p(k-1)+1]+1}{k-1} \right\rceil \right) =$$

$$= \sum_{j=1}^{\mu({}^{i}\Delta_y)} \left(\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k - p(k-1)}{k-1} \right\rceil \right) = \mu({}^{i}\Delta_y)p$$

oraz

$$\left\lceil \frac{{}^{i}\Delta_y - k}{k-1} \right\rceil + 1 = \left\lceil \frac{{}^{i}\Delta_y}{k-1} - \frac{k}{k-1} \right\rceil + 1 \leq \left\lceil \frac{(p+1)(k-1)+1}{k-1} - \frac{k}{k-1} \right\rceil + 1 = \left\lceil p+1-\frac{k-1}{k-1} \right\rceil + 1 = p+1,$$

stąd warunek minimalizacji (d1) sprowadza się do nierówności $\mu({}^{i}\Delta_{o_y})p > p+1$, która ze względu na założenie $\mu({}^{i}\Delta_y) > 2$ jest spełniona dla każdej wartości $p \in$N.

2°. Ponieważ $1 < {}^{i}\Delta_y < k$, stąd warunek minimalizacji (d1) można przekształcić

$$\sum_{j=1}^{\mu({}^{i}\Delta_y)} \left(\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil + 1 \right) - \sum_{j=1}^{\mu({}^{i}\Delta_y)} \left(\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - {}^{i}\Delta_y + 1 - k}{k-1} \right\rceil + 1 \right) > 1$$

i przedstawić w postaci:

$$\sum_{j=1}^{\mu({}^{i}\Delta_{o_y})} \left(\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_{o_y})} - k}{k-1} \right\rceil - \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_{o_y})} - {}^{i}\Delta_{o_y} + 1 - k}{k-1} \right\rceil \right) > 1 \qquad \text{(d2)}$$

Wartości $r_j^{\mu({}^{i}\Delta_y)}$ są resztami wyznaczonymi z kongruencji (2.4)

$${}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - 1 \equiv r_j^{\mu({}^{i}\Delta_y)} (\mathrm{mod}(k-1)), \qquad \text{(d3)}$$

gdzie j=1,2,..., $\mu({}^{i}\Delta_y)$.

Uwzględniając zależność (d3)

$${}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k \equiv r_j^{\mu({}^{i}\Delta_y)} (\mathrm{mod}(k-1))$$

istnieje wartość $p \in N$ taka, że

$$\frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} = p + \frac{r_j^{\mu({}^{i}\Delta_y)}}{k-1}, \qquad \text{(d4)}$$

stąd

$$\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil = p+1 \qquad \text{(d5)}$$

Niech ${}^{i}\Delta_y - 1 \equiv r_l (\mathrm{mod}(k-1))$. (d6)

Ponieważ dla wybranej reszty $r_a^{\mu({}^{i}\Delta_y)}$ zachodzą nierówności $0 < r_a^{\mu({}^{i}\Delta_y)} < {}^{i}\Delta_y < k$, stąd

$$r_l \geq r_a^{\mu({}^{i}\Delta_y)} \qquad \text{(d7)}$$

Uwzględniając nierówności $0 < r_a^{\mu({}^{i}\Delta_y)} < {}^{i}\Delta_y < k$ oraz zależności (d4), (d6)

$$\frac{{}^{i}\Delta_a^{\mu({}^{i}\Delta_y)} - k}{k-1} - \frac{{}^{i}\Delta_y - 1}{k-1} = p + \frac{r_a^{\mu({}^{i}\Delta_y)}}{k-1} - \frac{r_l}{k-1},$$

stąd po uwzględnieniu (d7)

$$\left\lceil \frac{{}^{i}\Delta_a^{\mu({}^{i}\Delta_y)} - k}{k-1} - \frac{{}^{i}\Delta_y - 1}{k-1} \right\rceil = p \qquad \text{(d8)}$$

Ponieważ (d5) i (d8), stąd

$$\left\lceil \frac{{}^{i}\Delta_a^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \left\lceil \frac{{}^{i}\Delta_a^{\mu({}^{i}\Delta_y)} - k}{k-1} - \frac{{}^{i}\Delta_y - 1}{k-1} \right\rceil = 1 \qquad \text{(d9)}$$

Podobnie

$$\left\lceil \frac{{}^{i}\Delta_b^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \left\lceil \frac{{}^{i}\Delta_b^{\mu({}^{i}\Delta_y)} - k}{k-1} - \frac{{}^{i}\Delta_y - 1}{k-1} \right\rceil = 1 \qquad \text{(d10)}$$

Zależności (d9), (d10) powodują, że zachodzi warunek minimalizacji (d2), ponieważ

$$\sum_{j=1}^{\mu({}^{i}\Delta_y)} \left(\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - {}^{i}\Delta_y + 1 - k}{k-1} \right\rceil \right) \geq 2 > 1$$

3.°Dla wierzchołka rzędu 2, dla którego ${}^{i}\Delta_y = k$, warunek (d1) jest spełniony, ponieważ

$$\sum_{j=1}^{2} \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \sum_{j=1}^{2} \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k - (k-1)}{k-1} \right\rceil = 2 > 1 = \left\lceil \frac{k-k}{k-1} \right\rceil + 1 = \left\lceil \frac{{}^{i}\Delta_y - k}{k-1} \right\rceil + 1$$

Warunek (d1) spełniony jest również dla wierzchołków, dla których ${}^{i}\Delta_y \geq 2k-1$, ponieważ

$$\sum_{j=1}^{2} \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \sum_{j=1}^{2} \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k - (2k-1) + 1)}{k-1} \right\rceil = 4 > 2 = \left\lceil \frac{2k-1-k}{k-1} \right\rceil + 1 = \left\lceil \frac{{}^{i}\Delta_y - k}{k-1} \right\rceil + 1$$

4. Dla wierzchołków 2 rzędu, dla których $k < {}^{i}\Delta_y < 2k-1$, warunek minimalizacji (d1) można przedstawić w postaci:

$$\sum_{j=1}^{2} \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \sum_{j=1}^{2} \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k - {}^{i}\Delta_y + 1}{k-1} \right\rceil > 2 \qquad \text{(d11)}$$

Tak jak wykazano w punkcie 2 (zależność d4, d5)

$$\frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} = p + \frac{r_j^{\mu({}^{i}\Delta_y)}}{k-1} \qquad \text{(d12)}$$

oraz

$$\left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil = p + 1 \qquad \text{(d13)}$$

Niech r_l będzie liczbą wyznaczoną z kongruencji ${}^{i}\Delta_y - 1 \equiv r_l (\mathrm{mod}(k-1))$.

Z własności kongruencji wynika, że

$${}^{i}\Delta_y - (k-1) - 1 \equiv r_l (\mathrm{mod}(k-1)) \qquad \text{(d14)}$$

Ponieważ $0 < r_a^{\mu({}^{i}\Delta_y)} < {}^{i}\Delta_y - (k-1) < k$, stąd uwzględniając zależność (d14)

$$r_l \geq r_a^{\mu({}^{i}\Delta_y)} \qquad \text{(d15)}$$

Uwzględniając nierówności $0 < r_a^{\mu({}^{i}\Delta_y)} < {}^{i}\Delta_y - (k-1) < k$ oraz zależności (d12), (d14)

$$\frac{{}^{i}\Delta_a^{\mu({}^{i}\Delta_y)} - k}{k-1} - \frac{{}^{i}\Delta_y - 1}{k-1} = p + \frac{r_a^{\mu({}^{i}\Delta_y)}}{k-1} - 1 - \frac{r_l}{k-1},$$

stąd po uwzględnieniu nierówności (d15)

$$\left\lceil \frac{{}^{i}\Delta_a^{\mu({}^{i}\Delta_y)} - k}{k-1} - \frac{{}^{i}\Delta_y - 1}{k-1} \right\rceil = p - 1 \qquad \text{(d16)}$$

Z zależności (d13) i (d16) wynika, że

$$\left\lceil \frac{{}^{i}\Delta_a^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \left\lceil \frac{{}^{i}\Delta_a^{\mu({}^{i}\Delta_y)} - k}{k-1} - \frac{{}^{i}\Delta_y - 1}{k-1} \right\rceil = p + 1 - (p-1) = 2 \qquad \text{(d17)}$$

Uwzględniając zależność (d17) oraz warunek początkowy $k<{}^{i}\Delta_y<2k-1$ okazuje się, że warunek minimalizacji (d11)

$$\sum_{j=1}^{2} \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k}{k-1} \right\rceil - \sum_{j=1}^{2} \left\lceil \frac{{}^{i}\Delta_j^{\mu({}^{i}\Delta_y)} - k - {}^{i}\Delta_y + 1}{k-1} \right\rceil \geq 2+1>2$$

jest spełniony.

cbdo.

Dodatek B

Formaty opisu przykładowych układów testowych

AHDL – format tablicowy

Układ testowy con1; plik wejściowy

```
SUBDESIGN con1
(
      in[6..0]                  : INPUT;
      out[1..0]                 : OUTPUT;
)

BEGIN
      TABLE
      in[6..0]     =>     out[1..0];

B"x1xx1xx" => B"10";
B"1x11xxx" => B"10";
B"x001xxx" => B"10";
B"01xxx1x" => B"10";
B"x0xx0xx" => B"01";
B"1xxx0xx" => B"01";
B"0xxxxx0" => B"01";
B"01xx1xx" => B"01";
B"10x0xxx" => B"01";

      END TABLE;
END;
```

VHDL – format funkcyjny

Układ testowy con1; plik wejściowy

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Entity con1f is
port( i_6,i_5,i_4,i_3,i_2,i_1,i_0  : in std_logic;
      o_1,o_0  : out std_logic);
end;
Architecture rtl of con1f is
begin
o_0 <= (i_5 and i_2 ) or
       (i_6 and i_4 and i_3 ) or
       (not i_5 and not i_4 and i_3 ) or
       (not i_6 and i_5 and i_1 );
o_1 <= (not i_5 and not i_2 ) or
       (i_6 and not i_2 ) or
       (not i_6 and not i_0 ) or
       (not i_6 and i_5 and i_2 ) or
       (i_6 and not i_5 and not i_3 );
end;
```

VHDL – format tablicowy

Układ testowy con1; plik wejściowy

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Entity con1t is
port( i_6,i_5,i_4,i_3,i_2,i_1,i_0  : in std_logic;
    o_1,o_0  : out std_logic);
end;
Architecture rtl of con1t is
signal inp: std_logic_vector(6 downto 0);
signal g: std_logic_vector(1 downto 0);
begin
inp(6)<=i_6;
inp(5)<=i_5;
inp(4)<=i_4;
inp(3)<=i_3;
inp(2)<=i_2;
inp(1)<=i_1;
inp(0)<=i_0;
with inp select
g <= std_logic_vector'("10")when"0000000",
std_logic_vector'("10")when"0000001",
std_logic_vector'("10")when"0000010",
std_logic_vector'("10")when"0000011",
std_logic_vector'("10")when"0000100",
std_logic_vector'("10")when"0000110",
std_logic_vector'("10")when"0010100",
std_logic_vector'("10")when"0010110",
std_logic_vector'("00")when"0000101",
std_logic_vector'("00")when"0000111",
std_logic_vector'("00")when"0010101",
std_logic_vector'("00")when"0010111",
std_logic_vector'("11")when"0001000",
std_logic_vector'("11")when"0001001",
std_logic_vector'("11")when"0001010",
std_logic_vector'("11")when"0001011",
std_logic_vector'("11")when"0001100",
std_logic_vector'("11")when"0001110",
std_logic_vector'("01")when"0001101",
std_logic_vector'("01")when"0001111",
std_logic_vector'("10")when"0010000",
std_logic_vector'("10")when"0010001",
std_logic_vector'("10")when"0010010",
std_logic_vector'("10")when"0010011",
std_logic_vector'("10")when"0011000",
std_logic_vector'("10")when"0011001",
std_logic_vector'("10")when"0011010",
std_logic_vector'("10")when"0011011",
std_logic_vector'("10")when"0011100",
std_logic_vector'("10")when"0011110",
std_logic_vector'("00")when"0011101",
std_logic_vector'("00")when"0011111",
std_logic_vector'("10")when"0100000",
std_logic_vector'("10")when"0101000",
std_logic_vector'("10")when"0110000",
std_logic_vector'("10")when"0111000",
std_logic_vector'("00")when"0100001",
std_logic_vector'("00")when"0101001",
std_logic_vector'("00")when"0110001",
std_logic_vector'("00")when"0111001",
std_logic_vector'("11")when"0100010",
std_logic_vector'("11")when"0100110",
std_logic_vector'("11")when"0101010",
std_logic_vector'("11")when"0101110",
std_logic_vector'("01")when"0100011",
std_logic_vector'("01")when"0101011",
std_logic_vector'("01")when"0110011",
std_logic_vector'("01")when"0111011",
std_logic_vector'("11")when"0100100",
std_logic_vector'("11")when"0100101",
std_logic_vector'("11")when"0101100",
std_logic_vector'("11")when"0101101",
std_logic_vector'("11")when"0100111",
std_logic_vector'("11")when"0101111",
std_logic_vector'("11")when"0110111",
std_logic_vector'("11")when"0111111",
std_logic_vector'("11")when"0110010",
std_logic_vector'("11")when"0110110",
std_logic_vector'("11")when"0111010",
std_logic_vector'("11")when"0111110",
std_logic_vector'("11")when"0110100",
std_logic_vector'("11")when"0110101",
std_logic_vector'("11")when"0111100",
std_logic_vector'("11")when"0111101",
std_logic_vector'("10")when"1000000",
std_logic_vector'("10")when"1000001",
std_logic_vector'("10")when"1000010",
std_logic_vector'("10")when"1000011",
std_logic_vector'("10")when"1000100",
std_logic_vector'("10")when"1000101",
std_logic_vector'("10")when"1000110",
std_logic_vector'("10")when"1000111",
std_logic_vector'("11")when"1001000",
std_logic_vector'("11")when"1001001",
std_logic_vector'("11")when"1001010",
std_logic_vector'("11")when"1001011",
std_logic_vector'("01")when"1001100",
std_logic_vector'("01")when"1001101",
std_logic_vector'("01")when"1001110",
std_logic_vector'("01")when"1001111",
std_logic_vector'("10")when"1010000",
std_logic_vector'("10")when"1010001",
std_logic_vector'("10")when"1010010",
std_logic_vector'("10")when"1010011",
std_logic_vector'("10")when"1010100",
std_logic_vector'("10")when"1010101",
std_logic_vector'("10")when"1010110",
std_logic_vector'("10")when"1010111",
std_logic_vector'("11")when"1011000",
std_logic_vector'("11")when"1011001",
std_logic_vector'("11")when"1011010",
std_logic_vector'("11")when"1011011",
std_logic_vector'("01")when"1011100",
std_logic_vector'("01")when"1011101",
std_logic_vector'("01")when"1011110",
std_logic_vector'("01")when"1011111",
std_logic_vector'("10")when"1100000",
std_logic_vector'("10")when"1100001",
std_logic_vector'("10")when"1100010",
std_logic_vector'("10")when"1100011",
std_logic_vector'("01")when"1100100",
std_logic_vector'("01")when"1100101",
std_logic_vector'("01")when"1100110",
std_logic_vector'("01")when"1100111",
std_logic_vector'("10")when"1101000",
std_logic_vector'("10")when"1101001",
std_logic_vector'("10")when"1101010",
std_logic_vector'("10")when"1101011",
std_logic_vector'("01")when"1101100",
std_logic_vector'("01")when"1101101",
std_logic_vector'("01")when"1101110",
std_logic_vector'("01")when"1101111",
std_logic_vector'("10")when"1110000",
std_logic_vector'("10")when"1110001",
std_logic_vector'("10")when"1110010",
std_logic_vector'("10")when"1110011",
std_logic_vector'("01")when"1110100",
std_logic_vector'("01")when"1110101",
std_logic_vector'("01")when"1110110",
std_logic_vector'("01")when"1110111",
std_logic_vector'("11")when"1111000",
std_logic_vector'("11")when"1111001",
std_logic_vector'("11")when"1111010",
std_logic_vector'("11")when"1111011",
std_logic_vector'("01")when"1111100",
std_logic_vector'("01")when"1111101",
std_logic_vector'("01")when"1111110",
std_logic_vector'("01")when"1111111",
std_logic_vector'("--")when others;
o_1 <= g(1);
o_0 <= g(0);
end rtl;
```


VHDL – format funkcyjny

Układ testowy xor5; plik wyjściowy; wynik syntezy dla 4-iloczynowych bloków typu PAL

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Entity b_1_1 is
port( i_4,i_3,i_2  : in std_logic;
      g_0_0_0  : out std_logic);
end;
Architecture rtl of b_1_1 is
signal g_0_0_0_oe : std_logic;
signal oe : std_logic;
begin
g_0_0_0_oe <= (not i_4 and not i_3 and i_2 ) or
              (not i_4 and i_3 and not i_2 ) or
              (i_4 and not i_3 and not i_2 ) or
              (i_4 and i_3 and i_2 );
oe <= '1';
g_0_0_0 <= ('Z') when oe = '0' else g_0_0_0_oe;
end;

Library ieee;
Use ieee.std_logic_1164.ALL;
Entity b_1_2 is
port( g_0_0_0,i_1,i_0  : in std_logic;
      o_0  : out std_logic);
end;
Architecture rtl of b_1_2 is
begin
o_0 <= (not g_0_0_0 and not i_1 and i_0 ) or
       (not g_0_0_0 and i_1 and not i_0 ) or
       (g_0_0_0 and not i_1 and not i_0 ) or
       (g_0_0_0 and i_1 and i_0 );
end;

Library ieee;
Use ieee.std_logic_1164.ALL;
Entity Xor5F is
port( i_4,i_3,i_2,i_1,i_0  : in std_logic;
      o_0  : out std_logic);
end;

Architecture rtl of Xor5F is
component b_1_1
port( i_4,i_3,i_2  : in std_logic;
      g_0_0_0  : out std_logic);
end component;
component b_1_2
port( g_0_0_0,i_1,i_0  : in std_logic;
      o_0  : out std_logic);
end component;

for U1:b_1_1 Use entity work.b_1_1(rtl);
for U2:b_1_2 Use entity work.b_1_2(rtl);
signal g_0_0_0:std_logic;
begin
U1:b_1_1 port map(i_4,i_3,i_2,g_0_0_0);
U2:b_1_2 port map(g_0_0_0,i_1,i_0,o_0);
end;
```

VHDL – format funkcyjny

Układ testowy xor5; plik wyjściowy; wynik syntezy dla bloków logicznych typu PAL zawierających 4 iloczyny z trójstanowymi buforami wyjściowymi (np. metoda B_JW)

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Entity b_1_xor5 is
port( d, c, b, a, e : in std_logic;
     y_b_1_xor5 : inout std_logic);
end;

Architecture rtl of b_1_xor5 is
signal  y : std_logic;
signal oe : std_logic;
begin
y <= ( not d and not c and b and a and e ) or
    ( not d and not c and not b and a and not e ) or
    ( not d and not c and b and not a and not e ) or
    ( not d and not c and not b and not a and e );
oe <= not d and not c;
y_b_1_xor5 <= ('Z') when oe = '0' else y;
end;
```

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Entity b_2_xor5 is
port( d, c, b, a, e : in std_logic;
     y_b_2_xor5 : inout std_logic);
end;

Architecture rtl of b_2_xor5 is
signal  y : std_logic;
signal oe : std_logic;
begin
y <= ( not d and c and b and a and not e ) or
    ( not d and c and not b and a and e ) or
    ( not d and c and b and not a and e ) or
    ( not d and c and not b and not a and not e );
oe <= not d and c;
y_b_2_xor5 <= ('Z') when oe = '0' else y;
end;
```

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Entity b_3_xor5 is
port( d, c, b, a, e : in std_logic;
     y_b_3_xor5 : inout std_logic);
end;

Architecture rtl of b_3_xor5 is
signal  y : std_logic;
signal oe : std_logic;
begin
y <= ( d and not c and b and a and not e ) or
    ( d and not c and not b and a and e ) or
    ( d and not c and b and not a and e ) or
    ( d and not c and not b and not a and not e );
oe <= d and not c;
y_b_3_xor5 <= ('Z') when oe = '0' else y;
end;
```

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Entity b_4_xor5 is
port( d, c, b, a, e : in std_logic;
     y_b_4_xor5 : inout std_logic);
end;

Architecture rtl of b_4_xor5 is
signal  y : std_logic;
signal oe : std_logic;
begin
y <= ( d and c and b and a and c ) or
    ( d and c and not b and a and not e ) or
    ( d and c and b and not a and not e ) or
    ( d and c and not b and not a and e );
oe <= d and c;
y_b_4_xor5 <= ('Z') when oe = '0' else y;
end;
```

```
Library ieee;
Use ieee.std_logic_1164.ALL;
Entity xor5 is
port( d, c, b, a, e : in std_logic;
     y_b_1_xor5, y_b_2_xor5, y_b_3_xor5, y_b_4_xor5 : inout
std_logic);
end;

Architecture rtl of xor5 is

component b_1_xor5
port( d, c, b, a, e : in std_logic;
     y_b_1_xor5 : inout std_logic);
end component;
component b_2_xor5
port( d, c, b, a, e : in std_logic;
     y_b_2_xor5 : inout std_logic);
end component;
component b_3_xor5
port( d, c, b, a, e : in std_logic;
     y_b_3_xor5 : inout std_logic);
end component;
component b_4_xor5
port( d, c, b, a, e : in std_logic;
     y_b_4_xor5 : inout std_logic);
end component;

for U1:b_1_xor5 Use entity work.b_1_xor5(rtl);
for U2:b_2_xor5 Use entity work.b_2_xor5(rtl);
for U3:b_3_xor5 Use entity work.b_3_xor5(rtl);
for U4:b_4_xor5 Use entity work.b_4_xor5(rtl);

begin
U1:b_1_xor5 port map(d, c, b, a, e , y_b_1_xor5);
U2:b_2_xor5 port map(d, c, b, a, e , y_b_2_xor5);
U3:b_3_xor5 port map(d, c, b, a, e , y_b_3_xor5);
U4:b_4_xor5 port map(d, c, b, a, e , y_b_4_xor5);
end;
```

Wydano za zgodą
Rektora Politechniki Śląskiej

WYDAWNICTWO POLITECHNIKI ŚLĄSKIEJ
ul. Akademicka 5, 44-100 Gliwice, tel. (0-32) 237-13-81
http://loki.polsl.pl/wydawnictwo

Sprzedaż i Marketing
tel. (0-32) 237-18-48
wydawnictwo_mark@polsl.pl

Nakł. 100+50	Ark. wyd. 19	Ark. druk. 13,375	Papier offset. 70x100, 80 g
Oddano do druku 22.04.2004 r.		Podpis. do druku 22.04.2004 r.	Druk ukończ. w maju 2004 r.

Druk wykonano w Zakładzie Graficznym Politechniki Śląskiej w Gliwicach, ul. Kujawska 1
zam. 173/04

Książki Wydawnictwa można nabyć w księgarniach

GLIWICE

- Punkt Sprzedaży – ul. Akademicka 2 (237-17-87)
- „FORMAT" – Akademicka 5 (architektura i budownictwo)
- „LAMBDA" – ul. Akademicka 2 (237-21-40)
- Punkt Sprzedaży – ul. Akademicka 16 (automatyka, elektronika, informatyka)
- „ŻAK" – ul. Kaszubska

DĄBROWA GÓRNICZA

- „ANEKS" – ul. Ludowa 19A/III

ŁÓDŹ

- „POLITECHNIKA 100" – Żeromskiego 116 PŁ.

KATOWICE

- Punkt Sprzedaży – ul. Krasińskiego 8
- Hurtownia „DIK" – ul. Dulęby 7
- Hurtownia „JERZY" – ul. Słoneczna 24 (258-99-58)

TYCHY

- „I Ja Tours" ul. Piłsudskiego 10 (217-00-91 w.130)

ZABRZE

- Punkt Sprzedaży – ul. Roosevelta 26

KRAKÓW

- Techniczna – ul. Podwale 4
- Punkt Sprzedaży WND – AGH, Al. Mickiewicza 30

GDAŃSK

- EKO-BIS – ul. Dyrekcyjna 6 (058) 305-28-53

WARSZAWA

- Studencka – Pl. Politechniki 1
- Techniczna - ul. Kaliskiego 15
- Techniczna – ul. Świętokrzyska 14
- MDM – ul. Piękna 31

BIAŁYSTOK

- Dom Książki (Księgarnia 84) – ul. Dolistowska 3

POZNAŃ

- Księgarnia „POLITECHNIK" – ul. Piotrowo 3
- DOM Książki – Księgarnia Techniczna – ul. Półwiejska 14

BIBLIOTEKA GŁÓWNA
Politechniki Śląskiej
P.4474/04/14

Druk: Drukarnia Gliwice, ul. Zwycięstwa 27, tel. 230749 50