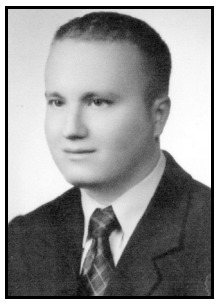
Maciej MAJCHRZYK, Grzegorz BOROWIK, Radoslav DARAKCHIEV
POLITECHNIKA WARSZAWSKA, INSTYTUT TELEKOMUNIKACJI

Realizacja dekodera adresów z zastosowaniem w pełni określonych funkcji boolowskich

Mgr inż. Maciej MAJCHRZYK



Ukończył studia na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej w 2006 r. Jest doktorantem w Instytucie Telekomunikacji PW. Jego zainteresowania naukowe obejmują zagadnienia syntezy, optymalizacji i dekompozycji układów cyfrowych oraz szeroko rozumianego programowania.

e-mail: M.Majchrzyk@stud.elka.pw.edu.pl

Dr inż. Grzegorz BOROWIK

Ukończył studia na Wydziale Fizyki Technicznej i Matematyki Stosowanej Politechniki Warszawskiej na kierunku – matematyka. Pracę doktorską obronił w 2007 r. na Wydziale Elektroniki i Technik Informacyjnych PW, uzyskując stopień doktora nauk technicznych w dyscyplinie Telekomunikacja. Aktualnie jest adiunktem w Instytucie Telekomunikacji PW. W polu jego zainteresowań naukowych leży szeroko rozumiana dekompozycja funkcjonalna oraz jej wykorzystanie w układach cyfrowych FPGA.

e-mail: gborowik@tele.pw.edu.pl

Inż. Radoslav DARAKCHIEV

Ukończył studia inżynierskie na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej w 2007 roku. Obecnie uczestnik studiów II stopnia na tym samym wydziale. Jest zatrudniony jako młodszy specjalista w Instytucie Tele- i Radiotechnicznym w Warszawie. Uprawiana dziedzina – układy cyfrowe.

e-mail: R.Darakchiev@stud.elka.pw.edu.pl

Streszczenie

Prezentujemy efektywną metodę syntezy w pełni określonych funkcji boolowskich charakteryzujących się dużą dysproporcją występującą na wyjściu. Opisywane funkcje zawierają jedynie mały podzbiór słów dla których wartość jest równa 1. Opracowano specjalny algorytm selekcji takich wektorów. Badania zostały wykonane na układach programowalnych FPGA Stratix firmy Altera. W porównaniu do klasycznych metod syntezy osiągnęliśmy, przy porównywalnym użyciu wbudowanych bloków pamięciowych EMB, redukcję zasobów logicznych LUT – średnio do 95%.

Słowa kluczowe: Synteza logiczna, Generator adresu, Układy sekwencyjne, Dekompozycja, FPGA, Pamięci wbudowane, Komórka logiczna.

Address Generator Realization Using Completely-Specified Boolean Functions

Abstract

We are proposing a cost-efficient realization scheme for completely-specified logic functions characterized by a huge disproportion. The functions described contain millions of input words but only few of them can give us information. An appropriate method of logic synthesis for identifying mentioned vectors (registered vectors) has been developed. In this method logic functions are implemented using both embedded memory blocks and LUT-based programmable logic blocks available in today's FPGAs. In comparison with the classical logic synthesis methods we have obtained extremely encouraging results: with a comparable number of EMBs, the number of logic cells has been reduced by 95%. The investigation has been implemented using Altera's Stratix devices.

Keywords: Logic Synthesis, Address Generator, Finite State Machine, Decomposition, FPGA, Embedded Memories, Logic Cell.

1. Wprowadzenie

Wielowejściowe funkcje logiczne charakteryzujące się dużą dysproporcją liczby słów, dla których wartość funkcji jest równa 1 (wektory rejestrowane), w stosunku do tych, które na wyjściu dają 0 (lub odwrotnie), są bardzo często wykorzystywane w układach cyfrowych dla różnych zastosowań.

Aby zaprezentować naszą metodę użyliśmy tzw. generatora adresów (GA). Układ taki, jest typowym reprezentantem opisywanych funkcji. Charakteryzuje się długim wektorem wejściowym oraz relatywnie małą liczbą wektorów rejestrowanych. Opisywany GA wykorzystywany jest w wielu dziedzinach, takich jak Internet - jako filtr adresów IP, w układach pamięciowych - memory patching circuits, pattern matching, itp. Wraz z rozwojem tych dziedzin należy zastanowić się nad efektywną realizacją GA.

Dokładny opis GA, wraz z wieloma propozycjami syntezy logicznej, został przedstawiony w [6, 7, 8, 9, 10]. Jednakże tylko nieliczne, opisane np. w [11] rozwiązania, wykorzystują architekturę FPGA i realizację z zastosowaniem pamięci wbudowanych.

Najprostszą metodą realizacji funkcji opisywanego generatora jest zastosowanie wbudowanych pamięci EMB będących integralną częścią każdego współczesnego układu FPGA. Jednakże wielkość w pełni określonej funkcji boolowskiej, która miałaby być umieszczona w blokach EMB dla przeciętnych długości wektora wejściowego, przekracza dostępny rozmiar pamięci współczesnych układów programowalnych [1, 2].

Niniejsza praca opisuje schemat realizacji dekodera adresu zapewniający efektywną realizację funkcji GA. Prezentowana metoda oparta jest na podziale funkcji boolowskiej względem jej argumentów na części oraz implementacji poszczególnych fragmentów za pomocą układów sekwencyjnych. Do realizacji automatów zastosowano specjalizowane oprogramowanie uniwersyteckie omówione w [1].

W artykule omówiono kolejno: informacje podstawowe dotyczące GA, następnie implementację GA, aplikację dekompozycji funkcji na automaty, schemat implementacji układów sekwencyjnych w FPGA oraz wyniki przeprowadzonych badań.

2. Generator adresu

Rozważmy zbiór k wektorów binarnych o długości n bitów – wektorów rejestrowanych. Dla każdego z nich przydzielamy jednoznacznie wartość całkowitą ze zbioru liczb naturalnych od 1 do k. Funkcja generatora adresu zwraca całkowity identyfikator, jeśli wektor wejściowy odpowiada jednemu z wektorów rejestrowanych, lub 0, gdy zadanego wektora nie ma w tablicy [10, 11].

Przykład. W pełni określona funkcja boolowska z tab.1 zawiera 5 wektorów rejestrowanych.

Tab. 1. Przykład funkcji boolowskiej
Tab. 1. Example of Boolean function

	0	0	0	0	1	1	1	1	x_2
	0	0	1	1	0	0	1	1	x_1
	0	1	0	1	0	1	0	1	x_0
00	0	0	0	0	**1**	0	**2**	0	
01	0	0	0	**3**	0	0	0	0	
10	0	0	0	0	0	0	0	0	
11	0	0	0	0	**4**	**5**	0	0	
x_4x_3									

3. Implementacja Generatora Adresów

Zadanie zaprojektowania generatora o 40 wejściach jest zadaniem trudnym [4, 5]. Ze względu na fakt iż w pełni określona funkcja boolowska zapisana w standardzie *PLA* typu .type fr [3] zawierała by około 10^{12} wierszy, plik tej wielkości nie byłby możliwy do przechowania na standardowej wielkości dysku, nie wspominając o przetwarzaniu takiego pliku.

3.1. Dedykowana metoda implementacji

Opisana w [11] przez T. Sasao i M. Matsuura metoda *super hybrid* implementuje funkcję GA z użyciem pamięci haszującej oraz reprogramowalnych struktur PLA. W funkcji GA, liczba wektorów rejestrowanych k, jest znacznie mniejsza od 2^n – liczby wszystkich wektorów wejściowych. Biorąc pod uwagę zbiór liniowych funkcji haszujących transformujących zbiór 2^n elementów na 2^p elementów, gdzie $2^p \geq k + 1$, T. Sasao i M. Matsuura przez użycie funkcji liniowej $y_i = x_i \oplus g_i(X)$, $(i = 1, 2, \ldots, p)$, potrafią zredukować przestrzeń 2^n-elementową na przestrzeń 2^p-elementową. Używając tego rozwiązania implementują oni funkcję GA przy użyciu p-wejściowej pamięci, zamiast n-wejściowej.

3.2. Nowa metoda

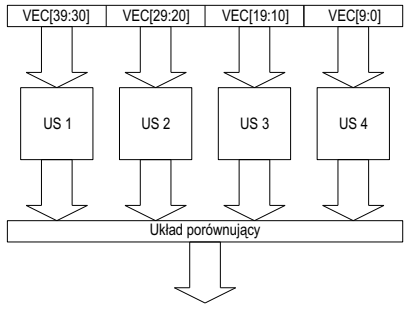
Nowy sposób implementacji GA, polega na podziale funkcji boolowskiej względem argumentów na t części, które są realizowane za pomocą układów sekwencyjnych. Każdy automat dostaje na wejście n/t bitów wektora wejściowego GA. Na wyjściach automatów pojawiają się identyfikatory (w postaci liczby całkowitej) tych wektorów, które pasują do odpowiednich n/t bitów.

Przykład. Rozpatrzmy funkcję GA o 5 wejściach oraz automat o 3 wejściach. Niech lista wektorów rejestrowanych GA będzie następująca:

1	00100	1
2	00110	1
3	01011	1
4	11100	1
5	11101	1

Na wejście podawane są 3 najbardziej znaczące bity wektora wejściowego GA. Stąd, kiedy podamy na wejście automatu 001 zwraca on indeksy 1, 2, natomiast, jeśli podamy 111 na wyjściu automatu pojawią się wartości 4 i 5.

W wyniku dekompozycji na t układów sekwencyjnych, dla każdego wektora wejściowego dostajemy t zbiorów indeksów. Jeśli którykolwiek z indeksów pojawi się we wszystkich t zbiorach oznacza to, że jest to wektor rejestrowany o danym identyfikatorze. Jeśli nie znaleziono indeksu występującego we wszystkich zbiorach, oznacza to, że dany wektor nie występuje w zbiorze wektorów rejestrowanych. Przykładowy schemat blokowy dla $n = 40$ i $t = 4$ został przedstawiony na rys. 1.



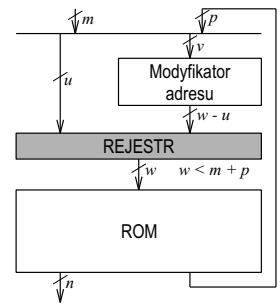
Rys. 1. Schemat realizacji GA z użyciem automatów
Fig. 1. Scheme of Adress Generator implemented using FSMs

W związku z powyższym, wpływ optymalnej realizacji składowych automatów na realizację całego układu będzie znaczący.

Każdy automat $A = \langle S, V, \delta, Y, \lambda \rangle$ o skończonej liczbie stanów, gdzie: S – zbiór stanów automatu, V – zbiór symboli wejściowych, δ – funkcja transformacji stanów, Y – zbiór symboli wyjściowych, λ – funkcja wyjść, może zostać zrealizowany w sposób przedstawiony na rys. 2, przy użyciu modyfikatora adresu.

Źródłem złożoności opisywanego modyfikatora adresu są zmienne adresowe zależne od więcej niż jednej zmiennej wejściowej lub stanu automatu. Dlatego ważnym aspektem staje się wybór odpowiedniego kodowania wejść i symboli stanów układów sekwencyjnych.

Stosownie dobrana strategia dekompozycji automatu pozwala osiągnąć oczekiwaną redukcję rozmiaru pamięci kosztem włączenia dodatkowych komórek logicznych do realizacji modyfikatora adresu. Daje to możliwość implementacji dużych automatów z zastosowaniem pamięci wbudowanych.



Rys. 2. Schemat implementacji automatu
Fig. 2. Implementation of FSM

W prezentowanej realizacji automatów logika kombinacyjna została podzielona na dwie części. Pierwsza część została zaimplementowana z użyciem pamięci wbudowanych. Pamięci konfiguruje się jako ROM, a ich zawartość jest ustalana podczas projektowania. Druga część - modyfikator adresu - został użyty do redukcji liczby słów zastosowanej pamięci. Modyfikator adresów został zaimplementowany z użyciem LUT.

Koncepcja modyfikatora adresu daje możliwości znacznej redukcji używanej pamięci i znacząco poprawia jakość realizacji automatów w architekturze FPGA. Dlatego, zastosowanie w projektowaniu GA nowatorskiej metody syntezy układów sekwencyjnych [1, 2] pozwala na osiągnięcie bardzo dobrych rezultatów.

Układ porównujący składa się z czterech kolejek FIFO zbudowanych z rejestrów. Jego cykl działania jest następujący: po otrzymaniu sygnału startowego, pobiera on z każdej kolejki kolejne wartości wystawiane na wyjściu przez automaty. Po zakończeniu tego procesu następuje szukanie wspólnej wartości we wszystkich czterech ciągach.

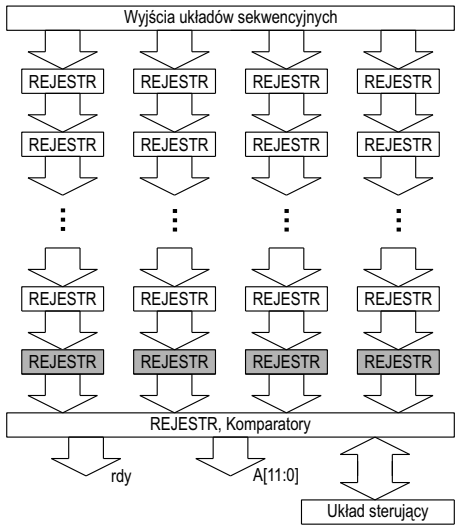
Wartości otrzymywane z automatów są podawane w kolejności rosnącej, co ułatwia proces porównania. W danym cyklu zegara z czterech wartości bieżących wybierana jest wartość największa, dana kolejka pozostaje bez zmian, a pozostałe są przesuwane o 1. Dzieje się tak do momentu ustalenia się jednakowej wartości we wszystkich 4 kolejkach, lub do momentu, aż na wyjściu będzie wartość 0. Gdy już znane jest rozwiązanie – indeks lub wartość 0, sygnał *rdy* przechodzi w stan wysoki, a na wyjściu układu porównującego pojawia się wynik. Schemat blokowy układu został przedstawiony na rys. 3.

4. Pakiet programowy

Dla celów badań został stworzony prototypowy pakiet programowy umożliwiający syntezę opisywanego generatora adresów.

W skład tego pakietu wchodzi program generujący listę n bitowych pseudolosowych wektorów o długości k. Kolejny program wydziela wybraną część wektora wejściowego i tworzy dwie tablice – pierwsza zawiera indeksy dla każdego fragmentu wekto-

ra, pod którym można go odnaleźć, a druga – indeksy wektorów oraz indeksy wektorów następnych, gdzie wybrany fragment się powtarza. Trzecim narzędziem programowym jest generator kodu w języku *Verilog*. Buduje on kod opisujący układ na podstawie stworzonych wcześniej plików.



Rys. 3. Schemat układu porównującego
Fig. 3. Comparing circuit scheme

Aby poprawić implementacje automatów w architekturze FPGA zastosowano oprogramowanie uniwersyteckie FSM*dec* [1].

5. Wyniki eksperymentalne

Przeprowadzono następujące testy: dla 2000 wektorów rejestrowanych, dla 4000 wektorów rejestrowanych oraz dla 9100 wektorów rejestrowanych. Testowany GA posiada 40 bitowe wejście. W Tab. 2 przedstawiono wyniki realizacji.

Realizacja generatora adresu jest charakteryzowana przez liczbę komórek logicznych i liczbę wbudowanych bloków pamięci EMB. W tablicy porównano otrzymane wyniki z wynikami zawartymi w pracy T. Sasao i M. Matsuura. Proponowana w niniejszym artykule realizacja używa podobnych zasobów pamięciowych, ale znacznie mniej komórek logicznych. W ostatniej kolumnie przedstawiono procentową redukcję zasobów sprzętowych potrzebnych na realizację obliczoną ze wzoru:

$$redukcja = (a - b) / a \cdot 100\%.$$

6. Podsumowanie

Otrzymane rezultaty dowodzą dużej efektywności przedstawionej metody. Okazują się być dużo lepsze od wyników uzyskanych za pomocą innych metod pod względem zajętości zasobów sprzętowych układów FPGA. Proponowane rozwiązanie cechuje się dwiema głównymi zaletami:

- ilość potrzebnych zasobów nie zależy od wartości wektorów, tylko od ich liczby. W rozwiązaniu opisanym w [11] wartości wektorów mogą wpływać niekorzystnie na ilość potrzebnych zasobów.
- nie ma potrzeby liczenia funkcji haszującej aby stworzyć generator adresów dla danego zbioru wektorów rejestrowanych. Proponowane rozwiązanie daje możliwość stworzenia GA dla dowolnego zbioru wektorów.

Wielowejściowe funkcje logiczne charakteryzujące się dużą dysproporcją mogą być realizowane przy pomocy opracowanego oprogramowania, które tworzy plik funkcji GA w języku *Verilog*.

Tab. 2. Wyniki eksperymentalne
Tab. 2. Experimental results

	Realizacja T.Sasao	Realizacja na automatach	Redukcja
	a	*b*	
Liczba wektorów rejestrowanych AG	1730	2000	
Długość wektora wejściowego AG	40	40	
Długość wyjścia AG	11	11	
Pamięci M4K	32	36	-14,0%
Pamięci M512	0	4	
Komórki logiczne	2426	132	94,6%
Liczba wektorów rejestrowanych AG	3366	4000	
Długość wektora wejściowego AG	40	40	
Długość wyjścia AG	12	12	
Pamięci M4K	64	60	5,5%
Pamięci M512	0	4	
Komórki logiczne	4889	141	97,1%
Liczba wektorów rejestrowanych AG	4705	9100	
Długość wektora wejściowego AG	40	40	
Długość wyjścia AG	13	13	
Pamięci M4K	121	120	0,4%
Pamięci M512	0	4	
Komórki logiczne	3560	167	95,3%

7. Literatura

[1] G. Borowik, Finite State Machines Synthesis for FPGA Structures with Embedded Memory Blocks (in Polish), PhD Dissertation, Faculty of Electronics and Information Technology, WUT, 2007.
[2] G. Borowik, B. Falkowski and T. Luba, Cost-Efficient Synthesis for Sequential Circuits Implemented Using Embedded Memory Blocks of FPGA's, Proc. of 10th IEEE Workshop on DDECS, pp. 99-104, 2007.
[3] R. Brayton, G. Hachtel, C. McMullen and A. Sangiovanni-Vincentelli, Logic minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Boston 1985.
[4] J. A. Brzozowski and T. Luba, Decomposition of Boolean Functions Specified by Cubes, Journal of Multi-Valued Logic & Soft Computing, Old City Publishing Inc., Philadelphia 2003, Vol. 9, pp. 377-417.
[5] J. Cong and K. Yan, Synthesis for FPGAs with embedded memory blocks, Proc. of the 2000 ACM/SIGDA 8th International Symposium on FPGAs, pp. 75-82, ACM Press NY, 2000, Monterey, California.
[6] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, Longest prefix matching using Bloom filters, ACM SIGCOMM' 03, August 25-29, 2003, Karlsruhe, Germany.
[7] J. Ditmar, K. Torkelsson, and A. Jantsch, A reconfigurable FPGA-based content addressable memory for internet protocol characterization, Proc. FPL2000, LNCS 1896, Springer, 2000, pp. 19-28.
[8] G. Nilsen, J. Torresen and O. Sorasen, A variable wordwidth content addressable memory for fast string matching, NorChip 2004, pp. 214-217.
[9] K. Pagiamtzis and A. Sheikholeslami, Content-addressable memory (CAM) circuits and architectures: A tutorial and survey, IEEE Journal of Solid-State Circuits, vol. 41, no. 3, pp. 712-727, March 2006.
[10] T. Sasao, Design methods for multiple-valued input address generators, (invited paper) International Symposium on Multiple-Valued Logic, Singapore, May 2006.
[11] T. Sasao, M. Matsuura, An Implementation of an Address Generator Using Hash Memories, 2007 IEEE.

Artykuł recenzowany