UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA

Otávio de Souza Martins Gomes

Desenvolvimento de Hardware Configurável de Criptografia Simétrica utilizando FPGA e linguagem VHDL

Itajubá, Janeiro de 2011

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA

Otávio de Souza Martins Gomes

Desenvolvimento de Hardware Configurável de Criptografia Simétrica utilizando FPGA e linguagem VHDL

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Área de Concentração: Microeletrônica

Orientador: Prof. Dr. Tales Cleber Pimenta

Co-orientador: Prof. Dr. Robson Luiz Moreno

Janeiro de 2011
Itajubá - MG

Ficha catalográfica elaborada pela Biblioteca Mauá –
Bibliotecária Margareth Ribeiro- CRB_6/1700

B813b

Gomes, Otávio de Souza Martins
Desenvolvimento de hardware configurável de criptografia simétrica utilizando FPGA e linguagem VHDL / Otávio de Souza Martins Gomes. -- Itajubá, (MG) : [s.n.], 2011.
69 p. : il.

Orientador: Prof. Dr. Tales Cleber Pimenta.
Coorientador: Prof. Dr. Robson Luiz Moreno.
Dissertação (Mestrado) – Universidade Federal de Itajubá.

1. Criptografia. 2. AES. 3. FPGA. 4. Hardware. 5. Segurança. 6. VHDL. 7. Microeletrônca. I. Pimenta, Tales Cleber, orient. II. Moreno, Robson Luiz, coorient. III. Universidade Federal de Ita_ jubá. IV. Título.

Dedico este trabalho à minha amada família:
Vandir, Regina, Matheus e Áthila.

"Riquezas e glória vêm de diante de Ti, e Tu dominas sobre tudo, e na Tua mão há força e poder; e na Tua mão está o engrandecer e o dar força a tudo. (...)
Como a sombra são os nossos dias sobre a terra,
e sem Ti não há esperança."
(I Cr 29:11-16)

Agradecimentos

Agradeço, primeiramente, a Deus por me capacitar a concluir este trabalho com êxito. *"Porque o SENHOR dá a sabedoria; da sua boca é que vem o conhecimento e o entendimento."* (Pv 2:6)

À minha família por ser meu suporte e auxílio em todo o tempo.

Agradeço a meus professores e orientadores, Tales Cleber Pimenta e Robson Luiz Moreno, pelo companheirismo, ajuda e esclarecimentos sem os quais a realização deste trabalho não seria possível.

Aos professores Paulo César Crepaldi e Luis Henrique Carvalho que, com valiosas sugestões, me auxiliaram durante a revisão do texto deste trabalho.

Aos colegas do Grupo de Microeletrônica pelo companheirismo, sugestões e contribuições que permitiram o aperfeiçoamento deste.

Ao CNPq, CAPES e FAPEMIG que, através do suporte financeiro, viabilizaram a realização desse trabalho.

Meus mais sinceros agradecimentos.

Resumo

O objetivo deste trabalho é disponibilizar um hardware de criptografia AES rápido e modular, pois com algumas alterações ele pode ser configurado em 128, 192 ou 256 bits de chave e ser aplicado em dispositivos de *smart metering*, criptografia USB, entre outros. A comparação com outros trabalhos foi feita utilizando a arquitetura de 128 bits. Este trabalho apresenta um núcleo do *Advanced Encryption Standard* (AES) desenvolvido em FPGA (*Field Programmable Gate Array*). Para o desenvolvimento foram utilizadas uma placa Spartan-3 FPGA e uma Virtex 5 FPGA. Foi desenvolvido um hardware eficiente (comparado a alguns trabalhos que serão descritos), com arquitetura de chave e palavra, ambos, de 128 bits. A freqüência obtida na Spartan-3 foi de 318 MHz (pelo menos 50% mais rápida que outros hardwares, conforme será mostrado neste trabalho) e obteve-se uma freqüência de 800 MHz na placa Virtex-5. Uma arquitetura de pipelines foi desenvolvida para testar o desempenho dos módulos.

Abstract

The main goal of this work was the implementation of a fast and modular AES algorithm, as it can be easily reconfigured to 128, 196 or 256 bits key, and can find a wide range of applications. Nevertheless, all the reported works used as comparison basis to our work were also implemented using 128 bits key. This article describes the core implementation of an Advanced Encryption Standard - AES in Field Programmable Gate Array - FPGA. The core was implemented in both Xilinx Spartan-3 and Xilinx Virtex-5 FPGAs. The algorithm was implemented for 128 bits word and key. The implementation was very efficient, achieving 318MHz on a Xilinx Spartan-3, representing at 50% faster than other reported works. The implementation can achieve 800MHz on a Xilinx Virtex-5. An pipelined architecture was developed to verify the modules performance.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

3DES	TRIPLE DATA ENCRYPTION STANDARD
AES	ADVANCED ENCRYPTION STANDARD
ASCII	AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE
ASIC	APPLICATION-SPECIFIC INTEGRATED CIRCUIT
CBC	CIPHER BLOCK CHAINING
CPLD	COMPLEX PROGRAMMABLE LOGIC DEVICE
DES	DATA ENCRYPTION STANDARD
FPGA	FIELD PROGRAMMABLE GATE ARRAY
IEEE	INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS
LUT	LOOK-UP TABLE
NIST	NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
RSA	RIVEST-SHAMIR-ADLEMAN
VHDL	VHSIC HARDWARE DESCRIPTION LANGUAGE
VHSIC	VERY HIGH SPEED INTEGRATED CIRCUIT
XOR	EXCLUSIVE OR (OU-EXCLUSIVO)

Capítulo 1

Introdução

A criptografia é importante no âmbito da tecnologia de informação para que se possa garantir a segurança em todo o ambiente computacional que necessita de sigilo em relação às informações. Desde o tempo dos egípcios a criptografia já estava sendo utilizada no sistema de escrita hieroglífica.

Historicamente, a privacidade e segurança na comunicação entre pessoas podiam ser garantidas pela realização de encontros reservados. Para a comunicação à distância, seria necessário enviar mensagens através de intermediários e, para manter a segurança, foram desenvolvidos códigos e cifras para esconder o conteúdo das mensagens daqueles que as interceptassem sem permissão.

A criptografia é a ciência de desenvolver e descobrir tais cifras, principalmente nas áreas diplomáticas e militares dos governos. As telecomunicações aumentaram a rapidez e confiabilidade da comunicação remota. No século 20, o uso da criptografia foi automatizado, para tornar mais rápida e eficaz sua aplicação [1].

Atualmente a criptografia é usada como uma técnica de transformação de dados, segundo um código, ou algoritmo, para que eles se tornem ininteligíveis, a não ser para quem possui a chave do código.

Existem várias técnicas e algoritmos para que a segurança na transferência de dados seja implementada, tanto em hardware, quanto em software. Há, também, diversos órgãos governamentais e instituições que normatizam e controlam os padrões de segurança de dados[2].

Em 1997, o NIST (*National Institute of Standards and Technology*) lançou um concurso para adotar o novo algoritmo de criptografia simétrica, que passaria a se chamar AES (*Advanced Encryption Standard*), para proteger dados confidenciais dos EUA. Havia a necessidade de um novo

algoritmo, pois o algoritmo DES (*Data Encryption Standard*) que estava em vigor não fornecia toda a segurança esperada para o tráfego de informações e possuía uma chave de 56 bits.

Este algoritmo deveria atender a alguns requisitos como: direitos autorais livres; maior rapidez em relação ao 3DES (*Triple DES* - uma otimização do DES); cifrar em blocos de 128 bits com chaves de 128, 192 e 256 bits; possibilidade de implementação em software e hardware [3].

Em 2000, após uma filtragem de alguns candidatos e análises de especialistas na área de criptografia, o vencedor foi o algoritmo Rijndael, que foi criado pelos belgas Vincent Rijmen e Joan Daemen [4]. Este algoritmo foi utilizado neste trabalho.

1.1 - Motivação e Objetivos

A criptografia em hardware é mais segura que aquela desenvolvida em software devido à dificuldade de se quebrar a chave e/ou descobrir como foi realizada a implementação. Os dispositivos reconfiguráveis permitem o desenvolvimento deste hardware de maneira segura e com uma grande flexibilidade [2][3].

Este trabalho foi desenvolvido tendo em vista as diversas aplicações que podem ser beneficiadas com uma interface de criptografia em hardware, além de contribuir com o Grupo de Microeletrônica da Universidade Federal de Itajubá para o aumento do conhecimento na área de segurança na transmissão de dados.

Escolheu-se trabalhar com FPGA (*Field Programmable Gate Array*) principalmente por causa de sua flexibilidade e capacidade de ser reconfigurado. Um dispositivo reconfigurável é muito vantajoso quando se utiliza um algoritmo de criptografia devido às alterações que podem ser feitas com um mínimo de custo e tempo adicionais. O desenvolvimento terá código portável podendo ser utilizado em qualquer família ou fabricante de FPGA's. Foi escolhido o algoritmo AES por ser amplamente utilizado em criptografia simétrica para transferência de informações sigilosas e confidenciais[4].

1.2 - Estrutura do trabalho

No capítulo 2 será feita uma explanação sobre FPGA's e linguagens de descrição de hardware, em especial VHDL, que é a linguagem utilizada neste trabalho. No Capítulo 3 será relatada, brevemente, a história e os tipos de criptografia, e o que é segurança da informação. Para que se

entenda como foi feito o desenvolvimento, no capítulo 4 serão explicadas as funções definidas no algoritmo AES.

A modelagem do trabalho, as fases e os desenvolvimentos realizados e seus resultados serão mostrados no capítulo 5. O capítulo 6 apresenta os diferenciais e vantagens deste trabalho com relação àqueles desenvolvidos anteriormente e o capítulo 7 trará as propostas para trabalhos futuros. As conclusões deste trabalho estão no capítulo 8.

Capítulo 2

Desenvolvimento de Hardware

2.1 - FPGA

FPGA é um circuito lógico-programável, isto é, uma classe de circuitos integrados com propósito geral. É um dispositivo semicondutor que é largamente utilizado para o processamento de informações digitais. A Computação Reconfigurável é uma solução intermediária na resolução de problemas complexos, possibilitando combinar a velocidade do hardware com a flexibilidade do software. Uma arquitetura reconfigurável possui várias metas, entre elas o aumento de desempenho.

Dentre os vários segmentos em relação às arquiteturas reconfiguráveis, destacam-se os Processadores Reconfiguráveis. Estes processadores combinam as funções de um microprocessador com uma lógica reconfigurável e podem ser adaptados depois do processo de desenvolvimento [6].

Foi criado pela Xilinx Inc., e teve o seu lançamento no ano de 1985 como um dispositivo que poderia ser programado de acordo com as aplicações do usuário (programador).

A flexibilidade de programação, associada a potentes ferramentas de desenvolvimento e modelagem, possibilita ao usuário acesso a projetos de circuitos integrados complexos sem os altos custos de engenharia associados aos ASICs (*Aplication-specific Integrated Circuit*).

O FPGA apresenta na ordem de milhares de unidades lógicas idênticas. Neste aspecto estas unidades lógicas podem ser vistas como componentes padrões que podem ser configurados independentemente e interconectados a partir de uma matriz de trilhas condutoras e chaves programáveis.

A estrutura básica de uma FPGA é formada pelo vetor de unidades lógicas e pela matriz de interconexão que podem ser programados pelo usuário.

Para configurar o FPGA é utilizado um arquivo binário que contém as informações necessárias para especificar a função de cada unidade lógica e fechar as chaves da matriz de interconexão necessárias. Para gerar o arquivo binário podem ser utilizadas ferramentas de software seguindo um determinado fluxo de projeto [7].

Os recursos adicionais como flip-flops, multiplexadores, lógica de transporte (*carry*) dedicado, e portas lógicas, podem ser utilizados em conjunto com os LUT (*Look-Up Tables*) para implementar diversas funções booleanas, multiplicadores e somadores, contadores, conversores serial-paralelo e paralelo-serial, e memórias com praticamente qualquer comprimento de palavra, fornecendo assim flexibilidade ao desenvolvimento [6].

2.2 - VHDL

VHDL é uma linguagem para descrição de hardware. Ela é utilizada para descrever o comportamento eletrônico de um circuito ou sistema.

VHDL é a abreviação de ***VHSIC H**ardware **D**escription **L**anguage*, sendo que VHSIC quer dizer *Very High Speed Integrated Circuits* (Circuitos integrados de altíssima velocidade) e foi criada pelo DoD (Departamento de Defesa Norte Americano) nos anos 80. A partir do VHSIC o VHDL se desenvolveu.

A primeira versão do VHDL foi chamada de VHDL'87. Foi a primeira linguagem de descrição de hardware padronizada pelo IEEE (*Institute of Electrical and Electronics Engineers*) através do padrão IEEE 1076. Depois surgiu o padrão IEEE 1164, que possui algumas modificações[8].

As aplicações básicas de VHDL são em FPGA's, CPLD's (*Complex Programmable Logic Devices*) e ASIC's (*Aplication Specified Integrated Circuits*). O VHDL é uma linguagem que trabalha de forma paralela, ao contrário de muitas linguagens de programação que são seqüenciais[9].

Há algumas linguagens amplamente utilizadas hoje, como: VHDL, Verilog, SystemVerilog e SystemC. Cada uma destas linguagens tem seus pontos fortes e fracos, além do que a utilização de qualquer uma delas no desenvolvimento dependerá do conhecimento por parte dos desenvolvedores e das ferramentas disponíveis. Um sistema pode ser modelado de maneira eficiente em qualquer uma destas linguagens [9].

Em VHDL, assim como nas outras linguagens, os projetos são organizados em hierarquias [9], isto é, interpretando componentes básicos como blocos que virão a formar um bloco maior, que no

caso é o projeto em si. Isso torna o código reutilizável, permitindo fazer alterações e reutilizar as "peças" já desenvolvidas.

Foi escolhida a linguagem VHDL devido ao conhecimento e experiência já existentes acerca dela e à sua ampla utilização.

De modo geral, VHDL é uma linguagem adequada para se trabalhar com grandes projetos, que possuem um alto nível de abstração[8].

Capítulo 3

Segurança da Informação: Criptografia

3.1 - Definições

A segurança da informação digital é uma questão de preocupação no mundo atual. Por isso foram desenvolvidas várias técnicas para dificultar a aquisição de dados confidenciais por aqueles que na deveriam ter acesso a eles. Uma dessas técnicas é a criptografia digital, que consiste em transformar informações em códigos para evitar o acesso de pessoas não autorizadas ao conteúdo da mensagem. Algumas destas técnicas são utilizadas desde a antigüidade [1].

Com o surgimento dos computadores vários tipos de algoritmos criptográficos foram desenvolvidos. Os algoritmos considerados seguros necessitam de grande processamento e algumas vezes o tamanho da mensagem criptografada aumenta significativamente [2].

Vários modelos de criptografia existentes se tornaram ultrapassados, devido à velocidade com que sua segurança pode ser violada. Para aumentar a segurança é necessário aumentar significativamente o tamanho da informação, o que acaba se tornando inviável para a transmissão em alguns tipos de rede. Novos modelos de criptografia são desenvolvidos visando aumentar a segurança e velocidade na transmissão de dados [3].

3.2 - Histórico

A criptografia é tão antiga quanto a própria escrita, visto que já estava presente no sistema de escrita hieroglífica dos egípcios. Os romanos utilizavam códigos secretos para comunicar planos de batalha. Com as guerras mundiais e a invenção do computador, a criptografia cresceu incorporando complexos algoritmos matemáticos.

A criptologia faz parte da história humana porque sempre houve fórmulas secretas e informações confidenciais que não deveriam cair no domínio público ou na mão de inimigos. O primeiro exemplo documentado da escrita cifrada relaciona-se aproximadamente ao ano de 1900 A.C., quando o escriba de *Khnumhotep II* teve a idéia de substituir algumas palavras ou trechos de texto. Caso o documento fosse roubado, o ladrão não encontraria o caminho que o levaria ao tesouro e morreria de fome perdido nas catacumbas da pirâmide [1].

Em 50 a.C, Júlio César usou sua famosa cifra de substituição para cifrar (criptografar) comunicações governamentais. Para compor seu texto cifrado, César alterou letras desviando-as em três posições; A se tornava D, B se tornava E etc. Às vezes, César reforçava seu método de criptografar mensagens substituindo letras latinas por gregas. O código de César é o único da Antigüidade que é usado até hoje. Atualmente qualquer cifra baseada na substituição cíclica do alfabeto denomina-se código de César. Apesar da sua simplicidade (ou exatamente por causa dela), essa cifra foi utilizada pelos oficiais sulistas na Guerra de Secessão americana e pelo exército russo em 1915.

Em 1901, iniciou-se a era da comunicação sem fio. Apesar da vantagem de uma comunicação de longa distância sem o uso de fios ou cabos, o sistema é aberto e aumenta o desafio da criptologia. Em 1921, Edward Hugh Hebern fundou a Hebern Electric Code, uma empresa produtora de máquinas de cifragem eletromecânicas baseadas em rotores que giram a cada caractere cifrado[2].

Entre 1933 e 1945, a máquina Enigma que havia sido criada por Arthur Scherbius foi aperfeiçoada até se transformar na ferramenta criptográfica mais importante da Alemanha nazista.

O sistema foi quebrado pelo matemático polonês Marian Rejewski que se baseou apenas em textos cifrados interceptados e numa lista de chaves obtidas por um espião.

Os computadores são a expressão maior da era digital, marcando presença em praticamente todas as atividades humanas. Da mesma forma com que revolucionaram a informação, também influenciaram a criptologia; por um lado, ampliaram seus horizontes, por outro, tornaram a criptologia indispensável [10].

3.3 - Tipos de Criptografia

Os algoritmos de criptografia podem ser classificados por meio do tratamento dado às informações que serão processadas; assim, têm-se os algoritmos de bloco e os algoritmos de fluxo.

A cifra de blocos opera sobre blocos de dados. O texto antes de ser cifrado é dividido em blocos que variam normalmente de 8 a 16 bytes que serão cifrados ou decifrados. Quando o texto não completa o número de bytes de um bloco, este é preenchido com dados conhecidos (geralmente valor zero "0") até completar o número de bytes do bloco, cujo tamanho já é predefinido pelo algoritmo que está sendo usado.

Um problema na cifra de bloco é que se o mesmo bloco de texto simples aparecer mais de uma vez, a cifra gerada será a mesma, facilitando o ataque ao texto cifrado. Para resolver esse problema são utilizados os modos de realimentação[10].

O modo mais comum de realimentação é a cifragem de blocos por encadeamento CBC (*Cipher Block Chaining*). Neste modo é realizada uma operação de XOR do bloco atual de texto simples com o bloco anterior de texto cifrado. Para o primeiro bloco, não há bloco anterior de texto cifrado; assim, faz-se uma XOR com um vetor de inicialização. Este modo não adiciona nenhuma segurança extra. Apenas evita o problema citado da cifra de bloco[1].

Portanto, os algoritmos de blocos processam os dados como um conjunto de bits, sendo os mais rápidos e seguros para a comunicação digital. Ainda, como vantagem, que os blocos podem ser codificados fora de ordem, o que é bom para acesso aleatório, além de ser resistente a erros, uma vez que um bloco não depende de outro. Entretanto, como desvantagem, se a mensagem possuir padrões repetitivos nos blocos, o texto cifrado também o apresentará, o que facilitará o serviço do criptoanalista.

Além dos algoritmos de bloco, existem os algoritmos de fluxo. Os algoritmos de fluxo criptografam a mensagem bit a bit, em um fluxo contínuo, sem esperar que se tenha um bloco completo de bits. É também chamado de criptografia em fluxo de dados, onde a criptografia se dá mediante uma operação XOR entre o bit de dados e o bit gerado pela chave.

Com relação ao número de chaves, pode-se classificar a criptografia como simétrica ou assimétrica. Na criptografia de chave simétrica, os processos de cifragem e decifragem são feitos com uma única chave, ou seja, tanto o remetente quanto o destinatário usam a mesma chave[10].

Em algoritmos simétricos, como, por exemplo, o DES, ocorre o chamado “problema de distribuição de chaves”. A chave tem de ser enviada para todos os usuários autorizados antes que as mensagens possam ser trocadas. Essa ação resulta num atraso de tempo e possibilita que a chave chegue a pessoas não autorizadas[3].

A criptografia assimétrica contorna o problema da distribuição de chaves mediante o uso de chaves públicas. A criptografia de chaves públicas foi inventada em 1976 por Whitfield Diffie e Martin Hellman, a fim de resolver o problema da distribuição de chaves. Neste novo sistema, cada pessoa tem um par de chaves denominado chave pública e chave privada. A chave pública é divulgada, enquanto a chave privada é mantida em segredo. Para mandar uma mensagem privada, o transmissor cifra a mensagem usando a chave pública do destinatário pretendido, que deverá usar a sua respectiva chave privada para conseguir recuperar a mensagem original[10].

Atualmente, um dos mecanismos de segurança mais utilizados é a assinatura digital, que precisa dos conceitos de criptografia assimétrica. A assinatura digital é uma mensagem que só uma pessoa poderia produzir, mas que todos possam verificar. Normalmente autenticação se refere ao uso de assinaturas digitais: a assinatura é um conjunto que não pode ser forjado de dados assegurando o nome do autor que funciona como uma assinatura de documentos, ou seja, que determinada pessoa concordou com o que estava escrito. Tal procedimento também evita que a pessoa que assinou a mensagem possa alegar que a mensagem foi forjada. Um exemplo de criptossistema de chave pública é o RSA (*Rivest-Shamir-Adleman*), cuja maior desvantagem é a sua capacidade de canal limitada, ou seja, o número de bits de mensagem que pode transmitir por segundo.

Neste trabalho é utilizado o algoritmo AES, que trabalha com cifragem de blocos e criptografia simétrica. A Figura 3.1 ilustra o funcionamento da criptografia simétrica. Na criptografia simétrica, a mesma chave utilizada para cifrar a mensagem, também é utilizada para decifrá-la.

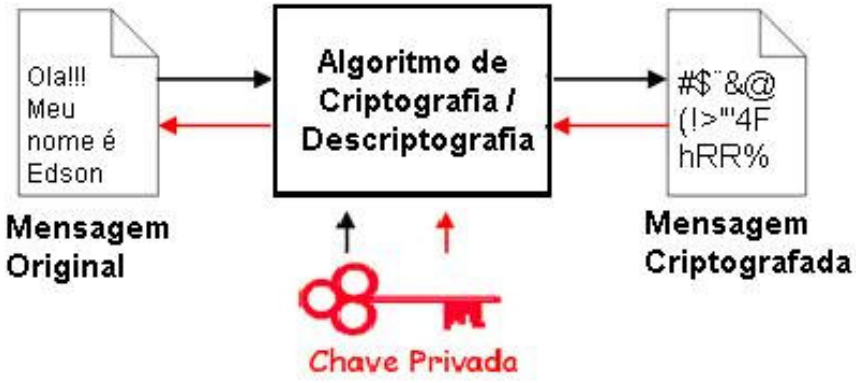

Figura 3.1: Exemplo de Criptografia Simétrica

Capítulo 4

Algoritmo AES

Toda explicação sobre o funcionamento da cada uma das funções, das rodadas de cifragem ou decifragem, e da expansão das chaves tem como fonte os documentos oficiais do *NIST* e dos criadores do algoritmo Rijndael ([4], [5], [12] e [13]).

4.1 - AES (*Advanced Encryption Standard*)

O atual padrão de criptografia dos EUA se originou de um concurso lançado em 1997 pelo NIST. Havia a necessidade de escolher um algoritmo mais seguro e eficiente para substituir o DES, que apresentou fragilidades.

O novo algoritmo deveria atender a certos pré-requisitos como: ser divulgado publicamente e não possuir patentes; cifrar em blocos de 128 bits usando chaves de 128, 192 e 256 bits; ser implementado tanto em software quanto em hardware; ter maior rapidez em relação ao 3DES, uma variação recursiva do antigo padrão DES.

Em 1998, na Primeira Conferência dos Candidatos AES, apresentaram-se 15 candidatos, como pode ser visto na Tabela 4.1.

Um ano depois, na Segunda Conferência, foram indicados 5 destes como finalistas: MARS, RC6, Rijndael, Serpent e Twofish.

Em 2000, foi conhecido o vencedor: Rijndael. O algoritmo, criado pelos belgas Vincent Rijmen e Joan Daemen, foi escolhido com base em qualidades como segurança, bom desempenho em software e hardware, entre outros atributos[4].

Tabela 4.1: Os 15 candidatos aceitos para a 1ª rodada de avaliação do AES

ALGORTIMO	RESPONSÁVEL	TIPO DE SUBMISSÃO
CAST-256	Entrust (CA)	Empresa
Crypton	Future Systems (KR)	Empresa
DEAL	Outerbridge, Knudsen (USA-DK)	Pesquisador
DFC	ENS-CNRS (FR)	Pesquisador
E2	NTT (JP)	Empresa
Frog	TecApro (CR)	Empresa
HPC	Schroeppel (USA)	Pesquisador
LOKI97	Brown et al. (AU)	Pesquisador
Magenta	Deutsche Telekom (DE)	Empresa
Mars	IBM (USA)	Empresa
RC6	RSA (USA)	Empresa
Rijndael	Daemen and Rijmen (BE)	Pesquisador
SAFER+	Cylink (USA)	Empresa
Serpent	Anderson, Biham, Knudsen (UK-IL-DK)	Pesquisador
Twofish	Counterpane (USA)	Empresa

4.2 - Algoritmo AES-Rijndael

A Tabela 4.2 mostra as possíveis configurações para que se implemente o algoritmo AES. Para a criptografia 128 bits é necessário um bloco de palavra de quatro linhas e quatro colunas com 8 bits em cada célula. Para que a criptografia esteja completa será necessário que se processem 10 rodadas. Os valores utilizados para os cálculos e matrizes estão em hexadecimal.

Para o AES 192 e 256 mantém-se a palavra de 128 bits, alterando o tamanho da chave para 192 e 256 bits, respectivamente, e o número de rodadas para 12 e 14 [5].

As seções seguintes irão detalhar as rodadas e o processamento da palavra e da chave.

Tabela 4.2: Configurações do AES

AES	Palavra	Chave	Rodadas
128	4 x 4	4 x 4	10
192	4 x 4	4 x 6	12
256	4 x 4	4 x 8	14

4.3 - Matemática utilizada no AES

O algoritmo de criptografia AES, trabalha sobre o Campo de Galois GF, ou seja, todas as operações matemáticas utilizadas são realizadas sobre este campo, e usa-se o polinômio irredutível $m(x) = x^8 + x^4 + x^3 + x + 1$.

O motivo de se trabalhar sobre campo de Galois, ou campos finitos, em criptografia, é a garantia da existência de uma operação inversa para cada etapa, que é fundamental no processo de decifragem. No caso de se utilizar $GF(2^n)$, com n pertencente aos números naturais, temos ainda outro aspecto interessante: a soma coincide com a operação Ou-Exclusivo, operação muito rápida computacionalmente[11].

Para a criptografia, em particular, o uso de $GF(2^8)$ é bastante adequado, visto que esse campo tem $2^8 = 256$ elementos, o mesmo número de caracteres da tabela ASCII (*American Standard Code for Information Interchange*) estendida. Assim, é possível cifrar e decifrar qualquer mensagem. Todas as etapas, exceto a ShiftRows, que é um rotacionamento circular dos bytes do estado, utilizam como base o campo $GF(2^8)$ [5] .

SubBytes - Consiste em uma substituição dos bytes do estado por outros contidos em uma caixa de substituição (S-box). Essa caixa de substituição é resultado de operações polinomiais realizadas no campo de Galois.

MixColumns - Pode ser representada como uma multiplicação de matrizes, onde os elementos do estado são considerados polinômios sobre $GF(2^8)$. Multiplica-se uma matriz fixa C pela matriz S que representa o estado, e obtemos a matriz S′; onde a operação realizada é o produto matricial em $GF(2^8)$, ou seja, a multiplicação é feita módulo m(x) e a soma corresponde ao XOR.

AddRoundKey - É um XOR byte a byte entre o estado e a chave de rodada. Isto quer dizer que se $s_{x,y}$ é um byte do estado S e $k_{x,y}$ um byte da chave, o byte $s'_{x,y}$ do novo estado S′ será igual a $s_{x,y} \oplus k_{x,y}$. A transformação inversa à AddRoundKey também consiste em um XOR entre o estado e a chave de rodada, ou seja, AddRoundKey é sua própria inversa, visto que $(s_{x,y} \oplus k_{x,y}) \oplus k_{x,y} = s_{x,y}$ [11].

4.4 - Expansão da Chave

Para a realização da expansão da chave utiliza-se uma matriz de constantes que é dada pela matemática polinomial de Galois. A matriz de 4 linhas e 10 colunas está representada na Figura 4.1 para 10 rodadas de expansão [4].

01	**02**	**04**	**08**	**10**	**20**	**40**	**80**	**1B**	**36**
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**

Figura 4.1: Constante das Rodadas

A expansão das chaves ocorre da seguinte maneira:

a) É recebido um bloco (palavra) que representa a chave de entrada e é composta de 4 linhas e 4 colunas.

b) As primeiras operações serão feitas sobre a quarta coluna, da seguinte forma:

b.1) É feita uma rotação na quarta coluna de modo que a célula que está acima de todas as outras fique abaixo delas.

b.2) A partir do resultado da rotação é feita a substituição utilizando a tabela S-Box (como mostrado na Tabela 4.3).

b.3) Após a substituição é feita uma operação de Ou-Exclusivo com a primeira coluna da matriz de constantes e, em seguida, é feita uma operação de Ou-Exclusivo com a primeira coluna do bloco inicial.

b.4) Tem-se assim a primeira coluna do próximo bloco de chave.

c) A segunda coluna do bloco de chave é o resultado de uma operação Ou-Exclusivo entre a primeira coluna do novo bloco (calculada em **b.4**) e a segunda coluna do bloco inicial.

d) A terceira coluna do bloco de chave é o resultado de uma operação Ou-Exclusivo entre a segunda coluna do novo bloco (calculada em **c**) e a terceira coluna do bloco inicial.

e) A quarta coluna do bloco de chave é o resultado de uma operação Ou-Exclusivo entre a terceira coluna do novo bloco (calculada em **d**) e a quarta coluna do bloco inicial.

f) Tem-se, assim, um segundo bloco formado por 4 linhas e 4 colunas com 8 bits em cada uma das células e que será a chave da primeira rodada.

g) A partir do bloco da primeira rodada (calculado em **f**) inicia-se todo este procedimento utilizando a quarta coluna. As operações se repetem até que todas as chaves estejam calculadas.

4.5 - Cifrando a Mensagem

A cifragem da palavra segue o fluxo mostrado na Figura 4.2: é inserida uma mensagem (texto) que passa por várias operações (que são descritas nas próximas seções) até que fique cifrada. Este processo de cifragem faz com que uma mensagem legível seja criptografada e não consiga ser lida por ninguém além daquele que possui a chave para decifrá-la.

O texto será processado N vezes em algumas das operações, sendo que N é o número de rodadas, que define também a segurança do algoritmo (10, 12 ou 14 rodadas).

As chaves para cada rodada já foram calculadas na operação de Expansão da Chave (seção 4.4) e são utilizadas nas operações AddRoundKey [12].

Cada uma destas operações será descrita com mais detalhes nas seções a seguir.

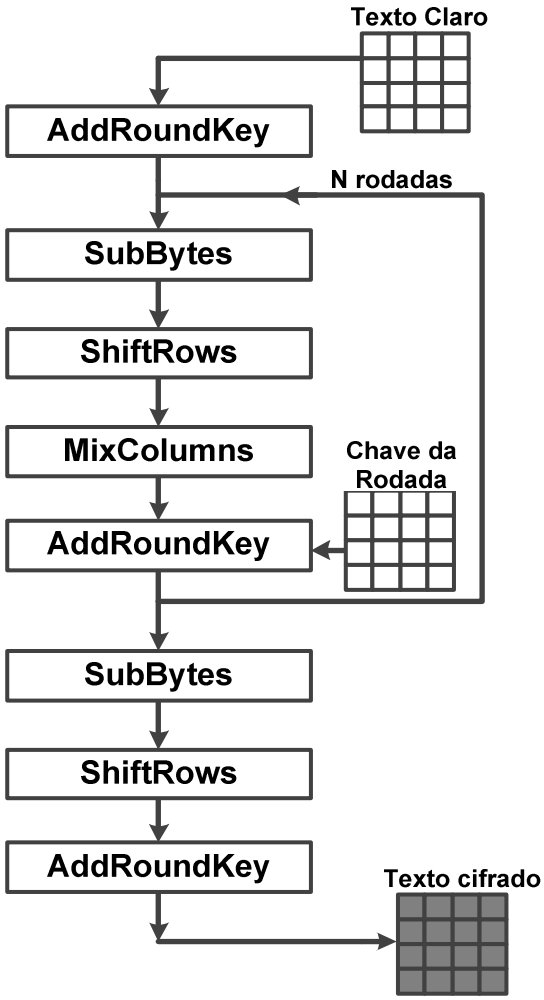

Figura 4.2: Processo de Cifragem

4.5.1 - SubBytes

A operação SubBytes utiliza a matriz S-Box calculada pela matemática polinomial de Galois para realizar as substituições. A matriz está representada na Tabela 4.3.

Tabela 4.3: S-Box

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

A Figura 4.3 demonstra de maneira sucinta como é feita a substituição dos valores. Tem-se o bloco de palavra representado por 4.3a e faz-se a operação SubBytes no primeiro termo: 5F. É selecionada a linha 5 e a coluna F da matriz S-Box, encontrando-se na intercessão desta linha e coluna o valor CF. É feita a substituição de 5F por CF para realizar a operação SubBytes, conforme indica a Figura 4.3b. Essa operação de busca e substituição é realizada para cada uma das células do bloco (palavra) que está sendo calculado [5].

5F	7D	23	6A
27	B9	CE	D2
E6	2D	A9	F0
E8	A0	F3	50

(a)

CF			

(b)

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 4.3: Execução da Função SubBytes

4.5.2 - ShiftRows

A função ShiftRows realiza um deslocamento das células que estão à esquerda. O número de células deslocadas obedece ao número da linha que sofrerá a alteração. A Figura 4.4 ilustra o bloco antes (4.4a) e depois (4.4b) da operação ShiftRows: a primeira linha (linha número 0) não sofre alteração, a segunda linha (linha número 1) sofre apenas um deslocamento, a terceira linha (linha número 2) sofre dois deslocamentos e a quarta linha (linha número 3) sofre três deslocamentos [13].

5F	7D	23	6A
27	B9	CE	D2
E6	2D	A9	F0
E8	A0	F3	50

(a)

5F	7D	23	6A
B9	CE	D2	27
A9	F0	E6	2D
50	E8	A0	F3

(b)

Figura 4.4: Execução da Função ShiftRows

4.5.3 - MixColmuns

Para a realização da função MixColumns, utiliza-se outra matriz calculada a partir da matemática de Galois, que se encontra na Figura 4.5b e é formada por 4 linhas e 4 colunas.

Será realizada uma operação com cada uma das colunas por todas as linhas da matriz constante. A figura 4.5 será empregada para exemplificar a operação, onde o bloco 4.5a é a palavra e o bloco 4.5b é a matriz utilizada nesta função, conforme especificação do algoritmo[4].

5F	7D	23	6A
27	B9	CE	D2
E6	2D	A9	F0
E8	A0	F3	50

(a)

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

(b)

Figura 4.5: Palavra e Constante utilizada para Função MixColumns

O resultado da primeira célula da primeira coluna será: $(5F \cdot 02) \oplus (27 \cdot 03) \oplus (E6 \cdot 01) \oplus (E8 \cdot 01)$

O resultado da segunda célula da primeira coluna será: $(5F \cdot 01) \oplus (27 \cdot 02) \oplus (E6 \cdot 03) \oplus (E8 \cdot 01)$

O resultado da terceira célula da primeira coluna será: $(5F \cdot 01) \oplus (27 \cdot 01) \oplus (E6 \cdot 02) \oplus (E8 \cdot 03)$

O resultado da quarta célula da primeira coluna será: $(5F \cdot 03) \oplus (27 \cdot 01) \oplus (E6 \cdot 01) \oplus (E8 \cdot 02)$

Tem-se, assim, a primeira coluna do bloco resultante desta operação. Para as outras colunas a operação é idêntica [4].

4.5.4 - AddRoundKey

A função AddRoundKey nada mais é do que uma operação Ou-Exclusivo entre um bloco palavra e um bloco chave (uma das chaves da expansão), resultando em um novo bloco de mesma dimensão, conforme Figura 4.6. A matriz S representa a palavra resultante da operação anterior e a matriz W representa a chave correspondente à rodada que está sendo calculada [5].

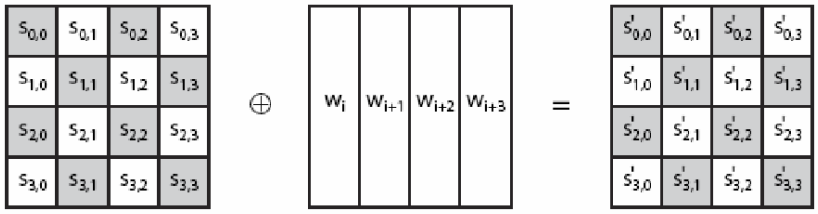

Figura 4.6: Função AddRoundKey

4.6 - Decifrando a Mensagem

A decifragem da palavra segue o fluxo mostrado na Figura 4.7, onde é inserido um texto cifrado que passa por várias transformações. Em algumas das transformações ou operações o texto passa N vezes, sendo que N é o número de rodadas, que define também a segurança do algoritmo (10, 12 ou 14 rodadas), assim como ocorre na cifragem.

As chaves para cada rodada já estão calculadas e são utilizadas nas operações AddRoundKey [4]. Cada uma destas operações será descrita com mais detalhes nas seções a seguir.

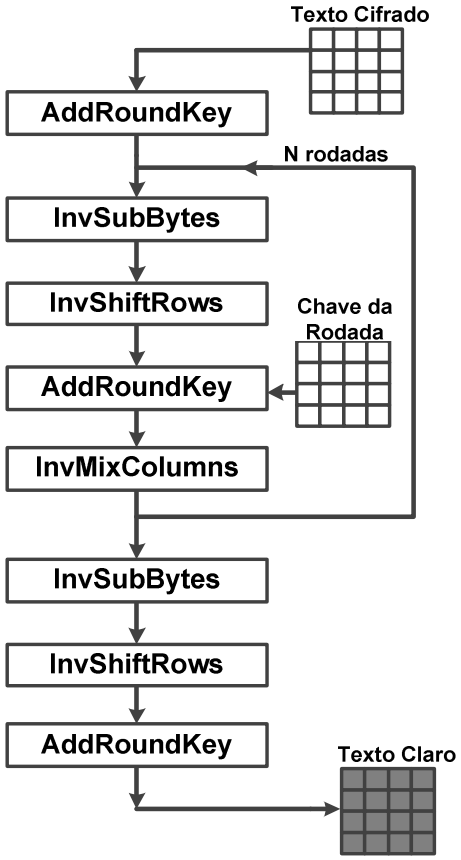

Figura 4.7: Processo de Decifragem

4.6.1 - InvSubBytes

A função InvSubBytes realiza uma operação similar à função SubBytes. É importante ressaltar que a matriz utilizada para as substituições é diferente da função SubByes, é chamada InvS-Box e apresentada na Tabela 4.4 [12].

Tabela 4.4: InvS-Box

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

4.6.2 - InvShiftRows

A função InvShiftRows é similar à função ShiftRows mas realiza um deslocamento das células que estão à direita. A Figura 4.8 mostra um esquemático de como é realizada esta operação. O deslocamento das células tem como objetivo fazer com que o bloco fique embaralhado. As funções de decifragem visam reverter as mudanças realizadas pela cifragem para que mensagem volte estar legível ao usuário[5].

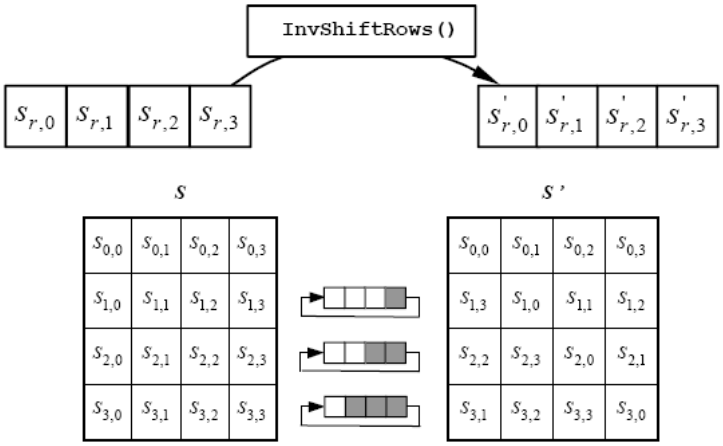

Figura 4.8: Função InvShiftRows

4.6.3 - InvAddRoundKey

As funções InvAddRoundKey e AddRoundKey são idênticas, isto é, uma operação Ou-Exclusivo entre um bloco palavra e um bloco chave (uma das chaves da expansão), resultando em um novo bloco de mesma dimensão.

4.6.4 - InvMixColumns

A função InvMixColumns realiza uma operação similar à função MixColumns. A matriz utilizada para realizar a operação é diferente daquela utilizada na operação MixColumns e está representada na Figura 4.9.

0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

Figura 4.9: Constante para a Função InvMixColumns

4.7 - Diferença entre o Rijndael e o AES

O Rijndael foi desenvolvido para suportar tamanhos de chave e bloco variando entre 128, 160, 192, 224 e 256 bits. O AES suporta somente blocos de 128 e chaves de 128, 192 e 256.

Os dois possuem o mesmo funcionamento, utilizando as mesmas funções e substituições, sendo que o Rijndael pode utilizar uma variação maior de tamanhos de chave e bloco do que o AES[5].

Capítulo 5

Modelagem

O desenvolvimento deste trabalho foi dividido em três fases, sendo que cada uma delas foi utilizada como ferramenta para a fase seguinte. Em primeiro lugar foi estudado o algoritmo AES, suas características e modos de implementação. Nesta fase foi possível obter todo o conhecimento necessário sobre o funcionamento do AES e suas possíveis aplicações.

A segunda fase foi o desenvolvimento de um software que aplicasse o estudo realizado e o verificasse funcionamento real do algoritmo AES. A terceira fase foi a implementação de todos estes conceitos em linguagem de descrição de hardware VHDL, utilizando FPGA.

A primeira fase foi explicada no embasamento teórico visto anteriormente neste trabalho, as fases dois e três serão descritas a seguir. Nos resultados apresentados utiliza-se criptografia AES 128 bits, que consiste em 10 rodadas para cifragem e/ou decifragem. Os valores utilizados para os cálculos e das matrizes estão em hexadecimal.

5.1 – Implementação em C++

Utilizou-se a linguagem C++ e a ferramenta *Microsoft Visual Studio C++ 2008 - Express Edition* para que se desenvolvesse o software e se verificasse o funcionamento de todos os conceitos do algoritmo. Como requisitos, o software deveria fornecer o resultado de cada um dos cálculos intermediários, isto é, após cada uma das funções (SubBytes, ShiftRows, etc.) ser executada, o software deveria disponibilizar o resultado para que o usuário tivesse acesso a todo o processo de cifragem, de decifragem e da expansão da chave. Na próxima sessão serão demonstrados os resultados do desenvolvimento do software em C++. Foi utilizada a mesma chave e palavra iniciais para os resultados que serão demonstrados a seguir. O número escolhido é um valor aleatório e é: *0x*5f27e6e87db92da023cea9f36ad2f050.

5.2 - Operações implementadas em C++

A Figura 5.1 apresenta o código responsável pela expansão da chave. O usuário insere o valor da chave (que fica armazenado no vetor *KEY*) e as chaves de todas as rodadas são calculadas e exibidas ao usuário até a rodada final. A variável *RCON* é a matriz de constantes da expansão das chaves (Figura 4.1).

```
// Realiza a expansão da Chave que será utilizada para cifrar/decifrar a palavra
void KeyExpansion(){
	int i,j;
	unsigned char temp[4],k;

	// A rodada inicial consiste na própria palavra
	for(i=0;i<Nk;i++){
		RoundKey[i*4]=Key[i*4];
		RoundKey[i*4+1]=Key[i*4+1];
		RoundKey[i*4+2]=Key[i*4+2];
		RoundKey[i*4+3]=Key[i*4+3];
	}

(...) Uma parte do código foi retirada

	// As rodadas seguintes utilizam as chaves calculadas anteriormente
	while (i < (Nb * (Nr+1))){
		for(j=0;j<4;j++){
			temp[j]=RoundKey[(i-1) * 4 + j];
		}
		if (i % Nk == 0){
			// Função Rotate
				k = temp[0];
				temp[0] = temp[1];
				temp[1] = temp[2];
				temp[2] = temp[3];
				temp[3] = k;

			// Função SubByte
				temp[0]=getSBoxValue(temp[0]);
				temp[1]=getSBoxValue(temp[1]);
				temp[2]=getSBoxValue(temp[2]);
				temp[3]=getSBoxValue(temp[3]);

			//Cálculo envolvendo a primeira coluna e a matriz Round Word Constant Array
			temp[0] =  temp[0] ^ Rcon[i/Nk];
		}

		//Cálculo utilizando XOR e combinação entre as colunas
		RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
		RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
		RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
		RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];

		printf("%02x ",RoundKey[i*4+0]);
		printf("%02x ",RoundKey[i*4+1]);
		printf("%02x ",RoundKey[i*4+2]);
		printf("%02x ",RoundKey[i*4+3]);
		printf(" | ");

		i++;
	}
}
```

Figura 5.1: Código da Expansão da Chave em C++

A Figura 5.2 lista o resultado da expansão da chave inicial. A chave fornecida pelo usuário é *START Key* e como resultado da expansão têm-se as chaves de todas as rodadas. Deve-se lembrar que está sendo implementado o AES-128.

```
                    COLUMN_1        COLUMN_2        COLUMN_3        COLUMN_4
                   ------------------------------------------------------------
START Key          : 5f 27 e6 e8   | 7d b9 2d a0   | 23 ce a9 f3   | 6a d2 f0 50   |
                   ------------------------------------------------------------
Key ROUND 01       : eb ab b5 ea   | 96 12 98 4a   | b5 dc 31 b9   | df 0e c1 e9   |
Key ROUND 02       : 42 d3 ab 74   | d4 c1 33 3e   | 61 1d 02 87   | be 13 c3 6e   |
Key ROUND 03       : 3b fd 34 da   | ef 3c 07 e4   | 8e 21 05 63   | 30 32 c6 0d   |
Key ROUND 04       : 10 49 e3 de   | ff 75 e4 3a   | 71 54 e1 59   | 41 66 27 54   |
Key ROUND 05       : 33 85 c3 5d   | cc f0 27 67   | bd a4 c6 3e   | fc c2 e1 6a   |
Key ROUND 06       : 36 7d c1 ed   | fa 8d e6 8a   | 47 29 20 b4   | bb eb c1 de   |
Key ROUND 07       : 9f 05 dc 07   | 65 88 3a 8d   | 22 a1 1a 39   | 99 4a db e7   |
Key ROUND 08       : c9 bc 48 e9   | ac 34 72 64   | 8e 95 68 5d   | 17 df b3 ba   |
Key ROUND 09       : 4c d1 bc 19   | e0 e5 ce 7d   | 6e 70 a6 20   | 79 af 15 9a   |
Key ROUND 10       : 03 88 04 af   | e3 6d ca d2   | 8d 1d 6c f2   | f4 b2 79 68   |
```

Figura 5.2: Resultado da Expansão da Chave em C++

Além do desenvolvimento da expansão da chave, há também o código de cifragem e o de decifragem. A Figura 5.3 mostra uma das fases da cifragem, que é a função ShiftRows, que foi explicada no capítulo anterior.

```
//Função ShiftRows - ENCRYPT
//Rotaciona as linhas para a esquerda conforme o número da linha
//sendo que a primeira linha (núm.=0) não sofre rotação
void ShiftRows(){
	unsigned char temp;

	//Rotaciona a 2ª linha uma coluna à esquerda
	temp=state[1][0];
	state[1][0]=state[1][1];
	state[1][1]=state[1][2];
	state[1][2]=state[1][3];
	state[1][3]=temp;

	//Rotaciona a 3ª linha duas colunas à esquerda
	temp=state[2][0];
	state[2][0]=state[2][2];
	state[2][2]=temp;

	temp=state[2][1];
	state[2][1]=state[2][3];
	state[2][3]=temp;

	//Rotaciona a 4ª linha três colunas à esquerda
	temp=state[3][0];
	state[3][0]=state[3][3];
	state[3][3]=state[3][2];
	state[3][2]=state[3][1];
	state[3][1]=temp;
}
```

Figura 5.3: Código da Função ShiftRows em C++

A Figura 5.4 lista a função de cifragem onde são utilizadas as funções de todos os outros cálculos realizados durante a encriptação da palavra. A chamada de cada uma destas funções pode ser

vista neste código (AddRoundKey, SubBytes, ShiftRows, MixColumns). O código está todo comentado para auxiliar futuras consultas ou modificações.

```
//Função Cipher - ENCRYPT
	//ROUND 0 - Rodada inicial
	//Realiza a função XOR entra a palavra inicial e a chave fornecida
	//Calcula a palavra para o 1º Round
	AddRoundKey(0);
	printf("\nText ROUND 00  : ");

	printf("\nText ROUND 01  : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			printf("%02x ",state[j][i]);
		}
	}

	// Número de rounds: Nr = 10 (128 bits)
	// Os primeiros 9 (Nr-1) rounds são idênticos
	for(round=1;round<Nr;round++)
	{
		SubBytes();
		printf("  SubBytes     : ");

		ShiftRows();
		printf("\n  ShiftRows    : ");

		MixColumns();
		printf("\n  MixColumns   : ");

		AddRoundKey(round);
	}

	// O último round (10°) não realiza o cálculo MixColumns
	SubBytes();
	ShiftRows();
	AddRoundKey(round);
```

Figura 5.4: Código da Função de Cifragem em C++

A Figura 5.5 mostra o resultado da cifragem (ou encriptação) da palavra. Como dito anteriormente, a palavra inicial e chave inicial são iguais a *0x*5f27e6e87db92da023cea9f36ad2f050, por este motivo o resultado da rodada inicial (round 00) é igual a *0x*00, pois é realizada a operação AddRoundKey (que é um Ou-Exclusivo) entre os valores de entrada.

Ainda pela Figura 5.5 pode-se consultar o resultado intermediário de cada uma das operações realizadas a partir da palavra inicial, até sua cifragem completa.

	COLUMN_1	COLUMN_2	COLUMN_3	COLUMN_4
START Text	**5f 27 e6 e8**	**7d b9 2d a0**	**23 ce a9 f3**	**6a d2 f0 50**
Text ROUND 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Text ROUND 01	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
SubBytes	63 63 63 63	63 63 63 63	63 63 63 63	63 63 63 63
ShiftRows	63 63 63 63	63 63 63 63	63 63 63 63	63 63 63 63
MixColumns	63 63 63 63	63 63 63 63	63 63 63 63	63 63 63 63
AddRoundKey	88 c8 d6 89	f5 71 fb 29	d6 bf 52 da	bc 6d a2 8a
Text ROUND 02	88 c8 d6 89	f5 71 fb 29	d6 bf 52 da	bc 6d a2 8a
SubBytes	c4 e8 f6 a7	e6 a3 0f a5	f6 08 00 57	65 3c 3a 7e
ShiftRows	c4 a3 00 7e	e6 08 3a a7	f6 3c f6 a5	65 e8 0f 57
MixColumns	13 e7 e5 08	52 1f 68 56	e0 2a c9 9a	b1 e8 6a e6
AddRoundKey	51 34 4e 7c	86 de 5b 68	81 37 cb 1d	0f fb a9 88
Text ROUND 03	51 34 4e 7c	86 de 5b 68	81 37 cb 1d	0f fb a9 88
SubBytes	d1 18 2f 10	44 1d 39 45	0c 9a 1f a4	76 0f d3 c4
ShiftRows	d1 1d 1f c4	44 9a d3 10	0c 0f 2f 45	76 18 39 a4
MixColumns	45 0e a5 f9	fe 15 53 a5	63 26 92 be	59 a9 eb e8
AddRoundKey	7e f3 91 23	11 29 54 41	ed 07 97 dd	69 9b 2d e5
Text ROUND 04	7e f3 91 23	11 29 54 41	ed 07 97 dd	69 9b 2d e5
SubBytes	f3 0d 81 26	82 a5 20 83	55 c5 88 c1	f9 14 d8 d9
ShiftRows	f3 a5 88 d9	82 c5 d8 26	55 14 81 83	f9 0d 20 c1
MixColumns	58 f8 2d 8a	b5 46 86 cc	94 66 c6 77	1f 42 ec a4
AddRoundKey	48 b1 ce 54	4a 33 62 f6	e5 32 27 2e	5e 24 cb f0
Text ROUND 05	48 b1 ce 54	4a 33 62 f6	e5 32 27 2e	5e 24 cb f0
SubBytes	52 c8 8b 20	d6 c3 aa 42	d9 23 cc 31	58 36 1f 8c
ShiftRows	52 c3 cc 8c	d6 23 1f 20	d9 36 8b 42	58 c8 aa 31
MixColumns	ba 0c 9d fa	ed 91 ab 1d	3a 71 24 49	68 07 8c e8
AddRoundKey	89 89 5e a7	21 61 8c 7a	87 d5 e2 77	94 c5 6d 82
Text ROUND 06	89 89 5e a7	21 61 8c 7a	87 d5 e2 77	94 c5 6d 82
SubBytes	a7 a7 58 5c	fd ef 64 da	17 03 98 f5	22 a6 3c 13
ShiftRows	a7 ef 98 13	fd 03 3c 5c	17 a6 58 da	22 a7 64 f5
MixColumns	f4 c2 56 a3	84 e3 62 9b	5d 72 74 68	27 2e 49 54
AddRoundKey	c2 bf 97 4e	7e 6e 84 11	1a 5b 54 dc	9c c5 88 8a
Text ROUND 07	c2 bf 97 4e	7e 6e 84 11	1a 5b 54 dc	9c c5 88 8a
SubBytes	25 08 88 2f	f3 9f 5f 82	a2 39 20 86	de a6 c4 7e
ShiftRows	25 9f 20 7e	f3 39 c4 2f	a2 a6 88 82	de 08 5f 86
MixColumns	ae 1e 78 2c	5d f9 28 ad	a4 f4 92 cc	66 a9 f9 39
AddRoundKey	31 1b a4 2b	38 71 12 20	86 55 88 f5	ff e3 22 de
Text ROUND 08	31 1b a4 2b	38 71 12 20	86 55 88 f5	ff e3 22 de
SubBytes	c7 af 49 f1	07 a3 c9 b7	44 fc c4 e6	16 11 93 1d
ShiftRows	c7 a3 c4 1d	07 fc 93 f1	44 11 49 b7	16 af c9 e6
MixColumns	b2 d0 d0 0f	73 bb ce 9f	45 0a 05 e1	e9 f5 01 8b
AddRoundKey	7b 6c 98 e6	df 8f bc fb	cb 9f 6d bc	fe 2a b2 31
Text ROUND 09	7b 6c 98 e6	df 8f bc fb	cb 9f 6d bc	fe 2a b2 31
SubBytes	21 50 46 8e	9e 73 65 0f	1f db 3c 65	bb e5 37 c7
ShiftRows	21 73 3c c7	9e db 37 8e	1f e5 46 0f	bb 50 65 65
MixColumns	2c 44 78 b9	e8 e4 a2 52	43 0b 67 9c	9d d1 8e 29
AddRoundKey	60 95 c4 a0	08 01 6c 2f	2d 7b c1 bc	e4 7e 9b b3
Text ROUND 10	60 95 c4 a0	08 01 6c 2f	2d 7b c1 bc	e4 7e 9b b3
SubBytes	d0 2a 1c e0	30 7c 50 15	d8 21 78 65	69 f3 14 6d
ShiftRows	d0 7c 78 6d	30 21 14 e0	d8 f3 1c 15	69 2a 50 65
AddRoundKey	d3 f4 7c c2	d3 4c de 32	55 ee 70 e7	9d 98 29 0d
Encrypted Text	**d3 f4 7c c2**	**d3 4c de 32**	**55 ee 70 e7**	**9d 98 29 0d**

Figura 5.5: Resultado da Cifragem em C++

A Figura 5.6 apresenta o resultado da decifragem (ou decriptação) da palavra. A palavra inicial é *0x*d3f47cc2d34cde3255ee70e79d98290d e chave inicial é *0x*5f27e6e87db92da023cea9f36ad2f050. É possível observar os valores de cada uma das operações realizadas a partir da palavra inicial que está criptografada, até sua decifragem completa.

	COLUMN_1	COLUMN_2	COLUMN_3	COLUMN_4
Encrypted Text	**d3 f4 7c c2**	**d3 4c de 32**	**55 ee 70 e7**	**9d 98 29 0d**
Text ROUND 00	d0 7c 78 6d	30 21 14 e0	d8 f3 1c 15	69 2a 50 65
Text ROUND 10	d0 7c 78 6d	30 21 14 e0	d8 f3 1c 15	69 2a 50 65
InvShiftRows	d0 2a 1c e0	30 7c 50 15	d8 21 78 65	69 f3 14 6d
InvSubBytes	60 95 c4 a0	08 01 6c 2f	2d 7b c1 bc	e4 7e 9b b3
AddRoundKey	2c 44 78 b9	e8 e4 a2 52	43 0b 67 9c	9d d1 8e 29
InvMixColumns	21 73 3c c7	9e db 37 8e	1f e5 46 0f	bb 50 65 65
Text ROUND 09	21 73 3c c7	9e db 37 8e	1f e5 46 0f	bb 50 65 65
InvShiftRows	21 50 46 8e	9e 73 65 0f	1f db 3c 65	bb e5 37 c7
InvSubBytes	7b 6c 98 e6	df 8f bc fb	cb 9f 6d bc	fe 2a b2 31
AddRoundKey	b2 d0 d0 0f	73 bb ce 9f	45 0a 05 e1	e9 f5 01 8b
InvMixColumns	c7 a3 c4 1d	07 fc 93 f1	44 11 49 b7	16 af c9 e6
Text ROUND 08	c7 a3 c4 1d	07 fc 93 f1	44 11 49 b7	16 af c9 e6
InvShiftRows	c7 af 49 f1	07 a3 c9 b7	44 fc c4 e6	16 11 93 1d
InvSubBytes	31 1b a4 2b	38 71 12 20	86 55 88 f5	ff e3 22 de
AddRoundKey	ae 1e 78 2c	5d f9 28 ad	a4 f4 92 cc	66 a9 f9 39
InvMixColumns	25 9f 20 7e	f3 39 c4 2f	a2 a6 88 82	de 08 5f 86
Text ROUND 07	25 9f 20 7e	f3 39 c4 2f	a2 a6 88 82	de 08 5f 86
InvShiftRows	25 08 88 2f	f3 9f 5f 82	a2 39 20 86	de a6 c4 7e
InvSubBytes	c2 bf 97 4e	7e 6e 84 11	1a 5b 54 dc	9c c5 88 8a
AddRoundKey	f4 c2 56 a3	84 e3 62 9b	5d 72 74 68	27 2e 49 54
InvMixColumns	a7 ef 98 13	fd 03 3c 5c	17 a6 58 da	22 a7 64 f5
Text ROUND 06	a7 ef 98 13	fd 03 3c 5c	17 a6 58 da	22 a7 64 f5
InvShiftRows	a7 a7 58 5c	fd ef 64 da	17 03 98 f5	22 a6 3c 13
InvSubBytes	89 89 5e a7	21 61 8c 7a	87 d5 e2 77	94 c5 6d 82
AddRoundKey	ba 0c 9d fa	ed 91 ab 1d	3a 71 24 49	68 07 8c e8
InvMixColumns	52 c3 cc 8c	d6 23 1f 20	d9 36 8b 42	58 c8 aa 31
Text ROUND 05	52 c3 cc 8c	d6 23 1f 20	d9 36 8b 42	58 c8 aa 31
InvShiftRows	52 c8 8b 20	d6 c3 aa 42	d9 23 cc 31	58 36 1f 8c
InvSubBytes	48 b1 ce 54	4a 33 62 f6	e5 32 27 2e	5e 24 cb f0
AddRoundKey	58 f8 2d 8a	b5 46 86 cc	94 66 c6 77	1f 42 ec a4
InvMixColumns	f3 a5 88 d9	82 c5 d8 26	55 14 81 83	f9 0d 20 c1
Text ROUND 04	f3 a5 88 d9	82 c5 d8 26	55 14 81 83	f9 0d 20 c1
InvShiftRows	f3 0d 81 26	82 a5 20 83	55 c5 88 c1	f9 14 d8 d9
InvSubBytes	7e f3 91 23	11 29 54 41	ed 07 97 dd	69 9b 2d e5
AddRoundKey	45 0e a5 f9	fe 15 53 a5	63 26 92 be	59 a9 eb e8
InvMixColumns	d1 1d 1f c4	44 9a d3 10	0c 0f 2f 45	76 18 39 a4
Text ROUND 03	d1 1d 1f c4	44 9a d3 10	0c 0f 2f 45	76 18 39 a4
InvShiftRows	d1 18 2f 10	44 1d 39 45	0c 9a 1f a4	76 0f d3 c4
InvSubBytes	51 34 4e 7c	86 de 5b 68	81 37 cb 1d	0f fb a9 88
AddRoundKey	13 e7 e5 08	52 1f 68 56	e0 2a c9 9a	b1 e8 6a e6
InvMixColumns	c4 a3 00 7e	e6 08 3a a7	f6 3c f6 a5	65 e8 0f 57
Text ROUND 02	c4 a3 00 7e	e6 08 3a a7	f6 3c f6 a5	65 e8 0f 57
InvShiftRows	c4 e8 f6 a7	e6 a3 0f a5	f6 08 00 57	65 3c 3a 7e
InvSubBytes	88 c8 d6 89	f5 71 fb 29	d6 bf 52 da	bc 6d a2 8a
AddRoundKey	63 63 63 63	63 63 63 63	63 63 63 63	63 63 63 63
InvMixColumns	63 63 63 63	63 63 63 63	63 63 63 63	63 63 63 63
Text ROUND 01	63 63 63 63	63 63 63 63	63 63 63 63	63 63 63 63
InvShiftRows	63 63 63 63	63 63 63 63	63 63 63 63	63 63 63 63
InvSubBytes	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
AddRoundKey	5f 27 e6 e8	7d b9 2d a0	23 ce a9 f3	6a d2 f0 50
Decrypted Text	**5f 27 e6 e8**	**7d b9 2d a0**	**23 ce a9 f3**	**6a d2 f0 50**

Figura 5.6: Resultado da Decifragem em C++

5.3 - Desenvolvimento de Hardware

Utilizou-se a linguagem VHDL e a ferramenta *Xilinx ISE 10.1.03* para que se desenvolvesse o hardware em FPGA. Como requisitos o hardware deveria fornecer o resultado de cada um dos cálculos intermediários, além de ser modular e permitir que várias configurações pudessem ser montadas utilizando-se cifragem, decifragem e expansão da chave. Na próxima sessão serão mostrados os resultados do desenvolvimento do hardware em FPGA e VHDL. Foi utilizada a mesma chave e palavra iniciais para os resultados, seu valor é: *0x*5f27e6e87db92da023cea9f36ad2f050.

5.4 - Descrição do Hardware desenvolvido em FPGA com VHDL

A Figura 5.7 mostra o bloco de expansão da chave. Quando o sinal *CARREGAR CHAVE* está em nível alto a *CHAVE INICIAL* é carregada para a posição zero do vetor de chaves. O bloco então recebe esta chave e a primeira constante da rodada, calculando assim a chave da primeira rodada. A partir daí o bloco calcula cada uma das chaves utilizando a chave anterior e a constante da rodada (conforme Fig. 4.1).

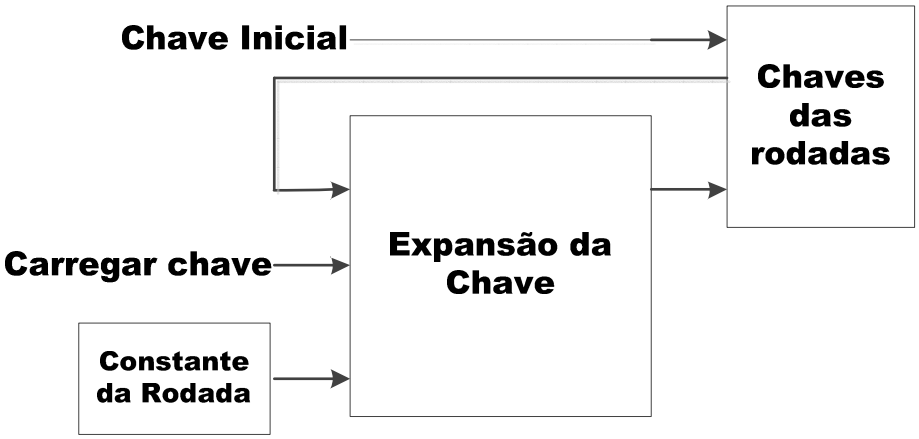

Figura 5.7: Blocos para o funcionamento da Expansão da Chave

A Figura 5.8 mostra o bloco de cifragem ou encriptação, que realiza o processamento da palavra conforme indicação do sinal *SELETOR*. O bloco recebe a chave da rodada do vetor de chaves, que já se encontra disponível e guarda a palavra intermediária em um buffer de saída, que pode ser utilizado como entrada para novo cálculo ou pode ser enviado para a saída do usuário.

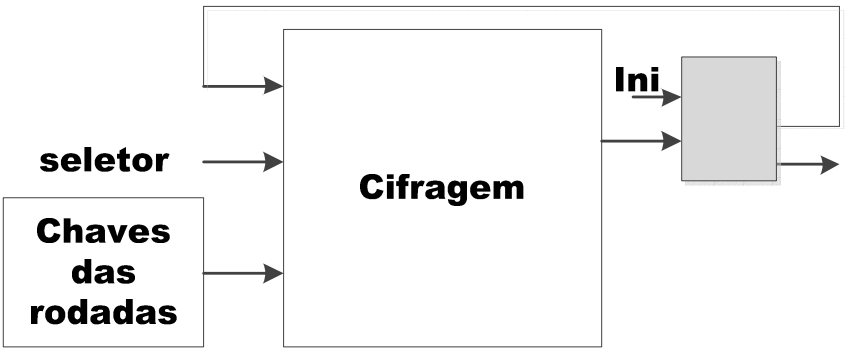

Figura 5.8: Blocos para o funcionamento da Cifragem

Pode-se verificar o funcionamento do sinal SELETOR a partir das Figuras 5.9 e 5.10, que apresentam os blocos internos e o código em VHDL, respectivamente. Na rodada inicial (round 00) a palavra para a primeira rodada é calculada através de um Ou-Exclusivo entre a chave inicial e a palavra inicial, isto é, da função AddRoundKey; neste caso o sinal seletor é igual a *0x*00. Na última rodada (neste caso - AES 128bits - round 10) a palavra final é calculada sem a utilização da função MixColumns; neste caso o sinal seletor é igual a *0x*01. Em todas as rodadas intermediárias (round 01 a round 09) são utilizadas todas as funções, então o sinal do seletor é igual a *0x*10 (poderia ser *0x*11 também).

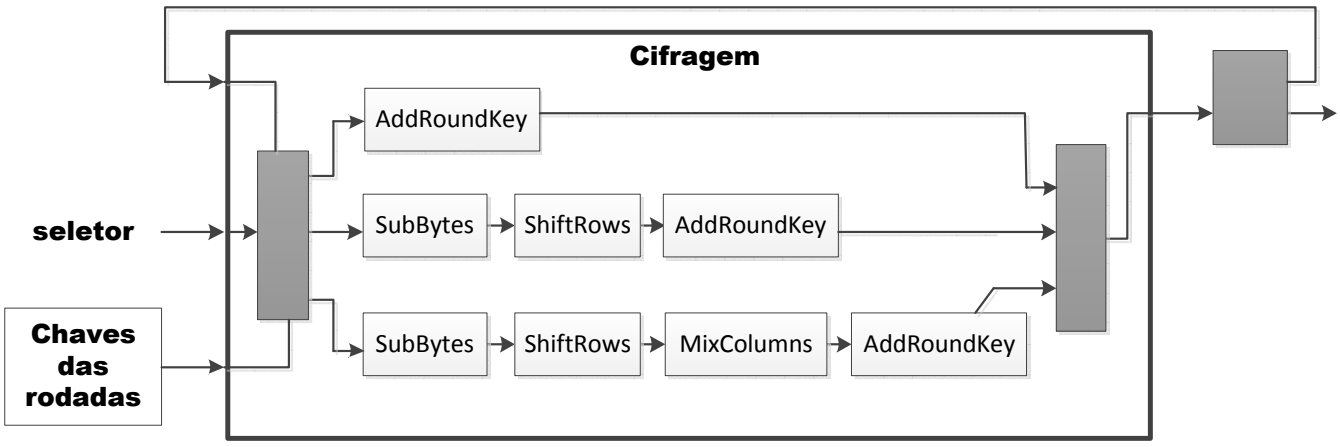

Figura 5.9: Função SELETOR em blocos

```
enc: process(clk, reset)
    BEGIN
        IF (reset='1') THEN
            key <= (OTHERS => '0');
            mux <= (OTHERS => '0');
        ELSIF (clk'event AND clk='1') THEN
            IF (in_mux_sel = "00") THEN
                mux <= in_data;
            ELSIF (in_mux_sel = "01") THEN
                mux <= shiftrow;
            ELSE
                mux <= mixcolumn;
            END IF;
            key <= in_key;
        END IF;
    END PROCESS;

    out_data <= mux XOR key;
```

Figura 5.10: Código em VHDL da função SELETOR

A Figura 5.11 mostra o bloco de decifragem ou decriptação, que realiza o processamento da palavra conforme indicação do sinal seletor. O bloco recebe a chave da rodada do vetor de chaves, que já se encontra disponível e guarda a palavra intermediária em um buffer de saída, que pode ser utilizado como entrada para novo cálculo ou pode ser enviado para a saída do usuário. O funcionamento é semelhante ao processo de cifragem.

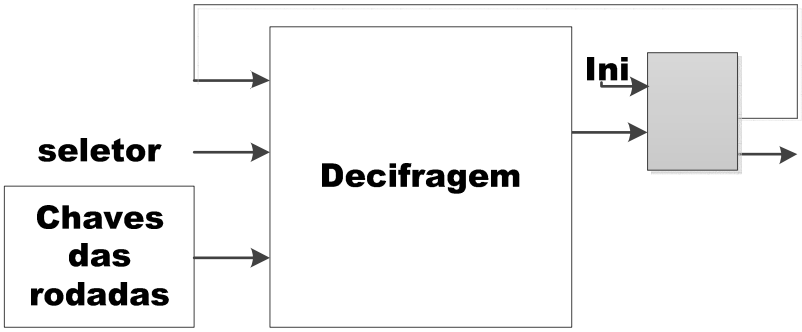

Figura 5.11: Blocos para funcionamento da Decifragem

Pode-se verificar o funcionamento na decifragem do sinal seletor a partir das Figuras 5.12 e 5.13, que apresentam os blocos internos e o código em VHDL, respectivamente. Na rodada inicial (round 00) a palavra para a primeira rodada é calculada através de um Ou-Exclusivo entre a chave final e a palavra inicial, isto é, da função AddRoundKey; neste caso o sinal seletor é igual a *0x*00. Na última rodada (neste caso, AES 128bits, round 10) a palavra final é calculada sem a utilização da função MixColumns; neste caso o sinal seletor é igual a *0x*01. Em todas as rodadas intermediárias (round 01 a round 09) são utilizadas todas as funções, então o sinal do seletor é igual a *0x*10 (poderia ser *0x*11 também).

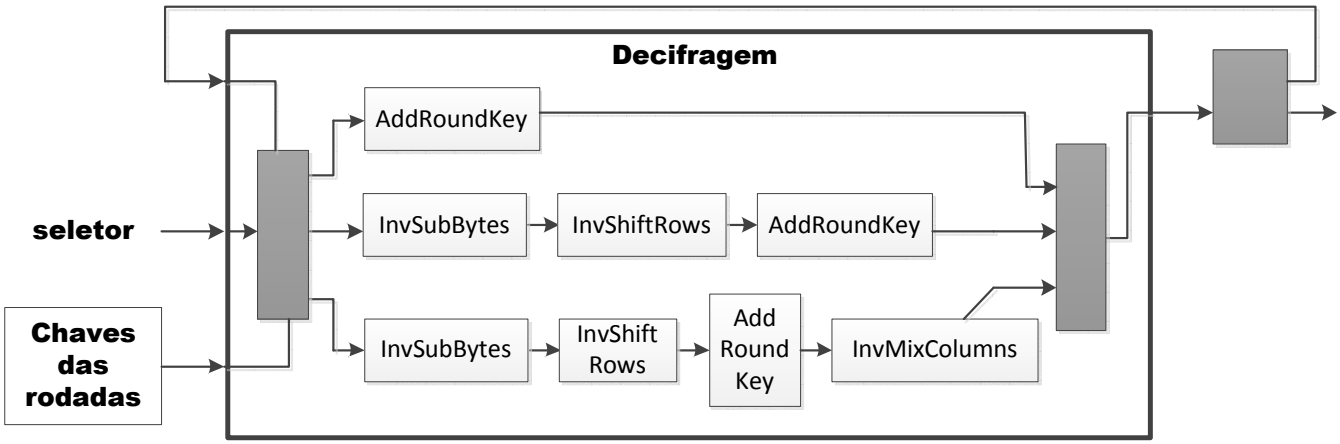

Figura 5.12: Função InvSELETOR em blocos

```
mux <= bytesub XOR in_key;

dec: PROCESS (clk, reset)
BEGIN
        IF (reset='1') THEN
                out_data <= (OTHERS => '0');

        ELSIF (clk'event AND clk='1') THEN
                IF (in_mux_sel = "00") THEN
                        out_data <= in_data xor in_key;
                ELSIF (in_mux_sel = "01") THEN
                        out_data <= mux;
                ELSE
                        out_data <= mixcolumn;
                END IF;
        END IF;
END PROCESS;
```

Figura 5.13: Código em VHDL da função InvSELETOR

A estrutura da montagem dos três blocos para o funcionamento do AES-128 pode ser vista na Figura 5.14. O bloco de expansão da chave processa a chave inicial e calcula suas expansões, guardando-as em um vetor que pode ser acessado tanto pela área de cifragem quanto de decifragem, conforme controle do sinal in_aes_mode no bloco A. Os processos de cifragem e decifragem tem acesso a esse vetor, mas somente um dos dois blocos trabalha de cada vez, isto é, se o processo de cifragem está ocorrendo, a decifragem está desativada, e vice-versa. O sinal in_aes_mode controla qual dos blocos de criptografia colocará a palavra na saída do hardware através do bloco C.

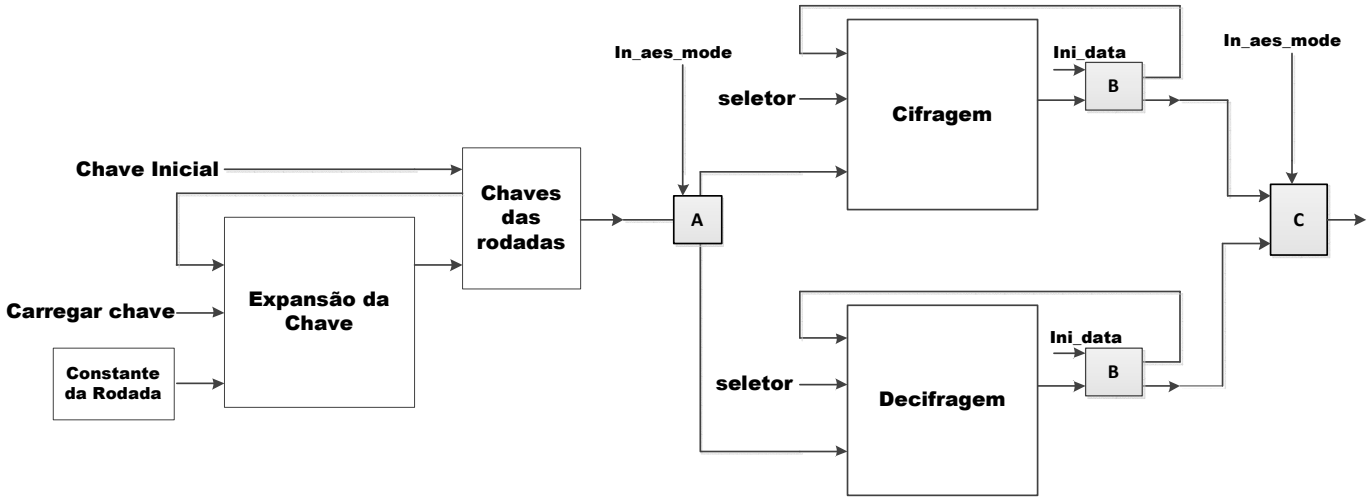

Figura 5.14: Blocos de Expansão da Chave, Cifragem e Decifragem

Pela Figura 5.15 pode-se ver o bloco RTL do hardware implementado na FPGA Virtex-5. O sinal `in_aes_mode` controla se a cifragem (nível baixo) ou decifragem (nível alto) estará sendo utilizada. Os sinais `in_ini_key` e `in_ini_data` recebem a palavra de entrada (texto claro ou cifrada) e a chave inicial. O sinal `out_data` entrega a palavra já calculada (cifrada ou decifrada).

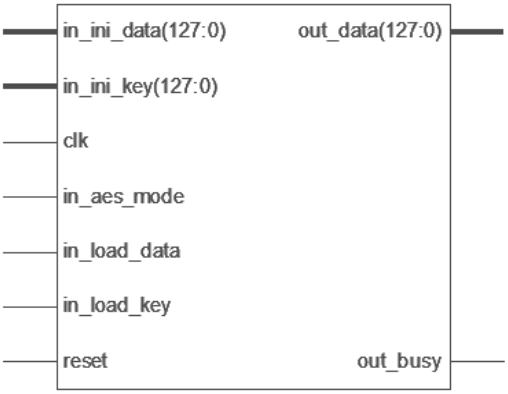

Figura 5.15: Blocos RTL do AES-128

Pode-se verificar o resultado do testbench dos blocos S-Box e InvS-Box através das Figuras 5.16 e 5.17, respectivamente. Os valores apresentados podem ser comparados com as tabelas S-Box (ver Tabela 4.3) e InvS-Box (ver Tabela 4.4).

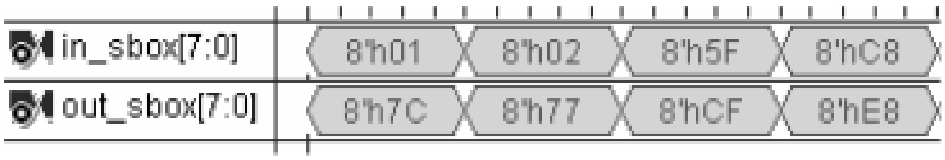

Figura 5.16: Resultado do Testbench S-Box desenvolvido em VHDL

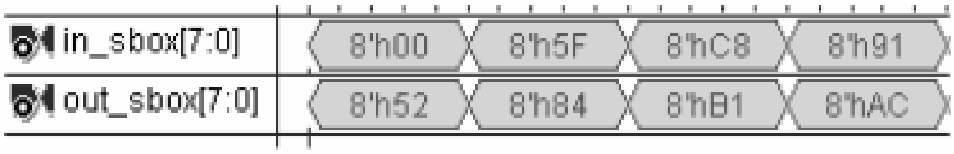

Figura 5.17: Resultado do Testbench InvS-Box desenvolvido em VHDL

Pode-se verificar o resultado do testbench dos blocos ShiftRows e InvShiftRows através das Figuras 5.18 e 5.19, respectivamente. Na Figura 5.20(a) tem-se o mesmo valor de entrada tanto da função ShiftRows, quanto da função InvShiftRows demonstrada nas Figuras 5.18 e 5.19. Na Figura 5.20(b) tem-se o resultado da função ShiftRows e na Figura 5.20(c) tem-se o resultado da função InvShiftRows.

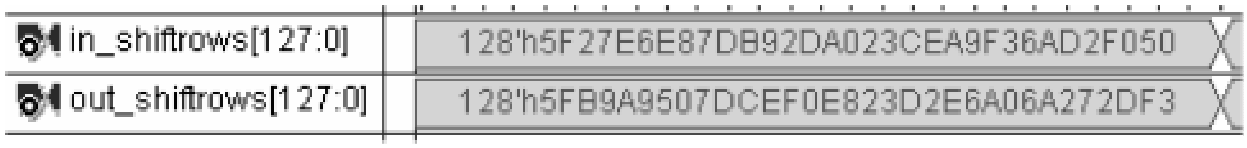

Figura 5.18: Resultado do Testbench ShiftRows desenvolvido em VHDL

Figura 5.19: Resultado do Testbench InvShiftRows desenvolvido em VHDL

5F	7D	23	6A
27	B9	CE	D2
E6	2D	A9	F0
E8	A0	F3	50

(a)

5F	7D	23	6A
B9	CE	D2	27
A9	F0	E6	2D
50	E8	A0	F3

(b)

5F	7D	23	6A
D2	27	B9	CE
A9	F0	E6	2D
A0	F3	50	E8

(c)

Figura 5.20: Aplicação das funções ShiftRows e InvShiftRows

Nas Figuras 5.21 e 5.22, respectivamente, tem-se os resultados dos testbenchs das funções MixColumns e InvMixColumns.

Figura 5.21: Resultado do Testbench MixColumns desenvolvido em VHDL

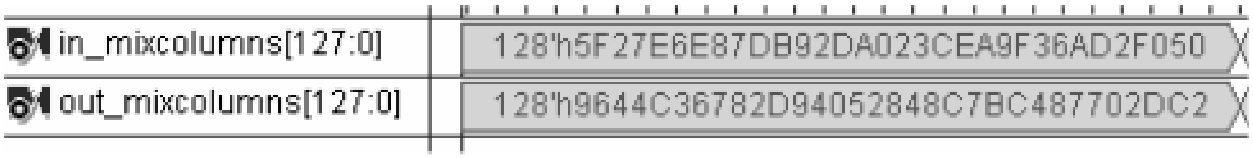

Figura 5.22: Resultado do Testbench InvMixColumns desenvolvido em VHDL

O processo de cifragem se diferencia do processo de decifragem em alguns detalhes como pode ser visto nos capítulos anteriores e um deles é a ordem de utilização das chaves expandidas. Enquanto a cifragem utiliza a chave 01 para a primeira rodada, a chave 02 para a segunda, e assim sucessivamente; a decifragem utiliza a chave 10 para a primeira rodada, a chave 09 para a segunda, e assim sucessivamente. Devido a este fato o tempo inicial gasto para decifragem depende da completa expansão das chaves, enquanto que para a cifragem, basta que eu esteja cifrando a palavra X enquanto faz-se a expansão da chave X+1. Após a expansão de todas as chaves, o tempo de cifragem e decifragem são idênticos.

A Figura 5.23 demonstra o que acabou de ser descrito, para que se cifre uma palavra, basta que a expansão da chave esteja um passo à frente da cifragem. Na imagem vê-se representado na primeira linha os ciclos (rodadas) de expansão da chave e na segunda linha os ciclos (ou rodadas) de cifragem de uma palavra utilizando o AES-128 (10 rodadas).

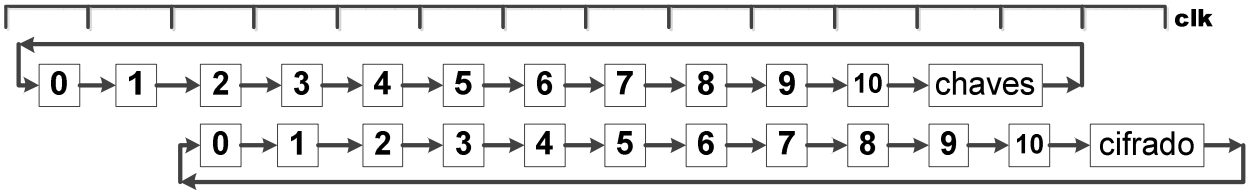

Figura 5.23: Período de Carga para Cifragem

A Figura 5.24 demonstra a decifragem de uma palavra. Para que se decifre uma palavra é preciso que todas as chaves estejam expandidas. Na imagem podem ser vistos, representado na primeira linha, os ciclos (rodadas) de expansão da chave e na segunda linha os ciclos (ou rodadas) de decifragem de uma palavra utilizando o AES-128 (10 rodadas). Só é possível iniciar a decifragem após a conclusão da expansão da chave.

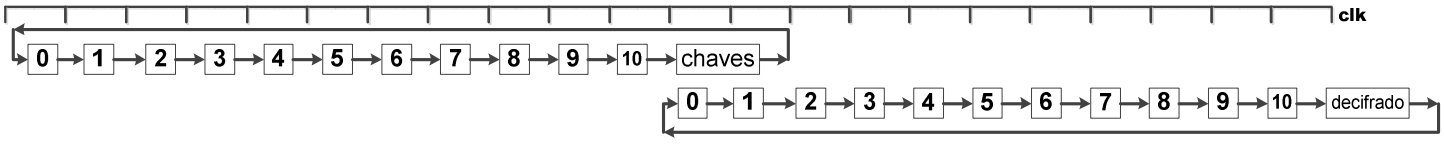

Figura 5.24: Período de Carga para Decifragem

Na cifragem, o tempo inicial necessário para a entrega da palavra cifrada é de aproximadamente 20 ciclos de clock. Esse tempo inclui o processo de expansão da chave e cifragem da palavra. A Figura 5.25 mostra o período em que o barramento de saída fica desabilitado durante a carga inicial de cifragem.

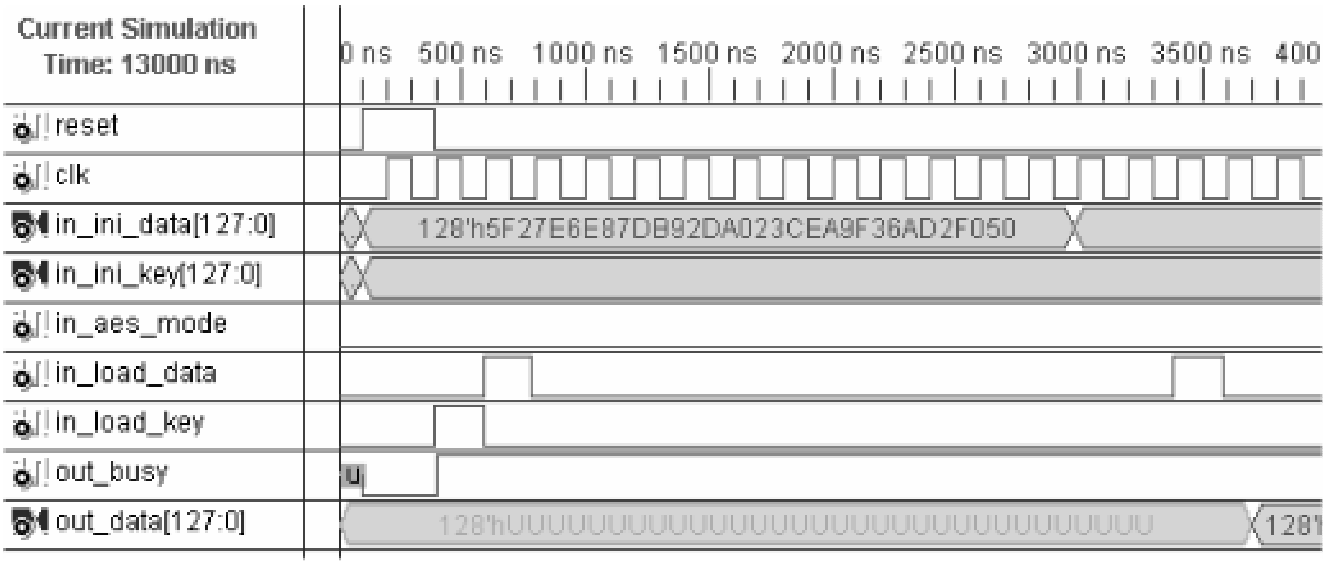

Figura 5.25: Testbench do AES - Carga para Cifragem

Na decifragem, o tempo inicial necessário para a entrega da palavra é de aproximadamente 30 ciclos de clock. Esse tempo inclui o processo de expansão da chave e decifragem da palavra. O tempo inicial para carregar a palavra, expandir a chave e decifrá-la é maior do que o tempo para cifrá-la.

O funcionamento do hardware AES-128 pode ser visto através das Figuras 5.26, 5.27 e 5.28, como será descrito a seguir. As três figuras citadas foram cortadas mas são partes de um mesmo testbench que verifica as funcionalidades do hardware desenvolvido.

A Figura 5.26 mostra que, inicialmente, o sinal in_load_key é colocado em nível alto e a palavra que está em in_ini_key é carregada. O sinal in_aes_mode está em nível baixo o que significa que o hardware está trabalhando em modo *encrypt* (cifragem). O sinal in_load_data é colocado em nível alto e a palavra que está em in_ini_data é carregada.

O resultado desta parte (I) será a cifragem entre:

PALAVRA: *0x*5f27e6e87db92da023cea9f36ad2f050

CHAVE: *0x*5f27e6e87db92da023cea9f36ad2f050

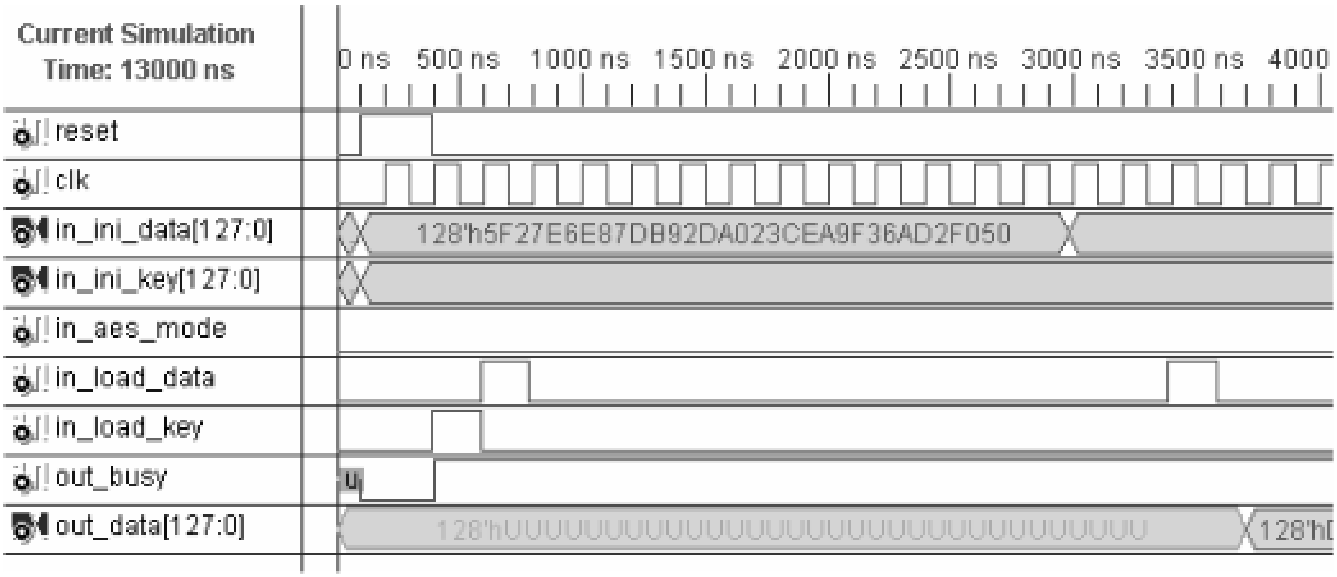

Figura 5.26: Testbench do AES – parte I

A Figura 5.27 mostra que o sinal in_load_data é colocado novamente em nível alto e a palavra que está em in_ini_data é carregada. O sinal in_aes_mode está em nível baixo o que significa que o hardware está cifrando. O resultado da parte I é entregue na saída out_data (0xd3f47cc2d34cde3255ee70e79d98290d).

O resultado desta parte (II) será a cifragem entre:

PALAVRA: *0x*d3f47cc2d34cde3255ee70e79d98290d

CHAVE: *0x*5f27e6e87db92da023cea9f36ad2f050

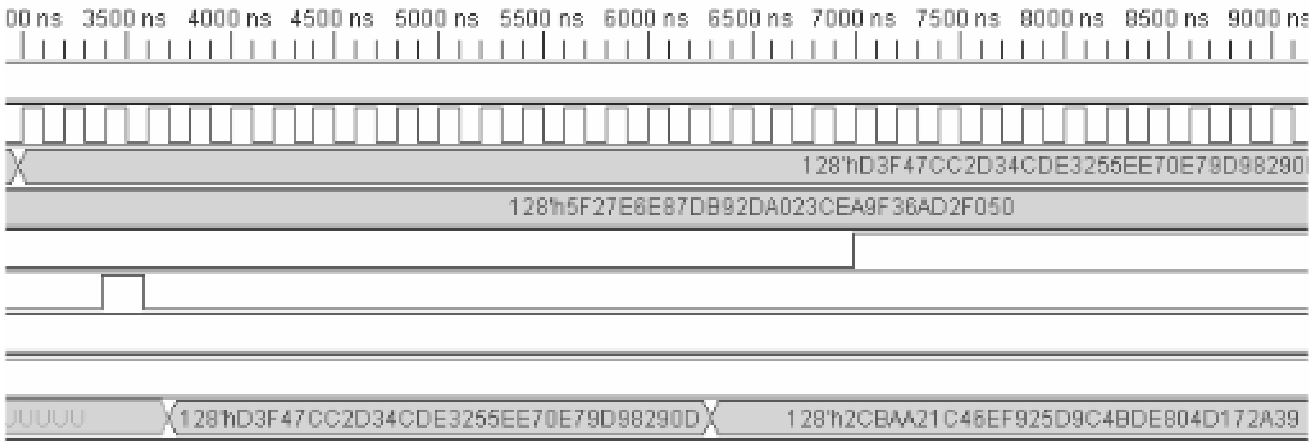

Figura 5.27: Testbench do AES – parte II

Na Figura 5.28 não há nova palavra, nem chave carregados. O sinal in_aes_mode é colocado em nível alto o que significa que o hardware irá decifrar. O resultado da parte II é entregue na saída out_data. O resultado desta parte (III) será a decifragem entre:

PALAVRA: *0x*d3f47cc2d34cde3255ee70e79d98290d

CHAVE: *0x*5f27e6e87db92da023cea9f36ad2f050

O resultado da fase III é entregue na saída out_data :

PALAVRA: *0x*5f27e6e87db92da023cea9f36ad2f050

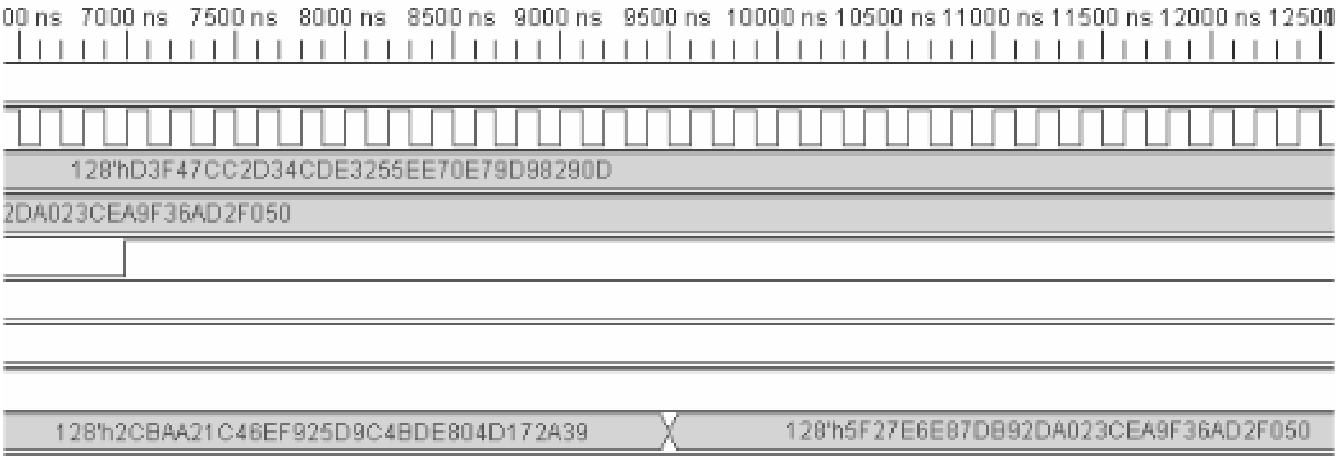

Figura 5.28: Testbench do AES – parte III

5.5 - Testes e validação do hardware

A validação do software e do hardware desenvolvidos foi feita utilizando o padrão fornecido pelos criadores do algoritmo conforme [09], [10] e [11].

A Tabela 5.1 mostra um vetor de teste fornecido para verificação de todas as fases de criptografia do AES-128, tanto para hardware quanto para software. Os valores das palavras/chaves estão em hexadecimal.

Tabela 5.1: Vetor de teste para todos os passos do AES

Fase de Cifragem	Rodada	Palavra/chave de 16 bytes	Rodada	Palavra/chave de 16 bytes
Chave Inicial	00	3243f6a8885a308d313198a2e0370734		
Palavra Inicial	00	2b7e151628aed2a6abf7158809cf4f3c		
Palavra da Rodada	01	193de3bea0f4e22b9ac68d2ae9f84808	06	fl006f55c1924cef7cc88b325db5d50c
Após SubBytes	01	d42711aee0bf98f1b8b45de51e415230	06	a163a8fc784f29df10e83d234cd503fe
Após ShiftRows	01	d4bf5d30e0b452aeb84111fl1e2798e5	06	a14f3dfe78e803fc10d5a8df4c632923
Após MixColumns	01	046681e5e0cb199a48f8d37a2806264c	06	4b868d6d2c4a8980339df4e837d218
Chave da rodada	01	a0fafe1788542cb123a339392a6c7605	06	6d88a37a110b3efddbf98641ca0093f
Palavra da Rodada	02	a49c7ff2689f352b6b5bea43026a5049	07	260e2e173d41b77de86472a9fdd28b
Após SubBytes	02	49ded28945db96f17f39871a7702533b	07	f7ab31f02783a9ff9b4340d354b53d3f
Após ShiftRows	02	49db873b453953897f02d2f177de961a	07	f783403f27433df09bb531ff54aba9d3
Após MixColumns	02	584dcaf11b4b5aacdbe7caa81b6bb0e5	07	1415b5bf461615ec274656d7342ad8
Chave da rodada	02	f2c295f27a96b9435935807a7359f67f	07	4e54f70e5f5fc9f384a64fb24ea6dc4f
Palavra da Rodada	03	aa8f5f0361dde3ef82d24ad26832469a	08	5a4142b11949dclfa3e019657a8c040
Após SubBytes	03	ac73cf7befcllldf13b5d6b545235ab8	08	be832cc8d43b86c00ae1d44dda64f2f
Após ShiftRows	03	acc1d6b8efb55a7b1323cfdf457311b5	08	be3bd4fed4e1f2c80a642cc0da83864
Após MixColumns	03	75ec0993200b633353c0cf7cbb25d0dc	08	00512fdlb1c889ff54766dcdfa1b9gea
Chave da rodada	03	3d80477d4716fe3e1e237e446d7a883b	08	ead27321b58dbad2312bf5607f8d292
Palavra da Rodada	04	486c4eee671d9d0d4de3b138d65f58e7	09	ea835cf00445332d655d98ad8596b0
Após SubBytes	04	52502f2885a45ed7e311c807f6cf6a94	09	87ec4a8cf26ec3d84d4c46959790e7a
Após ShiftRows	04	52a4c89485116a28e3cf2fd7f6505e07	09	876e46a6f24ce78c4d904ad897ecc39
Após MixColumns	04	0fd6daa9603138bf6fc0106b5eb31301	09	473794ed40d4e4a5a3703aa64c9f42
Chave da rodada	04	ef44a541a8525b7fb671253bdb0bad00	09	ac7766f319fadc2128d12941575c006
Palavra da Rodada	05	e0927fe8c86363c0d9b1355085b8be01	10	eb40f21e592e38848ba113e71bc342
Após SubBytes	05	e14fd29be8fbfbba35c89653976cae7c	10	e9098972cb31075f3d327d94af2e2cb
Após ShiftRows	05	e1fb967ce8c8ae9b356cd2ba974ffb53	10	e9317db5cb322c723d2e895faf09079
Após MixColumns	05	25d1a9adbd11d168b63a338e4c4cc0b0	10	
Chave da rodada	05	d4d1c6f87c839d87caf2b8bc11f915bc	10	d014f9a8cgee258ge13f0cc8b6630ca
			Saída	3925841d02dc09fbdc118597196a0b

```
                  COLUMN_1       COLUMN_2       COLUMN_3       COLUMN_4
                 ------------------------------------------------------------
START Key        : 2b 7e 15 16  | 28 ae d2 a6  | ab f7 15 88  | 09 cf 4f 3c  |
                 ------------------------------------------------------------
Key ROUND 01     : a0 fa fe 17  | 88 54 2c b1  | 23 a3 39 39  | 2a 6c 76 05  |
Key ROUND 02     : f2 c2 95 f2  | 7a 96 b9 43  | 59 35 80 7a  | 73 59 f6 7f  |
Key ROUND 03     : 3d 80 47 7d  | 47 16 fe 3e  | 1e 23 7e 44  | 6d 7a 88 3b  |
Key ROUND 04     : ef 44 a5 41  | a8 52 5b 7f  | b6 71 25 3b  | db 0b ad 00  |
Key ROUND 05     : d4 d1 c6 f8  | 7c 83 9d 87  | ca f2 b8 bc  | 11 f9 15 bc  |
Key ROUND 06     : 6d 88 a3 7a  | 11 0b 3e fd  | db f9 86 41  | ca 00 93 fd  |
Key ROUND 07     : 4e 54 f7 0e  | 5f 5f c9 f3  | 84 a6 4f b2  | 4e a6 dc 4f  |
Key ROUND 08     : ea d2 73 21  | b5 8d ba d2  | 31 2b f5 60  | 7f 8d 29 2f  |
Key ROUND 09     : ac 77 66 f3  | 19 fa dc 21  | 28 d1 29 41  | 57 5c 00 6e  |
Key ROUND 10     : d0 14 f9 a8  | c9 ee 25 89  | e1 3f 0c c8  | b6 63 0c a6  |
```

Figura 5.29: Utilização do vetor de testes na Expansão da Chave em C++

As Figuras 5.29 e 5.30 mostram os resultados obtidos em C++ e a Figura 5.31 o resultado utilizando FPGA, e correspondem ao vetor de teste da Tabela 5.1 e estão validados.

	COLUMN_1	COLUMN_2	COLUMN_3	COLUMN_4
START Text	**: 32 43 f6 a8**	**| 88 5a 30 8d**	**| 31 31 98 a2**	**| e0 37 07 34 |**
Text ROUND 00	: 19 3d e3 be	| a0 f4 e2 2b	| 9a c6 8d 2a	| e9 f8 48 08 |
Text ROUND 01	: 19 3d e3 be	| a0 f4 e2 2b	| 9a c6 8d 2a	| e9 f8 48 08 |
SubBytes	: d4 27 11 ae	| e0 bf 98 f1	| b8 b4 5d e5	| 1e 41 52 30 |
ShiftRows	: d4 bf 5d 30	| e0 b4 52 ae	| b8 41 11 f1	| 1e 27 98 e5 |
MixColumns	: 04 66 81 e5	| e0 cb 19 9a	| 48 f8 d3 7a	| 28 06 26 4c |
AddRoundKey	: a4 9c 7f f2	| 68 9f 35 2b	| 6b 5b ea 43	| 02 6a 50 49 |
Text ROUND 02	: a4 9c 7f f2	| 68 9f 35 2b	| 6b 5b ea 43	| 02 6a 50 49 |
SubBytes	: 49 de d2 89	| 45 db 96 f1	| 7f 39 87 1a	| 77 02 53 3b |
ShiftRows	: 49 db 87 3b	| 45 39 53 89	| 7f 02 d2 f1	| 77 de 96 1a |
MixColumns	: 58 4d ca f1	| 1b 4b 5a ac	| db e7 ca a8	| 1b 6b b0 e5 |
AddRoundKey	: aa 8f 5f 03	| 61 dd e3 ef	| 82 d2 4a d2	| 68 32 46 9a |
Text ROUND 03	: aa 8f 5f 03	| 61 dd e3 ef	| 82 d2 4a d2	| 68 32 46 9a |
SubBytes	: ac 73 cf 7b	| ef c1 11 df	| 13 b5 d6 b5	| 45 23 5a b8 |
ShiftRows	: ac c1 d6 b8	| ef b5 5a 7b	| 13 23 cf df	| 45 73 11 b5 |
MixColumns	: 75 ec 09 93	| 20 0b 63 33	| 53 c0 cf 7c	| bb 25 d0 dc |
AddRoundKey	: 48 6c 4e ee	| 67 1d 9d 0d	| 4d e3 b1 38	| d6 5f 58 e7 |
Text ROUND 04	: 48 6c 4e ee	| 67 1d 9d 0d	| 4d e3 b1 38	| d6 5f 58 e7 |
SubBytes	: 52 50 2f 28	| 85 a4 5e d7	| e3 11 c8 07	| f6 cf 6a 94 |
ShiftRows	: 52 a4 c8 94	| 85 11 6a 28	| e3 cf 2f d7	| f6 50 5e 07 |
MixColumns	: 0f d6 da a9	| 60 31 38 bf	| 6f c0 10 6b	| 5e b3 13 01 |
AddRoundKey	: e0 92 7f e8	| c8 63 63 c0	| d9 b1 35 50	| 85 b8 be 01 |
Text ROUND 05	: e0 92 7f e8	| c8 63 63 c0	| d9 b1 35 50	| 85 b8 be 01 |
SubBytes	: e1 4f d2 9b	| e8 fb fb ba	| 35 c8 96 53	| 97 6c ae 7c |
ShiftRows	: e1 fb 96 7c	| e8 c8 ae 9b	| 35 6c d2 ba	| 97 4f fb 53 |
MixColumns	: 25 d1 a9 ad	| bd 11 d1 68	| b6 3a 33 8e	| 4c 4c c0 b0 |
AddRoundKey	: f1 00 6f 55	| c1 92 4c ef	| 7c c8 8b 32	| 5d b5 d5 0c |
Text ROUND 06	: f1 00 6f 55	| c1 92 4c ef	| 7c c8 8b 32	| 5d b5 d5 0c |
SubBytes	: a1 63 a8 fc	| 78 4f 29 df	| 10 e8 3d 23	| 4c d5 03 fe |
ShiftRows	: a1 4f 3d fe	| 78 e8 03 fc	| 10 d5 a8 df	| 4c 63 29 23 |
MixColumns	: 4b 86 8d 6d	| 2c 4a 89 80	| 33 9d f4 e8	| 37 d2 18 d8 |
AddRoundKey	: 26 0e 2e 17	| 3d 41 b7 7d	| e8 64 72 a9	| fd d2 8b 25 |
Text ROUND 07	: 26 0e 2e 17	| 3d 41 b7 7d	| e8 64 72 a9	| fd d2 8b 25 |
SubBytes	: f7 ab 31 f0	| 27 83 a9 ff	| 9b 43 40 d3	| 54 b5 3d 3f |
ShiftRows	: f7 83 40 3f	| 27 43 3d f0	| 9b b5 31 ff	| 54 ab a9 d3 |
MixColumns	: 14 15 b5 bf	| 46 16 15 ec	| 27 46 56 d7	| 34 2a d8 43 |
AddRoundKey	: 5a 41 42 b1	| 19 49 dc 1f	| a3 e0 19 65	| 7a 8c 04 0c |
Text ROUND 08	: 5a 41 42 b1	| 19 49 dc 1f	| a3 e0 19 65	| 7a 8c 04 0c |
SubBytes	: be 83 2c c8	| d4 3b 86 c0	| 0a e1 d4 4d	| da 64 f2 fe |
ShiftRows	: be 3b d4 fe	| d4 e1 f2 c8	| 0a 64 2c c0	| da 83 86 4d |
MixColumns	: 00 51 2f d1	| b1 c8 89 ff	| 54 76 6d cd	| fa 1b 99 ea |
AddRoundKey	: ea 83 5c f0	| 04 45 33 2d	| 65 5d 98 ad	| 85 96 b0 c5 |
Text ROUND 09	: ea 83 5c f0	| 04 45 33 2d	| 65 5d 98 ad	| 85 96 b0 c5 |
SubBytes	: 87 ec 4a 8c	| f2 6e c3 d8	| 4d 4c 46 95	| 97 90 e7 a6 |
ShiftRows	: 87 6e 46 a6	| f2 4c e7 8c	| 4d 90 4a d8	| 97 ec c3 95 |
MixColumns	: 47 37 94 ed	| 40 d4 e4 a5	| a3 70 3a a6	| 4c 9f 42 bc |
AddRoundKey	: eb 40 f2 1e	| 59 2e 38 84	| 8b a1 13 e7	| 1b c3 42 d2 |
Text ROUND 10	: eb 40 f2 1e	| 59 2e 38 84	| 8b a1 13 e7	| 1b c3 42 d2 |
SubBytes	: e9 09 89 72	| cb 31 07 5f	| 3d 32 7d 94	| af 2e 2c b5 |
ShiftRows	: e9 31 7d b5	| cb 32 2c 72	| 3d 2e 89 5f	| af 09 07 94 |
AddRoundKey	: 39 25 84 1d	| 02 dc 09 fb	| dc 11 85 97	| 19 6a 0b 32 |
Encrypted Text	**: 39 25 84 1d**	**| 02 dc 09 fb**	**| dc 11 85 97**	**| 19 6a 0b 32 |**

Figura 5.30: Utilização do vetor de testes na Cifragem em C++

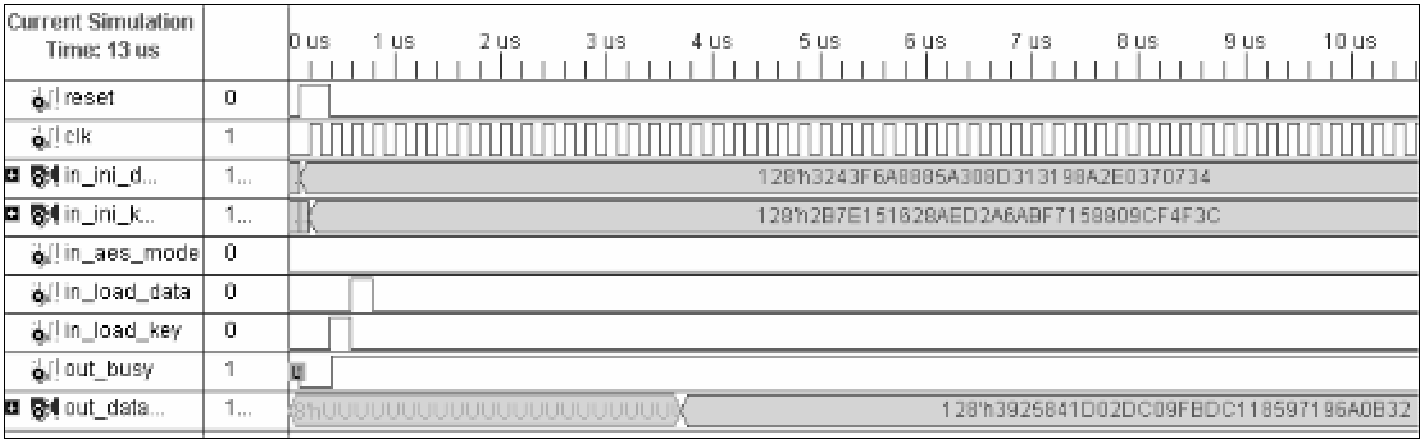

Figura 5.31: Utilização do vetor de testes na Cifragem em VHDL

5.6 - Utilização de Pipeline

Pipeline é uma técnica de implementação que permite a sobreposição temporal das diversas fases de execução das instruções. Aumenta o número de instruções executadas simultaneamente e a taxa de instruções iniciadas e terminadas por unidade de tempo. O pipeline não reduz o tempo gasto para completar cada instrução individualmente.

No desenvolvimento e nos resultados apresentados até agora não foi utilizado pipeline. Este hardware desenvolvido sem pipeline recebe/entrega uma palavra criptografada a cada 13,5ns e tem uma latência 13,5ns. Os resultados foram obtidos através de simulações utilizando a ferramenta Xilinx ISE.

A Figura 5.32 mostra como é o funcionamento sem pipeline. A primeira linha representa os 11 ciclos de expansão da chave, a segunda linha representa a cifragem da palavra. Nesta arquitetura não é possível o processamento de mais de uma palavra por vez. É necessário aguardar 10 ciclos de clock (representados pelos quadrados pretos) até que uma nova palavra possa ser carregada e processada.

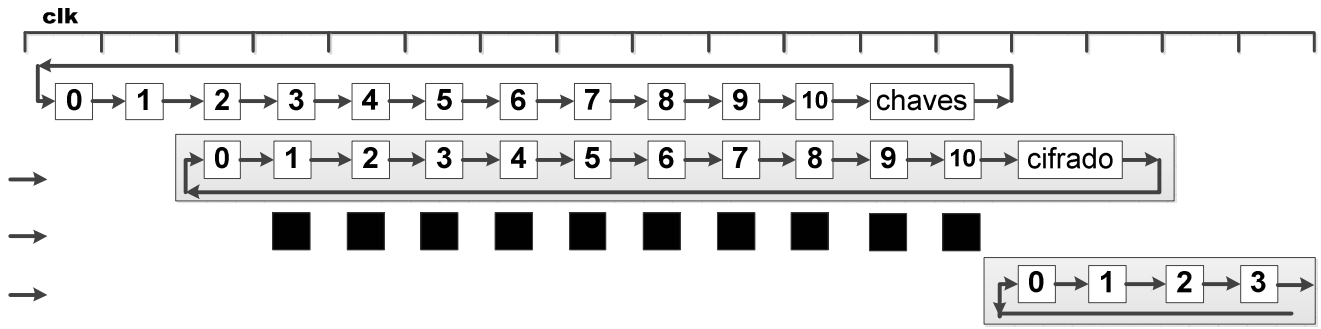

Figura 5.32: Funcionamento sem Pipeline

5.7 - Funcionamento do Pipeline

O hardware desenvolvido neste trabalho é modular e devido a este fato foi possível implementar pipelines de 2 e 5 níveis. Com os mesmos módulos é possível que se implemente outros níveis de pipeline. A Tabela 5.2 lista as características de cada uma das implementações com pipeline. Estas implementações foram feitas utilizando a FPGA Virtex-5 (XC5VFX70T) e os valores são resultado de simulações na mesma FPGA.

Tabela 5.2: Comparação entre as implementações com pipeline

Níveis de Pipeline	Tempo de Entrega e/ou Recebimento da Palavra [ns]	Latência [ns]
1	13,5	13,5
2	7,5	15,5
5	6	28

Na Tabela 5.2 pode ser visto que, com o aumento no número de níveis de pipeline tem-se uma maior velocidade na entrega e/ou recebimento das palavras, o que era de se esperar. Por outro lado a latência, que está ligada ao processamento da palavra, aumenta significativamente. A implementação de pipeline aumentou a velocidade com que as palavras foram lidas/entregues no barramento, mas a latência também aumentou o que não é desejável.

Uma das maneiras de diminuir a latência é verificar o roteamento da placa e otimizá-lo ou empregar alguma técnica de pipeline visando o mesmo objetivo. Espera-se que, em um desenvolvimento futuro, sejam estudadas e utilizadas técnicas de pipeline para que a latência venha a diminuir, aumentando assim a eficiência deste hardware.

A Figura 5.32 mostra o funcionamento do pipeline de criptografia de 5 níveis. A primeira linha representa os 11 ciclos de expansão da chave, a segunda linha representa a cifragem da primeira palavra recebida. A terceira linha representa a cifragem da segunda palavra recebida, e assim por diante até a sexta linha. A sétima linha representa que o hardware não possui capacidade para receber nenhuma outra palavra durante 6 ciclos de clock (representados pelos quadrados pretos) até que alguma das linhas esteja disponível para que uma palavra possa ser carregada e processada.

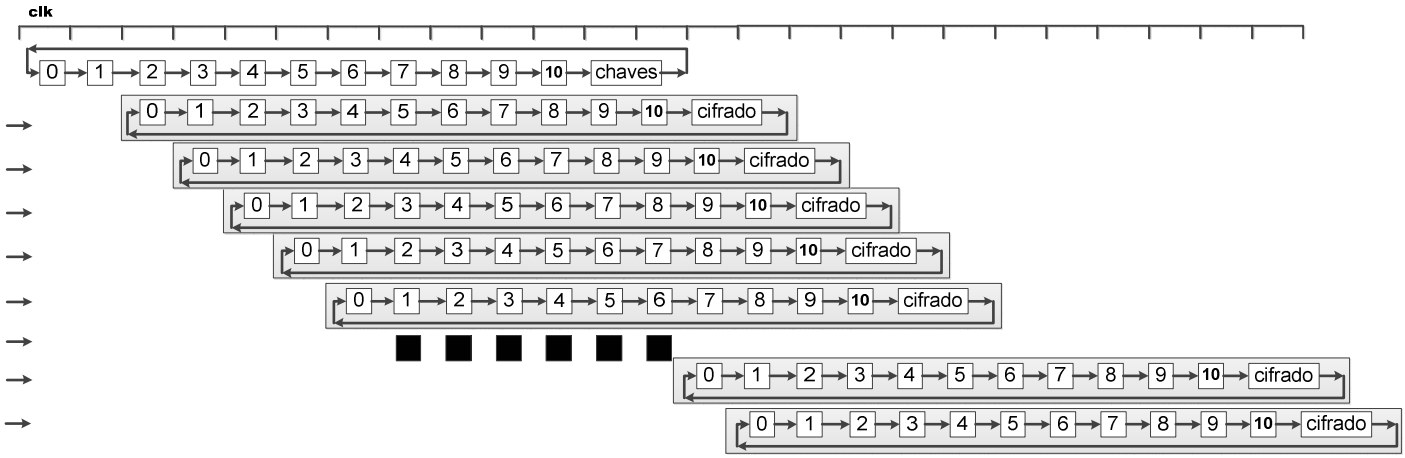

Figura 5.33: Funcionamento do Pipeline de 5 níveis

A utilização de níveis de pipeline foi feita para verificar a eficiência dos módulos e sua utilização em paralelo.

Capítulo 6

Diferenciais e Vantagens

6.1 - Comparação com outras implementações em FPGA

Com o objetivo de verificar a eficiência do hardware desenvolvido foram selecionados alguns trabalhos atuais e que tem como objetivo a implementação do AES-128 bits em FPGA. Foi utilizado o hardware sem pipelines para esta comparação.

Para que fosse feita a comparação de desempenho foram selecionados os seguintes trabalhos já publicados [14][15][16][17][18][19][20], e este trabalho foi sintetizado no dispositivo Xilinx Spartan-3 (XC3S4000) com o mesmo propósito. Conforme pode ser observado na Tabela 6.1, este trabalho obteve uma freqüência 50% maior que o dispositivo mais rápido dentre os publicados.

Para que a tecnologia da placa não influenciasse na comparação, a FPGA Xilinx Spartan-3 foi escolhida por ter sido utilizada por alguns dos trabalhos relatados ([18] e [20]). Verificando a velocidade em outro dispositivo, a FPGA Xilinx Virtex-5, obteve-se um valor maior, alcançando assim um resultado significativo no aumento da freqüência.

Tabela 6.1: Comparação com outras implementações do AES em FPGA

Implementação	FPGA Utilizada	Ano	Barramento	Freq. (MHz)
C. Chien [14]	Xilinx Virtex-II (XC2V1000)	2002	128	75,00
I. Aigredo-Badillo [15]	Xilinx Virtex-II (XC2V1000)	2006	128	96,42
J. Zambreno [16]	Xilinx Virtex-II (XC2V4000)	2004	128	110,16
E. J. Swankoski [17]	Xilinx Virtex-II Pro (XC2VP50)	2004	128	145,05
E. Lopez-Trejo [18]	Xilinx Spartan-3 (XC3S4000)	2005	128	100,08
A. Aziz & N. Ikram [19]	Xilinx Spartan-3 (XC3S50)	2007	128	165,00
Dur-e-Shahwar, Zaka, Qurat-Ul-Ain and Aziz [20]	Xilinx Spartan-3 (XC3S4000)	2009	128	206,28
Este trabalho	**Xilinx Spartan-3 (XC3S4000)**	**2011**	**128**	**318,49**
	Xilinx Virtex-5 (XC5VFX70T)	**2011**	**128**	**896,05**

Uma vantagem deste trabalho sobre [20] é que o código desenvolvido neste trabalho é portável podendo ser utilizado em quaisquer famílias de FPGA que aceitem os padrões VHDL, já [20] é um desenvolvimento que depende da arquitetura Xilinx para sua síntese.

6.2 - Artigos aceitos em Congressos Internacionais

Um artigo referente a este trabalho foi enviado ao Congresso IBERCHIP'2011, sob o título: *"A Fast AES Cryptography Circuit in FPGA"*. Este congresso é patrocinado pelo IEEE e ocorrerá na cidade de Bogotá, na Colômbia, no período de 23 a 25 de Fevereiro de 2011. O trabalho foi aceito para uma apresentação oral.

Um segundo artigo referente a este trabalho foi enviado ao Congresso LASCAS'2011, sob o título: *"A Highly Efficient FPGA Implementation of AES Cryptography"*. Este congresso é patrocinado pelo IEEE e ocorrerá na cidade de Bogotá, na Colômbia, no período de 23 a 25 de Fevereiro de 2011. O trabalho foi aceito para uma apresentação oral.

Um terceiro artigo foi enviado ao Congresso WCSIT'2011, sob o título: *"A Customized, Fast and Portable FPGA Implementation of AES Cryptography"*. Este congresso ocorrerá na cidade de Cairo, no Egito, no período de 24 a 27 de Janeiro de 2011. O trabalho foi aceito para uma apresentação oral. Um exemplar do artigo enviado consta no Apêndice deste trabalho e as cartas de aceitação estão mostradas a seguir:

The 2011 World Congress on Computer Science and Information Technology

WCSIT'11

January 24-27, 2011

Email: admin@infomesr.org		Phone: 002-010-6896063

Acceptance Letter

Dear: *Otávio S. M. Gomes, Robson L. Moreno, Tales C. Pimenta*

We would like to inform you that, after peer reviewing, the scientific committee of WCSIT'11 accepted your paper entitled with:

A Customized, Fast and Portable FPGA Implementation of AES Cryptography

By

Otávio S. M. Gomes, Robson L. Moreno, Tales C. Pimenta

The paper will be included in the conference program for oral presentation (not more than 20 minutes). According to the conference policy, the conference proceedings will be edited after the conference. Only the papers presented in the conference and accepted by the session board will be included in the proceedings. If the paper is not presented and discussed in the scientific sessions, it will not appear in the conference proceedings.

The conference is organized in Cairo, Egypt, in the period 24 – 27 January 2011. The exact timing of your paper presentation appears in the conference program.

Looking forward for meeting you in the conference, please accept my best wishes.

Sincerely Yours,

(*H. M. Abd-El-Moaatamad*)

Dr. H. M. Abd-El-Moaatamad
WCSIT-2011 Reporter

IBERCHIP 2011 submission 16

IBERCHIP 2011 <iberchip2011@easychair.org> **24 de dezembro de 2010 19:23**
Para: "Otávio S. M. Gomes"

Dear authors,

I am pleased to inform you that your paper has been accepted for an oral presentation during Iberchip 2011. Please prepare the final version of your article, carefully following all format instructions, and submit it before January 12, 2011.

Happy holidays and a great 2011!

Sincerely:

Roberto Murphy and Marius Strum.

LASCAS 2011 notification for paper 86

LASCAS 2011 <lascas2011@easychair.org> **27 de dezembro de 2010 20:06**
Para: Otavio Gomes

The program committee informs to you that your paper has accepted for presentation in the LASCAS 2011. Please send the camera-ready version before January 12-2011.
Please take note that the paper must comply with the following IEEE specifications (download IEEE Manuscript Template). :

- Paper Language: English;
- Paper Length: Maximum 4 pages, including figures, tables and references;
- Paper Size: US Letter (8.5" x 11");
- Paper formatting: single spaced, double-column, 9pt font or larger;
- Margins (nominal): Top: 0.85" (22mm), Bottom: 0.85" (22mm), Left &anp; Right: 0.7" (18mm);
- Fonts: Embed & Subset ALL fonts in your PDF file. No Type 3 fonts allowed;
- File Format: PDF only;
- File Size Limitation: 2.0MB;
- Do NOT page number your paper;
- Do NOT apply security settings to your PDF file;
- Note: Deviations from the above specifications could impede the review of your paper and result in automatic rejection.

---------- REVIEW 1 ----------
PAPER: 86
TITLE: A Highly Efficient FPGA Implementation of AES Cryptography

The paper is very well organized and addresses an interesting application. It should be included in the program.

Capítulo 7

Proposta para Aplicações e Trabalhos Futuros

A comparação de desempenho entre as arquiteturas de criptografia de 128, 192 e 256 bits é uma oportunidade para o desenvolvimento de trabalhos futuros a partir deste. A mudança no barramento das chaves (que está configurado para 128bits) e algumas alterações no bloco de expansão da chave proporcionarão a possibilidade de comparação de desempenho entre os níveis de segurança.

A adaptação deste trabalho para que funcione como interface de criptografia para um dispositivo USB também é uma proposta de aplicação e trabalho futuro. Esta oportunidade poderá unir este trabalho ao que está sendo desenvolvido pelo Grupo de Microeletrônica junto ao Programa Brazil-IP.

A utilização deste desenvolvimento em Smart Metering e Smart Grids, além de sua utilização em conjunto com os microprocessadores existentes das famílias de FPGA's (MicroBlaze, PicoBlaze e Nios II) também é uma proposta de aplicação e trabalho futuro.

Além das propostas acima, há, também, a utilização como interface de criptografia para aplicações biomédicas na Transferência de Dados Confidenciais, área que é de interesse do Grupo de Microeletrônica.

Capítulo 8

Conclusões

Durante o desenvolvimento deste trabalho foi possível conhecer mais sobre a área de segurança na transmissão de informações tanto em hardware quanto em software. Foi possível testar e validar este conhecimento através do desenvolvimento de um software que realiza a cifragem e decifragem seguindo o padrão AES.

Foi possível também realizar os testes e validações necessárias para que o hardware funcionasse conforme as especificações e atingisse a eficiência e a customização esperadas, além da análise de seu funcionamento com alguns níveis de *pipeline* e da comparação de eficiência com outros trabalhos que foram desenvolvidos na área acadêmica internacional.

Alcançou-se os objetivos e neste trabalho foi desenvolvido um hardware de criptografia eficiente e com estrutura modular, que poderá ser utilizado em várias aplicações.

Esse desenvolvimento representa uma melhoria nos trabalhos conhecidos (conforme foi mostrado) e um ponto de partida para futuros desenvolvimentos nesta área pelo Grupo de Microeletrônica.

Referências Bibliográficas

[1] TERADA, Routo. **Segurança de Dados: Criptografia em Redes de Computador**. Edgard Blucher, 2000.

[2] STALLINGS, William. **Cryptography and Networks Security: Principles and Practices**. Ed. Prentice Hall, Second Edition, 1999.

[3] MITSURI, Matsui. **Linear Criptanalysis Method for DES Cipher**. In advances in Criptology, Eurocrypt 93, Lectures Notes in Computer Science Vol. 765, Springer Verlag, pp. 386-397, 1993.

[4] FIPS-197, **Federal Information Processing Standards Publication FIPS-197, Advanced Encryption Standard (AES)**, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, October 2010.

[5] DAEMEN, J. and RIJMEN, V.. **The design of Rijndael: AES - The Advanced Encryption Standard**. Springer-Verlag. 2002.

[6] CASILLO, L. **Projeto e implementação em FPGA de um processador com conjunto de instrução reconfigurável utilizando VHDL**. Universidade Federal do Rio Grande do Norte, Natal-RN. 2005.

[7] ARANTES, D and CARDOSO, F. **FPGA e Fluxo de Projeto**. In: DECOM-FEECUNICAMP, Campinas-SP. 2008.

[8] PEDRONI, Volnei A. **Circuit Design with VHDL**. Cambridge, MA: MIT Press, 2004.

[9] BOTROS, Nazeih M. **HDL Programming Fundamentals: VHDL and Verilog**. Davinci Engineering, 2005.

[10] MORENO, E. D.; PEREIRA, F. D.; CHIARAMONTE, R. B. **Criptografia em Software e hardware**. Novatec. 2005.

[11] SOUZA, Raquel and BORGES, Flávio. **Aplicações de GF(28) no algoritmo AES**. Laboratório Nacional de Computação Científica, Petrópolis/RJ

[12] DAEMEN, J. and RIJMEN, V. **A Specification for The AES Algorithm**. NIST (National Institute of Standards and Technology). http://csrc.nist.gov/archive/aes/rijndael/wsdindex.html, October 2010.

[13] DAEMEN, J.; RIJMEN, V. **AES Proposal: Rijndael**. 1999.

[14] CHIEN, C; CHIEN, D; VERBAUWHEDE I. and CHANG F. **A hardware implementation in FPGA of the Rijndael algorithm.** The 2002 45th Midwest Symp. Circuits and Systems (MWSCAS-2002), Vol. 1,4--.7 August 2002, pp. 507-509.

[15] ALGREDO-BADILLO, I.; FEREGRINO-URIBE, C. and CUMLIDO-PARRA, R. **Design and implementation of an FPGA-based 1.452 Gbps non-pipelined AES architecture**, The 2006 Int. Conf. Computational Science and Its Applications (ICCSA 2006), Lecture Notes in Computer Science, Vol. 3982 (Springer-Verlag, 2006), pp. 446--455.

[16] ZAMBRENO, J.; NGUYEN, D. and CHOUDHARY, A. **Exploring area/delay tradeoffs in an AES FPGA implementation**, Proc. Int. Colif, FieldProgrammable Logic and Its Applications (FPL), Lecture Notes in Computer Science, Vol. 3203 (Springer-Verlag 2004), pp. 575-585.

[17] SWANKOSKI, E. J.; NARAYANAN, V.; KANDEMIR M. and Irwin, M. J. **A parallel architecture for secure FPGA symmetric encryption.** 18th Int. Parallel and Distributed Processing Symp. (IPDPS'04) - Workshop, Santa Fe, New Mexico, 26-.30 April 2004, p. 123.

[18] LOPEZ-TREJO, E.; RODRIGUEZ-HENRIQUEZ, F. and DIAZ-PEREZ, A. **An efficient FPGA implementation of CCM using AES**, The 8th Int. Conf. Information Security and Cryptology (ICJSC'05). Lecture Notes in Computer Science (Springer 2005), pp. 208-215.

[19] AZIZ, Arshad and IKRAM, Nassar. **Memory efficient implementation of AES S-boxes on FPGA**. Journal of Circuits, Systems, and Computers, Vol. 16, No.4 (2007) 603--611.

[20] KUNDI, D.; ZAKA, S., QURAT-UL-AIN and AZIZ, Arshad. **A Compact AES Encryption Core on Xilinx FPGA**. 2nd IEEE International Conference on Computer, Control & Communication (IEEE IC4-2009) Karachi, Pakistan Vol:1 pp:1-4. 2009.

Apêndice

- AES-128 em C++

```
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>

// Número de colunas na palavra do AES.
#define Nb 4

// Número de rounds na cifragem do AES. Nesse caso é igual a 128, que implica em (128/32)+6
= 10 rodadas [rounds]
int Nr=128;

// Número de palavras de 32 bits na chave. 128/32 = 4.
int Nk=4;

// in - a palavra de entrada, que representa o texto a ser encriptado ou decriptado.
// out - a palavra de saída, que representa o texto decriptado ou encriptado.
// state - palavra no seu estado intermediário, durante algum dos cálculos (Rotate, SubByte,
ShiftRow, MixColumns, RoundKey).
unsigned char in[16], out[16], state[4][4];

// O array que guarda a chave após sua expansão.
unsigned char RoundKey[240];

// A chave com que a palavra será cifrada ou decifrada
// No AES (Rijndael) usa-se chave simétrica
unsigned char Key[32];

#define xtime(x)   ((x<<1) ^ (((x>>7) & 1) * 0x1b))
#define Multiply(x,y) (((y & 1) * x) ^ ((y>>1 & 1) * xtime(x)) ^ ((y>>2 & 1) *
xtime(xtime(x))) ^ ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^ ((y>>4 & 1) *
xtime(xtime(xtime(xtime(x))))))


int getSBoxValue(int num){
       int sbox[256] =   {
//0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, 0xca,
0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, 0xb7, 0xfd,
0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
       };
```

```
	return sbox[num];
}

int getSBoxInvert(int num){
	int rsbox[256] = {
0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb
, 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb
, 0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e
, 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25
, 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92
, 0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84
, 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06
, 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b
, 0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73
, 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e
, 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b
, 0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4
, 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f
, 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef
, 0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61
, 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
	};
	return rsbox[num];
}

// A constante utilizada nos cálculos (round constant word array), Rcon[i]
// Seu cálculo utiliza conceitos da matemática polinomial de Galois.
// i inicia com 1, não na posição 0
int Rcon[255] = {
0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb
};


// Realiza a expansão da Chave que será utilizada para cifrar/decifrar a palavra
void KeyExpansion(){
	int i,j;
	unsigned char temp[4],k;

	// A rodada inicial consiste na própria palavra
	for(i=0;i<Nk;i++){
		RoundKey[i*4]=Key[i*4];
		RoundKey[i*4+1]=Key[i*4+1];
		RoundKey[i*4+2]=Key[i*4+2];
		RoundKey[i*4+3]=Key[i*4+3];
	}

	printf("\n           COLUMN_1       COLUMN_2       COLUMN_3       COLUMN_4");

	printf("\nSTART Key      : ");
	for(int y=0;y<Nb*4;y++){
		printf("%02x ",RoundKey[y]);
		if ((y+1)%4==0){ printf(" | "); }
	}
```

```
	// As rodadas seguintes utilizam as chaves calculadas anteriormente
	while (i < (Nb * (Nr+1))){
		for(j=0;j<4;j++){
			temp[j]=RoundKey[(i-1) * 4 + j];
		}
		if (i % Nk == 0){
			// Função Rotate
				k = temp[0];
				temp[0] = temp[1];
				temp[1] = temp[2];
				temp[2] = temp[3];
				temp[3] = k;

			// Função SubByte
				temp[0]=getSBoxValue(temp[0]);
				temp[1]=getSBoxValue(temp[1]);
				temp[2]=getSBoxValue(temp[2]);
				temp[3]=getSBoxValue(temp[3]);

	//Cálculo envolvendo a primeira coluna e a matriz Round Word Constant Array
				temp[0] =  temp[0] ^ Rcon[i/Nk];
		}

		//Cálculo utilizando XOR e combinação entre as colunas
		RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
		RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
		RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
		RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];

		printf("%02x ",RoundKey[i*4+0]);
		printf("%02x ",RoundKey[i*4+1]);
		printf("%02x ",RoundKey[i*4+2]);
		printf("%02x ",RoundKey[i*4+3]);
		printf(" | ");

		i++;
	}
	printf("\n
}


//Função AddRoundKey - ENCRYPT
//Adiciona a chave do round à palavra calculada (Rotate, SubByte,ShiftRows,MixColumns)
void AddRoundKey(int round){
	int i,j;
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb + j];
		}
	}
}


//Função SubBytes - ENCRYPT
//Realiza a substituição do valor da célula pelo valor correspondente na tabela S-Box
void SubBytes()
{
	int i,j;
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			state[i][j] = getSBoxValue(state[i][j]);
		}
	}
}
```

```
//Função ShiftRows - ENCRYPT
//Rotaciona as linhas para a esquerda conforme o número da linha
//sendo que a primeira linha (núm.=0) não sofre rotação
void ShiftRows(){
	unsigned char temp;

	//Rotaciona a 2ª linha uma coluna à esquerda
	temp=state[1][0];
	state[1][0]=state[1][1];
	state[1][1]=state[1][2];
	state[1][2]=state[1][3];
	state[1][3]=temp;

	//Rotaciona a 3ª linha duas colunas à esquerda
	temp=state[2][0];
	state[2][0]=state[2][2];
	state[2][2]=temp;

	temp=state[2][1];
	state[2][1]=state[2][3];
	state[2][3]=temp;

	//Rotaciona a 3ª linha três colunas à esquerda
	temp=state[3][0];
	state[3][0]=state[3][3];
	state[3][3]=state[3][2];
	state[3][2]=state[3][1];
	state[3][1]=temp;
}

//Função MixColumns - ENCRYPT
//Realiza a mistura (mix) das colunas com uma matriz de estados já calculada [Galois]
void MixColumns(){
	int i;
	unsigned char Tmp,Tm,t;
	for(i=0;i<4;i++){
	   t=state[0][i];
	   Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i] ;
	   Tm = state[0][i] ^ state[1][i] ; Tm = xtime(Tm); state[0][i] ^= Tm ^ Tmp ;
	   Tm = state[1][i] ^ state[2][i] ; Tm = xtime(Tm); state[1][i] ^= Tm ^ Tmp ;
	   Tm = state[2][i] ^ state[3][i] ; Tm = xtime(Tm); state[2][i] ^= Tm ^ Tmp ;
	   Tm = state[3][i] ^ t ; Tm = xtime(Tm); state[3][i] ^= Tm ^ Tmp ;
	}
}

//Função Cipher - ENCRYPT
//Encripta a palavra fornecida com a chave, também fornecida
void Cipher(){
	int i,j,round=0;

	printf("\n\n\n");
	printf("\n               COLUMN_1       COLUMN_2       COLUMN_3      COLUMN_4");

	printf("\nSTART Text     : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}

	//ROUND 0 - Rodada inicial
	//Realiza a função XOR entra a palavra inicial e a chave fornecida
	//Calcula a palavra para o 1º Round
	AddRoundKey(0);
```

```
printf("\nText ROUND 00  : ");
for(i=0;i<4;i++){
	for(j=0;j<4;j++){
		printf("%02x ",state[j][i]);
	}
	printf(" | ");
}

printf("\nText ROUND 01  : ");
for(i=0;i<4;i++){
	for(j=0;j<4;j++){
		printf("%02x ",state[j][i]);
	}
	printf(" | ");
}
printf("\n");

// Número de rounds: Nr = 10 (128 bits)
// Os primeiros 9 (Nr-1) rounds são idênticos
for(round=1;round<Nr;round++)
{
	SubBytes();
	printf("  SubBytes     : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}

	ShiftRows();
	printf("\n  ShiftRows    : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}

	MixColumns();
	printf("\n  MixColumns   : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}

	AddRoundKey(round);
	printf("\n  AddRoundKey  : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}

	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}
	printf("\n");
}
```

```
	// O último round (10°) não realiza o cálculo MixColumns
	SubBytes();
		printf("  SubBytes     : ");
		for(i=0;i<4;i++){
			for(j=0;j<4;j++){
				printf("%02x ",state[j][i]);
			}
			printf(" | ");
		}

		ShiftRows();
		printf("\n  ShiftRows    : ");
		for(i=0;i<4;i++){
			for(j=0;j<4;j++){
				printf("%02x ",state[j][i]);
			}
			printf(" | ");
		}

		AddRoundKey(round);
		printf("\n  AddRoundKey  : ");
		for(i=0;i<4;i++){
			for(j=0;j<4;j++){
				printf("%02x ",state[j][i]);
			}
			printf(" | ");
		}

	//A  Palavra fornecida já está encriptada
	//Saída de texto com a palavra encriptada
	printf("\nEncrypted Text : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			out[i*4+j]=state[j][i];
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}
	printf("\n");
}

//Função InvSubBytes - DECRYPT
void InvSubBytes(){
	int i,j;
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			state[i][j] = getSBoxInvert(state[i][j]);
		}
	}
}

//Função InvShiftRows - DECRYPT
void InvShiftRows(){
	unsigned char temp;

	//Rotaciona a linha1, uma coluna para a direita
	temp=state[1][3];
	state[1][3]=state[1][2];
	state[1][2]=state[1][1];
	state[1][1]=state[1][0];
	state[1][0]=temp;

	//Rotaciona a linha 2, duas colunas para a direita
	temp=state[2][0];
```

```
	state[2][0]=state[2][2];
	state[2][2]=temp;

	temp=state[2][1];
	state[2][1]=state[2][3];
	state[2][3]=temp;

	//Rotaciona a linha 3, três colunas para a direita
	temp=state[3][0];
	state[3][0]=state[3][1];
	state[3][1]=state[3][2];
	state[3][2]=state[3][3];
	state[3][3]=temp;
}

// Função MixColumns - DECRYPT
void InvMixColumns(){
	int i;
	unsigned char a,b,c,d;
	for(i=0;i<4;i++){

		a = state[0][i];
		b = state[1][i];
		c = state[2][i];
		d = state[3][i];

		state[0][i] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c, 0x0d) ^
Multiply(d, 0x09);
		state[1][i] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c, 0x0b) ^
Multiply(d, 0x0d);
		state[2][i] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c, 0x0e) ^
Multiply(d, 0x0b);
		state[3][i] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c, 0x09) ^
Multiply(d, 0x0e);
	}
}


// Função InvCipher - DECRYPT
void InvCipher(){
	int i,j,round=0;

	printf("\n\n\n");
	printf("\n               COLUMN_1       COLUMN_2       COLUMN_3       COLUMN_4");

	printf("\nEncrypted Text : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}

	AddRoundKey(Nr);
	printf("\nText ROUND 00  : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}

	printf("\nText ROUND 10  : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
```

```
                printf("%02x ",state[j][i]);
        }
        printf(" | ");
}
printf("\n");

for(round=Nr-1;round>0;round--){
        InvShiftRows();
        printf("  InvShiftRows : ");
        for(i=0;i<4;i++){
                for(j=0;j<4;j++){
                        printf("%02x ",state[j][i]);
                }
                printf(" | ");
        }

        InvSubBytes();
        printf("\n  InvSubBytes  : ");
        for(i=0;i<4;i++){
                for(j=0;j<4;j++){
                        printf("%02x ",state[j][i]);
                }
                printf(" | ");
        }

        AddRoundKey(round);
        printf("\n  AddRoundKey  : ");
        for(i=0;i<4;i++){
                for(j=0;j<4;j++){
                        printf("%02x ",state[j][i]);
                }
                printf(" | ");
        }

        InvMixColumns();
        printf("\n  InvMixColumns: ");
        for(i=0;i<4;i++){
                for(j=0;j<4;j++){
                        printf("%02x ",state[j][i]);
                }
                printf(" | ");
        }

        if (round < 10){ printf("Text ROUND 0%d  : ",round); }
        else{ printf("Text ROUND %d  : ", round); }

        for(i=0;i<4;i++){
                for(j=0;j<4;j++){
                        printf("%02x ",state[j][i]);
                }
                printf(" | ");
        }
        printf("\n");
}

InvShiftRows();
printf("  InvShiftRows : ");
        for(i=0;i<4;i++){
                for(j=0;j<4;j++){
                        printf("%02x ",state[j][i]);
                }
                printf(" | ");
        }

InvSubBytes();
```

```
	printf("\n  InvSubBytes  : ");
		for(i=0;i<4;i++){
			for(j=0;j<4;j++){
				printf("%02x ",state[j][i]);
			}
			printf(" | ");
		}

	AddRoundKey(0);
	printf("\n  AddRoundKey  : ");
		for(i=0;i<4;i++){
			for(j=0;j<4;j++){
				printf("%02x ",state[j][i]);
			}
			printf(" | ");
		}

	printf("\nDecrypted Text : ");
	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			out[i*4+j]=state[j][i];
			printf("%02x ",state[j][i]);
		}
		printf(" | ");
	}
	printf("\n");
}

int _tmain(int argc, _TCHAR* argv[]){

	int i,j ;
	char aux;
	unsigned char crytped_demo[32]= {0x39 ,0x25 ,0x84 ,0x1d ,0x02 ,0xdc ,0x09 ,0xfb ,0xdc
,0x11 ,0x85 ,0x97 ,0x19 ,0x6a ,0x0b ,0x32};
	unsigned char in_demo[32]= {0x5f  ,0x27  ,0xe6  ,0xe8  ,0x7d  ,0xb9  ,0x2d  ,0xa0
,0x23  ,0xce  ,0xa9  ,0xf3  ,0x6a  ,0xd2  ,0xf0  ,0x50};
	unsigned char key_demo[32] = {0x5f  ,0x27  ,0xe6  ,0xe8  ,0x7d  ,0xb9  ,0x2d  ,0xa0
,0x23  ,0xce  ,0xa9  ,0xf3  ,0x6a  ,0xd2  ,0xf0  ,0x50};

	Nr = 128;    Nk = Nr / 32;       Nr = Nk + 6;

	for(i=0;i<Nk*4;i++){
		Key[i]=key_demo[i];
	}
	KeyExpansion();

	for(i=0;i<Nk*4;i++){
		in[i]=in_demo[i];
	}

	for(i=0;i<4;i++){
		for(j=0;j<4;j++){
			state[j][i] = in[i*4 + j];
		}
	}
	Cipher();

	for(i=0;i<Nk*4;i++){
		in[i]=crytped_demo[i];
	}
	InvCipher();

	scanf("%d",in_demo[0]);
	return 0;
}
```

- AES-128 em VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY sbox IS
PORT(
   in_sbox    :    IN  std_logic_vector(7 downto 0);
   out_sbox   :    OUT std_logic_vector(7 downto 0)
   );
END sbox;

-- Uma parte do código foi retirada, não prejudicando o entendimento do mesmo.

ARCHITECTURE sbox_arch OF sbox IS

BEGIN

        WITH in_sbox(7 downto 0) SELECT
                out_sbox(7 downto 0) <=

                   "01100011" WHEN "00000000", --(X"63")
                   "01111100" WHEN "00000001", --(X"7C")
                   "01110111" WHEN "00000010", --(X"77")
                   "01111011" WHEN "00000011", --(X"7B")
                   "11110010" WHEN "00000100", --(X"F2")
                   "01101011" WHEN "00000101", --(X"6B")
                   "01101111" WHEN "00000110", --(X"6F")
                   "11000101" WHEN "00000111", --(X"C5")
                   "00110000" WHEN "00001000", --(X"30")
                   "00000001" WHEN "00001001", --(X"01")
                   "01100111" WHEN "00001010", --(X"67")
                   "00101011" WHEN "00001011", --(X"2B")
                   "11111110" WHEN "00001100", --(X"FE")
                   "11010111" WHEN "00001101", --(X"D7")
                   "10101011" WHEN "00001110", --(X"AB")
                   "01110110" WHEN "00001111", --(X"76")

                   "11001010" WHEN "00010000", --(X"CA")
                   "10000010" WHEN "00010001", --(X"82")
                   "11001001" WHEN "00010010", --(X"C9")
                   "01111101" WHEN "00010011", --(X"7D")
                   "11111010" WHEN "00010100", --(X"FA")
                   "01011001" WHEN "00010101", --(X"59")
                   "01000111" WHEN "00010110", --(X"47")
                   "11110000" WHEN "00010111", --(X"F0")
                   "10101101" WHEN "00011000", --(X"AD")
                   "11010100" WHEN "00011001", --(X"D4")
                   "10100010" WHEN "00011010", --(X"A2")
                   "10101111" WHEN "00011011", --(X"AF")
                   "10011100" WHEN "00011100", --(X"9C")
                   "10100100" WHEN "00011101", --(X"A4")
                   "01110010" WHEN "00011110", --(X"72")
                   "11000000" WHEN "00011111", --(X"C0")

                   "10111010" WHEN "11000000", --(X"BA")
                   "01111000" WHEN "11000001", --(X"78")
                   "00100101" WHEN "11000010", --(X"25")
                   "00101110" WHEN "11000011", --(X"2E")
                   "00011100" WHEN "11000100", --(X"1C")
                   "10100110" WHEN "11000101", --(X"A6")
                   "10110100" WHEN "11000110", --(X"B4")
                   "11000110" WHEN "11000111", --(X"C6")
                   "11101000" WHEN "11001000", --(X"E8")
                   "11011101" WHEN "11001001", --(X"DD")
```

```
                    "01110100" WHEN "11001010", --(X"74")
                    "00011111" WHEN "11001011", --(X"1F")
                    "01001011" WHEN "11001100", --(X"4B")
                    "10111101" WHEN "11001101", --(X"BD")
                    "10001011" WHEN "11001110", --(X"8B")
                    "10001010" WHEN "11001111", --(X"8A")

                    "XXXXXXXX" WHEN OTHERS;

END sbox_arch;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY inv_sbox IS
PORT(
   in_sbox    :    IN  std_logic_vector(7 downto 0);
   out_sbox   :    OUT std_logic_vector(7 downto 0)
   );
END inv_sbox;

-- Uma parte do código foi retirada, não prejudicando o entendimento do mesmo.

ARCHITECTURE inv_sbox_arch OF inv_sbox IS

BEGIN

        WITH in_sbox(7 downto 0) SELECT

                out_sbox(7 downto 0) <=

                "01010010" WHEN "00000000", --(X"52")
                "00001001" WHEN "00000001", --(X"09")
                "01101010" WHEN "00000010", --(X"6a")
                "11010101" WHEN "00000011", --(X"D5")
                "00110000" WHEN "00000100", --(X"30")
                "00110110" WHEN "00000101", --(X"36")
                "10100101" WHEN "00000110", --(X"A5")
                "00111000" WHEN "00000111", --(X"38")
                "10111111" WHEN "00001000", --(X"BF")
                "01000000" WHEN "00001001", --(X"40")
                "10100011" WHEN "00001010", --(X"A3")
                "10011110" WHEN "00001011", --(X"9E")
                "10000001" WHEN "00001100", --(X"81")
                "11110011" WHEN "00001101", --(X"F3")
                "11010111" WHEN "00001110", --(X"D7")
                "11111011" WHEN "00001111", --(X"FB")

                "01111100" WHEN "00010000", --(X"7C")
                "11100011" WHEN "00010001", --(X"E3")
                "00111001" WHEN "00010010", --(X"39")
                "10000010" WHEN "00010011", --(X"82")
                "10011011" WHEN "00010100", --(X"9B")
                "00101111" WHEN "00010101", --(X"2F")
                "11111111" WHEN "00010110", --(X"FF")
                "10000111" WHEN "00010111", --(X"87")
                "00110100" WHEN "00011000", --(X"34")
                "10001110" WHEN "00011001", --(X"8E")
                "01000011" WHEN "00011010", --(X"43")
                "01000100" WHEN "00011011", --(X"44")
                "11000100" WHEN "00011100", --(X"C4")
                "11011110" WHEN "00011101", --(X"DE")
                "11101001" WHEN "00011110", --(X"E9")
                "11001011" WHEN "00011111", --(X"CB")
```

```
                "00010111" WHEN "11110000", --(X"17")
                "00101011" WHEN "11110001", --(X"2B")
                "00000100" WHEN "11110010", --(X"04")
                "01111110" WHEN "11110011", --(X"7E")
                "10111010" WHEN "11110100", --(X"BA")
                "01110111" WHEN "11110101", --(X"77")
                "11010110" WHEN "11110110", --(X"D6")
                "00100110" WHEN "11110111", --(X"26")
                "11100001" WHEN "11111000", --(X"E1")
                "01101001" WHEN "11111001", --(X"69")
                "00010100" WHEN "11111010", --(X"14")
                "01100011" WHEN "11111011", --(X"63")
                "01010101" WHEN "11111100", --(X"55")
                "00100001" WHEN "11111101", --(X"21")
                "00001100" WHEN "11111110", --(X"0C")
                "01111101" WHEN "11111111", --(X"7D")

                "XXXXXXXX" WHEN OTHERS;
END inv_sbox_arch;



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY shift_rows IS
PORT(
   in_shiftrows     :   IN  std_logic_vector(127 downto 0);
   out_shiftrows    :   OUT std_logic_vector(127 downto 0)
   );
END shift_rows;

-- Uma parte do código foi retirada, não prejudicando o entendimento do mesmo.

ARCHITECTURE shift_rows_arch OF shift_rows IS

                TYPE matrix_index IS array (15 downto 0) OF std_logic_vector(7 downto 0);
                SIGNAL matrix1, matrix2 : matrix_index;

BEGIN
                matrix2(0)  <=  matrix1(0);
                matrix2(1)  <=  matrix1(5);
                matrix2(2)  <=  matrix1(10);
                matrix2(3)  <=  matrix1(15);

                matrix2(4)  <=  matrix1(4);
                matrix2(5)  <=  matrix1(9);
                matrix2(6)  <=  matrix1(14);
                matrix2(7)  <=  matrix1(3);

                matrix2(8)  <=  matrix1(8);
                matrix2(9)  <=  matrix1(13);
                matrix2(10) <=  matrix1(2);
                matrix2(11) <=  matrix1(7);

                matrix2(12) <=  matrix1(12);
                matrix2(13) <=  matrix1(1);
                matrix2(14) <=  matrix1(6);
                matrix2(15) <=  matrix1(11);

END shift_rows_arch;
```

```
ENTITY inv_shift_rows IS
        PORT(
                in_shiftrows    :   IN  std_logic_vector(127 downto 0);
                out_shiftrows   :   OUT std_logic_vector(127 downto 0)
        );
END inv_shift_rows;

-- Uma parte do código foi retirada, não prejudicando o entendimento do mesmo.

ARCHITECTURE inv_shift_rows_arch OF inv_shift_rows IS

                TYPE matrix_index IS array (15 downto 0) OF std_logic_vector(7 downto 0);
                SIGNAL matrix1, matrix2 : matrix_index;

BEGIN
                matrix1(0)  <=  matrix2(0);
                matrix1(5)  <=  matrix2(1) ;
                matrix1(10) <=  matrix2(2) ;
                matrix1(15) <=  matrix2(3) ;

                matrix1(4)  <=  matrix2(4);
                matrix1(9)  <=  matrix2(5);
                matrix1(14) <=  matrix2(6);
                matrix1(3)  <=  matrix2(7);

                matrix1(8)  <=  matrix2(8);
                matrix1(13) <=  matrix2(9);
                matrix1(2)  <=  matrix2(10);
                matrix1(7)  <=  matrix2(11);

                matrix1(12) <=  matrix2(12);
                matrix1(1)  <=  matrix2(13);
                matrix1(6)  <=  matrix2(14);
                matrix1(11) <=  matrix2(15);

END inv_shift_rows_arch;




ENTITY mix_column IS
PORT(
   in_mixcolumns    :   IN  std_logic_vector(127 downto 0);
   out_mixcolumns   :   OUT std_logic_vector(127 downto 0)
   );
END mix_column;

-- Uma parte do código foi retirada, não prejudicando o entendimento do mesmo.

ARCHITECTURE mix_column_arch OF mix_column IS

                TYPE matrix_index is array (15 downto 0) of std_logic_vector(7 downto 0);
                TYPE shift_index is array (15 downto 0) of std_logic_vector(8 downto 0);
                SIGNAL shiftby_2, shiftby_3, xored : shift_index;
                SIGNAL matrix, matrix_out, multby_2, multby_3 : matrix_index;
BEGIN
                matrix_out(0)  <= multby_2(0)  XOR multby_3(1)  XOR matrix(2)  XOR matrix(3);
                matrix_out(4)  <= multby_2(4)  XOR multby_3(5)  XOR matrix(6)  XOR matrix(7);
                matrix_out(8)  <= multby_2(8)  XOR multby_3(9)  XOR matrix(10) XOR matrix(11);
                matrix_out(12) <= multby_2(12) XOR multby_3(13) XOR matrix(14) XOR matrix(15);


                matrix_out(1)  <= matrix(0)  XOR multby_2(1)  XOR multby_3(2)  XOR matrix(3);
                matrix_out(5)  <= matrix(4)  XOR multby_2(5)  XOR multby_3(6)  XOR matrix(7);
                matrix_out(9)  <= matrix(8)  XOR multby_2(9)  XOR multby_3(10) XOR matrix(11);
                matrix_out(13) <= matrix(12) XOR multby_2(13) XOR multby_3(14) XOR matrix(15);
```

```
                matrix_out(2)  <= matrix(0)  XOR matrix(1)  XOR multby_2(2)  XOR multby_3(3);
                matrix_out(6)  <= matrix(4)  XOR matrix(5)  XOR multby_2(6)  XOR multby_3(7);
                matrix_out(10) <= matrix(8)  XOR matrix(9)  XOR multby_2(10) XOR multby_3(11);
                matrix_out(14) <= matrix(12) XOR matrix(13) XOR multby_2(14) XOR multby_3(15);

                matrix_out(3)  <= multby_3(0)  XOR matrix(1)  XOR matrix(2)  XOR multby_2(3);
                matrix_out(7)  <= multby_3(4)  XOR matrix(5)  XOR matrix(6)  XOR multby_2(7);
                matrix_out(11) <= multby_3(8)  XOR matrix(9)  XOR matrix(10) XOR multby_2(11);
                matrix_out(15) <= multby_3(12) XOR matrix(13) XOR matrix(14) XOR multby_2(15);

END mix_column_arch;



ENTITY inv_mix_column IS
PORT(
   in_mixcolumns     :   IN  std_logic_vector(127 downto 0);
   out_mixcolumns    :   OUT std_logic_vector(127 downto 0)
   );
END inv_mix_column;

-- Uma parte do código foi retirada, não prejudicando o entendimento do mesmo.

ARCHITECTURE inv_mix_column_arch OF inv_mix_column IS

                TYPE matrix_index is array (15 downto 0) of std_logic_vector(7 downto 0);
                TYPE shift_index is array (15 downto 0) of std_logic_vector(8 downto 0);
                SIGNAL matrix, matrix_out, multby_0e, multby_0b,multby_0d,multby_09 : matrix_index;
BEGIN
                matrix_out(0)  <= multby_0e(0)  XOR multby_0b(1)  XOR multby_0d(2)  XOR multby_09(3);
                matrix_out(4)  <= multby_0e(4)  XOR multby_0b(5)  XOR multby_0d(6)  XOR multby_09(7);
                matrix_out(8)  <= multby_0e(8)  XOR multby_0b(9)  XOR multby_0d(10) XOR multby_09(11);
                matrix_out(12) <= multby_0e(12) XOR multby_0b(13) XOR multby_0d(14) XOR multby_09(15);

                matrix_out(1)  <= multby_09(0)  XOR multby_0e(1)  XOR multby_0b(2)  XOR multby_0d(3);
                matrix_out(5)  <= multby_09(4)  XOR multby_0e(5)  XOR multby_0b(6)  XOR multby_0d(7);
                matrix_out(9)  <= multby_09(8)  XOR multby_0e(9)  XOR multby_0b(10) XOR multby_0d(11);
                matrix_out(13) <= multby_09(12) XOR multby_0e(13) XOR multby_0b(14) XOR multby_0d(15);

                matrix_out(2)  <= multby_0d(0)  XOR multby_09(1)  XOR multby_0e(2)  XOR multby_0b(3);
                matrix_out(6)  <= multby_0d(4)  XOR multby_09(5)  XOR multby_0e(6)  XOR multby_0b(7);
                matrix_out(10) <= multby_0d(8)  XOR multby_09(9)  XOR multby_0e(10) XOR multby_0b(11);
                matrix_out(14) <= multby_0d(12) XOR multby_09(13) XOR multby_0e(14) XOR multby_0b(15);

                matrix_out(3)  <= multby_0b(0)  XOR multby_0d(1)  XOR multby_09(2)  XOR multby_0e(3);
                matrix_out(7)  <= multby_0b(4)  XOR multby_0d(5)  XOR multby_09(6)  XOR multby_0e(7);
                matrix_out(11) <= multby_0b(8)  XOR multby_0d(9)  XOR multby_09(10) XOR multby_0e(11);
                matrix_out(15) <= multby_0b(12) XOR multby_0d(13) XOR multby_09(14) XOR multby_0e(15);

END inv_mix_column_arch;
```

- Artigo aceito no LASCAS'2011

A Highly Efficient FPGA Implementation of AES Cryptography

Otávio S. M. Gomes, Robson L. Moreno and Tales C. Pimenta
Universidade Federal de Itajubá - UNIFEI - Itajubá - Brazil

Abstract - **This article describes the core implementation of an Advanced Encryption Standard - AES in Field Programmable Gate Array - FPGA. The core was implemented in both Xilinx Spartan-3 and Xilinx Virtex-5 FPGAs. The algorithm was implemented for 128 bits word and key. The implementation was very efficient, achieving 318MHz on a Xilinx Spartan-3, representing at 50% faster than other reported works. The implementation can achieve 800MHz on a Xilinx Virtex-5. The main goal of this work was the implementation of a fast and modular AES algorithm, as it can be easily reconfigured to 128, 196 or 256 bits key, and can find a wide range of applications. Nevertheless, all the reported works used as comparison basis to our work were also implemented using 128 bits key.**

Keywords: Cryptography, AES, DES, FPGA, efficient encryption/decryption implementation.

I. INTRODUCTION

In 1997, the NIST (National Institute of Standards and Technology) released a contest to choose a new symmetric cryptograph algorithm that would be called Advanced Encryption Standard – AES to be used to protect confidential data in the USA. The algorithm should meet few requirements such copyright free, faster than the 3DES, cryptograph of 128 bit blocks using 128, 192 and 256 bit keys, possibility of hardware and software implementation, among others. In 2000, after analysis by cryptography experts, it was chosen the winner: Rijndael. The algorithm was created by the Belgians Vincent Rijmen e Joan Daemen [1][2].

The algorithm was implemented in FPGA due to its flexibility and reconfiguration capability. A reconfigurable device is very convenient for a cryptography algorithm since it allows cheap and quick alterations.

Section II provides a brief introduction of AES and its processing phases. Section III describes the chosen FPGA and the circuit implementation. Section IV compares the results of this work with others presented in the literature. Section V presents the conclusions of this work and finally Sections VI shows the authors expectations and proposals for continuing this work.

II. AES RIJNDAEL

In order to better understand the AES structure it is necessary to know the definition of state, in the algorithm. State is the matrix of bytes that is processed between the many stages, or rounds, and therefore, it will be modified in each stage. In the Rijndael algorithm, the matrix size depends on the block size being used, composed of 4 lines and Nb columns. Here, Nb is the number of bits in the block, divided by 32, since 4 bytes represent 32 bits. Since the AES algorithm uses 128 bit blocks, the state will be composed by 4 lines and 4 columns [3].

The key is grouped by the same fashion as the data block, whereas Nk is the number of columns. Nr is the number of rounds that will be run during the algorithm. The number of runs in the AES will depend on size of the key, where Nr will be 10, 12 and 14, for Nk equals to 4, 6 e 8, respectively [1].

On the encryption algorithm, there will be 4 phases: AddRoundKey, SubBytes, ShiftRows and MixColumns. Nevertheless, on the last stage, the MixColumns operation is suppressed. The decryption algorithm will use the respective inverse operations: InvAddRoundKey, InvSubBytes, InvMixColumns and InvShiftRows. As it was in the encryption phase, the InvMixColumns is suppressed on the last stage of decryption algorithm [2].

The algorithm will be explained based on its specification. The values shown in the example are presented in hexadecimal format.

A.SubBytes

Each state byte is replaced by another in the S-box (replacement Box), as indicated in Fig. 1. The replacement follows a matrix, where the first hexadecimal value corresponds to the line positioning, and the second hexadecimal value corresponds to the column positioning. The inverse operation (decryption) is called InvSubBytes, and uses an inverse S-Box.

As an example, the S-box outputs 24 for the input value A6. On the same way, the inverse S-Box outputs A6 for the input value 24.

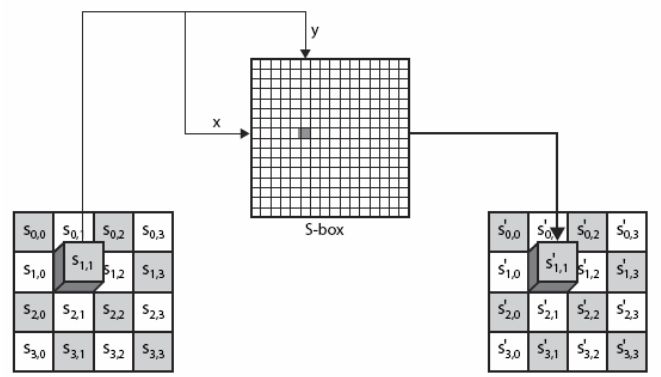

Figure 1 - SubBytes operation process.

B. ShiftRows

It consists of a left shift on the state lines, replacing therefore their byte position, as indicated in Fig. 2. Line 0 suffers 0 shifting. Line 1 is shifted by one position and line 2 undergoes do 2 shifting positions. Line 3 is shifted by 3 positions.

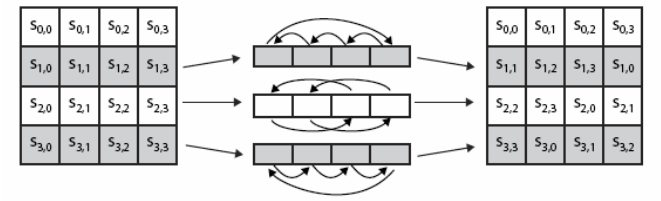

Figure 2 - ShiftRows operation process.

The decryption algorithm performs the inverse operation InvShiftRows that consists of similar shiftings as the ShiftRows, but shifted to the right.

C. MixColumns

In this operation, the state bytes are treated as polynomials of Galois Field algebra GF(28) [4]. The operation can be represented as a matrix multiplication, as indicated in Fig. 3, where S is the initial state and S´ is the final state, after the operation.

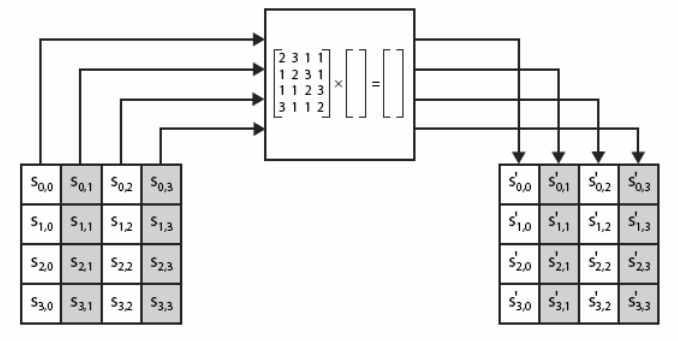

Figure 3 - MixColumns operation process.

The inverse operation, the InvMixColumns, consists of the multiplication using the inverse matrix.

In the last round, on both the encryption and decryption algorithms, the MixColumns operation is suppressed.

The C matrix (used in the encryption) and C´ matrix (used in the decryption) are:

$$C = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \qquad C' = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

D. AddRoundKey

It is an XOR operation between the state and the round key that it is generated from the main key through the Key Generation. The matrix of keys is represented by w columns or $k_{x,y}$ cells. AddRoundKey is used both in the encryption and decryption algorithms. The XOR is conducted on byte basis, as indicated in Fig 4, where the new byte $s'_{x,y}$ is given by $s_{x,y} \oplus k_{x,y}$.

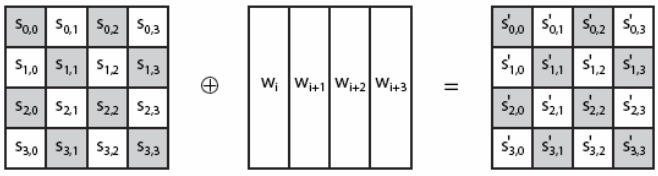

Figure 4 - AddRoundKey operation process

E. Key Expansion

The Key size defines the number of rounds in the encryption/decryption algorithm, and it also defines its expansion process. Basically, the Key Expansion operation consists of three operations, as presented in Fig. 5. The first operation, RotWord, makes a one byte circular shifting on the word. The second operation, SubWord replaces each byte of the input word according to the S-Box. The third operation consists of XOR operations, as indicated in Fig.5.

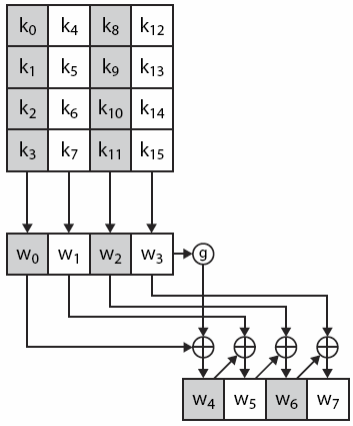

Figure 5 - KeyExpansion operation process

III. IMPLEMENTATION

The AES hardware was implemented in three modules: the encryption, the decryption and the key expansion module. All the modules were independently tested and characterized, and therefore they can be used in any combination, according to the application.

In order to conduct tests on all blocks, it was assembled a 128 bits encryption - decryption AES set in a Xilinx Spartan-3 FPGA. The test results are presented in Section IV.

After the tests on the Xilinx Spartan-3 FPGA, the hardware was also tested on a Xilinx Virtex-5 FPGA.

The VHDL description implemented on both FPGAs is exactly the same, and no change was made in the VHDL description to fit any of the FPGAs. Another important information is that the code is totally portable, the code can be used in any FPGA family because was developed using the VHDL patterns.

The hardware implemented is illustrated in Fig. 6. It is composed of two 128 bit inputs that receive the key and the initial word to be encrypted. The signal in_aes_mode defines an encryption or decryption operation. The load inputs are used to indicate that the data at the input is valid and can be loaded. The signal out_busy signals if the circuit is busy processing a word or is available for a new word.

Figure 6 - Inputs and Outputs of developed AES128.

On a Xilinx Virtex-5 FPGA, the initial encryption load takes 20ns and the decryption loads takes 30ns. The decryption loading process takes 10 cycles longer than the encryption since it requires loading and process the entire key in order to start the decryption process.

On the encryption process, at each key expansion it is possible to encrypt the word at next cycle.

On a Xilinx Virtex-5 FPGA, the cryptograph of each word runs at approximately 60MHz, since the hardware takes 12 clock cycles to process it. The FPGA operates at approximately 800MHz, as it can be seen from the listing shown in Fig 7.

The implementation of encryption/decryption algorithms using pipelining is left as a suggestion to improve this work. With this suggestion we believe that the hardware efficiency will be improved achieving high frequencies. Using a pipeline with 10 levels the frequency of cryptography of each word will be near to 800MHz.

```
FPGA device: Spartan-3 XC3S4000

Speed Grade: -5
Minimum period: 3.140ns
Maximum Frequency: 318.492MHz
Minimum input arrival time before clk: 9.834ns
Maximum output req time after clk: 6.216ns
```

```
FPGA device: Virtex-5 XC5VFX70T

Speed Grade: -1
Minimum period: 1.116ns
Maximum Frequency: 896.057 MHz
Minimum input arrival time before clk: 2.300ns
Maximum output req time after clk: 3.524ns
```

Figure 7 – Summary of FPGA speed achieved.

IV. TESTS AND HARDWARE VERIFICATION

The hardware was tested and the functions were verified according the patterns and test vectors of AES documentation design, available in [1][2][3]. All the results were obtained according the benchmarks.

IV. RESULTS COMPARISON

It was chosen the Xilinx Spartan-3 (XC3S4000) to conduct performance comparison of our work with others [5][6][7][8][9][10][11], since it was used to implement many of them. Table I summarizes the performance comparison.

As can be observed from the table, our work is at least 50% faster that the fastest circuit reported. This result is related to the architecture of the hardware developed.

TABLE I
COMPARISON WITH OTHER FPGA IMPLEMENTATIONS

Implementation	*Platform Device*	*Data Path*	*Frequency (MHz)*
C. Chien [5]	Xilinx Virtex-II (XC2V1000)	128	75
I. Aigredo-Badillo [6]	Xilinx Virtex-II (XC2V1000)	128	96.42
J. Zambreno [7]	Xilinx Virtex-II (XC2V4000)	128	110.16
E. J. Swankoski [8]	Xilinx Virtex-II Pro (XC2VP50)	128	145.05
E. Lopez-Trejo [9]	Xilinx Spartan-3 (XC3S4000)	128	100.08
A. Aziz & N. Ikram [10]	Xilinx Spartan-3 (XC3S50)	128	165
Dur-e-Shahwar, Zaka, Qurat-Ul-Ain and Aziz [11]	Xilinx Spartan-3 (XC3S4000)	128	206.28
Our Design	**Xilinx Spartan-3 (XC3S4000)**	**128**	**318.49**
	Xilinx Virtex-5 (XC5VFX70T)	**128**	**896.05**

Was used the same environment related in others works to make the correct comparison. Nevertheless, the Xilinx Virtex-5 offers an even higher speed.

V. CONCLUSIONS

This article presented a fast and efficient AES cryptography hardware structure that can have many applications. The circuit implementation is very efficient and can be customized to a wide range of applications. It represents an improvement to support new applications.

VI. FUTURE WORK

The Microelectronics Group at Universidade Federal de Itajuba intends to use this work as part of larger projects, including smart metering in power systems and cryptography interface in data communications [12].

ACKNOLEDGEMENTS

The authors would like to thank the Microelectronics Group at Universidade Federal de Itajubá. The authors acknowledge CAPES, CNPq and FAPEMIG for their financial support.

REFERENCES

[1] **FIPS-197**, "Federal Information Processing Standards Publication FIPS-197, Advanced Encryption Standard (AES)", **http://csrc.nist.gov/publications!fips!fips 1 97/1Ips-197.pdf, October 1999.**

[2] **Daemen, J. and Rijmen, V. (2002)**. The design of Rijndael: AES — The Advanced Encryption Standard. **Springer-Verlag.**

[3] **Daemen, J. and Rijmen, V.** A Specification for The AES Algorithm. **NIST (National Institute of Standards and Technology). http://csrc.nist.gov/archive/aes/rijndael/wsdindex.html , October 2010.**

[4] **Klima, R. E., Sigmon, N., and Stitzinger, E. (2000)**. Applications of abstract algebra with Maple. **CRC Press, Boca Raton, FL.**

[5] **C. Chien, D. Chien, C. Chien, I. Verbauwhede and F. Chang,** "A hardware implementation in FPGA of the Rijndael algorithm", **The 2002 45th Midwest Symp. Circuits and Systems (MWSCAS-2002), Vol. 1,4--.7 August 2002, pp. 507-509.**

[6] **I. Algredo-Badillo, C. Feregrino-Uribe and R. Cumlido-Parra,** "Design and implementation of an FPGA-based 1.452 Gbps non-pipelined AES architecture', **The 2006 Int. Conf Computational Science and Its Applications (ICCSA 2006), Lecture Notes in Computer Science, Vol. 3982 (Springer-Verlag, 2006), pp. 446--455.**

[7] **J. Zambreno, D. Nguyen and A. Choudhary,** "Exploring area/delay tradeoffs in an AES FPGA implementation", **Proc. Int. Colif, FieldProgrammable Logic and Its Applications (FPL), Lecture Notes in Computer Science, Vol. 3203 (Springer-Verlag 2004), pp. 575-585.**

[8] **E. J. Swankoski, V. Narayanan, M. Kandemir and M. J. Irwin,** "A parallel architecture for secure FPGA symmetric encryption", **18th Int. Parallel and Distributed Processing Symp. (IPDPS'04) - Workshop, Santa Fe, New Mexico, 26-.30 April 2004, p. 123.**

[9] **E. Lopez-Trejo, F. Rodriguez-Henriquez and A. Diaz-Perez,** "An efficient FPGA implementation of CCM using AES", **The 8th Int. Con! Information Security and Cryptology (ICJSC'05). Lecture Notes in Computer Science (Springer 2005), pp. 208-215.**

[10] **Arshad Aziz and Nassar Ikram,** "Memory efficient implementation of AES S-boxes on FPGA", **Journal of Circuits, Systems, and Computers, Vol. 16, No.4 (2007) 603--611.**

[11] **Dur-e-Shahwar Kundi, Saleha Zaka, Qurat-Ul-Ain and Arshad Aziz** "A Compact AES Encryption Core on Xilinx FPGA" **(2009) of 2nd IEEE International Conference on Computer, Control & Communication (IEEE IC4-2009) Karachi, Pakistan Vol:1 pp:1-4**

[12] **N. SKLAVOS, X. ZHANG**, "Wireless Security & Cryptography: Specifications and Implementations", **CRC-Press, A Taylor and Francis Group. ISBN: 084938771X, 2007.**