FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Caracterização de operadores modulares implementados em FPGA

Armando Filipe Carvalhido Fernandes

Programa de Mestrado Integrado em Engenharia Electrótécnica e de Computadores

Orientador: António José Duarte Araújo

Junho de 2010

Resumo

O sistema de numeração por resíduos (RNS) baseia-se na representação de um número através dos resíduos relativos a um conjunto de módulos. A aritmética com números assim representados designa-se por aritmética modular ou aritmética de resíduos, consistindo em operações aritméticas entre os respectivos resíduos. Estas operações são independentes umas das outras não havendo propagação de *carry* entre elas. Esta particularidade permite que operações comuns como a adição e a multiplicação possam ser realizadas em paralelo e por operadores de menor dimensão. O trabalho consistiu na implementação e caracterização de operadores modulares em FPGA usando como métricas os recursos ocupados e a latência de implementação. Foram exploradas particularidades das FPGAs tais como o paralelismo, os blocos multiplicadores dedicados e as ligações rápidas de propagação de *carry* no sentido de optimizar a implementação dos operadores modulares que incluía, além dos operadores RNS ou modulares, as necessárias conversões de binário para RNS e de RNS para binário. Foi feita a comparação dos operadores modulares com operadores convencionais e concluiu-se que, na FPGA disponível para teste, o sistema de numeração por resíduos traz poucas vantagens no que diz respeito à latência mas possibilita, para um número elevado de bits, a diminuição da área e do número de multiplicadores dedicados utilizados. Para uma aplicação envolvendo um elevado número de operações, para um elevado número de bits e mediante uma arquitectura *pipelined*, o RNS permite obter melhores resultados tanto na latência como no número de LUTs ocupados.

Abstract

The residue number system is based on a number representation through their residue related to a moduli set. The arithmetic with this number representation is called residue arithmetic. These operations do not depend on each other and do not have carry propagation from one to another. This feature allows common operations, like addition and multiplication, to be performed with smaller operators and by using parallelism. This work consisted on the residue operators implementation on a FPGA in order to measure the area and the latency in this reconfigurable device. Some FPGA specifics have been studied and explored such as dedicated multiplier blocks and carry chains in order to optimize, using parallelism, the residue operators implementation which include the conversions from binary to RNS as well as RNS to binary. A comparison between residue operators and conventional operators shows that, concerning latency, the residue number system is not advantageous. However, the RNS designs can reduce the area and the number of used dedicated multipliers for large bit-width numbers.

Agradecimentos

Várias pessoas deram a sua contribuição para que esta dissertação fosse realizada. Seja de forma directa ou indirecta, o apoio e a ajuda recebida foram, sem dúvida alguma, cruciais nesta recta final da minha vida académica.

Ao Professor António José Duarte Araújo, orientador da dissertação, agradeço o apoio e disponibilidade.

Aos colegas que se tornaram grandes amigos, André Leite, Tiago Mendes, Luís Cardoso e Francisco Basadre que me acompanharam ao longo destes anos académicos, nos bons e maus momentos, e em especial a Filipe Meireles que me acompanhou nesta dissertação. Obrigado pela preciosa ajuda.

Um obrigado especial a Rosa Milhazes pelo apoio incondicional.

Deixo também uma palavra de agradecimento á minha mãe Maria José Fernandes e á minha irmã Maria Celeste Fernandes.

Quero principalmente agradecer à pessoa que tornou possível a minha vida académica e a quem dedico esta dissertação mas que hoje já não faz parte deste mundo, ao meu pai Armando Fernandes.

A todos vós deixo aqui o meu agradecimento sincero.

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

mod	módulo
RNS	*Residue Number System*
FA	*Full Adder*
BK	*Brent-Kung*
MRS	*Mixed-Radix System*
FPGA	*Field Programmable Gate Array*
LUT	*Look up table*
CSA	*Carry Save Adder*
CPA	*Carry Propagate Adder*
EAC	*End Around Carry*
RCA	*Ripple Carry Adder*
CLA	*Carry Look-ahead Adder*
CRT	*Chinese Remainder Theorem*
ASIC	*Application Specific Integrated Circuit*
CLB	*Configurable Logic Block*

Capítulo 1

Introdução

O enorme avanço tecnológico obtido nas últimas décadas caracteriza-se pela proliferação de novidades no que diz respeito à tecnologia electrónica, de tal modo que se tornaram parte integrante do quotidiano humano, tornando-se um dos principais suportes de conhecimento que a sociedade actual possui. Estes avanços na tecnologia têm permitido o aparecimento de dispositivos cada vez mais sofisticados e rápidos mas também cada vez mais complexos. Estes aspectos estão fortemente relacionados com o crescente desempenho possibilitado pelos dispositivos micro-electrónicos e pela aritmética computacional.

No caso dos dispositivos micro-electrónicos, destacam-se os tradicionais ASICs (*Application Specific Integrated Circuits*) e mais recentemente as FPGAs (*Field Programmable Gate Array*). Os principais inconvenientes dos circuitos integrados do tipo ASICs são o seu elevado custo de projecto e fabrico e o extenso tempo necessário para a chegada desses circuitos ao mercado. Os circuitos FPGA, por sua vez, surgem como uma alternativa mais interessante pelas possibilidades que oferecem no desenvolvimento de sistemas digitais e por constituirem dispositivos genéricos onde o utilizador configura o circuito integrado para realizar uma função específica. À possibilidade de reconfiguração das FPGAs e ao contrário dos ASICs acresce ainda a facilidade de corrigir, modificar e estender as funcionalidades implementadas. O projecto de circuitos é geralmente baseado numa estrutura hierárquica de pequenos módulos interligados, permitindo melhor simulação, verificação e redução da complexidade, explorando possibilidades de paralelismo e formatos específicos dos dados.

No caso da aritmética computacional, os bits representando os números num computador podem ter diferentes significados. Podem ser positivos ou negativos e inteiros ou reais mas quanto menor for o número de bits a processar, mais rápido será a realização de uma determinada operação aritmética.

O sistema de numerção por resíduos, *Residue Number System* (RNS), é um sistema de numeração não posicional baseado no Teorema Chinês do Resto, *Chinese Remainder Theorem* (CRT), que permite explorar essa vantagem. Este teorema indica que é possível representar grandes in-

teiros utilizando um conjunto de pequenos inteiros, de forma que os cálculos possam ser realizados em paralelo daí resultando uma maior eficiência.

1.1 Sistema de numeração por resíduos

> ”Temos coisas das quais não sabemos o número
> se as contarmos por 3, o resto é 2,
> se as contarmos por 5, o resto é 3,
> se as contarmos por 7, o resto é 2,
> Quantas coisas temos?
> A resposta, 23.”

O filósofo chinês *Sun Tzu* escreveu este verso intitulado *t'ai-yen* (grande generalização) há mais de 1500 anos. Este verso descreve o CRT, usando como módulos a base $(3,5,7)$. Com base neste teorema, comprovado pelo matemático Euler em 1734, criaram-se os sistemas de numeração por resíduos (RNS) e desenvolveram-se unidades de cálculo aritmético que são hoje em dia utilizadas em aplicações diversas: no processamento de sinais, nomeadamente para cálculo de FFT, filtros digitais, mas também em criptografia para processamento aritmético como a adição, subtracção e multiplicação de números com um elevado número de bits.

O RNS é definido por um conjunto de números inteiros designados por módulos. Estes números devem ser todos co-primos [2], ou seja, um módulo não pode ser factor de nenhum outro módulo. O produto de todos os módulos define a gama dinâmica onde qualquer número inteiro (desde que esteja dentro da gama dinâmica) pode ser representado no sistema numérico por resíduos definido como um conjunto de números de menor dimensão representando a classe de resíduos do número inteiro relativo aos módulos escolhidos para a base. Usando este sistema de numeração, operações aritméticas como a adição e a multiplicação, por exemplo, podem ser realizadas em paralelo, pois as operações RNS ou modulares não possuem propagação de *carry*. Por exemplo, na realização da adição RNS entre dois operandos com uma gama dinâmica composta por uma determinada base, a operação produz-se num número de adições de menor dimensão igual ao número de módulos utilizados, podendo assim diminuir a latência e aumentando a rapidez do cálculo. No entanto, este sistema de numeração implica uma primeira conversão de ambos os números inteiros para RNS, realizada em paralelo, e uma conversão inversa após as adições modulares. A conversão de RNS para binário é complexa, mas só é necessária após a realização dos cálculos. Portanto, o seu custo pode ser amortizado na eventualidade de ser executada uma sequência de operações.

As metodologias e técnicas de projecto de circuitos têm um papel muito importante na implementação dos operadores modulares pois, sendo possível implementá-los de diversas formas e diferentes arquitecturas, pode significar a diminuição de área total e da latência. Dependendo

da tecnologia de implementação, a adição é a operação aritmética mais básica mas também das mais relevantes. A sua eficaz implementação pode trazer imensos benefícios à implementação de operações aritméticas mais complexas, tais como a multiplicação. Após a conversão para RNS, pode ser realizada uma sequência de cálculos, tais como um somatório ou um produtório, levando ao aumento dos benefícios introduzidos pela numeração por resíduos.

1.2 Motivação

A implementação de unidades aritméticas usando o sistema de numeração por resíduos é sobretudo encontrada recorrendo a soluções do tipo ASIC. A implementação baseada em FPGA é menos frequente levantando dúvidas sobre a eficácia da aritmética RNS, em termos de desempenho, quando realizada em FPGAs. O RNS mostra-se como uma opção interessante para o aumento do desempenho de operações e é muitas vezes invocado como sendo um sistema de maior rapidez de processamento devido à ausência de propagação de *carry* entre operações de menor dimensão e realizadas em paralelo. Mediante as particularidades que uma FPGA apresenta, tais como o paralelismo, as ligações rápidas de propagação de *carry* e a utilização de multiplicadores dedicados que aceleram a realização de cálculos aritméticos, o contributo do sistema de numeração por resíduos é certamente diferente em FPGA. Neste sentido, a principal motivação para a realização desta dissertação reside em investigar as vantagens na implementação de um sistema RNS em FPGAs.

1.3 Estrutura da dissertação

Para além desta introdução, esta dissertação é constituída por mais quatro capítulos. No capítulo 2, são apresentados os blocos do sistema de numeração por resíduos e as operações que neles são envolvidas. É ainda apresentado uma arquitectura geral de um sistema de numeração por resíduos. O capítulo 3 apresenta as principais arquitecturas de somadores utilizados na construção dos módulos RNS e é feita uma breve apresentação da estrutura dos dispositivos FPGA assim como a descrição da implementação das várias etapas dos operadores aritméticos no sistema RNS. No capítulo 4 são apresentados e analisados os resultados das diversas implementações realizadas e é feita a comparação dos operadores RNS com operadores aritméticos (convencionais) de inteiros. No capítulo 5 são apresentadas as conclusões deste trabalho assim como os possíveis trabalhos futuros que podem decorrer desta dissertação.

Capítulo 2

Aritmética modular

O RNS consiste na representação de um dado número X em resíduos $x_1, x_2, x_3, \ldots, x_n$ obtidos pelos respectivos módulos m_i representando os sucessivos restos da divisão de X/m_i, ou seja, $x_i = X \bmod m_i$ representando a classe de resíduos de X relativo aos módulos escolhidos para a base.

Usando este sistema, operações aritméticas como a adição e a multiplicação, por exemplo, podem ser realizadas em paralelo, pois operações RNS não possuem propagação de *carry*. Por exemplo, para calcular a adição entre A e B em RNS com uma base composta por $m_i (0 < i \leq K)$, a operação é processada através de K adições de menor dimensões $Add_i = (a_i + b_i) \bmod m_i$, $0 < i \leq K$ diminuindo a latência e aumentando a rapidez do cálculo. No entanto, este sistema de numeração implica uma primeira conversão de ambos os inteiros A e B para RNS que pode ser realizada em paralelo e uma segunda conversão inversa após as K adições.

Neste capítulo são apresentados várias operações necessárias para a realização de operadores aritméticos baseados nos sistemas de numeração por resíduos (RNS). A acompanhar esta apresentação, são mencionados exemplos de referência para uma melhor compreensão. A aritmética modular apresentada neste capítulo foi baseada em [3].

2.1 Representação RNS

Para a implementação de operações aritméticas de um número de bits considerável, a conversão dos números binários para resíduos de menor dimensão pode aumentar consideravelmente a velocidade de determinadas operações aritméticas. De forma a entender a representação RNS, retomamos o verso de *Sun Tzu* da secção 1.1. Este *puzzle* pede, basicamente, para converter a representação numérica de resíduos $(2|3|2)$, baseados nos módulos $(7|5|3)$, para decimal. Um sistema de numeração por resíduos é a representação de um número x pelos seus n resíduos constantes $x_0, x_1, x_2, \ldots, x_{n-1}$ definidos pelos seus n módulos co-primos $m_0, m_1, m_2, \ldots, m_{n-1}$ onde $m_{n-1} > \ldots > m_1 > m_0$.

O produto, M, de todos os módulos, m_i, é o número dos valores representáveis do sistema de numeração por resíduos definidos pelos módulos m e é conhecido como a gama dinâmica.

$$M = m_{n-1} \times \ldots \times m_1 \times m_0 \tag{2.1}$$

O resíduo x_i de um inteiro x com o seu respectivo módulo m_i pode ser representado no sistema numérico por resíduos desde que x seja menor que M. Desta forma:

$$x_i = x \; mod \; m_i = \langle x \rangle_{m_i} \tag{2.2}$$

A representação RNS específica de x pode ser feita de forma a visualizar tanto os resíduos como os módulos:

$$x = (x_1|x_2|x_3)_{RNS(m_1|m_2|m_3)} \tag{2.3}$$

No verso do capítulo 1, por exemplo,

$$x = (2|3|2)_{RNS(7|5|3)}$$

e a gama dinâmica é de

$$M = 7 \times 5 \times 3 = 105.$$

105 é o número total de diferentes valores disponíveis com os três módulos 7, 5 e 3. Desta forma, a gama de valores pode representar números de 0 até 104, -52 até 52 ou qualquer outro intervalo de 105 inteiros consecutivos.

A representação RNS negativa pode ser encontrada complementando cada dígito de x_i de acordo com o seu módulo m_i não havendo alteração para os dígitos nulos. Se, por exemplo,

$$21 = (5|0|1|0)_{RNS(8|7|5|3)},$$

$$-21 = (8-5|0|5-1|0)_{RNS} = (3|0|4|0)_{RNS}$$

Cada sistema de numeração residual têm um peso associado a cada uma das suas posições, M/m_i. Retomando o exemplo do verso da secção 1.1 do capítulo 1, RNS$(7|5|3)$ têm os pesos associados a cada uma das 3 posições

$$15 \quad 21 \quad 35$$

Na prática, cada resíduo deve estar representado em binário para o processamento digital das operações aritméticas. Por exemplo, para RNS$(7|5|3)$ tal representação requer $3+3+2=8$bits.

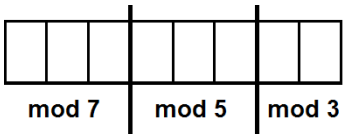

Figura 2.1: RNS(7|5|3) em BCD

No entanto, com uma gama dinâmica $M = 105$, 7 bits eram suficientes para a representação binária. A eficiência da representação binária é

$$\frac{105}{2^8} = 41\% \tag{2.4}$$

Apresentam-se a seguir os algoritmos a ter em conta para o processamento dos algoritmos aritmético. Os operandos de entrada têm representação binária ou decimal pois provém de máquinas ou de interfaces humanas e desta forma as saídas, após o processamento, devem manter a mesma representação. Será então necessário converter as entradas em representação binária/decimal para a representação RNS e vice-versa.

2.2 Conversão de binário para RNS

Dado um número y sem sinal,

$$y = \sum_{i=0}^{k-1} y_i \times 2^i, \tag{2.5}$$

encontrar os resíduos com os módulos m_i, $0 \leq i \leq k-1$, pode ser realizado através da seguinte equação da fonte [3]:

$$\langle (y_{k-1} \ldots y_1 y_0)_2 \rangle_{m_i} = \left\langle \left\langle 2^{k-1} y_{k-1} \right\rangle_{m_i} + \ldots + \langle 2y_1 \rangle_{m_i} + \langle y_0 \rangle_{m_i} \right\rangle_{m_i} \tag{2.6}$$

Segue-se um exemplo simplicado, de forma a obter uma melhor percepção na aritmética envolvida nesta operação, de como se poderia converter os números binário para RNS:

Se se definir e alocar para cada i e j o valor de $\left\langle 2^j \right\rangle_{m_i}$, o resíduo x_i de y *mod* m_i pode ser directamente calculado com a soma de constantes previamente definidas numa tabela (*look-up table*). A tabela dos resíduos das 10 primeiras potências de 2 mostra as constantes necessárias para a conversão de um número binário que abrange uma gama dinâmica superior ao do exemplo

j	2^j	$\langle 2^j \rangle_7$	$\langle 2^j \rangle_5$	$\langle 2^j \rangle_3$
0	1	1	1	1
1	2	2	2	2
2	4	4	4	1
3	8	1	3	2
4	16	2	1	1
5	32	4	2	2
6	64	1	4	1
7	128	2	3	2
8	256	4	1	1
9	512	1	2	2

Tabela 2.1: Tabela dos resíduos das 10 primeiras potências de 2

do verso da secção 1.1 (M= 105) para RNS(7|5|3) onde:

$$y = 23_{10} = 10111_2 = 2^4 + 2^2 + 2^1 + 2^0$$

Consultando a tabela, com $j = 4,2,1,0$ os módulos serão calculados da seguinte forma:

$$y_0 = \langle y \rangle_3 = \langle 1+1+2+1 \rangle_3 = 2$$

$$y_1 = \langle y \rangle_5 = \langle 1+4+2+1 \rangle_5 = 3$$

$$y_2 = \langle y \rangle_7 = \langle 2+4+2+1 \rangle_7 = 2$$

A representação RNS(7|5|3) de 23_{10} é, como já sabíamos, $(2|3|2)_{RNS}$. No pior dos casos serão necessários k adições para o cálculo de cada resíduo.

Associado a qualquer sistema de numeração por resíduos está o chamado sistema de numeração *mixed-radix* que representa um sistema de posição com o seu respectivo peso. Assim, para RNS$(m_{k-1}|\cdots|m_2|m_1|m_0)$ temos MRS$(m_{k-1}|\cdots|m_2|m_1|m_0)$ tal que

$$(m_{k-2}\cdots m_2 m_1 m_0) \;\; \cdots \;\; (m_2 m_1 m_0) \;\; (m_1 m_0) \;\; (m_0) \;\; (1) \tag{2.7}$$

são os pesos associados a cada k dígito do sistema de numeração posicional com as gamas de valores $[0, m_{k-1}-1], \cdots, [0, m_2-1], [0, m_1-1], [0, m_0-1]$. Para a determinação do dígito z_i da representação MRS, dado x_i da representação RNS, temos:

$$y = (x_{k-1}|\cdots|x_2|x_1|x_0)_{RNS} = (z_{k-1}|\cdots|z_2|z_1|z_0)_{MRS} \tag{2.8}$$

ou seja,

$$y = z_{k-1}(m_{k-2}\cdots m_2 m_1 m_0) + \cdots + z_2(m_1 m_0) + z_1(m_0) + z0$$

Claramente se verifica que $z_0 = x_0$, subtraindo x_0 em ambas as representações RNS e MRS, temos

$$y - x_0 = (x'_{k-1}|\cdots|x'_2|x'_1|0)_{RNS} = (z_{k-1}|\cdots|z_2|z_1|0)_{MRS}$$

onde $x'_j = \langle x_j - x_0 \rangle_{m_j}$. Dividindo, de seguida, ambas as representações RNS e MRS por m_0 obtemos:

$$(x''_{k-1}|\cdots|x''_2|x''_1|0)_{RNS} = (z_{k-1}|\cdots|z_2|z_1)_{MRS} \tag{2.9}$$

O resultado da divisão é exacta uma vez que o resto era x_0 e foi removido anteriormente na subtracção, poder-se-à dizer que se trata de uma multiplicação pelo inverso do módulo $\langle 1/m_0 \rangle_{m_0}$. Do resultado da divisão de $y' = (x'_{k-1}|\cdots|x'_2|x'_1|0)_{RNS}$ por m_0 se obtém $(x''_{k-1}|\cdots|x''_2|x''_1|0)_{RNS}$ e da mesma forma z_1 voltando a uma equação similar ao problema inicial. Com z_1 já calculado, repete-se o mesmo processo obtendo-se z_2 e assim sucessivamente. Trata-se, portanto, de um algoritmo iterativo que deveria ser repetido um número de vezes igual ao número total de dígitos z_i. A multiplicação pelo inverso do módulo poderia ser definida e alocada em tabela de forma a facilitar o processamento de conversão de RNS para MRS. Se tivermos a representação MRS de y, a conversão para o sistema de posição *standard* é facilmente obtida pelo seguinte calculo:

$$y = z_0 + \sum_{i=0}^{n-1} (\prod_{j=0}^{i-1} m_i) z_i \tag{2.10}$$

A representação MRS é um processo iterativo que permite comparar as magnitudes de dois números na representação RNS ou ainda detectar o sinal de um número.

2.3 Conversão de RNS para binário

Um dos métodos para converter um número na representação RNS para binário é, como já foi referido, converter inicialmente o número RNS para a representação MRS e usar os pesos das posições obtidas para completar a conversão. Outro método consiste na determinação dos

pesos das posições da representação RNS pelo CRT (*Chinese remainder theorem*) que a seguir se descreve:

$$x = (x_{k-1}|\cdots|x_2|x_1|x_0)_{RNS} = \left\langle \sum_{i=0}^{k-1} M_i \left\langle \alpha_i x_i \right\rangle_{m_i} \right\rangle_M \tag{2.11}$$

onde, por definição, $M_i = x/m_i$ e $\alpha_i = \left\langle M_i^{-1} \right\rangle_{m_i}$ representa a multiplicação do inverso do módulo de M_i. Os primeiros conversores de RNS para binário baseavam-se no CRT. No entanto, este método não pode ser implementado em hardware de uma forma eficaz. Relembrando o exemplo do verso referido na secção 1.1, se $y = (2|3|2)_{RNS(7|5|3)}$ e baseando-se nas propriedades do sistema de numeração por resíduos, pode-se escrever:

$$(2|3|2)_{RNS} = (2|0|0)_{RNS} + (0|3|0)_{RNS} + (0|0|2)_{RNS}$$

$$= 2 \times (1|0|0)_{RNS} + 3 \times (0|1|0)_{RNS} + 2 \times (0|0|1)_{RNS}$$

Assim, se se souber os pesos das posições RNS, será possível converter qualquer número pelos módulos RNS(7|5|3). Para tal, é necessário efectuar multiplicações ou pode-se simplesmente consultar uma tabela de tamanho total igual a $\sum_{i=0}^{k-1} m_i$ com os valores de $\left\langle M_i \left\langle \alpha_i x_i \right\rangle_{m_i} \right\rangle_M$ para todos os possíveis i e x_i. A tabela 2.2, por exemplo, mostra-nos todos os valores dos pesos possíveis para RNS$(7|5|3)$:

Desta forma, consultando a tabela, tem-se:

$$(1|0|0)_{RNS} = 120_{10};$$

$$(0|1|0)_{RNS} = 336_{10};$$

$$(0|0|1)_{RNS} = 280_{10}.$$

Para o passo final da conversão apenas é necessário fazer algumas operações aritméticas:

$$(2|3|2)_{RNS} = \langle (2 \times 120) + (3 \times 336) + (2 \times 280) \rangle_{105}$$

$$= \langle 1808 \rangle_{105} = 23$$

i	m_i	x_i	$\langle M_i \langle \alpha_i x_i \rangle_{m_i} \rangle_M$
2	7	0	0
		1	120
		2	240
		3	360
		4	480
		5	600
		6	720
1	5	0	0
		1	336
		2	672
		3	168
		4	504
0	3	0	0
		1	280
		2	560

Tabela 2.2: Valores dos pesos possíveis para RNS$(7|5|3)$

2.4 Operadores modulares

Apresentadas as operações necessárias para as conversões de binário para RNS e de RNS para binário chega a vez da apresentação dos operadores modulares de adição e multiplicação que tem como entrada os resíduos provenientes da conversão para RNS dos operandos binários e como saída os resíduos do resultado da operação aritmética em causa para (a jusante) passar para a conversão final de RNS para binário. Essas operações aritméticas de resíduos podem ser feitas de forma independente e em paralelo uma vez que não existe propagação de *carry* entre operações de diferentes resíduos de acordo com os respectivos módulos podendo diminuir o tempo de processamento de cálculo.

É possível implementar vários tipo de operações tais como a adição, a multiplicação, a subtracção, a comparação de magnitude entre outros. No entanto, a adição é de maior importância por estar presente em praticamente todos os operadores aritméticos como por exemplo a multiplicação que também é analisada nesta dissertação. A subtracção é realizada através de adições em complemento para dois, a divisão é de maior complexidade e dificilmente implementada para sistemas de numeração por resíduos.

Nesta dissertação foram analisados os operadores de adição e multiplicação devido à maior importância que estes assumem. Seja, então, x_i o resíduo de x pelo módulo m_i conforme mostra a equação **??** e y_i o resíduo de y pelo mesmo módulo mi. Se se pretender calcular $x+y$,

$$(x+y)\ mod\ m_i\ =\ (x_i\ mod\ m_i)\ +\ (y_i\ mod\ m_i)$$

ou de acordo com a equação 2.3:

$$(x+y)\ mod\ m_i\ =\ (x_1|x_2|x_3)_{RNS(m_1|m_2|m_3)}\ +\ (y_1|y_2|y_3)_{RNS(m_1|m_2|m_3)},$$

é equivalente a ter:

$$(x+y)\ mod\ m_i\ =\ (x_i\ +\ y_i)\ mod\ m_i$$

Se, com o exemplo da secção 1.1, for realizado uma soma $23\ +\ 5$ e já sabendo os resíduos do primeiro operando teríamos:

$$5\ =\ (5|0|2)_{RNS(7|5|3)}$$

ou seja,

$$23\ +\ 5\ mod\ m_i\ =\ ((2+5)\ mod\ 7|(3+0)\ mod\ 5|(2+2)\ mod\ 3)_{RNS(7|5|3)}$$

$$=\ (0|3|1)_{RNS(7|5|3)}$$

Tanto na adição como na multiplicação, o procedimento a seguir para as operações são semelhantes diferindo apenas no operador aritmético.

2.5 Estrutura de operadores aritméticos num sistema de numeração por resíduos

A figura 2.2 ilustra as diferentes etapas para a implementação de um sistema de numeração por resíduos. O RNS é divido pelas três etapas de conversão, operação modular e conversão final podendo no entanto realizar mais operações modulares entre as conversões conforme mostra a figura 2.3.

Convém realçar que sendo necessário realizar operações de adição ou multiplicação em sequência, não haveria necessidade de se converter o resultado para RNS após cada operação modular. No entanto, corria-se o risco de se obter resultados finais errados pois se, na sequência de operações modulares, ocorrer *overflow*, ou seja, se o resultado de uma das operações modulares for maior que o valor máximo permitido pela gama dinâmica dos módulos escolhidos (ver equação 2.1), então esse resultado seria errado e não recuperável. Para evitar o *overflow*, a cada operação modular está associada uma redução modular ou conversão para RNS de acordo com os mesmos módulos utilizados na conversão inicial dos operandos para RNS e na conversão final do resultado em RNS para binário. Este é talvez o maior inconveniente do sistema de numeração por resíduos pois resulta no aumento do tempo total de processamento assim como no aumento da área total do circuito.

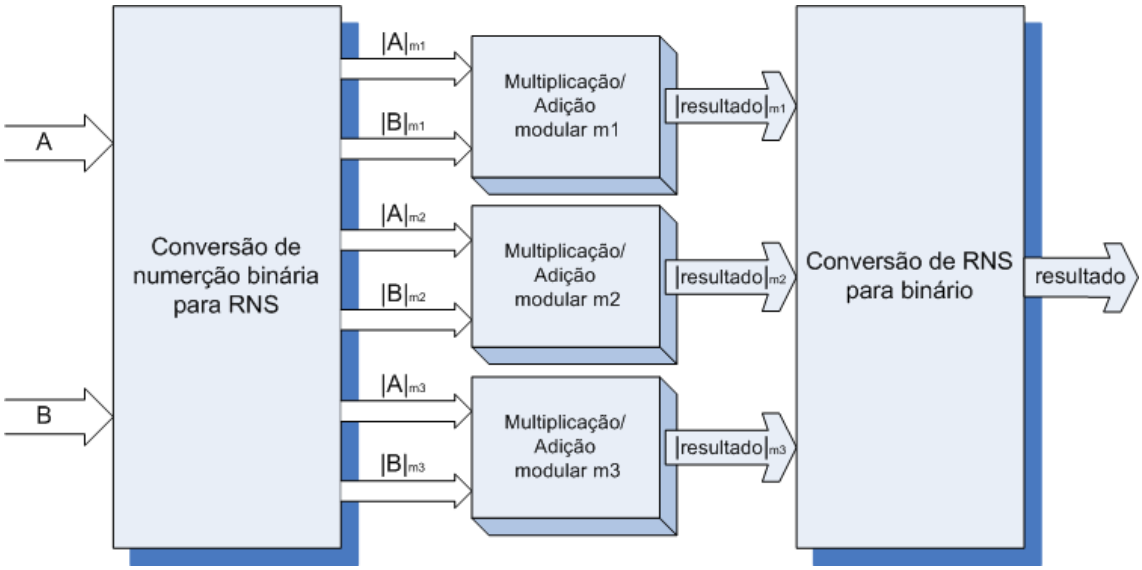

Figura 2.2: Diagrama de blocos de um operador modular

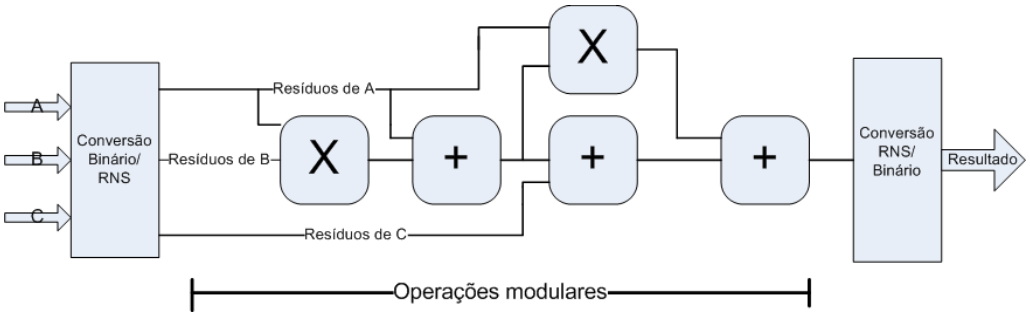

Figura 2.3: Sequência de operações aritméticas por resíduos

Uma boa forma de compensar este inconveniente é implementar um circuito utilizando a técnica de *pipelining* permitindo reduzir o atraso do circuito repartindo o caminho crítico através de registos sincronizados com um sinal de relógio comum. O caminho crítico será inferior ao original mas, na prática, a inserção de um registo de pipeline tem como consequência o atraso de um período de *clock* na geração de algum sinal ou evento. No entanto, este facto é desprezável ou irrelevante na grande maioria das aplicações uma vez que, num circuito *pipelined*, outros valores de entrada podem iniciar a operação do circuito sem ter que esperar que os valores anteriores de entrada cheguem à saída do circuito.

O esquema mostrado na figura 2.4 mostra de que forma se poderia dividir o circuito com os registos síncronos mas, dependendo do desempenho de cada etapa do circuito, este esquema poderia sofrer alterações no que diz respeito ao número de registos colocados e ao posicionamento deles. Este tema será abordado no próximo capítulo com mais detalhe.

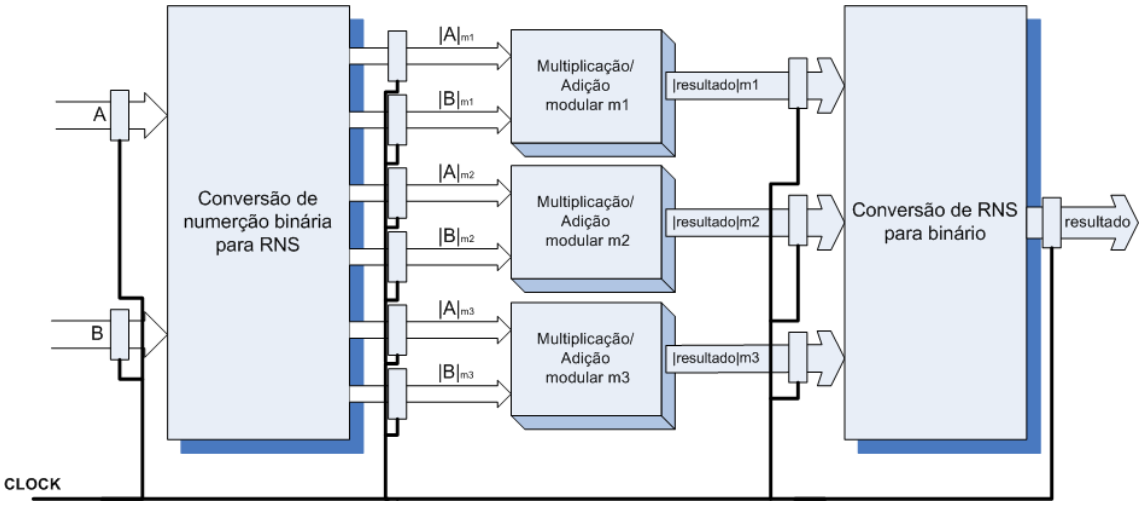

Figura 2.4: Estrutura de um operador RNS *pipelined*

Esta técnica pode levar a um mais alto nível de desempenho do circuito mas aumenta a área deste último.

Capítulo 3

Implementação

A aritmética modular permite aumentar a rapidez de cálculo em muitas aplicações. Ela é utilizada em diferentes contextos tais como o processamento de sinal, o tratamento algorítmico, a criptografia (entre outros). Esta dissertação aborda a aritmética modular num contexto diferente dos mencionados anteriormente, pretendendo caracterizar a implementação de operadores modulares em FPGA usando como métricas a área, ou recursos ocupados, e a latência de implementação. A área é expressa pelo número de LUTs total utilizado pelo circuito. A latência, por sua vez, representa o atraso combinacional do circuito. Numa arquitectura *pipelined* a latência deixa de ser o atraso combinacional passando a representar o período de um ciclo de relógio.

O capítulo inicia-se com a apresentação de algumas arquitecturas de somadores utilizados para a implementação dos vários blocos do RNS. São feitas algumas considerações sobre circuito do tipo FPGA com o objectivo de entender particularidades que posteriormente servirão para uma melhor compreensão dos resultados obtidos. São ainda apresentadas neste capítulo as diferentes implementações para os diversos blocos de operadores RNS, nomeadamente para a multiplicação e para a adição modulares, consideradas as mais importante no contexto desta dissertação devido à grande utilização da adição. Estas operações aritméticas foram implementadas explorando particularidades das FPGAs no sentido de optimizar os resultados obtidos, em termos de área e latência, dos operadores modulares mas também das necessárias conversões de binário para RNS e de RNS para binário.

3.1 Arquitectura de somadores

Todas as operações complexas de um computador digital são realizadas através de combinações de simples operações aritméticas e lógicas básicas tais como somar, complementar, comparar ou mover bits. Estas operações são fisicamente realizadas por circuitos electrónicos compostos por portas lógicas.

Os circuitos lógicos dos sistemas digitais podem ser do tipo combinacionais ou sequenciais. Um circuito combinacional é constituído por um conjunto de portas lógicas que determinam os valores das saídas directamente a partir dos valores actuais das entradas. Poder-se à dizer que um circuito combinacional realiza uma operação de processamento de informação calculada por um conjunto de equações booleanas de acordo com os valores de entrada deixando à saída o resultado da operação. Um circuito combinacional aritmético implementa operações aritméticas como a adição, subtracção, multiplicação e a divisão com números binários. De seguida são apresentados os circuitos combinacionais aritméticos considerados na realização deste trabalho.

3.1.1 *Ripple Carry Adder*

O *ripple carry adder* (RCA) é o somador de estrutura mais básica pois é simplesmente composto por *n full Adders* (FA) interligados em série. Com um RCA de *n* bits pode-se realizar um somador capaz de operar dois números binários de *n* bits. Particularmente, o dígito de ordem *n* do resultado, S_n, será obtido pela adição de a_n, b_n e C_{n-1}, onde este último representa o transporte proveniente do dígito anterior. O somador de índice *n* recebe como entradas a_n, b_n e C_{n-1}, gerando a soma S_n e o valor de transporte C_n, o qual será entrada para o FA do dígito seguinte (n+1).

A figura 3.1 ilustra o diagrama de blocos de um *ripple carry adder* de *n* bits.

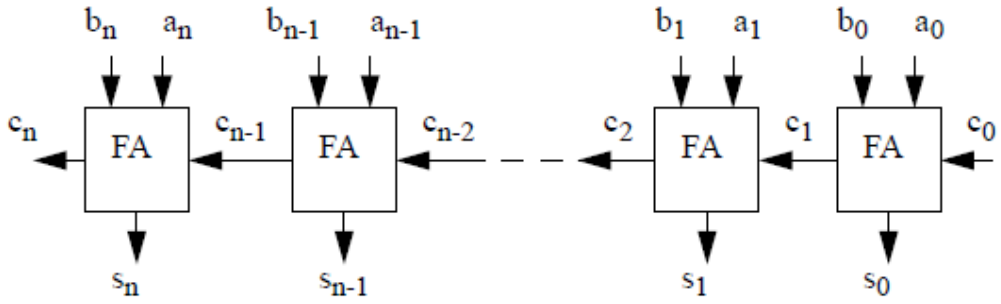

Figura 3.1: Diagrama de blocos de um RCA de *n* bits

Os somadores do tipo RCA são mais lentos à medida que aumenta o número de bits dos operandos uma vez que a propagação do *carry* passará por todos os *full adders* no caminho crítico e admitindo que cada *full adder* impõe um atraso, o tempo necessário para ser feita a soma será proporcional a *n*. Como o somador de entrada não deve receber nenhum bit de *carry in*, é necessário forçar que o primeiro *carry* seja zero. É de realçar que todos os circuitos são alimentados ao mesmo tempo mas o seu funcionamento é sequencial uma vez que C_n, sendo a entrada de um circuito, é um resultado do processamento de um circuito anterior (*Cout* do anterior).

3.1.2 *Carry Save Adder*

Existem muito casos onde é necessário somar mais do que dois números. A forma mais simples de somar m números de n bits é de adicionar os dois primeiros e somar o próximo ao resultado da primeira soma e assim sucessivamente. No entanto, este tipo de adição requer $m-1$ adições para um total de portas lógicas bem maior que o que é possível obter com outros somadores. Para reduzir a área e o tempo de propagação do caminho crítico poderá-se optar por implementar árvores de somadores e utilizando somadores do tipo *carry save adder* permitirá reduzir ainda mais o tempo de propagação.

Um *carry save adder* consiste em adicionar 3 operandos, x, y e z e convertê-los em dois números $C + S$ tal que

$$x + y + z = C + S$$

Esta operação não depende da propagação de *carries* de circuitos anteriores e é por isso uma enorme vantagem. O esquema da figura 3.2 representa o bloco CSA e como, através de um simples FA, é obtido.

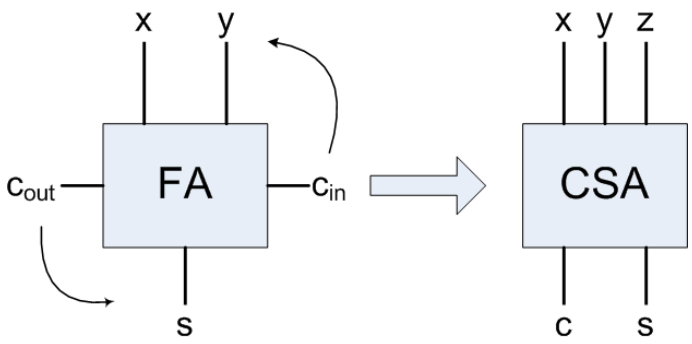

Figura 3.2: Diagrama de blocos de um *Carry Save Adder*

Na figura 3.3 é ilustrado um diagrama de blocos CSA que realiza a soma de cada bits dos três operandos de 5 bits obtendo como resultado dois novos números. Esses dois últimos conservem o valor numérico dos três primeiros.

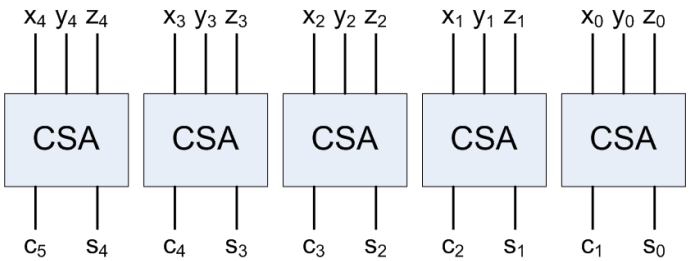

Figura 3.3: Diagrama de blocos de um *Carry Save Adder* de 5 bits

Apesar das vantagens deste tipo de somador, ele não resolve o problema de obtenção de uma única saída. O ponto mais relevante do CSA reside no facto das duas saídas C e S poderem ser calculadas de forma independentes entre elas e entre cada C's e S's.

3.1.3 *Carry Look-ahead Adder*

A principal função do somador *carry look-ahead* (CLA) é tentar gerar em paralelo todos os *carries* que estão para chegar e evitar a espera da propagação do *carry* correcto dos *full adders* de onde são gerados. Têm como objectivo aumentar a velocidade de propagação do *carry* em relação aos somadores RCA.

Os *carries*, C_{i+1}, gerados em i são dados por:

$$C_{i+1} = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_i \; ; \tag{3.1}$$

Desta equação entende-se que existe *carry* se este for gerado neste mesmo estado ou se chegou da propagação de *carry* de um estado anterior. Por outras palavras, o *carry* é gerado se ambos A_i e B_i forem iguais a "1" e propaga se um dos operandos é nulo e outro igual a "1". A função G_i indica, quando verdadeira, que o i-ésimo estado gerou um *carry* e, por consequência, do ponto de vista da propagação do *carry*, o *carry* de saída do estado anterior não precisará ser considerado. P_i, por sua vez, indica que o i-ésimo estado propaga o *carry* que lhe chega sendo ele "1"ou "0".

$$G_i = A_i \cdot B_i \; , \tag{3.2}$$

$$P_i = A_i \oplus B_i \; , \tag{3.3}$$

Substituindo P_i e G_i na equação de geração do *carry* obtêm-se:

$$C_{i+1} = G_i + P_i \cdot C_i \tag{3.4}$$

O somador CLA baseia-se em determinar todos os estados de entrada do somador e, simultaneamente, produzir os *carries* apropriados para cada um destes estados, ou seja, todos os *carries* são calculados em paralelo. Isto leva a que as referências aos *carries* anteriores na equação acima precisam de ser eliminadas. Deste modo, para um somador CLA de 4 bits, por exemplo, as equações de propagação do *carry* seriam:

$$C_0 = C_{in} \; ,$$

$$C_1 = G_0 + P_0 \cdot C_{in},$$

$$C_2 = G_1 + G_0 \cdot P_1 + P_0 \cdot P_1 \cdot C_{in},$$

$$C_3 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + P_0 \cdot P_1 \cdot P_2 \cdot C_{in},$$

$$C_4 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + P_0 \cdot P_1 \cdot P_2 \cdot P_3 \cdot C_{in},$$

Ao observar as equações de um somador *carry look-ahead*, fica claro que a sua implementação em hardware utiliza uma maior quantidade de recursos que uma implementação de um somador do tipo *ripple carry* por exemplo.

3.1.4 *Brent Kung*

A determinação do sinal de *carry* é feita através de blocos de 4 bits no CLA. Contudo, pode ser simplificada para apenas 2 bits introduzindo uma célula Brent Kung. Isto torna o circuito muito mais flexível possibilitando uma estrutura de blocos implementada em paralelo e em árvore. O somador é de alta importância no âmbito desta dissertação pois é um dos blocos mais utilizados e é nestes que reside grande parte dos resultados finais no que toca à rapidez com que os módulos serão processados. Para um reduzido número de bits, somadores convencionais seriam suficientes para obter bons resultados mas à medida que o número de bits cresce, o caminho crítico passa a ter cada vez mais importância no desempenho desses módulos. Como tal, é necessário pensar em outros tipo de somadores que permitem o paralelismo, tais como os somadores paralelos de prefixo em árvore multi-nível.

O somador *Brent-Kung* permite tal arquitectura e é considerada uma das melhores células para árvores de somadores de prefixo pois permite reduzir significativamente as pistas entre células.

A célula BK (*Brent-Kung*) permite calcular o *carry* de dois blocos de células FA. Segue-se, para relembrar, as funções que dependem unicamente das entradas A_i e B_i de dois operandos A e B a somar.

Propagate:

$$P_i = A_i \oplus B_i \,; \tag{3.5}$$

e *Generate*:

$$G_i = A_i \cdot B_i \,; \tag{3.6}$$

Uma célula *Brent-Kung*, por sua vez, representando uma soma de dois bits tem o esquema da figura 3.4 e as seguintes equações:

$$P_o = P_1 \cdot P_2 \,; \tag{3.7}$$

$$G_o = G_1 + (P_1 \cdot G_2) \,; \tag{3.8}$$

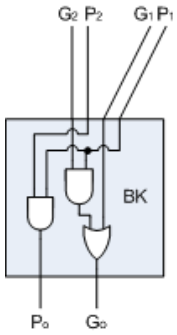

Figura 3.4: Bloco *Brent-Kung*

O somador possibilita uma estrutura de blocos implementada em paralelo e em árvore como se pode verificar na figura 3.5.

A figura permite visualizar de que forma é efectuado a adição de entradas de 16 bits, o número de células necessário para o cálculo e o paralelismo possibiltado pelo tipo de somador. O cálculo é efectuado em sete níveis de células BK e com um número relativamente reduzido de células.

Neste somador é possível ainda aumentar a rapidez alterando partes do circuito de forma a eliminar um nível de células BK mas isso só seria possível aumentando o número de células. A estrutura em árvore binária paralela permite obter um atraso no somador proporcional ao logaritmo do número de bits dos operandos.

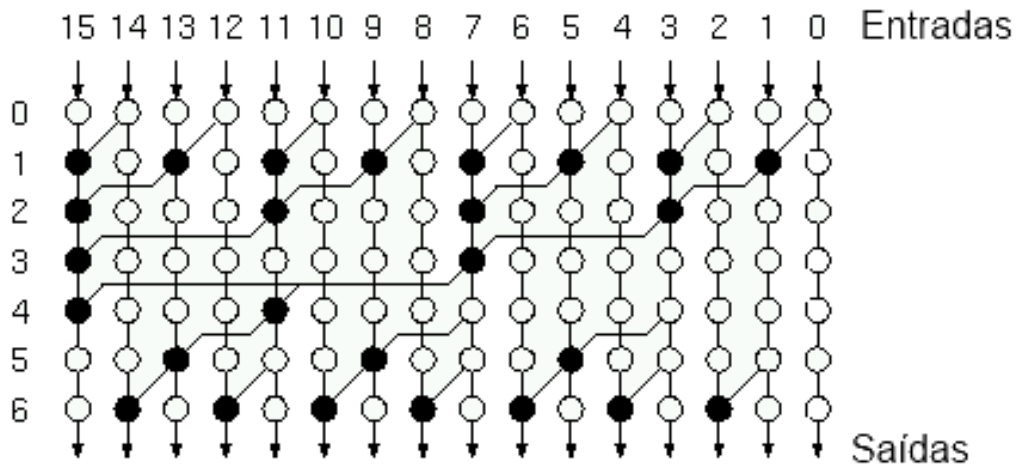

Figura 3.5: Representação gráfica do somador *Brent-Kung*

3.1.5 *Sklansky*

O somador *Sklansky* faz parte da família de somadores paralelos de prefixo que explore a antecipação de *carry*. É um somador com uma arquitectura dividida em três partes, a primeira corresponde ao cálculo dos P_i e G_i (*propagate* e *generate*) a partir dos operandos de entrada.

A segunda parte consiste no cálculo dos *carries* intermediários elaborados através de uma árvore binária.

Por último, a terceira parte calcula o resultado da soma final através de um somador constituído por portas XORs.

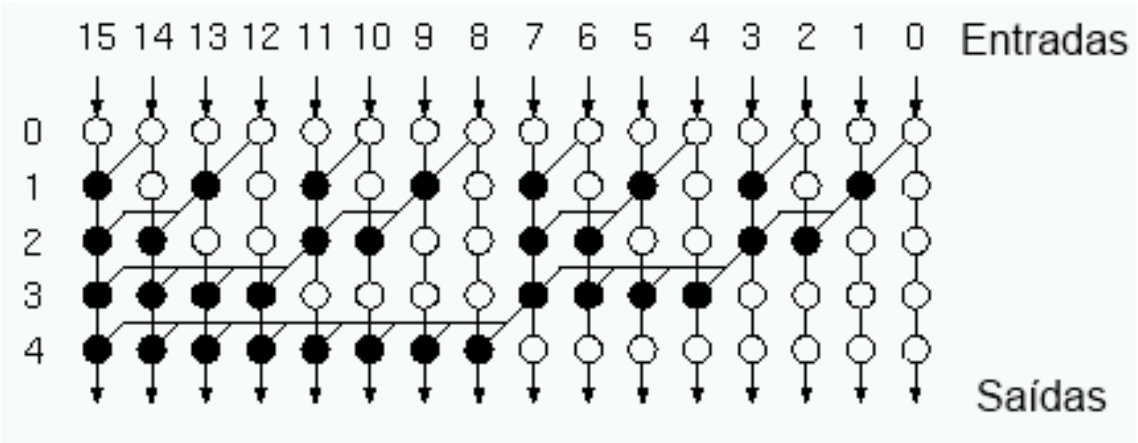

Figura 3.6: Representação gráfica do somador Sklansky

Na figura 3.6 verifica-se que o somador sklansky permite obter resultados bastante mais rápidos e tem uma arquitectura simples e regular mas nem sempre é vantajoso em termos de células utilizadas.

3.1.6 *Wallace tree*

Uma árvore de Wallace ou *Wallace tree* é parte de um somador de árvore que permite minimizar o atraso de propagação. Ao invés de adicionar os produtos parciais em pares como o faz o somador *ripple carry*, uma árvore de Wallace utiliza células CSA (*Carry Save Adder*) ligadas de forma a reduzir o número total de bits. A árvore tem como objectivo reduzir o número de operandos para dois (um vector de *carry* e outro de soma) para que no fim seja utilizado um somador convencional de forma a obter o resultado final da operação. A árvore de Wallace é muitas vez utilizada para a multiplicação binária como redução de produtos parciais.

Na figura 3.7 são apresentados dois esquemas através dos quais se verificam as vantagens e desvantagens na utilização de uma estrutura em forma de árvore de Wallace:

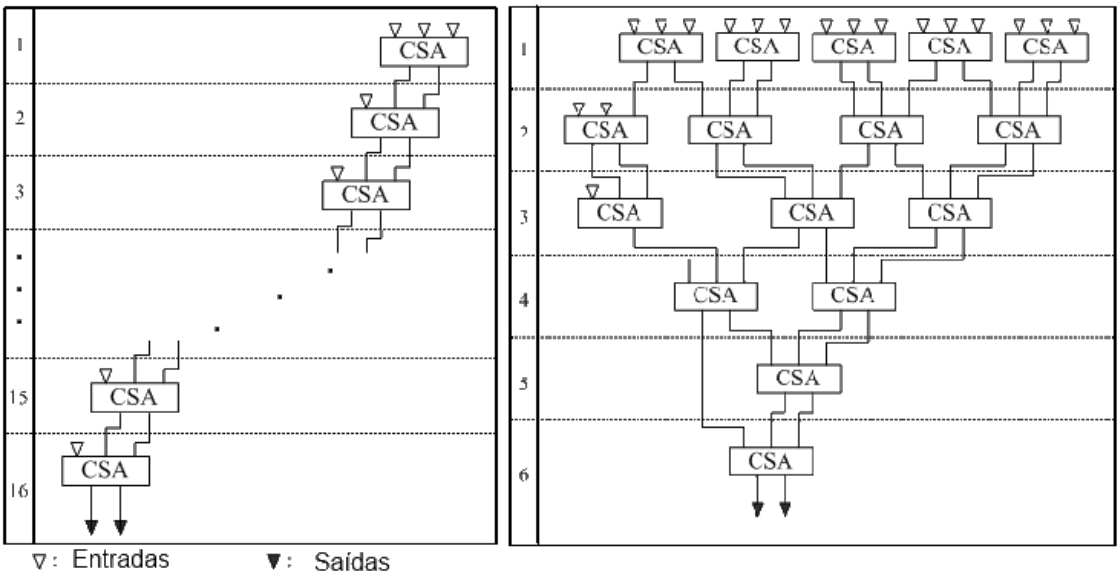

Figura 3.7: Estrutura em *array* e em árvore de Wallace de uma redução de produtos parciais

Um multiplicador em forma de árvore de Wallace não tem vantagens no que diz respeito ao número de *gates* pois este utiliza exactamente os mesmos recursos que um multiplicador em *array*. A vantagem da árvore de Wallace reside na diminuição do atraso que pela mudança de estrutura muda de $O(n+n)$ (proporcional à soma dos pesos de cada operando) para $O(n+log(n))$. Contudo se, numa estrutura em *array*, a interligação entre blocos lógicos era facilmente concebida e simples, numa estrutura em árvore de Wallace a interligação entre blocos lógicos torna-se muito mais complexa.

3.2 Projectos com FPGAs

3.2.1 Arquitectura da FPGA

As FPGAs (*Field Programmable Gate Array*) representam um dos maiores segmentos em crescimento da indústria de semicondutores e são largamente utilizadas para o processamento de informações digitais. Foram criadas pela Xilinx Inc. em 1985 e foram, desde então, rapidamente difundidas como uma excelente tecnologia para a implementação de circuitos digitais relativamente grandes e para baixos volumes de produção. Difere de outras tecnologias tais como a ASIC (*Application Specific Integrated Circuit*) pois é um dispositivo que pode ser programado de acordo com as aplicações do usuário (programador) e tem a particularidade de ser reprogramável. A FPGA disponível para testes é uma Spartan 3 da Xilinx e é segundo [1] divido pela seguinte arquitectura:

- Blocos lógicos configuráveis (CLBs) com as respectivas LUTs;
- Blocos de entrada e saída (IOBs) controlando o encaminhamento dos dados desde os pinos até à lógica interna do dispositivo;
- Blocos de memória RAM que permitem guardar dados;
- Blocos multiplicadores dedicados de dois números de até 18 bits;
- Gerador de *clock* digital (DCM) utilizados para regular uma frequência de clock multiplicando-a ou dividindo-a;
- Matriz de interconexões (*switch matrix*).

A FPGA contém células SRAM que são responsáveis por estabelecer as interligações entre blocos lógicos do circuito. Um bloco lógico (figura 3.8) é, de forma geral, constituído por uma LUT (*Look-Up-Table*) e de um *Flip-Flop*.

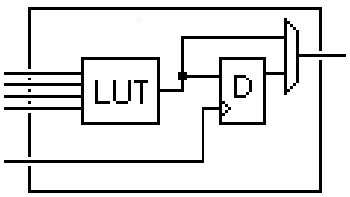

Figura 3.8: Bloco lógico

A LUT permite implementar equações lógicas tendo quatro entradas e uma saída e pode ser considerada como uma pequena memória, um multiplexador ou um *shift register*. O registo permite memorizar um estado de uma máquina de estado sequencial ou ainda sincronizar um sinal

permitindo usar técnicas utilizadas para diminuir o tempo de processamento (*pipeline*). Os inúmeros blocos lógicos são conectados entre eles por uma matriz de interligações configurável. Pode ser reconfigurável sempre que desejado e necessário mas ocupa um espaço significativo. A topologia é simples e fazem lembrar as ruas de ângulo recto de *Manhattan* formando matrizes simétricas e *switches* programáveis. A figura 3.9 mostra a arquitectura de uma FPGA.

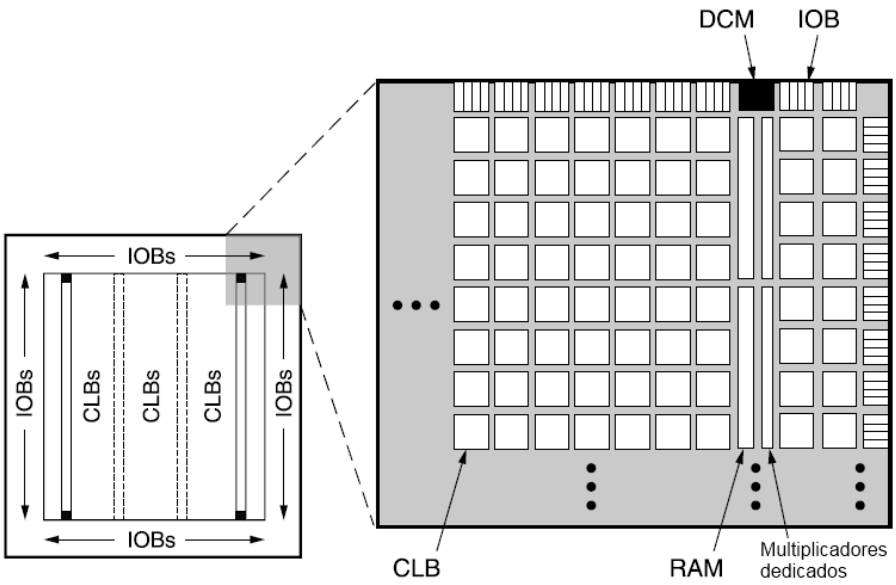

Figura 3.9: Arquitectura interna de uma FPGA (fonte: [1])

A densidade de LUTs existente numa FPGA já não permite a interligação de blocos manual, o esquema lógico desejado é então concretizado por uma interligação automática (*place and route*) de acordo com os recursos materiais da placa. Uma vez que os tempos de propagação dependem do comprimento das ligações entre células lógicas, os melhores resultados obtidos numa FPGA em termos de frequência diferem de esquema para outro, ou seja, os algoritmos de optimização dos *place and route* podem variar consoante o esquema. A ocupação dos blocos lógicos é, no entanto, muito boa permitindo a utilização quase total dos recursos necessários. Como a configuração ou *place and route* das LUT é feita por pontos de memórias voláteis, é necessário salvaguardar o design do circuito da FPGA numa memória não volátil externa como por exemplo numa memória *Flash* série.

Outra particularidade das FPGAs é o facto de possuir rápidas ligações dedicadas entre células vizinhas das quais, as mais comuns, são as *carry chains* que, conforme se vê na figura 3.10 permite rápida propagação de *carry* possibilitando a criação de funções aritméticas muito eficientes tais como as de uma adição ou de um contador.

Além dos CLBs presentes numa FPGA, existem blocos multiplicadores dedicados de 18×18 bits. Estes recursos são vantajosos porque realizam a multiplicação de forma mais rápida do que seria conseguido com a utilização de CLBs para o mesmo efeito. Permitem ainda a utilização de

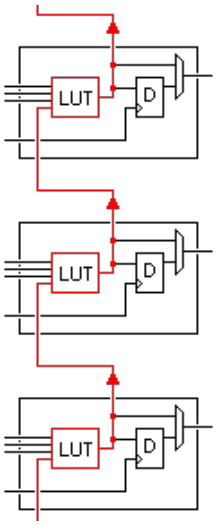

Figura 3.10: *Carry chain* de células vizinhas

vários multiplicadores em cadeia para realizar operações com mais de duas entradas ou com mais de 18 bits.

3.2.2 Procedimento de implementação

A implementação dos blocos operadores RNS requer um conhecimento prévio sobre a aritmética do sistema de numeração por resíduos pois o cálculo efectuado no RNS não é tão linear e requer um cálculo distinto daquele utilizado nos algorítmos do sistema de numeração por resíduos. Por exemplo, para representar um dado número pelos seus resíduos, é necessário conhecer os módulos antecipadamente para efectuar uma divisão do número pelo módulo e assim obter o resto da divisão e o respectivo resíduo. Sabendo que, devido à sua complexidade, a divisão não é viável no âmbito de uma implementação computacional de um RNS, foi necessário recorrer a outros métodos para a obtenção dos resíduos. O mesmo acontece para outros blocos operadores do RNS onde foi necessário estudar a aritmética do RNS antes da implementação.

Após o estudo da aritmética do RNS, foi determinado um fluxo de projecto que estabelecesse algumas etapas necessárias para o decorrer do projecto que permitiu estruturar o projecto dividindo-o em várias etapas. A primeira etapa consistiu na elaboração de um diagrama abstracto permitindo organizar as várias partes do operador RNS em blocos. Seguiu-se a construção de modelos Verilog sintetizáveis para cada bloco de operadores RNS. Fez-se a simulação funcional gerando estímulos através de *testbenches* para a validação dos módulos e foi feita a análise temporal.

Após a realização destes procedimentos para cada parte dos operadores RNS, procedeu-se à síntese XST de cada módulo. O *place and route* foi feito após validação dos passos anteriores e, por fim, realizou-se os testes dos operadores RNS numa FPGA.

A ferramenta utilizada para a simulação funcional foi o ModelSim XE da Xilinx [4]. A síntese, por sua vez, foi realizada através da ferramenta ISE igualmente da Xilinx. A ferramenta ISE permitiu também extrair os resultados relativos à área ocupada e à latência de cálculo. Por último, os testes realizados na FPGA consistiu na junção dos operadores RNS com um projecto distinto que permitiu a comumicação com a FPGA via porta série. O projecto utilizava uma interface que permitiu a escrita dos valores dos operandos nas entradas da placa e, da mesma forma, a leitura nas saídas dos resultados dos operadores.

3.3 Selecção de módulos

A selecção dos módulos para os sistemas de numeração por resíduos afecta tanto a eficiência da representação binária como a complexidade dos algoritmos aritméticos. Uma vez que a magnitude do maior módulo dicta a velocidade das operações aritméticas, a melhor escolha passará por minimizar quanto possível os módulos. O processo de escolha dos módulos RNS pode se tornar complexo, no entanto, a forma provavelmente mais simples para o efeito é de escolher números primos em sequência até que a gama dinâmica se torne adequada. O manuseamento dos módulos pode levar à diminuição do número de bits total e ainda à redução da magnitude do maior módulo aumentando assim a velocidade das operações aritméticas. Para a escolha dos módulos terá de se ter em conta ainda que potências de 2 simplificam as operações aritméticas. O módulo 16, por exemplo poderá ser melhor opção que o módulo 13 mesmo sendo este último menor que o primeiro. Outra opção para o mesmo efeito é a consulta de uma tabela (*table-lookup*) que acelera o processo mas têm o inconveniente de ocupar memória. Módulos na forma 2^{2n-1} e 2^{2n+1} ou ainda 2^n-1 e 2^n+1 são, à semelhança dos números potências de 2, os que apresentam melhores resultados no que diz respeito à latência e área. A restrição dos módulos implica o aumento do número de bits do maior resíduo mas pode diminuir o número de bits total a utilizar por operandos nos RNS e ainda melhora consideravelmente a simplicidade aritmética tornando-se num dos factores mais fortes a ter em conta no âmbito da selecção dos módulos para a realização dessa dissertação. O facto da conversão de um número x binário para RNS consistir em encontrar os módulos $x_i = x \ mod \ m_i$ que são os restos de uma divisão de x por m_i levaram à escolha de módulos de potências de 2 pois, apesar dos cálculos serem realizados simultâneamente, usar divisores não representa o melhor método. Assim sendo, para módulos do tipo:

$$\mathrm{RNS}(2^n-1|2^n|2^n+1) \tag{3.9}$$

temos as seguintes facilidades:

- O resto do módulo 2^n é imediato repartindo apenas os n bits menos significativos da variável binária x a converter;

n	Gama dinâmica	Número de bits
5	3276	15
6	2262080	18
7	2097024	21
8	16776960	24
9	134217216	27
12	68719472640	36
15	35184372056064	45
18	18014398509219840	54
21	9223372036852678656	63

Tabela 3.1: Tabela das gamas dinâmicas

- O resto do módulo $2^n - 1$ pode ser obtido apenas através de operadores aritméticos de adição;
- E por último o resto do módulo $2^n + 1$ requer operadores aritméticos de adição e eventualmente de subtracção.

Esses três módulos RNS $\{2^n - 1, 2^n, 2^n + 1\}$ são de especial interesse devido às inúmeras operações que podem ser realizadas de forma eficaz com área limitada e com ausência do uso de memória. Para este efeito árvores de somadores serão implementadas para a redução de x à soma de 2 inteiros de n bits sem alterar o valor numérico do resto do módulo m_i no sentido de aumentar as *performances* para a conversão binária/residual e ainda para somadores modulares mesmo para grandes valores de n. Uma abordagem mais detalhada será desenvolvida nos próximos capítulos para uma melhor percepção sobre as vantagens e os passos a realizar na utilização desses módulos. A tabela 3.1 mostra as gamas dinâmicas provenientes dos módulos $\{2^n - 1, 2^n, 2^n + 1\}$ para vários números de n.

3.4 Somador modular

A operação aritmética da adição é a mais relevante deste trabalho uma vez que está presente em todas as etapas do sistema de numeração por resíduos. Nesta secção são apresentados somadores modulares de estruturas diferentes, todos eles executam o cálculo apresentando à saída o resíduo do resultado da operação de adição.

3.4.1 Somador modular CLA

O somador modular CLA é baseado numa arquitectura sugerida por [5], apresenta uma estrutura semelhante ao CLA convencional mas está preparado para executar o cálculo para os módulos $2^n - 1$ e $2^n + 1$. A figura 3.11 mostra de que forma é realizado o cálculo.

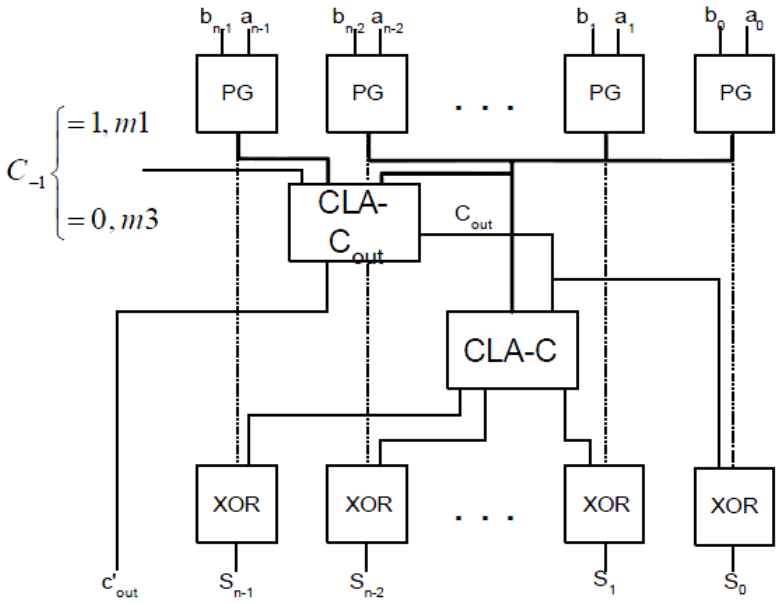

Figura 3.11: Somador módulo $2^n \pm 1$

Este somador difere de um CLA convencional no cálculo do bit de transporte (*Carry out*) que é gerado pelos sinais de geração (*generate*) e de propagação (*propagate*) calculados a jusante do circuito e depende do sinal que define se se pretende calcular o resíduo do módulo $2^n - 1$ ou o resíduo $2^n + 1$ (C_{-1} na figura 3.11).

$C_{out} = G_{n-1} + P_{n-1}G_{n-2} + \cdots + P_{n-1}P_{n-2}\cdots P_1P_0C_{-1}$

Os bits de transporte são gerados usando C_{out}:

$C_0 = G_0 + P_0C_{out}$

$C_1 = G_1 + P_1G_0 + P_1P_0C_{out}$

$\vdots$

$C_{n-1} = G_{n-2} + P_{n-2}G_{n-3} + \cdots + P_{n-2}\cdots P_1P_0C_{out}$

C_{out} é invertido para o cálculo do módulo $2^n + 1$ por um negador condicional (depende de C_{-1}) através de uma porta lógica XOR. É de salientar que um resíduo de um módulo $2^n + 1$ pode ter $n + 1$ bits quando este é igual a 2^n, neste caso outro sinal C_{out} representa o bit de transporte (C'_{out} na figura 3.12) :

$$C'_{out} = P_{n-1}Pn - 2\cdots P_1P_0C_{out}$$

A última etapa do somador executa o cálculo final por uma camada de XORs :

$$S_i = P_i \oplus C_{i-1}$$

$$S_0 = P_0 \oplus C_{out}$$

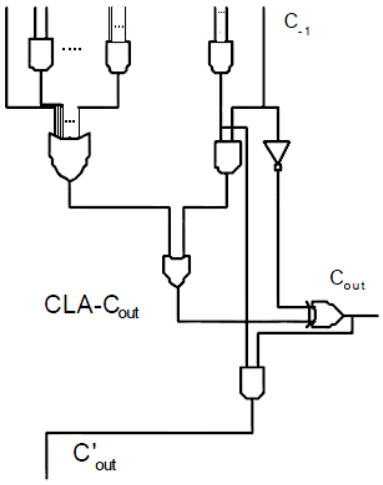

Figura 3.12: C_{out} do módulo 2^n+1

3.4.2 Somador modular *Brent Kung*

O somador *Brent kung* (BK) implementado é baseado segundo a fonte [6], trata-se de uma modificação de um somador Sklansky com realimentação de *carry* ou *End Around Carry* (EAC). É um somador de prefixo paralelo (ver secção 3.1.5) que utiliza células *Brent Kung* (ver secção 3.1.4) calculando espontaneamente uma soma 2^n e corrigindo o resultado quando se trata de uma soma 2^n-1 ou 2^n+1 com o EAC. Este somador é dividido por três etapas, na primeira são calculados os valores de geração (G_i) e propagação (P_i) de *carry* (ver fórmulas 3.5) em paralelo para todas as posições dos operandos de entrada através de células *Half Adder* (HA). A segunda parte já é executada na estrutura de prefixo onde em cada célula BK são gerados ou propagados o *carry* de acordo com as fórmulas 3.7 e 3.8 respectivamente calculando esses valores desde o bit menos significativo dos operandos até cada uma das outras posições. Por se tratar de um somador modular, ainda se deve realimentar a última etapa com o último carry obtido. A terceira e última etapa calcula os vectores de saída e de *carries* para calcular o resultado através de uma camada de XORs.

A figura 3.13 mostra o esquema de um somador modular 2^n-1 *Brent Kung* de dois operandos A e B que tem o seguinte funcionamento:

$$A+B<2^n-1 \Rightarrow S=A+B;$$

$$A+B\geq 2^n-1 \Rightarrow S=\langle A+B+1\rangle_{2^n};$$

Em ambos os casos temos $S=\langle A+B\rangle_{2^n-1}$. O resultado depende do último *carry* c_n, se $c_n = P$ (propagação de carry da última célula) então $A + B = 2^n-1$, se $c_n = G$ (geração de carry da

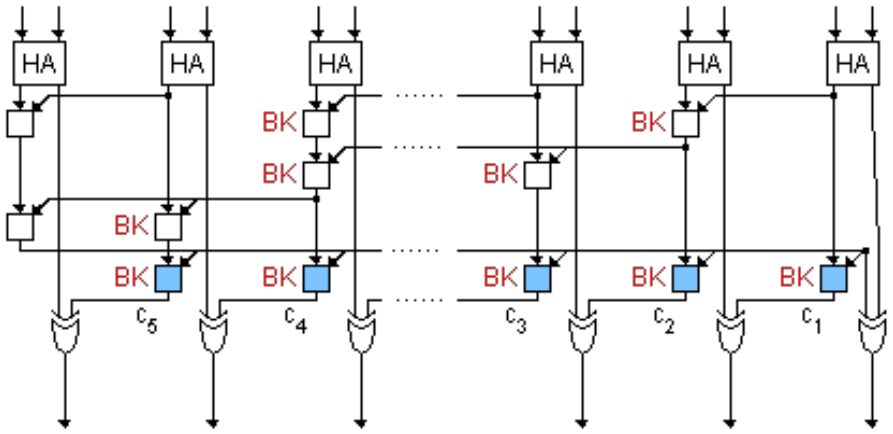

Figura 3.13: Somador módulo $2^n - 1$

última célula) então $A + B > 2^n - 1$, senão $A + B$ é garantidamente menor que $2^n - 1$. A figura 3.14 representa um somador que calcula a adição modular respectiva ao módulo $2^n + 1$.

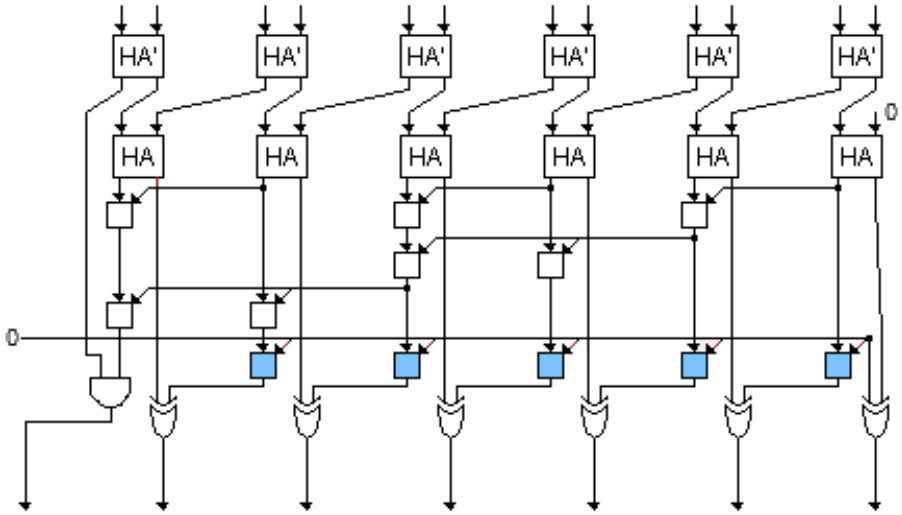

Figura 3.14: Somador módulo $2^n + 1$

O somador *Brent Kung* foi realizado de forma a calcular os três resíduos em paralelo, é de esperar que a latência deste somador seja igual ao do maior caminho crítico que (obviamente) provem do cálculo do resíduo que diz respeito a $2^n + 1$ por ter mais um bit nos resíduos de entrada do somador. O cálculo da soma do resíduo do módulo 2^n é o mais rápido pois só necessita guardar os n bits menos significativos para obter o resíduo não necessitando do EAC que contribui de forma significativa para o aumento da latência mas que é inevitável para o cálculo dos restantes resíduos.

3.4.3 Somador modular de menor complexidade

A estrutura de um somador modular pode ser diferente dos apresentados anteriormente. Tal como a multiplicação RNS, a adição modular pode ser dividida em uma adição seguida de uma redução modular diminuindo significativamente a complexidade do circuito. A estrutura deste somador é semelhante ao da multiplicação modular apresentado na próxima secção 3.5. Contrariamente à redução modular de inteiros, a redução modular que se segue à soma dos resíduos neste somador modular necessita apenas de uma única subtracção. A soma de dois resíduos a e b por um módulo m_i ($m_1 = 2^n - 1$, $m_2 = 2^n$ e $m_3 = 2^n + 1$) com $0 \leq a, b < m_i$ é determinada da seguinte forma:

$$s = a + b$$

$$t = s - m_i$$

Se ocorrer *overflow* na subtracção significa que s, a soma de a com b é menor que m_i não necessitando da redução modular após a soma. No caso contrário, se ocorrer *overflow*, o resultado final seria o resultado da subtracção. Isto só é possível porque se trata de uma soma modular de dois resíduos. Os operandos da operação são ambos de menor valor que m_i e o resultado máximo que se pode obter é $2 \cdot (m_i - 1)$. Esta estrutura permite ainda a utilização de diferentes somadores, do tipo RCA por exemplo, devido à sua baixa complexidade.

Este somador modular é composto por uma redução modular, redução essa que só pode ser utilizada com dois operandos de menor valor que o módulo m_i. Os resíduos são sempre menor que m_i mas no âmbito de uma redução modular de um número binário de $3n$ bits (ver tabela 3.1) tal operação seria impossível. Este é o motivo pelo qual é utilizado a camada de CSAs no inicio de uma conversão de binário para RNS, isto é, permite garantir que um número de $3n$ bits seja reduzido a dois operandos de n bits cada podendo desta forma utilizar este somador modular para a conversão de binário para RNS.

3.5 Multiplicador modular

Uma vez que a operação aritmética da multiplicação é de principal importância para quase todos os tipos de processadores, uma implementação eficiente da multiplicação modular é necessária para garantir a obtenção de resultados relevantes no âmbito desta dissertação. No entanto, o facto de se tratar de uma multiplicação por resíduos leva a uma redução modular do resultado da multiplicação aumentando significativamente o tempo de atraso de um multiplicador modular. Segue-se

a apresentação de multiplicadores de módulos 2^n-1 e 2^n+1 que tem como entrada dois operandos residuais e devolve à saída o resultado residual do produto. As multiplicações são efectuadas em três etapas conforme o demonstra a figura 3.15.

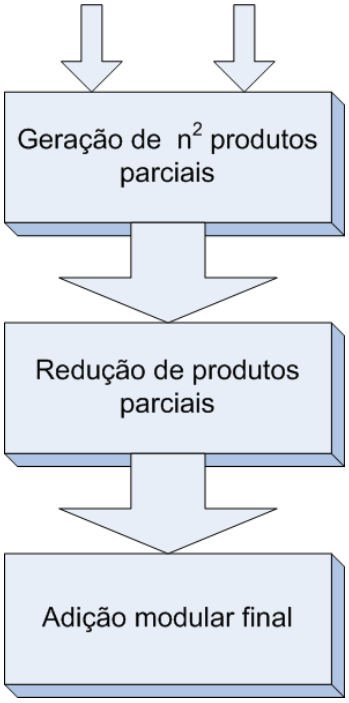

Figura 3.15: Multiplicador modular

Sejam dois operandos $A = \{a_{n-1}a_{n-2}\ldots a_0\}$ e $B = \{b_{n-1}b_{n-2}\ldots b_0\}$ que também podem ser representados da seguinte maneira:

$$A = \sum_{k=0}^{n-1} a_k 2^k \quad e \quad B = \sum_{l=0}^{n-1} b_l 2^l$$

O produto $A \cdot B$ será então:

$$AB = \sum_{k=0}^{n-1} a_k 2^k \cdot \sum_{l=0}^{n-1} b_l 2^l$$

e, respectivamente, o módulo do produto será:

$$\langle AB \rangle_{2^n \pm 1} = \left\langle \sum_{k=0}^{n-1} a_k 2^k \cdot \sum_{l=0}^{n-1} b_l 2^l \right\rangle_{2^n \pm 1}$$

São inicialmente calculados os produtos parciais dos módulos 2^n-1 e 2^n+1 de A e B numa matriz de portas lógicas do tipo AND interligando todos os bits de cada operando, ou seja, para operandos de n bits são gerados n números de n bits cada para um total de n^2 produtos parciais.

Na segunda etapa é efectuada uma redução dos produtos parciais resultantes da primeira etapa anterior para dois números de n bits. Esta etapa é efectuada através de uma matriz de *full adders* ou *carry save adders* que através do paralelismo diminui o tempo de atraso do circuito.

A última etapa consiste em somar os dois números de n bits obtidos anteriormente utilizando um dos somadores modulares.

3.6 Redução modular

A conversão de binário para RNS de um número X de n bits ou redução modular consiste em calcular os $x_i = X \bmod m_i$ para $0 < i \leq 3$ e onde $m_1 = 2^n - 1$, $m_2 = 2^n$ e $m_3 = 2^n + 1$. x_i representa o resto da divisão de X por m_i. Todos estes cálculos podem ser efectuados simultaneamente, no entanto, utilizar divisores levará ao aumento da complexidade e do tempo de processamento. A selecção dos módulos também foi pensada para este problema e o facto dos módulos serem potências de dois facilita a redução modular pois evita a implementação de divisores. A conversão para resíduos dos módulos m_1 e m_3 são obtidos através de operadores de adição e uma adição modular final enquanto que o resíduo correspondente a m_2 é obtido de forma directa através do n bits menos significativos do número X.

Foi implementado duas estruturas de conversores distintas, a primeira foi implementada utilizando árvores de Wallace com células FA convertidas em CSA. A implementação foi baseada em [6]. A figura 3.16 demonstra de que forma é feita a redução modular $m_1 = 2^n - 1$ de um número X de 15 bits, $n = 5$ bits e consequentemente $m_1 = 31$. A redução modular por árvore de wallace tem o inconveniente de ter como última camada uma "realimentação"de *carry* ou *End-Around-Carry* (EAC) que aumenta a latência do circuito. A redução modular de m_3 utiliza uma mesma estrutura mas ocupando mais recursos.

O segundo conversor de binário para RNS é muitas vezes utilizado nos estudos para a optimização do RNS ([7] e [5]) e consiste em repartir o número binário em parcelas de números mais pequenos e calcular os seus resíduos separadamente. Assim, com os módulos $m_1 = 2^n - 1$, $m_2 = 2^n$ e $m_3 = 2^n + 1$, esta operação é facilitada pois a gama dinâmica por eles estabelecida corresponde a números binários de $3n$ bits. Desta forma, o número binário é divido em três parcelas de n bits. Se a cada parcela for associado uma variável, A para os n bits mais significativos, B para os n bits intermediários e finalmente C para os n bits menos significativos, pode-se determinar a seguinte igualdade:

$$X = A \cdot 2^{2n} + B \cdot 2^n + C \tag{3.10}$$

O resíduo correspondente ao módulo $m_2 = 2^n$ é desde já determinado com os n bits menos

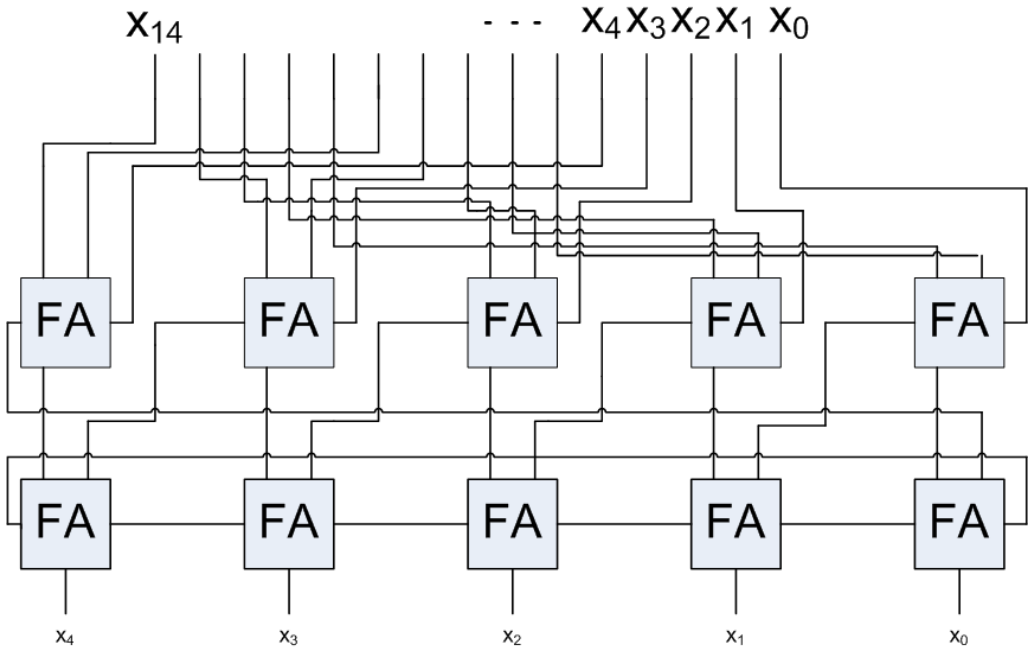

Figura 3.16: Redução modular m_1

significativos uma vez que ao dividir X por 2^n, o resto da divisão é igual aos n bits menos significativos de X, logo:

$$x_2 = \langle A \cdot 2^{2n} + B \cdot 2^n + C \rangle_{2^n}$$

$$x_2 = C. \tag{3.11}$$

Os restantes resíduos x_1 e x_3, para respectivamente $m_1 = 2^n - 1$ e $m_3 = 2^n + 1$, podem ser determinados através das equações que se seguem:

$$x_1 = \langle A \cdot 2^{2n} + B \cdot 2^n + C \rangle_{2^n - 1}$$

$$x_1 = \langle A \cdot (2^n - 1)(2^n + 1) + A + B \cdot (2^n - 1) + B + C \rangle_{2^n - 1}$$

$$x_1 = \langle A + B + C \rangle_{2^n - 1} \tag{3.12}$$

$$x_3 = \langle A \cdot 2^{2n} + B \cdot 2^n + C \rangle_{2^n+1}$$

$$x_3 = \langle A \cdot (2^n - 1)(2^n + 1) + A + B \cdot (2^n + 1) - B + C \rangle_{2^n+1}$$

$$x_3 = \langle A - B + C \rangle_{2^n+1} \tag{3.13}$$

Na figura 3.17 está representado o conversor de binário para RNS para os módulos $m_1 = 2^n - 1$ e $m_3 = 2^n + 1$, é composto por uma camada de CSAs reduzindo de três operandos para dois para terminar num somador modular que pode calcular o resíduo para ambos os módulos. Para m_3, tem que se complementar o *buffer* $B[n:0]$ e igualar *cin* a 1.

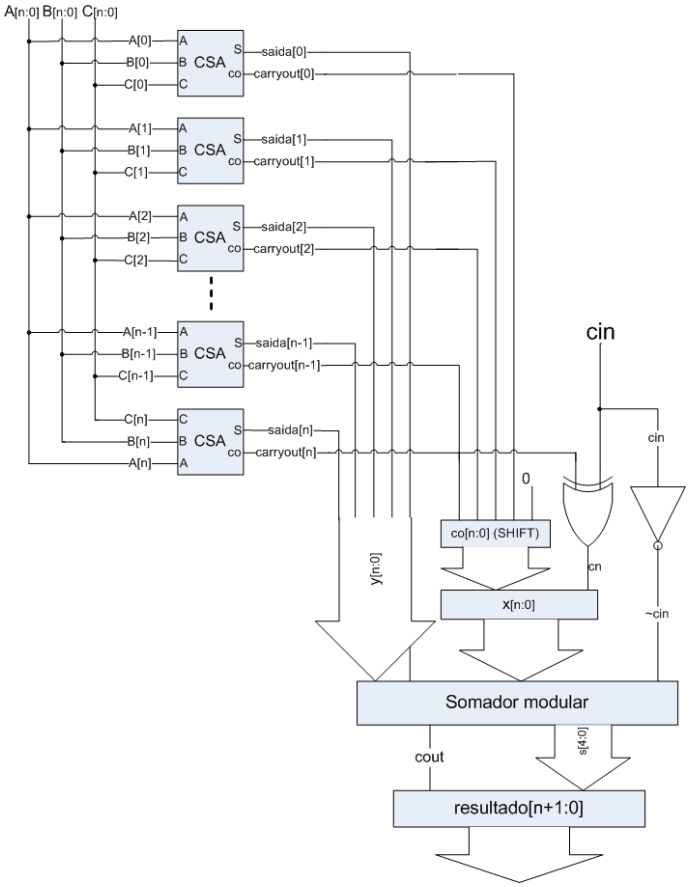

Figura 3.17: Redução modular m_1 e m_3

3.7 Conversão modular inversa

A conversão de RNS para binário ou conversor inverso é o processo mais complexo do sistema de numeração por resíduos. A sua implementação foi baseada em [8] e [5] por ser de menor complexidade levando ao menor tempo de atraso e à maior simplicidade de implementação. O esquema da figura 3.18 mostra as diferentes etapas do processo.

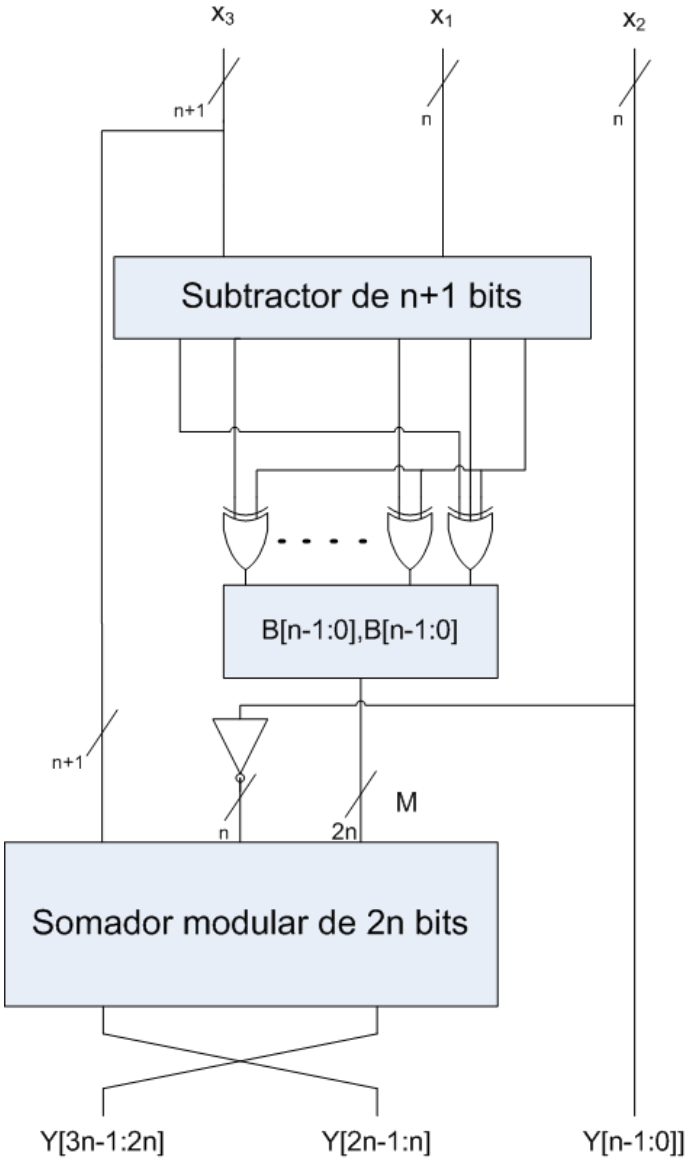

Figura 3.18: Conversor de RNS para binário

Este conversor é constituido por três partes, na primeira é subtraido o resíduo x_1 pelo resíduo x_3 provenientes dos operadores RNS a montante do circuito. A segunda parte gera M que é igual a $B \cdot (2^n + 1)$ que corresponde a concatenar B com ele próprio. Por fim, a terceira parte consiste num somador modular de $2n$ bits.

3.8 Conclusões

O somador modular presente tanto na multiplicação modular como em ambas as conversões, directa e inversa, pode assumir qualquer uma das estruturas apresentadas levando certamente a resultados diferentes em termos de área e de latência. Muitas das FPGAs possuem ligações dedicadas de elevado desempenho para a propagação de *carry* (*carry chains*), tornando difícil obter bons resultados aplicando as técnicas apresentadas neste capítulo. O atraso provocado pela complexidade das interconexões de blocos lógicos empregando essas técnicas pode eclipsar parte dos ganhos obtidos nos tempos de atraso. Por estas razões, vários testes serão realizados para a comparação das arquitecturas de somadores e multiplicadores presentes neste capítulo de forma a seleccionar os módulos RNS de menor latência e área para a comparação com os operadores convencionais de inteiros. A apresentação destes resultados e a respectiva análise é feita no próximo capítulo.

Capítulo 4

Resultados

No capítulo 3 foram apresentados diversas formas de implementação de operadores aritméticos modulares para as várias etapas de um sistema de numeração por resíduos. A implementação estrutural (completamente) combinacional pode não apresentar os melhores resultados mesmo utilizando uma variedade de estruturas visando a redução da latência e da área. Neste início de capítulo são analisadas as diferentes estruturas implementadas. Da mesma forma, são comparadas as estruturas anteriormente mencionadas com uns circuitos de estrutura semelhante diferindo apenas na abordagem de especificação dos modelos verilog, semi-comportamental e semi-combinacional. A parte comportamental apresenta a mesma funcionalidade e apenas tem como objectivo a implementação de operadores aritméticos (-, + e $\times$) que tiram proveito das particularidades da FPGA tais como a rápida propagação de *carry*, permitida pelas ligações dedicadas entre células vizinhas, e a utilização dos multiplicadores dedicados presentes na FPGA. Pretende-se, com estas comparações, averiguar quais das estruturas resultam num melhor desempenho e quais serão comparadas com os operadores convencionais.

Neste capítulo são ainda apresentados todos os resultados da comparação de um sistema de numeração por resíduos com os operadores convencionais de inteiros.

Todos os módulos foram sintetizados numa Spartan 3 xc3s200 da Xilinx. Os resultados das comparações entre operadores modulares e operadores convencionais são ainda comparados de forma expedita com os resultados obtidos num artigo ([7]). Daqui em diante, entende-se como implementação funcional de um operador o resultado inferido, durante o processo de síntese, de uma operação aritmética ($-$, $+$ e $\times$).

4.1 Selecção dos operadores modulares para a caracterização

A selecção dos módulos para a comparação dos operadores RNS com os operadores convencionais tem como objectivo a obtenção de uma análise optimizada comparando nas próximas secções os operadores convencionais com (apenas) os operadores RNS que apresentam a melhor

relação de latência e de área. Neste sentido, foram implementados operadores de estrutura semelhantes mas substituindo partes destes operadores modulares por operadores inseridos no processo de síntese a partir de modelos verilog com uma descrição funcional (-, + e $\times$) sempre que destes últimos resultassem melhorias em termos de latência e de área.

A comparação é feita para $n = 5$, ou seja, para números binários de 15 bits ($m = 3n$) tendo uma gama dinâmica de $\{0, 32735\}$.

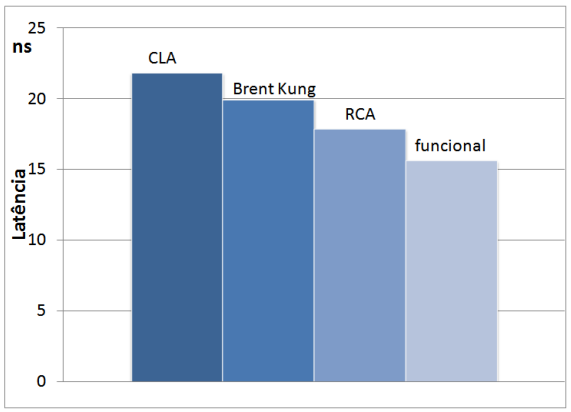

Figura 4.1: Latências de conversões de binário para RNS

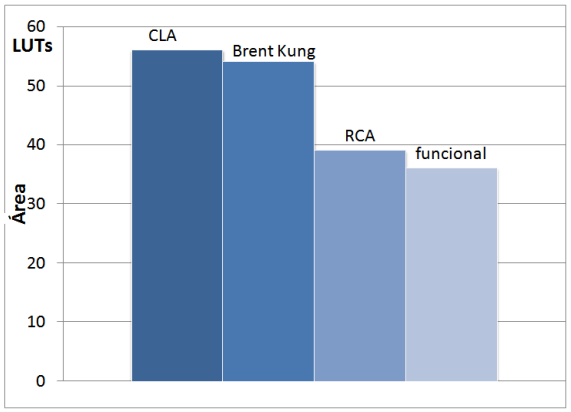

Figura 4.2: áreas de conversões de binário para RNS

As figuras 4.1 e 4.2 mostram a comparação de latências e de área respectivamente para a conversão de um número binário para RNS. É de realçar que a comparação é feita com conversores que calculam os três resíduos de um único operando. Claramente se verifica que a conversão de binário para RNS com um somador do tipo CLA é pior tanto em latência como em número de LUTs ocupados. A redução modular com RCA consegue ser mais rápida e de menor área que ambos os conversores com somadores CLA e *Brent Kung*. No entanto, a implementação funcional é a mais rápida e o circuito resultante é o que ocupa menor área. Os conversores de maior área são os circuitos de maior complexidade, embora sejam projectados tendo em consideração a exploração de paralelismo. O conversor com RCA e o conversor funcional são mais rápidos e de menor dimensões devido à utilização das ligações rápidas de transporte de *carry* entre células.

Pode-se verificar no gráfico da figura 4.3 que um somador RNS do tipo CLA tem um tempo de processamento aproximadamente igual ao de um somador RNS do tipo *Brent Kung*. O somador RNS RCA tem uma latência menor que os dois anteriores mas é mais lento que um somador RNS funcional que, conforme mostra o gráfico 4.4 também tem menor área.

Conclui-se que a FPGA gera operadores aritméticos mais rápidos e com um circuito de interligações mais rápidas do que um simples RCA.

As figuras 4.5 e 4.6 mostram os resultados das comparações de latência e área respectivamente dos multiplicador RNS. A estrutura dos multiplicadores RNS é igual para o multiplicador com somador modular final do tipo RCA e *Brent kung*, em ambos são gerados os produtos parciais

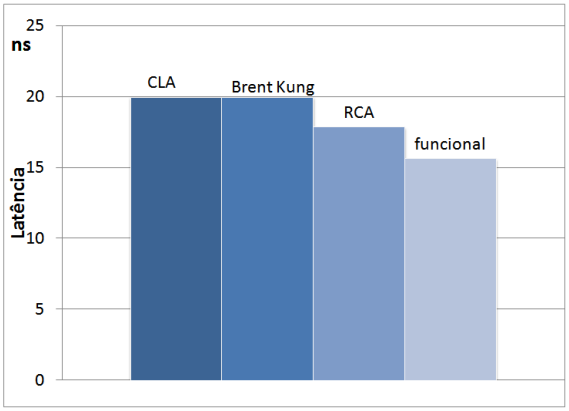

Figura 4.3: Latências de somadores RNS

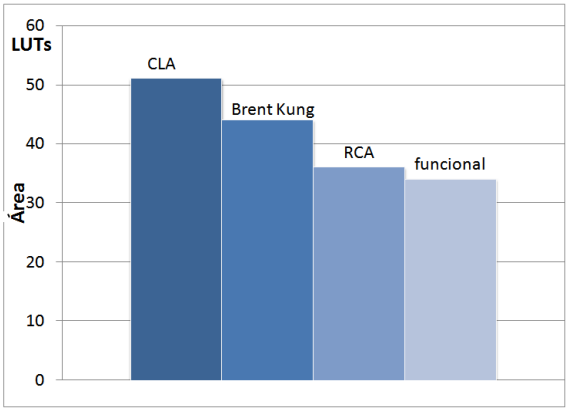

Figura 4.4: Áreas de somadores RNS

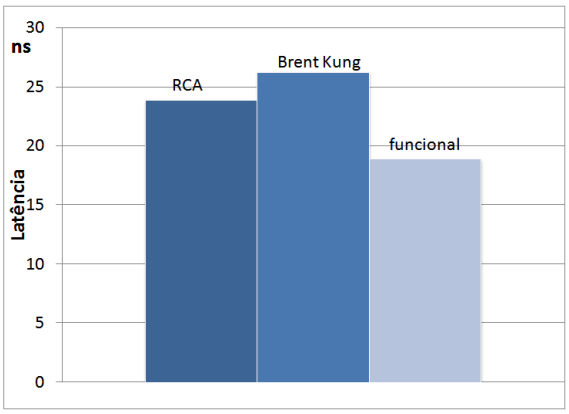

Figura 4.5: Latências de multiplicadores RNS

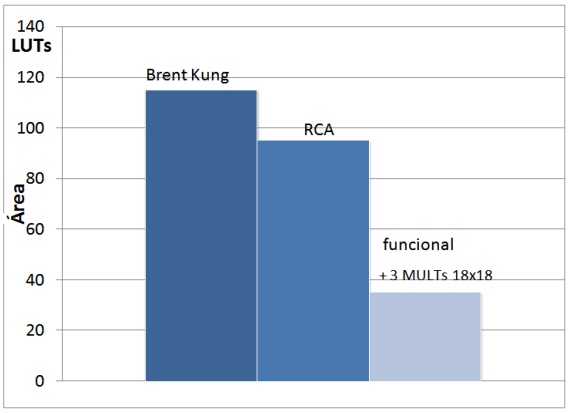

Figura 4.6: Áreas de multiplicadores RNS

e a redução dos mesmos acabando por passar por um somador modular do tipo RCA e *Brent kung* respectivamente.

A multiplicação denominada como funcional já é implementada de outra forma para poder utilizar os multiplicadores dedicados da FPGA e o impacto desses dispositivos embutidos são notáveis. O cálculo é mais rápido e o número de LUTs ocupados mais baixo, embora ocupando três multiplicadores dedicados.

Por fim, as conversões de RNS para binário são comparadas nas figuras 4.7 e 4.8. Mais uma vez, a implementação funcional apresentou os melhores resultados.

A utilização dos 12 multiplicadores dedicados da FPGA é uma mais valia para a diminuição do tempo de atraso mas pode não o ser no âmbito da comparação do RNS com os operadores inteiros uma vez que também utilizam esses recursos. O mesmo acontece com as *carry chains* que beneficia a rapidez do cálculo assim como a área ocupada mas o mesmo fará para os operadores convencionais.

Na próxima secção inicia-se a análise entre os operadores convencionais e os operadores RNS. Após a análise desta última secção e para a comparação de operadores aritméticos RNS com operadores aritméticos convencionais, foram escolhidos os operadores parcialmente funcionais

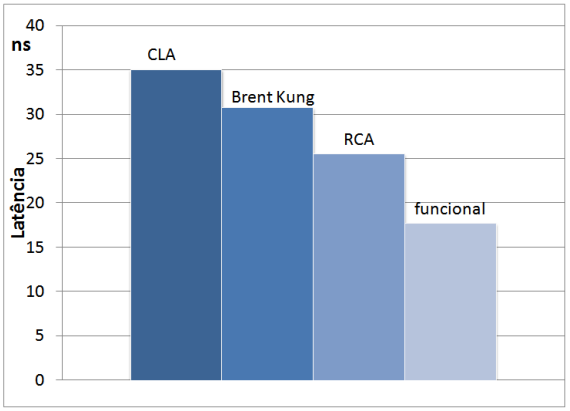

Figura 4.7: Latências de conversores de RNS para binário

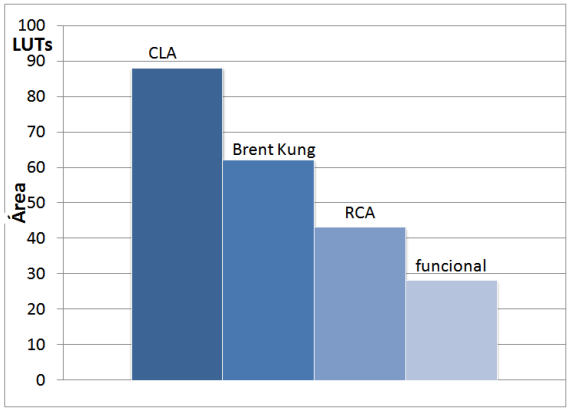

Figura 4.8: Áreas de conversores de RNS para binário

devido aos melhores resultados em todas partes dos operadores RNS.

4.2 Comparação entre somadores RNS e somadores de inteiros

Seja uma operação de adição de inteiros comum de dois operandos A e B e seja S o seu resultado. Para a adição de inteiros, se A e B possuem m bits, facilmente se chega à conclusão que S possuirá $m+1$ bits. No entanto, A e B podem ter um número de bits diferente um do outro e o número de bits de S dependerá dos seus operandos de entrada.

Na adição aritmética RNS tal não é possível por dois motivos, o primeiro diz respeito à conversão para RNS dos operandos A e B que terá de ser obrigatoriamente realizada pelos mesmos módulos $\{2^n-1, 2^n, 2^n+1\}$. Com uma redução modular de n bits, os operandos A e B serão de $m = 3n$ bits podendo (obviamente) assumir qualquer valor possível. É de realçar que tanto a redução modular dos dois operandos como a adição dos resíduos e a conversão inversa final do resultado tem que ser realizada utilizando o mesmo conjunto de módulos. Isto leva ao segundo motivo, ou seja, se a conversão inversa também é de n bits, o resultado terá de ser igualmente de m bits contra os $m+1$ da adição de inteiros.

Concluindo, a adição RNS está limitada pela gama de valores possíveis obtida com os módulos (ver página 5) e depende do maior valor envolvido na operação que é o resultado S. Então, se S é composto por m bits, o número de bits dos operandos de entrada A e B será igual. O utilizador terá que ter em conta que um resultado de uma operação de $m+m$ bits não ultrapassará os m bits. Este facto dificulta a comparação da adição aritmética de inteiros com RNS. Para que a comparação seja a mais justa ou equivalente possível, a adição aritmética de inteiros é realizada da mesma forma que a adição RNS, ou seja, A, B e S são ambos de m bits.

Em consequência, as figuras 4.9 e 4.10 mostram a comparação da latência e da área, respectivamente, entre a adição de inteiros e a adição modular para a variação de n bits, sendo n o número de bits que determina a gama dinâmica da operação (ver tabela 3.1), ou seja, compara a adição de

inteiros de $3n$ bits com as adições modulares de 3 resíduos de n bits cada executadas em paralelo com as conversões de binário para RNS e de RNS para binário (completo).

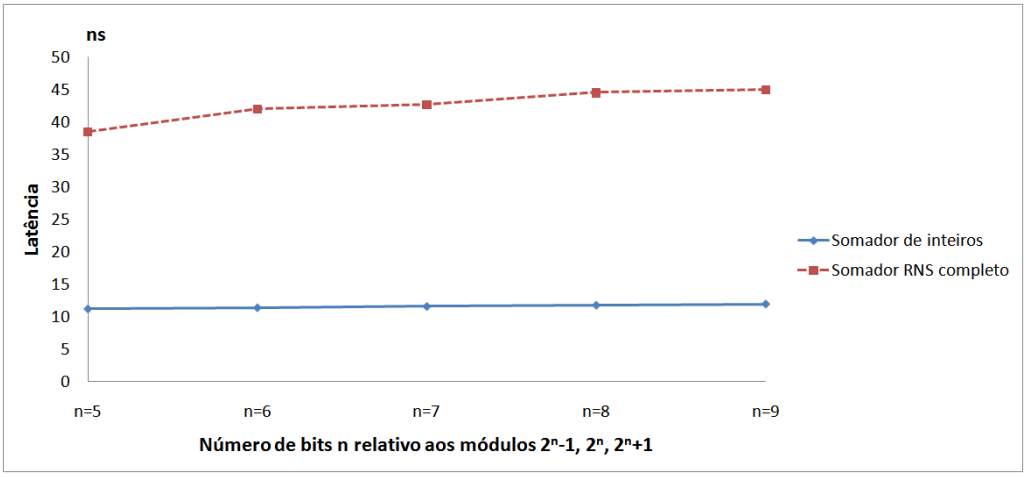

Figura 4.9: Latências de somadores inteiros vs RNS (completo)

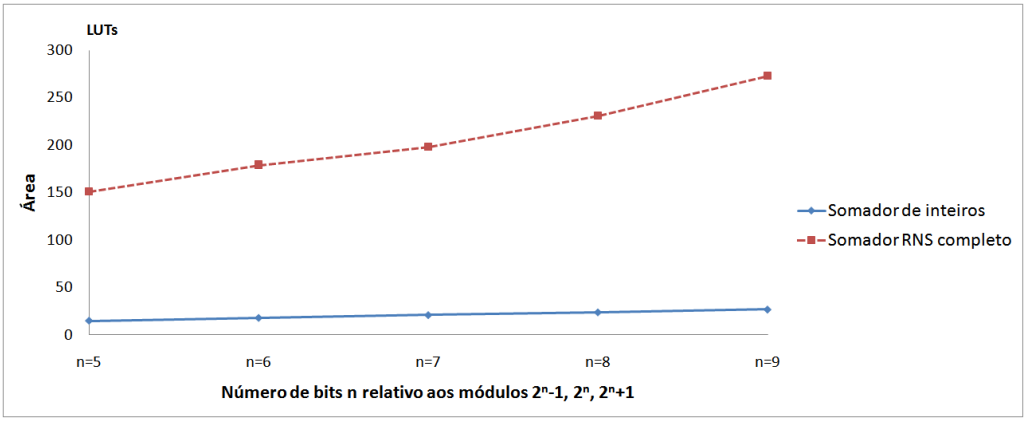

Figura 4.10: Áreas de somadores inteiros vs RNS (completo)

Perante os resultados obtidos claramente se verifica que tanto em área como em latência, a adição modular apresenta resultados piores que a correspondente operação de inteiros.

Nas latências, o somador RNS executa o cálculo num tempo aproximadamente quatro vezes superior ao de um somador convencional de inteiros e ambos aumentam de forma quase linear à medida que o número n de bits aumenta.

Verifica-se também que a área de um somador modular é sempre superior à área de um de inteiros e a diferença por eles obtidos acentua-se à medida que n aumenta.

É de realçar que o somador RNS utilizado nesta comparação é um operador completo com entradas e saída de m bits ($3n$ bits) sendo necessárias as conversões de binário para RNS dos

operandos e a conversão final do resultado em RNS para binário aumentando significatiamente a latência e a área.

Foram novamente comparados os dois somadores retirando esse acréscimo provocado pelas reduções modular e pela conversão inversa do somador RNS. Esse teste tinha como objectivo verificar se, no caso de uma sequência de adições, compensasse a utilização de um sistema de numeração por resíduos amortizando os inconvenientes obtidos, tanto em termos de latência como em termos de área, pelos conversores binário/RNS e RNS/binário.

As figuras 4.11 e 4.12 ilustram os resultados.

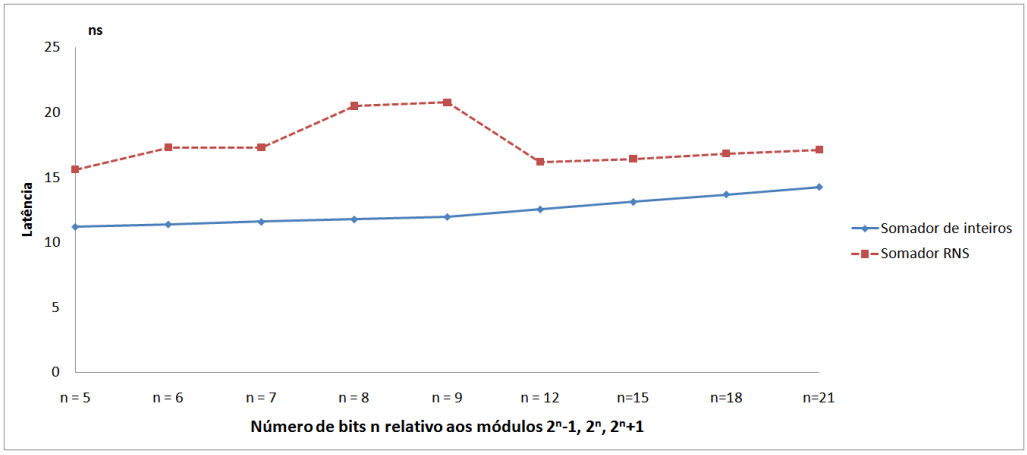

Figura 4.11: Latências de somadores inteiros vs RNS

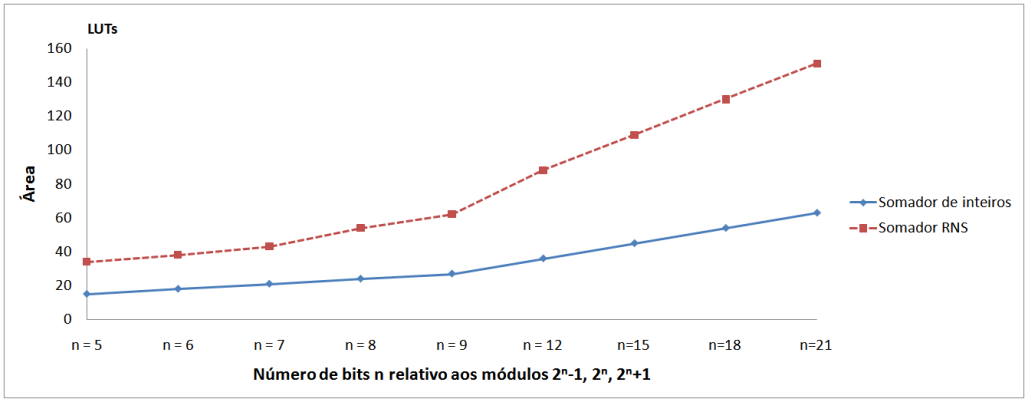

Figura 4.12: Áreas de somadores inteiros vs RNS

Verifica-se na figura 4.11 que a latência de um somador modular é sempre superior ao de um de inteiros e que ambos crescem linearmente com o aumento de n. No entanto, para um número de bits elevado, a latência do somador de inteiros aproxima-se muito lentamente do valor do somador RNS, a ordem do aumento de latência por parte do somador RNS é de $1,01\%$ contra os $1,04\%$ do

somador de inteiros. Quererá isso dizer que é provável que a adição modular compense a partir de um valor de n elevado, tão elevado que perde utilidade prática para a maioria das aplicações.

Por sua vez, a área do somador modular também é sempre superior à área do somador de inteiros tendo um racio médio de crescimento de $1,2\%$ contra os $1,15\%$ do somador de inteiros.

Os valores obtidos não são propriamente os esperados de um sistema de numeração por resíduos mas o facto deste último visar a implementação em FPGA explica os resultados. Uma vez que a FPGA possui rápidas ligações dedicadas entre células vizinhas (*carry chains*) que permite a rápida propagação de *carry*, o somador convencional de inteiros implementado mediante uma descrição funcional possibilita um cálculo muito eficiente e uma interligação de blocos lógicos mínima e directa levando a resultados dificilmente superáveis.

Na tabela 4.1 encontram-se as latências de interligações de blocos lógicos registadas das estimativas do programa ISE da xilinx, esses resultados são muito significativos pois mostram que aproximadamente 47% da latência do somador RNS é devida às interligações conectando os blocos lógicos da FPGA. Conclui-se que comparando com um somador de inteiros, no que diz respeito à latência das interligações de blocos lógicos, seria impossível obter valores inferiores com um somador RNS mesmo com o paralelismo e o menor número de bits das adições.

n	*ns*	%
5	18,394	47
6	20,371	48
7	21,037	48
8	20,933	47
9	20,568	45

Tabela 4.1: Tabela das latências de interligações

Para se diminuir a área e da mesma forma a latência das interligações de blocos lógicos de um somador, poderia somar-se um operando com ele próprio. A implementação deste tipo de operador seria interessante e vantajoso pois numa sequência de adições, retirando um operando e consequentemente uma redução modular de outro operando, se obteria melhores resultados em termos de latência quando comparado com somadores de inteiros.

A figura 4.13 ilustra as áreas e as latências obtidas com este processo, ambos os somadores efectuam $\sum_0^{10} A$, $\sum_0^{15} A$ e $\sum_0^{20} A$.

Os resultados são claros, mesmo efectuando uma conta com um único operando e implementando uma sequência de adições minimizando o impacto das conversões binário/RNS e RNS/binário realizadas no início e fim respectivamente da operação, a adição modular continua com maior área e latência.

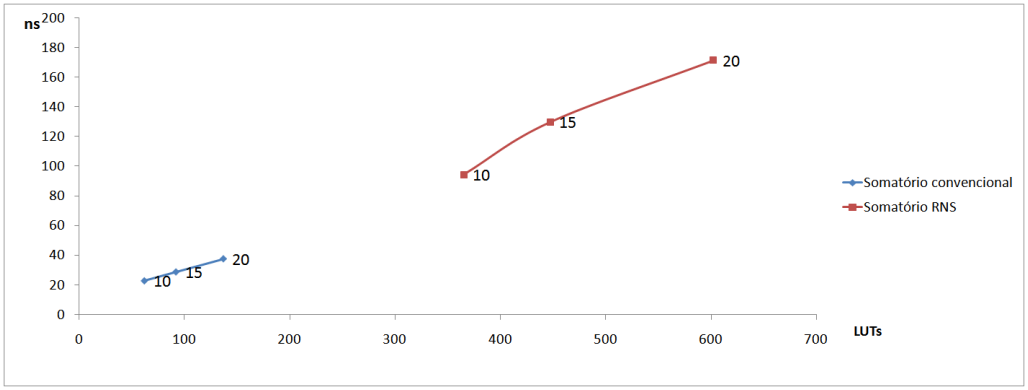

Figura 4.13: Áreas e latências para $\sum_0^{10} A$, $\sum_0^{15} A$ e $\sum_0^{20} A$

As ligações rápidas dedicadas entre as células vizinhas (*carry chains*) da FPGA não permitem a obtenção de bons resultados na utilização de um sistema de numeração por resíduos, a complexidade das interligações de blocos lógicos de um circuito combinacional descrito estruturalmente não consegue competir com a simplicidade da propagação de *carry* entre células.

O facto de se tratar de uma FPGA com essas particularidades não justifica por si só os valores obtidos não esperados e de certa forma desanimadores, outra razão diz respeito ao próprio algoritmo do sistema de numeração por resíduos utilizado em aritmética computacional, ou seja: O somador de inteiros tinha dois ou mais operandos de $m = 3n$ bits, efectuava a conta com esses operandos e obtinha um resultado de m bits. O somador RNS, por sua vez, também tinha dois ou mais operandos de m bits cada mas esses últimos eram convertidos para RNS de acordo com os módulos $\{2^n - 1, 2^n, 2^n + 1\}$ convertendo cada operando de m bits em três operandos de n bits com os quais eram realizadas as operações. O cálculo tornava-se assim mais rápido por ser executado em paralelo e com um menor número de bits. No entanto, não nos podemos esquecer que, uma vez obtidos os resíduos dos operandos, as contas a efectuar são adições modulares de n bits e não adições comuns, ou seja, para qualquer adição que se pretende efectuar, a cada uma delas está associada uma redução modular. Apesar de se tratar de números de menor dimensão, a redução não deixa de ser semelhante à exercida nos operandos de entrada. Este aspecto tem, obviamente, o seu custo em termos de área e latência e mesmo tendo uma redução modular eficaz, este factor acaba por ser crucial na comparação com um somador que utiliza as *carry chains* da FPGA.

No artigo [7] são também comparados somadores RNS com somadores convencionais. Nesse artigo, a comparação é feita para valores de n que variam de 4 a 20. Os módulos utilizados são, à semelhança desta dissertação, $\{2^n - 1, 2^n, 2^n + 1\}$ podendo os resultados ser alvo de comparação com as conclusões obtidas nesta dissertação.

Os testes efectuados no artigo revelam que o somador RNS ocupa mais recursos do que o somador convencional para todos os valores de n medidos. A latência é superior no somador RNS para quase todos os valores de n. Apenas para valores de $n = 18$, correspondendo a inteiros de 54

bits, se consegue melhorias e tempos de atraso do somador RNS menores que o atraso do somador convencional. Os resultados dessa comparação são semelhantes aos obtidos nesta dissertação diferindo apenas a partir de um valor de n igual a 18. Isto acaba por, de certa forma, comprovar o resultado da comparação obtido neste trabalho.

É de realçar que no artigo [7], os testes são realizados numa FPGA Virtex IV FX100 da Xilinx. O facto de se tratar de uma FPGA diferente da FPGA usada para testes neste trabalho pode originar a diferença de valores obtidos em ambos os trabalhos pois tanto no somador RNS como no somador convencional, os resultados da latência e da área são menores no artigo.

4.3 Comparação entre multiplicador RNS e multiplicador convencional

Seja uma operação de multiplicação de inteiros comum de dois operandos A e B e seja S o seu resultado. Tal como na operação de adição, A e B podem assumir qualquer valor e o número de bits m terá de ser previamente definido. Na multiplicação aritmética de inteiros, se A e B são ambos de m bits, S será de $2m$ bits podendo A e B ter um número de bits diferente um do outro, o número de bits de S dependerá desses valores de entrada. À semelhança do que acontece na adição RNS, também na multiplicação o resultado será limitado pela gama dinâmica dos módulos $\{2^n-1, 2^n, 2^n+1\}$ e como tal terá, com operandos de entrada de m bits cada, um resultado de m bits contra os $2m$ bits de uma multiplicação convencional de inteiros.

É importante salientar que este factor não é de todo um inconveniente pois seria possível obter uma multiplicação RNS semelhante à de uma outra multiplicação de números inteiros com entradas de $m/2$ bits para se obter um resultado com o dobro do número de bits das entradas (m bits) mas independentemente do número de bits escolhido para as entradas, as reduções modulares teriam que obrigatoriamente ser de um número de bits igual ao da conversão inversa do resultado de uma multiplicação por resíduos. Ao limitar o número de bits das entradas estaríamos, da mesma forma, a limitar as possibilidades da multiplicação por resíduos. Por este motivo, e para que a comparação seja equivalente, a multiplicação aritmética de inteiros é realizada da mesma forma que a multiplicação RNS, ou seja, A, B e S são ambos de m bits. O utilizador terá que ter em conta que, ao definir as entradas, o resultado de uma operação de $m \times m$ bits não ultrapassará os m bits.

As figuras 4.14 e 4.15 mostram as comparações da latência e de área respectivamente entre a multiplicação de inteiros e a multiplicação modular para a variação de n bits, sendo n o número de bits que determina a gama dinâmica da operação (ver tabela 3.1), ou seja, compara a multiplicação de inteiros de $3n$ bits com as multiplicações modulares de três resíduos de n bits cada executadas em paralelo e com, ainda, as conversões de binário para RNS e de RNS para binário.

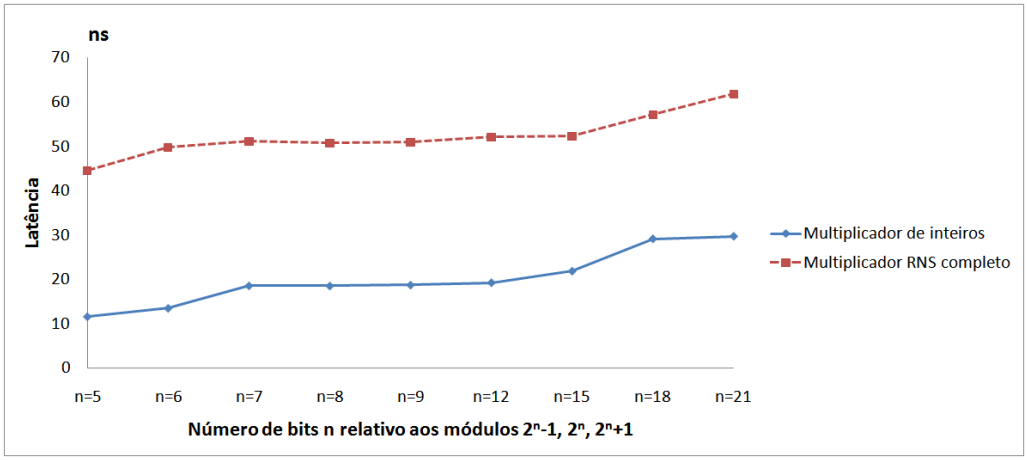

Figura 4.14: Latências de multiplicadores inteiros vs RNS (completo)

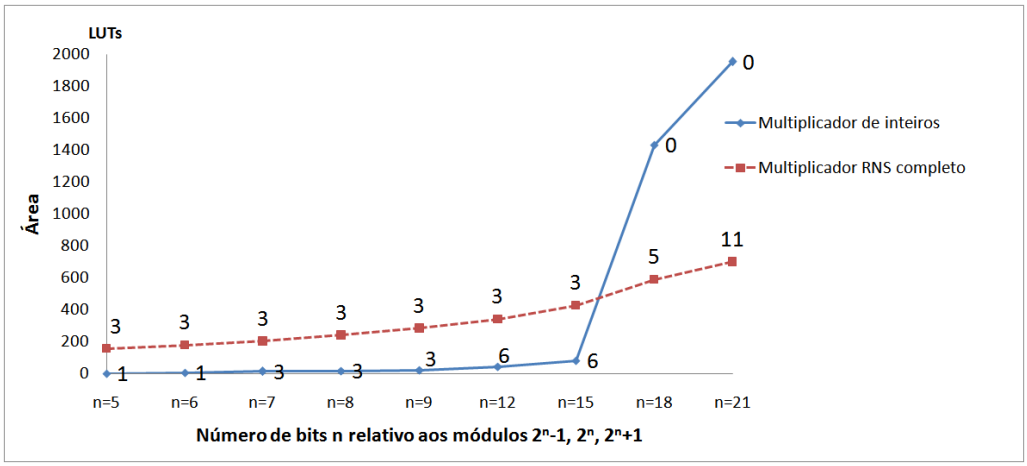

Figura 4.15: Áreas de multiplicadores inteiros vs RNS (completo)

Através dos resultados mostrados nas figuras 4.14 e 4.15 verifica-se que a latência de um multiplicador RNS é maior que a latência de um multiplicador convencional de inteiros para todo o n. O cálculo é executado em (aproximadamente) mais 30 *ns* que o cálculo do multiplicador de inteiros e essa diferença mantém-se à medida que o número de bits n aumenta.

A área, por sua vez, tem que ser analisada com cuidado redobrado pois a FPGA contém na sua arquitectura interna doze multiplicadores 18 por 18 dedicados e os resultados da comparação de áreas é divido pelo número de blocos multiplicadores utilizados mais o número de LUTs ocupados. No gráfico, o eixo vertical ilustra o número de LUTs ocupado e junto a cada valor medido encontra-se o número de multiplicadores dedicados utilizados para o cálculo.

Verifica-se que os multiplicadores RNS ocupam em média três multiplicadores dedicados da FPGA para executar as multiplicações dos três resíduos de n bits. Como se esperava, quando estes últimos atingem um número de bits maiores que 18, passam a utilizar mais multiplicadores

dedicados. Para a multiplicação convencional de inteiros, o processo de interligações de blocos lógicos é semelhante. No entanto, tratando-se de entradas de um maior número de bits (m), o número de multiplicadores dedicados utilizados cresce mais rapidamente até deixar de utilizar esses recursos passando a utilizar unicamente as LUTs da FPGA. Este aspecto faz com que, a partir desses valores de n, a área ocupada cresce subitamente.

A área do multiplicador RNS, no que diz respeito aos LUTs ocupados, aumenta quase linearmente à medida que n aumenta e mantém-se superior à área de um multiplicador de inteiros até que este último passe a utilizar as LUTs da FPGA para $n = 18$. A partir desse valor de n, a área do multiplicador RNS ocupa menos de metade dos recursos utilizados pelo multiplicador de inteiros.

As comparações foram novamente efectuadas nas figuras 4.16 e 4.17 sem as conversões binário/RNS e RNS/binário de forma a verificar se a latência e a área fossem menores mediante uma cadeia de multiplicações.

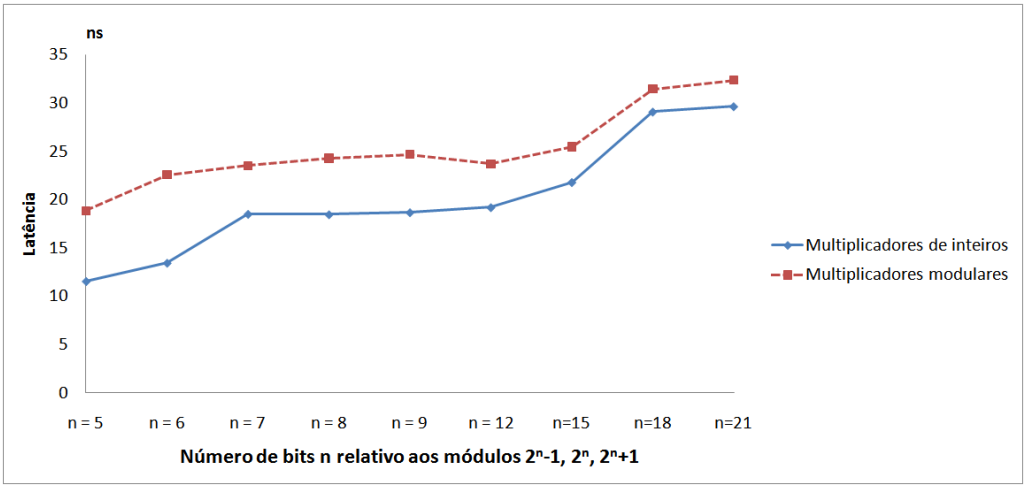

Figura 4.16: Latências de multiplicadores inteiros vs RNS

A figura 4.16 mostra que a latência de um multiplicador modular é sempre superior à latência de um multiplicador de inteiros embora o primeiro aproxima-se dos valores deste último com o aumento de n. À semelhança do que acontece no somador modular, a latência de um multiplicador de inteiro só será maior que a latência de um multiplicador RNS para um valor de n muito elevado. A área de um multiplicador RNS é ligeiramente maior que a área de um multiplicador de inteiros e só é ultrapassada por este último a partir de um valor de $n = 18$.

As conclusões que se pode tirar dos valores obtidos nestas comparações entre multiplicadores diferem um pouco daquelas retiradas dos somadores na medida em que não se obteve diferenças tão grandes como as registadas nestes últimos. A principal diferença provém da pouca utilização das ligações rápidas entre células vizinhas (*carry chains*). A FPGA utiliza os doze multiplicadores internos para realizar multiplicações e favorece assim a multiplicação de inteiros. A multiplicação

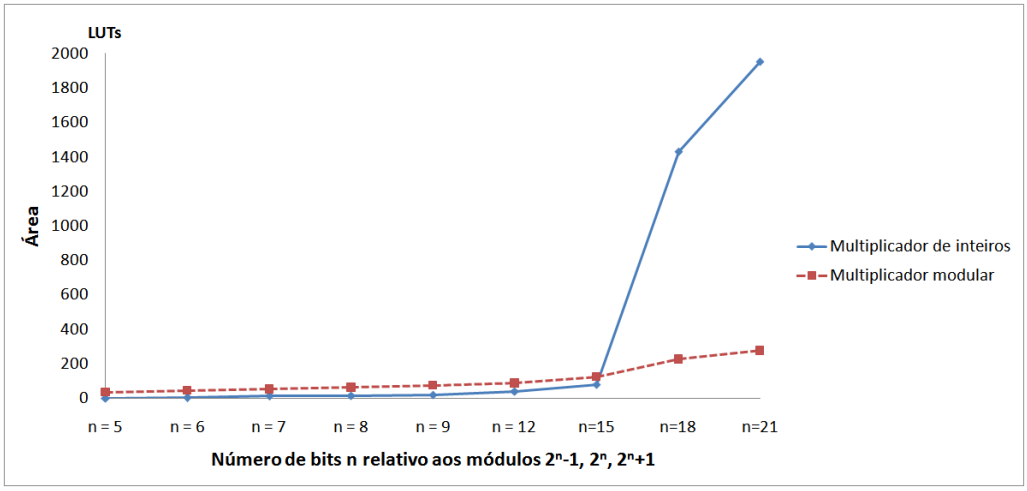

Figura 4.17: Áreas de multiplicadores inteiros vs RNS

RNS também tira proveito desses multiplicadores dedicados, efectua três vezes mais multiplicações pelos resíduos calculados mas essas são calculadas em paralelo e com um menor número de bits. A multiplicação modular deveria, teoricamente, ter maior rapidez na execução dos cálculos mas à semelhança do que acontece na adição, a cada multiplicação modular está associada uma redução modular que não permite, mesmo numa sequência de multiplicações, tirar proveito das particularidades do sistema de numeração por resíduos.

No artigo [7], os multiplicadores modulares são também comparados com multiplicadores convencionais. Nesse artigo, no que diz respeito ao número de *slices*, a área de um multiplicador RNS é maior que a área de um multiplicador convencional para todos os valores de n. No entanto, considerando apenas o número de multiplicadores dedicados utilizados, o multiplicador RNS utiliza metade dos multiplicadores dedicados que o multiplicador convencional para $n = 12, 14$ e 16. A latência do multiplicador RNS é maior para todos valores de n.

A comparação feita nesta dissertação é ligeiramente diferente do artigo anteriormente mencionado. Para valores entre $n = 12$ e $n = 15$, o operando RNS também utiliza metade dos multiplicadores dedicados utilizados pelo multiplicador convencional. A diferença reside no número de LUTs ocupados pois como já foi referido anteriormente, a partir de $n = 15$, o número de LUTs usados cresce subitamente pois o operador convencional deixa de utilizar os multiplicadores dedicados. Em ambos os casos a latência dos multiplicadores RNS é maior que a latência dos operadores convencionais. À semelhança do que aconteceu na comparação de somadores, o artigo vem reforçar as conclusões tiradas das comparações feitas anteriormente.

4.4 Arquitectura *Pipelined*

Os resultados até agora analisados mostraram que, perante as ligações rápidas de propagação de *carry* e os multiplicadores dedicados da FPGA, o sistema de numeração por resíduos não tira proveito das suas propriedades mesmo operando com números de menor dimensão.

Nesta secção vão ser comparados os operadores convencionais de adição e multiplicação com os mesmos circuitos comparados anteriormente adoptando, desta vez, uma arquitectura *pipelined*. Esta arquitectura pode ser vantajosa em termos de latência mas levará ao aumento da área total do circuito devido aos registos acrescentados para o sincronismo das várias etapas do circuito. É de realçar que, tratando-se de uma arquitectura *pipelined*, a comparação deve ser realizada com ambos os operadores RNS (somador e multiplicador) completos, ou seja, ambas as conversões de binário para RNS e de RNS para binário fazem parte do operador RNS tendo dois operandos de entrada e uma saída de $3n$ bits semelhantes aos operadores convencionais.

4.4.1 Somador RNS *pipelined*

A arquitectura *pipelined* foi realizada, tanto para a multiplicação como para o somador, conforme ilustra a figura 2.4, dividindo o circuito nos pontos mais relevantes, entre cada etapa, coincidindo com uma boa divisão de latência entre partes do circuito.

A figura 4.18 e 4.19 mostram as comparações da latência e da área respectivamente entre um somador convencional de inteiros e um somador RNS para a variação de n bits.

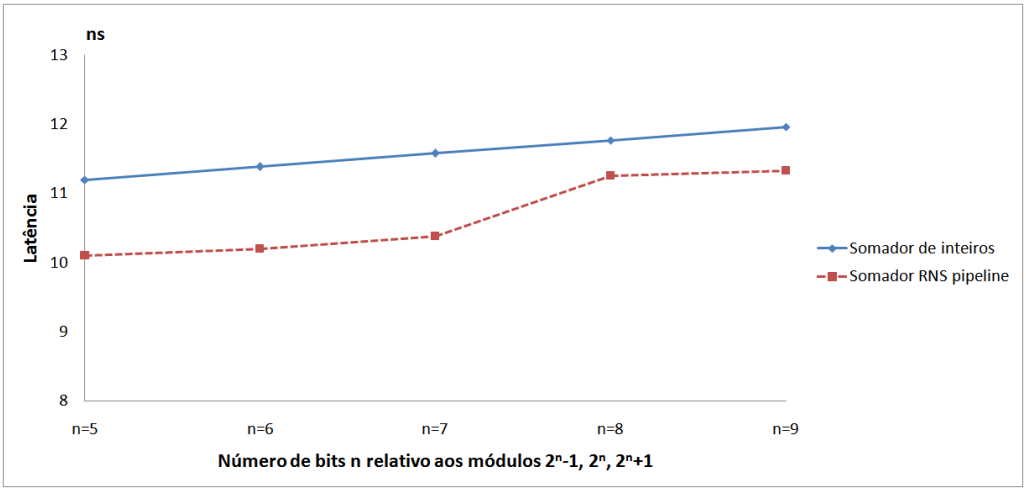

Figura 4.18: Latências de somadores inteiros vs RNS *pipelined*

Verifica-se na figura 4.18 que a arquitectura *pipelined* de um somador RNS consegue ter menor latência que um somador convencional de inteiros e menos de $1/3$ da latência total do somador RNS completo (4.9).

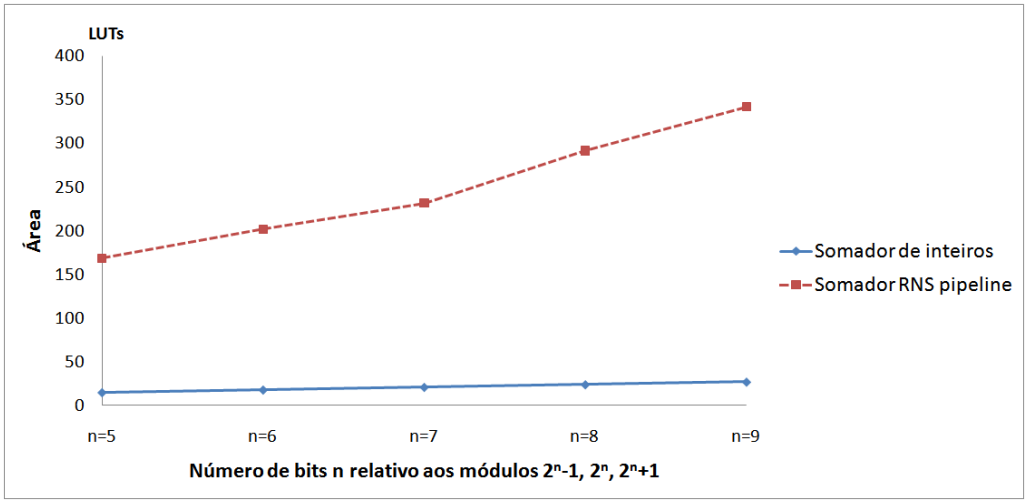

Figura 4.19: Áreas de somadores inteiros vs RNS *pipelined*

A latência do circuito somador RNS *pipelined* corresponde ao período mínimo admissível para o bom funcionamento do circuito, ou seja, corresponde ao período máximo das diferentes partes do circuito. Por exemplo, na figura 4.20 está representado o somador RNS *pipelined* dividido por registos entre cada etapa do somador com os respectivos períodos máximos de cada etapa para $n = 5$, o período mínimo para o circuito corresponde à latência provocado pelo conversor de RNS para binário pois, das três, é a etapa de maior latência.

É então possível obter um somador RNS mais rápido que um somador convencional em FPGA mas estes resultados só são válidos para uma sequência de operações uma vez que o circuito obtém um primeiro resultado após três ciclos de relógio, ou seja, $3 \times 10,172$ nano segundos. É possível obter um resultado de uma operação a cada ciclo de relógio (a cada $10,172$ ns no exemplo) após um primeiro resultado obtido em três ciclos de relógio. Se se pretender realizar uma sequência de operações, a selecção do somador mais rápido dependeria do número de operações a realizar. Para o exemplo da figura 4.20, o qual tem um período de aproximadamente 10 ns contra 11 do somador convencional de inteiros, seria necessário no mínimo 20 operações para se obter um ganho em latência com o somador RNS *pipelined* devido ao atraso provocado pelo resultado da primeira operação (30 ns).

A área por sua vez aumenta ligeiramente em relação ao somador RNS devido aos registos colocados para a divisão do circuito e continua (obviamente) maior que a área do somador convencional de inteiros.

4.4.2 Multiplicador RNS *pipelined*

No que diz respeito à multiplicação RNS *pipelined*, a estrutura é semelhante ao do somador dividindo o circuito entre as três etapas do circuito. As figuras 4.21 e 4.22 mostram as compara-

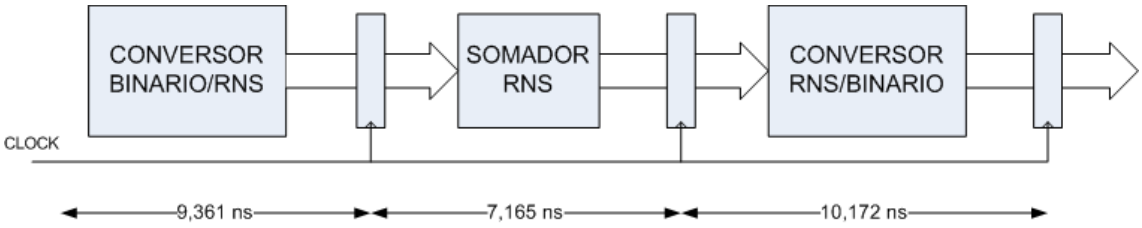

Figura 4.20: Somador RNS *pipelined*

ções da latência e da área respectivamente entre um multiplicador convencional de inteiros e um multiplicador RNS para a variação de n bits.

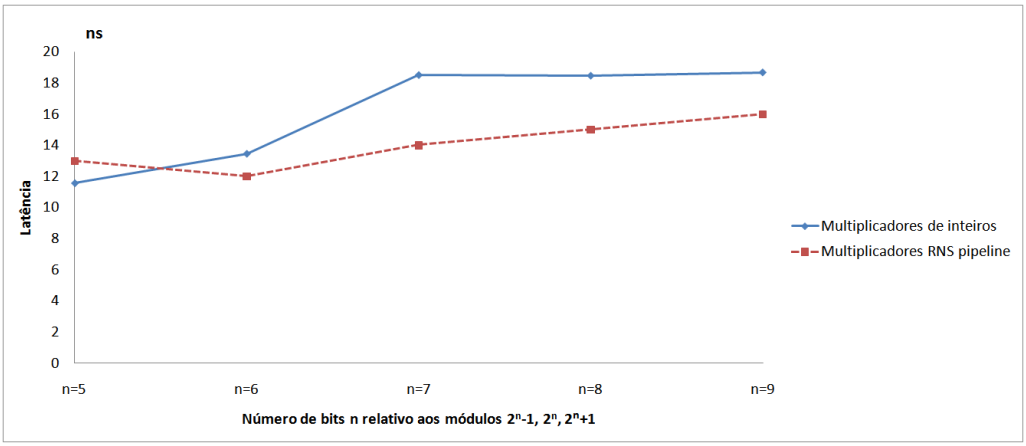

Figura 4.21: Latências de multiplicadores inteiros vs RNS *pipelined*

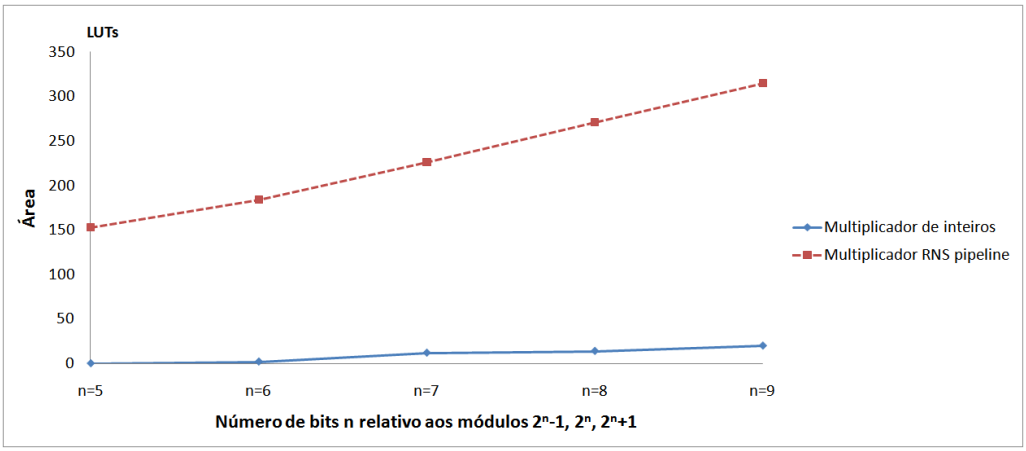

Figura 4.22: Áreas de multiplicadores inteiros vs RNS *pipelined*

Observando a figura 4.21, verifica-se que a partir de $n = 6$ também na multiplicação se consegue um cálculo mais rápido utilizando um multiplicador RNS *pipelined*. O período mínimo para

o circuito é (desta vez) obtido pelo tempo de atraso máximo do multiplicador modular uma vez que as conversões a montante e a jusante do circuito têm atraso menor. A área do circuito sofreu um ligeiro aumento e só será menor que a área de um multiplicador convencional a partir de $n = 18$ (operandos de 54 bits).

Conclui-se que a arquitectura *pipelined* dos operadores de soma ou multiplicação RNS ou ainda um circuito com ambos os operadores pode resultar em menor latência. Essas vantagens só serão obtidas se for implementado um sistema que realiza várias operações em cadeia devido ao facto do primeiro resultado ser obtido após um atraso total do circuito (três ciclos de relógio).

Quanto à área, a arquitectura *pipelined* dos operadores RNS aumenta com os registos colocados no circuito. A multiplicação RNS *pipelined* continua, no entanto, a usar o mesmo número de multiplicadores dedicados. O aumento da área provocado pelo *pipelining* não é significativo comparando com a área total do circuito.

4.4.3 Aplicação RNS em filtro FIR

Os filtros digitais são um dos grandes campos de aplicação de processamento digital de sinal que recorre a um número significativo de operações aritméticas e que podem ser garantidamente estáveis quando implementados não recursivamente. O filtro *Finite Impulse Response* (FIR) é, por consequência, um bom teste para a implementação e respectiva análise de um sistema de numeração por resíduos numa aplicação concreta.

O filtro digital do tipo FIR é definido pela equação 4.1 onde x_l é a entrada do filtro, h_k representa os seus coeficientes, N é a ordem do filtro e y_l é a saída do filtro.

$$y_l = \sum_{k=0}^{N} h_k \cdot x_{l-k} \tag{4.1}$$

A figura 4.23 ilustra o esquema de um filtro FIR implementado. Trata-se de um filtro de ordem $N = 11$ que envolve 11 operações de multiplicação e 10 operações de adição. Cada amostra de x_l é de $m = 3n = 15$ bits.

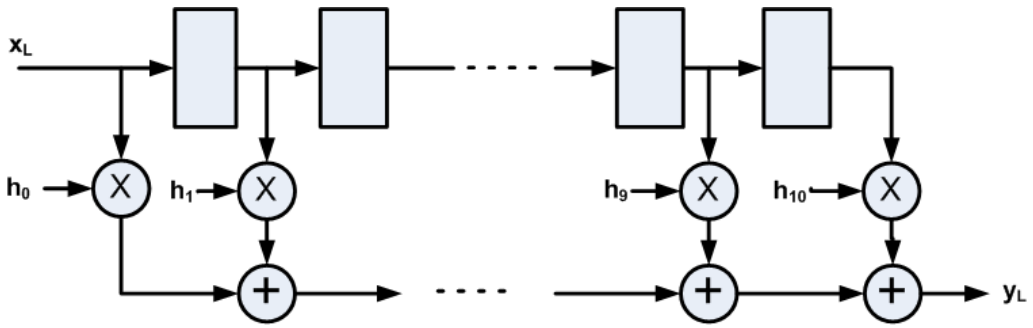

Figura 4.23: Filtro FIR

A equação 4.2 demonstra o cálculo a realizar para o filtro RNS.

$$\langle y_l \rangle_{m_i} = \left\langle \sum_{k=0}^{N} \langle h_k \cdot x_{l-k} \rangle_{m_i} \right\rangle_{m_i} \tag{4.2}$$

A implementação deste filtro FIR em RNS requer algumas modificações devido às conversões de binário para RNS e de RNS para binário. As conversões para RNS de todas as amostras de x_l são efectuadas em paralelo no início do cálculo e todos os resíduos são guardados em registos. Da mesma forma, são convertidos para RNS os coeficientes do filtro h_k. As 11 operações de multiplicação RNS são realizadas em paralelo para um menor tempo de atraso. As 10 adições RNS, por sua vez, são efectuadas seguindo uma estrutura em árvore obtendo-se os três resíduos de y_l. A saída y_l é finalmente obtida pela conversão de RNS para binário dos três resíduos. O circuito é dividido conforme ilustra a figura 4.24, os blocos "Filtro FIR modular"representam todas as operações de soma e multiplicação efectuadas para a realização do filtro.

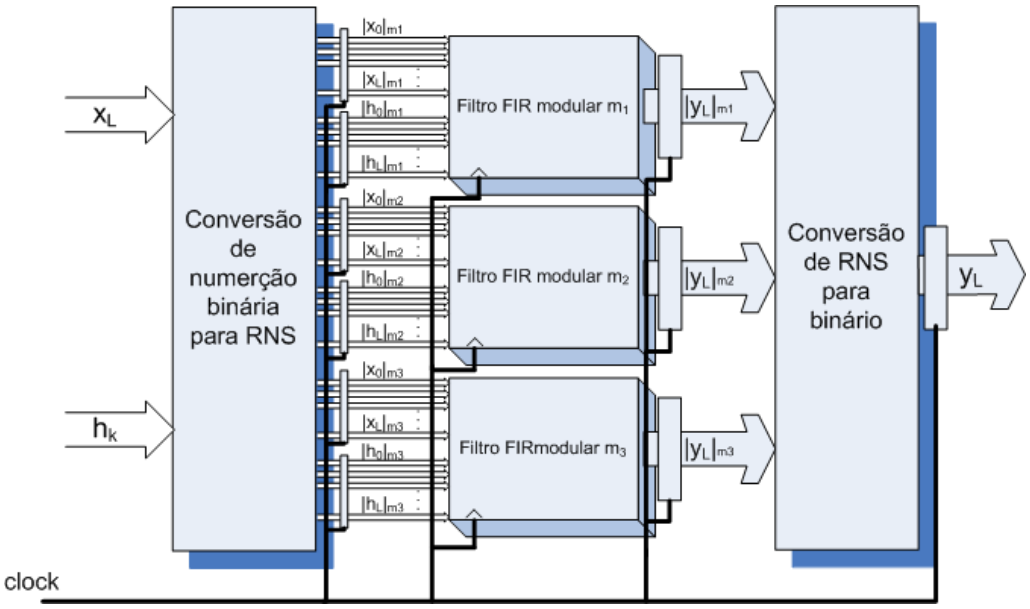

Figura 4.24: Filtro FIR RNS

Os filtros têm como entrada uma única amostra de x_l de um número de bits m que varia com n, $m = 3n$ bits. Ambos os filtros têm a respectiva saída y_l com um número de bits igual ao número de bits de x_l. Os operadores aritméticos são de m bits e n bits para os filtros de inteiros e RNS respectivamente e o número de operações é igual em ambos os casos.

A figura 4.25 mostra a comparação da área entre um filtro FIR de inteiros e um filtro FIR RNS. Verifica-se que o número de LUTs utilizadas é menor num filtro FIR RNS e que este último mantém uma menor área à medida que n aumenta. No entanto, o número de blocos multiplicadores

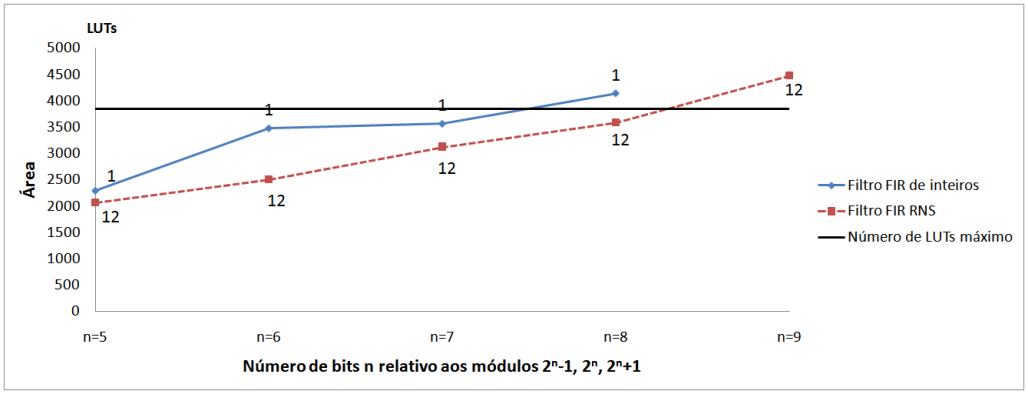

Figura 4.25: Áreas de filtros FIR inteiros vs RNS

dedicados da FPGA utilizados é maior no filtro FIR RNS pois utiliza todos os blocos disponíveis na FPGA. O filtro FIR de inteiros, por sua vez, utiliza apenas um dos blocos multiplicadores dedicados. A linha recta situada nos 3840 LUTs representa o limiar do número de LUTs máximo disponível na FPGA.

Perante estes resultados, verifica-se que a área de um filtro FIR RNS (de ordem $N = 11$) permite a sua implementação na FPGA usada até um valor de $n = 8$ (amostras de 24 bits). A implementação de um filtro FIR de inteiros na FPGA utilizada só é possível até um valor de $n = 7$ (amostras de 21 bits). O filtro FIR RNS utiliza a totalidade dos multiplicadores devido à menor dimensão dos operadores de multiplicação RNS. A área do filtro FIR RNS é menor que a área do filtro FIR de inteiros pois este último apenas utiliza, segundo o mapeamento efectuado pela ferramenta de síntese, um único bloco multiplicador dedicado tornando o circuito maior devido às várias multiplicações efectuadas.

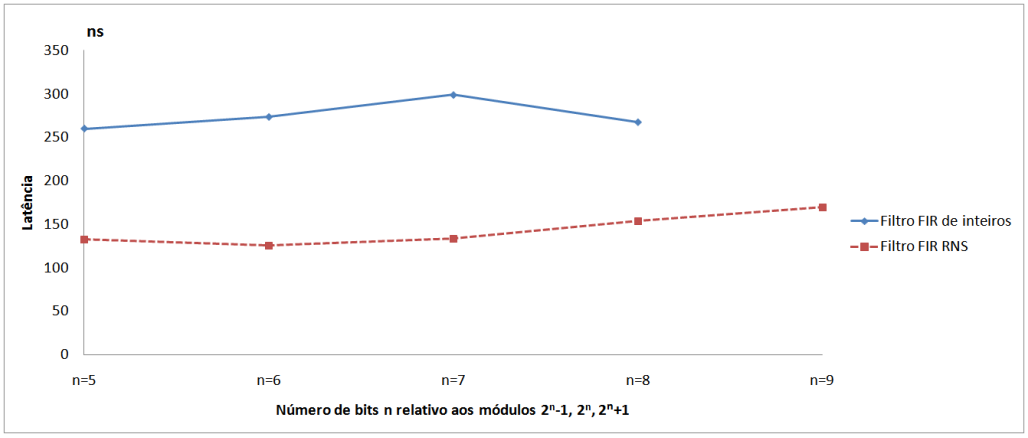

Figura 4.26: Latências de filtros FIR inteiros vs RNS

As comparações de latências são apresentadas na figura 4.26. As latências foram medidas, em ambos os casos, para valores de n que permitem a implementação do filtro na FPGA mais o

valor de n imediatamente acima do valor máximo possível. Desta forma, consegue-se obter um maior número de medidas para uma melhor percepção da comparação dos filtros.

Claramente se verifica que o filtro FIR RNS apresenta menor latência para todos os valores de n efectuando o cálculo em (aproximadamente) metade do tempo de atraso do filtro FIR de inteiros. O filtro FIR RNS tira proveito do paralelismo dos operadores RNS de menor dimensões, o sistema *pipelined* permitiu obter um período mínimo médio de aproximadamente 19 *ns* contra os 21 *ns* do filtro FIR de inteiros. É de realçar que a comparação é feita com um filtro FIR de inteiros básico (figura 4.23), seria possível implementar um filtro de inteiros com maior paralelismo diminuindo o tempo de processamento do mesmo. No entanto, com esta comparação, conclui-se que o sistema de numeração por resíduos pode ser vantajoso quando implementado de forma *pipelined* trazendo vantagens no que diz respeito à latência. A utilização de um maior número de blocos multiplicadores dedicados no filtro FIR RNS contribui significativamente para uma menor latência, o filtro FIR de inteiros não tira proveito destes blocos multiplicador e como tal vê a sua latência aumentar.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusão

Esta dissertação tinha como objectivo a caracterização de operadores modulares implementados em FPGA. Os operadores modulares são considerados como uma representação numérica promissora no domínio do processamento digital de sinal. Várias aplicações utilizam este sistema de numeração tirando proveito das suas particularidades, embora o RNS não seja extensivamente implementado em dispositivos reconfiguráveis do tipo FPGA.

As principais dificuldades encontradas na realização deste trabalho foram, numa primeira fase, a implementação combinacional dos módulos realizados que levou a um estudo exaustivo e que resultou em pouco proveito na implementação final dos operadores modulares pois os dispositivos do tipo FPGA possuem, em geral, uma rede de ligações particular que permitem optimizar a interconexão dos blocos lógicos jogando com a rede de ligações entre células CLBs vizinhas e a rede de ligações globais para a interconexão de blocos mais distantes. As FPGAs possuem ainda blocos dedicados, como os multiplicadores que permitem, mediante uma descrição comportamental do projecto, reduzir significativamente o tempo de atraso do cálculo.

Perante estas particularidades, os benefícios do sistema de numeração por resíduos acabam por ficar limitados pois a cada operador RNS está associado uma redução modular essencial para evitar o risco de um overflow não detectado durante o processo da operação que poderia levar a resultados finais errados. Este é sem dúvida o maior inconveniente do sistema de numeração por resíduos a par da necessidade dos conversores de binário para RNS e de RNS para binário.

O RNS efectua as operações com um menor número de bits utilizando, tal como nas operações convencionais, as ligações rápidas de células vizinhas sendo então este processo mais rápido que a operação convencional. No entanto, a redução modular, efectuada após a operação, utiliza a rede de ligações globais da FPGA resultando num maior tempo de atraso para a operação modular.

Por exemplo se, na operação da adição, k for a diferença dos números de bits entre um operando inteiro, de $m = 3n$ bits, e os respectivos resíduos, de n bits, então o número de vezes que é

propagado *carry* numa adição de um sistema de numeração por resíduos é reduzido $k = m - n = 2n$ vezes. De facto a diferença é considerável pois o RNS opera com apenas $1/3$ do número de bits de um operando inteiro e o tempo de atraso deveria ser, sendo directamente proporcional, $1/3$ da latência da operação de adição com inteiros. No entanto, a redução modular do resultado da operação RNS, sendo mais complexa que uma simples propagação de *carry*, tem um tempo de propagação maior que o ganho em latência obtido no RNS que corresponde à adição e propagação de *carry* de k bits.

Na operação de multiplicação, o RNS só utiliza mais recursos para operandos inteiros menores ou iguais a 18 bits pois a multiplicação de inteiros utiliza um multiplicador dedicado de 18 por 18 da FPGA. Para esses operandos, o RNS utiliza, de forma parcial, três multiplicadores dedicados para os três resíduos do operando inteiro. Quando o número de bits do operando inteiro aumenta, o número de blocos multiplicadores dedicados utilizados também aumenta. No RNS, o número de multiplicadores dedicados utilizados mantém-se até atingir resíduos de 18 bits correspondendo a operandos inteiros de 54 bits. A latência na multiplicação RNS poderia assim ser menor pois além de ocupar menos blocos multiplicadores dedicados, o cálculo é ainda efectuado em paralelo. A latência da multiplicação RNS resumia-se então à latência da multiplicação do resíduo m_3 sendo, dos três resíduos, o único operando RNS com $n+1$ bits contra os $m = 3n$ bits de uma multiplicação de inteiros. No entanto, tal como na operação RNS da adição, a redução modular do resultado da operação de multiplicação RNS tem um tempo de propagação maior que uma multiplicação de k bits efectuada por blocos multiplicadores dedicados.

Assim sendo, os operadores RNS não permitem a diminuição da latência de um operador aritmético. Mesmo aplicando uma sequência de operações, reduzindo assim o impacto dos tempos de atraso dos conversores de binário para RNS e de RNS para binário, a latência permanece maior que o tempo de atraso de um operador convencional.

Concluiu-se que a única forma de se optimizar a latência num sistema de numeração por resíduos passa pela implementação de uma estrutura *pipelined*. O contributo de um sistema de numeração por resíduos *pipelined* fica, no entanto, dependente do número de operações a realizar obtendo-se um ganho em latência após umas dezenas de operações em sequência. O sistema de numeração por resíduos apresenta, no entanto, alguns aspectos positivos no que diz respeito à operação de multiplicação. Pois, embora o número de LUTs ocupados seja maior, o número de blocos multiplicadores dedicados usados na FPGA pode ser reduzido para metade.

Mediante uma implementação *pipelined* de um filtro digital do tipo FIR, verificou-se que, quando comparado com um filtro FIR de inteiros, tanto a área como a latência podem ser menores num RNS. Numa aplicação onde várias operações aritméticas são efectuadas, um sistema de numeração por resíduos é uma boa opção quando, para um número elevado de bits, se pretender tirar partido dos blocos dedicados da FPGA. Estes últimos deixam de ser utilizados numa aplicação convencional de inteiros a partir de um número elevado de operações de grandes inteiros e

o mapeamento é feito através das interligações globais da FPGA. Este aspecto faz com que tanto a latência como a área aumentam significativamente. Para um número de bits elevados, o RNS consegue tirar proveito do facto de efectuar operações de menor dimensões continuando a utilizar os blocos multiplicadores dedicados quando a multiplicação convencional já não o permite. Outra vantagem ligada ao sistema de numeração por resíduos reside na possibilidade de criar circuitos de grandes dimensões na FPGA que de outra forma não seria possível dada a elevada área do circuito quando a aplicação é implementada com uma numeração convencional. Á medida que o número de bits aumenta numa aplicação, este aspecto pode revelar-se cada vez mais importante e o RNS pode tornar-se, cada vez mais, uma melhor opção.

5.2 Trabalho Futuro

Durante a realização deste trabalho foram identificados diversos algoritmos para a aritmética modular. Um estudo mais aprofundado da aritmética por resíduos poderia levar à implementação de operadores modulares mais eficientes conduzindo à obtenção de melhores resultados tanto nos tempos de atraso como nos recursos ocupados.

A selecção dos módulos para este trabalho foi focada na documentação existente para os módulos de potências de dois, a caracterização feita mediante módulos diferentes seria interessante na medida em que permitiria obter uma gama dinâmica diferente que poderia ser mais adequada para determinado tipo de aplicações tais como os filtros *Finite Impulse Response* (FIR).

Um outro trabalho de interesse seria a realização de uma biblioteca RNS no sentido de facilitar a implementação de operadores modulares facilitando, da mesma forma, a implementação de um sistema de numeração por resíduos em aplicações mais abrangentes onde o RNS fosse vantajoso.

Referências

[1] Xilinx. Spartan-3 FPGA family data sheet. http://www.xilinx.com/.

[2] Ricardo Chaves e Leonel Sousa. $2^n+1, 2^{n+k}, 2^n-1$: A new RNS moduli set extension. Em *DSD '04: Proceedings of the Digital System Design, EUROMICRO Systems*, páginas 210–217, Washington, DC, USA, 2004. IEEE Computer Society.

[3] Behrooz Parhami. *Computer arithmetic algorithms and hardware designs*. Oxford University Press, Primeira edição, 2000.

[4] Mentor Graphics. Modelsim. http://www.mentor.com/.

[5] Ricardo Chaves. Processamento de Sinal e Criptografia baseados em Sistemas de Numeração por Resíduos. Tese de mestrado, INSTITUTO SUPERIOR TÉCNICO, Lisboa, 2001.

[6] Alain Guyot. Architecture des equipements opérateurs arithmétiques. Disponivel em http://users-tima.imag.fr/cis/guyot/Cours/Oparithm/, acedido a última vez em 28 de Junho de 2010.

[7] Haohuan Fu, Oskar Mencer, e Wayne Luk. Optimizing residue arithmetic on FPGAs. *Proceedence of the international conference on Field-Programmable Technologie*, Dezembro 2008.

[8] Khalid M. Ibrahim e Salam N. Saloum. An efficient residue to binary converter design. *IEEE Transactions on Circuits and Systems*, Vol. 35 Number 9 Setembro 1988.

[9] Zhongde Wang, G. A. Jullien, e W. C. Miller. An algorithm for multiplication modulo $(2^n - 1)$. Em *Proceedence of the Asilomar Conference on Sinals, Systems and Computers*, Out 1995.

[10] Reto Zimmermann. Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication. Em *ARITH '99: Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, página 158, Washington, DC, USA, 1999. IEEE Computer Society.

[11] D. Soudris, K. Sgouropoulos, K. Tatas, V. Padidis, e A. Thanailakis. A methodology for implementing FIR filters and CAD tool development for designing RNS-based. Em *Proceedence of the IEEE Systems, International Symposium on Circuits System*, páginas 129–132, Maio 2003.

[12] Lampros Kalampoukas, Dimitris Nikolos, Costas Efstathiou, Haridimos T. Vergos, e John Kalamatianos. High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders. *IEEE Transactions on Computers*, 49:673–680, 2000.

[13] F. J. Taylor. Residue Arithmetic, A Tutorial with Examples. *Computer*, 17(5):50–62, 1984.

[14] Loonawat G. e Siferd R.E. FPGA Implementation of a FIR Filter using Residue Arithmetic. *Aerospace and Electronics Conference, 1996. NAECON 1996., Proceedings of the IEEE 1996 National*, 1:286–290, 1996.

[15] Marco Re, Alberto Nannarelli, Gian Carlo Cardarilli, e Roberto Lojacono. Fpga Realization of RNS to Binary SignedCconversion Architecture. Em *Proc. IEEE International Symposium on Circuits and Systems (ISCAS*, páginas 350–353, 2001.