Dominik BUDYN [1], Paweł SOKOŁOWSKI [1], Paweł RUSSEK [1,2], Kazimierz WIATR [1,2]
[1] AGH. KATEDRA ELEKTRONIKI, Al. Mickiewicza 30, 30-059 Kraków
[2] AGH. ACK CYFRONET, ul. Nawojki 11, 30-950 Kraków

Zastosowanie języka Impulse C do implementacji sprzętowej algorytmów kryptograficznych w FPGA na przykładzie algorytmu DES

Inż. Dominik BUDYN

Ukończył studia pierwszego stopnia na Akademii Górniczo – Hutniczej w Krakowie w 2011 roku. Obecnie jest studentem drugiego stopnia studiów na kierunku Elektronika i Telekomunikacja na Akademii Górniczo-Hutniczej. W ramach pracy magisterskiej zajmuje się implementacją algorytmów kryptograficznych w układach FPGA z wykorzystaniem języków wysokiego poziomu.

e-mail: budyn.dominik@gmail.com

Inż. Paweł SOKOŁOWSKI

Jest absolwentem studiów pierwszego stopnia na Katedrze Elektroniki Akademii Górniczo-Hutniczej w roku 2011. Obecnie kontynuuje studia drugiego stopnia na kierunku Elektronika i Telekomunikacja na Akademii Górniczo-Hutniczej.

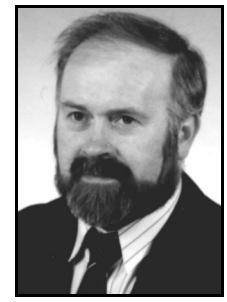
e-mail: pawel.sokol07@gmail.com

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 i pracownikiem ACK Cyfronet. Prowadzone prace badawcze dotyczą sprzętowej akceleracji obliczeń przy pomocy architektur dedykowanych, zagadnień realizacji obliczeń przy użyciu rekonfigurowalnego sprzętu oraz wykorzystania układów reprogramowalnych w obliczeniach naukowych i technicznych wielkiej skali.

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 Akademii Górniczo-Hutniczej oraz Dyrektor Akademickiego Centrum Komputerowego Cyfronet AGH. Prowadzone prace badawcze dotyczą komputerowego sterowania procesami, systemów wizyjnych, systemów wieloprocesorowych, układów programowalnych, rekonfigurowalnych systemów obliczeniowych i sprzętowych metod akceleracji obliczeń.

e-mail: wiatr@agh.edu.pl

Streszczenie

Artykuł opisuje implementację algorytmu DES z wykorzystaniem języka Impulse C. Język Impulse C należy do rodziny języków określanych wspólnym mianem języków HLL (High Level Language), których zadaniem jest, w stosunku do języków VHDL i Verilog, rozwinięcie możliwości opisu sprzętu na poziomie systemu. W założeniu, opis taki ma być syntezowalny i możliwy do implementacji w układach FPGA. W artykule skrótowo przedstawione zostały najważniejsze cechy charakterystyczne języka Impulse C oraz narzędzi programistycznych związanych z tym językiem. Przedstawiono również kilka sposobów optymalizacji projektów wykonywanych w języku Impulse C.

Słowa kluczowe: kryptografia, DES, języki HLL, procesory dedykowane, akceleracja obliczeń.

A case study on implementation of the DES algorithm on the FPGA platform using the Impulse-C language

Abstract

In this paper we describe an FPGA implementation of the DES algorithm using Impulse C language. Impulse C is the one of the representatives of a growing group of hardware description languages known as High Level Languages (HLLs). The Impulse C extends standard ANSI C by introducing an extensive set of pragmas, new data types and library functions [3]. The Impulse C compiler translates programs that are written in 'C' into RTL-level system description. Section 1 describes some of the most important properties of the Impulse C language that are used in discussion conducted on later sections. Section 2 presents briefly the DES algorithm. In the next section a basic implementation of the DES algorithm is given. The block diagram of the designed circuit is shown in Fig. 1. The design was implemented using Xilinx Virtex 5 LX 220 FPGA. The basic version originates from the software version of the algorithm. Thus it is not optimized for hardware implementation. In the last section some improvements of the basic design available in the Impulse C are described. Those include a migration of arrays from a block RAM to FPGA internal registers and replication combinatorial logic. The result for the basic version of the algorithm and its optimized versions are presented in Table 1. Fig. 2 depicts the final algorithm implementation. The optimized version allows for a 8,25 times speedup over the basic version.

Keywords: cryptography, DES, High Level Language, custom processors, computing acceleration.

1. Wstęp

Struktura algorytmów kryptograficznych umożliwia ich efektywną implementację w układach FPGA [1, 2]. Implementacja sprzętowa bardziej złożonych algorytmów przy wykorzystaniu wyłącznie języków opisu niskiego poziomu, takich jak VHDL czy Verilog, jest jednak pracochłonna i długotrwała. Dlatego poszukiwane są nowe metody zwiększenia produktywności projektowania. Jedną z metod jest wykorzystanie języków programowania wysokiego poziomu (HLL). Przykładem jest język Impulse C, który pozwala na generację syntezowalnego opisu układu w języku VHDL na podstawie algorytmu opisanego w standardzie języka ANSI C.

2. Język Impulse C

Język Impulse C należy do coraz popularniejszej rodziny narzędzi typu 'C-to-FPGA'. Kompilatory tego typu pozwalają skrócić czas konieczny na wykonanie implementacji sprzętowej algorytmu w porównaniu z projektowaniem wykonanym w tradycyjnych językach opisu sprzętu. Język Impulse C można określić jako rozwinięcie standardu ANSI C o zestaw dyrektyw kompilatora sterujących syntezą sprzętu, nowe typy danych oraz operujące na nich funkcje [3].

Impulse-C jest częścią pakietu Co-Developer. W skład pakietu CoDeveloper wchodzi również, między innymi, narzędzie Stage Master Explorer, pozwalające szacować wydajność układu. Dostępna jest również 'wtyczka' pozwalający na debugowanie aplikacji Impulse C w środowisku Visual Studio.

Model programistyczny języka Impulse C zawiera dwa podstawowe elementy: procesy i obiekty komunikacyjne. Proces to wykonująca się niezależnie od pozostałych sekcja programu, połączona z innymi procesami przy pomocy obiektów komunikacyjnych: strumieni, sygnałów, pamięci współdzielonej i semaforów.

Proces można klasyfikować jako programowy lub sprzętowy. Procesy sprzętowe obłożone są ściślejszym zestawem zasad, ponieważ na podstawie opisującego ich kodu generowany jest kod VHDL. Zasady ograniczające procesy programowe są luźniejsze, ponieważ procesy programowe wykonywane są przez CPU platformy. Procesy tworzone są statycznie, przy uruchamianiu programu, wtedy również podejmowana jest decyzja o przypisaniu procesu do jednostki wykonawczej. Preferowanym sposobem komunikacji w programach napisanych w języku Impulse C są strumienie. Strumień to jednokierunkowe, buforowane połączenie punkt – punkt pomiędzy dwoma procesami. Dla operacji wykonywanych na strumieniach w języku Impulse C zdefiniowane zostały funkcje, służące do otwierania i zamykania strumienia oraz zapisywania i odczytywania danych.

Język Impulse C pozwala na stosowanie w procesach sprzętowych technik stosowanych w języku VHDL, czyli na przykład: rozwijanie pętli (ang. loop unrolling) i potokowość (ang. pipelining). Rozwijanie pętli duplikuje ciało pętli tyle razy, ile razy miała zostać wykonana pętla. Pozwala to wykonywać wszystkie iteracje pętli równocześnie, jeśli nie występuje zależność danych między iteracjami. Potokowość pozwala zrównoleglić wykonywanie programu przez wykonywanie kilku etapów algorytmu równocześnie na różnych danych. Potokowość i rozwijanie pętli implementowane są w Impulse C przy pomocy dyrektyw prekompilatora.

3. Algorytm DES

Algorytm DES, wraz ze swymi pochodnymi, jest jednym z najpopularniejszych i najlepiej zbadanych algorytmów kryptograficznych. Obecnie wykorzystywane są przede wszystkim rozwiązania oparte o DES ze zwiększoną długością klucza: takie jak DESX, 3DES czy będący polskim standardem algorytm NASZ (oparty o 3DES).

Algorytm DES jest algorytmem blokowym operującym na bloku o długości 64 bitów. Struktura algorytmu oparta jest o strukturę sieci Feistela. Algorytm ma szesnaście rund, przy czym w każdej rundzie szyfrowana jest jedna połowa wiadomości (32 bity). Druga połowa przepisywana jest bez zmian do kolejnej rundy algorytmu. Do każdej z rund z 56-bitowego klucza wybieranych jest 48 bitów.

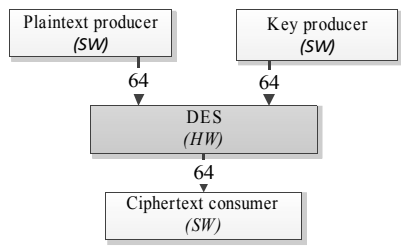
Bezpieczeństwo algorytmu DES opiera się na operacjach permutacji oraz podmiany. Operacje permutacji pozwalają zwiększyć wpływ każdego z bitów wejściowych na kształt zaszyfrowanej wiadomości. Operacje podmiany ukrywają powiązanie między tekstem wejściowym i wyjściowym. Podmiany wykonywane są w algorytmie DES z wykorzystaniem tak zwanych S-boxów, implementowanych zwykle jako bloki pamięci. Oba rodzaje funkcji wykorzystywanych przez algorytm DES umożliwiają wydajną implementację algorytmu w układach FPGA. Dokładne informacje na temat algorytmu DES znaleźć można w [5].

4. Sprzętowa implementacja algorytmu

Algorytm został zaimplementowany z wykorzystaniem wyłącznie języka Impulse C, bez stosowania modułów opisanych w języku VHDL. Schemat blokowy układu przedstawiony został na rysunku 3.

Do implementacji sprzętowej wybrany został proces „DES", wykonujący operację szyfrowania. Pozostałe procesy zaimplementowane zostały jako procesy programowe, ponieważ ich zadaniem jest odczytanie danych wejściowych z pliku (proces „plaintext producer") i zapisanie do pliku danych wyjściowych (proces „ciphertext consumer") lub wygenerowanie wartości inicjalizującej klucz. Do komunikacji pomiędzy procesami wykorzystane zostały strumienie o szerokości 64 bitów. Klucz i dane przesyłane są do układu FPGA przy pomocy osobnych strumieni.

Do wstępnego oszacowania wydajności zaprojektowanego układu wykorzystane zostało narzędzie Stage Master Explorer z pakietu CoDeveloper. Narzędzie to posługuje się dwoma podstawowymi parametrami, nazwanymi Rate i Max Unit Delay. Parametr Rate mówi, co ile cykli na wyjściu układu pojawi się nowy wynik. Parametr Max Unit Delay (MUD) informuje o liczbie poziomów logiki między kolejnymi rejestrami potokowymi, co przekłada się na maksymalną częstotliwość pracy układu.



Rys. 1. Schemat blokowy zaimplementowanego algorytmu
Fig. 1. Block diagram of the implemented algorithm

Uzyskane wyniki pozwalają zidentyfikować krytyczne sekcje programu, decydujące o szybkości wykonywania obliczeń. Mogą to być zarówno długie ścieżki logiki kombinacyjnej (duża wartość parametru Max Unit Delay), skutkujące niską częstotliwością zegara, jak również problemy z dostępem do współdzielonych zasobów – pamięci i elementów obliczeniowych (jedna z przyczyn wysokiej wartości parametru Rate). Język Impulse C pozwala optymalizować liczbę poziomów potoku vs. długość ścieżki między kolejnymi rejestrami potokowymi. Pozwala podnieść częstotliwość pracy układu kosztem zwiększenie latencji i liczby cykli potrzebnych do uzyskania nowego wyniku.

Tab. 1. Wyniki syntezy dla algorytmu w wersji podstawowej i po optymalizacji
Tab. 1. Synthesis results for the basic and optimized algorithm

	Wersja implementacji	RATE	MUD	f_{max} [MHz]	Wykorzystanie rejestrów [%]	Wykorzystanie LUT [%]	Wydajność [mln bloków/s]
1	Bazowa	8	25	187,5	8	4	23,43
2	Z pamięcią rozproszoną	3	25	193,5	20	6	63,5
3	Z powielonymi zasobami	1	31	___	___	___	___
4	Trzy bloki szyfrujące	1	25	193,4	68	25	193,4

Wyniki dla bazowej implementacji oraz dla implementacji z zastosowaniem kilku metod optymalizacji opisanych poniżej przedstawiono w tabeli 1. Platformą docelową był układ Virtex 5 LX220 [6]. W tabeli przedstawione zostały parametry wyznaczone przez narzędzie Stage Master Explorer (Rate i Max Unit Delay - MUD), maksymalna częstotliwość zegara uzyskana podczas syntezy, zajęte zasoby oraz teoretyczna wydajność.

5. Optymalizacja algorytmu

a. Ograniczenia wynikające z dostępu do pamięci przechowującej podklucze do kolejnych iteracji

Zgodnie z danymi zawartymi w tabeli 1 liczba cykli zegara między pojawieniem się kolejnych wyników na wyjściu potoku przetwarzającego dane (parametr Rate) wynosi 8. Przy generowaniu kodu VHDL na podstawie kodu Impulse C tablice domyślnie zamieniane są w bloki dwuportowych pamięci BRAM. Wskutek tego, w jednym cyklu zegara możliwe jest odczytanie dwóch elementów tablicy, więc do odczytania wszystkich kluczy potrzebnych jest osiem cykli zegara. Rozwiązaniem problemu jest umieszczenie tablicy jako pamięci rozproszonej w układzie FPGA. Wartości przechowywane w zdefiniowanej w ten sposób tablicy traktowane są jak niezależne zmienne i w każdym cyklu

zegara możliwy jest dostęp do wszystkich elementów tablicy. Aby umieścić tablicę w formie pamięci rozproszonej należy skorzystać z funkcji *co_array_config*.

Wyniki po wprowadzeniu zmian przedstawione zostały w drugim wierszu tabeli 1. Liczba cykli zegara między uzyskaniem kolejnych danych wyjściowych zmniejszyła się z 8 do 3, co oznacza zwiększenie wydajności 2,6 raza, za cenę niewielkiego zwiększenia wykorzystywanych zasobów.

b. Ograniczenie wynikające ze współdzielenia zasobów obliczeniowych

Rozwiązaniem ograniczeń wydajności wynikających ze współdzielenia zasobów obliczeniowych jest powielenie tych zasobów. W wypadku naszego algorytmu krytyczne były zasoby, które odpowiadały za realizację funkcji S-boxów. Język Impulse C pozwala na powielenie logiki realizującej funkcję przy pomocy dyrektywy *#pragma co inline*. Po zastosowaniu tej dyrektywy dla każdego wywołania funkcji w kodzie Impulse C tworzona jest osobna instancja w kodzie VHDL. Po wygenerowaniu kodu narzędzie Stage Master Explorer wskazuje, że możliwe będzie uzyskiwanie nowego wyniku w każdym cyklu zegara. Synteza układu nie była jednak możliwa, ponieważ wymagane zasoby przekraczały możliwości dostępnego układu FPGA.

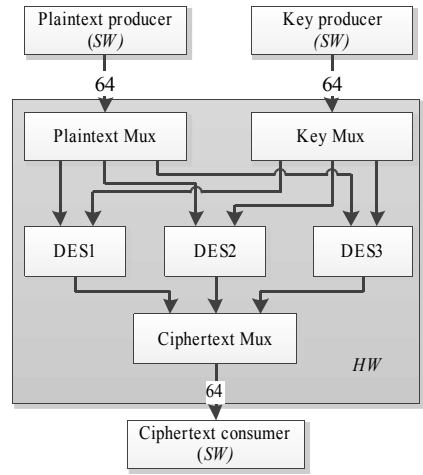
Rozwiązaniem problemu przekroczenie wykorzystywanych zasobów może być zastosowanie do realizacji funkcji S-boxów modułów napisanych w języku VHDL. Zamiast tego, postanowiono jednak powielić jednostkę szyfrującą (bez powielania logiki realizującej S-boxy), co również pozwoli zwiększyć wydajność układu.

c. Zrównoleglenie obliczeń przez dodanie kolejnych procesów szyfrujących

Język Impulse C umożliwia łatwą rozbudowę zaprojektowanego układu o kolejne bloki realizujące tę samą funkcję. W zaprojektowanym układzie blok odpowiadający za szyfrowanie został powielony trzykrotnie. Schemat układ przedstawiony został na rysunku 2. Konieczne było utworzenie dodatkowych bloków pełniących funkcję multiplekserów i demultiplekserów, ponieważ strumienie w języku Impulse C mogą być wykorzystywane jedynie do połączeń punkt – punkt. Bloki multiplekserów zaimplementowane zostały jako procesy sprzętowe. Proces „Key Mux" powiela wartość inicjującą klucz – odczytuje ją ze strumienia wejściowego, a następnie tę samą wartość przepisuje do trzech strumieni wyjściowych. Proces „Plaintext Mux" odczytuje w każdym cyklu nową wartość, a następnie zapisuje ją do jednego ze strumieni wyjściowych. W ten sposób wiadomość jest dzielona na trzy części, z których każda jest szyfrowana niezależnie przez różne procesy szyfrujące. Proces „Ciphertext Mux" wykonuje odwrotną operację – składa trzy otrzymane fragmenty zaszyfrowanego tekstu w jeden. Poza utworzeniem bloków multipleksujących zmodyfikowana została funkcja konfiguracyjna, w której umieszczone są deklaracje elementów układu (procesów i obiektów komunikacyjnych).

Wyniki syntezy dla układu zawierającego trzy bloki szyfrujące przedstawione zostały w ostatnim wierszu tabeli 1. Po potrojeniu układu szyfrującego nowa wartość będzie obliczana w każdym cyklu zegara. Częstotliwość pracy układu praktycznie nie uległa zmianie, co oznacza, że wydajność układu zwiększy się trzykrotnie. Wykorzystywane wcześniej bloki, zarówno sprzętowe jak programowe nie wymagały modyfikacji. Dzięki temu rozbudowa układu wymagała niewielkiego nakładu pracy.

Możliwe jest również wykorzystanie zaprojektowanych bloków do implementacji algorytmu 3DES, wzmocnionej wersji algorytmu DES. Algorytm 3DES wykorzystuje trzy bloki szyfrujące, które w zależności od wersji algorytmu wykorzystują jeden, dwa lub trzy różne klucze. Podobnie jak w przypadku rozbudowy układu o kolejne jednostki szyfrujące, również w przypadku przejścia z algorytmu DES do 3DES modyfikacje wiązałyby się przede wszystkim ze zmianami funkcji konfiguracyjnej oraz dodaniem procesów i strumieni komunikacyjnych.



Rys. 2. Schemat blokowy zoptymalizowanego algorytmu
Fig. 2. Block diagram of the optimized algorithm

6. Wnioski

W ramach projektu przy użyciu języka Impulse C zaimplementowany został algorytm DES. Wykonane i przedstawione zostały metody optymalizacji projektu, pozwalające zwiększyć wydajność zaprojektowanego układu. Język Impulse C pozwolił skrócić czas wykonywania opisu algorytmu oraz przede wszystkim wprowadzania modyfikacji w projekcie, przy jednoczesnym zachowaniu wysokiej wydajności układu. Docelowo, algorytm ma być uruchomiony na platformie DRC AC 2020 [7]. Kod programu w języku Impulse C umieszczony został na stronie [8].

Niniejsza praca jest finansowana przez Narodowe Centrum Badań i Rozwoju nr umowy SP/I/1/77065/10 w ramach programu strategicznego: Interdyscyplinarny system interaktywnej informacji naukowej i naukowo technicznej.

7. Literatura

[1] Gielata Artur, Russek Paweł, Wiatr Kazimierz: Implementacja standardu szyfrowania AES w układzie FPGA dla potrzeb sprzętowej akceleracji obliczeń, Pomiary, Automatyka, Kontrola, 2007 vol. 53 nr 5 s. 48-50.
[2] Janiszewski Marcin, Russek Paweł, Wiatr Kazimierz: Realizacja w układach FPGA mnożenia Montgomery dla akceleracji operacji kryptograficznych, Pomiary, Automatyka, Kontrola, 2008 vol. 54 nr 8 s.550-552. - Bibliogr. s. 552.
[3] Impulse C User Guide
[4] Pellerin David, Thibault Scott: Practical FPGA programming in C. Prentice Hall, 2005.
[5] Paar Christof, Pelzl Jan: Understanding Cryptography. Springer, 2010.
[6] http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf
[7] http://www.drccomputer.com/pdfs/DRC_Accelium_Coprocessors.pdf
[8] http://www.cyfronet.pl/en/?a=zao/ReconfigurableComputing

otrzymano / received: 17.04.2012
przyjęto do druku / accepted: 01.06.2012 *artykuł recenzowany / revised paper*