

UNIVERSIDADE DO ESTADO DE SANTA CATARINA
CCT – CENTRO DE CIÊNCIAS TECNOLÓGICAS
DEE- DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FILIPE KANEAKI IJUIM

CONFIGURAÇÃO DE CONTROLADOR DIGITAL DE SINAIS APLICADO A
CONVERSORES CHAVEADOS UTILIZANDO A FERRAMENTA
PSIM/SIMCODER

JOINVILLE – SC

2014

FILIPPE KANEAKI IJUIM

**CONFIGURAÇÃO DE CONTROLADOR DIGITAL DE SINAIS APLICADO A
CONVERSORES CHAVEADOS UTILIZANDO A FERRAMENTA
PSIM/SIMCODER**

Trabalho de Conclusão apresentado ao curso de Engenharia Elétrica do Centro de Ciências Tecnológicas, da Universidade do Estado de Santa Catarina, como requisito parcial para a obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Dr. Joselito Anastácio Heerdt

JOINVILLE – SC

2014

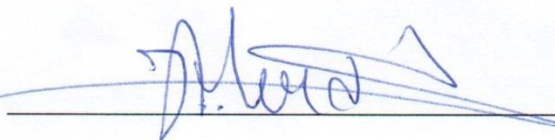
FILIPPE KANEAKI IJUIM

**CONFIGURAÇÃO DE CONTROLADOR DIGITAL DE SINAIS APLICADO A
CONVERSORES CHAVEADOS UTILIZANDO A FERRAMENTA
PSIM/SIMCODER**

Trabalho de Conclusão de Curso, Departamento de Engenharia Elétrica – DEE, Centro de Ciências Tecnológicas – CCT, Universidade do Estado de Santa Catarina – UDESC, Bacharel.

Banca Examinadora

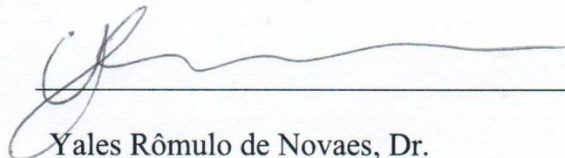
Orientador:



Joselito Anastácio Heerd, Dr.

Universidade do Estado de Santa Catarina – UDESC

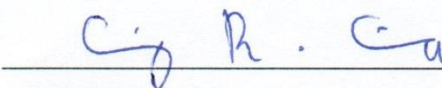
Membro:



Yales Rômulo de Novaes, Dr.

Universidade do Estado de Santa Catarina - UDESC

Membro:



Luiz Ricardo Lima, M.Eng.

Supplier IND. e COM. de Eletroeletrônicos LTDA.

JOINVILLE, SC

DEZEMBRO 2014

RESUMO

IJUIM, Filipe K. **Configuração de Controlador Digital de Sinais Aplicado a Conversores Chaveados Utilizando a Ferramenta PSIM/SimCoder**. 2014. TCC (Bacharelado em Engenharia Elétrica – Área: Eletrônica e Microprocessadores) – Universidade do Estado de Santa Catarina, Joinville, 2014.

Este trabalho de conclusão de curso apresenta o estudo de elementos do *software* de simulação PSIM[®], utilizados para geração de códigos em linguagem C. Os blocos do *SimCoder* podem ser configurados e utilizados em simulações de conversores chaveados. O código gerado pode ser importado para o programa *Code Composer Studio* v5.1, sem que sejam feitas modificações ou adequações no código. O *SimCoder* apresenta estes blocos para implementação de controle digital para um Controlador Digital de Sinais (DSC) da *Texas Instruments*, o TMS320F28335, um controlador de ponto flutuante. Utilizando os blocos do *SimCoder* podem ser configuradas as saídas de PWM, as entradas analógicas, as entradas e saídas digitais, entre outras opções encontradas no DSC. Após o estudo de conversores chaveados, são realizadas simulações com os blocos mencionados, gerados os códigos e realizados testes com o DSC e com o NPCm, um conversor bidirecional de três níveis. Por fim são apresentados os resultados obtidos experimentalmente, apresentando as formas de ondas do DSC e do conversor utilizado, verificando o funcionamento dos códigos gerados.

Palavras-chave: PSIM. Controlador Digital de Sinais. Conversores Chaveados. Controle Digital.

LISTA DE FIGURAS

Figura 1 - Topologia básica do Conversor Buck.....	18
Figura 2 - Primeira etapa de funcionamento do conversor Buck.	18
Figura 3 - Segunda etapa de operação do conversor Buck.....	19
Figura 4 - Formas de onda do conversor Buck: a) Tensão no diodo; b) Corrente no Indutor; c) Corrente na chave; d) Corrente no diodo.....	19
Figura 5 - Tensão de saída do conversor Buck.....	20
Figura 6 - Modelo simplificado do conversor Buck.....	21
Figura 7 - Comparação entre as tensões do conversor comutado e a resposta do modelo contínuo obtido.....	23
Figura 8 - Inversor Meia-Ponte.	24
Figura 9 - Inversor Ponte Completa	25
Figura 10 – Tensão de saída do Inversor Ponte Completa em três níveis e sinais de referência senoidais.	26
Figura 11 - Inversor Trifásico.....	27
Figura 12 - Sinais de referência e portadora para comando de um inversor trifásico.	28
Figura 13 - Inversor NPC trifásico.	29
Figura 14 - Sinais de referência e portadoras do NPC.	30
Figura 15 - Controle de Simulação do PSIM: (a) Símbolo e (b) Janela de parâmetros	31
Figura 16 - Circuito Modulador PWM.....	32
Figura 17 - Formas de onda de uma modulação PWM com referência constante.	33
Figura 18 - Formas de onda de uma modulação PWM com referência senoidal.	33
Figura 19– Bloco de PWM Único.	34
Figura 20– Configurações do bloco de PWM Único.	35
Figura 21 – Bloco PWM Monofásico.	36

Figura 22 – Configurações do gerador de PWM Monofásico.....	37
Figura 23 - Bloco de PWM Monofásico com mudança de fase.....	38
Figura 24 - Configurações do bloco de PWM Monofásico com mudança de fase.	39
Figura 25– Bloco de PWM de Duas Fases.	40
Figura 26 – Configurações do bloco de PWM de duas fases.	41
Figura 27 – Modos de operação do bloco de PWM de Duas Fases: (a)-(f) correspondem aos modos de I a VI.	42
Figura 28 – Bloco de PWM Trifásico.	44
Figura 29– Configurações do modulador de PWM Trifásico.	45
Figura 30 – Blocos (a) <i>Start</i> PWM e (b) <i>Stop</i> PWM.....	46
Figura 31 – Configurações dos blocos (a) <i>Start</i> e (b) <i>Stop</i> PWM.	46
Figura 32 – Bloco <i>Trip-Zone</i>	47
Figura 33 – Configurações do bloco <i>Trip-Zone</i>	47
Figura 34 – Bloco <i>Trip-Zone State</i>	48
Figura 35 – Configurações do bloco <i>Trip-Zone State</i>	48
Figura 36 – Bloco Conversor A/D.....	49
Figura 37 – Configurações do bloco Conversor A/D.	50
Figura 38 – Bloco de Entradas Digitais.....	50
Figura 39 – Configurações do bloco de Entradas Digitais.	51
Figura 40 – Bloco de Saídas Digitais.	51
Figura 41 – Configurações do bloco de Saídas Digitais.....	52
Figura 42 – Bloco de Captura.....	52
Figura 43 – Configurações do bloco de Captura.	53
Figura 44 – Bloco de Estado de Captura.	53
Figura 45 – Configurações do bloco de captura de estado.	54

Figura 46 – Bloco do <i>Encoder</i>	54
Figura 47 – Configurações do bloco <i>Encoder</i>	55
Figura 48 – Bloco do <i>Encoder State</i>	56
Figura 49 – Configurações bloco do <i>Encoder State</i>	56
Figura 50 – Bloco do Contador <i>Up/Down</i>	57
Figura 51 – Configurações do bloco Contador <i>Up/Down</i>	57
Figura 52 – Bloco de <i>DSP Clock</i>	58
Figura 53 – Configurações do bloco de <i>DSP Clock</i>	58
Figura 54 – Bloco de Configuração de Hardware.	59
Figura 55 – Configurações do bloco de Configuração de Hardware.	59
Figura 56 - Símbolo para elementos que podem ser utilizados para geração de código.	60
Figura 57 – Blocos sensores de: (a) tensão e (b) corrente.	60
Figura 58 - Controlador <i>On-Off</i>	61
Figura 59 - Sinal senoidal utilizado para as simulações.	61
Figura 60 - Sinal senoidal obtido utilizando uma onda dente-de-serra e uma função seno.	62
Figura 61 – Circuito de simulação para testes de alguns elementos do <i>SimCoder</i>	63
Figura 62 - Parâmetros configurados do bloco de PWM utilizado para teste.	64
Figura 63 – Kit de experimentação do TMS320F28335 da <i>Texas Instruments</i>	68
Figura 64 - <i>Code Composer Studio</i> versão 5.1.....	69
Figura 65 - Janela para importar um projeto do CCS versão 3.3.	70
Figura 66 – Visualização do código gerado através do CCSv5.1	71
Figura 67 – Simulação e configurações do DSC para o conversor Buck simulado em malha aberta.	72
Figura 68 - Janela de configurações do conversor A/D utilizado na simulação do conversor Buck em malha aberta.	73
Figura 69 - Configurações do bloco de PWM utilizado para o conversor Buck.....	74

Figura 70 - Formas de ondas de saída do conversor Buck em malha aberta.....	74
Figura 71 - Conversor Buck em malha fechada com controlador PI.	77
Figura 72 - Formas de onda de saída do conversor Buck em malha fechada com controlador PI.....	77
Figura 73 - Ferramenta <i>s2z Converter</i> do PSIM®.	78
Figura 74 - Conversor Buck com controlador PI discretizado.	79
Figura 75 - Formas de onda de saída do conversor Buck com controlador discretizado.	80
Figura 76 - Conversor Buck com controlador na forma de função de transferência no domínio z.	81
Figura 77 - Formas de onda de saída do conversor Buck com controlador em função de transferência no.....	82
Figura 78 - Conversor Buck em malha fechada com controlador digital utilizando o bloco de linguagem C.....	83
Figura 79 - Bloco de linguagem C simples com código escrito para o controle digital.....	84
Figura 80 - Formas de onda de saída do conversor Buck com controlador digital utilizando o bloco de linguagem C simples.....	85
Figura 81 - Formas de onda de saídas do conversor Buck em malha fechada para os três métodos de controle digital.....	86
Figura 82 – Circuito da simulação e configurações do DSC para o inversor ponte completa.	87
Figura 83 - Janela de configurações do bloco de PWM utilizado na simulação do inversor ponte completa.....	87
Figura 84 – Circuito de Comando do inversor ponte completa.....	88
Figura 85 - Tensão e corrente de saída do inversor ponte completa	89
Figura 86 - Circuito de potência do inversor trifásico.....	93
Figura 87 – Circuito de comando do inversor trifásico.	93
Figura 88 - Conversor A/D utilizado para o inversor trifásico.....	94
Figura 89 - Tensão de saída das três fases do inversor trifásico.	94

Figura 90 - Circuito da simulação e configurações do DSC para o inversor NPC simulado...	98
Figura 91 - Comando do inversor NPC simulado.	99
Figura 92 – Formas de onda das tensões de saída do NPC.	100
Figura 93 - Configuração de Dispositivo pelo CCSv5.1	104
Figura 94 - Leituras do sinal de PWM e da tensão de entrada do conversor A/D.	105
Figura 95 - Sinais PWM complementares obtidos.	106
Figura 96 - Tempo morto entre os canais de PWM.....	106
Figura 97 - Sinais obtidos a partir de uma referência senoidal.	107
Figura 98 - Sinal de referência externo e sinal PWM gerado.....	107
Figura 99 - Sinais gerados por sinais PWM complementares, após filtragem.	108
Figura 100 - Sinais obtidos do modulador PWM trifásico, após filtragem.	108
Figura 101 - Verificação da defasagem entre duas fases.....	109
Figura 102 - Teste da entrada digital.	110
Figura 103 - Topologia de uma fase do NPCm.	110
Figura 104 - NPCm utilizado para validação dos resultados.....	111
Figura 105 - Circuito de potência do Conversor NPCm.	112
Figura 106 - Geradores de PWM para comando do NPCm.	113
Figura 107 - Configurações de um dos geradores de PWM utilizados para comando do NPCm.	114
Figura 108 - Sinal de Referência e Sinais de PWM gerados.....	114
Figura 109 - Leituras realizadas pelo conversor A/D para o NPCm.	115
Figura 110 - Entradas e Saídas Digitais utilizadas.	116
Figura 111 - Elementos de Proteção do NPCm.....	117
Figura 112 - Tensão e corrente de saída da fase A do conversor NPCm.	118
Figura 113 - Tensão e corrente de saída do conversor NPCm sem perturbações.	118

Figura 114 - Formas de onda entre o <i>Gate</i> e o <i>Source</i> das chaves da fase A.....	125
Figura 115 - Formas de onda nas chaves Sp e S02.	125
Figura 116 - Tempo morto entre as chaves Sp e S02.	126
Figura 117 - Verificação da frequência de chaveamento.	126
Figura 118 - Verificação da frequência do sinal de referência.....	127
Figura 119 - Forma de onda da tensão entre o <i>Drain</i> e o <i>Source</i> do interruptor SpA.....	127
Figura 120 - Tensão entre o <i>Drain</i> e o <i>Source</i> obtido na simulação.	128
Figura 121 - Forma de onda da tensão entre o <i>Drain</i> e o <i>Source</i> do interruptor S02A.....	128
Figura 122 - Verificação da inexistência de tempo morto entre os sinais dos interruptores SpA e S02A.	129
Figura 123 - Tensão do barramento CC e corrente de saída da fase A.	130
Figura 124 - Tensão e corrente de saída da fase A do NPCm.	130
Figura 125 - Correntes das três fases do NPCm.....	131

LISTA DE TABELAS

Tabela 1 - Valores utilizados para simulação do Conversor Buck.....	22
Tabela 2 - Funcionamento do Inversor Ponte Completa.	25

LISTA DE ABREVIATURAS

ADC	<i>Analog to Digital Converter</i>
CA	Corrente Alternada
CC	Corrente Contínua
CCS	Code Composer Studio
DSC	<i>Digital Signal Controller</i>
IPD	<i>In-Phase Disposition</i>
LED	<i>Light Emitting Diode</i>
NPC	<i>Neutral Point Clamped</i>
NPC _m	<i>Modified Neutral Point Clamped</i>
PI	Proporcional-Integral
PLL	<i>Phase Locked Loop</i>
POD	<i>Phase Opposition Disposition</i>
PWM	<i>Pulse Width Modulation</i>
SCI	<i>Serial Communication Interface</i>
SPI	<i>Serial Peripheral Interface</i>
TI	Texas Instruments
ZOH	<i>Zero-Order Hold</i>

SUMÁRIO

1. INTRODUÇÃO.....	15
2. CONVERSORES ELETRÔNICOS BÁSICOS	17
2.1. CONVERSOR BUCK	17
2.2. CONVERSORES CC-CA.....	24
3. COMPONENTES DO SIMCODER	31
3.1. SAÍDAS COM MODULAÇÃO PWM.....	32
3.1.1. PWM Único	34
3.1.2. PWM Monofásico	35
3.1.3. PWM Monofásico com mudança de fase.....	38
3.1.4. PWM de Duas Fases	39
3.1.5. PWM Trifásico.....	43
3.2. START PWM E STOP PWM.....	46
3.3. TRIP-ZONE E TRIP-ZONE STATE.....	47
3.4. CONVERSOR A/D.....	49
3.5. ENTRADAS E SAÍDAS DIGITAIS	50
3.6. CAPTURE E CAPTURE STATE.....	52
3.7. ENCODER E ENCODER STATE	54
3.8. CONTADOR UP/DOWN	56
3.9. DSP CLOCK.....	57
3.10. CONFIGURAÇÃO DE HARDWARE.....	58
3.11. OUTROS COMPONENTES	59
4. CÓDIGO GERADO E O CODE COMPOSER STUDIO	63
4.1. ESTUDO DO CÓDIGO GERADO	63

4.2. O DSC E O CODE COMPOSER STUDIO.....	68
5. APLICAÇÕES DO SIMCODER EM CIRCUITOS DE ELETRÔNICA DE POTÊNCIA.....	72
5.1. CONVERSOR BUCK	72
5.2. INVERSOR PONTE COMPLETA	86
5.3. INVERSOR TRIFÁSICO	92
5.4. INVERSOR NPC	97
6. RESULTADOS EXPERIMENTAIS.....	104
6.1. TESTES COM O DSC	104
6.2. SIMULAÇÃO DO CONVERSOR NPC _m	110
6.3. IMPLEMENTAÇÃO DO CÓDIGO E TESTES	124
7. CONCLUSÕES.....	132
REFERÊNCIAS	134

1. INTRODUÇÃO

Em projeto de conversores, uma parte de extrema importância é o projeto do comando. Para conversores muito complexos são necessários comandos igualmente complexos, ocupando tanto tempo de projeto quanto outras etapas, como pesquisa, projeto do conversor e implementação. São utilizados normalmente para implementação de microcontroladores, Processadores Digitais de Sinais (DSP), Controladores Digitais de Sinais (DSC), FPGA's, entre outros. Entre os fatores que determinam a escolha do controlador, podemos destacar a sua linguagem de programação, o tempo de processamento, frequência máxima de *clock*, tipo de memória, quantidade de entradas e saídas disponíveis, entre outros fatores.

Outra parte importante do projeto do conversor são as simulações. Para tal processo é frequentemente utilizado algum *software* de simulação de circuitos. Por meio de testes nas simulações, pode-se obter uma estimativa de como o conversor irá se comportar, muitas vezes se aproximando bem do comportamento real obtido.

Ambas as etapas são de extrema importância para um projeto, sendo necessário tempo para pesquisa, tempo para aprendizado dos *softwares*, tempo para estudo da linguagem de programação, para domínio dos microcontroladores, e posteriormente, para a realização do projeto em si.

Eventualmente, os componentes deste projeto, isto é, o *software* de simulação e o microcontrolador, possuem ferramentas que auxiliam no processo, possibilitando uma drástica redução no tempo de realização do projeto. Estas ferramentas possibilitam também uma rápida aprendizagem, facilitando a utilização do *software* de simulação e do controlador.

Neste trabalho, será explorado uma das ferramentas, o *software* de simulação PSIM[®], em específico os elementos do *SimCoder*. Esta ferramenta possibilita, após a simulação de circuitos, que seja gerado código em linguagem C, podendo ser implementado em um DSC da *Texas Instruments*, o TMS320F28335. Para tal, serão estudados todos os elementos presentes no *SimCoder*, além de outros elementos do PSIM[®] que podem ser utilizadas em conjunto. Pelo fato de não existirem artigos científicos sobre esta ferramenta, serão utilizados como bibliografias os manuais do usuário, exemplos e a própria ferramenta de ajuda fornecida no *software*.

Após a geração do código, é necessário compreendê-lo, fazendo-se assim um estudo de linguagem C, do DSC utilizado e do *Code Composer Studio*, *software* responsável por compilar o código e passá-lo para o DSC. Como bibliografia, serão utilizados novamente manuais do usuário, notas de aplicação e exemplos.

Para compreender e testar esta ferramenta, serão realizados estudos e simulações de alguns conversores chaveados, possibilitando que sejam utilizados os elementos do *SimCoder* apresentados. Os conversores escolhidos para o estudo são circuitos de potência que possuem interruptores de silício como os transistores MOSFET e IGBT. Estes circuitos tem a função de conversão de energia elétrica, podendo ser de corrente contínua para corrente contínua (CC-CC), ou de corrente contínua para corrente alternada (CC-CA). Para tal estudo, serão utilizados livros de eletrônica de potência, (BARBI, 2006), (MARTINS e BARBI, 2006).

Por fim, serão realizados testes dos códigos gerados em conversores estudados. Serão utilizados modelos didáticos encontrados em laboratório. Têm-se como objetivo verificar a eficácia do processo de simulação, geração do código e implementação do controle no DSC, apresentando os resultados obtidos.

2. CONVERSORES ELETRÔNICOS BÁSICOS

A Eletrônica de Potência pode ser definida como uma ciência aplicada dedicada ao estudo dos conversores estáticos de energia elétrica (BARBI, 2006). Conversores estáticos são sistemas capazes de realizar algum tipo de tratamento da energia elétrica, apresentando elementos passivos, isto é, resistores, indutores e capacitores, e ativos, elementos que realizam algum tipo de interrupção, como diodos, tiristores e transistores.

Entre as principais funções dos conversores estáticos, podemos destacar: Retificador, capaz de transformar corrente alternada em corrente contínua; Conversor Direto de Frequência, capaz de converter tensão alternada em outra de frequência diferente; Inversor, encarregado de transformar corrente contínua em corrente alternada e Conversor CC-CC, capaz de alterar o nível de corrente contínua. Neste trabalho serão estudados exclusivamente os Conversores CC-CC e os Inversores, também conhecidos como Conversores CC-CC. Entre os conversores CC-CC, será estudado o conversor Buck, já entre os conversores CC-CA, serão estudados o conversor de ponte completa e um conversor trifásico.

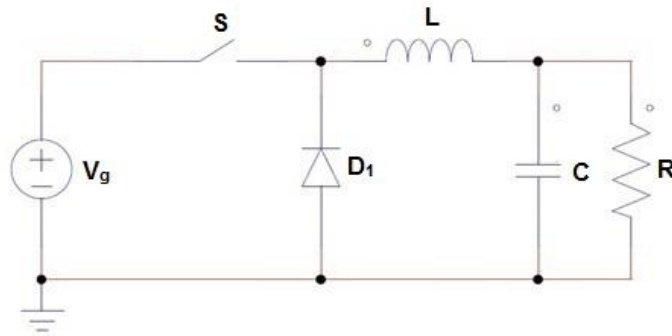
2.1. CONVERSOR BUCK

O conversor Buck é utilizado para se converter um valor médio de tensão contínua de entrada em um valor médio menor de tensão contínua na saída, ou seja, é um conversor abaixador de tensão. Este conversor é utilizado para controlar o fluxo de energia entre dois sistemas de corrente contínua.

Conforme mostrado na Figura 1, o conversor Buck apresenta alguns componentes que definem o seu funcionamento. O interruptor S tem a função de comutar, determinando duas etapas de funcionamento.

Na primeira etapa, em que o interruptor se encontra fechado, o circuito é alimentado pela fonte de tensão V_g . Já na segunda etapa, com o interruptor aberto, outro elemento será responsável por alimentar a carga, o indutor L , que funciona no circuito como uma fonte de corrente, liberando a energia armazenada durante a primeira etapa.

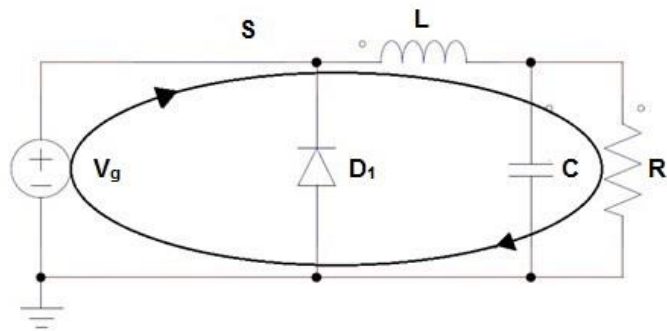
Figura 1 - Topologia básica do Conversor Buck



Fonte: Produção do próprio autor.

Se a corrente do indutor chegar a zero antes do final do período de comutação, considera-se que o conversor está funcionando em condução descontínua, caso contrário, a condução é contínua. Na Figura 2 é possível ver o caminho que a corrente percorre durante a primeira etapa.

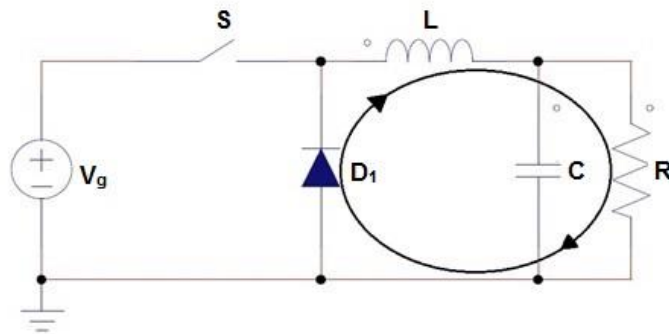
Figura 2 - Primeira etapa de funcionamento do conversor Buck.



Fonte: Produção do próprio autor.

O diodo D_1 exerce duas funções, dependendo da etapa em que o circuito se encontra. Na primeira etapa, com o interruptor fechado, o diodo é encarregado de bloquear qualquer possível corrente contrária à corrente imposta pela fonte. Com o interruptor aberto, ou seja, na segunda etapa, o diodo entra em roda livre, possibilitando que a corrente do indutor L circule pela carga. A Figura 3 representa o funcionamento do conversor Buck durante a segunda etapa.

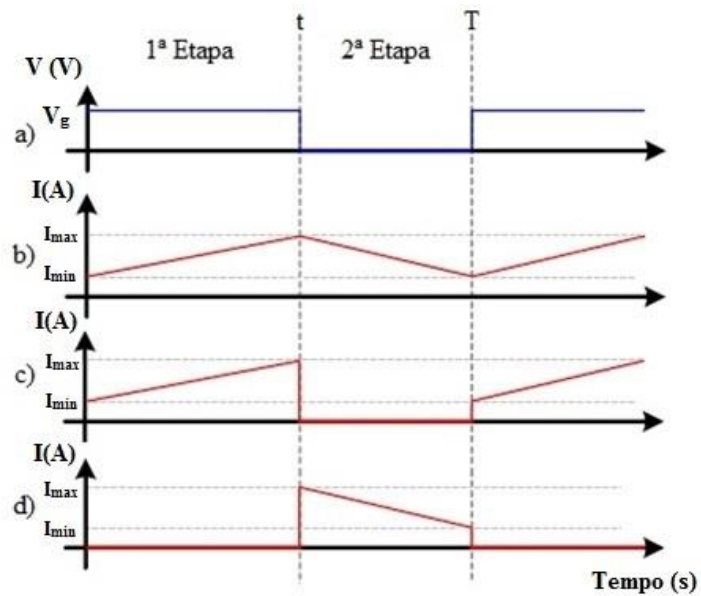
Figura 3 - Segunda etapa de operação do conversor Buck.



Fonte: Produção do próprio autor.

Operando em modo de condução contínua, o conversor Buck apresenta corrente eficaz menor que no modo de condução descontínua, tornando a condução contínua uma melhor opção. As principais formas de onda do funcionamento do conversor Buck em condução contínua podem ser observadas na Figura 4.

Figura 4 - Formas de onda do conversor Buck: a) Tensão no diodo; b) Corrente no Indutor; c) Corrente na chave; d) Corrente no diodo.

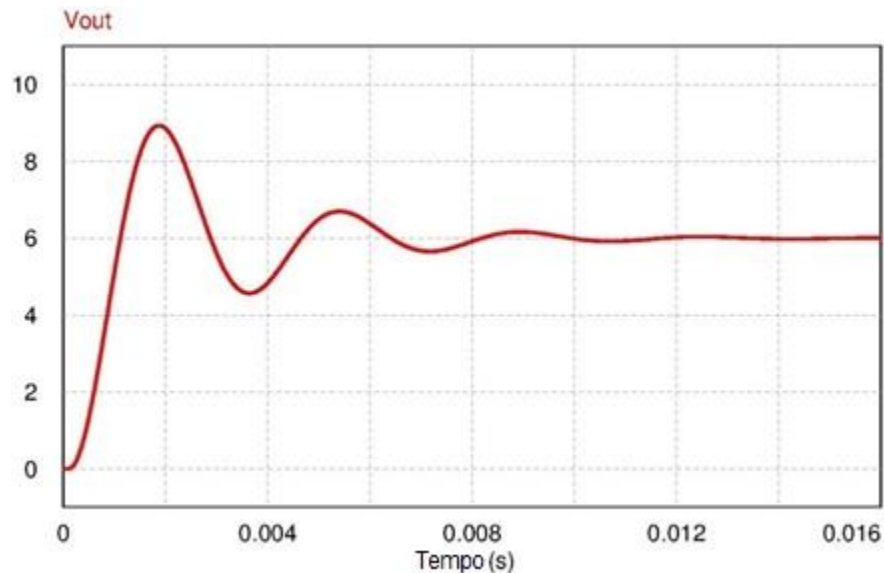


Fonte: Produção do próprio autor.

Observa-se que as formas de ondas apresentadas são encontradas quando o conversor encontra-se em regime permanente. Nas formas de onda apresentadas, observa-se que T é o período de chaveamento e t é o tempo em que o interruptor permanece fechado. A razão entre os dois tempos mencionados é chamado de razão cíclica, ou ciclo de trabalho. Pode-se observar também que as correntes apresentam um valor máximo e mínimo. Para condução descontinua, o valor mínimo da corrente seria igual à zero.

A partir de simulações no *software* PSIM, pode-se observar uma característica semelhante a um sistema de segunda ordem, conforme mostrado na Figura 5.

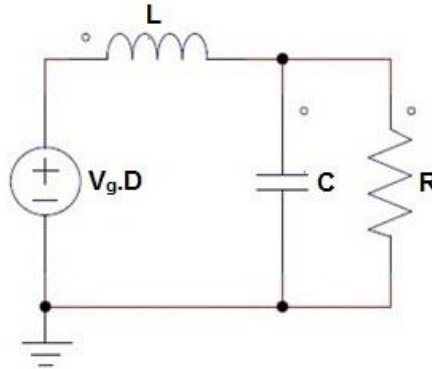
Figura 5 - Tensão de saída do conversor Buck.



Fonte: Produção do próprio autor.

Devido a esta característica observada do conversor Buck, é comum encontrarem-se projetos de controle, tendo como objetivo corrigir ou cancelar o pico de tensão observado ou reduzir o tempo de resposta. Para projetar o controlador é necessário obter o modelo matemático do conversor. Para isso, considera-se a simplificação do conversor Buck, observada na Figura 6.

Figura 6 - Modelo simplificado do conversor Buck.



Fonte: Produção do próprio autor.

A fonte de tensão do circuito corresponde à tensão média do diodo, onde V_g é a tensão de entrada e D é a razão cíclica. A partir do circuito apresentado, é possível obter as seguintes equações:

$$-V_g \cdot D + L \frac{di_L(t)}{dt} + v(t) = 0 \quad (1)$$

$$i_L(t) = i_C(t) + \frac{v(t)}{R} \quad (2)$$

$$i_C(t) = C \frac{dv(t)}{dt} \quad (3)$$

Onde i_L e i_C são, respectivamente, as correntes no indutor e no capacitor e $v(t)$ é a tensão de saída do circuito, ou seja, a tensão na carga. Aplicando-se a equação (2) na equação (1), obtemos:

$$-V_g \cdot D + L \frac{di_C(t)}{dt} + \frac{L}{R} \frac{dv(t)}{dt} + v(t) = 0 \quad (4)$$

Substituindo-se a equação (3) na equação (4), obtém-se a equação a seguir.

$$LC \frac{d^2v(t)}{dt^2} + \frac{L}{R} \frac{dv(t)}{dt} + v(t) = V_g \cdot D \quad (5)$$

Considerando o conceito de valor médio instantâneo e que o valor da razão cíclica também varia com o tempo, pode-se alterar a equação (6) e em seguida aplicar a Transformada de Laplace (7).

$$LC \frac{d^2 \hat{v}(t)}{dt^2} + \frac{L}{R} \frac{d\hat{v}(t)}{dt} + \hat{v}(t) = V_g \cdot D(t) \quad (6)$$

$$LC s^2 V(s) + \frac{L}{R} s V(s) + V(s) = V_g \cdot D(s) \quad (7)$$

Considerando-se $V(s)$ como o sinal de saída e $d(s)$ como sinal de entrada do conversor, encontramos a função de transferência do conversor Buck.

$$\frac{V(s)}{D(s)} = \frac{V_g}{LC s^2 + \frac{L}{R} s + 1} \quad (8)$$

Através do componente *s-domain Transfer Function*, do *software* PSIM, pode-se simular o modelo obtido, comparando com a forma de onda do conversor. Para esta simulação, foram utilizados os seguintes valores:

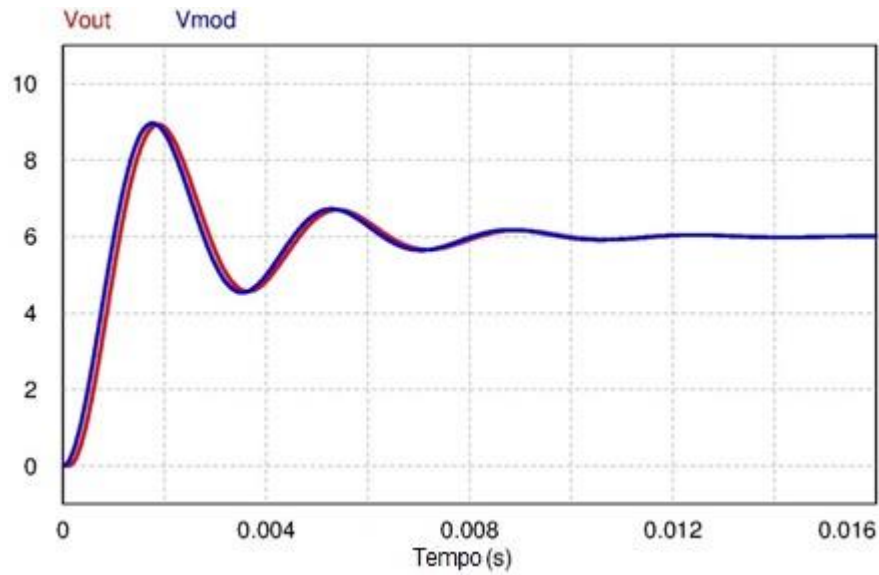
Tabela 1 - Valores utilizados para simulação do Conversor Buck.

V_g	12V
L	600μH
C	500μF
R	2,5Ω

Fonte: Produção do próprio autor.

O resultado obtido é mostrado na Figura 7. Observa-se que o modelo apresenta um comportamento muito próximo do comportamento do conversor.

Figura 7 - Comparação entre as tensões do conversor comutado e a resposta do modelo contínuo obtido.



Fonte: Produção do próprio autor.

Da mesma forma como é obtido o modelo de tensão do conversor Buck, pode-se encontrar o modelo de corrente, apresentado a seguir.

$$\frac{I_l(s)}{D(s)} = \frac{Vg}{R} \left[\frac{RCs + 1}{LCs^2 + \frac{L}{R}s + 1} \right] \quad (9)$$

Onde I_l é a corrente no indutor.

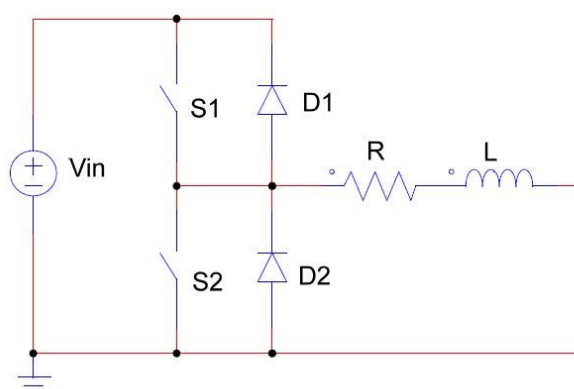
Outros conversores que não serão apresentados possuem comando similar ao conversor Buck. Outros conversores CC-CC, como o conversor Boost, ou o Buck-Boost apresentam, semelhantemente ao Buck, apenas um interruptor, sendo necessário, portanto, apenas um sinal de PWM para o comando. Pode-se obter os modelos destes conversores do mesmo modo como foi obtido para o conversor Buck, podendo-se realizar simulações com o conversor e projetar o controlador.

2.2. CONVERSORES CC-CA

Os inversores são circuitos estáticos (isto é, não tem partes móveis) que convertem potência CC em potência CA com frequência e tensão ou corrente de saída desejada (AHMED, 2000). Portanto, este conversor tem como entrada tensão contínua e como saída tensão alternada. Para tornar a tensão de saída uma onda senoidal, utiliza-se o método de modulação por largura de pulso (PWM), com um sinal senoidal como sinal de referência.

Existem diversos tipos de inversores, diferenciando-se entre si em números de interruptores utilizados, tipos de comando, números de fases, entre outros. O tipo mais básico é o inversor meia-ponte, mostrado na Figura 8.

Figura 8 - Inversor Meia-Ponte.



Fonte: Produção do próprio autor.

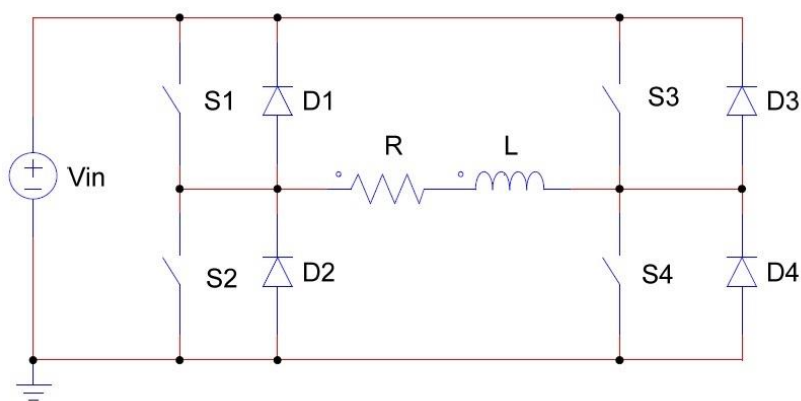
Este conversor apresenta dois interruptores, que possuem comandos complementares, isto é, quando um interruptor está ativado, o outro deve estar desativado, para que a fonte de tensão não fique em curto-circuito. Para garantir que os interruptores não sejam ativados simultaneamente, o comando é feito com um pequeno tempo de diferença entre o fim do ciclo de um interruptor e o início do ciclo do outro. Este tempo é chamado de tempo morto, ou *dead time*.

No circuito, quando o interruptor S_1 está conduzindo, a tensão na carga RL é a tensão da fonte de entrada. Quando o interruptor S_2 entra em condução, a tensão de saída é zero,

obtendo-se assim uma onda quadrada. Observa-se que em ambos os circuitos, os diodos D1 e D2 tem a função de conduzir correntes contrárias. Estes diodos são denominados diodos de retorno.

Outro tipo de inversor é o de ponte completa. Este conversor, conforme mostrado na Figura 9, possui quatro interruptores, distribuídos em dois braços. Com esta configuração, podem-se obter, dependendo do tipo de comando, diferentes formas de ondas na saída.

Figura 9 - Inversor Ponte Completa



Fonte: Produção do próprio autor.

A tabela abaixo apresenta as combinações possíveis, considerando que os interruptores no mesmo braço, isto é, S1 e S2 ou S3 e S4, não podem ser acionados simultaneamente.

Tabela 2 - Funcionamento do Inversor Ponte Completa.

Estado	S1	S2	S3	S4	Tensão de saída
1	Ligado	Desligado	Desligado	Ligado	V_{in}
2	Ligado	Desligado	Ligado	Desligado	0
3	Desligado	Ligado	Ligado	Desligado	$-V_{in}$
4	Desligado	Ligado	Desligado	Ligado	0

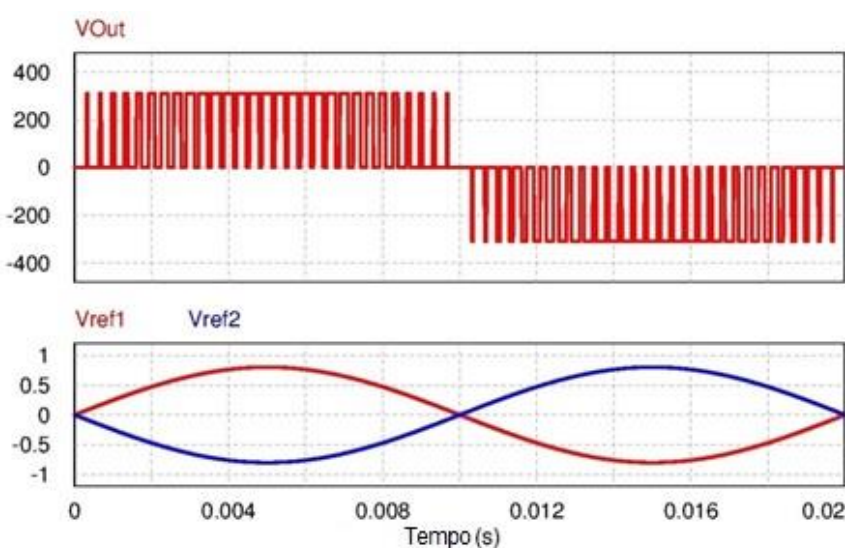
Fonte: Produção do próprio autor, baseado em Ahmed, Eletrônica de Potência. (AHMED, 2000)

Uma opção de comando para o inversor ponte completa, é utilizar os estados 2 e 3 apresentados na tabela. Utilizando apenas um sinal de referência, comandam-se simultaneamente os interruptores S1 e S4 com um sinal de PWM, e com o complemento deste sinal comandam-se simultaneamente os interruptores S2 e S3. Desta forma, obtém-se uma onda quadrada, de tensão máxima V_{in} e tensão mínima $-V_{in}$.

Porém, pode-se comandar este inversor utilizando os quatro estados apresentados. Iniciando pelo estado 2, com os interruptores S1 e S4 conduzindo e com S2 e S3 não conduzindo. Em seguida, liga-se o interruptor S3 e desliga-se S4, mantendo-se os interruptores S1 e S2 como estavam na etapa anterior, conforme mostra o estado 1. Mantém-se então os interruptores S3 e S4, invertendo os estados dos interruptores S1 e S2, referente ao estado 3. Por fim, no estado 4, mantém-se os interruptores S1 e S2, desativa-se S3 e liga-se o interruptor S4.

Estas etapas produzem na carga uma onda quadrada, que apresenta três níveis diferentes. A modulação para este tipo de comando é feita com duas senoidais de referência, defasadas entre si em 180° . O primeiro sinal de referência gera o sinal de PWM para os interruptores S1 e S2, sendo complementares entre si, sendo que o segundo sinal de referência gera o sinal PWM para os interruptores S3 e S4. Na Figura 10, é mostrado o sinal de saída do inversor e seus sinais de referência.

Figura 10 – Tensão de saída do Inversor Ponte Completa em três níveis e sinais de referência senoidais.

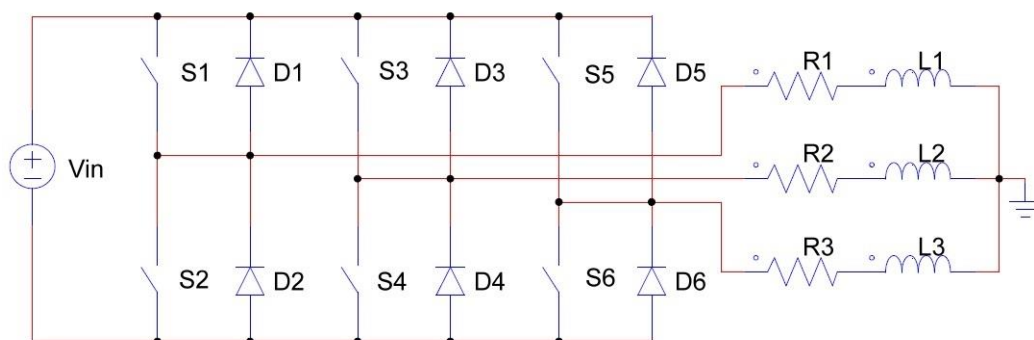


Fonte: Produção do próprio autor.

A partir da saída do inversor, pode-se obter uma onda senoidal filtrando-se os harmônicos indesejáveis. O estudo sobre diferentes tipos de inversores e seu comando tem por objetivo reduzir estas harmônicas, possibilitando assim a utilização de filtros mais simples e reduzindo as perdas no processo. Podem-se obter menos harmônicas aumentando a quantidade de interruptores em um mesmo braço do inversor, tornando também mais complexo o comando deste.

Um inversor trifásico apresenta três braços com dois interruptores e dois diodos de retorno, como mostrado na Figura 11. Utilizando um banco de carga trifásica, obtém-se como saída três formas de onda semelhantes às encontradas no inversor ponte completa, porém defasadas entre si em 120° .

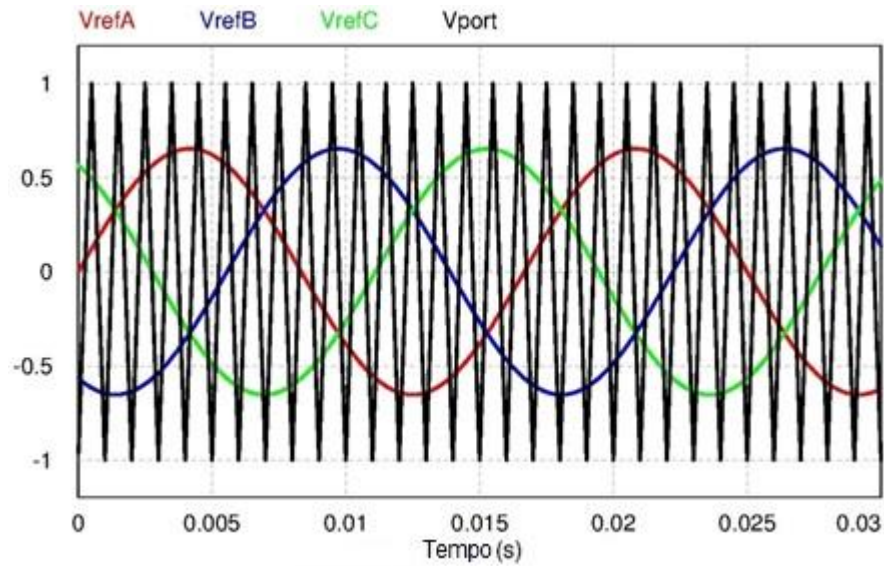
Figura 11 - Inversor Trifásico.



Fonte: Produção do próprio autor.

Para o comando deste inversor, usam-se seis sinais de PWM, um sinal para cada chave. Assim como no inversor de ponte completa, no inversor trifásico os interruptores em um mesmo braço não devem ser acionados simultaneamente. Os sinais de PWM são gerados por três referências senoidais defasadas em 120° entre si, comparando-se com uma mesma portadora triangular, conforme mostrado na Figura 12.

Figura 12 - Sinais de referência e portadora para comando de um inversor trifásico.



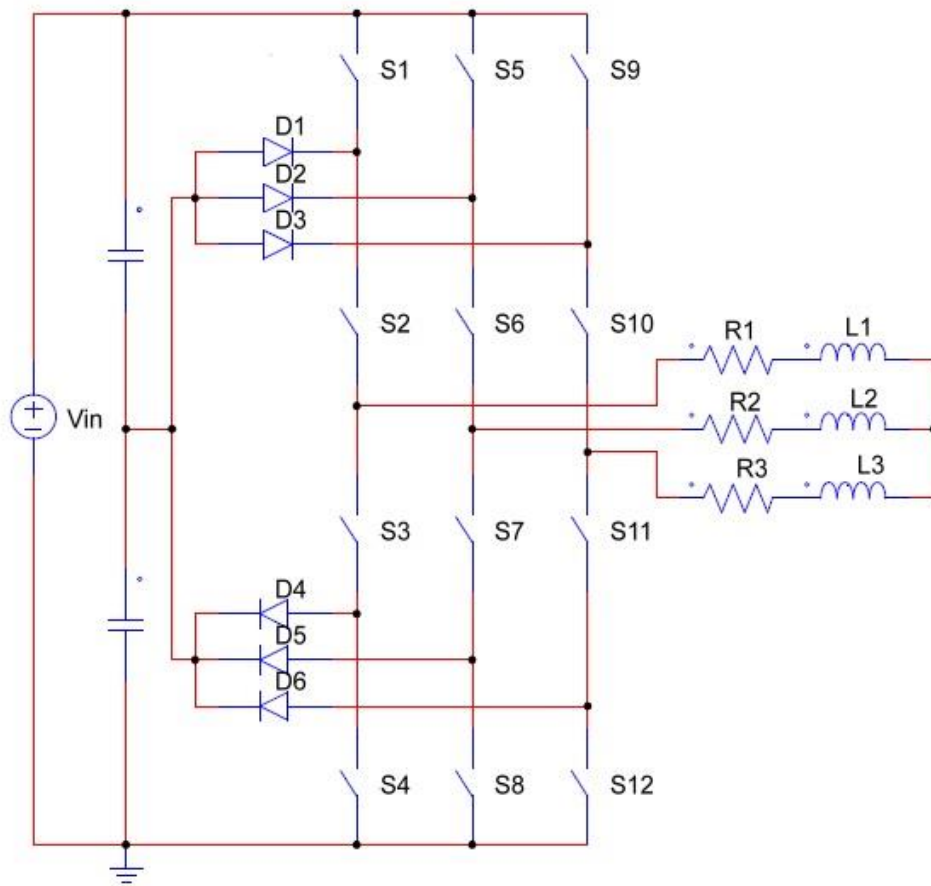
Fonte: Produção do próprio autor.

Outra topologia possível é apresentada na Figura 13. O circuito apresentado é chamado de Inversor NPC (*Neutral Point Clamped*). Este conversor possui três braços, com quatro interruptores em cada braço, apresentando saída para uma carga trifásica.

A razão de se ter mais chaves em cada braço é que através desta configuração pode-se obter mais níveis na saída do conversor, simplificando desta forma o projeto do filtro utilizado, devido à redução do conteúdo harmônico, reduzindo também os esforços nos interruptores.

O comando deste inversor é mais complexo, considerando que são necessários doze canais de PWM para as doze chaves observadas. Para o comando do inversor NPC, são utilizadas duas portadoras triangulares e três sinais de referência senoidais. Primeiramente as chaves são divididas entre as de número ímpar e de número par, vistas na figura acima. Para as chaves de número ímpar, é realizada a comparação das referências senoidais com a portadora triangular positiva, ou seja, a onda que tem valor mínimo zero e valor máximo 1.

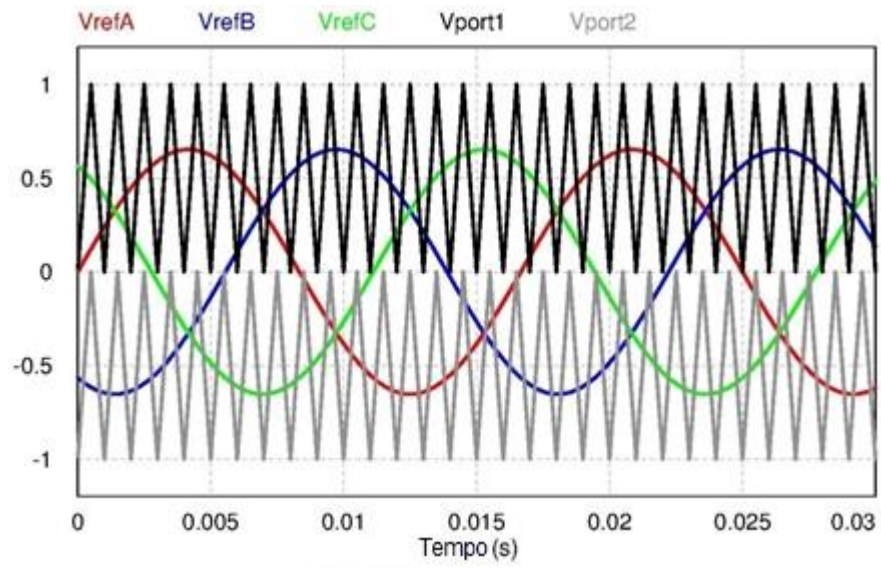
Figura 13 - Inversor NPC trifásico.



Fonte: Produção do próprio autor.

Igualmente ao inversor trifásico simples, para cada braço é utilizado uma referência senoidal, defasados entre si em 120° . Para as chaves S1 e S3, os canais de PWM devem ser complementares, assim como para os interruptores S5 e S7 e para os interruptores S9 e S11. O mesmo é feito para os interruptores pares, porém com a portadora triangular negativa, ou seja, a onda que tem valor máximo zero e valor mínimo -1. Na Figura 14 são mostradas as formas de onda dos sinais de referência e das portadoras da modulação IDP (In-Phase Disposition), utilizada para o comando do conversor NPC.

Figura 14 - Sinais de referência e portadoras do NPC.

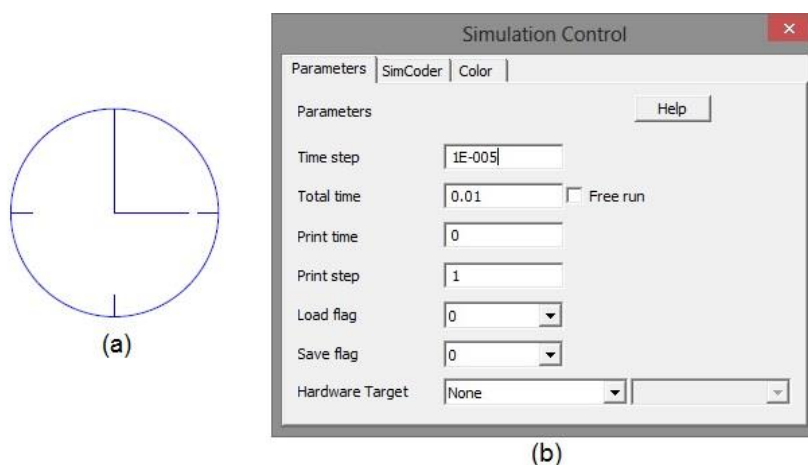


Fonte: Produção do próprio autor.

3. COMPONENTES DO SIMCODER

A ferramenta *SimCoder*, do software PSIM[®] possibilita que, a partir de um circuito de simulação, seja criado um código em linguagem C já pronto para implementação em um controlador digital de sinais (DSC). O controlador em questão é o TMS320F28335, da *Texas Instruments*, que será estudado posteriormente. O *SimCoder* apresenta diversos componentes, que podem ser encontrados no menu em *Elements/SimCoder for CodeGeneration/TI F28335 Target*. É importante que, antes de realizar uma simulação utilizando um dos componentes do *SimCoder*, seja configurado o controle de simulação. Na Figura 15 pode-se observar o símbolo e a janela de configurações do controle de simulação.

Figura 15 - Controle de Simulação do PSIM: (a) Símbolo e (b) Janela de parâmetros



Fonte: PSIM[®].

Na Figura 15b em *Hardware Target* deve-se selecionar a opção TI F28335, e à sua direita *RAM Debug*, para ser utilizado posteriormente no software do DSC, o *Code Composer Studio*. O processo para a geração do código será explicado após a apresentação dos componentes do *SimCoder*.

O DSC F28335 possui oitenta e oito pinos que podem ser configurados como saídas de PWM, entradas e saídas digitais, *encoders*, *trip-zones*, contadores ou *captures*. Ele possui também dezesseis pinos exclusivos para os conversores A/D.

3.1. SAÍDAS COM MODULAÇÃO PWM

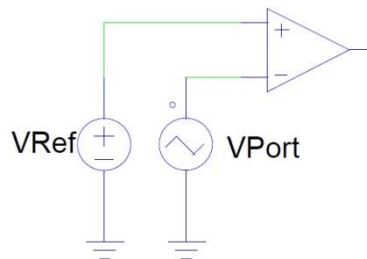
Uma modulação PWM consiste em transformar um nível de tensão de referência em uma onda quadrada de razão cíclica proporcional a esta tensão. A razão cíclica de um sinal PWM é a razão entre o tempo em que a onda se mantém em nível alto e o período da onda quadrada, dado por:

$$D = \frac{t}{T} \quad (10)$$

Onde D é a razão cíclica, ou ciclo de trabalho, t é o tempo em nível lógico alto e T é o período, que é determinado como sendo o inverso da frequência da onda quadrada.

Para se gerar um sinal PWM, é realizada a comparação de um sinal de referência com uma portadora com a forma de uma onda triangular ou dente-de-serra. A escolha da portadora, de sua amplitude e de sua frequência depende da aplicação. A comparação de forma análoga pode ser feita com um amplificador operacional, como visto na Figura 16.

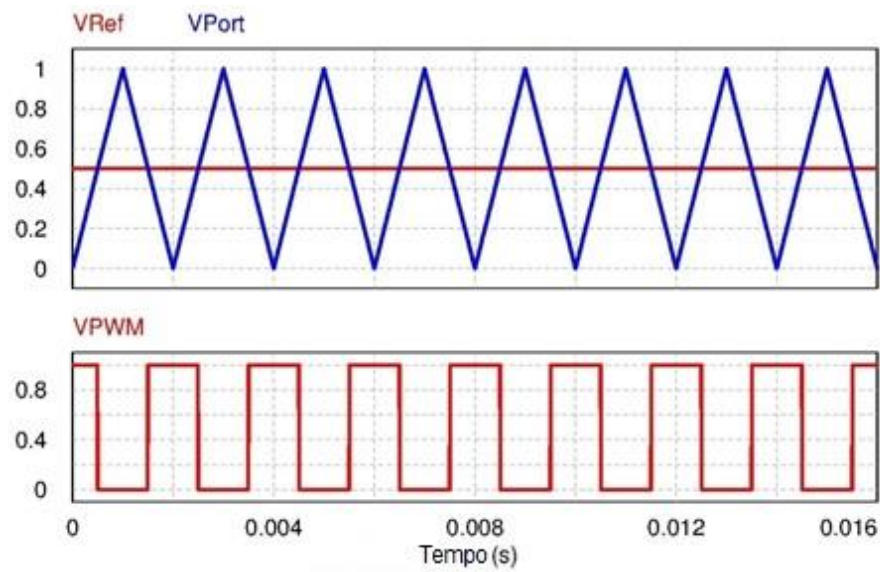
Figura 16 - Circuito Modulador PWM.



Fonte: Produção do próprio autor.

Dependendo da aplicação, o sinal de referência pode ser um sinal constante ou senoidal. Na Figura 17, é mostrado o sinal da portadora triangular (VPort) o sinal de referência constante (VRef) e a forma de onda do sinal PWM gerado (VPWM).

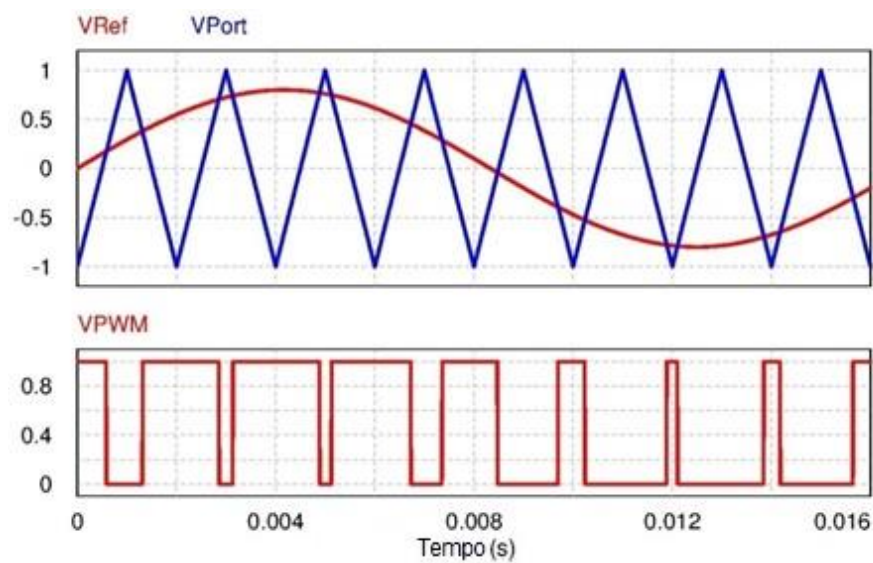
Figura 17 - Formas de onda de uma modulação PWM com referência constante.



Fonte: Produção do próprio autor.

Se sinal de referência for uma onda senoidal, a razão cíclica varia constantemente, na mesma proporção da variação do sinal de entrada. É importante que a frequência da onda portadora seja maior que a frequência do sinal de referência. Na Figura 18, são mostradas as formas de onda da portadora (VPort), do sinal senoidal (VRef), e o sinal PWM resultante (VPWM).

Figura 18 - Formas de onda de uma modulação PWM com referência senoidal.



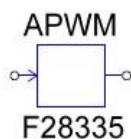
Fonte: Produção do próprio autor.

O *SimCoder* possui alguns blocos de configurações de saídas PWM. Esses blocos apresentados a seguir, realizam a comparação de uma portadora interna com um sinal de referência. Os blocos PWM encontrados no *SimCoder* são: PWM único, PWM de uma fase, de duas fases e trifásico.

3.1.1. PWM Único

O bloco de configuração de PWM Único apresenta apenas uma entrada e uma saída. Como entrada, deve-se ter um sinal, constante ou variável, que esteja entre o valor pico-a-pico configurado. Sua saída será um sinal de onda quadrada de nível baixo igual a 0 e nível alto igual a 1. Na Figura 19 é apresentado o bloco de PWM Único.

Figura 19– Bloco de PWM Único.

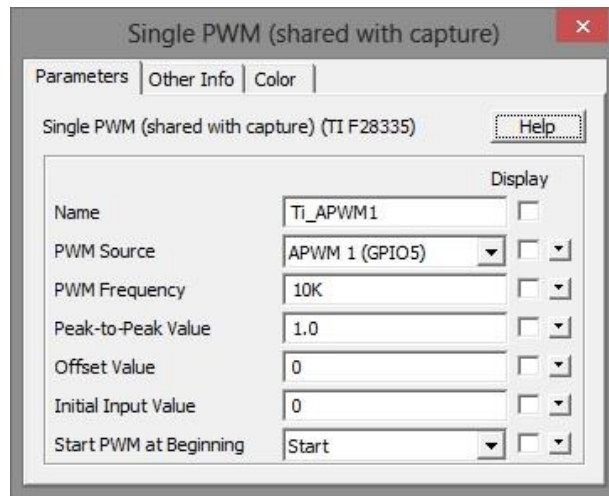


Fonte: Produção do próprio autor.

As configurações necessárias ao bloco de PWM Único, mostradas na Figura 6, são:

- *PWM Source*: Seleção da porta correspondente que será utilizada do DSC. São divididos de APWM1 a APWM6, sendo que cada uma possui a opção de seleção entre duas ou três portas GPIO distintas. Caso seja necessário utilizar mais de um gerador de PWM Único, deve-se configurá-los com diferentes APWM's, portanto pode-se utilizar até seis destes blocos em uma aplicação.
- *PWM Frequency*: Definição da frequência do sinal de saída, em hertz. Ao abrir a janela de configurações, este espaço já estará preenchido com 10 kHz.
- *Peak-to-Peak Value*: Definição do valor pico-a-pico do sinal de entrada.
- *Offset Value*: Definição do valor de offset do sinal de entrada (nível CC).

Figura 20– Configurações do bloco de PWM Único.



Fonte: PSIM®.

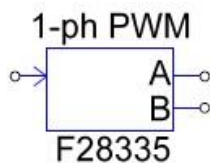
- *Initial Input Value*: Definição do valor inicial da entrada. Este valor interfere apenas nos primeiros ciclos do sinal de saída.
- *Start PWM at Beginning*: Seleção entre “Start”, em que o gerador de PWM começará ativado, ou “Do not Start”, em que o gerador de PWM dependerá de um bloco *Start PWM* para funcionar.

Os pinos que podem ser utilizados para este gerador de PWM são: GPIO1, GPIO3, GPIO5, GPIO7, GPIO9, GPIO11, GPIO24, GPIO25, GPIO26, GPIO27, GPIO34, GPIO37, GPIO48 e GPIO49. Porém, é preciso verificar nas configurações quais destas portas pertencem ao mesmo grupo de APWM's, pois dois blocos de PWM Único não podem ser configuradas para o mesmo grupo.

3.1.2. PWM Monofásico

O gerador de PWM Monofásico também possui uma única entrada, igual ao gerador de PWM Único, porém apresenta duas saídas, como pode ser visto na Figura 21. A saída B consiste em um sinal complementar ao sinal da saída A.

Figura 21 – Bloco PWM Monofásico.



Fonte: Produção do próprio autor.

Este bloco possui diversas opções de configuração que o gerador de PWM Único não possui, possibilitando uma maior variedade de aplicação. A janela de configurações do módulo de PWM Monofásico pode ser observada na Figura 22. Seus parâmetros de configuração são os seguintes:

- *PWM Source*: Seleção das portas correspondentes que serão utilizadas do DSC. São divididos de PWM1 a PWM6, sendo que cada um possui duas portas GPIO. Para o PWM1, as portas são GPIO0 e GPIO1, para o PWM2, as portas são GPIO2 e GPIO3, seguindo este padrão até o PWM6. Observa-se que podem ser utilizados seis geradores monofásicos em uma aplicação.
- *Output Mode*: Definição do modo de saída, entre utilizar as saídas A e B, ou utilizar apenas a saída A ou a saída B.
- *Dead Time*: Definição do tempo-morto entre os sinais de saída, em segundos. Este tempo é necessário em alguns casos para que dois interruptores não sejam comandados simultaneamente. Ao abrir a janela de configurações, este espaço já estará preenchido com 4 μ s.
- *PWM Frequency*: Definição da frequência do bloco de PWM, em hertz. Ao abrir a janela de configurações, este espaço já estará preenchido com 10kHz. Esta frequência escolhida será a frequência de amostragem dos Conversores A/D.
- *PWM Frequency Scaling Factor*: Seleção entre os fatores 1, 2 ou 3. Este fator de escala multiplica a frequência do PWM, determinando a frequência do sinal de saída do PWM. Pode ser utilizado para se utilizar um sinal PWM de frequência diferente da frequência de amostragem.
- *Carrier Wave Type*: Definição do tipo de forma de onda da portadora. Pode-se escolher entre as formas triangular ou dente-de-serra.

- *Trigger ADC*: Deve-se escolher se o bloco PWM irá disparar o Conversor A/D e qual grupo do conversor será acionado, grupo A, grupo B, ou ambos.
- *ADC Trigger Position*: Deve-se optar se será feito o gatilho em um conversor A/D, e se será feito para o grupo A, B ou para os dois grupos do conversor.
- *Use Trip-zones*: Definição de como serão utilizados os *Trip-zones* de 1 a 6, entre desabilitá-los, ou habilitá-los no modo *One-shot* ou *Cycle-by-Cycle*. Estes serão explicados nas configurações dos blocos *Trip-zones*.

Figura 22 – Configurações do gerador de PWM Monofásico.

Parameters		Other Info		Color	
Single-phase PWM generator (TI F28335) [Help]					
Name	TI_PWM2	Display			
PWM Source	PWM 1				
Output Mode	Use PWM A&B				
Dead Time	4us				
PWM Frequency	10K				
PWM Freq. Scaling Factor	1				
Carrier Wave Type	Triangular wave				
Trigger ADC	Do not trigger ADC				
ADC Trigger Position	0				
Use Trip-Zone 1	Disable Trip-Zone 1				
Use Trip-Zone 2	Disable Trip-Zone 2				
Use Trip-Zone 3	Disable Trip-Zone 3				
Use Trip-Zone 4	Disable Trip-Zone 4				
Use Trip-Zone 5	Disable Trip-Zone 5				
Use Trip-Zone 6	Disable Trip-Zone 6				
Trip Action	High impedance				
Peak-to-Peak Value	1				
Offset Value	0				
Initial Input Value	0				
Start PWM at Beginning	Start				

Fonte: PSIM®.

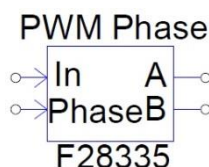
- *Trip Action*: Definição de como o bloco de PWM responderá à ação de um *Trip-zone*, podendo optar entre alta impedância para as duas saídas, para apenas a saída A ou apenas B, ou não tendo nenhuma ação.
- *Peak-to-Peak Value*: Deve-se preencher com o valor pico-a-pico do sinal de entrada.
- *Offset Value*: Deve-se preencher com o valor de offset do sinal de entrada (nível CC).

- *Initial Input Value*: Deve-se preencher com o valor inicial da entrada. Este valor interfere apenas nos primeiros ciclos do sinal de saída.
- *Start PWM at Beginning*: Deve-se selecionar entre “*Start*”, em que o gerador de PWM começará ativado, ou “*Do not Start*”, em que o gerador de PWM dependerá de um bloco *Start PWM* para funcionar.

3.1.3. PWM Monofásico com mudança de fase

Este bloco de PWM é semelhante ao anterior, apresentando uma entrada a mais. A entrada *In*, consiste na entrada do sinal de referência, também encontrado no bloco de PWM Monofásico. A entrada *Phase* é utilizada para defasar o sinal PWM gerado por este bloco em relação a um bloco de PWM Monofásico comum. Este bloco deve ser utilizado juntamente a um bloco de PWM Monofásico, pois usará o sinal PWM deste bloco como referência. Na Figura 23 é observado este bloco.

Figura 23 - Bloco de PWM Monofásico com mudança de fase.



Fonte: Produção do próprio autor.

Para a utilização deste bloco, os canais de PWM são divididos em dois grupos. No primeiro grupo estão os canais PWM1, PWM2 e PWM3, já no segundo grupo encontram-se os canais PWM4, PWM5 e PWM6. O bloco de PWM Monofásico de referência e este bloco devem ser configurados para canais pertencentes ao mesmo grupo, sendo que o bloco de PWM Monofásico com mudança de fase não pode ser configurado para o canal PWM1. Por exemplo, se o bloco de PWM Monofásico foi configurado para o canal PWM2, o bloco com mudança de fase deve ser configurado para o canal PWM3. Se o bloco de PWM Monofásico foi configurado para o canal PWM4, o bloco com mudança de fase pode ser configurado para os canais PWM5 ou PWM6. Se for configurado o PWM1 como referência,

o bloco com mudança de fase pode ser configurado para qualquer outro canal, pois o canal PWM1 pertence aos dois grupos.

Os parâmetros de configurações deste componente, com exceção do canal a ser escolhido, são os mesmos que os parâmetros mostrados para o bloco de PWM Monofásico, conforme visto na Figura 24.

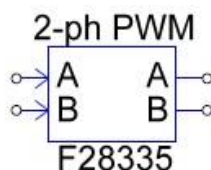
Figura 24 - Configurações do bloco de PWM Monofásico com mudança de fase.

Fonte: PSIM®.

3.1.4. PWM de Duas Fases

O bloco de PWM de Duas Fases, observado na Figura 25, possui duas entradas e duas saídas, porém, diferentemente do gerador monofásico, as saídas não são complementares. Este gerador tem como saídas dois sinais de frequências iguais, gerados por uma mesma portadora. Este gerador não possui a opção de utilizar a saída A ou B, ou seja, sempre as duas saídas estão ativadas. Também não possui a opção de escolher o tempo morto. Este gerador de PWM possui seis modos de utilização, em que se difere em tipo de portadora, triangular ou dente-de-serra, e no modo de comparação com a portadora.

Figura 25– Bloco de PWM de Duas Fases.

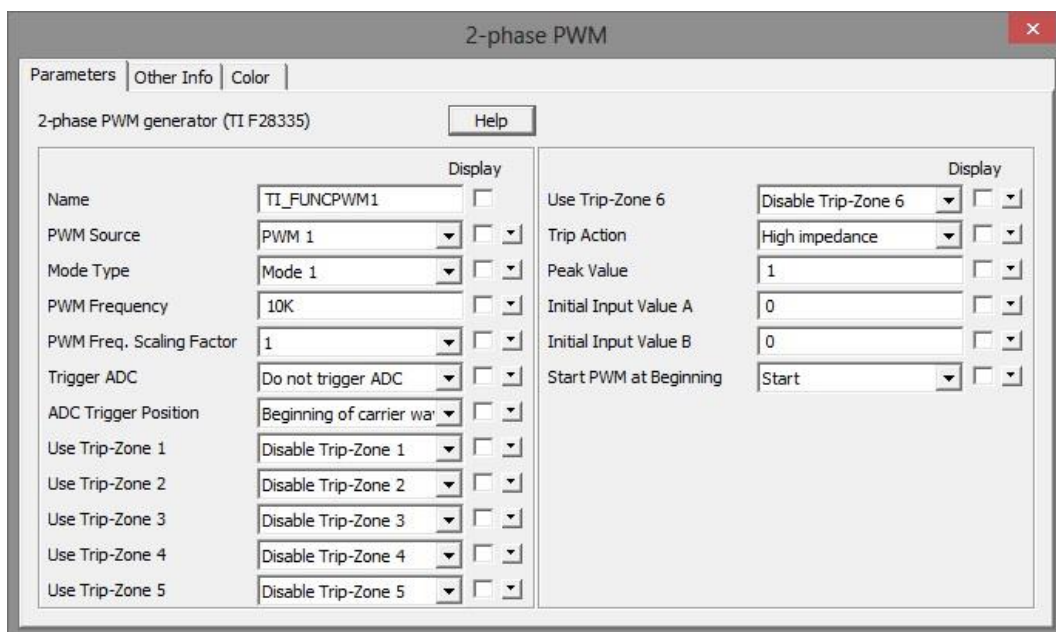


Fonte: Produção do próprio autor.

Na janela de configurações deste bloco, mostrada na Figura 26, podem ser alterados os seguintes parâmetros:

- *PWM Source*: Definição das portas correspondentes que serão utilizadas do DSC. São encontrados os canais de PWM1 a PWM6, sendo que cada canal utiliza duas portas GPIO. Para o PWM1, são usadas as portas GPIO0 e GPIO1, para o PWM2, são usadas as portas GPIO2 e GPIO3, seguindo este padrão até o PWM6. Observa-se que podem ser utilizados seis destes módulos em uma aplicação.
- *Mode Type*: Definição do modo de operação das saídas A e B, conforme será explicado posteriormente.
- *PWM Frequency*: Definição da frequência do bloco de PWM, em hertz, sendo esta a frequência de amostragem do Conversor A/D. Ao abrir a janela de configurações, este espaço já estará preenchido com 10 kHz.
- *PWM Frequency Scaling Factor*: Deve-se optar entre os fatores 1, 2 ou 3. Este fator de escala multiplica a frequência do PWM, determinando a frequência do sinal de saída do bloco de PWM.
- *Trigger ADC*: Deve-se escolher se o bloco PWM irá disparar o Conversor A/D e qual grupo do conversor será acionado, grupo A, grupo B, ou ambos os grupos.
- *ADC Trigger Position*: Deve-se optar se será feito o gatilho em um Conversor A/D, e se será feito no início ou no meio da portadora.
- *Use Trip-zones*: Definição de como serão utilizados os *Trip-zones* de 1 a 6, entre desabilitá-los, ou habilitá-los no modo *One-shot* ou *Cycle-by-Cycle*.
- *Trip Action*: Definição do gerador de PWM responderá à ação de um *Trip-zone*, podendo optar entre alta impedância para as duas saídas, para apenas a saída A ou apenas B, ou não tendo nenhuma ação.

Figura 26 – Configurações do bloco de PWM de duas fases.



Fonte: PSIM®.

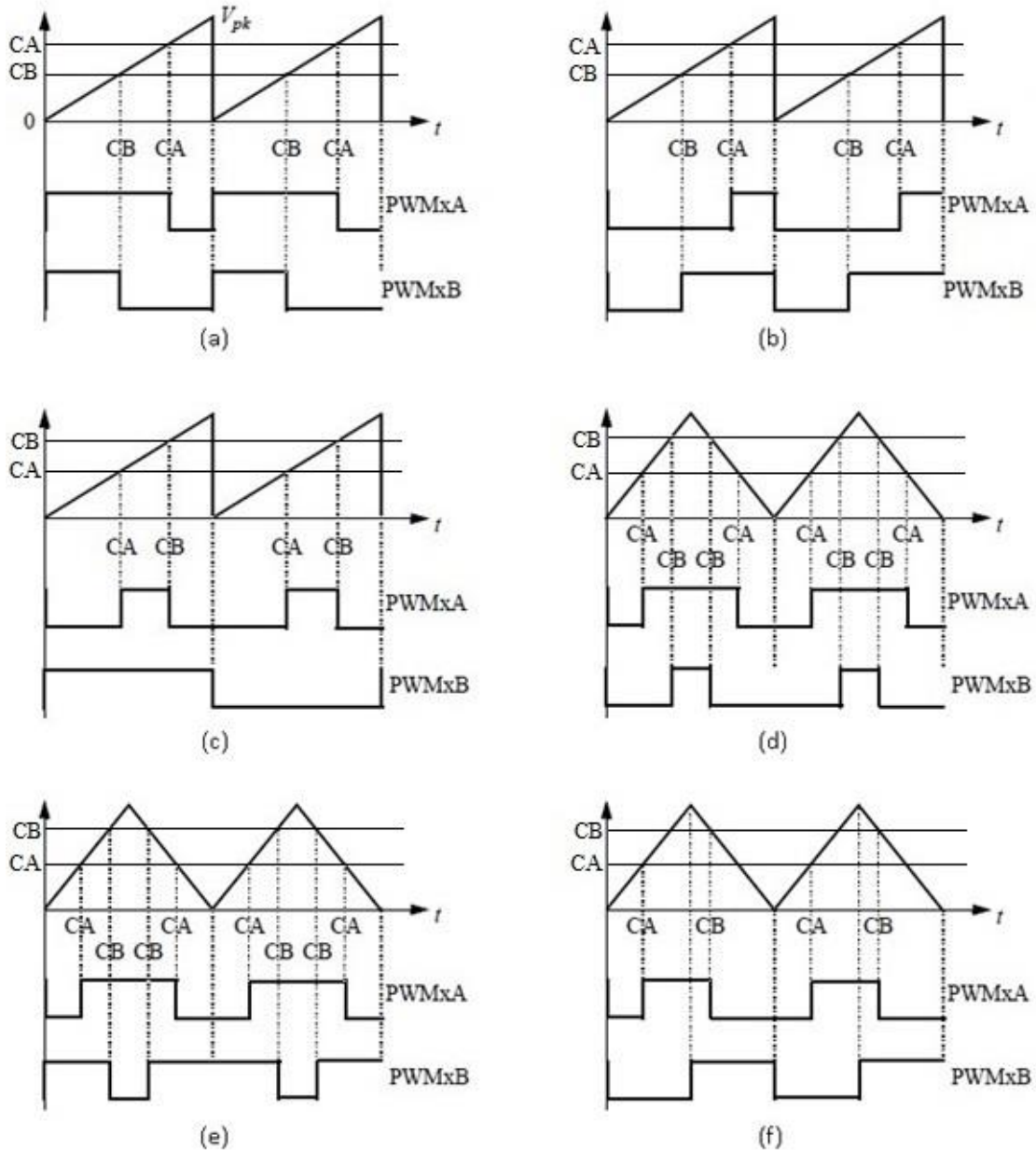
- *Peak-to-Peak Value*: Definição do valor pico-a-pico do sinal de entrada.
- *Initial Input Value*: Definição do valor inicial das entradas A e B. Este valor interfere apenas nos primeiros ciclos do sinal de saída.
- *Start PWM at Beginning*: Seleção entre “Start”, em que o gerador de PWM começará ativado, ou “Do not Start”, em que o gerador de PWM dependerá de um módulo Start PWM para funcionar.

O gerador de PWM de duas fases possui seis modos de operação, os quais serão explicados a seguir, podendo ser visualizados na Figura 27.

- 1) Modo 1: Utiliza uma onda dente-de-serra como portadora. Neste modo, enquanto os sinais de entrada são maiores que a portadora, os sinais de saída permanecem em nível lógico alto. Este modo de operação é mostrado Figura 27a.
- 2) Modo 2: Utiliza uma onda dente-de-serra como portadora. Neste modo ocorre o oposto do modo 1, ou seja, enquanto os sinais de entrada são maiores que a portadora, os sinais de saída permanecem em nível baixo, conforme mostrado na Figura 27b.
- 3) Modo 3: Utiliza uma onda dente-de-serra como portadora. Neste modo, as entradas do gerador mudam apenas a saída CA, sendo que a saída CB permanecendo inalterada,

com a metade da frequência e razão cíclica de 0,5. Como pode ser visto na Figura 27c, a entrada CA determina o ponto em que o sinal de saída PWMxA sobe para o nível alto, enquanto que a entrada B determina o ponto em que a saída PWMxB retorna para o nível lógico baixo.

Figura 27 – Modos de operação do bloco de PWM de Duas Fases: (a)-(f) correspondem aos modos de I a VI.



Fonte: Manual de Usuário do PSIM (POWERSIM, 2010).

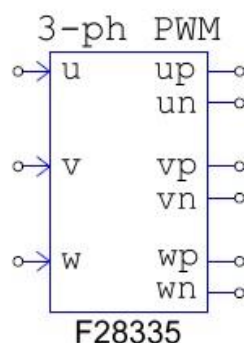
- 4) Modo 4: Utiliza uma onda triangular como portadora. Neste modo, enquanto os sinais de entrada são maiores que a portadora os sinais de saída permanece em nível lógico baixo, como pode ser visto na Figura 27d.
- 5) Modo 5: Utiliza uma onda triangular como portadora. Neste modo, a saída PWMxA se comporta igual ao modo 4, porém na saída PWMxB ocorre o contrário, enquanto o sinal de entrada for maior que a portadora, o sinal de saída permanece em nível alto, conforme ilustrado na Figura 27e.
- 6) Modo 6: Utiliza uma onda triangular como portadora. Neste modo as entradas interferem somente na saída PWMxA, enquanto a saída PWMxB permanece fixa com a mesma frequência da portadora e razão cíclica igual a 0,5. A entrada CA determina o ponto em que o sinal de saída passa do nível lógico baixo para o alto, tendo como referência a rampa de subida da portadora, enquanto a entrada CB determina o ponto em que o sinal volta para o nível baixo, tendo como referência a rampa de descida da portadora.

Os modos 1, 2, 4 e 5 podem ser utilizados quando se tem dois sinais de referência e não se deseja ter os sinais complementares do PWM. Já os modos 3 e 6 podem ser utilizados para se obter a diferença entre dois sinais constantes.

3.1.5. PWM Trifásico

O bloco de PWM Trifásico consiste em três moduladores monofásicos em somente um bloco. Este modulador possui três entradas de referência, uma para cada fase, chamadas de u , v e w , e seis saídas, duas para cada entrada, conforme visto na Figura 28. As saídas correspondentes às entradas u , v e w , estão representadas pelos índices p e n , representando, respectivamente, os sinais positivos e negativos, indicando que são complementares.

Figura 28 – Bloco de PWM Trifásico.



Fonte: Produção do próprio autor.

Estão disponíveis os seguintes parâmetros para a configuração do modulador de PWM Trifásico:

- *PWM Source*: Seleção das portas correspondentes que serão utilizadas do DSC. São divididos em dois canais, PWM123 e PWM456, sendo que cada canal utiliza seis portas GPIO. Para o canal PWM123, são usadas as portas GPIO0 a GPIO5 e para o canal PWM456, as portas usadas são GPIO6 a GPIO11. Observa-se que podem ser utilizados apenas dois geradores trifásicos em uma aplicação.
- *Dead Time*: Definição do tempo-morto entre os sinais de saída, em segundos. Este tempo é necessário, por exemplo, para que dois interruptores não sejam comandados a conduzir num mesmo instante de tempo. Ao abrir a janela de configurações, este espaço já estará preenchido com 4 μ s.
- *PWM Frequency*: Definição da frequência do PWM, em hertz, sendo a frequência de amostragem do Conversor A/D. Ao abrir a janela de configurações, este espaço já estará preenchido com 10 kHz.
- *PWM Frequency Scaling Factor*: Deve-se optar entre os fatores 1, 2 ou 3. Este fator de escala multiplica a frequência do PWM, determinando a frequência do sinal de saída do bloco de PWM.
- *Carrier Wave Type*: Definição do tipo de onda da portadora. Pode-se escolher entre a forma de onda triangular ou a dente-de-serra.
- *Trigger ADC*: Deve-se escolher se o bloco de PWM irá disparar o Conversor A/D e qual grupo do conversor será acionado, grupo A, grupo B, ou ambos os grupos.

- *ADC Trigger Position*: Deve-se optar se será feito o gatilho em um Conversor A/D, e se será feito para o grupo A, B ou para os dois grupos do conversor.
- *Use Trip-zones*: Deve-se escolher como serão utilizados os *Trip-zones* de 1 a 6, entre desabilitá-los, ou habilitá-los no modo *One-shot* ou *Cycle-by-Cycle*.
- *Trip Action*: Deve-se escolher como o bloco de PWM responderá à ação de um *Trip-zone*, podendo optar entre alta impedância para as duas saídas, para apenas a saída *p* ou apenas *n*, ou não executando nenhuma ação.
- *Peak-to-Peak Value*: Definição do valor pico-a-pico do sinal de entrada.
- *Offset Value*: Deve-se preencher com o valor de *offset* do sinal de entrada.
- *Initial Input Value*: Deve-se preencher com os valores iniciais das entradas *u*, *v* e *w*. Este valor interfere apenas nos primeiros ciclos do sinal de saída.
- *Start PWM at Beginning*: Deve-se selecionar entre “*Start*”, em que o gerador de PWM começará ativado, ou “*Do not Start*”, em que o gerador de PWM dependerá de um bloco *Start PWM* para funcionar.

Na Figura 29 pode ser observada a janela de configurações do modulador de PWM Trifásico.

Figura 29– Configurações do modulador de PWM Trifásico.

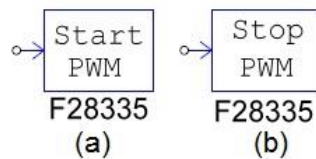
Parameters		Other Info		Color	
3-phase PWM generator (TI F28335)					
Name	TI_PWM3PH1	Display	Use Trip-Zone 5	Disable Trip-Zone 5	Display
PWM Source	3-phase PWM 123		Use Trip-Zone 6	Disable Trip-Zone 6	
Dead Time	4us		Trip Action	High impedance	
Sampling Frequency	10K		Peak-to-Peak Value	2	
PWM Freq. Scaling Factor	1		Offset Value	-1	
Carrier Wave Type	Triangular wave		Initial Input Value u	0	
Trigger ADC	Do not trigger ADC		Initial Input Value v	0	
ADC Trigger Position	0		Initial Input Value w	0	
Use Trip-Zone 1	Disable Trip-Zone 1		Start PWM at Beginning	Start	
Use Trip-Zone 2	Disable Trip-Zone 2				
Use Trip-Zone 3	Disable Trip-Zone 3				
Use Trip-Zone 4	Disable Trip-Zone 4				

Fonte: PSIM®.

3.2. START PWM E STOP PWM

Os blocos de *Start PWM* e *Stop PWM*, apresentados na Figura 30, têm como função ativar ou desativar os geradores de PWM. Ambos os componentes apresentam apenas uma entrada, sendo que o bloco é ativado se a entrada for de nível lógico alto (valor maior que 0.5).

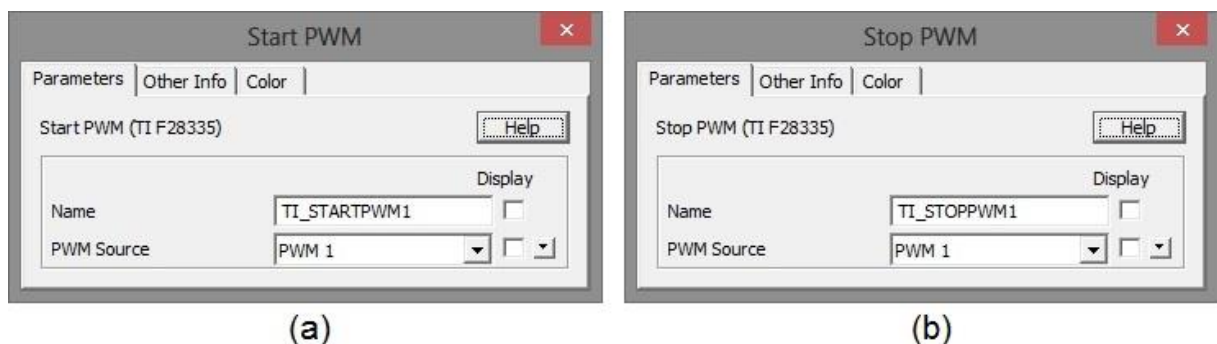
Figura 30 – Blocos (a) *Start PWM* e (b) *Stop PWM*.



Fonte: Produção do próprio autor.

Para configurar ambos os blocos, basta escolher qual canal de PWM será ativado ou desativado. Pode se optar entre os canais PWM1 a PWM6 para os blocos de PWM de uma ou duas fases, ou os canais PWM123 ou PWM456, se o bloco for o PWM Trifásico, ou Capture de 1 a 6, que podem ser usados tanto para os blocos de Captura, que serão mostrados em seguida, quanto para os moduladores de PWM Único. Na Figura 31 são mostradas as janelas de configurações dos blocos.

Figura 31 – Configurações dos blocos (a) *Start* e (b) *Stop PWM*.

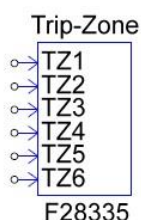


Fonte: PSIM®.

3.3. TRIP-ZONE E TRIP-ZONE STATE

Este bloco é utilizado para realizar uma interrupção nos moduladores de PWM. O bloco do *Trip-Zone* possui seis entradas, cada uma correspondendo a uma porta GPIO. Há somente seis portas GPIO disponíveis para se utilizar, portanto somente um bloco *Trip-Zone* pode ser utilizado por aplicação. A ação do *Trip-Zone* é ativada com a entrada em nível lógico baixo. Conforme mostrado na Figura 32, este bloco não apresenta nenhum pino de saída.

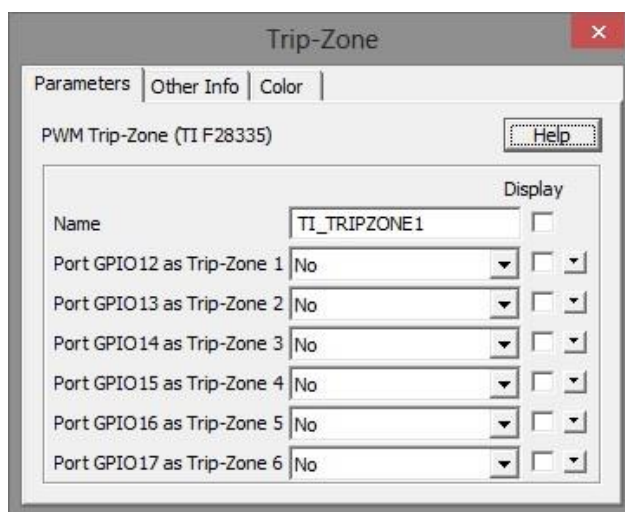
Figura 32 – Bloco *Trip-Zone*.



Fonte: Produção do próprio autor.

As portas disponíveis são de GPIO12 a GPIO17, correspondendo aos *Trip-Zones* de 1 a 6. Um modulador de PWM pode ser configurado para ser interrompido por mais de um *Trip-Zone*. Na Figura 33 pode-se ver a janela de configurações do *Trip-Zone*.

Figura 33 – Configurações do bloco *Trip-Zone*.



Fonte: PSIM®.

O *Trip-Zone* pode funcionar em dois modos, no modo *One-shot*, ou no modo *Cycle-by-cycle*. No modo *One-shot*, depois de ocorrida a interrupção da saída do bloco PWM, este deve ser reiniciado para funcionar novamente. Já no modo *Cycle-by-cycle*, apenas o ciclo atual do modulador de PWM é afetado pela interrupção. O modo de funcionamento deve ser escolhido nas configurações do bloco de PWM.

O bloco de estado do *Trip-Zone* tem apenas uma saída, conforme mostrado na Figura 34, tendo a função de indicar o modo de operação do sinal do *Trip-Zone*. Este bloco é ativado quando o modulador de PWM sofre uma interrupção, tendo sua saída em nível lógico alto para o modo *One-shot* e em nível lógico baixo para o modo *Cycle-by-cycle*. Não é verificado a opção de prioridade para os canais do *Trip-Zone*.

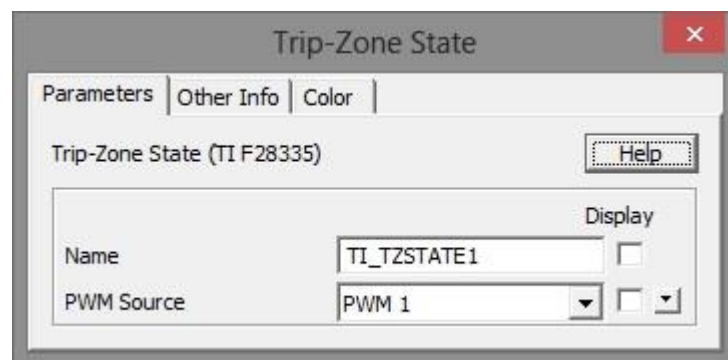
Figura 34 – Bloco *Trip-Zone State*.



Fonte: Produção do próprio autor.

Para configurar este bloco, deve-se apenas escolher qual PWM será utilizado para supervisionar. Portanto, podem-se utilizar um deste componente para cada bloco de PWM. Sua janela de configurações é mostrada na Figura 35.

Figura 35 – Configurações do bloco *Trip-Zone State*.

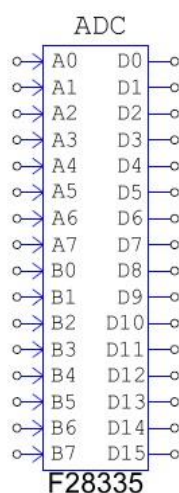


Fonte: PSIM®.

3.4. CONVERSOR A/D

O bloco do Conversor A/D possui dezesseis entradas, divididas em dois grupos A e B. A tensão em cada entrada deve estar entre 0 e 3V no modo DC, ou de -1,5V a 1,5V no modo AC, porém este modo não é possível com o DSC. Cada entrada possui uma saída correspondente, em que se pode ter um ganho de tensão, que pode ser utilizado para compensar as atenuações comuns aos circuitos de medição. O conversor A/D pode ser observado na Figura 36.

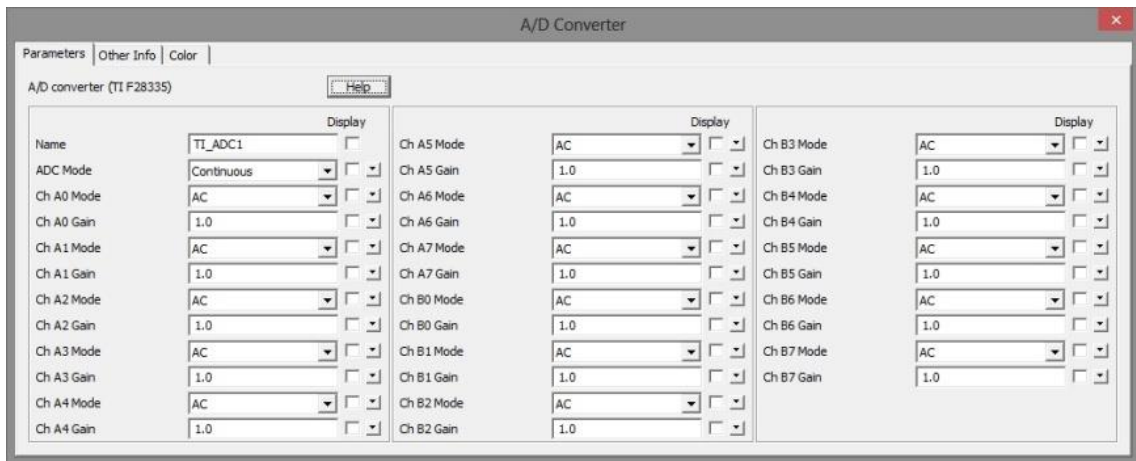
Figura 36 – Bloco Conversor A/D.



Fonte: Produção do próprio autor.

Na Figura 37, são mostrados os parâmetros de configuração do Conversor A/D. O primeiro parâmetro é o modo, onde deve-se definir entre *Contínuo*, *Start/Stop* de 8 canais ou *Start/Stop* 16 canais. No modo *Contínuo*, o conversor realiza continuamente a conversão de todas as entradas. Nos modos *Start/Stop*, é realizada a conversão somente quando pedida, para apenas um dos grupos de oito canais ou para os dois grupos, ou seja, para os 16 canais. Este modo é utilizado para o gatilho dos geradores de PWM. Para cada canal utilizado do conversor A/D, deve ser configurado um modo, entre AC e DC, e o ganho da saída.

Figura 37 – Configurações do bloco Conversor A/D.

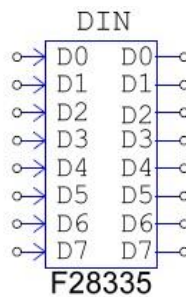


Fonte: PSIM®.

3.5. ENTRADAS E SAÍDAS DIGITAIS

Estes blocos têm como função configurar quais as entradas e saídas digitais. Ambos possuem oito entradas e oito saídas. Na Figura 38 é mostrado o bloco de entradas digitais.

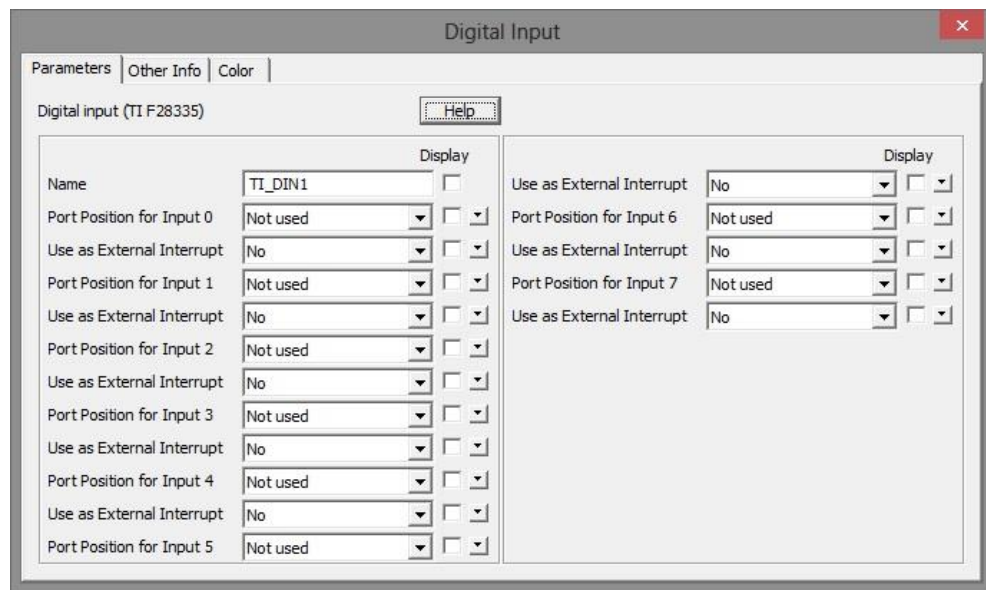
Figura 38 – Bloco de Entradas Digitais.



Fonte: Produção do próprio autor.

No bloco de Entradas Digitais, é preciso configurar quais portas GPIO serão usadas e se serão utilizadas como interrupção externa. A janela de configurações deste componente é mostrada na Figura 39.

Figura 39 – Configurações do bloco de Entradas Digitais.

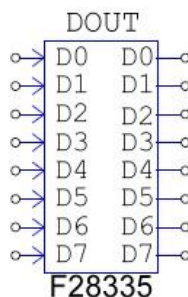


Fonte: PSIM®.

Qualquer uma das oitenta e oito portas disponíveis pode ser configurada como entrada ou saída digital, sendo que os pinos GPIO0 ao GPIO63 podem ser utilizados para interrupção externa. Portanto, podem ser utilizados diversos blocos de entradas e saídas digitais em uma simulação, porém deve-se ter atenção para não utilizar a mesma porta em mais de um bloco.

Já o bloco de Saídas Digitais, mostrado na Figura 40, deve ser configurado apenas quais portas GPIO serão utilizadas na aplicação. Percebe-se que tanto o bloco de Entrada Digital quanto o bloco de Saída Digital devem ser inseridos na aplicação de modo que não interfiram nos valores de tensão, ou seja, a tensão da entrada do bloco é igual à tensão de saída.

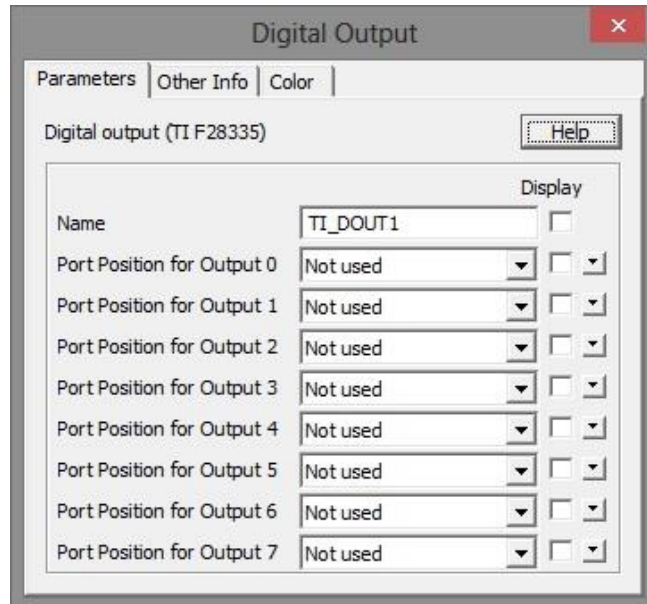
Figura 40 – Bloco de Saídas Digitais.



Fonte: Produção do próprio autor.

As configurações do bloco de Saídas Digitais podem ser vistas na Figura 41.

Figura 41 – Configurações do bloco de Saídas Digitais.

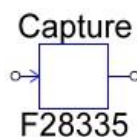


Fonte: PSIM®.

3.6. CAPTURE E CAPTURE STATE

O bloco de Captura, apresentado na Figura 42, é capaz de gerar interrupções a partir do sinal de entrada.

Figura 42 – Bloco de Captura.



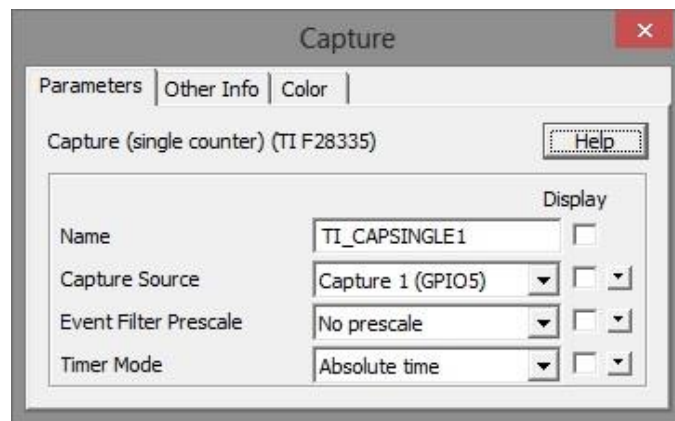
Fonte: Produção do próprio autor.

Nas configurações deste bloco, encontramos:

- *Capture Source*: Definição da porta correspondente que se deseja usar. Percebe-se que as portas disponíveis para este componente são as mesmas que para o gerador de PWM único.
- *Event Filter Prescale*: Definição do valor que dividirá o sinal de entrada.
- *Timer Mode*: Deve-se escolher o modo de contagem de tempo entre modo absoluto ou diferencial.

Na Figura 43, é mostrada a janela de configurações deste componente.

Figura 43 – Configurações do bloco de Captura.



Fonte: PSIM®.

O componente de Estado de Captura, apresentado na Figura 44, apresenta nível lógico alto em sua saída quando o sinal de captura é ativado, e nível lógico baixo, quando o sinal de captura é desativado.

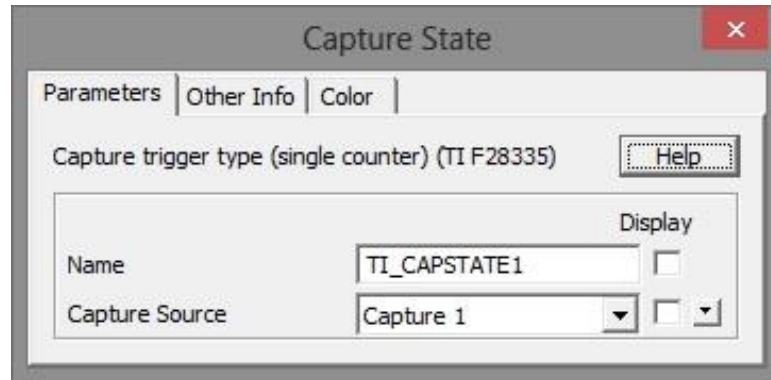
Figura 44 – Bloco de Estado de Captura.



Fonte: Produção do próprio autor.

Para configurá-lo, deve-se apenas selecionar a fonte do sinal de captura, ou seja, qual porta está sendo utilizada, conforme visto na Figura 45.

Figura 45 – Configurações do bloco de captura de estado.

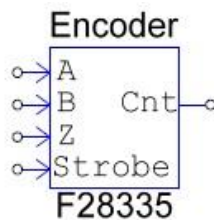


Fonte: PSIM®.

3.7. ENCODER E ENCODER STATE

Este bloco configura uma entrada de *Encoder*. Conforme mostrado na Figura 46, este componente possui quatro entradas e uma saída. Nas entradas A e B devem-se colocar ondas quadradas defasadas em 90° entre si. A entrada Z indica que ocorreu volta do *Encoder*, devendo ter um sinal quadrado como entrada. A entrada *Strobe* pode ser utilizada como fonte de interrupção, assim como a entrada Z.

Figura 46 – Bloco do *Encoder*.



Fonte: Produção do próprio autor.

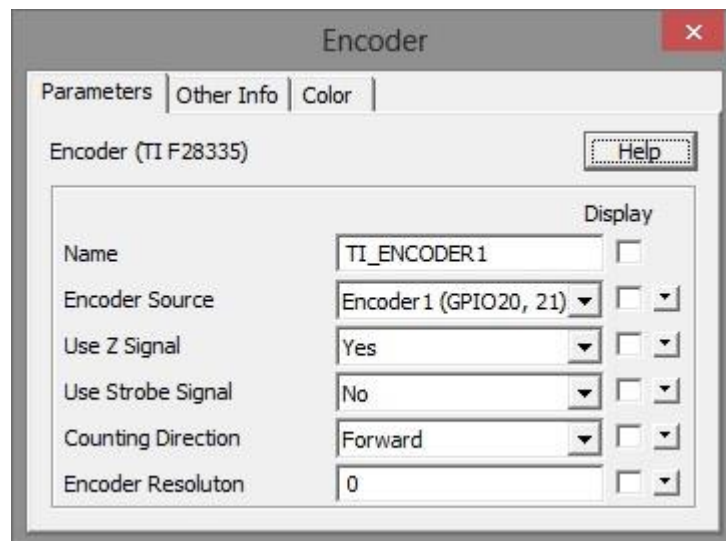
Este elemento conta a quantidade de pulsos das entradas A e B, tendo as entradas Z e *Strobe* como referência de contagem. Porém, pode-se optar por não utilizar as duas últimas entradas, fazendo com que o *Encoder* conte infinitamente.

Para configurar este componente, encontramos os seguintes itens:

- *Encoder Source*: Devem-se escolher quais portas GPIO serão utilizadas. Podem-se utilizar até dois *Encoders* por aplicação, podendo configurá-los em *Encoder1*, com as portas GPIO 20 e 21 ou 50 e 51, ou em *Encoder2*, com as portas GPIO 24 e 25, sendo que uma porta corresponde à saída *Cnt* e outra à saída do bloco *Encoder State*.
- *Use Z Signal*: Deve-se optar em utilizar a entrada Z ou não.
- *Use Strobe Signal*: Deve-se optar em utilizar a entrada *Strobe* ou não.
- *Counting Direction*: Definição da direção de contagem, sendo *Forward* a ordem crescente e *Reverse* a ordem decrescente.
- *Encoder Resolution*: Definição da resolução do *Encoder*, sendo o número de pulsos por volta.

Estas configurações podem ser vistas na Figura 47.

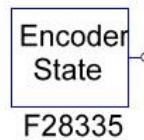
Figura 47 – Configurações do bloco *Encoder*.



Fonte: PSIM®.

O bloco de *Encoder State* mostrado na Figura 48, tem a função de indicar a fonte da interrupção. Este apresenta em sua saída nível lógico alto quando o sinal Z provoca a interrupção, e nível lógico baixo quando o sinal do *Strobe* gera a interrupção.

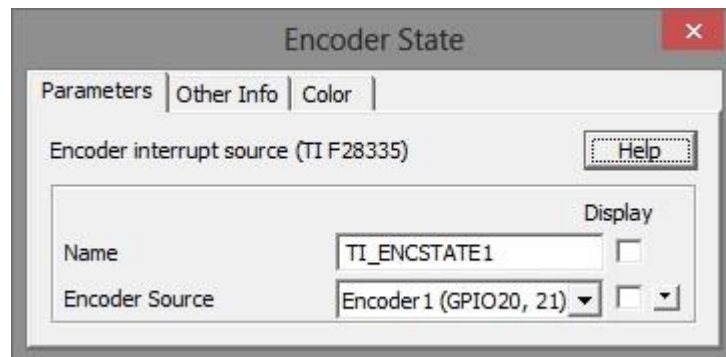
Figura 48 – Bloco do *Encoder State*.



Fonte: Produção do próprio autor.

Para configurá-lo, basta escolher quais portas serão utilizadas. Lembrando que se deve utilizar as mesmas portas que foram configuradas para o *Encoder*. Na Figura 49 é mostrada a janela de configurações do *Encoder State*.

Figura 49 – Configurações bloco do *Encoder State*.



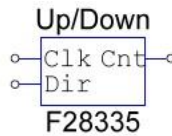
Fonte: PSIM®.

3.8. CONTADOR UP/DOWN

Este Contador é basicamente uma versão simplificada do *Encoder*. Ele possui apenas duas entradas, *Clk* e *Dir*, e a saída *Cnt*, conforme visto na Figura 50. A cada pulso na entrada *Clk*, o valor da saída *Cnt* é incrementado ou decrementado em uma unidade. A entrada *Dir*

tem como função determinar a direção de contagem. Se o valor da entrada *Dir*=1, a contagem é crescente, e se o valor de *Dir*=0, a contagem é decrescente.

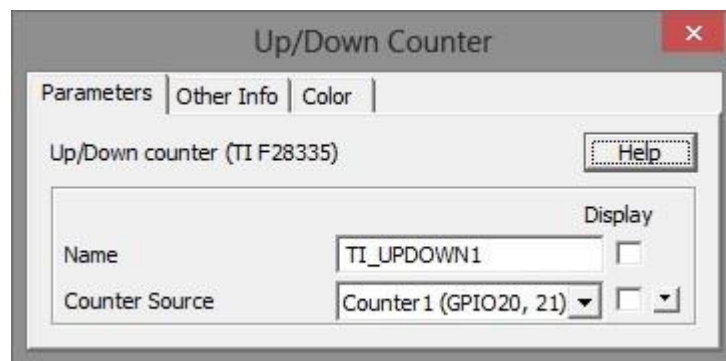
Figura 50 – Bloco do Contador *Up/Down*.



Fonte: Produção do próprio autor.

Para configurar este Contador, basta escolher qual pino será utilizado. Percebe-se que as portas GPIO disponíveis são as mesmas usadas para o *Encoder*, porém o contador utiliza somente uma porta. Igualmente ao *Encoder*, podem-se utilizar somente dois Contadores em uma aplicação. A janela de configurações do Contador *Up/Down* é mostrado na Figura 51.

Figura 51 – Configurações do bloco Contador *Up/Down*.



Fonte: PSIM®.

3.9. DSP CLOCK

A ferramenta *SimCoder* possui este componente, capaz de configurar uma fonte de *clock* externa. O componente é apresentado na Figura 52.

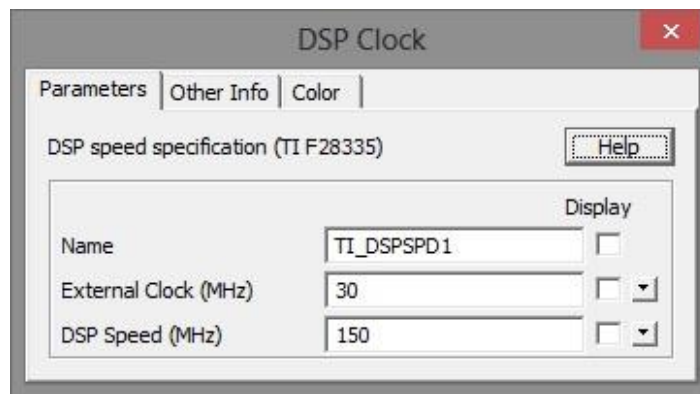
Figura 52 – Bloco de DSP Clock.



Fonte: Produção do próprio autor.

Pode-se configurar a frequência do *clock* externo inserindo um número inteiro até 30MHz, caso seja necessário utilizá-lo. Pode-se também determinar a velocidade do DSP, também em MHz, que deve ser um valor inteiro e múltiplo do valor determinado para o *clock* externo, sendo no máximo 150MHz. Na Figura 53 é mostrada a janela de configurações deste componente.

Figura 53 – Configurações do bloco de DSP Clock.



Fonte: PSIM®.

3.10. CONFIGURAÇÃO DE HARDWARE

Através deste componente, podem-se configurar todas as portas que serão utilizadas na aplicação. Podem-se verificar quais portas podem ser utilizadas para quais funções. É uma ferramenta muito útil para evitar configurar dois componentes com a mesma porta GPIO. Porém, mesmo utilizando este componente, se faz necessário configurar cada elemento individualmente. Na Figura 54 é mostrado o componente de Configuração de Hardware.

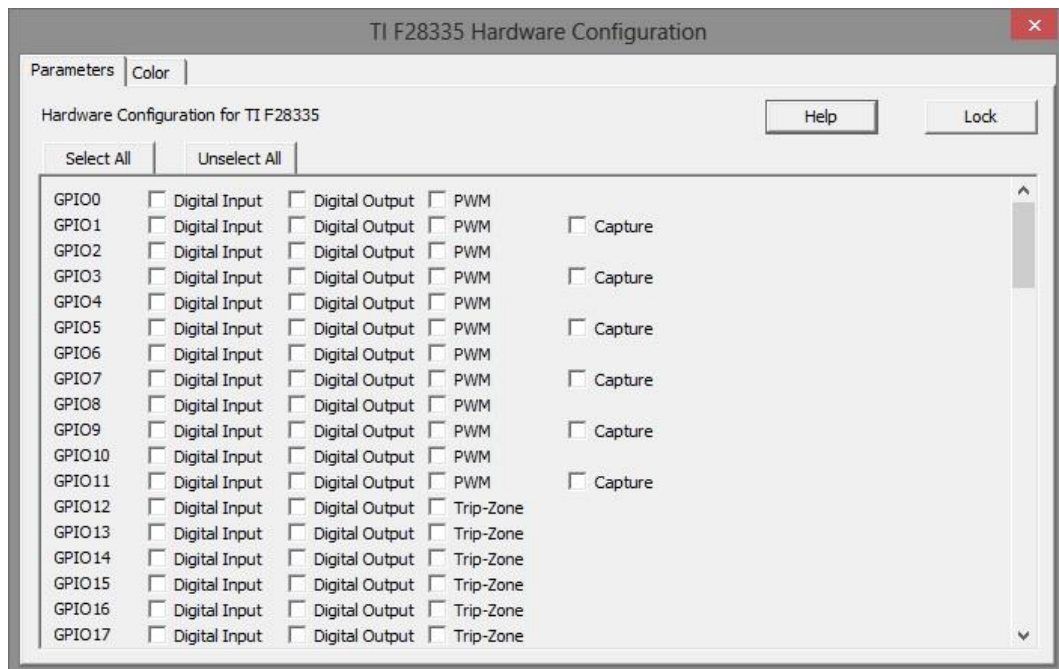
Figura 54 – Bloco de Configuração de Hardware.



Fonte: Produção do próprio autor.

Após configurá-lo, deve-se travar este componente pelo botão *Lock*, conforme mostrado na Figura 55. Para destravar, basta acioná-lo novamente.

Figura 55 – Configurações do bloco de Configuração de Hardware.



Fonte: PSIM®.

3.11. OUTROS COMPONENTES

Além dos componentes específicos para o DSC TI F28335, podem-se utilizar outros componentes encontrados no PSIM. Os componentes podem ser encontrados em *Elements*, sendo identificados pelo símbolo mostrado na Figura 56. Entre os elementos encontramos blocos de operações matemáticas, elementos de lógica, elementos de controle digital, entre outros.

Figura 56 - Símbolo para elementos que podem ser utilizados para geração de código.

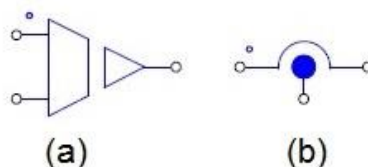


Fonte: Produção do próprio autor.

O software PSIM diferencia circuitos de potência de circuitos de controle. Os circuitos de potência têm ligações em vermelho, enquanto circuitos de controle têm ligações em verde. Os componentes do *SimCoder* e outros elementos identificados pelo símbolo acima são elementos de controle, portanto não podem ser ligados aos componentes, como resistores, transistores, amplificadores operacionais, entre outros.

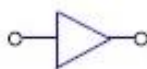
Para um elemento do *SimCoder*, como por exemplo, um conversor A/D ou uma entrada analógica, ler um valor de tensão ou corrente em um circuito simulado, é necessário utilizar um sensor de tensão ou de corrente, apresentados na Figura 57. Deste modo o valor de potência é transformado em um sinal de controle.

Figura 57 – Blocos sensores de: (a) tensão e (b) corrente.



Fonte: Produção do próprio autor.

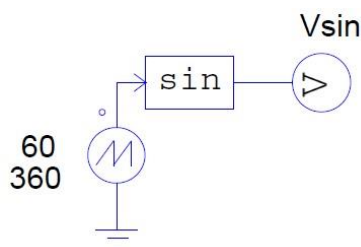
Para inserir uma saída de um componente do *SimCoder*, como por exemplo um gerador de PWM, como entrada em um circuito simulado, se faz necessário utilizar um controlador *On-Off*, que pode ser visto na Figura 58. Deste modo o sinal de controle é transformado em um valor de tensão que pode ser utilizado para acionar um transistor, por exemplo, em um circuito de potência.

Figura 58 - Controlador *On-Off*.

Fonte: Produção do próprio autor.

Eventualmente, será necessário utilizar em simulações uma entrada com sinal senoidal, como por exemplo, na modulação de um controle de um conversor cc-ca. Porém os geradores de PWM do *SimCoder* não aceitam uma fonte de tensão senoidal como entrada. Uma possível solução é utilizar uma fonte senoidal externa e um conversor A/D. Porém, podem-se utilizar alguns elementos do PSIM que são aceitos normalmente para geração de código. Na Figura 59 é mostrado como é gerado o sinal senoidal.

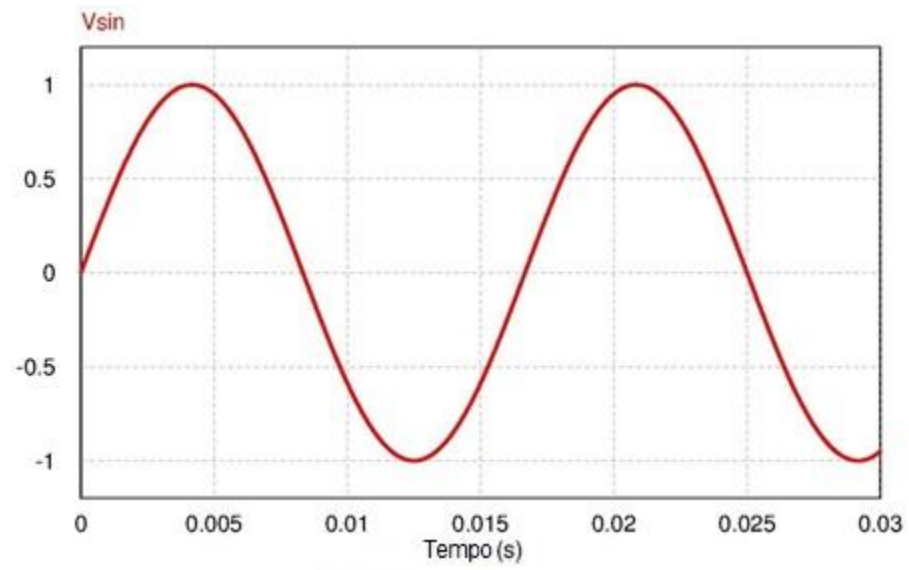
Figura 59 - Sinal senoidal utilizado para as simulações.



Fonte: Produção do próprio autor.

Para gerar uma onda senoidal, basta utilizar um gerador de onda dente de serra, com amplitude de 360, e uma função seno. Deste modo, obtém-se uma onda senoidal de amplitude unitária, de mesma frequência que a onda dente-de-serra, que pode ser usada como referência para um gerador PWM. A função seno é encontrada em *Elements>Control>Computational Blocks* e o gerador dente-de-serra em *Elements>Sources>Voltage*. Na Figura 60 é mostrada a forma de onda do sinal obtido.

Figura 60 - Sinal senoidal obtido utilizando uma onda dente-de-serra e uma função seno.



Fonte: Produção do próprio autor.

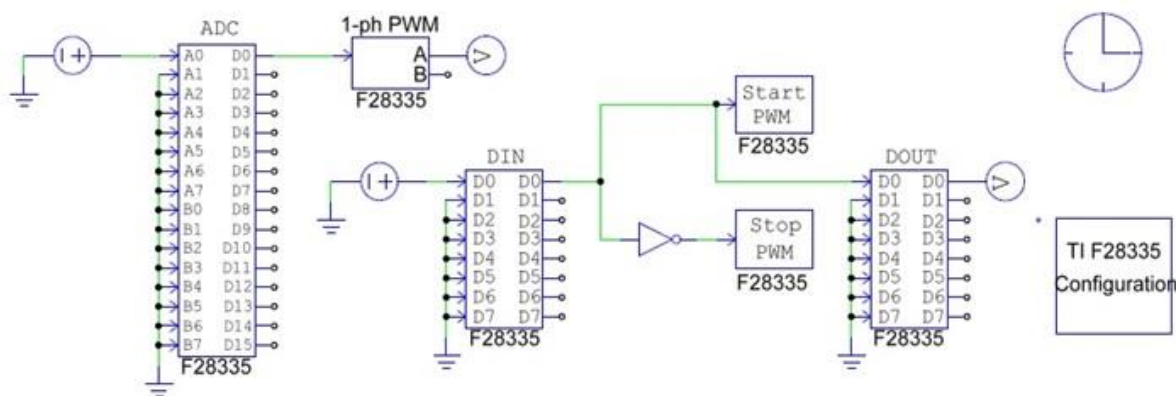
4. CÓDIGO GERADO E O CODE COMPOSER STUDIO

Após o estudo dos elementos presentes no *SimCoder*, é possível realizar simulações utilizando esses componentes e posteriormente gerando códigos em linguagem C para aplicação no *Code Composer Studio*.

4.1. ESTUDO DO CÓDIGO GERADO

Na Figura 61 pode ser visto um circuito de simulação para o teste de alguns elementos. Foi utilizado um conversor A/D para obter uma entrada analógica. O valor obtido do conversor A/D é usado como entrada para um gerador de PWM monofásico. Foi utilizada também uma entrada digital, um Start PWM e um Stop PWM para ligar e desligar o gerador de PWM. Por fim foi utilizada uma saída digital apenas para mostrar o estado de funcionamento do gerador de PWM.

Figura 61 – Circuito de simulação para testes de alguns elementos do *SimCoder*.



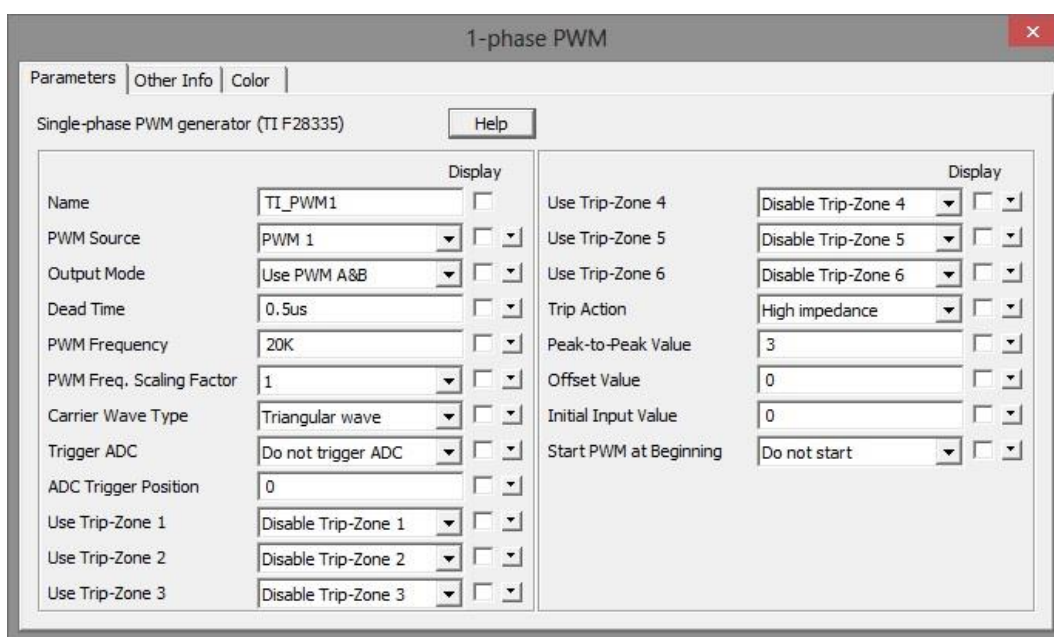
Fonte: Produção do próprio autor.

O conversor A/D foi configurado para leitura contínua, no modo DC e com ganho unitário. O bloco de PWM e os blocos *Start* e *Stop* PWM foram todos configurados para o canal PWM1, que utiliza as portas GPIO0 e GPIO1. Além disso, o bloco de PWM foi configurado para ter entrada de 0 a 3V e para não começar ligado. Para a entrada e para a

saída digital foram utilizadas, respectivamente as portas GPIO5 e GPIO6, não utilizando a entrada digital como interrupção externa.

O modulador de PWM foi configurado para valor de pico-a-pico de 3V, valor de offset 0, tempo morto de 0,5 μ s e frequência de 20kHz. Na Figura 62 é mostrada a janela de configurações do modulador PWM utilizado.

Figura 62 - Parâmetros configurados do bloco de PWM utilizado para teste.



Fonte: PSIM®.

Por fim, todas as entradas não utilizadas dos componentes foram devidamente aterradas. No bloco de configurações de hardware, foram marcadas as portas utilizadas e então travado. Foi também selecionada a opção TI F28335 no *Hardware Target*, do controle de simulação. Para finalmente gerar o código, foi simulada a aplicação e selecionado *Generate Code*, em *Simulate*. O código em linguagem C obtido pode ser dividido em partes, para auxiliar o entendimento, conforme visto abaixo.

A primeira parte do código tem como objetivo a inicialização das bibliotecas utilizadas e apresentação das funções do programa. Percebe-se que uma das bibliotecas é a *PS_bios.h* gerada a partir da simulação do PSIM.

```

/*****
// ThiscodeiscreatedbySimCoderVersion 2.0 for F28335 Hardware Target
//
// SimCoderis copyright byPowersim Inc., 2009
//
// Date: May 07, 2014 17:27:12
*****/

#include    <math.h>
#include    "PS_bios.h"
TypedeffloatDefaultType;
#define     GetCurTime() PS_GetSysTimer()

interruptvoidTask();
voidTask_1();

```

A parte do código abaixo representa uma função responsável pela leitura da entrada analógica e por determinar o ciclo de trabalho do PWM. Caso houvesse alguma lógica que envolvesse o valor da entrada analógica ou o ciclo de trabalho do PWM, esta lógica estaria nesta parte do código. Esta função não precisa ser chamada por ser do tipo *interrupt*, ou seja, a cada período de amostragem ela é ativada. Vemos na função *PS_GetDcAdc* que é lido o valor da entrada 0 do conversor A/D, conforme feito na simulação do PSIM.

```

interruptvoidTask()
{
    DefaultType fTI_ADC1;
    PS_EnableIntr();

    fTI_ADC1 = PS_GetDcAdc(0);
    PS_SetPwm1Rate(fTI_ADC1);
    PS_ExitPwm1General();
}

```

Já esta seção é a função encarregada de ligar e desligar o PWM. Esta função também se encarrega de verificar a entrada digital e de determinar o valor da saída digital. Nas funções

PS_StartPwm e *PS_StopPwm*, é possível ver que ambos estão associados ao PWM1, que será inicializado a seguir.

```
voidTask_1()
{
    DefaultType fTI_DIN2, fNOT1;

    fTI_DIN2 = (PS_GetDigitInA() & ((Uint32)1 << 5)) ? 1 : 0;
    if (fTI_DIN2 > 0)
    {
        PS_StartPwm(1);
    }
    fNOT1 = !fTI_DIN2;
    if (fNOT1 != 0)
    {
        PS_StopPwm(1);
    }
    (fTI_DIN2 == 0) ? PS_ClearDigitOutBitA((Uint32)1 << 6) :
PS_SetDigitOutBitA((Uint32)1 << 6);
}
```

Esta última função inicializa todos os elementos do programa, ou seja, o PWM, a entrada analógica, a entrada e a saída digital. Se houvesse outros elementos na simulação, todos seriam configurados nesta seção. É possível realizar comparações entre o código gerado e as configurações dos elementos da simulação do PSIM. Vemos na função *PS_InitPwm* que estão configurados: o número do PWM; o tipo de onda da portadora, sendo 1 para onda triangular e 0 para dente de serra; a frequência do PWM, sendo obtido pela multiplicação da frequência de amostragem pelo fator de escala; o tempo morto e o modo da saída. Nas funções abaixo, podem ser vistas as configurações da entrada do PWM, do gatilho do conversor A/D e configurações do conversor A/D. Por fim podem ser vistas as inicializações da entrada e da saída digital e referente aos blocos *Start e Stop PWM*.

```

void Initialize(void)
{
    PS_SysInit(30, 10);
    PS_StartStopPwmClock(0);
    PS_InitTimer(0, 0xffffffff);
    PS_InitPwm(1, 1, 20000*1, (0.5e-6)*1e6, PWM_TWO_OUT, 34968);    // pwnNo,
    waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(1, 3, 0, 1.0/3);
    PS_SetPwmIntrType(1, ePwmNoAdc, 1, 0);
    PS_SetPwmVector(1, ePwmNoAdc, Task);
    PS_SetPwm1Rate(0);
    PS_StopPwm(1);

    PS_ResetAdcConvSeq();
    PS_SetAdcConvSeq(eAdcCascade, 0, 1.0);
    PS_AdcInit(0, !0);

    PS_InitDigitIn(5, 100);

    PS_InitDigitOut(6);

    PS_StartStopPwmClock(1);
}

```

Por fim, podem ser vistas a parte principal do código. Primeiramente é chamada a função para inicializar o programa, em seguida habilita a interrupção interna e por fim, entra em um ciclo infinito, em que é constantemente chamada a função Task_1, responsável pelo funcionamento do PWM.

```

void main()
{
    Initialize();
    PS_EnableIntr();    // Enable Global interrupt INTM
    PS_EnableDbgm();
    for (;;) {
        Task_1();
    }
}

```

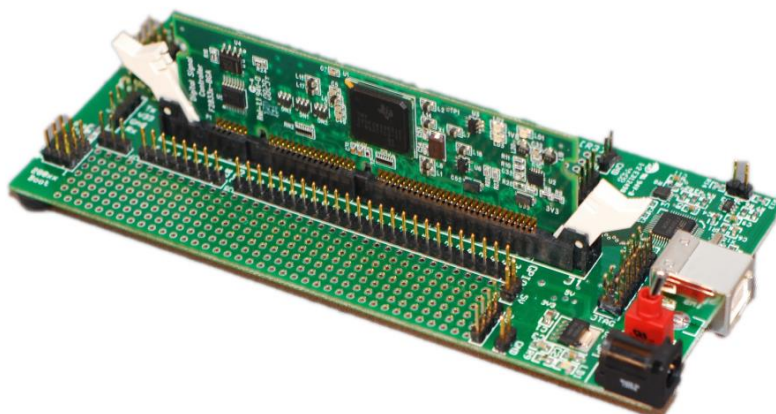
```
}  
}
```

É importante entender código gerado, pois é possível realizar pequenas modificações nas configurações. Porém quando é feita alguma modificação na simulação do PSIM® e gerado um novo código, o *Code Composer Studio* permite que o código seja atualizado automaticamente. Isto ocorre somente se o código encontra-se na mesma área de trabalho utilizada.

4.2. O DSC E O CODE COMPOSER STUDIO

O TMS320F28335 é um controlador digital de sinais de ponto flutuante da *Texas Instruments*. De acordo com o manual do usuário, o DSC possui frequência máxima de *clock* de 150 MHz e 6,67ns de ciclo de instruções. Possui interface externa de 16 a 32 bits e conversor A/D de 12 bits, sendo disponibilizados em dezesseis canais e tendo tempo de conversão de 80 ns. Apresenta seis canais de PWM, oitenta e oito portas disponíveis para entradas e saídas digitais e oito portas para interrupção externa. O DSC possui arquitetura Harvard Bus e memória de 256kB. Na Figura 63, é mostrado o kit de experimentação DSC utilizado.

Figura 63 – Kit de experimentação do TMS320F28335 da *Texas Instruments*.



Fonte: TEXAS INSTRUMENTS, 2011.

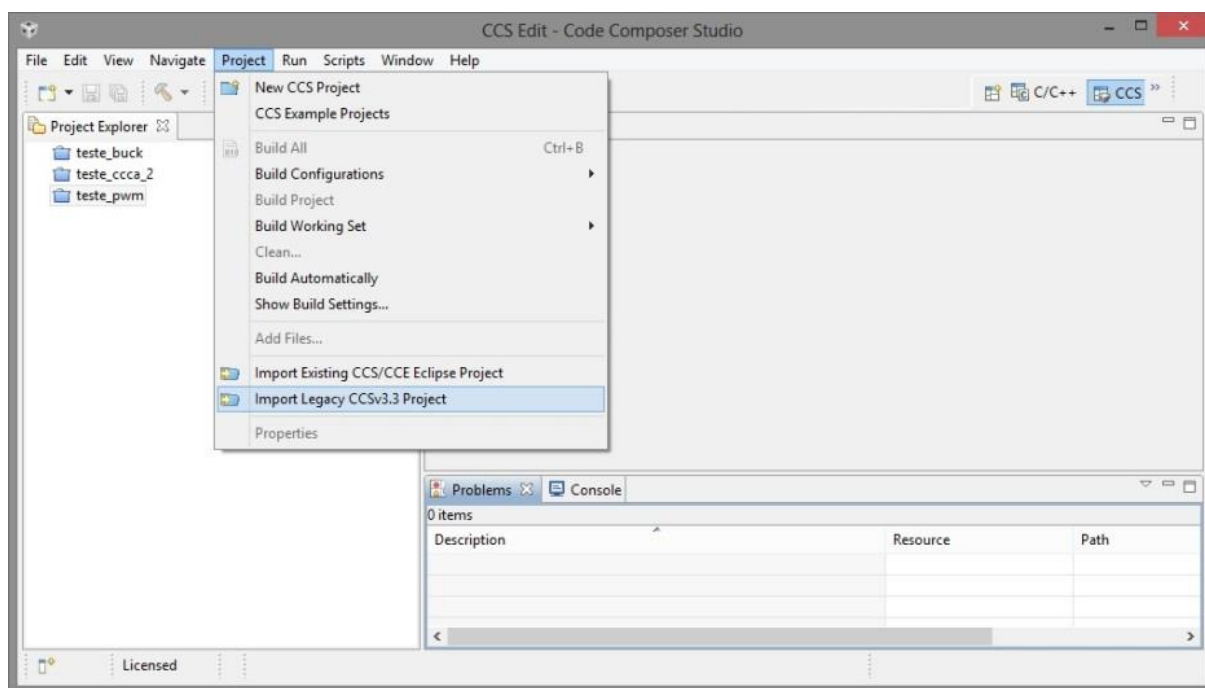
O *Code Composer Studio* (CCS) é o *software* desenvolvido para DSC's da *Texas Instruments* (TI), microcontroladores e microprocessadores. O *software* inclui um conjunto de ferramentas utilizadas para desenvolver e depurar códigos em linguagem C. Ele inclui compiladores para cada uma das famílias de dispositivos da TI, editor de código fonte, ambiente de construção de projetos, simuladores e muitos outros recursos.

O *software* PSIM, através da ferramenta *SimCoder*, é capaz de gerar um código pronto para implementar no DSC. O *SimCoder* gera o código para o CCS versão 3.3. Neste trabalho, será utilizada a versão 5.1 deste *software*, mostrado na Figura 64.

Quando gerado o código no PSIM, diversos arquivos são simultaneamente criados:

- Arquivo “.c”: este arquivo terá o mesmo nome do arquivo do PSIM, sendo o código principal da simulação.
- Arquivo “.pjt”: arquivo de projeto do CCS e também possui o mesmo nome do arquivo do PSIM.
- PS_bios.h: Arquivo cabeçalho para as bibliotecas do F28335 do *SimCoder*.

Figura 64 - *Code Composer Studio* versão 5.1.

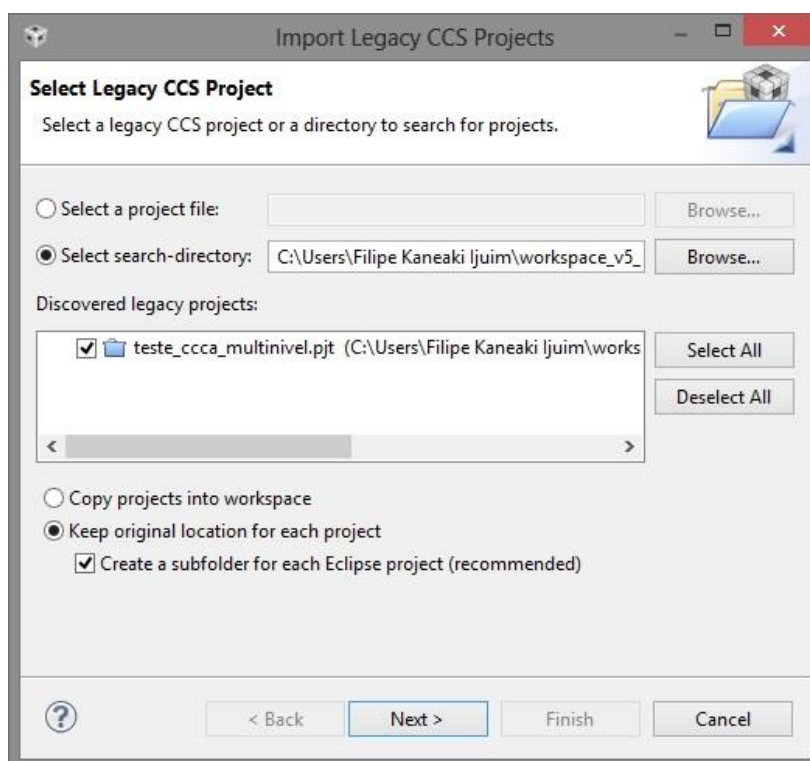


Fonte: Produção do próprio autor.

- PsBiosFlash.lib: Biblioteca do F28335 do *SimCoder*.
- PsBiosRam.lib: Biblioteca do F28335 do *SimCoder*.
- C28x_FPU_FastRTS_beta1.lib: Biblioteca da *Texas Instruments* para DSC de ponto-flutuante.
- Password.asm: arquivo para especificar o código senha do DSC.
- DSP2833x_Headers_nonBIOS.cmd: Arquivo de acesso ao registro periférico.
- F28335_FLASH_Lnk.cmd: Arquivo de acesso à memória *Flash*.
- F28335_FLASH_RAM_Lnk.cmd: Arquivo de acesso à memória *Flash Ram*.
- F28335_RAM_Lnk.cmd: Arquivo de acesso à memória *Ram*.

Todos os arquivos apresentados fazem parte do projeto gerado para o *Code Composer Studio* v3.3. Para abrir este projeto na versão 5.1 ou posterior deste *software*, deve-se selecionar *Project/Import Legacy CCSv3.3 Project*, como pôde ser observado na Figura 64. Na Figura 65 é mostrada a janela para importação do projeto.

Figura 65 - Janela para importar um projeto do CCS versão 3.3.

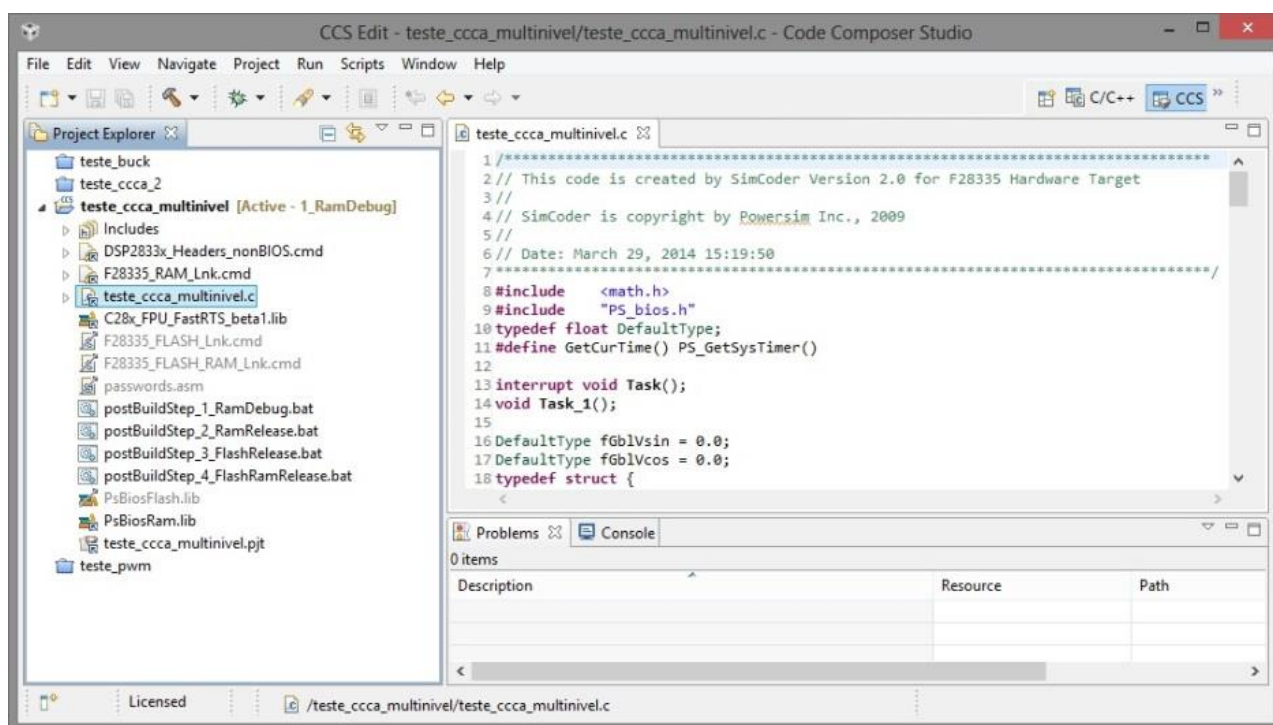


Fonte: Produção do próprio autor.

Depois de selecionada a pasta do projeto, basta clicar em *Next* e então em *Finish* para criar o projeto para o *Code Composer Studio v5.1*.

Selecionando o arquivo “teste_ccca_multinivel.c”, pode-se ver o código que foi gerado pelo *SimCoder*, conforme visto na Figura 66. Através deste código pode-se compilar o projeto sem que seja necessária nenhuma modificação.

Figura 66 – Visualização do código gerado através do CCSv5.1



Fonte: Produção do próprio autor.

Para facilitar o processo, sugere-se sempre salvar a simulação do PSIM na área de trabalho utilizada no CCS. Caso seja feita alguma modificação na simulação do PSIM, o código gerado será atualizado automaticamente no CCS, evitando que todo o processo de importação tenha que ser feito novamente.

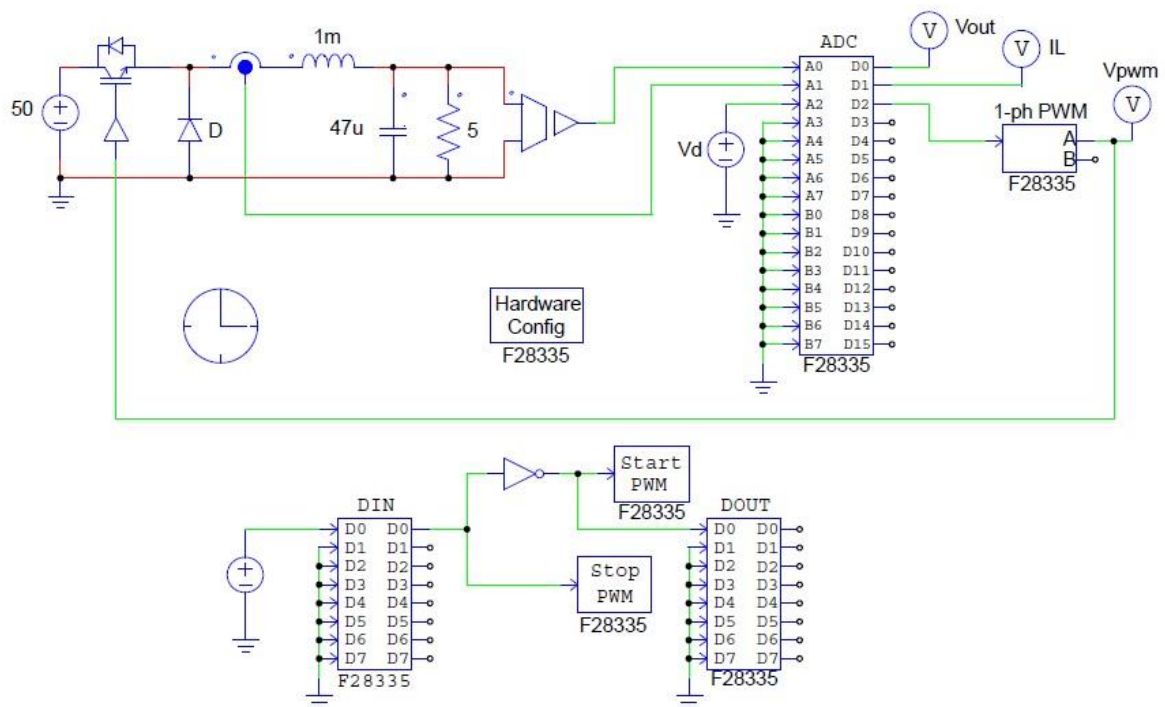
5. APLICAÇÕES DO SIMCODER EM CIRCUITOS DE ELETRÔNICA DE POTÊNCIA

Depois de realizado o estudo sobre os conversores estáticos, é possível realizar simulações no PSIM utilizando os elementos que possibilitam a geração de código em C.

5.1. CONVERSOR BUCK

Para a simulação do conversor Buck foi utilizado os elementos da Figura 67 em que podem ser vistos um conversor A/D, um modulador de PWM Monofásico, uma entrada e uma saída digital, um *Start PWM* e um *Stop PWM*.

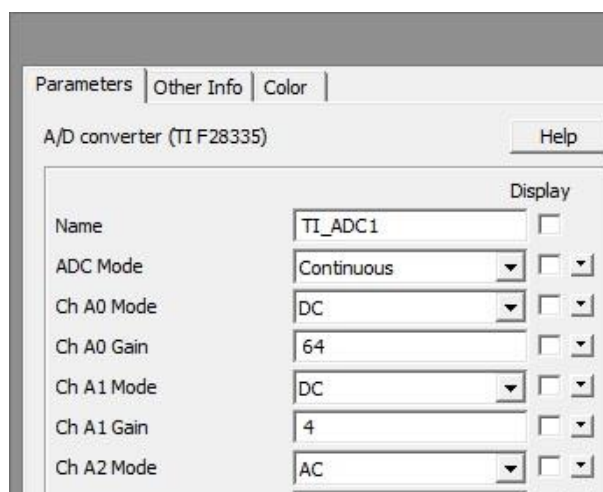
Figura 67 – Simulação e configurações do DSC para o conversor Buck simulado em malha aberta.



Fonte: Produção do próprio autor.

As funções do Conversor A/D no circuito são de obter os valores dos sensores de corrente e de tensão, e de obter o valor de referência para a razão cíclica do bloco de PWM, a partir de um sinal analógico. O modulador de PWM tem a função de controlar a chave do conversor. Observa-se que os sensores de tensão e de corrente foram determinados com um ganho de $1/64$ e $1/4$, respectivamente, para que o valor de entrada do Conversor A/D esteja abaixo de 3,0 V, ou seja, dentro da faixa de tensão permitida para o modo DC do conversor. Na Figura 68 é mostrada uma parte da janela de configurações do Conversor A/D.

Figura 68 - Janela de configurações do conversor A/D utilizado na simulação do conversor Buck em malha aberta.

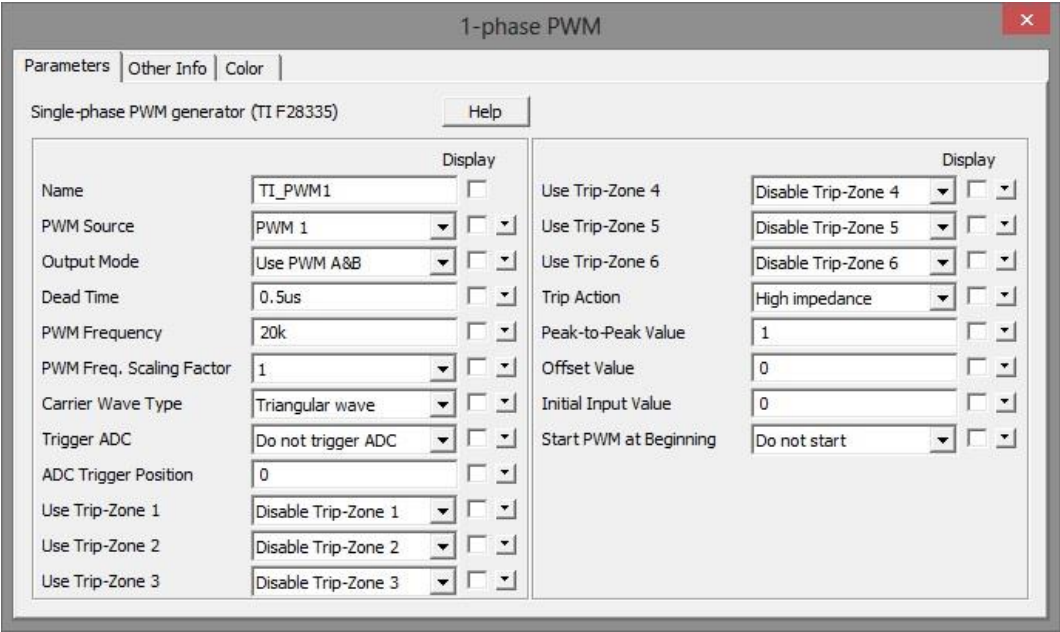


Fonte: Produção do próprio autor.

O conversor A/D foi configurado com ganho de 64 e 4 para os canais A0 e A1, respectivamente, para compensar a redução dos sensores de tensão e de corrente. Para o canal A2, foi utilizado ganho unitário e para todos os canais utilizados foram utilizados no modo DC. O bloco de PWM foi configurado conforme visto na Figura 69.

Os blocos de entrada digital, *Start* e *Stop* PWM constituem uma lógica para ativar e desativar o bloco de PWM, através de um botão que pode ser ligada ao DSC. A saída digital simplesmente indica se o bloco de PWM está ligado ou não, podendo ser visualizado através de um LED.

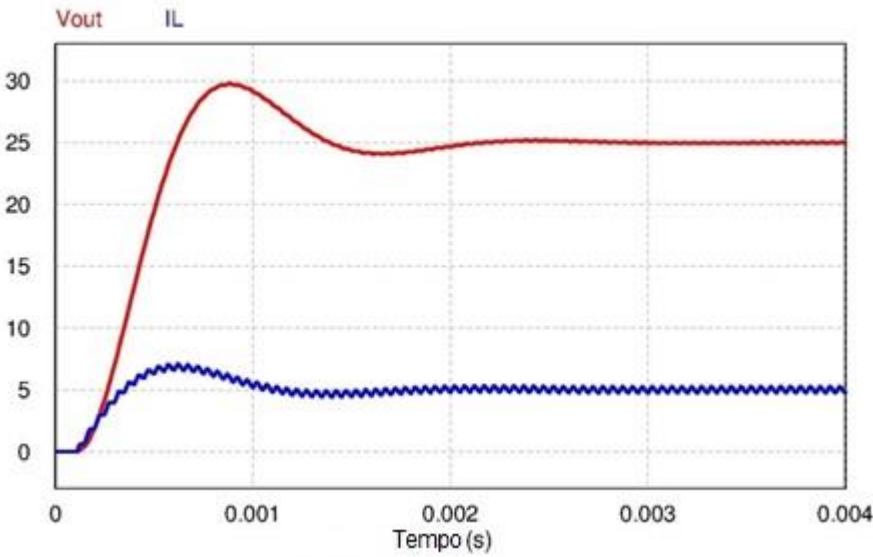
Figura 69 - Configurações do bloco de PWM utilizado para o conversor Buck.



Fonte: Produção do próprio autor.

A partir do circuito simulado, obtêm-se as formas de onda do conversor Buck em malha aberta, vistas na Figura 70.

Figura 70 - Formas de ondas de saída do conversor Buck em malha aberta.



Fonte: Produção do próprio autor.

A partir da simulação realizada, pode-se gerar o código em linguagem C para aplicação no DSC da *Texas Instruments*, obtendo-se como resultado o código abaixo.

```

/*****
// This code is created by SimCoder Version 2.0 for F28335 Hardware Target
//
// SimCoder is copyright by Powersim Inc., 2009
//
// Date: July 30, 2014 11:17:41
*****/
#include <math.h>
#include "PS_bios.h"
typedef float DefaultType;
#define GetCurTime() PS_GetSysTimer()

interrupt void Task();
void Task_1();

DefaultType fGblVadc2 = 0.0;
DefaultType fGblVadc1 = 0.0;

interrupt void Task()
{
    DefaultType fTI_ADC1_2;
    PS_EnableIntr();

    fTI_ADC1_2 = PS_GetDcAdc(2);
    PS_SetPwm1Rate(fTI_ADC1_2);
    PS_ExitPwm1General();
}

void Task_1()
{
    DefaultType fTI_DIN1, fNOT1, fTI_ADC1_1, fTI_ADC1;

    fTI_DIN1 = (PS_GetDigitInA() & ((Uint32)1 << 10)) ? 1 : 0;
    fNOT1 = !fTI_DIN1;
    if (fNOT1 > 0)
    {
        PS_StartPwm(1);
    }
    if (fTI_DIN1 != 0)
    {
        PS_StopPwm(1);
    }
    fTI_ADC1_1 = PS_GetDcAdc(1);
#ifdef _DEBUG
    fGblVadc2 = fTI_ADC1_1;
#endif
    fTI_ADC1 = PS_GetDcAdc(0);
#ifdef _DEBUG
    fGblVadc1 = fTI_ADC1;
#endif
}

```

```

#endif
    (fNOT1 == 0) ? PS_ClearDigitOutBitA((Uint32)1 << 11) :
PS_SetDigitOutBitA((Uint32)1 << 11);
}

void Initialize(void)
{
    PS_SysInit(30, 10);
    PS_StartStopPwmClock(0);
    PS_InitTimer(0, 0xffffffff);
    PS_InitPwm(1, 1, 10000*1, (4e-6)*1e6, PWM_POSI_ONLY, 157);    // pwnNo,
waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(1, 1, 0, 1.0/1);
    PS_SetPwmIntrType(1, ePwmNoAdc, 1, 0);
    PS_SetPwmVector(1, ePwmNoAdc, Task);
    PS_SetPwm1Rate(0);
    PS_StopPwm(1);

    PS_ResetAdcConvSeq();
    PS_SetAdcConvSeq(eAdcCascade, 0, 4);
    PS_SetAdcConvSeq(eAdcCascade, 1, 4);
    PS_SetAdcConvSeq(eAdcCascade, 2, 1.0);
    PS_AdcInit(0, !0);

    PS_InitDigitIn(10, 100);

    PS_InitDigitOut(11);

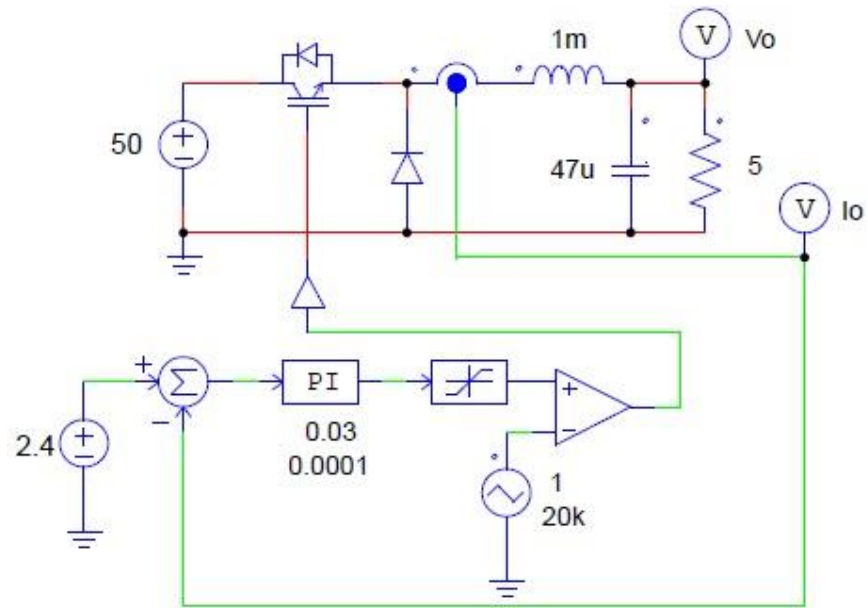
    PS_StartStopPwmClock(1);
}

void main()
{
    Initialize();
    PS_EnableIntr();    // Enable Global interrupt INTM
    PS_EnableDbgm();
    for (;;) {
        Task_1();
    }
}

```

A simulação acima foi feita em malha aberta. Porém, com o modelo obtido na seção anterior, pode-se projetar um controlador com a finalidade de obter-se alguma característica desejada, ou eliminar alguma característica indesejada. Na Figura 71 observa-se o conversor Buck em malha fechada, onde o controlador PI apresenta ganho de 0,03 e constante de tempo de 0,0001s.

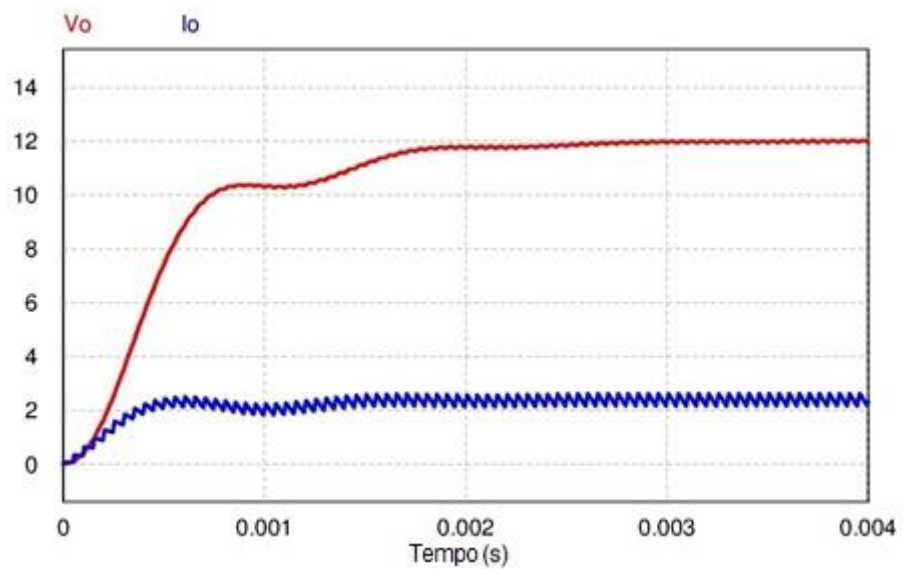
Figura 71 - Conversor Buck em malha fechada com controlador PI.



Fonte: Produção do próprio autor.

As formas de onda da tensão e da corrente são mostradas na Figura 72.

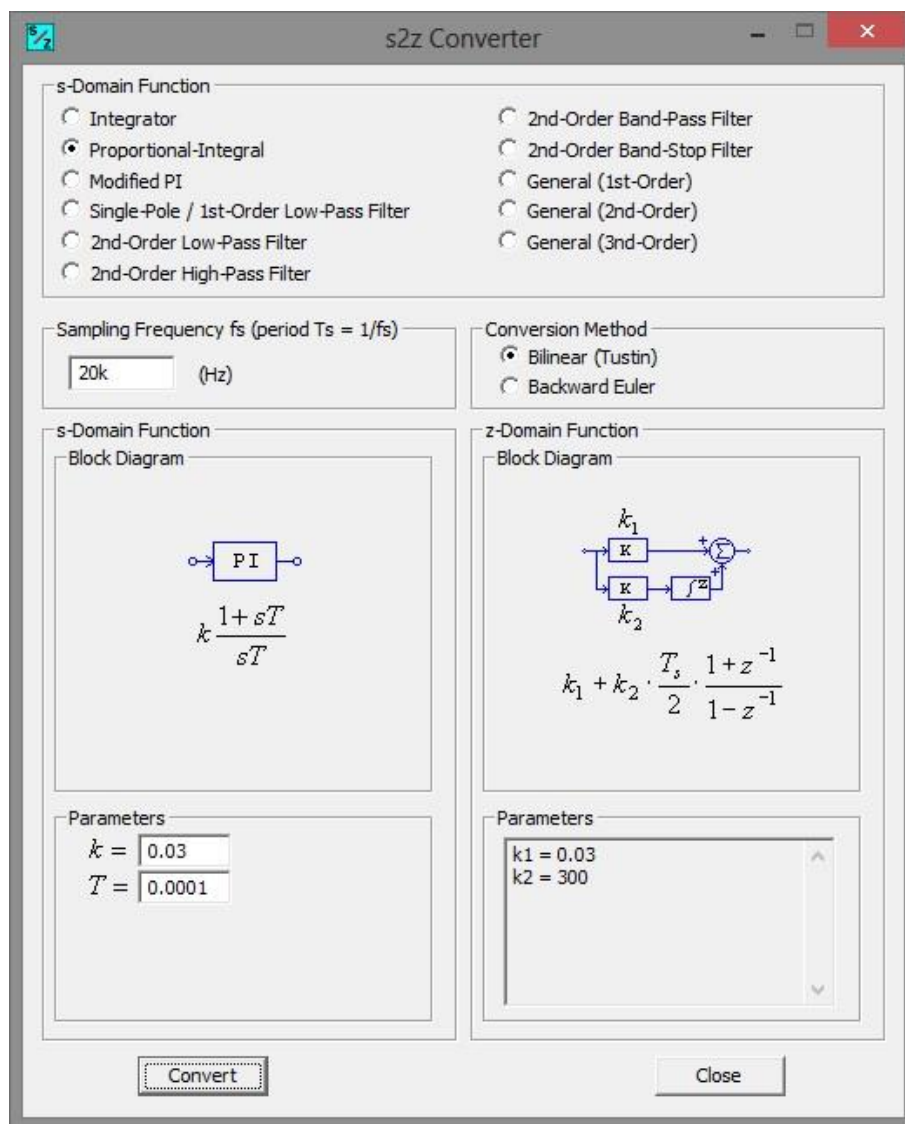
Figura 72 - Formas de onda de saída do conversor Buck em malha fechada com controlador PI.



Fonte: Produção do próprio autor.

O controlador PI mostrado na Figura 71 está representado no domínio da frequência, portanto deve antes ser convertido para o domínio discreto (z) para poder ser utilizado pela ferramenta *SimCoder* gerar o código em linguagem C. Uma maneira de fazer a conversão é utilizar a ferramenta *s2z Converter*, do PSIM®, encontrado no menu *Utilities*. Esta ferramenta é utilizada para se obter os valores discretizados do controlador. Na Figura 73 é mostrada a ferramenta, apresentando os valores encontrados.

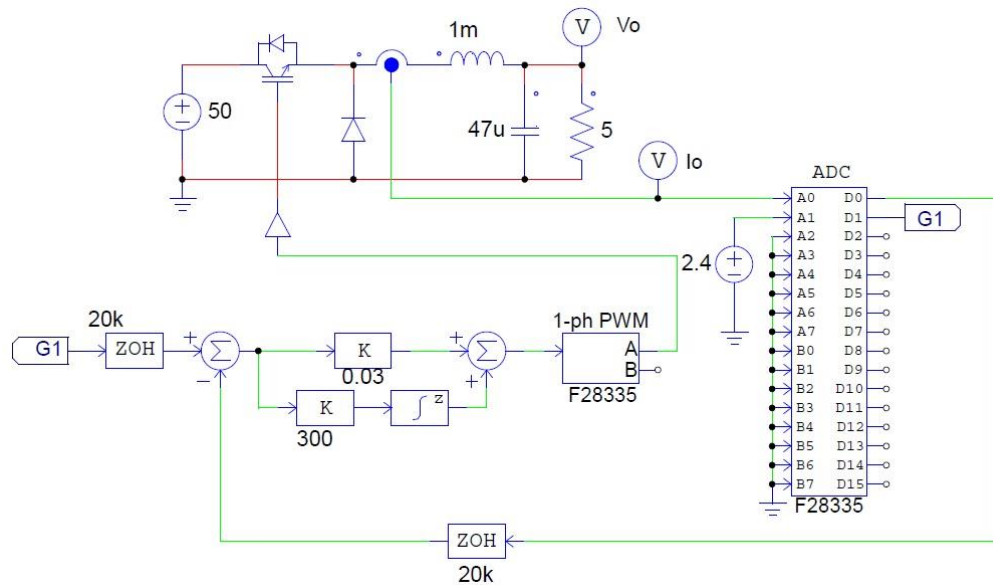
Figura 73 - Ferramenta *s2z Converter* do PSIM®.



Fonte: PSIM®.

A ferramenta *s2z Converter* já apresenta o circuito equivalente ao controlador PI, dando os valores do ganho proporcional e do ganho integral. Com os valores obtidos, pode-se simular o controlador em conjunto com os elementos do *SimCoder*, sendo possível a geração do código em linguagem C. Na Figura 74 é mostrado o conversor Buck em malha fechada com o controlador PI obtido.

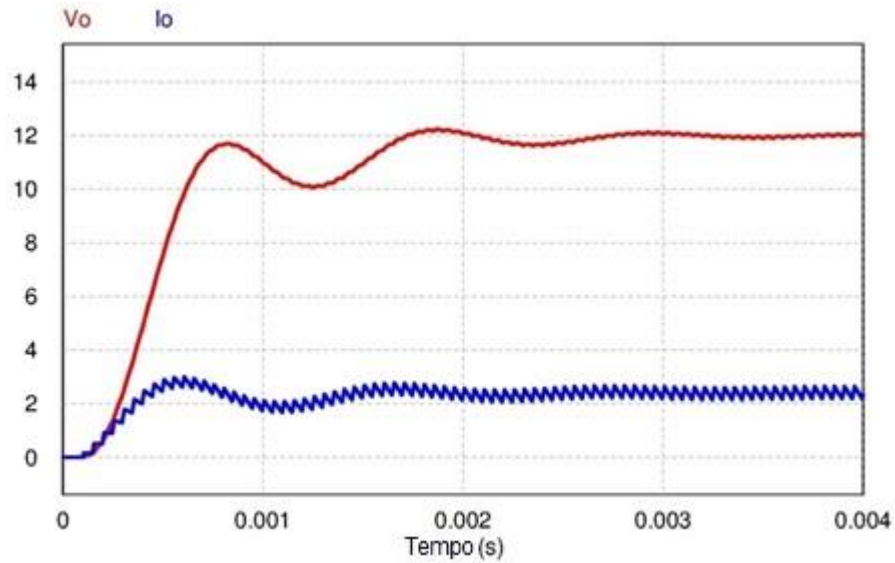
Figura 74 - Conversor Buck com controlador PI discretizado.



Fonte: Produção do próprio autor.

Observa-se que, por não se trabalhar mais com tempo contínuo, deve-se utilizar retentores de ordem zero, ou *Zero-Order Hold* (ZOH), após as leituras analógicas. Na Figura 75 são mostradas as formas de onda de saída do conversor Buck utilizando o controlador discretizado obtido. Observa-se que esta forma de onda, assim como para o controlador analógico, não apresenta um valor de ultrapassagem, possuindo um tempo de acomodação próximo.

Figura 75 - Formas de onda de saída do conversor Buck com controlador discretizado.



Fonte: Produção do próprio autor.

Abaixo é mostrada a parte do código correspondente ao controlador simulado.

```
interrupt void Task()
{
    DefaultType fTI_ADC1_1, fZOH2, fTI_ADC1, fZOH1, fSUM1, fP1, fP2, fB5,
fSUMP1;
    PS_EnableIntr();

    fTI_ADC1_1 = PS_GetDcAdc(1);
    fTI_ADC1 = PS_GetDcAdc(0);
    fZOH2 = fTI_ADC1_1;
    fZOH1 = fTI_ADC1;
    fSUM1 = fZOH2 - fZOH1;
    fP1 = fSUM1 * 0.03;
    fP2 = fSUM1 * 300;
    {
        static DefaultType out_A = 0, in_A = 0.0;
        fB5 = out_A + 0.5/20000 * (fP2 + in_A);
        out_A = fB5; in_A = fP2;
    }
    fSUMP1 = fP1 + fB5;
    PS_SetPwm1Rate(fSUMP1);
    PS_ExitPwm1General();
}
```

Outra opção é discretizar o controlador PI utilizando o *software* MATLAB®. O modelo do controlador é encontrado em suas configurações, pressionando o botão *Help*. O controlador PI apresenta a seguinte função de transferência no domínio s :

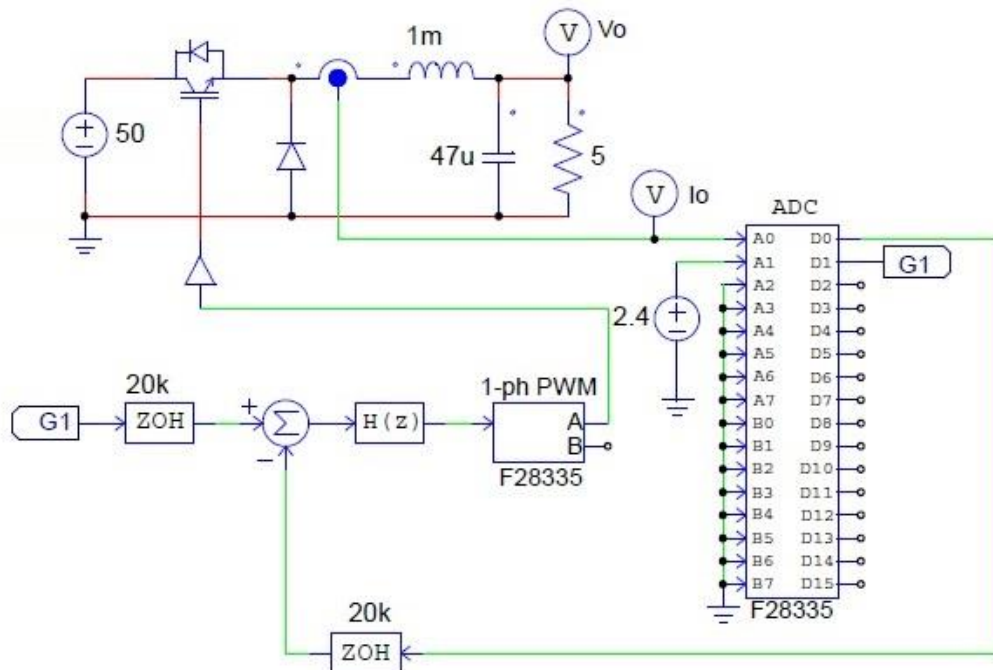
$$G(s) = k * \frac{1 + sT}{sT} \quad (11)$$

Onde k é o ganho e T é a constante de tempo do controlador PI. Aplicando-se os valores do controlador e utilizando o *software* MATLAB®, foi obtida a função de transferência no domínio z , através da função *c2d*, com tempo de amostragem de 0,05ms e utilizando o método de Tustin.

$$G(z) = \frac{0,0375z + 0,0225}{z - 1} \quad (12)$$

Na Figura 76 é mostrado o circuito utilizando o bloco de função de transferência no domínio z do PSIM® com os parâmetros da função de transferência obtida no MATLAB®.

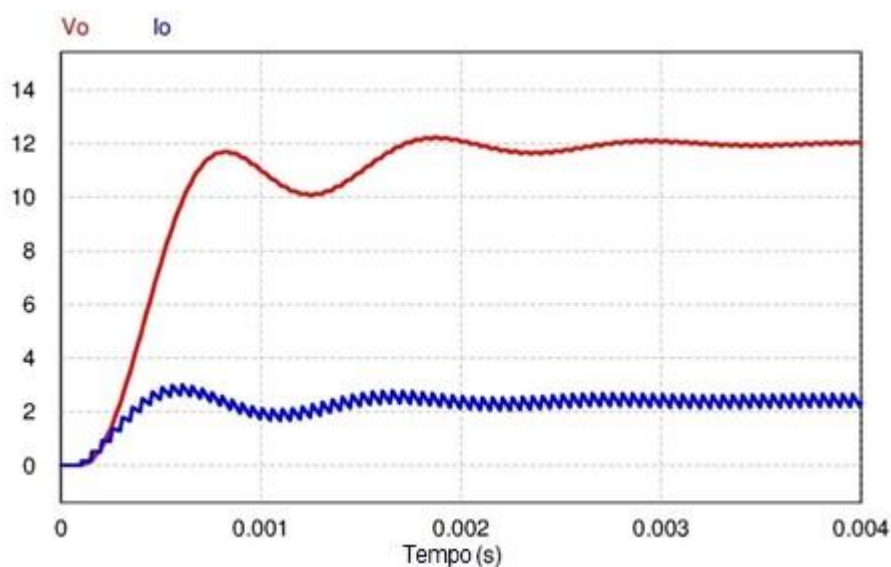
Figura 76 - Conversor Buck com controlador na forma de função de transferência no domínio z .



Fonte: Produção do próprio autor.

As formas de onda de saída do conversor em malha fechada com o controlador na forma de função de transferência no domínio z obtidas são iguais às formas de onda encontradas com o controlador discretizado, conforme mostrado na Figura 77.

Figura 77 - Formas de onda de saída do conversor Buck com controlador em função de transferência no domínio z.



Fonte: Produção do próprio autor.

A partir da simulação, é possível gerar o código para o DSC. No quadro abaixo é mostrado parte deste código, correspondente ao controlador simulado.

```
interrupt void Task()
{
    DefaultType fTI_ADC1_1, fZOH2, fTI_ADC1, fZOH1, fSUM1, fTF_D2;
    PS_EnableIntr();

    fTI_ADC1_1 = PS_GetDcAdc(1);
    fTI_ADC1 = PS_GetDcAdc(0);
    fZOH2 = fTI_ADC1_1;
    fZOH1 = fTI_ADC1;
    fSUM1 = fZOH2 - fZOH1;
    {
        static DefaultType fIn = 0.0;
        static DefaultType fOut = 0.0;
        fTF_D2 = 0.0375 * fSUM1 + (-0.0225) * fIn - (-1) * fOut;
        fIn = fSUM1;
        fOut = fTF_D2;
    }
    PS_SetPwm1Rate(fTF_D2);
}
```

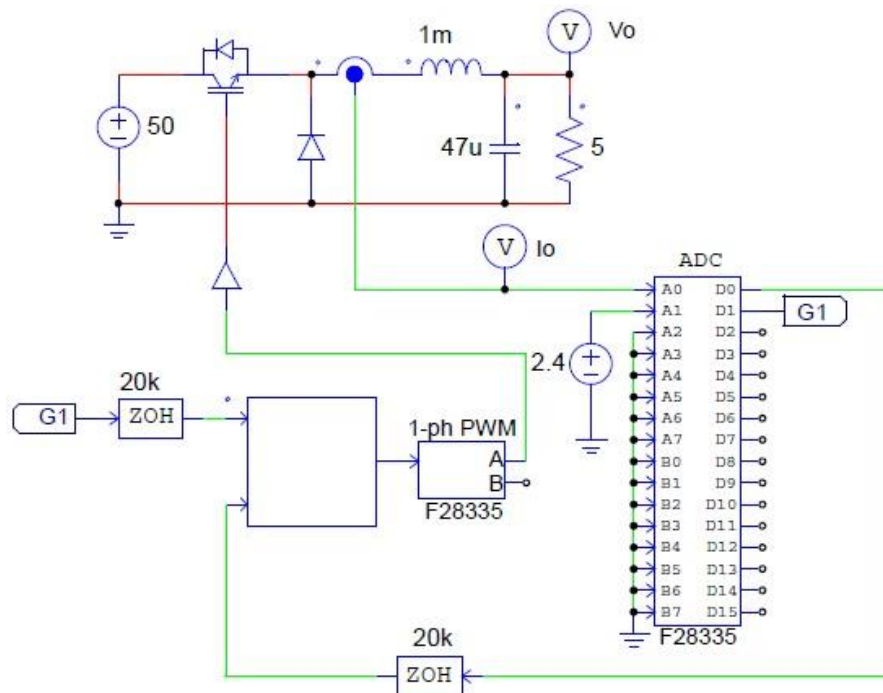
```

PS_ExitPwm1General();
}

```

Há ainda a alternativa de representar o controlador digital em equações de diferenças, escrevendo-as em um bloco de linguagem C do PSIM[®]. Na Figura 78 é mostrado o circuito simulado com o bloco de linguagem C simples.

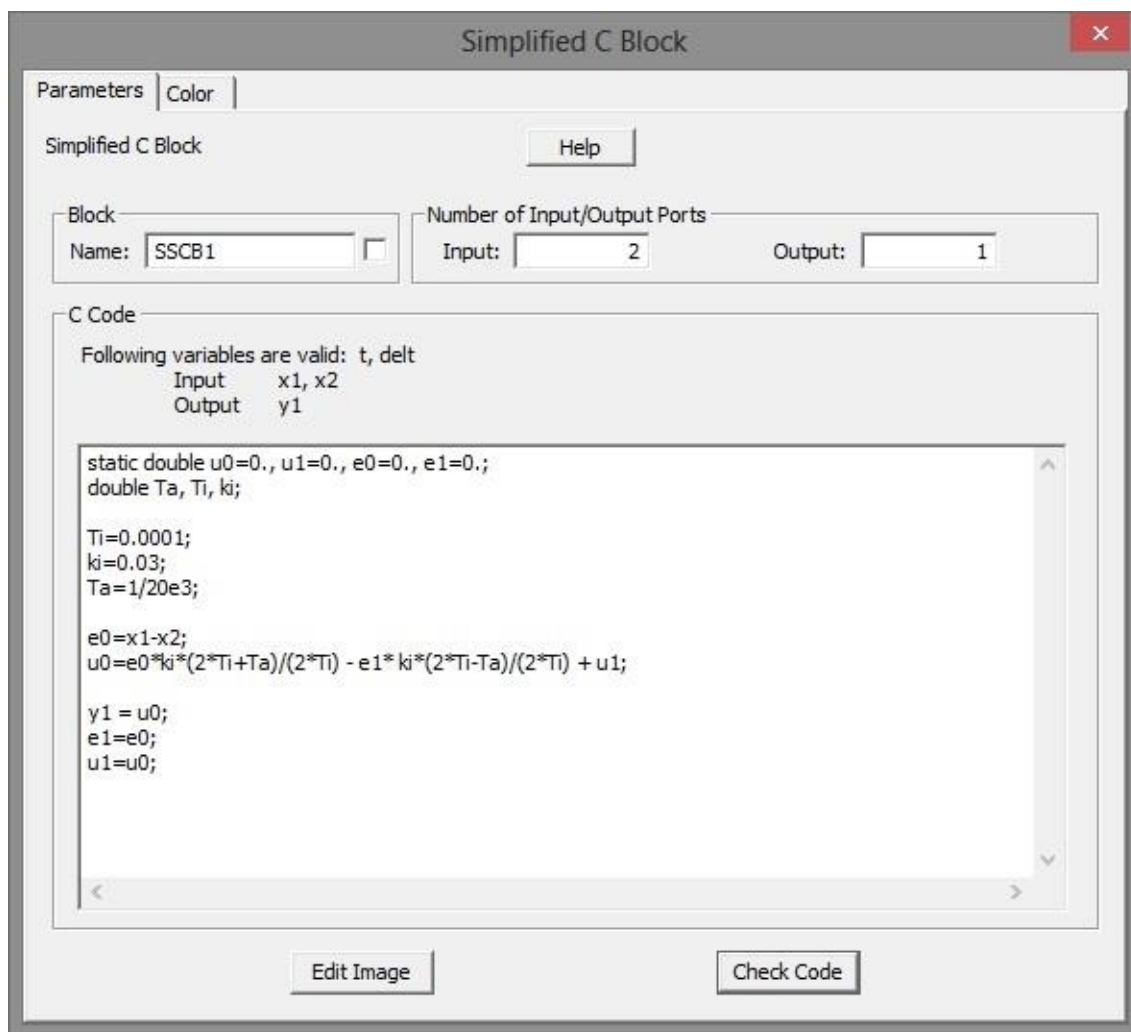
Figura 78 - Conversor Buck em malha fechada com controlador digital utilizando o bloco de linguagem C.



Fonte: Produção do próprio autor.

O código feito pode ser observado na Figura 79. Observa-se que na simulação não foi utilizado um subtrator para realizar a alimentação negativa. Esta operação foi feita pelo código, sendo representada pela variável $e0$. Os parâmetros k_i e T_i são o ganho e a constante de tempo do controlador PI, respectivamente. T_a é o período de amostragem, sendo o inverso da frequência de amostragem.

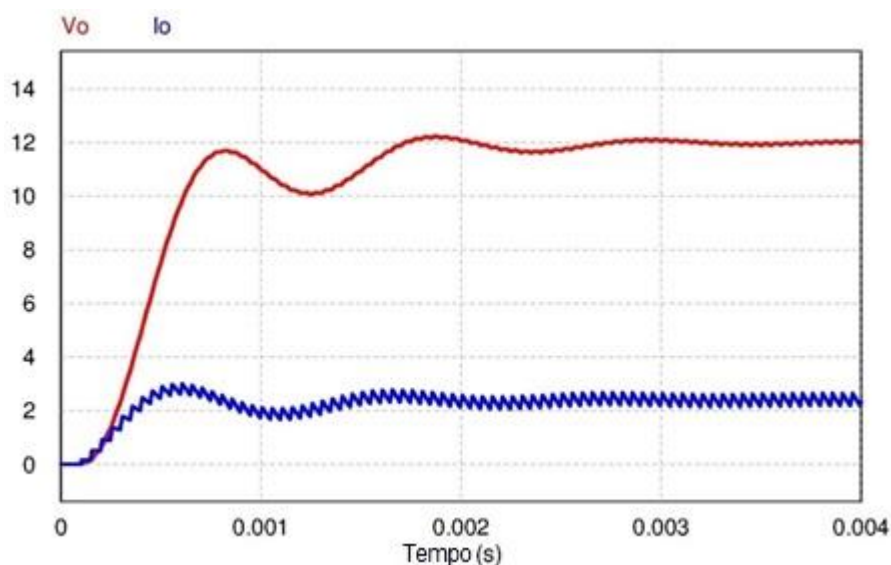
Figura 79 - Bloco de linguagem C simples com código escrito para o controle digital.



Fonte: PSIM®.

Na Figura 80 são mostradas as formas de onda de saída do conversor simulado com o bloco em linguagem C simples. Observa-se que o resultado verificado foi igual aos outros métodos.

Figura 80 - Formas de onda de saída do conversor Buck com controlador digital utilizando o bloco de linguagem C simples.



Fonte: Produção do próprio autor.

O código gerado pelo *SimCoder* mantém o código escrito para o controlador, conforme mostrado abaixo.

```
interrupt void Task()
{
    DefaultType fTI_ADC1_1, fZOH2, fTI_ADC1, fZOH1;
    PS_EnableIntr();

    fTI_ADC1_1 = PS_GetDcAdc(1);
    fTI_ADC1 = PS_GetDcAdc(0);
    fZOH2 = fTI_ADC1_1;
    fZOH1 = fTI_ADC1;
    {
        static double u0=0., u1=0., e0=0., e1=0.;
        double Ta, Ti, ki;

        Ti=0.0001;
        ki=0.03;
        Ta=1/20e3;

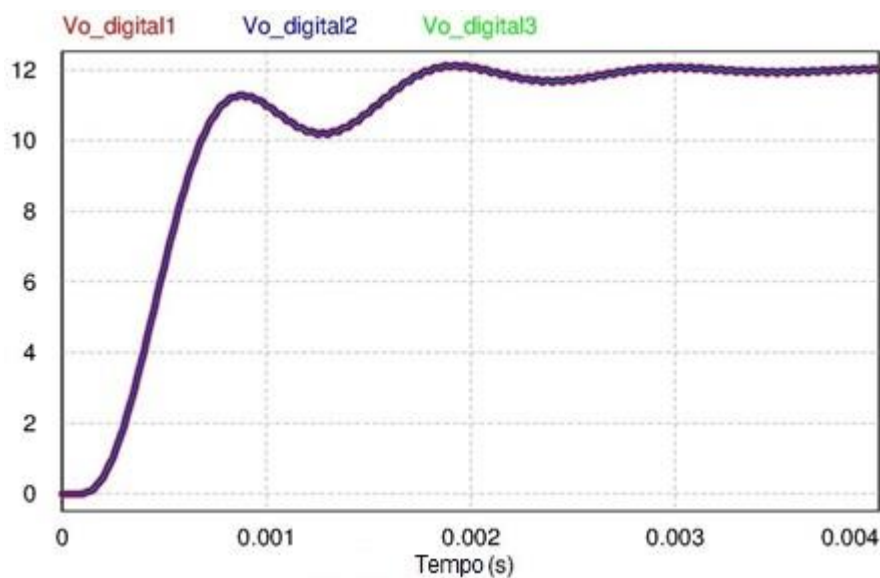
        e0=fZOH2-fZOH1;
        u0=e0*ki*(2*Ti+Ta)/(2*Ti) - e1* ki*(2*Ti-Ta)/(2*Ti) + u1;

        fGblSSCB1 = u0;
        e1=e0;
        u1=u0;
    }
    PS_ExitTimer1Intr();
}
```

```
}
```

Na Figura 81 são mostradas as formas de onda de saída do conversor Buck com os controladores dos três métodos utilizados para se gerar código. Observa-se que as formas de onda são iguais.

Figura 81 - Formas de onda de saídas do conversor Buck em malha fechada para os três métodos de controle digital.

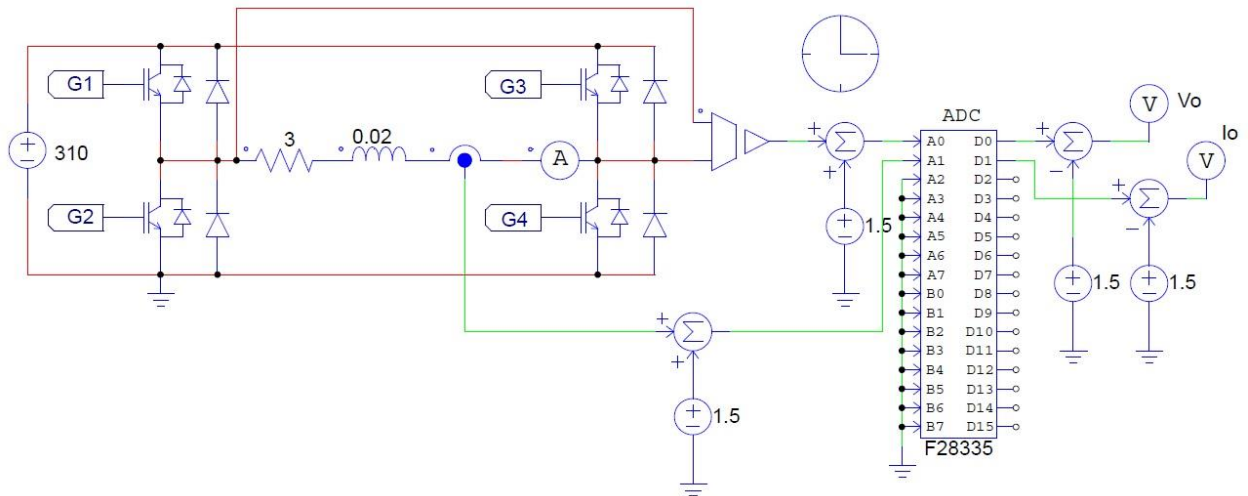


Fonte: Produção do próprio autor.

5.2. INVERSOR PONTE COMPLETA

Na simulação do inversor de ponte completa foi utilizado um conversor A/D, com o objetivo de obter as leituras da tensão e da corrente na carga. Para isto, foram utilizados sensores de tensão e de corrente, com ganhos de 1/256 e de 1/64, respectivamente. Foram utilizados os canais A0 para a tensão e A1 para a corrente, ambos com ganhos unitários. Para os dois canais foram adicionados um nível de *offset* de 1,5V, para permitir que o conversor A/D esteja no modo DC. O inversor simulado pode ser visto na Figura 82.

Figura 82 – Circuito da simulação e configurações do DSC para o inversor ponte completa.

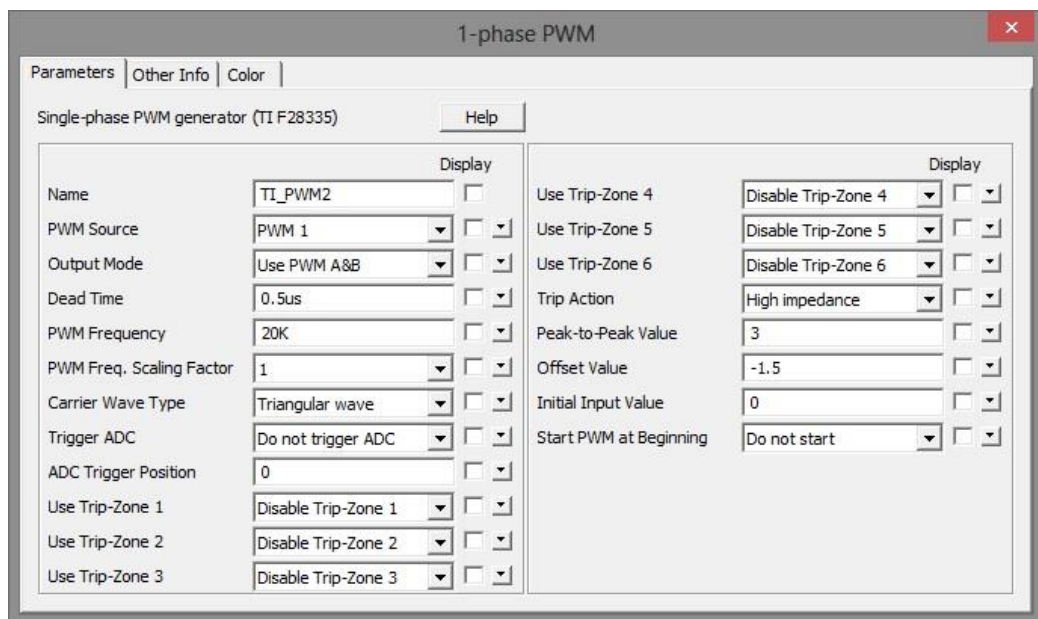


Fonte: Produção do próprio autor.

Observa-se que não foi utilizado nenhum filtro entre as saídas dos sensores e o Conversor A/D. Poderiam ser utilizados filtros para ter como saída uma forma de onda senoidal para a tensão, para tornar a simulação mais real.

Na Figura 83 são mostrados os parâmetros configurados do bloco PWM.

Figura 83 - Janela de configurações do bloco de PWM utilizado na simulação do inversor ponte completa.

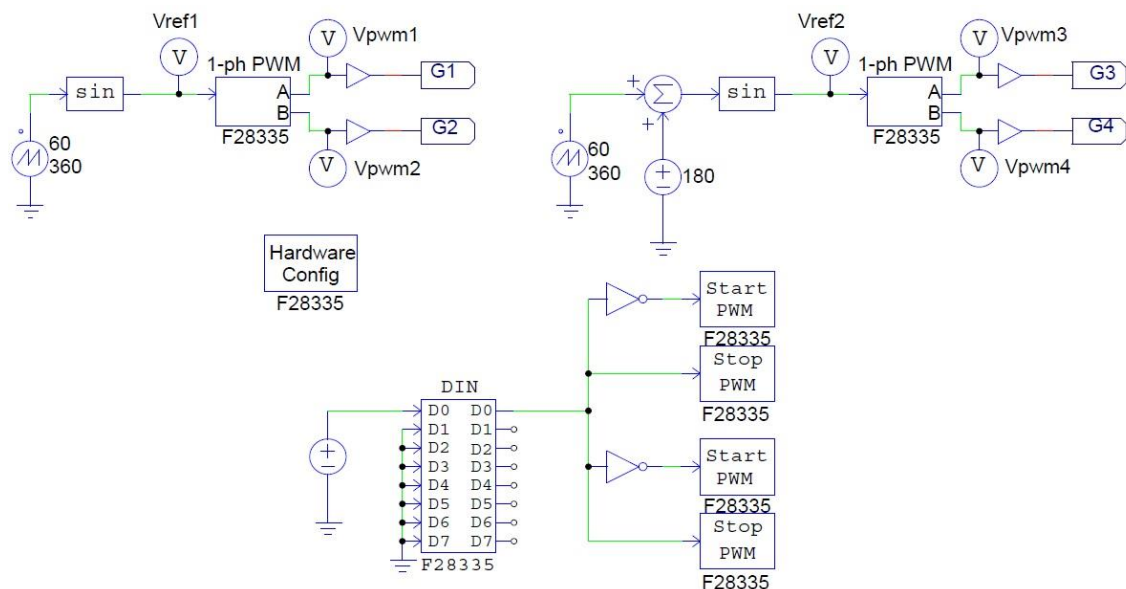


Fonte: PSIM®.

Para comandar os interruptores do Inversor Ponte Completa, foram utilizados dois blocos de PWM monofásicos, utilizando os canais PWM1 e PWM2. Ambos foram configurados para 20kHz de frequência, 0,5 μ s de tempo morto, 3V de valor pico-a-pico, -1.5V de valor de *offset* e para não iniciarem ativados. Como entrada para os moduladores de PWM, foram utilizadas duas fontes de dente de serra, de frequência 60Hz e valor máximo de 360V, e uma função seno, obtendo-se uma onda senoidal variando de -1 a 1V. Para o segundo bloco, o sinal dente-de-serra foi configurado com um *offset* de 180, possibilitando que a senoidal de referência seja defasada em 180° da senoidal utilizada para o primeiro modulador PWM.

Foram utilizados blocos *Start PWM* e *Stop PWM* para acionar os geradores, da mesma forma como foi feito para o conversor Buck, utilizando um bloco de entradas digitais. Na Figura 84 podem-se observar os componentes mencionados.

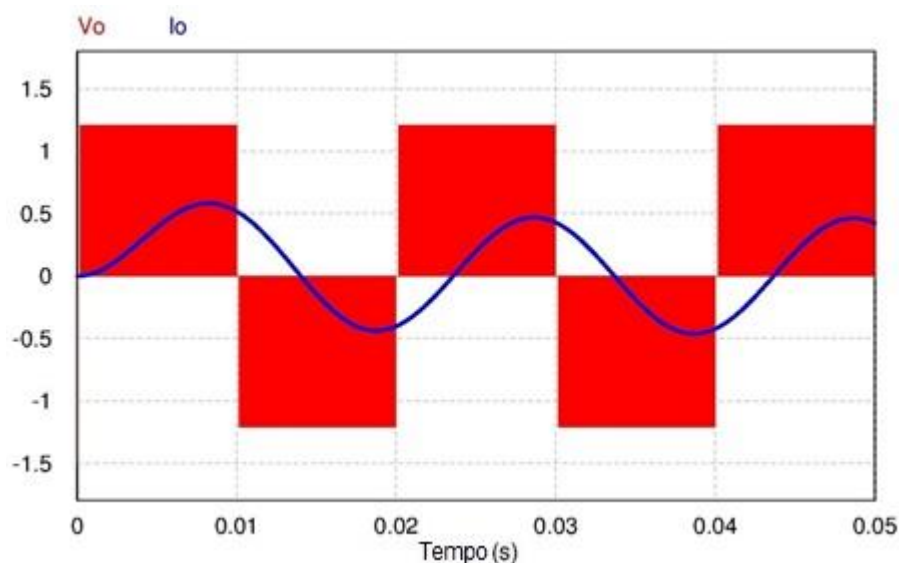
Figura 84 – Circuito de Comando do inversor ponte completa



Fonte: Produção do próprio autor.

As formas de onda obtidas são mostradas na Figura 85.

Figura 85 - Tensão e corrente de saída do inversor ponte completa



Fonte: Produção do próprio autor.

A partir da simulação realizada, foi gerado o código em linguagem C mostrado a seguir.

```

/*****
// This code is created by SimCoder Version 9.1 for TI F28335 Hardware Target
//
// SimCoder is copyright by Powersim Inc., 2009-2011
//
// Date: November 02, 2014 17:02:25
*****/
/
#include <math.h>
#include "PS_bios.h"
typedef float DefaultType;
#define GetCurTime() PS_GetSysTimer()

interrupt void Task();
void Task_1();

DefaultType fGblV13 = 0.0;
DefaultType fGblV17 = 0.0;
DefaultType fGblVo = 0.0;
DefaultType fGblIo = 0.0;
typedef struct {
    unsigned long tmLow;
    unsigned long tmHigh;
} _CBigTime;

_CBigTime GetBigTime(void)

```

```

{
    static _CBigTime tm = {0,0};
    unsigned long curTime = GetCurTime();
    if (curTime < tm.tmLow)
        tm.tmHigh++;
    tm.tmLow = curTime;
    return tm;
}

interrupt void Task()
{
    DefaultType fVSAW4, fVDC6, fSUMP1, fSIN3, fVSAW2, fSIN1;
    PS_EnableIntr();

    {
        static unsigned long period = (unsigned long)(15000000L / 50);
        static float fPeriod = ((float)50) / 15000000L;
        static _CBigTime tmCarrierStart = {0, 0};
        _CBigTime tm = GetBigTime();
        unsigned long tmp1, tmp2;
        tmp1 = tm.tmLow - tmCarrierStart.tmLow;
        tmp2 = tm.tmHigh - tmCarrierStart.tmHigh;
        if (tm.tmLow > tmCarrierStart.tmLow)
            tmp2++;
        if (tmp2 || (!tmp2 && (tmp1 >= period))) {
            tmp1 = tmCarrierStart.tmLow + period;
            if ((tmp1 < tmCarrierStart.tmLow) || (tmp1 < period))
                tmCarrierStart.tmHigh++;
            tmCarrierStart.tmLow = tmp1;
        }
        tmp1 = tm.tmLow - tmCarrierStart.tmLow;
        fVSAW4 = 360 * tmp1 * fPeriod;
    }
    fVDC6 = 180;
    fSUMP1 = fVSAW4 + fVDC6;
    fSIN3 = sin(fSUMP1 * (3.14159265 / 180.));
    PS_SetPwm2Rate(fSIN3);
    {
        static unsigned long period = (unsigned long)(15000000L / 50);
        static float fPeriod = ((float)50) / 15000000L;
        static _CBigTime tmCarrierStart = {0, 0};
        _CBigTime tm = GetBigTime();
        unsigned long tmp1, tmp2;
        tmp1 = tm.tmLow - tmCarrierStart.tmLow;
        tmp2 = tm.tmHigh - tmCarrierStart.tmHigh;
        if (tm.tmLow > tmCarrierStart.tmLow)
            tmp2++;
        if (tmp2 || (!tmp2 && (tmp1 >= period))) {
            tmp1 = tmCarrierStart.tmLow + period;
            if ((tmp1 < tmCarrierStart.tmLow) || (tmp1 < period))
                tmCarrierStart.tmHigh++;
            tmCarrierStart.tmLow = tmp1;
        }
        tmp1 = tm.tmLow - tmCarrierStart.tmLow;
        fVSAW2 = 360 * tmp1 * fPeriod;
    }
    fSIN1 = sin(fVSAW2 * (3.14159265 / 180.));
}

```

```

#ifdef _DEBUG
    fGblV13 = fSIN1;
#endif
    PS_SetPwm1Rate(fSIN1);
#ifdef _DEBUG
    fGblV17 = fSIN3;
#endif
    PS_ExitPwm2General();
}

void Task_1()
{
    DefaultType fTI_DIN1, fNOT5, fNOT9, fTI_ADC1, fVDC9, fSUM1, fTI_ADC1_1,
    fVDC10, fSUM2;

    fTI_DIN1 = (PS_GetDigitInA() & ((Uint32)1 << 5)) ? 1 : 0;
    fTI_ADC1 = PS_GetDcAdc(0);
    fTI_ADC1_1 = PS_GetDcAdc(1);
    fNOT5 = !fTI_DIN1;
    if (fNOT5 > 0)
    {
        PS_StartPwm(1);
    }
    if (fTI_DIN1 != 0)
    {
        PS_StopPwm(2);
    }
    fNOT9 = !fTI_DIN1;
    if (fNOT9 > 0)
    {
        PS_StartPwm(2);
    }
    if (fTI_DIN1 != 0)
    {
        PS_StopPwm(1);
    }
    fVDC9 = 1.5;
    fSUM1 = fTI_ADC1 - fVDC9;
#ifdef _DEBUG
    fGblVo = fSUM1;
#endif
    fVDC10 = 1.5;
    fSUM2 = fTI_ADC1_1 - fVDC10;
#ifdef _DEBUG
    fGblIo = fSUM2;
#endif
}

void Initialize(void)
{
    PS_SysInit(30, 10);
    PS_StartStopPwmClock(0);
    PS_InitTimer(0, 0xffffffff);
    PS_InitPwm(2, 1, 20000*1, (0.5e-6)*1e6, PWM_TWO_OUT, 34115);    // pwnNo,
waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(2, 3, (-1.5), 1.0/3);
    PS_SetPwmIntrType(2, ePwmNoAdc, 1, 0);
}

```

```

    PS_SetPwmVector(2, ePwmNoAdc, Task);
    PS_SetPwm2Rate(0);
    PS_StopPwm(2);

    PS_InitPwm(1, 1, 20000*1, (0.5e-6)*1e6, PWM_TWO_OUT, 34115);    // pwnNo,
waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(1, 3, (-1.5), 1.0/3);
    PS_SetPwmIntrType(1, ePwmNoAdc, 1, 0);
    PS_SetPwm1Rate(0);
    PS_StopPwm(1);

    PS_ResetAdcConvSeq();
    PS_SetAdcConvSeq(eAdcCascade, 0, 1);
    PS_SetAdcConvSeq(eAdcCascade, 1, 1);
    PS_AdcInit(0, !0);

    PS_InitDigitIn(5, 100);

    PS_StartStopPwmClock(1);
}

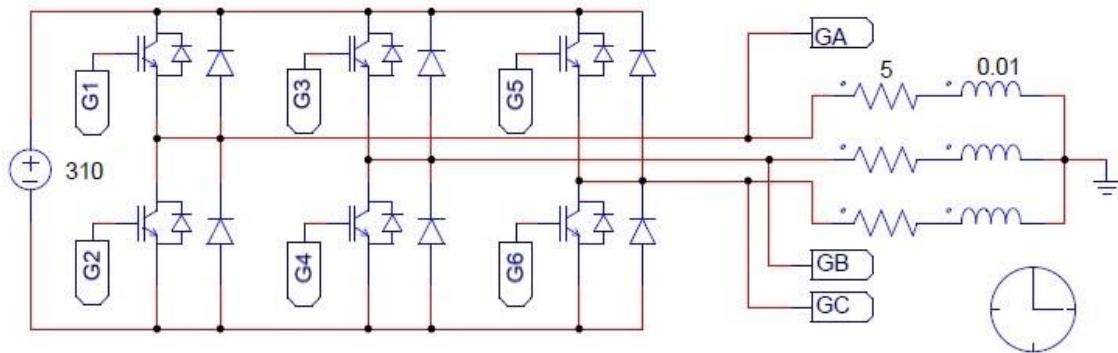
void main()
{
    Initialize();
    PS_EnableIntr();    // Enable Global interrupt INTM
    PS_EnableDbgm();
    for (;;) {
        Task_1();
    }
}

```

5.3. INVERSOR TRIFÁSICO

Para a simulação do inversor trifásico, foi utilizado o circuito apresentado na Figura 86, onde a fonte V_{in} é de 24V e a carga trifásica apresenta $R1 = R2 = R3 = 5\Omega$ e $L1 = L2 = L3 = 0,01H$. Para se medir a saída do inversor, foram utilizados três canais de um conversor A/D, medindo as tensões de fase na carga trifásica. Os sensores apresentam ganhos de 1/16, para que o valor de tensão após os filtros passa-bandas esteja entre -1,5V e 1,5V do modo AC do conversor A/D. Porém, são adicionados níveis de offset de 1,5V para utilizar o conversor A/D no modo DC. Para compensar este ganho dos sensores, o conversor A/D foi configurado para ter um ganho de 16V para os três canais utilizados.

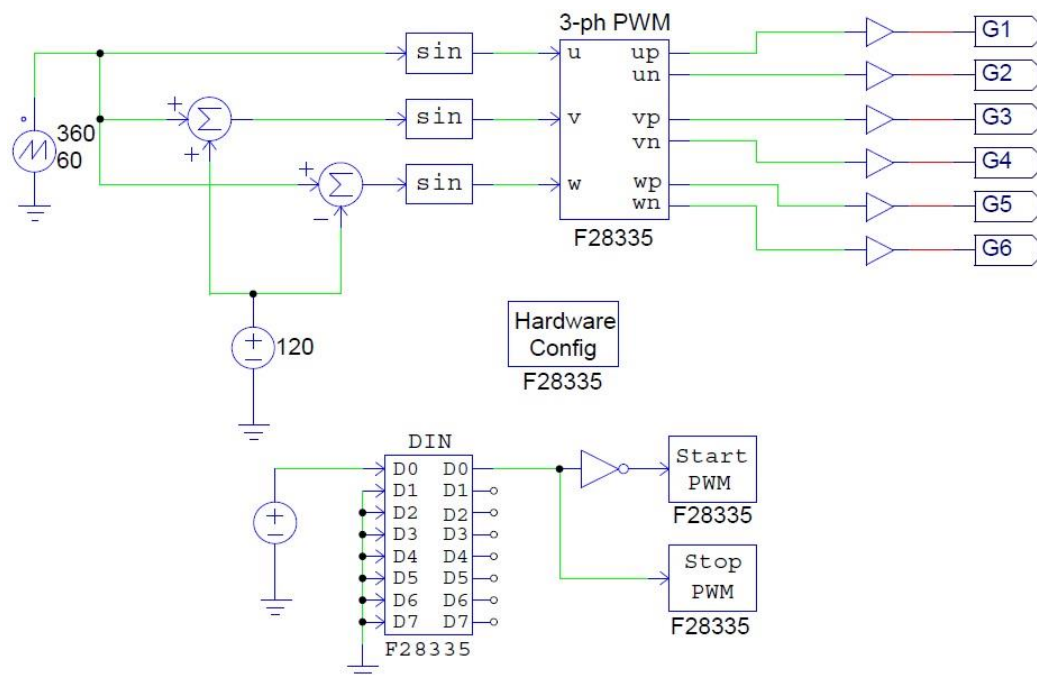
Figura 86 - Circuito de potência do inversor trifásico.



Fonte: Produção do próprio autor.

No comando deste inversor foi utilizado um bloco de PWM Trifásico, que foi configurado para $0,5\mu s$ de tempo morto, 20kHz de frequência, tensão de pico da portadora de 3V e -1,5V de *offset*. Como sinais de referência, foram utilizados três sinais senoidais, de amplitude de 1V, defasados em 120° entre si. Na Figura 87, pode-se observar o gerador de PWM e um circuito para ativar o gerador, semelhante ao apresentado para o conversor Buck.

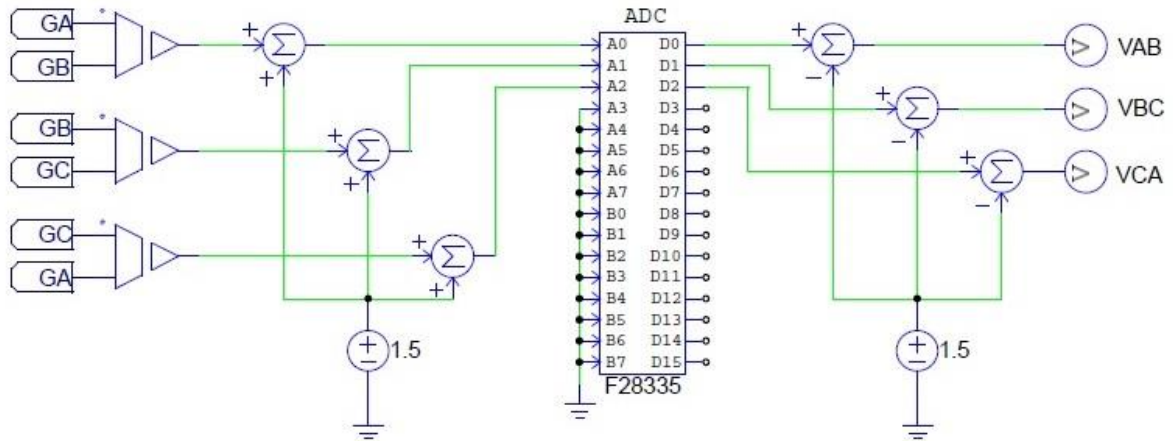
Figura 87 – Circuito de comando do inversor trifásico.



Fonte: Produção do próprio autor.

Na Figura 88 é mostrado o circuito das leituras analógicas através do Conversor A/D.

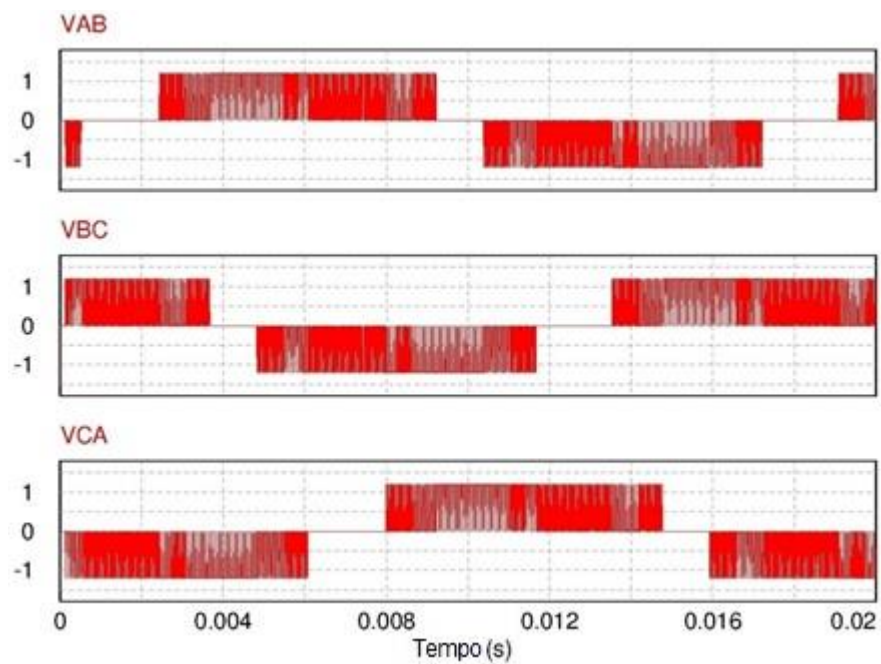
Figura 88 - Conversor A/D utilizado para o inversor trifásico.



Fonte: Produção do próprio autor.

As formas de ondas das tensões lidas pelo conversor A/D podem ser vistas na Figura 89.

Figura 89 - Tensão de saída das três fases do inversor trifásico.



Fonte: Produção do próprio autor.

A partir do circuito simulado foi gerado o código em linguagem C visto a seguir.

```

/*****
// This code is created by SimCoder Version 9.1 for TI F28335 Hardware Target
//
// SimCoder is copyright by Powersim Inc., 2009-2011
//
// Date: November 02, 2014 17:42:06
*****/
/
#include    <math.h>
#include    "PS_bios.h"
typedef float DefaultType;
#define     GetCurTime() PS_GetSysTimer()

interrupt void Task();
void Task_1();

DefaultType fGblVAB = 0.0;
DefaultType fGblVBC = 0.0;
DefaultType fGblVCA = 0.0;
typedef struct {
    unsigned long tmLow;
    unsigned long tmHigh;
}
_CBigTime;

_CBigTime GetBigTime(void)
{
    static _CBigTime tm = {0,0};
    unsigned long curTime = GetCurTime();
    if (curTime < tm.tmLow)
        tm.tmHigh++;
    tm.tmLow = curTime;
    return tm;
}

interrupt void Task()
{
    DefaultType fVSAW1, fSIN1, fVDC1, fSUMP1, fSIN2, fSUM1, fSIN3;
    PS_EnableIntr();

    {
        static unsigned long period = (unsigned long)(150000000L / 60);
        static float fPeriod = ((float)60) / 150000000L;
        static _CBigTime tmCarrierStart = {0, 0};
        _CBigTime tm = GetBigTime();
        unsigned long tmp1, tmp2;
        tmp1 = tm.tmLow - tmCarrierStart.tmLow;
        tmp2 = tm.tmHigh - tmCarrierStart.tmHigh;
        if (tm.tmLow > tmCarrierStart.tmLow)
            tmp2++;
        if (tmp2 || (!tmp2 && (tmp1 >= period))) {
            tmp1 = tmCarrierStart.tmLow + period;
            if ((tmp1 < tmCarrierStart.tmLow) || (tmp1 < period))

```

```

        tmCarrierStart.tmHigh++;
        tmCarrierStart.tmLow = tmp1;
    }
    tmp1 = tm.tmLow - tmCarrierStart.tmLow;
    fVSAW1 = 360 * tmp1 * fPeriod;
}
fSIN1 = sin(fVSAW1 * (3.14159265 / 180.));
fVDC1 = 120;
fSUMP1 = fVSAW1 + fVDC1;
fSIN2 = sin(fSUMP1 * (3.14159265 / 180.));
fSUM1 = fVSAW1 - fVDC1;
fSIN3 = sin(fSUM1 * (3.14159265 / 180.));
PS_SetPwm3ph1Uvw(fSIN1, fSIN2, fSIN3);
PS_ExitPwm1General();
}

void Task_1()
{
    DefaultType fTI_ADC1, fVDC6, fSUM2, fTI_ADC1_1, fVDC8, fSUM3, fTI_ADC1_2,
fVDC9, fSUM4;
    DefaultType fTI_DIN1, fNOT1;

    fTI_ADC1 = PS_GetDcAdc(0);
    fTI_ADC1_1 = PS_GetDcAdc(1);
    fTI_ADC1_2 = PS_GetDcAdc(2);
    fTI_DIN1 = (PS_GetDigitInA() & ((Uint32)1 << 10)) ? 1 : 0;
    fVDC6 = 1.5;
    fSUM2 = fTI_ADC1 - fVDC6;
#ifdef _DEBUG
    fGblVAB = fSUM2;
#endif
    fVDC8 = 1.5;
    fSUM3 = fTI_ADC1_1 - fVDC8;
#ifdef _DEBUG
    fGblVBC = fSUM3;
#endif
    fVDC9 = 1.5;
    fSUM4 = fTI_ADC1_2 - fVDC9;
#ifdef _DEBUG
    fGblVCA = fSUM4;
#endif
    fNOT1 = !fTI_DIN1;
    if (fNOT1 > 0)
    {
        PS_StartPwm3ph1();
    }
    if (fTI_DIN1 != 0)
    {
        PS_StopPwm3ph1();
    }
}

void Initialize(void)
{
    PS_SysInit(30, 10);
    PS_StartStopPwmClock(0);
    PS_InitTimer(0, 0xffffffff);
}

```

```

    PS_InitPwm3ph(1, 1, 20000*1, (0.5e-6)*1.e6, 1811); // pwnNo, waveType,
frequency, deadtime
    PS_SetPwm3phPeakOffset(1, 3, (-1.5), 1.0/3);
    PS_SetPwm3ph1AdcIntr(ePwmNoAdc, 1, 0);
    PS_SetPwm3ph1Vector(ePwmNoAdc, Task);
    PS_SetPwm3ph1Uvw(0, 0, 0);
    PS_StopPwm3ph1();

    PS_ResetAdcConvSeq();
    PS_SetAdcConvSeq(eAdcCascade, 0, 1);
    PS_SetAdcConvSeq(eAdcCascade, 1, 1);
    PS_SetAdcConvSeq(eAdcCascade, 2, 1);
    PS_AdcInit(0, !0);

    PS_InitDigitIn(10, 100);

    PS_StartStopPwmClock(1);
}

void main()
{
    Initialize();
    PS_EnableIntr(); // Enable Global interrupt INTM
    PS_EnableDbgm();
    for (;;) {
        Task_1();
    }
}

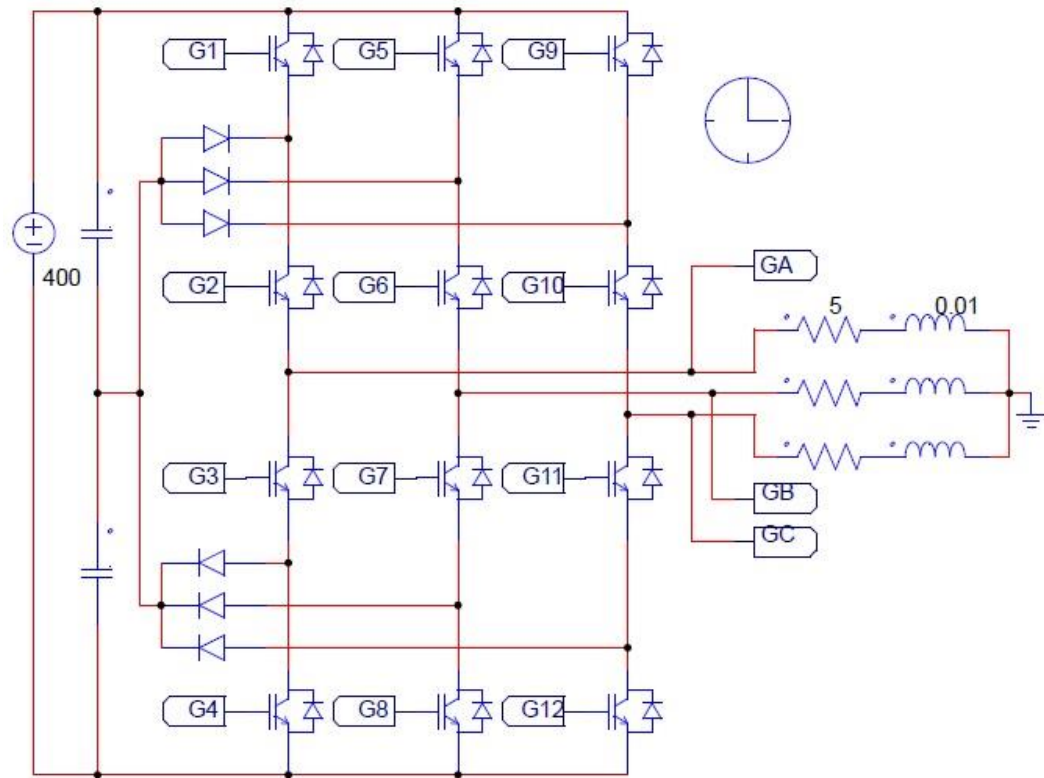
```

5.4. INVERSOR NPC

Na Figura 90 é observado o inversor NPC simulado. Este conversor apresenta três braços com quatro chaves em cada. Para cada chave é necessário um canal de PWM para seu comando, sendo necessários doze canais. A carga utilizada na simulação apresenta os mesmos valores utilizados para o inversor trifásico.

Para o comando deste inversor, foi utilizado dois blocos de PWM Trifásicos, um para as chaves de número ímpar e um para os interruptores de número par, conforme visto na figura acima. Os moduladores de PWM foram configurados para frequências de 20kHz, com portadora triangular e tempo morto de 0,5us. Somente os valores de pico das portadoras foram configurados diferentes, sendo que para o primeiro modulador foi configurado para 1V de pico e sem *offset*, e para o segundo, 1V de pico e *offset* de -1.

Figura 90 - Circuito da simulação e configurações do DSC para o inversor NPC simulado.

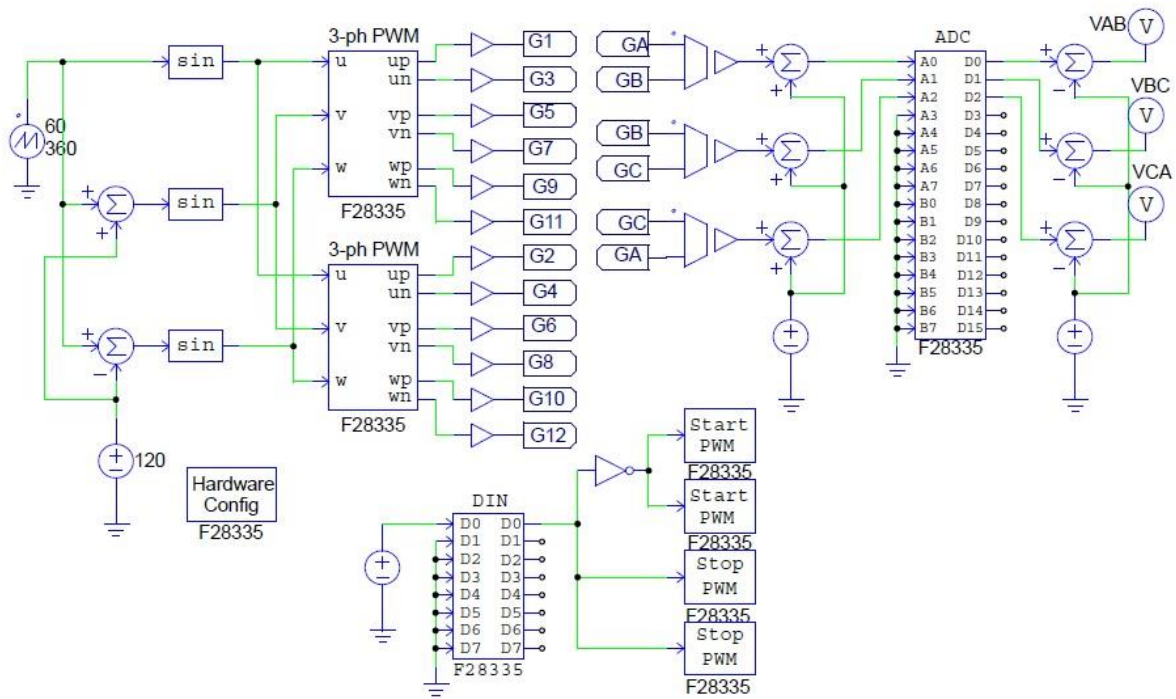


Fonte: Produção do próprio autor.

Esta estratégia de modulação é chamada de IPD (*In-Phase Disposition*), pois as duas portadoras estão em fase. Outro tipo semelhante de modulação é a POD (*Phase Opposition Disposition*), que também possui duas portadoras, uma positiva e uma negativa. Neste tipo de modulação as portadoras estão defasadas em 180° e não será utilizada, pois apresenta risco de acionar dois interruptores num mesmo braço simultaneamente, nos cruzamentos das portadoras com zero.

Foram utilizadas três ondas senoidais de referência, defasados em 120° entre si. Cada sinal foi conectado aos dois blocos de PWM Trifásico, um com portadora positiva e um com portadora negativa, conforme visto na Figura 91.

Figura 91 - Comando do inversor NPC simulado.

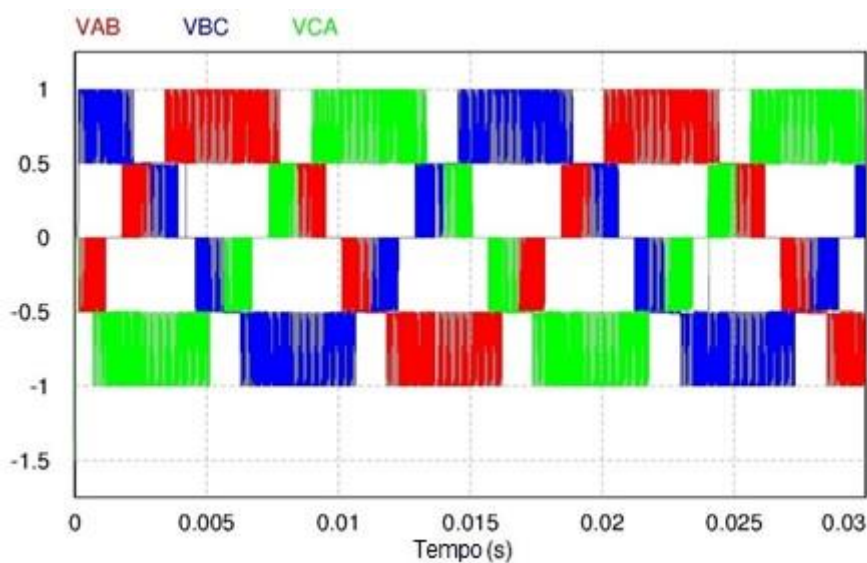


Fonte: Produção do próprio autor.

O restante dos componentes utilizados é igual aos utilizados nas simulações anteriores. O ganho dos sensores é de $1/400$, para a leitura do conversor A/D no modo DC. Para o circuito de ativação dos geradores de PWM, foram necessários dois blocos *Start PWM* e dois *Stop PWM*.

Como saída do NPC, obtém-se um sinal apresentando três níveis, conforme visto na Figura 92. Através da figura identificam-se claramente os níveis correspondentes a V_{in} , $V_{in}/2$, $-V_{in}$ e $-V_{in}/2$, sendo que na simulação V_{in} corresponde à 400V, porém, devido ao ganho dos sensores, os níveis foram alterados para 1V, 0,5V, -1V e -0,5V.

Figura 92 – Formas de onda das tensões de saída do NPC.



Fonte: Produção do próprio autor.

O código abaixo foi gerado a partir da simulação do NPC.

```

/*****
// This code is created by SimCoder Version 9.1 for TI F28335 Hardware Target
//
// SimCoder is copyright by Powersim Inc., 2009-2011
//
// Date: November 02, 2014 18:25:26
*****/
/
#include <math.h>
#include "PS_bios.h"
typedef float DefaultType;
#define GetCurTime() PS_GetSysTimer()

interrupt void Task();
void Task_1();

DefaultType fGblVAB = 0.0;
DefaultType fGblVBC = 0.0;
DefaultType fGblVCA = 0.0;
typedef struct {
    unsigned long tmLow;
    unsigned long tmHigh;
} _CBigTime;

_CBigTime GetBigTime(void)
{
    static _CBigTime tm = {0,0};

```

```

    unsigned long curTime = GetCurTime();
    if (curTime < tm.tmLow)
        tm.tmHigh++;
    tm.tmLow = curTime;
    return tm;
}

interrupt void Task()
{
    DefaultType fVSAW1, fSIN1, fVDC2, fSUMP1, fSIN2, fSUM1, fSIN3;
    PS_EnableIntr();

    {
        static unsigned long period = (unsigned long)(15000000L / 60);
        static float fPeriod = ((float)60) / 15000000L;
        static _CBigTime tmCarrierStart = {0, 0};
        _CBigTime tm = GetBigTime();
        unsigned long tmp1, tmp2;
        tmp1 = tm.tmLow - tmCarrierStart.tmLow;
        tmp2 = tm.tmHigh - tmCarrierStart.tmHigh;
        if (tm.tmLow > tmCarrierStart.tmLow)
            tmp2++;
        if (tmp2 || (!tmp2 && (tmp1 >= period))) {
            tmp1 = tmCarrierStart.tmLow + period;
            if ((tmp1 < tmCarrierStart.tmLow) || (tmp1 < period))
                tmCarrierStart.tmHigh++;
            tmCarrierStart.tmLow = tmp1;
        }
        tmp1 = tm.tmLow - tmCarrierStart.tmLow;
        fVSAW1 = 360 * tmp1 * fPeriod;
    }
    fSIN1 = sin(fVSAW1 * (3.14159265 / 180.));
    fVDC2 = 120;
    fSUMP1 = fVSAW1 + fVDC2;
    fSIN2 = sin(fSUMP1 * (3.14159265 / 180.));
    fSUM1 = fVSAW1 - fVDC2;
    fSIN3 = sin(fSUM1 * (3.14159265 / 180.));
    PS_SetPwm3ph1Uvw(fSIN1, fSIN2, fSIN3);
    PS_SetPwm3ph2Uvw(fSIN1, fSIN2, fSIN3);
    PS_ExitPwm1General();
}

void Task_1()
{
    DefaultType fTI_DIN1, fNOT1, fTI_ADC1, fVDC5, fSUM2, fTI_ADC1_1, fSUM3,
    fTI_ADC1_2, fSUM4;

    fTI_DIN1 = (PS_GetDigitInA() & ((Uint32)1 << 20)) ? 1 : 0;
    fTI_ADC1 = PS_GetDcAdc(0);
    fTI_ADC1_1 = PS_GetDcAdc(1);
    fTI_ADC1_2 = PS_GetDcAdc(2);
    fNOT1 = !fTI_DIN1;
    if (fNOT1 > 0)
    {
        PS_StartPwm3ph1();
    }
    if (fNOT1 > 0)

```

```

    {
        PS_StartPwm3ph2();
    }
    if (fTI_DIN1 != 0)
    {
        PS_StopPwm3ph1();
    }
    if (fTI_DIN1 != 0)
    {
        PS_StopPwm3ph2();
    }
    fVDC5 = 1.5;
    fSUM2 = fTI_ADC1 - fVDC5;
#ifdef _DEBUG
    fGblVAB = fSUM2;
#endif
    fSUM3 = fTI_ADC1_1 - fVDC5;
#ifdef _DEBUG
    fGblVBC = fSUM3;
#endif
    fSUM4 = fTI_ADC1_2 - fVDC5;
#ifdef _DEBUG
    fGblVCA = fSUM4;
#endif
}

void Initialize(void)
{
    PS_SysInit(30, 10);
    PS_StartStopPwmClock(0);
    PS_InitTimer(0, 0xffffffff);
    PS_InitPwm3ph(1, 1, 20000*1, (0.5e-6)*1.e6, 36118); // pwnNo, waveType,
frequency, deadtime
    PS_SetPwm3phPeakOffset(1, 1, 0, 1.0/1);
    PS_SetPwm3ph1AdcIntr(ePwmNoAdc, 1, 0);
    PS_SetPwm3ph1Vector(ePwmNoAdc, Task);
    PS_SetPwm3ph1Uvw(0, 0, 0);
    PS_StartPwm3ph1();

    PS_InitPwm3ph(2, 1, 20000*1, (0.5e-6)*1.e6, 36118); // pwnNo, waveType,
frequency, deadtime
    PS_SetPwm3phPeakOffset(2, 1, (-1), 1.0/1);
    PS_SetPwm3ph2AdcIntr(ePwmNoAdc, 1, 0);
    PS_SetPwm3ph2Uvw(0, 0, 0);
    PS_StartPwm3ph2();

    PS_ResetAdcConvSeq();
    PS_SetAdcConvSeq(eAdcCascade, 0, 1);
    PS_SetAdcConvSeq(eAdcCascade, 1, 1);
    PS_SetAdcConvSeq(eAdcCascade, 2, 1);
    PS_AdcInit(0, !0);

    PS_InitDigitIn(20, 100);

    PS_StartStopPwmClock(1);
}

```

```
void main()
{
    Initialize();
    PS_EnableIntr();    // Enable Global interrupt INTM
    PS_EnableDbgm();
    for (;;) {
        Task_1();
    }
}
```

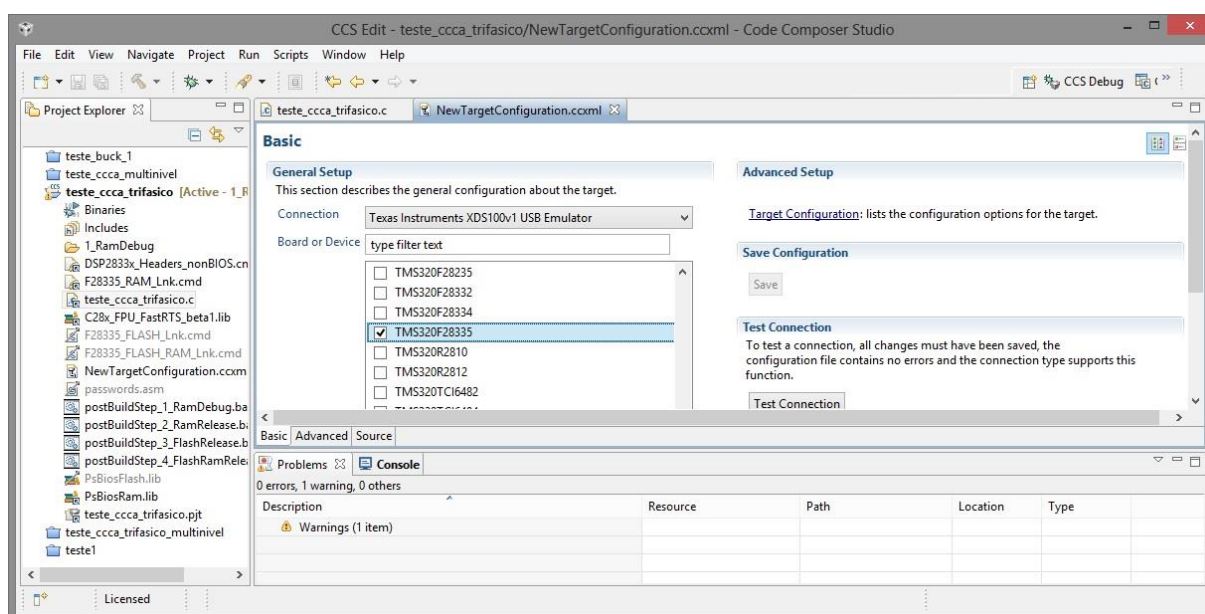
6. RESULTADOS EXPERIMENTAIS

Para realizar testes utilizando o TMS320F28335, será utilizado o *software Code Composer Studio*, de versão 5.1, para compilar e depurar os códigos gerados, observando as funcionalidades dos elementos do *SimCoder*.

6.1. TESTES COM O DSC

Primeiramente, deve-se abrir um projeto, seguindo os passos explicados no Capítulo 4. Para depurar o código para o DSC, deve-se criar um arquivo de configuração de destino, no qual se deve escolher qual dispositivo será utilizado. Para criar este arquivo, deve-se acionar o menu em *File>New>Target Configuration File*. Ao se abrir este arquivo, deve-se selecionar o tipo da conexão. Neste caso será utilizado *Texas Instruments XDS100v1 USB Emulator*. Deve-se selecionar então o dispositivo utilizado, neste caso o DSC TMS320F28335, e então salvá-lo. Na Figura 93 podem-se observar as configurações citadas.

Figura 93 - Configuração de Dispositivo pelo CCSv5.1



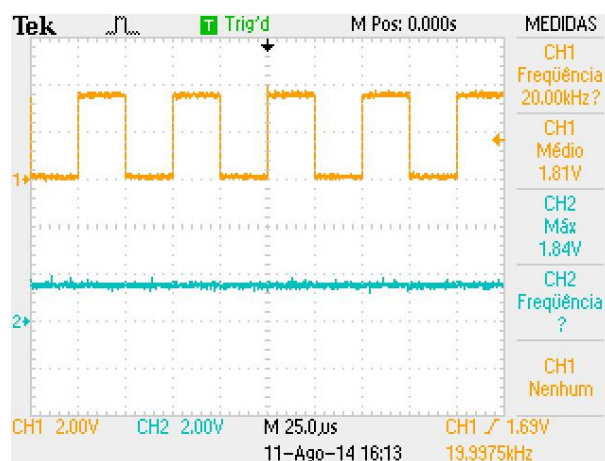
Fonte: Produção do próprio autor.

Na Figura 93 pode-se observar o ícone *Test Connection*, que permite que a conexão seja testada. Caso ocorra algum erro, é necessário fechar o CCS, trocar a porta USB utilizada e abri-lo novamente.

Para início dos testes, foi utilizado o circuito simulado e o código apresentado no Capítulo 4, na Figura 61. O circuito utiliza um valor obtido pelo conversor A/D para determinar a razão cíclica do modulador de PWM Monofásico. O bloco de PWM é acionado através dos componentes *Start PWM* e *Stop PWM*, que por sua vez são acionados por uma entrada digital. A partir do código em linguagem C gerado, pode-se então depurar o código para o DSC.

Para variar a tensão na entrada A0, correspondente à entrada do conversor A/D configurada na simulação, foi utilizado um potenciômetro, permitindo uma variação de 0 a 3,5V, aproximadamente. Na Figura 94, pode-se ver o sinal PWM gerado pelo DSC, medido na porta GPIO00, e o sinal de entrada do conversor A/D, que corresponde à razão cíclica do PWM, medido na porta ADCA0 do DSC.

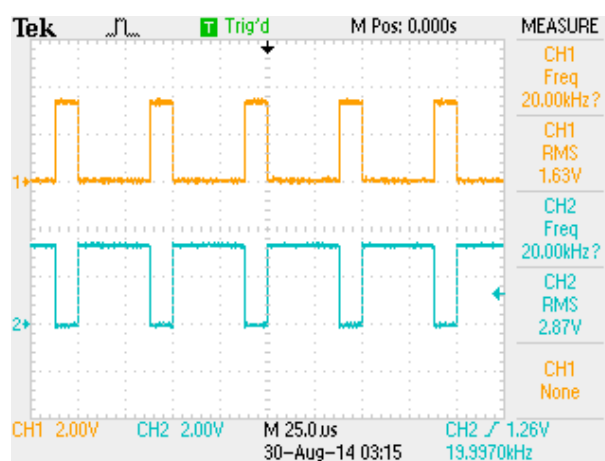
Figura 94 - Leituras do sinal de PWM e da tensão de entrada do conversor A/D.



Fonte: Produção do próprio autor.

O modulador PWM monofásico gera dois canais complementares, conforme visto na Figura 95. O segundo canal do PWM pode ser medido na porta GPIO 01.

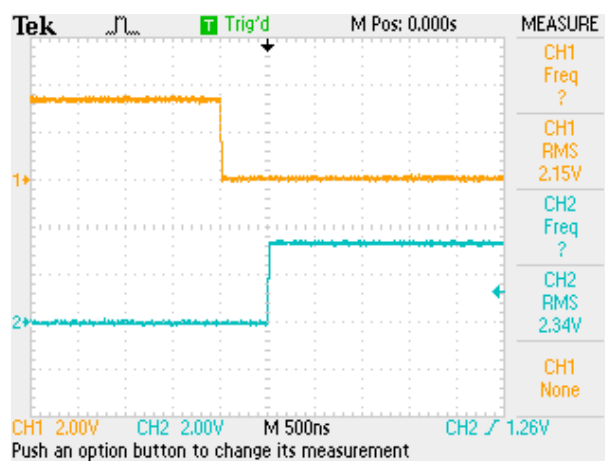
Figura 95 - Sinais PWM complementares obtidos.



Fonte: Produção do próprio autor.

Na Figura 96, pode-se ver através das linhas pontilhadas que o tempo morto entre os dois sinais de PWM gerados é realmente $0,5\mu\text{s}$, verificando com o valor configurado no componente através da simulação do PSIM.

Figura 96 - Tempo morto entre os canais de PWM.

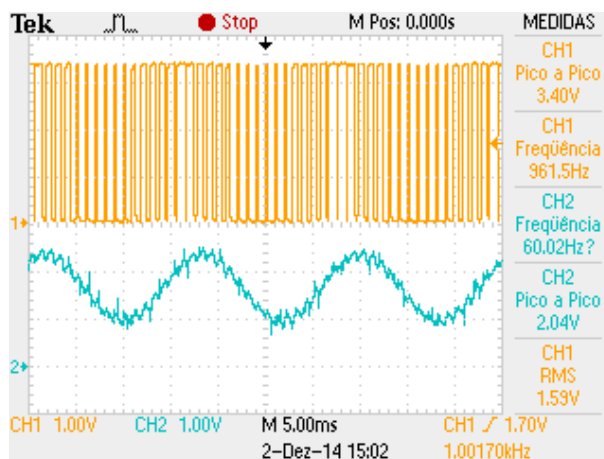


Fonte: Produção do próprio autor.

Utilizando uma referência senoidal para o modulador PWM, foi obtida a forma de onda observada na Figura 97. O sinal de referência utilizado foi inserido na própria simulação do PSIM, utilizado para gerar o código em linguagem C. Na figura encontra-se o sinal de PWM obtido e o sinal filtrado. Observa-se que este sinal obtido apresenta frequência de 60Hz

e foi obtido aplicando-se o sinal de PWM a um simples filtro RC. Para melhor visualização, foi utilizada frequência de 1kHz para o sinal de PWM.

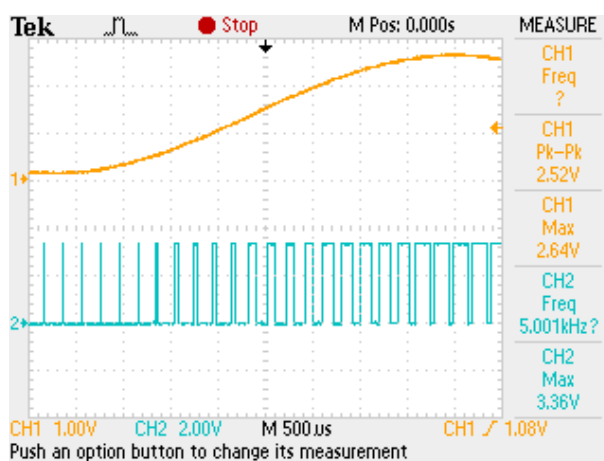
Figura 97 - Sinais obtidos a partir de uma referência senoidal.



Fonte: Produção do próprio autor.

Pode-se também utilizar o conversor A/D para se adquirir um sinal de referência externo para o gerador de PWM. Na Figura 98, observa-se uma onda senoidal de referência fornecido por uma fonte geradora de sinais e o sinal PWM gerado. Observa-se que a razão cíclica do sinal PWM varia de acordo com o sinal de referência.

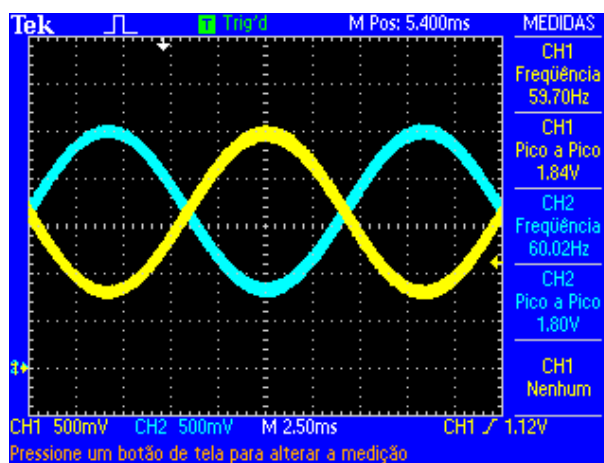
Figura 98 - Sinal de referência externo e sinal PWM gerado.



Fonte: Produção do próprio autor.

Na Figura 99, observam-se dois sinais gerados por um sinal PWM com referência senoidal após um filtro RC. Os sinais foram obtidos de dois canais de um mesmo gerador de PWM, portanto os sinais são complementares, gerando senoidais defasadas em 180° .

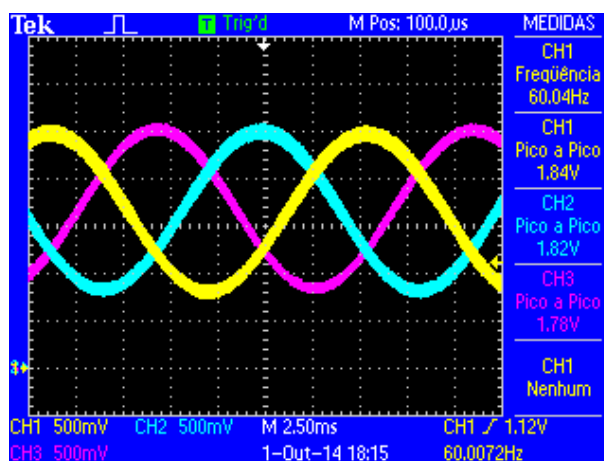
Figura 99 - Sinais gerados por sinais PWM complementares, após filtragem.



Fonte: Produção do próprio autor.

Na Figura 100, observam-se três sinais obtidos de um gerador PWM com referência senoidal, defasados entre si em 120° , para se verificar o funcionamento do gerador de PWM trifásico.

Figura 100 - Sinais obtidos do modulador PWM trifásico, após filtragem.

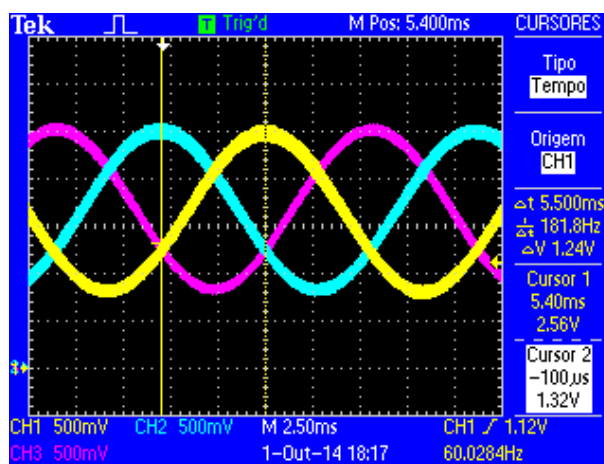


Fonte: Produção do próprio autor.

Os três sinais foram obtidos a partir de filtros RC iguais, utilizando um capacitor de 100nF e resistor de 10k Ω .

Pôde-se verificar a defasagem entre os sinais utilizando os cursores de tempo do osciloscópio. Considerando que os sinais estão defasados em 120°, a diferença de tempo entre os picos de duas fases deve ser igual a um terço do período da onda. Para a frequência de 60Hz utilizada, esta diferença de tempo deve ser aproximadamente igual a 5,55ms. Na Figura 101, pode-se observar que este tempo é verificado.

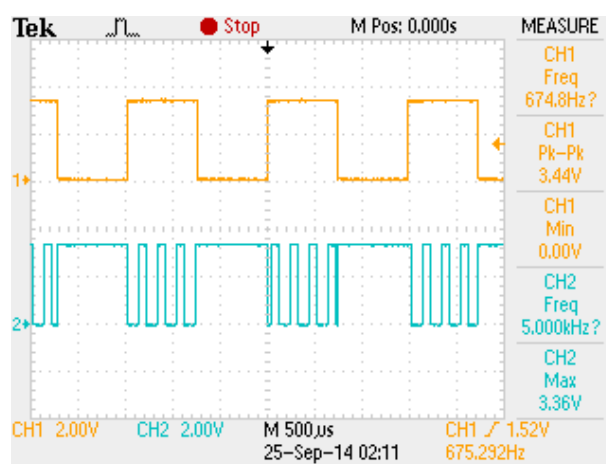
Figura 101 - Verificação da defasagem entre duas fases.



Fonte: Produção do próprio autor.

Por fim foi testada uma entrada digital. Na simulação do PSIM[®], foi utilizado um bloco de entradas digitais para acionar os componentes *Start* e *Stop* PWM, determinando o funcionamento do modulador PWM. Na Figura 102 são observadas uma forma de onda quadrada obtida por um gerador de sinais e o sinal de PWM.

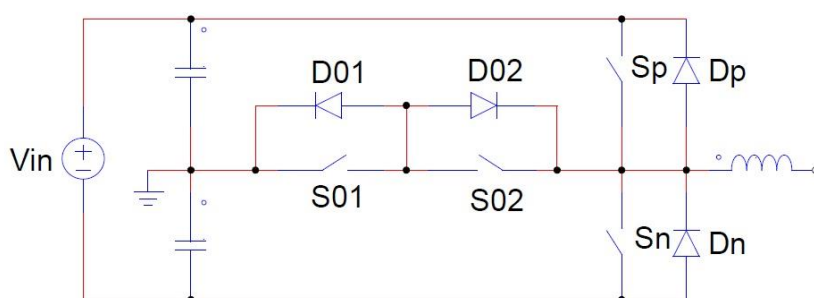
Figura 102 - Teste da entrada digital.



Fonte: Produção do próprio autor.

6.2. SIMULAÇÃO DO CONVERSOR NPC_m

O NPC_m (*Modified Neutral Point Clamped*) consiste em uma topologia alternativa para o NPC. Ambos os conversores apresentam três braços, possuindo quatro interruptores em cada braço, porém arranjadas de formas diferentes. Na Figura 103 é vista a representação de uma fase do NPC_m.

Figura 103 - Topologia de uma fase do NPC_m.

Fonte: Produção do próprio autor.

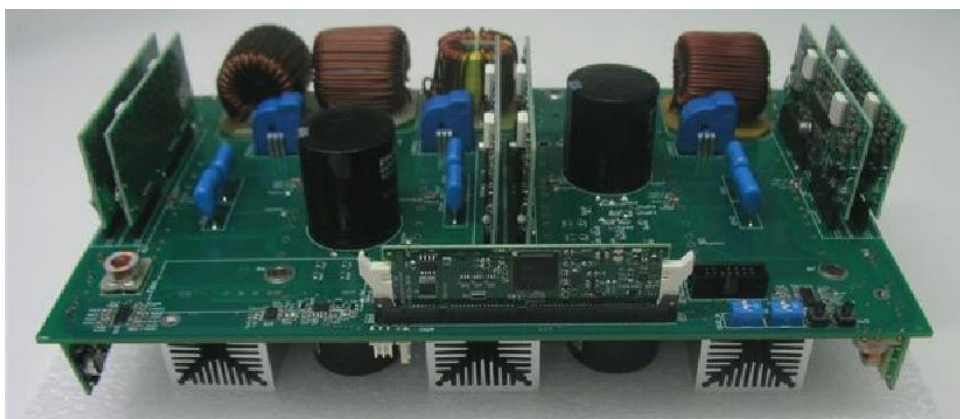
A principal diferença entre os semicondutores das duas estruturas é que no retificador NPC_m os semicondutores S_p , S_n , D_p e D_n , estão sujeitos à tensão total do barramento CC,

enquanto que todos os demais dispositivos, de ambas as estruturas, bloqueiam a metade desta tensão (HEERDT, 2013).

Apesar das diferenças de topologia, a estratégia de modulação deste conversor é a mesma utilizada para o NPC, apresentando as mesmas características do ponto de vista dos seus terminais. Porém o NPCm apresenta vantagens quanto ao número de componentes, por não necessitar de dois diodos de grampeamento.

A partir do estudo deste conversor, foi realizada a simulação utilizando os blocos do *SimCoder*, sendo gerado o código em linguagem C para o conversor NPCm mostrado na Figura 104. O conversor foi feito já com o encaixe para o DSC F28335, sendo que a implementação do código é realizada utilizando um cabo de comunicação serial.

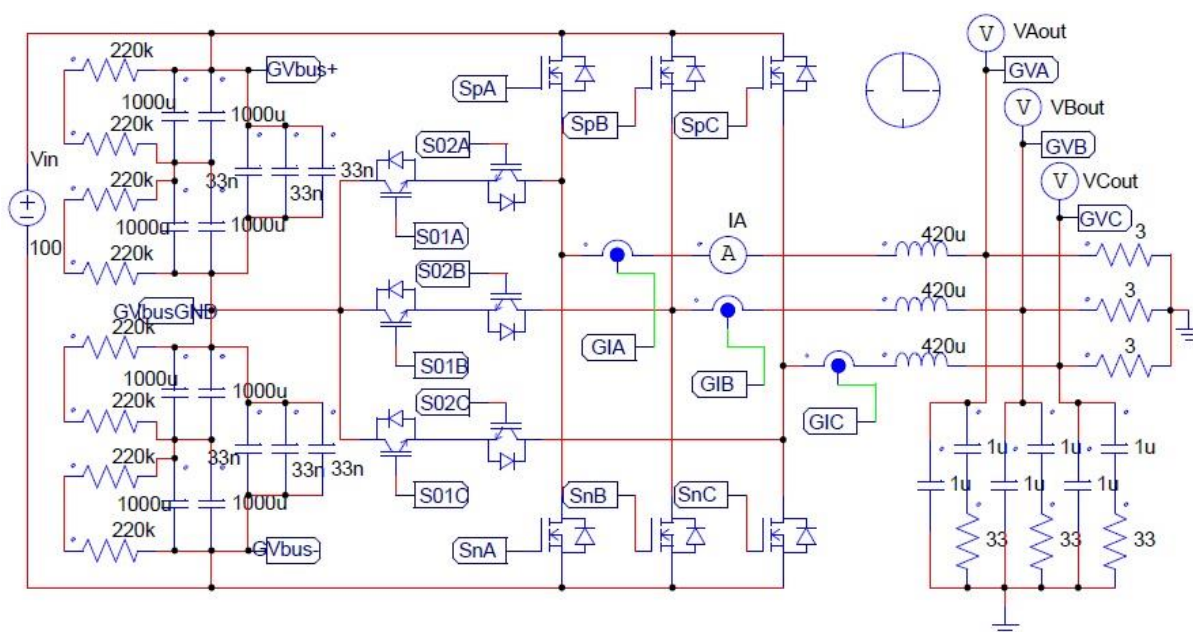
Figura 104 - NPCm utilizado para validação dos resultados.



Fonte: (HEERDT, 2013)

Na Figura 105 é mostrado o diagrama de simulação do circuito de potência do conversor NPCm trifásico, composto pelos filtros capacitivos de entrada, interruptores, filtros LC de saída e cargas resistivas de 3 Ohm. Em série com alguns capacitores são adicionados resistores de amortecimento de ressonâncias.

Figura 105 - Circuito de potência do Conversor NPCm.

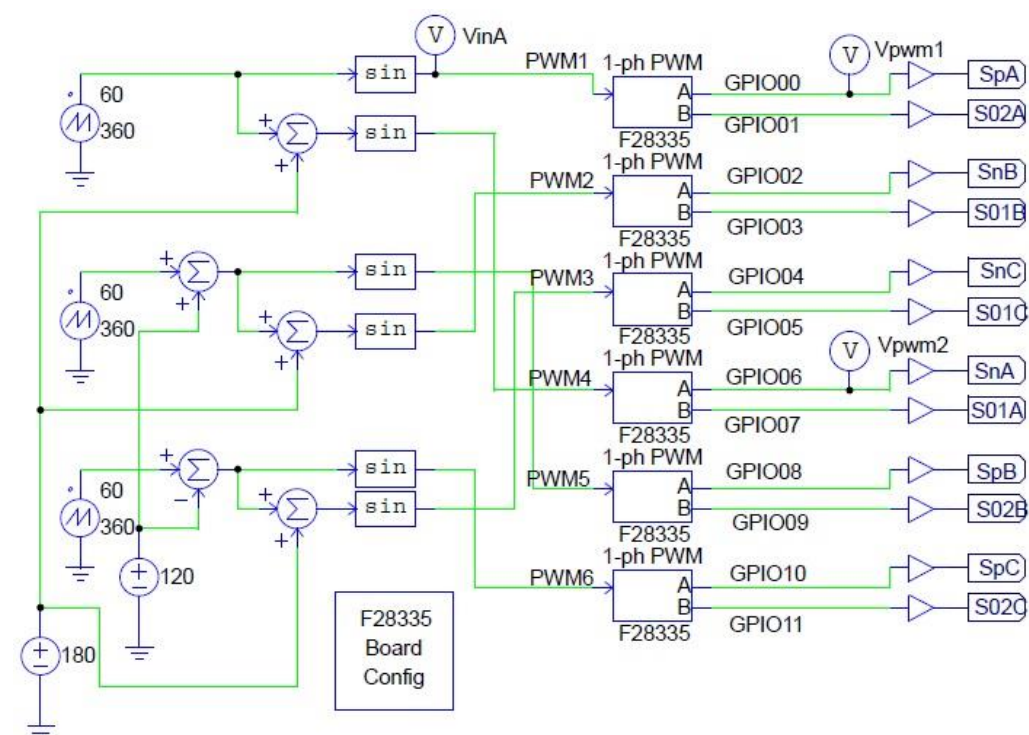


Fonte: Produção do próprio autor.

Para o comando do conversor, são necessários seis canais PWM. Como sinal de referência, foram utilizados senoides de 1V de amplitude, de frequência de 60Hz, defasados em 120° entre si. Poderia utilizar-se um *Phase Locked Loop* (PLL), para gerar estes sinais de referência, tendo-se como base a leitura da saída do conversor, que estaria conectada à rede elétrica. O PLL é utilizado para sincronizar as tensões da rede com a tensão produzida pelo inversor. Os blocos de PWM são observados na Figura 106.

A modulação utilizada é a IDP, onde são usadas duas portadoras triangulares, uma positiva e uma negativa. Para o conversor utilizado, a modulação foi projetada para realizar comparação positiva para a portadora positiva, e comparação negativa para a portadora negativa. Entende-se por comparação positiva, quando o sinal de referência está ligado à entrada positiva do comparador e a portadora, à entrada negativa. Quando o sinal de referência é ligado à entrada negativa, ocorre a comparação negativa.

Figura 106 - Geradores de PWM para comando do NPCM.

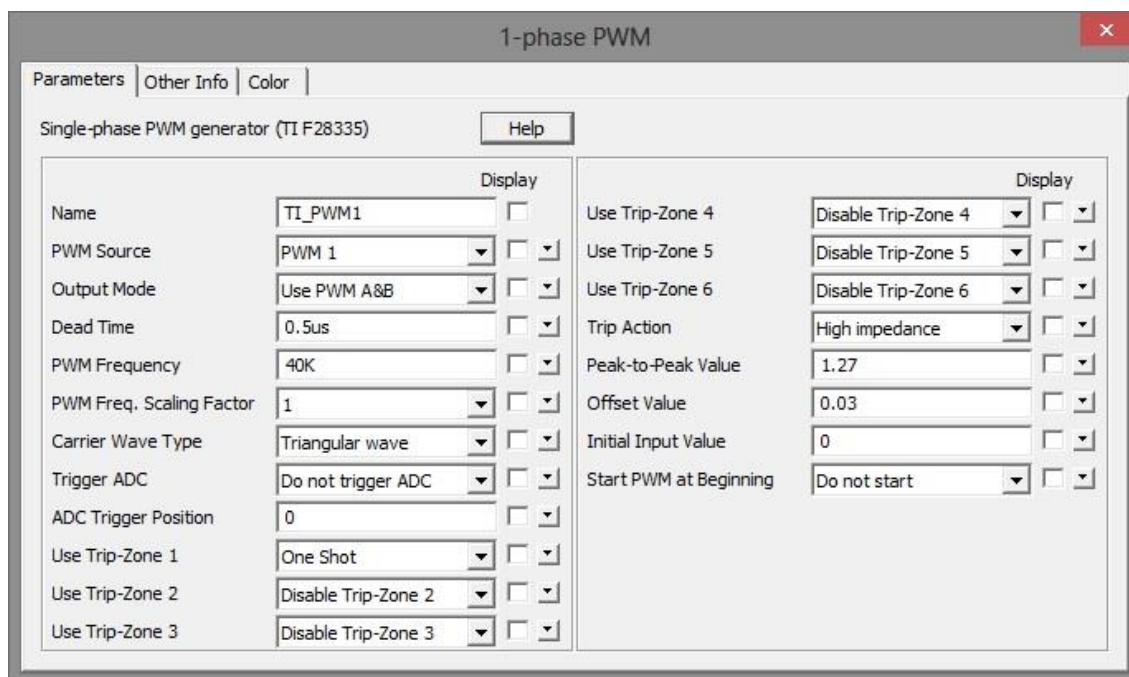


Fonte: Produção do próprio autor.

Porém, os blocos de PWM do *SimCoder* não podem ser configurados para realizar a comparação negativa. Os canais PWM1, PWM5 e PWM6 deveriam ser configurados para a portadora positiva e os canais PWM2, PWM3 e PWM4, para a portadora negativa. A solução encontrada foi utilizar apenas uma portadora para os seis blocos de PWM e defasar em 180° a senoide de referência dos canais que deveriam ter portadora negativa. As configurações do canal PWM1 podem ser vistas na Figura 107.

Foi configurado um pequeno nível de *offset* para simular o tempo morto encontrado na modulação IDP. Desta forma garante-se que os interruptores Sp e Sn não encontrem-se ativadas simultaneamente. Este nível de *offset* poderia ser desconsiderado, pois deforma os sinais de saída.

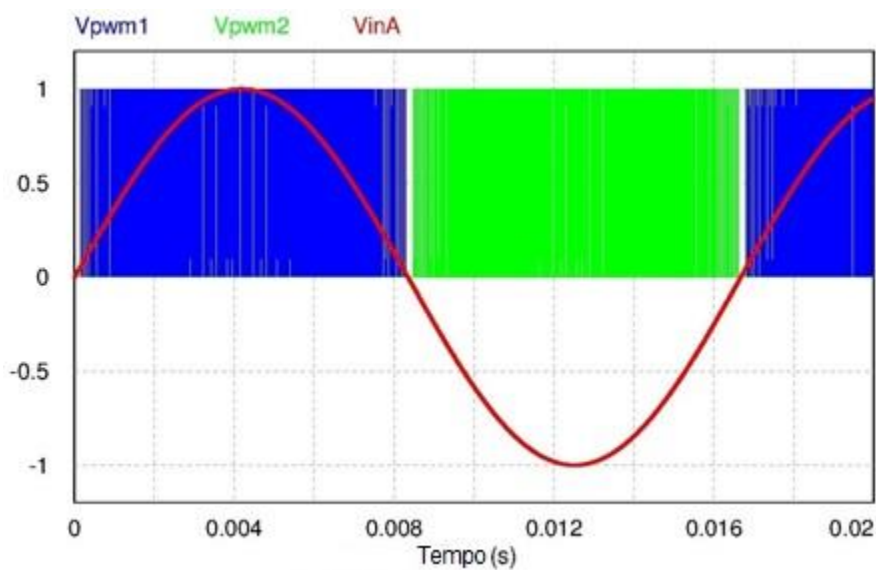
Figura 107 - Configurações de um dos geradores de PWM utilizados para comando do NPCM.



Fonte: PSIM®.

Na Figura 108 são observadas as formas de onda da senoidal de referência para a fase A do Conversor NPCM e as saídas dos geradores de PWM para as chaves Sp e Sn.

Figura 108 - Sinal de Referência e Sinais de PWM gerados.

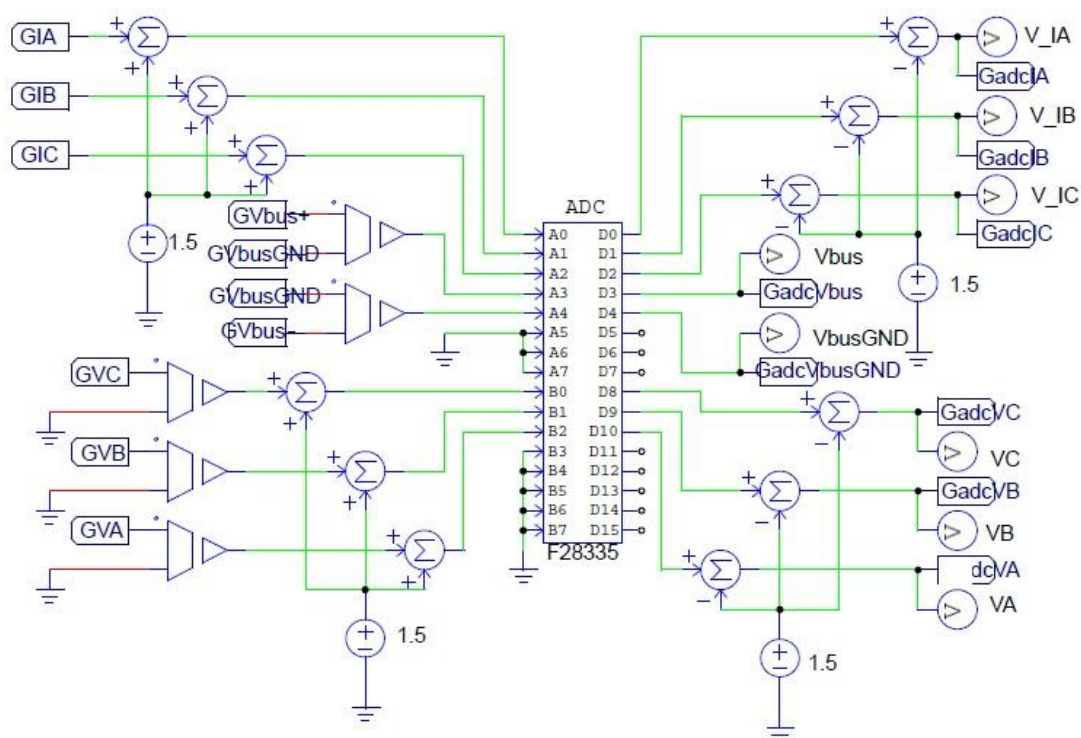


Fonte: Produção do próprio autor.

O NPCm utilizado apresentava diversos sensores, utilizados para leituras de tensão dos valores de entrada e de tensão e de corrente dos valores de saída. Para tal, foram utilizados oito canais do conversor A/D. Estes canais foram previamente determinados, portanto não foi possível alterá-los. Todos os canais foram configurados para o modo DC e para terem ganho unitário. Na Figura 109, observa-se o circuito em que se encontra o conversor A/D e os valores lidos.

Observa-se que para as leituras das tensões e de corrente de saída do NPCm, foram inseridos e posteriormente retirados um nível de *offset* de 1,5V, para apresentarem valores entre 0 e 3V, conforme requer o conversor A/D no modo DC. Observa-se que não é possível escolher a frequência de amostragem do Conversor A/D em suas configurações, mas sim nos blocos de PWM.

Figura 109 - Leituras realizadas pelo conversor A/D para o NPCm.



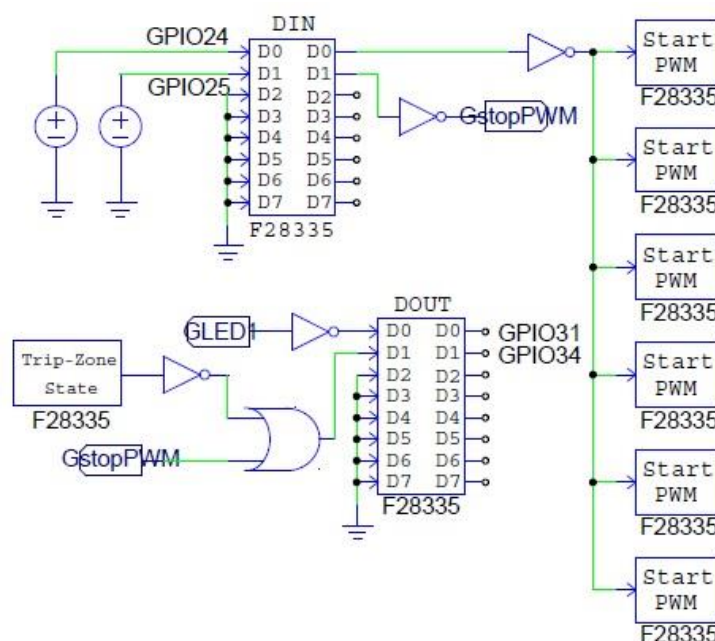
Fonte: Produção do próprio autor.

O conversor NPCm utilizado apresenta dois botões sendo utilizados como entradas digitais e dois LED's, presentes no próprio DSC, utilizados como saídas digitais. Ambas as

portas GPIO foram previamente definidas, não podendo ser alteradas. Para as entradas digitais, foram utilizadas as portas GPIO24 e GPIO25, e para as saídas, GPIO31 e GPIO34.

A porta GPIO24 é utilizada para acionar os geradores de PWM, através dos componentes *Start PWM*, sendo utilizados um para cada gerador. Já a porta GPIO25 é utilizada para parar os moduladores de PWM, utilizando os componentes *Stop PWM*, que serão apresentados posteriormente. Na Figura 110 são observados os blocos de entradas e saídas digitais.

Figura 110 - Entradas e Saídas Digitais utilizadas.



Fonte: Produção do próprio autor.

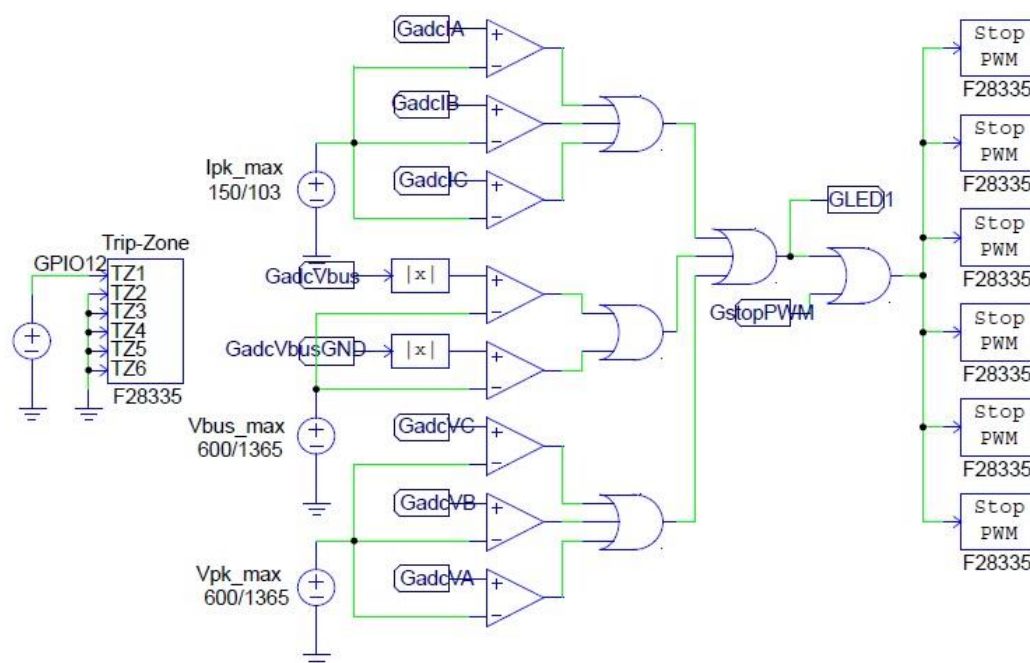
O LED ligado à porta GPIO31 é utilizado para indicar que as proteções do NPCm foram acionadas. Já o LED ligado à porta GPIO 34 indica se o botão que interrompe os blocos de PWM foi acionado ou se o *Trip-Zone* foi acionado, através do componente *Trip-Zone State*.

O *Trip-Zone* foi configurado para o canal *Trip-Zone 1* a porta GPIO12, que por sua vez recebe um sinal proveniente dos sensores de corrente, para proteção contra sobrecorrentes. Nas configurações dos moduladores de PWM, observado na Figura 86, foi

configurado para utilizar o *Trip-Zone* 1 no modo *One-Shot*, em que com um pulso na entrada do *Trip-Zone*, os geradores de PWM são desligados, sendo reativados somente se forem acionados os componentes *Start PWM*. O bloco *Trip-Zone State* também deve ser configurado para o canal *Trip-Zone* 1.

Na Figura 111, são observados os elementos de proteção do NPCM, sendo o *Trip-Zone* e os comparadores que acionam os blocos *Stop PWM*. Os valores de I_{pk_max} , V_{bus_max} e V_{pk_max} foram determinados para a devida proteção do NPCM, sendo que na simulação, estes valores foram divididos pelos ganhos dos sensores.

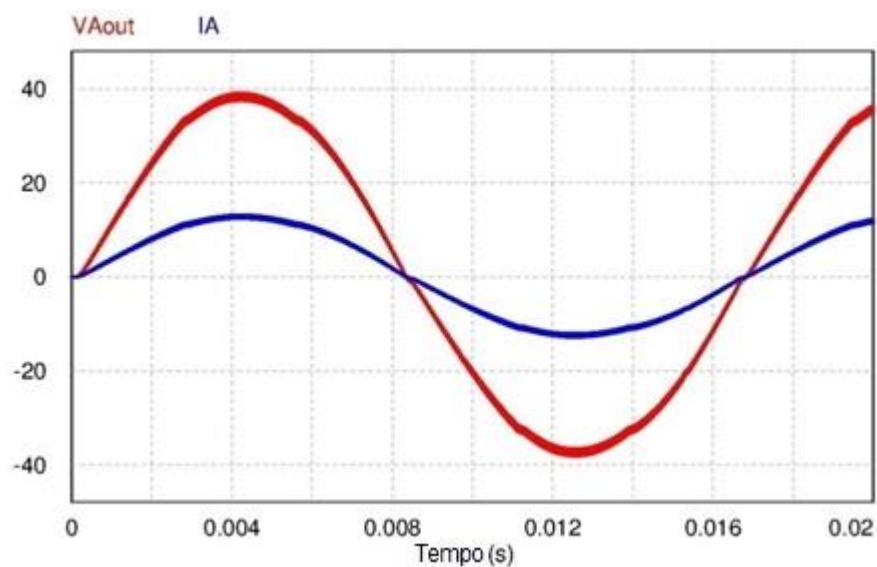
Figura 111 - Elementos de Proteção do NPCM.



Fonte: Produção do próprio autor.

É observado na Figura 112 a tensão e a corrente de saída na fase A do conversor. Observa-se que em cada cruzamento com zero há uma pequena perturbação, devido ao tempo morto entre os comandos dos interruptores S_p e S_n , e entre S_{01} e S_{02} . As perturbações nos picos das senóides possuem a mesma causa.

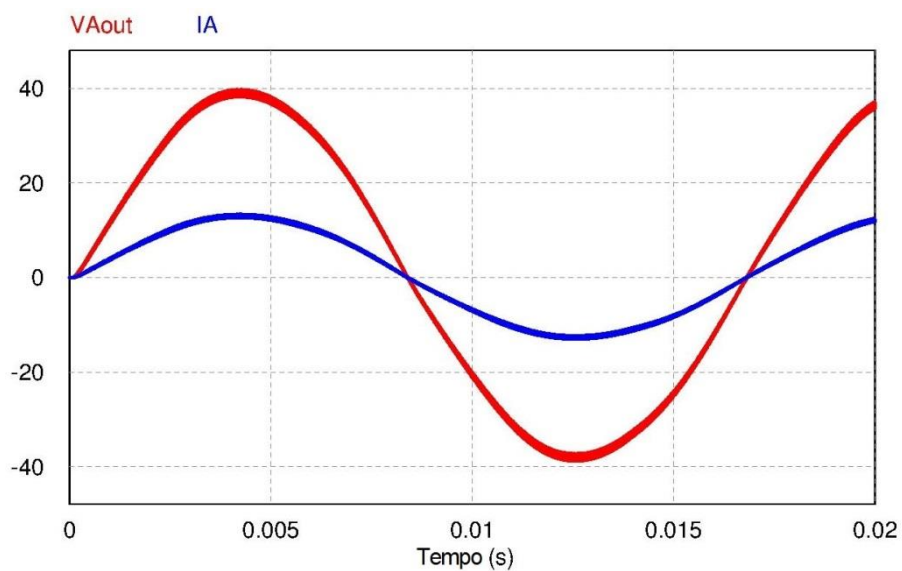
Figura 112 - Tensão e corrente de saída da fase A do conversor NPCm.



Fonte: Produção do próprio autor.

Retirando-se o nível de *offset* das portadoras dos moduladores PWM as perturbações são eliminadas, conforme mostrado na Figura 113.

Figura 113 - Tensão e corrente de saída do conversor NPCm sem perturbações.



Fonte: Produção do próprio autor.

A partir da realização da simulação do conversor NPCm, o código abaixo foi gerado.

```

/*****
// This code is created by SimCoder Version 9.1 for TI F28335 Hardware Target
//
// SimCoder is copyright by Powersim Inc., 2009-2011
//
// Date: December 08, 2014 14:10:43
*****/
#include <math.h>
#include "PS_bios.h"
typedef float DefaultType;
#define GetCurTime() PS_GetSysTimer()

interrupt void Task();
void Task_1();

DefaultType fGblVinA = 0.0;
DefaultType fGblV_IB = 0.0;
DefaultType fGblV_IC = 0.0;
DefaultType fGblVbus = 0.0;
DefaultType fGblVbusGND = 0.0;
DefaultType fGblVC = 0.0;
DefaultType fGblVB = 0.0;
DefaultType fGblV_IA = 0.0;
DefaultType fGblVA = 0.0;
typedef struct {
    unsigned long tmLow;
    unsigned long tmHigh;
} _CBigTime;

_CBigTime GetBigTime(void)
{
    static _CBigTime tm = {0,0};
    unsigned long curTime = GetCurTime();
    if (curTime < tm.tmLow)
        tm.tmHigh++;
    tm.tmLow = curTime;
    return tm;
}

interrupt void Task()
{
    DefaultType fVSAW10, fSIN13, fVDC19, fSUMP18, fVDC29, fSUMP26, fSIN24,
fSUM12, fSUMP25;
    DefaultType fSIN25, fSUMP27, fSIN23, fSIN16, fSIN17;
    PS_EnableIntr();

    {
        static unsigned long period = (unsigned long)(15000000L / 60);
        static float fPeriod = ((float)60) / 15000000L;
        static _CBigTime tmCarrierStart = {0, 0};
        _CBigTime tm = GetBigTime();
        unsigned long tmp1, tmp2;
        tmp1 = tm.tmLow - tmCarrierStart.tmLow;
    }
}

```

```

        tmp2 = tm.tmHigh - tmCarrierStart.tmHigh;
        if (tm.tmLow > tmCarrierStart.tmLow)
            tmp2++;
        if (tmp2 || (!tmp2 && (tmp1 >= period))) {
            tmp1 = tmCarrierStart.tmLow + period;
            if ((tmp1 < tmCarrierStart.tmLow) || (tmp1 < period))
                tmCarrierStart.tmHigh++;
            tmCarrierStart.tmLow = tmp1;
        }
        tmp1 = tm.tmLow - tmCarrierStart.tmLow;
        fVSAW10 = 360 * tmp1 * fPeriod;
    }
    fSIN13 = sin(fVSAW10 * (3.14159265 / 180.));
#ifdef _DEBUG
    fGblVinA = fSIN13;
#endif
    PS_SetPwm1Rate(fSIN13);
    fVDC19 = 120;
    fSUMP18 = fVSAW10 + fVDC19;
    fVDC29 = 180;
    fSUMP26 = fSUMP18 + fVDC29;
    fSIN24 = sin(fSUMP26 * (3.14159265 / 180.));
    PS_SetPwm2Rate(fSIN24);
    fSUM12 = fVSAW10 - fVDC19;
    fSUMP25 = fSUM12 + fVDC29;
    fSIN25 = sin(fSUMP25 * (3.14159265 / 180.));
    PS_SetPwm3Rate(fSIN25);
    fSUMP27 = fVSAW10 + fVDC29;
    fSIN23 = sin(fSUMP27 * (3.14159265 / 180.));
    PS_SetPwm4Rate(fSIN23);
    fSIN16 = sin(fSUMP18 * (3.14159265 / 180.));
    PS_SetPwm5Rate(fSIN16);
    fSIN17 = sin(fSUM12 * (3.14159265 / 180.));
    PS_SetPwm6Rate(fSIN17);
    PS_ExitPwm1General();
}

void Task_1()
{
    DefaultType fTI_DIN1, fNOT16, fTI_ADC1, fVDC20, fSUM15, fIpk_max, fCOMP8,
    fTI_ADC1_1;
    DefaultType fSUM13, fCOMP9, fTI_ADC1_2, fSUM14, fCOMP10, fOR31, fTI_ADC1_3,
    fABS1, fVbus_max;
    DefaultType fCOMP11, fTI_ADC1_4, fABS2, fCOMP12, fOR1, fTI_ADC1_8, fVDC10,
    fSUM7, fVpk_max;
    DefaultType fCOMP13, fTI_ADC1_9, fSUM6, fCOMP14, fTI_ADC1_10, fSUM9,
    fCOMP15, fOR32;
    DefaultType fOR33, fNOT18, fTI_TZSTATE1, fNOT9, fTI_DIN1_1, fNOT17, fOR4,
    fOR2;

    fTI_DIN1 = (PS_GetDigitInA() & ((Uint32)1 << 24)) ? 1 : 0;
    fTI_ADC1 = PS_GetDcAdc(0);
    fTI_ADC1_1 = PS_GetDcAdc(1);
    fTI_ADC1_2 = PS_GetDcAdc(2);
    fTI_ADC1_3 = PS_GetDcAdc(3);
    fTI_ADC1_4 = PS_GetDcAdc(4);
    fTI_ADC1_8 = PS_GetDcAdc(8);

```

```

fTI_ADC1_9 = PS_GetDcAdc(9);
fTI_ADC1_10 = PS_GetDcAdc(10);
fTI_DIN1_1 = (PS_GetDigitInA() & ((Uint32)1 << 25)) ? 1 : 0;
fNOT16 = !fTI_DIN1;
if (fNOT16 > 0)
{
    PS_StartPwm(1);
}
if (fNOT16 > 0)
{
    PS_StartPwm(2);
}
fVDC20 = 1.5;
fSUM15 = fTI_ADC1 - fVDC20;
fIpk_max = (150.0/103);
fCOMP8 = (fSUM15 > fIpk_max) ? 1 : 0;
fSUM13 = fTI_ADC1_1 - fVDC20;
fCOMP9 = (fSUM13 > fIpk_max) ? 1 : 0;
fSUM14 = fTI_ADC1_2 - fVDC20;
fCOMP10 = (fSUM14 > fIpk_max) ? 1 : 0;
fOR31 = (fCOMP8 != 0) || (fCOMP9 != 0) || (fCOMP10 != 0);
fABS1 = fabs(fTI_ADC1_3);
fVbus_max = (600.0/1365);
fCOMP11 = (fABS1 > fVbus_max) ? 1 : 0;
fABS2 = fabs(fTI_ADC1_4);
fCOMP12 = (fABS2 > fVbus_max) ? 1 : 0;
fOR1 = (fCOMP11 != 0) || (fCOMP12 != 0);
fVDC10 = 1.5;
fSUM7 = fTI_ADC1_8 - fVDC10;
fVpk_max = (600.0/1365);
fCOMP13 = (fSUM7 > fVpk_max) ? 1 : 0;
fSUM6 = fTI_ADC1_9 - fVDC10;
fCOMP14 = (fSUM6 > fVpk_max) ? 1 : 0;
fSUM9 = fTI_ADC1_10 - fVDC10;
fCOMP15 = (fSUM9 > fVpk_max) ? 1 : 0;
fOR32 = (fCOMP13 != 0) || (fCOMP14 != 0) || (fCOMP15 != 0);
fOR33 = (fOR31 != 0) || (fOR1 != 0) || (fOR32 != 0);
fNOT18 = !fOR33;
fTI_TZSTATE1 = PS_IsPwm1OneShotTZ();
fNOT9 = !fTI_TZSTATE1;
fNOT17 = !fTI_DIN1_1;
fOR4 = (fNOT9 != 0) || (fNOT17 != 0);
#ifdef _DEBUG
    fGblV_IB = fSUM13;
#endif
#ifdef _DEBUG
    fGblV_IC = fSUM14;
#endif
#ifdef _DEBUG
    fGblVbus = fTI_ADC1_3;
#endif
#ifdef _DEBUG
    fGblVbusGND = fTI_ADC1_4;
#endif
#ifdef _DEBUG
    fGblVC = fSUM7;
#endif

```

```

#ifdef _DEBUG
    fGblVB = fSUM6;
#endif
#ifdef _DEBUG
    fGblV_IA = fSUM15;
#endif
    if (fNOT16 > 0)
    {
        PS_StartPwm(3);
    }
    if (fNOT16 > 0)
    {
        PS_StartPwm(4);
    }
    if (fNOT16 > 0)
    {
        PS_StartPwm(5);
    }
    if (fNOT16 > 0)
    {
        PS_StartPwm(6);
    }
    FOR2 = (FOR33 != 0) || (fNOT17 != 0);
    if (FOR2 != 0)
    {
        PS_StopPwm(6);
    }
    if (FOR2 != 0)
    {
        PS_StopPwm(2);
    }
    if (FOR2 != 0)
    {
        PS_StopPwm(1);
    }
    if (FOR2 != 0)
    {
        PS_StopPwm(3);
    }
    if (FOR2 != 0)
    {
        PS_StopPwm(4);
    }
    if (FOR2 != 0)
    {
        PS_StopPwm(5);
    }
#ifdef _DEBUG
    fGblVA = fSUM9;
#endif
    (fNOT18 == 0) ? PS_ClearDigitOutBitA((Uint32)1 << 31) :
PS_SetDigitOutBitA((Uint32)1 << 31);
    (FOR4 == 0) ? PS_ClearDigitOutBitB((Uint32)1 << (34 - 32)) :
PS_SetDigitOutBitB((Uint32)1 << (34 - 32));
}
void Initialize(void)
{

```

```

    PS_SysInit(30, 10);
    PS_StartStopPwmClock(0);
    PS_InitTimer(0, 0xffffffff);
    PS_InitPwm(1, 1, 40000*1, (0.5e-6)*1e6, PWM_TWO_OUT, 10665); // pwnNo,
waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(1, 1.3, 0, 1.0/1.3);
    PS_SetPwmIntrType(1, ePwmNoAdc, 1, 0);
    PS_SetPwmVector(1, ePwmNoAdc, Task);
    PS_SetPwmTripZone(1, 1, eTzEnOneShot);
    PS_SetPwmTzAct(1, eTZHighImpedance);
    PS_SetPwm1Rate(0);
    PS_StopPwm(1);

    PS_InitPwm(2, 1, 40000*1, (0.5e-6)*1e6, PWM_TWO_OUT, 10665); // pwnNo,
waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(2, 1.3, 0, 1.0/1.3);
    PS_SetPwmIntrType(2, ePwmNoAdc, 1, 0);
    PS_SetPwmTripZone(2, 1, eTzEnOneShot);
    PS_SetPwmTzAct(2, eTZHighImpedance);
    PS_SetPwm2Rate(0);
    PS_StopPwm(2);

    PS_InitPwm(3, 1, 40000*1, (0.5e-6)*1e6, PWM_TWO_OUT, 10665); // pwnNo,
waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(3, 1.3, 0, 1.0/1.3);
    PS_SetPwmIntrType(3, ePwmNoAdc, 1, 0);
    PS_SetPwmTripZone(3, 1, eTzEnOneShot);
    PS_SetPwmTzAct(3, eTZHighImpedance);
    PS_SetPwm3Rate(0);
    PS_StopPwm(3);

    PS_InitPwm(4, 1, 40000*1, (0.5e-6)*1e6, PWM_TWO_OUT, 10665); // pwnNo,
waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(4, 1.3, 0, 1.0/1.3);
    PS_SetPwmIntrType(4, ePwmNoAdc, 1, 0);
    PS_SetPwmTripZone(4, 1, eTzEnOneShot);
    PS_SetPwmTzAct(4, eTZHighImpedance);
    PS_SetPwm4Rate(0);
    PS_StopPwm(4);

    PS_InitPwm(5, 1, 40000*1, (0.5e-6)*1e6, PWM_TWO_OUT, 10665); // pwnNo,
waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(5, 1.3, 0, 1.0/1.3);
    PS_SetPwmIntrType(5, ePwmNoAdc, 1, 0);
    PS_SetPwmTripZone(5, 1, eTzEnOneShot);
    PS_SetPwmTzAct(5, eTZHighImpedance);
    PS_SetPwm5Rate(0);
    PS_StopPwm(5);

    PS_InitPwm(6, 1, 40000*1, (0.5e-6)*1e6, PWM_TWO_OUT, 10665); // pwnNo,
waveType, frequency, deadtime, outtype
    PS_SetPwmPeakOffset(6, 1.3, 0, 1.0/1.3);
    PS_SetPwmIntrType(6, ePwmNoAdc, 1, 0);
    PS_SetPwmTripZone(6, 1, eTzEnOneShot);
    PS_SetPwmTzAct(6, eTZHighImpedance);
    PS_SetPwm6Rate(0);
    PS_StopPwm(6);

```

```

    PS_ResetAdcConvSeq();
    PS_SetAdcConvSeq(eAdcCascade, 0, 1.0);
    PS_SetAdcConvSeq(eAdcCascade, 1, 1.0);
    PS_SetAdcConvSeq(eAdcCascade, 2, 1.0);
    PS_SetAdcConvSeq(eAdcCascade, 3, 1.0);
    PS_SetAdcConvSeq(eAdcCascade, 4, 1.0);
    PS_SetAdcConvSeq(eAdcCascade, 8, 1.0);
    PS_SetAdcConvSeq(eAdcCascade, 9, 1.0);
    PS_SetAdcConvSeq(eAdcCascade, 10, 1.0);
    PS_AdcInit(0, !0);

    PS_InitDigitIn(24, 100);
    PS_InitDigitIn(25, 100);

    PS_InitDigitOut(31);
    PS_InitDigitOut(34);

    PS_StartStopPwmClock(1);
}

void main()
{
    Initialize();
    PS_EnableIntr();    // Enable Global interrupt INTM
    PS_EnableDbgm();
    for (;;) {
        Task_1();
    }
}

```

6.3. IMPLEMENTAÇÃO DO CÓDIGO E TESTES

Através do *software Code Composer Studio*, foi possível implementar o código gerado no DSC TI F28335. Como primeiro passo, foram verificadas as formas de onda nos interruptores de uma mesma fase do conversor, apenas com a alimentação dos *drivers*, ou seja, sem tensão de entrada no barramento CC. Na Figura 114, são vistas as formas de onda entre o *Gate* e o *Source* das chaves da fase A do Conversor NPCm.

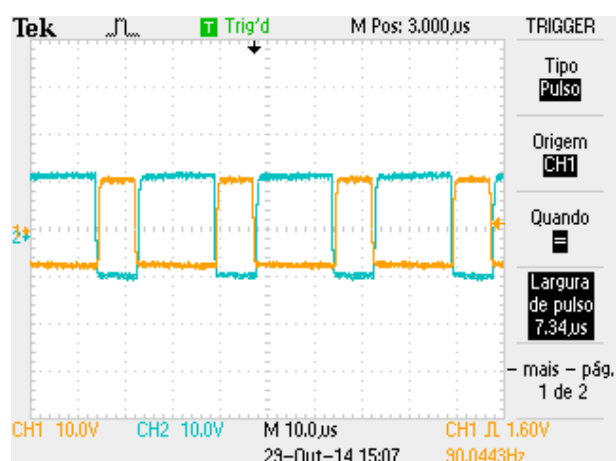
Figura 114 - Formas de onda entre o *Gate* e o *Source* das chaves da fase A.



Fonte: Produção do próprio autor.

No canal 1 do osciloscópio, visto na figura anterior, é mostrado o sinal do PWM do interruptor S02, no canal 2, de Sp, no canal 3, de Sn e no canal 4, do interruptor S01. Observa-se que, assim como na simulação, os sinais nos interruptores Sp e S02, assim como em Sn e S01, são complementares. Também é possível verificar que o sinal dos interruptores Sn e S01 estão defasados em 180° em relação aos interruptores Sp e S02. Ao se diminuir a escala do osciloscópio, é possível visualizar melhor que os sinais de PWM das chaves Sp e S02 são complementares, conforme mostrado na Figura 115.

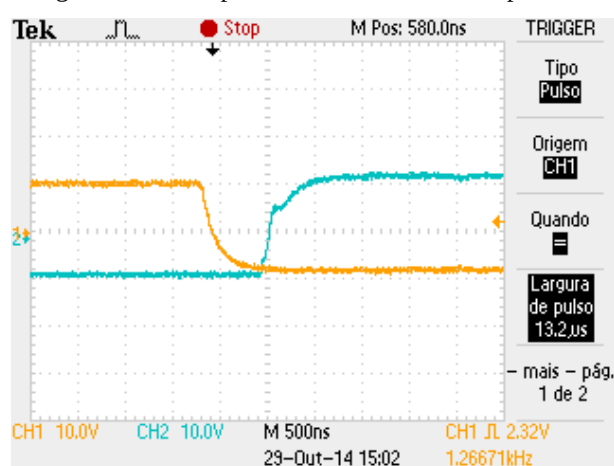
Figura 115 - Formas de onda nas chaves Sp e S02.



Fonte: Produção do próprio autor.

Ao se reduzir ainda mais a escala, pode-se observar através da grade pontilhada do osciloscópio, que o tempo morto configurado na simulação do PSIM de $0,5\mu\text{s}$ é verificado, conforme mostrado na Figura 116.

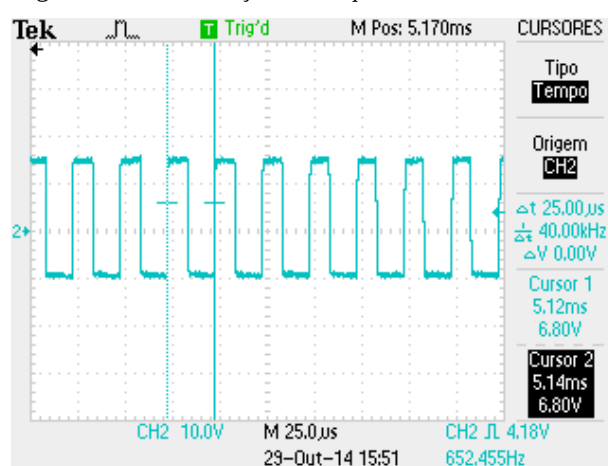
Figura 116 - Tempo morto entre as chaves Sp e S02.



Fonte: Produção do próprio autor.

Outra configuração verificada foi a frequência de chaveamento dos canais de PWM, utilizando-se cursores do osciloscópio. Na Figura 117 é mostrada a frequência de chaveamento, que é de 40kHz, assim como a que foi configurada na simulação para a geração do código.

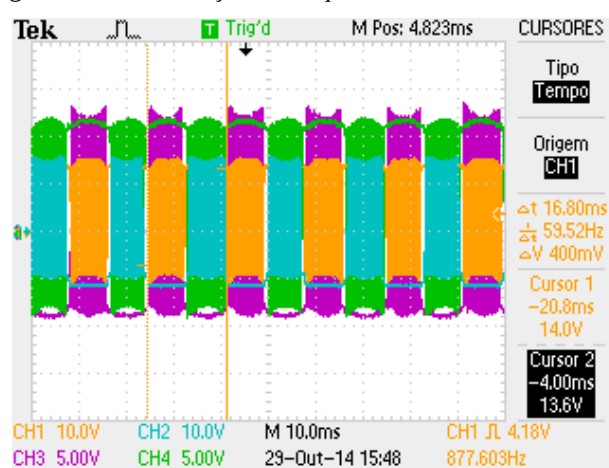
Figura 117 - Verificação da frequência de chaveamento.



Fonte: Produção do próprio autor.

A frequência do sinal de referência também foi verificada utilizando-se cursores do osciloscópio. Na Figura 118, foram utilizados os quatro canais do osciloscópio para verificar as formas de onda dos interruptores, assim como a frequência do sinal de referência.

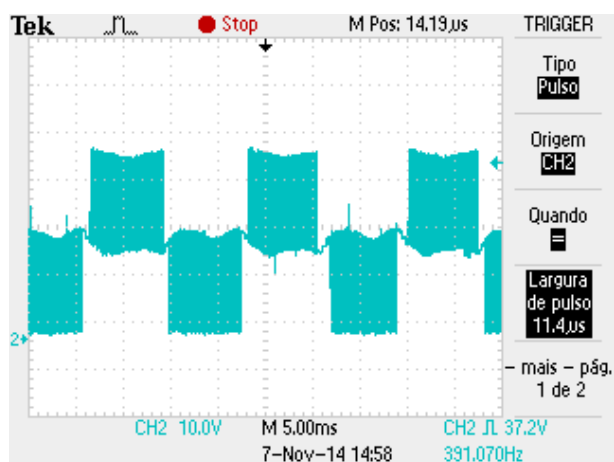
Figura 118 - Verificação da frequência do sinal de referência.



Fonte: Produção do próprio autor.

Com o funcionamento dos interruptores verificado, foi acionada então a tensão no barramento CC, utilizando um retificador alimentado por um Varivolt trifásico. Na Figura 119, observa-se a forma de onda da tensão entre o *Drain* e o *Source* da chave SpA, onde são mostrados dois níveis de tensão.

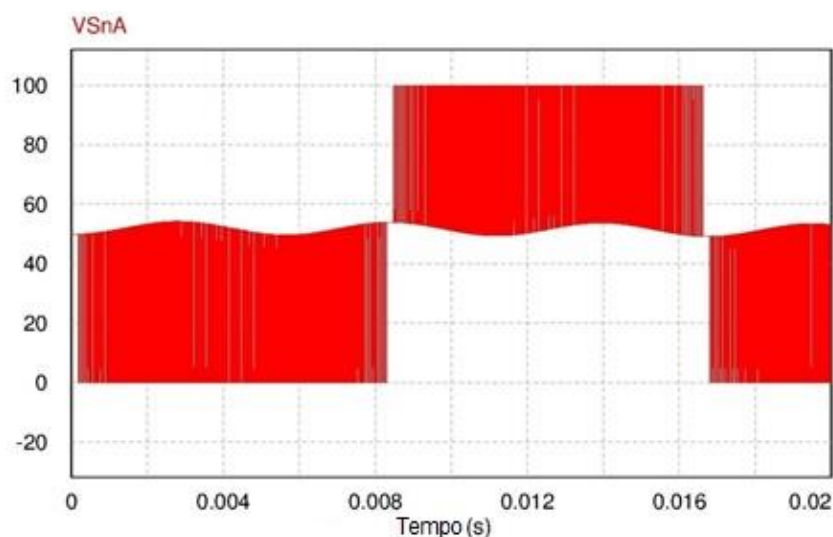
Figura 119 - Forma de onda da tensão entre o *Drain* e o *Source* do interruptor SpA.



Fonte: Produção do próprio autor.

Como comparativo, pode-se observar a forma de onda resultante da simulação, na Figura 120. Na simulação também são observados dois níveis de tensão, tal como a ondulação encontrada na forma de onda obtida experimentalmente.

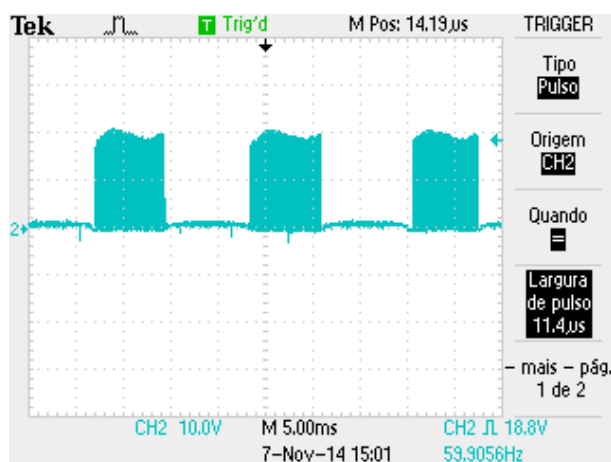
Figura 120 - Tensão entre o *Drain* e o *Source* obtido na simulação.



Fonte: Produção do próprio autor.

Da mesma forma como foi obtida a forma de onda da tensão para o interruptor SpA, foi obtida para S02A, ou seja, entre o *Drain* e o *Source*. Esta forma de onda é observada na Figura 121.

Figura 121 - Forma de onda da tensão entre o *Drain* e o *Source* do interruptor S02A.

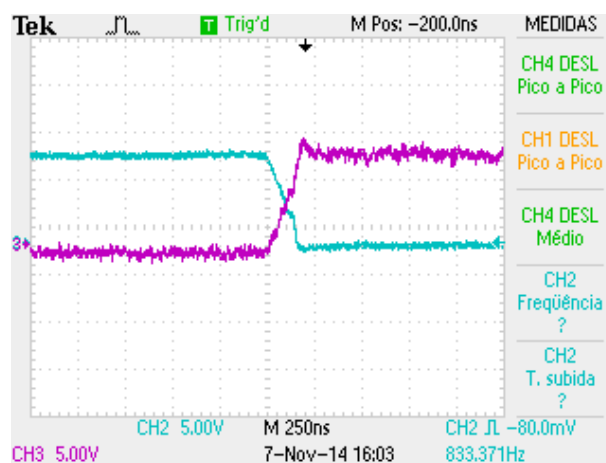


Fonte: Produção do próprio autor.

Observa-se que este interruptor apresenta forma de onda complementar para apenas meio período do sinal de referência, ou seja, para quando o sinal de referência é maior que zero. Para os interruptores SnA e S01A foram observadas formas de ondas semelhantes, defasadas em 180° em relação às formas de ondas mostradas anteriormente.

Na Figura 122 são mostradas as formas de onda entre o *Gate* e o *Source* dos interruptores SpA e S02A. Pode-se verificar que entre os sinais complementares não é verificado o tempo morto, devido á característica capacitiva dos interruptores.

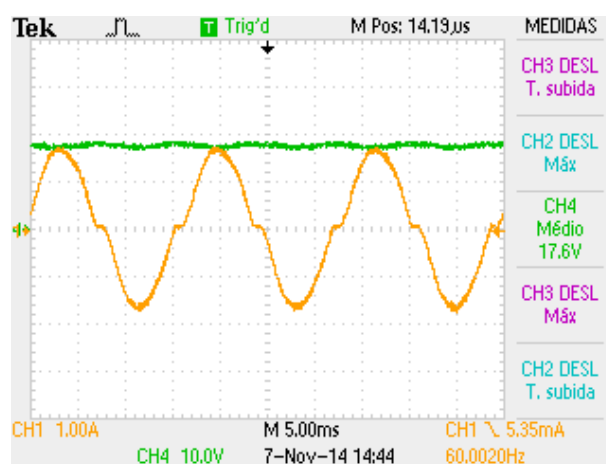
Figura 122 - Verificação da inexistência de tempo morto entre os sinais dos interruptores SpA e S02A.



Fonte: Produção do próprio autor.

Foram então obtidas as formas de onda da corrente e da tensão do barramento CC. Na Figura 123 podem ser observadas as formas de onda. Percebe-se que a corrente de saída do NPCm apresenta uma distorção quando passa por zero, assim como na simulação, que ocorre devido ao tempo morto entre os canais de PWM em um mesmo braço do conversor. Este tempo morto, por sua vez, ocorre devido ao pequeno nível de *offset* da portadora. Observa-se que a tensão do barramento CC é de 17,6V, enquanto que a corrente apresentava 1,6A de amplitude.

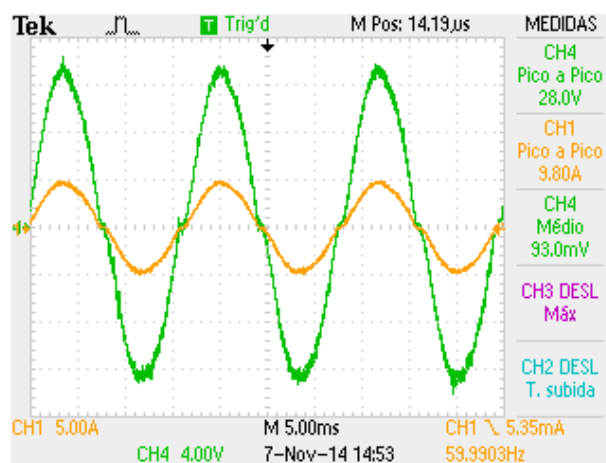
Figura 123 - Tensão do barramento CC e corrente de saída da fase A.



Fonte: Produção do próprio autor.

Na Figura 124 são vistos as formas de ondas da tensão e a corrente de saída da fase A do conversor. Observa-se que a tensão do barramento CC foi aumentada, de forma que a tensão de saída apresenta amplitude de aproximadamente 14V, e a corrente de 5A.

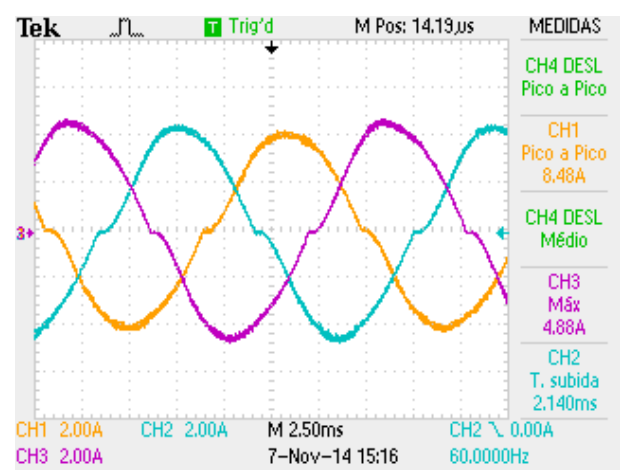
Figura 124 - Tensão e corrente de saída da fase A do NPCm.



Fonte: Produção do próprio autor.

Para mostrar o funcionamento das três fases, a Figura 125 apresenta as correntes obtidas das fases do conversor. É possível perceber diferenças nas formas de onda nas fases, devido a diferenças dos ponteiros de corrente do osciloscópio utilizados na obtenção dos dados.

Figura 125 - Correntes das três fases do NPCm.



Fonte: Produção do próprio autor.

7. CONCLUSÕES

Neste trabalho de conclusão de curso foram estudadas ferramentas disponíveis no *software* PSIM[®]. Utilizando os elementos do *SimCoder*, é possível realizar simulações de circuitos de eletrônica de potência, em que são realizadas configurações de PWM, entradas e saídas digitais, conversor A/D, entre outros, para a aplicação em um DSC. O PSIM[®] é capaz de gerar código em linguagem C para o DSC TMS320F28335, da *Texas Instruments*[®], podendo ser utilizado no *software Code Composer Studio* sem que sejam feitas modificações no código. Este trabalho foi elaborado para ser um guia para futuras aplicações destas ferramentas, apresentando os passos para suas aplicações. Os objetivos deste trabalho foram relacionados no Capítulo 1.

No Capítulo 2 foram estudados conversores que apresentam interruptores, entre eles o Conversor Buck e algumas topologias de inversores. Para o Conversor Buck, foi obtida a função de transferência da tensão em função da razão cíclica. Para os inversores, iniciou-se mostrando a topologia com dois interruptores e em seguida aumentando o número de interruptores, aumentando também o número de fases e por consequência a complexidade do comando dos conversores.

No Capítulo 3 foram apresentados os blocos do *SimCoder*, seus parâmetros de configurações e seu funcionamento. Pode-se destacar entre estes alguns que são mais utilizados, como por exemplo, os moduladores de PWM, as entradas e saídas digitais e o conversor A/D. Outros elementos que podem ser utilizados para geração de código também foram mostrados neste capítulo.

No Capítulo 4 foi mostrado o processo para se gerar o código em linguagem C no PSIM[®], e em seguida foi brevemente explicado a estrutura do código gerado. Foi observado que o CCS atualiza automaticamente o código gerado após modificações na simulação, apenas se o arquivo do código estiver na mesma área de trabalho utilizada no CCS. O processo para importar o código gerado no PSIM[®] para o CCS também foi explicado neste capítulo. Foram relacionados os arquivos gerados pelo PSIM[®] e as principais características do DSC utilizado, o TMS320F28335, da *Texas Instruments*[®].

No Capítulo 5 foram feitas simulações com os conversores explicados no Capítulo 2, utilizando os blocos de configurações do *SimCoder* para se gerar códigos em linguagem C. O Conversor Buck foi simulado em malha aberta e em malha fechada, com um controlador PI, enquanto que os conversores CC-CA foram simulados apenas em malha aberta, com referências senoidais.

Por fim, no Capítulo 6 foram feitos testes com o DSC TMS320F28335. Foram feitas simulações no PSIM[®] com os blocos mais utilizados, com a finalidade de testá-los individualmente. Foram obtidas formas de ondas utilizando osciloscópios e verificadas diversas características importantes dos moduladores de PWM, do conversor A/D e das entradas e saídas digitais. Ainda no Capítulo 6 foi feito um estudo do NPCm, um conversor cc-ca reversível trifásico. Utilizando os blocos para geração de código, foi feita a simulação do conversor. Nesta etapa foram encontradas algumas dificuldades para o comando, pois foi utilizado um conversor que já se encontrava pronto e as especificações de seu projeto deveriam ser seguidas. A principal dificuldade foi com os blocos de PWM, que deveriam seguir a ordem dos canais especificados. Porém foi possível realizar a simulação modificando a modulação *In-Phase Disposition*, utilizando apenas uma portadora triangular e utilizando duas referências senoidais para cada fase, defasadas em 180°. Posteriormente foi gerado o código em linguagem C e implementado no DSC encontrado no conversor. Utilizando um osciloscópio, foram obtidas diversas formas de onda dos sinais de comando nos interruptores, da corrente e da tensão de saída do NPCm, verificando a funcionalidade do código gerado.

A ferramenta *SimCoder* se mostrou eficiente para configurar o DSC. Porém ainda é uma ferramenta em desenvolvimento e precisa ser mais estudada. Como sugestão para futuros trabalhos, destaca-se a pesquisa e a realização de testes dos blocos SCI (*Serial Communication Interface*) e SPI (*Serial Peripheral Interface*), que não foram testados, pois estes são encontrados apenas na versão 9.1 do *software* PSIM[®], sendo utilizada neste trabalho a versão anterior. Outro ponto a ser sugerido é o teste prático do controle feito para o Conversor Buck e para outros conversores.

REFERÊNCIAS

AHMED, A. **Eletrônica de Potência**. São Paulo: Prentice Hall, 2000.

BARBI, I. **Eletrônica de Potência**. 6ª. ed. Florianópolis: [s.n.], 2006.

HEERDT, J. A. **Carga Eletrônica Ativa Trifásica**. Florianópolis: [s.n.], 2013.

MARTINS, D.; BARBI, I. **Conversores CC-CC Básicos Não Isolados**. 2ª. ed. Florianópolis: [s.n.], 2006.

POWERSIM. **SimCoder User's Guide**. [S.l.]: [s.n.], 2010.

POWERSIM. **PSIM User's Guide**. [S.l.]: [s.n.], 2013.

TEXAS INSTRUMENTS. **Code Composer Studio v5 User's Guide**. [S.l.]: [s.n.], 2011.

TEXAS INSTRUMENTS. **TMS320F28335 Data Manual**. [S.l.]: [s.n.], 2011.