AUTOMATYKA
CZASOPISMO TECHNICZNE
TECHNICAL TRANSACTIONS
AUTOMATIC CONTROL
WYDAWNICTWO POLITECHNIKI KRAKOWSKIEJ
1-AC/2012
ZESZYT 25
ROK 109
ISSUE 25
YEAR 109

ZBIGNIEW KOKOSIŃSKI*, TOMASZ MICHALSKI**

SYNTEZA DWUPOZIOMOWYCH UKŁADÓW KOMBINACYJNYCH W PROGRAMIE PKmin

SYNTHESIS OF 2-LEVEL COMBINATORIAL CIRCUITS WITH PKmin

Streszczenie

W artykule przedstawiono nowy program PKmin stanowiący użyteczne narzędzie do syntezy kombinacyjnych układów cyfrowych. Program został skonstruowany na podstawie wyników wieloletnich badań prowadzonych w Politechnice Krakowskiej i może wspierać syntezę dwupoziomowych układów Semi Custom i Full Custom realizowanych na bramkach logicznych, układów dwupoziomowych opartych na PLA, układów wielopoziomowych a także dekompozycję funkcjonalną funkcji logicznych pod kątem implementacji w układach FPGA. Artykuł zawiera wyniki badań porównawczych algorytmów syntezy implementowanych w programie PKmin oraz porównanie efektywności programów PKmin i Espresso w zakresie syntezy układów dwupoziomowych realizowanych na bramkach i PLA.

Słowa kluczowe: układ kombinacyjny, PLD, minimalizacja funkcji logicznej, algorytm Kapralskiego, algorytm Kazakowa, PKmin, Espresso

Abstract

In this paper a new design tool is presented that is useful in automated synthesis of combinatorial logic. PKmin program is devoted for synthesis of 2-level circuits composed of gates and PLAs, multi-level circuits and a functional decomposition of logical functions for LUT-based logic implementations in FPGA. It has been built on the basis of the research conducted at Cracow University of Technology. In the paper design algorithms implemented in PKmin are mutually compared. Then, an experimental efficiency comparison of gate and PLA-based 2-level synthesis with PKmin and Espresso design tools is reported.

Keywords: combinatorial logic, PLD, logical function minimization, Kapralski algorithm, Kazakov algorithm, PKmin, Espresso

* Dr inż. Zbigniew Kokosiński, Katedra Automatyki i Technik Informacyjnych, Wydział Inżynierii Elektrycznej i Komputerowej, Politechnika Krakowska.
** Mgr inż. Tomasz Michalski, Nokia Siemens Networks, Wrocław.

1. Wstęp

Synteza kombinacyjnych układów cyfrowych jest klasycznym problemem optymalizacji kombinatorycznej, posiadającym wiele wariantów związanych z rozwojem nowych technologii i zmianami technicznych i ekonomicznych kryteriów projektowych, takich jak minimalizacja liczby komponentów układu i/lub połączeń pomiędzy nimi, eliminacja hazardu, optymalizacja powierzchni układu, pobieranej mocy itp. [2, 3, 8, 11, 16].

Obecnie synteza układów kombinacyjnych jest wspierana komputerowo przez komercyjne środowiska projektowe wiodących firm, narzędzia stworzone w ośrodkach akademickich oraz oprogramowanie niezależne. W Politechnice Krakowskiej prace nad algorytmami syntezy i dekompozycji układów kombinacyjnych rozpoczęły się na przełomie lat 70. i 80. XX w. i były kontynuowane z przerwami do czasów współczesnych [5–7, 9, 10, 14]. W 2008 roku powstał pomysł zebrania większości stworzonych algorytmów i narzędzi projektowych w jeden program. W ciągu dwóch lat powstał PKmin [19], narzędzie oparte głównie na oryginalnych wynikach badań prowadzonych w PK i posiadający szerokie możliwości projektowe obejmujące syntezę jedno- i wielowyjściowych dwupoziomowych układów Semi Custom i Full Custom realizowanych na bramkach logicznych, układów dwupoziomowych opartych na PLA, układów wielopoziomowych, a także dekompozycję funkcjonalną układów logicznych pod kątem implementacji w układach FPGA [14]. Aktualne publikacje w tym obszarze obejmują pozycje [3, 4, 11, 15, 18].

W kolejnych rozdziałach przedstawiono podstawowe algorytmy syntezy układów jednowyjściowych ich warianty i uogólnienia dla układów wielowyjściowych, omówiono podstawowe funkcje programu PKmin oraz wyniki badań porównawczych implementowanych w nim metod. Do badań wykorzystano instancje problemów syntezy wygenerowane losowo. Następnie najbardziej efektywne narzędzia syntezy dwupoziomowej dostępne w programie PKmin zostały porównane z programem Espresso [2]. Pracę kończą wnioski z przeprowadzonych testów i zarys dalszych badań porównawczych programu PKmin z innymi dostępnymi narzędziami syntezy.

2. Algorytmy Kazakowa i Kapralskiego

Celem niniejszego rozdziału jest zaprezentowanie podstawowych zagadnień związanych z syntezą układów kombinacyjnych oraz algorytmów Kazakowa [8] i Kapralskiego [5]. W podrozdziale 2.1. przedstawiono niezbędne definicje i twierdzenia. Idee obu algorytmów opierają się na generacji pewnych kombinacji zmiennych wejściowych i sprawdzeniu możliwości utworzenia dla danej kombinacji implikantów, które mogą wejść w skład poszukiwanej postaci kombinacyjnej zależności. Algorytmy szczegółowo omówiono w podrozdziałach 2.2 i 2.3. W podrozdziale 2.4 podano wskaźniki jakości, jakie obowiązują przy minimalizacji układów kombinacyjnych w strukturach PLA oraz zaprezentowano modyfikacje podstawowych algorytmów Kazakowa i Kapralskiego, które nadają się do syntezy w tych strukturach oraz do syntezy dwupoziomowych układów nieprogramowalnych o strukturach bramkowych. W podrozdziale 2.5 omówiono z kolei syntezę układów typu Semi Custom i Full Custom.

2.1. Podstawowe pojęcia

Dane jest odwzorowanie boolowskie $Y = f(X)$, gdzie: X – macierz wejściowa $p \times n$, Y – macierz wyjściowa $p \times m$. Niech y oznacza p-wymiarowy wektor wyjściowy $y \in Y$.

Twierdzenie 1 [6]
Warunkiem koniecznym i wystarczającym na to, aby wektor $y \in Y$ był kombinacyjnie zależny z macierzą X jest, aby w macierzy X każdy wiersz $X(i)$ typu jeden (dla którego $y(i) = 1$) był różny od wszystkich wierszy $X(j)$ typu zero (dla których $y(j) = 0$), $1 \le i, j \le p$, $i \ne j$.

Powyższe twierdzenie stanowi podstawę teoretyczną do opracowania pierwotnych metod Kapralskiego i Kazakowa przeznaczonych do syntezy kombinacyjnych układów jednowyjściowych. Wyznaczona algorytmicznie funkcja logiczna $y = f(X)$ jest jedną z możliwych postaci kombinacyjnej zależności y z X.

Problem istnienia kombinacyjnej zależności macierzy Y z macierzą X można rozstrzygnąć poprzez sprawdzenie, czy każdy z wektorów macierzy Y jest kombinacyjnie zależny z macierzą X. Jest to warunek konieczny i wystarczający.

Twierdzenie 2 [6]
Macierz Y jest kombinacyjnie zależna z macierzą X wtedy i tylko wtedy, gdy dla każdej pary wierszy równych w X odpowiadająca im para wierszy w Y spełnia warunek równości.

Wyznaczony algorytmicznie zbiór funkcji logicznych $Y = f(X)$ jest jedną z postaci kombinacyjnej zależności Y z X.

Definicja 1 [8]
Implikant (implicent) jest to iloczyn (suma) literałów X o takiej własności, że jego wartość logiczna jest równa 1 (0), gdy wartość funkcji również wynosi 1 (0).

Literał to zmienna lub jej negacja. Implikant (implicent) prosty powstaje przez redukcję liczby zmiennych implikantu (implicentu) do takiej postaci minimalnej, która zachowuje wartość funkcji.

Definicja 2 [8]
Kanoniczna normalna dysjunkcyjna (sumo-iloczynowa) postać kombinacyjnej zależności wektora y z macierzą X, to suma iloczynów literałów utworzonych dla wszystkich różnych wierszy typu jeden w macierzy X.

Definicja 3 [8]
Kanoniczna normalna koniunkcyjna (iloczyno-sumacyjna) postać kombinacyjnej zależności wektora y z macierzą X, to iloczyn sum literałów utworzonych dla wszystkich różnych wierszy typu zero w macierzy X.

Przymiotnik „kanoniczna” w powyższych definicjach oznacza, że każda zmienna występuje w implikancie (implicencie) dokładnie jeden raz. Wyraz „normalna” informuje, że każdy implikant (implicent) występuje w danej postaci wyłącznie jeden raz.

Metody Kazakowa i Kapralskiego umożliwiają uzyskanie zminimalizowanej postaci kombinacyjnej zależności w obu postaciach normalnych. Alternatywne procesy syntezy są w stosunku do siebie dualne, dlatego w dalszej części pracy zostanie omówiona wyłącznie minimalizacja do postaci dysjunkcyjnej.

W algorytmach Kazakowa i Kapralskiego można wyodrębnić dwie fazy. Pierwsza z nich polega na systematycznym poszukiwaniu zbioru implikantów pokrywających wszystkie

wiersze typu jeden macierzy X. Jako wynik tej fazy otrzymuje się zwykle rozwiązanie problemu syntezy w postaci redundancyjnej. Druga faza polega na redukcji liczebności zbioru implikantów poprzez rozwiązanie problemu minimalnego pokrycia zbioru wierszy X typu jeden (zero). W programie PKmin zostało zaimplementowanych siedem różnych metod wyznaczania minimalnego pokrycia. Po tym etapie otrzymujemy suboptymalne rozwiązanie w formie nieredundancyjnej. Związane to jest z faktem, że w pierwszej fazie algorytmów nie są generowane wszystkie implikanty, natomiast w fazie drugiej, ze względu na NP-trudność problemu pokrycia zbioru [17, 18] stosuje się zwykle algorytmy przybliżone. Znalezione rozwiązanie suboptymalne minimalizuje jednocześnie liczbę literałów w implikancie i liczbę implikantów, chociaż mogą mieć zastosowanie również inne kryteria optymalizacji.

2.2. Algorytm Kazakowa

Algorytm Kazakowa znajduje dla danej macierzy wejściowej X i wektora wyjściowego y przybliżoną postać kombinacyjnej zależności y z X w formie dysjunkcyjnej. W podpunkcie 2.1 algorytmu należy poszukiwać k-literałowych implikantów dla kolejnych niepokrytych wierszy typu jeden, różnych od wszystkich wierszy typu zero. Znalezione implikanty, po rozwiązaniu problemu minimalnego pokrycia, tworzą dysjunkcyjną postać kombinacyjnej zależności.

Algorytm Kazakowa

1. Podstawić $k = 1$.
2. Budować kolejne k-literałowe wyrazy dla pierwszego w kolejności niepokrytego wiersza typu jeden.
   - 2.1. Sprawdzić czy wygenerowany wyraz implikuje jakiś wiersz typu zero. Jeżeli nie, to utworzyć implikant i zanotować go przy aktualnym wierszu typu jeden, po czym przejść do podpunktu 2.2. W przeciwnym wypadku:
     - 2.1.1. Jeśli rozpatrzono wszystkie k-literałowe wyrazy dla bieżącego wiersza, to inkrementować k ($k \leq n$).
     - 2.1.2. Powrócić do punktu 2.
   - 2.2. Zanotować implikant przy pozostałych wierszach typu jeden, które pochłania i następnie przejść do podpunktu 2.3.
   - 2.3. Ustawić $k = 1$ oraz przejść do punktu 2.
3. Rozwiązać problem minimalnego pokrycia.

W zmodyfikowanym algorytmie Kazakowa również dla wierszy już pokrytych, sprawdza się wyrazy o koszcie mniejszym od kosztu aktualnie pochłaniającego dany wiersz implikanta. W związku z tym, w pierwszej fazie algorytmu otrzymuje się rozszerzony zbiór implikantów, co może mieć pozytywny wpływ na wynik końcowy.

Dla zademonstrowania pierwszej fazy syntezy algorytmem Kazakowa rozważono układ kombinacyjny opisany macierzą X i wektorem y. Specyfikację macierzową układu wraz z wygenerowanymi implikantami przedstawiono na rys. 1. Znak ~ oznacza negację.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	y	**Znalezione implikanty:**
1	0	1	0	0	0	0	1	x1~x6, x1~x2~x4
1	1	1	1	0	1	1	1	x3x7
1	1	1	1	0	0	0	1	x1~x6
1	1	1	1	0	0	1	1	x1~x6, x3x7
1	0	1	0	0	1	1	1	x3x7, x1~x2~x4
0	1	1	0	1	0	0	1	x5
1	0	0	0	0	1	0	1	x1~x2~x4
1	1	0	0	0	1	1	0	
0	1	1	1	0	0	0	0	
1	0	1	1	0	1	0	0	
1	0	0	1	0	1	1	0	
0	1	0	0	0	0	0	0	
0	0	0	0	0	0	1	0	

Rys. 1. Reprezentacja macierzowa układu kombinacyjnego wraz z implikantami wygenerowanymi algorytmem Kazakowa

Fig. 1. Matrix representation of a combinatorial circuit and implicants generated by Kazakov algorithm

Rysunek 1 pochodzi z programu PKmin, w którym w taki właśnie sposób wizualizuje się pierwszy etap syntezy. Obok każdego wiersza typu jeden umieszcza się implikanty, które go pokrywają. Jako pierwszy jest rozpatrywany pierwszy wiersz typu jeden. Brak jest dla niego wyrazów 1-literałowych, które by go pochłaniały, wobec czego występuje potrzeba generowania wyrazów 2-literałowych. Znaleziony implikant ma postać $x_1 \sim x_6$. Implikant należy zanotować również przy pokrywanych wierszach trzecim i czwartym. Postępując analogicznie dla reszty niepokrytych wierszy, zostaje znaleziony zbiór implikantów pokrywających wszystkie wiersze macierzy X.

W algorytmie Kazakowa istnieje zawsze jeden ściśle określony porządek, według którego generowane są kolejne kombinacje podmacierzy. Rzecz dotyczy również wierszy, które sprawdzane są w kolejności od góry do dołu. Aby to zmienić, nie wystarcza nawet wstępne przetasowanie indeksów kolumn i/lub wierszy. W związku z tym zaimplemento-wano w programie PKmin opcję randomizacji, która polega na tym, że rozwiązanie poszukiwane jest losowo w całej przestrzeni zmiennych, a nie w określonym z góry deterministycznym porządku. Wadą tej metody jest powielanie się implikantów, gdyż ta sama kombinacja kolumn może być wylosowana wiele razy. Aby zredukować znacząco czas syntezy zrezygnowano z weryfikacji powtórzeń generowanych kombinacji, które są eliminowane na etapie poszukiwania minimalnego pokrycia. W przypadku wierszy zaimplementowano mechanizm zapamiętywania wierszy uprzednio sprawdzonych.

2.3. Algorytm Kapralskiego

Algorytm Kapralskiego znajduje dla danej macierzy wejściowej X i wektora wyjściowego y przybliżoną postać kombinacyjnej zależności y z X w formie dysjunkcyjnej. W podpunkcie 2 algorytmu należy poszukiwać k-literałowych implikantów dla kolejnych niepokry-

tych wierszy typu jeden, różnych od wszystkich wierszy typu zero. Znalezione implikanty po rozwiązaniu problemu minimalnego pokrycia utworzą dysjunkcyjną postać kombinacyjnej zależności.

Algorytm Kapralskiego

1. Podstawić $k = 1$.
2. Wygenerować kolejno k-kolumnowe podmacierze tablicy X aż do wyczerpania wszystkich $C(n, k)$ kombinacji kolumn.
   2.1. W każdej utworzonej podtablicy sprawdzać, czy istnieją wiersze typu jeden różne od wszystkich wierszy typu zero. Jeżeli tak, to utworzyć nowe implikanty i zanotować przy nich wszystkie wiersze, które pokrywają.
3. Jeżeli nie wszystkie wiersze typu jeden są pokryte, to inkrementować k ($k \leq n$) i powtórzyć krok 2.
4. Rozwiązać problem minimalnego pokrycia.

W programie PKmin zostały zaimplementowane 4 warianty algorytmu Kapralskiego:
- podstawowa – generuje się wszystkie podmacierze k-kolumnowe, nawet w przypadku, gdy wszystkie wiersze zostały już w pewnym momencie pokryte,
- zmodyfikowana I – nie rozpatruje się wszystkich kombinacji podmacierzy, przerywając ich tworzenie w momencie, w którym znaleziono implikant dla ostatniego niepokrytego dotychczas wiersza,
- zmodyfikowana II – nie poszukuje się implikantów dla wierszy już pokrytych,
- zmodyfikowana III – stanowi połączenie funkcjonalności wersji I i II.

W każdej wersji zmodyfikowanej otrzymuje się zbiór implikantów o liczebności nie większej niż w metodzie podstawowej. Dzięki temu należy się spodziewać, że czas wykonywania syntezy w wariantach I–III ulegnie skróceniu, jednak z drugiej strony otrzymane ostatecznie rozwiązanie może cechować się większym kosztem, ponieważ na etapie redukcji implikantów jest ich zwykle mniej niż w przypadku wersji podstawowej.

W metodach zmodyfikowanych, w których nie generuje się do końca wszystkich podmacierzy, a otrzymana postać redundancyjna zależy od kolejności tworzenia kolejnych kombinacji, zaimplementowano dodatkową opcję polegającą na przetasowaniu przed generacją indeksów kolumn, aby zniwelować efekt porządku leksykograficznego generacji.

Na rys. 2 zaprezentowano pierwszy etap minimalizacji metodą Kapralskiego (wariant zmodyfikowany I) dla tego samego układu kombinacyjnego, co w poprzednim podrozdziale. Na początku zostaje wygenerowana podmacierz 1-kolumnowa x_1, dla której nie istnieją wiersze typu jeden różne od wszystkich wierszy typu zero. Dopiero dla podmacierzy x_5 wiersz szósty jest różny od wszystkich wierszy typu zero i pokrywający go implikant x_5 zostaje zapamiętany. Wśród podmacierzy 2-kolumnowych pierwszy implikant zostaje znaleziony w podmacierzy x_1x_6 i jest nim $x_1{\sim}x_6$, który pochłania wiersze: pierwszy, trzeci i czwarty. Po wygenerowaniu ostatniej podmacierzy 2-kolum-nowej niepokryty zostaje wiersz siódmy, w związku z czym inkrementowane jest k. Algorytm zatrzymuje się na podtablicy $x_1x_2x_4$, a implikant $x_1{\sim}x_2{\sim}x_4$ pochłania wiersz siódmy.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	y	**Znalezione implikanty:**
1	0	1	0	0	0	0	1	x1~x6, x3~x4, x1~x2~x4
1	1	1	1	0	1	1	1	x3x7, x1x2x3
1	1	1	1	0	0	0	1	x1~x6, x1x2x3
1	1	1	1	0	0	1	1	x1~x6, x3x7, x1x2x3
1	0	1	0	0	1	1	1	x3~x4, x3x7, x1~x2~x4
0	1	1	0	1	0	0	1	x5, ~x1x5, x2x5, x3~x4, x3x5, ~x4x5, x5~x6, x5~x7
1	0	0	0	0	1	0	1	x1~x2~x4
1	1	0	0	0	1	1	0	
0	1	1	1	0	0	0	0	
1	0	1	1	0	1	0	0	
1	0	0	1	0	1	1	0	
0	1	0	0	0	0	0	0	
0	0	0	0	0	0	1	0	

Rys. 2. Reprezentacja macierzowa układu kombinacyjnego wraz z implikantami wygenerowanymi algorytmem Kapralskiego(wersja zmodyfikowana I)

Fig. 2. Matrix representation of a combinatorial circuit and implicants generated by Kapralski algorithm (the modified version 1)

Podobnie jak w przypadku metody Kazakowa rezultat syntezy zależy również od kolejności przeglądania wyrazów, w związku z czym zaimplementowano również opcję polegającą na przetasowaniu zbioru indeksów kolumn. Podobnie jak w metodzie Kazakowa, również dla algorytmu Kapralskiego można w programie wybrać opcję randomizacji, dzięki czemu rozwiązanie będzie poszukiwane w całej przestrzeni zmiennych.

2.4. Modyfikacje algorytmów dla układów wielowyjściowych

Wersje oryginalne algorytmów Kapralskiego i Kazakowa oraz wprowadzone w poprzednich podrozdziałach ich wersje zmodyfikowane są przystosowane do minimalizacji układów jednowyjściowych oraz wielowyjściowych programowalnych matryc PAL. Syntezę pozostałych układów wielowyjściowych można by co prawda przeprowadzić jako zbiór syntez układów jednowyjściowych, niemniej jednak takie postępowanie w układach bramkowych niekoniecznie prowadziłoby do najlepszych rezultatów końcowych, natomiast w realizacjach na matrycach PLA układ po procesie syntezy mógłby cechować się większym kosztem niż przed przystąpieniem do minimalizacji.

Za kryterium oceny jakości rozwiązania w układach PLA przyjęto stosować wielkość powierzchni płytki krzemu, jaką zajmuje układ kombinacyjny w strukturze scalonej. Powierzchnia ta szacowana jest przybliżonym wzorem:

$$S = (2n + m) \cdot P \tag{1}$$

gdzie:

- n – liczba wejść układu (mnożnik 2 wynika ze stosowania zmiennych prostych i zanegowanych),
- m – liczba wyjść układu,
- P – liczba implikantów.

Jak można zauważyć, podstawowym zadaniem optymalizacji jest redukcja liczby implikantów, gdyż to od niej w największym stopniu zależy powierzchnia zajmowana przez układ. Drugorzędnym celem jest natomiast redukcja liczby literałów i związane z nią zmniejszenie poboru mocy pobieranej przez układ. Podstawowe algorytmy Kazakowa i Kapralskiego mogą doprowadzić do sytuacji, w której liczba implikantów po syntezie przewyższa liczbę implikantów jako pełnych iloczynów zmiennych przed dokonywaniem minimalizacji, w rezultacie powierzchnia układu ulega zwiększeniu. Dlatego należy odpowiednio zmodyfikować obie metody.

W przypadku algorytmu Kazakowa i Kapralskiego w wersji zmodyfikowanej II oraz III realizuje się to przez korektę tych metod o następujący tok postępowania. Niech y_i należy do zbioru kolumn macierzy wyjść Y. Dla każdej kolumny y_i w przypadku odnalezienia wyrazu, który stanowi implikant dla któregoś z wierszy typu jeden należy wykonać następujące czynności:

1. Sprawdzić, czy dany wyraz pochłania wiersze typu jeden dla pozostałych kolumn y_j $(i \neq j)$ macierzy X.
   - 1.1. Jeśli implikant pokrywa każdy z niepokrytych do tej pory wierszy pozostałych kolumn y_j, przejść do podpunktu 1.2. W przeciwnym wypadku pominąć fakt istnienia tego implikanta.
   - 1.2. Zanotować term przy wierszu typu jeden dla każdej implikowanej kolumny.
   - 1.3. Sprawdzić dla każdej z kolumn oddzielnie, czy znaleziony implikant pochłania inne wiersze typu jeden i odnotować istniejące pokrycia.

W przypadku algorytmu Kapralskiego w wersji podstawowej i zmodyfikowanej I, zmianie ulegają podpunkty 1.1 oraz 1.3. Term przy wierszu notuje się tylko i wyłącznie wtedy, gdy implikuje on wszystkie wiersze typu 1 dla pozostałych kolumn, niezależnie od tego, czy wiersze te są już pochłaniane przez jakiś wcześniejszy implikant, czy też nie. Dzięki takiemu postępowaniu może często dochodzić do sytuacji, że pierwotna liczba termów nie zostanie zredukowana, ale istnieje pewność, że ich liczba nie wzrośnie, a tym samym powierzchnia układu. W najgorszym przypadku algorytm zredukuje tylko liczbę literałów, a więc zmniejszy pobór mocy pobieranej przez układ. W rozdziale 5 porównana zostanie efektywność zaproponowanego algorytmu w stosunku do systemu Espresso, który jest standardowym narzędziem w zakresie syntezy matryc PLA.

2.5. Synteza układów kombinacyjnych Semi Custom i Full Custom

Układy kombinacyjne ASIC typu Semi Custom i Full Custom to układy o strukturach bramkowych, w których za kryterium oceny jakości syntezy przyjmuje się liczbę literałów. Szerokość bramek jest w przybliżeniu proporcjonalna do liczby polikrzemowych ścieżek do nich dochodzących, a każda ścieżka symbolizuje literał. Można więc w przybliżeniu powiedzieć, że powierzchnia zajmowana przez cały układ kombinacyjny jest proporcjonalna do liczby literałów. Układy Semi Custom i Full Custom są przeważnie realizowane

w strukturach wielopoziomowych, tym niemniej zanim powstanie sieć wielopoziomowa najpierw należy dokonać syntezy dwupoziomowej i następnie przekształcić uzyskane postaci kombinacyjnej zależności w wielopoziomową sieć bramek. W ten sposób efekt minimalizacji dwupoziomowej przekłada się na jakość optymalizacji wielopoziomowej. Z tego względu dokonano modyfikacji pierwotnych algorytmów, przyjmując założenie, że ważne jest występowanie w zminimalizowanych funkcjach jak największej liczby implikantów wspólnych. W sieci wielopoziomowej różne funkcje odwoływać się będą do tego samego wierzchołka. W tym celu, po każdorazowym wykryciu implikanta dla jakiegoś wiersza typu jeden pewnej funkcji y_i, sprawdza się, czy utworzony właśnie term jest również implikantem dla tego wiersza w przypadku pozostałych funkcji y_j ($i \neq j$). Przy dużej liczbie wspólnych termów istnieją szanse, że w rozwiązaniach syntezy łącznej będą występowały wspólne implikanty.

3. Metody rozwiązania problemu minimalnego pokrycia

Problem minimalnego pokrycia, który pojawia się w drugim etapie algorytmów Kazakowa i Kapralskiego polega na znalezieniu najmniej licznego podzbioru implikantów tworzącego pokrycie mintermów (wierszy typu jeden). W celu wyznaczenia rozwiązania tworzy się tablicę pokryć, której kolumny odpowiadają implikantom, natomiast wiersze symbolizują elementy pokrywane. Znak *x* umieszcza się na przecięciu wiersza z kolumną wtedy, kiedy implikant reprezentujący daną kolumnę pokrywa wiersz. Rozwiązaniem tablicy implikantów jest zbiór kolumn pokrywających łącznie wszystkie elementy pokrywane. Problem minimalnego pokrycia należy do klasy problemów NP-trudnych [18]. Czas obliczeń zależy od rozmiaru tablicy pokryć. Dla funkcji boolowskiej o *n* wejściach i jednym wyjściu może ona w najgorszym przypadku posiadać $3^n/n$ kolumn oraz 2^n wierszy. Dlatego, stosuje się algorytmy heurystyczne o złożoności wielomianowej, które dają rozwiązania przybliżone w zadowalającym czasie [4]. W programie PKmin zastosowano następujące metody wyznaczania minimalnego pokrycia:

- Exact-Cover (algorytm dokładny),
- Algorytm „podstawowy” (przybliżony),
- Algorytm randomizowany (przybliżony),
- Algorytm randomizowany, uwzględniający koszty (przybliżony),
- Algorytm Chvatala,
- Algorytm Bar-Yehudy i Evena,
- Algorytm Bowmana-McVey’a.

Poniżej przedstawimy pokrótce trzy z nich.

3.1. Algorytm podstawowy

Algorytm podstawowy wyznaczania minimalnego pokrycia [6], który zostanie przedstawiony jako pierwszy, jest najprostszym ze wszystkich metod dla tego zadania zaimplementowanych w programie PKmin.

Algorytm podstawowy

1. Podstawić za rozwiązanie zbiór pusty.
2. Dla każdej kolumny tablicy pokryć obliczyć liczbę występujących w niej znaków *x*.
3. Wybrać pierwszą od lewej kolumnę z największą sumą znaków i dodać odpowiadający jej implikant do rozwiązania.
4. Wykreślić w tablicy pokryć kolumnę wybraną w poprzednim punkcie i usunąć wiersze, które pokrywała.
5. Sprawdzić, czy pojawiły się w tablicy kolumny puste, tzn. nie pokrywające żadnego mintermu. Jeśli tak, to należy je wykreślić.
6. Powtarzać kroki 2–5 aż tablica pokryć będzie pusta, co jest równoznaczne z tym, że wszystkie wiersze zostały pokryte.

3.2. Algorytm randomizowany

W przypadku występowania w tablicy pokryć dwóch lub więcej kolumn o największej liczbie znaków, w wersji randomizowanej algorytmu podstawowego dokonywane jest losowanie, który implikant wybrać do rozwiązania. W metodzie podstawowej wybierana jest zawsze pierwsza kolumna od lewej. Algorytm Kapralskiego generuje implikanty do tablicy pokryć w porządku niemalejących kosztów. Na skutek tego, do rozwiązania dołączany jest zawsze najbardziej efektywny implikant, złożony z najmniejszej liczby literałów. Dlatego efektem randomizacji algorytmu pokrycia przy wyborze metody Kapralskiego może być zwiększona liczba literałów w rozwiązaniu. Z tego powodu nie zaleca się stosować tego wariantu algorytmu redukcji implikantów do syntezy układów bramkowych, gdy wskaźnikiem jakości jest liczba literałów. W przypadku algorytmu Kazakowa efekt randomizacji może okazać się korzystny. Z kolei dla matryc PLA, gdzie wskaźnik jakości jest określony wzorem (1), zarówno dla metody Kapralskiego, jak i Kazakowa randomizacja może zmniejszyć wartość funkcji celu.

3.3. Algorytm randomizowany z minimalizacją kosztu implikantów

Implikanty znajdowane kolejno algorytmem Kazakowa nie są nieuporządkowane ze względu na ich rozmiar. W takim wypadku, przed przystąpieniem do wyznaczenia minimalnego pokrycia, można je posortować w kolejności niemalejących kosztów bądź zastosować procedurę, która w przypadku występowania dwóch lub więcej kolumn o największej sumie znaków wybierze losowo jedną z kolumn odpowiadających implikantom o najmniejszym koszcie.

4. Metody redukcji liczby argumentów

Redukcja liczby argumentów opiera się na pojęciu bazy macierzy. Problem minimalnej bazy macierzy jest NP-zupełny [7]. W programie PKmin zastosowano następujące metody redukcji liczby argumentów [4, 7]:

– MINBASE (algorytm dokładny),
– DRMAX (algorytm przybliżony typu bottom-up),
– MINHMAX (algorytm przybliżony typu bottom-up),
– MAXERMIN (algorytm przybliżony typu top-down),
– Algebra podziałów.

Opis algorytmów redukcji argumentów przekracza ramy niniejszego artykułu.

5. Badania eksperymentalne

Program PKmin został udostępniony na dedykowanej stronie WWW [19]. Metody wspomagające syntezę kombinacyjnych układów dwupoziomowych, wchodzące w skład programu PKmin, zostały przebadane eksperymentalnie. Badania obejmowały porównanie wersji podstawowych, zmodyfikowanych i randomizowanych algorytmów Kazakowa i Kapralskiego, a także algorytmów dokładnych i przybliżonych wyznaczania minimalnego pokrycia oraz algorytmów redukcji argumentów. Najlepsze konfiguracje metod programu PKmin wyłonione w badaniach porównano z programem Espresso w zakresie syntezy dwupoziomowych kombinacyjnych układów wielowyjściowych.

5.1. Porównanie algorytmów Kazakowa i Kapralskiego

W celu porównania obu algorytmów przyjęto następujące założenia:
1. Leksykograficzne przeszukiwanie kolumn macierzy X od strony lewej do prawej.
2. Limit liczby generowanych kombinacji podmacierzy ustalono na 100 tys.
3. Do rozwiązywania problemu minimalnego pokrycia zastosowano metodę podstawową.
4. Liczbę losowych instancji problemu testowanych w każdym eksperymencie określono na 100.

Algorytmy przebadano dla różnych zakresów rozmiarów macierzy. Testowano również czas wykonywania fazy minimalizacji. Wynik pomiaru czasu zaokrąglany był do dwóch miejsc po przecinku, a inne wyniki do najbliższej liczby całkowitej.

5.1.1. Zmiana zakresu liczby wierszy przy stałej liczby kolumn

Liczbę kolumn macierzy wejść ustalono na 30. Zmianie podlegała liczba wierszy w tablicy prawdy. Na podstawie uzyskanych wyników (tabela 1) można stwierdzić, że najlepszym algorytmem pod względem minimalizacji powierzchni matryc PLA i liczby literałów w układach nieprogramowalnych jest algorytm podstawowy Kapralskiego.

W przebadanych zakresach danych wejściowych, przy kryterium minimalnej powierzchni układu, w porównaniu z podstawową metodą Kazakowa daje on rezultaty lepsze średnio o 44–46%, natomiast przy minimalizacji liczby literałów wyniki są lepsze średnio o 33–45%. Algorytm Kazakowa wygrywa z algorytmem Kapralskiego pod względem czasu wykony-wania syntezy w przypadku dużych macierzy o 200–500 wierszach. Można wtedy zastoso-wać wariant I metody Kapralskiego, który znacząco redukuje czas metody w wersji podstawowej. Na uwagę zasługuje fakt, że metoda podstawowa Kazakowa i jej wersja zmodyfikowana prowadzą do jakościowo porównywalnych rezultatów podobnie jak algorytmy Kapralskiego warianty II i III. Efektywność czasowa tych ostatnich jest gorsza od algorytmów Kazakowa, co jest widoczne w przypadku macierzy o rozmiarach 500–1000 wierszy.

Tabela 1

Wyniki syntezy w zależności od algorytmu generacji implikantów

Algorytmy	Kazakow podst.	Kazakow zmod.	Kapralski podst.	Kapralski zmod. I	Kapralski zmod. II	Kapralski zmod. III
Zakres wierszy	10–50					
Średni czas syntezy [s]	**0,00**		0,03	**0,00**	**0,00**	
Średnia liczba termów	7		**4**	6	7	
Średnia liczba argumentów	10		**9**	10	10	
Średnia powierzchnia	136		**73**	120	137	
Średnia liczba tranzystorów	21		**14**	19	21	
Średnia liczba literałów	15		**10**	14	15	
Zakres wierszy	50–100					
Średni czas syntezy [s]	**0,00**		0,38	0,03	0,05	0,02
Średnia liczba termów	13		**8**	11	13	
Średnia liczba argumentów	19		**18**	19	20	
Średnia powierzchnia	484		**274**	409	491	
Średnia liczba tranzystorów	50		**32**	44	51	
Średnia liczba literałów	38		**25**	33	38	
Zakres wierszy	500–1000					
Średni czas syntezy [s]	14,10	**14,06**	145,41	77,40	45,70	43,82
Średnia liczba termów	94		**53**	68	95	
Średnia liczba argumentów	**30**		**30**	**30**	**30**	
Średnia powierzchnia	5716		**3222**	4133	5741	
Średnia liczba tranzystorów	627		**361**	456	629	
Średnia liczba literałów	533		**308**	388	535	

Tabela 2

Wyniki syntezy w zależności od algorytmu generacji implikantów

Algorytmy	Kazakow podst.	Kazakow zmod.	Kapralski podst.	Kapralski zmod. I	Kapralski zmod. II	Kapralski zmod. III
Zakres kolumn	10–50					
Średni czas syntezy [s]	**0,00**		0,06	**0,00**	0,01	**0,00**
Średnia liczba termów	10		**6**	8	10	
Średnia liczba argumentów	14		**12**	13	14	
Średnia powierzchnia	251		**134**	212	252	
Średnia liczba tranzystorów	36		**23**	32	36	
Średnia liczba literałów	27		**17**	24	27	
Zakres kolumn	50–100					
Średni czas syntezy [s]	**0,00**		0,56	0,01	0,10	0,01
Średnia liczba termów	8		**5**	7	8	
Średnia liczba argumentów	14		**11**	13	14	
Średnia powierzchnia	213		**100**	183	214	
Średnia liczba tranzystorów	27		**16**	24	27	
Średnia liczba literałów	19		**12**	17	19	
Zakres kolumn	500–5000					
Średni czas syntezy [s]	**0,01**	0,02	0,37	0,07	0,21	0,06
Średnia liczba termów:	7		**4**	6	7	
Średnia liczba argumentów	11		**8**	10	11	
Średnia powierzchnia	139		**63**	120	140	
Średnia liczba tranzystorów	19		**12**	18	19	
Średnia liczba literałów	13		**8**	12	13	

5.1.2. Zmiana zakresu liczby kolumn przy stałej liczbie wierszy

Liczbę wierszy w tablicy prawdy ustalono na 50. Zmianie podlegała liczba kolumn macierzy wejść. Otrzymane wyniki zawarte są w tabeli 2.

W tym przypadku wraz ze wzrostem liczby kolumn macierzy, czas syntezy nie rośnie. Powierzchnia układu oraz liczba literałów ulegają zmniejszeniu, pomimo iż przy większej liczbie kolumn macierzy wejściowej rozwiązania powinny mieć tendencje do większej różnorodności. Z drugiej strony, wraz ze wzrostem przestrzeni zmiennych rośnie szansa na to, że generowane implikanty będą krótsze.

5.2. Porównanie wersji algorytmów podstawowych i randomizowanych

Dla wybranych algorytmów syntezy zakres wierszy i kolumn macierzy wejściowych ustalono na 500–1000. Zmianie podlegała maksymalna liczba sprawdzanych kombinacji. Liczbę testowanych instancji ustalono w każdym eksperymencie na 100. Wyniki zestawione są w tabeli 3.

Tabela 3

Wyniki porównawcze syntezy algorytmami deterministycznymi i randomizowanymi

Algorytmy	Kazakow	Kazakow randomizowany	Kapralski	Kapralski randomizowany	Kapralski zmod. I	Kapralski zmod. I random.
Zakres wierszy i kolumn	500–1000					
Limit liczby kombinacji	100					
Średni czas syntezy [s]	0,22	**0,21**	2,66	1,92	1,79	1,50
Średnia ilość termów	155	123	128	**101**	128	119
Średnia liczba argumentów	70	486	71	376	71	362
Średnia powierzchnia	22 145	12 2597	**18 379**	77 428	18 473	87 766
Średnia liczba tranzystorów	1307	943	1079	**814**	1084	941
Średnia ilość literałów	1152	820	952	**713**	956	823
Limit liczby kombinacji	1000					
Średni czas syntezy	0,76 s	0,73 s	8,89 s	13,15 s	6,11 s	10,45 s
Średnia ilość termów	124	102	93	**74**	95	97
Średnia liczba argumentów	106	416	**90**	339	91	380
Średnia powierzchnia	27 227	87 365	**17 394**	51 920	18 061	76 529
Średnia liczba tranzystorów	957	724	726	**553**	741	713
Średnia liczba literałów	834	623	634	**480**	647	617

cd. tab. 3

Limit liczby kombinacji podmacierzy	10 000					
Średni czas syntezy	**3,86**	4,01	34,16	95,46	22,39	74,58
Średnia liczba termów	105	88	69	**58**	75	82
Średnia liczba argumentów:	107	360	**82**	275	87	343
Średnia powierzchnia:	23 011	65 578	**11 583**	33 406	13 334	57 062
Średnia liczba tranzystorów:	648	510	434	**352**	470	481
Średnia liczba literałów:	752	598	502	**410**	545	562

Otrzymane wyniki wskazują, że randomizacja pozytywnie wpływa na zmniejszenie liczby literałów. Jeśli chodzi o matryce PLA, randomizacja prowadzi do znacznego wzrostu powierzchni układu spowodowanego zbytnią różnorodnością argumentów w rozwiązaniu.

Porównując szybkość wykonywania metod oryginalnych z randomizowanymi, można zauważyć, że rezultaty zależą od wprowadzonego limitu generowanych kombinacji podmacierzy. Dla badanego zakresu prób losowych w przypadku algorytmu Kazakowa redukcja czasu następuje dla 1 tys. kombinacji, natomiast w metodzie Kapralskiego – dla 100 kombinacji. Mimo iż w większości wypadków algorytm randomizowany kończy swoje działanie na wcześniejszej kombinacji niż algorytm podstawowy, to jednak czas sprawdzania jednej kombinacji jest w nim dłuższy, wskutek czego całkowity czas wykonywania algorytmu jest większy.

5.3. Porównanie algorytmów wyznaczania pokrycia

Selekcji implikantów wykonywano dla dwóch zakresów rozmiarów macierzy: dla kolumn i wierszy w liczbie 10–50 oraz 500–1000. Ponieważ w algorytmie Kapralskiego termy ustawiają się w kolejności nierosnących kosztów, natomiast w Kazakowa w sposób nieuporządkowany, zbadano skuteczność metod rozwiązywania redukcji tablicy pokryć odrębnie dla obu algorytmów. Badania wykonywano na 100 próbach losowych. Limit generowanych kombinacji wynosił 100 tys. Wyniki zestawione są w tabelach 4 i 5.

Pewne wątpliwości metodologiczne dotyczą pytania, czy nie należało najpierw przeprowadzić badań metod selekcji implikantów, a dopiero potem wykonać testy porównawcze algorytmów Kapralskiego i Kazakowa, z optymalnymi dla każdego z nich metodami minimalizacji pokryć. W takiej sytuacji dla algorytmu Kazakowa należałoby wybrać metodę Bowmana-McVeya. Jednak różnice pomiędzy metodą podstawową a Bowmana-McVeya dla algorytmu Kazakowa, biorąc pod uwagę wszystkie kryteria, są rzędu kilku procent. Ponadto przy badaniu metod rozwiązywania problemu minimalnego pokrycia porównywano tablice prawdy o zakresie 500–1000 wierszy na 500–1000 kolumn. Macierze wejściowe użyte w testach z podrozdziału 6.1 były dużo mniejsze, maksymalnie rzędu 30 kolumn na 1000 wierszy oraz 50 wierszy na 5000 kolumn, a różnica w wynikach pomiędzy algorytmem

Kazakowa a Kapralskiego była na poziomie 33–46%. Stąd wniosek, że przeprowadzone testy porównawcze pomiędzy algorytmami generacji implikantów, przy takich samych metodach rozwiązywania problemu minimalnego pokrycia, są poprawne.

Tabela 4

Wpływ algorytmu wyznaczania pokrycia na wyniki syntezy algorytmem Kapralskiego

Metody:	Podstawowa	Z losowaniem	Z losow. uwzgl. koszty	Chvatala	Bar-Yehudy i Evena	Bowmana-McVeya	Dokładna
Zakres wierszy i kolumn:	10–50						
Średni czas redukcji [s]	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	1.53
Średnia liczba termów	**3**	**3**	**3**	**3**	6	4	**3**
Średnia liczba argumentów	**5**	**5**	**5**	**5**	8	6	**5**
Średnia powierzchnia	26	28	28	27	91	44	**25**
Średnia liczba tranzystorów	**8**	**8**	**8**	**8**	17	11	**8**
Średnia liczba literałów	**5**	**5**	**5**	**5**	11	7	**5**
Zakres wierszy i kolumn	500–1000						
Średni czas redukcji [s]	**0,87**	0,89	0,89	1,24	5,82	1,80	–
Średnia liczba termów	**72**	**72**	**72**	**72**	153	140	–
Średnia liczba argumentów	**85**	87	88	**85**	141	105	–
Średnia powierzchnia	**12 469**	12 895	13 029	12 591	44 310	29 774	–
Średnia liczba tranzystorów	**528**	546	531	529	1106	1119	–
Średnia liczba literałów	**456**	475	459	457	953	979	–

Tabela 5

Wpływ algorytmu wyznaczania pokrycia na wyniki syntezy algorytmem Kapralskiego

Metody	Podstawowa	Z losowaniem	Z losow. uwzgl. koszty	Chvatala	Bar-Yehudy i Evena	Bowmana-McVeya	Dokładna
Zakres wierszy i kolumn	10–50						
Średni czas redukcji [s]	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
Średnia liczba termów	**4**	**4**	**4**	**4**	**4**	**4**	**4**
Średnia liczba argumentów	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Średnia powierzchnia	**58**	**58**	**58**	**58**	73	**58**	**58**
Średnia liczba tranzystorów	**11**	**11**	**11**	**11**	12	**11**	**11**
Średnia liczba literałów	7	7	7	7	8	7	7
Zakres wierszy i kolumn	500–1000						
Średni czas redukcji [s]	**0,00**	**0,00**	**0,00**	**0,00**	0,01	**0,00**	0,01
Średnia liczba termów	101	100	100	101	118	98	**98**
Średnia liczba argumentów	103	103	103	103	116	102	**101**
Średnia powierzchnia	21 372	21 104	21 097	21 357	27 924	20 487	**20 386**
Średnia liczba tranzystorów	718	713	713	718	839	701	**700**
Średnia liczba literałów	618	613	613	617	721	603	**602**

5.4. Porównanie algorytmów redukcji argumentów

Przeprowadzając syntezę zmodyfikowaną metodą Kapralskiego I, wybierając podstawowy algorytm selekcji implikantów i algebrę podziałów jako narzędzie redukcji argumentów, otrzymuje się dla zadania syntezy z rozdziału 2 następujące rozwiązanie:

$$y = x_3\overline{x}_4 + x_1x_2x_3 + x_1\overline{x}_2\overline{x}_4$$

złożone z 4 argumentów (8 literałów) i 3 termów. Pod względem wskaźnika jakości (1) dla układów PLA, nawet z wykorzystaniem metody dokładnej rozwiązywania problemu minimalnego pokrycia, nie uzyskano tak zadowalających wyników jak w tym przypadku. Wynika stąd, jak wielkie znaczenie dla wyniku syntezy ma redukcja argumentów zwłaszcza, że w rozpatrywanych przypadku tablica oryginalna była niewielkich rozmiarów: 13 × 7. Pamiętając jednak o fakcie, iż usuwając argumenty można nieumyślnie pozbyć się argumentów najbardziej korzystnych, przeprowadzono szersze badania porównawcze syntezy z redukcją zmiennych i bez (tab. 6). Przy okazji porównano algorytmy znajdowania bazy i oparty na rachunku podziałów. Do badań wybrano algorytm Kazakowa w połączeniu z metodą Bowmana-McVeya, limit generowanych kombinacji podmacierzy ustalono na 100 tys. Badania przeprowadzono na 50 losowych tablicach prawdy o zakresach liczby wierszy i kolumn macierzy wejść 10–50. Macierz wyjść w każdym przypadku była jednokolumnowa.

Tabela 6

Wyniki syntezy w zależności od metody redukcji argumentów

Metody redukcji	Brak	MINBASE	MINHMAX	DRMAX	MAXERMIN	Algebra podziałów
Zakres wierszy i kolumn	10–50					
Średni czas redukcji argum. [s]	0,00	3,50	1,23	**0,02**	**0,02**	127,36
Średni czas syntezy [s]	0,00	3,50	1,23	**0,02**	**0,02**	127,36
Średnia liczba termów	**6**	7	**6**	**6**	**6**	**6**
Średnia liczba argumentów	8	6	7	7	7	**5**
Średnia powierzchnia	98	**83**	85	85	87	60
Średnia liczba tranzystorów	18	26	23	24	24	**22**
Średnia liczba literałów	12	20	18	18	18	**16**

5.5. Porównanie algorytmów syntezy układów dwupoziomowych

Jeśli przez zadeklarowany odcinek czasu algorytmy nie zwrócą rozwiązania, wówczas następuje przedwczesne zakończenie procesu redukcji. Dla algorytmu MAXERMIN, który jako jedyny ustala bazę macierzy wejść od góry (top-down), do syntezy jest brana macierz zredukowana, a w pozostałych przypadkach niezredukowana. W niniejszym eksperymencie ustalono maksymalny czas na 3600 s, aby umożliwić porównanie wszystkich metod.

Pod względem wskaźnika jakości jakim jest powierzchnia PLA (wzór (1)), redukcja argumentów korzystnie wpływa na syntezę logiczną. Redukcja argumentów wpływa negatywnie

na liczbę literałów w minimalizacji układów Semi Custom i Full Custom. Co do skuteczności metod usuwania zmiennych, która jest mierzona liczbą argumentów w rozwiązaniu, najlepszą jest redukcja argumentów algebrą podziałów. Na drugim miejscu uplasował się algorytm MINBASE. Pozostałe metody przybliżone dają wyniki porównywalne. Biorąc pod uwagę szybkość procesu redukcji, najlepszymi algorytmami są DRMAX oraz MAXERMIN. Metoda oparta na algebrze podziałów jest najbardziej czasochłonna. Można również zauważyć, że całkowity czas syntezy prawie w całości zdeterminowany jest redukcją argumentów.

Tabela 7

Wyniki porównawcze algorytmu Espresso z algorytmami Kazakowa i Kapralskiego

Algorytm:	Espresso	Kazakow	Kazakow z DRMAX	Kapralski	Kapralski z DRMAX
Zakres wierszy i kolumn	10–50 (20 prób losowych)				
Limit kombinacji podmacierzy	–	100 tys.	100 tys.	100 tys.	100 tys.
Średni czas syntezy [s]	0,32	**0,01**	0,03	1,95	0,19
Średnia ilość termów:	**22**	25	25	24	25
Średnia ilość argumentów:	20	19	**7**	19	**7**
Średnia powierzchnia:	1419	1465	**759**	1453	760
Średnia ilość tranzystorów:	**257**	271	296	276	301
Zakres wierszy i kolumn	200–500 (10 prób losowych)				
Limit liczby kombinacji	–	10 tys.	10 tys.	1 tys.	1 tys.
Średni czas syntezy[s]	1762,01	**10,04**	61,42	100,64	135,65
Średnia liczby termów	**199**	287	290	294	292
Średnia liczby argumentów	252	185	78	170	**73**
Średnia powierzchnia	110 038	120 081	65 700	113 766	**63 525**
Średnia liczby tranzystorów	**3081**	3904	3994	4227	4213

W tym podrozdziale zawarte są wyniki z przeprowadzonych badań porównawczych pomiędzy Espresso a algorytmami Kazakowa i Kapralskiego. Testowano wersje wielowyjścio-

we obu metod nadające się do syntezy matryc PLA. Sugerując się wynikami z podrozdziału 5.3 w zakresie badań skuteczności metod rozwiązywania problemu minimalnego pokrycia, dla algorytmu Kazakowa wybrana została metoda Bowmana-McVeya, natomiast dla algorytmu Kapralskiego – metoda podstawowa. Liczbę wyjść testowanych tablic prawdy losowano z przedziału od 1 do 30. Badano dwa zakresy wymiarów macierzy wejść: o kolumnach i wierszach z przedziału 10–50 oraz 200–500. Ponadto sprawdzano metody z redukcją argumentów (za pomocą algorytmu DRMAX) i bez redukcji. Algorytm DRMAX wybrano dlatego, ponieważ w poprzednim eksperymencie dawał najlepsze wyniki. Maksymalny czas, po którym redukcja zmiennych wejściowych jest automatycznie przerywana, ustawiono na 60 s. Porównywano rezultaty uśrednione minimalizacji dla 20 (dla pierwszego) i 10 (dla drugiego zakresu) prób losowych. Wyniki zestawiono w tabeli 7.

Chociaż algorytm Espresso zapewnia rozwiązania o najmniejszej średniej liczbie termów i tranzystorów, to algorytmy Kazakowa i Kapralskiego uzupełnione o algorytm redukcji argumentów DRMAX zwyciężają pod względem średniej liczby argumentów i powierzchni projektowanego układu. Efektywność czasowa obu tych metod wchodzących w skład programu PKmin jest w porównaniu z programem Espresso bardzo wysoka.

6. Podsumowanie

W artykule przedstawiono wybrane algorytmy implementowane w programie PKmin oraz przebadano właściwości tego programu pod kątem syntezy dwupoziomowych układów logicznych strukturze bramkowej i PLA. Wybrane konfiguracje narzędzi programu PKmin porównano z programem Espresso stanowiącym światowy standard w tej dziedzinie projektowania. Wskazano w jakim zakresie własności programu PKmin są konkurencyjne z tym programem. Ponieważ przedstawione wyniki nie wyczerpują wszystkich możliwości programu PKmin, który można zastosować również do syntezy wielopoziomowej i dekompozycji funkcjonalnej układów kombinacyjnych w najbliższej przyszłości zostaną przedstawione wyniki uzyskane w tym zakresie [1]. Program PKmin zostanie porównany eksperymentalnie z programami: DEMAIN [20], stworzonym w Zakładzie Podstaw Telekomunikacji Politechniki Warszawskiej oraz ZUBR [21], zbudowanym na Politechnice Białostockiej we współpracy z Białoruskim Państwowym Uniwersytetem Informatyki i Radioelektroniki w Mińsku, z wykorzystaniem tych samych problemów testowych.

Literatura

[1] Byrtek M., *Porównanie efektywności programów PKmin i DEMAIN we wspomaganej komputerowo syntezie wielowyjściowych układów kombinacyjnych*, praca magisterska, Politechnika Krakowska, Kraków 2012.

[2] De Micheli G., *Synteza i optymalizacja układów cyfrowych*, WNT, Warszawa, 1998.

[3] Jacobson H.M., Myers C.J., *Efficient algorithms for exact two-level hazard-free logic minimization*, IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 21(11), 2002, 1269-1283.

[4] Kaglial A., Balachandran S., *Set-cover heuristics for two-level logic minimization*, VLSI Design, 2012, 197-202.
[5] Kapralski A., *Przestrzenie boolowskie i ich zastosowanie do syntezy układów przełączających*, Wydawnictwo Politechniki Krakowskiej, Kraków 1979.
[6] Kapralski A., *Zarys teorii układów przełączających i teorii informacji*, Wydawnictwo Politechniki Krakowskiej, Kraków 1985.
[7] Kapralski A., Skarbek W., *Problem of searching minimum base in Boolean tables*, Podstawy Sterowania, 1986, 257-265.
[8] Kerntopft P., Michalski A., *Wybrane zagadnienia syntezy kombinacyjnych układów logicznych*, PWN, Warszawa 1972.
[9] Kokosiński Z., *O efektywności algorytmów syntezy kombinacyjnych układów logicznych*, Prace III Ogólnopolskiej Konferencji Naukowo-Technicznej SEMTRAK, Kraków–Janowice 1986.
[10] Kokosiński Z., *Analysis of the Sarje method for minimization multi-output switching circuits*, Bull. Applied Math., vol. XLIX, 1987, 299-306.
[11] Liu Y.-Y., Wang K.-H., Hwang T.-T., *Crosstalk minimization in logic synthesis for PLAs*, ACM Trans. Design Autom. Electr. Syst., vol. 11(4), 2006, 890-915.
[12] Łuba T., *Synteza układów logicznych*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2005.
[13] Łuba T. (red.), *Synteza układów cyfrowych*, WKiŁ, Warszawa 2003.
[14] Michalski T., *Przybliżone algorytmy syntezy wielowyjściowych kombinacyjnych układów logicznych*, praca magisterska, Politechnika Krakowska, Kraków 2010.
[15] Rushdi A.M., Ba-Rukab O.M., *A purely map procedure for two-level multiple-output logic minimization*, Int. J. Comput. Math., vol. 84(1), 2007, 1-10.
[16] Salauyou V., Klimowicz A., *Synteza logiczna układów cyfrowych w strukturach programowalnych*, Oficyna Wydawnicza Politechniki Białostockiej, Białystok 2010.
[17] Sysło M., Deo N., Kowalik J., *Algorytmy optymalizacji dyskretnej*, Wydawnictwo Naukowe PWN, Warszawa 1995.
[18] Umans C., Villa T., Sangiovanni-Vincentellio A.L., *Complexity of two-level logic minimization*, IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 25(7), 2006, 1230-1246.
[19] Strona programu PKmin (2010): http://www.pkmin.za.pl/
[20] Strona DEMAIN: http://www.zpt.tele.pw.edu.pl/oprogramowanie/demain.html
[21] Strona programu ZUBR (2007): http://aragorn.pb.bialystok.pl/~zubr/