Krzysztof MACHACZEK [2], Paweł RUSSEK [1,2], Ernest JAMRO [1,2], Kazimierz WIATR [1,2]
[1] AKADEMIA GÓRNICZO-HUTNICZA, KATEDRA ELEKTRONIKI
[2] AKADEMIA GÓRNICZO-HUTNICZA, ACK CYFRONET

Realizacja szybkiego wyszukiwania wzorców w układach FPGA

Inż. Krzysztof MACHACZEK

Studiował w Akademii Górniczo-Hutniczej w Krakowie na kierunku Elektronika i Telekomunikacja. Studia ukończył z tytułem inżyniera w 2007 roku. Zajmuje się projektowaniem systemów cyfrowych z wykorzystaniem układów FPGA.

e-mail: Krzysztof.Machaczek@cyfronet.krakow.pl

Dr inż. Ernest JAMRO

Ukończył studia na AGH na kierunku Elektronika oraz na University of Huddersfield (UK) na kierunku Elektronika i Telekomunikacja. Obronił pracę doktorską w 2001 roku na AGH na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki. Aktualnie jest adiunktem w Katedrze Elektroniki na AGH. Jego zainteresowania naukowe to sprzętowa akceleracja obliczeń, niskopoziomowe przetwarzanie obrazów, sieci neuronowe.

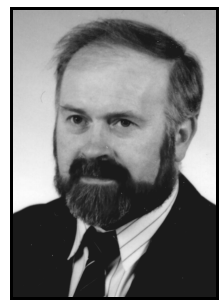
e-mail: jamro@agh.edu.pl

Dr inż. Paweł RUSSEK

Ukończył studia na wydziale Elektrotechniki, Automatyki i Elektroniki AGH Kraków (1994), dr nauk technicznych (2003). Jest adiunktem w Katedrze Elektroniki AGH. Prowadzone prace badawcze dotyczą sprzętowej akceleracji obliczeń przy pomocy architektur dedykowanych, zagadnień realizacji obliczeń przy użyciu rekonfigurowalnego sprzętu oraz problemu przyśpieszania obliczeń w środowisku komputerów dużej mocy obliczeniowej.

e-mail: russek@agh.edu.pl

Prof. dr hab. inż. Kazimierz WIATR



Studia AGH Kraków (1980), dr nauk technicznych (1987), dr habilitowany (1999) i profesor (2002). Profesor zwyczajny na AGH w Krakowie oraz Dyrektor ACK Cyfronet AGH. Prowadzone prace badawcze dotyczą systemów wizyjnych, systemów wieloprocesorowych, rekonfigurowalnych systemów obliczeniowych i sprzętowych metod akceleracji obliczeń.

e-mail: wiatr@agh.edu.pl

Streszczenie

Niniejszy artykuł prezentuje sprzętową realizację filtracji Bloom'a w układach FPGA. Implementacja ta służy do szybkiego wyszukiwania wielu wzorców binarnych bądź znakowych w dużym zbiorze danych. Podczas filtracji Bloom'a sekwencyjnie podawane dane wejściowe są haszowane, a następnie obliczony hash jest porównywany w pamięci z podanymi wzorcami. Proces haszowania ciągu wejściowego jak i porównywanie z wzorcami odbywa się potokowo. Zaproponowana implementacja równoległa w jednym takcie zegara porównuje 16-bajtowy fragment ciągu wejściowego ze wszystkimi wzorcami. Przy uzyskanej szybkości zegara 100 MHz, szybkość przeszukiwania danych wejściowych wynosi 1.6 GB/s.

Słowa kluczowe: FPGA, Filtr Bloom'a, wyszukiwanie wzorców.

FPGA implementation of fast patterns search

Abstract

This paper presents FPGAs implementation of Bloom filters. Consequently a great number of both binary and text patters can be quickly searched for in a large database. For Bloom filters, sequencially fed input data are hashed, then addresses a special memory which output data indicates whether the input string is or is not one of paterns. The whole implementation is strongly pipelined and parallel. Consequently, 16-byte of input data are processed in a single clock cycle at clock frequency 100 MHz, therefore the search throughput is 1.6 GB/s.

Keywords: FPGAs, Bloom Filter, patterns search.

1. Wstęp

Szybkie przeszukiwanie bazy danych w celu znalezienia wzorców znakowych lub binarnych wymaga znacznego czasu obliczeniowego procesora. Dlatego istnieje wiele różnych implementacji sprzętowych przyspieszających taką operację. Większość implementacji jest używana w systemach detekcji włamań w sieci - NIDS (ang. network intrusion detection system). W systemach tych zakłada się, że wyszukiwane wzorce są stałe – aktualizacja wzorców przebiega relatywnie rzadko i wymaga nowej implementacji w układzie FPGA. Dlatego w tych systemach stosuje się zaawansowane metody optymalizacji np. kodowanie znaków 8–256 czy dzielenie wspólnej podstruktury [1, 2]. Alternatywną metodą wyszukiwania ciągów jest użycie filtrów Bloom'a [3]. Zaproponowany w tym artykule algorytm bazuje właśnie na tej metodzie. Istnieje wiele różnych implementacji filtrów Bloom'a w układach FPGA [4, 5, 6]. Pierwsza z nich [4] rozważa implementacje, dla której w każdym takcie zegara pobierany jest tylko jeden bajt wejściowy, równolegle natomiast wykonywany jest algorytm dla różnych długości filtru. Szybkość przetwarzania danych wynosi 600 Mb/s i jest ona wykorzystywana do przeszukiwania ciągów znaków w sieci. D. C. Suresh, et. al. [5] położyli nacisk na automatyczną generację kodu VHDL przy pomocy programu w języku C. Narzędzie to, jest wykorzystywane do wyszukiwania wirusów, a szybkość przetwarzania wynosi 18.6 Gb/s dla układu FPGA Virtex XC2V8000. Szybkość ta dotyczy tylko jednej długości słowa. Inne rozwiązanie zostało zaprezentowane przez A. Jacobs'a et. al. [6]. Służy ono do określenia języka, w którym jest napisany dokument tekstowy. Dlategozastosowano wiele niezależnych filtrów Bloom'a, po jednym dla każdego języka. Pamięć filtru Bloom'a jest odpowiednio zaprogramowana podstawowymi słowami z danego języka. Teoretyczna szybkość sprawdzania dokumentu jest 1.4 GB/s, ale niestety z powodu przepustowości łącza jest ona ograniczona do 500 MB/s. Aby osiągnąć taką szybkość przepustowość wynosi 8 bajtów na jeden takt zegara (194 MHz). Niniejszy sprzętowa realizacja przedstawiona w artykule wyróżnia się na tle wspomnianych dużą przepustowością (teoretycznie 1.6 GB/s) oraz możliwością szybkiej zmiany wyszukiwanych wzorców.

2. Zaproponowany algorytm

2.1. Haszowanie

Podstawową operacją wykonywaną podczas wyszukiwania wzorców z użyciem filtru Bloom'a jest operacja haszowania. Polega ona na przekształceniu długiego ciągu znakowego (lub bitowego) o liczbie bitów *w,* w mniejszy ciąg bitowy o długości *h* bitów.

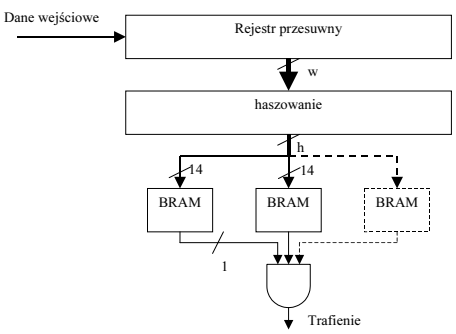
Warto szerzej opisać dwie negatywne cechy operacji haszowania. Po pierwsze przeszukiwanie z haszowaniem nie jest deterministyczne, tj. istnieje pewne znikome prawdopodobieństwo P_{error} błędnego wskazania ciągu wejściowego który nie jest wzorcem:

$$P_{error} \approx 2^{-h} \quad (1)$$

gdzie *h*- szerokość bitowa argumentu wyjściowego po operacji haszowania.

Jeżeli algorytm wyszukiwania ma być bezbłędny, to wynik wyszukiwania algorytmem haszującym musi być sprawdzany przez inny algorytm deterministyczny. Warto podkreślić, że przeszukiwanie z haszowaniem może wskazać błędny danej wejściowej, która nie jest wzorcem, natomiast sytuacja odwrotna nie jest możliwa, czyli wzorzec jest zawsze wskazywany.

W przypadku szukania wielu różnych wzorców S_i równocześnie porównywanie każdego z nich z wynikiem haszowania wymaga relatywnie dużo zasobów. Dlatego lepszym rozwiązaniem jest zastosowanie pamięci LUT, która odpowiednio zaprogramowana wskazywałaby czy dany wynik haszowania wskazuje szukaną daną. Niestety aby prawdopodobieństwo mylnego wskazania P_{error} było małe, szerokość bitowa *h* magistrali adresowej pamięci LUT powinna być względnie duża, dlatego użycie pojedynczej pamięci LUT jest nieefektywne. Lepszym rozwiązaniem, zaproponowanym przez Bloom'a [3], jest podzielenie tej pamięci na *k* niezależnych mniejszych pamięci oraz równoczesne sprawdzanie stanu wszystkich tych pamięci (bramka AND łącząca wyjścia tych pamięci). Jeżeli wszystkie pamięci wskażą zgodność to nastąpi trafienie. Ponieważ w zastosowanych układach Virtex 4 pamięć BRAM jest wielkości 16k×1 (14-bitowy adres) to założono, że szerokość bitowa *h* wyjścia operacji haszowania będzie równa 2×14=28 lub 3×14=42 bity, czyli zostaną użyte k=2 lub 3 pamięci BRAM. Schemat blokowy modułu przedstawiono na rys. 1.



Rys. 1. Schemat blokowy modułu przeszukującego
Fig. 1. Block diagram of the search module

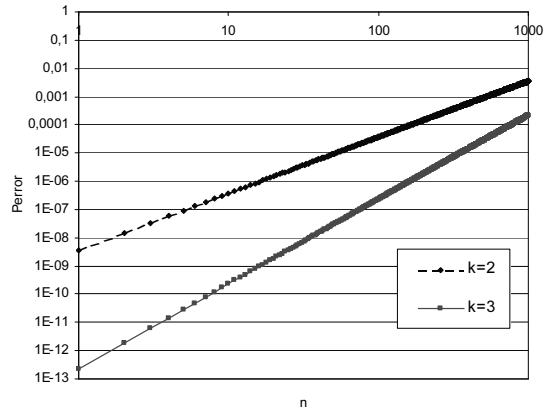
Podzielenie pamięci LUT na *k* niezależne pamięci oraz dodanie bramki AND powoduje, że prawdopodobieństwo mylnego wskazania wzrasta i wynosi ono [3]:

$$P_{error} = (1-(1-\frac{1}{m})^n)^k \approx (1-e^{-n/m})^k \quad (2)$$

gdzie *n*- liczba szukanych wzorców, *m*- wielkość pamięci: $m=2^a$, *a*- liczba linii adresowych pamięci.

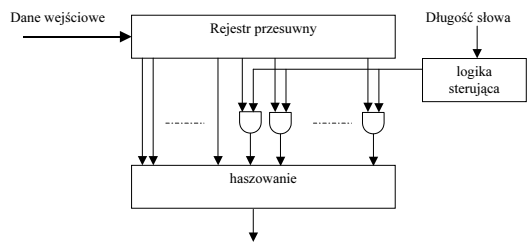
Fundamentalną zaletą powyższej architektury jest to, że definiowanie szukanych wzorców nie wymaga przeprojektowywania oraz rekonfiguracji układu FPGA. Z punktu widzenia użytkownika proces definiowanie wyszukiwanych wzorców polega na ich podaniu na wejście prezentowanego modułu oraz ustawieniu odpowiedniego sygnału sterującego wprowadzającego układu w tryb programowania.

Pewną niedogodnością wyszukiwania wzorców przy użyciu algorytmu haszowania jest to, że umożliwia ono wyszukiwanie wzorców tylko dla jednej określonej długości. Rozwiązaniem tego problemu jest skracanie długości wzorców do najkrótszej występującej długości. Lepszym rozwiązaniem jest uruchomienie procedury wyszukiwania dla różnych długości wzorców. Podstawowym problemem takiego rozwiązania jest to, że projektant musi zaprojektować i zaimplementować niezależne konfiguracje układu FPGA dla różnych długości wzorców.



Rys. 2. Prawdopodobieństwo mylnego wskazania dla różnej liczby wyszukiwanych wzorców *n* dla liczby pamięci LUT równej *k*= 2 oraz *k*= 3 i szerokości adresowej *a*=14
Fig. 2. Probability of searching errors for different number of patterns *n* and the number of BRAMs *k*=2 or k=3, BRAM address width *a= 14*

Zaproponowanym przez autorów rozwiązaniem powyższego problemów jest dodanie dodatkowych bramek AND przed modułem haszującym. Bramki AND są odpowiednio sterowane w zależności od wymaganej długości wzorców, co zostało przedstawione na rys. 3. W konsekwencji zmiana szerokości szukanego wzorca nie wymaga rekonfiguracji układu FPGA lecz wymaga tylko odpowiedniego sterowania bramkami AND. Wadą tego rozwiązania jest to, że powyższy układ zajmuje zasoby sprzętowe w przybliżeniu takie jak dla maksymalnej wymaganej szerokości wzorca. Dlatego ciągle zalecane jest posiadanie różnych konfiguracji układu FPGA dla różnych długości wzorców, jednak liczba tych kombinacji jest zdecydowanie niższa.



Rys. 3. Schemat układu z programowalną szerokością wyszukiwanego wzorca
Fig. 3. Block diagram for programmable patern length

2.2. Równoległa praca modułów

Implementacja powyższego modułu umożliwiłaby wyszukiwanie bardzo wielu różnych wzorców z szybkością 1 znaku ciągu wejściowego na takt zegara. Znak może być zdefiniowany jako 8 lub też 16-bitów. W dalszej części tego artykułu zakłada się że znak jest 8-bitowy. Szybkość działania jest niezależna od liczby wyszukiwanych równocześnie wzorców, wzrost liczby wzorców powoduje tylko większe prawdopodobieństwo mylnego wskazania.

Aby zwiększyć szybkość wyszukiwania zastosowano architekturę równoległą składającą się z maksymalnie 16 jednostek równoległych. Każda z nich wykonuje operacje przeszukiwania na kolejnych przesuniętych o jeden znak próbkach wejściowych. W konsekwencji w jednym takcie zegara moduł równoległy jest w stanie przetworzyć 16×8= 128 bitów danych wejściowych o określonej długości słowa, co daje szybkość przeszukiwania równą 1.6 GB/s dla zegara o częstotliwości 100 MHz.

Wyżej wspomniana szybkość przeszukiwania dotyczy tylko pojedynczej szerokości wzorca. Dlatego w niektórych przypadkach konieczne byłoby sekwencyjne przeszukiwanie dla różnych szerokości wzorców, co wydłużałoby całkowity czas wyszukiwania. W konsekwencji zastosowano następny stopień zrównoleglenia, dla którego w jednym takcie zegara wyszukiwanych jest wiele różnych długości wzorców. Wszystkie te operacje wykonywane są na 16

bajtach równocześnie. Warto tutaj wspomnieć, że szerokość każdego przeszukiwania jest ustawiana za pomocą bramek AND (zob. rys. 3).

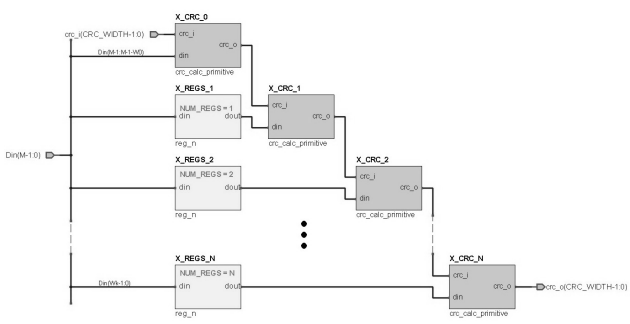
3. Implementacja

Zaprojektowany moduł jest opisany w języku opisu sprzętu VHDL oraz silnie sparametryzowany przy użyciu słowa kluczowego *generic*. Poniżej podane są podstawowe parametry modułu:

- SDIN_WIDTH – szerokość magistrali danych wejściowych przetwarzanych w jednym takcie zegara. Parametr ten określa z reguły szerokość bitową dostępu do pamięci zewnętrznej. Dodatkowo ten parametr oraz BITS_PER_CHAR określają liczbę równolegle pracujących jednostek równą SDIN_WIDTH / BITS_PER_CHAR.
- NUM_MATCHERS – liczba ta określa dla ilu różnych długości wzorców przeprowadza się proces przeszukiwania. Warto wspomnieć, że ten parametr określa następny poziom zrównoleglenia w porównaniu z parametrem SDIN_WIDTH / BITS_PER_CHAR. Dlatego całkowita liczba równolegle pracujących jednostek N jest określona liczbą: N= NUM_MATCHERS × SDIN_WIDTH / BITS_PER_CHAR.
- WORD_LENGTHS – maksymalna długości szukanego słowa wzorcowego w bitach, może być ona różna dla każdej jednostki równoległej 1.. NUM_MATCHERS.
- BITS_PER_CHAR – liczba bitów na znak. Niektóre słowniki traktują jeden znak jako 8-bitów, dla innych jeden znak to 16-bitów.
- h– zastosowane wielomianu haszującego rzędu 28 lub 42-bitów.
- PROGR_WORD_LENGTH – parametr określający czy dana jednostka ma możliwość programowania długości słowa za pomocą dodatkowych bramek AND (zob. rys. 3), czy wyszukuje słów o jednej ustalonej na stałe długości.

Moduł haszujący

Dla dużej liczby bitów danej wejściowej wykonywanie operacji haszowania w jednym takcie zegara jest niemożliwe ze względu na duże czasy propagacji. Dlatego zastosowano architekturę potokową, dla której operacje haszowania wykonywane są na 64-bitach lub też 128-bitach (dwie różne wersje) w jednym takcie zegara. Schemat blokowy całego modułu haszującego został przedstawiony na rys. 4.



Rys. 4. Potokowa architektura modułu haszującego
Fig. 4. Pipeline architecutre of the hashing module

Jednym z podstawowych parametrów jest wybór jednego z dwóch wielomianów haszujących. Wybór wielomianu rzędu 28 ($h=28$), pociąga za sobą użycie 2 pamięci BRAM (16k×1) ($k=2$) na pojedynczy moduł. Natomiast wybór wielomianu rzędu 42 wymaga 3 pamięci BRAM ($k=3$). Jak pokazują wyniki implementacji jednym z ograniczeń poziomu zrównoleglenia jest liczba dostępnych pamięci BRAM, z drugie strony zastosowanie wielomianu rzędu 28 przy dużej liczbie wzorców n powoduje gwałtowny wzrost mylnych wskazań (zob. rys. 2).

Pełną implementację całego modułu przeprowadzono dla płyty RASC firmy SGI [7]. Przykład implementacji całego układu wraz z interfejsami do pamięci i podłączeniem do całego systemu (tzw. CoreServices) o poniższych parametrach jest podany w tab. 1.

- SDIN_WIDTH = 128;
- NUM_MATCHERS = 6;
- WORD_LENGTHS = „256, 256, 192, 192, 128, 128”;
- BITS_PER_CHAR = 8;
- CRC_WIDTH = 42;
- PROGR_WORD_LENGTH = „1, 1, 1, 1, 1, 1”;

Tab. 1. Zajmowane zasoby przez CoreServices i matcher o konfiguracji przedstawionej powyżej (wyniki końcowe z raportu acs_top_map.mrp)
Tab. 1. FPGAs resources occupied by CoreServices and the matcher - parameters are given above

Zasoby	*Użytych szt.*	*Użytych zasobów układu xc4vlx200*
Bloki SLICEs	70015	78%
Przerzutniki bloków SLICEs	47065	26%
Bloki LUT4	109127	61%
FIFO16/RAMB16	311	92%

Moduł ten umożliwia równoległe wyszukiwanie wzorców o 6 różnych długościach o przepustowości 16 B / takt zegara (100 MHz). Otrzymana częstotliwość 100 MHz nie jest najwyższą częstotliwością pracy prezentowanego układu przeszukującego, niestety platforma RASC nie umożliwia w prosty sposób ustawienia częstotliwości pośrednich pomiędzy 100 MHz a 200 MHz.

4. Podsumowanie

Niniejszy artykuł przedstawia wynik implementacji w układach FPGA szybkiego przeszukiwania ciągów. Wybrano algorytm – filtr Bloom’a który jest preferowany w przypadku równoczesnego wyszukiwania przy bardzo dużej liczbie wzorców. Osiągnięto również bardzo duży stopień zrównoleglenia dzięki czemu prezentowana implementacja jest najszybszą znaną autorom implementacja filtrów Bloom’a w układach FPGA. Główny nacisk niniejszej implementacji został położony na łatwy i szybki sposób zmiany wyszukiwanych wzorców bez konieczności ponownej implementacji i konfiguracji układu FPGA. Dlatego możliwa jest prosta zmiana długości obliczanej funkcji haszujacej za pomoca dodatkowych bramek AND, oraz szybkie zerowanie pamięci – drugi port pamięci dwuportowej o szerszej magistrali danych jest używany do kasowania pamięci.

Praca naukowa finansowana ze środków na naukę jako projekt badawczy.

5. Literatura

[1] Cheng-Hung Lin, Chih-Tsun Huang, Chang-Ping Jiang, Shih-Chieh Chang, Optimization of Pattern Matching Circuits for Regular Expression on FPGA, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 15, NO. 12, pp. 1303-1310, Dec. 2007
[2] I. Sourdis, J. Bispo, et. al. Regular Expression Matching in Reconfigurable Hardware, Int. Journal of Signal Processing Systems for Signal, Image, and Video Technology (Springer), 2007.
[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422–426, July 1970.
[4] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, J. Lockwood, Deep Packet Inspection Using Parallel Bloom Filters , (HOTI'03): Hot Interconnects 11: Stanford, CA, 8/03, 2003.
[5] D. C. Suresh , Z. Guo, B. Buyukkurt, W.A. Najjar Automatic Compilation Framework for Bloom Filter Based Intrusion Detection, Automatic Compilation Framework for Bloom Filter Based Intrusion Detection. ARC 2006 - Lecture Notes in Computer Science, Volume 3985: pp. 413-418, 2006.
[6] A. Jacob, M. Gokhale, Language classification using n-grams accelerated by FPGA-based Bloom filters, Conference on High Performance Networking and Computing, Reno, Nevada pp. 31-37, November 11 - 11, 2007
[7] Silicon Graphics, Inc., Reconfigurable Application-Specific Computing User’s Guide, Ver. 004, Mar 2006, SGI

Artykuł recenzowany