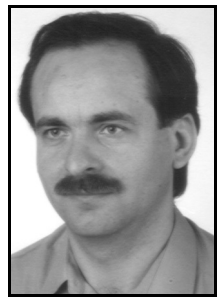
Stanisław DENIZIAK, **Karol WIECZOREK**
POLITECHNIKA ŚWIĘTOKRZYSKA, KATEDRA INFORMATYKI
Al. 1000-lecia Państwa Polskiego 7, 25-314 Kielce

Dekompozycja funkcji logicznych metodą rozwojowego programowania genetycznego

Dr hab. inż. Stanisław DENIZIAK

Ukończył studia na Wydziale Elektroniki Politechniki Warszawskiej, obronił pracę doktorską w 1994 r. Stopień naukowy doktora habilitowanego uzyskał w 2006. Jest profesorem nadzwyczajnym w Katedrze Informatyki WEAiI na Politechnice Świętokrzyskiej w Kielcach, oraz Katedrze Informatyki Technicznej Politechniki Krakowskiej. Jego zainteresowania naukowe to szybkie prototypowanie systemów cyfrowych, synteza systemowa, systemy wbudowane.



e-mail: S.Deniziak@computer.org

Mgr inż. Karol WIECZOREK

Tytuł zawodowy mgr inż. uzyskał w 2008 na kierunku Elektrotechnika, specjalność Informatyka techniczna - Wydział EAiI Politechniki Świętokrzyskiej. Pracuje na stanowisku asystenta w Katedrze Informatyki na Politechnice Świętokrzyskiej od 2008r. Jego działalność naukowa obejmuje zagadnienia związane z rozwojowym programowaniem genetycznym, dekompozycją funkcji logicznych, metodami heurystycznymi.

e-mail: k.wieczorek@tu.kielce.pl

Streszczenie

Praca przedstawia metodę wyszukiwania strategii dekompozycji funkcji logicznych za pomocą rozwojowego programowania genetycznego. Strategia dekompozycji jest reprezentowana w formie drzewa decyzyjnego, w którym węzły określają jeden krok dekompozycji. Drzewo podlega ewolucji, której celem jest uzyskanie jak najlepszego rozwiązania. Otrzymane wyniki wykonanych eksperymentów wskazują na wysoką skuteczność przedstawionej metody w porównaniu z dotychczas stosowanym podejściem deterministycznym.

Słowa kluczowe: dekompozycja funkcji logicznych, rozwojowe programowanie genetyczne, układy FPGA.

Functional decomposition of logical functions using developmental genetic programming

Abstract

Functional decomposition splits logical function into two simpler functions. For complex functions the decomposition should be repeated iteratively for the result functions. It was observed that types of decomposition applied during each step have strong influence on the final result. Thus, a proper decomposition strategy should be used to find optimal FPGA implementation for a given function. This paper presents the method for searching the decomposition strategy for logical functions specified by cubes. The strategy is represented using the decision diagram, in which each node corresponds to a single decomposition step. In this way the multistage decomposition of a complex logical function can be specified. The diagram is evolved using the developmental genetic programming. In opposite to classical genetic methods, in our approach the methods producing solutions, instead of the solutions, are evolved. The goal of the evolution is to find the decomposition strategy for which the cost of FPGA implementation of a given function is minimal. The experimental results show that our approach gives significantly better solutions than other known methods.

Keywords: functional decomposition, developmental genetic programming, FPGA devices.

1. Wstęp

Problem dekompozycji funkcji logicznych jest jednym z podstawowych problemów syntezy układów cyfrowych [1, 2]. W szczególności, efektywność stosowanych metod dekompozycji ma istotny wpływ na jakość syntezy logicznej funkcji implementowanych w układach FPGA [3]. Celem dekompozycji jest podział funkcji logicznej na jak najmniejszą liczbę funkcji, z których każda może być zaimplementowana w jednej komórce FPGA.

Opracowano wiele metod dekompozycji funkcji logicznych [3, 4]. W przypadku, gdy docelową technologią są układy FPGA złożone z komórek typu LUT, najefektywniejszą metodą dekompozycji okazała się dekompozycja funkcjonalna [4]. Polega ona na sukcesywnym podziale funkcji na funkcje prostsze, stosując dekompozycję szeregową i dekompozycję równoległą.

Opracowano efektywne metody znajdujące dekompozycję szeregową lub równoległą [5]. Jednak wymagają one wskazania zbiorów separowanych wejść. Zatem w wieloetapowej dekompozycji złożonych funkcji problemem pozostaje optymalny wybór dekompozycji stosowanych w kolejnych krokach. W dotychczasowych metodach stosowano tzw. dekompozycję zrównoważoną, polegającą na wyborze dekompozycji równoległej lub szeregowej w zależności od tego czy funkcja ma więcej wejść czy wyjść [5]. Jednak nie wykazano czy takie podejście jest zawsze efektywne. Również w zakresie wyboru zbiorów separowanych wejść stosowane są podejścia heurystyczne [6]. Metody te są zachłanne i wskazują najlepszą dekompozycję dla danego kroku, natomiast niekoniecznie taka dekompozycja może się okazać najlepsza dla całego procesu dekompozycji wielopoziomowej.

W pracy przedstawiono nową metodę funkcjonalnej dekompozycji wielopoziomowej. Kolejność dekompozycji złożonych funkcji logicznych jest optymalizowana za pomocą algorytmu genetycznego. Dekompozycja funkcji wykonywana jest z wykorzystaniem istniejących algorytmów dekompozycji równoległej i szeregowej. Natomiast sposób dekompozycji (tzn. typ dekompozycji oraz zbiory separowanych wejść), stosowanych na poszczególnych etapach, jest określany na podstawie ewolucji. W efekcie uzyskuje się sekwencje dekompozycji prowadzących do uzyskania funkcji o jak najmniejszym koszcie implementacji w FPGA.

W następnym rozdziale przedstawione są ogólne zasady rozwojowego programowania genetycznego. Rozdział 3 zawiera opis opracowanej metody. W rozdziale 4 przedstawione są wyniki wykonanych eksperymentów. Rozdział 5 zawiera wnioski.

2. Rozwojowe programowanie genetyczne

Algorytm genetyczny [7] jest probabilistyczną metodą optymalizacji polegającą na symulowaniu procesu ewolucji zbioru rozwiązań. Ewolucja ma na celu uzyskanie coraz lepszych rozwiązań poprzez tworzenie nowych pokoleń osobników. Każdy osobnik jest opisywany przez tzw. chromosomy składające się z genów, każdy gen reprezentuje jedną cechę rozwiązania. Tworzenie nowego pokolenia osobników odbywa się poprzez krzyżowanie, mutację i selekcję osobników z ostatniego pokolenia. Celem algorytmu jest poszukiwanie rozwiązania o optymalnym zestawie cech, czyli poszukiwanie optymalnego chromosomu. Najistotniejszą cechą algorytmu genetycznego jest możliwość wydobywania się z tzw. lokalnych ekstremów optymalizowanej funkcji, cechy tej nie posiadają metody deterministyczne.

Programowanie genetyczne [8] polega na generowaniu programów komputerowych poprzez zastosowanie algorytmu genetycznego. Osobnikami są programy komputerowe, najczęściej reprezentowane w formie drzewa struktury. W rozwojowym programowaniu genetycznym (RPG) [9] zamiast programów ewolucji podlega metoda tworzenia rozwiązania. W RPG wyróżnia się genotypy i fenotypy. Genotyp reprezentuje procedurę tworzącą

rozwiązanie, poszczególne geny reprezentują tzw. funkcje konstruujące rozwiązanie. Natomiast fenotyp opisuje cechy rozwiązania. Ewolucji podlegają jedynie genotypy, następnie każdy genotyp odwzorowywany jest w fenotyp poprzez wykonanie funkcji odpowiadających kolejnym genom. Dla każdego fenotypu obliczana jest jakość rozwiązania i na tej podstawie dokonywany jest ranking genotypów, służący do dalszej ewolucji. Celem RPG jest znalezienie metody konstruującej najlepsze rozwiązanie.

Metoda RPG jest szczególnie efektywna dla optymalizacji tzw. problemów silnie ograniczonych. Wtedy większość rozwiązań, generowanych losowo lub deterministycznie jest niepoprawna. Powoduje to konieczność narzucenia ograniczeń sterujących procesem ewolucji w taki sposób, aby rozpatrywane były tylko poprawne rozwiązania. Jednak ograniczenia te jednocześnie zawężają przestrzeń przeszukiwań, uniemożliwiając znalezienie rozwiązań najlepszych, jeśli są one efektem ewolucji rozwiązań niepoprawnych. Problemy te nie występują w RPG, gdzie ewolucja przebiega bez żadnych ograniczeń, natomiast poprawność rozwiązań jest zapewniona poprzez odpowiednie odwzorowanie genotypu w fenotyp. Optymalizacja metodą RPG okazała się skuteczna w takich problemach jak synteza układów elektronicznych, synteza algorytmów sterujących [10], kosynteza systemów wbudowanych [11].

3. Ewolucja procesu dekompozycji

W opisywanym podejściu genotyp jest reprezentowany przez drzewo binarne opisujące kolejność dekompozycji zadanej funkcji. Węzły drzewa odpowiadają procedurom wykonującym dekompozycję funkcji wejściowej na dwie funkcje wyjściowe, a poziomy drzewa kolejnym etapom dekompozycji. Funkcje będące wynikiem dekompozycji są argumentami dekompozycji na następnym poziomie drzewa genotypu. Za lepiej przystosowanego osobnika uznaje się tego, który tworzy tańszą implementację zadanej funkcji, czyli wymagającą mniejszej liczby komórek LUT.

3.1. Reprezentacje genotypu i fenotypu

Stosowanych jest 18 typów genów, określających procedury dekompozycji różniące się typem dekompozycji (szeregowa lub równoległa) oraz sposobem wyboru separowanych wejść lub wyjść. Do określenia zbiorów separowanych wejść/wyjść wykonywana jest analiza funkcji opisanej w formie sześcianów. Cechami wejść/wyjść uwzględnianymi w miarach, będących kryteriami separacji są: nie występowanie wejścia w sześcianach, występowanie wejść w tych samych sześcianach, występowanie wejść w sześcianach tylko części funkcji, itp. Ponadto liczba odseparowywanych wejść/wyjść może być zrównoważona (około połowy wszystkich wejść/wyjść) lub jest ograniczona do liczby wejść/wyjść docelowej komórki LUT.

Pokolenie początkowe jest złożone z losowo wygenerowanych osobników. Wszystkie geny są losowane z jednakowym prawdopodobieństwem. Aby uzależnić wielkość genotypów od złożoności dekomponowanej funkcji, przyjęto eksperymentalnie następującą liczbę węzłów w generowanym genotypie:

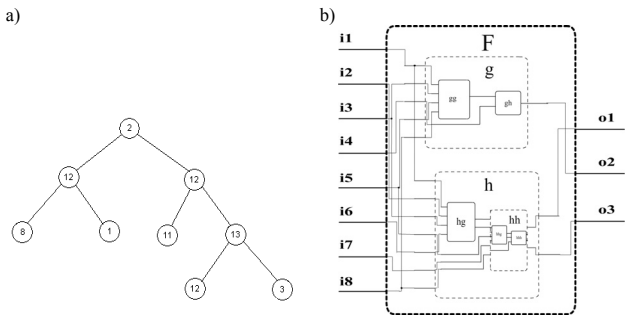
$$LW = \Theta(\frac{n*m}{k} * \mathrm{A}_{0.8}^{1.2}) \tag{1}$$

gdzie: n - liczba wejść funkcji, m - liczba wyjść, k - liczba wejść LUT, $\mathrm{A}_{0.8}^{1.2}$ jest liczbą losową z przedziału [0.8…1.2] a $\boldsymbol{\Theta}$ jest funkcją zaokrąglającą argument w górę do najbliższej naturalnej liczby nieparzystej. Na rys.1a przedstawiony jest przykładowy genotyp dla funkcji o 8 wejściach i 3 wyjściach, przy założeniu, że dekompozycja jest wyznaczana dla komórek LUT o 4 wejściach i jednym wyjściu, zatem liczba węzłów w drzewie genotypu wyniesie 5, 7 lub 9.

Odwzorowanie genotypu w fenotyp polega na przeglądaniu wszerz drzewa genotypu i wykonywaniu dekompozycji określonych przez przeglądane węzły. Argumentem dekompozycji związanej z korzeniem drzewa jest funkcja wejściowa. Funkcje będące wynikiem dekompozycji są argumentami dekompozycji określonej przez węzły będące następnikami danego węzła. Gdy funkcja nie wymaga dalszej dekompozycji dalsze przeglądanie poddrzewa nie jest wykonywane. Mogą zaistnieć dwie wyjątkowe sytuacje. Po pierwsze, węzeł może posiadać węzły potomne, a funkcja nie będzie wymagać dalszej dekompozycji. Po drugie, może się zdarzyć, że napotkany jest liść drzewa, a funkcja wymaga dalszej dekompozycji. W pierwszym przypadku przyjęto, że takie geny są od razu usuwane, podobnie jak nieużywane cechy osobników żywych. W drugim przypadku wynik dekompozycji jest szacowany według następującej zasady:

$$OKI = 2^{n-k} * m \tag{2}$$

Jest to tzw. oczekiwany koszt implementacji szacowany w sposób dostatecznie pesymistyczny, aby ewolucja eliminowała osobników zawierających węzły z szacowanymi rozwiązaniami, a preferowała te, które dekomponują funkcję do końca.



Rys. 1. Przykładowy genotyp a) i odpowiadający mu fenotyp b)
Fig. 1. Sample genotype and the corresponding phenotype

Fenotyp opisujący cechy otrzymanego osobnika jest zbiorem funkcji, na które została zdekomponowana funkcja *F*. Każda z tych funkcji ma co najwyżej *k* argumentów. Dla przykładowego genotypu przedstawionego na rys.1a, funkcja *F* została zdekomponowana na 5 funkcji: *gg, gh, hg, hhg, hhh* (rys.1b). Odwzorowanie genotypu w fenotyp przebiegało następująco:

- metoda 2 określa dekompozycję równoległą, równomiernie dzielącą zbiór wyjść, wg tej zasady funkcja *F* została podzielona na funkcje *g* i *h,*
- funkcja *g* jest dekomponowana wg metody 12, oznaczającej dekompozycję szeregową, zbiorem separowanych wejść jest zbiór mający wpływ na jak najmniejszą liczbę wyjść, wynikiem są funkcje *gg* i *gh,*
- funkcja *h* jest dekomponowana również wg metody 12, wynikiem są funkcje *hg* i *hh,*
- funkcja *hh* jest dekomponowana metodą 13, co również odpowiada dekompozycji szeregowej ale z inną separacją wejść,
- pozostałe węzły dotyczą funkcji, które nie wymagają dalszej dekompozycji, zatem węzły te zostaną usunięte z dalszej ewolucji genotypu.

3.2. Operatory genetyczne

W każdym pokoleniu jest niezmienna liczba osobników. Aby uzależnić liczebność pokolenia od złożoności funkcji, liczba osobników jest określona jako:

$$N = (n+m)*W \tag{3}$$

gdzie W jest parametrem algorytmu.

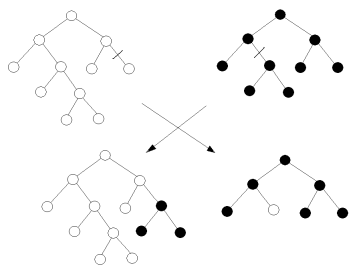
Ranking rozwiązań jest określony na podstawie kosztu implementacji (liczba komórek LUT docelowego układu FPGA). Lepszym osobnikiem jest ten, którego fenotyp ma niższy koszt implementacji. Na podstawie pozycji w rankingu wyznaczane jest

prawdopodobieństwo P udziału osobnika w ewolucji (udział w reprodukcji, krzyżowaniu oraz mutacji):

$$P = \frac{N - R}{N} \tag{4}$$

gdzie R – pozycja osobnika w rankingu.

Reprodukcja kopiuje r najlepszych osobników z pokolenia n do pokolenia $n+1$, aby najlepiej przystosowane osobniki przetrwały. Operator krzyżowania wybiera losowo z prawdopodobieństwem P dwóch osobników, a następnie wyznacza losowo punkt przecięcia genotypu dla obu rodziców. Operacji krzyżowania nie podlegają tylko osobniki, których genotyp składa się z jednego węzła. Przykład krzyżowania się genotypów przedstawia rys. 2.



Rys. 2. Krzyżowanie się genotypów dwóch osobników
Fig. 2. Cross-over of sample genotypes

Operator mutacji losuje osobnika z rankingu, a następnie genotyp wybranego osobnika poddawany jest jednej z trzech operacji:
- wymiana węzła, gdzie jeden gen jest zastępowany innym, losowo wybranym,
- dodanie do losowo wybranego liścia dwóch losowo wygenerowanych węzłów,
- usunięcie części genotypu w losowo wybranym punkcie.

Wybór typu mutacji odbywa się losowo z jednakowym prawdopodobieństwem oprócz sytuacji, gdzie do mutacji przystępuje osobnik, którego genotyp ma tylko jeden węzeł. Wtedy nie jest brana pod uwagę operacja usuwania części genotypu.

Nowe pokolenie jest tworzone w następujących proporcjach: reprodukcji podlega $r=\alpha*N$ osobników, krzyżowaniu $k=\beta*N$ osobników, a mutacji $m=\gamma*N$. Liczba osobników N jest liczona ze wzoru (3), a współczynniki α, β, oraz γ są parametrami ewolucji, dla których zawsze zachodzi zależność:

$$\alpha + \beta + \gamma = 1 \tag{5}$$

Algorytm zatrzymuje się, jeżeli w λ kolejnych pokoleniach nie znaleziono lepszego rozwiązania.

4. Wyniki wykonanych eksperymentów

W celu oceny skuteczności opisywanej metody, zaimplementowano ją i wykonano eksperymenty dla popularnych benchmarków MCNC. Dobrano eksperymentalnie następujące parametry: W=10, α=0.05, β=0.70, γ=0.25, λ=25. Uzyskane wyniki znajdują się w tab.1 (kolumna RPG). Dla porównania w tab.1 zostały zamieszczone również wyniki otrzymane za pomocą narzędzia GUIDek[12], wykonującego dekompozycję algorytmem deterministycznym [6]. Średni zysk uzyskany dzięki zastosowaniu RPG w porównaniu z podejściem deterministycznym wynosi 24%.

5. Wnioski

W pracy przedstawiono zastosowanie rozwojowego programowania genetycznego do dekompozycji funkcji logicznych. Według naszej wiedzy jest to pierwsze wykorzystanie tej metody do rozwiązania tego problemu. Olbrzymia ilość możliwych rozwiązań sprawia, że przeszukiwanie wyczerpujące, jest zbyt czasochłonne, aby można je było stosować w praktyce. Każda dekomponowana funkcja ma swoje indywidualne cechy, dlatego deterministyczne podejście nie zawsze jest skuteczne. RPG ma tę zaletę, że elastycznie dopasowuje strategię dekompozycji wielopoziomowej do rozpatrywanego problemu, a także wykonuje globalną optymalizację dekompozycji wielopoziomowej.

Tab. 1. Wyniki dekompozycji
Tab. 1. Decomposition results

Nazwa	RPG	GUIDek
5xp1	16	22
dk17	24	32
dk27	12	13
inc	27	35
m1	20	25
misex1	15	20
newcpla2	23	33
rd53	5	5
seq	11	23
sqrt8	10	13
squar5	10	14
t4	12	17
tms	59	73
SUMA	**251**	**325**

Wstępne badania pokazały, że metoda jest skuteczna, dlatego dalsze prace będą polegać na badaniu wpływu parametrów algorytmu na otrzymywane wyniki dekompozycji, a także na badaniu innych metod implementacji operatorów genetycznych. Dalszych badań wymaga również opracowanie optymalnego zbioru możliwych genów, definiujących wszystkie możliwe pojedyncze kroki dekompozycji. Przewiduje się tutaj implementację i badania istniejących metod separacji wejść [13].

6. Literatura

[1] Ashenhurst R.L.: The Decomposition of Switching Functions, Proc. of Int. Symp. on Theory of Switching Functions, pp.74-116, 1957.
[2] Ashar P., Devadas S., Newton A. R.: Sequential Logic Synthesis, Kluwer Academic Publisher, Norwell, 1992.
[3] Scholl C.: Functional Decomposition with Application to FPGA Synthesis, Kluwer Academic Publishers, 2001.
[4] Brzozowski J., Łuba T.: Decomposition of Boolean Functions Specified by Cubes, Journal of Mult.-Valued Logic & Soft Computing, vol.9, 2003, pp. 377-417.
[5] Nowicka M., Łuba T., Rawski M.: FPGA-Based Decomposition of Boolean Functions. Algorithms and Implementation, Proc. of the 6th International Conference on Advanced Computer Systems, 1999.
[6] Rawski M.: Efficient Variable Partitioning Method for Functional Decomposition, Electronics and Telecommunications Quarterly, vol. 53, no 1, pp. 63-81, 2007.
[7] Goldberg D. E.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.
[8] Koza J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.
[9] Keller R.E., Banzhaf W.: The evolution of genetic code in genetic programming, Proc. of the Genetic and Evolutionary Computation Conference, 1999, pp.1077-1082.
[10] Koza J.R., Keane M.A., Streeter M.J., Mydlowec W., Yu J., Lanza G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Kluwer, 2003.
[11] Deniziak S., Górski A.: Hardware/Software Co-Synthesis of Distributed Embedded Systems Using Genetic Programming, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp.83-93.
[12] http://rawski.zpt.tele.pw.edu.pl/pl/node/161
[13] Muthukumar V., Bignall R.J., Selvaraj H.: An efficient variable partitioning approach for functional decomposition of circuits, Journal of Systems Architecture, vol.53 (2007), pp. 53-67.

otrzymano / received: 18.09.2010
przyjęto do druku / accepted: 01.11.2010 *artykuł recenzowany*