Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Programa de Pós-graduação em Engenharia Elétrica

Circuito Integrado para Multiplicação em GF(2^4) Utilizando Portas de Limiar Linear

Dissertação de mestrado submetida à Coordenação do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, em cumprimento às exigências para obtenção do grau de Mestre em Ciências no Domínio da Engenharia Elétrica.

Mestrando:

Cristóvão Mácio de Oliveira Lima Filho

Orientadores:

Prof. Dr. Raimundo Carlos Silvério Freire (UFCG)

Prof. Dr. Francisco Marcos de Assis (UFCG)

Campina Grande, maio de 2010

Dedicatória

À minha mãe, Ana Lúcia, como gratidão por todo o esforço, dedicação e amor que tem para conosco.

À minha grande amiga, Ana Elizabeth, com quem compartilhei os melhores momentos da minha vida.

Agradecimentos

Aos meus orientadores Raimundo Carlos Silvério Freire e Francisco Marcos de Assis, pela orientação e incentivo durante todo o desenvolvimento do trabalho.

Aos amigos da pós-graduação Alan, Bob, Carol, Eline, Euler, Georgina, Ilton, Karolie, Sabiniano, Selma e Thiago pela amizade e companheirismo e aos demais colegas que ao longo desses anos contribuíram para minha formação profissional e pessoal.

Resumo

Esta dissertação descreve o desenvolvimento de um leiaute de uma nova arquitetura de multiplicador em corpos finitos baseada no multiplicador de Mastrovito. Tal arquitetura tem como unidades de processamento as portas de limiar linear, que é o elemento básico de uma rede neural discreta.

As redes neurais discretas implementadas com portas de limiar linear permitem reduzir a complexidade de certos circuitos antes implementados com lógica tradicional (Portas AND, OR e NOT). Com isso, a idéia de estender o uso de portas de limiar linear em operações aritméticas em corpos finitos se torna bastante atraente. Assim, para comprovar de forma prática, a eficiência das portas de limiar linear, a arquitetura de um multiplicador em GF(2^4), proposta em (LIDIANO - 2000), foi implementada utilizando as ferramentas de desenho de leiaute de circuito integrado da *Mentor Graphics*®.

Os resultados da simulação do leiaute do circuito integrado do multiplicador em GF(2^4) são apresentados. Os mesmos indicaram um desempenho abaixo do esperado, devido a complexidade espacial do multiplicador em GF(2^n) com $n = 4$ não ser suficiente para que as vantagens da implementação com portas de limiar linear sejam visualizada.

Palavras-chave: Multiplicador de Mastrovito, Redes Neurais Discretas, Portas de Limiar Linear.

Abstract

This dissertation describes the development of a layout of new multiplication architecture in Galois field based on the Mastrovito multiplier. The processing unit of this new architecture is a threshold logic gate, which is a basic element of a discrete neural network.

The discrete neural network built with threshold logic gates allow reduce de complexity of a certain circuits once built using traditional boolean gates (AND, OR and NOT). Therewith, the idea of extending the advantages of the threshold logic gates for arithmetic operations in Galois field to become very attractive. Thus, to confirm into practice form, the advantages of the threshold logic gates, a multiplier architecture in $GF(2^4)$, proposed in (LIDIANO - 2000), was implemented using the integrated circuit layout tools of *Mentor Graphics®*.

The results from simulations of the layout of multiplier in $GF(2^4)$ are presented. These results indicated a low performance, due to the space complexity of $GF(2^n)$ multiplier with $n = 4$ is not enough for show the advantages of the multiplier implementation with threshold logic gates.

Key Words: Mastrovito Multiplier, Discrete Neural Networks, Threshold Logic Gates.

Sumário

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1 – Motivação

Desde a publicação do artigo "*A mathematical theory of communication*" [1] por Claude Shannon, em 1948, a teoria da informação tem sido de grande importância para o desenvolvimento dos sistemas de comunicações digitais modernos e dos sistemas de armazenamento de dados. Nesses sistemas, dados são processados (recebidos, transmitidos ou armazenados) a velocidades elevadas e também em alguns desses sistemas está envolvida a transmissão de dados através de canais ruidosos, dessa forma há sempre a possibilidade de ocorrência de erros.

Resultados da teoria de Shannon garantem que 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 utilizado. Assim, o uso de códigos corretores de erros (*Error Correcting Codes* – ECC), tais como BCH (Bose, Chaudhuri e Hocquenghem), Goppa e RS (*Reed-Solomon Codes*) são uma constante nos meios de comunicação.

Na figura 1.1 é mostrada a estrutura de um sistema de comunicação digital simples. Nesse sistema, o dado digital proveniente da fonte é codificado, criptografado e modulado para a comunicação através do canal. O dado recebido é demodulado, descriptografado e decodificado para então ser utilizado. Todos os elementos desse sistema de comunicação são descritos matematicamente pelos teoremas da Teoria da Informação [2].

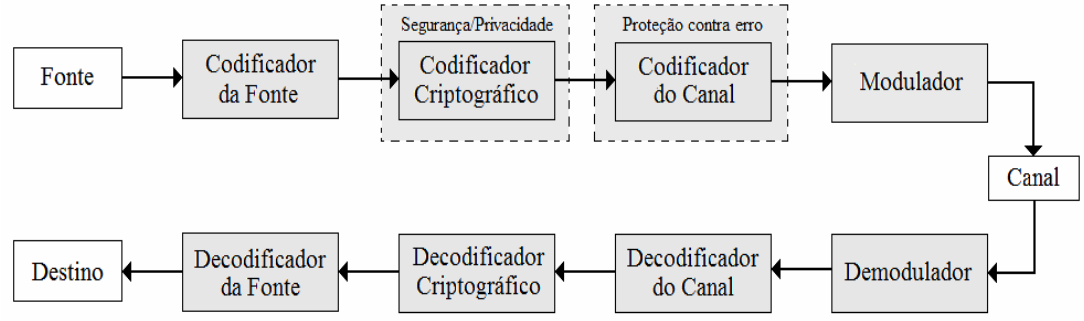

Figura 1.1 – Estrutura geral de um sistema de comunicação digital.

Como pode ser visto na figura 1.1, na transmissão de dados digitais há a possibilidade de ocorrência de erro nos dados, devido às interferências inseridas pelo canal, portanto, faz-se necessário a utilização de códigos corretores de erros, tais como BCH, Goppa e RS, para solucionar o problema mencionado. Uma outra questão é a garantia da transmissão segura da informação em canais públicos e privados, ou seja, sem o risco de que pessoas não autorizadas possam ler a informação transmitida. As técnicas para uma transmissão segura de informação são proporcionadas pela criptografia. Todas essas técnicas fazem uso da geometria algébrica como ferramenta para manipulação da informação.

A geometria algébrica, em particular os corpos finitos ou campos de Galois, tem se mostrado útil no desenvolvimento de soluções em diversas áreas da engenharia, como na codificação para controle de erros, em criptografia, em processamento digital de sinais, entre outras. Operações aritméticas em corpos finitos, tais como soma, multiplicação, divisão, cálculo de inversa e exponenciação, são de fundamental importância para as áreas mencionadas. Por exemplo, as operações de multiplicação e divisão são utilizadas nos codificadores e decodificadores em sistemas RS de correção de erros. Por sua vez, a soma, a multiplicação, o cálculo de inversa e exponenciação são fundamentais para o desenvolvimento de criptosistemas que operam com curvas elípticas.

Dentre as operações aritméticas em um corpo finito, a multiplicação é de grande importância, já que é a base de outras operações mais complexas, tais como divisão e exponenciação. Assim, são desejáveis multiplicadores em corpos finitos cada vez mais eficientes para a melhoria do desempenho dos sistemas que os utiliza em virtude da demanda por altas velocidades para esses sistemas.

Multiplicadores em corpos finitos podem ser implementados em *software* e em *hardware*. Implementações em *software* possuem a desvantagem da necessidade de grande quantidade de memória e capacidade de processamento [3]. Muitas pesquisas [4-11] têm sido desenvolvidas em busca de métodos de multiplicação em corpos finitos, implementados em *hardware*, cada vez mais rápidos e eficientes. A eficiência dessas arquiteturas depende fundamentalmente da forma de representação dos elementos do corpo [12]. Considerando um corpo finito com q elementos, representado por GF(q), em que os campos de interesse para aplicações em engenharia são os campos de extensão em GF(2) em razão da facilidade de mapeamento dos elementos do corpo em bits, já que os campos com extensão GF(2) são representados por 2 elementos (0 e 1) e são

designados por $GF(2^m)$. Existem distintas bases de representação dos elementos de $GF(2^m)$, sendo as mais populares a base canônica, a base normal e a base dual. Estas bases caracterizam os diferentes tipos de multiplicadores bem como sua complexidade.

Eficientes implementações em *hardware* têm sido realizadas em FPGA [13-17]. Muitas dessas implementações são realizadas utilizando portas AND, OR e NOT, chamadas de portas AON ou tradicionais. Uma possível solução para se obter arquiteturas de multiplicadores ainda mais eficientes é a utilização de portas com um potencial maior que as portas AON.

Arquiteturas que fazem uso de redes neurais têm se destacado em diversas aplicações [18-22], sobretudo no que diz respeito à complexidade computacional dos circuitos. O elemento básico de processamento dessas redes são portas que calculam uma função de limiar linear ou um elemento analógico que executa uma função sigmoidal. Essas portas de limiar vêm sendo estudadas desde 1960 e eficientes implementações em CMOS são descritas em [23-30]. Aplicações com portas de limiar lógicas [31,32] demonstram um grande potencial em obter altas velocidades de operação e redução de área do *chip*, se comparadas com a lógica convencional.

Este trabalho de dissertação descreve a implementação da arquitetura de um multiplicador em $GF(2^4)$ descrita em [33], no qual apresenta uma nova arquitetura para multiplicação em corpo finito utilizando portas de limiar linear, visando circuitos com complexidade reduzida.

Os resultados teóricos obtidos em [33] demonstraram que a utilização de portas de limiar linear em arquiteturas de multiplicação em corpos finitos é justificada, já que diminui consideravelmente a complexidade espacial e temporal do circuito do multiplicador se comparado com o mesmo multiplicador com portas tradicionais. Desta forma, torna-se necessário a implementação física da arquitetura proposta para comprovar sua viabilidade.

1.2 – Objetivo da Dissertação

O objetivo dessa dissertação de mestrado consiste no desenvolvimento de um circuito integrado que realize a multiplicação em $GF(2^4)$ utilizando portas de limiar linear como elemento de processamento. O multiplicador implementado é baseado na arquitetura de Mastrovito que apresenta uma das melhores medidas de complexidade

espacial se comparado com outras arquiteturas de multiplicadores em paralelo. Para tal objetivo, os seguintes passos são necessários:

1. Avaliar, de forma prática, o uso das portas de limiar linear em multiplicadores em corpos finitos, visando circuitos com menor complexidade espacial e temporal;
2. Escolher qual a melhor implementação, em CMOS, da porta de limiar linear para a utilização em multiplicadores em corpos finitos, tomando como parâmetros a potência dissipada, o atraso no cálculo da multiplicação e a área em *chip* ocupada;
3. Projetar o leiaute do circuito impresso do multiplicador em $GF(2^4)$ com as ferramentas de desenho de circuitos da *Mentor Graphics*.

1.3 – Organização do Trabalho

Este trabalho está organizado da seguinte maneira: no capítulo 1 é feita uma breve introdução, onde é apresentada a motivação deste trabalho, bem como os passos necessários para realização do mesmo.

No capítulo 2 é realizada uma revisão da aritmética dos corpos finitos com uma abordagem na aplicação em Hardware.

No capítulo 3 são apresentadas as Redes Neurais Discretas. Algumas características e propriedades são levantadas, bem como conceitos relevantes para o desenvolvimento do trabalho. Neste capítulo é explicado como são construídos os circuitos que utilizam redes neurais discretas para o cálculo de operações de adição e multiplicação.

No capítulo 4 é apresentada a implementação de um multiplicador em $GF(2^4)$ construído com redes neurais discretas e baseado na arquitetura de Mastrovito.

No capítulo 5 são descritos os passos tomados para a construção do leiaute do multiplicador em $GF(2^4)$ construído com portas de limiar linear. Neste capítulo, também são apresentados os resultados da simulação do leiaute do circuito do multiplicador.

No capítulo 6 são apresentadas as conclusões e sugestões para trabalhos futuros.

2. Aritmética de GF(2^m) em *Hardware*

Neste capítulo são apresentados os fundamentos e as definições matemáticas para uma melhor compreensão do trabalho. Um estudo mais amplo a respeito da aritmética dos corpos finitos pode ser visto em [34-36].

No que diz respeito às implementações, a eficiência dos operadores em campos de Galois são caracterizadas pelas seguintes medidas:

1. **Complexidade espacial**: definida pelo número de portas lógicas requeridas para a construção do circuito;
2. **Complexidade temporal**: definida pelo número de retardos das portas ao longo da rota dos dados no circuito.

Uma arquitetura para unidades aritméticas em corpo finito pode ser serial ou paralela. Uma arquitetura paralela produz as saídas a partir de dados de entrada normalmente no período de um único ciclo de relógio. São circuitos tipicamente combinatórios, que favorecem o tempo de processamento em detrimento da complexidade espacial. Por outro lado, uma arquitetura serial utiliza em geral menos recursos materiais, em termos de portas lógicas, mas necessita de vários ciclos de relógio para produzir as saídas a partir das entradas. Como consequência, em uma arquitetura serial existem sempre elementos de memória, como *flip-flops* e registradores, o que implica na necessidade de um controle mais complexo em comparação com os circuitos paralelos.

Uma vez que o objetivo desta dissertação é produzir um multiplicador que responda às altas exigências em potência de cálculo, optou-se pela utilização exclusiva de arquiteturas paralelas para as unidades aritméticas em corpo finito.

2.1 – Álgebra dos Corpos Finitos

A aritmética dos Corpos Finitos é extensivamente usada em códigos corretores de erros, na geração de números Pseudo-aleatórios, em criptografia e em processamento digital de sinais. Um corpo finito, também chamado de campo de Galois, é um conjunto finito de elementos no qual são definidas algumas operações aritméticas. Nesta subseção são apresentadas definições algébricas básicas e propriedades dos corpos finitos, as quais são relevantes para o desenvolvimento deste trabalho.

2.1.1 – Aritmética Modular

Dado um inteiro n, todo inteiro i pode ser unicamente expressado como $i = qn + r$ para algum inteiro r, chamado de resto, no intervalo $0 \leq r \leq n - 1$ e para algum inteiro q, chamado de quociente.

O resto r, denotado por $r = i \bmod n$, em que "*mod n*" significa a operação em módulo n, é a parte mais importante da expressão, já que ele define a operação modular. O conjunto de possíveis restos *mod n* é o conjunto de n inteiros $R_n = \{0, 1, ..., n - 1\}$. É evidente que n é divisor de i se e somente se $i \bmod n = 0$.

A aritmética que usa o conjunto de restos *mod n* é chamada de aritmética modular ou aritmética *mod n*. As leis da aritmética *mod n* seguem as mesmas leis da aritmética dos inteiros. Sendo $r = i \bmod n$ e $s = j \bmod n$, ou seja, $r = i - qn$ e $s = j - tn$, então:

$$r + s = i + j - (q + t)n; \tag{2.1}$$
$$rs = ij - (qj + ti)n + qtn^2. \tag{2.2}$$

Portanto, $(r + s) \bmod n = (i + j) \bmod n$ e $rs \bmod n = ij \bmod n$, ou seja, o resto *mod n* da soma ou produto de dois inteiros é igual ao resto *mod n* da soma ou produto de dois restos *mod n* inteiros. Assim as leis da adição e da multiplicação *mod n* são definidas como:

$$r \oplus s = (r + s) \bmod n; \tag{2.3}$$
$$r * s = (rs) \bmod n. \tag{2.4}$$

em que no lado esquerdo da equação, r e s são elementos do conjunto de restos R_n, enquanto que no lado direito da equação, r e s correspondem a elementos do conjunto dos números inteiros.

2.1.2 – Grupos

Definição 2.1. Um grupo é um conjunto de elementos $G=\{a, b, c, ...\}$ e uma operação $\circ$, tal que as seguintes propriedades são satisfeitas:

1) Lei associativa: a operação é associativa, ou seja:

$$a \circ (b \circ c) = (a \circ b) \circ c \tag{2.5}$$

em que *a, b* e $c \in G$.

2) Identidade: existe um elemento identidade e em G, tal que:

$$a \circ e = e \circ a = a \tag{2.6}$$

para todo $a \in G$.

3) Inversa: para cada $a \in G$, existe um elemento $a' \in G$, tal que:

$$a' \circ a = a \circ a' = e \tag{2.7}$$

4) Condição de fechamento: para qualquer $a \in G$ e $b \in G$, o elemento $a \circ b$ é também um elemento de G.

Em todo grupo G o elemento identidade é único e ainda o inverso de cada elemento de G também é único. Portanto, dado um conjunto G e uma operação $\circ$, existem quatro propriedades que devem ser satisfeitas para que G seja um grupo: condição de fechamento, associatividade, existência de um elemento identidade e existência uma inversa para cada elemento de G.

Como exemplos de operações em um grupo, têm-se a adição e a multiplicação denotadas por $\oplus$ e por $*$, respectivamente. No caso da adição, o elemento identidade é o 0 e a inversa de *a* é *–a*. Já na multiplicação, o elemento identidade é o 1 e a inversa de *a* é a^{-1}. Um grupo G é dito ser comutativo ou abeliano quando $a \circ b = b \circ a$, para todo $a, b \in G$. Portanto, a notação de adição é geralmente usada para grupos comutativos e a notação de multiplicação é usada para grupos comutativos e não comutativos.

Um exemplo de grupo é o conjunto dos $\Re$ com a operação de adição, em que o elemento identidade é 0 e a inversa de *a* é *–a*. O conjunto dos $\Re$ menos o 0, denotado por $\Re^*$ forma um grupo com a operação de multiplicação, em que o elemento identidade é o 1 e a inversa de *a* é $a^{-1} = 1/a$.

2.1.3 – Anéis

Definição 2.2. Um anel, representado por *(R, ⊕, *)*, consiste em um conjunto *R* com duas operações $\oplus$ e $*$ sobre *R*, tal que as seguintes propriedades são satisfeitas:

1) *(R, ⊕)* forma um grupo comutativo com o elemento identidade sendo 0;

2) *(R, *)* forma um semigrupo, com a operação $*$ sendo associativa: $a*(b*c) = (a*b)*c$ para todo *a, b, c* $\in$ *R*;

3) Existe uma unidade multiplicativa, denotada por 1, tal que: $1 * a = a * 1$, para todo $a \in R$ e $1 \neq 0$;

4) As operações estão relacionadas pela lei distributiva: $(a \oplus b) * c = (a * c) \oplus (b * c)$, para todo $a, b, c \in R$.

Definição 2.3. Se R é um anel arbitrário e existe um inteiro estritamente positivo n tal que $nr = 0$ para qualquer $r \in R$, então n é chamado de característica do anel R. Se tal inteiro positivo não existe, se diz que R possui característica 0.

Definição 2.4. Um elemento a de um anel R é chamado de elemento inversível se existe um elemento $b \in R$ tal que $a * b = 1$.

2.1.4 – Corpos

Definição 2.5. Um corpo é um conjunto F com pelo menos dois elementos e com duas operações $\oplus$ e $*$, para os quais as seguintes propriedades são satisfeitas:

1) O conjunto F forma um grupo comutativo sobre a operação $\oplus$;
2) O conjunto $F^* = F - \{0\} = \{a \in F, a \neq 0\}$ forma um grupo comutativo sobre a operação $\oplus$;
3) Lei distributiva: Para todo $a, b, c \in F$, $(a \oplus b) * c = (a * c) \oplus (b * c)$.

Um corpo é um anel comutativo, no qual todos os elementos não nulos possuem inversos multiplicativos.

Definição 2.6. Um corpo finito é um conjunto F que contém um número finito de elementos. A ordem de F é seu número de elementos.

Teorema 2.1. Existência e Unicidade dos Corpos Finitos:

1) Se F é um corpo finito, então F contém $q = p^m$ elementos para algum número primo p e um inteiro $n \geq 1$;
2) Para todo número q da forma p^m, em que p é primo e $m \geq 1$, existe um único campo finito de ordem p^m. Este campo é denotado por F_q ou por GF(q).

Teorema 2.2. Subcorpo de um Corpo Finito: Seja F_q um campo finito de ordem $q = p^m$, então todo subcorpo de F_q tem ordem p^n, para algum $n \leq m$ que seja divisor de m.

2.1.5 – Polinômios Sobre Corpos Finitos

Um polinômio *f(x)* não-nulo de grau *m* sobre um corpo *F* é expresso da seguinte forma:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_m x^m, \tag{2.8}$$

em que $f_i \in F$, $0 \le i \le m$, e $f_m \neq 0$. O grau de *f(x)*, denotado por *deg(f(x))*, é *m*. O símbolo *x* é uma incógnita. O conjunto de todos os polinômios sobre *F* é denotado por *F[x]* e forma um anel de polinômios.

Definição 2.7. Se *R* é um anel comutativo, o anel de polinômios *R[x]* é formado pelos polinômios com variável *x* e com coeficientes em *R*.

Definição 2.8. Seja $f(x) = \sum_{i=0}^{m} a_i x^i$ um polinômio não nulo sobre *R*, então $a_m \neq 0$. O elemento a_m é chamado de coeficiente principal de *f(x)*. Se o coeficiente principal de *f(x)* é 1, então *f(x)* é chamado de polinômio mônico.

Definição 2.9. Seja $f(x) \in F[x]$, assim *f(x)* é chamado de polinômio irredutível sobre *F* se este não pode ser representado como produto de dois polinômios em *F[x]*.

2.1.6 – Aritmética Modular com Polinômios

Dado um polinômio mônico *g(x)* de grau *m*, todo polinômio *f(x)* pode ser expresso como *f(x) = q(x)g(x) + r(x)* em que *deg r(x) < m*. Assim, o resto polinomial *r(x)* é denotado por *r(x) = f(x) mod g(x)*. O conjunto de todas as possibilidades de restos polinomiais é designado por $R_{f,m} = \{r_0 + r_1 x + \cdots + r_{m-1} x^{m-1} \mid r_j \in F, 0 \le j \le m-1\}$.

As leis da aritmética *mod g(x)* seguem as mesmas leis da aritmética polinomial. Sendo *r(x) = f(x) mod g(x)* e *s(x) = h(x) mod g(x)*, ou seja, *r(x) = f(x) – q(x)g(x)* e *s(x) = h(x) – t(x)g(x)*, então:

$$f(x) + h(x) = r(x) + s(x) - (q(x) + t(x))g(x); \tag{2.9}$$

$$f(x)h(x) = r(x)s(x) - (q(x)s(x) + t(x)r(x))g(x) + q(x)t(x)g^2(x). \tag{2.10}$$

Portanto, *(f(x) +h(x)) mod g(x) = (r(x) + s(x)) mod g(x)* e *f(x)h(x) mod g(x) = r(x)s(x) mod g(x)*. Assim as leis da adição e multiplicação polinomiais em *mod g(x)* são definidas por:

$$r(x) \oplus s(x) = (r(x) + s(x)) \bmod g(x); \quad (2.11)$$
$$r(x) * s(x) = (r(x)s(x)) \bmod g(x). \quad (2.12)$$

em que o lado esquerdo da equação, *r(x)* e *s(x)* são elementos do conjunto de restos polinomiais $R_{F,\,m}$, enquanto que no lado direito da equação, *r(x)* e *s(x)* correspondem a polinômios ordinários.

2.1.7 – Construção de Corpos com p^n Elementos

Uma representação muito utilizada para os elementos de um corpo finito F_q, com $q = p^n$ e p sendo um número primo, é mediante uma base polinomial. Nesta base, os elementos de um corpo finito F_q serão representados por polinômios em $Z_p[x]$ de grau < n, e com coeficientes em Z_p.

Exemplo 2.1. O campo finito F_{16}, em que $16 = 2^4$, pode ser representado pelo conjunto de polinômios sobre F_2 de grau < 4. Ou seja,

$$F_{16} = \{a_3x^3 + a_2x^2 + a_1x + a_0 \mid a_i \in \{0,1\}\}. \quad (2.13)$$

Por conveniência, cada polinômio $a_3x^3 + a_2x^2 + a_1x + a_0$ é representado pelo correspondente vetor de coeficientes (a_3, a_2, a_1, a_0) de dimensão 4, assim:

$$F_{16} = \{(a_3, a_2, a_1, a_0) \mid a_i \in \{0,1\}\}. \quad (2.14)$$

Este tipo de representação é muito adequado para propósitos computacionais, já que o polinômio $f(x) = x^4 + x + 1$ é irredutível sobre Z_2. As operações no corpo são realizadas em módulo *f(x)*.

Definição 2.10. Um polinômio irredutível $f(x) \in Z_p[x]$ de grau n é chamado de polinômio primitivo se x é um gerador de $F^*_{p^n}$.

Exemplo 2.2. Mediante o polinômio irredutível $f(x) = x^4 + x + 1$, se tem o polinômio $p(x) = x$ como sendo um polinômio primitivo, e se diz que o elemento $\alpha = (0, 0, 1, 0)$ do campo é um gerador de $F^*_{2^4}$. Dessa forma, tem-se:

$$F^*_{2^4} = \{x^i \bmod f(x) \mid 0 \geq i < 2^4 - 1\}. \quad (2.15)$$

2.1.8 – Bases de Corpos Finitos

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) se um elemento $x \in$ GF(2^n) for representado unicamente como uma soma ponderada dessa base sobre GF(2).

$$x = a_0 x_0 + a_1 x_1 + \cdots + a_{n-1} x_{n-1}, \tag{2.16}$$

em que $a_0, a_1, \ldots, a_{n-1} \in$ GF(2).

Existem diferentes tipos de bases para representar os elementos de um corpo finito. As três bases mais utilizadas para projetar operadores aritméticos sobre corpos finitos são: a base polinomial, a base normal e a base dual. Neste trabalho será utilizada apenas a base polinomial, já que o multiplicador desenvolvido é baseado na arquitetura de Mastrovito que é um multiplicador em GF(2^n) com base polinomial.

Base Polinomial

Seja *p(x)* um polinômio irredutível sobre 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 2.3. Considerando GF(2^4) e o polinômio irredutível em GF(2) sendo $p(x) = x^4 + x + 1$. 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:

$$a = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3, \tag{2.17}$$

em que $a_i \in$ GF(2). Na tabela 2.1 são mostradas as diferentes formas de representação dos elementos de GF(2^4).

A vantagem dos multiplicadores em base polinomial é que os mesmos não requerem conversão de base, tornando o projeto de multiplicadores em GF(2^n) mais simples e mais eficiente se comparado, por exemplo, com o de multiplicadores de base dual.

Elemento de GF(2^4)	Representação Polinomial	Representação Vetorial
0	0	(0, 0, 0, 0)
α	α	(0, 0, 1, 0)
α^2	α^2	(0, 1, 0, 0)
α^3	α^3	(1, 0, 0, 0)
α^4	$\alpha^3 + 1$	(1, 0, 0, 1)
α^5	$\alpha^3 + \alpha + 1$	(1, 0, 1, 1)
α^6	$\alpha^3 + \alpha^2 + \alpha + 1$	(1, 1, 1, 1)
α^7	$\alpha^2 + \alpha + 1$	(0, 1, 1, 1)
α^8	$\alpha^3 + \alpha^2 + \alpha$	(1, 1, 1, 0)
α^9	$\alpha^2 + 1$	(0, 1, 0, 1)
α^{10}	$\alpha^3 + \alpha$	(1, 0, 1, 0)
α^{11}	$\alpha^3 + \alpha^2 + 1$	(1, 1, 0, 1)
α^{12}	$\alpha + 1$	(0, 0, 1, 1)
α^{13}	$\alpha^2 + \alpha$	(0, 1, 1, 0)
α^{14}	$\alpha^3 + \alpha^2$	(1, 1, 0, 0)
α^{15}	1	(0, 0, 0, 1)

Tabela 2.1 – Representação dos elementos de GF(2^4).

2.2 – Multiplicação em GF(2^n)

2.2.1 – Considerações Prévias

Com base no que foi visto na seção anterior, têm-se as seguintes considerações:

Seja p um número primo. É sabido que o conjunto de restos módulo p, GF(p) = *{0, 1, ..., p – 1}* é um corpo finito chamado primo. Todo corpo finito possui um elemento nulo e um elemento unidade e é uma extensão de grau finito de um corpo primo. Os corpos finitos ou campos de Galois são da forma GF(p^n), em que p é primo, n é um inteiro positivo. GF(p^n) contém p^n elementos.

Em particular, para $p = 2$, GF(2^n) contém 2^n elementos e seu corpo primo é GF(2) = {0, 1}. Seu grupo multiplicativo possui $2^n - 1$ elementos, no qual contém um

elemento α, chamado de elemento primitivo, tal que todos os elementos de GF(2^n) – {0} podem ser representados como potências de tal elemento. Assim, tem-se:

$$GF(2^n) = \{0\} \bigcup \{\alpha^1, \alpha^2, \ldots, \alpha^{2n-2}, \alpha^{2n-1} = 1\}. \tag{2.18}$$

Fazendo P(α) = 0, tem-se:

$$\alpha^n = Q(\alpha) = q_{n-1}\alpha^{n-1} + q_{n-2}\alpha^{n-2} + \cdots + q_1\alpha + q_0 \tag{2.19}$$

A equação 2.19 é utilizada para representar qualquer elemento do corpo como um combinação linear única das potencias 1, α, α^2, ..., α^{n-1}. Portanto, qualquer elemento de GF(2^n) pode ser representado por uma forma polinomial em α de grau *n* e coeficientes em GF(2). Em outras palavras, qualquer elemento $A \in$ GF(2^n) pode ser representado da forma:

$$A(\alpha) = \sum_{i=0}^{n-1} a_i \alpha^i, \tag{2.20}$$

em que $a_i \in$ GF(2) para cada $0 \leq i \leq n-1$. O conjunto { 1, α, α^2, ..., α^{n-1}} é dito ser a base polinomial.

Para implementar operações em *hardware*, a representação polinomial de cada elemento pode ser feita com um vetor binário com *n* elementos, da forma $(a_{n-1}, a_{n-2}, \ldots, a_0)$, esta representação é chamada de vetorial. Como $p = 2$, os elementos do campo podem ser representados convenientemente pelos dois sinais lógicos "0" e "1". Desta forma, a operação de adição módulo 2 é equivalente a uma operação XOR e a operação de multiplicação módulo 2 a uma operação AND.

2.2.2 – Multiplicador de Mastrovito

Em [8], Mastrovito desenvolve um multiplicador sobre GF(2^n), paralelo de base polinomial. Sua arquitetura apresenta baixa complexidade temporal e espacial se comparada com a de outros multiplicadores de base polinomial [9]. Nessa subseção será mostrado como é realizada a multiplicação em GF(2^n), baseada na arquitetura do multiplicador de Mastrovito.

Para o cálculo do produto de dois elementos arbitrários, considere *A(x)*, *B(x)* e *C(x)* $\in$ GF(2^n) com GF(2^n) escrito na forma polinomial; a multiplicação é realizada da seguinte forma:

$$\begin{aligned} C(x) &= A(x) \cdot B(x) \bmod P(x) \\ &= (a_{n-1}x^{n-1} + \ldots + a_0) \cdot \\ &\quad (b_{n-1}x^{n-1} + \ldots + b_0) \bmod P(x) \end{aligned} \tag{2.21}$$

em que *P(x)* é o polinômio irredutível de grau *n* em GF(2). O multiplicador de Mastrovito realiza a multiplicação em duas etapas, como mostrado na figura 2.1.

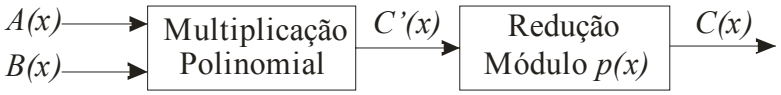

Figura 2.1 – Diagrama de blocos do multiplicador de Mastrovito.

Na primeira etapa é efetuada a multiplicação polinomial, *A(x)B(x)*, como mostrado na Equação (2.22).

$$\begin{aligned} C'(x) &= A(x) \cdot B(x) \\ &= (a_{n-1}x^{n-1} + \ldots + a_0) \cdot (b_{n-1}x^{n-1} + \ldots + b_0) \\ &= a_0b_0 + (a_1b_0 + a_0b_1)x + \\ &+ (a_2b_0 + a_1b_1 + a_0b_2)x^2 + \ldots + (a_{n-1}b_{n-1})x^{2n-2} \end{aligned} \tag{2.22}$$

Esta etapa requer n^2 portas AND e $n^2 - 2n + 1$ portas XOR [22].

Na segunda etapa é calculada a redução módulo *P(x)* para encontrar o resultado final *C(x)* $\in$ GF(2^n). Esta etapa depende do polinômio irredutível escolhido, que para este caso foi escolhido um trinômio da forma $P(x) = x^n + x + 1$. A expressão final para *C(x)* é mostrada na Equação (2.23):

$$\begin{aligned} C(x) &= (c'_0 + c'_n) + (c'_1 + c'_n + c'_{n+1})x + \ldots + \\ &+ (c'_{n-2} + c'_{2n-2} + c'_{2n-3})x^{n-2} + \\ &+ (c'_{n-1} + c'_{2n-2})x^{n-1} \end{aligned} \tag{2.23}$$

Para a redução em módulo *P(x)* são usadas *2n – 2* portas XOR. O atraso total, considerando todas as portas, é dado por:

$$T \leq \tau_{AND} + 2\tau_{XOR} \lceil \log_2 n \rceil \tag{2.24}$$

em que τ_{AND} e τ_{XOR} são os atrasos correspondentes às portas AND e XOR, respectivamente.

2.2.3 – Considerações Finais

Como mencionado na introdução, as implementações em *hardware* de operadores aritméticos em corpos finitos GF(2^n) são uma boa alternativa em relação às implementações em *software*, devido à maior segurança e desempenho obtidas as mesmas. Tais implementações em *hardware* podem ser feitas de duas formas: utilizando circuitos VLSI (*Very Large Scale Integration*) ou circuitos programáveis, como FPGA.

Dentre essas formas de implementação, optou-se pela utilização de circuitos VLSI, que cada vez mais se identificam com os ASIC (*Application Specific Integrated Circuit*) [35].

3. Redes Neurais Discretas

3.1 – Introdução

A idéia do modelamento de neurônios em unidades discretas foi introduzida por McCulloch e Pitts [37] em 1943. Cada neurônio é modelado como um elemento de limiar linear com uma saída binária. Desde o início dos estudos de redes neurais, modelos discretos dessas redes tem sido parte integrante nas pesquisas em computação neural.

Indubitavelmente, tal modelo binário do neurônio não pode modelar toda a complexidade inerente ao sistema neural biológico. Entretanto, o interesse em um modelo binário reside em sua simplicidade. O modelo binário de McCulloch e Pitts não é usado somente em pesquisas na área de redes neurais, é usado, também, no desenvolvimento de computadores digitais. Recentes pesquisas em redes neurais demonstram um contínuo esforço no desenvolvimento de princípios computacionais baseados na neurobiologia. Nesse capítulo são introduzidos aspectos importantes das redes neurais discretas.

3.2 – Portas de Limiar linear

A demanda por alto desempenho de processadores em escala de integração VLSI tem impulsionado as pesquisas por sistemas que garantam alto desempenho utilizando pouca área em *chip* e que apresentem baixo consumo de potência.

As portas lógicas de limiar (*Threshold Logic Gate* - TLG) vêm sendo estudadas desde 1960 e implementações eficientes em CMOS são descritas em [23-30]. Portas lógica de limiar podem calcular qualquer função linearmente separável, ou seja, funções cuja saída só depende do somatório de suas entradas ponderadas, e seu funcionamento básico é baseado na comparação entre a soma ponderada das entradas da porta e um certo limiar.

3.2.1 – Comportamento das Portas de Limiar

Uma porta de limiar linear é um elemento básico de uma rede neural que pode conter *n* entradas binárias, representadas por $x_1, x_2, ..., x_n$, e apenas uma saída binária *y*, como pode ser visto na figura 3.1.

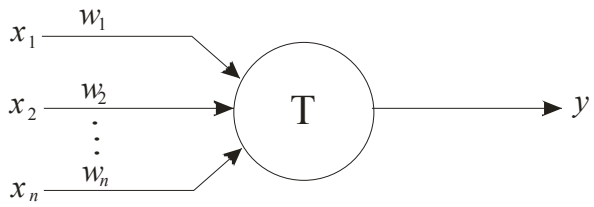

Figura 3.1 – Modelo de uma porta de limiar

A função booleana calculada por uma porta de limiar é chamada de função de limiar e é especificada pelo seu limiar T e seus pesos $w_1, w_2, ..., w_n$, em que w_i é o peso correspondente a cada variável de entrada x_i. Uma porta de limiar linear pode calcular qualquer função booleana dada por:

$$f(X) = Y = \begin{cases} 1 & se \sum_{i=1}^{k} w_i \cdot x_i \geq t \\ 0 & se \sum_{i=1}^{k} w_i \cdot x_i < t \end{cases}, \tag{3.1}$$

A função ainda pode ser escrita de uma forma mais compacta utilizando a notação *sgn*, dada por:

$$Y = \text{sgn}\left(\sum_{i=1}^{n} w_i x_i - t \right), \tag{3.2}$$

em que x_i é a *i*-ésima entrada booleana, w_i é comumente referido como peso da *i*-ésima entrada e *t* é o limiar da função *f(X)*. A função *sgn* é definida por $sgn(\alpha)=1$ se $\alpha \geq 0$, $sgn(\alpha)=0$ caso contrário. Dessa forma, uma porta de limiar linear realiza a comparação entre o valor do somatório das entradas multiplicadas pelos seus respectivos pesos, $\sum x_i w_i$, e o valor do limiar *t*. Se o valor do somatório for maior ou igual ao limiar a saída da porta será o nível lógico 1, caso contrário, a saída será o nível lógico 0. Diferentes escolhas de *w* e *t* resultam em portas de limiar que calculam diferentes funções booleanas, por exemplo, na figura 3.2 são mostradas portas de limiar configuradas para calcular as funções booleanas AND e OR.

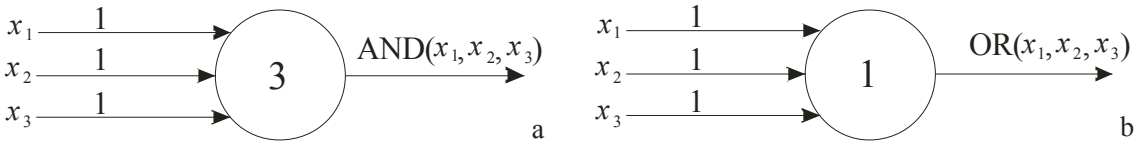

Figura 3.2 – Portas de limiar que calculam funções booleanas: a) função AND e b) função OR.

Essa versatilidade das portas de limiar linear oferece um avanço computacional significativo em comparação com as portas lógicas tradicionais, AND-OR-NOT [38], especialmente em aplicações que requerem um alto número de variáveis de entrada.

3.3 – Funções Simétricas

As portas de limiar linear estão restritas à soluções de problemas que sejam linearmente separáveis, ou seja, a problemas cuja solução pode ser obtida pela separação de duas regiões por meio de uma reta (ou um hiperplano para o caso *n*-dimensional). A equação de separação linear pode ser facilmente visualizada para o caso bidimensional. Considere como exemplo ilustrativo, um nó com duas entradas x_1 e x_2, pesos w_1 e w_2, limiar t e saída y executando uma função qualquer. A condição de disparo do nó ($y=1$) é então definida por $x_1w_1 + x_2w_2 = t$, que pode ser descrita na forma geral da equação de uma reta em que $x_2=f(x_1)$, conforme é mostrada na equação 3.3. Portanto, a superfície de decisões de uma porta de limiar linear está restrita a uma reta. Na figura 3.3 é mostrada a solução para o problema do AND lógico por meio de uma porta de limiar linear.

$$x_2 = -\left(\frac{w_1}{w_2}\right)x_1 + \left(\frac{t}{w_2}\right) \tag{3.3}$$

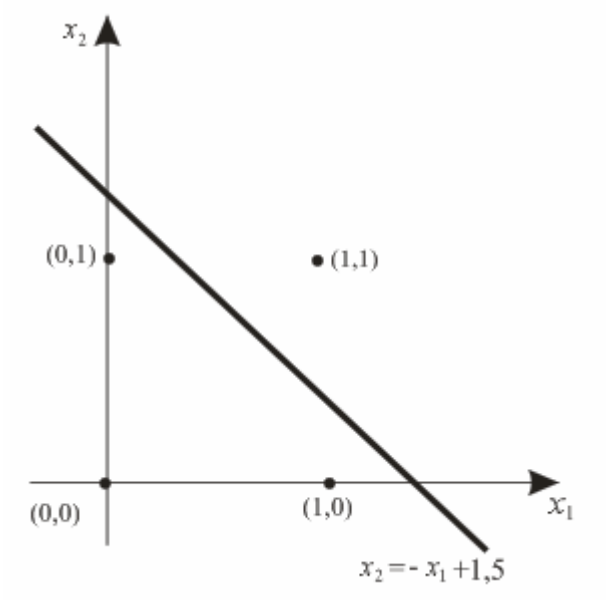

Figura 3.3 – Solução para o problema do AND lógico utilizando uma porta de limiar linear. Para este caso, tem-se $w_1=w_2=1$ *e* $t=1{,}5$.

Para o caso particular em que $x \in \{0, 1\}^n$, que restringe as entradas a valores binários, tem-se que $y : \{0, 1\}^n \rightarrow \{0, 1\}$ para $w \in R^n$. Apesar de estarem limitadas à resolução de problemas linearmente separáveis, que corresponde a uma pequena parcela do total de 2^{2^n} funções booleanas possíveis com *n* entradas, as portas de limiar lineares são mais poderosas do que as portas lógicas convencionais. Com uma mesma porta de limiar linear pode-se implementar qualquer uma das funções AND, OR, NAND e NOR, entre outras, bastando mudar os parâmetros da porta.

3.4 – Construção de Circuitos de Retardo Fixo

Como as portas de limiar linear estão restritas à solução de funções simétricas, o projeto de circuitos de limiar compreenderá, para este trabalho, a implementação de funções booleanas simétricas. O modelo de rede neural utilizada neste trabalho para a implementação dos circuitos é o de propagação direta, em que cada elemento em uma camada calcula uma função de limiar que de pende apenas dos valores das saídas da camada anterior, como pode ser visto na figura 3.4. Nessa seção serão apresentadas formas eficientes de construção de circuitos de limiar com duas e três camadas.

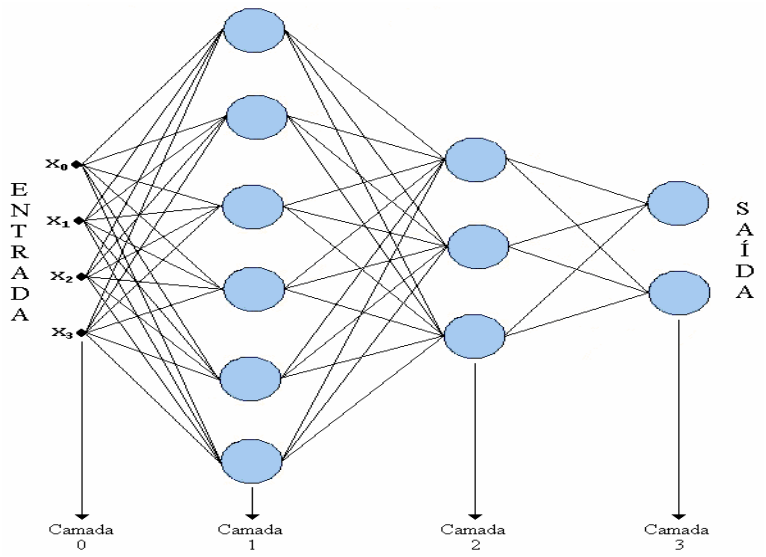

Figura 3.4 – Modelo de uma Rede Neural com propagação direta.

Definição 3.1. Uma função booleana f é simétrica se sua saída depende apenas do somatório dos seus valores de entrada.

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

Prova: Qualquer função booleana pode ser escrita como uma Soma de Produtos (SDP), da seguinte forma:

$$f(x_1, x_2, ..., x_n) = P_1 \vee P_2 \vee \cdots \vee P_k \tag{3.4}$$

em que cada termo P_i é um produto implementado por uma função AND de n variáveis e $k \leq 2^n$. Alternativamente, f pode ser escrita como um Produto de Somas (PDS), da seguinte forma:

$$f(x_1, x_2, ..., x_n) = S_1 \wedge S_2 \wedge \cdots \wedge S_l \tag{3.5}$$

em que cada termo S_i é uma soma implementada pela função OR de n variáveis e $l \leq 2^n$.

Sendo 2^n entradas distintas, segue que $k + l = 2^n$, já que a SDP e a PDS formam subconjuntos de variáveis complementares, ou seja, 2^n é o número máximo de produtos e somas com n variáveis booleanas. Dessa forma, a primeira camada do circuito de limiar implementará, para a SDP, os produtos P_i e a segunda camada é uma função OR que calculará f, utilizando assim $2^{n-1}+1$ portas tradicionais. Como uma porta neural pode simular portas 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. Portanto, faz se necessário o estudo de técnicas, que fazem uso das redes neurais discretas, para a construção de circuitos de limiar que calculam certas classes de funções de maneira eficiente e que o número de portas de limiar aumenta polinomialmente com o aumento do número de entradas. Essas técnicas serão apresentadas a seguir.

3.4.1 – Circuitos de duas camadas

Nessa subseção é apresentada uma técnica para construção de circuitos de limiar linear de 2 camadas que computa qualquer função booleana simétrica f com no máximo $n+1$ portas, em que *n* é o número de entradas do circuito.

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 em $\sum_{k=1}^{n} x_k$ no qual a função f é 1. Agrupando esses valores em subintervalos de $[0, n]$ tem-se s subintervalos dados a seguir:

$$[q_1, \tilde{q}_1], [q_2, \tilde{q}_2], ..., [q_s, \tilde{q}_s] \tag{3.6}$$

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, tal que:

$$\sum_{k=1}^{n} x_k \in [q_j, \widetilde{q}_j] \tag{3.7}$$

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

$$y_{q_j} = \text{sgn}\left(\sum_{k=1}^{n} x_k - q_j \right) \tag{3.8}$$

e

$$\widetilde{y}_{q_j} = \text{sgn}\left(\widetilde{q}_j - \sum_{k=1}^{n} x_k \right) \tag{3.9}$$

em que $j = 1, 2, ..., s$.

A segunda camada contém uma única porta neural que calcula:

$$f(x_1, x_2, ..., x_n) = \text{sgn}\left(\sum_{k=1}^{s} (y_{k_j} + \widetilde{y}_{k_j}) - s - 1 \right) \tag{3.10}$$

Para comprovar se o circuito fornece a resposta certa temos que, para $j = 1, 2, ..., s$, se $\sum_{k=1}^{n} x_k \notin [q_j, \widetilde{q}_j]$, então $y_{k_j} + \widetilde{y}_{k_j} = 1$ para todos os j. Assim:

$$\text{sgn}\left(\sum_{k=1}^{s} (y_{k_j} + \widetilde{y}_{k_j}) - s - 1 \right) = \text{sgn}(s - s - 1) = 0 \tag{3.11}$$

Caso $\sum_{k=1}^{n} x_k \in [q_j, \widetilde{q}_j]$, então $y_{k_j} + \widetilde{y}_{k_j} = 2$ e $y_{k_i} + \widetilde{y}_{k_i} = 1$ para $i \neq j$. Assim:

$$\text{sgn}\left(\sum_{k=1}^{s} (y_{k_j} + \widetilde{y}_{k_j}) - s - 1 \right) = \text{sgn}(s + 1 - s - 1) = 1 \tag{3.12}$$

Portanto, a primeira camada possui apenas $2s$ 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 $O(n^2)$ (ver apêndice A). Para ilustrar o projeto de circuitos de limiar linear com 2 camadas, na figura 3.5 é mostrado o diagrama da implementação de uma função de paridade de 4 variáveis. Esse circuito necessita de 5 portas de limiar linear para implementar a paridade. Utilizando a lógica tradicional são necessárias 8 portas AND e 1 porta OR para implementar a mesma função. Para 10 variáveis de entrada são necessárias 512 portas AND e 1 porta OR, enquanto que se for utilizada a lógica neural, apenas 11 portas seriam necessárias.

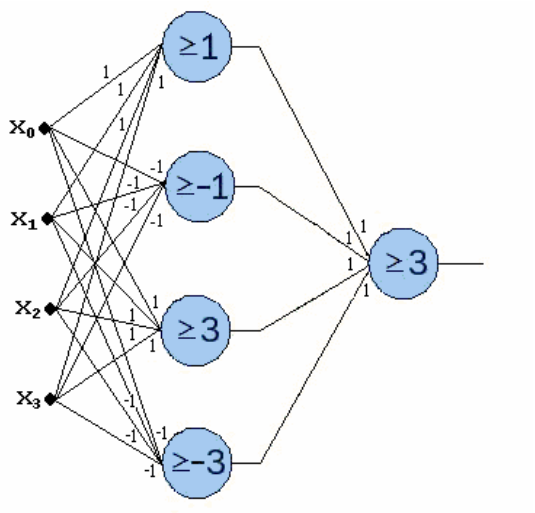

Figura 3.5 – Paridade de 4 variáveis com portas de limiar linear.

É possível diminuir ainda mais o número de portas de limiar linear, para isso é necessário utilizar de uma técnica chamada de técnica telescópica, que será apresentada a seguir.

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

$$\sum_{j=1}^{k}(a_j - a_{j-1})y_j = a_m \tag{3.13}$$

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.

Prova: Como $y_i = \text{sgn}\left(\sum_{j=1}^{n} x_j - b_i\right) = 1$ 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 $y_1 = y_2 = ... = y_m = 1$ e $y_{m+1} = ... = y_k = 0$. Dessa forma, tem-se:

$$\sum_{j=1}^{k}(a_j - a_{j-1})y_j = \sum_{j=1}^{m}(a_j - a_{j-1}) = a_m \tag{3.14}$$

Note que $\sum_{j=1}^{n} x_j \in [b_0, b_1 - 1]$, ou seja $m = 0$, 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 1, pode-se obter um circuito de limiar de duas camadas com no máximo $\left\lceil \frac{n}{2} \right\rceil + 1$ portas que calculam funções simétricas.

Seja $f(X)$ uma função simétrica de n variáveis. Seja 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 com:

$$s_i \leq \sum_{j=1}^{n} x_j \leq S_i \tag{3.15}$$

A primeira camada do circuito consiste de τ portas de limiar linear calculando $y_i = \text{sgn}\left(\sum_{j=1}^{n} x_j - s_i\right)$ para cada i, sendo $1 \leq i \leq \tau$. A segunda camada possui uma única porta que calcula a seguinte função:

$$z = \text{sgn}\left(\sum_{j=1}^{n} (S_j - S_{j-1}) y_j - \sum_{j=1}^{n} x_j \right) \tag{3.16}$$

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, tem-se:

$$\sum_{j=1}^{\tau} (S_j - S_{j-1}) y_j = S_m \tag{3.17}$$

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

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 $\left\lceil \frac{n}{2} \right\rceil + 1$. Na figura 3.6 é mostrado um circuito que calcula a paridade de 4 variáveis utilizando apenas 3 portas de limiar linear, o que comprova a validade da técnica telescópica.

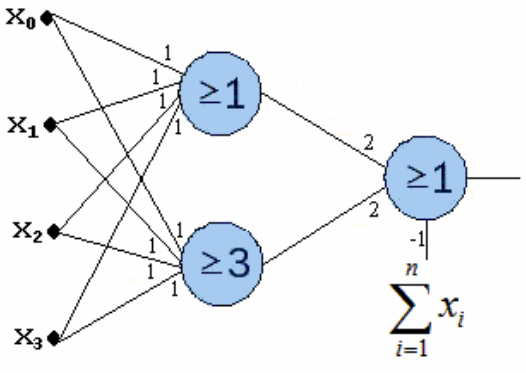

Figura 3.6 – Paridade de 4 variáveis com circuito de 2 camadas, construído com a técnica telescópica.

3.4.2 – Circuitos de três camadas

Generalizando a técnica telescópica para construções de circuitos de 3 camadas, em que, neste caso, também é possível uma redução do número de portas de limiar linear. Como resultados, são obtidos circuitos que calculam funções simétricas com $2\sqrt{n}+1$ 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:

$$s_i \leq \sum_{j=1}^{n} x_j \leq S_i \tag{3.18}$$

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, os quais calculam a seguinte função:

$$z_i = \operatorname{sgn}\left(\sum_{j=1}^{n} x_j - s_{i_1} \right) \tag{3.19}$$

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

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

$$\begin{aligned} T_k &= S_{1_k} z_1 + (S_{2_k} - S_{1_k}) z_2 + (S_{3_k} - S_{2_k}) z_3 + ... + (S_{d_k} - S_{d-1_k}) z_d \\ t_k &= s_{1_k} z_1 + (s_{2_k} - s_{1_k}) z_2 + (s_{3_k} - s_{2_k}) z_3 + ... + (s_{d_k} - s_{d-1_k}) z_d \end{aligned} \tag{3.20}$$

Observe que t_k e T_k são combinações lineares das saídas da primeira camada. A segunda camada consiste em $2l$ portas de limiar linear, cada uma utiliza os inteiros t_k ou T_k como valor de limiar para o cálculo de Q_k e q_k definidos a seguir:

$$\begin{aligned} Q_k &= \operatorname{sgn}\left(T_k - \sum_{j=1}^{n} x_j \right) \\ q_k &= \operatorname{sgn}\left(t_k - \sum_{j=1}^{n} x_j \right) \end{aligned} \tag{3.21}$$

A terceira camada é uma única porta que calcula a seguinte função:

$$f(X) = \operatorname{sgn}\left(\sum_{k=1}^{l} 2(Q_k + q_k) - 2l - 1 \right) \tag{3.22}$$

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

$$\begin{aligned} T_k &= S_{m_k} \\ t_k &= s_{m_k} \end{aligned} \tag{3.23}$$

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

$$s_{m_k} \leq \sum_{j=1}^{n} x_j \leq S_{m_k} \tag{3.24}$$

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 sendo:

$$Q_k + q_k = \begin{cases} 2 & se\ k = i \\ 1 & se\ k \neq i \end{cases} \tag{3.25}$$

Portanto, o elemento de saída da terceira camada é:

$$\operatorname{sgn}\left(\sum_{k=1}^{l} 2(Q_k + q_k) - 2l - 1\right) = \operatorname{sgn}(2l + 2 - 2l - 1) = 1 \tag{3.26}$$

Similarmente, se $f(X) = 0$, então não existe k tal que $s_{m_k} \le \sum_{j=1}^{n} x_j \le S_{m_k}$.

Portanto, $Q_k + q_k = 1$ para todo k e a porta da terceira camada calcula:

$$\operatorname{sgn}\left(\sum_{k=1}^{l} 2(Q_k + q_k) - 2l - 1\right) = \operatorname{sgn}(2l - 2l - 1) = 0 \tag{3.27}$$

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 $2l \le \left\lceil \frac{n}{d} \right\rceil + 1$ 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 3.7, a seguir, é mostrado um diagrama de implementação de 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 3.6 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 que o circuito de 3 camadas necessitará de 13.

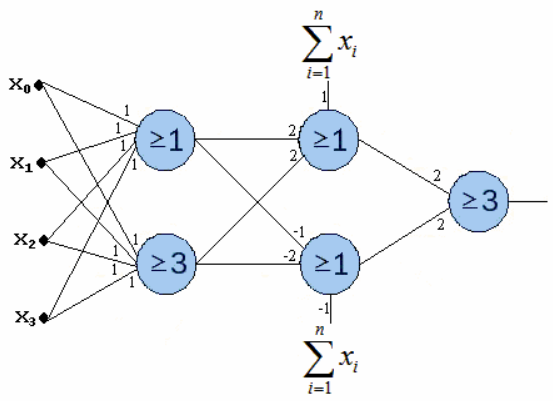

Figura 3.7 – Paridade de 4 variáveis com circuito de 3 camadas, construído com a técnica telescópica.

3.5 – 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 aumenta exponencialmente [33]. Para que o tamanho do chip aumente polinomialmente, é necessário projetar circuitos com retardo não fixo o que acarreta 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.

3.5.1 – 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.

Inicialmente, é necessário encontrar os intervalos em que a função seja igual a 1, como mostrado na tabela 3.1.

(x_1, x_0)	(y_1, y_0)	$soma(z_2\, z_1\, z_0)$	(x_1, x_0)	(y_1, y_0)	$soma(z_2\, z_1\, z_0)$
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	011=3	11	10	101=5
01	11	100=4	11	11	110=6

Tabela 3.1 – Soma de dois números de 2 bits.

Note que o peso de x_1 e y_1 é 2 e o peso de x_0 e y_0 é 1. Separando os resultados em que cada z_i é 1, tem-se:

- Bit z_0 $\rightarrow$ Os resultados em que z_0 é 1 são: 1, 3 e 5. Mas como z_0 é função, apenas de x_0 e y_0, então é utilizado apenas o resultado 1;
- Bit z_1 $\rightarrow$ Resultados 2, 3 e 6;
- Bit z_2 $\rightarrow$ Resultados 4, 5 e 6;

Assim, as portas da primeira camada podem ser implementadas da seguinte forma:

- Bit 1

$$y_0 = \text{sgn}\left(\sum_{i=1}^{n} w_i x_i - 1\right) \tag{3.28}$$

$$\bar{y}_0 = \text{sgn}\left(1 - \sum_{i=1}^{n} w_i x_i\right) \tag{3.29}$$

$$z_0 = \text{sgn}\left(y_0 + \bar{y}_0 - 2\right) \tag{3.30}$$

- Bit 2

$$y_1 = \text{sgn}\left(\sum_{i=1}^{n} w_i x_i - 2\right) \tag{3.31}$$

$$\bar{y}_1 = \text{sgn}\left(3 - \sum_{i=1}^{n} w_i x_i\right) \tag{3.32}$$

$$z_1 = \text{sgn}\left(y_1 + \bar{y}_1 - 2\right) \tag{3.33}$$

- Bit 3

$$y_2 = \text{sgn}\left(\sum_{i=1}^{n} w_i x_i - 4\right) \tag{3.34}$$

$$\bar{y}_2 = \text{sgn}\left(6 - \sum_{i=1}^{n} w_i x_i\right) \tag{3.35}$$

$$z_2 = \text{sgn}\left(y_2 + \bar{y}_2 - 2\right) \tag{3.36}$$

Para implementar esse somador são utilizadas 9 portas neurais, enquanto que, se utilizada a lógica tradicional seriam necessárias 27 portas. Embora, tanto o somador implementado com a lógica tradicional quanto o somador implementado com a lógica neural possuam número de camadas constantes, apenas o somador implementado com a lógica neural possui tamanho polinomial.

3.5.2 – Multiplicação

A multiplicação de n bits pode ser dividida em três etapas, como mostrado na figura 3.8.

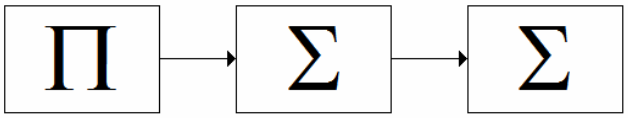

Figura 3.8 – Bloco funcional do multiplicador.

O primeiro bloco calcula 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 como *carry-save*, para implementar circuitos que executem a adição. O terceiro bloco é uma adição propriamente dita.

- Bloco I - AND

A multiplicação pode ser reduzida 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:

$$Z = \sum_{i=0}^{n-1} z_{i_{2n-1}} z_{i_{2n-2}} \cdots z_{i_0} \tag{3.37}$$

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

$$z_i = \underbrace{0\cdots 0}_{n-i}(x_{n-1} \wedge y_i)(x_{n-2} \wedge y_i)\cdots(x_0 \wedge y_i)\underbrace{0\cdots 0}_{i} \tag{3.38}$$

Utilizando uma porta de limiar linear, para implementar a lógica AND da equação 3.38, seriam necessárias n^2 portas para calcular todos os z_i's. Portanto, a multiplicação foi reduzida a uma soma de n números de $2n$ bits.

- Bloco II – Técnica *Block-Save*

A principal dificuldade no cálculo de uma soma com uma grande quantidade de números é o cálculo do bit de *carry* em paralelo. A tradicional técnica *carry-save* reduz a soma de três números em uma soma de dois números em um único passo.

Sejam três números $X = x_{n-1}\,x_{n-2}\ldots x_0$, $Y = y_{n-1}\,y_{n-2}\ldots y_0$ e $Z = z_{n-1}\,z_{n-2}\ldots z_0$ todos em representação binária. O *i*-ésimo bit de cada número pode ser somado, resultando em $x_i + y_i + z_i = 2c_{i+1} + w_i$, em que c_{i+1} é o bit de *carry* gerado pela soma dos *i*-ésimos bits x_i, y_i e z_i. Dessa forma, tem-se $X + Y + Z = C + W$, em que $C = c_n\,c_{n-1}\ldots c_0$ com $c_0 = 0$ e $W = w_{n-1}\,w_{n-2}\ldots w_0$. Note que $c_{i+1} = x_i y_i \vee y_i z_i \vee z_i x_i$ e $w_i = x_i \oplus y_i \oplus z_i$, portanto, os c_{i+1} e w_i podem ser calculados paralelamente em um único passo. Por exemplo, seja $X = 1001$, $Y = 0111$ e $Z = 1101$, assim $C = 11010$ e $W = 0011$.

A técnica *block-save* é um aprimoramento da técnica *carry-save*. Ela reduz a soma de n números $O(n)$ bit para uma soma de dois números de $O(n)$ bit em um único passo. A técnica *block-save* é apresentada por meio de um exemplo em que os n números do somatório possuem n bits.

Escrevendo cada um dos n bits como sendo $x_i = x_{i_{n-1}}\,x_{i_{n-2}}\ldots x_{i_0}$, para $i = 1,\ldots,n$. Para simplificar, assumiremos que ao invés de termos n inteiros de n bits cada, teremos n inteiros com $n/(\log n)$ bits, e fazendo $N = n/\log n$. Separando cada número binário x_i em N blocos consecutivos de $\tilde{x}_{i_0}, \tilde{x}_{i_1}, \ldots, \tilde{x}_{i_{N-i}}$ para cada $(\log n)$ bits, teremos:

$$x_i = \sum_{j=0}^{N-1} \tilde{x}_{i_j} \cdot 2^{\log n \cdot j} \tag{3.39}$$

em que $0 \le \tilde{x}_{i_j} < 2^{\log n}$. Logo, a soma total, depois de reorganizados os índices do somatório, será:

$$S = \sum_{i=1}^{n} x_i = \sum_{j=0}^{N-1} \left(\sum_{i=1}^{n} \widetilde{x}_{i_j} \right) \cdot 2^{\log n \cdot j} \tag{3.40}$$

Observe que para cada $j = 0, \ldots, N-1$, a soma de cada bloco $\widetilde{s}_j$ será:

$$\widetilde{s}_j = \sum_{i=1}^{n} \widetilde{x}_{i_j} < \sum_{i=1}^{n} 2^{\log n \cdot j} = 2^{2 \log n} \tag{3.41}$$

Portanto, $\widetilde{s}_j$ pode ser representado por $2 \log n$ bits. Então, cada bloco de somas $\widetilde{s}_j$ pode ser expresso por:

$$\widetilde{s}_j = \widetilde{c}_{j+1} 2^{\log n} + \widetilde{w}_j \tag{3.42}$$

em que $0 \le \widetilde{c}_{j+1} < 2^{\log n}$ são os $(\log n)$ bits do *carry* das somas $\widetilde{s}_j$ com $\widetilde{c}_0 = 0$ e $\widetilde{w}_j < 2^{\log n}$. Assim, tem-se:

$$\begin{aligned} S &= \sum_{j=0}^{N-1} \left(\sum_{i=1}^{n} \widetilde{x}_{i_j} \right) \cdot 2^{\log n \cdot j} \\ &= \sum_{j=0}^{N-1} \widetilde{s}_j 2^{\log n \cdot j} \\ &= \sum_{j=0}^{N-1} \widetilde{c}_{j+1} 2^{(j+1) \log n} + \sum_{j=0}^{N-1} \widetilde{w}_j 2^{j \log n} \end{aligned} \tag{3.43}$$

Desde que ambos os $\widetilde{c}_{j+1}$ e $\widetilde{w}_j$ sejam $< 2^{\log n}$, a representação binária de $\sum_{j=0}^{N-1} \widetilde{c}_{j+1} 2^{(j+1) \log n}$ é simplesmente a concatenação dos bits $\widetilde{c}_N \widetilde{c}_{N-1} \cdots c_0$. Similarmente, concatenando os bits $\widetilde{w}_{N-1} \cdots \widetilde{w}_0$ para a representação binária de $\sum_{j=0}^{N-1} \widetilde{w}_j 2^{j \log n}$. Logo, pode ser observado que ocorre uma redução na soma de n números de n bits para a soma de dois números de $O(n)$ bits. Além disso, ao calcular todos os $\widetilde{c}_j$ e $\widetilde{w}_j$ em paralelo, é obtido como resultado dois números.

Só resta mostrar como calcular a representação em $2\log n$ bits de cada bloco de somatório $\tilde{s}_j = \tilde{c}_{j+1} 2^{\log n} + \tilde{w}_j$. Desde que cada número de somas seja da forma $\sum_{k=0}^{\log n - 1} 2^k x_k < n$, a soma total é um polinômio limitado em somas de $n \log n$ variáveis x_i, em que $1 \le i \le n$ e $(j-1)\log n < 1 \le j \log n$. Assim, cada bit de $\tilde{c}_{j+1}$ e $\tilde{w}_j$ é uma função polinomial limitada em $n \log n$ variáveis. Em outras palavras, cada bit $\tilde{c}_{j+1}$ e $\tilde{w}_j$ é uma função simétrica generalizada. Portanto, segue que cada bit do bloco de soma de $\tilde{s}_j$ pode ser representado como uma soma de polinômios com variáveis em x_{i_j}.

Na figura 3.9 é mostrado a técnica *block-save* calculando a soma de quatro inteiros de 16 bits. Como $n = 16$, a soma múltipla é dividida em 4 colunas.

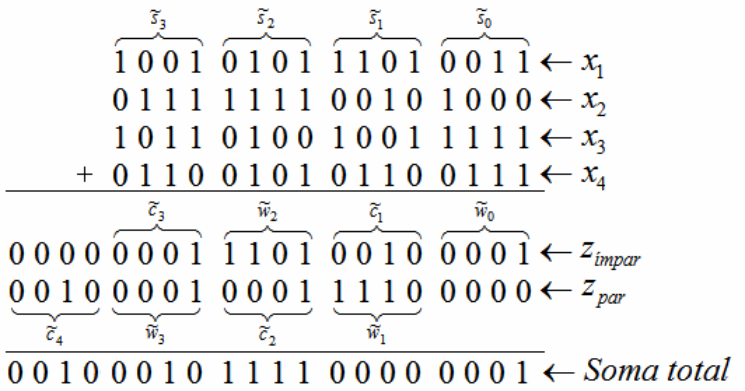

Figura 3.9 – Exemplo de cálculo da soma múltipla usando a técnica *block-save*.

Assim cada coluna é somada separadamente da seguinte forma:

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

Pode-se calcular z_{par} e z_{impar} utilizando um circuito de limiar linear de duas camadas.

- Bloco III – Soma de 2 números

Por fim, o bloco restante calcula a soma de $z_{par} + z_{ímpar}$, como mostrado na figura 3.10. É necessário um circuito de 2 camadas para calcular essa soma.

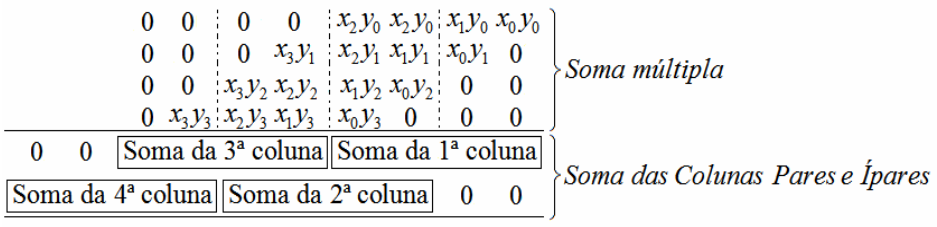

Figura 3.10 – Esquema de cálculo da soma múltipla usando a técnica *block-save*.

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 calcula o produto (AND lógico), bit a bit, entre os dois valores da entrada; a segunda e a terceira calcula z_{par} e $z_{ímpar}$; e a quarta e quinta calcula a soma $z_{par} + z_{ímpar}$. Considerando que a soma múltipla é uma combinação linear entre a segunda e a terceira camada [33], pode-se conectar um somador na saída da terceira camada, dessa forma só são necessárias quatro camadas para calcular a multiplicação.

3.6 – Conclusão

Neste capítulo foi apresentada a arquitetura da rede neural utilizada na implementação do multiplicador em corpos finitos. As portas de limiar linear, que é a estrutura básica dessa rede, possuem as portas tradicionais (AND, OR e NOT) como subconjunto, o que resulta em uma grande vantagem.

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, que fazem uso de técnicas algébricas, para construir circuitos de limiar linear. Por exemplo, a função paridade, que requer $2^{n-1}+1$ portas AON, pode ser calculada utilizando $\left\lceil \frac{n}{2} \right\rceil + 1$ portas de limiar linear. Ambas as arquiteturas possuem retardo equivalente a 2 portas. Caso seja necessário 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 é bem explorada em aplicações de adição, multiplicação e divisão em aritmética tradicional [18,19]. A possibilidade de utilizar essa potencialidade em outros sistemas numéricos tais como os corpos finitos, pode resultar em uma poderosa ferramenta na busca de circuitos mais eficientes no cálculo de operações aritméticas complexas.

4. Implementação do Multiplicador GF(2^4) com Portas de Limiar Linear

4.1 – Introdução

Como visto no capítulo 2, o multiplicador de Mastrovito possui uma das menores complexidades espaciais e temporais dentre os multiplicadores em paralelo, portanto sua arquitetura é a mais adequada para ser implementada com redes neurais discretas. Nesse capítulo é apresentada uma arquitetura de multiplicador em corpo finito utilizando portas de limiar linear como elemento de processamento. Inicialmente será projetado um multiplicador GF(2^4) com portas lógicas tradicionais, em seguida é feito o projeto do multiplicador GF(2^4) utilizando portas de limiar linear, desse modo será possível realizar uma comparação entre as duas arquiteturas.

4.2 – Multiplicador de Mastrovito com Portas AON

Como visto no capítulo 2, o multiplicador de Mastrovito é implementado em duas etapas: a multiplicação polinomial e a redução módulo $p(x)$. Essas etapas são detalhadas a seguir.

4.2.1 – Multiplicação Polinomial Ordinária

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

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

A equação 4.1 pode ser reescrita como:

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

Dessa forma, tem-se:

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

Em que cada elemento c'_i é dado por:

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

Independentemente do polinômio $p(x)$ escolhido, são necessárias n^2 portas AND para calcular todos os $a_i b_j$, em que $i = 1, \ldots, n$ e $j = 1, \ldots, n$. Para cada c_j, o número de elementos no somatório da equação 4.4 é dado por:

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

Como visto na seção 3.5, para implementar a função paridade com n variáveis e *fan-in* ilimitado são necessárias $2^{n-1}+1$ portas AON e retardo constante e igual a $2\tau_{AON}$, em que τ_{AON} corresponde ao retardo de uma porta 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'_i é:

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

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.

A implementação da primeira etapa do multiplicador GF(2^4) com $p(x) = x^4 + x + 1$, será realizada da seguinte forma:

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

Assim, cada elemento c'_i de $C'(x)$ é dado por:

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

Para calcular a multiplicação polinomial convencional de 2 elementos em GF(2^4) são necessárias 16 portas que implementam funções AND para o cálculo de cada $a_i b_k$ e 25 portas para implementar as somas. Essa etapa da multiplicação possui retardo igual a $3\tau_{AON}$.

4.2.2 – Redução Módulo *p(x)*

Esta etapa depende do polinômio irredutível escolhido, que para este caso foi escolhido um trinômio da forma $p(x) = x^n + x + 1$. Em [8,41] é mostrado que a escolha

de um trinômio irredutível leva a uma redução da complexidade do circuito do multiplicador.

A redução módulo $p(x)$ é realizada de acordo com a equação 4.2. Como é mostrado a seguir.

$$\begin{gathered} C(x) = C'(x) \bmod p(x) \\ (c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1}) = (c'_0 + c'_1 x + c'_2 x^2 + \cdots + c'_{n-1} x^{n-1}) \bmod (x^n + x + 1) \end{gathered} \tag{4.10}$$

Igualando os coeficientes dos dois lados da equação 4.10, tem-se que cada coeficiente c_i de $C(x)$ é dado por:

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

Calculando o número de portas necessárias para gerar cada termo c_i da equação 4.11, tem-se:

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

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

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

Logo, para o caso do multiplicador em GF(2^4) a redução módulo $p(x)$, mostrada na equação 4.14 com $p(x) = x^4 + x + 1$, são necessárias 16 portas AON.

$$C(x) = C'(x) \bmod p(x)$$

$$(c_0 + c_1 x + c_2 x^2 + c_3 x^3) = (c'_0 + c'_1 x + c'_2 x^2 + c'_3 x^3 + c'_4 x^4 + c'_5 x^5 + c'_6 x^6) \bmod (x^4 + x + 1) \quad (4.14)$$

$$(c_0 + c_1 x + c_2 x^2 + c_3 x^3) = (c'_0 + c'_4) + (c'_1 + c'_4 + c'_5) x + (c'_2 + c'_5 + c'_6) x^2 + (c'_3 + c'_6) x^3$$

Cada bit de $C(x)$ é dado por:

$$\begin{aligned} c_0 &= c'_0 + c'_4 \\ c_1 &= c'_1 + c'_4 + c'_5 \\ c_2 &= c'_2 + c'_5 + c'_6 \\ c_4 &= c'_3 + c'_6 \end{aligned} \quad (4.15)$$

Essas 16 portas AON mais as 41 portas AON necessárias para o cálculo da primeira etapa da multiplicação formam as 57 portas AON necessárias para a multiplicação em GF(2^4). Na figura 4.1 é mostrado um diagrama em blocos do processo de multiplicação em corpos finitos em *hardware*.

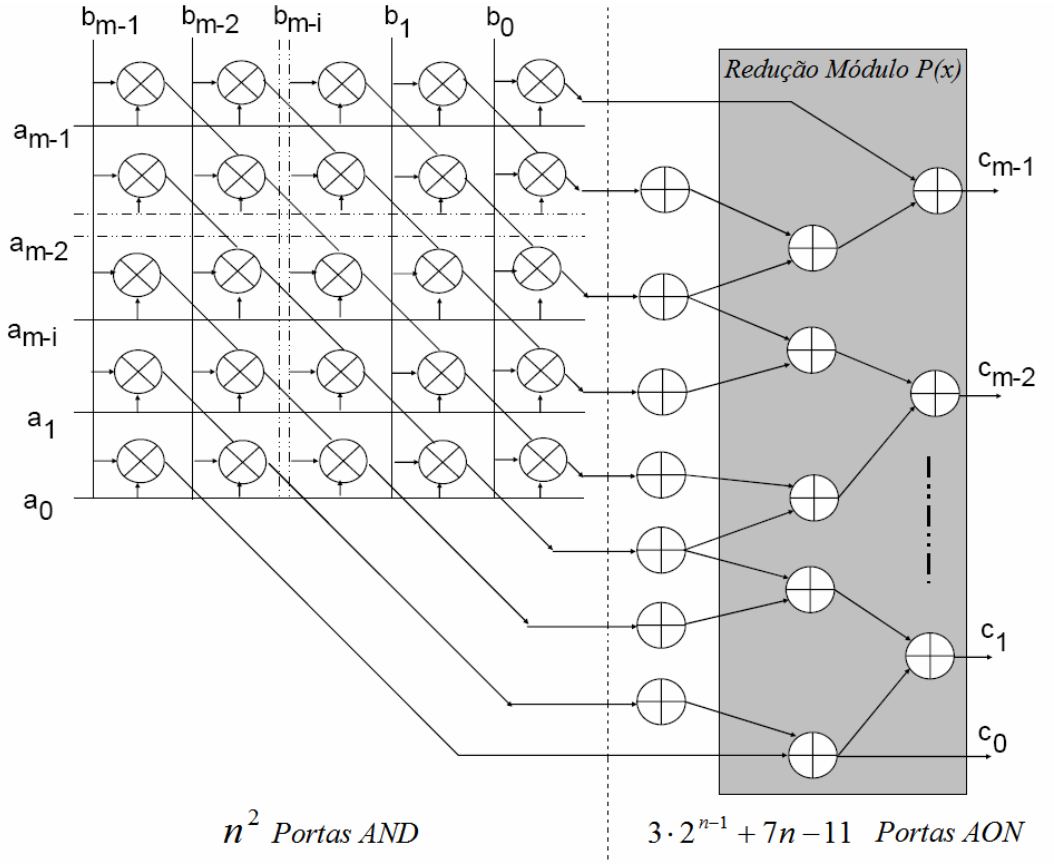

Figura 4.1 – Diagrama da implementação em *hardware* de multiplicadores em GF(2^n).

Os retardos de cada camada dessa arquitetura do multiplicador de Mastrovito com portas AON com *fan-in* ilimitado são:

$$\begin{cases} AND\ bit\ a\ bit = 1\tau_{AON} \\ Multiplicação\ Polinomial = 2\tau_{AON} \\ \text{Re}dução\ Módulo\ p(x) = 2\tau_{AON} \end{cases} \tag{4.16}$$

Dessa forma, somando os retardos produzidos por cada camada do multiplicador de Mastrovito é obtido o retardo total do multiplicador que é de $5\tau_{AON}$, em que τ_{AON} é o atraso correspondente a uma porta AON.

4.3 – Multiplicador de Mastrovito com Portas de Limiar Linear

Para o projeto do multiplicador em GF(2^4) com portas de limiar linear será utilizada, inicialmente a mesma técnica descrita por Mastrovito em [8], sendo que ao invés de utilizar portas AON serão utilizadas portas de limiar linear. Em seguida será aplicada a técnica telescópica para reduzir ainda mais a complexidade do multiplicador.

4.3.1 – Multiplicação Polinomial Ordinária

Nessa subseção será utilizado o mesmo procedimento da subseção 4.2.1 para o projeto de um multiplicador GF(2^n), porém substituindo as portas AON tradicionais por portas de limiar linear. Para a primeira camada do multiplicador são necessárias n^2 portas de limiar linear, cada uma realizando a função AND bit a bit para o cálculo da multiplicação de cada $a_i b_k$ da entrada.

Como visto na seção 3.5, 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 para implementar a multiplicação polinomial, com $n > 2$, é calculado a seguir:

$$\#TG = 2 \cdot \sum_{i=2}^{n} \left(\left\lceil \frac{i}{2} \right\rceil + 1 \right) - \left(\left\lceil \frac{n}{2} \right\rceil + 1 \right) \tag{4.17}$$

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

$$
\begin{aligned}
\#TG &= 2 \cdot \sum_{i=2}^{n} \left(\left\lceil \frac{i}{2} \right\rceil + 1 \right) + 2(n-1) - \frac{n}{2} - 1 \\
&= 2 \cdot \left(\frac{n^2}{4} + \frac{n}{2} - 1 \right) + 2(n-1) - \frac{n}{2} - 1 \quad (4.18) \\
&= \frac{n^2}{2} + \frac{5n}{2} - 5
\end{aligned}
$$

Para n ímpar, tem-se:

$$
\begin{aligned}
\#TG &= 2 \cdot \sum_{i=2}^{n} \left\lceil \frac{i}{2} \right\rceil + 2(n-1) - \frac{n}{2} - \frac{1}{2} - 1 \\
&= 2 \cdot \left(\frac{n^2}{4} + \frac{n}{2} - \frac{3}{4} \right) + \frac{3n}{2} - \frac{7}{2} \qquad (4.19) \\
&= \frac{n^2}{2} + \frac{5n}{2} - 5
\end{aligned}
$$

Portanto, para qualquer n, tem-se:

$$
\#TG = \frac{n^2}{2} + \frac{5n}{2} - 5 \qquad (4.20)
$$

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

Para projetar o multiplicador GF(2^4) com $p(x) = x^4 + x + 1$, primeiramente, é realizado um AND bit a bit entre os coeficientes de $A(x)$ e $B(x)$ utilizando portas de limiar linear, como mostrado na figura 4.2.

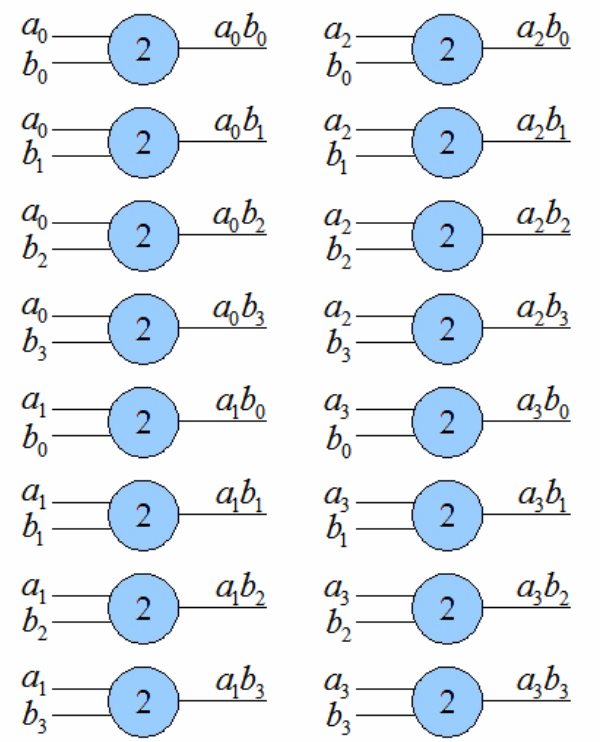

Figura 4.2 – Primeira camada do multiplicador em GF(2^4).

Portanto, para GF(2^4) são necessárias 16 portas de limiar linear que calculam um AND bit a bit para o cálculo de cada $a_i b_k$ da primeira camada do multiplicador. A segunda e terceira camada do multiplicador calcula os bits c'_i como é mostrado na figura 4.3.

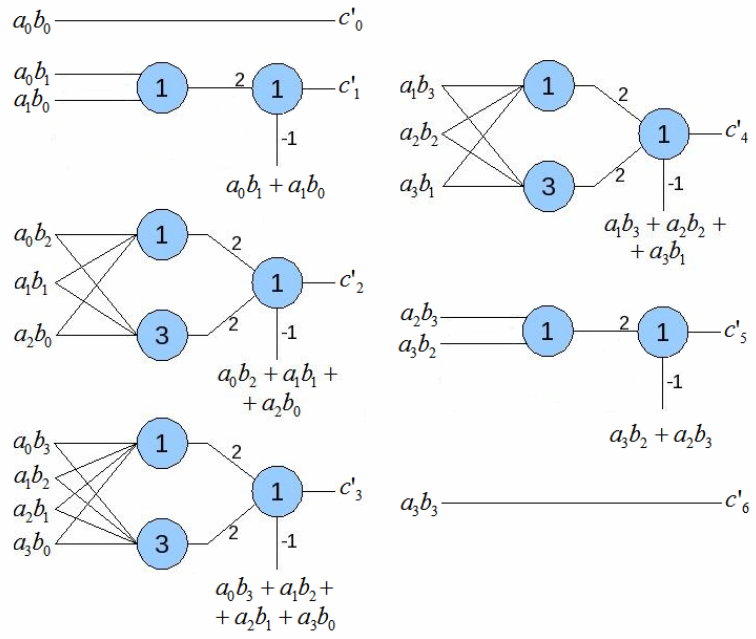

Figura 4.3 – Implementação do cálculo dos bits c'_i 's utilizando portas de limiar linear.

4.3.2 – Redução Módulo $p(x)$

Para realizar a redução módulo $p(x)$ é utilizada a equação 4.11. O número de portas de limiar linear é dado por:

$$\begin{aligned} \#TG &= 2 + \sum_{j=1}^{n-2} 3 + 2 \\ &= 2 + 3(n-2) + 2 \\ &= 3n - 2 \end{aligned} \tag{4.21}$$

Dessa forma, o número total de portas de limiar linear para um multiplicador em GF(2^n), utilizando redes neurais discretas, é dado por:

$$\#TG = \frac{3n^2}{2} + \frac{11n}{2} - 7 \tag{4.22}$$

A implementação dos bits c_i's do multiplicador GF(2^4) com portas de limiar linear é mostrada na figura 4.4.

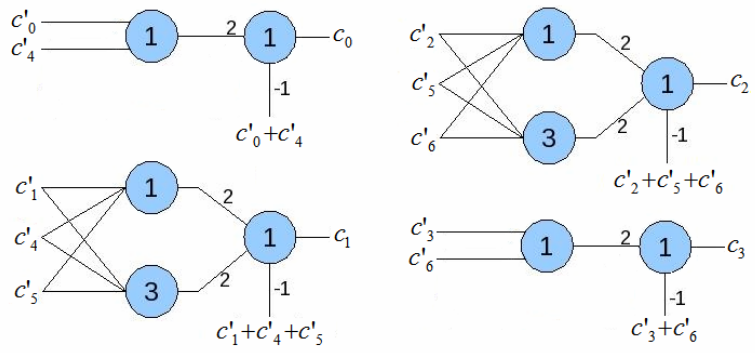

Figura 4.4 – Implementação do cálculo dos bits c_i's utilizando portas de limiar linear.

Observe que para o cálculo da redução módulo $p(x)$ são necessárias 2 camadas e, portanto, é produzido um atraso de $2\tau_{TG}$. Logo, o atraso total para o multiplicador GF(2^4) é de $5\tau_{TG}$. Portanto, é possível reduzir ainda mais a quantidade de portas de limiar utilizadas e o retardo total do multiplicador usando a técnica telescópica, como mostrada a seguir. Desenvolvendo a equação 4.10, é encontrada a seguinte expressão:

$$C(x) = (c'_0 + c'_n) + \sum_{j=1}^{n-2} (c'_j + c'_{j+n-1} + c'_{j+n}) x^j + (c'_{n-1} + c'_{2n-2}) x^{n-1} \tag{4.23}$$

Substituindo cada c'_j pelos seus respectivos valores, dados na equação 4.4, resulta na seguinte equação:

$$\begin{aligned} C(x) = & \left(a_0 b_0 + \sum_{i=1}^{n-1} a_i b_{n-i} \right) + \sum_{j=1}^{n-2} \left(\sum_{i=1}^{j} a_i b_{j-i} + \sum_{i=j}^{n-1} a_i b_{j+n-1-i} + \sum_{i=j+1}^{n-1} a_i b_{j+n-i} \right) x^j + \\ & + \left(\sum_{i=0}^{n-1} (a_i b_{n-1-i}) + a_{n-1} b_{n-1} \right) x^{n-1} \end{aligned} \tag{4.24}$$

O número de portas de limiar linear para o cálculo de cada coeficiente c_i da equação 4.24, é dado por:

$$\begin{aligned} \#c_0 &= 1 + n - 1 = n \\ \#c_j &= (j+1) + 2n - j - n + 1 - 1 + 2n - j - n - 1 \\ &= 2n - j \quad para\ j = 1, 2, \ldots, n-2 \\ \#c_{n-1} &= n + 2n - 2n + 2 - 1 = n + 1 \end{aligned} \tag{4.25}$$

Assim, o número total de portas de limiar linear necessárias para a multiplicação em corpos finitos é dado por:

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

De modo que para n par, tem-se:

$$\begin{aligned} \#TG &= 2n+1+\left(\frac{3n^2}{4}-n-1\right) \\ &= \frac{3n^2}{4}+n \end{aligned} \tag{4.27}$$

E para n ímpar, tem-se:

$$\begin{aligned} \#TG &= 2n+1+\left(\frac{3n^2}{4}n-\frac{3}{4}\right) \\ &= \frac{3n^2}{4}+n+\frac{1}{4} \end{aligned} \tag{4.28}$$

Somando os resultados das equações 4.27 e 4.28 com o número de portas da primeira camada, que é n^2 tem-se o total de portas necessárias para a multiplicação em GF(2^n), dado na equação 4.29:

$$\#TG = \begin{cases} \frac{7}{4}n^2+n & para\ n\ par \\ \frac{7}{4}n^2+n+\frac{1}{4} & para\ n\ ímpar \end{cases} \tag{4.29}$$

O multiplicador GF(2^4) com portas de limiar linear é projetado a partir do desenvolvimento da equação 4.23, resultando na seguinte expressão:

$$\begin{aligned} C(x) = {} & (a_0b_0 + a_1b_3 + a_2b_2 + a_3b_1) + (a_0b_1 + a_1b_0 + \\ & + a_1b_3 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2)x + \\ & + (a_0b_2 + a_1b_1 + a_2b_0 + a_2b_3 + a_3b_2 + a_3b_3)x^2 + \\ & + (a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0 + a_3b_3)x^3 \end{aligned} \tag{4.30}$$

Logo, cada bit de $C(x)$ é dado por:

$$\begin{aligned} c_0 &= a_0b_0 + a_1b_3 + a_2b_2 + a_3b_1 \\ c_1 &= a_0b_1 + a_1b_0 + a_1b_3 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2 \\ c_2 &= a_0b_2 + a_1b_1 + a_2b_0 + a_2b_3 + a_3b_2 + a_3b_3 \\ c_3 &= a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0 + a_3b_3 \end{aligned} \tag{4.30}$$

Para o cálculo da multiplicação de todos os $a_i b_j$ é utilizado o mesmo conjunto de portas da figura 4.2. Nas figuras 4.5, 4.6, 4.7 e 4.8 são mostradas os diagramas da implementação de cada bit c_i de $C(x)$.

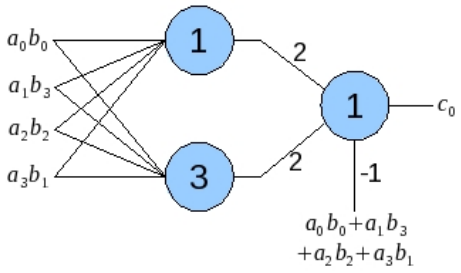

Figura 4.5 – Diagrama da implementação do *bit* c_0.

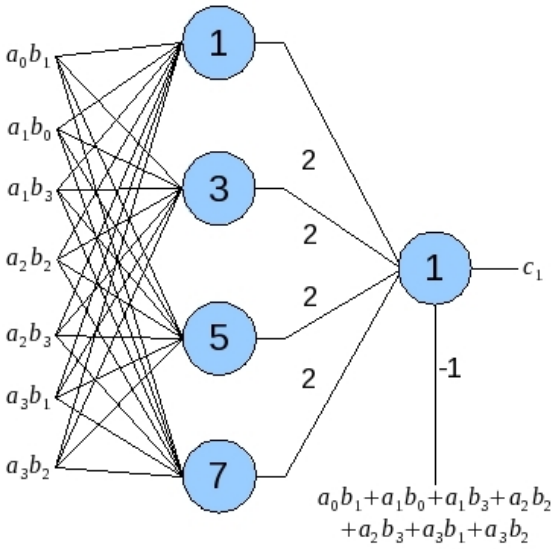

Figura 4.6 – Diagrama da implementação do *bit* c_1.

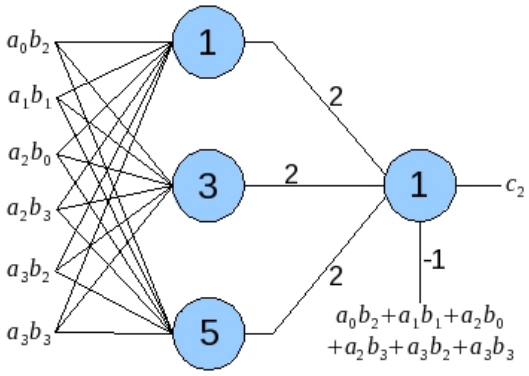

Figura 4.7 – Diagrama da implementação do *bit* c_2.

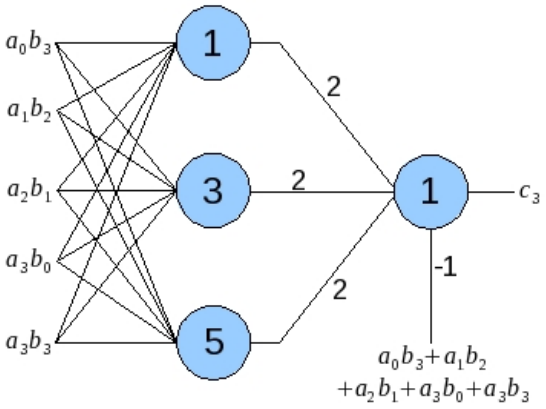

Figura 4.8 – Diagrama da implementação do *bit* c_3.

Como pode ser observado, o multiplicador em GF(2^4) possui três camadas. A primeira camada calcula a multiplicação bit a bit $a_i \bullet b_j$, para $i=0,...,3$ e $j=0,...,3$. A segunda e terceira camadas é um circuito que calcula a paridade com relação aos coeficientes a_ib_j de cada c_i.

4.4 – Conclusão

Na implementação de multiplicadores utilizando lógica tradicional, geralmente é encontrado na literatura circuitos com *fan-in* limitado. Isso é justificado para diminuir o tamanho do circuito, já que o tamanho de circuitos para multiplicação em corpos finitos cresce exponencialmente com o aumento do número de entradas. Como mostrado em [39], com a utilização de redes neurais discretas, ao tornar a profundidade do circuito ilimitada, é possível reduzir o tamanho de circuitos de exponencial para polinomial.

Dessa forma, ao utilizar arquiteturas de multiplicadores em corpos finitos projetados com portas de limiar linear é possível obter circuitos cujo tamanho não aumenta exponencialmente com o aumento do número de entradas. Assim, é possível obter circuitos com processamento rápido que utilizam menor quantidade de portas se comparado com arquiteturas de multiplicadores em corpos finitos que utilizam a lógica tradicional.

5. Leiaute do Multiplicador

5.1 – Introdução

Nesse capitulo, são apresentados os procedimentos realizados para o desenho do leiaute do multiplicador em GF(2^4), bem como os resultados de sua simulação. O leiaute foi projetado com as ferramentas de desenho de circuitos integrados da *Mentor Graphics*®, utilizando a tecnologia CMOS de 0,6 μm .

Para o desenho do leiaute do multiplicador GF(2^4), inicialmente, cada porta de limiar linear utilizada no multiplicador foi desenhada e testada. Em seguida foi desenhado o leiaute da primeira camada do multiplicador e verificado seu funcionamento. O próximo passo foi o leiaute da segunda e terceira camadas, que implementam a função paridade. Após o teste de todas as camadas separadamente e verificação de seu funcionamento, as mesmas foram interligadas em um único leiaute formando o multiplicador em GF(2^4). Nas próximas seções são mostrados os leiautes e os resultados da simulação.

5.2 – Implementação da Porta de Limiar Linear

Há muitas formas de implementação de portas de limiar linear em CMOS [23-30], também chamadas de portas lógicas de limiar (*Threshold Logic Gates* – TLG). No projeto de portas lógicas de limiar há características importantes a serem consideradas, tais como velocidade de operação, máximo somatório-de-pesos, área ocupada e potência dissipada.

A porta lógica de limiar utilizada nesse trabalho para implementar o multiplicador é proposta em [24] e mostrada na figura 5.1. A razão para essa escolha é a sua facilidade de integração. Esta porta, chamada *Latched Comparator Threshold Logic gate*, realiza a comparação entre as correntes nos pontos *x* e *y*, em outras palavras, um conjunto de transistores MOS que implementam o somatório das entradas formam uma rede de condutância que é comparada com outra rede de condutância similar formada por transistores MOS que implementam o limiar.

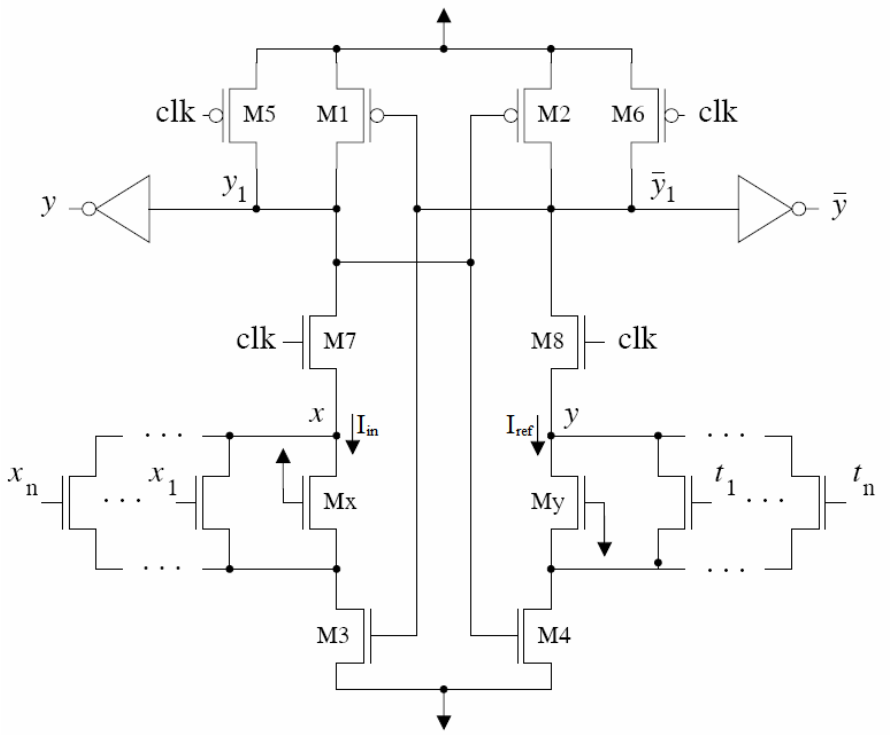

Figura 5.1 – Esquema elétrico da porta de limiar baseada no comparador *latch*.

Essa porta de limiar é baseada em um comparador do tipo *latch* com fases de pré-carga e avaliação. O elemento básico do comparador é um *latch* controlado por corrente, formado pelos pares de transistores M1/M3 e M2/M4, que fornecem a saída da porta e seu complemento. Os pares de transistores M5/M7 e M6/M8 controlam as fases de pré-carga e avaliação. As entradas da porta de limiar, $x_1,\ldots,x_n$, são conectadas nas respectivas portas de um conjunto de transistores NMOS idênticos e em paralelo que representam o banco de transistores da entrada. De modo similar, os sinais do limiar, $t_1,\ldots,t_n$, são conectados às respectivas portas de um outro conjunto de transistores idênticos e em paralelo que representam o banco de transistores do limiar. Os dois transistores adicionais, M_x e M_y, garantem a operação correta da porta no ponto em que o valor do somatório das entradas multiplicadas por seus respectivos pesos for igual ao valor do limiar.

A fase de pré-carga ocorre quando o sinal *clk* está em um nível lógico baixo. Nessa fase os transistores M5 e M6 estão ligados, enquanto que os transistores M7 e M8 estão desligados; isso faz com que as saídas y_1 e $\bar{y}_1$ fiquem em nível lógico alto. A fase de avaliação ocorre quando *clk* está em nível lógico alto. Nessa fase os transistores M5 e M6 serão desligados e os transistores M7 e M8 serão ligados; fazendo com que os nós y_1 e $\bar{y}_1$ iniciem seu processo de descarga. Dependendo do valor do limiar, $t_1 \ldots t_n$, e do

número de entradas, $x_1,\ldots,x_n$, em nível lógico alto, podem ocorrer duas situações: a corrente no ponto x ser maior que a corrente no ponto y, $I_{in} > I_{ref}$, ou pode acontecer de a corrente no ponto x ser menor que a corrente no ponto y, $I_{in} < I_{ref}$. Quando $I_{in} > I_{ref}$, um nível lógico alto em y_1 é obtido e quando $I_{in} < I_{ref}$, um nível lógico baixo é obtido em y_1. As correntes I_{in} e I_{ref} só circulam nos pontos x e y durante os períodos de transição do sinal *clk*, consequentemente essa porta de limiar não consome potência estática. Na figura 5.2 é mostrado o leiaute da porta de limiar implementada com duas entradas. O tamanho de cada transistor é descrito na tabela 5.1.

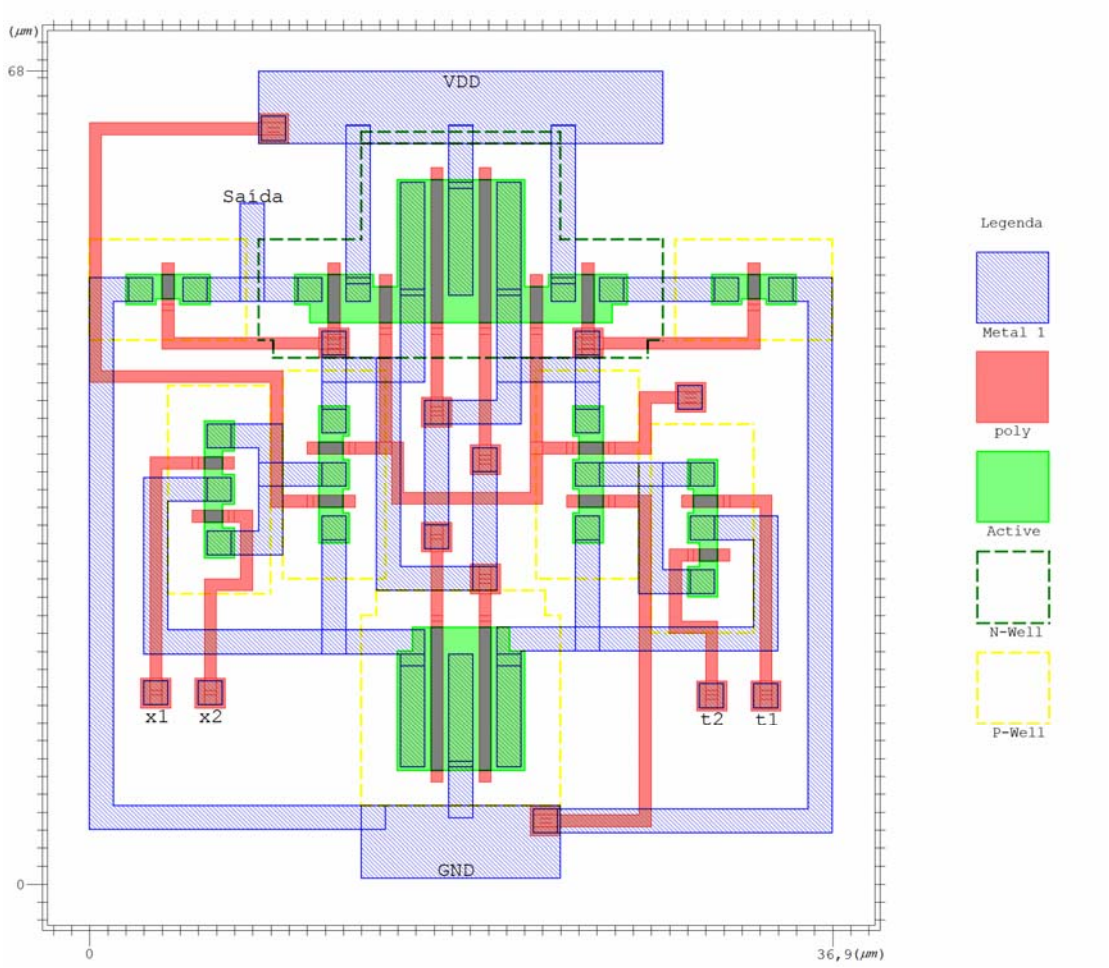

Figura 5.2 – Leiaute da porta de limiar linear com duas entradas.

Transistores	Razão de Aspecto (W/L)
M5 e M6	6/1
M7 e M8	2/1
M1 e M3	12/1
M2 e M4	12/1
M_x e M_y	4/4
M_{x1} a M_{xn} e M_{t1} a M_{tn}	4/2

Tabela 5.1 – Razão W/L dos transistores da porta de limiar linear implementada.

O leiaute da porta de limiar foi simulado, primeiramente, implementando uma função AND de duas entradas, para isso foram colocados os níveis 11 nas entradas do limiar ($t_1 t_2$). Para implementar a função OR, as entradas de limiar, $t_1 t_2$, foram carregadas com os níveis 10. Para alimentação do circuito foi utilizada uma fonte de tensão de 5 volts. Para o sinal *clk* foi utilizada uma onda quadrada com período de 20MHz. Os gráficos da simulação do leiaute da porta de limiar estão mostrados nas figuras 5.3 e 5.4.

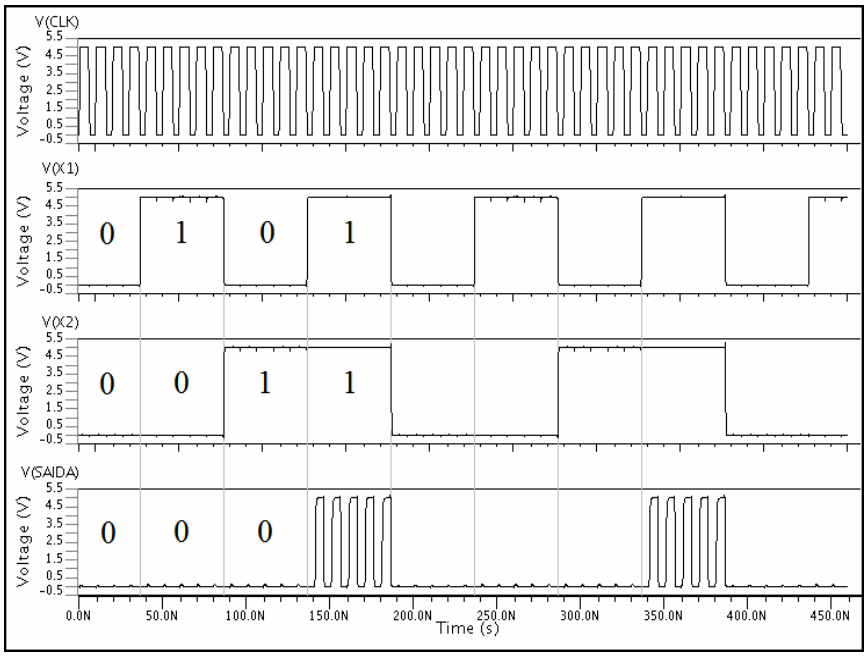

Figura 5.3 – Simulação do leiaute da porta de limiar implementando uma função AND.

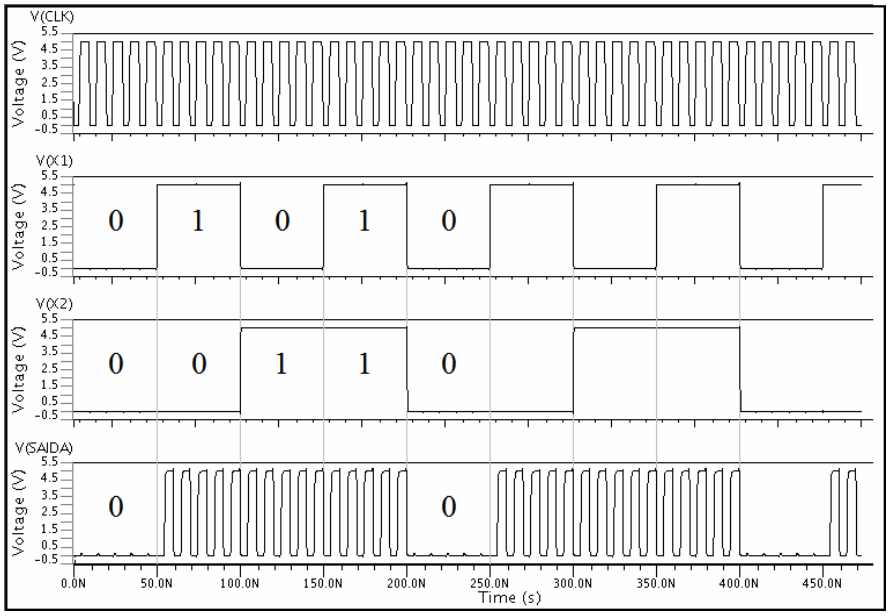

Figura 5.4 – Simulação do leiaute da porta de limiar implementando uma função OR.

Os resultados da simulação são descritos na tabela 5.2.

Área Ocupada	Potência Dissipada	Tempo de Processamento
2509,2 μm^2	376,04 pW	1,9 ns

Tabela 5.2 – Resultados da simulação do leiaute da porta de limiar com duas entradas.

A porta de limiar linear implementada em [24] utiliza a tecnologia CMOS 0,7 μm e o mesmo dimensionamento dos transistores. Porém, para a implementação da porta de limiar nesse trabalho foi utilizada a tecnologia de 0,6 μm, devido ao fato de que o único processo disponível pelo *Mentor*, que chegasse mais próximo da tecnologia CMOS de 0,7 μm, foi o processo AMI 0,5, que possui o menor valor de λ (unidade do leiaute) igual a 0,3 μm, além do mais as simulações do leiaute demonstraram um bom funcionamento da porta de limiar com essa tecnologia. Portanto, o menor comprimento do canal para os transistores utilizados no leiaute do presente trabalho é de 0,6 μm.

Pode-se notar que, apesar dessa estrutura de porta de limiar linear não implementar entradas com peso negativo, é possível obtê-las. Observe que, de acordo com a equação 3.2, que descreve o comportamento da porta de limiar linear, o sinal do limiar t é negativo, portanto, para implementar entradas com pesos negativos, as mesmas terão de ser colocadas no banco de transistores que implementam o limiar.

5.3 – Leiaute do Multiplicador em GF(2^4) Utilizando Portas de Limiar Linear

Para o desenho do leiaute do multiplicador, cada porta de limiar utilizada para realizar a multiplicação em GF(2^4) foi desenhada separadamente e testada. Em seguida foi desenhado o leiaute de cada camada do multiplicador. A primeira camada é constituída de 16 portas de limiar linear de duas entradas que implementam a função AND para o cálculo da multiplicação bit a bit de cada a_i e b_j da entrada do multiplicador. O leiaute da primeira camada é mostrado na figura 5.5 e os resultados da simulação mostrados na tabela 5.3.

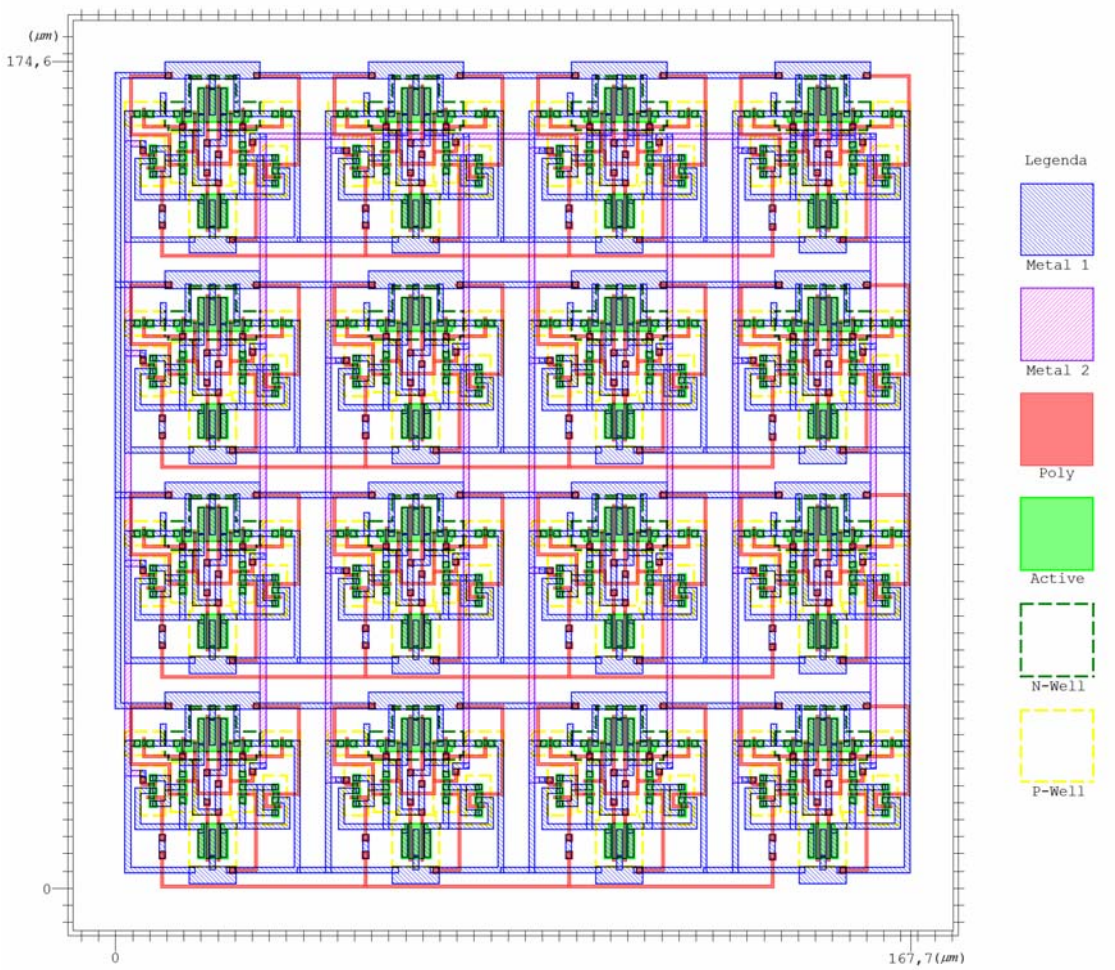

Figura 5.5 – Leiaute da primeira camada do multiplicador em GF(2^4).

Área Ocupada	Potência Dissipada	Tempo de Processamento
29280,42 μm^2	6,02 nW	2,2 ns

Tabela 5.3 – Resultados da simulação do leiaute da 1ª camada do multiplicador em GF(2^4).

A segunda e a terceira camada do multiplicador em GF(2^4) implementam, juntas, a função de paridade, cujo número de entradas depende do bit do multiplicador a ser calculado. As funções de paridade de 4, 7, 6 e 5 variáveis calculam, respectivamente, os bits c_0, c_1, c_2 e c_3 do multiplicador em GF(2^4). A terceira camada do circuito do multiplicador em GF(2^4) pode ser considerada a camada mais complexa em termos de implementação, pois compõe portas com pesos negativos e entradas com peso igual a dois. Para implementar os pesos negativos, como já foi dito, a entrada correspondente que tem peso negativo, ao invés de ser ligada no banco de transistores de entrada da porta, é ligada no banco de transistores do limiar. Como por exemplo, a porta usada na segunda camada do circuito que implementa a função paridade de 7 variáveis possui quatro entradas com peso 2 e no banco de transistores do limiar contém 8 transistores. Um desses transistores implementa o limiar, que vale 1 para essa porta, e os outros 7 transistores calculam a soma $a_0b_1 + a_1b_0 + a_1b_3 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2$ que possui peso negativo, Como pode ser visto no diagrama de implementação do bit c_1, mostrado na figura 4.5. Já o peso 2 é obtido dobrando, em relação aos outros transistores com entradas de peso unitário, a largura (W) do transistor correspondente a entrada com peso 2. O leiaute dessa porta com as entradas indicadas é mostrado na figura 5.6.

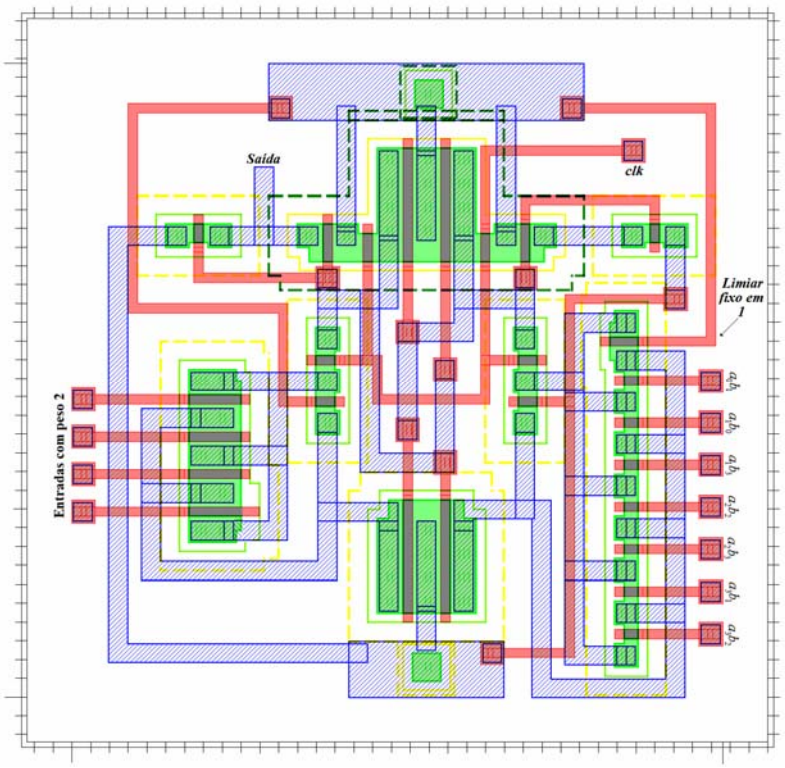

Figura 5.6 – Leiaute da porta de limiar da segunda camada da função paridade de 7 variáveis.

Com as simulações dessa porta de limiar linear, observou-se que a medida que o número de entradas da porta aumentava, o efeito das capacitâncias parasitas, associadas aos transistores de entrada, também aumentava, fazendo com que a porta não funcionasse da forma correta. Esse problema se deve ao efeito de Miller sobre as capacitâncias parasitas entre porta e dreno. Esse problema foi solucionado aumentando a largura (W) dos transistores de entrada, assegurando uma baixa impedância no dreno $\left(\approx \frac{1}{g_m}\right)$ [42], porém, esse aumento não deve ser tão grande para não provocar o funcionamento incorreto da porta de limiar. O leiaute do circuito que implementa a função paridade de 7 variáveis é mostrado na figura 5.7.

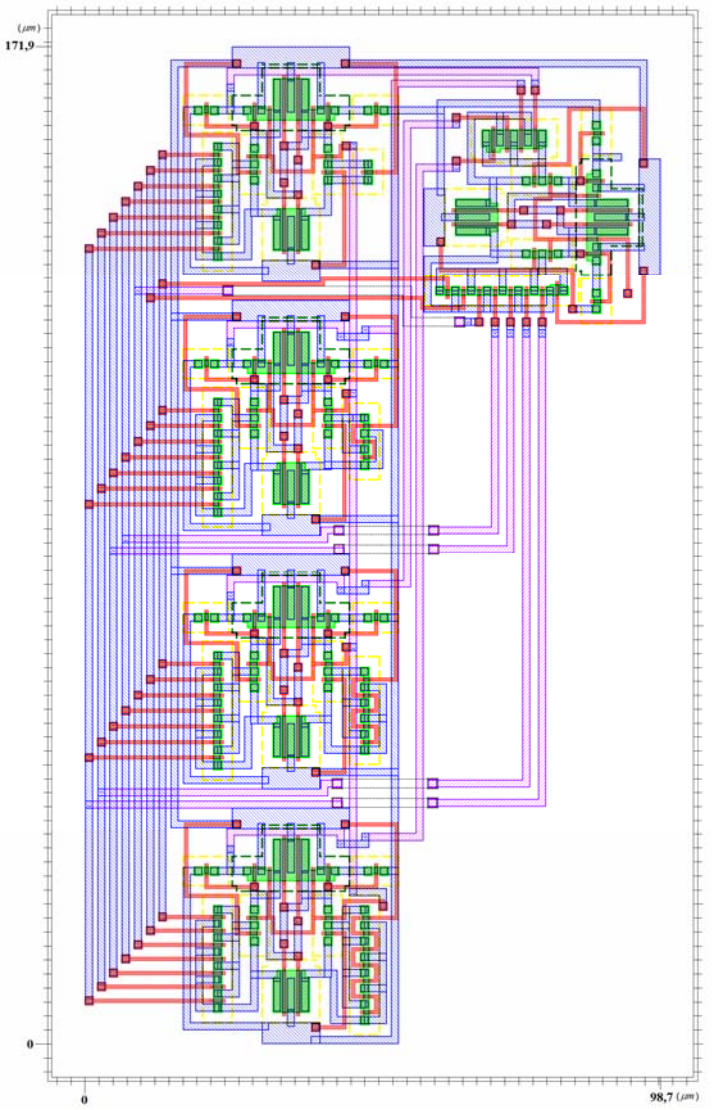

Figura 5.7 – Leiaute do circuito da função paridade de 7 variáveis.

Os resultados da simulação do leiaute da figura 5.7, estão descritos na tabela 5.4.

Área Ocupada	Potência Dissipada	Tempo de Processamento
16966,53 μm^2	1,88 nW	6,2 ns

Tabela 5.4 – Resultados da simulação do leiaute da função paridade de 7 variáveis.

Com os leiautes de cada camada do multiplicador pronto, o mesmo foi construído interligando cada uma das camadas gerando o leiaute da figura 5.8.

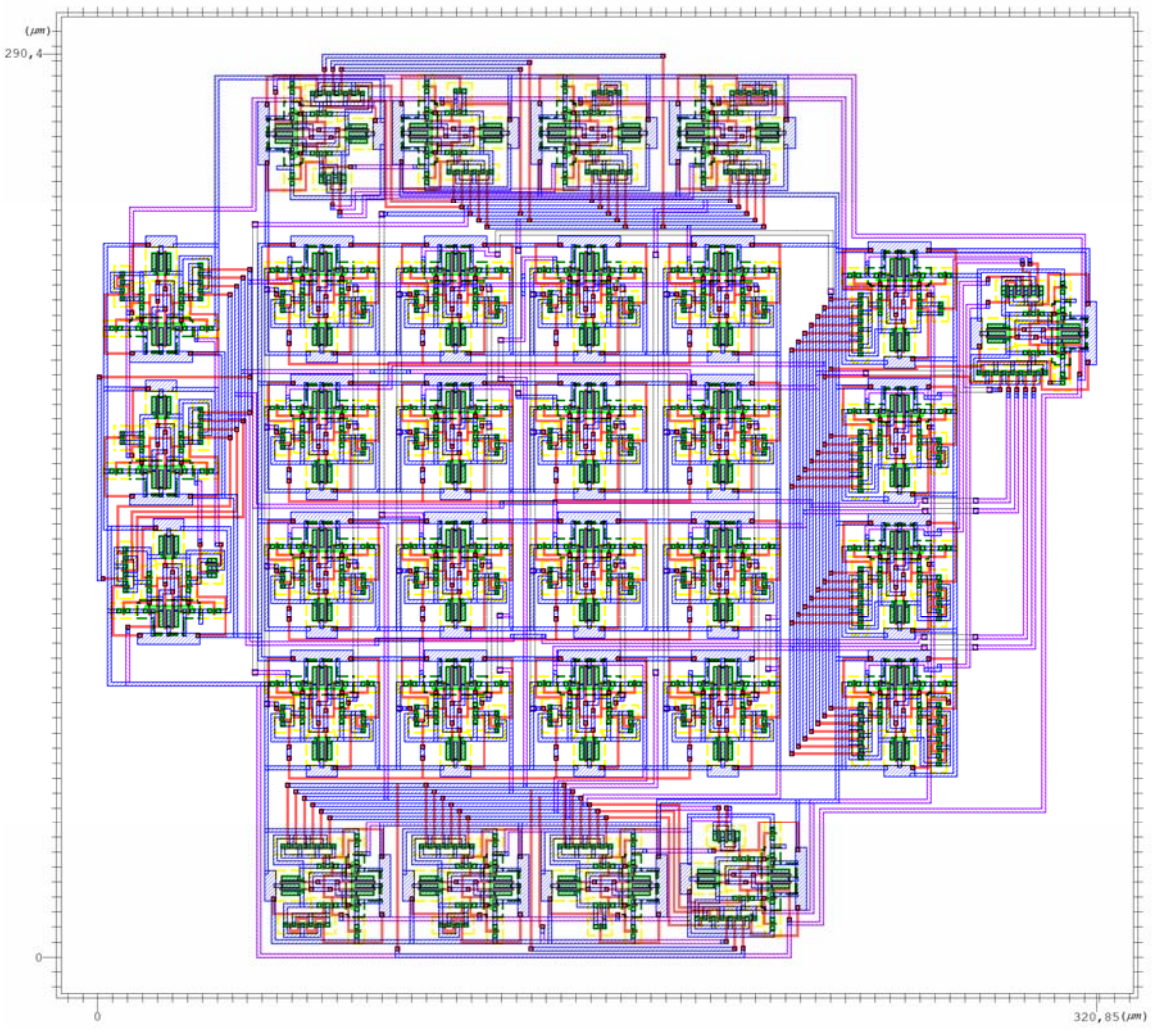

Figura 5.8 – Leiaute do multiplicador em GF(2^4).

Os resultados da simulação do multiplicador em GF(2^4), estão descritos na tabela 5.5

Área Ocupada	Potência Dissipada	Tempo de Processamento
93174,84 μm^2	12,03 nW	8,5 ns

Tabela 5.5 – Resultados da simulação do multiplicador em GF(2^4).

5.4 – Conclusão

Nesse capítulo foram descritos os passos desenvolvidos para a construção do leiaute do multiplicador em GF(2^4). Também foram apresentados os resultados das simulações de cada leiaute. Com isso, o objetivo de comprovar o funcionamento prático da arquitetura do multiplicador em GF(2^4) , proposto em [33], foi cumprido, entretanto, sua eficiência, em termos de tempo de processamento e área ocupada em *chip* está abaixo do esperado.

Como foi dito no capítulo 4, para o projeto do multiplicador em GF(2^4) utilizando portas de limiar linear, são necessárias 32 portas, contra as 57 portas se for utilizada a lógica tradicional. Portanto, levando-se em conta o número de portas, a lógica neural é melhor que a lógica tradicional, na implementação do multiplicador em GF(2^4). Entretanto, essa não é uma comparação justa. Se levarmos em conta que para implementar uma porta AND tradicional são utilizados 6 transistores, enquanto que uma porta de limiar linear, implementando uma função AND, utiliza 18 transistores; se a comparação entre o projeto do multiplicador em GF(2^4) utilizando a lógica tradicional e o mesmo multiplicador utilizando a lógica neural, for feita com relação a quantidade de transistores necessários, a lógica neural perde, já que seriam necessários 384 transistores para implementar a multiplicação em GF(2^4) com a lógica tradicional, contra os 657 transistores necessários para implementar o mesmo multiplicador usando a lógica neural. Portanto, o multiplicador em GF(2^4) construído com portas de limiar linear ocupa uma área em *chip* maior do que o mesmo multiplicador utilizando a lógica tradicional. Além disso, em geral a lógica tradicional utiliza transistores com dimensões mínimas.

Se comparamos as duas arquiteturas de multiplicação em GF(2^4) com relação ao tempo de processamento, a lógica neural também não é melhor, apesar de sua capacidade de processamento paralelo ser maior. Em [4], foi construído em FPGA, um multiplicador em GF(2^4) baseado na arquitetura de Mastrovito com a lógica tradicional, cujo tempo de processamento é de 6,481 *ns* , contra os 8,5 *ns* da arquitetura apresentada nesse trabalho.

Vale salientar que as arquiteturas que utilizam lógica neural dependem da tecnologia empregada na implementação das portas de limiar linear, ou seja, se o mesmo multiplicador em GF(2^4), implementado nesse trabalho, for construído com portas de limiar mais rápidas e que ocupem uma área menor em *chip*, a lógica neural

será mais eficiente. Além do mais, a grande vantagem na construção de circuitos utilizando redes neurais discretas, está no fato de que esse circuito terá uma complexidade polinomial. Se o mesmo circuito for construído com a lógica tradicional terá complexidade exponencial. Dessa forma é possível obter uma arquitetura do multiplicador de Mastrovito com uma complexidade espacial menor.

Na tabela 5.6 é mostrada uma comparação entre a implementação do multiplicador de Mastrovito construído com a lógica tradicional, ou seja, utilizando portas AON e o mesmo multiplicador construído com portas de limiar linear (TG), para diferentes valores de n. A tabela 5.6 mostra também a quantidade de transistores utilizados nas duas arquiteturas. Para o cálculo do número de transistores foi considerado uma média na qual uma porta AON possui 6 transistores e uma porta de limiar linear possui 18 transistores. Para o cálculo do número de portas de cada implementação foram utilizadas as equações 4.13 e 4.29.

n	Número de Portas		Número de Transistores	
	AON	TG	AON	TG
8	493	120	2958	2160
10	1695	185	10170	3330
12	6361	264	38166	4752
14	24859	357	149154	6426
16	98661	464	591966	8352

Tabela 5.6 – Comparação das arquiteturas de multiplicadores em GF(2^n).

De acordo com a tabela 5.6, a partir de $n=8$ já se observa a vantagem em termos de complexidade espacial, já que a quantidade de transistores da implementação com portas de limiar é menor que a quantidade de transistores da implementação com portas AON.

6. Conclusão e Trabalhos Futuros

6.1 – Conclusões

Nesse trabalho foi apresentado o projeto do leiaute de uma nova arquitetura de multiplicador em corpos finitos, tal arquitetura utiliza as portas de limiar linear como elemento de processamento em substituição da lógica tradicional que usa portas AND, OR e NOT. A porta de limiar linear é o elemento básico de uma rede neural discreta, cuja utilização visa diminuir a complexidade espacial e temporal dos circuitos para multiplicação em corpos finitos. A arquitetura de um multiplicador em GF(2^4) utilizando portas de limiar linear, apresentada em [33], foi implementada nesse trabalho, dando origem a um leiaute de um circuito. O circuito projetado possui alimentação de 5V e opera na freqüência de 20 MHz. O consumo de potência do circuito é de 12,03 nW e o tempo de processamento para uma multiplicação é de 8,5 ns.

Para a realização do projeto, foi necessário um estudo dos conceitos relacionados à multiplicação em corpos finitos, bem como as arquiteturas existentes para implementação em *hardware*. Foi necessário ainda, realizar um estudo de algumas das implementações existentes de portas de limiar linear para daí escolher a porta de limiar linear mais viável para o projeto.

Deve-se observar que os resultados, obtidos com as simulações do leiaute do multiplicador em GF(2^4), demonstram que o mesmo possui tempo de processamento um pouco maior e ocupa quase que o dobro da área em chip que os multiplicadores em GF(2^4) construídos com a lógica tradicional. Porém, com o avanço da tecnologia novas portas de limiar, que ocupam uma menor área no *chip* e um menor tempo de processamento, serão projetadas. Desse modo, o projeto de multiplicadores em corpos finitos, que utilizam portas de limiar como elemento de processamento, se torna bem promissor.

6.2 – Trabalhos Futuros

Como sugestão para a seqüência deste trabalho, recomenda-se o projeto de um multiplicador maior, como por exemplo, um GF(2^8) que possui aplicações na área de

códigos corretores de erros, ou até mesmo um multiplicador em GF(2^{233}), que é um dos multiplicadores padrões proposto pelo NIST (*National Institute of Standards and Technology*).

Uma outra sugestão para trabalhos futuros é projetar o multiplicador em GF(2^4) com outros tipos de portas de limiar, como por exemplo, a portas de limiar feitas com DG-MOSFET, que são MOSFET em escala de nanômetros e com porta dupla, como descrito em [44].

Apêndice A - Complexidade

Se não existir limite em tamanho de hardware é possível implementar qualquer circuito com portas AND, OR e NOT. 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 necessita 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 retardo no calculo de funções.

Dessa forma, tem-se o seguinte problema: dada uma certa função, qual a quantidade mínima de hardware necessária para sua implementação? Para responder a essa questão é necessário conhecer a complexidade da função a ser calculada, bem como qual arquitetura será utilizada para implementar o circuito

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. 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 função é o estudo da complexidade computacional.

Várias medidas de complexidade, como tamanho do circuito, *fan-in*, *fan-out*, etc, podem ser definidas para um determinado modelo de circuito. Nesse apêndice são discutidos alguns conceitos de complexidade de circuitos.

A.1 – Notações Assintóticas

Uma forma de caracterizar uma função é por meio da sua taxa de crescimento. Alguns símbolos 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 executam a mesma função. Cada símbolo é descrito a seguir:

Definição A.1. A notação "O Grande"

Sejam $f(x)$ e $g(x)$ funções reais definidas no conjunto dos números naturais, isto é, $f: N \to R$ e $g: N \to R$. Dizemos que $f(x)$ é $O(g(x))$ se existe um número positivo M tal que, a menos de um número finito de exceções,

$$|f(x)| \leq M|g(x)| \qquad ()$$

Em outras, palavras dizer que $f(x)$ é $O(g(x))$ significa que $|f(x)|$ não supera um múltiplo constante de $|g(x)|$ (com algumas exceções eventuais). Por exemplo, seja $f(x) = \binom{x}{2}$, afirmando que $f(x)$ é $O(x^2)$. Recordemos que $\binom{x}{2} = x(x-1)/2$; assim,

$$f(x) = \frac{x(x-1)}{2} \leq \frac{x^2}{2} \qquad ()$$

para todo x. Podemos, pois, tomar $M = \frac{1}{2}$ na definição de O Grande e concluir que $f(x)$ é $O(x^2)$.

Quando dizemos que $f(x)$ é $O(g(x))$, a função $g(x)$ atua como uma cota para $|f(x)|$. Isto é, $|f(x)|$ não cresce mais rapidamente do que um múltiplo de $|g(x)|$.

Definição A.2. A notação Ω

A notação "O Grande" estabelece uma cota superior para o crescimento de $|f(x)|$. Reciprocamente, a notação Ω (ômega grande) define uma cota inferior para esse crescimento.

Sejam $f(x)$ e $g(x)$ funções com valores reais definidas no conjunto dos números naturais, isto é, $f: N \to R$ e $g: N \to R$. Dizemos que $f(x)$ é $\Omega(g(x))$ se existe um número positivo M tal que, a menos de um número finito de exceções,

$$|f(x)| \geq M|g(x)| \qquad ()$$

Existe uma relação simples entre as notações O e Ω. Sejam $f(x)$ e $g(x)$ funções de $N \to R$. Então $f(x)$ é $O(g(x))$ se e somente se $g(x)$ é $\Omega(f(x))$. A notação O é uma cota superior e Ω é uma cota inferior. A notação a seguir combina as duas. O símbolo Θ é a letra grega maiúscula "Theta".

Definição A.3. A notação Θ

Sejam $f(x)$ e $g(x)$ funções com valores reais definidas no conjunto dos números naturais, isto é, $f : N \to R$ e $g : N \to R$. Dizemos que $f(x)$ é $\Theta(g(x))$ se existem números positivos A e B tais que, a menos de um número finito de exceções,

$$A|g(x)| \leq |f(x)| \leq B|g(x)| \qquad ()$$

Em outras palavras, a afirmação de que $f(x)$ é $\Theta(g(x))$ nos diz, com efeito, que, quando x cresce, $f(x)$ e $g(x)$ crescem aproximadamente à mesma taxa.

Definição A.4. A notação "*o* Pequeno"

Sejam $f(x)$ e $g(x)$ funções com valores reais definidas no conjunto dos números naturais, isto é, $f : N \to R$ e $g : N \to R$. Dizemos que $f(x)$ é $o(g(x))$ se e somente se

$$\lim_{n \to \infty} \frac{f(x)}{g(x)} = 0 \qquad ()$$

A afirmação de que $f(x)$ é $O(g(x))$ nos diz que $f(x)$ não cresce mais depressa do que $g(x)$, quando x se torna grande. Às vezes, convém dizer que $f(x)$ cresce "muito" mais devagar do que $g(x)$. Para isso, dispomos da notação "*o* Pequeno"

A.2 – *Fan-in/Fan-out*

Uma das definições importante para o estudo de complexidade de circuitos é o conceito de *fan-in/fan-out*. O número de conexões que entra em uma porta é chamado de *fan-in* e 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.

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 do circuito em questão. Nesse trabalho de dissertação não há restrições no *fan-in/fan-out* dos circuitos de limiar linear utilizados. Como redes neuroniais são caracterizadas por um massivo processamento paralelo, assumir um modelo de circuito de limiar linear com *fan-in/fan-out* não fixo é justificado.

A.3 – Complexidade Espacial

A quantidade de *hardware* necessária para implementação física de circuitos corresponde a complexidade espacial ou o tamanho do circuito. É óbvio que a complexidade espacial para executar uma determinada função depende do tipo de tecnologia utilizada para a implementação. Intuitivamente, quanto mais poderosa for o tipo de porta lógica utilizada melhor será o desempenho 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, se existir uma constante $k > 0$ tal que $S(n) = O(n^k)$. Caso o seu tamanho seja $\Omega(2^{n^{\varepsilon}})$ para um valor fixo de $\varepsilon > 0$, então o circuito possui tamanho exponencial.

Em geral, para saber se a 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 calculada por meio de circuitos AON com profundidade $O(\log n)$, tamanho $O(n)$ e *fan-in* limitado. Entretanto, a mesma necessita de tamanho exponencial para ser executada por circuitos com 2 camadas e *fan-in* ilimitado. Caso seja utilizado circuitos de limiar linear com *fan-in* ilimitado, a mesma poderá ser executada com circuito de 2 camadas e tamanho $O(n)$.

A.4 – Complexidade Temporal (Profundidade)

A complexidade temporal ou profundidade do circuito corresponde ao tempo necessário para calcular 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.

Referências Bibliográficas

[1] C. Shannon, “A mathematical theory of communication,” *Bell System Tech*. J. 27, pp. 379–423 e 623–656, 1948.

[2] Todd K. Moon, Error Correction Coding: Mathematical methods and algorithms, John Wiley and Sons, New Jersey, 2005.

[3] M. A. Hasan, “Look-up table-based large finite field multiplication in memory constrained cryptosystems,” IEEE Transactions on Computers, vol. 49, no. 7, pp. 749–758, July 2000.

[4] E. R. Berlekamp, “Bit-serial Reed-Solomon encoders,” IEEE Trans. on Information, 28(6), pp. 869-874, 1982.

[5] C. S. Yeh, S. Reed, and T. K. Truong, “Systolic multipliers for finite fields GF(2^m),” IEEE Trans. Computers, vol. 33, no. 4, pp. 357 – 360, Apr. 1984.

[6] J. L. Massey and J. K. Omura, “Computational method and apparatus for finite field arithmetic,” U. S. Patent Number 4,587,627, May 1986.

[7] T. Itoh and S. Tsujii, “Structure of parallel multipliers for a class of fields GF(2^m),” Information and Computation, vol. 83, pp. 21-40, 1989.

[8] E. D. Mastrovito, “VLSI architectures for computation in Galois field,” PhD thesis, Linkoping Univ., Linkoping, Sweden, 1991.

[9] C. Paar, “Efficient VLSI architectures for bit-parallel computation in Galois field,” PhD thesis, Institute for Experimental Mathematics, University of Essen, Essen Germany, June 1994.

[10] 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.

[11] C.-Y. Lee, E.-H. Lu, and J.-Y. Lee, “Bit-parallel systolic multipliers for GF(2^m) fields defined by all-one and equally spaced polynomials,” IEEE Trans. Computers, vol. 50, pp. 385-392, May 2001.

[12] Imaña J. L., Sánchez J. M., “Multiplicadores canônicos sobre campos GF(2^m) generados por um tipo de trinômios irreducibles implementados em hardware reconfigurable,” III Jornadas sobre Computacion Reconfigurable y Aplicaciones – JCRA2003, pp. 65–72, 2003.

[13] Edgar Ferrer, Dorothy Bollman, Oscar Moreno, “A fast finite field multiplier,” Reconfigurable Computing: Architectures , Tools and Applications, Third International Workshop, ARC 2007, Mangaratiba, Brazil, pp. 238–246, March 2007.

[14] Vladimir T. Olaya, Julio C. L. Hernandez, Jaime V. Medina, “Design of polynomial basis multipliers over GF(2233),” XIII Workshop Iberchip, Peru, 2007.

[15] Turki F. Al-Somani, Alaaeldin Amin, “Hardware implementation of GF(2^m) arithmetic using normal basis,” Journal of Applied Sciences, vol. 6, issue 6, pp. 1362–1372, 2006.

[16] Huapeng Wu, M. Anwarul Hasan, Ian F. Blake, Shuhong Gao, “Finite field multiplier using redundant representation,” IEEE Trans. Computers 51(11), pp. 1306–1316, 2002.

[17] Ashkan H. Namin, Huapeng Wu, Majid Ahmadi, “High speed word-parallel bit-serial normal basis finite field multiplier and its FPGA implementation,” Proc. 39th Asilomar Conf. Signals, Systems, and Computers, pp. 1338–1341, November 2005.

[18] M. Paterson, N. Pippenger, and U. Zwick, “Faster circuits and shorter formulae for multiple addition, multiplication and symmetric Boolean functions,” In Proceedings of the 31st Annual Symposium on Foundations of Computer Science, pp. 642-650, 1990.

[19] R. Paturi and M. Saks, “On threshold circuits for parity,” In Proceedings of the 31st Annual Symposium on Foundations of Computer Science, pp. 397-404, 1990.

[20] Lidiano A. N. Oliveira, 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. 744–65, Setembro 1997.

[21] Y. Amit, “A neural network architecture for visual selection,” Neural Computation, vol. 12, no. 5, pp. 1141–1164, 2000.

[22] Lidiano A. N. Oliveira, F. M. Assis, “Arquitetura eficiente de um multiplicador de Mastrovito usando redes neurais discretas,” Anais do III Congresso Brasileiro de Redes Neurais, vol. 1, Santa Catarina, Brasil, pp. 12–15, Julho 2007.

[23] T. Shibata and T. Ohmi, “An intelligent MOS transistor featuring gate-level weighted sum and threshold operation,” IEDM, Technical Digest, IEEE, New York, December 1991.

[24] M. Avedillo, J. Quitana, A. Rueda, and E. Jiménez, “Low-power CMOS threshold-logic gate,” IEE Electronics Letters 31, pp. 2157–2159, December 1995.

[25] K. Kotani, T. Shibata, M. Imai, and T. Ohmi, “Clocked-neuron-MOS logic circuits employing auto-threshold-adjustment,” ISSCC Digest of Technical Papers, pp. 320–321, 1995.

[26] H. Özdemir, A. Kepkep, B. Pamir, Y. Leblebici, and U. Çiliniroglu, “A capacitive threshold-logic gate,” IEEE JSSC 31, pp. 1141–1149, August 1996.

[27] S. Bobba and I. Hajj, “Current-mode threshold logic gate,” in Proceedings of the IEEE International Conference on Computer Design, Austin, pp. 235–240, 2000.

[28] P. Celinski, J. F. López, S. AL-Sarawi, and D. Abbott, “Low power, high speed, charge recycling CMOS threshold logic gate,” IEE Electronics Letters 37, , pp. 1067–1069, August 2001.

[29] J. F. Ramos, J. A. H. Lopez, M. J. Martin, J. C. Tejero, and A. Gago, “A threshold logic gate base don clocked coupled inverters,” International Journal of Electronics, 84(4), pp. 371-382, 2001.

[30] M. Padure, S. Cotofana, and S. Vassiliadis, “A low-power threshold logic family,” in Proc. IEEE International Conference on Electronics, Circuits and Systems, pp. 657–66, 2002.

[31] Peter Celinski, Sorin D. Cotofana, and Derek Abbot, “Threshold logic parallel counters for 32-bit multipliers,” Proceedings of SPIE, vol. 4935, pp. 205–214, 2002.

[32] Y. Leblebici, H. Özdemir, A. Kepkep, and U. Çilingiroglu, “A compact high-speed (31,5) parallel counter circuit based on capacitive threshold-logic gates,” IEEE Journal of Solid-State Circuits, vol. SC-31, pp. 1177–1183, 1996.

[33] Lidiano A. N. Oliveira, “Multiplicador em corpo finito utilizando redes neurais discretas,” Dissertação de Mestrado, Universidade Federal de Campina Grande, Campina Grande, PB, Brasil, Outubro de 2000.

[34] Rudolf Lidl and Harald Niederreiter, Introduction to finite fields and their applications, Cambridge University Press, Cambridge – UK, Revised Edition, 1994.

[35] Mario Alberto G. Martinez, “Construción de operadores básicos sobre campos finitos GF(2^m),” Tese de Doutorado, México, Dezembro 2004.

[36] Jean-Pierre Deschamps, José L. Imaña, Gustavo D. Sutter, Hardware implementation of finite-field arithmetic, McGraw-Hill, United States, 2009

[37] W. S. McCulloch and W. Pitts. “A logical calculus of ideas immanent in nervous activity”. Bulletin of Mathematical Biophysics”, Vol. 5, pp. 115–133, 1943.

[38] Peter Celinski, Sorin D. Cotofana, José F. López, Said Al-Sarawi, Derek Abbott, “State-of-art in CMOS threshold-logic VLSI gate implementations and applications,” Proceedings of the VLSI Circuits and Systems Conference, vol. 5117, Spain, pp. 53–64, 2003.

[39] K-Y Siu, V. Roychowdhury, T Kailath, Discrete Neural Computation: A theoretical Foundation, Prentice-Hall, 1995.

[40] Lidiano A. N. Oliveira, F. M. de Assis, “A new architecture for multipliers in GF(2^n) using discrete neural networks,” Proceedings of IV Brazilian Conference on Neural Networks, São José dos Campos – SP, Brazil, pp. 888–999, July 1999.

[41] Ç. K. Koç, B Sunar, “Mastrovito multiplier for all trinomials,” IEEE Transactions on Computers, vol. 48, no. 5, pp. 522–527, 1999.

[42] Elkim Felipe Roa Fuentes, “Metodologia de Projeto para Amplificadores de Baixo Ruído em CMOS”, Dissertação de Mestrado, Universidade de São Paulo, 2003.

[43] Captain G. C. Ahlquist, Brent Nelson, and Michael Rice, “Optimal Finite Field Multipliers for FPGAs”, Lecture Notes in Computer Science;vol.1673, pp. 51–60, 1999.

[44] Darwin Ta-Yueh Ting, “Reconfigurable Threshold Logic Gates Implemented in Nanoscale Double-Gate,” PhD thesis, Russ College of Enginnering and Technology of Ohio University, Ohio, EUA, August 2008.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo