UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

YUMI MONMA

PROJETO DE DIPLOMAÇÃO

VERIFICAÇÃO FUNCIONAL DE PROCESSADOR PARA CONTROLE DE RUÍDO ACÚSTICO

Porto Alegre
23 de Junho de 2008

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

VERIFICAÇÃO FUNCIONAL DE PROCESSADOR PARA CONTROLE DE RUÍDO ACÚSTICO

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para Graduação em Engenharia Elétrica.

ORIENTADOR: Prof. Dr. Eric Ericson Fabris

Porto Alegre
2008

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

YUMI MONMA

VERIFICAÇÃO FUNCIONAL DE PROCESSADOR PARA CONTROLE DE RUÍDO ACÚSTICO

Este projeto foi julgado adequado para fazer jus aos créditos da Disciplina de "Projeto de Diplomação", do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

ORIENTADOR: _____
Prof. Dr. Eric Ericson Fabris, UFRGS

Banca Examinadora:

Prof. Dr. Eric Ericson Fabris, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Altamiro Amadeu Susin, UFRGS
Doutor pelo Institut National Polytechnique – Grenoble, França

Eng. Alcides Silveira Costa, NSCAD Microeletrônica
Engenheiro pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Porto Alegre, Junho de 2008

SUMÁRIO

ÍNDICE DE FIGURAS

LISTA DE ABREVIATURAS

ASIC : *Application-specific integrated circuit*

Corner cases : Cenários de pouca probabilidade de ocorrência

Design : Circuito digital

DUT : *Design under test*

EDA : *Electronic design automation*

IES : *Incisive Enterprise Simulator*

LMS : *Least mean square*

MCT : Ministério de ciência e tecnologia

OVM : *Open Verification Methodology*

PCM : *Pulse code modulation*

Top-level : Módulo que engloba os blocos do sistema

VMM : *Verification Methodology Management*

RESUMO

Este trabalho apresenta os aspectos de estudo de algoritmo e verificação funcional do projeto IP02-ANC. O projeto, executado na empresa NSCAD Microeletrônica entre Março e Julho de 2008, tem por objetivo desenvolver um circuito integrado para controle de ruído acústico. O capítulo inicial apresenta o problema de exposição da população ao ruído e a relevância do desenvolvimento de produtos que o minimizem. O capítulo seguinte apresenta o projeto IP02-ANC e as tarefas definidas em seu contexto. O capítulo 3 aborda o estudo do algoritmo utilizado e a implementação de um modelo que reproduz as características especificadas para o projeto. A verificação funcional do projeto do circuito digital, bem como uma introdução aos conceitos e metodologias utilizadas para tal fim são apresentadas no capítulo 4. Por fim, são apresentados os resultados experimentais e as conclusões sobre o trabalho desenvolvido.

Palavras-chave: Verificação funcional, controle de ruído ativo, processamento de sinais, filtros digitais, circuitos integrados

ABSTRACT

This document presents the implementation of the functional model and the development of the verification environment for the IP02-ANC project, which aims to produce an integrated circuit for active noise control. The project was done in NSCAD Microeletrônica between March and July of 2008. The first chapter of this document introduces the problem of the constant exposure of the urban population to acoustic noise, and how important it is to develop solutions for it. The second chapter gives more explanation about the IP02-ANC project, including the tasks that were defined within it. The third chapter explores the algorithm study and implementation of the functional model for the project. After that, the fourth chapter contains a more detailed explanation about functional verification and the methodology that was adopted. Finally, the experimental results and the conclusions are presented in chapters five and six.

Keywords: Functional verification, active noise control, digital signal processing, digital filters, integrated circuits

1 INTRODUÇÃO

1.1 O PROBLEMA DA EXPOSIÇÃO AO RUÍDO AMBIENTE

Nos últimos anos, devido ao aumento da população nos grandes centros urbanos, combinado com desenvolvimento tecnológico e uma constante busca por redução de custos, é inegável a observação dos seguintes fatos [1]:

- Aumento da densidade de habitantes e moradias em grandes cidades
- Aumento do número de equipamentos industriais de grande porte, como motores, exaustores, ventiladores e compressores
- Uso de materiais mais leves e baratos na construção civil

Os fatos acima trazem como conseqüências:

- Aumento da exposição da população ao ruído causado pela vizinhança e pelo tráfego
- Aumento da exposição de trabalhadores ao ruído industrial
- Habitações com menor isolamento contra ruído externo

A exposição ao ruído pode acarretar em doenças como perda auditiva, e também é associada à qualidade de vida da população. Devido a isso, as conseqüências e formas de reduzir a exposição ao ruído são objeto de estudos realizados por pesquisadores de diversas áreas.

1.2 FORMAS DE REDUÇÃO DA EXPOSIÇÃO AO RUÍDO

Para a redução da exposição ao ruído, podem ser usados sistemas passivos, ativos ou combinações de ambos.

Sistemas passivos abafam o ruído através do uso de barreiras perpendiculares ou paralelas ao sentido de propagação do som. Podem ser citados como exemplos de barreira perpendicular os protetores auriculares utilizados na indústria, e como exemplos de barreira paralela os silenciadores acoplados aos canos de armas de fogo. Os dois exemplos citados são mostrados na Figura 1.

FIGURA 1. SISTEMAS PASSIVOS DE REDUÇÃO DE RUÍDO

Sistemas passivos possuem alta atenuação em uma larga banda de freqüência, porém, eles são grandes, caros e pouco eficientes em freqüências mais baixas. Assim. De acordo com o ambiente, o uso de sistemas passivos não é suficiente.

1.3 O CONCEITO DE CONTROLE ATIVO DE RUÍDO ACÚSTICO

Também é possível utilizar sistemas ativos de controle de ruído, cujo funcionamento ocorre a partir da geração de um sinal acústico semelhante ao ruído ambiente, porém em fase contrária. Quando combinados no meio físico, o ruído e o sinal de "anti-ruído" se cancelam e tem-se apenas o erro residual. O princípio físico do cancelamento ativo de ruído é mostrado na Figura 2, extraída de [2].

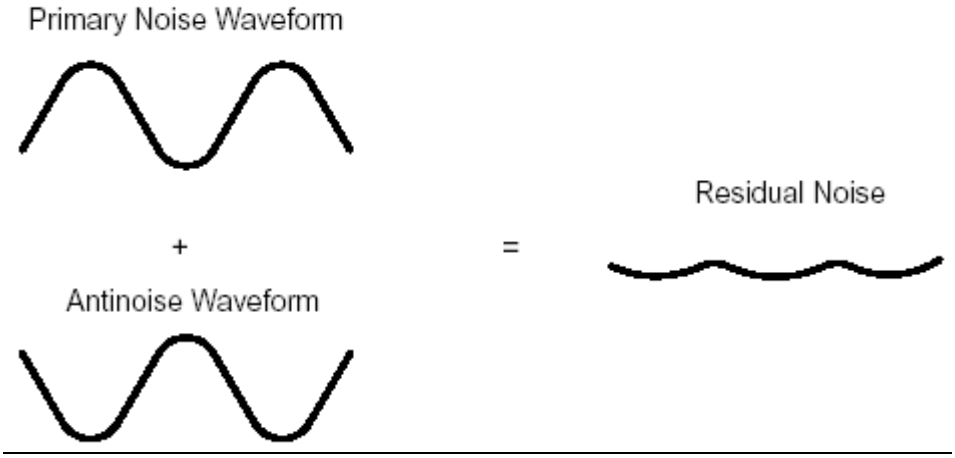


FIGURA 2. CONCEITO FÍSICO DE CANCELAMENTO DE RUÍDO ACÚSTICO.

Combinando-se conceitos de cancelamento de ruído com projetos de circuitos digitais é possível desenvolver dispositivos para diversas aplicações, tanto para a indústria como para o usuário final. Uma das aplicações é o projeto de fones de ouvido portáteis com a capacidade de cancelar o ruído externo.

1.4 SOLUÇÕES COMERCIAIS PARA CONTROLE ATIVO DE RUÍDO ACÚSTICO

Como exemplo de soluções comerciais de uso doméstico para controle de ruído acústico, pode-se citar os fones de ouvido *Quietcomfort 3*, da *Bose*, e *RP-HC500*, da *Panasonic*, mostrados na Figura 3. Ambos são alimentados por pilhas do tipo AAA e podem ser conectados a qualquer dispositivo reprodutor de áudio, como fones de ouvido convencionais.

FIGURA 3. FONES DE OUVIDO ANTI-RUÍDO COMERCIAIS

2 O PROJETO IP02-ANC

O projeto IP02-ANC, criado em Março de 2008 no NSCAD Microeletrônica, consiste na especificação e desenvolvimento de um processador de áudio para fone de ouvido com controle ativo de ruído acústico. O produto final desejado é um protótipo na forma de um circuito integrado de silício (também chamado de *ASIC: application-specific integrated circuit*).

2.1 DIVISÃO DO PROJETO

O projeto IP02-ANC foi dividido em três subprojetos, que foram designados a diferentes projetistas, resultando em três diferentes subprodutos:

- Projeto de circuito digital
- Projeto de circuito analógico
- Projeto de verificação funcional

Cada um dos três subprojetos é tema do trabalho de diplomação de um aluno do curso de engenharia elétrica no primeiro semestre de 2008.

2.2 CRONOGRAMA DO PROJETO

Para a correta comunicação e sincronização entre as atividades dos subprojetos, foi definido um cronograma contendo 95 tarefas distribuídas entre Março e Outubro de 2008. Foi utilizado o software *OpenProj* [3] de gestão de projetos, que possui código livre. As tarefas foram definidas levando-se em conta sua duração, relevância para o projeto e responsável. A Figura 4 mostra parte da janela principal do software e algumas tarefas definidas no projeto:

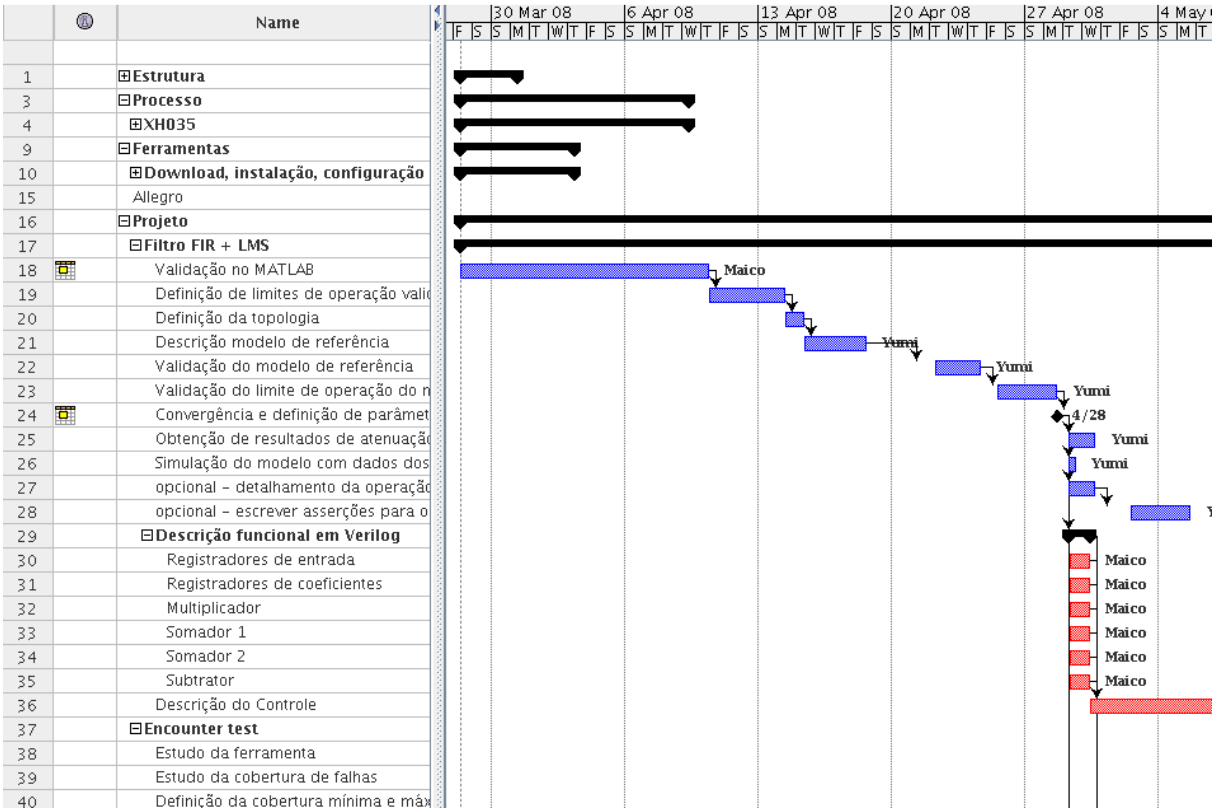


FIGURA 4. TAREFAS DO PROJETO IP02-ANC

2.3 TECNOLOGIA UTILIZADA

Devido a questões de custo e acesso a recursos financeiros para prototipação, optou-se por utilizar no projeto a tecnologia XL 0.35µm, da *foundry* (fabricante de *chips* de silício) *X-Fab* [4].

2.4 FERRAMENTAS DE EDA UTILIZADAS

Para a escolha das ferramentas de projeto de circuitos integrados (também conhecidas por ferramentas de *EDA* - *Electronic Design Automation*) a serem utilizadas, foram levados em conta os seguintes fatos:

- Atualmente a Universidade Federal do Rio Grande do Sul possui contratos de fornecimento de licenças firmados com os três maiores provedores de

ferramentas de *EDA* da indústria: *Synopsys* [5], *Mentor Graphics* [6] e *Cadence Design Systems* [7].

- A *Cadence Design Systems*, como parte do programa *CI Brasil*, do Ministério de Ciência e Tecnologia (MCT), ministrou 17 treinamentos abordando diversas etapas do fluxo de projeto de um *ASIC*. O público dos treinamentos foi composto por projetistas de centros filiados ao MCT e projetistas do NSCAD Microeletrônica. Os instrutores do curso foram consultores da empresa enviados de seu centro de treinamento nos Estados Unidos para o Brasil. Os treinamentos ocorreram entre o final de 2007 e meados de 2008.
- Ainda no contexto do programa *CI Brasil*, foi montado pelo NSCAD Microeletrônica um centro de treinamento que ofereceu 100 vagas para formação de projetistas de circuitos digitais, analógicos e de radiofreqüência. Da mesma forma que nos cursos anteriores, os instrutores foram profissionais da *Cadence Design Systems.*

Visando o aproveitamento do conhecimento adquirido nos cursos, e a possibilidade de contato com os consultores para recomendações técnicas, optou-se pela preferência à utilização das ferramentas da *Cadence Design Systems* para a execução do projeto.

2.5 OUTROS SOFTWARES UTILIZADOS

Para controle de versões de arquivos, foi configurado um repositório SVN [8]. O uso de repositórios é importante em projetos em equipe por permitir que os integrantes acompanhem as modificações nos arquivos e os sincronizem com arquivos locais. Além

disso, o repositório contém o histórico de todas as modificações e permite comparação e restauração de versões antigas com versões recentes.

Todas as ferramentas foram instaladas e configuradas na estação de trabalho de um dos membros da equipe. A seguir, foi utilizado o recurso de exportação de diretórios para a rede (*Network File System* [9], disponível em sistemas Unix/Linux) para disponibilizar a árvore instalada para os outros membros.

Para manter-se um registro das atividades diárias do projeto, foram criados diários online (*weblogs*) para os membros. A plataforma utilizada foi a *wordpress* [10] e o acesso foi limitado aos computadores da empresa. Os *weblogs* se mostraram muito úteis para a comunicação interna do projeto.

Além disso, a suíte de aplicativos online *Zoho* [11] foi largamente utilizada para gerenciamento de e-mail, comunicação instantânea, calendário e acompanhamento de tarefas.

2.6 ESCOPO DO TRABALHO

Este trabalho de diplomação aborda o subprojeto de verificação funcional do projeto IP02-ANC, além da criação de um modelo de referência funcional para o mesmo. O motivo pela qual a segunda tarefa foi atribuída para este trabalho é devido ao fato do modelo de referência ser muito relevante para a verificação funcional, e ao fato de ambos poderem ser construídos utilizando a mesma linguagem e simulados utilizando as mesmas ferramentas.

Este trabalho de diplomação abrange as atividades executadas entre Fevereiro e Junho de 2008.

Devido aos motivos apresentados na seção 2.4, foi utilizado majoritariamente a ferramenta *Incisive Enterprise Simulator*, da *Cadence Design Systems*.

Embora a opção inicial tenha sido o uso da ferramenta *Incisive Enterprise Manager* para a mensuração do atingimento do plano de verificação, não houve tempo hábil para tal tarefa ser incluída no escopo deste trabalho. Assim, a fim de ilustrar esta importante característica da verificação funcional, foi utilizado o simulador *QuestaSim*, da *Mentor Graphics*.

Para a geração e edição de amostras de sinal e ruído, foi utilizado o editor de áudio de código livre *Audacity* [12]

Para cálculos matemáticos e plotagem de resultados foi utilizada a plataforma de computação matemática *Scilab* [13], alternativa de código livre para o *Matlab*.

3 MODELO DE REFERÊNCIA

Modelo de referência é um módulo escrito em linguagem de alto nível que apresenta as mesmas características funcionais especificadas para o projeto. O modelo de referência do projeto IP02-ANC é um filtro adaptativo para cancelamento de ruído. A implementação de um modelo de referência, antes do início da codificação do projeto digital, traz muitos benefícios. Entre eles pode-se citar:

- A validação do algoritmo matemático utilizado, bem como a mensuração dos resultados ainda durante a fase preliminar de desenvolvimento;
- A definição de pontos vitais da especificação, como número de bits e freqüência de amostragem. A definição tardia da especificação é um erro comum em muitos projetos, e a implementação de um modelo de referência demanda teste com diferentes parâmetros e escolha de valores ideais;
- A utilização do modelo de referência para *debugging* do ambiente de verificação. É mais fácil detectar um *bug* no ambiente utilizando o modelo do que utilizando as descrições do projeto parcialmente finalizadas;

As linguagens mais utilizadas para este fim são *C* e *C++*. Também são utilizadas linguagens de *scripting* como *Perl*, *Python* e *Tcl*. Optou-se por utilizar a linguagem *SystemVerilog* devido à mesma se tratar de uma linguagem para descrição e verificação de hardware, contendo construções tanto para descrição de registradores e operadores lógicos como para criação de classes e métodos.

3.1 FORMULAÇÃO MATEMÁTICA

3.1.1 Processamento digital de sinais

Processamento de sinais é a disciplina que trata da identificação, modelamento e utilização de padrões de um sinal [14]. Estes processos podem ser realizados sobre sinais na forma analógica ou digital. O uso da abordagem digital insere no projeto a limitação do tempo de computação numérica, que varia de acordo com a complexidade das operações e sinais envolvidos. Esta limitação pode não tornar o projeto possível de ser executado em tempo real (tempo de computação que permite que o sistema opere para a finalidade desejada).

Por outro lado, o uso de processamento de sinais digitais traz como principais vantagens o ganho em:

- Flexibilidade: Uma mesma máquina digital pode ser usada para implementação de diferentes algoritmos ou diferentes versões do mesmo algoritmo;
- Repetitividade: Uma mesma operação de processamento pode ser repetida de maneira exatamente igual, sem variação de parâmetros em função das condições do ambiente, como podem ocorrer em sistemas de processamento analógico [15].

3.1.2 Filtro digital Wiener

O filtro digital Wiener [14], formulado a partir da teoria escrita por Norbert Wiener, tem por objetivo filtrar um sinal de entrada de forma a transformá-lo na forma mais próxima possível a um sinal de referência. A topologia do filtro Wiener é mostrada na Figura 5.

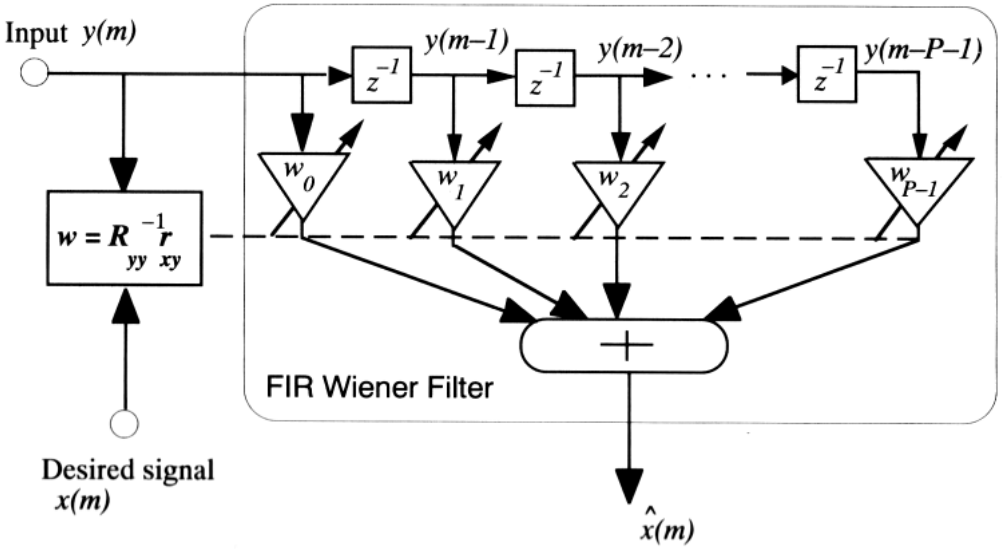


FIGURA 5. TOPOLOGIA DO FILTRO WIENER

Seus coeficientes são calculados de forma a minimizar a distância média quadrática entre os dois sinais. Na sua forma mais básica, a teoria assume que os sinais envolvidos são estacionários, o que na prática não ocorre com sinais de áudio. Contudo, quando os coeficientes do filtro são recalculados a cada bloco de amostras processadas, o filtro Wiener torna-se um filtro adaptativo apropriado para uso com sinais não-estacionários. Entre suas possíveis aplicações, pode-se citar equalização, codificação e restauração de sinais, predição linear, cancelamento de eco e identificação de sistemas.

3.1.3 Método de adaptação LMS

Uma forma computacionalmente simples de se atualizar os valores dos coeficientes, em função da saída do filtro, é utilizando o algoritmo de menor erro médio quadrático (*Least Mean Square - LMS*). No algoritmo, o valor atual de coeficiente é somado ao gradiente instantâneo da função de erro quadrático, multiplicado por um passo de adaptação.

A fórmula que sintetiza o algoritmo é definida por:

$$w(m+1) = w(m) + \mu \cdot e(m) \cdot y(m)$$

Sendo nesta fórmula:

- w(m+1): Novo valor do coeficiente
- w(m): Valor atual do coeficiente
- μ: Passo de adaptação
- e(m): Erro atual
- y(m): Valor atual da entrada do filtro

3.2 GERAÇÃO DE DADOS

De acordo com a especificação do projeto, o sinal de áudio de entrada do processador de controle de ruído provém de um conversor analógico/digital (*A/D*). Para a simulação do modelo com dados semelhantes, foram utilizados arquivos de áudio no formato WAVE, que não possui compressão.

Foi escolhido o software *Audacity* para a geração de amostras para teste dos módulos. As amostras foram geradas tanto a partir dos geradores de ruído e de ondas senoidais incluídos no software quanto a partir de extração de trechos de músicas. No software, as amostras foram salvas como arquivos WAVE de um canal. A taxa de amostragem utilizada foi de 44100Hz, e cada amostra tem o tamanho de 16 bits, o equivalente à qualidade de CD. As amostras são codificadas no formato PCM.

Para que os módulos de hardware desenvolvidos no projeto utilizassem como entrada os arquivos de áudio, foram desenvolvidos módulos em linguagem *SystemVerilog* para a conversão dos arquivos WAVE em vetores de teste de 16 bits. Em

uma etapa inicial, os vetores convertidos foram salvos em novos arquivos de texto, porém eles podem ser acoplados aos módulos do projeto para fazer a conversão durante a simulação, retirando a necessidade de arquivos intermediários.

Os módulos foram desenvolvidos a partir da especificação do formato RIFF [16], que é utilizado para escrita e leitura de arquivos de áudio WAVE. Foi necessária a consulta a *websites* [17] de desenvolvimento de software que trazem explicações mais detalhadas dos campos incluídos na especificação.

A Figura 6, encontrada na referência [17], ilustra os campos que constam no cabeçalho de arquivos WAVE em seu formato canônico:

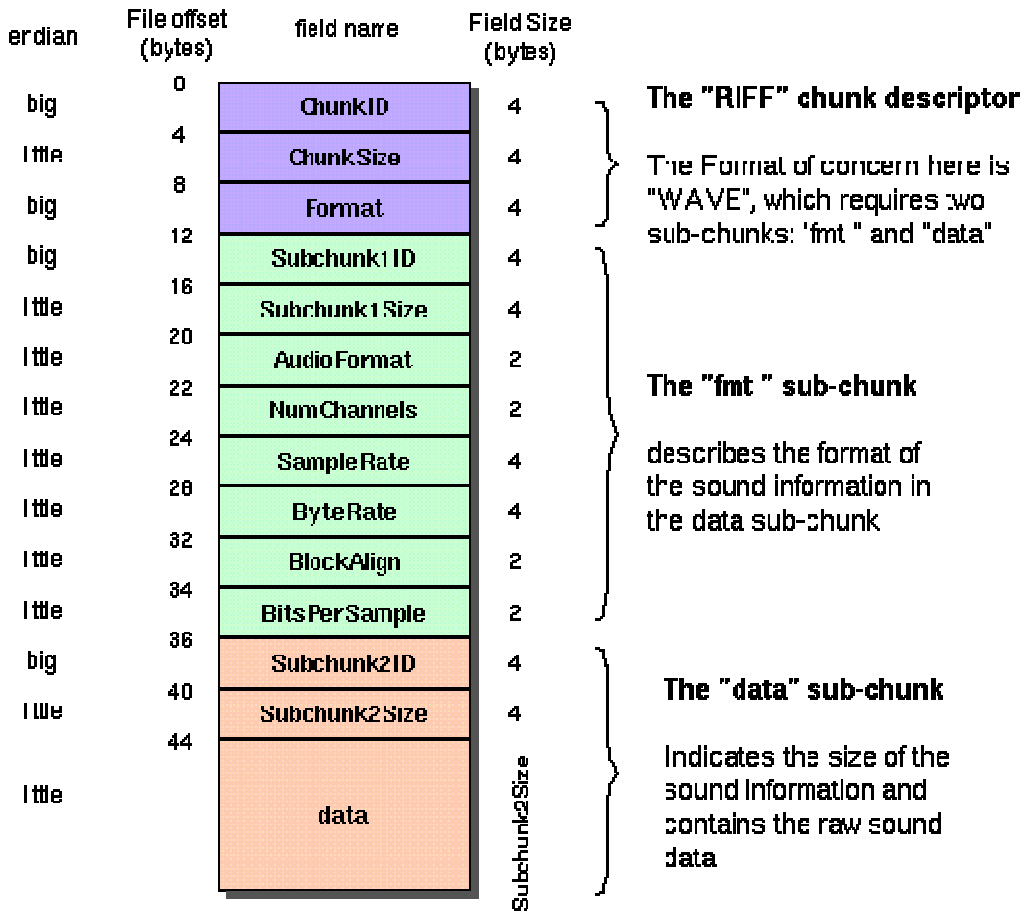


FIGURA 6. CABEÇALHO DO FORMATO DE ARQUIVO WAVE

3.2.1 Módulos de conversão

A descrição em linguagem *SystemVerilog* do módulo de conversão de arquivos de áudio em arquivos de texto resultou em um arquivo de 173 linhas. A Figura 7 mostra a arquitetura do módulo obtida no simulador *IES*. Pode-se ver na figura que os dados constantes no cabeçalho do formato WAVE canônico são extraídos do arquivo.

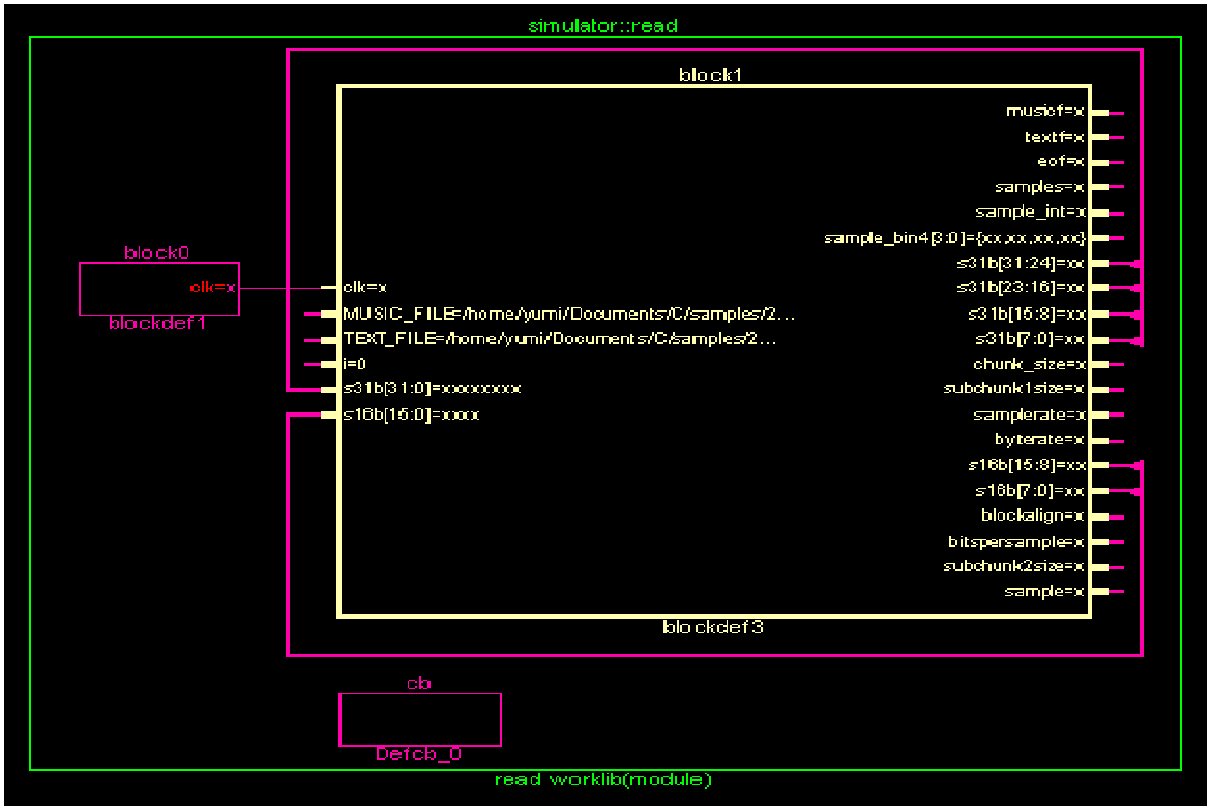


FIGURA 7. MÓDULO CONVERSOR ÁUDIO WAVE -> ARQUIVO DE TEXTO

O módulo que perfaz a operação contrária foi descrito em um arquivo de 240 linhas. Devem-se definir no momento da simulação os parâmetros do arquivo de áudio de saída.

Com a implementação dos dois módulos pode-se utilizar arquivos de música para teste dos módulos do projeto. Além disso, o resultado da simulação pode ser ouvido em *players* digitais. Embora o resultado audível não seja utilizado para avaliação dos

resultados da operação dos módulos, ele permite que eles sejam demonstrados com mais facilidade.

3.2.2 Automatização da conversão

Para a conversão de diversos arquivos de áudio de forma seqüencial, o *IES* foi automatizado com o uso de *script* escrito em linguagem *Perl*. O script executa a ferramenta variando parâmetros dos módulos.

3.3 DESENVOLVIMENTO DO MODELO DE REFERÊNCIA

A primeira versão do modelo implementa o filtro Wiener com comprimento de filtro (número de *taps*) igual a 10. A primeira versão atenuou corretamente ruídos com freqüência de até 440Hz, sendo ineficiente para ruídos de maior freqüência.

Após cerca de um mês de refinamento do algoritmo e topologias utilizadas, foi definida a versão final do modelo de referência, que implementa as características funcionais que o projeto do circuito digital do fone deve apresentar.

O modelo definido processa sinais de áudio amostrados a uma freqüência de 96KHz. Os dados de entrada são de 15 bits, e os dados de saída são de 16 bits. O comprimento de filtro permaneceu igual a dez, e internamente os cálculos são executados com registradores de 28 bits.

3.3.1 Topologia utilizada

A Figura 8 mostra a topologia implementada no modelo de referência:

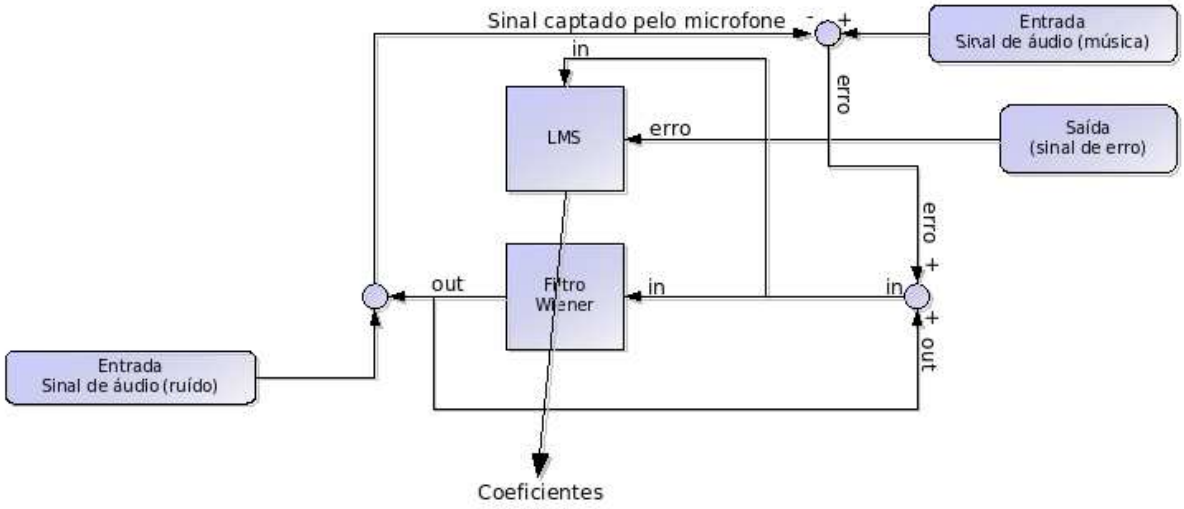


FIGURA 8. TOPOLOGIA INTERNA DO MODELO

O objetivo desta topologia foi simular a topologia esperada durante a operação do chip, mostrada na Figura 9.

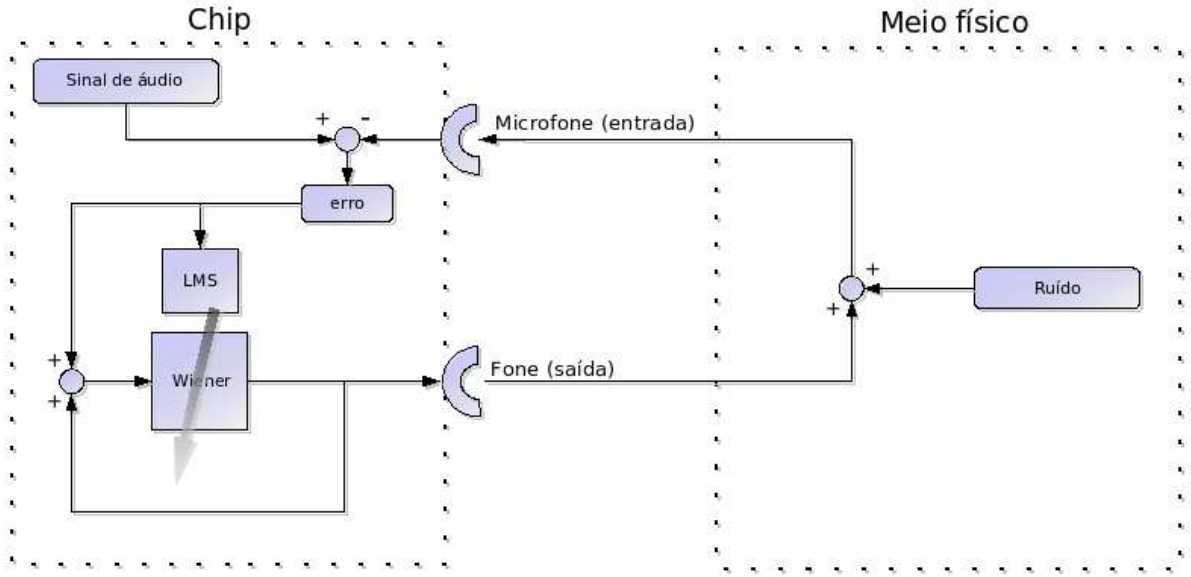


FIGURA 9. TOPOLOGIA DO FONE ANTI-RUÍDO

4 VERIFICAÇÃO FUNCIONAL

O trabalho de verificação funcional [18] de circuitos digitais (a partir deste ponto, chamados de *designs*) tem como principais objetivos:

- *Completeness*: Assegurar que o objetivo do *design* é implementado no produto final
- *Correctness*: Assegurar que toda a implementação está contida na especificação do *design*

Em um caso ideal onde a especificação pudesse ser automaticamente implementada, com total garantia de isenção de falhas, o resultado seria correto por definição. Porém, o que ocorre é que diversos agentes estão envolvidos no processo, como arquitetos de sistema, engenheiros de produto, projetistas e engenheiros de verificação. Além disso, durante a execução do projeto de um *design*, é gerada uma grande quantidade de resultados intermediários, incluindo modelos e documentação.

O papel da verificação funcional é assegurar, em cada etapa, a convergência do circuito integrado em seu objetivo, atendendo à especificação funcional e limites de tempo e recursos. Atualmente, os projetos de verificação são implementados de forma paralela à implementação do *design*. A adoção e acompanhamento de planos de verificação têm papel fundamental na finalização de um projeto.

Um projeto típico de verificação funcional aplica conceitos de engenharia e gestão, e envolve os seguintes agentes:

- Engenheiros de diversas fases do projeto: Especificação, implementação, software, hardware, entre outros.
- Múltiplos subprodutos do projeto: Projetos muito complexos são divididos em módulos menores, cada qual com sua própria especificação, cronograma, documentação, e controle de versões.
- Múltipla documentação e documentação não relacionada diretamente ao produto final: Conceitos, requerimentos, planos de verificação, implementações, algoritmos matemáticos, formatos de arquivos.
- Dependências entre os times: Times independentes necessitam de uma especificação única e completa para que não insiram falhas no produto final.
- Ferramentas comuns de planejamento e acompanhamento de projetos: Tabelas de *Gantt* para acompanhamento de tarefas e recursos, abordagem iterativa de planejamento, gerenciamento e execução.

A construção de um projeto de verificação funcional tem início a partir da coleta de informações sobre a demanda das áreas envolvidas e dos marcos do cronograma do projeto (versões intermediárias prontas, definição de formato de entrada e saída, por exemplo).

Entre as informações essenciais que devem constar em um plano de verificação, é possível citar as seguintes:

- Características do produto que devem ser verificadas para cada marco específico
- Cenários e condições de operação do produto

- Características que devem estar implementadas em determinado prazo durante a implementação do projeto

A Figura 10 mostra o fluxo da elaboração e implementação de um projeto de verificação funcional:

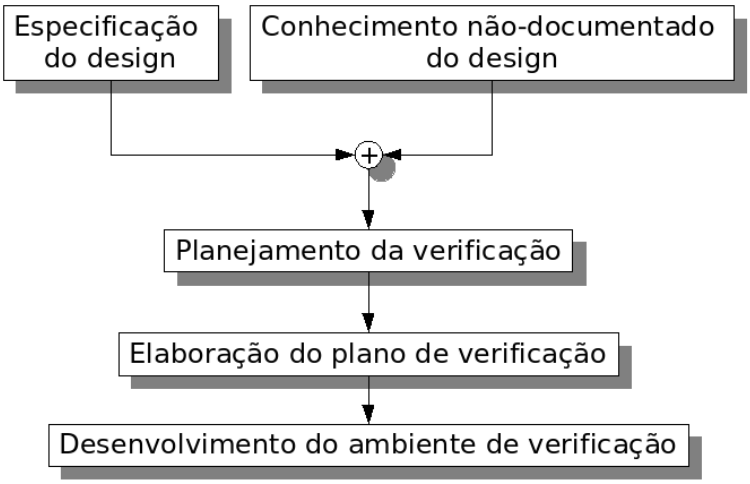


FIGURA 10. FLUXO DE PROJETO DE VERIFICAÇÃO FUNCIONAL.

Conforme a Figura 10, inicia-se coletando toda a documentação possível que tenha relação com o projeto que será desenvolvido. A partir desta documentação, e tendo a experiência não-documentada do tipo de *design* a ser implementado, são definidos quais aspectos da especificação funcional serão incluídos no plano de verificação.

4.1 METODOLOGIA UTILIZADA

Metodologia de verificação é o conjunto de práticas seguidas e técnicas utilizadas para se realizar a verificação funcional de um projeto. A escolha e adoção rigorosa de uma metodologia são fundamentais em projetos grandes, pois asseguram a coerência do trabalho de verificação e documentação das equipes do projeto.

Em geral, a adoção de uma metodologia requer o uso de ferramentas que ofereçam suporte a ela. Devido a isso, os grandes fabricantes de ferramentas de verificação funcional oferecem suporte para metodologias suportadas por suas ferramentas. Da mesma forma, fabricantes também podem se unir para dar suporte a uma única metodologia de acordo com seus interesses. Como exemplos, é possível citar o suporte oferecido pela empresa *Synopsys* para a metodologia *Verification Methodology Management* (VMM) [19], recentemente tornada de código livre; e o recente lançamento da metodologia *Open Verification Methodology* (OVM) [20], também de código livre e amplamente divulgada pelas empresas *Cadence Design Systems* e *Mentor Graphics*. A OVM é uma fusão das metodologias criadas anteriormente por cada fabricante, respectivamente *Coverage-Driven Verification* e *Advanced Verification Methodology*.

O projeto de verificação funcional do processador anti-ruído foi baseado na metodologia *Coverage-Driven Verification*. Ela é baseada em três pilares: geração de dados aleatórios, checagem e cobertura funcional.

4.1.1 Geração aleatória

No momento da especificação de um circuito integrado, é esperado que sejam previstos todos os seus possíveis cenários de operação e seu comportamento para cada um deles. Porém, nem sempre os autores da especificação têm compreensão suficiente do projeto naquele momento para que toda a informação necessária seja incluída. Devido a isso, a verificação funcional deve ser capaz de estimular o *design* também com cenários não incluídos na especificação, que podem trazer à tona comportamentos inesperados.

Assim, é necessário simular o *design* tanto com cenários previstos na especificação quanto com cenários não-previstos (chamados deste ponto em diante de *corner cases*). Idealmente o *design* seria simulado com todos os cenários e dados possíveis. Porém, isto não é possível por inviabilizar o cronograma do projeto. Assim, é necessário priorizar os cenários que serão testados, porém mantendo-se a aleatoriedade dos dados. A Figura 11 ilustra como o os testes podem ser escolhidos no espaço de geração de três formas: testes direcionados, testes aleatórios restritos e testes aleatórios direcionados:

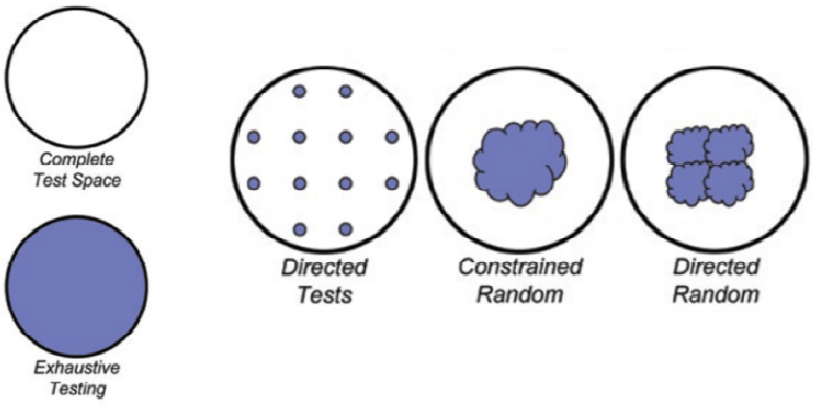


FIGURA 11. FORMAS DE ESCOLHA DOS CENÁRIOS TESTADOS

As três formas possuem vantagens e desvantagens. Testes direcionados (*directed tests*) validam de forma mais rápida a operação esperada do *design* nos cenários de operação esperados, porém não traz à tona aspectos desconhecidos do projeto. Testes aleatórios restritos (*constrained random tests*) estimulam o *design* exaustivamente nos cenários mais prováveis de operação, porém podem deixar sem serem testados casos com pouca probabilidade de ocorrência. Testes aleatórios direcionados (*directed random tests*) geram dados significativos sobre o comportamento do *design* em *corner cases*, porém estes cenários não costumam ser conhecidos inicialmente, portanto só recebendo prioridade ao final do processo de verificação.

Uma abordagem que permite o aproveitamento das vantagens de cada forma de escolha de testes é sua utilização em diferentes fases do processo de verificação. Durante o desenvolvimento do ambiente de verificação, convém a utilização de testes direcionados, para verificar os modos básicos de funcionamento do *design* e também corrigir eventuais defeitos no próprio ambiente de verificação. Após esta fase, utiliza-se a geração aleatória restrita para se executar um grande volume de testes, e garantir que a maior parte da especificação seja verificada. Ao final do processo de verificação funcional, tendo como limite o tempo restante no projeto, utiliza-se geração aleatória direcionada para a verificação de *corner cases*.

A relação entre a linha do tempo da execução de um projeto de verificação funcional e o uso dos três modos é mostrada na Figura 12. O eixo X indica o tempo e o eixo Y indica o espaço de geração aleatória de testes.

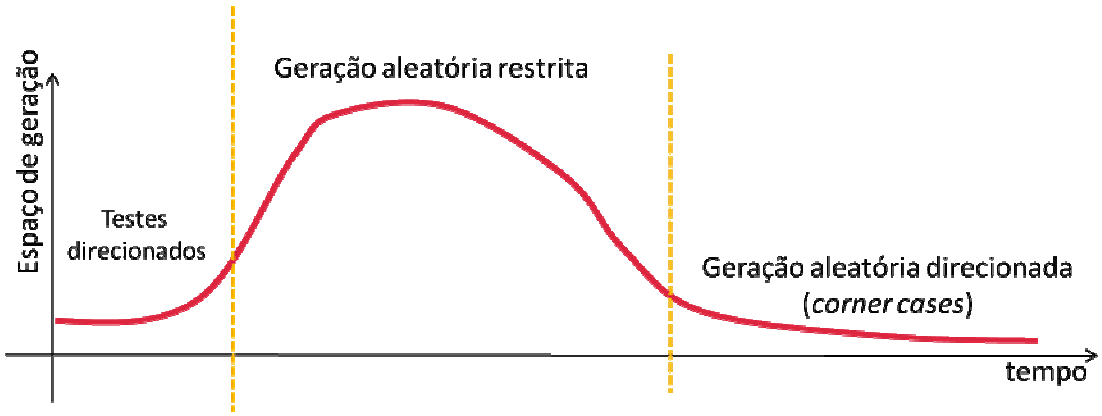


FIGURA 12. ESCOLHA DOS MODOS DE GERAÇÃO DE TESTES AO LONGO DO TEMPO

4.1.2 Checagem

É pouco proveitoso simular o *design* exaustivamente se não for possível garantir que em cada teste a operação ocorreu da forma esperada e produziu resultados corretos. Assim, é de extrema importância que o ambiente de verificação realize pelo

menos as checagens de resultado produzido e conformidade ao protocolo durante a simulação. Outras checagens, como funcionamento esperado de máquinas de estado ou de ocorrência de determinados eventos e *corner cases*, são muito úteis para detectar *bugs* de forma rápida e precisa.

A checagem pode ser feita através de asserções, que consistem em sentenças referentes a dados ou seqüências, e que podem incluir informações temporais. As asserções sempre são relacionadas a algum evento de amostragem, no momento do qual seu conteúdo é analisado. Caso a informação expressa na sentença seja falsa, o ambiente pode responder produzindo um erro, um aviso ou uma informação.

O uso de asserções para verificação de um *design* demanda uma boa compreensão do mesmo e permite que erros sejam localizados e corrigidos rapidamente. Além disso, asserções podem ser reaproveitadas em outros blocos dentro do mesmo projeto ou mesmo em projetos diferentes.

4.1.3 Cobertura funcional

Geração aleatória e checagem podem ser utilizadas para se verificar um *design* até que todos os possíveis cenários sejam simulados, porém o tempo necessário para tal não costuma ser factível. Por isso, utiliza-se o plano de verificação para planejamento e priorização. A mensuração da execução do plano de verificação é feita através da cobertura funcional, que demonstra o quanto das características funcionais de um bloco ou de um sistema inteiro já foram testados durante a fase de simulação. A definição das faixas de valores para cobertura funcional deve constar no plano de verificação.

4.2 ESPECIFICAÇÃO DO PROJETO

A topologia do circuito digital do fone anti-ruído contém oito blocos. A Figura 13 mostra um documento do editor de planilhas *OpenOffice Calc* contendo as especificações dos blocos do IP02-ANC.

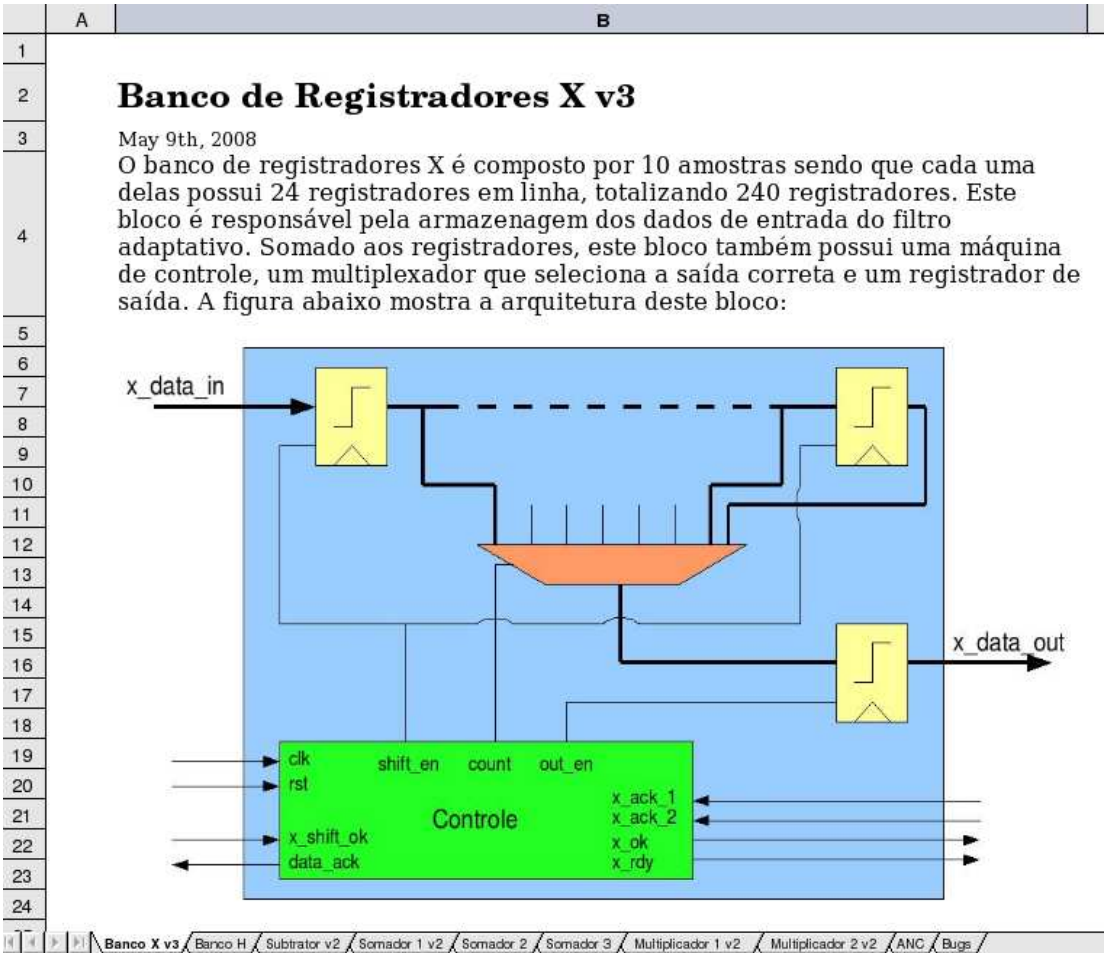


FIGURA 13. ESPECIFICAÇÕES DOS BLOCOS DO PROJETO

4.3 *TESTBENCHES* PARA OS BLOCOS

Após o término dos trabalhos com o modelo de referência, e à medida que as primeiras versões das descrições dos blocos foram feitas, foram desenvolvidos *testbenches* rápidos para os blocos descritos. O sistema completo é composto pelos seguintes blocos:

- Subtrator
- Somador 1
- Somador 2
- Multiplicador 1
- Multiplicador 2
- Multiplicador 3
- Banco de registradores X
- Banco de registradores H

O objetivo de cada *testbench* foi verificar a conformidade com a especificação, os modos de funcionamento e o resultado da operação executada. Para ilustrar as facilidades de verificação funcional contidas na linguagem *SystemVerilog*, a seguir são apresentadas as construções usadas nos *testbenches* dos blocos do sistema. Uma vez que os *testbenches* dos blocos somadores e multiplicadores, e dos bancos de memória são semelhantes entre si, apenas um exemplo de cada será listado. A partir deste ponto, o texto irá referir-se ao módulo sob verificação pela sigla de DUT (*design under test*, acrônimo criado anteriormente à popularização da verificação funcional).

4.3.1 Subtrator

O subtrator é o bloco de funcionamento mais simples do sistema. Devido a isso, o desenvolvimento do seu *testbench* será explicado em mais detalhes do que os outros blocos.

O desenvolvimento do *testbench* inicia-se a partir da descrição da operação executada pelo DUT (no caso, “subtração dos dados de entrada, em complemento de dois”) e pela especificação de sua interface externa, mostrada na Figura 14.

Pino	Direção	Largura	Ativo	Descrição
clk	input	1	rising	Relógio do sistema
rst	input	1	low	Reset síncrono do sistema
sub_s1_in	input	24	-	Entrada de dados no subtrando
sub_s2_in	input	24	-	Entrada de dados no subtrador
sub_s1_ok	input	1	low	Quando ativo, indica que há dado na entrada 1 do subtrator
sub_s2_ok	input	1	low	Quando ativo, indica que há dado na entrada 2 do subtrator
sub_rd_1	input	1	low	Quando ativo, indica que o dado na saída do bloco foi registrado pelo próximo correspondente.
sub_rd_2	input	1	low	Quando ativo, indica que o dado na saída do bloco foi registrado pelo próximo correspondente.
sub_data_out	output	24	-	Saída de dados do subtrador
sub_rdy_1	output	1	low	Quando ativo, indica ao próximo bloco correspondente que já existe dado válido na saída.
sub_rdy_2	output	1	low	Quando ativo, indica ao próximo bloco correspondente que já existe dado válido na saída.
data_ack	output	1	low	Quando ativo, indica para o bloco anterior que os dados foram armazenados corretamente.

FIGURA 14. PINOS DE ENTRADA E SAÍDA DO BLOCO SUBTRATOR

Como se pode observar, esta especificação lista os pinos de entrada e saída, sua largura em bits, seu valor ativo e seu propósito.

Outro dado indispensável para o desenvolvimento do *testbench* é o diagrama de formas de onda, mostrado na Figura 15.

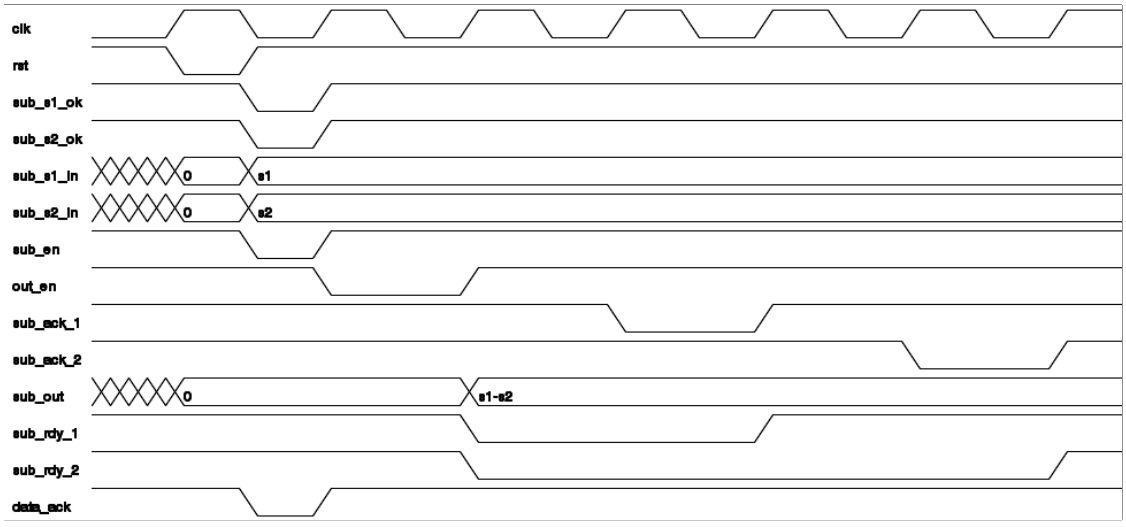


FIGURA 15. DIAGRAMA DE FORMAS DE ONDA DO BLOCO SUBTRATOR

De acordo com a figura, no momento que os sinais de entrada *sub_s1_ok* e *sub_s2_ok* se tornam ativos, a saída *data_ack* é ativada no mesmo ciclo e, após certo

tempo, a saída *sub_out* vai receber a diferença entre os sinais *sub_s1_in* e *sub_s2_in*. Além disso, os sinais de saída *sub_rdy_1* e *sub_rdy_2* se tornam ativos até que os sinais de entrada *sub_ack_1* e *sub_ack_2* se tornem ativos. Nesse momento o subtrator retorna para seu estado inicial.

O código do *testbench* do bloco subtrator é apresentado na Figura 16.

```
module subtrator_tb;
	logic clk, rst, sub_s1_ok, sub_s2_ok, sub_ack_1, sub_ack_2, sub_rdy_1, sub_rdy_2, data_ack;
	logic [27:0] sub_s1_in, sub_s2_in, sub_data_out;
	logic signed [27:0] data1, data2, res;

	integer test_counter = 1,ok, random_delay, random_delay_sub_s1_ok, random_delay_sub_s2_ok;
	event op_end;

	assign res = data1 - data2;
	assign sub_s1_in = data1;
	assign sub_s2_in = data2;

	default clocking cb @(posedge clk);
		output sub_s1_ok,sub_s2_ok;
	endclocking

	subtrator subtrator(.*);

	initial
	begin
		clk = '0;
		forever #1 clk = ~clk;
	end

	always @(posedge clk)
	begin
		sub_s1_ok = '1;
		sub_s2_ok = '1;
		sub_ack_1 = '1;
		sub_ack_2 = '1;
		rst = '1;
		##1;
		rst = '0;
		##1;
		rst = '1;
		##5;
		while (test_counter != 0) begin
			ok = randomize(data1);
			ok = randomize(data2);
			cb.sub_s1_ok <= '0;
			cb.sub_s2_ok <= '0;
			wait(!data_ack);
			##1;
			cb.sub_s1_ok <= '1;
			cb.sub_s2_ok <= '1;
			##5;
			sub_ack_1 = '0;
			##10;
			sub_ack_1 = '1;
			##1;
			sub_ack_2 = '0;
			##1;
			sub_ack_2 = '1;
			##1;
			-> op_end;
			test_counter--;
		end
		$stop();
	end
	checagem: assert property (@(op_end) res == sub_out);
endmodule
```

FIGURA 16. *TESTBENCH* DO BLOCO SUBTRATOR

A seguir são apresentados comentários para construções de verificação contidas no código:

- Declarações de tipos *logic* (linhas 2 a 4): *Logic* é um tipo de dado existente na linguagem *SystemVerilog* que possui quatro estados: 1, 0, *X* e *Z*. São declarados sinais para estímulo e coleta de dados do DUT, de acordo com os sinais constantes na especificação.
- Instanciamento do bloco com *dot star connection* (*".*"*, linha 17) : Em *SystemVerilog* módulos podem ser instanciados sem a declaração da conexão de seus dados de entrada e saída. O simulador resolve as conexões no momento da elaboração automaticamente, desde que os sinais tenham o mesmo nome.
- *Clocking blocks* (linhas 13 a 15): *Clocking blocks* são construções que permitem que se estabeleça um ou mais sinais de referência para o módulo, sendo um deles o padrão do sistema (marcado pela palavra-chave *default*). Também é possível estabelecer o momento de amostragem de sinais de entrada ou modificação de sinais de saída em relação ao sinal de referência. No caso do subtrator, estabelece-se o sinal *clk* como sinal de referência, e determina-se que as entradas *sub_s1_ok* e *sub_s2_ok* serão amostradas um pico segundo antes da alteração do sinal *clk*. Este *clocking block* permite que as condições de simulação sejam a mesmas tanto durante a simulação individual do subtrator quanto durante a simulação do sistema inteiro.

- Geração aleatória de dados (linhas 38 e 39): A função *randomize* gera dados aleatórios para os sinais em seu argumento. Podem ser incluídas restrições para a geração.
- Checagem da operação executada (linha 60): É declarada uma asserção que determina que, no momento do evento *op_end*, emitido após o término da operação do bloco, o valor de saída do bloco deve ser igual a o valor esperado pelo *testbench*.

O *testbench* do bloco subtrator executa testes com valores aleatórios, estimulando o DUT de acordo com as formas de onda descritas na especificação, e verificando o valor de saída após cada teste.

4.3.2 Somadores e multiplicadores

O funcionamento dos blocos somadores e multiplicadores é bastante semelhante ao do bloco subtrator. Porém, no somador 1 existem dois possíveis modos de operação: soma das entradas 0 e 2 ou soma das entradas 1 e 2. Para que a simulação teste aleatoriamente os dois modos foi utilizada a construção *randsequence*, onde um entre vários cenários é escolhido pela ferramenta no momento da simulação (pode-se atribuir probabilidades para os modos). A Figura 17 mostra o trecho do código do *testbench* contendo a declaração dos cenários.

Durante a simulação, a cada passagem pelo laço *for*, a ferramenta irá escolher um dos blocos de código *s1_s1* ou *s0_s2* para executar.

```
40 for (test_counter=0;test_counter<300;test_counter++) begin
41.         ##20;
42.         randsequence (main)
43.         .       main : s1_s2 | s0_s2;
44.         .       s1_s2 : {
45.         .       .       ok = randomize(add1_s1_in);
46.         .       .       ok = randomize(add1_s2_in);
47.         .       .       cb.add1_s1_ok <= '0;
48.         .       .       cb.add1_s2_ok <= '0;
49.         .       .       ##1;
50.         .       .       cb.add1_s1_ok <= '1;
51.         .       .       cb.add1_s2_ok <= '1;
52.         .       .       wait(!add1_rdy);
53.         .       .       result = add1_s1_in+add1_s2_in;
54.         .       .       ##1;
55.         .       .       add1_ack = '0;
56.         .       .       ##1;
57.         .       .       add1_ack = '1;
58.         .       .       ->> check;
59.         .       };
60.         .       s0_s2 : {
61.         .       .       ok = randomize(add1_s1_in);
62.         .       .       cb.add1_s0_ok <= '0;
63.         .       .       cb.add1_s2_ok <= '0;
64.         .       .       ##1;
65.         .       .       cb.add1_s0_ok <= '1;
66.         .       .       cb.add1_s2_ok <= '1;
67.         .       .       wait(!add1_rdy);
68.         .       .       result = add1_s1_in;
69.         .       .       ##1;
70.         .       .       add1_ack = '0;
71.         .       .       ##1;
72.         .       .       add1_ack = '1;
73.         .       .       ->> check;
74.         .       };
75.         endsequence
76 end
```

FIGURA 17. DECLARAÇÃO DE CENÁRIOS UTILIZANDO *RANDSEQUENCE*

4.3.3 Bancos de registradores

Nos *testbenches* desenvolvido para os bancos de registradores X e H, devido à maior complexidade da operação dos DUTs, a realização da checagem se torna um pouco mais sofisticada.

O banco X armazena os dez coeficientes do filtro, e após avisa o próximo bloco que os coeficientes foram armazenados. Isso significa que, ao ser estimulado

continuamente da mesma forma, a cada dez operações a configuração dos sinais de saída do DUT será diferente.

Para a checagem do banco de registradores X foi criada uma classe em *SystemVerilog*, contendo métodos correspondentes aos modos de operação do bloco. O trecho inicial do *testbench* do bloco, contendo a declaração da classe e de sua instância, é mostrado na Figura 18.

```
1  module banco_reg_tb;
2.         class dummy;
3.         .         logic [23:0] data [9:0];
4.         .         int counter;
5.         .         function new();
6.         .         .         int i;
7.         .         .         for (i=0;i<10;i++) data[i] = '0;
8.         .         .         counter = 0;
9.         .         endfunction
10.        .         task shift (input logic [23:0] data_in);
11.        .         .         int i;
12.        .         .         for (i=9;i>0;i--) begin
13.        .         .         .         data[i] = data[i-1];
14.        .         .         end
15.        .         .         data[0] = data_in;
16.        .         endtask
17.        .         task get (output logic [23:0] data_out);
18.        .         .         counter++;
19.        .         .         if (counter > 10) counter = 1;
20.        .         .         data_out = data[counter-1];
21.        .         endtask
22.        endclass
23.
24.        // block inputs:
25.        logic clk, rst, x_shift_ok,  x_ack_1, x_ack_2;
26.        logic [23:0] x_data_in;
27.        // block outputs:
28.        logic x_rdy, x_ok, data_ack;
29.        logic [23:0] x_data_out;
30.        // other:
31.        logic spmn_go;
32.        logic [23:0] res;
33.
34.        integer ok,test_counter,data_counter;
35.
36.        event check;
37.
38.        x_registers x_registers(.*);
39.
40.        dummy dummy1 = new();
```

FIGURA 18. DEMONSTRAÇÃO DO USO DE CLASSES PARA CHECAGEM

4.4 DESENVOLVIMENTO DE AGENTES PARA OS BLOCOS

Agentes estão entre os componentes fundamentais da metodologia *Coverage-driven Verification*. Agentes são módulos do ambiente de verificação desenvolvidos para um DUT, com construções para estímulo e monitoramento. Os parâmetros dos agentes podem ser alterados, e seus componentes internos podem ser instanciados separadamente, possibilitando uma infinidade de configurações de simulação com pouca alteração de código. Além disso, uma vez desenvolvido um agente para um DUT específico, boa parte de seu código pode ser reaproveitado no momento do desenvolvimento de um DUT semelhante.

4.4.1 Topologia

Um agente é composto por um módulo monitor e um *driver*. O *driver* é um gerador de seqüências aleatórias de estímulo de acordo com as especificações do DUT para o qual o agente está sendo desenvolvido. Já o monitor é um módulo que monitora os estímulos recebidos pelo DUT, seu comportamento e seus sinais de saída. No monitor são incluídas declarações de checagem e cobertura funcional.

Agentes podem ser instanciados em dois modos: ativo ou passivo. No modo ativo o *driver* e o monitor são ativados, assim, enquanto o *driver* gera o estímulo para o DUT o monitor coleta e analisa os dados. No modo passivo, o *driver* é desativado, e espera-se que o estímulo do bloco seja enviado por alguma outra entidade durante a simulação. No modo passivo, o monitor continua funcionando normalmente. O modo de funcionamento é definido através do parâmetro *modo*, no momento em que o agente é instanciado. Outro parâmetro do agente é o número de testes executados quando em modo ativo. A Figura 19 ilustra a topologia descrita.

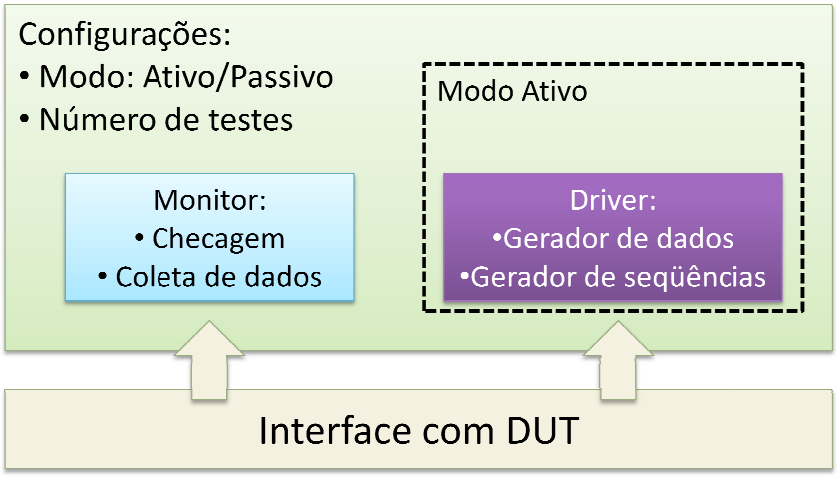


FIGURA 19. TOPOLOGIA DE UM AGENTE

Para se executar uma simulação de um DUT isolado, basta instanciar o DUT e seu agente, configurado no modo ativo, em um módulo de topo. Esta topologia de simulação é mostrada na Figura 20.

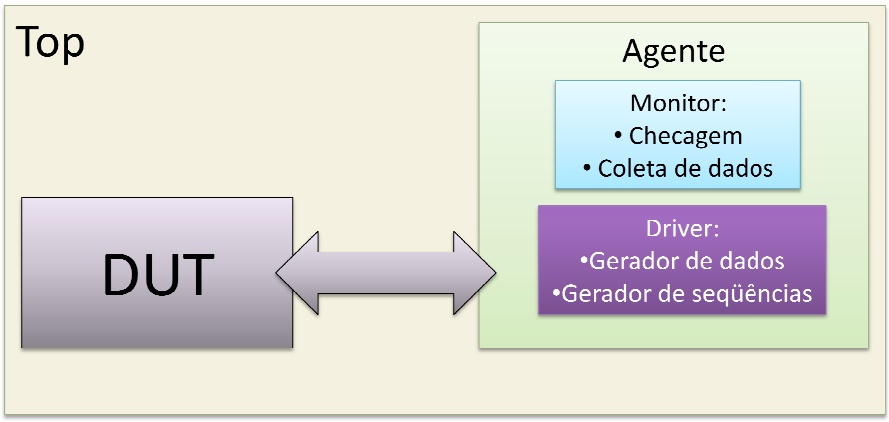


FIGURA 20. TOPOLOGIA UTILIZANDO O AGENTE EM MODO ATIVO

Como exemplo de agente desenvolvido no trabalho, os códigos do agente do bloco somador 1 serão mostrados nas figuras a seguir. Neste caso, o monitor foi descrito em um arquivo separado. No código do agente, mostrado na Figura 21, o monitor é instanciado na linha 20, e o *driver* está contido no bloco *always*, cuja descrição vai das linhas 22 a 68.

```
module adder1_agent(
        // inputs
        output logic [27:0] add1_s1_in, add1_s2_in,
        input logic clk, rst,
        output logic add1_s0_ok, add1_s1_ok, add1_s2_ok, add1_ack,
        //outputs
        input logic add1_rdy, data_ack,
        input logic [27:0] add1_data_out);

        parameter active_mode = 1'b0;
        parameter test_counter = 300;

        // other
        integer i,ok;

        default clocking cb @(posedge clk);
                output #1ps add1_s0_ok,add1_s1_ok,add1_s2_ok;
        endclocking

        adder1_monitor adder1_monitor(.*);

        always @(posedge clk)
        begin
                if (active_mode == 1'b1) begin
                        add1_s1_in = 28'b0;
                        add1_s2_in = 28'b0;
                        add1_s0_ok = '1;
                        add1_s1_ok = '1;
                        add1_s2_ok = '1;
                        add1_ack = '1;
                        ##50;
                        for (i=0;i<test_counter;i++) begin
                                ##20;
                                randsequence (main)
                                        main : s1_s2 | s0_s2;
                                        s1_s2 : {
                                                ok = randomize(add1_s1_in);
                                                ok = randomize(add1_s2_in);
                                                cb.add1_s1_ok <= '0;
                                                cb.add1_s2_ok <= '0;
                                                ##1;
                                                cb.add1_s1_ok <= '1;
                                                cb.add1_s2_ok <= '1;
                                                wait(!add1_rdy);
                                                ##1;
                                                add1_ack = '0;
                                                ##1;
                                                add1_ack = '1;
                                        };
                                        s0_s2 : {
                                                ok = randomize(add1_s1_in);
                                                cb.add1_s0_ok <= '0;
                                                cb.add1_s2_ok <= '0;
                                                ##1;
                                                cb.add1_s0_ok <= '1;
                                                cb.add1_s2_ok <= '1;
                                                wait(!add1_rdy);
                                                ##1;
                                                add1_ack = '0;
                                                ##1;
                                                add1_ack = '1;
                                        };
                                endsequence
                        end
                        ##50;
                        $stop();
                end
        end
endmodule
```

FIGURA 21. AGENTE DESENVOLVIDO PARA O BLOCO SOMADOR 1

Já na descrição do monitor, mostrada na Figura 22, as linhas 14 a 24 contêm construções para a verificação de cobertura funcional, enquanto o bloco *always* iniciado na linha 26 realiza monitoramento passivo e checagem.

```
1  module adder1_monitor(
2.         // inputs
3.         input logic [27:0] add1_s1_in, add1_s2_in,
4.         logic clk, rst,
5.         logic add1_s0_ok, add1_s1_ok, add1_s2_ok, add1_ack,
6.         //outputs
7.         logic add1_rdy, data_ack,
8.         logic [27:0] add1_data_out);
9.
10.        // other
11.        logic [27:0] result;
12.        logic op_sign;
13.
14.        covergroup adder1_cg @(posedge clk);
15.        .       S1: coverpoint add1_s1_in
16.        .       {
17.        .       .       bins zero = {28'b0};
18.        .       .       bins low = {[28'b1:28'h7FFFFF]};
19.        .       .       bins high = {[28'h8000000:28'hFFFFFFE]};
20.        .       .       bins one = {28'hFFFFFFF};
21.        .       }
22.        endgroup
23.
24.        adder1_cg adder1_cg1 = new;.
25.
26.        always @(posedge clk)
27.        begin
28.        .       if (rst == '0) begin
29.        .       .       result = '0;
30.        .       end else begin
31.        .       .       $display("adder1 started");
32.        .       .       if ((add1_s1_ok == '0) && (add1_s2_ok == '0)) begin
33.        .       .       .       result = add1_s1_in+add1_s2_in;
34.        .       .       .       if (add1_s2_in[27] == add1_s1_in[27]) begin
35.        .       .       .       .       op_sign = add1_s1_in[27];
36.        .       .       .       .       overflow_somador1 : assert (result[27] == op_sign);
37.        .       .       .       end
38.        .       .       .       wait(!add1_rdy);
39.        .       .       .       checagem_somador1_modo1 : assert (result == add1_data_out);
40.        .       .       end else begin
41.        .       .       .       if ((add1_s0_ok == '0) && (add1_s2_ok == '0)) begin
42.        .       .       .       .       result = add1_s1_in;
43.        .       .       .       .       wait(!add1_rdy);
44.        .       .       .       .       checagem_somador1_modo2 : assert (result == add1_data_out);
45.        .       .       .       end
46.        .       .       end
47.        .       .       $display("adder1 finished");
48.        .       end
49.        end
50 endmodule
```

FIGURA 22. MONITOR DESENVOLVIDO PARA O AGENTE DO BLOCO SOMADOR 1

4.5 VERIFICAÇÃO FUNCIONAL DO SISTEMA

A simulação do sistema completo (a partir daqui chamado de *top-level*), contendo todos os blocos descritos na especificação conectados, é o momento onde a verificação funcional se mostra determinante para que um projeto possa ser concluído respeitando-se prazo e qualidade.

Para o desenvolvimento do ambiente de verificação que será usado para simular o *top-level*, recorre-se novamente à especificação. Para a determinação da prioridade em que os cenários devem ser testados, recorre-se ao plano de verificação. É desenvolvido um agente para o sistema, com topologia semelhante aos agentes desenvolvidos para os blocos.

4.5.1 Agente do sistema

No caso do projeto IP02-ANC, não há sentido em se estimular o *top-level* com dados aleatórios, uma vez que se trata de um filtro adaptativo. Dados aleatórios fariam o cálculo divergir e os resultados de saída não seriam significativos. Devido a isso, ao invés de dados aleatórios, o *driver* do agente desenvolvido para o *top-level* obtém de arquivos de texto os estímulos que devem ser enviados ao DUT. Os arquivos contêm amostras adquiridas de arquivos de áudio referentes ao sinal e ao ruído. Estes arquivos são gerados anteriormente através dos módulos descritos na sessão 0 deste trabalho.

O monitor do agente não calcula o resultado que o DUT deve produzir, pois também os obtêm de um arquivo de texto gerado pelo modelo de referência. A checagem é feita comparando-se o resultado emitido pelo DUT com a amostra correspondente do arquivo.

Assim, a checagem do *top-level* consiste em comparar se o resultado produzido pelo DUT é correspondente ao resultado produzido pelo modelo de referência. Uma vez que o modelo de referência foi implementado em total conformidade com a especificação, a verificação da equivalência do *design* com o modelo assegura a conformidade do primeiro.

4.5.2 *Testbench* do sistema

O *testbench* gerado para o sistema é semelhante aos *testbenches* utilizados pelos blocos, ou seja, contêm uma instância da DUT e uma de seu agente configurado em modo ativo. Além destes componentes, pode-se aproveitar uma das principais possibilidades que a metodologia utilizada oferece: a possibilidade de se instanciar os agentes dos blocos internos em modo passivo. Assim, enquanto o agente do *top-level* estimula e monitora a DUT, os estímulos recebidos e operações executadas pelos blocos internos também são monitorados. Com esta configuração, eventuais erros de projeto se tornam facilmente localizáveis, e o tempo necessário para a sua correção diminui drasticamente. Outra vantagem é a obtenção de maior cobertura funcional para os blocos internos.

A Figura 23 ilustra a topologia apresentada, onde são instanciados o DUT e seu agente em modo ativo, e os agentes para blocos internos do DUT são instanciados em modo passivo (ou seja, com seu *driver* desativado).

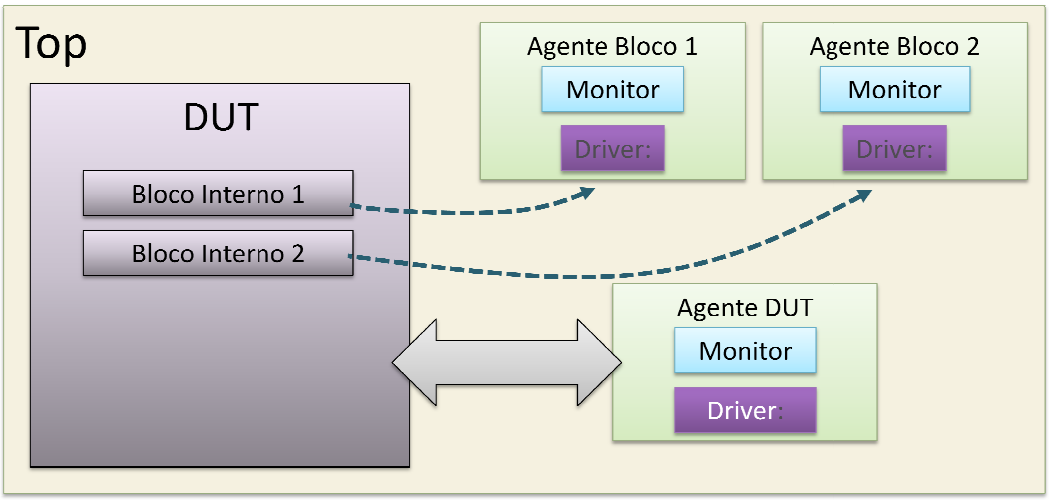


FIGURA 23. TOPOLOGIA DE *TESTBENCH* PARA O SISTEMA

5 RESULTADOS EXPERIMENTAIS

5.1 *TESTBENCHES* E AGENTES DOS BLOCOS

Durante a elaboração dos *testbenches* para os blocos internos, foram detectados nos blocos erros de diversas naturezas:

- Não-conformidade com a especificação para estímulo
- Não-conformidade com a especificação para o resultado
- Erros na especificação ou especificação incompleta
- Erros na operação executada

Nesta fase do projeto, foram detectados, em média, dois erros por dia. O tempo médio de correção dos erros foi de poucos minutos, por se tratarem, em sua grande maioria, de erros pequenos.

A figura a seguir mostra alguns dos erros detectados durante a simulação individual dos blocos:

Numero	Data	Bloco	Erro	Causa	Solução
3	12 de Maio	Banco H	Um ciclo a mais que o especificado	Valor inicial errado na contagem	Correção do código
4	13 de Maio	Somador1	Não resetava corretamente	Erro no código	Correção do código
5	13 de Maio	Somador1	Não iniciava	Erro na especificação	Correção da especificação
6	14 de Maio	Multiplicador1	Resultado errado	Erro no código	Correção do código
7	14 de Maio	Multiplicador2	Sinal inválido na saída	Erro no código	Correção do código
8	15 de Maio	Multiplicador2	Valor errado na saída (checagem de sinal)	Erro no código	Correção do código

FIGURA 24. EXEMPLOS DE ERROS DETECTADOS NOS BLOCOS

5.2 *TESTBENCH* DO SISTEMA

A Figura 25 mostra a topologia interna do *testbench top-level*, adquirida com a opção *schematic tracer* do simulador *IES*. Pode-se observar o DUT (realçado com a linha pontilhada) e os monitores instanciados no mesmo nível hierárquico.

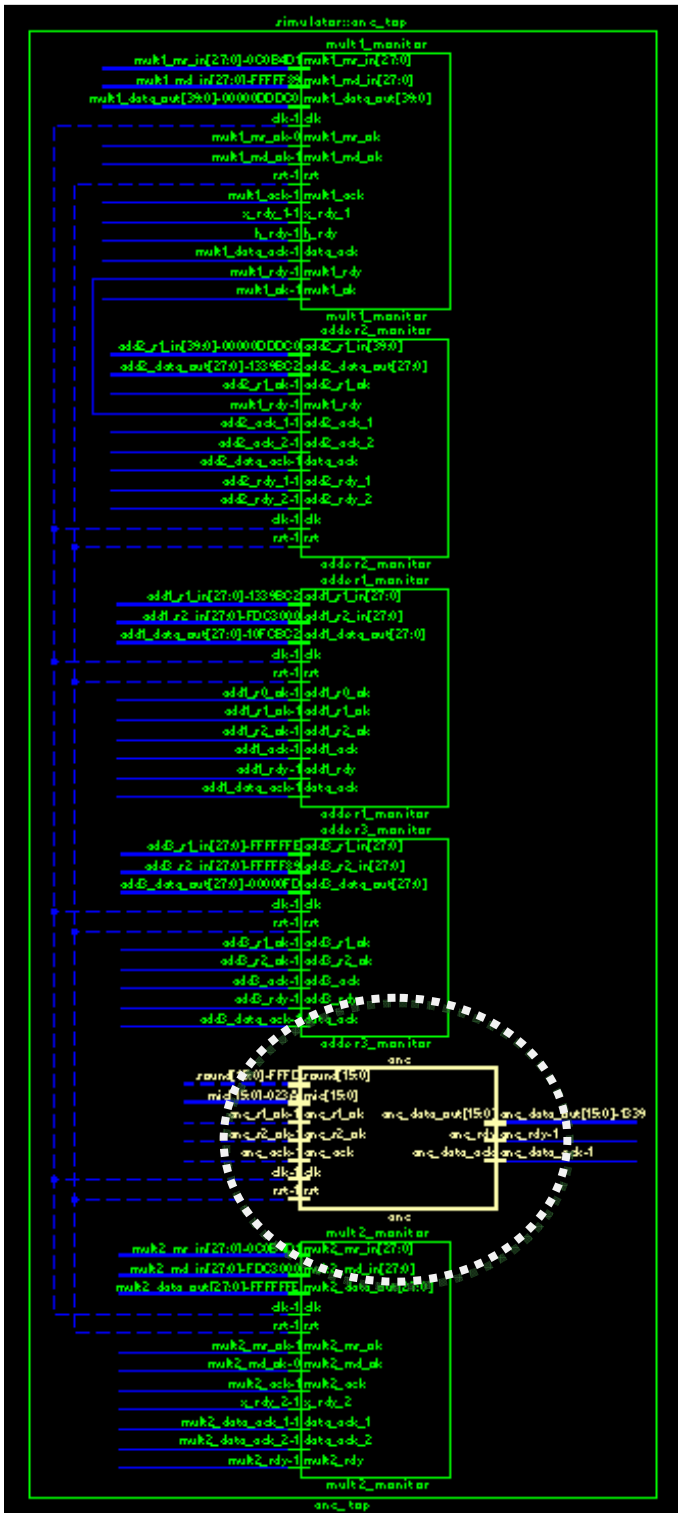


FIGURA 25. TOPOLOGIA DE SIMULAÇÃO DO SISTEMA

Durante a fase de simulação do sistema, o uso de agentes passivos mostrou-se fundamental para entendimento e correção de erros. Há uma clara diferenciação na

natureza dos erros ocorridos no sistema. Nesta fase, o funcionamento dos blocos é correto, porém a operação executada por seu conjunto apresenta problemas. No caso do IP02-ANC, foram incluídos nos monitores checagens de situações que não devem ocorrer, como *overflow* dos somadores e multiplicadores. Através destas checagens foi possível isolar os momentos e locais dos problemas. Pode-se citar como erros detectados no *top-level*:

- Quantidade insuficiente de bits nos registradores internos dos acumuladores
- Erros de sincronização entre os blocos
- Falta de detalhamento na especificação de interfaces dos blocos

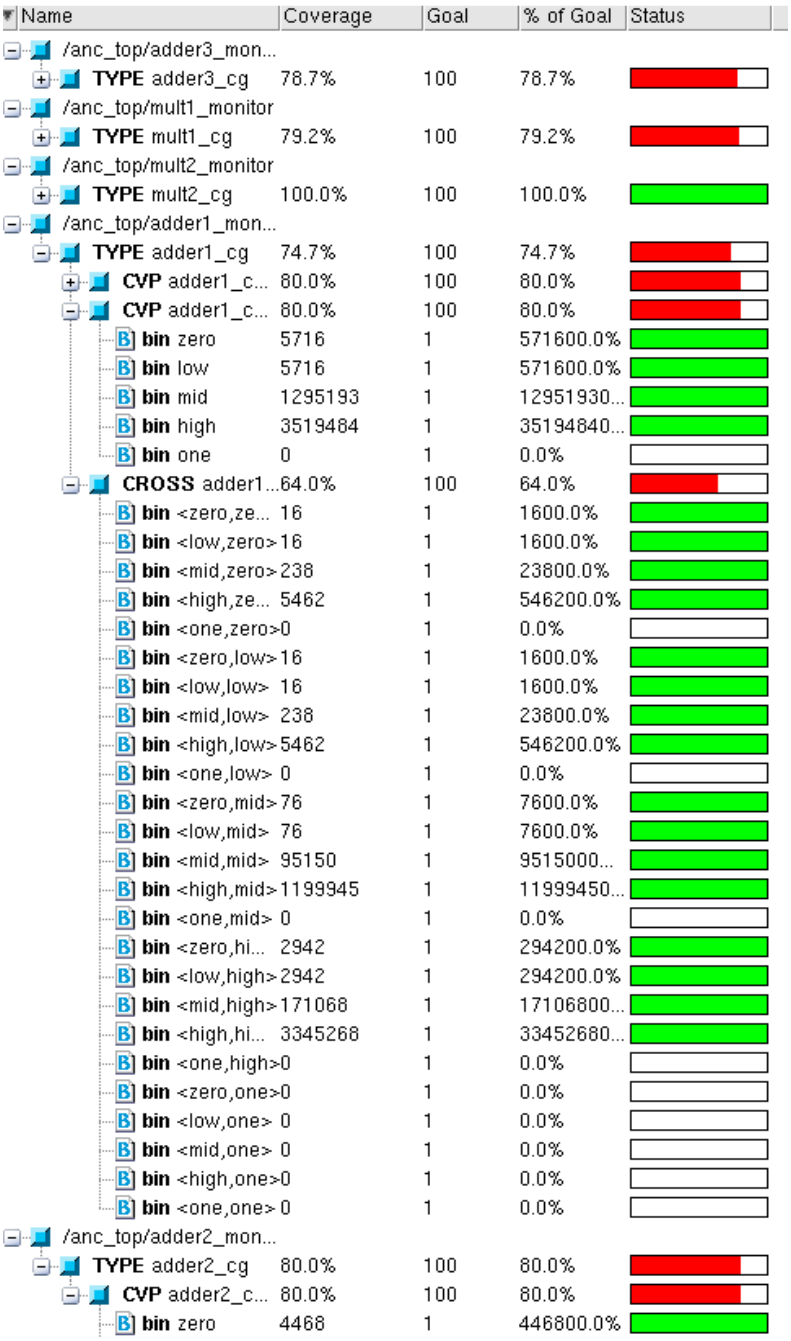
Uma amostra de como o simulador *IES* apresenta para o usuário as declarações de checagem dos blocos internos do projeto IP02-ANC é mostrada na Figura 26. Em momentos em que alguma declaração é avaliada como falsa, a simulação pára e a contagem de falhas da declaração é incrementada.

Assertion Name	Type	Module/Unit	Instance	Checked	Failed Count
overflow_entrada_mult1	assert	overflow_entrada_mult1	anc_top.mult1_monitor.overflow_entrada_mult1	379827	0
checagem_mult1	assert	checagem_mult1	anc_top.mult1_monitor.checagem_mult1	379827	0
overflow_saida_mult1	assert	overflow_saida_mult1	anc_top.mult1_monitor.overflow_saida_mult1	379827	0
overflow_mult2	assert	overflow_mult2	anc_top.mult2_monitor.overflow_mult2	379828	0
checagem_mult2	assert	checagem_mult2	anc_top.mult2_monitor.checagem_mult2	379827	0
overflow_somador1	assert	overflow_somador1	anc_top.adder1_monitor.overflow_somador1	27520	0
checagem_somador1_modo1	assert	checagem_somador1_modo1	anc_top.adder1_monitor.checagem_somador1_modo1	37982	0
checagem_somador1_modo2	assert	checagem_somador1_modo2	anc_top.adder1_monitor.checagem_somador1_modo2	1	0
overflow_adder2	assert	overflow_adder2	anc_top.adder2_monitor.overflow_adder2	228076	0
overflow_saida_adder2	assert	overflow_saida_adder2	anc_top.adder2_monitor.overflow_saida_adder2	37982	0
checagem_adder2	assert	checagem_adder2	anc_top.adder2_monitor.checagem_adder2	37982	0
overflow_somador3	assert	overflow_somador3	anc_top.adder3_monitor.overflow_somador3	225133	0
checagem_somador3	assert	checagem_somador3	anc_top.adder3_monitor.checagem_somador3	379827	0

FIGURA 26. DECLARAÇÕES DE CHECAGEM DOS BLOCOS DO SISTEMA

Por fim, a cobertura funcional durante a simulação tornou possível medir o quanto dos possíveis modos de operação e combinações de dados de entrada foi

simulado. A Figura 27 mostra as declarações de cobertura funcional contidas no código apresentadas graficamente pelo simulador. Para a aquisição desta figura, foi utilizado o simulador *QuestaSim*.



Name	Coverage	Goal	% of Goal	Status
/anc_top/adder3_mon...				
TYPE adder3_cg	78.7%	100	78.7%	
/anc_top/mult1_monitor				
TYPE mult1_cg	79.2%	100	79.2%	
/anc_top/mult2_monitor				
TYPE mult2_cg	100.0%	100	100.0%	
/anc_top/adder1_mon...				
TYPE adder1_cg	74.7%	100	74.7%	
CVP adder1_c...	80.0%	100	80.0%	
CVP adder1_c...	80.0%	100	80.0%	
bin zero	5716	1	571600.0%	
bin low	5716	1	571600.0%	
bin mid	1295193	1	12951930...	
bin high	3519484	1	35194840...	
bin one	0	1	0.0%	
CROSS adder1...	64.0%	100	64.0%	
bin <zero,ze...	16	1	1600.0%	
bin <low,zero>	16	1	1600.0%	
bin <mid,zero>	238	1	23800.0%	
bin <high,ze...	5462	1	546200.0%	
bin <one,zero>	0	1	0.0%	
bin <zero,low>	16	1	1600.0%	
bin <low,low>	16	1	1600.0%	
bin <mid,low>	238	1	23800.0%	
bin <high,low>	5462	1	546200.0%	
bin <one,low>	0	1	0.0%	
bin <zero,mid>	76	1	7600.0%	
bin <low,mid>	76	1	7600.0%	
bin <mid,mid>	95150	1	9515000...	
bin <high,mid>	1199945	1	11999450...	
bin <one,mid>	0	1	0.0%	
bin <zero,hi...	2942	1	294200.0%	
bin <low,high>	2942	1	294200.0%	
bin <mid,high>	171068	1	17106800...	
bin <high,hi...	3345268	1	33452680...	
bin <one,high>	0	1	0.0%	
bin <zero,one>	0	1	0.0%	
bin <low,one>	0	1	0.0%	
bin <mid,one>	0	1	0.0%	
bin <high,one>	0	1	0.0%	
bin <one,one>	0	1	0.0%	
/anc_top/adder2_mon...				
TYPE adder2_cg	80.0%	100	80.0%	
CVP adder2_c...	80.0%	100	80.0%	
bin zero	4468	1	446800.0%	

FIGURA 27. COBERTURA FUNCIONAL DO SISTEMA

6 CONCLUSÕES

Este trabalho apresentou o processo de criação de um modelo funcional e de verificação do processador de áudio anti-ruído. Ao final das etapas incluídas no escopo deste texto, três conclusões foram observadas:

1. A relevância da verificação funcional: Quanto mais complexo é um projeto de circuito digital, maior é o esforço necessário para que se assegure que o projeto não contém erros de lógica. Como foi observado neste trabalho, o projeto de verificação funcional guiou o circuito integrado em direção à conformidade com a especificação. Caso não houvesse sido feito este esforço, o tempo para *debugging* do *design* seria muito mais longo, e seria muito mais trabalhoso afirmar que o processador pode ser fabricado sem risco de perda de produção.
2. A importância da adoção de uma metodologia de verificação funcional: A disciplina de verificação funcional engloba conceitos de gestão de tempo e recursos de projeto. O uso de uma metodologia de verificação já existente otimiza o trabalho, reduz erros, facilita a documentação e reaproveitamento de código.
3. A importância da busca de entendimento sobre o projeto digital: Desde a popularização da verificação funcional no anos 80, é costume que as equipes de desenvolvimento de circuitos digitais e de verificação trabalhem de forma completamente independente. O motivo é evitar a influência de uma equipe sobre a outra, para que erros não passem

despercebidos por ambas. Porém, esta forma de trabalho acarreta em dois grandes problemas:

- A falta de compreensão do *design* faz com que as equipes de verificação funcional não possam criar projetos mais inteligentes e otimizados.
- Já a falta de familiaridade da equipe de projeto digital com a verificação funcional faz com que o tempo para entendimento dos erros ocorridos seja muito longo, portanto aumentando o tempo necessário para sua correção.

Os fatos acima foram claramente observados durante o projeto IP02-ANC. A própria indústria chega a um momento onde o trabalho de desenvolvimento de circuitos digitais e de verificação funcional precisam se integrar para garantir que o tempo de entrega de um circuito integrado não se torne excessivamente longo. Um artigo que entra em mais detalhes sobre esta nova forma de trabalho é encontrado na referência [21].

7 REFERÊNCIAS

[1] Elliot, S., & Nelson, P. (1993). Active Noise Control. *Signal Processing Magazine , 10* (4), 12-35.

[2] Texas Instruments. (1996). *Design of Active Noise Control Systems with the TMS320 family.* Application report.

[3] Projity. (s.d.). *OpenProj*. Acesso em 15 de Maio de 2008, disponível em http://www.openproj.org

[4] X-Fab. (s.d.). *X-FAB Semiconductor Foundries*. Acesso em 4 de Junho de 2008, disponível em http://www.xfab.com

[5] Synopsys, Inc. (s.d.). *Synopsys*. Acesso em 15 de Maio de 2008, disponível em http://www.synopsys.com

[6] Mentor Graphics Corp. (s.d.). *Mentor Graphics*. Acesso em 15 de Maio de 2008, disponível em http://www.mentor.com

[7] Cadence Design Systems. (s.d.). *Cadence Design Systems*. Acesso em 15 de Maio de 2008, disponível em http://www.cadence.com

[8] Tigris.org. (s.d.). *Subversion*. Acesso em 15 de Maio de 2008, disponível em http://subversion.tigris.org

[9] Smith, C. M. (s.d.). *Linux NFS FAQ*. Acesso em 15 de Maio de 2008, disponível em http://nfs.sourceforge.net

[10] *Wordpress > Blog tool and Weblog Platform*. (s.d.). Acesso em 15 de Maio de 2008, disponível em http://wordpress.org

[11] AdventNet. (s.d.). *Zoho*. Acesso em 15 de Maio de 2008, disponível em http://www.zoho.com

[12] Audacity: Free Audio Editor and Recorder (s.d.). Acesso em 31 de Março de 2008, disponível em http://audacity.sourceforge.net

[13] Scilab Consortium. (s.d.). *Scilab Home Page*. Acesso em 15 de Maio de 2008, disponível em http://www.scilab.org

[14] Vaseghi, S. (1996). *Advanced Signal Processing and Digital Noise Reduction.* Wiley Teubner.

[15] Haykin, S., & Van Veen, B. (2000). *Sistemas e Sinais.* Bookman.

[16] Microsoft Corporation. (s.d.). *Microsoft AVI RIFF File Reference.* Acesso em 31 de Março de 2008, disponível em Microsoft Developer's Network: http://msdn2.microsoft.com/en-us/library/ms779636.aspx

[17] *Microsoft WAVE Soundfile Format.* (s.d.). Acesso em 31 de Março de 2008, disponível em http://ccrma.stanford.edu/courses/422/projects/WaveFormat

[18] Cadence Design Systems. *Verification Planning and Management (VPM) Manual Beta.*

[19] *VMM Verification Methodology*, (s.d). Acesso em 20 de Junho de 2008. Disponível em http://www.vmmcentral.org

[20] *Open Verification Methodology,* (s.d.). Acesso em 20 de Junho de 2008. Disponível em http://www.ovmworld.org

[21] Chris Wilson, *Leveraging Design Insight for Intelligent Verification Methodologies.* Acesso em 20 de Julho de 2008, disponível em http://www.edadesignline.com/howto/verification/208401374