Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

DANILO BARRETO CAVALCANTI

SIMULADOR TEMPO REAL BASEADO EM FPGA DE CONVERSORES CC-CC MODULADO

Campina Grande, Paraíba
Maio de 2016

DANILO BARRETO CAVALCANTI

SIMULADOR TEMPO REAL BASEADO EM FPGA DE CONVERSORES CC-CC MODULADO

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Energia

Orientador:
Professor Alexandre Cunha Oliveira, D. Sc.

Campina Grande, Paraíba
Maio de 2016

DANILO BARRETO CAVALCANTI

SIMULADOR TEMPO REAL BASEADO EM FPGA DE CONVERSORES CC-CC MODULADO

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Energia

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Alexandre Cunha Oliveira, D. Sc.
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho a todos os que tiveram empatia em minha jornada na graduação e a todos aqueles que ainda estão trilhando seus próprios caminhos.

AGRADECIMENTOS

Agradeço à minha família, pelo exemplo dado nas diversas áreas do conhecimento humano e por sempre fornecerem um espaço aberto a discussões sobre temas novos e antigos.

Agradeço aos meus amigos de diversas origens que fizeram parte da minha rotina de uma forma ou de outra, dando novas cores às experiências de vida durante a graduação.

Agradeço aos mestres professores que com certeza lembrarei pelo resto da vida pela paixão e dedicação por ensinar, com fome por conhecimento e desejo de compartilhá-lo de várias maneiras até serem compreendidos, independentemente da existência de exames avaliativos.

Agradeço especialmente ao meu orientador, que me norteou em muitas disciplinas durante a graduação sempre com muito bom humor, ética e compreensão, inclusive durante a concepção deste projeto.

Agradeço à Instituição pelo fornecimento de condições para a realização deste trabalho e pela boa vontade dos funcionários de diversos setores que me ajudaram a consolidá-lo.

Agradeço especialmente a Sarah Albuquerque por ter acompanhado o processo do início ao fim de forma mais próxima do que qualquer outra pessoa, fornecendo apoio de várias maneiras.

“Tinha que ser Bikevow...”

RESUMO

Fontes de corrente contínua fornecem energia para uma diversa gama de circuitos eletrônicos utilizados no dia a dia. Conversores CC-CC chaveados compõem uma classe interessante de fontes de corrente contínua devido à controlabilidade de suas grandezas e da possibilidade de miniaturização de seus componentes. Este trabalho fornece modificações a um simulador tempo real em FPGA de forma a tornar-se programável em módulos por descrição de *hardware* em linguagem Verilog. Essas modificações possibilitam diversas aplicações no estudo de conversores CC-CC em Eletrônica de Potência para estudos em malha aberta e dos sinais de controle gerados a partir de estímulos, por exemplo.

Palavras-chave: Conversores CC-CC, simulador tempo real, Verilog, FPGA

ABSTRACT

DC current power supplies feed a vast array of electronic circuits used in many daily applications. Nowadays, DC-DC converters correspond to an interesting class of DC power supplies given controllability of its characteristics and the small achievable size of their components. This work provides modifications to a FPGA real time simulator in order to make it programmable in Verilog hardware description language modules. These modifications allow diverse applications concerning the study of DC-DC converters in Power Electronics for open loop analyses and control signal generation from stimulae, for example.

Keywords: DC-DC converters, real time simulator, Verilog, FPGA

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

CA – Corrente Alternada

CC – Corrente Contínua

D/A – Digital-analógico

FPGA – *Field Programmable Gate Array*

PLL – *Phase Locked Loop*

LISTA DE SÍMBOLOS

I_L – Corrente do indutor

V_C – Tensão sobre capacitor de saída

V_{DA} – Tensão de saída do conversor digital-analógico

SUMÁRIO

1 INTRODUÇÃO

Processamento de Energia tem sido uma área de grande importância para o desenvolvimento da sociedade uma vez que se busca a otimização do consumo de energia, de diversos tipos, em seus vários pontos e formas de conversão. A energia elétrica passa por uma ampliação na diversidade com que é gerada, distribuída e convertida.

O aumento da gama de eletrônicos no mundo também aumentou a necessidade de produzir fontes de tensão mais sofisticadas para seus aparelhos, uma vez que cada um possui suas características de funcionamento e pontos ótimos de operação. O uso de corrente contínua para alimentar essa diversidade de eletrônicos passou pelo uso de simples pontes retificadoras a circuitos complexos com correção automática de características de circuito; coexistindo hoje ambos os tipos a depender da aplicação.

Este trabalho aborda a continuação do desenvolvimento do sistema simulador tempo real em FPGA para simuladores CC-CC desenvolvido na Universidade Federal de Campina Grande durante atividade de estágio, revisando as características do projeto consolidado e adicionando características de independência entre seus componentes por meio da divisão do sistema em módulos a fim de que seja expansível e individualmente configurável.

O objetivo principal deste trabalho, portanto, é obter um sistema simulador tempo real de conversores CC-CC modulado a partir do sistema anterior. Como objetivos secundários, listam-se:

- Inserção de comentários nos códigos-fonte componentes;
- Revisão bibliográfica para justificar o uso da aritmética computacional adotada;
- Revisão das simulações em *software* das equações de diferenças dos modelos dos conversores;
- Revisão dos resultados da implementação em FPGA;
- Obtenção de uma equação para relacionar o nível de saída do conversor D/A do sistema ao valor inteiro armazenado;

- Criar módulos e modelos de descrição de *hardware* em Verilog para utilização e modificação;

Para alcançar estes objetivos, foi realizada, inicialmente, uma revisão bibliográfica de alguns simuladores tempo real e uma revisão com o material disponível do simulador existente. Em seguida, foram inseridos comentários pertinentes a trechos de código em C e em Verilog para indicar suas funções. Um conjunto de testes foi efetuado de forma a confirmar os resultados e a ser reproduzível. Após a confirmação dos modelos matemáticos e do correto funcionamento das partes componentes da descrição de *hardware*, o sistema escrito em Verilog foi subdividido em módulos distintos correspondentes às partes funcionais dele. Finalmente, todos os módulos referentes aos conversores foram modificados para serem moldes parametrizáveis em tempo de compilação.

A Seção 2 trata da fundamentação teórica envolvida no trabalho, tratando dos modelos matemáticos utilizados, da teoria que embasa os conversores CC-CC e de simuladores tempo real. A Seção 3 explicita a sequência de passos tomada para realizar as modificações no sistema a fim de cumprir os objetivos e promover a verificação de cada passo. A Seção 4 apresenta os resultados das simulações feitas em *software* e *hardware*, além de comentários sobre resultados esperados e erros encontrados. A Seção 5 resume os resultados quanto ao cumprimento dos objetivos e das aplicações prováveis do sistema final.

2 FUNDAMENTAÇÃO TEÓRICA

A elaboração do simulador depende do entendimento sobre sistemas simuladores tempo real, da compreensão dos circuitos-modelo, do entendimento do tratamento matemático envolvido desde o sistema contínuo à sua representação discreta, da compreensão da linguagem de descrição de *hardware* adotada (Verilog) e do cumprimento das condições de tempo real do sistema.

Ressalta-se que o simulador já é suposto tempo real em razão do trabalho utilizado como base (Mendes, 2015). Logo, o foco do tratamento deste está voltado para os outros aspectos do simulador mencionados.

2.1 SIMULAÇÃO TEMPO REAL

A simulação de sistemas por computadores de propósito geral é uma ferramenta útil para auxiliar a execução de projetos de sistemas de várias áreas. A simulação tempo real se insere nesse campo como um aprimoramento nas restrições de tempo a serem consideradas. A simulação de sistemas por *software* pode requerer passos de cálculo de magnitude variável de ordem de microssegundos a alguns minutos, a depender da complexidade de modelagem (Hüller, 2014).

Testes de controladores e demais componentes que interagem com o sistema precisam ser feitos considerando a resolução temporal de atuação deles. Em vez de modelar tais subsistemas e inseri-los à simulação do sistema principal, emprega-se a técnica de simulação *hardware in the loop*, onde o sistema é representado por uma unidade de computação programada para simulá-lo, enviando sinais adequados aos demais componentes. Essas unidades podem ser desde microprocessadores e microcontroladores a computadores de propósito geral ou computadores dedicados.

Essa técnica esbarra em limitações de tempo provenientes da complexidade com que os modelos podem ser concebidos. Consequentemente, fez-se necessária a adição de restrições temporais aos sistemas computacionais simuladores, culminando na utilização de simulação tempo real.

Simulação tempo real implica a existência de uma janela estrita de tempo a ser respeitada durante duas requisições distintas ao sistema. A maneira como esses requisitos são atendidos depende do modelo tempo real adotado, mas, de forma geral, é pouco ou nunca aceitável a ausência de um dado do sistema após um passo temporal estabelecido. Um sistema tempo real que substitui um sistema propriamente dito deve obedecer aos tempos de resposta do sistema original de acordo com uma modelagem feita.

A utilização de sistemas simuladores tempo real é interessante devido à impossibilidade ou dificuldade de operação ou manuseio do sistema físico correspondente. Ademais, um simulador que possui essa característica pode, então, ser inserido a um conjunto *hardware in the loop* e promover operações de entrada e saída com a resolução temporal adequada.

Embora haja sistemas comerciais de simulação tempo real, a concepção de um deles depende tão somente do respeito a limitações temporais impostas. Ou seja, é

possível obter um sistema classificado tempo real pelo emprego de microcontroladores, *FPGA* e computadores de propósito geral, por exemplo, garantindo o atendimento das condições tempo real do projeto.

O simulador modificado neste trabalho foi criado a partir da explicitação de suas equações físicas, considerando certo grau de complexidade ao modelo, seguida de uma transformação do sistema de tempo contínuo a discreto, pela transformação de Tustin. Avaliou-se a utilização de uma aritmética computacional adequada a fim de garantir um passo de cálculo próprio para respeito das condições tempo real. Finalmente, o número de bits utilizado no sistema precisava garantir a convergência do sistema com o modelo discreto obtido aplicado à aritmética considerada, dado um valor de precisão aceitável.

2.2 TRANSFORMAÇÃO DE TUSTIN

A representação de sinais e sistemas analógicos para o processamento em microcomputadores requer a transformação deles para grandezas digitais. Essa transformação ocorre primeiramente pelo passo de amostragem, seguida de um passo de quantização.

A transformação de Tustin, ou transformação bilinear, transforma um sistema de tempo contínuo em um de tempo discreto segundo a aproximação em que

$$z = e^{sT} = \frac{e^{\frac{sT}{2}}}{e^{-\frac{sT}{2}}} \approx \frac{1 + \frac{sT}{2}}{1 - \frac{sT}{2}}$$

Desta forma, um sistema de tempo contínuo é aproximado para um de tempo discreto tomando

$$s = \frac{2}{T} \cdot \frac{z - 1}{z + 1}$$

Onde T é o passo de cálculo considerado na integração numérica utilizando regra do trapézio na derivação da transformada bilinear (ou, aplicável para outros casos, a taxa de amostragem do sistema). (Oppenheim, 2010)

2.3 CONVERSORES CC-CC

Os conversores CC-CC são amplamente empregados em fontes de alimentação chaveadas e no acionamento de motores elétricos. Eles possuem três topologias básicas: Buck, Boost e Buck-Boost. A primeira é abaixadora de tensão, a segunda elevadora, e a terceira possui uma combinação de ambas as características podendo ser abaixadora ou elevadora. Além destas, há topologias derivadas, como os conversores isolados Flyback, Forward e ainda conversores ressonantes, por exemplo. (Robert W. Erickson, 2001) (Siskind, 1963)

Os conversores CC-CC possuem dois modos de funcionamento, condução contínua ou condução descontínua. O modo de condução é caracterizado pela corrente no indutor. Em regime permanente, se a corrente não atinge um valor nulo, então o conversor está operando no modo de condução contínua (CCM – *continuous conduction mode*). Se a corrente atinge esse valor a cada etapa de comutação, então se está operando no modo de condução descontínua (DCM - *discontinuous conduction mode*). (Petry, 2014)

Os modelos de conversores CC-CC utilizados apresentam-se em modo de condução contínua. Este trabalho aborda a criação de módulos para conversores Buck, Boost e Buck-Boost, criando um modelo para a inserção dos demais tipos.

2.3.1 CONVERSOR BUCK

Esse conversor é o mais simples conversor abaixador chaveado. Sua representação, na Figura 1, possui um circuito retificador resumido a uma fonte CC de entrada, V_i, e um modelo ideal de chave controlada, S_1.

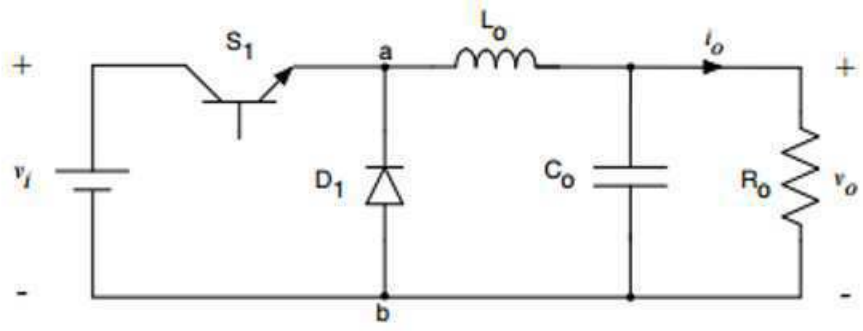

FIGURA 1 DIAGRAMA DO CONVERSOR BUCK

Ele possui dois modos de funcionamento de acordo com o acionamento da chave. Como pode ser observado na Figura 2, há um instante em que a malha da carga inclui a fonte e outro instante possuindo apenas elementos passivos. O ciclo de trabalho da chave é fator determinante da tensão de saída.

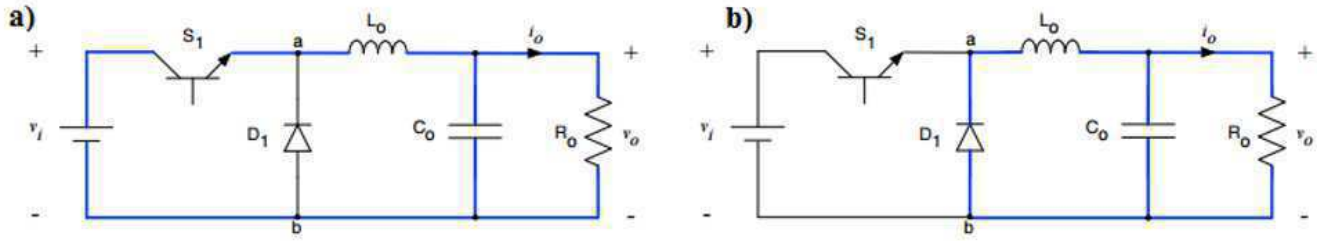

FIGURA 2 MODOS DE FUNCIONAMENTO DO CONVERSOR BUCK; A) CHAVE LIGADA B) CHAVE DESLIGADA

A aplicação da transformação bilinear produz as equações de diferenças necessárias para a aproximação do sistema em grandezas digitais. Elas estão apresentadas conforme o desenvolvimento de Mendes (2015).

Chave ligada:

$$i(k) = \frac{T}{L}V_i - \frac{T}{2L}[V_c(k-1) + V_c(k-2)] + i(k-1)$$

$$V_c(k) = \frac{T}{2C}[i(k) + i(k-1)] - \frac{T}{2CR}[V_c(k-1) + V_c(k-2)] + V_c(k-1)$$

Chave desligada:

$$i(k) = \frac{T}{2L}[V_c(k-1) + V_c(k-2)] + i(k-1)$$

$$V_c(k) = \frac{T}{2C}[i(k) + i(k-1)] - \frac{T}{2CR}[V_c(k-1) + V_c(k-2)] + V_c(k-1)$$

2.3.2 CONVERSOR BOOST

O conversor Boost representa a mais simples topologia elevadora de tensão para o caso de conversão CC-CC. Ele encontra-se representado na Figura 3.

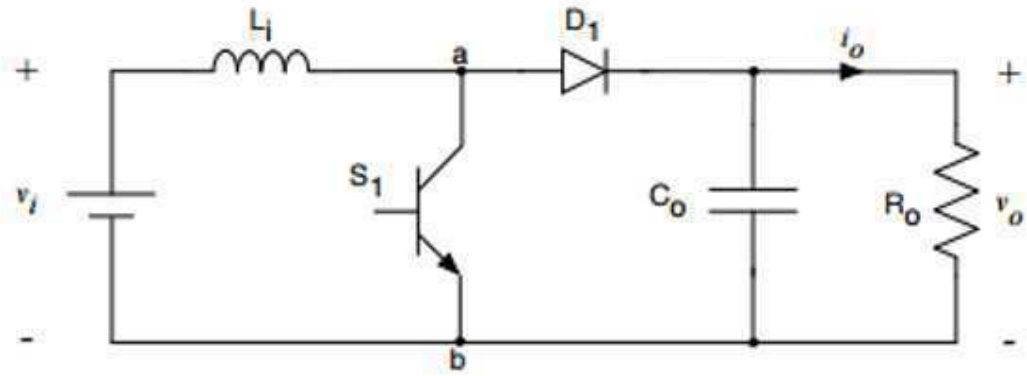

FIGURA 3 DIAGRAMA DO CONVERSOR BOOST

A partir do controle do acionamento da chave S_1 e supondo que V_i representa um sistema retificador de entrada, é possível estabelecer dois modos de operação para o sistema, como representado na Figura 4.

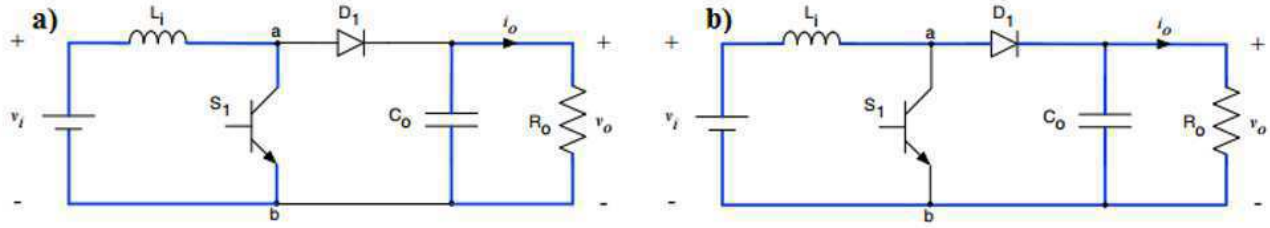

FIGURA 4 MODOS DE FUNCIONAMENTO DO CONVERSOR BOOST; A) CHAVE LIGADA B) CHAVE DESLIGADA

Analogamente, são obtidas suas equações aproximadas para tempo discreto.

Chave ligada:

$$i(k) = \frac{T}{L} V_i + i(k-1)$$

$$V_c(k) = \frac{T}{2CR}[V_c(k-1) + V_c(k-2)] + V_c(k-1)$$

Chave desligada:

$$i(k) = \frac{T}{L} V_i - \frac{T}{2L}[V_c(k-1) + V_c(k-2)] + i(k-1)$$

$$V_c(k) = \frac{T}{2C}[i(k) + i(k-1)] - \frac{T}{2CR}[V_c(k-1) + V_c(k-2)] + V_c(k-1)$$

2.3.3 CONVERSOR BUCK-BOOST

Esse tipo de conversor pode ser um abaixador ou elevador de tensão a depender do seu ciclo de trabalho. Ele representa a mais básica topologia para esse tipo de operação e seu circuito corresponde à Figura 5.

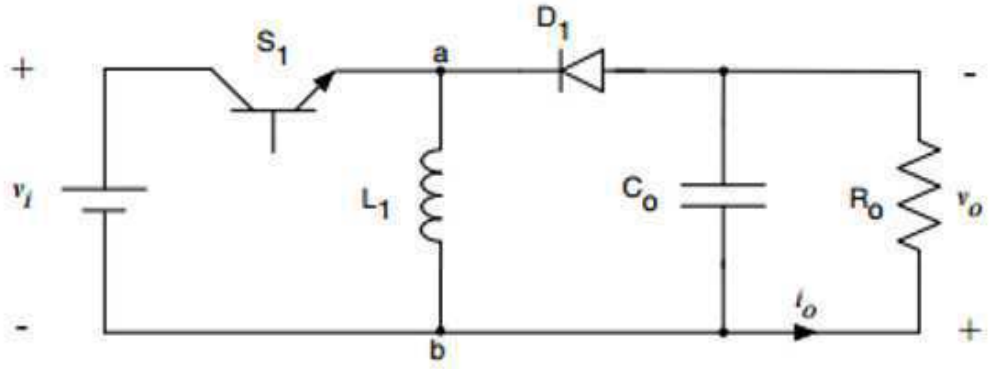

FIGURA 5 DIAGRAMA DO CONVERSOR BUCK-BOOST

Seguindo as analogias feitas até o momento, o acionamento de sua chave produz dois modos de operação, segundo ilustrado na Figura 6, tendo suas equações características como segue.

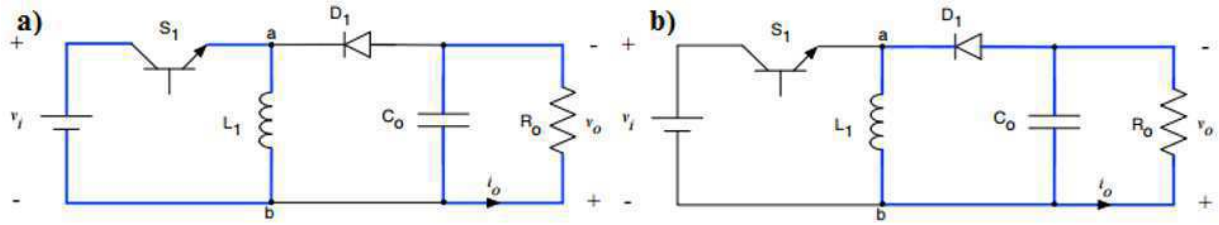

FIGURA 6 4 MODOS DE FUNCIONAMENTO DO CONVERSOR BUCK-BOOST; A) CHAVE LIGADA B) CHAVE DESLIGADA

Chave ligada:

$$i(k) = \frac{T}{L} V_i + i(k-1)$$

$$V_c(k) = -\frac{T}{2CR}[V_c(k-1) + V_c(k-2)] + V_c(k-1)$$

Chave desligada:

$$i(k) = \frac{T}{2L}[V_c(k-1) + V_c(k-2)] + i(k-1)$$

$$V_c(k) = \frac{T}{2C}[i(k) + i(k-1)] - \frac{T}{2CR}[V_c(k-1) + V_c(k-2)] + V_c(k-1)$$

2.4 MODELOS DE ARITMÉTICA COMPUTACIONAL

Sinais digitais precisam ser operados segundo uma sequência de procedimentos. A aritmética computacional possui dois modos básicos: aritmética de ponto fixo e aritmética de ponto flutuante.

A aritmética de ponto fixo consiste na normalização de grandezas com casas decimais em números inteiros. Todas as operações são efetuadas como se o número fosse inteiro, mas há conhecimento do programador da posição da vírgula durante todos os procedimentos. Esse tipo de aritmética necessita de várias preparações em relação à normalização dos valores a serem operados e requer uma análise da faixa dinâmica de possíveis valores a serem representados. Operações de multiplicação e divisão alteram a base de representação e precisam ser normalizadas.

A aritmética de ponto flutuante geralmente segue o padrão IEEE 754. Esse padrão guarda todos os números em notação normalizada contendo mantissa (parte fracionária do número após normalização) e expoente (a ser aplicado no processo de decodificação) onde se supõe que o número sempre inicia com 1, seguido de *bits* de parte fracionária. Há valores especiais para representação de números entre 0 e 1 e de números especiais como $-\infty$ e $+\infty$ e NaN, indicando que o resultado não representa um número (por uma operação não permitida). Essa notação normalizada faz com que seja possível representar uma faixa muito maior de números do que com ponto fixo, para o mesmo número de bits de representação, uma vez que a ordem de grandeza do valor é definida pelo expoente de forma dinâmica, e não inicialmente. Um número de ponto flutuante pode ser decodificado por meio da seguinte expressão:

$$sinal \cdot 2^{ExpoenteReal} \cdot (baseDeNormalizaçãoo + mantissa)$$

Onde o sinal pode ser substituído por +1 ou -1; o termo ExpoenteReal é calculado de acordo com a decodificação do campo de expoente da representação; a base de normalização é um número sempre implícito ao representar valores (em geral, adota-se 0 ou 1 como valores implícitos); a mantissa é a da diferença entre a base de normalização e o número a ser representado.

Em comparação, para o mesmo número de bits de representação, a aritmética de ponto fixo possui precisão fixa que depende da normalização efetuada e do número de bits disponível; a de ponto flutuante possui passo variável entre seus valores de

representação a depender do expoente, porém possui maior liberdade quanto aos valores válidos a serem armazenados ou computados.

Neste trabalho, observou-se que o uso adequado da aritmética de ponto fixo pôde se sobrepor às facilidades de utilizar o padrão IEEE 754 uma vez que é mais simples de ser programada e requer apenas operações de deslocamento de *bits* após operações de multiplicação e divisão para retornar à representação original; em comparação com as normalizações requeridas pelo ponto flutuante. As simulações comprovam que um número idêntico de *bits* provoca resultados de precisão idêntica, para certas faixas, ou até melhor para ponto fixo, supondo que os números nunca saem da faixa predefinida. Essa decisão foi tomada levando em consideração diversos trabalhos semelhantes. (Hüller, 2014; França, 2009; Abourida, Belanger, & Dufour, 2005)

A representação tomada para qualquer número foi

$$G_Q = \frac{G}{G_{base}} \cdot NQ$$

Onde

- G_Q é a grandeza normalizada para representação em Q+1 *bits*;
- G é a grandeza a ser normalizada, representada com quantas casas decimais for possível;
- G_{base} é o valor base da grandeza para o sistema por unidade;
- NQ é o maior número para Q bits (considerando apenas números positivos): $2^Q - 1$.

Toma-se o complemento de 2 do número correspondente positivo para o caso em que G é negativo.

3 METODOLOGIA

Uma série lógica de passos foi tomada a fim de garantir que todas as condições anteriormente estabelecidas para o simulador fossem mantidas, acessíveis para verificação e devidamente documentadas.

3.1 CONDIÇÕES INICIAIS

O simulador original é fruto de um trabalho de estágio realizado na Universidade Federal de Campina Grande (UFCG) no Laboratório de Eletrônica Industrial e Acionamento de Máquinas (LEIAM) (Mendes, 2015).

Nesse trabalho, há códigos-fonte de linguagem C escritos para a criação de arquivos de biblioteca dinâmica (DLL) utilizados em simulação comparativa no *software* PSIM. Os arquivos contém as representações em ponto fixo e em ponto flutuante das equações de tempo discreto, obtidas pela transformada de Tustin das equações a tempo contínuo, dos conversores. A simulação disponível é realizada com 20 bits de representação em ponto fixo e utilizava os valores padrão de tamanho para ponto flutuante (64 bits em precisão dupla). Esse simulador possui um único arquivo englobando todas as simulações comparativas utilizadas no trabalho (conversores Buck, Boost e Buck-Boost em circuitos isolados) para PSIM.

O simulador tempo real está descrito em um único código-fonte escrito em Verilog, compatível com a placa de desenvolvimento em FPGA DE-2 da Altera. Ele contém a especificação de cada conversor para aritmética de ponto fixo com 20 bits dentro de uma estrutura que seleciona o módulo a ser exibido na saída. Essa, por sua vez, é feita por transmissão serial a um conversor D/A de 16 bits de 0 a 5V cujos dados são os 16 bits mais significativos dos valores de tensão e corrente dos conversores, acrescidos de um valor numérico para os números negativos serem representados a partir de 0V.

Os códigos mencionados carecem de documentação e não demonstram um grau considerável de parametrização para modificação rápida das grandezas dos componentes do circuito.

Os códigos-fonte, programas e equipamentos disponíveis são:

- Códigos-fonte em C para compilação de DLL para os três tipos de conversores abordados;
- Placa de desenvolvimento DE-2 da Altera;
- Código-fonte em Verilog HDL contendo descrição dos módulos compatível com DE-2;
- Conversor D/A de 16 bits LTC2600CGN;
- Dev-C++;
- PSIM;
- Quartus II;

A Figura 7 representa, em forma de diagrama de blocos, o sistema descrito em linguagem Verilog inicialmente. Há uma entrada de seleção, controlada por chaves do kit De-2, e outra *areset* que desabilita o relógio principal do sistema.

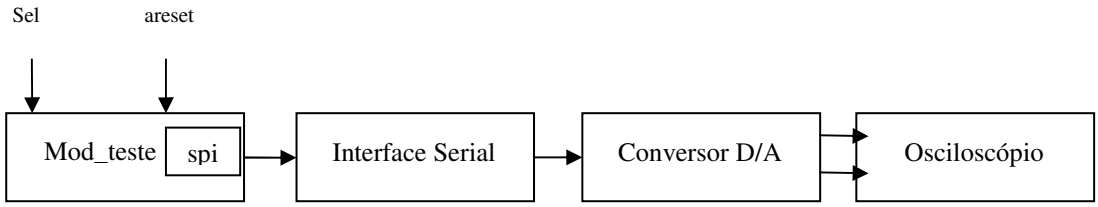

FIGURA 7 - DIAGRAMA DE BLOCOS DO SIMULADOR INICIAL

A montagem final utilizada como simulador, tanto no trabalho anterior quanto no atual, está representada na Figura 8, onde podem ser observados o kit de desenvolvimento para FPGA da Altera DE-2 ligado via cabo *flat* a um circuito de comunicação serial, que por sua vez se liga ao conversor D/A LTC2600CGN. Há uma fonte de alimentação auxiliar para fornecer 15V, 5V e um referencial de terra ao circuito de comunicação serial e ao conversor e um osciloscópio digital para aferição das tensões e correntes codificadas de 0 a 5V.

Salienta-se que as condições de tempo real são consideradas satisfeitas tomando passo de cálculo de $100ns$, a partir de um relógio secundário de 200MHz (uma amostra nova calculada a cada vinte pulsos de relógio) obtido por um módulo PLL que opera com um relógio principal de 50MHz. (Mendes, 2015)

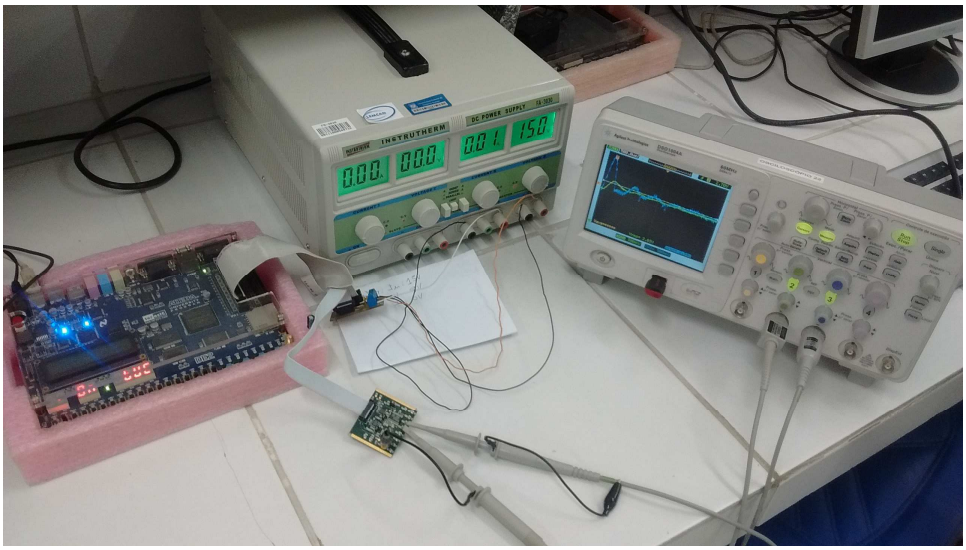

FIGURA 8 - COMPONENTES DO SIMULADOR TEMPO REAL E INSTRUMENTAÇÃO DE EXPERIMENTAÇÃO

3.2 PARAMETRIZAÇÃO DAS SIMULAÇÕES EM PSIM

O primeiro passo tomado foi compreender o código-fonte escrito para as simulações em PSIM e parametrizá-lo a fim de ver os diferentes efeitos de aritméticas computacionais distintas com números de bits utilizados distintos na representação dos sistemas. Todas as simulações, que constavam em apenas um arquivo PSIM, foram separadas em um para cada conversor, para facilitar a análise individual.

Foram inseridos comentários para cada trecho pertinente do código para facilitar a alteração por futuros usuários e para justificar as variáveis adotadas em ambas as representações.

Em seguida, as equações foram parametrizadas definindo constantes globais no código, QBASE e NQ, acessíveis em linhas específicas. Desta forma, é possível mudar os parâmetros de circuito pela interface do PSIM e modificar a representação de aritmética de ponto fixo pela alteração dessas duas constantes.

Além disso, a parte referente à aritmética de ponto flutuante foi posta em representação por unidade para ter o mesmo tratamento da de ponto fixo.

O código para o conversor Buck está disponível em Anexo I. Os demais códigos modificados são semelhantes, sendo apenas a parte referente às equações de diferenças diferente.

3.3 REVISÃO DAS SIMULAÇÕES ANTERIORES

Para confirmar que as alterações feitas foram pertinentes e não introduziram erros às versões anteriores, todos os códigos foram testados para representações de 20 bits em ponto fixo e variáveis de precisão dupla em ponto flutuante.

O caso original apresentava comportamentos oscilatórios em regime permanente para aritmética de ponto fixo de magnitudes não previstas pelas decisões de projeto ou pelas simulações do próprio PSIM e de ponto flutuante. A hipótese levantada foi de que o número de bits utilizado em ponto fixo não conseguia representar o valor de regime permanente calculado quando convertido no sistema por unidade. Para uma análise própria entre os tipos de aritmética computacional, foi proposta a utilização de um número igual de bits para ambas.

A representação em Q31 para ponto fixo para variáveis em C do tipo *long long int* resulta em 64 bits disponíveis, sendo $2^{31} - 1$ o número que representa 1pu e seu complemento de 2 o que representa -1pu. Dessa forma, a multiplicação de dois números com essa representação resulta em um de até 64 bits, podendo ser positivo ou negativo. Todos os inteiros são normalizados novamente para a representação Q31 deslocando o resultado 31 bits para a direita.

As mesmas características notadas para o caso original foram observadas quando foram inseridos parâmetros para representação em Q20, como originalmente, para só então efetuar o teste em Q31. Com essa nova representação, notou-se melhora nos resultados, possuindo até uma proximidade maior à simulação de referência, a do PSIM, do que a representação em ponto flutuante com precisão dupla.

Foi mantida a propriedade de que um sinal não poderia mudar de sentido uma vez estabelecido um referencial. Ou seja, tensões e correntes para Buck e Boost sempre positivos e tensão de saída para o Buck-Boost sempre negativa; em caso de discordância com essa condição, a variável referida tornar-se-ia nula pelo passo de cálculo.

Todos os códigos foram editados e compilados pelo pacote incluso na IDE Dev-C++ 5.11 no modo 32 bits (TDM-GCC 4.9.2 32-bit modo Release).

3.4 PARAMETRIZAÇÃO E PARTIÇÃO EM MÓDULOS NO FPGA

Uma vez que os sistemas foram validados em simulação *offline*, foram tomados os mesmos procedimentos para modificar a descrição de *hardware* da DE-2.

Primeiramente, foi criado um módulo para exibir os estados de funcionamento do simulador, que irá comandar displays de 7 segmentos para exibir:

- On/Of, indicando se o relógio principal está ativo;
- bst,buc,b-b indicando se o conversor habilitado é o Boost, Buck ou Buck-Boost, respectivamente.

Em seguida, um módulo de interface com o conversor D/A foi criado a fim de separar todos os passos de preparação referentes à comunicação serial com o *hardware* específico.

Segundo a descrição original, havia uma resposta ao degrau toda vez que uma chave seletora de modo era acionada. Isso causava, às vezes, falha no envio de dados ao conversor D/A, resultando em uma saída inesperada. Para contornar esse problema,

cada conversor foi posto em um módulo próprio, que seria instanciado de tal forma a operar em paralelo. O seletor de modo iria tão somente conectar a saída de um dos módulos ao registrador referente aos dados do conversor D/A. Desta forma, cada módulo pode ser instanciado, modificado e replicado como necessário.

Depois de confirmado o funcionamento de cada módulo individualmente a partir da descrição original, eles foram modificados para ser possível a parametrização deles de tal forma a utilizar o número mínimo de bits para não haver falhas e de poder definir todos os principais parâmetros de cada conversor: indutância, capacitância, carga, passo de cálculo considerado, e bases do sistema por unidade. Uma entrada de reinício foi adicionada a cada módulo a fim de poder simular a resposta ao degrau inicial de cada modelo no instante em que for de interesse.

Finalmente, todos os comentários pertinentes ao funcionamento dos trechos de código foram adicionados e revisados para utilizações e modificações futuras e foi possível estabelecer uma equação para a equivalência entre a tensão exibida em osciloscópio e o valor inteiro obtido, considerando que serão considerados apenas os 16 bits mais significativos na representação de Q+1 bits:

$$V_{DA} = \left(\left\lfloor\left(\frac{V_{Q+1}}{2^{Q-16}}\right)\right\rfloor + 32768\right) \cdot \frac{5}{2^{16}}$$

Onde V_{DA} é a tensão analógica de saída do conversor D/A e V_{Q+1} é o número em representação Q tomado até seu bit Q+1 (para considerar sinal). A função piso é calculada antes de efetuada a multiplicação para considerar a perda da parte fracionária provocada pelo deslocamento de Q bits.

Por exemplo, para $V_{base} = 100V$ e $V_{conv} = 10V$:

$$V_{pu} = \frac{V_{conv}}{V_{base}} = 0{,}10pu$$

$$V_{Q31} = 0{,}10 \cdot (2^{31} - 1) = 214748364{,}7$$

Aplica-se a função piso (truncamento para variável inteira):

$$V_{int} = 214748364$$

Toma-se os 16 bits mais significativos a partir do bit Q+1:

$$V'_{int} = 0CCCH = 3276$$

$$V'_{int} + 32678 = 36044$$

$$V_{DA} = 36044 \cdot \frac{5}{2^{16}} = 2{,}82V$$

Para representar números negativos, toma-se o complemento de 2 do seu valor em pu na representação Q31 e repete-se o procedimento truncando o bit mais significativo durante a soma com 32768 (soma de dois números de 16 bits desconsiderando *overflow*).

Sempre se considera que o número de bits de representação Q é maior ou igual ao de bits disponíveis de saída (16, neste caso).

4 RESULTADOS E ANÁLISES

4.1 SIMULAÇÃO PSIM

As simulações em PSIM foram feitas para avaliar a precisão da aproximação adotada frente um *software* de referência para, em seguida, avaliar o desempenho de cada método de aritmética computacional empregado quanto ao erro gerado.

As análises têm foco nos valores de tensão do capacitor e corrente do indutor estipulados por projeto e tomados por referência da resposta do PSIM.

4.1.1 CONVERSOR BUCK

O circuito adotado na simulação está representado na Figura 9. A cada passo de cálculo do programa são obtidas respostas tanto da DLL carregada quanto do sistema equivalente gerado pelo PSIM internamente.

A tensão no capacitor e corrente no indutor, de interesse, foram calculadas e expressas em gráficos, presentes nas Figuras 10 e 11.

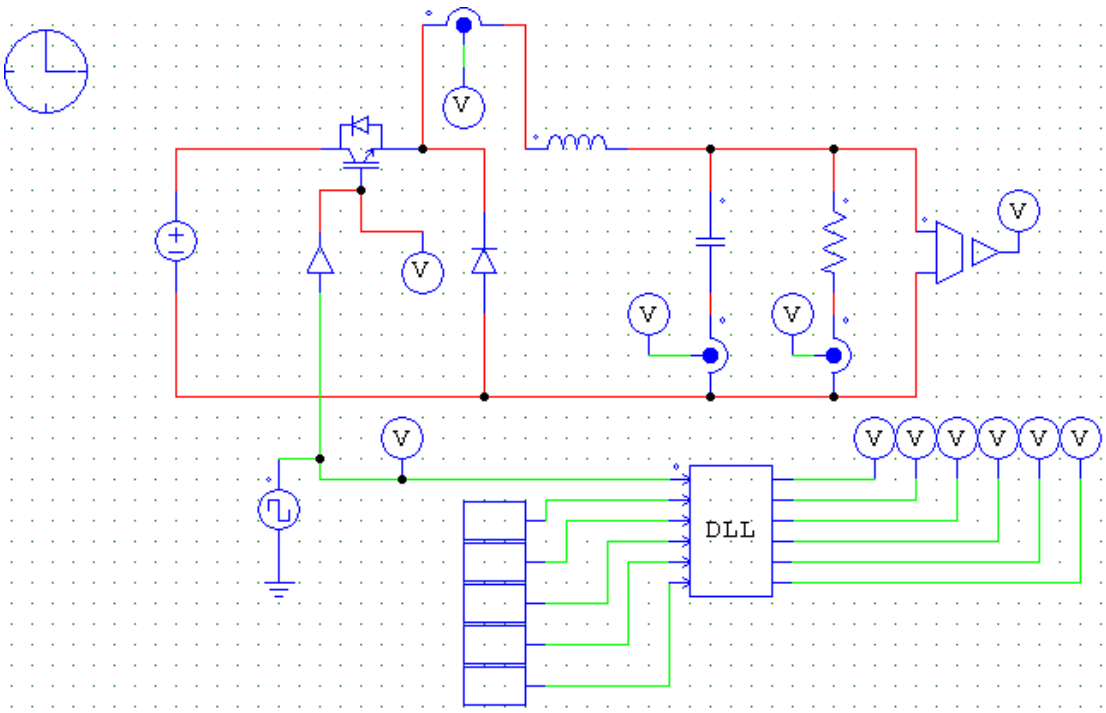

FIGURA 9 DIAGRAMA DE SIMULAÇÃO DO CONVERSOR BUCK

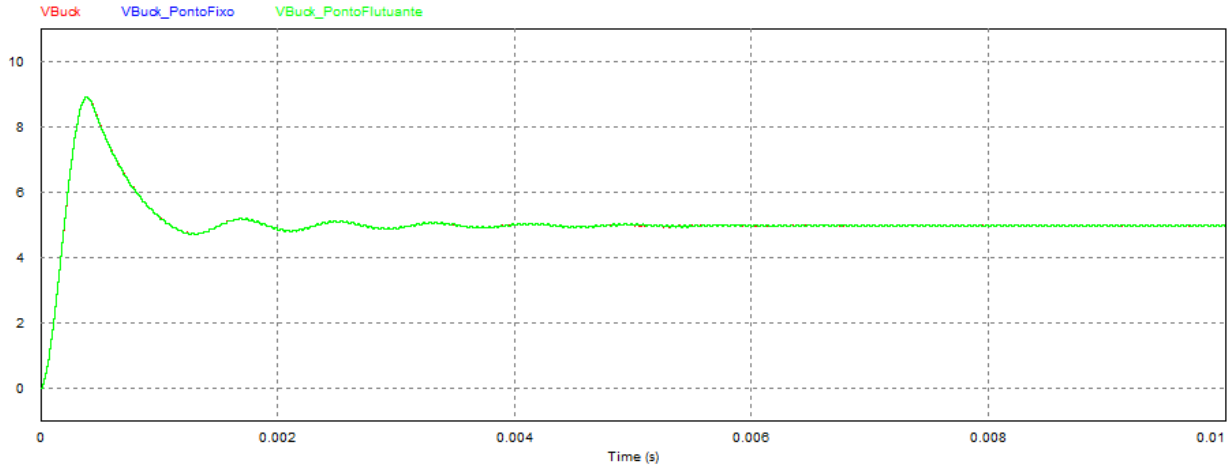

FIGURA 10 CONVERSOR BUCK - TENSÃO NO CAPACITOR

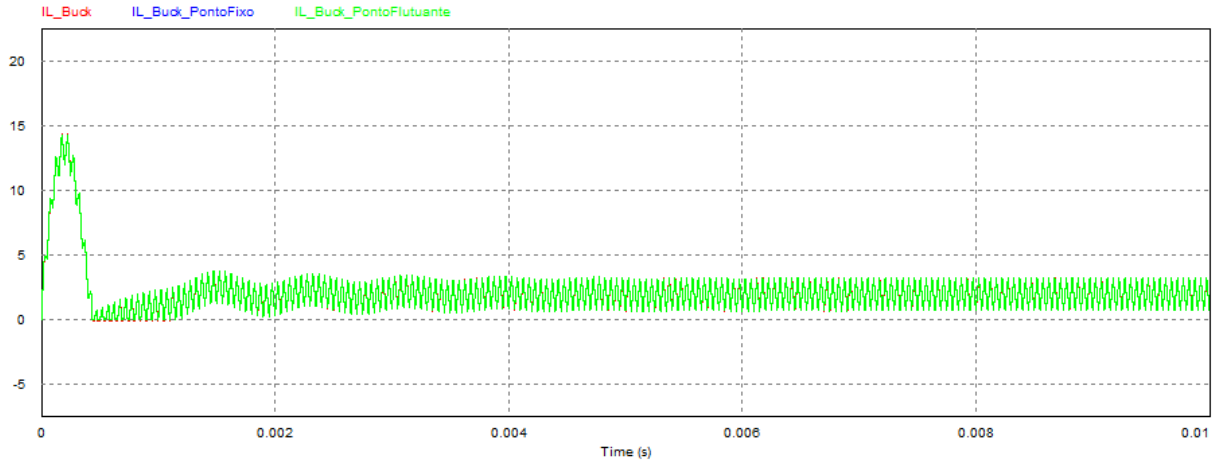

FIGURA 11 CONVERSOR BUCK - CORRENTE NO INDUTOR

Com essas Figuras é possível notar que não há discrepâncias significativas entre os modelos propostos, independentemente da aritmética considerada. As Figuras 12 e 13 representam detalhes de ambas as características para confirmar essa afirmação.

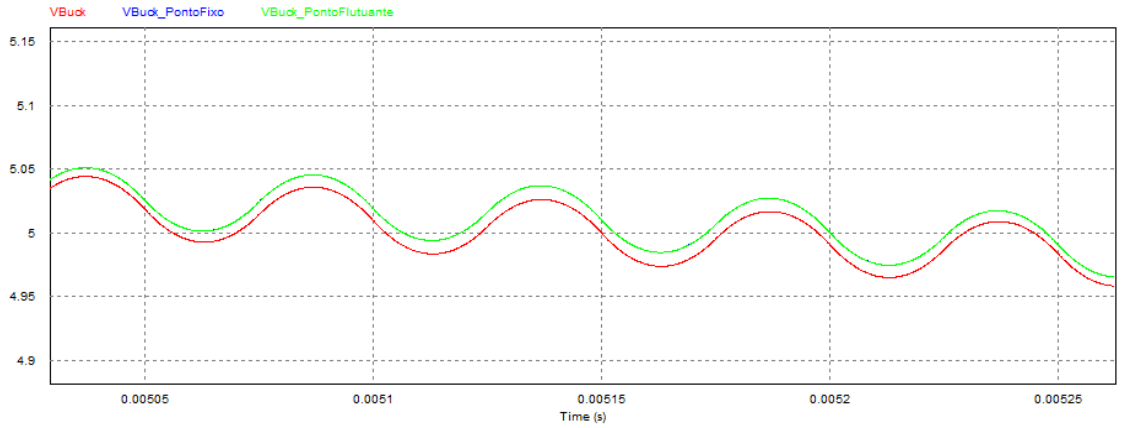

FIGURA 12 CONVERSOR BUCK - TENSÃO NO CAPACITOR (DETALHE)

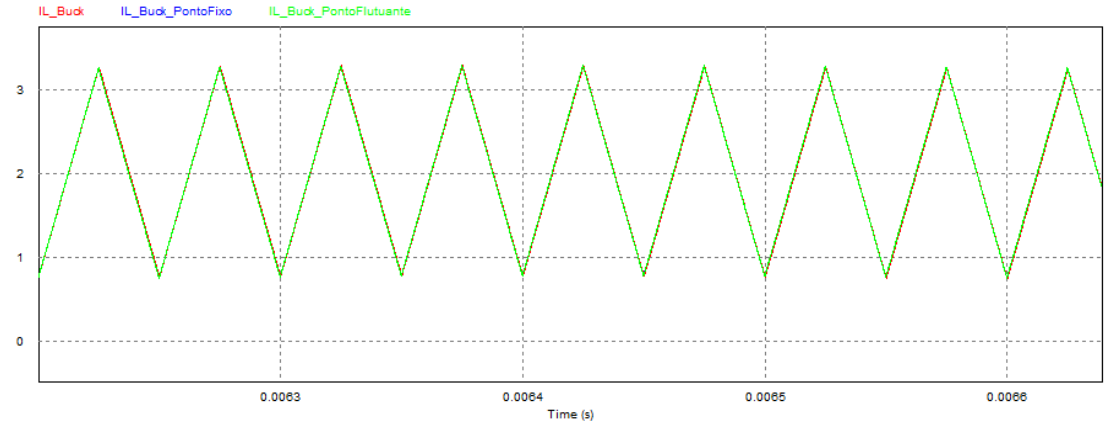

FIGURA 13 CONVERSOR BUCK - CORRENTE NO INDUTOR (DETALHE)

4.1.2 CONVERSOR BOOST

O circuito de simulação foi montado como na Figura 14, que contém elementos do PSIM e o bloco DLL de interface para os modelos criados em linguagem C.

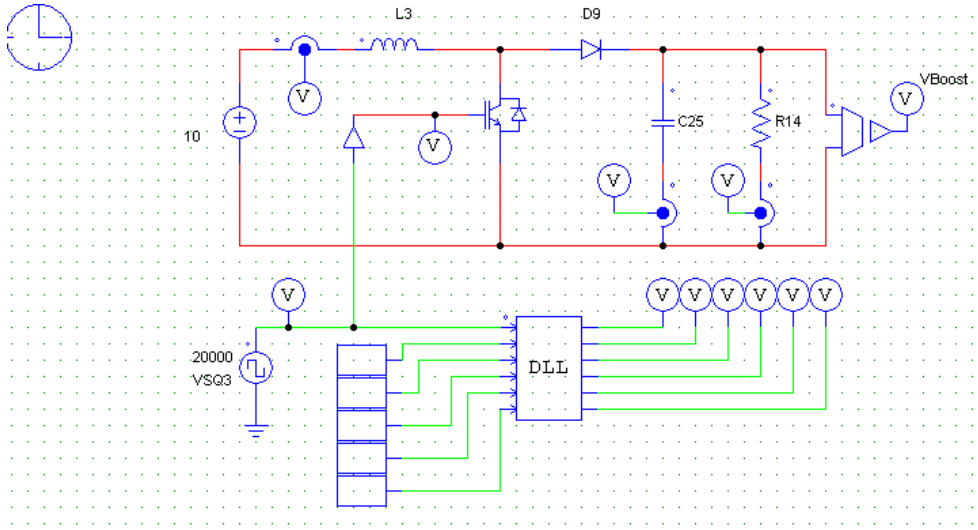

FIGURA 14 DIAGRAMA DE SIMULAÇÃO DO COVERSOR BOOST

A partir deles, obtiveram-se as Figuras 15 e 16 para as grandezas de circuito de interesse e nota-se a convergência das características de regime permanente tanto da simulação do PSIM em relação aos modelos matemáticos aproximados quanto de ambas as representações deste entre si.

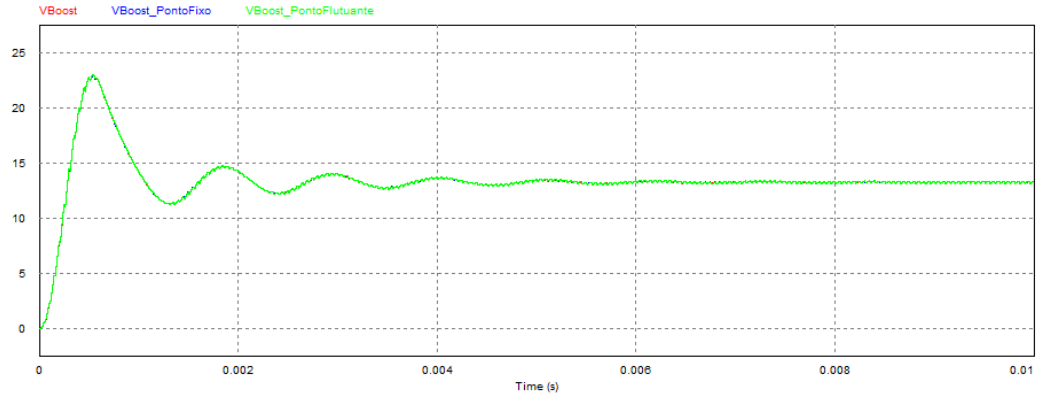

FIGURA 15 CONVERSOR BOOST - TENSÃO NO CAPACITOR

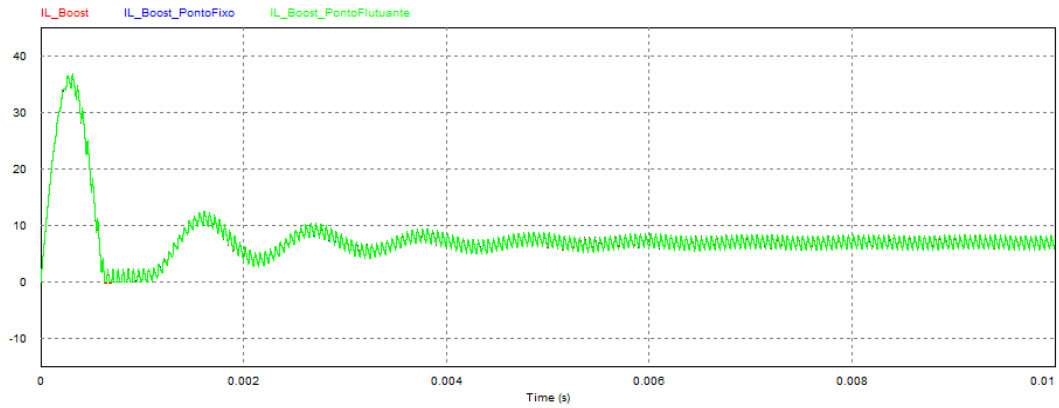

FIGURA 16 CONVERSOR BOOST - CORRENTE NO INDUTOR

As Figuras 17 e 18 comprovam a pequena diferença entre as características tomando detalhes de cada curva.

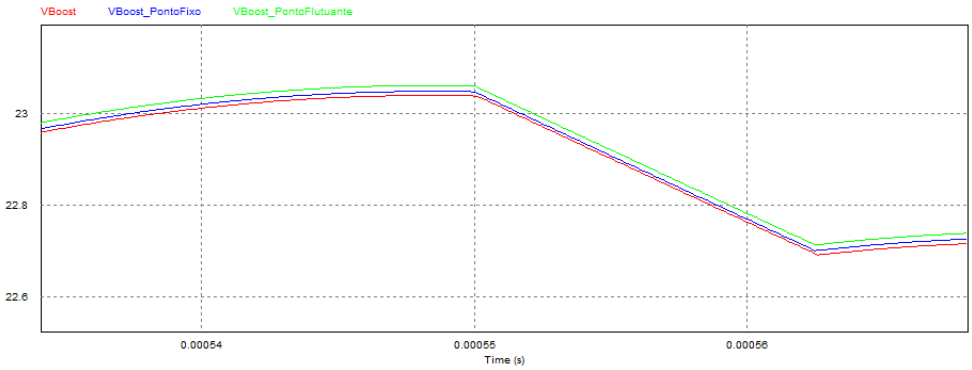

FIGURA 17 CONVERSOR BOOST - TENSÃO NO CAPACITOR (DETALHE)

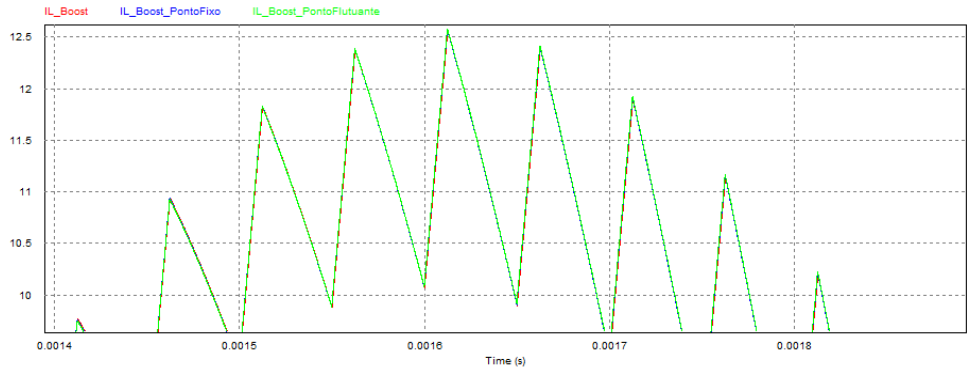

FIGURA 18 CONVERSOR BOOST - CORRENTE NO INDUTOR (DETALHE)

4.1.3 CONVERSOR BUCK-BOOST

Seguindo a sequência realizada com os conversores anteriores, o circuito utilizado para simulação encontra-se representado na Figura 19.

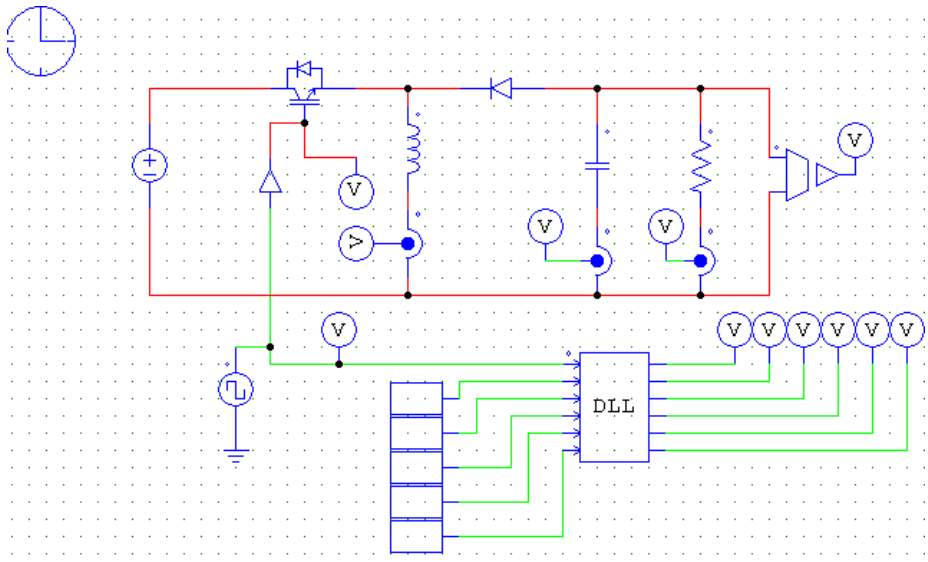

FIGURA 19 DIAGRAMA DE SIMULAÇÃO DO CONVERSOR BUCK-BOOST

Os resultados para tensão no capacitor de saída e corrente no indutor encontram-se desenhados nas Figuras 20 e 21.

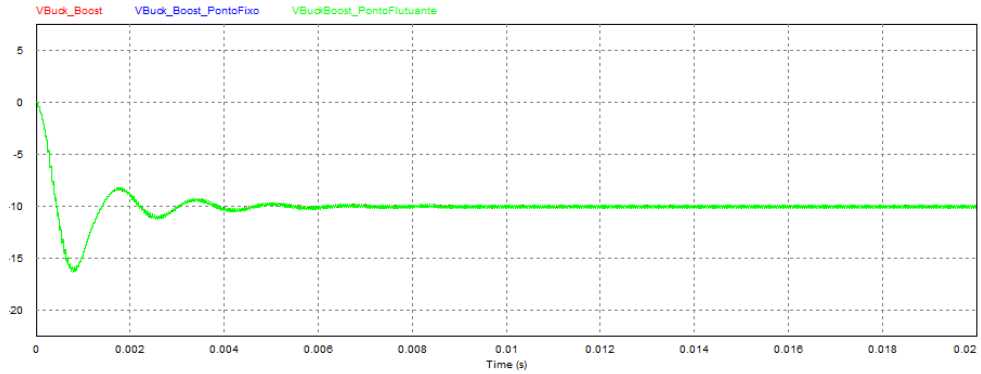

FIGURA 20 CONVERSOR BUCK-BOOST - TENSÃO NO CAPACITOR

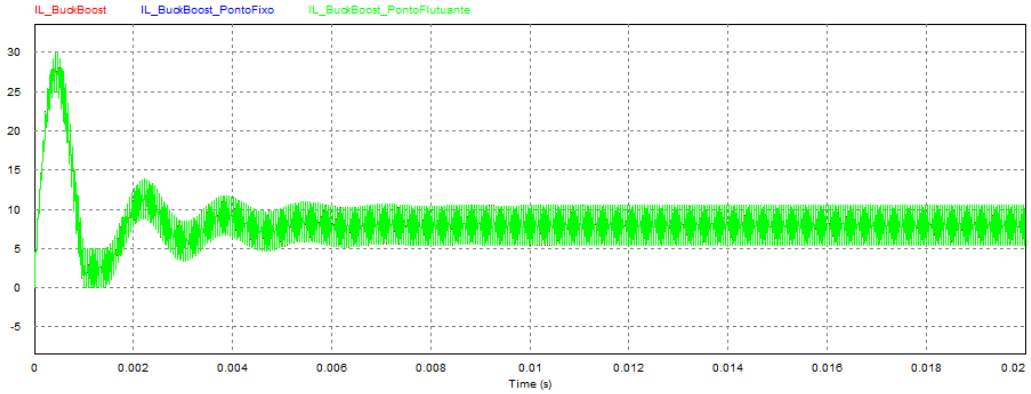

FIGURA 21 CONVERSOR BUCK-BOOST - CORRENTE NO INDUTOR

Nota-se que não há divergência entre o modelo proposto e o de referência do PSIM, e observa-se a pequena diferença entre eles pelos detalhes nas Figuras 22 e 23.

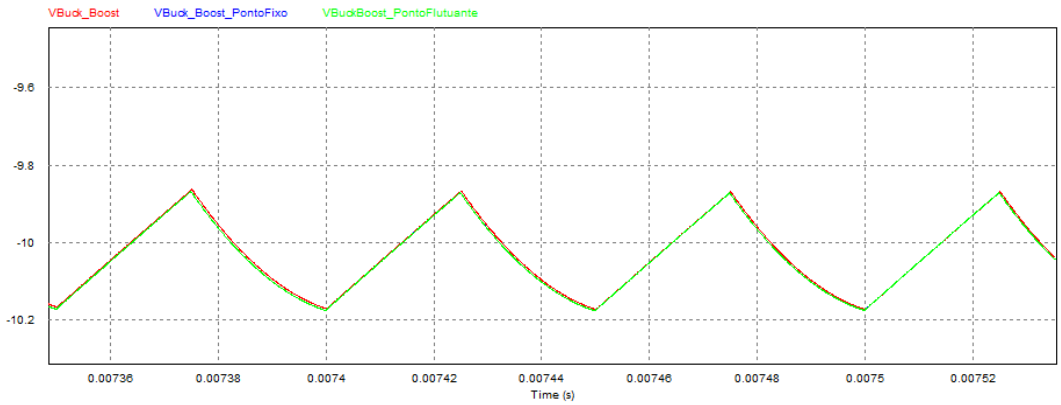

FIGURA 22 CONVERSOR BUCK-BOOST - TENSÃO NO CAPACITOR (DETALHE)

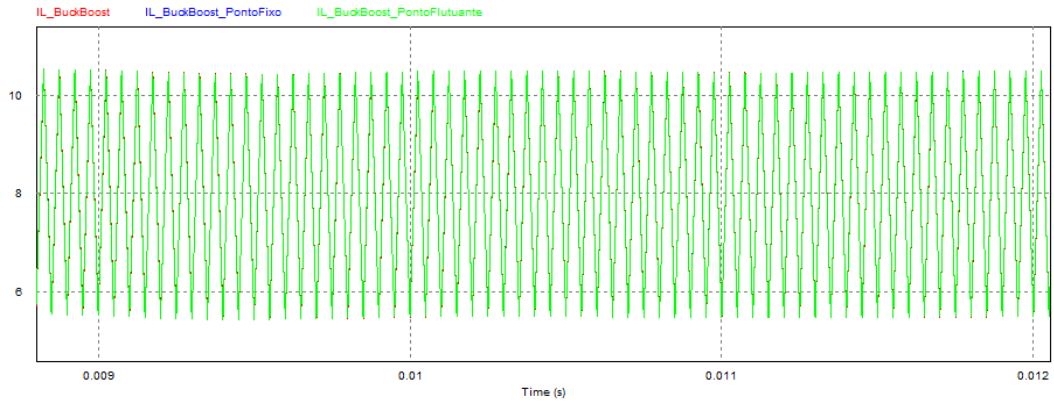

FIGURA 23 CONVERSOR BUCK-BOOST - CORRENTE NO INDUTOR (DETALHE)

4.2 SIMULAÇÃO FPGA

Cada parte componente do módulo denominado *mod_teste* foi testada individualmente a fim de garantir seus funcionamentos. O exibidor de estados foi verificado para haver correspondência entre o desenhado nos *displays* de 7 segmentos com a lógica de seleção da saída; o conversor D/A teve sua saída definida pela posição de 16 chaves digitais que definiam cada um de seus *bits* a fim de confirmar os valores esperados de tensão de saída para certa palavra de entrada. Os módulos do exibidor de estados, da interface com o conversor, da comunicação serial, do módulo Buck e do módulo de testes encontram-se nos Anexos II, III, IV, V e VI, respectivamente.

Buscou-se atender a todas as mensagens de erro e advertência exibidas pelo compilador durante suas várias fases. Embora vários dos principais problemas tenham sido resolvidos, resultando em nenhum erro, porém várias advertências, uma das principais advertências não pôde ser resolvida: "*Critical warning: timing requirements not met*". Isso indica que há a possibilidade do sistema não estar operando exatamente no tempo de execução proposto, porém os resultados do osciloscópio indicam o contrário.

O processo de parametrização dos módulos teve um conjunto de testes isolado próprio para garantir que as constantes eram iniciadas com os valores calculados externamente, para garantir que a sequência de multiplicações e truncamentos para o número de *bits* desejado fosse mantido.

4.2.1 MÓDULO BUCK

O módulo foi configurado para o funcionamento igual à simulação para representação Q31. Ao configurar o osciloscópio de modo a exibir na tela seu regime transitório, foram obtidas as curvas características da Figura 24. A tensão esperada calculada é 2,75V. O canal 2 representa a tensão e o canal 4 a corrente do modelo. A forma de onda de ambas as grandezas condizem com o valor de simulação, porém o valor de tensão, por exemplo, aproxima-se de 2,60V, 15mV aquém do valor calculado.

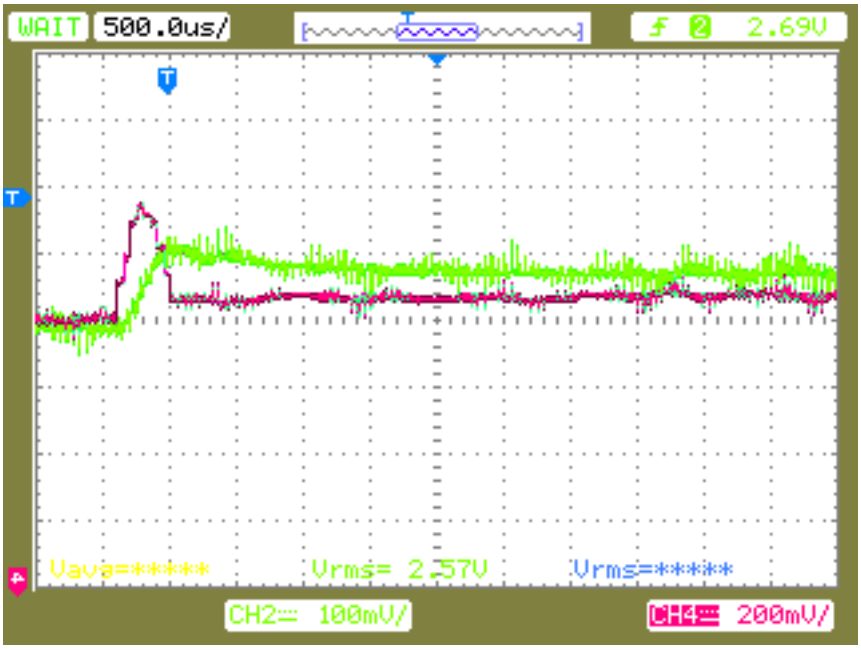

FIGURA 24 MÓDULO BUCK - TENSÃO E CORRENTE

4.2.2 MÓDULO BOOST

Os mesmos procedimentos foram feitos para a criação do módulo Boost, colocando as equações adequadas nas posições correspondentes do código em Verilog. O resultado é apresentado pela Figura 25. A tensão esperada é de 2,82V. Semelhantemente, nota-se que ambas as formas de onda se aproximam dos resultados de simulação, sendo o canal 2 representativo da tensão sobre o capacitor, e o canal 4 da corrente no indutor. A configuração mostra que o valor calculado e o valor experimentalmente obtido são muito próximos, comprovando que o modelo simulado está refletido na resposta deste módulo.

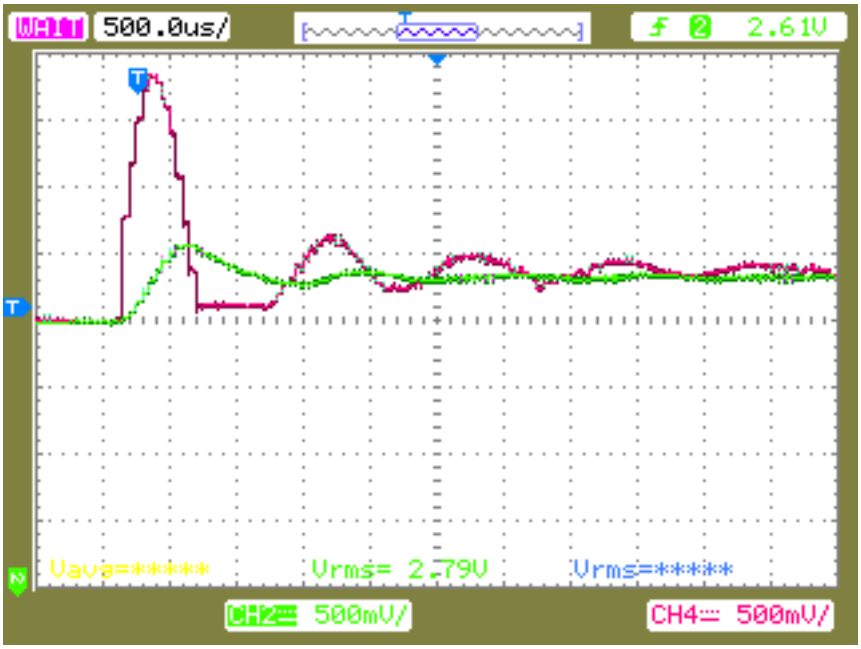

FIGURA 25 - MÓDULO BOOST - TENSÃO E CORRENTE

4.2.3 MÓDULO BUCK-BOOST

O módulo segue os procedimentos utilizados para os anteriores. As suas equações características foram inseridas nos locais adequados e o módulo segue os parâmetros dos demais para se igualar a sua simulação. A Figura 26 representa o transitório capturado pelo osciloscópio quando este módulo está acionado. Similarmente, o canal 2 representa a tensão sobre o capacitor e o canal 4 a corrente do indutor.

O valor esperado para este módulo é de 2,25V. Entretanto, o valor lido no osciloscópio tende a 2,4V em regime permanente. Embora ambos os sinais tenham as características de forma dos sinais da simulação, não há correspondência entre a tensão medida e o valor de cálculo esperado.

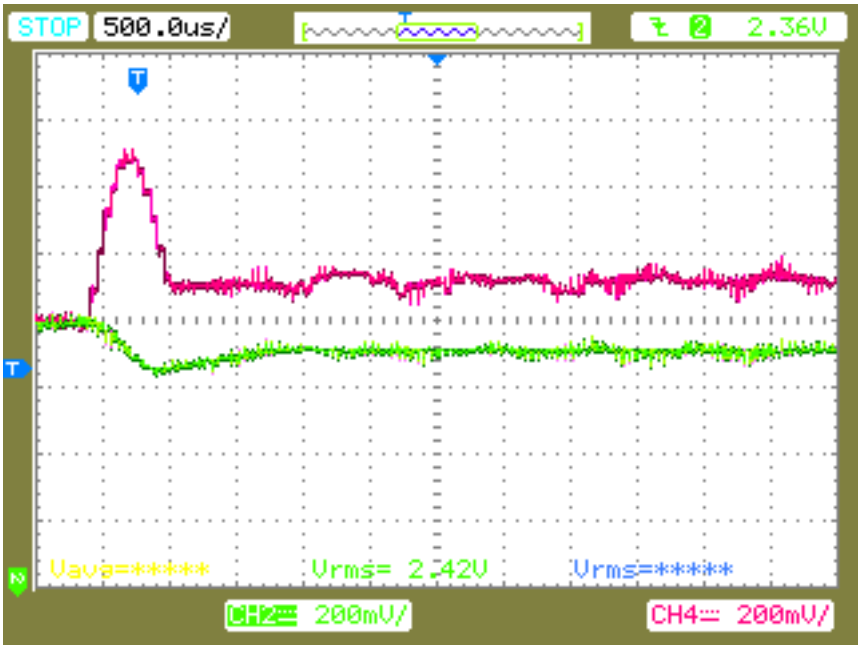

FIGURA 26 MÓDULO BUCK-BOOST - TENSÃO E CORRENTE

4.2.4 SISTEMA DESCRITO EM VERILOG

O sistema obtido por descrição em *hardware* em linguagem Verilog está representado de forma sucinta em diagrama de blocos na Figura 27. Destaca-se que módulos foram explicitamente instanciados, o que era feito de forma implícita no sistema anterior (Figura 7). Uma chave adicional foi atribuída a um sinal *reset* que irá impor condições nulas aos módulos instanciados paralelamente. Um multiplexador (feito em estrutura *always* não descrito em módulo específico devido à ocorrência de erros) seleciona a tensão e corrente de apenas um dos módulos e guarda o sinal em registradores adequados para passagem ao módulo spi que se comunica com o a interface do conversor D/A. Há também um módulo exibidor_estado que indica nos mostradores de sete segmentos da placa se o relógio principal está habilitado, a depender de areset, e qual saída foi multiplexada, a depender de um conjunto de chaves Sel adotado.

FIGURA 27 - DIAGRAMA DE BLOCOS DO SISTEMA MODIFICADO

4.2.5 RESULTADOS INESPERADOS

Foi observado durante a experimentação alguns resultados cuja origem não foi identificada. A instanciação dos módulos Buck e Boost promove um funcionamento adequado do sistema. Porém, instanciar conjuntamente a ambos ou a um deles o módulo Buck-Boost faz com que o conversor D/A não funcione corretamente ou de forma linear após reinício. Por exemplo, o valor de regime permanente do Boost pode aparecer como 1,4V ou 3,7V (diferentemente dos 2,82V esperados), além de que os transitórios de operação aparentam possuir tensões menores ou maiores, de acordo.

Talvez haja a necessidade de colocar mais camadas de chaveamento na codificação em Verilog, porém as soluções exploradas não resolveram esses problemas. A instanciação de cada módulo individualmente parece fornecer sistemas mais próximos dos esperados, embora não haja um motivo claro para tal.

Outro fato que não pôde ser explicado é a inserção de mais dados em outros canais do conversor D/A provocar interferência nos demais. Embora seus canais sejam modificados por comandos via transmissão serial, há oito registradores de dados para os oito canais disponíveis do conversor. Se apenas dois canais forem inseridos (tensão do capacitor e corrente no indutor de um dos módulos), o dispositivo funciona como nos resultados mostrados. Caso outro valor diferente de uma constante seja imposto em um dos registradores de dados, este provoca interferência na saída que descaracteriza totalmente o sinal esperado.

5 CONCLUSÃO

A principal contribuição desse trabalho foi a partição dos módulos simuladores em partes independentes entre si. A partir desse ponto, basta criar outro modelo que siga os mesmos requisitos de tempo que os atualmente existentes e instanciá-lo no módulo principal de testes.

Como efeito secundário, várias pendências referentes a documentação e confirmação de hipóteses puderam ser resolvidas em cada uma das partes constituintes do projeto, além de indicar novas possibilidades de desenvolvimento.

Espera-se que o sistema criado possa ser utilizado para diversos fins, como acadêmicos ou ilustrativos, tanto no âmbito da UFCG, quanto de outras instituições que atuam na área de Eletrônica de Potência e Processamento de Energia.

BIBLIOGRAFIA

Abourida, S., Belanger, J., & Dufour, C. (2005). Real-Time HIL Simulation of a Complete PMSM Drive at 10us Time Step. *IEEE Power Electronics and Applications* , 9 pp. - P.9.

França, B. W. (2009). HARDWARE-IN-THE-LOOP PARA DESENVOLVIMENTO DE SOFTWARE EMBARCADO EM DSPs UTILIZANDO AMBIENTE PSCAD/EMTDC. Rio de Janeiro: Universidade Federal do Rio de Janeiro.

Hüller, D. R. (Maio de 2014). SIMULAÇÃO EM TEMPO REAL DA MÁQUINA SÍNCRONA A IMÃ PERMANENTE IMPLEMENTADA EM DISPOSITIVOS LÓGICOS PROGRAMÁVEIS (FPGA - FIELD PROGRAMMABLE GATE ARRAY). Campina Grande: Universidade Federal de Campina Grande.

Mendes, L. L. (2015). *SIMULAÇÃO DE CONVERSORES CC-CC EM TEMPO REAL COM FPGA.* Campina Grande: Universidade Federal de Campina Grande.

Oppenheim, A. (2010). In: *Discrete Time Signal Processing* (3ª ed., p. 504). Upper Saddle River, New Jersey: Pearson Higher Education.

Petry, C. A. (2014). Capítulo 12: Conversores CC-CC: Conversor Buck.

Robert W. Erickson, D. M. (2001). Fundamentals of Power Eletronics. Bolder, Colorado: Kluwer Academic Publishers.

Siskind, C. S. (1963). *Electrical Control Systems in Industry.* Nova Iorque: McGraw-Hill.

ANEXO I – SIMULAÇÃO C – BUCK

```
#include <math.h>
#include <stdio.h>

//Define-se inicialemente
#define QBASE 31
//NQ = 2^QBASE - 1
//#define NQ 1048575 //2^20-1
#define NQ 2147483647 //2^31-1
__declspec(dllexport) void simuser (t, delt, in, out)

// Note that all the variables must be defined as "double"
double t, delt;
double *in, *out;

{
// Place your code here.....begin
double static vetor[6]={0,0,0,0,0,0};

double Tpwm=1*delt;
int ind=0;
double static Tch=0.0;
double static Tch1=0.0;

static double h=0;
static double vin=0;
static double indut=0;
static double indutf=0;
static double cap=0;
static double capf=0;
static double res=0;
static int gate=0;

/*
O valor maior sempre representa a saída. x[2] [e o valor mais atual e x[0]
o mais antigo
*/
static double il[3]={0,0,0};
static double vl[3]={0,0,0};
static double ic[3]={0,0,0};
static double vc[3]={0,0,0};
static double ir[3]={0,0,0};

/*
Base para o sistema PU utilizado
*/
```

```
const static double vbase = 100;
const static double ibase = 50;
const static double zbase = 100.0/50.0;

/*
Constantes de 64bits (63b+1bsinal)
*/
static signed long long int k1 = 0;
static signed long long int k2 = 0;
static signed long long int k3 = 0;
static signed long long int k4 = 0;
static signed long long int kcr = 0;

/*
Variáveis em ponto fixo de 64bits (63b+1bsinal)
*/
static signed long long int ilf[3]={0,0,0};
static signed long long int vlf[3]={0,0,0};
static signed long long int icf[3]={0,0,0};
static signed long long int vcf[3]={0,0,0};
static signed long long int irf[3]={0,0,0};

static signed long long int vinf=0;

/*
Alocação das variáveis de entrada no PSIM p/ variáveis deste programa
*/
gate=in[0];
indut=in[1];
cap=in[2];
vin=in[3];
res=in[4];
h = Tpwm;

/*Variáveis de circuito em PU*/
indut = indut/zbase;
cap = cap*zbase;
vin = vin/vbase;
res = res/zbase;


/*
Inicialização de constantes de ponto fixo
*/
indutf = indut;
capf = cap;

k1 = NQ*(double)(h/indutf);
k2 = NQ*(double)(h/(2*capf));
k3 = NQ*(double)(h/(2*indutf));
```

```
k4 = NQ*(double)(h/(2*capf));

kcr = NQ*(double)(1/(res));
k2 = ((signed long long int)(k2*kcr))>>QBASE;


vinf = NQ*(vin);

// Rotina em ponto flutuante
if (t> Tch)
{

      Tch = Tch + Tpwm;
//chave na posicao 1
      if(gate)
      {
            il[1]=il[0];
            il[0]=vin*(h/indut)-(vc[1]+vc[2])*(h/(2*indut))+il[1];
            if(il[0]<0.0) il[0]=0.0;

            vc[2]=vc[1];
            vc[1]=vc[0];
            vc[0]=(il[0] + il[1])*(h/(2*cap)) - (vc[1] + vc[2])*(h/(res*2*cap)) + vc[1];
            if(vc[0]<0.0) vc[0]=0.0;
      }
//chave na posicao 2
      else
      {
            il[1]=il[0];
            il[0]=-(vc[1]+vc[2])*(h/(2*indut))+il[1];
            if(il[0]<0.0) il[0]=0.0;

            vc[2]=vc[1];
            vc[1]=vc[0];
            vc[0]=(il[0] + il[1])*(h/(2*cap)) - (vc[1] + vc[2])*(h/(res*2*cap)) + vc[1];
            if(vc[0]<0.0) vc[0]=0.0;
      }
}
// Rotina em ponto fixo
if (t> Tch1)
{
      Tch1 = Tch1 + Tpwm;
//chave na posicao 1
      if(gate)
      {
            ilf[1] = ilf[0];
            ilf[0] = (((signed long long int)(k1*vinf))>>QBASE) - (((signed long long
int)(k3*(vcf[1]+vcf[2])))>>QBASE) + ilf[1];
            if(ilf[0]<0) ilf[0]=0;
```

```
            vcf[2] = vcf[1];
            vcf[1] = vcf[0];
            vcf[0] = (((signed long long int)(k4*(ilf[0] + ilf[1])))>>QBASE) - (((signed
long long int)(k2*(vcf[1] + vcf[2])))>>QBASE) + vcf[1];
            if(vcf[0]<0) vcf[0]=0;

      }
//chave na posicao 2
      else
      {
            ilf[1] = ilf[0];
            ilf[0] = -(((signed long long int)(k3*(vcf[1]+vcf[2])))>>QBASE) + ilf[1];
            if(ilf[0]<0) ilf[0]=0;

            vcf[2] = vcf[1];
            vcf[1] = vcf[0];
            vcf[0] = (((signed long long int)(k4*(ilf[0] + ilf[1])))>>QBASE) - (((signed
long long int)(k2*(vcf[1] + vcf[2])))>>QBASE) + vcf[1];
            if(vcf[0]<0) vcf[0]=0;
      }
}

out[0]=vc[0]*vbase;
out[1]=il[0]*ibase;
out[2]=(vc[0]/res)*ibase;
out[3]=(il[0]-vc[0]/res)*ibase;
out[4]=(vcf[0]*vbase)/(NQ);
out[5]=(ilf[0]*ibase)/(NQ);

}
```

ANEXO II – EXIBIDOR DE ESTADOS

```
module exibidor_estado(chave,
                                                    fHEX0,
                                                    fHEX1,
                                                    fHEX2,
                                                    fHEX4,
                                                    fHEX5);
        input [17:0]chave;
        output reg [7:0]fHEX0;
        output reg [7:0]fHEX1;
        output reg [7:0]fHEX2;
        output reg [7:0]fHEX4;
        output reg [7:0]fHEX5;

        always @ (chave[17:16])
        begin
        if(chave[0])
                begin
                        //OF
                        fHEX4           =       7'h0E;
                        fHEX5           =       7'h40;
                end
        else
                begin
                        //ON
                        fHEX4           =       7'h2b;
                        fHEX5           =       7'h40;
                end
        case(chave[17:16])
                2'b00://BST
                        begin
                                fHEX0           =       7'h07;
                                fHEX1           =       7'h12;
                                fHEX2           =       7'h03;
                        end
                2'b01://BUC
                        begin
                                fHEX0           =       7'h46;
                                fHEX1           =       7'h41;
                                fHEX2           =       7'h03;
                        end

                2'b10://BB
                        begin
                                fHEX0           =       7'h03;
                                fHEX1           =       7'h3F;
```

```
                        fHEX2       =       7'h03;
                    end
                2'b11://NOP
                    begin
                        fHEX0       =       7'h0c;
                        fHEX1       =       7'h40;
                        fHEX2       =       7'h2b;
                    end
            endcase
        end//endalways
endmodule
```

ANEXO III – INTERFACE D/A

```
module interface_DA_LTC2600CGN(
	clk_in,
	start,
	new_data,
	fio_busy,
	fio_mosi,
	fio_sck,
	fio_cs,
	index_dado
);
//Conversor D/A de 16 bits LTC 2600CGN
input clk_in;
output reg start;
output reg new_data;
input fio_busy;
input fio_mosi;
input fio_sck;
input fio_cs;


reg [19:0] cont2;
reg [19:0] cont_start;
reg [19:0] cont_ndata;
reg [5:0] cont_busy;

reg busy_state;

output reg [3:0]index_dado;

always@(posedge clk_in)
begin

	cont2 = cont2 + 20'b1;
	if(cont2 >= 20'd10000)
	begin
		cont2 = 20'd0;
		if(!fio_busy)
		begin
			if(!start) cont_start = 20'd0;
			if(!new_data) cont_ndata = 20'd0;
			start = 1;
			index_dado = 4'd0;
		end
	end
```

```
        if(start)
        begin
                cont_start = cont_start + 20'b1;
                if(cont_start == 20'd10)
                begin
                        new_data = 1;
                        start = 0;
                        cont_start = 20'd0;
                end
        end

        if(new_data)
        begin
                cont_ndata = cont_ndata + 20'd1;
                if(cont_ndata == 20'd10)
                begin
                        new_data = 0;
                        cont_ndata = 0;
                        if(index_dado < 4'd7) busy_state = 1;
                        cont_busy = 6'd0;
                end
        end

        if( (busy_state) && (!fio_busy) )
        begin
                cont_busy = cont_busy + 6'd1;
                if(cont_busy == 6'd10)
                begin
                        busy_state = 0;
                        if(!start) cont_start = 0;
                        if(!new_data) cont_ndata = 0;
                        start = 1;
                        index_dado = index_dado + 4'b1;
                        cont_busy=6'd0;
                end
        end
end

endmodule
```

ANEXO IV – COMUNICAÇÃO SERIAL

```
module spi (
	sck_in,
	miso,
	dado,
	new_data,
	start,
	sck,
	mosi,
	cs,
	busy);

input sck_in;
input miso;
input [23:0] dado [7:0];
input new_data;
input start;

output reg sck;
output reg mosi;
output reg cs;
output reg busy;

//reg [15:0] cont;
//reg [25:0] cont_ck;
reg [7:0] cont_ck1;
reg [4:0] index_bit;
reg [3:0] index_dado;

always@(posedge sck_in)
begin
//	cont_ck = cont_ck+1;
//	if(cont_ck>=2)
//	begin
//		cont_ck=0;
//		cont = cont+1;
//	end

	if(busy)
	begin
		cont_ck1 = cont_ck1 + 1;

		if(cont_ck1 == 2) mosi = dado[index_bit-1][index_dado];
		if(cont_ck1 == 100) sck = sck^1;
		if(cont_ck1 == 200)
		begin
```

```
                        cont_ck1 = 0;
                        sck = sck^1;
                        index_bit = index_bit - 1;
                        if(index_bit == 0)
                        begin
                                index_dado = index_dado + 1;
                                index_bit = 24;
                        end

                        if(index_dado>7)
                        begin
                                busy = 0;
                                cont_ck1 = 0;
                                index_bit = 24;
                                index_dado = 0;
                                cs = 1;
                        end
                end
        end

        if(new_data)
        begin
                busy = 1;
                cs = 0;
        end

        if(start)
        begin
                busy = 0;
                cont_ck1 = 0;
                index_bit = 24;
                index_dado = 0;
        end


end
endmodule
```

ANEXO V – MÓDULO BUCK

```
module conv_cc_cc_buck(clk_200M,gate,reset,vc,il);
////////////////////////////////////////////////////////////
//                vbase = 100;
//                ibase = 50;

//                indutor = 50uH
//                capacitor = 330uF
//                resistor = 2.5 ohms
//                Vin = 10V
////////////////////////////////////////////////////////////
input clk_200M;
input gate;
input reset;
output reg signed [lim_sup:0]vc;
output reg signed [lim_sup:0]il;

reg signed [lim_sup:0]vcf[2:0];
reg signed [lim_sup:0]ilf[1:0];

////////////////////////////////////////////////////
//Representacao em ponto fixo
parameter      Q                              =      31;//31bits p/ 0-0.99pu
//parameter NQ = 1048575//2^20-1
parameter NQ = 2147483647; //2^31-1
parameter num_bits=2*(Q+1);
parameter lim_sup=num_bits-1;
////////////////////////////////////////////////////
////////////////////////////////////////////////////
//Parametros de circuito
parameter      real vbase                     =      100;//V
parameter      real ibase                     =      50;//A
parameter      real zbase                     =      vbase/ibase;//ohms

parameter      real indut                     =      50e-6;//H
parameter      real cap                       =      330e-6;//F
parameter      real res               =      2.5;//ohms
parameter      real vin                       =      10;//V
parameter      real h                         = 1e-7;//passo de calculo

parameter real indutp = indut/zbase;
parameter real capp           = cap*zbase;
parameter real vinp           = vin/vbase;
parameter real resp           = res/zbase;

//Constantes em Ponto Fixo
```

```
parameter integer               k1              = NQ*(h/indutp);      // k1 = 4294967
parameter integer       k2      = NQ*(h/(2*capp*resp));       //
parameter integer       k3      = NQ*(h/(2*indutp));      //
parameter integer       k4      = NQ*(h/(2*capp));        //
parameter integer               vinf     = NQ*vinp;
//////////////////////////////////////////////////
//Inicialização
parameter integer VAL_INIT=0;

reg [7:0] cont;

always @ (posedge clk_200M)
begin
        if(reset)
        begin
                vcf[2]=VAL_INIT;
                vcf[1]=VAL_INIT;
                vcf[0]=VAL_INIT;
                ilf[1]=VAL_INIT;
                ilf[0]=VAL_INIT;

                vc=VAL_INIT;
                il=VAL_INIT;

                cont=0;
        end
        else
                begin
                cont = cont + 8'd1;
                if(cont>8'd19)
                begin
                        if(gate)
                        begin
                //chave na posição 1
                                ilf[1]=ilf[0];
                                ilf[0]=((k1*vinf)>>>Q)-
((k3*(vcf[1]+vcf[2]))>>>Q)+ilf[1];
                                if(ilf[0]<0) ilf[0]=0;

                                vcf[2]=vcf[1];
                                vcf[1]=vcf[0];

        vcf[0]=((k4*(ilf[0]+ilf[1]))>>>Q)+((k2*(vcf[1]+vcf[2]))>>>Q)+vcf[1];
                                if(vcf[0]<0) vcf[0]=0;
                        end
                //chave na posição 2
                        else
                                begin
                                        ilf[1]=ilf[0];
                                        ilf[0]=ilf[1]-((k3*(vcf[1]+vcf[2]))>>>Q);
```

```
                                        if(ilf[0]<0) ilf[0]=0;

                                        vcf[2]=vcf[1];
                                        vcf[1]=vcf[0];
                                        vcf[0]=((k4*(ilf[0]+ilf[1]))>>>Q)-
((k2*(vcf[1]+vcf[2]))>>>Q)+vcf[1];
                                        if(vcf[0]<0) vcf[0]=0;
                                end
                                cont=8'd0;
                                vc=vcf[0];
                                il=ilf[0];
                        end
        end//else-reset
end//always
endmodule
```

ANEXO VI – MÓDULO DE TESTES

```
module Mod_Teste
	(
		////////////////////	Clock Input	 	////////////////////
		CLOCK_27,				//	27 MHz
		CLOCK_50,				//	50 MHz

		////////////////////	Push Button		////////////////////
		KEY,					//	Pushbutton[3:0]

		////////////////////	DPDT Switch		////////////////////
		SW,					//	Toggle Switch[17:0]

		////////////////////	7-SEG Display	////////////////////
		HEX0,					//	Seven Segment Digit 0
		HEX1,					//	Seven Segment Digit 1
		HEX2,					//	Seven Segment Digit 2
		HEX3,					//	Seven Segment Digit 3
		HEX4,					//	Seven Segment Digit 4
		HEX5,					//	Seven Segment Digit 5
		HEX6,					//	Seven Segment Digit 6
		HEX7,					//	Seven Segment Digit 7

		////////////////////////	LED		////////////////////////
		LEDG,				//	LED Green[8:0]
		LEDR,				//	LED Red[17:0]

		////////////////////////	UART ////////////////////////
		UART_TXD,			//	UART Transmitter
		UART_RXD,			//	UART Receiver

		////////////////////	LCD Module 16X2		////////////////
		LCD_ON,			//	LCD Power ON/OFF
		LCD_BLON,			//	LCD Back Light ON/OFF
		LCD_RW,		//	LCD Read/Write Select, 0 = Write, 1 = Read
		LCD_EN,		//	LCD Enable
		LCD_RS,//	LCD Command/Data Select, 0 = Command, 1 = Data
		LCD_DATA,	//	LCD Data bus 8 bits

		////////////////////	GPIO	////////////////////////////
		GPIO_0,			//	GPIO Connection 0
		GPIO_1			//	GPIO Connection 1
	);

////////////////////////	Clock Input	 	////////////////////////
input			CLOCK_27;		//	27 MHz
```

```
input               CLOCK_50;                   //      50 MHz

//////////////////////    Push Button     //////////////////////
input   [3:0]   KEY;                        //      Pushbutton[3:0]

//////////////////////    DPDT Switch     //////////////////////
input   [17:0]  SW;                         //      Toggle Switch[17:0]

//////////////////////    7-SEG Dispaly   //////////////////////
output  [6:0]   HEX0;                       //      Seven Segment Digit 0
output  [6:0]   HEX1;                       //      Seven Segment Digit 1
output  [6:0]   HEX2;                       //      Seven Segment Digit 2
output  [6:0]   HEX3;                       //      Seven Segment Digit 3
output  [6:0]   HEX4;                       //      Seven Segment Digit 4
output  [6:0]   HEX5;                       //      Seven Segment Digit 5
output  [6:0]   HEX6;                       //      Seven Segment Digit 6
output  [6:0]   HEX7;                       //      Seven Segment Digit 7

////////////////////////////    LED         ////////////////////////////
output  [8:0]       LEDG;                   //      LED Green[8:0]
output  [17:0]  LEDR;                       //      LED Red[17:0]

////////////////////////////    UART ////////////////////////////
output          UART_TXD;                   //      UART Transmitter
input               UART_RXD;               //      UART Receiver

////////////////////// LCD Module 16X2  ////////////////////////////
inout   [7:0]       LCD_DATA;               //      LCD Data bus 8 bits
output          LCD_ON;                     //      LCD Power ON/OFF
output          LCD_BLON;//     LCD Back Light ON/OFF
output          LCD_RW;//   LCD Read/Write Select, 0 = Write, 1 = Read
output          LCD_EN;     //      LCD Enable
output          LCD_RS;//   LCD Command/Data Select, 0 = Command, 1 = Data

//////////////////////    GPIO    ////////////////////////////////
inout   [35:0]  GPIO_0;             //      GPIO Connection 0
inout   [35:0]  GPIO_1;             //      GPIO Connection 1

`timescale 1ns/1ps
//
reg [7:0] cont;
reg gate;
reg gate1;
reg [15:0] compara;
reg [15:0] triangular;
////////////////////////////////////////////////////////////
//                vbase = 100;
//                ibase = 50;
//                indutor = 50uH
//                capacitor = 330uF
```

```
//                  resistor = 2.5 ohms
//                  Vin = 10V
//      indutf = indut/(vbase/ibase);
//      capf = cap*(vbase/ibase);
////////////////////////////////////////////////////////////////

parameter Q = 31;
parameter NQ = 2147483647; //2^31-1
parameter num_bits=2*(Q+1);
parameter lim_sup=num_bits-1;

reg signed [lim_sup:0] vinf;

reg signed [lim_sup:0] ida;
reg signed [lim_sup:0] vda;

//Fios para os modulos
wire [lim_sup:0] vc_boost;
wire [lim_sup:0] il_boost;
wire [lim_sup:0] vc_buck;
wire [lim_sup:0] il_buck;
wire [lim_sup:0] vc_bb;
wire [lim_sup:0] il_bb;

wire reset;
assign reset=SW[1];
assign LEDR[1]=SW[1];

//Instanciacao dos modulos
conv_cc_cc_boost     convBst(.clk_200M(fio_c0),
                                                    .gate(gate),
                                                    .reset(reset),
                                                    .vc(vc_boost),
                                                    .il(il_boost)
                                                    );
//conv_cc_cc_buck    convBck(.clk_200M(fio_c0),
//                                                  .gate(gate),
//                                                  .reset(reset),
//                                                  .vc(vc_buck),
//                                                  .il(il_buck)
//                                                  );
//conv_cc_cc_buckboost convBB(.clk_200M(fio_c0),
//                                                  .gate(gate),
//                                                  .reset(reset),
//                                                  .vc(vc_bb),
//                                                  .il(il_bb)
//                                                  );

always @ (posedge fio_c0)
begin
```

```
        compara = 16'd2500;
        triangular = triangular + 16'd1;
        if(triangular>16'd9999) triangular = 16'd0;

        if(compara>triangular) gate=1;
        else gate =0;

end

always @ (posedge fio_c0)
begin

        cont = cont + 8'd1;
        if(cont>8'd19)
        begin
                gate1 = gate1^1'b1;
                case(SW[17:16])
                        2'b00   :       begin   //conversor boost
                                                vda=vc_boost;
                                                ida=il_boost;
                                        end

                        2'b01   :       begin   //conversor buck
                                                vda=vc_buck;
                                                ida=il_buck;
                                        end

                        2'b10   :       begin   //conversor buck-boost
                                                vda=vc_bb;
                                                ida=il_bb;
                                        end

                        2'b11   :       begin   //reservado
                                        end
                endcase

                cont=0;

        end//endifcont>19
end//endalways@posedgefioc0


wire fio_c0;
//Gera Clock 200MHz em fio_c0
PLL PLL1(     .areset(SW[0]),
                        .inclk0(CLOCK_50),
                        .c0(fio_c0),
                        .locked(LEDG[8])
                );
```

```
//Fios para comunicação SPI
wire start;
wire new_data;
wire fio_busy;
wire fio_mosi;
wire fio_miso;
wire fio_sck;
wire fio_cs;
wire [3:0] index_dado;


//Vetor de comando+dados enviado por spi ao DA
reg [23:0] data_spi [7:0];
always @(posedge fio_c0)
begin
	data_spi[0] = {8'h00,vda[Q:Q-15]+16'h8000};
	data_spi[1] = {8'h21,ida[Q:Q-15]+16'h8000};
	data_spi[2] = {8'h02,16'h0000};
	data_spi[3] = {8'h03,16'h0000};
	data_spi[4] = {8'h04,16'h0000};
	data_spi[5] = {8'h05,16'h0000};
	data_spi[6] = {8'h06,16'h0000};
	data_spi[7] = {8'h07,16'h8000};

end
//Comunicação com Conv. D/A
//Módulo de interface
interface_DA_LTC2600CGN DA1(
			.clk_in(fio_c0),
			.start(start),
			.new_data(new_data),
			.fio_busy(fio_busy),
			.fio_mosi(fio_mosi),
			.fio_sck(fio_sck),
			.fio_cs(fio_cs),
			.index_dado(index_dado)
	);

//Módulo de comunicação SPI
spi spi1(
	.sck_in(fio_c0),
	.miso(fio_miso),
	.dado(data_spi[index_dado]),
	.new_data(new_data),
	.start(start),
	.sck(fio_sck),
	.mosi(fio_mosi),
	.cs(fio_cs),
	.busy(fio_busy)
	);
```

```
//Pinos de saida relacionados ao cabo flat
assign GPIO_0[11] = start;
assign GPIO_0[13] = new_data;
assign GPIO_0[15] = fio_busy;

assign GPIO_0[17] = fio_mosi;
assign GPIO_0[19] = fio_sck;
assign GPIO_0[21] = fio_cs;

assign GPIO_0[12] = fio_mosi;
assign GPIO_0[16] = fio_sck;
assign GPIO_0[24] = fio_cs;

//Fios auxiliares para displays de 7 segmentos
wire [6:0]fHEX0;
wire [6:0]fHEX1;
wire [6:0]fHEX2;
wire [6:0]fHEX4;
wire [6:0]fHEX5;

//Módulo que exibe conversor atual simulado
exibidor_estado exib1(SW[17:0],
                                                    fHEX0,
                                                    fHEX1,
                                                    fHEX2,
                                                    fHEX4,
                                                    fHEX5);

assign  HEX0        =       fHEX0;//codigo do conv.
assign  HEX1        =       fHEX1;//codigo do conv.
assign  HEX2        =       fHEX2;//codigo do conv.
assign  HEX3        =       7'h7F;//apagado
assign  HEX4        =       fHEX4;//Liga e desliga
assign  HEX5        =       fHEX5;//Liga e desliga
assign  HEX6        =       7'h7F;//apagado
assign  HEX7        =       7'h7F;//apagado

//desliga LCD
assign  LCD_ON              =       1'b0;
assign  LCD_BLON    =       1'b0;



//        All inout port turn to tri-state
assign  LCD_DATA    =       8'hzz;

assign  GPIO_1[0] = gate1;
assign  GPIO_1[1] = gate;
```

```
assign LEDR[17:15] = SW[17:15];
assign LEDG[7:0] = 8'b0;
assign LEDR[14:2]= 13'h0;
assign LEDR[0]= 0;
endmodule
```