Alexander BARKALOV, Larysa TITARENKO, **Jacek BIEGANOWSKI**
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI
ul. Licealna 9, 65-417 Zielona Góra

Implementacja sprzętowa algorytmu MD5 w układach FPGA z użyciem mikroprogramowanego układu sterującego

Prof. dr hab. inż. Alexander BARKALOV

Prof. Alexander A. Barkalov w latach 1976-1996 był pracownikiem dydaktycznym w Instytucie Informatyki Narodowej Politechniki Donieckiej. Współpracował aktywnie z Instytutem Cybernetyki im. V.M. Glushkova w Kijowie, gdzie uzyskał tytuł doktora habilitowanego ze specjalnością informatyka. W latach 1996-2003 pracował jako profesor w Instytucie Informatyki Narodowej Politechniki Donieckiej. Od 2004 pracuje jako profesor na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego.

e-mail: a.barkalov@iie.uz.zgora.pl

Dr hab. inż. Larysa TITARENKO

Dr hab. Larysa Titarenko w 2004 roku obroniła rozprawę habilitacyjną i uzyskała tytuł doktora habilitowanego ze specjalnością telekomunikacja. W latach 2004-2005 pracowała jako profesor w Narodowym Uniwersytecie Radioelektroniki w Charkowie. Od 2005 pracuje na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego.

e-mail: l.titarenko@iie.uz.zgora.pl

Streszczenie

W artykule przedstawiona została koncepcja implementacji sprzętowej algorytmu MD5 z wykorzystaniem mikroprogramowanego układu sterującego. Cechą charakterystyczną rozwiązania jest wykorzystanie osadzonych bloków pamięci do realizacji układu sterującego. Przedstawione rozwiązanie jest przeznaczone przede wszystkim do realizacji w układach FPGA. W artykule przedstawione zostały wyniki syntezy kilku wybranych struktur układów mikroprogramowanych. Otrzymane wyniki zostały porównane do typowej realizacji w postaci automatu Moore'a.

Słowa kluczowe: mikroprogramowany układ sterujący, osadzony blok pamięci, algorytm MD5, FPGA.

Hardware implementation of MD5 algorithm in FPGAs using compositional microprogram control unit

Abstract

The paper presents an example of application of Compositional Microprogram Control Unit (CMCU) to hardware implementation of MD5 algorithm. The MD5 algorithm is a widely used hash function with a 128-bit hash value. MD5 is used in many security applications, for example to hash passwords in FreeBSD operating system [14]. MD5 is also commonly used to check the integrity of files. MD5 was designed by Ron Rivest in 1991 [10]. Other similar algorithms are SHA [7] and RIPEMD [6]. The hardware implementation of MD5 in FPGAs is usually based on embedded memory blocks (EMB) because the algorithm uses a lot of constants during calculations [8]. In the paper the authors present an alternative solution in which constants are generated by CMCU (Fig. 3) circuit. The CMCU is also based on EMB. It can generate constants for MD5 and also signals for other tasks. The research results show that CMCU requires less hardware amount when compared to traditional Moore FSM (Tab. 1). The results were obtained using Xilinx ISE 12.1 and Xilinx Spartan-3 (xc3s50-5pq208) [13]. The models of control units were generated by the authors' software.

Keywords: MD5, compositional microprogram control unit, field-programmable gate arrays, embedded-memory block.

Mgr inż. Jacek BIEGANOWSKI

Ukończył w roku 2003 studia na kierunku informatyka o specjalności inżynieria komputerowa. Od 2004 roku pracuje w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego na stanowisku asystenta. Jego zainteresowania związane są z programowaniem, systemami operacyjnymi, techniką cyfrową oraz algorytmami ewolucyjnymi.

e-mail: j.bieganowski@iie.uz.zgora.pl

1. Wprowadzenie

Algorytm MD5 jest stosowany do wyznaczania 128-bitowego skrótu na podstawie dowolnego ciągu danych. Algorytm MD5 jest stosowany powszechnie w różnego rodzaju aplikacjach związanych z bezpieczeństwem, na przykład do kodowania haseł [14] czy sprawdzania poprawności przesyłania plików. Algorytm MD5 został opracowany w 1991 przez R. Rivesta [10]. Inne znane algorytmy wyznaczania skrótu to SHA [7] i RIPMD [6].

Sprzętowa implementacja algorytmu MD5 w układach FPGA jest realizowana zazwyczaj z użyciem osadzonych bloków pamięci (Embedded Memory Blocks - EMB) [8] ze względu na dużą liczbę stałych używanych podczas obliczeń. Jednak w przypadku wielu układów FPGA osadzone bloki pamięci mają na tyle dużą pojemność, że nie będą one w pełni wykorzystane – stałe wykorzystywane podczas obliczeń zajmą tylko niewielką cześć pamięci. Autorzy postanowili wykorzystać osadzone bloki do realizacji mikroprogramowanego układu sterującego. Zadaniem układu sterującego będzie generowanie stałych potrzebnych podczas obliczeń oraz wykorzystanie pozostałej wolnej przestrzeni w pamięci do realizacji innych zadań. Ze względu na charakter algorytmu – dużą liczbę sekwencyjnie wykonywanych operacji zdecydowano się zastosować mikroprogramowany układ sterujący [1, 2]. Układy mikroprogramowane kojarzą się przede wszystkim z systemami komputerowymi jednak mogą one z powodzeniem zostać użyte do realizacji układu sterującego w dowolnym systemie cyfrowym.

2. Algorytm MD5

Algorytm MD5 składa się z następujących etapów [10]:

Uzupełnienie wiadomości o dodatkowe bity

Do *b* bitowej wiadomości dołączany jest jeden bit o wartości '1' na końcu wiadomości. Następnie na początku wiadomości należy dołączyć tyle bitów o wartości '0' aby reszta z dzielenia całkowitego długości wiadomości była równa 448.

Dodanie długości wiadomości

Na początek wiadomości dołączana jest wartość *b* zapisana na 64 bitach. W ten sposób pierwszy blok wiadomości ma dokładnie 512 bitów.

Zainicjowanie bufora

W trakcie obliczeń algorytm operuje na 4 zmiennych 32 bitowych (*A*, *B*, *C*, *D*). Przed rozpoczęciem obliczeń należy przypisać tym zmiennym następujące wartości:

$A = 0123\ 4567_{(16)}$
$B = 89ab\ cdef_{(16)}$
$C = fedc\ ba98_{(16)}$
$D = 7654\ 3210_{(16)}$

Przetworzenie wiadomości

Wiadomość jest dzielona na 512 bitowe bloki, które są kolejno przetwarzane. Niech X_j oznacza j-ty blok wiadomości. Przetwarzanie bloku X_j rozpoczyna się od zapamiętania bieżącego stanu zmiennych A, B, C, D z użyciem tymczasowych zmiennych AA, BB, CC i DD. Następnie algorytm realizuje kolejno 64 kroki, w których wykonywane są następujące operacje:

$$A=B+((A+\xi(B,C,D) + X\_j[k]+T[i]) << s) \qquad (1)$$
$$A \leftarrow D,\ B \leftarrow A,C \leftarrow B,\ D \leftarrow C,$$

gdzie operator << oznacza cykliczne przesunięcie o s bitów, $X_j[k]$ oznacza k-te 32 bitowe słowo w bloku X_j, a $T[i]$ oznacza i-tą 32 bitową stałą używaną podczas obliczeń. Funkcja $\xi(B, C, D)$ zależy od numeru kroku. W krokach od 1 do 16 obliczana jest funkcja

$$F(X,Y,Z)= XY + \neg X\ Z,$$

w krokach 17 do 32 obliczana jest funkcja

$$G(X,Y,Z)=XZ + Y\neg Z,$$

w krokach 33 do 48 obliczana jest funkcja

$$H(X,Y,Z)=X \oplus Y \oplus Z,$$

w krokach 49 do 64 obliczana jest funkcja

$$I(X,Y,Z) = Y \oplus (X+ \neg Z),$$

gdzie operator $\oplus$ oznacza funkcję XOR. Ostatnim etapem przetwarzania bloku X_j jest dodanie do A, B, C i D wartości zapamiętanych na początku obliczeń.

A =A + AA;
B =B + BB;
C =C + CC;
D =D + DD;

Na rysunku 1 przedstawiony został fragment programu w języku C, który realizuje obliczenia dla bloku X_j. Funkcje FF, GG, HH i II odpowiadają operacjom (1).

```
FF(a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
FF(d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
FF(c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
...
FF(d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
FF(c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
FF(b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

/* Cykl 2 */
GG(a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
GG(d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
GG(c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
...
GG(d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
GG(c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
GG(b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

/* Cykl 3 */
HH(a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
HH(d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
HH(c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
...
HH(d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
HH(c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
HH(b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

/* Cykl 4 */
II(a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
II(d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
II(c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
...
II(d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
II(c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
II(b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */
```

Rys. 1. Fragment algorytmu MD5 w języku C [10]
Fig. 1. Part of MD5 algorithm in C [10]

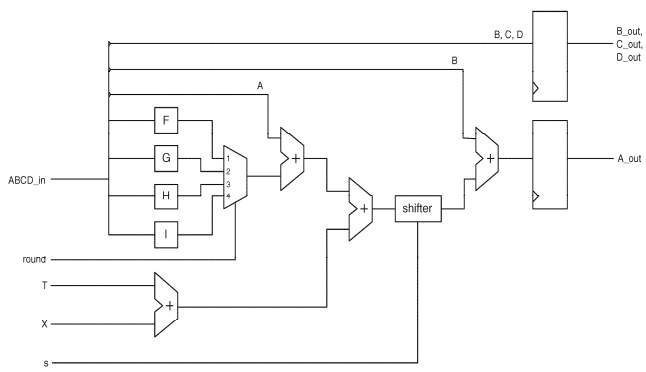
Zwrócenie wyniku obliczeń

Gdy zostaną przetworzone wszystkie bloki X_j zawartość zmiennych A, B, C, i D jest zwracana jako skrót całej wiadomości.

3. Realizacja sprzętowa algorytmu MD5

Na rysunku 2 przedstawiony został ogólny schemat układu realizującego pojedynczy krok obliczeń. Wejście *round* określa realizowaną funkcję (2 bity), wejście T określa stałe wykorzystywane przy obliczeniach (32 bity), wejście X określa fragment aktualnie przetwarzanej wiadomości (32 bity), wejście s określa przesunięcie bitowe (5 bitów). Źródłem sygnałów *round*, T, X i s może być pamięć [8], zbiór rejestrów [8], lub układ mikroprogramowany.

Pamięć potrzebna do przechowywania stałych *round*, T i s to: (32+5+2)*64=2496b, co w przypadku implementacji w układach firmy Xilinx stanowi około 14 % pojemności 18Kb bloku EMB [13]. Warto więc przeznaczyć blok EMB na mikroprogramowany układ sterujący, który może generować stałe T i s oraz inne dodatkowe mikrooperacje.



Rys. 2. Układu realizującego pojedynczy krok obliczeń w algorytmie MD5 [8]
Fig. 2. Single step of MD5 algorithm [8]

W literaturze dostępnych jest wiele różnych struktur układów mikroprogramowanych (Compostional Microprogram Control Unit – CMCU) [1,2]. Autorzy zdecydowali się przebadać następujące:

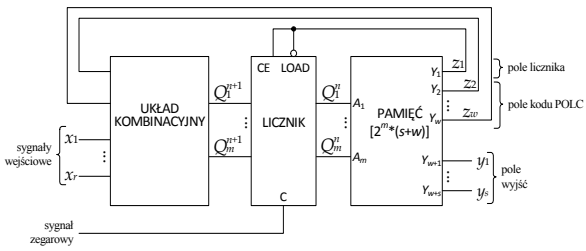
- mikroprogramowany układ sterujący ze wspólną pamięcią (*CMCU with Common Memory* – CMCU-CM) [1, 2],
- mikroprogramowany układ sterujący ze współdzieleniem kodów (*CMCU with Code Sharing* – CMCU-CS) [1, 2],
- mikroprogramowany układ sterujący z elementarnymi łańcuchami (*CMCU with Elementary Chains* – CMCU-EC) [1, 2],

oraz ich modyfikacje:

- wykorzystujące rozszerzony format mikroinstrukcji: CMCU-CME [5], CMCU-CSE [11] i CMCU-ECE [11],

- wykorzystujące mikroinstrukcje sterujące: CMCU-CMC [4], CMCU-CSC [3] i CMCU-ECC [3].

Mikroprogramowany układ sterujący składa się z dwóch zasadniczych części: *układu adresowego* oraz *pamięci sterującej* (rys. 3). Zadaniem układu adresowego jest wytworzenie adresu mikroinstrukcji dla pamięci sterującej. W pamięci sterującej umieszczone są mikroinstrukcje, które są wyprowadzane na wyjście układu ($y_1, y_2, \ldots, y_s$) oraz tworzą zmienne sterujące ($z_1, z_2, \ldots, z_w$) wykorzystywane do sterowania pracą części adresowej. Na rysunku 3 przedstawiony został układ CMCU-CME, w którym układ adresowy składa się z układu kombinacyjnego oraz licznika. Jeżeli z_1=1, wtedy zawartość licznika jest inkrementowana, w przeciwnym wypadku nowa wartość licznika jest ustalana przez układ kombinacyjny. Pierwszy wariant pracy odpowiada przejściu bezwarunkowemu - realizacji kolejnej mikroinstrukcji względem bieżącej. Drugi wariant odpowiada przejściu warunkowemu (wykonaniu skoku) na podstawie zewnętrznych sygnałów wejściowych (x_i) oraz zmiennych ($z_2, z_3, \ldots, z_w$) z pamięci sterującej.



Rys. 3. Mikroprogramowany układ sterujący ze wspólną pamięcią oraz rozszerzonym formatem mikroinstrukcji
Fig. 3. Microprogram control unit with common memory and extended microinstruction format

Pozostałe, wymienione wcześniej układy mikroprogramowanych różnią się od układu CMCU-CME sposobem generowania adresu mikroinstrukcji oraz sposobem rozmieszczenia mikroinstrukcji w pamięci sterującej. W strukturach CMCU-CS i CMCU-EC układ adresowy składa się z układu kombinacyjnego, licznika oraz rejestru. Adresowanie pamięci odbywa się z wykorzystaniem łańcuchów operacyjnych (OLC) lub łańcuchów elementarnych (EOLC) [1, 2]. W strukturach z mikroinstrukcjami sterującymi bieżący stan układu jest przechowywany w dodatkowych mikroinstrukcjach wstawionych do mikroprogramu.

W tabeli 1 przedstawione zostały wyniki syntezy (liczba bloków slice) sieci działań Γ realizującej algorytm MD5 wraz dodatkową funkcjonalnością, którą zasymulowano dodając sieć działań *s1* z pakietu LGSynth91 [12]. Sieć działań Γ posiada: 8 zmiennych wejściowych, 49 zmiennych wyjściowych, 84 stany, 170 przejść. Syntezę wszystkich układów przedstawionych w tabeli 1 wykonano w środowisku Xilinx ISE 12.1 dla układu Xilinx Spartan 3 (xc3s50-5pq208).

Tab. 1. Wynik syntezy sieci działań Γ
Tab. 1. Synthesis results of GSA Γ

FSM	CMCU								
	CM	CMC	CME	CS	CSC	CSE	EC	ECC	ECE
387	184	115	127	114	127	141	189	139	127

Na podstawie wyników przedstawionych w tabeli 1 można stwierdzić, że zastosowanie układu mikroprogramowanego pozwala na zmniejszenie zasobów sprzętowych potrzebnych do realizacji układu sterującego o ponad połowę względem układu zrealizowanego jako tradycyjny automat typu Moore'a (kolumna FSM). Automat Moore'a zrealizowany został zgodnie z wytycznymi dostępnymi w [13] . Stany automatu zostały zakodowane z użyciem kodowania „compact".

4. Podsumowanie

Obecnie produkowane układy FPGA są zazwyczaj wyposażone w osadzone bloki pamięci [9]. W przypadku sprzętowej realizacji algorytmu MD5 układy pamięci można wykorzystać do przechowywania stałych wykorzystywanych podczas obliczeń [8]. W artykule zaproponowane zostało inne rozwiązanie, w którym stałe używane podczas obliczeń są generowane przez mikroprogramowany układ sterujący. Zastosowanie układu mikroprogramowanego pozwala nie tylko generować stałe dla algorytmu MD5 ale jednocześnie wykorzystać pozostałą wolną przestrzeń pamięci do realizacji innych zadań. Badania potwierdzają, że zastosowanie układu mikroprogramowanego wymaga znacznie mniej tablic LUT w porównaniu do tradycyjnego rozwiązania z wykorzystaniem automatu Moore'a.

5. Literatura

[1] Adamski M., Barkalov A.: Architectural and Sequential Synthesis of Digital Devices. Oficyna Wydawnicza Uniwersytetu Zielonogórskiego, 2006.
[2] Barkalov A., Titarenko L.: Logic synthesis for Compositonal Microprogram Control Units, Springer, 2008.
[3] Barkalov A., Titarenko L., Bieganowski J.: Reduction in the number of LUT elements for control units with code sharing, Inter. Jurnal of Applied Mathematics and Computer Science, 2010, Vol. 20, Nr. 4.
[4] Barkalov A., Titarenko L., Bieganowski J.: Synthesis of microprogram control unit with control microinstructions, DES-Des'09: preprints of the 4th IFAC Workshop, 2009.
[5] Barkalov A., Titarenko L., Bieganowski J.: Synthesis of compostional microprogram control unit with extended microinstruction format, Mixed Design of Integrated Circuits and Systems – MIXDES'2009: 16th Inter. Conf., 2009.
[6] Dobbertin H., Bosselaers A., Preneel B. RIPEMD-160: A Strengthened Version of RIPEMD, Proc. of 3rd Inter. Workshop on Fast Software Encryption, 1996.
[7] Eastalke D, Jones P.: RFC:3174 US secure hash algorithm 1 SHA1, Network Working Group, 2001.
[8] Jarvinen K., Tommiska M., Skytta J.: Hardware implementation of the MD5 hash algorithm, HICSS'05 Proc. of the 38th Annual Hawaii Inter. Conf. on System Sciences, 2005.
[9] Maxfield C.: The Design Warrior's Guide to FPGAs, Academic Press, Inc., Orlando FL, USA, 2004.
[10] Rivest R. L.: RFC:1321 The MD5 message-digest algorithm, Internet Activities Board, Internet Privacy Task Force, 1992.
[11] Titarenko L., Bieganowski J.: Optimization of compositional microprogram control unit by modification of microinstruction format, Electronics and Telecommunications Quarterly, Vol. 55, Nr. 2, 2009.
[12] Yang S.: Logic Synthesis and optimization benchmarks user guide, Technical report, Microelectronics Center of North Carolina, 1991.
[13] Xilinx, Synthesis and Simulation Design Guide, 2008.
[14] FreeBSD Handbook, Security DES, Blowfish, MD5, and Crypt, http://www.freebsd.org/doc/en/books/handbook/crypt.html

otrzymano / received: 13.05.2011
przyjęto do druku / accepted: 04.07.2011 *artykuł recenzowany*