UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Gabriel Bruno Monteiro Fernandes

Infraestrutura automática para aritmética computacional baseada em RNS

Florianópolis
2021

Gabriel Bruno Monteiro Fernandes

Infraestrutura automática para aritmética computacional baseada em RNS

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Engenharia Elétrica.
Orientador: Prof. Hector Pettenghi Roldan, Dr.

Florianópolis
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Fernandes, Gabriel Bruno Monteiro
Infraestrutura automática para aritmética computacional baseada em RNS / Gabriel Bruno Monteiro Fernandes ; orientador, Hector Pettenghi Roldan, 2021.
90 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia Elétrica, Florianópolis, 2021.

Inclui referências.

1. Engenharia Elétrica. 2. Sistema Numérico Residual. 3. Circuitos Aritméticos. 4. Computação de Alta Performance. 5. Arquiteturas Paralelas. I. Roldan, Hector Pettenghi. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

Gabriel Bruno Monteiro Fernandes

Infraestrutura automática para aritmética computacional baseada em RNS

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Cleumar da Silva Moreira , Dr.
Instituto Federal da Paraíba

Prof.Roberto de Matos, Dr.
Instituto Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Engenharia Elétrica.

Documento assinado digitalmente
Telles Brunelli Lazzarin
Data: 12/11/2021 21:03:10-0300
CPF: 023.400.719-25
Verifique as assinaturas em https://v.ufsc.br

Prof. Telles Brunelli Lazzarin, Dr.
Coordenação do Programa de
Pós-Graduação

Documento assinado digitalmente
Hector Pettenghi Roldan
Data: 11/11/2021 14:54:41-0300
CPF: 068.054.601-40
Verifique as assinaturas em https://v.ufsc.br

Prof. Hector Pettenghi Roldan, Dr.
Orientador

Florianópolis, 2021.

Este trabalho é dedicado aos meus aos meus pais José Raimundo e Jenete e à minha noiva Daniella.

AGRADECIMENTOS

Agradeço, a meu orientador, Prof. Dr. Hector Pettenghi Roldan pela transmissão de seu vasto conhecimento em *design* digital, método científico e incentivo pela exploração de novas temáticas fora de minha zona de conforto.

Aos meus pais, pelo incentivo aos meus sonhos mesmo que isso se traduza em eventuais ausências.

A Daniella, minha noiva, a quem atribuo a completa mudança de minha vida. Não apenas pelo amor e carinho diário, nem pela inseparabilidade ou carreira que seguimos juntos e por isso usufruímos de conselhos mútuos. Mas sim, por ter influenciado e mudado quem sou hoje.

“Learn what is to be taken seriously and laugh at the rest.”
"Aprenda aquilo que deve ser levado a sério, do resto, apenas gargalhe"
(Herman Hesse, 1877-1962)

RESUMO

O projeto de hardware para sistemas de processamento digital de sinais tem recebido atenção considerável nas últimas décadas. O crescimento da demanda por arquiteturas com alta eficiência energética, segurança e alto nível de paralelismo elevou o interesse em circuitos aritméticos baseados em sistemas de representação numérica não convencionais. Um destes sistemas é o *Residue Number System* (RNS), o qual oferece soluções arquiteturais com alta velocidade de computação, efetuando cálculos livres de *carry*. A técnica consiste na decomposição de um número inteiro em um conjunto de valores independentes, também chamados de resíduos. Com estes é possível realizar operações como adição, multiplicação, acumulação e subtração, de forma paralela. Sistemas RNS são frequentemente interfaceados com sistemas binários comuns. Neste sentido, dada a necessidade de troca de informações entre as porções binária e RNS, tem-se como interface entre blocos, um conversor binário para RNS como entrada, também chamado de conversor direto, e um conversor RNS para binário como elemento de saída, intitulado de conversor reverso. O objetivo deste trabalho é propor uma arquitetura eficiente a qual poderá ser aplicável tanto na implementação de conversores RNS diretos e reversos, como também em operações aritméticas modulares existentes em aplicações de DSP (*Digital Signal Processing*). A proposta baseia-se em uma nova tendência de solução a qual faz uso de unidades aritméticas compressoras sendo implementadas de forma similar à uma árvore de Wallace. Tal tendência também traz a reinserção de bits de *carry-out* à própria árvore, com o intuito de evitar o uso de *Lookup Tables*, o qual apresenta aumento exponencial em área a medida que utiliza-se um número de bits elevado. Tais arquiteturas terão foco em conjuntos modulares $2^n \pm k$ os quais encontram-se fora da faixa de uso convencional baseada nos valores $2^n, 2^n - 1, 2^n + 1$, com o intuito de futuras investigações de aplicações com ampla faixa dinâmica. Resultados experimentais demonstram possíveis ganhos no atraso de computação de até 12% nas distintas comparações entre o método desenvolvido e circuitos do estado da arte. Por fim, com as arquiteturas construídas, foi realizada uma nova proposta de seleção de conjuntos modulares a partir da fatoração da propriedade $\prod_{i=1}^{M} m_i = 2^{\alpha} - 1$. É demonstrado que tal fatoração pode proporcionar balanceamento entre canais aritméticos e conversores reversos por meio da aplicação de pipeline e prover possíveis ganhos de área e atraso quando comparado a conjuntos de módulos comumente empregados em projetos RNS.

Palavras-chave: Sistema Numérico Residual (RNS), Circuitos Aritméticos, Computação de Alta Performance, Arquiteturas Paralelas, Representações Numéricas.

ABSTRACT

Hardware design for digital signal processing has received considerable attention in recent decades. The growing demand for architectures with characteristics such as high energy efficiency, security and high level of parallelism has raised interest in arithmetic circuits based on non-conventional numerical representations. One approach that fits in such tendency is the Residue Number System (RNS), which offers architectural solutions with carry free operations leading to high computation speed. The technique consists of decomposing an integer into a set of independent values, also called residues. With these residues it is possible to perform operations such as addition, multiplication, accumulation and subtraction, in a parallel scheme. RNS systems are often interfaced with common binary systems. In this sense, given the need to exchange information between the binary and RNS blocks, their interface is composed by a binary to RNS converter, also called direct converter, and an RNS to binary converter, that can be refered to as reverse converter. The objective of this dissertation is to propose an efficient architecture which can be applied both in the implementation of forward and reverse RNS converters, as well as in modular arithmetic operations existing in DSP (Digital Signal Processing) applications. The proposal is based on a new trend which makes use of arithmetic units being implemented similarly to a Wallace tree. This trend also brings the reinsertion of carry-out bits that are fed back into the tree, in order to avoid the use of Lookup Tables. The objective of avoiding these Tables is due to their exponential increase in area as the number of bits increases. Such architectures will focus on modular sets in the form of $2^n \pm k$ which fall outside the conventional range of the most used values of $2^n, 2^n \breve{} 1, 2^n + 1$. The intent behind such choice is to enable future investigations of applications with wide dynamic range. Experimental results show possible gains in circuit delays of up to 12% in the different comparisons between the developed method and state-of-the-art architectures. Finally, by using the built architectures, a new proposal for the selection of modular sets was conducted based on the factorization of the property $\prod_{i=1}^{M} m_i = 2^{\alpha} - 1$. It is demonstrated that such factorization can provide critical path balance between arithmetic channels and reverse converters through the application of a pipeline, providing possible gains of area and delay when compared to modules sets commonly used in RNS projects.

Keywords: Residue Number System (RNS), Arithmetic Circuits, High-Performance Computation, Parallel architecture, Number Representation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CLA	Carry Look Ahead Addition
CPA	Carry Propagate Addition
CPG	Carry Propagate and Generate
CRT	Chinese Remainder Theorem
CSA	Carry Save Addition
CSD	Canonic Signed-Digit
EAC	End around Carry
FA	Full Adder
HA	Half Adder
LUTs	Look Up Tables
MRC	Mixed Radix Conversion
MSD	Minimal Signed-Digit
RNS	Residue Number System
ROM	Read Only Memory
SAC	Sum and Carry
TGA	The Greedy Approach
VLSI	Very Large Scale Integration

SUMÁRIO

1 INTRODUÇÃO

A popularidade de aplicações implementadas em sistemas embarcados portáteis, tais como telefones celulares e câmeras digitais aumentou significativamente ao longo dos anos. Esses dispositivos fazem uso frequentemente de baterias como fontes de energia e requerem sistemas de computação sofisticados com alto desempenho e baixa dissipação de energia para executar suas funcionalidades.

Não somente isto, mas o advento do conceito de internet das coisas gerou uma maior demanda de conexão remota entre dispositivos, forçando a necessidade de segurança, processamentos mais rápidos, armazenamento de dados e redução no consumo de potência.

Tais sistemas de computação digital são projetados convencionalmente utilizando sistema de numeração binária. Entretanto, com o intuito de buscar a melhora de desempenho, outros sistemas de numeração podem ser explorados, por exemplo, o Sistema Numérico Residual (em inglês, Residue Number System (RNS)). Os primeiros sistemas RNS foram implementados inicialmente por volta de 1960 (MERRILL, 1964) (GUFFIN, 1962), e consequentemente vieram como solução para aplicações em que era necessária a realização de várias multiplicações e adições de forma eficiente. Desde então, pesquisas vêm sendo realizadas para superar certos obstáculos e aumentar a eficiência em questões como escalabilidade, aumento da faixa dinâmica, operações de comparação e conversão direta e reversa. Soluções para criptografia (BAJARD; IMBERT, 2004) (MEHRABI; DOCHE; JOLFAEI, 2020), comunicação (YANG; HANZO, 2002) (HAI *et al.*, 2017) e filtros digitais (CARDARILLI; NANNARELLI; RE, 2000) (LYAKHOV *et al.*, 2020) são recorrentes na literatura e demonstram ganhos em eficiência bastante expressivos.

Operações modulares na forma $2^n, 2^n - 1, 2^n + 1$, onde n equivale ao número de bits por canal, possuem alta eficiência e receberam extensa atenção em publicações (ZIMMERMANN, R., 1999). Estes simplificam amplamente as operações modulares e as conversões entre sistemas de numeração residual e binário. Entretanto, a exploração de diversos módulos fora da fração indicada prova-se necessária para ampliação de faixa dinâmica e melhoras nos níveis de paralelismo.

Desta forma, a motivação principal deste trabalho é explorar arquiteturas na forma $2^n \pm k$. Tais arquiteturas serão baseadas em compressores, tendo similaridades à uma árvore de Wallace e tratarão bits maiores que 2^n sendo reinseridos na própria árvore. O intuito do uso de tal estratégia é de prover uma solução alternativa às arquiteturas mais amplamente difundidas, as quais baseiam-se no uso de Tabelas de pesquisa (em inglês, Look Up Tables (LUTs)). Apesar de eficientes para canais com um número pequeno de bits, o uso de LUTs torna-se bastante ineficiente à medida em que tal número de bits cresce dependendo de uma determinada aplicação.

1.1 OBJETIVO GERAL

O objetivo deste trabalho é propor uma metodologia com um intuito de atingir uma alocação eficiente de compressores, baseados em *carry-save adders*, para circuitos aritméticos modulares. Neste contexto, será proposta uma arquitetura eficiente, sendo esta aplicável tanto na implementação de conversores RNS diretos e reversos, bem como para operações aritméticas modulares existentes em aplicações de DSP. A proposta deve se basear em uma nova tendência de solução a qual faz uso de unidades aritméticas compressoras sendo implementadas de forma similar a uma árvore de Wallace e que traz a reinserção de bits de *carry-out* à própria árvore, com o intuito de evitar o uso de LUTs. Tal arquitetura terá foco em conjuntos modulares fora da faixa convencional $2^n, 2^n-1, 2^n+1$ com a finalidade de explorar aplicações com ampla faixa dinâmica. Nesse sentido, são elencados abaixo os objetivos específicos para realização de tal tarefa.

1.1.1 Objetivos Específicos

1. Dado que a arquitetura apresentada possui o objetivo de comprimir árvores de informação, o circuito apresentado será composto por dois blocos fundamentais, sendo eles o bloco de compressão e outro de soma final. Tendo em vista estes dois blocos colocam-se os objetivos:
    - Estabelecer um método de compressão a fim de reduzir o atraso em circuitos para operações modulares baseados no uso de compressores;
    - Otimizar unidades de soma modular a fim de garantir soma final eficiente;
2. Aplicar o método em circuitos de conversão direta e reversa, bem como multiplicação modular, para avaliação do método de compressão desenvolvido;
3. Avaliar desempenho de multiplicadores modulares $2^n \pm k$ quando comparados aos $2^n \pm 1$, sendo tal comparação útil para avaliações de escolhas modulares visando ampliação de faixas dinâmicas
4. Avaliar aplicação de pipeline para otimização de camadas aritméticas.

1.2 CONTRIBUIÇÕES

Todos os circuitos mencionados serão aprofundados ao decorrer do texto. Entretanto, em resumo, arquiteturas para realização de operações modulares baseadas em LUTs, as quais fazem uso principalmente de memórias, possuem o custo de área elevado a medida que um projeto demande uso de um grande número de bits. Estas

memórias são necessárias para tratamento de bits que saem dos bloco de compressão. Uma solução para este problema é o uso da reinserção dos bits do bloco de compressão ao próprio bloco de circuito que os gera. Tal proposta foi primeiramente apresentada em (ELLEITHY; BAYOUMI, M.A., 1995).

Já em (PATRONIK; PIESTRAK, S. J., 2017) foi realizada uma nova proposta a qual, além de implementar a reinserção de bits, traz uma solução para o bloco de compressão baseada apenas em componentes aritméticos como Full Adders, Half adders e compressores 5:3. Na tese (PALUDO, 2020) foi realizada uma adaptação da implementação de tal trabalho citado, com o intuito de avaliar arquiteturas para conversores reversos. A tese gerou uma ferramenta de software a qual é responsável pela construção automática dos circuitos utilizados no respectivo trabalho. A atual dissertação de mestrado tem como objetivo ampliar a ferramenta gerada dando foco em módulos $2^n \pm k$ para melhora do atraso de computação de tais circuitos.

Uma segunda contribuição será o de estabelecer um circuito de soma final para os módulos $2^n \pm k$ garantindo a melhor associação de compressão modular com o bloco de soma final, propondo assim, uma arquitetura completa para aplicações em RNS.

1.3 METODOLOGIA

Como indicado, este trabalho teve como ponto de partida a ferramenta desenvolvida em (PALUDO, 2020), o qual por sua vez, foi baseado no método descrito em (PATRONIK; PIESTRAK, S. J., 2017). Visto que a ferramenta atingiu resultados bastante eficientes, quando comparado ao trabalho que o deu origem, serão apresentados apenas os resultados baseados em (PALUDO, 2020). Chamaremos então tal ferramenta inicial de ferramenta versão 0, enquanto que as arquiteturas próprias e otimizadas serão atribuídas a ferramenta versão 1. Os circuitos são geradas de forma automática na linguagem VHDL, pelas ferramentas e mapeadas na biblioteca de *Standard Cells* de 65 nm da UMC (TSAI; ZHOU, 2006). A ferramenta Genus Synthesis Solution (versão 16.24-S065-1) foi configurada para aplicar seus recursos de otimização para o determinado processo de síntese.

2 FUNDAMENTAÇÃO TEÓRICA

Os aprimoramentos realizados nas tecnologias VLSI renovaram o interesse na exploração do sistema numérico residual nas últimas duas décadas. O RNS oferece suporte às principais propriedades e recursos de design VLSI, as quais podem ser elencadas como:

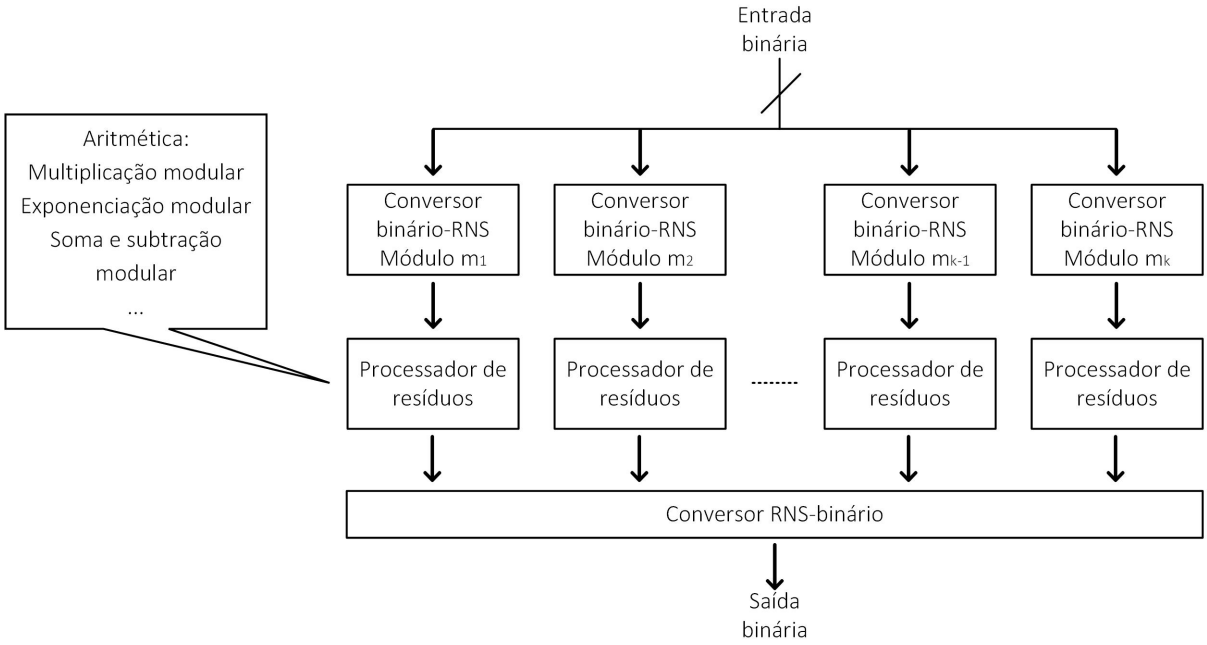
1. Congruência: implementações RNS possuem paralelismo inerente onde as operações aritméticas são realizadas de forma independente (para cada módulo) em canais separados.
2. Modularidade: O RNS oferece modularidade tanto em níveis funcionais quanto a nível de layout, uma vez que a estrutura de processamento para cada módulo é semelhante em arquitetura e desempenho (BAYOUMI, M.; JULLIEN, G.; MILLER, 1986).
3. Tolerância a falhas: Dígitos podem ser adicionados para expandir o comprimento de palavras, ou excluídos devido a falhas de hardware, sem comprometimento da legibilidade dos dados. Além disso, erros podem ser isolados facilmente.

Entretanto, o sistema de numeração residual não possui eficiência máxima para todos os tipos de operações comumente empregadas em sistemas digitais. Os maiores ganhos são alcançados para operações como soma, subtração, multiplicação e exponenciação (MEHER; STOURAITIS, Thanos, 2017) enquanto que operações de comparação, divisão, detecção de sinal e *overflow* ainda permanecem como elementos com baixo desempenho.

Desta forma, sistemas RNS são frequentemente interfaceados com sistemas binários comuns, onde operações eficientes na forma modular são aceleradas, enquanto que as demais operações genéricas ao sistema são computadas pelo bloco binário comum. Uma exemplificação de tal sistema é exposta pela Figura 1. Uma vez que dados precisam ser comunicados entre as porções binária e RNS, tem-se como interface entre blocos, um conversor binário para RNS. Este também é conhecido como conversor direto, cujas X palavras de saída correspondentes a X módulos do conjunto modular. A partir de então, estes módulos serão processadas por X blocos paralelos de processamento de resíduos produzindo o mesmo número de palavras de saída. Por fim, ocorre o processo de conversão reversa para o sistema binário.

Entre os blocos apresentados, o de conversão reversa consiste em um dos mais complexos, existindo uma quantidade elevada de pesquisas para desenvolvimento de arquiteturas eficientes e conjuntos de módulos apropriados (MOHAN, P. V. A., 2007) (PATRONIK; PIESTRAK, S. J., 2014) (PATRONIK; PIESTRAK, S. J., 2018) (PETTENGHI *et al.*, 2018). Cada bloco será abordado nas seguintes seções.

Figura 1 – Unidade completa baseada em RNS.



Fonte – Adaptado de (MOHAN, P. A.; MOHAN, P. A., 2016)

2.1 PROPRIEDADES DO SISTEMA RNS

Em um sistema de numeração binária utilizado normalmente em aplicações computacionais, há informações que são transportadas dos dígitos de menor importância para aqueles de maior significância. Como resultado, existe uma desaceleração da aritmética devido ao sistema de computação destes transportes de informação, aqui chamados de sinais de *carry* (por exemplo, somadores *ripple carry*, somadores Carry Look Ahead Addition (CLA) (KOYADA *et al.*, 2017)). O cálculo destes sinais pode ser acelerado, mas apenas às custas de hardware adicional. Já, ao utilizar o sistema residual, garante-se a realização de operações sem a propagação de sinais de *carry*.

No sistema de numeração residual, um número x é representado por uma coleção de resíduos os quais estão relacionados a um conjunto de módulos relativamente primos $m_{i-1},..,m_1,m_0$. Desta forma, dizemos que o resíduo y_i do número x estão associado ao módulo m_i. A notação mais comumente utilizada é descrita pela representação (1)

$$R_i = x \ mod \ m_i = |x|_{mi} \tag{1}$$

Para representação de todos os resíduos pode-se incluir todos os resíduos, que

estão associados aos seus módulos pela forma representada em (2).

$$M = m_{i-1} \; x \; .. \; x \; m_1 \; x \; m_0 \tag{2}$$

O produto M do conjunto de módulos equivale ao número de diferentes valores em que se é possível representar dado este determinado conjunto. Tal produto também pode ser chamado de faixa dinâmica como exibido em (3). Ou seja, para o conjunto [16, 15, 17] existe 16 *x* 15 *x* 17 = 4080 números distintos para servir como representações possíveis.

$$x = (2|10|8)_{RNS(16|15|17)} \tag{3}$$

Perceba que existe uma relação de congruência no sistema. Esta significa que, dado dois inteiros *a* e *b*, estes são considerados módulo m congruentes se m divide exatamente a diferença de a e b. Esta relação é definida matematicamente pela equação (4), onde *m* é definido como a base ou módulo da operação.

$$a \equiv b \; (mod \; m) \tag{4}$$

Desta forma, como exemplo temos que, 8 $\equiv$ 5 (*mod* 3), 16 $\equiv$ 10 (*mod* 3) e 10 $\equiv$ –2 (*mod* 3). Assim definimos o resíduo R_i como o resto da divisão inteira do número X pelo módulo m_i. Por exemplo, considerando um sistema RNS com conjunto de módulos 3,7,11, teríamos a conversão dos valores 95 e 10 como $|2|_3$,$|4|_7$,$|7|_{11}$ e $|1|_3$,$|3|_7$,$|10|_{11}$ respectivamente. Caso realizemos as somas destes resíduos gerados teríamos $|2 + 1|_3 = |0|_3$,$|4 + 3|_7 = |0|_7$,$|7 + 10|_7 = |6|_{11}$. Para confirmação desta soma podemos realizar a adição dos número 95 + 10 = 105 e então transformar o resultado em representação RNS. Teríamos então o resultado igual a $|0|_3$,$|0|_7$,$|6|_{11}$, confirmando o cálculo anterior.

2.2 CONJUNTOS MODULARES

Para aplicações computacionais é de grande importância a utilização de conjuntos modulares que forneçam tanto representações eficientes quanto balanceamento. Conjuntos de módulos desequilibrados levam a arquiteturas desiguais, fazendo com que módulos maiores sejam excessivamente dominantes por possuírem maior custo.

A capacidade de simplificar as implementações das operações aritméticas também é um fator de extrema importância. Isto geralmente traduz-se no fato de projetos não realizarem grandes desvios da forma de potências de dois. Uma escolha bastante utilizada e a qual possui aritmética simplificada é $m_i = 2^n - 1$. Entretanto nem todos os pares de número no formato $2^n - 1$ são coprimos. Isto ocorre apenas se, dado o par $2^j - 1$ e $2^t - 1$, os dois elementos *j* e *t* forem coprimos (OMONDI; PREMKUMAR, 2007). Neste contexto, um dos conjuntos mais comuns e eficientes é o conjunto

$[2^n - 1, 2^n, 2^n + 1]$. De forma geral, a escolha do módulo depende principalmente das seguintes considerações:

a) **A faixa dinâmica** a qual consiste no número de valores representados dentro de um determinado conjunto. Este valor é dado pelo produto de todos os módulos $[m_1 \ x \ m_2 \ x \ ... \ x \ m_i]$. A aplicação será o elemento que determinará a necessidade da extensão para garantir que as representações sejam únicas.

b) Eficiência para aplicação dos algoritmos de **conversão reversa** de RNS para binário.

c) **Tamanho individual dos módulos** e balanceamento, dado que uma das principais vantagens do RNS ocorre devido à quebra das palavras binárias em vários pequenos resíduos que não exigem transporte de sinais de *carry* entre canais.

2.3 IMPLEMENTAÇÕES RNS

As implementações realizadas neste trabalho podem ser simplificadas em três casos. A primeira pode ser entendida como uma única multiplicação entre uma constante e uma variável, neste caso, correspondendo à conversão direta binária para RNS. A segunda seria a multiplicação modular entre duas variáveis. Já a terceira consiste no somatório de várias constantes multiplicadas por várias variáveis, sendo a operação equivalente à conversão reversa. Todas estas aplicações podem ser implementadas por meio do fluxo apresentado de forma genérica na Figura 2.

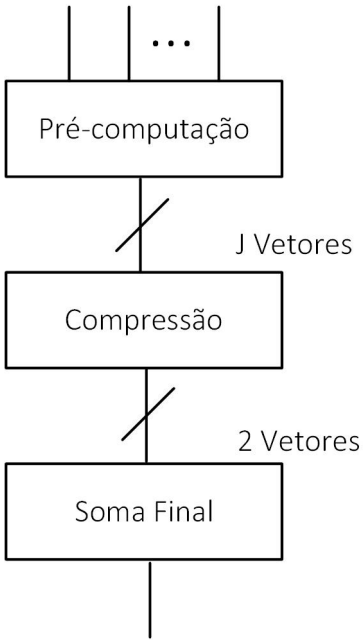
Dado que o processo de pré-computação é específico para cada aplicação mencionada, primeiramente será especificado os blocos compressores sob um contexto mais genérico. O objetivo de tal elemento é o de comprimir informações binárias, realizando a soma dos vetores, que compõe a matriz de informação, provenientes do processo de pré-computação. Por fim, serão apresentadas as diversas aplicações para a arquitetura genérica exibida.

2.3.1 Compressão com uso de *Ripple Carry*

Os circuitos básicos para constituição da etapa de compressão são os meio somadores e somadores completos, como demonstrado na Figura 3.

Uma das formas de resolver o problema da compressão é por meio da soma das informações binárias a partir de somadores *Ripple-Carry* como demonstrado na Figura 4. De forma intuitiva e simplificada, a propagação do sinal de *carry* em cada nível fica 1 unidade de tempo atrás do nível anterior.

Figura 2 – Arquitetura Genérica para Operações Modulares



Fonte – Elaborado pelo autor, 2021

Já o circuito chamado de Carry Propagate Addition (CPA) constitui-se como uma outra forma de se referir à tal estrutura. Ou seja, quando somadores *Ripple-Carry* ligam-se por meio de sinais de *carry in* e *carry out* formando uma sequência, como demonstrando na Figura 5a. Este é um importante circuito constitutivo e amplamente utilizado em arquiteturas para operações modulares.

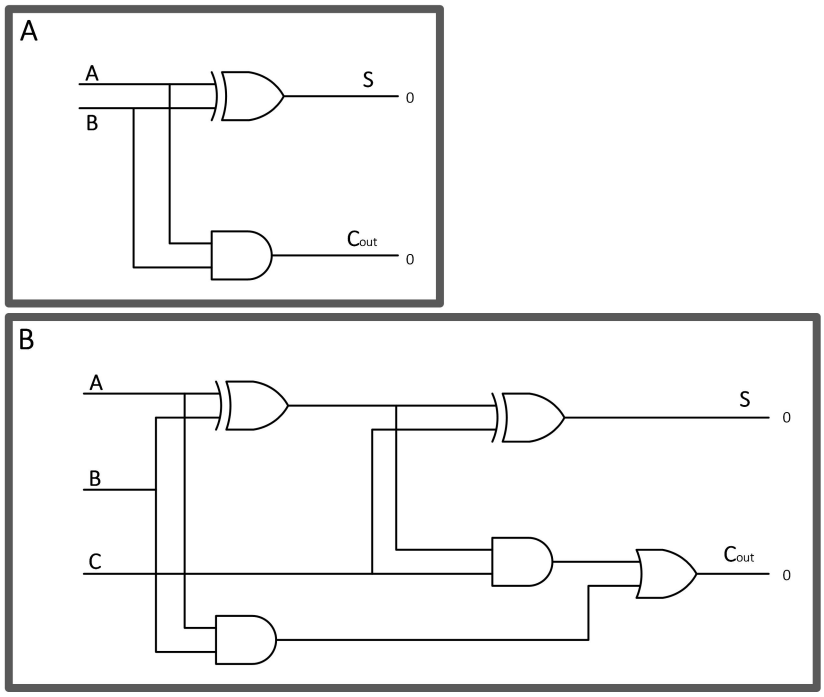
2.3.2 Compressão com uso de CSA

Um outro tipo de solução poderia ser provida a partir de uma rede de Carry Save Addition (CSA)s. Este pode ser definido caso os sinais de *carry out* dos somadores completos não sejam propagados, e ao invés disto, sejam armazenados, como indicado pela Figura 5b. Ao aplicar uma configuração utilizando CSAs obtém-se uma versão completamente paralela de um somador multioperando.

A partir da definição do uso de CSAs, um outro conceito que pode ser introduzido é o conceito de compressor. Um somador completo (Full Adder (FA)), na configuração de CSA, pode ser considerado como um elemento de redução de três números para dois. Em outras palavras, o FA pode ser considerado como um compressor de 3 para 2 ou [3:2].

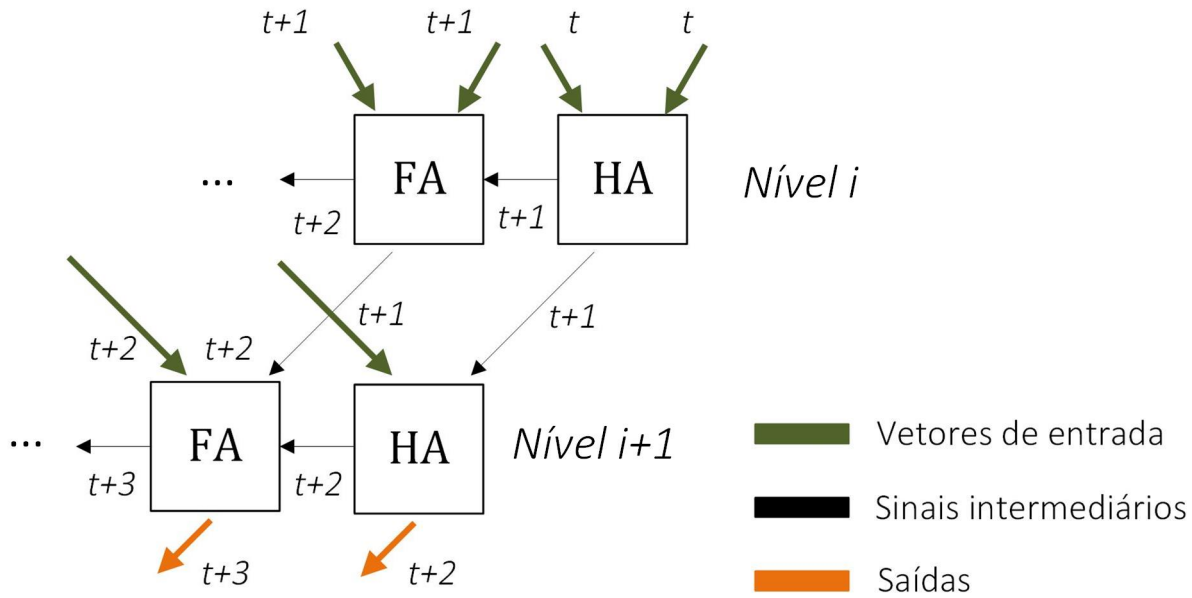
Neste mesmo sentido, o meio somador ou (Half Adder (HA)) realiza a transformação de dois bits verticais para dois bits horizontais, e portanto pode ser descrito como um compressor [2:2].

Figura 3 – Composição dos circuitos (a) Meio Somadores (HA) e de (b) Somadores Completos (FA).



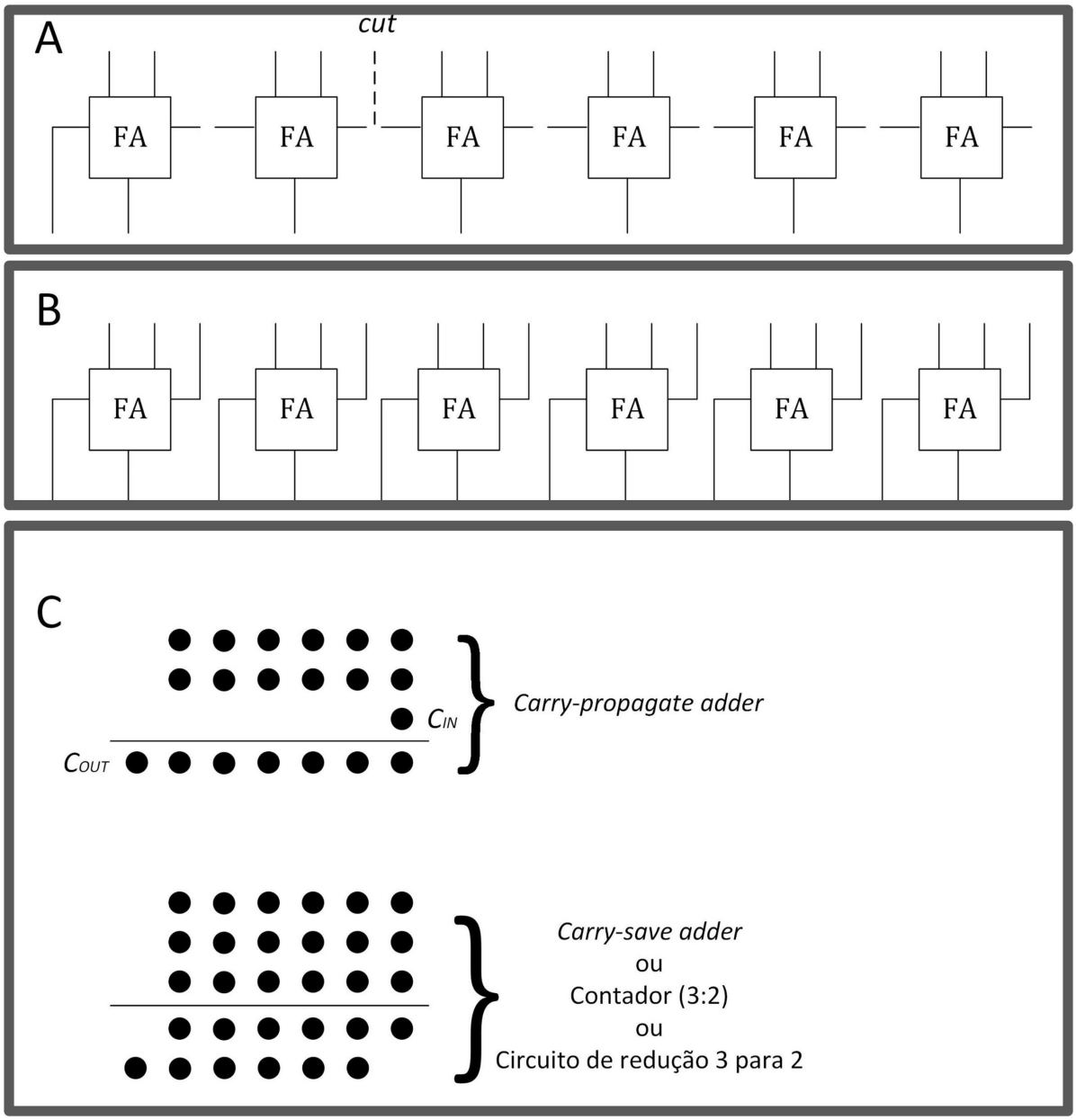
Fonte – Elaborado pelo autor, 2021.

Figura 4 – Árvore de Somadores utilizando somadores *Ripple-Carry*.



Fonte – Adaptado de (PARHAMI, 2000)

Figura 5 – Comparação entre somadores *Ripple-Carry* e CSA

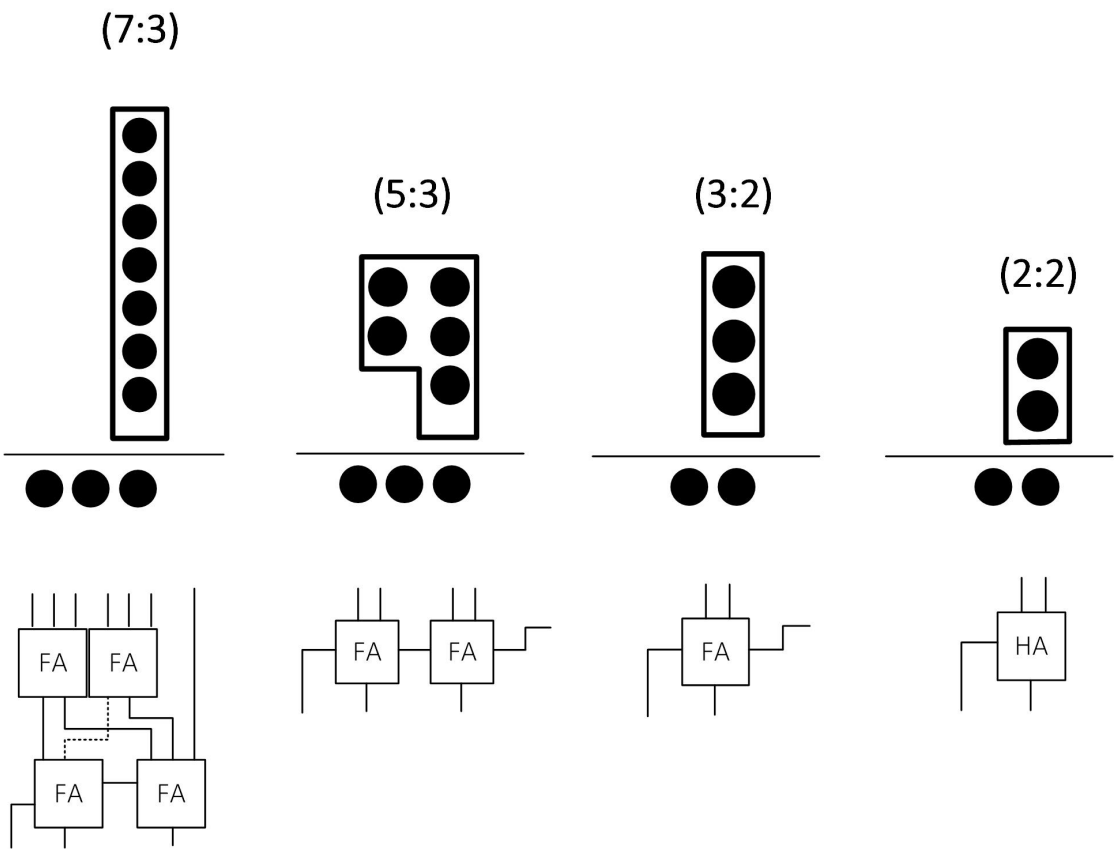


Fonte – Adaptado de (PARHAMI, 2000)

A utilização de blocos compressores nos permite montagem rápidas em projeto, facilitando que o projetista conceda mais foco em otimizações globais para remoção de redundâncias e melhora de interações entre os componentes, enquanto que o foco de melhora de desempenho de consumo e velocidade pode ser separada em análises dos blocos compressores isolados como demonstrado em (VILLEGER; OKLOBDZIJA, 1993) (FRITZ; FAM, 2017) (SAYED; AL-ASAAD, 2004) (HASAN *et al.*, 2020). As estru-

turas de alguns compressores são ilustradas na Figura 6.

Figura 6 – Exemplos de compressores



Fonte – Elaborado pelo autor, 2021.

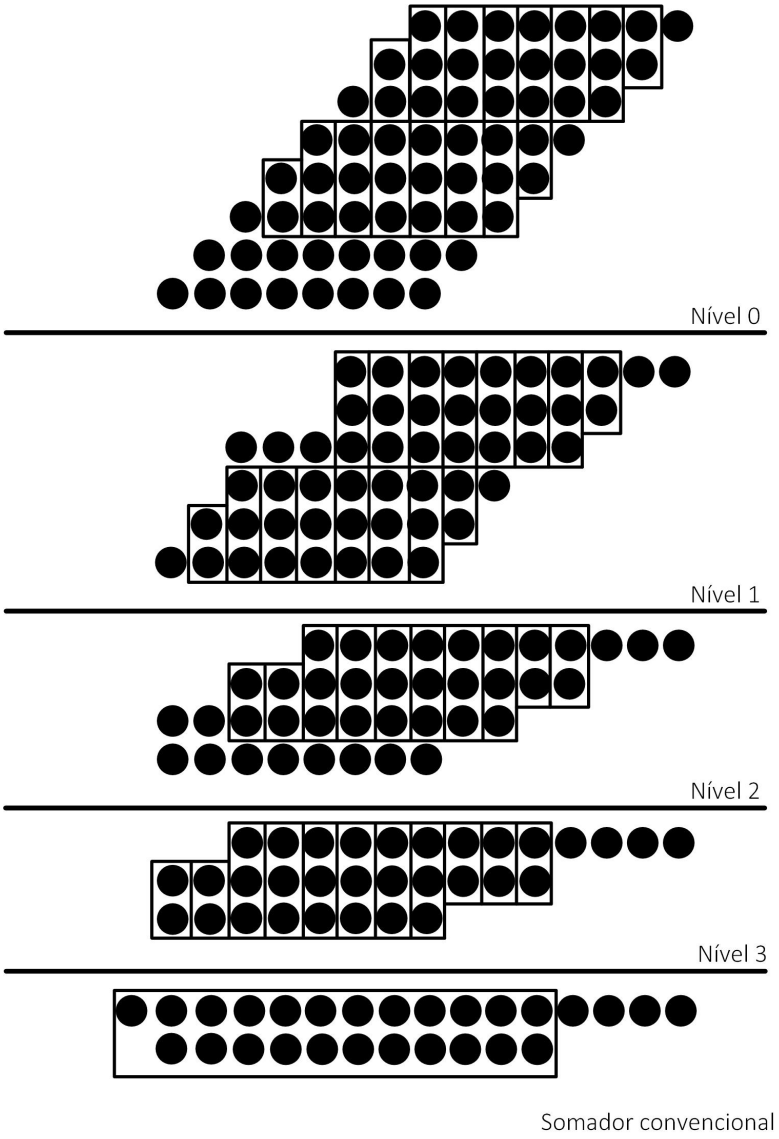
Neste contexto, várias propostas que fazem uso de tais compressores foram desenvolvidas nas últimas décadas. Uma aplicação de grande notabilidade é a de multiplicadores. A primeira estrutura com arranjo iterativo de compressores foi proposta por Wallace (WALLACE, 1964). Esta proposta faz uso de FAs e HAs, como demonstrado na Figura 7.

Na proposta de Wallace são alocados o máximo número de compressores a cada estágio. Cada linha é agrupada em conjuntos de três elementos não sobrepostos. Dentro de cada conjunto de três linhas, os Somadores Completos reduzem as colunas com três bits e os meio somadores reduzem as colunas com dois bits. As linhas que não fazem parte do conjunto de três linhas são transferidas para o próximo estágio para redução.

Por sua vez, Dadda (DADDA, 1965) realizou a generalização do esquema proposto por Wallace buscando minimizar o número de compressores utilizados na árvore de compressão.

Na verdade, encontrar a implementação ideal de uma árvore de compressores

Figura 7 – Representação da árvore de Wallace



Fonte – Elaborado pelo autor, 2021

ainda hoje permanece como uma tarefa desafiadora. Apesar das árvores de Wallace e Dadda serem propostas muito difundidas, a emergência de melhorias destas configurações é notável.

Com o tempo vários trabalhos introduziram o uso de grandes compressores. Tais compressores traziam implementação baseada na interconexões entre vários blocos de compressores (3 : 2). Estes blocos maiores poderiam, até certo ponto, prover atrasos menores, uma vez que suas interconexões internas são mais adequadas e consideram os tempos de chegada, nos elementos do circuito, para interconectar seus blocos (3 : 2) constituintes.

Alguns trabalhos demonstram o uso de tais compressores como por exemplo

(4:2), (5,5 : 4), (9 : 2), (7 : 3), entre outros (WEINBERGER, 1981), (SONG; DE MICHELI, 1991) (MA; LI, 2008) (GHASEMZADEH *et al.*, 2015) (RASHID; MUHTAROĞLU, 2017).

2.3.3 Alocação ótima de compressores e algoritmos de otimização para circuitos binários

Para discorrer sobre as otimização de caminho crítico em circuitos de compressão, devemos entender um pouco sobre o processo de síntese lógica. Este processo obteve grandes progressos na última década sendo parte essencial do design digital e substituindo, quase que universalmente, as técnicas manuais. Nele existem fases de otimização em nível de RTL e em nível de portas lógicas. As otimizações em nível de portas são executadas para satisfazer o tempo de ciclo caso este não tenha sido alcançado através da síntese RTL. Reestruturação de redes e redimensionamento de tamanho de portas são as técnicas utilizadas em tal otimização.

Entretanto, uma restrição notável ao processo de otimização encontra-se no nível de RTL, onde em problemas de aritmética computacional e projetos de caminhos de dados (*"datapaths"*), os projetistas ainda dependem principalmente de estruturações manuais baseadas em arquiteturas clássicas ou já bem estudadas, em que a síntese lógica possui um papel mais secundário no processo de otimização.

Compressores de coluna baseados em CSAs são normalmente construídos explorando a regularidade do circuito e, devido ao grande número de operações XOR, dificilmente são otimizados por sintetizadores lógicos. Na verdade, devido às deficiências da fatoração algébrica, as operações XOR geralmente não são otimizadas por estes sintetizadores (VERMA; IENNE, 2007b).

Nesse sentido, são apresentadas algumas abordagens para tratamento da alocação de compressores em circuitos aritméticos utilizando Full Adders e Half Adders.

- As abordagens desenvolvidas em (STELLING *et al.*, 1998), também chamadas de "Greedy Approach" são algoritmos que buscam encontrar as interconexões ótimas entre os vários compressores (3 : 2), considerando os tempos de chegada de suas entradas.

  Em The Greedy Approach (TGA), cada posição de bit é considerada individualmente da direita para a esquerda e os bits em uma coluna são reduzidos a três elementos ou menos usando compressores (3: 2). Ao escolher as entradas de um compressor (3 : 2), soma-se aqueles bits cujo tempo de chegada de entrada é o menor entre todos os bits. Uma vez que cada posição de bit tem três ou menos bits, uma sequência final de compressores (3: 2) é usada para reduzir os três últimos elementos em dois para que finalmente sejam adicionados usando um somador apropriado.

- Um outro trabalho que discorre de forma similar ao método TGA, para alocação ótima de FAs é (KIM; JAO; TJIANG, 1998). Algumas extensões a este trabalho foram propostas, como por exemplo em (YU, Z.; YU, M.-L.; WILLSON, 2001), para inclusão de operações entre registradores.

O desenvolvimento de trabalhos continuou a partir do fato de que nas abordagens de alocação citadas consideram apenas o atraso das entradas para realização das interconexões enquanto que o tamanho das interconexões em si, não são componentes primários a serem otimizados na síntese. Todavia, devido à alta complexidade de integração de um circuito ou efeitos de fio imprevisíveis, estas interconexões podem levar a resultados de layout insatisfatórios com conexões longas e desordenadas.

- Em (UM; KIM, 2002) é realizada a proposição de uma topologia que prioriza não só a otimização de temporização, como também atende a questão de construção de interconexões regulares entre CSAs, dando foco no aspecto de layout.

Deste modo, a partir do surgimento destes métodos de alocação, baseados em compressores (3:2) e (2:2), o leitor pode questionar o uso de compressores maiores, uma vez que vasta parte das propostas destes grandes compressores são baseadas no uso de compressores (3:2) como bloco básico. Entretanto vários autores demonstram ainda vantagens na utilização de grandes compressores em multiplicadores binários, provando sua competitividade (VERMA; IENNE, 2007a) (ESPOSITO *et al.*, 2018) (FATHI; MASHOUFI; AZIZIAN, 2020).

Alguns outros trabalho não focam exatamente na otimização da árvore compressora, mas sim na porção de soma final, a qual realiza soma dos dois últimos operandos na saída da árvore. Neste caso, a escolha de um somador ideal depende dos perfis de atraso dos bits das duas palavras de saída. Um exemplo dessa otimização é apresentado em (FADAVI-ARDEKANI, 1993). Nele, o somador gerado usa vários estágios de somadores *carry-select*. No entanto, o método proposto assume que todas as entradas da árvore do compressor estão disponíveis ao mesmo tempo, o que nem sempre é verdade. Já (STELLING P.F., 1996) faz a utilização de um somador híbrido para conduzir tal soma.

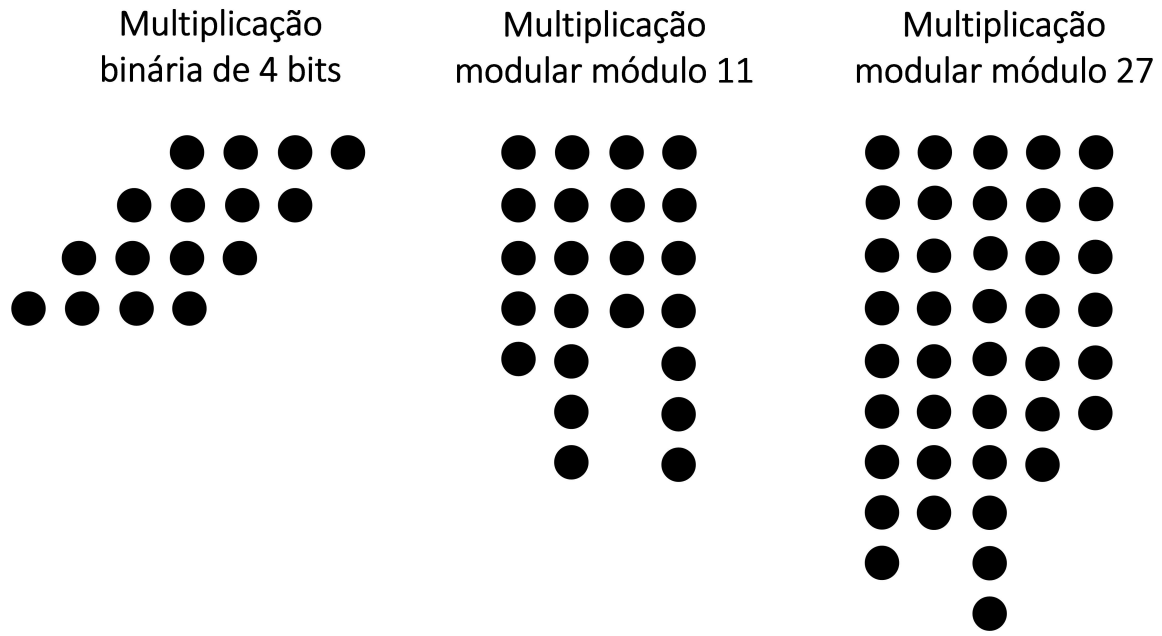
É importante citar ainda, como indicado em (VERMA; IENNE, 2007b), que muitas propostas assumem que as entradas da árvore de compressão são independentes umas das outras, o que em geral, não é o caso. No trabalho do autor indicado é apresentado formas de simplificação lógicas baseadas nas relações entre as equações lógicas em um multiplicador binário.

2.3.4 Compressor Modular

Nesse sentido, o problema abordado por este trabalho foge um pouco dos esquemas de multiplicadores binários normais. A primeira razão é que os formatos iniciais

das matrizes, que serão comprimidas, podem encontrar-se bastante desbalanceadas. Significando que pode existir colunas com um grande número de elementos enquanto outras possuem um número menor de elementos, como demonstrado pela Figura 8.

Figura 8 – Exemplos de formatação de produtos parciais.



Fonte – Elaborado pelo autor, 2021.

Uma segunda consideração que deve ser observada é que em uma árvore binária convencional, à medida em que são realizadas as somas pelos compressores, novos bits são inseridos nas posições com pesos maiores do que o MSB de peso $2^n - 1$, devido aos bits de *carry-out*, fazendo com que a árvore cresça lateralmente. Já em um compressor modular, o número de colunas permanece fixa em "n"enquanto que os bits de *carry-out* são reintroduzidos às colunas já existentes da matriz.

Dessa maneira, para entendimento de como é realizado a compressão modular proposta por este trabalho, deve-se entender primeiramente como realizar a representação e recodificação de números inteiros.

Dado um sistema de radix-2 e um conjunto de dígitos $[\overline{1}, 0, 1]$ onde $\overline{1}$ significa -1, podemos, por exemplo, representar o número 23 utilizando seis bits pelo indicado abaixo:

- 010111 (ou seja, $16 + 4 + 2 + 1$)
- $01100\overline{1}$ (ou seja, $16 + 8 - 1$)
- $10\overline{1}00\overline{1}$ (ou seja, $32 - 8 - 1$)

Esta formulação é muito utilizada em operações de multiplicação. Como exemplo, uma multiplicação direta pelo número positivo 1111111111 requer dez adições,

mas apenas duas são necessárias se o multiplicador for recodificado em $100000000\overline{1}$

A representação a qual possui o menor número de dígitos diferentes de zero e não contém a presença de "1s"adjacentes, é conhecida como Canonic Signed-Digit (CSD), ou representação canônica.

Entretanto, qualquer número inteiro tem um número infinito de representações. Isso pode ser facilmente observador pelo número "1", o qual pode ser representado como $[001]_2$ ou $[01\overline{1}]_2$ (ou seja, 2-1). Novamente, pode-se utilizar a mesma ideia para formar $[01\overline{11}]_2$ (ou 4-2-1) e assim por diante. Qualquer número inteiro positivo com um 1 à esquerda, pode ser substituído por qualquer número dessas sequências. Da mesma forma para qualquer número inteiro negativo, que tem um 1 inicial que pode ser substituído por $[0\overline{1}1]$. Desta forma, podemos produzir todas as representações com o mesmo número de 1s que a representação canônica. Tal representação é intitulada de Minimal Signed-Digit (MSD).

Para exemplo de representações MSD do número 181:

- $[10\overline{1}0\overline{1}0101] = 256 - 64 - 16 + 4 + 1 = 181$
- $[010110101] = 128 + 32 + 16 + 4 + 1 = 181$
- $[0110\overline{1}0101] = 128 + 64 - 16 + 4 + 1 = 181$
- $[01100\overline{11}01] = 128 + 64 - 8 - 4 + 1 = 181$
- $[10\overline{1}00\overline{11}01] = 256 - 64 - 8 - 4 + 1 = 181$
- $[01100\overline{1}0\overline{11}] = 128 + 64 - 8 - 2 - 1 = 181$
- $[10\overline{1}00\overline{1}0\overline{11}] = 256 - 64 - 8 - 2 - 1 = 181$

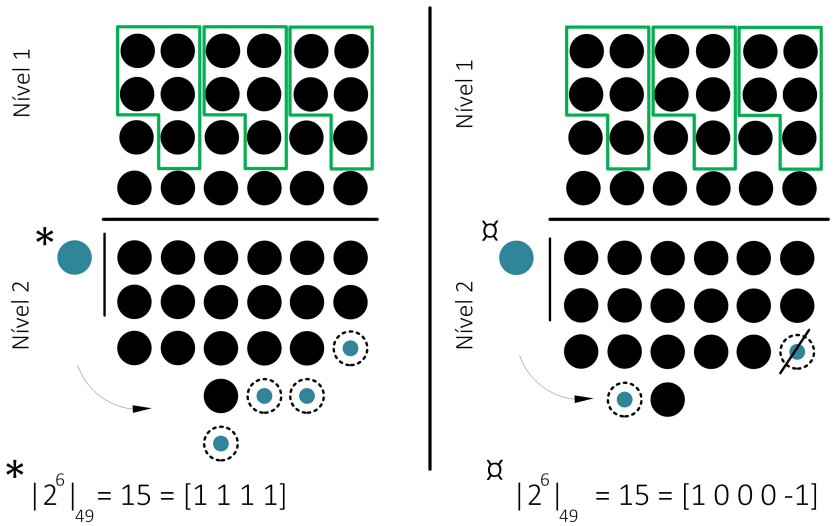
A distinção entre representação canônica e MSD será aplicada nas diferentes versões das ferramentas. A ferramenta versão 0 utilizará a representação canônica na reinserção dos bits durante o processo de compressão modular, enquanto que a ferramenta versão 1 utilizará representações MSD.

Assim sendo, para resolver a problemática de compressão modular, a ferramenta versão 0 realiza uma colocação de compressores similar à estratégia de Wallace, ou seja, o máximo número de compressores é alocado por iteração. Pelo fato do compressor [5:3] possuir um atraso similar ao compressor [3:2] estes são escolhidos como blocos fundamentais do projeto, assim como em (PATRONIK; PIESTRAK, S. J., 2017) e (PALUDO, 2020). É adicionado então o compressor [2:2] completando o conjunto de compressores utilizados. Tais compressores recebem uma ordem de prioridade com base em seus fatores de compressão, o que significa que o compressor [5:3] virá primeiro, seguido por FAs e finalmente HAs. Após a escolha do tipo de compressor na lista de prioridades, estes serão alocados, partindo da coluna do bit menos significativo e incrementando uma posição até a coluna do bit mais significativo. A redução da matriz de informação ocorrerá até que não haja nenhum encaixes disponível para aquele determinado compressor. A seguir, o próximo compressor da

lista de prioridades será escolhido para satisfazer qualquer possível encaixe restante. O processo se repetirá até que todos os compressores sejam cobertos. Três soluções de compressão podem ser geradas individualmente. Uma usando apenas FAs, uma segunda usando compressores 5: 3 e FAs e uma terceira usando todos os compressores indicados. A solução que fornecer o melhor atraso será escolhida como a solução final. Os bits que reingressam à matriz são representados apenas pela representação normal ou recodificação em sua forma canônica, da seguinte forma:

1. É comparado o número de 1s presentes na representação positiva ou negativa do determinado peso modular que deve reingressar. Aquele que tiver menor número de 1s é escolhido, dando prioridade para representação positiva.
2. É comparado o número de bits "1s" do peso a ser reintroduzido, tanto na sua representação normal quanto na sua recodificação. Caso o número de 1s da recodificação seja igual à representação normal, a representação normal é escolhida para reingressar à matriz. Ou seja, para o número 3 com representações [011] ou [10$\bar{1}$], teríamos a escolha de [011]. Caso o número de 1s da representação normal seja maior que a recodificação, então escolhe-se a versão recodificada.

Figura 9 – Exemplo de recodificação.



Fonte – Elaborado pelo autor, 2021.

Na figura 9 pode-se observar um exemplo da vantagem proporcionada pela recodificação de um determinado bit que é reintroduzido à matriz durante o processo de compressão módulo 49. Note que como $|2^6|_{49} = 15$ o determinado bit pode ser reinserido nas posições 2^0,2^1,2^2,2^3 sendo representado no formato [0,0,1,1,1,1], ou

pode ser reduzido por meio recodificação e retornar à posição 2^0 como um bit negado e 2^4 sendo então formatado como [0,1,0,0,0,$\overline{1}$]. Portanto, em todos os níveis de compressão estaríamos reduzindo o número de elementos reinseridos de 4 para 2, tornando o processo de compressão menos custoso.

Assim, com os compressores definidos, apresentam-se então as aplicações utilizando a versão 0 da ferramenta como solução. 9

2.3.5 Conversores Diretos

Os conversores diretos são responsáveis pela conversão de um número binário, com um determinado número de bits, para uma representação em RNS. Este seria o elemento inicial de qualquer bloco RNS. Um método simples para tal conversão seria usar um divisor para obter o resíduo, ignorando o quociente obtido. No entanto, a divisão é um processo de complexidade de implementação alta. Esta é a razão pela qual a conversão reversa é realizada pelo esquema genérico apresentado anteriormente.

Nesse sentido, considere um determinado número inteiro X o qual deseja-se calcular seu resíduo em relação a um módulo M. Tal número X pode ser representado de forma binária como x_{n-1},x_{n-2},...x_0, assim temos que seu valor de resíduo equivale a:

$$|X|_m = |x_{n-1},x_{n-2},...x_0|_m \tag{5}$$

Equivalendo também à:

$$|X|_m = ||\ 2^{n-1}x_{n-1}|_m + |2^{n-2}x_{n-2}|_m + ...|2^0x_0|_m\ |_m \tag{6}$$

Dado que x_i pode assumir apenas os valores de 0 ou 1, torna-se apenas necessário avaliar os valores $|2^i|_m$, os quais serão somados e aplicados à redução modular.

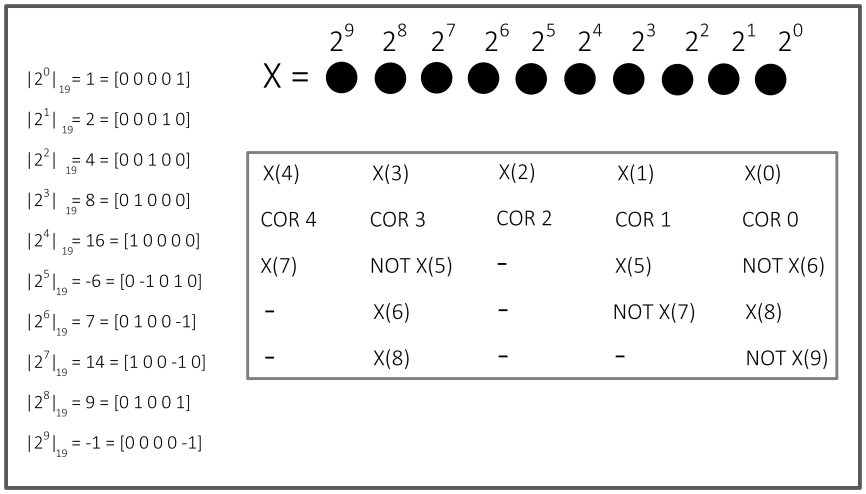
Observa-se na figura 10, um exemplo de construção da pré-computação para um conversor direto módulo 19. A variável a ser transformado em formato modular possui 10 bits. Os pesos maiores que 2^y sendo $y = log_2(19) \approx 4$ são redistribuídos nas posições menores que y. Tanto representações positivas quanto negativas podem ser utilizadas. Um fator corretor pode ser adicionado devido aos elementos negativos. Por fim, a ferramenta versão 0 utilizará a forma canônica de representação, ou seja, não possuindo 1s adjacentes.

2.3.6 Aritmética modular

2.3.6.1 Somadores Modulares

A soma modular consiste em uma operação essencial aplicada em sistemas RNS, sendo também um bloco fundamental para operações de multiplicação modular e conversões reversas de resíduos para representações binárias. Tais aplicações utilizam

Figura 10 – Exemplo de pré-computação para conversor direto módulo 19.



Fonte – Elaborado pelo autor, 2021.

somadores de dois operandos para realizar o estágio de soma final, dos operandos provenientes da etapa de compressão. Tal bloco também é conhecido como conversor final (FC).

Para este conversor final utilizado em multiplicações modulares é necessário realizar a adição de dois operandos $|X|_M = |A + B|_M$. Tendo A e B o tamanho de n bits, temos que o valor máximo para soma de tais operandos A e B é de $A + B = (2^n–1)+(2^n–1)$. Dependendo do módulo essa operação pode ultrapassar 3 vezes o valor deste próprio módulo, mais especificamente nos casos de módulos 2^n+k. Portanto para realizar a soma modular destes operandos, é necessária a transformação apresentada em (7):

$$|X|_M = |A+B|_M = \begin{cases} A+B-3m, (caso\ A+B \geqslant 3m) \\ A+B-2m, (caso\ 2m \leqslant A+B < 3m) \\ A+B-m, (caso\ m \leqslant A+B < 2m) \\ A+B, (demais\ \ casos) \end{cases} \tag{7}$$

Uma arquitetura amplamente utilizada para calcular tais somas modulares é apresentada na Figura 11 a. A arquitetura genérica, contemplando o caso em que $A + B \leqslant m$, é formada por dois somadores que ligam-se em paralelo. O primeiro realiza a soma $A + B$ enquanto que o segundo realiza a soma $A + B – m$. A medida em que é necessário realizar as subtrações por 2m ou 3m, somadores pode ser conectador em paralelo e ligados a um multiplexador para seleção correta dos sinais, como demonstrado em 11 b e 11 c. Perceba que para arquitetura que faz uso da soma $A + B – 3m$ é necessário o uso de uma lógica de redução indicada pelas relações (8) e

(9), para redução do número de entradas para ligação do mux 4 para 1.

$$control(0) <= D_{Cout}\ or\ (not(C_{Cout}\ and\ B_{Cout}));\tag{8}$$

$$control(1) <= C_{Cout}\ or\ D_{Cout};\tag{9}$$

O somador indicado pode ser implementado tanto por CPAs quanto por somadores mais rápidos como de prefixo paralelo. No caso desta dissertação, utilizou-se somadores de prefixo paralelo do tipo Brent-Kung (BRENT; KUNG, 1982).

2.3.6.2 Multiplicadores Modulares

Caso 2^n

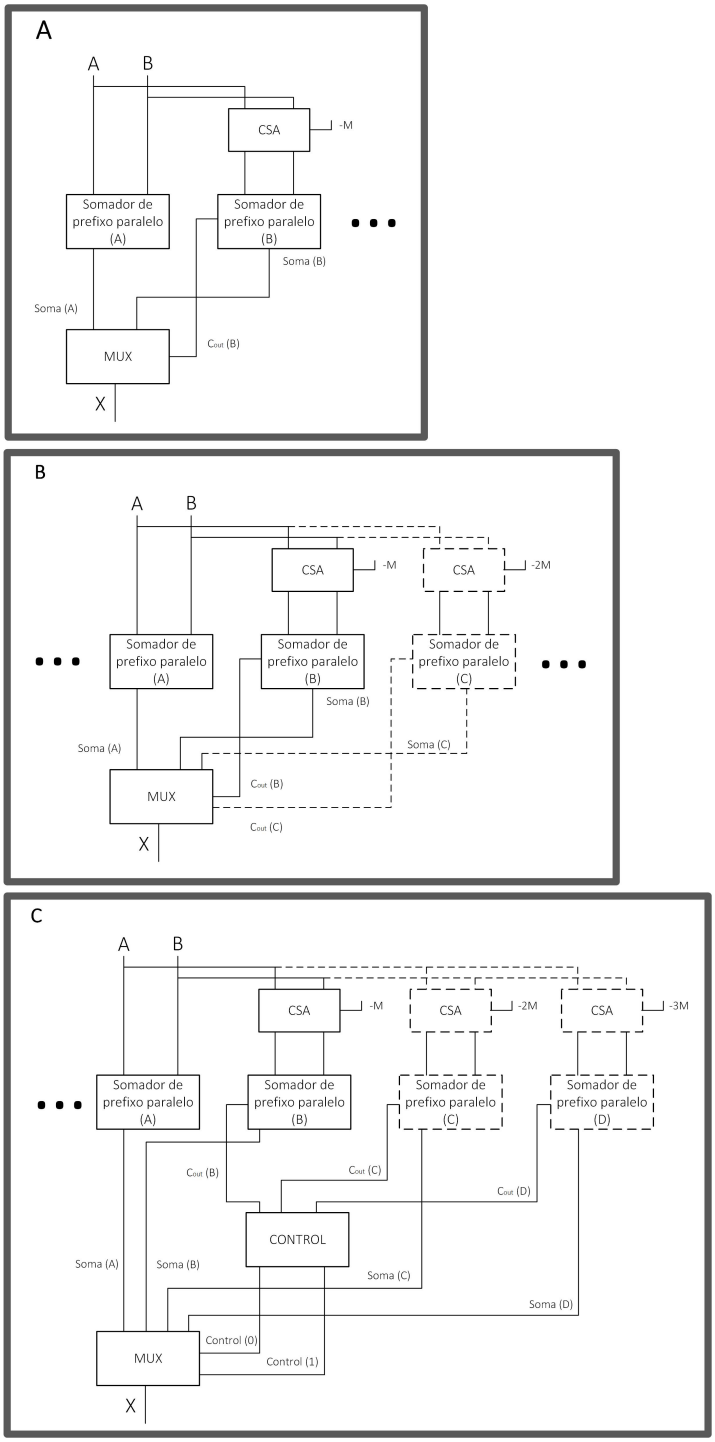
Para os casos 2^n, a multiplicação modular pode ser obtida truncando os produtos parciais com peso maior que 2^n, uma vez que $|2^n|_{2^n} = 0$, no processo de pré-computação. Os vetores resultantes são comprimidos por meio de uma estrutura de árvores com sinais de *carry-out* truncados. No último estágio são somados os dois vetores restantes por meio da utilização de um único somador final, podendo este ser um somador de prefixo paralelo ou um CPA. Uma vez que os bits maiores que 2^n são iguais a zero, o somador não utilizará seu sinal de *carry-out*. A estrutura para o cálculo completo pode ser observada na Figura 12.

Caso $2^n - 1$

Para os casos $2^n - 1$, a multiplicação modular pode ser obtida redistribuindo as posições dos produtos parciais com peso maior que 2, uma vez que, $|2^n + i|_{2^n-1} = 2^i$ para $i \geq n$, como demonstrado na Figura 13. Perceba que o peso de 2^i, aplicado à redistribuição na matriz, sempre irá possuir um único 1 em seu valor de representação binária, e desta forma já encontra-se condicionado à sua melhor representação possível, não sendo necessário nenhum tipo de recodificação. A redução da matriz de informação pode ser realizada por meio da utilização de End around Carry (EAC) em todos os níveis da árvore de CSA, dado que $|2^n|_{2^n-1} = 1$.

A representação máxima na saída do compressor será $A + B = 2x(2^n - 1)$, ou seja, para o módulo 31, como exemplificado na 13, as saídas da compressão poderiam assumir como maior valor o número 30, e por consequência, o maior valor possível na soma seria igual a 60. Como este valor é igual a duas vezes o valor do módulo é necessário condicionar a saída a um formato modular. Desta forma utiliza-se o somador modulo $2^n - 1$, o qual realiza de forma paralela o cálculo da soma (A+B) e (A+B -M), como apresentado no capítulo de somas modulares. É importante observar ainda que no exemplo dado na Figura 13, ambos os vetores estão completos e consequentemente não é possível realizar simplificações no hardware da soma final. Entretanto esta é uma solução específica. De modo geral, os multiplicadores

Figura 11 – Arquitetura para soma modular.



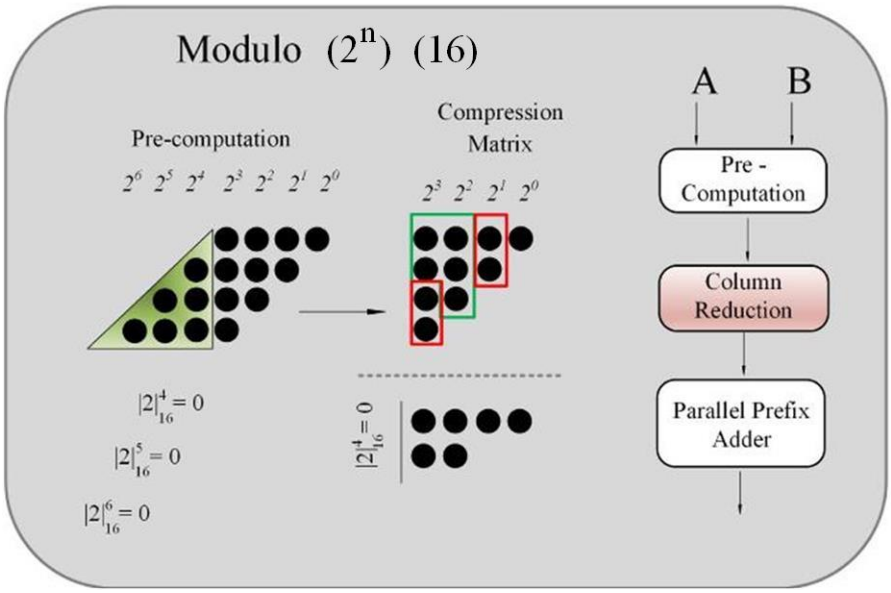
Fonte – Elaborado pelo autor, 2021.

modulares gerados neste trabalho terão somas simplificadas automaticamente quando detectadas as possibilidades.

Caso $2^n + 1$

Para os casos $2^n + 1$, a multiplicação modular pode ser obtida redistribuindo

Figura 12 – Formação de Produtos Parciais 2^n



Fonte – Elaborado pelo autor, 2021.

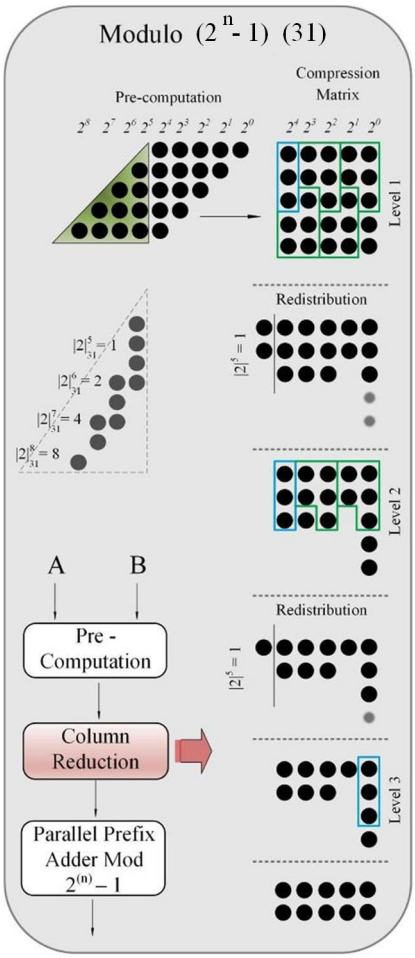
as posições dos produtos parciais com peso maior que 2^n dado que $|2^{n+i}|_{2^n+1} = -2^i$ para $i \geq n$. É importante notar que os pesos reintroduzidos na matriz de informação são negativos e portanto é necessário a colocação de um fator corretor para o correto funcionamento do cálculo.

A adição dos vetores realizada na árvore de compressão pode ser realizada por meio de *Inverted End-Around-Carry* (IEAC) uma vez que $|2^n|_{2^n+1} = -1$, ou seja, o peso possui um valor negativo. A saída da compressão é realizada por meio de um somador de prefixo paralelo módulo $2^n + 1$. Novamente, o somador final será simplificado caso haja possibilidade. A exemplificação do caso é dada pela Figura 14.

Caso $2^n \pm k$

Neste caso, a multiplicação modular pode ser obtida redistribuindo as posições com peso maior que 2^n uma vez que $|2^{n+i}|_{2^n-k} = \pm k \times 2^i$ para $i \leq n$, a fim de obter o vetores A_j. Dígitos com sinais são usados tanto para a formação dessas matrizes quanto como para os bits que são reintroduzidos na árvore do compressão durante a propagação do sinal de *carry*. A escolha da representação dependerá do número de 1s no determinado peso específico. Se as representações positivas e negativas tiverem o mesmo número de 1s, a representação positiva terá prioridade. Portanto, neste caso, a necessidade de inclusão do fator de corretor dependerá do valor de k. Os vetores A_j podem ser somados aplicando EAC ou IEAC, nos níveis de CSA, dependendo da escolha de representação. Finalmente, as saídas da compressão são adicionadas por um somador modular.

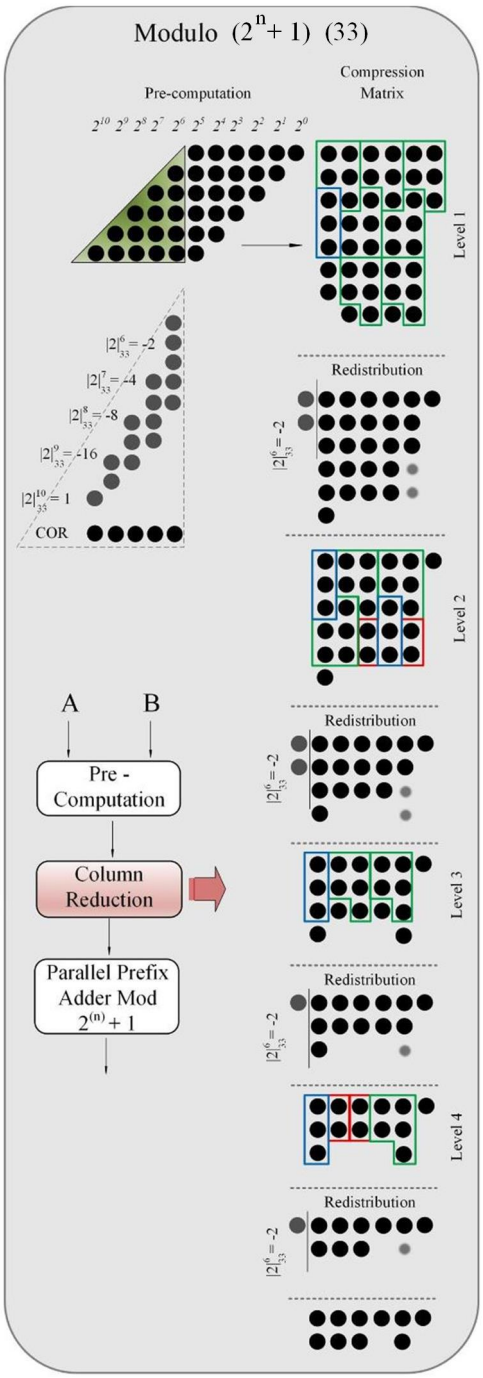
Figura 13 – Formatação de produtos parciais $2^n - 1$



Fonte – Elaborado pelo autor, 2021.

Na Figura 15b é apresentada a pré-computação dos produtos parciais e a arquitetura de hardware para $n = 5$ e $k = +11$. Observa-se que o peso 2^6 pode ser representado como $|2^6|_{43}$ e é igual a 21 ou -22. Como os dois números possuem três 1s em suas representações possíveis, o número positivo 21 é escolhido para retornar como um resíduo parcial. O restante dos pesos $(2^7, 2^8, 2^9, 2^{10})$ são escolhidos como -1, -2, -4, -8 respectivamente, visto que eles têm apenas um único bit 1 em sua representação. Após a pré-computação, é realizada a compressão e posteriormente a soma final, concluindo o cálculo. Salienta-se, que neste exemplo, o peso $2^6 = 21$ levará a um aumento de complexidade, uma vez que cada bit que sai da matriz será transformado em três bits quando reintroduzido em todos os níveis de compressão. Módulos menores e mais simples na forma de $2^n - k$ com k contendo um número reduzido de 1s é preferível, mas nem sempre possível.

Figura 14 – Formatação de produtos parciais $2^n + 1$

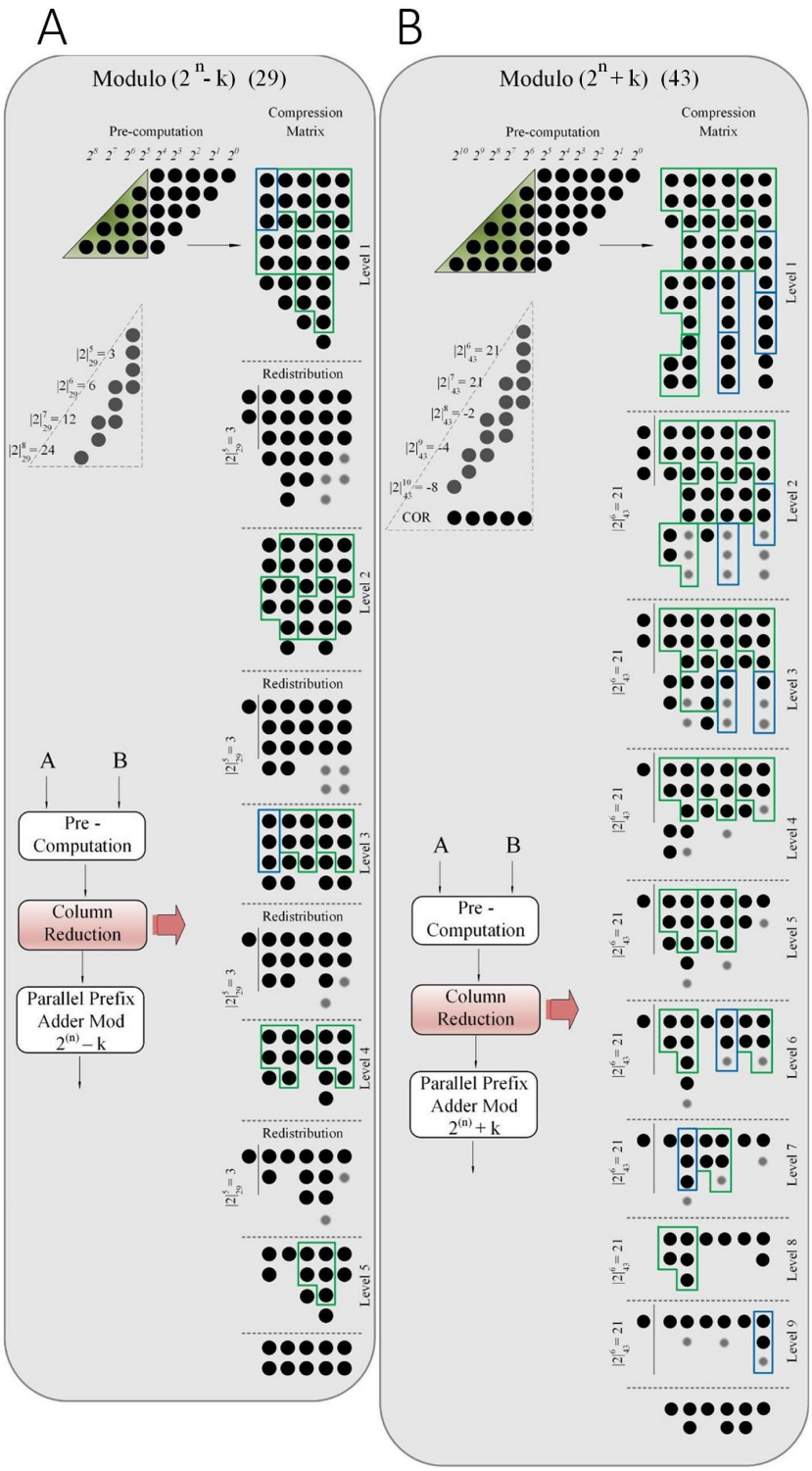


Fonte – Elaborado pelo autor, 2021.

2.3.7 Conversores Reversos

Os algoritmos mais utilizados para conversão reversa são o Chinese Remainder Theorem (CRT), Mixed Radix Conversion (MRC)(SZABO; TANAKA, 1967) e o New CRT-I (WANG, 2000). Tais algoritmos são computacionalmente exigentes e fortemente

Figura 15 – Formatação de produtos parciais (a) $2^n - k$ e (b) $2^n + k$.



Fonte – Elaborado pelo autor, 2021.

dependentes do conjunto de módulos. Para o MRC é necessário a formação de operações modulares em série enquanto o CRT demanda apenas uma única operação. Portanto, esta dissertação dará foco no CRT uma vez que ele proporciona soluções mais rápidas quando comparado ao MRC. Não somente isso mas o CRT pode ser expandido para o Novo CRT-I caso uma potência de dois for escolhida para o valor do módulo m_1.

De acordo com o CRT, um valor em RNS pode ser convertido novamente para binário por meio da equação (10):

$$X = \left| \sum_{i=1}^{N} \left| B_i R_i \right|_M \right|_M , \tag{10}$$

onde N equivale ao número de módulos no conjunto modular, R_i corresponde ao resíduo de entrada modulo m_i, $M = \prod_{i=1}^{N} m_i$, $B_i = \hat{m}_i \left| \hat{m}_i^{-1} \right|_{m_I}$, $\hat{m}_i = \frac{M}{m_i}$ e por fim $\left| \hat{m}_i^{-1} \right|_{m_i}$ o qual representa a multiplicativa inversa de $\hat{m}_i$ em relação ao módulo m_i.

O algoritmo de Euclides pode ser aplicado para obtenção da multiplicativa inversa $\left| \hat{m}_i^{-1} \right|_{m_i}$ para o conjunto modular, considerando a condição $\left| (\hat{m}_i) \times (\hat{m}_i^{-1}) \right|_{m_i} = 1$ with $i = 1,2,\ldots,N$.

Já para o algoritmo novo CRT-I, um valor representado em RNS pode ser convertido de volta para uma representação binária (X) por meio da equação (11)

$$X = \left| \sum_{i=1}^{N} \left| V_i R_i \right|_{\hat{m}_1} \right|_{\hat{m}_1} m_1 + R_1, \tag{11}$$

onde $V_1 = \left| \hat{m}_1^{-1} \right|_{m_1} \frac{(\hat{m}_1+1)}{m_1}$ e $V_i = \left| \hat{m}_i^{-1} \right|_{m_i} \frac{\hat{m}_i}{m_1}$ for $2 \leq i \leq N$. Os demais parâmetros são equivalentes ao que foi apresentado para o algoritmo CRT.

Como exemplo apresenta-se na Figura 16 a implementação para $\{m_1,m_2\} = \{2^4-1,2^4+1\}$ e $\{m_1,m_2,m_3\} = \{2^4,2^4-1,2^4+1\}$ ao utilizar o CRT e Novo CRT-I respectivamente. Os casos apresentados serão aplicados à compressões modulares do tipo 2^n-1, entretanto para os demais casos 2^n+1 e $2^n \pm k$ a compressão é realizada da mesma forma como descrita na seleção de multiplicadores.

Para computação das adições modulares $\sum_{i=1}^{n} \left| B_i R_i \right|_{2^8-1}$ e $\sum_{i=1}^{3} \left| V_i R_i \right|_{2^8-1}$ fazemos com que os resíduos R_i, associados ao respectivo módulo m_i, sejam expandidos para representação de n-bits, $r_{i(n-1)},\ldots,r_{i0}$. A Tabela 1 expões os parâmetros B_i para os conjuntos $2^4-1,2^4+1$ e $\{3,5,17\}$ e V_i para $\{2^4,2^4-1,2^4+1\}$, no qual cada valor pode ser expresso por termos de potência de dois.

Para redução da complexidade do *hardware* pode-se aplicar as seguintes propriedades:

i) as multiplicações modulares $\left| B_i R_i \right|_{2^8-1}$ e $\left| V_i R_i \right|_{2^8-1}$ podem ser implementadas por meio do uso das operações rotação à esquerda (*ROL*) e/ou rotação à direita (*ROR*).

Tabela 1 – Exemplo numérico para $\{m_1, m_2\} = \{2^4 - 1, 2^4 + 1\}$ e $\{m_1, m_2, m_3\} = \{2^4, 2^4 - 1, 2^4 + 1\}$ ao plaicar o CRT e Novo CRT-I respectivamente.

CRT					
i	m_i	$\hat{m}_i$	$\left\|\hat{m}_i^{-1}\right\|_{m_i}$	B_i	$\left\|B_i R_i\right\|_{2^8-1}$
1	15	17	8	136	$(2^7 + 2^3)R_1$
2	17	15	8	120	$(2^7 - 2^3)R_2$
Novo CRT-I					
i	m_i	$\frac{\hat{m}_i}{m_1}$	$\left\|\hat{m}_i^{-1}\right\|_{m_i}$	V_i	$\left\|V_i R_i\right\|_{2^8-1}$
1	16	–	15	–15	$(-2^4 + 1)R_1$
2	15	17	8	136	$(2^7 + 2^3)R_2$
3	17	15	9	120	$(2^7 - 2^3)R_3$

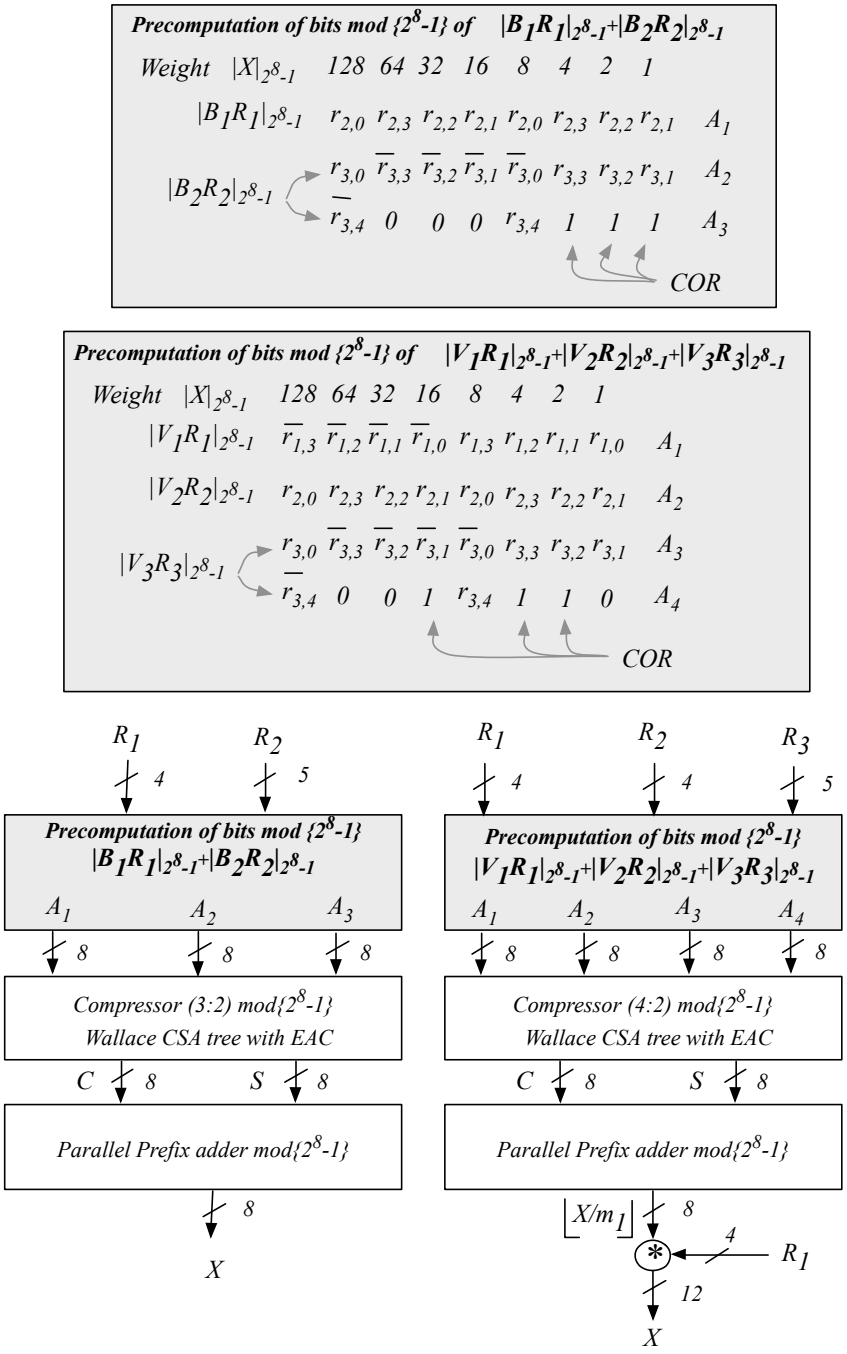
Fonte – Elaborado pelo autor, 2021.

ii) O módulo $2^8 - 1$ de um número negativo pode ser alcançado por meio da adição do seu complemento à um fator corretor *COR*, por meio da expressão: $\left|\bar{R}_i + COR\right|_{2^8-1} = 0$.

A propriedades aplicadas podem ser visualizadas na Figura 16 resultando nos vetores A_j. A adição de tais vetores é conduzida por meio de uma árvore de Wallace, fazendo uso de CSAs com EAC, dado que $\left|2^8\right|_{2^8-1}$. Já a soma final é alcançada por meio do uso de somadores modulares do tipo $2^n - 1$

A operação de concatenação apresentada na Figura 16b para o novo CRT-I é denotada pelo símbolo $*$.

Figura 16 – Exemplo de conversores reversos.



Fonte – Elaborado pelo autor, 2021.

3 COMPARAÇÕES ENTRE ARQUITETURAS DE COMPRESSORES DO ESTADO DA ARTE

As arquiteturas *Very Large Scale Integration (VLSI)* propostas na literatura para operações baseadas em RNS podem ser caracterizadas pelo uso de lógica combinacional direta, LUTs principalmente baseadas em memórias, ou uma combinação de ambas (PALIOURAS, V. e. a., 2001).

A utilização do método baseado em LUTs implementadas em *Read Only Memory (ROM)* é uma alternativa mais simples do ponto de vista de projeto. Entretanto, o uso de memória pode elevar o custo de área de forma exponencial para aplicações em que tem-se como resultado um número elevado de bits para os canais modulares, tornando-se ineficientes para módulos grandes. Neste sentido, existem propostas para redução do tamanho da memória utilizada, por exemplo, em operações de multiplicação de resíduos ao aplicar a restrição de escolha de apenas módulos primos para a operação modular (JULLIEN, 1980) (RADHAKRISHNAN; YUAN, 1992).

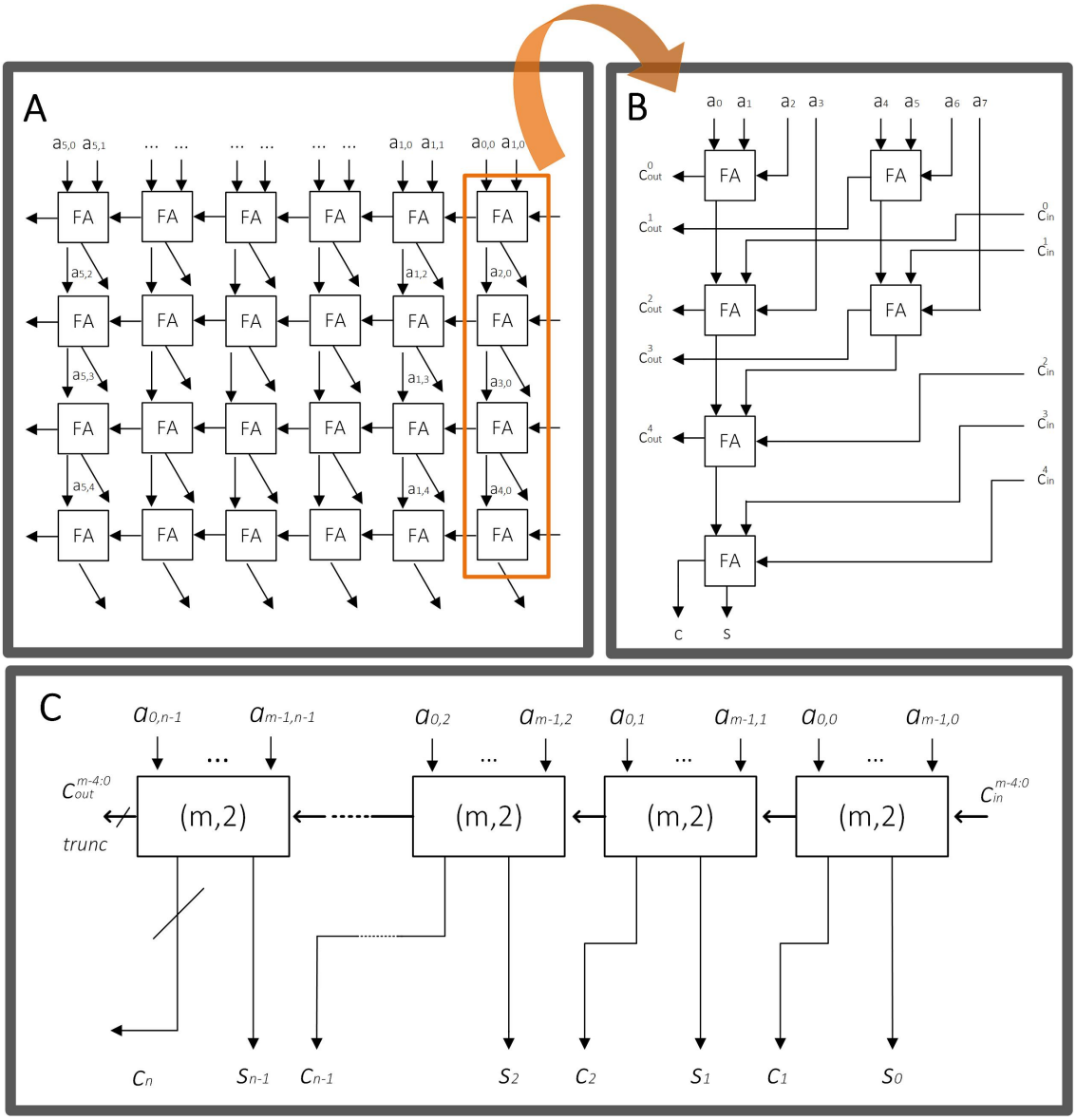
A exploração de arquiteturas combinacionais foi uma tendência criada para solucionar o problema de ineficiência de área citado. Outras arquiteturas baseadas no conceito de multifunção (PALIOURAS, V.; STOURAITIS, T., 1999) podem ser encontradas na literatura. Nestas o hardware é compartilhado entre as unidades de cálculo, podendo executar operações de diferentes módulos. No entanto, as restrições impostas para garantir o correto funcionamento para mais de um módulo, limitam a melhoria de desempenho. Projetos de multiplicadores modulares restritos a módulos específicos também foram abordados na literatura (HIASAT, Ahmad, 1992).

Dessa maneira, será realizada comparações de compressores para a arquitetura apresentada na versão 0 da ferramenta com as arquiteturas do estado da arte a fim de demonstrar as vantagens e desvantagens dos circuitos utilizados neste trabalho. Para conduzir tal comparação serão geradas soluções, para caracterização em 65 nm, utilizando o conjunto de compressores ([5:3],[3:2],[2,2]), ([5:3],[3:2]), ([3:2],[2,2]) e finalmente apenas [3:2]. Aquele que prover melhor resultado de atraso, será considerada com solução ótima e apresentada para comparação. Além dos compressores básicos ([3:2] e [2,2]), o compressor (5:3) foi utilizado por possuir atraso mais próximo destes elementos básicos. Por essa razão, outros compressores maiores, como por exemplo o compressor (7:3), não foram utilizados.

A estratégia proposta pela ferramenta 0 já é capaz de gerar resultados de compressão mais breves possíveis, em número de níveis, para os casos $2^n, 2^n{-}1$ *e* $2^n{+}1$ quando comparados à compressões do estado da arte. Já, para os casos $2^n \pm k$, foi identificado possíveis melhoras para redução do número de níveis de compressão, Sendo assim, a ferramenta otimizada versão 1 teve o objetivo de aprimorar os casos $2^n \pm k$. Portanto as comparações com estado da arte serão realizadas com a ferramenta 0 e posteriormente serão construídas as possíveis melhoras à ela.

Na Figura 17 pode-se observar um compressor bastante utilizado para compressão de vetores modulares. Entretanto, se tal circuito for dividido por colunas, como demonstrado em (ZIMMERMANN, R., 1999), pode-se realizar ligações mais eficientes entre os elementos desta determinada coluna, evitando realizar ligações consecutivas entre tais elementos, como indicado na Figura 17 b. Portanto, os compressores do estado da arte a serem comparados com a ferramenta farão compressões por coluna. Por sua vez, cada coluna estará ligada a próxima para assim efetuar a compressão de toda a matriz de informação, como indicado na Figura 17 c. O número de *full adders* em uma determinada coluna será proporcional ao número de elementos desta determinada coluna.

Figura 17 – Compressor estado da arte.



Fonte – Elaborado pelo autor, 2021.

3.1 COMPARAÇÃO CASO 2^n

Dado que o compressor, caso 2^n, utiliza o truncamento de bits que saem do compressor, não é necessário esquemas mais complexos para tratamento destes sinais de *carry out*. Assim, as estruturas utilizadas para comparação são similares, como demonstrado na Figura 18. A arquitetura "A" é construída a partir de biblioteca disponibilizada por (ZIMMERMANN, Reto, 1998) e faz uso de uma estrutura em árvore linear com *carry outs* de uma determinada coluna ligados à próxima. Apenas os *carry outs* da última linha são salvo e disponibilizados na saída do compressor. Tais estruturas são também descritas em (ZIMMERMANN, R., 1999). Já a proposição B consiste na proposta construída pela ferramenta 0. Neste caso, os compressores que forneceram melhor resultado para proposição da ferramenta 0 foram os conjuntos ([5:3],[3:2],[2:2]) e ([3:2],[2:2]).

Tabela 2 – Comparações 2^n

Modulo	*Atraso* Zimmermann (ps)	Área Zimmermann (μm^2)	*Atraso* Versão 0 (ps)	Área Versão 0 (μm^2)
2^8	964	**956**	**961**	1012
2^{10}	**982**	**2167**	1009	2279
2^{12}	1068	**2935**	**1059**	3231
2^{15}	**1179**	5332	1224	**5199**
2^{18}	**1246**	**7118**	1272	7329
2^{22}	**1332**	12464	1414	**11473**

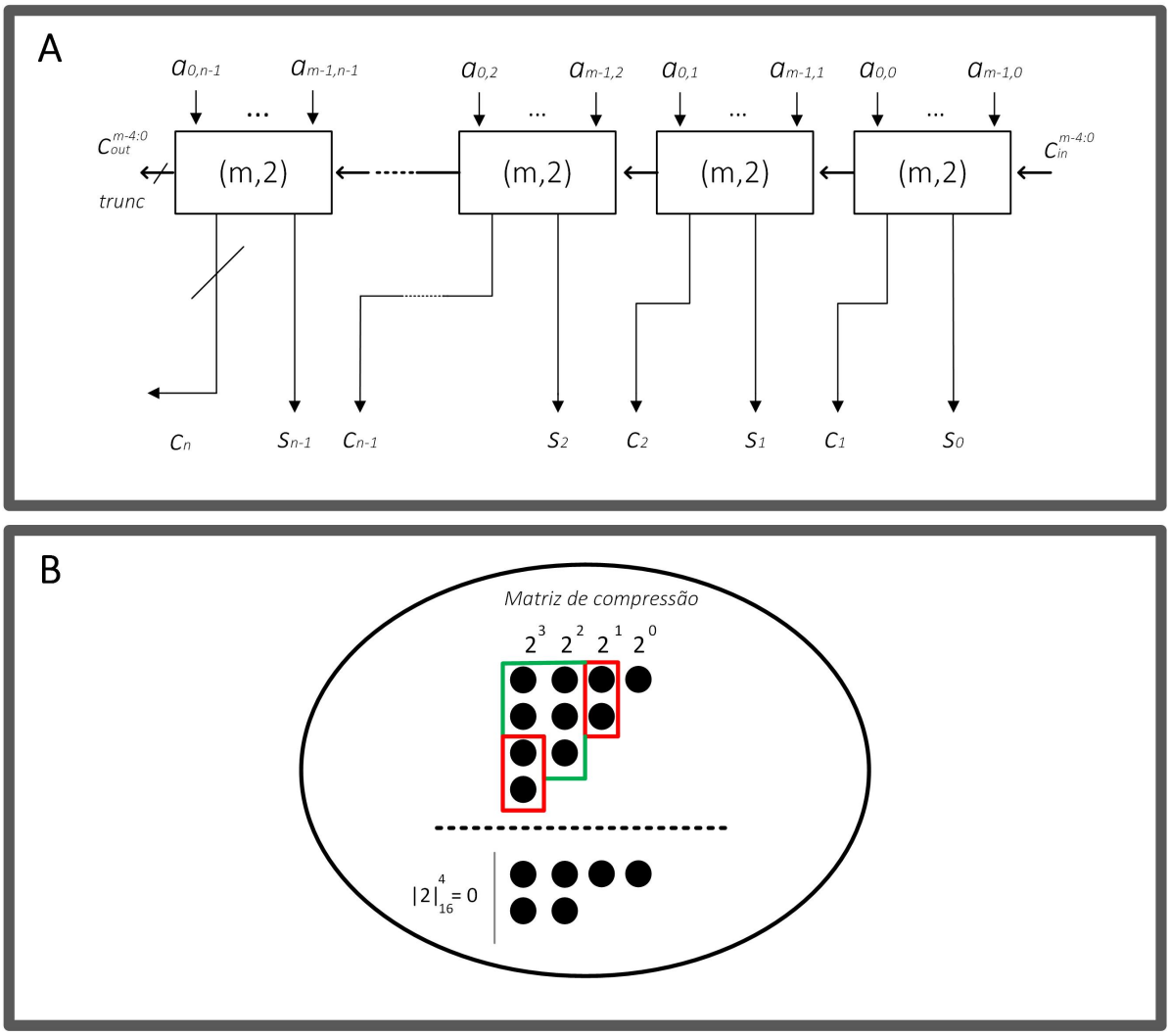
Fonte – Elaborado pelo autor, 2021.

Dado que as estruturas são similares no uso de Full Adders, obtém-se resultados também similares, entretanto, ocorrendo variação média de 3% como demonstrado na Tabela 2. A variação pode ocorrer pois na ferramenta a ligação dos compressores pode gerar um layout não tão regular quando comparado à versão de árvore linear. Na comparação, há casos de menor área e maior atraso, como em 2^{22} e 2^{15}, ou maior área e menor atraso como em 2^{12}, mas ainda sim resultados próximos.

3.2 CASO $2^n - 1$

As duas arquiteturas para comparação referente ao caso 2^n–1 são apresentadas pela Figura 19. Novamente a proposição A faz uso de uma árvore linear com *carry outs* dos elementos de uma coluna ligados à próxima. No entanto, visto que $|2^n|_{2^n-1} = 1$ ocorre o uso de EAC, fazendo com que os bits de *carry outs* retornem ao compressor. Este retorno é realizado como um *feedback* no Full Adder de mesmo nível em que o sinal foi gerado. Já a segunda proposição, produzida pela ferramenta 0, realiza o

Figura 18 – Arquiteturas 2^n

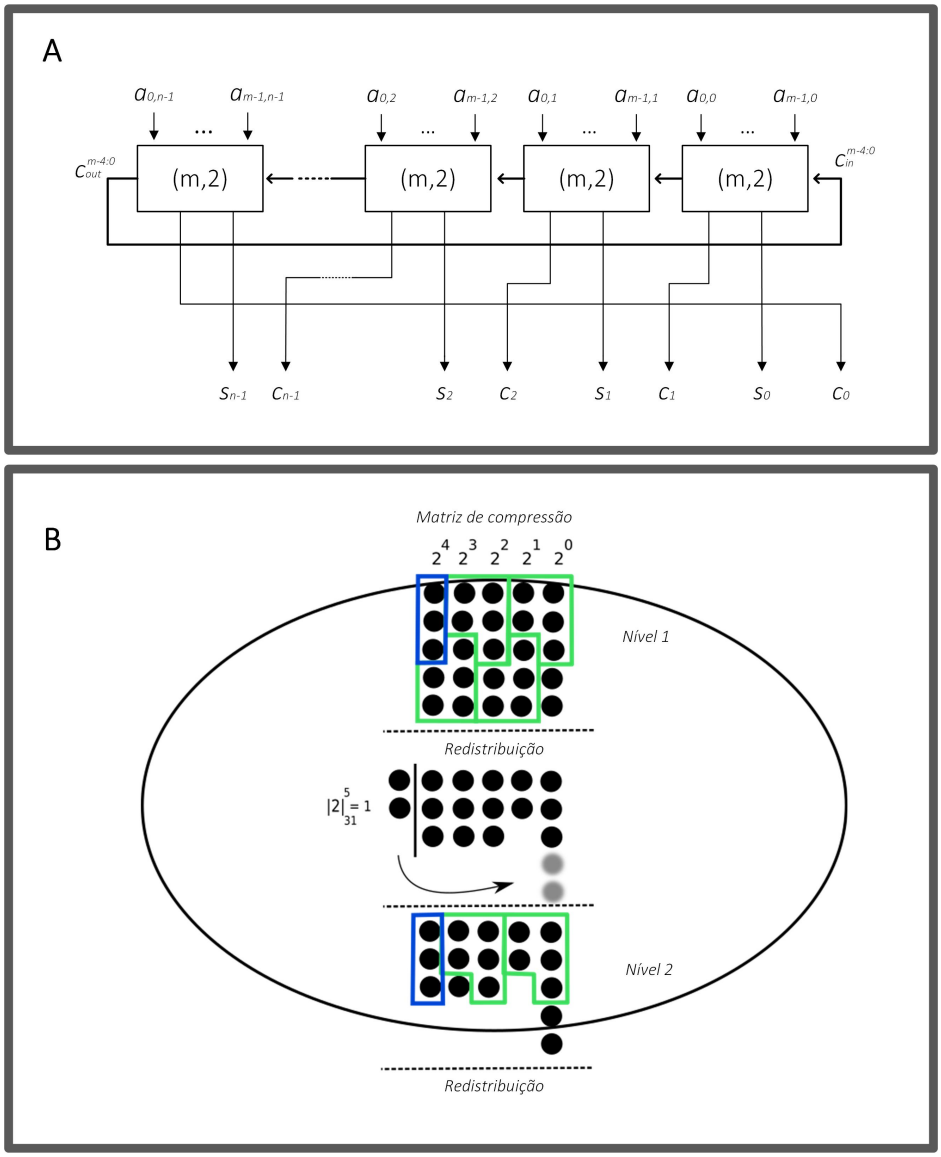


Fonte – Elaborado pelo autor, 2021.

processo de EAC, introduzindo os bits de *carry outs* para estarem disponíveis no nível seguinte de compressão. Um retorno do tipo *feedback* para o mesmo nível pode gerar preocupações com a estabilidade do sinal, no entanto em contrapartida, gera-se um circuito mais rápido uma vez que não aloca-se bits para os próximos níveis.

Para o circuito da ferramenta versão 0, foi possível chegar a soluções ótimas de compressão em casos $2^n - 1$ realizando apenas o uso de compressores [3:2]. Devido aos possíveis encaixe apenas com *Full Adders* foi possível atingir um atraso de tempo de computação com uma estrutura bastante regular. Na Tabela 3 é possível observar os resultados das comparações. Novamente os resultados são próximos, contudo observa-se que a proposta A atinge atrasos menores, enquanto a proposta da ferramente produz circuitos com menor área.

Figura 19 – Arquiteturas $2^n - 1$



Fonte – Elaborado pelo autor, 2021.

3.3 CASO $2^n + 1$

Os casos de arquiteturas para comparação referentes aos módulos $2^n + 1$ são ilustradas pela Figura 20. Neste caso também é utilizado a árvore linear com ligação *carry outs* internos da árvore de um determinado nível para o próximo. Já, neste caso dado que $|2^n|_{2^n+1} = -1$ ocorre o uso de IEAC, ou seja, bits de *carry outs* são reintroduzido de forma similar ao caso $2^n - 1$, com a diferença de que agora estes estarão invertidos. Para o caso da solução produzida pela ferramenta temos a utilização dos compressores ([5:3],[3:2] e [2:2]) para os melhores casos. Uma observação para tal solução é que módulos do tipo $2^n + 1$ acabam gerando o padrão descrito pela

Tabela 3 – Comparações $2^n - 1$

Modulo	*Atraso* Zimmerman (ps)	Área Zimmerman (μm^2)	*Atraso* Versão 0 (ps)	Área Versão 0 (μm^2)
$2^6 - 1$	956	630	**953**	**579**
$2^8 - 1$	**962**	**2079**	964	2273
$2^{10} - 1$	1086	**4288**	**1084**	4346
$2^{20} - 1$	**1329**	23126	1397	**20264**
$2^{26} - 1$	**1392**	39574	1484	**34707**

Fonte – Elaborado pelo autor, 2021.

Tabela 4 – Comparações $2^n + 1$

Modulo	*Atraso* Zimmerman (ps)	Área Zimmerman (μm^2)	*Atraso* Versão 0 (ps)	Área Versão 0 (μm^2)
2^{10}	**1103**	**5683**	1297	6454
2^{12}	**1294**	**6686**	1416	9382
2^{18}	**1397**	**16590**	1500	19541
2^{20}	**1426**	**21045**	1659	25936
2^{24}	**1482**	**32109**	1658	36119

Fonte – Elaborado pelo autor, 2021.

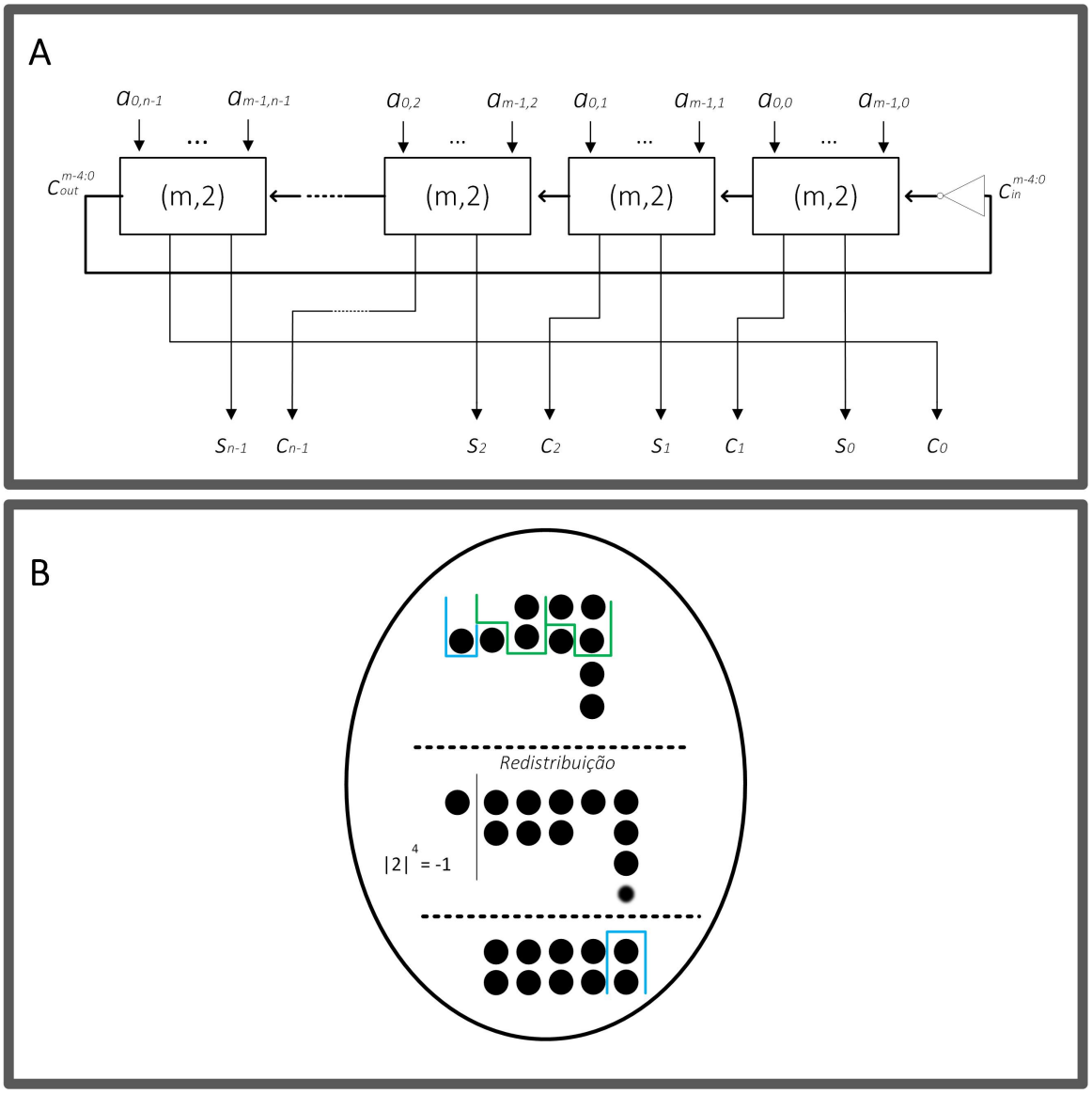
Figura 21. O padrão é ampliado dependendo do número de bits do módulo, forçando a colocação de mais compressores [2:2] de acordo com o número de bits. Uma vez que os compressores [2:2] não efetuam compressão propriamente dita, mas sim apenas um deslocamento de bits, as soluções irão apresentar uma elevação em área para poder atingir resultados de atraso com maior competitividade.

A Tabela 4 demonstra a comparação entre as arquiteturas mencionadas. Para o caso da ferramenta versão 0, devido ao fato mencionado do uso de [2:2] e ainda devido ao uso de IEAC reinseridos nos níveis posteriores a sua geração, é possível notar que a arquitetura Zimmerman é mais eficiente tanto em área quanto em atraso.

3.4 CASO $2^n \pm k$

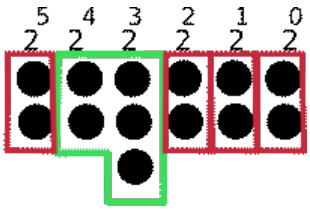
Para o caso $2^n \pm k$ existe a necessidade de utilização de uma arquitetura que faça uso de LUTs. Neste caso, a implementação é realizada pela utilização de ROMS. Sua utilização possui o intuito de processar os pesos com valor $|2^n|_{2^n \pm k} = \pm k$. Como mencionado nas seções introdutórias deste trabalho, apesar de eficientes, estas arquiteturas produzem resultados de área elevados à medida em que utiliza-se mais bits. Na Figura 22 é possível visualizar as arquiteturas as quais serão comparadas.

Figura 20 – Arquiteturas $2^n + 1$



Fonte – Elaborado pelo autor, 2021.

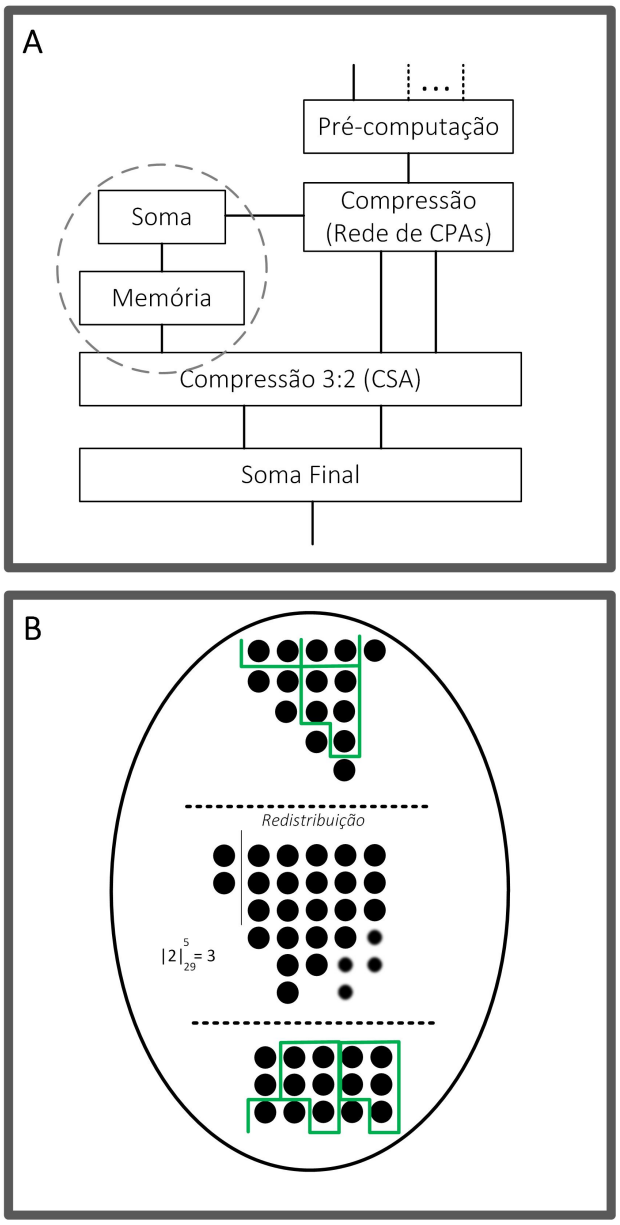
Figura 21 – Padrão no último nível de compressão para $2^n + 1$



Fonte – Elaborado pelo autor, 2021.

O elemento "A"da figura ilustra o circuito com memórias enquanto que o elemento "B"exibe a arquiteturas com reinserção de bits produzida pela ferramenta 0.

Figura 22 – Arquiteturas $2^n \pm k$



Fonte – Elaborado pelo autor, 2021.

Os casos $+K$ e $-K$, para um mesmo valor de K, assemelham-se possuindo tendências semelhantes. Portanto, para análise, apresenta-se um caso $2^n + 3$.

A estrutura produzida pela ferramenta 0 possui um atraso mais competitivo enquanto sua área é mais elevada. Em relação à arquitetura com memória apresentada, sua constituição é relativa a uma versão que prioriza área em troca de atraso. Devido a este fato, o circuito formado pela ferramenta 0 apenas ultrapassará sua eficiência em

Tabela 5 – Comparações $2^n + 3$

Modulo	*Atraso* Memoria (ps)	Área Memoria (μm^2)	*Atraso* Versão 0 (ps)	Área Versão 0 (μm^2)
2^8	1949	**5713**	**1525**	7473
2^{10}	2077	**8278**	**1683**	11424
2^{14}	2376	**14021**	**1796**	19943
2^{18}	2489	**23049**	**1863**	31802
2^{24}	2577	44050	**2225**	**40455**

Fonte – Elaborado pelo autor, 2021.

área por volta de 2^{24}.

4 OTIMIZAÇÕES

A proposta de otimização será aplicada com o intuito de melhora da compressão para os casos $2^n \pm k$ garantindo a resolução do problema em um número de níveis ótimo e compatível com circuito similares do estado da arte. Serão explorados aspectos como as codificações do bits a serem reinseridos, a colocação de compressores por colunas para gerar diferentes resultados formando também soluções múltiplas as quais serão comparadas para busca do melhor resultado de forma iterativa.

Dado que os módulos 2^n,$2^n + 1$ *e* $2^n - 1$ possuem formatos mais simples para as matrizes iniciais de compressão, uma vez que apresentam formato simétrico, e ainda, dado que a reinserção de bits à própria matriz já apresenta-se simplificado, pois retornam apenas no máximo um único bit em sua representação, teremos que a metodologia de otimização não afetará tais casos.

Por mais, será realizada otimização dos somadores modulares a partir das soluções encontradas na etapa de compressão.

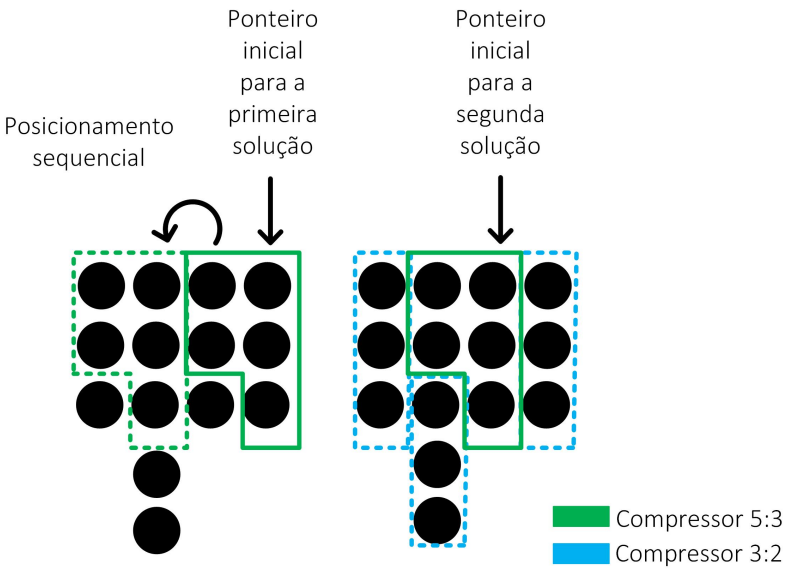
4.1 OTIMIZAÇÃO PARA ETAPA DE COMPRESSÃO

A matriz de produto parcial gerada pode assumir um grande número de formatos diferentes e dependerá essencialmente dos parâmetros de entrada da aplicação que o projetista deseja implementar.

No encaixe dos compressores, a primeira alocação influenciará nas possibilidades de inserção do próximo compressor. Ainda, diferentes alocações podem gerar uma solução que atinge um maior número de compressores por interação. Ao utilizar compressores [5:3] e [3:2], por exemplo, a estratégia do algoritmo visará inserir o máximo possível de compressores 5:3 seguindo por completar o restante da matriz com somadores completos nas posições remanescentes. Na Figura 23 é possível observar um exemplo de tal processo. Inicialmente, um ponteiro indica a posição da primeira inserção do compressor [5:3]. Sequencialmente, mais compressores são alocados nas colunas subsequentes, até que nenhuma outra inserção possa ocorrer. Por fim, compressores 3:2 preenchem as posições possíveis para tal, gerando uma solução para o nível atual. No entanto, para o exemplo dado na Figura 23, existem duas soluções possíveis que irão gerar resultados diferentes. Assim, para gerar uma outra solução diferente, é feito um incremento ao ponteiro inicial (de zero para um). Esse ponteiro indicará a posição para primeira alocação do compressor 5:3 inicial. Com isso o processo de colocação sequencial será repetido, formando um segundo resultado.

Quanto mais compressores forem usados na matriz de produtos parciais, maior será o fator de compressão. Como tentativa de avaliar se cada interação atinge o melhor cenário, são dados pesos referentes à área aos três possíveis compressores utilizados para resolver o problema. Na Figura 24 é realizada a exemplificação da

Figura 23 – Indicação de duas possibilidades de alocação de compressores



Fonte – Elaborado pelo autor, 2021.

relação entre os compressores [5:3] e [3:2] ao comprimir uma matriz com todos os vetores completos, demonstrando um fator de compressão análogo. O termo fator de compressão é definido como o produto do número de compressores usados em uma determinada interação pela relação de área correspondente dos compressores, como indicado na Tabela 6. Assim, duas soluções com o mesmo fator de compressão levarão à mesma resposta, ainda que seus respectivos compressores assumam posições diferentes. Em termos de atraso, será considerado que os compressores [5:3] e [3:2] possuem tempos iguais de propagação de seus sinais internos para realizar suas computações, assim como foi especificado no trabalho o qual gerou a ferramenta versão 0 (PALUDO, 2020).

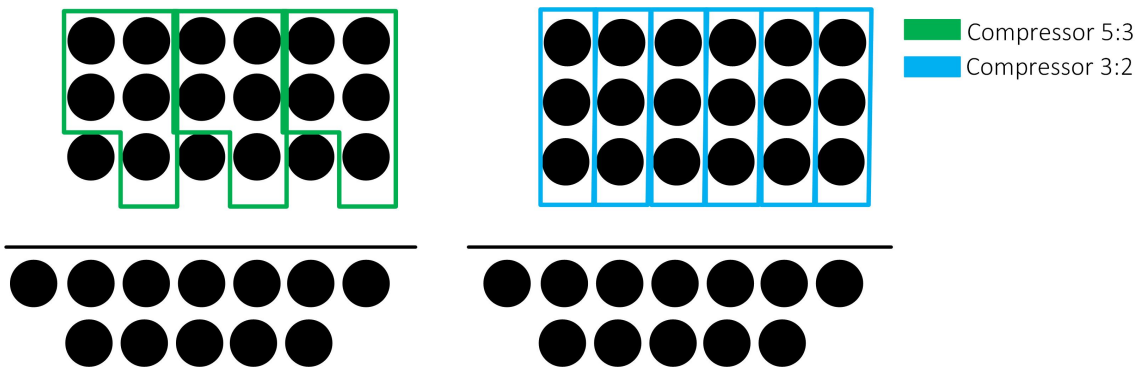
Tabela 6 – Relação de área entre os compressores

Compressor	Area
5:3	2
3:2	1
2:2	0.5

Fonte – Elaborado pelo autor, 2021.

Observa-se que, caso não houvesse diferença em usar o 5:3 ou 3:2 para atingir o mesmo fator de compressão, não haveria razão para diversificar os compressores. Seria lógico usar apenas aquele que apresenta o melhor desempenho, ao invés de misturá-los. Isso pode ser razoável para matrizes bem balanceadas, porém o formato

Figura 24 – Indicação da relação de compressão entre compressores [5:3] e [3:2]



Fonte – Elaborado pelo autor, 2021.

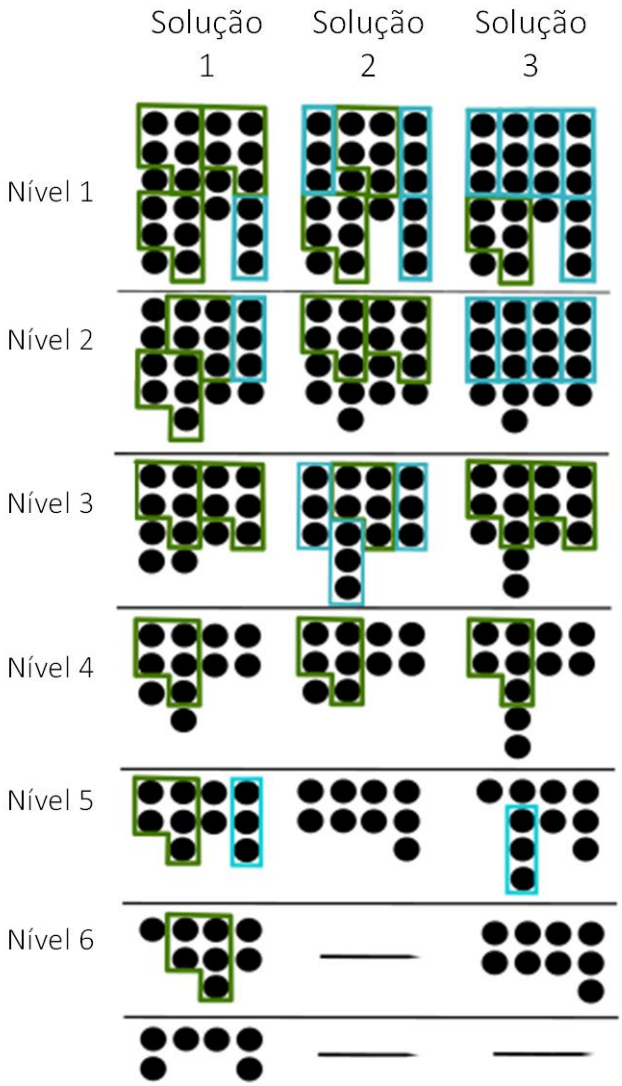
da matriz ao misturar os dois compressores possibilita alcançar um fator de compressão melhor. Na Figura 25 é exemplificado um problema de compressão módulo 11 onde procura-se a compressão para dois vetores finais, podendo haver mais um elemento no peso 2^0, para servir de *carry in* para próxima etapa, caso fosse utilizado um CPA por exemplo. A determinada matriz reintroduz os pesos 2^4 nos bits 2^0 e 2^2. As soluções apresentadas demonstram como a permutação das alocações dos compressores pode alterar o resultado final.

Na primeira iteração, os compressores são alocados em posições diferentes, resultando em um número de compressores 5:3 e 3:2 distintos em cada solução. Porém, o fator de compressão é igual em cada um dos três exemplos no nível 1, o que significa que o resultado da iteração será igual para todos os três. No nível 2, o fator de compressão da solução 1 é diferente das outras duas soluções. É importante notar que mesmo o fator de compressão da solução 1 seja maior neste nível, ele falha em alcançar a melhor solução final. Na verdade, dos três exemplos, ele gera o pior número total de compressores instalados, o que se reflete na área do circuito e pior número de iterações, ou seja, um atraso maior. Dito isso, a solução ótima parece não ser possível de ser alcançada ao tentar selecionar a melhor solução na iteração atual com base no melhor fator de compressão de um nível. Ela dependerá do formato da matriz em cada iteração e o algoritmo deve ser capaz de adaptar o posicionamento das peças à ela.

De forma prática, a implementação faz uso de conjuntos de “fragmentos” que irão indicar em qual posição deve ocorrer a alocação dos compressores. Considerando um exemplo de 4 bits, esses fragmentos assumem a forma de uma lista tal como [(0,1), (2,3)], para compressores 5:3, ou [(0), (1), (2), (3)] para somadores completos e meio-somadores.

Como o compressor 5:3 ocupa 2 colunas de espaço, então a lista [(0,1), (2,3)]

Figura 25 – Exemplo de alocações de compressores



Fonte – Elaborado pelo autor, 2021.

indica que um compressor deve ser alocado nas posições 0 e 1 e o outro nas posições 2 e 3. Pela mesma regra, uma lista [(0), (1), (2), (3)] indica que somadores completos ou meio somadores devem ser alocados em colunas de 0 a 3.

4.2 DEMONSTRAÇÃO DA APLICAÇÃO DA METODOLOGIA

Para exemplificação do funcionamento da metodologia de compressão serão apresentados casos que ajudem no entendimento dos elementos que possuíram influência para o alcance das soluções. No caso 1, será conduzido a comparação entre a metodologia proposta por este trabalho e um circuito o qual faz uso de LUT, tendo

sido apresentado em (LOW; CHANG, 2013). Tal circuito com LUT difere um pouco do que foi apresentado anteriormente. Ele pode ser observado na Figura 26. Nele é realizado um processo de compressão por meio de uso de compressores [3:2] e [2:2], entretanto não utiliza a técnica de reinserção de bits à própria matriz. Ao invés disto, utiliza uma LUT para tratar tais bits que saem do compressor.

Já para o caso 2 de demonstração dos aspectos da otimização desenvolvida, serão apresentados elementos como impacto da variação do posicionamento dos compressores e construção da matriz inicial de compressão como elementos para redução dos níveis.

4.2.1 Caso 1

Assim, como mencionado, é apresentado um exemplo de conversão direta retirado de (LOW; CHANG, 2013). Na Figura 27 demonstra-se a solução de aleatoriedade proposta neste trabalho, com reinserção dos bits maiores do que 2^n de volta a matriz. Já na Figura 26 representa-se a solução dada pelos autores do artigo mencionado, a qual baseia-se em um circuito com LUT.

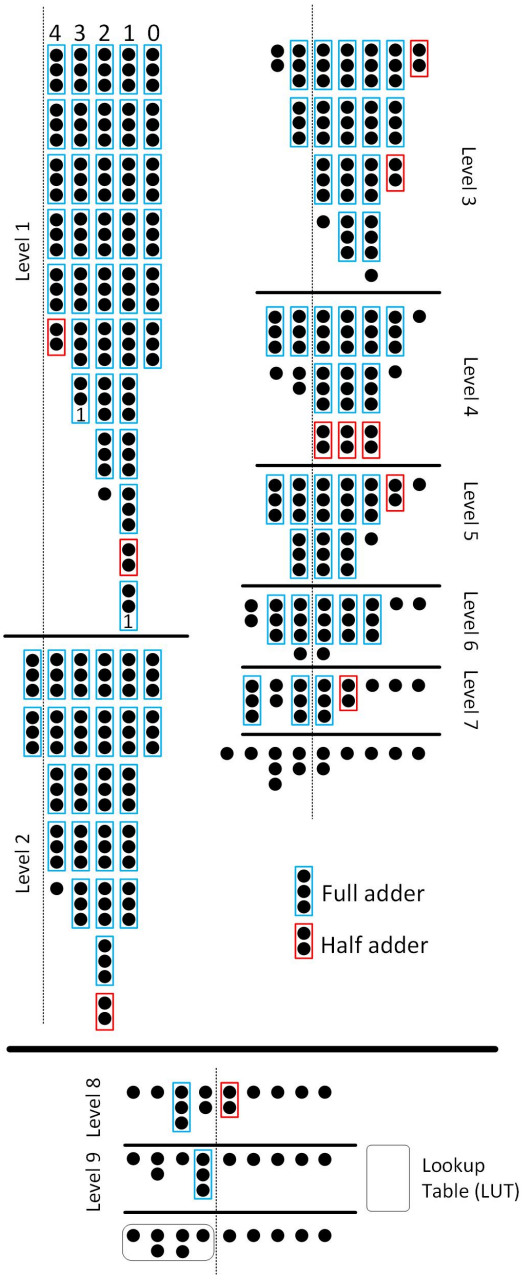
Ambas as soluções partem do mesmo formato. Neste caso em especial, pode-se observar que a matriz possui, em seu primeiro nível, um número de elementos bastante distintos em cada coluna, possuindo então um certo grau de desbalanceamento.

Apesar de ser possível realizar reorganização dos bits presentes na determinada matriz e que seu formato inicial apresenta importância para que seja possível alcançar o melhor número de níveis de compressão, neste caso, não será dado foco em como é realizada a construção dos elementos que a formam inicialmente. O objetivo de tal exemplo é garantir a compressão ótima a partir de uma matriz genérica qualquer.

Primeiramente, tomando como base a Figura 27 observa-se que é feito o uso de compressores [5:3], [3:2] e [2:2]. Neste caso, o algoritmo de colocação incia a compressão por meio da utilização de compressores [5:3], os quais possuem prioridade mais alta em relação aos outros. A colocação é realizada por meio da metodologia otimizada, ou seja, variando as posições das colunas de colocação dos compressores. Uma observação que deve ser feita é que não são colocados meio somadores na coluna de peso mais significativo. A razão parte do princípio de que o meio somador apenas realiza um deslocamento de bits, não efetuando redução de bits na matriz. Caso este fosse colocado na última coluna estaríamos deslocando um bit e forçando que este fosse reintroduzido na matriz novamente, podendo causar *loops* infinitos.

Os elementos marcados como "X" enfatizam apenas que não há bits naquela determinada posição. Estes foram adicionados apenas com propósito visual, devido ao reposicionamento dos pontos para melhor ilustração do encaixe dos compressores.

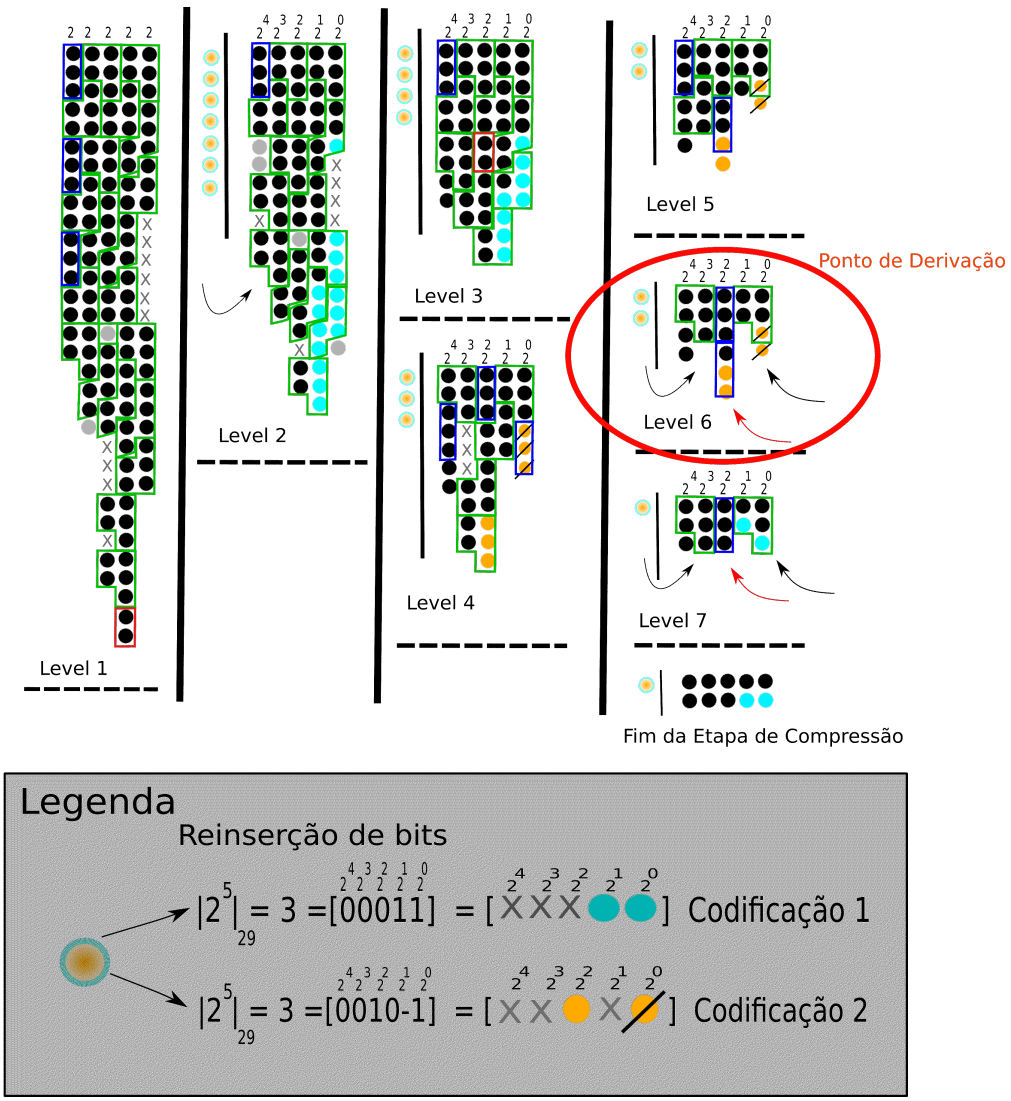
Figura 26 – Alocação de compressores - (Descrito em (LOW; CHANG, 2013))



Fonte – Elaborado pelo autor, 2021.

A cada nível de compressão deve-se reinserir bits com peso $|2^5|_{29}$. Este pode assumir dois valores distintos, sendo eles os valores de 3 ou -26 como sua representação positiva e negativa, respectivamente. Contudo, observa-se que o número 3 pode ser representado apenas por dois 1's [0 0 0 1 1], enquanto que para o número -26 é necessário três 1's [–1 – 1 0 – 1 0]. Dessa maneira, é optado pela escolha de reinserção dos bits possuindo o valor 3, uma vez que este possui menos valores diferentes de zero.

Figura 27 – Alocação de compressores - (Ferramenta Versão 1)

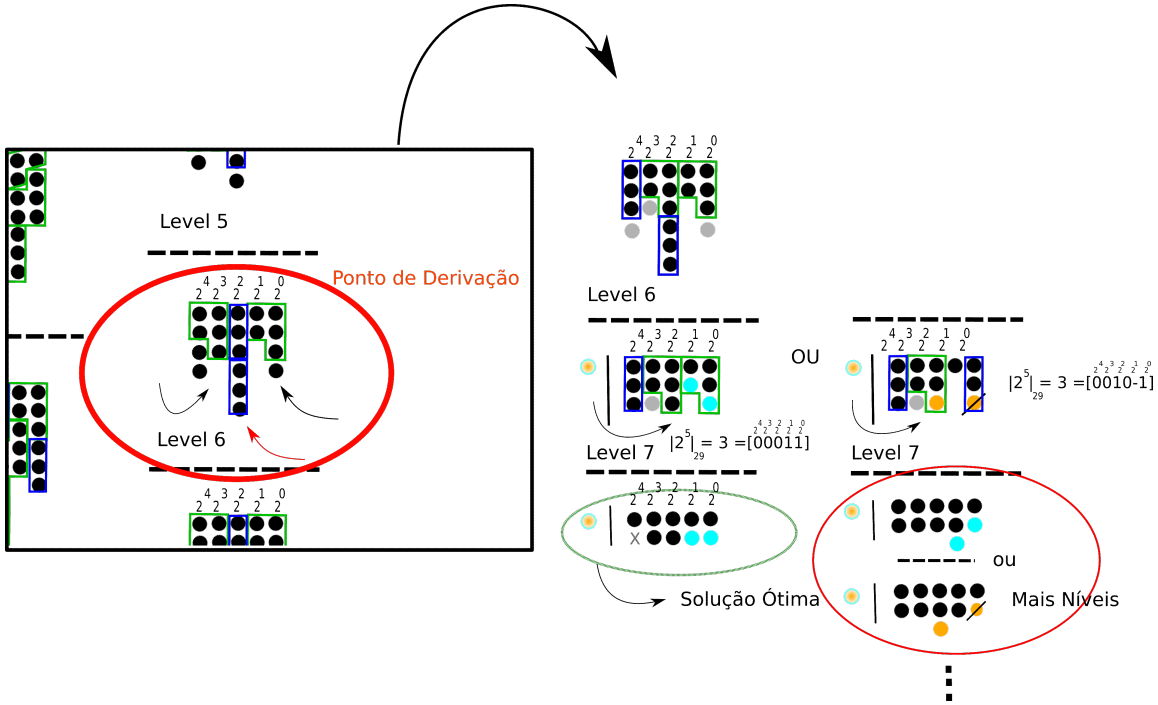


Fonte – Elaborado pelo autor, 2021.

Ainda sim, a reinserção do número 3 promove uma segunda possível escolha de representação em razão deste poder assumir a forma de $[0\ \ 0\ \ 0\ \ 1\ \ 1]$ e $[0\ \ 0\ \ 1\ \ 0-1]$, como representados na legenda da Figura 27. Observe que são realizadas escolhas distintas a cada nível para garantir uma compressão ótima. Os níveis 2, 3, 7 e realocação final fazem uso da codificação indicada em azul enquanto que os demais fazem uso da codificação indicada em laranja.

Com a compressão apresentada é possível chegar ao resultado de 7 níveis, enquanto que para o caso apresentado na Figura 26 são necessários 7 níveis para compressão inicial dos bits, mais 2 níveis para compressão das informações de *carry out* e finalmente mais 2 níveis aproximadamente de atraso devido à LUT, como indicado em (LOW; CHANG, 2013).

Figura 28 – Alocação de compressores - Caso de Derivação (Ferramenta Versão 1)



Fonte – Elaborado pelo autor, 2021

Apesar da compressão eficiente pelo método proposto, ainda sim é possível realizar melhoras no caso apresentado. Indica-se então um ponto de derivação na Figura 27 o qual será utilizado para demonstrar outras possíveis escolhas para gerar um resultado final distinto. Tais escolhas são apresentadas na Figura 28. Nela é possível observar tanto modificação nas colunas em que os compressores são colocados quanto escolhas distintas nos pesos de retorno do *carry out*.

Vejamos então a solução proposta para derivação e que encontra-se circulada em verde. Esta possui o custo de 7 níveis e 124 FAs, tendo então custo igual à solução original demonstrada anteriormente. Entretanto, é possível observar que esta nova solução gera um elemento igual a zero, representado na Figura por um "X". Esta nova solução seria mais eficiente, uma vez que a presença de tal zero simplificará a etapa de soma final sem adicionar custo à etapa de compressão, quando comparada à solução anterior.

Uma comparação entre o número de níveis e número de *Full Adders* necessários para comprimir a matriz de informação pelos diferentes métodos apresentados é realizada na Tabela 7. Para estimativa de área da LUT, foi realizada a síntese de uma ROM para um exemplo de compressão de 5 bits. Esta apresentou a área igual à aproximadamente 2 Full Adders.

Tabela 7 – Comparações entre circuito do estado da arte e ferramenta versão 0 e versão 1

Compressor (LOW; CHANG, 2013)	Níveis	Área FA
[3:2],[2,2]	11	105
Compressor (Ferramenta Versão 0)	**Níveis**	**Área FA**
[5:3],[3:2],[2,2]	8	127
[5:3],[3:2]	8	125
[3:2],[2,2]	11	125
[3:2]	11	125
Compressor (Ferramenta Versão 1)	**Níveis**	**Área FA**
[5:3],[3:2],[2,2]	7	123
[5:3],[3:2]	7	124
[3:2],[2,2]	10	125
[3:2]	9	123

Fonte – Elaborado pelo autor, 2021.

4.2.2 Caso 2

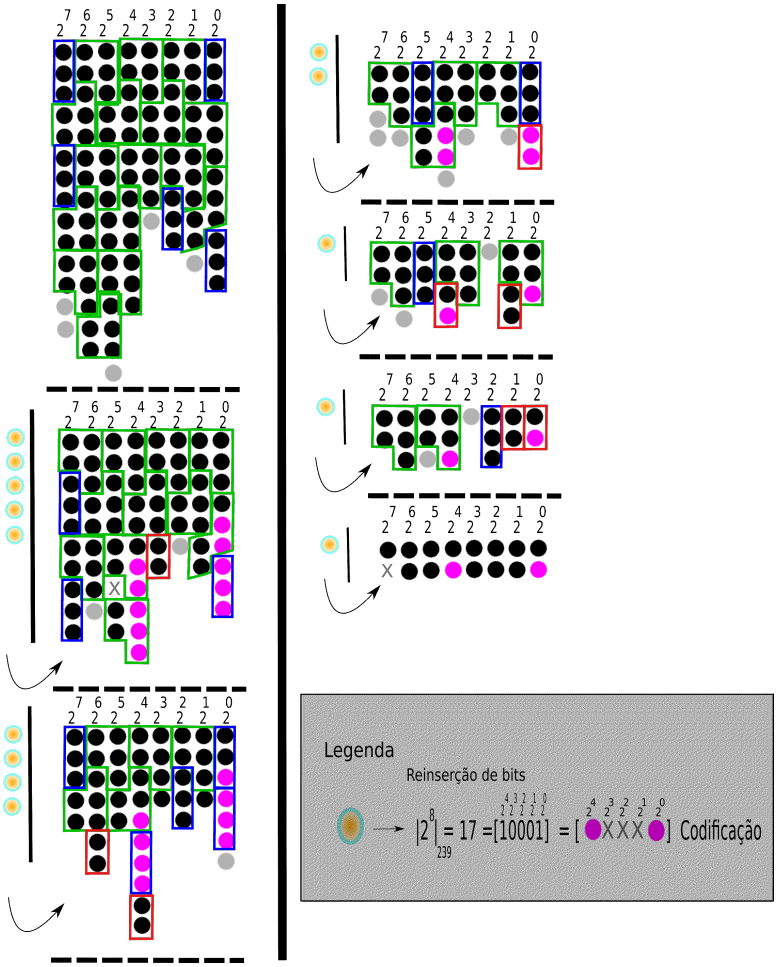
Apesar da variação nas recodificações dos pesos reinseridos na matriz de informação, o leitor atento perceberá que a solução ótima, indicada no exemplo anterior, coincidentemente é atingida por meio da colocação sequencial de compressores como havia sido feita na versão inicial da ferramenta que gerou este trabalho. Para demonstrar a necessidade real da variação de posicionamentos dos compressores, é apresentado o exemplo de multiplicação modulo 239 entre variáveis na Figura 29.

Neste exemplo, os pesos reinseridos $|2^8|_{239}$ assumirão o valor igual a 17 e terão codificação única de [0 0 0 1 0 0 0 1], sendo o bit menos significativo localizado à direita. Com uma colocação sequencial, como na versão inicial da ferramenta, não é possível alcançar a compressão em 6 níveis. Observe que nos níveis 4 e 5 fica evidente a sequência de compressores [5:3] sendo interrompida por *Full Adders* entre eles.

Tal solução é gerada a partir do processo de 100 iterações. Neste processo, são alcançadas 4 casos de compressão com número mínimo de 6 níveis, como demonstrado na Tabela 8. Dentro dos quatro casos a solução da iteração de número 22 é selecionado como solução ótima, visto que ela possui a menor área com a maior presença de zeros na saída do compressor. Logo, para escolha das soluções, são avaliados os quesitos em ordem de prioridade, sendo eles:

1. Nível
2. Área
3. Número de zeros na saída do compressor

Figura 29 – Solução de compressão Modulo 239 - Caso 2



Fonte – Elaborado pelo autor, 2021.

Tabela 8 – Resultados de iterações

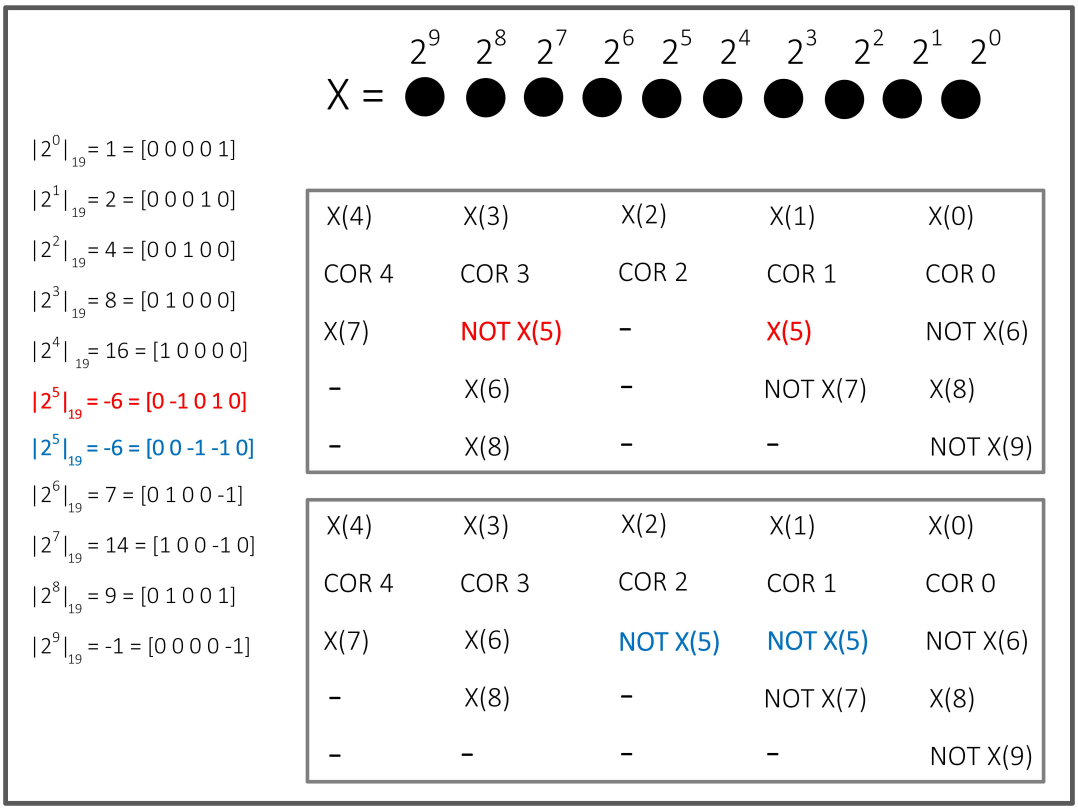
Iteração	Níveis	Área FA	Zeros na saída do compressor
14	6	103.5	1
22	**6**	**103.0**	**1**
25	6	103.5	1
82	6	103.0	0

Fonte – Elaborado pelo autor, 2021.

Por fim, como elemento final de otimização do processo de compressão, também é utilizado o processo de representação MSD para dar variabilidade no processo de formação inicial dos vetores que compõe a matriz de compressão. Vejamos então a Figura 30, a qual é referente à computação de vetores para um conversor direto módulo 19. Perceba que o peso $|2^5|_{19}$ = -6 uma vez que tal representação garante o uso do menor número de 1s. Entretanto note que o número -6 pode ser representado

de duas formas diferentes. Estas duas formatações pode levar à mudança na alocação dos elementos dentro da matriz inicial. Neste exemplo há apenas uma pequena modificação, dado que o número -6 é o único elemento em que é possível dar esta variabilidade de representação. Contudo, existem casos em que é possível construir uma variedade de formatos, na matriz inicial, por meio de tal técnica. Ao variar estes formatos, o algoritmo é capaz de encontrar melhores encaixes para os compressores e em consequência, diminuir o número de níveis.

Figura 30 – Exemplo de formatação inicial para compressão.



Fonte – Elaborado pelo autor, 2021.

4.3 OTIMIZAÇÕES SOMADORES

Para realização da somas finais, da proposta apresentada por este trabalho, foram comparados dois circuitos do estado da arte. O primeiro, apresentado nas seção anterior (PIESTRAK, S., 1994) e ilustrado na Figura 31 a) constitui-se como um circuito bastante difundido na literatura, enquanto que o segundo, ilustrado na Figura 31 b) e introduzido em (HIASAT, A.A., 2002) constitui-se como uma alternativa para redução de área.

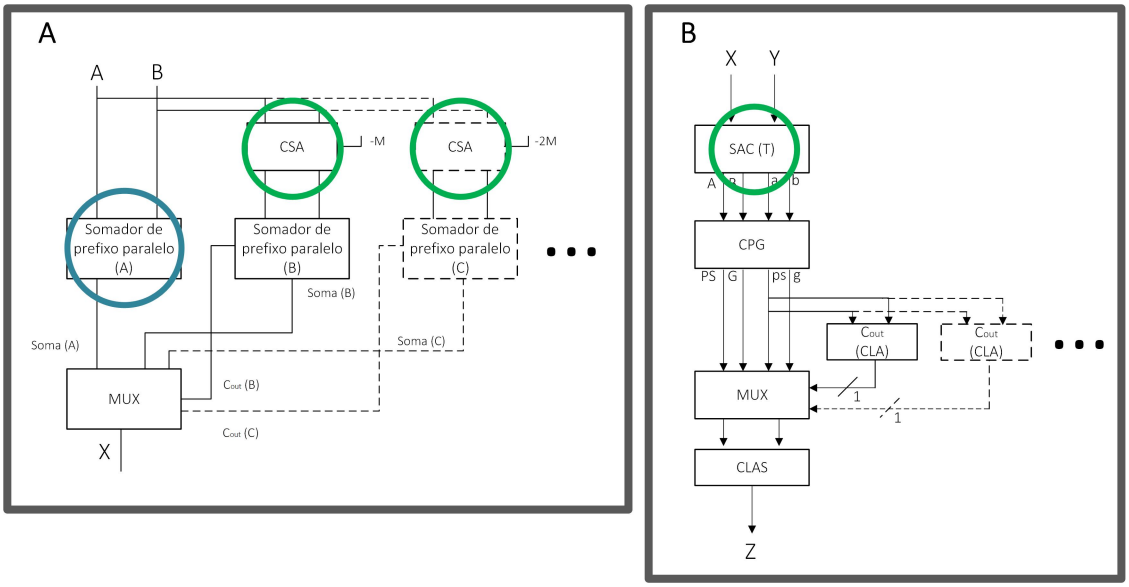
Tais circuitos possuem funcionamento similar. Enquanto as subtrações -M,-2M ou -3M são realizadas pelas unidades CSA na arquitetura *Piestrak* (PIESTRAK, S.,

1994), tais subtrações são realizadas na chamada unidade Sum and Carry (SAC) na proposta *Hiasat* (HIASAT, A.A., 2002).

Para o circuito *Piestrak*, para realização de tais subtrações, como visto anteriormente, faz-se necessário a anexação de somadores de forma paralela ao circuito. Já para o circuito *Hiasat*, o bloco Carry Propagate and Generate (CPG) é responsável pelo cálculos dos sinais *propagate* e *generate*, enquanto que o bloco C_{out} realiza as relações de prefixo desses sinais. São adicionados blocos de cálculo C_{out} caso seja necessário mais representações devido à ultrapassagem do valor por duas ou três vezes o módulo . Desta forma, percebe-se que os blocos C_{out} constituem-se de forma similar à adição de um somador de forma paralela como na arquitetura *Piestrak*.

Considerando uma entrada em que não se possa prever o número de zeros que cada operador pode assumir, existirá a possibilidade de simplificação do circuito apenas nos blocos de CSA ou o bloco SAC como demonstrado na Figura 31, circulados em verde. Ou seja, para módulos com o mesmo número de bits, a mudança em área será proporcional ao número de zeros que existirá na representação de complemento de dois binário do módulo, que será somado aos operadores de entrada. A Figura 32 exemplifica as funções lógicas que poderão ser simplificadas para uma operação de módulo 29, dado que o complemento de dois de 29, em binário, é '11000'.

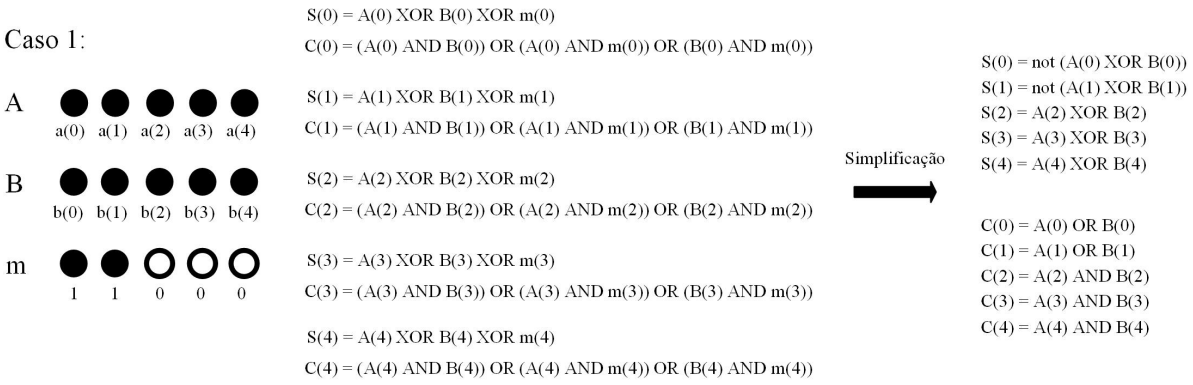
Figura 31 – Blocos de Simplificação do somador modular.
a) *Piestrak* b) *Hiasat*



Fonte – Elaborado pelo autor, 2021.

Já, caso seja conhecida as entradas do somador a priori e consequentemente, caso seja conhecido as entradas iguais a zero, poderá ser conduzida simplificações tanto nos CSAs que encontrem-se nas estruturas paralelas para subtrações -M,-2M e

Figura 32 – Equações de Simplificação do CSA.

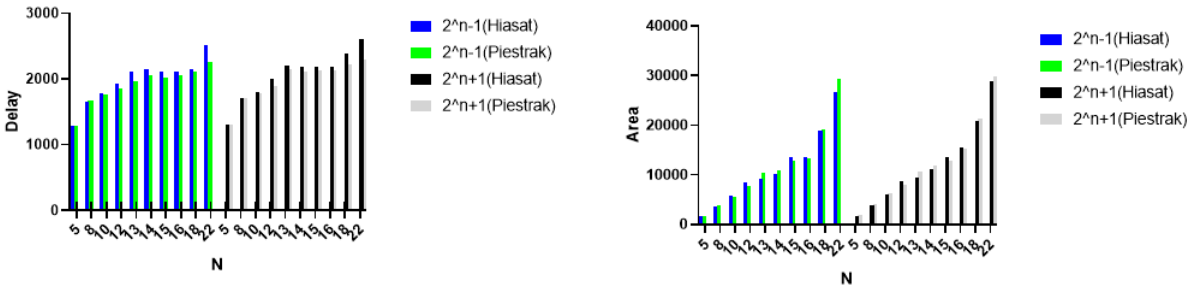


Fonte – Elaborado pelo autor, 2021.

-3M, como também no somador de prefixo paralelo A da arquitetura Piestrak, circulados em azul na Figura 31. Para este último, cada sinal de prefixo é quebrado e alocado em uma variável diferente. Uma determinada função inicial possuirá uma outra função subsequente, formando uma hierarquia. A partir de uma sequência, cada função simplificada será utilizada como entrada para sua função subsequente e assim serão verificadas se possíveis simplificações irão se propagar a partir de seus elementos precedentes. Ao fim a redução de área irá depender do número de zeros nas entradas.

O somador a ser utilizado pelo compressor é calculado pela ferramenta automática de acordo com o número de zeros nas saídas. Entretanto, de maneira geral operações do tipo $2^n \pm 1$ farão uso de somadores que necessitam apenas de uma subtração -M. Já para os casos $2^n - k$ teremos a necessidade de subtração por -M e -2M. Por fim, para os casos $2^n + k$ serão necessárias as 3 subtrações -M, -2M e -3M.

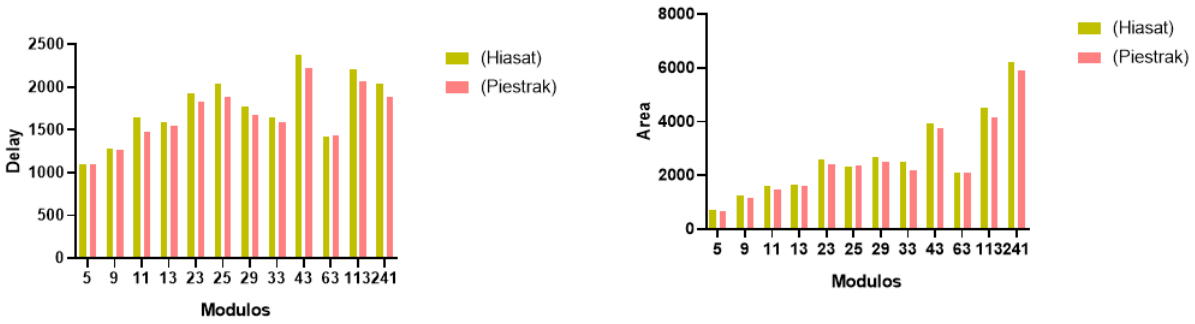
Figura 33 – Comparação entre arquiteturas de adição modular $2^n - 1$



Fonte – Elaborado pelo autor, 2021.

Para comparação entre os somadores finais, foi sintetizado circuitos multiplicadores em casos que utilizam o somador *Piestrak* ou o somador *Hiasat* para resolução

Figura 34 – Comparação entre arquiteturas de adição modular $2^n \pm k$



Fonte – Elaborado pelo autor, 2021.

da soma final. Utilizou-se a síntese de multiplicadores completos com os blocos de compressão e soma final, ao invés de uma comparação apenas do circuito de soma separados, pois houve o objetivo de avaliar o impacto das simplificações por zero a medida que os somadores fossem conectados aos blocos compressores.

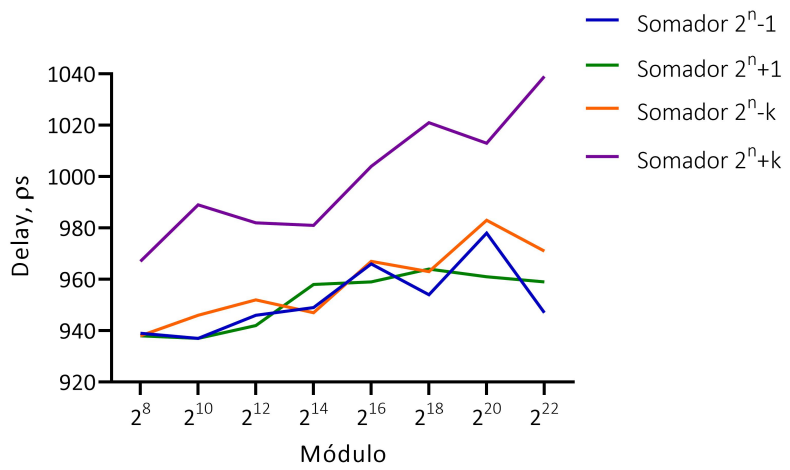
Apesar de apresentar possíveis reduções de área, a arquitetura *Hiasat* depende do número de zeros no valor de representação binária do módulo, para alcançar maiores reduções. Como demonstrado nas Figuras 33 e 34, e de acordo com comparações similares feitas em (PATEL *et al.*, 2007), a arquitetura *Hiasat* possui atraso mais elevado do que quando comparado à solução *Piestrak*. É possível notar que para módulos 2^n-1 e 2^n+1 existe ganho em área para o *Hiasat*, uma vez que tais módulos possuem poucos 1s em suas representações. Já para módulos $2^n \pm k$ pode ser notado que existem possibilidades de ganho em área em alguns módulos contudo de forma marginal, para estes apresentados. Diante destes fatos, foi escolhido a arquitetura *Piestrak* como estrutura de formação dos circuitos de multiplicação, conversores diretos e reversos que serão apresentados adiante.

As Figuras 35 e 36 demonstram as relações de área e atraso para apenas o circuito de soma *Piestrak* em função do valor do módulo.

4.4 COMPARAÇÕES ENTRE VERSÕES DA FERRAMENTA

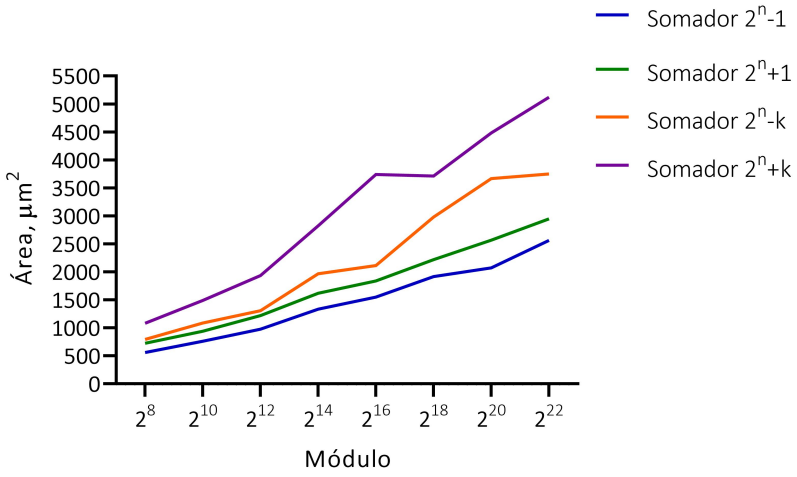
Deste modo, são apresentadas as comparações entre versão 0 e versão 1 da ferramenta para **multiplicadores** do tipo $2^n \pm k$ na Tabela 9. Nos resultados fica evidenciado o objetivo de alcançado de compressão em um número de níveis menores, reduzindo o atraso dos circuitos. Já para o requisito de área, por vezes é possível chegar em soluções mais eficientes. Contudo, neste quesito, não há uma consistência para todos os casos, havendo a produção de circuitos maiores. Elevações de área

Figura 35 – Atraso referente aos somadores modulares



Fonte – Elaborado pelo autor, 2021.

Figura 36 – Área referente aos somadores modulares



Fonte – Elaborado pelo autor, 2021.

substantivas ocorrem em alguns casos quando se é utilizado compressores [2:2] como discutido no caso onde foi apresentado multiplicação modular $2^n + 1$, na seção 3.3. Alguns padrões forçam a utilização de um número elevado de compressores [2:2] para garantir a compressão no menor número de níveis. No entanto, uma vez que tais compressores não comprimem informação, mas sim realizam apenas deslocamentos, pode haver um aumento de área deste circuito.

De modo geral o objetivo de aceleração dos módulos $2^n \pm k$ é alcançado, tra-

Tabela 9 – Comparações - Versões da Ferramenta para Multiplicadores.

Modulo	Ferramenta Versão 0		Ferramenta Versão 1	
	Atraso (ps)	Área (μm^2)	Atraso (ps)	Área (μm^2)
13	1543	1611	**1463**	**1526**
29	1678	2505	**1632**	**2493**
43	2111	**3575**	**2052**	4406
27	1683	**2522**	**1640**	2869
19	1938	**2746**	**1803**	3061
25	1812	**2396**	**1739**	2759
683	2736	**11655**	**2558**	12881
61	1883	3960	**1669**	**3658**

Fonte – Elaborado pelo autor, 2021.

zendo a possibilidade de aplicação em cenários em que se requer, por exemplo, ampla faixa dinâmica, com reduções de atraso de até 12%. Um exemplo de projeto será dado na próxima seção, o qual fará uso de tais módulos.

Já os resultados para conversores diretos e reversos podem ser observados na Tabelas 10 e 11 respectivamente. Para tais casos, também são comparados operações modulares do tipo $2^n \pm k$ cujo é o foco deste trabalho. Os ganhos para o conversor direto seguem a mesma tendência de redução de atraso, demonstrando melhora na alocação de compressores e eventual redução no número de níveis de compressão. Em relação a área, de modo similar, existem casos de redução e aumento.

Já para os conversores reversos, é necessário relembrar que a sua construção é dependente de constantes geradas pelos algoritmos do CRT ou Novo-CRT-I, por exemplo. Para os resultados da Tabela 11, estas constantes foram recodificadas apenas na forma canônica, ou seja, sem a presença de 1s adjacentes. Devido a isto, as matrizes de compressão inicial, para tal caso foram iguais tanto para ferramenta versão 0 quanto para versão 1. Os resultados obtidos pela ferramenta 1 foram iguais aos da ferramenta 0 em número de níveis de compressão e em número de FAs utilizados. Portanto os resultados demonstrados refletem apenas uma variabilidade que a ferramenta de síntese gerou a partir de descrições similares de um mesmo circuito. Mesmo tendo a alocação de compressores de forma igual, a diferença entre as resoluções da versão 0 e 1 deu-se apenas por questões como diferença nomenclatura dos compressores em cada coluna e de forma geral, pequenas diferenças na descrição, mesmo em circuitos em que permanece funcionalidade lógica idêntica, houve uma diferença em como a ferramenta de síntese interpretou os determinados circuitos. Apesar disto os resultados ainda foram próximos, tanto para área quanto para atraso.

Ainda no sentido dos resultados de otimização, visto que a metodologia fornece resultados ainda mais competitivos em relação a atraso comparado ao estado da

Tabela 10 – Comparações - Versões da Ferramenta para Conversores Diretos

Modulo	Variável de Entrada	Ferramenta Versão 0		Ferramenta Versão 1	
		Atraso (ns)	Área (μm^2)	Atraso (ns)	Área (μm^2)
23	25Bits	1635	**2380**	**1544**	2567
41	25Bits	1587	2444	**1574**	**2210**
235	25Bits	1602	**2690**	**1589**	3183
267	25Bits	1727	**3045**	**1550**	3757
119	30Bits	1808	**3974**	**1691**	4057
331	30Bits	1723	4168	**1709**	**3903**

Fonte – Elaborado pelo autor, 2021.

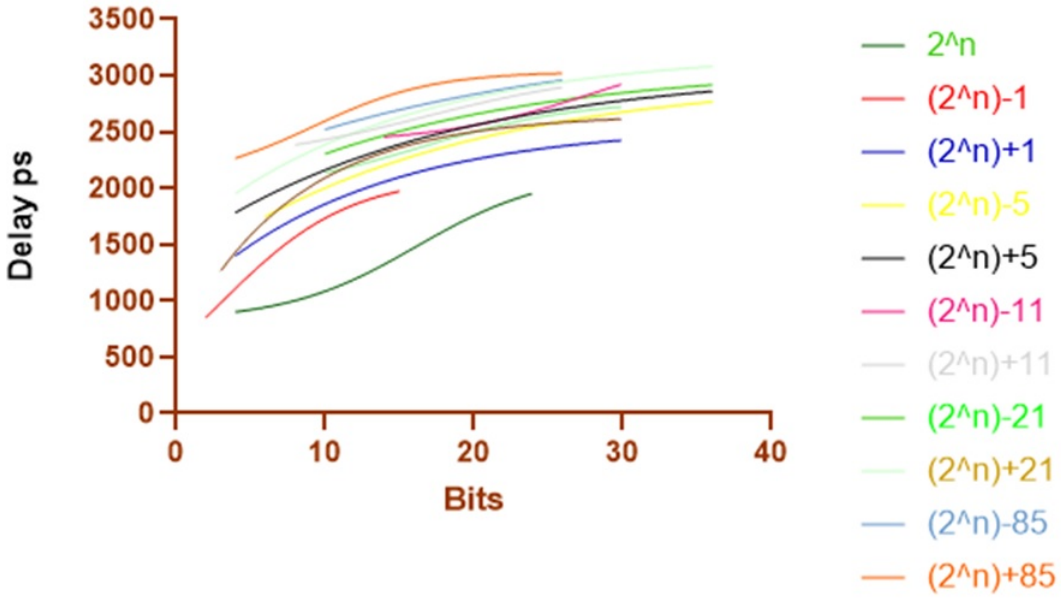
Tabela 11 – Comparações - Versões da Ferramenta para Conversores Reversos

Usando Novo-CRT-I				
Conjunto Modular	Ferramenta Versão 0		Ferramenta Versão 1	
	Atraso (ns)	Área (μm^2)	Atraso (ns)	Área (μm^2)
16,15,17	1440	**2752**	**1370**	2975
32,31,33	1528	**3847**	**1512**	4196
64,63,65	1622	**5262**	**1593**	5773
Usando CRT				
Conjunto Modular	Ferramenta Versão 0		Ferramenta Versão 1	
	Atraso (ns)	Área (μm^2)	Atraso (ns)	Área (μm^2)
(31,33)	**1080**	1166	1127	**1130**
(63,65)	**1129**	**1852**	1134	1969
(255,257)	1208	**2222**	**1194**	2240
(1024,1025)	**1199**	3530	1260	**3129**
(7,31,151)	**1705**	**4908**	1732	5006
(127,3,43)	**1384**	**3465**	1409	3520
(9,11,331)	**1704**	7610	1712	**7549**
(3,5,29,43,113,127)	2046	**12331**	**2045**	12989

Fonte – Elaborado pelo autor, 2021.

arte, é de grande importância demonstrar como multiplicadores genéricos $2^n \pm k$ se comportam em relação aos módulos 2^n,$2^n - 1$ e $2^n + 1$. Esta avaliação também tem o objetivo de demonstrar módulos os quais são favoráveis a operações aritméticas, impactando assim na escolha do conjunto modular para o projetista.

Figura 37 – Atraso Multiplicadores Modulares



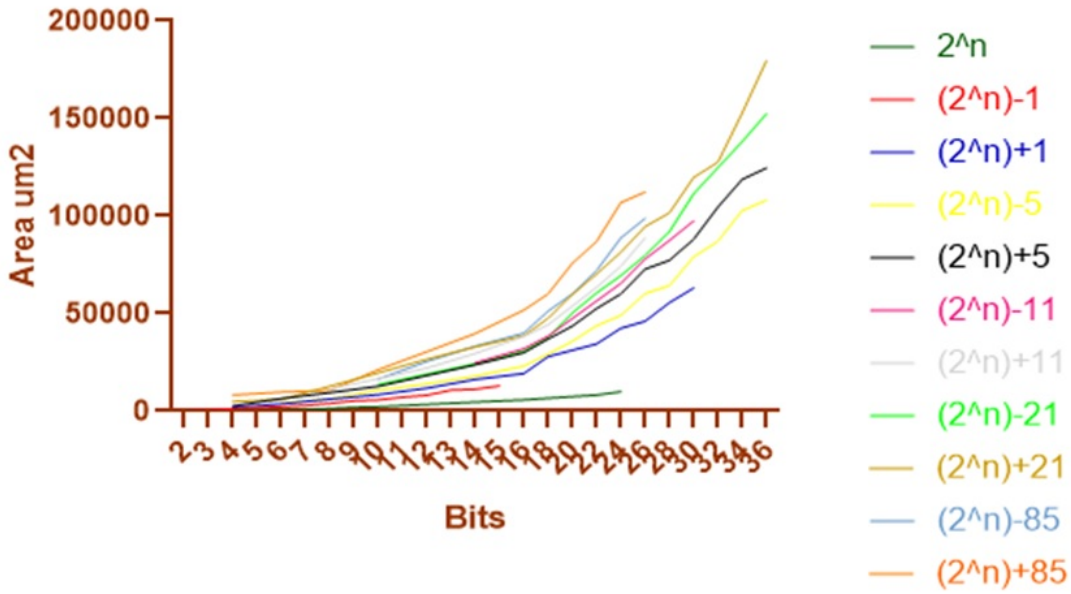
Fonte – Elaborado pelo autor, 2021.

Na Figura 37 é demonstrado o desempenho em atraso dos multiplicadores modulares. Nota-se que tal atraso é menor para 2^n,$2^n - 1$ e $2^n + 1$ como indicado na literatura. Já para os módulos $2^n \pm k$ existe uma relação entre atraso e número de 1s presentes no valor de K. Ou seja, para $k = 5 = [101]$ teremos atraso menor do quando $k = 21 = [10101]$ e assim por diante. Quando k possuir o mesmo número de 1s, existirá uma semelhança entre os resultados para tais módulos, por exemplo, quando $k = 3[011]$ e $k = 5[101]$.

Já em termos de área, como demonstrado na Figura 38 , as tendências seguirão de forma similar ao que já foi descrito. Ou seja, maior eficiência para módulos do tipo 2^n,$2^n - 1$ e $2^n + 1$, seguido pelos módulos $2^n \pm k$ os quais possuem proporcionalidade com o número de 1s presentes no elemento k.

Por fim, como componente de conclusão da apresentação do comportamento dos circuitos otimizados, será demonstrado a contribuição de área e atraso dos blocos de soma final e compressão dentro de circuitos completos de multiplicadores modulares. O intuito de tal análise é demonstrar a contribuição de cada bloco dentro da

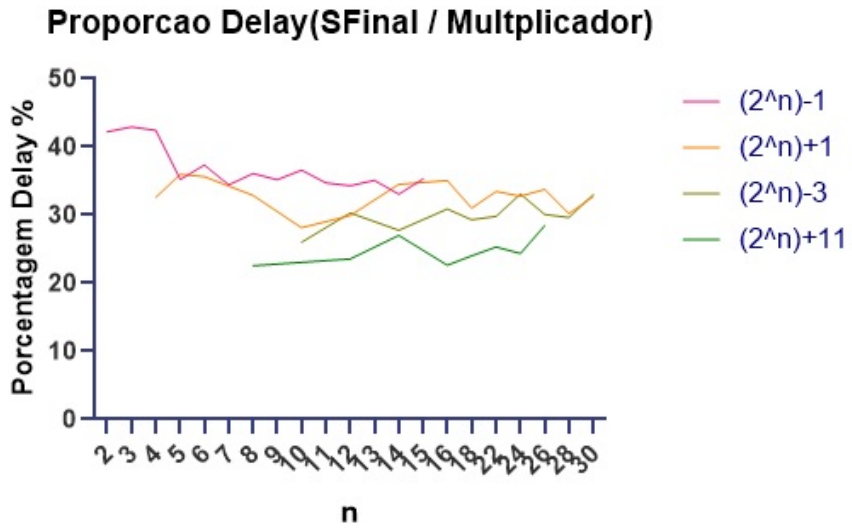
Figura 38 – Área Multiplicadores Modulares



Fonte – Elaborado pelo autor, 2021.

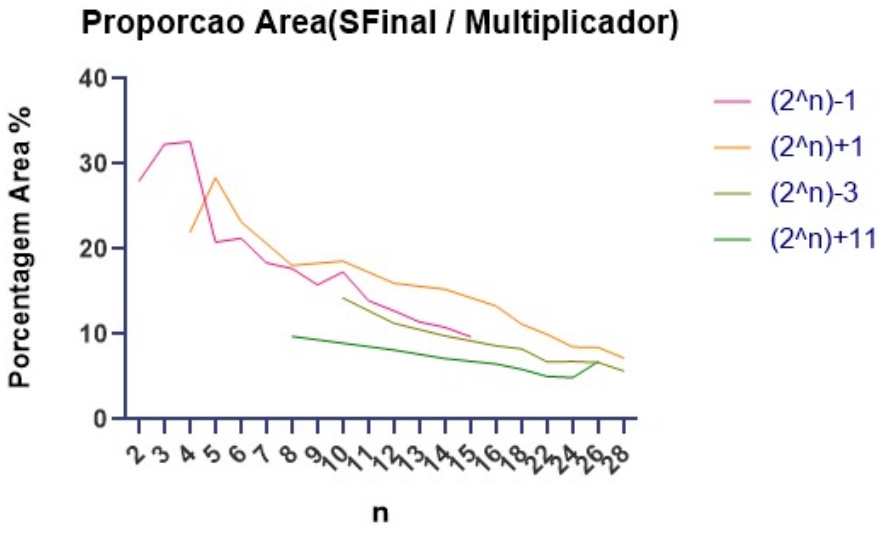
aplicação em camada aritmética. Na Figura 39 é possível observar a proporção do somador final que compõe o multiplicador. Nota-se que o somador é responsável por, em média, 30% do caminho crítico dos multiplicadores. Esse valor é maior para os módulos $2^n - 1$ e decresce ao longo dos módulos $2^n \pm k$. Isso significa que o compressor é responsável pela maior parte do caminho crítico para estes casos. Já em relação a área, demonstrado na Figura 40, a proporção do somador final decresce a medida que cresce o número de bits. Tal proporção inicia em 30% da totalidade do multiplicador, para o módulo $2^n - 1$ com $n = 2$ e decresce para 10% quando atinge-se $n = 15$. Novamente, tal fato significa que o compressor é o elemento de maior contribuição tanto para área quanto para atraso em um somador modular.

Figura 39 – Proporção de atraso para Somador Final



Fonte – Elaborado pelo autor, 2021.

Figura 40 – Proporção de Area para Somador Final



Fonte – Elaborado pelo autor, 2021.

5 AVALIAÇÃO DE DESEMPENHO DE MULTIPLICADORES UTILIZANDO CONJUNTOS DE MÓDULOS FATORADOS

Com as otimizações apresentadas, agora propõe-se demonstrar uma solução para um problema completo baseado no sistema numérico residual. Nos trabalhos voltados à pesquisa em RNS é dada grande atenção na etapa de conversão do RNS para o binário, uma vez que tais circuitos apresentam complexidade elevada. Vários trabalhos têm se concentrado neste assunto, apresentando conjuntos de módulos e suas arquiteturas correspondentes.

Por essa razão, a escolha do conjunto de módulos é de fundamental importância para a obtenção de implementações eficientes. Isso se deve ao fato de que o conjunto de módulos definirá não apenas a operação modular na aritmética, mas também a própria conversão reversa. Os algoritmos de conversão RNS incluem operações modulares, que estão relacionadas ao conjunto de módulos escolhido. Apesar de algumas das soluções mais eficientes serem escaláveis, o nível de paralelismo não é suficiente para os intervalos dinâmicos (DR) exigidos pelas aplicações de processamento digital de sinais. Para aumentar o DR, pode-se adicionar mais módulo no conjunto ao custo de conversores reversos com pior desempenho.

A proposição aqui exposta vem com o objetivo de utilizar uma condição para a escolha do conjunto de módulos a fim de fornecer o conversor reverso mais eficiente existente no estado da arte e, consequentemente, realizar uma comparação justa do desempenho dos canais de aritmética modular. Esta condição consiste em impor operações modulares do tipo $2^{\alpha} - 1$ ao algoritmo reverso, tornando a decomposição deste valor nos módulos do conjunto modular.

5.1 EXPLORAÇÃO DA FATORAÇÃO COMO ESTRATÉGIA DE FORMAÇÃO DO CONJUNTO MODULAR

A fatoração do valor $2^{\alpha} - 1$ não foi amplamente explorada no estado da arte primeiramente devido à este normalmente fornecer números primos no conjunto e, em segundo lugar, não fornecer soluções escaláveis. No entanto, será demonstrado que a aritmética, baseada em multiplicações de variáveis, pode ser implementada de forma mais eficiente em comparação com o conjunto de módulos genéricos mais amplamente utilizado.

No estado da arte, a decomposição de $2^{\alpha} - 1$ em módulos menores é amplamente utilizada, e adicionalmente dando preferência a módulos com formato $2^{n} - 1$ e $2^{n} + 1$. Isto ocorre devido a este formato normalmente fornecer melhor desempenho do que os módulos genéricos do tipo $2^{n} \pm k$. Outra sugestão nesta temática é demonstrada em (PETTENGHI; CHAVES; SOUSA, 2013), no qual foi apresentada a decomposição $2^{4n} - 1 = (2^{n} - 1)(2^{n} + 1)(2^{n} - 2^{\frac{n+1}{2}} + 1)(2^{n} + 2^{\frac{n+1}{2}} + 1)$ para valores ímpares de $n \geq 5$

obtendo $\{m_1, m_2, m_3, m_4\} = \{2^n - 1, 2^n + 1, 2^n - 2^{\frac{n+1}{2}} + 1, 2^n + 2^{\frac{n+1}{2}} + 1\}$. Essa abordagem fornece dois módulos no conjunto que não estão na forma $2^n - 1$ ou $2^n + 1$. Porém, a aritmética pode ser implementada de forma mais eficiente do que outros módulos genéricos, como será exposto adiante.

Por fim, a decomposição de $2^{6n} - 1$ em oito módulos foi apresentada em (PETTENGHI; MATOS; MOLAHOSSEINI, 2016) com uma limitação dos valores de $n = 6,10,14,\ldots$ e consequentemente uma restrição na granularidade. A principal vantagem de todas essas decomposições é a escalabilidade dos módulos definidos em termos de n e o equilíbrio do número de bits por canal.

A fatoração de $2^\alpha - 1$ fornece os menores valores de módulo possíveis para um conjunto modular. No entanto, o fato de que geralmente não são valores no formato $2^n - 1$ ou $2^n + 1$ significa, a priori, que sua aritmética não é tão vantajosa. Outra razão pela qual essa solução não foi amplamente explorada no estado da arte é a sua impossibilidade de escalonamento com n. Porém, será demonstrado que a fatoração pode fornecer conjuntos de módulos com soluções mais eficientes para as operações aritméticas em comparação com as decompostas apresentadas no estado da arte. Por fim, outro aspecto a levar em consideração é que alguns valores de α não permitem decomposição, mas fatoração. Por exemplo, $2^\alpha - 1$ com $\alpha = 11$, não pode ser decomposto em um conjunto de módulos com módulo no formato $2^n - 1$ e $2^n + 1$. Contudo, pode ser fatorado em $2^{11} - 1 = 23\ x\ 89$ fornecendo $\{m_1, m_2\} = \{23,89\}$.

5.2 ARITMÉTICA DOS MÓDULOS FATORADOS

As operações aritméticas que podem ser implementadas com eficiência no RNS são adição, subtração e multiplicação. Como as multiplicações são as operações mais comuns em aplicações DSP, será dado foco nas multiplicações entre variáveis. A implementação de hardware é realizada por meio da arquitetura proposta e discutida nos capítulos anteriores, a qual faz uso de compressores similares à uma árvore de Wallace com reinserção de bits de *carry out* à matriz.

Na Tabela 12 é demonstrado a avaliação de desempenho dos multiplicadores para módulos propostos, com a propriedade $\prod_{i=1}^{M} m_i = 2^\alpha - 1$, comparados com outras decomposições. Na Tabela é possível observar que a fatoração do conjunto de módulos é vantajosa em alguns casos. A primeira coluna contém os conjuntos de módulos agrupados pelo mesmo valor de α. Já na segunda coluna é apresentado o atraso dos multiplicadores. Os valores em negrito indicam o menor caminho crítico para o conjunto de módulos analisado com o mesmo α. A terceira coluna indica a área por canal e, por fim, a última coluna demonstra a área total.

Em termos de atraso, no caso $2^\alpha - 1$, $\alpha = 24$, a decomposição clássica $2^n \pm 1$ fornece um atraso de 2,036 ns. Esta solução pode ser aperfeiçoada em termos de

Tabela 12 – Avaliação da performance de multiplicadores entre variáveis para conjuntos modulares com propriedade $\prod_{i=1}^{M} m_i = 2^{\alpha} - 1$

Moduli set	Delay per channel (ns)	Area per channel (um)	Area total (um)
$\{2^8-1\}$	{1.533}	{3798}	3798
$\{2^4-1, 2^4+1\}$	{1.099, **1.564**}	{1143, 2062}	3205
{3, 5, **17**} *Factor*	{0.960, 1.100, **1.564**}	{117, 664, 2062}	**2843**
$\{2^{10}-1\}$	{1.710}	{5530}	5530
$\{2^5-1, 2^5+1\}$	{1.258, 1.588}	{1493, 2170}	3663
{31, 3, **11**} *Factor*	{1.258, 0.960, **1.476**}	{1493, 117, 1491}	**3101**
$\{2^{12}-1\}$	{1.756}	{8400}	8400
$\{2^6-1, 2^6+1\}$	{1.310, 1.647}	{2044, 3493}	5537
{7, 9, 5, **13**} *Factor*	{1.000, 1.259, 1.100, **1.543**}	{486, 1151, 664, 1611}	**3912**
$\{2^{14}-1\}$	{1.918}	{11235}	11235
$\{2^7-1, 2^7+1\}$	{1.474, **1.753**}	{2924, 5576}	8500
{127, 3, **43**} *Factor*	{1.474, 0.960, 2.111}	{2924, 117, 3575}	**6616**
$\{2^{16}-1\}$	{2.048}	{12879}	12879
$\{2^8-1, 2^8+1\}$	{1.533, **1.869**}	{3798, 6447}	10245
{3, 5, 17, **257**} *Factor*	{0.960, 1.100, 1.564, **1.869**}	{117, 664, 2062, 6447}	**9290**
$\{2^{18}-1\}$	{2.061}	{17631}	17631
$\{2^9-1, 2^9+1\}$	{1.607, 1.960}	{4672, 7841}	12513
{7, **73**, 27, 19} *Factor*	{1.000, **1.872**, 1.683, 1.938}	{486, 4809, 2522,2746}	**10563**
$\{2^{20}-1\}$	{2.077}	{24840}	24840
$\{2^{10}-1, 2^{10}+1\}$	{1.710, 1.913}	{5530, 9117}	14647
$\{2^5-1, 2^5+1, 25,$ **41**$\}$	{1.258, 1.588, 1.812, **1.899**}	{1493, 2170, 2396, 3726}	9785
{31, 3, 11, 25, **41**} *Factor*	{1.258, 0.960, 1.476, 1.812, **1.899**}	{1493, 117, 1491, 2396, 3726}	**9223**
$\{2^{22}-1\}$	{2.224}	{26957}	26957
$\{2^{11}-1, 2^{11}+1\}$	{1.758, **1.975**}	{6481, 9572}	**16053**
{3, 23, 89, 683} *Factor*	{0.960, 1.821, 2.339, 2.736}	{117, 2412, 7232, 11655}	21416
$\{2^{24}-1\}$	{2.203}	{33315}	33315
$\{2^{12}-1, 2^{12}+1\}$	{1.756, 2.036}	{8400, 11655}	20055
{3, 7, 9, 13, 17, **241**} *Factor*	{0.960, 1.000, 1.259, 1.543, 1.564, **1.862**}	{117, 486, 1151, 1611, 2062, 5597}	**11024**
$\{2^{26}-1\}$	{2.281}	{37860}	37860
$\{2^{13}-1, 2^{13}+1\}$	{1.812, **2.039**}	{10132, 11890}	**22022**
{3, 2731, 8191} *Factor*	{0.960,2.927, 1.812}	{117, 18221, 10132}	28470
$\{2^{28}-1\}$	{2.323}	{45576}	45576
$\{2^{14}-1, 2^{14}+1\}$	{1.918, 2.124}	{11235, 16556}	27791
$\{2^7-1, 2^7+1, 113, 145\}$	{1.474, 1.753, 2.025, 2.141}	{2924, 5576, 3969, 4692}	17161
{3, 5, 29, **43**, 113, 127} *Factor*	{0.960, 1.100, 1.678, **2.111**, 2.025, 1.474}	{117, 664, 2505, 3575, 3969, 2924}	**13754**
$\{2^{30}-1\}$	{2.353}	{53973}	53973
$\{2^{15}-1, 2^{15}+1\}$	{1.991, **2.118**}	{11286, 17613}	28899
{7, 9, 11, 31, 151, 331} *Factor*	{1.000, 1.259, 1.476, 1.281, 2.509, 2.761}	{486, 1151, 1491, 1626, 7987, 9881}	**22622**

Fonte – Elaborado pelo autor, 2021.

equilíbrio se 3, 7, 9 e 13 forem implementados como $2^{12}-1$, e $2^{12}+1 = 17 \times 241$, obtendo atrasos de 0,960, 1,00, 1,259, 1,543, 1,564 e 1,862 para 3, 7, 9, 13, 17 e 241, respectivamente. Nesse caso, a fatoração fornece uma redução no caminho crítico de 7%.

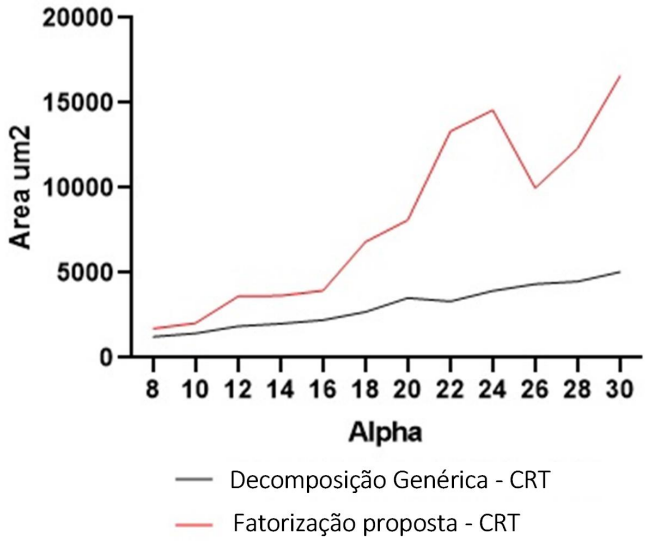
Alguns módulos, quando decompostos, podem se tornar iguais a um número primo na forma de 2^n+1, não podendo ser mais fatorados. Nestes casos, a solução fatorada apresentará um caminho crítico igual à instância $2^n-1,2^n+1$, como quando α = 8, 16 e 20.

Em termos de área, a fatoração proposta geralmente fornece um circuito total menor para os canais de aritmética. Exceções podem ocorrer quando os módulos produzem alguns números primos maiores que impedem a fatoração de distribuir a solução em vários módulos menores.

5.3 AVALIAÇÃO DE DESEMPENHO DE CONVERSORES REVERSOS

Ao aplicar o CRT nos conjuntos de módulos, a solução fatorada irá gerar mais constantes quando comparada com os pares $2^n-1,2^n+1$. Isso eventualmente levará a um número maior de matrizes a serem compactadas na árvore de compressão. Assim, em termos de área todas as soluções fatoradas apresentarão uma área maior, conforme ilustrado na Figura 41. Já, a análise do atraso dos reversos, deve ser realizado em conjunto com as camadas aritméticas para verificação dos ganhos globais do sistema.

Figura 41 – Comparação de área para conversores reversos

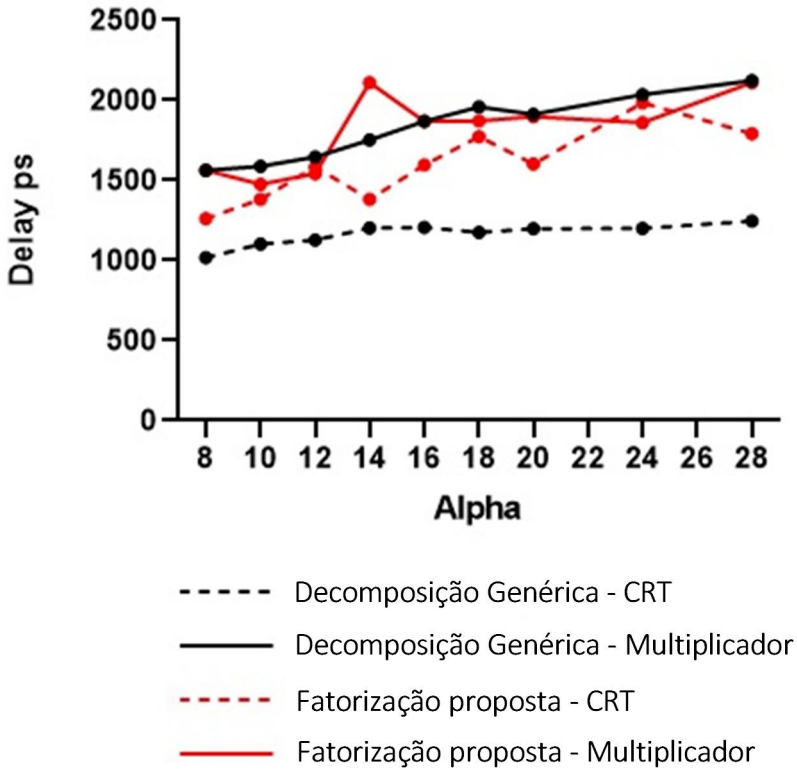


Fonte – Elaborado pelo autor, 2021.

5.4 GANHOS GLOBAIS DA PROPOSTA

Como afirmado antes, a fatoração também gerará casos em que grandes números primos impedem que a solução seja distribuída em vários módulos menores. Esses casos são considerados como fatoração malsucedida e são excluídos da análise posterior das contribuições de conversão reversa. Isso ocorre quando $\alpha = 22, 26, 30$. Na Figura 42 são ilustrados as comparações entre os caminhos críticos dos multiplicadores e conversores reversos baseados no CRT. É possível notar que, considerando apenas os conversores reversos, o atraso da abordagem de fatoração é maior do que a decomposição clássica. Contudo, é importante considerar que as soluções fatoradas fornecem um atraso mais equilibrado em relação aos seus respectivos canais aritméticos. Este fato pode ser interpretado pela grande aproximação das linhas que representam o atraso tanto para o multiplicador quanto para o CRT dos módulos fatorados. Devido à este fato, indica-se que o conjunto de módulos fatorados é adequado para o *pipelining*.

Figura 42 – Comparação de atrasos para fatoração e decomposição genérica.

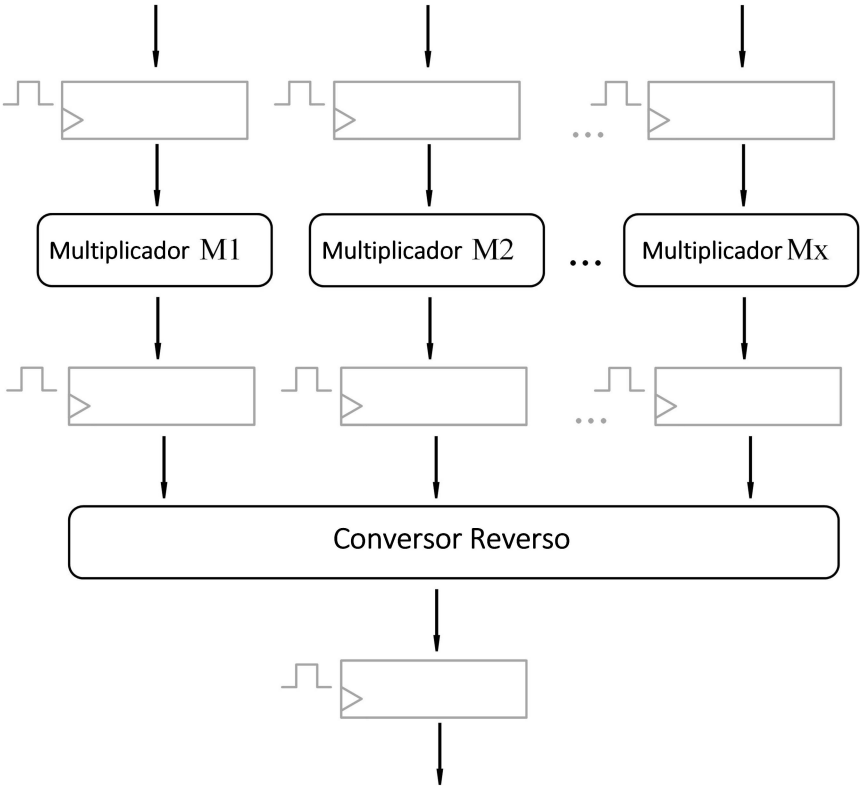


Fonte – Elaborado pelo autor, 2021.

Outro ponto a ser destacado é que o atraso dos conversores reversos geralmente não supera o atraso dos canais aritméticos, com o exceção de quando

$\alpha = 12, \alpha = 24$. A estrutura de pipeline proposta está presente na Figura 43 e consiste em um estágio de multiplicadores paralelos e um estágio de conversão reversa. Tomando o exemplo de $2^{\alpha} - 1$, $\alpha = 10$, o caso $2^5 - 1, 2^5 + 1$ tem um atraso máximo de 1,588 ns e 1.104 ns para seu canal aritmético e conversor reverso respectivamente, enquanto que o caso com a fatoração proposta possui um atraso máximo de 1,476 ns e 1,383 ns. Isso significa que a parte aritmética do circuito será aquela que ditará a velocidade do sistema na estrutura proposta de *pipeline*. Na avaliação feita, enquanto o caso $2^5 - 1, 2^5 + 1$ exigiria um sistema com caminho crítico intra-registradores de 1,588 ns, a solução fatorada exigiria 1,476 ns.

Figura 43 – Arquitetura em pipeline para conversor reverso.



Fonte – Elaborado pelo autor, 2021.

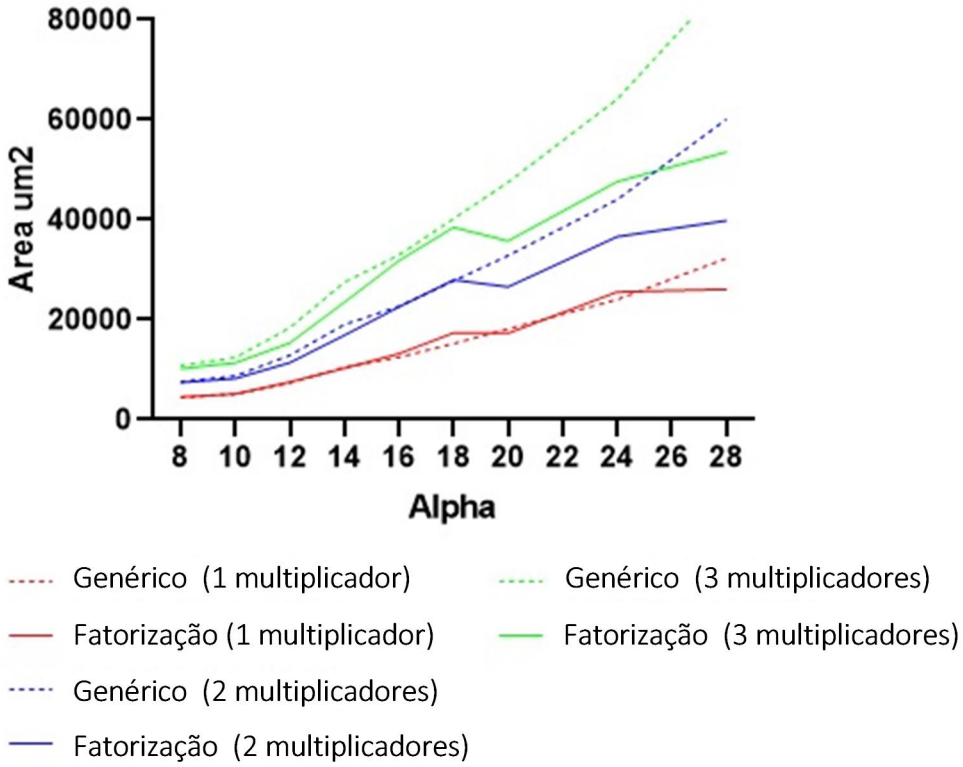
Para $\alpha = 12, \alpha = 24$ o atraso do conversor reverso fatorado é ligeiramente maior do que seu canal aritmético, mas ainda menor do que o multiplicador e conversor reverso da decomposição genérica. Portanto, neste caso, o elemento que ditaria o caminho crítico seria o conversor reverso dos módulos fatorados para a estrutura de *pipeline* proposta.

Já em termos de área, a fatoração proposta levará à um circuito maior para conversores reversos. Entretanto, conforme mencionado em seção anterior, a fatoração também proporcionará possíveis ganhos em termos de área nos canais aritméticos. Desta forma, para avaliar os ganhos globais, há a necessidade de verificar se os

ganhos na aritmética superam as perdas da conversão reversa. Há também a necessidade de considerar que em aplicações DSP, um processador RNS pode executar blocos de circuitos aritméticos em paralelo.

Na Figura 44 é exibido uma comparação entre a soma da área do canal aritmético e do conversor reverso para decomposição genérica e fatoração. Cada curva indica a área de um, dois ou três multiplicadores somados à área do conversor reverso. Ao usar apenas um multiplicador, percebe-se que os resultados dos casos $2^n-1,2^n+1$ e fatorados se aproximam. Além disso, o ganho nos canais aritméticos por vezes supera a perda da conversão reversa para a fatoração proposta. Portanto, em alguns casos, a área do método proposto é menor do que as decomposições $2^n-1,2^n+1$. Ao considerar dois multiplicadores, as distâncias entre as soluções se afastam e pode-se notar que, na maioria dos casos, a solução fatorada corre abaixo da linha tracejada que indica a solução genérica. A partir de 3 multiplicadores os ganhos de área para a fatoração proposta são sempre superiores.

Figura 44 – Comparação de área entre fatorização e decomposição genérica.



Fonte – Elaborado pelo autor, 2021.

Nesse sentido, o circuito proposto com aplicação de *pipeline* constitui-se como uma alternativa para equilibrar os atrasos das unidades aritméticas e conversores reversos demonstrando ganhos também em área.

Sob o aspecto de atraso, os ganhos presentes nas camadas aritméticas podem chegar a 7%. Há ocorrências em que a frequência do sistema será ditada tanto pela camada aritméticas quanto pela etapa de conversão reversa, mas ainda sim apresentando ganhos em certos casos.

6 CONCLUSÃO

Neste trabalho foi desenvolvida uma proposta de arquitetura eficiente para realização de somas modulares multi-operando. Arquitetura a qual pode ser aplicada para construção de conversores diretos e reversos, como também em operações aritméticas modulares existentes em aplicações de DSP. Em resposta ao problema do crescimento exponencial de área das arquiteturas baseadas no uso de LUTs, as quais são bastante difundidas no estado da arte, foi utilizada uma proposta baseada tanto no uso de unidades aritméticas nos blocos compressores como também a reinserção de bits de *carry-out* à própria árvore durante os níveis de compressão. As propostas de melhoria foram direcionadas para módulos do tipo $2^n \pm k$ os quais encontram-se fora da faixa mais convencionalmente explorada de 2^n, $2^n - 1$ e $2^n + 1$, com o objetivo de fornecer solução que possa ser utilizada em aplicações com ampla faixa dinâmica.

O trabalho partiu de ferramenta de software desenvolvida em um trabalho de tese de doutorado e demonstrou ganhos no atraso do tempo de computação do circuito de até 12%, quando comparado as versões inicial e aprimorada da ferramenta.

A maior contribuição foi dada a partir da criação de uma metodologia para alocação eficiente de compressores para resolução da estrutura voltada a realização de operações modulares, similar a uma árvore de Wallace. O algoritmo de alocação proposto fez uso de uma estratégia iterativa onde soluções globais são comparadas a cada iteração. Ao fim destas iterações, é selecionada a melhor solução, dando prioridade a aquela que atingiu a compressão em um menor número de níveis. Em seguida, de forma secundária, são observados os número de compressores utilizados e o número de zeros que irão ser gerados para etapa de soma final. Um elemento importante para alcance dos resultados apresentados foi garantir a variabilidade na representação dos pesos que formam as matrizes de informação a serem comprimidas. Nomeadamente, foram utilizadas as possíveis representações sob a forma MSD.

Outro elemento desenvolvido por esta dissertação foi uma proposta de escolha do conjunto modular a partir da propriedade $\prod_{i=1}^{M} m_i = 2^{\alpha} - 1$, a qual demonstrou possíveis ganhos de até 7% no atraso quando comparado à conjuntos modulares do estado da arte.

A partir dos resultados e do que foi discutido nesta dissertação existe a sugestão de novas investigações como elencado abaixo.

Trabalho Futuros

- Explorar a utilização de compressores diversos descritos na literatura, os quais não foram utilizados na dissertação, como por exemplo os compressores [7:3] (GHASEMZADEH *et al.*, 2015), [4:2] (WEINBERGER, 1981) e [2,2:3] (MADON, 1993), com o intuito de verificar possíveis reduções de atraso ao utilizar a metodologia de alocação de compressores apresentada.

- Verificar a viabilidade de utilização de pipeline aplicado aos níveis internos do compressor, dividindo a etapa de compressão em diferentes blocos, com o intuito de balanceamento das etapas das operações modulares ou dos canais paralelos da aritmética.
- Avaliar a possibilidade de redução do número de níveis ao utilizar um padrão de saída diferente para os compressores. Ou seja, buscar por dois vetores de saída mais um terceiro bit na posição 2^0 para aproveitamento de um possível *carry in* no circuito de soma final.
- Verificar outras estratégias que priorizem a redução de área em oposição ao foco completo em atraso. Para atingir tal objetivo, explorar não somente outros tipos de compressores na etapa de compressão, como já citado, mas também estratégias diferenciadas de soma final, como por exemplo, eliminação das porções paralelas de hardware visando uma estratégia de compartilhamento de hardware, como descrito em (DUGDALE, 1992) e (PATEL *et al.*, 2007) .

REFERÊNCIAS

BAJARD, J.-C.; IMBERT, L. a full RNS implementation of RSA. **IEEE Transactions on Computers**, v. 53, n. 6, p. 769–774, 2004.

BAYOUMI, M.; JULLIEN, G.; MILLER, W. A VLSI array for computing the DFT based on RNS. *In*: ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing. [*S.l.*: *s.n.*], 1986. v. 11, p. 2147–2150.

BRENT, Richard P; KUNG, Hsiang T. A regular layout for parallel adders. **IEEE transactions on Computers**, IEEE Computer Society, v. 31, n. 03, p. 260–264, 1982.

CARDARILLI, G.C.; NANNARELLI, A.; RE, M. Reducing power dissipation in FIR filters using the residue number system. *In*: PROCEEDINGS of the 43rd IEEE Midwest Symposium on Circuits and Systems (Cat.No.CH37144). [*S.l.*: *s.n.*], 2000. v. 1, 320–323 vol.1.

DADDA, L. SOME SCHEMES FOR PARALLEL MULTIPLIERS. **Alta Frequenza**, v. 34, p. 349–356, 1965.

DUGDALE, M. VLSI implementation of residue adders based on binary adders. **IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing**, v. 39, n. 5, p. 325–329, 1992.

ELLEITHY, K.M.; BAYOUMI, M.A. A systolic architecture for modulo multiplication. **IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing**, v. 42, n. 11, p. 725–729, 1995.

ESPOSITO, Darjn; STROLLO, Antonio Giuseppe Maria; NAPOLI, Ettore; DE CARO, Davide; PETRA, Nicola. Approximate Multipliers Based on New Approximate Compressors. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 65, n. 12, p. 4169–4182, 2018.

FADAVI-ARDEKANI, J. MxN Booth encoded multiplier generator using optimized Wallace trees. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 1, n. 2, p. 120–125, 1993.

FATHI, Amir; MASHOUFI, Behbood; AZIZIAN, Sarkis. Very Fast, High-Performance 5-2 and 7-2 Compressors in CMOS Process for Rapid Parallel Accumulations. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 28, n. 6, p. 1403–1412, 2020.

FRITZ, Christopher; FAM, Adly T. Fast Binary Counters Based on Symmetric Stacking. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 25, n. 10, p. 2971–2975, 2017.

GHASEMZADEH, Mehdi; AKBARI, Amin; SOLTANI, Arefeh; HADIDI, Khayrollah. A new ultra high speed 7-2 compressor with a new structure. *In*: 2015 22nd International Conference Mixed Design of Integrated Circuits Systems (MIXDES). [*S.l.*: *s.n.*], 2015. p. 262–265.

GUFFIN, Ronald M. A Computer for Solving Linear Simultaneous Equations Using the Residue Number System. **IRE Transactions on Electronic Computers**, EC-11, n. 2, p. 164–173, 1962.

HAI, Han; JIANG, Xue-Qin; WEN, Miaowen; LEE, Moon Ho; JEONG, Yongchae. Efficient Transmission and Detection Based on RNS for Generalized Space Shift Keying. **IEEE Wireless Communications Letters**, v. 6, n. 4, p. 486–489, 2017.

HASAN, Mehedi; HOSSEIN, Md. Jobayer; HOSSAIN, Mainul; ZAMAN, Hasan; ISLAM, Sharnali. Design of a Scalable Low-Power 1-Bit Hybrid Full Adder for Fast Computation. **Circuits and Systems II: Express Briefs, IEEE Transactions on**, v. 67, p. 1464–1468, ago. 2020.

HIASAT, A.A. High-speed and reduced-area modular adder structures for RNS. **IEEE Transactions on Computers**, v. 51, n. 1, p. 84–89, 2002.

HIASAT, Ahmad. New memoryless, mod (2n+or-1) residue multiplier. **Electronics Letters**, v. 28, p. 314–315, mar. 1992.

JULLIEN. Implementation of Multiplication, Modulo a Prime Number, with Applications to Number Theoretic Transforms. **IEEE Transactions on Computers**, v. C-29, n. 10, p. 899–905, 1980.

KIM, Taewhan; JAO, W.; TJIANG, S. Circuit optimization using carry-save-adder cells. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 17, n. 10, p. 974–984, 1998.

KOYADA, Bhavani; MEGHANA, N.; JALEEL, Md. Omair; JERIPOTULA, Praneet Raj. A comparative study on adders. *In*: 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). [*S.l.*: *s.n.*], 2017. p. 2226–2230.

LOW, Jeremy Yung Shern; CHANG, Chip-Hong. A New Approach to the Design of Efficient Residue Generators for Arbitrary Moduli. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 60, n. 9, p. 2366–2374, 2013.

LYAKHOV, Pavel; VALUEVA, Maria; VALUEV, Georgii; NAGORNOV, Nikolai. High-Performance Digital Filtering on Truncated Multiply-Accumulate Units in the Residue Number System. **IEEE Access**, v. 8, p. 209181–209190, 2020.

MA, Weinan; LI, Shuguo. A new high compression compressor for large multiplier. *In*: 2008 9th International Conference on Solid-State and Integrated-Circuit Technology. [*S.l.*: *s.n.*], 2008. p. 1877–1880.

MADON, Bakri. A COMS High-Speed Array Multiplier Based on (2, 2, 3) Counters. **ASEAN Journal on Science and Technology for Development**, v. 10, n. 2, p. 67–80, 1993.

MEHER, Pramod Kumar; STOURAITIS, Thanos. RNS-Based Arithmetic Circuits and Applications. *In*: ARITHMETIC Circuits for DSP Applications. [*S.l.*: *s.n.*], 2017. p. 186–236.

MEHRABI, Mohamad Ali; DOCHE, Christophe; JOLFAEI, Alireza. Elliptic Curve Cryptography Point Multiplication Core for Hardware Security Module. **IEEE Transactions on Computers**, v. 69, n. 11, p. 1707–1718, 2020.

MERRILL, Roy D. Improving Digital Computer Performance Using Residue Number Theory. **IEEE Transactions on Electronic Computers**, EC-13, n. 2, p. 93–101, 1964.

MOHAN, Pemmaraju V. Ananda. RNS-To-Binary Converter for a New Three-Moduli Set $\{2^{n+1}-1, 2^n, 2^n-1\}$. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 54, n. 9, p. 775–779, 2007.

MOHAN, PV Ananda; MOHAN, PV Ananda. **Residue number systems**. [*S.l.*]: Springer, 2016.

OMONDI, Amos; PREMKUMAR, Benjamin. **Residue Number Systems: Theory and Implementation**. [*S.l.*: *s.n.*], set. 2007. ISBN 978-1-86094-866-4.

PALIOURAS, V.; STOURAITIS, T. Multifunction architectures for RNS processors. **IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing**, v. 46, n. 8, p. 1041–1054, 1999.

PALIOURAS, V. et al. A low-complexity combinatorial RNS multiplier. **IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing**, v. 48, n. 7, p. 675–683, 2001.

PALUDO, Rogério. **Implementações eficientes de conversores reversos e multiplicações por constante usando residue number systems**. 2020. Tese de Doutorado – Universidade Federal de Santa Catarina.

PARHAMI, Behrooz. **Computer arithmetic - algorithms and hardware designs**. [*S.l.*]: Oxford University Press, 2000. ISBN 978-0-19-512583-2.

PATEL, Riyaz A.; BENAISSA, Mohammed; POWELL, Neil; BOUSSAKTA, Said. Novel Power-Delay-Area-Efficient Approach to Generic Modular Addition. **IEEE**

Transactions on Circuits and Systems I: Regular Papers, v. 54, n. 6, p. 1279–1292, 2007.

PATRONIK, Piotr; PIESTRAK, Stanisław J. Design of residue generators with CLA/compressor trees and multi-bit EAC. *In*: 2017 IEEE 8th Latin American Symposium on Circuits Systems (LASCAS). [*S.l.*: *s.n.*], 2017. p. 1–4.

PATRONIK, Piotr; PIESTRAK, Stanisław J. Design of Reverse Converters for General RNS Moduli Sets $\{2^k, 2^n-1, 2^n+1, 2^{n+1}-1\}$ and $\{2^k, 2^n-1, 2^n+1, 2^{n-1}-1\}$ (n even). **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 61, n. 6, p. 1687–1700, 2014.

PATRONIK, Piotr; PIESTRAK, Stanisław J. Design of RNS Reverse Converters with Constant Shifting to Residue Datapath Channels. **J. Signal Process. Syst.**, Kluwer Academic Publishers, USA, v. 90, n. 3, p. 323–339, mar. 2018. ISSN 1939-8018.

PETTENGHI, Hector; CHAVES, Ricardo; SOUSA, Leonel. RNS Reverse Converters for Moduli Sets With Dynamic Ranges up to $(8n+1)$-bit. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 60, n. 6, p. 1487–1500, 2013.

PETTENGHI, Hector; MATOS, Roberto de; MOLAHOSSEINI, Amir. RNS reverse converters for moduli sets with dynamic ranges of 9n-bit. *In*: 2016 IEEE 7th Latin American Symposium on Circuits Systems (LASCAS). [*S.l.*: *s.n.*], 2016. p. 143–146.

PETTENGHI, Hector; PALUDO, Rogerio; MATOS, Roberto; LYAKHOV, Pavel A. Efficient RNS Reverse Converters for Moduli Sets with Dynamic Ranges Up to $(10n+1)$-Bit. **Circuits Syst. Signal Process.**, Birkhauser Boston Inc., USA, v. 37, n. 11, p. 5178–5196, nov. 2018. ISSN 0278-081X.

PIESTRAK, S.J. Design of high-speed residue-to-binary number system converter based on Chinese Remainder Theorem. *In*: PROCEEDINGS 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors. [*S.l.*: *s.n.*], 1994. p. 508–511.

RADHAKRISHNAN, D.; YUAN, Y. Novel approaches to the design of VLSI RNS multipliers. **IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing**, v. 39, n. 1, p. 52–57, 1992.

RASHID, M.; MUHTAROĞLU, A. A novel multiplier design for data rendering. *In*: 2017 IEEE Biomedical Circuits and Systems Conference (BioCAS). [*S.l.*: *s.n.*], 2017. p. 1–4.

SAYED, Ahmed; AL-ASAAD, Hussein. Survey and Evaluation of Low-Power Full-Adder Cells. *In*: ESA/VLSI. [*S.l.*: *s.n.*], jun. 2004. p. 332–338.

SONG, P.J.; DE MICHELI, G. Circuit and architecture trade-offs for high-speed multiplication. **IEEE Journal of Solid-State Circuits**, v. 26, n. 9, p. 1184–1198, 1991.

STELLING, P.F.; MARTEL, C.U.; OKLOBDZIJA, V.G.; RAVI, R. Optimal circuits for parallel multipliers. **IEEE Transactions on Computers**, v. 47, n. 3, p. 273–285, 1998.

STELLING P.F., Oklobdzija V.G. Design strategies for optimal hybrid final adders in a parallel multiplier. **J VLSI Sign Process Syst Sign Image Video Technol**, v. 14, p. 321–331, 1996.

SZABO, N.S.; TANAKA, R.I. **Residue Arithmetic and Its Applications to Computer Technology**. [*S.l.*]: McGraw-Hill, 1967. (McGraw-Hill series in information processing and computers).

TSAI, Soo-Hung Terence; ZHOU, Chang-hui. Taiwan's United Microelectronics Corporation (UMC). *In*: THE Silicon DragonHigh-Tech Industry in Taiwan. Cheltenham, UK: Edward Elgar Publishing, 2006. ISBN 9781840642407.

UM, Junhyung; KIM, Taewhan. Layout-aware synthesis of arithmetic circuits. *In*: PROCEEDINGS 2002 Design Automation Conference (IEEE Cat. No.02CH37324). [*S.l.*: *s.n.*], 2002. p. 207–212.

VERMA, Ajay K.; IENNE, Paolo. Automatic Synthesis of Compressor Trees: Reevaluating Large Counters. *In*: 2007 Design, Automation Test in Europe Conference Exhibition. [*S.l.*: *s.n.*], 2007. p. 1–6.

VERMA, Ajay K.; IENNE, Paolo. Improving XOR-Dominated Circuits by Exploiting Dependencies between Operands. *In*: 2007 Asia and South Pacific Design Automation Conference. [*S.l.*: *s.n.*], 2007. p. 601–608.

VILLEGER, D.; OKLOBDZIJA, V.G. Analysis of Booth encoding efficiency in parallel multipliers using compressors for reduction of partial products. *In*: PROCEEDINGS of 27th Asilomar Conference on Signals, Systems and Computers. [*S.l.*: *s.n.*], 1993. 781–784 vol.1.

WALLACE, C. S. A Suggestion for a Fast Multiplier. **IEEE Transactions on Electronic Computers**, EC-13, n. 1, p. 14–17, 1964.

WANG, Yuke. Residue-to-binary converters based on new Chinese remainder theorems. **IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing**, v. 47, n. 3, p. 197–205, 2000.

WEINBERGER, A. 4-2 carry-save adder module. **IBM Technical Disclosure Bulletin**, ; v. 23, n. 8, p. 3811–3814, 1981.

YANG, Lie-Liang; HANZO, L. A residue number system based parallel communication scheme using orthogonal signaling .I. System outline. **IEEE Transactions on Vehicular Technology**, v. 51, n. 6, p. 1534–1546, 2002.

YU, Zhan; YU, Meng-Lin; WILLSON, Alan N. Signal Representation Guided Synthesis Using Carry-Save Adders for Synchronous Data-Path Circuits. *In*: PROCEEDINGS of the 38th Annual Design Automation Conference. Las Vegas, Nevada, USA: Association for Computing Machinery, 2001. (DAC '01), p. 456–461.

ZIMMERMANN, R. Efficient VLSI implementation of modulo $2^n \pm 1$ addition and multiplication. *In*: PROCEEDINGS 14th IEEE Symposium on Computer Arithmetic (Cat. No.99CB36336). [*S.l.*: *s.n.*], 1999. p. 158–167.

ZIMMERMANN, Reto. VHDL library of arithmetic units. *In*: CITESEER. PROC. First Int. Forum on Design Languages (FDL'98), Lausanne, Switzerland. [*S.l.*: *s.n.*], 1998. p. 267–272.

APÊNDICE A – DESCRIÇÃO DA FERRAMENTA DE SOFTWARE PARA GERAÇÃO DOS CIRCUITOS RNS

A ferramenta foi construída utilizando a linguagem Python, versão 3.7.3. (Python Software Foundation, Wilmington, Delaware, Estados Unidos) e faz uso de várias bibliotecas que devem ser configuradas preliminarmente.

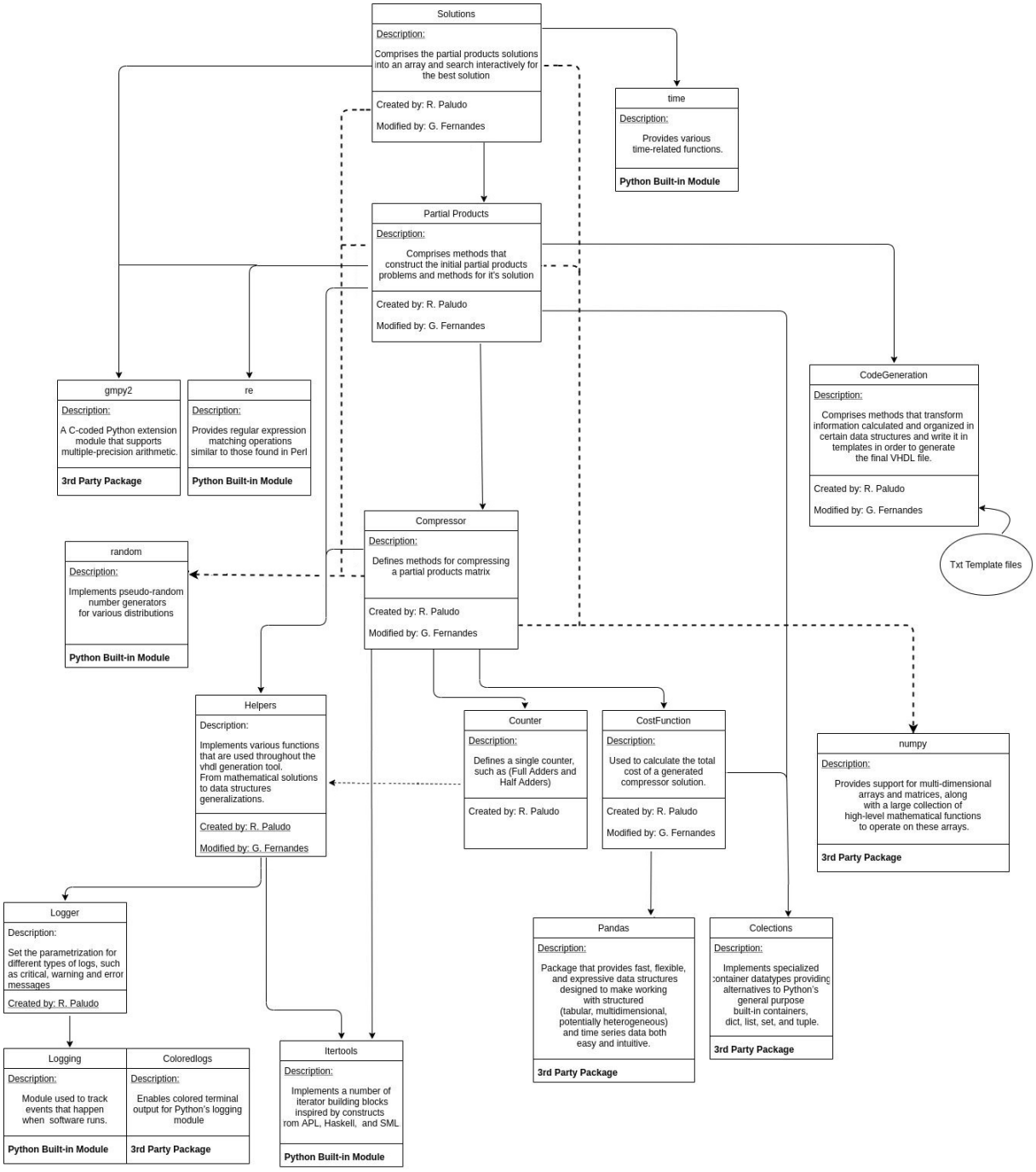
Observando a Figura A.1 na qual representa-se o diagrama de classes da ferramenta, é exposto o funcionamento de modo simplificado. A classe "Solutions"configura-se como ponto de partida para criação dos circuitos desejados. Para tal, existe a necessidade de informar os parâmetros de entrada, como por exemplo o conjunto modular, o número bits em cada canal modular, os compressores a serem utilizados para solução e o tipo de caso de compressão, seja ele uma multiplicação por constante ou multiplicação de variável por variável.

Ao realizar a descrição dos variados parâmetros de entrada ocorrerá o processamento dos sinais iniciais principalmente pelas heranças provindas de "Partial Products"e "Helpers". A classe "Partial Products"será responsável pela formação da representação inicial da matriz e sua compressão. Já a classe "Helpers"fornecerá um conjunto variado de métodos para garantir o fornecimento de representações, formatações e cálculos para auxiliar no funcionamento do programa. Um destes cálculos consiste na computação de todas a representações para os bits associados a um determinado valor binário. Ainda ligado à classe de formação das representações iniciais, encontra-se a classe "Compressor", na qual será definido todos os compressores a serem utilizados, bem como informações de compressão que realizam durante o processo iterativo.

O processo de resolução ocorrerá em um *loop*. A cada iteração serão avaliadas a solução atual e a anterior, para que seja escolhida a resolução mais eficiente até o momento. A classe responsável por esta avaliação é chamada de "CostFunction".

Ao fim de um número de simulações especificadas na entrada do algoritmo, ocorrerá a "tradução" de todos o valores calculados para a linguagem VHDL por meio de um template já disponível.

Figura A.1 – Diagrama de Classes da Ferramenta de Software Construída



Fonte – Elaborado pelo autor, 2021.