FLÁVIO HENRIQUE DE FREITAS
LEANDRO CORDEIRO DAVID
ODUVALDO VICK NETO
RODOLFO JOSÉ BATISTA DE SOUZA

COPROCESSADOR CRIPTOGRÁFICO RSA

São Paulo
2009

FLÁVIO HENRIQUE DE FREITAS
LEANDRO CORDEIRO DAVID
ODUVALDO VICK NETO
RODOLFO JOSÉ BATISTA DE SOUZA

COPROCESSADOR CRIPTOGRÁFICO RSA

Texto apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro da Computação.

São Paulo
2009

FLÁVIO HENRIQUE DE FREITAS
LEANDRO CORDEIRO DAVID
ODUVALDO VICK NETO
RODOLFO JOSÉ BATISTA DE SOUZA

COPROCESSADOR CRIPTOGRÁFICO RSA

Texto apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro da Computação.

Área de concentração:
Engenharia da Computação

Orientador:
Prof. Livre-Docente Wilson Vicente Ruggiero

Co-orientadores:
Dr. Armin Werner Mitelsdorf
Prof. MSc. Stephan Kovach

São Paulo
2009

Eu sou uma parte de tudo o que encontrei.
(Alfred Lord Tennyson)

AGRADECIMENTOS

Ao professor Dr. Wilson Ruggiero, pela orientação e confiança depositada em nós.

Aos co-orientadores, professor Stephan Kovach e Dr. Armin Werner Mitelsdorf por toda a ajuda.

A João Carlos Neto, pela pronta disposição em nos ajudar.

A todos que contribuíram direta ou indiretamente para o desenvolvimento do projeto.

DEDICATÓRIA

Dedico este trabalho a minha família pelo incontestável e incondicional apoio, carinho, esforço e atenção de todos os momentos.

Flávio Henrique de Freitas

Dedico este trabalho a Ana Maria Batista de Souza, minha mãe, que sempre me apoiou de todas as formas nas minhas decisões e abriu mão muitas coisas para que eu pudesse chegar até onde cheguei e a Luiz Alberto de Souza, meu pai, que também abriu mão de realizações pessoais para que eu pudesse concretizar as minhas.

Rodolfo José Batista de Souza

Dedico este trabalho a João David, meu pai, por me ensinar o valor do conhecimento, a minha mãe, Eva Alves Cordeiro David, por me ensinar o valor da compaixão, tão importante nos momentos difíceis de um trabalho em grupo, e a meus irmãos, Geziel e Giovanni, por todo o apoio e inúmeros pedidos de favores atendidos prontamente.

Leandro Cordeiro David

Dedico este trabalho principalmente a Deus, por estar presente em todos os momentos de minha vida e me carregar nos momentos mais difíceis, a minha namorada, Kelly Cabrera, por todo o apoio emocional, e a minha família, pelo apoio material.

Oduvaldo Vick Neto

RESUMO

Estações de trabalho são alvos constantes de ataques e não são ambientes seguros para a execução de operações que envolvam informações confidenciais. O uso de dispositivos criptográficos dedicados e externos permite que tais informações sejam devidamente tratadas e manipuladas antes de entrarem em um sistema possivelmente infectado. Com isso, diminui-se o risco de ocorrências de uso ou acesso indevido a tais informações. Esse trabalho consiste na especificação, projeto e desenvolvimento de um dispositivo dedicado, um coprocessador, capaz de executar funções criptográficas. Tal dispositivo realiza funções de assinatura digital, encriptação e decriptação de dados. Uma camada de software facilita o acesso ao coprocessador, além de prover funções complementares para o funcionamento do *hardware*. O sistema especificado foi desenvolvido utilizando a linguagem de descrição de *hardware*, VHDL. O código gerado foi sintetizado e o resultado da síntese, executado em um dispositivo FPGA. O coprocessador desenvolvido é capaz de executar operações de encriptação e decriptação utilizando o algoritmo RSA, de encriptação com chave pública, operações de cálculo da função de *hash* SHA-1 e cálculo de assinaturas digitais usando SHA-1 e RSA. Os cálculos de exponenciação modular do RSA são calculadas com o uso do algoritmos de multiplicação de Montgomery.

Palavras-chave: Segurança. Criptografia. RSA. FPGA. Montgomery. SHA-1.

ABSTRACT

Workstations are constant targets of attacks, and there is no guarantee that transactions involving sensitive information are executed safely on them. The use of dedicated and external cryptographic devices allows that such informations are properly handled and treated before entering in a possibly infected system, reducing the risk of unauthorized use or access. This work consists of the specification, design and implementation of a dedicated device capable of executing cryptographic functions. The device performs functions of digital signature, encryption and decryption of data. A layer of software facilitates access to the coprocessor, and provides additional functions for the operation of the textit (hardware). The specified system was developed using the hardware description language, VHDL. The code generated was synthesized and, the synthesis result, executed in a FPGA device. The coprocessor developed is capable of performing encryption and decryption using the RSA algorithm of public-key cryptography, hash function using the SHA-1 algorithm and digital signature using both algorithms. The modular exponentiation of the RSA is calculated by the algorithms of multiplication of Montgomery.

Keywords: Security. Cryptography. RSA. FPGA. Montgomery. SHA-1.

LISTA DE FIGURAS

1.1 Diagrama Geral do Projeto 18

2.1 Exemplo de encriptação e decriptação 27

2.2 Cálculo de Assinatura Digital 32

2.3 Verificação de Assinatura Digital 33

3.1 Arquitetura Geral do Projeto 40

3.2 Arquitetura do Software 41

3.3 Diagrama de Classes 42

3.4 Demostração: Envio de mensagens eletrônicas assinadas 43

3.5 Padrão do Bloco de Assinatura 44

3.6 Fluxo de Informação da Demonstração 45

3.7 Entradas e saídas do bloco Interface USB 48

3.8 Entradas e saídas do bloco RSA 48

3.9 Entradas e saídas do bloco Multiplicador Modular 49

3.10 Entradas e saídas do bloco Exponenciador Modular 50

3.11 Função de *Hash*: entradas e saídas 50

3.12 Organização do Hardware 51

4.1 Estrutura de blocos da XEM 3005 54

4.2 Exemplo de painel virtual criado com o FrontPanel 55

6.1 Programa para Envio de *email* 65

6.2 Geração do Par de Chaves Criptográficas 66

6.3 Complemento identificou Assinatura Correta 67

6.4 Complemento identificou Assinatura incorreta 68

6.5 Diagramas de estado para a escrita e leitura na interface USB 70

6.6 Diagramas de estado para a escrita e leitura na interface USB 71

6.7 Fluxograma da máquina de estados de controle do bloco RSA 75

6.8 Fluxograma da máquina de estados do Exponenciador Modular 76

6.9 Fluxograma da máquina de estados do Multiplicador de Montgomery 77

8.1 Comparação entre o desempenho medido em software e calculado em hardware para operações de encriptação com RSA 256 82

IV.1 Carta de tempos com os sinais do bloco Interface USB durante o envio de dados do software para o hardware 97

IV.2 Carta de tempos com os sinais do bloco Interface USB durante o envio de dados do hardware para o software 98

V.1 Sinais de entrada e saída do módulo okHostInterface 99

V.2 Diagrama de tempos do módulo okPipeIn 100

V.3 Diagrama de tempos do módulo okPipeOut 101

VIII.1Carta de tempos do bloco Multiplicador Modular 108

IX.1 Carta de tempos do bloco Exponenciador Modular 110

X.1 Carta de tempo do bloco SHA-1 112

LISTA DE TABELAS

4.1 Características consideradas na escolha do modelo de kit de desenvolvimento de hardware 53

VII.1 Formato de Dados da palavra de operação de entrada 104

VII.2 Códigos de Operações solicitadas pelo software 104

VII.3 Códigos de Operações devolvidos pelo hardware 105

LISTA DE ABREVIATURAS

API Application Programming Interface

EEPROM Electrically-Erasable Programmable Read-Only Memory

FPGA Field Programmable Gate Array

HDL Hardware Description Language

IHM Interface Homem Máquina

LabVIEW Laboratory Virtual Instrument Engineering Workbench

LED Light Emitting Diode

PC Personal Computer

PROM Programable Read Only Memory

RSA Ronald **R**ivest, Adi **S**hamir e Leonard **A**dleman, os criadores do algoritmo

SHA Secure Hash Algorithm

SDRAM Synchronous Dynamic Random Access Memory

USB Universal Serial Bus

VHDL VHSIC Hardware Description Language (ver VHSIC abaixo)

VHSIC Very High Speed Integrated Circuits

SUMÁRIO

1 Introdução **17**

1.1 Objetivo . . . 17

1.2 Justificativa e Motivação . . . 17

1.3 Organização . . . 19

2 Aspectos Conceituais **21**

2.1 Segurança . . . 21

2.2 Visão Geral da Criptografia . . . 22

2.3 Base Matemática . . . 22

2.3.1 Aritmética Modular . . . 22

2.3.2 Algoritmo de Euclides Estendido . . . 23

2.3.3 Função totiente de Euler . . . 24

2.3.4 Redução de Montgomery . . . 24

2.3.5 Exponenciação Modular . . . 26

2.3.6 Multiplicação Modular . . . 26

2.4 Criptografia . . . 27

2.5 Algoritmo de Chave Simétrica . . . 28

2.5.1 Cifras de Fluxo . . . 28

2.5.2 Cifra de Blocos . . . 28

2.6 Algoritmo de Chave Assimétrica . . . 28

2.6.1 O Algoritmo RSA . . . 29

2.7 Funções *HASH* . . . 30

2.7.1 SHA - 1 . . . 31

2.8 Autoridade Certificadora . . . 31

2.9 Assinatura Digital . . . 31

2.10 Ataques . . . 33

2.10.1 Ataques de Segurança . . . 33

2.10.2 Ataques por Canais secundários . . . 35

2.11 Dispositivos FPGA . . . 36

2.11.1 Funcionamento e Arquitetura de um FPGA . . . 36

3 Especificação Do Projeto **39**

3.1 Descrição Do Projeto . . . 39

3.2 Arquitetura Do Sistema . . . 40

3.3 Software . . . 40

3.3.1 Software de Teste . . . 42

3.3.2 Software De Demonstração . . . 43

3.3.3 Software Com Funções Criptográficas . . . 46

3.3.4 Driver USB . . . 46

3.4 Hardware . . . 47

3.4.1 Interface USB . . . 47

3.4.2 Bloco RSA . . . 48

3.4.3 Arquitetura do bloco RSA . . . 48

3.4.4 Multiplicador Modular . . . 49

3.4.5 Exponenciador Modular . . . 49

3.4.6 Bloco HASH (SHA-1) . . . 50

4 Ferramentas Escolhidas **52**

4.1 Hardware - *Kit* De Desenvolvimento . . . 52

4.1.1 XEM3005 53
4.1.2 Front Panel 53
4.1.3 API De Linguagem De Alto Nível 54
4.2 Software De Desenvolvimento 55
4.2.1 ISE 55
4.2.2 ModelSim 56
4.3 Linguagens 57
4.3.1 Java 57
4.3.2 VHDL 57
5 Metodologia **59**
5.1 Definição do escopo 59
5.2 Escolha de tecnologias e ferramentas 60
5.3 Estudos teóricos e especificação técnica e funcional 61
5.4 Implementação e testes 61
5.4.1 Hardware 61
5.4.2 Comunicação USB 62
5.4.3 Software 63
6 Implementação **64**
6.1 Software 64
6.1.1 Interface Gráfica 64
6.1.2 Software Com Funções Criptográficas 66
6.1.3 Driver USB 66
6.2 Hardware 69
6.2.1 Interface USB 69
6.2.2 Bloco RSA 72

6.2.3 Exponenciador Modular 72
6.2.4 Multiplicador Modular 73
6.2.5 Bloco HASH (SHA-1) 73

7 Testes E Avaliação **78**
7.1 Metodologia de testes 78
7.1.1 Testes e Simulações de Hardware 78
7.2 Desempenho 79
7.3 Comparação de Desempenho 79

8 Resultados **80**
8.1 Resultados funcionais 80
8.2 Comparação de desempenho *Software* x *Hardware* 80

9 Conclusão **83**
9.1 Considerações Finais 83
9.2 Perspectivas Futuras 84

Apêndices **86**

I Especificação dos Elementos da Interface de Teste **86**

II Funções Criptográficas **89**

III Funções do Driver USB **94**

IV Sinal do Hardware da Interface USB **96**

V Elementos Disponibilizados Pela OpalKelly **99**
okHostInterface 99
okPipeIn 99

okPipeOut 100

VI Funções da biblioteca okjFrontPanel **102**

VII Bloco RSA **103**

VIII Multiplicador Modular **108**

IX Exponenciador Modular **109**

X Bloco Hash - SHA-1 **111**

1 INTRODUÇÃO

1.1 Objetivo

O objetivo desse projeto é criar um *hardware* capaz de executar funções criptográficas de assinatura digital, encriptação e decriptação de dados.

Estações de trabalhos são alvos constantes de ataques, portanto suscetíveis a invasão e perda de controle de seus recursos por entidades não autorizadas. O uso de dispositivos criptográficos dedicados e externos permite que tais informações sejam devidamente tratadas e manipuladas antes de entrarem em um sistema possivelmente infectado. Isto diminui os riscos de uso ou acesso indevido aos recursos computacionais de uma estação de trabalho.

A proposta envolve o desenvolvimento de uma arquitetura de um coprocessador e seu detalhamento lógico utilizando uma linguagem de descrição de *hardware*, com posterior implantação física em um dispositivo FPGA. Como os dados para tratamento no coprocessador têm origem em um computador, faz-se necessária a comunicação entre ambos os dispositivos. A apresentação e testes dos resultados é feita por uma interface de *software*.

1.2 Justificativa e Motivação

Vê-se um crescente aumento no uso de processamento de informações em sistemas digitais. Com isso, percebe-se a necessidade de manter sigilosas as informações tramitadas entre estes sistemas. A violação destas informações pode ser prejudicial à sociedade e aos indivíduos que delas dependem. Exemplos deste cenário são a troca de informações financeiras entre sistemas bancários, a autenticação de usuários em sistemas computacionais que necessitam de alguma restrição de uso e a privacidade de informação como serviços de *e-mail* e *sites* de comércio eletrônico.

Um dos problemas existentes hoje é que grande parte dos métodos de segurança de informações se baseiam no uso de senhas eletrônicas, com as quais é possível obter acesso a

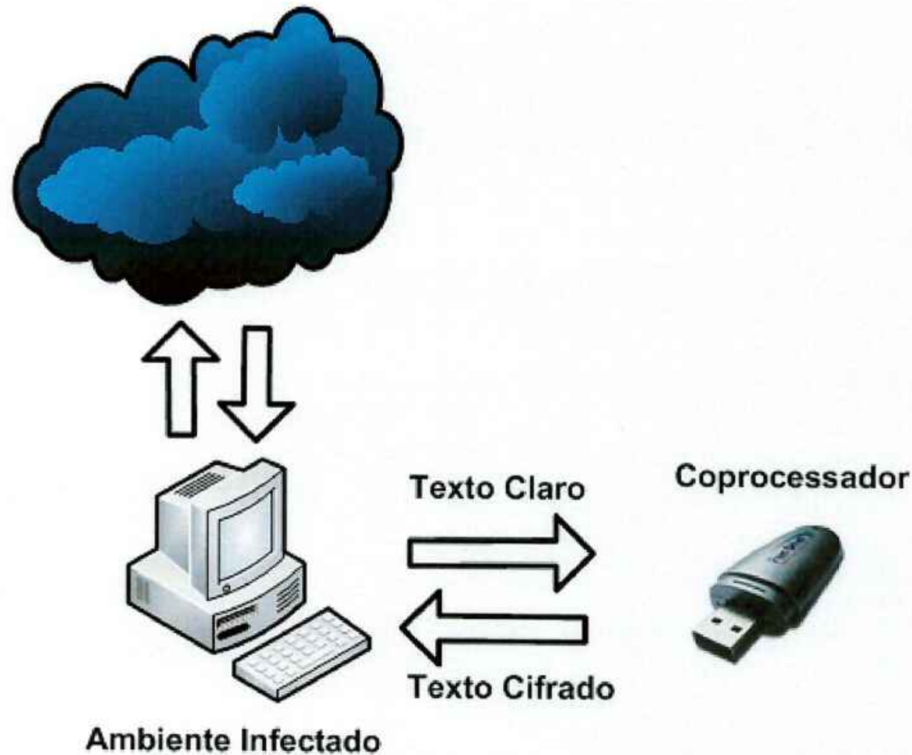

Figura 1.1: Diagrama Geral do Projeto

informações sigilosas. Esta senha é suscetível a interceptação e eventual utilização inadequada por pessoas mal intencionadas. Aliado a isso existe o problema do usuário acessar sistemas em um ambiente com alto potencial de infecção por softwares não confiáveis, aumentando assim a probabilidade de ter os dados roubados e utilizados de forma indevida.

Uma das maneiras utilizadas para roubar dados é, por exemplo, o ataque *man-in-the-middle* (**??**). Nessa técnica o atacante posiciona-se entre as duas partes que desejam trocar informações e manipula os dados conforme o próprio interesse, sem a consciência das duas outras partes envolvidas. Em uma transação bancária, por exemplo, um atacante pode fazer transações em benefício próprio a partir da conta de um cliente, sem o conhecimento deste ou do banco.

Verifica-se a necessidade de métodos que aumentem o nível de segurança, principalmente nos aspectos relacionados ao controle, armazenamento e utilização de senhas. Pensando nisso, uma das propostas viáveis que pode ser desenvolvida é a criação de uma chave física, implementada em um ambiente seguro, em que apenas quem tiver posse desta terá acesso ao sistema protegido. Para desenvolvimento dessa chave física é possível criar um dispositivo de processamento otimizado para a execução de funções criptográficas, e que garanta a manipulação dessas informações sigilosas, ou seja, um coprocessador criptográfico. Esse dispositivo protegeria o usuário de ataques de mascaramento de informação, ou seja, aqueles em que é

mostrado o conteúdo modificado ao invés do original. Outro ataque que também poderia ser evitado é o *man-in-the-middle*, pois possuiria seu próprio hardware, *display* e teclado, assim, qualquer informação tramitada por esse dispositivo, poderia ser verificada e necessitaria da confirmação do usuário para a continuação da transação.

Nesse projeto foi necessária uma limitação de escopo, devido ao tempo disponível, assim o dispositivo construído não apresenta uma tela gráfica própria e nem um teclado para interação.

1.3 Organização

Este documento está organizado em 9 capítulos.

Neste primeiro capítulo foi apresentada uma introdução ao trabalho com seu objetivo, motivação e as justificativas que levaram a fazê-lo, além da organização do documento.

No segundo capítulo, são apresentados conceitos e definições que foram utilizadas durante todo o projeto, necessários à compreensão do sistema desenvolvido. Dentre essas definições estão os conceitos de segurança e criptografia, um maior aprofundamento dos algoritmos matemáticos e criptográficos utilizados no projeto, alguns tipos de ataques possíveis a hardware, assim como conceitos a respeito das técnicas de desenvolvimento de *hardware* utilizadas na implementação.

O terceiro capítulo apresenta a especificação do projeto, com sua descrição e arquitetura gerais, e detalhes específicos de *software* e *hardware*.

No quarto capítulo são apresentadas todas as ferramentas utilizadas na criação do coprocessador criptográfico: kit de desenvolvimento de hardware, *software* de descrição de *hardware*, *software* de simulação de *hardware*, linguagem de descrição de *hardware*, e linguagem usada na implementação dos *softwares* de testes e de apresentação.

O quinto capítulo apresenta a metodologia utilizada durante todo o desenvolvimento do projeto, com todas as etapas, decisões e problemas encontrados.

O sexto capítulo contém a descrição da implementação do projeto em si, como foram feitos os módulos de *software*, como o algoritmo RSA foi implementado em *hardware*, como foi feita a comunicação do coprocessador com o PC, quais as técnicas usadas e como as ferramentas foram utilizadas.

No sétimo capítulo estão descritos os testes realizados, os resultados e avaliações, e a comparação de desempenho com outras implementações (principalmente aquelas usando apenas

software).

No oitavo capítulo são apresentados os resultados obtidos com esse trabalho, os objetivos alcançados, as dificuldades e as soluções encontradas.

No nono capítulo é apresentada uma conclusão final de todo o projeto, as perspectivas futuras em torno dele e as considerações finais.

2 ASPECTOS CONCEITUAIS

Nesse capítulo são apresentados conceitos que foram importantes para elaboração do projeto. Há uma descrição das teorias que foram aplicadas e algumas que foram estudadas e consideradas para a definição dos padrões e escolhas de tecnologias implementadas, mas que não foram diretamente aplicadas.

2.1 Segurança

Os principais serviços de segurança da informação que garantem que uma comunicação seja dita segura, segundo (**??**), são:

- *Confidencialidade:*

  O serviço de confidencialidade garante que as informações estejam resguardadas. O acesso e a modificação podem ser apenas realizadas por pessoas devidamente autorizadas.

- *Irretratabilidade:*

  Impossibilita que haja negação da autoria de uma mensagem enviada. Aplicável tanto às informações geradas pelo destinatário quanto pelo remetente.

- *Autenticação:*

  Permite que a identidade do usuário seja verificável e instranferível, ou seja, não é possível a um intruso afirmar ser quem ele não é.

- *Integridade:*

  Esse serviço visa garantir que a mensagem não sofreu alteração, proposital ou não, durante a transmissão. É responsável por identificar mudanças, caso ocorram, mas não corrigi-las ou evitá-las.

2.2 Visão Geral da Criptografia

Nesta seção é apresentada a base teórica criptográfica utilizada para o desenvolvimento do coprocessador.

2.3 Base Matemática

A base matemática que foi utilizada para entendimento dos algoritmos e elaboração do hardware e software é explorada nessa seção. Fórmulas e teoria dos números auxiliam no entendimento do algoritmo criptográfico analisado e implementado nesse projeto.

2.3.1 Aritmética Modular

Uma propriedade explorada durante a implementação do sistema é a Aritmética Modular, também conhecida como Aritmética do Relógio ou Calculadora-Relógio (**??**).

Definição, segundo (**??**): dados dois inteiros *a* e *n* , com *n* não negativo, temos:

$$a = nq + r \tag{2.1}$$

em que: $0 \leq r < q$. Chamamos q - quociente e r - resto.

Para inteiros a e $n > 0$, dizemos ser $a \bmod n$ o resto da divisão: $\frac{a}{n}$. Por exemplo:

$$15 \bmod 6 = 3 \text{ e } -15 \bmod 9 = 3.$$

Leitura: $a \bmod n$ é dito "a módulo n".

Dois números a e b são ditos congruentes módulo n quando: $a \bmod n = b \bmod n$ e representamos como: $x \equiv b \pmod{n}$. Por exemplo: $12 \equiv 7 \pmod{5}$.

Propriedades de congruência

1. Reflexixa: $a \equiv a \pmod{n}$.
2. Simétrica: $a \equiv b \pmod{n}$, então: $b \equiv a \pmod{n}$.
3. Transitiva: $a \equiv b \pmod{n}$ e $b \equiv c \pmod{n}$, então: $a \equiv c \pmod{n}$.

Vê-se, portanto, que a congruência define uma relação de equivalência , já que atende as propriedades reflexiva, simétrica e transitiva.

2.3.2 Algoritmo de Euclides Estendido

O inverso multiplicativo a^{-1} de um número a em módulo n é o valor que satisfaz a expressão:

$$aa^{-1} \bmod n = 1 \tag{2.2}$$

Exemplo: $12^{-1} \bmod 5 = 3$.

Para o cálculo do inverso multiplicativo será utilizado o *Algoritmo de Euclides Estendido*.

O *Algoritmo de Euclides Estendido* calcula o máximo divisor comum (MDC ou $\gcd$ - *greater commom divisor*) entre dois números inteiros a e b, além de dois outros valores x e y que satisfaçam a expresão: $ax + by = d$. Segue descrição retirada de (**??**):

Algoritmo: *Algoritmo de Euclides Estendido*

ENTRADA: dois inteiros não negativos a e b com $a \geq b$.
SAÍDA: $d = \gcd(a, b)$ e inteiros x e y que satisfaçam a $ax + by = d$.

1. Se $b = 0$ então definir $d \leftarrow a$, $x \leftarrow 1$, $y \leftarrow 0$, e retorna(d,x,y).
2. Defina $x_2 \leftarrow 1$, $x_1 \leftarrow 0$, $y_2 \leftarrow 0$, $y_1 \leftarrow 1$.
3. Enquanto $b > 0$ faça o seguinte:
   (a) $q \leftarrow \lfloor \frac{a}{b} \rfloor$, $r \leftarrow a - qb$, $x \leftarrow x_2 - qx_1$, $y \leftarrow y_2 - qy_1$.
   (b) $a \leftarrow b$, $b \leftarrow r$, $x_2 \leftarrow x_1$, $x_1 \leftarrow x$, $y_2 \leftarrow y_1$ e $y_1 \leftarrow y$.
4. Defina $d \leftarrow a$, $x \leftarrow x_2$, $y \leftarrow y_2$, e retorna(d,x,y).

$\lfloor x \rfloor$: representa o maior valor inteiro menor ou igual a x.

A obtenção do inverso multiplicativo faz-se utilizando o seguinte algoritmo, retirado de (**??**):

Algoritmo: *Calculando inversos multiplicativos em* $\mathbb{Z}_n$

ENTRADA: $a \in \mathbb{Z}_n$
SAÍDA: $a^{-1} \bmod n$, provado que existe

1. Usar o Algoritmo de Euclides Estendido para encontrar os inteiros x e y, tal que $ax+by = d$, em que $d = \gcd(a, n)$

2. Se $d > 1$, então $a^{-1} \bmod n$ não existe. Caso contrário, retorna(x).

2.3.3 Função totiente de Euler

A função totiente, também conhecida como função phi é representada por $\phi()$. Essa função tem como definição (??):

$$\phi(n) = \{k \in \mathbb{Z} | 1 \leq k \leq n, mdc(k, n) = 1\} \tag{2.3}$$

Em que $\phi(n)$ é a quantidades de números relativamente primos com n existentes entre 1 e n. Entende-se por números relativamente primos aqueles que compartilham apenas um termo em comum, o 1 (??).

Exemplos:

- $\phi(1) = \phi(2) = 1$
- $\phi(3) = \phi(4) = \phi(6) = 2$
- $\phi(5) = 4$

Com isso: Se n é primo, então: $\phi(n) = n - 1$ Se $n = pq$ em que p e q são primos, então: $\phi(n) = (p - 1)(q - 1)$

2.3.4 Redução de Montgomery

É um método apresentado pelo matemático Peter Montgomery (??) em 1985 para realizar operações em aritmética modular com números muito grandes. A redução de Montgomery é utilizada principalmente para o cálculo de exponenciação e multiplicação modulares de grandes números rapidamente.

O método utiliza um passo que transforma os números em questão em um valor que será utilizado para as operações em aritmética modular. Com esses valores na forma reduzida, as funções convergem mais rapidamente para o resultado esperado.

A redução ocorre da seguinte maneira:
Dado um valor T ao qual se deseja aplicar a redução de Montgomery, dizemos que a forma reduzida de T é:

Redução Montgomery de $T = MRed(T) = TR^{-1} \pmod m$

Em que:

- R e T são inteiros
- $R > m$
- $gcd(m, R) = 1$
- $0 \leq T < mR$
- R^{-1} é o inverso multiplicativo de R

A seguir é apresentado o algoritmos de redução de Montgomery utilizado como base conceitual neste trabalho, retirados de (**??**).

Para cálculo de $MRed(T)$ é executado o seguinte algoritmo:

Algoritmo: Redução Montgomery

ENTRADA: inteiros $m = (m_{n-1}...m_1m_0)_b$ com $\gcd(m, b) = 1$, $R = b^n$, $m' = -m^{-1} \bmod b$, e $T = j(t_{2n-1}...t_1t2)_b < mR$.
SAÍDA: $TR^{-1} \bmod m$.

1. $A \leftarrow T$. (Notação: $A = (a_{2n-1}...a_1a_0)_b$.)
2. Para i de 0 a $(n-1)$ fazer o seguinte:
   (a) $u_i \leftarrow a_i m' \bmod b$.
   (b) $A \leftarrow A + u_i m b^i$
3. $A \leftarrow \dfrac{A}{b^n}$.
4. Se $A > m$ então $A \leftarrow A - m$.
5. Retorna(A).

Com a redução em mãos, pode-se executar os algoritmos de multiplicação e exponenciação modular.

2.3.5 Exponenciação Modular

Para o cálculo de exponenciação modular, podemos aplicar o seguinte algoritmo (Retirado de (??)):

Algoritmo: Exponenciação Montgomery

ENTRADA: $m = (m_{l-1}...m_1m_0)_b$, $R = b^l$, $m' = -m^{-1} \bmod b$, $e = (e_t...e_0)_2$ com $e_t = 1$, e um inteiro x, $1 \leq x \leq m$.
SAÍDA: $x^e \bmod m$.

1. $\tilde{x} \leftarrow Mont(x, R^2 \bmod m)$, $A \leftarrow R \bmod m$. ($R \bmod m$ e $R^2 \bmod m$ devem ser fornecidos como entradas).
2. Para i de t a 0 fazer o seguinte:
   (a) $A \leftarrow Mont(A, A)$
   (b) Se $e_i = 1$ então $A \leftarrow Mont(A, \tilde{x})$.
3. $A \leftarrow Mont(A, 1)$.
4. Retorna(A).

2.3.6 Multiplicação Modular

Para multiplicação, utilizando a redução de Montgomery, podemos aplicar o seguinte algoritmo (Retirado de (??)):

Algoritmo: Multiplicação Montgomery

ENTRADA: inteiros $m = (m_{n-1}...m_1m_0)_b$, $x = (x_{n-1}...x_1x_0)_b$, $y = (y_{n-1}...y_1y_0)_b$ com $0 \leq x, y < m$, $R = b^n$ com $\gcd(m, b) = 1$, e $m' = -m^{-1} \bmod b$.
SAÍDA: $xyR^{-1} \bmod m$.

1. $A \leftarrow 0$. (Notação: $A = (a_{2n-1}...a_1a_0)_b$.)
2. Para i de 0 a $(n-1)$ fazer o seguinte:
   (a) $u_i \leftarrow (a_0 + x_iy_0)m' \bmod b$.

(b) $A \leftarrow \frac{(A + x_i y + u_i m)}{b}$

3. Se $A \geq m$ então $A \leftarrow A - m$.

4. Retorna(A).

2.4 Criptografia

O termo criptografia tem origem nas palavras gregas "kryptós" (oculto) e "gráphein" (escrever) (??). Trata-se do uso de técnicas que visam transformar um texto claro em um texto ilegível. Sendo que somente pessoas com posse de informações devidas possam ter acesso ao conteúdo da mensagem original.

Para a transformação das mensagens são utilizados algoritmos criptográficos que permitem tais conversões. Dois termos utilizados para descrever os processos de transformação de mensagens são: decriptar ou decifrar e encriptar ou cifrar. São elas:

- Encriptar é o ato de transformar um texto claro em um texto ilegível, através da utilização de uma função e uma chave criptográfica.

- Decriptar é o ato reverso de encriptar, ou seja, transformar um texto ilegível em um texto claro.

A figura 2.1 abaixo retrata essas funções.

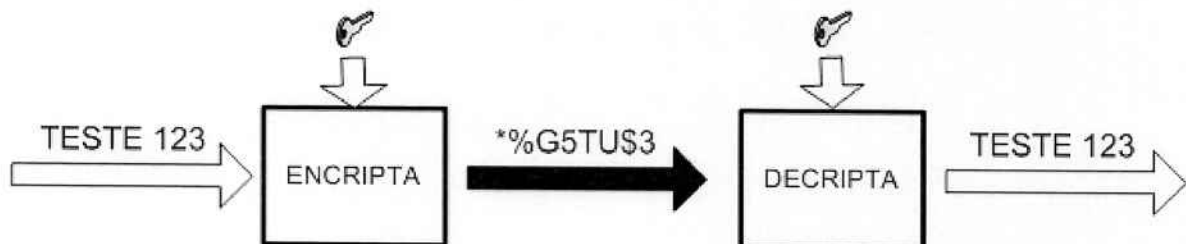

Figura 2.1: Exemplo de encriptação e decriptação

Algoritmos de transformação de mensagens muito explorados atualmente são os baseados em chaves criptográficas. Nesse modelo os possuidores das devidas chaves podem decodificar o texto cifrado. Duas técnicas desse tipo de manipulação de dados são: Algoritmos de Chave Simétrica e Assimétrica.

2.5 Algoritmo de Chave Simétrica

Essa técnica explora o uso de uma mesma chave para encriptar e decriptar as mensagens. Com isso, se duas pessoas tiverem um segredo compartilhado, uma chave que ambos conhecem, é possível utilizá-la para encriptar/decriptar as mensagens.

Esse algoritmo também é conhecido como algoritmo de chave privada e se opõe ao de chave pública, também conhecido como assimétrico que será explicado futuramente.

Algoritmos conhecidos de criptografia simétrica são: AES (*Advanced Encryption Standard*) (**??**), DES (*Data Encryption Algorithm*), *TripleDES* (**??**), entre outros.

São dois os tipos de cifras simétricas existentes, são eles: cifras de fluxo e de bloco.

2.5.1 Cifras de Fluxo

O algoritmo se baseia na expansão da chave criptográfica para uma sequência de *bits* do tamanho da mensagem. A geração do texto cifrado é feita através da operação XOR entre a mensagem a ser encriptada e a chave expandida. O mesmo ocorre para decriptação, gera-se do lado do destinatário a mesma chave expandida e através da operação XOR, obtém-se o texto claro.

Um famoso algoritmo de cifra de fluxo é o RC4 (Rivest Cipher 4) (**??**).

2.5.2 Cifra de Blocos

Esse algoritmo trabalha com tamanhos fixos de blocos de mensagens. Cada bloco é encriptado e decriptado com a mesma chave criptográfica. Para mensagens de tamanho maior que o do bloco são executadas funções chamadas de modos de operação (**??**). Alguns modos de operação são: CBC, ECB, XTS, entre outros.

Alguns dos algoritmos de cifra de blocos são: AES, DES e TripleDES.

2.6 Algoritmo de Chave Assimétrica

Essa técnica foi desenvolvida e proposta por Diffie e Hellman em 1970 (**??**), e atualmente são vários os pesquisadores que a exploram.

Algoritmo de chave assimétrica, também chamado de algoritmo de chave pública, é um

método criptográfico em que existe um par de chaves, sendo uma chave pública e outra privada. A primeira é de conhecimento geral, ou seja, distribuída livremente. Já a segunda é conhecida apenas pelo seu proprietário e não deve ser divulgada. A geração das duas chaves é feita simultaneamente sendo que uma das chaves depende da outra.

Com as chaves em mãos é possível encriptar e decriptar uma mensagem. Para encriptar uma mensagem é utilizada a chave pública do destinatário da mensagem. A mensagem encriptada só poderá ser decifrada pelo destinatário com sua própria chave privada. Por exemplo:

Se Alice quiser enviar uma mensagem para seu amigo Balls, o primeiro deverá saber a chave pública do segundo. Assim que o Balls receber a mensagem cifrada, ele poderá decifrá-la utilizando sua chave privada.

Essa classe de algoritmo é utilizada para gerar assinaturas digitais, que serão explicadas mais adiante.

Cifras assimétricas conhecidas são: RSA (**??**), Rabin (**??**) e ElGamal (**??**).

2.6.1 O Algoritmo RSA

O RSA é o algoritmo criptográfico baseado em criptografia de chave pública desenvolvido por Rivest, Shamir e Adlerman em 1977 e publicado pela primeira vez em 1978 (**??**). Foi o primeiro algoritmo completo a permitir cifração e assinatura. A geração de chaves através desse algoritmo, faz-se da seguinte maneira, retirado de (**??**):

Algoritmo: Geração do par de chaves do RSA

ENTRADA: parâmetro seguro l

SAÍDA: chave pública (e,n) e chave privada d.

1. Escolher aleatoriamente dois primos p e q de $\frac{l}{2}$ bits de comprimento.
2. Calcular $n \leftarrow pq$ e $\phi(n) \leftarrow (p - 1)(q - 1)$.
3. Escolher um inteiro $e \in [1,\phi(n)]$ tal que $\gcd(e, \phi(n)) = 1$
4. Calcular um inteiro d tal que $d \in [1,\phi(n)]$ e $ed \equiv 1 \pmod{\phi(n)}$

Para encriptação de uma mensagem, retirado de (**??**):

Algoritmo: Encriptação básica do RSA

ENTRADA: chave pública (e,n) e texto claro $m \in$ [0, n - 1].
SAÍDA: texto encriptado c.

1. Calcular $c \leftarrow m^e \bmod n$.
2. Retorna c.

Para decriptação, retirado de (**??**):

Algoritmo: Decriptação básica do RSA

ENTRADA: chave pública (e,n), chave privada d e texto encriptado c.
SAÍDA: texto claro m.

1. Calcular $m \leftarrow c^e \bmod n$.
2. Calcular Retorna m.

2.7 Funções HASH

Funções de *hash*, ou funções de resumo, são aquelas que mapeiam uma mensagem de tamanho qualquer em um valor de *hash* de tamanho fixo. Estas mensagens resumidas são geralmente anexadas às mensagens originais para permitir a detecção de possíveis alterações.

Algumas propriedades fundamentais dessas funções, segundo (**??**):

- Resistência à primeira inversão: Tendo um resumo R, é inviável encontrar uma mensagem que tenha o mesmo resumo R.
- Resistência à segunda inversão: Tendo um resumo R e uma mensagem M, é inviável encontrar uma outra mensagem M_1 que também tenha resumo R.
- Resitência a colisões: É inviável encontrar duas mensagens com o mesmo resumo R.

Alguns dos algoritmos de hash são: família MD 2 , 4 e 5, família SHA (0 , 1 e 2) e Whirpool (**??**).

2.7.1 SHA - 1

A função SHA(*Secure Hash Algorithm*)-1 assim como sua família SHA, foi desenvolvida pela NSA(*National Security Agency*). O SHA-1 e é uma função de resumo que foi considerada a sucessora do MD5.

O SHA-1 recebe uma mensagem de tamanho qualquer e mapeia para uma mensagem de tamanho de 160 $bits$.

Ataques recentes em 2005 (**??**) descobriram algumas fraquezas nessa técnica.

2.8 Autoridade Certificadora

Autoridades certificadoras (AC ou CA - *Certification Authority*) são entidades confiáveis que tem como finalidade atestar a identidade de entidades finais, para tanto utiliza-se da emissão de Certificados Digitais.

O Certificado Digital é um documento eletrônico que contém informações sobre a entidade certificada, como, por exemplo, dados gerais (nome, endereço, telefone, outros) além da chave pública. Esse certificado é assinado pela Autoridade Certificadora.

Aplicação: caso Alice queira enviar uma mensagem para seu amigo Balls, é necessário que a primeira tenha a chave pública do segundo e deve se certificar se a chave é autêntica, ou seja, que realmente pertence a Balls. Para tanto, Alice de posse do Certificado Digital recebido, verifica a autenticidade da assinatura do certificado junto à Autoridade Certificadora. Se o certificado realmente for o gerado pela AC, a chave pública de fato pertence a Balls.

2.9 Assinatura Digital

A Assinatura Digital é um método de autenticação de informação, ou seja, através dela é possível garantir quem foi o emissor de tal assinatura e, portanto, ter uma prova inegável de quem assinou tal conteúdo.

Alguns dos algoritmos de Assinatura Digital conhecidos são: DSA, assinatura utilizando encriptação RSA e assinaturas BLS (**??**). Um projeto que baseou-se na utilização da assinatura digital é o "Assinatura Digital para o Selo Verde", desenvolvido para certificar computadores adquiridos pela Universidade de São Paulo livres de algumas substâncias tóxicas. Maiores detalhes em (**??**).

Para gerar a assinatura utilizando o algoritmo de RSA utiliza-se o seguinte algoritmo, retirado de (**??**):

Algoritmo: Assinatura básica do RSA

ENTRADA: chave pública (e,n), chave privada d e texto claro m.
SAÍDA: assinatura de m.

1. Calcular $s \leftarrow h(m)^d \bmod n$, onde $h(m)$ é o resumo da mensagem m.
2. Retorna s.

A figura 3.2 ilustra o cálculo da assinatura.

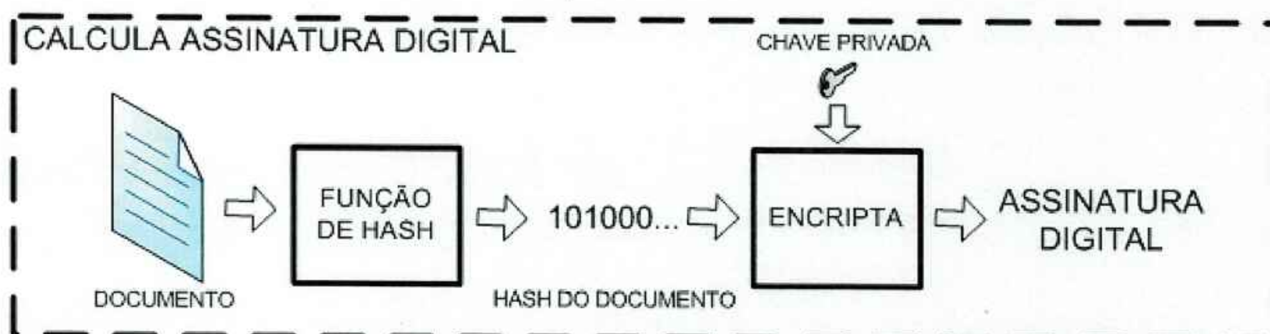

Figura 2.2: Cálculo de Assinatura Digital

A verificação da assinatura pode ser feita como segue, retirado de (**??**):

Algoritmo: Verificação Assinatura básica do RSA

ENTRADA: chave pública (e,n), chave privada d, texto claro m e assinatura s.
SAÍDA: verificação da assinatura s.

1. Calcular $h_1 \leftarrow h(m)$, onde $h(m)$ é o resumo da mensagem m.
2. Calcular $h_2 \leftarrow h(m)^e \bmod n$.
3. Retorna "$verdadeiro$" se $h_1 = h_2$, caso contrário Retorna "$falso$".

A figura 2.3 ilustra a verificação da assinatura.

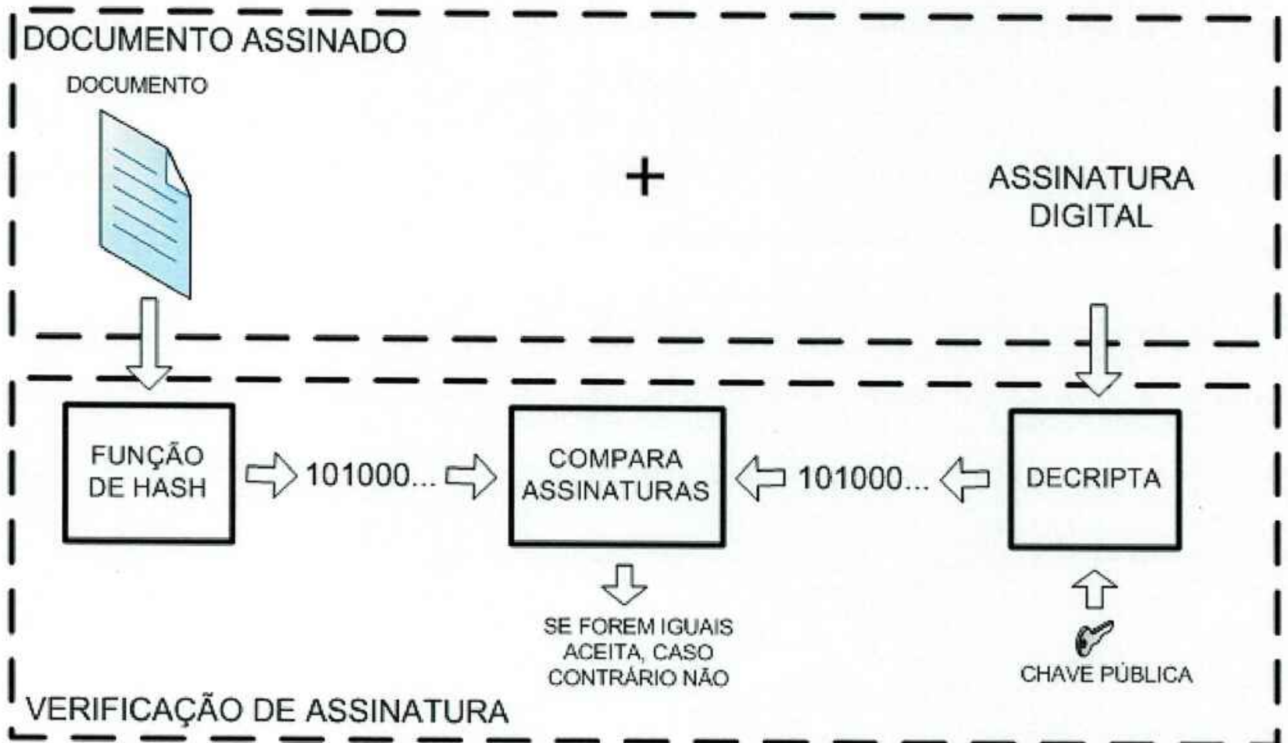

Figura 2.3: Verificação de Assinatura Digital

2.10 Ataques

Esta seção lista alguns dos ataques possíveis a segurança e a *hardwares* criptográficos, que um sistema pode sofrer.

2.10.1 Ataques de Segurança

Stallings (2006) classifica os ataques de segurança em quatro classes:

2.10.1.1 Ataque de Fabricação

O atacante nesse tipo de ataque visa inserir mensagens falsas no sistema. Pretende, assim, fazer com que o destinatário acredite que a mensagem teve origem no remetente confiável.

Algumas técnicas de ataque dessa classe segundo (**??**): *masquerade*, *replay*, *IP spoofing*, baseado em senha, baseado na exploração do acesso confiável.

2.10.1.2 Ataque de Interceptação

O atacante consegue acessar informações que trafegam pelo canal de comunicação, sem o consentimento ou conhecimento das vítimas.

Algumas técnicas de ataque dessa classe segundo (??): análise de tráfego, divulgação do conteúdo das mensagens e *snooping*.

Um ataque específico de interesse para esse texto é o ataque *man-in-the-middle*.

2.10.1.3 Ataque man-in-the-middle

Esse ataque ocorre quando o agressor tem o controle sobre o meio de comunicação. Segundo Stallings (2006) o ataque ocorre da seguinte maneira:

Alice quer conversar com Balls e pretende criar uma chave secreta que apenas os dois conhecem. Existe porém um invasor, Ernesto, que controla o meio de comunicação.

- Alice gera o par de chaves pública e privada e transmite uma mensagem para o Balls com a chave pública.
- Ernesto intercepta a mensagem. Gera seu próprio par de chaves e transmite a sua chave pública para o Balls.
- Balls gera uma chave secreta com a chave de Ernesto (acreditando ser a de Alice) e a sua. Transmite.
- Ernesto novamente intercepta a mensagem, e tem uma chave secreta para comunicação com o Balls.
- Ernesto cria uma chave secreta com a chave da Alice e de Ernesto e envia para Alice.

Após esses passos, Alice acredita que está se comunicando diretamente com Balls e vice-versa. Na realidade, todas as mensagens cifradas são interceptadas por Ernesto.

2.10.1.4 Ataque de Interrupção

Ocorre uma interferência no sistema de forma que a mensagem não chegue ao destinatário.

Algumas técnicas de ataque dessa classe segundo (??): DoS (*Denial of Service*) e DDoS (*Distributed Denial of Service*).

2.10.1.5 Ataque de Modificação

Ocorre uma mudança na mensagem por parte do atacante e essa é enviada ao destinatário como se fosse a mensagem original.

2.10.2 Ataques por Canais secundários

(??) listou alguns dos ataques secundários possíveis contra sistemas criptográficos. Aqui são descritos alguns desses métodos que permitem ataques para obtenção da chave secreta de uma comunicação em um sistema.

2.10.2.1 Análise simples de potência

Essa análise é também conhecida simplesmente como SPA (*Simple Power Analysis*). Baseado no rastro de consumo de potência (variação do consumo de potência) de um sistema, pode-se descobrir sua chave criptográfica. Para tanto o agressor necessita conhecer o algoritmo utilizado.

É possível verificar em cada passo do algoritmo, quais os instantes em que são realizadas determinadas funções matemáticas, e assim, através do rastro de potência determinar cada bit da chave. (??)

2.10.2.2 Análise simples e diferencial de campos eletromagnéticos

Sistema eletrônicos emanam radiações assim que seus dispositivos estão sobre um diferencial de tensão. Isso é suficiente para que, através da análise das emanações eletromagnéticas, atacantes possam coletar e analisar dados de forma a descobrir a chave criptográfica utilizada. Alguns dos métodos possíveis para esse tipo de análise são: Análise Diferencial de Ondas Eletromagnéticas (DEMA *ou Differencial EletroMagnetic Analysis*) ou Análise Simples de Ondas Eletromagnéticas (SEMA - *Simple EletroMagnetic Analysis*). (??)

2.10.2.3 Análise de tempo

Pode-se explorar a característica de que o tempo gasto no processamento de uma função é relacionado diretamente com os operandos. Através de uma análise do algortimo que em funcionamento e do tempo em cada parcela, pode-se utilizar técnicas para obtenção da chave criptográfica. (??)

2.10.2.4 Análise de falhas

Outra técnica explorada é a de geração de falhas durante o processamento de um algoritmo em um dispositivo embarcado, de forma que a análise das saídas possibilite recuperar a chave secreta. (??)

2.11 Dispositivos FPGA

O coprocessador criptográfico foi desenvolvido com o uso de um dispositivo FPGA - *Field Programable Gate Array* ou Arranjo de Portas Programável em Campo. O uso deste tipo de dispositivo no desenvolvimento de sistemas de *hardware* é um dos métodos mais utilizados atualmente pela indústria eletrônica. Nesta seção, é apresentada uma sucinta descrição do funcionamento e da arquitetura de um FPGA.

2.11.1 Funcionamento e Arquitetura de um FPGA

Dispositivos FPGA são circuitos integrados projetados para serem configurados por seus usuários (projetistas e desenvolvedores de hardware). São formados por uma grande quantidade de células lógicas ou blocos lógicos e inteconexões configuráveis. Cada célula pode ser configurada para executar uma função lógica e se interconectar a outras células conforme a necessidade do projetista. Dessa forma, circuitos lógicos de diferentes tamanhos e complexidades podem ser configurados em um FPGA.

A descrição do comportamento de um FPGA é feita em Linguagem de Descrição de Hardware (HDL - *Hardware Description Language*), ou em um diagrama lógico. Esta descrição é processada e gera um arquivo binário utilizado para configurar o dispositivo. O uso de diagramas lógicos facilita a visualização e entendimento do comportamento desejado enquanto o uso de Linguagens de Descrição de Hardware simplifica e flexibiliza o desenvolvimento.

As linguagens mais utilizadas são a VHDL -*VHSIC Hardware Description Language*; VH-SIC: *Very High Speed Integrated Circuit*) - e a Verilog.

As células e blocos de um FPGA são classificadas como :

- **Blocos Lógicos Configuráveis (CLB-Configurable Logic Block)** - Compostos por tabelas que descrevem uma função lógica (LUT - *Look Up Tables*) e por *flip flops* que registram estas saídas. As LUTs funcionam como células de memória que armazenam o resultado esperado (1 bit). Cada possível valor de entrada de uma LUT endereça

uma destas células. Usualmente são utilizadas tabelas com entradas de 4 bits. Como as células de memória de uma LUT são voláteis, um FPGA precisa ser reconfigurado sempre que é energizado. Para isso, estes dispositivos costumam ser montados juntamente com uma memória não volátil que armazena a configuração do FPGA e automaticamente a transfere quando o sistema é energizado.

- **Blocos de Entrada e Saída (IOB-Input Output Block)** - Estes blocos fazem a conexão com os pinos físicos do FPGA. Podem configurar um pino como sendo um sinal de saída, entrada, ou entrada e saída (e.g. quando conectado a um barramento). Permite que os níveis lógicos internos sejam convertidos em sinais elétricos adequados ao exterior da FPGA com valores adequados de tensão e corrente permitindo também que um sinal seja colocado em alta impedância quando operando em *tri-state*. Os IOBs podem também conter algum mecanismo de proteção para que o interior do FPGA não seja danificado quando um sinal de entrada estiver fora dos limites elétricos ideais.
- **Matriz de Interconexão** - A conexão entre os blocos lógicos configuráveis e os blocos de entrada e saída é feita através de uma matriz de conexão. Trata-se simplificadamente de uma rede de trilhas que possuem chaveamentos em seus cruzamentos permitindo que diferentes rotas sejam criadas para conectar os CLBs e IOBs.

A conversão de uma descrição de *hardware* (em HDL ou em diagramas lógicos) em um arquivo binário usado para configurar os CLBs, IOBs e a matriz de interconexão é feita por *softwares* poderosos do tipo CAD-*Computer Aided Design*- em geral fornecidos pelas empresas que fabricam dispositivos FPGA ou por empresas independentes especializadas neste tipo de software.

Tal conversão constuma seguir as seguintes etapas:

1. **Tradução** - Os comandos HDL ou os diagramas lógicos são traduzidos para blocos de circuito lógico seguindo padrões pré-definidos pelo programa de síntese.
2. **Otimização** - Os blocos padronizados são analisados com o intuito de otimizar o circuito sintetizado segundo critérios estabelecidos pelo projetista.
3. **Mapeamento** - O circuito lógico obtido é mapeado nos componentes lógicos básicos disponíveis na tecnologia escolhida para a síntese. Os blocos lógicos do circuito são mapeados nos blocos típicos da FPGA do componente escolhido para a síntese.
4. **Posicionamento** - Os blocos lógicos da FPGA identificados na etapa anterior são posicionados dentro daqueles disponíveis na FPGA escolhida para síntese.

5. **Roteamento** - As interligações entre os blocos lógicos posicionados na etapa anterior são interconectados compondo o circuito final sintetizado.

3 ESPECIFICAÇÃO DO PROJETO

3.1 Descrição Do Projeto

O projeto do coprocessador criptográfico consiste de uma parte em *hardware* e outra em *software*. O *software* tem por objetivo testar, apresentar e sincronizar o que foi implementado em *hardware*.

O coprocessador é capaz de encriptar e decriptar dados utilizando o algoritmo de criptografia assimétrico RSA, e de assinar digitalmente utilizando o mesmo algoritmo juntamente com a função de hash SHA-1. Ele se comunica com PCs utilizando comunicação USB 2.0, sendo capaz de receber dados e devolvê-los criptografados, assinados ou apenas com o hash calculado.

O *hardware* foi implementado na FPGA XEM3005 da empresa Opal Kelly, cuja escolha se deu principalmente à existência de um módulo USB, proporcionando a facilidade de implementação da comunicação entre o coprocessador e o computador. O kit da placa XEM3005 veio acompanhado do *software* FrontPanel que é o responsável por facilitar a comunicação USB do lado do PC - e de diversas APIs para desenvolvimento dessa comunicação nos mais variados ambientes e linguagens (Java, C++, Python, LabVIEW, etc).

O *hardware* consiste de dois módulos principais: o módulo USB, responsável pela comunicação através da interface USB do computador; e o módulo RSA, responsável pelas funções criptográficas. Os dois módulos se comunicam através de *buffers* e sinais de controle. Ambos também possuem uma modularização interna, descrita no item seguinte.

Existem dois *softwares* no projeto, sendo um de testes e outro de demonstração. O de testes foi totalmente implementado em Java e consiste de uma interface gráfica, que invoca métodos que utilizam algumas funções de uma biblioteca criptográfica e um módulo para comunicação USB, que se utiliza da API Java do FrontPanel. O *software* de demonstração também foi implementado em Java e consiste de uma interface gráfica para envio de *email* acompanhadas da Assinatura Digital da mensagem enviada. A assinatura digital é calculada

utilizando o dispositivo conectado à USB. Além disso foi desenvolvido um complemento para o navegador Mozilla Firefox integrado ao gerenciador de *email* Gmail. Este complemento reconhece a assinatura digital dos *e-mails* automaticamente se ela existir.

3.2 Arquitetura Do Sistema

A arquitetura do sistema é dividida em duas partes, *software* e *hardware*. Na figura 3.1 temos uma visão da arquitetura criada.

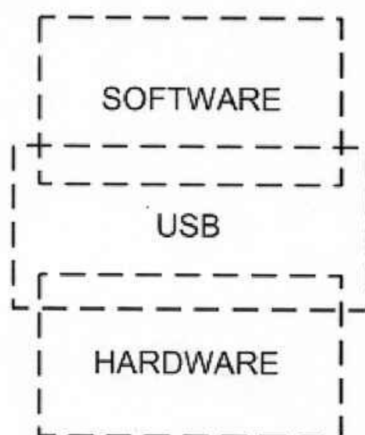

Figura 3.1: Arquitetura Geral do Projeto

A camada de *software* está dividida em três parte partes: interface, funções criptográficas e acesso ao coprocessador. A parte de *hardware* está dividida em seis blocos VHDL: Interface USB, RSA, Unidade de Controle, Exponenciador Modular, Multiplicador Modular e SHA-1. Este blocos serão detalhados mais a frente neste documento. A comunicação entre *software* e *hardware* é feita via USB.

Para simplificar o desenvolvimento da comunicação via USB, utilizamos os elementos em VHDL e a API em Java fornecidos pela Opal Kelly, que encapsulam todo o protocolo de comunicação USB. Desta forma, não foi necessário implementar este protocolo de comunicação.

3.3 Software

Para se ter acesso ao *hardware* criptográfico, transmissão e recebimento de dados através de um computador é necessário ter uma interface de comunicação via *software* para realizar as funções.

Para realização da comunicação foram desenvolvidos alguns tipos de programas, cada um com uma função específica. Quatro tipos foram construídos:

- *Software de Teste*, que é uma interface que permite ao usuário executar todas as funções disponíveis em *hardware* através de mensagens e valores relacionados com o algoritmo criptográfico, comparar o resultado da execução em hardware com a mesma operação em *software* e obter algumas métricas de desempenho;
- *Software de Demonstração* é uma interface que permite apresentar uma funcionalidade que demonstre como o sistema desenvolvido pode ser utilizado de forma integrada a outros já existentes;
- *Driver USB* realiza e controla a interação computador - coprocessador, além do sincronismo entre os dois dispositivos;
- *Software Com Funções Criptográficas* que são as funcionalidades responsáveis por receber informações vindas de todas as outras partes e manipulá-las, assim como fazer a integração com o *hardware*. Nessa parte do sistema encontram-se funções criptográficas que dão suporte ao funcionamento do sistema como um todo, ou seja, funções criptográficas que foram implementadas em software ao invés de hardware, tais como geração de chaves e função de *hash* para teste e comparação dos valores gerados pelo coprocessador.

Os programas interagem como representado pela figura 3.2.

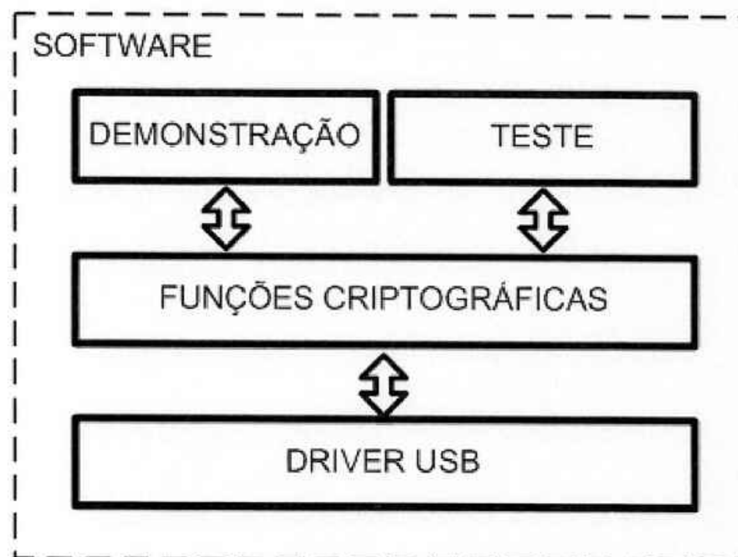

Figura 3.2: Arquitetura do Software

A implementação desse sistema foi baseada no modelo de três camadas (apresentação, negócio e dados)(**??**), sendo que a última, foi adaptada e substituída pelo coprocessador.

Através do *Diagrama de Classes* da figura 3.3, mostra-se o conjunto de classes necessárias para implementação desse projeto, assim como as respectivas interações.

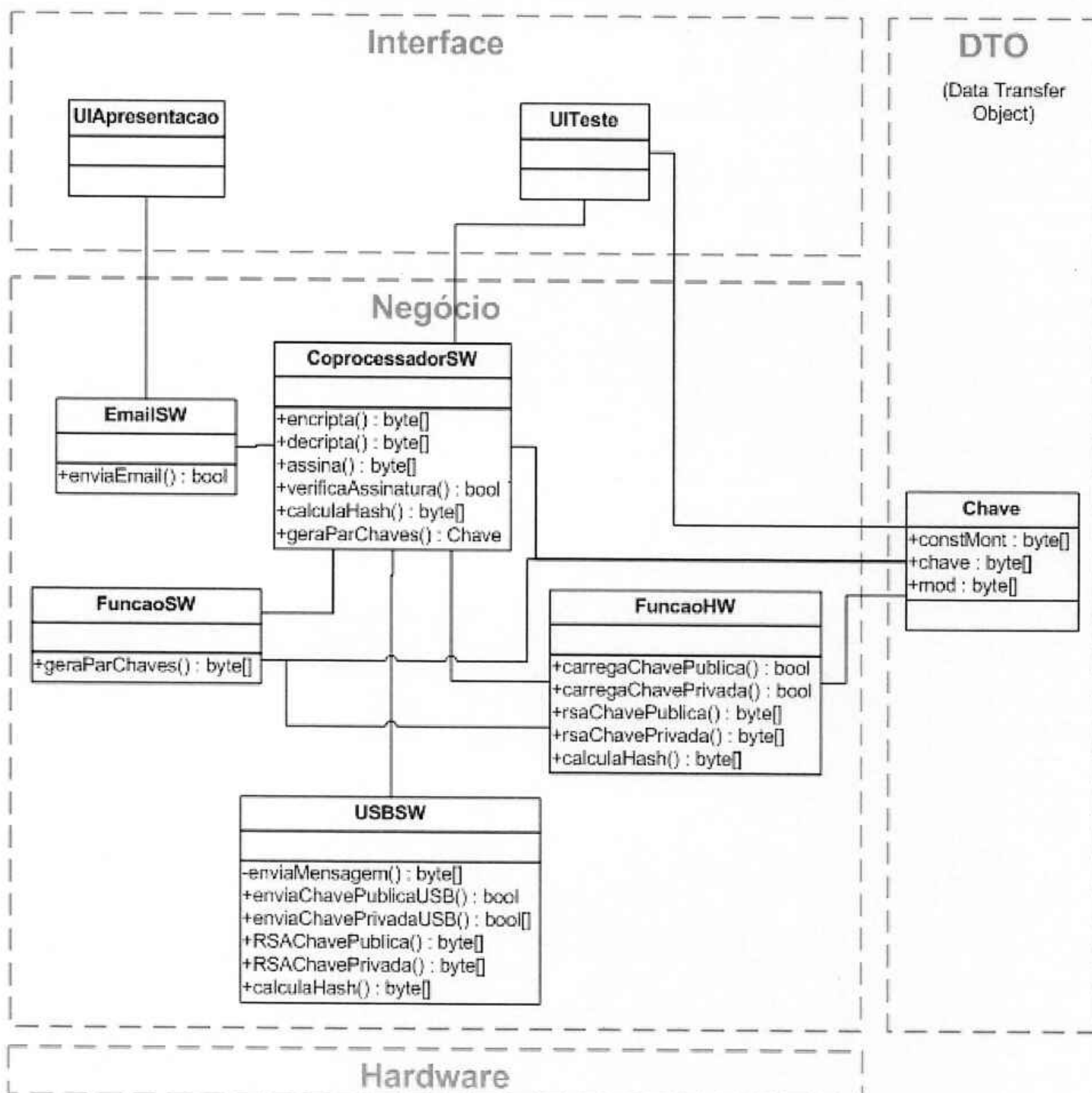

Figura 3.3: Diagrama de Classes

Abaixo são especificados cada um dos diferentes softwares.

3.3.1 Software de Teste

O Software de Teste é a camada mais alta do projeto desenvolvido. É constituído por uma interface gráfica simples que permite verificar o resultado de todas as funcionalidades do coprocessador criptográfico e das camadas inferiores do software elaborado.

Todas as descrições dos elementos da interface gráfica podem ser vistas no Apêndice I

3.3.2 Software De Demonstração

O programa de demonstração explora as funcionalidades do *hardware* criptográfico, através de uma aplicação prática.

O sistema será demonstrado através de uma aplicação que faz o envio e verificação de *e-mails* assinados digitalmente. No computador A, será criada uma mensagem assinada pelo coprocessador e enviada para um computador B. O computador B recebe a mensagem e, conhecendo o remetente, verifica se a Assinatura Digital confere com o que foi recebido, aceitando-a ou não.

O software de demonstração consiste de dois sistemas, um no computador A, responsável pela geração da assinatura e envio do *email* assinado e no computador B a verificação da assinatura do *email*.

A figura 3.6 representa a estrutura da demonstração.

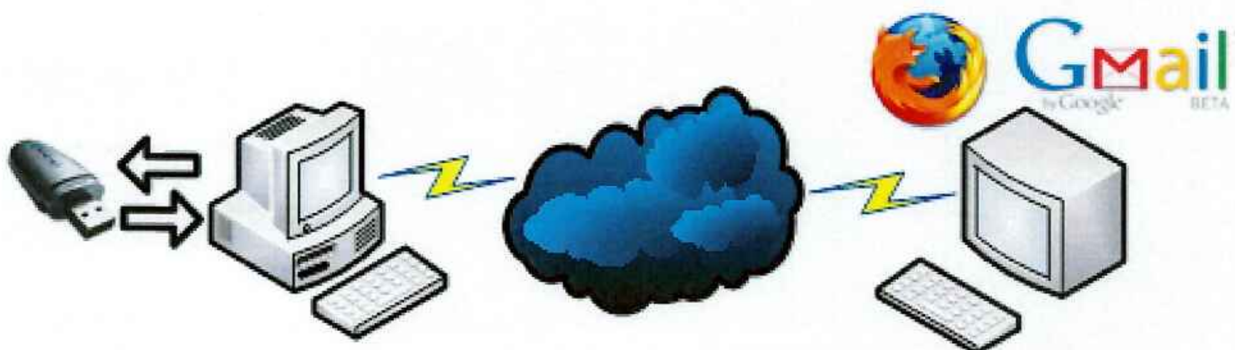

Figura 3.4: Demostração: Envio de mensagens eletrônicas assinadas

O sistema presente no computador A é a interface para preenchimento e envio do *email*, através da qual o usuário completa os campos de destino, corpo da mensagem e assunto. O sistema exige também que seja escolhido um par de chaves, assim é possível enviar o email assinado para o destinatário. A criação do par de chaves criptográficas é feita utilizando o software de demonstração, em que uma nova aba solicita o nome e *email* do assinador, além de uma data de quando esse par de chaves irá expirar.

No computador B, há um complemento do navegador Mozilla Firefox desenvolvido para reconhecer Assinaturas Digitais dentro da mensagem segundo padrão definido(vide *Formato do Bloco de Mensagem*). O complemento funcionará em conjunto com o site de emails Gmail. Assim que se abre um email, o complemento verifica se a mensagem foi gerada pelo nosso programa de teste e caso afirmativo verifica se a assinatura digital está correta.

3.3.2.1 Formato do Bloco de Mensagem

Para transferência de emails com assinatura, foi definido um padrão sobre como deve ser cada mensagem. A mensagem assinada deve respeitar o padrão visto na figura 3.5. Os dados da assinatura são apresentados na base 64 com o padrão MIME (??).

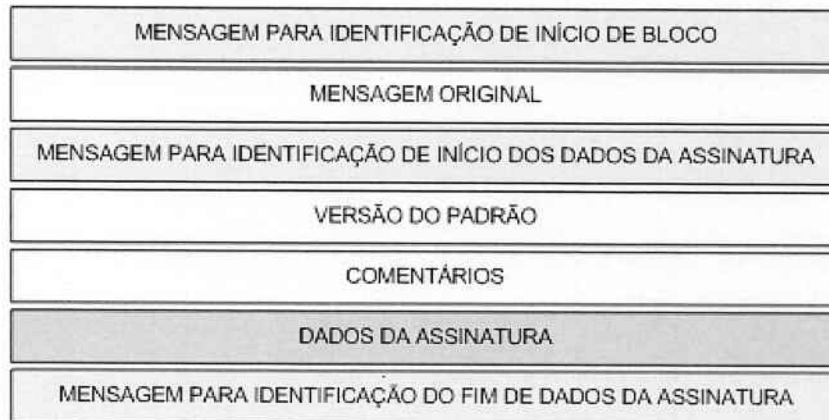

Figura 3.5: Padrão do Bloco de Assinatura

O bloco dados da assinatura é composto por:

1. *Email* do remetente.
2. Nome do remetente.
3. *Timestamp* da assinatura.
4. Módulo n da chave pública do remetente.
5. Expoente da chave pública do remetente.
6. Algoritmo criptográfico, nesse caso, sempre RSA.
7. Função de *hash* utilizada, nesse caso, sempre SHA-1.
8. Assinatura Digital.
9. Verificador de Redundância Cíclica, CRC (??), dos dados da assinatura.

3.3.2.2 Verificação de Assinatura Digital

No computador B será utilizado para verificação da assinatura um complemento para o navegador Firefox, que verifica as assinaturas digitais. Esse complemento foi desenvolvido para

reconhecer padrões em *emails* do *site* Gmail. Os padrões são blocos que contêm a mensagem e a assinatura.

Será identificado na própria interface do Gmail quando uma mensagem possui uma assinatura correta ou não.

Duas funções criptográficas são necessárias para realizar a verificação de assinatura, cálculo de RSA e resumo (SHA-1).

3.3.2.3 Geração de Assinatura Digital

Para explorar e demonstrar as funcionalidades do hardware criptográfico criou-se um programa que envia *emails*. Essa aplicação utilizar o coprocessador para assinar as mensagens que deseja transmitir.

O fluxo de informação da mensagem partindo da interface de demonstração até o seu envio via *email*, segue as seguintes etapas, demonstradas pela figura 3.6:

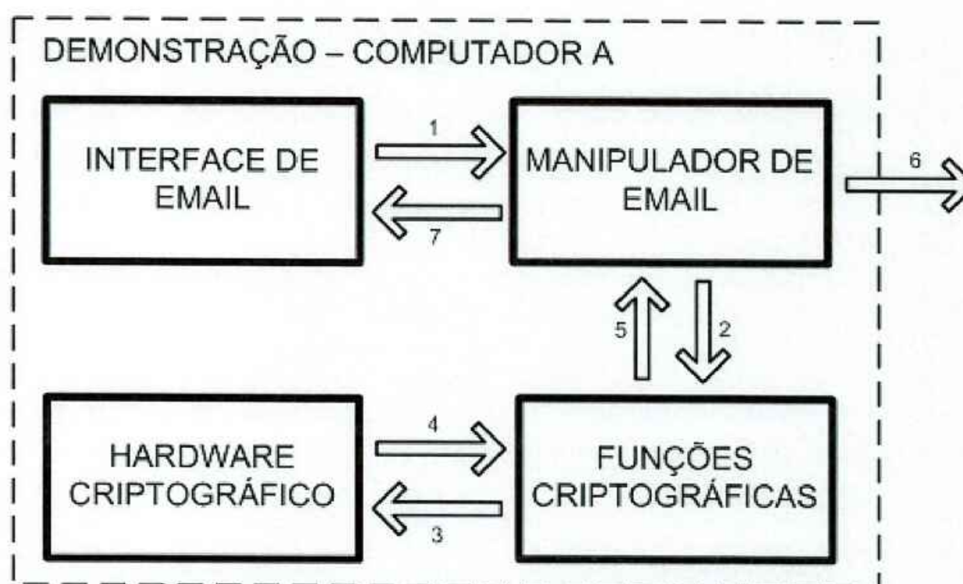

Figura 3.6: Fluxo de Informação da Demonstração

Estará disponível para o usuário definir os seguintes campos na *Interface de Email*: endereço eletrônico do destinatário, assunto, corpo da mensagem e par de chaves criptográficas para serem utilizadas na assinatura. Será possível para o usuário gerar as chaves criptográficas que deseja utilizar para assinar.

No segundo bloco, *Manipulador de Dados*, irá tratar os dados vindos da interface de envio de *email* e solicitar a geração de assinatura digital. Após tais funções, será enviada a mensagem assinada via correio eletrônico.

O terceiro bloco é responsável por chamar as funções criptográficas necessárias, além de formatar as mensagens para transferir para o coprocessador.

O último, *Hardware Criptográfico*, realiza as funções criptográficas solicitadas.

Analisando a figura 3.6, tem-se o que cada passo executa:

1. Transferência de todos os dados coletados na interface (Email do Destinatário, Assunto e Corpo do *Email* e Par de Chaves Criptográficas)
2. Chamada de funções criptográficas com os parâmetros definidos.
3. Solicitação ao *hardware* dos cálculos das funções.
4. Devolução dos resultados obtidos.
5. Devolução da assinatura digital.
6. Envio do *email*, com o bloco definido contendo assinatura, chave do emissor e mensagem original.
7. Devolução de parâmetros de sucesso.

3.3.3 Software Com Funções Criptográficas

Essa camada de software será responsável gerenciar o acesso às funções criptográficas implementadas em *hardware* e a algumas em *software*. Parte das funções foram desenvolvidas em *software*, pois na fase inicial houve limitações no escopo do projeto para que o desenvolvimento pudesse ser possível dentro do prazo.

As funções dessa camada do projeto são responsáveis também por interfacear a camada de visualização com a camada de hardware, como visto pela figura 2.3.

Essa parte do projeto foi dividida em algumas classes que podem ser vistas no Apêndice II

3.3.4 Driver USB

O módulo Driver USB foi criado com o intuito de coordenar todas as atividades relacionadas com a comunicação USB. Ele é responsável por:

- Estabelecer a comunicação do lado do PC com o coprocessador através de uma porta USB;

- Receber dados a serem enviados para o coprocessador, formatá-los de acordo com um padrão estabelecido para depois enviá-los;
- Receber dados do coprocessador em um formato estabelecido, interpretá-los de acordo com seu cabeçalho e encaminhá-los para o devido tratamento;

Ele é composto por diversas funções relacionadas com a comunicação USB, formatação e interpretação de dados. Essas funções podem ser divididas em: as que fazem as requisições para o coprocessador; as que auxiliam nas requisições para o coprocessador; as que realizam a comunicação com o coprocessador; auxiliares; e de inicialização. Detalhes dessas funções podem ser vistas no Apêndice III.

3.4 Hardware

Esta seção contém a especificação do hardware desenvolvido. A figura 3.12 (final do capítulo) exibe todos os elementos da organização de hardware definida. Cada um destes elementos é descrito a seguir.

3.4.1 Interface USB

A interface USB é responsável por estabelecer a comunicação entre a FPGA e um PC usando uma porta USB e por formatar os dados recebidos e enviados de acordo com um formato estabelecido. Este bloco tem como entradas os pinos de comunicação com os circuitos de interface USB do kit de desenvolvimento utilizado e os pinos de dados e de sinalização vindos do bloco RSA. As saídas deste bloco são os pinos de dados e sinalização enviados ao bloco RSA. A Interface USB também é responsável por gerar o sinal de *Clock* para sincronia do sistema. Este sinal é gerado a partir do *Clock* de sincronia da porta USB.

Na figura 3.7 podemos ver quais as entradas e saídas deste bloco. No Apêndice IV é encontrada a descrição de cada sinal.

Internamente, a Interface USB faz uso de alguns elementos disponibilizados pela Opal Kelly. São eles:

- **okHostInterface** - Este elemento faz a conexão com a interface física da USB e com os *endpoints* internos.

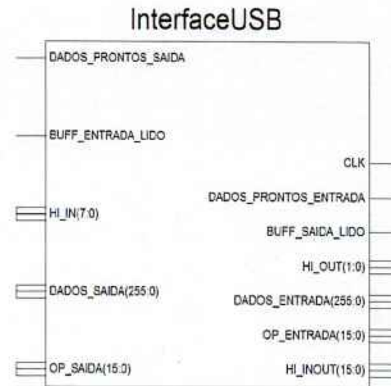

Figura 3.7: Entradas e saídas do bloco Interface USB

- **okPipeIn** - Este é um dos *endpoints* disponíveis que implementa um *pipeline* com os dados recebidos pela USB.
- **okPipeOut** - Este é um dos *endpoints* disponíveis que implementa um *pipeline* com os dados a serem enviados pela USB.

Esses elementos estão detalhados no Apêndice V.

3.4.2 Bloco RSA

O bloco RSA é a parte do *hardware* responsável por executar as operações criptográficas. A figura IV.2 apresenta as entradas e saídas deste bloco.

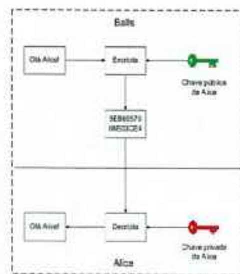

Figura 3.8: Entradas e saídas do bloco RSA

Detalhes sobre o bloco RSA podem ser encontrados no Apêndice VII.

3.4.3 Arquitetura do bloco RSA

O bloco RSA faz uso dos seguintes componentes:

- **Multiplicador Modular** - Implementa o algoritmo de redução de Montgomery para executar os cálculos de multiplicação modular utilizados pelo Exponenciador Modular.
- **Exponenciador Modular** - Executa os cálculos de exponenciação modular do algoritmo de criptografia RSA.
- **Sha1** - Executa a função de hash Sha1 utilizada na operação de assinatura.

O funcionamento destes componentes é descrito a seguir.

3.4.4 Multiplicador Modular

Este componente executa o cálculo da redução de Montgomery entre dois números, A e B em relação ao módulo M. O resultado é a multiplicação reduzida $A \times B \times R^{-1} modM$, onde $R = 2^{256}$. O cálculo é feito através do algoritmo de Multiplicação Modular de Montgomery apresentado no item 2.3 deste documento. A figura 3.9 mostra as entradas e saídas deste bloco.

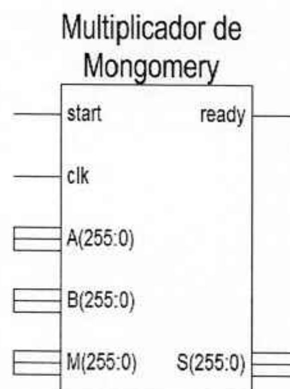

Figura 3.9: Entradas e saídas do bloco Multiplicador Modular

A descrição de cada sinal pode ser vista no Apêndice VIII.

3.4.5 Exponenciador Modular

Este componente executa o cálculo de exponenciação modular entre dois números, P e E com relação ao módulo M - $P^E modM$. A exponenciação é feita através do algoritmo de Exponenciação Modular apresentado no item 2.3 deste documento. A constante C $= 2^{2n} modM$ deve ser pré-calculada externamente e utilizada como um dos parâmetros de entrada do exponenciador. As operações de multiplicação modular utilizam o Multiplicador Modular apresentado. A figura 3.10 mostra as entradas e saídas deste bloco.

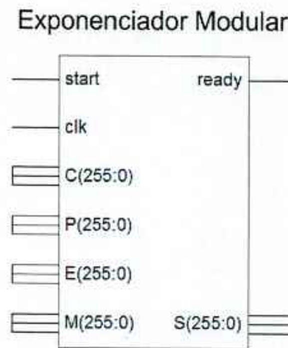

Figura 3.10: Entradas e saídas do bloco Exponenciador Modular

A descrição de cada sinal pode ser vista no Apêndice IX.

3.4.6 Bloco HASH (SHA-1)

Durante o planejamento do projeto foi decidido por não implementar a função de *hash* em *hardware*, pois isso demandaria muito tempo, e o projeto não seria acabado no tempo previsto. Para solucionar tal problema, foi decidido por procurar um solução já existente e adicioná-la ao projeto.

Uma implementação de Paul Hartke (phartke@stanford.edu) do SHA-1 existente, é disponibilizada para uso no portal OpenCores.org (??). Essa versão foi utilizada como parte do coprocessador. A figura 3.11 demonstra as entradas e saídas deste componente.

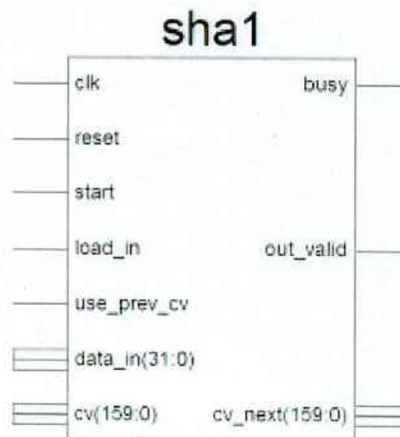

Figura 3.11: Função de *Hash*: entradas e saídas

Detalhes dos sinais desse bloco podem ser vistos no Apêndice X.

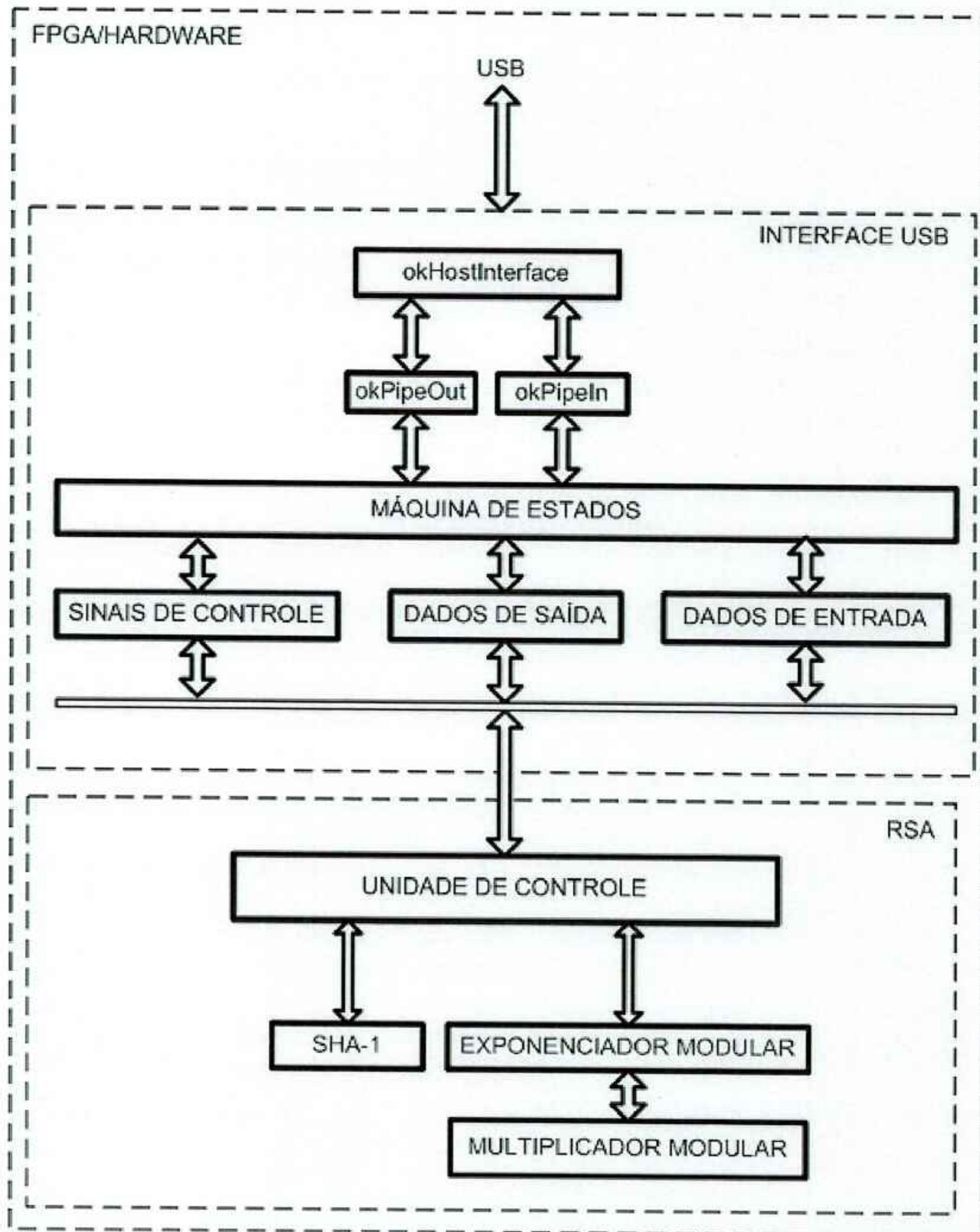

Figura 3.12: Organização do Hardware

4 FERRAMENTAS ESCOLHIDAS

Nesse capítulo são apresentadas as ferramentas de desenvolvimento de *software* e *hardware* utilizadas durante o projeto, os motivos que justificam as escolhas feitas e de que forma cada uma delas contribuiu para o projeto.

4.1 Hardware - Kit De Desenvolvimento

Para o desenvolvimento do hardware foi adquirido um *kit* de desenvolvimento para FPGA fabricado pela empresa Opal Kelly. Trata-se de uma empresa localizada em Portland - Oregon nos EUA, especializada na fabricação de módulos para desenvolvimento de hardware com dispositivos FPGA.

Os módulos são compostos por um dispositivo FPGA, um circuito de comunicação USB e alguns periféricos de entrada e saída conectados ao FPGA. Os periféricos disponibilizados nos módulos fornecidos, além da comunicação USB, incluem LEDs, circuito de geração de sinal de *clock*, memória SDRAM, memória PROM para armazenar o arquivo de configuração do FPGA, conectores de expansão e leitor de JTAG.

O dispositivos FPGA que a Opal Kelly utiliza em seus módulos são os fabricados pela empresa Xilinx, que desenvolve e fabrica componentes FPGA em várias linhas de produtos como a Virtex, a Spartan e a Micro Blaze.

A escolha da Opal Kelly como fornecedora do kit do desenvolvimento utilizado no projeto teve como principal motivo a facilidade em utilizar a comunicação via USB 2.0. Os módulos fabricados possuem uma arquitetura que permite a utilização da porta USB, tanto como meio de configuração do FPGA, como para uso em tempo de execução para troca de dados entre um PC e o hardware em execução. A comunicação USB pode ser feita através de uma API disponibilizada em algumas linguagens de alto nível típicas de mercado ou por meio de um *software* da própria Opal Kelly denominado FrontPanel (disponível para quem adquire algum dos módulos). Este *software* permite a criação de IHM (interface homem-máquina)

Característica	XEM3050	XEM3010	XEM3005	XEM3001
Modelo de FPGA	Spartan-3 XC3S4000-5	Spartan-3 XC3S1500/XC3S1000	Spartan-3E XC3S1200E	Spartan-3 XC3S400
Quantidade de CLBs	6912	3328/1920	2168	896
Quantidade de Slices (CLBs x 4)	27648	13312/7680	8672	3584
Integração com Front Panel e API USB?	SIM	SIM	SIM	SIM

Tabela 4.1: Características consideradas na escolha do modelo de kit de desenvolvimento de hardware

em software que permite acessar diretamente alguns sinais internos do hardware sintetizado e em execução na FPGA. Desta forma, é possível estabelecer troca de dados entre um PC e o *hardware* via USB sem a necessidade de que seja feita implementação própria do protocolo USB, reduzindo substancialmente o tempo e a complexidade do projeto.

4.1.1 XEM3005

A Opal Kelly disponibiliza módulos com diferentes capacidades e recursos. Para a decisão de qual módulo seria utilizada no projeto, foi feito um dimensionamento levando em conta a quantidade de recursos utilizada em implementações semelhantes. Uma implementação apresentada em(**??**) utilizou 10369 *slices* de um FPGA da XIlinx, modelo Virtex. Sabendo desse valor, a decisão da placa foi feita baseando-se no cruzamento das especificações da família XEM da Opal Kelly e das características de cada modelo de FPGA utilizado em cada módulo. As características principais consideradas são exibidas na tabela 4.1.

Os modelos XEM3005 com 8672 *slices* e XEM3010 com 13312 slices se mostraram suficientes para os objetivos deste projeto. Considerando o custo de cada um, o modelo XEM3005 foi escolhido.

Funcionalmente, a placa XEM3005 possui as estrutura em blocos mostrada na figura 4.1. A imagem representa a FPGA Spartan3E interligada à memória PROM, à memória SDRAM, ao conversor USB, e às vias de I/O configuradas pelas chaves JP3 e JP4, além da interligação do conversor de USB com ao bloco PLL responsável pela geração de sinais de *clock*.

4.1.2 Front Panel

Um dos principais motivos para o uso da XEM3005 nesse projeto foi a possibilidade de ter uma aplicação em software sendo executada em um computador que se comunica diretamente

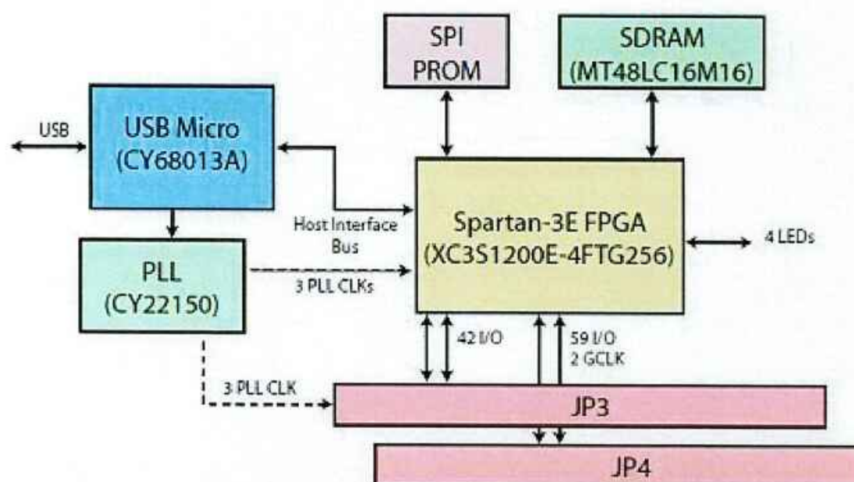

Figura 4.1: Estrutura de blocos da XEM 3005

com a aplicação em hardware na placa. Isso possibilita um maior controle e flexibilidade sobre o projeto realizado em descrição de hardware, além de reduzir o tempo e complexidade do projeto levando em conta a dificuldade de se configurar a comunicação USB manualmente mesmo através de uma linguagem de alto nível.

Primeiramente, o FrontPanel possui a funcionalidade de fazer a gravação do código objeto da aplicação no dispositivo FPGA através da USB por meio de uma interface gráfica na qual se procura o arquivo .bit nos diretórios do computador. Adicionalmente, o FrontPanel permite que sejam feitos testes e *debug* da aplicação no FPGA através de um painel virtual que pode conter LED's, *pushbuttons*, caixas de texto, *labels*, *displays* de 7 segmentos, chaves lógicas, e outros elementos de representação de sinais e valores numéricos. O formato desse painel é personalizável. Após carregar o código objeto na placa, carrega-se o painel que é descrito por meio de um arquivo XML. Tal arquivo possui a estrutura dos componentes da interface e seus atributos. Através desse arquivo é possível configurar o tamanho do painel, a posição e tamanho dos elementos com os valores descritos em número de pixels e demais características referentes a cada componente. A figura 4.2 mostra o exemplo de um painel virtual criado com o FrontPanel. Os dados apresentados no painel são obtidos em tempo de execução diretamente de estruturas descritas em VHDL e sintetizadas na FPGA.

4.1.3 API De Linguagem De Alto Nível

Além do software FrontPanel, a Opall Kelly fornece com a placa XEM 3005 uma dll e a biblioteca *okjFrontPanel* para desenvolvimento de aplicações em Java. Essa biblioteca disponibiliza

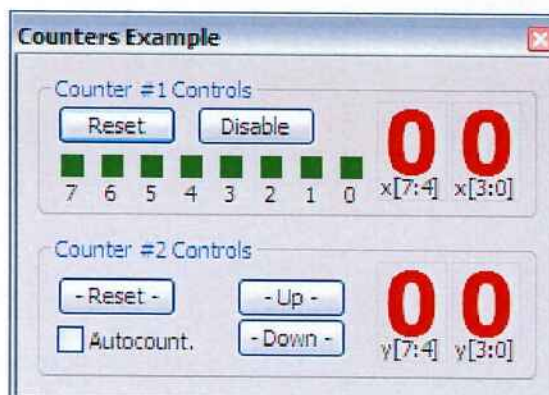

Figura 4.2: Exemplo de painel virtual criado com o FrontPanel

uma API de desenvolvimento com a qual é possível implementar e executar aplicações que utilizam a comunicação do computador com o *hardware* sintetizado através da porta USB. Essa API também apresenta versões em C, Python e MatLab. Devido aos conhecimentos prévios da equipe, a linguagem escolhida para desenvolvimento da aplicação foi Java. Com a biblioteca okjFrontPanel é possível utilizar métodos e classes prontos facilitando a codificação da gravação do código objeto na placa, envio e recebimento de dados do PC com a placa, configuração de parâmetros de execução como frequência de clock, versões da placa utilizada (a API dá suporte para toda a linha XEM), além de permitir maior flexibilidade para a aplicação, ao permitir utilizar outros componentes presentes nos ambientes Java.

Para a comunicação da placa com a USB o FrontPanel utiliza três estruturas que representam sinais lógicos, sendo elas: *wire*, *pipe* e *trigger*. A partir delas foi possível determinar o modo de troca de bits entre placa e PC.

4.2 Software De Desenvolvimento

4.2.1 ISE

O ISE é uma IDE de desenvolvimento da Xilinx através do qual é possível editar, compilar, depurar, simular e sintetizar descrições de hardware em VHDL e Verilog. Dentre os tipos de descrições o ISE permite: descrições comportamentais, descrições estruturais (descrição dos componentes) e descrição baseada no desenho do circuito lógico nos dispositivos lógico programáveis da Xilinx.

O ISE oferece suporte a todas as linhas de FPGAs da Xilinx inclusive o Spartan3E utilizado no projeto. Através deste software é possível obter todas as informações relevantes da FPGA durante a implementação e a síntese. Esta ferramenta disponibiliza recursos como:

- Relatórios informativo contendo os resultados das etapas do processo de síntese como a quantidade de recursos utilizados em uma síntese;
- Sugestões de otimização das descrições HDL feitas;
- Verificação automática de sintaxe;
- Console de relatório de erros e avisos;
- Editor de texto (HDL) e de desenho (Schematic);
- Paleta de componentes padronizados como portas lógicas, decodificadores, multiplexadores, registradores e contadores;
- Editor de diagramas de tempo para a análise de resultados de simulação
- *Wizards* (janelas de configuração) para criação de alguns componentes otimizados para o FPGA em uso como registradores e blocos de memória.

4.2.2 ModelSim

O ModelSim é um simulador de descrições de hardware baseadas em texto (HDL), sem os recursos gráficos e de síntese dedicada do ISE, mas em compensação permite que após concluída uma implementação em hardware, seja possível fazer a simulação com os inputs de dados baseados em arquivos HDL de maneira que os arquivos de simulação descrevam outros componentes de hardware ou outros dispositivos, tornando a simulação mais prática e realista do que uma simples waveform de sinais. O principal arquivo de teste é denominado test fixture, e pode ser escrito tanto em VHDL e Verilog, independente da linguagem utilizada pela implementação do hardware. No caso do projeto, o ModelSim foi utilizado para simular toda a comunicação USB uma vez que inúmeras sínteses poderiam danificar tanto FPGA quanto qualquer outro componente da placa XEM3005, então através do ModelSim foi possível entender como funcionavam cada uma das três estruturas de comuniação via USB do FrontPanel (*wire*, *pipe* e *trigger*), podendo variar alguns de seus parâmetros para analisar qual seria o impacto do uso de cada uma das estruturas dentro do projeto e quanto a resultados foi possível verificar se um conjunto de bytes pode ser enviado para a implementação em hardware do RSA, ser corretamente cifrado e o valor devolvido para a USB, sem utilizar em nenhum momento a placa e sem ter que realizar nenhuma síntese.

4.3 Linguagens

No projeto foram utilizadas linguagens de *software* e de descrição de *hardware*, esta seção descreve sucintamente essas linguagens e o motivo da adoção de cada uma.

4.3.1 Java

Linguagem em alto nível orientada a objetos desenvolvida pela empresa Sun Microsystems. Atualmente é umas das linguagens com mais desenvolvedores no mundo, e vasta em frameworks (APIs) que oferecem ao desenvolvedor maior facilidade e versatilidade para a codificação de uma aplicação.

Na camada do projeto de software de criptografia foram utilizadas algumas APIs para fornecer as funcionalidades desejadas a essa camada, a biblioteca Java security permitiu a criação de chaves e outros parâmetros relevantes a criptografia RSA.

As classes desenvolvidas em Java além de produzirem esses parâmetros criptográficos invocam métodos responsáveis por enviar os dados para a USB e receber os dados dessa porta.

4.3.2 VHDL

VHDL (*Very high speed integrated circuits Hardware Description Language*) é uma linguagem de descrição de hardware utilizada para síntese de FPGAs ou dispositivos lógicos programáveis. Foi desenvolvido inicialmente para uso militar e passou a ter padrões definidos pelo IEEE.

Uma descrição em VHDL pode ser estrutural ou comportamental ou de máquina de estados. A estrutural consiste de uma descrição das entradas e saídas de um componente, descrição interna dos sinais e com quais outros componentes este se comunica.

A descrição comportamental é sintaticamente similar ao código gerado por uma linguagem imperativa (e.g. linguagem C), pois as operações desejadas são descritas sequencialmente uma a uma. Porém, diferentemente de uma linguagem imperativa de software, as transições no hardware acontecem simultaneamente e não sequencialmente. A descrição por máquina de estados representa um algoritmo por meio da definição de transições de estados baseadas no estado atual e nas entradas do componente descrito.

Para entrada e saída essa linguagem permite a definição de bits, vias ou vetor de bits, sinais de *clock* (relógio) e eventualmente outros sinais. Na definição dos sinais é possível estabelecer

procedures que contenham laços (while ou for), blocos condicionais (*if*, *else if* e *else*) e blocos de escolha (*switch case*) para tratamento dos sinais. Também é possível importar bibliotecas que contenham componentes prontos como memórias, registradores de tamanhos diversos, multiplexadores, decoders e outros dispositivos lógicos. Durante a edição de esquema lógico as IDEs fazem a construção automática do código em VHDL equivalente à implementação feita, de forma a facilitar a construção do hardware e do entendimento do projeto.

5 METODOLOGIA

Este capítulo apresenta a metodologia empregada no desenvolvimento deste projeto descrevendo as diversas etapas e desafios transpostos, dificuldades encontradas e soluções adotadas.

5.1 Definição do escopo

Inicialmente foi feita a delimitação do escopo do projeto e definição clara dos objetivos a serem atingidos. Algumas necessidades no campo da criptografia foram apresentadas pelo orientador e, aliadas ao interesse do grupo em executar um projeto relacionado a desenvolvimento de hardware, a criação de um coprocessador criptográfico se mostrou um denominador comum entre os diversos interesses apresentados.

A grande motivação inicial deste projeto era o desenvolvimento de um dispositivo USB dotado de processamento criptográfico, um visor e um mini teclado. O usuário deste dispositivo seria capaz de inserir senhas ou outros dados a serem encriptados ou assinados. Dessa forma, tais dados estariam protegidos imediatamente após a sua inserção e antes de serem enviados a alguma sistema potencialmente infectado como computadores ou a própria internet.

Percebeu-se porém que tal projeto demandaria mais tempo do que o disponível para sua execução.

Dessa forma, o escopo foi limitado ao desenvolvimento de um coprocessador criptográfico que pudesse servir como base para a proposta inicial.

Em seguida foi iniciado um processo de especificação dos detalhes do escopo do projeto, quando definiu-se quais funções criptográficas o coprocessador (*hardware*) deveria ser capaz de executar, quais algoritmos seriam utilizados, e como seu funcionamento seria demonstrado.

Decidiu-se que o coprocessador deveria ser capaz de encriptar e decriptar dados além de assiná-los digitalmente. Para tal, foi feita a opção pelo algoritmo RSA para a função de encriptação por ser este um dos algoritmos mais poderosos e amplamente utilizados em todo

o mundo. A necessidade do aprendizado e uso de métodos matemáticos eficientes para cálculo de exponenciação e multiplicação modular rapidamente veio à tona e, com algum estudo inicial e sugestões do orientador e coorientadores, foram escolhidos os algoritmos matemáticos que seriam estudados e que são apresentados no item *Aspectos Conceituais*.

Com o andamento das discussões surgiu a necessidade de estipular um foco principal para o projeto. Decidiu-se que tal foco seria a implementação em *hardware* do algoritmo criptográfico RSA. Assim, para a operação do cálculo de função de *hash*, necessária à geração de assinaturas digitais,optou-se pelo reuso de módulos prontos desenvolvidos em alguma implementação prévia. Uma implementação do algoritmo SHA1 para cálculo de resumos (hash) foi obtida para uso em domínio público e sua integração com o módulo RSA já então estruturado se mostrou como a opção mais prática e natural.

Verificou-se a necessidade de criação de uma aplicação em software que pudesse verificar e demonstrar a correta operação das funcionalidades desenvolvidas para o coprocessador criptográfico. Especificou-se que seriam desenvolvidas duas aplicações em Java, uma para testes funcionais e de desempenho e outra para uma demonstração prática de uma aplicação real do dispositivo.

Decidiu-se que a comunicação entre tal aplicação rodando em um PC e o coprocessador seria através do barramento USB. Caso fossem encontrados muitos problemas para implementação deste, a comunicação utilizaria a porta serial do computador. A facilidade da implementação da comunicação USB foi um dos principais aspectos utilizados na etapa seguinte.

5.2 Escolha de tecnologias e ferramentas

A próxima etapa do projeto foi a escolha de tecnologias e ferramentas para sua implementação, principalmente do dispositivo FPGA. Foram usados como base projetos semelhantes publicados em artigos. Isso permitiu estimar a quantidade de recursos necessária ao kit de desenvolvimento para FPGA para comportar as funções do coprocessador.

Também foram decididos os fabricantes de FPGA que seriam considerados: Altera e Xilinx. A decisão foi tomada pela qualidade das placas e pelos *softwares* de desenvolvimento que as empresas disponibilizam para desenvolvimento.

Inicialmente uma placa mais simples foi adquirida por doação via parceria acadêmica (placa Basys da empresa Digilent) e foi utilizada para testes iniciais e aprendizado no manuseio de FPGAs. Por fim foi adquirido o kit XEM3005 da Opal Kelly com a FPGA Spartan-3E da Xilinx.

A escolha do kit se deveu principalmente ao suporte que o mesmo oferece para a implementação da comunicação USB com o PC, facilitando o trabalho nesse eixo e permitindo que o foco do projeto (funções criptográficas no *hardware*) fosse mantido. Visto que o FPGA em uso é fornecido pela empresa Xilinx, o *software* de desenvolvimento utilizado passou a ser o ISE 10.1 da Xilinx. A linguagem de descrição de *hardware* escolhida foi o VHDL.

5.3 Estudos teóricos e especificação técnica e funcional

Paralelamente ao trabalho de escolha de tecnologias e ferramentas, foi feito um estudo da bibliografia previamente levantada. Foram estudados os principais conceitos de criptografia e desenvolvimento de hardware necessários.

A etapa seguinte foi estudar todos as partes do projeto de *hardware* e a parte de comunicação USB do *software*, como seriam implementados, as tecnologias envolvidas na implementação, e como as partes seriam integradas. Dessa maneira chegou-se a uma arquitetura do *hardware* e a um padrão de mensagens na comunicação.

5.4 Implementação e testes

De posse de uma especificação funcional do coprocessador criptográfico e munidos de conhecimento suficiente, iniciou-se o trabalho de implementação.

5.4.1 Hardware

O desenvolvimento do hardware foi feito de forma incremental. O objetivo era o de obter um hardware que operasse com chaves de 1024 bits. Inicialmente foram desenvolvidos módulos de multiplicação e exponenciação modular de 8 bits. Tais módulos foram estruturados e parametrizados de forma que, partindo deles, o desenvolvimento de módulos que trabalhassem com palavras de dados maiores fosse imediato. De fato, foram gerados módulos operando com chaves criptográficas de 16, 32, 64, 128, 256, 512 e 1024 bits. Todos os módulos foram testados através de simulações utilizando o ISE. As simulações se tornaram cada vez mais lentas a cada novo módulo desenvolvido de forma que a simulação do módulo exponenciador modular de 1024 bits chegou consumir quase uma hora de processamento dedicado e inúmeras tentativas de simulação mal sucedidas por ausência de recursos suficientes no computador utilizado.

Após as simulações s confirmação do correto funcionamento dos módulos multiplicador e exponenciador, iniciou-se a etapa de síntese destes. Rapidamente percebeu-se que os módulos de 1024 bits necessitariam de muito mais recursos do que os disponíveis na XEM3005. Inicialmente acreditamos que seria possível otimizar o uso dos recursos disponíveis com o uso dos blocos de memória disponíveis na Spartan 3E. Após algumas tentativas fracassadas concluímos que os possíveis modos de endereçamento dos blocos de memória não permitiam que fossem compostas palavras de dados com mais de 64 bits e que, mesmo compondo diversos blocos, não era possível utilizar este recurso com o coprocessador. Surgiram então duas alternativas. A primeira foi reestruturar o exponenciador e multiplicador modular, implementando estágios de pipeline de forma a executar os algoritmos sobre pedaços de dados menores de forma iterativa conforme proposto em (**??**). A segunda alternativa foi a de manter a arquitetura já desenvolvida e reduzir o escopo do projeto para implementar encriptação RSA de 256 bits. Embora encriptação RSA de 256 não seja considerada segura, seria suficiente para comprovar o correto funcionamento do nosso sistema. A implementação de versões maiores dependeria portanto apenas da disponibilidade de um FPGA com mais recursos disponíveis.

Paralelamente aos módulos de multiplicação e exponenciação, foi desenvolvido o bloco Interface USB sem grandes dificuldades além do grande nível de detalhes desta comunicação, que atrasou um pouco o andamento do trabalho. Na sequência foi iniciado o desenvolvimento do bloco rsa, com a unidade de controle comandando a execução as diversas operações de carga de valores de chave, encriptação/decriptação e integração com o bloco sha1. Inicialmente foram testadas e implementadas as operações de carga de chaves e encriptação/decriptação. Em seguida, foi feita a integração entre o módulo RSA e o módulo Interface USB. Após consolidada a integração entre os blocos RSA e interface USB e a camada de acesso à USB em software,iniciou-se o trabalho de integração com o bloco SHA-1 e implementação da operação de assinatura.

5.4.2 Comunicação USB

Durante a implementação da interface USB viu-se que não era possível simular essa comunicação com o simulador do ISE, devido a esse fato um novo simulador foi buscado. Os dois ao qual o kit da Opal Kelly tinha suporte (possuía bibliotecas prontas para simular a comunicação USB e outros aspectos da placa) eram o ModelSim XE III 6.1e e o Aldec ActiveHDL 7.1 ou superiores. Foi feita a opção pelo ModelSim por ser mais adequado ao tipo de simulações que se desejava fazer e pela disponibilidade de uma grande quantidade de material de estudo no próprio site da Opal Kelly.

5.4.3 Software

A próxima fase consistiu em implementar o *software* de testes e o *software* de demonstração. Para o *software* de testes foram definidas e implementadas as funções criptográficas necessárias (tanto aquelas usadas para testar o *hardware* totalmente quanto aquelas que não puderam ser inseridas no mesmo), a interface de exibição dos resultados do teste, e a comunicação USB. Para o *software* de demonstração muitas alternativas foram estudadas, e escolheu-se a comparação dos resultados do coprocessador com os resultados de um complemento que desenvolvemos para o navegador Mozilla Firefox. Ambos foram totalmente implementados em Java, com a utilização da API da Opal Kelly no módulo USB.

Finalmente foram definidas métricas de análise de desempenho e foram comparados diversos resultados do coprocessador (*hardware*) com diversas soluções existentes feitas em *software*, incluindo o complemento do Firefox.

6 IMPLEMENTAÇÃO

6.1 Software

São descritas aqui as linguagens, bibliotecas e ferramentas que auxiliaram durante a implementação do projeto. Serão também exibidos os resultados finais dos entregáveis desenvolvidos tanto em *hardware* quanto em *software*.

6.1.1 Interface Gráfica

A Interface Gráfica que exibe todos os valores e comandos relevantes às funções criptográficas foi construída utilizando a IDE NetBeans de desenvolvimento em Java e o *framework* de interface gráfica *Swing*. Ela é constituída de um *frame* ou janela contendo os *labels* ou nomes dos parâmetros a serem calculados pelo hardware e suas correspondentes caixas de texto onde serão exibidos tais valores na base hexadecimal. Há também botões que disparam as funções da classe coproocessador, relativas às funções criptográficas, sendo elas: encriptar mensagem, gerar resumo criptográfico de mensagem (hash), assinar mensagem e gerar par de chaves.

6.1.1.1 Software de Demonstração

Para explorar os aspectos práticos do coprocessador RSA foi desenvolvida uma aplicação baseada na biblioteca gráfica *Swing*. Nesta interface o usuário preenche os campos relativos ao corpo de um *email*, assunto e destinatário e solicita a geração da assinatura digital da mensagem. A geração da assinatura é feita através das funções de *hash* e assinatura com chave privada do coprocessador. A última função dessa interface é a de enviar a mensagem assinada por *email* de acordo com o formato proposto no item 3.3 - Especificação de software, figura 3.5.

A figura 6.1 mostra o programa de envio de emails, desenvolvido para testar a aplicação e a figura 6.2 a interface que permite a criação de chaves criptográficas.

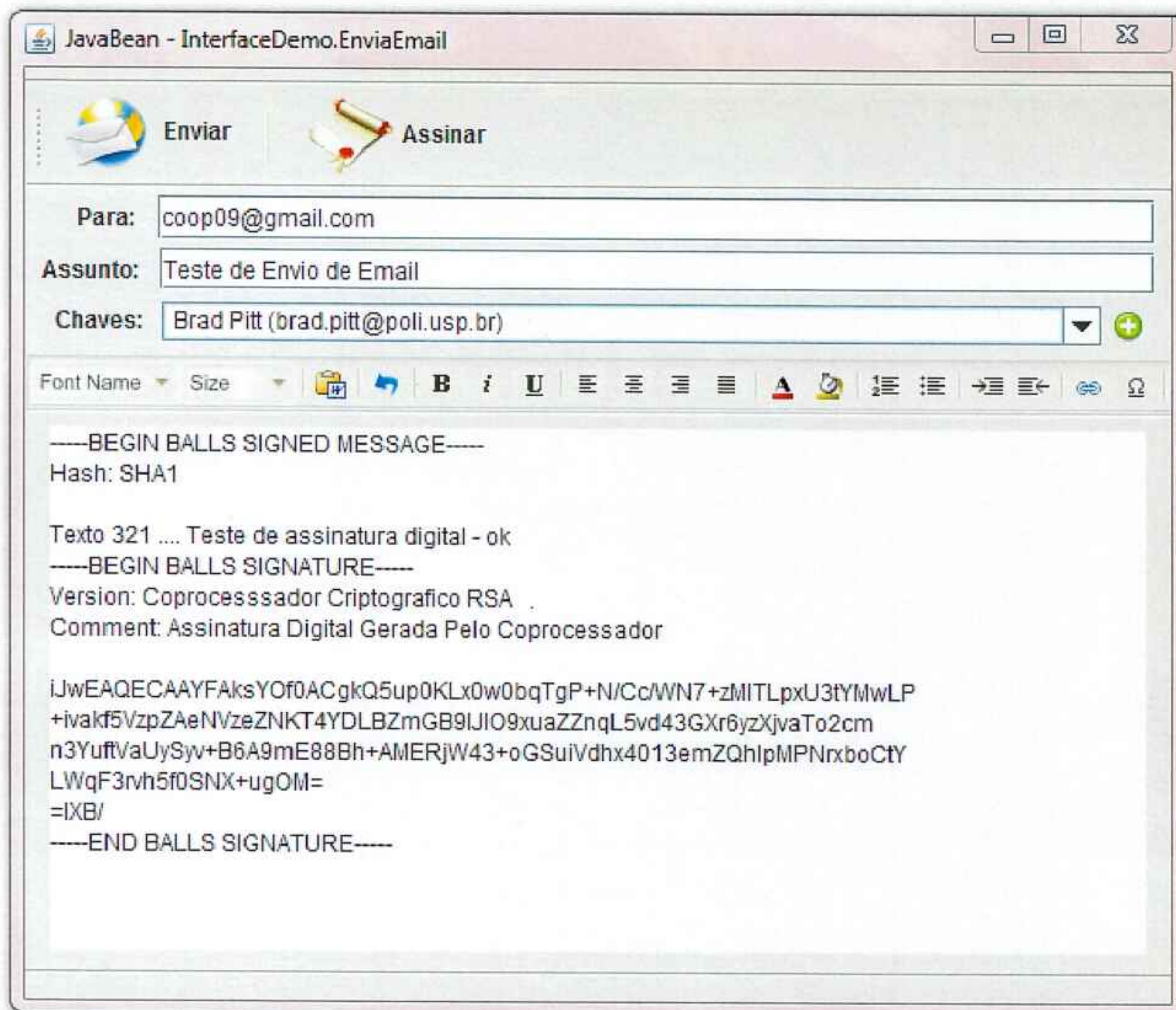

Figura 6.1: Programa para Envio de *email*

Para o recebimento do *email* foi desenvolvido um complemento do navegador Mozilla Firefox para o Gmail que analisa a mensagem recebida, sabendo previamente o formato da mensagem com os campos relevantes à Assinatura Digital e verifica se a chave pública do remetente é válida para essa Assinatura Digital. Dessa forma, o usuário poderá conferir no próprio navegador se a mensagem foi realmente enviada pela pessoa indificada no *email*.

O código detector de erros utilizado para verificar se a mensagem não sofreu nenhuma alteração é o CRC-32 (**??**).

A figura 6.3 demonstra que o complemento do Firefox identificou a assinatura e ela está correta. A figura 6.4 apresenta um caso em que foi identificada uma assinatura incorreta.

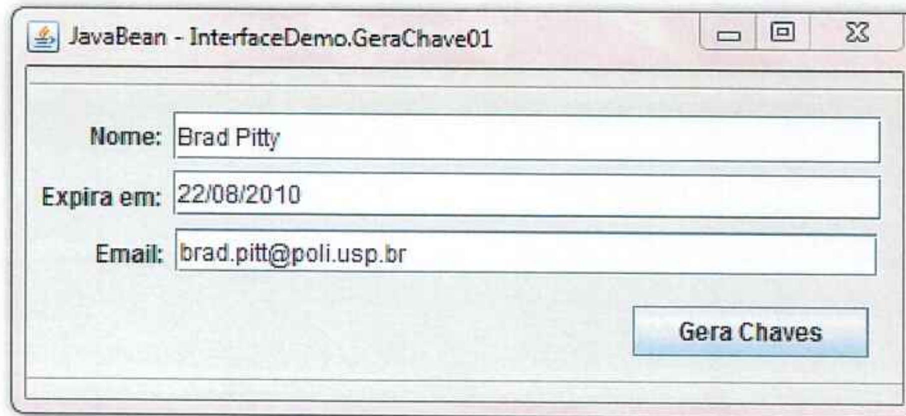

Figura 6.2: Geração do Par de Chaves Criptográficas

6.1.2 Software Com Funções Criptográficas

Para desenvolvimento dessa camada de software foi necessária a utilização de algumas bibliotecas. O envio de *email* utilizou uma das *APIs* do *Java*, o *JavaMail*.

A manipulação de números de precisão estendida foi feita com o auxílio da classe nomeada *BigInteger*. Essa classe também provê métodos para geração de números pseudo-aleatórios e primos, que foram utilizados para criação de chaves criptográficas.

A classe *MessageDigest* foi utilizada para realizar cálculo da funçao de *hash* (SHA-1) para ser possível a comparação com os valores calculados pelo coprocessador. Essa classe é parte do pacote "java.security".

6.1.3 Driver USB

A implementação do módulo *Driver USB* foi toda feita em Java juntamente com os módulos de *Interface Gráfica* e *Software Com Funções Criptográficas*.

Como esse módulo é o responsável pela comunicação USB entre o coprocessador e o PC, a primeira ação realizada foi criar um novo projeto Java e incluir nele a biblioteca da Opal Kelly fornecida no arquivo jar **okjFrontPanel.jar**. Essa biblioteca possui funções prontas para carregamento do arquivo .bit na FPGA via USB, funções para troca de mensagens via porta USB, e funções que recebem informações da própria FPGA sobre a mesma como *ID*, *Serial Number* e versão do *firmware*.

Quando a classe USB.java é instanciada, o construtor é responsável por inicialmente obter algumas informações da FPGA e exibi-las na tela, depois por configurar a FPGA com o arquivo .bit gerado pelo processo de síntese, e então verificar se nesse estado a FPGA suporta corre-

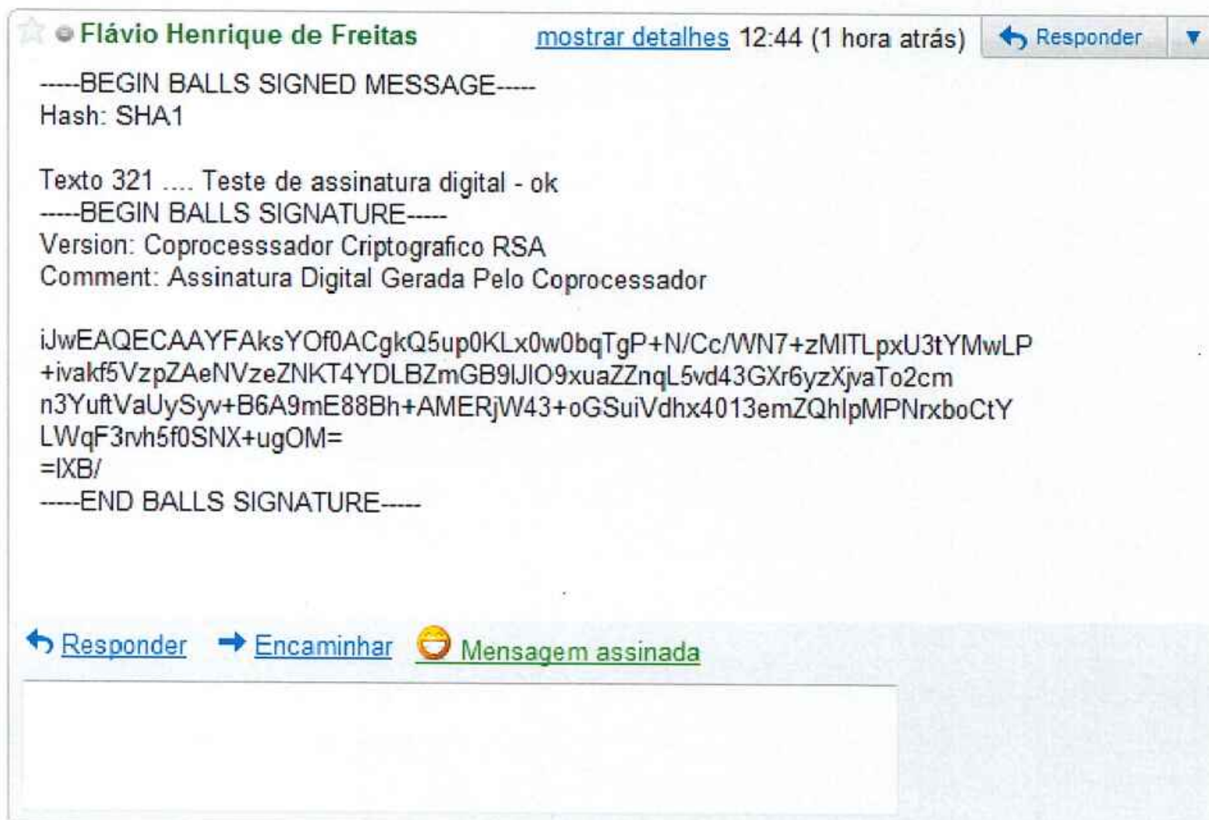

Figura 6.3: Complemento identificou Assinatura Correta

tamente o *software* **FrontPanel** para futuras trocas de dados. A troca de informações entre o coprocessador e o PC é feita apenas por duas funções: *enviaParaUSB* e *recebeDaUSB* (a descrição dessas duas funções está em *Driver USB* no item *Especificação do Projeto*). Essas duas funcões se utilizam de métodos da biblioteca **okjFrontPanel.jar** para troca de dados e são descritas no Apêndice VI.

Na troca de dados existem 3 tipos de *endpoints* que podem ser utilizados:

- Wire: Envio e recebimento de 2 *bytes* por vez
- Pipe: Envio e recebimento de até 1024 *bytes* por vez
- Trigger: Envio e recebimento de um pulso

Na comunicação usada no projeto são apenas utilizados *Pipes*, principalmente devido ao seu tamanho, visto que as mensagens trocadas quase sempre são grandes (maiores que os 2 *bytes* do wire) e por ser esta a sugestão do fornecedor para otimização de desempenho. Portanto, as duas funções da biblioteca **okjFrontPanel.jar** utilizadas nas funções *enviaParaUSB* e *recebeDaUSB* são:

Mensagem alterada Entrada | X

Flávio Henrique de Freitas mostrar detalhes 12:41 (1 hora atrás) Responder

-----BEGIN BALLS SIGNED MESSAGE-----
Hash: SHA1

Texto 123 .... Teste de assinatura digital - modificada
-----BEGIN BALLS SIGNATURE-----
Version: Coprocesssador Criptografico RSA
Comment: Assinatura Digital Gerada Pelo Coprocessador

iJwEAQECAAYFAksYOf0ACgkQ5up0KL
x0w0bqTgP+N/Cc/WN7+zMITLpxU3tYMwLP
+ivakf5VzpZAeNVzeZNKT4YDLBZmGB9lJIO9xuaZZnqL5vd43GXr6yzXjvaTo2cm
n3YuftVaUySyv+B6A9mE88Bh+AMERjW43+oGSuiVdhx4013emZQhlpMPNrxboCtY
LWqF3rvh5f0SNX+ugOM=
=lXB/
-----END BALLS SIGNATURE-----

Responder Encaminhar Assinatura errada

Figura 6.4: Complemento identificou Assinatura incorreta

- Função: WriteToPipeIn(int address, int length, byte[] message)

  Retorno: int

  Descrição: Recebe como parâmetros o endereço do *endpoint* na FPGA, o tamanho da mensagem a ser enviada e a própria mensagem, para então enviá-la via USB para a FPGA.

- Função: ReadFromPipeOut(int address, int length, byte[] message)

  Retorno: int

  Descrição: Recebe como parâmetros o endereço do *endpoint* na FPGA, o tamanho da mensagem a ser recebida e um vetor de *bytes* que armazenará a mensagem, para então recebê-la via USB e guardá-la.

A seção *Driver USB* do item *Especificação do Projeto* juntamente com essa encerram todo o entendimento do módulo USB da parte de *software* do projeto.

6.2 Hardware

O desenvolvimento do hardware foi feito em linguagem VHDL utilizando descrições comportamentais e máquinas de estado. Apenas o bloco que implementa a função SHA-1 utiliza a linguagem Verilog pois não foi desenvolvido ao longo deste projeto e sim retirado do repositório público OpenCores.org . A seguir temos uma lista dos arquivos de código que compõem este projeto sendo que cada arquivo contém a descrição de uma entidade:

- **coprocessadorrsa.vhd** - Entidade de maior nível, entradas e saídas são pinos físicos da FPGA conectados ao controlador USB.
- **InterfaceUSB.vhd** - Entidade que realiza o processamento dos sinais oriundos do controlador USB e os disponibiliza para no formato aceito pela entidade rsa.
- **rsa.vhd** - Principal entidade do projeto, realiza todas as operações criptográficas desenvolvidas.
- **uc.vhd** - Unidade de Controle do componente rsa.
- **MonExpLR256** - Entidade que implementa o cálculo de exponenciação modular para dados de 256 bits.
- **MonPro256** - Entidade que implementa o cálculo de multiplicação modular para dados de 256 bits.
- **sha1.verilog** - Entidade que implementa o cálculo da função de hash, SHA-1.

6.2.1 Interface USB

A implementação do bloco Interface USB foi feita no arquivo *InterfaceUSB.vhd* utilizando descrição comportamental de uma máquina de estados. Conforme descrito na especificação do projeto, este bloco faz uso dos elementos fornecidos pela Opal Kelly: okHostInterface, okPipeIn e okPipeOut.

A figura 6.5 mostra o diagrama de estados da máquina de estados utilizada para leitura na porta USB e a figura 6.6 para a de escrita.

Quando o software em execução no PC em que o dispositivo está conectado deseja enviar dados pela porta USB, ocorre a seguinte sequência de etapas:

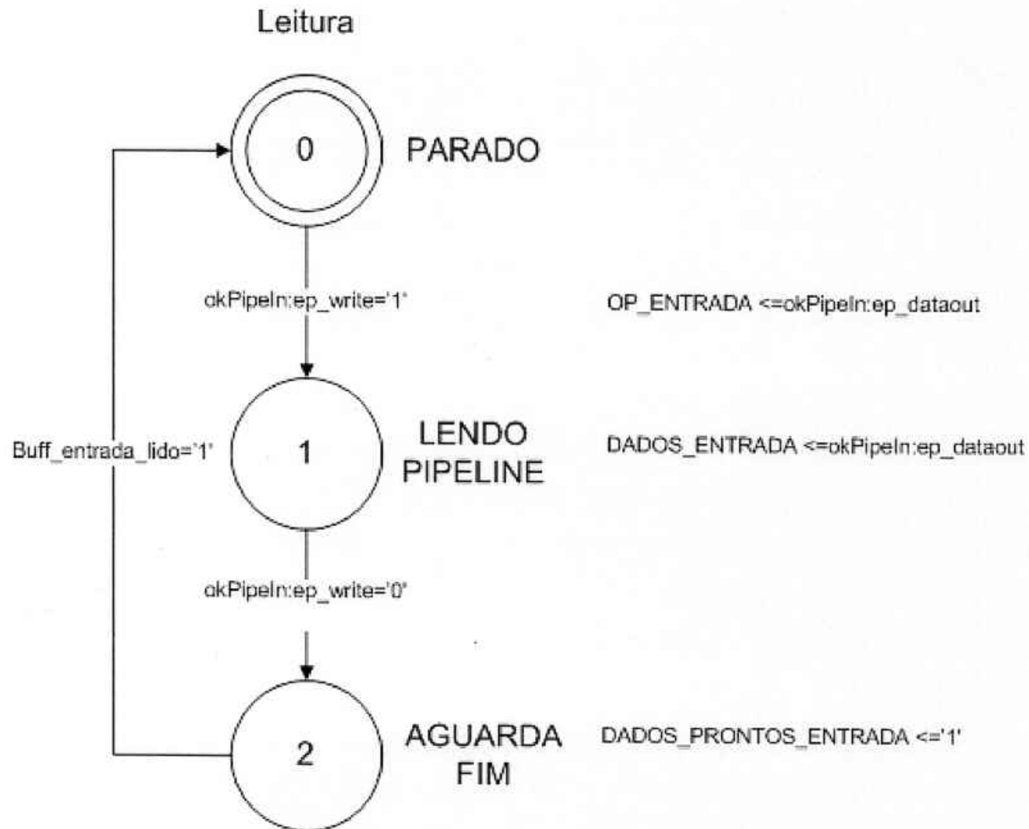

Figura 6.5: Diagramas de estado para a escrita e leitura na interface USB

1. No estado 0-PARADO, o sinal **EP_WRITE** é constantemente lido. Quando estiver ativo, indicando que os primeiros 16 bits de dados estão disponíveis no sinal **EP_DATAOUT**, estes são transferidos para **OP_ENTRADA**. Ocorre então uma transição para o estado 1-LENDO PIPELINE.

2. No estado 1-LENDO PIPELINE, a cada ciclo de *clock* os 16 bits de dados disponíveis em **EP_DATAOUT** são transferidos para **DADOS_ENTRADA**. Um ponteiro é utilizado para indicar a posição dos 16 bits recebidos dentre os 256 bits de **DADOS_ENTRADA**. Quando o sinal **EP_WRITE** muda para '0', indicando que todos os dados do pipeline já foram enviados, ocorre uma transição para o estado 2-AGUARDA FIM

3. No estado 2 - AGUARDA FIM, o sinal **DADOS_PRONTOS_ENTRADA** é ativado, sinalizando o término do recebimento de uma nova solicitação. É verificado constantemente o sinal **BUFF_ENTRADA_LIDO** até que esteja ativo, indicando que os dados recebidos já foram lidos pelo bloco RSA e a Interface USB retorna ao estado 0-PARADO.

Quando o software em execução no PC em que o dispositivo está conectado deseja ler

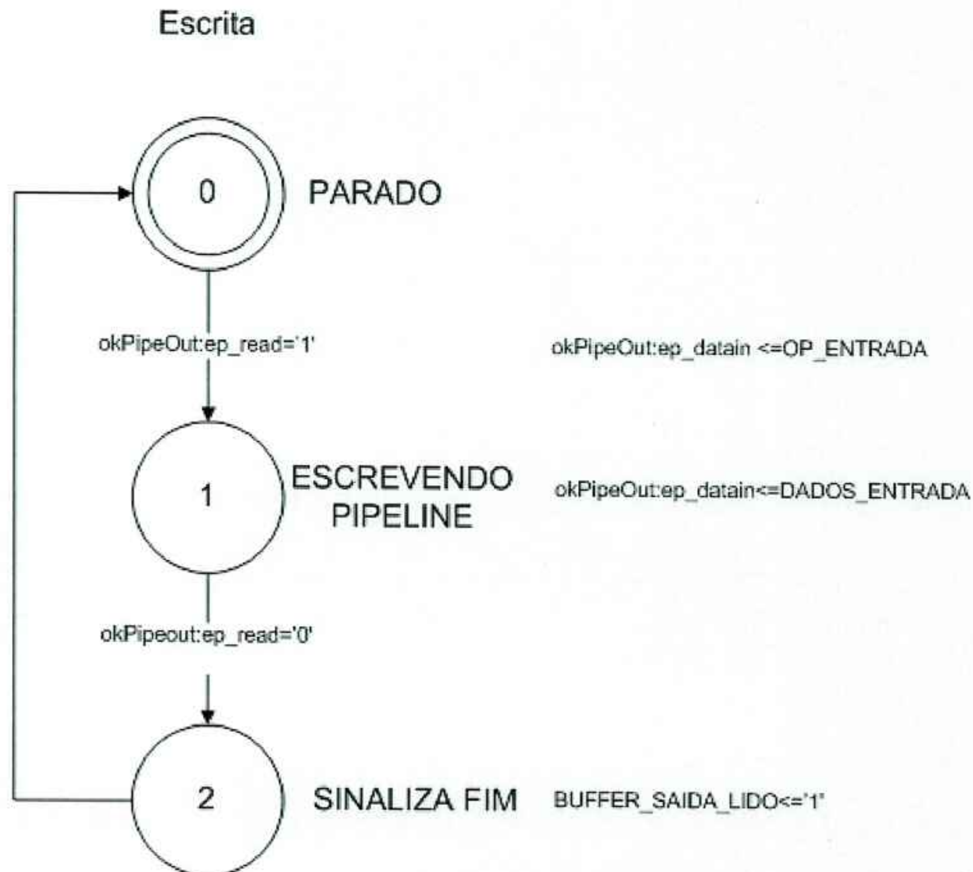

Figura 6.6: Diagramas de estado para a escrita e leitura na interface USB

dados pela porta USB, o envio de dados ocorre a seguinte sequência:

1. No estado 0-PARADO, o sinal **EP_READ** é constantemente lido. Quando estiver ativo, indicando que os primeiros 16 bits de dados devem ser enviados, os dados em **OP_ENTRADA** são transferidos para **EP_DATAOUT**. Ocorre então uma transição para o estado 1-ESCREVENDO PIPELINE.

2. No estado 1-ESCREVENDO PIPELINE, a cada ciclo de *clock* 16 bits de dados de **DADOS_ENTRADA** são transferidos para **EP_DATAOUT**. Um ponteiro é utilizado para indicar a posição dos 16 bits enviados dentre os 256 bits de **DADOS_ENTRADA**. Quando o sinal **EP_READ** muda para '0', indicando o término da solicitação pelo software ou quando todos os dados de **DADOS_ENTRADA** já foram enviados, ocorre uma transição para o estado 2-SINALIZA FIM.

3. No estado 2-SINALIZA FIM, o sinal **BUFFER_SAIDA_LIDO** é ativado, sinalizando o término do envio dos dados solicitados e ocorre uma transição para o estado 0-PARADO.

6.2.2 Bloco RSA

O bloco RSA contém a implementação das funções criptográficas do sistema. Ele faz uso dos blocos Exponenciador Modular, para as operações de encriptação, bloco SHA1, para o cálculo de hash e bloco UC como unidade de controle. O arquivo *uc.vhd* contém a descrição comportamental em VHDL de uma máquina de estados que faz o papel de unidade de controle do coprocessador. Esta máquina de estados lida com os sinais de controle, decodifica e executa as operações solicitadas no sinal **OP_ENTRADA**. O diagrama de estados da figura 6.7 expõe o funcionamento interno deste bloco.

6.2.3 Exponenciador Modular

O bloco de exponenciação modular foi desenvolvido como uma máquina de estados que implementa o método L-R(*Left to Right*) de exponenciação de números binários. Tal algoritmo foi retirado de (**??**). A exponenciação modular é executada como multiplicações modulares em sequência. Este método também é conhecido como *Square-Multiply* pois uma das multiplicações do laço principal é um cálculo quadrático. O método L-R, em que o valor da base da exponenciação é percorrido da esquerda para a direita, contrasta com o método R-L, com o valor percorrido da direita para a esquerda, sendo o primeiro otimizado para obtenção de um hardware mais compacto e o segundo para obtenção de um hardware mais veloz. A seguir é apresentado o algoritmo do método L-R utilizado nesta implementação.

Algoritmo: Exponenciação Modular L-R

ENTRADA: P - Texto Puro,E-Expoente,M-Módulo, C-Constante 2^{2n} pré calculada

MontExpLR(P,E,M)

{

$C := 2^{2n} mod M;$

$P := MonPro(C, P, M);$

$R := MonPro(C, 1, M);$

for i=k-1 downto 0 do

R:=MonPro(R,R,M);

$if(E_i = 1)then$

$R := MonPro(R, P, M);$

end if;

end for;

$R := MonPro(1, R, M);$

$ReturnR;$

}

A máquina de estados desenvolvida segue o fluxograma da figura 6.8.

6.2.4 Multiplicador Modular

O bloco de multiplicação modular foi desenvolvido como uma máquina de estados que implementa o algoritmo de multiplicação modular de Montgomery retirado de (**??**) e apresentado a seguir.

Algoritmo: Multiplicação Modular De Montgomery

ENTRADA: A - Fator A,B-Fator B,M-Módulo

MonPro3(A,B,M)

{

$S_{-1} := 0;$

$A := 2 \times A;$

for i = 0 to n do

$q_i := (S_{i-1})Mod2; (LSBofS_{i-1})$

$S_i := (S_{i-1} + q_{iM} + b_{iA}) = 2;$

end for

Return S_n *;*

}

A máquina de estados desenvolvida segue o fluxograma da figura 6.9.

Quando o sinal **START** é ativado, o valor $A = 2xA$ é calculado e a soma principal do algoritmo é executada n+1 vezes . Por último, o valor residual de M é subtraído caso necessário e o resultado é liberado na saída juntamente com a ativação do sinal **READY**

6.2.5 Bloco HASH (SHA-1)

O cálculo da função de *hash* no coprocessador é feito pelo bloco SHA-1. Foi utilizada uma implementação desenvolvida por Paul Hartke obtida no portal OpenCores.org. Tal implementação utiliza elementos em Verilog bem estruturados. Não foi feita uma análise profunda

destes elementos. Para validar o bloco sha1, de forma a garantir seu correto funcionamentos, foram executadas diversas simulações e os resultados obtidos foram comparados com a função de cálculo de *hash* desenvolvida em software. Como os resultados foram positivos, assumimos que o funcionamento está correto e pudemos utilizar este componente neste projeto.

Essa função calcula o resumo para entrada de 32 *bits* e tem como saída um *hash* de 160 *bits*. Para mensagens que tenham tamanho maiores que a entrada dessa implementação, a camada de de software intermediária com as funções criptográficas divide as mensagens em blocos de tamanhos compatíveis para se adequar ao algoritmo. O algoritmo utiliza os valores dos resumos anteriores para gerar o *hash* para o próximo bloco, no caso de haver uma mensagem maior que 32 *bits*.

A integração deste bloco foi feita adicionando uma instância deste elemento na entidade rsa. A unidade de controle se encarrega de gerar os sinais de controle e transferência de dados corretamente conforme exposto na especificação de hardware, item 3.4, deste documento.

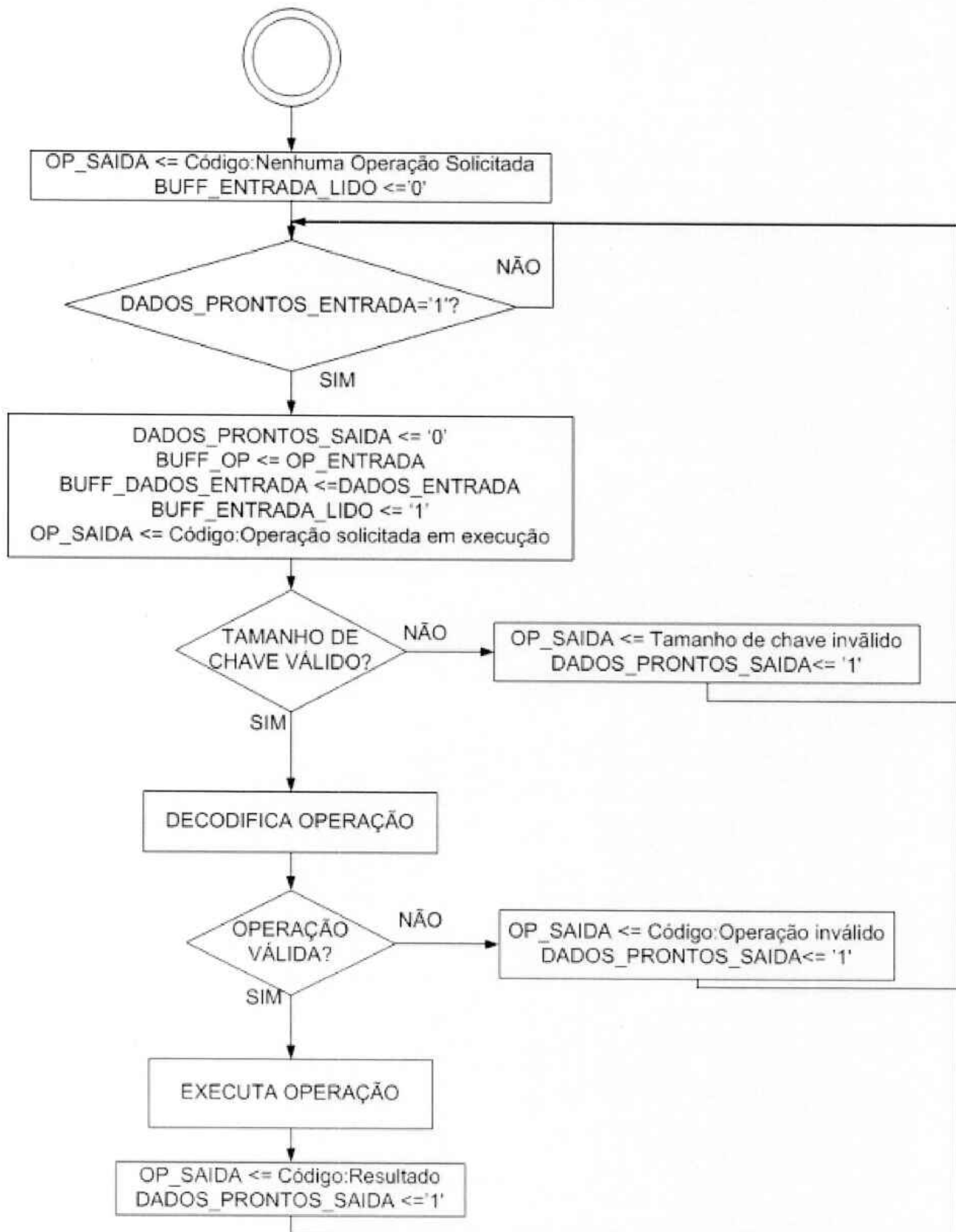

Figura 6.7: Fluxograma da máquina de estados de controle do bloco RSA

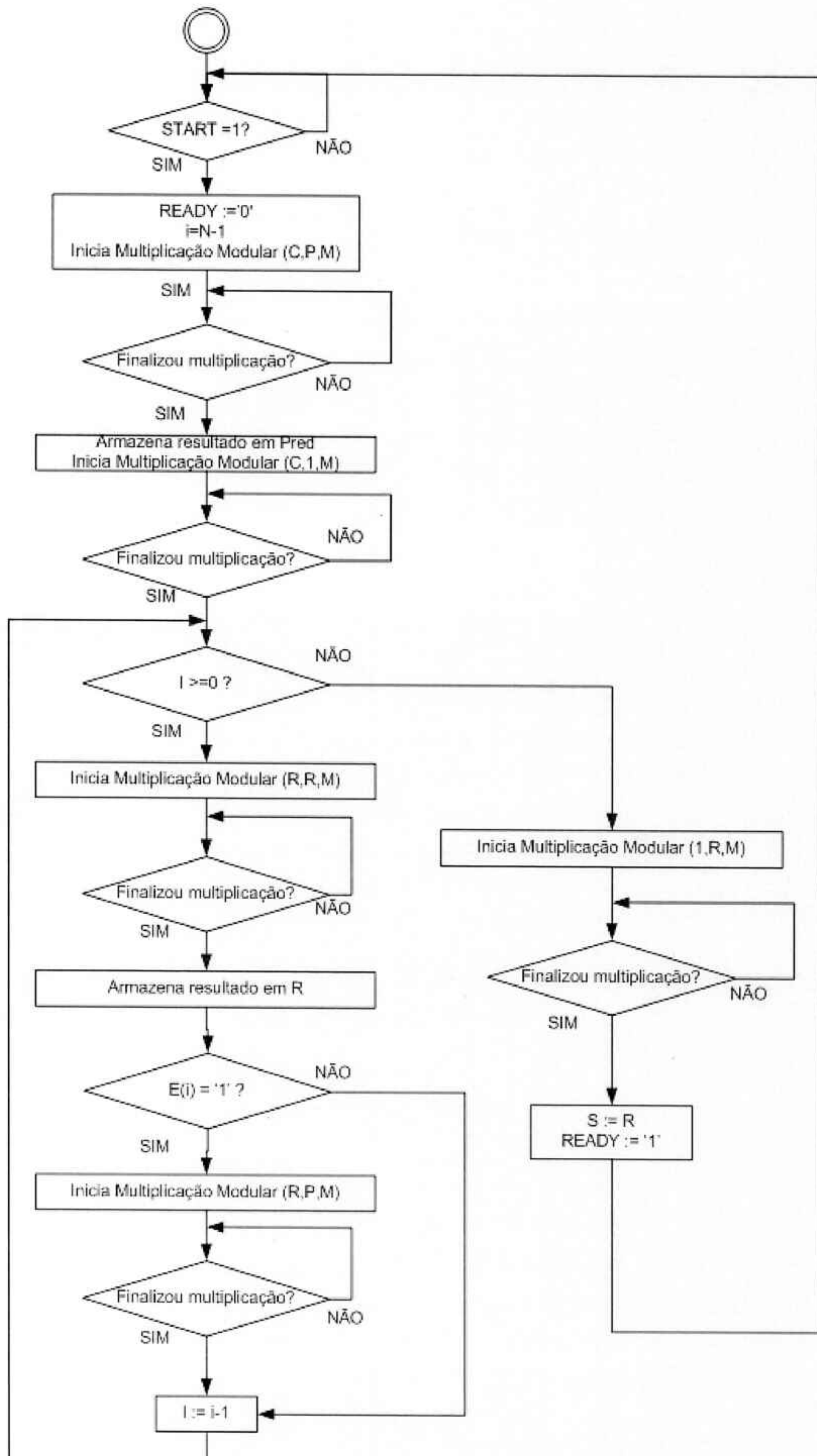

Figura 6.8: Fluxograma da máquina de estados do Exponenciador Modular

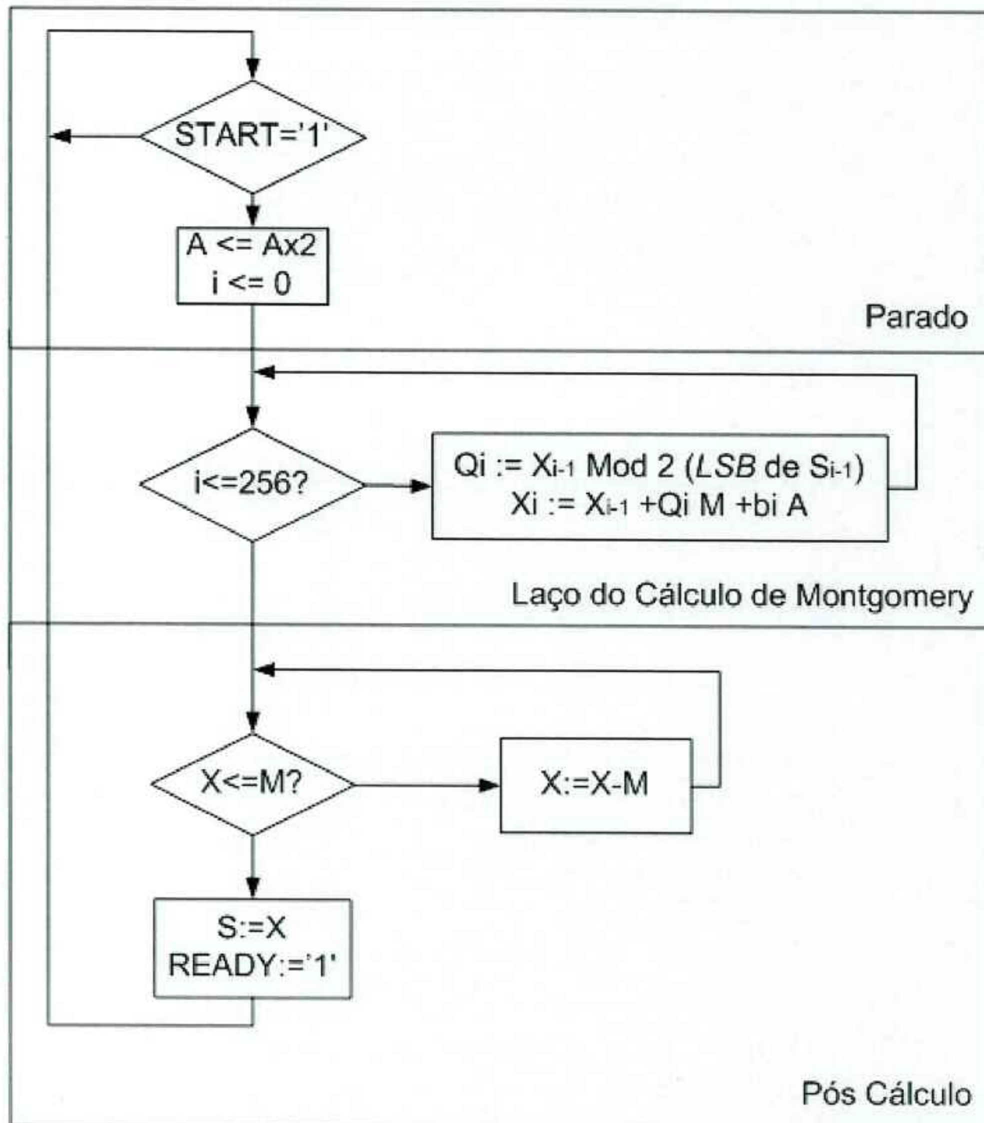

Figura 6.9: Fluxograma da máquina de estados do Multiplicador de Montgomery

7 TESTES E AVALIAÇÃO

7.1 Metodologia de testes

Como o projeto envolve diversos módulos, foram adotados dois tipos de testes: unitário e de integração. No primeiro, cada módulo do projeto (*hardware*, funções criptográficas, *software* de teste e de demonstração) foi testado isoladamente. Na etapa de teste de integração todos os módulos foram testados juntos para verificar se era necessária alguma alteração nas interfaces de conexão entre os módulos. O *software* de teste tem essa finalidade de verificar se o sistema e algumas partes estão ou não funcionais. Para teste da própria interface e das funções criptográficas, foram verificados os resultados da geração das chaves, encriptação, e cálculo de *hash* através de funções disponibilizadas pela biblioteca de segurança java.security, e para o teste do *hardware* foram comparados os valores de resposta desse último com os apresentados pelas funções em *software* anteriormente calculadas. Juntamente com o teste funcional, foi proposto realizar um teste de desempenho para verificar o ganho de uma aplicação de criptografia RSA desenvolvida em *hardware* com uma desenvolvida em *software*, além disso foi interessante verificar o impacto que a implementação do algoritmo de redução de Montgomery causou.

7.1.1 Testes e Simulações de Hardware

Todos os elementos de *hardware* foram testados isoladamente por meio de simulações executadas pelo sofware ISE da Xilinx. Desta forma, foi possível depurar o sistema e corrigir erros antes de executar a síntese e configuração do FPGA. Para cada entidade criada em vhdl, foi criado um arquivo em VHDL para simular os respectivos sinais de entrada. O software ISE disponibiliza uma ferramenta que automaticamente cria tais arquivos de simulação bastando inserir os estímulos dos sinais de entrada desejados e saídas esperadas. Estes arquivos são chamados de *VHDL test bench* pelo ISE. Para simulação da comunicação USB, foi utilizado o simulador ModelSim configurado com arquivos de dll fornecidos pela Opal Kelly exclusivamente para este fim. O ambiente de simulação configurado permite simular as mesmas chamadas à

USB disponíveis na API da Opal Kelly utilizada na camada de software.

7.2 Desempenho

Através do algoritmo de encriptação é possível prever qual é a quantidade de ciclos que uma implementação em *hardware* irá utilizar para realizar uma operação deste tipo, então, em linhas gerais, o desempenho de uma aplicação em hardware é previsível, dependendo apenas da frequência de *clock* para determinar o tempo de uma operação. Através da carta de tempos dos sinais da implementação em *hardware* é possível verificar o tempo que o dispositivo gasta para realizar a operação de encriptação de um texto claro, além disso foi inserido um contador de tempo na própria aplicação em *hardware* para o mesmo tipo de verificação.

7.3 Comparação de Desempenho

Para comparar o desempenho da aplicação em *hardware* comparou-se com o tempo que o *software* gastou para realizar uma mesma operação, registrando esse tempo na interface gráfica do *software* de teste. Ao realizar qualquer operação disponível na interface gráfica será registrada nela o tempo decorrido para se realizar a operação tanto em *hardware* quanto em *software*, a partir desses registros na interface é possível verificar o ganho de uma aplicação em relação a outra.

8 RESULTADOS

8.1 Resultados funcionais

Até o momento da finalização deste documento, os seguintes itens foram finalizadas de acordo com a especificação apresentada e os resultados obtidos foram corretos:

- Encriptação e decriptação em hardware utilizando RSA 256.
- Integração entre software e hardware via USB
- Software de comunicação USB
- Software de funções criptográficas
- Software de testes
- Software de demonstração

Para a finalização da implementação da funcionalidade de cálculo de hash, existe um trabalho de integração ainda não finalizado e que deve ocorrer nos dias subsequêntes à entrega deste documento. O mesmo vale para a operação de assinatura digital por ser esta dependente do cálculo de hash.

As funcionalidades da aplicação de teste e de demonstração que necessitam de tais operações utilizam a implementação em software disponível na camada de funções criptográficas. Dessa forma, a assinatura de mensagens do software de demonstração executa o cálculo de hash em software e a encriptação em hardware.

8.2 Comparação de desempenho Software x Hardware

Um dos objetivos deste trabalho era verificar o possível ganho de desempenho no uso de um dispositivo dedicado a cálculos criptográficos em comparação à equivalente execução em

software. Considerando a grande diferença entre os recursos computacionais de uma estação de trabalho moderna e de um FPGA, principalmente com relação às altas frequencias de clock do primeiro, foi feita a opção por realizar uma comparação baseada em estimativas de quantidade de ciclos de clock necessários para executar as operações testadas. A estimativa da quantidade de ciclos de clocks utilizados nas execuções em software foi feita pela multiplicação do tempo necessário para tal execução pela freqüência de clock do processador em uso. A tomada do tempo de execução foi feita calculando-se a diferença entre os instantes imediatamente posterior e anterior à execução de uma operação. O valor dos instantes foi obtido com precisão de nanosegundos por chamadas de sistema ao sistema operacional em execução. Para obtenção dos valores, cada operação foi executada mil vezes, e a média dos tempos de execução foi utilizada.

Para a estimativa de quantidade de ciclos de clock utilizados nas execuções em hardware, inicialmente houve o desejo de se implementar um módulo de contagem diretamente no hardware. Por restrições de tempo, tal implementação não pode ser executada. Surgiram então duas alternativas. A primeira seria a de executar a mesma medida descrita acima para a operação em software, tomando como instantes de medição, o momento em que uma solicitação ao hardware é feita pelo software de funções criptográfias na classe FuncoesHW. Esta medição foi incluída no software de teste devido à sua facilidade de implementação. Não acreditamos porém, que esta seja uma medida de fato válida, visto que o software de funções criptográficas não foi desenvolvido com vistas à otimização de desempenho e também pela dificuldade em se estimar qual parcela do tempo de execução seria responsabilidade das chamadas de métodos de software e da API da Opal Kelly utilizada. A segunda alternativa, e que consideramos aqui como sendo a mais válida, é a de realizar a estimativa desejada com base na análise cuidadosa da execução das máquinas de estado implementadas. As fórmulas de tempo de execução obtidas para os elementos de hardware importantes nesta análise foram as seguintes, sendo N o tamanho das chaves criptográficas implementadas.

- Tempo de execução do módulo de multiplicação modular:
  $TMULT = N + 3$;
- Tempo de execução do módulo de exponenciação modular:
  $TEXP = 3 + 3 \times TMULT + N \times (1 + TMULT + K \times (1 + TMULT))$;
  Onde K representa a quantidades de bits '1' no valor do expoente utilizado no cálculo de exponenciação (este fator é compreendido quando se verifica a linha 2(b) do algoritmo de exponenciação modular apresentado no item 2.3).
- Tempo de execução da operação de encriptação/decriptação pelo módulo RSA:

$TRSA = 3 + TEXP;$

Para a implementação de 256 bits desenvolvida chegamos aos seguintes valores:

- $TMULT = 259$
- $TEXP = 67340 + 66560 \times K$
- $TRSA = 67343 + 66560 \times K$

O gráfico da figura 8.1 apresenta o resultado calculados para a execução em hardware para diferentes valores de K. A area sombreada representa a faixa de resultados obtidos em software, medidos com a a aplicação de testes, com o tempo de execução entre 0,5 e 1,5 milhão de ciclos de clock.

Figura 8.1: Comparação entre o desempenho medido em software e calculado em hardware para operações de encriptação com RSA 256

Estes resultados mostram que a implementação em hardware desenvolvida não pode obter um melhor desempenho do que uma implementação simples em software do algoritmo RSA 256. Acreditamos que tal tarefa será dificilmente alcançada com um coprocessador de arquitetura simples. Acreditamos que isso se deva principalmente a não termos explorado o paralelismo do hardware e buscado otimizar a implementação do algoritmo, já que otimização não era um dos atributos necessários no escopo do projeto. Não nos aprofundamos na análise do algoritmo de exponenciação modular da classe BigInteger, utilizada na implementação em software avaliada mas, é fácil imaginar que um grande trabalho de otimização faça parte do

histórico de desenvolvimento desta classe. Não podemos esquecer também que a metodologia de medição utilizada é bastante simples e carrega provavelmente um erro de medição considerável que não pode ser desconsiderado mas é dificilmente mensurável.

9 CONCLUSÃO

9.1 Considerações Finais

Esse projeto teve como objetivo discutir e desenvolver a construção de um coprocessador criptográfico. Para tanto nos baseamos em teorias de computação voltadas a *hardware* e a *software*. Para construção do dispositivo foi utilizada a síntese em FPGA dos algoritmos criptográficos necessários. A interface de acesso aos recursos do *hardware* foi disponibilizada por funções em *software*. A comunicação *software-hardware* deu-se através da utilização da interface USB.

Como o dispositivo calcula funções com números de precisão estendida, o desenvolvimento aconteceu por iterações, isso possibilitou o entendimento do funcionamento do algoritmo e do hardware, assim como descobrir suas limitações. Ao passos que o projeto se adiantava, passava a ter mais capacidade computacional, ou seja, maiores números de *bits* sendo computados ao mesmo tempo. O desenvolvimento do projeto partiu de cálculos com 8 *bits* e alcançou 1024 *bits*, onde quería-se chegar.

Para desenvolver o algoritmo em hardware, primeiramente era desenvolvido o projeto em software, testado e feitas as devidas simulações. Após verificar que estava funcionando adequadamente, ele era carregado, testado e adaptado para o hardware. Ao alcançar o número de 512 *bits* percebeu-se que a placa didática não mais suportava o projeto. Uma iteração a mais para o desenvolvimento foi feita, pois acreditou-se ser um problema momentâneo de entendimento da placa, e chegou-se a 1024 bits testados em projeto, mas que não podia ser executado na placa por limitações físicas. Após alguns estudos aprofundados verificou-se que essa limitação de hardware não permitiria que a implementação utilizasse mais que 256 *bits*. Decidiu-se, então, por continuar o projeto e finalizar o dispositivo com o número de bits alcançados, já que estaría sendo criado um coprocessador completo, como planejado.

A limitação encontrada poderia ter sido facilmente contornada com a aquisição de uma placa com maior capacidade computacional. Decidiu-se por permanecer com a mesma placa,

já que tería-se ganhos pouco significativos quanto a modificação ou entendimento do sistema, pois toda as partes teórica e prática já tinham sido dominadas. Como o projeto visava ser uma prova de conceito e de possibilidade de estudos, um coprocessador com maior capacidade, nesse escopo de projeto, apenas iria diferenciar quanto ao tipo de possíveis demonstrações do sistema. Como o número de 256 *bits* de chaves criptográficas para o algoritmo RSA não é o recomendado por entidades de padronização, o número de *softwares* de aplicações que respeitassem isso era muito limitada. Isso nos trouxe uma certa dificuldade, já que não havia muitas escolhas quanto a um *software* de demonstração. Como o grupo estava decidido e motivado a criar um ambiente interessante e que de fato retratasse e utilizasse ferramentas conhecidas e populares, decidimos por criar nosso próprio *software* de demonstração. Após várias horas e noites de estudos e pesquisas, conseguiu-se desenvolver um sistema que interagia com o provedor de emails mais popular atualmente, o Gmail. Dessa forma, foi possível criar um sistema que demonstra, através de um recurso do cotidiano de utilização de muitas pessoas, o nosso dispositivo e estudos.

Através desse projeto foi possível demonstrar a criação de um dispositivo em *hardware* que realiza funções criptográficas. Isso possibilitou ao grupo, estudos de teorias criptográficas, assim como o de desenvolvimento e implementação de *hardware*. Foi demonstrado, mesmo que não construído por limitação no escopo do projeto, métodos e técnicas para construção de um dispositivo que pudessem resolver ou dificultar alguns ataques de segurança de informação, tais como o *man-in-the-middle*.

9.2 Perspectivas Futuras

Mesmo tendo criado e demonstrado um dispositivo funcional, existem características que podem ser evoluídas ou modificadas.

Para a construção de um sistema que permita o total controle das ações desse pelo usuário, pode-se adicionar aos dispositivos alguns componentes tais como um *display* e um teclado. Isso possibilitará que a cada ação praticada pelo hardware, o usuário deverá aceitá-la ou não. Dessa forma será possível verificar se a ação solicitada ao dispositivo, realmente é a mesma que o usuário solicitou ao sistema que utiliza. Por exemplo, através do *internet banking* o pagamento de uma conta de determinado valor, ao invés da transferência de todo o dinheiro da conta para um usuário mal intencionado.

Outra melhoria é o desenvolvimento de sistemas com maior nível de segurança, ou seja, trabalhar com chaves maiores para o algoritmo RSA, tais como 1024, 2048 ou 4096 bits.

Evoluir o sistema para que trabalhe e calcule outros algoritmos criptográficos, como exemplo pode-se citar o AES, TripleDES, entre outros.

Desenvolver uma arquitetura que permita o uso de *pipeline* e, portanto, a paralelização de cálculos.

Outra interessante modificação possível é a criação de algoritmos que criem as chaves criptográficas utilizadas, dentro do próprio dispositivo.

Ficam, portanto, algumas sugestões de mudanças que podem ser realizadas no sistema, sendo que o detalhamento e implementação podem ser realizadas conforme as necessidades.

REFERÊNCIAS BIBLIOGRÁFICAS

QIANG Meng and Yuan-feng Liu and Zi-bin Dai. Microsoft Windows Driver Foundation. Disponível em: http://www.microsoft.com/whdc/driver/wdf/default.mspx. Acesso em 16 abr. 2009.

Microsoft Kernel-Mode Driver Framework. Disponível em: http://www.microsoft.com/whdc/driver/wdf/KMDF.mspx. Acesso em 16 abr. 2009.

Microsoft Kernel-Mode Driver Framework. Disponível em: http://www.microsoft.com/whdc/driver/wdf/UMDF.mspx. Acesso em 16 abr. 2009.

Documentação da Propriedade Intelectual da Xilinx OPB USB 2.0. Disponível em: http://www.xilinx.com/support/documentation/ip_documentation/usb_ds591.pdf. Acesso em 16 abr. 2009.

BONEH D., L. B.; SHACHAM, H. *Short signatures from the weil pairing*. 2004. JournalofCryptology.

BURNETT, S. *How RSA Works*. 1996.

DAEMEN, J.; RIJMEN, V. *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002. ISBN 3540425802.

DALY, A.; MARNANE, W. Efficient architectures for implementing montgomery modular multiplication and rsa modular exponentiation on reconfigurable logic. In: *FPGA '02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. New York, NY, USA: ACM, 2002. p. 40–49. ISBN 1-58113-452-5.

ECRYPT. *ECRYPT Yearly Report on Algorithms and Keysizes (2007-2008)*. 2008. D.SPA.28 Rev. 1.1, IST-2002-507932 ECRYPT, 07/2008. http://www.ecrypt.eu.org/ecrypt1/documents/D.SPA.28-1.1.pdf.

ELGAMAL, T. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithm*. 1985. IEEE Transactions on Information Theory.

FREED, N.; BORENSTEIN, N. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. 1996.

FREITAS, F. H.; BARRETO, P. S. L. M. B.; CARVALHO, T. C. M. B. *Assinatura Digital para o Selo Verde*. 2009. Workshop de Trabalhos de Iniciação Científica e de Graduação do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais.

GAUBATZ, G. *Versatile Montgomery Multiplier Architectures*. April 2002. Master's thesis, Electrical and Computer Eng. Dept., Worcester Polytechnic Inst., Worcester, Mass.,.

HARTKE, P. *Projeto de hardware SHA1 no portal OpenCore.org*. 2004. Projeto de hardware SHA1 no portal OpenCore.org. Disponível publicamente para download em: http://www.opencores.org/project,sha1. Acesso em 05 dez. 2009.

KINGPIN. *Attacks on and Countermeasures for USB Hardware Token Devices*. 2000.

KOC Çetin K. *High-Speed RSA Implementation*. 1994.

KOOPMAN, P. *32-Bit Cyclic Redundancy Codes for Internet Applications*. 2002.

MAIER, R. R. *Teoria dos Números*. 2005.

MARTINS, R. d. R. R. P. R. *Projeto de uma classificação de tráfego hostil em redes de computadores*. 2008.

MENEZES, A.; OORSCHOT, P. van; VANSTONE, S. *Handbook of Applied Cryptography*. 1st. ed. [S.l.]: CRC Press, 1996.

MENG, Q.; LIU, Y. feng; DAI, Z. bin. *FPGA Implementation of Expandable RSA Pubic-Key Cryptographic Coprocessor*. 2006.

MONTGOMERY, P. L. *Modular Multiplication Without Trial Division*. 1985. pp. 519-521 p. Mathematics of Computation.

National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES)*. [s.n.], 1999. Supersedes FIPS 46-2. Disponível em: <http://www.itl.nist.gov/fipspubs/fip186-2.pdf>.

NIST. *Recommendation for Key Management – Part 1: General*. 2007. http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf.

OPPLIGER, R. *Contemporary Cryptography*. [S.l.: s.n.], 2005.

PRESSMAN, R. S. *Engenharia de Software*. Secaucus, NJ, USA: MCGRAW-HILL BRASIL, 2006. ISBN 8586804576.

RABIN, M. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*. 1979. MIT Laboratory for Computer Science.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, v. 21, p. 120–126, 1978.

Sá, I. P. de. *Tratamento da Informação na educação básica: aritmética modular e os códigos de identificação do Cotidiano*. 2007. UERJ / USS/ PEDRO II.

SAVAS, E.; TENCA, A. F.; KOç, e. K. A scalable and unified multiplier architecture for finite fields gf(p) and gf(2m). In: *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*. London, UK: Springer-Verlag, 2000. p. 277–292. ISBN 3-540-41455-X.

SOUZA, R. S. de. *Método de Multipllicação de Números Grandes*. 2007.

STALLINGS, W. *Cryptography and Network Security Principles and Practices*. 4th. ed. [S.l.]: Prentice Hall, 2006. 379 p.

TANENBAUM, A. S. *Computer Networks*. 4th. ed. [S.l.]: Prentice Hall, 2003. 208-212 p.

TENCA, A. F.; KOç, e. K. A scalable architecture for modular multiplication based on montgomery's algorithm. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 52, n. 9, p. 1215–1221, 2003. ISSN 0018-9340.

TENCA, A. F.; TODOROV, G.; KOç, e. K. High-radix design of a scalable modular multiplier. In: *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*. London, UK: Springer-Verlag, 2001. p. 185–201. ISBN 3-540-42521-7.

UTO, N.; MELO, S. Minicurso vulnerabilidades em aplicações web e mecanismos de proteção. In: . [S.l.: s.n.], 2009. IX Simpósio Brasileiro em Segurança da Informação e De Sistemas Computacionais.

VENTURA, J. P. F.; DAHAB, R. Minicurso introdução a ataques por canais secundários. In: . [S.l.: s.n.], 2009. IX Simpósio Brasileiro em Segurança da Informação e De Sistemas Computacionais.

WANG, H. Y. X.; YIN, Y. L. *Finding Collisions in the Full SHA-1*. 2005. CRYPTO 2005.

WOLLINGER, T.; PAAR, C. How secure are fpgas in cryptographic applications? In: *Proceedings of International Conference on Field Programmable Logic and Applications (FPL 2003), Lecture Notes in Computer Science Volume 2778*. [S.l.]: Springer-Verlag, 2003. p. 91–100.

Apêndice I – ESPECIFICAÇÃO DOS ELEMENTOS DA INTERFACE DE TESTE

A interface gráfica contém os seguintes elementos:

- Expoente da Chave Pública - Campo de texto que exibe o valor do expoente da chave pública em uso. O valor pode ser inserido manualmente ou calculado pelo software.
- Módulo da Chave Pública - Campo de texto que exibe o valor do módulo da chave pública em uso. O valor pode ser inserido manualmente ou calculado pelo software.
- Constante de exponenciação pré calculada da Chave Pública - Campo de texto que exibe o valor da constante de exponenciação pré cálculada referente à chave pública em uso. O valor pode ser inserido manualmente ou calculado pelo software.
- Mensagem em texto puro (Exibida como texto) - Campo de texto que permite digitar uma mensagem sobre a qual será executada alguma operação disponível.
- Mensagem em texto puro (Convertida para formato numérico) - Campo de texto que exibe o valrro em hexadecimal da mensagem digitada no campo anterior de acordo com o padrão ASCII Estendido. Este é o valor numérico utilizado para a execução das operações criptográficas disponíveis. Este campo é do tipo somente leitura e é atualizado cada vez que um caracter é inserido ou alterado no campo anterior.
- Mensagem em texto puro com *padding* (em formato numérico) - Exibe o conteúdo da mensagem em texto puro com a inserção de valores de *padding* que adequam o tamanho da mensagem para a aplicação do cálculo de hash seguindo o padrão indicado na seção 6.3 da RFC 5652. A mensagem é concatenada com M bytes de valor M sendo M = 160 - (<Tamanho da mensagem> mod 160).
- Tamanho da mensagem - Um campo texto indicando a quantidade de caracteres já digitados e a quantidade de bits que representa estes caracteres. Cada caracter ocupa 8 bits.

- Mensagem cifrada (Em formato numérico) - Valor em hexadecimal da mensagem cifrada resultante das operações de encriptação.
- Mensagem cifrada (Exibida como caracteres ASCII extendido) - Conversão para o formato texto do valor resultante das operações de encriptação.
- Valor do hash sha1 - Valor em hexadecimal resultante da operação de cálculo de hash.
- Assinatura - Valor em hexadecimal resultante da operação de cálculo de assinatura.
- Botão Gerar Chaves - este botão executa o método gerarparChaves da classe Coprocessador e exibe os valores de chaves criados nos respectivos campos.
- Botão Carregar Chaves - Este botão executa em sequência os métodos que enviam ao coprocessador os valores de chaves exibidos nos respectivos campos
- Botão Encriptar com Chave Pública - Este botão executa a operação de encriptação com a chave pública previamente armazenada usando a mensagem inserida no seu respectivo campo e exibe o resultado obtido no campo de mensagem cifrada.
- Botão Encriptar com Chave Privada - Este botão executa a operação de encriptação com a chave pública previamente armazenada usando a mensagem inserida no seu respectivo campo e exibe o resultado obtido no campo de mensagem cifrada.
- Botão Calcular Hash - Este botão executa a operação de cálculo de hash usando a mensagem inserida no seu respectivo campo e exibe o resultado obtido no campo de *hash*.
- Botão Assinar - Este botão executa a operação de assinatura usando a mensagem inserida no seu respectivo campo e a chave privada previamente armazenada no coprocessador e exibe o resultado obtido no campo de assinatura.
- Botão Verificar Assinatura - Este botão executa a operação de verificação de assinatura usando a mensagem inserida no seu respectivo campo e a chave pública previamente armazenada no coprocessador e exibe o resultado obtido no campo de assinatura.
- Tempo de execução em hardware - Campo de texto que exibe o tempo em milisegundos utilizado para a execução em hardware da última operaçao solicitada. É atualizado ao término de cada operação executada.

- Tempo de execução em software - Campo de texto que exibe o tempo em milisegundos utilizado para a execução em software da última operaçao solicitada. É atualizado ao término de cada operação executada.

- Estimativa de ciclos de *clock* em sofware - Calculada como sendo o tempo de execução em software dividido pelo período do ciclo de clock do computador em uso.

- Estimativa de ciclos de *clock* em hardware - Calculada como sendo o tempo de execução em hardware dividido pelo período do ciclo de clock do kit FPGA (freq. = 48MHz) em uso.

- Frequência do *clock* processador executando este software - Frequência em GHz do *clock* do processador do computador em uso.

Apêndice II – FUNÇÕES CRIPTOGRÁFICAS

Coprocessador: é uma classe de interface para as funções desenvolvidas, tanto as implementadas em *software* quanto em *hardware*. Responsável também por manipular os dados e integrar métodos a fim de realizar funções mais complexas. Por exemplo, permite gerar assinatura digital, através da utilização das funções *rsaChavePrivada* e *calculaHash*.

Funções:

- *Função: encripta(byte[] textoClaro, Chave chavePublica)*
  Retorno: byte[] - (texto cifrado)
  Descrição: Encripta a mensagem (textoClaro) com a chave pública passada. Para tanto, duas funções da classe FuncaoHW são executadas, *carregaChavePublica* e *rsaChavePublica*.

- *Função: encripta(byte[] textoClaro)*
  Retorno: byte[] - (texto cifrado)
  Descrição: Encripta a mensagem (textoClaro) com a chave pública que já está gravada no coprocessador. Essa função será útil para encriptar mais de uma mensagem com a mesma chave, já que não será necessário carregar a chave várias vezes no *hardware*. Executa a função *rsaChavePublica* da classe FuncaoHW.

- *Função: decripta(byte[] textoCifrado)*
  Retorno: byte[] - (texto claro)
  Descrição: Decripta a mensagem com a chave privada que está gravada na memória do coprocessador, para realizar essa ação executa a função rsaChavePrivada.

- *Função: assina(byte[] mensagem)*
  Retorno: byte[]
  Descrição: Assina a mensagem com a chave privada que está gravada na memória, através da função *calculaHash* e *rsaChavePrivada* da classe FuncaoHW.

- *Função: verificaAssinatura(byte[] mensagem, byte[] assinatura, Chave chavePublica)*
  Retorno: boolean
  Descrição: Verifica se a assinatura digital da mensagem está correta. Solicita para a classe FuncaoHW a execuçao das funções *calculaHash* e *rsaChavePublica*, por fim compara os valores obtidos com os passados.

- *Função: verificaAssinatura(byte[] mensagem, byte[] assinatura)*
  Retorno: boolean
  Descrição: Verifica a assinatura digital, utilizando a chave pública que já se encontra no coprocessador. Execução idem a da função anterior.

- *Função: calculaHash(byte[] mensagem)*
  Retorno: byte[] (resumom da mensagem)
  Descrição: Calcula função de *hash* da mensagem passada. Executa a função *calculaHash* da classe FuncaoHW.

- *Função: geraParChaves()*
  Retorno: Chave (chave pública)
  Descrição: Gera par de chaves criptográficas. Retorna a chave pública e a privada é gravada no coprocessador. Todas as operações que necessitarem do uso de chave privada, utilizarão esse valor. Para tanto, executa a função *geraChaves* da classe FuncaoSW.

FuncoesHW: apresenta as funções que foram implementadas em *hardware*, sendo elas: cálculo do algoritmo de RSA para encriptação e decriptação e cálculo de hash e assinatura digital.

Funções:

- *Função: carregaChavePublica(Chave chavePublica)*
  Retorno: boolean
  Descrição: Carrega a chave pública no coprocessador.

- *Função: carregaChavePrivada(Chave chavePrivada)*
  Retorno: boolean
  Descrição: Carrega a chave privada no coprocessador.

- *Função: rsaChavePublica(byte[] mensagem, Chave chavePublica)*
  Retorno: byte[] (texto encriptado)
  Descrição: Executa o algoritmo de RSA utilizando a chave pública passada. Nesse caso é realizado a encriptação da mensagem.

- *Função: rsaChavePublica(byte[] mensagem)*
  Retorno: byte[] (texto encriptado)
  Descrição: Executa o algoritmo de RSA utilizando a chave pública armazenada no coprocessador.

- *Função: rsaChavePrivada(byte[] mensagem)*
  Retorno: byte[] (texto claro ou mensagem assinada)
  Descrição: Executa o algoritmo de RSA utiliando a chave privada. Com esse método é possível decriptar e assinar um mensagem.

- *Função: calculaHash(byte[] mensagem)*
  Retorno: byte[] (resumo da mensagem)
  Descrição: Calcula resumo da mensagem.

FuncoesSW: apresenta as funções que forma implementadas em *software*. As funções delegadas a software foram a geracão de chaves criptográficas e o cálculo e verificação da assinatura digital. O dois últimos foram desenvolvidos, também por software, para permitir a comparação com os valores calculados pelo coprocessador.

Funções:

- *Função: gerarChaves(int Nbits)*
  Retorno: Chave (chave pública)
  Descrição: Gera par de chaves criptográficas. Retorna a chave pública e a privada é armazenada no coprocessador. Todas as operações que necessitarem do uso de chave privada, utilizarão esse valor.

Email: classe responsável pelo envio de *email*, assim como a padronização das mensagens segundo o formato definido pelo *software* de demonstração.

Funções:

- *Função: sendMail(String from, String to, String subject, String message)*
  Retorno: void
  Descrição: Função responsável pelo envio de email.

- *Função: enviaEmailFormatado(String from, String to, String subject, String message)*
  Retorno: boolean
  Descrição: Formata as mensagens passadas segundo os padrões do bloco e envia a mensagem.

Chave: armazena informações relacionadas às chaves criptográficas (pública ou privada) e ao algoritmo de exponenciação modular (Constante C).

USBSW: apresenta as funções para acesso ao *hardware*, assim como as padronizações e sincronismo necessários. Descrito mais detalhadamente em *Driver USB*.

Funções:

- *Função: enviaMensagem(byte[] mensagem, int idMensagem)*
  Retorno: byte[]
  Descrição: Envia mensagem para o coprocessador e define o tipo da mensagem segundo o idMensagem recebido. Os tipos das mensagens estão definidos em *Especificação do Projeto - Hardware* , na tabela VII.2.
- *Função: enviaChavePublicaUSB(Chave chavePublica)*
  Retorno: boolean
  Descrição: Carrega a chave pública no coprocessador.
- *Função: enviaChavePrivadaUSB(Chave chavePrivada)*
  Retorno: boolean
  Descrição: Carrega a chave privada no coprecessador.
- *Função: RSAChavePublica(byte[] textoClaro)*
  Retorno: byte[]
  Descrição: Envia a mensagem e solicita ao coprocessador a execução do algoritmo RSA com chave pública.
- *Função: RSAChavePrivada(byte[] textoClaro)*
  Retorno: byte[]
  Descrição: Envia a mensagem e solicita ao coprocessador a execução do algoritmo RSA com chave privada.
- *Função: calculaHash(byte[] mensagem)*
  Retorno: byte[]
  Descrição: Solicita ao *hardware* o cálculo da função criptográfica.
- *Função: enviaParaUSB(byte[] valor)*
  Retorno: void
  Descrição: Envia via USB um vetor de bytes já previamente formatado em *Little-Endian* com os bytes menos significativos alocados nos menores índices do vetor. O envio pela porta USB também segue esta ordem, o bytes menos significativos são enviados primeiro.

- Função: recebeDaUSB()

  Retorno: byte[]

  Descrição: Faz a leitura de dados pela porta USB e carrega os valores em um vetor de bytes no formato *Little-Endian*, ou seja, os bytes menos significativos ocupam as posições de menor índice no vetor.

- Função: assina(byte[] textoClaro)

  Retorno: byte[]

  Descrição: Monta um vetor de bytes em formato *Little-Endian* contendo o código da operação Assinar ocupando os dois primeiros bytes do vetor e o envia à porta USB com a função enviaParaUSB.

- Função: decodificarOP(byte valor)

  Retorno: int

  Descrição: Decodifica os 2 primeiros bytes de uma mensagem de retorno da porta USB. Os 2 primeiros bytes contém o significado da mensagem.

- Função: InitializeDevice(String arquivoBinario)

  Retorno: boolean

  Descrição: Inicia e configura o dispositivo FPGA transferindo o arquivo binário de configuração indicado pelo parâmetro arquivoBinario. Este parâmetro contém o endereço do arquivo de configuração (e.g C:\MeuArquivos \Poli \coprocessadorrsa.bit).

- Função: InitializeDeviceFromEEPROM()

  Retorno: boolean

  Descrição: Inicia e configura o dispositivo FPGA considerando que o arquivo binário de configuração já foi previamente carregado na EEPROM da XEM3005.

Apêndice III – FUNÇÕES DO DRIVER USB

Funções que fazem as requisições para o coprocessador:

- enviaChavePublicaUSB - envia a chave pública gerada em software (a geração de chaves não está implementada em hardware, para maiores detalhes ver item *Software Com Funções Criptográficas* para ser armazenada no coprocessador;
- enviaChavePrivadaUSB - envia a chave privada gerada em software (a geração de chaves não está implementada em hardware, para maiores detalhes ver item *Software Com Funções Criptográficas* para ser armazenada no coprocessador;
- RSAChavePublica - envia a mensagem a ser encriptada e requisita essa encriptação com a chave pública;
- RSAChavePrivada - envia a mensagem a ser encriptada e requisita essa encriptação com a chave privada;
- calculaHash - envia a mensagem e solicita que seja calculado o resumo dessa;
- assina - envia a mensagem e solicita a assinatura com a chave privada armazenada no coprocessador;

Funções que auxiliam nas requisições para o coprocessador:

- enviaMensagem - chamada por todas as funções do tipo anterior. É responsável por montar o cabeçalho da mensagem a ser enviada, e então encaminhar o pacote para outra função que realiza o envio;

Funções que realizam a comunicação com o coprocessador:

- enviaParaUSB - utilizando funções da biblioteca fornecidas pela Opal Kelly, essa função tem por objetivo receber o pacote pronto da função *enviaMensagem* e então transmiti-lo via porta USB para o coprocessador;

- recebeDaUSB - utilizando funções da biblioteca fornecidas pela Opal Kelly, essa função tem por objetivo receber um pacote do coprocessador criptográfico, interpretar seu cabeçalho através de outra função (a ser descrita mais adiante), para então saber o resultado da última requisição e poder iniciar algum tratamento, exibição de mensagem de erro ou exibição do resultado obtido;

Funções auxiliares:

- decodificarOP - chamada pela função *recebeDaUSB*, e tem como objetivo interpretar o cabeçalho do pacote recebido pela porta USB e então encaminhar o pacote para o tratamento devido;

Funções de inicialização:

- InitializeDevice - responsável por configurar a FPGA (gravar o programa de descrição de *hardware* na mesma) através da interface USB com um arquivo padrão armazenado no PC, para isso se utiliza de funções da biblioteca fornecida pela Opal Kelly. Também é a responsável por criar o *link* de comunicação USB entre o coprocessador e o PC.

- InitializeDeviceFromEEPROM - responsável por configurar a FPGA (gravar o programa de descrição de *hardware* na mesma) com o arquivo armazenado na EEPROM da placa onde se encontra a FPGA (XEM3005), para isso se utiliza de funções da biblioteca fornecida pela Opal Kelly. Também é a responsável por manter o *link* de comunicação USB entre o coprocessador e o PC.

Apêndice IV – SINAL DO HARDWARE DA INTERFACE USB

A seguir é apresentada a descrição de cada sinal.

- **HI_IN** - Sinais de entrada vindos do controlador USB
- **HI_OUT** - Sinais de saída que vão para o controlador USB
- **HI_INOUT** - Sinais bidirecionais conectados ao controlador USB
- **CLK** - Sinal de saída de 1 bit - *Clock* de sincronia do sistema. Gerado a partir do sinal de *clock* de sincronia da comunicação USB.
- **DADOS_ENTRADA** - Sinal de saída de 256 bits - Palavra de dados enviada para ser processada pelo bloco RSA.
- **OP_ENTRADA** - Sinal de saída de 16 bits - 16 bits que representam a operação a ser executada pelo bloco RSA.
- **DADOS_PRONTOS_ENTRADA** - Sinal de saída de 1 bit - Bit de sinalização. Quando os valores **DADOS_ENTRADA** e **OP_ENTRADA** estão disponíveis, o *bit* fica ativo após o recebimento de uma nova solicitação.
- **BUFF_SAIDA_LIDO** - Sinal de saída de 1 bit - Ativo após finalizar o envio de um conjunto de dados pela USB, indicando que o bloco Interface USB está pronto para enviar mais dados ao PC.
- **DADOS_SAIDA** - Sinal de entrada de 256 bits - Palavra de dados de 256 bits a ser enviada ao PC quando uma requisição de dados for feita.
- **OP_SAIDA** - Sinal de entrada de 16 bits - 16 bits que representam o resultado de uma operação executada pelo bloco RSA.

•**DADOS_PRONTOS_SAIDA** - Sinal de entrada de 1 bit - Bit de sinalização. Ativo quando houver um novo conjunto de dados pronto para ser enviado ao PC pela Interface USB.

•**BUFF_ENTRADA_LIDO** - Sinal de entrada de 1 bit - Ativo após o bloco que recebe os dados efetuar a leitura destes. Indica que o sistema está pronto para receber novos pedidos.

Quando uma nova operação é solicitada pelo software, o bloco Interface USB executa as seguintes ações:

1. verifica se o sinal **BUFF_ENTRADA_LIDO** está ativo indicando que o sistema está pronto para receber novos pedidos.
2. Recebe os valores de **OP_ENTRADA**.
3. Recebe os valores de **DADOS_ENTRADA**.
4. Ativa o sinal **DADOS_PRONTOS_ENTRADA**.

Na figura IV.1 está apresentada a carta de tempos para a operação de envio de dados do software para o hardware pela USB e na figura IV.2 para a operação de envio de dados do hardware para o software.

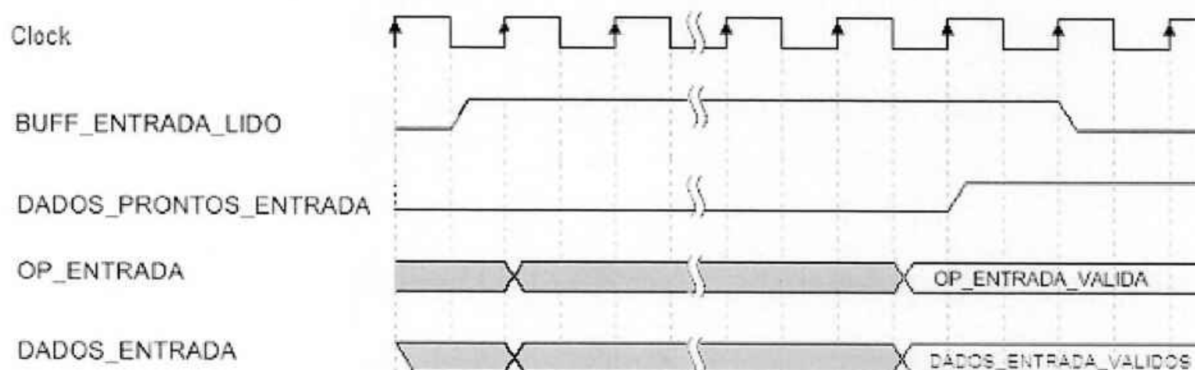

Figura IV.1: Carta de tempos com os sinais do bloco Interface USB durante o envio de dados do software para o hardware

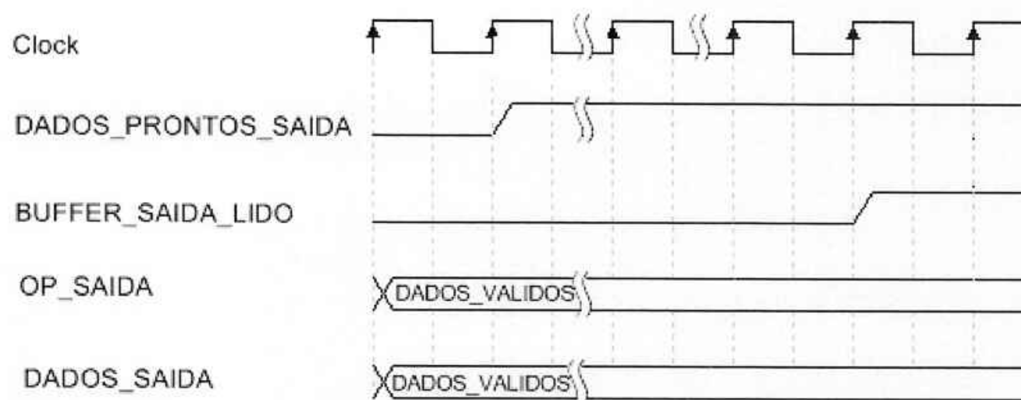

Figura IV.2: Carta de tempos com os sinais do bloco Interface USB durante o envio de dados do hardware para o software

Apêndice V – ELEMENTOS DISPONIBILIZADOS PELA OPALKELLY

okHostInterface

Este módulo é fornecido pela empresa Opal Kelly juntamente com o kit de desenvolvimento para o qual este projeto é especificado. Ele deve ser obrigatoriamente instanciado para permitir o uso da interface USB e dos módulos terminais (**endpoint**) disponibilizados. O módulo okHostInterface faz a conexão com os sinais vindos do controlador USB presente no kit de desenvolvimento e envia dados aos demais módulos terminais utilizados.

A figura V.1 mostra as entradas e saídas deste módulo.

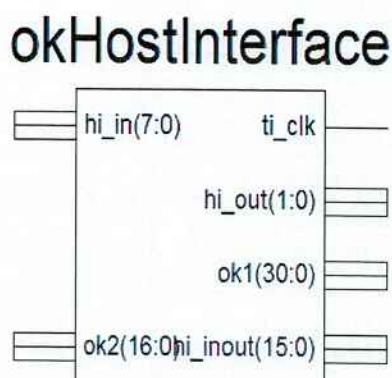

Figura V.1: Sinais de entrada e saída do módulo okHostInterface

Os sinais ok1 e ok2 são utilizados respectivamente para envio e recebimento de dados aos terminais e o sinal **TI_CLOCK** é uma cópia do sinal de *clock* da comunicação USB (48MHz).

okPipeIn

O módulo okPipeIn, também disponibilizado pela Opal Kelly, é um terminal que representa um *pipeline* de entrada de dados. Ele disponibiliza dados em sequência na forma de palavras de 16 bits. Este módulo é utilizado como um meio de enviar múltiplos bytes de forma síncrona a partir

do software em execução em um PC. O sinal **EP_WRITE**, quando ativo, indica que existem novos dados sendo enviados e disponíveis em **EP_DATAOUT**. Quando o sinal **EP_WRITE** permanece ativo por ciclos de *clock* consecutivos, o conteúdo de **EP_DATAOUT** é atualizado a cada *clock*. Desta forma, o uso do módulo okPipeln torna necessária a existência de um módulo subsequente que faça o devido tratamento e armazenamento dos dados recebidos em **EP_DATAOUT**.

A figura V.2 apresenta um diagrama de tempos que mostra como os dados são disponibilizados na saída deste módulo. **EP_DATAOUT** contém dados válidos para qualquer ciclo de *clock* em que **EP_WRITE** está ativo durante a borda de subida de **TI_CLK**. No exemplo, 4 palavras de 16bits são transferidas.

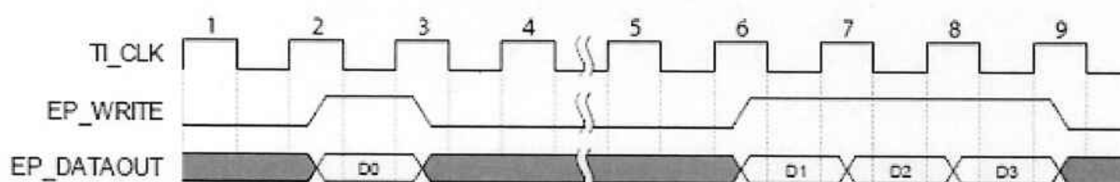

Figura V.2: Diagrama de tempos do módulo okPipeln

okPipeOut

O módulo okPipeOut, também disponibilizado pela Opal Kelly, funciona de modo análogo ao módulo okPipeln. É um terminal que representa um *pipeline* de saída de dados e envia dados em sequência na forma de palavras de 16 bits. Este módulo é utilizado como um meio de enviar múltiplos bytes de forma síncrona a partir do hardware em execução no FPGA. O sinal **EP_READ**, quando ativo, indica a solicitação por novos dados por parte do software em execução no PC e estes devem ser disponibilizados em **EP_DATAIN** na próxima borda de subida de **TI_CLK**. Como a comunicação USB é controlada pelo PC, os dados só podem ser enviados de forma passiva por parte do hardware, sempre que uma solicitação é feita. Não é possível enviar dados ativamente a partir do FPGA. Quando o sinal **EP_READ** permanece ativo por ciclos de *clock* consecutivos, o conteúdo de **EP_DATAIN** deve ser atualizado a cada ciclo de *clock*. Desta forma, o uso do módulo okPipeOut torna necessária a existência de um módulo subsequente que faça a atualização dos dados enviados a partir de **EP_DATAIN** a cada ciclo de *clock*.

A figura V.3 apresenta um diagrama de tempos que mostra como os dados são enviados com este módulo. **EP_DATAIN** deve conter dados válidos na borda de subida de **TI_CLK** subsequente à ativação de **EP_READ**. No exemplo, 4 palavras de 16bits são transferidas.

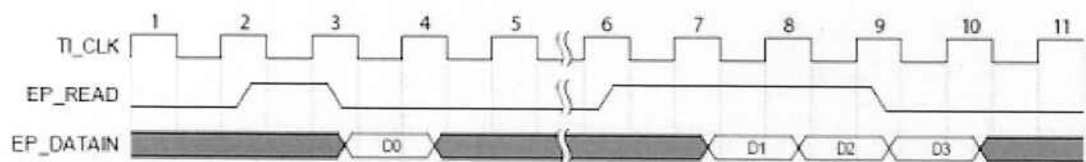

Figura V.3: Diagrama de tempos do módulo okPipeOut

Apêndice VI – FUNÇÕES DA BIBLIOTECA OKJFRONTPANEL

A seguir, as principais funções usadas da biblioteca **okjFrontPanel.jar** para as tarefas descritas no capítulo de Implementação:

- Função: GetDeviceMajorVersion()
  Retorno: int
  Descrição: Devolve primeira parte da versão do *firmware* que está na FPGA

- Função: GetDeviceMinorVersion()
  Retorno: int
  Descrição: Devolve segunda parte da versão do *firmware* que está na FPGA

- Função: GetSerialNumber()
  Retorno: String
  Descrição: Devolve *Serial Number* da FPGA

- Função: GetDeviceID()
  Retorno: String
  Descrição: Devolve *ID* da placa FPGA

- Função: ConfigureFPGA(String binaryFile)
  Retorno: ErrorCode
  Descrição: Carrega arquivo de configuração .bit na FPGA, configurando-a segundo um programa feito em linguagem de descrição de *hardware*. O caminho para acesso ao arquivo .bit precisa ser passado na chamada da função.

- Função: IsFrontPanelEnabled()
  Retorno: boolean
  Descrição: Verifica se o **FrontPanel** é suportado na configuração carregada na FPGA

Apêndice VII - BLOCO RSA

A seguir é apresentada a descrição de cada sinal:

- **CLK** - *Clock* de sincronia do sistema
- **DADOS_ENTRADA** - Sinal de entrada de 256 bits - Palavra de dados a ser processada pelo bloco RSA.
- **OP_ENTRADA** - Sinal de entrada de 16 bits - Palavra que representam a operação a ser executada pelo bloco RSA de acordo com a tabela VII.2 .
- **DADOS_PRONTOS_ENTRADA** - Sinal de entrada de 1 bit - Bit de sinalização. Indica a solicitação de uma nova operação. Se o bloco RSA estiver disponível, a execução da operação solicitada é iniciada.
- **BUFF_SAIDA_LIDO** - Sinal de entrada de 1 bit - Quando ativo, indica que o ultimo bloco de dados disponibilizado em **DADOS_SAIDA** e **OP_SAIDA** já foram lidos.
- **DADOS_SAIDA** - Sinal de saída de 256 bits - Palavra de dados de 256 bits resultante da última operação executada.
- **OP_SAIDA** - Sinal de saída de 16 bits - 16 bits que descrevem o conteúdo de **DADOS_SAIDA**.
- **DADOS_PRONTOS_SAIDA** - Sinal de saída de 1 bit - Bit de sinalização. Ativo ao término da execução de uma operação solicitada indicando que os dados em **OP_SAIDA** e **DADOS_SAIDA** são válidos.
- **BUFF_ENTRADA_LIDO** - Sinal de saída de 1 bit - Ativo após serem lidos os dados em **OP_ENTRADA** e **DADOS_ENTRADA**. Indica que o bloco anterior pode disponibilizar a próxima solicitação enquanto a última operação solicitada é executada.

Nº do bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0(LSB)
Conteúdo	T	T	T	T	T	T	T	T	T	T	T	T	OP	OP	OP	OP

Tabela VII.1: Formato de Dados da palavra de operação de entrada

Código	Operação
0000	Nada
0001	Vazio
0010	Ler E - Pública
0011	Ler M - Pública
0100	Hash
0101	Gerar chaves
0110	Assina
0111	Verifica assinatura
1000	Encriptar com Chave publica
1001	Carregar C - Pública
1010	Carregar E - Pública
1011	Carregar M - Pública
1100	Encriptar com Chave privada
1101	Carregar C - Privada
1110	Carregar E - Privada
1111	Carregar M - Privada

Tabela VII.2: Códigos de Operações solicitadas pelo software

Os dados recebidos em **OP_ENTRADA** segue o formato mostrado na figura VII.1

Os 4 bits menos significativos (bits OP) indicam a operação que está sendo solicitada de acordo com os valores indicados na tabela VII.2 . Os 12 bits mais significativos (bits T) indicam o tamanho do bloco de dados enviado. Embora o bloco RSA tenha sido desenvolvido para lidar apenas com blocos de 256 bits, de tal forma que apenas o bit 7 deve estar ativo, este formato permite modificações para implementações de blocos maiores até o limite de 4096 bits.

Os dados nos sinais **OP_SAIDA** e **DADOS_SAIDA** seguem o mesmo formato mostrado na figura VII.1. Os dados resultantes das operações executadas pelo bloco RSA são enviados no sinal **DADOS_SAIDA**. O sinal **OP_SAIDA**, contém códigos indicativos da situação do bloco RSA de acordo com a tabela VII.3.

A seguir é apresentada a descrição dos códigos de operação solicitadas pelo software ao bloco RSA que são listados na tabela VII.2:

- **Hash**- A operação Hash calcula um resumo criptográfico da mensagem, esse resumo tem um número definido de bits para qualquer mensagem, esse resumo após calculado

Código	Operação
0000	Ack
0001	Resultado
0010	Erro - Tamanho de chave inválido
0011	Erro - Operação inválida
0100	Erro durante execução
0101	Erro não definido
0110	Operação solicitada em execução
0111	Vazio
1000	Vazio
1001	Vazio
1010	Vazio
1011	Vazio
1100	Vazio
1101	Vazio
1110	Vazio
1111	Nenhuma operação solicitada

Tabela VII.3: Códigos de Operações devolvidos pelo hardware

é devolvido para o software através da USB.

•**Gerar chaves**- A operação de geração de chaves deve gerar um par de chaves pública e privada e armazená-las. O valor do Expoente (E) integrante da chave pública é devolvido junto com a sinalização de finalização da operação.

•**Ler E - Pública**- Esta operação disponibiliza na saída do bloco RSA o valor do Expoente (E) da chave pública armazenada internamente.

•**Ler M - Pública**- Esta operação disponibiliza na saída do bloco RSA o valor do Módulo (M) da chave pública armazenada internamente.

•**Carregar C - Pública**- Esta operação armazena o valor da constante $C = 2^{2*256} mod M$, referente à chave pública em uso, utilizada no cálculo de exponenciação modular. Esta opção é disponibilizada pois o cálculo da constante C é muito complexo e demorado para ser executado internamente.

•**Carregar E - Pública**- Esta operação armazena o valor do Expoente E, referente à chave pública em uso.

•**Carregar M - Pública**- Esta operação armazena o valor do Módulo M, referente à chave pública em uso.

- **Carregar C - Privada**- Esta operação armazena o valor da constante $C = 2^{2*256} mod M$, referente à chave privada em uso

- **Carregar E - Privada**- Esta operação armazena o valor do Expoente E, referente à chave privada em uso.

- **Carregar M - Privada**- Esta operação armazena o valor do Módulo M, referente à chave privada em uso.

- **Encriptar com Chave publica**- Esta operação inicia o algoritmo de encriptação RSA utilizando os valores da chave pública previamente armazenados. A informação a ser encriptada é recebida no sinal **DADOS_ENTRADA**

- **Encriptar com Chave privada**- Esta operação inicia o algoritmo de encriptação RSA utilizando os valores da chave privada previamente armazenados. A informação a ser encriptada é recebida no sinal **DADOS_ENTRADA**

- **Assina**- Esta operação executa em seqüência as operações de cálculo de *hash* e encriptação RSA com chave privada e tem como saída o valor de 256 bits da assinatura gerada.

- **Verifica assinatura**- Esta operação executa em sequência a operação de encriptação com chave pública em cima da assinatura recebida e compara com o valor de hash recebido.

A seguir é apresentada a descrição dos códigos de operação devolvidos pelo hardware listados natabela VII.3:

- **Ack** Valor devolvido para confirmar que uma operação solicitada foi recebida. Não necessariamente esta operação já foi iniciada.

- **Resultado** Valor devolvido quando última operação iniciada foi concluída. Para operações que retornam algum valor(e.g.,Ler E), tal valor está disponível em **DADOS_SAIDA**. Para operações que não retornam nenhum valor (e.g,Carregar E) o conteúdo de **DADOS_SAIDA** deve ser desconsiderado.

- **Erro - Tamanho de chave inválido** Valor devolvido quando o campo tamanho de chave contém um valor inválido.

- **Erro - Operação não identificada** Valor devolvido quando é feita uma solicitação utilizando um código de operação inválido ou um código de uma operação não implementada.
- **Erro durante execução** Valor devolvido quando um erro desconhecido ocorreu durante o processamento de uma operação solicitada.
- **Operação solicitada em execução**
- **Erro não definido** Valor devolvido quando um erro desconhecido ocorreu.
- **Operação solicitada em execução** Valor devolvido enquanto uma operação solicitada está em execução.
- **Nenhuma operação solicitada** Valor devolvido quando o bloco RSA está em um estado inicial e nenhuma operação foi solicitada.

Apêndice VIII – MULTIPLICADOR MODULAR

A seguir é apresentada a descrição de cada sinal:

- **CLK** - Sinal de entrada de 1 bit - *Clock* de sincronia do sistema .
- **START** - Sinal de entrada de 1 bit - A borda de subida indica a solicitação do início de um novo cálculo.
- **A** - Sinal de entrada de 256 bits - Fator A da multiplicação modular.
- **B** - Sinal de entrada de 256 bits - Fator B da multiplicação modular.
- **M** - Sinal de entrada de 256 bits - Módulo da multiplicação modular.
- **S** - Sinal de saída de 256 bits - Resultado do cálculo da multiplicação modular.
- **READY** - Sinal de saída de 1 bit - Ativo ao término do cálculo, indicando que o valor em **S** é válido.

A figura figura VIII.1 apresenta a carta de tempos deste componente

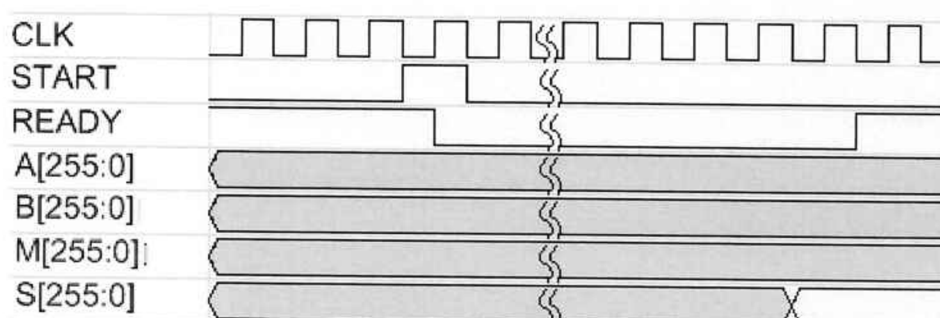

Figura VIII.1: Carta de tempos do bloco Multiplicador Modular

Apêndice IX – EXPONENCIADOR MODULAR

A seguir é apresentada a descrição de cada sinal:

- •**CLK** - Sinal de entrada de 1 bit - *Clock* de sincronia do sistema .
- •**C** - Sinal de entrada de 256 bits - Constante de exponenciação de Montgomery pré calculada em software.
- •**P** - Sinal de entrada de 256 bits - Valor da Base na exponenciação.
- •**E** - Sinal de entrada de 256 bits - Valor do Expoente na exponenciação.
- •**M** - Sinal de entrada de 256 bits - Valor do Módulo na exponenciação.
- •**S** - Sinal de saída de 256 bits - Resultado do cálculo da exponenciação.
- •**START** - Sinal de entrada de 1 bit - A borda de *sub*ida indica a solicitação do início de um novo cálculo.
- •**READY** - Sinal de entrada de 1 bit - Ativo ao término do cálculo, indicando que o valor em **S** é válido.

A figura IX.1 apresenta a carta de tempos deste componente.

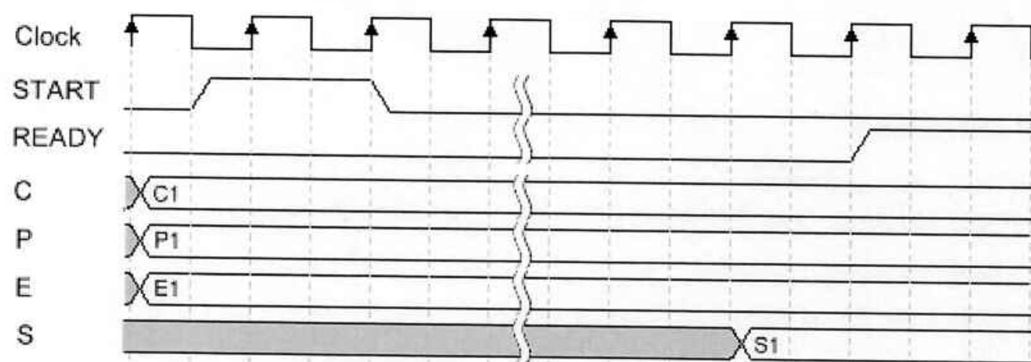

Figura IX.1: Carta de tempos do bloco Exponenciador Modular