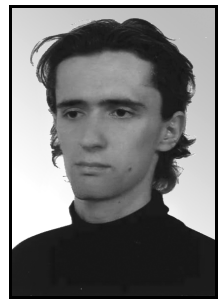
Marcin JANISZEWSKI [1], **Paweł RUSSEK** [1,2], **Kazimierz WIATR** [1,2]
[1] AKADEMICKI CENTRUM KOMPUTEROWE „CYFRONET" AGH
[2] AKADEMIA GÓRNICZO-HUTNICZA, KATEDRA ELEKTRONIKI

Realizacja w układach FPGA mnożenia Montgomery dla akceleracji operacji kryptograficznych

Mgr inż. Marcin JANISZEWSKI



Ukończył studia na wydziale Elektrotechniki, Automatyki i Elektroniki AGH Kraków (2007). Obecnie pracownik ACK Cyfronet AGH na stanowisku referenta technicznego. Zainteresowania naukowe: systemy i aplikacje oparte na układach reprogramowalnych z wykorzystaniem języka opisu sprzętu VHDL.

e-mail: marcin.jan@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. 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

Streszczenie

W niniejszej pracy podjęto temat realizacji modułu sprzętowego, mogącego skutecznie przyspieszyć programowe realizacje operacji kryptograficznych. Rozpatrywanym algorytmem jest szyfrowanie asymetryczne RSA. Moduł został zaimplementowany w układzie firmy Xilinx – Virtex 4 LX200. Prędkość działania modułu została porównana z najpopularniejszymi rozwiązaniami programowymi. Rezultaty pokazują, że rozwiązania bazujące na układach rekonfigurowanych mogą konkurować z implementacjami opartymi na procesorach ogólnego przeznaczenia (GPP).

Słowa kluczowe: mnożenie Montgomery, mnożenie modulo, FPGA, RSA.

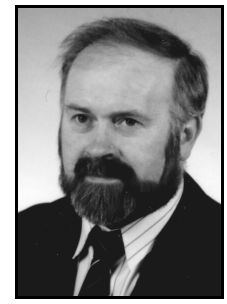
Implementation of Montgomery multiplication for cryptographic algorithm acceleration in FPGA

Abstract

Modular exponentiation is a key operation for RSA cryptographic algorithm. There are many algorithms for computing modular exponentiation – equation 1. The most basic are right to left and left to right binary algorithms. For key length k=1024 bits, 1024 modular squarings and 512 modular multiplications on average must be performed. There are many optimization which allows to minimize the number of multiplications, however they are more suited for software implementations. Therefore key factor for faster modular exponentiation is fast multiplier module. This work presents example implementation of modulo multiplier using Montgomery multiplication algorithm [1]. Montgomery multiplication is the most efficient algorithm when large number of multiplications must be performed with respect to the same modulus n. Our results show that timings comparable with modern processors can be achieved – table 2. This works also presents optimizations of proposed module, which allow greater speedup and application of FPGA based modules as hardware accelerators.

Keywords: Montgomery multiplication, modular multiplication, FPGA, RSA.

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

1. Wstęp

RSA jest najbardziej popularnym algorytmem kryptografii klucza publicznego. Nawet jeśli dwie strony przeprowadzają pierwsze połączenie i nie ustaliły wcześniej wspólnego hasła bezpieczna komunikacja jest możliwa. Teza ta została udowodniona przez Diffiego i Hellmana w 1976 r. [2]. RSA stanowi jedną z realizacji tego pomysłu, wynalezioną przez R. Rivesta, A. Shamira oraz L. Adelmana w 1977 r. [3]. Główna operacją związaną zarówno z szyfrowanie jak i deszyfrowaniem jest obliczanie eksponenty modulo – równanie 1.

$$a = c^d \bmod n. \tag{1}$$

Jeśli c oznacza wiadomość zaszyfrowaną oraz d prywatną eksponentę (klucz prywatny) równanie takie reprezentuje proces deszyfrowania. Z uwagi, że w praktycznych rozwiązaniach prywatna eksponenta jest bardzo długa (dużo większa od publicznej) proces jej obliczania jest bardzo czasochłonny. Z tego powodu analiza możliwości przyspieszenia tej operacji jest uzasadniona.

W celu obliczenia eksponenty dla przykładowego klucza d o długości k_d=1024 bitów 1024 podnoszenia do kwadratu oraz średnio musi zostać wykonanych 512 mnożeń. Istnieją metody zmniejszenia liczby potrzebnych mnożeń, jednak kluczowym aspektem dla szybkości obliczania eksponenty modulo, jest czas obliczania pojedynczego mnożenia modularnego.

W przypadku, gdy duża liczba mnożeń musi zostać wykonanych względem tego samego modułu n, najwydajniejszy jest algorytm Mnożenia Montgomery [1, 4]. Podczas obliczania eksponenty modulo mamy do czynienia z takim właśnie przypadkiem. Ponadto algorytm Montgomery dobrze nadaje się do implementacji sprzętowych z uwagi na proste operacje składowe (dodawania, przesunięcia logiczne).

Algorytm ten oblicza:

$$Montgomery(\bar{x}, \bar{y}) = \bar{z} = \bar{x} \cdot \bar{y} \cdot R^{-1} \bmod n. \tag{2}$$

Podczas realizacji mnożenia z użyciem powyższego argumenty wejściowe muszą zostać przeniesione do dziedziny n-reszt. W tym przypadku:

$$\begin{aligned} \bar{x} &= x \cdot R \bmod n \\ \bar{y} &= y \cdot R \bmod n \end{aligned} \tag{3}$$

Wyniki działania powyższego algorytmu są prawidłowe, jeśli $x, y < n$ oraz NWD(R,n)=1 (liczby względnie pierwsze). Szczególnie interesujący dla nas jest przypadek, gdy $R = 2^k$. Umożliwia to całkowite wyeliminowanie potrzeby wykonywania dzielenia w trakcie algorytmu mnożenia metodą Montgomery. Ponadto, aby uzyskać właściwy wynik musimy przeprowadzić jeszcze

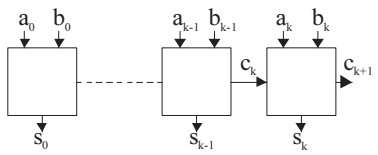
dodatkową operację, aby przenieść wynik z dziedziny *n*-reszt to do dziedziny liczb naturalnych.

$$z = Montgomery(\bar{z},1) \tag{4}$$

Konieczność przeprowadzania transformacji zarówno wejściowych jak i wyjściowych argumentów powoduje, że metoda ta nie jest opłacalna przy niewielkiej liczbie mnożeń. W przypadku eksponencjacji modulo, podczas kolejnych mnożeń transformacje takie mają jedynie miejsce na początku i na samym końcu obliczeń. W trakcie procesu wyniki częściowe są utrzymywane w postaci *n*-reszt. Dla takich działań algorytm Montgomery jest uznawany za najszybszą metodę obliczania mnożenia modulo. Podstawową wersję binarnego, uproszczonego algorytmu mnożenia Montgomery przedstawia poniższy kod:

```
1.  for i=0 to k
2.      if (a_i)      c = c + b
3.      if (c_0)      c = c + m
4.      c = c / 2
5.  end for
```

W powyższym algorytmie zakładamy, że możemy pominąć końcowe odejmowanie zgodnie z [5]. Główna pętla zawiera co najwyżej dwa dodawania i jedno przesunięcie w prawo. Sposób realizacji operacji dodawania w powyższym algorytmie jest więc głównym czynnikiem determinującym szybkość działania docelowego modułu. Z uwagi na to, że argumenty składając się z tysięcy bitów, dobrze jest aby operacje na takich argumentach przeprowadzić w dedykowanych jednostkach obliczeniowych o odpowiednio dopasowanej szerokości. Istnieje ogromna liczba sposobów implementacji sumatorów w układach FPGA. Najprostszym z nich, zarazem najbardziej wydajnym. W FPGA znajdują się dedykowane ścieżki propagacji przeniesienia dla sumatorów typu „carry ripple". Pozwalają one na wydajną implementację krótkich (do 64 bitów) sumatorów. W przypadku sumatorów większych należy stosować dodatkowe techniki.



Rys. 1. Sumator typu „carry ripple"
Fig. 1. Carry ripple adder

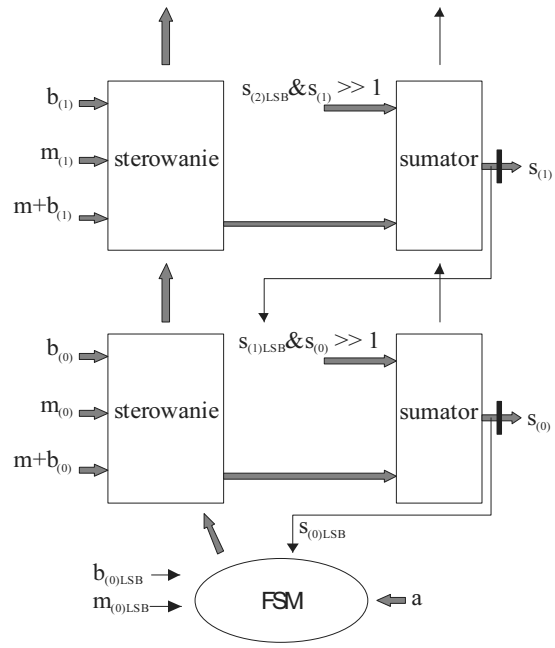
Każdy z bloków wewnątrz ścieżki zbudowany jest z pełnego sumatora z wejściem oraz wyjściem przeniesienia odpowiednio c_i oraz c_{i+1}. Ścieżka krytyczna takiego układu powstaje między wejściem najmłodszego bitu (a_0 lub b_0) a najstarszym bitem wyniku (s_i). W przypadku długich argumentów wejściowych istnieje potrzeba rozbicia takiego łańcucha na mniejsze i umieszczenie pomiędzy nimi rejestrów. W ten sposób, zwiększa się jednak liczba cykli wymagana do ukończenia pojedynczej operacji dodawania. Rozwiązania sprzętowe mają na celu rozwiązanie problemu krytycznej ścieżki poprzez wykorzystanie specyficznych cech algorytmu. Najczęściej stosowanymi metodami w tym wypadku są:

a) wprowadzenie redundancyjnej reprezentacji liczb i zastosowanie sumatorów typu CSA (carry save adder)
b) podział sumatora na mniejsze części i zastosowanie pipelingu sygnałów sterujących

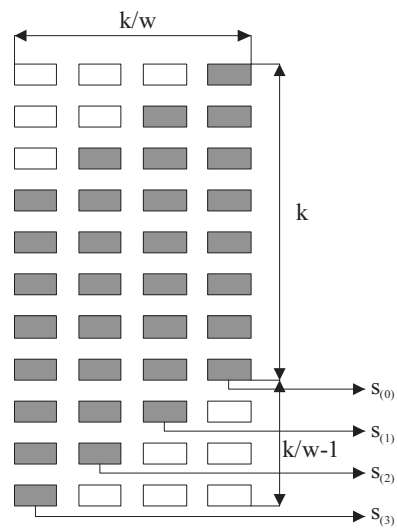
Architektura z podpunktu a) jest stosunkowo popularna [6]. Dzięki niej problem propagacji przeniesienia zostaje odłożony do końca realizacji algorytmu. Istnieją specjalne techniki, umożliwiające utrzymywanie redundancyjnej reprezentacji pomiędzy kolejnymi mnożeniami w trakcie obliczania eksponenty modulo [6]. Architektura umożliwia równoległe obliczanie wszystkich bitów, jednak pewnym problem pozostaje kontrola tak dużej liczby bloków podstawowych (każda kolejna iteracja zależy od wartości najmłodszego bitu wyniku częściowego).

2. Implementacja

Zaproponowana w pracy architektura składa się z rozbitego na wiele elementów sumatora typu „carry ripple". Podobne podejście, możemy znaleźć na przykład w [7]. Uproszczony schemat został przedstawiony na rysunku 2. Długości elementów pojedynczych sumatorów mogą być tak dobrane, aby najlepiej wykorzystać możliwości realizacji sumowania docelowego układu FPGA. Długość bitową pojedynczego elementu oznaczymy *w*, długość bitową argumentów wejściowych *k*. Pojedyncze mnożenie może zostać ukończone w czasie $k + \frac{k}{w}$ cykli. W rzeczywistości każde z mnożeń musi zostać dodatkowo poprzedzoną fazą obliczeń wstępnych trwającą około 10 cykli. Z uwagi, że *k* domyślnie wynosi 2048, tak więc cykl obliczeń wstępnych trwa pomijalnie krótko.



Rys. 2. Schemat blokowy modułu
Fig. 2. Module block diagram



Rys. 3. Działanie modułu dla w=2 oraz k=8
Fig. 3. Module operation for w=2 and k=8

Zaprojektowany układ składa się z $\frac{k}{w}$ elementów, każdy złożony z sumatora i jednostki sterującej procesem dodawania. Schemat obliczania wyniku przy użyciu tej architektury przedstawia rysunek 3. Wiersze przedstawiają działanie elementów dla kolejnych taktów zegara. Każda kolumna odpowiada pojedynczemu elementowi. Uproszczony schemat działania dla szerokości elementu w=2, oraz k=8.

3. Parametry otrzymanego modułu

Zaprojektowany moduł został zaimplementowany dla różnych długości sumatorów składowych oraz rozmiarze k=2048 bitów. Zestawienie otrzymanych parametrów takiego modułu przedstawia tabela 1.

Tab. 1. Parametry zaimplementowanego modułu dla k=2048
Tab. 1. Module parameters for k=2048

Długości sumatorów [bit]	Powierzchnia [LUT]	Częstotliwość zegara [MHz]
2	30 043 (16%)	200
4	27 402 (15%)	164
8	26 629 (14%)	165
16	26 237 (14%)	169

Jeśli założymy że zaproponowany moduł zostanie wykorzystanie przy obliczaniu eksponenty metodą binarną od prawej do lewej, przy dwóch równolegle działających modułach (mnożący i podnoszący do kwadratu) oraz że prędkość ich działania w takim układzie znacząco się nie zmieni, możemy określić teoretyczny czas obliczania eksponenty modulo – tabela 2.

Tab. 2. Porównanie czasów obliczania eksponenty
Tab. 2. Exponent calculation time comparison

Liczba bitów eksponenty [bit]	32 bit GPP[1] [ms]	FPGA[3] [ms]	64 bit GPP[2] [ms]
1024	9.4	5.3	2.7
2048	60	21	17.8

[1]-1916 MHz, 512 MB Ram
[2]-2210 MHz, 8 GB Ram
[3]Virtex 4 LX200

Porównanie czasów obliczania pojedynczej eksponenty modulo pokazuje, że dla zaproponowanej architektury przyspieszenie jest możliwe jedynie w stosunku do 32 bitowych procesorów ogólnego przeznaczenia. Istnieją jednak metody dalszych optymalizacji modułu w szczególności:

a) Optymalizacje zajętości powierzchni, umożliwiające zaimplementowanie kilku modułów jednocześnie dla równoległych obliczeń, co pozwoli na zwiększenie przepustowości układu
b) Zastosowanie wyższych podstaw obliczeniowych. Oznacza to, że zamiast analizować kolejne bity liczby a, w każdym z cykli analizowane są większe ich ilości, co pozwala na zmniejszenie liczby cykli obliczeniowych.

Otrzymane wyniki potwierdzają przydatność układów FPGA jako platformy przyśpieszającej obliczenia stosowane w algorytmach kryptograficznych. Jednocześnie wyniki wskazują na potrzebę i możliwość dalszych optymalizacji architektury dla szybszego jej działania.

Praca naukowa finansowana ze środków na naukę w roku 2008 jako projekt badawczy nr 3 T11C 031 30.

4. Literatura

[1] P. L. Montgomery. Modular Multiplication without Trivial Division. Mathematics of Computation, 1985 r.
[2] W. Diffie, M. E. Hellman. New Directions in Cryptography. IEEE Transactions on Infromation Theory, 1976 r.
[3] R. Rivest, A. Shamir, L. Adelman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Communications of the ACM, 1978 r.
[4] Tolga Acar, Çetin Kaya KoÇ, Burton S. Jr. Kaliski. Analyzing and Comparing Montgomery Multiplication Algorithms. IEEE Micro, 1996 r.
[5] C. D. Walter. Montgomery exponentiation needs no final substractions. Electronic Letters, 1999 r.
[6] C. McIvor, M. McLoone, J.V. McCanny. Modified Montgomery modular multiplication and RSA exponentiation techniques. Computers and Digital Techniques, IEE Proceedings, 2004 r.
[7] Thomas Blum, Christoph Paar. Montgomery modular exponentiation on reconfigurable hardware. 14th IEEE Symposium on Computer Arithmetic (ARITH-14), 1999 r.

Artykuł recenzowany