Otimização em VHDL e Desempenho em FPGAs do Algoritmo de Criptografia DES

Fábio Dacêncio Pereira, Edward David Moreno
Fundação Eurípides de Marília
fabiopereira@fundanet.br, edmoreno@fundanet.br

Resumo

A segurança nas transmissões de informações torna-se cada vez mais importante, exigindo que novas técnicas e algoritmos de criptografia de informações sejam desenvolvidos para promover um ambiente de transmissão seguro. Para muitas aplicações não só a segurança é prioridade no fundamento do algoritmo utilizado, mas também a velocidade do processo de cifragem e decifragem.

Este artigo destaca o algoritmo de criptografia DES e sua descrição em VHDL. O Data Encryption Standard (DES) - é um padrão criptográfico criado em 1977 através de uma licitação aberta pela antiga Agência Nacional de Segurança americana - National Security Agency (NSA).

Neste artigo, é definido seu funcionamento, discutindo as metodologias de implementação em hardware, propondo uma otimização para a metodologia mais utilizada por projetistas hardware. Os resultados obtidos são importantes para a conclusão da primeira fase do projeto de um criptoprocessador.

Estatísticas de desempenho temporal e espacial são geradas para a comparação entre as metodologias, discutindo a otimização obtida.

1. Introdução

A segurança nas transmissões de informações torna-se cada vez mais importante, exigindo que novas técnicas e algoritmos de criptografia de informações sejam desenvolvidos para promover um ambiente de transmissão seguro. Para muitas aplicações não só a segurança é prioridade no fundamento do algoritmo utilizado, mas também a velocidade do processo do cifragem/decifragem. Portanto um bom sistema criptográfico deve ser robusto, seguro e veloz, sendo que a última característica, a velocidade, é altamente dependente do ambiente e da metodologia de implementação.

A implementação de um sistema criptográfico em um hardware de alto desempenho como FPGA, juntamente com técnicas de otimização de circuitos digitais, promovem teoricamente um ambiente bastante favorável para o desenvolvimento de sistemas criptográficos capazes de alcançar um bom nível de desempenho funcional e temporal a um baixo custo.

Desenvolver um novo algoritmo de criptografia demanda tempo e custo elevados, além de um complicado e demorado processo para que este torne um padrão e atinja o mercado. Ao desenvolver um algoritmo criptográfico algumas das características analisadas são o nível de segurança e a velocidade de processamento. Assim um determinado algoritmo tem melhor aplicabilidade do que outro, mesmo que este tenha um nível de segurança menor.

Este artigo apresenta algumas otimizações na descrição VHDL do algoritmo de criptografia DES. Os resultados obtidos são importantes para a conclusão da primeira fase do projeto de um criptoprocessador, isto é, um processador capaz de executar algoritmos criptográficos. Algumas características diferenciais foram adicionadas a este criptoprocessador, como a capacidade de ser adaptativo a diferentes configurações impostas pelo usuário ou pelo ambiente de funcionamento, detalhes este que serão discutidos neste artigo.

A arquitetura e o conjunto de instruções do criptoprocessador são modelados especialmente para a execução de instruções específicas dos algoritmos criptográficos, eliminando parte das instruções encontradas em processadores de uso geral. Este tipo de processador teoricamente atinge um melhor desempenho que processadores de uso geral, pois podem ser utilizadas com maior facilidade, algumas técnicas de processamento especulativo e outros artifícios para aumentar a performance do sistema. Neste caso o criptoprocessador utilizará técnicas de *pipeline* e possuirá uma arquitetura adaptada para a execução de diferentes algoritmos criptográficos, entre eles o DES, AES, RC5, IDEA, RSA.

O projeto do criptoprocessador pode ser dividido em quatro fases principais: (i) a descrição algorítmica em VHDL dos principais algoritmos criptográficos, (ii) a implementação em software, utilizando a linguagem C, (iii) o projeto, a descrição em VHDL e a implementação em FPGAs do criptoprocessador e por fim, (iv) a análise de desempenho do criptoprocessador.

Para avaliar e classificar o criptoprocessador são necessários parâmetros de comparação. Desta forma, perguntas como: quanto mais rápido é o criptoprocessador em relação a uma implementação em software? quanto mais rápido é em relação a uma implementação algorítmica em FPGA? Teremos estas respostas apenas na conclusão deste projeto. Mas alguns resultados importantes já foram alcançados, um deles, será apresentado neste artigo, especificamente à otimização na descrição VHDL do algoritmo de criptografia DES.

2. O criptoprocessador

Nesta sessão, apresentam-se algumas características do criptoprocessador com o objetivo esclarecer e melhor visualizar o projeto como um todo.

O processador é um circuito lógico responsável por executar instruções predefinidas armazenadas em uma memória. Na história da computação, o processador esteve sempre presente. Com a evolução da microeletrônica, propostas de novas arquiteturas surgiram e os processadores atuais possuem uma grande capacidade de processamento.

Propostas de técnicas para aumentar a performance de processadores são baseadas na exploração do paralelismo em nível de instruções. Dentre estas técnicas, destaca-se o pipeline, a execução superescalar, a execução fora-de-ordem e a execução especulativa. Ainda existem as técnicas que exploram o paralelismo em nível de threads ou processos, como a multithread e a multithread simultânea. Independente da técnica utilizada, o processamento sempre executa um ciclo básico, que seria a busca, decodificação e execução de instruções.

Os processadores atuais são divididos em diversas categorias, dependendo da sua finalidade, sendo os principais grupos: (i) os processadores de uso geral, dentre os quais destaca-se o Pentium, o Athlon, o Sparc, o MIPS, o PowerPC, entre outros, (ii) os processadores digitais de sinais genéricos, tais como a família TMS320, o ADSP21020, o DSP32, entre outros (iii) os microcontroladores, como o 8051, PIC, entre outros (iv) os processadores embarcados, com co-processadores matemáticos e (v) os processadores dedicados, como processadores de vídeo, áudio e criptoprocessadores para aplicações específicas.

Neste artigo, destaca-se a implementação de um criptoprocessador. O criptoprocessador consiste num processador dedicado para a execução de algoritmos criptográficos. Os algoritmos inicialmente destacados para implementação são o DES, AES, RC5, IDEA e o RSA.

Observe que a maioria dos algoritmos selecionados são simétricos e apenas um assimétrico, o RSA. Este último será utilizado especialmente para troca de chaves entre as partes comunicantes, já que a literatura mostra que os algoritmos de chave pública oferecem uma maior segurança e comodidade para a distribuição de chaves.

A definição de um conjunto de instruções e uma arquitetura que atenda a maioria dos algoritmos disponíveis é uma tarefa complicada, pois exige o conhecimento de detalhes de cada um dos algoritmos, extraindo assim, as operações realizadas com maior freqüência e instruções exclusivas de cada um deles.

Justificando a implementação em software e a implementação algorítmica em FPGA, pois, destas implementações serão extraídos detalhes importantes para definição do conjunto de instruções e arquitetura do criptoprocessador. De qualquer forma, a descrição geral do projeto pode ser visualizada na figura 1.

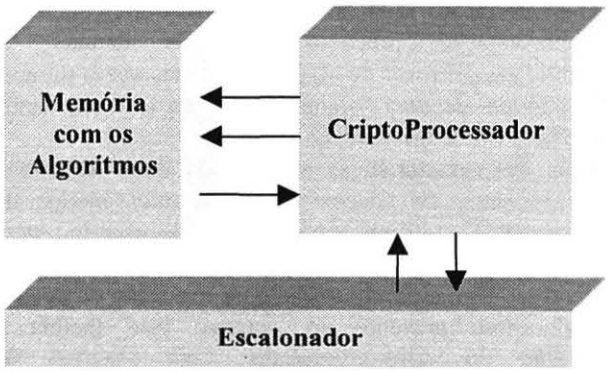

Figura 1 – Descrição geral do criptoprocessador

Como destacado na introdução este criptoprocessador possui características particulares como a capacidade de se adaptar a configurações impostas pelo usuário. Essas configurações serão realizadas via software e o usuário terá opções como:

- **Selecionar o algoritmo que será executado** (DES, AES, RC5, IDEA ou RSA): nesta opção o usuário pode configurar com qual algoritmo deseja executar, tanto para o processo de cifragem com para decifragem.
- **Escalonamento programado**: esta configuração permite o usuário selecionar dentre os algoritmos disponíveis quais ele deseja que o criptoprocessador execute em modo escalonado, isto é, a cada quantidade X bytes cifrados ou Y segundos o algoritmo será trocado por outro da lista.
- **Escalonamento aleatório**: esta configuração faz com que o criptoprocessador selecione aleatoriamente qual será o algoritmo utilizado.

Fica claro que as configurações de escalonamento programado e aleatório só serão bem sucedidas se as

partes comunicantes estiverem utilizando o criptoprocessador.

Interno ao escalonador existe um PRNG que auxiliará no processo de escalonamento aleatório. O PRNG é o gerador de números pseudo-aleatórios, isto é, alimentando o PRNG com dados, chamados de sementes, é gerado uma seqüência de números aleatório ou melhor pseudo-aleatórios, já que a seqüência é gerada através de uma fórmula matemática. Esta seqüência aleatória definirá qual será o algoritmo que será executado pelo criptoprocessador.

Para o sistema como um todo funcionar é necessário que as partes comunicantes estejam preparadas e configuradas no mesmo modo de funcionamento. Para isso, ocorre uma troca de mensagens entre os sistemas, configurando, assim, os parâmetros de funcionamento.

O escalonador é uma das características diferenciais do criptoprocessador proposto para os demais. A possibilidade de trocar de algoritmos várias vezes durante a transmissão de uma mesma mensagem dificulta ainda mais a invasão da privacidade alheia.

Uma das características importantes deste módulo é sua capacidade de adaptar-se a diferentes modos de funcionamento conforme a necessidade do usuário. Além dos dois modos de funcionamento do escalonador é possível também a configuração para execução de apenas um algoritmo presente no sistema. Isso facilita a integração do criptoprocessador com sistemas de criptografia existentes.

3. Algoritmo DES

O *Data Encryption Standard* (DES) [2] - é um padrão criptográfico criado em 1977 através de uma licitação aberta pela antiga Agência Nacional de Segurança americana - *National Security Agency* (NSA).

O DES é classificado com um algoritmo de criptografia simétrico, ou seja, a chave utilizada na cifragem é a mesma da decifragem de um texto. A estrutura do funcionamento do DES pode ser dividida em três partes: permutação inicial, iterações e permutação final.

As permutações iniciais e finais são processos de transposição dos blocos de entrada, executando a leitura da esquerda para direita. Já as iterações são operações que utilizam subchaves sobre o texto, repetindo 16 vezes a mesma operação. Inicialmente o bloco de entrada e a chave são divididos em duas partes de mesmo tamanho, executando processos de permutação, transformação e expansão de chave. A cada bloco executado, uma nova subchave é gerada a partir da chave original.

Tanto no processo de cifragem como na decifragem do DES é utilizado o mesmo algoritmo. Apenas muda a ordem de aplicação das chaves, isto é, para cifrar, deve-se utilizar as subchaves 1 a 16 correspondentes a cada iteração. Para decifrar deve-se utilizar as chaves na ordem inversa, de 16 a 1 respectivamente para cada iteração. O esquema de funcionamento do DES encontra-se na figura 2.

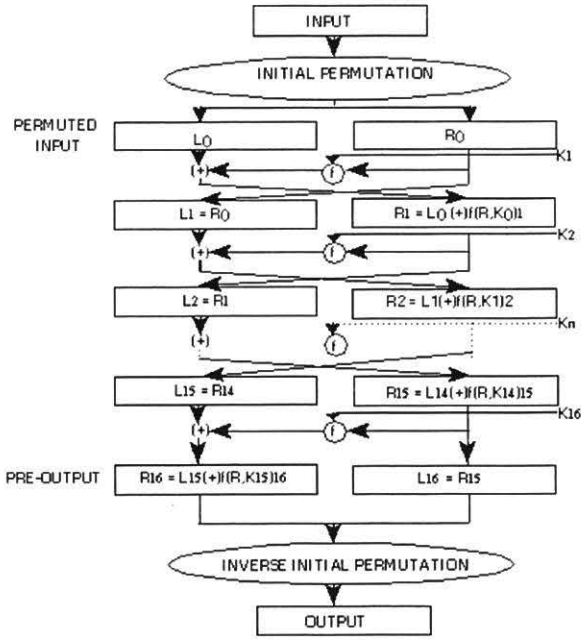

Figura 2. Esquema de funcionamento do DES

4. Descrição VHDL do DES

A otimização de um circuito digital está relacionada a diversos aspectos, que envolvem desde a otimização temporal e espacial até consumo de energia e custo do circuito projetado.

Focalizando os aspectos temporal e espacial, sabe-se que a busca por um circuito veloz e cada vezes menor é o desejo de qualquer projetista de hardware. Para alcançar este objetivo utilizam-se várias técnicas para aumentar a performance explorando principalmente o paralelismo do hardware. No caso de processadores existe algumas técnicas que foram citadas na sessão 2. Definir qual a melhor técnica para sua aplicação pode ter reflexos na otimização do hardware implementado.

No processamento do algoritmo de criptografia DES em hardware, a geração das dezesseis subchaves é o maior gargalo de todo processamento. Muitos projetistas descrevem o módulo de geração de subchaves de forma a executar em paralelo ao processamento principal. Observe a figura 3.

Esta técnica otimiza o tempo de execução do algoritmo DES, já que a execução do módulo principal

trata apenas das dezesseis iterações do algoritmo, requisitando sempre a cada iteração a próxima subchave a ser utilizada; ficando claro que existe um sincronismo entre a execução principal e a geração de subchaves.

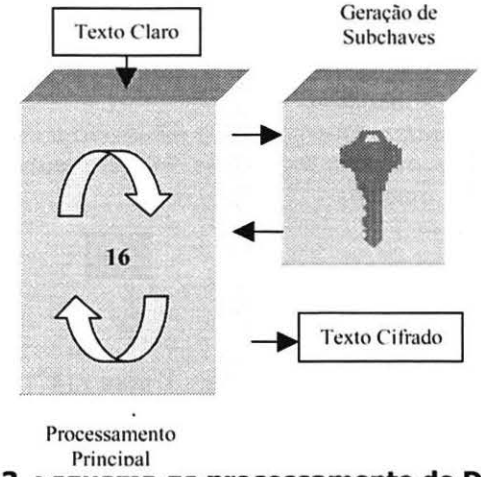

Figura 3. Esquema de processamento do DES

A otimização do algoritmo de geração de subchaves pode melhorar de forma significativa o processamento do DES, pois mesmo com a execução paralela, a geração de subchaves é um processo lento. Para isso, deve-se conhecer detalhes do algoritmo de geração de subchaves.

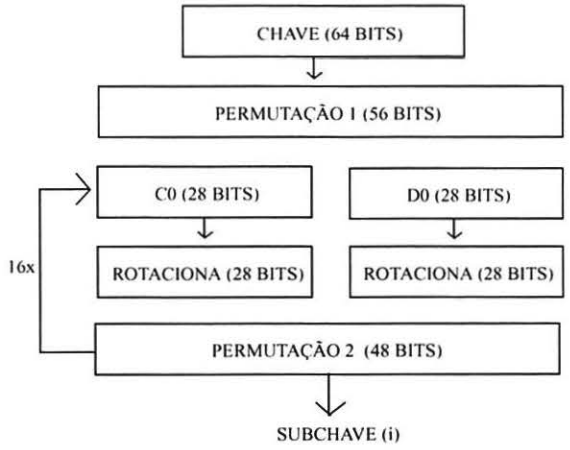

Figura 4. Algoritmo de geração de subchaves do DES

Este algoritmo tem passos simples, mostrados na figura 4. Os objetivos básicos é a difusão e a confusão da mensagem cifrada. A difusão visa eliminar a redundância existente na mensagem original, distribuindo-a pela mensagem cifrada. O propósito da confusão é tornar a relação entre a mensagem e a chave tão complexa quanto possível, de forma a impedir a dedução da chave a partir de características especiais da mensagem.

Para gerar as subchaves realiza-se uma permutação inicial sobre a chave original, dividindo-a posteriormente em duas metades de 28 bits. Estas metades são rotacionadas à esquerda um ou dois bits, conforme a subchave a ser gerada. Finalmente é executada uma permutação de compressão gerando a subchave desejada.

O mais importante a observar é que neste processo todos os bits da subchave têm uma correspondência direta com os bits da chave original, isto é, após a execução do algoritmo gerador de subchaves, cada bit da chave original foi deslocado para diferentes posições, gerando a subchave desejada. Por exemplo: O bit 1 da chave original foi deslocado para o bit 40 na subchave 1 e para o bit 30 na subchave 2. Assim, cada subchave é formada por bits da chave original.

Percebendo isso, os projetistas montaram uma tabela onde se demostrava para cada subchave quais são os bits correspondentes da chave original, eliminando assim a necessidade anterior de implementar operações como as duas permutações e as rotações para a geração de subchaves, bastando agora uma simples permutação para gerar as subchaves. Isto melhorou de forma significativa este processo.

Uma das propostas deste artigo é apresentar uma forma otimizada da geração de subchaves. A descrição VHDL deste algoritmo pode ser feita de várias maneiras, refletindo diretamente no desempenho do circuito final. A seguir, a entidade do módulo gerador de subchaves do DES:

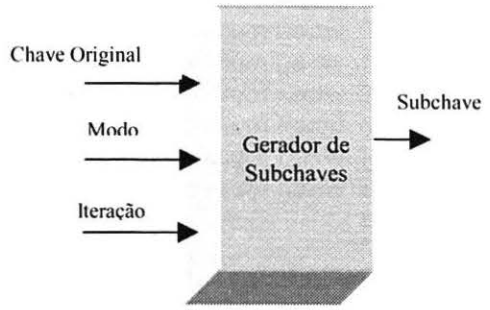

Figura 5 – Módulo gerador de Subchaves

Na figura 5, demonstra-se a interface do módulo gerador de subchaves. Onde se tem como parâmetros de entrada a iteração que está sendo executada, a chave original de 64 bits e o modo de execução (cifragem ou decifragem) e, como saída, as subchaves geradas. O funcionamento é simples: o processo principal define o modo de execução, a chave original e a iteração que está sendo executada. Assim subchave é gerada através de uma lógica combinacional.

O que esta lógica combinacional faz? Primeiramente define qual subchave será gerada. Tsto é possível através do modo de execução (cifragem/decifragem) e da iteração do processo principal. Por exemplo: Se a iteração for 1 e o modo for cifragem deve-se gerar a subchave 1, mas se o modo for decifragem deve-se gerar a subchave 16.

Isto acontece porque o processo de cifragem utiliza as subchaves de 1 a 16, então a iteração 1 utiliza a subchave 1 a iteração 2 a subchave 2 e assim por diante. Já no processo de decifragem as subchaves são utilizadas na ordem inversa: de 16 a 1; assim a iteração 1 irá utilizar a subchave 16, a iteração 2, a chave 15, assim por diante. A seguir, mostra-se um trecho da descrição em VHDL [1] mais utilizada pelos projetistas.

```
-- seleciona a subchave desejada conforme a
     iteracao.
-- a saida "chave" recebe a subchave Ki
     correspondente
-- a iteracao (round) que esta sendo executada.

        chave <=
              k1  when round="0000" else
              k2  when round="0001" else
              k3  when round="0010" else
              k4  when round="0011" else
              k5  when round="0100" else
              k6  when round="0101" else
              k7  when round="0110" else
              k8  when round="0111" else
              k9  when round="1000" else
              k10 when round="1001" else
              k11 when round="1010" else
              k12 when round="1011" else
              k13 when round="1100" else
              k14 when round="1101" else
              k15 when round="1110" else
              k16 ;

-- Este trecho Gera todas as subchaves Ki
-- obedecendo a ordem imposta pelo processo
-- de cifragem ou decifragem.
-- onde K1 a K16 são as subchaves, K a chave
     original
-- e sel define o modo cifragem/decifagem

K16(0) <= K(47) when sel='1' else K(40); --
subchave 16
K16(1) <= K(11) when sel='1' else K(04);
K16(2) <= K(26) when sel='1' else K(19);
K16(3) <= K(03) when sel='1' else K(53);
K16(4) <= K(13) when sel='1' else K(06);
K16(5) <= K(41) when sel='1' else K(34);
K16(6) <= K(27) when sel='1' else K(20);
K16(7) <= K(06) when sel='1' else K(24);
K16(8) <= K(54) when sel='1' else K(47);
 ...
K16(46) <= K(07) when sel='1' else K(00);
K16(47) <= K(28) when sel='1' else K(21);

-- Se o modo (sel) for cifragem, K16 receberá a
     subchave
-- 16 senão receberá a subchave 1, gerando as
     subchaves
-- na ordem inversa, como explicado anteriormente.
-- Da mesma forma, as demais subchaves são
     geradas.

K15 ...
 ...
K14 ...
...
K1(0) <= K(40)  when sel='1' else K(47);
K1(1) <= K(04)  when sel='1' else K(11);
K1(2) <= K(19)  when sel='1' else K(26);
K1(3) <= K(53)  when sel='1' else K(03);
K1(4) <= K(06)  when sel='1' else K(13);
...
K1(46) <= K(00) when sel='1' else K(07);
K1(47) <= K(21) when sel='1' else K(28);
```

Neste trecho de código VHDL percebe-se que foi bem aplicada a metodologia descrita anteriormente, onde os bits da chavé original são permutados para formar a subchave desejada. Para gerar as subchaves, foi utilizada a expressão:

Ki (j) = K(x) when sel='1' else K(y),

$0 \leq j \leq 47$ e $1 \leq i \leq 16$

Onde *Ki* é a subchave gerada, *K*, a chave original, *x* e *y*, número do bit da chave original a ser permutado e *sel* o modo cifragem ou decifragem. Assim, bit a bit, as subchaves são geradas.

Depois de geradas as subchaves, basta selecionar a subchave correspondente à iteração que está sendo executada.

Desta forma, algumas conclusões podem ser exploradas. Sabendo que através da expressão acima será gerada, bit a bit, cada subchave, pode-se concluir:

Se uma subchave tem 48 bits, e no total são 16 subchaves, são necessárias 768 expressões (48*16) para gerar todas as subchaves.

Isso torna o código VHDL extenso, mas esta metodologia é mais otimizada que a geração de subchaves através de fórmulas matemáticas.

5. Nova proposta para a metodologia de geração de subchaves do DES

Na sessão 4, ficou claro como é robusta a geração de subchaves. Neste contexto, observando o código VHDL e analisando as temporizações geradas pela ferramenta de síntese foram realizadas otimizações consideráveis nesta metodologia.

Fixando que o objetivo principal é a otimização da temporização e da ocupação deste algoritmo em um FPGA, o passo mais importante seria a redução do número de expressões para gerar as subchaves. Como fazer isto?

A resposta para esta pergunta pode ser obtida através de uma observação bastante simples. Observando o trecho de código VHDL anterior, nota-se que:

Se o modo (sel) for cifragem, K16 receberá a subchave 16, K15, a subchave 15. Caso o modo seja decifragem K16 e K15 receberão respectivamente as subchaves 1 e 2. Desta forma, sempre Ki pode gerar uma subchave i ou uma subchave 17-i, camada de subchave complementar.

Assim tanto K1 como K16 podem gerar as subchaves 1 e 16, tanto K2 como K15 podem receber as subchaves 2 e 15. Com esta observação, nota-se que existe um redundância nesta metodologia. Portanto não tem a necessidade de descrever K1 e K16, pois ambos geram as mesmas subchaves 1 e 16.

Desta forma a proposta para geração de subchaves substituiria o par K1 e K16 por K1_16, o par K2 e K15 por K2_15, assim por diante. Esta otimização pode ser observada no trecho de código VHDL a seguir

```
-- "modo" auxilia no processo de seleção de subchaves.
-- o valor de "modo" é definido pela operação cifragem
-- ou decifragem (sel) e pelo bit mais significativo do
-- número da iteração (round).

modo<='1' when round(3)='0' and sel='0' else
        '1' when sel='1' and round(3)='1' else
        '0';

chave <=
   k1_16  when round="0000" else
   k2_15  when round="0001" else
   k3_14  when round="0010" else
   k4_13  when round="0011" else
   k5_12  when round="0100" else
   k6_11  when round="0101" else
   k7_10  when round="0110" else
   k8_9   when round="0111" else
   K8_9   when round="1000" else
   K7_10  when round="1001" else
   K6_11  when round="1010" else
   K5_12  when round="1011" else
   K4_13  when round="1100" else
   K3_14  when round="1101" else
   K2_15  when round="1110" else
   K1_16 ;

-- gera as de subchaves 1 ou 16
k1_16(0) <= K(47) when modo='1' else K(40);
k1_16(1) <= K(11) when modo='1' else K(04);
k1_16(2) <= K(26) when modo='1' else K(19);
k1_16(3) <= K(03) when modo='1' else K(53);
k1_16(4) <= K(13) when modo='1' else K(06);
k1_16(5) <= K(41) when modo='1' else K(34);
k1_16(6) <= K(27) when modo='1' else K(20);
k1_16(7) <= K(06) when modo='1' else K(24);
...
k1_16(46) <= K(7) when modo='1' else K(0);
k1_16(47) <= K(28) when modo='1' else K(21);
...
K2_15... -- gera as de subchaves 2 ou 15
...
K3_14... -- gera as de subchaves 3 ou 14
...
-- gera as de subchaves 8 ou 9
k8_9(0) <= K(24) when modo='1' else K(06);
k8_9(1) <= K(20) when modo='1' else K(27);
k8_9(2) <= K(03) when modo='1' else K(10);
k8_9(3) <= K(12) when modo='1' else K(19);
k8_9(4) <= K(47) when modo='1' else K(54);
k8_9(5) <= K(18) when modo='1' else K(25);
k8_9(6) <= K(04) when modo='1' else K(11);
k8_9(7) <= K(40) when modo='1' else K(47);
...
k8_9(46) <= K(43) when modo='1' else K(50);
k8_9(47) <= K(09) when modo='1' else K(16);
```

Qual é o ganho obtido com esta alteração? Anteriormente para gerar todas as subchaves eram necessárias 768 expressões do tipo **Ki (j) = K(x) when sel='1' else K(y),** este número foi reduzido à metade, sendo necessárias agora 384 expressões.

Na tabela 1, tem-se uma comparação temporal e espacial das duas metodologias descritas em VHDL e implementadas em FPGA.

Tabela 1 – Otimização obtida na geração de chaves do DES

Desempenho espacial em FPGA	
Metodologia	**CLBs**
Atual	457
Proposta	316

Desempenho temporal em FPGA	
Metodologia	**Tempo de propagação**
Atual	29,542ns
Proposta	23,860ns

Observa-se na tabela 1 que a otimização obtida pela metodologia proposta, tanto no desempenho temporal, expresso pelo tempo de propagação do circuito, como o desempenho espacial ou número de células lógicas Básicas (CLB) utilizadas, foram significativos. No desempenho espacial a otimização foi de aproximadamente 31% e no desempenho temporal de aproximadamente 20%

É importante destacar qual a ferramenta CAE, o software de síntese e o FPGA utilizados. Estes são, respectivamente, Xilinx Foundation Series 3.1, FPGA Express 3.4 e VIRTEX-E.

6. Conclusão

O algoritmo de criptografia DES já foi muito explorado e, mesmo assim, há trechos que sempre podem sofrer otimizações conforme a tecnologia utilizada e o propósito de sua aplicação.

A otimização obtida e descrita neste artigo não é necessariamente a melhor forma de descrição para este algoritmo, mas apenas uma otimização em uma das metodologias mais utilizadas para a implementação do DES em hardware.

Mediante os dados estatísticos obtidos, ficou comprovado a otimização propostas. O DES provavelmente será substituído pelo AES, mas o DES ainda é um dos algoritmos simétricos mais disseminado no mundo, o que justifica o estudo e pesquisa realizado aqui.

7. Trabalhos futuros

Este estudo sobre o DES foi realizado como parte de um trabalho maior de pesquisa que visa a implementação de um criptoprocessador adaptivo com recursos de escalonamento de algoritmos criptográficos. A seguir os trabalhos futuros:

- Estudo do algoritmos AES, RC5, RSA e implementação em C e VHDL
- Estudar o ambiente e protocolo de comunicação entre os criptoprocessadores.

8. Referências

[1] Ordonez E. D. M, Pereira F. D. et al. Projeto, Desempenho e Aplicações de Sistemas Digitais em Circuitos Programáveis (FPGAs), Bless Gráfica e Editora, Marília/SP, 2003.

[2] NIST - National Institute of Standards and Technology, "DATA ENCRYPTION STANDARD (DES) FISP 46-3", USA, 1999.

[3] Routo Terada, Segurança de Dados Criptografia em Redes de Computadores, Ed. Edgard Blücher, 1ª Edição, 2000.

[4] Schneier B. Applied Cryptography: Protocols, Algorithms and Source in C. 2nd Ed. John Wiley and Sons, New York, 1996.

[5] An FPGA Implementation and Performance Evaluation of the Serpent Block Cipher. AJ. Elbirt, C.Paar. In Proc.of FPGAs 2000.

[6] A Current Perspective on Encryption Algorithms. Lawrie Brown. Technical report, School of Computer Science, Australian Defense Force Academy, Canberra, Australia. www.adfa.oz.au/~lpb/papers. UniforumNZ'99, May, 1999. Also, UUG98, Sept., 1998.

[7] Reconfigurable Hardware in Modern Cryptography. Technical Report, ECC 2000, Prof. C. Paar. Cryptography and Information Security Group, Electrical and Computer Engineering Department. Worcester Polytechnic Institute. http://www.ece.wpi.edu/Research/crypt.

[8] Implementations Options for Finite Field Arithmetic for Elliptic Curve Cryptosystems. Technical Report, ECC '99, Prof. C. Paar. Cryptography and Information Security Group, Electrical and Computer Engineering Department. Worcester Polytechnic Institute. http://www.ece.wpi.edu/Research/crypt.

[9]Network and Computer Science. Course Information, 1999. R. L. Rivest. web.mit.edu/6.85//www/announce.html.

[10] How Well Are High-End DSPs Suited for the AES Algorithms ? AES Algorithms on the TMS320C6x DSP. Thomas J. Wollinger, Min Wang, Jorge Guajardo, C. Paar. The Third Advance Encryption Standard (AES3) Candidate Conference, April, 2000, New York, U.S.A

[11] A Rapid Prototyping Enviroment for Processor Design and Simulation to Teach Computer Architecture. Sebatiano Pizzutilo, Filippo Tangorra. Università di Bari, Italy. May, 2001. PDCS-2002.

[12] Serpent : A Proposal for the Advanced Encryption Standard. Ross Anderson, Eli Biham, LarsKnudsen. Cambridge University, England.

[13] AES Proposal: Rijndael Block Cipher. Joan Daemen, Vincent Rijmen. 03/09/99.

[14] Fast Implementations of RSA Cryptography. M Shand, J. Vuillemin,. Digital Equipment Corp (DEC). Paris Research Laboratory, France.

[15] Implementations of the TWOFISH Cipher Using FPGA Devices. Pawel Chodowiec, Kris Gaj. Technical Report,

Electrical and Computer engineering, George Mason University, July, 1999.

[16] MARS: A Candidate Cipher for AES. Carolynn Burwick, Don Coppersmith, Edward D´Avignon. IBM Corporation, Sep., 1999.

[17] Architectural Synthesis from Behavioral Code to Implementation in a XILINX FPGA. Doug Johnson, Business Development Manager, Forntier Design Inc. www.frontierd.com

[18] An Algorithm-Agile Cryptographic Coprocessor Based on FPGAs. Christof Paar, Brebdon Chetwynd, Thomas Connor, Sheng Yung Deng, Steve Marchant. ECE Dep. Worcester Polytechnic Institute, Worcester, U.S.A, The SPIE´s Symposium on Voice and Data Communications, Sep. 1999, Bosto, U.S.A

[19] An Architectural Analysis of Cryptographic Applications for Network Processors. Haiyong Xie, Li Zhou, Laxmi Bhuyan. Department of Computer Science and Engineering, Univ. of California, Riverside. IEEE First Workshop on Network Processors, with HPCA-8, Bosto, U.S.A., Feb., 2002.

[20] CryptoManiac: A Fast Flexible Architecture for Secure Communication. Lisa Wu, Chris Weaver, Todd Austin. Advanced Computer Architecture Laboratory, Univ. of Michigan, U.S.A, In Proc. of ISCA-2001 (Intl. Conf. On Computer Architecture), Goteborg, Sweden.

[21] Architectural Support for Fast Symmetric-Key Cryptography. Jerome Burke, John McDonald, Todd Austin. Advanced Computer Architecture Laboratory, Univ. of Michigan, U.S.A, In Proc. of ASPLOS IX, Cambridge, 2000.

[22] The Simulation and Implementation of Next Generation Encryption Algorithm RIJNDAEL. Dong Bok Yeom, Jong Su Lee, Jong Sou Park, Sang Tae Kim. In Proc. of MIC-2002.