Maurício Araújo Dias

UM SISTEMA CRIPTOGRÁFICO PARA CURVAS ELÍPTICAS SOBRE GF(2^m) IMPLEMENTADO EM CIRCUITOS PROGRAMÁVEIS

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Banca Examinadora:
Prof. Dr. José Raimundo de Oliveira – UNICAMP
Prof. Dr. Ricardo Dahab – UNICAMP
Prof. Dr. Furio Damiani – UNICAMP
Prof. Dr. Peter Jürgen Tatsch – UNICAMP
Prof. Dr. Norian Marranghello – UNESP

Campinas, SP
2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

D543u	Dias, Maurício Araújo Um sistema criptográfico para curvas elípticas sobre GF (2^m) implementado em circuitos programáveis / Maurício Araújo Dias. --Campinas, SP: [s.n.], 2007. Orientador: José Raimundo de Oliveira. Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação. 1. Criptografia. 2. Circuitos digitais. 3. Curvas elípticas. 4. Hardware. 5. Circuitos integrados. 6.VHDL (Linguagem descritiva de hardware). I. Oliveira, José Raimundo. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: A cryptosystem for elliptic curves over GF(2^m) implemented in FPGAs.
Palavras-chave em Inglês: Point doubling, Point addition, Combinatorial circuit, Cryptography, Elliptic curve, FPGA.
Área de concentração: Engenharia de Computação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Ricardo Dahab, Furio Damiani, Peter Jürgen Tatsch e Norian Marranghello.
Data da defesa: 28/02/2007
Programa de Pós-Graduação: Engenharia Elétrica

Resumo. Este trabalho propõe um sistema criptográfico para Criptografia baseada em Curvas Elípticas (ECC). ECC é usada alternativamente a outros sistemas criptográficos, como o algoritmo RSA (Rivest-Shamir-Adleman), por oferecer a menor chave e a maior segurança por bit. Ele realiza multiplicação de pontos ($Q = kP$) para curvas elípticas sobre corpos finitos binários. Trata-se de um criptosistema programável e configurável. Graças às propriedades do circuito programável (FPGA) é possível encontrar soluções otimizadas para diferentes curvas elípticas, corpos finitos e algoritmos. A característica principal deste criptosistema é o uso de um circuito combinacional para calcular duplicações e adições de pontos, por meio da aritmética sobre corpos finitos. Os resultados deste trabalho mostram que um programa de troca de chaves fica aproximadamente 20.483 vezes mais rápido com a ajuda do nosso sistema criptográfico. Para desenvolver este projeto, nós consideramos que o alto desempenho tem prioridade sobre a área ocupada pelos seus circuitos. Assim, nós recomendamos o uso deste circuito para os casos em que não sejam impostas restrições de área, mas seja exigido alto desempenho do sistema.

Palavras-chave: Criptografia, Curvas Elípticas, Circuito Combinacional, FPGA Duplicação de Ponto, Adição de Pontos, Inversão Modular, Algoritmo de Stein.

Abstract. This work proposes a cryptosystem for Elliptic Curve Cryptography (ECC). ECC has been used as an alternative to other public-key cryptosystems such as the RSA (Rivest-Shamir-Adleman algorithm) by offering the smallest key size and the highest strength per bit. The cryptosystem performs point multiplication ($Q = kP$) for elliptic curves over binary polynomial fields ($GF(2^m)$). This is a programmable and scalable cryptosystem. It uses the abilities of reconfigurable hardware (FPGA) to make possible optimized circuitry solutions for different elliptic curves, finite fields and algorithms. The main feature of this cryptosystem is the use of a combinatorial circuit to calculate point doublings and point additions, through finite field arithmetic. The results of this work show that the execution of a key-exchange program is, approximately, 20,483 times faster with the help of our cryptosystem. To develop this project we considered that high-performance has priority over area occupied by its circuit. Thus, we recommend the use of this circuit in the cases for which no area constraints are imposed but high performance systems are required.

Keywords: Cryptography, Elliptic Curve, Combinatorial Circuit, FPGA, Point Doubling, Point Addition, Modular Inversion, Stein's Algorithm.

Índice Geral

Índice de Figuras

Índice de Tabelas

Índice de Gráficos

Trabalhos do Autor Relacionados com a Tese

[DO03] M. A. Dias e J. R. Oliveira. “Elliptic Curve Algorithms: An Analysis of Cost Between Operations of the Scalar Multiplication”. *Accepted for Publication in the Proceedings of the Technical Track of the ACNS'03, ICISA Press*, Kunming, China, mauricioaraujodias@hotmail.com, October 2003.

[DO1-07] M. A. Dias e J. R. Oliveira. “An Inverter Architecture for ECC-GF(2^m) Based on the Stein’s Algorithm”. 3rd *International Workshop on Boolean Functions Cryptography and Applications (BFCA’07)*, Paris, France, May 2007.

[DO2-07] M. A. Dias e J. R. Oliveira. “Speeding Up Point Doublings and Point Additions for ECC-GF(2^m)”. *Relatório Técnico DCA*, mauricioaraujodias@hotmail.com, 2007.

Glossário

ANSI (*American National Standards Institute*) – Instituto Nacional Americano de Padrões (dos EEUU).

ASIC (*Application-Specific Integrated Circuit*) – um circuito integrado projetado para implementar uma função específica.

CHES *(Cryptographic Hardware and Embedded Systems) – workshop* que apresenta o estado da arte em sistemas criptográficos implementados em *hardware*.

CPU (*Central Processing Unit*) – Unidade Central de Processamento.

Criptoanálise (*cryptoanalysis*) – processo usado para decifrar mensagens criptografadas.

ECC (*Elliptic Curve Cryptography*) – criptografia baseada em curvas elípticas.

ECC-GF(2^m) (*Elliptic Curve Cryptography – Binary Galois Field*) – criptografia baseada em curvas elípticas sobre corpos finitos binários (corpos de Galois).

FPGA (*Field Programmable Gate Array)* – é um circuito lógico-programável, isto é, uma classe de *chips* com propósito geral que pode ser configurado para uma variedade de aplicações.

GF(2^m) (*Binary Galois Field*) – corpo finito binário.

HACKER *(Hacker)* – criptoanalista; pessoa que tenta decifrar mensagens criptografadas.

HDL *(Hardware Description Language)* – linguagem de descrição de *hardware*.

IEEE (*Institute of Electrical and Eletronics Engineers*) – Instituto de Engenheiros Eletricistas e Eletrônicos.

IPSec (*Internet Protocol Security*) – Segurança do Protocolo IP.

ISA (*Industry Standard Architecture*) – barramento formado por slots que trabalham com palavras de 8 ou 16 bits.

LUT (*LookUp Table*) – Tabela de LookUp.

NIST (*National Institute of Standards and Technology*) – Instituto Nacional de Padrões e Tecnologia (dos EEUU).

PC (*Personal Computer*) – Computador pessoal.

PCI (*Peripheral Component Interconnect*) – Padrão de conexão para componentes periféricos de PCs.

RAM (*Random Access Memory*) – memória de acesso aleatório.

RNG *(Random Number Generator)* – Gerador de números aleatórios.

RSA (*Rivest-Shamir-Adleman*) – um algoritmo de criptografia assimétrica.

VHDL (***V****ery* ***H****igh speed integrated circuit hardware* ***D****escription* ***L****anguage*) – uma linguagem de descrição de *hardware*.

VLSI (*Very Large Scale Integration*) – integração em escala muito larga.

WAP (*Wireless Application Protocol*) – Protocolo para Aplicações sobre Redes sem Fio.

Capítulo 1
Introdução Geral

1.1 Considerações iniciais

Órgãos governamentais, instituições financeiras, agências militares, dentre outras organizações, necessitam garantir o sigilo de suas informações, que trafegam por redes de computadores, pois essas redes podem estar sujeitas à espionagem.

Um código secreto, quando associado a uma informação, pode permitir que a mesma seja transmitida de forma segura, através de um meio compartilhado.

A associação desse código (também chamado de chave secreta) aos dados transmitidos por uma rede é tarefa dos algoritmos de criptografia simétrica. Para maiores detalhes sobre algoritmos de criptografia simétrica, vide [T96] e [S98].

Todas as entidades (um cliente e um servidor, por exemplo) envolvidas em uma comunicação de dados criptografados precisam conhecer simultaneamente a chave secreta. Compartilhar essa chave através da rede sujeita à espionagem não é seguro.

Alternativamente, pode-se empregar, por exemplo, um modelo de troca de chaves como o Diffie-Hellman (descrito em [HMV04]) para garantir a geração da chave secreta, tanto no lado cliente como no servidor, sem necessidade de transmiti-la pela rede.

Esse modelo está inserido no contexto da criptografia assimétrica para a qual pode-se considerar:

- *cli* – o lado cliente da comunicação;
- *serv* – o lado servidor da comunicação;
- *P* – um ponto (vide Capítulo 2) previamente escolhido por *cli* e *serv*;

- k – uma chave privada;
- Q_P – uma chave pública;
- Q_S – a chave secreta;
- Q_P, Q_S e P possuem a mesma origem.

Para maiores detalhes sobre algoritmos de criptografia assimétrica vide [T96] e [S98].

A troca de chaves através desse modelo é realizada da seguinte forma:

1. O *cli* gera aleatoriamente um número inteiro k_{cli};

   O *serv* gera aleatoriamente um número inteiro k_{serv};

2. O *cli* calcula:

   $$Q_{Pcli} = k_{cli}\, P \qquad (1.1);$$

   O *serv* calcula:

   $$Q_{Pserv} = k_{serv}\, P \qquad (1.2);$$

3. O *cli* transmite Q_{Pcli} para o *serv*;

   O *serv* transmite Q_{Pserv} para o *cli*;

4. O *cli* calcula:

   $$Q_S = k_{cli}\, Q_{Pserv} \qquad (1.3);$$

   O *serv* calcula:

   $$Q_S = k_{serv}\, Q_{Pcli} \qquad (1.4);$$

Ao final dos cálculos, o cliente e o servidor alcançam o mesmo resultado Q_S. Em outras palavras, cada um gera, separadamente, a mesma chave secreta, como mostrado pelas equações de (1.5) a (1.7), a seguir:

$$\begin{array}{cccc} & cli & & serv \\ Q_S = k_{cli}\, Q_{Pserv} & = & Q_S = k_{serv}\, Q_{Pcli} & (1.5) \\ Q_S = k_{cli}\, (k_{serv}\, P) & = & Q_S = k_{serv}\, (k_{cli}\, P) & (1.6) \\ Q_S = (k_{cli}\, k_{serv})\, P & = & Q_S = (k_{serv}\, k_{cli})\, P & (1.7) \end{array}$$

Como Q e P possuem a mesma origem, a equação $Q = kP$ representa a operação fundamental para a realização da troca de chaves dentro do modelo.

Para o trabalho desta tese, pode-se calcular $Q = kP$ por intermédio de qualquer algoritmo de criptografia baseado em curvas elípticas sobre corpos finitos binários. Uma introdução de curvas elípticas sobre corpos finitos é apresentada no capítulo 2. Maiores detalhes sobre este tema podem ser encontrados em [HMV04], [BSN99] ou em [M93].

Quando a execução do algoritmo citado for auxiliada por um *hardware*, seu processo é agilizado, permitindo uma troca mais rápida de chaves. Esse aumento de velocidade viabiliza a substituição da chave secreta com maior freqüência.

Como o alvo do *hacker (*aquele que decifra mensagens criptografadas*)*, em muitos casos, costuma ser a chave secreta, mudanças freqüentes dessa mesma chave dificultam o seu trabalho, proporcionando maior segurança às informações.

Dizendo de outro modo, quando o *hacker* desvendar o código secreto da chave, esta já terá sido trocada.

Além disso, identificar o conteúdo de uma mensagem criptografada por uma única chave secreta já é uma tarefa difícil para qualquer *hacker*. Se porções diferentes dessa mesma mensagem estiverem cada uma criptografada com uma chave secreta distinta, o trabalho do *hacker* torna-se quase impossível.

1.2 Objetivo

Este trabalho tem como objetivo e motivação o desafio de propor a criação de um sistema criptográfico de alto desempenho que trabalhe com coordenadas afins (vide Capítulo 2). Este objetivo atende a uma fatia específica de aplicação, que necessita empregar sistemas criptográficos que permitam troca frequente de chaves.

Com este objetivo em mente, durante o projeto, não impusemos restrições de área de circuito. O sistema desenvolvido é implementado com circuitos lógico-programáveis – *FPGAs* (descritos em [VCC00]) – e executa um algoritmo de *ECC-GF(2^m)* com o auxílio de dois circuitos combinacionais – um circuito para a inversão modular e outro para a duplicação e a adição de pontos de uma curva elíptica (vide Capítulos 3 e 4). O emprego desse *hardware* permite alta freqüência na troca de chaves para, por exemplo, um sistema cliente-servidor que usa o modelo Diffie-Hellman.

1.3 Organização deste Trabalho

Seguem, neste trabalho, sete capítulos e três apêndices. Nos capítulos, o sistema é apresentado desde os seus aspectos mais globais até os seus detalhes de construção.

Os capítulos ainda discutem os resultados obtidos a partir da simulação dos circuitos que compõem o sistema. Os apêndices apresentam informações importantes relacionadas a este trabalho.

O Capítulo 2 introduz o leitor aos conhecimentos de criptografia baseada em curvas elípticas sobre corpos finitos binários, necessários à compreensão do restante deste trabalho.

O Capítulo 3 apresenta um circuito combinacional de inversão modular para curvas elípticas sobre corpos finitos binários, baseado no algoritmo de Stein (descrito em [S67]).

O Capítulo 4 descreve o desenvolvimento de um circuito combinacional, usado para executar as operações de duplicação e adição de pontos de uma curva elíptica sobre corpos finitos binários.

O Capítulo 5 apresenta um sistema criptográfico para curvas elípticas sobre corpos finitos binários. Este sistema foi implementado em uma placa adaptadora para *PC*, como exemplificação.

O Capítulo 6 comenta os resultados obtidos pelas simulações e compilações deste sistema criptográfico.

O Capítulo 7 faz as considerações finais, incluindo comentários, conclusões e sugestões para trabalhos futuros.

Após o Capítulo 7, encontram-se as referências bibliográficas.

O apêndice A mostra o esquema elétrico completo de um bloco básico (fatia) usado para construir o circuito combinacional para a inversão modular, apresentado no Capítulo 3.

O apêndice B traz o código *VHDL* completo do circuito combinacional para as demais operações. Para uma introdução à VHDL vide [P99]. Este circuito e o circuito combinacional para a inversão modular, juntos, formam o circuito combinacional para a duplicação e a adição de pontos, conforme apresentado no Capítulo 4.

Capítulo 2
Criptografia Baseada em Curvas Elípticas sobre Corpos Finitos Binários – *ECC-GF(2^m)*

2.1 Introdução

Este capítulo introduz resumidamente alguns conceitos relativos às curvas elípticas. O leitor deve procurar [HMV04] para uma melhor compreensão sobre o assunto.

Para conseguirem êxito em suas funções, os algoritmos de criptografia assimétrica necessitam empregar chaves com muitos bits. O *RSA* (Rivest-Shamir-Adleman, como descrito em [T96]), por exemplo, emprega chaves com pelo menos 1024 bits. Já os algoritmos baseados em *ECC*, quando usam chaves de 160 bits, alcançam o mesmo grau de segurança do *RSA*, quando este último usa chaves de 1024 bits.

Embora ambos empreguem chaves com muitos bits, existe uma diferença significativa entre o número de bits das chaves do *RSA* e dos algoritmos baseados em *ECC*, para um mesmo nível de segurança. Esse menor número de bits desperta a atenção dos pesquisadores para as curvas elípticas, pois esses algoritmos oferecem um maior nível de segurança por bit.

Morain e Olivos afirmam em [MO89]: "(...) todos os algoritmos que usam curvas elípticas sobre corpos finitos requerem a computação do ponto kP sobre E, onde k é um número inteiro grande e P é um ponto sobre a curva elíptica E".

Dito de outra forma, encontrar o ponto $Q = kP$ constitui a operação fundamental de qualquer algoritmo de criptografia assimétrica baseado em *ECC*. A variável k representa um número inteiro da ordem de dezenas de bits, gerado aleatoriamente. Ela pode ser chamada de chave privada. Q e P são da mesma natureza, isto é, ambos são pontos definidos por um par de coordenadas $x, y \in GF(q)$ e pertencem a uma mesma curva elíptica E.

O ponto Q pode representar tanto uma chave pública (vide equações (1.1) e (1.2)), como uma chave secreta (vide equações (1.3) e (1.4)). P é um ponto pré-determinado. Já a curva E é representada pela equação:

$$y^2 + xy = x^3 + ax^2 + b \qquad (2.1)$$

onde: a e b são os parâmetros que definem a curva; x e y representam as coordenadas dos pontos que constituem a curva.

A operação $Q = kP$ pode ser realizada somando o ponto P, a ele mesmo, k vezes. Apesar de simples, esse é um método de baixo desempenho, devido ao número proibitivo de adições. Por isso, costuma-se empregar algoritmos que reduzem o número de adições para o cálculo do produto kP.

Uma alternativa é obter Q por meio de sucessivas duplicações de um ponto e adições de dois pontos distintos [IEEE99] e [DO03], conforme o valor dos bits da variável k – toda ocorrência de um bit 0 indica uma duplicação, enquanto toda ocorrência de um bit 1 indica uma duplicação seguida de uma adição.

A duplicação de um ponto pode ser compreendida considerando-se uma curva elíptica de parâmetros a e p (a e p pertencem ao corpo finito binário ($GF(2^m)$)), para a qual é dado um ponto $P'(P'_X, P'_Y)$, sendo $P' = P$, e um ponto $Q(Q_X, Q_Y) = 2P$, cujas coordenadas são expressas por [SEC1-00]:

$$S = P_X + ((P'_Y) / (P'_X)) \bmod p \qquad (2.2)$$

$$Q_X = (S^2 + S + a) \bmod p \qquad (2.3)$$

$$Q_Y = (S(P_X + Q_X) + P_Y + Q_X) \bmod p \qquad (2.4)$$

Já a adição de dois pontos distintos pode ser compreendida considerando-se a mesma curva anterior, para a qual a adição entre seus pontos $P(P_X, P_Y)$ e $P'(P'_X, P'_Y)$ resulta em um ponto $Q(Q_X, Q_Y)$, cujas coordenadas são expressas por [SEC1-00]:

$$S = ((P_Y + P'_Y) / (P_X + P'_X)) \bmod p \quad (2.5)$$

$$Q_X = (S^2 + S + P_X + P'_X + a) \bmod p \quad (2.6)$$

$$Q_Y = (S(P_X + Q_X) + P_Y + Q_X) \bmod p \quad (2.7)$$

As operações de duplicação e adição efetuadas com pontos pertencentes a uma determinada curva elíptica sobre um corpo finito permitem calcular um ponto Q também pertencente a essa mesma curva.

Conforme é possível observar nas equações de (2.2) a (2.7), tanto a duplicação de um ponto como a adição de dois pontos é realizada por intermédio do cálculo de um conjunto de operações mais elementares descritas em [WBVGV96]. Elas são a adição modular, o quadrado, o módulo, a multiplicação e a divisão modular, todas efetuadas sobre elementos pertencentes a um dado corpo finito. Cada uma delas está descrita detalhadamente em [LN94], constituindo a chamada aritmética sobre corpos finitos.

Este capítulo prossegue da seguinte maneira: a seção 2.2 comenta a origem do circuito combinacional para a inversão modular; a seção 2.3 explica a origem do circuito combinacional para a duplicação e a adição de pontos. O capítulo é encerrado com os comentários finais na seção 2.4.

2.2 A Origem do Circuito Combinacional para a Inversão Modular

A divisão modular é a operação mais complexa e lenta dentro do conjunto abrangido pela aritmética sobre corpos finitos binários. Ela aparece apenas duas vezes nas equações de (2.2) a (2.7), porém o tempo gasto para executá-la uma única vez supera a soma dos tempos requeridos para calcular todas as demais operações. Dessa forma, sendo o alto desempenho uma prioridade no projeto deste sistema criptográfico, encontrar meios de desenvolver um circuito menos complexo, a fim de executar a divisão modular de forma mais rápida, torna-se uma necessidade.

Tendo isso em mente, deve-se considerar que uma operação de divisão pode ser substituída por uma inversão seguida de uma multiplicação. Mostrando de outra maneira, $P'_Y / P'_X \bmod p$ é equivalente a $P'_Y * P'^{-1}_X \bmod p$ ou ainda a $P'_Y * (1 / P'_X)$ mod p. Isso é

possível pois a inversão é um caso particular da própria divisão, para a qual o dividendo é uma constante de valor igual a um.

Assim sendo, pode-se usar um algoritmo que calcula uma operação de divisão modular também para calcular uma operação de inversão modular. A diferença está apenas na questão do dividendo possuir um valor constante igual a um para o caso da inversão modular.

É justamente esse valor constante e pequeno que torna possível projetar e construir um circuito para a inversão modular um pouco menos complexo, possibilitando, dessa forma, ganho em velocidade de execução da operação.

Com essa modificação, a operação de duplicação de um ponto *P* passa a ser expressa por:

$$S = P_X + ((P'_Y) * (P'_X)^{-1}) \bmod p \quad (2.8)$$

$$Q_X = (S^2 + S + a) \bmod p \quad (2.9)$$

$$Q_Y = (S(P_X + Q_X) + P_Y + Q_X) \bmod p \quad (2.10)$$

Considerando a mesma mudança, a operação de adição de dois pontos distintos passa a ser expressa por:

$$S = ((P_Y + P'_Y) * (P_X + P'_X)^{-1}) \bmod p \quad (2.11)$$

$$Q_X = (S^2 + S + P_X + P'_X + a) \bmod p \quad (2.12)$$

$$Q_Y = (S(P_X + Q_X) + P_Y + Q_X) \bmod p \quad (2.13)$$

2.3 A Origem do Circuito Combinacional para a Duplicação e a Adição de Pontos

Além da divisão modular, é possível otimizar também o restante das operações presentes nas equações de (2.8) a (2.13), a fim de projetar um circuito mais simples e com maior desempenho. Para tanto, faz-se necessário unificar as equações (2.8) e (2.11), (2.9) e (2.12)

e, finalmente, (2.10) e (2.13), de modo a obter um conjunto único de equações, conforme mostrado a seguir:

$$S = F + ((G + P'_Y) * (H + P'_X)^{-1}) \bmod p \qquad (2.14)$$

$$Q_X = (S^2 + S + P_X + P'_X + a) \bmod p \qquad (2.15)$$

$$Q_Y = (S(P_X + Q_X) + P_Y + Q_X) \bmod p \qquad (2.16)$$

onde P'_X e P'_Y representam as coordenadas do ponto a ser duplicado ou somado, P_X e P_Y representam as coordenadas do ponto P fixo, pré-estabelecido, e Q_X e Q_Y representam as coordenadas do ponto Q, o qual pode ser um resultado parcial ou final da equação $Q = kP$.

As equações de (2.14) a (2.16) são válidas tanto para a duplicação de ponto como para a adição de pontos. Apenas a equação (2.14) sofre variações: para a adição de pontos, F assume valor igual a 0, G é igual a P_Y e H é igual a P_X; para a duplicação de ponto, F é igual a P_X, enquanto G e H são zerados, isto porque, na duplicação, $P_X = P'_X$ e $P_Y = P'_Y$.

Essas equações permitem o desenvolvimento de um circuito combinacional capaz de realizar tanto a duplicação de um ponto, como a adição de dois pontos distintos.

2.4 Comentários Finais

Uma vez que algoritmos baseados em *ECC* trabalham com chaves de tamanho relativamente pequeno, eles podem ser implementados por circuitos menos complexos e mais rápidos, sem degradação do grau de segurança oferecido pelo sistema no qual eles são usados.

Qualquer algoritmo de *ECC* trabalha com chaves através da equação $Q = kP$. É possível obter Q por meio de sucessivas duplicações de um ponto e adições de dois pontos distintos. Estas, por sua vez, são realizadas por intermédio do cálculo de um conjunto de operações mais elementares: a adição modular, o quadrado, o módulo, a multiplicação e a divisão modular, sendo esta última a mais complexa e lenta dentre elas.

Todas essas operações podem ser realizadas de forma otimizada, a fim de projetar um circuito de duplicação e de adição mais simples e com maior desempenho, para manipulação de chaves em *ECC*.

Antes, porém, é preciso converter a divisão em uma inversão seguida de uma multiplicação e implementar exclusivamente a operação de inversão em um circuito combinacional.

Capítulo 3
Um Circuito Combinacional para a Inversão Modular

3.1 Introdução

Este capítulo descreve um circuito de inversão modular para criptografia assimétrica baseada em curvas elípticas sobre corpos finitos binários [DO1-06]. Em outras palavras, ele apresenta um circuito combinacional capaz de calcular x^{-1} *mod p* para a equação (2.14). Ele executa a operação de inversão modular, por meio do emprego de uma lógica baseada no algoritmo de Stein (descrito em [S67]). O algoritmo de inversão modular empregado aqui é semelhante ao algoritmo de divisão modular descrito por Wu e outros em [WWSH04].

Ele pode ser usado em conjunto com grande número de algoritmos de *ECC-GF*(2^m), sendo, esses últimos, preferencialmente implementados em *hardware*. Os valores de m escolhidos para a demonstração do funcionamento do dispositivo são 113, 131, 147 e 163. Para *GF*(2^{113}), *GF*(2^{131}) e *GF*(2^{163}) os parâmetros escolhidos para definir as curvas elípticas, assim como seus pontos, estão em conformidade com os padrões *NIST, IEEE P1363, IPSec, WAP, eCheck, ANSI X9.62* e *ANSI X9.63* [SEC2-00]. O corpo finito *GF*(2^{147}) foi escolhido aleatoriamente e não faz parte de nenhum padrão. Os demais corpos finitos foram escolhidos de modo a evitar a construção de um circuito excessivamente denso, visto que são os menores corpos finitos padronizados.

O circuito tem desempenho da ordem de nanossegundos e cabe em *FPGAs* comerciais. O projeto priorizou o alto desempenho em detrimento à área. Recomenda-se o emprego deste circuito combinacional para os casos em que seja exigido um sistema de alto desempenho, porém não haja restrições de área.

O circuito trabalha com pontos representados diretamente através de coordenadas afins, sem qualquer necessidade de conversão para coordenadas projetivas. Coordenadas projetivas são abordadas em [LD99].

O restante deste capítulo está dividido em seis seções descritas a seguir: a seção 3.2 faz um breve comentário a respeito de alguns trabalhos, direta ou indiretamente

relacionados com este projeto, escritos por outros autores; a seção 3.3 explica como a inversão modular é calculada através do algoritmo de Stein; a seção 3.4 mostra como as Mega-Funções da Altera são usadas para implementar a lógica do algoritmo empregado; a seção 3.5 comenta as implementações do circuito na *FPGA* Stratix II EP2S180F1020C4; a seção 3.6 apresenta os resultados obtidos a partir dessas implementações. O capítulo é encerrado com os comentários finais na seção 3.7.

3.2 Trabalhos Prévios

Objetivando diminuir a complexidade e melhorar o desempenho da inversão e da divisão, há tempos, pesquisadores trabalham na elaboração de novos algoritmos. Com esse propósito em mente, eles têm focado seu trabalho em três diferentes métodos para calcular essas operações sobre corpos finitos binários. Tais métodos são citados a seguir.

Nos anos oitenta, Itoh e Tsujii [IT88] propuseram um algoritmo para computar a inversão modular, para o qual era usada representação em base normal. A essência desse algoritmo era o cálculo $Y \cdot X^{2^m-2}$, acompanhando o Teorema de Fermat.

Na década seguinte, Hasan e Bhargava [HB92] apresentaram o algoritmo de um divisor bit-serial, com base na eliminação Gaussiana aplicada a um sistema de equações lineares.

Quando nós implementamos esses dois algoritmos como circuitos combinacionais, as complexidades dos circuitos ficaram muito acima do necessário para atingir os objetivos deste projeto. Assim, esses dois primeiros métodos citados não são usados neste trabalho. Resultados mais satisfatórios são atingidos, ao se considerar trabalhos apresentando algoritmos baseados em um terceiro método, citado a seguir.

De 2001 a 2003, Eberle, Gura, Shantz, Gupta, entre outros, publicaram diversos artigos [S01], [GSEGGFGS02] e [EGSG03] descrevendo implementações fundamentadas no cálculo do maior divisor comum (MDC). Elas realizavam a operação de divisão modular através de uma otimização do algoritmo de Euclides.

Essa versão do algoritmo de Euclides também pode ser implementada como um circuito combinacional, porém a área ocupada por ele cresce demasiadamente conforme

varia o corpo finito. Assim, sua implementação supera os limites de qualquer *FPGA* comercial atual.

Após exaustivas implementações de diversos algoritmos diferentes, chegamos à conclusão que a solução para este projeto seria continuar trabalhando com o cálculo do maior divisor comum, mas agora baseado em um outro algoritmo que nós implementamos, o de Stein. A primeira descrição desse algoritmo data de 1967, em um artigo escrito por J. Stein [S67].

Wu e outros [WWSH04] publicaram, em 2004, uma importante otimização do algoritmo de Stein. Nela, a onerosa comparação entre graus de polinômios foi completamente eliminada.

Em outro artigo, do ano de 2005, Dormale e Quisquater [DQ05], apresentaram otimizações sobre pequenas implementações seriais desse mesmo algoritmo.

Esses três últimos artigos deram importantes contribuições para o desenvolvimento do alicerce fundamental do circuito descrito neste capítulo.

3.3 O algoritmo de Stein

O cálculo do maior divisor comum baseado neste algoritmo representa a alternativa menos complexa para efetuar a operação de inversão modular. Fez-se necessário determinar qual versão desse algoritmo permitiria alcançar os requisitos de desempenho e área buscados para este projeto.

Várias otimizações foram propostas para o algoritmo de Stein [S67]: [T98], [WT00], [GC01], [WTT02], [WWSH04] e [DQ05]. Um exemplo dessas otimizações, é representado pelo código a seguir:

Algoritmo 1

(A, B, U, V) ← (P'x, p, 1, 0)

while A != 0 and B != 1

if $A_0 = 1$

if deg(A) >= deg(B)

(A, B) ← *(A + B, U + V)*

else

(A, B, U, V) ← *(A + B, A, U + V, U)*

endif

endif

(A, U) ← *(A / 2, (U / 2) mod p)*

endwhile

Para o algoritmo 1, o valor a ser invertido e o polinômio redutor são representados, respectivamente, por *P'x* e *p*. A função *deg()* retorna o grau do polinômio passado como seu parâmetro de entrada. As operações **+** e ***/2*** podem ser entendidas respectivamente como um *xor* e um deslocamento à direita.

O exemplo foi destacado aqui, porque seu código ainda possui a operação *if deg(A) >= deg(B)*. Trata-se de uma comparação entre os graus de dois polinômios. Essa operação diminui significativamente o desempenho do algoritmo devido à sua complexidade.

Na tentativa de encontrar uma solução para esse problema, Wu e outros [WWSH04] propuseram um algoritmo similar ao apresentado a seguir:

Algoritmo 2

(A, B, U, V, DCC, Flag, slice) ← *(P'x, p, 1, 0, 2, 1, 2m-1)*

while slice > 0

if $A_0 = 1$

if Flag = 1 and $DCC_0 = 0$

$(A, B, U, V, Flag) \leftarrow (A + B, A, U + V, U, 0)$

else

$(A, B) \leftarrow (A + B, U + V)$

endif

endif

$(A, U) \leftarrow (A / 2, (U / 2) \bmod p)$

if Flag = 0 and $DCC_0 = 0$

$DCC \leftarrow DCC / 2$

else

$(DCC, Flag) \leftarrow ((DCC * 2), 1)$

endif

$slice \leftarrow slice - 1$

endwhile

No algoritmo 2, duas variáveis (*Flag* e *DCC*) são usadas em conjunto, de modo a substituir a comparação entre polinômios. Isso permite ao algoritmo 2 chegar aos mesmos

resultados muito mais rapidamente, quando comparado ao algoritmo 1. Ele precisa de 2^m-1 iterações para completar uma operação de inversão.

3.4 O Circuito Combinacional

A base fundamental do circuito combinacional apresentado neste capítulo é o algoritmo 2. Quando o código desse algoritmo é representado através de um esquema elétrico, ele constitui uma fatia, a qual equivale exatamente a uma única iteração do algoritmo. Para realizar uma operação completa de inversão, o algoritmo 2 requer *2m-1* iterações sobre seu código. Dessa forma, é preciso unir serialmente *2m-1* fatias para formar um circuito combinacional de inversão modular.

Todas as fatias são iguais (vide Apêndice A). Cada fatia possui seis entradas e seis saídas, sendo elas: *Ain, Bin, Uin, Vin, DCCin, Aout, Bout, Uout, Vout* e *DCCout*, todas com *m+1* bits; *FLAGin e FLAGout*, ambas com 1bit. A saída *Aout* da primeira fatia alimenta a entrada *Ain* da segunda fatia e assim por diante. O mesmo ocorre para as demais entradas e saídas.

Na primeira fatia, as entradas são alimentadas com os seguintes valores: *(Ain, Bin, Uin, Vin, DCCin, FLAGin)* ← *(P'x, p, 1, 0, 2, 1)*.

Na última fatia, a saída *Vout* apresenta o valor referente ao inverso modular de *P'x* após algumas poucas centenas de nanossegundos.

A seguir, o esquema elétrico de uma fatia é apresentado através de figuras e comentários breves.

Cada Mega-Função do programa Quartus II, usada no projeto, está representada graficamente por meio de uma porta lógica.

Aquilo que pode ser considerado como o circuito de "controle" da fatia é mostrado na figura 3.1. Nela, aparecem portas lógicas "gerando" os sinais *AUX* e *FLAGout*, responsáveis por determinar o fluxo de dados entre os componentes da fatia.

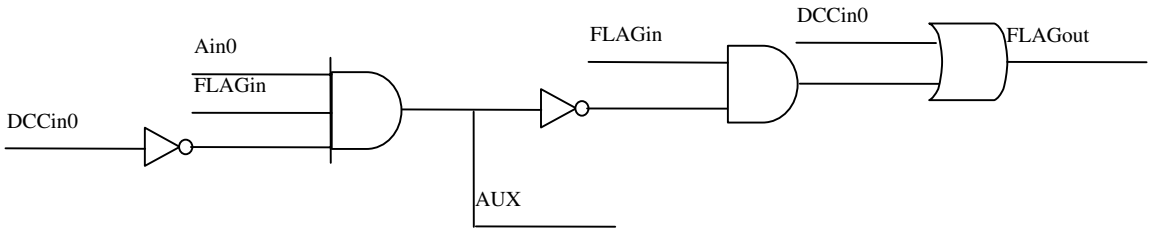

Figura 3.1: Tratamento de *FLAG* e *AUX*.

Na figura 3.2, o contador *DCC* da fatia é representado por um circuito multiplexador. Quando *FLAGout* é igual a 1, o valor de *DCC* é deslocado um bit à esquerda; quando *FLAGout* é igual a 0, o valor de *DCC* é deslocado um bit à direita (como em um contador Johnson).

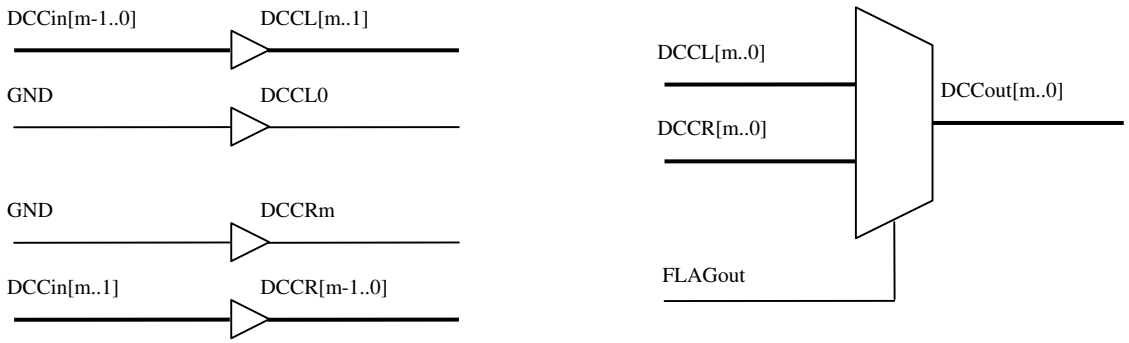

Figura 3.2: Tratamento de *DCC*.

Conforme mostrado na figura 3.3, para gerar *Aout*, é preciso realizar uma operação de *xor* entre *Ain* e *Bin*, condicionada por *Ain0*. Para esta mesma operação, *Ain* e *Bin* estão ambos deslocados de um bit para a direita.

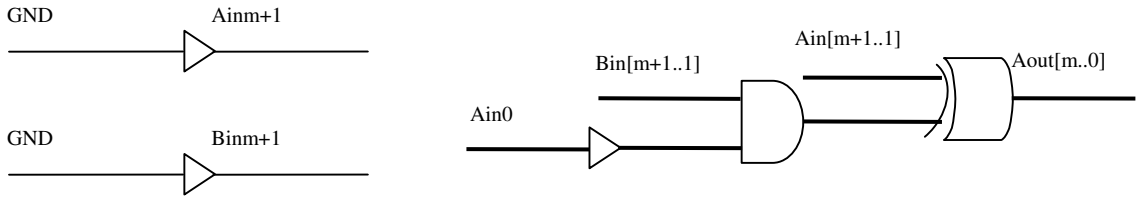

Figura 3.3: Tratamento de *A*.

A saída *Bout* apresenta o valor da entrada *Ain*, quando *AUX* assume o valor 1. Quando *AUX* for igual a 0, o valor de *Bin* é colocado na saída *Bout*, sem qualquer alteração (figura 3.4).

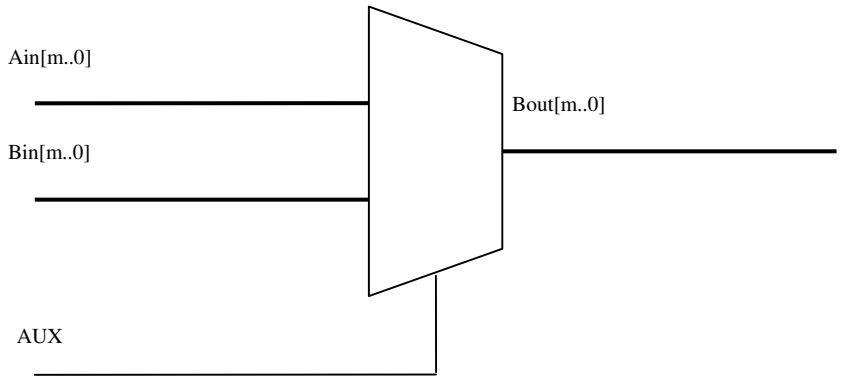

Figura 3.4: Tratamento de *B*.

Ain0 determina se haverá ou não um *xor* entre *Uin* e *Vin* para gerar o valor intermediário *U* (figura 3.5). Este por sua vez, é deslocado de um bit para a direita, o

mesmo ocorrendo com *p*. Uma vez deslocados, ambos serão submetidos a uma operação de *xor* para gerar *Uout*.

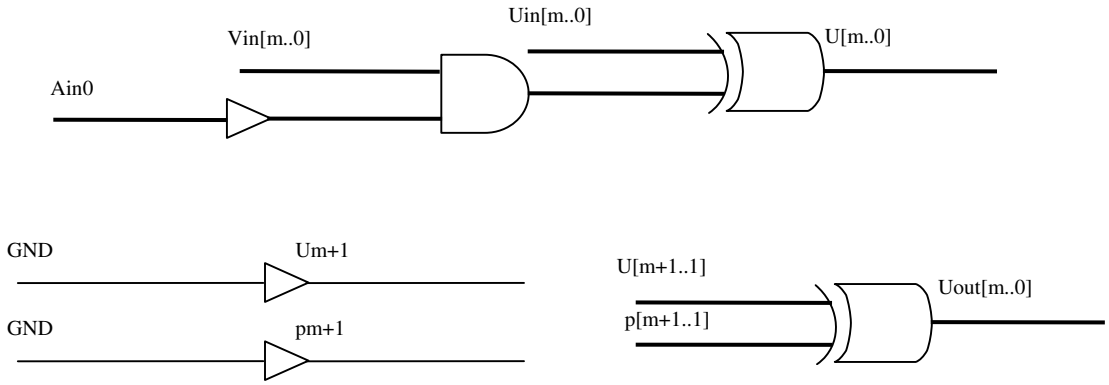

Figura 3.5: Tratamento de *U*.

Se *AUX* é igual a 0, a saída *Vout* recebe o valor da entrada *Vin*. Do contrário, o valor de *Uin* aparece na saída *Vout*, segundo apresenta a figura 3.6.

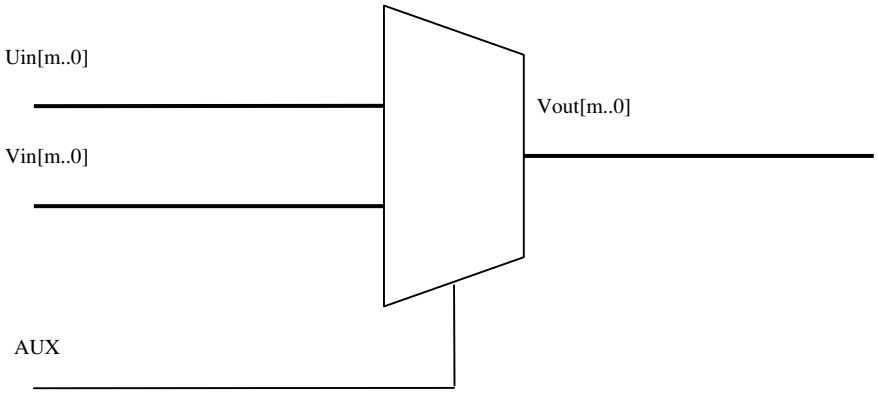

Figura 3.6: Tratamento de *V*.

Juntas, as figuras de 3.1 a 3.6 compõem o esquema elétrico de qualquer uma das *2m-1* fatias do circuito combinacional de inversão modular.

3.5 Implementações

O programa Quartus II v5.0, para *FPGAs* da Altera foi a ferramenta usada para desenvolver, compilar e simular o esquema elétrico do circuito. Este programa foi executado em um *PC* Pentium 4, trabalhando a 3,2 GHz, com 1 GB de memória RAM e 30 GB de HD.

A Stratix II EP2S1801020C4 da Altera foi a *FPGA* escolhida para a implementação deste trabalho. Ela é indicada para este projeto por sua capacidade de suportar o circuito, mesmo quando os corpos finitos são superiores a 163.

São consideras aqui implementações para quatro corpos finitos $GF(2^m)$ distintos, com m tendo as variações de 113, 131, 147 e 163. A intenção maior é revelar quanto tempo o circuito leva para realizar a operação de inversão para cada uma das quatro configurações do projeto.

Para $GF(2^{113})$, $GF(2^{131})$ e $GF(2^{163})$ os parâmetros escolhidos para definir as curvas elípticas, assim como seus pontos, estão em conformidade com os padrões *NIST, IEEE P1363, IPSec, WAP, eCheck, ANSI X9.62* e *ANSI X9.63* [SEC2-00]. O corpo finito $GF(2^{147})$ foi escolhido aleatoriamente e não faz parte de nenhum padrão. Os demais corpos finitos foram escolhidos de modo a evitar a construção de um circuito excessivamente denso, visto que são os menores corpos finitos padronizados.

De um modo geral, as implementações e testes procuram obter valores para mostrar como o circuito combinacional pode beneficiar a *ECC*.

3.6 Resultados

As implementações citadas na seção anterior permitem obter os resultados mostrados na tabela 3.1 e representados através dos gráficos 3.1 e 3.2. A tabela e os gráficos são explicados detalhadamente a partir de agora.

Através da tabela 3.1 é possível ver os resultados dos testes feitos com as implementações do circuito combinacional.

Tabela 3.1: Resultados obtidos com o circuito para a inversão modular.

Corpo Finito	Número de Pinos	Número de Fatias	Número de *LUTs*	Atraso do Circuito (ns)	Tempo de Compilação (dd:hh:mm:ss)
113	229	225	60.361	323,779	01:18:41:19
131	265	261	82.082	374,252	03:04:51:36
147	297	293	104.179	448,885	05:16:56:59
163	329	325	128.265	490,559	10:03:31:41

Os corpos finitos, usados nas diferentes implementações, aparecem em cada uma das linhas da primeira coluna da tabela 3.1. As linhas das demais colunas referem-se a cada um desses corpos finitos.

A segunda coluna mostra o número de pinos usados na *FPGA*. A *FPGA* EP2S1801020C4 possui um total de 1020 pinos, porém destes somente 743 podem ser utilizados pela aplicação, os demais são reservados para alimentação, relógio e carga de configuração. Isso significa que, para os corpos finitos 113, 131, 147 e 163, são usados respectivamente 31%, 36%, 40% e 44% do total de pinos disponíveis para uso pelo circuito.

A terceira coluna apresenta o número total de fatias ligadas em série dentro circuito combinacional.

A área do circuito é representada na coluna 4, através do número de *LUTs*. A *FPGA* EP2S1801020C4 possui 179.400 *LUTs*. Deste total, 143.520 estão disponíveis para implementar circuitos e os demais são reservados para operação do próprio dispositivo. Dessa forma, as implementações do circuito para os corpos finitos 113, 131, 147 e 163 usam respectivamente 42%, 57%, 73% e 89% do número total de *LUTs* disponibilizadas para uso.

O atraso do circuito, revelado pela coluna 5, informa quantos nanossegundos o circuito leva para realizar uma operação de inversão modular.

Finalmente, a coluna 6 apresenta quantos dias, horas, minutos e segundos o circuito levou para ser compilado em um *PC* Pentium 4, trabalhando a 3,2 GHz, com 1 GB de memória RAM e 30 GB de HD.

O gráfico 3.1 estabelece relações entre as implementações do circuito combinacional para os diferentes corpos finitos e as áreas ocupadas por elas, dadas em número de *LUTs*.

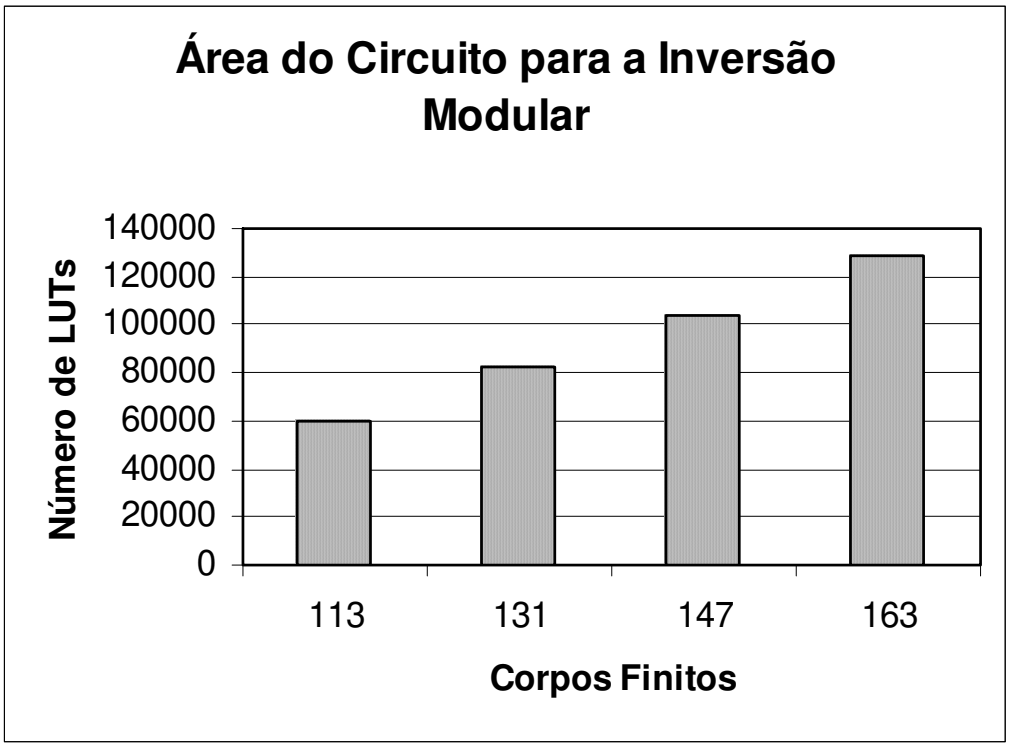

Gráfico 3.1: Área do circuito para a inversão modular.

Os corpos finitos binários são apresentados pelo eixo das ordenadas.

O eixo das abscissas mostra valores referentes à quantidade de *LUTs* usada na *FPGA* para cada implementação.

É possível fazer a seguinte constatação através da observação do gráfico 3.1: aumentos de aproximadamente 16 unidades nos corpos finitos causam adições de aproximadamente 22.000 *LUTs* na área de cada nova implementação do circuito combinacional.

A relação entre o corpo finito binário na implementação do circuito combinacional e o tempo (em nanossegundos) que este último leva para realizar uma operação de inversão é apresentada pelo gráfico 3.2.

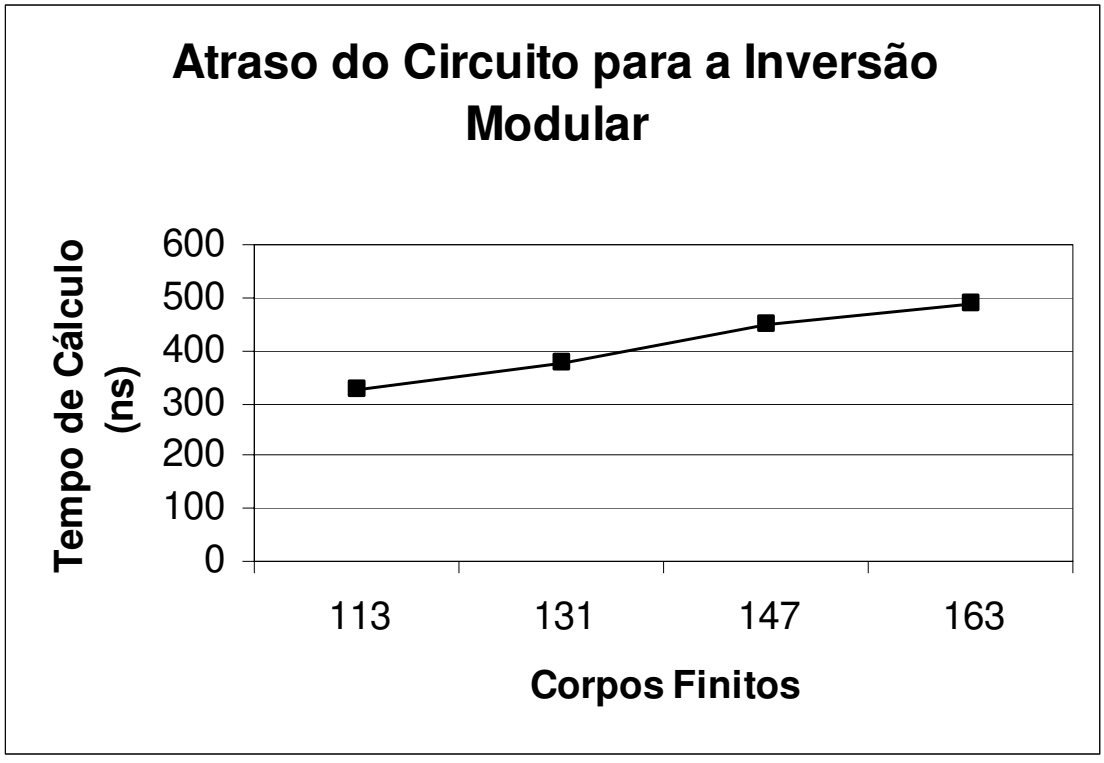

Gráfico 3.2: Atraso do circuito para a inversão modular.

O eixo das ordenadas compreende os valores referentes aos diferentes corpos finitos.

Os valores correspondentes ao tempo gasto pelo circuito ao realizar seu trabalho são mostrados através do eixo das abscissas.

A observação do gráfico 3.2 permite a seguinte constatação: variações do atraso do circuito entre, aproximadamente, 8% e 18% de uma implementação para outra permitem uma projeção quase linear do tempo necessário para realizar uma operação de inversão em função do corpo finito.

Os resultados apresentados neste capítulo podem auxiliar no cálculo do desempenho de sistemas criptográficos que venham fazer uso deste circuito combinacional.

3.7 Comentários Finais

Este capítulo apresentou um circuito combinacional de inversão modular para curvas elípticas sobre corpos finitos binários $GF(2^m)$, baseado no algoritmo de Stein.

O circuito é indicado para operar em sistemas criptográficos de alto desempenho, sem restrições de área, que trabalham com coordenadas afins.

O desempenho do circuito pôde ser constatado graças às implementações feitas na *FPGA* Stratix II EP2S180F1020C4 da Altera, para os corpos finitos 113, 131, 147 e 163.

Para os mesmos corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$, trabalhando diretamente com coordenadas afins e base polinomial, cada operação de inversão pode ser realizada, respectivamente, nos tempos 323,779 ns, 374,252 ns, 448,885 ns e 490,559 ns.

Dentro do sistema criptográfico proposto através deste trabalho, o circuito combinacional para a inversão modular é parte integrante de um outro circuito combinacional ainda maior, responsável pelas operações de duplicação de um ponto e adição de dois pontos distintos.

Capítulo 4
Um Circuito Combinacional para a Duplicação de um Ponto e para a Adição de Dois Pontos Distintos

4.1 Introdução

O circuito combinacional apresentado neste capítulo realiza tanto a duplicação como a adição de pontos de uma curva elíptica para corpos finitos binários $GF(2^m)$ [DO2-06]. Ele é indicado para sistemas de alto desempenho, os quais operam diretamente com coordenadas afins. Foi dada prioridade à velocidade em detrimento à área.

Ele serve para operar em conjunto com algum algoritmo baseado em $ECC(GF(2^m))$, também preferencialmente implementado em *hardware*. Conforme a necessidade do algoritmo, o circuito realiza ou a duplicação de um ponto ou a adição de dois pontos distintos, através do uso de operações mais elementares como a inversão modular, a multiplicação, a adição modular, o quadrado e o módulo, seguindo as equações de (2.14) a (2.16) .

Cada duplicação ou adição gera um ponto Q. Para o circuito mostrar as coordenadas de Q nas suas saídas, basta alimentar as suas entradas com as coordenadas P'_X e P'_Y.

Resumindo, o circuito implementa as equações de (2.14) a (2.16), de tal modo a obter as coordenadas parciais (ou finais) do ponto Q, na casa dos nanossegundos.

A continuação deste capítulo é dada em mais cinco seções comentadas a seguir: a seção 4.2 comenta alguns trabalhos prévios na área; a seção 4.3 revela como cada operação elementar tornou-se uma parte do circuito; a seção 4.4 comenta as implementações do circuito nas *FPGAs* da Altera; a seção 4.5 apresenta os resultados obtidos a partir dessas implementações. O capítulo termina mostrando os comentários finais na seção 4.6.

4.2 Trabalhos Prévios

Esta seção faz comentários sucintos a respeito de dois trabalhos que possuem aspectos relacionados com o circuito apresentado neste capítulo e faz comparações entre eles.

O primeiro deles é um artigo publicado por Orlando e Paar [OP00] em 2000, no *Workshop CHES*. O segundo é um relatório técnico publicado por Shantz e outros [EGSG03] em 2003, pelos Laboratórios da Sun Microsystems.

Os dois trabalhos descrevem processadores criptográficos para *ECC-GF(*2^m*)*. Eles calculam $Q = kP$ por meio de duplicações e adições de pontos representados através de coordenadas projetivas. O algoritmo empregado é o Montgomery.

Já o circuito proposto neste capítulo duplica e adiciona pontos representados diretamente através de coordenadas afins. Além disso, ele pode trabalhar em conjunto com uma grande variedade de algoritmos de criptografia.

Para os dois trabalhos citados nesta seção, a vantagem de trabalhar com pontos representados através de coordenadas projetivas é que, em função disso, o algoritmo Montgomery só precisa realizar uma operação de inversão modular durante o cálculo $Q = kP$. Em contraste, para trabalhar diretamente com coordenadas afins, o circuito apresentado neste capítulo realiza uma inversão modular para cada operação de duplicação ou de adição de pontos, quando ele é usado por um algoritmo para calcular $Q = kP$. Vale lembrar que a inversão modular é uma das operações mais complexas e demoradas da aritmética sobre corpos finitos.

Os dois trabalhos comentados aqui, usam circuitos seqüenciais para realizar a inversão modular, o que contribui significativamente para a redução da área dos mesmos. A vantagem desses trabalhos é atender a uma crescente demanda por circuitos que caibam em ambientes restritos, como por exemplo, sistemas embarcados. Porém, para ocuparem uma área menor, tais circuitos penalizam seu próprio desempenho.

Em nosso trabalho a inversão modular está implementada através de um circuito combinacional muito rápido (vide Capítulo 3). Isso contribui significativamente para que este circuito apresente um desempenho superior em relação aos dois trabalhos citados nesta seção, mesmo realizando uma inversão modular para cada operação de duplicação ou de adição de pontos.

A tendência atual das pesquisas sobre criptografia implementada em *hardware* é propor sistemas, que trabalham com pontos de curvas elípticas representados através de coordenadas projetivas e base normal, implementados através de circuitos seqüenciais. Embora este trabalho não siga essa tendência, as características deste circuito podem atender às aplicações que necessitem empregar um sistema de alto desempenho sem impor restrições de área.

Neste ponto, é válido lembrar que, desde o início, este trabalho teve como motivação o desafio de propor a criação de um sistema criptográfico de alto desempenho que trabalhasse com coordenadas afins.

Alguns outros aspectos relacionados com este circuito podem ser encontrados em [EKHH01], [TOIT00], [S01] e [GSS99].

4.3 O Circuito Combinacional

Este circuito trabalha, internamente, com as constantes P_X, P_Y, a e p, já anteriormente definidas, as quais servem respectivamente para informar o ponto fixo P e os parâmetros a e p de uma curva elíptica. Resumindo, P é um ponto pertencente a uma curva elíptica definida por a sobre um corpo finito de tamanho p (vide Capítulo 2).

Todos esses parâmetros e pontos são definidos pelos padrões *NIST, IEEE P1363, IPSec, WAP, eCheck, ANSI X9.62* e *ANSI X9.63* [SEC2-00] para uma curva elíptica sobre $GF(2^{113})$, $GF(2^{131})$ e $GF(2^{163})$. O corpo finito $GF(2^{147})$ foi escolhido aleatoriamente e não faz parte de nenhum padrão.

Além dessas constantes internas, o circuito recebe, através de suas entradas, as coordenadas de um ponto *P'*. Todas essas informações juntas são usadas para calcular as coordenadas de um novo ponto Q, as quais aparecem nas saídas do circuito.

O circuito possui ao todo três entradas e duas saídas, como pode ser observado por meio do diagrama básico da figura 4.1.

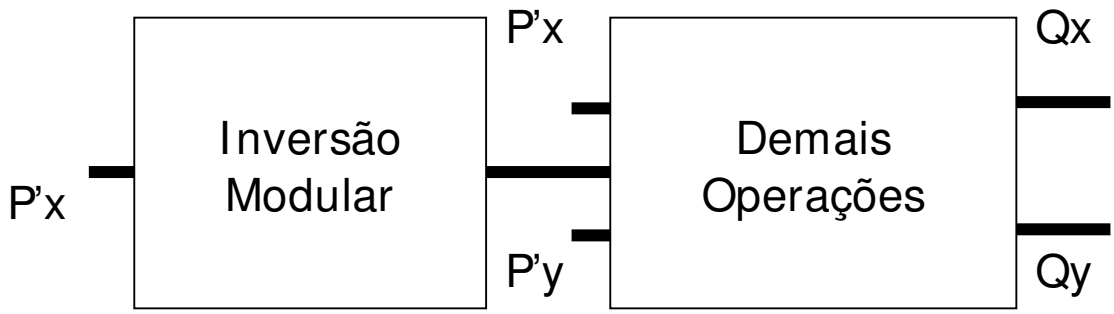

Figura 4.1: Diagrama básico do circuito para a duplicação e a adição de pontos.

As entradas P'_X e P'_Y recebem as coordenadas, respectivamente, x e y do ponto P'. No caso da duplicação de um ponto, as coordenadas x e y do ponto P' são repetidas, internamente, também para P_X e P_Y. As saídas Q_X e Q_Y informam as coordenadas x e y do ponto Q gerado pelo circuito.

Em outras palavras, toda a tarefa do circuito aqui apresentado é executar as equações de (2.14) a (2.16) (vide Capítulo 2). Isso quer dizer que internamente o circuito executa aquelas operações mais básicas, comentadas na seção 4.1: a inversão modular, a multiplicação, o módulo, o quadrado e a adição modular, respeitando a regra matemática de precedência dentro das equações de (2.14) a (2.16).

A *FPGA* Stratix II EP2S180F1020C4 executa exclusivamente a inversão modular, como descrito no Capítulo 3, enquanto a *FPGA* Stratix II EP2S90F1508C3 executa as demais operações. Com exceção da inversão modular, cada uma dessas operações básicas está descrita em *VHDL*, conforme é mostrado a seguir para $GF(2^{163})$.

A adição é descrita por uma simples operação de *xor*. Por exemplo, o fragmento da equação (2.15) $P_X + P'_X$ está descrito como P_X *XOR* P_{Xin}. Esta operação aparece descrita várias vezes dentro do código. A figura 4.2 mostra um trecho do código do circuito com algumas dessas adições.

```
Qx := temp XOR S XOR Px XOR Pxin XOR a;

temp := Px XOR Qx;
```

Figura 4.2: Adições.

Representada por S^2 na equação (2.15), a operação do quadrado é descrita através de um código bastante simples. Basta colocar um bit zero entre cada um dos bits de *S*, para descrevê-la em *VHDL*. Ela aparece uma única vez em todo o código, pois faz parte apenas do fragmento $(S^2 + S + P_X + P'_X + a) \bmod p$. O código do quadrado é representado na figura 4.3.

```
FOR i IN 0 TO 163 LOOP

  dbltemp(2*i) := S(i);

END LOOP;
```

Figura 4.3: Quadrado.

A multiplicação usa um algoritmo bastante tradicional. Ela é descrita em *VHDL* através de um laço, efetuando repetidos deslocamentos à esquerda, seguidos de operações de *xor*, como é mostrado na figura 4.4. Essa operação aparece duas vezes no código, nos fragmentos: $((P_Y + P'_Y) * (P_X + P'_X)^{-1})$ e $S(P_X + P'_X)$.

```
FOR i IN 0 TO 163 LOOP

  IF temp( i ) = '1' THEN

    dbltemp((163+i) DOWNTO i) :=
    S XOR dbltemp((163+i) DOWNTO i);

  END IF;

END LOOP;
```

Figura 4.4: Multiplicação.

Fazendo uso de um algoritmo muito conhecido, o módulo é responsável por extrair o resto de uma operação de divisão. Para implementar o módulo, uma descrição em *VHDL* é feita para um laço, que realiza repetidos deslocamentos à direita, acompanhados de operações de *xor*, conforme pode ser visto pela figura 4.5.

Apesar de ser representado como *mod p* para cada uma das equações de (2.14) a (2.16), ele aparece apenas três vezes no código, uma após o quadrado e uma para cada multiplicação.

```
FOR i IN 0 TO 163 LOOP

 IF dbltemp( (326-i) ) = '1' THEN

  dbltemp((326-i) downto (163-i)) :=
  p xor dbltemp((326-i) downto (163-i));

 END IF;

END LOOP;
```

Figura 4.5: Módulo.

O circuito também implementa funções para encontrar o ponto *Q*, quando uma ou as duas coordenadas de pontos têm valor nulo.

Por exemplo, se $P_X = 0$ e $P_Y = 0$, então $Q = P'$. Do mesmo modo, se $P'_X = 0$ e $P'_Y = 0$, então $Q = P$. Para $P = P' = 0$, obtêm-se $Q = 0$, isto é, um ponto no infinito.

Q também será um ponto no infinito em duas outras condições: se $P_X = 0$, no caso da duplicação de um ponto; se $P_Y = P'_Y$, no caso da adição de dois pontos distintos.

O Apêndice B mostra esse código completo descrito em *VHDL*.

Diferentemente das demais operações mostradas aqui, a inversão modular é bastante complexa. Ela foi efetuada por um código baseado no algoritmo de Stein [WWSH04], o qual encontra-se representado em esquema elétrico (vide Capítulo 3).

A operação de inversão modular, representada por x^{-1}, aparece uma única vez nas equações de (2.14) a (2.16).

4.4 Implementações

O circuito foi desenvolvido, compilado e simulado através da ferramenta Quartus II v5.0, para *FPGAs* da Altera. Duas *FPGAs* diferentes, trabalhando em conjunto, implementam o circuito combinacional completo. A primeira delas é a EP2S180F1020C4, que implementa o algoritmo de inversão modular. A outra é a EP2S90F1508C3, que implementa as demais operações presentes nas equações de (2.14) a (2.16). Essas *FPGAs* são apropriadas para o projeto por serem capazes de suportar o circuito para corpos finitos até mesmo superiores a 163. Formas de particionamento são apresentadas em [GVNG94].

Objetiva-se com isso mostrar a porção de tempo consumida pelo circuito ao realizar as operações de duplicação ou adição de pontos, considerando cada um dos quatro valores de *m* usados.

4.5 Resultados

Os valores mostrados na tabela 4.1 e representados através dos gráficos 4.1 e 4.2 são resultados obtidos a partir das implementações citadas na seção anterior. Os mesmos serão explicados detalhadamente adiante nesta seção.

A tabela 4.1 mostra os valores obtidos a partir dos testes feitos com implementações do circuito combinacional para $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$, nas *FPGAs* EP2S1801020C4 e EP2S90F1508C3 da Altera.

As linhas da primeira coluna apresentam os diferentes corpos finitos usados para as implementações. Cada uma das linhas das colunas seguintes faz referência a um corpo finito da primeira coluna.

Tabela 4.1: Resultados obtidos com o circuito para a duplicação e a adição de pontos.

Corpo Finito	Número de *LUTs*	Atraso do Circuito (ns)	Tempo de Compilação (dd:hh:mm:ss)
113	74.990	412,873	01:22:22:52
131	101.889	478,501	03:11:35:31
147	128.797	565,883	05:25:31:21
163	158.796	613,801	10:15:50:54

Através do número de *LUTs*, a coluna 2 representa a área ocupada pelo circuito. Juntas, as *FPGAs* EP2S1801020C4 e EP2S90F1508C3 disponibilizam 216.288 *LUTs*. Assim sendo, para as implementações do circuito, considerando os corpos finitos de 113, 131, 147 e 163, o número de *LUTs* em uso representa respectivamente 35%, 47%, 60% e 73% do total de *LUTs* disponíveis para uso nas duas *FPGAs*.

A coluna 3 informa quantos nanossegundos o circuito gasta ao efetuar uma operação de duplicação de um ponto ou de adição de dois pontos, isto é, o atraso do circuito durante cada operação.

Finalmente, a coluna 4 mostra o tempo necessário para o circuito ser compilado em um *PC* Pentium 4, trabalhando a 3,2 GHz, com 1 GB de memória RAM e 30 GB de HD, dado em número de dias, horas, minutos e segundos.

O gráfico 4.1 relaciona os diferentes corpos finitos e as áreas ocupadas (em número de *LUTs*) para as distintas implementações do circuito combinacional.

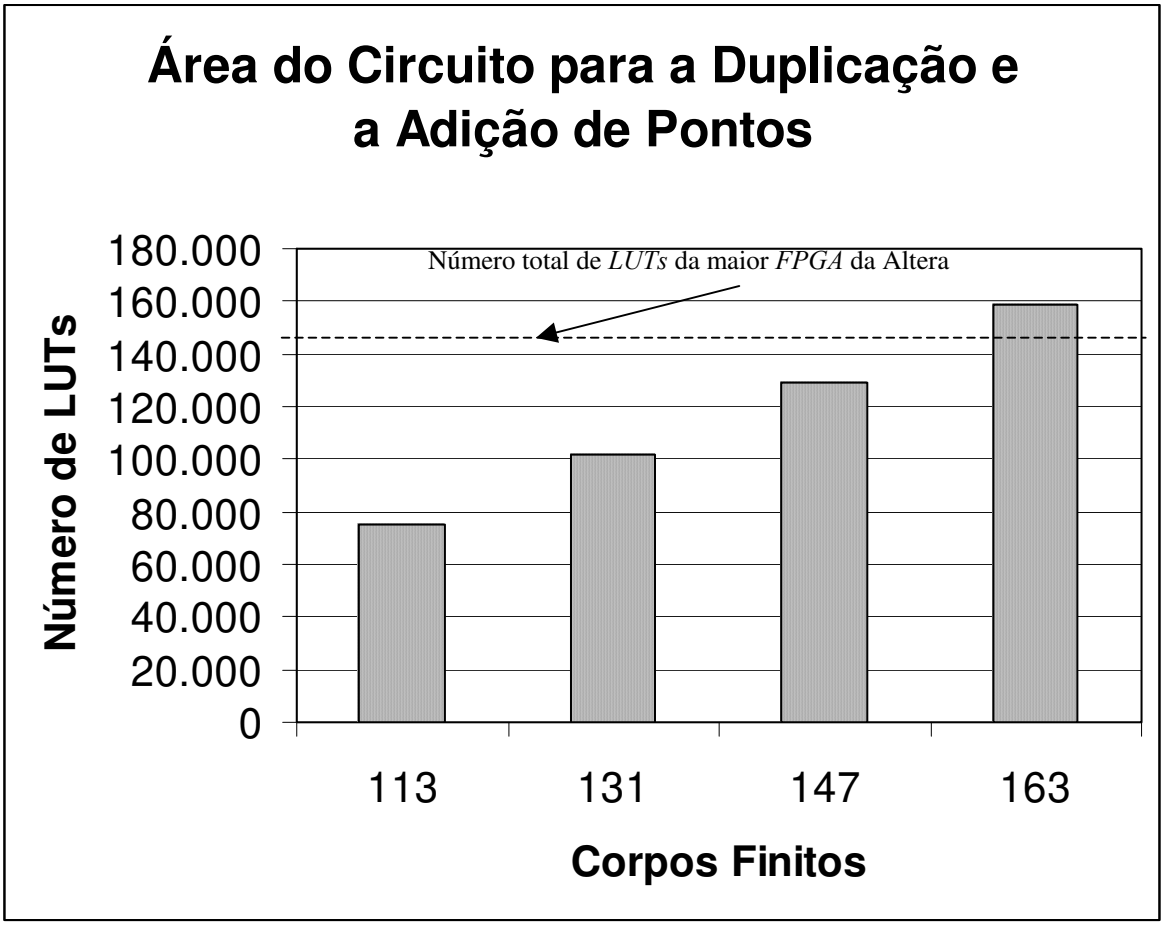

Gráfico 4.1: Área do circuito para a duplicação e a adição de pontos.

O eixo das ordenadas apresenta os corpos finitos binários. Os valores referentes à quantidade de *LUTs* usada na *FPGA* para cada implementação são mostrados através do eixo das abscissas.

Ao observar o gráfico 4.1 é possível fazer a constatação a seguir: para o corpo finito 163, o número total de *LUTs* usado pelo circuito supera o total de *LUTs* da maior *FPGA* da

Altera no mercado. Por isso o circuito foi implementado em duas *FPGAs*. Entretanto, para os demais corpos finitos, o circuito poderia ser implementado em uma única *FPGA*.

O gráfico 4.2 apresenta a relação entre o corpo finito binário na implementação do circuito combinacional e o tempo (em nanossegundos) que este último leva para realizar uma operação de duplicação de um ponto ou de adição de dois pontos.

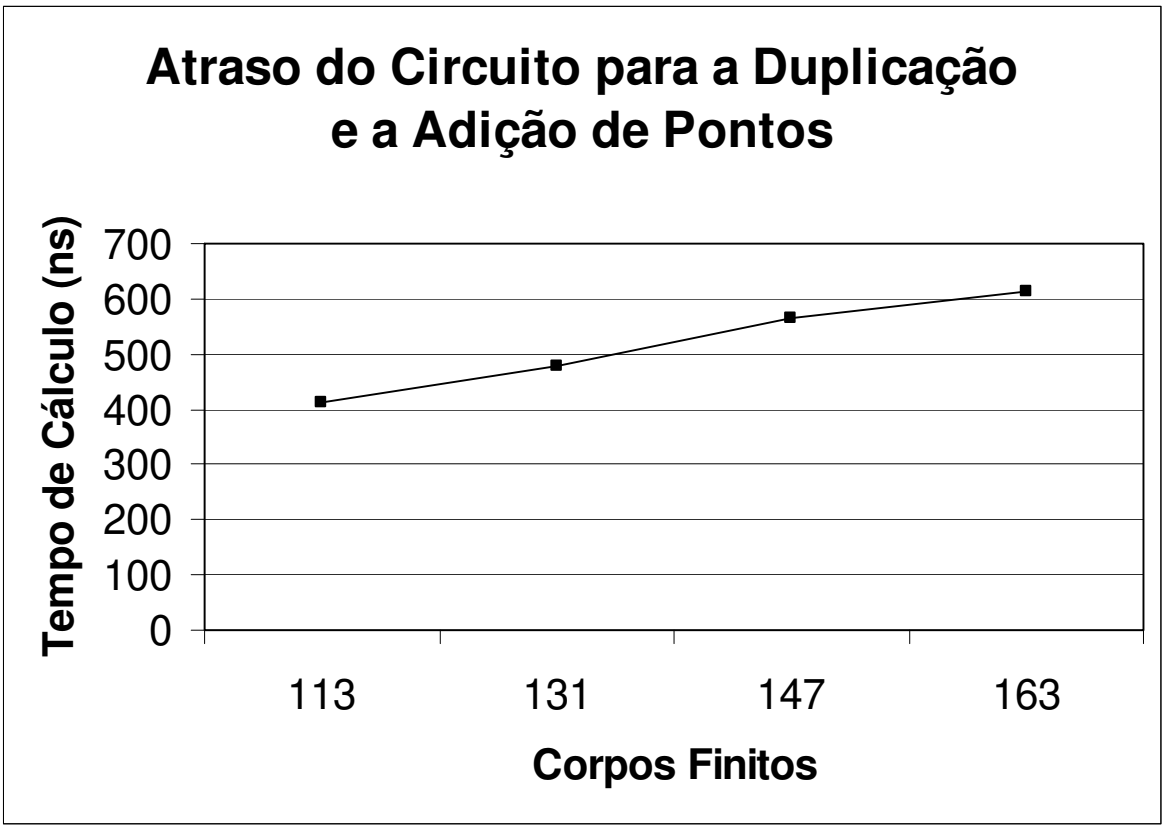

Gráfico 4.2: Atraso do circuito para a duplicação e a adição de pontos.

Os valores referentes aos diferentes corpos finitos são apresentados pelo eixo das ordenadas.

O eixo das abscissas compreende os valores referentes ao tempo gasto pelo circuito ao realizar seu trabalho.

Pela observação do gráfico 4.2, constata-se o seguinte: de uma implementação para outra, há uma variação sutil no atraso do circuito, fazendo com que o gráfico do tempo seja quase linear.

Apesar da área do circuito aumentar bastante em função do corpo finito, o atraso do circuito não sofre grandes variações entre as implementações.

Sistemas criptográficos que façam uso deste circuito combinacional podem ter seu desempenho calculado com o auxílio dos resultados apresentados neste capítulo.

4.6 Comentários Finais

Apresentamos um circuito combinacional capaz de efetuar a duplicação ou a adição de pontos de uma curva elíptica para corpos finitos binários.

Ele foi descrito parte em *VHDL*, parte em esquema elétrico e implementado em duas *FPGAs* da Altera.

O circuito pode ser usado para ajudar na execução de uma variedade de algoritmos de *ECC*, sendo indicado para sistemas de alto desempenho, operando diretamente com coordenadas afins e base polinomial.

As operações de duplicação e adição de pontos podem ser realizadas nos tempos 412,873 ns, 478,501 ns, 565,883 ns e 613,801 ns, respectivamente, para os corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{131})$ e $GF(2^{163})$.

Capítulo 5
Um Sistema Criptográfico Implementado em Circuitos Programáveis

5.1 Introdução

Este capítulo apresenta um sistema criptográfico para curvas elípticas sobre corpos finitos binários - $GF(2^m)$, implementado em circuitos programáveis.

Ele é indicado para trabalhar em conjunto com programas que necessitam utilizar a criptografia assimétrica. Com a ajuda desse *hardware*, tais programas aumentam seu desempenho.

O sistema criptográfico é apresentado na forma de um exemplo de implementação em uma placa adaptadora para *PC*. Isso é feito apenas com o objetivo de facilitar a compreensão das operações do sistema criptográfico, além de inseri-lo em um contexto prático.

Para a demonstração do funcionamento da placa que implementa o sistema são utilizados os corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$.

Os resultados, apresentados na seção 5.5, mostram que o sistema criptográfico é capaz de efetuar a operação $Q = kP$ em algumas dezenas de microssegundos.

Seu funcionamento é descrito resumidamente a seguir.

Inicialmente, um programa que, por exemplo, implementa o modelo Diffie-Hellman através do algoritmo descrito no Capítulo 1, é processado em um *PC*. Durante seu processamento, esse modelo necessita obter um ponto Q, conseguido por meio do cálculo $Q = kP$.

Para alcançar um melhor desempenho, o programa transfere esse cálculo para a placa que implementa o sistema criptográfico.

Quando a placa está pronta para trabalhar, ela avisa o programa por meio de uma interrupção. O programa, então, transfere dados para ela através do barramento de dados do *PC*. Em outras palavras, o programa endereça um registrador na placa e escreve um ponto *P'* nele (vide seção 2.3).

Paralelamente, dentro da placa, uma chave privada *k* é gerada através do gerador de números aleatórios (*RNG*). Uma vez possuindo a chave privada *k* e o ponto *P'*, a placa pode realizar a sua tarefa, isto é, calcular $Q = kP$, sob o comando de algum algoritmo de criptografia implementado na mesma placa.

Finalizando o processo, a placa causa outra interrupção para avisar o programa sobre o encerramento da sua tarefa e envia dados de volta para ele por meio do mesmo barramento de dados. Dito de outra forma, após sofrer a interrupção, o programa endereça um registrador na placa e lê o ponto *Q* (calculado) a partir dele. Desse modo, o programa comentado anteriormente recebe o ponto *Q* conforme esperado.

Este capítulo segue da maneira citada a seguir: a seção 5.2 comenta um trabalho prévio relacionado com este sistema; a seção 5.3 revela detalhes sobre o funcionamento do sistema criptográfico na placa; a seção 5.4 apresenta algumas características das implementações do sistema criptográfico feitas para quatro corpos finitos distintos; a seção 5.5 mostra os resultados alcançados com essas implementações. O capítulo é encerrado com os comentários finais na seção 5.6.

5.2 Trabalhos Prévios

Na seção anterior comenta-se que um programa pode transferir a responsabilidade de efetuar o cálculo $Q = kP$ para a placa que implemente um sistema criptográfico, a fim de alcançar um melhor desempenho. Para se ter uma idéia desse ganho em desempenho, é preciso comparar a execução do programa sem ou com o auxílio da placa.

O programa escolhido para ser usado nessa comparação faz parte de um trabalho apresentado pelos membros do grupo REGRA_C, em 2001 [MAH01].

Nele é possível encontrar a descrição da implementação em *software* de um sistema criptográfico baseado em curvas elípticas similar ao sistema proposto neste trabalho.

Para a implementação em questão, a codificação foi feita inteiramente em linguagem C. O código está dividido em camadas, como representado através da figura 5.1 e explicado a seguir.

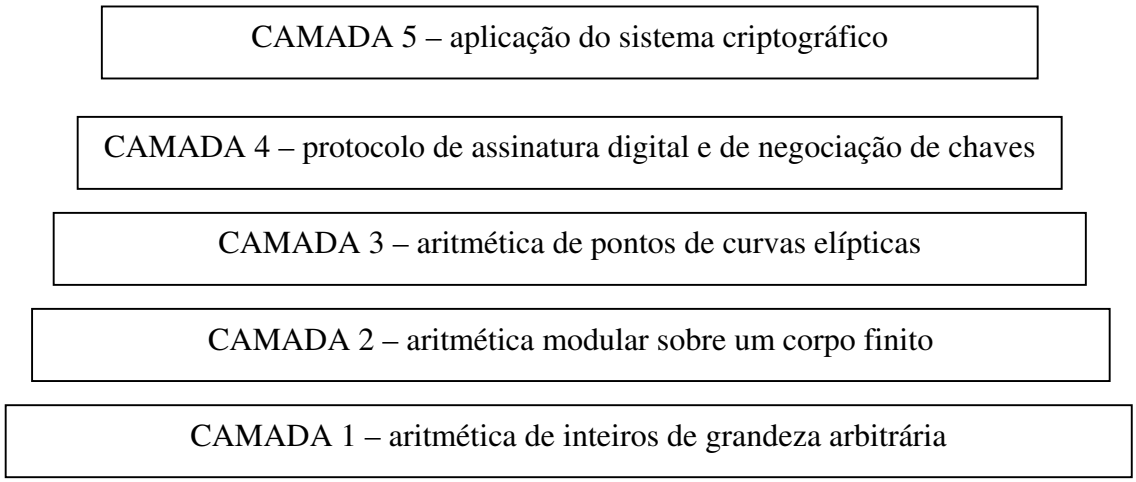

Figura 5.1: Implementação de um *software* de criptografia em camadas.

A ordem de grandeza dos números empregados em *ECC* impossibilita que eles sejam manipulados pelos *PCs* atuais através de variáveis inteiras comuns. Por esse motivo, a camada 1 oferece rotinas de propósito geral, para uso de números inteiros de tamanho arbitrário, específicas para implementações em *software*.

Em outras palavras, a camada 1 permite ao sistema usar números da ordem de 114 bits ou mais, mesmo que a arquitetura suporte apenas variáveis inteiras de 32 bits.

Na camada 2 estão presentes as funções que implementam as operações elementares pertencentes à aritmética sobre corpos finitos, isto é, a adição modular, o quadrado, o módulo, a multiplicação e a inversão modular.

Já a camada 3 é composta por funções que executam a duplicação de um ponto e a adição de dois pontos distintos para resolver a equação $Q = kP$.

A camada 4 usa as operações matemáticas das camadas inferiores para implementar, dentre outros serviços, o modelo de troca de chaves Diffie-Hellman.

A quinta camada representa a aplicação usuária do sistema criptográfico.

Voltando à questão do desempenho, por exemplo, para $GF(2^{163})$, o programa leva 5 segundos para ser executado em um *PC* Pentium 4, trabalhando a 3,2 GHz, com 1 GB de memória RAM e 30 GB de HD.

O mesmo programa, sendo executado na mesma máquina, gasta 244 microssegundos para efetuar sua tarefa, quando auxiliado pela placa. Isso quer dizer que a execução do programa fica 20.483 vezes mais rápida com a ajuda do sistema criptográfico proposto através deste trabalho. Esses valores serão discutidos mais detalhadamente na seção 6.5.

Neste exemplo, para fazer a comparação de desempenho, ambos os sistemas usam o *Double-and-Add* (descrito em [MAH01]) como algoritmo criptográfico.

Além dessa comparação de desempenho entre as diferentes implementações, o programa do REGRA_C é usado também para a validação deste trabalho, uma vez que as camadas 2 e 3 implementadas no trabalho do REGRA_C realizam as mesmas tarefas do sistema criptográfico apresentado nesta obra.

Vale destacar que o grupo REGRA_C implementa as camadas 2 e 3 inteiramente em *software* e utiliza um algoritmo de inversão modular diferente do algoritmo de Stein.

5.3 A Placa Adaptadora para PC

A figura 5.2 apresenta o diagrama básico simplificado da placa adaptadora para *PC*, usada para exemplificar uma implementação do sistema criptográfico. Ela mostra os elementos que compõem a placa e também os elementos do computador com os quais a placa se comunica. A fim de sermos didáticos, escolhemos mostrar este exemplo simples de

implementação do sistema criptográfico, porém outras formas alternativas de implementação também são possíveis.

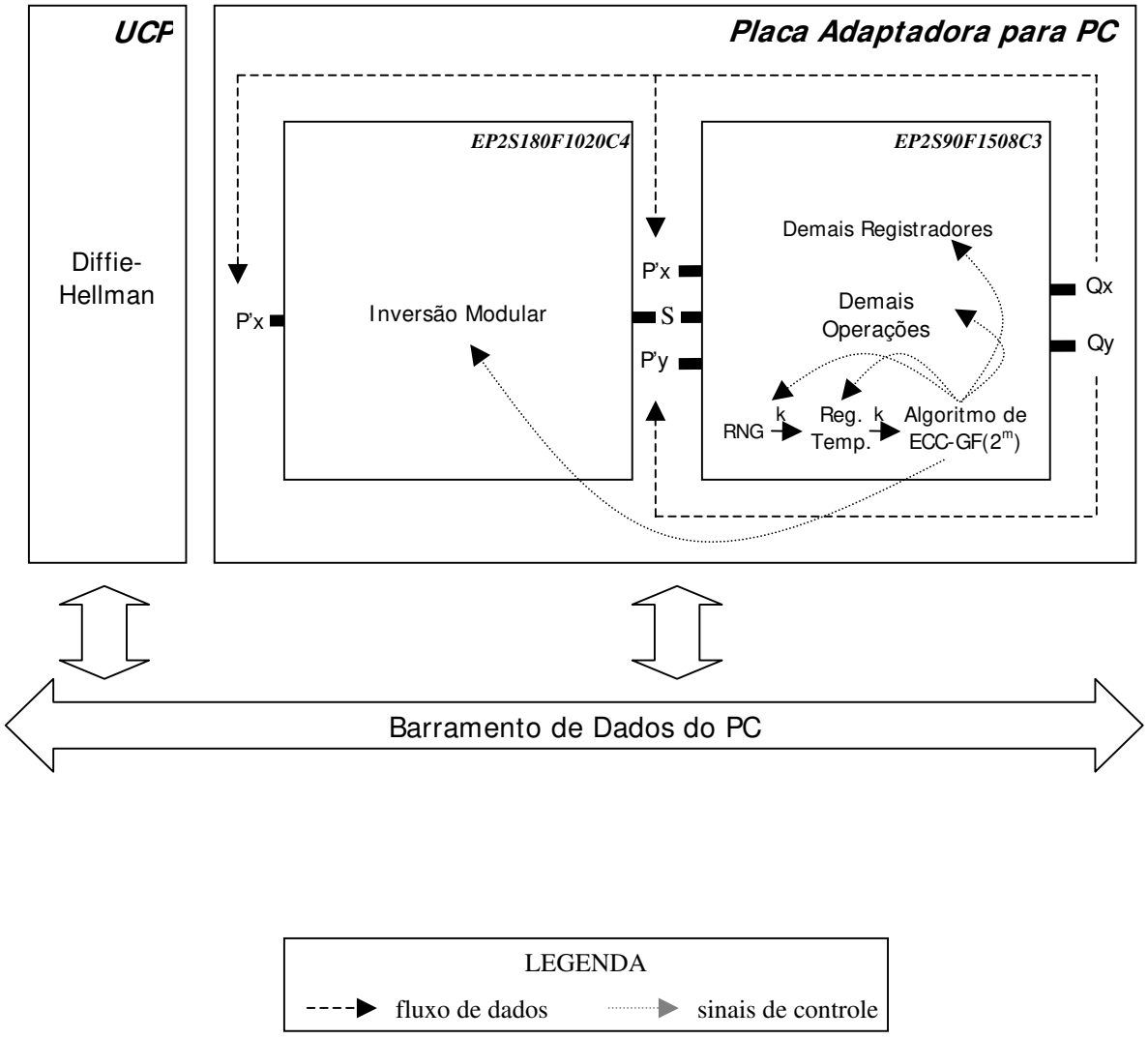

Figura 5.2: Diagrama básico simplificado da placa adaptadora para PC.

A placa é composta basicamente por duas *FPGAs* Stratix II da Altera: a EP2S180F1020C4 e a EP2S90F1508C3. A primeira delas implementa o circuito combinacional usado para realizar a operação de inversão modular presente na equação (2.14). A segunda implementa o circuito combinacional necessário para o cálculo das demais operações presentes nas equações de (2.14) a (2.16). Além disso, a segunda *FPGA* também implementa o algoritmo criptográfico, o gerador de números aleatórios (RNG), os registradores e a lógica responsável pela interface entre a placa e o barramento de dados do *PC*.

Este sistema não trabalha com um algoritmo específico. Um algoritmo qualquer de *ECC-GF(*2^m*)* pode ser descrito em *VHDL* e implementado na *FPGA* destinada a esse fim. Porém, para o correto funcionamento da placa, esse algoritmo deve gerar alguns sinais de controle, de modo a garantir um determinado fluxo de dados no interior da placa. Esse fluxo de dados será explicado mais à frente, ainda nesta seção. Ele também deve considerar os pontos da curva como sendo representados através de coordenadas afins e base polinomial.

A placa também não opera com um gerador de números aleatórios exclusivo. Ao contrário, ela permite a implementação de muitos tipos de *RNG*. Entretanto, como sugestão, é possível trabalhar com algum dos geradores descritos em *VHDL* apresentados por Abcunas et al. [ACPR04] e Martin [M02].

A implementação dos registradores também deve ser flexível. Isso possibilita estabelecer a dimensão dos registradores conforme o corpo finito com o qual se deseja trabalhar.

Além disso, os circuitos lógicos responsáveis pela entrada e saída de dados da placa, devem ser descritos e implementados de acordo com o tipo de barramento do *PC* (*ISA*, *PCI*, *AGP* ou outro).

A tabela 5.1 mostra as características mais comuns de alguns barramentos.

Tabela 5.1: Características mais comuns de alguns barramentos.

Tipo do Barramento*	Largura do Barramento (Bytes)*	Taxa de Transferência (MB/s)*	Tempo de Acesso à Placa (μs) Por Coordenada de Ponto			
			GF(2^{113})	GF(2^{131})	GF(2^{147})	GF(2^{163})
ISA	2	4	4,000	4,500	5,000	5,500
PCI - 33	4	133	0,120	0,150	0,150	0,180
PCI - 66	4	266	0,060	0,075	0,075	0,090
PCI - 33	8	266	0,060	0,090	0,090	0,90
PCI - 66	8	533	0,030	0,045	0,045	0,045
PCI-X 66	8	533	0,030	0,045	0,045	0,045
PCI-X 133	8	1.066	0,015	0,022	0,022	0,022
PCI-X 266	8	2.133	0,007	0,011	0,011	0,011
PCI-X 533	8	4.266	0,003	0,005	0,005	0,005
AGP x1	4	266	0,060	0,075	0,075	0,090
AGP x2	4	533	0,030	0,037	0,037	0,045
AGP x4	4	1.066	0,015	0,018	0,018	0,022
AGP x8	4	2.133	0,007	0,009	0,009	0,011

* www.clubledohardware.com.br e www.infowester.com

A primeira coluna da tabela 5.1 mostra os diferentes tipos de barramentos mais comumente encontrados no mercado de *PCs* atualmente. Os números, que acompanham os tipos, referem-se às diferentes freqüências oferecidas para aquele tipo de barramento.

A segunda e a terceira colunas informam, respectivamente, a largura dos barramentos, expressa em número de bytes e a taxa de transferência de dados, expressa em milhões de bytes por segundo, para cada um dos tipos de barramento.

A última coluna apresenta o tempo gasto pelo programa para escrever cada coordenada de um ponto na placa ou para ler cada coordenada de um ponto a partir da placa, dado em microssegundos, para os corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$. Para encontrar os valores desta coluna é necessário, primeiramente, calcular $I = (m+1) / (\text{largura do barramento} * 8)$, sendo m o corpo finito. A parte inteira do número I é chamada de J. Para concluir, basta calcular a seguinte fórmula:

$$(J + 1) * (\text{taxa de transferência} / \text{largura do barramento})^{-1} \qquad (5.1)$$

Como dito antes, a lógica para implementar a interface entre a placa e o barramento do *PC* deve descrever adequadamente o padrão de comunicação específico do barramento em uso. Entretanto, para o correto funcionamento da placa, essa lógica deve ser coerente, também, com o fluxo de dados na placa.

A figura 5.3 apresenta o diagrama de fluxo de dados da placa adaptadora para *PC*. Através dela é possível compreender todo o processo de funcionamento da placa, assim como o caminho seguido pelas informações por entre os circuitos que a compõem, conforme é explicado a seguir.

Como citado anteriormente, a placa recebe dados através do barramento de dados do *PC*. Esse barramento possui w bits de largura, sendo w dependente do tipo de barramento empregado no *PC* (vide tabela 5.1). Os dados são enviados à placa, por exemplo, por um programa que implementa o algoritmo do modelo Diffie-Hellman de troca de chaves.

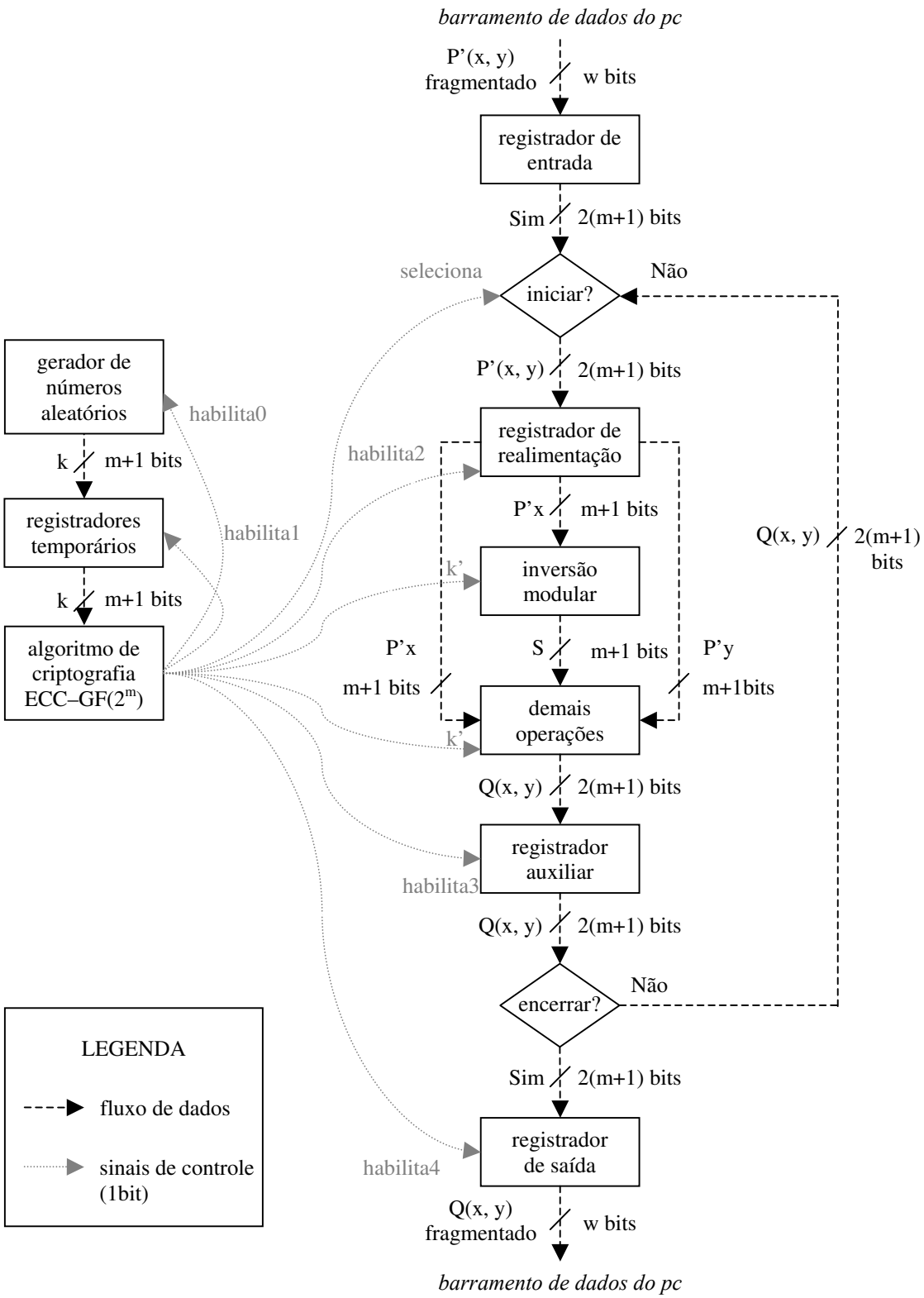

Figura 5.3: Diagrama de fluxo de dados da placa adaptadora para PC.

Esses dados podem representar uma chave pública *Qp*, usada para o cálculo da chave secreta *Qs*, ou o valor *P*, usado para gerar chaves públicas, conforme apresentado na seção 1.1. Tanto *Qp* como *P* são pontos pertencentes à curva elíptica usada pelo sistema criptográfico. Ambos são chamados de *P'(P'x, P'y)* por uma questão de simplificação e cada um possui *2(m+1) bits* de tamanho, ou seja, *m+1 bits* por coordenada, sendo *m* o corpo finito.

O ponto *P'(P'x, P'y)* chega à placa fragmentado em *2(m+1)/w* partes, sendo *w* a largura do barramento dada em número de bits. Esse mesmo ponto é remontado e armazenado no registrador de entrada da placa, começando pelos *w bits* menos significativos até os mais significativos.

Do registrador de entrada, o ponto *P'(P'x, P'y)* segue para o registrador de realimentação, passando antes porém por um multiplexador entre um registrador e outro. Esse multiplexador permite a passagem do ponto *P'(P'x, P'y),* caso o algoritmo de criptografia tenha iniciado seu trabalho naquele instante. Do contrário, um ponto *Q(Qx, Qy)* parcial, proveniente de um registrador auxiliar, ganha a passagem para ser realimentado no circuito, conforme é explicado mais adiante nesta mesma seção.

Mais uma vez, por uma questão de simplificação, qualquer ponto, independentemente de sua origem, ao chegar no registrador de realimentação, passa a ser chamado de *P'(P'x, P'y).* Após serem armazenadas no registrador de realimentação, as coordenadas *P'x* e *P'y* seguem adiante por caminhos independentes.

Primeiramente, a coordenada *P'x* é passada a um circuito combinacional para o cálculo de inversão modular. Em outras palavras, esse circuito usa a coordenada *x* do ponto *P'* para calcular *S*, sendo $S = (P'x)^{-1} \bmod p$. Essa operação faz parte da equação (2.14) apresentada no Capítulo 2.

Em seguida, as coordenadas *x* e *y* do ponto *P'* (*P'x* e *P'y*) e mais o inverso modular de *P'x* (o valor *S*) são passados para um circuito combinacional responsável por realizar as demais operações presentes nas equações de (2.14) a (2.16).

O resultado da passagem do ponto *P'(P'x, P'y)* por esses dois circuitos combinacionais é a geração de um ponto *Q(Qx, Qy)*, o qual é, então, armazenado em um registrador auxiliar. Esse ponto *Q(Qx, Qy)* pode representar um resultado final ou apenas parcial das operações do algoritmo de criptografia.

Caso seja um ponto *Q(Qx, Qy)* parcial, ele deve ser passado para o registrador de realimentação, através do multiplexador comentado anteriormente. Esse processo de realimentação é repetido várias vezes, segundo o controle do algoritmo de criptografia, até o ponto *Q(Qx,Qy)* final ser encontrado. O algoritmo se baseia no valor de uma chave privada *k* para definir quantas vezes a realimentação será repetida. Isso está explicado detalhadamente mais à frente nesta seção.

Uma vez calculado, o ponto *Q(Qx, Qy)* final é passado para o registrador de saída. A partir desse registrador, este ponto segue, fragmentado em *2(m+1)/w* partes, pelo barramento de dados do *PC*.

Então, o programa que implementa o algoritmo do modelo Diffie-Hellman de troca de chaves, finalmente recebe o ponto *Q(Qx, Qy)* aguardado. Vale lembrar que esse novo ponto *Q(Qx, Qy)* pode ser ou uma chave pública *Qp* ou uma chave secreta *Qs*.

O controle das etapas citadas acima fica a cargo do algoritmo de criptografia, conforme explicado a seguir.

O algoritmo de criptografia baseado em curvas elípticas sobre corpos finitos binários recebe uma chave privada *k*, de *m+1 bits* de tamanho, a partir do circuito gerador de números aleatórios.

Com base na interpretação do valor de cada um dos bits da chave *k*, o algoritmo de criptografia controla todo o conjunto de circuitos já citados nesta seção.

A figura 5.4 mostra dois exemplos de como um algoritmo, neste caso o *Double-and-Add*, pode interpretar os bits de *k*, a fim de realizar a duplicação e a adição de pontos. Para esses exemplos, onde *k = 10* e *k = 17*, respectivamente, o algoritmo analisa os bits da esquerda para a direita até encontrar a primeira ocorrência do bit 1. Esse será o bit de

referência. Serão considerados apenas os bits à direita do bit de referência. Dessa forma, o algoritmo interpreta cada bit 0 de *k* como sendo uma ordem para realizar uma operação de duplicação de um ponto e cada ocorrência de um bit 1 de *k* como sendo uma ordem para realizar uma duplicação seguida de uma adição de dois pontos distintos.

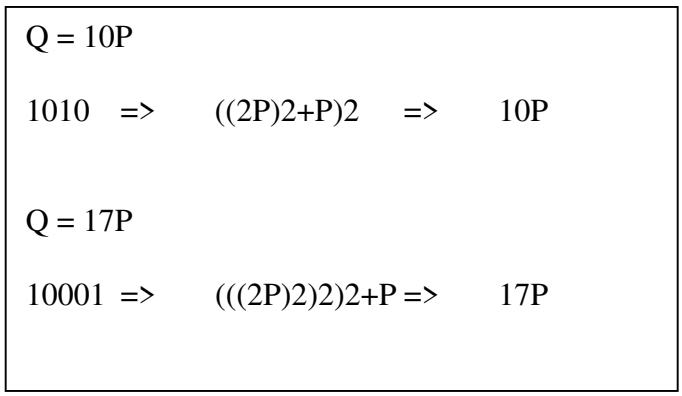

Figura 5.4: Exemplos da interpretação do número inteiro *k* pelo algoritmo de criptografia.

Isso significa que os valores dos bits da chave privada *k* original são submetidos a uma interpretação particular para cada algoritmo de criptografia empregado. O conjunto de bits oriundo dessa interpretação constitui os sinais de controle representados por *k'*, na figura 5.3. Este pode assumir os valores 0 e 1, representando, respectivamente, uma duplicação de um ponto e uma adição de dois pontos distintos, para os circuitos combinacionais responsáveis por realizar essas duas operações.

Há também um sinal de controle para selecionar as entradas do multiplexador. Quando o algoritmo começa a realizar seu trabalho, ele mantém esse sinal em 1, para que o ponto *P'(P'x, P'y)*, proveniente do barramento de dados do *PC*, ganhe acesso aos circuitos da placa. Depois disso, esse sinal é conservado em 0 para permitir o processo de realimentação.

Os demais sinais de controle permitem habilitar ou não a entrada de dados nos registradores, controlando, dessa forma, o fluxo de dados através dos circuitos da placa.

5.4 Implementações

Nesta seção propõe-se a implementação do sistema criptográfico em uma placa adaptadora para *PC*. Essa placa integra circuitos desenvolvidos, compilados e simulados por meio da ferramenta Quartus II v5.0, para *FPGAs* da Altera, conforme mostrado anteriormente.

Existem duas *FPGAs* diferentes presentes na placa. Elas operam em conjunto para resolver a equação $Q = kP$. A primeira delas é a Stratix II EP2S180F1020C4, que implementa apenas o algoritmo de inversão modular. A segunda é a Stratix II EP2S90F1508C3, que implementa todas as demais operações necessárias para resolver a equação $Q = kP$. Além disso, ela também implementa um algoritmo criptográfico, um gerador de números aleatórios, registradores e a interface de comunicação entre a placa e o barramento do *PC*.

A placa pode implementar o sistema criptográfico para um dos quarto corpos finitos $GF(2^m)$ diferentes, usados neste trabalho. Para $GF(2^m)$, são considerados os valores de m iguais a 113, 131, 147 e 163.

As diferentes implementações permitem mostrar quanto tempo cada uma delas consome ao realizar a operação $Q = kP$, para cada uma das quatro configurações do projeto.

O objetivo maior dos testes realizados a partir dessas implementações é obter valores, que mostrem o desempenho do sistema criptográfico proposto, ao realizar sua tarefa.

5.5 Resultados

Esta seção apresenta e comenta os resultados alcançados por meio de cada uma das quatro diferentes implementações citadas na seção 5.4. Tais resultados aparecem resumidos na tabela 5.2. Os gráficos 5.1 e 5.2, mostrados mais adiante, contribuem para facilitar a interpretação desses mesmos resultados.

A tabela 5.2 concentra os resultados obtidos com as implementações do sistema criptográfico na placa adaptadora para *PC*, para cada um dos quatro corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$.

Tabela 5.2: Resultados obtidos com o sistema criptográfico.

Corpo Finito	**Área Máxima Estimada (LUTs)**	**Tempo Médio de Processamento (μs)**	**Tempo de Compilação Previsto (dd:hh:mm:ss)**
113	132.482	46,655	+ de 01:22:22:52
131	159.381	62,684	+ de 03:11:35:31
147	186.289	83,185	+ de 05:25:31:21
163	216.288	100,050	+ de 10:15:50:54

Cada linha da tabela refere-se a um dos quatro corpos finitos apresentados na primeira coluna.

A segunda coluna mostra uma estimativa da área máxima ocupada pelo conjunto de circuitos implementados nas *FPGAs* que constituem a placa. Esses valores são dados em número de *LUTs*.

As estimativas mostradas na segunda coluna são feitas com base na quantidade máxima de *LUTs* disponível para a implementação do algoritmo criptográfico, do *RNG*, dos registradores e da lógica para a interface com o barramento, em uma implementação do corpo finito 163.

Em outras palavras, a placa dispõe de um total de 216.288 *LUTs*, distribuídas em duas *FPGAs* distintas. Desse total, considerando o maior corpo finito implementado, isto é $GF(2^{163})$, 158.796 *LUTs* são empregadas pelo circuito que realiza a duplicação e a adição de pontos, conforme apresentado na seção 4.5. As demais 57.492 *LUTs* restantes, são disponibilizadas para implementar os circuitos restantes.

Dessa forma, estima-se que, se essas 57.492 *LUTs* são suficientes para implementar os demais circuitos para o maior dos corpos finitos, então elas também são suficientes para os mesmos circuitos nas implementações dos outros três corpos finitos menores.

Desse modo, os valores da segunda coluna da tabela 5.2 são resultantes da soma dessas 57.492 *LUTs* com cada um dos valores da segunda coluna da tabela 4.1.

A terceira coluna mostra quantos microssegundos, em média, são necessários para a placa efetuar o cálculo $Q = kP$. Esses valores são baseados em implementações que usam o algoritmo de criptografia *Double-and-Add*. Para ele, considera-se um total de *m* operações, sejam duplicações de ponto, adições de pontos ou ambas, a fim de encontrar o tempo médio de processamento da placa (para saber maiores detalhes, vide seção 6.2.1). Vale lembrar que *m* representa o corpo finito.

O emprego de outros algoritmos [G98], [GO98] e [DO03] pode fornecer resultados ainda melhores que os apresentados na tabela 5.2.

A última das colunas informa quantos dias, horas, minutos e segundos um *PC* Pentium 4, trabalhando a 3,2 GHz, com 1 GB de memória RAM e 30 GB de HD, gasta para compilar o conjunto de circuitos empregados na placa.

O gráfico 5.1 apresenta relações entre os corpos finitos e a área ocupada (em número de *LUTs*) pelas respectivas implementações do sistema criptográfico na placa.

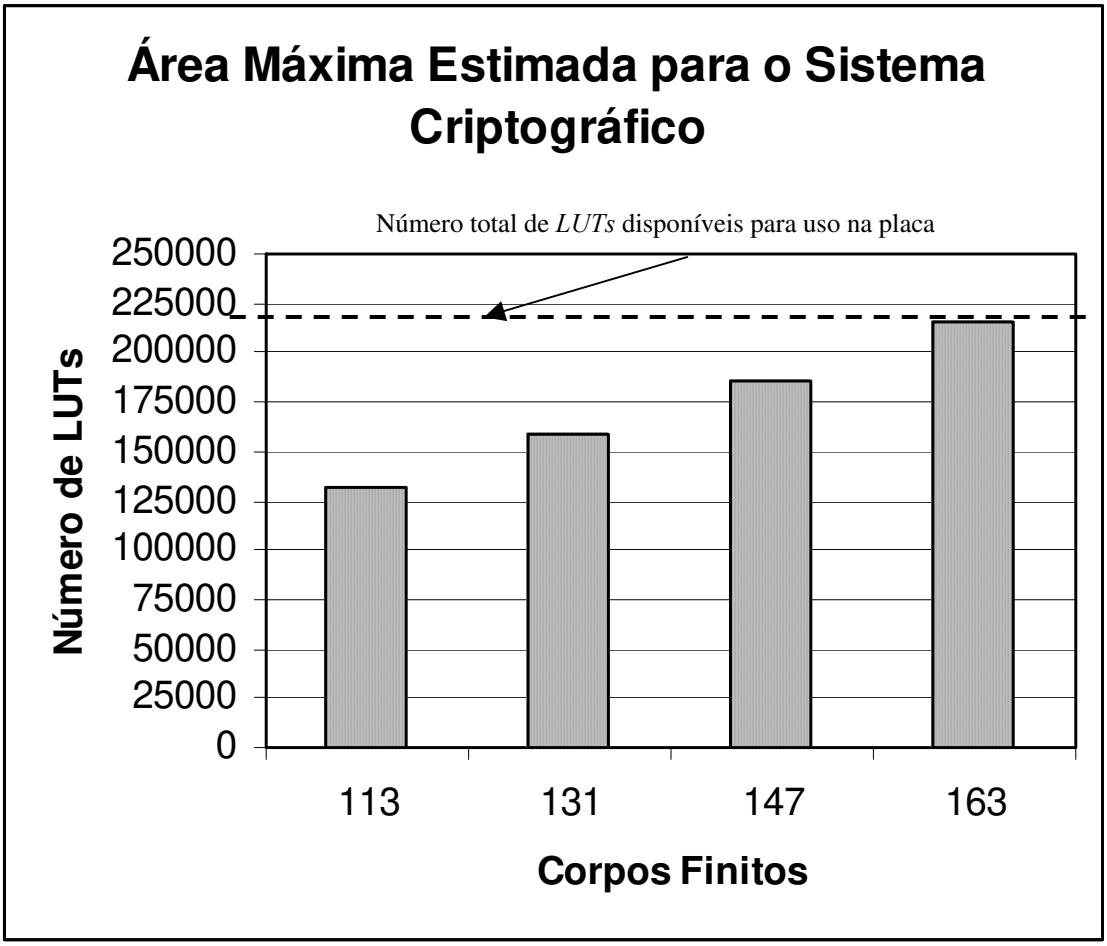

Gráfico 5.1: Área máxima estimada para o sistema criptográfico.

Os diferentes corpos finitos binários aparecem representados pelo eixo das ordenadas.

Já o eixo das abscissas informa a quantidade de *LUTs* usada para cada implementação.

Pela observação do gráfico 5.1 é possível constatar o seguinte: estima-se que as implementações do sistema criptográfico para os corpos finitos 113, 131, 147 e 163 deixam de utilizar, respectivamente, 39%, 26%, 14% e 0% do total de *LUTs* disponíveis para uso na placa.

O gráfico 5.2 mostra a relação existente entre o corpo finito binário usado na implementação do sistema criptográfico e o tempo (em microssegundos) consumido por ele ao realizar uma operação $Q = kP$.

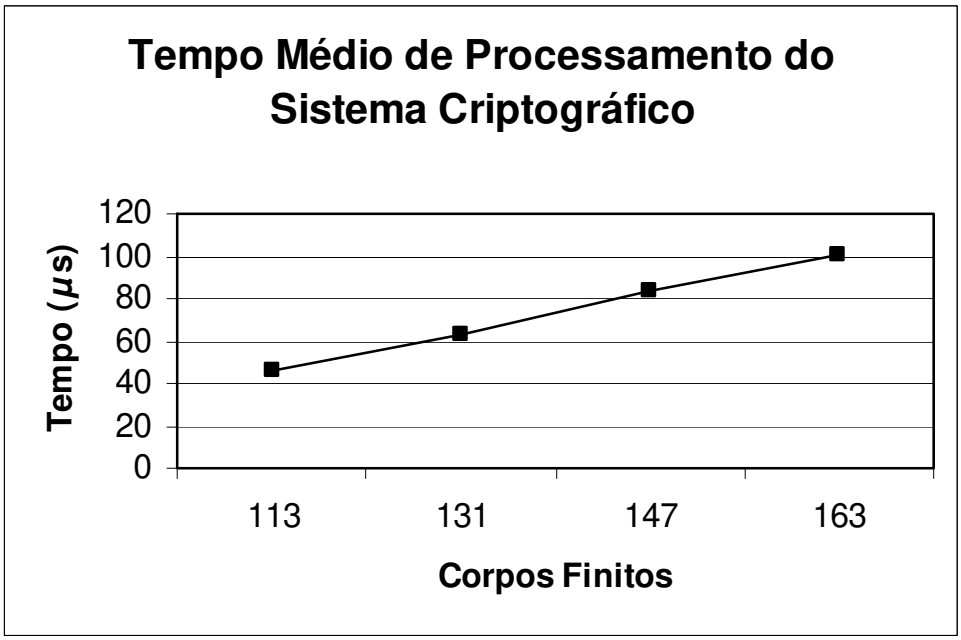

Gráfico 5.2: Tempo médio de processamento do sistema criptográfico.

O eixo das ordenadas apresenta os valores referentes aos diferentes corpos finitos usados para as implementações.

O tempo necessário para o sistema criptográfico realizar seu trabalho aparece representado pelo eixo das abscissas.

Observando o gráfico 5.2, constata-se o seguinte: o gráfico do tempo é quase linear, porque o tempo de processamento do sistema criptográfico aumenta proporcionalmente ao corpo finito e esse tempo de processamento não sofre variações bruscas entre as implementações feitas na placa.

5.6 Comentários Finais

Este capítulo apresentou um sistema criptográfico para curvas elípticas sobre corpos finitos binários, implementado em uma placa adaptadora para *PC*.

Seu desempenho ao efetuar a operação $Q = kP$ é da ordem de algumas dezenas de microssegundos, por isso ele é indicado para auxiliar programas de alto desempenho que trabalham com criptografia assimétrica, operando diretamente com coordenadas afins.

O desempenho do sistema criptográfico é favorecido pelo fato do tempo gasto por um programa para escrever um ponto na placa, somado ao tempo gasto para ele ler um ponto a partir da placa, não ultrapassar o tempo de processamento da própria placa. Isso ocorre para qualquer um dos barramentos apresentados pela tabela 5.1.

Isso implica que, enquanto a placa realiza o processamento para encontrar o ponto *Q(Qx, Qy)* final, paralelamente, o registrador de entrada da placa já pode receber um novo ponto *P'(P'x, P'y)*. Terminado esse recebimento, um outro ponto *Q(Qx, Qy)* final, fruto de um processamento anterior ao atual, pode deixar o registrador de saída da placa, onde ele aguardava para ser enviado pelo barramento de dados do *PC*.

Conseqüentemente, como o tempo de processamento da placa é superior ao tempo necessário para ela receber um novo ponto *P'(P'x, P'y)* e enviar um ponto *Q(Qx, Qy)* calculado anteriormente, o tempo de uso do barramento de dados do *PC* torna-se desprezível. Assim sendo, considerando o uso constante da placa, o tempo total necessário para o cálculo $Q = kP$ limita-se ao tempo do processamento da própria placa.

Os resultados alcançados apontam que o sistema criptográfico implementado na placa é capaz de realizar a operação $Q = kP$, nos tempos 46,655 μs, 62,684 μs, 83,185 μs e 100,050 μs, respectivamente, para os corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{131})$ e $GF(2^{163})$.

Estes e outros resultados são apresentados mais detalhadamente no próximo capítulo.

Capítulo 6
Resultados Finais

6.1 Introdução

Este capítulo apresenta os resultados finais deste trabalho, obtidos a partir das compilações e das simulações do sistema criptográfico. A ferramenta usada para realizar tais compilações e simulações apresentadas é o Quartus II, da Altera.

O objetivo deste capítulo é apresentar dados relativos à velocidade e à área dos circuitos empregados para compor o sistema, assim como valores referentes ao tempo consumido durante a compilação dos mesmos.

Além disso, o capítulo também mostra valores relacionados à questão de desempenho entre trabalhos em *software* e *hardware*, a fim de estabelecer relações com o sistema criptográfico.

As implementações descritas nos capítulos anteriores permitem alcançar os resultados mostrados nas tabelas deste capítulo. As mesmas serão explicadas detalhadamente mais à frente.

Os resultados apresentados neste capítulo são frutos de diversas simulações realizadas com as implementações do sistema criptográfico para os corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$. Para cada um desses corpos finitos, esses resultados dependem dos parâmetros que definem os pontos das suas respectivas curvas elípticas.

Para $GF(2^{113})$, $GF(2^{131})$ e $GF(2^{163})$ os parâmetros escolhidos para definir as curvas elípticas, assim como seus pontos, estão em conformidade com os padrões *NIST, IEEE P1363, IPSec, WAP, eCheck, ANSI X9.62* e *ANSI X9.63* [SEC2-00]. O corpo finito $GF(2^{147})$ foi escolhido aleatoriamente e não faz parte de nenhum padrão. Os demais corpos finitos foram escolhidos de modo a evitar a construção de um circuito excessivamente denso, visto que são os menores corpos finitos padronizados.

Todos os resultados exibidos aqui são alcançados pelos circuitos descritos, quando os mesmos empregam, respectivamente para os corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$, os seguintes parâmetros para definição de curvas elípticas:

Para $GF(2^{113})$:

p: $X^{113} + X^9 + 1$

a: 003088250CA6E7C7FE649CE85820F7

X: 009D73616F35F4AB1407D73562C10F

Y: 00A52830277958EE84D1315ED31886

Para $GF(2^{131})$:

p: $X^{131} + X^8 + X^3 + X^2 + 1$

a: 03E5A88919D7CAFCBF415F07C2176573B2

X: 0356DCD8F2F95031AD652D23951BB366A8

Y: 0648F06D867940A5366D9E265DE9EB240F

Para $GF(2^{147})$:

p: $X^{147} + X^7 + X^6 + X^3 + 1$

a: 000000000000000000000000000000000001

X: BA16286A2D57EA0991168D4994637E8343E36

Y: FBC6C71A0094FA2CDD545B11C5C0C797324F1

Para $GF(2^{163})$:

p: $X^{163} + X^7 + X^6 + X^3 + 1$

a: 000000000000000000000000000000000000001

X: 3F0EBA16286A2D57EA0991168D4994637E8343E36

Y: 0D51FBC6C71A0094FA2CDD545B11C5C0C797324F1

A organização deste capítulo é dada da seguinte maneira: a seção 6.2 exibe os resultados das simulações, sendo dividida em duas partes: a subseção 6.2.1 aborda os resultados relativos à velocidade dos circuitos e a subseção 6.2.2 abrange os resultados referentes à área dos circuitos; a seção 6.3 mostra os resultados das compilações; a seção 6.4 apresenta os resultados de desempenho de implementações em *software* e *hardware*. A seção 6.5 discute os resultados referentes a um modelo de troca de chaves. O capítulo é finalizado com os comentários finais na seção 6.6.

6.2 Resultados das Simulações

Nesta seção encontram-se descritos todos os resultados das simulações referentes à velocidade e à área dos circuitos que compõem o sistema criptográfico.

6.2.1 Velocidade

Esta subseção comenta todos os resultados relacionados à velocidade dos circuitos, alcançados através das simulações do sistema criptográfico, começando pela tabela 6.1.

Tabela 6.1: Tempo gasto pelo sistema criptográfico para realizar suas operações.

Corpo Finito	**Tempo Gasto pela Inversão Modular (ns)**	**Tempo Gasto pelas Demais Operações (ns)**	**Tempo Gasto pela Adição ou Duplicação de Ponto (ns)**	**Tempo Médio Gasto pela Operação $Q = kP$ (μs)**
113	323,779	89,094	412,873	46,655
131	374,252	104,249	478,501	62,684
147	448,885	116,998	565,883	83,185
163	490,559	123,242	613,801	100,050

A tabela 6.1 apresenta o tempo gasto pelo sistema criptográfico para realizar: a operação de inversão modular presente na equação (2.14); as demais operações presentes nas equações de (2.14) a (2.16); a duplicação de um ponto ou a adição de dois pontos distintos; a operação $Q = kP$. Todas elas simuladas para os corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$, representados pela primeira coluna.

Os valores da segunda coluna são resultantes do atraso da propagação de sinais através do circuito combinacional para a inversão modular.

A terceira coluna apresenta valores oriundos do atraso da propagação de sinais através do circuito combinacional para as operações presentes nas equações de (2.14) a (2.16), exceto a inversão modular, segundo detalhado pelo Capítulo 4.

O tempo consumido pela duplicação ou pela adição de pontos, apresentado pela quarta coluna, é resultante da soma dos valores presentes nas duas colunas anteriores.

Quanto à última coluna, para realizar a operação $Q = kP$, o sistema criptográfico realiza uma média de m operações, dentre duplicações e adições de pontos, onde m representa o corpo finito.

Sendo assim, basta multiplicar o tempo gasto por uma operação de duplicação ou de adição de pontos pelo número médio dessas operações dentro do calculo $Q = kP$, que é dado por m. O resultado dessa conta é apresentado na quinta coluna, para cada um dos quatro corpos finitos binários distintos.

A quantidade de duplicações e adições de pontos que ocorrem durante o cálculo $Q = kP$ depende do número inteiro k e do algoritmo empregado no sistema criptográfico.

Dessa forma, para um mesmo algoritmo criptográfico, o tempo gasto pela operação $Q = kP$ varia em função do valor k.

Isso pode ser exemplificado através das tabelas 6.2, 6.3, 6.4 e 6.5, referentes, respectivamente, aos corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$. O algoritmo empregado para obter os valores dessas tabelas é o *Double-and-Add*.

A tabela 6.2 apresenta o tempo, em microssegundos, gasto pelo sistema criptográfico ao realizar a operação $Q = kP$ para diferentes valores de k, considerando o corpo finito $GF(2^{113})$.

Tabela 6.2: Tempo gasto pela operação $Q = kP$ em função do inteiro k para $GF(2^{113})$.

k **(hexadecimal)**	***Q = kP*** **(μs)**
00000000000000000000000000002	0,412873
20000000000000000000000000000	46,655
3FFFFFFFFFFFFFFFFFFFFFFFFFFFF	93,309

A primeira coluna da tabela 6.2 mostra três valores diferentes para representar o número k: um valor baixo, referente a uma única operação de duplicação de ponto; um valor médio, equivalente a m operações de duplicação e/ou adição de pontos; um valor alto, indicando a ocorrência de $2m$ operações de duplicação e/ou adição de pontos, sendo m o corpo finito. Todos esses valores são representados em hexadecimal.

A segunda coluna mostra o tempo gasto pela operação $Q = kP$ para cada um dos valores de k presentes na coluna 1.

Quando o sistema criptográfico emprega o algoritmo *Double-and-Add* para o cálculo $Q = kP$, ele analisa um a um os bits de k da esquerda para a direita (vide seção 5.3). Após o primeiro bit 1 encontrado: cada ocorrência de um bit 0 em k representa apenas uma duplicação de ponto; cada ocorrência de um bit 1 em k representa uma duplicação seguida por uma adição de pontos.

Os valores da segunda coluna são então encontrados somando os tempos de todas as duplicações e adições efetuadas durante o cálculo $Q = kP$. A fim de obter um resultado ainda mais exato para compor os valores apresentados na segunda coluna, nós consideramos também o tempo de atraso do circuito de realimentação, descrito no Capítulo 5.

A tabela 6.3 mostra quanto tempo o sistema criptográfico leva para efetuar a operação $Q = kP$ para o corpo finito $GF(2^{131})$, considerando diferentes valores de k. O tempo é dado em microssegundos.

Tabela 6.3: Tempo gasto pela operação $Q = kP$ em função do inteiro k para $GF(2^{131})$.

k **(hexadecimal)**	***Q = kP*** **(μs)**
00000000000000000000000000000002	0,478501
80000000000000000000000000000000	62,684
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	125,367

Três valores diferentes, um baixo (uma operação de duplicação de ponto), um médio (m operações de duplicação e/ou adição de pontos) e um alto ($2m$ operações de duplicação e/ou adição de pontos), apresentados em hexadecimal, são usados para representar o número k na primeira coluna dessa tabela.

O tempo gasto pela operação $Q = kP$ para cada um desses valores de k é calculado seguindo o mesmo conjunto de procedimentos descritos para o cálculo da coluna 2 da tabela 6.2. Os resultados obtidos são exibidos na segunda coluna da tabela 6.3.

A tabela 6.4 considera o corpo finito $GF(2^{147})$, ao mostrar o tempo, em microssegundos, consumido durante a operação $Q = kP$, realizada pelo sistema criptográfico.

Tabela 6.4: Tempo gasto pela operação $Q = kP$ em função do inteiro k para $GF(2^{147})$.

k **(hexadecimal)**	***Q = kP*** **(μs)**
000000000000000000000000000000000000002	0,565883
800000000000000000000000000000000000000	83,185
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	166,370

A primeira coluna dessa tabela apresenta, em hexadecimal, um valor baixo (uma operação de duplicação de ponto), um médio (m operações de duplicação e/ou adição de pontos) e um alto ($2m$ operações de duplicação e/ou adição de pontos), para representar cada ocorrência do número k.

Os valores apresentados na segunda coluna da tabela 6.4 são calculados conforme o conjunto de procedimentos descritos para o cálculo da coluna 2 da tabela 6.2. Eles representam o tempo gasto pela operação $Q = kP$, para cada um desses valores de k.

Considerando o corpo finito $GF(2^{163})$, a tabela 6.5 exibe, para diferentes valores de k, quantos microssegundos o sistema criptográfico gasta ao realizar a operação $Q = kP$.

Tabela 6.5: Tempo gasto pela operação $Q = kP$ em função do inteiro k para $GF(2^{163})$.

k **(hexadecimal)**	***Q = kP*** **(μs)**
000000000000000000000000000000000000000002	0,613801
800000000000000000000000000000000000000000	100,050
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	200,100

O número k é representado por três diferentes valores em hexadecimal na primeira coluna dessa tabela, os quais referem-se, respectivamente, a 1, m e $2m$ operações de duplicação e/ou adição de pontos.

A segunda coluna mostra o tempo gasto pela operação $Q = kP$ para cada um dos valores de k presentes na coluna 1. Esses resultados são obtidos seguindo o mesmo conjunto de procedimentos descritos para o cálculo da coluna 2 da tabela 6.2.

Tomando por base os valores apresentados pelas tabelas 6.2, 6.3, 6.4 e 6.5, mostradas anteriormente, é possível representar graficamente as relações existentes entre os seus valores, considerando os diferentes corpos finitos.

Dessa forma, a relação entre a quantidade de duplicações e/ou adições de pontos e o tempo (em microssegundos) que o sistema criptográfico gasta para realizar uma operação $Q = kP$ é apresentada pelo gráfico 6.1.

O eixo das ordenadas compreende os valores referentes às diferentes quantidades de duplicações e/ou adições de pontos, dadas em função de m, que também representa o corpo finito.

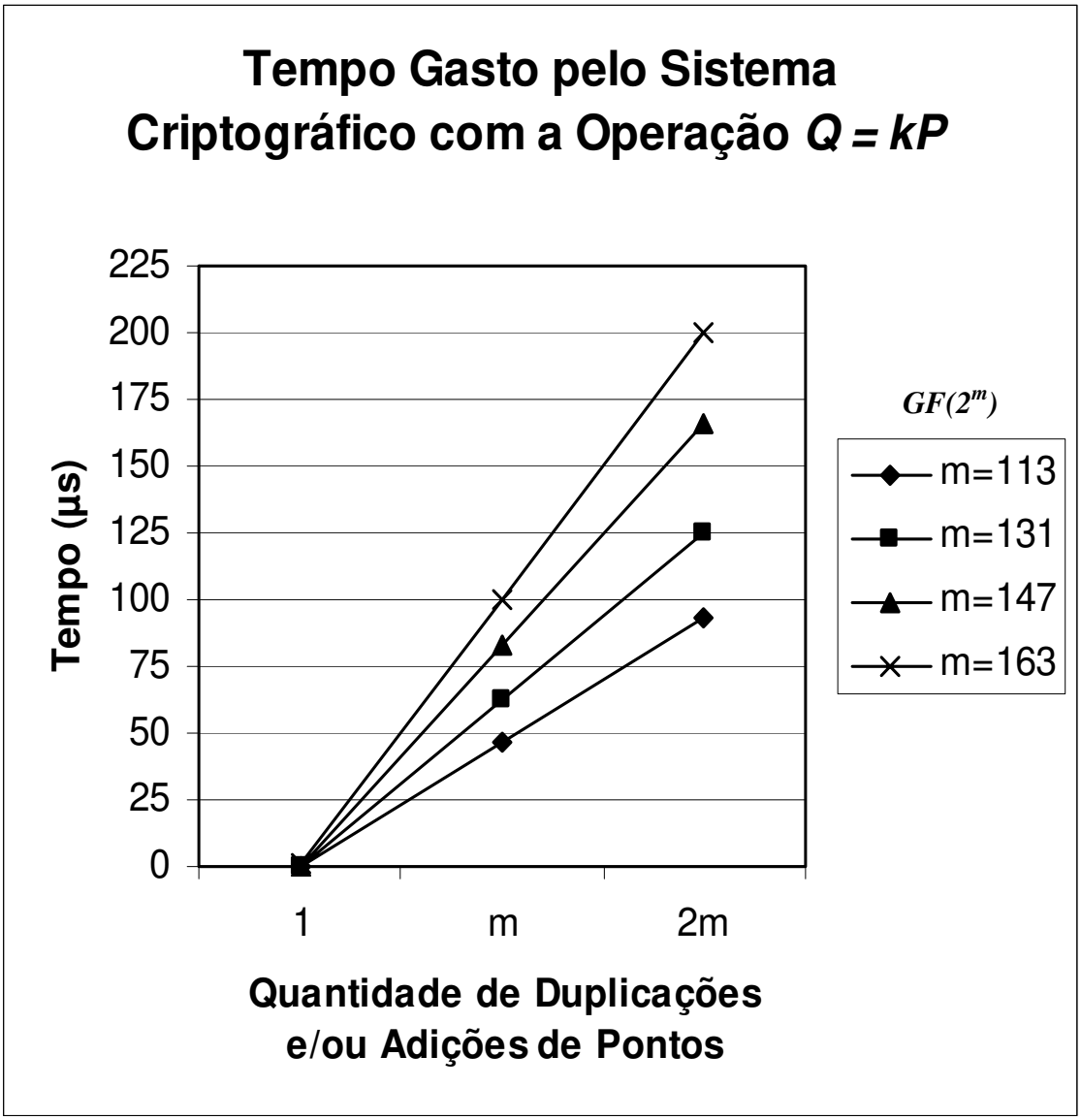

Gráfico 6.1: Tempo gasto pelo sistema criptográfico com a operação $Q = kP$ em relação à quantidade de duplicações e/ou adições de pontos.

Os valores referentes ao tempo gasto pelo sistema criptográfico ao realizar seu trabalho são mostrados através do eixo das abscissas.

Cada corpo finito é representado por um tipo de linha diferente, para facilitar sua identificação.

A observação do gráfico 6.1 permite a seguinte constatação: entre os diferentes corpos finitos, há uma dispersão do tempo gasto pelo sistema criptográfico com a operação $Q = kP$, conforme o número de operações de duplicação e/ou adição de pontos aumenta.

6.2.2 Área

Esta subseção comenta números relacionados às áreas ocupadas pelos circuitos do sistema criptográfico proposto neste trabalho, começando pela tabela 6.6. Eles resultam das simulações desse mesmo sistema.

Tabela 6.6: Área ocupada pelos circuitos que compõem o sistema criptográfico.

Corpo Finito	Área do Circuito para a Inversão Modular (LUTs)	Área do Circuito para as Demais Operações (LUTs)	Área do Circuito para a Duplicação ou a Adição de Pontos (LUTs)	Área Máxima Ocupada por: *RNG*, Algoritmo e Interface (LUTs)	Área Total do Sistema Criptográfico (LUTs)
113	60.361	14.629	74.990	57.492	132.482
131	82.082	19.807	101.889	57.492	159.381
147	104.179	24.618	128.797	57.492	186.289
163	128.265	30.531	158.796	57.492	216.288

A tabela 6.6 apresenta a área ocupada pelos circuitos que compõem o sistema criptográfico, em implementações para os corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$.

Cada linha da tabela refere-se a um dos quatro corpos finitos apresentados pela primeira coluna.

A segunda coluna mostra a área ocupada pelo circuito combinacional responsável por efetuar a operação de inversão modular, presente na equação (2.14), para o sistema criptográfico.

A terceira coluna exibe a área ocupada pelo circuito combinacional empregado para realizar as demais operações, presentes nas equações de (2.14) a (2.16), para o sistema criptográfico.

A área do circuito responsável pela duplicação e pela adição de pontos é apresentada através da quarta coluna. Os valores dessa coluna são resultantes da soma das respectivas áreas mostradas nas duas colunas anteriores.

A quinta coluna traz a área máxima empregada para implementar o gerador de números aleatórios, o algoritmo criptográfico e a interface de comunicação com o barramento (para saber maiores detalhes, vide seção 5.5).

Por último, na sexta coluna, estão os dados referentes à área total ocupada por todo o conjunto de circuitos que constituem o sistema criptográfico, para cada um dos quatro corpos finitos binários distintos.

Uma comparação entre os valores apresentados pela tabela 6.6 pode ser mais claramente demonstrada com a ajuda do gráfico 6.2, que mostra a porção da área do sistema criptográfico ocupada individualmente pelos circuitos que o compõem, isto é, o circuito combinacional responsável por efetuar a operação de inversão modular, o circuito combinacional empregado para realizar as demais operações, o gerador de números aleatórios, o algoritmo criptográfico e a interface de comunicação.

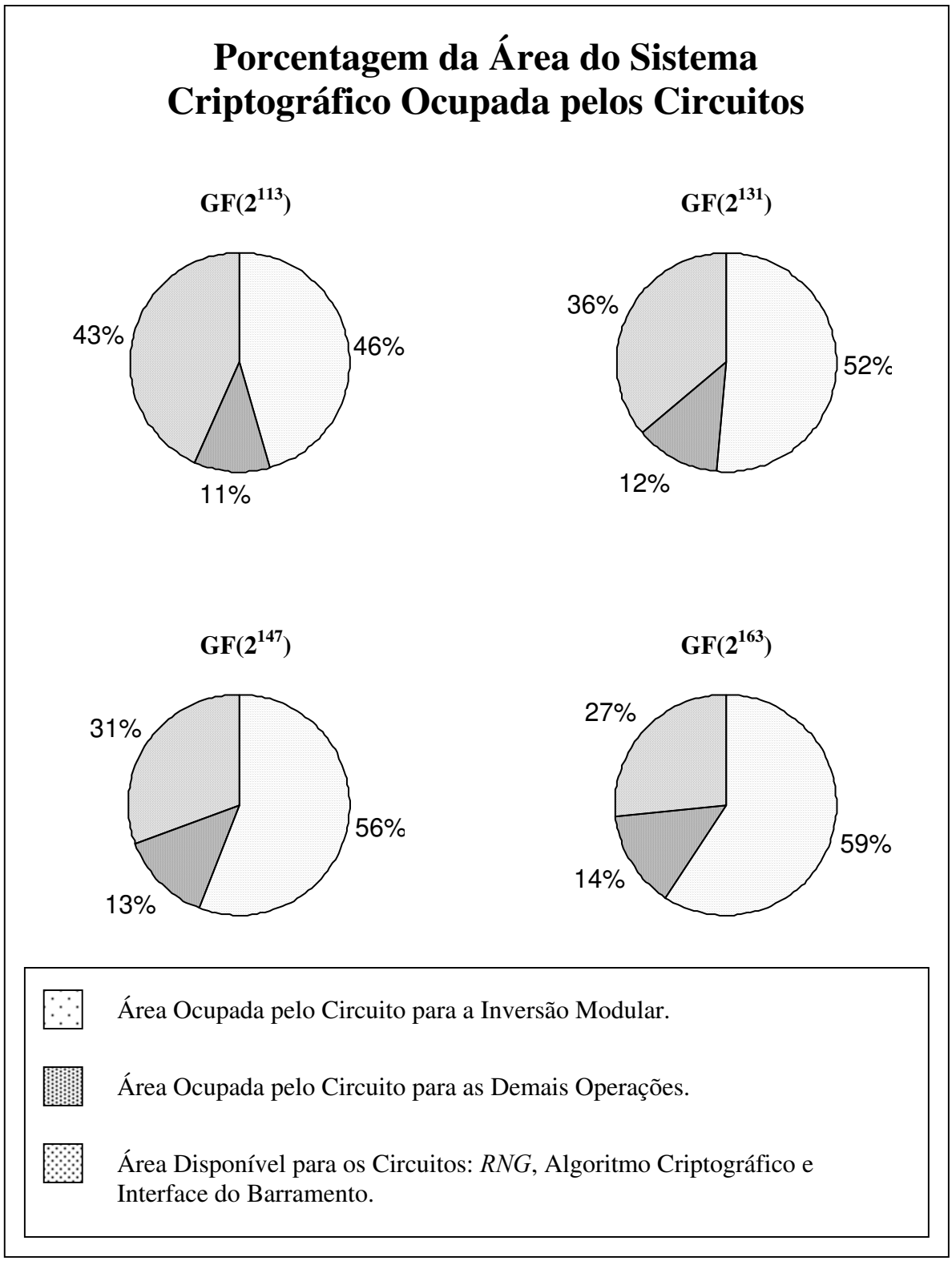

Gráfico 6.2: Porcentagem da área do sistema criptográfico ocupada pelos circuitos.

Por este motivo, o gráfico 6.2 estabelece relações entre a área do circuito combinacional responsável por efetuar a operação de inversão modular, a área do circuito combinacional empregado para realizar as demais operações e a área disponível para os circuitos que implementam o gerador de números aleatórios, o algoritmo criptográfico e a interface de comunicação.

Houve uma divisão do gráfico em quatro partes, uma para cada um dos quatro corpos finitos diferentes. Sendo assim, o gráfico 6.2 apresenta-se constituído por quatro gráficos menores, cada um representando as áreas, dadas em porcentagem, ocupadas por esses mesmos circuitos, referentes a um corpo finito específico.

É possível fazer a seguinte constatação através da observação do gráfico 6.2: o circuito combinacional responsável por efetuar a operação de inversão modular ocupa a fatia mais significativa da área do sistema criptográfico, independentemente do corpo finito.

6.3 Resultados das Compilações

As compilações do sistema criptográfico foram feitas através do programa Quartus II, executado sobre um processador Intel Pentium 4, de 3,2 GHz de freqüência, com 1 GB de memória RAM e 30 GB de HD.

A tabela 6.7 apresenta o tempo gasto para a compilação dos circuitos do sistema criptográfico, para os corpos finitos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$. Os corpos finitos aparecem representados pela primeira coluna.

Os dados da segunda coluna revelam quanto tempo é preciso esperar para compilar o circuito de inversão modular, descrito no Capítulo 3.

A terceira coluna exibe o tempo consumido durante a compilação do circuito combinacional para as operações presentes nas equações de (2.14) a (2.16), exceto a inversão modular, explicado no Capítulo 4.

O circuito que faz a duplicação ou a adição de pontos é compilado no tempo apresentado pela quarta coluna.

Finalmente, a última coluna mostra quanto tempo leva para compilar por completo o sistema criptográfico proposto neste trabalho.

Os tempos são dados em dias, horas, minutos e segundos para cada uma das quatro últimas colunas da tabela.

Tabela 6.7: Tempo gasto para a compilação dos circuitos do sistema criptográfico.

Corpo Finito	Inversão Modular (dd:hh:mm:ss)	Demais Operações (dd:hh:mm:ss)	Duplicação ou Adição de Pontos (dd:hh:mm:ss)	Sistema Criptográfico na Placa (dd:hh:mm:ss)
113	01:18:41:19	03:41:33	01:22:22:52	+ de 01:22:22:52
131	03:04:51:36	06:43:55	03:11:35:31	+ de 03:11:35:31
147	05:16:56:59	08:34:22	05:25:31:21	+ de 05:25:31:21
163	10:03:31:41	12:19:13	10:15:50:54	+ de 10:15:50:54

Os valores apresentados na segunda e na terceira colunas da tabela 6.7 também podem ser relacionados de forma gráfica, como pode ser visto no gráfico 6.3.

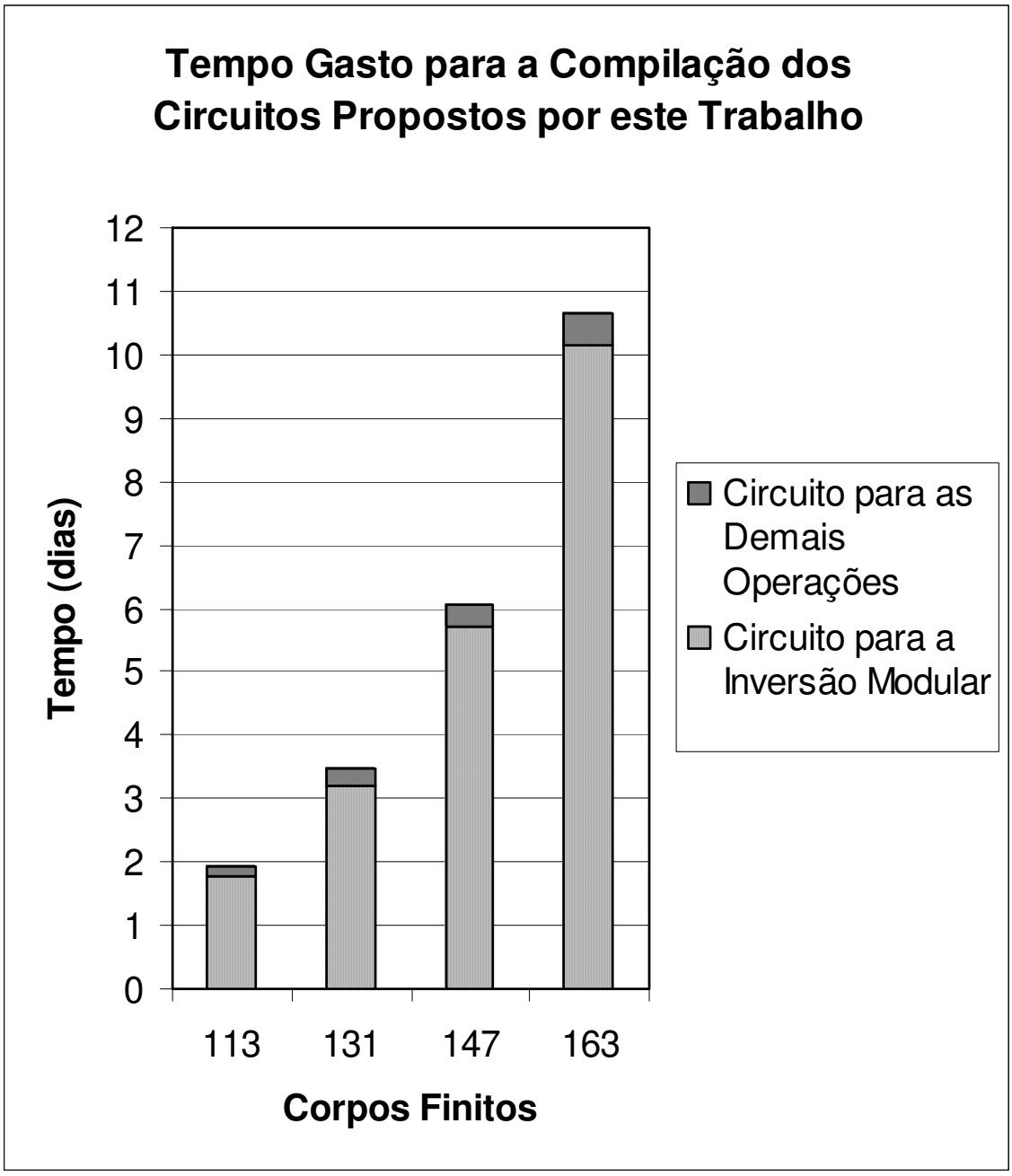

Gráfico 6.3: Tempo gasto para a compilação dos circuitos propostos neste trabalho.

O gráfico 6.3 apresenta relações entre os corpos finitos binários usados nas implementações dos circuitos combinacionais e o tempo (em dias) que estes levam para serem compilados.

Os valores referentes aos diferentes corpos finitos são apresentados pelo eixo das ordenadas.

O eixo das abscissas compreende os valores referentes ao tempo, dado em dias, gasto para compilar os dois circuitos combinacionais.

Pela observação do gráfico 6.3, constata-se o seguinte: para qualquer um dos corpos finitos, o tempo gasto para compilar o circuito para a inversão modular é muito mais significativo que o tempo consumido pela compilação do circuito para as demais operações.

6.4 Resultados de Desempenho

Esta seção tem como objetivo reunir um conjunto de informações que permitem relacionar o desempenho deste sistema criptográfico com implementações publicadas através de outros trabalhos. Essas implementações estão reunidas nesta seção, pelo fato de todas trabalharem com curvas elípticas, muito embora elas empreguem algoritmos bastante distintos para alcançar um mesmo fim: o cálculo da operação $Q = kP$.

A tabela 6.8 mostra as variações de desempenho entre várias implementações diferentes feitas em *software* ou em *hardware*.

A primeira coluna identifica cada uma das nove implementações listadas na tabela, assim como suas respectivas fontes. Os dados presentes em cada linha das demais colunas referem-se à implementação especificada mais à esquerda.

A segunda coluna distingue as implementações feitas em *software* daquelas feitas em *hardware*.

O corpo finito é apresentado na terceira coluna.

Na quarta coluna são apresentadas as plataformas usadas para desenvolver cada uma das implementações.

Tabela 6.8: Desempenho de implementações em *software* ou *hardware*.

Implementação	***Software ou Hardware***	**Corpo Finito**	**Plataforma**	***Q = kP* (ms)**
Montgomery [LD99]	*Software*	$GF(2^{163})$	UltraSparc 64-bit 300 Mhz	13,5
Almost Inv. [SOOS95]	*Software*	$GF(2^{155})$	DEC Alpha 64-bit 175 MHz	7,8
ASIC Coprocessor [AMV93]	*Hardware*	$GF(2^{155})$	VLSI 40 MHz	3,9 est.
FPGA Coprocessor [SES98]	*Hardware*	$GF(2^{155})$	Xilinx FPGA XC4020XL, 15 MHz	18,4 est.
Composite Fields [R98]	*Hardware*	$GF(((2^4)^2)^{21})$ $GF((2^8)^{21})$	Xilinx FPGA XC4062, 16 MHz	4.5 est.
ECP [OP00]	*Hardware*	$GF(2^{167})$	Xilinx FPGA XCV400E, 76,7 MHz	0,21
Montgomery [EGSG03]	*Software*	$GF(2^{163})$	Sun Fire™280R Server 900 MHz	3,11
Cryptographic Processor [EGSG03]	*Hardware*	$GF(2^{163})$	Xilinx Virtex-II XCV2000E-7 FPGA 66,4 MHz	0,14
Sistema Criptográfico Proposto neste Trabalho	*Hardware*	$GF(2^{163})$	Altera FPGAs Stratix II EP2S180F1020C4, 250 MHz e EP2S90F1508C3, 250 MHz	0,1

Finalmente, na quinta e última coluna, aparece o tempo necessário para a realização da operação $Q = kP$, dado em milissegundos, para cada uma das implementações.

As informações apresentadas nas seis primeiras linhas da tabela 6.8 são encontradas juntas em [OP00]. Os dados da antepenúltima e da penúltima linhas podem ser encontrados juntos em [EGSG03]. A última linha da mesma tabela apresenta informações referentes a este trabalho.

O sistema criptográfico proposto se mostra como um dos mais rápidos dentre todas as implementações apresentadas pela tabela 6.8,

6.5 Resultados Referentes a um Modelo de Troca de Chaves

O programa desenvolvido pelo REGRA_C e comentado na seção 5.2 (para saber maiores detalhes, vide [MAH01]) implementa o modelo de troca de chaves Diffie-Hellman, descrito no Capítulo 1. Esta seção discute valores referentes ao desempenho desse programa.

A tabela 6.9 mostra a diferença entre os tempos médios gastos para a execução desse programa sem ou com a ajuda do sistema criptográfico implementado, por exemplo, em uma placa adaptadora conectada a um barramento *ISA* de um *PC*.

Na primeira coluna encontram-se os quatro corpos finitos binários, aos quais os valores das duas colunas seguintes se referem.

A segunda coluna exibe o tempo médio aproximado necessário para executar o modelo inteiramente em *software*. Os dados dessa coluna são obtidos quando o programa fornecido pelo REGRA_C é executado pelo sistema utilizado.

A terceira coluna revela quanto tempo, em média, é gasto para a execução do programa de troca de chaves do REGRA_C, quando o mesmo é auxiliado pelo *hardware* deste sistema criptográfico.

Conforme explicado no Capítulo 1, o modelo de troca de chaves requer a execução do cálculo $Q = kP$ duas vezes. Sendo assim, os resultados apresentados pela tabela 6.9 representam o dobro do tempo necessário para realizar uma operação $Q = kP$, sem ou com a ajuda deste sistema criptográfico.

Para essa tabela, foram desconsiderados os tempos gastos pelos meios de comunicação e armazenamento das chaves durante o processo de troca de chaves, em função de existir um grande número de combinações possíveis para os mesmos.

Para compor os valores presentes na terceira coluna, deve-se considerar ainda o tempo de acesso à placa através do barramento *ISA* (vide Tabela 5.1).

Tabela 6.9: Tempo médio gasto para executar um programa sem ou com a ajuda do sistema criptográfico.

Corpo Finito	**Diffie-Hellman**	
	Sem Placa (seg)	**Com Placa (*μs*)**
113	1	125,309
131	2	161,367
147	3	206,370
163	5	244,100

Para realizar cada cálculo $Q = kP$ por completo, a placa recebe um ponto *P'(P'x, P'y)* e retorna como resultado um ponto *Q(Qx, Qy)*.

Como são quatro coordenadas de ponto envolvidas em cada uma das duas operações de $Q = kP$, deve-se multiplicar o valor do tempo de acesso à placa por oito. Isso vale para qualquer um dos corpos finitos.

Por exemplo, considerando $GF(2^{163})$ e o barramento *ISA*, o tempo médio que o programa do REGRA_C leva para ser executado com a ajuda do sistema criptográfico é encontrado da seguinte maneira: o dobro do tempo médio gasto por uma operação $Q = kP$ (vide Tabela 6.1) mais oito vezes o tempo de acesso à placa (vide Tabela 5.19), ou seja, (2 * 100,050 μs) + (8 * 5,5 μs), que é igual a 244,100 μs.

Entretanto, quando há um uso ininterrupto da placa para atender a trocas de chaves freqüentes, o tempo de acesso à mesma, através do barramento *ISA*, torna-se desprezível, conforme explicado na seção 5.6. Para os demais barramentos, o tempo de acesso à placa do sistema criptográfico pode ser considerado sempre desprezível (vide Tabela 5.1).

Para o mesmo exemplo acima, nota-se que a execução do programa que implementa o modelo Diffie-Hellman fica, aproximadamente, 20.483 vezes mais rápida, quando tem a ajuda do *hardware* do sistema criptográfico.

6.6 Comentários Finais

Os resultados finais deste trabalho foram apresentados neste capítulo. Eles foram obtidos a partir das compilações e das simulações do sistema criptográfico feitas através da ferramenta Quartus II da Altera.

As compilações e as simulações revelam resultados relacionados tanto à velocidade quanto à área dos circuitos que constituem este sistema criptográfico.

No que diz respeito aos valores relativos à velocidade dos circuitos, os resultados permitem conhecer o tempo de processamento consumido em cada uma das partes do sistema.

Quanto aos dados ligados à área dos circuitos, os resultados possibilitam conhecer as dimensões do sistema.

Eles também permitem estabelecer limites físicos para a implantação do sistema em placas adaptadoras, uma vez que as *FPGAs* utilizadas neste trabalho possuem tamanhos bem definidos.

Os resultados das comparações entre as implementações em *software* e em *hardware* para a execução do modelo Diffie-Hellman de troca de chaves, revelam as diferenças de desempenho entre as duas abordagens.

Concluindo, um melhor desempenho é alcançado, quando o programa que implementa o modelo Diffie-Hellman recebe o auxílio do *hardware* do sistema criptográfico durante a sua execução. Aliás, este sistema criptográfico apresenta o melhor desempenho dentre todas as implementações citadas neste capítulo.

Capítulo 7
Considerações Finais

7.1 Comentários

O trabalho descreveu o desenvolvimento de um sistema criptográfico para curvas elípticas sobre corpos finitos binários.

Sua lógica foi carregada em dois circuitos lógico-programáveis, implementados em uma placa adaptadora para *PC*, para exemplificação.

Foram apresentados os resultados provenientes da compilação e da simulação do projeto para os quatro corpos finitos distintos $GF(2^{113})$, $GF(2^{131})$, $GF(2^{147})$ e $GF(2^{163})$.

Os testes práticos do sistema criptográfico foram baseados em curvas elípticas e pontos definidos em conformidade com os padrões *NIST, IEEE P1363, IPSec, WAP, eCheck, ANSI X9.62* e *ANSI X9.63*, sobretudo para $GF(2^{163})$.

Os comentários foram feitos com base na comparação dos resultados obtidos através desses testes com resultados de testes provenientes de implementações tanto em *software* como em *hardware* realizadas em outros trabalhos.

Em outro tipo de simulação, os resultados indicam que a execução do programa que implementa o modelo Diffie-Hellman de troca de chaves fica, aproximadamente, 20.483 vezes mais rápida, com a ajuda do *hardware* deste sistema criptográfico, quando comparada com a nossa implementação em *software*.

O objetivo principal foi mostrar a viabilidade do uso predominante de lógica combinacional para implementar sistemas criptográficos dessa natureza e dessa dimensão em circuitos programáveis.

O sistema criptográfico proposto tem como objetivo principal garantir uma alta freqüência de troca de chaves, a fim de inibir a *criptoanálise*, por exemplo, em um sistema cliente-servidor.

7.2 Conclusões

Este trabalho teve um resultado bastante positivo, tanto em termos de realização de pesquisa, como em termos de desenvolvimento de projetos. Representa uma fonte complementar de pesquisa para aqueles que visam aprofundar-se no estudo da criptografia, sobretudo para aqueles que têm a intenção de trabalhar com projetos de *hardware*, voltados a ambientes onde se faz necessário empregar sistemas de garantia de segurança para as suas informações.

Conclui-se que é possível implementar o sistema criptográfico para curvas elípticas sobre corpos finitos binários proposto. Além disso, os resultados das simulações comprovam que o sistema funciona de forma adequada.

O sistema criptográfico apresenta um alto desempenho e considerável ocupação de área, devido a quatro fatores principais:

- desenvolvimento predominante de circuitos combinacionais de grande velocidade e densidade;
- os parâmetros de compilação do projeto do sistema criptográfico foram escolhidos de forma a priorizar um maior desempenho do sistema, em detrimento a uma menor área;
- as *FPGAs* escolhidas para implementar o sistema criptográfico pertencem à família Stratix II, com os dispositivos mais densos da Altera;
- grande parte dos circuitos que compõem o sistema criptográfico foi projetada usando Mega-Funções, as quais são códigos *VHDL* otimizados para implementações em *FPGAs* da Altera. Elas contribuem significativamente para o alto desempenho de sistema.

Ainda com relação à questão de desempenho, os resultados obtidos permitem concluir que uma maior velocidade de processamento pode ser alcançada através de circuitos exclusivamente combinacionais.

Portanto, também é demonstrado que, se a alta velocidade representar uma questão crucial em um determinado projeto, este deve ser projetado preferencialmente de forma combinacional, independentemente da área consumida.

Ele também deverá ser descrito através das Mega-Funções, visto que elas contribuem para um melhor desempenho dos circuitos implementados em *FPGAs*.

É possível implementar um sistema criptográfico para curvas elípticas, projetado predominantemente através de circuitos combinacionais, em duas *FPGAs* da Altera, conforme foi explicado no Capítulo 5.

A tendência atual das pesquisas sobre criptografia implementada em *hardware* é propor sistemas, que trabalham com pontos de curvas elípticas representados através de coordenadas projetivas e base normal, implementados com circuitos seqüenciais.

Essas características permitem o desenvolvimento de circuitos menos densos, ideais para atender à crescente demanda por circuitos que caibam em ambientes restritos, como por exemplo, sistemas embarcados.

Entretanto, por ocuparem uma área menor, tais circuitos penalizam seu desempenho. Apesar das vantagens citadas, essa penalidade se opõe ao desafio de focar o alto desempenho em detrimento à área, proposto inicialmente.

Por isso, este trabalho em nenhum momento objetivou se prender à tendência atual dos pesquisadores em seguir as características consideradas mais ideais para a implementação de sistemas criptográficos em *hardware*.

O sistema criptográfico proposto trabalha com coordenadas afins e base polinomial. Essas características não são as mais ideais para implementações em *hardware*, por gerarem circuitos mais complexos e conseqüentemente mais densos.

Mesmo assim os resultados deste trabalho demonstram que a área deste sistema não está muito acima da capacidade atual das *FPGAs*. Em outras palavras, é provável que, muito em breve, seja lançada no mercado uma *FPGA* capaz de abrigar este sistema por

completo em um único *chip*, melhorando, assim, as suas condições em relação a ambientes restritos, embora isso não represente os objetivos deste projeto.

Os resultados também demonstram que mesmo sem as características ideais para a implementação em *hardware*, este sistema criptográfico apresenta benefícios à criptografia.

7.3 Sugestões para Trabalhos Futuros

Com isso em mente, pode-se sugerir como trabalho futuro a implementação de um novo sistema como este, porém que atenda mais diretamente às tendências atuais de pesquisa em criptografia voltada para *hardware*.

Em outras palavras, o sistema pode trabalhar com pontos de curvas elípticas representados através de coordenadas projetivas e base normal, não mais sendo implementado através de circuitos seqüenciais mas sim combinacionais.

Essa proposta pode trazer benefícios ainda mais significativos à criptografia, não somente em termos de velocidade como também em termos de área.

Outras sugestões para trabalhos futuros são:

- Implementar o circuito combinacional necessário para o cálculo das demais operações presentes nas equações de (2.14) a (2.16), também através das Mega-Funções do programa Quartus II, como foi feito com o circuito combinacional para a divisão modular;
- Usar outros algoritmos criptográficos, como aqueles descritos em [G98] e [GO98], na placa adaptadora para *PC*, em substituição ao *Double-and-Add*, e estabelecer comparações entre os resultados obtidos;
- Implementar, através de *ASICs*, o sistema criptográfico apresentado;
- Implementar fisicamente em uma placa *PCI* o sistema criptográfico proposto e simulado;
- Registrar os resultados alcançados com uma implementação física deste sistema e compará-los com os resultados das simulações apresentados;

- Construir um sistema criptográfico completo, composto por circuitos que implementam a criptografia assimétrica (como é o caso deste trabalho) e trabalham em conjunto com circuitos que implementam também a criptografia simétrica.

7.4 Principal Contribuição

Nós contribuímos com um trabalho atual, em uma área muito ativa, de grande interesse teórico e prático.

Referências Bibliográficas

[AC99] Altera Corporation. "Quartus II, Programmable Logic Development System & Software". *Data Sheet*, ver. 1.01, 1999.

[ACPR04] B. J. Abcunas, S. P. Coughlin, G. T. Pedro and D. C. Reisberg. "Evaluation Random Number Generators on FPGAs", *Worcester Polytechnic Institute*, BS2-0301, 2004.

[AMV93] G. B. Agnew, R. C. Mullin and S. A. Vanstone. "An Implementation of Elliptic Curve Cryptosystems over $F_{2^{155}}$", *IEEE Journal on Selected Areas in Communications*, 11(5):804-813, 1993.

[BSN99] I. Blake, G. Seroussi and S. Nigel. "Elliptic Curves in Cryptography", *Cambridge University Press*, 1999.

[DO03] M. A. Dias e J. R. Oliveira. "Elliptic Curve Algorithms: An Analysis of Cost Between Operations of the Scalar Multiplication". *Accepted for Publication in the Proceedings of the Technical Track of the ACNS'03, ICISA Press*, Kunming, China, mauricioaraujodias@hotmail.com, October 2003.

[DO1-07] M. A. Dias e J. R. Oliveira. "An Inverter Architecture for ECC-GF(2^m) Based on the Stein's Algorithm". 3rd *International Workshop on Boolean Functions Cryptography and Applications (BFCA'07)*, Paris, France, May 2007.

[DO2-07] M. A. Dias e J. R. Oliveira. "Speeding Up Point Doublings and Point Additions for ECC-GF(2^m)". *Relatório Técnico DCA*, mauricioaraujodias@hotmail.com, 2007.

[DQ05] G. M. Dormale and J-J. Quisquater.

"Novel Iterative Digit-Serial Modular Division over GF(2^m)", *http://www.dice.ucl.ac.be/crypto/files/publications/pdf254.pdf*, 2005.

[EGSG03] H. Eberle, N. Gura, S. C. Shantz and V. Gupta. "A Cryptographic Processor for Arbitrary Elliptic Curves over GF(2^m)". *Sun Microsystems Laboratories*, SMLI TR-2003-123, May 2003.

[EKHH01] M. Ernst, S. Klupsch, O. Hauck and S. A. Huss. "Rapid Prototyping for Hardware Accelerated Elliptic Curve Public-Key Cryptosystems". *12th IEEE Workshop on Rapid System Prototyping*, Monterey, CA, June 2001.

[G98] D. M. Gordon. "A Survey of Fast Exponentiation Methods". *Journal of algorithms 27*, pp. 129-146, article nº. AL970913, 1998.

[GC01] J. Goodman and A. P. Chandrakasan. "An Energy-Efficient Reconfigurable Public-Key Cryptography Processor". *IEEE J. Solid-State Circuits*, vol. 36, pp. 1808-1820, Nov. 2001.

[GO98] P. Guillot and O. Orcière. "Speeding up Elliptic Curve Computations Using Addition-Substraction Chains and Horner Rule". *Thomson-CSF Communications*, March 4, 1998.

[GSEGGFGS02] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy and D. Stebila. "An End-to-End Systems Approach to Elliptic Curve Cryptography". *CHES 2002*, LNCS 2523, pp. 349-365, 2002.

[GSS99] L. Gao, S. Shrivastava and G. E. Sobelman. "Elliptic Curve Scalar Multiplier Design Using FPGAs", *Department of Electrical & Computer Engineering, University of Minnesota*, Twin-Cities, USA, 1999.

[GVNG94] D. D. Gajski, F. Vahid, S. Narayan and J. Gong. "Specification and Design of Embedded Systems", *Prentice-Hall*, 1st ed., 1994.

[HB92] M. A. Hasan and V. K. Bhargava. "Bit-Serial Systolic Divider and Multiplier for Finite Fields GF(2^m)". *IEEE Transaction on Computers*, vol. 41, no. 8, pp. 972-980, 1992.

[HMV04] D. Hankerson, A. Menezes and S. Vanstone. "Guide to Elliptic Curve Cryptography", *Springer Professional Computing*, 2004.

[IEEE99] IEEE P1363/D13 (Draft Version 13). "Standard Specification for Public Key Cryptography". *IEEE Inc.*, 1999.

[IT88] T. Itoh and S. Tsujii. "A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases". *Information and Computation*, vol. 78, pp. 171-177, 1988.

[LD99] J. López and R. Dahab. "Fast Multiplication on Elliptic Curves over GF(2^m) without Precomputation". *Workshop on Cryptographic Hardware and Embedded Systems* (*CHES) 1999*, LNCS 1717, pp. 316-327, 1999.

[LN94] R. Lidl and H. Niederreiter. "Introduction to Finite Fields and their Applications". *Cambridge University Press*, Cambridge, UK, revised edition, 1994.

[M02] P. Martin. "An Analysis of Random Number Generators for a Hardware Implementation of Genetic Programming using FPGAs and Handel-C", *Department of Computer Science, University of Essex*, UK, CSM-358, 2002.

[M93] A. J. Menezes, "Elliptic Curve Public Key Cryptosystems", *Kluwer Academic Publishers*, 1993.

[MAH01] R. G. Macedo, A. J. de Almeida Jr e M. A. A. Henriques. “Criptografia em Ambientes Computacionais Restritos: Implementação de Curvas Elípticas em Software Estruturado”, *DCA/FEEC/UNICAMP*, REGRA-C, 2001.

[MO89] F. Morain and J. Olivos. “Speeding up the Computations on an Elliptic Curve Using Addition-Subtraction Chains”. *Rapport INRIA*, issue 983, March 1989.

[OP00] G. Orlando and C. Paar. “A High-Performance Reconfigurable Elliptic Curve Processor for GF (2^m)”. *Workshop on Cryptographic Hardware and Embedded Systems* (*CHES) 2000*, LNCS 1965, pp. 41-56, 2000.

[P99] D. Perry. “VHDL”, *McGraw-Hill*, 3rd ed., 1999.

[R98] M. Rosner. “Elliptic Curve Cryptosystems on Reconfigurable Hardware”, *Master’s thesis, ECE Dept., Worcester Polytechnic Institute,* Worcester, USA, 1998.

[S01] S. C. Shantz. “From Euclid's GCD to Montgomery Multiplication to the Great Divide”. *Sun Microsystems Labs*, SMLI TR-2001-95, 2001.

[S67] J. Stein. “Computational problems associated with Racah algebra”. *J. Computational Physics*, vol. 1, pp. 397--405, 1967.

[S98] W. Stallings. “Network and Internetwork Security”, *Prentice-Hall*, 2nd ed., 1998.

[SEC1-00] Certicom Research. “SEC 1: Elliptic Curve Cryptography”, *Standards for Efficient Cryptography*, Version 1.0, www.certicom.com, 2000.

[SEC2-00] Certicom Research. “SEC 2: Recommended Elliptic Curve Domain Parameters”, *Standards for Efficient Cryptography*, Version 1.0, www.certicom.com, 2000.

[SES98] S. Sutikno, R. Effendi and A. Surya. “Design and Implementation of Arithmetic Processor $F_{2^{155}}$ for Elliptic Curve Cryptosystems”, *The 1998 IEEE Asia-Pacific Conference on Circuits and Systems*, pages 647-650, 1998.

[SOOS95] R. Schroeppel, H. Orman, S. O’Malley and O. Spatscheck. “Fast Key Exchange with Elliptic Curve Systems”, *In D. Coppersmith*, *editor*, *Advances in Cryptography, Crypto 95*, volume LNCS 963, Springer-Verlag, 1995.

[T96] A. S. Tanenbaum. “Computer Networks”, *Prentice-Hall*, 3rd ed., 1996.

[T98] N. Takagi. “A VLSI Algorithm for Modular Division Based on the Binary GCD Algorithm”. *IEICE Trans. Fundamentals*, vol. E81-A, pp. 724-728, May 1998.

[TOIT00] N. Torii, S. Okada, K. Itoh and M. Takenaka. “Implementation of Elliptic Curve Cryptographic Coprocessor over GF(2^m) on an *FPGA*”, *Fujitsu Laboratories Ltd.*, 2000.

[VCC00] Virtual Computer Corporation. “Field Programmable Gate Arrays (*FPGAs*) – An Enabling Technology”, *Virtual Computer Corp.*, 2000.

[WBVGV96] E. D. Win, A. Bosselaers, S. Vandenberghe, P. D. Gersem and J. Vandewalle. “A Fast Software Implementation for Arithmetic Operations in GF(2^n)”. *Advances in Cryptology - ASIACRYPT'96*, Lecture Notes in Computer Science 1163, pp.65-76, Springer-Verlag, 1996.

[WT00] Y. Watanabe and N. Takagi. “A VLSI Algorithm for Division on GF(2^m) Based on Binary Method”. *Proc. 2000 Eng. Sciences Soc. Conf. IEICE*, A-3-15, p. 82, Sept. 2000.

[WTT02] Y. Watanabe, N. Takagi, and K. Takagi. “A VLSI Algorithm for

Division in GF(2^m) Based on Extended Binary GCD Algorithm" *IEICE Trans. Fundamentals*, vol. E85-A, pp. 994-999, May 2002.

[WWSH04] C. H. Wu, C. M. Wu, M. D. Shieh and Y. T. Hwang. "High-Speed, Low-Complexity Systolic Designs of Novel Iterative Division Algorithms in GF(2^m)". *IEEE Trans. on Computers*, vol. 53, no. 3, pp. 375-380, 2004.

Apêndice A
Esquema Elétrico Completo de uma Fatia Usada no Circuito Combinacional para a Inversão Modular

Este apêndice apresenta o esquema elétrico completo de uma das *2m-1* fatias usadas no circuito combinacional para a inversão modular, descrito no Capítulo 3. Esse esquema elétrico aparece representado através da figura A.2, na próxima seção. Entretanto, para entender melhor o esquema elétrico, antes, é interessante fazer um breve comentário a respeito da figura A.1, que mostra a representação gráfica do conjunto de fatias do circuito para a inversão modular.

Através da figura A.1, é possível notar que as entradas da primeira fatia recebem os seguintes valores de início: *(Ain, Bin, Uin, Vin, DCCin, FLAGin)* ← *(P'x, p, 1, 0, 2, 1)*. *P'x* é o valor submetido à inversão modular. Se *k = 0*, *Ain = P'x*. Se *k = 1*, *Ain = P'x xor P*. O *p* é um polinômio redutor. Os demais valores são constantes. A saída *Vout* da última fatia apresenta o valor relativo ao inverso modular de *P'x*, aqui chamado de *S*.

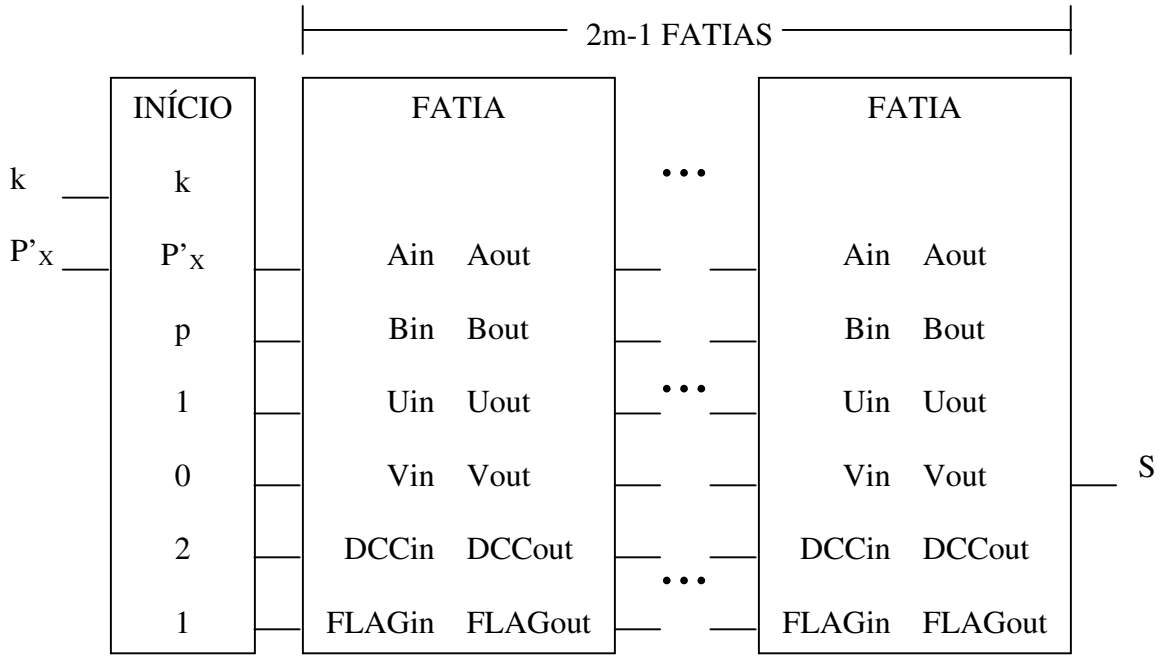

Figura A.1: Representação gráfica das fatias do circuito para a inversão modular.

A.1 Esquema Elétrico Completo de uma Fatia

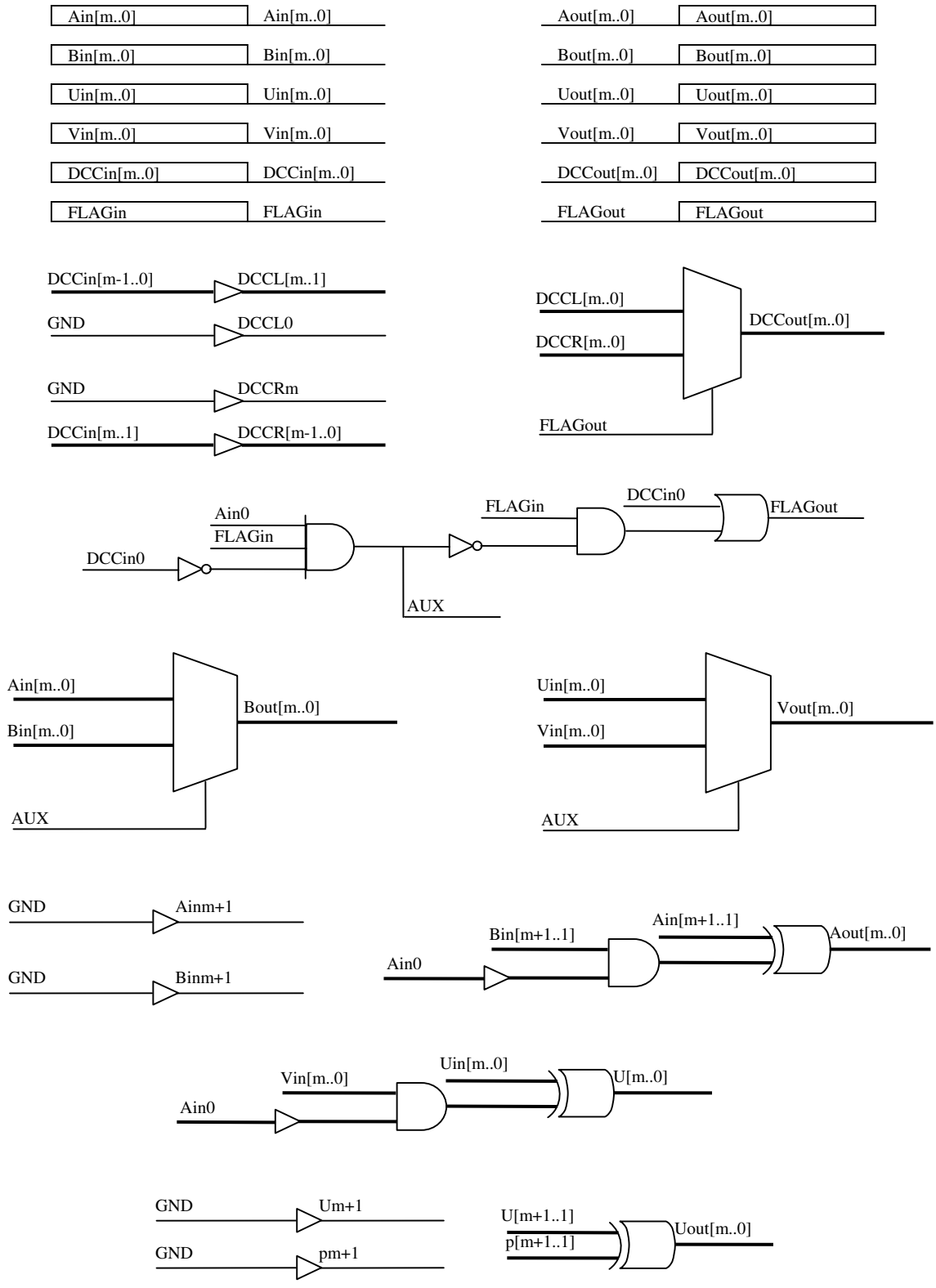

Figura A.2: Esquema elétrico completo de uma fatia do circuito para a inversão modular.

A.2 Comentários

Os elementos que constituem o esquema elétrico da fatia apresentado na seção A.1 são representações gráficas das Mega-Funções do Quartus II. Todas as fatias são iguais. A união em série de *2m-1* fatias como esta forma o circuito combinacional para a inversão modular, para o qual *m* representa o corpo finito.

Com base nas informações contidas neste apêndice, é possível usar o Quartus II para criar, compilar e simular o circuito combinacional para a inversão modular.

Apêndice B
Código *VHDL* completo que Descreve o Circuito Combinacional para as Demais Operações

Este apêndice apresenta o código *VHDL* completo que descreve o circuito combinacional usado para realizar as demais operações, menos a inversão modular, presentes nas equações de (2.14) a (2.16) (vide Capítulo 2).

Para esse circuito, a entrada *Sin* recebe o valor apresentado na saída *Vout* do circuito combinacional para a inversão modular. Além disso, as entradas *Pxin* e *Pyin* recebem as coordenadas do ponto *P'*. As saídas *Qxout* e *Qyout* apresentam as coordenadas do novo ponto *Q* calculado. Considera-se aqui $GF(2^{163})$.

Quando estão juntos, os circuitos combinacionais para a inversão modular e para as demais operações constituem o circuito combinacional para a Duplicação e para a Adição de Pontos, descrito no Capítulo 4.

O código VHDL em questão é apresentado pela seção B.1, a seguir.

B.1 Código Completo

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY demais164b IS

      PORT(
            k               : boolean;
            Pxin, Pyin, Sin : IN STD_LOGIC_vector( 163 DOWNTO 0 );
            Qxout, Qyout    : BUFFER STD_LOGIC_vector( 163 DOWNTO 0 )
            );

END demais164b;
```

```
ARCHITECTURE demais164bBHV OF demais164b IS


BEGIN
-- Processo demais164b

      demais164bp : process( k, Pxin, Pyin, Sin )

                  VARIABLE Px, Py   : STD_LOGIC_vector( 163 DOWNTO 0 );
                  VARIABLE Qx, Qy   : STD_LOGIC_vector( 163 DOWNTO 0 );
                  VARIABLE temp, S  : STD_LOGIC_vector( 163 DOWNTO 0 );
                  VARIABLE dbltemp  : STD_LOGIC_vector( 326 DOWNTO 0 );
                  CONSTANT a        : STD_LOGIC_vector( 163 DOWNTO 0 ) :=
                  X"0000000000000000000000000000000000000001";
                  CONSTANT p        : STD_LOGIC_vector( 163 DOWNTO 0 ) :=
                  X"800000000000000000000000000000000000000C9";



      BEGIN

-- Inicio
               CASE k is

                  WHEN false =>
                       Px     := Pxin;
                       Py     := Pyin;
                       temp := Pyin;

                  WHEN true =>
                       Px     :=
                       X"3F0EBA16286A2D57EA0991168D4994637E8343E36";
                       Py     :=
                       X"0D51FBC6C71A0094FA2CDD545B11C5C0C797324F1";
                       temp := Pyin XOR Py;
               END CASE;

                  S := Sin;
                  dbltemp :=
                  "000000000000000000000000000000000000000000000000000000000
                  000000000000000000000000000000000000000000000000000000000
                  000000000000000000000000000000000000000000000000000000000
                  000000000000000000000000000000000000000000000000000000000
                  000000000000000000000000000000000000000000000000000000000
                  000000000000000000000000000000000000000000000000000000000";



-- Multiplicador
                  FOR i IN 0 TO 163 LOOP
                        IF temp( i ) = '1' THEN
                              dbltemp((163+i) DOWNTO i) :=
                              S XOR dbltemp((163+i) DOWNTO i);
                        END IF;
                  END LOOP;
```

```
-- Redutor
                FOR i IN 0 TO 163 LOOP
                      IF dbltemp( (326-i) ) = '1' THEN
                            dbltemp((326-i) DOWNTO (163-i)) :=
                            p XOR dbltemp((326-i) DOWNTO (163-i));
                      END IF;
                END LOOP;

                S := dbltemp( 163 DOWNTO 0 );
                dbltemp :=
                "00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000";



-- Somador
                IF NOT(k) THEN
                      S := S XOR Px;
                END IF;



-- Squaring
                FOR i IN 0 TO 163 LOOP
                      dbltemp(2*i) := S(i);
                END LOOP;



-- Redutor
                FOR i IN 0 TO 163 LOOP
                      IF dbltemp( (326-i) ) = '1' THEN
                            dbltemp((326-i) DOWNTO (163-i)) :=
                            p XOR dbltemp((326-i) DOWNTO (163-i));
                      END IF;
                END LOOP;

                temp  := dbltemp( 163 DOWNTO 0 );
                dbltemp :=
                "00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000
                00000000000000000000000000000000000000000000000000000000";



-- Somador
                Qx := temp XOR S XOR Px XOR Pxin XOR a;
                temp := Px XOR Qx;
```

```
-- Multiplicador
                  FOR i IN 0 TO 163 LOOP
                        IF temp( i ) = '1' THEN

                              dbltemp((163+i) DOWNTO i) :=
                              S XOR dbltemp((163+i) DOWNTO i);
                        END IF;
                  END LOOP;



-- Redutor
                  FOR i IN 0 TO 163 LOOP
                        IF dbltemp( (326-i) ) = '1' THEN
                              dbltemp((326-i) DOWNTO (163-i)) :=
                              p XOR dbltemp((326-i) DOWNTO (163-i));
                        END IF;
                  END LOOP;



-- Somador
                  Qy := dbltemp( 163 DOWNTO 0 ) XOR Py XOR Qx;



-- Saída
                  IF (((Pxin = Px) AND k) OR (Px =
                  X"00000000000000000000000000000000000000000000")) THEN
                        Qx :=
                        X"00000000000000000000000000000000000000000000";
                        Qy :=
                        X"00000000000000000000000000000000000000000000";
                  END IF;

                  IF (Pxin =
                  X"00000000000000000000000000000000000000000000") THEN
                        Qx := Px;
                        Qy := Py;
                  END IF;

                  IF (((Pxin =
                  X"3F0EBA16286A2D57EA0991168D4994637E8343E36") AND (Pyin
                  = X"0D51FBC6C71A0094FA2CDD545B11C5C0C797324F1")) AND k)
                  THEN
                        Qx :=
                        X"1AEB33FED9C49E0200A0C561EA66D5AB85BD4C2D4";
                        Qy :=
                        X"530608192CD47D0C24C20076475FD625CC82895E8";
                  END IF;

                  Qxout <= Qx;
                  Qyout <= Qy;

      end process demais164bp;
END demais164bBHV;
```

B.2 Comentários

Este apêndice mostra o código *VHDL* que descreve o circuito combinacional para as demais operações de forma integral. Em outras palavras, se ele for transcrito para o Quartus II, exatamente da maneira como se apresenta, ele pode ser compilado e simulado.