Multiplicador em Corpo Finito Utilizando Redes Neurais Discretas.

Lidiano Augusto Nóbrega de Oliveira

Dissertação de Mestrado submetida à Coordenação dos Cursos de Pós-Graduação em Engenharia Elétrica da Universidade Federal da Paraíba - Campus II como parte dos requisitos necessários para obtenção do grau de Mestre no domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação - Comunicações

Francisco Marcos de Assis
Orientador

Campina Grande, Paraíba, Brasil

O48m Oliveira, Lidiano Augusto Nobrega de
Multiplicador em corpo finito utilizando redes neurais discretas / Lidiano Augusto Nobrega de Oliveira. - Campina Grande, 2000.
84 f.

Dissertacao (Mestrado em Engenharia Eletrica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Corpos Finitos 2. Redes Neurais Discretas 3. Dissertacao - Engenharia Eletrica I. Assis, Francisco Marcos de II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 512.624(043)

MULTIPLICADOR EM CORPO FINITO UTILIZANDO REDES NEURAIS DISCRETAS

LIDIANO AUGUSTO NÓBREGA DE OLIVEIRA

Dissertação Aprovada em 17.11.2000

PROF. FRANCISCO MARCOS DE ASSIS, Dr., UFPB
Orientador

PROF. ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFPB
Componente da Banca

PROF. ELMAR UWE KURT MELCHER, Dr., UFPB
Componente da Banca

PROF. RAIMUNDO CARLOS SILVÉRIO FREIRE, Dr., UFPB
Componente da Banca

PROF. RICARDO MENEZES CAMPELLO DE SOUZA, Ph.D., UFPE
Componente da Banca

CAMPINA GRANDE - PB
Novembro - 2000

Dedicatória

Esta dissertação é dedicada aos meus pais e a minha irmã.

Agradecimentos

- A Deus, por tudo;
- Aos meus pais, Ari e Nevinha, pelo amor, compreensão e incentivo em toda minha vida;
- A minha irmã, Lisiane, pelo incentivo e carinho;
- Ao Prof. Francisco Marcos de Assis, pela orientação, incentivo e amizade;
- Aos Professores Marcelo Sampaio, Bruno Albert e João Marques, pelos ensinamentos e amizade;
- Aos demais professores do DEE-UFPB, pelos ensinamentos e incentivo;
- Aos colegas da pós-graduação Gustavo, Waslon, Mohit, Suzete, Izabel, Alexandre e Wamberto pela amizade e companheirismo e aos demais colegas que contribuiram direta ou indiretamente para a realização deste trabalho;
- Aos funcionários do DEE;
- A Coordenação do Curso de Pós-Graduação em Eng. Elétrica da UFPB;
- Ao CNPq que financiou este trabalho;

Resumo

Cálculo em corpos finitos são amplamente utilizados em códigos corretores de erros, processamento digital de sinal, geração de números pseudoaleatórios e esquemas de criptografia. Geralmente essas aplicações modernas necessitam de implementações que satisfaçam as exigências de alta velocidade. Assumindo a representação em base polinomial dos elementos do corpo, adição é simples de implementar enquanto a multiplicação paralela rápida necessita de uma estrutura mais complexa. Outras operações aritméticas importantes dos corpos finitos, tais como exponenciação e divisão, podem ser realizadas através de repetidas multiplicações.

Consequentemente, multiplicadores eficientes são desejados já que a maioria das operações aritméticas avançadas são baseadas na multiplicação. As redes neurais discretas implementadas com portas de limiar linear permitem reduzir a complexidade de certos circuitos antes implementado com lógica tradicional (portas AND, OR e NOT). Muitas pesquisas têm sido desenvolvidas com relação a aplicação das portas de limiar linear em operações aritméticas básicas (adição, multiplicação e divisão). Essas operações podem ser implementadas com portas de limiar linear utilizando um baixo número de portas e retardo fixo. A idéia de estender o uso de portas de limiar linear em operações aritméticas básicas para aritmética em corpo finito é proposta neste trabalho. Esta dissertação apresenta uma arquitetura de um multiplicador paralelo em $GF(2^n)$ baseado no multiplicador de Mastrovito com portas de limiar linear. Também uma nova arquitetura é proposta, mais eficiente em termos da complexidade temporal e espacial e mantendo uma baixa complexidade espacial.

Abstract

Finite field computations are widely used in error correcting codes, digital signal processing, pseudorandom number generation and cryptographic schemes. These modern applications in many cases require specific implementation in order to satisfy the high speed requirements. Assuming a polynomial basis representation of the field elements, addition is easy to implement whereas fast bit-parallel multiplication requires a more complex structure. Other important arithmetic operations in finite fields, such as exponentiation and division, can be performed by applying multiplication operations repeatedly. Consequently, efficient multipliers are desired since most advanced arithmetic functions are based on multiplication. The discrete neural networks built with threshold gates can reduce the complexity of a number of circuits once built using traditional boolean gates (AND, OR and NOT). Many researchs have been developed about the application of threshold gates in basic arithmetic operations (addition, multiplication and division). These operations can be built using threshold gates with a low number of gates e fixed delay. The idea of extending the advantages of threshold gates from basic arithmetic operations to finite fields arithmetic is proposed. This work presents the architecture of a bit-parallel multiplier over $GF(2^n)$ based on the Mastrovito multiplier using threshold gates. In addition, a novel architecture is proposed, which is more efficient concerning time and space complexity.

Lista de Figuras

Lista de Tabelas

Conteúdo

Capítulo 1

Introdução

Na era da informação, dados são recebidos, processados e transmitidos a velocidades bastante elevadas. Na transmissão e armazenamento digital de dados, há sempre a possibilidade de ocorrência de erros. Mas, segundo a teoria de Shannon, é possível inserir redundâncias no sinal antes da transmissão ou armazenamento de modo que seja possível a detecção e/ou correção de erros. Essas redundâncias são incorporadas aos dados seguindo especificações definidas pelo código. O uso de códigos corretores de erros é uma constante nos meios de comunicação. Códigos como BCH, Goppa e RS são utilizados em aplicações que variam de CD a transmissão por satélite. Muitas vezes é necessário criptografar esses dados para que possam ser transmitidos com segurança.

Como ferramenta matemática na construção de códigos e de sistemas de criptografia, foi desenvolvido por *Évariste Galois* um sistema algébrico chamado de corpo finito, ou campo de Galois (*Galois Field*). Um corpo é essencialmente um conjunto de elementos no qual é possível adicionar, subtrair, multiplicar e dividir elementos no corpo e sempre obter um elemento que pertence ao corpo. Dentre as várias operações em um corpo finito, a multiplicação é um dos blocos funcionais básicos para muitas aplicações como criptografia, geradores de sequências pseudoaleatórias, código corretores de erros e processamento digital de sinais [1–4]. Devido a grande utilização e a sua complexidade elevada, a multiplicação em corpos finitos é uma operação chave.

O projeto de multiplicadores eficientes melhora sensivelmente o desempenho dos sistemas que os utilizam. Assim, para se obter aplicações cuja velocidade

atendam as necessidades, é inevitável a utilização de circuitos que executem essa aritmética de forma mais eficiente. Usualmente utiliza-se *hardware* específico para esse cálculo devido a sua complexidade elevada [5, 6]. Na prática, corpos finitos de característica 2 com elementos em $GF(2^n)$ são utilizados em razão da facilidade de mapeamento dos elementos do corpo em bits. Muitas abordagens e arquiteturas têm sido propostas para implementar a multiplicação em corpos finitos de forma mais eficiente [7–9]. Diferentes representações de base, como canônicas, duais e normais, têm sido utilizadas para se obter algumas arquiteturas satisfatórias [10].

Os multiplicadores em um corpo finito são implementados usualmente com portas AND, OR e NOT, chamadas de portas AON ou tradicionais. Muitos sistemas são propostos tendo em vista a obtenção de circuitos mais eficazes [11–13]. A maioria desses sistemas estão baseados na arquitetura do multiplicador proposto por Mastrovito [14], que é um dos multiplicadores mais eficientes. Pouco tem sido feito relativo a diminuição de complexidade nas arquiteturas que computam aritmética em corpos finitos nos últimos anos. Uma possível solução é a utilização de arquiteturas com um potencial maior que as arquiteturas que utilizam portas AON. Dentre diversas arquiteturas, as que utilizam rede neurais vem se destacando em diversas aplicações modernas [12, 13, 15].

Na maioria dos modelos de redes neurais, a unidade de processamento básico é uma porta que calcula uma função de limiar linear ou um elemento analógico que executa uma função sigmoidal. Para trabalhar com sinais digitais são utilizados elementos de limiar linear (ELL), que serão objetos de estudo para implementação de redes neurais discretas, incluindo a estimação de sua capacidade, limitações computacionais e sua comparação com outros circuitos lógicos booleanos, como as portas AON . Muitas pesquisas têm usado portas com *fan-in ilimitado* [16, 17] para implementar circuitos que executam a aritmética tradicional (soma, multiplicação, divisão,etc.) utilizando redes neurais discretas. Esses circuitos apresentam um número menor de portas do que o seu análogo de lógica AON [18].

A pesquisa sobre elementos de limiar linear tem caminhado por duas direções diferentes: teoria e implementação. A implementação em circuitos eletrônicos vem sendo proposta desde 1960. Pesquisas nesse campo ainda estão ativas hoje.

Por outro lado, os mais recentes resultados de pesquisas teóricas relacionadas aos ELL têm sido conduzidos no campo da complexidade computacional de circuitos [7, 11]. Nesta dissertação será focalizado o estudo teórico dessa complexidade, priorizando uma baixa complexidade temporal e espacial e considerando *fan-in* ilimitado.

Neste trabalho é apresentado uma arquitetura de multiplicador paralelo em corpo finito utilizando redes neurais discretas, visando circuitos mais rápidos e com um menor número de portas. Resultados obtidos [8, 19, 20] comprovam a vantagem da utilização de redes neurais discretas ao invés da lógica tradicional com o número de portas passando de tamanho exponencial (no caso de lógica AON) para tamanho polinomial (lógica neural).

A estrutura dessa dissertação está organizada da seguinte forma: No Capítulo 1 são apresentadas as Redes Neurais Discretas. Algumas características e propriedades relevantes são levantadas, assim como conceitos como profundidade, camada e tamanho são discutidos. Também são implementadas, a título de exemplo, algumas operações aritméticas comuns (como adição e multiplicação).

No Capítulo 2 é apresentada uma introdução sobre corpos finitos e a sua aritmética, enfatizando algumas arquiteturas mais conhecidas de multiplicadores.

No Capítulo 3 são discutidas algumas operações aritméticas em corpos finitos, focalizando o Multiplicador de Mastrovito, que é uma das arquiteturas mais otimizadas em termos de complexidade espacial e temporal.

No Capítulo 4 são apresentadas duas arquiteturas de multiplicadores propostas no trabalho. A primeira se baseia no multiplicador de Mastrovito, cuja lógica boolena é substituida pela lógica neural. Já na segunda arquitetura, o multiplicador é feito diretamente com redes neurais discretas.

No Capítulo 5 estão as conclusões e sugestões para trabalhos que, futuramente, venham a ser realizados.

Capítulo 2

Redes Neurais Discretas

2.1 Introdução

O interesse no desenvolvimento de pesquisas sobre redes neurais artificiais surgiu da convicção que o cérebro humano é bastante superior aos computadores em resolver diversos tipos de problemas, tais como reconhecimento de voz e imagens. Essas rede neurais podem ser melhor descritas com redes de alto grau de interconexão com várias unidades básicas chamadas de portas neurais, inspiradas no neurônio biológico.

A idéia de modelamento do neurônio por unidades discretas foi introduzida por McCulloch e Pitts em 1943. Eles publicaram um artigo propondo um modelo matemático do neurônio capaz de computar funções antes calculada com lógica booleana. A saída desse modelo de neurônio artifical era 1 ou 0, refletindo o estado *tudo-ou-nada* do neurônio biológico. Nesse estado, quando o total de entradas alcançasse níveis críticos, o neurônio enviaria sua saída para outros neurônios no qual ele estaria conectado. Embora não seja capaz de capturar toda complexidade de um neurônio biológico, o modelo proposto alia a simplicidade com potencial, pois organizando neurônios em uma rede é possível obter estruturas bem mais complexas. Essa teoria foi tão influente que o tipo de neurônio ficou conhecido como neurônio de McCulloch-Pitts. Algumas redes neurais modernas [21] utilizam neurônios que são essencialmente extensões do neurônio de McCulloch-Pitts.

Diferentes tipos de redes neurais têm sido propostas. Cada tipo restrige alguns

tipos de conexões. Por exemplo, pode ser especificado que se um neurônio é conectado a outro, então o segundo neurônio não pode ter outra conexão em direção ao primeiro. Assim, o modo como os neurônios interagem um com outro através dessas conexões é chamado de arquitetura da rede neural.

Um aspecto importante das redes neurais é a capacidade de aprendizagem. O mecanismo de aprendizagem pode ser visto como um mecanismo de adaptação de parâmetros, tornando possível que a rede resultante forneça o resultado esperado. A regra que governa como essas mudanças ocorrem é chamada de algoritmo de aprendizagem. Na realidade, o algoritmo de aprendizagem é quem faz o mapeamento da entrada para a saída através de alterações de parâmetros internos da rede. Para isso é necessário assumir uma arquitetura inicial para a rede de forma que seja obtido o mapeamento adequado, ou seja, o algoritmo de aprendizagem irá convergir para um resultado satisfatório. Quando não for possível o término do algoritmo de aprendizagem, o mapeamento desejado não poderá ser representado pela arquitetura utilizada. Portanto é interessante utilizar uma arquitetura adequada de forma que sempre seja possível a representação do mapeamento. Também é possível indicar uma arquitetura ótima de tal forma que para representar um certo mapeamento seja necessário um número mínimo de recursos como, por exemplo, o número de portas neurais. Ao invés de utilizar arquiteturas baseadas em neurônios que de alguma forma aprendem, é possível utilizar arquiteturas cujos neurônios já possuam seus parâmetros previamente calculados. Dessa forma, pode-se projetar diretamente redes ótimas para representar certos mapeamentos sem utilizar nenhum processo de aprendizagem.

Os parâmetros das redes neurais discretas examinadas neste trabalho serão previamente calculados e não aprendidos. A utilização de circuitos de limiar linear foi escolhido devido a tendência do seu uso nas técnicas analíticas e resultados já conhecidos na literatura. No apêndice A algumas notações e noções de complexidade temporal e espacial são apresentadas.

Neste capítulo é introduzido a arquitetura das redes neurais discretas utilizadas ao longo da dissertação. Na seção 2.2 é apresentado o elemento de limiar linear, que desempenha a função de uma porta neural ou neurônio artificial. Na seção 2.3 é mostrado como construir circuitos de limiar linear com 2 e 3 camadas. Por fim, na seção 2.4 exemplos de circuitos já conhecidos, como adição e

multiplicação, são implementados através de redes neurais discretas.

2.2 Circuito de Limiar Linear

O modelo da rede neural utilizada neste trabalho é implementado por um circuito de limiar linear (neurônio), modelado segundo o modelo de propagação direta, em que cada elemento em uma camada computa uma função de limiar que depende apenas dos valores da camada anterior. As saídas de cada neurônio são atualizadas camada por camada, sendo que a camada mais próxima da entrada é atualizada primeira. Portanto a rede neural discreta é um circuito formado por ELL organizados em camadas. A figura 2.1 ilustra uma rede neural com quatro camadas.

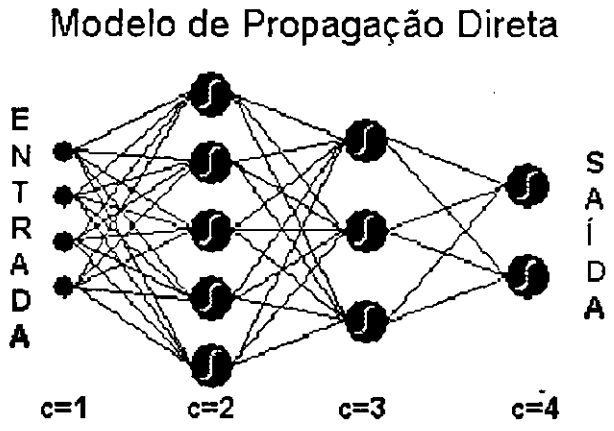

Figura 2.1: Rede Neural com Propagação Direta.

Algumas definições importantes são apresentadas a seguir [4]:

- O tamanho do circuito é o número de portas que o circuito possui, ou seja, o número de portas necessárias para implementar determinada função.
- A profundidade do circuito é o número de camadas de um circuito, que está intimamente relacionada com a velocidade de execução.
- O número de conexões que entram em um porta é chamada de *fan-in*. Similarmente, o número de conexões que saem de uma porta é chamado de

fan-out. O maior *fan-in/fan-out* de uma porta do circuito é o *fan-in/fan-out* do circuito.

Uma porta de limiar linear é um elemento básico de uma rede neural que executa a função booleana $\{0,1\}^n \rightarrow \{0,1\}$. Dado um conjunto de entradas $X = [x_1, x_2, ..., x_n]$ a saída y é determinada a seguir:

$$y = \begin{cases} 1 & \text{se } \left(\sum_{k=1}^{n} w_k x_k - t\right) \geq 0 \\ 0 & \text{se } \left(\sum_{k=1}^{n} w_k x_k - t\right) < 0 \end{cases} \tag{2.1}$$

Equivalentemente temos:

$$y = sgn\left(\sum_{k=1}^{n} w_k x_k - t\right) \tag{2.2}$$

em que $w_1, ..., w_n$ são reais e o limiar t inteiro. Embora sejam permitidos pesos w_i reais, eles podem ser representados com $n \log n$ dígitos binários [22, 23], em que n é o número de entradas . Assim no resto do trabalho considera-se, sem perda de generalidade, que todos os pesos são inteiros. A função $sgn(\cdot)$ é definida por $sgn(\alpha) = 1$ se $\alpha \geq 0$, $sgn(\alpha) = 0$ caso contrário.

Pode-se utilizar portas de limiar linear para implementar portas tradicionais, tais como portas AND,OR e NOT. Exemplos de implementações de funções tradicionais são mostradas a seguir:

Uma simples função booleana é a função OR de n variáveis dada por:

$$OR(x_1, ..., x_n) = \begin{cases} 0 & \text{se } (x_1, ..., x_n) = (0, ..., 0) \\ 1 & \text{caso contrário} \end{cases} \tag{2.3}$$

Essa função pode ser implementada utilizando uma porta de limiar linear dada por:

$$OR(x_1, ..., x_n) = sgn(\sum_{k=1}^{n} x_k - 1) \tag{2.4}$$

Observe que uma porta de limiar linear ou elemento de limiar linear é representado graficamente como mostrado na figura 2.2, em que o número no interior do círculo representa o valor do limiar t.

Outra função booleana tradicional é a função AND, definida como:

$$AND(x_1, ..., x_n) = \begin{cases} 1 & \text{se } (x_1, ..., x_n) = (1, ..., 1) \\ 0 & \text{caso contrário} \end{cases} \tag{2.5}$$

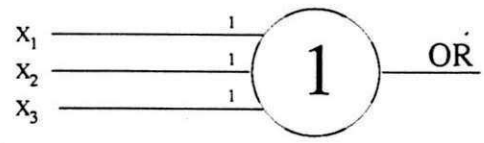

Figura 2.2: Modelo Neural da porta OR.

Essa função pode ser implementada utilizando uma porta de limiar linear dada por:

$$AND(x_1, ..., x_n) = sgn(\sum_{k=1}^{n} x_k - n) \tag{2.6}$$

A função AND pode ser representada graficamente como na figura 2.3.

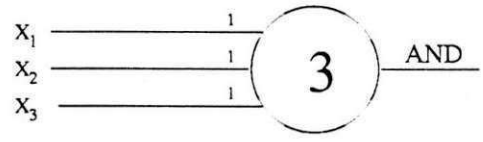

Figura 2.3: Modelo Neural da porta AND.

A porta NOT é implementada como:

$$NOT\,(x_1) = sgn(-x_1)$$

Graficamente a porta NOT pode ser representada com na figura 2.4

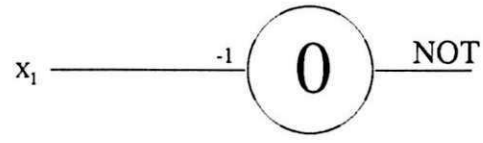

Figura 2.4: Modelo Neural da porta NOT.

A seguir são discutidas técnicas para elaboração de circuitos mais complexos que utilizem toda a potencialidade das portas de limiar linear.

2.3 Construção de Circuitos com Retardo Fixo

O projeto de circuitos de limiar linear compreende a implementação de funções booleanas. Entre tais funções, as funções booleana simétricas são particularmente interessantes para este trabalho. Os resultados apresentados estão mais detalhados em [4].

Definição 1 *Uma função booleana f é dita simétrica se sua saída depende apenas do somatório dos seus valores de entrada. Portanto, $f(x_1, x_2, \ldots, x_n) = f(x_{(1)}, x_{(2)}, \ldots, x_{(n)})$, para qualquer permutação de $(x_1, x_2, \ldots, x_n)$.*

Um exemplo de função simétrica é a função *Paridade*, cuja saída é 1 se o somatório da entrada for ímpar e 0 caso contrário. Se a função depender apenas da soma ponderada das entradas, então a chamamos de *função booleana generalizada.*

Teorema 1 *Toda função booleana $f(x_1, x_2, \ldots, x_n) : \{0,1\}^n \rightarrow \{0,1\}$ pode ser computada por um circuito de limiar com duas camadas e no máximo com $2^{n-1}+1$ elementos de limiar.*

Como qualquer função booleana pode ser escrita na forma *Soma de Produtos (SDP)* ou *Produto da Soma (PDS),* utilizando por exemplo *SDP*, uma função booleana qualquer pode ser escrita na forma:

$$f(x_1, x_2, \ldots, x_n) = S_1 \vee S_2 \vee \cdots \vee S_x, \tag{2.7}$$

em que $x \leq 2^n$ e o símbolo $\vee$ é a função AND.

Para *PDS*

$$f(x_1, x_2, \ldots, x_n) = P_1 \wedge P_2 \wedge \cdots \wedge P_y$$

em que $y \leq 2^n$.

Como $x + y = 2^n$, observe que $x \leq 2^n$ e $y \leq 2^n$ porque 2^n é o número máximo de somas ou produtos com n variáveis booleanas. Assim, a primeira camada do circuito limiar irá implementar, para SDP, os produtos S_i e a segunda camada usando a função OR irá computar f, utilizando assim $2^{n-1} + 1$ portas tradicionais [24]. Como a porta neural pode simular a porta AND e OR, é possível implementar qualquer função booleana utilizando $2^{n-1}+1$ portas neurais. Entretanto, essa construção requer o uso de circuitos cujo número de portas cresce exponencialmente com o aumento do número de entradas, pois a lógica neural apenas simula a lógica tradicional. Portanto é necessário utilizar uma nova estrutura para calcular funções simétricas eficientes, já que elas desempenham um papel importante na aritmética e funções relacionadas. Para projetar circuitos *eficientes* que executem funções simétricas são utilizadas estruturas com 2 ou 3 camadas introduzidas a seguir.

2.3.1 Circuitos de 2 camadas

Agora é apresentada uma técnica para construção de circuito de limiar linear de 2 camadas para computar qualquer função booleana simétrica f com no máximo $n+1$ portas .

Seja $X = [x_1, x_2, ..., x_n]$ a entrada booleana do circuito e f uma função booleana simétrica. Como f depende apenas do somatório das entradas, existe um conjunto de valores de $\sum_{k=1}^{n} x_k$ no qual a função f é 1. Agrupando esses valores em subintervalos em $[0, n]$ temos s subintervalos dados a seguir:

$$[q_1, \tilde{q}_1], [q_2, \tilde{q}_2], ..., [q_s, \tilde{q}_s] \tag{2.8}$$

em que q_k e $\tilde{q}_k$ são inteiros, $q_{k+1} > \tilde{q}_k + 1$ e $q_k \leq \tilde{q}_k$ de tal forma que $f(x_1, x_2, ..., x_n) = 1$ se e somente se para algum j

$$\sum_{k=1}^{n} x_k \in [q_j, \tilde{q}_j] \tag{2.9}$$

Na primeira camada do circuito, são necessárias $2s$ portas de limiar linear calculando:

$$y_{q_j} = sgn\left(\sum_{k=1}^{n} x_k - q_j\right) \tag{2.10}$$

e

$$\tilde{y}_{q_j} = sgn\left(\tilde{q}_j - \sum_{k=1}^{n} x_k\right) \tag{2.11}$$

em que $j = 1, ..., s$.

Na segunda camada, a porta neural computa

$$f(x_1, x_2, ..., x_n) = sgn\left(\sum_{k=1}^{s} \left(y_{k_j} + \tilde{y}_{k_j}\right) - s - 1\right) \tag{2.12}$$

Para comprovar se o circuito fornece a resposta certa temos que, para $j = 1, ..., s$, se $\sum_{k=1}^{n} x_k \notin [q_j, \tilde{q}_j]$, então $y_{q_j} + \tilde{y}_{q_j} = 1$ para todos os j. Assim

$$sgn\left(\sum_{k=1}^{s} \left(y_{k_j} + \tilde{y}_{k_j}\right) - s - 1\right) = sgn(s - s - 1) = 0 \tag{2.13}$$

Caso $\sum_{k=1}^{n} x_k \in [q_j, \tilde{q}_j]$, então $y_{q_j} + \tilde{y}_{q_j} = 2$ e $y_{q_i} + \tilde{y}_{q_i} = 1$ para $i \neq j$. Assim $sgn\left(\sum_{k=1}^{s}\left(y_{k_j} + \tilde{y}_{k_j}\right) - s - 1\right) = sgn\left(s + 1 - s - 1\right) = 1$.

Portanto, a primeira camada possui $2s$ portas. Como s é no máximo $\left\lceil \frac{n}{2} \right\rceil$ e a segunda camada possui apenas uma porta, o número total de portas de limiar linear utilizadas é de n +1. Como cada porta no circuito possui *fan-in* de no máximo n, o número de conexões é $O\left(n^2\right)$ (ver apêndice A). Para ilustrar o projeto de circuito de limiar linear com 2 camadas, na figura 2.5 está a implementação da função de paridade de 4 variáveis. Esse circuito necessita de 5 portas de limiar linear para implementar a paridade. Utilizando a lógica tradicional seriam necessárias 8 portas AND e 1 porta OR para implementar a mesma função. Se o número de variáveis fosse 10, seriam necessárias 512 portas AND e 1 portar OR, enquanto utilizando a lógica neural apenas 11 portas são necessárias.

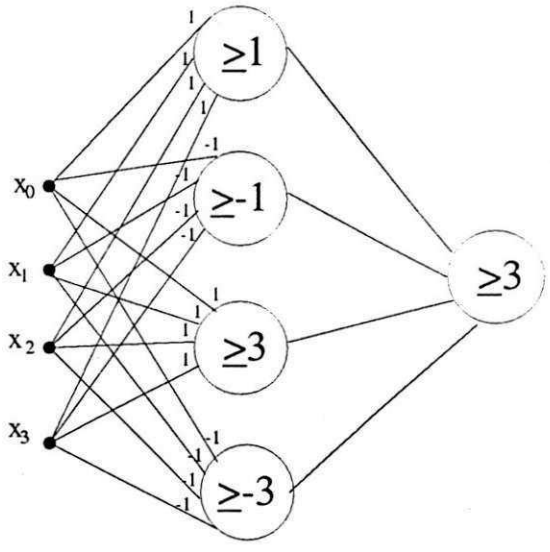

Figura 2.5: Paridade de 4 variáveis com circuito de 2 camadas.

É possível diminuir ainda mais o número de portas utilizadas aplicando uma técnica conhecida como *técnica telescópica.*

Lema 1 *Seja o intervalo* $[0, n]$ *dividido em* $s+1$ *subintervalos* $[b_0, b_1 - 1], [b_1, b_2 - 1], ..., [b_{s-1}, b_s - 1], [b_s, n]$, *em que* $0 = b_0 < b_1 < ... < b_k < n$.

Seja $y_i = sgn\left(\sum_{j=1}^{n} x_j - b_i\right)$, para todo $i = 1, ..., k$. Então:

$$\sum_{j=1}^{k} (a_j - a_{j-1})\, y_j = a_m \tag{2.14}$$

se $\sum_{j=1}^{n} x_j \in [b_m, b_{m-1} - 1]$, em que $a_0 = 0$ e $a_1, ..., a_k$ são números reais arbitrários. Como $y_i = sgn\left(\sum_{j=1}^{n} x_j - b_i\right) = 1$ se e somente se $\sum_{j=1}^{n} x_j \geq b_i$, portanto se $\sum_{j=1}^{n} x_j \in [b_m, b_{m+1}]$ então $y_1 = ... = y_m = 1$ e $y_{m+1} = ... = y_k = 0$. Assim:

$$\sum_{j=1}^{k} (a_j - a_{j-1})\, y_j = \sum_{j=1}^{m} (a_j - a_{j-1}) = a_m \tag{2.15}$$

Note que se $\sum_{j=1}^{n} x_j \in [b_0, b_1 - 1]$, isto é $m = 0$, então $y_i = 0$ para todo $i = 1, ..., k$ e consequentemente $\sum_{j=1}^{k} (a_j - a_{j-1})\, y_j = 0 = a_0$

Utilizando o lema acima pode-se obter um circuito de 2 camadas com $\left\lceil \frac{n}{2} \right\rceil + 1$ portas.

Seja $f(X)$ uma função simétrica de n variáveis. Definindo um conjunto de inteiros, s_i e S_i, com $0 \leq s_i \leq S_i \leq n$ para $i = 1, ..., \tau$ e $S_i + 1 < s_{i+1}$ para $i < \tau$, tal que $f(X) = 1$ se e somente se para algum i

$$s_i \leq \sum_{j=1}^{n} x_j \leq S_i \tag{2.16}$$

A primeira camada do circuito consiste de τ portas de limiar linear calculando $y_i = sgn\left(\sum_{j=1}^{n} x_j - s_i\right)$ para cada i, $1 \leq i \leq \tau$. A porta da saída na segunda camada executa

$$z = sgn\left(\sum_{j=1}^{n} (S_j - S_{j-1})\, y_j - \sum_{j=1}^{n} x_j\right) \tag{2.17}$$

em que é definido $S_0 = -1$.

Para verificar se o circuito fornece a resposta correta note que, para algum m, $\sum_{j=1}^{n} x_j \in [s_m, s_{m+1} - 1]$. Utilizando o lema 1 temos:

$$\sum_{j=1}^{\tau} (S_j - S_{j-1})\, y_j = S_m \tag{2.18}$$

Se $f(X) = 1$ então $s_m \leq \sum_{j=1}^{n} \leq S_m$ e portanto $z = sgn\left(S_m - \sum_{j=1}^{n} x_j\right) = 1$.

Por outro lado, se $f(X) = 0$, então $S_m < \sum_{j=1}^{n} x_j \leq s_{m+1} - 1$ e portanto $z = sgn\left(S_m - \sum_{j=1}^{n} x_j\right) = 0$

O número de portas de limiar linear no circuito é de $\tau + 1$. Como τ é no máximo $\left\lceil \frac{n}{2} \right\rceil$, então o número máximo de portas de limiar linear é de $\left\lceil \frac{n}{2} \right\rceil + 1$. Para ilustrar esse resultado na figura 2.6 está implementado um circuito que calcula a paridade de 4 variáveis utilizando apenas 3 portas de limiar linear.

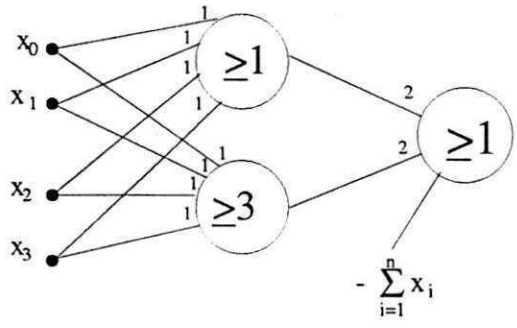

Figura 2.6: Paridade de 4 variáveis com circuito de 2 camadas com técnica telescópica.

2.3.2 Circuitos de 3 camadas

Generalizando a técnica telescópica para construções de 3 camadas é possível uma redução do número de portas de limiar linear. Como resultado são obtidos circuitos que executam funções simétricas exigindo apenas $2\sqrt{n} + 1$ portas de limiar linear.

Novamente utilizando um conjunto de inteiros s_i e S_i, em que $i = 1, ..., \tau$, com $s_i \leq S_i \leq s_{i+1}$ tal que $f(X) = 1$ se e somente se:

$$s_i \leq \sum_{j=1}^{n} x_j \leq S_i \tag{2.19}$$

Dividindo o intervalo $[0, n]$ em d subintervalos consecutivos $[s_1, s_{2_1} - 1], [s_2, s_{3_1} - 1], ..., [s_{d_1}, n]$ tal que cada subintervalo (exceto possivelmente o último) contenha o mesmo número l de inteiros s_i e S_i, em que $l \leq \left\lceil \frac{n}{2d} \right\rceil$. O i-ésimo subintervalo conterá os inteiros $s_{i_1} \leq S_{i_1} < s_{i_2} \leq S_{i_2} < ... < S_{i_1} \leq S_{i_1}$.

A primeira camada do circuito consiste em d elementos de limiar linear, cada um executando

$$z_i = sgn\left(\sum_{j=1}^{n} x_j - s_{i_1}\right) \tag{2.20}$$

para $i = 1, ..., d$

Para cada $k = 1, ..., l$ são definidas duas soma telescópicas dadas por:

$$\begin{aligned} T_k &= S_{1_k} z_1 + (S_{2_k} - S_{1_k}) z_2 + (S_{3_k} - S_{2_k}) z_3 + ... + (S_{d_k} - S_{d-1_k}) z_d \\ t_k &= s_{1_k} z_1 + (s_{2_k} - s_{1_k}) z_2 + (s_{3_k} - s_{2_k}) z_3 + ... + (s_{d_k} - s_{d-1_k}) z_d \end{aligned} \tag{2.21}$$

Observe que t_k e T_k são combinações lineares das saídas da primeira camada.

A segunda camada consiste em $2l$ portas, cada uma utilizando os inteiros t_k ou T_k como valor de limiar e computando Q_k e q_k definido a seguir:

$$Q_k = sgn(T_k - \sum_{j=1}^{n} x_j) \tag{2.22}$$

e

$$q_k = sgn(t_k - \sum_{j=1}^{n} x_j) \tag{2.23}$$

A terceira camada é um único elemento calculando

$$f(X) = sgn\left(\sum_{k=1}^{l} 2\left(Q_k + q_k\right) - 2l - 1\right) \tag{2.24}$$

Para verificar se o circuito fornece a resposta correta, supomos que $\sum_{j=1}^{n} x_j$ pertence ao m-ésimo intervalo $\sum_{j=1}^{n} x_j \in [s_{m_1}, s_{(m-1)_1} - 1]$. Utilizando o Lema 1, a soma telescópica de T_k e t_k assume os seguintes valores:

$$T_k = S_{m_k} \tag{2.25}$$

$$t_k = s_{m_k} \tag{2.26}$$

Por definição, $f(X) = 1$ se e somente se para algum k

$$s_{m_k} \leq \sum_{j=1}^{n} x_j \leq S_{m_k} \tag{2.27}$$

Na segunda camada $\sum_{j=1}^{n} x_j$ é comparado com $T_k = S_{m_k}$ e $t_k = s_{m_k}$ para cada k. Como $s_{m_i} \leq \sum_{j=1}^{n} x_j \leq S_{m_i}$ o valor da saída da segunda camada (Q_k, q_k) pode ser visto como

$$Q_k + q_k = \begin{cases} 2 & \text{se } k = i \\ 1 & \text{se } k \neq i \end{cases} \tag{2.28}$$

Portanto o elemento de saída da terceira camada é

$$sgn\left(\sum_{k=1}^{l} 2\left(Q_k + q_k\right) - 2l - 1\right) = sgn\left(2l + 2 - 2l - 1\right) = 1 \tag{2.29}$$

Similarmente, se $f(X) = 0$ então não existe k tal que $s_{m_k} \leq \sum_{j=1}^{n} \leq S_{m_k}$. Portanto $Q_k + q_k = 1$ para todo k e a porta da terceira camada executa

$$sgn\left(\sum_{k=1}^{l} 2\left(Q_k - q_k\right) - 2l - 1\right) = sgn\left(2l - 2l - 1\right) = 0 \tag{2.30}$$

Portanto o circuito fornece a saída correta para qualquer entrada $X = (x_1, ..., x_n)$. A primeira camada do circuito consiste em d elementos. A segunda camada possui $2l \leq \left\lceil \frac{n}{d} \right\rceil + 1$ portas e a terceira apenas uma porta. Como $d = \sqrt{n}$ então o tamanho do circuito de 3 camadas é de $2\sqrt{n} + 1$ portas de limiar linear.

Na figura 2.7 está implementado um circuito que calcula a paridade de 4 variáveis utilizando um circuito de 3 camadas com 5 portas de limiar linear. Observe que, embora esse circuito de três camadas necessite de 5 portas e o da figura 2.6 necessite de 3, quando o número de entradas aumentar o circuito de 3 camadas necessitará de menos portas. Por exemplo, se o número de entradas for 36, então o circuito de 2 camadas necessitará de 19 portas enquanto o circuito de 3 camadas necessitará de 13.

2.4 Implementação de Operações Aritméticas

Algumas funções booleanas, tais como adição e multiplicação, só podem ser calculadas utilizando as portas AON com número de camadas fixas em d se o tamanho aumentar exponencialmente. Isso resulta em uma área de chip que também

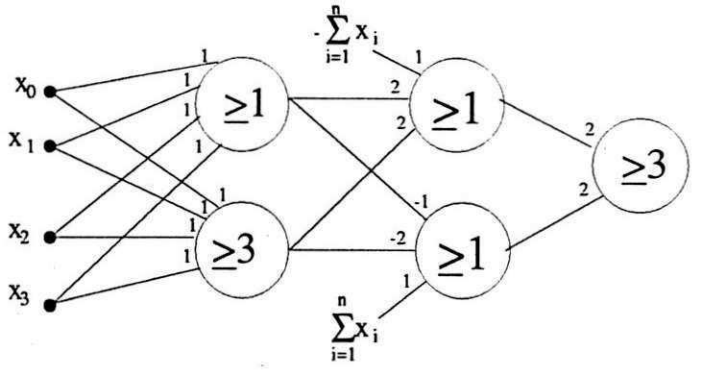

Figura 2.7: Paridade de 4 variáveis com circuito de 3 camadas com técnica telescópica

aumenta exponencialmente [25]. Para o tamanho do chip aumentar polinomialmente, seria necessário projetar circuitos com retardo não fixo, que acarretaria em circuitos mais lentos e portanto ineficientes para algumas aplicações.

Com o uso de redes neurais é possível obter circuitos com profundidade fixa e tamanho polinomial. A seguir, são mostrados alguns exemplos de operações aritméticas implementadas com redes neurais discretas.

2.4.1 Adição

Para um somador de dois números de 2 bits, $X = \{x_1, x_0\}$ e $Y = \{y_1, y_0\}$, é possível calcular a soma $Z = X + Y$ utilizando um circuito de limiar linear com 2 camadas.

Primeiramente é necessário encontrar os intervalos em que a função seja 1, como mostrado na tabela 2.1.

Note que o peso de x_1e y_1 é 2 e o peso de x_0 e y_0 é 1 (utilizando pesos inteiros). Dividindo os intervalos em que cada z_i seja 1 então:

- Bit $z_0 \rightarrow$ Os intervalos em que z_0 é 1 são $[1,1], [3,3], [5,5]$. Mas como z_0 é função apenas de x_0 e y_0 então é utilizado apenas o intervalo $[1,1]$
- Bit $z_1 \rightarrow$ Intervalos $[2;3]$
- Bit $z_2 \rightarrow$ Intervalos $[4;6]$

(x_1x_0)	(y_1y_0)	$soma(z_2z_1z_0)$	(x_1x_0)	$y_1y_0)$	$soma(z_2z_1z_0)$
00	00	000=0	10	00	010=2
00	01	001=1	10	01	011=3
00	10	010=2	10	10	100=4
00	11	011=3	10	11	101=5
01	00	001=1	11	00	011=3
01	01	010=2	11	01	100=4
01	10	010=2	11	10	101=5
01	11	100=4	11	11	110=6

Tabela 2.1: Soma de dois números de 2 bits

Com os intervalos obtidos, as portas da primeira camada podem ser implementadas da seguinte forma:

Bit 1

$$y_0 = sgn\left(\sum_{i=1}^{n} w_i x_i - 1\right)$$

$$\overline{y}_0 = sgn\left(1 - \sum_{i=1}^{n} w_i x_i\right)$$

$$z_0 = sgn\left(y_0 + \overline{y}_0 - 2\right)$$

Bit 2

$$y_1 = sgn\left(\sum_{i=1}^{n} w_i x_i - 2\right)$$

$$\overline{y}_1 = sgn\left(3 - \sum_{i=1}^{n} w_i x_i\right)$$

$$z_1 = sgn\left(y_1 + \overline{y}_1 - 2\right)$$

Bit 3

$$y_2 = sgn\left(\sum_{i=1}^{n} w_i x_i - 4\right)$$

$$\overline{y}_2 = sgn\left(6 - \sum_{i=1}^{n} w_i x_i\right)$$

$$z_2 = sgn\left(y_2 + \overline{y}_2 - 2\right)$$

Para esse somador são utilizadas 9 portas neurais, valor bem menor do que as 27 portas necessárias caso utilize a lógica tradicional. Embora tanto o somador com lógica tradicional quanto o somador com lógica neural possuam número de camadas constante, apenas o somador neural possui tamanho polinomial.

2.4.2 Multiplicação

A multiplicação de n bits pode ser divida em três etapas, como se vê na figura 2.8.

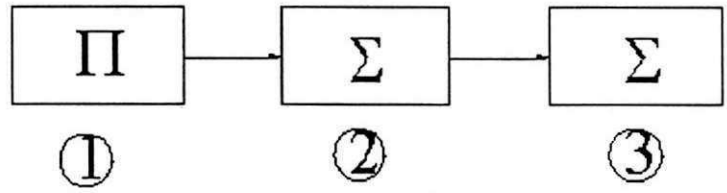

Figura 2.8: Bloco Funcional do Multiplicador.

O primeiro bloco computa apenas o produto de cada bit de uma entrada com cada bit da outra entrada. O segundo bloco utiliza a técnica *block-save*, que é um aprimoramento da técnica já conhecida *carry-save*, para implementar circuitos que executem a adição. O terceiro bloco é uma adição propriamente dita.

Bloco 1 - AND

O produto da multiplicação pode ser reduzido em uma soma múltipla. Seja $X = x_{n-1}x_{n-2}\cdots x_0$ e $Y = y_{n-1}y_{n-2}\cdots y_0$ dois números de n bits. O produto $Z = X \cdot Y$ é dado por:

$$Z = \sum_{i=0}^{n-1} z_{i_{2n-1}} z_{i_{2n-2}} \cdots z_{i_0}$$

para $i = 0, ..., n-1$ e

$$z_i = \underbrace{0 \cdots 0}_{n-i} (x_{n-1} \wedge y_i)\,(x_{n-2} \wedge y_i) \cdots (x_0 \wedge y_i) \underbrace{0 \cdots 0}_{i} \tag{2.31}$$

Utilizando uma porta de limiar linear para implementar a lógica AND da equação 2.31 seriam necessárias n^2 portas para encontrar todos os z_i 's. Portanto a multiplicação foi reduzida a uma soma de n números de $2n$ bits.

Bloco 2 - Técnica *Block-Save*

Utilizando a técnica *block-save*, a soma múltipla de n parcelas pode ser transformada em uma soma simples. Dividindo a soma múltipla em $\log n$ colunas a figura abaixo é obtida.

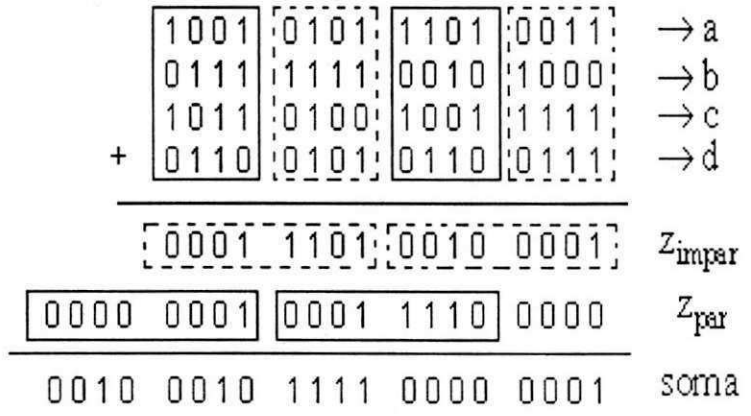

Figura 2.9: Exemplo da divisão em Soma Múltipla.

Assim cada coluna é somada separadamente da seguinte forma:

- Zerando as colunas pares, são somados os n números e o resultado armazenado em $z_{\text{ímpar}}$.
- Zerando as colunas ímpares, são somados os n números e o resultado armazenado em z_{par}.

Pode-se calcular z_{par} e $z_{\text{ímpar}}$ utilizando um circuito de limiar linear de duas camadas.

Bloco 3 - Soma de 2 números

Por fim, o bloco restante calcula a soma de $z_{\text{par}} + z_{\text{ímpar}}$(vê figura 2.11) Assim é necessário um circuito de 2 camadas para computar essa soma.

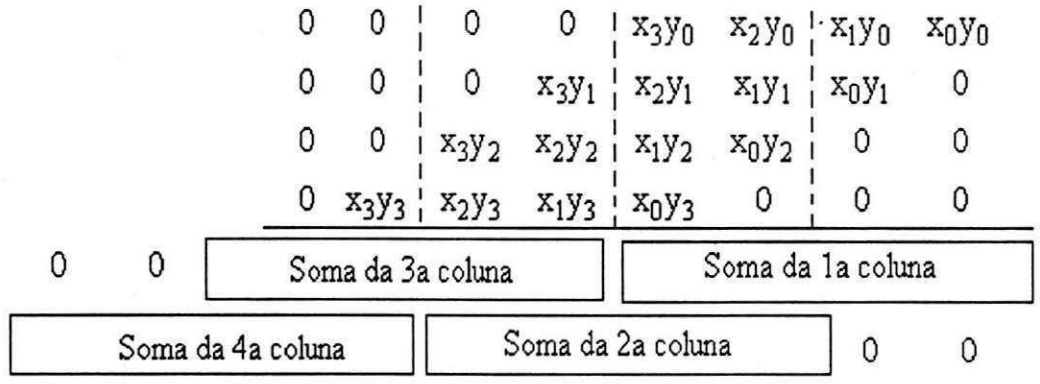

Figura 2.10: Soma Múltipla.

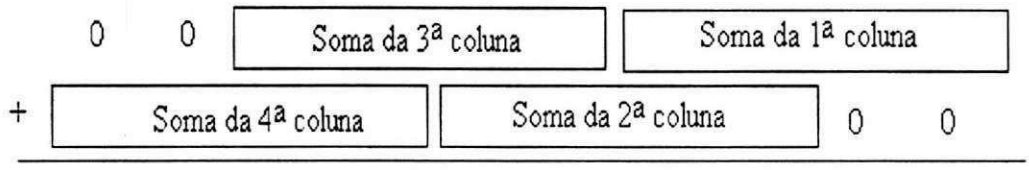

Figura 2.11: Soma das Colunas Pares e Ímpares.

Portanto a multiplicação de dois números de n bits pode ser realizada com um circuito de limiar linear de 5 camadas, em que a primeira computa um AND, a segunda e a terceira computa z_{par} e $z_{\text{ímpar}}$ e a quarta e a quinta computa a soma $z_{\text{par}} + z_{\text{ímpar}}$. Considerando que a soma múltipla é uma combinação linear entre a segunda e a terceira camada [25], pode-se conectar um somador na saída da terceira e a entrada da quarta camada. Assim só é preciso de quatro camadas para computar a multiplicação.

2.5 Conclusão

Neste capítulo foi apresentada a arquitetura da rede neural utilizada no trabalho. As portas de limiar linear, que é a estrutura básica dessa rede, possuem as portas tradicionais (AND,OR e NOT) como um subconjunto. Por isso, na pior hipótese, elas têm um comportamento igual ao da lógica tradicional [26].

Para extrair todas as vantagens das portas de limiar linear, como a diminuição da complexidade espacial de funções simétricas de exponencial para polinomial, são utilizadas estruturas específicas para construir circuitos de limiar linear. Por exemplo, a função paridade, que requer $2^{n-1}+1$ porta AON pode ser computada utilizando $\left\lceil \frac{n}{2} \right\rceil + 1$ portas de limiar linear. Ambas arquiteturas possuem retardo equivalente a 2 portas. Caso seja necessária uma diminuição do número de portas é possível utilizar um circuito com retardo igual a 3 que implementaria a paridade com $2\sqrt{n}+1$ portas.

Essa potencialidade das portas de limiar linear, já bem conhecidas em aplicações como adição, multiplicação e divisão em aritmética comum, pode ser estendida a outros tipos de sistemas numéricos. Um dos mais importantes sistemas numéricos em comunicação digital é o corpo finito, muito utilizado em códigos corretores de erro, sistemas de criptografia e processamento de sinais. Esse sistema possui várias operações aritméticas complexas, necessitando de uma quantidade muito grande de portas, para uma velocidade de processamento alta. Por isso, a rede neural discreta descrita anteriormente pode ser útil na busca de circuitos em corpos finitos mais eficientes.

Capítulo 3

Aritmética em Corpos Finitos

Diversas aplicações contam com poderosas estruturas algébricas chamadas de Corpos Finitos. Esses corpos são definidos por um conjunto finito de elementos e duas operações aritméticas fechadas.

Algumas aplicações de aritmética de corpo finitos são:

- **Códigos Corretores de Erro** - Uma idéia chave para desenvolver decodificadores de erro eficientes foi a introdução de um polinômio cuja raízes determinam a localização do erro. Para isso, utiliza-se um algoritmo específico como o algoritmo de Berlekamp-Massey ou algoritmo Euclidiano [27]. Outros exemplos de classes de códigos corretores de erros algébricos são os códigos Goppa, códigos de resíduo quadrático e códigos geométricos. Todos esses códigos dependem de uma *pesada* aritmética em campo de *Galois*.

- **Criptografia** - Na criptografia, o campo de *Galois* tornou-se uma ferramenta importante com a introdução de criptografia de chave pública. A segurança de vários sistemas para autenticação ou troca de chaves depende da dificuldade de inverter a função exponencial, que é o cálculo do logaritmo discreto do campo de Galois $GF(q)$.

- **Sequências Pseudo-Aleatórias** - O campo de *Galois* pode ser utilizado também na geração de sequências pseudo aleatórias (PN) através de configurações de registradores ou de exponenciação discreta. Essas sequências são utilizadas tanto em criptografia quanto em comunicação por espalhamento espectral.

Nesse capítulo são discutidas algumas operações aritméticas importantes em corpo finito. Circuitos tradicionais que implementam a adição, multiplicação, divisão, exponenciação e logaritmo de *Zech* são introduzidos, com destaque especial aos multiplicadores. Se o leitor achar necessário, informações referentes a algumas propriedades e características de um corpo finito podem ser encontradas no apêndice B. Informações mais detalhadas podem ser encontradas em [1, 28].

3.1 Adição

Um circuito somador em corpo finito é bastante simples de ser implementado. Sejam A, B e C elementos de $GF(2^n)$ representados em base polinomial por $A = a_0+a_1x+\cdots+a_{n-1}x^{n-1}$, $B = b_0+b_1x+\cdots+b_{n-1}x^{n-1}$ e $C = c_0+c_1x+\cdots+c_{n-1}x^{n-1}$. A soma $C = A + B$ pode ser calculada da seguinte forma:

$$\begin{aligned} C &= \left(a_0 + a_1x + \cdots + a_{n-1}x^{n-1}\right) + \left(b_0 + b_1x + \cdots + b_{n-1}x^{n-1}\right) \\ C &= (a_0 + b_0) + (a_1 + b_1)\,x + (a_2 + b_2)\,x^2 + \cdots + (a_{n-1} + b_{n-1})\,x^{n-1} \end{aligned} \tag{3.1}$$

Portanto para computar a adição são necessárias $3n$ portas AON.

3.2 Multiplicadores por Constante

Em codificadores e decodificadores, é frequente a multiplicação por valores constantes, como por exemplo no *codec* Reed-Solomon.

Seja $a = a_0 + a_1y + \cdots + a_{n-1}y^{n-1}$ um elemento de $GF(2^n)$ em que y seja a raiz do polinômio primitivo $p(x) = 1 + p_1x + \cdots + x^n$. Dessa forma

$$a \cdot y = a_0x + a_1x^2 + \cdots + a_{n-1} \bmod p(x) \tag{3.2}$$

Exemplo 1 *Seja* $p(x) = x^4 + x + 1$. *Assim a multiplicação de um elemento a por um valor constante x é dado por:*

$$\begin{aligned} a \cdot y &= a_0y + a_1y^2 + a_2y^3 + a_3(1 + y) \\ a \cdot y &= a_3 + (a_3 + a_0)\,y + a_1y^2 + a_2y^3 \end{aligned} \tag{3.3}$$

A figura 3.1 ilustra o circuito que computa $a \to a^2$ *em* $GF(2^n)$.

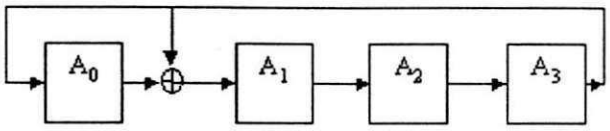

Figura 3.1: Circuito que executa $a \cdot x$ em $GF(2^n)$.

Os registradores são inicializados com $A_i = a_i$ e depois do primeiro ciclo do relógio $a \cdot x$ é gerado. Esse algoritmo pode ser estendido para a multiplicação $a \cdot x^j$.

3.3 Multiplicadores Seriais

A multiplicação serial é util em algumas arquiteturas cuja velocidade não é um fator preponderante. Embora seja lento, o multiplicador serial possui uma complexidade espacial bastante baixa devido ao uso de registradores. A seguir, alguns multiplicadores seriais são descritos.

3.3.1 Multiplicadores de Berlekamp

O multiplicador de Berlekamp utiliza duas representações: a base polinomial para o multiplicador e a base dual para o multiplicando e o produto. Como as entradas normalmente estão na mesma base, algum tipo de transformação de base dever ser executado. Porém, para $n = (3, 4, 5, 6, 7, 9, 10)$, a conversão entre base polinomial e dual, ou vice-versa, é meramente uma reordenação das bases do coeficiente (depende de $p(x)$).

Sejam $a, b, c \in GF(2^n)$ tal que $c = a \times b$ e $b = \sum_{k=0}^{n-1} b_k \cdot x^k$ representado em base polinomial e $\{\mu_0, \mu_1, \cdots, \mu_{n-1}\}$ a base dual da base polinomial para algum f e β. Assim $a = \sum_{i=0}^{n-1} a_i \mu_i$ e $c = \sum_{i=0}^{n-1} c_i \mu_i$ é a representação da base dual de a e c.

Lema 2 *Seja $\{\mu_0, \mu_1, \cdots, \mu_{n-1}\}$ a base dual da base polinomial em $GF(2^n)$ para algum f e β e seja $a = \sum_{i=0}^{n-1} a_i \mu_i$ a representação da base dual, em que $a_i \in GF(2^n)$. Então $a_i = f(a\beta\alpha^i)$ para $i = 0, 1, \cdots, n-1$.*

A multiplicação $c = a \times b$ pode ser representada na forma matricial

$$\begin{bmatrix} a_0 & a_1 & \cdots & a_{n-1} \\ a_1 & a_2 & \cdots & a_n \\ \vdots & \vdots & \cdots & \vdots \\ a_{n-1} & a_n & \cdots & a_{2n-2} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} \tag{3.4}$$

em que $a_i = f(a\beta\alpha^i)$ e $c_i = f(c\beta\alpha^i)$ são os coeficientes da base dual de a e c respectivamente.

Pode ser mostrado que:

$$a_{n+k} = f\left(a\beta\alpha^{m+k}\right) = \sum_{j=0}^{n-1} p_j \cdot a_{j+k} \tag{3.5}$$

em que p_j são os coeficientes de $p(x)$ [1].

Os valores de a_{n+k} podem ser obtidos através de um registrador de deslocamento linear de n estágios com realimentação. A cada ciclo de relógio, a_m é gerado, e no próximo ciclo a_{m+1} é produzido. Os n vetores da multiplicação listados na matriz são executados por um estrutura que compreende n portas AND e $(n-1)$ portas XOR.

Exemplo 2 *Para $GF(2^4)$ e $p(x) = x^4 + x + 1$ o multiplicador de Berlekamp é mostrador na figura 3.2 a seguir:*

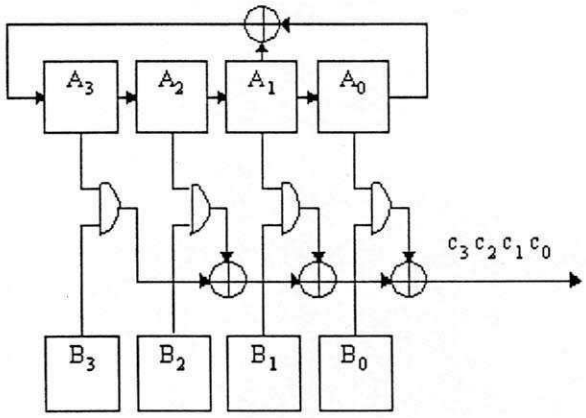

Figura 3.2: Multiplicador de Berlekamp em $GF(2^4)$.

Os registradores são inicializados com $A_i = a_i$ e $B_i = b_i$ para $i = 0, 1, 2$ e 3. Nesse ponto, o primeiro bit do produto c_0 é gerado na saída. Os valores de c_1, c_2 e c_3 são obtidos nos demais três ciclos de relógio.

Nesse esquema, no mínimo uma conversão de base é exigida no caso em que as entradas e a saída serem representadas na mesma base. Essa mudança pode ser implementada na própria estrutura do multiplicador

3.3.2 Multiplicador de Massey-Omura

O multiplicador de Massey-Omura opera inteiramente sobre a base normal e assim nenhuma conversão de base é necessária. A idéia do multiplicador de Massey-Omura é que a função booleana que gera o primeiro bit do produto é a mesma que gera os demais bits, sendo com a entrada deslocada.

Exemplo 3 *Seja a base normal* $\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^9\}$ *para* $GF(2^4)$ *e* $p(x) = y^4+y+1$ *o seu polinômio gerador. Assim a multiplicação* $c = a \times b$ *representada na base normal é dada por:*

$$c = c_0\alpha^3 + c_1\alpha^6 + c_2\alpha^{12} + c_3\alpha^9 \tag{3.6}$$

$$c = \left(a_0\alpha^3 + a_1\alpha^6 + a_2\alpha^{12} + a_3\alpha^9\right)\left(b_0\alpha^3 + b_1\alpha^6 + b_2\alpha^{12} + b_3\alpha^9\right) \tag{3.7}$$

$$\begin{aligned} c \;=\; & (a_0b_0)\,\alpha^6 + (a_0b_1 + a_1b_0)\,\alpha^9 + (a_1b_1 + a_0b_3 + a_3b_0)\,\alpha^{12} \\ & + (a_0b_2 + a_2b_0 + a_1b_3 + a_3b_1)\,\alpha^{15} + (a_1b_2 + a_2b_1 + a_3b_3)\,\alpha^{18} \\ & + (a_2b_3 + a_3b_2)\,\alpha^{21} + (a_2b_2)\,\alpha^{24} \end{aligned} \tag{3.8}$$

Como:

$$\begin{aligned} \alpha^{15} &= \alpha^3 + \alpha^6 + \alpha^{12} + \alpha^9 \\ \alpha^{18} &= \alpha^3 \\ \alpha^{21} &= \alpha^6 \\ \alpha^{24} &= \alpha^9 \end{aligned} \tag{3.9}$$

Temos:

$$\begin{aligned} c_0 &= a_0b_2 + a_1b_2 + a_1b_3 + a_2b_0 + a_2b_1 + a_3b_1 + a_3b_3 \\ c_1 &= a_1b_3 + a_2b_3 + a_2b_0 + a_3b_1 + a_3b_2 + a_0b_2 + a_0b_0 \\ c_2 &= a_2b_0 + a_3b_0 + a_3b_1 + a_0b_2 + a_0b_3 + a_1b_3 + a_1b_1 \\ c_3 &= a_3b_1 + a_0b_1 + a_0b_2 + a_1b_3 + a_1b_0 + a_2b_0 + a_2b_2 \end{aligned} \tag{3.10}$$

Pelas equação 3.10 pode-se notar que a função geratriz de c_0 é a mesma de c_1 (acrescentando 1 no índice, ou seja, deslocando a entrada).

O multiplicador está ilustrado na figura 3.3. Os registradores são inicializados por $A_i = a_i$ e $B_i = b_i$. Primeiramente o bit c_0 do produto é gerado na saída. A cada ciclo de relógio c_1, c_2 e c_3 são obtidos.

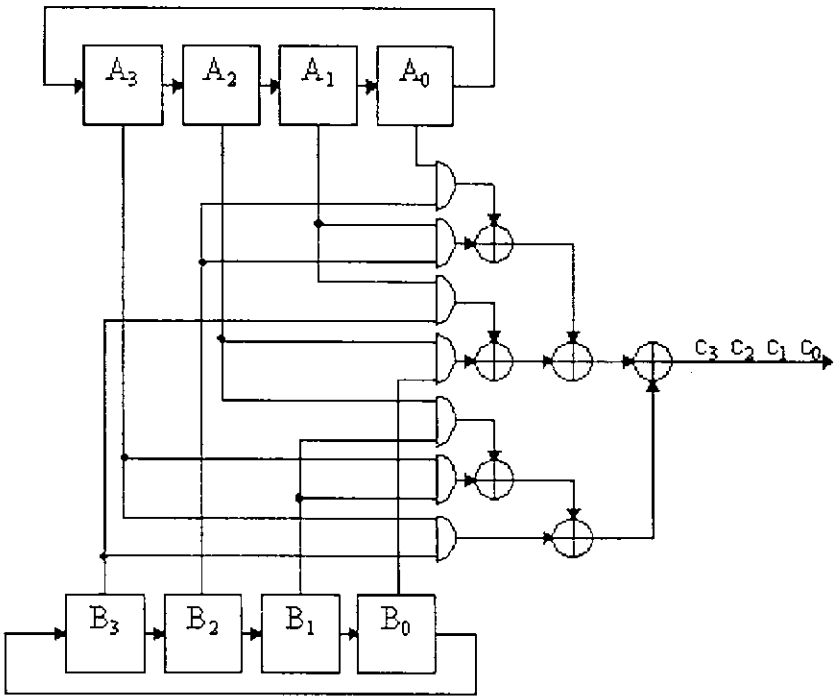

Figura 3.3: Multiplicador de Massey-Omura em $GF(2^4)$.

Em geral o multiplicador de Massey-Omura necessita de no mínimo $(2n-1)$ portas AND e no mínimo $(2n-2)$ portas XOR.

3.3.3 Multiplicador de Base Polinomial

O multiplicador de base polinomial opera totalmente em base polinomial. Esse multiplicador serial na realidade tem a entrada serial mas a saída paralela. O retardo do multiplicador é igual ao dos multiplicadores de Berlekamp e de Massey-Omura. Esse multiplicador pode ser classificado dependendo de qual bit entra primeiro no multiplicador.

Bit menos significativo primeiro

Nessa opção o bit menos significativo aparece primeiro na entrada do multiplicador.

Sejam $a, b, c \in GF(2^n)$ representados na forma polinomial. Assim:

$$\begin{aligned} c &= a \times b \\ c &= \left(a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}\right) \times b \\ c &= a_0 b + a_1 bx + \cdots + a_{n-1} bx^{n-1} \end{aligned} \tag{3.11}$$

Para implementar a equação acima é utilizado novamente o registrador de deslocamento para executar a multiplicação por x. O registrador é inicializado com b e a cada ciclo de relógio, bx é gerado. Os valores de $(a_0, a_1, \cdots, a_{n-1})$ são alimentados em série no multiplicador para cada valor de $a_i bx^i$, que são acumulados em um registrador para formar os bits do produto $(c_0, c_1, \ldots, c_{n-1})$.

Exemplo 4 *Considere o corpo finito $GF(2^3)$, gerado pelo polinômio primitivo $p(x) = x^3 + x + 1$. O multiplicador polinomial $c = a \cdot b$ com bit menos significativo primeiro é na figura 3.4.*

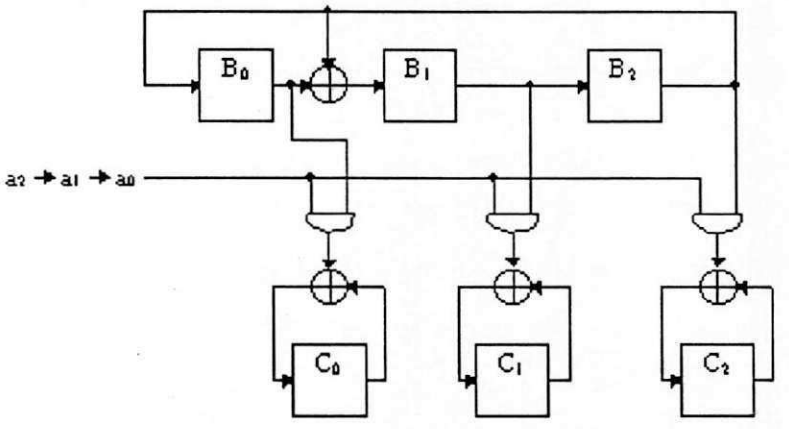

Figura 3.4: Circuito de um multiplicador polinomial em $GF(2^3)$ com bit menos significativo primeiro.

Os registradores são inicializados com $B_i = b_i$ e $C_i = 0$ para $(i = 0, 1, 2)$. Os valores a_0, a_1, a_2 são alimentados em série e depois de 3 ciclos de relógio o valor de c está disponível no registrador C.

Bit mais significativo primeiro

Nessa opção o bit mais significativo aparece primeiro na entrada do multiplicador.

A multiplicação $c = a \times b$ em base polinomial é:

$$\begin{aligned} c &= a \times b \\ c &= a_0 b + a_1 bx + \cdots + a_{n-1} bx^{n-1} \\ c &= (\cdots((a_{n-1}b)\,x + a_{n-2})\,x + a_{n-3}b)\,x + \cdots)x + a_0 b \end{aligned} \tag{3.12}$$

Exemplo 5 *Seja $p(x) = x^3 + x + 1$ o polinômio primitivo que gera elementos em $GF(2^3)$. O multiplicador polinomial $c = a \cdot b$ com bit mais significativo primeiro é implementado da seguinte forma:*

$$c = ((a_2 b)\,x + a_1 b)\,x + a_0 b$$

O circuito para implementar a equação acima é ilustrado na figura 3.5.

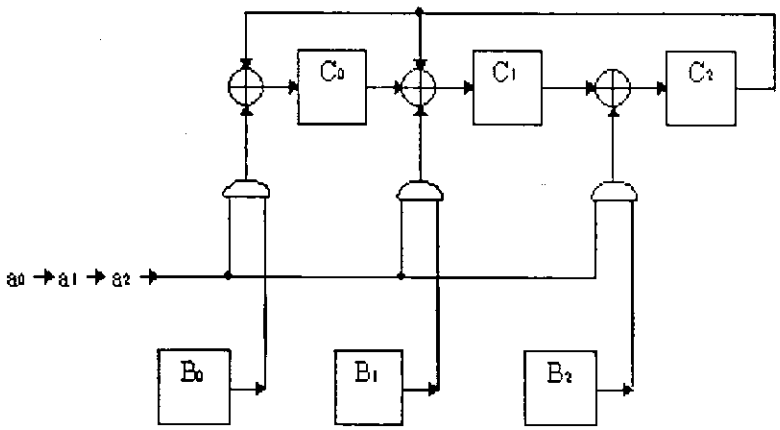

Figura 3.5: Circuito de um multiplicador polinomial em $GF(2^3)$ com bit mais significativo primeiro.

Inicialmente o registrador C_i é zerado e o registrador B_i é inicializado com $B_i = b_i$. O bit mais significativo a_2 alimenta o circuito e $a_2 b$ é carregado no registrador. Então, no próximo ciclo de relógio, a_1 entra no circuito e o registrador conterá $((a_2 b)\,x + a_1 b)$. Finalmente, no próximo ciclo de relógio, é gerado $(a_2 bx + a_1 b)\,x$ e esse valor é adicionado a $a_0 b$ para formar o produto. Esse resultado é obtido no registrador C_i depois de n ciclos de relógio.

3.3.4 Comparação dos Multiplicadores Seriais

- O Multiplicador de Berlekamp é conhecido por ter a menor exigência de *hardware*. Ele também pode ser facilmente modificado para uma particular e eficiente multiplicação por constante. A desvantagem desse multiplicador é que ele opera sobre as bases duais e polinomiais. Mas, na maioria dos casos, a conversão da base é apenas uma permutação dos coeficientes e portanto nenhum *hardware* adicional é necessário. Devido a essas razões, o multiplicador de Berlekamp é amplamente utilizado em projetos de *codecs*.

- O multiplicador de Massey-Omura opera totalmente sobre a base normal e nenhuma conversão de base é necessária. A representação é eficiente para algumas operações, como a potenciação. O circuito do multiplicador é ineficiente no que diz respeito a sua complexidade temporal e espacial (comparado ao multiplicador de Berlekamp) e não pode ser modificado para executar uma multiplicação por constante. Além disso, o multiplicador de Massey-Omura não pode ser facilmente estendido para valores diferentes de n.

- O multiplicador de base polinomial não necessita de conversores de base e é quase tão eficiente quanto o multiplicador de Berlekamp. Entretanto possui um *interface* diferente do multiplicador de Berlekamp, possuindo entrada serial e saída paralela.

Portanto, a escolha entre os multiplicadores de Berlekamp e os de base polinomial depende do circuito no qual o multiplicador irá ser implementado. Por exemplo, se o resultado tiver que ser representado em paralelo, o multiplicador polinomial pode ser utilizado. Caso contrário, o multiplicador de Berlekamp deve ser adotado.

Comparando diretamente os multiplicadores nota-se que todos necessitam de n ciclos de relógio para gerar a solução assim como necessitam de $2n$ *flip-flops*. Para comparar a complexidade desses multiplicadores é utilizada a notação *#(porta do tipo X)* para designar o número de portas do tipo X utilizadas na arquitetura.

Berlekamp:

$$\#AND = n \qquad \#XOR = n + H(p) - 3$$

$$\text{Retardo} = \tau_{AND} + \lceil \log_2 (n-1) \rceil \cdot \tau_{XOR}$$

obs: $H(p)$ é o peso de *Hamming* (número de coeficientes igual a 1) do polinômio primitivo em $GF(2^n)$ e τ_{AND} e τ_{XOR} são os retardos relativos as portas AND e XOR respectivamente.

Base Polinomial

Bit menos significativo primeiro:

$$\#AND = n \qquad \#XOR = n + H(p) - 2$$
$$\text{Retardo} = \tau_{AND} + \tau_{XOR}$$

Bit mais significativo primeiro:

$$\#AND = n \qquad \#XOR = n + H(p) - 2$$
$$\text{Retardo} = \tau_{AND} + 2 \cdot \tau_{XOR}$$

Massey-Omura

Para o multiplicador de Massey-Omura o número de portas não pode ser explicitado analiticamente.

A tabela 3.1 abaixo mostra quantas portas cada multiplicador serial necessita.

m	Massey-Omura	Berlekamp	Polinomial
3	$5AND + 4XOR$	$3AND + 3XOR$	$3AND + 4XOR$
4	$7AND + 6XOR$	$4AND + 4XOR$	$4AND + 5XOR$
5	$9AND + 8XOR$	$5AND + 5XOR$	$5AND + 6XOR$
6	$11AND + 10XOR$	$6AND + 6XOR$	$6AND + 7XOR$
7	$19AND + 18XOR$	$7AND + 7XOR$	$7AND + 8XOR$
8	$21AND + 20XOR$	$8AND + 10XOR$	$8AND + 11XOR$
9	$17AND + 16XOR$	$9AND + 9XOR$	$9AND + 10XOR$
10	$19AND + 18XOR$	$10AND + 10XOR$	$10AND + 11XOR$

Tabela 3.1: Comparação dos Multiplicadores Seriais

3.4 Multiplicadores Paralelos

Atualmente cada vez mais aplicações necessitam de velocidade maiores para serem compatíveis as aplicações. Por isso torna-se necessário adotar arquiteturas paralelas ao invés da serial para melhorar o desempenho, pois em aplicações em tempo real o retardo do multiplicador é um parâmetro crucial. Existem diversos tipos de multiplicadores paralelos. Para cada tipo de base são analisados os multiplicadores paralelos mais eficientes.

3.4.1 Multiplicadores de Base Dual

Sejam $a, c \in GF(2^n)$ representados na base dual como $a = a_0\mu_0 + a_1\mu_1 + \cdots + a_{n-1}\mu_{n-1}$ e $c = c_0\mu_0 + c_1\mu_1 + \cdots c_{n-1}\mu_{n-1}$. Seja $b \in GF(2^n)$ representado na base polinomial como $b = b_0x_0 + b_1x_1 + \cdots + b_{n-1}x_{n-1}$. A multiplicação $c = a \times b$ é portanto representada por:

$$\begin{bmatrix} a_0 & a_1 & \cdots & a_{n-1} \\ a_1 & a_2 & \cdots & a_n \\ \vdots & \vdots & \cdots & \vdots \\ a_{n-1} & a_n & \cdots & a_{2n-2} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} \tag{3.13}$$

em que $a_i = f(a\beta\alpha^i)$ e $c_i = f(c\beta\alpha^i)$ são os coeficientes da base dual de a e c respectivamente.

Os valores de a_i são obtidos da seguinte forma:

$$a_{n+k} = f\left(a\beta\alpha^{n+k}\right) = \sum_{j=0}^{n-1} p_j \cdot a_{j+k} \tag{3.14}$$

Utilizando essas equações e as características do multiplicador serial de Berlekamp (visto na seção anterior) , o multiplicador de base dual pode ser derivado [15] e implementado através das equações:

$$c_j = a_jb_0 + a_{j+1}b_1 + a_{j+2}b_2 + \ldots + a_{j+n+1}b_{n+1} \tag{3.15}$$

para $j = 0, 1, \ldots, n-1$ e

$$a_{n+k} = \sum_{j=0}^{n-1} p_j \cdot a_{j+k} \qquad (k = 0, 1, \ldots, n-1) \tag{3.16}$$

em que p_j são os coeficientes do polinômio primitivo $p(x) = p_0 + p_1 x + \ldots + x^n$.

Em geral, o multiplicador dual em $GF(2^n)$ é formado por um módulo do tipo A que gera a_{m+i} $(i = 0, 1, \ldots, n-1)$ da equação 3.16 e n módulos do tipo B que geram o produto interno dos vetores de comprimento n de $GF(2)$. Os módulos A e B são implementados como na figura 3.6.

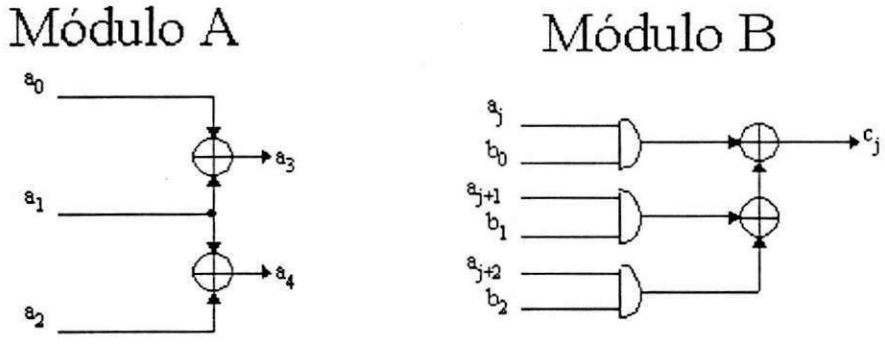

Figura 3.6: Módulo A e B para multiplicador paralelo de base dual em $GF(2^3)$.

Exemplo 6 *O multiplicador dual em* $GF(2^3)$ *para* $p(x) = x^3 + x + 1$ *é ilustrado na figura 3.7.*

3.4.2 Multiplicadores de Base Normal

O multiplicador paralelo de base normal foi primeiramente apresentado por Massey e Omura [1]. Esse multiplicador, também chamado de Multiplicador Massey-Omura, foi originalmente descrito para multiplicação serial mas posteriormente seu conceito foi utilizado em multiplicação em paralelo. A sua arquitetura é desenvolvida da seguinte forma:

Considere dois elementos A e B em base normal:

$$\begin{aligned} A &= a_0 x + a_1 x^2 + a_2 x^{2^2} + \cdots + a_{n-1} x^{2^{n-1}} \\ B &= b_0 x + b_1 x^2 + b_2 x^{2^2} + \cdots + b_{n-1} x^{2^{n-1}} \end{aligned} \tag{3.17}$$

Uma propriedade interessante da representação em base normal é que o quadrado de um elemento é simplesmente um deslocamento cíclico de seus coeficien-

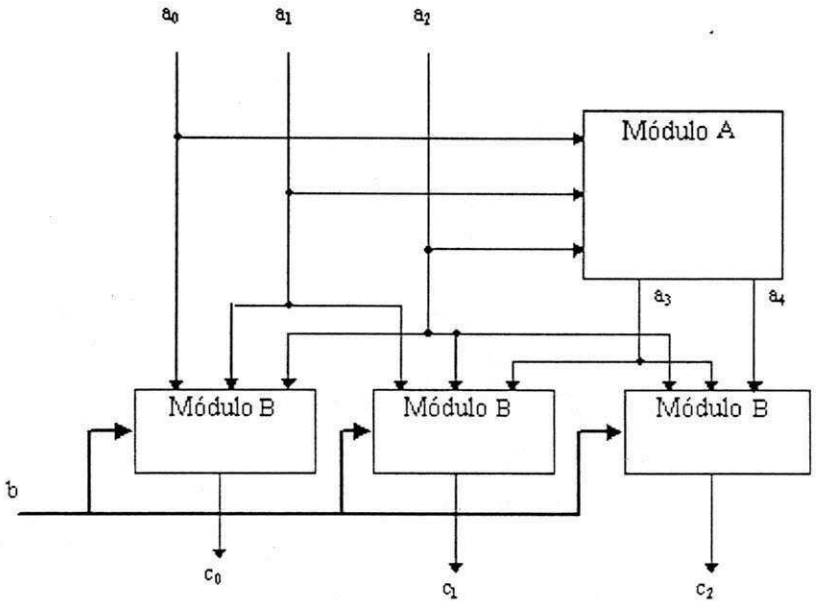

Figura 3.7: Multiplicador paralelo de base dual em $GF(2^3)$.

tes.

$$\begin{aligned} A^2 &= a_{n-1}x + a_0x^2 + a_1x^{2^2} + \cdots + a_{n-2}x^{2^{n-1}} \\ B^2 &= b_{n-1}x + b_0x^2 + b_1x^{2^2} + \cdots + b_{n-2}x^{2^{n-1}} \end{aligned} \tag{3.18}$$

A multiplicação $C = A \cdot B$ é dada por

$$C = c_0x + c_1x^2 + c_2x^{2^2} + \cdots + c_{n-1}x^{2^{n-1}} \tag{3.19}$$

Primeiramente, só o coeficiente de maior grau c_{n-1} é considerado. Ele pode ser especificado como função dos coeficientes de A e B.

$$c_{n-1} = f\left(a_0, a_1, \ldots, a_{n-1}; b_0, b_1, \ldots, b_{n-1}\right) \tag{3.20}$$

Elevando os dois membros da equação $C = A \cdot B$ ao quadrado têm-se:

$$\begin{aligned} C^2 &= A^2 \cdot B^2 \\ C^2 &= c_{n-1}x + c_0x^2 + \cdots + c_{n-2}x^{2^{n-1}} \end{aligned} \tag{3.21}$$

Dessa forma é obtido uma equação similar a 3.20 para o coeficiente c_{n-2}.

$$c_{n-2} = f\left(a_{n-1}, a_0, \ldots, a_{n-2}; b_{n-1}, b_0, \ldots, b_{n-2}\right) \tag{3.22}$$

A função da equação acima é a mesma da equação 3.20 mas com os dois conjuntos de entradas $(a_{n-1}, a_0, ..., a_{n-2})$ e $(b_{n-1}, b_0, ..., b_{n-2})$. Os outros coeficientes de C podem ser obtidos a partir da função f através do mesmo procedimento, ou seja, através de deslocamentos repetitivos dos elementos da entrada.

Exemplo 7 *Considere um elemento em* $GF(2^4)$ *com polinômio primitivo* $p(x) = x^4 + x^3 + 1$. *Seja base normal* (x^8, x^4, x^2, x). *A multiplicação de dois elementos* $C = A \cdot B$ *em base normal é dada por:*

$$\begin{aligned} C &= c_3x^8 + c_2x^4 + c_1x^2 + c_0x \\ &= \left(a_3x^8 + a_2x^4 + a_1x^2 + a_0x\right)\left(b_3x^8 + b_2x^4 + b_1x^2 + b_0x\right) \\ &= x^{12}\left(a_2b_3 + a_3b_2\right) + x^{10}\left(a_1b_3 + a_3b_1\right) + x^9\left(a_3b_0 + a_0b_3\right) \\ &\quad + x^8\left(a_2b_2\right) + x^6\left(a_2b_1 + a_1b_2\right) + x^5\left(a_2b_0 + a_0b_2\right) \\ &\quad + x^4\left(a_1b_1\right) + x^3\left(a_0b_1 + a_1b_0\right) + x^2\left(a_0b_0\right) + w\left(a_3b_3\right) \end{aligned} \tag{3.23}$$

Essa multiplicação gera elementos $(x^{12}, x^{10}, x^9, x^6, x^5, x^3)$ *que devem ser expressos na base normal:*

$$\begin{aligned} x^{12} &= x^8 + x^4 + x^2 \\ x^{10} &= x^8 + x^2 \\ x^9 &= x^8 + x^4 + x \\ x^6 &= x^4 + x^2 + x \\ x^5 &= x^4 + x \\ x^3 &= x^8 + x^2 + x \end{aligned}$$

Portanto o coeficiente c_3 *é*

$$\begin{aligned} c_3 &= f\left(a_0, a_1, a_2, a_3; b_0, b_1, b_2, b_3\right) \\ c_3 &= a_2b_3 + a_3b_2 + a_1b_3 + a_3b_1 + a_3b_0 + a_0b_3 + a_2b_2 + a_0b_1 + a_1b_0 \end{aligned} \tag{3.24}$$

A soma dos produtos na equação acima é a função f que precisa ser determinada.

É óbvio que a multiplicação em base normal para um corpo é determinada pela função f, que também determina a complexidade do multiplicador. Mas, como a função f é determinada pelo polinômio $p(y)$, então a complexidade do multiplicador depende apenas desse polinômio. O número de produtos C_n da função f é usualmente tomado como a medição da complexidade. Nesse exemplo $C_n = 9$.

Em [29] é mostrado que a complexidade C_n tem como limitante inferior $C_n \geq 2n-1$ e $C_n = 2n-1$, sendo chamada de base normal ótima. Assim a complexidade total do multiplicador é:

$$\begin{aligned} \#AND &= C_n \geq 2n^2 - n \\ \#XOR &= C_n \geq 2n^2 - 3n + 1 \end{aligned}$$

3.4.3 Multiplicadores de Base Polinomial

O multiplicador executa a mesma sequência de cálculos do multiplicador serial de base polinomial.

Sejam $a, b, c \in GF(2^n)$ representado por:

$$\begin{aligned} a &= a_0 + a_1\alpha + \cdots + a_{n-1}\alpha^{n-1} \\ b &= b_0 + b_1\alpha + \cdots + b_{n-1}\alpha^{n-1} \\ c &= c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1} \end{aligned} \tag{3.25}$$

Para gerar $c = a \times b$, a representação abaixo é utilizada:

$$c = (\cdots((a_{n-1}b)\,\alpha + a_{n-2})\,\alpha + a_{n-3}b)\,\alpha + \cdots)\alpha + a_0 b \tag{3.26}$$

O Multiplicador de base polinomial consiste em $(m-1)$ blocos que executam as operações:

$$\begin{aligned} y_{m-1} &= a_{m-1}b \\ y_j &= a_j b + y_{j+1}\alpha \bmod p(\alpha) \qquad \text{para } m-1 \geq j \geq 0 \end{aligned} \tag{3.27}$$

em que o resultado final da multiplicação é $c = y_0$ e $p(x)$ é o polinômio gerador de $GF(2^n)$.

3.4.4 Multiplicador de Mastrovito

Em [14], Mastrovito apresentou um tipo diferente de multiplicador de base polinomial. Esse multiplicador gera:

$$c = a \cdot b \mod p(x) = \sum_{j=0}^{m-1} c_j \alpha^j \tag{3.28}$$

utilizando a matriz de produto M :

$$\begin{bmatrix} c_{m-1} \\ c_{m-2} \\ \vdots \\ c_0 \end{bmatrix} = \begin{bmatrix} f_{m-1}^{m-1} & f_{m-2}^{m-1} & \cdots & f_0^{m-1} \\ f_{m-1}^{m-2} & f_{m-2}^{m-2} & \cdots & f_0^{m-1} \\ \vdots & \vdots & \vdots & \vdots \\ f_{m-1}^0 & f_{m-2}^0 & \cdots & f_0^0 \end{bmatrix} \cdot \begin{bmatrix} a_{m-1} \\ a_{m-2} \\ \vdots \\ a_0 \end{bmatrix} = M \cdot A^T \tag{3.29}$$

O maior dificuldade no algoritmo de Mastrovito é encontrar a matriz M. Devido a essa dificuldade, ele é difícil de ser representado através de algoritmos. Entretanto, o algoritmo de Mastrovito tem a vantagem de possuir um retardo bem menor que o multiplicador polinomial.

A multiplicação paralela também pode ser calculada através da multiplicação polinomial convencional e operação módulo executadas separadamente. Assim, para obter $c = a \times b$, primeiramente é obtido um polinômio de grau $2n-2$ da seguinte forma:

$$\begin{aligned} c \;=\; & (a_0 b_0) + (a_0 b_1 + a_1 b_0)\, x \\ & + (a_0 b_2 + a_1 b_1 + a_2 b_0)\, x^2 + \cdots \\ & + (a_{n-1} b_{n-2} + a_{n-2} b_{n-1})\, x^{2n-3} \\ & + (a_{n-1} b_{n-1})\, x^{2n-2} \end{aligned} \tag{3.30}$$

Depois a operação módulo é utilizada para reduzir o grau de c de $2n-2$ para $n-1$. O multiplicador de Mastrovito é mais detalhado no capítulo seguinte, onde será discutido a sua complexidade espacial e temporal.

3.4.5 Multiplicador de Paar

O multiplicador de Paar opera sobre a corpo de extensão $GF((2^n)^m)$, com $k = nm$. A multiplicação é dividida em dois passos [29]. O primeiro é a multiplicação polinomial, que é feita utilizando o algoritmo de *Karatsuba-Ofman (KOA)* e a redução módulo tradicional.

- Algoritmo de *Karatsuba-Ofman*

Seja $A = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$ e $B = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$ dois elementos de $GF(2^n)$. Aplicando o algoritmo, ambos os polinômios são divididos em uma metade inferior e outra superior, como mostrado a seguir:

$$\begin{aligned} A &= x^{\frac{n}{2}} \left(x^{\frac{n}{2}-1} a_{n-1} + \cdots + a_{\frac{n}{2}} \right) + \left(x^{\frac{n}{2}-1} a_{\frac{n}{2}-1} + \cdots + a_0 \right) \\ A &= x^{\frac{n}{2}} A_h + A_l \qquad (3.31) \\ B &= x^{\frac{n}{2}} \left(x^{\frac{n}{2}-1} b_{n-1} + \cdots + b_{\frac{n}{2}} \right) + \left(x^{\frac{n}{2}-1} b_{\frac{n}{2}-1} + \cdots + b_0 \right) \qquad (3.32) \\ B &= x^{\frac{n}{2}} B_h + B_l \end{aligned}$$

Um conjunto de polinômios $D(x)$ é definido como:

$$\begin{aligned} D_0(x) &= A_l(x) B(x) \qquad (3.33) \\ D_1(x) &= [A_l(x) + A_h(x)] [B_l(x) + B_h(x)] \\ D_2(x) &= A_h(x) B_h(x) \end{aligned}$$

Utilizando a equação 3.31, o produto polinomial $C'(x) = A(x) \cdot B(x)$ é obtido por:

$$C'(x) = D_0(x) + x^{\frac{n}{2}} [D_1(x) - D_0(x) - D_2(x)] + x^n D_2(x) \qquad (3.34)$$

Esse procedimento reduz o número de multiplicações por $\frac{3}{4}n^2$. Aplicando o algoritmo recursivamente nos três polinômios da eq. 3.31, ele irá finalizar depois de $\log_2 n$ passos. No último passo, os polinômios $D_i^j(x)$ são reduzidos a simples coeficientes, ou seja, $\deg(D^j(x)) = 0$. O número de portas necessárias para executar esse algoritmo é:

$$\#AND = n^{\log_2 3} \qquad (3.35)$$

$$\#XOR \leq 6m^{\log_2 3} - 8m + 2 \tag{3.36}$$

O retardo desse algoritmo pode ser calculado como:

$$T = \tau_{AND} + 3\left(\log_2 n\right)\tau_{XOR} \tag{3.37}$$

em que τ_{AND} e τ_{XOR} denota o retardo relativo a porta AND e XOR respectivamente.

Note que as operações executadas na eq. 3.31, quando o algoritmo está finalizado (polinômios de grau zero), são operações em $GF(2)$. Portanto, para calcular a multiplicação em $GF(2^n)$ são utilizadas estruturas baseadas no corpo de base $GF(2)$.

3.4.6 Comparação dos Multiplicadores Paralelos

Nessa seção foi discutido os tipos de multiplicadores paralelos. Uma comparação no retardo máximo e no número de portas XOR e AND necessárias para esses multiplicadores estão apresentadas nas tabelas 3.2 e 3.3.

n	Base Dual	Base Normal	Base Polinomial	Mastrovito	Paar
4	3	3	4	3	4
5	5	3	6	5	
6	4	4	6	4	5
7	4	5	7	4	
8	7	5	11	5	5
9	6	4	11	5	
10	6	5	12	6	7

Tabela 3.2: Retardo referente aos Multiplicadores Paralelos em $GF(2^n)$

O multiplicador de base normal é pouco utilizado devido a sua alta exigência de portas. Os multiplicadores polinomiais, Paar e Mastrovito não necessitam de nenhuma conversão de base e portanto o projeto desses multiplicadores polinomiais é mais simples e eficiente com respeito a complexidade espacial do que o Multiplicador Dual, especialmente se o trinômio primitivo de $GF(2^n)$ não existir.

n	Base Dual		Base Normal		Base Polinomial		Mastrovito		Paar	
	AND	XOR	AND	XOR	AND	XOR	AND	XOR	AND	XOR
4	16	15	28	24	16	15	16	15	12	18
5	25	25	45	40	25	24	25	24		
6	36	35	66	60	36	35	36	35	27	37
7	49	48	133	126	49	48	49	48		
8	64	77	160	160	64	77	64	90	48	62
9	81	80	153	144	81	80	81	80		
10	100	99	190	180	100	99	100	99	75	95

Tabela 3.3: Número de Portas AND e XOR referente aos Multiplicadores Paralelos em $GF(2^n)$

O multiplicador de Mastrovito e de base Polinomial possuem complexidade muito próximas, mas o multiplicador Polinomial possui um retardo maior. O multiplicador de Paar possui uma complexidade espacial menor que o Mastrovito para alguns n específicos, como em $GF(2^8)$. Mas em outros corpos sua complexidade é maior do que o Mastrovito [7]. Assim, o multiplicador de Mastrovito possui as melhores características dentre os multiplicadores paralelos revistos.

Deve ser mencionado que os multiplicadores paralelos propostos sempre levam em conta qual polinômio primitivo $p(x)$ está sendo utilizado. Isso ocorre devido a dificuldade de se obter uma solução geral para o problema.

3.5 Exponenciação

Seja $A \in GF(2^n)$ representado por:

$$A = a_0 + a_1 x + a_2 x^2 + \cdots a_{n-1} x^{n-1} \tag{3.38}$$

Agora seja $B \in GF(2^n)$ tal que $B = A^k$, ou

$$B = \underbrace{A \cdot A \cdot A}_{k} \tag{3.39}$$

Portanto a exponenciação pode ser executada utilizando um simples multiplicador k vezes consecutivas.

3.6 Divisão

Existem vários algoritmos que executam a divisão $c = \frac{a}{b}$ em um corpo finito, entre eles o algoritmo euclidiano. Essa operação é constante em decodificadores BCH, que necessitam que o resultado da divisão esteja disponível mais rápido do que esse algoritmo permite. Entretanto, b geralmente está disponível antes que a e assim pode-se utilizar esse benefício para primeiramente gerar b^{-1} (através do inversor de Fermat) e depois calcular a multiplicação $c = a \times b^{-1}$, que é uma operação rápida.

3.7 Logaritmo de Zech

A multiplicação utilizando notação exponencial pode ser facilmente calculada por $x^a x^b = x^{a+b \bmod (q-1)}$. Mas o mesmo não ocorre na adição, sendo mais fácil calculá-la através da notação polinomial. Seria conveniente utilizar apenas uma representação para calcular ambas as operações. Uma tabela pré-calculada pode ser utilizada para facilitar a adição de corpos representados como potência. Essa tabela é chamada de *logaritmo de Zech* (ver apêndice C) . O logaritmo de *Zech* é calculado da seguinte forma:

$$x^a + x^b = x^a x^{Z(b-a)} \tag{3.40}$$

em que $a < b$ e $x^{Z(i)} = 1 + x^i$.

De posse da tabela, a soma $x^{Z(i)} = 1 + x^i$ pode ser facilmente calculada. Para $GF(8)$ e $P(x) = x^3 + x + 1$, a seguinte tabela de logaritmos de *Zech* é obtida. *Obs:* por convenção $Z(0) = \infty$ e $Z(\infty) = 0$

Com a tabela formada pode-se calcular a soma e a multiplicação utilizando para ambos a representação exponencial. Por exemplo, para calcular $x^4 + x^2$ utilizando a tabela acima têm-se:

$$x^4 + x^2 = x^2(1 + x^2) \tag{3.41}$$

Procurando na tabela quem é $Z(2)$ é encontrado 6. Então:

$$x^4 + x^2 = x^2 \times x^6 = x^{8 \bmod 7} = x. \tag{3.42}$$

i	Z (i)
∞	0
0	∞
1	3
2	6
3	1
4	5
5	4
6	2

Tabela 3.4: Logaritmo de Zech em $GF(8)$

O único inconveniente do logaritmo de *Zech* é que para um corpo maior, como por exemplo $GF(256)$, a tabela tornar-se-ia bastante grande. Existem algumas propriedades que ajudam a construir a tabela de logaritmo de *Zech* [30]. São elas:

i) $Z(0) = Z(\infty)$

ii) $Z(\infty) = Z(0)$

iii) $Z(px) = pZ(x)$

iv) $Z(q-1-x) = Z(x) - x(\text{mod}(q-1))$

v) Se a característica do corpo for 2, então $Z(a) = b$ e $Z(b) = a$

3.8 Conclusão

Neste capítulo foram revistos alguns operadores aritméticos em corpos finitos, bem como o projeto de circuitos que os implementam. Em aplicações que utilizam constantemente multiplicações em corpo finito, a aritmética padrão presente nos microprocessadores não é desejável. Uma solução simples em termos de custo é a implementação por software, porém ela é bastante lenta. Um outra possível solução é empregar uma unidade de Processador Digital de Sinal (DSP), mas essa opção é cara e só pode ser adotada quando o número de variáveis é pequeno.

Assim, em aplicações em que a rapidez seja necessária, a solução ideal é utili-

zar *hardware* dedicado. Em particular, projetos de multiplicadores paralelos em $GF(2^n)$ de diversos tipos foram apresentados e o multiplicador de Mastrovito se destacou com respeito a complexidade temporal e espacial.

Como visto anteriormente, as redes neurais empregadas em aritmética tradicional proporcionaram um desempenho melhor. O próximo capítulo verifica que ao utilizar redes neurais discretas em aritmética de corpos finitos, em particular a multiplicação, há também uma melhora no desempenho da função.

Capítulo 4

Análise de Complexidade e Resultados

Como visto no capítulo anterior, o multiplicador de Mastrovito apresenta uma das menores complexidades espaciais e temporais dentre os multiplicadores em paralelo. A arquitetura desse multiplicador torna ele mais adequado para a sua implementação com redes neurais discretas. Nesse capítulo são apresentadas duas arquiteturas de multiplicadores em corpos finitos utilizando redes neurais discretas.

- *Arquitetura 1* - Partindo da arquitetura proposta por Mastrovito (multiplicação polinomial e redução módulo $p(x)$), um novo multiplicador em $GF(2^n)$ é obtido utilizando redes neurais discretas. Essa novo multiplicador apresenta um número de portas menor do que o obtido com o uso de portas AON, considerando portas com *fan-in* ilimitado.

- *Arquitetura 2* - Nesta outra arquitetura proposta, o multiplicador em corpo em $GF(2^n)$ é obtido com a multiplicação polinomial e redução módulo $p(x)$ em um único passo. Essa arquitetura apresenta um retardo menor do que a primeira arquitetura e para $n < 17$ o número de portas de limiar linear também é menor do que a *arquitetura 1*.

No apêndice C está uma implementação de uma solução com portas de limiar linear para o logaritmo de *Zech*. Essa solução permite o cálculo desse logaritmo sem a utilização de tabelas [31].

Na seção 4.1 é obtida, algebricamente, a complexidade temporal e espacial do multiplicador de Mastrovito para um polinômio primitivo da forma $p(x) = x^n + x + 1$ e um polinômio irredutível da forma $p(x) = x^n + x^{n-1} + \cdots + x + 1$. Na seção 4.2 é apresentada a *arquitetura 1*, que é baseado na arquitetura apresentada por Mastrovito. Na seção 4.3 a *arquitetura 2* é introduzida. Finalmente, a seção 4.4 compara os multiplicadores com redes neurais propostos com o multiplicador de Mastrovito.

4.1 Multiplicador de Mastrovito

Sejam $A(x)$ e $B(x)$ polinômios em $GF(2^n)$. A multiplicação $A(x) \cdot B(x)$ é dada por:

$$C(x) = A(x) \cdot B(x) \bmod p(x) \tag{4.1}$$

Como é conhecido [14], o multiplicador de Mastrovito é implementado em duas etapas: a multiplicação polinomial e a redução módulo $p(x)$, como pode ser observado na figura 4.1.

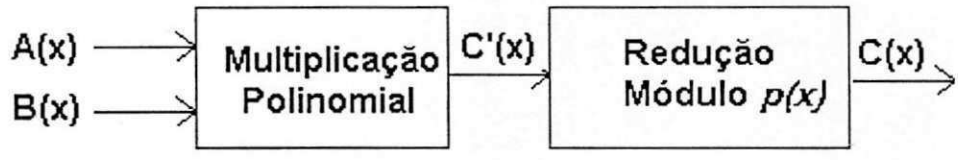

Figura 4.1: Diagrama de Bloco do Multiplicador de Mastrovito

Essas duas etapas são detalhadas a seguir.

4.1.1 Multiplicação Polinomial Ordinária

A multiplicação descrita na equação 4.1 pode ser reescrita como:

$$C(x) = C'(x) \bmod p(x) \tag{4.2}$$

Reescrevendo temos:

$$C'(x) = c'_0 + c'_1 x + c'_2 x^2 + \cdots c'_{2n-2} x^{2n-2} \tag{4.3}$$

Cada elemento c'_i é dado por:

$$c'_j = \begin{cases} \sum_{i=0}^{j} a_i b_{j-i} & \text{para } j < n \\ \sum_{i=j-n+1}^{n-1} a_i b_{2n-j-1} & \text{para } j \geq n \end{cases} \tag{4.4}$$

Independente do polinômio $p(x)$, são suficientes n^2 portas AND para executar as operações $a_i b_j$, em que $i = 1, ..., n$ e $j = 1, ..., n$. Para cada c_j, o número de elementos no somatório da equação 4.4 é dado por:

$$\begin{array}{rl} (j+1) \text{ elementos} & \text{para } j < n \\ (2n-j-1) \text{ elementos} & \text{para } j \geq n \end{array} \tag{4.5}$$

Como visto na seção 2.3, para implementar a função paridade com n variáveis e *fan-in* ilimitado são necessárias $2^{n-1}+1$ portas AON e retardo constante igual a $2\tau_{AON}$, em que τ_{AON} corresponde ao retardo de uma porta de lógica tradicional. Seja $\#AON$ o número de portas AON. Como cada c'_i é obtido a partir de um somatório de elementos, então o número de portas AON necessárias para obter os coeficientes c'_j é:

$$\begin{aligned} \#AON &= \sum_{j=1}^{n-1} \left(2^j + 1\right) + \sum_{j=n}^{2n-3} \left(2^{2n-j-2} + 1\right) \\ &= (n-1) + \sum_{j=1}^{n-1} 2^j + (n-2) + 2^{2n-2} \sum_{j=n}^{2n-3} 2^{-j} \\ &= (2n-3) + 2^n - 2 + 2^{2n-2} \left(\frac{2^{2n-2}-1}{2^{2n-3}} - \frac{2^n - 1}{2^{n-1}} \right) \\ &= 2n - 5 + 2^n + 2^{2n-1} - 2 - 2^{2n-1} + 2^{n-1} \\ &= 2n - 7 + 2^{n-1}(2+1) \\ \#AON &= 3 \cdot 2^{n-1} + 2n - 7 \end{aligned} \tag{4.6}$$

Considerando a operação AND bit a bit inicial (que necessita de n^2 portas), o número total de portas para executar a multiplicação polinomial convencional é:

$$\#AON = 3 \cdot 2^{n-1} + n^2 + 2n - 7 \tag{4.7}$$

O retardo da multiplicação polinomial convencional é $1\tau_{AON}$ para o AND bit a bit e $2\tau_{AON}$ para as somas dos coeficientes (assumindo *fan-in* ilimitado).

Exemplo 8 *Sejam $A(x)$ e $B(x)$ dois elementos em $GF(2^4)$, para $p(x) = x^4 + x + 1$. A multiplicação convencional $C'(x) = A(x) \cdot B(x)$ é realizada da seguinte forma:*

$$\begin{aligned} C'(x) &= A(x) \cdot B(x) \\ C'(x) &= \left(a_0 + a_1 x + a_2 x^2 + a_3 x^3\right)\left(b_0 + b_1 x + b_2 x^2 + b_3 x^3\right) \\ C'(x) &= a_0 b_0 + (a_0 b_1 + a_1 b_0) x + (a_0 b_2 + a_1 b_1 + a_2 b_0) x^2 + \\ &\quad + (a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0) x^3 + (a_1 b_3 + a_2 b_2 + a_3 b_1) x^4 \\ &\quad + (a_2 b_3 + a_3 b_2) x^5 + a_3 b_3 x^6 \end{aligned} \tag{4.8}$$

Cada elemento c'_i de $C'(x)$ é dado por:

$$\begin{aligned} c'_0 &= a_0 b_0 \\ c'_1 &= a_0 b_1 + a_1 b_0 \\ c'_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0 \\ c'_3 &= a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0 \\ c'_4 &= a_1 b_3 + a_2 b_2 + a_3 b_1 \\ c'_5 &= a_2 b_3 + a_3 b_2 \\ c'_6 &= a_3 b_3 \end{aligned} \tag{4.9}$$

Para executar a multiplicação polinomial convencional de 2 elementos em $GF(2^4)$ são necessárias 16 portas para implementar a função AND bit a bit e 25 portas para implementar as somas. Esse multiplicador possui retardo igual a $3\tau_{AON}$.

4.1.2 Redução Módulo $p(x)$

Devido a dificuldade de obter a complexidade espacial do multiplicador de Mastrovito para qualquer polinômio $p(x)$, usualmente alguns polinômios são especificados. Dessa forma é possível comparar diferentes estruturas de multiplicadores para um dado polinômio. Neste trabalho são utilizados dois polinômios: o trinômio primitivo da forma $p(x) = x^n + x + 1$, que é o polinômio mais utilizado para comparar os multiplicadores e o polinômio irredutível AOP (All One

Polynomial), $p(x) = x^n + x^{n-1} + \cdots + x + 1$, que possui algumas características interessantes [32]. Mais informações sobre polinômios irredutíveis consultar [27].

Trinômios da forma $p(x) = x^n + x + 1$

Seja o trinômio primitivo $p(x) = x^n + x + 1$, para $n = 2, 3, 4, 6, 7, 9, 10, 11, 15$. Reescrevendo novamente a equação 4.2

$$\left(c_0 + \cdots + c_{n-1}x^{n-1}\right) = \left(c'_0 + \cdots + c'_{n-1}x^{n-1}\right) \bmod (x^n + x + 1) \tag{4.10}$$

Igualando os coeficientes dos dois lados da equação 4.10, os termos c_i são obtidos:

$$\begin{aligned} c_0 &= c'_0 + c'_n \\ c_1 &= c'_1 + c'_n + c'_{n+1} \\ c_2 &= c'_2 + c'_{n+1} + c'_{n+2} \\ \vdots &= \vdots \\ c_{n-2} &= c'_{n-2} + c'_{2n-3} + c'_{2n-2} \\ c_{n-1} &= c'_{n-1} + c'_{2n-2} \end{aligned} \tag{4.11}$$

Calculando o número de portas necessárias para gerar cada termo c_i da equação 4.11 temos:

$$\begin{aligned} \#AON &= 3 + \sum_{j=1}^{n-2} 5 + 3 \\ \#AON &= 6 + 5(n-2) \\ \#AON &= 5n - 4 \end{aligned} \tag{4.12}$$

A soma total de portas do multiplicador em $GF(2^n)$, considerando a multiplicação polinomial convencional e a redução módular, resulta em:

$$\#AON = 3 \cdot 2^{n-1} + n^2 + 7n - 11 \tag{4.13}$$

Exemplo 9 *Para executar a redução módulo $p(x) = x^4 + x + 1$ da equação 4.8 são necessárias* 16 *portas AON.*

$$
\begin{aligned}
C(x) &= C'(x) \bmod x^4 + x + 1 \\
C(x) &= c'_0 + c'_1 x + c'_2 x^2 + c'_3 x^3 + c'_4 x^4 + c'_5 x^5 + c'_6 x^6 \bmod x^4 + x + 1 \quad (4.14) \\
C(x) &= (c'_0 + c'_4) + (c'_1 + c'_4 + c'_5)\, x + (c'_2 + c'_5 + c'_6)\, x^2 + (c'_3 + c'_6)\, x
\end{aligned}
$$

Cada bit de $C(x)$ *é igual a*

$$
\begin{aligned}
c_0 &= c'_0 + c'_4 \\
c_1 &= c'_1 + c'_4 + c'_5 \\
c_2 &= c'_2 + c'_5 + c'_6 \\
c_3 &= c'_3 + c'_6
\end{aligned} \qquad (4.15)
$$

Essas 16 *portas AON mais as* 41 *portas do exemplo* 10 *formam 57 portas necessárias para executar a multiplicação em* $GF(2^4)$.

Polinômios AOP $p(x) = x^n + x^{n-1} + \cdots + x + 1$

Os coeficientes de $C(x) = C'(x) \bmod p(x)$ para o polinômio $p(x) = x^n + x^{n-1} + \cdots + x + 1$ são:

$$
\begin{aligned}
c_0 &= c'_0 + c'_n + c'_{n+1} \\
c_1 &= c'_1 + c'_n + c'_{n+2} \\
\vdots &= \vdots \\
c_{n-3} &= c_{n-3} + c_n + c_{2n-2} \\
c_{n-2} &= c'_{n-2} + c'_n \\
c_{n-1} &= c'_{n-1} + c'_n
\end{aligned} \qquad (4.16)
$$

A lei de formação para a a equação 4.16 é dada abaixo:

$$
c_j = \begin{cases} c'_j + c'_n + c'_{n+j+1} & 0 \le j < n-2 \\ c'_j + c'_n & j \ge n-2 \end{cases} \qquad (4.17)
$$

Assim o número de portas AON necessárias para implementar a redução módular é:

$$
\#AON = \sum_{j=0}^{n-3} \left(2^{3-1} + 1\right) + \sum_{j=n-2}^{n-1} \left(2^{2-1} + 1\right)
$$

$$
\begin{aligned}
&= 5(n-2)+3\cdot 2 \qquad (4.18)\\
\#AON &= 5n-4
\end{aligned}
$$

O total de portas para o multiplicador em $GF(2^n)$ para $p(x) = x^n + x^{n-1} + \cdots + x + 1$ é:

$$\#AON = 3\cdot 2^{n-1} + n^2 + 7n - 11 \qquad (4.19)$$

Para executar a redução módulo $p(x) = x^4 + x^3 + x^2 + x + 1$ da equação 4.8 são necessárias 16 portas AON.

$$
\begin{aligned}
C(x) &= C'(x) \bmod x^4 + x^3 + x^2 + x + 1 \qquad (4.20)\\
C(x) &= c'_0 + c'_1 x + c'_2 x^2 + c'_3 x^3 + c'_4 x^4 + c'_5 x^5 + c'_6 x^6 \bmod x^4 + x^3 + x^2 + x + 1\\
C(x) &= (c'_0 + c'_4) + (c'_1 + c'_4)\, x + (c'_2 + c'_4)\, x^2 + (c'_3 + c'_4)\, x^3
\end{aligned}
$$

Essas 16 portas AON mais as 41 portas do exemplo 10 totalizam 57 portas necessárias para executar a multiplicação em $GF(2^4)$.

O retardo necessário para implementar o multiplicador de Mastrovito para portas AON com *fan-in* ilimitado é:

$$
\begin{aligned}
AND \text{ bit a bit} &= 1\tau_{AON}\\
\text{Multiplicação Polinomial} &= 2\tau_{AON}\\
\text{Redução Módulo} &= 2\tau_{AON}
\end{aligned}
$$

Assim o retardo do Multiplicador de Mastrovito é de $5\tau_{AON}$, em que τ_{AON} é o retardo correspondente a uma porta AON. Note que, embora sejam utilizados dois polinômios específicos neste trabalho, é possível utilizar essa técnica para obter um multiplicador em $GF(2^n)$ utilizando qualquer polinômio irredutível.

4.2 Arquitetura 1

A primeira arquitetura proposta utiliza a mesma técnica descrita por Mastrovito para elaboração do multiplicador, sendo que ao invés de utilizar portas AON são utilizadas portas de limiar linear.

4.2.1 Multiplicação Polinomial

Novamente utilizando a equação 4.4, são necessárias n^2 portas de limiar linear para simular a porta AND. Como visto anteriormente na seção *2.3*, circuitos de n variáveis de 2 camadas necessitam de $\left\lceil \frac{n}{2} \right\rceil + 1$ portas de limiar linear com *fan-in* ilimitado. Seja $\#TG$ o número de portas de limiar linear. O número de portas de limiar linear para implementar a multiplicação polinomial (para $n > 2$) é calculado a seguir:

$$\#TG = 2 \cdot \sum_{i=2}^{n} \left(\left\lceil \frac{i}{2} \right\rceil + 1 \right) - \left(\left\lceil \frac{n}{2} \right\rceil + 1 \right) \tag{4.21}$$

Caso n seja par, o número de portas é dado por:

$$\begin{aligned} \#TG &= 2 \cdot \sum_{i=2}^{n} \left\lceil \frac{i}{2} \right\rceil + 2(n-1) - \frac{n}{2} - 1 \\ &= 2 \cdot \left(\frac{n^2}{4} + \frac{n}{2} - 1 \right) + 2(n-1) - \frac{n}{2} - 1 \\ &= \frac{n^2}{2} + \frac{5n}{2} - 5 \end{aligned} \tag{4.22}$$

Para n ímpar temos:

$$\begin{aligned} \#TG &= 2 \cdot \sum_{i=2}^{n} \left\lceil \frac{i}{2} \right\rceil + 2(n-1) - \frac{n}{2} - \frac{1}{2} - 1 \\ &= 2 \cdot \left(\frac{n^2}{4} + \frac{n}{2} - \frac{3}{4} \right) + \frac{3n}{2} - \frac{7}{2} \\ \#TG &= \frac{n^2}{2} + \frac{5n}{2} - 5 \end{aligned} \tag{4.23}$$

Portanto, para qualquer n:

$$\#TG = \frac{n^2}{2} + \frac{5n}{2} - 5 \tag{4.24}$$

O retardo do multiplicador polinomial é $1\tau_{TG}$ para implementar a porta AND e $2\tau_{TG}$ para as somas.

Exemplo 10 *Sejam $A(x)$ e $B(x)$ dois elementos em $GF(2^4)$ para $p(x) = x^4 + x + 1$. A multiplicação convencional $C(x) = A(x) \cdot B(x)$ é realizada da seguinte forma:*

Primeiramente é realizado um AND bit a bit entre os coeficientes de $A(x)$ e $B(x)$ utilizando redes neurais discretas. A porta AND é implementada utilizando portas de limiar linear como na figura 2.3. Para $GF(2^4)$, são necessárias 16 *portas (ver figura 4.2)*

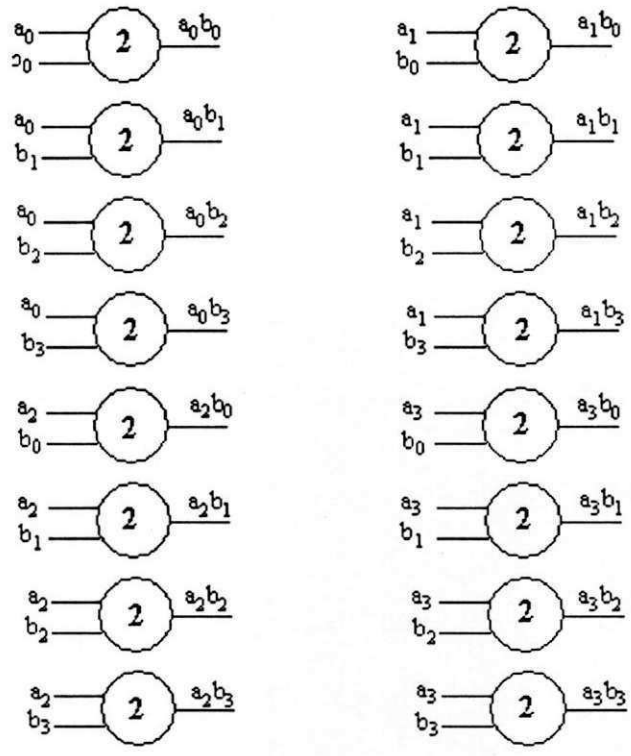

Figura 4.2: Implementação neural da função AND bit a bit dos elementos de $A(x)$ e $B(x)$

Os bits c_i' descritos na equação 4.9 podem ser implementados com portas de limiar linear com ilustrados nas figuras 4.3,4.4,4.5,4.6,4.7,4.8 e 4.9

4.2.2 Redução módulo

A redução módulo $p(x)$ utiliza a equação 4.11.

Trinômios $p(x) = x^n + x + 1$

$$\begin{aligned} \#TG &= 2 + \sum_{j=1}^{n-2} 3 + 2 = \\ &= 2 + 3(n-2) + 2 \end{aligned}$$

Figura 4.3: Bit 1 do Multiplicador Polinomial Ordinário

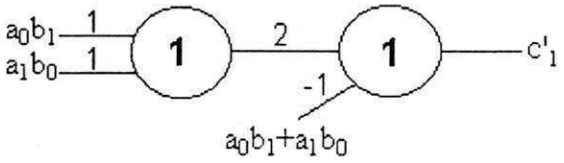

Figura 4.4: Bit 2 do Multiplicador Polinomial Ordinário

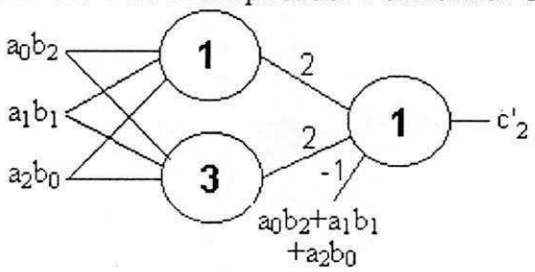

Figura 4.5: Bit 2 do Multiplicador Polinomial Ordinário

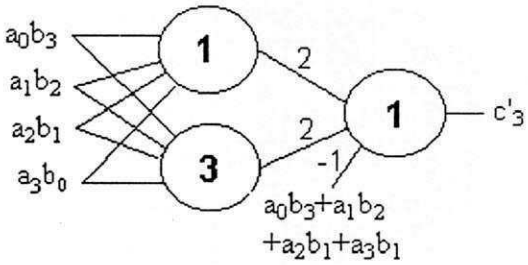

Figura 4.6: Bit 3 do Multiplicador Polinomial Ordinário

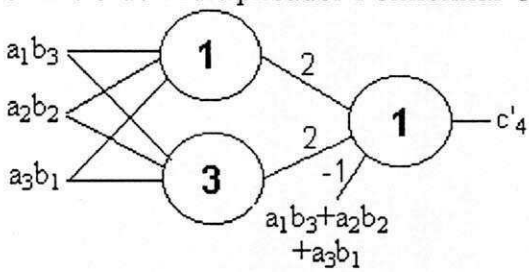

Figura 4.7: Bit 4 do Multiplicador Polinomial Ordinário

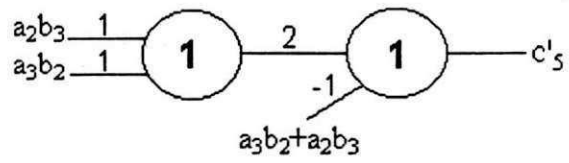

Figura 4.8: Bit 5 do Multiplicador Polinomial Ordinário

Figura 4.9: Bit 6 do Multiplicador Polinomial Ordinário

$$\#TG = 3n - 2$$

O número total de portas para o multiplicador em corpo finito utilizando redes neurais discretas é:

$$\#TGs = \frac{3n^2}{2} + \frac{11n}{2} - 7 \tag{4.25}$$

Polinômios AOP $p(x) = x^n + x^{n-1} + \cdots + x + 1$

Através da equação 4.16, a complexidade espacial é obtida:

$$\begin{aligned} \#TG &= \sum_{j=0}^{n-3} \left\lceil \frac{3}{2} + 1 \right\rceil + \sum_{j=n-2}^{n-1} \left\lceil \frac{2}{2} + 1 \right\rceil \\ &= 3(n-2) + 2 \cdot 2 \\ \#TG &= 3n - 2 \end{aligned} \tag{4.26}$$

O retardo referente a redução módulo é $2\tau_{TG}$.

Exemplo 11 *Para implementar a redução módulo* $p(x) = x^n + x^{n-1} + \cdots + x + 1$ *são utilizadas portas de limiar linear, como ilustrado nas figuras 4.10,4.11,4.12 e 4.13.*

4.3 Arquitetura 2

Nessa arquitetura, a multiplicação polinomial e a redução módulo $p(x)$ é executada em uma única etapa. A complexidade temporal é reduzida de $5\tau_{TG}$ para $3\tau_{TG}$.

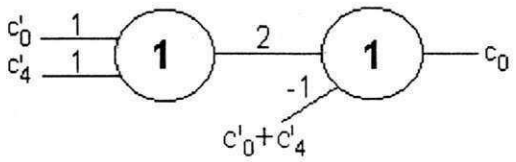

Figura 4.10: Implementação neural bit c_0

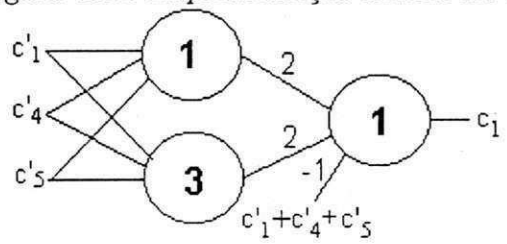

Figura 4.11: Implementação neural bit c_1

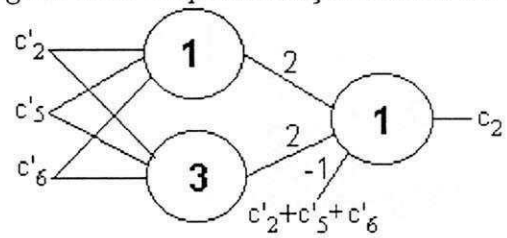

Figura 4.12: Implementação neural bit c_2

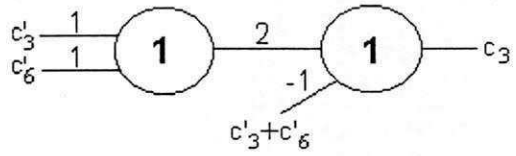

Figura 4.13: Implementação neural bit c_3

A complexidade espacial do multiplicador depende da escolha do polinômio.

Trinômios da forma $p(x) = x^n + x + 1$

Reescrevendo a equação 4.11 temos:

$$C(x) = (c'_0 + c'_n) + \sum_{j=1}^{n-2} \left(c'_j + c'_{j+n-1} + c'_{j+n}\right) x^j + \left(c'_{n-1} + c'_{2n-2}\right) x^{n-1} \tag{4.27}$$

Substituindo c'_j pelos seus respectivos valores da equação 4.4:

$$\begin{aligned} C(x) &= \left(a_0b_0 + \sum_{i=1}^{n-1} a_ib_{n-i}\right) + \sum_{j=1}^{n-2}\left(\sum_{i=0}^{j} a_ib_{j-i} + \right. \\ &\quad \left. + \sum_{i=j}^{n-1} a_ib_{j+n-1-i} + \sum_{i=j+1}^{n-1} a_ib_{j+n-i}\right)x^j + \\ &\quad + \left(\sum_{i=0}^{n-1}(a_ib_{n-1-i}) + a_{n-1}b_{n-1}\right)x^{n-1} \end{aligned} \tag{4.28}$$

O número de elementos da equação 4.11 para cada coeficiente c_i é dado por:

$$\begin{aligned} \#c_0 &= 1+n-1=n \\ \#c_j &= (j+1)+2n-j-n+1-1+2n- \\ &\quad -j-n-1 \\ &= 2n-j \qquad \text{para } j=1,2\ldots,n-2 \\ \#c_{n-1} &= n+2n-2n+2-1=n+1 \end{aligned} \tag{4.29}$$

Assim, o número de portas necessárias é dado por:

$$\#TG = \left\lceil\frac{n}{2}\right\rceil + 1 + \sum_{j=1}^{n-2}\left(\left\lceil\frac{2n-j}{2}\right\rceil + 1\right) + \left\lceil\frac{n+1}{2}\right\rceil + 1 \tag{4.30}$$

$$\begin{aligned} \#TGs &= \left\lceil\frac{\#c_0}{2}+1\right\rceil + \sum_{j=1}^{n-2}\left(\left\lceil\frac{\#c_j}{2}\right\rceil + 1\right) + \\ &\quad + \left\lceil\frac{\#c_{n-1}}{2}+1\right\rceil \\ &= \left\lceil\frac{n}{2}+1\right\rceil + \sum_{j=1}^{n-2}\left(\left\lceil\frac{2n-j}{2}\right\rceil + 1\right) + \\ &\quad + \left\lceil\frac{n+1}{2}+1\right\rceil \\ &= n+3+\sum_{j=1}^{n-2}\left(\left\lceil\frac{2n-j}{2}\right\rceil + 1\right) \\ &= n+3+(n-2)+\sum_{j=1}^{n-2}\left\lceil\frac{2n-j}{2}\right\rceil \\ &= 2n+1+\sum_{j=1}^{n-2}\left\lceil\frac{2n-j}{2}\right\rceil \end{aligned} \tag{4.31}$$

Para n par temos:

$$\begin{aligned} \#TGs &= 2n+1+\left(\frac{3n^2}{4}-n-1\right) \\ &= \frac{3n^2}{4}+n \end{aligned} \tag{4.32}$$

Para n ímpar temos:

$$\begin{aligned} \#TGs &= 2n+1+\left(\frac{3n^2}{4}-n-\frac{3}{4}\right) \\ &= \frac{3n^2}{4}+n+\frac{1}{4} \end{aligned} \tag{4.33}$$

O total de portas necessárias para a multiplicação em $GF(2^n)$ é dado por:

$$\#TG = \begin{cases} \frac{7}{4}n^2+n & n \text{ par} \\ \\ \frac{7}{4}n^2+n+\frac{1}{4} & n \text{ ímpar} \end{cases} \tag{4.34}$$

Exemplo 12 *Sejam $A(x)$ e $B(x)$ dois elementos em $GF(2^4)$ para $p(x) = x^4 + x + 1$. O multiplicador $C(x) = A(x) \cdot B(x) \bmod p(x)$ é construído a partir da equação 4.27 da seguinte forma:*

$$\begin{aligned} C(x) &= (a_0b_0 + a_1b_3 + a_2b_2 + a_3b_1) + (a_0b_1 + a_1b_0 + a_1b_3 + a_2b_2 + \\ &\quad + a_3b_1 + a_2b_3 + a_3b_2)x + (a_0b_2 + a_1b_1 + a_2b_0 + a_2b_3 + a_3b_2 + a_3b_3)x^2 \\ &\quad + (a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0 + a_3b_3)x^3 \end{aligned} \tag{4.35}$$

Cada bit de $C(x)$ é dado por:

$$\begin{aligned} c_0 &= a_0b_0 + a_1b_3 + a_2b_2 + a_3b_1 \\ c_1 &= a_0b_1 + a_1b_0 + a_1b_3 + a_2b_2 + a_3b_1 + a_2b_3 + a_3b_2 \\ c_2 &= a_0b_2 + a_1b_1 + a_2b_0 + a_2b_3 + a_3b_2 + a_3b_3 \\ c_3 &= a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0 + a_3b_3 \end{aligned} \tag{4.36}$$

Implementando a equação 4.36 com portas de limiar linear obtém-se as figuras 4.14,4.15,4.16 e 4.17 :

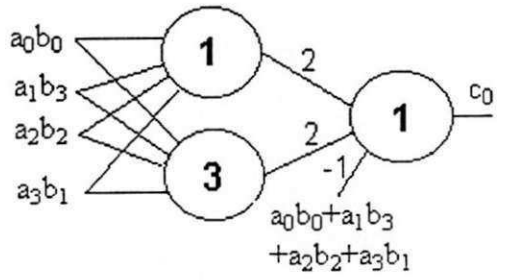

Figura 4.14: Implementação neural bit c_0

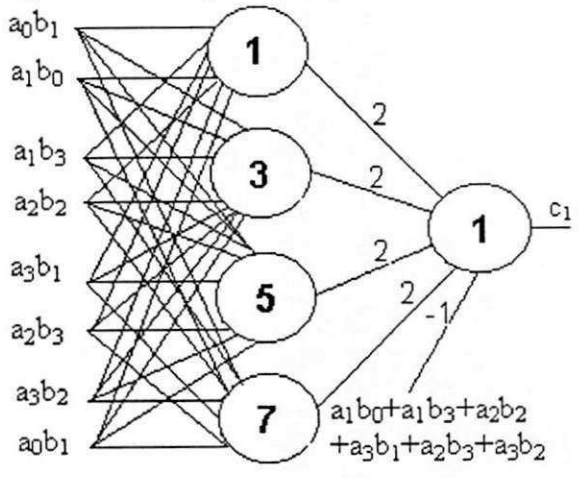

Figura 4.15: Implementação neural bit c_1

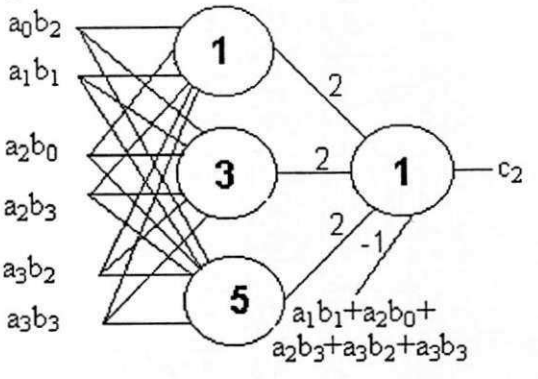

Figura 4.16: Implementação neural bit c_2

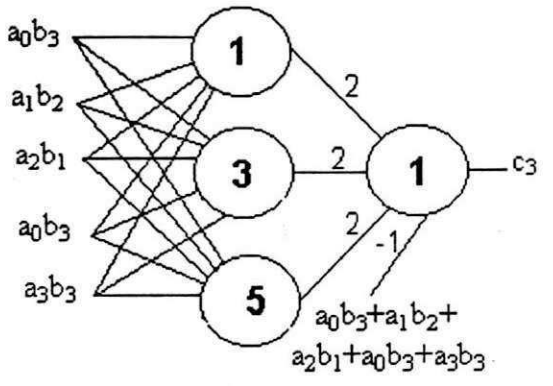

Figura 4.17: Implementação neural bit c_3

O retardo é fixo e igual a $3\tau_{TG}$.

Polinômios AOP $p(x) = x^n + x^{n-1} + \cdots + x + 1$

Para polinômios irredutíveis do tipo $p(x) = x^n + x^{n-1} + \cdots + x + 1$ a saída c_j é dada por:

$$c_j = \begin{cases} c'_j + c'_n + c'_{n+j+1} & 0 \leq j < n-2 \\ c'_j + c'_n & j \geq n-2 \end{cases} \tag{4.37}$$

O número de parcelas da soma de c'_j segundo a equação 4.9 é:

$$\begin{array}{rl} (j+1) \text{ elementos} & \text{para } j < n \\ (2n-j-1) \text{ elementos} & \text{para } j \geq n \end{array}$$

Portanto, para cada c_j, o número de parcelas' é:

$$\begin{aligned} \#c_j &= (j+1) + (n-1) + (n-j-2) \ , 0 \leq j < n-2 \\ &= 2n-2 \\ \#c_j &= (j+1) + (n-1) \ , j \geq n-2 \\ &= j+n \end{aligned} \tag{4.38}$$

O número de portas de limiar linear é:

$$\begin{aligned} \#TG &= \sum_{j=0}^{n-3} \left\lceil \frac{2n-2}{2} + 1 \right\rceil + \sum_{j=n-2}^{n-1} \left\lceil \frac{j+n}{2} + 1 \right\rceil \\ &= \sum_{j=0}^{n-3} \lceil n \rceil + \sum_{j=n-2}^{n-1} \left\lceil \frac{j+n}{2} + 1 \right\rceil \end{aligned}$$

n	AON	TG 1	TG 2
2	13	10	9
3	31	28	19
4	57	39	32
5	101	58	49
6	163	80	69
7	279	105	93
8	517	135	184
9	901	164	151
10	1695	198	185
11	3259	235	223
12	6369	281	321
13	12600	335	389
14	24679	383	373
15	98623	413	409
16	33411	481	584

Tabela 4.1: Comparação do número de portas das Arquiteturas de Multiplicadores em $GF(2^n)$

4.5 Conclusão

Na implementação de multiplicadores utilizando lógica tradicional, geralmente é encontrado na literatura circuitos com *fan-in* limitado. Esse procedimento é usado para diminuir o tamanho do circuito, já que como foi visto, o tamanho do circuito para o multiplicador de Mastrovito é exponencial dado por $3 \cdot 2^{n-1} + n^2 + 7n - 11$. Como mostrado em [25], ao tornar a profundidade do circuito ilimitada, é possível reduzir o tamanho de exponencial para polinomial. Dessa forma é possível obter uma arquitetura do multiplicador de Mastrovito que forneça um número menor de portas AON, mas, em compensação, o retardo não será fixo. Como em algumas aplicações a velocidade é primordial, o uso de circuitos com profundidade que cresce com o aumento do número de entradas não é desejável. Assim, deve haver um compromisso entre o tamanho e a profundidade de forma

que sejam obtidos circuitos mais eficientes para determinada aplicação.

No entanto, ao utilizar as arquiteturas apresentadas neste trabalho para a multiplicação, o tamanho do circuito não aumenta exponencialmente com o aumento do número de entradas. Dessa forma, é possível obter *circuitos rápidos* e com *baixo número de portas*, o que não é sempre possível utilizando a lógica tradicional.

Capítulo 5

Considerações Finais e Perspectivas

Neste trabalho foi apresentada duas novas arquiteturas para multiplicadores paralelos em corpos finitos. Esses multiplicadores foram analizados sob o ponto de vista da complexidade temporal e espacial.

Devido a rapidez das aplicações que utilizam o multiplicador em corpo finito, foi enfatizada a minimização do retardo. Ao invés de circuitos ineficientes com relação ao tamanho (como encontrado nas portas tradicionais), devido à potencialidade das portas de limiar linear foram obtidos resultados bastante satisfatórios. Para um multiplicador de Mastrovito em $GF(2^{16})$, a utilização de portas de limiar linear diminui em cerca de 70 vezes o número de portas necessárias para o multiplicador quando comparado ao mesmo multiplicador com portas tradicionais. Outra arquitetura proposta diminui o número de camadas do multiplicador, e consequentemente o seu retardo, de $5\tau_{TG}$ para $3\tau_{TG}$, independente do tamanho do corpo.

Essas características tornam o multiplicador em $GF(2^n)$ utilizando redes neurais discretas bastante adequado para operações em criptografia, cuja operações geralmente utilizam $n > 128$. Outras aplicações podem se tornar mais eficientes, já que o multiplicador proposto é rápido e requer uma área de chip menor. Foi também implementado, através de redes neurais discretas, um circuito que gera o logaritmo de Zech de um elemento, sem necessidade do uso de tabelas.

Assim, supondo que o custo de uma porta de limiar linear equivalente ao de

uma porta tradicional, a utilização de redes neurais discretas em multiplicadores em corpos finitos é justificada.

A seguir são feitas algumas sugestões para continuação das atividades de pesquisa:

- Utilizar métodos computacionais para encontrar polinômios $p(x)$ ótimos de forma que a complexidade espacial seja reduzida;
- Utilizar a construção de 3 camadas de forma que o tamanho do circuito seja minimizado;
- Implementação física de uma porta de limiar linear para analisar outros fatores (dissipação, facilidade de implementação,etc.);
- Projeto de um codificador/decodificador inteiramente construido por portas de limiar linear;

Apêndice A

Complexidade

Fornecendo hardware suficiente (ex: número de portas neurais) é possível executar qualquer mapeamento de função através de circuitos com portas AND, OR e NOT (também chamado de circuitos AON). Por exemplo, qualquer função booleana com n variáveis pode ser implementada com no máximo $2^{n-1}+1$ variáveis e com até 2 camadas. Entretanto, essa implementação necessitaria de uma grande quantidade de recursos, mesmo para um pequeno número de entradas. Na realidade, o objetivo do projeto de circuitos é minimizar a quantidade de hardware utilizado e o seu tempo de execução.

Assim, temos a seguinte questão: Dada uma certa função, qual a quantidade mínima de *hardware* necessária para sua implementação? Para responder essa questão é necessário conhecer a complexidade da função a ser computada bem como qual arquitetura utilizar para implementar o circuito, já que de uma arquitetura para outra há diferenças na complexidade obtida.

Um algoritmo é um método para resolver uma determinada classe de problemas. A complexidade de um algoritmo é o custo, medido em tempo de execução, armazenamento, ou qualquer unidade relevante, usado para resolver esse problema. Alguns problemas necessitam de um tempo muito longo, outros podem ser executados rapidamente. O estudo da quantidade de esforço computacional necessária para executar certo tipo de computação é o estudo da complexidade computacional.

Várias medidas de complexidade, como tamanho, retardo, *fan-in*, dentre outras, podem ser definidas para um determinado modelo de circuito. Antes de

discutir alguns pontos de complexidade de circuitos, é necessário introduzir algumas terminologias e notações para facilitar entendimentos futuros.

A.1 Notações Assintóticas

Nessa seção são discutidas as taxas de crescimento de diferentes funções e introduzidos alguns símbolos assintóticos que são utilizados para descrever essas taxas de crescimento. A razão para utilização desses símbolos é a padronização de uma linguagem que torne possível comparar as complexidades de diferentes algoritmos que executem o mesmo trabalho.

Por exemplo, analisando a performance do algoritmo de inversão de matrizes quadradas não singulares, chegamos a seguinte questão: Como deve ser medida a velocidade? Mais comumente é dito, por exemplo, que se a matriz for $n \times n$ o algoritmo irá ser executado em, por exemplo, $16,8n^3$. Portanto, se em um determinado computador o algoritmo de inversão necessitará de de 1 minuto para computar uma matriz 100×100 então numa matriz 200×200 irá necessitar cerca de $2^3 = 8$ vezes o tempo anterior, ou seja 8 minutos. A constante $16,8$ não foi usada nesse exemplo. Apenas o fato que o tempo de execução cresce na terceira potência do tamanho da matriz é relevante. Assim torna-se necessário utilizar uma linguagem que permita informar que o tempo de computação cresce 'na ordem de n^3 ,' 'no máximo tão rápido quanto n^3', ou 'no mínimo tão rápido quanto $n^5 \log n$', etc.

Para comparar a taxa de crescimento das funções novos símbolos são introduzidos. Sejam $f(x)$ e $g(x)$ duas funções de x. Cada símbolo pretende comparar quão rapidamente cresce f e g. Se $f(x) = o(g(x))$, então é porque f cresce mais lentamente que g quando x é muito alto. Formalmente temos:

$$f(x) = o(g(x))(x \to \infty) \text{ se } lim_{x\to\infty}\frac{f(x)}{g(x)} \text{ existe e é igual a 0.} \qquad \text{(A.1)}$$

Alguns exemplos:

(a) $x^2 = o(x^5)$

(b) $\sin x = o(x)$

(c) $23 \log x = o(x^{0,2})$

O segundo símbolo do vocabulário assintótico é *'O '*. Se $f(x) = O(g(x))$ significa que f certamente não cresce a uma taxa mais rápida que g. A função f cresce na mesma taxa ou mais lento. Formalmente:

$$f(x) = O(g(x))(x \to \infty) \text{ se } \exists C; x_0 \text{ tal que } |f(x)| < Cg(x) \, (\forall x > x_0) \quad \text{(A.2)}$$

O qualificador $x \to \infty$ pode ser normalmente omitido pois o interesse é em valores altos das variáveis envolvidas.

O terceiro símbolo da linguagem assintótica é θ, definido como

$$f(x) = \theta(g(x)) \quad \text{(A.3)}$$

se existe constantes $c_i \neq 0$ e $c_2 \neq 0$, x_0 tal que para todo $x > x_0$

$$c_1 g(x) < f(x) < c_2 g(x) \quad \text{(A.4)}$$

Se f e g crescem na mesma taxa, apenas as constantes multiplicativas são incertas. Alguns exemplos do uso de θ são:

(a) $(x+1)^2 = \theta(3x^2)$

(b) $(x^2 + 5x) / (x^3 + 7x + 2) = \theta(\frac{1}{x})$

(c) $23 \log x = o(x^{0,2})$

O último símbolo do conjunto assintótico é Ω, que pode ser entendido como a negação de o. Ou seja, $f(x) = \Omega(g(x))$ significa que não é verdade que $f(x) = o(g(x))$. A exata definição de Ω (para diferenciar de o) é que se $f = o(g)$ significa que $f/g \to \infty$, enquanto se $f = \Omega(g)$ significa que f/g não se aproxima de zero e sim de uma sequência infinita de x tendendo a ∞. Ou seja, existe $\varepsilon > 0$ tal que $\frac{|f|}{|g|} > \varepsilon$. Mais formalmente

$$f(x) = \Omega(g(x)) \text{ se existe } \varepsilon > 0 \quad \text{(A.5)}$$

A.2 *Fan-in/Fan-out*

O estudo teórico de *fan-in/fan-out* não fixo para portas tradicionais mostra que ao limitar o *fan-in/fan-out* torna-se necessário aumentar o número de portas ou o retardo. No trabalho proposto não há restrição no *fan-in/fan-out* dos circuitos de

limiar linear utilizados. Dessa forma é possível determinar o máximo de recursos que pode ser poupado com respeito a complexidade espacial e temporal.

Como redes neurais são caracterizadas por um massivo processamento paralelo, assumir modelo de circuito de limiar linear com *fan-in/fan-out* não fixo é bastante razoável.

A.3 Complexidade Espacial

A quantidade de hardware necessária para implementação física de circuitos corresponde a complexidade espacial ou tamanho do circuito. É óbvio que a complexidade espacial para executar uma determinada função depende do tipo de portas utilizadas para implementação. Intuitivamente, quanto mais poderosa for o tipo de portas utilizadas melhor será a performance do circuito. Por exemplo, um circuito C_n com tamanho $\theta(n^2)$ necessitará de mais portas do que um circuito C_n com tamanho $\theta(n \log n)$ para n suficientemente grande. O tamanho de um circuito C_n é dito polinomial se sua complexidade espacial $S(n)$ é limitada por uma potência fixa de n, ou mais formalmente, existe uma constante $k > 0$ tal que $S(n) = O\left(n^k\right)$. Caso o seu tamanho seja $\Omega\left(2^{n^\epsilon}\right)$ para um valor fixo $\epsilon > 0$, então o circuito possui tamanho exponencial.

Em geral, para saber se função tem tamanho polinomial ou exponencial é necessário conhecer a característica do tipo de porta utilizada no circuito e a restrição imposta em outra medição de complexidade. Por exemplo, a função paridade pode ser computada através de circuitos AON com profundidade $O(\log n)$, tamanho $O(n)$ e *fan-in* limitado. Por outro lado, ela necessita de tamanho exponencial para ser executado com 2 camadas e *fan-in* ilimitado. Caso utilize circuitos de limiar linear com *fan-in* ilimitado, ela pode ser executada com circuito de 2 camadas e tamanho $O(n)$.

A.4 Complexidade Temporal (Profundidade)

A complexidade temporal ou a profundidade de um circuito corresponde ao tempo necessário para computar as saídas em paralelo, determinando assim a velocidade do circuito. Um circuito C_n possui profundidade limitada ou retardo constante se

existe um inteiro fixo d tal que, para qualquer n, a profundidade de C_n é limitada abaixo de d. Caso contrário, o circuito C_n possui profundidade ilimitada.

Apêndice B

Corpos Finitos

Neste Apêndice algumas definições básicas e propriedades dos corpos finitos relevantes a este trabalho são revistas. Algumas operações aritméticas em corpos finitos também são apresentadas.

B.1 Grupos

Um grupo é definido pela combinação de um conjunto e de uma operação binária. A operação binária * é definida como uma operação que leva dois operandos a e b e produz um único resultado $c = a * b$. Um conjunto é dito fechado sobre a operação binária $*$ se, para qualquer dois elementos a e b pertencente ao conjunto, $c = a * b$ é também um elemento do conjunto.

O conjunto e uma operação binária satisfazem a exigência de um grupo $G(S, +)$ se

1) O conjunto S é fechado sobre a operação binária $+$ e essa operação satisfaz a propriedade associativa

$$a + (b + c) = (a + b) + c \tag{B.1}$$

2) Existe um elemento e em S tal que qualquer elemento a em S satisfaz

$$a + e = e + a = a \tag{B.2}$$

Esse elemento é chamado de identidade.

3) Existe um elemento a' para cada elemento a em S tal que

$$a + a' = e = a' + a \tag{B.3}$$

Esse elemento a' é chamado de inverso de a.

Por exemplo, o conjunto R^* (números reais sem o elemento zero) e a operação binária soma (ou multiplicação) formam um grupo. O conjunto de todos os números reais é fechado sobre adição e multiplicação (quaisquer dois números reais somados ou multiplicados produzem outro número real). Adição e multiplicação são associativa no conjunto dos números reais. O elemento identidade da adição é 0 enquanto na multiplicação é 1.

Um grupo é chamado de comutativo ou abeliano se

$$a + b = b + a \tag{B.4}$$

Um grupo é chamado de cíclico se existe um elemento no conjunto cuja potência constitui todos os elementos diferente de zero do conjunto.

B.2 Corpos

Um corpo é definido pela combinação de um conjunto e duas operações binárias + e *. Se um conjunto e duas operações binárias satisfizerem as seguintes existências eles formam um corpo:

1) Se o conjunto é fechado sobre + e forma um grupo comutativo;

2) Se todos os elementos diferentes de zero do conjunto são fechados sobre * e formam um grupo comutativo;

3) Se o conjunto e as duas operações + e * satisfazem a propriedade distributiva

$$a * b + a * c = a * (b + c) \tag{B.5}$$

Por exemplo, o conjunto $\{0, 1, 2, 3, 4, 5, 6\}$ com adição e multiplicação módulo 7 forma um corpo, pois:

1) O conjunto é fechado sobre adição módulo 7

$$\begin{aligned} 3 + 4 &= 7 \bmod 7 = 0 \\ 2 + 6 &= 8 \bmod 7 = 1 \end{aligned}$$

Note que na adição módulo X, $a+(b+c)$ é sempre igual a $(a+b)+c$ e $a+b=b+a$

2) O conjunto de todos os elementos diferentes de zero é fechado sobre a multiplicação módulo 7

$$\begin{aligned} 3*4 &= 12 \bmod 7 = 5 \\ 6*5 &= 30 \bmod 7 = 2 \end{aligned}$$

Note que na multiplicação módulo 7 $a*(b*c)$ é sempre igual a $(a*b)*c$ e $a*b=b*a$

3) Finalmente, a propriedade distributiva

$$a*(b+c)=a*b+a*c \tag{B.6}$$

Por exemplo

$$3*(4+5)=3*9=27 \bmod 7=6$$

e

$$3*4+3*5=12+15=27 \bmod 7=6$$

B.3 Corpos Finitos

Corpos Finitos são denotados por $GF(q)$, onde q é algum número primo positivo e o conjunto $S\{0,1,2,...,q-1\}$ e as operações binárias + e * são adição e multiplicação módulo q respectivamente. Como na comunicação digital são utilizados bits, estamos interessados na subclasse de corpos finitos de q tal que q seja potência de 2. Isso é considerado principalmente porque em $GF(2)$ a representação em corpo finito é mapeada convenientemente no domínio digital.

Utilizando a notação $GF(q)$, a subclasse será denotada como $GF(2^n)$. Portanto $GF(2), GF(4), GF(8)$, etc são corpos finitos $GF(2^n)$.

O campo de Galois $GF(q)$, em que q é primo, é chamado de corpos primos. Qualquer corpo no qual q seja primo pode ter adição e multiplicação módulo q como suas duas operações binárias. O campo de Galois, $GF(q^n)$, é chamado de extensão de corpo com $GF(q)$ sendo o corpo de base de $GF(q^n)$.

O campo de Galois $GF(2)$ é chamado de corpo binário. Por exemplo, $GF(2)$ contém o conjunto $\{0,1\}$ e utiliza adição e multiplicação módulo 2.

Adição *mod* 2	Multiplicação *mod* 2
$0+0=0$	$0*0=0$
$0+1=1$	$0*1=0$
$1+0=1$	$1*0=0$
$1+1=0$	$1*1=1$

Note que a adição módulo 2 é equivalente a operação XOR e a multiplicação módulo 2 é equivalente a porta AND.

O polinômio em $GF(2)$ pode ser representado como coeficientes de $GF(2)$

$$a_0 + a_1x^1 + a_2x^2 + \cdots + a_nx^n \tag{B.7}$$

em que $a_i = 0$ ou 1 para $0 \leq i \leq n$.

Um polinômio é de grau n se não existe nenhum termo x no polinômio com potência maior que n.

Propriedades

Corpos finitos podem ser obtidos a partir de anéis polinomiais. Seja $F[x]$ um anel polinomial sobre o corpo F, escolhendo qualquer polinômio $p(x)$ pertencente a $F[x]$

Teorema 2 *Para qualquer polinômio mônico em F, o anel dos polinômios módulo $p(x)$ é o conjunto de todos os polinômios com grau menor que $p(x)$ junto com adição e multiplicação polinomial módulo $p(x)$. Se o polinômio $p(x)$ for primo então o anel de polinômios é um corpo.*

Dessa forma encontrando um polinômio primo $p(x)$ de grau n em $GF(q)$, pode-se construir um corpo de Galois com q^n elementos.

Antes de introduzir $GF(2^n)$, algumas definições são necessárias. Um polinômio $p(x)$ de grau n sobre $GF(2^n)$ é um polinômio da forma:

$$p(x) = p_0 + p_1x + p_2x^2 + \cdots p_nx^n \tag{B.8}$$

em que os coeficientes p_i são elementos de $GF(2) = \{0,1\}$. Polinômios em $GF(2)$ podem ser adicionados, subtraidos, multiplicados e divididos na forma usual.

Uma propriedade útil dos polinômios em $GF(2^n)$ é que:

$$p^2(x) = (p_0 + p_1 x + ... + p_n x_n)^2 = p_0 + p_1 x^2 + ... + p_n x^{2n} = p(x^2) \qquad \text{(B.9)}$$

A noção de polinômio irredutível é agora introduzida.

Definição 2 *Um polinômio $p(x)$ sobre $GF(2)$ de grau n é irredutível se $p(x)$ não for divisível por nenhum polinômio sobre $GF(2)$ de grau menor que n e maior que zero.*

Para gerar a extensão do corpo $GF(2^n)$, um polinômio $p(x)$ mônico e irredutível de grau n em $GF(2)$ é escolhido. O conjunto de 2^n polinômios de grau menor que n em $GF(2)$ é formado e chamado de F. Pode ser provado [28] que, quando a adição e multiplicação desses polinômios são tomados módulo $p(x)$, o conjunto F forma um corpo com 2^n elementos, denotado por $GF(2^n)$. Note que $GF(2^n)$ é uma extensão de $GF(2)$ de modo análogo ao modo que os números complexos são formados a partir dos números reais, nesse caso $p(x) = x^2 + x + 1$. Os elementos do corpo são representados pelo conjunto $\{0, 1, x, 1+x\}$. Na tabela abaixo são mostrados diversos tipos de notações para esses elementos.

Notação Polinomial	Notação Binária	Notação Inteira	Notação Exponencial
0	00	0	0
1	01	1	x^0
x	10	2	x^1
$x+1$	11	3	x^2

Se $p(x)$ é um polinômio primitivo de grau n sobre $GF(2)$ então ele divide $x^m + 1$, em que $m = 2^n - 1$. A tabela a seguir mostra um lista de polinômios primitivos sobre $GF(2)$ com número mínimo de termos.

Grau n	Polinômio Primitivo
3	$1+x+x^3$
4	$1+x+x^2$
5	$1+x^2+x^5$
6	$1+x+x^6$
7	$1+x+x^7$
8	$1+x^2+x^3+x^4+x^8$
9	$1+x^4+x^9$
10	$1+x^3+x^{10}$
11	$1+x^2+x^{11}$
12	$1+x+x^4+x^6+x^{12}$
13	$1+x+x^3+x^4+x^{13}$
14	$1+x+x^6+x^{10}+x^{14}$
15	$1+x+x^{15}$
16	$1+x+x^3+x^{12}+x^{16}$

Teorema 3 *Para cada elemento não nulo $\alpha \in GF(q)$, $ord(\alpha)$ divide $q-1$.*

Podemos facilmente observar isso pelo algoritmo de Euclides para divisão, em que $t = ord(\alpha)$

$$\begin{aligned} q-1 &= at+r \\ \alpha^{q-1} &= \alpha^{at+r} \\ \alpha^{q-1} &= \alpha^{at}\alpha^{r} \end{aligned} \tag{B.10}$$

Como $\alpha^t = 1$ e $\alpha^{q-1} = 1$ então:

$$1 = 1 \cdot \alpha^r \tag{B.11}$$

Assim, $\alpha^r = 1$, ou seja, $r = 0$.

Teorema 4 *O grupo de elementos não nulos diferentes de $GF(q)$ é um grupo cíclico sobre a multiplicação [27].*

Seja $GF(q)$ um corpo e $GF(Q)$ sua extensão ($q^m = Q$). Seja β elemento de $GF(Q)$. O polinômio f de menor grau tal que $f(\beta) = 0$ é chamado de *polinômio mínimo de* β. Esse polinômio sempre existe e é único e irredutível. O polinômio mínimo é utilizado para gerar palavras códigos, pois a palavra $P(x)$ só é código se e somente se $P(\beta_1) = 0$.

Para $\alpha \in GF(q)$, seja t o menor inteiro tal que $\alpha^{p^t} = \alpha$. Define-se *conjugado de* α

$$C = \{\alpha, \alpha^{p^1}, \alpha^{p^2}, \alpha^{p^3}, \ldots, \alpha^{p^t}\}$$

Para representar os 2^n elementos o conceito de base é introduzido. A escolha de como representar elementos de corpo sobre bases ou potências de elementos primitivos usualmente depende da implementação em hardware ou software que seja adotada.

B.4 Bases

Um conjunto de n elementos linearmente independentes $x = \{x_0, x_1, x_2, \ldots, x_{n-1}\}$ em $GF(2^n)$ é chamado de base de $GF(2^n)$. Dessa forma, um elemento $x \in GF(2^n)$ pode ser representado unicamente como uma soma ponderada dessa base sobre $GF(2)$ (para um mapeamento no corpo binário)

$$a = a_0 x_0 + a_1 x_1 + \cdots + a_{n-1} x_{n-1} \tag{B.12}$$

onde $a_0, ..., a_{n-1} \in GF(2)$.

B.4.1 Base Polinomial

Seja $p(x)$ um polinômio irredutível em $GF(2)$. Tomando α como raiz de $p(x)$, então $A = \{1, \alpha, \ldots, \alpha^{n-1}\}$ é uma base polinomial de $GF(2^n)$.

Exemplo 14 *Considerando* $GF(2^4)$ *e o polinômio irredutível em* $GF(2)$ $p(x) = x^4 + x + 1$. *Tomando* α *como raiz de* $p(x)$ *então* $A = \{1, \alpha, \alpha^2, \alpha^3\}$ *forma a base polinomial pois todos os 16 elementos podem ser representados como:*

$$a = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 \qquad \text{em que } a_i \in GF(2) \tag{B.13}$$

B.4.2 Base Dual

Sejam $\{\lambda_i\}$ e $\{\mu_i\}$ bases de $GF(2^n)$ e seja f uma função linear tal que $GF(2^n) \rightarrow GF(2)$ e $\beta \in GF(2^n)$. Então $\{\lambda_i\}$ e $\{\mu_i\}$ são duais com respeito a f e β :

$$f(\beta\lambda_i\mu_j) = \begin{cases} 1 \text{ se } i = j \\ 0 \text{ se } i \neq j \end{cases} \tag{B.14}$$

Nesse caso, $\{\lambda_i\}$ é a base padrão e $\{\mu_i\}$ é a base dual.

Exemplo 15 *Seja $p(x) = x^4 + x + 1$ o polinômio primitivo em $GF(2^4)$ e seja α uma raiz de $p(x)$. Tomando $\beta = 1$ e f como o coeficiente menos significativo da base polinomial, a base dual da base polinomial é $\{1, \alpha^3, \alpha^2, \alpha\}$*

$$\lambda_0 = 1 \quad \begin{cases} f(1\cdot 1) = 1 & f(1\cdot\alpha^2) = 0 \\ f(1\cdot\alpha) = 0 & f(1\cdot\alpha^3) = 0 \end{cases}$$

$$\lambda_0 = \alpha \quad \begin{cases} f(\alpha\cdot 1) = 0 & f(\alpha\cdot\alpha^2) = 0 \\ f(\alpha\cdot\alpha) = 0 & f(\alpha\cdot\alpha^3) = 1 \end{cases}$$

$$\lambda_0 = \alpha^2 \quad \begin{cases} f(\alpha^2\cdot 1) = 0 & f(\alpha^2\cdot\alpha^2) = 1 \\ f(\alpha^2\cdot\alpha) = 0 & f(\alpha^2\cdot\alpha^3) = 0 \end{cases}$$

$$\lambda_0 = \alpha^3 \quad \begin{cases} f(\alpha^3\cdot 1) = 0 & f(\alpha^3\cdot\alpha^2) = 0 \\ f(\alpha^3\cdot\alpha) = 1 & f(\alpha^3\cdot\alpha^3) = 0 \end{cases}$$

$$\left\{1, \alpha, \alpha^2, \alpha^3\right\} = \left\{1, \alpha^3, \alpha^2, \alpha\right\} \tag{B.15}$$

Variando o valor de β é possível obter até 2^{n-1} bases duais para qualquer base dada. Assim, a base dual com melhor característica (dependendo da aplicação) pode ser utilizada.

B.4.3 Base Normal

Uma base normal é uma base da forma $B = \left\{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\right\}$, em que $\beta \in GF(2^n)$. Para cada corpo existe no mínimo uma base normal. Ela é bastante útil quando a situação exige potenciação, pois se $a = (a_0, a_1, \cdots, a_{n-1})$ é a representação na base normal, então $a^2 = (a_{n-1}, a_0, \cdots, a_{n-2})$.

Apêndice C

Logaritmo de Zech

Um novo método de calcular o Logaritmo de *Zech* sem o uso de tabela foi desenvolvido em [31]. Para isso são utilizadas algumas definições e propriedades das classes ciclotômicas laterais.

Definição 3 *Classe ciclotômica lateral de x é o conjunto dos elementos $\{2^0x, 2^1x, 2^2x, 2^3x, \cdots, 2^{m-1}x\}$, em que m é o menor número tal que $2^m x (\mathrm{mod}\, q-1) = x$.*

Definição 4 *Líder da classe ciclotômica de x é o menor elemento da classe na qual x pertence.*

Uma classe lateral qualquer dada por $C_x = \{\xi^i x(\xi)\}$, para $i = (0, 1, \cdots, m-1)$ está no anel $F_2[\xi](\mathrm{mod}\, \xi^m - 1)$, ou seja, a sua representação binária é um deslocamento cíclico do líder de classe na forma binária. Vale notar que o logaritmo de *Zech* de um elemento também está num anel cíclico. Por isso se o logaritmo de *Zech* do líder de classe for encontrado e em seguida deslocado pode-se achar todos os logaritmos de *Zech* dos elementos da classe em que o líder de classe faz parte.

Exemplo 16 *Em $GF(32)$ o logaritmo de Zech do elemento $\alpha^1 = 1+\alpha^{18}$, podendo ser escrito como $Z(1) = 18$, em que $Z(\cdot)$ denota o logaritmo de Zech. Então pode-se determinar todos os logaritmos de Zech dos elementos que pertence a sua classe.*

São três passos para determinar o logaritmo de *Zech* de um elemento do corpo.

- Determinar a classe no qual o elemento faz parte (achar o líder de classe).
- Determinar o logaritmo de *Zech* do líder de classe.

C_x	$\vec{D}(x)$	$S\left(\vec{D}\right)$	$Zech\,(C_x)$
C_0	$(0,0)$	0	11111
C_1	$(1,2)$	5	10010
C_3	$(2,2)$	6	11101
C_5	$(2,4)$	10	00010
C_7	$(3,2)$	7	10110
C_{11}	$(3,4)$	11	10011
C_{15}	$(4,2)$	8	11000

Tabela C.2: Cálculo do logaritmo de Zech em $GF(2^4)$.

conservando ainda a separação das classes. Essa função é definida como $S(\vec{D}) \triangleq \sum_{i=0}^{l-1} p_i \rho_i$, em que os parâmetros $p_i \in N$ são escolhidos os menores possíveis (devido a implementação física).

Com posse dos valores de $S(\vec{D})$, que é único para cada líder de classe, são utilizadas redes neurais para encontrar o seu Logaritmo de *Zech*.

Primeiramente são determinados os intervalos $[a_j, b_j]$ de $S(\vec{D})$ tal que $z_k = 1$, para $j = 1, ..., s$. O segundo passo é construir as funções limiares:

$$\begin{cases} Ya_j(S) = sgn(S - a_j) \\ Yb_j(S) = sgn(b_j - S) \end{cases}$$

Então para o bit z_k temos:

$$z_k = sgn\left\{\sum_{j=1}^{s}(Ya_j + Yb_j) - s\right\}$$

Utilizando o mesmo procedimento para $k = 1, ..., n$, em que n é o número de bits, é obtido o logaritmo de *Zech* desejado. Para encontrar o vetor $\vec{D}$ e os parâmetros que minimizam a função $S\left(\vec{D}\right)$ foi utilizado busca exaustiva através de métodos computacionais. A seguir um exemplo de implementação do logaritmo de Zech para $GF\,(2^5)$.

Exemplo 17 *Em $GF\,(2^5)$ existem sete classes laterais. Separando essas classes encontra-se o vetor $\vec{D} = \{(w(x), w(X^{l_1}), w(X^{l_2}), \cdots, w(X^{l_m})\}$, que é encontrado testando cada polinômio $X^{k_1^i, k_2^i, \ldots, k_j^i}(\zeta)$ de tal sorte que o vetor $\vec{D}$ seja diferente para cada classe. Para $GF\,(32)$ foi encontrado o vetor $\vec{D} = \{(w(x), w(X^1)\}$.*

De posse desses valores foram encontrados os parâmetros $p_1 = 1$ e $p_1 = 2$ que minimizam a função $S(\vec{D})$. A partir desse ponto utilizando a construção telescópica de 2 camadas, é possível obter o logaritmo de Zech utilizando 14 portas de limiar linear.

Bibliografia

[1] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, Inc., 1984.

[2] C. C. Wang and D. Pei. "A VLSI Design for Computing Exponentiations in $GF(2^m)$ and Its Application to Generate Pseudorandom Number Sequences". *IEEE Transactions on Computers*, Vol. 39, No. 2, pp. 258–266, February 1991.

[3] S. B. Wicker. *Error Control Systems for Digital Communication and Storage*. Ms Gra-Hill, Inc, February 1995.

[4] Kailath, Siu and Roychaudhury. *Discrete Neural Computation, A Theoretical Foundation*. McGraw-Hill, November 1995.

[5] M. Hasan, M. Wang and V. Bhargava. "Modular Construction of Low Complexity Parallel Multipliers for a Class of Finite Fields $GF(2^m)$". *IEEE Transactions on Computers*, Vol. 41, No. 8, pp. 199–202, August 1992.

[6] S. T. J. Fenn, M. Benaissa and D. Taylor. "$GF(2^m)$ Multiplication and Division Over the Dual Basis". *IEEE Transactions on Computers*, Vol. 45, No. 3, pp. 319–327, March 1994.

[7] C. Paar. "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields". *IEEE Transactions on Computers*, Vol. 45, No. 7, pp. 856–861, July 1996.

[8] L. A. N. Oliveira and F. M. Assis. "Arquitetura Eficiente de Um Multiplicador de Mastrovito Usando Redes Neurais Discretas". *Anais do III Congresso Brasileiro de Redes Neurais,SC*, Vol. 1, pp. 12–15, Julho 1997.

[9] C. C. Wang. "An Algorithm to Design Finite Fields Multipliers Using a Self-Dual Normal Basis". *IEEE Transactions on Computers*, Vol. 38, No. 10, pp. 1457–1460, October 1989.

[10] I.-S. Hsu, T. K. Truong, L. J. Deutsch and I. S. Reed. "A Comparison of VLSI Architecture of Finite Field Multipliers Using Dual, Normal, or Standard Bases". *IEEE Transactions on Computers*, Vol. 37, No. 6, pp. 735–739, June 1988.

[11] Ç. K. Koç and B. Sunar. "Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields". *IEEE Transactions on Computers*, Vol. 47(3), pp. 353–356, March 1998.

[12] C. Paar, P. Fleischmann and P. Roelse. "Efficient Multiplier Architectures for Galois Fields $GF(2^{4n})$". *IEEE Transactions on Computers*, Vol. 42, No. 47(2), pp. 162–170, March 1998.

[13] C. Paar, P. Fleischmann and P. Soria-Rodriguez. "Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents". *IEEE Transactions on Computers*, Vol. 48, No. 10, pp. 1025–1034, October 1999.

[14] E. Mastrovito. "VLSI Architecture for Computations in Galois Fields". Ph.D. thesis, Department of Electrical Engineering of Linkoping, Sweeden, March 1991.

[15] Y. Amit. "A Neural Network Architecture for Visual Selection". *Neural Computation*, Vol. 12, No. 5, pp. 1141–1164, 2000.

[16] K.-Y. Siu, V. P. Roychowdhury and T. Kailath. "Depth-Size Tradeoffs for Neural Computation". *IEEE Transactions on Computers*, Vol. 40, No. 12, pp. 1402–1412, May 1991.

[17] M. Goldmann and M. Karpinski. "Simulating Threshold Circuits by Majority Circuits". *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, Vol. 36, No. 9, pp. 551–560, March 1993.

[18] Siu and Bruck. "Neural Computation of Arithmetic Functions". *IEEE Transactions on Information Theory*, Vol. 47, No. 8, pp. 1669–1675, October 1998.

[19] L. A. N. Oliveira and F. M. Assis. "Arquitetura de Um Multiplicador de Mastrovito de Complexidade Reduzida Pelo Uso de Redes Neurais Discretas". *XV Simpósio Brasileiro de Telecomunicações*, , No. 2, pp. 744–765, Setembro 1997.

[20] F. M. Assis and L. A. N. Oliveira. "A Parallel Multiplier for $GF(2^n)$". *XVII Simpósio Brasileiro de Telecomunicações*, pp. 677–681, Setembro 1999.

[21] L. Zhang and B. Zhang. "A Geometrical Representation of McCulloch-Pitts Neural Model and Its Applications". *IEEE Transactions on Neural Networks*, Vol. 10, No. 4, pp. 925–929, April 1999.

[22] K.-Y. Siu and J. Bruck. "On the Power of Threshold Circuits with Small Weights". *SIAM J. Disc. Math*, Vol. 4, No. 3, pp. 423–435, 1991.

[23] V. Bohossian and J. Bruck. "On Neural Networks with Minimal Weights". *Electronic Colloquium on Computational Complexity*, Vol. 3, No. 7, pp. 34–37, May 1996.

[24] J. Bruck and R. Smolensky. "Polynomial Threshold Functions, AC 0 Functions and Spectral Norms". *In Proc. 31st IEEE Symposium on Foundations of Computer Science*, Vol. 44, pp. 632–641, February 1990.

[25] K. Y. Siu and V. P. Roychowdhury. "On Optimal Depth Threshold Circuits for Multiplication and Related Problems". *SIAM Journal on Discrete Mathematics*, Vol. 7, No. 3, pp. 946–950, May 1993.

[26] B. Siu, Kailath and Hofmeiste. "Depth Efficient Neural Networks for Division and Related Problems". *IEEE Transactions on Information Theory*, Vol. 39, No. 5, pp. 946–956, May 1993.

[27] S. Lin and D. J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall International, Inc., May 1983.

[28] v. O. Scott A. Vanstone abd Oayk C. *An Introduction to Error Correcting Codes with Applications*. Kluwer Academic, 1989.

[29] C. Paar. "Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields". Ph.D. thesis, Institute for Experimental Mathematics, Univ. of Essen, 1994.

[30] K. Huber. "Some Comments on Zech Logarithms". *IEEE Transations on Information Theory*, Vol. 36, pp. 946–950, July 1990.

[31] F. M. Assis and C. E. Pedreira. "An Architecture for Computing Zech's Logarithms in $GF(2^m)$". *IEEE Transactions on Computers*, Vol. 49, No. 5, pp. 256–261, May 2000.

[32] A. Halbutogullari and Ç. K. Koç. "Mastrovito Multiplier for General Irreducible Polynomials". *IEEE Transactions on Computers*, Vol. 49, No. 5, pp. 503–518, May 2000.