**UNIVERSIDADE FEDERAL DE ITAJUBÁ**

**DISSERTAÇÃO DE MESTRADO**

# Uma Metodologia de Desenvolvimento do Controle Digital de Conversores Estáticos Utilizando FPGA

por

**Michel Santana**

**Orientador:** Robson Luiz Moreno, Dr.

**Co-orientador:** Enio Roberto Ribeiro, Dr

Dissertação apresentada à Universidade Federal de Itajubá como parte dos requisitos necessários para a obtenção de título de Mestre em Engenharia Elétrica.

**Julho de 2006**

*"Comece fazendo o que é necessário,*
*depois o que é possível e de repente*
*você estará fazendo o impossível."*
*São Francisco de Assis*

Aos meus pais, José Benedito e Rosangela, dedico.

# *Agradecimentos*

Agradeço a Deus e a Nossa Senhora Aparecida, pelas oportunidades e dádivas na minha vida.

A todos os meus familiares, especialmente aos meus pais, irmãs, tios e avó, por seu incondicional incentivo e amor.

Ao professor orientador, Robson Luiz Moreno, e ao professor co-orientador Enio Roberto Ribeiro, pelo crédito, pela confiança, pela amizade e pela ajuda na realização desse trabalho.

Aos colegas do Grupo de Microeletrônica da UNIFEI, pelas constantes sugestões e pelas contribuições para esse trabalho.

Aos amigos, por seu incontestável apoio, pelos momentos de descontração e, principalmente, pela paciência.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), que, através do programa "Demanda Social", viabilizou a realização desse trabalho.

Meus mais sinceros agradecimentos.

# *Resumo*

Este trabalho apresenta uma metodologia de desenvolvimento de controle digital para conversores de potência, onde se pode modelar toda a estrutura – conversor e sistema de controle – num mesmo ambiente, através da linguagem de descrição de hardware VHDL-AMS. Esta linguagem possibilita descrições comportamentais de toda a estrutura, podendo assim simular, testar e validar o sistema de controle digital sem a necessidade de um protótipo.

O código VHDL do controle digital, devido à portabilidade da linguagem, pode então, ser utilizado para a implementação em uma FPGA ou ASIC. Os resultados de simulação mostram a possibilidade de implementação da metodologia, oferecendo possibilidades interessantes no controle de conversores de potência.

# *Abstract*

This work describes a methodology for the development of the digital control of power converters, where the whole structure – converter and control system – can be modeled in the same environment through the hardware description language VHDL-AMS. This language allows a behavioral description for the whole structure, so that the control system can be simulated, tested and validated without the need of a prototype.

The portability of the VHDL language allows the implementation of the digital control code in an ASIC or FPGA. Simulations results show the feasibility of the methodology offering interesting possibilities in power converter control.

# *Índice*

Capítulo 4



Capítulo 5



Capítulo 6

# *Lista de Figuras*

# *Lista de Tabelas*

# *Lista de Abreviações*

| | |
|---|---|
| ***HDL*** | *Hardware Description Language* |
| ***CAD*** | *Computer Aided Design* |
| ***DSP*** | *Digital Signal Processing* |
| ***FPGA*** | *Field-Programmable Gate Array* |
| ***VHDL*** | *VHSIC hardware Description Language* |
| ***AMS*** | *Analog and Mixed-Signals* |
| ***A/D*** | *Analógico/Digital* |
| ***IEEE*** | *Institute of Electrical and Electronics Engineers* |
| ***CC*** | *Corrente Contínua* |
| ***CA*** | *Corrente Alternada* |
| ***PWM*** | *Pulse Width Modulation* |
| ***CI*** | *Circuito Integrado* |
| ***UPS*** | *Uninterruptible Power Suply* |
| ***ULA*** | *Unidade Lógica e Aritmética* |
| ***MUX*** | *Multiplexer* |
| ***PCB*** | *Printed Circuit Board* |
| ***MCM*** | *Multichip Module* |
| ***RTL*** | *Register Transfer Logic* |

***CPLD*** *Complex Programmable Logic Device*

***FPAA*** *Field-Programmable Analog Array*

***VHSIC*** *Very-High-Speed Integrated Circuit*

***Std*** *Standard*

***CAE*** *Computer Aided Engineering*

***ASIC*** *Application-Specific Integrated Circuit*

# Capítulo 1

# *Introdução*

## 1.1. Considerações iniciais

No desenvolvimento de circuitos eletrônicos, tanto analógicos como digitais, uma diversa gama de ferramentas é utilizada para a confecção de circuitos, que depois de simulados e devidamente validados, serão transformados em protótipos e, enfim, em produtos finais. Nesse contexto, tem-se a divisão do projeto em circuitos analógicos e circuitos digitais [1-2].

Com a maior utilização de circuitos digitais na atualidade, a necessidade de se obter circuitos mistos, que utilizam tanto a tecnologia analógica como a tecnologia digital, vem se tornando cada vez mais pertinente [3]. Tradicionalmente a modelagem de projetos, tanto de eletrônica digital quanto de eletrônica analógica, era executada separadamente e, através das respostas obtidas por cada parte, podia-se prever o resultado do conjunto. Com o auxílio das ferramentas computacionais modernas, a simulação e a validação de projetos que utilizam eletrônica digital e analógica em conjunto podem ser feitas de forma mais natural, num mesmo ambiente de simulação e com uma chance maior de êxito [3-5].

Durante muitos anos, as diversas áreas da engenharia estavam separadas, cada comunidade cientifica tinha sua própria metodologia de desenvolvimento. Com o advento das linguagens de descrição de hardware (HDL – Hardware Description Languages) e com o auxílio das ferramentas de desenvolvimento assistida por computador (CAD – Computer-aided design) tornou-se possível a modelagem e simulação de sistemas nos mais variáveis campos da engenharia, desde eletrônica digital e analógica até a mecânica e a química [4],[2].

## 1.2. Justificativas

A tecnologia digital vem crescendo de forma exponencial, novos dispositivos são especialmente projetados para trabalhar em sistemas digitais. Os controles de sistemas eletrônicos tradicionais, projetados com técnicas de controle clássico, utilizando dispositivos discretos passivos (resistores, capacitores, indutores, etc.) e ativos (diodos, transistores, amplificadores operacionais, etc.), têm vasta utilização, sobretudo em estruturas de potência. No entanto, a utilização de controles com tecnologia digital vem se tornando muito comum e proporcionando ótimos resultados ao sistema [3],[6].

Através do controle digital de estruturas de potência, a sua malha de controle pode ser otimizada com a utilização de (DSP – Digital Signal Processing) ou microcontroladores [7]. Entretanto, com a utilização de circuitos digitais, diversas vantagens podem ser obtidas em virtude da característica concorrente de seu controle. Um importante advento para a confecção de projetos em eletrônica, são as matrizes de elementos básicos digitais programáveis eletronicamente (FPGA – Field Programmable Gate Array), as quais diminuem consideravelmente os custos de prototipagem e possibilitam a integração do circuito em uma única pastilha de silício. As FPGAs podem facilmente ser reprogramadas para atender uma modificação do sistema sem a necessidade da mudança do hardware [3],[6],[8-9].

## 1.3. Objetivos

Apresentar uma metodologia de desenvolvimento de toda a estrutura de controle digital de conversores estáticos de potência, através de circuitos lógicos em um único ambiente de simulação, teste e validação; bem como promover a otimização do tempo e custo dos projetos.

Com a utilização da linguagem de descrição de hardware para circuitos integrados de velocidade muito alta (VHDL–AMS – **V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage – Analog and Mixed-Signals) é possível descrever o comportamento de toda estrutura – estrutura de potência, conversores A/D e o circuito de controle – num mesmo ambiente de simulação, padronizado pelo Instituto de Engenharias Elétrica e Eletrônica (IEEE – Institute of Electrical and Electronics Engineers) [10-11].

## 1.4. Estrutura do Trabalho

O trabalho está organizado em seis capítulos, sendo um de introdução, um de conclusão e os demais de desenvolvimento.

O Capítulo 2 apresenta a estrutura de potência utilizada para testar a metodologia, suas características e formas de controle. A estrutura de potência a ser utilizada é um conversor de corrente alternada (CA) para corrente contínua (CC), abaixador de tensão, também conhecido como BUCK. Existem, basicamente, duas formas de controle do BUCK: o controle analógico e o controle digital. A opção pelo controle digital através de um processador de sinais digitais (DSPs) ou de matrizes de portas digitais reprogramáveis (FPGA) é abordada.

O Capítulo 3 apresenta um estudo sobre as linguagens de descrição de hardware (HDL – Hardware Description Language) que podem descrever sistemas multidisciplinares com um mesmo formato ou padrão e em um mesmo ambiente de simulação. A metodologia de desenvolvimento através de linguagens de descrição de hardware pode ser feita através de vários

níveis de abstração e, em alguns casos, a descrição em HDLs pode ser diretamente implementada no sistema com grande chance de êxito.

O Capítulo 4 apresenta a linguagem de descrição de hardware para sistemas mistos VHDL-AMS e identifica as principais características da linguagem. Os modos de descrição na linguagem são discutidos e exemplificados. Esse capítulo tem como referência principal o manual de referência da linguagem VHDL-AMS [10],[11].

O Capítulo 5 apresenta a descrição comportamental e os resultados de simulação de todos os componentes analógicos necessários para a implementação do conversor BUCK. Apresenta ainda os componentes que possibilitam o desenvolvimento de um controle digital, tais como o conversor análogo-digital (conversor A/D), uma chave digital ideal e circuitos aritméticos que viabilizam o controle. A utilização de sistemas digitais em estratégias de controle precisa ser cuidadosamente ponderada quanto ao dispositivo utilizado na sua implementação, geralmente uma FPGA, que tem uma capacidade limitada de portas, é utilizada para tal propósito. No final do capítulo, é apresentada a descrição estrutural do conversor BUCK com um controle por modulação por largura de pulso (PWM – Pulse Width Modulation).

O Capítulo 6 apresenta as conclusões e as perspectivas de trabalhos futuros.

# Capítulo 2

# *Sistemas de Eletrônica de Potência*

## 2.1. Considerações Iniciais

Os setores industriais, comerciais e domésticos vêm apresentando, cada vez mais, equipamentos eletrônicos que proporcionam maior comodidade e eficiência. Entretanto alguns equipamentos eletrônicos apresentam cargas não-lineares, proporcionando uma relação não-linear entre a tensão e a corrente da rede [11],[12].

Estes equipamentos drenam da rede correntes não senoidais, que provocam uma série de problemas nas instalações e para os sistemas de distribuição e transmissão, tais como [13-14]:

- ✓ Baixo fator de potência;
- ✓ Distorção da tensão da rede no ponto de acoplamento comum, devido à impedância do circuito ou da instalação;

- ✓ Circulação de correntes harmônicas pelo neutro em sistemas trifásicos, provocando queda de tensão neste condutor, principalmente quando existem cargas monofásicas, pois a terceira harmônica e seus múltiplos ímpares se somam no neutro, havendo necessidade de sobredimensioná-lo;

- ✓ Baixa eficiência;

- ✓ Interferência em alguns instrumentos e equipamentos;

- ✓ Sobredimensionamento dos sistemas de distribuição; e

- ✓ Aquecimento em transformadores devido ao efeito pelicular (aumento da resistência do cobre com a freqüência), à histerese e às correntes parasitas.

Existem algumas soluções preventivas para diminuir, ou até mesmo eliminar, os efeitos destas cargas não-lineares (equipamentos eletrônicos) conectados ao sistema elétrico, entre as a quais a utilização de equipamentos que apresentem uma característica resistiva para a rede ou uma baixa distorção harmônica de corrente. Neste contexto, pode-se citar os pré-reguladores de alto fator de potência [15] e as conexões especiais de transformadores [16-17].

Uma dessas soluções preventivas é o conversor estático tipo BUCK [14], que pode ser usado com um pré-regulador de fator de potência elevado. Ele é formado por três etapas: o estágio de retificação e filtragem; o conversor BUCK e um controle através de modulação por largura de pulso (PWM – Pulse Width Modulation).

O conversor BUCK é um conversor abaixador que tem como característica principal a diminuição da tensão média de saída em relação ao seu valor de pico da entrada. O BUCK vem sendo muito utilizado, pois com o mesmo pode-se obter uma correção do fator de potência para muito próximo da unidade, possui proteção contra curto-circuito e inexistência de corrente de partida excessiva (INRUSH). O conversor abaixador CA/CC é formado por três estágios conforme ilustrado na Figura 2.1.

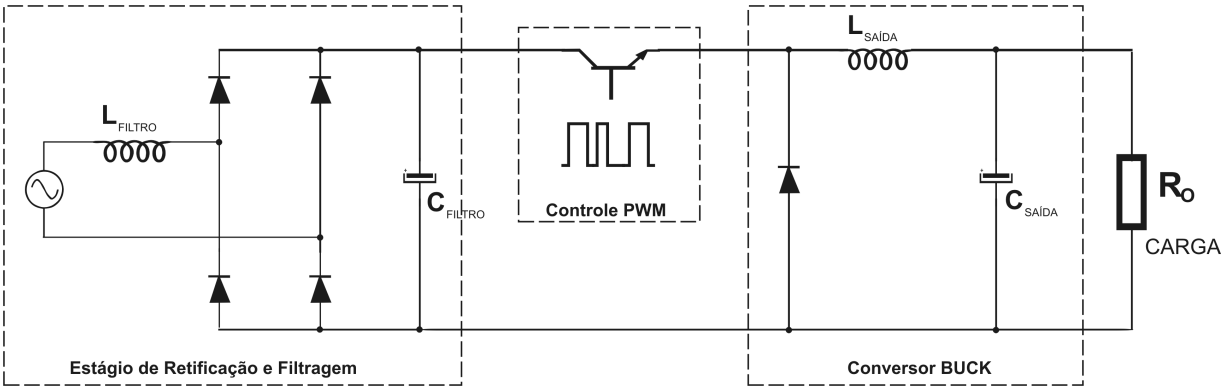



Figura 2.1 – Conversor CA/CC tipo BUCK.

Este conversor será utilizado para testar a metodologia de implementação de circuitos de potência, utilizando VHDL-AMS, e tem como objetivo proporcionar um alto fator de potência para o circuito, bem como uma corrente de entrada em conformidade com a tensão de entrada do circuito.

## 2.2. Metodologias de Controle

Os conversores de potência são tipicamente controlados por circuitos analógicos, que são facilmente encontrados no mercado. Esses circuitos fornecem funções básicas para o controle e têm como principais vantagens o seu baixo custo e a facilidade de uso. Os circuitos de correção de fator de potência não fogem a essa regra e usualmente utilizam CIs comerciais para resolver seus problemas de controle [3],[15],[18].

Existem basicamente duas estratégias de controle de um conversor estático. Uma estratégia é baseada no monitoramento da corrente da carga e do conversor e a outra estratégia é baseada no monitoramento da corrente drenada da rede.

Para as duas estratégias de controle, tem-se uma malha de tensão que deve garantir que o valor médio da tensão na saída do conversor seja mantido constante, e uma malha de corrente que efetivamente faz com que o conversor compense as harmônicas de corrente da carga, impondo uma corrente na rede, de acordo com a estratégia de controle empregada.

O chaveamento do conversor BUCK é definido, através de uma estratégia de controle, pela amostra dos sinais da malha de corrente e de tensão. Estas amostras devem ser devidamente digitalizadas, caso seja feita a opção pelo controle digital.

## 2.3. O Controle Digital

Ao utilizar-se de circuitos analógicos para o controle da correção do fator de potência, têm-se como vantagens o baixo custo e a facilidade de uso, entretanto eles não são configuráveis. Para uma melhoria ou até mesmo uma proteção adicional, faz-se necessário um novo circuito com tal objetivo.

Os circuitos digitais vêm diminuindo seu custo em relação aos circuitos analógicos, bem como aumentando seu desempenho. Em virtude dessas condições, os controles de estruturas de potência vêm migrando para a tecnologia digital, que traz alguns benefícios para o controle, dentre eles pode-se destacar [3]:

- ✓ Facilidade de expansão: O circuito pode ser facilmente reconfigurado para implementar mais funções, tais como proteções de sobre-tensão e sobre-corrente, inicialização, monitoramento, entre outros;

- ✓ Aumento no desempenho: Algumas características podem ser melhoradas devido à alta velocidade de processamento dos atuais dispositivos digitais, em especiais àquelas que necessitam de rápida realimentação; e

- ✓ Múltiplos propósitos: Em sistemas mais complexos, um mesmo circuito digital pode ser utilizado para realizar diversos processamentos.

Com a opção do controle digital, todas as grandezas precisam ser digitalizadas. Essa digitalização pode ser realizada através de conversores análogo-digitais (conversor A/D). No entanto, algumas dessas grandezas necessitam de conversores com alta taxa de amostragem e,

por este motivo, tem seu custo elevado. Devido ao fato de praticamente todos os sistemas necessitarem de pelo menos um conversor A/D, o seu uso constitui na principal desvantagem do controle digital.

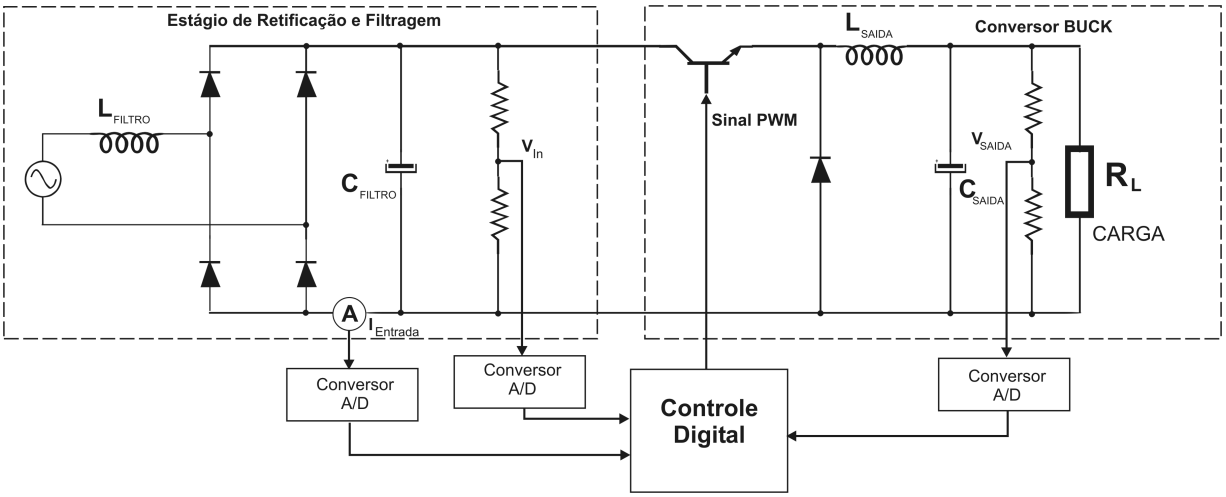



Figura 2.2 – Esquema de controle digital de um BUCK através do monitoramento da corrente drenada para a fonte.

Na Figura 2.2, tem-se o esquema geral de um controle digital através do monitoramento da corrente, que é drenada para a fonte, e da tensão de saída. Nessa figura, também é mostrado uma implementação de um *feed-foward* da tensão de entrada. Nota-se que todas as grandezas analógicas envolvidas no controle necessitam ser transformadas em grandezas digitais, ou seja, digitalizadas.

A implementação do controle digital se difere no hardware utilizado e no tipo de estratégia de controle. As principais formas de implementação são [3]:

- ✓ O uso de um processador digital de sinais (DSP – Digital Signal Processing) ou microcontrolador comercial executando o algoritmo do controle. Muitos DSPs e microcontroladores têm em seu hardware interno circuitos que facilitam a sua utilização em estruturas analógicas, tais como conversores A/D e temporizadores (timers) para geração de sinais do tipo PWM. Esta solução é amplamente utilizada em aplicações em eletrônica de potência; e

- ✓ A utilização de sistemas digitais em conjunto com conversores A/D. Essa solução tem como vantagens a possibilidade de processamento concorrente (diferentemente do processamento seqüencial dos DSPs), o processamento em tempo real, a personalização máxima do circuito utilizado e a implementação em um único CI através de FPGAs.

### 2.3.1. O Controle Digital Utilizando DSP

Os DSPs têm sido amplamente utilizados nas áreas de controle de motor, automação, fontes de alimentação ininterruptas (UPS – Uninterruptible Power Supply), entre outras. A sua característica de processamento matemático, alta velocidade, gerador PWM e conversores A/D integrados e a facilidade de programação através de linguagens de alto nível de abstração estão expandindo as possibilidades de seu uso. Comparado com o controle analógico tradicional, o uso de DSPs provê as seguintes vantagens[3][7]:

- ✓ Uma descrição padronizada do hardware utilizado;
- ✓ Facilidade de implementação;
- ✓ Uso de sofisticadas técnicas de controle, devido à natureza matemática de suas instruções; e
- ✓ Flexibilidade para mudança de parâmetros da estratégia de controle escolhida.

### 2.3.2. O Controle Digital Utilizando Circuitos Lógicos

O controle digital normalmente é feito utilizando um DSP. Mas existe ainda a possibilidade de se confeccionar um sistema digital dedicado e implementá-lo em um FPGA. Em comparação com a solução em DSP, pode-se fazer as seguintes considerações [3],[7],[19]:

- ✓ O controle também pode ser descrito em uma linguagem de alto nível padronizada como, por exemplo, o VHDL e o Verilog [20];

- ✓ Os DSPs já têm instruções matemáticas implementadas em seu conjunto de instruções, que são muito complexas para serem desenvolvidas com um circuito lógico digital. Entretanto, fazendo o controle com operações matemáticas mais simples, pode-se proporcionar um resultado satisfatório se for compensado com a altíssima velocidade utilizada em FPGAs; e

- ✓ Os DSPs, devido a sua característica de programação, são componentes seqüenciais, ou seja, uma instrução só é executada após o término da anterior. Já no FPGA, pode-se ter processos concorrentes, ou seja, dois processos distintos, sendo executados ao mesmo tempo, possibilitando alguns recursos que não são possíveis com a solução em DSP.

Ao fazer o controle de estruturas analógicas utilizando circuitos digitais, é necessário um ambiente de teste e simulação que comporte a descrição de toda a estrutura – sistema de potência e o controle. Por terem naturezas diferentes – analógica e digital – a utilização de uma linguagem que possa utilizar circuitos analógicos, digitais e mistos se torna muito conveniente.

No próximo capítulo, será apresentado um estudo sobre as linguagens de descrição de hardware que permitem a descrição de circuitos analógicos.

# Capítulo 3

# *As Linguagens de Descrição de Hardware*

## 3.1. Introdução

A dificuldade de desenvolvimento e gerenciamento de sistemas de grande complexidade, sobretudo quando envolvem equipes que trabalham separadamente e que, normalmente, têm habilidades diferentes (por pertencerem a ramos diferentes da engenharia), motivaram a criação de linguagens que possibilitam a interação entre as engenharias usando um formato ou uma linguagem comum. Essas linguagens utilizam um alto nível de abstração e foram definidas como linguagens de descrição de hardware (HDLs), devido ao fato de ser possível descrever o comportamento dos componentes do sistema de forma natural. As principais vantagens do uso das HDLs são [5]:

- ✓ Permite fases mais curtas no desenvolvimento dos projetos;

- ✓ Provê um monitoramento e uma verificação constante do desempenho e comportamento do sistema, pois utiliza um mesmo simulador com os mesmos sinais para todo o sistema;

- ✓ O sistema é modelado independente da tecnologia alvo, e pode ser implementado e sintetizado de diversos modos. O que provê ao modelo a possibilidade de ser reutilizado em outros projetos, reduzindo assim o custo do desenvolvimento; e

- ✓ Promove uma interface comum entre os diferentes grupos envolvidos no projeto.

Com essas vantagens, a probabilidade de erro no desenvolvimento é consideravelmente reduzida, bem como o tempo e custo do projeto.

## 3.2. Níveis de Abstração

Tem-se, nas linguagens de descrição de hardware, três domínios para representar os níveis de abstração em um projeto [5] [21]:

- ✓ Estrutural – onde o sistema é definido como um conjunto de componentes e interconexões (quais componentes o sistema tem);

- ✓ Comportamental – onde é descrita a funcionalidade do componente (o que o sistema faz); e

- ✓ Físico – onde o tamanho, a tecnologia e a implementação física são considerados (como o sistema é feito).

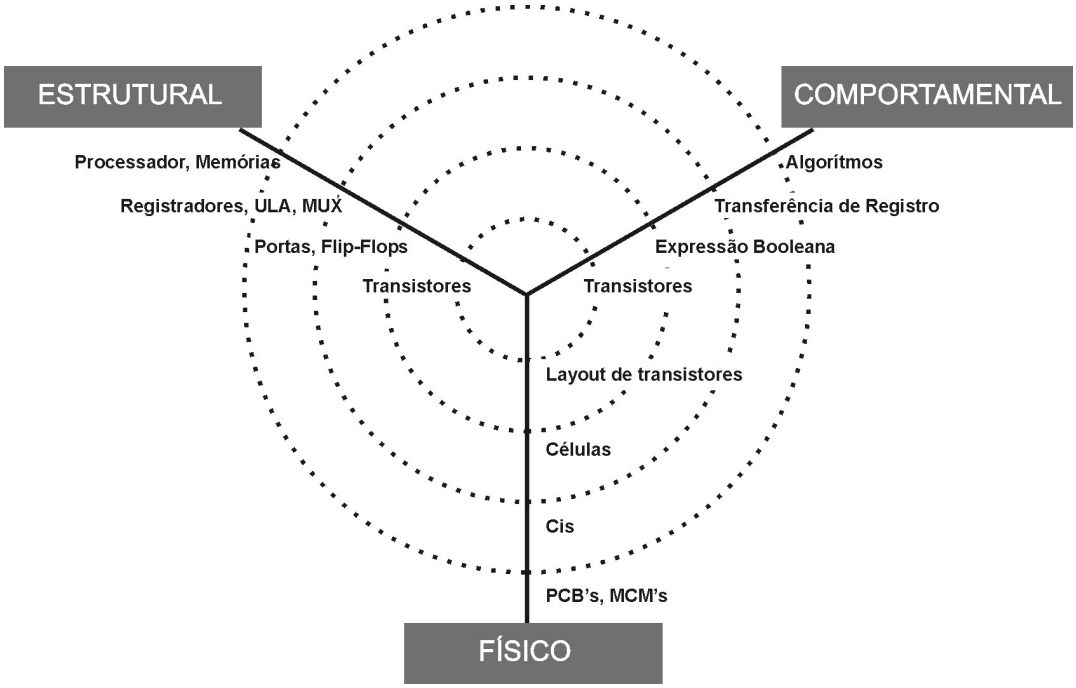



Figura 3.1 – Níveis de abstração para os domínios de desenvolvimento.

A Figura 3.1 representa os diferentes níveis de abstração para esses domínios. Nesse diagrama, o nível de abstração cresce de acordo com a distância do centro e é representado para cada domínio.

No domínio comportamental, o nível de abstração mais alto é o da descrição do algoritmo do componente a ser desenvolvido, ou seja, somente com o conhecimento do comportamento ou funcionalidade do componente é possível descrevê-lo. Para os domínios físico e estrutural, o nível de abstração mais alto permite apenas que o desenvolvedor descreva o componente como um conjunto de blocos e suas interconexões, sendo assim, é necessário conhecer precisamente a configuração interna e as características de seus sinais de entrada e saída de todos os blocos que constituem o sistema.

Nos níveis de abstração mais baixos, o domínio comportamental também se mostra mais simples. Para o nível de abstração de expressões booleanas no domínio comportamental, por exemplo, a descrição de uma função lógica ou-exclusivo (XOR) faz-se somente através da expressão booleana $S = A \oplus B$, já no domínio estrutural, é necessário especificar as operações lógicas básicas, previamente implementadas, $S = \overline{A} \cdot B + A \cdot \overline{B}$ (caso a

porta ou-exclusivo não tenha sido implementada). No domínio físico, a confecção se dá através das ligações entre transistores em uma determinada tecnologia de integração.

## 3.3. Fluxo de Desenvolvimento

O fluxo de desenvolvimento em um sistema é a forma como o mesmo é concebido. Tradicionalmente, o desenvolvimento de sistemas é baseado na construção de componentes elementares como portas lógicas e transistores. Blocos mais complexos são desenvolvidos através de componentes elementares já criados. A essa metodologia, dá-se o nome de *Bottom-up*, ou seja, descrição de mais baixa hierarquia para uma hierarquia mais alta.

Com o advento das linguagens de descrição de hardware, pode-se basear a geração de código através de sua funcionalidade ou comportamento, simulação e posterior síntese. A esse método dá-se o nome de metodologia *Top-Down*, ou seja, a descrição de uma hierarquia mais alta para uma outra mais baixa. A Figura 3.2 ilustra a diferença das metodologias de desenvolvimento *bottom-up* e *top-down* [5].

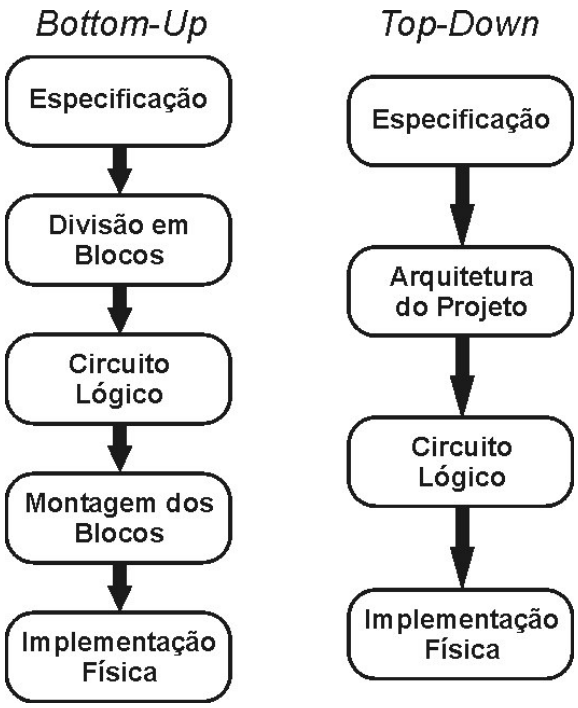



Figura 3.2 – Fluxos de desenvolvimento *Bottom-Up* e *Top-Down*.

A metodologia de fluxo de desenvolvimento *Top-Down* é mais apropriada para o uso de metodologias de desenvolvimento baseadas em HDLs, e seu uso é possível devido às ferramentas de CAD, que promovem a simulação e síntese do código gerado [3],[22],[23] .

Para a obtenção do circuito eletrônico, descrito através de uma metodologia específica, faz-se a síntese da descrição, a qual permite a sua implementação física.

## 3.4. Síntese

A síntese é a transformação de uma descrição de alto nível de abstração em outra com um nível de abstração mais baixo. Dependendo do nível de abstração da descrição do sistema, dois tipos de sínteses podem ser considerados [5]:

- ✓ A síntese comportamental, que traduz a descrição do algoritmo em uma lógica de transferência de registro (RTL – Register Transfer Logic); e

- ✓ E a síntese de RTL, que traduz uma descrição de transferência de registro em um modelo de níveis lógicos.

A principal diferença entre as descrições em algoritmos e de transferência de registro é o esquema de sincronismo (*clock*) utilizado.

Em uma segunda etapa, chamada de processo de refinamento ou síntese RTL, se dá a tradução da descrição RTL como um conjunto de interconexões de portas lógicas. Nessa descrição, contem informações sobre atrasos de sinais e problemas particulares que podem aparecer no circuito.

A síntese RTL tem sido disponibilizada há alguns anos e muitos desenvolvedores de CADs oferecem produtos de síntese para diferentes linguagens, como o VHDL e o Verilog. Pelo ponto de vista prático, na maioria dos casos, a especificação da descrição é diretamente

redefinida para uma descrição RTL e uma ferramenta de síntese RTL é usada para gerar o esquemático.

As ferramentas de síntese têm algumas limitações em relação à descrição utilizada na modelagem do componente, dependendo do nível de abstração que é considerado. Uma comparação entre diferentes ferramentas de síntese RTL pode ser encontrado em [24]. Na descrição de um código em HDL é importante observar como a ferramenta de síntese trabalha para que o circuito final gerado seja o mais otimizado possível.

Os métodos de sínteses automáticas são largamente utilizados na atualidade, no entanto, têm algumas diferenças em relação a síntese manual [5]:

- ✓ O desenvolvedor perde o conhecimento da estrutura que está sendo realmente implementada no circuito;
- ✓ A qualidade do projeto final pode ser inferior ao esperado, com o resultado de uma síntese automática; e
- ✓ O método tradicional de desenvolvimento de hardware, através de esquemáticos, é alterado, implicando assim numa mudança de mentalidade do desenvolvedor.

Atualmente, tem-se no mercado diversos fabricantes de dispositivos que proporcionam a síntese de circuitos lógicos digitais através de HDLs. O circuito lógico final é então implementado em um dispositivo lógico complexo programável (CPLD – Complex Programmable Logic Device) ou FPGA, que podem conter circuitos dos mais variados níveis de complexidade [25]. O uso dos FPGAs diminuem, consideravelmente, o tempo de prototipagem e seu custo de projeto [3].

Com o avanço das HDLs, sistemas que englobam tecnologias analógicas e mistas podem ser descritos de forma comportamental. A possibilidade de síntese desses sistemas tem sido amplamente estudada e alguns fabricantes já produzem matrizes analógicas reprogramáveis

(FPAA – Field Programmable Analog Array) [26] através de HDLs , sobretudo através da linguagem VHDL-AMS. Diversas técnicas de construção de FPAAs são apresentadas em [27-31].

No desenvolvimento de sistemas mistos, pode-se utilizar, para uma eficiente e rápida prototipagem, uma FPGA para basear o circuito digital e uma FPAA para basear o circuito analógico [32-34], podendo assim modificar facilmente o comportamento e as características do sistema a ser desenvolvido.

Conforme abordado, a utilização de HDLs é uma vantagem para o desenvolvimento de um projeto. No próximo capítulo, será abordada a linguagem VHDL-AMS, a qual permite uma grande flexibilidade na descrição de circuito digitais analógicos e mistos e será a linguagem utilizada na metodologia apresentada nesse trabalho.

# Capítulo 4

# *Descrição de Circuitos Utilizando a Linguagem VHDL-AMS*

## 4.1. A Linguagem VHDL-AMS

A linguagem VHSIC Hardware Description Language (VHDL) foi desenvolvida com o intuito de ser utilizada em todas as fases da criação de um sistema eletrônico. Tal linguagem promove o desenvolvimento, verificação, síntese e teste no desenvolvimento do hardware, bem como sua manutenção, modificação e expansão [35-37].

Sua primeira padronização ocorreu no ano de 1987, através do padrão IEEE Std 1076-1987 e adotou a versão 7.2, concluída no ano de 1986. A partir desta primeira versão, muitas modificações foram introduzidas na versão original e, em 1993, uma versão mais atualizada e com novas especificações da linguagem foi compilada através do padrão IEEE Std 1076-1993 [10].

Com o crescente uso de sistemas digitais e a sua integração com sistemas analógicos, sentiu-se a necessidade de uma atualização na linguagem VHDL. Em 1999, um conjunto de atualizações (chamado de *superset*) da linguagem VHDL foi compilado. A esse *superset* deu-se o nome de VHDL-AMS. O padrão IEEE Std 1706.1-1999 [11] viabiliza a descrição e simulação de sistemas analógicos e sistemas mistos. A linguagem suporta vários níveis de abstração na descrição de circuitos elétricos e não-elétricos. A padronização da linguagem promove uma interação entre a parte digital e a parte analógica de maneira flexível e eficiente.

A grande vantagem do uso do VHDL está no fato da linguagem ser portátil e independente da ferramenta de projeto utilizada. Por esses motivos, a linguagem vem sendo empregada em padronização de entradas e saídas de ferramentas CAD e CAE (Computer Aided Engineering – Engenharia Assistida por Computador), incluindo simulação, síntese e implementação de componentes digitais. O desenvolvimento de um componente digital usando VHDL é multiplataforma, ou seja, o código pode ser utilizado em qualquer Sistema Operacional que tenha uma ferramenta de projeto compatível com esta linguagem.

## 4.2. A Modelagem de Componentes em VHDL-AMS

Na descrição de um componente através da linguagem VHDL-AMS, deve-se analisar o seu propósito, definindo suas características como as interfaces de entrada e saída, seus atributos e descrevendo sua estrutura ou comportamento. Após a concepção do componente, pode-se então começar a descrição do modelo.

Têm-se, para a linguagem VHDL-AMS, basicamente, três etapas na modelagem de componentes:

- ✓ A definição das bibliotecas utilizadas em sua descrição;
- ✓ A identificação e caracterização de suas entradas, saídas e atributos; e

- ✓ A descrição de seu comportamento ou estrutura.

Para facilitar o entendimento, usar-se-á o exemplo da modelagem de um multiplexador de um dígito (*bit*), com duas entradas e uma saída, além de uma entrada de seleção. Para descrever o multiplexador, deve-se, em sua concepção, definir suas entradas, saídas e sua funcionalidade. A Figura 4.1 mostra o multiplexador em questão. As duas entradas (*A* e *B*), saída e o sinal de seleção (*Sel*) são sinais digitais e por esta razão podem utilizar o tipo lógico padrão (*standard logic*) para sua modelagem. Os tipos de portas de entrada e saída disponíveis na linguagem VHDL-AMS podem ser encontrados em [10], [11].

A funcionalidade do multiplexador também é ilustrada na Figura 4.1. Quando a entrada digital *Sel* tem nível lógico baixo, a saída *Output* recebe a entrada *A* e, quando *Sel* tem nível lógico alto, a saída *Output* recebe a entrada *B*.

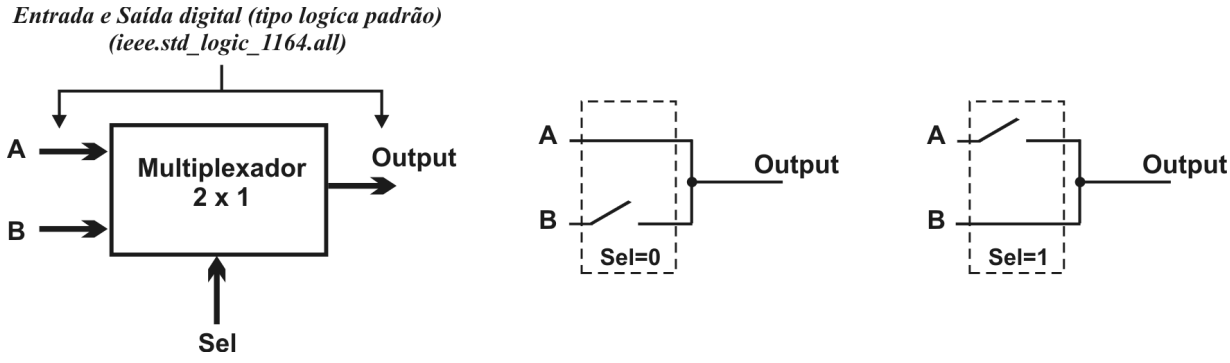



Figura 4.1 – Multiplexador de duas entradas (A e B) e uma saída (Output).

Já definido os tipos de sinais de entrada e saída e o funcionamento do componente, pode-se agora descrevê-lo. Inicialmente, definem-se as bibliotecas utilizadas.

### 4.2.1. Definição das Bibliotecas

Para se modelar um componente em VHDL-AMS, é necessário que o desenvolvedor tenha pacotes com informações sobre a linguagem, as quais ajudarão na análise sintática e semântica do compilador da ferramenta CAD, na qual o código será executado. Essas informações podem ser acerca dos tipos de variáveis que serão utilizados para se modelar o sistema, operações matemáticas, constantes matemáticas e físicas ou um pacote de componentes modelados pelo desenvolvedor. A esses pacotes dá-se o nome de biblioteca (*library*) [35].

A linguagem VHDL permite que o desenvolvedor mantenha, em uma única descrição, várias bibliotecas, onde cada parte de seu componente esteja em bibliotecas distintas.

Tabela 4.1 – Exemplos de bibliotecas da linguagem VHDL-AMS.

| **ALGUMAS BIBLIOTECAS DA LINGUAGEM VHDL-AMS** | | |
|---|---|---|
| **Biblioteca (*library*)** | **Exemplo de uso** | **Finalidade** |
| *std_logic_1164* | *library ieee;*<br>*use ieee.std_logic_1164.all;* | *Fornecer informações sobre o tipo padrão lógico (Standard Logic), que é utilizado para descrever componentes digitais. Define para esse tipo os níveis lógicos alto, baixo e alta-impedância.* |
| *std_logic_arith* | *library ieee;*<br>*use ieee.std_logic_arith.all;* | *Fornecer as operações matemáticas de soma, subtração, multiplicação, etc. para o tipo std_logic.* |
| *math_real* | *library ieee;*<br>*use ieee.math_real.all;* | *Fornecer as operações matemáticas e algumas grandezas matemáticas e físicas para o tipo real.* |
| *electrical_systems* | *library ieee;*<br>*use ieee.electrical_systems.all;* | *Fornecer informações sobre o tipo sistemas elétricos (electrical_systems), que é utilizado para descrever componentes analógicos.* |

Diversas bibliotecas pertencem ao padrão da linguagem VHDL-AMS [11]. Para utilizá-las, deve-se invocar a biblioteca através da diretiva de linguagem ***library*** como ilustrado na Tabela 4.1. Observe que as bibliotecas podem ser divididas em partes menores, chamadas de classes. Nos exemplos da tabela, têm-se na biblioteca principal *ieee* diversas classes, as quais contêm funções específicas. A diretiva ***use*** possibilita o uso dessas classes. A linguagem é baseada no conceito de hierarquia de classes, portanto é necessário invocar a biblioteca antes de instanciar as classes. Cada classe pode ser utilizada separadamente bastando referenciá-la através de um ponto (por exemplo, *ieee.std_logic*). Para utilizar todas as funções de uma classe, faz-se uma referencia geral (*.all*).

Através das bibliotecas adequadas, podem-se descrever componentes analógicos, digitais, com características mecânicas, etc. As descrições contidas nesse trabalho utilizam as bibliotecas desenvolvidas pelo IEEE.

Na modelagem de um componente, devem-se observar os tipos que serão utilizados e quais funções serão necessárias, para que seja feita uma escolha adequada das bibliotecas.

No exemplo do multiplexador, precisa-se somente da biblioteca que define os tipos de entrada e saída do componente e suas operações lógicas (*ieee.std_logic_1164.all*). A descrição das bibliotecas para o exemplo está ilustrada na figura 4.2.

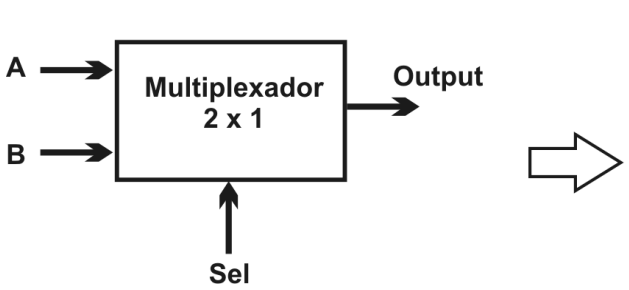
Definidas as bibliotecas, deve-se especificar como é o componente, suas interfaces de entrada e saída e seus atributos.

### 4.2.2. A identificação e caracterização de suas entradas, saídas e atributos

Todo componente precisa ter uma estrutura que defina suas entradas, saídas e atributos. Na linguagem VHDL-AMS, cria-se uma entidade (*entity*) que define a interface do componente com o exterior. A estrutura de uma entidade é a seguinte [35]:

```
entity nome_do_componente is
        generic lista de atributos;
        ports lista de interfaces;
end entity nome_do_componente;
```

Note que, em uma definição de entidade, o componente já deve estar concebido pelo desenvolvedor de forma que todos os atributos e portas de interface possam ser instanciados. No exemplo do multiplexador, não se tem atributos internos que necessitem ser instanciados, somente portas de entradas e saídas. A descrição de sua estrutura será como ilustrado na Figura 4.2.




```
library ieee;
use ieee.std_logic_1164.all;


entity mux2 is
  port ( Sel          : in  std_logic;  -- sinal de seleção
         A,B : in  std_logic;         --sinais de entrada
         Output       : out std_logic );  -- sinal de saída
end entity mux2;
```

Figura 4.2 – Definição de *library* e *entity* para o multiplexador 2 x 1.

Definido como será a interface, quais os atributos necessários ao componente e quais as bibliotecas necessárias à sua confecção, deve-se descrever qual a funcionalidade do componente.

## 4.2.3. Descrição do componente

Têm-se para a linguagem VHDL, basicamente, dois tipos de descrição:

- ✓ Comportamental (Behavorial) – descreve a funcionalidade de uma sistema, sem especificar as suas arquiteturas de registros; e

- ✓ Estrutural (Structural) – descreve o sistema como um conjunto de componentes básicos interconectados entre si.

Para descrever, de maneira comportamental, um circuito lógico digital, pode-se utilizar, através da sintaxe da linguagem, uma função boolena ou simplesmente o seu comportamento através de um algoritmo. A Figura 4.3 ilustra esses dois tipos de descrição para o exemplo do multiplexador. Na descrição comportamental, traduz-se a funcionalidade através de um algoritmo, no multiplexador analisa-se a entrada de seleção e, através de uma estrutura condicional, toma-se uma decisão. Já na descrição estrutural, os componentes já criados são instanciados de forma a definir a arquitetura interna do multiplexador e implementado de forma a prover ao componente a funcionalidade requerida, conforme ilustra a Figura 4.3.

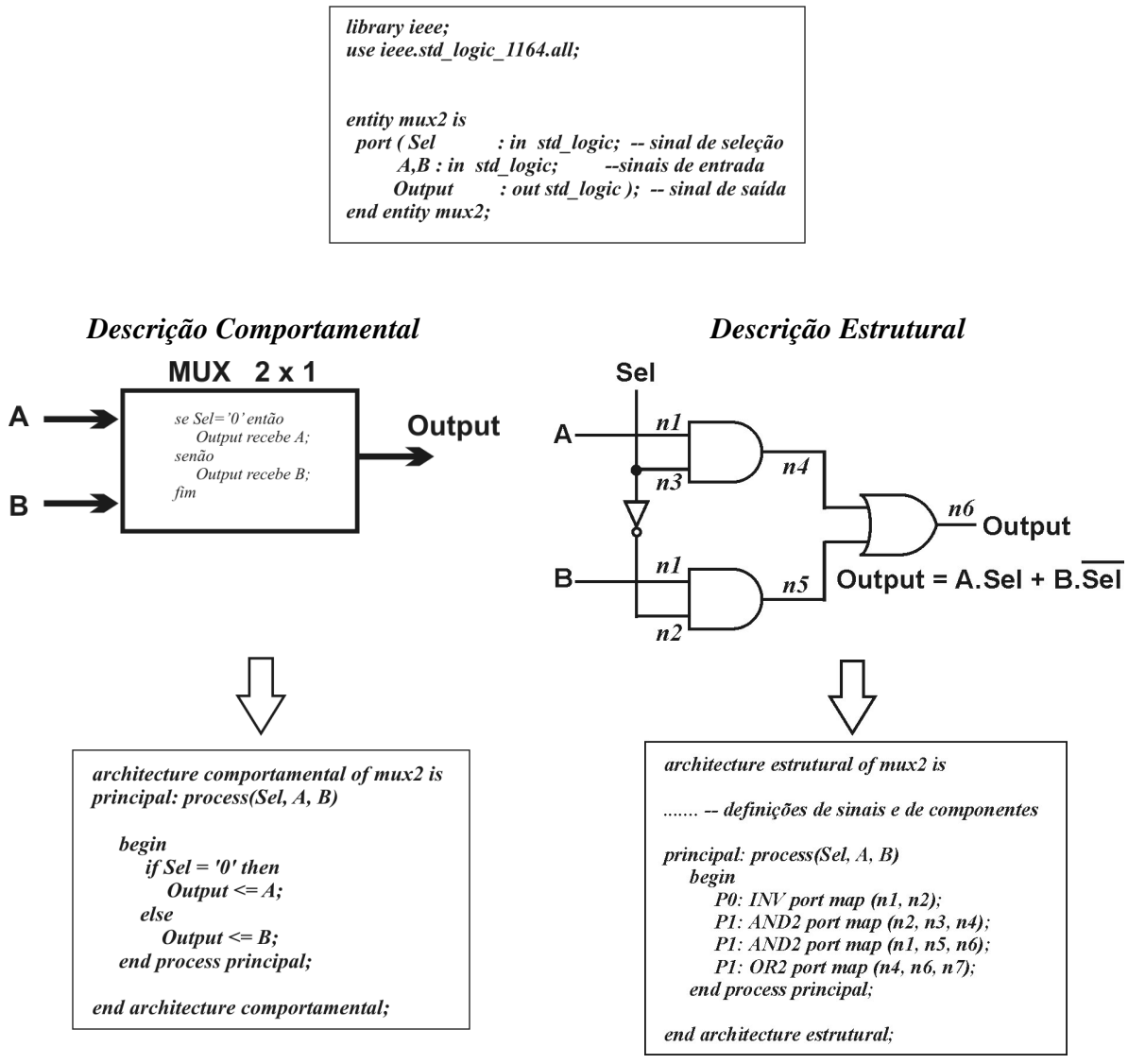



Figura 4.3 – Descrição comportamental e estrutural do multiplexador 2 x 1.

Para descrever a funcionalidade de um componente, constrói-se uma arquitetura (*arquitecture*) que vai informar qual o método de descrição a ser utilizado pelo projetista. Uma mesma entidade pode ter diversas arquiteturas. Quando o componente for utilizado no desenvolvimento, a arquitetura é referenciada.

# Capítulo 5

# *Descrição da Estrutura*

## 5.1. Descrição da Estrutura de potência em VHDL-AMS

Como discutido nos capítulos anteriores, neste capítulo, será apresentada a descrição de toda a estrutura envolvida no sistema. Para obter os resultados de simulação foram utilizados os softwares SMASH® e MAX+PLUS II®.

Para descrever componentes analógicos através da linguagem VHDL-AMS, faz-se necessário a utilização da biblioteca que dispõe das especificações de sinais elétricos. A biblioteca de sistemas elétricos pode ser instanciada como ilustrado na Tabela 4.1. E os tipos de atributos e grandezas utilizados pela linguagem estão disponíveis nas especificações do padrão da linguagem [11]. Os componentes serão descritos de forma comportamental e o conversor BUCK será descrito de forma estrutural.

Ao se descrever componentes analógicos, associa-se ao mesmo uma tensão e uma corrente, isso se dá através das diretivas de quantidade (*quantity*), através de (*across*) e de um

terminal a outro (*through terminal1 to terminal2*). Por exemplo, no resistor da Figura 5.1, tem-se a tensão *V* sobre o resistor (de resistência igual a *res*) e uma corrente *i,* que o atravessa do terminal de entrada *pos,* até o terminal de saída *neg*. Todas as descrições da estrutura analógica apresentadas nesse trabalho utilizam tais diretivas para a definição de tensão e da corrente do componente.

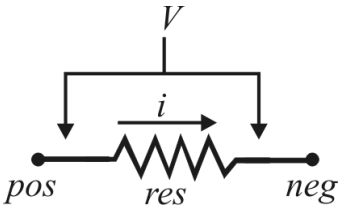



Figura 5.1 – Corrente e tensão do resistor gerada pelo *(quantity v across i through pos to neg).*

Os modelos descritos nesse trabalho têm comportamento próximo ao real, entretanto algumas modificações podem ser feitas caso haja necessidade. Como, por exemplo, a introdução de condições iniciais em indutores e capacitores e os atrasos proporcionados pelos circuitos lógicos.

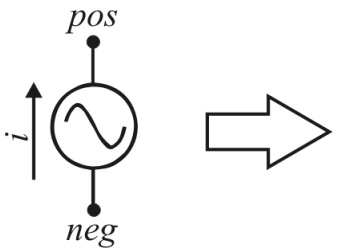
## 5.1.1. Descrição Comportamental da Fonte CA

O primeiro componente a ser desenvolvido é a fonte de tensão senoidal de corrente alternada. O seu comportamento consiste em gerar um sinal senoidal com uma tensão, freqüência, fase e deslocamento (*offset*) pré-definidos. A Figura 5.2 contém o símbolo do componente, a sua descrição e seu resultado de simulação. Para descrever o componente é necessário a utilização de duas bibliotecas, uma que descreve os sistemas elétricos (linha 3) e outra que contem operações matemáticas para o tipo real (linha 2). As especificações dos tipos de sinais utilizados pode ser observado na descrição da fonte de alimentação, da linha 4 a linha 10. Observa-se que os atributos estão especificados de forma genérica (linha 5 à linha 8), mas o desenvolvedor pode instanciar qualquer valor quando o componente for descrito estruturalmente.

O comportamento de uma fonte senoidal é dado pela função seno, que tem como atributo um valor em radianos. A base de tempo para que seja calculado o valor instantâneo da

tensão nos terminais da fonte é dada pela diretiva agora (*NOW*). Nas linhas 15 e 16, pode-se observar o comportamento da fonte CA. O valor da fase, dado em graus, é somado ao ângulo obtido pela base de tempo e então convertido para radianos. Aplica-se então o valor do ângulo ao seno, que é multiplicado pela tensão de pico da fonte (*amplitude*) e soma-se uma tensão de *offset* para se obter a tensão de saída (*V*) da fonte de tensão.

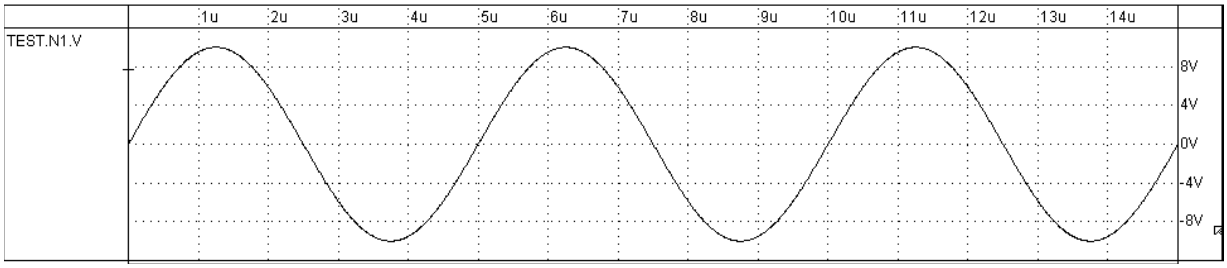
Esse componente foi descrito com o intuito de viabilizar a simulação de toda a estrutura e representa a tensão (rede de distribuição) fornecida ao sistema.




(a)

```
1.  library ieee;
2.  use ieee.math_real.all;
3.  use ieee.electrical_systems.all;

4.  entity v_sine is
5.  generic (     freq      : real := 100.0e3;
6.                amplitude : voltage := 10.0;
7.                phase     : real    := 0.0;
8.                offset    : voltage := 0.0);
9.  port ( terminal pos, neg : electrical);
10. end entity v_sine;

11. architecture behav of v_sine is
12. quantity v across i through pos to neg;
13. quantity phase_rad : real;
14. begin
15.       phase_rad == math_2_pi *(freq * NOW + phase / 360.0);
16.       v == offset + amplitude * sin(phase_rad);
17. end architecture ideal;
```

(b)




(c)

Figura 5.2 – Representação de uma Fonte CA (a), sua descrição comportamental (b) e seus resultados de simulação (c).

## 5.1.2. Descrição Comportamental do Resistor

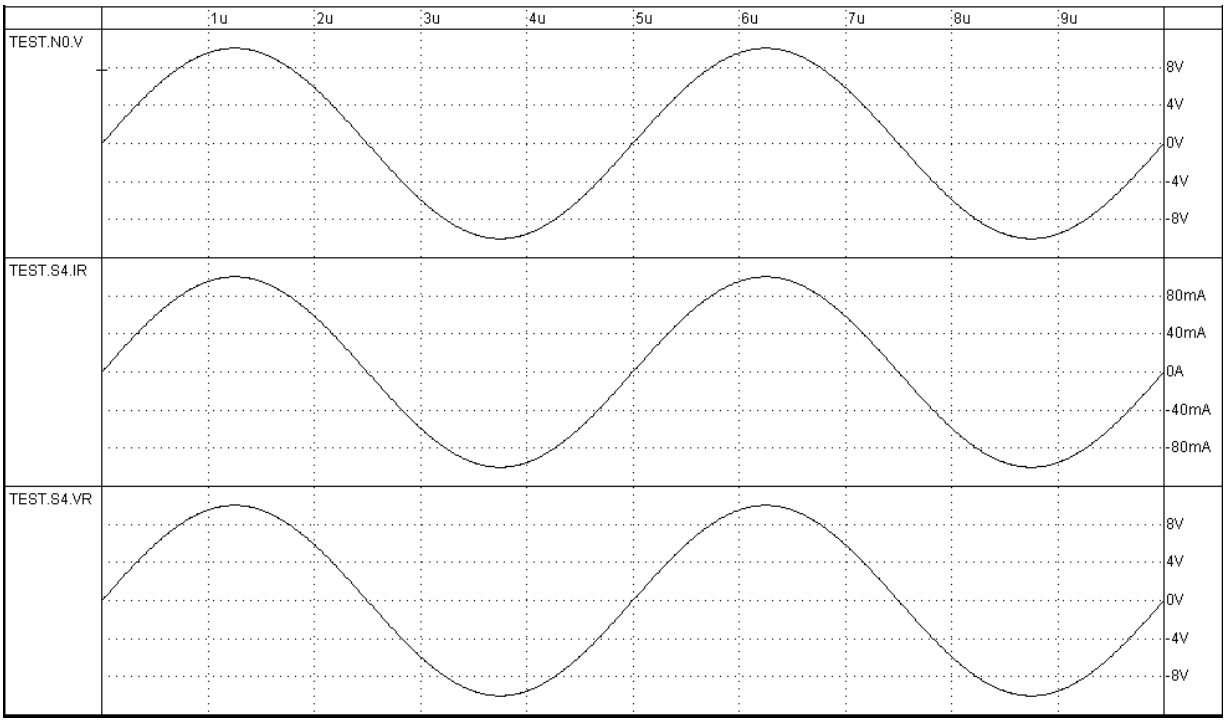
O resistor é um componente muito utilizado em eletrônica analógica, oferecendo à corrente elétrica uma resistência, que é caracterizada através de um valor numérico. Esse valor é usualmente dado em ohm (Ω) e a queda de tensão nesse componente é proporcional ao valor em ohm da resistência. Esse componente é representado na Figura 5.3, bem como sua descrição e o resultado de simulação, utilizando como fonte de tensão o exemplo da Figura 5.2. Sua equação fundamental é especificada na linha 10 do código e seus valores de tensão e corrente são associados a *v* e *i*, respectivamente, através da linha 8 da descrição. Na figura 5.3c, tem-se a tensão da fonte CA (TEST.N0.V) a corrente no resistor (TEST.N1.IR) e a tensão sobre o resistor (TEST.N1.VR). Observe que os valores utilizados tanto na fonte CA quanto no resistor são seus valores genéricos.

O resistor descrito na Figura 5.3b tem como resistência um valor genérico de 100 ohms, no entanto essa resistência pode ser modificada no momento em que o componente é instanciado. O valor genérico é, então, o valor padrão associado ao componente descrito.


(a)

```
1. library ieee;
2. use ieee.electrical_systems.all;

3. entity resistor is
4.         generic ( res : resistance : 100.0);
5.         port (terminal pos, neg : electrical);
6. end entity resistor;

7. architecture behav of resistor is

8. quantity v across i through pos to neg;
9. begin
10.        v == i*res;
11. end architecture behav;
```

(b)

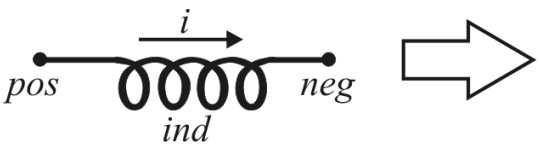



(c)

Figura 5.3 – Representação (a), descrição comportamental (b) e resultados de simulação (c) de um resistor ideal.

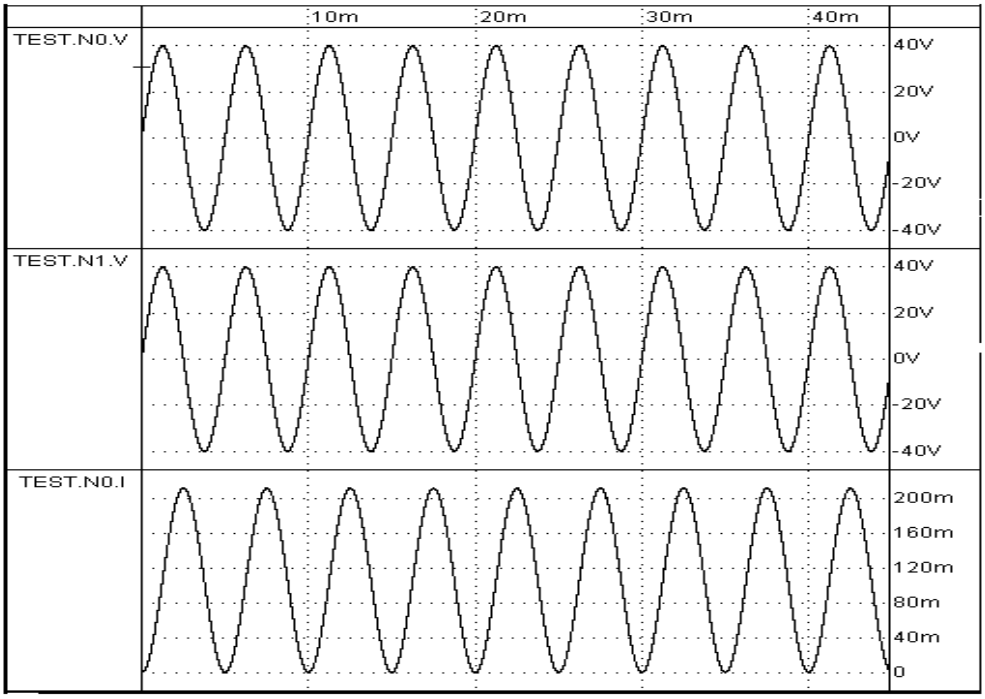
## 5.1.3. Descrição Comportamental do Indutor

O comportamento físico do indutor é descrito através de uma equação fundamental, que diz que a tensão no indutor é igual a sua indutância em Henry, multiplicado pelo valor da derivada da corrente que o atravessa. Esse comportamento pode ser observado na descrição da Figura 5.4b, na linha 10. A Figura 5.4 ilustra o indutor, sua descrição e seu resultado de simulação quando submetido a fonte de tenção CA da Figura 5.2. Como pode-se observar na Figura 5.4c, o sinal de entrada TEST.N0.V é introduzido no indutor com indutância genérica descrita na linha 5, obtendo-se então uma corrente defasada em 90º e uma tensão em fase com o sinal de entrada, como pode ser observado nos sinais TEST.N0.I e TEST.N1.V.




```
1.  library ieee;
2.  use ieee.electrical_systems.all;

3.  entity indutor is
4.   generic ( ind  : inductance: 300.0e-3);
5.   port (terminal pos, neg : electrical);
6.  end entity indutor;

7.  architecture behav  of indutor is
8.  quantity v across i through pos to peg;
9.  begin
10.  v   == ind * i'dot;
11. end architecture behav;
```

(a) (b)



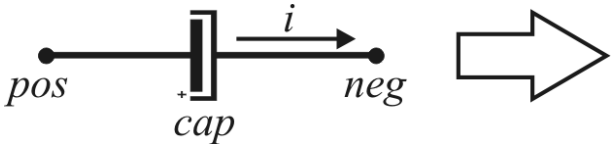

(c)

Figura 5.4 – Representação (a), descrição comportamental (b) e resultados de simulação (c) de um indutor ideal.

## 5.1.4. Descrição Comportamental do Capacitor

O comportamento físico de um capacitor é descrito através de uma equação fundamental, que diz que a corrente no capacitor é igual a sua capacitância em Faraday, multiplicada pelo valor da derivada da tensão que o atravessa. Esse comportamento pode ser observado na descrição da Figura 5.5b, na linha 10. A Figura 5.5 ilustra o capacitor, sua descrição

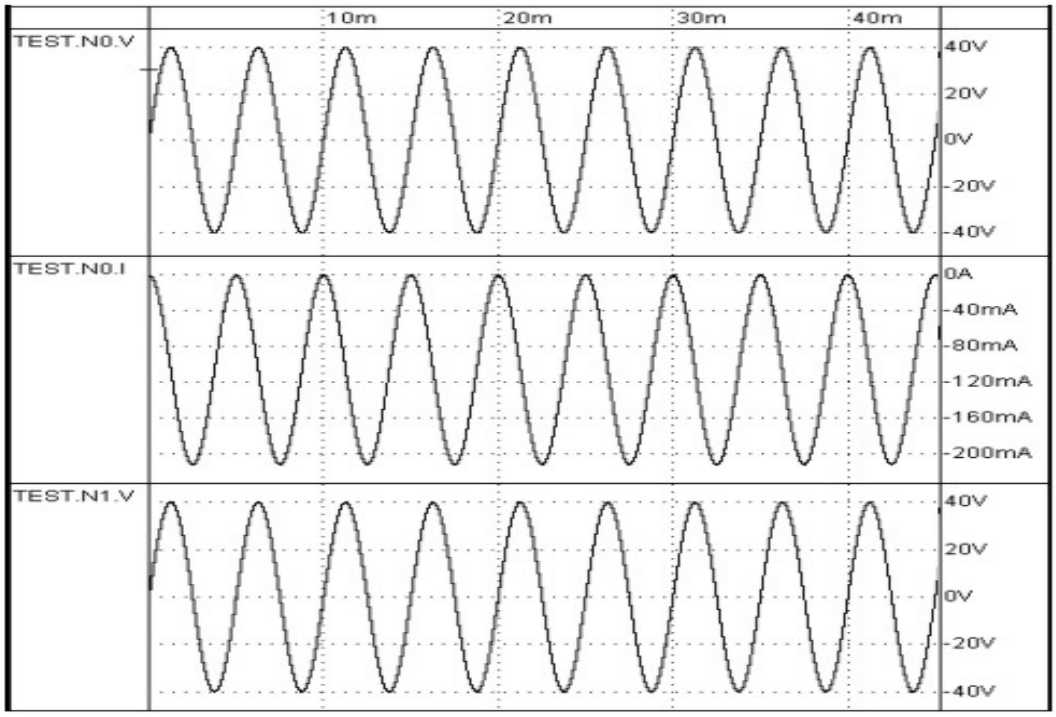
e seus resultados de simulação quando submetido à fonte de tensão CA da Figura 5.2. Como podemos observar na Figura 5.5c, o sinal de entrada TEST.N0.V é introduzido no capacitor com a capacitância genérica descrita na linha 4. Tem-se então uma corrente adiantada em 90º e uma tensão em fase com o sinal de entrada, como pode ser observado nos sinais TEST.N0.I e TEST.N1.V.




(a)

```
1.  library ieee;
2.  use ieee.electrical_systems.all;

3.  entity capacitor is
4.      generic ( cap    : capacitance : 10.0e-9 );
5.      port (   terminal pos, neg : electrical);
6.  end entity capacitor;

7.  architecture behavorial of capacitor is
8.  quantity v across i through pos to neg;
9.  begin
10.     i   == cap * v'dot;
11. end architecture ideal;
```

(b)

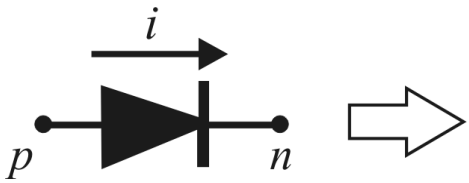



(c)

Figura 5.5 – Representação (a), descrição comportamental (b) e resultados de simulação (c) de um capacitor ideal.

## 5.1.5. Descrição Comportamental do Diodo

O diodo pode ser modelado de diversas maneiras. Nesse trabalho, está descrito um modelo comportamental de um diodo ideal, onde o semicondutor conduz somente se o nível de tensão em seus terminais for positivo.

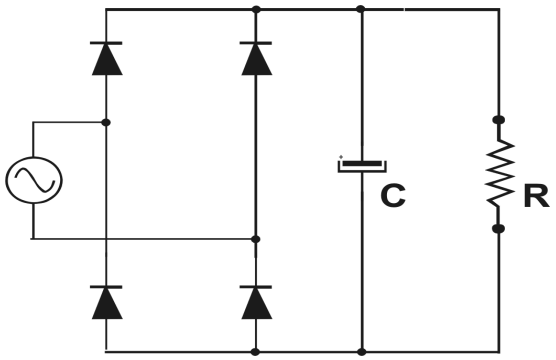



***Modelo do Diodo***

(a)

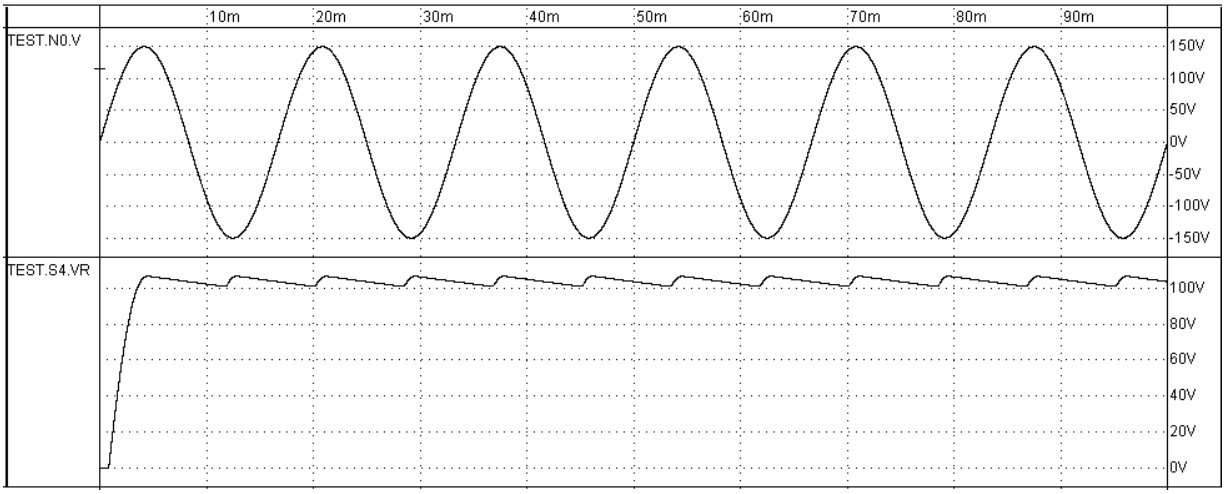
```
1.  library ieee;
2.  use ieee.math_real.all;
3.  use ieee.electrical_systems.all;

4.  entity diodo is
5.      generic (Isat   : current := 1.0e-14);
6.      port (     terminal p, n : electrical);
7.  end entity diodo;

8.  architecture behav of diodo is
9.  quantity v across i through p to n;

10. function limit_exp( x : real ) return real is
11. variable abs_x      : real := abs(x);
12. variable result     : real;
13. begin
14.     if abs_x < 100.0 then
15.             result := exp(abs_x);
16.     else
17.             result := exp(100.0) * (abs_x - 99.0);
18.     end if;
19.     if x < 0.0 then
20.             result := 1.0 / result;
21.     end if;
22.     return result;
23. end function limit_exp;

24. begin
25.     i == Isat*(limit_exp(v/0.6) - 1.0);
26. end architecture behav;
```

(b)




***Circuito utilizado na simulação***

(c)




(d)

Figura 5.6 – Representação (a) e descrição comportamental (b) de um diodo ideal, circuito utilizado para a simulação (c) e resultados para um retificador de onda completa.

Uma função foi criada para que fosse possível a utilização da função exponencial, que não aceita atributos negativos. A Figura 5.6 ilustra o diodo e sua descrição. Para exemplificar seu funcionamento, foi criado um retificador de onda completa com os componentes já criados anteriormente e com o diodo descrito. Os resultados de simulação estão na Figura 5.6c. O sinal TEST.N0.V é o sinal de entrada CA do retificador e o sinal TEST.S4.VR, a tensão de saída do circuito. A função *limit_exp* (da linha 10 à linha 23 da Figura 5.6b) confere se o valor de tensão é negativo, caso o seja, a função o transforma em um valor válido. O único atributo do diodo está descrito na linha 5 e define qual será a sua corrente de saturação.

## 5.2. Implementação dos Elementos de Controle

Após a modelagem dos componentes analógicos do sistema, é possível simular qualquer circuito que os utilizem em sua estrutura.

Para que seja viável a implementação de um controle digital de um sistema de potência, faz-se necessário o modelamento de blocos de interface entre as tecnologias digitais e analógicas, bem como funções aritméticas que viabilizem o controle.

No controle através de DSPs, devido a sua natureza matemática, as técnicas de controle digital (que utilizam operações aritméticas complexas) podem ser diretamente empregadas. No entanto, ao se utilizar sistemas digitais baseados em FPGA, é desejável que o controle seja composto por blocos mais simples, que exerçam a mesma função, porém com um circuito menos complexo. Isso é possível através da alta velocidade das atuais FPGAs[3]. Por exemplo, uma integral pode ser substituída por um somador acumulador com altíssima taxa de amostragem e com uma perda desprezível no rendimento do sistema. Nessa seção, serão descritos alguns circuitos digitais, que podem ser úteis no desenvolvimento de um controle, através de FPGA.

## 5.21. Descrição do Conversor A/D

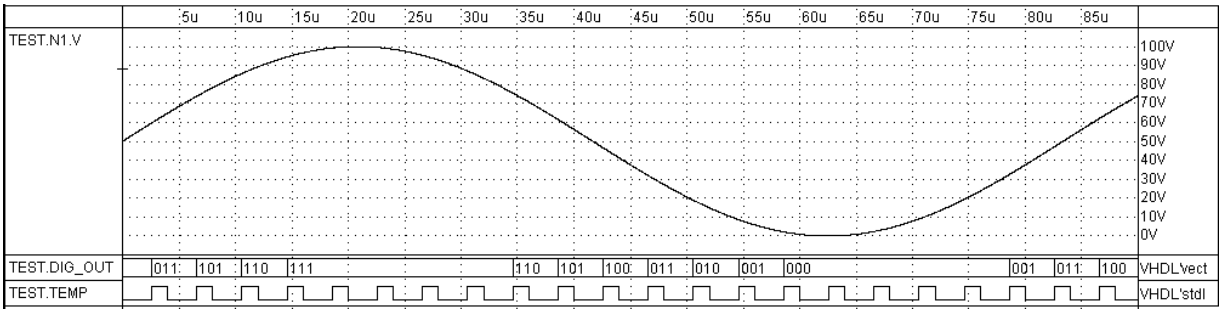
O conversor A/D é um componente externo ao controle digital, porém essencial, já que as grandezas do conversor estático são todas analógicas. A precisão e a velocidade de aquisição do conversor A/D devem ser bem analisadas e deve haver um compromisso entre o custo e o benefício proporcionado. Os resultados de simulação do controle com a utilização de conversores A/D podem ser minuciosamente estudados antes de se adquirir um conversor comercial. Quanto mais preciso e mais rápido for o conversor, mais preciso será o controle. Entretanto, os conversores de alta precisão e alta taxa de amostragem têm um custo bastante elevado e exigem circuitos lógicos mais complexos.

Diversos algoritmos de conversão de sinais analógicos para digitais podem ser utilizados [38][39][40]. Neste trabalho, foi escolhido um conversor que trabalha através de aproximações sucessivas do valor amostrado.

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.  use ieee.std_logic_arith.all;
4.  use ieee.electrical_systems.all;

5.  entity a2d_nbit is
6.      generic (Vmax  : voltage    := 100.0;
7.                Nbits : integer := 3;
8.                delay : time := 4us);
9.
10.     port (    signal   start : in  std_logic;     -- Amostragem
11.               signal   clk   : in  std_logic;     -- Clock
```

```
12.                terminal ain   :    electrical;    -- Entrada Analógica
13.                signal   eoc  : out std_logic;  -- Saída digital de final de conversão
14.                signal   dout  : out std_logic_vector(0 to (nbits-1)));  -- Saída Digital de nbits dígitos
15.  end entity a2d_nbit;

16.  architecture sar of a2d_nbit is
17.  type fase is (entrada, conversao,saida);  -- Tipo fases definindo as 3 fases do algoritmo
18.  constant bit_range : integer := Nbits-1;  -- constante para representar os nbits
19.  quantity Vin across Iin through ain to electrical_ref;

-- Começo do algoritmo
20.  begin

21.        sa_adc : process
22.                variable thresh  : voltage    := Vmax;
23.                variable Vtmp    : voltage;
24.                variable dtmp    : std_logic_vector(0 to (Nbits-1));
25.                variable status  : states  := input;
26.                variable bit_cnt : integer := Nbits -1;
27.                begin
28.                case status is
29.                        when input   =>            -- Le o sinal de entrada quando start é alto
30.                                wait on start until start = '1';
31.                                thresh         := Vmax;
32.                                Vtmp           := Vin;
33.                                eoc <= '0';
34.                                status         := convert;     -- Vai para a etapa de conversão
35.                        when convert =>            -- Começa a conversão
36.                                thresh         := thresh / 2.0;  -- Pega o valor do bit mais significativo
37.                                wait on clk until clk = '1';
38.                                if Vtmp > thresh then
39.                                        dtmp(bit_cnt) := '1';
40.                                        Vtmp         := Vtmp - thresh;
41.                                else
42.                                        dtmp(bit_cnt) := '0';
43.                                end if;
44.                                bit_cnt        := bit_cnt - 1;
45.                                if (bit_cnt + 1) < 1 then
46.                                        status       := output;      -- Após o término da conversão vai para saída
47.                                end if;
48.                        when output  =>  -- Espera a habilitação da saída
49.                                wait for delay/2;
50.                                eoc <= '1';   -- sinal de final de conversão vai para alto
51.                                for i in bit_range downto 0 loop
52.                                        dout(i)      <= dtmp(i);
53.                                end loop;
54.                                bit_cnt        := bit_range;
55.                                status         := input;       -- Pronto para próxima conversão
56.                end case;
57.        end process sa_adc;
-- Fim do algoritmo

58.        Iin == 0.0;       --  Para um A/D ideal tem-se a corrente nula

59.  end architecture sar;
```

**(a)**




**(b)**

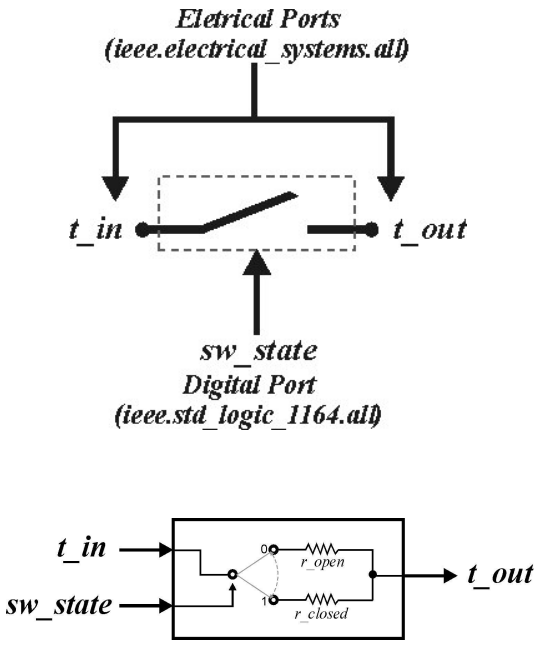
Figura 5.7 – Descrição comportamental (a) e resultados de simulação (b) de um Conversor A/D

A Figura 5.7 apresenta um conversor A/D de 3 bits de resolução e com um tempo entre as amostragem de 4μs. Para alterar a sua resolução e taxa de amostragem, deve-se trocar os valores iniciais (linha 6 à linha 8). O algoritmo é baseado em divisões sucessivas e comparações com o máximo valor da tensão de entrada. Na Figura 5.7a, o algoritmo é detalhado através de comentários, e seu resultado de simulação encontra-se na Figura 5.7b. O sinal TEST.N1.V é o sinal analógico a ser digitalizado, o barramento de saída do conversor está ilustrado pelo sinal TEST.DIG_OUT e o sinal TEST.TEMP indica o final da conversão. Observa-se, ainda, que algoritmo de divisões sucessivas, apresentado na Figura 5.7a, não prevê em sua entrada uma tensão negativa.

## 5.2.2. Descrição da Chave Digital

Para que seja feita a conexão entre circuitos analógicos e circuitos digitais, faz-se necessário o modelamento de um componente com característica mista. A chave digital é acionada por um tipo lógico digital e cria um curto-circuito entre os terminais elétricos analógicos.

A Figura 5.8 apresenta a descrição de uma chave digital ideal. Através dessa descrição, pode-se observar que os terminais *t_in* e *t_out* são definidos como grandezas elétricas (linha 9) e o acionamento do interruptor é definido como uma grandeza digital (linha 8). Essa característica proporciona ao componente a capacidade de controle digital externo, que pode ser através de um gerador de pulsos PWM ou de uma malha de controle digital descrita em VHDL.




(a)

```
 1. library ieee;
 2. use ieee.electrical_systems.all;
 3. use ieee.std_logic_1164.all;

 4. entity switch_digital is
 5.       generic (r_open     : resistance := 1.0e6;
 6.                 r_closed   : resistance := 1.0;
 7.                 transient_time     : real      := 1.0e-6);
 8.       port    (  sw_state          : in std_logic;
 9.                 terminal t_in, t_out     :    electrical);
10. end entity switch_digital;

11. architecture behav of switch_digital is
12. signal   r_signal : resistance := resistance_open;
13. quantity r        : resistance;
14. quantity v across i through t_in to t_out;
15. begin
16.       switch_state: process (sw_state)
17.       begin
18.                 if (sw_state'event and sw_state = '0') then
19.                           r_signal <= r_open;
20.                 elsif (sw_state'event and sw_state = '1') then
21.                           r_signal <= r_closed;
22.                 end if;
23.       end process switch_state;
24.       r == r_signal'ramp(transient_time, transient_time);
25.       v == r *  i;
26. end architecture behav;
```

(b)

Figura 5.8 – Descrição comportamental de uma chave ideal.

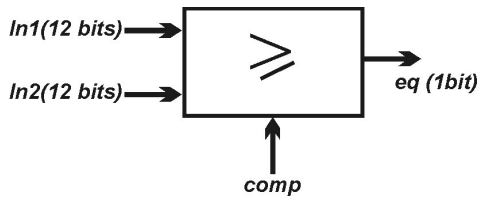
## 5.2.3. Descrição do Comparador Lógico

Freqüentemente, em sistemas de controle, faz-se necessário a comparação entre dois sinais analógicos distintos. O circuito que oferece tal funcionalidade é o comparador lógico.

Na digitalização dos sinais, utilizam-se conversores A/D com a precisão adequada ao sistema de controle e nem sempre os conversores têm a mesma resolução. Na descrição do comparador lógico, é possível comparar sinais com diferentes resoluções, desde que se faça a concatenação de níveis lógicos baixos ao sinal de menor resolução. Ao utilizar a biblioteca

*std_logic_arith* pode-se, ainda, efetuar comparações de valores considerando o seu sinal (*signed*) ou não considerando o sinal (*unsigned*).

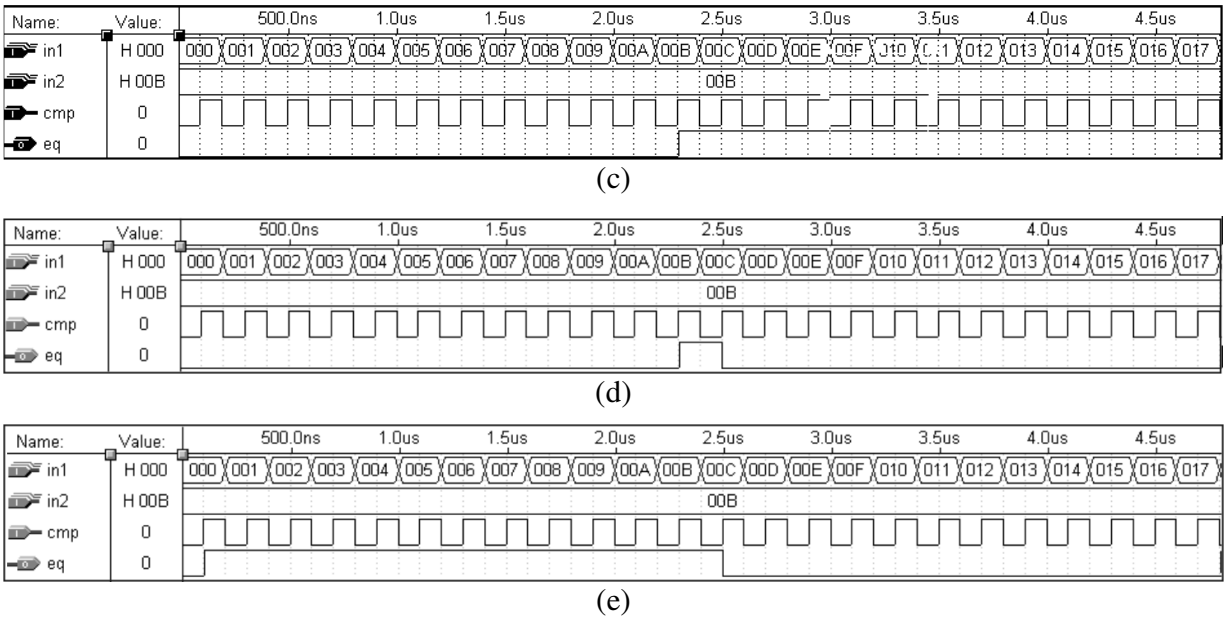
Na descrição da Figura 5.9a, tem-se um comparador de dois barramentos de 12 bits e terá como saída o nível lógico alto se *in1* for maior ou igual a *in2*. Para modificar a descrição, basta modificar o sinal da linha 20 para o sinal de igualdade (=) ou o sinal que represente a relação menor ou igual que (<=). A Figura 5.9b ilustra a descrição do comparador de 12 bits, na Figura 5.9c o resultado de simulação para o comparador, onde o sinal *eq* tem nível lógico alto se *in1* for maior ou igual a *in2*; a Figura 5.9d ilustra o comparador, onde o sinal *eq* tem nível lógico alto se *in1* for igual a *in2*; a Figura 5.9e ilustra o comparador onde o sinal *eq* tem nível lógico alto se *in1* for menor ou igual a *in2*. Em ambos resultados de simulação a comparação é feita através de uma transição do nível lógico baixo para o alto do sinal *comp*.




(a)

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.  use ieee.std_logic_arith.all;

4.  entity comp_20b is
5.  generic (Nbits :    integer   := 12);
6.  port(          eq        : out std_logic := '0';
7.       in1       : in  std_logic_vector (0 to Nbits-1);
8.       in2       : in  std_logic_vector (0 to Nbits-1);
9.       cmp       : in  std_logic := '0' );
10. end entity comp_20b;

11. architecture simple of comp_20b is
12. begin
13. compare : process (cmp)
14. variable in1_hold : std_logic_vector (0 to (Nbits-1));
15. variable in2_hold : std_logic_vector (0 to (Nbits-1));
16. begin
17.      in1_hold := in1;
18.      in2_hold := in2;
19.      if cmp'event and cmp = '1' then
20.               if signed(in1_hold) >= signed(in2_hold) then
21.                        eq    <= '1';
22.               else
23.                        eq <= '0';
24.               end if;
25.      end if;
26. end process;
27. end architecture simple;
```

(b)




(c)

(d)

(e)

Figura 5.9 – Descrição comportamental (b) e resultados de simulação para um comparador de maior ou igual(c), igual a (d) e menor ou igual (e).

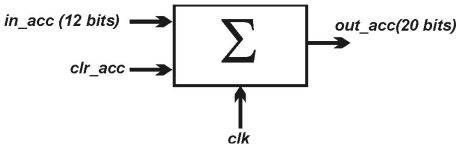
## 5.2.4. O Somador e o Somador Acumulador

O somador é uma das operações aritméticas mais importantes para o controle digital, pois pode-se produzir o resultado de uma série de somas num intervalo de tempo muito pequeno, apresentado assim um comportamento muito próximo de um integrador, que é um elemento largamente utilizado em estratégias de controle.

Tem-se para um somador simples um sinal em sua saída igual à soma aritmética de suas entrada, o componente pode ser descrito de forma comportamental ou estrutural. A biblioteca *ieee.std_logic_arith.all* contém operações aritméticas para tipos lógicos padrão, entretanto a qualidade de circuito lógico final vai depender da ferramenta de síntese utilizada.

A Figura 5.10 apresenta a descrição e os resultados de simulação para um somador acumulador, que tem como função somar e armazenar o sinal presente no barramento de entrada (*in_acc*) de 12 bits a cada sinal de sincronismo (*clock*). Como o sinal da entrada pode ser somado diversas vezes, o sinal de saída necessita ter uma quantidade de bits maior. Na descrição da figura tem-se o sinal no barramento de saída (*out_acc*) com 20 bits. Na linha 19, adicionou-se um ajuste

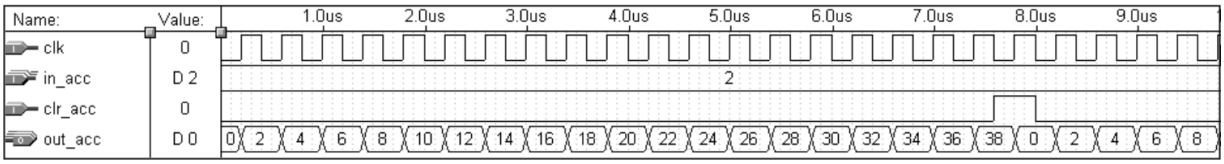
para que a soma de barramentos com quantidades diferentes de dígitos (*bits*) pudesse ser efetuada.




(a)

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.  use ieee.std_logic_arith.all;

4.  entity  sumacc is
5.      port(     clk,clr_acc  :in std_logic;
6.                in_acc :in std_logic_vector  (11 downto 0);
7.                out_acc  :out std_logic_vector (19 downto 0));
8.  end sumacc;


9.  architecture acumulador of sumacc is
10. begin
11. process
12. variable tempa, tempb   : std_logic_vector (19 downto 0);
13.     begin
14.               wait until clk'event and clk = '1';
15.               if clr_acc='1' then
16.                         out_acc <= (others => '0');
17.                         tempa := "00000000000000000000";
18.               else
19.                         tempb := "00000000" & in_acc;
20.                         tempa := (signed(tempa)) + (signed(tempb));
21.                         out_acc <= tempa (19 downto 0);
22.               end if;
23.     end process ;
24. end acumulador;
```

(b)



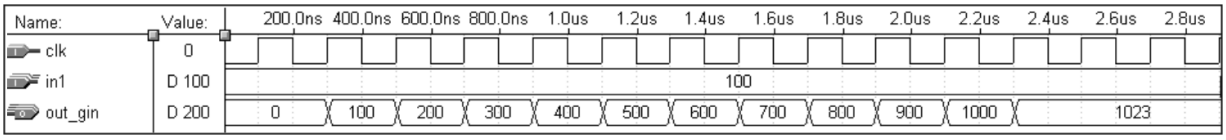

(c)

Figura 5.10 – Descrição comportamental (b) e o resultado de simulação (c) de um somador acumulador.

Se for necessário que o somador, em determinado valor, pare de efetuar a sua operação, soma-se à descrição uma estrutura condicional, conforme ilustrado nas linhas de 17 a 21 da Figura 5.11. Nesse exemplo, o valor máximo permitido pelo somador é 1023; qualquer soma efetuada, após esse valor, será descartada. A Figura 5.11b ilustra o resultado de simulação para o somador acumulador com checagem de valor máximo (*range checker*).

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.  use ieee.std_logic_arith.all;

4.  entity  sumacc_rc is
5.      port(     clk  :in std_logic;
6.                in1  :in std_logic_vector  (11 downto 0);
7.                out_gin  :out std_logic_vector (11 downto 0));
8.  end sumacc_rc;


9.  architecture range_checker of sumacc_rc is
10. begin
11. process
12. variable tempa, tempb   : std_logic_vector (11 downto 0);
13.     begin
14.               wait until clk'event and clk = '1';
15.               tempb := in1;
16.               tempa := (signed(tempa)) + (signed(tempb));
17.               if tempa < "001111111111" then
18.                         out_gin <= tempa (11 downto 0);
19.               else
20.                         out_gin <= "001111111111";
21.               end if;
22. end process ;
23. end range_checker;
```

(a)

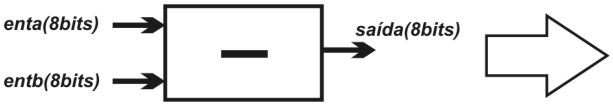



(b)

Figura 5.11 – Descrição comportamental (a) e o resultado de simulação (b) de um somador acumulador com checagem de valor máximo permitido.

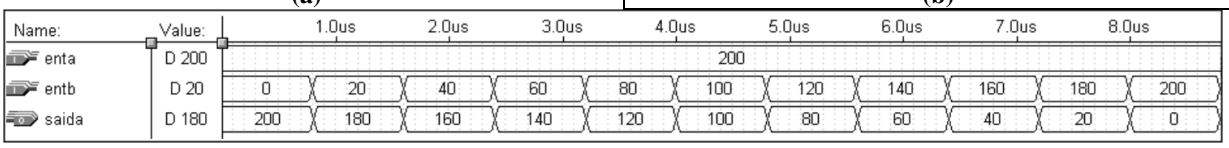
## 5.2.5. O Subtrator

Assim como o somador, o subtrator também pode ser descrito de forma comportamental ou estrutural. A figura 5.12 apresenta um subtrator de 8 dígitos binários, não sinalizados e descrito de forma comportamental. Para descrever um subtrator de 8 bits sinalizados, basta, na linha 14 da Figura 5.12a, substituir a diretiva *unsigned* (não sinalizado) pela diretiva *signed* (sinalizado), uma vez que utiliza-se o tipo de dado *std_logic*.




```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.  use ieee.std_logic_arith.all;

4.  entity sub8b is
5.  port (
6.  enta: in std_logic_vector (7 downto 0);
7.  entb: in std_logic_vector (7 downto 0);
8.  saida: out std_logic_vector (7 downto 0));
9.  end sub8b;

10. architecture arch of sub8b is
11. begin
12. process
13. begin
14.     saida  <= unsigned(enta) - unsigned(entb);
15. end process;
16. end arch;
```

**(a)** **(b)**




**(c)**

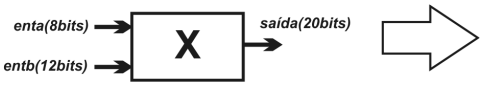
Figura 5.12 – Descrição comportamental (b) e o resultado de simulação (c) de um subtrator de 8 bits.

## 5.2.6. O Multiplicador

Outro componente importante na implementação de estratégias de controle é o multiplicador. Devido ao fato do multiplicador ser um elemento que necessita de uma grande quantidade de portas lógicas para ser implementado, algumas simplificações podem ser bastante convenientes. Um deslocamento para a esquerda em uma palavra binária proporciona a mesma função de uma multiplicação por dois. Desta maneira, dois deslocamentos para a esquerda proporcionam uma multiplicação por quatro e assim por diante. A substituição de multiplicadores por deslocadores é muito utilizada [3], mas deve ser usada com cautela para que o resultado final não seja alterado.

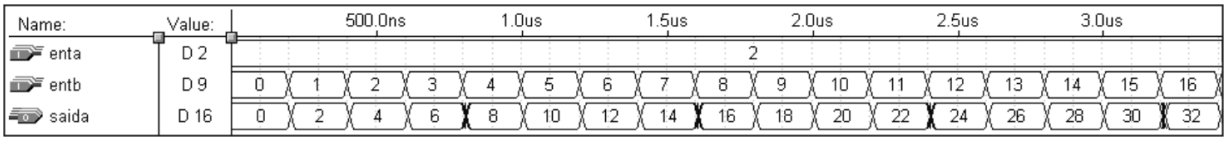
Na figura 5.13b, tem-se um multiplicador descrito de forma comportamental através do operador vezes (*) da biblioteca *std_logic_arith*, no entanto essa descrição pode não se mostrar eficiente, dependendo da ferramenta de síntese utilizada. As ferramentas de síntese mais

modernas têm conseguido diminuir consideravelmente a complexidade do circuito final. Na Figura 5.13b, observa-se ainda que as entradas têm diferentes quantidades de bits, as linhas 14 e 15 fazem o acerto necessário à operação.




```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.  use ieee.std_logic_arith.all;

4.  entity mult is
5.  port(       enta:  in std_logic_vector (7 downto 0);
6.              entb:  in std_logic_vector (11 downto 0);
7.              saida: out std_logic_vector (19 downto 0));
8.  end mult;

9.  architecture arch of mult is
10. begin
11.     process
12.     variable temp,tempa,tempb : std_logic_vector(19 downto 0);
13.     begin
14.             tempa  := "000000000000" & enta;
15.             tempb  := "00000000" & entb;
16.             temp   := unsigned(tempa) * unsigned(tempb);
17.             saida  <= temp(19 downto 0);
18.     end process;
19. end arch;
```

**(a)** (b)

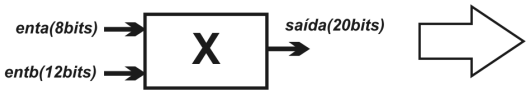



(c)

Figura 5.13 – Descrição comportamental (b) e o resultado de simulação (c) de um multiplicador de 8 bits por 12 bits.

Outra opção de multiplicador é apresentada na Figura 5.14. Tem-se um multiplicador baseado em deslocamentos, cada dígito do dado de entrada *shift* representa uma multiplicação por um múltiplo de dois, multiplicações estas somadas através do algoritmo. Nem todas as multiplicações são possíveis, no entanto, pode-se obter uma boa aproximação comparada à simplicidade de seu circuito lógico final. Os múltiplos possíveis são dois, quatro, oito e dezesseis e suas possíveis combinações. Por exemplo, o dado binário "0011" tem uma multiplicação por 2 ("0001") e uma multiplicação por 4 ("0010"), somando as duas

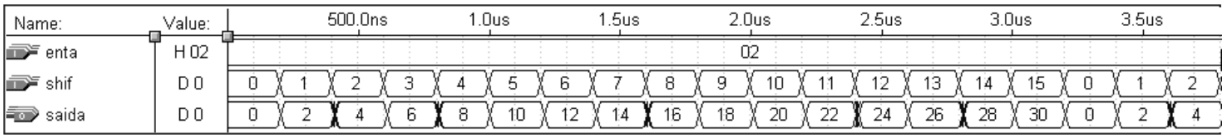
multiplicações tem-se uma multiplicação final por 2+4=6. A Figura 5.14b mostra o resultado da simulação da descrição em questão.




(a)

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.  use ieee.std_logic_arith.all;

4.  entity desl_b2 is
5.  port (          enta:  in std_logic_vector (7 downto 0);
6.      shif:  in std_logic_vector (3 downto 0);
7.      saida: out std_logic_vector (11 downto 0));
8.  end desl_b2;

9.  architecture arch of desl_b2 is
10. begin
11.     process
12.     variable temp,tempa : std_logic_vector(11 downto 0);
13.     begin
14.             if enta(7)='0' then
15.                     tempa := "0000" & enta;
16.             else
17.                     tempa := "1111" & enta;
18.             end if;
19.             if shif(0)='1' then
20.                     tempa := tempa(10 downto 0) & '0';
21.             end if;
22.             if shif(1)='1' then
23.                     tempa := tempa(9 downto 0) & "00";
24.             end if;
25.             if shif(2)='1' then
26.                     tempa := tempa(8 downto 0) & "000";
27.             end if;
28.             if shif(3)='1' then
29.                     tempa := tempa(7 downto 0) & "0000";
30.             end if;
31.             saida <= tempa;
32.     end process;
33. end arch;
```

(b)




(c)

Figura 5.14 – Descrição comportamental (b) e o resultado de simulação (c) de um multiplicador utilizando deslocadores.

## 5.3. Descrição Estrutural do Conversor Buck

Com todos os componentes necessários à construção do conversor BUCK da Figura 5.15, modelados, faz-se a descrição estrutural do mesmo. Tem-se na Figura 5.16a, implementado cada nó do circuito do conversor discutido no Capítulo 2, através de processos (*process*) e instancia seus componentes por intermédio do mapeamento de portas (*port map*). As linhas de 9 a 14 definem os terminais necessários à interconexão dos processos e são transferidos entre os nós através do mapeamento das portas. A Figura 5.16b mostra o resultado de simulação para os parâmetros determinado em sua descrição.

Tem-se ainda, na Figura 5.16b, o sinal TEST.N10.V, que é a tensão de saída do conversor CA/CC; o sinal TESTE.N0.V, que é a tensão de entrada do conversor e o sinal TEST.N10.I, que é a corrente de saída do conversor.

Para o conversor apresentado na Figura 5.16, o controle é feito através de uma modulação por largura de pulso (PWM), que é gerada através de um processo (linhas 16 a 22). O controle PWM é projetado para cada conversor e pode ter modificado seu ciclo de trabalho através das linhas 19 e 21. Observa-se que o tipo de sinal empregado à chave digital (*switch_dig*) é do tipo lógico padrão (*std-logic*). Numa implementação de uma estratégia de controle digital, a estrutura de controle seria aplicada a esse terminal digital.

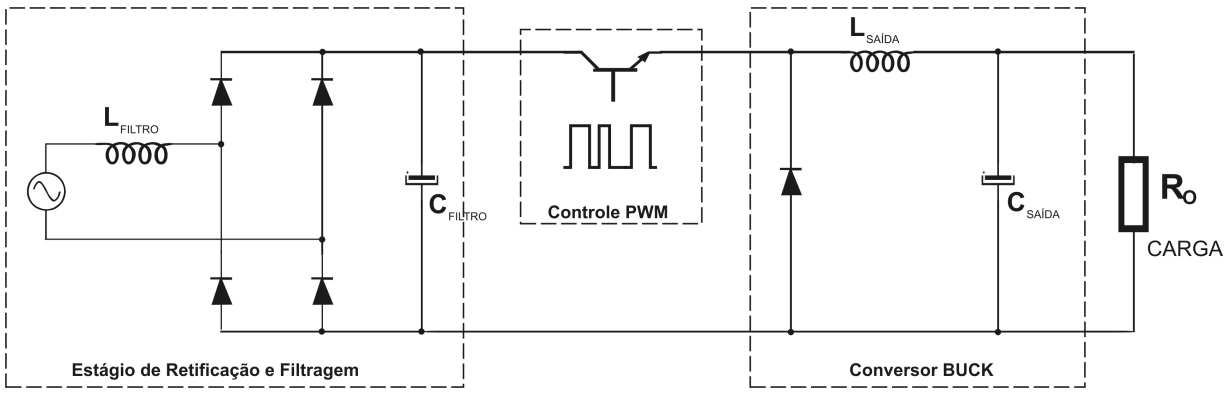
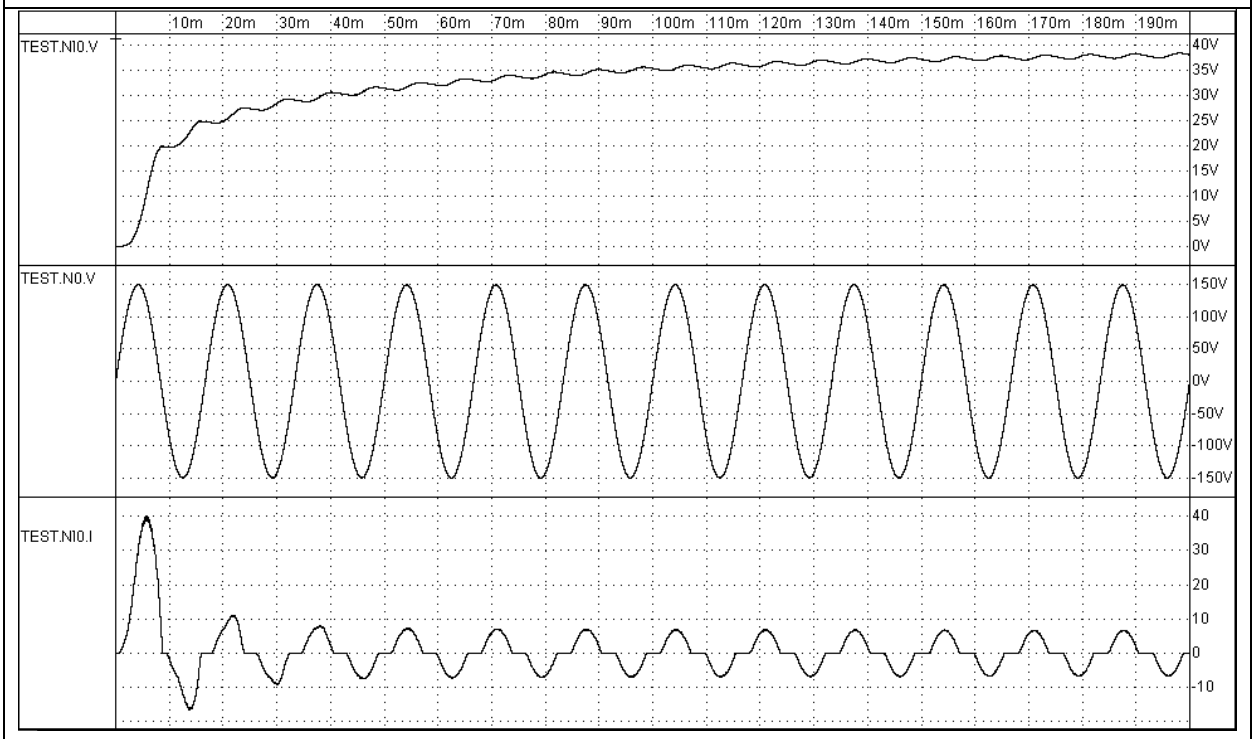



Figura 5.15 – O Conversor CA/CC tipo BUCK

```
 1. library IEEE;
 2. use IEEE.electrical_systems.all;
 3. use IEEE.std_logic_1164.all;
 4. use work.all;

 5. entity Test is
 6. end Test;

 7. architecture Top of  Test  is
 8. signal  clk_out : std_logic;
 9. terminal AVHDL: electrical;
10. terminal BVHDL: electrical;
11. terminal CVHDL: electrical;
12. terminal DVHDL: electrical;
13. terminal EVHDL: electrical;
14. terminal FVHDL: electrical;
15. begin

16.     CreateClock: process
17.             begin
18.                     clk_out <= '0';
19.                     wait for 102us ;        ---------- TEMPO DO CICLO EM ZERO
20.                     clk_out <= '1';
21.                     wait for 64us ;         ---------- TEMPO DO CICLO EM UM
22.             end process CreateClock;

23.     N0:     entity v_sine(behav)
24.             generic map(freq => 60.0, amplitude => 150.0)
25.             port map(pos =>AVHDL, neg=>FVHDL);

26.     N1:     entity indutor(behav)
27.             generic map (ind => 3.85e-3)
28.             port map(pos=> AVHDL, neg=> ZVHDL);

29.     N2:     entity diodo(behav)
30.             port map(p=>ZVHDL, n=>EVHDL);

31.     N3:     entity diodo(behav)
32.             port map(p=>ground, n=>ZVHDL);

33.     N4:     entity diodo(behav)
34.             port map(p=>FVHDL, n=>EVHDL);

35.     N5:     entity diodo(behav)
36.             port map(p=>ground, n=>FVHDL);

37.     N6:     entity capacitor(behav)
38.             generic map (cap => 18.0e-6)
39.             port map(pos=> EVHDL, neg=>ground);

40.     N7:     entity switch_dig(behav)
41.             port map(sw_state => clk_out, pos=> EVHDL, neg=> CVHDL);

42.     N8:     entity indutor(behav)
43.             generic map (ind => 156.0e-6)
```

```
44.             port map(pos => CVHDL, neg =>BVHDL);

45.     N9:     entity diodo(behav)
46.             port map(p=>ground, n=>CVHDL);

47.     N10:    entity resistor(behav)
48.             generic map (R => 7.2)
49.             port map (pos => BVHDL, neg => GROUND);

50.     N11:    entity capacitor(behav)
51.             generic map (cap => 24.0e-3)
52.             port map(pos=>BVHDL, neg => ground);

end Top;
```

(a)




(b)

Figura 5.16 – Descrição (a) e resultados de simulação (b) para do conversor

A conexão entre os componentes que compõem o conversor BUCK é realizada através de terminais elétricos, os quais estão descritos das linhas 9 a 14. Esses terminais têm como função a transferência das características elétricas entre os processos, que são concorrentes. Os processos de *N0* a *N11* definem como cada elemento do sistema está disposto na estrutura e são executados de forma paralela, ou seja, as mudanças nos parâmetros elétricos são

compartilhadas entre em tempo real e não de forma seqüencial, proporcionando uma resposta dinâmica às alterações de comportamento.

Para modificar as características do conversor, basta, modificar os parâmetros genéricos dos elementos. Por exemplo, para modificar a tensão de entrada no estágio de retificação e filtragem, deve-se, na linha 24, alterar os parâmetros *amplitude* ou *freq* da fonte CA, instanciada no processo *N0* da linha 23.

A utilização de um controle digital para o BUCK proporcionaria um tempo de resposta mais rápido ao conversor do que o tempo apresentado no resultado de simulação da Figura 5.16b, através de TEST.N10.V, bem como uma corrente de entrada (ilustrada em TEST.N10.I) com as deformidades muito reduzidas em relação ao controle por modulação por largura de pulso.

Uma melhoria que poderia ser acrescentada com o controle através de um sistema digital baseado em FPGA seria a introdução de dispositivos de segurança que não permitiriam que o sistema alcançasse níveis críticos de tensões e correntes.

Pode-se, então, com os componentes desenvolvidos anteriormente, implementar uma estratégia de controle digital para qualquer estrutura analógica, através da descrição estrutural de seus elementos analógicos e digitais. Uma vez testado, validado e simulado o resultado do controle digital é então transferido a um FPGA e, finalmente, confeccionado o protótipo do sistema.

# Capítulo 6

# *Conclusões e Trabalhos Futuros*

Esta dissertação apresenta uma metodologia de modelamento de conversores de potência, baseada na descrição de circuitos, utilizando a linguagem VHDL-AMS. Esta metodologia permite a descrição e simulação do código de controle digital, em VHDL-AMS, em conjunto com os componentes analógicos do conversor.

A simulação de estruturas, empregando a descrição em MATLAB®, é muito utilizada na área de eletrônica de potência. Entretanto, esse ambiente não foi desenvolvido com o objetivo de suportar os dispositivos lógicos programáveis (CPLDs e FPGAs), embora existam atualizações que permitam a síntese através dessa ferramenta.

A metodologia apresentada nesta dissertação demonstra que é possível descrever os blocos analógicos que compõem um sistema de potência, como os conversores de potência, em conjunto com circuitos digitais com descrições que podem ser diretamente transportadas para um ambiente de síntese de dispositivos lógico programáveis.

A utilização da linguagem VHDL-AMS facilita a descrição de circuitos analógicos, uma vez que mesmo sem o modelamento ou equacionamento de um circuito, através da descrição comportamental, pode-se simular a operação de circuitos que deverão se comunicar com o circuito digital, que será sintetizado em um dispositivo lógico programável. Torna-se possível assim a comunicação do circuito digital de controle com os componentes do sistema analógico, permitindo uma simulação próxima da situação real de operação, diminuindo assim as chances de operação incorreta do circuito digital na implementação física do sistema.

Assim é possível reduzir o tempo de desenvolvimento do sistema total, bem como os custos finais do projeto.

Uma outra vantagem da metodologia de desenvolvimento, utilizando a linguagem VHDL-AMS, é a possibilidade de transporte imediato da descrição do circuito para o projeto de um circuito integrado de aplicação específica (ASIC), uma vez que a linguagem VHDL, utilizada como entrada em ambientes de síntese de circuitos integrados, como Mentor Graphics® e Cadence®, permitirá a obtenção de um circuito integrado com uma substituição simples da biblioteca de células básicas. No ambiente de síntese de circuitos integrados, a biblioteca não estará mais baseada em blocos lógicos como é o caso de dispositivos lógicos programáveis, e sim estará baseada em células como flip-flops, portas lógicas básicas, células de memórias, etc.

Este trabalho é pioneiro na Universidade Federal de Itajubá, considerando que a área de eletrônica de potência não utilizava a descrição de circuitos com VHDL-AMS como uma opção de projeto. Acredita-se que uma continuação natural para este trabalho, seria a implementação completa de um circuito de controle digital de um conversor estático de potência, uma vez que no modelamento apresentado nessa dissertação, o controle digital foi substituído por um sinal PWM.

# Apêndice

## *Artigo Publicado*

Um artigo relacionado a este trabalho foi publicado no 8º Congresso Brasileiro de Eletrônica de Potência (COBEP), na cidade de Recife, no Brazil em junho de 2005.

- ✓ A methodology for the Development of the Digital Control Of Static Converters Using FPGA

O citado artigo encontra-se em anexo.

# *Referências bibliográficas*


[1] Alex Doboli and Ranga Vemuri, “Behavioral Modeleing fo High-Level Sysnthesis of Analog and Mixed-Signal Systems From VHDL-AMS”, IEEE Transactions On Computer Aided Design fo Integrated Circuits anda Systems, Vol. 22, No. 11, pp.1504-1520, November 2003

[2] G. Gielen and R. Rutenbar, “Computer-aided design of analog and mixed-signal integrated circuits”, Proc. IEEE, vol. 88, pp. 1825-1854, December 2000.

[3] A. de Castro, P. Zumel, O. García, T. Riesgo and J. Uceda, “Concurrent and Simple Digital Controller of an AC/DC Converter with Power Factor Correction”, IEEE Trans. Ind. Electron., vol. 46, pp. 3-12, Fevereiro de 1999.

[4] François Pêcheux, Christophe Lallement and Alain Vachoux, “VHDL-AMS and Verilog-AMS as Alternative Hardware Description Language for Efficient Modeling of Multidiscipline Systems”, IEEE Transaction on Computer-Aided Design of Intregated Circuits and Systems, Vol. 24, pp. 204-225, February 2005.

[5] Teresa Riesgo, Yago Torroja and Eduardo de la Torre, “Design methodologies Based on Hardware Description Languages”, IEEE Transaction on Industrial Electronics, Vol. 46, No.1, pp. 3-12, February 1999.

[6] J. van den Keybus, B. Bolsens, K. De Brabandere, J. Driesen, R. Belmans, “DSP and FPGA based platform for rapid prototyping of power electronic converters and its application to a sampled-data three-phase dual-band hysteresis current controller”, IEEE Power Electronics Specialists Conference, Vol. 4,pp. 23-27 June 2002.

[7] M. Fu, Q. Chen, “A DSP Based Controller for Power Factor Correction (PFC) in a Rectifier Circuit”, IEEE Applied Power Electronics Conference and Exposition, Vol. 1, pp. 144 – 149,

March 2001.

[8] S. Hauck, "The roles of FPGAs in reprogrammable systems**",** Proceedings of the IEEE, Vol. 86, pp. 615-638, April 1998.

[9] A. M. Wu, J. Xiao, D. Markovic, S. Sanders, "Digital PWM Control: Application in Voltage Regulation Modules", IEEE Power Electronics Specialists Conference, Vol.1, pp. 77-83, June 1999.

[10] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993.

[11] IEEE Standard VHDL Analog and Mixed-Signal Extensions, IEEE Std 1076.1-1999.

[12] Dirk Lindeke, "Projeto de um Filtro Ativo Paralelo de 1KVA Usando Técnicas de Controle Analógico e Digital", Dissertação de Mestrado, Fevereiro de 2003.

[13] Aloísio de Oliveira, José C. de Oliveira, Anderson L. A. Vilaça, Anésio L. F. Filho, "Uma Contribuição para a Quantificação e Qualificação da Distorção Harmônica", Congresso Brasileiro de Eletrônica de Potência, pp. 665-670, 1997.

[14] I. Barbi, F. Pöttker de Souza, "A Unity Power Factor Buck Pre-Regulator with Feedforward of the Output Inductor Current", IEEE Applied Power Electronics Conference (APEC), 1999.

[15] C.S Silva, "Power Factor Correction with UC3854", Unitrode Application Note U-125, 1990.

[16] D. Paice, "Power Electronics Converter Harmonic Multipulse Methods for Clean Power", IEEE Press, 1996.

[17] F.J.M. Seixas, I. Barbi, "A New 18-Pulse AC-DC Converter with Regulated DC Output and High Power Factor for Three-Phase Applications", Congresso Brasileiro de Eletrônica de Potência, pp. 582-587, 1999.

[18] Integrated Circuit UC3854, Product & Application Handbook 1995, Unitrode.

[19] Y. T. Feng, G. L. Tsai, Y. Y. Tzou, "Digital control of Single-State Single Switch Flyback PFC AC/DC Converter with Fast Dynamic Response", IEEE Power Electronics Specialists Conference, 2001.

[20] Verilog-A Language Reference Manual – Analog Extensions to Verilog HDL Version 1.0, 1996.

[21] D. D. Gajski, R Kuhn, "Guest editors' introduction: New VLSI tools.", IEEE Computer, Vol. 16, pp. 11-14, December 1983.

[22] E. de la Torre, T. Riesgo, J. Uceda, E. Macip, M. Rizzi, "Highly Configurable Control Boards: a tool and a design experience", Int. Workshop on rapid system Prototyping, June 2000.

[23] M. Glesner, A. kirschbaum, "State-of-the-art in rapid prototyping", XI Brazilian Symposium on Integrated Circuit Design,pp. 60-65, October 1998.

[24] L. Lindqvist, “Evaluating the applicability of current VHDL synthesis tools to an industrial top-down development procedure”, Proc. VHDL Int. User’s Forum, pp. 1-7, October 1993.

[25] A. A. L. Ribeiro, “Reconfigurabilidade dinâmica e emota da FPGA’s”, Dissertação de Mestrado, Univerdade de São Paulo, São Carlos, Julho de 2002.

[26] H. Wang, S. B. K. Vrudhula, “Behavioral Synthesis of Field Programmable Analog Array Circuits”, ACM Transactions on Design Automation of Electronic Systems, Vol. 7, No. 4, pp. 563-604, October 2002.

[27] P. Gulak, “Field programmable Analog Arrays: Past, Present and Future perspectives”, Int. Conference on Microelectronics and VLSI, pp. 123-126, November 1995

[28] EPAC, “Electronically Programmable Analog Circuit”. IMP, inc., California, USA.

[29] H. Kutuk, S. M. Kang, “A Field Programmable Analog Array (FPAA) using Switched-Capacitor Techniques”, Int. Symposium on Circuit and System, pp. 41-44, May 1995.

[30] E. K. F. lee, P. Gulak, “A CMOS Field-Programmable Analog Array”, IEEE Solid State Circuit 26, pp. 1860-1867, February 1991.

[31] D. Anderson, C. Macjan, D. Bersch, H. Anderson, P. Hu, O. Palusinski, A. Gettman, I. Macbeth, A. Bratt, “A Field Programmable Analog Array and its Application”, proceedings of Custom integrated Circuits Conference, pp. 555-558, may 1997.

[32] O. A. Palusinski, D. Anderson, D. Gettman, C. Marcjan, H. Anderson, “Motorola Field Programmable Analog Arrays in Simulation, Control, and Circuit Design Laboratories.”,

[33] Y. Li, F. De bernardinis, B. Otis, J. M. Rabaey, A. S. Vincentelli, “A Low-Power Mixed-Signal baseband System Design for Wireless Sensor Networks”, IEEE Custom Integrated Circuits Conference, pp. 55-58, 2005.

[34] S. Ganesan, A. Nunez, N. Dhanwada, A. Dolobi, R. Vemuri, “Rapid Prototyping of Mixed Signal Systems from VHDL-AMS”.

[35] R. Lipsett, C. Schaefer, C. Ussery,”VHDL: Hardware Description and Design”, Kluwer Academic Publishers, 1989.

[36] A. Rushton, “VHDL for Logic Synthesis”, McGraw-Hill book Company, 1995.

[37] D. Damon, E. Christen, “Introduction to VHDL-AMS – Part 1: Structural and Discrete Concepts”, IEEE International Symposium on Computer-Aided Control System Design, pp. 264-269, September 1996.

[38] B. R. Stanisic, M. W. Brown, “VHDL Modeling for Analog-Digital Hardware Designs”, IEEE Int. Cof. Computer-Aided Design, 1989.

[39] M. Schubert, “VHDL based simulation of a sigma-delta A/D converter”, IEEE/ACM International Workshop on Behavioral Modeling and Simulation, pp. 71-76, October 2000.

[40] E. Peralías, A. J. Acosta, A. rueda, j. L. Huertas,“A VHDL-based Methodology for the Design and Verification of Pipeline A/D”, Proceedings of the conference on Design, automation and test in Europe, pp. 534-538, 2000.

# ANEXO

*Artigo publicado no COBEP 2005*

# A METHODOLOGY FOR THE DEVELOPMENT OF THE DIGITAL CONTROL OF STATIC CONVERTERS USING FPGA

Michel Santana, Enio Roberto Ribeiro, Robson Luiz Moreno
Itajubá Federal University
Av. BPS 1303 - Bairro Pinheirinho - Itajubá - Minas Gerais - Brazil
email: {michel;enio;moreno}@iee.efei.br

*Abstract* - **This article describes a methodology for the development of the digital control of power converters, where the whole structure – converter and control system – can be modeled in the same environment through the hardware description language VHDL. This language allows a behavioral description for the whole structure, so that the control system can be simulated, tested and validated without the need of a prototype. The portability of the VHDL language allows the implementation of the digital control code in an ASIC or FPGA. Simulations results show the feasibility of the methodology offering interesting possibilities in power converter control.**

*Keywords* - **ASIC, Static Converter, Digital Control, FPGA, VHDL-AMS.**

## I. INTRODUCTION

Currently there are many types of power converters being used and all of them require a control system for proper operation. The control system can be implemented with analog circuits [1]. They meet the needs of simple control systems (single input single output) and even complex control systems (two or more controlled variables), and are attractive due to their simplicity of use and low cost. While analog based systems have proven successful, several reasons make digital control attractive.

Another alternative is the use of digital circuits to implement the converter control. They have been showing improvements in performance and reduction of costs. In general, the digital control systems use Digital Signal Processors (DSP) due to their mathematical resources and the auxiliary subsystems (timer module, PWM generator, etc) [2]. One disadvantage of DSP is its sequential operation, in which the instructions are executed sequentially, although the subsystems add up the simultaneous processing capability.

Digital control allows for the implementation of more functional control schemes. Digital circuits are potentially less susceptible to noise and parameter variations.

In order to meet the demand on simultaneous processing, the digital control systems are implemented using Field Programmable Gate Array (FPGA) or Application-Specific Integrated Circuit (ASIC) [2]. They are capable of performing simultaneous processing and can operate at high frequencies. These features allow the implementation of control algorithms with simultaneous procedures. The control algorithm must be properly developed to produce an object suitable to be programmed on a FPGA. This requires that both the converter and the control system be developed on the same environment, which unfortunately is not a favorable aspect on the use of FPGA.

This article presents a methodology to overcome this unfavorable aspect on the use FPGA. The methodology describes an integrated approach to build, test and validate the complete system (converter and control system), thus producing an object suitable to be programmed on a FPGA.

The control systems developed for FPGA avoid the arithmetic operations, very common on DSP, in favor of combinational, sequential and conditional operations that are more suitable for FPGA [2,3].

## II. GENERAL SYSTEM

The proposed methodology will be developed using the Buck converter, with power factor correction [1], as shown in Figure 1. The goal of the converter control is the output voltage regulation in accordance to the input voltage. The converter signals to be sampled are: the input current, the rectified voltage and the output voltage. An analog or a digital system can be controlled using these signals.

The objectives of the methodology are the development, test and validation of a digital control system, as well as the generation of a programming code in hardware description language suitable to be implemented on a FPGA. This phase should be implemented on a single environment capable of integrating the whole structure – the converter and the control system.

Many circuit description models are available and they can be used in softwares, such as ORCAD®, Matlab®, among others. One option, which have been widely chosen [4,5,6], is the use of the Very High Scale Integration Circuit Hardware Description Language (VHDL). This language allows the digital controll integration into an FPGA, and increases the solution portability, once the final code can be easily transported to any manufacturer's FPGAs. Using the VHDL language, in particular the VHDL-AMS (Analog and Mixed-Signals), analog and digital circuits can be described by their behavior.

This aspect facilitates the integration of components of a variety of features (diodes, A/D converters, digital circuits, etc) on the same environment. The VHDL-AMS meets the need of description and integration of analog and digital circuits in the same environment, and it was standardized in 1999 [7,8].

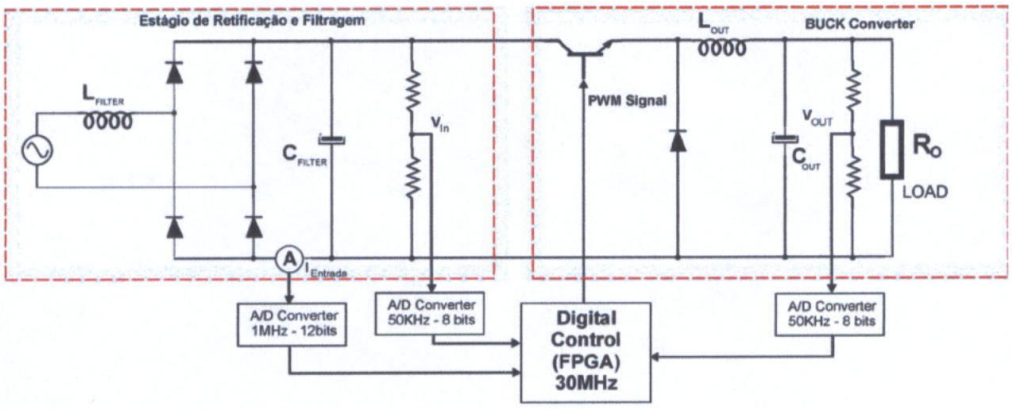



Fig. 1. Buck Converter using digital control implemented on a FPGA.

## III. METHODOLOGY

The methodology consists in the construction of the whole power system – converter and the digital control system – from its description, simulation and validation in a single environment. The system is composed of distinct elements, as shown in Figure 1, that will be described separately, by their behavior and by the description language. After completing all the elements, the system will be described by a structural description [7,8].

### *3.1. Modeling of the components*

The VHDL-AMS is used to model each element, since it allows integration and portability to the code. The elements that compose the conversion system will be evaluated in analog (capacitor, inductor, resistor, etc), digital (logic circuits, digital pulse generators, etc) or mixed mode (A/D Converters, switches, etc). In order to describe a component in a hardware description language, it is necessary to define its inputs, outputs and its behavior.

The detailing or precision level of a component can be easily adapted to the system's development requirements, using most detailed component's description. As an example, one switch can be modeled like a simple conditional test, discarding all internal resistance and capacitance, making easier the simulation process, which is responsible for the system's digital control development [5,6]. If a high precision on the analog components behavior becomes necessary, you can add to the original code the desired changes [9,10]. At this point, some of the most common components used on converters modeling are going to be presented as a sample of the language description facility. These models uses simplified description of the components behavior.

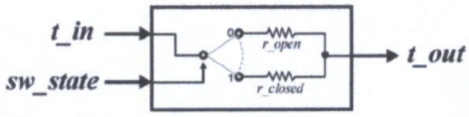
*a) Modeling mixed mode components (analog-digital)* - The mixed mode components offer the capability of integrating the analog and the digital parts of the system in the same structure. The switch of the Buck converter presents this characteristic, since it has analog inputs and outputs, and it is digitally activated. The switch is described as a resistance whose value is altered according to the logic value applied to its digital input, as presented in Figure 2. The model of a component described in the VHDL hardware description language (see Figure 2c) is basically composed of three parts:

- The definition of the libraries used in its description, lines 1 to 3 *(library)*;
- The identification and characterization of its inputs, outputs and atributes, lines 4 to 10 *(entity)*;
- The behavioral or structural description, lines 11 to 26 *(architecture)*.

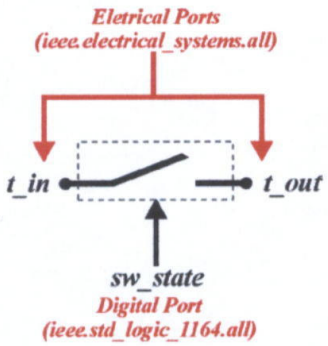
From this description it is possible to observe that terminals *t_in* and *t_out* are defined as electrical values (line 9), and the switch operation is defined as a digital value (line 8). This property allows the feature of external control to the component, which can be obtained from a PWM pulse generator or from a digital control loop, which can be described in VHDL. Another component that has this feature is the analog/digital converter that can also be modeled by the hardware description language [11].

*b) Modeling of analog components* - In order to implement the converter, it is necessary to describe, in VHDL-AMS, the other elements, such as power supplies, resistors, inductors, etc. Table I shows the behavioral description of few elements used in converters. The references to libraries and the characterization of elements were omitted from the description.

*c) Digital components modeling* - The digital control system is a complex structure composed of logic and arithmetic operators. Its implementation becomes easy from the modularization of logic blocks, such as logic operators, adders, shifters, comparators, etc. A PWM pulse generator is classified as a digital component, since its output presents a signal of logic value. Figure 3 presents the modeling of this component. Lines 5 to 8 present the behavioral description of the pulse width for the high level (*t_up*) and the low level (*t_down*). These values can be inserted by the user through the structural description, which will be analyzed on the next section.




(a) Switch signals.




(b) Characterization of Input and Output types.

```
1.  library ieee;
2.  use ieee.electrical_systems.all;
3.  use ieee.std_logic_1164.all;

4.  entity switch_digital is
5.  generic (r_open     : resistance := 1.0e6;
6.           r_closed   : resistance := 1.0;
7.           transient_time    : real     := 1.0e-6);
8.  port   (sw_state         : in std_logic;
9.          terminal t_in, t_out   :   electrical);
10. end entity switch_digital;

11. architecture behav of switch_digital is
12. signal  r_signal : resistance := resistance_open;
13. quantity r       : resistance;
14. quantity v across i through t_in to t_out;
15. begin
16.   switch_state: process (sw_state)
17.   begin
18.     if (sw_state'event and sw_state = '0') then
19.       r_signal <= r_open;
20.     elsif (sw_state'event and sw_state = '1') then
21.       r_signal <= r_closed;
22.     end if;
23.   end process switch_state;
24.   r == r_signal'ramp(transient_time, transient_time);
25.   v == r * i;
26. end architecture behav;
```

(c) VHDL-AMS description.

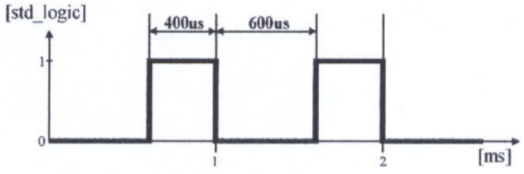
Fig. 2. Modeling of the switch.

**TABLE I**
**VHDL-AMS behavioral description of analog elements**

| Component | VHDL Behavior | Comments |
|---|---|---|
|   | *architecture behav of resistor is*<br>*quantity v across i through pos to neg;*<br>*begin*<br>***v == res * i;***<br>*end behav;* | *The behavior of the resistor is given by its fundamental equation:* ***v = res * i****, where res is the resistance value in ohms, to be specified by the user.* |
|   | *architecture behav of resistor is*<br>*quantity v across i through pos to neg;*<br>*begin*<br>***i == cap * v'dot;***<br>*end behav;* | *The capacitor fundamental equation is:* ***i = cap * dv/dt****, where cap is capacitance value given in Faraday, to be specified. The directive 'dot describes the behavior of the derivative.* |
|   | *architecture behav of resistor is*<br>*quantity v across i through pos to neg;*<br>*begin*<br>***v == ind * i'dot;***<br>*end behav;* | *The inductor fundamental equation is :* ***v = ind * di/dt****, where ind is the inductance in Henry, to be specified. The directive 'dot describes the behavior of the derivative.* |
|   | *architecture behav of v_sine is*<br>*quantity v across i through pos to neg;*<br>*begin*<br>***v == amplitude * sin(math_2_pi * freq * NOW);***<br>*end behav;* | *The simusoidal voltage is described by its fundamental equation:* ***v(t) = amplitude * sin(2 * π * freq * t)****. The amplitude and the frequency (freq) are specified by the user. The directive NOW emulates the behavior as function of the time.* |

```
1.  architecture behav of pwm_generator is
2.    begin
3.      CreateClock: process
4.        begin
5.          clk_out <= '0';
6.          wait for t_down ;
7.          clk_out <= '1';
8.          wait for t_up ;
9.      end process CreateClock;
10  end behav;
```

(a) VHDL description.




(b) PWM Generator waveform.

Fig. 3. Modeling of the PWM generator.

*3.2. Structural description*

The structural description is composed of instantiations of components which have already been validated. Figure 4a presents a circuit that used the formerly described components. Figure 4b shows the result of the simulation of a circuit. The signal TEST.CO.V represents the sinusoidal output signal. The digitally described signal TEST.CLK_PWM is the switch control signal. The signal TEST.C3.V is the voltage between *node2* and *ground*. From the simulation result presented in Figure 4b it is possible to validate the structural description. The structural description is presented in Figure 5.

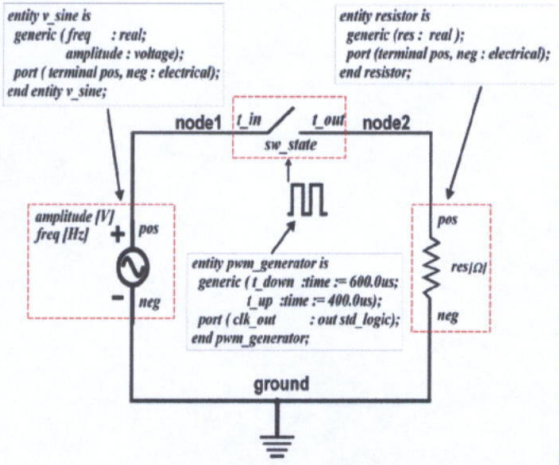
All components utilized in the structure must have already been inserted in the environment, as exemplified in line 4 of this procedure. The instantiation of the used components is performed through the processes (lines 11, 14, 17 and 20). Signals (lines 8 and 9) are used in order to integrate the components on the same environment. The signals allow the connectivity of the blocks on the structural description and are similar to the nodes of an electric circuit.

The attributes of the signals used in the description are set through the VHDL directive *generic map*. The connectivity is performed by the *port map* using temporary signals [7,8].

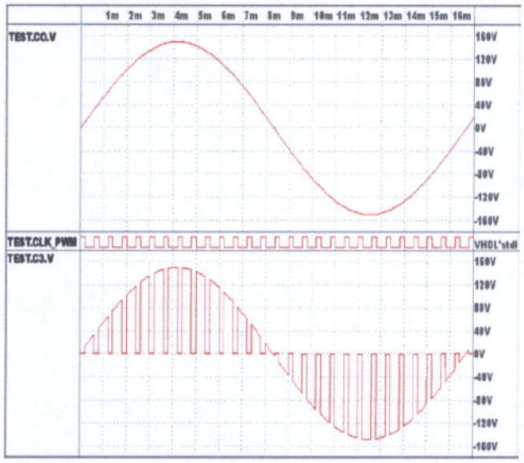
## IV. CONVERTER SIMULATION

Figure 6 present the circuit of a Buck converter. Figure 7 presents the SMASH® simulation results, in order to demonstrate the feasibility of the methodology. The signals TEST.C0.V, TEST.C0.I and TEST.C10.V represent, respectively, the inductor current, the capacitor current and the voltage at the output of the Buck converter.

From the results shown in Figure 7, it can be verified that the converter structural description is valid and the circuit has a settling time of approximately 16 ms according to the parameters described in Figure 6.




(a) Circuit example.




(b) Simulation results.

Fig. 4. A structural description example.

```
1.  library IEEE;
2.  use IEEE.electrical_systems.all;
3.  use IEEE.std_logic_1164.all;
4.  use work.all;

5.  entity Test is
6.  end Test;

7.  architecture Top of Test is
8.  signal clk_pwm : std_logic;
9.  terminal node1, node2: electrical;
10. begin
11. C0: entity v_sine (behav)
12.       generic map (amplitude => 150.0, freq => 60.0)
13.      port map(pos => node1, neg => ground);
14. C1: entity pwm_generator(behav)
15.       generic map (t_down => 600.0us, t_up => 400.0us)
16.       port map(clk_out => clk_pwm);
17. C2: entity switch_digital(behav)
18.       generic map (r_open => 0.001, r_closed => 1.0e6, transient_time => 1.0e-6)
19.       port map(sw_state => clk_out, t1=> node1, t2=> node2);
20. C3: entity resistor(behav)
21.      generic map (res => 100.0)
22.      port map (t1 => node2, t2 => ground);
23. end Top;
```

Fig. 5. Structural description of circuit in Fig. 4.

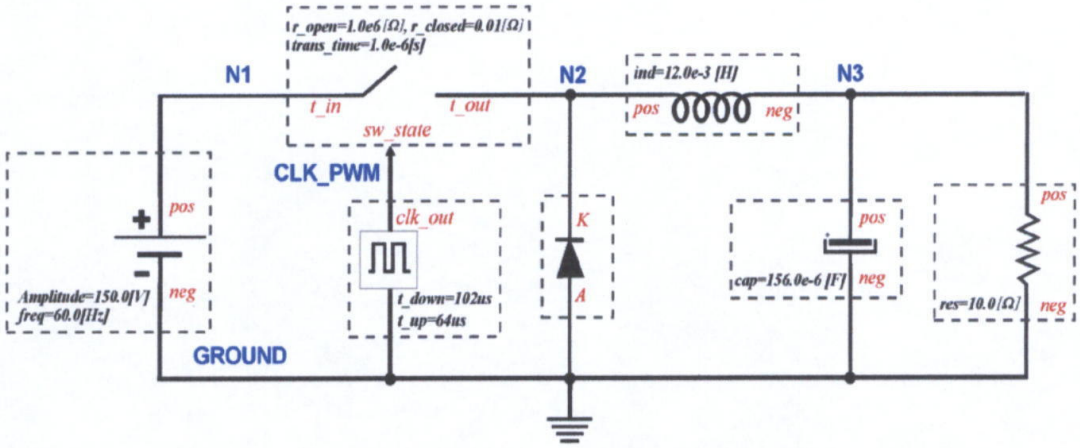



Fig. 6. DC/DC converter.

## IV. DIGITAL CONTROL

To the digital control development, some digital components creation, which makes the implementation possible, becomes needed. Making use of combinational circuits, it is possible to have two or more processes running at the same time, what allows easy implementation of protection to max input currents, max output voltage, max duty cycle, etc [2]. The most commonly used components on controlling are adders, accumulators, multipliers, and comparators, because they can emulate the classical controllers' behavior, which make use of integration and derivation techniques to implement the controlling strategy. The comparators use, with A/D converters, makes possible the implementation of circuit's needed protection. Table II shows the behavioral description of few elements used in digital control.

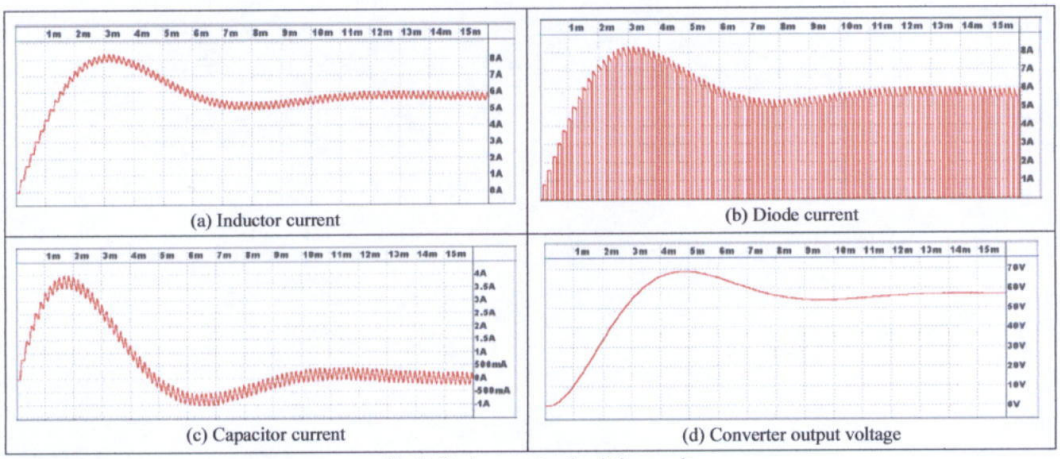
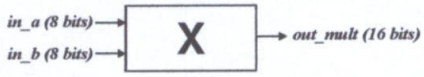
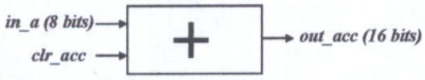
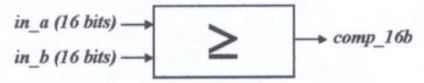



Fig. 7. Buck converter simulation results.

**TABLE II**
**VHDL-AMS behavioral description of digital elements**

| Component | VHDL Behavior |
|---|---|
|   | *architecture behav of mult is*<br>*begin*<br>*process*<br>*variable temp,tempa,tempb : std_logic_vector(15 downto 0);*<br>*begin*<br>*tempa := "00000000" & in_a;*<br>*tempb := "00000000" & in_b;*<br>*temp := unsigned(tempa) * unsigned(tempb);*<br>*out_mult <= temp(15 downto 0);*<br>*end process;*<br>*end behav;* |
|   | *architecture behav of sumacc is*<br>*begin*<br>*process*<br>*variable tempa, tempb : std_logic_vector (15 downto 0);*<br>*begin*<br>*wait until clk'event and clk = '1';*<br>*if clr_acc='1' then*<br>*out_acc <= (others => '0');*<br>*tempa := (others => '0');*<br>*else*<br>*tempb := "00000000" & in_a;*<br>*tempa := (unsigned(tempa)) + (unsigned(tempb));*<br>*out_acc <= tempa (15 downto 0);*<br>*end if;*<br>*end process ;*<br>*end behave;* |
|   | *architecture behav of comp_16b is*<br>*begin*<br>*compare : process (cmp)*<br>*variable in1_hold : std_logic_vector (15 downto 0);*<br>*variable in2_hold : std_logic_vector (15 downto 0);*<br>*begin*<br>*in1_hold := in_a;*<br>*in2_hold := in_b;*<br>*if cmp'event and cmp = '1' then*<br>*if signed(in1_hold) >= signed(in2_hold) then*<br>*eq <= '1';*<br>*else*<br>*eq <= '0';*<br>*end if;*<br>*end if;*<br>*end process;*<br>*end architecture behav;* |

## V. CONCLUSION

This article presented a methodology for the design of a power converter system. The fundamentals necessary for the description of circuits using VHDL-AMS were presented. The methodology allows description, test and validation the digital control code, in VHDL-AMS, along with the other converter components. By using this methodology it is possible to develop the whole structure of the power conversion, considering the responses of the iteration between the control system and the converter. This reduces time and costs at the experimental phase. The VHDL code obtained at the end of the process can be transferred to any FPGA or ASIC programming environment. The Physical implementation, as well as the choose of an economically viable controlling system, comparing to the analog controllers, are future works proposals.

## REFERENCES


[1] Barbi, F. Pöttker de Souza, "A Unity Power Factor Buck Pre-Regulator with Feedforward of the Output Inductor Current", *IEEE Applied Power Electronics Conference (APEC)*, 1999.

[2] A. de Castro, P. Zumel, O. García, T. Riesgo and J. Uceda, "Concurrent and Simple Digital Controller of an AC/DC Converter with Power Factor Correction", *IEEE Trans. Ind. Electron.*, vol. 46, pp. 3-12, Fevereiro de 1999.

[3] Wu, A.M.; Jinwen Xiao; Markovic, D.; Sanders, S.R., "Digital PWM control: application in voltage regulation modules", *Power Electronics Specialists Conference*, vol. 1, pp. 77 – 83, 1999.

[4] Lukasz Starzak, Andrzej Napieralski, Jean-Jacques Charlot, "VHDL-AMS: A Competitor for SPICE", *Modeling of Semiconductor Devices*, pp. 353-356, TCSET 2002, Frebruary 2002.

[5] Teresa Riesgo, Yago Torroja, Eduardo de la Torre, "Design Methodologies Based on Hardware Description Languages", *IEEE Transactions on Industrial Electronics*, pp. 3-12, Vol. 46, No. 1, February 1999.

[6] Ahmed Fakhfakh, H. Levi, N. Milet-Lewis, Y. Danto. "Behavioral modeling of analogue and mixed integrated systems with VHDL-AMS for RF applications", *XV Brazilian Symposium on Integrated Circuits and Systems Design*.

[7] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993.

[8] IEEE Standard VHDL Analog and Mixed-Signal Extensions, IEEE Std 1076.1-1999.

[9] Alex Doboli, Ranga Vemuri, "Behavioral Modeling for High-Level Synthesis of Analog and Mixed-Signal Systems From VHDL-AMS", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1504-1520,Vol. 22, No. 11, November 2003.

[10] Ernst Christen, Kenneth Bakalar, "VHDL-AMS – A Hardware Description Language for Analog and Mixed-Signal Applications", *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, pp. 1263-1272, Vol. 46, No. 10, October 1999.

[11] Peter J. Ashenden, Gregory D. Peterson, Darrell A. Teegarden, *The System Designer's Guide to VHDL-AMS*, Morgan Kaufmann, 2002.


## BIOGRAPHIES

**Michel Santana** was born in São Paulo, Brazil, in 1980. He received the B.S. degree in Computer Engineering from the Itajubá Federal University, Brazil, where he currently is working toward the M.S. degree in electrical engineering. His research interests include design of digital CMOS integrated circuits and VHDL-AMS.

**Enio R. Ribeiro** received the B.S. and M.Sc.A. degrees in Electrical Engineering from the Itajubá Federal University (Brazil) and École Polytechnique de Montréal (Canada), in 1990 and 1993 and the Ph.D. degree from the Federal University of Santa Catarina (Brazil) in 2003.
In 1993, he joined the Department of Electronics at Itajubá Federal University where he is presently Professor. His research interests are digital control of static converters, active filtering and frequency converters.

**Robson Luiz Moreno** was born in Varginha, Brazil, in 1963. He received the B.S. and M.S. degrees in Electrical Engineering from the Itajubá Federal University, Brazil, and University of Campinas, Brazil, in 1988 and 1996 and the Ph.D. degree from the Polytechnic School – University of São Paulo, Brazil, in 2002.
In 2002, he joined the Department of Electronics at Itajubá Federal University where he is presently Professor. His research interests include VHDL-AMS, design of analog and digital CMOS integrated circuits.