Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Engenharia Elétrica

Trabalho de Conclusão de Curso

Simulação em Tempo Real Aplicada ao Controle de Máquina CC e Inversor Monofásico Conectado

Pedro Henrique Barcellos Linhares

João Monlevade, MG
2021

Pedro Henrique Barcellos Linhares

Simulação em Tempo Real Aplicada ao Controle de Máquina CC e Inversor Monofásico Conectado

Trabalho de Conclusão de curso apresentado à Universidade Federal de Ouro Preto como parte dos requisitos para obtenção do Título de Bacharel em Engenharia Elétrica pelo Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto.
Orientador: Prof Dr. Renan Fernandes Bastos

Universidade Federal de Ouro Preto
João Monlevade
2021

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

L755s Linhares, Pedro Henrique Barcellos.
Simulação em tempo real aplicada ao controle de máquina CC e inversor monofásico conectado. [manuscrito] / Pedro Henrique Barcellos Linhares. - 2022.
120 f.: il.: color., tab..

Orientador: Prof. Dr. Renan Fernandes Bastos.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia Elétrica .

1. Engenharia elétrica. 2. Máquinas elétricas - Corrente contínua. 3. Controladores elétricos. 4. Inversores elétricos. I. Bastos, Renan Fernandes. II. Universidade Federal de Ouro Preto. III. Título.

CDU 621.313.2

Bibliotecário(a) Responsável: Sione Galvão Rodrigues - CRB6 / 2526

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE ENGENHARIA ELETRICA

FOLHA DE APROVAÇÃO

Pedro Henrique Barcellos Linhares

Simulação em Tempo Real Aplicada ao Controle de Máquina CC e Inversor Monofásico Conectado

Monografia apresentada ao Curso de Engenharia Elétrica da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Engenheiro Eletricista

Aprovada em 14 de janeiro de 2022

Membros da banca

Dr - Renan Fernandes Bastos - Orientador(a) - Universidade Federal de Ouro Preto
Dr - Welbert Alves Rodrigues - Universidade Federal de Ouro Preto
Dr - Igor Dias Neto de Souza - Universidade Federal de Ouro Preto

Renan Fernandes Bastos, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 14/01/2022

Documento assinado eletronicamente por **Renan Fernandes Bastos**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 20/01/2022, às 17:35, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0270575** e o código CRC **2EB51063**.

Referência: Caso responda este documento, indicar expressamente o Processo nº 23109.000807/2022-53 SEI nº 0270575

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: - www.ufop.br

"Sábio é aquele que conhece os limites da própria ignorância"
– Sócrates

Resumo

Esta monografia apresenta um estudo sobre a técnica de simulação em tempo real, conhecida como simulação *hardware in the loop*, para validação de controladores aplicados em uma máquina CC e inversor monofásico. A simulação é feita discretizando os controladores e equações dinâmicas dos sistemas em *hardwares* distintos fechando um *loop* de simulação. Um controlador de corrente e velocidade em cascata é implementado para a máquina, e um controlador de corrente injetada para o inversor. O *hardware* utilizado para os controladores é o ESP32, já o *hardware* responsável pela simulação da máquina e inversor é o DSP da *Texas Instruments* 28379D. Todos os controladores são projetados via *MatLab* através do *sisotool* com base nos modelos dinâmicos específicos de cada sistema. Os controladores são testados e analisados em simulações computacionais tradicionais via *Psim* utilizando dois blocos *C Block*, contendo os códigos referentes ao controlador e as equações do sistema em questão discretizadas, respectivamente. Após validação dos controladores via simulações tradicionais, os mesmos são validados na simulação em tempo real. Os códigos são implementados em linguagem C utilizando as plataformas Arduino, para os controladores, e *Code Composer* da *Texas Instruments* para a máquina e inversor. Um conversor *boost* também fora simulado utilizando a técnica HIL, porém, por não fazer parte da proposta inicial, seus resultados estão presentes no appêndice A do trabalho.

Palavras-chave: *Hardware in the loop*, Inversor monofásico, Corrente injetada, Máquina de corrente contínua, Controladores PI, Simulação em tempo real, Controle, *Psim*, *Matlab*, ESP32, 28379D, *boost*.

Abstract

This monography presents a study on a real time simulation technique, known as hardware in the loop simulation, for validating controllers applied in a DC machine and a single phase inverter. To perform the simulation, the controllers and the dynamic equations are discretized into two different hardwares closing a simlation loop. A current and speed controller in cascate is implemented for the DC machine while a injected current controller is implemented for the inverter. The hardware used for the controllers is the ESP32, the hardware responsible to simulate the machine and inverter is the texas instrument's DSP 28379D. Every controller is designed via Matlab's sisotool based on the system's specifc dynamic model. Then the designed controllers are tested and analized in traditional computational simulation via Psim using two C block blocks, containing the controllers' and the system's specific dynamic model discretized codes. After the controllers' validation via traditional simulation methods, the controllers are validated in the real time simulation method. The codes are implemented in C language using Arduino's plataform, for the controllers, and the Texas Instrument's plataform, Code Composer, for the DC machine and inverter. A boost converter were also simulated using the HIL technicque, it's results can be found in the appendix at the end of this paper.

Keywords: Hardware in the loop, Separately excited direct current machine, Single phase inverter, Current injected, Real time simulation, Controller, *boost*.

Lista de ilustrações

Lista de tabelas

Lista de Siglas

CA	Corrente Alternada
CC	Corrente Contínua
MIT	Motor de Indução Trifásico
TI	Texas Instruments
PI	Proporcional Integral
HIL	Hardware in the Loop
ω_s	Velocidade angular síncrona
ω_r	Velocidade angular do rotor
i_a	Corrente de armadura da máquina CC
i_f	Corrente de campo da máquina CC
E_a	Tensão induzida na armadura da máquina CC
L_{af}	Indutância mútua entre armadura e campo
J	Momento de inércia
PWM	Modulação por largura de pulso
SPWM	Modulação por largura de pulso senoidal
DAC	Conversor analógico digital

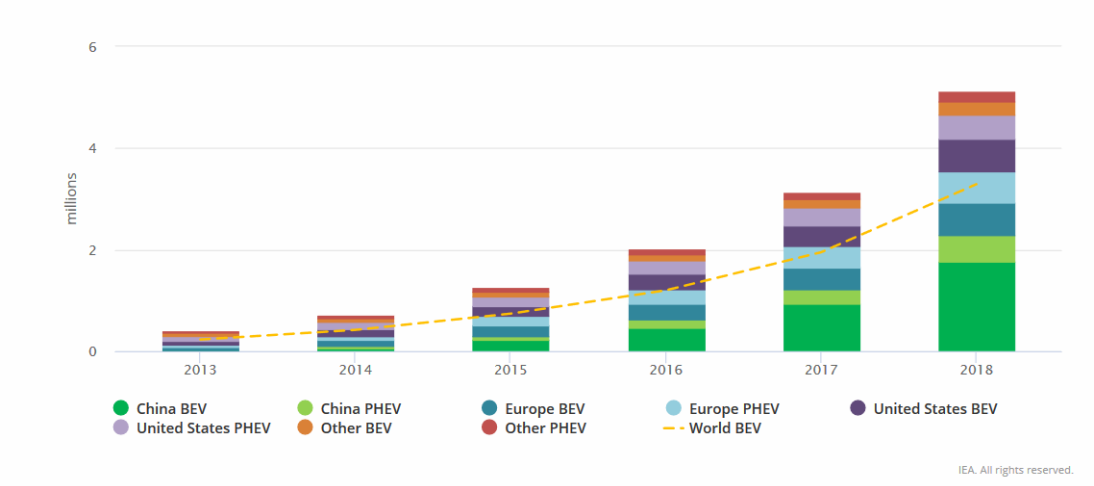
Sumário

1 Introdução

1.1 Contextualização

Máquinas elétricas são essenciais para industrias dos mais diversos setores, tais como, locomotivo, automotivo, siderúrgico, elétrico (WEG, 2012). Um dos grandes mercados com uma demanda crescente é o setor automotivo, crescendo rapidamente com o passar dos anos, a Figura 1 ilustra este crescimento, e que segundo IEA (2019), estima-se que os veículos elétricos representem cerca de 30% do total de carros nas ruas do mundo até 2030.

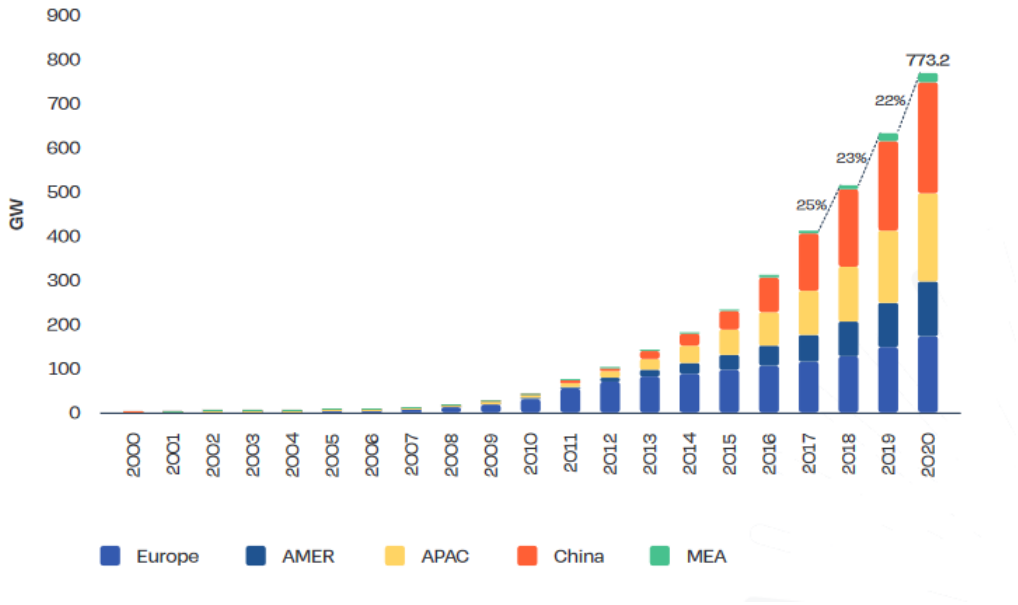
Figura 1 – Crescimento de veículos elétricos no mundo.



Fonte: IEA (2019)

Estima-se, segundo IEA (2021), que no ano de 2020 o crescimento de geração fotovoltaica tenha crescido 22% em relação ao ano de 2019, um valor recorde conforme ilustrado na Figura 2. É esperado, segundo IEA (2021), que para atender exigências climáticas do *Net Zero Emissions*, que visa zerar as emissões de poluentes no processo de geração até o ano de 2050, que até o ano 2030, deva ser mantida uma média de crescimento da capacidade instalada de 24% por ano.

Figura 2 – Capacidade total instalada de geração solar.



Fonte: HEMETSBERGER, SCHMELA e CHIANETTA (2021)

Dada a importância que as máquinas e inversores representam para a indústria mundial, técnicas de controle que visam melhorar o desempenho e eficiência do funcionamento destes equipamentos, são estudadas cada vez mais (RASHID, 2014).

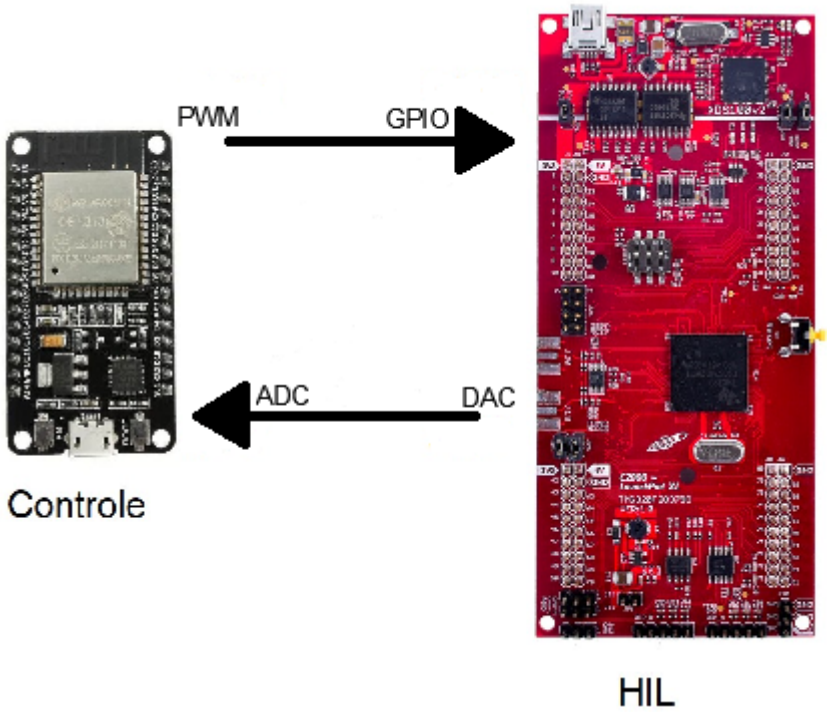
Segundo Lu et al. (2007), a técnica conhecida como simulação *hardware in the loop*, se trata de uma forma efetiva de se testar os controladores reais dando maior realismo à simulação, que é feita em tempo real ao contrário da simulação tradicional que simula em modelos puramente matemáticos via $software$.

A proposta desta técnica é a utilização de *hardwares* no *loop* da malha de controle, conforme Figura 3, onde o primeiro *hardware* atua como controlador e o segundo como a planta do sistema. O *hardware* atuando como o controlador, lê as variáveis de referência do *hardware* que simula o sistema, através das portas DAC, aplicando as ações de controle necessárias, e envia um sinal em forma de onda quadrada (*Duty Cycle*) responsável pela comutação adequada das chaves semicondutoras e correção das variáveis do sistema. Por não realizar o teste diretamente nas máquinas e inversores, não existe o risco de danos aos mesmos, caso o controlador não tenha sido implementado da maneira adequada, e torna possível, ainda, testes de casos extremos antes que o controlador possa ser implementado no sistema real.

O trabalho apresenta uma proposta de baixo custo para a realização da simulação HIL. Utilizando o ESP32 atuando como controlador, escolhido devido ao seu baixo custo e sua robustez na execução de operações matemáticas e pela possibilidade de se programar em C++. O segundo *hardware* escolhido foi o DSP F28379D simulando a máquina CC e o inversor monofásico, selecionado pela velocidade de clock de 200 MHz e baixa latência

e pela possibilidade de se programar, também, em C++.

Figura 3 – Estrutura genérica do HIL ESP32 e DSP.



Fonte: Autor

1.2 Objetivos

O trabalho tem por objetivo a validação de controladores aplicados em máquinas elétricas e inversores através de uma simulação em tempo real conhecida como *hardware in the loop*. Para tal simulação, é necessário fazer a modelagem do sistema, máquina, inversor e controladores, inserindo-os em dois *hardwares* distintos, simulando, respectivamente, a dinâmica da máquina ou inversor em um dos *hardwares* e seus respectivos controladores em outro.

1.2.1 Objetivos específicos

- Descrição dos modelos matemáticos;
- Validação dos modelos;
- Projeto dos controladores em ambiente simulado;
- Validação dos Controladores em ambiente simulado computacionalmente;
- Teste dos controladores em malha fechada com o modelo HIL.

1.3 Estrutura do Trabalho

O trabalho está organizado da seguinte maneira.

- Capítulo 1: Apresenta a contextualização do tema e objetivos do trabalho.
- Capítulo 2: Apresenta a revisão bibliográfica, explanando conceitos sobre o tema abordado e diferentes modelos de controle.
- Capítulo 3: Apresenta a metodologia utilizada.
- Capítulo 4: Apresenta os resultados e análises das implementações.
- Capítulo 5: Apresenta as conclusões acerca do trabalho.

2 Revisão Bibliográfica

Este capítulo tem por finalidade, apresentar uma revisão dos conceitos e técnicas referentes ao acionamento de máquina elétrica de corrente contínua de excitação independente, inversores, controladores, discretização e HIL (*hardware in the loop*).

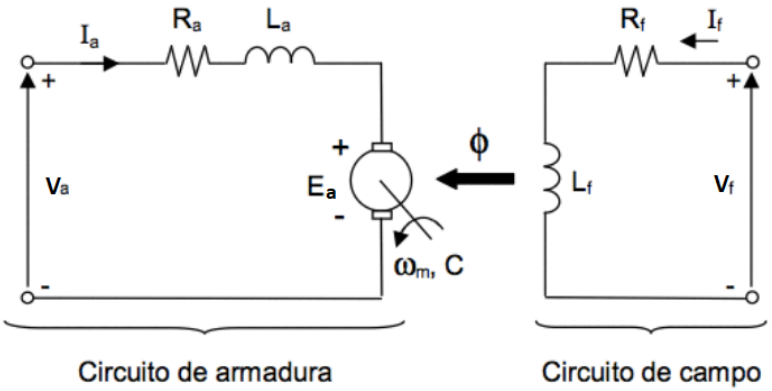
2.1 Máquina elétrica de corrente contínua de excitação independente

Máquinas elétricas são compostas, basicamente, de um estator, parte não móvel, e um rotor girante. Segundo Chapman (2013), máquinas elétricas são dispositivos conversores de energia, trabalhando como motores quando alimentados a partir de uma fonte de energia elétrica, e como geradores quando uma energia mecânica é aplicada ao seu eixo, fazendo-o girar.

Em uma máquina CC, além do estator e rotor, existe também a presença de comutadores, responsáveis por retificar tensões em corrente alternada produzidas pelos enrolamentos de armadura que giram junto ao rotor, concatenando fluxo CC (UMANS, 2014).

Existem variados tipos de máquinas CC, neste trabalho, porém, será considerado apenas a máquina CC de excitação independente. Nesta conFiguração os enrolamentos de campo e armadura são alimentados de forma indepenente, não havendo ligação elétrica entre eles, sendo necessária a alimentação do circuito de campo para em seguida alimentar a armadura. O circuito equivalente desta máquina é mostrado na Figura4.

Figura 4 – Máquina CC de Excitação Independente.



Fonte: Adaptado de Petry (2015)

onde

V_a = Tensão Corrente Contínua (CC) aplicada no terminal da armadura, em V.

i_a = Corrente CC que circula na armadura, em A.

i_f = Corrente CC que circula no campo, em A.

R_a = Resistência do enrolamento da armadura, em Ω.

L_a = Indutância do enrolamento de armadura, em H.

E_a = Tensão interna gerada na armadura, em V.

ϕ =Fluxo magnético CC gerado pela indutância de campo, em Wb.

ω_m = Velocidade de rotação da máquina, em rad/s.

L_f =Indutância de campo, em H.

R_f = Resistência do enrolamento do campo, em Ω.

V_f =Tensão CC aplicada ao terminal de campo, em V.

As equações dos circuitos de armdura e de campo são descritas por Chapman (2013), respectivamente:

$$V_a = i_a R_a + L_a \frac{di_a}{dt} + E_a \tag{2.1}$$

$$V_f = i_f R_f + L_f \frac{di_f}{dt} \tag{2.2}$$

A tensão E_a gerada, é dada por:

$$E_a = K_a \phi \omega_r = L_{a_f} i_f \omega_r \tag{2.3}$$

Onde L_{a_f} é a indutância mútua entre campo e armadura dada em H, e K_a uma constante relacionada ao enrolamento da máquina,

A dinâmica de velocidade da máquina é descrita conforme a equação 2.4:

$$\sum T = J \frac{d\omega_r}{dt} = T_e - T_l - B\omega_r \tag{2.4}$$

O T_l, depende da natureza da carga acoplada ao eixo, o T_e é diretamente proporcional às correntes de armadura e campo, B é o atrito de fricção devido ao fluido do rolamento, J é o momento de inércia da máquina. A equação 2.5, expressa o torque T_e. Como mencionado anteriormente, primeiramente excita-se o campo permanecendo com uma i_f constante, o torque produzido será dado pela variação da corrente de armadura, i_a:

$$T_e = K_a \phi i_a = L_{a_f} i_f i_a \tag{2.5}$$

2.1.0.1 Modelo em espaço de estados da máquina CC de excitação separada

As variáveis capazes de descrever todo o comportamento do motor CC, também chamadas variáveis de estados, são i_a, i_f e ω_r. As equações em espaço de estados, obtidas a partir das equações 2.1, 2.2, 2.4, são:

$$\frac{di_a}{dt} = \frac{1}{L_a}(V_a - i_a R_a - L_{af} i_f \omega_r) \tag{2.6}$$

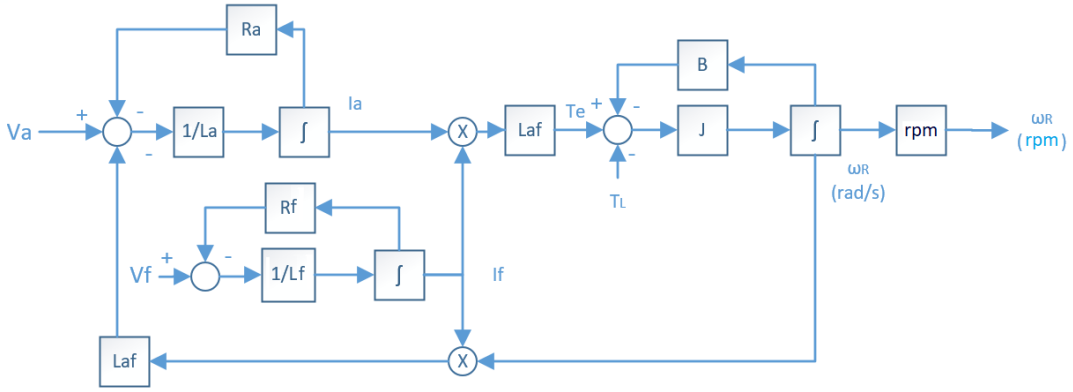
$$\frac{di_f}{dt} = \frac{1}{L_a}(V_f - i_f R_f) \tag{2.7}$$

$$\frac{d\omega_r}{dt} = \frac{1}{J}(T_e - T_l - B\omega_r) \tag{2.8}$$

2.1.0.2 Diagrama de blocos da máquina CC

O diagrama de blocos da máquina, obtido através das equações do modelo em espaço de estados é mostrado na Figura 5.

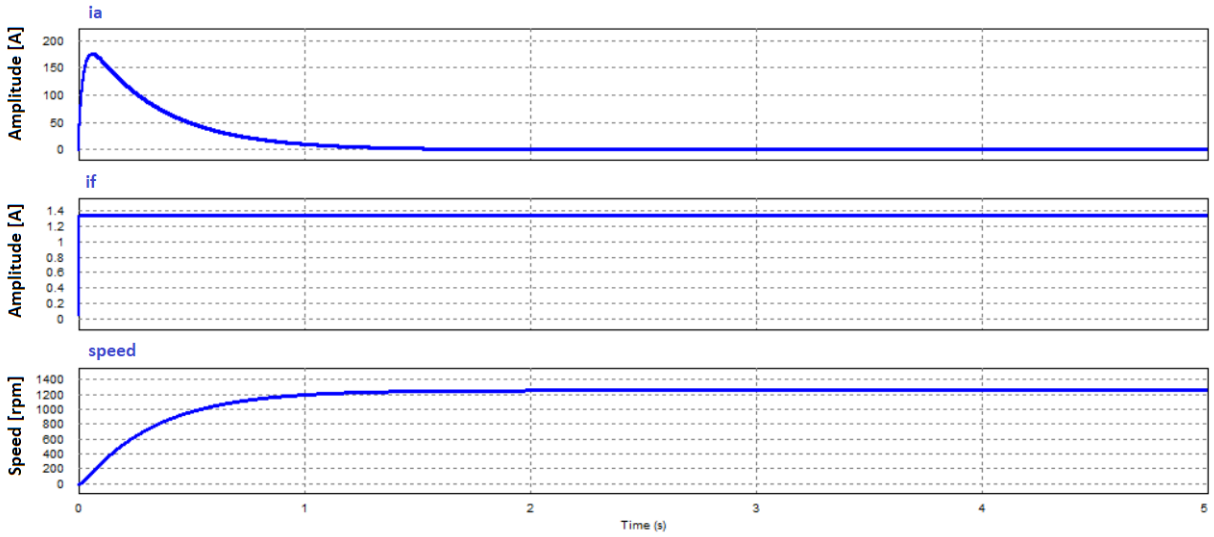
Figura 5 – Diagrama de blocos de uma máquina CC de excitação independente.



Fonte: Autor

A Figura 6 ilustra o comportamento das correntes i_a e i_f bem como a velocidade em rpm de uma máquina CC de excitação independente dado um degrau de tensão Va = 100 V e Vf = 100 V, para a máquina cujos parâmetros constam na Tabela 1.

Figura 6 – Dinâmica da máquina CC.



Fonte: Autor

Tabela 1 – Dados de placa da máquina CC..

Dados de Placa	
R_a	0,5 Ω
R_f	75 Ω
L_a	10 mH
L_f	20 mH
J	0,4 kg.m^2
$V_a^{nominal}$	120 V
i_a	10 A
i_f	1,6 A
n	1200 rpm
L_{af}	0,57196 H

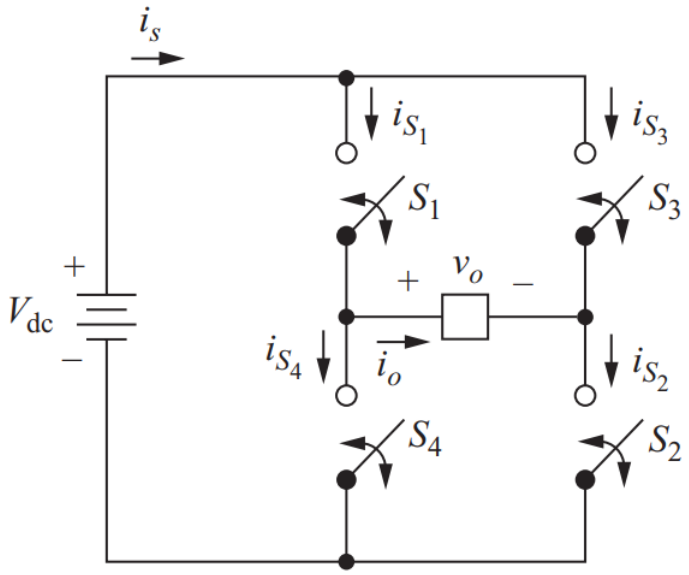
Fonte: Psim

2.2 Inversor monofásico

Circuito cuja finalidade consiste em converter tensões CC em tensões CA de frequência e amplitude definidas em projeto (RASHID, 2014). São utilizados em acionamentos de máquinas CA, como veículos elétricos (HART, 2011), bem como em sistemas fotovoltaicos (BOYLESTAD, 2011).

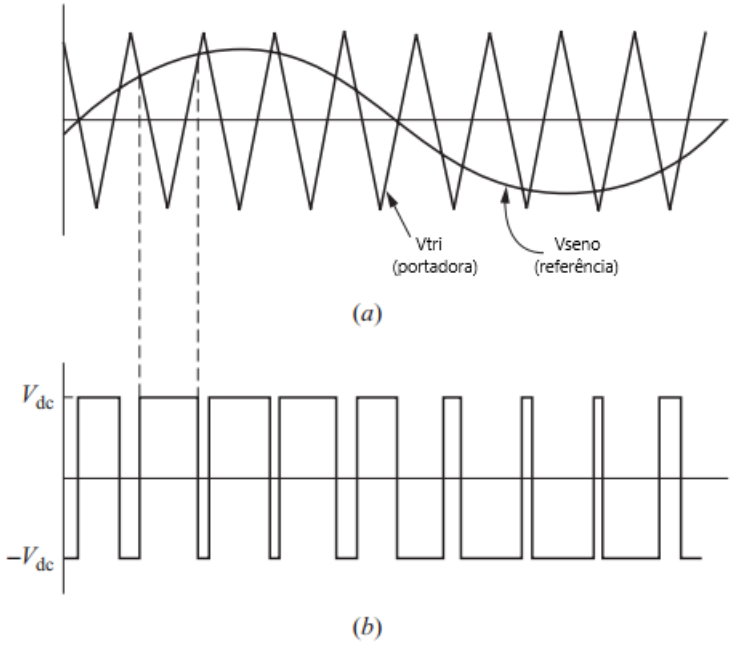
A Figura 7 representa o circuito do inversor monofásico conectado a uma carga genérica. Para se se obter uma tensão CA, as chaves S1, S2, S3 e S4 devem ser abertas e fechadas em sequências específicas via modulação por largura de pulso senoidal (SPWM). Um sinal senoidal é utilizado como referência, cuja frequência, f_r, e amplitude determinam a frequência e tensão eficaz de saída. A frequência de chaveamento, f_s, é determinada por uma portadora triangular (RASHID, 2014). Neste trabalho o chaveamento adotado é o conhecido como bipolar, cujos sinais de saída do processo de modulação são ilustrados na Figura 8.

Figura 7 – Circuito genérico do inversor monofásico.



Fonte: Hart (2011)

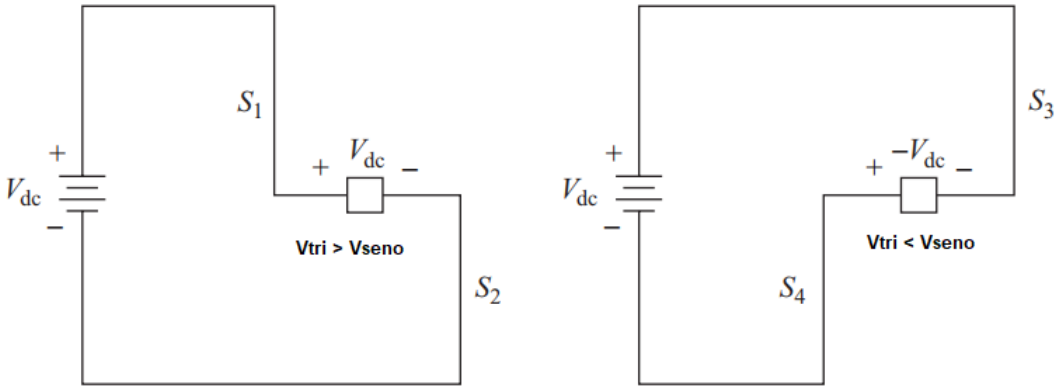
Figura 8 – PWM bipolar. (a) Referência senoidal e portadora triangular; (b) Saída $+V_{dc}$ para $V_{seno}>V_{tri}$ (chaves S1 e S2) e $-V_{dc}$ para $V_{seno}<V_{tri}$ (chaves S3 e S4).



Fonte: Adaptdado de Hart (2011)

A comutação das chaves semicondutoras, descrito pela Figura 8.b, é ilustrado na Figura 9

Figura 9 – Fechamento das chaves S1, S2, S3 e S4.



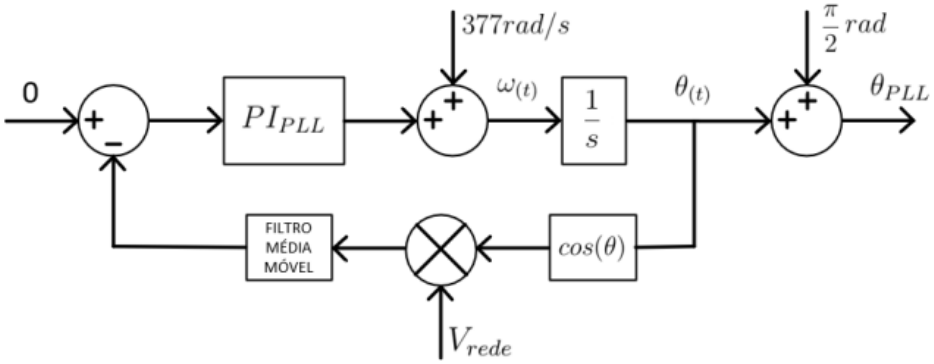
Fonte: Adaptado de Hart (2011)

2.3 Algoritmo PLL

O algoritmo tem por finalidade realizar a sincronização entre um sistema acessante e o sistema elétrico de potência (PADUA; DECKMAN, 2006). O ângulo obtido pelo

algoritmo, θ_{pll}, é utilizado para produzir um sinal senoidal unitário e em fase com a tensão de referência da rede, que multiplicado a um determinado valor constante, define um valor de pico de referência para o controlador da corrente injetada pelo acessante à rede elétrica.

Figura 10 – Algoritmo PLL.



Fonte: Adaptado de AGUIAR (2013)

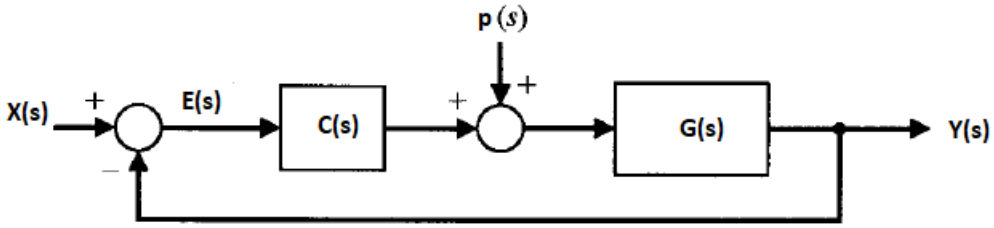
O algoritmo gera uma senoide com um $\theta_{(t)}$ ortogonal ao sinal V_{rede} lido, quando a diferença entre o *setpoint* e a média móvel são iguais a 0, a ortogonalidade ocorre devido a integração de $\omega_{(t)}$ para a obtenção do $\theta(t)$, $d\theta_{(t)}/dt = \omega_{(t)}$ (AGUIAR, 2013). Como o sinal gerado pelo algoritmo é ortogonal ao sinal real, é necessário somar $\pi/2$ rad, para se obter uma referência, θ_{PLL} perfeitamente em fase com a rede, utilizado no controlador de corrente. Neste trabalho, o filtro média móvel possui uma amostra N igual a 200 pontos.

2.4 Controladores

Controladores são dispositivos responsáveis por manter certas variáveis de interesse de algum sistema em algum valor desejado, chamado valor de referência. Um controlador realimentado, ou controlador em malha fechada, é aquele que, a partir da diferença entre um valor de referência e o valor medido de uma variável de interesse do sistema sendo controlado, realiza uma ação de controle proporcional a essa diferença adequando a saída (FRANKLIN; POWELL; EMANI-NAEINI, 2014).

A Figura 11 apresenta um esquemático em diagrama de blocos ilustrando a estrutura de uma malha de controle.

Figura 11 – Ilustração em malha fechada do controle de um sistema G(s).



Fonte: Adaptado de Dorf e Bishop (2001)

onde
G(s) = Modelo do Sistema.
C(s) = Controlador.
X(s) = Entrada.
Y(s) = Saída.
p(s) = Perturbações.
E(s) = Erro entre a entrada e saída.

2.4.1 Controlador PI

O controlador PI é composto por uma parcela proporcional e integral. A parcela proporcional é simplesmente um ganho, e tem a capacidade de diminuir mas não eliminar o erro em regime permanente, já a parcela integral, é a soma de todos os valores passados do erro. Une-se ambas características em um único controlador PI com o objetivo de reduzir o erro em regime permanente do sistema (FRANKLIN; POWELL; EMANI-NAEINI, 2014).

A equação que representa um controlador PI é:

$$u(t) = k_p e(t) + k_i \int_{t0}^{t} e(\tau) d\tau \tag{2.9}$$

O projeto de controladores é realizado no domínio da frequência, portanto, é necessário realizar a transformação das equações no espaço de estados por meio da transformada de Laplace, para que então, seja projetado o controlador com base na função de transferência do sistema. Aplicada a transformada de Laplace na equação 2.9 resulta em:

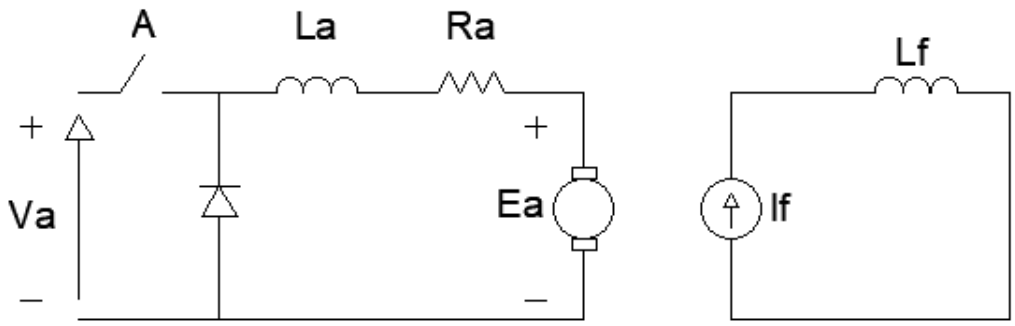
$$U(s) = k_p E(s) + \frac{k_i}{s} E(s) \tag{2.10}$$

2.4.2 Controle de velocidade e torque da máquina CC

Para a realização do controle, utiliza-se, o circuito conversor CC-CC conhecido como *chopper* de um quadrante, ilustrado na Figura 12. Nesta conFiguração, a máquina CC, atua como motor (UMANS, 2014). Uma técnica de controle utilizada, segundo Rashid

(2001), é o controlador em cascata contendo uma malha interna de controle de corrente, conforme ilustrado pela Figura 6.

Figura 12 – Circuito conversor *chopper* de um quadrante ligado a uma máquina CC.

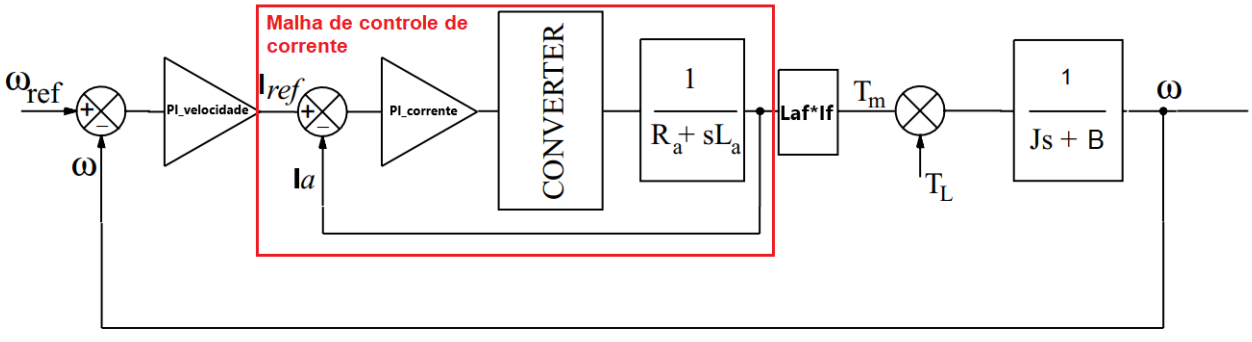


Fonte: Autor

Conforme descrito pela equação 2.5, o torque da máquina CC é diretamente proporcional à corrente de armadura I_a, considerando I_f constante, sendo, portanto, a variável de referência para se controlar o torque. O controlador PI, projetado específicamente para controle de torque, possui como entrada a diferença entre o valor de I_a de referência e medido, obtendo um valor de duty cycle, D, responsável por realizar o chaveamento dos $IGBT's$ via PWM, atuando para manter a corrente no valor de referência.

Para o controle de velocidade, o PI é projetado específicamente para tal controle, com base na equação 2.4, recebe como entrada a diferença entre uma velocidade de referência ω_{ref} e a velocidade ω_r medida. O erro é compensado pelo controlador obtendo o valor de I_{ref} utilizado no controlador de corrente. Na prática é inviável realizar um controle sem incluir a malha interna de corrente, pois a corrente i_a, conforme descrito pela equação 2.8, é diretamente proporcional à velocidade, ou seja, variações na referência de velocidade resultarão em variações de corrente de armadura na máquina.

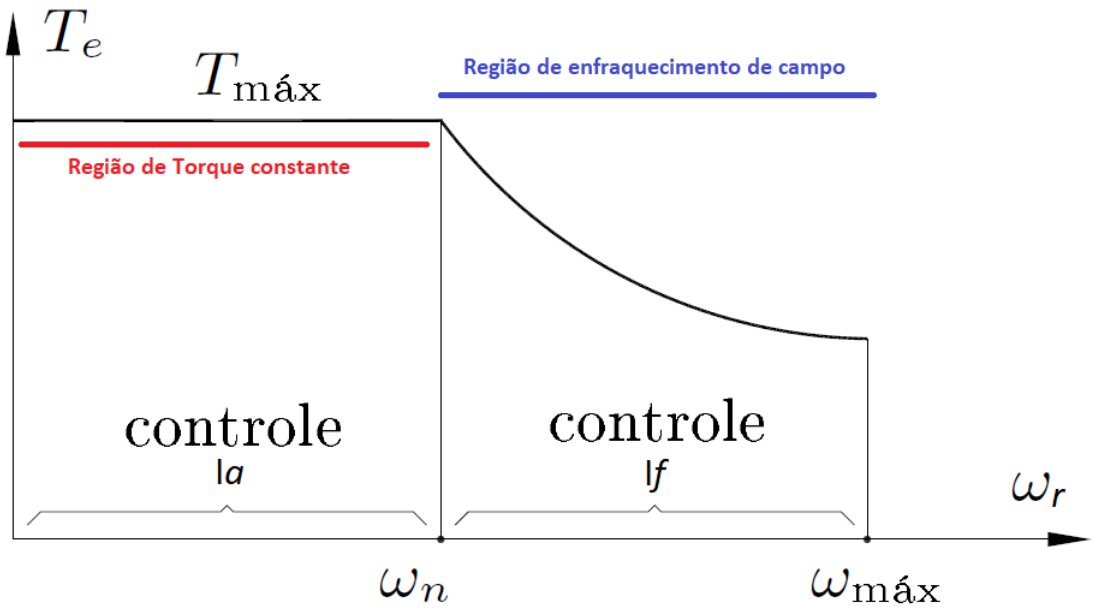
Figura 13 – Diagrama para controle de velocidade em máquina CC de excitação separada.



Fonte: Adaptado de Rashid (2001)

Em máquinas elétricas existem, segundo Bose (2015), diferentes regiões de operação, conforme ilustrado pela Figura 14. Neste trabalho, os controladores atuarão até a velocidade nominal da máquina, conhecida como região de torque constante, para velocidades mais elevadas, região conhecida como região de enfraquecimento de campo, é necessária a utilização de outros controladores que atuem na componente proporcional ao fluxo magnético i_f.

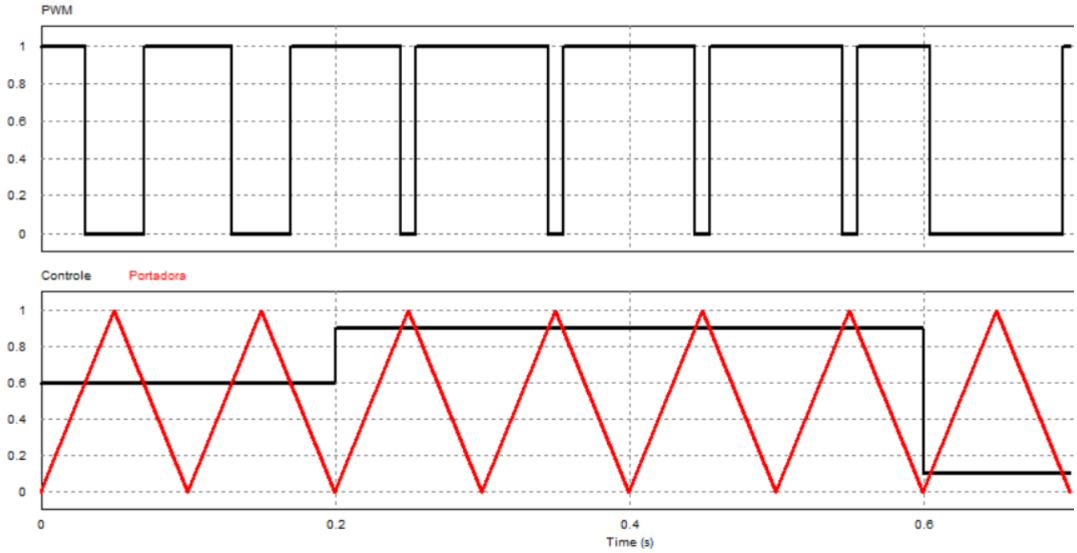
Figura 14 – Regiões de torque constante e de enfraquecimento de campo.



Fonte: Adaptado de Bim (2012)

A modulação PWM utilizada segue os mesmos requisitos do SPWM mencionados na sessão sobre inversores, porém, a natureza do sinal de referência, ou controle, não é senoidal e sim constante, conforme ilustrado pela Figura 15 (HART, 2011). A saída da modulação, é responsável pelo chaveamento do *chopper* ilustrado na Figura 12.

Figura 15 – Exemplo de modulação PWM com modificação da largura de pulso no instante 0,2s e 0,8s.



Fonte: Autor

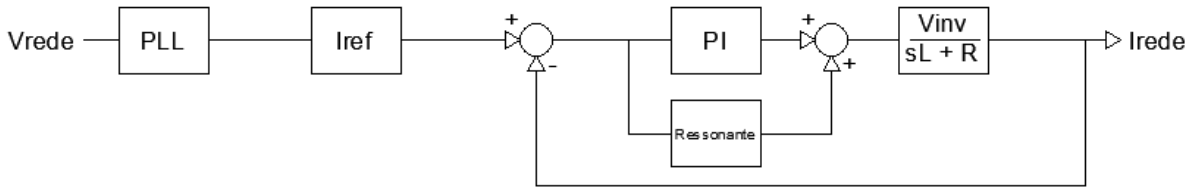
2.4.3 Controle inversor monofásico

Conforme descrito na seção 3, a saída do PLL produz um sinal perfeitamente em fase com o sinal de referência da rede, este por sua vez é multiplicado a um valor de referência de corrente.

Dada a natureza senoidal do sinal de referência, o controlador PI não é capaz de compensar completamente o erro em regime permanente, portanto, é necessário adicionar em paralelo ao controlador PI um controlador ressonante. Este controlador tem como objetivo adicionar um ganho, de forma a compensar a componente fundamental do sinal de controle de entrada (AGUIAR, 2013). A combinação entre PI e ressonante, gera o sinal de referência ilustrado pela Figura 8.a, responsável pelo chaveamento do inversor.

A Figura 16 ilustra o diagrama de blocos para o controle aplicado ao inversor monofásico.

Figura 16 – Diagrama para controle de corrente do inversor monofásico.

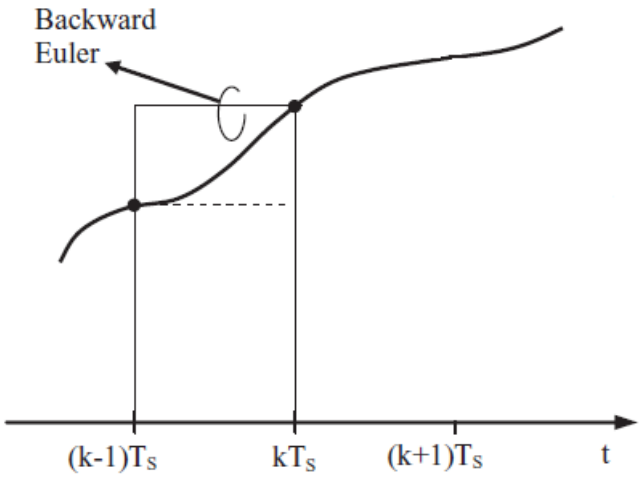


Fonte: Autor

2.4.4 Discretização *backwards*

A Figura 17 ilustra o método de discretização baseado em integração numérica, com o objetivo de realizar a computação de integrais por aproximações numéricas, ao invés de computá-las em tempo contínuo. Substitui-se a variável independente da transformada de laplace, s, na função de transferência do controlador pela variável z da transformada Z (BUSO; MATTAVELLI, 2006).

Figura 17 – Diagrama para controle de corrente do inversor monofásico.



Fonte: Adaptado de Buso e Mattavelli (2006)

A variável s da transformada de laplace é substituída conforme expressão 2.11, respeitando o limite de 3% de distorção causados pelo processo de discretização, conforme razão entre frequência de amostragem e a frequência do sistema, expressão 2.12.

$$s = \frac{z-1}{zT_s} \tag{2.11}$$

$$\frac{f_s}{f} > 20 \tag{2.12}$$

A expressão 2.11 aplicada a um integrador, expressão 2.13 resulta na expressão 2.17.

$$G_{integral}(s) = \frac{Y(s)}{X(s)} = \frac{k_i}{s} \tag{2.13}$$

$$(z-1)Y(k) = X(k)K_iT_sz \tag{2.14}$$

$$Y(k)z - Y(k) = X(k)K_iTsz \tag{2.15}$$

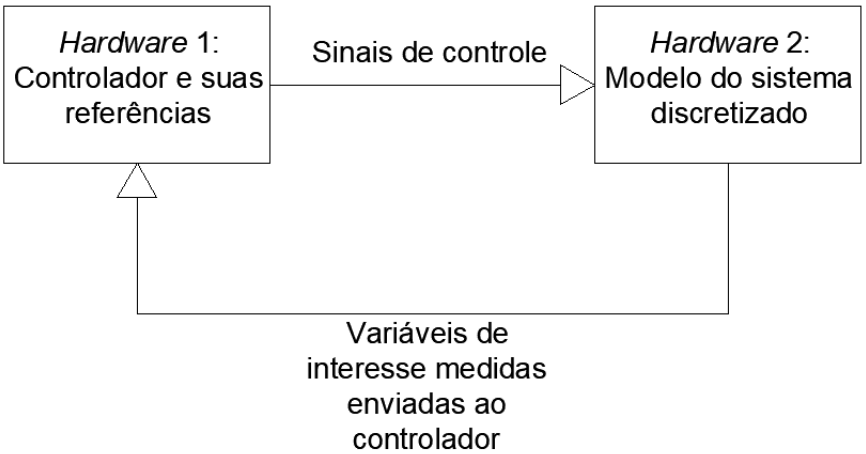
$$Y(k) - Y(k-1) = X(k)K_iT_s \tag{2.16}$$

$$Y(k) = Y(k-1) + X(k)K_iT_s \tag{2.17}$$

2.5 Simulação Hardware in the loop

Segundo Bacic (2005), trata-se de uma técnica de avaliação de controladores, onde um controlador real é conectado a outro hardware contendo o modelo do sistema onde o controlador atuará. Quando conectados, o controlador entende que está conectado a um sistema real, sendo possível testar o controlador em tempo real, sem cogitar a hipótese de danos ao equipamento real, evitando-se portanto, prejuízos materiais, financeiros e possíveis danos a saúde física de pessoas em campo. A Figura 18 ilustra, de forma simplificada, o processo de simulação HIL.

Figura 18 – Simulação *hardware in the loop*.



Fonte: Autor

Por utilizar *hardwares* reais, as simulações HIL apresentam grande vantagem em relação aos métodos de simulação tradicionais que utilizam apenas simulação computacional, dando mais realismo à simulação. Segundo Bastos et al. (2020), o modelo digital é

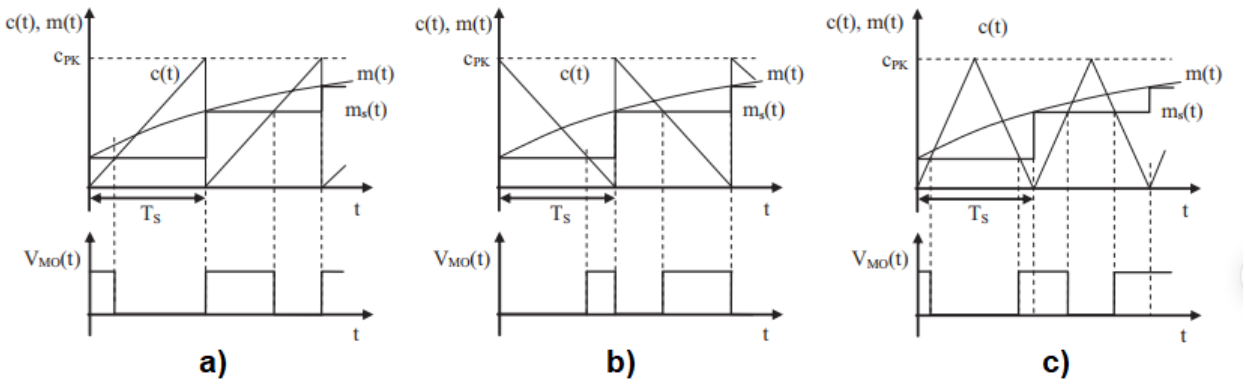
uma série de equações diferenciais que descrevem o comportamento dinâmico do sistema real que são processadas em períodos de amostragem bem curtos (T_s). Para se representar os sistemas de forma fidedigna recomenda-se períodos de amostragem da ordem de $T_s \ll 10\mu s$.

Esta técnica de simulação possui grande aplicabilidade, tais como qualidade de energia e controle eletrônico automotivo (LU et al., 2007) até gerenciamento de energia de um navio (TYPHOON-HIL, 2019). Para este trabalho, o foco será, conforme mencionado em seções anteriores, em motor CC e inversor monofásico.

Em eletrônica de potência, na geração de um sinal PWM, os dois tipos de portadora mais comuns a serem utilizados são, a triangular simétrica e a triangular dente de serra. Ao se utilizar a triangular simétrica, os processadores comerciais, são conFigurados de forma a gerar sinais de interrupção centralizados na forma de onda, conforme Figuras 19(c) e 20. Já nas triangulares dente de serra, o sinal de interrupção é gerado na borda de transição, conforme Figuras 19(a,b) e 21. Assim sendo, o formato da portadora tem impacto profundo no sinal amostrado pelo controlador discreto (BUSO; MATTAVELLI, 2006).

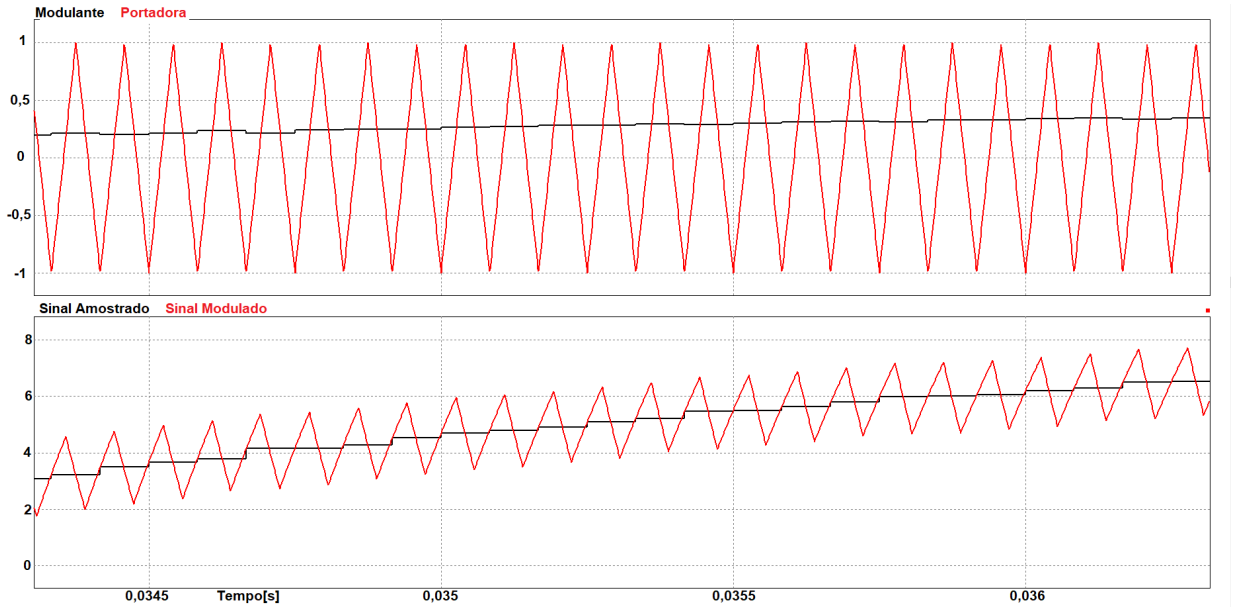
Na triangular simétrica, como a interrupção ocorre no pico da onda, a amostra do controlador ocorre exatamente na média do sinal triangular de corrente do sistema chaveado pelo PWM, Figura 20. Já a dente de serra, as amostras acontecem nos picos do sinal de corrente, criando um "$offset$" no valor médio discreto amostrado, Figura 21. Na implementação deste trabalho, devido a natureza do processador utilizado, apenas a portadora do tipo dente de serra estava disponível. Logo, como mencionado acima, os sinais de corrente amostrados pelo controlador discreto sempre contém um "$offset$" equivalente a metade do valor do "$ripple$".

Figura 19 – Amostragem de PWM: (a) Modulação por borda de descida, portadora dente de serra, (b) Modulação por borda de subida, portadora dente de serra, (c) Modulação por portadora triangular simétrica.



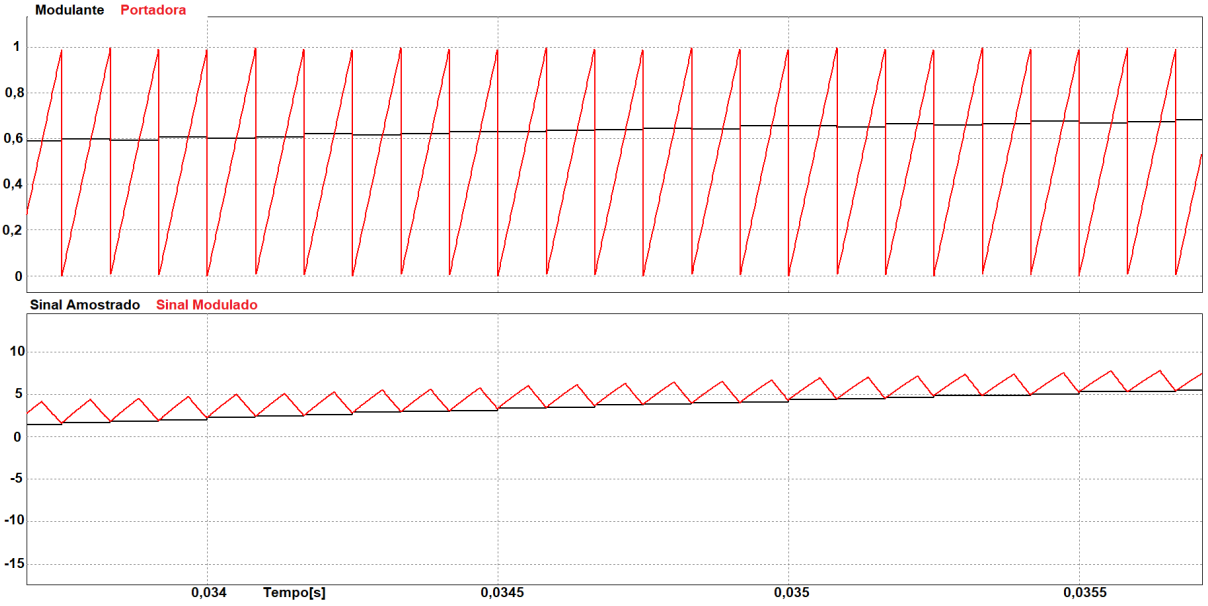
Fonte: Adaptado de Buso e Mattavelli (2006)

Figura 20 – Sinal amostrado em portadora triangular simétrica.



Fonte: Autor

Figura 21 – Sinal amostrado em portadora dente de serra, borda de descida.



Fonte: Autor

3 Metodologia

No decorrer deste capítulo serão descritos os procedimentos metodológicos implementados para cada estrutura de controle mencionadas no capítulo anterior. As simulações serão realizadas no *software* Powersim (Psim) utilizando dois blocos C contendo os códigos dos controladores discretizados em um primeiro, e em um segundo, o modelo discretizado dos sistemas conforme Figura 18. O projeto dos controladores é feito com base nas equações dinâmicas da máquina CC e inversor. A ferramenta *sisotool* do *software* Matlab é utilizada para determinação dos ganhos do controlador PI.

Para a estrutura experimental os controladores serão discretizados e inseridos no microcontrolador de baixo custo $ESP32$. Já os modelos HIL da máquina e do inversor serão discretizados e inseridos no DSP da *texas instruments*, modelo $LAUNCHXL-F28379D$. Os códigos dos controladores serão implementados no *software* do arduino utilizando linguagem C, já o DSP possui uma plataforma própria disponibilizada pela TI, conhecida como *Code Composer Studio*. Todos os códigos implementados estão anexados no fim do trabalho. Para coleta dos dados experimentais, as portas DAC (conversor analógico digital) são usadas para exportar as variáveis analógicas dos modelos discretos, sendo estas variáveis coletadas para medição utilizando um osciloscópio.

Este capítulo será dividido em duas subseções, a primeira relacionada à máquina CC e a segunda ao inversor monofásico.

3.1 Máquina CC

Os parâmetros da máquina CC utilizada, são os mesmos parâmetros base do *software* Psim, já listados na Tabela 1, apresentada no capítulo anterior.

3.1.1 Projeto do controlador de torque

Conforme ilustrado pelo diagrama de blocos da Figura 13, a planta, $G(s)$, para o controle de torque, é descrita pela expressão:

$$G_{cor}(s) = \frac{i_a(s)}{D(s)} = \frac{V_a}{sL_a + R_a} = \frac{V_a/R_a}{s\tau_e + 1} \tag{3.1}$$

A tensão de alimentação utilizada, V_s, é de 100 V, portanto, substituindo os valores da Tabela 1 na equação 3.1, a equação é escrita da seguinte maneira:

$$\frac{i_a(s)}{D(s)} = \frac{100}{0,01s + 0,5} \tag{3.2}$$

A partir da expressão 3.2, projeta-se o controlador PI, com o auxílio da ferramenta *sisotool*. Para o projeto, a constante de tempo elétrica da máquina, τ_e, expressa por 3.3,

é muito pequena e que portanto a planta de corrente é muito rápida, significando que o controlador precisará de uma largura de banda alta para que o mesmo responda de forma rápida às variações de comando. Segundo Villalva (2010) a largura de banda do sistema chaveado, deve ser inferior a um décimo da frequência de chaveamento. Para este trabalho, o chaveamento terá uma frequência, f_s, de 5kHz, portanto a largura de banda em rad/s deve ser:

$$\tau_e = \frac{L_a}{R_a} = 0,02 \tag{3.3}$$

$$\omega_b < \frac{2\pi 5000}{10} = 3141,6 \tag{3.4}$$

Adotados estes critérios, obtêm-se os ganhos k_p e k_i do controlador, conforme mostrado na expressão 3.5.

$$Pi_{cor} = k_p + \frac{k_i}{s} = 0,005 + \frac{3}{s} \tag{3.5}$$

A saída da parcela integral, bem como saída do controlador PI, devem ser limitadas entre zero e um, visto que a portadora possui, conforme Figura 15, a mesma amplitude. Valores superiores aos mencionados extrapolariam os limites toleráveis.

3.1.2 Projeto do controlador de velocidade

Na prática, dado que τ_e é muito pequena, é inviável realizar um controle exclusivo para velocidade, pois a corrente i_a, conforme descrito pela equação 2.8, é diretamente proporcional à velocidade, ou seja, variações na referência de velocidade resultarão em grandes variações transientes de corrente de armadura na máquina. Uma técnica de controle utilizada , segundo Rashid (2001), é o controlador em cascata contendo uma malha interna de controle de corrente, cujo referencial de corrente saí do controlador de velocidade da malha externa.

A partir do diagrama de blocos da Figura 13 e pela equação 2.8, considerando T_l igual a zero a planta $G(s)$ do sistema é dada por:

$$G_{vel}(s) = \frac{\omega_r}{D(s)} = \frac{T_e}{Js} = \frac{L_{af} i_f i_a}{Js} = \frac{9,1514}{0,4s} \tag{3.6}$$

A partir da expressão 3.6, projeta-se o controlador PI utilizando a ferramenta *sisotool*. Dada a constante de tempo mecânica da máquina expressa em 3.7, considerando B igual a zero a constante de tempo é muito alta. Como critério de sintonia da malha externa, é recomendado que esta tenha uma frequênca de corte, pelo menos dez vezes menor que a malha interna (VILLALVA, 2010).

$$\tau_m = \frac{J}{B} \tag{3.7}$$

Os dados referentes ao controlador de velocidade são mostrados na expressão 3.8.

$$Pi_{vel} = k_p + \frac{k_i}{s} = 0,196 + \frac{0,1862}{s} \tag{3.8}$$

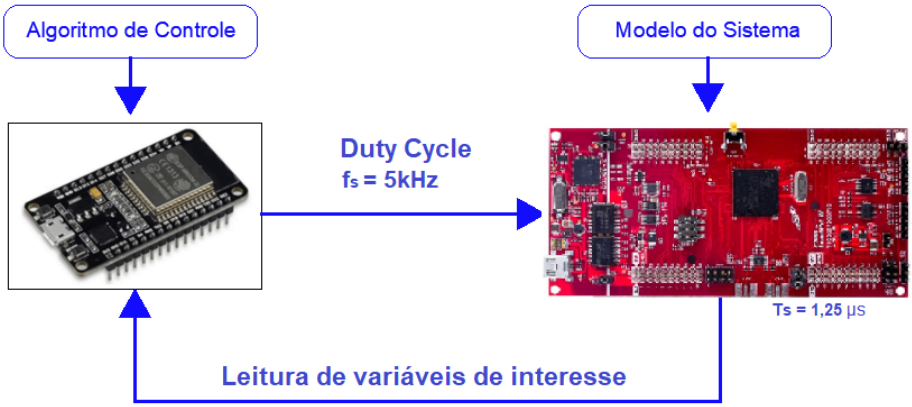
A saída do controlador de velocidade é a referência do controle de corrente, Figura 13, conforme mencionado na seção referente ao controle de máquina CC, essa saída deve ser limitada de modo a evitar grandes transitórios na planta de corrente. A parcela integral e a saída do PI de velocidade serão limitadas entre zero e trinta, isso garante que a corrente de armadura seja limitada no valor máximo de trinta amperes.

Para validar os controladores projetados, referências de velocidade distintas são aplicadas ao controlador. Bem como perturbações no eixo, para diferentes valores de T_l. Após validação dos controladores em ambiente de simulação computacional, é realizada a simulação HIL experimental.

A variação de torque na parte prática é realizada de maneira manual, o *software code composer*, possui uma área chamada *expression* em que variáveis utilizadas no corpo do programa podem ser monitoradas e alteradas, através da coluna *value*, durante a execução do mesmo.

A frequência máxima alcançada pelo HIL para o modelo em questão foi de 800 kHz, com T_s igual a 1,25 μs para uma simulação mais fiel (BASTOS et al., 2020). A Figura 22 mostra as conFigurações de tempo e frequência utilizados no experimento HIL para a máquina CC.

Figura 22 – Frequência de chaveamento enviada pelo ESP e período de amostragem do DSP.

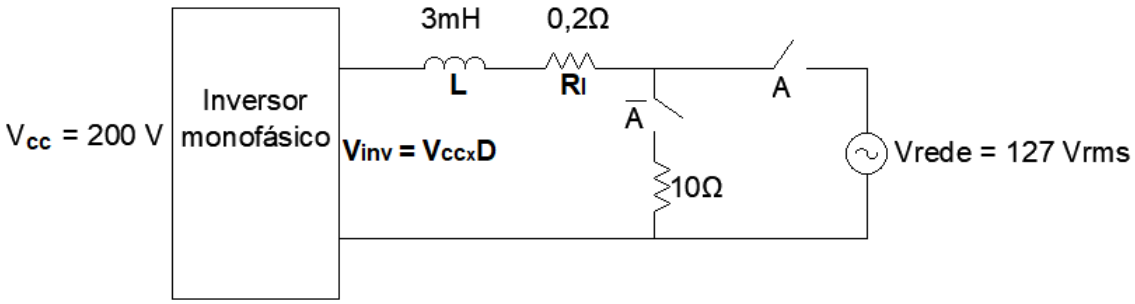


Fonte: Autor

3.2 Inversor monofásico

A Figura 23, apresenta o circuito simplificado do inversor conectado à rede. O controlador será testado conectado a uma carga de 10 Ω isolada da rede, utilizando Vrede como referência senoidal, quando A = 0. Na condição A=1, o inversor é testado para a situação conectado à rede. O chaveamento será feito em uma frequência, f_s, de 12 kHz. A frequência máxima alcançada pelo HIL para o modelo em questão foi de 1 MHz, com um T_s igual a $1\mu s$.

Figura 23 – Diagrama simplificado do inversor colectado à rede.



Fonte: Autor

A planta $G_{(s)}$ ilustrada na Figura 16, é obtida através da Figura 23, considerando o inversor conectado à rede, A = 1.

$$V_{linha}(t) = V_{inv}(t) - V_{rede}(t) = \Delta V(t) \tag{3.9}$$

$$V_{inv}(t) = V_{cc}D(t) \tag{3.10}$$

$$V_{linha}(t) = I_{rede}(t)Rl + L\frac{d_{I_{rede}(t)}}{dt} \tag{3.11}$$

Aplicando a transformada de Laplace na equação 3.11 e rearranjando a expressão:

$$\frac{I_{rede}(s)}{D(s)} = V_{inv}\frac{1}{sL + Rl} = \frac{200}{sL + Rl} = G_{inv}(s) \tag{3.12}$$

O ganho V_{inv} representa a tensão CC do barramento. Dada a função de transferência em malha aberta apresentada pela expressão 3.12, utiliza-se o *sisotool* para o projeto do controlador cujas características são mostradas na expressão 3.13, fora considerada uma largura de banda de 1885rad/s e uma margem de fase de 70°.

$$Pi_{inv} = k_p + \frac{k_i}{s} = 0,1 + \frac{20}{s} \tag{3.13}$$

A função de transferência do controlador ressonante é apresentado pela expressão 3.14.

$$G_{res}(s) = \frac{Y(s)}{erro(s)} \frac{(k_{res})s}{s^2 + Bs + \omega^2} = \frac{30s}{s^2 + 5s + (2\pi 60)^2} \tag{3.14}$$

Aplicando a discretização *backward* no ressonante, e manipulando as expressões, obtêm-se a expressão 3.15.

$$G(k) = \frac{(bT_s + 2)Y[k-1] - Y[k-2] + k_{res}T_s erro[k] - k_{res}T_s erro[k-1]}{\omega_0^2 T s^2 + bt_s + 1} \tag{3.15}$$

Por se tratar de um processo realizado por um *hardware* os valores constantes da expressão 3.15 são calculados no cabeçalho do código, visando eficiência de execução do programa evitando divisões constantes, ficando conforme a expressão 3.16.

$$G(k) = erro[k]C1 - erro[k-1]C2 - Y[k-2]C3 - Y[k-1]C4 \tag{3.16}$$

onde:

$$C1 = \frac{k_{res}T_s}{d} \tag{3.17}$$

$$C2 = \frac{k_{res}T_s}{d} \tag{3.18}$$

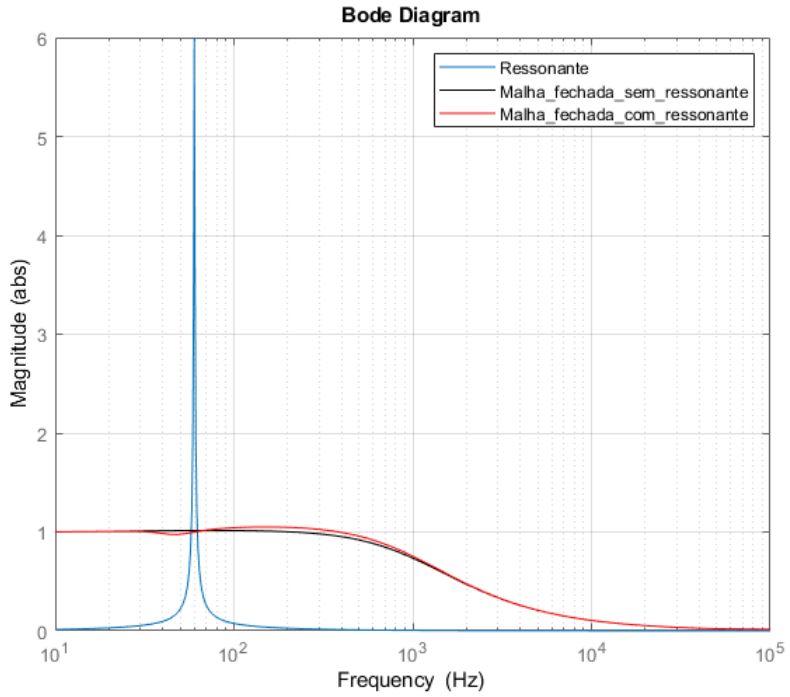
$$C3 = \frac{1}{d} \tag{3.19}$$

$$C4 = \frac{(-2 - T_s b)}{d} \tag{3.20}$$

$$d = 1 + T_s B + \omega_0 T_s^2 \tag{3.21}$$

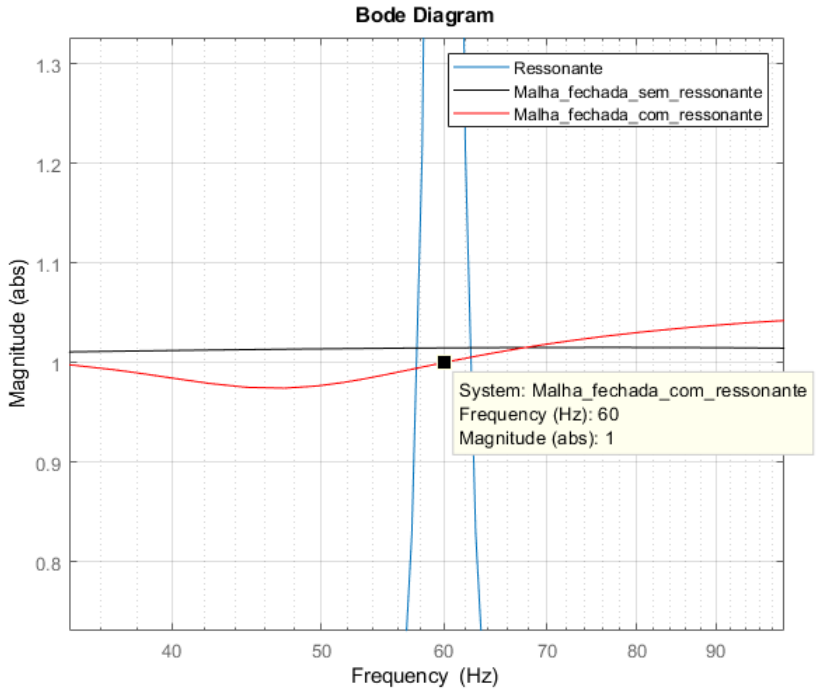
As Figuras 24 e 25 apresentam o diagrama de bode no domínio s para o sistema com e sem a presença do controlador ressonante. O detalhe da Figura 25, ilustra que a malha sem a presença do ressonante apresenta um ganho diferente do unitário, indicando a existência de erro em regime. Já a malha com o ressonante, apresenta um ganho unitário, indicando erro zero em regime permanente, justificando, portanto, a sua aplicação.

Figura 24 – Diagrama de bode do projeto do controlador de corrente.



Fonte: Autor

Figura 25 – Detalhe do diagrama de bode do controlador de corrente.



Fonte: Autor

A simulação será realizada analisando o inversor desconectado da rede, A = 0, o momento de conexão entre o inversor e a rede, A = 1, e o inversor conectado a rede. Em uma segunda simulação, a referência de corrente será alterada sequencialmente de forma abrupta, com o intuito de avaliar o comportamento do controlador projetado. Validado o controlador via simulação computacional, o mesmo é validado experimentalmente.

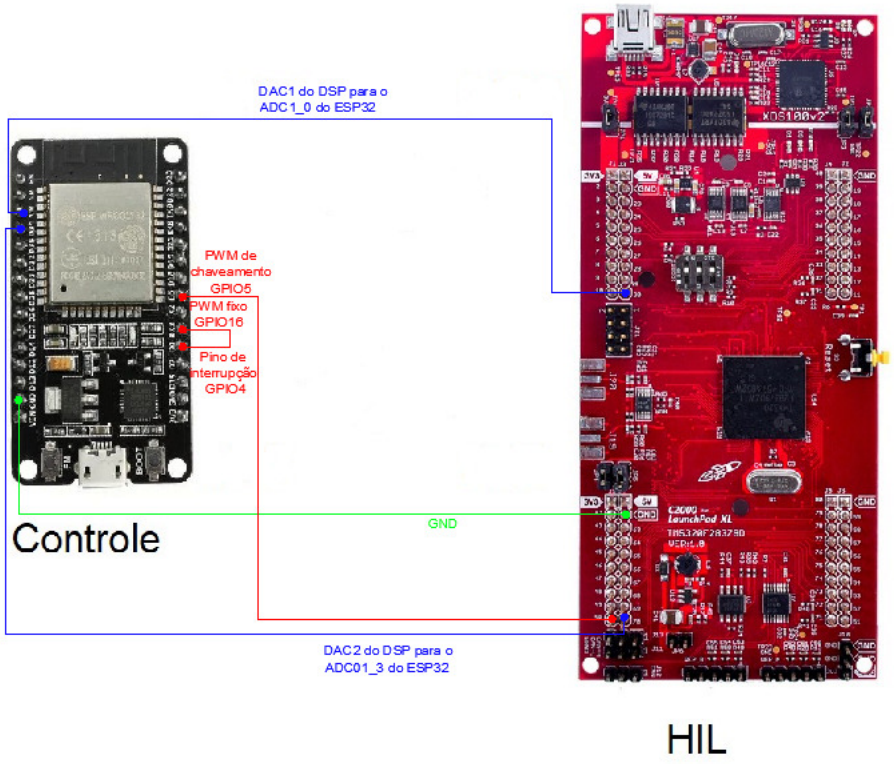
Na prática, a comutação de A é feito de forma manual utilizando o *expression* da plataforma *code composer*. As alterações de referência de corrente são feitas internamente no $ESP32$, contando aproximadamente três segundos cada alteração.

4 Resultados e Discussões

Neste capítulo serão apresentados os resultados simulados e experimentais conforme descritos no capítulo anterior. Serão mostrados os resultados simulados via *Psim* e logo em seguida a sua versão experimental HIL para comparação.

Figura 26 apresenta estrutura do HIL montado em bancada para a coleta dos dados experimentais.

Figura 26 – HIL implementado em bancada para coleta dos resultados experimentais.



Fonte: Autor

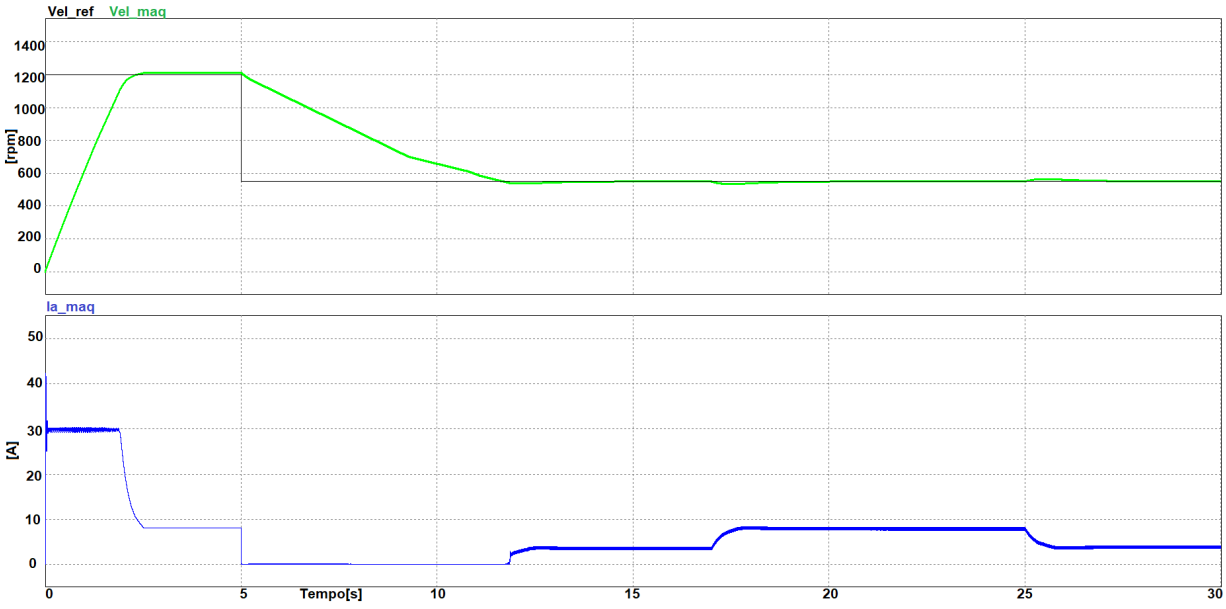
4.1 Controle da máquina CC

4.1.1 Resultados simulados via *Psim vs* resultados experimentais HIL

A Figura 27 apresenta o resultado simulado conforme Tabelas 2 e 3. Analisando, primeiramente, o gráfico de velocidade da Figura 27, percebe-se que o controlador atua conforme desejado, atuando para manter a velocidade de interesse, mesmo sob perturbações no eixo da máquina e mudanças bruscas de referência de velocidade, a velocidade adequa-se a nova referência bem lentamente, devido a lentidão da planta mecânica, conforme descrito pela expressão 3.7.

A corrente, por sua vez, dada a velocidade da planta elétrica, expressão 3.3, varia mais rapidamente. Na partida da máquina $E_a = 0$, expressões 2.1 e 2.3, a corrente tende a ser muito alta dada a baixa impedância do circuito de armadura, conforme descrito no processo metodológico uma limitação no valor de corrente de armadura durante o processo de controle é necessário para que os elevados transientes de corrente não danifiquem os enrolamentos. Durante a partida, o controlador mantém o valor máximo permitido de 30A até que a máquina atinja a velocidade de desejada, decrescendo após atingi-la para que a velocidade se mantenha. Após alteração da referência, o controle leva a corrente a zero, com o intuito de desacelerar a máquina até que a nova velocidade de interesse seja atingida, elevando-a novamente após o controle de velocidade atingir o valor de interesse, com o intuito de manter a nova velocidade de referência. A Figura 28 apresenta o resultado experimental da Figura 27.

Figura 27 – Resultados simulados do controle em cascata, referências das Tabelas 2 e 3.



Fonte: Autor

Tabela 2 – Referências do controlador de velocidade, simulação 1.

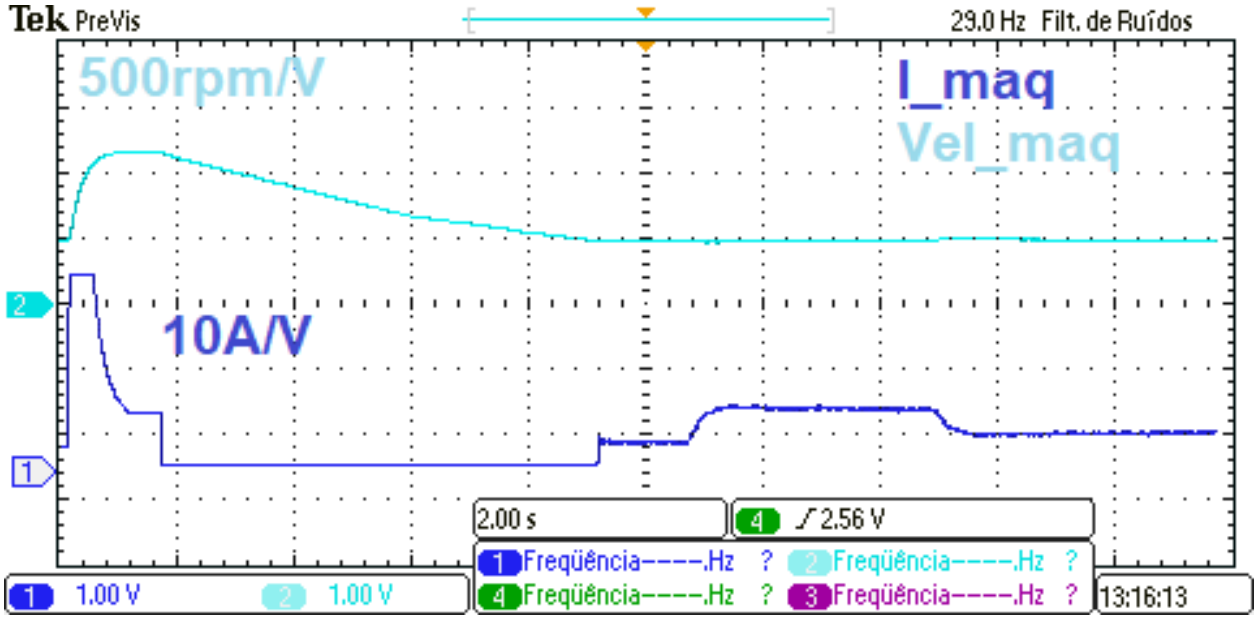
Valores de referência	
$n_r^*(rpm)$	Tempo (s)
1200	0 s
-650	5 s

Tabela 3 – Torque aplicado ao eixo, simulação 1.

Valores de referência	
$T_l(N.m)$	Tempo (s)
1	0
5	17
2	25

Fonte: Autor

Figura 28 – Resultado experimental, Tabelas 2 e 3.



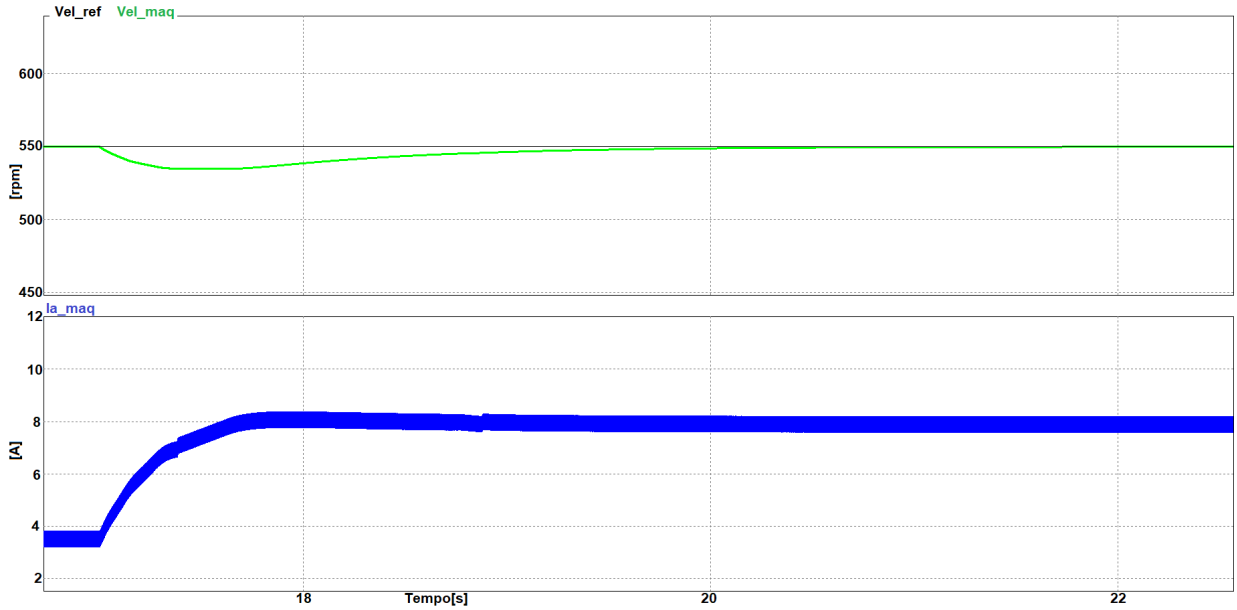
Ao se analisar as Figuras 27 e 28 percebe-se a semelhança entre as duas, exceto a dinâmica na partida da máquina, sendo bem mais rápida no resultado experimental. Conforme descrito no procedimento metodológico, a referência de 1200 rpm é obtida de forma manual ao pressionar o *enable* do $ESP32$, porém ao iniciar a simulação o controlador já começa a atuar para a referência embutida de 550 rpm. Tal dinâmica pode ser observada nos sinais de corrente e velocidade com aproximadamente 4A de corrente e a velocidade de referência.

Analisando a dinâmica como um todo, enquanto a velocidade não atinge a referência inicial, botão *enable* pressionado, a corrente permanece constante em 30A, conforme resultado simulado via $Psim$. Ao atingir a velocidade nominal, a corrente é ajustada para manter a velocidade de referência, aproximadamente 9A, novamente coerente com os resultados computacionais, o valor lido de velocidade durante esta etapa é de aproximadamente 1175 rpm. Em seguida, o controlador ajusta a corrente de modo a desacelerar

a máquina até que a nova referência de velocidade seja alcançada. Ao atingir a nova velocidade de interesse, o controlador novamente ajusta a corrente para manter a velocidade, o valor lido é de aproximadamente 4A para a corrente e 550 rpm para a velocidade. O HIL se apresenta estável durante todo o processo de simulação.

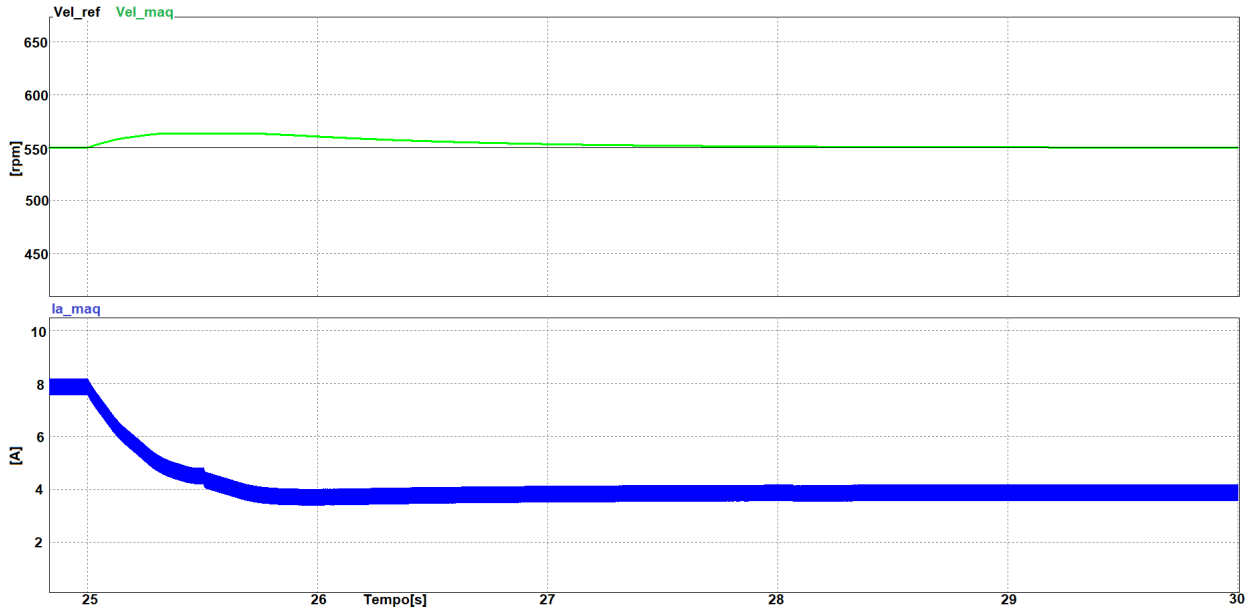
As perturbações no eixo da máquina são melhor vistas nos detalhes das Figuras 29 e 30. Após um acréscimo de carga no eixo da máquina a corrente aumenta, conforme descrito pelas expressões 2.5 e 2.8, a velocidade naturalmente diminui, porém, através das ações do controlador retorna para velocidade de referência em instantes. O inverso ocorre com a retirada de carga do eixo, a corrente diminui e a velocidade aumenta momentaneamente, retornando à velocidade de interesse via ações de controle.

Figura 29 – Detalhe do acréscimo de carga no eixo do motor.



Fonte: Autor

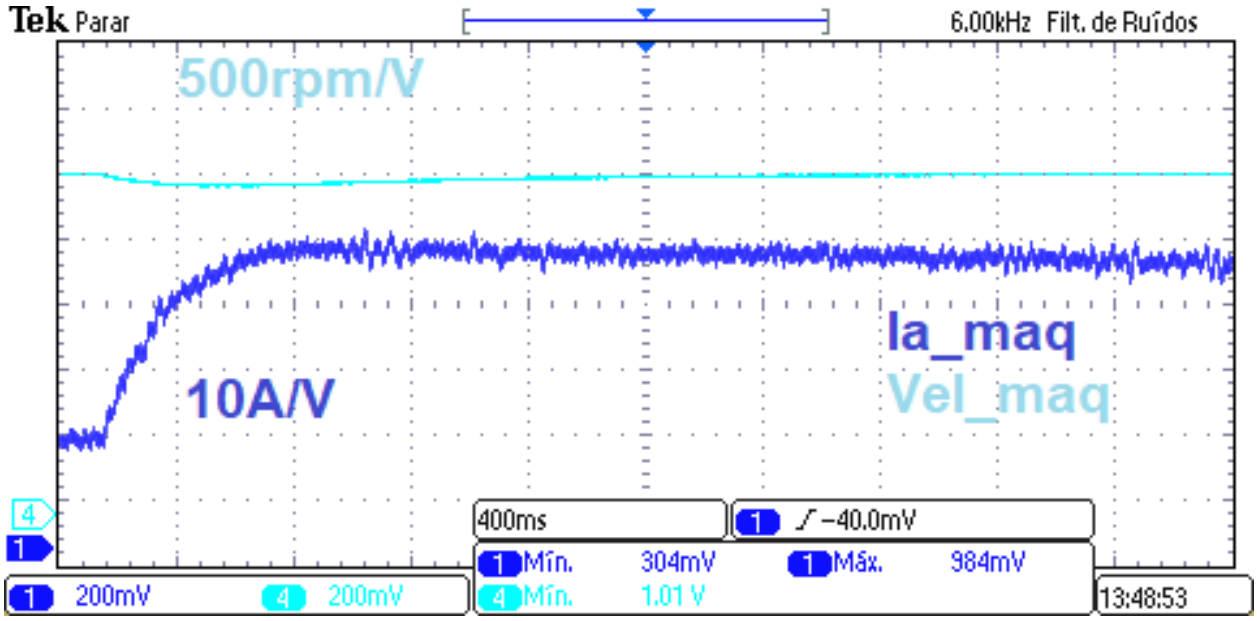
Figura 30 – Detalhe da redução de carga no eixo do motor.



Fonte: Autor

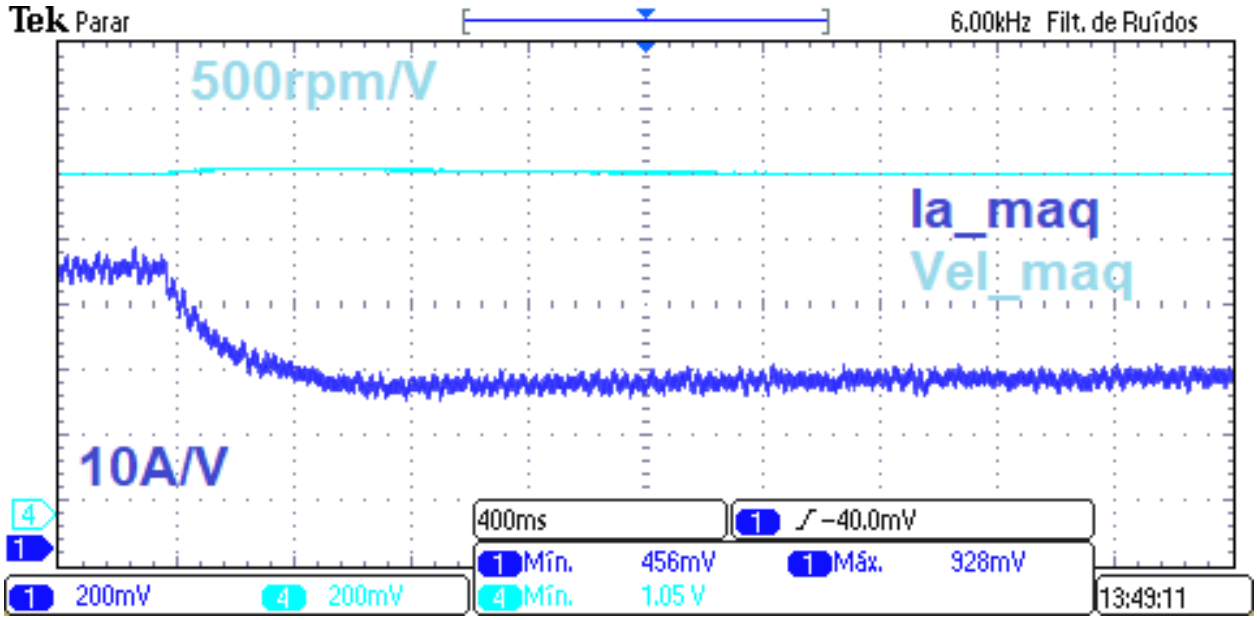
Os resultados experimentais são vistos através das Figuras 31 e 32, uma ligeira queda na velocidade ocorre ao adicionar carga ao motor através do *expression* do *code composer*, o controle ajusta a corrente de modo a incrementar T_e o suficiente para suprir a nova necessidade de torque, aproximadamente 8,4A. A velocidade retorna em instantes à velocidade de referência após a atuação do controlador. Conforme esperado o oposto ocorre ao se reduzir a carga no eixo do motor, a velocidade aumenta ligeiramente, estabilizando novamente em 550 rpm, já a corrente se estabiliza em aproximadamente 5A.

Figura 31 – Detalhe experimental, acréscimo de carga no eixo do motor.



Fonte: Autor

Figura 32 – Detalhe experimental, redução de carga no eixo do motor.



Fonte: Autor

A Figura 33 possui apenas uma referência de velocidade, conforme Tabela 4, após o controlador atingir a velocidade desejada, o sistema apresenta a mesma dinâmica descrita pelas Figuras 27, 29 e 30. As Tabelas 4 e 5 apresentam os instantes em que a referência e as perturbações são aplicadas na simulação.

Figura 33 – Resultados simulados do controle em cascata, referências das Tabelas 4 e 5.

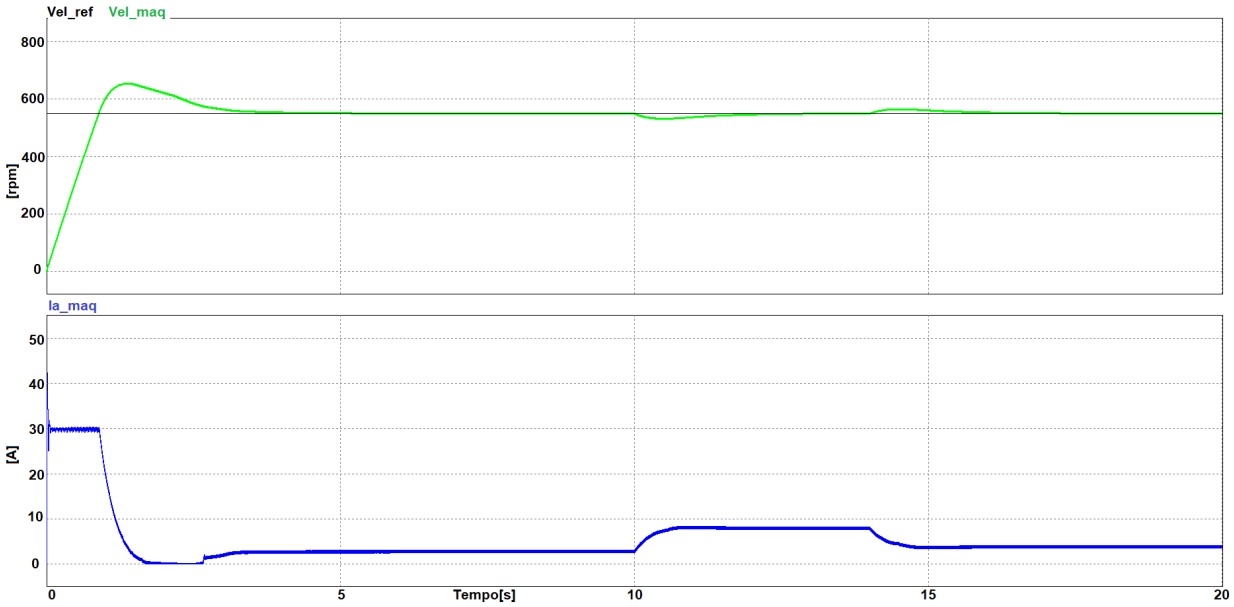


Tabela 4 – Referências do controlador de velocidade, simulação 2.

Valores de referência	
$n_r^*(rpm)$	Tempo (s)
550	0 s

Tabela 5 – Torque aplicado ao eixo, simulação 2.

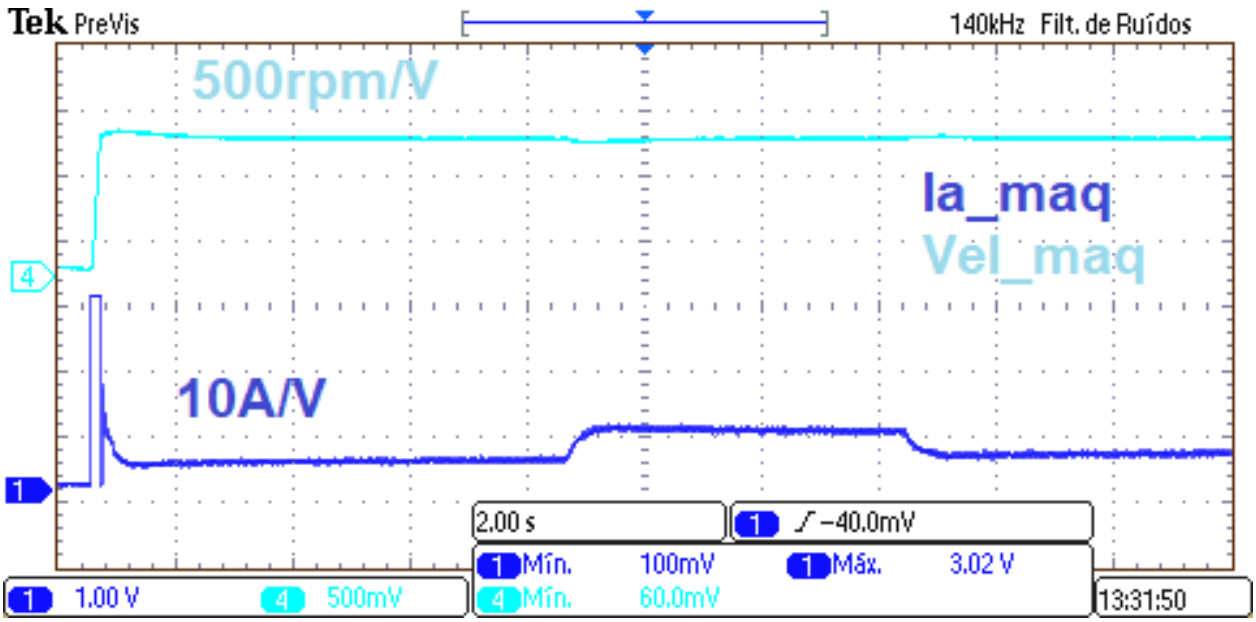
Valores de referência	
$T_l(N.m)$	Tempo (s)
1	0
5	10
2	14

Fonte: Autor

A Figura 34, representa a correspondência experimental da Figura 33, possuindo apenas uma referência de velocidade, a máquina novamente, por característica já explicadas referente as portas DAC, já inicia com uma pequena dinâmica, partindo abruptamente ao se iniciar a simulação, atingindo o limite de 30A, estabilizando em seguida em aproxi-

madamente em 4A, e a velocidade em aproximadamente 550 rpm. Seguindo exatamente a mesma dinâmica das Figuras 31 e 32, durante as perturbações no eixo do motor.

Figura 34 – Resultado experimental, Tabelas 4 e 5.



Fonte: Autor

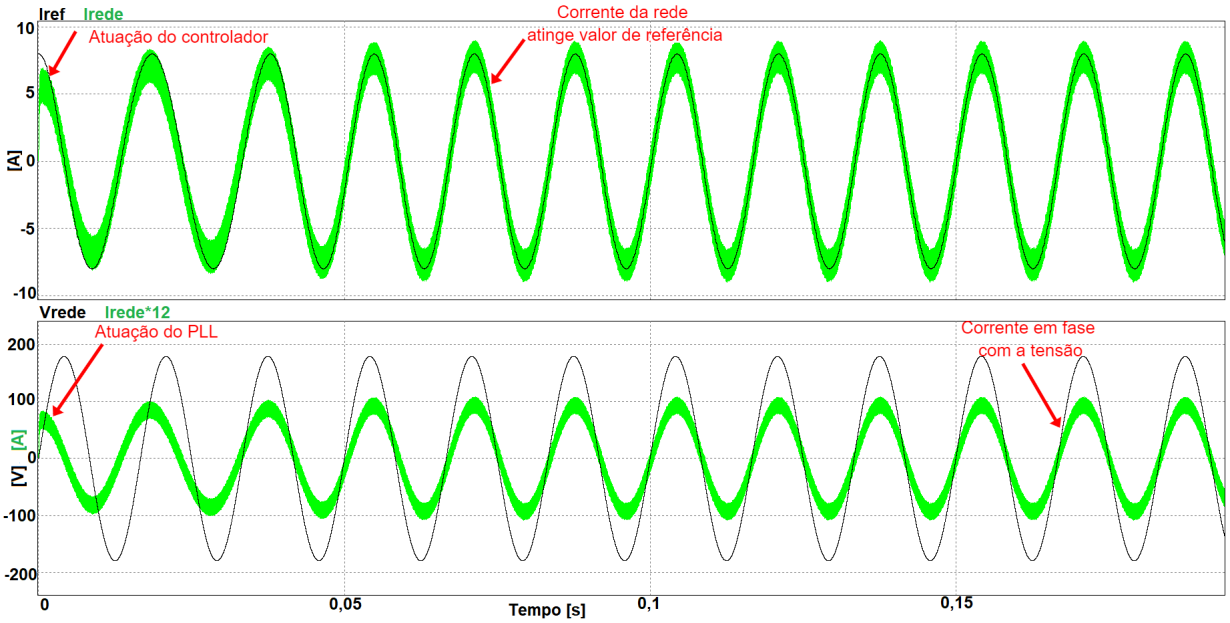
Os resultados experimentais HIL se apresentam convergentes com as simulações computacionais. O controlador em cascata compensa de forma rápida as perturbações e variações bruscas dos valores de referência, mantendo as variáveis de interesse em valores desejáveis conforme projeto. O sistema HIL se apresenta estável em todos os resultados, não tendendo à instabilidade em instante algum.

4.2 Controle de corrente do inversor monofásico

4.2.1 Resultados simulados via $Psim$ vs resultados experimentais HIL

A Figura 35 representa a simulação do inversor conectado a uma carga de 10Ω, por ela é possível destacar o funcionamento do PLL, que em poucos instantes atua deixando a corrente e tensão em fase, bem como o PI de corrente, encontrando a referência igualmente em milésimos de segundo. A corrente foi multiplicada por 12 nos resultados simulados via $Psim$ para facilitar a visualização do sincronismo.

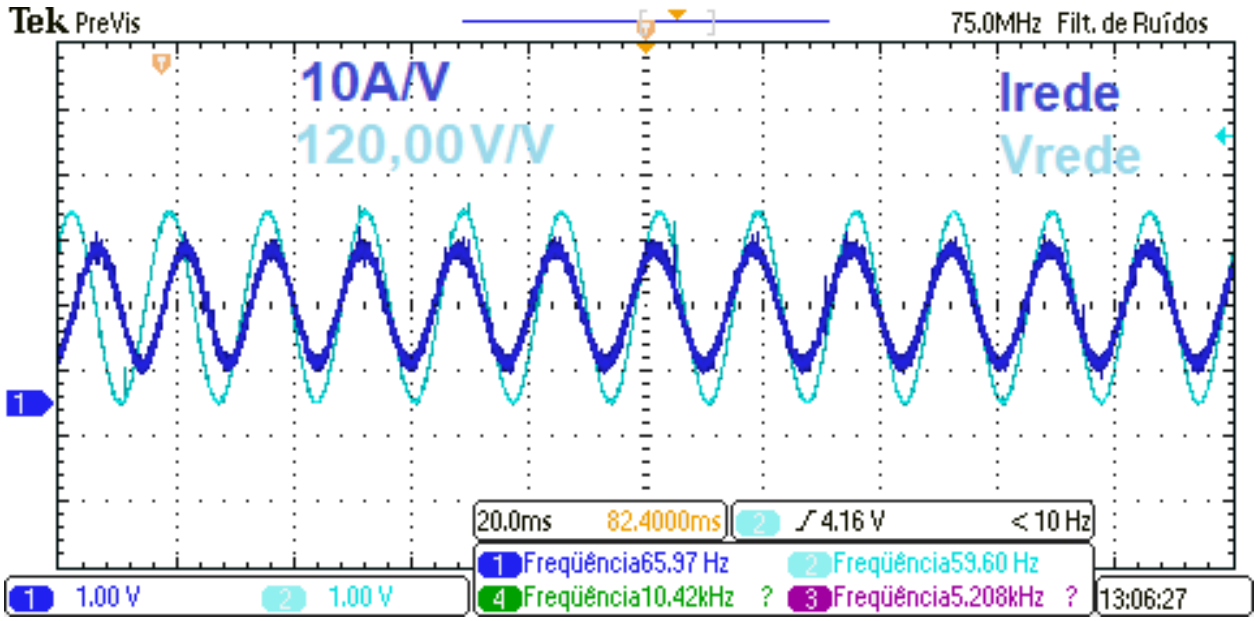
Figura 35 – Inversor conectado a carga de 10Ω, A = 0.



Fonte: Autor

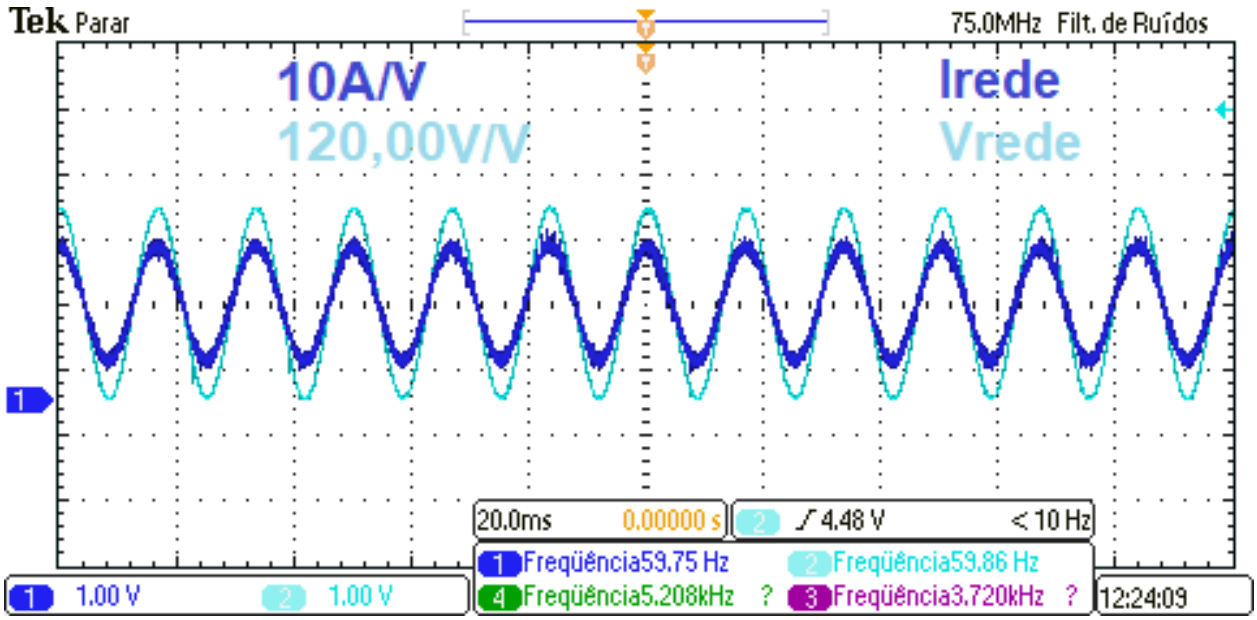
As Figuras 36 e 37 apresentam os resultados da simulação HIL, equivalente à Figura 35. Observa-se a atuação do PLL no início da operação, adequando a fase da corrente até que a mesma esteja em sincronismo com o sinal de tensão de referência. A referência de corrente é fixa em 8A de pico, os valores medidos de aproximadamente 8,5A e 179,6V de pico, valores coerentes com o resultado simulado. A Figura 38 ilustra os sinais de corrente e tensão da Figura 37 separadamente para melhor visualização dos sinais de forma individualizada.

Figura 36 – Resultado experimental, inversor desconectado. A Figura mostra o PLL entrando em ação e sincronizando a fase da corrente com a tensão.



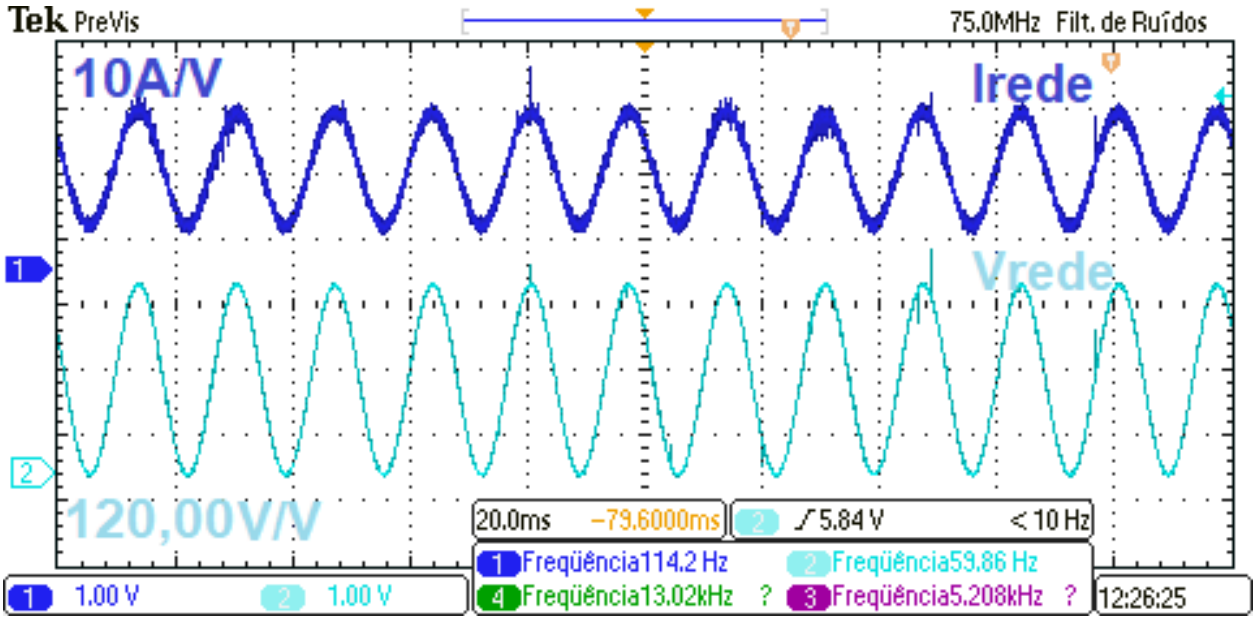
Fonte: Autor

Figura 37 – Corrente e tensão sincronizadas, após o PLL atingir o regime permanente.



Fonte: Autor

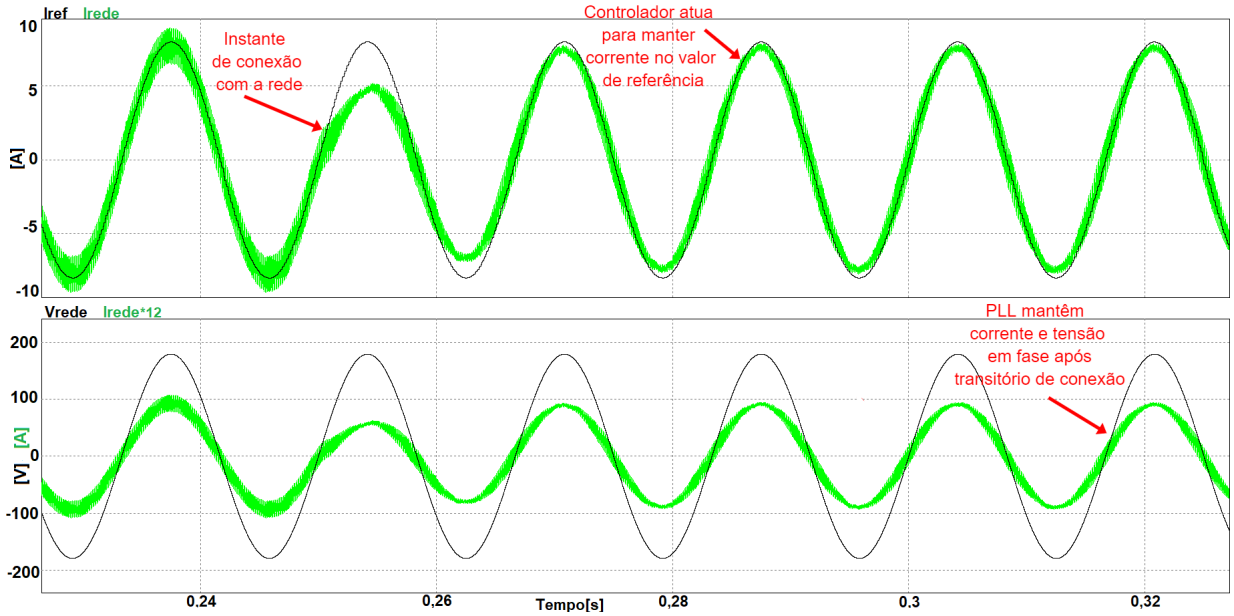
Figura 38 – Resultado experimental, detalhe dos sinais de tensão e corrente da Figura 37.



Fonte: Autor

A Figura 39 ilustra o instante em que a conexão com a rede é efetuada, o PLL não aparenta sofrer nenhuma perturbação, apenas a corrente no instante de conexão sofre um transitório, que se estabiliza rapidamente devido a atuação do controlador PI. A forma de onda da corrente muda ligeiramente após conexão com a rede, isso é explicado pelo diagrama de polos e zeros do sistema, em que, quando conectado a uma carga de 10Ω, introduz um amortecimento no sistema, os polos se localizam em uma região de maior estabilidade, quando conectado à rede, o valor da resistência diminui em cerca de 10 vezes, este polo está localizado próximo a 0, que é a região próxima a instabilidade do sistema (DORF; BISHOP, 2001).

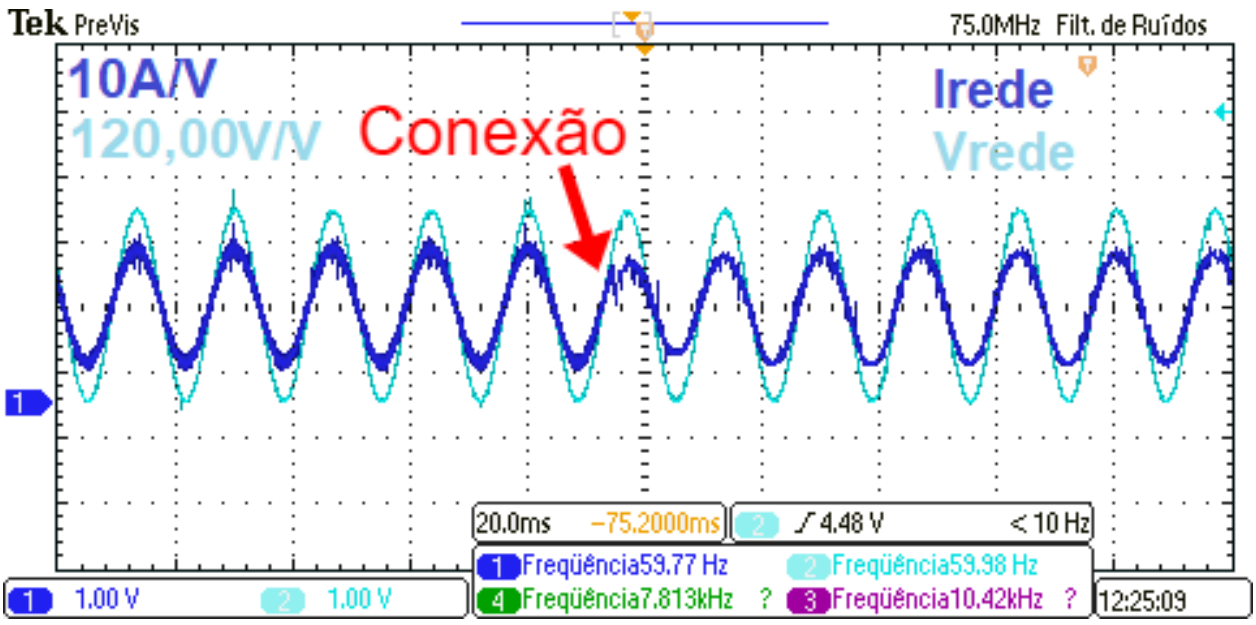
Figura 39 – Instante de conexão do inversor com a rede, transição A = 0 para A = 1.



Fonte: Autor

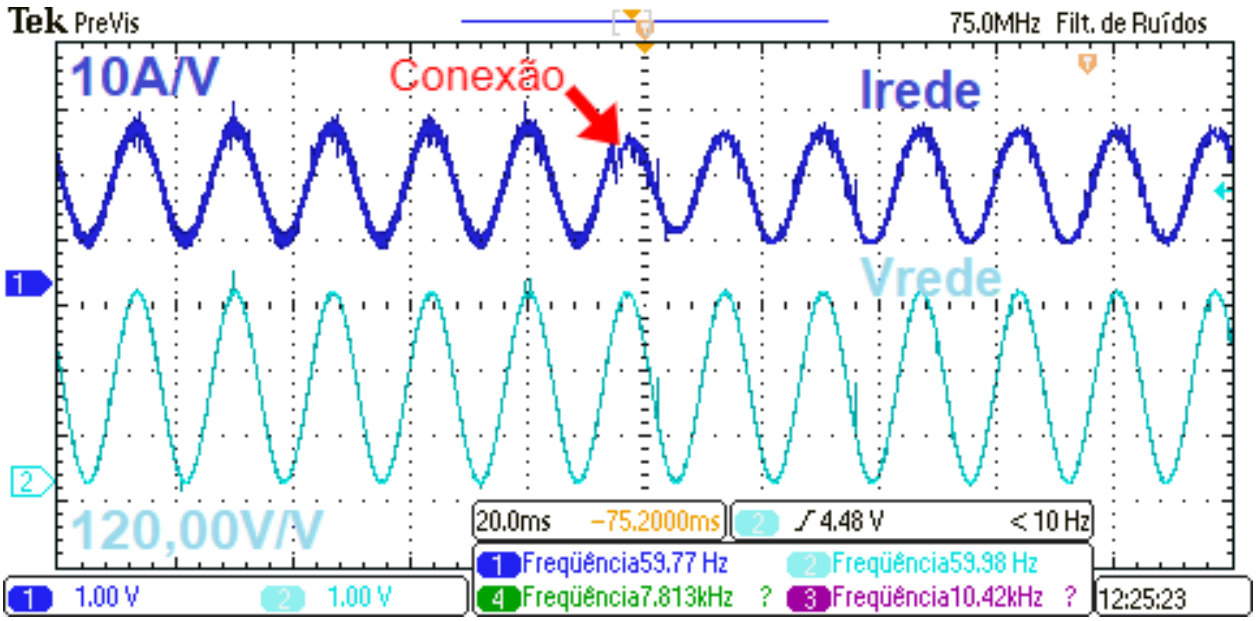
As Figuras 40 e 41 ilustram o momento de conexão com a rede na simulação HIL, após um pequeno transitório o controlador de corrente atua rapidamente para manter a variável de interesse no valor desejado de 8A de pico. O sistema já se encontrava em sincronismo no momento de conexão, o PLL não apresenta problemas para manter corrente e tensão em fase e o sistema permanece estável durante e após a conexão com a rede.

Figura 40 – Resultado experimental, instante de conexão do inversor com a rede, transição A = 0 para A = 1.



Fonte: Autor

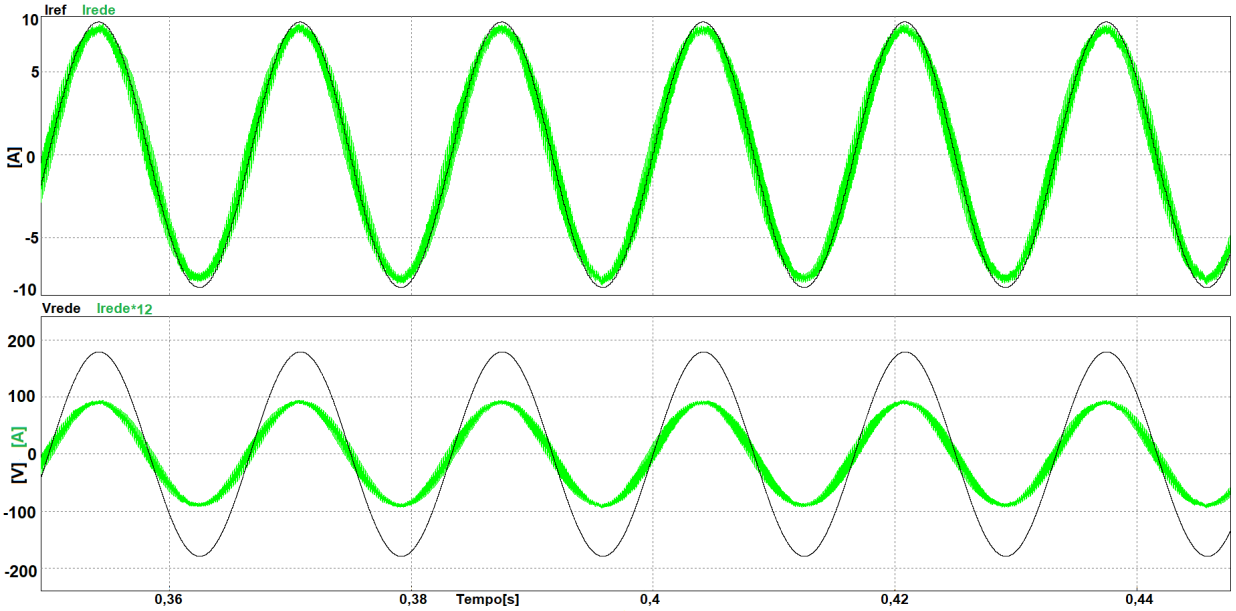
Figura 41 – Resultado experimental, detalhe individual dos sinais de tensão e corrente no instante de conexão do inversor com a rede, transição A = 0 para A = 1.



Fonte: Autor

A Figura 42 ilustra o comportamento do inversor em regime permanente após conexão com o sistema elétrico, a corrente e tensão permanecem em fase e o controlador mantém a corrente no valor de interesse.

Figura 42 – Inversor conectado à rede em regime permanente, A = 1.

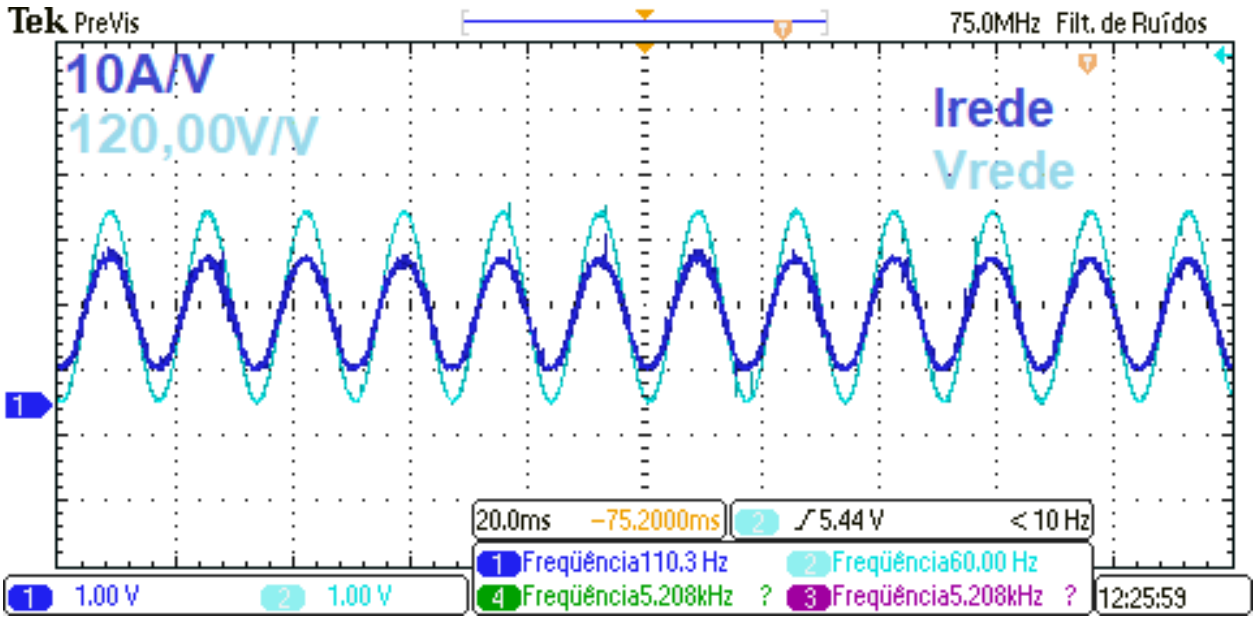


Fonte: Autor

O resultado experimental do inversor conectado à rede em regime permanente é ilustrado nas Figuras 41 e 43. O sistema se apresenta estável após a conexão com a rede

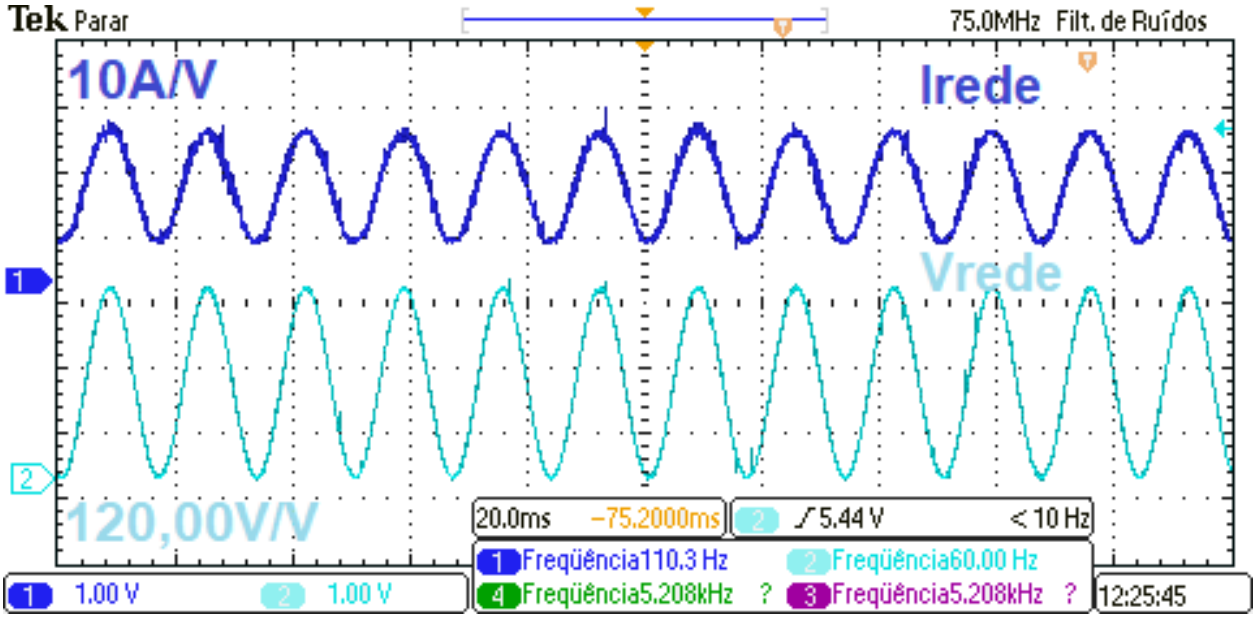
mantendo o resultado de referência desejado, com uma pequena alteração na forma de onda, provocada pelos mesmos motivos explicados previamente.

Figura 43 – Resultado experimental, inversor conectado à rede em regime permanente, A = 1.



Fonte: Autor

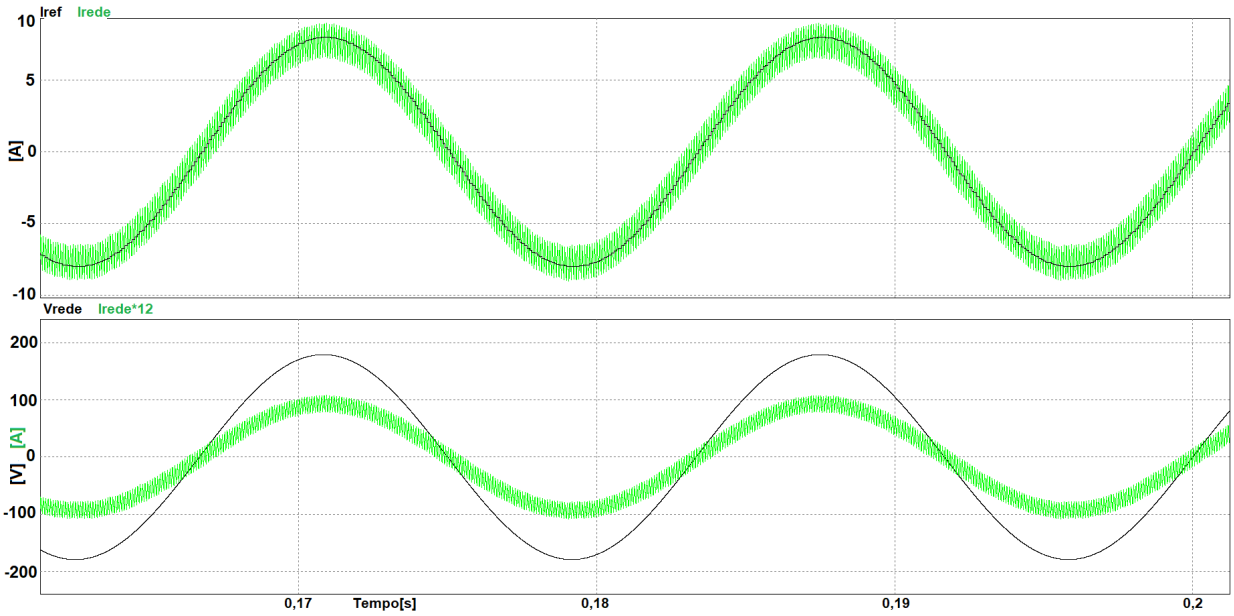
Figura 44 – Resultado experimental, detalhe dos sinais de tensão e corrente da Figura 41.



Fonte: Autor

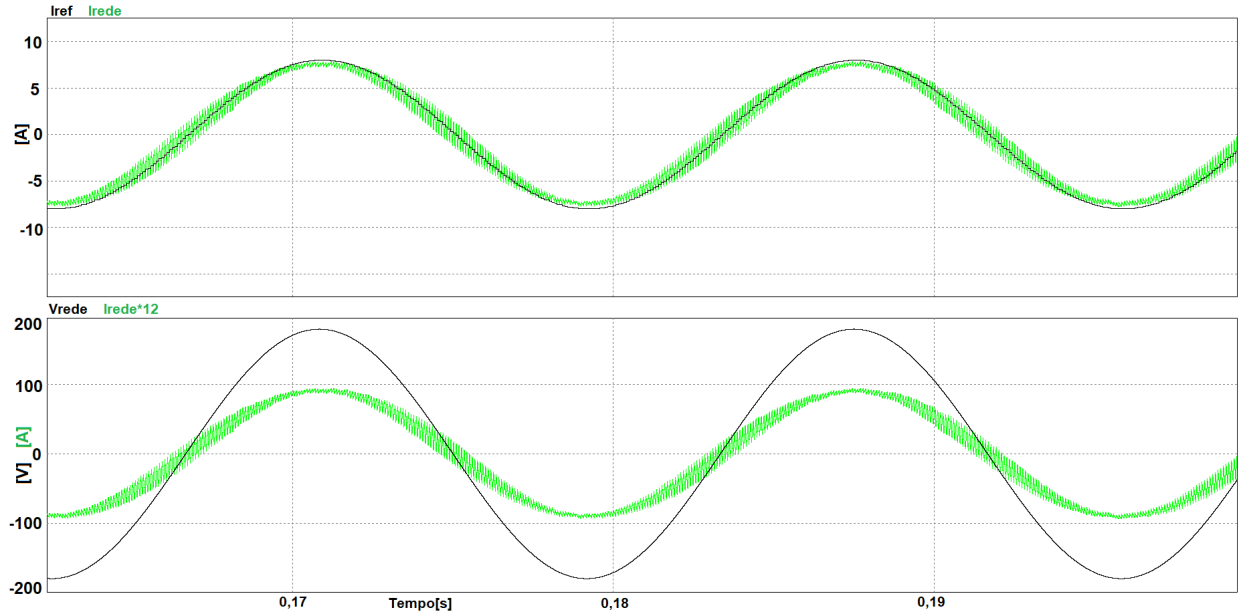
As Figuras 45 e 46, são detalhes aproximados das formas de onda de corrente e tensão para o inversor desconectado e conectado à rede, respectivamente.

Figura 45 – Detalhe dos sinais para o inversor desconectado, A = 0.



Fonte: Autor

Figura 46 – Detalhe dos sinais para o inversor conectado, A = 1.

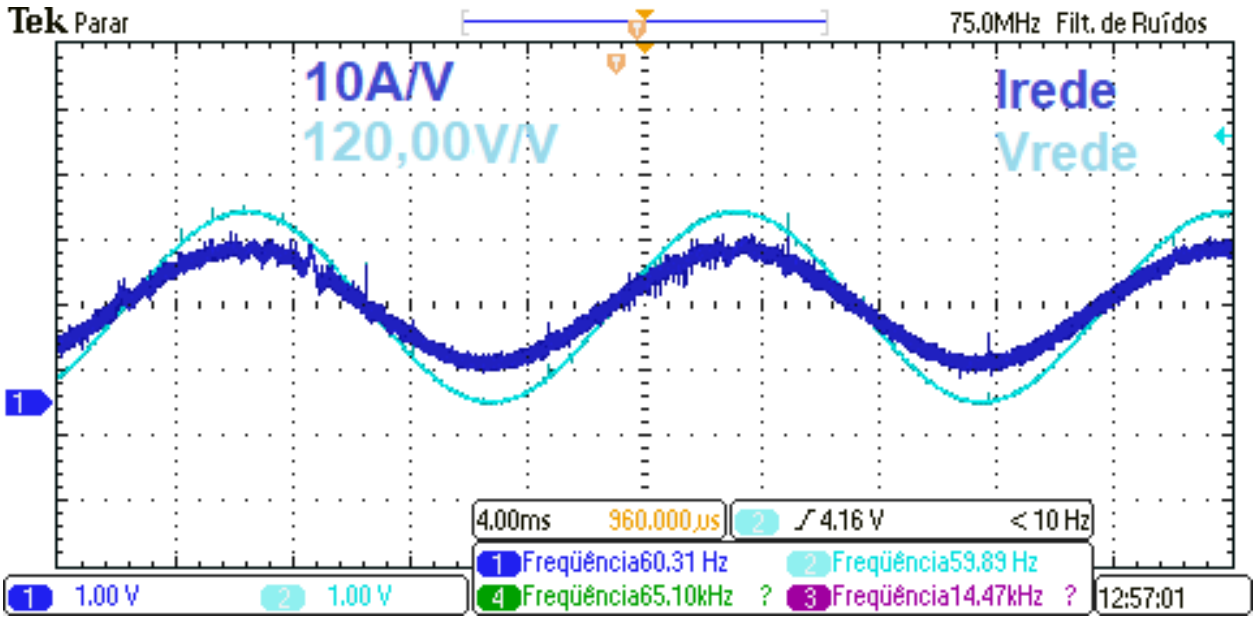


Fonte: Autor

As Figuras 47 e 48 são os detalhes aproximados dos sinais de tensão e corrente para o inversor desconectado e conectado, respectivamente, equivalentes aos resultados simulados das Figuras 45 e 46. Ficam mais nítidos, através delas, a análise dos valores medidos de aproximadamente 8,5A para o sinal de corrente e 179,6V para o sinal de

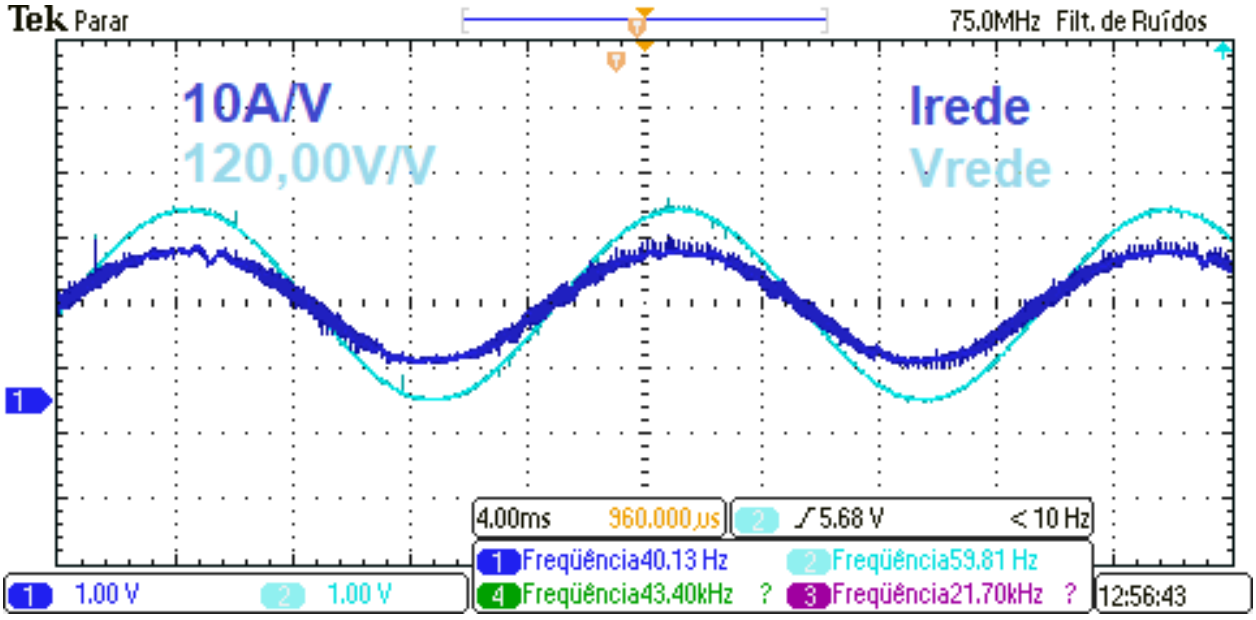
tensão. É possível perceber também, com mais detalhes, a alteração do formato da onda de corrente após conexão com a rede.

Figura 47 – Resultado experimental, detalhe dos sinais para o inversor desconectado, A = 0.



Fonte: Autor

Figura 48 – Resultado experimental, detalhe dos sinais para o inversor conectado, A = 1.

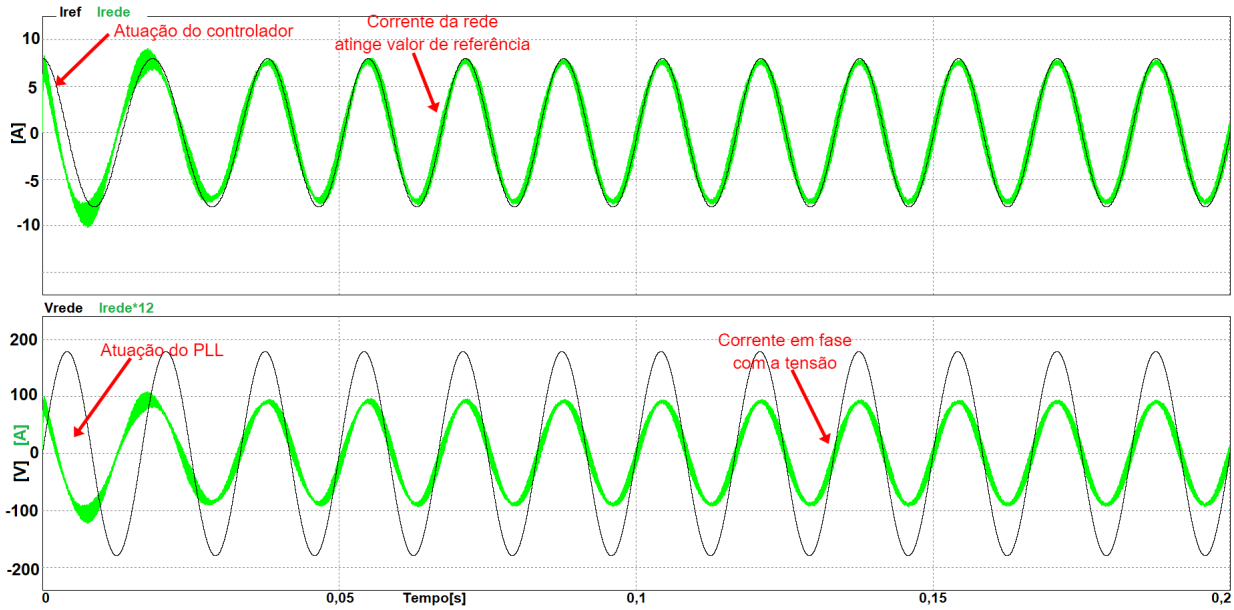


Fonte: Autor

A Figura 49 apresenta a simulação do inversor operando conectado à rede, nota-se o transitório da corrente no instante de conexão, rapidamente compensado pelo controlador

PI, bem como a atuação do PLL, sincronizando a corrente em questão de milésimos de segundos.

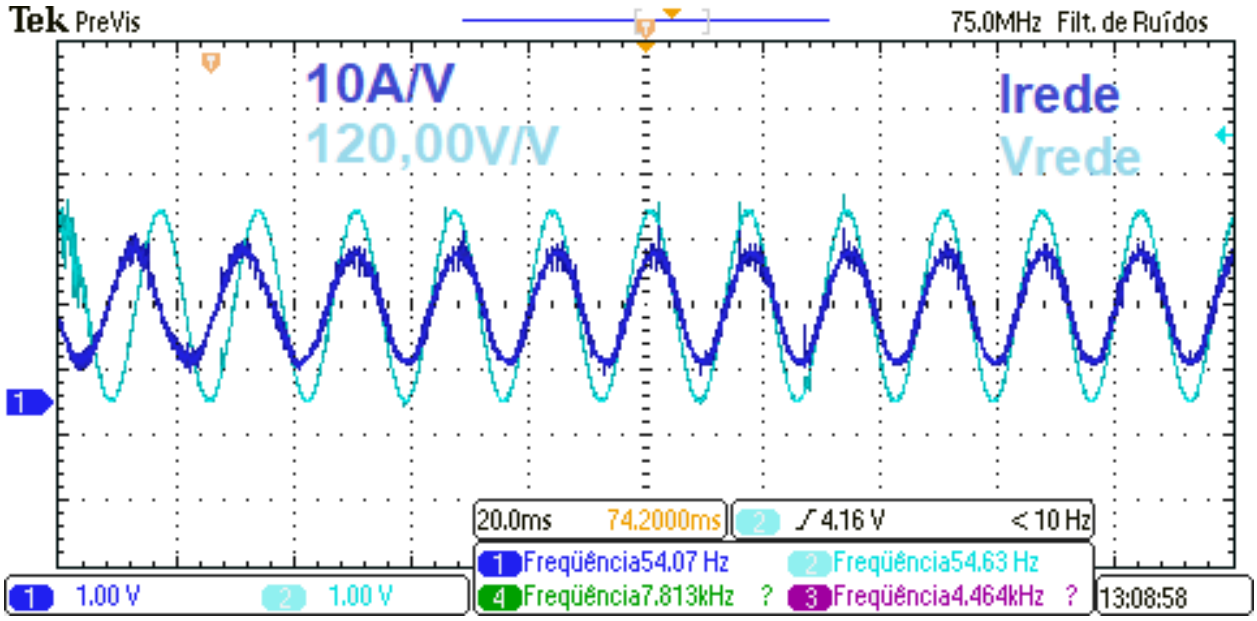
Figura 49 – Inversor iniciando conectado à rede, A = 1.



Fonte: Autor

A Figura 50 apresenta o resultado da simulação HIL com o inversor conectado, fora de fase inicialmente e com um referencial fixo de corrente de 8A de pico. O controlador de corrente atua no início, para manter a corrente na referência desejada, o PLL em poucos instantes sincroniza os sinais de corrente e tensão. O sistema permanece estável em todos os instantes. O controlador permanece estável durante toda a simulação.

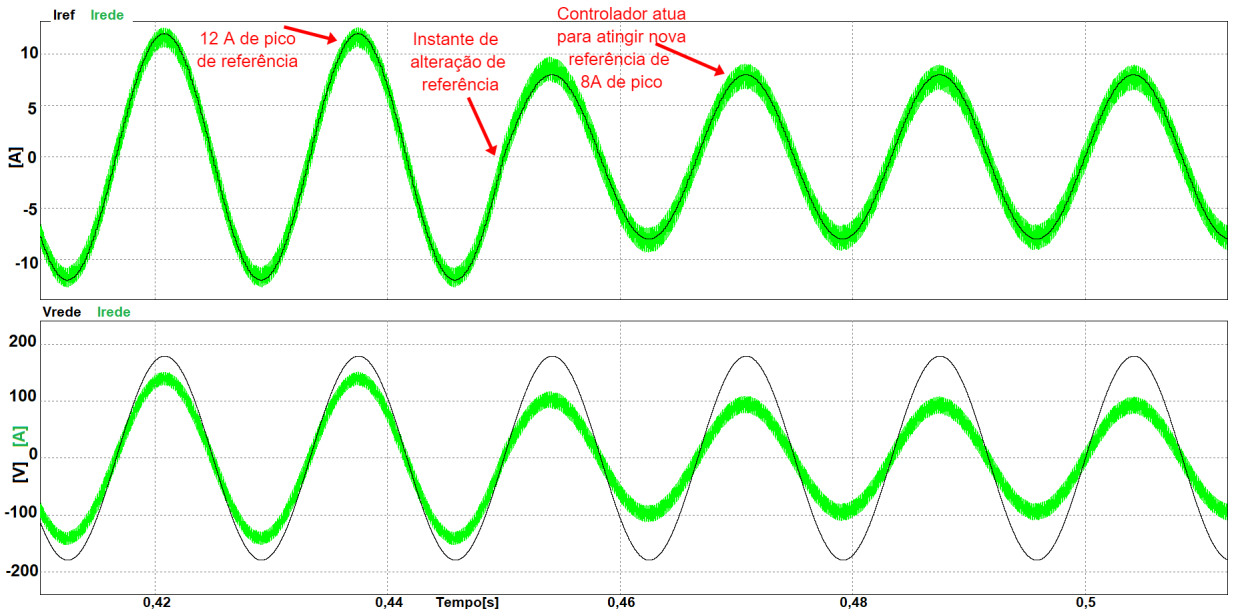
Figura 50 – Resultado experimental, inversor iniciando conectado à rede, A = 1.



Fonte: Autor

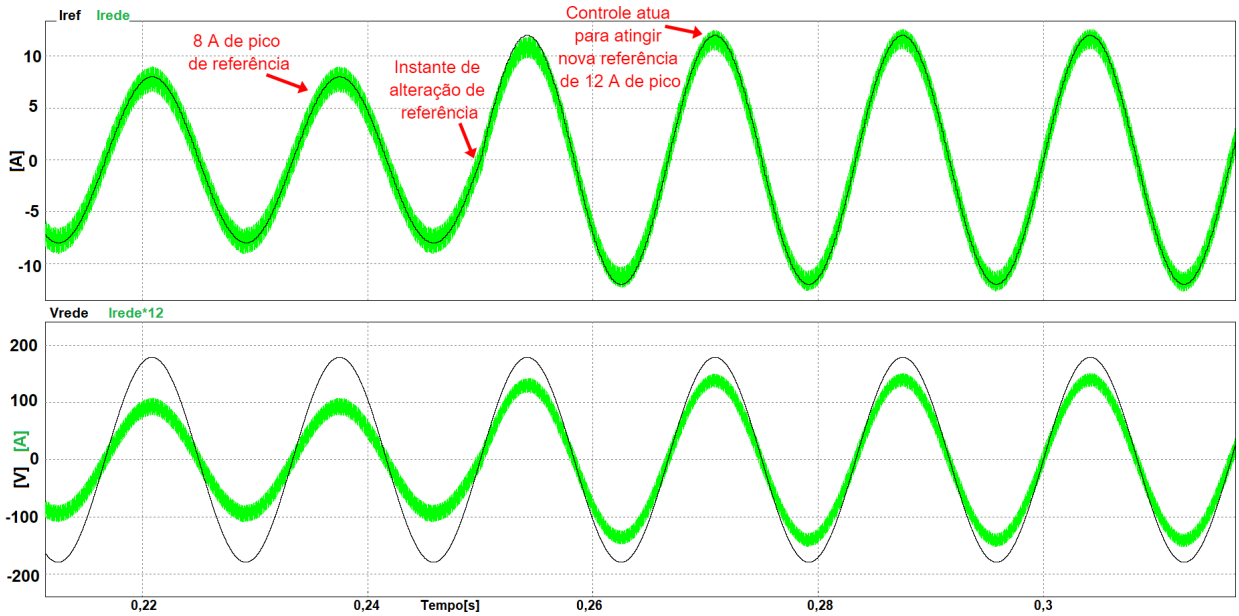
As Figuras 51 e 52, apresentam os resultados para a atuação do controlador de corrente para o inversor desconectado da rede, quando submetido a variações de referências em degrau, em ambos os casos, o controlador se comporta muito bem, compensando a variação rapidamente e mantendo o sistema estável.

Figura 51 – Inversor desconectado, A = 0, alteração de referência de 12A para 8A.



Fonte: Autor

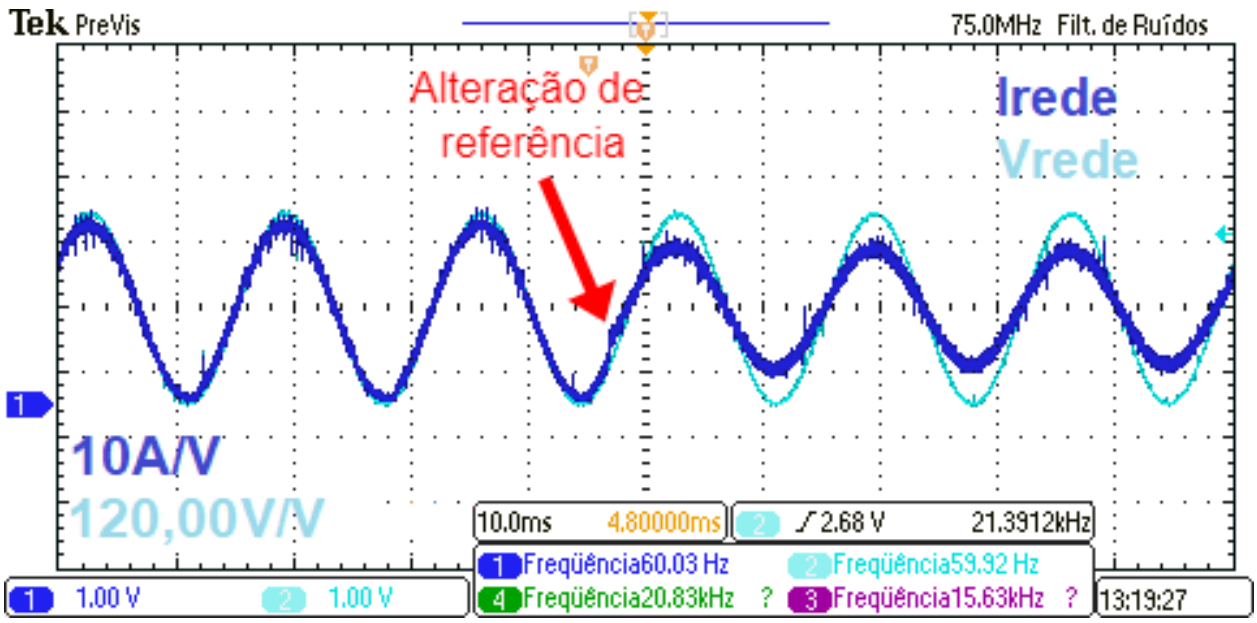
Figura 52 – Inversor desconectado, A = 0, alteração de referência de 8A para 12A.



Fonte: Autor

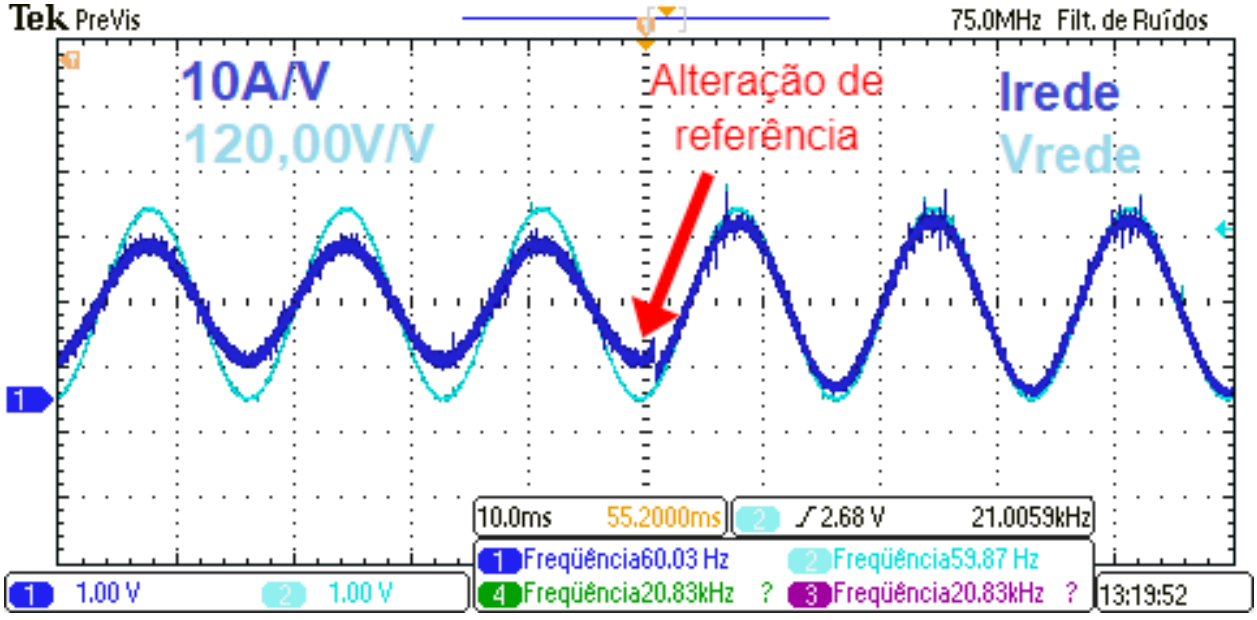
As Figuras 54 e 57 ilustram os resultados experimentais referentes às alterações em degrau da referencia de corrente para o inversor desconectado, equivalente às simulações presentes nas Figuras 51 e 52. Os resultados práticos em ambas alterações se apresentam estáveis e com atuações rápidas do controlador PI adequando a corrente injetada conforme desejado, aproximadamente 8,5A de referência inferior, e 12,5A de referência superior. Novamente os resultados se apresentam estáveis e conforme projetado.

Figura 53 – Resultado experimental, inversor desconectado, A = 0, alteração de referência de 12A para 8A (pico).



Fonte: Autor

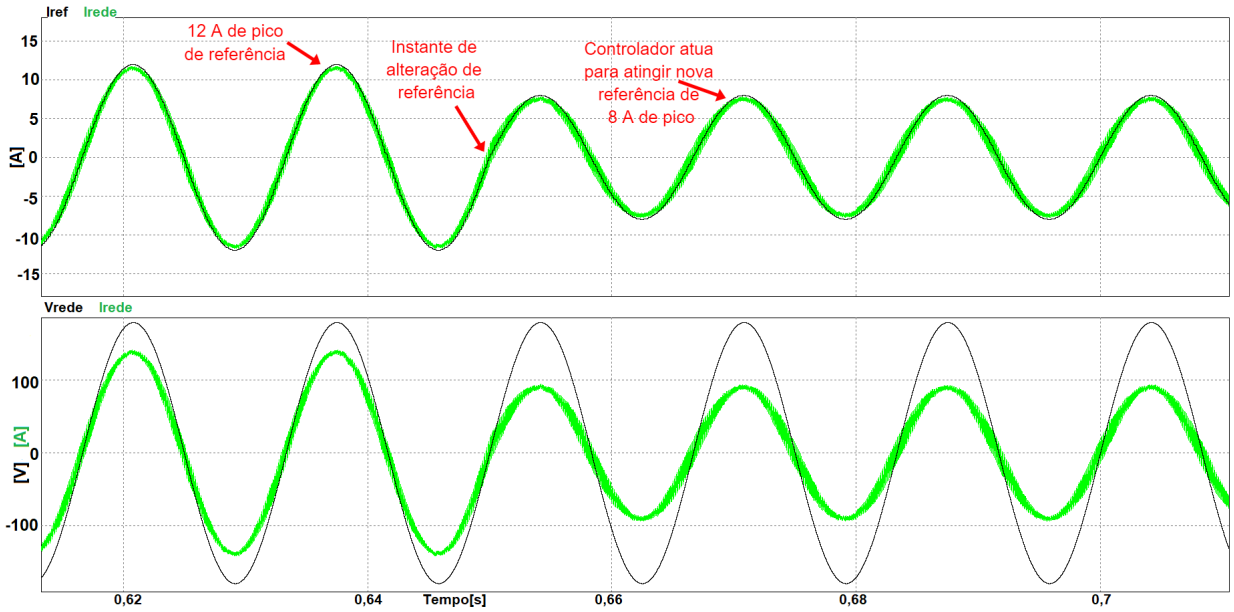
Figura 54 – Resultado experimental, inversor desconectado, A = 0, alteração de referência de 8A para 12A (pico).



Fonte: Autor

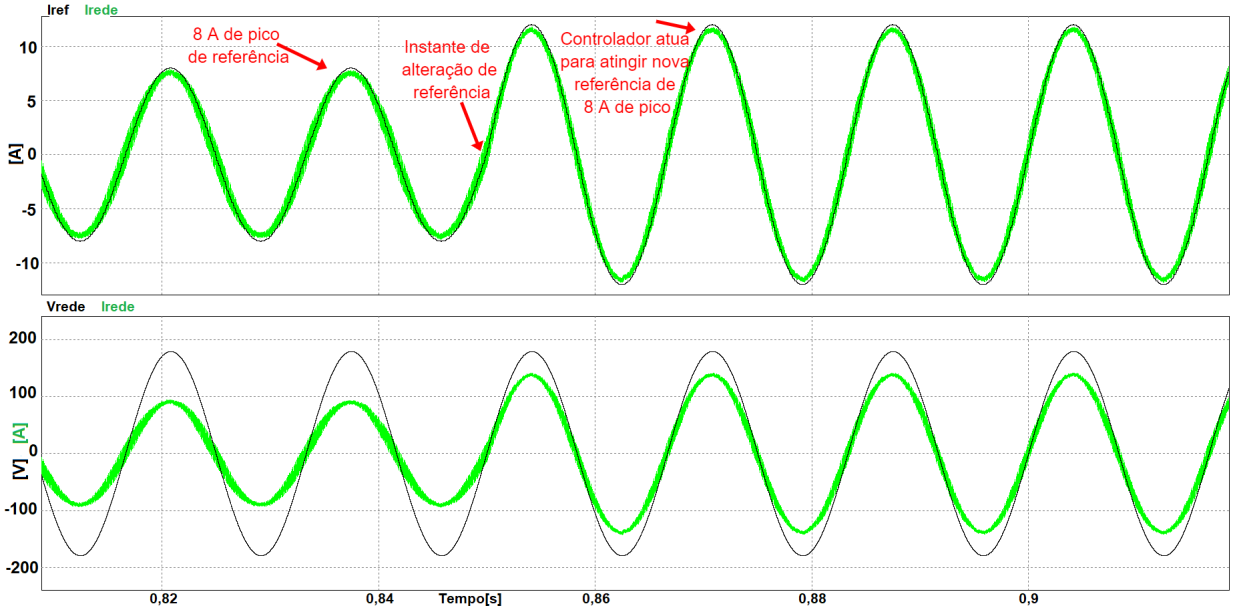
As Figuras 55 e 56, ilustram o mesmo teste realizado nas Figuras 51 e 52, porém, agora, o inversor está conectado à rede. Novamente, os controladores atuam bem nas alterações de corrente de referência, mantendo o sistema estável.

Figura 55 – Inversor conectado, A = 1, alteração de referência de 12A para 8A.



Fonte: Autor

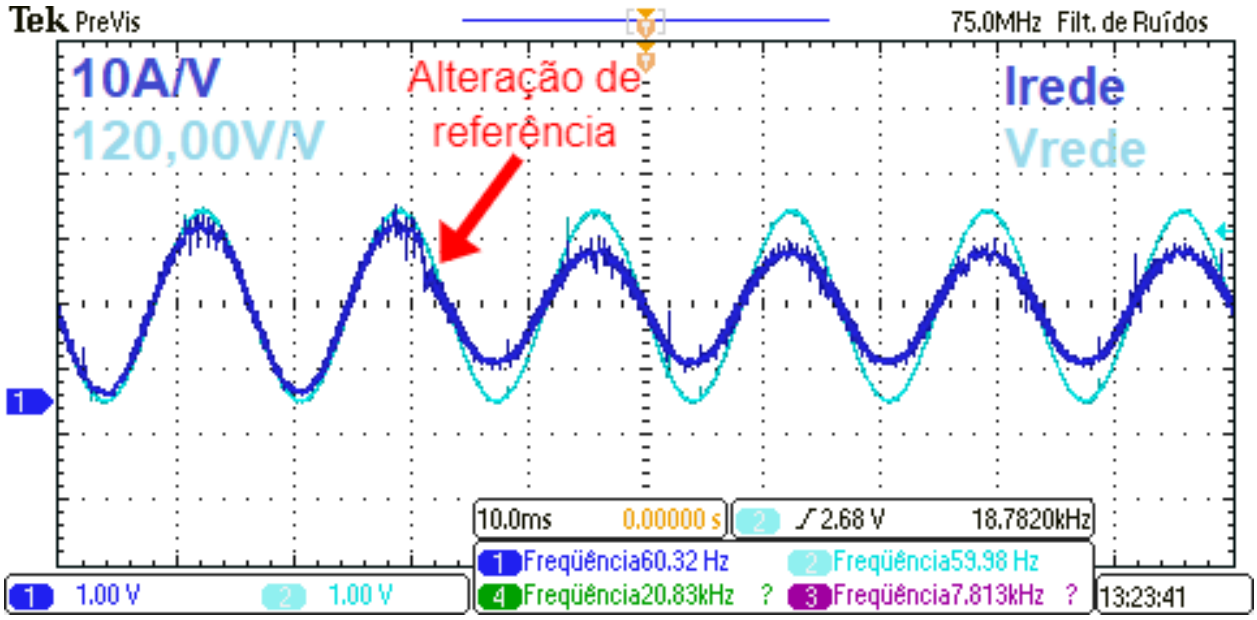
Figura 56 – Inversor conectado, A = 1, alteração de referência de 8A para 12A.



Fonte: Autor

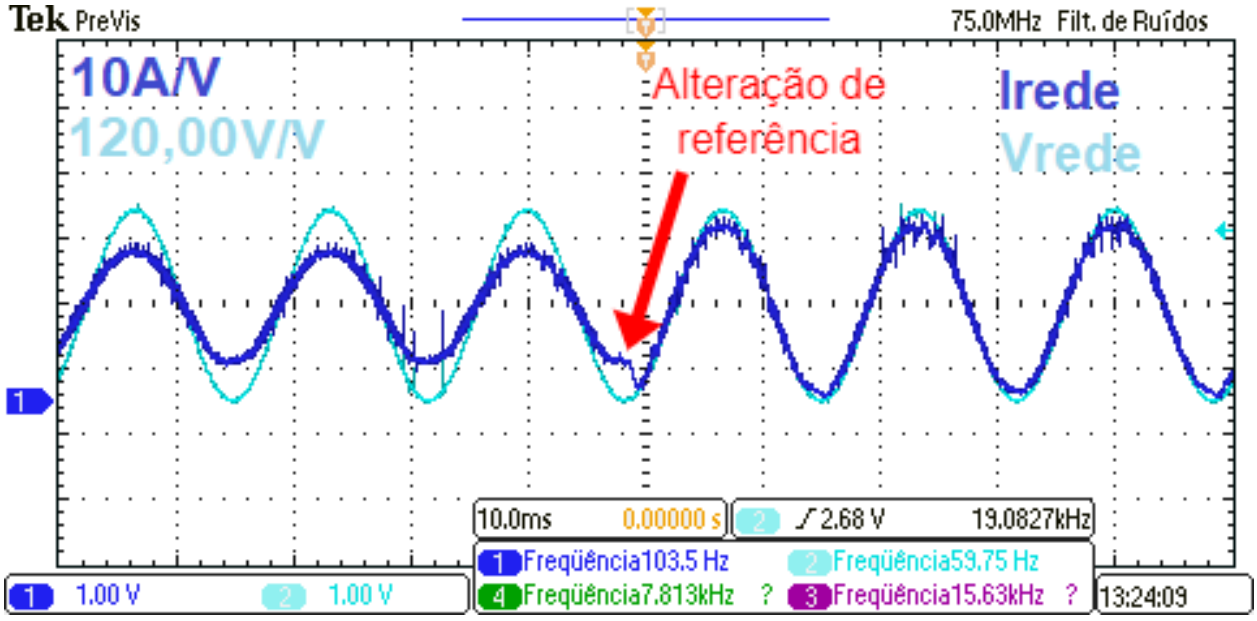
As Figuras 57 e 58, são as versões experimentais das simulações 55 e 56, para o inversor conectado à rede, as alterações em degrau da referência apresentam a mesma dinâmica vista na versão experimental desconectada, com o controlador atuando rapidamente nas alterações de referência e mantendo o sistema estável.

Figura 57 – Resultado experimental, inversor desconectado, A = 1, alteração de referência de 12A para 8A (pico).



Fonte: Autor

Figura 58 – Resultado experimental, inversor desconectado, A = 0, alteração de referência de 8A para 12A (pico).

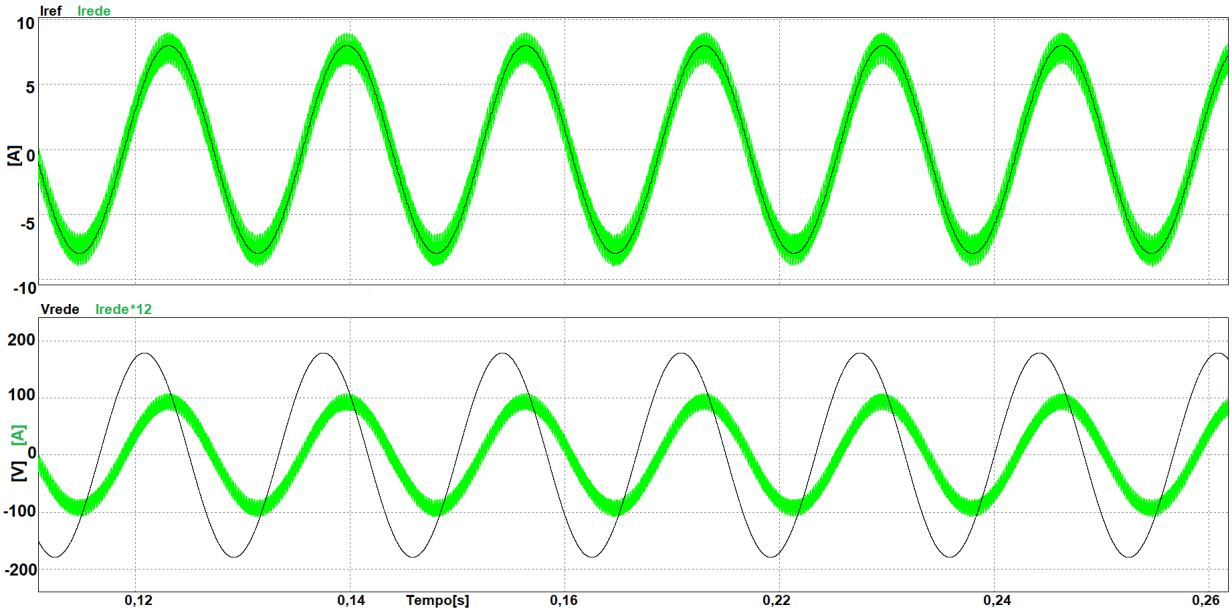


Fonte: Autor

A Figura 59, ilustra o que ocorre quando a compensação de $\pi/2$ não é adicionada no PLL, neste caso uma compensação de $\pi/4$ foi realizada, o controlador de corrente atua e mantêm a corrente no valor de referência, o PLL, porém, não sincroniza os sinais de corrente e tensão. O mesmo comportamento é observado com o circuito conectado à

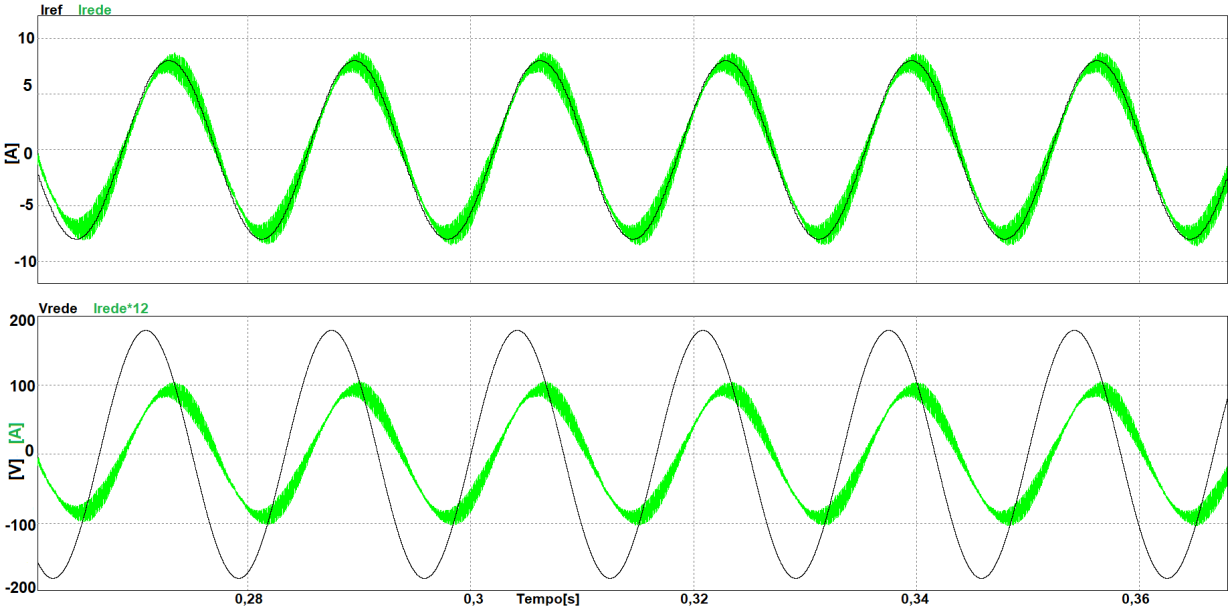
rede, Figura 60, com um aumento da distorção da forma de onda de corrente, devido o defasamento e o polo estar próximo a 0.

Figura 59 – Inversor desconectado, A = 0, com senoide interna do PLL defasada em $\pi/4$ da referência.



Fonte: Autor

Figura 60 – Inversor desconectado, A = 1, com senoide interna do PLL defasada em $\pi/4$ da referência.

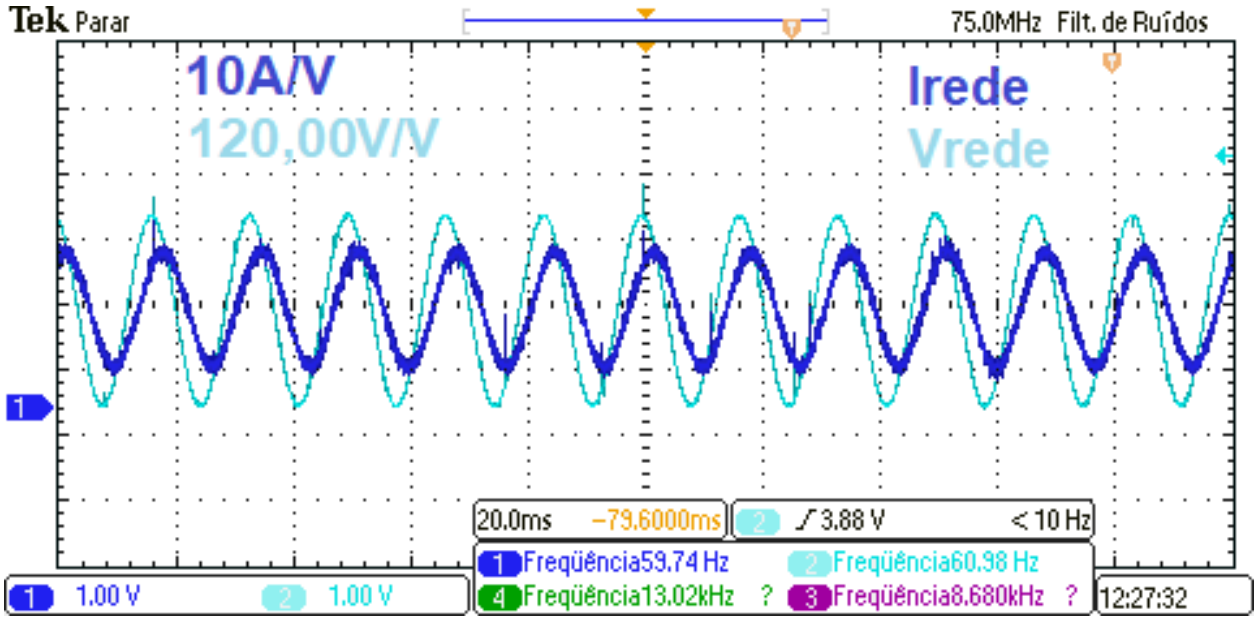


Fonte: Autor

A sequência de Figuras, 61, 62, 63 e 64, são os resultados para uma compensação de 45º no PLL para o inversor desconectado e conectado, respectivamente, conforme

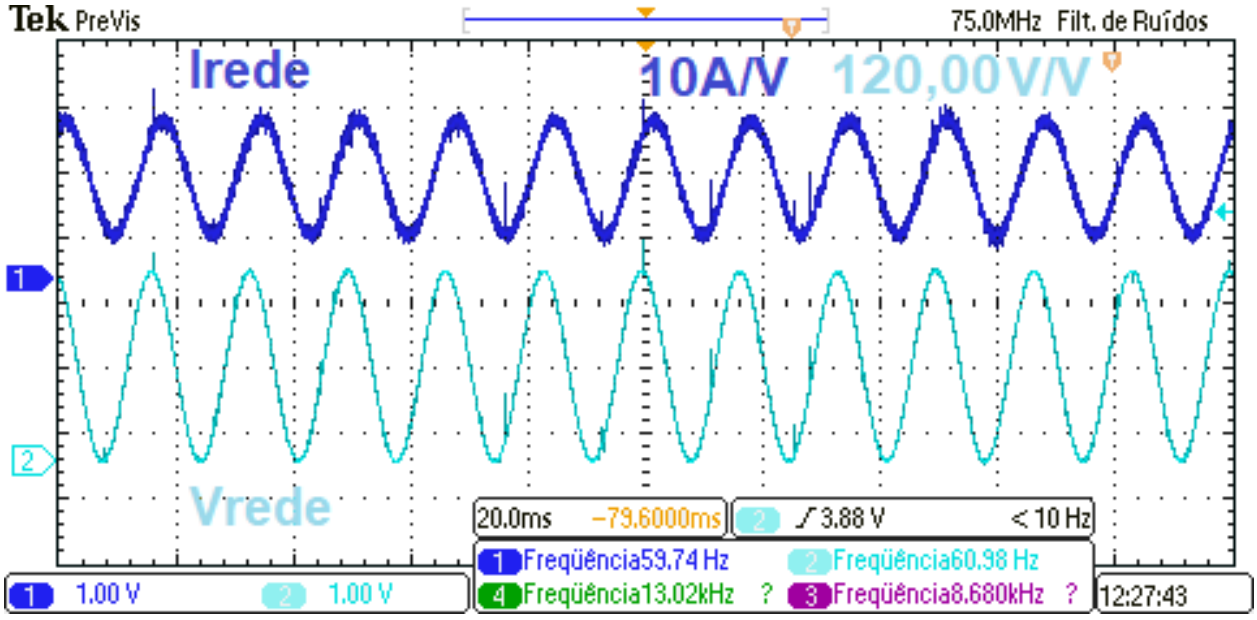
resultados simulados das Figuras 59 e 60. Em ambos os casos, a referência de corrente é de 8A de pico, a dinâmica, conforme esperado, é idêntica às simulações computacionais correspondentes.

Figura 61 – Resultado experimental, inversor desconectado, A = 0, com senoide interna do PLL defasada em $\pi/4$ da referência.



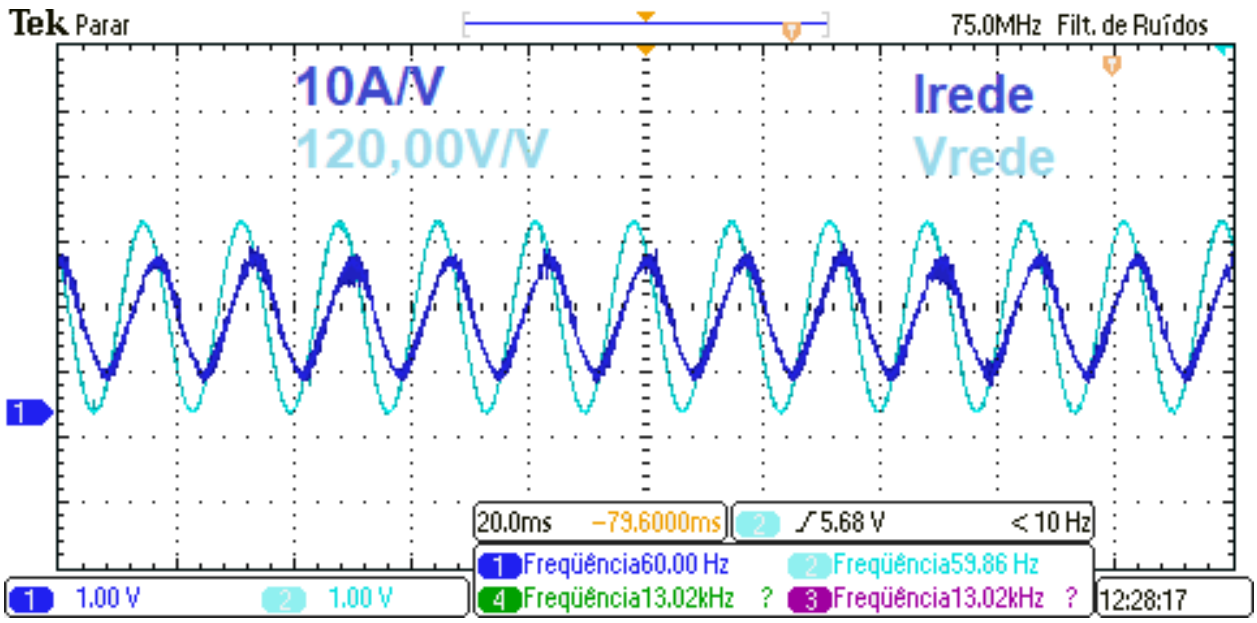
Fonte: Autor

Figura 62 – Resultado experimental, visualização individual dos sinais tensão e corrente da Figura 61.



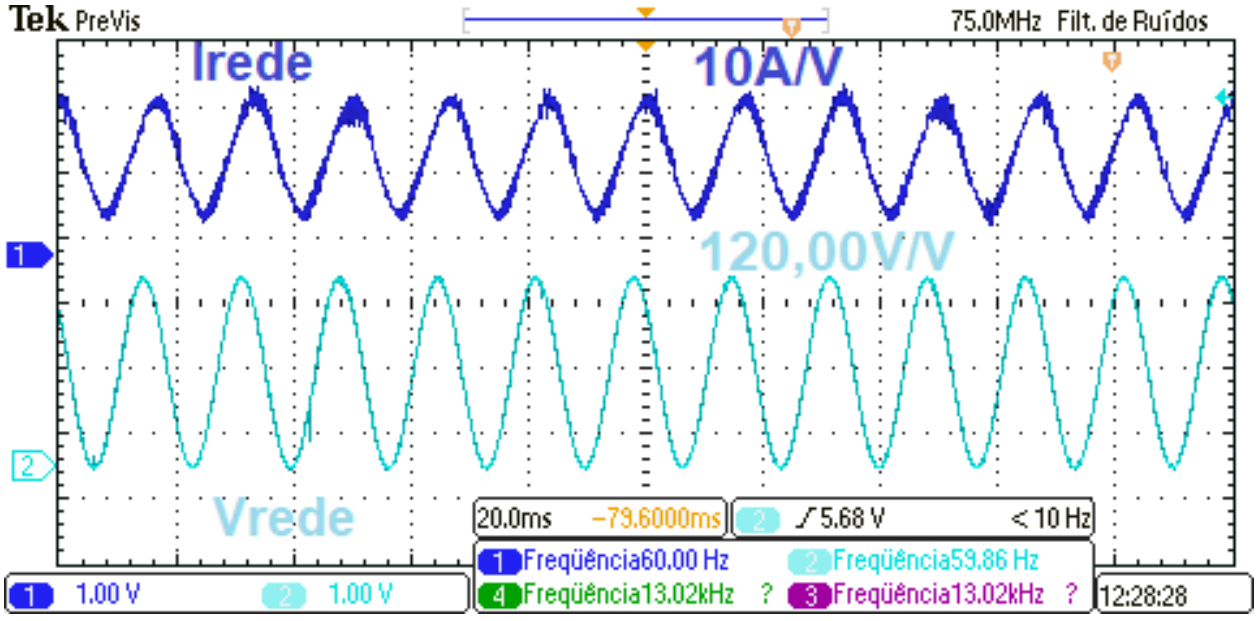
Fonte: Autor

Figura 63 – Resultado experimental, inversor conectado, A = 1, com senoide interna do PLL defasada em $\pi/4$ da referência.



Fonte: Autor

Figura 64 – Resultado experimental, visualização individual dos sinais de tensão e corrente da Figura 63.



Fonte: Autor

Os resultados experimentais, quando comparados aos resultados simulados via *software*, se apresentam em concordantes. Todos os dados colhidos se apresentam estáveis mesmo quando submetidos à variações em degrau dos valores de referências, não tendendo a uma instabilidade em momento algum. Os controladores atuam conforme projeto em

todas as situações, compensando rapidamente as perturbações e alterações dos valores das variáveis de interesse.

5 Conclusão

Conforme constatado pelos resultados experimentais e simulados, através da metodologia empregada, apresentados no capítulo 4, a simulação HIL foi realizada de forma adequada dado que a dinâmica da simulação HIL não foi alterada em nada e representou experimentalmente a simulação computacional de forma perfeitamente satisfatória.

O HIL se mostra uma ferramenta poderosa de simulação em tempo real, acelerando o processo de simulação, poupando tempo e memória de processamento de computadores. A depender complexidade do sistema simulado uma simulação computacional via *software* pode demandar muito tempo para ser finalizada além de prejudicar a realização de outras tarefas no computador utilizado, o HIL, por sua vez, realiza a simulação instantaneamente em *hardwares* dedicados exclusivamente ao processo de simulação.

Dado o procedimento metodológico simples e baixo custo do HIL empregado na realização do trabalho, é possível, também, a aplicação didática em laboratórios, para aprendizado de alunos. Alguns laboratórios podem não possuir certos equipamentos físicos, por questões financeiras, a simulação HIL de baixo custo apresentada aqui, se apresenta como uma solução viável neste quesito, proporcionando aos alunos uma ferramenta rápida e eficaz para visualização da dinâmica do sistema sendo simulado. Por se tratar de alunos em processo de aprendizagem, a aplicação do HIL em laboratórios evitaria que possíveis danos a certos equipamentos ocorram, caso a metodologia empregada no projeto dos controladores não seja adequada.

O baixo custo dos materiais utilizados na simulação HIL deste trabalho também é um atrativo para a realização da simulação em tempo real, o DSP custando cerca de U$ 33,79, e o ESP32 cerca de R$ 42,00, preço relativamente baixo dado o custo benefício recebido pela aplicação da técnica HIL, tendo como principal atrativo se ter uma simulação mais realista através da utilização de equipamentos reais.

5.0.1 Sugestão para trabalhos futuros

A seguir são apresentadas sugestões para continuidade do estudo abordado neste trabalho:

- Projeto e implementação de controle em quatro quadrantes em máquina CC.
- Projeto e implementação de controle em motor trifásico.
- Projeto e implementação de controle em inversor trifásico conectado à rede.
- Controle aplicado em microrede.

Referências

AGUIAR, C. *Estudo e Análise de Algoritmos de Detecção de Ilhamento em Sistemas de Geração Distribuída Conectados à Rede de Distribuição*. [S.l.]: Dissertação (Mestrado) - USP São Carlos, 2013. 11, 15

BACIC, M. On Hardware-in-the-loop simulation. *IEEE: Institute of Electrical and Electronics Engineers*, 2005. 17

BASTOS, R. et al. Low-cost hardware in the loop for real-time simulation of electric machines and electric drive. *IET: Institution of Engineering and technology*, 2020. 17, 22

BIM, E. *Máquinas Elétricas e Acionamento, 2ª Edição*. [S.l.]: Elsevier, 2012. 14

BOSE, B. K. *Modern Power Electronics and AC Drives*. [S.l.]: Pearson Education, 2015. 14

BOYLESTAD, R. L. *Introdução à Análise de Circuitos, 12ª Edição*. [S.l.]: Pearson, 2011. 8

BUSO, S.; MATTAVELLI, P. *Digital Control in Power Electronics*. [S.l.]: Morgan Claypool Publishers, 2006. 16, 18

CHAPMAN, S. J. *Electric Machinery Fundamentals, 5th Edition*. [S.l.]: McGraw-Hill, 2013. 5, 6

DORF, R. C.; BISHOP, R. H. *Sistemas de Controle Modernos, 8ed Rio de Janeiro*. [S.l.]: LTC, 2001. 12, 37

ESPRESSIF. *ESP-IDF Programming Guide*. [S.l.]: Espressif Systems, 2021. 119

FRANKLIN, G. F.; POWELL, J. D.; EMANI-NAEINI, A. *Feedback control of dynamic systems,7th Edition*. [S.l.]: Pearson, 2014. 11, 12

HART, D. W. *Power Electronics*. [S.l.]: McGraw-Hill, 2011. 8, 9, 10, 14

HEMETSBERGER, W.; SCHMELA, M.; CHIANETTA, G. *Global Market Outlook for Solar Power*. [S.l.]: Solar Power Europe, 2021. 2

IEA. *Global EV Outlook 2019*. [S.l.]: International Energy Agency, 2019. 1

IEA. *Solar PV - Analysis*. [S.l.]: International Energy Agency, 2021. 1

LU, B. et al. A Low-Cost Real-Time Hardware-in-the-Loop Testing Approach of Power Electronics Controls. *IEEE: Institute of Electrical and Electronics Engineers*, 2007. 2, 18

PADUA, M. S. d.; DECKMAN, S. M. *Técnicas digitais para sincronização com a rede elétrica, com aplicação em geração distribuida*. [S.l.]: Dissertação (Mestrado) - Universidade Estadual de Campinas, 2006. 10

PETRY, C. A. *Acionamentos de Motores CC*. [S.l.]: Pearson Education, 2015. 5

RASHID, M. H. *Power Electronics Handbook, 4ª Edição*. [S.l.]: Academic Press, 2001. 13, 21

RASHID, M. H. *Eletrônica de Potência, Circuitos, Dispositivos e Aplicações, 4ª Edição*. [S.l.]: Pearson, 2014. 2, 8, 9

TI. *User's Guide LAUNCHXL-F28379D Overview*. [S.l.]: Texas Instruments, 2019. 119, 120

TYPHOON-HIL. Do you trust your shipboard power management system? *Disponível em < https://www.typhoon-hil.com/applications/marine-testing>*, 2019. 18

UMANS, S. D. *Fitzgerald Kingsley's Electric Machinery, 7th edition*. [S.l.]: McGraw-Hill Education, 2014. 5, 12

VILLALVA, M. Conversor eletrônico de potência trifásico para sistema fotovoltaico conectado à rede elétrica. *UNICAMP. Campinas*, 2010. 21

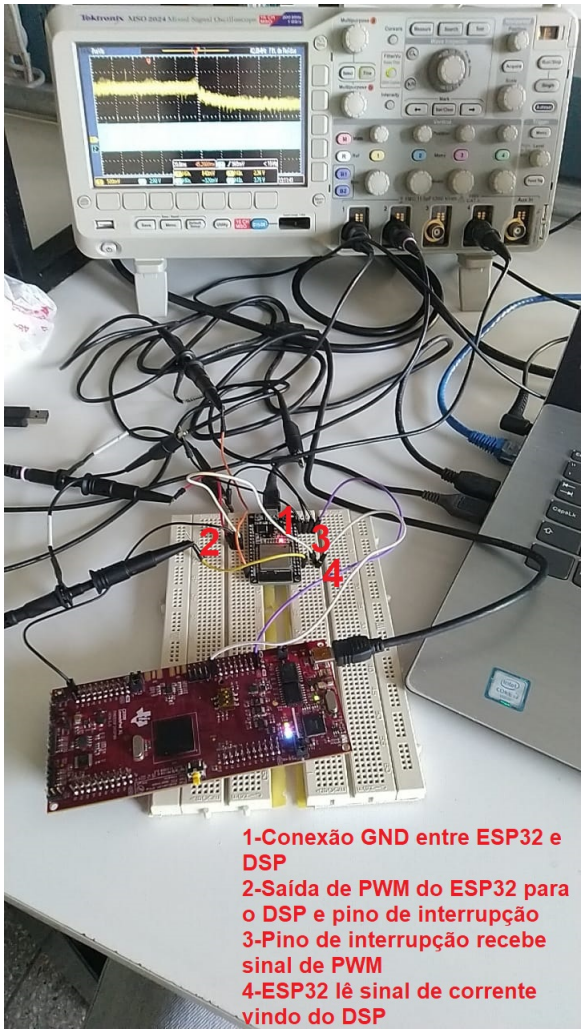
WEG. *Motores elétricos de corrente alternada*. [S.l.]: Weg, 2012. 1

A Apêndices

A.1 Simulação HIL de um conversor *boost*

Apesar de não fazer parte da proposta inicial do trabalho, um conversor CC CC também fora implementado e testado, tendo sua corrente controlada por um controlador PI. A Figura 65 apresenta o circuito montado em bancada, com uma conFiguração um pouco diferente do circuito da Figura 26, sendo lida apenas a corrente, possuindo um único pino para PWM, cujo *duty cycle* é atualizado a cada ação do controle PI.

Figura 65 – HIL implementado em bancada para simulação do conversor *boost*.

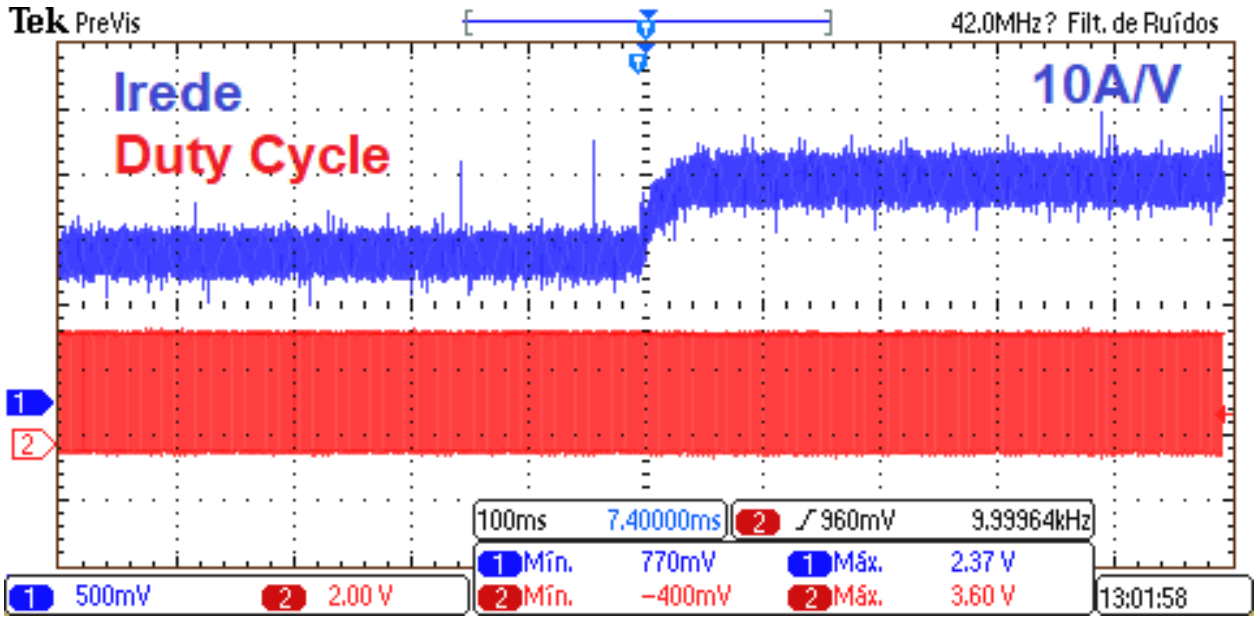


Fonte: Autor

No processo de simulação, o controlador de corrente teve sua corrente de referência alterada em degrau a cada 1,5s, entre 10A e 15A. As Figuras 66 e 67. No início da Figura 66, o valor de corrente medido é de aproximadamente 10,5A, já na segunda metade, o

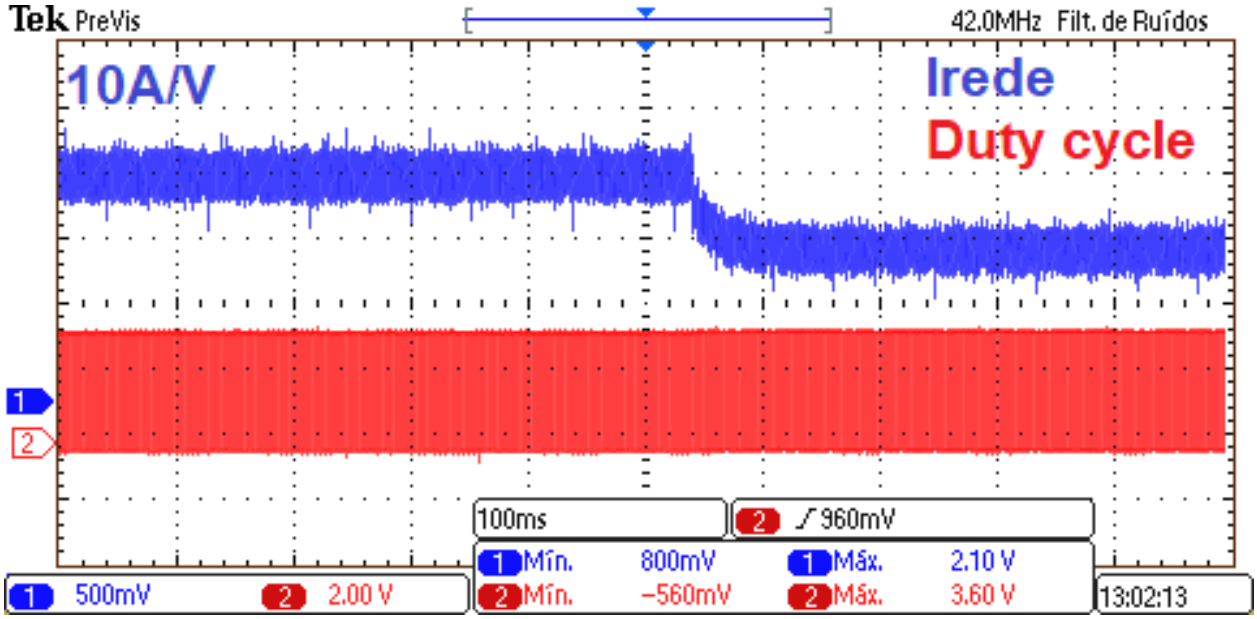
valor medido é de aproximadamente 15,5A. A mesma dinâmica é observada na Figura 67, sendo medido inicialmente 15,5A e na segunda metade 10,5A.

Figura 66 – Resultado experimental, visualização individual dos sinais de corrente e *duty cycle*, alteração de referência de 10A para 15A.



Fonte: Autor

Figura 67 – Resultado experimental, visualização individual dos sinais de corrente e *duty cycle*, alteração de referência de 15A para 10A.



Fonte: Autor

A.2 Códigos referentes às simulações computacionais e HIL da máquina CC

Programa A.1 – Controlador de velocidade e corrente em cascata, código do *Psim*

```
// Parametros dos controladores

#include <Stdlib.h>
#include <String.h>


float kii=3.0, kpi=0.005, kiv=0.196,kpv=0.1862;
float intgi=0.0,intgv=0.0,pii=0.0,piv=0.0,propi=0.0,propv=0.0;
float vref=0.0,iaref=0.0,vmec=0,imed=0;
float erroi=0.0,errov=0.0;
float ts=0.0002;
float ia=0.0;


vref=in[0];      //{
ia = in[1];      //{ Entrada de variaveis
vmec=in[2];      //{

errov=vref-vmec;
propv = kpv*errov;
intgv = intgv + kiv*errov*ts;
if(intgv<0.0)
  intgv=0.0;
if(intgv>30.0)
  intgv=30.0;       //{ PI de velocidade
piv = propv + intgv;

if(piv<0.0)
  piv=0.0;
if(piv>30.0)
  piv=30.0;

iaref = piv;          //Referencia de corrente


erroi = iaref-ia;
propi = kpi*erroi;
intgi = intgi + erroi*ts*kii;
if(intgi<0.0)
  intgi=0.0;
if(intgi>1.0)
  intgi=1.0;        //{ PI de corrente
```

```
pii = intgi+propi;

if(pii<0.0)
   pii=0.0;
if(pii>1.0)
   pii=1.0;

out[0]=pii;            //{ Saida que vai ao PWM
```

Programa A.2 – Modelo da máquina CC discretizado, código do *Psim*

```
#include <Stdlib.h>
#include <String.h>


int DA=0;    //Duty cicle

// parametros da maquina

float Vt=0.0, Vin=0, Vdc=120.0;
float Ea=0, La=0.01, Ra=0.5, Lf=0.02, Laf=0.0, ia=0, Dia=0, Kd=0.05;
float Rf=75.0, iff=1.6;
float Wn=0, Dwn=0, Vmec=0;
float Tem=0, Tload=1.0;
float IaR=10.0, IfR=1.6, WnR=125.664, J=0.4, VtR=120.0;
float La1=0, J1=0, Lf1=0;
float if_bat=0.0;
float T=0.00000125;

DA=in[0];       //{Entrada de variaveis
Tload=in[1];        //{


          Laf= (VtR - IaR*Ra)/(IfR*WnR);   //Equacoes da maquina CC
           La1=1/La;
           Lf1=1/Lf;
           J1=1/J;


                      Vt=Vdc*DA;      // Tensao aplicada na armadura


                      Dia = (Vt - Ea - ia*Ra)*La1; // Variacao de corrente

                      Dwn=(Tem-Tload - Wn*Kd)*J1;  // Variacao de velocidade

                      ia = ia + Dia*T;      //Corrente de armadura
                      if(ia<=0.0){ia=0.0;}  //Limitador para evitar corrente
                          negativa
                      iff =1.6;       //Corrente de campo
                      Wn = Wn + Dwn*T;      //Velocidade da m quina

                     if(Wn<=0){Wn=0.0;}     //Limitador para evitar velocidade
                         negativa

                      Ea=Laf*iff*Wn;      //Tensao induzida na armadura
                      Tem=Laf*iff*ia;     //Torque produzido pela maquina
```

```
                    Vmec=Wn*9.549;      //Velocidade rad/s para rpm


out[0]=ia;      //{Corrente de armadura lida pelo controlador
out[1]=iff;     //{Corrente de campo para medicao
out[2]=Vmec;      //{Velocidade lida pelo controlador
```

Programa A.3 – Controlador de velocidade e corrente em cascata, código implementado no ESP32 para simulação HIL

```
//
//  Codigo implementado para controle da maquina
//  TCC
//  Pedro Henrique Barcellos Linhares - 15.1.5907
//

const int pwmfixo = 16;             //Pino de PWM fixo
const int pwmchave = 5        //Pino de PWM de chaveamento, D
const int f = 5000;                 //Frequencia do pwm
const int resolution = 8;    // Resolucao do PWM
const int pwmchannel = 0;    //Canal do PWM fixo
const int D = 1;             //Canal do PWM de chaveamento D
const int pwminterrupt = 4;         //Pino que recebe interrupcao
const int tempInterrupt = 22;       //Pino para medicao de tempo de
    interrupcao
int flag = 0;         // Flag de interrupcao

// Controlador

const int entcorr=36;               //Pino que recebe o sinal de corrente
    vindo do dsp
const int entvmec=39;             //Pino que recebe o sinal de Velocidade
    vindo do DSP
float count = 0.0;
float kii=3.0, kpi=0.005, kiv=0.196,kpv=0.1862;
float intgi=0.0,intgv=0.0,pii=0.0,piv=0.0,propi=0.0,propv=0.0;
float vref=0.0,iaref=0.0,vmec=0,imed=0;         //{Constantes referentes ao
    controlador
float erroi=0.0,errov=0.0;
float ts=0.0002;
float ia=0.0;
int dutyCycle=1;


void setup() {
  // put your setup code here, to run once:
     pinMode(pwm, OUTPUT);
                                                          //Saida PWM
     pinMode(pwminterrupt, INPUT);
                                                  //Pino que reconhece a
         interrupcao
     pinMode(tempInterrupt, OUTPUT);
                                                 //Pino que verifica o tempo
         de execucao da interrupcao
```

```
    ledcSetup(pwmchannel,f,resolution);            //Pwm fixo para entrar
        na interrupcao
    ledcAttachPin(pwmfixo, pwmchannel);
    attachInterrupt(digitalPinToInterrupt(pwminterrupt),ISR,RISING);
            //Interrupcao na borda de subida do pwm fixo
    ledcWrite(pwmchannel,dutyCycle);
                                                //Inicializa o PWM fixo com o
         duty declarado
    ledcSetup(D,f,resolution);                  //PWM responsavel pelo
        chaveamento D
    ledcAttachPin(pwmchave,D);

    //ledcWrite(D,0);
    //delay(10000);                    // descomentar para dinamica da
        maquina iniciar em velocidade 0
}

void loop() {

  if (flag == 1.0)
  {
    digitalWrite(tempInterrupt, HIGH);          // Ativa pino de contagem
        de tempo da interrupcao
    //count=count+1;                // Contador para contagem de tempo para
        simulacao de alteracao de referencia de velocidade
    ia = analogRead(entcorr)*0.0075;            // Pino que recebe o sinal de
         corrente vindo do DSP
    vmec = analogRead(entvmec)*0.495;           // Pino que recebe o sinal de
         velocidade vindo do DSP
    //if (count == 35000)
      //{
        //vref = 850.0;
      //}                // Descomentar os if's quando simular variacao de
          referencia de velocidade
    //if (count == 70000)
      //{
       // vref = 550.0;
        //count = 0;
      //}
    errov=vref-vmec;
    propv = kpv*errov;
    intgv = intgv + kiv*errov*ts;
    if(intgv<0.0)
      {intgv=0.0;}
    if(intgv>30.0)                //{ Controlador PI da velocidade
      {intgv=30.0;}
    piv = propv + intgv;
```

```
    if(piv<0.0)
      {piv=0.0;}
    if(piv>30.0)
      {piv=30.0;}
    iaref = piv;

    erroi = iaref-ia;
    propi = kpi*erroi;
    intgi = intgi + erroi*ts*kii;
    if(intgi<0.0)
      {intgi=0;}
    if(intgi>0.98)
      {intgi=0.98;}              //{ Controlador PI da corrente
    pii = intgi+propi;

    if(pii<0.0)
      {pii=0.0;}
    if(pii>0.98)
      {pii=0.98;}

    dutyCycle = 255.0*(pii);              //Atualiza o Duty Cycle
    flag = 0.0;                           //Reseta a flag
    digitalWrite(tempInterrupt, LOW); //Desliga pino de medicao de tempo de
        interrupcao
    ledcWrite(pwmchannel,100);            //Mantem o PWM da interrupcao
    ledcWrite(D,dutyCycle);               //Atualiza o duty de chaveamento
  }
  }

void ISR()     //Funcao da interrupcao
{
 flag = 1.0;
}
```

Programa A.4 – Modelo da máquina CC discretizado, código implementado no DSP para simulação HIL

```
#include "F28x_Project.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


//
// Globals
//


volatile struct DAC_REGS* DAC_PTR[4] = {0x0,&DacaRegs,&DacbRegs,&DaccRegs};
      // necessario para o DAC funcionar, Mistrio da f


#define EPWM1_TIMER_TBPRD  400 // 400khz  200khz/fsw
#define EPWM1_MAX_CMPA     100
#define EPWM1_MIN_CMPB      50


//Defines para frequencia de chaveamento do PWM
#define EPWM2_TIMER_TBPRD  400 // 400khz  200khz/fsw
#define EPWM2_MAX_CMPA     100
#define EPWM2_MIN_CMPB      50



//Defines para frequencia de chaveamento do PWM
#define EPWM3_TIMER_TBPRD 400  //       200Mhz/(Freq);
#define EPWM3_MAX_CMPA     250
#define EPWM3_MIN_CMPB      50


//Defines para frequencia de chaveamento do PWM
#define EPWM4_TIMER_TBPRD 400      //  200Mhz/(Freq);
#define EPWM4_MAX_CMPA     250
#define EPWM4_MIN_CMPB      50



//Prototipo das funcoes
void InitEPwm2Example(void);     //Configura o PWM e inicializa
void InitEPwm3Example(void);     //Configura o PWM e inicializa
void InitEPwm4Example(void);     //Configura o PWM e inicializa
```

```
void InitEPwm1Example(void);      //Configura o PWM e inicializa

void ConfigureADC(void);          //Configura e inicializa o ADC
void SetupADCSoftware(void);      //Configura os canais de aquisicao do ADC

interrupt void adca1_isr(void); //interrupcao do ADC

void configureDAC(Uint16 dac_num);

int DA=0, DB=0, DC=0, DD=0;
int botao1=0, botao2=0, botao3=0, botao4, chave=0;

// parametros da maquina



float Vt=0.0, Vin=0, Vdc=120.0;
float Ea=0, La=0.01, Ra=0.5, Lf=0.02, Laf=0.0, ia=0, Dia=0, Kd=0.05;
float Rf=75.0, iff=1.6;
float Wn=0, Dwn=0, Vmec=0;
float Tem=0, Tload=1.0;
float IaR=10.0, IfR=1.6, WnR=125.664, J=0.4, VtR=120.0;
float La1=0, J1=0, Lf1=0;
float if_bat=0.0;
float T=0.0000025;

int IA_DAC=0.0, VM_DAC=0;

void main(void){
    //Inicializa o sistema (watchdogtimer, pll, sysclock)
    InitSysCtrl();

    //Habilita as GPIO
    InitGpio();

    //Habilita o PWM2
    CpuSysRegs.PCLKCR2.bit.EPWM2=1;
    CpuSysRegs.PCLKCR2.bit.EPWM3=1;
    CpuSysRegs.PCLKCR2.bit.EPWM4=1;
    CpuSysRegs.PCLKCR2.bit.EPWM1=1;

    //Habilita o pino do PWM2
    InitEPwm1Gpio();
    InitEPwm2Gpio();
    InitEPwm3Gpio();
    InitEPwm4Gpio();
```

```
//Limpa todas as interrupcoes e inicializa a tabela de vetores de
    interrupcao
DINT;

//Inicializa em seu estado padrao os registradores de controle de
    interrupcoes PIE
InitPieCtrl();



//Desabilita interrupcoes da CPU e limpa todas as flags de interrupcao
    da CPU
IER = 0x0000;
IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
InitPieVectTable();

//Interrupcoes
EALLOW; //Habilita escrita em registradores protegidos
PieVectTable.ADCA1_INT = &adca1_isr; //funcao para interrupcao do ADCA
EDIS;    //Desabilita escrita em registradores protegidos

// Inicializa o PWM2 e sincroniza
EALLOW;
CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;
EDIS;

InitEPwm4Example();
InitEPwm2Example();
InitEPwm3Example();
InitEPwm1Example();


EALLOW;
CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
ClkCfgRegs.PERCLKDIVSEL.bit.EPWMCLKDIV=0; /// seta div do pwm
EDIS;

//Habilita interrupcoes globais
IER |= M_INT1; //Habilita grupo 1 das interrupcoes
EINT;  // Enable Global interrupt INTM
ERTM;  // Enable Global realtime interrupt DBGM
```

```
PieCtrlRegs.PIEIER1.bit.INTx1 = 1;


EALLOW;
    GpioCtrlRegs.GPBPUD.bit.GPIO40= 0; //Enable pullup on GPI32; porta
        36 habilitada
    GpioCtrlRegs.GPBMUX1.bit.GPIO40= 0; //GPI42= porta IO
    GpioCtrlRegs.GPBDIR.bit.GPIO40= 0; //GPI42= input

    GpioDataRegs.GPBCLEAR.bit.GPIO40=1;


    GpioCtrlRegs.GPBPUD.bit.GPIO41= 0; //Enable pullup on GPI32; porta
        36 habilitada
    GpioCtrlRegs.GPBMUX1.bit.GPIO41= 0; //GPI43= porta IO
    GpioCtrlRegs.GPBDIR.bit.GPIO41= 0; //GPI43= input

    GpioDataRegs.GPBCLEAR.bit.GPIO41=1;


    GpioCtrlRegs.GPBPUD.bit.GPIO52= 0;
    GpioCtrlRegs.GPBMUX2.bit.GPIO52= 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO52= 0; // GPIO as input

    GpioDataRegs.GPBCLEAR.bit.GPIO52=1;


    GpioCtrlRegs.GPCPUD.bit.GPIO65= 0;
    GpioCtrlRegs.GPCMUX1.bit.GPIO65= 0;
    GpioCtrlRegs.GPCDIR.bit.GPIO65= 0;    // GPIO as input

    GpioDataRegs.GPCCLEAR.bit.GPIO65=1;


    GpioCtrlRegs.GPCPUD.bit.GPIO94= 0;
    GpioCtrlRegs.GPCMUX2.bit.GPIO94= 0;
    GpioCtrlRegs.GPCDIR.bit.GPIO94= 1;    // GPIO as outtput

    GpioDataRegs.GPCCLEAR.bit.GPIO94=1;


    GpioCtrlRegs.GPBPUD.bit.GPIO32= 0;
    GpioCtrlRegs.GPBMUX1.bit.GPIO32= 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO32= 0;

    GpioDataRegs.GPBCLEAR.bit.GPIO32=1;
```

```
GpioCtrlRegs.GPAPUD.bit.GPIO19= 0;
GpioCtrlRegs.GPAMUX2.bit.GPIO19= 0;
GpioCtrlRegs.GPADIR.bit.GPIO19= 0; //= input

GpioDataRegs.GPACLEAR.bit.GPIO19=1;


GpioCtrlRegs.GPAPUD.bit.GPIO18= 0;
GpioCtrlRegs.GPAMUX2.bit.GPIO18= 0;
GpioCtrlRegs.GPADIR.bit.GPIO18= 0; //= input

GpioDataRegs.GPACLEAR.bit.GPIO18=1;


GpioCtrlRegs.GPCPUD.bit.GPIO67= 0;
GpioCtrlRegs.GPCMUX1.bit.GPIO67= 0;
GpioCtrlRegs.GPCDIR.bit.GPIO67= 0;    // GPIO as output

GpioDataRegs.GPCCLEAR.bit.GPIO67=1;


GpioCtrlRegs.GPDPUD.bit.GPIO111= 0;
GpioCtrlRegs.GPDMUX1.bit.GPIO111= 0;
GpioCtrlRegs.GPDDIR.bit.GPIO111= 0;    // GPIO as output

GpioDataRegs.GPDCLEAR.bit.GPIO111=1;


                                EPwm2Regs.CMPA.bit.CMPA = 100;
                                EPwm2Regs.CMPB.bit.CMPB =100;

                                EPwm6Regs.CMPA.bit.CMPA = 12500;
                                EPwm6Regs.CMPB.bit.CMPB =12500;


EDIS;


// definição das constantes;


 Laf= (VtR - IaR*Ra)/(IfR*WnR);
 La1=1/La;
 Lf1=1/Lf;
 J1=1/J;
```

```
    // Configura o ADC e inicializa
    ConfigureADC();
    SetupADCSoftware();
    configureDAC(1);
    configureDAC(2);

    //      configureDAC(1); para DACa (Na porta ADCINA0)
    //      configureDAC(2); para DACb (Na porta ADCINA1)
    //      configureDAC(3); para DACc (Na porta ADCINB1)

    do{

    }while(1);
}


void InitEPwm1Example()
{
    // Setup TBCLK
    EPwm1Regs.TBPRD = EPWM1_TIMER_TBPRD;           // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm1Regs.TBPHS.bit.TBPHS = 0x0000;            // Phase is 0
    EPwm1Regs.TBCTR = 0x0000;                      // Clear counter


    // Setup counter mode
    EPwm1Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
        SYSCLKOUT
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    //  Valores de compara      o inicial
    EPwm1Regs.CMPA.bit.CMPA = EPWM1_MAX_CMPA;    // valor inicial do duty p/
        pwmA
```

```
    EPwm1Regs.CMPB.bit.CMPB = EPWM1_MIN_CMPB;      // valor inicial do duty
        p/ pwmB


    EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;             // Set PWM1A on Zero
    EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // Clear PWM1A on event A
        ,
                                                          // up count

    EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;             // Set PWM1B on Zero
    EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;           // Clear PWM1B on event B
        ,
                                                          // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;           // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;         // Clear PWM2A on event B,
        down count

    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}


// InitEPwm2Example - Inicializa ePWM2 e configura
void InitEPwm2Example()
{
    // Setup TBCLK
    EPwm2Regs.TBPRD = EPWM2_TIMER_TBPRD;           // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm2Regs.TBPHS.bit.TBPHS = 0x0000;            // Phase is 0
    EPwm2Regs.TBCTR = 0x0000;                      // Clear counter


    // Setup counter mode
    EPwm2Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
        SYSCLKOUT
    EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
```

```
    EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    //EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;
    EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
    //  Valores de compara o inicial
    EPwm2Regs.CMPA.bit.CMPA = EPWM2_MAX_CMPA;    // valor inicial do duty p/
        pwmA
    EPwm2Regs.CMPB.bit.CMPB = EPWM2_MIN_CMPB;     // valor inicial do duty
        p/ pwmB


      EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;            // Set PWM1A on Zero
      EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;          // Clear PWM1A on event
           A,
                                                    // up count

      EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;            // Set PWM1B on Zero
      EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;          // Clear PWM1B on event
           B,
                                                    // up count


    // Set actions
    //EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;          // Set PWM2A on event A,
        up count
    //EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;        // Clear PWM2A on event B,
         down count


    //EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm2Regs.AQCTLB.bit.CBD = AQ_SET;

    // Interrupt where we will change the Compare Values
    EPwm2Regs.ETSEL.bit.SOCAEN = 1;      //Habilita o pulso de EPWM2SOCA
    EPwm2Regs.ETSEL.bit.SOCASEL = 2;     // Este bit determina quando o
        pulso de convers o do ePWMA ser     gerado
    EPwm2Regs.ETPS.bit.SOCAPRD = 1;      //Determina quantos periodos devem
        ocorrer antes de gerar o pulso do EPWM2SOCA
}

void InitEPwm3Example()
{
    // Setup TBCLK
    EPwm3Regs.TBPRD = EPWM3_TIMER_TBPRD;          // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm3Regs.TBPHS.bit.TBPHS = 0x0000;           // Phase is 0
```

```
    EPwm3Regs.TBCTR = 0x0000;                       // Clear counter


    // Setup counter mode
    EPwm3Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE;         // Disable phase loading
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;        // Clock ratio to
        SYSCLKOUT
    EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    //  Valores de compara      o inicial
    EPwm3Regs.CMPA.bit.CMPA = EPWM3_MAX_CMPA;    // valor inicial do duty p/
        pwmA
    EPwm3Regs.CMPB.bit.CMPB = EPWM3_MIN_CMPB;      // valor inicial do duty
        p/ pwmB


    EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;              // Set PWM1A on Zero
    EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;            // Clear PWM1A on event A
        ,
                                                          // up count

    EPwm3Regs.AQCTLB.bit.ZRO = AQ_SET;              // Set PWM1B on Zero
    EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;            // Clear PWM1B on event B
        ,
                                                          // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;           // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;         // Clear PWM2A on event B,
        down count

    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}
```

```
void InitEPwm4Example()
{
    // Setup TBCLK
    EPwm4Regs.TBPRD = EPWM4_TIMER_TBPRD;           // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm4Regs.TBPHS.bit.TBPHS = 0x0000;            // Phase is 0
    EPwm4Regs.TBCTR = 0x0000;                      // Clear counter


    // Setup counter mode
    EPwm4Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm4Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
        SYSCLKOUT
    EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    //  Valores de compara o inicial
    EPwm4Regs.CMPA.bit.CMPA = EPWM4_MAX_CMPA;   // valor inicial do duty p/
        pwmA
    EPwm4Regs.CMPB.bit.CMPB = EPWM4_MIN_CMPB;     // valor inicial do duty
        p/ pwmB


    EPwm4Regs.AQCTLA.bit.ZRO = AQ_SET;             // Set PWM1A on Zero
    EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // Clear PWM1A on event A
        ,
                                                          // up count

    EPwm4Regs.AQCTLB.bit.ZRO = AQ_SET;             // Set PWM1B on Zero
    EPwm4Regs.AQCTLB.bit.CBU = AQ_CLEAR;           // Clear PWM1B on event B
        ,
                                                          // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;           // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;         // Clear PWM2A on event B,
        down count
```

```
    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}

void ConfigureADC(void)
{
    EALLOW;

    //write configurations
    AdcaRegs.ADCCTL2.bit.PRESCALE = 6;
    AdcbRegs.ADCCTL2.bit.PRESCALE = 6;
    //Seta o ADC que sera utilizado, a resolucaoo e o modo de operacao
    AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
    AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);

    //Set pulse positions to late
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    //power up the ADC
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;

    //delay for 1ms to allow ADC time to power up
    DELAY_US(1000);

    EDIS;
}

void SetupADCSoftware(void)
{
    Uint16 acqps = 14;  //75ns aquisition time

    //Seleciona os canais para conversao, tempo de aquisicao, e trigger
        select (ePWM2 SOCA)
    //ADCA
    EALLOW;
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 0x00;  //SOC0 will convert pin A0
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = acqps; //sample window is acqps +1
        SYSCLK cycles
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 0x07; //trigger on ePWM1 SOCA/C
    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 0x02;  //SOC1 will convert pin A2
    AdcaRegs.ADCSOC1CTL.bit.ACQPS = acqps; //sample window is acqps +1
        SYSCLK cycles
```

```
    AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 0x07; //trigger on ePWM1 SOCA/C

    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //end of SOC1 will set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared


    //ADCB
    AdcbRegs.ADCSOC0CTL.bit.CHSEL = 0x00;  //SOC0 will convert pin B0
    AdcbRegs.ADCSOC0CTL.bit.ACQPS = acqps; //sample window is acqps +1
        SYSCLK cycles
    AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 0x07; //trigger on ePWM1 SOCA/C

    AdcbRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //end of SOC1 will set INT1 flag
    AdcbRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
    EDIS;
}

//Interrupcao do ADCA
interrupt void adca1_isr(void)
{

    GpioDataRegs.GPCSET.bit.GPIO94=1;    // j5-46


                    // leitura=AdcaResultRegs.ADCRESULT0;
                   // Tcarga= 0.0;
                    //leitura1=AdcaResultRegs.ADCRESULT1;
                    //leitura2=AdcaResultRegs.ADCRESULT2;
                    //leitura3=AdcaResultRegs.ADCRESULT3;


                     DA=GpioDataRegs.GPBDAT.bit.GPIO40;    //j5-50
            //          DB=GpioDataRegs.GPBDAT.bit.GPIO41;
            //          DC=GpioDataRegs.GPBDAT.bit.GPIO52;
            //          DD=GpioDataRegs.GPCDAT.bit.GPIO65;

                    // botao1=GpioDataRegs.GPBDAT.bit.GPIO32;
                    // botao2=GpioDataRegs.GPADAT.bit.GPIO19;
                    // botao3=GpioDataRegs.GPADAT.bit.GPIO18;
                    // botao4=GpioDataRegs.GPCDAT.bit.GPIO67;
                    // chave=GpioDataRegs.GPDDAT.bit.GPIO111;

                    //  Tload = 10.0*(1-chave);
```

```
                Vt=Vdc*DA;
                Dia = (Vt - Ea - ia*Ra)*La1;

                //Dwn=(Tem-Tload - Wn*Kd)*J1;
                Dwn=(Tem-Tload)*J1;

                ia = ia + Dia*0.0000025;
                if(ia<=0.0){ia=0.0;}
                iff =1.6;
                Wn = Wn + Dwn*0.0000025;

                if(Wn<=0){Wn=0.0;}

                Ea=Laf*iff*Wn;
                Tem=Laf*iff*ia;

                 Vmec=Wn*9.549;  // em rpm

                  IA_DAC=ia*133.3333;
                  VM_DAC=Vmec*2.666;

                   if(VM_DAC>4000){VM_DAC=4000.0;}
                   if(IA_DAC>4000){IA_DAC=4000.0;}

               DAC_PTR[1]->DACVALS.all = IA_DAC;        // 30-j3
                    escala de 10A = 1V
               DAC_PTR[2]->DACVALS.all = VM_DAC;    // 70 - j7
                    escala de 1200 rpm = 2.4V

//             EPwm1Regs.CMPA.bit.CMPA = ia*2.0;    //0 a 1200 rpm
//             EPwm1Regs.CMPB.bit.CMPB = Ea*5.0 + 200;    //0 a 1200
    rpm

//             EPwm3Regs.CMPA.bit.CMPA =  Vmec*0.1 + 200;    //0 a
    +-2000 rpm
//             EPwm3Regs.CMPB.bit.CMPB =  iff*10.0 +200;

 //             EPwm4Regs.CMPA.bit.CMPA = pot;
 //             EPwm4Regs.CMPB.bit.CMPB = direcao*400;



                //EPwm4Regs.CMPB.bit.CMPB = 225*senoide[TETA] + 225;
```

```
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

      //GpioDataRegs.GPBTOGGLE.bit.GPIO42=1;
    GpioDataRegs.GPCCLEAR.bit.GPIO94=1;


}


//
// configureDAC - Enable and configure the requested DAC module
//
void configureDAC(Uint16 dac_num)
{
    EALLOW;

    DAC_PTR[dac_num]->DACCTL.bit.DACREFSEL = 1;  // 1 referencia de tensao
        interna, 0 referencia externa 3.3V no pino ADCINB0
    DAC_PTR[dac_num]->DACOUTEN.bit.DACOUTEN = 1; // habilita o DAC
    DAC_PTR[dac_num]->DACVALS.all = 0;           // inicia o DAC em zero

    DELAY_US(10); // Delay for buffered DAC to power up

    EDIS;
}
```

A.3 Códigos referentes às simulações computacionais e HIL do inversor monofásico

Programa A.5 – Controlador de corrente + ressonante

```
#include <Stdlib.h>
#include <String.h>


//==================== ENTRADA ==============================

float Ts = 0.000083333, Vrede=0.0, D=0.0, Irede=0.0;

//Variaveis PLL

float media [200] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

float SENO[360]={0,0.0175,0.0349,0.0523,0.0698,0.0872,0.1045,0.1219,
0.1392,0.1564,0.1736,0.1908,0.2079,0.2250,0.2419,0.2588,0.2756,0.2924,
0.3090,0.3256,0.3420,0.3584,0.3746,0.3907,0.4067,0.4226,0.4384,0.4540,
0.4695,0.4848,0.5000,0.5150,0.5299,0.5446,0.5592,0.5736,0.5878,0.6018,
0.6157,0.6293,0.6428,0.6561,0.6691,0.6820,0.6947,0.7071,0.7193,
0.7314,0.7431,0.7547,0.7660,0.7771,0.7880,0.7986,0.8090,0.8192,0.8290,
0.8387,0.8480,0.8572,0.8660,0.8746,0.8829,0.8910,0.8988,0.9063,0.9135,
0.9205,0.9272,0.9336,0.9397,0.9455,0.9511,0.9563,0.9613,0.9659,0.9703,
0.9744,0.9781,0.9816,0.9848,0.9877,0.9903,0.9925,0.9945,0.9962,0.9976,
0.9986,0.9994,0.9998,1.0000,0.9998,0.9994,0.9986,0.9976,0.9962,0.9945,
0.9925,0.9903,0.9877,0.9848,0.9816,0.9781,0.9744,0.9703,0.9659,0.9613,
0.9563,0.9511,0.9455,0.9397,0.9336,0.9272,0.9205,0.9135,0.9063,
0.8988,0.8910,0.8829,0.8746,0.8660,0.8572,0.8480,0.8387,0.8290,0.8192,
0.8090,0.7986,0.7880,0.7771,0.7660,0.7547,0.7431,0.7314,0.7193,0.7071,
0.6947,0.6820,0.6691,0.6561,0.6428,0.6293,0.6157,0.6018,0.5878,0.5736,
0.5592,0.5446,0.5299,0.5150,0.5000,0.4848,0.4695,0.4540,0.4384,0.4226,
0.4067,0.3907,0.3746,0.3584,0.3420,0.3256,0.3090,0.2924,0.2756,0.2588,
0.2419,0.2250,0.2079,0.1908,0.1736,0.1564,0.1392,0.1219,0.1045,0.0872,
0.0698,0.0523,0.0349,0.0175,0.0000,-0.0175,-0.0349,-0.0523,
-0.0698,-0.0872,-0.1045,-0.1219,-0.1392,-0.1564,-0.1736,-0.1908,-0.2079,
-0.2250,-0.2419,-0.2588,-0.2756,-0.2924,-0.3090,-0.3256,-0.3420,-0.3584,
-0.3746,-0.3907,-0.4067,-0.4226,-0.4384,-0.4540,-0.4695,-0.4848,-0.5000,
-0.5150,-0.5299,-0.5446,-0.5592,-0.5736,-0.5878,-0.6018,-0.6157,-0.6293,
-0.6428,-0.6561,-0.6691,-0.6820,-0.6947,-0.7071,-0.7193,-0.7314,-0.7431,
-0.7547,-0.7660,-0.7771,-0.7880,-0.7986,-0.8090,-0.8192,-0.8290,-0.8387,
```

```
−0.8480,−0.8572,−0.8660,−0.8746,−0.8829,−0.8910,−0.8988,−0.9063,−0.9135
,−0.9205,−0.9272,−0.9336,−0.9397,−0.9455,−0.9511,−0.9563,−0.9613,−0.9659,
−0.9703,−0.9744,−0.9781,−0.9816,−0.9848,−0.9877,−0.9903,−0.9925,−0.9945,
−0.9962,−0.9976,−0.9986,−0.9994,−0.9998,−1.0000,−0.9998,−0.9994,−0.9986,
−0.9976,−0.9962,−0.9945,−0.9925,−0.9903,−0.9877,−0.9848,−0.9816,−0.9781,
−0.9744,−0.9703,−0.9659,−0.9613,−0.9563,−0.9511,−0.9455,−0.9397,−0.9336,
−0.9272,−0.9205,−0.9135,−0.9063,−0.8988,−0.8910,−0.8829,−0.8746,−0.8660,
−0.8572,−0.8480,−0.8387,−0.8290,−0.8192,−0.8090,−0.7986,−0.7880,−0.7771,
−0.7660,−0.7547,−0.7431,−0.7314,−0.7193,−0.7071,−0.6947,−0.6820,−0.6691,
−0.6561,−0.6428,−0.6293,−0.6157,−0.6018,−0.5878,−0.5736,−0.5592,−0.5446,
−0.5299,−0.5150,−0.5000,−0.4848,−0.4695,−0.4540,−0.4384,−0.4226,−0.4067,
−0.3907,−0.3746,−0.3584,−0.3420,−0.3256,−0.3090,−0.2924,−0.2756,−0.2588,
−0.2419,−0.2250,−0.2079,−0.1908,−0.1736,−0.1564,−0.1392,−0.1219,−0.1045
,−0.0872,−0.0698,−0.0523,−0.0349,−0.0175};


float PI=3.14159265359;

int pos=0;
float soma=0, mediaM=0;
float P=150, I=1500, pi=0;
float Wref=0;
float integral=0;
float teta=0;
float T=0.000083333;
float Vinterno=0;

int TETA=0;
int TETA90=0;



//================ PI CORRENTE + RESSONANTE ================

float Iref=0.0, kp = 0.1,Pi = 0.0, ki =20.0, erro = 0.0,erro1=0.0;
float  B = 5.0, w0 = 142122.289,kres = 30.0;
float yres = 0.0, y2=0.0, y01=0.0;
float prop=0, intg=0;
int tempo=0;
float d = 1+Ts*B+w0*Ts*Ts;
float x = kres*Ts/d, y = kres*Ts/d, w = 1/d, z = (−2−Ts*B)/d;



Vrede= in[0];      //
Irede = in[1];       //Entrada de variaveis
Iref = in[2];      //
```

```
Vinterno=SENO[TETA];
Vrede=Vrede*0.005;        // divide por 200 para normalizar

//media movel

soma= soma - media[pos];  //
soma = soma + (Vinterno*Vrede); //
media[pos]=(Vinterno*Vrede);  //

 pos = pos +1;        // Algoritmo PLL
if(pos==200) { pos=0;}     //
mediaM= soma*0.005;       // divide por 200


integral = integral -  mediaM*I*T;  //
pi = -P*mediaM + integral;     //
Wref = (377+pi);        //


    teta = teta + Wref*0.00477;    // 60*360/freq
    if(teta>=359.0){teta=teta-359.0;} //
    TETA = teta;  //


// add + 90 graus no teta
TETA90 = TETA+90.0;

if(TETA90>=360){TETA90=TETA90-360;} // Retorna angulo para 0   para
          //trabalhar com valores menores
erro = Iref*SENO[TETA90] - Irede;

prop = erro*kp;         //
intg = intg + erro*Ts;        //
if(intg>1.0){intg==1.0;}     //
if(intg<-1.0){intg=-1.0;}      //Pi de corrente
yres = erro*x - erro1*y - y2*w - z*y01; //+ ressonante
Pi = prop+intg+yres ;       //
if(Pi>1.0){Pi=1.0;}       //
if(Pi<-1.0){Pi=-1.0;}       //

y2 = y01;         //
y01=yres;         //
erro1=erro;          //
```

```
out[0]= Pi;          //Saida para modulacao pwm
```

Programa A.6 – Modelo do inversor monofásico discretizado, código do *Psim*

```
#include <Stdlib.h>
#include <String.h>
#include<Math.h>


//================== Modelo inversor ============================


float Rl=0.2,L=0.003,Irede=0.0,Vdc=200.0,Idc=0,D=0.0,Ts = 0.000001 ;
float DIrede=0.0,Vrede=0.0,Vbarramento=0; // Variaveis do inversor
int conecta=0;

D = in[0];  //Entradas de variaveis do sistema
Vrede= in[1];
conecta = in[2];  //Conecta, variavel A presente no texto do trabalho

if(conecta==0){   //Condicoes para conexao com a rede ou carga
                Vrede=0.0;
                    Rl=10.0;
                    }
                    else{
                   Vrede= in[1];
                    Rl=0.5;
                    }



Vbarramento = Vdc*(2*D-1);        //Tensao de saida do inversor
DIrede = (Vbarramento - Vrede - Irede*Rl)/L;  //Variacao de corrente
Irede = Irede + DIrede*Ts;        //Nova corrente

out[0]=Irede;           //saida lida pelo controlador
```

Programa A.7 – Controlador de corrente injetada, código implementado no ESP32 para simulação HIL

```
//
//  Codigo implementado para controle da maquina
//  TCC
//  Pedro Henrique Barcellos Linhares - 15.1.5907
//

const int pwmfixo = 16;            //Pino de PWM fixo
const int pwmchave = 5       //Pino de PWM de chaveamento, D
const int f = 12000;               //Frequencia do pwm
const int resolution = 10;     // Resolucao do PWM
const int pwmchannel = 0;    //Canal do PWM fixo
const int D = 1;                //Canal do PWM de chaveamento D
const int pwminterrupt = 4;        //Pino que recebe interrupcao
const int tempInterrupt = 22;      //Pino para medicao de tempo de
    interrupcao
int flag = 0;           // Flag de interrupcao

// Controlador

const int entcorr=36;          //Pino que recebe entrada de I vindo do dsp
const int entvm=39;
float count = 0.0;


//Vari veis  PLL

float media [200] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

float SENO[360]={0,0.0175,0.0349,0.0523,0.0698,0.0872,0.1045,0.1219,
0.1392,0.1564,0.1736,0.1908,0.2079,0.2250,0.2419,0.2588,0.2756,0.2924,
0.3090,0.3256,0.3420,0.3584,0.3746,0.3907,0.4067,0.4226,0.4384,0.4540,
0.4695,0.4848,0.5000,0.5150,0.5299,0.5446,0.5592,0.5736,0.5878,0.6018,
0.6157,0.6293,0.6428,0.6561,0.6691,0.6820,0.6947,0.7071,0.7193,0.7314,
0.7431,0.7547,0.7660,0.7771,0.7880,0.7986,0.8090,0.8192,0.8290,0.8387,
0.8480,0.8572,0.8660,0.8746,0.8829,0.8910,0.8988,0.9063,0.9135,0.9205,
0.9272,0.9336,0.9397,0.9455,0.9511,0.9563,0.9613,0.9659,0.9703,0.9744,
0.9781,0.9816,0.9848,0.9877,0.9903,0.9925,0.9945,0.9962,0.9976,0.9986,
0.9994,0.9998,1.0000,0.9998,0.9994,0.9986,0.9976,0.9962,0.9945,0.9925,
```

```
0.9903,0.9877,0.9848,0.9816,0.9781,0.9744,0.9703,0.9659,0.9613,0.9563,
0.9511,0.9455,0.9397,0.9336,0.9272,0.9205,0.9135,0.9063,0.8988,0.8910,
0.8829,0.8746,0.8660,0.8572,0.8480,0.8387,0.8290,0.8192,0.8090,0.7986,
0.7880,0.7771,0.7660,0.7547,0.7431,0.7314,0.7193,0.7071,0.6947,0.6820,
0.6691,0.6561,0.6428,0.6293,0.6157,0.6018,0.5878,0.5736,0.5592,0.5446,
0.5299,0.5150,0.5000,0.4848,0.4695,0.4540,0.4384,0.4226,0.4067,0.3907,
0.3746,0.3584,0.3420,0.3256,0.3090,0.2924,0.2756,0.2588,0.2419,0.2250,
0.2079,0.1908,0.1736,0.1564,0.1392,0.1219,0.1045,0.0872,0.0698,0.0523,
0.0349,0.0175,0.0000,-0.0175,-0.0349,-0.0523,-0.0698,-0.0872,-0.1045,
-0.1219,-0.1392,-0.1564,-0.1736,-0.1908,-0.2079,-0.2250,-0.2419,-0.2588,
-0.2756,-0.2924,-0.3090,-0.3256,-0.3420,-0.3584,-0.3746,-0.3907,
-0.4067,-0.4226,-0.4384,-0.4540,-0.4695,-0.4848,-0.5000,-0.5150,-0.5299,
-0.5446,-0.5592,-0.5736,-0.5878,-0.6018,-0.6157,-0.6293,-0.6428,-0.6561,
-0.6691,-0.6820,-0.6947,-0.7071,-0.7193,-0.7314,-0.7431,-0.7547,-0.7660,
-0.7771,-0.7880,-0.7986,-0.8090,-0.8192,-0.8290,-0.8387,-0.8480,-0.8572,
-0.8660,-0.8746,-0.8829,-0.8910,-0.8988,-0.9063,-0.9135,-0.9205,-0.9272,
-0.9336,-0.9397,-0.9455,-0.9511,-0.9563,-0.9613,-0.9659,-0.9703,-0.9744,
-0.9781,-0.9816,-0.9848,-0.9877,-0.9903,-0.9925,-0.9945,-0.9962,-0.9976,
-0.9986,-0.9994,-0.9998,-1.0000,-0.9998,-0.9994,-0.9986,-0.9976,-0.9962,
-0.9945,-0.9925,-0.9903,-0.9877,-0.9848,-0.9816,-0.9781,-0.9744,
-0.9703,-0.9659,-0.9613,-0.9563,-0.9511,-0.9455,-0.9397,-0.9336,-0.9272,
-0.9205,-0.9135,-0.9063,-0.8988,-0.8910,-0.8829,-0.8746,-0.8660,-0.8572,
-0.8480,-0.8387,-0.8290,-0.8192,-0.8090,-0.7986,-0.7880,-0.7771,-0.7660,
-0.7547,-0.7431,-0.7314,-0.7193,-0.7071,-0.6947,-0.6820,-0.6691,-0.6561,
-0.6428,-0.6293,-0.6157,-0.6018,-0.5878,-0.5736,-0.5592,-0.5446,-0.5299,
-0.5150,-0.5000,-0.4848,-0.4695,-0.4540,-0.4384,-0.4226,-0.4067,-0.3907,
-0.3746,-0.3584,-0.3420,-0.3256,-0.3090,-0.2924,-0.2756,-0.2588,-0.2419,
-0.2250,-0.2079,-0.1908,-0.1736,-0.1564,-0.1392,-0.1219,-0.1045,-0.0872,
-0.0698,-0.0523,-0.0349,-0.0175};


#define PI=3.14159265359;

int pos=0;
float soma=0, mediaM=0;
float P=150, I=1500, pi=0;
float Wref=0;
float integral=0;
float teta=0;
float T=0.000083333;
float Vinterno=0;

int TETA=0;
int TETA90=0;
float fq=0.0;
```

```
//========================= PI CORRENTE + RESSONANTE
    ==========================

float Ts = 0.000083333,Vrede=0.0,D=0.0,Irede=0.0;
//float Iref=8.0, kp = 0.01,Pi = 0.0, ki =10.0, erro = 0.0,erro1=0.0;
float Iref=8.0, kp = 0.1,Pi = 0.0, ki =20.0, erro = 0.0,erro1=0.0;
float B = 5.0, w0 = 142122.289,kres = 30.0;
float yres = 0.0,y2=0.0;
float y01=0.0;
float prop=0,intg=0;
float d = 1/(1+Ts*B+w0*Ts*Ts);
float x = kres*Ts*d, y = kres*Ts*d, w = d, z = (-2-Ts*B)*d;
int dutyCycle=1;

void setup() {
  // put your setup code here, to run once:
  analogReadResolution(12);
     pinMode(pwm_fixo, OUTPUT);
                                                          //Saida PWM
pinMode(pwminterrupt, INPUT);//Pino que reconhece a interrup   o
pinMode(tempInterrupt, OUTPUT);//Pino que verifica o tempo de execu   o da
    interrupcao


     attachInterrupt(digitalPinToInterrupt(pwminterrupt),ISR,RISING);
              //Interrup   o na borda de subida do pwm
     ledcSetup(pwmchannel,f,resolution);
     ledcAttachPin(pwmfixo,pwmchannel);
     ledcWrite(pwmchannel,512);          //Inicializa o PWM fixo com o duty
         declarado


     ledcSetup(D,f,resolution);  // pwm do controle de 10 bits
     ledcAttachPin(pwmchave,D);
}

void loop() {

if (flag == 1.0)
{
digitalWrite(tempInterrupt, HIGH);     //Pino de verificacao de tempo de
    interrupt
```

```
Irede = (analogRead(entcorr)-1550.0)*0.0073;  //Entrada de dados vinda do
    DSP
Vrede = (analogRead(entvm)-1550.0)*0.000322;  //Entrada de dados vinda do
    DSP

Vinterno=SENO[TETA]; //Inicio do algoritmo do PLL
Vrede=Vrede*0.005;  // divide por 200 para normalizar

//media movel

soma= soma - media[pos];
soma = soma + (Vinterno*Vrede);
media[pos]=(Vinterno*Vrede);

 pos = pos +1;
if(pos==200) { pos=0;}
mediaM= soma*0.005;  // divide por 200


integral = integral -  mediaM*I*T;
pi = -P*mediaM + integral;
Wref = (377+pi);


                                    teta = teta + Wref*0.00477;    // 60*360/
                                        freq
                                    if(teta>=359.0){teta=teta-359.0;}
                                    TETA = teta;


// add + 90 graus no teta
TETA90 = TETA+90;
if(TETA90>=360){TETA90=TETA90-360;} //Fim do PLL



    erro = Iref*SENO[TETA90] - Irede;  //Inicio do PI de corrente

     prop = erro*kp;
     intg = intg + erro*0.0001*ki;
     if(intg>1.0){intg==1.0;}
     if(intg<-1.0){intg=-1.0;}

     yres = erro*x - erro1*y - y2*w - z*y01;
     Pi = prop+intg+yres;
     if(Pi>1.0){Pi=1.0;}
     if(Pi<-1.0){Pi=-1.0;}
```

```
    y2 = y01;
    y01=yres;
    erro1=erro;

    dutyCycle = 512.0*(Pi+1.0);        //Atualiza o Duty Cycle, fim do pi de
        corrente
    flag = 0.0;                        //Reseta a flag
    digitalWrite(tempInterrupt, LOW);
    ledcWrite(D,dutyCycle);            //Atualiza o duty que vai ao DSP
  }
  }

void ISR()     //Funcao da interrupcao
{
 flag = 1.0;
}
```

Programa A.8 – Modelo do inversor monofásico discretizado, código implementado no DSP para simulação HIL

```
// Marreta DSP
// C digo gera uma interrup   o por ciclo de chaveamento com ADC
    sincronizado.

#include "F28x_Project.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


//
// Globals
//


volatile struct DAC_REGS* DAC_PTR[4] = {0x0,&DacaRegs,&DacbRegs,&DaccRegs};
      // necess rio para o DAC funcionar, Mist rio da f



//Defines para frequencia de chaveamento do PWM
#define EPWM2_TIMER_TBPRD  220  // 400khz  200khz/fsw
#define EPWM2_MAX_CMPA     100
#define EPWM2_MIN_CMPB      50



//Defines para frequencia de chaveamento do PWM
#define EPWM3_TIMER_TBPRD  220  //      200Mhz/(Freq);
#define EPWM3_MAX_CMPA     100
#define EPWM3_MIN_CMPB      50


//Defines para frequencia de chaveamento do PWM
#define EPWM4_TIMER_TBPRD  220      //  200Mhz/(Freq);
#define EPWM4_MAX_CMPA     100
#define EPWM4_MIN_CMPB      50



//Prototipo das funcoes
void InitEPwm2Example(void);      //Configura o PWM e inicializa
void InitEPwm3Example(void);      //Configura o PWM e inicializa
void InitEPwm4Example(void);      //Configura o PWM e inicializa
void ConfigureADC(void);          //Configura e inicializa o ADC
```

```
void SetupADCSoftware(void);      //Configura os canais de aquisicao do ADC

interrupt void adca1_isr(void); //interrupcao do ADC

void configureDAC(Uint16 dac_num);


float Vdc=200.0, iL=0.0, diL=0.0;
float RL=0.2, L=333.3333, Vrede=0.0;
float teta=0.0, D=0.0;
int TETA=0;
float T=0.0000011;        //Perguntar Renan 1/0.0000011 = 909090,9 Hz
int conecta=0;
int cont=0;
float SENO[360];

int il_dac=0;



void main(void){
    //Inicializa o sistema (watchdogtimer, pll, sysclock)
    InitSysCtrl();

    //Habilita as GPIO
    InitGpio();

    //Habilita o PWM2
    CpuSysRegs.PCLKCR2.bit.EPWM2=1;
    CpuSysRegs.PCLKCR2.bit.EPWM3=1;
    CpuSysRegs.PCLKCR2.bit.EPWM4=1;


    //Habilita o pino do PWM2
    InitEPwm2Gpio();
    InitEPwm3Gpio();
    InitEPwm4Gpio();

    //Limpa todas as interrupcoes e inicializa a tabela de vetores de
        interrupcao
    DINT;

    //Inicializa em seu estado padrao os registradores de controle de
        interrupcoes PIE
    InitPieCtrl();
```

```
//Desabilita interrupcoes da CPU e limpa todas as flags de interrupcao
    da CPU
IER = 0x0000;
IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
InitPieVectTable();

//Interrupcoes
EALLOW; //Habilita escrita em registradores protegidos
PieVectTable.ADCA1_INT = &adca1_isr; //funcao para interrupcao do ADCA
EDIS;    //Desabilita escrita em registradores protegidos

// Inicializa o PWM2 e sincroniza
EALLOW;
CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;
EDIS;

InitEPwm4Example();
InitEPwm2Example();
InitEPwm3Example();


EALLOW;
CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
ClkCfgRegs.PERCLKDIVSEL.bit.EPWMCLKDIV=0; /// seta div do pwm
EDIS;

//Habilita interrupcoes globais
IER |= M_INT1; //Habilita grupo 1 das interrupcoes
EINT;  // Enable Global interrupt INTM
ERTM;  // Enable Global realtime interrupt DBGM

PieCtrlRegs.PIEIER1.bit.INTx1 = 1;


EALLOW;
      GpioCtrlRegs.GPBPUD.bit.GPIO40= 0; //Enable pullup on GPI32; porta
          36 habilitada
      GpioCtrlRegs.GPBMUX1.bit.GPIO40= 0; //GPI42= porta IO
      GpioCtrlRegs.GPBDIR.bit.GPIO40= 0; //GPI42= input

      GpioDataRegs.GPBCLEAR.bit.GPIO40=1;
```

```
GpioCtrlRegs.GPBPUD.bit.GPIO41= 0; //Enable pullup on GPI32; porta
    36 habilitada
GpioCtrlRegs.GPBMUX1.bit.GPIO41= 0; //GPI43= porta IO
GpioCtrlRegs.GPBDIR.bit.GPIO41= 0; //GPI43= input

GpioDataRegs.GPBCLEAR.bit.GPIO41=1;


GpioCtrlRegs.GPBPUD.bit.GPIO52= 0;
GpioCtrlRegs.GPBMUX2.bit.GPIO52= 0;
GpioCtrlRegs.GPBDIR.bit.GPIO52= 0; // GPIO as input

GpioDataRegs.GPBCLEAR.bit.GPIO52=1;


GpioCtrlRegs.GPCPUD.bit.GPIO65= 0;
GpioCtrlRegs.GPCMUX1.bit.GPIO65= 0;
GpioCtrlRegs.GPCDIR.bit.GPIO65= 1;    // GPIO as output

GpioDataRegs.GPCCLEAR.bit.GPIO65=1;


GpioCtrlRegs.GPBPUD.bit.GPIO32= 0;
GpioCtrlRegs.GPBMUX1.bit.GPIO32= 0;
GpioCtrlRegs.GPBDIR.bit.GPIO32= 0;

GpioDataRegs.GPBCLEAR.bit.GPIO32=1;


GpioCtrlRegs.GPAPUD.bit.GPIO19= 0;
GpioCtrlRegs.GPAMUX2.bit.GPIO19= 0;
GpioCtrlRegs.GPADIR.bit.GPIO19= 0; //= input

GpioDataRegs.GPACLEAR.bit.GPIO19=1;


GpioCtrlRegs.GPAPUD.bit.GPIO18= 0;
GpioCtrlRegs.GPAMUX2.bit.GPIO18= 0;
GpioCtrlRegs.GPADIR.bit.GPIO18= 0; //= input

GpioDataRegs.GPACLEAR.bit.GPIO18=1;


GpioCtrlRegs.GPCPUD.bit.GPIO67= 0;
GpioCtrlRegs.GPCMUX1.bit.GPIO67= 0;
GpioCtrlRegs.GPCDIR.bit.GPIO67= 0;    // GPIO as output
```

```
        GpioDataRegs.GPCCLEAR.bit.GPIO67=1;


        GpioCtrlRegs.GPDPUD.bit.GPIO111= 0;
        GpioCtrlRegs.GPDMUX1.bit.GPIO111= 0;
        GpioCtrlRegs.GPDDIR.bit.GPIO111= 0;    // GPIO as output

       GpioDataRegs.GPDCLEAR.bit.GPIO111=1;


                                  EPwm2Regs.CMPA.bit.CMPA = 100;
                                  EPwm2Regs.CMPB.bit.CMPB =100;

                                  EPwm6Regs.CMPA.bit.CMPA = 12500;
                                  EPwm6Regs.CMPB.bit.CMPB =12500;


        EDIS;


        for(cont=0;cont<=359;cont++){
                    teta = 0.0175*cont;
                    SENO[cont]=sin(teta);
                   }

                  teta=0;




    // Configura o ADC e inicializa
    ConfigureADC();
    SetupADCSoftware();
    configureDAC(1);
    configureDAC(2);

    //      configureDAC(1); para DACa (Na porta ADCINA0)
    //      configureDAC(2); para DACb (Na porta ADCINA1)
    //      configureDAC(3); para DACc (Na porta ADCINB1)

    do{

    }while(1);
}

// InitEPwm2Example - Inicializa ePWM2 e configura
```

```
void InitEPwm2Example()
{
    // Setup TBCLK
    EPwm2Regs.TBPRD = EPWM2_TIMER_TBPRD;           // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm2Regs.TBPHS.bit.TBPHS = 0x0000;            // Phase is 0
    EPwm2Regs.TBCTR = 0x0000;                      // Clear counter


    // Setup counter mode
    EPwm2Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
        SYSCLKOUT
    EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    //EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;
    EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
    //  Valores de compara      o inicial
    EPwm2Regs.CMPA.bit.CMPA = EPWM2_MAX_CMPA;    // valor inicial do duty p/
        pwmA
    EPwm2Regs.CMPB.bit.CMPB = EPWM2_MIN_CMPB;     // valor inicial do duty
        p/ pwmB


      EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;           // Set PWM1A on Zero
      EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;         // Clear PWM1A on event
          A,
                                                   // up count

      EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;           // Set PWM1B on Zero
      EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;         // Clear PWM1B on event
          B,
                                                   // up count


    // Set actions
    //EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;           // Set PWM2A on event A,
        up count
    //EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;         // Clear PWM2A on event B,
```

```
        down count


    //EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm2Regs.AQCTLB.bit.CBD = AQ_SET;

    // Interrupt where we will change the Compare Values
    EPwm2Regs.ETSEL.bit.SOCAEN = 1;      //Habilita o pulso de EPWM2SOCA
    EPwm2Regs.ETSEL.bit.SOCASEL = 2;     // Este bit determina quando o
        pulso de convers o do ePWMA ser     gerado
    EPwm2Regs.ETPS.bit.SOCAPRD = 1;      //Determina quantos periodos devem
        ocorrer antes de gerar o pulso do EPWM2SOCA
}

void InitEPwm3Example()
{
    // Setup TBCLK
    EPwm3Regs.TBPRD = EPWM3_TIMER_TBPRD;           // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm3Regs.TBPHS.bit.TBPHS = 0x0000;            // Phase is 0
    EPwm3Regs.TBCTR = 0x0000;                      // Clear counter


    // Setup counter mode
    EPwm3Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
        SYSCLKOUT
    EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    //  Valores de compara      o inicial
    EPwm3Regs.CMPA.bit.CMPA = EPWM3_MAX_CMPA;    // valor inicial do duty p/
        pwmA
    EPwm3Regs.CMPB.bit.CMPB = EPWM3_MIN_CMPB;      // valor inicial do duty
        p/ pwmB


    EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;             // Set PWM1A on Zero
```

```
    EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;            // Clear PWM1A on event A
        ,
                                                    // up count

    EPwm3Regs.AQCTLB.bit.ZRO = AQ_SET;              // Set PWM1B on Zero
    EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;            // Clear PWM1B on event B
        ,
                                                    // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;            // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;          // Clear PWM2A on event B,
        down count

    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}


void InitEPwm4Example()
{
    // Setup TBCLK
    EPwm4Regs.TBPRD = EPWM4_TIMER_TBPRD;            // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm4Regs.TBPHS.bit.TBPHS = 0x0000;             // Phase is 0
    EPwm4Regs.TBCTR = 0x0000;                       // Clear counter


    // Setup counter mode
    EPwm4Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm4Regs.TBCTL.bit.PHSEN = TB_DISABLE;         // Disable phase loading
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;        // Clock ratio to
        SYSCLKOUT
    EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    //  Valores de compara      o inicial
```

```
    EPwm4Regs.CMPA.bit.CMPA = EPWM4_MAX_CMPA;    // valor inicial do duty p/
        pwmA
    EPwm4Regs.CMPB.bit.CMPB = EPWM4_MIN_CMPB;    // valor inicial do duty
        p/ pwmB


    EPwm4Regs.AQCTLA.bit.ZRO = AQ_SET;           // Set PWM1A on Zero
    EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR;         // Clear PWM1A on event A
        ,
                                                        // up count

    EPwm4Regs.AQCTLB.bit.ZRO = AQ_SET;           // Set PWM1B on Zero
    EPwm4Regs.AQCTLB.bit.CBU = AQ_CLEAR;         // Clear PWM1B on event B
        ,
                                                        // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;         // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;       // Clear PWM2A on event B,
        down count

    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}

void ConfigureADC(void)
{
    EALLOW;

    //write configurations
    AdcaRegs.ADCCTL2.bit.PRESCALE = 6;
    AdcbRegs.ADCCTL2.bit.PRESCALE = 6;
    //Seta o ADC que sera utilizado, a resolucaoo e o modo de operacao
    AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
    AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);

    //Set pulse positions to late
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    //power up the ADC
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;

    //delay for 1ms to allow ADC time to power up
```

```
    DELAY_US(1000);

    EDIS;
}

void SetupADCSoftware(void)
{
    Uint16 acqps = 14;  //75ns aquisition time

    //Seleciona os canais para conversao, tempo de aquisicao, e trigger
        select (ePWM2 SOCA)
    //ADCA
    EALLOW;
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 0x00;  //SOC0 will convert pin A0
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = acqps; //sample window is acqps +1
        SYSCLK cycles
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 0x07; //trigger on ePWM1 SOCA/C
    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 0x02;  //SOC1 will convert pin A2
    AdcaRegs.ADCSOC1CTL.bit.ACQPS = acqps; //sample window is acqps +1
        SYSCLK cycles
    AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 0x07; //trigger on ePWM1 SOCA/C

    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //end of SOC1 will set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared


    //ADCB
    AdcbRegs.ADCSOC0CTL.bit.CHSEL = 0x00;  //SOC0 will convert pin B0
    AdcbRegs.ADCSOC0CTL.bit.ACQPS = acqps; //sample window is acqps +1
        SYSCLK cycles
    AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 0x07; //trigger on ePWM1 SOCA/C

    AdcbRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //end of SOC1 will set INT1 flag
    AdcbRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
    EDIS;
}

//Interrupcao do ADCA
interrupt void adca1_isr(void)
{

    GpioDataRegs.GPCSET.bit.GPIO65=1;  // j5-47


                    //  leitura=AdcaResultRegs.ADCRESULT0;
```

```
// Tcarga= leitura*0.005;
//leitura1=AdcaResultRegs.ADCRESULT1;
//leitura2=AdcaResultRegs.ADCRESULT2;
//leitura3=AdcaResultRegs.ADCRESULT3;


  D=GpioDataRegs.GPBDAT.bit.GPIO40;     //j5 50
  //DB=GpioDataRegs.GPBDAT.bit.GPIO41;
  //DC=GpioDataRegs.GPBDAT.bit.GPIO52;


  if(conecta==0){
  Vrede=0.0;
  RL=10.0;
  }
  else{
  Vrede=179.6*SENO[TETA];
  RL=0.5;
  }


  diL = (Vdc*(2.0*D-1.0) - RL*iL - Vrede)*L;
  iL = iL + diL*T;

  teta = teta + 0.02376;      // 60*360/500k
  if(teta>=359.0){teta=teta-359.0;}
  TETA= teta;

  il_dac= iL*133.333 + 2000.0;
  if(il_dac>4000){il_dac=4000;}
  if(il_dac<0){il_dac=0;}


  DAC_PTR[1]->DACVALS.all =il_dac;        // J3 - 30
  DAC_PTR[2]->DACVALS.all = 2000*SENO[TETA]+2000;   // J7
      - 70



AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

  //GpioDataRegs.GPBTOGGLE.bit.GPIO42=1;
GpioDataRegs.GPCCLEAR.bit.GPIO65=1;
```

```
}


//
// configureDAC - Enable and configure the requested DAC module
//
void configureDAC(Uint16 dac_num)
{
    EALLOW;

    DAC_PTR[dac_num]->DACCTL.bit.DACREFSEL = 1;   // 1 referencia de tensao
        interna, 0 referencia externa 3.3V no pino ADCINB0
    DAC_PTR[dac_num]->DACOUTEN.bit.DACOUTEN = 1; // habilita o DAC
    DAC_PTR[dac_num]->DACVALS.all = 0;            // inicia o DAC em zero

    DELAY_US(10); // Delay for buffered DAC to power up

    EDIS;
}
```

A.4 Códigos referentes às simulações HIL do conversor CC CC *boost*

Programa A.9 – Controlador de corrente, código implementado no ESP32 para simulação HIL

```
const int pwm = 16;                 //Pino de PWM
const int f = 10000;
const int resolution = 8;
const int pwmchannel = 0;
const int pwminterrupt = 35;    //Pino que recebe interrup   o
const int tempInterrupt = 22;
int flag = 0;

//Entradas
float Ts = 0.0001;
float Irede = 0.0;

// Controlador
float dutyCycle = 200.0;          // Valor arbitrario ao iniciar.
const int entcorr=34;            //Pino que recebe entrada de I vindo do dsp
float Iref = 10.0;                 //Referencia de corrente.
float count = 0.0;
float kp = 0.003;
float ki = 1.2;
float pi = 0.0;
float erro = 0.0;
float prop = 0.0;
float intg = 0.0;



void setup() {
  // put your setup code here, to run once:
    pinMode(pwm, OUTPUT);
                                                          //Saida PWM
    pinMode(pwminterrupt, INPUT);
                                                  //Pino que reconhece a
       interrup   o
    pinMode(tempInterrupt, OUTPUT);
                                                 //Pino que verifica o tempo
       de execucao da interrupcao
    ledcSetup(pwmchannel,f,resolution);
    ledcAttachPin(pwm, pwmchannel);
    attachInterrupt(digitalPinToInterrupt(pwminterrupt),ISR,RISING);
            //Interrupcao na borda de subida do pwm
```

```
      ledcWrite(pwmchannel,dutyCycle);
                                                    //Inicializa o PWM com o duty
          declarado
}

void loop() {

  if (flag == 1.0)
  {
    digitalWrite(tempInterrupt, HIGH);
    count = count+1;
    Irede = analogRead(entcorr);
    erro = Iref - Irede*0.0075;
    if (count == 15000)
      {
        Iref = 10.0;
      }
    if (count == 30000)
      {
        Iref = 15.0;
        count = 0;
      }
    prop = kp*erro;
    intg = intg + erro*ki*Ts;
    pi = prop + intg;
    if (pi<0)
      pi=0;
    if (pi>1)
      pi=1;
    pi = 255.0*(pi);
    dutyCycle = pi;                          //Atualiza o Duty Cycle
    flag = 0.0;                              //Reseta a flag
    digitalWrite(tempInterrupt, LOW);
    ledcWrite(pwmchannel,dutyCycle);         //Atualiza o duty
  }
  }

void ISR()     //Fun   o da interrupcao
{
 flag = 1.0;
}
```

Programa A.10 – Modelo conversor *boost* discretizado, código implementado no DSP para simulação HIL

```
#include "F28x_Project.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


//
// Globals
//


volatile struct DAC_REGS* DAC_PTR[4] = {0x0,&DacaRegs,&DacbRegs,&DaccRegs};









#define EPWM1_TIMER_TBPRD  200 // 400khz  200khz/fsw
#define EPWM1_MAX_CMPA     100
#define EPWM1_MIN_CMPB      50


//Defines para frequencia de chaveamento do PWM
#define EPWM2_TIMER_TBPRD  200 // 333khz            200khz/fsw
#define EPWM2_MAX_CMPA     100
#define EPWM2_MIN_CMPB      50



//Defines para frequencia de chaveamento do PWM
#define EPWM3_TIMER_TBPRD  400  //      200Mhz/(Freq);
#define EPWM3_MAX_CMPA     250
#define EPWM3_MIN_CMPB      50


//Defines para frequencia de chaveamento do PWM
#define EPWM4_TIMER_TBPRD  4000     //  200Mhz/(Freq);
#define EPWM4_MAX_CMPA     2000
#define EPWM4_MIN_CMPB     2000

//Defines para frequencia de chaveamento do PWM
#define EPWM5_TIMER_TBPRD  400      //  200Mhz/(Freq);
```

```
#define EPWM5_MAX_CMPA      250
#define EPWM5_MIN_CMPB      50

//Defines para frequencia de chaveamento do PWM
#define EPWM6_TIMER_TBPRD  20000      //  200Mhz/(Freq);
#define EPWM6_MAX_CMPA      10000
#define EPWM6_MIN_CMPB      10000



//Prototipo das funcoes
void InitEPwm6Example(void);     //Configura o PWM e inicializa
void InitEPwm5Example(void);     //Configura o PWM e inicializa
void InitEPwm2Example(void);     //Configura o PWM e inicializa
void InitEPwm3Example(void);     //Configura o PWM e inicializa
void InitEPwm4Example(void);     //Configura o PWM e inicializa
void InitEPwm1Example(void);     //Configura o PWM e inicializa

void ConfigureADC(void);         //Configura e inicializa o ADC
void SetupADCSoftware(void);     //Configura os canais de aquisicao do ADC

interrupt void adca1_isr(void); //interrupcao do ADC

void configureDAC(Uint16 dac_num);

// modelo para BODE il=0.0, Vc=0.0, R=0.1, RL=0.1, L=200.0, C=2500.0, Vin
    =100;

float D=0.0;
float  i=0, di=0, Vb=20.0;
float L=0.0003, RL=0.1;
float C=0.0004, R=10.0;
float Vc=0.0,dVc=0.0;

float LL=0.0, CC=0.0, RR=0.0;
float T=0.000001;



void main(void){
    //Inicializa o sistema (watchdogtimer, pll, sysclock)
    InitSysCtrl();

    //Habilita as GPIO
    InitGpio();
```

```
CpuSysRegs.PCLKCR2.bit.EPWM1=1;
CpuSysRegs.PCLKCR2.bit.EPWM2=1;
CpuSysRegs.PCLKCR2.bit.EPWM3=1;
CpuSysRegs.PCLKCR2.bit.EPWM4=1;
CpuSysRegs.PCLKCR2.bit.EPWM5=1;
CpuSysRegs.PCLKCR2.bit.EPWM6=1;


//Habilita o pino do PWM2
InitEPwm1Gpio();
InitEPwm2Gpio();
InitEPwm3Gpio();
InitEPwm4Gpio();
InitEPwm5Gpio();
InitEPwm6Gpio();
//Limpa todas as interrupcoes e inicializa a tabela de vetores de
    interrupcao
DINT;

//Inicializa em seu estado padrao os registradores de controle de
    interrupcoes PIE
InitPieCtrl();




//Desabilita interrupcoes da CPU e limpa todas as flags de interrupcao
    da CPU
IER = 0x0000;
IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
InitPieVectTable();

//Interrupcoes
EALLOW; //Habilita escrita em registradores protegidos
PieVectTable.ADCA1_INT = &adca1_isr; //funcao para interrupcao do ADCA
EDIS;    //Desabilita escrita em registradores protegidos

// Inicializa o PWM2 e sincroniza
EALLOW;
CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;
EDIS;

InitEPwm6Example();
InitEPwm5Example();
```

```
InitEPwm4Example();
InitEPwm3Example();
InitEPwm2Example();
InitEPwm1Example();


EALLOW;
CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
ClkCfgRegs.PERCLKDIVSEL.bit.EPWMCLKDIV=0; /// seta div do pwm
EDIS;

//Habilita interrupcoes globais
IER |= M_INT1; //Habilita grupo 1 das interrupcoes
EINT;  // Enable Global interrupt INTM
ERTM;  // Enable Global realtime interrupt DBGM

PieCtrlRegs.PIEIER1.bit.INTx1 = 1;


EALLOW;
    GpioCtrlRegs.GPBPUD.bit.GPIO40= 0; //Enable pullup on GPI32; porta
        36 habilitada
    GpioCtrlRegs.GPBMUX1.bit.GPIO40= 0; //GPI42= porta IO
    GpioCtrlRegs.GPBDIR.bit.GPIO40= 0; //GPI42= input

    GpioDataRegs.GPBCLEAR.bit.GPIO40=1;


    GpioCtrlRegs.GPBPUD.bit.GPIO41= 0; //Enable pullup on GPI32; porta
        36 habilitada
    GpioCtrlRegs.GPBMUX1.bit.GPIO41= 0; //GPI43= porta IO
    GpioCtrlRegs.GPBDIR.bit.GPIO41= 0; //GPI43= input

    GpioDataRegs.GPBCLEAR.bit.GPIO41=1;


    GpioCtrlRegs.GPBPUD.bit.GPIO52= 0;
    GpioCtrlRegs.GPBMUX2.bit.GPIO52= 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO52= 0; // GPIO as input

    GpioDataRegs.GPBCLEAR.bit.GPIO52=1;


    GpioCtrlRegs.GPCPUD.bit.GPIO65= 0;
    GpioCtrlRegs.GPCMUX1.bit.GPIO65= 0;
    GpioCtrlRegs.GPCDIR.bit.GPIO65= 0;    // GPIO as intput
```

```
GpioDataRegs.GPCCLEAR.bit.GPIO65=1;



GpioCtrlRegs.GPCPUD.bit.GPIO94= 0;
GpioCtrlRegs.GPCMUX2.bit.GPIO94= 0;
GpioCtrlRegs.GPCDIR.bit.GPIO94= 1;    // GPIO as output

GpioDataRegs.GPCCLEAR.bit.GPIO94=1;



GpioCtrlRegs.GPBPUD.bit.GPIO32= 0;
GpioCtrlRegs.GPBMUX1.bit.GPIO32= 0;
GpioCtrlRegs.GPBDIR.bit.GPIO32= 0;

GpioDataRegs.GPBCLEAR.bit.GPIO32=1;


GpioCtrlRegs.GPAPUD.bit.GPIO19= 0;
GpioCtrlRegs.GPAMUX2.bit.GPIO19= 0;
GpioCtrlRegs.GPADIR.bit.GPIO19= 0; //= input

GpioDataRegs.GPACLEAR.bit.GPIO19=1;


GpioCtrlRegs.GPAPUD.bit.GPIO18= 0;
GpioCtrlRegs.GPAMUX2.bit.GPIO18= 0;
GpioCtrlRegs.GPADIR.bit.GPIO18= 0; //= input

GpioDataRegs.GPACLEAR.bit.GPIO18=1;


GpioCtrlRegs.GPCPUD.bit.GPIO67= 0;
GpioCtrlRegs.GPCMUX1.bit.GPIO67= 0;
GpioCtrlRegs.GPCDIR.bit.GPIO67= 0;    // GPIO as output

GpioDataRegs.GPCCLEAR.bit.GPIO67=1;


GpioCtrlRegs.GPDPUD.bit.GPIO111= 0;
GpioCtrlRegs.GPDMUX1.bit.GPIO111= 0;
GpioCtrlRegs.GPDDIR.bit.GPIO111= 0;    // GPIO as output

GpioDataRegs.GPDCLEAR.bit.GPIO111=1;
```

```
                                        /*   EPwm2Regs.CMPA.bit.CMPA = 100;
                                          EPwm2Regs.CMPB.bit.CMPB =100;

                                          EPwm6Regs.CMPA.bit.CMPA = 12500;
                                          EPwm6Regs.CMPB.bit.CMPB =12500;*/


        EDIS;




    // Configura o ADC e inicializa
    ConfigureADC();
    SetupADCSoftware();
    configureDAC(1);
    configureDAC(2);

    LL=1/L;
    CC=1/C;
    RR=1/R;

    //      configureDAC(1); para DACa (Na porta ADCINA0)
    //      configureDAC(2); para DACb (Na porta ADCINA1)
    //      configureDAC(3); para DACc (Na porta ADCINB1)

    do{

    }while(1);
}


void InitEPwm1Example()
{
    // Setup TBCLK
    EPwm1Regs.TBPRD = EPWM1_TIMER_TBPRD;          // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm1Regs.TBPHS.bit.TBPHS = 0x0000;           // Phase is 0
    EPwm1Regs.TBCTR = 0x0000;                     // Clear counter


    // Setup counter mode
    EPwm1Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
```

```
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
        SYSCLKOUT
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    //  Valores de compara      o inicial
    EPwm1Regs.CMPA.bit.CMPA = EPWM1_MAX_CMPA;    // valor inicial do duty p/
        pwmA
    EPwm1Regs.CMPB.bit.CMPB = EPWM1_MIN_CMPB;      // valor inicial do duty
        p/ pwmB


    EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;             // Set PWM1A on Zero
    EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // Clear PWM1A on event A
        ,
                                                          // up count

    EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;             // Set PWM1B on Zero
    EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;           // Clear PWM1B on event B
        ,
                                                          // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;          // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;        // Clear PWM2A on event B,
        down count

    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}



// InitEPwm2Example - Inicializa ePWM2 e configura
void InitEPwm2Example()
{
```

```
// Setup TBCLK
EPwm2Regs.TBPRD = EPWM2_TIMER_TBPRD;           // para 12 kHz, TBPRD =
    1/[freq*(4,00019e-8) ]
EPwm2Regs.TBPHS.bit.TBPHS = 0x0000;            // Phase is 0
EPwm2Regs.TBCTR = 0x0000;                      // Clear counter


// Setup counter mode
EPwm2Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
    SYSCLKOUT
EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;


// Setup shadowing
EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

//EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;
EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
//  Valores de compara      o inicial
EPwm2Regs.CMPA.bit.CMPA = EPWM2_MAX_CMPA;    // valor inicial do duty p/
    pwmA
EPwm2Regs.CMPB.bit.CMPB = EPWM2_MIN_CMPB;     // valor inicial do duty
    p/ pwmB


  EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;           // Set PWM1A on Zero
  EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;         // Clear PWM1A on event
      A,
                                               // up count

  EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;           // Set PWM1B on Zero
  EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;         // Clear PWM1B on event
      B,
                                               // up count


// Set actions
//EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;           // Set PWM2A on event A,
    up count
//EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;         // Clear PWM2A on event B,
     down count
```

```
    //EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm2Regs.AQCTLB.bit.CBD = AQ_SET;

    // Interrupt where we will change the Compare Values
    EPwm2Regs.ETSEL.bit.SOCAEN = 1;      //Habilita o pulso de EPWM2SOCA
    EPwm2Regs.ETSEL.bit.SOCASEL = 2;     // Este bit determina quando o
        pulso de convers o do ePWMA ser     gerado
    EPwm2Regs.ETPS.bit.SOCAPRD = 1;      //Determina quantos periodos devem
        ocorrer antes de gerar o pulso do EPWM2SOCA
}

void InitEPwm3Example()
{
    // Setup TBCLK
    EPwm3Regs.TBPRD = EPWM3_TIMER_TBPRD;           // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm3Regs.TBPHS.bit.TBPHS = 0x0000;            // Phase is 0
    EPwm3Regs.TBCTR = 0x0000;                      // Clear counter


    // Setup counter mode
    EPwm3Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
        SYSCLKOUT
    EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    //  Valores de compara      o inicial
    EPwm3Regs.CMPA.bit.CMPA = EPWM3_MAX_CMPA;    // valor inicial do duty p/
        pwmA
    EPwm3Regs.CMPB.bit.CMPB = EPWM3_MIN_CMPB;      // valor inicial do duty
        p/ pwmB


    EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;             // Set PWM1A on Zero
    EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // Clear PWM1A on event A
        ,
```

```
                                                    // up count

    EPwm3Regs.AQCTLB.bit.ZRO = AQ_SET;              // Set PWM1B on Zero
    EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;            // Clear PWM1B on event B
        ,
                                                    // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;            // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;          // Clear PWM2A on event B,
        down count

    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}


void InitEPwm4Example()
{
    // Setup TBCLK
    EPwm4Regs.TBPRD = EPWM4_TIMER_TBPRD;            // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm4Regs.TBPHS.bit.TBPHS = 0x0000;             // Phase is 0
    EPwm4Regs.TBCTR = 0x0000;                       // Clear counter


    // Setup counter mode
    EPwm4Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm4Regs.TBCTL.bit.PHSEN = TB_DISABLE;         // Disable phase loading
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;        // Clock ratio to
        SYSCLKOUT
    EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    //  Valores de compara o inicial
    EPwm4Regs.CMPA.bit.CMPA = EPWM4_MAX_CMPA;   // valor inicial do duty p/
        pwmA
```

```
    EPwm4Regs.CMPB.bit.CMPB = EPWM4_MIN_CMPB;      // valor inicial do duty
        p/ pwmB


    EPwm4Regs.AQCTLA.bit.ZRO = AQ_SET;             // Set PWM1A on Zero
    EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // Clear PWM1A on event A
        ,
                                                       // up count

    EPwm4Regs.AQCTLB.bit.ZRO = AQ_SET;             // Set PWM1B on Zero
    EPwm4Regs.AQCTLB.bit.CBU = AQ_CLEAR;           // Clear PWM1B on event B
        ,
                                                       // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;           // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;         // Clear PWM2A on event B,
        down count

    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}
void InitEPwm5Example()
{
    // Setup TBCLK
    EPwm5Regs.TBPRD = EPWM5_TIMER_TBPRD;           // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm5Regs.TBPHS.bit.TBPHS = 0x0000;            // Phase is 0
    EPwm5Regs.TBCTR = 0x0000;                      // Clear counter


    // Setup counter mode
    EPwm5Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm5Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
    EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
        SYSCLKOUT
    EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    // Setup shadowing
    EPwm5Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm5Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm5Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm5Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
```

```
    EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    // Valores de compara      o inicial
    EPwm5Regs.CMPA.bit.CMPA = EPWM4_MAX_CMPA;    // valor inicial do duty p/
        pwmA
    EPwm5Regs.CMPB.bit.CMPB = EPWM4_MIN_CMPB;     // valor inicial do duty
        p/ pwmB


    EPwm5Regs.AQCTLA.bit.ZRO = AQ_SET;            // Set PWM1A on Zero
    EPwm5Regs.AQCTLA.bit.CAU = AQ_CLEAR;          // Clear PWM1A on event A
        ,
                                                        // up count

    EPwm5Regs.AQCTLB.bit.ZRO = AQ_SET;            // Set PWM1B on Zero
    EPwm5Regs.AQCTLB.bit.CBU = AQ_CLEAR;          // Clear PWM1B on event B
        ,
                                                        // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;          // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;        // Clear PWM2A on event B,
        down count

    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}


void InitEPwm6Example()
{
    // Setup TBCLK
    EPwm6Regs.TBPRD = EPWM6_TIMER_TBPRD;          // para 12 kHz, TBPRD =
        1/[freq*(4,00019e-8) ]
    EPwm6Regs.TBPHS.bit.TBPHS = 0x0000;           // Phase is 0
    EPwm6Regs.TBCTR = 0x0000;                     // Clear counter


    // Setup counter mode
    EPwm6Regs.TBCTL.bit.CTRMODE = 0x03; // counter mode: freeze
    EPwm6Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
    EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to
        SYSCLKOUT
    EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV1;
```

```
    // Setup shadowing
    EPwm6Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm6Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm6Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm6Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;

    //  Valores de compara      o inicial
    EPwm6Regs.CMPA.bit.CMPA = EPWM6_MAX_CMPA;    // valor inicial do duty p/
        pwmA
    EPwm6Regs.CMPB.bit.CMPB = EPWM6_MIN_CMPB;      // valor inicial do duty
        p/ pwmB


    EPwm6Regs.AQCTLA.bit.ZRO = AQ_SET;             // Set PWM1A on Zero
    EPwm6Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // Clear PWM1A on event A
        ,
                                                          // up count

    EPwm6Regs.AQCTLB.bit.ZRO = AQ_SET;             // Set PWM1B on Zero
    EPwm6Regs.AQCTLB.bit.CBU = AQ_CLEAR;           // Clear PWM1B on event B
        ,
                                                          // up count
    // Set actions
    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;          // Set PWM2A on event A,
        up count
   // EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;        // Clear PWM2A on event B,
        down count

    //EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;


}

void ConfigureADC(void)
{
    EALLOW;

    //write configurations
    AdcaRegs.ADCCTL2.bit.PRESCALE = 6;
    AdcbRegs.ADCCTL2.bit.PRESCALE = 6;
    //Seta o ADC que sera utilizado, a resolucaoo e o modo de operacao
    AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
    AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
```

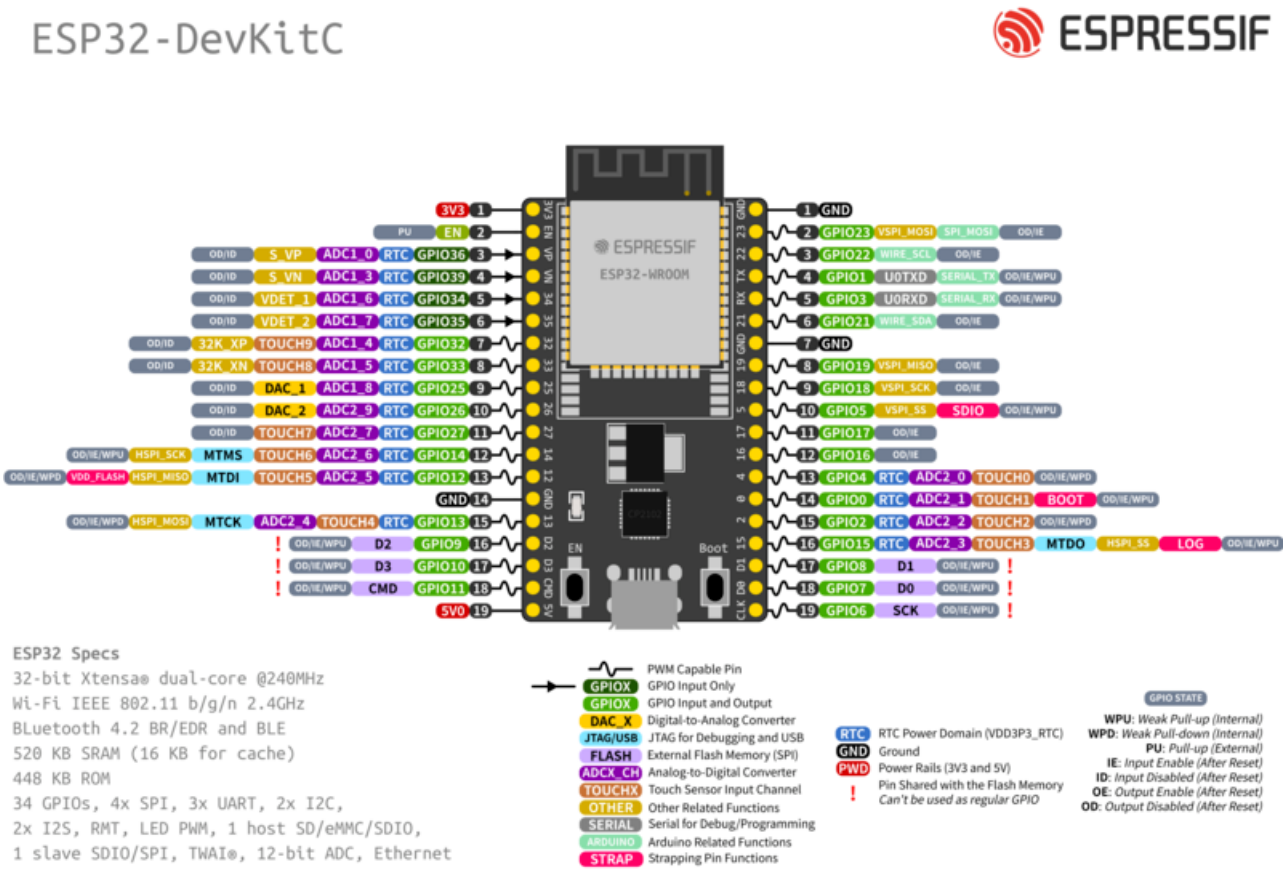
```
    //Set pulse positions to late
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    //power up the ADC
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;

    //delay for 1ms to allow ADC time to power up
    DELAY_US(1000);

    EDIS;
}

void SetupADCSoftware(void)
{
    Uint16 acqps = 14;  //75ns aquisition time

    //Seleciona os canais para conversao, tempo de aquisicao, e trigger
        select (ePWM2 SOCA)
    //ADCA
    EALLOW;
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 0x00;  //SOC0 will convert pin A0
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = acqps; //sample window is acqps +1
        SYSCLK cycles
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 0x07; //trigger on ePWM1 SOCA/C
    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 0x02;  //SOC1 will convert pin A2
    AdcaRegs.ADCSOC1CTL.bit.ACQPS = acqps; //sample window is acqps +1
        SYSCLK cycles
    AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 0x07; //trigger on ePWM1 SOCA/C

    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //end of SOC1 will set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared


    //ADCB
    AdcbRegs.ADCSOC0CTL.bit.CHSEL = 0x00;  //SOC0 will convert pin B0
    AdcbRegs.ADCSOC0CTL.bit.ACQPS = acqps; //sample window is acqps +1
        SYSCLK cycles
    AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 0x07; //trigger on ePWM1 SOCA/C

    AdcbRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //end of SOC1 will set INT1 flag
    AdcbRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
    EDIS;
```

```
}

//Interrupcao do ADCA
interrupt void adca1_isr(void)
{

    GpioDataRegs.GPCSET.bit.GPIO94=1;     //pino j5-46


    D=GpioDataRegs.GPBDAT.bit.GPIO40;      // pino j5-50
    //D = 0.68;
  //  D2=GpioDataRegs.GPBDAT.bit.GPIO41;
  //  D3=GpioDataRegs.GPBDAT.bit.GPIO52;
  //  D4=GpioDataRegs.GPCDAT.bit.GPIO65;

   //    botao1=GpioDataRegs.GPBDAT.bit.GPIO32;
   //    botao2=GpioDataRegs.GPADAT.bit.GPIO19;
   //    botao3=GpioDataRegs.GPADAT.bit.GPIO18;
   //    botao4=GpioDataRegs.GPCDAT.bit.GPIO67;

     //chave=GpioDataRegs.GPDDAT.bit.GPIO111;



    di = (Vb - RL*i - Vc*(1-D))*LL;
    dVc = (i*(1-D) - Vc*RR)*CC;

    i = i + di*T;
    if(i<=0.0){i=0.0;}
    Vc = Vc + dVc*T;


     DAC_PTR[1]->DACVALS.all = i*133.33;   // 30A = 3V       pino j3-30
         *133.33
     DAC_PTR[2]->DACVALS.all = Vc*40.0;    //   100V = 3V     pino j7-70
         *40.0

     //EPwm1Regs.CMPA.bit.CMPA = 200.0;
     //EPwm1Regs.CMPB.bit.CMPB = 200.0;

     //EPwm2Regs.CMPA.bit.CMPA = 20.0;
     //EPwm2Regs.CMPB.bit.CMPB = 20.0;

     //EPwm3Regs.CMPA.bit.CMPA = 20.0;
     //EPwm3Regs.CMPB.bit.CMPB = 20.0;

      //EPwm5Regs.CMPA.bit.CMPA = 20.0;
```

```
        //EPwm5Regs.CMPB.bit.CMPB = 20.0;

        //EPwm6Regs.CMPA.bit.CMPA = 400;
        //EPwm6Regs.CMPB.bit.CMPB = 400;




    // modelo em sequ ncia negativa de acionamento A=0 B =+120 C=-120



    GpioDataRegs.GPCCLEAR.bit.GPIO94=1;



    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

        //GpioDataRegs.GPBTOGGLE.bit.GPIO42=1;



}


//
// configureDAC - Enable and configure the requested DAC module
//
void configureDAC(Uint16 dac_num)
{
    EALLOW;

    DAC_PTR[dac_num]->DACCTL.bit.DACREFSEL = 1;  // 1 referencia de tensao
        interna, 0 referencia externa 3.3V no pino ADCINB0
    DAC_PTR[dac_num]->DACOUTEN.bit.DACOUTEN = 1; // habilita o DAC
    DAC_PTR[dac_num]->DACVALS.all = 0;           // inicia o DAC em zero

    DELAY_US(10); // Delay for buffered DAC to power up

    EDIS;
}
```

A.5 *Layout* e Tabelas dos pinos dos *hardwares*

Figura 68 – *Layout* dos Pinos do ESP32



Fonte: Espressif (2021)

Tabela 6 – Pinos J1 e J3 do DSP

Mux Value				J1 Pin	J3 Pin	Mux Value			
X	2	1	0			0	Alt Function	2	X
			3.3V	1	21	5V			
			GPIO32	2	22	GND			
	SCIRXDB		GPIO19	3	23	ADCIN14	CMPIN4P		
	SCITXDB		GPIO18	4	24	ADCINC3	CMPIN6N		
			GPIO67	5	25	ADCINB3	CMPIN3N		
			GPIO111	6	26	ADCINA3	CMPIN1N		
SPICLKA(1)			GPIO60	7	27	ADCINC2	CMPIN6P		
			GPIO22	8	28	ADCINB2	CMPIN3P		
		SCLA	GPIO105(2)	9	29	ADCINA2	CMPIN1P		
		SDAA	GPIO104(2)	10	30	ADCINA0	DACOUTA		

Fonte: TI (2019)

Tabela 7 – Pinos J4 e J2 do DSP

Mux Value				J4 Pin	J2 Pin	Mux Value			
X	2	1	0			0	1	2	X
		EPWM1A	GPIO0	40	20	GND			
		EPWM1B	GPIO1	39	19	GPIO61			
		EPWM2A	GPIO2	38	18	GPIO123			SD1_C1(1)
		EPWM2B	GPIO3	37	17	GPIO122			SD1_D1(1)
		EPWM3A	GPIO4	36	16	RST			
		EPWM3B	GPIO5	35	15	GPIO58			SPISIMOA(1)
		OUTPUTXBAR1	GPIO24	34	14	GPIO59			SPISOMIA(1)
OUTPUTXBAR7(1)			GPIO16	33	13	GPIO124			SD1_D2(1)
			DAC1	32	12	GPIO125			SD1_C2(1)
			DAC2	31	11	GPIO29(2)			OUTPUTXBAR6(1)

Fonte: TI (2019)

Tabela 8 – Pinos J5 e J7 do DSP

Mux Value				J5 Pin	J7 Pin	Mux Value			
X	2	1	0			0	Alt Function	2	X
			3.3V	41	61	5V			
			GPIO95	42	62	GND			
SCIRXDC(1)			GPIO139	43	63	ADCIN15	CMPIN4N		
SCITXDC(1)			GPIO56	44	64	ADCINC5	CMPIN5N		
			GPIO97	45	65	ADCINB5			
			GPIO94	46	66	ADCINA5	CMPIN2N		
SPICLKB(1)			GPIO65	47	67	ADCINC4	CMPIN5P		
			GPIO52(2)	48	68	ADCINB4			
SCLB(1)			GPIO41(2)	49	69	ADCINA4	CMPIN2P		
SDAB(1)			GPIO40(2)	50	70	ADCINA1	DACOUTB		

Fonte: TI (2019)

Tabela 9 – **Pinos J8 e J6 do DSP**

Mux Value				J8 Pin	J6 Pin	Mux Value			
X	2	1	0			0	1	2	X
		EPWM4A	GPIO6	80	60	GND			
		EPWM4B	GPIO7	79	59	GPIO66			
		EPWM5A	GPIO8	78	58	GPIO131			SD2_C1(1)
		EPWM5B	GPIO9	77	57	GPIO130			SD2_D1(1)
		EPWM6A	GPIO10	76	56	RST			
		EPWM6B	GPIO11	75	55	GPIO63			SPISIMOB(1)
OUTPUTXBAR3(1)			GPIO14	74	54	GPIO64			SPISOMIB(1)
OUTPUTXBAR4(1)			GPIO15	73	53	GPIO26			SD2_D2(1)
			DAC3	72	52	GPIO27			SD2_C2(1)
			DAC4	71	51	GPIO25			OUTPUTXBAR2(1)

Fonte: TI (2019)