CRIPTOCORE: PROJETO, VALIDAÇÃO E PROTOTIPAÇÃO DE UM IP PARA APLICAÇÕES CRIPTOGRÁFICAS

Carlos R. T. Fernandes, Gabriel R. Laureano, Luiz F. P. Santos, Luiz C. V. dos Santos

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina
Florianópolis, SC, Brasil

{carlosrf, laureano, penkal, santos}@inf.ufsc.br

ABSTRACT

The rise of SoCs was the result of two interacting factors: the increasing supply of hardware ruled by Moore's Law and the demand of complex industrial applications, especially in the domains of consumer electronics and telecommunications. Platform-based design is the methodological response to the high costs of non-recurring engineering associated with contemporary technologies. Architectural reference, IP reuse and ESL are key mechanisms to cope with complexity and time-to-market. The need for productivity gains asks for IP models in different levels of abstraction. This paper describes the design, validation and prototyping of an IP in three distinct abstraction levels. The IP, called CriptoCore, implements modular exponentiation, a crucial operation used in important cryptographic applications. Experimental results give evidence of proper validation and allow the quantitative assessment of productivity gain.

1. INTRODUÇÃO

Systems-on-Chip (SoCs), que são sistemas dedicados de hardware e software, surgiram da interação de dois fatores [1]: a crescente oferta de hardware ditada pela Lei de Moore e a demanda de aplicações cada vez mais complexas, especialmente nas áreas de eletrônica de consumo e de telecomunicações.

O projeto de SoCs baseado em plataforma [2] é uma resposta metodológica aos elevados custos de engenharia não-recorrente das tecnologias contemporâneas e consiste em uma arquitetura de referência e no reuso de blocos de propriedade intelectural (IPs).

A diminuição do ciclo de desenvolvimento de produtos eletrônicos e a pressão do *time-to-market* requerem ganhos de produtividade crescentes. Para isso, o projeto de SoCs deve iniciar-se em níveis de abstração cada vez mais altos. Um exemplo desta tendência na indústria de EDA ("*Electronic Design Automation*") é a crescente oferta de ferramentas em nível ESL ("*Electronic System Level*") para complemetar o fluxo de projeto convencional, que se iniciava no nível RTL ("*Register Transfer Level*").

Assim, além dos vários requisitos impostos para garantir o reuso de IPs, faz-se necessário prover modelos de IPs em níveis mais abstratos que o RTL.

Este trabalho reporta o projeto, a validação e a prototipação de um IP em três níveis distintos de abstração. O IP denomina-se CriptoCore e implementa a exponenciação modular, uma operação crucial em importantes aplicações criptográficas.

A prototipação do CriptoCore concorreu no âmbito da Segunda Olimpíada Brasileira da Altera, tendo obtido a terceira colocação.

O restante deste artigo é organizado da seguinte forma. A Seção 2 revisa brevemente trabalhos correlatos. A Seção 3 descreve o projeto e a validação do IP, enquanto que a Seção 4 descreve sua prototipação. Os resultados experimentais são apresentados na Seção 5. A Seção 6 resume as conclusões e aponta direções para investigações futuras.

2. TRABALHOS CORRELATOS

Vários IPs para aplicações criptográficas são reportados na literatura [3, 4, 5]. A maior parte dos IPs implementa algoritmos criptográficos simétricos, tais como o DES [4] e o AES [5]. Há IPs que implementam operações freqüentemente utilizadas em algoritmos criptográfricos, como a exponenciação modular [3, 6].

Para realizar a exponenciação modular, todos os IPs encontrados pelos autores utilizam o algoritmo *Left-to-Right binary* [3]. Por sua vez, para realizar a multiplicação modular (operação primitiva utilizada pela exponenciação), a literatura reporta uma gama bastante extensa de algoritmos. Dentre os mais usados estão *Interleaved Standard Multiplication* [3], o Optimized *Interleaved Standard Multiplication* [6] e variações do algoritmo de Montgomery [7].

No entanto, em sua grande maioria, os IPs são descritos apenas no nível RT e poucos são disponibilizados ao domínio público.

A carência de IPs de domínio público descritos em vários níveis de abstração motivou a proposta de repositórios orientados ao desenvolvimento de plataformas, tais como os dos projetos BrazilIP[8] e ArchC [9] (especialmente em sua iminente versão 2.0).

Este trabalho apresenta uma contribuição técnica na forma de um IP sintetizável, mas descrito em três diferentes níveis de abstração, a ser doado para repositórios públicos, tais como os acima citados.

3. PROJETO E VALIDAÇÃO DO IP

3.1. Metodologia de projeto

Foi adotada uma metodologia *top-down* [10], que consiste em descrever a funcionalidade do IP em um alto nível de abstração e, através de refinamentos sucessivos, descrever o IP em níveis de abstração cada vez mais baixos até que se obtenha uma descrição sintetizável. Neste trabalho, são usados os seguintes níveis de descrição: algorítmico, funcional com precisão de bits e RT.

3.2. Descrições do IP

O IP realiza a operação de exponenciação modular:

,

freqüentemente utilizada em aplicações criptográficas, tais como encriptação RSA, algoritmo de Diffie-Hellman e assinatura digital [11].

O IP possui três entradas (Base, Expoente e Módulo) e uma saída (Resultado).

Entrada: X, e, M
Saída: $C = X^e \ mod \ M$
$e_i : i^{ésimo} \ bit \ de \ e$
n: número de bits em e
(1) $C := 1$
(2) $For \ \ n-1 \ downto \ 0 \ do \ \{$
(2a) $\quad C := C * C$
(2b) $\quad if \ e_i = 1 \ then \ C := C * X \ mod \ M \ \}$

Figura 1: Algoritmo para exponenciação modular

Para realizar a exponenciação modular, escolheu-se o algoritmo *Left-to-Right binary*, por ser o mais adequado à manipulação de números inteiros representados com um grande número de bits.

A Figura 1 descreve o algoritmo escolhido, onde **P** representa uma estimativa parcial da exponenciação modular a ser refinada iterativamente. Dada a representação binária do expoente **e**, seus bits são visitados do mais significativo (MSB) para o menos significativo (LSB), de forma que a estimativa corrente é elevada ao quadrado para cada bit visitado. A multiplicação modular é invocada somente se o bit visitado for "1".

Para realizar a multiplicação modular no passo 2a da Figura 1 escolheu-se o algoritmo *Standard Interleaved*, por razões de praticidade.

A Figura 2 descreve o algoritmo escolhido, onde ***P*** representa uma estimativa parcial do produto modular. Dada a representação binária da base **X**, seus bits são visitados do MSB para o LSB, de forma que uma nova estimativa **P** é obtida a cada bit $\mathbf{X_i}$ visitado. Para obter uma estimativa parcial, a idéia é intercalar multiplicação e módulo de maneira que os resultados intermediários sejam mantidos tão pequenos quanto possível. O passo 3 garante o correto alinhamento dos produtos parciais: o produto acumulado na iteração anterior é o dobro do produto parcial calculado na iteração atual, pois os bits são visitados do MSB para o LSB. O passo 4 realiza um produto parcial e o acumula. Os passos 5 e 6 aplicam a operação de módulo ao produto acumulado, dele subtraindo o valor **M** até se obter ***P < M***.

Entrada: X, M, Y com $X, Y, M \geq 0$
Saída: $P = (X * Y) \ mod \ M$
$X_i : i^{ésimo} \ bit \ de \ X$
n: número de bits em X
(1) $P := 0$
(2) $For \ \ n-1 \ downto \ 0 \ do \ \{$
(3) $\quad P := 2 * P$
(4) $\quad P := P + X_i * Y$
(5) $\quad While \ (P \geq M)$
(6) $\quad\quad P := P - M \ \}$

Figura 2: Algoritmo para multiplicação modular

3.2.1 Descrição algorítmica

Neste nível o IP tem seu algoritmo implementado em alto nível. A comunicação entre os módulos e a representação interna dos números ainda não leva em conta efeitos de representação finita. As entradas e saídas do IP são modeladas em SystemC [12] através de canais do tipo FIFO. Sempre que é detectada a chegada de dados nas entradas, uma *thread* é disparada, a qual executa o algoritmo da exponenciação modular. Após isso, o resultado é disponibilizado na saída.

3.2.2 Descrição funcional
Neste nível o módulo que representa o Criptocore já passa a ter precisão de bits, ou seja, a representação dos dados será feita com o número de bits que será efetivamente usado no sistema real. Essencialmente, a descrição funcional é um refinamento da descrição algorítimica em que a nova representação númerica é adotada.

3.2.3 Descrição RT
No nível RT, a estrutura do IP é descrita em termos de seus componentes básicos (tais como somador, registradores, deslocadores, etc.), os quais realizam o comportamento do algoritmo, dirigidos por uma unidade de controle. Esboços dos principais blocos do IP são apresentados nas Figuras 3 e 4.

A Figura 3 mostra os principais componentes do multiplicador modular. Vale ressaltar que o *shifter* implementa o passo 3 do algoritmo da Figura 2, enquanto o somador é usado para implementar os passos 5 e 7. O registrador armazena as estimativas parciais **P** da mutiplicação modular.

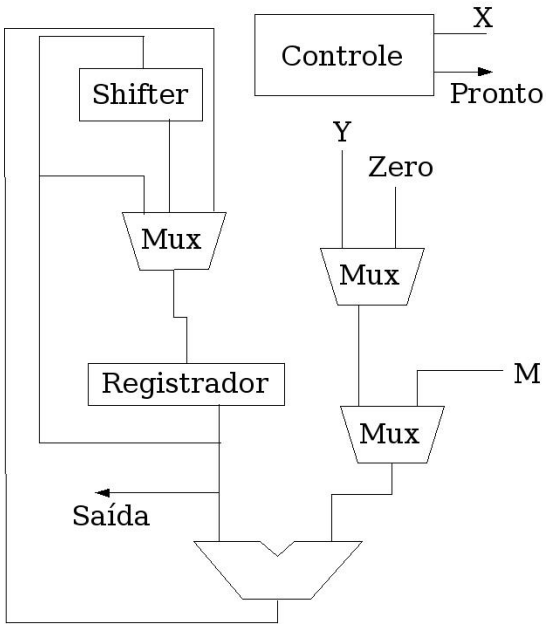

Figura 3: Estrutura do multiplicador modular

Por sua vez, a Figura 4 ilustra os principais componentes do exponenciador modular. O bloco **`ModMul`** encapsula o circuito da Figura 3, enquanto o registrador armazena as estimativas parciais **P** da exponenciação modular.

Ao contrário das descrições anteriores, a comunicação entre os módulos é modelada através de sinais em SystemC. Por isso, novos canais foram criados para que a unidade de controle pudesse gerenciar a comunicação entre módulos.

Nesta fase de projeto, percebeu-se a necessidade de criar duas entradas e uma nova saída na interface: uma entrada para comandar o início de uma operação no IP, uma entrada para o *clock* e uma saída para sinalizar a disponibilidade do resultado.

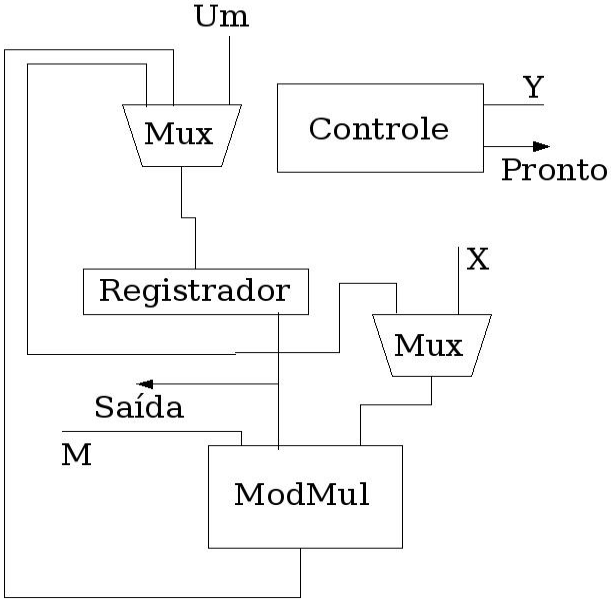

Figura 4: Estrutura do exponenciador modular

3.2.4 Tradução para VHDL
Por não se dispor de uma ferramenta que sintetize uma descrição SystemC em nível RT, uma tradução manual para VHDL teve que ser feita. Como não há diferença conceitual entre as duas descrições, todos os canais e módulos mantiveram-se inalterados. Apenas mudanças sintáticas foram necessárias.

3.3. Metodologia de validação
Dois módulos adicionais foram criados para fazer a validação do IP. O módulo STIMGEN que lê valores em um arquivo e os repassa ao CriptoCore e o módulo MONITOR que salva os resultados gerados pelo CriptoCore num arquivo. A estrutura de validação é ilustrada na Figura 5.

Figura 5: Estrutura de módulos para validação

A descrição algorítmica é a implementação em alto nível do algoritmo que implementa a funcionalidade do IP.

Por hipótese, assumimos a corretude da descrição algorítmica, para poder validar os demais. Um vetor de estímulos é aplicado ao modelo e seus resultados servem de referência para a descrição de nível inferior.

A partir da descrição funcional, cada descrição é submetida ao mesmo vetor de estímulos e seus resultados comparados com a descrição no nível imediatamente superior. Uma vez que o vetor de resultados é igual ao nível superior, a descrição é considerada validada. Os resultados produzidos pelo CriptoCore em todos os níveis de descrição foram exatamente os mesmos.

4. PROTOTIPAÇÃO

4.1. A plataforma de prototipação

A plataforma de que se dispunha para prototipação (veja Seção 5.2.1) permite o uso de um *softcore*, a CPU NIOS I, e a conexão de IPs é feita através do barramento AVALON. Para permitir a interação com a CPU, um *wrapper* foi criado para o CriptCore, compatibilizando-o com o protocolo daquele barramento.

4.2. Driver de acesso ao IP

Um *driver* de acesso ao CriptoCore foi construído para viabilizar sua interação com a CPU. Como o IP está diretamente conectado ao barramento, o *driver* consiste essencialmente na criação de ponteiros que representam o endereço associado ao CriptoCore no barramento. O *driver* possui três métodos de acesso ao IP:

- **init(int x, int y, int m)**: Inicia o IP com os valores passados como parâmetro.
- **isReady()**: Retorna 1 quando o resultado já está disponível na saída.
- **getOutput():** Retorna o resultado da operação.

Após a inicialização do CriptoCore, são necessários alguns ciclos de *busy waiting*, enquanto aguarda-se a disponibilidade do resultado, para então capturá-lo.

4.3. Restrições impostas pela plataforma-alvo

As descrições do CriptoCore nos três níveis de abstração adotados permitem a representação de valores com um número genérico n de bits. Preferencialmente, o número de bits deveria ser grande (e.g. n = 1024 bits) para uso em aplicações reais. Entretanto, considerando a exigüidade de elementos lógicos na placa disponibilizada como plataforma-alvo, a representação adotada nas descrições restringe-se a n=32 bits, inclusive as entradas e saídas do IP. Note que esta é uma restrição imposta exclusivamente para fins de prototipação, sem perda da generalidade das descrições do CriptoCore.

5. RESULTADOS EXPERIMENTAIS

5.1. Experimentos através de simulação

5.1.1 Configuração

Todos os experimentos foram executados em um notebook Toshiba com processador Intel 1.7GHz, sob o sistema operacional Gentoo Linux (kernel-2.6.12). Para modelagem e compilação utilizaram-se SystemC-2.0.1 e GCC-3.3.5. As simulações VHDL foram feitas noa ferramenta Quartus-4.2.

5.1.2 Procedimento

Gerou-se um arquivo com 3 milhões de números aleatórios de 32 bits. Estes números servem de entrada para o módulo de estímulo, que irá atribuir o primeiro número do arquivo a X, o segundo a Y e o terceiro a M até que se esgotem os números do arquivo. Com isso o módulo que implementa o IP foi estimulado 1 milhão de vezes.

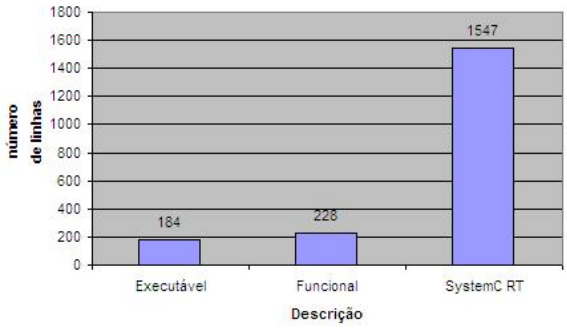

Figura 6: Complexidade das descrições

5.1.3 Resultados

Os resultados experimentais são ilustrados nas Figuras 6 e 7. A Figura 6 ilustra a complexidade das descrições em SystemC, expressa em número de linhas, enquanto a Figura 7 mostra os tempos de simulação das descrições em SytemC e o tempo de execução no protótipo.

Como esperado, os tempos de simulação aumentam à medida que o nível de abstração das descrições diminui.

Note que a simulação funcional é cerca de 56 vezes mais lenta que a algorítmica, embora não haja diferença sensível em sua complexidade. Isto se deve à utilização do tipo *fixed point* de SystemC para modelar os efeitos de precisão finita. Note que a simulação no nível RT é cerca de 5 vezes mais lenta do que a simulação no nível funcional, dando uma evidência quantitativa do ganho de produtividade potencial ao se trabalhar em níveis mais altos de abstração.

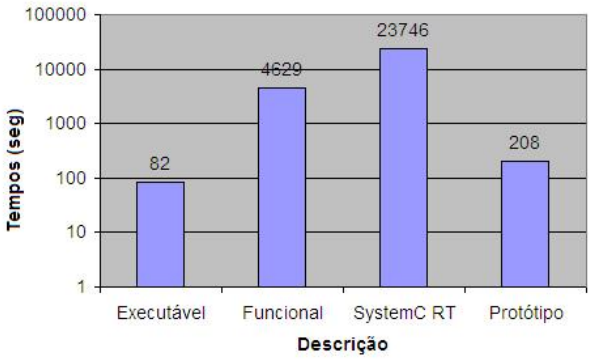

Figura 4: Tempos de simulação/execução

5.2. Experimentos com o protótipo

5.2.1 Configuração

O CriptoCore foi implementado em uma plataforma de prototipação da Altera, modelo APEX20K [13]. A freqüência de relógio utilizada foi de 33MHz. Os dados de estímulo foram enviados à CPU NIOS através da porta serial da plataforma de prototipação.

5.2.2 Procedimento

Devido a uma dificuldade técnica de controlabilidade, não foi possível capturar o tempo do protótipo com exatamente os mesmo estímulos utilizados nas simulações. O tempo que aparece na Figura 7 é o tempo que o protótipo demorou para executar o mesmo número de operações do arquivo de estímulos usado nas simulações das descrições.

5.2.3 Resultados

Conforme ilustra a Figura 7, o tempo de execução no protótipo é cerca de 2,5 vezes maior que o tempo de simulação da descrição algorítmica. Isso se deve ao fato de tal descrição ser bastante abstrata e executar numa máquina hospedeira operando a uma freqüência muito mais alta que a da plataforma-alvo.

6. CONCLUSÃO E TRABALHOS FUTUROS

Os resultados experimentais dão claras evidências da corretude e da robustez do IP projetado, cuja funcionalidade foi validada submetendo-o a um milhão de estímulos gerados aleatoriamente.

Ademais, os resultados permitem estimar um limite inferior para o ganho de produtividade potencial obtido com a utilização de níveis mais abstratos de modelagem. Por exemplo, para o CriptoCore, o ganho de produtividade da simulação entre os níveis funcional e RT é cerca de 5. O ganho de produtividade total é provavelmente bem maior, considerando-se a maior dificuldade de codificação e depuração da descrição RT.

O CriptoCore encontra-se validado e pretende-se disponibilizá-lo ao domínio público no futuro próximo. Para isso, será preciso submetê-lo a um processo de certificação preliminar à sua integração em repositório de domínio público, como por exemplo o do Projeto BrazilIP ou o do Projeto ArchC, versão 2.0.

A prototipação do CriptoCore concorreu no âmbito da Segunda Olimpíada Brasileira da Altera, tendo obtido a terceira colocação.

REFERÊNCIAS

[1] BERGAMASCHI, Reinaldo. **A to Z of SoCs**. Tutorial apresentado na Escola de Microeletrônica da SBC Sul (EMICRO 2002), Florianópolis, Brazil, 2002.

[2] SANGIOVANNI-VINCENTELLI, A., MARTIN, G, "*Platform-Based Design and Software Design Methodology for Embedded Systems*". **IEEE Design & Test of Computers**, v. 18, n.6, p.23-33. November-December, 2001.

[3] KOç Çetin K. **RSA Hardware implementation**. [S.l], 1995.

[4] HOORNAERT, Frank. GOUBERT, Jo. DESMEDT, Yvo. **Efficient Hardware Implementation of the DES**. In: Advances in Cryptology: Proceedings of CRYPTO 84. v. 196. p.147. 1985.

[5] ELBIRT, AJ. YIP, W. CHETWYND, B. PAAR, C. **An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists**.

[6] BONIMOV, M. S. V. **Optimized Interleaved algorithm**. In: The IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'03). [S.l.: s.n.], 2003.

[7] AMANOR, D. N. **Efficient Hardware Architectures for Modular Multiplication**. MSc. Dissertation — The University of Applied Sciences Offenburg, Germany, 2005.

[8] **BrazilIP**.Disponível em http://www.brazilip.org.br

[9]**The ArchC Architectural Description Language**. Disponível em http://www.archc.org.

[10] WOLF, Wayne Hendrix. **Computers as components: principles of embedded computing system design**. San Francisco: Morgan Kaufmann, 2001.

[11] STALLINGS, William. **Cryptography and network security: principles and practice**. 2nd ed. Upper Saddle River: Prentice Hall, 1998.

[12] SystemC Community. **SystemC Home Page**, Disponível em: http://www.systemc.org. 2003.

[13] ALTERA. **APEX 20K Programmable Logic Device Family Data Sheet**. March, 2004.