FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DESIRÉE BUENO DO NASCIMENTO

PROPOSTA DE UMA ARQUITETURA DE REFERÊNCIA PARA O ALGORITMO KECCAK

MARÍLIA
2011

DESIRÉE BUENO DO NASCIMENTO

PROPOSTA DE UMA ARQUITETURA DE REFERÊNCIA PARA O ALGORITMO KECCAK

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides de Marília (UNIVEM), mantido pela Fundação Eurípedes Soares da Rocha, para obtenção do Título de Bacharel em Ciência da Computação.

Orientador:

Prof. Dr. Fábio Dacêncio Pereira

MARÍLIA
2011

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Desirée Bueno do Nascimento

PROJETO, DESENVOLVIMENTOE ANÁLISE DA ALGORITMO KECCAK EM FPGA

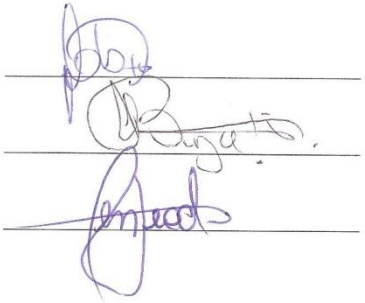
Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 8,5 (Oito e meio)

Orientador: Fábio Dacêncio Pereira

1°. Examinador: Ildeberto de Gênova Bugatti

2°. Examinador: César Giacomini Penteado



Marília, 28 de novembro de 2011.

AGRADECIMENTOS

A minha família pelo apoio nos momentos difíceis.

Aos colegas do curso que me ajudaram e contribuíram de alguma forma para esta conquista.

Ao professor Fábio D. Pereira por ter aceitado me orientar e pelo livro de hardware presenteado.

Ao professor Ildeberto de G. Bugatti, por colaborar diretamente na realização deste trabalho, dispondo do seu tempo e paciência.

Ao amigo Goffredo pelo apoio, incentivo e conselhos, nos momentos de necessidade e dificuldades.

"O passado é história, o futuro é mistério, o presente é uma dádiva e por isso se chama presente."

Deepak Chopra.

RESUMO

Com a finalidade de se garantir integridade, confidencialidade e consistência das informações, usa-se como um dos métodos de segurança as funções de *hash*, *National Institute of Standards and Technology* (NIST) está promovendo um concurso para a criação de um novo algoritmo de *hash*, o SHA - 3. Este trabalho propõe o estudo e implementação de um dos algoritmos finalistas SHA-3, usando a tecnologia de circuitos programáveis (FPGA) e linguagem de descrição de hardware (VHDL), uma vez que esse tipo de tecnologia é um grande atrativo para soluções que exigem segurança.

Palavras chaves: Funções de *hash*, criptografia, SHA-3, FPGA, VHDL.

ABSTRACT

In order to ensure integrity, confidentiality and consistency of information, is used as a security methods of hash functions, National Institute of Standards and Technology (NIST) is holding a contest to create a new hash algorithm the SHA - 3. This paper proposes the study and implementation of one of the finalist algorithms SHA-3, using the technology of programmable circuits (FPGA) and hardware description language (VHDL), since this type of technology is a major attraction for solutions that require security.

Keywords: *hash functions, encryption, SHA-3, FPGA, VHDL.*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

A criptografia é o estudo de técnicas matemáticas relacionadas aos aspectos de segurança da informação, tais como a confidencialidade, integridade de dados, autenticação de uma entidade, e não repúdio, (MENEZES; OORSCHOT; VANSTONE;1996, p.4).

Sendo muito usada nas Grandes Guerras, não se sabe quem foi seu criador, mas ramificou-se em várias vertentes e tornou-se uma das principais formas de assegurar a integridade das informações nos dias atuais.

Assim como surge o interesse na confiabilidade das informações, existe também o interesse em decifrar sem autorização utilizando métodos matemáticos ou força bruta.

Um dos meios de assegurar a confiabilidade e integridade das informações é a utilização das funções de *hash*, que recebem um dado de tamanho arbitrário e retornam uma cadeia de bits usada para garantir a integridade de dados. Os *hashes* estão contidos na área de codificação dentro da criptografia.

O *hash* é uma das ferramentas utilizadas para prover segurança, mas não à única, associada a outras ferramentas e técnicas dificulta o acesso ou a interpretação não autorizado dos dados criptografados.

As funções de *hash* podem apresentar vulnerabilidades, este projeto sugere a implementação em linguagem de hardware de um dos algoritmos que são concorrentes na nova função de *hash* SHA-3.

A intenção não é prever o algoritmo escolhido como nova função SHA-3, mas sim prover uma implementação de um dos finalistas que contribuindo para as pesquisas e estudos na área de criptografia.

Essa implementação será feita utilizando-se linguagem de hardware, juntamente com o uso de circuito programáveis do tipo FPGA.

PROBLEMÁTICA E JUSTIFICATIVA

Com o advento e popularização da Internet, como um dos principais meios de comunicação existente, há a crescente necessidade de manter a segurança das informações que trafegam na rede mundial de computadores.

Uma das formas de manter a integridade dessas informações é utilizar a criptografia, embora existam muitos métodos criptográficos o método abordado no projeto é a utilização das funções de *hash*.

Muitos algoritmos de *hash* criados há alguns anos, são utilizados até hoje como o MD4, MD5 e SHA-1, mas todos apresentaram notificações de vulnerabilidades, tais como *hashes* iguais para dados diferentes.

Para garantir que a integridade e segurança das informações transmitidas e armazenadas sejam mantidas o NIST (*National Institute of Standards and Technology*) está promovendo uma chamada pública para o desenvolvimento de um novo algoritmo de criptografia utilizando *hash.*

Este novo algoritmo será conhecido como SHA-3 e substituirá os algoritmos de criptografia utilizados atualmente. Esta chamada encontra-se na fase final e conta com cinco finalistas.

Neste contexto, esse projeto se propõe a selecionar um dos algoritmos finalistas da chamada pública do SHA-3, implementá-lo usando linguagem de descrição de hardware, visando a obtenção de dados sobre área e desempenho do algoritmo.

OBJETIVOS

Realizar uma revisão sistemática de implementações do algoritmo Keccak em hardware. Analisar e comparar implementações para servir como base de decisões de projetos de desenvolvimento e otimizações.

Propor uma arquitetura de referência para o algoritmo Keccak em hardware que privilegia a relação área, desempenho e consumo em sua síntese.

O projeto contribuirá para geração de dados de área e desempenho do algoritmo Keccak e poderá ser utilizado como referencia para novos trabalhos ou comparação com a tecnologia utilizada neste projeto.

METODOLOGIA

A metodologia adota as quatro seguintes etapas de desenvolvimento:

- Etapa um, abrange a escolha da função de *hash*, envolve o estudo dos algoritmos finalistas e suas comparações, a escolha é embasada no algoritmo que se adapte melhor a tecnologia neste projeto.
- Etapa dois, abrange a implementação do algoritmo escolhido, usando linguagem VHDL, juntamente com o estudo mais aprofundado da linguagem.
- Etapa três, abrange testes e análise do projeto, consiste na síntese e testes sobre o algoritmo implementado em VHDL e FPGA.
- Etapa quatro propõe uma arquitetura de referência para o algoritmo selecionado.

Ainda que não como uma etapa a metodologia a sugere o acompanhamento das implementações do algoritmo escolhido em outras tecnologias como base de estudo e parâmetros para comparação.

ORGANIZAÇÃO DO TRABALHO DE CONCLUSÃO

Este trabalho é organizado da seguinte forma:

O capítulo 1 contém um breve histórico sobre segurança da informação, criptografia e suas vertentes, métodos criptográficos usados ao longo dos tempos, mostrando os diferentes tipos de chaves criptográficas.

A motivação da competição para o novo algoritmo de *hash*, mostra as etapas da competição e os algoritmos finalistas, está contido no capítulo 2.

O capítulo 3 aborda o algoritmo Keccak com uma sucinta discussão sobre sua escolha e suas funções, descreve o pseudocódigo das suas funcionalidades e implementações em hardware.

A proposta de arquitetura para o algoritmo Keccak, apresentada no capitulo 4 descreve o algoritmo na linguagem VHDL e validação através de sua simulação.

O capitulo 5 mostra a metodologia de implementação utilizada e os resultados obtidos em FPGA com o código implementado, por fim, o capítulo 6 descreve as conclusões obtidas com a realização desse projeto.

1. SEGURANÇA DA INFORMAÇÃO

Não é de hoje que existe a preocupação com a segurança das informações enviadas ou recebidas, mas com a globalização e evolução das tecnologias pelos quais possibilitaram à redução significativa do tempo, as medidas para a proteção de dados tornaram-se essenciais.

A informação tonou-se objeto de valor e assim juntamente com os meios de comunicação existentes tornou-se muito mais fácil a troca dos conhecimentos adquiridos.

No passado, as questões referentes à segurança da informação eram muito mais simples, as preocupações limitavam-se apenas ao acesso físico, os documentos poderiam ser trancados ou escondidos, hoje com o advento da Internet, a portabilidade e o fácil acesso aos meios de comunicação o problema de se assegurar os dados torna-se maior.

Segundo Menezes, Oorschot e Vanstone (1996, p.2):

> Segurança da informação se manifesta de muitas maneiras de acordo com a situação e exigência. Independentemente de quem esteja envolvido, de uma forma ou de outra, todas as partes de uma transação deve ter confiança de que certos objetivos associados a segurança da informação foram cumpridos.

Embora precisos, não há garantias que os objetivos necessários à segurança da informação foram cumpridos, conforme mostrado na Tabela 1, são vários os objetivos e assegurar todos eles torna-se uma tarefa trabalhosa e complicada.

Tabela 1 - Objetivos da segurança da informação.

Privacidade ou confidencialidade	Manter em segredo as informações, permitindo acesso apenas às pessoas autorizadas.
Integridade dos dados	Garantir que a informação não tenha sido alterada por pessoas não autorizadas ou por meios desconhecidos.
Autenticação de entidade ou identificação	Comprovar a identidade de uma entidade (por exemplo: uma pessoa, um computador, um cartão de crédito, etc).
Autenticação de mensagens	Legitimando a fonte de informação.
Assinatura	Um meio para relacionar informações a uma entidade.
Autorização	Permitir a outra entidade ter acesso ou fazer alguma alteração.

Validação	Um meio para proporcionar oportunidade de autorização de utilização ou manipulação de informações ou recursos.
Controle de acesso	Restringindo o acesso a recursos para entidades não privilegiadas.
Certificado	Endosso da informação por uma entidade confiável.
Registro de tempo	Registar o tempo de criação ou da existência da informação.
Veracidade das informações	Verificar a criação ou a existência de informação por uma entidade que não seja o criador.
Recebimento	Reconhecimento de que a informação foi recebida.
Confirmação	Reconhecimento de que os serviços foram prestados.
Propriedade	Um meio para fornecer uma entidade com o direito legal de uso ou transferência de um recurso para os outros.
Anonimato	Omitir a identidade de uma entidade envolvida em algum processo.
Não repúdio	Impedir que uma entidade a negação de compromissos anteriores ou ações.
Revogação	Retração de certificação ou autorização.

Fonte: MENEZES; OORSCHOT; VANSTONE;1996,p. 3, tradução nossa.

Um dos métodos usados para auxiliar na segurança da informação, no qual podem ser utilizados e aplicados vários dos objetivos listados na Tabela 1, sendo empregado atualmente em larga escala é o recurso da criptografia.

1.1. BREVE HISTÓRICO

"Cripto" vem do grego *kryptós* e significa oculto, envolto, escondido. Também do grego, *graphos* significa escrever, *logos* significa estudo, ciência e *analysis* significa decomposição, (TKOTZ, 2005). A criptografia converte dados legíveis em algo sem sentido, com a capacidade de recuperar os dados originais a partir desses dados sem sentido, (BURNETT; PAINE, 2002, p.11).

Muito difundida e usada durante as Grandes Guerras, a criptografia não é algo novo, historicamente ela vem sendo usada desde o tempo das antigas civilizações, como os egípcios e romanos. Estima-se que com o surgimento da escrita e das formas de comunicação também houve a necessidade de se manter segredo sobre alguma informação, nasce nesse momento à criptografia e a criptoanálise para desvendá-los.

Restrita pela tecnologia e necessidade de cada época, a criptologia cresceu conforme sanava as necessidades. Com o passar do tempo se aliando com outras ciências como a matemática, os avanços e recursos tecnológicos, produzindo outras vertentes, como mostrado na Figura 1.

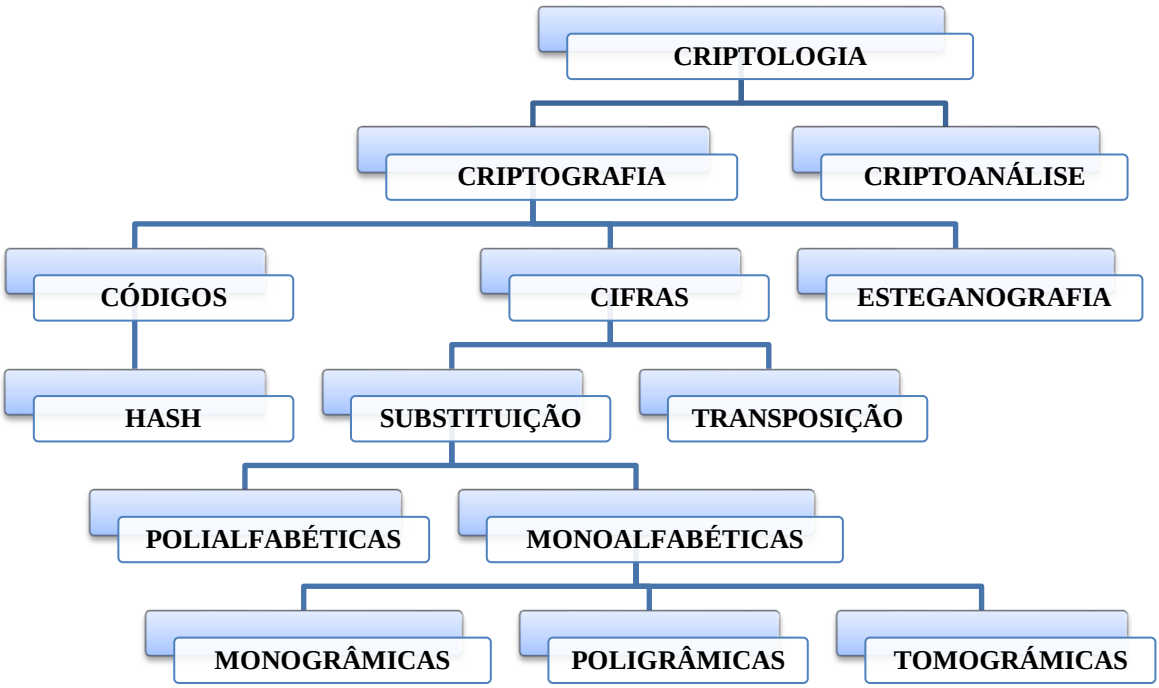


Figura 1 – Vertentes da Criptologia.

Fonte: Modificado de TKOTZ, 2005.

Datando cerca de 1900 a.C, os primeiros relatos sobre criptografia encontrados são das antigas civilizações, os Egípcios cujos meios de criptografia utilizados eram a esteganografia e substituição simples, possuem através dos hieróglifos as primeiras informações criptografadas documentas.

Uma das cifras mais conhecidas e usadas até hoje é o Código de César (também conhecido como Cifra de César), pelo qual as letras são trocadas por letras posicionadas três

posições à frente, como representado na Tabela 2. Alternativamente, César reforçava sua cifragem substituindo letras latinas por letras gregas, (TKOTZ, 2005).

Tabela 2 – Código de César

↓ ALFABETO NORMAL ↓																								
A	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***	***J***	***K***	***L***	***M***	***N***	***O***	***P***	***Q***	***R***	***S***	***T***	***U***	***V***	***W***	***X***	***Y***	***Z***
D	***E***	***F***	***G***	***H***	***I***	***J***	***K***	***L***	***M***	***N***	***O***	***P***	***Q***	***R***	***S***	***T***	***U***	***V***	***W***	***X***	***Y***	***Z***	***A***	***B***	***C***
↑ ALFABETO CODIFICADO ↑																								

Fonte: (Próprio autor, 2011).

Várias formas de criptografia foram desenvolvidas através dos tempos, como visto desde antes da era Cristã até os dias atuais, a criptografia foi estudada, modificada e aperfeiçoada.

A criptografia vem aliada a um conjunto de métodos e técnicas ajudar a segurança da informação a alcançar seus objetivos, pelos quais tenta incorporar a ideia da associação entre teoria e prática focando - se principalmente em quatro aspectos:

- Confidencialidade: Manter as informações oclusas para pessoas não autorizadas. Compreendendo desde meios físicos a algoritmos matemáticos para tornar os dados ilegíveis, (MENEZES; OORSCHOT; VANSTONE, 1996).
- Integridade: Visa tratar a alteração não autorizada dos dados. Tendo a capacidade de detectar a manipulação dos dados através de inserção, substituição e exclusão, (MENEZES; OORSCHOT; VANSTONE, 1996).
- Autenticação: Está relacionada a garantir a identificação das entidades envolvidas assim como as informações submetidas.
- Não repúdio: Uma entidade não pode negar suas ações, assim como quaisquer informações por ela dispostas.

A criptografia não é a única ferramenta para assegurar a segurança dos dados, e não resolverá todos os problemas de segurança. É um instrumento entre vários outros, (BURNETT; PAINE, 2002).

1.2. TIPOS DE ALGORITMOS DE CRIPTOGRAFIA

Os principais tipos de criptografia que foram utilizados ao longo dos anos são a codificação e a cifragem, a codificação trabalhando com chaves e a cifragem usando métodos de substituição e transposição. A codificação tem se destacado por mostrar maior segurança, rapidez na administração de suas chaves, (MORENO; PEREIRA; CHIRAMONTE, 2005).

Na Figura 1 podemos verificar dentro da criptologia, na área de criptografia no contexto de codificação, encontramos as chaves simétricas, assimétricas e as funções de *hash* (cadeia de bits), sendo a ultima foco deste trabalho de conclusão de curso.

1.2.1 CHAVE SIMÉTRICA

O algoritmo de chave simétrica (também conhecido como algoritmo de chave privada) usa somente uma única chave para criptografar e descriptografar a mesma mensagem, como ilustrado na Figura 2. Por consequente o emissor e o receptor devem ter o conhecimento da chave utilizada e o meio de envio dessa mesma chave deve ser assegurado. Uma vez que uma pessoa não autorizada tem acesso a essa chave ela torna-se capaz de visualizar, interpretar e modificar o conteúdo cifrado.

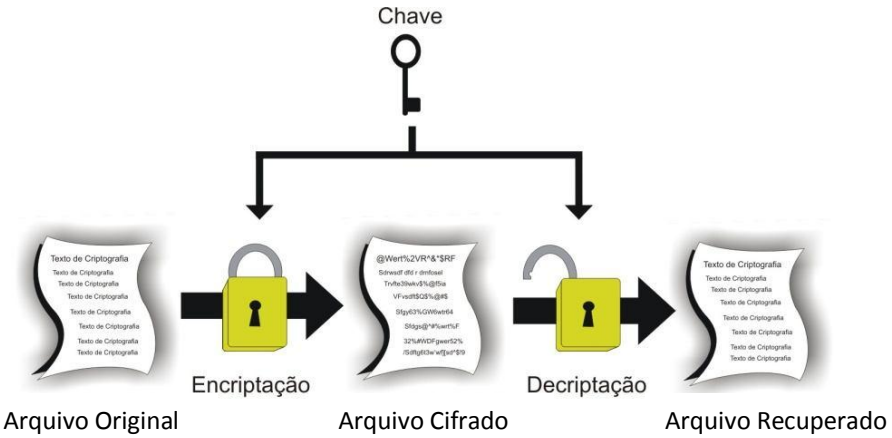


Figura 2 – Criptografia com chave simétrica.

Fonte: Modificado de Filho, 2008.

Os algoritmos de chave simétrica podem ser dados pelas seguintes funções (MENEZES; OORSCHOT; VANSTONE, 1996):

- Cifragem: $E_K(M) = C$
- Decifragem: $D_K(C) = M$

Os algoritmos simétricos ainda podem ser subdivididos em dois tipos, os algoritmos de cifras de fluxo e os algoritmos de cifras de bloco. As cifras de fluxo encriptam um texto claro bit a bit, ao passo que as cifras de bloco usam conjuntos com um número fixo de bits (geralmente 64 bits nas cifras modernas) como unidades de cifragem, (TKOTZ,2005).

Um dos algoritmos mais famosos da cifra de blocos é o DES (*Data Encryption Standard*), criado na década 1970, embora já tenha sido quebrado, em sua época ele representou um grande avanço na área da criptografia.

Já os algoritmos de cifra de fluxo, embora ainda tenham suas aplicações, por tratar as informações bit a bit, consomem grande poder computacional tanto para implementações em software como em hardware. Como exemplo pode-se citar o algoritmo RC4 (*Rivest Ciphers 4*) criado na década de 1980, teve uma grande popularidade na época tornando-se muito utilizado.

1.2.2 CHAVE ASSIMÉTRICA

O algoritmo de chave assimétrica (também conhecido como algoritmo de chave pública) usa dois tipos de chaves, uma para criptografar (chave publica) e outra para descriptografar (chave privada) a mesma mensagem, como mostrado na Figura 3. O funcionamento das chaves se dá no seguinte modo, qualquer pessoa coma chave pública pode encriptar uma mensagem, mas não decifrá-lo. Apenas a pessoa com a chave privada pode decifrar a mensagem, (SCHNEIER, 1996, p.15).

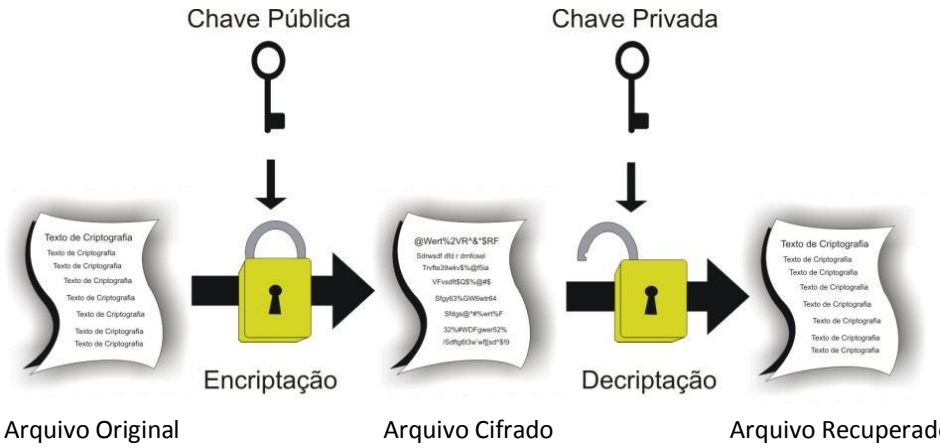


Figura 3 – Criptografia com chave assimétrica.

Fonte: Modificado de Filho, 2008.

Embora sejam usadas chaves diferentes, as funções do algoritmo de criptografia são dadas respectivamente por:

- Chave pública: $E_K(M) = C$
- Chave privada: $D_K(C) = M$

Um dos algoritmos de chave assimétrica muito utilizado na década de 1970 e ainda utilizado é o RSA (*Rivest, Shamirand Adleman*), onde suas chaves trabalhavam com fatoração de números primos, requisitando muito potencial de processamento caso o número fatorado era de grande valor.

1.2.3 FUNÇÕES DE *HASH*

As funções *hash* são funções que recebem dados de comprimento arbitrário, comprimem estes dados e devolvem um número fixo de bits, o resultado *hash*, (TKOTZ,2005). Ou seja, o *hash* é uma cadeia de bits usada para assegurar a integridade de dados, como mostra a Figura 4. Sendo que uma vez gerado o *hash*, os dados originais não poderão ser recuperados através dele.

Figura 4 – *Hash*.

Fonte: (Próprio Autor,2011).

As funções *hash* (também conhecidos como digestos ou resumos de mensagens), de forma geral são definidas pelas seguintes propriedades distintas, (MENEZES; OORSCHOT; VANSTONE, 1996):

- Compressão: através de uma função h, e uma entrada x de tamanho arbitrário, para uma saída contida em h(x), a saída possui um tamanho n fixo de bits.
- Facilidade computacional: dado um h e uma entrada x cujo cálculo de h(x) seja de fácil calculo computacional.
- Resistencia de pré - imagem: é inviável computacionalmente encontrar x' tal que h(x') = y
- Resistencia da segunda pré – imagem: é inviável computacionalmente encontrar um segundo dado de entrada que resulte no mesmo resultado de saída, $x \neq x'$ tal que h(x) = h(x').
- Resistencia a colisões: é inviável computacionalmente encontrar duas entradas (x ou x'), que resultem no mesmo *hash*, ou seja, h(x) = h(x').

Segundo Tkotz (2005), um exemplo de aplicação do *hash*, comum e pouco conhecida entre a maioria das pessoas é o seu uso em aplicações financeiras. Ele é usado de forma complementar a outras medidas de segurança, uma vez que é muito arriscado para uma instituição financeira guardar as senhas de todos os usuários, eles simplesmente guardam o *hash* gerado á partir das senhas.

Esses *hash* são armazenados e quando um usuário acessa um dos terminais com a sua senha, é gerado um novo *hash* dessa senha e comparado ao *hash* armazenado no banco de dados da instituição, se os *hashes* forem iguais o usuário tem acesso a todas as funcionalidades.

Conforme Moreno, Pereira e Chiaramonte (2005), alguns algoritmos de *hash* são usados para fins militares, além da forma comercial privativa, há os mais conhecidos no meio

acadêmico e comercial, como os algoritmos MD4 (*Message Digest 4*), MD5 (*Message Digest 5*) e o SHA-1(*Secure Hash Algorithm 1*).

O MD4 é um algoritmo criado em 1990, por Ron Rivest ele usa uma entrada de tamanho arbitrário que resulta em uma saída de 128 bits, já o MD5 desenvolvido também por Ron Rivest em 1992 é uma versão mais segura do MD4 possuindo as mesmas características de entrada e saída de dados.

O SHA-1 é um algoritmo de *hash* usado desde 1995, o arquivo de entrada deve ser menor que 2^{64} e o *hash* gerado possui 160 bits, o SHA-1 foi desenvolvido usando como base o algoritmo SHA (*Secure Hash Algorithm*), com várias melhorias.

O SHA-2 (*Secure Hash Algorithm 2*) é o algoritmo de *hash* mais atual e seguro, não possui nenhuma notificação de vulnerabilidade, por ser da mesmo família que o SHA-1 a estrutura do seu algoritmo é semelhante ao SHA-1.

2. SHA-3 CANDIDATOS

Com essa maior facilidade de acesso e troca de informação proporcionada pela Internet, torna-se necessário manter a segurança dos dados enviados e recebidos e como facilitadores para obter essa segurança são largamente utilizados os *hashes*. Muitos já foram usados como padrão de segurança, como o MD4, MD5, SHA-1 entre outros.

Com o uso desses *hashes*, ocorrem ataques a alguns desses algoritmos que quebraram o principio de sua segurança, como o caso de *hashes* iguais para arquivos diferentes, segundo Wang, Feng, Lai, e Yu, (2004) ataques ao algoritmo MD5 foram realizados com sucesso, gerando grande preocupação, pois todos ainda são largamente usados..

2.2 MOTIVAÇÃO DO EDITAL DE CONVOCAÇÃO DO SHA-3

Assim como relatados, vários algoritmos de *hash* foram quebrados inclusive no SHA-1 foram constatados falhas de segurança, atualmente o algoritmo SHA-2 continua inquebrável, mas por possuir semelhanças com o algoritmo SHA-1 os especialistas estão preocupados com a possível invalidação da sua integridade.

Para garantir que a integridade e segurança das informações enviadas sejam mantidas o NIST (*National Institute of Standards and Technology*) está promovendo uma chamada pública para o desenvolvimento de um novo algoritmo de criptografia utilizando *hash*. Este algoritmo será conhecido como SHA-3 e substituirá os algoritmos de criptografia utilizados (NIST, 2005).

2.3 TIMELINE SHA-3

Desde que foram constatadas falhas no algoritmo SHA-1 em 2005, houve a preocupação com os demais algoritmos da família SHA. Essa preocupação resultou na

decisão de se criar um novo algoritmo de *hash*, oficializando-se durante o *Second Cryptographic Hash Workshop* (Segundo Seminário de Criptografia de *Hash*) em 2006.

As datas preliminares segundo o NIST (2006) ficaram dispostas de forma geral do seguinte modo:

2006:

- Avaliação dos requisitos mínimos de aceitação, submissão e avaliação das funções *hash* a serem desenvolvidas.

2007:

- Finalização e publicação dos requisitos mínimos de aceitação, submissão e avaliação das funções *hash*.

2008:

- Publicação do prazo de submissão das funções de *hash*.

2009:

- Revisão dos projetos submetidos e seleção dos candidatos que atenderam os requisitos mínimos exigidos.
- Realização da Primeira Conferencia de Candidatos das Funções *Hash*, onde foram divulgados os projetos aceitos na primeira fase e exposição das funções.

2010:

- Realização da Segunda Conferencia de Candidatos das Funções *Hash*, onde foram discutidos a análise dos projetos aceitos na primeira fase, sendo que o candidatos poderiam apontar melhorias para seus algoritmos.
- Seleção dos algoritmos finalistas e publicação dos mesmos.
- Prazo final para qualquer alteração dos algoritmos finalistas.

2011:

- Período final de análise pública do algoritmo.

2012:

- Realização da Conferencia Final dos Candidatos das Funções *Hash*.
- Seleção do algoritmo vencedor.
- Publicação da nova função *hash*.

2.4 FINALISTAS

Entre muitos candidatos os algoritmos finalistas foram decididos depois de três rodadas, no qual a primeira rodada iniciou-se no final do ano de 2008. Durante o período de inscrições foram cadastrados 64 projetos, dos quais 51 atendiam os requisitos mínimos de aprovação, (NIST, 2008).

Durante a segunda rodada, apenas 14 dos 51 selecionados na primeira fase passaram nos requisitos mínimos de aprovação. Agora apenas 5 algoritmos competidores, conforme mostrado na Tabela 3 restaram e dentre deles apenas um será escolhido e anunciado em 2012 para a nova função *hash*.

Tabela 3 – Algoritmos Finalistas

Algoritmo	Principal Autor
Blake	Aumasson
Grostl	Knudsen
JH	Hongjun Wu
Keccak	Joan Daemen
Skein	Bruce Schneier

Fonte: NIST, 2009.

Entre os critérios de seleção mais importantes, estão a performance hardware e software em diferentes tecnologias, assim como sua robustez matemática da geração do *hash*, análise reportadas e constatadas de falhas e erros durante as fases do concurso.

3. ALGORITMO KECCAK

Além das indicações por pessoas influentes da área (Paulo Barreto, USP), a escolha desse algoritmo para realização do projeto foi embasada levando-se em consideração a tecnologia adotada que se difere na implementação das demais tecnologias, assim como o tempo de realização. Outro fator considerável foi o ato do algoritmo em questão ser adotado pelo grupo de pesquisa do COMPSI (*Computingand Information Systems Research Lab*) do UNIVEM.

3.1 INTRODUÇÃO AO KECCAK

O algoritmo Keccak, é um algoritmo voltado à geração de *hash*, um dos finalistas do NIST. Sua estrutura básica trabalha com permutações, suas larguras de permutação são dadas por $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ dada pela própria forma da concepção do algoritmo fundamentado em funções matemáticas. As interações consistem de uma série de permutações que ocorrerão conforme a escolha de um *b*. Conforme o b escolhido, quanto maior seu valor, maior o número de permutações necessárias.

Denominada como *Keccak-f[b]*, as operações consistem em uma sequencia de passos, onde os estados são organizados em uma matriz de dimensões 5x5, e suas posições tem o comprimento definido por $w \in \{1, 2, 4, 8, 16, 32, 64\}$, onde $b = 25 * w$. O número de rodadas n_r depende da largura de permutação, e é dada por $n_r = 12 + 2^{\ell}$, (BERTONI,et al, 2010).

As operações são efetuadas em cinco passos, realizando as permutações dos bits dentro dos bytes contido na matriz 5x5, além das operações lógicas XOR, NOT e AND. Junto com essas operações o algoritmo trabalha com outros dois parâmetros, que determinam o *bitrate* (taxa de transferência de bits), e a capacidade. Todas as operações e passos aplicados resultarão no *hash* de saída.

3.2 ALGORITMO E PSEUDOCÓDIGO DO KECCAK

Como citado anteriormente o algoritmo trabalha com cinco passos, esses passos são respectivamente denominados θ (theta), ρ (rho), π (pi), χ (chi) e ι (iota). Cada um desses passos trata de forma específica os dados contidos na matriz de entrada.

A Figura 5 mostra a função *Keccak-f[b]*, recebendo a matriz de entrada denominada "A" e os passos aplicando suas funções a partir da matriz de entrada.

```
Keccak-f[b](A) {
  forall i in 0…nr-1
    A = Round[b](A, RC[i])
  return A
}

Round[b](A,RC) {
  θ step
  C[x] = A[x,0] xor A[x,1] xor A[x,2] xor A[x,3] xor A[x,4],   forall x in 0…4
  D[x] = C[x-1] xor rot(C[x+1],1),                            forall x in 0…4
  A[x,y] = A[x,y] xor D[x],                          forall (x,y) in (0…4,0…4)

  ρ and π steps
  B[y,2*x+3*y] = rot(A[x,y], r[x,y]),                forall (x,y) in (0…4,0…4)

  χ step
  A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y]), forall (x,y) in (0…4,0…4)

  ι step
  A[0,0] = A[0,0] xor RC

  return A
}
```

Figura 5 - *Keccak-f[b]*.

Fonte: BERTONI,et al, 2010

Em uma segunda parte de do algoritmo são tratados os outros dois parâmetros, que determinam o *bitrate* e a capacidade. A função é denominada *Keccak-f[r,c]*, onde a capacidade é representada por *c* e o *bitrate* é representado por *r*, como mostrado na Figura 6, (BERTONI,et al, 2010).

```
KECCAK[r,c](M) {
  Initialization and padding
  S[x,y] = 0,                                  forall (x,y) in (0…4,0…4)
  P = M || 0x01 || 0x00 || … || 0x00
  P = P xor (0x00 || … || 0x00 || 0x80)

  Absorbing phase
  forall block Pi in P
    S[x,y] = S[x,y] xor Pi[x+5*y],             forall (x,y) such that x+5*y < r/w
    S = KECCAK-f[r+c](S)

  Squeezing phase
  Z = empty string
  while output is requested
    Z = Z || S[x,y],                           forall (x,y) such that x+5*y < r/w
    S = KECCAK-f[r+c](S)

  return Z
}
```

Figura 6 - *Keccak-f[r, c].*

Fonte: BERTONI,et al, 2010

Essa segunda parte trata os dados de entrada e saída da função principal *Keccak-f[b]*,especificando como devem ser divididos e reagrupados os dados recebidos e gerados ao final do *hash*, tratando – se da interface da função.

3.3 IMPLEMENTAÇÕES EM HARDWARE (FPGA)

Field Programmable Gate Arrays (FPGAs) são dispositivos semicondutores programáveis que são baseados em torno de uma matriz de blocos lógicos configuráveis (CLBs) conectado por interconexões programáveis, FPGAs podem ser programados para a aplicação desejada ou para requisitos de funcionalidade, (Xilinx, 2011). A Figura 7 apresenta uma placa de desenvolvimento contendo FPGAs.

Figura 7 – Placa de FPGA.
Fonte: DIGILENTINC, 2010.

As implementações que utilizam FPGA, podem ser desenvolvidas em mais de uma linguagem de descrição como VHDL e Verilog, elas se diferem das linguagens convencionais, pois descrevem o comportamento que o hardware deve assumir para ser realmente efetivo.

Por se tratar exatamente de hardware, se torna um atrativo para o desenvolvimento de projetos que exigem segurança, e como tal é uma boa opção para o desenvolvimento para o algoritmo Keccak por conceder uma maior velocidade e segurança.

Algumas implementações são apresentadas e disponibilizadas pelo próprio NIST e permitem que sejam usados como parâmetro de comparação. Como demonstra a Figura 8 e a Figura 9, podem ser implementados de diversas maneiras.

```
- chi computation and iota when needed
        when chi0 =>
            enW<='0';
            enR<='1';
            counter_plane<=0;
            counter_sheet<=0;
            mem_addr<=25;
            fsm_st<=chi1;
            command<="00000000";
        when chi0bis =>
            enW<='0';
            enR<='1';

            mem_addr<=25+5*counter_plane+counter_sheet;
            fsm_st<=chi1;
            command<="00000000";
        when chi1 =>
            --mem_addr<=25+5*counter_plane+((counter_sheet+1)mod 5);
            mem_addr<=25+5*counter_plane+f_mod5_p1(counter_sheet);
            command<="00000001";
            fsm_st<=chi2;
```

Figura 8 - Código voltado à diminuição de área.

Fonte: NIST, 2010.

Dependendo do foco pelo qual se deseja tratar, existe a possibilidade de direcionar a implementação para dois principais aspectos, maior velocidade ou menor área. Neste acaso com mostrado na Figura 8 e Figura 9 que são imagens das implementações correlatas, embora aparentem ser diferentes, ambas desempenham a mesma tarefa.

Na Figura 8, o foco do desenvolvimento é a redução de área ocupada dentro dos módulos do FPGA e na Figura 9 o foco é a maior velocidade de desempenho do algoritmo.

```
--chi
i0000: for y in 0 to 4 generate
    i0001: for x in 0 to 2 generate
        i0002: for i in 0 to 63 generate
            chi_out(y)(x)(i)<=chi_in(y)(x)(i) xor  ( not(chi_in (y)(x+1)(i))and chi_in (y)(x+2)(i));

        end generate;
    end generate;
end generate;

    i0011: for y in 0 to 4 generate
        i0021: for i in 0 to 63 generate
            chi_out(y)(3)(i)<=chi_in(y)(3)(i) xor  ( not(chi_in (y)(4)(i))and chi_in (y)(0)(i));

        end generate;
    end generate;

    i0012: for y in 0 to 4 generate
        i0022: for i in 0 to 63 generate
            chi_out(y)(4)(i)<=chi_in(y)(4)(i) xor  ( not(chi_in (y)(0)(i))and chi_in (y)(1)(i));

        end generate;
    end generate;
```

Figura 9 -Código voltado à maior velocidade.

Fonte: NIST, 2010.

As implementações do Keccak referentes à FPGA que utilizam VHDL para descrição, disponibilizados pelo NIST, são voltadas para ambos os aspectos, ele dispõe tanto de implementações que priorizam velocidade de desempenho quanto redução de área.

4. ARQUITETURA E DESCRIÇÃO EM VHDL DO ALGORITMO KECCAK

O algoritmo Keccak foi desenvolvido até a função *Keccak-f[b]*, onde se encontra o núcleo principal da função de *hash* especificada. Sua descrição foi feita usando-se a linguagem VHDL.

A utilização de linguagens para descrição de hardware ao invés de utilização de esquemático proporciona um nível maior abstração do circuito projetado, tornando mais fácil a descrição de projetos com maior complexidade de implementação.

Seu uso também proporciona várias vantagens como redução de tempo e custo de desenvolvimento, maior nível de abstração, projetos independentes de tecnologia, facilidade de atualização dos projetos, (MORENO, et al, 2003).

4.1 ARQUITETURA PROPOSTA PARA O KECCAK

Na arquitetura proposta para o algoritmo Keccak, foi utilizada à linguagem VHDL para implementação do código, assim como resultado final pode-se obter um componente de hardware.

Para isso foi utilizada a placa de FPGA Virtex4 com dispositivo XC4VLX15, vista na Figura 10, fabricada pela Xilinx, inventora e maior fabricante de FPGAs, sendo líder mundial em fornecimento de plataformas programáveis, (Xilinx, 2011).

Figura 10 –Virtex4.

Fonte: HITECHGLOBAL, 2011.

Essa placa foi escolhida em função do algoritmo Keccak descrito em VHDL, pois depois de sintetizado, os modelos de placas disponíveis para uso não conseguiram comportar o programa descrito.

Como a ferramenta CAD (*Computer-Aided Design*) fornece as especificações da placa, pode-se obter os resultados estatísticos relevantes ao projeto, mesmo sem ter os modelos físicos das placas para testes reais.

4.2 DESCRIÇÃO E SIMULAÇÃO DO KECCAK

Como propõe o pseudocódigo mostrado anteriormente na Figura 5, a implementação em VHDL foi baseada na estrutura apresentada por ele adaptada as restrições e peculiaridades da linguagem.

Na Tabela 4 são mostrados todas as matrizes recebidas e geradas na implementação da função de *hash*, a a_matriz em especial começa na função com os valores à serem criptografados e termina com os valores iniciais sobrescritos, recebendo os valores do *hash* de saída.

Tabela 4 – Matrizes da Função.

Passo Teta	
Valores recebidos	*Valores gerados*
a_matriz	c_matriz
	d_matriz
	ro_matriz
Passo Ro e Pi	
Valores recebidos	*Valores gerados*
ro_matriz	b_matriz
Passo Chi	
Valores recebidos	*Valores gerados*
b_matriz	chi_matriz
Passo Iota	
Valores recebidos	*Valores gerados*
chi_matriz	a_matriz

Fonte: (Próprio autor, 2011).

Em VHDL a descrição da função Keccak gera depois da síntese um hardware de tamanho grande, pois a cada chamada da função é gerado um novo bloco do hardware, assim para economia de área não foi utilizado esse método.

O código descrito compreende a codificação para palavras de 64 bits, pois se tratando de hardware existe vantagem quanto à velocidade de processamento.

Além das bibliotecas padrões do VHDL disponível no software usado para implementação, foi criada mais uma biblioteca denominada Keccak_tipos para ser utilizada no projeto, nela foram definidos os tipos que possibilitam que os vetores e as matrizes comportem palavras de 64 bits, não suportada por outros tipos, mostrado no código abaixo.

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

library work;
package keccak_tipos is
--types
type bi_matriz is array(0 to 4, 0 to 4) of std_logic_vector(63 downto 0);
type matriz is array(0 to 4) of std_logic_vector(63 downto 0);
end package;
```

A biblioteca criada contém o tipo bi_matriz que é uma matriz de tamanho 5 x 5 com palavras de 64 bits e o tipo matriz é um vetor de cinco posições com palavras de 64 bits. Sendo a biblioteca chamada de dentro da biblioteca padrão work, como indica o código logo abaixo. Nessa biblioteca padrão são armazenadas todas as bibliotecas criadas durante uma implementação.

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

library work;
use work.keccak_tipos.all;
```

O programa é composto pelas portas de entrada clock e reset e a porta de saída saidabyte, que mostra os valores do cálculo para um processo paralelo ao principal, essa saidabyte possibilita que o código seja sintetizado, como demonstra o código.

```
--- KECCAK ROUNDS

Entity Keccak_rounds is
port(clk: in std_logic; --CLOCK
rst: in std_logic; --RESET
saidabyte: out std_logic_vector(63 downto 0) ); -- SAÍDA DE 64 BITS
end Keccak_rounds;
```

Na implementação conforme se pode observar, existem ainda os sinais de comunicação rc_matriz e saída, sendo o rc_matriz um sinal que recebe os valores constantes de RC, dada pelo próprio algoritmo e a saída que retorna os valores dos cálculos.

As variáveis são locais aos processos, dentro das variáveis pode-se ver as matrizes e os vetores intermediários, eles possibilitam que o algoritmo seja executado em um único período de tempo (uma borda de subida do clock), pois seus valores são assíncronos ao clock.

```
architecture Behavioral of Keccak_rounds is
type RC is array (0 to 23)of std_logic_vector(63 downto 0); -- MATRIZ DA CONSTANTE
RC
signal rc_matriz: RC; -- SINAL DE COMUNICAÇÃO DA MATRIZ RC
signal saida:  bi_matriz; -- SAIDA DA FUNÇÃO
begin


process (clk,RST)
     variable aux: std_logic_vector(63 downto 0); -- variavelauxiliar de 64 bits
     variablerc: std_logic_vector(63 downto 0); -- variavel auxiliar de 64 bits
     variablea_matriz: bi_matriz; -- matriz 5x5 de 64 bits
     variableb_matriz: bi_matriz;-- matriz 4x4 de 64 bits
     variablec_matriz: matriz; -- vetor de 5 posições de 64 bits
     variabled_matriz: matriz; -- vetor de 5 posições de 64 bits
     variablero_matriz: bi_matriz; -- matriz 5x5 de 64 bits
     variablechi_matriz: bi_matriz; -- matriz 5x5 de 64 bits
     variablesos: std_logic_vector(63 downto 0); -- variavel auxiliar de 64 bits
     variable itera: integer; -- variavel auxiliar do tipo inteiro
```

Durante o reset a matriz rc_matriz que recebe os valores de cada uma das suas vinte e quatro posições, cada uma dessas posições é dada pela quantidade de repetições na execução do algoritmo. Essa quantidade de repetições é definida pela largura de permutação utilizada, que no caso é a largura de 1600.

Sendo b =1600, então pela fórmula de rotações em que $n_r = 12 + 2^{\ell}$, assim para definir 2^{ℓ} usa-se a função $2^{\ell} = b/25$. Obtém-se que 2^{ℓ}é igual a seis, assim o cálculo de n_r resulta no valor vinte e quatro, a quantidade de rotações.

Também durante o reset é zerado o valor da variável itera, que serve de auxiliar para cálculo de qual valor rotacionado será adicionado na b_matriz para o cálculo subsequente e cálculo na próxima iteração.

Como no código abaixo o reset é assíncrono ao clock, uma vez que se deve primeiro inicializar a simulação do circuito com o reset dos valores.

```
begin
IF RST='1' THEN --reset
     itera := 0;
     -- constantes rc
     rc_matriz(0)<= x"0000000000000001";
     rc_matriz(1)<= x"0000000000008082";
     rc_matriz(2)<= x"800000000000808A";
     rc_matriz(3)<= x"8000000080008000";
     rc_matriz(4)<= x"000000000000808B";
     rc_matriz(5)<= x"0000000080000001";
     rc_matriz(6)<= x"8000000080008081";
     rc_matriz(7)<= x"8000000000008009";
     rc_matriz(8)<= x"000000000000008A";
     rc_matriz(9)<= x"0000000000000088";
     rc_matriz(10)<= x"0000000080008009";
     rc_matriz(11)<= x"000000008000000A";
     rc_matriz(12)<= x"000000008000808B";
     rc_matriz(13)<= x"800000000000008B";
     rc_matriz(14)<= x"8000000000008089";
     rc_matriz(15)<= x"8000000000008003";
     rc_matriz(16)<= x"8000000000008002";
     rc_matriz(17)<= x"8000000000000080";
     rc_matriz(18)<= x"000000000000800A";
     rc_matriz(19)<= x"800000008000000A";
     rc_matriz(20)<= x"8000000080008081";
     rc_matriz(21)<= x"8000000000008080";
     rc_matriz(22)<= x"0000000080000001";
     rc_matriz(23)<= x"8000000080008008";
```

Logo abaixo, o algoritmo demonstra o passo θ (theta), nele são atribuídos valores para os vetores c_matriz e d_matriz e a matriz ro_matriz, onde os valores são gerados através de operações lógicas dos elementos da matriz a_matriz.

```
--passo teta
for i in 0 to 4 loop
     c_matriz(i):= a_matriz(i,0) xor a_matriz(i,1) xor a_matriz(i,2) xor
     a_matriz(i,3) xor a_matriz(i,4);
end loop;

for i in 0 to 4 loop
     aux:= c_matriz((i+1) mod 5);
     d_matriz(i):= c_matriz((i+4) mod 5) xor (aux (62 downto 0) & aux (63));

     for j in 0 to 4 loop
          ro_matriz(i,j):= a_matriz(i,j) xor d_matriz(i);
     end loop;
end loop;
```

O próximo passo é o ρ (rho) eπ (pi), nele são adicionados na matriz b_matriz os valores da ro_matriz rotacionados.

```
for i in 0 to 4 loop
       for j in 0 to 4 loop
             aux:=ro_matriz(i,j);
             itera:=5*i+j;

             case itera is
                  when 0 =>sos:= aux;--00
                  when 1 =>sos:=(aux(27 downto 0)& aux(63 downto 28));--01
                  when 2 =>sos:=(aux(60 downto 0)& aux(63 downto 61));--02
                  when 3 =>sos:=(aux(22 downto 0)& aux(63 downto 23));--03
                  when 4 =>sos:=(aux(45 downto 0)& aux(63 downto 46));--04
                  when 5 =>sos:=(aux(61 downto 0)& aux(63 downto 62));--10
                  when 6 =>sos:=(aux(19 downto 0)& aux(63 downto 20));--11
                  when 7 =>sos:=(aux(53 downto 0)& aux(63 downto 54));--12
                  when 8 =>sos:=(aux(18 downto 0)& aux(63 downto 19));--13
                  when 9 =>sos:=(aux(61 downto 0)& aux(63 downto 62));--14
                  when 10 =>sos:=(aux(1 downto 0)& aux(63 downto 2));--20
                  when 11 =>sos:=(aux(57 downto 0)& aux(63 downto 58));--21
                  when 12 =>sos:=(aux(20 downto 0)& aux(63 downto 21));--22
                  when 13 =>sos:=(aux(48 downto 0)& aux(63 downto 49));--23
```

```
                when 14 =>sos:=(aux(2 downto 0)& aux(63 downto 3));--24
                when 15 =>sos:=(aux(35 downto 0)& aux(63 downto 36));--30
                when 16 =>sos:=(aux(8 downto 0)& aux(63 downto 9));--31
                when 17 =>sos:=(aux(38 downto 0)& aux(63 downto 39));--32
                when 18 =>sos:=(aux(42 downto 0)& aux(63 downto 43));--33
                when 19 =>sos:=(aux(7 downto 0)& aux(63 downto 8));--34
                when 20 =>sos:=(aux(36 downto 0)& aux(63 downto 37));--40
                when 21 =>sos:=(aux(43 downto 0)& aux(63 downto 44));--41
                when 22 =>sos:=(aux(24 downto 0)& aux(63 downto 25));--42
                when 23 =>sos:=(aux(55 downto 0)& aux(63 downto 56));--43
                when 24 =>sos:=(aux(49 downto 0)& aux(63 downto 50));--44
                when others=>null;
            end case;
            b_matriz(j,(2*i+3*j)mod 5):=sos;
        end loop;
end loop;
```

Os próximos passos são χ (chi) e ι (iota), respectivamente no passo chi são feitas operações lógicas entre os valores da matriz b_matriz e os valores obtidos são inseridos na matriz auxiliar chi_matriz.

No passo subsequente a todos os valores da matriz auxiliar são inseridos na a_matriz, depois é feito a operação xor com o elemento da primeira posição, e todo processo descrito é repetido mais vinte e três vezes, para então obter os valores de saída.

```
-- chi
for i in 0 to 4 loop
     for j in 0 to 4 loop
          aux:= (not (b_matriz(((i+1) mod 5),j)))and b_matriz(((i+2) mod 5),j);

          chi_matriz(i,j):= b_matriz(i,j) xor aux ;
     end loop;
end loop;

-- iota
a_matriz := chi_matriz;
a_matriz(0,0):= chi_matriz(0,0) xor rc;

-- saída
for i in 0 to 4 loop
     for j in 0 to 4 loop
          saida(i,j)<= a_matriz(i,j);
     end loop;
```

```
end loop;
```

A simulação inicia-se com o reset do circuito, durante o reset são atribuídos valores para a matriz que contém a constante RC e zerado o valor da variável itera.

Assim que se inicia a borda de subida são adicionados os valores da matriz de entrada "A", todos os valores contidos nessa matriz para gerar a simulação na Figura 11 são palavras que contém o valor zero.

Depois de um ciclo de clock todas as vinte e quatro iterações são realizadas e os valores resultantes das operações são observáveis no sinal denominado saída, onde os valores são palavras de 64 bits, como se pode verificar na Figura 11.

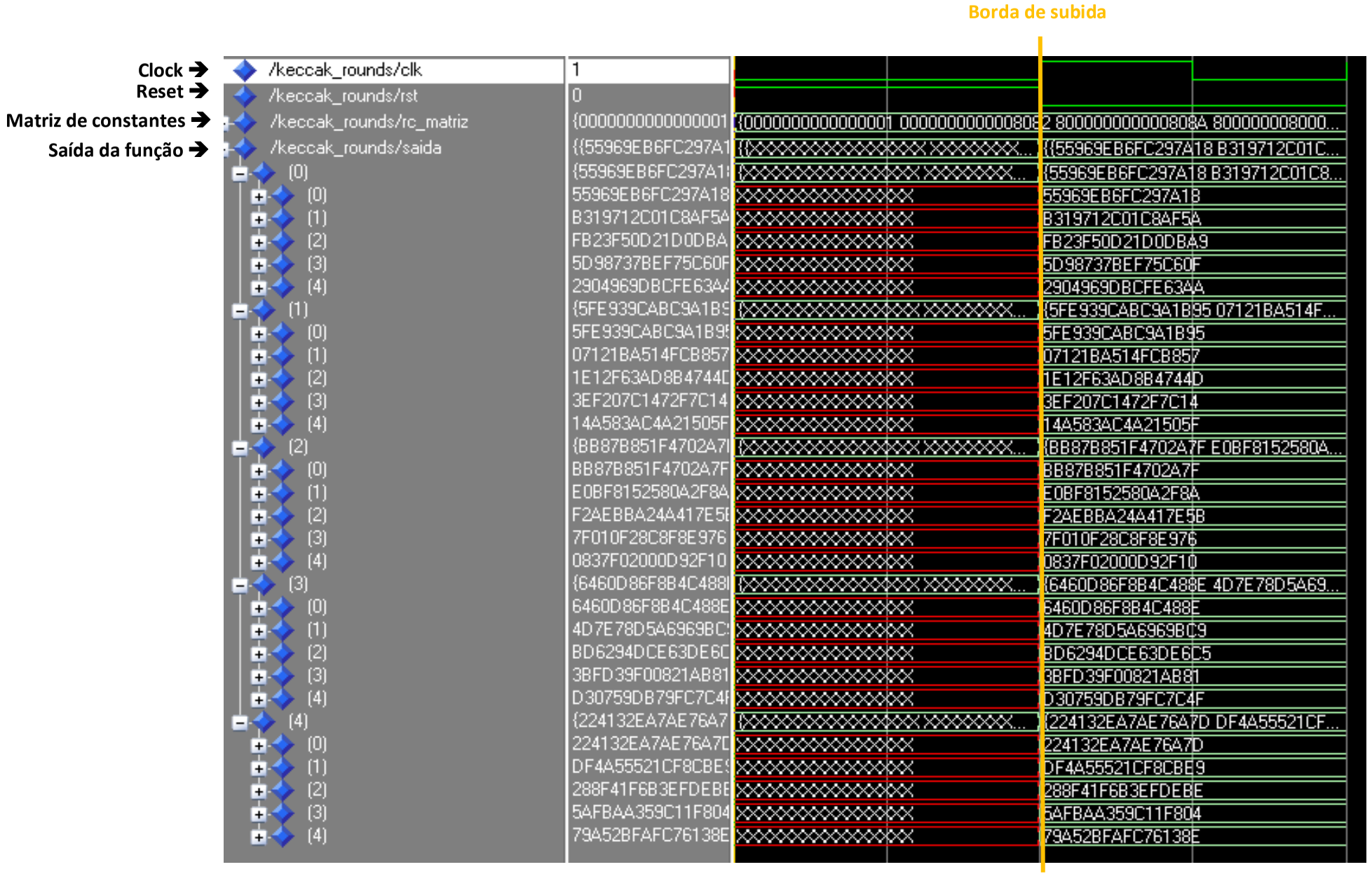


Figura 11 – Simulação.

Fonte: (Próprio autor, 2011).

5. IMPLEMENTAÇÃO EM FPGA DO ALGORITMO KECCAK

5.1 METODOLOGIA DE IMPLEMENTAÇÃO

A metodologia de implementação do algoritmo, foi realizada de uma forma sequencial que acompanhasse a ordem de execução do pseudocódigo, como é possível ver nos códigos mostrados no capítulo anterior.

Embora, havendo peculiaridades da linguagem que necessitaram de tratamento, como o principal caso que são as posições das matrizes em algumas operações, os autores do método dão uma solução usando o módulo de cinco para definir as posições. Essa solução foi utilizada, durante o desenvolvimento do algoritmo.

O VHDL permitindo a programação em paralelo, o código foi descrito de forma que a sequencia fosse estabelecida pela ordem de execução e dependência de dados do algoritmo Keccak. Como representado pela Figura 12, existe uma dependência direta dos valores calculados nos passos anteriores.

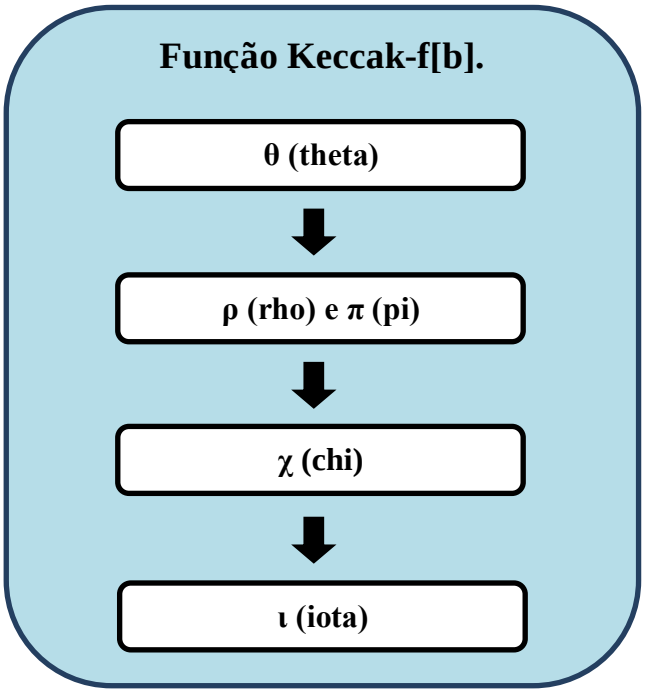


Figura 12 – Sequências de passos do algoritmo Keccak.

Fonte: (Próprio autor, 2011).

Para implementação, compilação e levantamento estatístico do algoritmo descrito foi utilizada a ferramenta Xilinx ISE 9.1i, por ter sido utilizada em sala de aula proporcionou uma maior familiaridade e facilidade de desenvolvimento do projeto.

Como simulador foi utilizado o simulador presente na ferramenta ModelSim SE 6.1b, por ser um simulador muito menos complexo do que o utilizado no software do Xilinx ISE 9.1i.

5.2 RESULTADOS EM FPGA

Depois de finalizada a implementação e compilada o software da Xilinx faz o levantamento estatístico dos componentes utilizados no projeto, esses valores são gerados á partir de dos dados da placa contidos no software.

Como o FPGA possui em seu circuito integrado milhares de portas logicas semelhantes, a menor unidade lógica é denominada *Slice*, sendo uma unidade muito versátil, entre uma das suas diversas configurações está à possibilidade de configurá-la como uma LUT (*LookUp Tables*). Essas LUTs por sua vez armazenam as tabelas de especificação do modo de funcionamento do projeto.

Todos os dados gerados do algoritmo Keccak implementado utilizando a placa de FPGA Virtex4 podem ser observados nas tabelas. A Tabela 5 mostra os dados visando a síntese de desempenho em hardware e a Tabela 6 os dados visam a síntese de ocupação em hardware.

Tabela 5 – Resultados da síntese voltados a desempenho.

Componentes	Utilizados	Disponíveis	% Utilizado
LUTs	424	12288	3%
Registradores lógicos	132	12288	1%
Slices	218	6144	3%

Fonte: (Próprio autor, 2011).

Tabela 6 – Resultados da síntese voltados á área.

Componentes	Utilizados	Disponíveis	% Utilizado
LUTs	331	12288	2%
Registradores lógicos	64	12288	0%
Slices	175	6144	2%

Fonte: (Próprio autor, 2011).

Como pode ser observada nas tabelas anteriores, a porcentagem dos componentes utilizados foi mínima, uma vez que essa placa é uma placa de FPGA de grande capacidade, para a aplicação desenvolvida.

6. CONCLUSÕES

Como proposto, foi implementada uma versão do algoritmo de função de *hash* Keccak, finalista da chamada do NIST, sendo uma versão validada logicamente. Embora não ocupe muito espaço dentro de uma placa de FPGA utilizada no projeto, não são todas as placas que suportam o programa implementado.

Os requisitos mínimos de uma placa de FPGA que execute a aplicação desenvolvida é uma placa que possua pelo menos quinhentos LUTs, duzentos registradores lógicos e trezentos Slices.

Apesar de não ser um resultado tangível, o projeto proporcionou um aprendizado mais aprofundado da linguagem VHDL e FPGAs, assim como suas limitações e inúmeras possibilidades.

Também tornou mais visível a correlação entre as disciplinas ministradas durante todo o curso de Ciência da Computação, como a importância e necessidade dos estudos referentes à criptografia nos dias atuais.

O código implementado pode contribuir para comparação com eventuais implementações em hardware, possuindo uma implementação simples e de fácil entendimento por qualquer pessoa que possua conhecimentos básicos de lógica e programação, independente da linguagem.

Como foi implementada apenas a função *Keccak-f[b]* que representa aproximadamente 75% do algoritmo, para futuros trabalhos o código já descrito pode ser melhorado e completado com a função *Keccak-f[r, c]* que é a função que determina interface, então com os devidos testes e validações pode-se obter um hardware para criptografia.

7. REFERÊNCIAS

BERTONI, DAEMEN, PEETERS E ASSCHE; **The Keccak sponge function Family**, 2010. Diponível em:<**http://keccak.noekeon.org/specs_summary.html** >. Acesso em: 18 Ago. 2011.

BURNETT, Steve; PAINE, Stephen. **Criptografia e Segurança:** O Guia Oficial RSA. Tradução de Edson Fumankiewicz. Rio de Janeiro: Campus, 2002.

DIGILENTINC, **Spartan-6 FPGA DevelopmentBoard,** 2010. Diponível em:<http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS>. Acesso em: 16 Nov. 2011.

FILHO, Audir C. O.; **Estudos de algoritmos criptográficos e implementação de uma entidade certificadora**, 2008. Disponível em: <http://aldeia3.computacao.net/greenstone/cgi-bin/library.cgi?e=d-01000-00---off-0trabalho--00-1----0-10-0---0---0direct-10---4-------0-1l--11-es-50---20-about---00-3-1-00-0-0-11-1-0gbk-00&cl=CL1.3&d=HASH0145dde354a98f5f1f8a6532&x=1>. Acesso em: 19 Jul. 2011.

HITECHGLOBAL: **V4IP-500 :XilinxVirtex 4 LX60 & ARM926/1136 Verification IP Platform,** 2011. Disponível em: <http://hitechglobal.com/Boards/V4LX60.htm>. Acessoem 12 Nov. 2011.

MENEZES, Alfred J.; OORSCHOT, Paul C.; VANSTONE, Scott A.; **Handbook of Applied Cryptography.**CRC Press, 1996. Disponível em: <http://www.cacr.math.uw aterloo.ca/hac/>. Acesso em: 14 Mai. 2011.

MORENO, Edward D.; PEREIRA, Fábio D.; CHIARAMONTE, Rodolfo B.; **Criptografia em Software e Hardware.** São Paulo: Novatec, 2005.

MORENO, Edward D.; PEREIRA, Fábio D.; PENTEADO, Cesar G.; PERICINI, Rodrigo A.; **Projeto, Desenvolvimento e Aplicações de Sistemas Digitais em Circuitos Programáveis (FPGAs)**. Pompéia: Bless, 2003.

NIST, National Institute of Standards and Technology. **Cryptographic hash Algorithm Competition,**2005. Diponível em:<http://csrc.nist.gov/groups/ST/hash/sha-3/index.html >. Acessoem: 05Ago. 2011.

NIST, National Institute of Standards and Technology.**Tentative Timeline of the Development of New Hash Functions**, 2006.Disponível em: < http://csrc.nist.gov/groups/ST/hash/timeline.html >Acesso em: 08 Ago. 2011.

NIST, National Institute of Standards and Technology.**Third (Final) Round Candidates**, 2009. Disponível em: Disponivel em:< http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html >Acesso em: 10 Ago. 2011.

SCHNEIER, Bruce. **Applied Cryptography:** Protocols, Algorithms, and Source Code in C. 2nd Edition.New York: John Wiley& Sons, 1996.

TKOTZ, V. Criptografia Numaboa. **A Linha do Tempo da Criptografia na Antiguidade**, 2005. Disponível em: < http://www.numaboa.com/criptografia/historia/157-antiguidade>. Acesso em: 14 de Jul. 2011

TKOTZ, V. Criptografia Numaboa. **Algoritmos Básicos da Criptografia Atual**, 2005. Disponível em: <http://www.numaboa.com/index.php?option=com_content&view=article&id=311&Itemid=99 >. Acesso em: 18 de Jul. 2011

TKOTZ, V. Criptografia Numaboa. **As Funções Hash**, 2005**.** Disponível em: < http://www.numaboa.com/criptografia/hashes/338-hash>. Acesso em: 20 de Jul. 2011

TKOTZ, V. Criptografia Numaboa. **Introdução à Criptogrfia**, 2005**.** Disponível em: < http://www.numaboa.com/criptografia/gerais/153-introducao >. Acesso em: 14 de Jul. 2011

TKOTZ, V. Criptografia Numaboa. **O que é um Hash**, 2005**.** Disponível em: < http://www.numaboa.com/criptografia/hashes/340-o-que-e-hash>. Acesso em: 20 de Jul. 2011

XILINX. **FPGA Families,** 2011. Disponível em: < http://www.xilinx.com/products/silicon-devices/fpga/index.htm>, Acesso em Maio 2011.

XILINX. **Company Overview,** 2011. Disponível em: <http://www.xilinx.com/about/company-overview/index.htm>, Acesso em Maio 2011.

WANG, X., FENG, D., LAI, X. e YU, H. **"Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD"**, 2004. Disponível em: < http://eprint.iacr.org/2004/199.pdf >. Acesso em: 25 Mai. 2011.