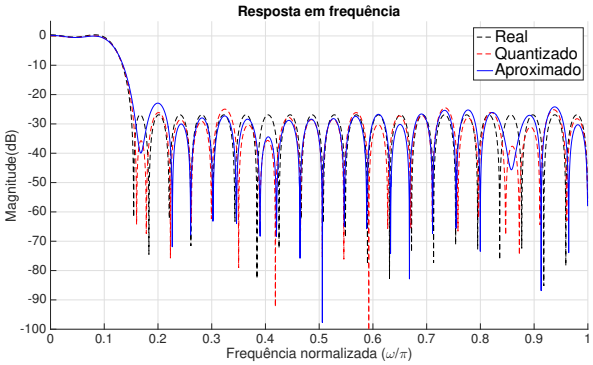
TÉCNICO LISBOA



Técnicas de Computação Aproximada para Implementação de Filtros FIR

Tiago Galvão Viegas

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Prof. Paulo Ferreira Godinho Flores

Júri

Presidente: Prof. Gonçalo Nuno Gomes Tavares
Orientador: Prof. Paulo Ferreira Godinho Flores
Vogal: Prof. José Carlos Alves Pereira Monteiro

Novembro 2016

Agradecimentos

Agradeço à minha família por me ter apoiado e motivado durante todo o meu percurso académico. Agradeço também ao Prof. Paulo Flores por me ter orientado e ajudado durante todo o processo da elaboração desta dissertação.

Especial agradecimentos à minha namorada Leonor por me ter motivado e incentivado na realização desta dissertação.

Resumo

A computação aproximada é uma técnica de computação que produz resultados não exatos, mas que são suficientemente aproximados para o fim a que se destinam. Aplicações das áreas de áudio e vídeo permitem que sejam usadas aproximações suficientemente boas dado que para o ser humano é difícil de distingui-las do resultado exato ou, mesmo com ruído, consegue extrair informação útil dos estímulos auditivos ou visuais resultantes. A computação aproximada tem como principal objetivo a troca da exatidão dos resultados por poupanças de recursos, tais como energia, tempo de computação ou área do circuito.

Nesta dissertação são apresentadas algumas técnicas existentes de computação aproximada que são resultantes de alterações do sistema em três níveis: nível físico, nível lógico e nível arquitetural. Para o projeto de filtros FIR em hardware com arquiteturas baseadas em somas/subtrações e deslocamentos são propostos e desenvolvidos algoritmos para dois métodos de computação aproximada, que visam reduzir os seus custos de implementação (área do circuito) e operação (potência). No primeiro método a redução dos custos é obtida retirando somadores de uma arquitetura base e substituindo-os por outras somas parciais existentes. Com este método obtiveram-se ganhos na área e potência do circuito que vão até 33% e 22,6%, respetivamente, com uma relação sinal-ruído de 30 dB.

No segundo método os recursos são poupados pela utilização de somadores aproximados, que introduzem erros em k bits. Neste método foram conseguidas reduções na área de até 56,7% e na potência de 45,6%, para uma relação sinal-ruído de 60 dB.

Palavras-chave: Computação aproximada, Filtros de resposta ao impulso finita (FIR), Eficiência energética, Somadores aproximados.

Abstract

Approximate computing is a computation technique which produces non exact results, but that are sufficiently approximated for the purpose they were intended. Applications in the areas of audio or video allow good enough approximations to be used, since for the human being can not distinguish them from the exact result or it can extract useful information from the resulting noisy auditory or visual stimuli. Approximate computing has as its main objective to trade the accuracy of the output for savings in resources, such as energy, computation time or circuit area.

In this dissertation some existing techniques of approximate computing are presented, which result of alterations of the system at three levels: physical level, logic level and architectural level. For the project of FIR filters with shift-adds architectures two different approximate computing methods and algorithms are proposed and developed, that aim to reduce the FIR filter's implementation (circuit area) and operation (power) costs. In the first method the cost reduction is obtained by removing adders from a base architecture and substituting them by other existing partial sums. With this method gains in area and power up to 33% and 22.6%, respectively, were obtained with a signal-to-noise ratio of 30 dB. In the second method resources are saved by utilizing approximate adders, which introduce errors in k bits. In this method reductions in area up to 56.7% and in power up to 45.6% were obtained, for a signal-to-noise ration of 60 dB.

Keywords: Approximate computing, Finite-impulse response (FIR) filters, Energy efficiency, Approximate adders.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Abreviações

SNR	-	Relação sinal-ruído.
FIR	-	Filtro de resposta ao impulso finita.
IIR	-	Filtro de resposta ao impulso infinita.
FFT	-	Transformada de Fourier rápida.
CMOS	-	Semicondutor metal-óxido complementar.
RISC	-	Computação com conjunto de instruções reduzida.
DSP	-	Processamento digital de sinal.
VOS	-	Sobre escalonamento de tensão.
RCA	-	Somador *Ripple-Carry*.
FA	-	Somador completo.
MCM	-	Multiplicação por múltiplas constantes.
PSNR	-	Relação sinal-ruído de pico.

Capítulo 1

Introdução

Em algumas aplicações a avaliação do resultado que estas geram é realizada de forma qualitativa. Os sistemas que executam estas aplicações podem ter então alguma flexibilidade na exatidão com que geram os resultados, sendo por vezes apenas necessário um resultado suficientemente aproximado.

A computação aproximada é uma área de investigação, que engloba todas as técnicas que trocam a exatidão dos resultados gerados por um sistema de computação, pela redução dos recursos necessários, como a energia ou área, à realização desse sistema [1].

A computação aproximada tem vindo a ser abordada em diferentes níveis de abstração, desde das propriedades físicas dos transístores [2], às funções lógicas de operadores aritméticos [3, 4, 5, 6, 7, 8, 9, 10] às alterações nas arquiteturas dos sistemas computacionais [11, 12], tendo obtido ganhos em termos de eficiência energética ou reduções na área dos circuitos implementados para algumas aplicações em particular.

1.1 Motivação

O aumento da densidade de transístores em cada nova tecnologia CMOS (*"Complementary Metal-Oxide-Semiconductor"*), leva a um aumento no consumo de energia dos circuitos, o que fez com que um dos objetivos principais no projeto de circuitos digitais seja a melhor eficiência energética [10]. Numa grande parte de aplicações em multimédia, por exemplo no processamento de imagem ou áudio, o resultado final pode conter pequenos erros sem que a sua qualidade seja muito afetada. Nas aplicações nas áreas do áudio ou video, isto deve-se ao facto de que o ser humano é tolerante ao erro e consegue obter informações úteis de estímulos (visuais ou auditivos) com algum ruído [3], e portanto não é necessário que, neste tipo de aplicações, a saída esteja sempre numericamente correta, sendo apenas necessário um valor suficientemente aproximado do valor exato.

A resiliência ao erro pode ser explorada usando técnicas de computação aproximada, que tornam mais flexível a relação entre a implementação e a especificação de um sistema de computação permitindo trocar a exatidão dos resultados numéricos pela redução no consumo energético, na área utilizada e/ou no atraso [13]. Nem todas as aplicações podem usufruir das vantagens da computação aproxi-

mada. No caso de um processador RISC (*"Reduced Instruction Set Computing"*) embebido, os blocos aritméticos apenas consomem 6% da energia total consumida pelo processador [14]. Isto significa que usar computação aproximada neste caso não resultaria em grandes ganhos na eficiência energética comparando com o processador total. Outro motivo para qual a computação aproximada não é adequada para processadores genéricos ou programáveis é que estes processadores são desenhados para executar aplicações genéricas e não possuem qualquer especialização para uma aplicação específica. Portanto, pode existir aplicações que não são tolerantes ao erro introduzido pela computação aproximada [3]. No entanto a computação aproximada pode ter interesse em ser utilizada em aplicações tolerantes ao erro e que usam intensamente operações aritméticas, tal como blocos DSP.

Os blocos DSP implementam funções de processamento digital de sinal, como por exemplo a filtragem digital, por filtros FIR (*"Finite Impulse Response"*) ou IIR (*"Infinite Impulse Response"*), e a transformada rápida de Fourier, FFT (*"Fast Fourier Transform"*). As operações aritméticas fundamentais nestas funções são a soma e a multiplicação. Ao usar a computação aproximada para estas operações básicas pode ser possível obter blocos DSP com um menor consumo energético ou área, sabendo que estes têm algum erro na sua saída.

Por haver um erro resultante de se realizar computações aproximadas, este deverá ser quantificado, tanto quanto possível, e os circuitos resultantes devem ser avaliados no contexto da aplicação a que se destinam.

1.2 Objetivos

O objetivo desta tese é estudar e implementar técnicas de computação aproximada que possam ser usadas para implementar em *hardware* blocos DSP, mais especificamente filtros FIR com arquiteturas com apenas somas e deslocamentos, que sejam vantajosos em termos de área ou energia, quando comparados ao métodos convencionais de implementar estes mesmo blocos. O erro introduzido no resultado por estas técnicas é também caracterizado.

Foram desenvolvidas dois métodos de computação aproximada para a implementação de filtros FIR em *hardware*, que podem ser brevemente descritos da seguinte forma:

- **Método de computação aproximada ao nível arquitetural** - Este método reduz o número de somadores numa arquitetura de um filtro FIR já existente, apenas pela remoção de alguns somadores, refazendo depois as ligações necessárias para obter a melhor aproximação do valor calculado pelos somadores retirados.

- **Método de computação aproximada ao nível lógico** - Este método reduz a área necessária à implementação de um filtro FIR, substituindo alguns somadores, numa arquitetura de um filtro FIR já existente, por somadores mais simples mas que produzem um cálculo aproximado.

1.3 Organização da Tese

O resto desta dissertação está organizado da seguinte forma: No Capítulo 2 é introduzida a computação aproximada e são apresentados alguns dos métodos existentes na literatura desta área de investigação, sendo estes divididos em três categorias de aproximações: nível físico, lógico e arquitetural.

No Capítulo 3 são abordadas as diferentes etapas na realização de filtros FIR, desde as suas especificações até à sua implementação em *hardware*. Em particular são abordadas as arquiteturas baseadas em somas/subtrações e deslocamentos.

No Capítulo 4 é proposto o método de computação aproximada ao nível arquitetural para a realização de filtros FIR em *hardware*. A implementação deste método é explicada, apresentando as opções tomadas, e de seguida são apresentados os resultados experimentais para 10 filtros FIR.

No Capítulo 5 é introduzido o método de computação aproximada ao nível lógico, também para a implementação de filtros FIR em *hardware*. Este método é baseado num somador/subtrator modificado cujos os erros são caracterizados. Um algoritmo para a utilização deste operadores aproximados foi desenvolvido para a implementação de filtros FIR sendo os resultados avaliados num conjunto de filtros.

O Capítulo 6 apresenta as conclusões desta dissertação, propondo o trabalho futuro a realizar dentro desta área de investigação.

Capítulo 2

Computação Aproximada

A computação aproximada (ou inexata) é uma área que abrange uma grande variedade de atividades de investigação, que têm como objetivo comum encontrar soluções, que permitam sistemas de computação trocar recursos, como por exemplo área ou energia, pela qualidade do resultado computado [1].

Desde o aparecimento desta área que vários métodos de aproximação em diferentes níveis de abstração têm sido propostos [13, 1], como por exemplo ao nível físico, nível lógico e nível arquitetural.

Os métodos ao nível físico referem-se a métodos que alteram as condições de operação de um circuito, como por exemplo a tensão de operação, de modo a reduzir o consumo de energia. Métodos ao nível lógico incluem técnicas que alteram as funções lógicas de um bloco computacional, como por exemplo um somador ou multiplicador, de forma a reduzir a complexidade do circuito, e consequentemente a área de implementação, ou reduzir o consumo de energia. A alteração das funções lógicas pode ser realizada na implementação por transístores ou ao nível das portas lógicas. Alterações nas implementações por transístores, como por exemplo remover transístores, modificam as funções lógicas realizadas pelo circuito e portanto estas técnicas estão incluídas no nível lógico. Por fim os métodos ao nível arquitetural referem-se a métodos que alteram de alguma forma a arquitetura de um sistema, normalmente pela remoção de componentes e ligações, de forma a poupar recursos.

Nas próximas secções são apresentados algumas técnicas para a computação aproximada presentes na literatura desta área de investigação.

2.1 Técnicas de computação aproximada ao nível físico

As primeiras técnicas de computação aproximada, ou seja, que introduzem erros em troca de poupança de recursos, que foram desenvolvidas utilizam o método de sobre escalonamento da tensão, VOS (*"Voltage overscalling"*) [1].

Ao usar o método de VOS os elementos computacionais são operados a uma tensão inferior à tensão mínima que garante a correta operação. De forma a explicar este método tomemos o exemplo para um somador *ripple-carry* (RCA) de 8 bits ilustrado na Figura 2.1 [2].

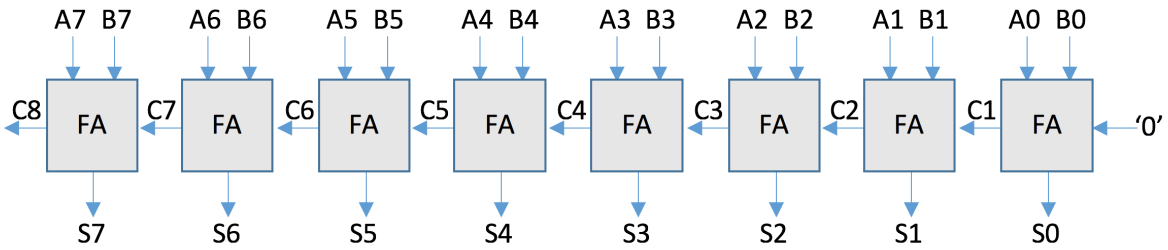


Figura 2.1: Somador *ripple-carry* de 8 bits.

Consideremos que $A = 01001010$ e $B = 01000110$, com o bit mais significativo à esquerda. O cálculo do segundo bit do resultado da soma (S) de A e B gera um *carry* que é propagado para o bit seguinte. Por sua vez o cálculo do terceiro bit gera um *carry* que quando somado no quarto bit gera outro *carry* que faz com que o quinto bit do resultado fique com o valor "1". Chamemos a isto uma cadeia de *carry* de comprimento 3 com origem na posição 2. Este processo está esquematizado na Figura 2.2 onde os *carry* gerados estão representados por C e a cadeia de *carry* está indicada.

C		01001110 0
A		01001010
B	+	01000110
S		10010000

Figura 2.2: Resultado da soma $S = A + B$ e sequência de *carry* gerada (C).

Se o tempo de atraso do cálculo de um *carry* mais o seu tempo de propagação para o próximo bit mais significativo for designado por d, no pior caso, isto é, se existir uma cadeia de *carry* de comprimento 8 e origem na posição 1, o tempo máximo de atraso deste somador é $D = 8d$. Este tempo máximo define a frequência máxima, $f = \frac{1}{D}$, a que este somador pode operar de modo a garantir resultados corretos. A frequência máxima define a tensão mínima de operação de cada somador completo, FA (*"Full Adder"*), visto que em circuitos CMOS $V \propto f$. A tensão e a frequência definem a energia consumida pelo circuito, pois $E \propto V^2 f$ [2].

Se considerarmos que a tensão a qual o circuito opera diminui para metade, ou seja $V' = \frac{1}{2}V$ e consequentemente $d' = 2d$, mas a frequência de operação é a mesma, o comprimento máximo da cadeia de *carry*, que garante o resultado correto, é metade em relação ao caso anterior, isto é $\frac{8}{2} = 4$. Para o caso da soma dos números A e B do exemplo anterior, o resultado estaria correto, mas a energia despendida seria quatro vezes menor visto que $E' \propto (\frac{V}{2})^2 = \frac{V^2}{4}$, logo $E' = \frac{E}{4}$.

Se usarmos operandos em que a sua operação de soma geraria uma cadeia de *carry* com comprimento superior ou igual a 4, o resultado obtido estaria incorreto. Um exemplo para esta situação está exemplificado na Figura 2.3.

	V	V'
C	011111000	000111000
A	01010100	01010100
B	+ 00101100	+ 00101100
S	10000000	01000000

Figura 2.3: Resultado produzido pelo RCA com tensões de alimentação V e $V' = V/2$.

Neste exemplo a magnitude do erro (valor absoluto da diferença entre o resultado impreciso e o resultado correto) é 64. Mas no caso de se ter $A = 00010101$ e $B = 00001011$, apesar do comprimento da cadeia de *carry* gerada ser igual ao do caso anterior (5), como a cadeia tem origem numa posição inferior, a magnitude do erro seria menor (16), mesmo que as condições de operação do circuito, tensão e frequência, fossem as mesmas. Perante este resultado, podemos inferir que erros em bits mais significativos afetam mais a qualidade (maior magnitude do erro) da solução do que o mesmo número de erros em bits menos significativos.

Quando a tensão de alimentação de todo o circuito, neste caso em todos os FA, é diminuída de igual forma, o método usado chama-se VOS uniforme. De forma a minimizar a magnitude do erro introduzido no resultado, outros métodos chamados de VOS não-uniforme surgiram e consistem em alimentar os circuitos que calculam, ou processam, os bits mais significativos do resultado a tensões maiores, de forma a que os erros apenas ocorram em bits menos significativos. Estes métodos estão limitados ao número de tensões de alimentação disponíveis quando estes circuitos são implementados usando a tecnologia CMOS [2].

Para o caso do método VOS uniforme foram conseguidas reduções de 21,7% no consumo energético e SNR de 5,79 dB num circuito que calcula a transformada de Fourier discreta quando comparado com o circuito alimentado de forma convencional a funcionar com a mesma frequência de operação. O método de VOS não uniforme obtém reduções no consumo energético de 27,2% quando comparado com o método de VOS uniforme e SNR de 19,63 dB [2].

2.2 Técnicas de computação aproximada ao nível lógico

A computação aproximada ao nível lógico refere-se a técnicas que modificam as funções lógicas de um circuito de forma a reduzir a sua complexidade, e por sua vez obter reduções em termos de área, ou consumo energético.

Ao modificar as funções lógicas dos circuitos estamos a permitir que erros sejam introduzidos, ou seja, o resultado na saída não estará correto para todas as combinações dos sinais de entrada. No entanto, se apenas existirem erros para algumas combinações dos sinais de entrada e a redução na complexidade ou no consumo energético for significativa, estes circuitos poderão ser vantajosos para aplicações resilientes ao erro.

Nas subsecções que se seguem são apresentados arquiteturas de elementos aritméticos que usam a computação aproximada. Na subsecção 2.2.1 é apresentado um multiplicador aproximado, obtido através da modificação das funções lógicas, usando o mapa de Karnaugh. Na subsecção 2.2.2 são apresentados diferentes somadores aproximados, uns obtidos pela remoção de transístores de implementações existentes e outros pela a alteração de somadores convencionais ao nível das portas lógicas.

2.2.1 Multiplicador aproximado

Um multiplicador aproximado de 2x2 bits foi obtido modificando a função lógica de um multiplicador exato [4]. A única entrada do mapa de Karnaugh do multiplicador que é modificada é a referente à multiplicação de 11_2 com 11_2, que em vez de ser 1001_2 (9_{10}) é 111_2 (7_{10}), como ilustrado na Figura 2.4. Esta alteração permite aumentar a partilha de termos da função e reduzir o número de portas lógicas necessárias à implementação do multiplicador. A Figura 2.5 apresenta as implementações por portas lógicas de cada um dos multiplicadores: exato e inexato.

A_1A_0 \ B_1B_0	00	01	11	10
00	000	000	000	000
01	000	000	011	010
11	000	011	**111**	110
10	000	010	110	100

Figura 2.4: Mapa de Karnaugh do multiplicador 2x2 inexato.

A magnitude do erro máxima deste multiplicador é de 22,22% ($\frac{|7-9|}{9}$), com uma probabilidade de $\frac{1}{16}$ (apenas uma entrada, do mapa de Karnaugh, em dezasseis está incorreta). De forma a implementar multiplicadores maiores, os blocos de multiplicadores 2x2 inexatos são usados para produzir produtos parciais, que depois são deslocados e somados usando somadores exatos. Um exemplo de um multiplicador 4x4, usando multiplicadores 2x2 inexatos, é apresentado na Figura 2.6.

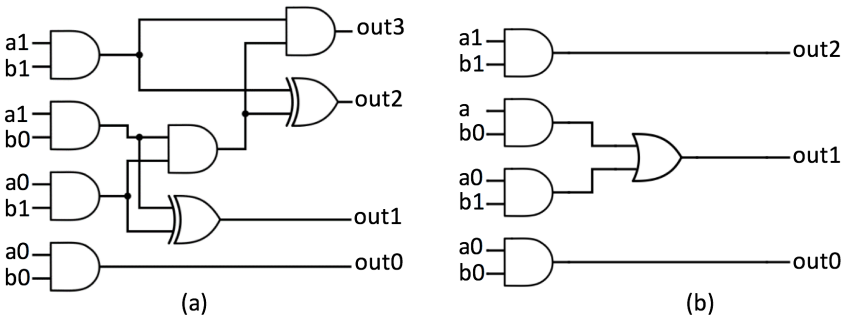


Figura 2.5: Multiplicadores 2x2 exato (a) e inexato (b).

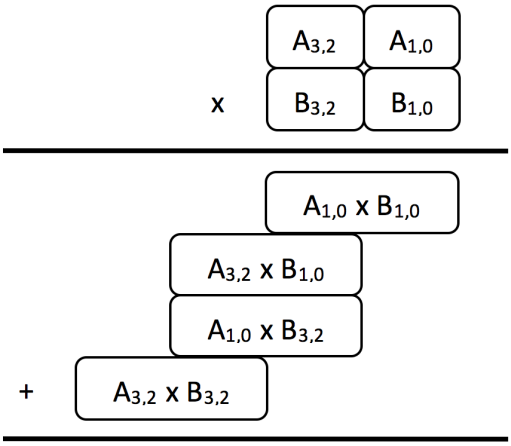


Figura 2.6: Multiplicador 4x4 inexato.

Usando modelos de simulação em *C++* para multiplicadores inexatos de diferentes tamanhos, mostrou-se que o erro relativo máximo é constante (22,22%). Verificou-se também que a probabilidade de erro aumenta com o número de bits dos operandos, como seria de esperar. O erro relativo médio obtido aumenta também com o número de bits dos operandos, no entanto quase que satura entre 3,3% e 3,35% [4].

De modo a comparar o consumo energético dos multiplicadores inexatos e dos multiplicadores exatos, as arquiteturas destes foram descritas em *Verilog* e otimizadas por uma ferramenta de síntese. O multiplicador exato foi descrito de duas maneiras diferentes, uma usando uma arquitetura semelhante ao multiplicador inexato (através de produtos parciais, deslocamentos e somas) e a outra foi selecionada automaticamente e otimizada pela ferramenta de síntese. O melhor resultado destas duas arquiteturas foi selecionado como circuito de referência para cada caso. Usando um simulador para diferentes larguras de bits e frequências de operação, foram obtidos resultados com reduções na potência dinâmica entre 31,8% e 45,4% [4].

Para confirmar reduções no consumo energético para arquiteturas maiores, o multiplicador inexato foi testado em três aplicações diferentes, FFT, filtro FIR e processador RISC. A maior redução no consumo energético (18,30%) foi verificada para o filtro FIR, sendo que a redução do consumo de energia no processador RISC foi de apenas 1,51%.

2.2.2 Somadores aproximados

Os multiplicadores não são os únicos operadores aritméticos que podem ser modificados, aceitando que algum erro possa ser introduzido nos seus resultados, de forma a reduzir os seus custos de implementação em *hardware*. De seguida alguns somadores aproximados obtidos de diferentes formas são apresentados.

AMA (*"Aproximate Mirror Adder")*

Os somadores AMA resultam de cinco aproximações diferentes para um FA, que têm como base o somador em espelho, MA ("Mirror adder"), apresentado na Figura 2.7, que é composto por 24 transístores [3].

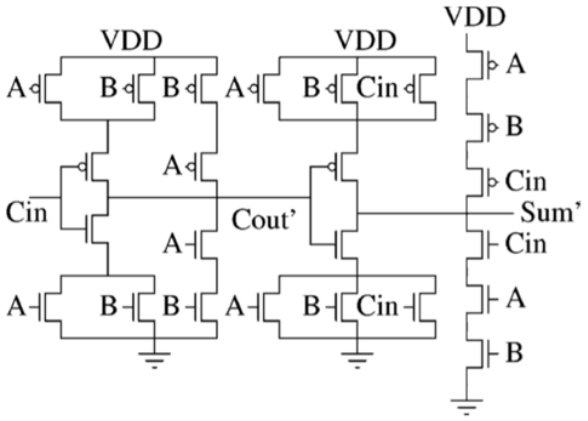


Figura 2.7: Circuito CMOS de um somador em espelho. [3].

Removendo transístores do circuito convencional de forma a que qualquer combinação de A, B ou C_{in} não resulte em curto-circuito ou circuito aberto e que o número de erros nas tabelas de verdade do FA seja mínimo, foram derivados diferentes circuitos para um FA aproximado. A Tabela 2.1 mostra o número de erros nas tabelas de verdade para cada FA aproximado, bem como o número de transístores necessários para a sua implementação. As funções lógicas implementadas por cada FA estão presentes na Tabela 2.2.

Aproximação	Número de erros no bit de soma	Número de erros no bit de *carry*	Número de transístores na implementação
# 1	2	1	16
# 2	2	0	14
# 3	3	1	11
# 4	3	2	11
# 5	4	2	-

Tabela 2.1: Número de erros nas tabelas de verdade dos FA aproximados (AMA).

Estes FA foram testados numa aplicação de compressão de imagem. Os FA aproximados apenas foram usados nos bits menos significativos dos somadores de forma a que a qualidade da saída não

Aproximação	Função lógica do bit de soma	Função lógica do bit de carry
# 1	$\overline{A}C_{in}\overline{B} + ABC_{in}$	$AC_{in} + B$
# 2	$\overline{AC_{in} + BC_{in} + AB}$	$AC_{in} + BC_{in} + AB$
# 3	$\overline{AC_{in} + B}$	$AC_{in} + B$
# 4	$\overline{A}C_{in} + BC_{in}$	A
# 5	B	A

Tabela 2.2: Funções lógicas dos FA aproximados (AMA).

fosse demasiado degradada. Os somadores usados nestas aplicações foram o RCA e o CSA ("*Carry Save Adder*"). Foram comparados os resultados para o caso em que todos os somadores são exatos, o caso em que todos os somadores são aproximados em n bits e para o caso de se truncar n bits os resultados dos somadores exatos.

Na aplicação de compressão de imagem, para o caso em que todos os somadores são exatos foi obtida uma relação sinal-ruído de pico (PSNR) de 31.16 dB. No caso de se usar apenas somadores aproximados em 9 bits (usando a aproximação # 5 nos FA) foi obtido um nível de PSNR de 25.46 dB, e uma redução no consumo de energia de $\approx 60\%$. Para o caso de se truncar 9 bits no resultado, usando todos os somadores exatos, foi obtida uma redução no consumo de energia de $\approx 61\%$, no entanto o nível de PSNR foi de apenas 13,87 dB [3].

AXA (*"Aproximate XOR/XNOR-based Adder"*)

Outro exemplo de FAs aproximados obtidos pela remoção de transístores de uma implementação de um FA exato são os somadores AXA [5]. Existem três AXA diferentes e estes foram obtidos a partir de duas implementações de FA exatos, uma realizada com portas XOR e outra com portas XNOR. Ambos os FA exatos tem uma implementação com 10 transístores, como se pode visualizar na Figura 2.8, que ilustra a implementação por transístores do FA exato baseado em portas XNOR.

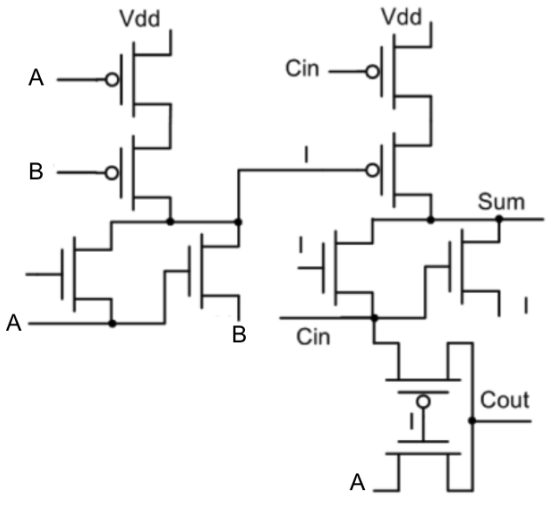


Figura 2.8: Circuito de um FA exato com 10 transístores [5].

O primeiro FA aproximado (AXA1) teve como base o FA exato realizado com portas XOR e a sua implementação requer 8 transístores. Os outros dois FA aproximados (AXA2 e AXA3) tiveram como base o FA realizado com portas XNOR e as suas implementações requerem 6 e 8 transístores, respetivamente. As funções lógicas e o número de erros no bit de soma e no bit de *carry* para os FAs aproximados propostos encontram-se nas Tabelas 2.3 e 2.4, respetivamente.

	Função lógica do bit de soma	Função lógica do bit de *carry*
AXA1	C_{in}	$\overline{(A \oplus B)C_{in} + \overline{A} \cdot \overline{B}}$
AXA2	$\overline{(A \oplus B)}$	$(A \oplus B)C_{in} + AB$
AXA3	$\overline{(A \oplus B)}C_{in}$	$(A \oplus B)C_{in} + AB$

Tabela 2.3: Funções lógicas dos FA aproximados (AXA).

	Número de erros no bit de soma	Número de erros no bit de carry
AXA1	4	4
AXA2	4	0
AXA3	2	0

Tabela 2.4: Número de erros nas tabelas de verdade dos FA aproximados (AXA).

Comparando os FA aproximados com o FA exato, representado na Figura 2.8, foram obtidas reduções na potência dinâmica de 30,57% (AXA3), na potência estática de 65,45% (AXA2) e no atraso do *carry* na saída de 76,09% (AXA1).

LOA (*"Lower-part-OR Adder"*)

Além de remover transístores de uma implementação de um somador existente, somadores aproximados podem ser obtidos eliminando e trocando portas lógicas. O somador LOA[6] é obtido pela modificação da parte de um somador exato que calcula os bits menos significativos do resultado. Este somador é dividido em duas partes uma parte precisa e uma parte aproximada. A estrutura de um somador LOA de p bits está esquematizada na Figura 2.9.

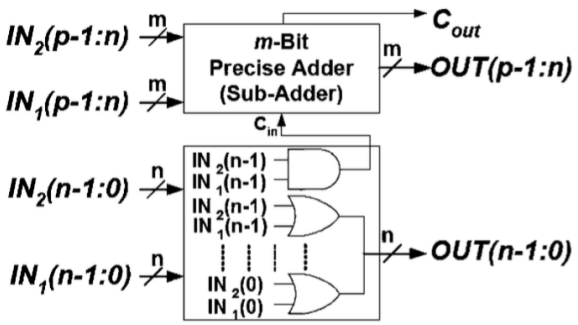


Figura 2.9: Estrutura de um somador LOA [6].

A parte precisa, isto é, a que calcula os m bits mais significativos na Figura 2.9 pode ser realizada

com uma arquitetura de um somador convencional, tal como um RCA. Os n bits menos significativos do resultado são calculados aplicando uma operação OR bit a bit nos respetivos bits de entrada. O *carry* de entrada para a parte precisa é gerado utilizando uma porta AND em ambos os bits mais significativos das entradas da parte imprecisa. A precisão deste somador apenas depende do tamanho (número de bits) da parte aproximada. Ao usar este somador, juntamente com um multiplicador aproximado, numa rede neural utilizada para o reconhecimento facial obteve-se uma redução de área de 54%, mantendo um comportamento semelhante à rede neural implementada com operadores exatos [6].

ETA (”*”Error-Tolerant Adder*)

A área necessária para a implementação de um operador aritmético pode não ser o único recurso que se pretende poupar. Por vezes é desejado que o consumo de energia seja reduzido, mesmo que a área necessária para implementar esses operadores seja maior que para o operador exato homólogo.

Os somadores ETA[7] reduzem a propagação de *carry*, e consequentemente a energia associada a este processo. De modo a reduzir a propagação de *carry* estes somadores realizam uma aritmética diferente para a adição. A aritmética alternativa para a adição é dividida em duas partes: uma parte exata que inclui os bits mais significativos e uma parte inexata que inclui os restantes bits, sendo que o tamanho das duas partes não necessita de ser igual.

A adição na parte exata é realizada de forma convencional utilizando os bits mais significativos dos operandos. O processo de adição na parte inexata é realizado no sentido do bit mais significativo (desta parte) para o menos significativo. Este processo é realizado pela a verificação dos bits dos operandos. Se qualquer um dos bits dos operandos é diferente de ”1”, é realizada uma adição normal destes bits (gerando assim o bit correspondente do resultado) e o processo continua, caso contrário, se ambos os bits forem ”1”o processo de soma é parado e os restantes bits menos significativos do resultado são colocados todos a ”1”. Um exemplo desta aritmética esta representado na Figura 2.10.

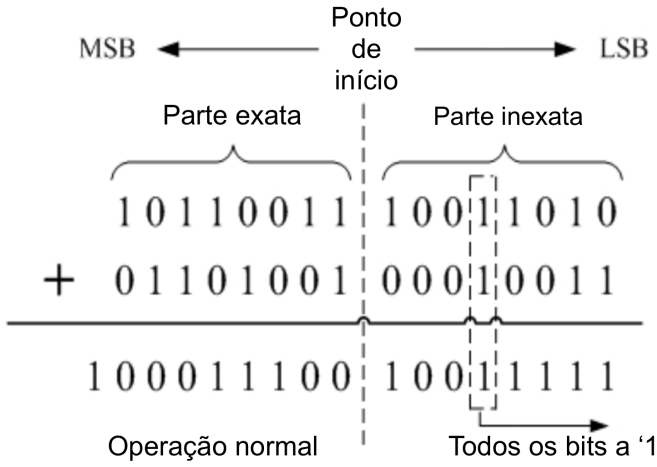


Figura 2.10: Adição num somador ETA [7].

A implementação em *hardware* deste somador está esquematizada na Figura 2.11. A parte exata

pode ser realizada com um somador como o RCA, com o *carry* de entrada com valor lógico ”0”. A parte inexata é constituída por dois blocos: um bloco de adição sem propagação de *carry* e um bloco de controlo. O bloco de adição é composto por portas XOR modificadas, representadas na Figura 2.12. Estas portas XOR modificadas têm 3 transístores extra que permitem alterar o comportamento da porta através do sinal de controlo CTL proveniente do bloco de controlo. Quando CTL = 0, a porta atua como uma porta XOR convencional e quando CTL = 1, o bit de soma é colocado a ”1”. Cada bit do resultado da parte inexata é gerado por uma porta XOR modificada.

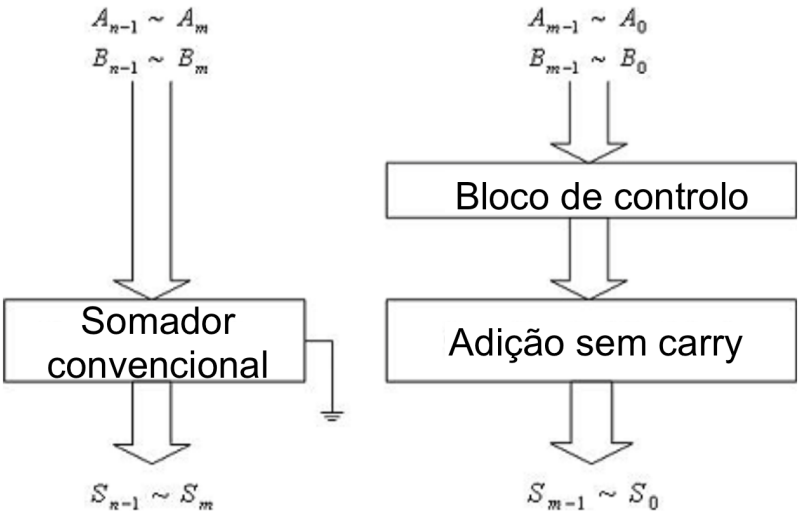


Figura 2.11: Arquitetura de um somador ETA [7].

O bloco de controlo, representado na Figura 2.13 tem a função de detetar a primeira posição em que ambos os bits dos operandos são ”1”, e é formado por vários grupos de células. O início de cada grupo (à exceção do primeiro grupo) é constituído por uma célula CSGC (*control signal generating cell*) do tipo II sendo as restantes células CSGC do tipo I, ambas representadas na Figura 2.14. A divisão por grupos de células permite reduzir o caminho crítico e portanto diminuir o atraso no bloco de controlo. Cada célula tem a sua saída ligada ao sinal CTL da porta XOR modificada correspondente.

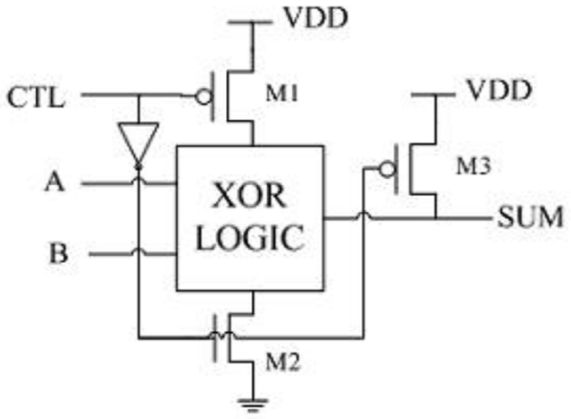


Figura 2.12: Porta XOR modificada [7].

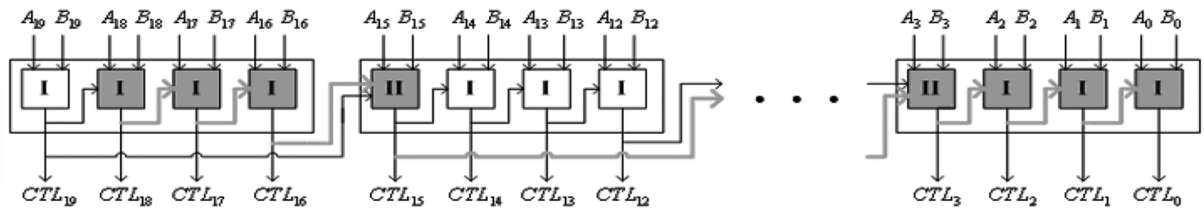
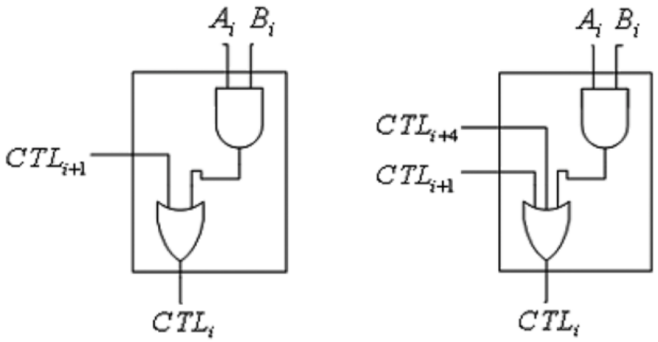


Figura 2.13: Bloco de controlo num somador ETA [7].



(a) Célula CSGC do tipo I. (b) Célula CSGC do tipo II

Figura 2.14: Esquema das células CSGC [7].

Através de simulações mostrou-se que, em termos do produto potência-atraso, um somador ETA de 32 bits, com 20 bits aproximados e a parte exata implementada com um RCA, é melhor em 66.29% que um somador RCA de 32 bits, no entanto necessita de mais transístores para ser implementado (para o ETA são necessários 1006 transístores, enquanto para o RCA são precisos apenas 896). No entanto comparando com outros tipos de somadores (CSK, CSL e CLA), o somador ETA obtém reduções no produto potência-atraso de até 83,70% e reduções na área de até 53,77% [7].

Outros tipos de somadores aproximados chamados de ETAII, ETAIIM e ETAIV foram propostos [8, 9], e pretendem melhorar a precisão, bem como a eficiência energética e o atraso.

O somador ETAII, ilustrado na figura 2.15, é obtido através da divisão de um somador RCA em vários sub-somadores mais pequenos. O *carry* de entrada de cada sub-somador é gerado, através de um bloco gerador de *carry* (implementado usando um CLA), utilizando os operandos do sub-somador anterior. O somador ETAIIM é obtido ligando os blocos de geração de *carry*, nos sub-somadores que geram os bits mais significativos do resultado, em cadeia, melhorando assim a exatidão do somador (visto que os *carry* dos bits mais significativos são obtidos com mais precisão). Estes somadores introduzem erros no resultado visto que a cadeia de *carry* não é exata.

O somador ETAIV é inspirado num somador CSL (*carry-select adder*), em que a geração do *carry* é dividida em duas partes com a adição de um multiplexer 2 para 1. O diagrama deste somador está representado na Figura 2.16. Este somador tem melhor precisão do que o somador ETAIIM.

Em termos do produto potência-atraso o somador ETAII é 33,33% melhor que o somador ETA, o somador ETAIIM é pior em 10% que o somador ETA, no entanto, tem melhor precisão que o somador ETAII e ETA. O somador ETAIV é melhor em 13,33% no produto potência-atraso que o somador ETA, à custa de um aumento em 43% no número de transístores (1444), sendo este o somador da família ETA

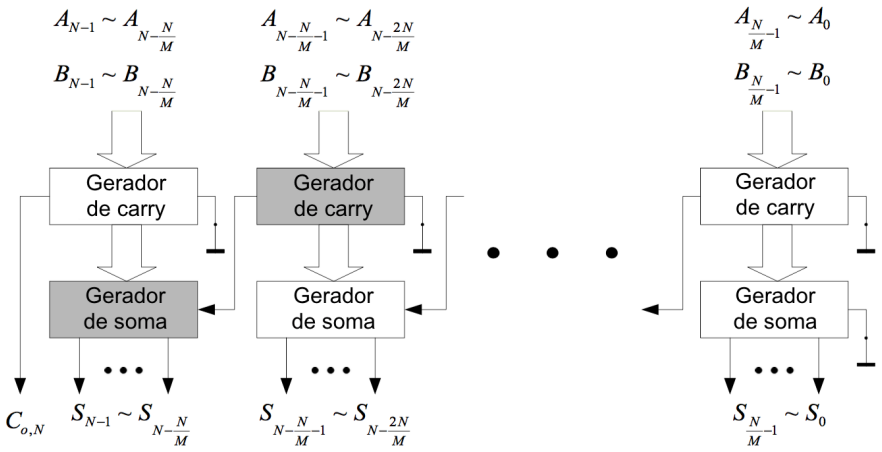


Figura 2.15: Somador ETAII [8].

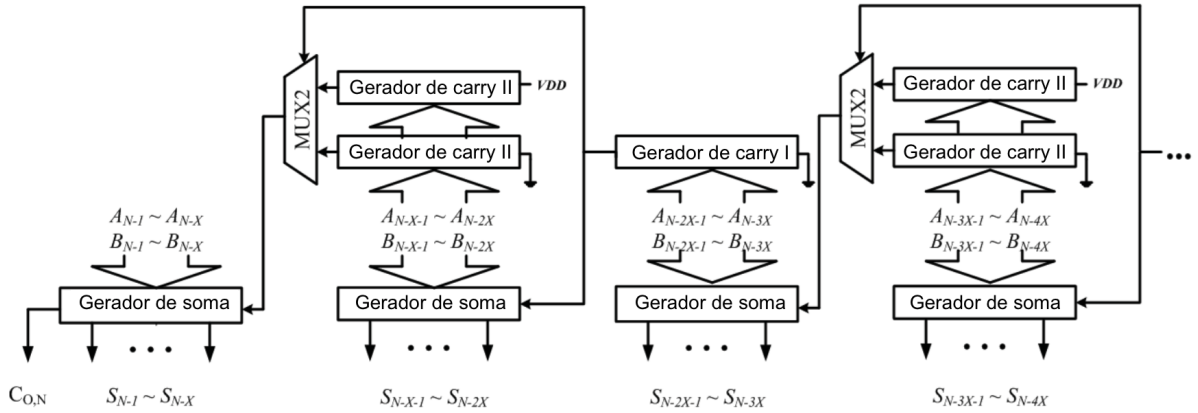


Figura 2.16: Somador ETAIV [9].

com melhor precisão.

Somador de cópia

O consumo de energia de um bloco aritmético pode estar associado à área necessária para a implementação do mesmo, visto que normalmente quantos mais componentes um circuito tem mais energia este consome. Portanto uma maneira de reduzir o consumo energético de um bloco aritmético é simplesmente reduzir a área necessária à sua implementação.

O somador de cópia [10] reduz a área necessária à implementação de um somador, e consequente consumo de energia, removendo os FA que geram os bits menos significativos do resultado. Este somador é baseado num RCA e é formado por duas partes: a parte exata e a parte aproximada. A parte aproximada calcula os bits menos significativos do resultado, que apenas corresponde a uma cópia dos bits menos significativos de um operando do somador. A parte exata corresponde a um RCA convencional e calcula os restantes bits do resultado. Este somador está esquematizado na Figura 2.17 onde k corresponde ao número de bits aproximados e n ao número de bits totais.

Este somador foi testado em implementações de filtros FIR com coeficientes quantizados com 16 bits. Foi usada uma implementação MCM para os filtros FIR (esta implementação está descrita

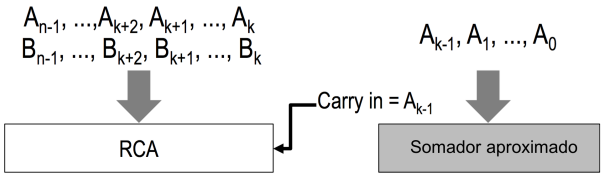


Figura 2.17: Somador RCA aproximado [10].

no Capítulo 3). Foram usadas duas aproximações diferentes (k_1 e k_2), uma para os somadores no bloco MCM e outra para os somadores na linha de atraso do filtro FIR. Para determinar o melhor par de aproximações (k_1, k_2) foram simulados 10 filtros diferentes, usando em cada um deles todas as combinações possíveis (com os valores de k_1 e k_2 inferiores ou igual a 11). Usando a métrica de SNR (*"Signal-to-Noise Ratio"*), em que o ruído corresponde à diferença entre os sinais filtrados com o filtro aproximado e o filtro exato, concluíram que a melhor aproximação seria 1 bit aproximado para os somadores no bloco MCM e 8 bits aproximados para os somadores na linha de atraso. Com esta aproximação obtiveram SNRs entre 60 e 80 dB.

Os filtros aproximados e exatos foram otimizados e sintetizados para diferentes frequências de operação (entre 200 MHz e 780 MHz). Comparando os filtros aproximados com a sua respetiva versão exata foram obtidas reduções de área até 18,8% e reduções de energia até 15,5%.

2.3 Técnicas de computação aproximada ao nível arquitetural

Outras formas para obter sistemas computacionais aproximados é alterar arquiteturas de sistemas computacionais exatos existentes. Um exemplo de um método que modifica a arquitetura de um sistema é um que permite eliminar componentes e os fios associados ao longo dos caminhos que têm menor probabilidade de estar ativos durante o funcionamento de um circuito [11]. A remoção destes componentes é realizada de forma a que os erros estejam dentro dos limites impostos pela a aplicação para qual o circuito foi projetado. O modo de funcionamento deste método depende do tipo de aplicação. As aplicações resilientes ao erro podem ser dividas em duas categorias: as que são sensíveis ao número de cálculos errados (como cálculo de endereços de memória num microprocessador), e as que são sensíveis à magnitude do erro (como processamento de imagem). No caso da última categoria cada saída tem um peso diferente dependendo da sua importância para a qualidade do resultado.

A primeira parte deste método estima a probabilidade de cada caminho estar ativo recorrendo a modelos matemáticos ou simulações. Depois os elementos no caminho com menor probabilidade de estar ativo (ou que tenham a menor probabilidade de estar ativo vezes a significância da saída que geram, para o caso do circuito ter saídas com pesos diferentes) são removidos. A seguir é calculado o erro no novo circuito através de simulações ou estimativas matemáticas. Este processo é repetido até o erro calculado ser maior do que o erro pretendido. Quando esta situação é atingida, a última remoção é revertida, sendo assim obtido o circuito final.

Testando este método para vários circuitos somadores de 64 bits foram obtidos ganhos normalizados do produto área-atraso-energia de 2X para um erro relativo de 10^{-6}% em arquiteturas de soma-

dores paralelas, como o somador Kogge-Stone. No entanto em somadores com arquiteturas em série, como o RCA, ganhos normalizados de 2X para esta métrica implicam erros relativos de 56% [11].

Outro algoritmo usado para a realização de arquiteturas para cálculo aproximado e que foi desenvolvido para filtros FIR é o NAIAD [12]. Este algoritmo é composto por duas partes principais. Na primeira parte conjuntos de coeficientes que satisfazem as especificações do filtro FIR são analisados e são encontrados os que necessitam do menor número de bits para a sua representação em formato de vírgula fixa. Na segunda parte é explorada uma vizinhança de cada solução encontrada, na primeira parte, de forma a encontrar um conjunto de coeficientes que resulta numa implementação em hardware menos complexa, ou seja, com menor número de somadores e subtratores, devido ao aumento na partilha de termos comuns.

Os conjuntos de coeficientes que satisfazem a especificações do filtro são encontrados resolvendo vários problemas de programação linear. Para cada solução é computado o número mínimo de bits para representar os coeficientes de forma a cumprir a especificações e o custo total de cada solução em termos do número de operações necessárias para a sua implementação em hardware. Em cada solução o limite inferior e superior de cada coeficiente é obtido.

A vizinhança de cada solução é explorada selecionando um coeficiente. Mantendo os outros coeficientes fixos, é calculado o intervalo de valores que o coeficiente selecionado pode tomar, de modo a que as especificações sejam cumpridas. Depois para cada valor deste intervalo é calculado o custo de implementação por hardware, e é guardado o valor para qual o custo é menor. Fazendo isto iterativamente para cada coeficiente pode ser encontrada uma solução com um custo de implementação menor do que as soluções encontradas na primeira parte.

Capítulo 3

Filtros FIR e sua implementação

Os filtros FIR (*"Finite Impulse Response"*) são regularmente usados em aplicações de processamento digital de sinal pois têm um conjunto de características importantes para estas aplicações, tais como a estabilidade e fase linear [15]. A implementação de um filtro FIR num circuito digital requer vários passos, desde do cálculo dos seus coeficientes a partir das especificações, à escolha da arquitetura a utilizar e otimização da mesma. Neste capítulo, que está divido em duas secções, começamos por apresentar, na secção 3.1, um filtro FIR genérico e alguns dos métodos existentes para o cálculo dos coeficientes. Em seguida, na secção 3.2, são apresentados algumas das diferentes arquiteturas que permitem implementar um filtro FIR em *hardware*, bem como métodos que minimizam os custos da sua implementação.

3.1 Filtros FIR

Um filtro FIR é um um filtro cuja resposta ao impulso é de duração finita, ou seja, torna-se e mantém-se nula passado um tempo finito. A resposta $y[n]$ de um filtro FIR a um sinal de entrada $x[n]$ é dada por:

$$y[n] = h_0 \cdot x[n] + h_1 \cdot x[n-1] + \cdots + h_{N-1} \cdot x[n-(N-1)] = \sum_{i=0}^{N-1} h_i \cdot x[n-i] \tag{3.1}$$

onde N é a ordem do filtro e h_i é o coeficiente número i do filtro.

Os coeficientes h_i, com $i = 0, ..., N-1$, podem ser determinados a partir das especificações da resposta em frequência do filtro pretendido. No caso do filtro FIR ser um filtro passa-baixo os parâmetros para a especificações da resposta em frequência do filtro são, em geral, os seguintes:

- ω_p - Frequência da banda de passagem.
- ω_s - Frequência da banda de atenuação.
- δ_p - Ondulação na panda de passagem.
- δ_s - Ondulação na banda de atenuação.

Para outros tipos de filtros, como filtros passa-alto, passa-banda ou rejeita-banda, os parâmetros são diferentes, no entanto existe pouca diferença nos métodos usados para estes filtros.

Na Figura 3.1 está representada a magnitude da resposta em frequência para um filtro FIR passa-baixo, onde se pode ver o efeito de cada um dos parâmetros referidos da especificação na resposta em frequência do filtro.

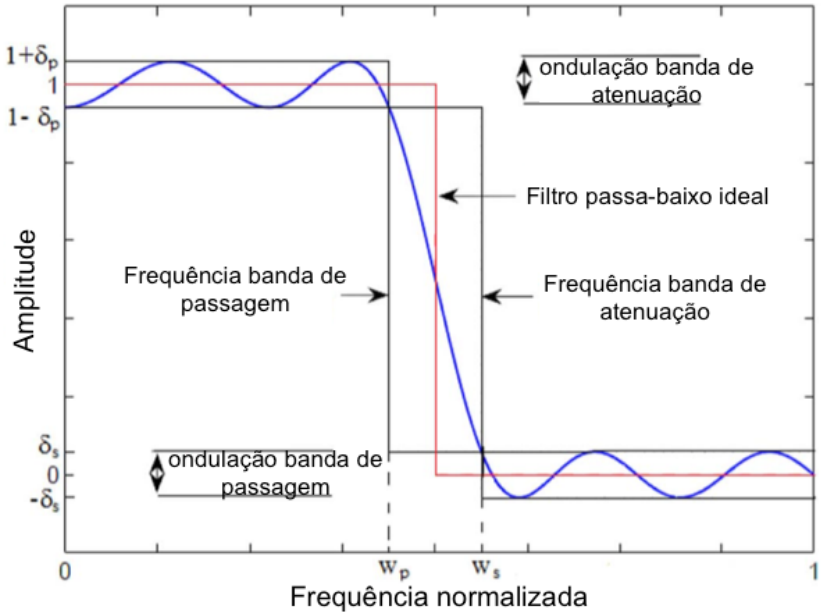


Figura 3.1: Magnitude da resposta em frequência de um filtro passa baixo[12].

Dados os parâmetros para a especificação da resposta em frequência do filtro vários métodos, como por exemplo o método da janela ou o método de Parks-McClellan [16], podem ser usados para determinar os coeficientes do filtro. Nas subsecções 3.1.1 e 3.1.2 são explicados os métodos da janela e de Parks-McClellan, respetivamente.

3.1.1 Método da janela

O método da janela [16] começa com uma resposta em frequência do filtro ideal, $H_d(e^{j\omega})$. Um exemplo de uma resposta em frequência ideal para um filtro passa-baixo encontra-se descrito na expressão (3.2), em que ω_c é a frequência de corte do filtro.

$$|H_d(e^{j\omega})| = \begin{cases} 1, & |\omega| \leq \omega_c \\ 0, & \omega_c < |\omega| \leq \pi \end{cases} . \tag{3.2}$$

A partir da resposta em frequência desejada é calculada a resposta ao impulso do filtro ideal, usando a transformada inversa de Fourier com a seguinte expressão:

$$h_d[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(e^{j\omega}) e^{j\omega n} d\omega, \tag{3.3}$$

onde $h_d[n]$ é a resposta ao impulso do filtro ideal, ou seja coincide com os coeficientes do filtro. Para o caso da resposta ideal da expressão (3.2) a resposta ao impulso seria: $h_d[n] = \frac{\sin(\omega_c n)}{n\pi}$.

Tendo em conta que o filtro ideal teria de ter infinitos coeficientes é necessário multiplicar $h_d[n]$ por uma função de janela de duração finita $w[n]$ para obter N coeficientes h_i do filtro:

$$h[n] = h_d[n] \cdot w[n], \tag{3.4}$$

$$h_i = h[i], \quad 0 \leq i < N. \tag{3.5}$$

Existem várias funções de janela diferentes, sendo as mais comuns a rectangular (3.6), a de Hanning (3.7) e a de Hamming (3.8) [16]. Ao usar a janela rectangular as ondulações na banda de passagem e de atenuação do filtro resultante são elevadas. As janelas de Hamming e de Hanning permitem diminuir estas ondulações.

$$w_{rec}[n] = \begin{cases} 1, & 0 \leq n < N \\ 0, & \text{caso contrário} \end{cases}, \tag{3.6}$$

$$w_{Hanning}[n] = \begin{cases} 0,5 - 0,5\cos(2\pi n/(N-1)), & 0 \leq n < N \\ 0, & \text{caso contrário} \end{cases}, \tag{3.7}$$

$$w_{Hamming}[n] = \begin{cases} 0,54 - 0,46\cos(2\pi n/(N-1)), & 0 \leq n < N \\ 0, & \text{caso contrário.} \end{cases} \tag{3.8}$$

3.1.2 Método de Parks-McClellan

Outro método para o cálculo dos coeficientes de um filtro FIR é o método de Parks-McClellan [16], que tem como objetivo encontrar o filtro ótimo de ordem N, que satisfaça as especificações de ω_p e ω_s, minimizando δ_p e δ_s.

Este método utiliza uma aproximação polinomial para obter os coeficientes, que cuja resposta em frequência se aproxima da resposta pretendida. O polinómio de ordem N que melhor aproxima a resposta desejada é encontrado utilizando um algoritmo iterativo chamado de algoritmo de Remez. Este algoritmo utiliza os polinómios de Chebyshev e segue os seguintes passos para obter um polinómio de ordem N que melhor aproxima a função f, começando com os pontos $x_1, x_2, ..., x_{N+2}$:

- O sistema linear de equações é resolvido em ordem a $b_0, b_1, ..., b_N$ e E:

$$b_0 + b_1 x_i + b_N x_i^N + (-1)^i E = f(x_i), \quad i = 1, 2, ..., N+2.$$

- Usar os b_i como coeficientes para formar o polinómio P_N.

- Encontrar o conjunto M de máximos locais de $|P_N(x) - f(x)|$.

- Se todos os máximos locais são de igual magnitude e alternam no sinal, então o polinómio P_N é o que melhor aproxima a função f. Caso contrário substituir os pontos $x_1, x_2, ..., xN + 2$ pelos pontos em M e repetir os passos em cima.

Depois de encontrado o polinómio de ordem N que melhor aproxima a resposta desejada a transformada de Fourier é usada para obter os coeficientes do filtro. Uma descrição mais detalhada do método de Parks-McClellan encontra-se no Apêndice A.

3.2 Implementação de filtros FIR

A implementação de um filtro FIR pode ser realizada em *hardware* de diversas maneiras, tendo cada uma delas as suas vantagens e desvantagens. De seguida são apresentadas algumas arquiteturas tradicionais para implementar filtro FIR.

3.2.1 Arquiteturas tradicionais

A implementação direta da equação (3.1), representada na Figura 3.2, é designada como a forma direta. Outra forma de implementar um filtro FIR é a forma transposta, representada na Figura 3.3, que é obtida a partir da forma direta, alterando a posição dos registos. Existem ainda outras formas, chamadas de formas híbridas, que misturam a forma direta com a forma transposta.

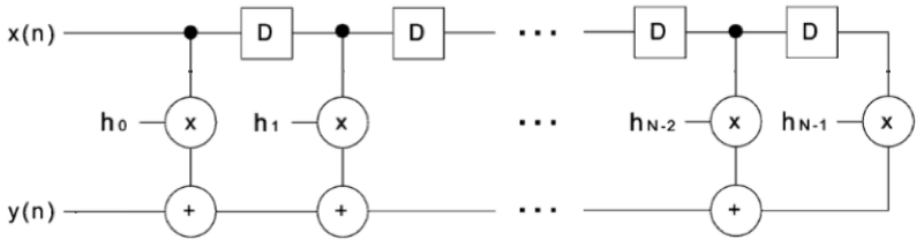


Figura 3.2: Forma direta de um filtro FIR [12].

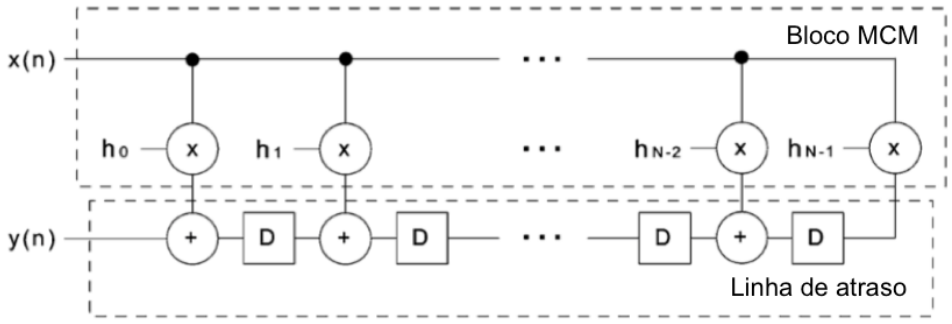


Figura 3.3: Forma transposta de um filtro FIR [12].

Analisando as Figuras 3.2 e 3.3 nota-se que as únicas operações aritméticas necessárias para implementação de um filtro FIR são a soma e a multiplicação. Para implementar um filtro FIR, na forma direta ou na forma transposta, são necessários exatamente N multiplicadores e $N-1$ somadores e registos para um filtro de ordem N. No entanto existem diferenças entre a forma direta e a forma transposta, especificamente no tamanho dos registos, no caminho crítico e na capacitância da entrada. Os registos na forma direta têm todos o mesmo tamanho, ao contrário no que acontece na forma transposta em que o tamanho dos registos depende do número de bits de cada coeficiente. O caminho crítico na forma transposta é menor (um multiplicador e um somador) do que na forma direta (um multiplicador e $N-1$ somadores). A capacitância de entrada na forma transposta é maior pois a entrada está conectada a $N-1$ multiplicadores, enquanto que na forma direta apenas está ligada a um registo e a um multiplicador [15]. As formas híbridas permitem conjugar as vantagens das formas direta e transposta, obtendo assim arquiteturas com diferentes características.

Depois de escolhida a arquitetura a usar para implementar o filtro FIR é ainda possível otimizá-la de forma a que os seus custos de implementação sejam reduzidos.

Sabendo que coeficientes de um filtro FIR são fixos e pré determinados a partir das suas especificações (N, ω_p, ω_s, δ_p e δ_s), como explicado na secção 3.1, e que a realização de multiplicadores em *hardware* traduz-se num aumento em termos de área, atraso e de consumo de energia, estes filtros podem ser implementados usando arquiteturas que apenas usam somadores, subtratores e deslocamentos [17].

Em seguida são apresentados alguns métodos que permitem realizar as arquiteturas apenas com somadores e deslocamentos, apresentando o exemplo para a forma transposta.

3.2.2 Arquitetura com somas e deslocamentos

Uma multiplicação por uma constante pode ser realizada apenas com somas e deslocamentos, por exemplo a multiplicação pela constante 11, $y = 11x$, pode ser escrita usando uma representação binária, $y = (1011)x$ que por sua vez pode ser decomposta em:

$$y = x \times 2^3 + x \times 2^1 + x = x \ll 3 + x \ll 1 + x, \tag{3.9}$$

o que corresponde a duas operações de soma e duas operações de deslocamento.

Na forma transposta de um filtro FIR (Figura 3.3) a multiplicação da entrada pelos coeficientes do filtro chama-se um bloco MCM (*"Multiple Constant Multiplication"*), ou seja, é um bloco que possui apenas uma entrada e várias saídas, em que cada saída corresponde ao valor da entrada multiplicado por uma constante, que corresponde aos coeficientes do filtro. Nesta arquitetura a entrada é comum para todos os multiplicadores pelo pode haver partilha de somas parciais reduzindo o custo de implementação.

Assim, blocos MCM podem ser implementados eficientemente recorrendo apenas a somadores/subtratores e a deslocamentos. Como os deslocamentos não tem custo em *hardware*, visto que são apenas fios, o problema de otimização dos blocos MCM reduz-se a minimizar o número de somadores/subtratores de modo a reduzir os custos de implementação destes blocos. É de notar que este pro-

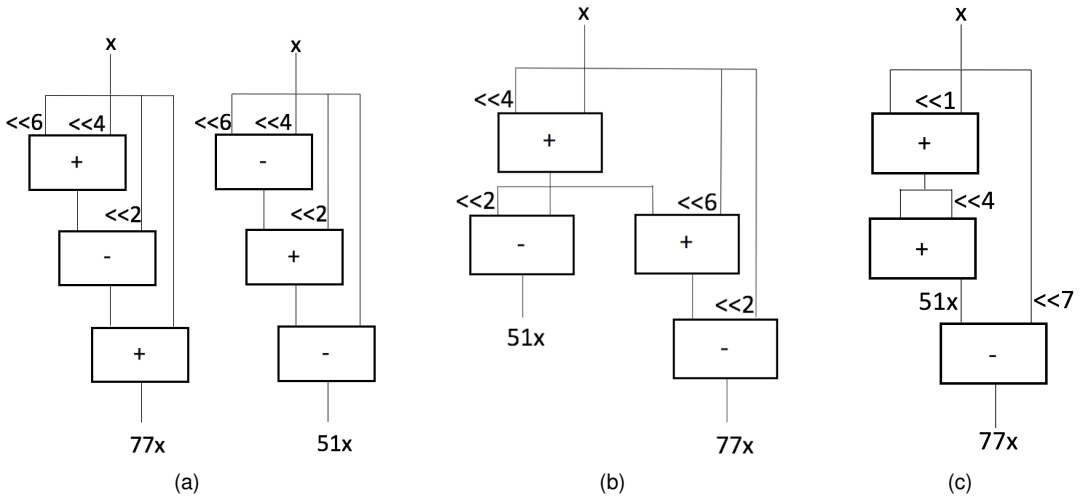


Figura 3.4: Arquiteturas para multiplicação pelas constantes 77 e 51 geradas por: (a) representação CSD individualmente, (b) Método CSE [19], (c) Método GB [17].

blema de otimização é NP-difícil [18]. Os algoritmos existentes para este problema reduzem o número de somadores/subtratores aumentando a partilha de produtos parciais, e podem ser divididos em duas categorias[15]: Os baseados na eliminação de subexpressões, ou CSE (*"Common Subexpression Elimination"*) [19, 20, 21, 22, 23], e os baseados em grafos, ou GB (*"Graph-based"*) [17, 24, 25, 26, 27].

Nos métodos CSE as constantes são primeiro descritas usando uma representação, sendo as mais comuns a representação binária, CSD (*"Canonical Signed Digit"*) ou MSD (*"Minimal Signed Digit"*).

A representação CSD é definida da seguinte forma:

- Um inteiro pode ser representado em CSD usando n dígitos como $\sum_{i=0}^{n-1} d_i 2^i$, onde $d \in \{1, 0, \overline{1}\}$ e $\overline{1}$ tem o valor de -1 .
- Os dígitos não nulos não podem ser adjacentes.
- Uma constante é representada com o mínimo de dígitos não nulos.

De forma a tornar clara esta representação tomemos exemplos das constantes 51 e 77 [15]:

$$51 = (10\overline{1}010\overline{1})_{CSD} = 2^6 - 2^4 + 2^2 - 1$$

$$77 = (1010\overline{1}01)_{CSD} = 2^6 + 2^4 - 2^2 + 1$$

A representação MSD apenas difere da CSD no facto que dígitos não nulos podem ser adjacentes. Ao usar estas representações a multiplicação por qualquer constante traduz se apenas em operações de soma, subtração e deslocamento. Por exemplo para multiplicação de 51 e 77 por x, usando apenas individualmente a representação CSD de cada constante, seriam necessárias três operações para obter cada multiplicação portanto, num total de seis operações como exemplificado na Figura 3.4(a).

Ao usar um algoritmo CSE exato [19] é possível reduzir o número de operações necessárias para a multiplicação de x por 51 e 77 simultaneamente, usando produtos parciais comuns a ambas as multiplicações. É de notar que o número total de operações foi reduzido de 6 para apenas 4, como

se pode verificar na Figura 3.4 (b)

Os métodos CSE estão limitados no espaço de procura de soluções devido ao tipo de representação que usam. No entanto, os métodos GB não têm esta limitação abrangendo um espaço de procura maior pelo que encontram soluções melhores, mas requerem mais computação [15]. Na Figura 3.4(c) está representada a solução gerado por um método GB para a multiplicação pelas constantes 51 e 77, onde apenas três operações são necessárias.

Usando estes métodos todo o bloco MCM do filtro FIR pode ser implementado usando uma arquitetura de somas e deslocamentos com um número de operações reduzido, que consequentemente reduzem os custos de implementação.

As arquiteturas de filtros FIR baseadas em somas/subtracções e deslocamentos serão as arquiteturas alvo para aplicar os métodos de computação aproximada descritos nos próximos capítulos.

Capítulo 4

Filtros FIR - Computação aproximada ao nível arquitetural

Neste capítulo é proposto um método desenvolvido para a implementação de filtros FIR, que faz uso da computação aproximada ao nível arquitetural. Este método tem como objetivo principal reduzir os custos de implementação de um filtro FIR, removendo somadores/subtratores e alterando ligações no bloco MCM de uma implementação de um filtro FIR já existente. Este capítulo está dividido em duas secções, sendo que na primeira é apresentado o método e dois algoritmos, um exaustivo e outro heurístico, para a sua implementação. Na segunda secção são apresentados os resultados obtidos pelo método proposto para 10 filtros passa-baixo diferentes.

4.1 Método proposto

O método proposto pretende reduzir o número de somadores necessários à implementação do bloco MCM de um filtro FIR, tendo como base uma arquitetura com somas e deslocamentos de um bloco MCM existente. A redução do número de somadores é conseguida pela remoção de alguns somadores do bloco MCM existente, refazendo as ligações necessárias para minimizar o erro nas saídas do bloco MCM.

O método proposto é constituído por um algoritmo que, dada uma arquitetura de um bloco MCM, procura as modificações que podem ser feitas a este bloco que minimizam o erro total, introduzido pelas alterações, nas suas saídas. O erro total que o algoritmo pretende minimizar tem a seguinte expressão:

$$E = \sum_{i=1}^{N} |\tilde{y}_i - y_i|, \tag{4.1}$$

em que $\tilde{y}_i$ corresponde ao valor da saída i do bloco MCM modificado, com o valor da entrada igual a 1, e y_i corresponde ao valor da saída i do bloco MCM inicial, também com o valor da entrada igual a 1. As únicas modificações, ao bloco MCM inicial, que são permitidas pelo o algoritmo correspondem à remoção de somadores. Ao remover um somador, o nó onde se encontrava ligada a sua saída é

conectado à saída de um dos somadores restantes, introduzindo deslocamentos se necessário.

Nas seguintes subsecções é explicado com mais detalhe o funcionamento do algoritmo proposto. Na primeira subsecção é apresentada a forma como a arquitetura do bloco MCM inicial é representada, na segunda subsecção é explicado em detalhe a versão exaustiva do algoritmo e na terceira é introduzido um algoritmo heurístico que permite diminuir o tamanho do espaço de procura, encontrando uma solução mais rapidamente.

4.1.1 Representação da arquitetura do bloco MCM

O ponto de partida do algoritmo proposto é uma arquitetura de um bloco MCM. Esta arquitetura corresponde a uma árvore de somadores e é obtida efetuando os seguintes passos:

- A partir das especificações do filtro são obtidos os coeficientes usando o método de Parks-McClellan (comando *firpm* no MATLAB);
- Os coeficientes são quantizados com o número de bits desejado, resultando assim em coeficientes inteiros;
- É obtida a árvore de somadores que implementa o bloco MCM usando o algoritmo Hcub [28].

Por exemplo para um bloco MCM de um filtro FIR passa-baixo de ordem $N = 10$, com $\omega_s = 0.1\pi$, $\omega_p = 0.5\pi$ e com 10 bits de quantização resultou nos seguintes coeficientes: -22, -13, 60, 193, 302, 302, 193, 60, -13, -22. A árvore de somadores obtida pelo Hcub para a implementação deste filtro usando um bloco MCM está representada na Figura 4.1. Nesta figura as operações de deslocamento, sem custo, estão representadas a cinzento e os somadores/subtratores a vermelho escuro. Em ambos os casos é indicado o valor resultante na saída desse operador. Note-se que existem um conjunto de operadores que apesar de representados não tem que ser implementados pois não são necessários; deslocamentos de zero posições ou "multiplicação" de saída por -1.

A partir desta árvore de somadores são retiradas as seguintes informações:

- O número total de somadores/subtratores, designado por S;
- O valor que cada somador/subtrator calcula;
- As dependências, ou seja, ligações entre somadores/subtratores;
- O valor e posição dos deslocamentos;
- Os somadores/subtratores que geram as saídas do bloco MCM para determinar o erro resultante da aproximação.

Com base nestas informações é criado um vetor linha de comprimento S com os valores que cada somador/subtrator calcula, chamado $somadores$, sendo que este vetor é ordenado por ordem crescente da profundidade de cada somador/subtrator. A informação das dependências entre somadores/subtratores, bem como a posição e valor dos deslocamentos é guardada numa matriz SxS, de-

signada por dep, e num vetor de expressões, com comprimento S, nomeado de dep_exp. A correspondência de cada saída do bloco MCM ao somador/subtrator que a gera é feita usando uma matriz SxN, chamada de $output\_map$ e um vetor linha de comprimento N, designado por $output\_c$.

Para o exemplo da Figura 4.1 seriam guardados os seguintes valores (que são explicados em seguida):

$$somadores = [15, 17, 11, 13, 151, 193]$$

$$dep = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$dep\_exp = \{0, 0, a1 - 4, a1 - 2, a1 + 8 * a2, -a1 + 16 * a4\}$$

$$output\_map = \begin{bmatrix} 0 & 0 & 4 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$output\_c = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

O vetor $somadores$ além de conter os valores calculados por cada somador/subtrator, também permite identificar os somadores/subtratores. Os somadores/subtratores são identificados pelo o índice correspondente neste vetor, por exemplo, o subtrator que calcula o valor de $11x$ é identificado como sendo o somador 3 (por motivos de simplificação os somadores não são distinguidos dos subtratores, e por isso embora o cálculo de $11x$ seja feito por um subtrator este é chamado de somador) .

A matriz dep é construída tendo em conta as ligações entre os somadores. O valor $'1'$ num elemento na coluna j e na linha i significa que o resultado do somador j depende do resultado do somador i. O número máximo de elementos não nulos numa coluna da matriz dep é dois, visto que o resultado de um somador depende no máximo de dois resultados de somadores diferentes. Na matriz dep do exemplo, as duas primeiras colunas são nulas porque os somadores 1 e 2 não dependem de nenhum outro somador. A terceira e quarta coluna apenas têm um único valor não nulo, ambos na primeira linha, porque os somadores 3 e 4 só dependem do somador 1. A quinta e sexta coluna têm dois elementos com o valor $'1'$ pois o somador 5 depende dos somadores 1 e 2 e o somador 6 depende dos somadores 1 e 4.

O vetor dep_exp contém as expressões que geram o resultado de cada somador, quando estes dependem de outros somadores. No exemplo os dois primeiros elementos deste vetor são nulos pois,

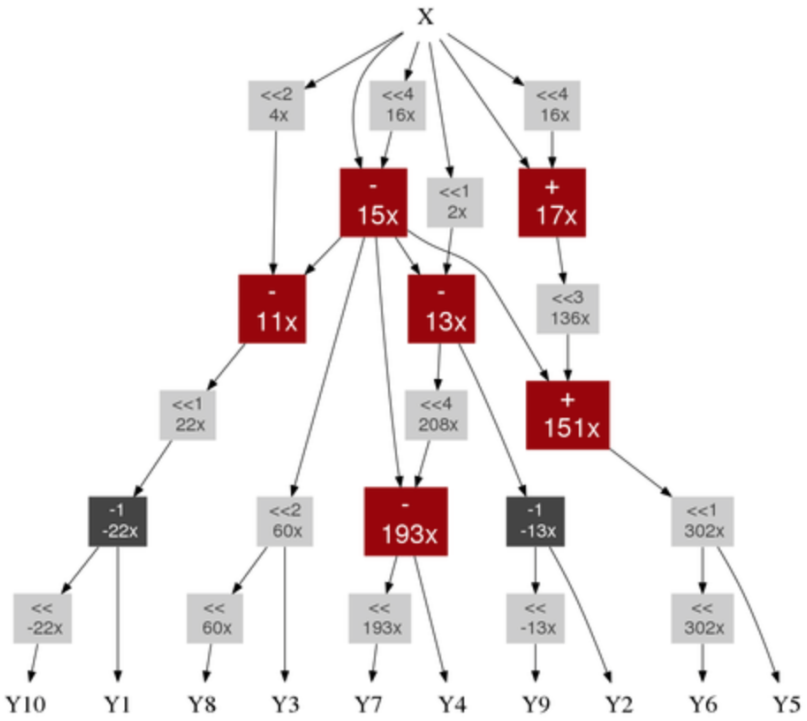


Figura 4.1: Árvore de somadores que implementa o bloco MCM.

como referido anteriormente, os somadores 1 e 2 não dependem de nenhum outro somador. O terceiro elemento é $a1 - 4$ porque o resultado do somador 3 corresponde à subtração de $4x$, ou seja $(x << 2)$, ao resultado do somador 1, representado por $a1$. O sexto elemento deste vetor é $-a1 + 16 * a4$, pois o resultado do somador 6 corresponde à subtração do resultado do somador 1 do resultado do somador 4, deslocado à esquerda por 4 bits (que corresponde a uma multiplicação por 16).

A matriz $output\_map$ e o vetor $output\_c$ são preenchidos de forma a que satisfaça a seguinte expressão:

$$somadores \cdot output\_map + output\_c = h \tag{4.2}$$

em que h é o vetor linha de comprimento N que contém todos os coeficientes do filtro. A matriz $output\_map$ contém os valores dos deslocamentos e negações aplicadas aos resultados dos somadores, que geram as saídas do bloco MCM, e o vetor $output\_c$ contém as saídas do bloco que não são geradas por somadores, que é o caso para coeficientes que correspondem a potências de 2. No exemplo o vetor $output\_c$ é nulo porque todas as saídas do bloco MCM são geradas por pelo menos um somador. A posição dos valores não nulos da matriz $output\_map$ indica qual o somador que gera cada saída, sendo que um valor não nulo na coluna j e linha i significa que a saída j é gerada pelo resultado do somador i multiplicado pelo valor deste elemento. No exemplo anterior é possível verificar que o elemento da matriz $output\_map$ na coluna 1 e linha 3 tem o valor de -2 porque a saída 1 é gerada pelo resultado do somador 3 multiplicado por -2, ou seja, negado e deslocado à esquerda de um bit.

4.1.2 Algoritmo exaustivo

Tendo todas as informações necessárias sobre sobre a arquitetura do bloco MCM inicial guardadas, o algoritmo exaustivo pode iniciar a procura da solução que minimiza o erro total E. O algoritmo tem como um dos parâmetros iniciais o número de somadores que se pretendem remover, designado por N_{sub}.

De forma a tornar mais claro como o algoritmo funciona os seguintes processos são definidos:

- O processo de remover um somador será chamado de eliminação de um somador.
- O processo de conectar o nó onde estava ligada a saída de um somador que foi eliminado à saída de um outro somador ou à entrada do bloco MCM será chamado de substituição de um somador.

O algoritmo exaustivo começa por calcular todas as combinações possíveis com N_{sub} elementos de somadores a eliminar, sendo que o número de combinações total testadas pelo algoritmo é dado por $C^S_{N_{sub}}$.

A seguir para cada combinação de somadores testada os seguintes passos são efetuados:

- É eliminado o somador com o menor índice (posição no vetor $somadores$);
- São analisadas as possíveis hipóteses para substituir o somador eliminado;
- É escolhida a hipótese que gera o menor erro e é efetuada a substituição;
- As modificações causadas pela substituição são propagadas para toda a árvore de somadores;
- Os pontos anteriores são efetuados até se eliminar todos os somadores da combinação.

Tendo eliminado todos os somadores de uma combinação o erro total E é calculado usando a matriz $output\_map$ e o vetor $output\_c$. Realizando os mesmos passos para todas as combinações possíveis é guardada a combinação que gerou o menor valor de E, correspondendo esta à solução encontrada pelo algoritmo exaustivo.

Os somadores são eliminados e substituídos por ordem crescente do índice, ou seja, pela ordem crescente de profundidade, de forma a garantir que todas as substituições efetuadas são válidas, pois ao substituir um somador, os valores calculados nos somadores a profundidades maiores podem ser modificados.

As hipóteses possíveis para a substituição de um somador eliminado são calculadas pela função $CalculaSubs$ e a propagação das modificações causadas pela substituição de um somador são efetuadas pela função $PropagaMod$, que serão explicadas de seguida, tomando como exemplo o bloco MCM anterior.

Considerando que se pretende eliminar 2 somadores, ou seja $N_{sub} = 2$, e a combinação de somadores a ser eliminados testada for $[1, 6]$, ou seja eliminar os somadores 1 e 6, o algoritmo começaria por eliminar o somador 1 e depois analisava quais as substituições possíveis para substituir este somador. Para realizar este processo é chamada a função $CalculaSubs$ que aceita como argumentos o valor calculado pelo somador eliminado s_{elim}, os valores calculados pelos somadores que podem ser usados

para substituir o somador eliminado, $disp$, e o erro relativo máximo permitido para uma substituição, tol. Esta função retorna todas as substituições possíveis que satisfazem a seguinte condição:

$$(1 - tol)s_{elim} < sub < (1 + tol)s_{elim}, \tag{4.3}$$

em que sub é o valor da possível substituição. O valor de tol é um parâmetro inicial do algoritmo e é definido à priori.

As substituições que satisfazem a condição (4.3) são encontradas considerando deslocamentos à esquerda de todos os valores de $disp$. Além dos valores de $disp$ também é considerada a hipótese de apenas deslocar a entrada do bloco MCM.

No exemplo considerado, ou seja, para substituir o somador 1 apenas pode ser considerado como possível substituição deslocamentos do valor da entrada do bloco MCM, porque é imposto que apenas somadores com índices inferiores ao somador que foi eliminado sejam usados para substituir, de forma a garantir que todas as substituições efetuadas são válidas. Como o somador 1 calcula o valor de $15x$ é escolhido apenas o valor $16x$ (que corresponde ao deslocamento de 4 bits para a esquerda da entrada do bloco MCM) como possível substituição.

O próximo passo é escolher a substituição que gera o menor erro. Como apenas existe uma possibilidade a substituição por $16x$ é efetuada. A seguir as modificações causadas por esta substituição necessitam de ser propagadas para a restante árvore de somadores, o que é efetuado usando a função $PropagaMod$.

Esta função tem como argumentos o índice do somador eliminado, o valor pelo qual o somador eliminado foi substituído e a lista de somadores que possam ter sido afetados por esta modificação, $som_{actualiza}$. A propagação das modificações para toda a árvore de somadores é efetuada da seguinte forma:

- 1. A lista de somadores afetados som_{afect} é inicializada com o somador eliminado;
- 2. É selecionado o somador da lista $som_{actualiza}$ com o menor índice e removido desta lista;
- 3. É verificado se o somador selecionado depende de algum somador presente na lista som_{afect}, usando a matriz dep;
- 4. Se o somador selecionado depender de algum somador da lista som_{afect} o valor calculado por este é atualizado usando a expressão correspondente presente no vetor dep_exp. Caso contrário voltar ao ponto 2;
- 5. O somador selecionado é adicionado à lista som_{afect}.

Quando a lista $som_{actualiza}$ ficar vazia o processo pára.

A árvore de somadores que resulta da remoção do somador 1 e a sua substituição por $16x$ está ilustrada na Figura 4.2.

Tendo removido o somador 1 e propagado as alterações para o resto da árvore, o algoritmo contínua para o próximo somador a ser removido, que no exemplo é o somador 6. Para este somador a

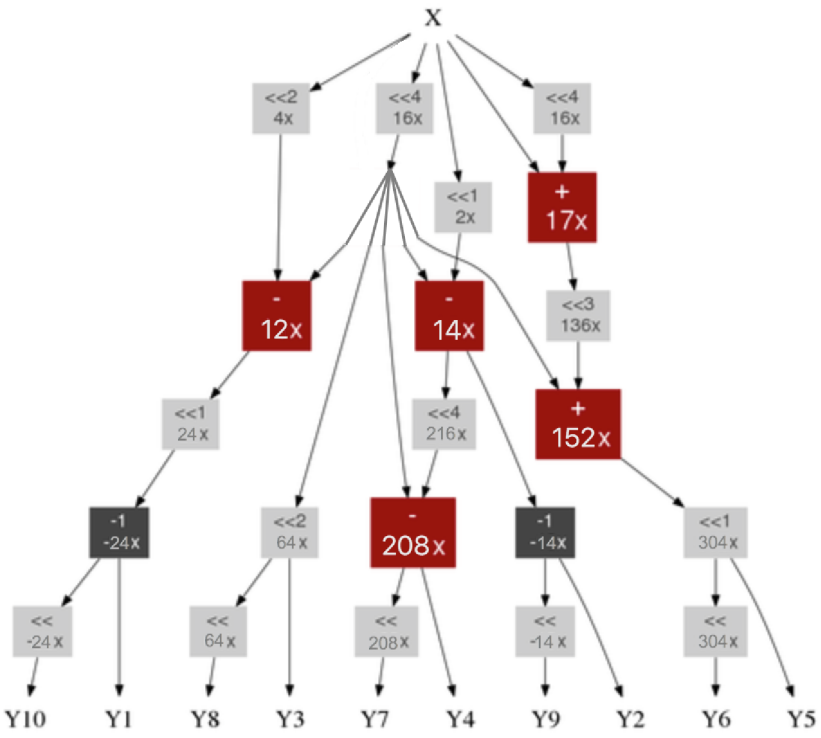


Figura 4.2: Árvore de somadores resultante da remoção do somador 1.

função $CalculaSubs$ retorna mais substituições possíveis, visto que todos os outros somadores podem ser utilizados. As substituições possíveis, para $tol = 0,35$, que a função retorna são as seguintes: $[128x, 136x, 192x, 224x, 152x]$. Como a substituição que introduz menos erro é $192x$ (erro igual a $|192 - 193| = 1$), o resultado do somador 6 é substituído por este valor. A árvore de somadores que corresponde à remoção dos somadores 1 e 6 está representada na Figura 4.3.

Terminada a remoção dos somadores na combinação testada, que neste neste exemplo é a combinação $[1, 6]$, os valores na saída do bloco MCM são determinados multiplicando os novos valores calculados pelos somadores e a matriz $output\_map$, depois o erro total é calculado, sendo neste caso igual a:

$$\begin{aligned} E &= 2(|-24-(-22)| + |-14-(-13)| + |64-60| + |192-193| + |304-302|) \\ &= 2(2+1+4+1+2) = 20. \end{aligned}$$

De forma a compreender como as soluções encontradas são afetadas pelo valor de tol, o algoritmo exaustivo foi testado para diferentes filtros e diferentes valores de tol e N_{sub}. O erro total E de cada solução encontrada para um filtro de ordem $N = 30$, com $\omega_p = 0,3\pi$, $\omega_s = 0,5\pi$ e bits quantizados com 10 bits, designado por Filtro A, e para um filtro de ordem $N = 50$, com $\omega_p = 0,1\pi$, $\omega_s = 0,15\pi$ e bits quantizados com 8 bits, designado por Filtro B, encontra-se representado nas Figuras 4.4 e 4.5, respetivamente. O erro total E de diferentes soluções encontradas usando diferentes valores de tol para o filtro do exemplo anterior encontra-se ilustrado na Figura 4.6. É possível verificar que usar valores para tol entre 0,2 e 0,45 resulta nas soluções com o menor valor do erro total E, para todos os valores de N_{sub}. A partir destes resultados definiu-se que o valor de tol a ser usado nos testes a serem

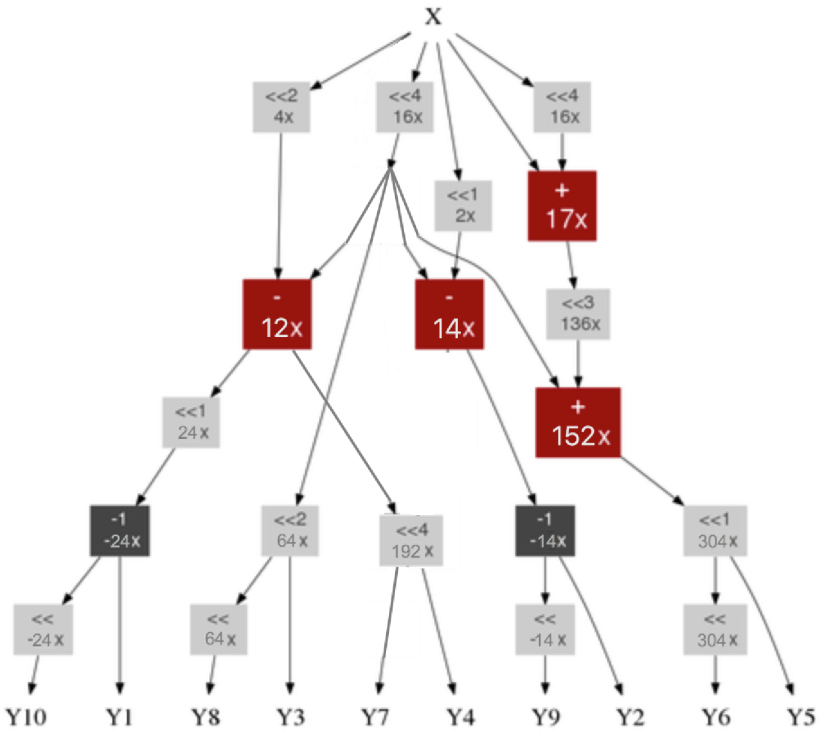


Figura 4.3: Árvore de somadores resultante da remoção dos somadores 1 e 6.

realizados a este método seria 0,35, visto ser um valor que se encontra dentro do intervalo que gerou as melhores soluções.

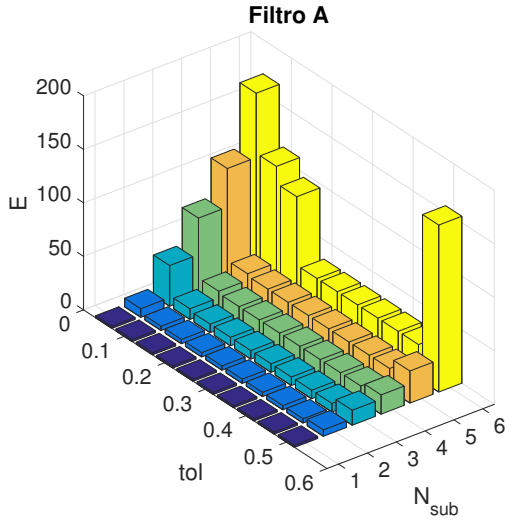


Figura 4.4: Erro total das soluções encontradas para o filtro A pelo algoritmo exaustivo para diferentes valores de tol e N_{sub}.

As soluções encontradas para diferentes valores de N_{sub} e $tol = 0,35$ para o filtro do exemplo encontram-se na Tabela 4.1 onde se pode notar que quanto mais somadores são removidos maior é o erro total, como seria esperado.

Nas Figuras 4.7 e 4.8 estão ilustradas as respostas em frequência do filtro FIR aproximado gerado pelo algoritmo proposto, com $N_{sub} = 1$ e $N_{sub} = 3$, respetivamente.

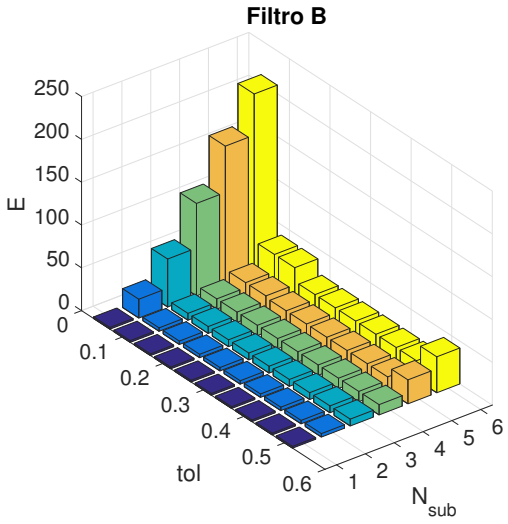


Figura 4.5: Erro total das soluções encontradas para o filtro B pelo algoritmo exaustivo para diferentes valores de tol e N_{sub}.

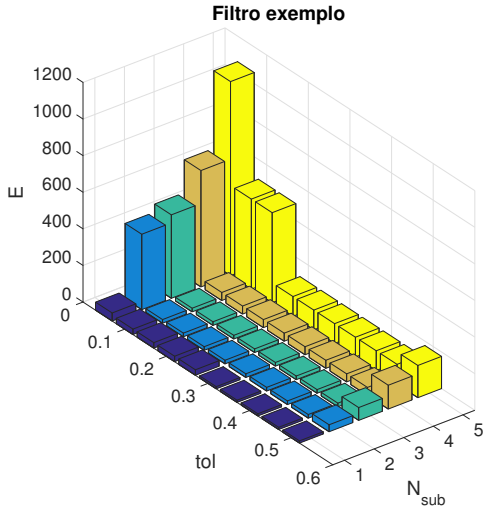


Figura 4.6: Erro total das soluções encontradas para o filtro do exemplo pelo algoritmo exaustivo para diferentes valores de tol e N_{sub}.

N_{sub}	Somadores eliminados	Erro total	Erro de δ_p (dB)	Erro de δ_s (dB)
1	3	12	0,086	3,149
2	1,6	20	0,146	3,010
3	1,4,6	20	0,056	2,935
4	1,2,4,6	44	0,205	3,761
5	1,2,4,5,6	108	0,782	11,319

Tabela 4.1: Erro total para diferentes valores de N_{sub} e $tol = 0,35$.

É possível verificar que as respostas em frequência dos filtros FIR aproximados, são semelhantes à resposta em frequência do filtro exato na banda de passagem. Na banda de atenuação as respostas do filtro exato e aproximado são diferentes, no entanto a atenuação mínima não difere muito da resposta do filtro exato. É de notar que não existe um correlação muito forte entre o valor do erro total e os erros

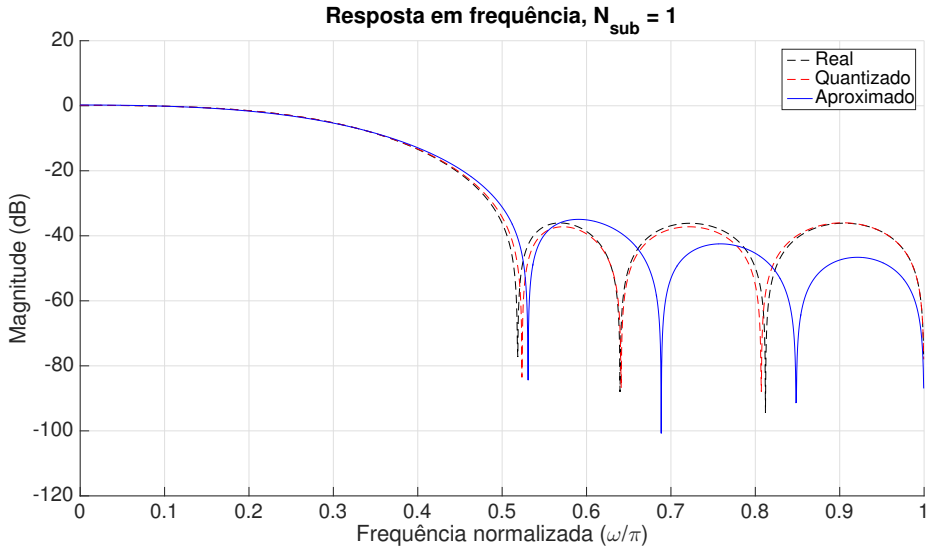


Figura 4.7: Resposta em frequência do filtro FIR aproximado para $N_{sub} = 1$.

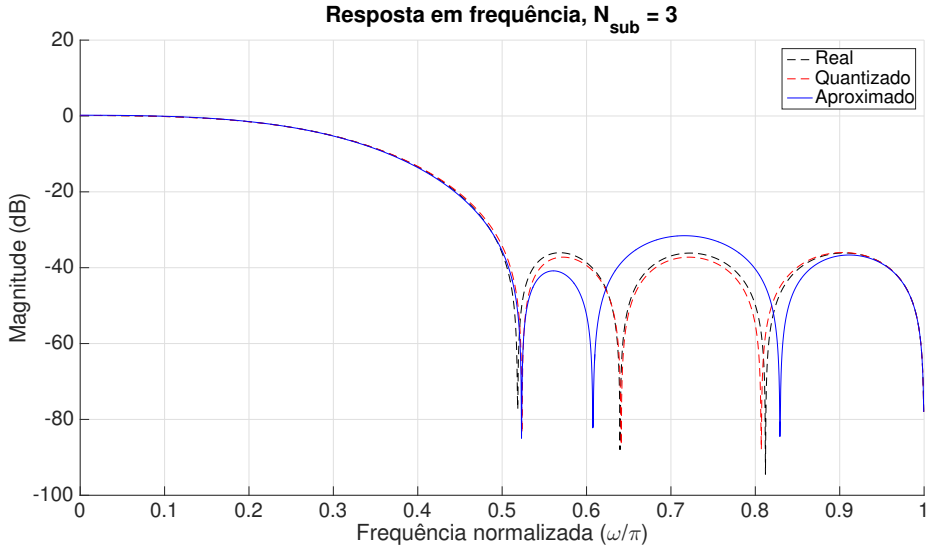


Figura 4.8: Resposta em frequência do filtro FIR aproximado para $N_{sub} = 3$.

nas ondulações, por exemplo no caso de $N_{sub} = 1$ o erro total é de 12 e tem erros nas ondulações maiores do que para $N_{sub} = 3$, onde o erro total é de 20, como se pode verifcar na Tabela 4.1. No entanto, de uma forma geral quando maior o erro total maior os erros nas ondulações.

4.1.3 Algoritmo heurístico

O algoritmo exaustivo torna-se computacionalmente exigente quando o número de somadores do bloco MCM inicial aumenta, visto que o número de combinações a ser testadas aumenta fatorialmente. Para obter soluções para filtros maiores num tempo útil o algoritmo exaustivo proposto foi modificado, introduzindo uma heurística que diminui o número de combinações a ser testadas.

A análise das soluções encontradas pelo algoritmo exaustivo para vários filtros com poucos somadores revelou que, geralmente, as soluções encontradas para $N_{sub} = i + 1$ diferem em apenas de um

somador das soluções encontradas para $N_{sub} = i$, $i = 1, 2, ..., S-1$. As soluções encontradas pelo algoritmo exaustivo para 3 filtros FIR diferentes encontram-se na Tabela 4.2, onde os valores a negrito correspondem aos valores que não estão presentes na solução presente na linha anterior. É possível

Filtro A		Filtro B		Filtro C	
$N = 30$	$bits = 10$	$N = 50$	$bits = 8$	$N = 105$	$bits = 8$
$\omega_p = 0,3$	$\omega_s = 0,5$	$\omega_p = 0,1$	$\omega_s = 0,15$	$\omega_p = 0,2$	$\omega_s = 0,24$
N_{sub}	Somadores eliminados	N_{sub}	Somadores eliminados	N_{sub}	Somadores eliminados
1	3	1	4	1	5
2	3,**6**	2	4,**5**	2	5,**8**
3	**2**,3,6	3	**2**,4,5	3	**3**,5,8
4	2,3,6,**7**	4	2,**3**,4,5	4	3,**4**,5,8
5	2,3,**4**,6,7	5	2,3,4,5,**7**	5	3,4,5,8,**11**
6	2,3,4,**5**,6,7	6	2,3,4,5,**6**,7	6	3,4,5,**6**,8,11
				7	**2**,3,4,**7**,8,9,11
				8	2,3,4,7,8,9,**10**,11
				9	3,4,**5**,**6**,7,8,9,10,11
				10	**2**,3,4,5,6,7,8,9,10,11

Tabela 4.2: Soluções encontradas pelo algoritmo ótimo para três filtros FIR diferentes.

notar que as soluções para valores de N_{sub} consecutivos diferem na maioria dos casos apenas de um elemento e em mais de um elemento em apenas dois casos, que ocorrem no Filtro C de $N_{sub} = 6$ para $N_{sub} = 7$ e de $N_{sub} = 8$ para $N_{sub} = 9$.

Tendo em conta este facto, foi criada a heurística que consiste em executar o algoritmo exaustivo para valores incrementais de N_{sub}, começando em $N_{sub} = 1$ e terminando no número desejado de somadores a remover. Cada vez que o algoritmo exaustivo é executado apenas as combinações que contém todos os somadores eliminados presentes na solução encontrada anteriormente são testadas.

Com esta heurística o número de combinações testadas diminui, encontrando uma solução mais rapidamente. A Figura 4.9 mostra como o número de combinações testadas pelos algoritmos exaustivo e heurístico varia com o número de somadores substituídos, N_{sub}, para um bloco MCM inicial com 50 somadores. Neste caso, apenas para $N_{sub} \geq 48$, o número de combinações testadas pelo o algoritmo heurístico é maior que pelo algoritmo ótimo, no entanto para outros valores de N_{sub} o número de combinações testadas pelo algoritmo heurístico é substancialmente menor, por exemplo para $N_{sub} = 25$ o algoritmo heurístico testa 950 combinações enquanto o algoritmo ótimo testa $\approx 1,26 \times 10^{14}$ combinações.

As soluções encontradas pelo algoritmo heurístico, quando aplicado aos mesmo filtros descritos na Tabela 4.2, são iguais às obtidas pelo algoritmo exaustivo para qualquer valor de N_{sub} nos filtros A, B e C, com exceção para as soluções para $N_{sub} = 7$ e $N_{sub} = 8$ no filtro C. Apesar das soluções serem diferentes das encontradas pelo algoritmo exaustivo, os valores do erro total são iguais, o que nos leva a concluir que para este problema de otimização pode existir mais que uma solução que minimiza o erro total definido em 4.1. No entanto as respostas em frequência das soluções encontradas pelos dois

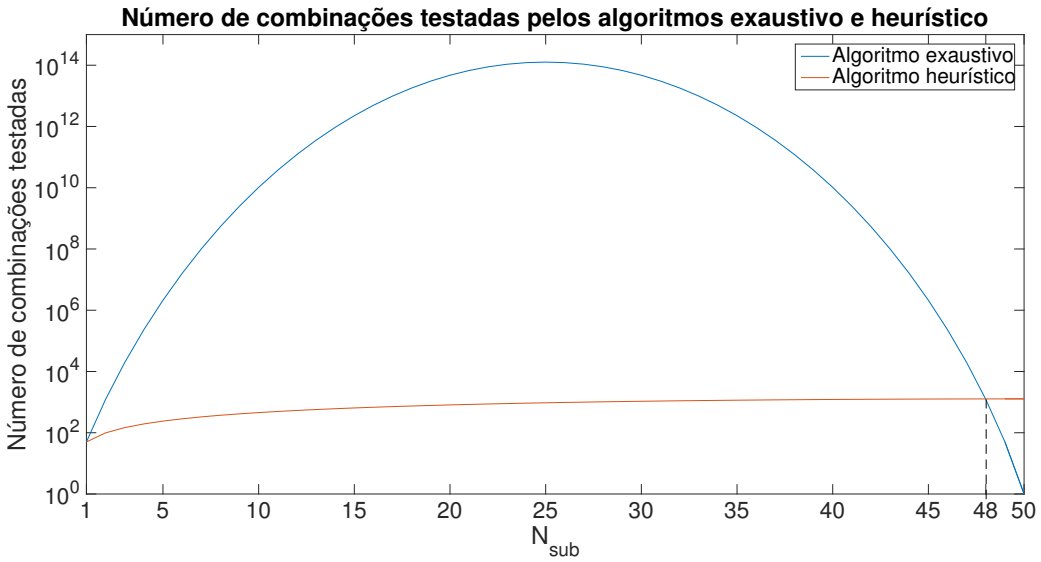


Figura 4.9: Número total de combinações testadas pelos os algoritmos exaustivo e heurístico para um bloco MCM inicial com 50 somadores.

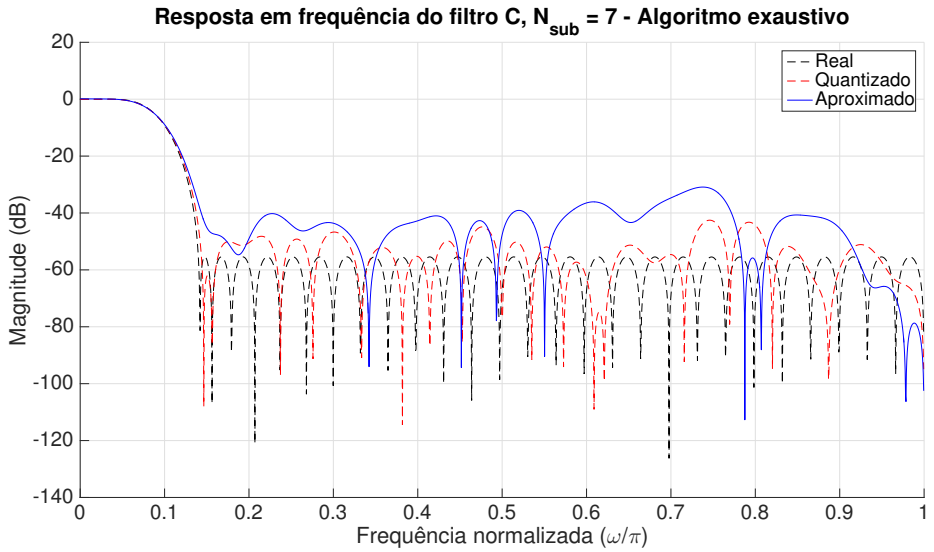


Figura 4.10: Resposta em frequência do filtro C aproximado para $N_{sub} = 7$, obtido pelo algoritmo exaustivo.

algoritmos são diferentes, apesar do erro total ser igual, como verificado nas Figuras 4.10 e 4.11.

As soluções geradas pelos dois algoritmos para estes casos encontram-se na Tabela 4.3.

	Algoritmo exaustivo		Algoritmo heurístico	
N_{sub}	Somadores eliminados	Erro total	Somadores eliminados	Erro total
7	2,3,4,7,8,9,11	38	3,4,**5**,**6**,8,9,11	38
8	2,3,4,7,8,9,10,11	50	3,4,**5**,**6**,8,9,10,11	50

Tabela 4.3: Soluções encontradas pelos algoritmos ótimo e heurístico para o Filtro C.

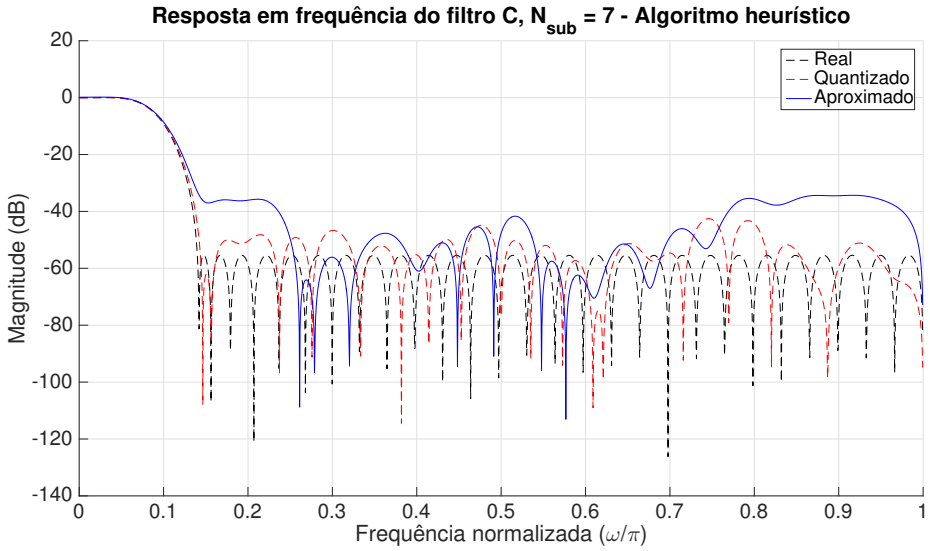


Figura 4.11: Resposta em frequência do filtro C aproximado para $N_{sub} = 7$, obtido pelo algoritmo heurístico.

4.2 Resultados

O algoritmo heurístico foi testado para 10 filtros FIR passa-baixo, de diferentes ordens e especificações. As especificações dos filtros testados estão presentes na Tabela 4.4, onde ω_p e ω_s são as frequências da banda de passagem e banda de atenuação, respetivamente, N a ordem do filtro e # bits o número de bits usados para representar cada coeficiente.

Filtro FIR	ω_p/π	ω_s/π	N	# bits
1	0,300	0,50	30	9
2	0,150	0,25	40	16
3	0,100	0,15	50	8
4	0,042	0,14	60	10
5	0,150	0,20	60	16
6	0,120	0,18	80	16
7	0,120	0,15	80	16
8	0,180	0,2	100	16
9	0,200	0,24	105	8
10	0,200	0,22	120	16

Tabela 4.4: Especificações dos filtros FIR testados.

Primeiro, as implementações de cada bloco MCM, um para cada filtro, foram obtidas seguindo os passos descritos na subsecção 4.1.1. De seguida os filtros foram descritos em VHDL, usando a representação de complemento para dois e definindo a entrada com o mesmo número de bits que os coeficientes e a sua saída com o dobro dos bits da entrada. Depois os filtros foram sintetizados pela ferramenta de síntese Synopys Design Vision ™ versão C-2009.06-SP3, usando a biblioteca de células genéricas Faraday ™ UMC L180, sendo que a escolha da arquitetura dos somadores e a sua otimização foi feita automaticamente pela ferramenta. A área que cada filtro ocupa foi obtida direta-

mente dos relatórios de síntese. A potência de cada circuito sintetizado foi calculada pela ferramenta de síntese, definindo uma probabilidade estática, ou seja a probabilidade de uma entrada ter o valor lógico '1' num ciclo de relógio, de 50% e uma taxa de alternância de 50% para todas as entradas (exceto a entrada do sinal de relógio e entrada de reinicio dos registos). Na Tabela 4.5 está sumarizado os resultados obtidos para a área e potência dinâmica para os filtros exatos, em que δ_p e δ_s representam as ondulações na banda de passagem e atenuação, respetivamente, S representa o número de somadores no bloco MCM e ST o número de somadores totais para realizar o filtro.

Filtro FIR	δ_p (dB)	δ_s(dB)	S	ST	Área (μm^2)	Potência dinâmica (mW)
1	0,0923	-44,13	7	28	21436	35,60
2	0,1027	-38,66	23	62	70055	102,04
3	0,7710	-22,73	7	52	33262	56,28
4	0,0758	-42,53	11	66	52021	88,47
5	0,2343	-31,51	31	90	101008	146,98
6	0,0476	-45,11	38	117	135096	204,97
7	0,3751	-27,45	38	117	137718	203,01
8	0,5219	-24,67	47	146	168647	257,97
9	0,9157	-21,70	7	69	56225	109,98
10	0,3463	-28,13	54	173	201525	310,90

Tabela 4.5: Resultados obtidos para os filtros exatos.

De seguida foi aplicado a cada filtro o algoritmo heurístico, com valores de $N_{sub} = 1, ..., S-1$ e $tol = 0,35$, obtendo múltiplos blocos MCM aproximados para cada filtro. As respostas em frequência de cada filtro aproximado foram obtidas e foram calculados os erros relativos das ondulações na banda de passagem e na banda de atenuação, presentes nas Figuras 4.12, 4.13 e 4.14. A Figura 4.13 corresponde a uma ampliação da Figura 4.12.

Visualizando as Figuras 4.12 e 4.13 pode-se notar que o erro de δ_p, de uma forma geral, aumenta à medida que N_{sub} aumenta, no entanto para valores de $N_{sub}/S \leq 0.2$, ou seja, eliminando menos de 20% dos somadores do bloco MCM, o erro relativo é inferior a 7% (à exceção do filtro 1 que para $N_{sub} = 1$ tem um erro relativo de δ_p de 36%). Para $N_{sub}/S \leq 0.5$ o erro relativo de δ_p é inferior a 50% para qualquer filtro. É de notar que não existe uma correlação forte entre o número de somadores no bloco MCM inicial e o erro relativo de δ_p. No entanto, para este conjunto de filtros, os filtros que tem os maiores erros relativos de δ_p são os que têm menor valor de δ_p no filtro exato, como seria de esperar.

Inspecionando a Figura 4.14 observa-se que o erro relativo máximo de δ_s é inferior a 50% em qualquer filtro aproximado. O erro relativo de δ_s cresce devagar, para $N_{sub}/S \leq 0.7$, mantendo-se inferior a 6,5% nos filtros que têm o valor de $S > 23$. No caso do filtro 9 o valor de δ_s é menor do que no filtro exato para $N_{sub}/S \leq 0.7$. Os filtros que apresentam maiores erros relativos de δ_s são os que no filtro exato respetivo têm o valor de δ_s menor, o que seria de esperar. No entanto, os filtros com um valor de S menor têm maiores erros relativos de δ_s para valores de $N_{sub}/S \leq 0.7$.

Comparando os resultados obtidos para os filtros 1, 4 e 9 com os resultados obtidos pelo algoritmo

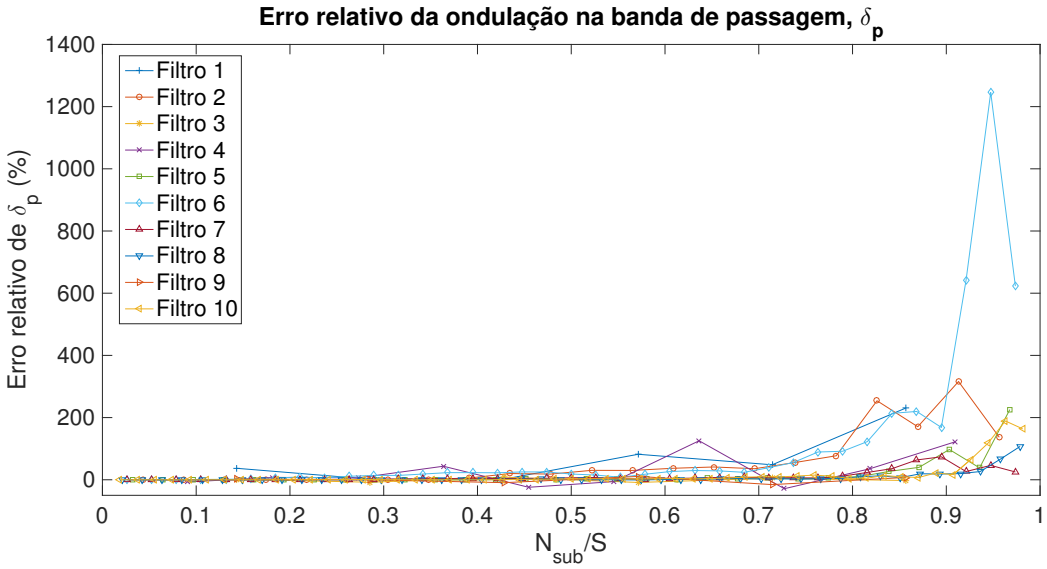


Figura 4.12: Erro relativo da ondulação na banda de passagem, δ_p.

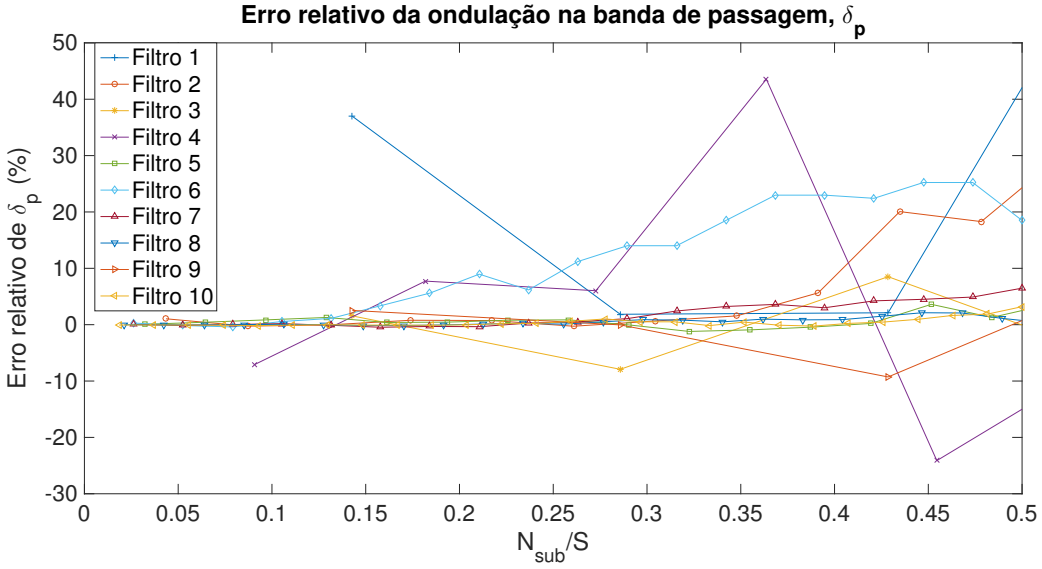


Figura 4.13: Erro relativo da ondulação na banda de passagem para $N_{sub}/S \leq 0.5$.

NAIAD [12] para filtros com as mesmas especificações em termos de N, ω_p, ω_s e número de bits com que os coeficientes são quantificados, o número de somadores totais obtido pelo método apresentado neste capítulo é menor, no entanto as ondulações, tanto na banda de passagem como na banda atenuação são maiores (com exceção para δ_p no filtro 4), como se pode verificar na Tabela 4.6. Por exemplo no caso do filtro 1, o número de somadores totais obtidos pelo método proposto, para $N_{sub} = 3$ é de 25, enquanto o algoritmo NAIAD obtém uma solução com 30 somadores no total, no entanto o filtro obtido por este algoritmo tem $\delta_p = 0,0274$dB e $\delta_s = -50,00$dB e o filtro obtido pelo método proposto neste capítulo tem $\delta_p = 0,0943$ dB e $\delta_s = -39,37$ dB.

Outra de forma a avaliar a qualidade das respostas dos filtros FIR aproximados é calcular o valor da relação sinal-ruído (SNR) para cada um deles. Este valor foi obtido gerando aleatoriamente um sinal de ruído branco gaussiano com 2000 amostras e filtrando-o usando os filtros exatos e aproximados.

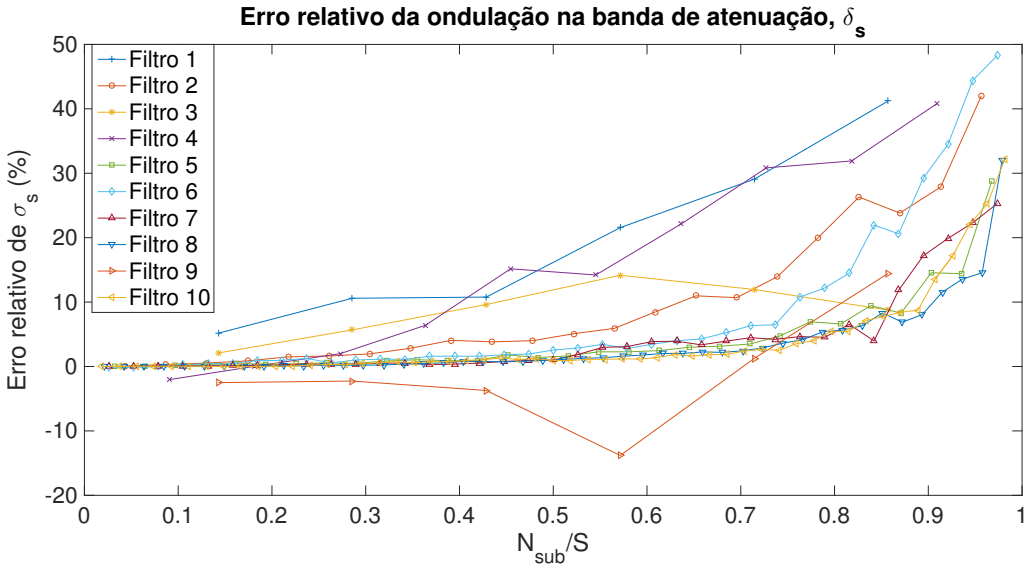


Figura 4.14: Erro relativo da ondulação na banda de atenuação.

	NAIAD			Método aproximado		
Filtro	ST	δ_p(dB)	δ_s(dB)	ST	δ_p(dB)	δ_s(dB)
1($N_{sub} = 3$)	30	0.0274	-50	25	0.9530	-39.37
4($N_{sub} = 2$)	72	0.1036	-60	64	0.0575	-42.52
9($N_{sub} = 5$)	109	0.0864	-40	64	0.7739	-21.43

Tabela 4.6: Comparação com o algoritmo NAIAD [12].

O ruído foi então definido como a diferença entre o sinal filtrado usando um filtro aproximado e o sinal filtrado com o filtro exato respetivo. Este processo foi repetido 10 vezes para cada filtro aproximado, obtendo assim um valor da relação sinal-ruído médio. Os valores da relação sinal-ruído de cada filtro aproximado encontram-se na Figura 4.15, onde é possível verificar que o valor de SNR diminui com o aumento de N_{sub}, como seria esperado. É possível notar também que os filtros aproximados com o valor de S pequeno (< 23) tem valores de SNR muito inferiores aos dos outros filtros. Os filtros que possuem valores de SNR mais elevados são os que contém mais somadores no bloco MCM inicial, pois quando um somador é eliminado existem mais alternativas para substituí-lo, o que se traduz num menor erro na substituição.

Os filtros aproximados cujo SNR é próximo de 90, 80, 70, 60, 50, 40 ou 30 dB foram descritos em VHDL e sintetizados da mesma forma que os filtros exatos, obtendo os valores de área e potência, apresentados na Tabela 4.7. Analisando os resultados, podemos observar que os maiores ganhos em área e potência, para o menos nível de SNR, ocorrem nos filtros em que $S > 11$, por exemplo para um valor de SNR próximo de 40 dB os filtros com mais que 11 somadores no bloco MCM inicial têm uma redução média na área de 21,8% e na potência de 12,6% enquanto os restantes filtros apenas têm reduções médias na área de 3,4% e na potência de 1,7%. A maior redução na área obtida, para um valor de SNR de 50 dB, foi de 19% e na potência de 12,18%. Para um nível de SNR maior que 65 dB, as reduções na área e na potência são inferiores a 5% e 3%, respetivamente.

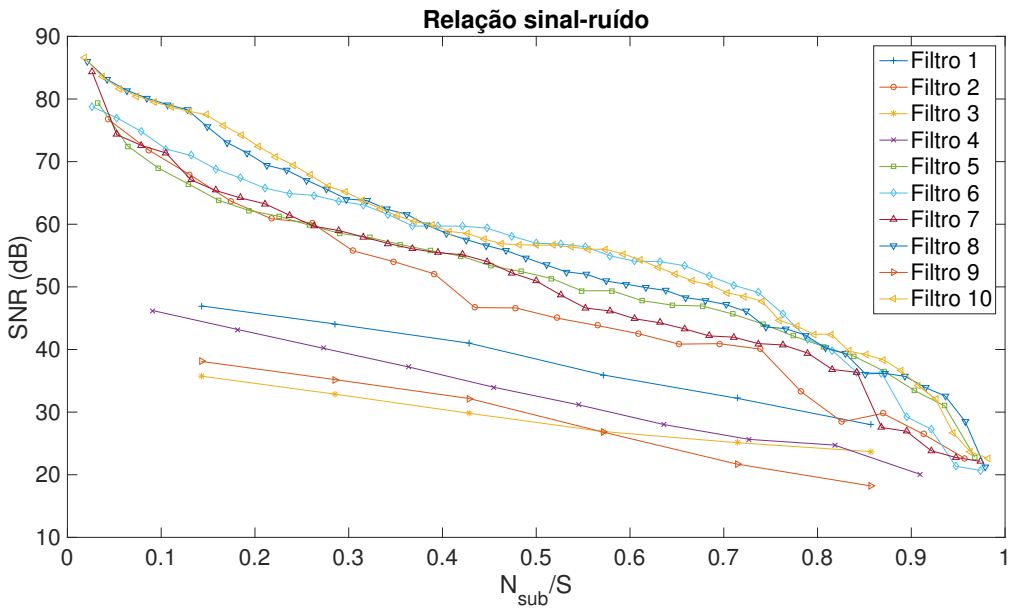


Figura 4.15: Relação sinal-ruído para diferentes aproximações.

Com este método pode-se conseguir reduções significativas de área e potência à custa da degradação da qualidade do filtro. No entanto, devido a ser uma técnica de aproximação a nível arquitetural de eliminação e substituição de operadores das somas parciais por outros valores aproximados a degradação do filtro é significativa para uma pequena redução da área ou potência. Por isso no próximo capítulo será abordada uma técnica de aproximação a nível lógico de forma a reduzir o erro introduzido no cálculo de cada soma parcial.

	N_{sub}	SNR (dB)	Red. na área (%)	Red. na potência (%)
Filtro 1				
	1	46,9	2,16	1,52
	3	41,0	8,20	4,23
	6	28,0	15,54	6,42
Filtro 2				
	1	76,8	1,73	0,98
	3	67,9	4,95	2,72
	6	60,1	8,71	4,70
	9	52,1	12,66	7,24
	17	40,1	24,58	12,97
	20	29,8	34,96	22,59
Filtro 3				
	1	35,7	1,09	0,16
	3	29,8	1,34	1,48
Filtro 4				
	1	46,2	0,80	1,19
	3	40,2	3,29	2,29
	6	31,2	3,31	3,31
Filtro 5				
	1	79,4	0,73	0,66
	3	69,0	2,46	1,73
	8	59,8	6,25	3,64
	18	49,4	14,66	4,51
	25	40,5	22,05	11,37
	29	31,0	32,09	19,71
Filtro 6				
	1	78,8	0,46	0,38
	5	71,0	2,70	1,92
	16	59,7	9,06	5,92
	27	50.3	19,00	12,18
	30	41,7	21,39	13,29
	34	29,3	25,10	13,54
Filtro 7				
	1	84,3	0,58	0,49
	4	71,4	2,67	1,55
	10	59,7	6,12	3,42
	19	50,9	12,72	11,03
	30	39,4	20,02	13,51
	33	27,5	28,02	15,60
Filtro 8				
	1	86,0	0,53	-0.03
	5	79,1	2,14	0,98
	10	69,4	4,48	2,81
	18	59,9	8,55	4,54
	30	49,4	15,56	8,19
	39	39,3	21,40	11,80
	45	28,5	32,97	20,84
Filtro 9				
	1	38,1	1,09	0,16
	3	32,2	1,34	1,48
Filtro 10				
	1	86,5	0,52	0,45
	4	80,5	1,64	1,13
	12	70,8	5,14	3,00
	20	60,4	8,63	4,61
	37	50,3	15,65	8,53
	45	39,8	21,74	11,52
	50	32,1	26,02	12,64

Tabela 4.7: Resultados da síntese dos filtros aproximados.

Capítulo 5

Fitros FIR - Computação aproximada ao nível lógico

Neste capítulo é apresentado um método para realizar filtros FIR em *hardware*, que tem como objetivo principal reduzir a área de implementação utilizando somadores aproximados, tanto no bloco MCM do filtro como na linha de atraso, minimizando o ruído, introduzido pelas aproximações, na saída do filtro. Este capítulo está dividido em três secções. Na primeira secção é apresentado o somador aproximado utilizado e as suas características, na segunda secção é introduzido o método desenvolvido para a substituição de somadores exatos por aproximados. Na última secção são apresentados os resultados obtidos pelo método para 10 filtros FIR passa-baixo diferentes.

5.1 Somador aproximado

O somador aproximado utilizado corresponde ao somador de cópia, descrito na subsecção 2.2.2. Foi este o somador aproximado escolhido pois é baseado num somador RCA, que é o somador convencional que requer menos área para ser implementado, e a aproximação utilizada é a que reduz mais a área, visto que não são necessárias portas lógicas para implementar a parte aproximada do somador. Além de reduzir a área necessária à implementação, este somador aproximado diminui também o atraso, quando comparado ao somador RCA exato, visto que a cadeia de *carry* é reduzida. Na figura 5.1 está ilustrado o somador aproximado utilizado, onde k corresponde ao número de bits aproximados e n ao número de bits total.

Um exemplo da soma efetuada por um somador aproximado deste tipo, com 4 bits e $k = 2$ está representado na Figura 5.2. Neste exemplo os operandos A e B correspondem aos valores numéricos 11 e 3, respetivamente, e o resultado obtido, representado por S, corresponde ao valor 15. O erro, definido como a diferença entre o resultado aproximado e o resultado exato, introduzido pelo somador aproximado neste exemplo é 1, pois o resultado exato seria 14.

O erro introduzido por este somador pode ter origem em duas situações diferentes. Pela cópia dos k bits menos significativos do operando A para o resultado e/ou pelo o valor de *carry* de entrada

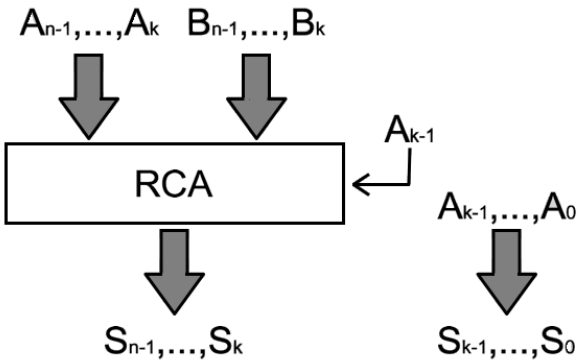


Figura 5.1: Somador de cópia.

no somador RCA. A operação de cópia dos k bits menos significativos do operando A para os bits correspondentes do resultado, pode ser vista como uma soma convencional em que os k bits menos significativos do operando B têm todos o valor lógico '0', e portanto o *carry* usado no cálculo do bit k do resultado, se esta operação se trata-se de uma soma exata, seria sempre '0'. Logo para obter o resultado do somador aproximado apenas é necessário somar o valor do bit $k-1$ do operando A, deslocado de um bit à esquerda. No caso do exemplo anterior a soma usando o somador aproximado pode ser representada na forma ilustrada na Figura 5.3, em que B' corresponde ao operando B com os k bits menos significativos colocados a '0' e C ao valor somado devido ao *carry* de entrada da parte exata deste somador.

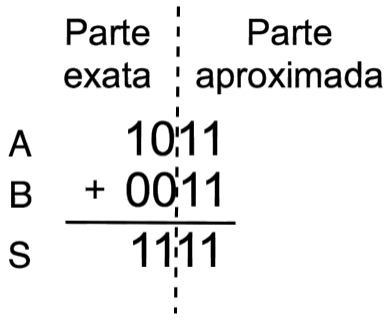


Figura 5.2: Soma de 11 com 3 usando um somador de cópia de 4 bits e $k = 2$.

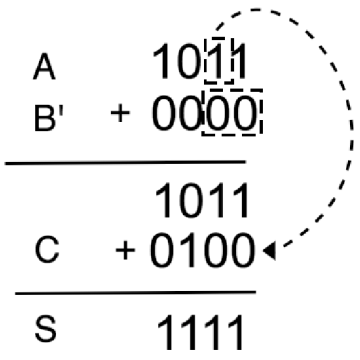


Figura 5.3: Soma de 11 com 3 usando um somador de cópia de 4 bits e $k = 2$.

De forma a caracterizar o erro introduzido pelo somador de n bits aproximado com k bits, este somador foi modelado usando as seguintes expressões:

$$s = a + b + e(A, B), \tag{5.1}$$

$$a = \sum_{i=0}^{n-1} A_i 2^i, \tag{5.2}$$

$$b = \sum_{i=0}^{n-1} B_i 2^i, \tag{5.3}$$

$$e(A, B) = A_{k-1} 2^k - \sum_{i=0}^{k-1} B_i 2^i, \tag{5.4}$$

onde s é valor numérico do resultado do somador aproximado, $e(A, B)$ é a função de erro e A e B são os operandos binários do somador aproximado.

Sabendo que cada bit dos operandos tem probabilidade de 50% de ter o valor lógico '1', isto é, o seu valor esperado é igual a $0,5$, podemos calcular o erro médio neste somador:

$$\begin{aligned} E(e(A, B)) &= E(A_{k-1}) 2^k - \sum_{i=0}^{k-1} E(B_i) 2^i \\ &= 2^{k-1} - 2^{-1} \sum_{i=0}^{k-1} 2^i \\ &= 2^{k-1} - 2^{-1} \left(\frac{2^k - 1}{2 - 1} \right) \\ &= 2^{k-1} - 2^{k-1} + 2^{-1} = 0,5. \end{aligned} \tag{5.5}$$

Podemos concluir que o erro médio deste somador é constante e igual a $0,5$, não dependendo do parâmetro k. No entanto o erro máximo e mínimo depende de k, como se pode verificar nas expressões seguintes:

$$max(e(A, B)) = 2^k, \tag{5.6}$$

$$min(e(A, B)) = -\sum_{i=0}^{k-1} 2^i = -2^k + 1. \tag{5.7}$$

Tanto o erro máximo como o erro mínimo aumentam exponencialmente, em valor absoluto, com o aumento de k.

Num bloco MCM as entradas de cada somador podem ter sido deslocadas para a esquerda anteriormente, e portanto é necessário ter este facto em conta no modelo do somador aproximado. Consideremos de novo o exemplo da soma exata, entre 11 e 3, em que o operando B foi anteriormente deslocado à esquerda de 2 bits, ilustrada na Figura 5.4. Apesar do operando B ter 6 bits apenas é necessário utilizar um somador de 4 bits para realizar esta soma, porque o operando B terá sempre o valor lógico '0' nos 2 bits menos significativos, o que significa que os dois bits menos significativos do resultado correspondem aos dois bits menos significativos do operando A. Se alterarmos o somador exato pelo o somador aproximado com $k = 2$, os 4 bits menos significativos do operando A seriam copiados para

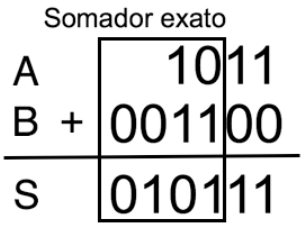


Figura 5.4: Soma exata em que o operando B foi deslocado à esquerda de 2 bits.

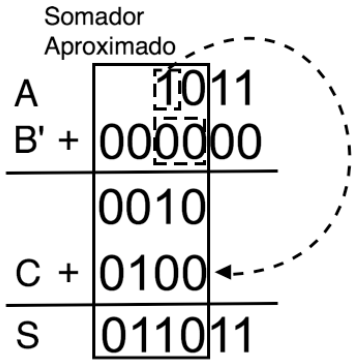


Figura 5.5: Soma aproximada em que o operando B foi deslocado à esquerda de 2 bits.

o bits correspondentes do resultado e seria adicionado o valor do bit $k+1$ do operando A, deslocado de três bits para a esquerda, devido ao deslocamento de dois bits do operando B. A soma realizada pelo somador aproximado encontra-se ilustrada na Figura 5.5. Neste exemplo o erro introduzido pelo somador aproximado é igual a 4 $(27-23)$.

De forma a contabilizar estes casos a função de erro no modelo do somador foi alterada para a seguinte expressão:

$$e(A,B) = A_{k+s_B-1}2^{k+s_B} - \sum_{i=s_B}^{k+s_B-1} B_i 2^i, \tag{5.8}$$

onde s_B corresponde ao número de bits que o operando B foi deslocado para a esquerda. É imposto que o operando B seja o que sofreu o deslocamento, de forma a simplificar o modelo.

Com esta nova expressão para a função de erro obteve-se as seguintes expressões para o erro médio, máximo e mínimo:

$$E(e(A,B)) = 2^{s_B-1}, \tag{5.9}$$

$$max(e(A,B)) = 2^{k+s_B}, \tag{5.10}$$

$$min(e(A,B)) = -\sum_{i=s_B}^{k+s_B-1} 2^i = -2^{s_B}\left(2^k-1\right). \tag{5.11}$$

Podemos notar que o erro médio já não é constante, pois depende do número de bits que o operando B for deslocado. O erro máximo e mínimo aumentam em valor absoluto com o aumento de s_B, mantendo k fixo.

O modelo com a função de erro (5.8) apenas é válido para o caso de se realizar uma soma, no

entanto no bloco MCM é necessário realizar operações de subtração. Com somadores convencionais, e utilizando uma representação em complemento para 2, as subtrações são realizadas negando um dos operandos e colocando o valor lógico '1' no *carry* de entrada do somador exato. No entanto, o somador aproximado não possui entrada de *carry* no primeiro bit menos significativo, pelo que não é possível realizar a subtração da mesmo forma que se realizaria com um somador convencional. Para realizar a subtração com o somador aproximado apenas uma das entradas é negada, desprezando o *carry* de entrada, ou seja, estamos a permitir que seja introduzido um erro de -1 num dos operandos. De forma a simplificar o modelo é considerado que o operando B é o operando que é negado. Tal como no somador, os bits que são copiados para o resultado pertencem ao operando sem deslocamento. O modelo para o subtrator é descrito pelas seguintes expressões:

$$s = a - b + e^{-}(A, B), \tag{5.12}$$

$$a = \sum_{i=0}^{n-1} A_i 2^i, \tag{5.13}$$

$$b = \sum_{i=0}^{n-1} B_i 2^i, \tag{5.14}$$

$$e^{-}(A,B) = \begin{cases} -2^{s_B} + A_{k+s_B-1}2^{k+s_B} - \sum_{i=s_B}^{k+s_B-1} \overline{B}_i 2^i, & \text{se } s_B \neq 0, s_A = 0 \\ -1 + \overline{B}_{k+s_A-1}2^{k+s_A} - \sum_{i=s_A}^{k+s_A-1} A_i 2^i, & \text{se } s_A \neq 0, s_B = 0 \end{cases} \tag{5.15}$$

onde, s_A corresponde ao número de bits que o operando A foi deslocado para a esquerda.

As expressões para o erro médio, máximo e mínimo, para o caso do subtrator são obtidas de forma semelhante e são as seguintes:

$$E(e^{-}(A,B)) = \begin{cases} -2^{s_B-1}, & \text{se } s_B \neq 0, s_A = 0 \\ -1 + 2^{s_A-1}, & \text{se } s_A \neq 0, s_B = 0, \end{cases} \tag{5.16}$$

$$max(e^{-}(A,B)) = \begin{cases} 2^{s_B}\left(2^k - 1\right), & \text{se } s_B \neq 0, s_A = 0 \\ -1 + 2^{k+s_A}, & \text{se } s_A \neq 0, s_B = 0, \end{cases} \tag{5.17}$$

$$min(e^{-}(A,B)) = \begin{cases} -2^{k+s_B}, & \text{se } s_B \neq 0, s_A = 0 \\ -1 - 2^{k+s_A} + 2^{s_A}, & \text{se } s_A \neq 0, s_B = 0. \end{cases} \tag{5.18}$$

Podemos notar que, no caso de $s_B \neq 0, s_A = 0$, o erro médio do subtrator é simétrico ao erro médio no somador, o erro máximo no subtrator é simétrico ao erro mínimo do somador e o erro mínimo no subtrator é simétrico ao erro máximo do somador. Este resultado leva-nos a concluir que numa cadeia de somadores e subtratores aproximados os erros introduzidos poderão se anular. Um exemplo de uma cadeia com um somador e um subtrator aproximados, que faz o cálculo de $(A + 2B) - C$, em que o erro se anula para alguns casos está representada na Figura 5.6. No caso de $A = 28$, $B = 14$ e $C = 12$, o

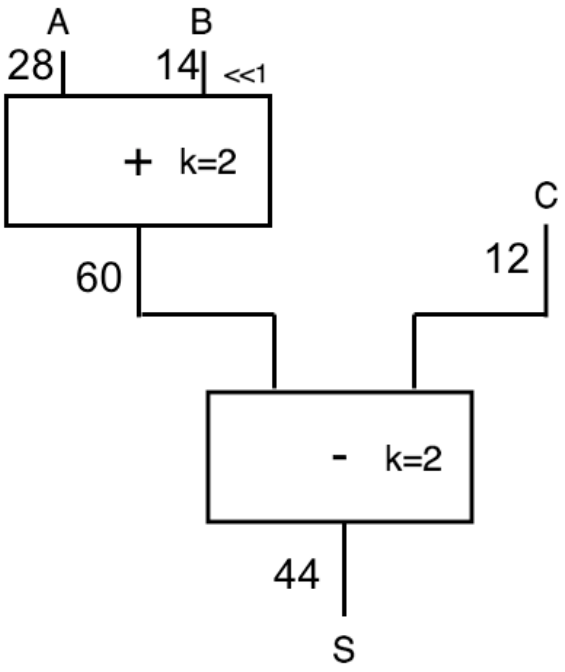


Figura 5.6: Cadeia com um somador aproximado e um subtrator aproximado.

resultado S calculado (44) pela cadeia de somadores na Figura 5.6 estaria correto.

5.2 Método proposto

O método proposto para a implementação de filtros FIR utilizando a computação aproximada ao nível lógico, pretende reduzir a área necessária à implementação do filtro, maximizando o uso de somadores aproximados, bem como o número de bits na parte aproximada de cada somador aproximado, tanto no bloco MCM como na linha de atraso do filtro, mas garantindo que a qualidade do filtro não é muito degradada.

Note-se que os somadores aproximados a serem utilizados para implementar o filtro não são lineares, visto que a seguinte propriedade não é válida para todos operandos possíveis:

$$\alpha * a + \alpha * b = \alpha(a + b), \tag{5.19}$$

em que α é uma constante e a e b os valores numéricos dos operandos A e B, respetivamente. Assim, os filtros implementados usando estes somadores serão então filtros não lineares, pelo que não é possível obter a sua resposta em frequência e comparar as suas especificações com o filtro exato respetivo. Portanto de forma a medir a qualidade do filtro aproximado foi utilizada a medida da relação sinal-ruído (SNR), definindo o ruído como a diferença entre a resposta do filtro aproximado e do filtro exato ao mesmo sinal de entrada. O sinal de entrada utilizado para medir a relação sinal-ruído foi um sinal com 2000 amostras, correspondente a ruído branco gaussiano.

À semelhança do método proposto no Capítulo 4, que faz uso da computação aproximada ao nível

arquitetural, o método apresentado neste capítulo começa com uma implementação de um filtro FIR já existente. O bloco MCM desta implementação é obtido seguindo os mesmo passos presentes na subsecção 4.1.1. Após obter a implementação do filtro exato um algoritmo de procura é usado para encontrar uma implementação que faça uso dos somadores aproximados e que satisfaça o nível de relação sinal-ruído pretendido. O algoritmo usado para determinar os somadores/subtratores a serem aproximados e com que valor de k será explicado na próxima subsecção.

5.2.1 Algoritmo de procura

O algoritmo apresentado nesta subsecção tem como objetivo encontrar uma implementação, tendo como base a implementação do filtro exato, que use os somadores aproximados apresentados anteriormente, e ao mesmo tempo que tenha um valor da relação sinal-ruído pretendido. A implementação do filtro aproximado é obtida trocando os somadores exatos, da implementação do filtro exato, por somadores aproximados. Para obter um filtro aproximado com um nível de SNR pretendido é necessário descobrir que somadores exatos podem ser trocados por somadores aproximados, e quantos bits aproximados cada somador aproximado deve ter.

Se considerarmos todas as hipóteses possíveis para quantos somadores aproximados usar, em que sítio usá-los e quantos bits aproximados usar em cada um deles, podemos inferir que um algoritmo que considere todas estas variáveis seria computacionalmente exigente. Por exemplo, considerando um filtro com 10 somadores de 8 bits no total, o que seria um filtro pequeno, o número de hipóteses a testar seria de $\approx 1,23 \times 10^9$, sendo que em cada uma das hipóteses era necessário calcular o valor de SNR, o que implica o cálculo da resposta do filtro aproximado a um sinal gerado aleatoriamente. De modo a reduzir o número de hipóteses a testar os somadores foram agrupados em diferentes grupos, sendo que dentro do mesmo grupo todos os somadores têm o mesmo número de bits aproximados, k_g. Um dos grupos criados contêm todos os somadores na linha de atraso do filtro e os outros grupos são criados tendo em conta o nível de profundidade dos somadores no bloco MCM, sendo que somadores com o mesmo nível de profundidade estão incluídos no mesmo grupo. A troca dos somadores exatos por somadores aproximados é então feita grupo a grupo.

O primeiro passo do algoritmo é criar uma simulação do filtro que tenha como parâmetros o número de bits aproximados em cada grupo de somadores, de forma a poder calcular o SNR de cada filtro aproximado diferente. Para realizar esta simulação foi definida uma função, chamada de $SomadorAprox$, que utiliza os modelos do somador e subtrator aproximado apresentados na secção anterior, e retorna o valor calculado por um somador/subtrator aproximado. Esta função tem como parâmetros:

- a - valor do operando A, antes de ser deslocado,
- b - valor do operando B, antes de ser deslocado,
- s_a - número de bits que o operando A é deslocado,
- s_b - número de bits que o operando B é deslocado,
- op - operação efetuada (1 - soma, -1 - subtração),

- k - número de bits aproximados,

- n - número de bits total.

Quando o parâmetro $k = 0$, a função $SomadorAprox$, retorna o valor exato da soma (ou subtração). Utilizando a função $SomadorAprox$ podemos definir os vários somadores (e subtratores) existentes na implementação do filtro e calcular a resposta do filtro para diferentes hipóteses de aproximação.

Tendo a simulação do filtro é então aplicado um processo iterativo de forma a descobrir um conjunto de valores de k dos diferentes grupos, que satisfazem o nível de SNR pretendido. Os valores dos diferentes k são guardados num vetor linha chamado $aprox$, em que o primeiro elemento corresponde ao valor de k do grupo que contêm os somadores na linha de atraso e os seguintes elementos aos valores de k dos restantes grupos, ordenados pela ordem crescente de profundidade. O processo iterativo é efetuado da seguinte forma:

- 1. Todos os somadores do filtro começam por ser somadores exatos, ou seja, com $k = 0$ em todos os grupos .

- 2. O valor de k é incrementado por 1 num dos grupos, mantendo o valor de k nos restantes grupos.

- 3. É calculado o valor de SNR.

- 4. Repetir os pontos 2. e 3. para todos os grupos, e guardar a aproximação com melhor SNR.

- 5. Começando com a aproximação guardada, repetir 2., 3. e 4. até o valor de SNR atingir o valor de SNR pretendido.

O valor máximo de k para cada grupo corresponde ao número de bits com que os coeficientes foram representados, visto que este número corresponde ao número mínimo de bits que um somador pode ter na arquitetura do filtro. Quando o valor de k de um grupo atinge o valor máximo este grupo é ignorado, ou seja, o seu valor não é mais incrementado, nas iterações posteriores.

De forma a tornar este processo mais claro, tomemos o exemplo para um filtro de ordem $N = 10$, $\omega_s = 0.1\pi$, $\omega_p = 0.5\pi$ e coeficientes quantificados com 10 bits, em que na sua implementação o bloco MCM tem um profundidade máxima de 3 e se pretende obter um filtro aproximado com SNR próximo de 60 dB. A ordem de hipóteses testadas (vetor $aprox$), bem como o SNR correspondente, está presente na Tabela 5.1. Neste caso o filtro aproximado com SNR mais próximo 60 dB e que tem mais bits aproximados será o que tem o vetor $aprox = [5, 0, 1, 2]$, ou seja os somadores na linha de atraso são aproximados e têm 5 bits na parte aproximada, os somadores no bloco MCM com profundidade 1 são exatos, os com profundidade 2 são aproximados e têm 1 bit na parte aproximada e os com profundidade 3 são aproximados com parte aproximada de 2 bits.

Iteração	Hipótese testada ($aprox$)	SNR(dB)	
1	[1, 0, 0, 0]	**90,2**	← melhor aproximação
	[0, 1, 0, 0]	39,3	
	[0, 0, 1, 0]	61,7	
	[0, 0, 0, 1]	67,7	
2	[2, 0, 0, 0]	**80,8**	← melhor aproximação
	[1, 1, 0, 0]	38,8	
	[1, 0, 1, 0]	61,2	
	[1, 0, 0, 1]	66,8	
		⋮	
6	[6, 0, 0, 0]	60,8	
	[5, 1, 0, 0]	39,6	
	[5, 0, 1, 0]	60,0	
	[5, 0, 0, 1]	**62,7**	← melhor aproximação
		⋮	
8	[6, 0, 0, 2]	58,4	
	[5, 1, 0, 2]	40,4	
	[5, 0, 1, 2]	**60,8**	← melhor aproximação
	[5, 0, 0, 3]	57,4	
9	[6, 0, 1, 2]	56,5	$SNR < 60dB$
	[5, 1, 1, 2]	38,6	Algoritmo pára
	[5, 0, 2, 2]	53,6	
	[5, 0, 1,3]	55,1	

Tabela 5.1: Iterações do algoritmo de procura para um filtro de ordem 10.

5.3 Resultados

O método apresentado neste capítulo foi testado para os 10 filtros FIR passa-baixo tal como no capítulo anterior. As especificações destes filtros estão presentes na Tabela 4.4 e o resultado para a área e potência dinâmica estão apresentados na Tabela 4.5.

Para cada filtro exato foram obtidos diferentes implementações de filtros aproximados, com níveis de SNR próximos de 80, 70, 60, 50, 40 e 30 dB. Cada filtro aproximado foi descrito em VHDL, utilizando a representação de complemento para dois, realizando as negações existentes no bloco MCM com somadores exatos, e definindo a entrada do filtro com o mesmo número de bits que os coeficientes e a saída do filtro com o dobro dos bits que a entrada. Estes filtros foram sintetizados e otimizados pela ferramenta de síntese e foi calculada a área e a potência para cada filtro aproximado, do mesmo modo que para os filtros exatos. Na Tabela 5.2 encontra-se os resultados obtidos para a área e potência dos filtros aproximados sintetizados.

Para os filtros com coeficientes quantificados com 16 bits (filtros 2, 5, 6, 7, 8 e 10), foi obtida uma redução média na área de 29,7% e na potência de 21,2% para um nível de SNR de 80 dB. Nos filtros

com coeficientes quantificados com menos de 16 bits (filtros 1, 3, 4 e 9) o nível de SNR máximo obtido é inferior a 70 dB, e a redução média de área e potência para um nível de SNR de 40 dB foi de 23,3% e 17,2%, respetivamente. Nos filtros 3 e 9, ambos com coeficientes com 8 bits, a redução de área e potência foi menor do que nos outros filtros, tendo uma redução máxima de 15,19% na área e 10,49% na potência para um SNR de 40 dB. É possível notar na Tabela 5.2 que existem alguns valores negativos para a redução da potência. Este valores ocorrem quando a redução da área é menor que 5%. Os resultados negativos nas reduções da potência poderão ter ocorrido devido à forma como a potência dinâmica foi calculada pela ferramenta de síntese, pelo que seria de esperar que ao reduzir a área do circuito a potência também diminuísse.

As maiores reduções de área e potência foram obtidas para os filtros com coeficientes de 16 bits, pelo que se pode deduzir que os erros introduzidos pelos somadores aproximados nos bits menos significativos do resultado influenciam menos a qualidade nestes filtros comparativamente aos filtros com coeficientes quantificados com menor número de bits.

Observando a Tabela 5.3, onde estão indicados o número de bits aproximados nos diferentes grupos de somadores dos filtros aproximados sintetizados, pode-se concluir que as maiores reduções de área acontecem nos somadores da linha de atraso, porque além de ser o grupo onde são permitidos o maior número de bits aproximados é geralmente o grupo que contém o maior número de somadores. É possível verificar também que geralmente os somadores do bloco MCM com menor profundidade têm menos bits aproximados ou são exatos, pelo que se conclui que erros neste somadores são posteriormente amplificados pelos vários deslocamentos existentes ao longo do bloco MCM, gerando erros maiores que afetam demasiado a qualidade do filtro.

O método introduzido neste capítulo consegue reduções maiores, tanto na área como na potência, comparativamente ao resultados apresentados em [10], em que foram obtidas reduções médias na área de 18,8 % e 15,5% na potência, para filtros com coeficientes quantificados com 16 bits e SNR de 80 dB. Esta melhoria deve-se ao facto de o método apresentando neste capítulo considerar diferentes aproximações para os somadores no bloco MCM enquanto em [10], os somadores deste bloco possuem todos o mesmo número de bits aproximados.

O método apresentado neste capítulo obteve melhores resultados para a redução de área e potência do que o método de computação aproximada ao nível arquitetural proposto, para o mesmo nível de relação sinal-ruído. Por exemplo para um nível de SNR de 40 dB o método ao nível lógico apresenta uma redução média na área e potência de 41,94% e 33,23%, respetivamente, enquanto o método a nível arquitetural obteve reduções médias na área de 14,49% e na potência de 8,13%. Isto deve-se ao facto de que o método ao nível lógico também considera os somadores na linha de atraso do filtro, onde mais somadores estão disponíveis, que podem ser aproximados sem afetar demasiado a qualidade do filtro.

	SNR (dB)	Red. na área (%)	Red. na potência (%)
Filtro 1			
	68,7	1,06	-2,78
	59,0	15,54	14,00
	49,1	21,17	18,88
	40,0	32,06	29,70
	30,5	39,44	37,29
Filtro 2			
	79,2	31,32	21,69
	68,2	37,82	29,06
	59,6	56,65	44,90
	49,5	56,67	43,93
	39,7	54,98	43,06
	29,6	54,94	42,97
Filtro 3			
	51,1	2,01	-2,87
	40,7	6,67	1,14
	31,7	25,53	23,51
Filtro 4			
	72,8	1,30	-6,42
	64,8	4,97	-2,64
	51,9	19,16	12,25
	37,3	39,36	27,50
	29,6	46,20	34,92
Filtro 5			
	76.0	32,12	26,98
	68.2	37,05	25,27
	58.4	42,57	31,53
	49.9	46,56	37,09
	38.9	47,37	35,96
	31.0	48,21	37,00

	SNR (dB)	Red. na área (%)	Red. na potência (%)
Filtro 6			
	77,8	31,20	22,15
	66,3	44,18	33,73
	58.6	56,35	45,55
	48,8	58,38	48,08
	37,3	57,32	46,91
	29,5	55,20	45,10
Filtro 7			
	75,2	32,06	20,69
	67,7	35,71	24,76
	57,6	44,46	35,95
	49,0	47,02	39,54
	39,5	48,04	40,54
	29,9	49,16	41,66
Filtro 8			
	81,8	27,13	21,17
	75,3	31,29	23,17
	63,2	43,84	31,19
	48,4	63,15	52,22
	39,8	60,60	49,67
	29,5	57,41	46,78
Filtro 9			
	40,0	15,19	10,49
	28,8	28,11	23,58
Filtro 10			
	84,5	24,51	14,77
	74,1	30,70	20,95
	62,1	38,10	30,65
	47,5	57,64	47,69
	39,2	57,77	47,34
	29,7	57,56	47,18

Tabela 5.2: Resultados da síntese dos filtros de teste usando somadores aproximados.

	SNR (dB)	# bits aproximados em cada grupo
Filtro 1		
	70	[1,0,0,0]
	60	[4,0,0,0]
	50	[5,0,0,3]
	40	[7,0,0,4]
	30	[8,0,2,6]
Filtro 2		
	80	[12,0,3,2,6,8,6,8]
	70	[14,0,5,4,8,10,7,11]
	60	[16,2,6,6,10,11,8,12]
	50	[16,4,8,8,11,13,10,14]
	40	[16,6,10,9,13,15,12,15]
	30	[16,7,11,11,15 ,16,14,16]
Filtro 3		
	50	[1,0,0]
	40	[2,0,0]
	30	[5,0,0]
Filtro 4		
	70	[1,0,0]
	60	[2,0,0]
	50	[5,0,0]
	40	[7,1,0]
	30	[8,2,3]
Filtro 5		
	80	[13,0,0,0,2,11]
	70	[14,1,2,0,3,12]
	60	[15,3,3,2,5,14]
	50	[16,4,5,4,7,15]
	40	[16,6,7,6,8,16]
	30	[16,8,8,7,10,16]
Filtro 6		
	80	[12,0,0,5,5,6]
	70	[14,1,0,6,7,7]
	60	[15,3,2,8,8,9]
	50	[16,4,4,9,10,11]
	40	[16,7,6,11,12,13]
	30	[16,8,7,13,13,14]
Filtro 7		
	80	[12,0,2,0,4,7,4,11]
	70	[13,0,4,0,6,8,6,12]
	60	[15,2,5,3,7,10,7,13]
	50	[16,4,7,5,9,11,9,15]
	40	[16,6,9,6,11,13,11,16]
	30	[16,7,10,8,12,15,13,16]
Filtro 8		
	80	[11,0,0,0,5]
	70	[12,0,0,2,7]
	60	[14,1,0,3,8]
	50	[16,4,4,7,12]
	40	[16,6,6,8,13]
	30	[16,8,7,10,15]
Filtro 9		
	40	[3,0,0]
	30	[5,0,2]
Filtro 10		
	80	[10,0,0,2,5,9]
	70	[12,0,0,4,6,11]
	60	[14,0,2,5,8,13]
	50	[16,4,6,8,11,16]
	40	[16,6,8,10,13,16]
	30	[16,7,10,12,15,16]

Tabela 5.3: Número de bits aproximados em cada grupo de somadores.

Capítulo 6

Conclusão

A computação aproximada tem-se apresentado como uma opção viável para reduzir os custos de implementação em aplicações tolerantes ao erro, permitindo que os resultados destas aplicações possam conter alguns erros. No entanto, os resultados devem-se manter qualitativamente aceitáveis de acordo com a utilização a que se destinam. Aplicações como o processamento de imagem ou áudio, que podem ser aplicações resilientes ao erro, podem ser implementadas em *hardware* com computação aproximada, o que permite reduzir o problema do aumento do consumo de energia por unidade de área nas novas tecnologias CMOS [10].

Dentro desta área de investigação diferentes técnicas têm vindo a ser desenvolvidas para um vasta variedade de blocos DSP, tal como filtros FIR ou FFT. Estas técnicas seguem diferentes abordagens, desde da redução na tensão de alimentação dos circuitos digitais, à modificação nas funções lógicas de multiplicadores, à remoção de transístores num circuito de um FA, à introdução de somadores com partes exatas para cálculo dos bits mais significativos do resultado e partes aproximadas para o cálculo dos bits menos significativos e à modificação das arquiteturas dos sistemas computacionais.

6.1 Contribuições

Nesta dissertação foram estudados e avaliados dois métodos de computação aproximada para a implementação de filtros FIR em que o bloco MCM é implementado apenas com somas, subtrações e deslocamentos. Foram propostos métodos e implementados algoritmos para o projecto de filtros FIR usando computação aproximada ao nível arquitetural e ao nível lógico.

O método proposto para a computação aproximada ao nível arquitetural reduz a área necessária à implementação do filtro FIR, através da remoção de somadores do bloco MCM. As somas parciais gerada por esses somadores vão ser obtidas pelos somadores restantes, de forma a que o erro total nas saídas do bloco MCM seja minimizado. Os resultados obtidos experimentalmente apresentaram reduções de área de até 33% e de potência de até 22,6%, para um SNR de 30 dB. Mas para SNR superiores, 60 dB, as reduções são apenas de 9% e 6% na área e potência, respetivamente. Este método revelou-se pouco eficaz para filtros de pequena dimensão, isto é, com poucos somadores no

bloco MCM, visto que nestes filtros as hipóteses para substituir um somador são reduzidas, pelo que substituições com maior erro são efetuadas.

O método proposto para a computação aproximada ao nível lógico, consiste em trocar os somadores exatos, numa implementação de um filtro FIR existente baseado em arquiteturas de somas e deslocamentos, por somadores aproximados, de forma a diminuir a área necessária à implementação do filtro e ao mesmo tempo obter um valor de SNR definido. O somador aproximado usado foi o somador de cópia [10] que aproxima k bits do resultado, usando os bits de menor peso de um dos operandos. Os resultados experimentais para este método apresentaram reduções de área e potência de até 57,6% e 47,2%, respetivamente, para um SNR de 30 dB, e reduções na área de até 56,7% e na potência de 45,6%, para um nível de SNR de 60 dB.

O método de computação ao nível lógico proposto para a implementação de filtros FIR revelou-se mais eficaz que o método proposto de computação aproximada ao nível arquitetural, pois obteve reduções na área e na potência maiores para o mesmo valor de SNR. Este método obteve melhores resultados pois considera também os somadores na linha de atraso do filtro, que ,como foi mostrado, é onde as maiores reduções de área ocorrem. Foi mostrado também que alterações nos somadores de menor profundidade no bloco MCM afetam mais a qualidade do filtro do que alterações nos somadores a profundidades maiores. Isto deve-se à existência de deslocamentos para a esquerda ao longo da árvore de somadores do bloco MCM que ampliam o erro gerado nos somadores a menores profundidades.

Concluindo, a computação aproximada é ainda uma área de investigação recente, e como tal, novos métodos, em diferentes níveis de abstração, para diferentes aplicações resilientes ao erro podem ser introduzidos, havendo espaço para reduções na área e energia maiores, dependendo da aplicação e ambiente de utilização a que se destina o sistema.

6.2 Trabalho Futuro

No método proposto para a computação aproximada a nível lógico melhorias podem ser feitas, tal como aumentar o espaço de procura das soluções, aumentando o número de grupos de somadores, ou em última instância, considerar os somadores individualmente.

Este método pode ainda ser aplicado a outras aplicações DSP com somadores, apenas alterando a métrica da qualidade do circuito aproximado obtido. Poderia-se também realizar o estudo usando diferentes somadores aproximados, alternativos ao somador de cópia, tal como o somador LOA [6] ou utilizar um misto de somadores aproximados.

Bibliografia

[1] K. Palem and A. Lingamneni. Ten years of building broken chips: The physics and engineering of inexact computing. *ACM Trans. Embed. Comput. Syst*, 12(2s), Maio 2013.

[2] L. N. B. Chakrapani, K. K. Muntimadugu, A. Lingamneni, J. George, and K. V. Palem. Highly energy and performance efficient embedded computing through approximately correct arithmetic: A mathematical foundation and preliminary experimental validation. In *Proceedings of IEEE/ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 187–196, 2008.

[3] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design Of Integrated Circuits and Systems*, 32(1):124–137, Janeiro 2013.

[4] P. Kulkami, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *Proceedings of* 24^{th} *Annual Conference on VLSI Design*, pages 356–352, 2011.

[5] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi. Approximate XOR/XNOR-based adders for inexact computing. In *Proceedings of* 13^{th} *IEEE International Conference on Nanotechnology*, 2013.

[6] H. R. Mahdiani, A. Ahmadi, and C. Lucas. Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems–I: Regular Papers*, 57(4):850–862, Abril 2010.

[7] N. Zhu, W. L. Goh, K. S. Yeo, and Z. H. Kong. Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(8):1225–1229, Agosto 2010.

[8] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo. An enhanced low-power high-speed adder for error-tolerant application. In *Proceedings of the* 12^{th} *International Symposium on Integrated Circuits*, pages 69–72, 2009.

[9] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo. Enhanced low-power high-speed adder for error-tolerant application. In *Proceedings of IEEE International SoChip Design Conference*, pages 323–327, 2010.

[10] L. B. Soares and S. Bampi. Approximate adder synthesis for area- and energy-efficient FIR filters in CMOS VLSI. In *Proceedings of* 13^{th} *Internacional New Circuits and Systems Conference (NEWCAS)*, 2015.

[11] A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet. Energy parsimonious circuit design through probabilistic pruning. In *Proceedings of Design, Automation and Test in Europe Conference*, pages 764–769, 2011.

[12] L. Aksoy, P. Flores, and J. Monteiro. Exact and approximate algotithms for the filter design optimization problem. *IEEE Transactions on Signal Processings*, 63(1):142–154, Janeiro 2015.

[13] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *Proceedings of* 18^{th} *IEEE European Test Symposium (ETS)*, 2013.

[14] W. J. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R. C. Harting, V. Parikh, J. Park, and D. Sheffield. Efiicient embedded computing. *Computer*, 41(7):27–32, Julho 2008.

[15] L. Aksoy, P. Flores, and J. Monteiro. A tutorial on multiplierless design of fir filters: Algorithms and architectures. *Circuits, Syst., Signal Process.*, 33(6):1689–1719, Janeiro 2014.

[16] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete- Time Signal Processing*, chapter 7.2, 7.4.3. Prentice-Hall, Inc., 2nd edition, 1999.

[17] L. Aksoy, E. Gunes, and P. Flores. Search algorithms for the multiple constant multiplications problem: Exact and approximate. *Elsevier J. Microprocessors and Microsystems*, 34(5):151–162, 2010.

[18] P. Cappello and K. Steiglitz. Some complexity issues in digital signal processing. *IEEE Tran. on Acoustics, Speech, and Signal Processing*, 32(5):1037–1041, 1984.

[19] L. Aksoy, E. Costa, P. Flores, and J. Monteiro. Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 27(6):1013–1026, Junho 2008.

[20] L. Aksoy, E. Costa, P. Flores, and J. Monteiro. Optimization algorithms for the multiplierless realization of linear transforms. *ACM Transactions on Design Automation of Electronic Systems*, 17(1), 2012.

[21] N. Boullis and A. Tisserand. Some optimizations of hardware multiplication by constant matrices. *IEEE Transactions on Computers*, 54(10):1271–1282, 2005.

[22] R. Hartley. Subexpression sharing in filters using canonic signed digit multipliers. *IEEE Transactions on Circuits and Systems II*, 43(10):677–688, 1996.

[23] Y. Ho, C. Lei, H. Kwan, and N. Wong. Global optimization of common subex- pressions for multiplierless synthesis of multiple constant multiplications. In *Proceedings of Asia and South Pacific Design Automation Conference*, volume 43, pages 119–124, 2008.

[24] A. Dempster and M. Macleod. Constant integer multiplication using minimum adders. In *IEEE Proceedings - Circuits, Devices and Systems*, volume 141, pages 407–413, 1994.

[25] A. Dempster and M. Macleod. Use of minimum-adder multiplier blocks in fir digital filters. *IEEE Transactions on Circuits and Systems II*, 42(9):569–577, 1995.

[26] O. Gustafsson. A difference based adder graph heuristic for multiple constant multiplication problems. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 1097–1100, 2007.

[27] O. Gustafsson, A. Dempster, and L. Wanhammar. A difference based adder graph heuristic for multiple constant multiplication problems. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 73–76, 2002.

[28] Y. Voronenko and M. Püschel. Multiplierless multiple constant multiplication. *ACM Transactions on Algorithms*, 3(2), 2007.

Apêndice A

Método de Parks-McClellan

O método de Parks-McClellan [16] tem como objetivo encontrar o filtro ótimo de ordem N, que satisfaça as especificações de ω_p e ω_s, minimizando δ_p e δ_s. Este método é explicado em seguida para um filtro passa-baixo causal de fase linear e de ordem N ímpar. Para que o filtro tenha fase linear a condição

$$h_e[n] = h_e[-n], \tag{A.1}$$

tem de ser respeitada. A resposta em frequência de $h_e[n]$ é dada por

$$\begin{aligned} A_e(e^{j\omega}) &= \sum_{n=-L}^{L} h_e[n]e^{-j\omega n} \\ &= h_e[0] + \sum_{n=1}^{L} 2h_e[n]\cos(\omega n), \end{aligned} \tag{A.2}$$

com $L = (N-1)/2$. Para o filtro ser causal um atraso de L tem de ser aplicado, sendo que a resposta em frequência do filtro é dada por

$$H(e^{j\omega}) = A_e(e^{j\omega})e^{-j\omega(N-1)/2}, \tag{A.3}$$

e a sua resposta ao impulso por

$$h[n] = h_e[n-(N-1)/2]. \tag{A.4}$$

O algoritmo de Parks-McClellan utiliza uma aproximação polinomial para os termos $\cos(\omega n)$ na equação (A.2) que podem ser escritos na forma

$$\cos(\omega n) = T_n(\cos(\omega)), \tag{A.5}$$

onde $T_n(x)$ é o polinómio de Chebychev de ordem n dado por

$$T_n(x) = \cos(n\cos^{-1}(x)). \tag{A.6}$$

A equação (A.2) pode ser então rescrita como um polinómio em $\cos(\omega)$ de ordem L,

$$A_e(e^{j\omega}) = \sum_{k=0}^{L} a_k(\cos(\omega))^k, \tag{A.7}$$

onde os a_k são constantes relacionadas com os valores da resposta ao impulso $h_e[n]$.

A função de erro da aproximação é definida da seguinte forma:

$$E(\omega) = W(\omega)[H_d(e^{j\omega} - A_e(e^{j\omega}], \tag{A.8}$$

onde $W(\omega)$ é uma função de peso que incorpora os parâmetros de erro da aproximação e $H_d(e^{j\omega})$ é a resposta em frequência do filtro ideal. Para o caso do exemplo do filtro passa-baixo de fase linear,

$$W(\omega) = \begin{cases} \frac{\delta_s}{\delta_p} = \frac{1}{K}, & 0 \leq \omega \leq \omega_p \\ 1, & \omega_s \leq \omega \leq \pi. \end{cases} \tag{A.9}$$

A melhor aproximação (valores da resposta ao impulso) é encontrada aplicando o critério de Chebychev (ou minimax), ou seja, minimizando o máximo da função de erro (A.8), nos intervalos de frequência de interesse (banda de passagem e banda de atenuação).

O algoritmo faz uso do teorema da alternância que diz que se F_p é um sub-conjunto fechado que consiste na união de sub-conjuntos fechados disjuntos, $D_p(x)$ uma função desejada contínua em F_p, $W_p(x)$ uma função positiva e contínua em F_p, $P(x)$ um polinómio de ordem r e $E_p(x) = W(x)[D_p(x) - P(x)]$ então uma condição necessária e suficiente para que $P(x)$ seja o único polinómio de ordem r que minimiza o erro máximo, definido por $\|E\| = \max_{x \in F_p} |E_p(x)|$, é que $E_p(x)$ tenha pelo menos $(r+2)$ alternâncias, ou seja, existem pelo menos $(r+2)$ valores de $x_i \in F_p$, em que $x_1 < x_2 < \cdots < x_{r+2}$, tal que $E_p(x_i) = -E_p(x_{i+1}) = \pm\|E\|$ para $i = 1, 2, ..., (r+1)$.

Pelo teorema da alternância é possível concluir que o filtro ótimo $A_e(e^{j\omega})$ satisfaz o conjunto de equações

$$W(\omega_i)[H_d(e^{j\omega_i} - A_e(e^{j\omega_i}] = (-1)^{i+1}\delta, \qquad i = 1, 2, 3..., (L+2), \tag{A.10}$$

onde δ é o erro ótimo.

O algoritmo de Parks-McClellan é iterativo e o seu fluxograma está representado na Figura A.1. Quando entre iterações o valor de δ não é alterado em mais que um valor predeterminado o algoritmo pára, encontrando assim o polinómio de ordem L que melhor aproxima a resposta em frequência do filtro desejado. A resposta ao impulso do filtro, e consequentemente os coeficientes do filtro, é então obtida a partir de amostras do polinómio usando a transformada de Fourier discreta.

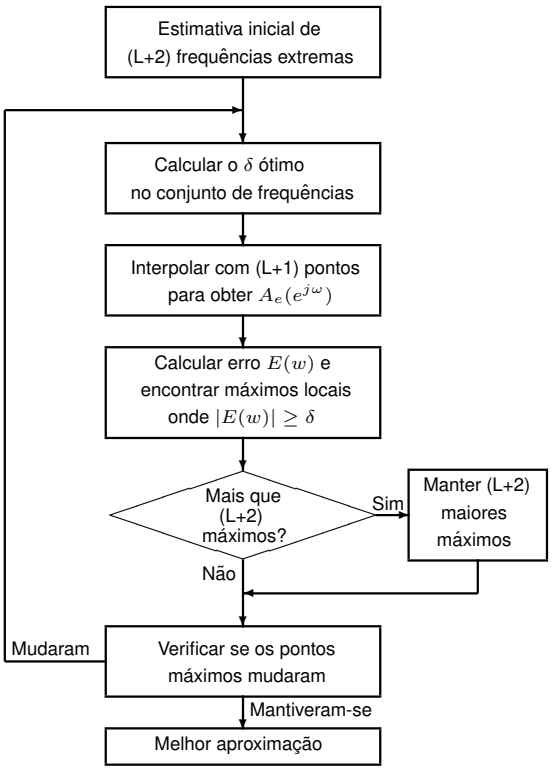


Figura A.1: Fluxograma do algoritmo de Parks-McClellan.