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

MARIUSZ WIŚNIEWSKI, STANISŁAW DENIZIAK*

DEKOMPOZYCJA SYMBOLICZNA WIELOWARTOŚCIOWYCH FUNKCJI IMPLEMENTOWANYCH W TECHNOLOGII FPGA

SYMBOLIC DECOMPOSITION OF MULTI-VALUED FUNCTIONS IMPLEMENTED IN FPGAs

Streszczenie

W artykule przedstawiono metody dekompozycji wielowartościowych funkcji logicznych z przeznaczeniem do implementacji w układach FPGA opartych o komórki LUT. Zaprezentowana metodologia obejmuje algorytmy dekompozycji oraz kodowania funkcji symbolicznych, omówione zostały również zastosowania tej techniki do kodowania stanów oraz logicznej sieci wielowartościowej.

Słowa kluczowe: FPGA, dekompozycja, kodowanie, logika wielowartościowa

Abstract

The paper presets methods used in decomposition of multi-valued logic functions for LUT-based FPGAs. The methodology consists of algorithm of decomposition and coding technique of symbolic functions as well as using of this method for state encoding and decomposition of multi-valued logic network is presented.

Keywords: FPGA, decomposition, encoding, multi-valued logic

* Mgr inż. Mariusz Wiśniewski, dr hab. inż. Stanisław Deniziak, Katedra Informatyki, Wydział Elektrotechniki, Automatyki i Informatyki, Politechnika Świętokrzyska.

Artykuł złożony do Wydawnictwa: 22.06.2010

1. Wstęp

Najpopularniejsze układy FPGA składają się z programowalnych tabel LUT. Z tego powodu istotne jest, aby funkcja binarna, będąca wynikiem syntezy, została przekształcona do zbioru funkcji implementowalnych w jednej komórce LUT. Dlatego dekompozycja funkcji binarnych jest jednym z podstawowych problemów w syntezie układów przeznaczonych do implementacji w układach FPGA opartych na LUTach [1].

Specyfikacja układów, mogąca być również wynikiem syntezy wyższego poziomu [2], często zawiera zmienne przyjmujące więcej niż dwie wartości [3], nazywane zmiennymi wielowartościowymi. Podczas syntezy zmienne wielowartościowe są kodowane zmiennymi binarnymi. Możemy wyróżnić: kodowanie wejść, kodowanie wyjść oraz kodowanie stanów. Kodowanie jest jednym z etapów syntezy układów logicznych i można je wykonać przed dekompozycją [4] lub na dalszym etapie syntezy. W każdym z tych przypadków kodowanie binarne ma wpływ na jakość końcowego wyniku syntezy i powinno zostać wykonane tak, aby uzyskać jak najlepszą implementację [5].

Wpływ kodowania wejść na wyniki syntezy był badany w [3, 6, 7, 8, 9]. Jednakże metody te są dedykowane logice dwupoziomowej i nie są skuteczne w implementacjach komórek LUT. Opracowane zostały również algorytmy wykonujące faktoryzację [10] oraz dekompozycję symboliczną [11], ale w obu przypadkach kodowanie wykonywane jest po dekompozycji i nie uwzględnia możliwości separacji bitów kodu pomiędzy funkcjami będącymi efektem dekompozycji. W [12] wprowadzone zostało pojęcie dekompozycji szeregowej, a metoda była przeznaczona do syntezy z implementacją w układach FPGA i ukazana jako problem kodowania wejść. Podobne podejścia prezentowano w [13, 14]. Metody te były jednak wbudowane w istniejące algorytmy i nie są efektywne dla implementacji w komórkach LUT. Pierwsze metody syntezy funkcji symbolicznych uwzględniające właściwości układów FPGA opartych o komórki LUT wprowadzono w [15, 16]. W [17, 18] przedstawiono idee symbolicznej dekompozycji funkcjonalnej, jak również wykazano, że metody syntezy oparte na proponowanym podejściu znacząco zmniejszają koszt implementacji funkcji w układach FPGA. Algorytmy ukierunkowane na implementację FSM są przedstawione w [19–22].

W pracy przedstawione są metody dekompozycji funkcji wielowartościowych minimalizujące koszt implementacji funkcji w układach FPGA zbudowanych z komórek LUT. Dekompozycja jest wykonywana iteracyjnie. W każdym kroku zmienne wielowartościowe kodowane są w taki sposób, aby uzyskać jak najlepszą dekompozycję funkcji. Proces dekompozycji każdej z tak otrzymanych funkcji przebiega aż do chwili, kiedy funkcja będzie mogła być zaimplementowana w jednej komórce LUT. W następnym rozdziale przedstawiony jest problem dekompozycji funkcji wielowartościowych. Rozdział 3 zawiera opis własności algebry nakryć wykorzystywanych w opracowanej metodzie. W rozdziale 4 przedstawiono algorytmy dekompozycji symbolicznej. Rozdział 5 przedstawia wyniki wykonanych eksperymentów, a rozdział 6 wnioski.

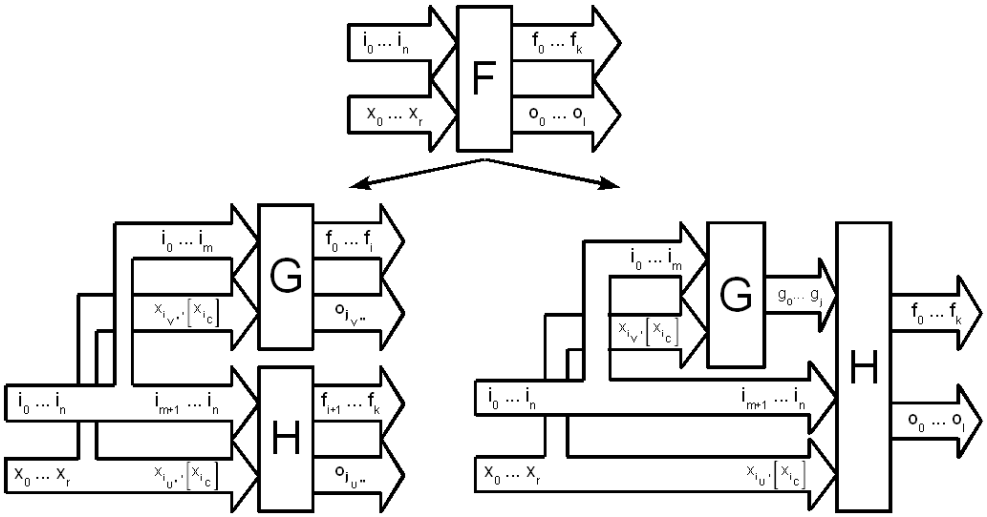
2. Dekompozycja symboliczna wielowartościowych funkcji logicznych

Wielowartościowa funkcja logiczna jest funkcją $F = f(I, X)$ gdzie $F = \{f_0, ..., f_k, o_0, ..., o_l\}$ jest zbiorem funkcji binarnych f_i i/lub wielowartościowych o_j, $I = \{i_0, ..., i_n\}$ jest

zbiorem wejść binarnych oraz $X = \{x_0, ..., x_r\}$ jest zbiorem wejść wielowartościowych. Istotą dekompozycji funkcjonalnej jest przedstawienie zbioru funkcji F w postaci funkcji o mniejszej liczbie wejść. Wyróżnia się dwa typy dekompozycji funkcjonalnej [19]: równoległą i szeregową.

2.1. Dekompozycja równoległa

W trakcie dekompozycji równoległej następuje podział zbioru wejść i/lub wyjść funkcji wejściowej F na dwa zbiory G i H (rys. 1a). W przypadku zakodowania wejść x_i i wyjść o_j przed dekompozycją, możliwa jest separacja zmiennych kodujących. Po zakodowaniu wejść można uzyskać dekompozycję rozłączną lub nierozłączną. W pierwszym przypadku kodowane wejście x_i jest zastępowane przez dwa nowe wejścia $x_{i_{V'}}$ w funkcji G oraz $x_{i_{U'}}$ w funkcji H. W drugim przypadku kodowane wejście x_i jest zastępowane przez trzy nowe wejścia $x_{i_{V'}}$ w funkcji G oraz $x_{i_{U'}}$ w funkcji H, jednocześnie wyznaczane jest kolejne wejście x_{i_C}, będące dodatkowym argumentem dla obu funkcji G i H. Podobnie, wyjścia wielowartościowe o_l są zastępowane przez nowe wyjścia $o_{j_{V''}}$ dla funkcji G oraz $o_{j_{U''}}$ dla funkcji H. Po dekompozycji funkcji F wejścia $x_{i_{V'}}$, $x_{i_{U'}}$ i x_{i_C} oraz wyjścia $o_{j_{V''}}$ i $o_{j_{U''}}$ mogą być kodowane binarnie lub symbolicznie. W drugim przypadku funkcje G i/lub H będą także wielowartościowe, stanowiąc tym samym obiekty dla dekompozycji funkcji wielowartościowej. Dekompozycja równoległa nie powoduje zwiększenia liczby poziomów logicznych w funkcjach wynikowych.



Rys. 1. Dekompozycja równoległa (a) i szeregowa (b) wielowartościowej funkcji F

Fig. 1. Parallel (a) and serial (b) decomposition of multivalued function F

Przedstawiony schemat dekompozycji z kodowaniem wielowartościowych wejść i wyjść pozwala na odpowiednie dobranie kodów dla wartości symbolicznych, aby uzyskane uproszczenie funkcji wejściowej było jak najbardziej korzystne z punktu widzenia

dalszej dekompozycji lub implementacji. Dzięki temu uzyskuje się możliwość optymalizacji funkcji w dużo większym zakresie, niż ma to miejsce w przypadku kodowania wartości symbolicznych, wykonywanego po dekompozycji.

2.2. Dekompozycja szeregowa

Dekompozycja szeregowa przekształca funkcję F na dwie nowe funkcje G i H (rys. 1b), których złożenie reprezentuje funkcję wejściową. Zakodowanie wejść wielowartościowych x_i przed dekompozycją, umożliwia separację zmiennych kodujących, w wyniku której można uzyskać dekompozycję rozłączną lub nierozłączną. W pierwszym przypadku kodowane wejście x_i jest zastępowane przez dwa nowe wejścia x_{i_V} w funkcji G oraz x_{i_U} w funkcji H. W drugim przypadku kodowane wejście x_i jest zastępowane przez trzy nowe wejścia x_{i_V} w funkcji G oraz x_{i_U} w funkcji H, jednocześnie wyznaczane jest kolejne wejście x_{i_C}, będące dodatkowym argumentem dla obu funkcji G i H. Wyjścia funkcji G są jednocześnie wejściami funkcji H, co wynika z własności złożenia funkcji. Natomiast wyjścia funkcji H są identyczne z wyjściami funkcji F, zatem dekompozycja szeregowa nie powoduje separacji wyjść funkcji F, a co za tym idzie kodowanie wyjść nie będzie miało wpływu na jej ostateczny wynik. Po dekompozycji funkcji F wejścia x_{i_V}, x_{i_U} oraz wejście x_{i_C}, jak również wyjścia funkcji G mogą być zakodowane binarnie lub symbolicznie. W drugim przypadku funkcja G i/lub H będą także wielowartościowe, stanowiąc tym samym obiekty dla dekompozycji symbolicznej. Wykonanie dekompozycji szeregowej powoduje zwiększenie liczby poziomów logicznych.

3. Algebra nakryć w odniesieniu do funkcji wielowartościowych

Algebra nakryć jest bardzo wygodnym narzędziem, które można wykorzystać do reprezentacji funkcji logicznych. Nakrycia stanową uogólnienie zbioru podziałów [24] i formalnie do syntezy logicznej zostały wprowadzone w [1]. Nakrycia można także wykorzystać do przedstawiania wielowartościowych funkcji logicznych. Poniższe definicje i twierdzenia stanowią uogólnienie algebry nakryć na tego typu funkcje.

Definicja D3a

Niech funkcja F będzie określona przy pomocy zbioru wartości $S = \{s_1, s_2, ..., s_n\}$. Nakrycie β_F powstaje poprzez podział zbioru sześcianów x_i z tabeli prawdy funkcji F, na podzbiory B_j będące blokami w nakryciu, takimi że: $\beta_F = \{B_{x_i \ni s_1}, B_{x_i \ni s_2}, ..., B_{x_i \ni s_n}\}$. W przypadku wartości „dowolnej” sześcian należy do wszystkich bloków nakrycia β_F.

—

W rachunku nakryć zdefiniowano następujące podstawowe operacje: $\beta * \beta'$ oraz $\beta \leq \beta'$. Symbol $|\beta|$ w odniesieniu do nakrycia oznacza liczbę bloków w nakryciu β.

Definicja D3b [11]

Iloczyn dwóch nakryć $\beta * \beta'$ dany jest następującą zależnością: $\beta * \beta' = ne\{B_i \cap B_j \mid B_i \in \beta, B_j \in \beta'\}$, gdzie $ne\{S_i\} = \{S_i\} - \{\emptyset\}$.

—

Definicja D3c [11]

Jeśli β i β′ są nakryciami tego samego zbioru S, to zachodzi zależność $\beta \leq \beta'$, jeśli dla każdego $B_i \in \beta$ istnieje $B_j \in \beta'$ taki, że $B_i \subseteq B_j$.

—

Dla funkcji wielowartościowych wartości symboliczne podczas syntezy są kodowane sekwencjami binarnymi. Warunkiem poprawności kodowania jest, aby funkcja kodująca była funkcją różnowartościową oraz aby wszystkie wystąpienia tej samej wartości były zakodowane tym samym kodem, mówimy wtedy, że kodowanie jest jednoznaczne.

Definicja D3d

Mówimy, że nakrycie β_A jest pokryciem nakrycia β_B, jeśli każdy blok $A_i \in \beta_A$ jest sumą różnych bloków $B_i \in \beta_B$ oraz każdy blok B_i jest zawarty tylko w jednym bloku A_i.

—

Łatwo zauważyć, że nakrycie każdej zmiennej kodującej musi być pokryciem nakrycia zmiennej kodowanej. Jednak nie jest to warunkiem wystarczającym, aby funkcja kodująca była różnowartościowa.

Twierdzenie T3a

Niech S będzie zmienną wielowartościową oraz niech zmienna S będzie zakodowana zmiennymi A i B (wielowartościowymi lub binarnymi), takimi że nakrycia β_A i β_B są pokryciami nakrycia β_S. Jeśli zależność $\beta_A * \beta_B = \beta_S$ jest prawdziwa, to zakodowanie zmiennej S jest jednoznaczne.

Dowód:

Niech zmienna wielowartościowa S przyjmuje n różnych wartości – $\beta_S = \{S_1 \ldots S_n\}$, niech zmienna A przyjmuje r różnych wartości, wtedy: $\beta_A = \{A_1 \ldots A_r\}$ i niech zmienna B przyjmuje s różnych wartości, wtedy: $\beta_B = \{B_1 \ldots B_s\}$; oraz niech β_A i β_B będą pokryciami β_S, takimi że $\beta_A * \beta_B = \beta_S$. Załóżmy, że dwie różne wartości odpowiadające blokom S_i i S_j mają przyporządkowany ten sam kod odpowiadający blokom (A_k, B_k), wtedy $S_i \cup S_j \subseteq A_k$ i $S_i \cup S_j \subseteq B_k$, zatem $(A_k * B_k) \subseteq S_i \cup S_j$, co oznacza, że $(A_k * B_k) \not\in \subseteq \beta_S$. Czyli założenia twierdzenia są wystarczające, aby kodowanie było różnowartościowe. Załóżmy, że dwa wystąpienia wartości odpowiadającej blokowi S_m występujące w sześcianach x_i i x_j, zostały zakodowane różnymi kodami odpowiadającymi blokom (A_k, B_k) i (A_l, B_l). Wtedy $x_i \in A_k$, $x_i \notin A_l$, $x_j \in A_l$, $x_j \notin A_k$ i $\{x_i, x_j\} \subseteq S_m$. Zatem $S_m \not\subseteq (A_l, A_k)$, czyli β_A nie jest pokryciem β_S. Analogiczną sprzeczność otrzymamy dla β_B. Wynika stąd, że założenia twierdzenia są wystarczające, aby wszystkie wystąpienia tej samej wartości były zakodowane tym samym kodem.

—

Kodując zmiennymi wielowartościowymi, należy wziąć pod uwagę minimalizację długości kodu. Minimalna długość binarnego kodu dla zmiennej symbolicznej S wynosi $[\log_2|\beta_S|]$, z drugiej strony zmienne kodujące muszą spełniać warunek $\beta_A * \beta_B = \beta_S$. Zatem minimalizacja długości kodu będzie polegać na szukaniu nakryć spełniających warunki twierdzenia T3a dla których $[\log_2|\beta_A|] + [\log_2|\beta_B|]$ będzie najmniejsze.

Definicja D3e

Dane jest nakrycie β_X oraz nakryciaβ_A i β_B, będące pokryciami nakryciaβ_X, dla których zachodzi zależność $\beta_X = \beta_A * \beta_B$: N-dopasowanie nakryć β_A i β_B jest to czynność polegająca

na zmniejszaniu lub zwiększaniu liczby bloków w jednym z dopasowywanych nakryć (*A* lub *B*), w taki sposób, aby w drugim nakryciu liczba bloków była równa wartości *N* oraz zależność$\beta_X = \beta_A * \beta_B$ pozostała prawdziwa.

—

Dla binarnej funkcji *F* można wykonać dekompozycję równoległą, której celem jest uzyskanie zbioru prostszych funkcji, spełniających kryteria optymalizacji, np. mających mniej wejść binarnych. Podobnie dla funkcji wielowartościowych można wykonać ten typ dekompozycji, dla której założenia zostały zawarte w twierdzeniu T3b:

Twierdzenie T3b

Dana jest wielowartościowa funkcja *F* będąca zbiorem {*Y*, *O*} funkcji zmiennych ze zbiorów *I* oraz *X*, tj. $F = f(I, X)$. Jeśli istnieją zbiory *V* i *U* będące podzbiorami zbioru *I*, zbiory Q_G i Q_H takie że $\beta_{Q_G} * \beta_{Q_H} = \beta_X$ oraz zbiory *G* i *H*, takie że: $G \cup H = F$ i $G \cap H = \emptyset$ oraz $\beta_V * \beta_{Q_G} \leq \beta_G$ i $\beta_U * \beta_{Q_H} \leq \beta_H$ to istnieje dekompozycja równoległa funkcji *F* wyrażona przez funkcje $F_G = f(V, Q_G)$ i $F_H = f(U, Q_H)$, oraz $\beta_{F_G} = \beta_G$ i $\beta_{F_H} = \beta_H$.

Dowód:

Niech $\beta_V \leq \beta_G$ oraz niech funkcja F_G zależy również od wartości $i_k \notin V$. Wtedy istnieją dwa sześciany różniące się jedynie na pozycji i_k, dla których funkcja F_G przyjmuje różne wartości. Oznacza to, że te sześciany zawarte są w różnych blokach nakrycia β_{F_G}. Jednakże, ze względu na to, że wartości obu sześcianów, odpowiadające wszystkim wejściom ze zbioru *V*, są takie same, to te sześciany także należą do tego samego bloku β_V, z czego wynika, że zależność $\beta_V \leq \beta_{F_G}$ nie jest prawdziwa i funkcja F_G nie zależy od żadnego z wejść spoza zbioru *V*. Analogicznie można wykazać, że funkcja F_H zależy jedynie od wartości wejść ze zbioru *U*.

—

Dekompozycja wykonana zgodnie z twierdzeniem T3b może dotyczyć funkcji wielowartościowych przed zakodowaniem. Wtedy zbiory Q_G i Q_H są podzbiorami zbioru *X*. Możliwe jest również wcześniejsze zakodowanie zmiennymi symbolicznymi, wtedy możliwa jest separacja zmiennych kodujących w zbiorach Q_G i Q_H. Podobnie można postępować z wyjściami wielowartościowymi.

Drugim typem dekompozycji, jaki można zastosować dla funkcji *F*, jest dekompozycja szeregowa, którą również można wykonać dla funkcji wielowartościowych, a twierdzenie określające zasady jej wykonania jest następujące:

Twierdzenie T3c

Jeśli istnieją nakrycia β_G, β_{Q_V} oraz β_{Q_u} takie, że $\beta_{Q_V} * \beta_{Q_U} = \beta_X$ oraz

$$\beta_V * \beta_{Q_V} = \beta_G \tag{1}$$

$$\beta_U * \beta_{Q_U} * \beta_G = \beta_F \tag{2}$$

to istnieje dekompozycja szeregowa funkcji *F* taka, że $F(I, X) = H(U, Q_U, G(V, Q_V))$, gdzie $V \cup U = I$ oraz $Q_V \cup Q_U = X$.

Dowód:

Dowód jest taki jak dla twierdzenia o dekompozycji szeregowej funkcji binarnych z [11].

—

Aby uprościć obliczenia związane z dekompozycją szeregową, wyróżnia się nakrycie β_M, wyznaczane z zależność (3.2) poprzez wyłączenie nakryć β_U i β_F, których wartości nie zmieniają się w trakcie wykonywania kolejnych etapów algorytmu

$$\beta_U * \beta_M = \beta_F \tag{3}$$

gdzie β_M spełnia zależność: $\beta_{Q_U} * \beta_G \leq \beta_M$.

W celu znalezienia dekompozycji funkcji istnieje konieczność konstruowania nowego nakrycia β_C, na podstawie zawartości znanych nakryć β_A i β_B, tak aby zachodziło $\beta_A * \beta_C \leq \beta_B$. Zaobserwowano, że lepsze wyniki dekompozycji uzyskuje się konstruując bloki nakrycia β_C poprzez sumowanie bloków nakrycia ($\beta_A * \beta_B$), a nie poprzez poszukiwanie dowolnego, spełniającego wymagane zależności. Jednocześnie najlepszym β_C jest nakrycie utworzone w podany powyżej sposób, posiadające najmniejszą z możliwych liczbę bloków.

Jakość dekompozycji jest uzależniona od wyboru funkcji G i H. Poniższa definicja pozwala oszacować koszt implementacji funkcji F po dekompozycji.

Definicja D3f

Niech funkcja F posiada n_F wejść i m_F wyjść. Niech komórka LUT, posiada n_L wejść i m_L wyjść. Spodziewany koszt implementacji (SKI) funkcji F jest równy: $SKI(F) = [n_F/n_L] * [m_F/m_L]$. Jeśli wejście lub wyjście jest wielowartościowe, to $n/m = [\log_2 S_{n/m}]$, gdzie S jest liczbą wartości symbolicznych wejścia (n) lub wyjścia (m).

4. Algorytmy dekompozycji symbolicznej

Poniżej zostały przedstawione algorytmy dekompozycji symbolicznej funkcji wielowartościowej F. Dla uproszczenia przyjęto, że funkcja F będzie posiadała jedno lub więcej wejść i/lub wyjść binarnych oraz dokładnie jedno wejście X lub wyjście wielowartościowe O. Przedstawione algorytmy można uogólnić na funkcje o dowolnej liczbie zmiennych wielowartościowych.

4.1. Dekompozycja równoległa z kodowaniem wejść

Dekompozycja z kodowaniem wejść jest stosowana w przypadku, gdy funkcja F posiada co najmniej jedno wejście, które przyjmuje wartości symboliczne i polega na znalezieniu takiego zakodowania każdego wejścia wielowartościowego x_i dwoma zmiennymi Q_{i_V} i Q_{i_U} aby możliwa była dekompozycja funkcji F na funkcje G i H, gdzie zmienne Q_{i_V} są wejściami funkcji G, a zmienne Q_{i_U} wejściami funkcji H, oraz aby funkcje G i H były jak najprostsze.

Dekompozycja równoległa wykonywana jest zgodnie z twierdzeniem T3b. Wynikiem działania algorytmu jest dekompozycja funkcji F na dwie funkcje G i H oraz zakodowanie wejścia wielowartościowego dwoma zmiennymi wielowartościowymi Q_V i Q_U, dla których

spodziewany koszt implementacji $SKI(G) + SKI(H)$ jest najmniejszy. W celu uproszczenia algorytmu przyjęto następujące założenia:

– funkcja F posiada co najmniej jedno wejście binarne $i \in I$,
– funkcja F posiada dokładnie jedno wejście wielowartościowe X,
– wyjścia funkcji F muszą zostać wcześniej podzielone na rozłączne zbiory G i H.

Zbiory G i H wyznaczane są w taki sposób, aby końcowy wynik dekompozycji był jak najlepszy, tj. aby spodziewany koszt implementacji $SKI(G + H)$ był najmniejszy. Po wykonaniu algorytmu otrzymuje się funkcję $G = \{f_0, ..., f_i\}$, wejście wielowartościowe lub binarne Q_V, funkcję $H = \{f_{i+1}, ..., f_k\}$, wejście wielowartościowe lub binarne Q_U oraz wejście wielowartościowe lub binarne Q_C (dla dekompozycji nierozłącznej).

Parametrami algorytmu dekompozycji równoległej są nakrycia binarnych wejść funkcji G i H (zbiory V i U), nakrycie wejścia wielowartościowego X i wyjść binarnych i/lub wielowartościowych funkcji F (podzielone na zbiory G i H). Poniżej został przedstawiony szkic zawierający główne kroki algorytmu:

```
function dr_kodowanie_wejść(β_G, β_V, β_H, β_U, β_X, l_blk_ β_Q_V)
  if β_V == Ø then β_Q_V = β_G; else β_Q_V = gen_min_nakr(β_V, β_X, β_G);
  if H ≠ Ø then {
    if β_U == Ø then β_Q_U = β_H; else β_Q_U = gen_min_nakr(β_U, β_X, β_H);
    if |β_Q_V| < l_blk_β_Q_V then
      N-dopasowanie liczby bloków w β_Q_V z uwzględnieniem nakrycia β_Q_u;
    if β_U == Ø then β_Q_U = gen_min_nakr_Q(β_Q_V, Ø, β_X, β_H);
    else β_Q_U = gen_min_nakr_Q(β_Q_V, β_U, β_X, β_H);
  }
  if |β_Q_V| < l_blk_β_Q_V then N-dopasowanie liczby bloków nakrycia β_Q_V;
  return β_G, β_Q_V, β_H, β_Q_U, U;
```

Podczas dekompozycji najpierw wyznaczane jest nakrycie β_{Q_V} składające się z jak najmniejszej liczby bloków, będące pokryciem nakrycia β_X i spełniające warunek $\beta_V * \beta_{Q_V} \leq \beta_G$ (funkcja *gen_min_nakr*). W podobny sposób wyznaczane jest nakrycie β_{Q_U}. Następnie, po N-dopasowaniu nakrycia β_{Q_V} wyznaczane jest ostateczne nakrycie β_{Q_U} zapewniające poprawne kodowanie zmiennej X zmiennymi Q_V i Q_U (funkcja *gen_min_nakr_Q*). Na zakończenie wykonywane jest ponowne N-dopasowanie β_{Q_V}.

4.2. Dekompozycja równoległa z kodowaniem wyjść

Dekompozycję równoległą z kodowaniem wyjść wykonuje się zgodnie z twierdzeniami T3a i T3b. W wyniku wykonania algorytmu następuje podział funkcji F na dwie funkcje G i H oraz wyznaczane jest zakodowanie wyjścia wielowartościowego dwoma zmiennymi wielowartościowymi O_G i O_H. Dla uproszczenia algorytmu przyjęto, że:

– funkcja F posiada co najmniej jedno wejście binarne $i \in I$ oraz dokładnie jedno wyjście wielowartościowe O, przy czym zachodzi zależność $\beta_I \leq \beta_O$,
– funkcja F nie posiada wyjść binarnych, a jeśli funkcja wejściowa miała jakieś wyjścia binarne, to powinny zostać wyłączone z funkcji F w drodze dekompozycji równoległej.

Podczas dekompozycji poszukiwane są takie zakodowania wyjścia wielowartościowego O, aby wyznaczone funkcje G i H były jak najprostsze, tj. miały jak najmniej wejść binarnych oraz liczba bloków w β_{O_G} była możliwie zbliżona do liczby bloków w β_{O_H}.

W wyniku wykonania algorytmu można otrzymać funkcje G i H wielowartościowe lub binarne, przy czym algorytm preferuje rozwiązania, w których funkcja G jest binarna. Wtedy, po każdym wykonaniu algorytmu, dalszej dekompozycji wymaga jedynie funkcja H. Dla zaproponowanych funkcji G i H algorytm wybiera zbiory wejść V i U o najmniejszej z możliwych liczbie elementów.

Parametrami algorytmu dekompozycji równoległej z kodowaniem wyjść jest zbiór wejść binarnych I oraz nakrycie wyjście wielowartościowego funkcji F. Poniżej został przedstawiony szkic zawierający główne etapy algorytmu:

function *dr_kodowanie_wyjść*(I, β_O, *l_blk_*β_{O_G})
  $\beta_{O_G} = \emptyset$; $V = \emptyset$; $\beta_{O_H} = \emptyset$; $U = \emptyset$; E = *podzbiory zbioru* I;
  for *każdy element* $E_i \in E$ **do** {
    $\beta_{O_{G_{tmp}}}$ = *gen_min_nakr_O*(β_{E_i}, β_O);
  if $\beta_{O_{G_{tmp}}} \neq$ **then** {
    if $\left|\beta_{O_{G_{tmp}}}\right| <$ *l_blk_*β_{O_G} **then** *N-dopasowanie liczby bloków nakrycia* $\beta_{O_{G_{tmp}}}$;
    $V_{tmp} = E_i$; $\beta_{O_{H_{tmp}}}$ = *gen_min_nakr*($\beta_{O_{G_{tmp}}}$, β_O, β_O); *wyznaczenie* U_{tmp};
}
if $U_{tmp} = \emptyset$ **and** $|U_{tmp}|$ **and** $\left|\beta_{O_{H_{tmp}}}\right| \leq \left|\beta_{O_H}\right|$ **then** {
    $\beta_{O_G} = \beta_{O_{G_{tmp}}}$; $V = V_{tmp}$; $\beta_{O_H} = \beta_{O_{H_{tmp}}}$; $U = U_{tmp}$;
} }
return β_{O_G}, V, β_{O_H}, U;

Podczas wykonania algorytmu zostają wyznaczone podzbiory zbioru wejść binarnych funkcji F (umieszczane w zbiorze E), następnie podejmowana jest próba wyznaczenia zakodowania wyjścia wielowartościowego O dla funkcji G. Wyznaczone zakodowanie wyjścia O zostaje umieszczane w $\beta_{O_{G_{tmp}}}$. Następnie, jeśli jest taka możliwość, to nastąpi zwiększenie liczby bloków w $\beta_{O_{G_{tmp}}}$. Wtedy, ze względu na twierdzenie T3a, liczba bloków w nakryciu wyjścia O_H może ulec zmniejszeniu. W dalszej części algorytmu wyznaczane jest tymczasowe nakrycie funkcji O_H (zapamiętywane w $\beta_{O_{H_{tmp}}}$), jednocześnie dla funkcji H wyznaczane są wejścia binarne. Jeśli wybrane w ten sposób tymczasowe rozwiązanie jest lepsze od dotychczasowego rozwiązania, to zostaje ono zapamiętane. Lepsze jest to roz-

wiązanie, w którym liczba bloków w nakryciu wyjścia O_G jest zbliżona do oczekiwanej wartości, a liczba wejść w funkcji H jest najmniejsza z możliwych. W przypadku braku możliwości zakodowania wyjścia O, upraszczającego funkcję F, sprawdzany jest kolejny element zbioru E.

4.3. Dekompozycja szeregowa

Dekompozycja szeregowa wykonywana jest zgodnie z twierdzeniem T3c. Wynikiem działania algorytmu jest dekompozycja funkcji F na dwie funkcje G i H oraz zakodowanie wejścia wielowartościowego dwoma zmiennymi wielowartościowymi Q_V i Q_U. Powyższe elementy są ze sobą ściśle związane i są obliczane jednocześnie. Dla uproszczenia algorytmu przyjęto następujące założenia:

- funkcja F posiada co najmniej jedno wejście binarne $i \in I$,
- funkcja F posiada dokładnie jedno wejście wielowartościowe X,
- wejścia binarne funkcji F muszą zostać wcześniej podzielone na rozłączne zbiory U i V, przy czym zbiór U może być pusty.

Zbiory V i U wyznaczane są w taki sposób, aby końcowy wynik dekompozycji był jak najlepszy, tj. aby spodziewany koszt implementacji $SKI(G) + SKI(H)$ był najmniejszy:

1. Tworzony jest początkowy zestaw zbiorów V i U, zgodnie z poniższymi zasadami:
   - Z wejść I funkcji F do zbioru U wybierane są podzbiory o liczbie elementów najbardziej zbliżonej do wartości $n/2$, gdzie n jest liczbą wejść funkcji F, przy czym w zbiorze U ma być mniej elementów niż w V;
   - Do zbioru V wybierane są wejścia binarne, które nie zostały wybrane do zbioru U (domyślnie poszukiwana jest dekompozycja rozłączna);
   - Wybierany jest pusty zbiór U, a do zbioru V przypisywane są elementy zbioru I.
2. Dla każdej pary zbiorów U i V wyznacza się przypuszczalny spodziewany koszt implementacji $SKI(H)$ oraz $SKI(G)$ i minimalną liczbę wyjść funkcji G.
3. Spośród wszystkich par zbiorów U i V wybiera się do dekompozycji te pary, które mają najmniejszą z możliwych wartość liczby wyjść funkcji G oraz najmniejszy koszt $SKI(H) + SKI(G)$, przy czym koszt brany jest pod uwagę w drugiej kolejności.

Dla zbiorów U i V wybranych w kroku 3 wykonuje się dekompozycję szeregową i jako ostateczny wynik wybierana jest najlepsza z uzyskanych.

Obliczenie najmniejszej z możliwych liczby wyjść, jakie po wyznaczeniu może posiadać funkcja G – wykonuje się to w oparciu o twierdzenie T3c. Jeśli przyjmie się najgorszy przypadek β_{Q_V}, równy nakryciu wejścia wielowartościowego β_X, co odpowiada sytuacji, w której $Q_V = X$, to zależność (1) przyjmie postać: $\beta_{V^*} * \beta_X = \beta_G$. Wynika z tego, że najgorszy przypadek β_G będzie równy $(\beta_{V^*} * \beta_X)$. Następnie, przyjmując najgorszy przypadek β_{Q_U}, także równy β_X i podstawiając do zależności (2), otrzymuje się:

$$\beta_{U^*} * \beta_X * \beta_G \leq \beta_F \tag{4}$$

gdzie:

$$\beta_{U^*} * \beta_G \leq \beta_F \tag{5}$$

po podstawieniu równania (5) do zależności (4) otrzymuje się: $\beta_U * \beta_X * \beta_V * \beta_X \leq \beta_F$. Zgodnie z własnością rachunku nakryć: $\beta_X * \beta_X = \beta_X$, jednocześnie biorąc pod uwagę zależność (5), otrzymuje się:

$$\beta_U * \beta_G \leq \beta_F \tag{6}$$

Zależność (6) jest prawdziwa tylko, gdy nakrycia β_{Q_V} i β_{Q_U} będą równe nakryciu β_X, czyli przyjmą najgorszy z możliwych przypadków (odpowiadający sytuacji, w której X jest wejściem funkcji G i H). Wtedy także zależność (4.3) pozwala na obliczenie liczby bloków, jakie muszą znaleźć się w nakryciu funkcji G, aby w wyniku dekompozycji funkcji wejściowej F można było uzyskać deterministyczne funkcje G i H. Do wyznaczenia tej wartości wykorzystuje się funkcję *gen_min_nakr*.

Na podstawie liczby bloków w nakryciu β_G, wyznaczonym w powyższy sposób, można obliczyć liczbę wyjść funkcji G (równą $[\log_2|\beta_G|]$), a więc także wskazać prawdopodobną liczbę wejść funkcji H. Wtedy można oszacować prawdopodobną liczbę wejść funkcji G i H w zależności od wyboru elementów (wejść) zbiorów V i U. W wyniku wykonania algorytmu zostaje wyznaczona funkcja G, funkcja H, wejścia wielowartościowe lub binarne Q_V i Q_U oraz wejście wielowartościowe lub binarne Q_C, jeśli została wykona dekompozycja nierozłączna.

Parametrami algorytmu dekompozycji szeregowej są nakrycia binarnych wyjść funkcji F, nakrycia wejść binarnych funkcji G i H (zbiory V i U) oraz nakrycie wejścia wielowartościowego X. Poniższy szkic zawiera główne kroki algorytmu:

function *ds_kodowanie_wejść*(β_{Q_V}, β_V, β_U, β_X, *l_blk_*β_G, *l_blk_*β_{Q_V})
β_M = *gen_min_nakr*(β_U, ($\beta_U * \beta_F$), β_F); *obliczenie* $|\beta_G|_{\min}$;
minimalna liczba wyjść w funkcji G = $[\log_2|\beta_G|_{\min}$; β_{X_1} = *pierwszy blok z* β_X;
$\beta_{Q_V} = \{\beta_{X_1}\}$; $\beta_{Q_U} = \{\beta_{X_1}\}$; $\beta_G = \beta_{V_i} * \beta_{Q_V}$; $\beta_X = \beta_X - \{\beta_{X_1}\}$;
obliczenie liczby bloków β_{Q_V};
do **{**
$\beta_{Q_{V_{\min}}} = \beta_{Q_V}$; $\beta_{Q_{U_{\min}}} = \beta_{Q_U}$; $\beta_{G_{\min}} = \beta_G$;
for *każdy blok* B_i *w nakryciu* β_X ***do*** **{**
wprowadzenie bloku B_i *do nakrycia* $\beta_{Q_{V_{tmp}}}$; $\beta_{G_{tmp}} = \beta_V * \beta_{Q_{V_{tmp}}}$;
for *każdy blok w nakryciu* β_{Q_U} ***do*** **{**
wprowadzenie bloku B_i *do* $\beta_{Q_{V_{tmp}}}$;
$\beta_{G_{tmp-\min}}$ = *gen_min_nakr*($\beta_{Q_{U_{tmp}}}$, $\beta_{G_{tmp}}$, β_M);
$\beta_{Q_U G} = \beta_{Q_{U_{tmp}}} * \beta_{G_{tmp-\min}}$; $\beta_{Q_U M} = \beta_{Q_{U_{tmp}}} * \beta_M$;
wyznaczenie warunków jakości rozwiązania częściowego;
if *lepsze rozwiązanie częściowe* ***then*** **{**
$\beta_{Q_{V_{\min}}} = \beta_{Q_{V_{tmp}}}$; $\beta_{Q_{U_{\min}}} = \beta_{Q_{U_{tmp}}}$; $\beta_{G_{\min}} = \beta_{G_{tmp-\min}}$;
} } }

if *nie było możliwości wyznaczenia lepszego rozwiązania* ***then*** **{**
$\beta_{Q_{V_{\min}}} = \beta_{Q_{V_{\min}}} +$ *pierwszy blok* β_X; $\beta_{Q_{U_{\min}}} = \beta_{Q_{U_{\min}}} +$ *pierwszy blok* β_X;
$\beta_{G_{\min}} =$ *gen_min_nakr*($\beta_{Q_{U_{\min}}}$, ($\beta_V * \beta_{Q_{V_{\min}}}$, β_m);
}
$\beta_{Q_V} = \beta_{Q_{V_{\min}}}$; $\beta_{Q_U} = \beta_{Q_{U_{\min}}}$; $\beta_G = \beta_{G_{\min}}$; *usunięcie wykorzystanego bloku z* β_X;
} ***while*** $\beta_X == \emptyset$;
return β_G, β_{Q_V}, β_G;

Na początku dekompozycji wyznaczane jest nakrycia β_M, obliczane z zależności (3). Nakrycie β_M upraszcza obliczanie innych zależności w algorytmie. Następnie wyznaczana jest najmniejsza z możliwych liczba wyjść funkcji G, która jest równa $[\log_2(|\beta_G|_{\min})]$, gdzie $|\beta_G|_{\min}$ oznacza najmniejsza z możliwych liczbę bloków w β_G. W dalszej części algorytm konstruuje bloki nakryć β_{Q_V} i β_{Q_U}, wykorzystując bloki nakrycia β_X. Algorytm wyznacza tymczasowe nakrycie $\beta_{Q_{V_{tmp}}}$, dla którego wyznaczane są tymczasowe nakrycia $\beta_{Q_{U_{tmp}}}$ oraz $\beta_{G_{tmp}}$. Odbywa się to zgodnie z tw. T3a. Spośród wszystkich, wyznaczonych w ten sposób nakryć tymczasowych $\beta_{Q_{U_{tmp}}}$ i $\beta_{G_{tmp}}$. wybierane są te, które pozwalają na najtańszą tymczasową implementacji funkcji F. Każdorazowo, po zaakceptowaniu jednego z rozwiązań, z nakrycia β_X wykorzystywany blok jest usuwany.

4.4. Kodowanie stanów

Kodowanie stanów wykonuje się, gdy funkcja F posiada wielowartościowe wejście X i wielowartościowe wyjście O, przyjmujące te same wartości symboliczne. Wtedy funkcja F może zostać uznana za reprezentującą automat FSM, gdzie wejście X reprezentuje stan bieżący, a wyjście O stan następny. Algorytm dekompozycji z kodowaniem stanów wykorzystuje przedstawione w poprzednich rozdziałach metody dekomponowania funkcji wielowartościowych i został podzielony na dwa etapy.

1. Utworzenie jednego poziomu sieci funkcji symbolicznych – funkcja wejściowa F zostaje zdekomponowana na dwie funkcje G i H. Wykorzystuje się do tego jeden z algorytmów dekompozycji z kodowaniem wejść. Wyjście wielowartościowe O zostaje przypisane do jednej z funkcji G lub H, albo zostaje zakodowane w obu funkcjach i w każdym z tych przypadków jest traktowane jako zwykłe wyjście wielowartościowe, które nie jest powiązane z wejściem X. Takie uproszczenie pozwala traktować funkcje G i H jako jeden z poziomów sieci funkcji symbolicznych – wyznaczone funkcje będą argumentami dla następnego wykonania tego etapu algorytmu, przy czym ten etap będzie powtarzany osobno dla obu funkcji, co spowoduje wygenerowanie kolejnego poziomu sieci (dwóch nowych węzłów sieci).
2. Pierwszy etapu algorytmu jest powtarzany aż do uzyskania takiego zakodowania zmiennych symbolicznych, w którym wszystkie funkcje sieci będą zależały od wejść binarnych w liczbie nie większej niż posiada komórka LUT.
3. Na tym etapie następuje przypisanie kodów binarnych do wartości symbolicznych wyjścia O lub, gdy było ono dekomponowane, wyjść O_{G_n} i O_{H_n} (n – numer węzła sieci).

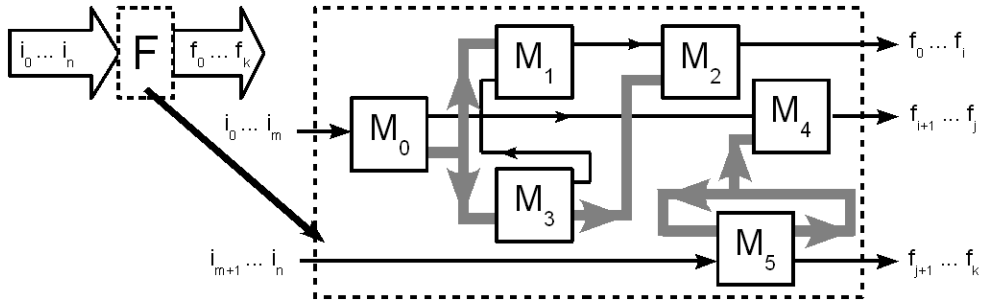
Ponieważ w funkcji F wejście wielowartościowe X przyjmuje takie same wartości jak wyjście O, to powyższa czynność sprowadza się do wyznaczenia kodów dla wejścia X. Stąd kod binarny dla wartości symbolicznej S_i wejścia X zostaje utworzony poprzez połączenie kodów binarnych wejść Q_{V_n} i Q_{U_n} (n – numer węzła sieci), wyznaczonych dla tego samego sześcianu co wartość S_i.

Wykorzystanie w powyższym algorytmie metod dekompozycji z kodowaniem wejść wymaga wyznaczania parametrów wejściowych, właściwych dla danej metody. Związane z tym czynności są identyczne, jak zostało to przedstawione wcześniej.

Kodowanie stanów wymaga uwzględnienia obecności zmiennych wielowartościowych występujących po stronie wejść oraz wyjść układu, dla których, po wykonaniu kodowania, wartości symboliczne muszą mieć identyczne kody. Z tego powodu, podczas wykonywania czynności z pierwszego etapu, kodowane jest tylko wejście wielowartościowe, przy czym poszukiwane są funkcje G i H posiadające określone parametry, np. funkcje o zadanej liczbie wejść. Rodzaj tych parametrów jest uzależniony od zapotrzebowania algorytmu na określoną funkcję tj. takie, które nie spowodują zwiększenia liczby bitów kodu potrzebnego dla zakodowania stanów FSM i dodatkowo uproszczą dekomponowaną funkcję. Dodatkowo preferowane są nierozłączne dekompozycje z kodowaniem wejść. Spowodowane jest to faktem, że każdy dodatkowy bit kodujący wartości symboliczne będzie wymagał do implementacji kolejnej komórki LUT.

4.5. Dekompozycja sieci wielowartościowej

Sieć wielowartościowa (*MV-network*) jest złożoną strukturą logiczną, stanowiącą uogólnienie sieci binarnej. Składa się z modułów połączonych ze sobą co najmniej jednym wejściem/wyjściem, przyjmującym wartości symboliczne. Obok wejść i wyjść wielowartościowych w strukturze *MV* mogą występować także wejścia i wyjścia binarne. Na rysunku 2 została przedstawiona przykładowa struktura układu składającego się z modułów (M_0 ... M_5), zawierającego sieć wielowartościową (gruba linia) oraz sygnały binarne (cienkie linie).



Rys. 2. Przykład struktury sieci wielowartościowej

Fig. 2. Example of Multivalued Network

Sieć wielowartościowa może zostać zdefiniowana na etapie projektowania układu lub może zostać wyznaczona jako wynik syntezy wysokopoziomowej. W zależności od rodzaju

wejść i wyjść, jakie posiadają moduł sieci *MV*, można wykonać określone czynności związane z jej dekompozycją. Jeśli moduł posiada:

– tylko wejścia symboliczne (np. moduły M_1, M_2 i M_4), to można jedynie wykonać dekompozycję z kodowaniem wejść – moduł będzie nazywany *BI*;
– jedynie wyjście wielowartościowe (np. moduł M_0), to można jedynie wykonać dekompozycję z kodowaniem wyjść – taki moduł będzie dalej nazywany *BO*;
– wejścia i wyjścia wielowartościowe, które przyjmują różne wartości symboliczne (np. moduł M_3), to możliwe są: dekompozycja z kodowaniem wejść, dekompozycja z kodowaniem wyjść, dekompozycja z kodowaniem wyjść i wejść – moduł *BIO*;
– wejście i wyjście wielowartościowe, które przyjmuje takie same wartości symboliczne (np. moduł M_5), to realizuje on automat FSM i będzie podlegał dekompozycji z kodowaniem stanów – taki moduł będzie dalej nazywany *BS*.

O rodzaju zastosowanego algorytmu dekompozycji w głównej mierze decydują rodzaje wejść i wyjść, jakie posiada moduł. Dla powyższego przykładu w pierwszej kolejności wykonuje się dekompozycję modułu M_5, ze względu na konieczność zapewnienia poprawności kodowania stanów. W takiej sytuacji wejścia wielowartościowe modułu M_4 przyjmą kody binarne wyznaczone dla modułu M_5. Wykonanie dekompozycji z kodowaniem wejść dla jednego z modułów M_1 lub M_3 spowoduje wyznaczenia także wyjść binarnych dla modułu M_0. Jeśli dla modułu M_3 zostanie wykonana dekompozycja z kodowaniem wyjść i wejść, to zostaną także wyznaczone wejścia binarne dla modułu M_2.

Powyższe przykłady oraz właściwości algorytmów dekompozycji pozwalają na określenie porządku stosowanego w dekompozycji sieci wielowartościowej *MV*. Moduły sieci *MV* są wybierane do dekompozycji w następującej kolejności:

1) moduł typu *BS* – dekompozycja z kodowaniem stanów – ten rodzaj dekompozycji ma największe możliwości optymalizacji,
2) moduł typu *BIO* – dekompozycja z kodowaniem wyjść i z kodowaniem wejść,
3) moduł typu *BI* – dekompozycja z kodowaniem wejść,
4) moduł typu *BO* – dekompozycja z kodowaniem wyjść – ten rodzaj dekompozycji ma najmniejszą skuteczność optymalizacji wynikającą z możliwości wykonania tylko dekompozycji równoległej wyjść wielowartościowych.

Dekompozycja dowolnego modułu typu *BS* lub *BIO* może przekształcić połączone z nimi inne moduły typu *BIO* w moduły typu *BI* lub *BO*, dlatego bardzo istotna jest kolejność dekompozycji.

5. Wyniki badań eksperymentalnych

Przedstawione powyżej algorytmy dekompozycji z kodowaniem wartości symbolicznych zostały zaimplementowane, a uzyskany w ten sposób program narzędziowy posłużył do wykonania badań eksperymentalnych. W przypadku dekompozycji z kodowaniem wejść do badań wykorzystano układy benchmarkowe MCNC PLA, poddane konwersji przy pomocy programu ESPRESSO (opcja -o fr). Dla pozostałych algorytmów do badań wykorzystano układy testowe MCNC FSM. W przypadku dekompozycji z kodowaniem wyjść wejścia układów testowych zostały zakodowane losowymi wartościami binarnymi. Dla algorytmu z kodowania wyjść i wejść przyjęto, że układy testowe mają rozdzielone wejście i wyjście wielowartościowe. Taki sposób przygotowania układów do eksperymentów był

również stosowany przez innych autorów [3, 10]. Dla algorytmu dekompozycji z kodowaniem stanów układy testowy były wykorzystane bez zmian. W poniższych tabelach zostały zamieszczone wyniki uzyskane dla powyższych algorytmów dekompozycji w porównaniu z innymi algorytmami, najczęściej pojawiającymi się w literaturze. Eksperymenty zostały wykonane dla komórek LUT o 4 wejściach i 1 wyjściu – ten typ jest najczęściej wykorzystywany przez innych autorów.

Tabela 1

Wyniki dekompozycji z kodowaniem wejść w porównaniu z innymi metodami

Układ	Liczba LUT / liczba poziomów logicznych								
	Dekompozycja z kodowaniem wejść	DEMAIN	BDD	Mispga-d	Chortle-d	Flow-Map	ASYL	Quartus II 9.0	ABC
5px1	**9 / 1**	9 / 2	19 / 2	21 / 2	29 / 4	25 / 3	13 / 3	24 / 4	25 / 3
9sym	**5 / 3**	5 / 3	6 / 3	10 / 2	20 / 3	13 / 3	4 / 2	9 / 3	65 / 5
9symml	**5 / 3**	–	6 / 3	7 / 3	130 / 5	58 / 5	8 / 3	9 / 3	91 / 6
clip	**10 / 3**	16 / 2	–	54 / 4	–	–	46 / 4	76 / 7	22 / 3
f51m	**8 / 1**	10 / 2	81 / 3	23 / 4	65 / 5	–	16 / 3	19 / 4	14 / 3
misex1	**8 / 2**	8 / 2	18 / 2	17 / 2	25 / 3	25 / 3	13 / 2	23 / 4	15 / 2
rd73	**5 / 2**	5 / 2	8 / 2	8 / 2	52 / 4	–	8 / 2	12 / 3	14 / 3
rd84	**6 / 3**	7 / 2	13 / 3	13 / 3	61 / 4	43 / 4	14 / 3	18 / 3	39 / 4
sao2	**12 / 3**	18 / 3	26 / 4	45 / 5	58 / 5	–	36 / 3	63 / 7	42 / 4
alu2	**15 / 4**	53 / 5	95 / 6	122 / 6	227 / 9	162 / 8	60 / 6	35 / 4	48 / 5
z4ml	**4 / 1**	4 / 2	6 / 2	10 / 2	20 / 3	13 / 3	4 / 2	8 / 3	10 / 3
root	**5 / 1**	37 / –	–	–	–	–	–	53 / 5	56 / 5

Wyniki zamieszczone w powyższej tabeli dla DEMAIN, BDD, Mispga-d, Chortle-d, Flow-Map i ASYL pochodzą z [23], natomiast wyniki dla Quartus II 9.0 [24] i ABC 1.01 (wersja 70930) [25] uzyskano wprost z tych systemów, w których syntezę wykonano dla układów testowych bez wprowadzania żadnych zmian, a powyższe wyniki także należy traktować jak dla jednego z możliwych przypadków kodowania binarnego wejścia wielowartościowego. Uzyskane wyniki potwierdzają, że liczba komórek LUT potrzebnych do implementacji funkcji, jak i liczba poziomów logicznych, zależą od sposobu kodowania wartości symbolicznych wejścia wielowartościowego. Jednocześnie krótszy kod binarny nie zapewnia mniejszego wykorzystania zasobów układu FPGA niż kod o większej liczbie bitów.

Do oceny algorytmu dekompozycji z kodowaniem wyjść zostały zastosowane benchmarki MCNC dla FSM, w których wyjście wielowartościowe zostało zakodowane losowo wartościami binarnymi, przy czym wybrana została minimalna długość kodu. W ten sposób powstały układy posiadające jedynie wejścia i wyjścia binarne.

Tabela 2

Wyniki dekompozycji z kodowaniem wyjść oraz z kodowaniem wyjść i wejść w porównaniu z syntezą w systemach Quartus II 9.0 i ABC 1.01

Układ	Kodowanie wyjść			Kodowanie wyjść i wejść		
	proponowane rozwiązanie	Quartus II 9.0	ABC	**proponowane rozwiązanie**	Quartus II 9.0	ABC
bbara	**27 ***	22	27	**19**	22	27
bbtas	**5 ***	4	7	**5 ****	4	7
beecount	**9 ***	7	10	**5**	7	10
dk15	**4**	4	5	**4**	4	5
dk17	**8**	8	12	**7**	8	12
dk27	**4 ***	3	3	**3**	3	3
dk512	**9**	10	17	**5**	10	17
donfile	**22 (2)**	26	31	**8 (1)**	26	31
ex2	**44**	68	57	**38**	68	57
ex3	**14**	20	19	**14**	20	19
ex4	**14**	16	16	**9**	16	16
ex5	**11**	13	15	**10**	13	15
ex6	**18 ***	17	17	**16**	17	17
lion	**2**	2	2	**2**	2	2
lion9	**3 (2)**	13	15	**3 (1)**	13	15
mc	**3**	3	3	**3**	3	3
opus	**15 (1)**	19	21	**10 (1)**	19	21
s27	**4**	6	6	**4**	6	6
s8	**9**	13	17	**5**	13	17
shiftreg	**0 (3)**	0 (3)	3	**2 (1) ****	0 (3)	3
tav	**1 (1)**	1 (1)	2	**2 ****	1 (1)	2
tma	**44**	70	57	**16**	70	57
train11	**8 (1)**	18	16	**6**	18	16
train4	**2**	2	2	**2**	2	2
Σ	**278 (10)**	365 (4)	380 (0)	**198 (4)**	365 (3)	380 (0)

Dla tych układów została wykonana synteza w systemach Quartus II 9.0 i ABC 1.01. Uzyskane wyniki przedstawia tabela 2 (lewa część). Algorytm dekompozycji z kodowaniem wyjść ma ograniczone możliwości optymalizacji (w stosunku do algorytmów z kodowaniem wejść) – możliwe jest tylko wykonanie dekompozycji równoległej. Z tego względu, w kilku przypadkach, otrzymano gorsze wyniki niż z syntezy innymi metodami (zaznaczone znakami *). W tabeli 2 wartości umieszczone w nawiasach oznaczają liczbę wyjść, które zależą od jednego wejścia i nie wymagają do implementacji żadnych komórek LUT.

W tabeli 2 zamieszczone zostały także wyniki dekompozycji z kodowaniem wyjść i wejść (prawa część tabeli) w porównaniu z wynikami uzyskanymi z systemów Quartus II 9.0 i ABC 1.01. Ze względu na ograniczenia w optymalizacji algorytmem dekompozycji z kodowaniem wyjść – możliwość wykonania jedynie dekompozycji równoległej – nie we wszystkich przypadkach uzyskano lepsze wyniki w stosunku do syntezy innymi metodami (przypadki zaznaczone w tabeli 2 znakami **). Układ *bbtas* posiada małą liczbę sześcianów i małą liczbę wartości symbolicznych w wyjściu wielowartościowym. Z tego powodu pominięcie etapu dekompozycji z kodowaniem wyjść, a wykonanie samej dekompozycji z kodowaniem wejść pozwala na uzyskanie implementacji wymagającej czterech komórek LUT. Benchmarki *shiftreg* i *tav* posiadają wyjścia wielowartościowe o małej liczbie wartości symbolicznych. W obu tych przypadkach możliwość wyznaczenia wyjścia binarnego zależącego od jednego wejścia binarnego jest ograniczona po wykonaniu etapu algorytmu z dekompozycją z kodowaniem wejść. Jednakże taka możliwość pojawiała się w innych przykładach *donfile*, *lion9*, *opus*.

Tabela 3

Wyniki dekompozycji z kodowaniem stanów w porównaniu z innymi metodami

Układ	**Dekompozycja z kodowaniem stanów**	Quartus II 9.0			ABC		
		JEDI	NOVA	one-hot	JEDI	NOVA	one-hot
bbara	**19 ****	18	17	22	20	28	83
bbtas	**6**	6	6	9	6	8	17
beecount	**12**	16	19	31	22	19	70
dk15	**14**	14	15	35	16	17	49
dk17	**13 ****	11	11	23	17	17	53
dk27	**5**	5	5	8	5	5	19
dk512	**13**	17	15	18	20	18	71
donfile	**21 ***	32	31	49	39	35	173
ex2	**30**	51	62	30	62	52	157
ex3	**12**	22	26	17	31	30	62
ex4	**22**	36	54	24	27	37	94
ex5	**12**	18	21	13	20	26	52
ex6	**33**	59	58	48	36	45	103
lion	**3**	3	3	8	3	3	12
lion9	**6**	15	20	20	14	23	45
mc	**6 (2)**	6 (2)	7 (1)	10	10	9	19
opus	**24**	32	34	29	36	39	90
s27	**11 ****	5	10	21	8	13	40
s8	**6 ***	21	23	23	19	19	29
shiftreg	**3**	3	3	9	4	4	21
tav	**9**	9	9	11	9	9	19
tma	**37**	122	131	84	97	124	194
train11	**15**	19	24	18	28	23	60
train4	**3**	3	3	6	3	3	11
Σ	**335 (2)**	543 (2)	607 (1)	566	552	606	1543

Do oceny algorytmu dekompozycji z kodowaniem stanów zostały zastosowane benchmarki MCNC dla FSM, przygotowane w sposób podany na początku tego rozdziału. Następnie w tych układach wykorzystano metody kodowania stanów JEDI [26], NOVA [27] oraz one-hot. Dla tak zakodowanych układów została wykonana synteza w systemach Quartus II 9.0 i ABC 1.01. Uzyskane wyniki przedstawia tabela 3.

W powyższej tabeli układy zaznaczone * posiadają wyjścia binarne, które można zaimplementować niezależnie od sposobu kodowania automatu FSM. W tych przypadkach przyjęto, że układ będzie dodatkowo posiadał wyjścia binarne powstałe w wyniku kodowania wyjścia wielowartościowego (reprezentującego stan następny FSM). W tabeli 3 wartości umieszczone w nawiasach oznaczają liczbę wyjść, które zależą od jednego wejścia i nie wymagają do implementacji żadnych komórek LUT.

W przypadku algorytmu dekompozycji z kodowaniem stanów, większa liczba komórek LUT potrzebna do implementacji układów wynika z bardziej złożonej struktury sieci wielowartościowej, powstałej na skutek dekompozycji z kodowaniem wejść tych układów (benchmarki zaznaczone w tabeli 3 znakami **). W benchmarku *s27* istnieje możliwość uzyskania implementacji wymagającej pięciu komórek LUT, jednakże algorytm dekompozycji z kodowaniem wejść należałoby wykonać jednokrotnie – czyli do utworzenia jednego poziomu sieci wielowartościowej. Wtedy pozostałe etapy syntezy powinno się wykonać poprzez optymalizację funkcji binarnych. Podobnie lepsze wyniki można uzyskać dla *bbara* i *dk17*.

6. Wnioski

Przedstawione algorytmy są efektem badań nad metodami i skutecznością technik kodowania wartości symbolicznych w połączeniu z dekompozycją funkcji wielowartościowych przeznaczonych do implementacji w układach FPGA zawierających komórki LUT. Przedstawiono zarówno metodologie, jak i teorie uzupełniające rachunek nakryć, stanowiący podstawowy aparat matematyczny, wykorzystany w algorytmach. Omówione zostały także różne aspekty dekompozycji połączonej z kodowaniem wartości symbolicznych, co zaowocowało opracowaniem czterech nowych metod syntezy układów zawierających zmienne symboliczne – dekompozycji z kodowaniem wejść, dekompozycji z kodowaniem wyjść, dekompozycji z kodowaniem wyjść i z kodowaniem wejść oraz dekompozycji z kodowaniem stanów.

Jak pokazały eksperymenty prezentowane algorytmy są stosunkowo skuteczne. Uzyskane wyniki dla dekompozycji z kodowaniem wejść potwierdzają, że liczba komórek LUT potrzebnych do implementacji funkcji, jak i liczba poziomów logicznych, zależą od sposobu kodowania wartości symbolicznych wejścia wielowartościowego. Jednocześnie krótszy kod binarny nie zapewnia mniejszego wykorzystania zasobów układu FPGA niż kod o większej liczbie bitów. W przypadku dekompozycji z kodowaniem wyjść wyniki pokazują, że także odpowiednie zakodowanie wyjść wielowartościowych pozwala na uzyskanie lepszych implementacji w stosunku do funkcji, których wyjścia symboliczne zostały wcześniej zakodowane binarnie. Natomiast połączenie obu technik kodowania umożliwia uzyskanie najlepszych dekompozycji, co zostało potwierdzone po wykonaniu eksperymentów dla algorytmu dekompozycji z kodowaniem wyjść i wejść. Także w przypadku dekompo-

zycji z kodowaniem stanów uzyskane wyniki pokazują, że powyższe algorytmy można również z powodzeniem stosować dla układów FSM.

Omówiono również zagadnienie dekompozycji sieci wielowartościowej. W podrozdziale 4.6 została przedstawiona wstępna analiza problemu oraz możliwe rozwiązania.

Najważniejszą cechą zaproponowanych rozwiązań jest stosunkowo duża szybkość działania. Przykładowo dekompozycja funkcji z wejściem wielowartościowym posiadającym 100 wartości symbolicznych jest wykonywana w czasie krótszym niż 100 ms (na komputerze z procesorem Athlon XP 2500+ 1.83 MHz). Także wymagania pamięciowe nie są wygórowane. Na przykład dla benchmarka *clip*, posiadającego po zakodowaniu 2380 sześcianów, wykorzystanie pamięci nie przekroczyło kilku megabajtów.

Przyszłe prace związane z przedstawioną metodologią będą obejmowały:

– zwiększenie jakości algorytmów szacujących efektywności dekompozycji,
– opracowanie strategii dekompozycji sieci wielowartościowych (*MV*),
– wbudowanie metod kodowania wejść, wyjść, stanów i sieci *MV* w narzędzia syntezy wysokopoziomowej.

Literatura

[1] Brzozowski J., Łuba T., *Decomposition of Boolean Functions Specified by Cubes*, Journal of Mult.-Valued Logic & Soft Computing, Vol. 9, 2003, 377-417.

[2] Deniziak S., Wiśniewski M., *A symbolic RTL synthesis for LUT-based FPGAs*, Design and Diagnostics of Electronic Circuits & Systems, 102-107.

[3] Ashar P., Devadas S., Newton A.R., *Sequential Logic Synthesis*, Kluwer Academic Publisher, Norwell 1992.

[4] Brayton R.K., Khatri S., *Multi-valued Logic Synthesis*, Proc. of the International Conference on VLSI Design, 1999, 196-205.

[5] Deniziak S., Wiśniewski M., *An Integrated Input Encoding and Symbolic Functional Decomposition for LUT-Based FPGAs*, IEEE DDECS 2008, 22-25.

[6] Saldanha A., Villa T., Brayton R.K., Sangiovanni-Vincentelli A., *Satisfaction of input and output encoding constraints*, IEEE Trans. on CAD, Vol. 13, No. 5, 1994, 589-602.

[7] Martinez M., Avedillo M., Quintana J., Huertas J., *COPAS: A New Algorithm for the Partial Input Encoding Problem*, VLSI Design, Vol. 14 (2), 2002, 171-181.

[8] Martinez M., Avedillo M., Quintana J., Huertas J., *An Algorithm for Face-Constrained Encoding of Symbols Using Minimum Code Length*, Proc of the DATE, 1999.

[9] Yang S., Cieselski M., *Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization*, IEEE Trans. on CAD 10(1), 1991, 4-12.

[10] Malik S., Lavagno L., Brayton R.K., *Symbolic Minimization of Multilevel Logic and the Input Encoding Problem*, IEEE Trans. on CAD, Vol. 11, No. 7, 1992.

[11] Brzozowski J., Lou J., *Blanket algebra for multiple-valued function decomposition, Intern. Workshop on Formal Languages and Computer Systems 1997, in. Algebraic Engineering*, C.L. Nehaniv and M. Ito, eds. World Scientific, 1999, 262-276.

[12] Murgai R., Brayton R.K., Sangiovanni-Vincentelli A., *Optimum Functional Decomposition Using Encoding*, Proc. of the DAC, 1994, 408-414.
[13] Burns M., Perkowski M., Grygiel S., Jóźwiak L., *An Efficient and Effective Approach to Column-Based Input/Output Encoding in Functional Decomposition*, Proc. of the 3rd International Workshop on Boolean Problems, 1998, 19-29.
[14] Muthukumar V., *An Improved Input-Output Encoding Approach for Functional Decomposition*, Proc. of the Euromicro DSD, 2001.
[15] Mishchenko A., Brayton R.K., *A Boolean Paradigm in MultiValued Logic Synthesis*, Proc. of the IWLS, 2002.
[16] Mishchenko A., Sasao T., *Encoding of Boolean Functions and Its Application to LUT Cascade Synthesis*, Proc. of the IWLS, 2002.
[17] Deniziak S., *Kodowanie stanów dla układów FPD o architekturze opartej o komórki LUT*, VII Krajowa Konferencja Reprogramowalne Układy Cyfrowe, 2004, 19-26.
[18] Rawski M., Selvaraj H., Łuba T., Szotkowski P., *Application of Symbolic Functional Decomposition Concept in FSM Implementation targeting FPGA devices*, 6th International Conference on Computational Intelligence and Multimedia Applications, ICCIMA 2005, Las Vegas, Nevada, August 16–18, 2005, 153-158.
[19] Szotkowski P., Rawski M., *Symbolic Functional Decomposition Algorithm for FSM Implementation*, Int. Conf. on „Comuter as a Tools", EUROCON, 2007.
[20] Szotkowski P., Rawski M., *A Graph-Based Symbolic Functional Decomposition Algorithm for FSM Implementation*, Int. Conf. on Human System Interactivus, 2008, 34-39.
[21] Szotkowski P., Rawski M., *Improvements to Symbolic Functional Decomposition Algorithm for FSM Implementation in FPGA Devices*, Electronics and Telecomunications Quarterly, 55(2), 2009, 335-354.
[22] Szotkowski P., Rawski M., Selvaraj H., *A Graph-Based Approach to Symbolic Functional Decomposition of FSM*, System Science, 35(2), 2009, 41-47.
[23] Nowicka M., Łuba T., Rawski M., *FPGA-Based Decomposition of Boolean Functions. Algorithms and Implementation*, Proc. of the 6th Intern. Conf. on ACS, Szczecin 1999.
[24] Altera Corporation, *Quartus II Software*.
[25] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification* (http://www.eecs.berkeley.edu/ ~alanmi/abc/).
[26] Lin B., Newton A.R., *Synthesis of Multiple-Level Logic from Symbolic High-Level Description Languages*, IFIP International Conference on VLSI, 1989, 187-196.
[27] Villa T., Sangiovanni-Vincentelli A., *NOVA: State Assignment for Finite State Machines for Optimal Two-level Logic Implementation*, IEEE Transactions on CAD/ICAS, Vol. C-9, No. 9, 1990, 905-924.
[28] Hartmanis J., Stearns R.E., *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, Englewood Cliffs, New Jersey 1966.