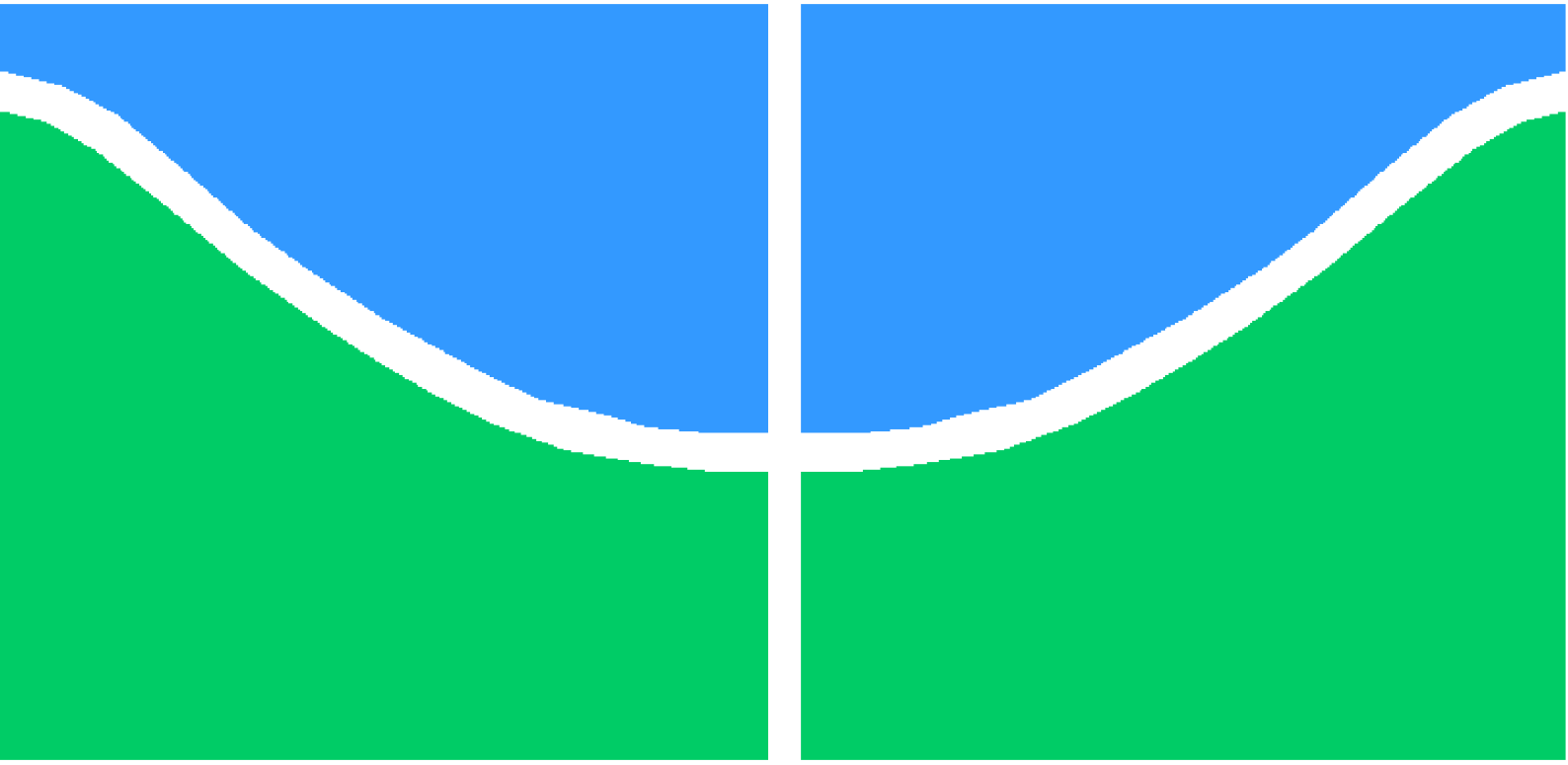

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia Eletrônica

Projeto de Implementação VLSI de um Filtro Equalizador Least Mean Squares Adaptativo ao Canal de Comunicação em Tecnologia CMOS 0.18 μm

Autor: João Pedro da Silva Cerqueira

Orientador: Prof. Dr. Sandro Augusto Pavlik Haddad

Brasília, DF

2014

João Pedro da Silva Cerqueira

Projeto de Implementação VLSI de um Filtro Equalizador Least Mean Squares Adaptativo ao Canal de Comunicação em Tecnologia CMOS 0.18 μm

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do título de Bacharel em Engenharia Eletrônica.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Sandro Augusto Pavlik Haddad

Brasília, DF

2014

João Pedro da Silva Cerqueira

Projeto de Implementação VLSI de um Filtro Equalizador *Least Mean Squares* Adaptativo ao Canal de Comunicação em Tecnologia CMOS 0.18 μm/ João Pedro da Silva Cerqueira. – Brasília, DF, 2014-

155 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Sandro Augusto Pavlik Haddad

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2014.

1. VLSI. 2. Equalização Adaptativa. I. Prof. Dr. Sandro Augusto Pavlik Haddad. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. **Projeto de Implementação VLSI de um Filtro Equalizador *Least Mean Squares* Adaptativo ao Canal de Comunicação em Tecnologia CMOS 0.18 μm**

CDU 02:141:005.6

João Pedro da Silva Cerqueira

Projeto de Implementação VLSI de um Filtro Equalizador Least Mean Squares Adaptativo ao Canal de Comunicação em Tecnologia CMOS 0.18 μm

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do título de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Brasília, DF, 29 de abril de 2014:

Prof. Dr. Sandro Augusto Pavlik Haddad
Orientador

Prof. Dr. Daniel Mauricio Muñoz Arboleda
Membro Convidado

Prof. Dr. Gilmar Silva Beserra
Membro Convidado

Brasília, DF
2014

Este trabalho é dedicado àqueles que amo incondicionalmente:
Marcelo Cerqueira Moacyr e Maria Ozilene da Silva Cerqueira,
meus pais, minha vida.

Agradecimentos

Eu gostaria de expressar minha gratidão à todos que me ajudaram na superação das dificuldades para completar mais uma etapa da minha graduação.

Agradeço ao Prof. Dr. **Sandro Augusto Pavlik Haddad**, meu orientador, que me ajudou e me guiou profissionalmente e pessoalmente durante essa jornada. Ele me encorajou à dar continuidade à um projeto trazido do período do intercâmbio, na área de projeto de circuitos integrados, bem como à completar o ciclo de disciplinas na área de Microeletrônica do curso de Engenharia Eletrônica na Universidade de Brasília, Faculdade UnB Gama. Além do professor orientador, agradeço à banca de Trabalho de Conclusão de Curso, composta pelos professores doutores **Daniel Mauricio Muñoz Arboleda** e **Gilmar Silva Beserra**, pelas críticas construtivas e também pela ajuda e atenção dispensada, nos momentos que precisei.

Agradeço também aos meus chefes e mentores durante os períodos de estágio na *Samsung Electronics* e *Hana Micron*, Dr. **Minsuk Song** e Dr. **Hyouk Lee**, respectivamente. Eles me ajudaram na iniciação profissional e na experiência de trabalhar na área de circuitos integrados.

Expresso também minha gratidão ao Sr. **Daniel Fink**, assessor de Ciência e Tecnologia da Embaixada do Brasil na Coreia do Sul, bem como ao embaixador do Brasil na Coreia do Sul, Sr. **Edmundo Fujita**, pelas diversas oportunidades durante o período de intercâmbio.

Agradeço ao Prof. Dr. **Cristiano Jacques Miosso Rodrigues Mendes** por sua dedicação, comprometimento e prestatividade em todos os momentos. O professor Cristiano se tornou e ainda é, de longe, a minha maior inspiração profissional.

Aos meus amigos de Engenharia, **Vinícius Oliveira**, **Pablo Henrique**, **Mileny Ximenes**, **Guilherme Peron**, **Lucas Coelho**, **Nicholas Gustavo**, **José Alberto**, **Heider Marconi**, **Eduardo Martins** e **Rafael Ferreira** pela ajuda em diferentes situações.

Por último, agradeço à meus pais, Sr. **Marcelo Cerqueira Moacyr** e Sra. **Maria Ozilene da Silva Cerqueira**, por todo o suporte e amor incondicional. Sem eles eu não teria chegado até aqui.

"We make a living by what we get, but we make a life by what we give."

Resumo

Em sistemas de comunicação não-ideais, os meios físicos entregam, aos receptores, versões deterioradas e transformadas dos sinais de entrada por eles transmitidos. A deterioração dessas formas de onda é normalmente atribuída ao ruído (intrínseco aos sistemas), enquanto que as transformações podem ser diversas. Por exemplo, uma dessas transformações, executada pelo canal, a distorção temporal causada pela propagação multicaminho do sinal transmitido, em sistemas de comunicação digital, incorre em um problema conhecido por interferência entre símbolos. Esta é uma forma de deformação de um sinal, na qual, o símbolo presente/atual interfere em símbolos subsequentes, tornando o processo de comunicação menos confiável. Sendo assim, o presente trabalho propõe o projeto e implementação de um filtro digital do tipo resposta finita ao impulso, para a equalização adaptativa de um canal de comunicação não-ideal, sujeito à ruído e distorção temporal causada por propagação multicaminho. Com o desenvolvimento da proposta em questão, espera-se que esta situação, não-desejável, possa ser revertida, mitigando-se o problema da interferência entre símbolos. Este trabalho, dividido em duas partes, visa uma implementação através do fluxo de projeto de circuitos integrados digitais. Dessa forma, o Trabalho de Conclusão de Curso 1 objetivou a apresentação da proposta, revisão bibliográfica e simulação/projeto parcial do sistema. O projeto e implementação definitivos do filtro equalizador e *tape-out* do *chip* **UNB0414** foram feitos durante o Trabalho de Conclusão de Curso 2. Bons resultados foram obtidos durante os testes com o modelo comportamental e funcional na atenuação das dispersões causadas devido à um canal não-ideal de propagação multicaminho. Trabalhos futuros visarão o tratamento para o problema de *feedback acústico* em aparelhos auditivos com especificações mais rígidas para área e potência dispensada pelo filtro.

Palavras-chaves: Filtros Equalizadores Adaptativos. *Least Mean Square*. Projeto de Circuitos Eletrônicos Integrados.

Abstract

In non-ideal communication systems, the physical medium delivers, to the receivers, deteriorated and transformed versions of the input signals they transmitted. Deteriorating these waveforms is usually due to the noise (intrinsic to the system), while transforming might be owing to several causes. For example, one of these transformations, performed by the channel, the time dispersion caused by multipath propagation of the transmitted signal in digital communication systems incurs a problem named intersymbol interference. This is a form of distortion of a signal in which one current symbol interferes with subsequent symbols. Thus, this work proposes the design and implementation of a digital adaptive equalizer for a non-ideal communication channel, subjected to noise and time dispersion from multipath propagation. By developing the proposal project, it is expected that this time dispersion, which is non-desirable, can be treated, attenuating the intersymbol interference issue. This work, divided into two parts, aims an implementation following the digital integrated circuit design flow. Therefore, the goals for the Final Paper 1 were the proposal, literature review, and system's partial simulation/design. The definitive design and implementation parts of the equalizer and the tape-out regarding the chip **UNB0414** were covered in the Final Paper 2. Very good results were obtained from tests and analysis with both behavioral and functional models for treating/mitigating distortions due to a non-ideal multipath propagation channel. Future works will focus on acoustic feedback in hearing aid devices, with more aggressive area and power specifications.

Key-words: Adaptive Equalizers. *Least Mean Squares*. Integrated Circuit Design.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AD – *Analog-to-digital*

ADC – *Analog-to-Digital Converter*

AMS – *Analog and Mixed-signal*

ASIC – *Application-specific Integrated Circuit*

AWGN – *Additive White Gaussian Noise*

BER – *Bit Error Rate*

BPSK – *Binary Phase-shift Keying*

CAD – *Computer-aided Design*

CAT – *Computer-aided Techniques*

CI – Circuito Integrado

CIC – *Cascaded Integrator-comb*

CMOS – *Complementary Metal–oxide–semiconductor*

CTS – *Clock Tree Synthesis*

IES – *Intersymbol Interference*

SNR – *Signal-to-noise Ratio*

FPGA – *Field-programmable Gate Array*

DF-I – *Direct Form I*

DK – *Design Kit*

DRC – *Design Rule Check*

DSP – *Digital Signal Processor*

EDA – *Electronic Design Automation*

EDI – *Encounter Digital Implementation*

FIR – *Finite Impulse Response*

FS – *Fully Serial*

GDS – *Graphic Database System*

HDL – *Hardware Description Language*

I/O – *Input/Output*

IEEE – *Institute of Electrical and Electronics Engineers*

IIR – *Infinite Impulse Response*

LMS – *Least Mean Squares*

LPF – *Low-pass Filter*

LSB – *Least Significant Bit*

LVS – *Layout vs. Schematic*

MSB – *Most Significant Bit*

MSE – *Mean Square Error*

P&R – *Place and Route*

PA – *Power Amplifier*

PCM – *Pulse-code Modulation*

PD – Pré-distorção Digital

PDK – *Process Design Kit*

PDM – *Pulse-density Modulation*

RF – Radio Frequência

RLS – *Recursive Least Squares*

RTL – *Register Transfer Level*

TB – *Testbench*

TCC – Trabalho de Conclusão de Curso

TDF – *Transposed Direct Form*

TI – *Texas Instruments*

TSMC – *Taiwan Semiconductor Manufacturing Company*

VLSI – *Very-large-scale Integration*

Sumário

Parte I

Fundamentos

1 Introdução

1.1 Aspectos Gerais

A integridade do sinal é um ponto crítico para que um sistema de comunicação seja de confiança. Segundo [Qureshi 1985], em sistemas não-ideais, os canais de comunicação transmitem uma versão deteriorada e transformada das formas de onda de um sinal de entrada. A deterioração — normalmente estatística -– dessas formas de onda, pode ser aditiva e/ou multiplicativa, devido ao ruído, que é intrínseco do sistema. Já as transformações realizadas pelo meio físico de transmissão são translações na frequência, distorção não-linear ou harmônica, e dispersão temporal.

Em linhas de telefone, por exemplo, a dispersão temporal, é resultado da resposta em frequência do canal de comunicação que se desvia do ideal, ou seja, amplitude não-constante e fase não-linear. Neste caso, processo de equalização, via de regra, através de um filtro equalizador, tem a capacidade de compensar essas características não-ideias [Qureshi 1985].

Em sistemas de comunicação digital de largura de banda eficiente, o efeito de cada símbolo transmitido através de um canal (com a característica não-ideal de dispersão temporal), se estende além do intervalo de tempo normalmente utilizado para representar aquele símbolo. A distorção causada pela interposição de diferentes símbolos no receptor é chamada de Interferência entre Símbolos (IES). Esse tipo de distorção é o maior dos obstáculos para sistemas de transmissão de dados de alta velocidade sobre canais de baixo ruído e largura de banda limitada. Nesse sentido, o termo "equalizador"se aplica à qualquer dispositivo de processamento de sinais projetado para lidar com a IES [Qureshi 1985].

Dessa forma, é reconhecido que, para que as comunicações de dados modernas sejam eficientes e confiáveis, é necessário que se reduza a interferência entre símbolos introduzida pelo canal de comunicação. Além disso, em situações práticas, as características do meio físico de transmissão de dados não são conhecidas previamente e variam em função do tempo. Portanto, a transmissão eficientemente de dados se faz viável pelo uso, além de outras técnicas, de equalização adaptativa: (1) para compensação da distorção temporal do sinal, introduzida pelo canal de comunicação, e (2) para adaptação da do filtro equalizador às características, variáveis em função do tempo, do canal [Qureshi 1985].

Um equalizador deve funcionar de tal forma que a taxa de erro de bit (*Bit Error Rate – BER*) deva ser baixa e a relação sinal-ruído (*Signal-to-noise Ratio – SNR*) deva ser alta. Assim, o filtro equalizador fornece a função de transferência inversa do canal de

comunicação para o sinal recebido. A Figura 1 ilustra o conceito.

Figura 1 – Conceito de equalizador.

O filtro equalizador estático é de fácil implementação, entretanto, como mencionado anteriormente, as variações do canal facilmente afetam sua performance. Dessa forma, os esquemas mais utilizados envolvem equalização adaptativa, onde o próprio filtro se adapta às variações no tempo do meio físico [Malik e Sappal 2011].

O projeto e implementação de um filtro equalizador adaptativo pode ser aplicado à *Field-programmable Gate Arrays* (FPGAs), *Application-specific Integrated Circuits* (ASICs) ou *Digital Signal Processor* (DSP). Os requisitos e especificações do produto ditam a necessidade de uso de uma plataforma em específico. Neste trabalho, particularmente, busca-se a implementação voltada à um ASIC.

1.2 Objetivos

1.2.1 Objetivo Geral

O trabalho de conclusão de curso é dividido em duas partes. Cada uma dessas, desenvolvida durante um semestre letivo. Dessa forma, o objetivo geral do Trabalho de Conclusão de Curso 1 foi o estudo das técnicas, metodologias e tecnologias para o projeto e implementação de um filtro digital equalizador adaptativo para um canal de comunicação não-ideal (sujeito à ruído e interferência entre símbolos causada pela propagação multicaminho).

Com base no estudo que foi desenvolvido no Trabalho de Conclusão de Curso 1, o objetivo geral do Trabalho de Conclusão de Curso 2 é a implementação da aplicação em questão. Sendo assim, o objetivo final da disciplina Trabalho de Conclusão de Curso é a realização do filtro digital equalizador adaptativo através do fluxo de projeto de circuitos integrados digitais.

Este documento, por sua vez, apresenta uma compilação de informações importantes para entender e implementar o sistema proposto. Além disso, reporta-se também os resultados obtidos durante o projeto, bem como sugestões de trabalhos que podem ser desenvolvidos no futuro.

1.2.2 Objetivos Específicos

Podem ser citados como objetivos específicos do Trabalho de Conclusão de Curso:

1. Estudar e analisar diferentes técnicas de equalização adaptativa;
2. Aprender e praticar o fluxo de projeto de circuitos integrados digitais;
3. Desenvolver habilidades de programação em linguagens de descrição de *hardware*, como Verilog e sua derivativa Verilog-AMS;
4. Desenvolver habilidades de programação em MATLAB;
5. Desenvolver habilidades de projeto de sistemas em ponto-fixo;
6. Estudar e aprender as ferramentas CAD/EDA disponíveis nos laboratórios da Universidade de Brasília;
7. Ajudar no desenvolvimento de *scripts* de síntese lógica utilizando-se do *software Cadence* RTL *Compiler*, para o laboratório SS da Faculdade UnB Gama;
8. Ajudar no desenvolvimento de *scripts* a fim de automatizar a síntese física, utilizando-se do *software Cadence Encounter Digital Implementation* (versão 13.17), para o laboratório SS da Faculdade UnB Gama;
9. Estudar diferentes técnicas de conversão analógico-digital;
10. Ajudar os alunos da disciplina Projeto de Circuitos Eletrônicos Integrados 2 no projeto de um modulador $\Sigma\Delta$, de primeira ordem;
11. Desenvolver um filtro decimador digital para o modulador $\Sigma\Delta$, com o objetivo de formar um conversor analógico-digital $\Sigma\Delta$ completo;
12. Ajudar no *tape-out* do *chip* **UNB0414**, levado para fabricação no dia 15 de abril de 2014, sendo composto por diversos projetos, inclusive os filtros decimador e equalizador LMS adaptativo, apresentados neste documento.

1.3 Motivações

Embora o Trabalho de Conclusão de Curso se limite ao tratamento de um problema específico (o da equalização de canais de comunicação), há diversas motivações (*i.e.*, vários outros problemas) para o estudo e desenvolvimento de soluções em áreas do conhecimento relacionadas.

Esta seção descreve problemas que podem ser tratados através das técnicas apresentadas durante esse Trabalho de Conclusão de Curso. Essas técnicas se referem principalmente à (1) utilização de filtros digitais equalizadores como um recurso para o problema da propagação multicaminho, e, também, (2) à utilização de estratégias de adaptação para linearização.

Com relação à (1), o capítulo apresenta, como motivação, a equalização adaptativa aplicada à aparelhos auditivo para solução do problema de acoplamento acústico entre o microfone e o alto-falante do aparelho. Esse problema se assemelha com o de equalização de canais de comunicação, porém, deve-se levar em conta outros fatores importantes no projeto. Um deles, é o consumo de energia, que no caso, deve ser mínimo. Técnicas de projeto na região de *subthreshold* são exploradas afim de solucionar o problema.

Por outro lado, com relação à (2), o capítulo apresenta a motivação do uso de estratégias de adaptação para linearização de amplificadores de potência (*Power Amplifiers* – PAs). Amplificadores de potência, principalmente os das classes E, F e EF, atingem alta eficiência em regiões de não-linearidade. No entanto, para diversas aplicações, eficiência em termos de consumo de energia é uma especificação importante a ser cumprida. Surge, portanto, a necessidade de linearizar a resposta desses amplificadores em regiões de alta eficiência. Dentre as diversas técnicas para linearização destes dispositivos, este trabalho enfatiza *pré-distorção digital*, que faz uso de estratégias de adaptação.

1.3.1 Equalização Adaptativa em Aparelhos Auditivos

Os aparelhos auditivos são pequenos dispositivos eletroacústicos projetados com a finalidade de ajudar pessoas que sofrem de alguma deficiência ou perda auditiva [Spriet et al. 2006]. Segundo o mesmo autor citado, estes artefatos têm a capacidade de amplificar o som, promovendo uma melhor compreensão do discurso recebido e corrigindo deficiências auditivas de seus usuários.

O problema de realimentação acústica, em inglês *acoustic feedback*, em aparelhos auditivos, se refere ao acoplamento acústico entre o alto-falante e o microfone de um mesmo dispositivo [Spriet et al. 2006]. Este defeito ocorre quando o sinal amplificado, proveniente do alto-falante, é indevidamente capturado pelo microfone. Devido à este acoplamento, ao passo que aumenta-se o ganho, observa-se uma distorção grave do sinal desejável. Como resultado, uma sonoridade desconfortável, irritante e inconveniente é produzida, deteriorando a qualidade do som audível e limitando o ganho efetivo do dispositivo para uma operação estável [Spriet et al. 2006]. O problema do *feedback* acústico é ilustrado na Figura 2 para um aparelho auditivo com um único microfone.

Diferentes métodos para redução ou solução do problema de realimentação acústica em aparelhos auditivos têm sido investigados. Nota-se que entre as diversas abordagens,

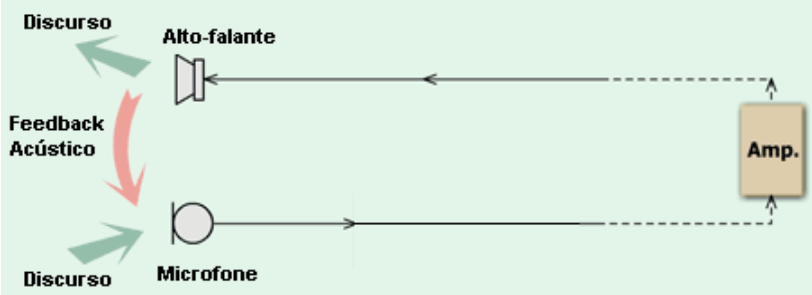

Figura 2 – Ilustração do problema de *feedback* acústico. O microfone indevidamente captura o sinal provido pelo alto-falante do próprio dispositivo, criando uma realimentação indesejável no sistema.

utilizam-se, fundamentalmente, técnicas de equalização adaptativa [Kim e Roy 2007]. De fato, o problema se à interferência intersimbólica, causada pela propagação multicaminho de uma sequência de símbolos. Portanto, uma possível aproximação ao problema de realimentação acústica é dada pela modelagem do sistema aparelho-canal auditivo como um sistema de comunicação.

Por outro lado, os avanços na tecnologia permitiram aparelhos auditivos cada vez mais compactos e sofisticados. Os dispositivos mais modernos são convenientemente pequenos para serem introduzidos no canal auditivo e utilizam circuitos integrados (CIs) que podem implementar algoritmos complexos de processamento de sinais [Kim e Roy 2007]. Devido ao tamanho miniaturizado de suas baterias e de aplicações dispendiosas, soluções para estes artefatos tornam-se cada vez mais desafiantes, uma vez que, existe a necessidade de um compromisso entre performance necessária, eficiência em termos de área e baixa potência [Kim e Roy 2007].

1.3.2 Técnicas de Adaptação na Linearização de Amplificadores de Potência

Consumo de energia eficientemente tem se tornado cada vez mais importante na área de comunicações móveis. Em amplificadores de potência (*Power Amplifiers* – PAs) de RF, o *trade-off* entre linearidade e eficiência torna-se uma questão crucial. Tipicamente, a linearidade é atingida ou pela redução da eficiência energética ou pela utilização de técnicas de linearização. Considerando o cenário atual, um alto consumo de energia, para algumas aplicações (especialmente comunicações móveis) não é desejável. Portanto, surge a necessidade de desenvolvimento e aprimoramento de técnicas de linearização para que seja possível obter alta eficiência aliada à linearidade em PAs.

Existem vários métodos de linearização de amplificadores de potência. Um dos mais utilizados, por exemplo, é conhecido como técnica de *feed-forward*. Quando empregada, geralmente, resulta em boa linearidade, porém com baixa eficiência [Razavi 1998].

O método ênfase dessa subseção é conhecido como *pré-distorção digital adaptativa*

em banda base. Esta técnica faz uso dos mesmos algoritmos de adaptação utilizados no filtro equalizador adaptativo.

1.3.3 Pré-Distorção Digital Adaptativa em Banda Base

Pré-distorção digital (PD) é utilizada para linearizar a resposta não-linear de um PA sobre um faixa de potência desejável. Este método utiliza técnicas de processamento de sinais para introduzir uma pré-distorção no sinal em banda base, antes de ser modulado, convertido para alta frequência e amplificado pelo PA. A Figura 3 mostra um diagrama de blocos simplificado [Mekechuk et al. 2004].

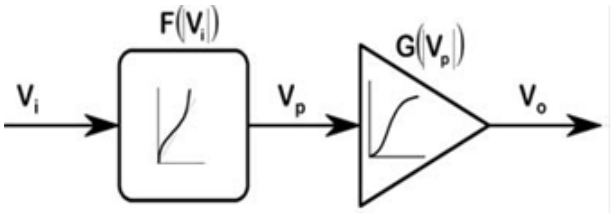

Figura 3 – Diagrama de blocos simplificado mostrando o bloco de pré-distorção, que compensa a não-linearidade do PA [Mekechuk et al. 2004].

Na figura, $G(|V_p|)$ é a função de transferência do PA com o sinal de entrada V_p e $F(|V_i|)$ é a função de transferência do *predistorter* com entrada V_i. Portanto, deseja-se que a configuração em cascata do *predistorter* e do PA resulte em uma resposta linear, tal que $F(|V_i|) \times G(|V_p|) = k$, onde k é uma constante. A Figura 4 ilustra o relacionamento entre a potência de entrada e a potência de saída do PA [Mekechuk et al. 2004].

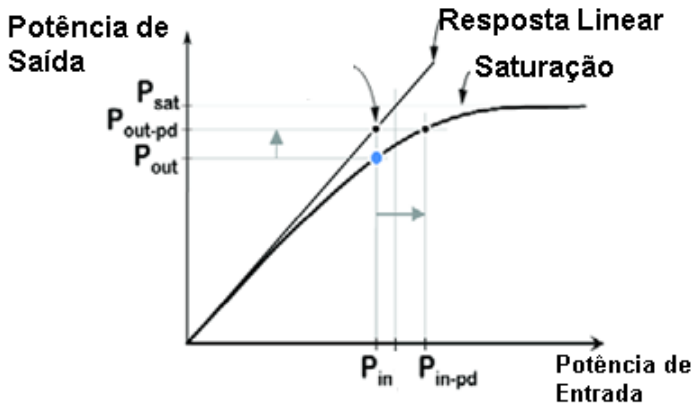

Figura 4 – Relacionamento entre a potência de entrada e a potência de saída do PA [Mekechuk et al. 2004].

Ao incluir-se o bloco *predistorter*, como mostrado na primeira figura, introduz-se expansão – a amplitude do sinal de entrada é aumentada para que seja atingida a potência de saída desejada (leia-se linear).

O diagrama de blocos de um sistema de pré-distorção digital adaptativa é mostrado na Figura 5. O bloco destacado em vermelho corresponde à aplicação de algoritmos de adaptação, que também podem ser usados no projeto do filtro equalizador adaptativo, desenvolvido durante este Trabalho de Conclusão de Curso (neste trabalho, em particular, decidiu-se pela aplicação do algoritmo LMS). Observa-se, portanto, mais uma motivação para o estudo de técnicas de adaptação.

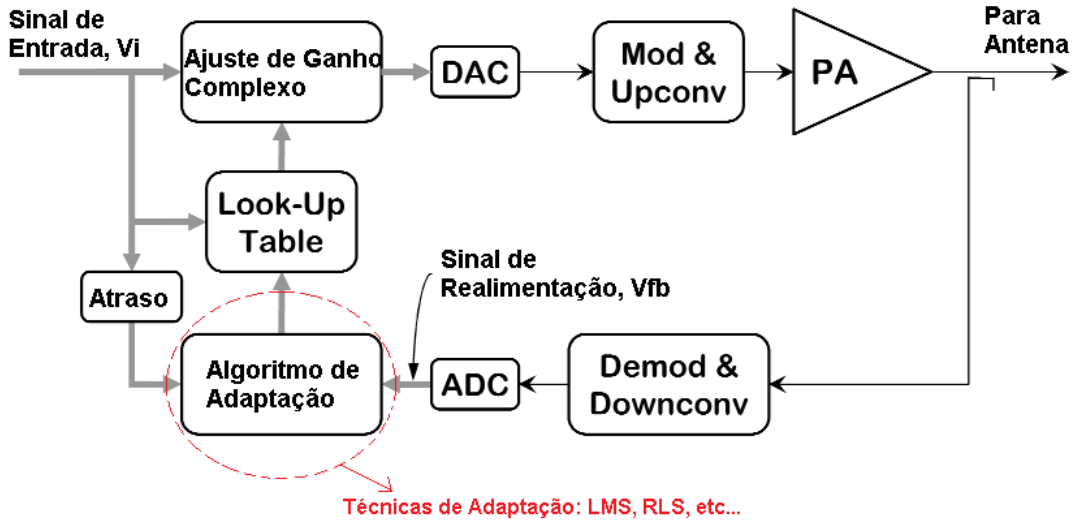

Figura 5 – Sistema de pré-distorção digital adaptativa. O bloco circundado corresponde à aplicação de algoritmos de adaptação, por exemplo, o LMS, apresentado no capítulo 3.

1.4 A Organização deste Trabalho

Para melhor entendimento e didática, esta monografia é dividida em 3 partes: Fundamentos, Projeto e Implementação e Resultados. A parte I é referente aos fundamentos necessários para entendimento do problema e da solução implementada. Dessa forma, o capítulo 1 faz uma introdução ao problema, propõe uma solução e descreve os objetivos e motivações deste trabalho. Os capítulos subsequentes, do primeiro até o capítulo 6, didaticamente, resumem a teoria por trás deste projeto. A parte II visa o desenvolvimento da solução. Sendo assim, todos os capítulos que compõe a segunda parte são referentes à implementação do filtro equalizador LMS adaptativo. Por último, a parte III – Resultados e Conclusão – descreve os resultados obtidos.

Capítulo 1 Faz uma introdução ao problema e a solução proposta. Endereça o objetivo geral e objetivos específicos do trabalho. Além disso, traz uma subseção que apresenta outras motivações (outros projetos interessantes) que poderiam ser abordados em trabalhos futuros utilizando-se dos mesmos conceitos apresentados durante este projeto.

Capítulo 2 Introduz o processo de comunicação, ressaltando o problema de propagação multicaminho e seu efeito: a interferência entre símbolos.

Capítulo 3 Apresenta filtros adaptativos, bem como introduz o algoritmo de adaptação *Least Mean Squares*.

Capítulo 4 Apresenta fundamentos sobre circuitos integrados digitais. São apresentados, neste capítulo, os processos de projeto bem como as metodologias utilizadas.

Capítulo 5 Apresenta filtros digitais do tipo FIR e IIR para processamento digital de sinais e mostra diversas topologias de filtros digitais FIR.

Capítulo 6 Introduz alguns conceitos importantes referentes à representação numérica em sistemas digitais.

Capítulo 7 Descreve considerações importantes para o projeto e implementação do filtro equalizador LMS adaptativo ao canal de comunicação.

Capítulo 8 Apresenta os modelos do sistema e do canal.

Capítulo 9 Descreve a implementação (propriamente dita) do projeto.

Capítulo 10 Apresenta os resultados obtidos para o filtro equalizador LMS adaptativo.

Capítulo 11 Mostra os resultados do *chip* **UNB0414**, que comporta vários projetos, entre eles, o filtro equalizador e o filtro decimador, apresentado no Apêndice B.

Apêndice A Explica a implementação de um conversor analógico-digital de 16 bits em Verilog-AMS, como estrutura de testes para o filtro equalizador LMS.

Apêndice B Ilustra a implementação da parte digital (filtro decimador de terceira ordem) para moduladores $\Sigma\Delta$, também levado para fabricação compondo o *chip* **UNB0414**.

2 O Processo de Comunicação

2.1 Sistemas de Comunicação

Nos dias de hoje, o processo de comunicação está presente no dia-a-dia dos seres humanos em muitos e diferentes meios. Telefones, rádios, televisões, terminais de computadores com acesso à internet, estão entre milhares de exemplos. No senso mais fundamental, comunicação envolve, implicitamente, a transmissão de informação de um ponto para outro, através de uma sucessão de processos, como descrito em [Haykin 2001]:

1. A geração do sinal mensagem. Por exemplo: voz, música, figuras ou dados de computadores;

2. A descrição do sinal mensagem através de um conjunto de símbolos;

3. A codificação desses símbolos em uma forma adequada para transmissão através de um meio físico de interesse;

4. A transmissão dos símbolos codificados para o destino desejado;

5. A decodificação e reprodução dos símbolos originais;

6. A re-criação do sinal de mensagem original, com uma degradação definível em qualidade; a degradação é causada pela imperfeição do sistema.

Segundo [Haykin 2001], independentemente da forma do processo de comunicação sendo considerado, há três elementos básicos de todos os sistemas de comunicação, nomeadamente, transmissor, canal e receptor. O transmissor é localizado em um ponto no espaço, o receptor é localizado em outro ponto, separado do transmissor, e o canal é o meio físico que conecta esses dois elementos. O propósito do transmissor é converter o sinal de mensagem produzido pela fonte de informação em uma forma adequada para a transmissão através do canal. Entretanto, como o sinal transmitido se propaga ao longo do canal, ele é distorcido devido às imperfeições deste. Além disso, ruído e sinais interferentes (originados de outras fontes) são adicionados à saída do canal, com o resultado de que o sinal recebido é uma versão corrompida do sinal transmitido. O receptor possui a tarefa de operar no sinal recebido para reconstruir uma forma reconhecida do sinal de mensagem original. A Figura 6 ilustra os elementos básicos de um sistema de comunicação genérico.

[Haykin 2001] afirma que há basicamente dois modos de comunicação:

- ***Broadcasting***. Envolve o uso de um único transmissor potente e vários receptores. Neste, a informação flui em apenas uma direção.

- **Comunicação ponto-à-ponto**. O processo de comunicação ocorre ao longo de um *link* entre um único transmissor e um receptor. Neste caso, existe, normalmente, um fluxo bidirecional de informação, que requer o uso de um transmissor e um receptor em cada *link*.

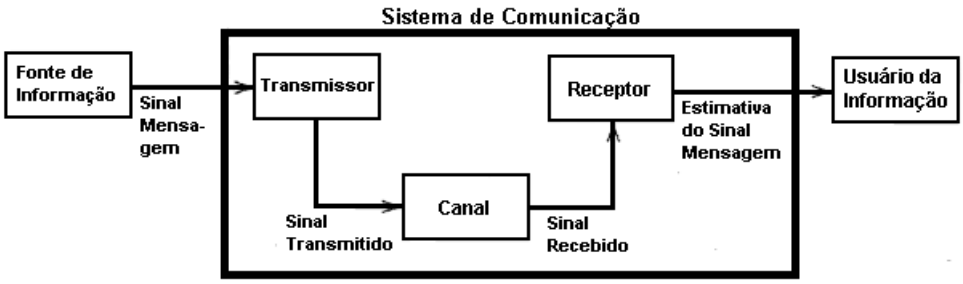

Figura 6 – Elementos de um sistema de comunicação genérico [Haykin 2001].

O modo de comunicação de *broadcasting* é exemplificado pelo rádio e televisão. Por outro lado, a telefonia é uma forma de comunicação ponto-à-ponto. Outro exemplo de comunicação ponto-à-ponto é o *link* entre uma estação na Terra e um robô navegando na superfície de um planeta distante [Haykin 2001].

Todos esses sistemas de comunicação, bem como outros, não mencionados anteriormente, compartilham uma característica em comum: o processo de comunicação em todos eles é de natureza estatística. O sinal transmitido, que sofre ruído e distorção, características intrínsecas do canal, precisa ser recuperado no receptor. Surge a necessidade de reverter a distorção do canal de comunicação. Um dos maiores obstáculos, como mencionado, é a propagação multicaminho em sistemas de comunicações digitais.

2.2 Propagação Multicaminho

Um sinal de rádio de espalha em diferentes direções à medida que ele se afasta da antena de transmissão. Algumas partes da onda que se espalha encontrarão superfícies refletoras. Em um ambiente urbano, por exemplo, a onda pode refletir em prédios, trens ou até mesmo aviões [McClaning 2012].

De acordo com [McClaning 2012], a propagação multicaminho ocorre quando um sinal chega ao receptor a partir de mais de uma rota de propagação. Um único sinal é transmitido por mais de um caminho: o caminho direto (*direct path*) e caminho refletido (*reflected path*). Considerando-se que ambos viajam a uma mesma velocidade e que os caminhos configuram distâncias diferentes entre o transmissor e o destino final, no recep-

tor, a forma de onda resultante torna-se distorcida. A Figura 7 ilustra o fenômeno da propagação multicaminho.

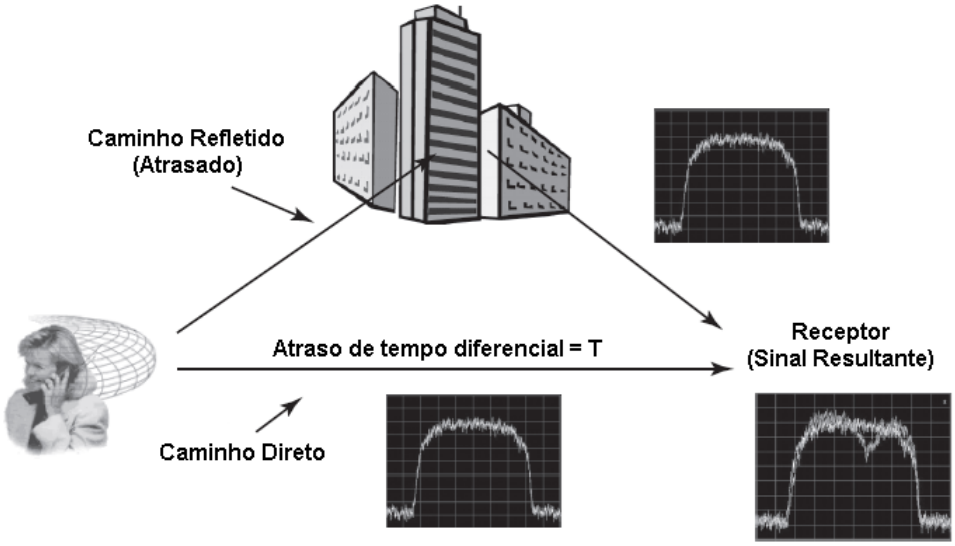

Figura 7 – Ilustração do problema da propagação multicaminho [McClaning 2012].

2.2.1 Modelo Matemático

O modelo matemático do fenômeno da propagação multicaminho pode ser mostrado utilizando-se do método de resposta ao impulso, usado para o estudo de sistemas lineares. Suponha que deseja-se transmitir apenas um único pulso de *Dirac* no tempo 0, isto é:

$$x(t) = \delta(t)$$

No receptor, devido à presença de múltiplos caminhos, mais de um pulso será recebido (suponha-se aqui que o canal possui largura de banda infinita, portanto a forma do pulso não é modificada). Cada um dos pulsos recebidos chegará ao receptor em tempos diferentes. De fato, uma vez que sinais eletromagnéticos viajam a velocidade da luz e sabendo-se que cada caminho tem um tamanho geométrico possivelmente diferente dos outros, há diferentes tempos de propagação. Portanto, o sinal recebido pode ser expresso por:

$$y(t) = h(t) = \sum_{n=0}^{N-1} \rho_n e^{j\phi_n} \delta(t - \tau_n) \tag{2.1}$$

$$\tau_n = \tau_n(t)$$

$$\rho_n = \rho_n(t)$$

$$\phi_n = \phi_n(t)$$

Na equação 2.1, N é o número de impulsos recebidos (equivalente ao número de caminhos), τ_n é o atraso do *i-ésimo* pulso e $\rho_n e^{j\phi_n}$ corresponde à amplitude complexa (*i.e.*, magnitude e fase) do *i-ésimo* pulso. Como uma consequência, $y(t)$ também representa a função de resposta ao impulso, $h(t)$, do modelo equivalente de propagação multicaminho. A Figura 8 ilustra a resposta ao impulso de um canal multicaminho.

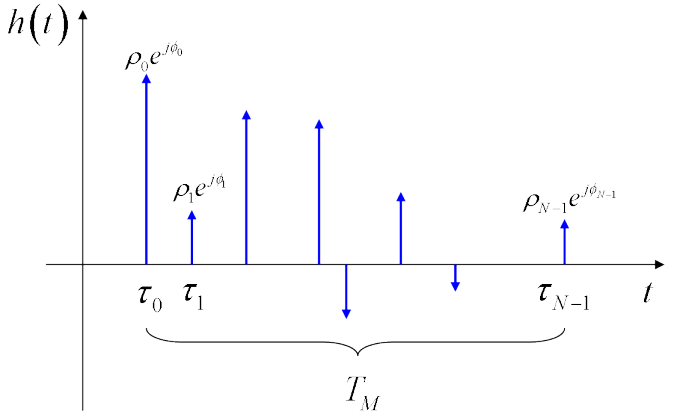

Figura 8 – Resposta ao impulso de um canal multicaminho.

Muito frequentemente, apenas um parâmetro é usado para denotar a intensidade das características/condições do canal multicaminho. Este é chamado de T_M e é definido como o tempo de atraso existente entre o primeiro e o último impulso recebido.

$$T_M = \tau_{N-1} - \tau_0$$

Considerando sistemas lineares, invariantes no tempo, o fenômeno pode ser caracterizado pela função de transferência $H(f)$, mostrada na Equação 2.2, que é definida como a transformada de *Fourier* da resposta ao impulso $h(t)$.

$$H(f) = \int_{-\infty}^{+\infty} h(t)\mathrm{e}^{-j2\pi ft}\,\mathrm{d}x = \sum_{n=0}^{N-1} \rho_n e^{j\phi_n}\mathrm{e}^{-j2\pi ft} \tag{2.2}$$

2.3 Interferência entre Símbolos

Uma das principais causas da interferência entre símbolos é a propagação multicaminho de um sinal. Por exemplo, um sinal *wireless* proveniente de um transmissor, que chega ao receptor via diferentes caminhos, devido à reflexão, refração e outros efeitos do meio físico. Desde que há vários caminhos propagando um sinal específico, isto resulta em diferentes versões do mesmo sinal chegando no receptor em diferentes tempos. Estes atrasos significam que um dado símbolo é espalhado em símbolos subsequentes e, portanto, interferem na detecção correta do sinal enviado. Além disso, vários caminhos frequentemente distorcem a amplitude e/ou fase de um sinal e consequentemente contribuem para a interferência com o sinal recebido [McClaning 2012].

3 Equalização e Estratégias de Adaptação

Nas últimas décadas, o campo de processamento digital de sinais e, particularmente, processamento adaptativo, desenvolveu-se bastante devido à disponibilidade crescente de tecnologia para a implementação de novos algoritmos. Esses algoritmos foram aplicados a um extensivo número de problemas, incluindo cancelamento de ruído e eco, equalização do canal, predição do sinal, bem como muitos outros [Diniz 2008].

Segundo [Diniz 2008], um filtro adaptativo pode ser entendido como um filtro digital que se automaticamente se modifica para ajustar seus coeficientes para minimizar uma função erro. Essa função erro, também referida como função custo, é a medida da distância entre o sinal de referência (desejado) e a saída do filtro adaptativo.

A configuração básica de um filtro adaptativo, operando em um domínio k, discreto no tempo, é ilustrado na Figura 9. Neste esquema, o sinal de entrada é denotado por $x(k)$, enquanto o sinal de referência, $d(k)$, representa o sinal de saída que se deseja. A saída do filtro adaptativo é denotada por $y(k)$, e o sinal de erro é definido como $e(k) = d(k) - y(k)$.

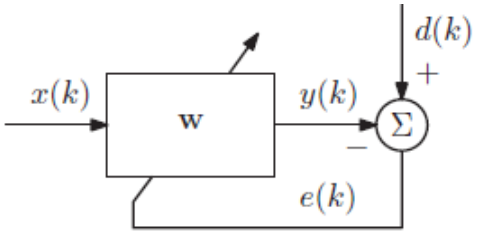

Figura 9 – Diagrama de blocos genérico de um filtro adaptativo [Diniz 2008].

O sinal de erro é usado pelo algoritmo de adaptação para atualizar os coeficientes do filtro adaptativo, denotados pelo vetor $w(k)$, de acordo com algum critério de performance. No geral, o processo de adaptação objetiva minimizar alguma métrica do sinal de erro, forçando o sinal de saída do filtro adaptativo se aproximar do sinal de referência [Diniz 2008].

3.1 Equalização do Canal de Comunicação

Como mencionado anteriormente, em sistemas de comunicação práticos, o sinal transmitido pode ser altamente distorcido pelo canal de transmissão. Uma forma de tratar esse problema é pelo uso de um filtro adaptativo na configuração de equalização do canal, como mostrado na Figura 10. Na ilustração, a sequência $s(k)$, conhecida pelo receptor, é enviada através de um dado canal de comunicações gerando um sinal distorcido. A mesma sequência $s(k)$, depois de sofrer um deslocamento temporal adequado para compensação

dos atrasos de transmissão, é usada como um sinal de referência no receptor para o filtro adaptativo, cuja entrada é o sinal distorcido. Quando a função erro se aproxima de zero, o sinal de saída $y(k)$ é muito próximo do sinal transmitido, $s(k)$, indicando que o filtro adaptativo está compensando as distorções impostas pelo meio físico. Após esse processo de treinamento, a informação desejada pode ser enviada através do canal, que está adequadamente equalizado pelo filtro adaptativo [Diniz 2008].

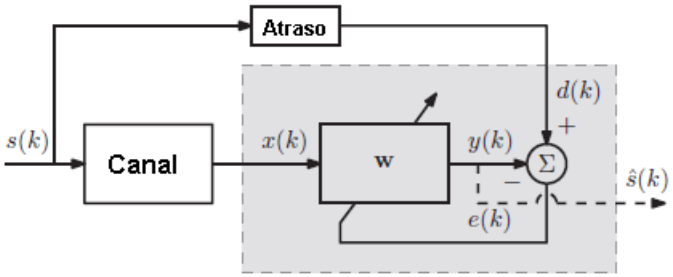

Figura 10 – Filtro adaptativo em configuração para equalização do canal: O sinal de saída $y(k)$ estima o sinal transmitido $s(k)$ [Diniz 2008].

Da discussão, até agora, é possível observar que o sinal de referência, através da definição do sinal de erro, age como um guia geral para todo o processo de adaptação. Na configuração da Figura 10 é possível identificar o bloco do filtro adaptativo mostrado na Figura 9. Como descrito em [Diniz 2008], para caracterizar completamente essa célula básica, três aspectos devem ser considerados:

1. **Estrutura do filtro adaptativo**. Este Trabalho de Conclusão de Curso foca nas estruturas de filtros digitais do tipo FIR, apresentados no capítulo 5. O relacionamento entrada-saída nesses filtros é descrito pela Equação 3.1, onde N é a ordem do filtro, $x(k)$ e w são vetores, compostos pelas amostras do sinal de entrada e os coeficientes do filtro, respectivamente.

$$y(k) = w_0 x(k) + w_1 x(k-1) + ... + w_N x(k-N)$$

$$y(k) = \sum_{i=0}^{N} w_i x(k-i)$$

$$y(k) = w^T x(k) \tag{3.1}$$

2. **Métrica de erro**. Como já mencionado, o algoritmo de adaptação ajusta os coeficientes adaptativos do filtro em uma tentativa de minimizar uma dada métrica de erro. Diferentes métricas requerem processos de adaptação diferentes, com características distintas. Neste Trabalho de Conclusão de Curso, o erro, denotado por $e(k)$, é sempre calculado conforme a Equação 3.2. Nos gráficos apresentados no capítulo 9, *Implementação do Filtro Equalizador*, os valores de $e(k)$ são positivos e negativos.

Dessa forma, para facilitar a análise, são utilizados os valores quadráticos do erro, conforme a Equação 3.3.

$$e(k) = d(k) - y(k) \tag{3.2}$$

$$e^2(k) = e(k) \times e(k) \tag{3.3}$$

3. **Algoritmo de adaptação**. Diversos procedimentos de otimização podem ser aplicados para o ajuste dos coeficientes do filtro. A técnica escolhida para esse Trabalho de Conclusão de Curso é apresentada na subseção 3.2.2 – *Least Mean Squares* – deste mesmo capítulo.

3.2 Algoritmos de Adaptação

Dentre os algoritmos de adaptação, este trabalho se limita à descrição do algoritmo *Least Mean Squares* (LMS), um dos mais conhecidos e utilizados em diversos projetos, devido à sua simplicidade e baixa exigência computacional.

3.2.1 Erro Quadrático Médio (MSE)

O erro quadrático médio é usado para expressar positivamente a medida do erro, $e(k)$, conforme apresentado na modelagem comportamental deste projeto. Além disso, esta subseção é requisito para a apresentação do modelo matemático do algoritmo LMS, na subseção a seguir [Diniz 2008].

O MSE é definido como:

$$\xi(k) = E[e^2(k)] = E[|d(k) - y(k)|^2] \tag{3.4}$$

Escrevendo-se o sinal de saída $y(k)$, como dado na Equação 3.1, obtem-se:

$$\xi(k) = E[e^2(k)] = E[|d(k) - w^T x(k))|^2]$$

$$\xi(k) = E[d^2(k)] - 2w^T E[d(k)x(k)] + w^T E[x(k)x^T(k)]w$$

$$\xi(k) = E[d^2(k)] - 2w^T p + w^T R w \tag{3.5}$$

Onde R e p é a matriz de correlação dos sinais de entrada e o vetor de correlação entre o sinal de referência e o sinal de entrada, respectivamente, e são definidos como:

$$R = E[x(k)x^T(k)] \tag{3.6}$$

$$p = E[d(k)x^T(k)] \tag{3.7}$$

Nota-se, das equações acima, que R e p não são representados como funções da iteração k, ou seja, não variam no tempo, devido à estacionariedade assumida da entrada e dos sinais de referência.

Da Equação 3.5, o vetor gradiente da função MSE, com respeito ao vetor de coeficientes do filtro adaptativo, é dada por:

$$\nabla_W \xi(k) = -2p + 2Rw \tag{3.8}$$

A solução de Wiener, w_o, que minimiza a função de custo $e(k)$, é obtida igualando-se o vetor gradiente na Equação 3.8 à zero. Assumido-se que R é não-singular, obtém-se:

$$w_o = R^{-1}p \tag{3.9}$$

3.2.2 Least Mean Squares (LMS)

O algoritmo *Least Mean Squares* (LMS) é um algoritmo de adaptação no qual a simplificação do cálculo do vetor gradiente é possível pela modificação da função objetivo. O algoritmo, bem como outros relacionados à ele, é amplamente utilizado em várias aplicações de filtragem devido à sua simplicidade computacional. As características de convergência do algoritmo LMS são examinadas para estabelecer uma faixa para o fator convergência, que garantirá estabilidade. A velocidade de convergência do LMS é dependente do espalhamento dos autovalores da matriz de correlação do sinal de entrada [Diniz 2008]. Esta seção apresenta a formulação do algoritmo, bem como discute algumas de suas propriedades e performance.

[Diniz 2008] explica que o algoritmo LMS é, de longe, o mais utilizado em filtragem adaptativa por muitas razões. Dentre suas principais características, vale destacar a baixa complexidade computacional, prova de convergência em um ambiente estacionário e comportamento estável, quando implementado com aritmética de precisão finita.

3.2.2.1 O Algoritmo Least Mean Squares

Na subseção *Erro Quadrático Médio (MSE)*, foi derivada a solução ótima para os parâmetros do filtro adaptativo. Esta solução leva à estimativa do sinal de referência (desejado), $d(k)$, através do erro quadrático médio (MSE). A solução ótima de Wiener é dada pela Equação 3.9.

Se uma boa estimativa da matriz R, denotada por $R'(k)$, e do vetor p, denotado por $p'(k)$, estão disponíveis, um algoritmo baseado no método *steepest-decent* pode ser

usado para achar a solução de Wiener da Equação 3.9, como segue:

$$w(k+1) = w(k) - \mu g'_W(k)$$

$$w(k+1) = w(k) + 2\mu(p'(k) - R'(k)w(k)) \tag{3.10}$$

Para $k = 1, 2, ...$, onde $g'_W(k)$ representa uma estimativa do vetor gradiente da função objetivo com respeito aos coeficientes do filtro.

Uma solução possível é estimar o vetor gradiente adotando-se estimativas instantâneas para R e p, como segue:

$$R'(k) = x(k)x^T(k) \tag{3.11}$$

$$p'(k) = d(k)x(k) \tag{3.12}$$

A estimativa do gradiente é dada por:

$$g'_W(k) = -2d(k)x(k) + 2x(k)x^T(k)w(k)$$

$$g'_W(k) = 2x(k)(-d(k) + x^T(k)w(k))$$

$$g'_W(k) = -2e(k)x(k) \tag{3.13}$$

Nota-se que, se a função objetivo, é trocada pelo erro quadrático instantâneo, $e^2(k)$, em vez do MSE, a estimativa do gradiente, mostrada acima, representa o verdadeiro vetor gradiente, desde que:

$$\frac{de^2(k)}{dw} = g'_W(k) \tag{3.14}$$

O algoritmo resultante, baseado no gradiente, é conhecido por algoritmo *Least Mean Squares* (LMS). Sua equação é dada por:

$$w(k+1) = w(k) + 2\mu e(k)x(k) \tag{3.15}$$

Onde o fator de convergência 2μ é também chamado de α, e deve ser escolhido de tal forma que garanta a convergência do algoritmo.

4 Circuitos Integrados Digitais

Em 1958, Jack Kilby construiu o primeiro Circuito Integrado (CI) de um *flip-flop*, com dois transistores, na empresa *Texas Instruments*. Já em 2003, o microprocessador *Intel Pentium* 4 continha 55 milhões de transistores. Isso corresponde à uma taxa composta de crescimento anual de 53% ao longo de 45 anos. Nenhuma outra tecnologia na história manteve uma taxa tão alta por tanto tempo [Weste e Harris 2004].

Esse crescimento foi possível devido à miniaturização dos transistores e aos avanços e melhorias nos processos de manufatura. A maioria dos outros campos da engenharia envolvem uma permuta (troca) entre potência, performance e preço. Entretanto, ao passo que os transistores ficaram menores, eles também dissipavam menos energia, e se tornaram mais rápidos e mais baratos para serem fabricados, segundo descrito em [Weste e Harris 2004]. Esse fato revolucionou não apenas a eletrônica, mas também a sociedade moderna.

4.1 Sistemas VLSI

O nível de integração de transistores em um único *chip* têm sido classificado como escala pequena, média, alta e muito alta. A Integração em Escala Muito Alta – mais conhecida do inglês *Very-large-scale Integration* (VLSI) – é o processo de criação de circuitos integrados pela combinação de centenas de milhares de transistores em um único *chip*. Esse tipo de integração começou nos anos 70, nos Estados Unidos, quando os semicondutores e as tecnologias de comunicação mais complexas e modernas estava sendo desenvolvidas [Weste e Harris 2004].

4.2 Projeto de Sistemas Digitais

Ao passo que a tecnologia avança, os sistemas digitais tornam-se cada vez mais complexos. Em seus níveis mais complexos, consistem de centenas de milhares de elementos, se vistos como uma coleção de portas lógicas ou transistores. De um ponto de vista mais abstrato, esses elementos podem ser agrupados em componentes funcionais como memórias *cache*, unidades de ponto flutuante, processadores de sinais ou controladores [Thomas e Moorby 2002].

[Thomas e Moorby 2002] descrevem que o processo de projeto de sistemas digitais começa como uma ideia conceitual de (1) um sistema lógico a ser construído, (2) um conjunto de restrições que a implementação final deve respeitar e (3) um conjunto de componentes primitivos que servirão na construção do sistema. O projeto é tipicamente

dividido em sub-projetos menores (seguindo a metodologia dividir-para-conquistar). Cada sub-projeto é, por sua vez, dividido em menores até que o original seja especificado em termos dos componentes primitivos que o construirão.

Segundo [Palnitkar 2003], com o advento da tecnologia VLSI (*Very-large-scale Integration*), devido à complexidade envolvida nos projetos de sistemas digitais, não era mais possível verificar a funcionalidade dos mesmos em uma *protoboard* ou outra plataforma física. Dessa forma, técnicas de projeto auxiliadas por computador (*Computer-aided Techniques* – CAT) se tornaram críticas para verificação e projeto.

[Thomas e Moorby 2002], similarmente, apontam que, no mesmo contexto, também criou-se a necessidade de uma linguagem padrão para descrição de circuitos digitais. Como resultado, tem-se o crescimento das linguagens de descrição de *hardware* (*Hardware Description Language* – HDL) como ferramentas importantes no projeto de sistemas digitais, com um número muito grande de elementos, e grande faixa de abstração eletrônica e lógica.

4.2.1 Verilog Hardware Description Language

Verilog é uma linguagem de descrição de *hardware* (HDL) utilizada para projetar e documentar sistemas eletrônicos. Segundo [Palnitkar 2003], ela se tornou uma das HDLs mais utilizadas na industria e academia.

[Palnitkar 2003] explica que Verilog é uma linguagem de programação com dois paradigmas: comportamental e estrutural. Detalhes de um módulo podem ser definidos em até quatro níveis de abstração, dependendo da necessidade do projeto. O módulo comporta-se identicamente para com o ambiente externo independentemente do nível de abstração em que o mesmo foi descrito. Os diferentes níveis de abstração fornecidos por Verilog são descritos abaixo:

1. ***Switch-level***: Este é o nível mais baixo de abstração. Um módulo pode ser implementado em termos de chaves, nós e conexões entre eles;

2. ***Gate-level***: O módulo é implementado em termos de portas lógicas e conexões entre elas. O projeto neste nível é similar ao projeto em termos de um diagrama à nível de portas-lógicas;

3. ***Data-flow***: Abstração à nível de fluxo de dados. Neste, o projetista trata de como ocorre o fluxo de dados entre registradores e como esses dados são processados no projeto.

4. ***Behavioral*** (ou comportamental): Esta é a abstração de mais alto-nível permitida. Um módulo pode ser implementado sem a necessidade da descrição de detalhes do

hardware ou circuito físico. O projeto, neste nível, é similar ao de linguagens de alto-nível como C.

Verilog permite que o projetista misture todos os quatro níveis de abstração em um único sistema. Na comunidade de projeto digital, o termo *Register Transfer Level* (RTL) é frequentemente usado para uma descrição em Verilog que use uma combinação entre os níveis gate, *data-flow* e *behavioral* e que seja aceitável de um ponto de vista de síntese lógica.

Devido às vantagens citadas, e ao conhecimendo prévio de Verilog pelo autor, decidiu-se pela utilização dessa linguagem de descrição de *hardware* no desenvolvimento do código a nível RTL deste Trabalho de Conclusão de Curso.

4.2.2 Metodologia de Projeto para Sistemas Digitais

Segundo [Palnitkar 2003] e [Thomas e Moorby 2002], há basicamente dois tipos de metodologias para sistemas digitais: *top-down* e *bottom-up*. Na metodologia *top-down*, primeiramente define-se o bloco de nível mais alto e, a partir deste, identifica-se e define-se os sub-blocos necessários para construção do bloco de nível mais alto original. A Figura 11 ilustra o conceito.

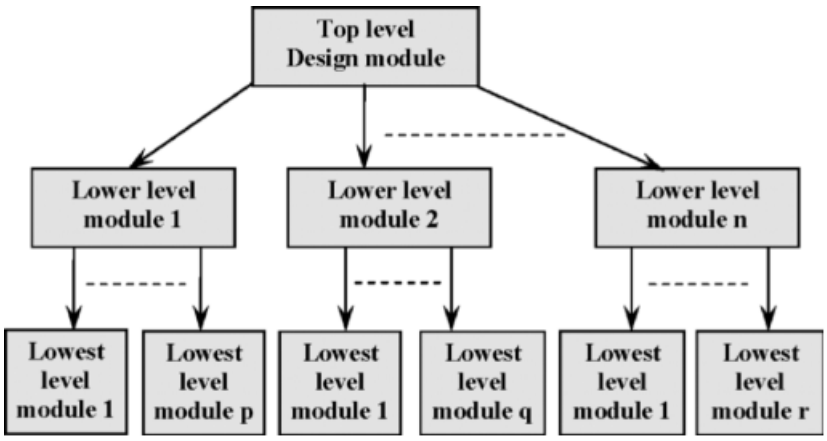

Figura 11 – Metodologia de projeto *top-down* [Palnitkar 2003].

Na metodologia *bottom-up*, faz-se o contrário. Primeiramente são identificados os blocos que estão disponíveis. A partir do uso destes blocos, outros blocos de maior complexidade são construídos até que se alcance o bloco de nível mais alto. A Figura 12 mostra a metodologia *bottom-up* de projeto.

Na prática, normalmente se utiliza uma combinação das duas metodologias. A partir de uma arquitetura definida, os dois métodos são mesclados ao decorrer do projeto.

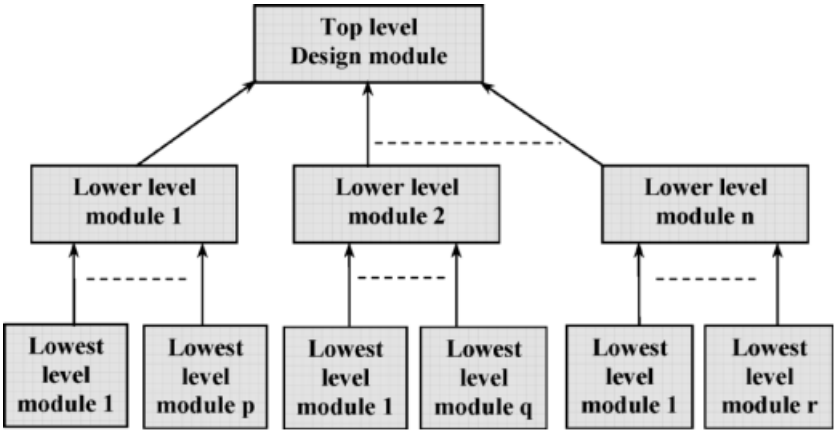

Figura 12 – Metodologia de projeto *bottom-up* [Palnitkar 2003].

4.3 Fluxo de Projeto VLSI

Segundo [Thomas e Moorby 2002], o fluxo de projeto é o conjunto de procedimentos que permite projetistas de circuitos integrados progredirem da especificação de um produto, para a implementação final do mesmo, de uma forma sistemática e livre de erros. [Ramachandran 2007], define um fluxo genérico de projetos VLSI, focado em FPGAs e ASICs, como mostrado nas Figuras 13, 14 e 15.

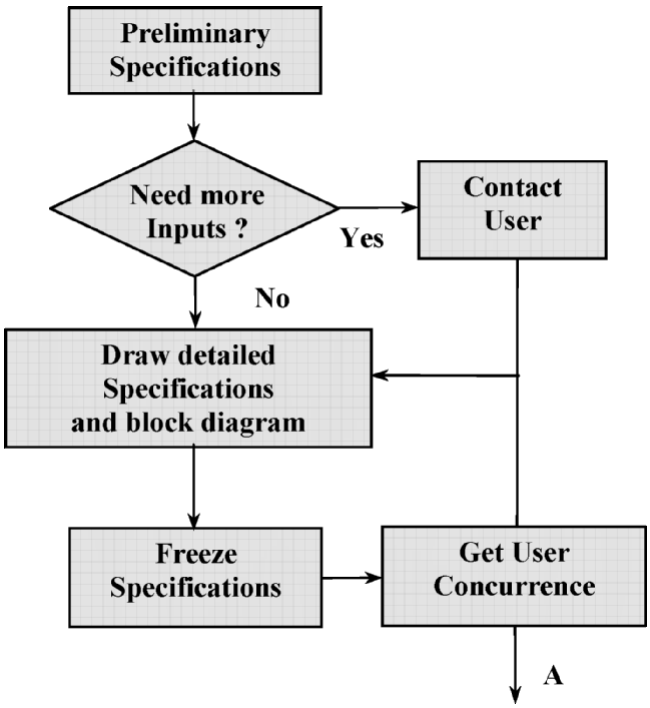

Figura 13 – Fluxo de projetos VLSI – Parte 1 [Ramachandran 2007].

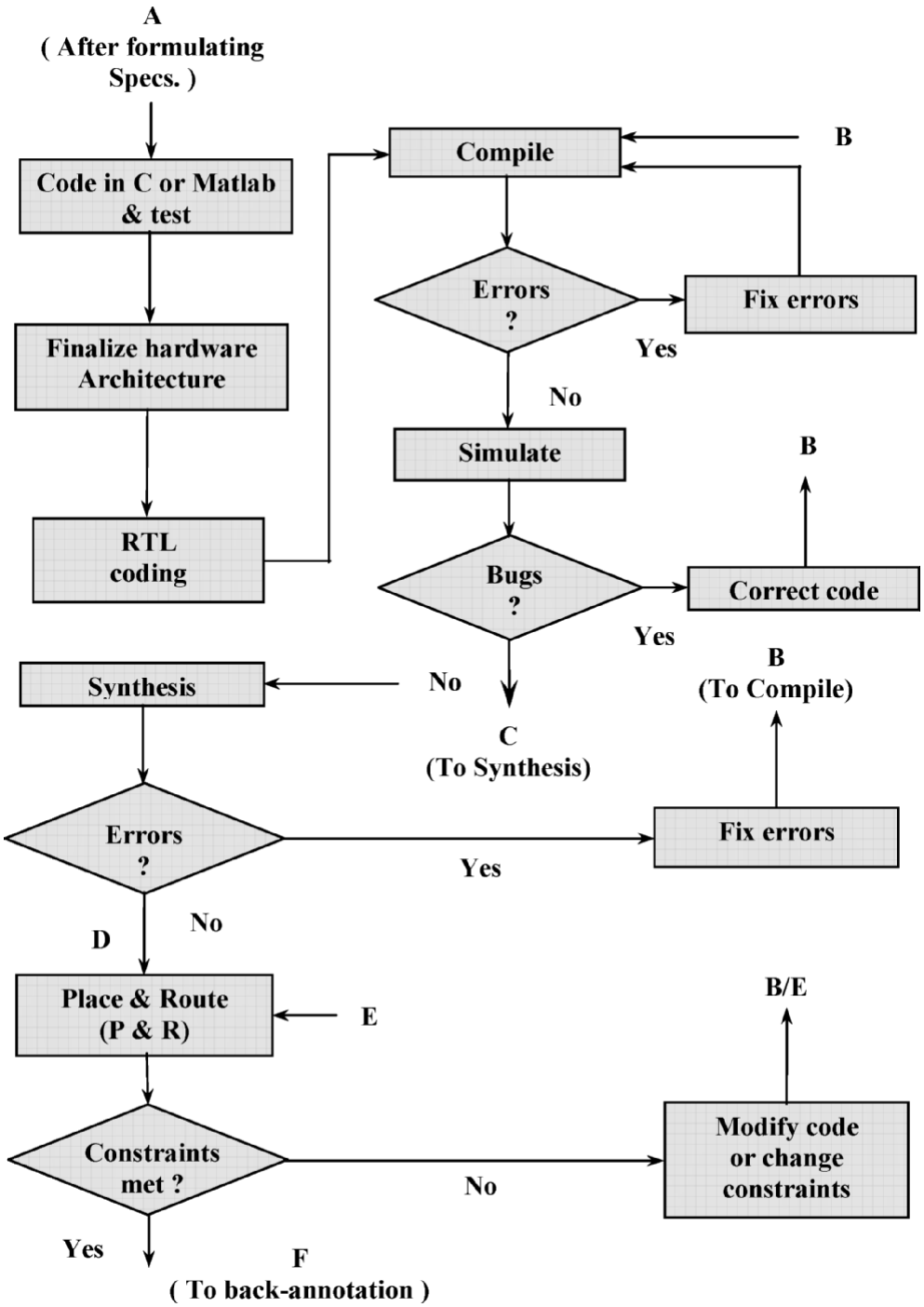

Figura 14 – Fluxo de projetos VLSI – Parte 2 [Ramachandran 2007].

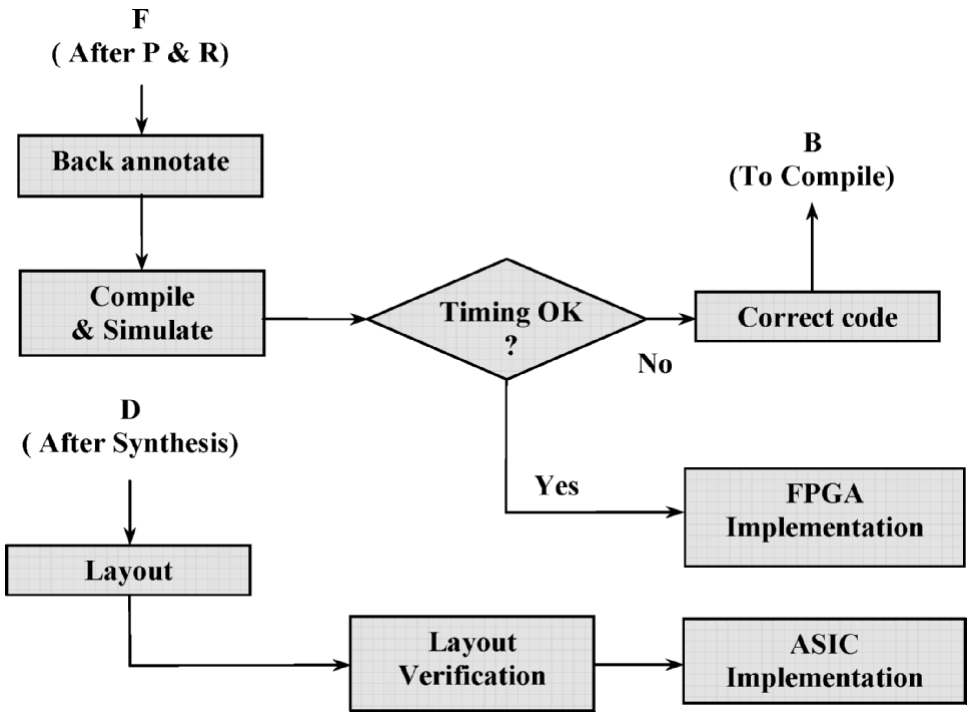

Figura 15 – Fluxo de projetos VLSI – Parte 3 [Ramachandran 2007].

4.3.1 Especificações

Uma vez que o produto é identificado, o primeiro passo consiste em formular suas especificações preliminares. Estas, são discutidas e avaliadas com possíveis futuros usuários, e, uma vez que estão prontas, é feito um diagrama de blocos do sistema, que será, posteriormente, validado e aprovado, finalizando a primeira parte do projeto [Ramachandran 2007].

4.3.2 Modelagem Comportamental

Uma vez que as especificações do sistema estão prontas, faz-se a implementação dos algoritmos ou da arquitetura proposta em uma linguagem de programação de alto-nível, tal como C, C++ ou MATLAB. Após a validação dessa etapa, gera-se todas as possíveis combinações de entradas para o sistema, ao passo que, mede-se suas respectivas saídas. Estes dados serão utilizados na validação do sistema em linguagem de descrição de *hardware* [Ramachandran 2007].

4.3.3 Modelagem Funcional

No estágio subsequente, faz-se tradução dos algoritmos ou da arquitetura descrita anteriormente, em uma linguagem de descrição de *hardware*, tal como Verilog. Este processo deve ser feito cuidadosamente, levando-se sempre em consideração a circuitaria resultante da síntese lógica e física do sistema. Como já mencionado, os testes de fun-

cionalidade são feitos com os vetores gerados na etapa anterior à esta [Ramachandran 2007].

4.3.4 Síntese Lógica

O próximo passo consiste na síntese lógica: a partir de um algoritmo ou arquitetura descrito(a) em uma linguagem de descrição *hardware* (HDL), sintetiza-se uma nova *netlist*, dessa vez, mapeada de acordo com as *standard cells* da tecnologia que está sendo utilizada. O novo código HDL, com informações de *timing*, *fan-in*, *fan-out*, etc, deve ser novamente simulado e validado. Além da *netlist*, a síntese lógica fornece relatórios de *timing*, potência e área. Estes documentos devem ser levados em consideração com as especificações do projeto (*constraints*), que não podem ser violadas [Ramachandran 2007].

4.3.5 Síntese Física

Após a validação da saída do estágio anterior, a etapa seguinte é relacionada à síntese física (também conhecida como *layout* e *place & route*). Há alguns programas de computador especializados nessa etapa. Um exemplo é o *Encounter Digital Implementation* da *Cadence*. Verificações de tempo, conectividade, *antenna*, densidade de metal, *design rule check*, entre outras, devem ser feitas à fim de validar este passo [Ramachandran 2007].

4.3.6 Pós-Síntese Física

Uma vez que não há violações, o projeto pode ser implementado em um ASIC. Dessa forma, ferramentas de *back-end* são usadas para combinação com outros blocos, roteamento, posicionamento e alinhamento dos pinos, além de outros *checks* [Ramachandran 2007].

5 Algoritmos e Arquiteturas para Processamento Digital de Sinais

Um algoritmo é uma sequência finita de instruções para criar e/ou transformar dados. Uma sistema de propósito geral, ou processador, pode ser programado, através de uma linguagem de programação, para executar uma variedade de algoritmos. Entretanto, sua arquitetura pode não render a melhor performance para uma aplicação em particular ou ser um desperdício, em termos de *hardware*, para outras aplicações. Comparando-se com circuitos integrados de aplicação específica (*Application-specific Integrated Circuits* – ASICs), um sistema de propósito geral pode consumir mais energia, requerer maior área efetiva e, consequentemente, possuir maior custo [Ciletti 2005].

ASICs são projetados para otimização de algoritmos particulares para uma aplicação específica. A arquitetura desses sistemas é customizada para que se alcance o melhor benefício entre performance, área e consumo, para uma certa aplicação, diferentemente de sistemas de propósito geral, que priorizam flexibilidade. Dessa forma, são especialmente adequados para processamento digital de sinais e comunicações de dados. Filtros digitais são aplicações comuns desses sistemas.

Este capítulo introduz filtros digitais como um sistema de processamento de sinais. Dois tipos básicos de filtros digitais são apresentados, citando-se suas vantagens e desvantagens. Além disso, 3 diferentes topologias de filtros FIR são analisadas. Por fim, é descrita uma seção relacionada ao processo de projeto de filtros FIR, mostrando e analisando diferentes topologias e implementações, de acordo com os objetivos do trabalho.

5.1 Filtros Digitais

Segundo [Ciletti 2005], um filtro digital é um sistema que executa operações matemáticas em sinais amostrados, discretos no domínio do tempo, para reduzir ou acentuar certos aspectos ou características desses sinais. Mais especificamente, esses sistemas transformam representações digitais de sinais analógicos para remover ruído ou componentes não desejadas e também para tratar as características espectrais do sinal resultante. Filtros digitais operam em uma representação digital de precisão finita de um sinal. Consequentemente, no projeto de filtros digitais deve-se levar em conta os efeitos do tamanho da palavra (*word length*) da amostragem do sinal analógico, os coeficientes do filtro, do número de *taps* (que também define sua ordem) e as operações aritméticas que executa.

Filtros digitais operam no domínio do tempo, recebendo uma sequência de pala-

vras digitais, discretas no tempo e de tamanho específico, para produzir uma sequência de símbolos em sua saída. A sequência de entradas, $x(t)$, analógica, é primeiramente discretizada e quantizada no domínio do tempo, gerando a sequência, $x(n)$ – a qual, no domínio-z é representada por $X(z)$. Esta, serve de entrada para os filtros digitais. Duas arquiteturas para filtros digitais lineares são representadas pelo diagrama de blocos da Figura 16. O filtro de resposta finita de duração ao impulso (também conhecido por filtro FIR, do inglês *Finite Impulse Response*) tem sua saída como uma soma ponderada de suas entradas. Por outro lado, o filtro IIR (*Infinite Impulse Response*) forma sua saída como uma soma ponderada de suas entradas e de valores passados de sua própria saída [Ciletti 2005].

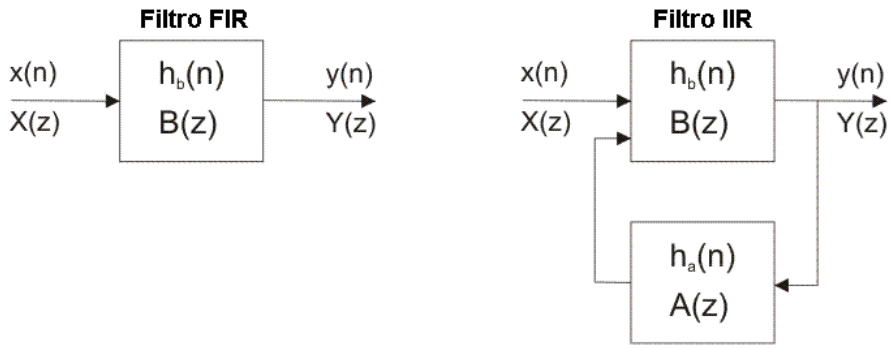

Figura 16 – Filtros digitais FIR e IIR, respectivamente.

O projeto de filtros digitais visa duas importantes características: causalidade e fase linear. Um filtro é dito causal se sua resposta ao impulso é nula (igual à zero) antes do estímulo ser aplicado. Já a fase linear se refere à propriedade em que a resposta em fase do filtro é uma função linear da frequência [Oppenheim e Schafer 2009]. De acordo com [Ciletti 2005], filtros do tipo FIR são amplamente utilizados em projetos práticos devido à possibilidade de serem projetados para apresentaram fase linear, o que garante que o sinal de saída do filtro seja deslocado no tempo (atrasado), mas continue sendo uma cópia não distorcida do sinal de entrada. Por outro lado, filtros IIR projetados para possuírem fase linear não podem apresentar a característica de causalidade. Além disso, outra distinção importante é que os filtros do tipo FIR não acumulam erros de quantização. Contrariamente, filtros IIR podem acumular erros de quantização uma vez que sua saída é sucessivamente realimentada.

Devido às importantes características apresentadas (causalidade e fase linear), somadas às vantagens de implementação e prévio conhecimento do autor, decidiu-se pela utilização de um filtro do tipo FIR para o Trabalho de Conclusão de Curso em questão.

5.2 Filtros Digitais Finite Impulse Response

Um filtro digital do tipo FIR forma sua saída como uma soma ponderada das amostradas presentes e passadas de suas entradas, como descrito pela equação 5.1 abaixo, onde $x[n]$ é o sinal de entrada, $y[n]$ é o sinal de saída, b são os coeficientes e N é a ordem do filtro (um filtro de ordem N, possui $N+1$ termos no lado direito da equação).

$$y[n] = b_0 x[n] + b_1 x[n-1] + ... + b_N x[n-N]$$

$$y[n] = \sum_{i=0}^{N} b_i x[n-i] \tag{5.1}$$

O diagrama de blocos funcional de um filtro digital do tipo FIR no domínio-z é apresentado na Figura 17. O diagrama representa os caminhos de dados e operações que devem ser executadas. Os blocos z^{-1} denotam um atraso de um ciclo de relógio. Cada estágio do filtro trata uma amostra atrasada no tempo do sinal de entrada. As conexões entre o barramento de entrada e os estágios de saída são chamadas de *taps* e o conjunto de números b_i são chamados de coeficientes. Os somadores e multiplicadores do filtro devem ser projetados para uma arquitetura específica (considerando-se o tamanho da palavra digital) e devem ser rápidos o suficiente para formar a saída $y[n]$ antes do próximo ciclo de relógio [Ciletti 2005].

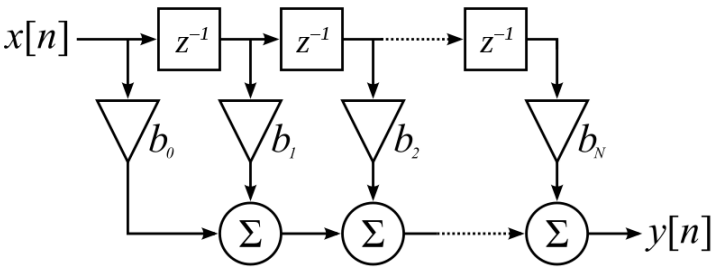

Figura 17 – Diagrama de blocos de um filtro FIR.

5.2.1 Processo de Projeto de Filtros FIR

De acordo com [Ciletti 2005], um processo para projeto orientado à ASIC ou FPGA de filtros digitais FIR tem os principais passos apresentados na Figura 18. O projeto começa com as especificações e requerimentos para frequência de corte, limites de transição, *ripple* na banda passante, atenuação mínima, etc. A partir desse ponto, o filtro é, primeiramente, descrito por uma linguagem de alto nível, como C, C++ ou MATLAB, e então, convertido em um modelo em Verilog, que posteriormente, será sintetizado em *hardware*, de acordo com a tecnologia. Entretanto, o processo não é ideal e alguns pontos devem

ser seriamente considerados durante o projeto. Por exemplo, em C, C++ ou MATLAB, as variáveis são representadas por números ponto-flutuante, mas em Verilog, os parâmetros e outros valores de dados são expressados em formato ponto-fixo, de tamanho finito. Essa representação, intrinsecamente, introduz erros de quantização e truncamento no modelo, pois há uma limitação no tamanho de bits de uma palavra digital. Similarmente, as representações dos coeficientes numéricos do filtro também contribuem para erros de quantização e truncamento adicionais, pois estão sujeitas ao mesmo problema. Deve-se ainda levar em consideração possíveis situações de *overflow* e *underflow* que podem ocorrer devido às operações matemáticas executadas no filtro.

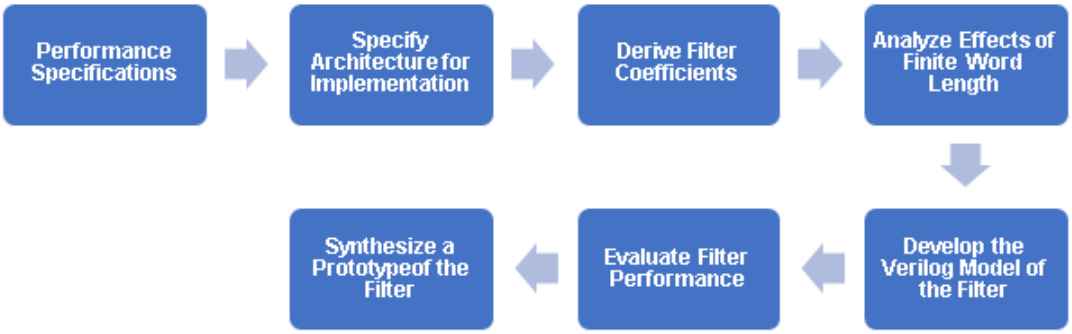

Figura 18 – Processo de projeto de filtros digitais FIR.

Várias arquiteturas podem ser utilizadas na implementação de filtros digitais do tipo FIR. Para uma dada topologia há vantagens e desvantagens associadas à seu uso. Os requerimentos do produto final (isto é, performance, área, consumo de energia e custo) geralmente definem a arquitetura mais adequada. Três diferentes topologias serão apresentadas e analisadas.

5.2.2 Filtro FIR de Forma Direta I (DF-I)

A Figura 19 mostra um filtro FIR de Forma Direta I (DF-I). Essa estrutura é um das mais populares. Por utilizar os elementos de atraso imediatamente após o ponto de entrada do sinal, $x(n)$, sua saída fica dependente do maior atraso – neste caso, $x(n-4)$. Na ilustração, a linha vermelha mostra o caminho crítico do sistema em comparação com o filtro FIR de Forma Direta-Transposta (TDF) – que será apresentado na próxima subseção –, enquanto que a linha vermelha tracejada mostra o caminho crítico real. Nota-se que o sinal deve propagar-se por todos os elementos de atraso, do primeiro ao quarto, para que todos os coeficientes sejam multiplicados. Essa *latência*, não desejada, de pelo menos 4 ciclos de relógio, é uma característica intrínseca desse tipo de topologia. O caminho crítico para o filtro em questão é de 6 elementos de atraso (*flip-flops* do tipo D), somados à um multiplicador, mais um somador.

Vale ressaltar que o filtro DF-I possui uma estrutura bastante regular, o que facilita

sua implementação lógica e física. Além disso, se comparado ao filtro FIR de Forma Direta-Transposta (TDF) e ao filtro FIR Totalmente Serial (FS), o DF-I é um intermediário: é menor que o TDF, porém mais lento; é maior que o FS, apresentado subsequentemente, porém mais rápido.

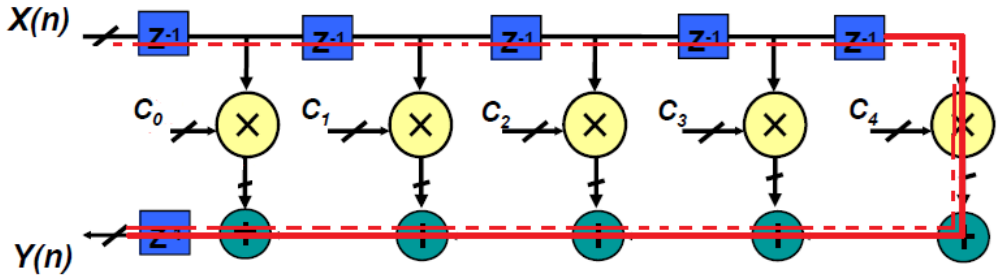

Figura 19 – Filtro FIR de Forma Direta I (DF-I).

5.2.3 Filtro FIR de Forma Direta-Transposta (TDF)

A arquitetura do filtro FIR de Forma Direta-Transposta (TDF) é apresentada na Figura 20. Diferentemente da topologia anterior, esta apresenta os elementos de atraso localizados após a saída dos multiplicadores. Apesar de não ser tão intuitiva, essa diferença é essencial para otimização do caminho crítico do circuito. O novo caminho crítico é de apenas 2 *flip-flops* do tipo D (FF-D), somados à um multiplicador e um somador. Entretanto, o *encurtamento* do caminho crítico e, consequentemente, a melhora na performance do circuito, é compensado pelo aumento da área efetiva. De fato, posicionar os elementos de atraso imediatamente após a saída dos multiplicadores é um custo alto em termos de área. Por exemplo, suponha-se que o filtro FIR (independentemente de sua topologia) deva suportar palavras digitais de entrada de 8 bits. Suponha-se, ainda, que foi definido que os coeficientes também são dados em palavras de 1 *byte*. Sendo assim, a saída do multiplicador, para que não haja *overflow*, deve garantir, pelo menos, 16 bits de largura. Dessa forma, o filtro FIR de Forma Direta I implementa 5 vezes 8 elementos de atraso, enquanto que o filtro FIR TDF implementa 5 vezes 16 elementos de atraso (não considerando os FF-D da saída). Portanto, o filtro FIR de Forma Direta-Transposta, apesar de melhor performance, perde para a estrutura direta em termos de área e regularidade.

5.2.4 Filtro FIR Totalmente Serial (FS)

A Figura 21 ilustra um filtro FIR Totalmente Serial (FS). Considerando-se um mesmo projeto, com mesmo número de *taps* e mesma largura de bits de entrada, a arquitetura FS é a que apresenta a área efetiva significantemente menor, dentre as três mostradas. No entanto, como compensação, esta estrutura possui uma característica de baixa performance. Nota-se que, além da latência intrínseca, similar ao filtro direto, devido aos

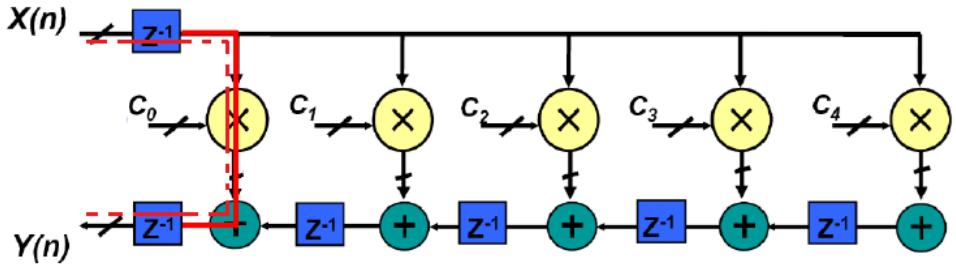

Figura 20 – Filtro FIR de Forma Direta-Transposta (TDF).

atrasos posicionados imediatamente à entrada do filtro, esta topologia implementa apenas 1 multiplicador e 1 somador. Dessa forma, todas as multiplicações necessárias são feitas em apenas 1 multiplicador, a cada passo do contador. A arquitetura *fully serial* é também a mais complexa, em termos de desenvolvimento (implementação à nível de código HDL). A sincronia entre o multiplexador do sinal de entrada e dos coeficientes, o multiplicador e somador deve ser perfeita para que o funcionamento do filtro seja correto e adequado.

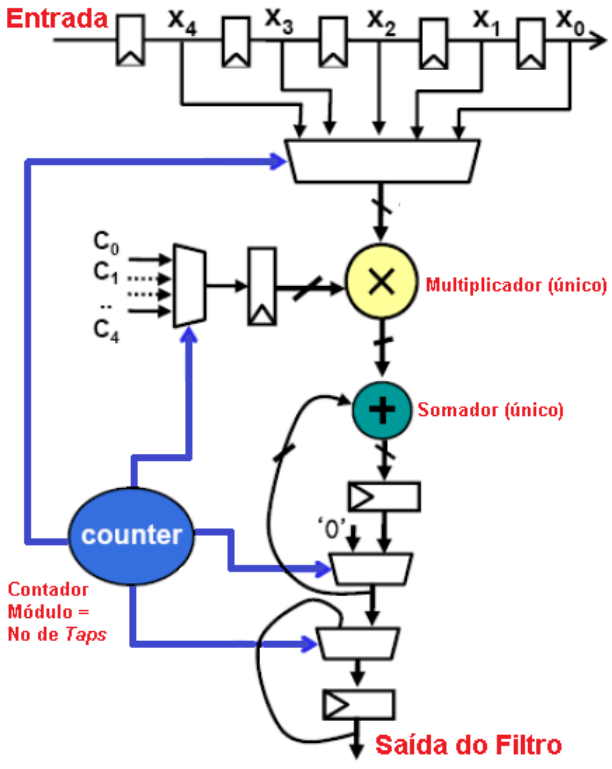

Figura 21 – Filtro FIR Totalmente Serial (*Fully Serial* – FS).

5.3 Chips FIR Comerciais

A Figura 22 mostra alguns *chips* FIR comerciais para diversas aplicações, tais como decimação, equalização adaptativa, filtragem em bandas de frequência adequadas,

etc [Wanhammar 1999].

Company	Model	Description
Inmos	A100	4-, 8-, 12-, or 16-bit coeff., 16-bit input data, 16 × 16-bit mult. with 36-bit accum., and 24-bit-rounded output 32-taps (fixed length) Throughput 15.00 MHz, 4-bit coeff., 3.75 MHz, 16-bit coeff 84-pin PGA–flatpack package
Harris	HSP43891	9-bit coeff., 9-bit input data, 9 × 9-bit mult. with 26-bit accum., and 26-bit output 8 taps per device Throughput 30 MHz Support for decimation by 2, 3, or 4 High-speed synchronous inputs and outputs 84-pin PGA–PLCC package
Harris	HSP43881	8-bit coeff., 8-bit input data, 8 × 8-bit mult. with 26-bit accum., and 26-bit output 8 taps per device Throughput 30 MHz Support for decimation by 2, 3, or 4 84-pin PGA–PLCC package
Harris	HSP43168	10-bit data and coeff Dual FIR with 8 taps or single FIR with 16 taps Support for decimation up to a factor 16 Throughput 40 MHz 84-pin PGA–PLCC package
Harris	HSP43220	20-bit coeff. and 16-bit input data Two-stage FIR filter for decimation. The first stage can decimate with a factor up to 1024. The second stage has up to 512 taps and can decimate with a factor 16. Throughput 30 MHz 84-pin PGA–PLCC package
Motorola	DSP56200	24-bit coeff., 16-bit input data, 24 × 16-bit mult. with 40-bit accum. 32-bit or 16-bit-rounded output. 4–256 taps (selectable) Throughput Single FIR filter 227 kHz, 32-taps, 1-device, 37 kHz, 256-taps, 1-device, 37 kHz, 1024-taps, 4-devices Two FIR filters: 123 kHz, 32-taps, 1-device, 36 kHz, 128-taps, 1-device Adaptive FIR filter: 19 kHz, 256-taps, 1-device, 19 kHz, 1024-taps, 4-devices, 115 kHz, 256-taps, 8-devices 28-pin DIP package

Figura 22 – *Chips* FIR Comerciais.

6 Representação Numérica em Sistemas Digitais

Muitos dos sistemas de processamento digitais de sinais, como os filtros mostrados no capítulo 5, requerem a execução repetitiva de operações aritméticas. Dessa forma, é importante que a implementação dessas operações, em um sistema maior, seja o mais eficiente possível. A implementação desse projeto, entretanto, utiliza arquiteturas de processamento aritmético (*e.g.*, somadores, multiplicadores, subtratores, etc) providos pela tecnologia, no momento da síntese lógica.

Este capítulo tem como objetivo explanar os fundamentos de representação numérica em sistemas digitais, uma vez que, esses conceitos foram utilizados no projeto e implementação do sistema aqui proposto.

6.1 O Sistema de Representação Binário

Números são representados por uma *string* de caracteres em um sistema de notação posicional tendo uma dada raiz ou base. O sistema de números binários tem dois símbolos e uma raiz de 2. Todas as máquinas digitais codificam números em palavras bits. A palavra tem um tamanho fixo e a interpretação do padrão de bits depende do formato de codificação usada pelo sistema [Ciletti 2005].

6.1.1 Representação de Frações

Em geral, um número binário pode ser expressado como uma soma ponderada de potências de 2 ascendentes e descendentes, de acordo com a Equação 6.1 [Ciletti 2005].

$$B = b_{n-1}b_{n-2}...b_1b_0b_{-1}b_{-2}...b_{-m+1}b_{-m} \tag{6.1}$$

No sistema decimal, cada dígito de um número representa uma potência crescente de 10 a esquerda do ponto decimal, e uma potência decrescente de 10 a direita do ponto decimal. Já no sistema binário, o conceito é o mesmo, exceto que os dígitos a direita do ponto-fixo representam uma potência decrescente de 2. As Figuras 23 e 24 ilustram as respectivas representações.

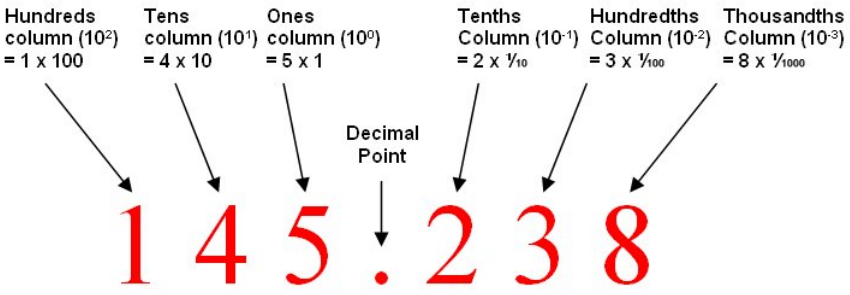

Figura 23 – Representação de Decimal-fração para Binário.

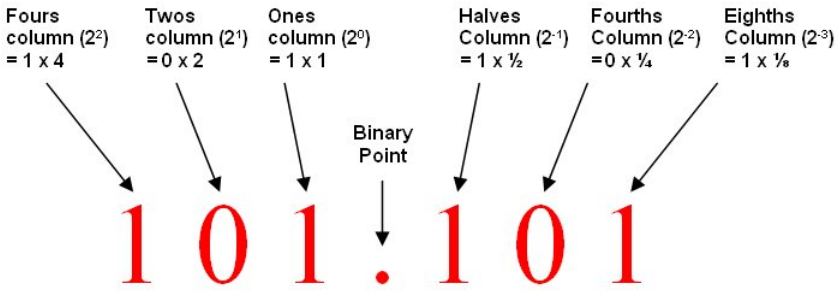

Figura 24 – Representação de Binário para Decimal-fração.

6.1.2 Representação de Números Binários com Sinal e Magnitude

Segundo [Ciletti 2005], nesse tipo de representação de números positivos e negativos, o bit mais significativo da palavra, também chamado de MSB, é o bit codificado como o de sinal. O valor 0 representa um número positivo, enquanto que 1 representa um valor negativo. Os bits restantes da palavra, representam a magnitude do número. Por exemplo, 0111_2 corresponde à $+7_{10}$, e 1111_2 representa -7_{10}. A representação para números de 8 bits é mostrada na Figura 25.

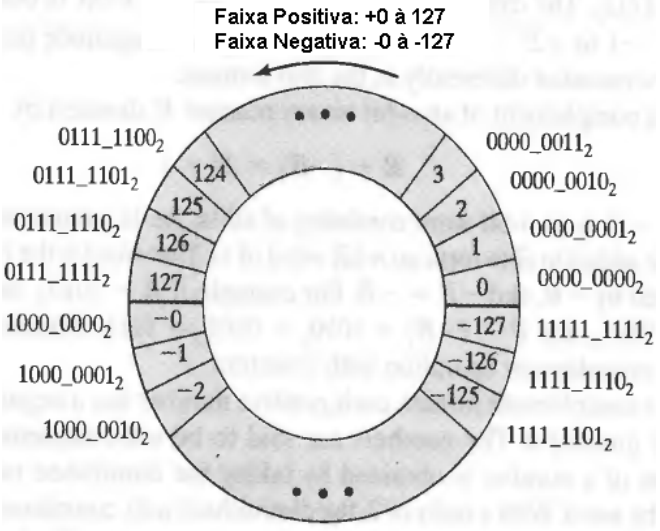

Figura 25 – Representação de binários com sinal e magnitude [Ciletti 2005].

Se os sinais de dois números nesse formato são iguais, adição é executada diretamente pela soma das magnitudes (não dos bits de sinal). O sinal do resultado é o mesmo dos operandos. Se os bits de sinal dos números não são iguais, os sinais e magnitudes relativas das palavras devem ser usados para determinar se deve-se adicionar ou subtrair os números. Essa representação tem a desvantagem de utilizar duas representações para o número 0 [Ciletti 2005].

6.1.3 Representação em Complemento de 1 para Números Negativos

Números positivos são representados nesse sistema da mesma forma que no sistema de sinal e magnitude. Entretanto, números negativos são representados diferentemente. A Figura 26 ilustra a representação em complemento de 1.

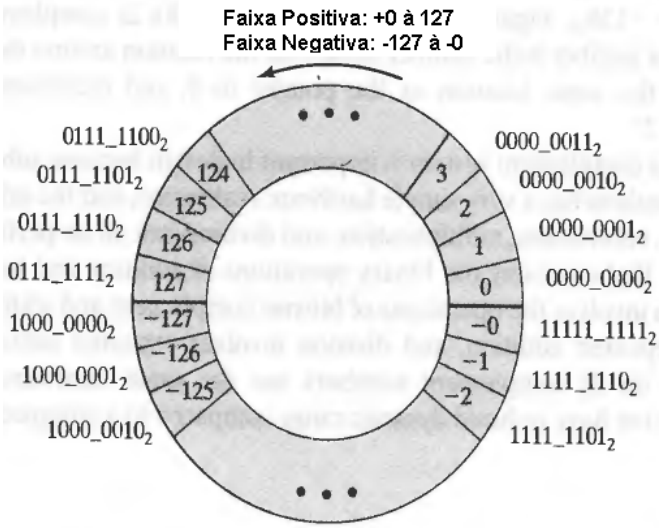

Figura 26 – Representação de binários em complemento de 1 [Ciletti 2005].

O complemento de 1 de um número binário B de n-bits é denotado por $-B$ e definido como na equação 6.2. Nesse formato, cada número positivo tem uma contraparte negativa, inclusive o número zero. Os números são ditos serem auto complementares, pois o complemento de um número é obtido tomando-se o inverso do original (*bitwise not*). Nessa representação, para implementar a subtração de dois números A e B, basta fazer $A - B = A + (\sim B)$. Nota-se, entretanto, que o número 0_{10} também possui duas representações: 00000000_2 e 11111111_2. Por isso, a maioria das máquinas digitais usa a representação em complemento de 2, que possui uma representação única para cada número, inclusive o 0 [Ciletti 2005].

$$B + (-B) = 2^n - 1 \tag{6.2}$$

6.1.4 Representação em Complemento de 2 para Números Negativos

O complemento de 2 de um número binário B de n-bits é definido por $B^* = 2^n - B$. Logo, $B + (B^*) = 2^n = 0$ módulo n. Somando-se 1 ao complemento de 1 de uma palavra, forma-se seu complemento de 2. A Figura 27 ilustra a faixa de valores compreendida pelos números em complemento de 2.

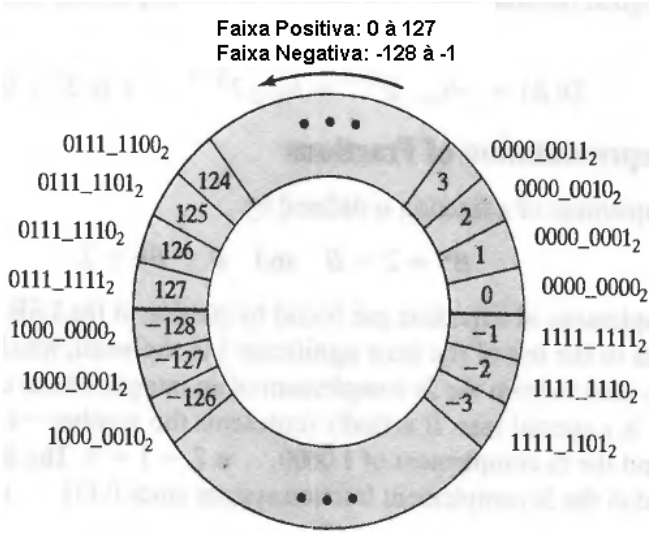

Figura 27 – Representação de binários em complemento de 2 [Ciletti 2005].

Parte II

Projeto e Implementação

7 Considerações Importantes de Projeto

Este capítulo descreve alguns pontos importantes que devem ser considerados no projeto e implementação do *design* em questão, como o fluxo de ferramentas computacionais, implementação em ponto-fixo ou ponto-flutuante e a geração de entradas em ponto-fixo para o modelo funcional.

7.1 Fluxo de Ferramentas Computacionais (CAD/EDA)

Considerando-se o diagrama de blocos das Figuras 13, 14 e 15, fica claro que é importante que se defina as ferramentas computacionais CAD/EDA (*Computer-aided Design/Electronic Design Automation*) que serão utilizadas em cada estágio da implementação do produto que foi proposto.

7.1.1 Especificações

Partindo das especificações do produto junto aos usuários prospectivos, o primeiro passo consiste no desenho do diagrama de blocos detalhado do sistema. Para tanto, utiliza-se alguma ferramenta de desenho (e simulação, se necessário). O autor optou pelo desenho do diagrama em um *software* comum, básico, já proveniente do sistema operacional *Windows* 7.

7.1.2 Modelagem Comportamental

Uma vez que o diagrama de blocos do sistema está pronto, a etapa que se sucede é referente à implementação dos algoritmos e da arquitetura proposta em uma linguagem de programação de alto-nível, tal como C, C++ ou MATLAB. Por sua familiaridade, além das diversas *toolboxes* e recursos disponíveis, o autor deste trabalho optou pelo uso do MATLAB. As *toolboxes* que estão sendo utilizadas neste trabalho são: *Fixed-Point Designer* e *Signal Processing*. No entanto, essas ferramentas adicionais, não são essenciais (leia-se obrigatórias) para a implementação: todas as funções providas por elas também podem ser codificadas pelo projetista.

7.1.3 Modelagem Funcional

Concluída a validação dos algoritmos e arquitetura proposta, na etapa anterior, faz-se a codificação em uma linguagem de descrição de *hardware*. O autor decidiu por fazer esta implementação em Verilog, pois já é familiarizado com esta linguagem de programação.

Além disso, optou-se pelo uso do editor de textos *Sublime Text 2*, junto com uma extensão para programação em Verilog.

Com os códigos *.v* prontos, a compilação foi feita com o auxílio da ferramenta computacional NC*Launch* (compilador: NC-Verilog), disponível ao usuário após a instalação do pacote *Encounter Digital Implementation*. Os testes de pré-síntese lógica foram feitos com o *software Cadence SimVision*, utilizando-se dos vetores de teste gerados anteriormente, com o MATLAB.

7.1.4 Síntese Lógica

O autor decidiu por fazer a síntese lógica utilizando-se da ferramenta *Cadence* RTL *Compiler*. Visando automatização do processo, o autor também fez a codificação dos *scripts* necessários para este estágio. Esses *scripts* estão anexados à este documento. Após a análise dos relatórios de tempo, área e potência, fez-se os testes pós-síntese lógica com o mesmo *software* utilizado anteriormente, o *SimVision*, da *Cadence*.

7.1.5 Síntese Física

Com os resultados da síntese lógica, a etapa subsequente consiste na síntese física do projeto (*i.e.*, *layout*, *place & route* e verificações). Para tanto, autor utilizou o *software Cadence* RTL-*to*-GDSII do pacote *Encounter Digital Implementation*. Além disso, o autor/projetista também foi responsável por implementar todos os *scripts* referentes à este estágio. Esses códigos estão disponíveis em anexo à este documento.

7.1.6 Pós-Síntese Física

Uma vez que o projeto em questão foi realmente levado à fabricação junto com outros projetos no *chip* **UNB0414**, o produto gerado na etapa anterior foi levado ao *software Cadence Virtuoso* para compor o topo do **UNB0414**. Essa parte da implementação foi feita pelo autor e pelos projetistas da empresa DFChip.

7.2 Implementação em Ponto-fixo vs. Ponto-flutuante

Segundo [Khan 2012], de uma perspectiva de projeto, um algoritmo pode ser implementado usando formato ponto-fixo ou ponto-flutuante. O formato ponto-flutuante armazena um número em termos de mantissa e expoente. O *hardware* que provê suporte à este formato, depois de executar cada cálculo, automaticamente mede a mantissa e atualiza o expoente para fazer com que o resultado caiba (possa ser expressado) em uma quantidade de números de bits em uma forma definida. Todas essas operações fazem o

hardware necessário para o formato ponto-flutuante mais caro, em termos de área e energia consumida, comparado ao *hardware* para operações em ponto-fixo.

Um sistema projetado em ponto-fixo, após a execução de um cálculo, não mede ou atualiza a posição do ponto decimal. Pelo contrário, nesse tipo de arquitetura, a responsabilidade é do desenvolvedor. O ponto decimal é fixado para cada variável e pré-definido. Ao se fixar o ponto em uma determinada posição, a variável terá uma faixa fixa de valores. Como existe essa limitação, se o resultado do cálculo não está dentro da faixa de dados possíveis, então a informação é perdida ou corrompida – *overflow* e *underflow* [Khan 2012].

A implementação de algoritmos de processamento de sinais, como os filtros mostrados no capítulo 5, em um formato ponto-fixo é direta. Devido ao baixo consumo de energia e menor área, arquiteturas que utilizam formato ponto-fixo são muito comuns em aplicações com poucos recursos disponíveis. Projetos em ponto-fixo são amplamente utilizados em soluções para comunicação e multimídia [Khan 2012].

Devido às características do projeto em questão, a implementação em ponto-fixo torna-se mais atraente. Observa-se que, apesar de sua inflexibilidade, a implementação em ponto-fixo resultaria em maior performance, menor área e menor consumo de energia. O *trade-off* entre os dois formatos, nesse caso, tende mais para o ponto-fixo em detrimento do ponto-flutuante. Dessa forma, o filtro equalizador adaptativo descrito nesta seção será implementado utilizando o formato ponto-fixo.

7.2.1 Formato Qn.m para Aritmética de Ponto-fixo

A maioria dos projetos de processamento de sinais e sistemas de comunicação são primeiramente implementados em precisão *double* em formato ponto-flutuante, usando uma ferramenta computacional como o MATLAB ou alguma linguagem de programação como C ou C++. Essa implementação inicial foca principalmente na assimilação da funcionalidade correta do algoritmo/sistema. O código MATLAB, C ou C++ é então convertido em um formato ponto-fixo. Para tanto, as variáveis de ponto-flutuante e constantes, na simulação, são convertidas em um formato ponto-fixo, *Qn.m*. Esse é um sistema de números posicionados fixamente para representação de números em ponto-flutuante. A Figura 28 ilustra os campos de bits e seus pesos em uma palavra finita de 9 bits [Khan 2012].

O formato *Qn.m*, de um conjunto de números de N bits, define n bits para a esquerda e m bits para a direita do ponto binário. Nos casos de números com sinal, o MSB é usado para o sinal e este tem um peso negativo. O complemento de dois de um número em ponto-fixo no formato *Qn.m* é equivalente à Equação 6.1, apresentada no capítulo 6.

Figura 28 – Campos de bits e seus pesos [Khan 2012].

7.3 Geração de Entradas: Quantização em Ponto-Fixo

No sistema da Figura 30 deve-se considerar que o sinal transmitido, usualmente uma sequência de símbolos em binário (ou polar, caso use-se uma codificação BPSK) sofre a ação do canal ISI e do ruído. Este último, distorce o sinal em amplitude, de forma que, um símbolo pode passar a não ter mais um valor 0 ou 1 (-1 ou 1 na BPSK), mas algo entre esses dois extremos, normalmente fracionário. Nesse caso, deve-se quantizar o valor de forma a adequá-lo ao formato em ponto-fixo *Qn.m*, já explicado. Para tanto, utilizou-se a *toolbox Fixed-point Designer* do MATLAB. Com a função *fi*, é possível a geração de números quantizados em formato ponto-fixo *Qn.m* [Khan 2012].

Um exemplo é o valor de π, que é igual à 3.1415, aproximadamente. Para descrever esse número no formato de ponto-fixo *Q3.5*, pode-se utilizar a função fi da *toolbox* mencionada. A Figura 29 ilustra o exemplo.

```
>> PI = fi(pi, 1, 3+5, 5)

PI =

    3.1563

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 8
        FractionLength: 5
>> bin(PI)

ans =

01100101

>> double(PI)

ans =

    3.1563
```

Figura 29 – Quantização de π igual à 3.1415, em um número de formato ponto-fixo de palavra de tamanho 8 bits, sendo 5 a direita do ponto binário.

8 Modelo do Sistema

Este capítulo aborda a modelagem do sistema e do canal ISI.

8.1 Descrição e Diagrama de Blocos do Sistema

O problema, a ser descrito em MATLAB, é apresentado na Figura 30 abaixo. Como mostrado, o transmissor envia para o receptor uma sequência de símbolos através de um canal dispersivo (interferência entre símbolos). A distorção imposta pelo Canal ISI, se soma ao ruído gaussiano branco (AWGN – *Additive White Gaussian Noise*). O sinal, recebido pelo receptor, passa pelo bloco Equalizador, que, por sua vez, tenta reverter a distorção do canal, fazendo com que $x'(n)$ seja o mais próximo possível de $x(n)$. Para tanto, o equalizador adaptativo é projetado para trabalhar em dois modos: (1) modo treino e (2) modo de decisão direta. Em (1), uma sequência, previamente conhecida, é enviada através do canal de comunicação. O equalizador ajusta os seus coeficientes adaptativos para que a resposta em frequência do equalizador compense as distorções do meio físico. Uma vez que os coeficientes ótimos são definidos, o sistema entra no modo (2), o qual funciona normalmente como um equalizador.

8.2 Modelagem do Sistema em MATLAB

O sistema da Figura 30 foi descrito em MATLAB, seguindo o passo do fluxo de projetos apresentado no capítulo 4. Os códigos em ponto-flutuante e ponto-fixo estão anexados à este documento.

O projeto em MATLAB tem como objetivo fixar o aprendizado dos algoritmos que serão utilizados e entender o comportamento do sistema como um todo. Uma vez que a modelagem está correta e se comportando de forma estável, gera-se as entradas do sistema (como descrito na subseção 7.3 do capítulo 7). Uma vez que se tenha um conjunto de entradas, pode-se aplica-las ao sistema a fim de se obter as suas respectivas respostas/saídas (que expressam o correto funcionamento do projeto). Essas saídas são, então, comparadas às saídas do modelo em HDL, para verificação funcional do mesmo. Esse procedimento é mais conhecido como *vector matching*.

8.3 O Modelo Matemático do Canal

Considere o modelo em banda base complexo de transmissão de uma sequência de símbolos através de um canal dispersivo, como mostrado na Figura 31. O sinal enviado

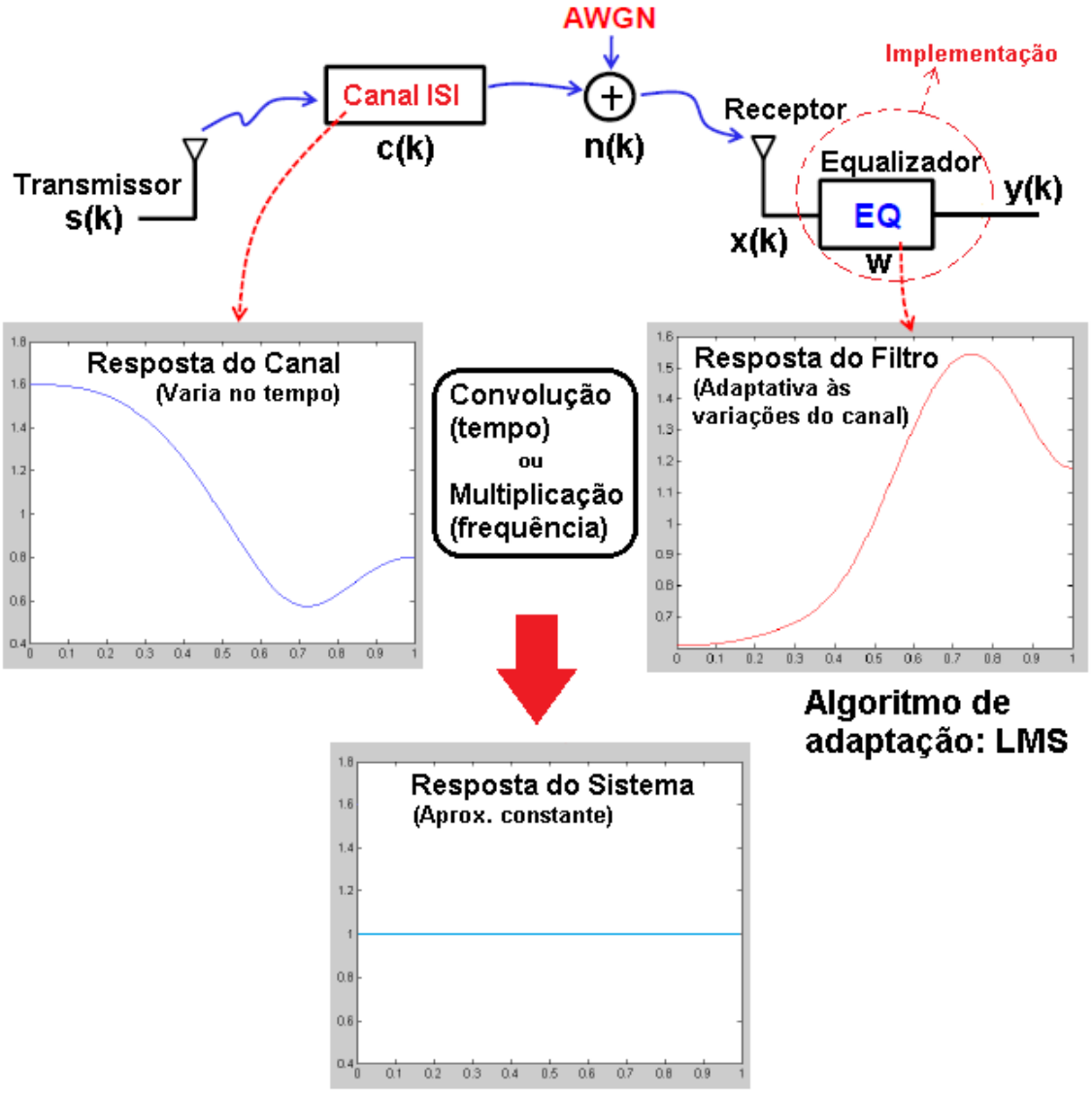

Figura 30 – Diagrama de blocos do sistema.

pelo canal é dado pela Equação 8.1 [Madhow 2008].

$$u(t) = \sum_{n=-\infty}^{\infty} b[n] g_{TX}(t - nT) \tag{8.1}$$

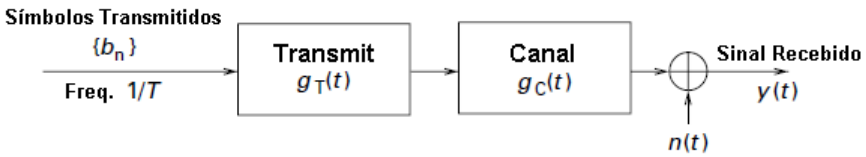

Figura 31 – Modelo do sistema de transmissão de uma sequência de símbolos através de um canal dispersivo [Madhow 2008].

Na Equação 8.1, $g_{TX}(t)$ é a resposta do impulso do filtro transmissor e $b[n]$ é a sequência de símbolos, transmitidos a taxa de $1/T$. O canal é modelado como um filtro

com resposta ao impulso $g_c(t)$, seguido pelo ruído $n(t)$ (AWGN). Assim, o sinal recebido é dado por:

$$y(t) = \sum_{n=-\infty}^{\infty} b[n]p(t - nT) + n(t) \tag{8.2}$$

$$p(t) = (g_{TX} * g_c)(t) \tag{8.3}$$

A Equação 8.3 é a resposta ao impulso da cascata do filtro transmissor e do canal. A tarefa do equalizador é extrair a sequência transmitida $b = b[n]$ do sinal recebido $y(t)$ [Madhow 2008].

Um exemplo, apresentado em [Madhow 2008], considera a ilustração da Figura 32. A taxa de símbolos é 1/2 (*i.e.*, um símbolo a cada dois unidades de tempo). O pulso transmitido $g_{TX}(t) = I_{[0,2]}(t)$ é um pulso retangular ideal no domínio do tempo, enquanto a resposta do canal $g_c(t) = \delta(t-1) - \frac{1}{2}\delta(t-2)$ corresponde à dois caminhos discretos (multicaminhos).

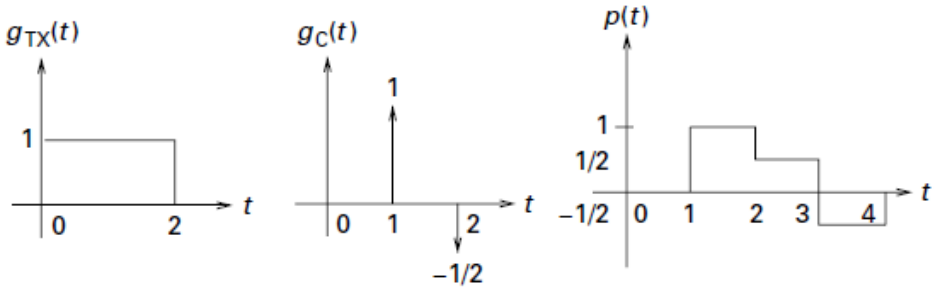

Figura 32 – Exemplo de transmissão em um canal dispersivo de dois caminhos [Madhow 2008].

9 Implementação do Filtro Equalizador

9.1 Parâmetros do Filtro Equalizador Adaptativo

Esta seção descreve os resultados dos efeitos da variação de alguns parâmetros do filtro digital equalizador adaptativo. Estes parâmetros são importantes para o projeto e para implementação do dispositivo em questão, pois afetam diferentemente a qualidade do modelo, performance, área e potência dispensada.

9.1.1 Efeitos da Variação do Valor da Constante

Dado o modelo do sistema em MATLAB, apresentado na subseção anterior, é possível simular e extrair os valores dos parâmetros do filtro equalizador adaptativo de forma a definir os próximos passos do projeto (codificação em HDL, síntese, *place & route*, *layout*, etc.).

Dado o modelo em MATLAB, simulações foram feitas com o intuito de entender melhor o efeito da constante 2μ na resposta do sistema. Para tanto, variou-se os valores de 2μ e analisou-se o efeito do valor do erro (isto é, $e[n] = d[n] - y[n]$). Neste caso, desejou-se a análise de valores positivos do erro, portanto, os gráficos mostram o valor quadrático de $e[n]$. Outro ponto importante, é que a variável *err_est* é uma **média** do valor quadrático do erro, $e^2[n]$, depois de 2000 iterações – a partir da iteração número 1000. Dessa forma, *err_est* estima o valor do erro por meio de uma média, após 1000 iterações do sistema.

A Figura 33 ilustra os vários valores de $\alpha = 2\mu$. Na figura, para todas os diferentes valores da constante, o número de *taps* do filtro equalizador é sempre o mesmo e igual à 6.

Pela análise dos resultados mostrados na figura, ao aumentar-se o valor de α até certo ponto, é possível se obter uma convergência mais rápida (melhor, do ponto de vista de performance) da medida de erro do filtro equalizador adaptativo em questão. Os valores 0.0625 e 0.03125 foram pensados pelo projetista e autor deste trabalho considerando a implementação física do filtro. Estes dois, em particular, podem ser obtidos simplesmente pelo deslocamento a direita dos bits do valor do multiplicando. Por exemplo, para efetuar a multiplicação de um multiplicando qualquer pelo valor 0.03125, basta deslocar (através de um *shift* bits) 5 bits para a direita (o que significa dividir por 2 elevado à quinta potência). Dessa forma, pode-se dispensar a utilização de um multiplicador para efetuar tais operações, diminuindo-se a área, custo e energia consumida no projeto em foco. Neste projeto considerou-se o valor de α **igual à 0.03125**.

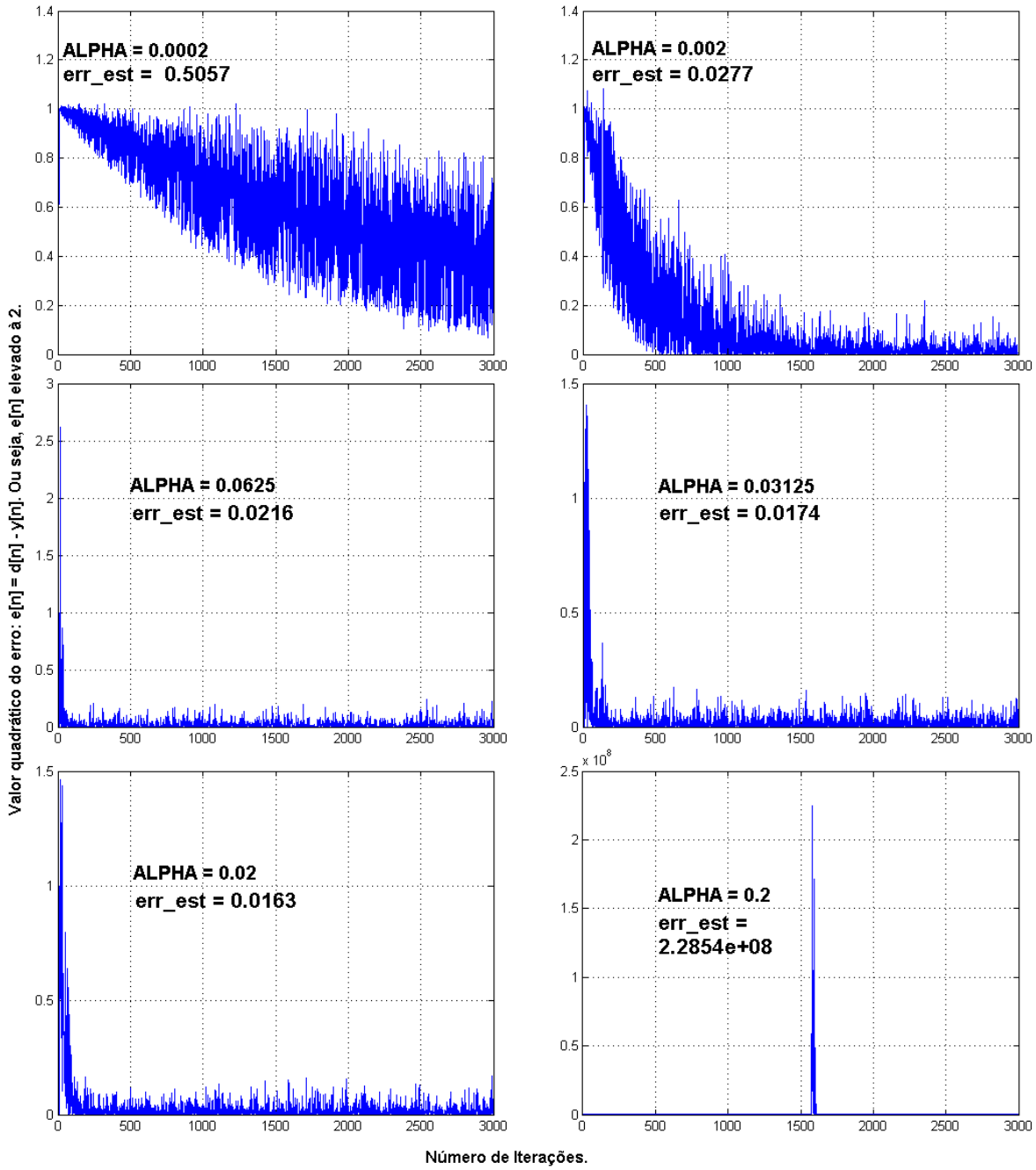

Figura 33 – Efeito na variação da constante α.

9.1.2 Efeitos da Variação da Ordem do Filtro

Da mesma forma que a constante α, também deseja-se entender o efeito da variação do número de *taps* do filtro equalizador adaptativo da saída do sistema. Dessa forma, fixou-se um valor de α e variou-se o número de *taps* do filtro. Nesse caso, $\alpha = 2\mu = 0.001$. A Figura 34 mostra os resultados.

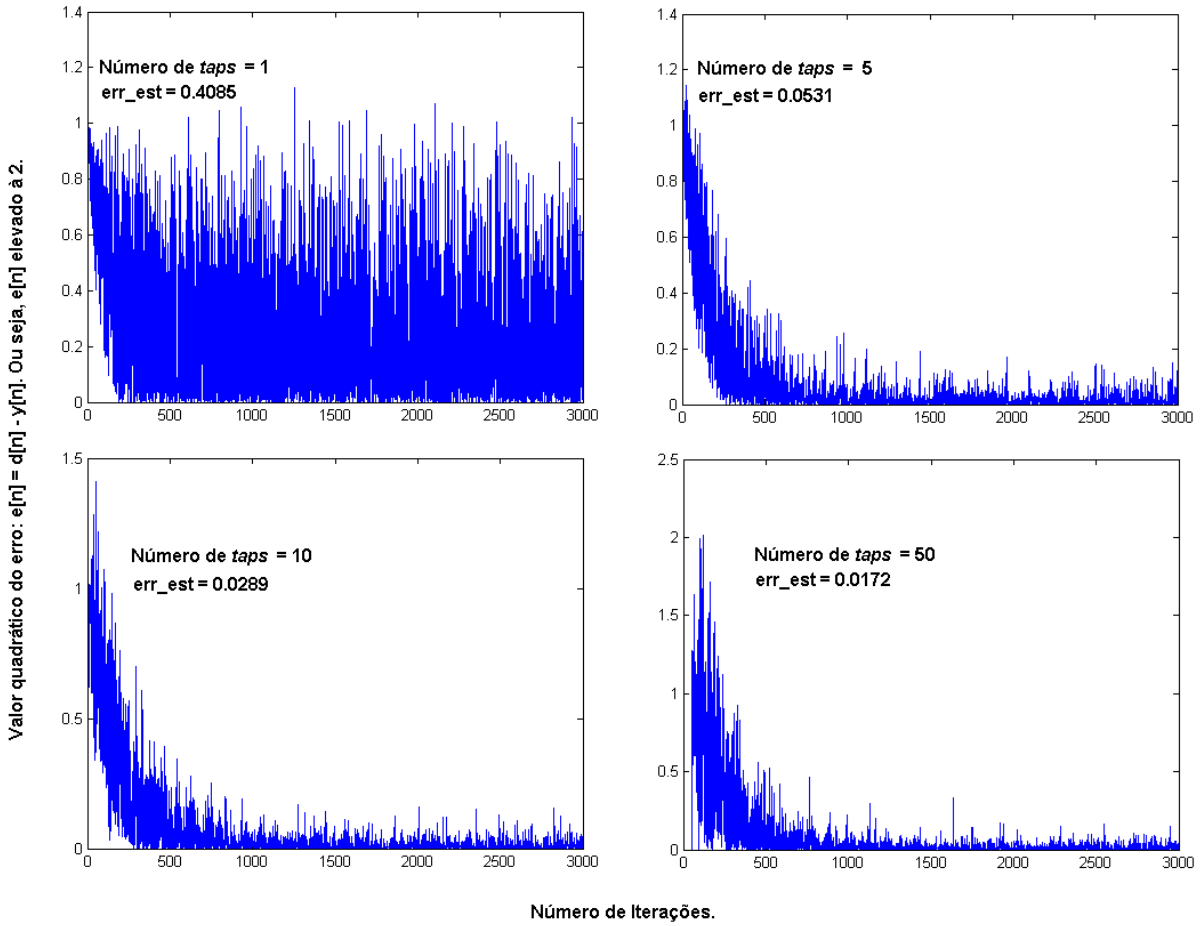

Figura 34 – Efeito na variação da ordem do filtro (número de *taps*).

Analisando-se a figura, é possível observar que a variação na convergência do filtro, para um número de *taps* **maior ou igual à 5**, é bastante sutil. Apesar de haver uma certa melhora na convergência à medida que o número de *taps* aumenta, existe também, uma compensação em área e em consumo de energia. Isto é, a medida em que se aumenta o número de *taps*, melhora-se a convergência (diminui-se o erro) do filtro equalizador adaptativo, entretanto, aumenta-se (e muito) o consumo, a área efetiva e o custo do projeto.

Observa-se que a troca de melhor performance por maior área, custo e consumo de energia, nesse caso, não é interessante para a aplicação. Nota-se que o ganho (melhor performance) não supera as perdas (maior área, consumo e custo).

Levando-se em consideração os pontos descritos acima, um bom *design* teria o número de *taps* entre 5 ou 6. Dessa forma, projetista e autor deste documento escolheu o *design* de um **filtro de ordem 6**.

9.2 Realização do Filtro Equalizador

Esta seção define a realização do filtro equalizador *Least Mean Squares* adaptativo ao canal de comunicação. Para tanto, explica-se a tecnologia, arquitetura e os passos seguidos em cada etapa de projeto.

9.2.1 Tecnologia Utilizada

A tecnologia utilizada para a síntese lógica e física, bem como todas as verificações mencionadas, foi a CMOS **TSMC 0.18 μm** (tsmc18). O ***Process Design Kit* (PDK)** digital é o **TCB018GBWP7T**. As bibliotecas digitais escolhidas foram as de caso típico (*typical case*), referentes à **TCB018GBWP7TTC**. A síntese física foi feita utilizando-se dos arquivos LEF e *CapTable* de 6 metais. Em resumo:

- **PDK Analógico**: tsmc18 (CMOS TSMC 0.18 μm);
- **PDK Digital**: TCB018GBWP7T;
- **Biblioteca Digital**: TCB018GBWP7TTC (*typical case*);
- ***I/O Library***: TPAN18LPGV2;
- ***Timing Libraries* (LIB)**: tcb018gbwp7ttc.lib;
- ***Library Exchange Format* (LEF)**: tcb018gbwp7t_6lm.lef;
- ***CapTable***: t018_1p6m_typical.captable.

9.2.2 Arquitetura Escolhida e Especificações

Após a definição das especificações do algoritmo ou sistema proposto, define-se a arquitetura que se deseja implementar em *hardware*.

No caso deste Trabalho de Conclusão de Curso, após a análise de diferentes arquiteturas para o sistema e topologias para o filtro FIR, decidiu-se pela implementação de um equalizador digital *least mean squares* (LMS), combinado à um filtro FIR de Forma Direta I (DF-I), conforme mostrado na Figura 35.

Informações importantes para a implementação da arquitetura do projeto são apresentadas abaixo. As decisões de topologia, ordem, tamanho das palavras e algoritmo de

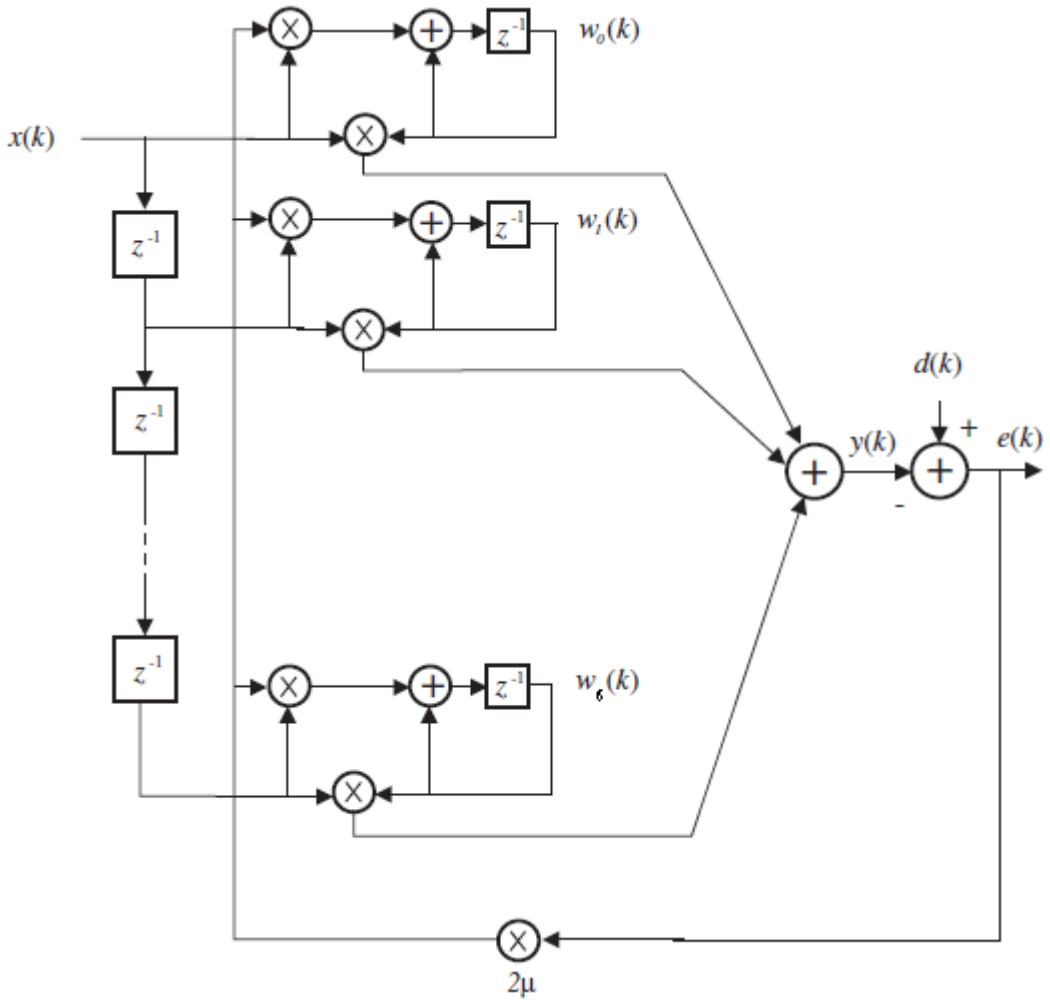

Figura 35 – Arquitetura do filtro equalizador escolhida para implementação. Essa arquitetura combina o algoritmo LMS com um filtro FIR de forma direta I.

adaptação foram feitas com base em análises apresentadas no decorrer deste documento, especialmente na parte II – Projeto e Implementação.

1. **Topologia do filtro**: FIR de forma direta I.
2. **Ordem do filtro**: Sexta ordem (6 *taps*).
3. **Algoritmo de adaptação**: *Least Mean Squares*.
4. **Tamanho da palavra de entrada**: 16 bits.
5. **Formato da palavra de entrada**: *Q3.13*.
6. **Tamanho da palavra de treino**: 24 bits (16 bits, extendido à 24 bits).
7. **Formato da palavra de treino**: *Q6.18*.
8. **Modos de operação**: 2 modos – treino e decisão direta.

9. **Tamanho da palavra de erro**: 25 bits.

10. **Formato da palavra de erro**: *Q7.18*.

11. **Valor da constante** α: 0.03125.

12. **Tamanho da palavra de saída**: 24 bits.

13. **Formato da palavra de saída**: *Q6.18*.

9.2.3 Modelagem Comportamental

Partindo das especificações, foi feito um diagrama de blocos do sistema, o qual foi mostrado na Figura 30. Além disso, os algoritmos e a arquitetura foram validados através dos códigos em MATLAB. Primeiramente, fez-se a codificação em ponto-flutuante e, então, num segundo estágio, em ponto-fixo, conforme explicado na seção que descreve o fluxo de projeto de circuitos integrados digitais. Com os algoritmos e arquitetura validados, foram gerados os vetores de teste. Os códigos são apresentados em anexo à este documento.

- Os entregáveis da etapa de modelagem comportamental são:

  1. Diagrama de blocos do sistema;
  2. Modelo comportamental do sistema em MATLAB (ponto-flutuante);
  3. Modelo comportamental do sistema em MATLAB (ponto-fixo).
  4. Vetores de teste (*inputs/outputs).*

9.2.4 Modelagem Funcional

Uma vez que os algoritmos e arquitetura do sistema foram validados no estágio de modelagem comportamental, faz-se a implementação em linguagem de descrição de *hardware*. O autor e projetista usou Verilog para fazê-lo. A simulação pré-síntese foi feita utilizando-se dos vetores de teste gerados anteriormente, no *software SimVision*.

- Os entregáveis da etapa de modelagem funcional são:

  1. Modelo funcional em Verilog do projeto;
  2. Modelo funcional do *testbench*;
  3. Validação do modelo funcional pré-síntese lógica com os vetores de teste no *software Cadence SimVision*.

9.2.5 Síntese Lógica

Tendo o estágio referente à modelagem funcional pronto, a síntese lógica foi feita com o objetivo de otimizar e mapear o código em RTL escrito em Verilog com as *standard cells* providas pela tecnologia CMOS TSMC 0.18 μm. Como mencionado na seção *Fluxo de Ferramentas Computacionais (CAD/EDA)*, no capítulo 7, o *Cadence* RTL *Compiler* foi utilizado nesta etapa. Os *scripts* de síntese lógica, bem como a *netlist* mapeada, relatórios de tempo, área e potência, podem ser encontrados no CD em anexo à este documento.

Após a síntese lógica foi feita a verificação pós-síntese lógica. Para tanto, utilizou-se o modelo funcional das *standard cells*, provido pelo DK digital da TSMC.

- Os entregáveis da etapa de síntese lógica são:

  1. *Scripts* para a síntese lógica (RTL *Compiler*);
  2. *Netlist* (arquivo .v) otimizada e mapeada na tecnologia digital TCBGBWP7TTC;
  3. Arquivo .sdc, que contém informações de tempo relacionadas à tecnologia e a *netlist*;
  4. Relatório de tempo, área e potência;
  5. Validação do modelo funcional pós-síntese lógica com os vetores de teste no *software Cadence SimVision*.

9.2.6 Síntese Física

Uma vez que a etapa de síntese lógica foi finalizada, seus entregáveis são *inputs* para o passo subsequente, que se refere à síntese física. Todo o processo de implementação física foi automatizado pelo autor e projetista deste Trabalho de Conclusão de Curso. As etapas de síntese física, seguiram os passos sugeridos por [Brunvand 2006], conforme apresentado:

1. ***Floorplan***: Definição da planta-baixa do projeto, definição dos pinos e suas localizações (*pin mapping*);
2. ***Powerplan***: Definição do esquema de alimentação do projeto, criação de *power rings* e *power stripes*;
3. ***Placement***: Colocação das *standard cells* em suas devidas linhas (*rows*);
4. ***Special Route***: Definição de um roteamento preliminar, antes da síntese da árvore de *clock* – *Clock Tree Synthesis* (CTS);
5. **Verificação pré-CTS**: Verificação do projeto antes da etapa CTS;

6. **Otimizações pré-CTS**: Otimizações do projeto antes da etapa CTS;
7. ***Clock Tree Synthesis* (CTS)**: Síntese da árvore de *clock*;
8. **Verificação pós-CTS**: Verificação do projeto depois da etapa CTS;
9. **Otimizações pós-CTS**: Otimizações do projeto depois da etapa CTS;
10. ***Fillers***: Adição de *filler cells*, para fechar os espaços;
11. ***Nano Route***: Roteamento (propriamente dito) do circuito;
12. **Otimizações pós-Route**: Otimizações do projeto depois da etapa *Nano Route*;
13. ***Metal Fill***: Adição de *layers* extra de metal, para balanceamento;
14. **Verificações de síntese lógica**: Verificações de *connectivity*, *antenna*, DRC, LVS, *metal density*, etc;
15. ***Export***: Geração dos arquivos correspondentes ao *layout*: DEF, OA e/ou GDS.

A etapa de síntese física também foi automatizada por meio de *scripts* pelo autor e projetista do Trabalho de Conclusão de Curso. Todos os *scripts* estão anexados ao documento.

- Os entregáveis referentes à este estágio são:
    1. Arquivo .DEF, correspondente ao *layout* final, exportado do EDI;
    2. Arquivos .v (HDL), correspondentes ao circuito final, com informações de atraso, capacitância, etc, para testes pós-síntese e *import* funcional no *Cadence Virtuoso*;
    3. Arquivo .pins, correspondente ao circuito final, exportado do EDI, para verificação LVS (*layout vs. schematic* com o *Cadence Assura* ou *Calibre*);
    4. Relatórios correspondentes às etapas citadas anteriormente;
    5. Sumário do projeto, *gates* mapeados, *clocks*, *I/O*s, etc.

9.2.7 Pós-síntese Física

Tendo validado a etapa de síntese física e também atingido as especificações propostas, o *layout* foi importado na ferramenta *Cadence Virtuoso*, para compor o topo do *chip* **UNB0414**.

- Os entregáveis da etapa pós-síntese física são:

1. Relatórios de verificações de DRC e LVS;
2. Arquivo GDS, correspondente ao produto final, pronto para levar à fabricação.

Parte III

Resultados e Conclusão

10 Resultados do Filtro Equalizador

Este capítulo tem como objetivo apresentar os resultados obtidos com a implementação do filtro equalizador LMS adaptativo ao canal de comunicação.

10.1 Diagrama Final do Sistema

O diagrama de blocos do sistema, foi apresentado na subseção *Descrição e Diagrama de Blocos do Sistema*, do capítulo 8 deste documento. A Figura 30 é novamente apresentada, nesta seção de resultados.

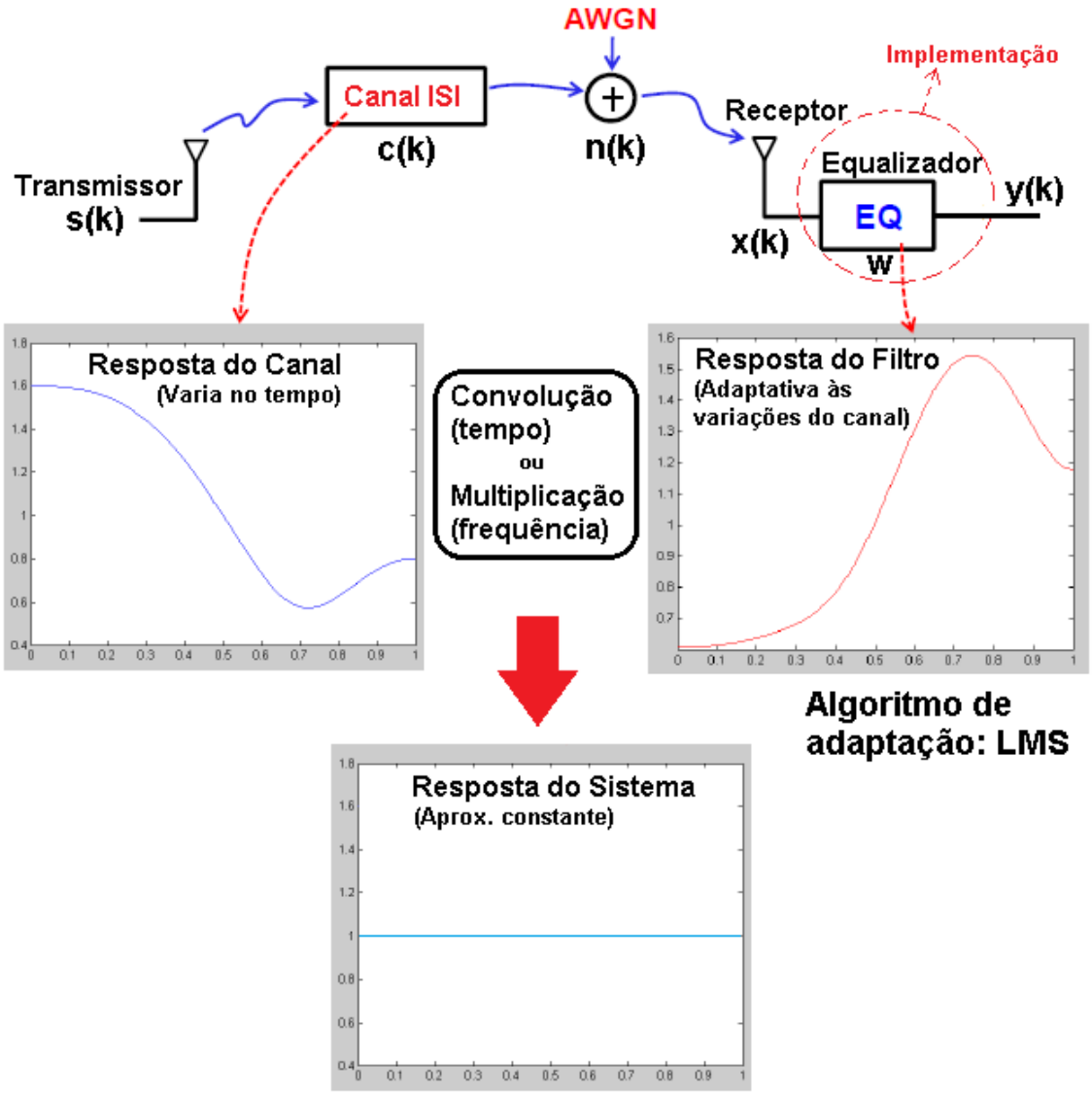

Figura 36 – Diagrama de blocos do sistema, reapresentado.

10.2 Resultados da Modelagem Comportamental

O modelo comportamental do sistema, descrito na linguagem de alto-nível MATLAB, em ponto-flutuante e ponto-fixo, conforme mencionado nas seções anteriores, é apresentado em anexo à este documento. Além deste, os códigos de modelagem comportamental das arquiteturas alternativas, descritas na subseção *Arquiteturas Alternativas Analisadas*, do capítulo 9, também estão em anexo.

10.3 Resultados da Modelagem Funcional

O modelo funcional do filtro equalizador LMS adaptativo ao canal de comunicação, descrito em Verilog HDL é apresentado em anexo à este trabalho.

A simulação pré-síntese lógica do filtro equalizador LMS adaptativo é mostrada na Figura 37. Na figura em questão, compara-se os resultados obtidos com a simulação do modelo comportamental, em ponto flutuante, proveniente do MATLAB, com o modelo funcional, descrito em linguagem de descrição de *hardware*, simulado no *software Cadence SimVision*.

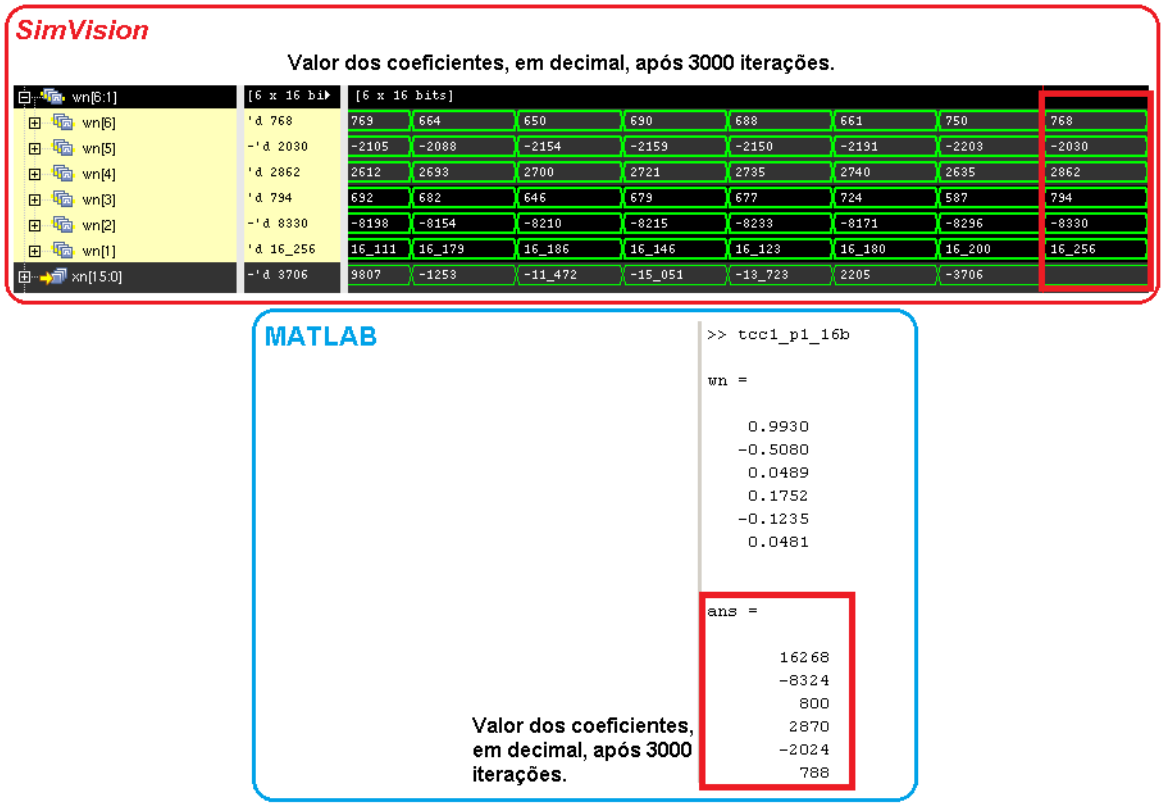

Figura 37 – Comparação dos resultados obtidos do modelo funcional, no *SimVision*, com o modelo comportamental, no MATLAB.

Pela Figura 37 nota-se que os resultados, são bastante próximos. A diferença entre

eles é devido ao erro de quantização em ponto-fixo, uma vez que, o tamanho da palavra é limitado à alguns bits, diferentemente do MATLAB. No entanto, ainda assim, o erro é pequeno. A Tabela 1 compara os resultados das duas simulações.

Tabela 1 – Comparação entre os resultados obtidos com o modelo comportamental, em MATLAB, e o modelo funcional, descrito em Verilog, no *SimVision*.

Simulação				
Coeficiente	MATLAB	*SimVision* [Q16.14]	*e(k)* [Q25.18]	[*e(k)*/MATLAB]×100
W_{n1}	16268	16256	12	0,07%
W_{n2}	-8324	-8330	6	-0,07%
W_{n3}	800	794	6	0,75%
W_{n4}	2870	2862	8	0,28%
W_{n5}	-2024	-2030	6	-0,30%
W_{n6}	788	768	20	2,54%

10.4 Resultados da Síntese Lógica

A síntese lógica foi feita com o RTL *Compiler*, de acordo com os procedimentos especificados nas seções *Síntese Lógica* dos capítulos 5 e 9.

10.4.1 Scripts de Síntese Lógica

O processo de síntese lógica foi automatizado pelo autor/projetista deste trabalho afim de padronizar e acelerar este estágio de desenvolvimento. A Tabela 2 apresenta os *scripts* e suas definições.

Tabela 2 – *Scripts* de Síntese Lógica

Scripts do RTL *Compiler*		
Arquivo	**Definição**	**Caminho**
synth_rtl_tsmc.tcl	Executa a síntese lógica	project_dir
synth_rtl_tsmc.tcl	Entregáveis deste estágio	project_dir/output_files

10.4.2 Estrutura e Esquemático da Síntese Lógica

A estrutura da síntese é mostrada na Figura 38. O esquemático do topo (*top level*) do projeto é mostrado na Figura 39.

10.4.3 Resumo dos Relatórios de Síntese Lógica

Um resumo dos relatórios de tempo (performance), área e potência do filtro equalizador é apresentado na Tabela 3. Estes relatórios são estimativas de performance, área

```
lms_direct
├─add_156_41 [add_signed_703]
├─add_157_41 [add_signed_703_977]
├─add_158_41 [add_signed_703_976]
├─add_159_41 [add_signed_703_975]
├─add_160_41 [add_signed_703_974]
├─add_161_41 [add_signed_703_973]
├─csa_tree_add_95_58_groupi [csa_tree_add_95_58_group_989]
├─mul_116_41 [mult_signed_227_1125]
├─mul_117_41 [mult_signed_227_1126]
├─mul_118_41 [mult_signed_227_1127]
├─mul_119_41 [mult_signed_227_1128]
├─mul_120_41 [mult_signed_227_1129]
├─mul_121_41 [mult_signed_227]
└─sub_106_18 [sub_signed]
```

Figura 38 – Estrutura da síntese lógica do filtro equalizador LMS adaptativo.

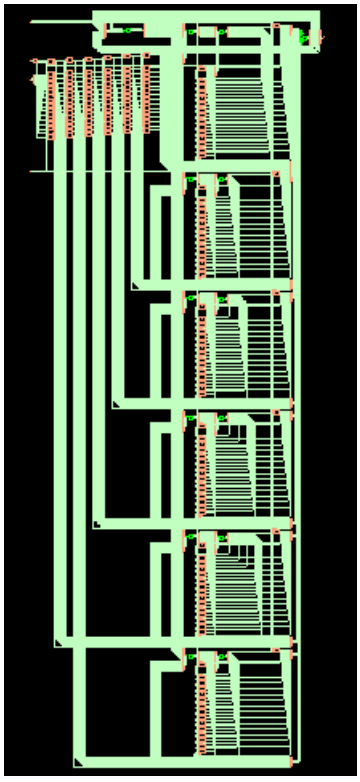

Figura 39 – Esquemático do topo do filtro equalizador LMS adaptativo.

e potência, levando em conta a tecnologia CMOS 0.18 μm, e podem ser encontrados, completos, na parte de anexos deste mesmo documento.

Tabela 3 – Relatórios de tempo, área e potência do estágio de síntese lógica.

Resultados	
Relatórios	**Período do *clock*: 20000 ps**
Tempo	5454 ps (MET)
Área	157602 μm^2
Potência	65933.27 μW

10.4.4 Simulação Pós-síntese Lógica

A Figura 40 mostra os resultados da simulação feita com a *netlist* gerada a partir do processo de síntese lógica. Para tal simulação, fez-se o *import* e compilação dos arquivos Verilog da biblioteca digital no *software SimVision*.

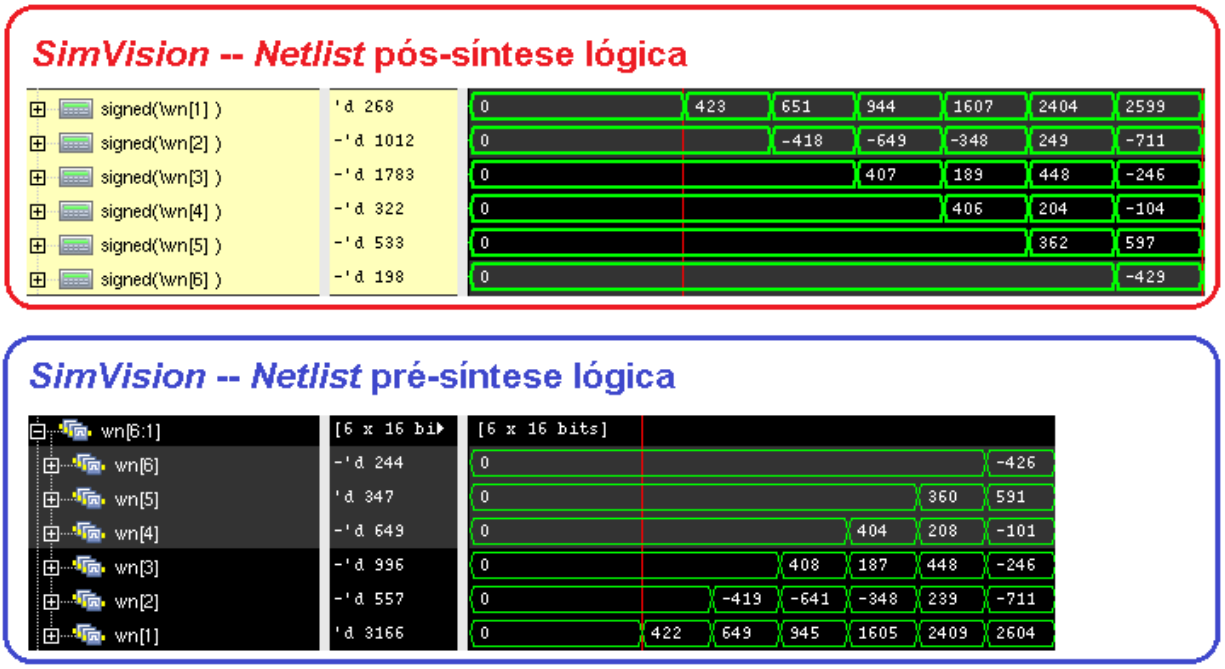

Figura 40 – Comparação entre os resultados antes e depois da síntese lógica para os coeficientes do filtro equalizador.

10.5 Resultados da Síntese Física

A síntese física foi feita no *software Cadence* RTL-*to*-GDSII, proveniente do pacote *Encounter Digital Implementation*. Os passos e procedimentos da implementação física foram explanados detalhadamente na seção *Síntese Física* do capítulo 9 – *Implementação do Filtro Equalizador*.

10.5.1 Scripts de Síntese Física

Todo o processo de implementação física foi automatizado por *scripts*. Estes são considerados resultados do Trabalho de Conclusão de Curso e são apresentados em anexo à este documento. A Tabela 4 apresenta os arquivos necessários para este estágio do projeto.

Tabela 4 – *Scripts* de Síntese Física

***Scripts* do RTL-*to*-GDSII (EDI)**		
Arquivo	**Definição**	**Caminho**
setup.sh	Configuração do ambiente	project_dir
tsmc_init_design.conf	Inicialização	project_dir/data
Default.view	Configuração do MMMC	project_dir/data
lms_direct.tcl	Script principal	project_dir/scripts
floorplan.tcl	*Floorplan*	project_dir/scripts
powerplan.tcl	*Powerplan*	project_dir/scripts
placement.tcl	*Placement*	project_dir/scripts
placement.tcl	*Special Route*	project_dir/scripts
placement.tcl	Verificação pré-CTS	project_dir/scripts
placement.tcl	Otimizações pré-CTS	project_dir/scripts
clocktree.tcl	*Clock Tree Synthesis*	project_dir/scripts
postCTS.tcl	Verificação pós-CTS	project_dir/scripts
postCTS.tcl	Otimizações pós-CTS	project_dir/scripts
fillcore.tcl	Adição de *Fillers*	project_dir/scripts
nanoroute.tcl	*Nano Route*	project_dir/scripts
nanoroute.tcl	Otimizações pós-Route	project_dir/scripts
metalfill.tcl	*Metal Fill*	project_dir/scripts
verify.tcl	Verificações de síntese lógica	project_dir/scripts
generate.tcl	*Export*	project_dir/scripts

10.5.2 Layout do Circuito Final

Executando-se de todos os passos da síntese física, apresentados no capítulo 9, utilizando-se dos *scripts* mostrados na subseção anterior, além de algumas correções manuais feitas, chegou-se aos resultados desta etapa. As Figuras 41 e 42 mostram o *layout* do circuito final, ocultando-se e mostrando-se a camada de *metal fill*, respectivamente. As implementações resultantes, ilustradas nas figuras, passaram por todas as verificações do *software* RTL-*to*-GDSII, não apresentando quaisquer violações.

10.5.3 Resumo dos Relatórios de Síntese Física

Todos os relatórios importantes para a etapa de síntese física estão em anexo à este documento. A Figura 43 ilustra parte do resumo do relatório final deste estágio. Nota-se que a área do *design* é de aproximadamente 0.25 mm^2 ($\approx 500{\times}500\ \mu m^2$).

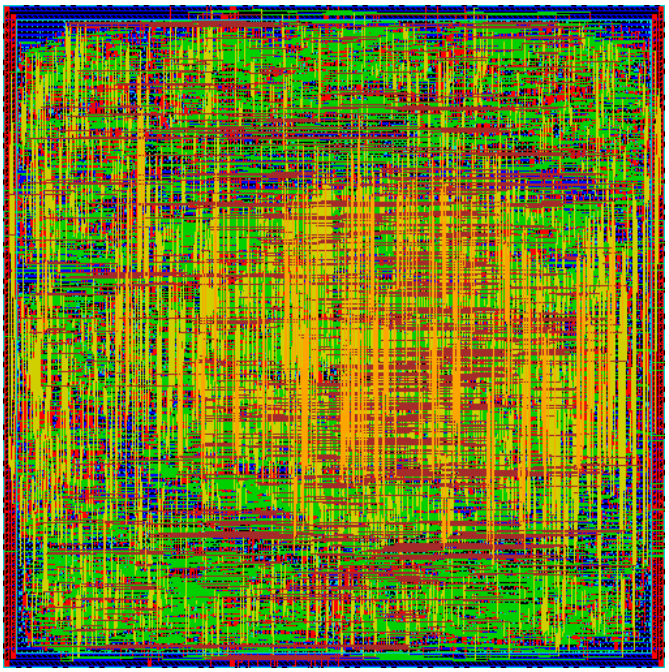

Figura 41 – *Layout* do circuito final (ocultando-se a camada de *metal fill*).

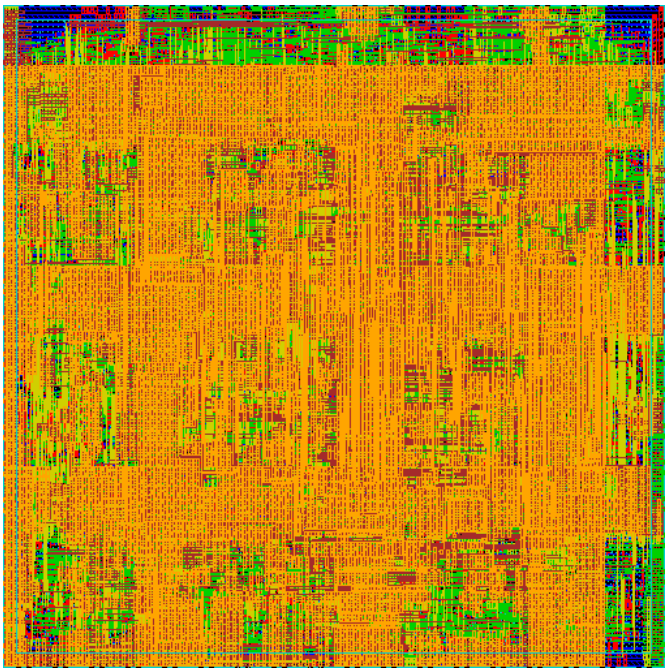

Figura 42 – *Layout* do circuito final (com a camada de *metal fill*).

```
==============================
Floorplan/Placement Information
==============================
Total area of Standard cells: 224979.462 um^2
Total area of Standard cells(Subtracting Physical Cells): 159549.331 um^2
Total area of Macros: 0.000 um^2
Total area of Blockages: 0.000 um^2
Total area of Pad cells: 0.000 um^2
Total area of Core: 225147.846 um^2
Total area of Chip: 244606.872 um^2
Effective Utilization: 1.0168e+00
Number of Cell Rows: 121
```

Figura 43 – Parte do resumo do relatório final de síntese física. A área do *chip* é de aproximadamente 0.25 mm^2, em tecnologia CMOS 0.18 μm.

10.6 Resultados Pós-síntese Física

Os procedimentos pós-síntese lógica foram apresentados na subseção *Síntese Lógica*, do capítulo 9. Os resultados são apresentados nesta seção.

10.6.1 Export do RTL-to-GDSII e Import no Virtuoso

Foi feito a exportação do circuito completo e finalizado do *software Cadence* RTL-*to*-GDSII para o *software Cadence Virtuoso*. Para tanto, utilizou-se o formato DEF 5.4, conforme descrito no capítulo 9, relativo à implementação do projeto. A Figura 44 ilustra o bloco do filtro equalizador digital LMS adaptativo, importado no *Cadence Virtuoso*.

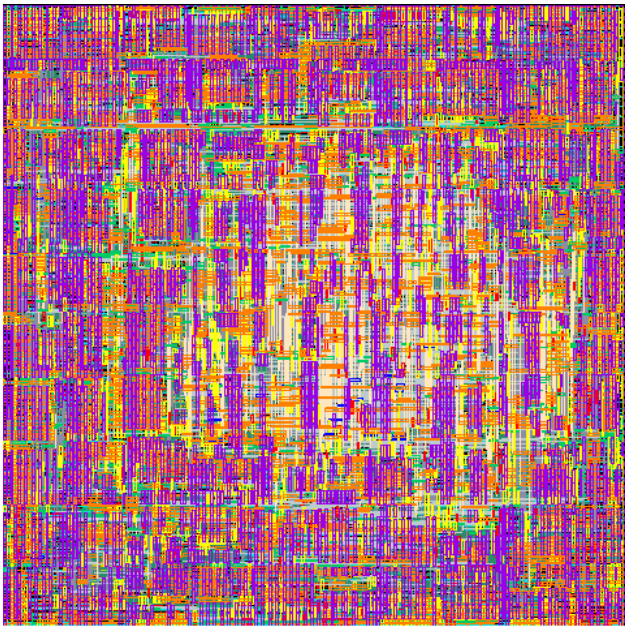

Figura 44 – *Layout* do circuito final importado no *Cadence Virtuoso*.

Além do *layout*, pode-se também fazer o *import* dos esquemáticos e símbolo referentes ao filtro equalizador LMS adaptativo, com os arquivos *.v* e *.pins*, gerados no estágio de síntese física, pelo *Cadence* RTL-*to*-GDSII. Com as novas *views*, pode-se fazer

a verificação *Layout Versus Schematic* (LVS) e testes deste bloco digital fazendo parte de um sistema maior, com outros blocos. Um exemplo disso, seria o *testbench* do conversor analógico-digital, em Verilog-AMS, mostrado no Apêndice A, provendo o sinal de entrada, digital, de 16 bits, para o filtro equalizador LMS adaptativo. As Figuras 45 e 46 ilustram os símbolos do projeto, no *Cadence Virtuoso*, com as entradas e saídas em paralelo e em serial. A versão serial foi feita devido às poucas interfaces de *input/output* disponíveis no *chip* **UNB0414**.

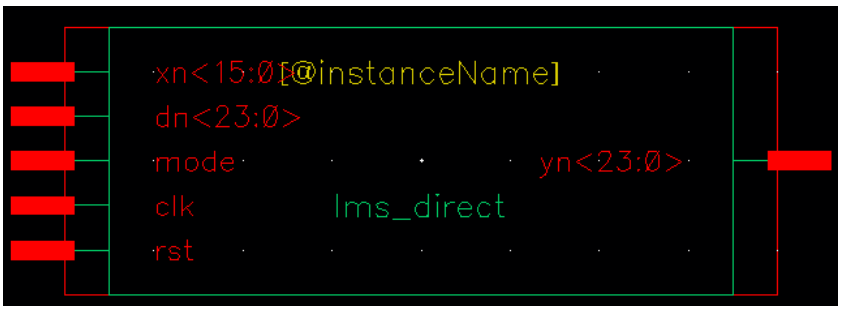

Figura 45 – Símbolo do filtro equalizador LMS adaptativo no *Cadence Virtuoso* (versão com I/Os em formato paralelo).

Figura 46 – Símbolo do filtro equalizador LMS adaptativo no *Cadence Virtuoso* (versão com I/Os em formato serial).

11 Resultados do Chip UNB0414

Este capítulo tem como objetivo apresentar os resultados obtidos com a implementação do *chip* **UNB0414**. Este *chip* é composto por diferentes projetos, entre eles, os blocos digitais do equalizador LMS adaptativo, explanado nos capítulos anteriores, e do decimador de terceira ordem para moduladores $\Sigma\Delta$, apresentado no Apêndice B deste documento.

11.1 Resultado Final do UNB0414

As Figuras 47 e 48 mostram o topo do *chip* **UNB0414**, com os blocos do filtro LMS equalizador adaptativo, mostrado no capítulo *Resultados do Filtro Equalizador*, e do decimador de terceira ordem, mostrado no capítulo *Resultados do Filtro Decimador*, além de outros projetos que também foram levados à fabricação. Verificações de DRC e LVS nos circuitos foram feitas com sucesso com o *software Cadence Assura*.

As figuras mostram o *chip* pronto para exportação do arquivo GDSII e fabricação.

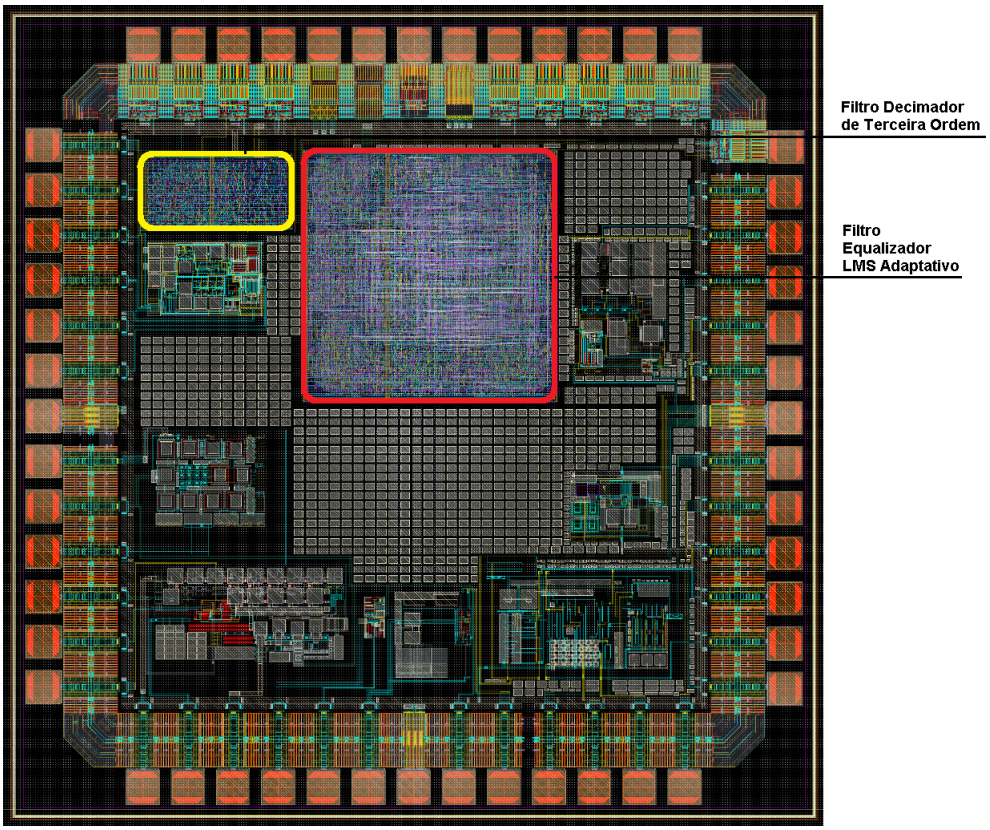

Figura 47 – Topo do *chip* **UNB0414** no *Cadence Virtuoso* (parte 1).

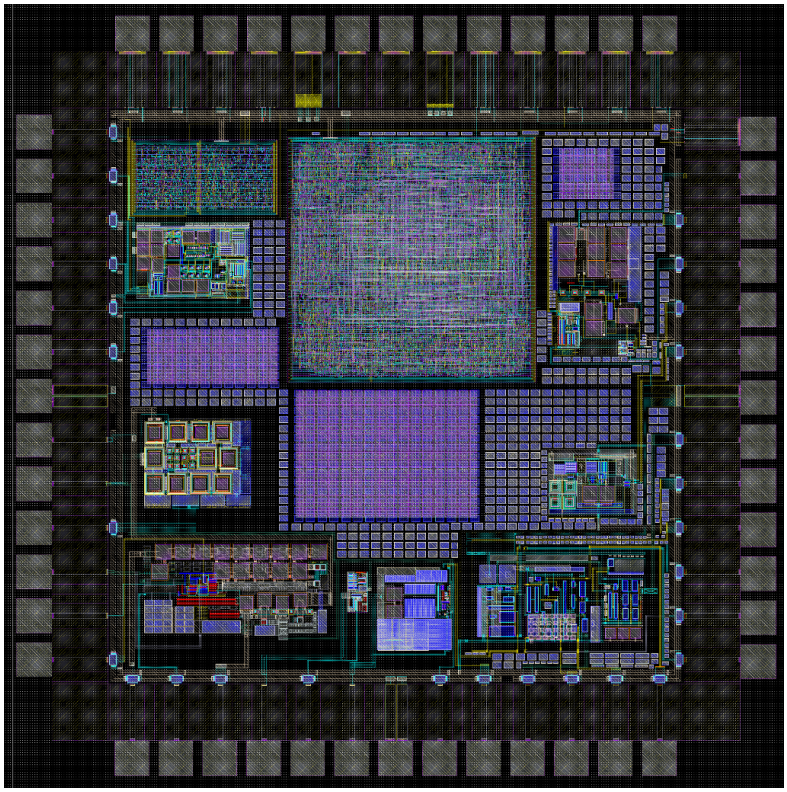

Figura 48 – Topo do *chip* **UNB0414** no *Cadence Virtuoso* (parte 2).

12 Conclusão

Um filtro equalizador *least mean squares* adaptativo ao canal de comunicação e um filtro decimador de terceira ordem para moduladores $\Sigma\Delta$ foram projetados, implementados e mandados para a fabricação em tecnologia CMOS 0.18 μm.

O filtro equalizador em questão é implementado pela combinação de um filtro FIR de forma direta (DF-I) com a circuitaria referente ao algoritmo LMS. Este dispositivo, pode ser usado em canais de comunicação não-ideais para a atenuação ou tratamento do problema de propagação multicaminho. Uma boa utilização seria em conjunto com circuitos que executam algoritmos de correção de erro de bit e tratamento de ruído.

Para o projeto do equalizador, foi gerado, primeiramente, um modelo comportamental do sistema, em MATLAB, pelo qual foi possível entender melhor o funcionamento do algoritmo LMS, bem como validar a solução proposta. A partir deste modelo, foi feita a implementação funcional, síntese lógica e síntese física. Cada etapa do processo foi validada de acordo com os métodos descritos nos capítulos 5 e 9, nas subseções referentes ao fluxo de projeto de circuitos integrados digitais.

Considerando-se a etapa de modelagem comportamental, com a descrição do sistema em MATLAB, observa-se que a variação de parâmetros, tais como os valores escolhidos para a constante de convergência, ordem e quantização em ponto-fixo, do filtro equalizador LMS adaptativo, têm efeitos diretos na resposta do sistema. Ao manipular-se estas variáveis, buscou-se um conjunto de parâmetros, em particular, que fornecessem uma resposta adequada para as especificações do filtro. Foi observado, que, aumentando-se a ordem do filtro para valores muito acima de 5, o custo-benefício entre a estimativa de erro e o *hardware* necessário, seria muito alto. Dessa forma, decidiu-se por implementar um filtro de ordem 6. Por outro lado, no caso da constante de convergência, houve a necessidade de testar uma faixa de valores para a quais, a estimativa do erro era conveniente para o projeto. Portanto, optou-se por $\alpha = 0.03125$, que pode ser feito, à nível de *hardware*, sem a necessidade de multiplicadores ou somadores adicionais.

Com relação ao modelo funcional, em Verilog, foi provado, através de testes com o modelo comportamental, em ponto-flutuante, que os valores dos coeficientes, após um certo número de iterações, a variar de acordo com as características da função de transferência do canal e do tipo de sinal aplicado à entrada do filtro, convergem para os seus valores *ótimos*, ou seja, os valores que, iterativamente, foram calculados para a correção das distorções/dispersões impostas pelo meio físico.

As etapas de síntese lógica e física fizeram a implementação real do circuito, visando a fabricação através do *chip* **UNB0414**. O autor e projetista deste trabalho considera

estes estágios os mais importantes pelo conhecimento e prática obtidos.

Referindo-se à tecnologia empregada na fabricação do *chip* **UNB0414** e dos blocos digitais apresentados neste documento, observou-se uma grande diferença entre a tecnologia antes utilizada nos laboratórios da Universidade de Brasília, a AMS 0.35 μm. Esta última, possui a vantagem de ser um PDK *mixed-signal*, integrado e didático, no entanto, peca em relação às poucas *standard cells* disponíveis em relação ao PDK atual. Já a tecnologia TSMC 0.18 μm, mostrou-se muito mais profissional, com muito mais possibilidades, entretanto, é também mais complicada e sua instalação não é integrada, como a tecnologia 0.35 μm. Portanto, o autor e projetista deste trabalho recomenda o uso da tecnologia CMOS 0.35 μm, para iniciantes no mundo da microeletrônica, e a tecnologia CMOS 0.18 μm, da TSMC, para usuários um pouco mais avançados.

Com relação aos resultados, aponta-se a validação de todas as etapas do fluxo de projeto de circuitos integrados digitais, descrito nos capítulos 5 e 9, para o sistema implementado. Durante esse processo, observa-se que os valores obtidos para os coeficientes do filtro no modelo funcional, em Verilog, e no modelo comportamental, em MATLAB, são muito próximos (quando convertidos do formato decimal inteiro para decimal fracionário, são nota-se que são quase iguais). Usando $\alpha = 0.03125$, requer-se na faixa de 1000 à 1500 iterações até que se encontre os "coeficientes ótimos", como descrito no capítulo 3.

Dentre as implementações extras, mostradas nos apêndices deste trabalho, vale comentar sobre o decimador de terceira ordem para moduladores $\Sigma\Delta$. Este, apresentou resultados condizentes com os sinais, modulados em PDM, aplicados à sua entrada (*e.g.*, escalão, rampa ascendente e descendente e pulso). Este tipo de implementação é essencial em conversores analógico-digitais $\Sigma\Delta$. O autor e projetista deste trabalho almeja, no futuro, após a chegada dos protótipos do *chip* **UNB0414**, fazer os testes com os blocos digitais integrados, isto é, sinais analógicos sendo transformados em digitais pelo conversor AD e servindo de entradas para o filtro equalizador LMS adaptativo.

Por fim, vale a reflexão de todo conhecimento que está envolvido no projeto de um *chip*, do nível mais baixo ao ponto de levá-lo para manufatura. Todo o processo foi realidade, durante o desenvolvimento deste TCC, e é documentado através desta monografia.

Trabalhos Futuros. Almeja-se, após a defesa do Trabalho de Conclusão de Curso 2, a continuação do projeto. Para tanto, será feita a extensão do modelo do filtro equalizador para tratamento do problema de *feedback* acústico em aparelhos auditivos, mostrado na seção *Motivações*, do capítulo 1. Além disso, visar-se-á especificações mais agressivas de área e potência em implementações futuras.

Referências

BRUNVAND, E. In: *Crafting a Chip: A Practical Guide to the UofU VLSI CAD Flow*. School of Computing, University of Utah, United States of America: [s.n.], 2006. Citado na página 83.

CILETTI, M. D. In: *Advanced Digital Design with the Verilog HDL*. Prentice Hall of India Private Limited, New Delhi: [s.n.], 2005. Citado 8 vezes nas páginas 15, 55, 56, 57, 63, 64, 65 e 66.

DINIZ, P. S. R. Adaptive filtering. In: *Springer Science+Business Media*. [S.l.: s.n.], 2008. p. 77–185. Citado 5 vezes nas páginas 15, 41, 42, 43 e 44.

HAYKIN, S. In: *Communication Systems*. United States of America: [s.n.], 2001. Citado 3 vezes nas páginas 15, 37 e 38.

KHAN, S. A. In: *Digital Design of Signal Processing Systems – A Practical Approach*. Wiley, Noida, India: [s.n.], 2012. Citado 4 vezes nas páginas 15, 70, 71 e 72.

KIM, H.; ROY, K. Ultra-low power dlms adaptive filter for hearing aid applications. In: . IEEE Transactions on Very Large Scale Integration (VLSI) Systems: [s.n.], 2007. v. 11. Citado na página 33.

MADHOW, U. In: *Fundamentals of Digital Communication*. Cambridge University Press, United States of America: [s.n.], 2008. Citado 3 vezes nas páginas 16, 74 e 75.

MALIK, G.; SAPPAL, A. S. Adaptive equalization algorithms: An overview. In: *(IJACSA) International Journal of Advanced Computer Science and Applications*. [S.l.: s.n.], 2011. v. 2, n. 3. Citado na página 30.

MCCLANING, K. In: *Wireless Receiver Design for Digital Communications*. Raleigh, NC, United States of America: [s.n.], 2012. Citado 4 vezes nas páginas 15, 38, 39 e 40.

MEKALA, H. In: *Third Order CMOS Decimator Design for Sigma-Delta Modulators*. A Thesis Submitted to the Graduate Faculty of the Louisiana State University and Agricultural and Mechanical College, Louisiana, United States of America: [s.n.], 2009. Citado 2 vezes nas páginas 112 e 113.

MEKECHUK, K. et al. Linearizing power amplifiers using digital predistortion, eda tools and test hardware. In: *High Frequency Electronics*. [S.l.: s.n.], 2004. p. 18–27. Citado 2 vezes nas páginas 15 e 34.

OKAWARA, H. In: *DSP-Based Testing – Fundamentals 51: CIC Decimation and FIR Filter*. Tokyo, Japan: [s.n.], 2013. Citado 2 vezes nas páginas 113 e 114.

OPPENHEIM, A. V.; SCHAFER, R. W. In: *Digital Signal Processing*. Prentice Hall, United States of America: [s.n.], 2009. Citado na página 56.

PALNITKAR, S. In: *Verilog HDL – A Guide to Digital Design and Synthesis*. United States of America: [s.n.], 2003. Citado 4 vezes nas páginas 15, 48, 49 e 50.

QURESHI, S. U. H. Adaptive equalization. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1985. v. 73, n. 9, p. 1340–1387. Citado na página 29.

RAMACHANDRAN, S. In: *Digital VLSI Systems Design*. Indian Institute of Technology Madras, India: [s.n.], 2007. Citado 5 vezes nas páginas 15, 50, 51, 52 e 53.

RAZAVI, B. In: *Prentice Hall Communications Engineering and Emerging Technologies Series*. United States of America: [s.n.], 1998. Citado na página 33.

SPRIET, A. et al. Adaptive feedback cancellation in hearing aids. In: . Journal of the Franklin Institute: [s.n.], 2006. v. 343, p. 545–573. Citado na página 32.

THOMAS, E. D.; MOORBY, P. R. In: *The Verilog Hardware Description Language*. United States of America: [s.n.], 2002. Citado 4 vezes nas páginas 47, 48, 49 e 50.

WANHAMMAR, L. In: *DSP Integrated Circuits*. Linköping, Sweden: [s.n.], 1999. Citado na página 61.

WESTE, N. H. E.; HARRIS, D. In: *CMOS VLSI Design – A Circuits and Systems Perspective*. United States of America: [s.n.], 2004. Citado na página 47.

Apêndices

APÊNDICE A – Implementação de um Conversor AD (ADC) de 16 bits em Verilog-AMS

Além do filtro equalizador LMS adaptativo ao canal de comunicação, foram feitas algumas implementações extras: (1) um conversor analógico-digital de 16 bits em Verilog-AMS e (2) um decimador de terceira ordem, para compor um conversor analógico-digital sigma-delta.

Apesar de não fazerem parte da ideia inicial do Trabalho de Conclusão de Curso, os dois projetos em questão, são de grande importância. Conversores analógico-digitais fazem-se essenciais para a interface entre o mundo analógico e digital. O filtro equalizador LMS, por exemplo, trabalha em tempo discreto, no domínio digital. Para testá-lo, ou até mesmo em uma aplicação real, faz-se necessário o uso desse tipo de dispositivo.

O primeiro projeto, referenciado como (1), faz uma implementação em Verilog-AMS, que, embora não seja sintetizável, é bastante utilizado para testes, e, conforme sugerido pela banca, é um tópico de interesse. Este capítulo descreve esta implementação.

- **OBSERVAÇÃO**: O projeto da parte digital (filtro decimador de terceira ordem) para um conversor sigma-delta é discutido no Apêndice B – *Implementação de um Decimador de Terceira Ordem para Moduladores* $\Sigma\Delta$.

A.1 Introdução

Esta seção descreve o projeto de um conversor analógico-digital de 16 bits em Verilog-AMS. Esta implementação foi feita pelo autor e projetista do Trabalho de Conclusão de Curso para testar o filtro equalizador LMS adaptativo ao canal de comunicação no *Cadence Virtuoso*, no estágio pós-síntese física.

A.1.1 Visão Geral da Linguagem de Programação Verilog-AMS

A linguagem de descrição de *hardware* Verilog-AMS define uma linguagem de programação comportamental para sistemas analógicos e *mixed-signal*. Ela é derivada da linguagem de programação Verilog (subconjunto puramente digital) e Verilog-A (subconjunto puramente analógico).

O objetivo de Verilog-AMS é permitir que seus usuários projetem sistemas analógicos e/ou *mixed-signal*, que encapsulam descrições comportamentais e estruturais de alto-nível de seus componentes. O comportamento de cada módulo pode ser descrito matematicamente em termos de seus terminais ou parâmetros externos aplicados ao módulo. A estrutura de cada componente pode ser descrita em termos de sub-componentes interconectados.

A.2 Realização do ADC

Para realização do conversor analógico-digital de 16 bits, utilizou-se primeiramente de um modelo de interfaces *mixed-signal* já disponível no *Cadence ModelWriter*. Uma vez que definiu-se a interface, customizou-se o modelo, conforme as especificações esperadas para um ADC, que seria utilizado em conjunto com o filtro equalizador LMS adaptativo. Ao fazer a adaptação do modelo, buscou-se uma codificação que descrevia, mais acuradamente possível, o dispositivo, ainda que ideal.

- Os passos para realização do conversor AD de 16 bits foram os seguintes:

  1. Execução do *Cadence Virtuoso*;
  2. Criação de uma nova biblioteca – anexando a biblioteca *ahdlLib*;
  3. Criação de uma nova *Cell View*, especificando a biblioteca do passo anterior, a ferramenta *ModelWriter* e o *View Name* "**veriloga**"ou "**verilogams**";
  4. Seleção da interface *Mixed-signal* » *Analog-to-Digital Converter*;
  5. Especificação dos parâmetros desejados:
     - Número de bits do ADC: 16;
     - Tempo de subida para sinais digitais: 4 μs;
     - Tempo de descida para sinais digitais: 4 μs;
     - Tensão máxima na entrada do ADC: 5 V;
     - Tensão mínima na entrada do ADC: -5 V;
     - Tensão correspondente ao nível lógico 1 do ADC: 5 V;
     - Tensão correspondente ao nível lógico 0 do ADC: 0 V.
  6. Customização (adaptação) do código ao projeto do equalizador LMS adaptativo ao canal de comunicação;
  7. Criação do símbolo (*symbol*) referente ao ADC;
  8. Execução de testes com a ferramenta ADE (usando o simulador *spectre*) do *Cadence Virtuoso*, através de um novo *testbench*;
  9. Validação do projeto do ADC de 16 bits;

10. Interfaceamento com o projeto do filtro equalizador LMS adaptativo, descrito no capítulo 9, *Implementação do Filtro Equalizador*.

A.3 Resultados do ADC

As Figuras 49 e 50 ilustram, respectivamente, o *testbench* e a simulação do conversor analógico-digital de 16 bits projetado em Verilog-AMS, no ambiente do *software Cadence Virtuoso*.

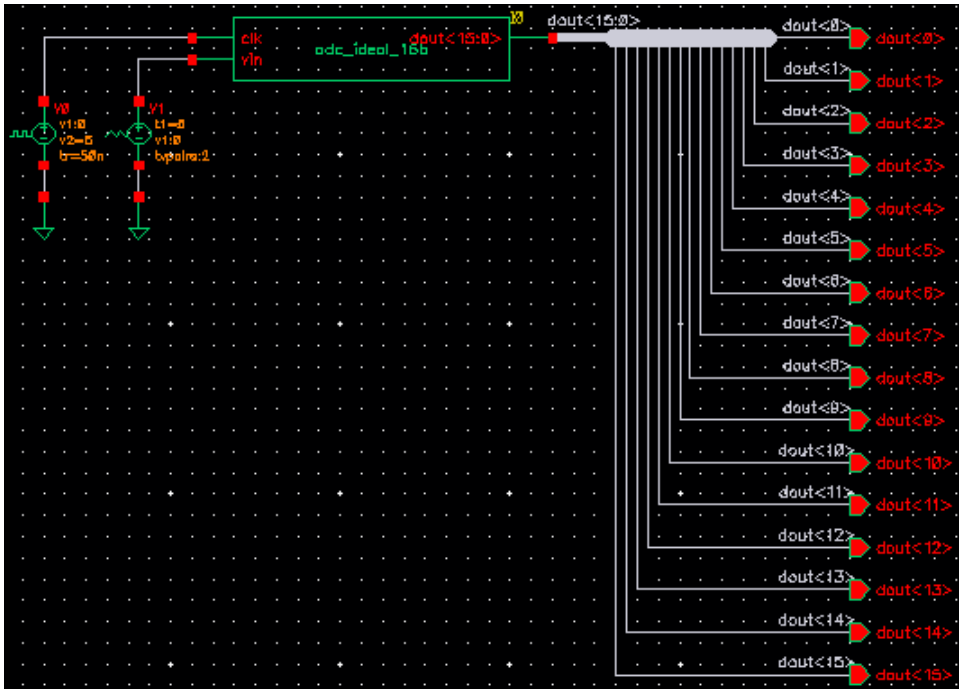

Figura 49 – *Testbench* do ADC de 16 bits projetado em Verilog-AMS.

Uma vez que o funcionamento do conversor AD fosse validado, seriam feitos testes, juntamente com o modelo funcional, instanciado no *Virtuoso*, do filtro equalizador LMS adaptativo. No entanto, por ser um *design* com muitas instâncias de *standard cells* e elementos mais complexos (*e.g.*, somadores, multiplicadores, etc), sua simulação exige bastante esforço computacional. Por esse motivo, além de outros problemas, esses testes não foram viáveis de ser feitos em tempo.

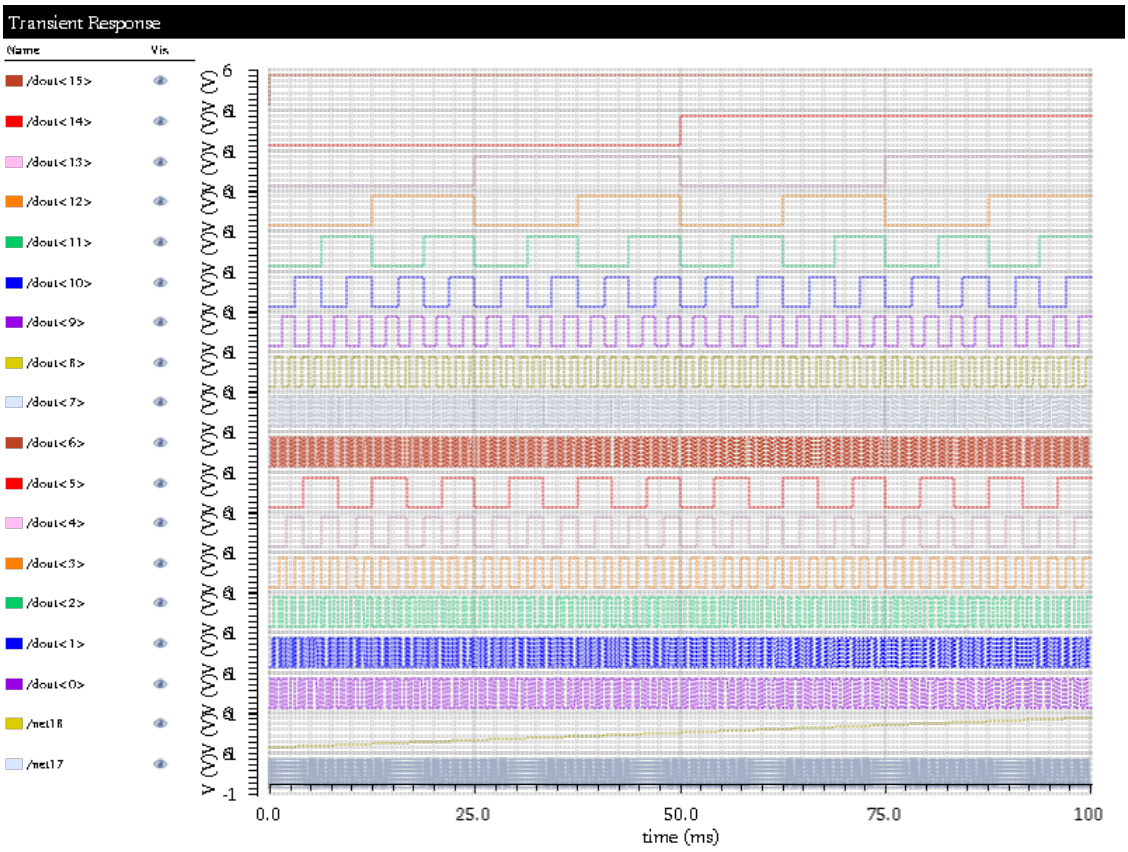

Figura 50 – Simulação do ADC de 16 bits projetado em Verilog-AMS.

APÊNDICE B – Implementação de um Decimador de Terceira Ordem para Moduladores ΣΔ

Além do filtro equalizador LMS adaptativo ao canal de comunicação, foram feitas algumas implementações extras: (1) um conversor analógico-digital de 16 bits em Verilog-AMS e (2) um decimador de terceira ordem, para compor um conversor analógico-digital sigma-delta.

Apesar de não fazerem parte da ideia inicial do Trabalho de Conclusão de Curso, os dois projetos em questão, são de grande importância. Conversores analógico-digitais fazem-se essenciais para a interface entre o mundo analógico e digital. O filtro equalizador LMS, por exemplo, trabalha em tempo discreto, no domínio digital. Para testá-lo, ou até mesmo em uma aplicação real, faz-se necessário o uso desse tipo de dispositivo.

O segundo projeto extra, (2), implementa a parte digital (filtro *cascaded integrator-comb* do tipo decimador) de um conversor analógico-digital sigma-delta de primeira ordem, também levado à fabricação, compondo o *chip* **UNB0414**. Este capítulo tem como objetivo descrever esta implementação.

- **OBSERVAÇÃO**: O projeto do conversor analógico-digital de 16 bits em Verilog-AMS é discutido no Apêndice A – *Implementação de um Conversor AD (ADC) de 16 bits em Verilog-AMS*.

B.1 Introdução

Esta seção descreve o projeto de um decimador de terceira em tecnologia CMOS 0.18 μm, para moduladores ΣΔ. Esta implementação foi feita pelo autor e projetista do Trabalho de Conclusão de Curso como incentivo ao *design* de conversores AD para interfaceamento de sinais analógicos com o domínio digital, necessário para o filtro equalizador LMS adaptativo.

B.1.1 Especificações

O projeto em questão leva em conta que a saída do modulador ΣΔ é um *stream* de bits em alta frequência, modulados por densidade de pulso (*i.e., pulse density modulation*). Alguns pontos importantes são citados abaixo:

- Constante de decimação do filtro (K): 64.

- Utilização do menor número de entradas e saídas que fosse possível, uma vez que não havia um número bastante limitado de *pads* para fabricação.

B.1.2 Conversores Analógico-Digital ΣΔ

O conversor analógico-digital ΣΔ trabalha nos princípios da modulação sigma-delta. A Figura 51 mostra um diagrama de blocos, genérico, do conversor AD ΣΔ.

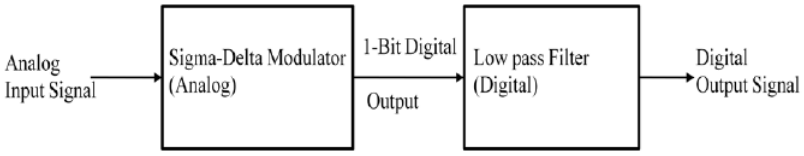

Figura 51 – Diagrama de blocos do conversor AD sigma-delta.

***Pulse-density Modulation* (PDM)**. Este tipo de modulação é um método para codificar sinais analógicos em informação digital. Em um sinal PDM, valores específicos de amplitude não são codificados em pulsos de diferentes tamanhos, como na *pulse-code modulation* (PCM). Em vez disso, é a densidade relativa de pulsos que corresponde-se com a amplitude do sinal analógico [Mekala 2009].

Um exemplo de um sinal em PDM correspondente com a saída de um modulador ΣΔ é mostrado na Figura 52.

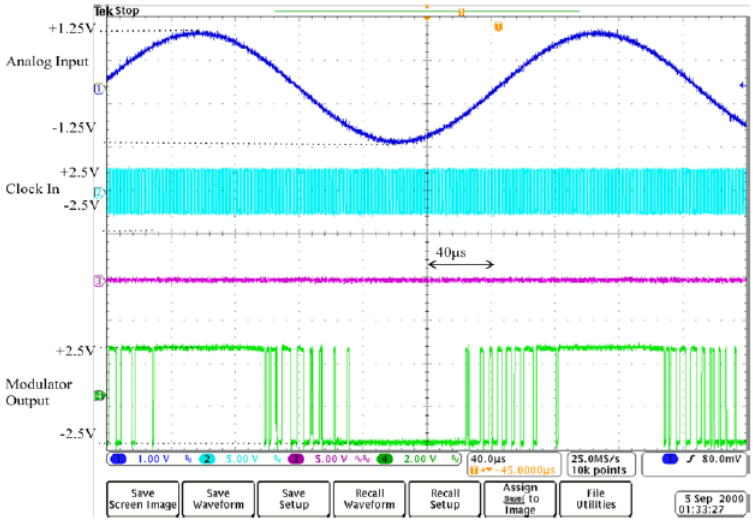

Figura 52 – Saída do modulador sigma-delta (*pulse-density modulation*).

B.1.3 Decimadores: Visão Geral

Decimação é uma técnica usada para reduzir o número de amostras em um sinal de frequência mais alta em tempo-discreto. O processo de decimação é usado um conversores $\Sigma\Delta$ para eliminar informação redundante, introduzida pelo *oversampling*. Na prática, isto normalmente implica na convolução de um sinal por um filtro passa-baixas (do tipo CIC), e então, a redução de sua taxa efetiva de amostragem. O filtro decimador acumula um *stream* de bits por um certo tempo e divide a soma pelo número de períodos [Mekala 2009].

O diagrama de blocos geral de um decimador é mostrado na Figura 53, abaixo.

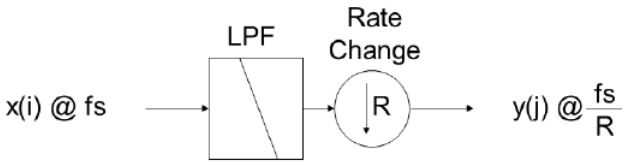

Figura 53 – Diagrama de blocos geral de um decimador.

B.1.4 Filtros CIC (Cascaded Integrator-Comb)

Em processamento digital de sinais, filtros CIC são classes otimizadas de filtros de resposta finita ao impulso (*i.e., finite impulso response*), utilizados em aplicações de interpolação e decimação. A Figura 54 apresenta as topologias e equações dos filtros *comb*, integrador e CIC [Okawara 2013].

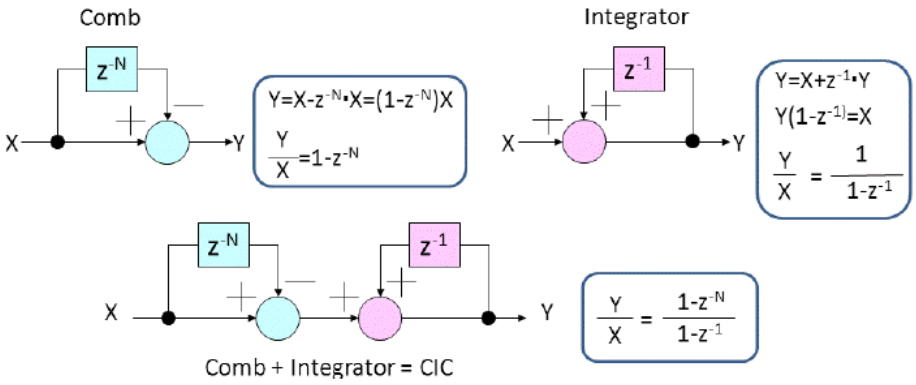

Figura 54 – Topologias e equações dos filtros *comb*, integrador e CIC.

Para melhorar a característica de um passa-baixas, estágios múltiplos de um único CIC podem ser cascateados, como mostrado na Figura 55. Vale salientar que todos as topologias apresentadas são equivalentes [Okawara 2013].

Uma implementação em *hardware*, mais interessante, do ponto de vista de projeto é mostrada na Figura 56. Essa topologia permite que os blocos *comb* trabalhem na frequên-

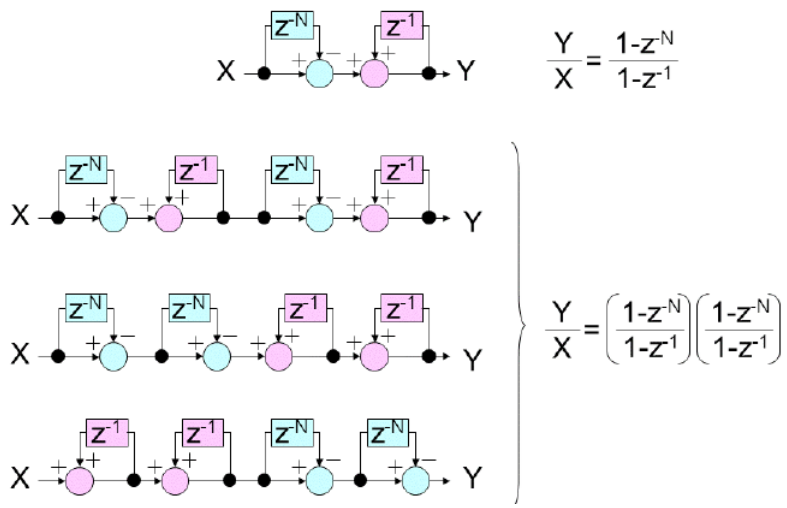

Figura 55 – Filtros CIC de múltiplos estágios, equivalentes.

cia decimada em vez da frequência de original de amostragem. Dessa forma, consumo e área são diminuídos [Okawara 2013].

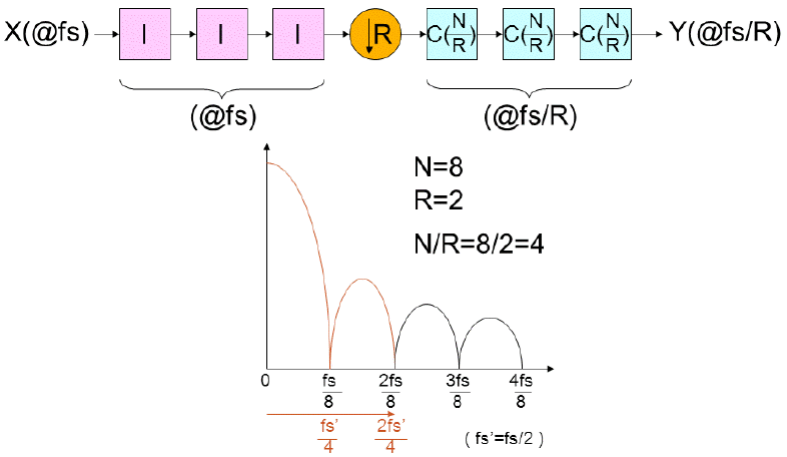

Figura 56 – Uma implementação do *hardware* do filtro CIC mais interessante.

B.2 Realização do Filtro Decimador

A realização do filtro decimador para modulares ΣΔ seguiu o fluxo de circuitos integrados digitais mostrado na seção *Fluxo de Projeto VLSI*, do capítulo 5. Neste caso, em particular, não houve a realização dos testes pós-síntese lógica e pós-síntese física.

Como o fluxo de implementação é o mesmo mostrado para o filtro equalizador LMS adaptativo, descrito na seção *Realização do Filtro Equalizador* do capítulo 9, decidiu-se por não incluí-lo novamente, neste apêndice. Entretanto, o autor explica sobre a arquitetura escolhida e o modelo comportamental feito no *software* MATLAB.

B.2.1 Arquitetura Escolhida

A arquitetura escolhida para implementação é mostrada na Figura 57. Dessa forma, optou-se por um filtro *Cascaded Integrator-Comb* de 3 estágios, com 20-bits (complemento de 2) em sua saída.

- ***Coder Circuit***. Esse bloco faz a extensão do sinal binário proveniente da saída do modulador $\Sigma\Delta$. O circuito faz a extensão para 20-bits, conforme a regra:

  1 – (representação em 20-bits) – ...00000001 – (+1 em complemento de 2)

  0 – (representação em 20-bits) – ...11111111 – (-1 em complemento de 2)

- **Filtro Integrador**. Consiste em três estágios de integradores cascateados. Os somadores mostrados na Figura 57, nos estágios de integração, são acumuladores.

- **Filtro *Comb***. Consiste em três estágios de diferenciadores (ou filtros *comb*) cascateados. A entrada deste estágio são os 20 bits de saída do estágio de integração e o sinal de *clock* decimado.

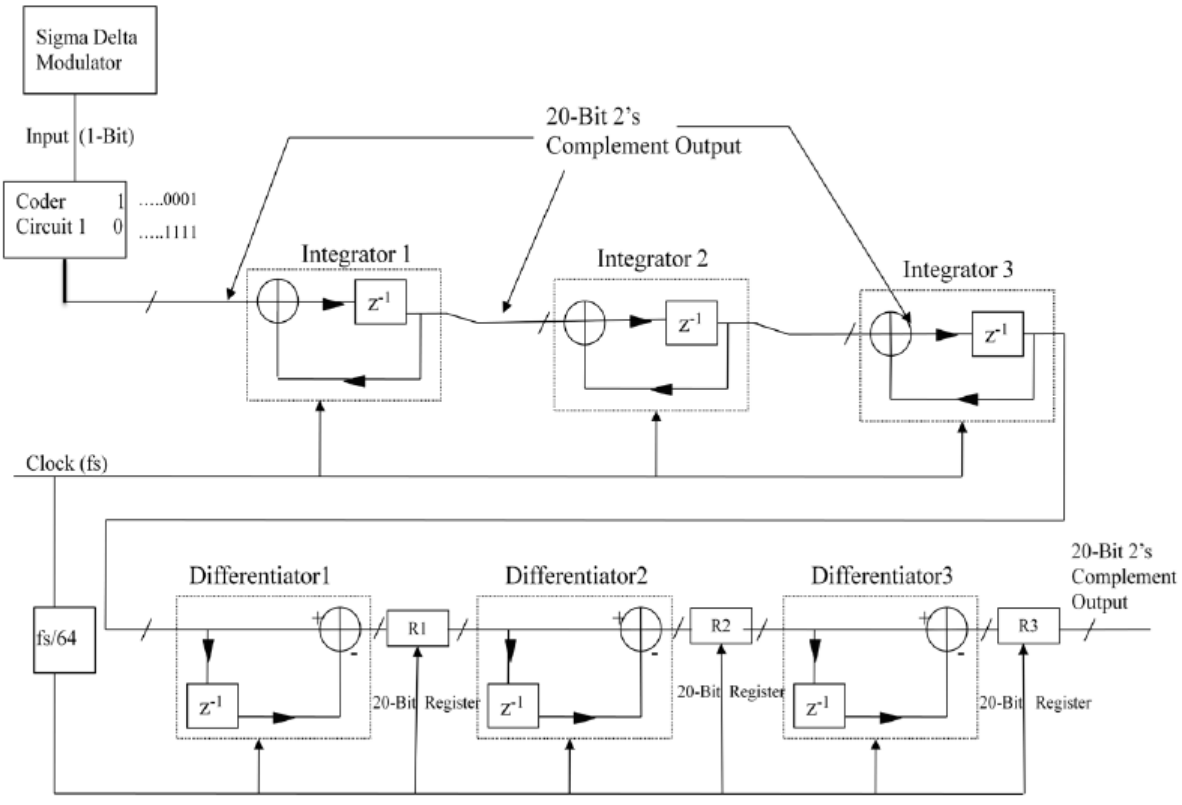

Figura 57 – Arquitetura escolhida para implementação do CIC.

B.2.2 Modelo Comportamental

Algumas características também podem ser obtidas com a função mfilt.cicdecim(K, M, N), onde K é o *oversample*, M é o atraso diferencial e N é a ordem do filtro. A Figura 58 mostra a resposta em magnitude do filtro decimador para $K = 64$, $M = 1$ e $N = 3$.

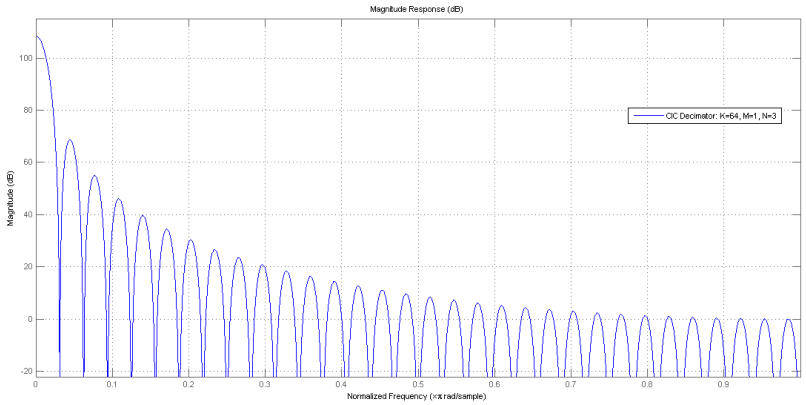

Figura 58 – Resposta em magnitude do filtro decimador.

Já a Figura 59 ilustra a resposta ao impulso do filtro decimador com $K = 64$, $M = 1$ e $N = 3$. Todos os códigos usados para gerar os resultados mostrados das Figuras 58 e 59 estão anexados à este documento.

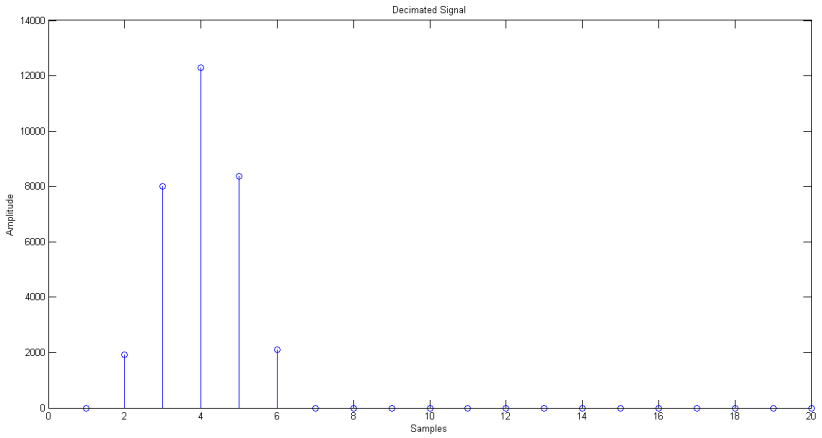

Figura 59 – Resposta ao impulso do filtro decimador.

B.3 Resultados do Filtro Decimador

Esta seção tem como objetivo apresentar os resultados obtidos com a implementação do filtro decimador de terceira ordem para moduladores ΣΔ.

B.3.1 Resultados da Modelagem Funcional e da Síntese Lógica

B.3.1.1 Circuito Divisor de Clock

O circuito usado para decimar a frequência do *clock* em 64 vezes foi implementado por um *ripple carry counter* de 6 bits. Sua concepção é baseada pura e simplesmente na disposição de *flip-flops* T em cascata. As Figuras 60 e 61 mostram os resultados obtidos da simulação funcional pré-síntese-lógica e síntese lógica, utilizando-se da tecnologia TSMC 0.18 μm.

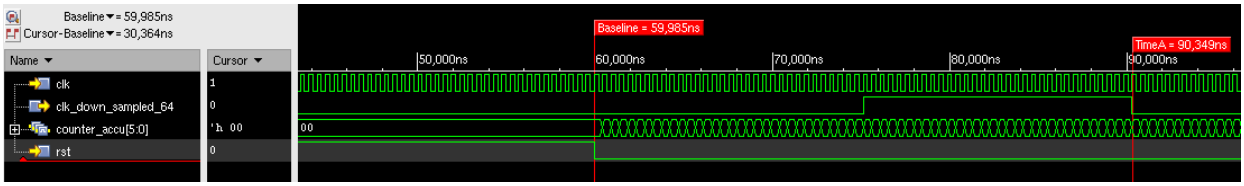

Figura 60 – Simulação funcional do circuito divisor de *clock*.

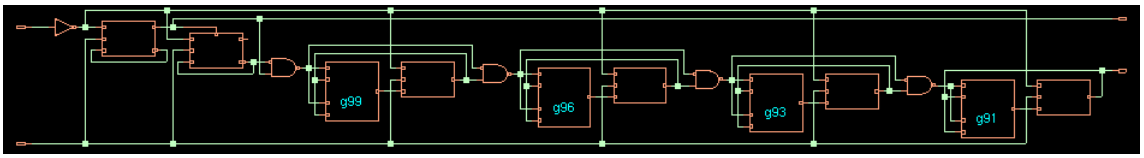

Figura 61 – Síntese lógica do circuito divisor de *clock*.

B.3.1.2 Topo do Filtro Decimador

O topo do filtro decimador, pós-síntese lógica é mostrado na Figura 62. Nota-se basicamente dois blocos, o circuito divisor de *clock* e o filtro, respectivamente.

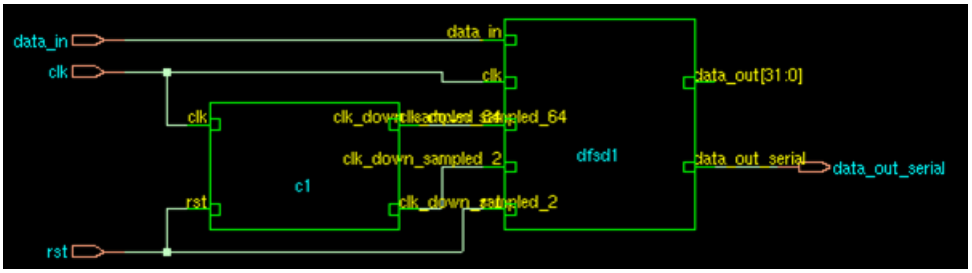

Figura 62 – Topo do filtro decimador.

B.3.1.3 Resposta à um Pulso

A resposta à um pulso, que vai do valor lógico zero para o valor lógico um, e depois, volta ao valor zero, é mostrada abaixo. Outros testes para validação da arquitetura pré-síntese lógica foram feitos e serão mostrados durante a apresentação do projeto.

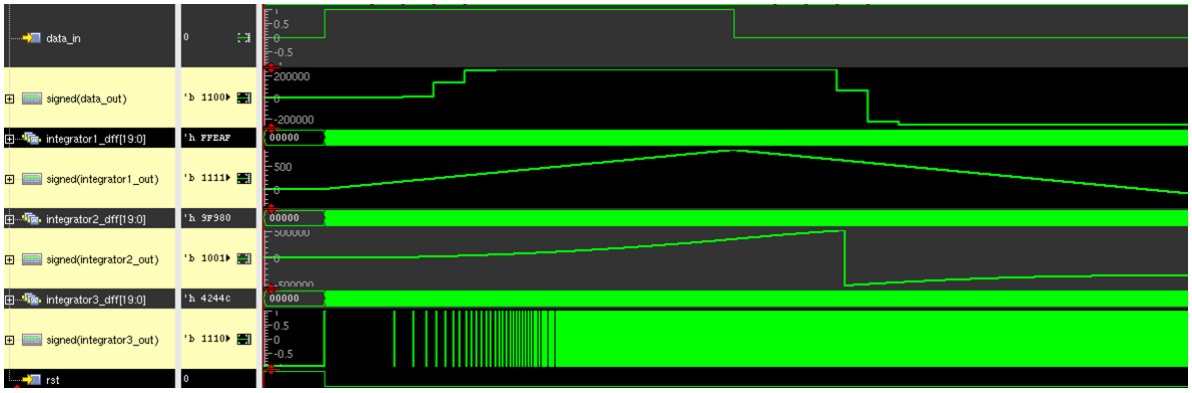

Figura 63 – Resposta à um pulso.

B.3.2 Resultados da Síntese Física

B.3.2.1 Layout do Circuito Final do Decimador

A Figura 64 mostra o resultado da implementação física feita no *software Cadence* RTL-*to*-GDSII. Nesse ponto, as *standard cells* do projeto já haviam sido colocadas, organizadas e roteadas. A camada extra de metais, para balanceamento (*metal density*) foi omitida, para melhor visualização.

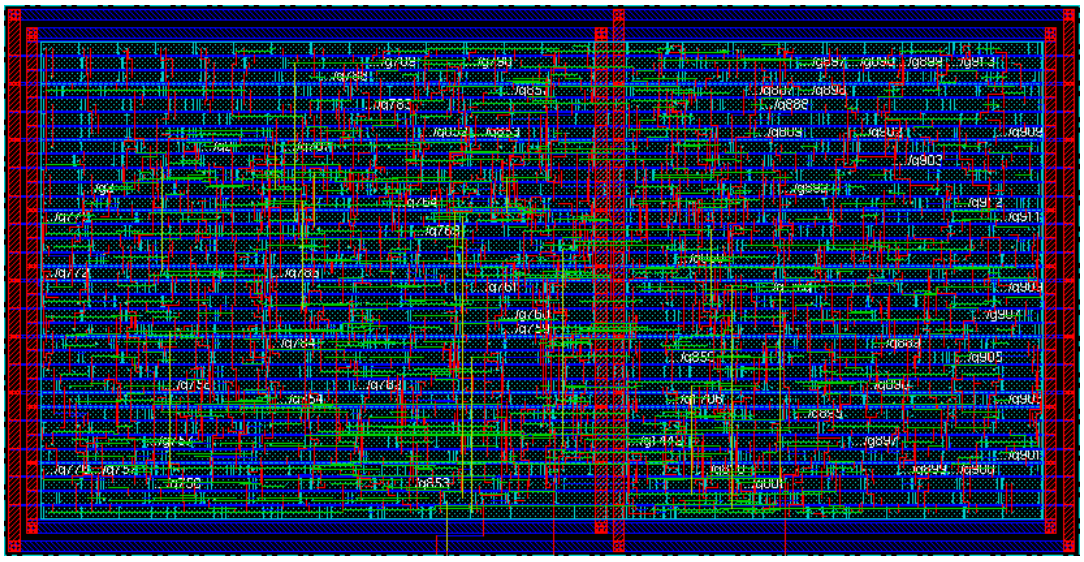

Figura 64 – Implementação Física no RTL-*to*-GDSII.

B.3.2.2 Resumo do Relatório Final de Síntese

Um trecho do resumo do relatório final de síntese (*i.e., summaryReport*), provido pelo *Cadence* RTL-*to*-GDSII, é mostrado. A área total do *chip* é de aproximadamente 150×270 ($\approx 0.046\ mm^2$).

```
==============================
Floorplan/Placement Information
==============================
Total area of Standard cells: 37243.763 um^2
Total area of Standard cells(Subtracting Physical Cells): 30137.901 um^2
Total area of Macros: 0.000 um^2
Total area of Blockages: 0.000 um^2
Total area of Pad cells: 0.000 um^2
Total area of Core: 37310.403 um^2
Total area of Chip: 46011.067 um^2
Effective Utilization: 1.0289e+00
Number of Cell Rows: 34
```

Figura 65 – Resumo do relatório final de síntese do filtro decimador.

Anexos

- **Os anexos mostrados aqui são referentes à:**

  1. Modelagem do Sistema em MATLAB (Ponto-flutuante) – primeiro modelo
  2. Modelagem do Sistema em MATLAB (Ponto-fixo) – modelo usado
  3. Exemplo Prático da Filtragem de um Sinal de Audio para o MATLAB e *SimVision*
  4. Código Verilog do Filtro Equalizador LMS Adaptativo (Paralelo)
  5. Código do Conversor Analógico-digital em Verilog-AMS
  6. Código Verilog do Filtro Decimador de Terceira Ordem para Modulares $\Sigma\Delta$

- **OBSERVAÇÃO**: Todos os arquivos de projetos (*project directories*), *scripts* e relatórios de sínteses lógica e física e códigos referentes à todas as implementações apresentadas neste documento estão gravados no CD-ROM em anexo.

ANEXO A – Modelagem do Sistema em MATLAB (Ponto-flutuante)

```
%-------------------------------------------------------------
%
%   Projeto e Implementação de um Filtro Equalizador LMS
%   Adaptativo ao Canal de Comunicação
%
%   Trabalho de Conclusão de Curso
%   Universidade de Brasília - UnB/FGA
%   Aluno: João Pedro da Silva Cerqueira
%   Matrícula: 10/45717
%   File: tcc2_ponto_flutuante.m
%
%-------------------------------------------------------------

clc % Limpa a tela
clear all; % Limpa todas as variáveis
close all; % Fecha todas as janelas

% Tamanho da sequência aleatória
length = 3000;


%% Geração da sequência de entrada

% randint(0..n, N): gera N inteiros aleatoriamente entre 0 e n (inclusivo).
% msg é a sequência de entrada aleatória de 0s e 1s.
msg = randint(1, length);

% O próximo passo consiste em aplicar um esquema de modulação à sequência
% gerada. O esquema aplicado será o Binary Phase-Shift Keying (BPSK).
xk = (2 * msg) - 1;

% Modelagem do canal (ck)
% 4-taps multipath, espaçamento de 1T
% rk é o sinal obtido após a convolução de xk e o ck.
rk = filter([1 0.5 0.2 -0.1], [1], xk);

% Geração do ruído
noise = 0.1 * randn(1, length);

% Adição do ruído à sequência rk.
yk = rk + noise;

%% Especificação do Filtro

% Coeficientes do equalizador
wn = zeros(5, 1);

% Valor da constante alpha
alph = 0.03125;

mse = zeros(1, length);
```

```
 for k = 10:length

     % Cálculo do erro
     err = xk(k) - yk(k:-1:k-4) * wn;

     % Eleva o valor err ao quadrado
     mse(k) = err^2;

     % Atualiza o coeficiente wn
     wn = wn + alph * err * yk(k:-1:k-4)';
 end

%% Resultados
plot(mse) % gráfico do mean squared error (MSE)
grid;
wn % mostra os coeficientes depois de 3000 iterações
```

ANEXO B – Modelagem do Sistema em MATLAB (Ponto-fixo)

```
%-------------------------------------------------------------
%
%   Projeto e Implementação de um Filtro Equalizador LMS
%   Adaptativo ao Canal de Comunicação
%
%   Trabalho de Conclusão de Curso
%   Universidade de Brasília - UnB/FGA
%   Aluno: João Pedro da Silva Cerqueira
%   Matrícula: 10/45717
%   File: tcc2_lms_direct_16b.m
%
%-------------------------------------------------------------


clc % Limpa a tela
clear all; % Limpa todas as variáveis
close all; % Fecha todas as janelas


% Tamanho da sequência aleatória
length = 3000;


%% Geração da sequência de entrada

% randint(0..n, N): gera N inteiros aleatoriamente entre 0 e n (inclusivo).
% sk é a sequência de entrada aleatória de 0s e 1s.
%sk = randint(1, length);
sk = importdata('sk.mat');

% O próximo passo consiste em aplicar um esquema de modulação à sequência
% gerada. O esquema aplicado será o Binary Phase-Shift Keying (BPSK).
sk_bpsk = (2 * sk) - 1;

% Modelagem do canal (ck)
% 4-taps multipath, espaçamento de 1T
% rk é o sinal obtido após a convolução de sk_bpsk e o ck.
rk = filter([1 0.5 0.2 -0.1], [1], sk_bpsk);

% Geração do ruído
%nk = 0.1 * randn(1, length);
nk = importdata('nk.mat');

% Adição do ruído à sequência rk.
% xk é a entrada do filtro equalizador adaptivo.
xk = rk + nk;


%% Quantização de sk_bpsk ==> (min,max) = (-1.5, +1.5)
dk_quant = sfi(sk_bpsk, 24, 18);
```

```
%% Quantização de xk ==> (min,max) = (-2.1, +2.1)
xk_quant = sfi(xk, 16, 13);


%% Especificação do Filtro

% Coeficientes do equalizador
wn = zeros(6, 1);

% Valor da constante alpha
alph = 0.03125;

mse = zeros(1, length);

yk_aux = zeros(1, length-5);

 for k = 6:length,

     % Cálculo de yk
     yk = xk(k:-1:k-5) * wn;
     yk_aux(k:-1:k-5) = yk;

     % Cálculo do erro
     err = sk_bpsk(k) - yk;

     % Eleva o valor err ao quadrado
     mse(k) = err^2;

     % Cálculo de alph_err
     alph_err = alph * err;

     % Cálculo do alph_err * xn_reg
     m_alph_err_xn_reg = alph_err * xk(k:-1:k-5);

     % Atualiza o coeficiente wn
     wn = wn + m_alph_err_xn_reg';

 end


%% Resultados
plot(mse) % gráfico do mean squared error (MSE)
grid;

wn % mostra os coeficientes depois de 3000 iterações
wnq = sfi(wn,16,13);
wnq * 2^14
```

ANEXO C – Exemplo Prático da Filtragem de um Sinal de Audio para o MATLAB e SimVision

```
%-------------------------------------------------------------
%
%   Projeto e Implementação de um Filtro Equalizador LMS
%   Adaptativo ao Canal de Comunicação
%
%   Trabalho de Conclusão de Curso
%   Universidade de Brasília - UnB/FGA
%   Aluno: João Pedro da Silva Cerqueira
%   Matrícula: 10/45717
%   File: tcc2_lms_direct_16b.m
%
%-------------------------------------------------------------

clc % Limpa a tela
clear all; % Limpa todas as variáveis
close all; % Fecha todas as janelas


%% Aquisição e quantização do sinal
quantiza_sinal;


%% Geração da sequência de entrada
sk = importdata('sk.mat');


% Tamanho da sequência aleatória
length = length(sk);


% O próximo passo consiste em aplicar um esquema de modulação à sequência
% gerada. O esquema aplicado será o Binary Phase-Shift Keying (BPSK).
sk_bpsk = (2 * sk) - 1;


% Modelagem do canal (ck)
% 4-taps multipath, espaçamento de 1T
% rk é o sinal obtido após a convolução de sk_bpsk e o ck
rk = filter([1 0.5 0.2 -0.1], [1], sk_bpsk);


% Geração do ruído
nk = 0.1 * randn(1, length);


% Adição do ruído à sequência rk:
% xk é a entrada do filtro equalizador adaptivo
xk = rk + nk;


%% Quantização de sk_bpsk ==> (min,max) = (-1.5, +1.5)
```

```
dk_quant = sfi(sk_bpsk, 24, 18);


%% Quantização de xk ==> (min,max) = (-2.1, +2.1)
xk_quant = sfi(xk, 16, 13);


%% Especificação do Filtro
% Coeficientes do equalizador:
wn = zeros(6, 1);
%wn = [0.9983 -0.4995 0.0524 0.1682 -0.1406 0.0477];
%wn = wn';


% Valor da constante alpha (2u)
alph = 0.03125;


% Vetor MSE (valor quadrático do erro, e^2(k))
mse = zeros(1, length);


yk_aux = zeros(1, length-6);


%% Iteração -- Algoritmo LMS
 for k = 6:length,

     % Cálculo de yk
     yk = xk(k:-1:k-5) * wn;
     yk_aux(k:-1:k-5) = yk;

     % Cálculo do erro
     err = sk_bpsk(k) - yk;

     % Eleva o valor err ao quadrado
     mse(k) = err^2;

     % Cálculo de alph_err
     alph_err = alph * err;

     % Cálculo do alph_err * xn_reg
     m_alph_err_xn_reg = alph_err * xk(k:-1:k-5);

     % Atualiza o coeficiente wn
     wn = wn + m_alph_err_xn_reg';

 end


%% Resultados
```

```
plot(mse) % gráfico do erro (parte positiva)
grid;

wn % mostra os coeficientes depois de <length> iterações
wnq = sfi(wn,16,13);
wnq * 2^14 % mostra os coeficientes em formato decimal (signed)


% Estimativa do erro
err_est = sum(mse(1,3000:10000));
err_est = err_est/(10000-3000)


%% Reconstrução do sinal yk (filtrado pelo equalizador LMS)
% O sinal yk é a saída do filtro equalizador LMS
reconst_yk = zeros (1, length);

for i = 1:1:length,
    if (yk_aux(i) < 0)
        reconst_yk(i) = 0;
    else
        reconst_yk(i) = 1;
    end
end

savefile = 'reconst_yk.mat';

save(savefile, 'reconst_yk');

reconstruindo_yk; % reconstrução do sinal yk (processado)


%% Reconstrução do sinal xk (sem processamento)
% O sinal xk é corrompido pelo ruído e distorções
reconst_xk = zeros (1, length);

for i = 1:1:length,
    if (xk(i) < 0)
        reconst_xk(i) = 0;
    else
        reconst_xk(i) = 1;
    end
end

savefile = 'reconst_xk.mat';

save(savefile, 'reconst_xk');

reconstruindo_xk; % reconstrução do sinal xk (sem processamento)
```

```
%-------------------------------------------------------------
%
%   Projeto e Implementação de um Filtro Equalizador LMS
%   Adaptativo ao Canal de Comunicação
%
%   Trabalho de Conclusão de Curso
%   Universidade de Brasília - UnB/FGA
%   Aluno: João Pedro da Silva Cerqueira
%   Matrícula: 10/45717
%   File: quantiza_sinal.m
%
%-------------------------------------------------------------

% Lê o sinal e a sua freq. de amostragem em flute e fs, respectivamente
[flute,fs]=wavread('flute.wav');

% Separa apenas 1 canal (mono)
sig=flute(:,1);

% Tamanho da vetor sig
length_sig = 96375;

partition = [-1:2/65535:1]; % quantização em 16 bits

codebook = [0:1:65536];

% Quantiza o sinal sig
quants = quantiz (sig, partition, codebook);

% Converte sig para binário
r = de2bi (quants, 'left-msb');

% res_2 é a entrada serial de bits
res_2 = zeros (1, 16*length_sig);

count = 0;

% Converte r (matriz) em um vetor de números binários
for i = 1:1:length_sig,
    for j = 1:1:16,
        count = count + 1;
        aux = r(i, j);
        res_2(1, count) = aux;
    end
end

savefile = 'sk.mat'; % nome do arquivo a ser salvo

% Salva a sequência de símbolos a serem transmitidos, sk
save(savefile, 'res_2');
```

```
%-------------------------------------------------------------
%
%   Projeto e Implementação de um Filtro Equalizador LMS
%   Adaptativo ao Canal de Comunicação
%
%   Trabalho de Conclusão de Curso
%   Universidade de Brasília - UnB/FGA
%   Aluno: João Pedro da Silva Cerqueira
%   Matrícula: 10/45717
%   File: reconstruindo_xk.m
%
%-------------------------------------------------------------


% reconst_xk recebe o sinal xk, corrompido
reconst_xk = importdata ('reconst_xk.mat');

count_xk = 0;


% Reconvertendo xk para formato matriz
for i = 1:1:length_sig,
    for j = 1:1:16,
        count_xk = count_xk + 1;
        r_xk(i, j) = reconst_xk(count_xk);
    end
end

% Reconvertendo xk para decimal
xk_reconstruido = bi2de(r_xk);

% Transposta de xk
xk_reconstruido = xk_reconstruido';
```

```
%-------------------------------------------------------------
%
%   Projeto e Implementação de um Filtro Equalizador LMS
%   Adaptativo ao Canal de Comunicação
%
%   Trabalho de Conclusão de Curso
%   Universidade de Brasília - UnB/FGA
%   Aluno: João Pedro da Silva Cerqueira
%   Matrícula: 10/45717
%   File: reconstruindo_yk.m
%
%-------------------------------------------------------------


% reconst_yk recebe o sinal yk, corrompido
reconst_yk = importdata ('reconst_yk.mat');

count_yk = 0;


% Reconvertendo yk para formato matriz
for i = 1:1:length_sig,
    for j = 1:1:16,
        count_yk = count_yk + 1;
        r_yk(i, j) = reconst_yk(count_yk);
    end
end

% Reconvertendo yk para decimal
yk_reconstruido = bi2de(r_yk);

% Transposta de yk
yk_reconstruido = yk_reconstruido';
```

ANEXO D – Código Verilog do Filtro Equalizador LMS Adaptativo (Paralelo)

```
//=============================================================================
/*

    This file is part of the 6-Tap Direct FIR LMS Equalizer.
    2012-2013 by Joao P S Cerqueira <joaopscerqueira@gmail.com>.

    The 6-Tap Direct FIR LMS Equalizer operates in two different modes:
    Training and Direct Decision. In order to find the optimal coefficients
    that best fit the inverse of the channel transfer function it is needed
    to assert the variable 'mode' to 1'b1 (training mode). When 'mode' is set
    to 1'b0, the module works as a simple equalizer (direct decision).

    - Design:       6-Tap Direct FIR Least Mean Squares Equalizer
    - Module:       lms_direct
    - File:         lms_direct.v
    - Version:      1.0

    - Author:       Joao Pedro da Silva Cerqueira
    - Email:        joaopscerqueira@gmail.com

*/
//=============================================================================


//=============================================================================
/*
    Module: 6-Tap Direct FIR LMS Equalizer
*/
//=============================================================================
module lms_direct (xn, dn, mode, clk, rst, yn);

    input signed [15:0] xn; // x[n] is in Q16.13 format

    input signed [23:0] dn; // d[n] is in Q24.18 format

    input mode; // mode = 1 ==> training mode

    input clk;

    input rst;

    output signed [23:0] yn; // y[n] is in Q24.18 format

    reg signed [15:0] xn_reg[6:1]; // Q16.13

    wire signed [24:0] err; // Q25.18

    wire signed [24:0] alph_err; // Q25.18

    wire signed [40:0] m_alph_err_xn_reg[6:1]; // Q41.31

    wire signed [15:0] m_alph_err_xn_reg_16_14[6:1]; // Q16.14
```

```
    wire signed [36:0] yn_reg; // Q37.27

    reg signed [15:0] wn[6:1]; // Q16.14


    always @(posedge clk or posedge rst)
    begin
        if (rst)
        begin

            // reset
            xn_reg[1] <= 16'd0;
            xn_reg[2] <= 16'd0;
            xn_reg[3] <= 16'd0;
            xn_reg[4] <= 16'd0;
            xn_reg[5] <= 16'd0;
            xn_reg[6] <= 16'd0;

        end
        else
        begin

            // Q16.13
            xn_reg[1] <= xn;
            xn_reg[2] <= xn_reg[1];
            xn_reg[3] <= xn_reg[2];
            xn_reg[4] <= xn_reg[3];
            xn_reg[5] <= xn_reg[4];
            xn_reg[6] <= xn_reg[5];

        end
    end


    // yn_reg is in Q37.27 format
    // wn ==> Q16.14
    // xn_reg ==> Q16.13
    assign yn_reg = wn[1] * xn_reg[1] + // Q32.27
                    wn[2] * xn_reg[2] + // Q32.27
                    wn[3] * xn_reg[3] + // Q32.27
                    wn[4] * xn_reg[4] + // Q32.27
                    wn[5] * xn_reg[5] + // Q32.27
                    wn[6] * xn_reg[6];  // Q32.27


    // Q24.18
    assign yn = {yn_reg[36], yn_reg[31:27], yn_reg[26:9]};


    // dn ==> Q24.18
    // dff_yn ==> Q24.18
    // err ==> Q25.18
    assign err = dn - yn;
```

```
// Q25.18 ==> 25.18
assign alph_err = {err[24], err[24], err[24], err[24], err[24], err[24:5]};


// Q25.18
// Q16.13
// Q41.31
assign m_alph_err_xn_reg[1] = alph_err * xn_reg[1];
assign m_alph_err_xn_reg[2] = alph_err * xn_reg[2];
assign m_alph_err_xn_reg[3] = alph_err * xn_reg[3];
assign m_alph_err_xn_reg[4] = alph_err * xn_reg[4];
assign m_alph_err_xn_reg[5] = alph_err * xn_reg[5];
assign m_alph_err_xn_reg[6] = alph_err * xn_reg[6];


// Q16.14
assign m_alph_err_xn_reg_16_14[1] = {m_alph_err_xn_reg[1][40], m_alph_err_xn_reg[1][31:17]};
assign m_alph_err_xn_reg_16_14[2] = {m_alph_err_xn_reg[2][40], m_alph_err_xn_reg[2][31:17]};
assign m_alph_err_xn_reg_16_14[3] = {m_alph_err_xn_reg[3][40], m_alph_err_xn_reg[3][31:17]};
assign m_alph_err_xn_reg_16_14[4] = {m_alph_err_xn_reg[4][40], m_alph_err_xn_reg[4][31:17]};
assign m_alph_err_xn_reg_16_14[5] = {m_alph_err_xn_reg[5][40], m_alph_err_xn_reg[5][31:17]};
assign m_alph_err_xn_reg_16_14[6] = {m_alph_err_xn_reg[6][40], m_alph_err_xn_reg[6][31:17]};


always @(posedge clk or posedge rst)
begin
    if (rst)
    begin

        // reset
        // Q16.14
        wn[1] <= 16'd0;
        wn[2] <= 16'd0;
        wn[3] <= 16'd0;
        wn[4] <= 16'd0;
        wn[5] <= 16'd0;
        wn[6] <= 16'd0;

    end
    else
    begin

        if (mode == 1'b1)
        begin

            // m_alph_err_xn_reg_16_14 ==> Q16.14
            // wn ==> Q16.14
            wn[1] <= m_alph_err_xn_reg_16_14[1] + wn[1];
            wn[2] <= m_alph_err_xn_reg_16_14[2] + wn[2];
            wn[3] <= m_alph_err_xn_reg_16_14[3] + wn[3];
            wn[4] <= m_alph_err_xn_reg_16_14[4] + wn[4];
```

```
                wn[5] <= m_alph_err_xn_reg_16_14[5] + wn[5];
                wn[6] <= m_alph_err_xn_reg_16_14[6] + wn[6];

            end
            else
            begin

                wn[1] <= wn[1];
                wn[2] <= wn[2];
                wn[3] <= wn[3];
                wn[4] <= wn[4];
                wn[5] <= wn[5];
                wn[6] <= wn[6];

            end
        end
    end

endmodule
```

ANEXO E – Código do Conversor Analógico-digital em Verilog-AMS

```
//
// CONVERSOR AD DE 16 BITS EM VERILOG-AMS
// AUTOR: Joao P S Cerqueira
// REF: CADENCE MODELWRITER
//
// PARAMETERS:
//   slack = The smallest time interval considered negligible for
// cross event on clock [S]
//   tconv = Delay from threshold crossing to output change [S]
//   trise = Rise time for digital output signals [S]
//   trise = Rise time for digital output signals [S]
//    vmax = ADC Full scale output voltage [V]
//    vmin = ADC Zero scale output voltage [V]
//    vone = The voltage of a logical 1 on digital outputs [V]
//     vth = Threshold value of clock signal [V]
//   vzero = The voltage of a logical 0 on digital outputs [V]
//

`include  "discipline.h"
`include  "constants.h"
`define NUM_ADC_BITS   16

module adc_ideal_16b (vin, clk, dout);
 input   vin, clk;
 electrical vin, clk;

 output  [`NUM_ADC_BITS-1:0] dout;
 electrical [`NUM_ADC_BITS-1:0] dout;

  parameter real  vmax = 5;
  parameter real  vmin = -5;
  parameter real   one = 5.0;
  parameter real  zero = 0.0;
  parameter real   vth = 3.0;
  parameter real slack = 10.0p from (0:inf);
  parameter real trise = 4.0u from (0:inf);
  parameter real tfall = 4.0u from (0:inf);
  parameter real tconv = 2.0u from [0:inf);
  parameter integer traceflag = 0;

     real   sample, vref, lsb, voffset;
     real   vd[0:`NUM_ADC_BITS-1];
     integer ii, binvalue;

    analog begin
      @(initial_step or initial_step("dc", "ac", "tran", "xf"))  begin
        vref = (vmax - vmin) / 2.0;
        lsb  = (vmax - vmin) / (1 << `NUM_ADC_BITS) ;
        voffset = vmin;

        if (traceflag)
        $display("%M ADC  range ( %g v ) /  %d bits  = lsb %g volts.\n",
                    vmax - vmin, `NUM_ADC_BITS, lsb );
```

```
        generate i ( `NUM_ADC_BITS-1, 0) begin
            vd[i] = 0 ;
        end
    end

      @(cross ( V(clk)-vth,  1, slack, clk.potential.abstol)) begin
          binvalue = 0;
          sample = V(vin) - voffset;
          for ( ii = `NUM_ADC_BITS -1 ; ii>=0 ; ii = ii -1 ) begin
            vd[ii] = 0;
            if (sample > vref ) begin
              vd[ii] = one;
              sample = sample - vref;
              binvalue = binvalue + ( 1 << ii );
            end
            else begin
             vd[ii] = zero;
            end
            sample = sample * 2.0;
          end
          if (traceflag)
            $strobe("%M at %g sec. digital out: %d   vin: %g  (d2a: %g)\n",
                       $abstime, binvalue,  V(vin), (binvalue*lsb)+voffset);
      end

      generate i ( `NUM_ADC_BITS-1, 0) begin
         V(dout[i])  <+   transition ( vd[i] , tconv, trise, tfall );
      end
    end
endmodule

`undef NUM_ADC_BITS
```

ANEXO F – Código Verilog do Filtro Decimador de Terceira Ordem para Modulares $\Sigma\Delta$

```
//================================================================================
/*

    This file is part of the Decimator Filter (CIC 3rd Order), used in
    Sigma-delta Analog-to-Digital Converters.
    Mar 21, 2014 by Joao P S Cerqueira <joaopscerqueira@gmail.com>.

    - Design:       Decimator Filter (CIC 3rd Order)
    - Module:       df_cic_sigma_delta
    - File:         df_cic_sigma_delta.v
    - Version:      1.0

    - Author:       Joao Pedro da Silva Cerqueira
    - Email:        joaopscerqueira@gmail.com

*/
//================================================================================


//================================================================================
/*
    Module: Decimator Filter (CIC 3rd Order)
*/
//================================================================================
module df_cic_sigma_delta (data_in, clk, clk_down_sampled, rst, data_out);

    input signed data_in;
    input clk;
    input clk_down_sampled;
    input rst;
    output signed [19:0] data_out;


    wire signed [19:0] coder_out;

    wire signed [19:0] integrator1_out;
    wire signed [19:0] integrator1_dff;

    wire signed [19:0] integrator2_out;
    wire signed [19:0] integrator2_dff;

    wire signed [19:0] integrator3_out;
    wire signed [19:0] integrator3_dff;

    wire signed [19:0] comb1_out;
    wire signed [19:0] comb1_out_dff;
    wire signed [19:0] comb1_dff;

    wire signed [19:0] comb2_out;
    wire signed [19:0] comb2_out_dff;
    wire signed [19:0] comb2_dff;

    wire signed [19:0] comb3_out;
```

```
    wire signed [19:0] comb3_dff;


    coder_circuit cc1 (data_in, coder_out);


    D_FF_hf dff0 [19:0] (integrator1_dff[19:0], integrator1_out[19:0], clk, rst);
    D_FF_hf dff1 [19:0] (integrator2_dff[19:0], integrator2_out[19:0], clk, rst);
    D_FF_hf dff2 [19:0] (integrator3_dff[19:0], integrator3_out[19:0], clk, rst);

    D_FF_lf dff3 [19:0] (comb1_dff[19:0], integrator3_out[19:0], clk, clk_down_sampled, rst);
    D_FF_lf dff4 [19:0] (comb2_dff[19:0], comb1_out_dff[19:0], clk, clk_down_sampled, rst);
    D_FF_lf dff5 [19:0] (comb3_dff[19:0], comb2_out_dff[19:0], clk, clk_down_sampled, rst);

    D_FF_lf dff6 [19:0] (comb1_out_dff[19:0], comb1_out[19:0], clk, clk_down_sampled, rst);
    D_FF_lf dff7 [19:0] (comb2_out_dff[19:0], comb2_out[19:0], clk, clk_down_sampled, rst);
    D_FF_lf dff8 [19:0] (data_out[19:0], comb3_out[19:0], clk, clk_down_sampled, rst);


    assign integrator1_out = coder_out + integrator1_dff;
    assign integrator2_out = integrator1_out + integrator2_dff;
    assign integrator3_out = integrator2_out + integrator3_dff;


    assign comb1_out = integrator3_out - comb1_dff;
    assign comb2_out = comb1_out_dff - comb2_dff;
    assign comb3_out = comb2_out_dff - comb3_dff;

endmodule


//================================================================================
/*
    Module: Counter-Clock-Divider
*/
//================================================================================
module counter_clk_div (clk, rst, clk_down_sampled);

    input clk;
    input rst;
    output clk_down_sampled;

    reg [5:0] counter_accu;

    always @(posedge clk or posedge rst)
    begin
        if (rst)
        begin
            // reset
            counter_accu <= 6'b00_0000;
        end
        else
        begin
```

```
            counter_accu <= counter_accu + 1;
        end
    end


    assign clk_down_sampled = counter_accu[5];

endmodule


//================================================================================
/*
    Module: Coder Circuit
*/
//================================================================================
module coder_circuit (data_in, data_out);

    input data_in;
    output [19:0] data_out;

    assign data_out = (data_in == 1'b0) ? 20'b1111_1111_1111_1111_1111 :
    20'b0000_0000_0000_0000_0001;

endmodule


// 1-bit D Flip-flop
// Author: Joao P S Cerqueira
module D_FF_hf (q, d, clk, reset);
    output reg q;
    input d;
    input clk, reset;

    always@(posedge clk)
    begin
        if (reset) q <= 1'b0;
        else q <= d;
    end
endmodule


// 1-bit D Flip-flop
// Author: Joao P S Cerqueira
module D_FF_lf (q, d, clk, clk_down_sampled, reset);
      output reg q;
    input d;
    input clk, clk_down_sampled, reset;

    always@(posedge clk)
    begin
        if (reset) q <= 1'b0;
    end
```

```
    always @(posedge clk_down_sampled)
    begin
        q <= d;
    end
endmodule
```