UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
FACULDADE DE ENGENHARIAS, ARQUITETURA E URBANISMO E GEOGRAFIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM SISTEMAS DE ENERGIA

Implementação Dos Algoritmos De Funcionamento E Controle De Um Inversor Bidirecional Conectado À Rede De Um Eletroposto Com Aplicação V2g Em Um Microcontrolador C2000 F2837xd

Artur Alves de Carvalho

Campo Grande - MS
17 de outubro de 2023

UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
FACULDADE DE ENGENHARIAS, ARQUITETURA E URBANISMO E GEOGRAFIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM SISTEMAS DE ENERGIA

Implementação Dos Algoritmos De Funcionamento E Controle De Um Inversor Bidirecional Conectado À Rede De Um Eletroposto Com Aplicação V2g Em Um Microcontrolador C2000 F2837xd

Artur Alves de Carvalho

Dissertação de mestrado apresentada como exigência para obtenção do título de Mestrado em Engenharia Elétrica na área de Sistemas de Energia da Universidade Federal de Mato Grosso do Sul – UFMS.

Orientador: Moacyr Aureliano Gomes de Brito

Campo Grande - MS
17 de outubro de 2023

Implementação Dos Algoritmos De Funcionamento E Controle De Um Inversor Bidirecional Conectado À Rede De Um Eletroposto Com Aplicação V2g Em Um Microcontrolador C2000 F2837xd

Dissertação de mestrado apresentada como exigência para obtenção do título de Mestrado em Engenharia Elétrica na área de Sistemas de Energia da Universidade Federal de Mato Grosso do Sul – UFMS.

Banca Examinadora:

_____

Prof. Dr. Moacyr Aureliano Gomes de Brito

Orientador

_____

Prof. Dr. Luigi Galotto Junior

_____

Prof. Dr. Luis de Oros Arenas

Campo Grande - MS

17 de outubro de 2023

RESUMO

Este trabalho é baseado no projeto de pesquisa e desenvolvimento de um Sistema Nacional de Recarga Rápido de Bicicletas e Veículos Elétricos para Aplicações V2G (Vehicle-to-Grid). Dentre os conversores que compõem o sistema do eletroposto, o inversor bidirecional é responsável por fazer o interfaceamento entre o barramento CC e a rede elétrica. Foram implementados os algoritmos de travamento de fase do tipo PLL (phase-locked-loop), das malhas de tensão de corrente, e da comunicação com os demais conversores e a nuvem, em um microprocessador da Texas Instruments, TMS320F28379D. O firmware desenvolvido propicia ao conversor boa capacidade de rastreamento das tensões de referência da rede elétrica, e possui desempenho satisfatório quando atua nas potências para qual foi projetado, injetando corrente com formas de onda que atende as normativas internacionais com relação à distorção da onda.

Palavras-Chave: Conversores de Potência, Firmware, Microprocessamento.

ABSTRACT

This work is based on the research and development project of a National System for Fast Recharging of Bicycles and Electric Vehicles for V2G Applications (Vehicle to Grid). Among the converters that make up the charging station system, the bidirectional inverter is responsible for interfacing between the DC bus and the electrical grid. The PLL-type (phase-locked-loop) phase-locking algorithms, the current-voltage loops, and the communication with the other converters and the cloud were implemented in a Texas Instruments microprocessor, TMS320F28379D. The developed firmware provides the converter with a good ability to track the reference voltages of the electrical network and has satisfactory performance when it operates at the powers for which it was designed, injecting currents with reduced harmonic content.

Keywords: Firmware, Microprocessing, Power Converters.

LISTA DE FIGURAS

Número *Página*

LISTA DE TABELAS

Número *Página*

SUMÁRIO

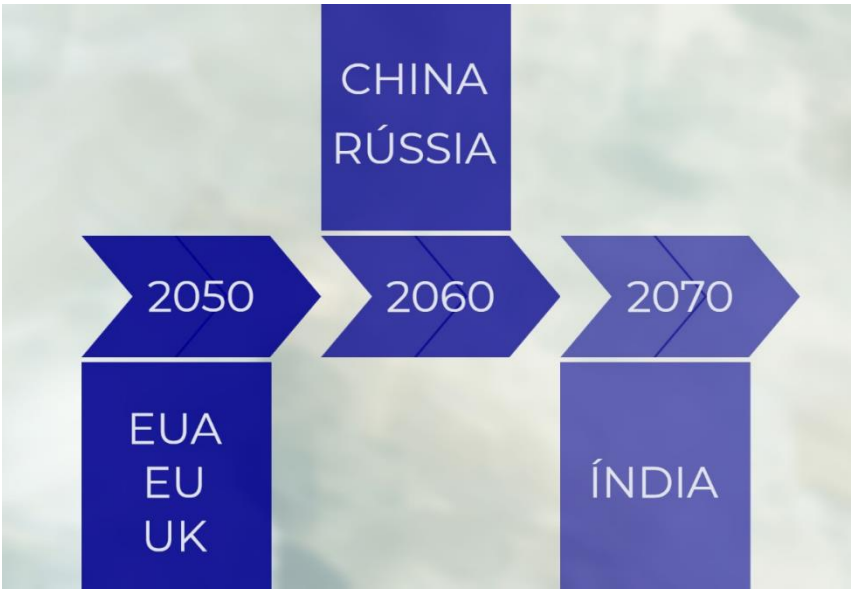
1. INTRODUÇÃO

Este capítulo tratará brevemente sobre os aspectos socioeconômicos que motivam a busca pelo desenvolvimento de tecnologias relacionadas à mobilidade elétrica.

1.1. CONTEXTUALIZAÇÃO, JUSTIFICATIVA (MOTIVAÇÃO) PARA O DESENVOLVIMENTO DO TRABALHO

Líderes e entidades do mundo estão participando de agendas de redução de emissões de gases causadores do efeito estufa. Os Estados Unidos da América, a União Européia e o Reino Unido planejam alcançar o balanço zero de emissões em 2050, a China e a Rússia em 2060, e a Índia em 2070 [1]. A Figura 1 ilustra esta expectativa.

Figura 1. Linha do tempo para alcance de emissão zero de países diversos.



Fonte: Adaptado de [1].

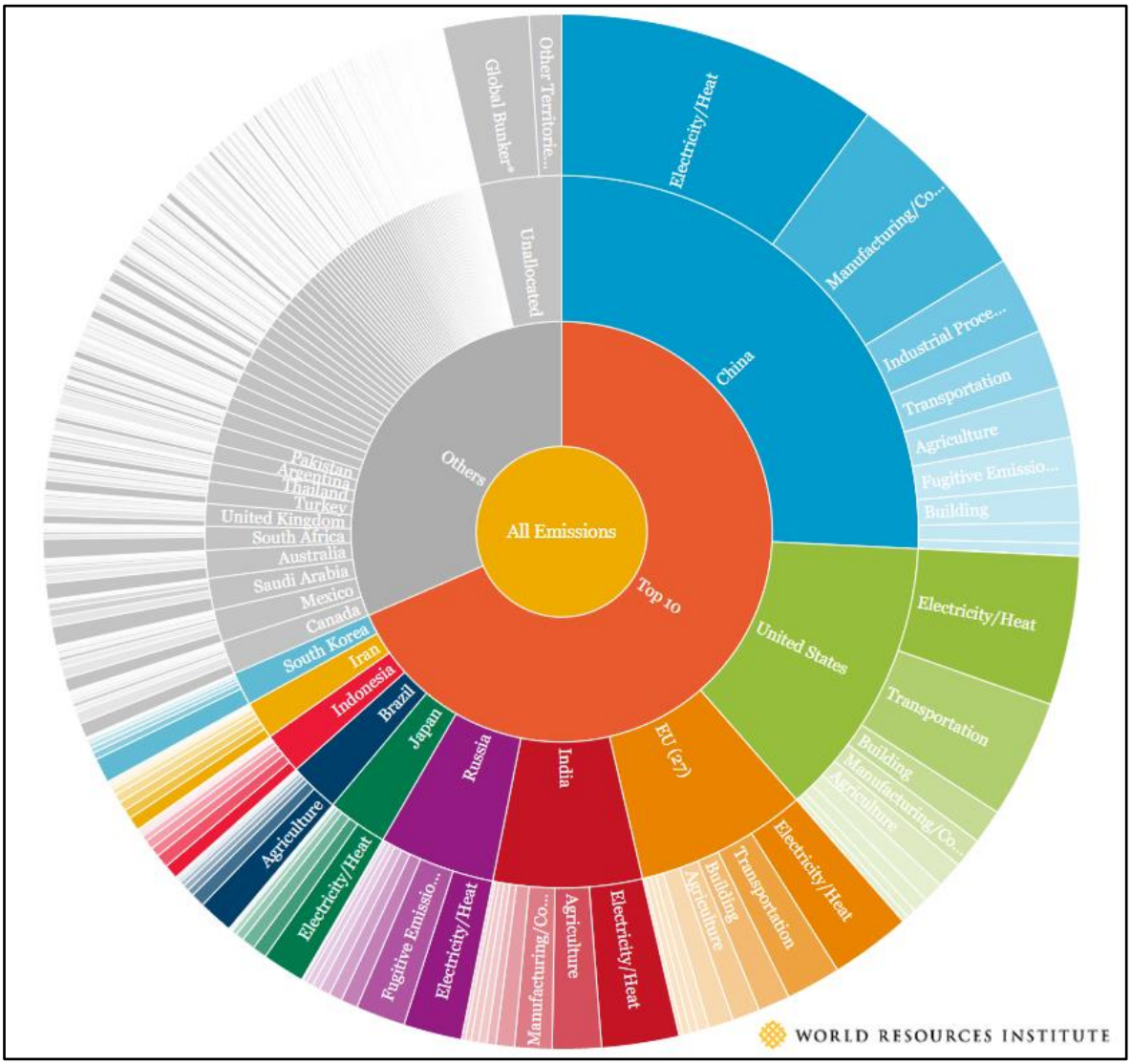
Destaca-se na Figura 2 que esses países representam 59% de toda a emissão de gases de efeito estufa do planeta, sendo o setor de transportes o responsável por 8,6% da emissão global desses gases [2].

O crescimento populacional aliado ao aumento da circulação de produtos implicará em um aumento de 77% no uso do transporte até 2055 [3]. Neste sentido, a demanda por veículos elétricos (VEs) crescerá de forma exponencial [4], pois, a utilização destes veículos pode reduzir as emissões de CO2 em 28% comparados ao uso de veículos à combustão, já em 2030 [5]. Um dos maiores desafios para a utilização de VEs é a falta de infraestrutura de carregamento, que precisa ser confiável e apresentar baixo tempo de recarga [4], além do elevado custo de aquisição do veículo.

Existem duas categorias de sistemas para estações de recarga de veículos elétricos (ERVEs), as ERVEs de condução e indução. A primeira utiliza contato físico por meio de cabos e conectores, e é subdividida em duas categorias: onboard, quando o sistema de carregamento é instalado no VE, e off-board quando o carregador está localizado fora do veículo. A segunda categoria para ERVEs é a indutiva ou carregamento sem fio, cuja energia é transferida por meio de campos magnéticos [6]. A principal vantagem deste sistema é a segurança, porém, existem diversos desafios, como perdas de potência, baixa eficiência, instabilidades frente a variações de acoplamento, dentre outras [7].

Outro desafio na implementação de ERVEs está associado à rede de distribuição, caso muitos VEs realizem o carregamento simultaneamente, um desbalanço de potência entre carga e geração pode ocorrer [8], diminuindo a estabilidade da rede no ponto de acoplamento [9]. Tensões e frequências instáveis afetam a qualidade da energia fornecida pelas ERVEs, podendo diminuir a vida útil das baterias dos VEs [10].

Figura 2. Top 10 países emissores de gases do efeito estufa.



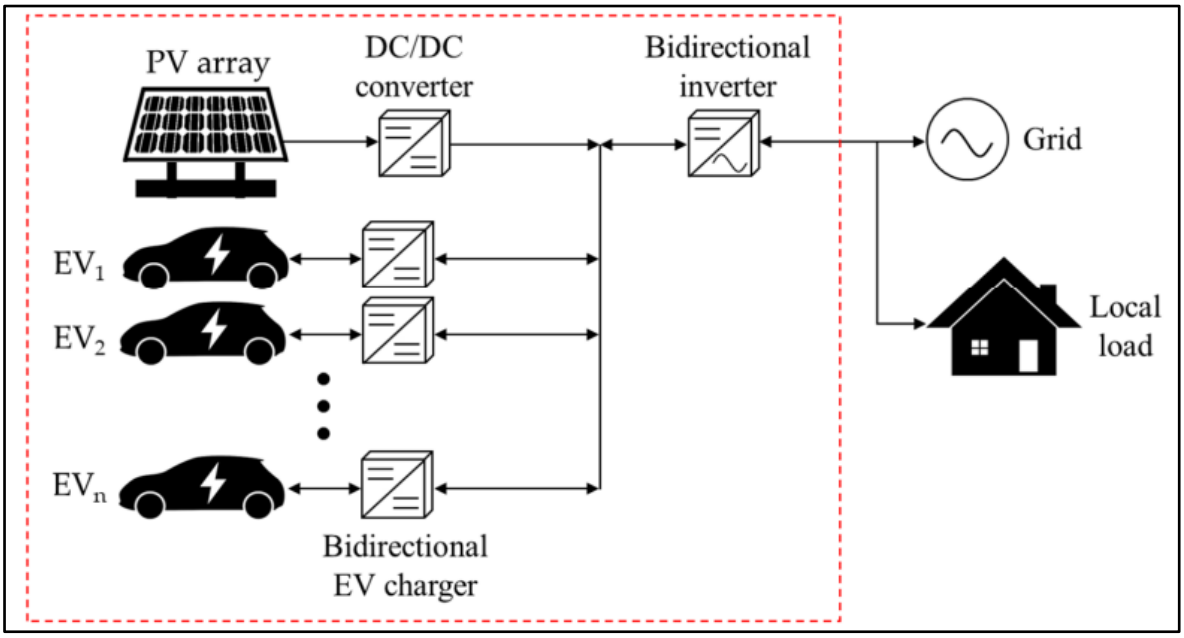
Fonte: [2]

A estabilidade da rede, incluindo queda de tensão, podem ser melhorados adotando ERVEs conectadas à sistemas de energia solar fotovoltaica [11]; ainda, a utilização de sistemas vehicle-to-grid (V2G) podem também contribuir para a estabilidade da rede elétrica [11,12]. A Figura 3 exibe a configuração de uma ERVE com um sistema de geração de energia solar fotovoltaica.

Um sistema V2G é um conceito que envolve o fluxo bidirecional de eletricidade entre VEs e a rede elétrica. Neste sistema, os VEs podem não apenas consumir eletricidade da rede, mas também fornecer o excesso de eletricidade de volta à rede quando necessário [13]

Um dos principais benefícios do V2G é sua capacidade de melhorar a confiabilidade e a estabilidade da rede. Ao utilizar a energia armazenada nas baterias dos EVs, os sistemas V2G podem ajudar a equilibrar a oferta e a demanda de energia, especialmente durante os horários de pico [13]. O V2G também pode contribuir para a estabilidade da rede, fornecendo serviços auxiliares (denominados, também, de ancilares), como regulação de frequência e suporte de tensão [14], absorvendo ou injetando energia reativa. Num futuro próximo, também a mitigação ou compensação harmônica.

Figura 3. Estação de Recarga com Sistema de Geração de Energia Solar Fotovoltaica.



Fonte: [15]

Neste sentido, esta dissertação de mestrado é fruto de um projeto de pesquisa, desenvolvimento e inovação (P&D&I) em mobilidade elétrica, intitulado Desenvolvimento de Sistema Nacional de Recarga Rápida de Bicicletas e Veículos Elétricos para Aplicações V2G (Vehicle-to-Grid), fomentado pela ANEEL. Este projeto aportou recursos para a implementação de uma microrrede laboratorial contendo os conversores necessários para a emulação de um eletroposto de 20 kW.

Os conversores desta microrrede são um conversor CC-CC bidirecional para a carga e a descarga de baterias estacionárias, um conversor bidirecional wireless para a carga e descarga das baterias das bicicletas elétricas, um conversor CC-CA bidirecional trifásico e,

para garantir o teste do sistema com geração fotovoltaica de energia, um emulador solar foi adquirido.

Neste contexto, o foco desta dissertação é a implementação do firmware do conversor trifásico bidirecional de conexão à rede e do sistema de comunicação entre os conversores da microrrede.

1.2. OBJETIVOS

Esta seção apresenta os objetivos desta dissertação.

1.2.1. Objetivo Geral

Investigar, projetar e implementar algoritmos de controle eficazes para um inversor bidirecional do eletroposto no modo grid-tied, com ênfase na aplicação V2G, utilizando um microcontrolador C2000 F2837xD como plataforma de desenvolvimento.

1.2.2. Objetivos Específicos

I. Selecionar e Caracterizar o Hardware:

• Escolher os componentes e dispositivos adequados para a implementação do inversor bidirecional que fará parte do eletroposto laboratorial.

• Realizar testes e medições para caracterizar o hardware selecionado, especialmente o sistema de condicionamento de sinais.

II. Desenvolver Algoritmos de Controle Eficientes:

• Projetar e implementar algoritmos de modulação PWM para controle eficaz da saída do inversor.
• Desenvolver algoritmos de sincronização com a rede elétrica para garantir a injeção de energia em fase.
• Criar algoritmos de controle de potência ativa e reativa, incluindo ajuste dinâmico de potência.

• Implementar algoritmos de controle online para interação V2G e garantia de estabilidade.

III. Programação do Firmware no Microcontrolador C2000 F2837xD:

• Configurar o ambiente de desenvolvimento para programação do microcontrolador.

• Adaptar os algoritmos de controle desenvolvidos para a arquitetura do microcontrolador C2000 F2837xD.

• Realizar testes e depuração para garantir o funcionamento correto dos algoritmos no microcontrolador.

IV. Preparar o sistema de comunicação do eletroposto laboratorial:

• Desenvolver firmware para a comunicação entre os conversores.

• Disponibilizar os dados de potência, tensão e corrente dos conversores.

V. Contribuir com o Conhecimento Científico e Tecnológico:

• Gerar conhecimento novo na área de sistemas de energia, controle de inversores e aplicação V2G.

• Apresentar resultados e conclusões que possam ser utilizados como referência para futuros trabalhos de pesquisa e desenvolvimento.

1.3. ORGANIZAÇÃO DO TRABALHO (RESUMO DOS CAPÍTULOS)

Este trabalho está organizado em 4 capítulos mais a conclusão. O primeiro capítulo possui uma contextualização sobre os efeitos e impactos da utilização de veículos elétricos na emissão dos gases do efeito estufa.

O segundo capítulo possui uma fundamentação teórica sobre os inversores e os principais algoritmos para a implementação das equações de controle discreto. São exibidas as características do microcontrolador utilizado, o C2000™ Delfino™ F2837xD da Texas

Instruments, com suas funcionalidades, e é introduzida a forma de comunicação com os demais conversores do eletroposto.

O capítulo 3 possui a apresentação do eletroposto comercial, com seus conversores integrantes, os algoritmos implementados no microcontrolador para o funcionamento do inversor bidirecional, através das equações de diferenças.

No capítulo 4 são exibidos os resultados em bancada dos algoritmos implementados, seguidos por uma conclusão dos resultados obtidos.

2. FUNDAMENTAÇÃO TEÓRICA

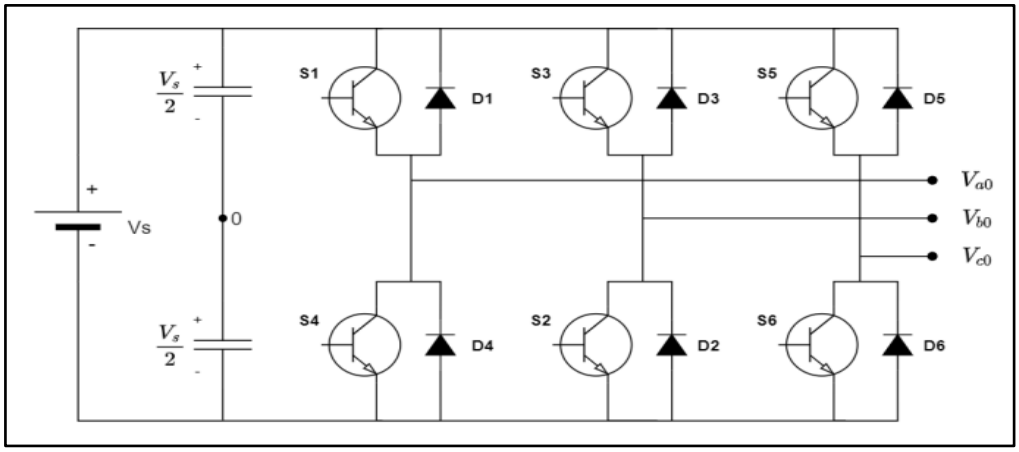
Neste capítulo é apresentada a revisão bibliográfica com os princípios técnicos que embasam o desenvolvimento do inversor bidirecional conectado à rede.

2.1. INVERSORES BIDIRECIONAIS

Inversores bidirecionais são conversores de energia que possibilitam o fluxo bidirecional de potência entre o sistema de armazenamento de energia e a rede elétrica [16].

Tais inversores estão presentes em sistemas de energias alternativas e renováveis e em veículos elétricos [17], além de diversas outras aplicações, como por exemplo, acionamento de máquinas elétricas. Nessas aplicações, especialmente em conexão à rede, o conversor precisa garantir alto fator de potência e baixa distorção harmônica total (DHT) da corrente injetada, além de regular a tensão no barramento do lado de corrente contínua [18]. O controle da tensão do barramento CC é fundamental pois os demais conversores da microrrede (eletroposto laboratorial) operam injetando ou absorvendo corrente deste barramento, e, necessitam de tensão regulada e estabilizada para operarem de forma adequada. A Figura 4 exibe o esquema de um inversor trifásico do tipo fonte de tensão (VSI - *Voltage Source Inverter*).

Figura 4. Esquema do inversor fonte de tensão trifásico.



Fonte: [19]

2.1.1. Princípio de Acionamento das Chaves

Uma das técnicas mais utilizadas para o controle da saída AC de um conversor de potência é a modulação por largura de pulso (PWM), que consiste em variar o duty cycle dos interruptores de potência em altas frequências para atingir na saída uma tensão ou corrente média instantânea desejada em cada período de chaveamento e, assim, obter um sinal aproximadamente senoidal em baixa frequência (50 - 60Hz) [21].

Para redução das harmônicas, escolha-se uma frequência de portadora que seja um múltiplo ímpar triplen das frequências de referência, como 3, 9, 15 vezes a frequência [22]. Quanto maior este valor mais fácil é a filtragem do sinal.

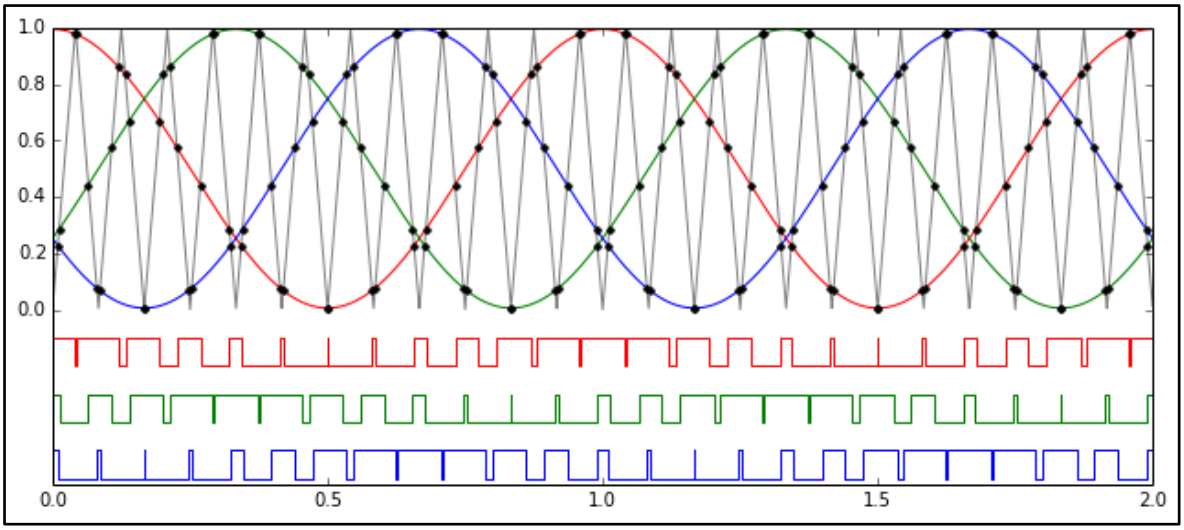
Para gerar pulsos de largura variável em sua saída, o PWM requer um sinal senoidal de referência, também chamado de sinal de controle ou modulante, e, um sinal triangular, chamado de sinal portador, que irá definir a frequência de chaveamento [22]. Esta modulação recebe o nome de SPWM, do inglês, Sinusoidal Pulse Width Modulation, a modulação por largura de pulsos senoidal.

Para inversores trifásicos, são necessários três sinais senoidais, um para cada fase, defasados em 120º, e uma única onda portadora [22]. As chaves são controladas em pares, enquanto uma chave é aberta, a outra chave do mesmo braço é fechada [22]. Um esquema para compreensão desta modulação é apresentado na Figura 5. Nesta, os sinais senoidais de

amplitude unitária, em vermelho, verde e azul, representam as referências para os respectivos braços do inversor. Os pulsos em vermelho, verde e azul representam as tensões instantâneas Vao, Vbo e Vco, da Figura 4.

Ressalta-se que a forma de onda exibida na Figura 5 possui apenas valores positivos pois se faz um paralelo com a forma de onda inserida no conversor analógico digital do microcontrolador. A forma de onda do SPWM precisa conter valores negativos para a sintetização do semiciclo negativo da senoide de tensão amostrada.

Figura 5. Esquema da modulação SPWM do inversor fonte de tensão trifásico



Fonte: [23]

2.1.2. Filtro de saída do Inversor Trifásico

O filtro de conexão com a rede tem como função filtrar os harmônicos produzidos pela comutação proveniente da modulação do conversor CC-CA para proporcionar uma forma de onda de corrente injetada ou absorvida que se aproxime da frequência fundamental da rede elétrica, ou, tecnicamente, que tenha baixo conteúdo harmônico.

Dentre as opções de filtros, se destacam o L e o LCL, por possuírem um indutor de acoplamento à rede e facilitar o controle da corrente injetada. Por apresentar maior atenuação dB/década, este trabalho utilizou o filtro LCL. No entanto, este filtro apresenta um pico de ressonância que dificulta o controle. Neste sentido, este trabalho utilizou o amortecimento passivo como técnica de mitigação do pico de ressonância, inserindo um resistor em série ao

ramo capacitivo. Outras técnicas de amortecimento, passivo ou ativo, podem ser verificadas em [24].

2.2. DISCRETIZAÇÃO DAS EQUAÇÕES DE CONTROLE

Uma das vantagens da técnica de modelagem no domínio da frequência, baseada na conversão das equações diferenciais do sistema em uma função de transferência, é a fácil observação da resposta transitória e da estabilidade do sistema, permitindo que controladores sejam projetados variando parâmetros do sistema [25].

O controle digital utiliza sinais gerados por computadores para controlar sistemas analógicos, de tempo contínuo. Um dos métodos para o projeto de controladores digitais é o redesenho digital, que consiste em desenvolver um controlador no tempo contínuo, e depois discretizar as equações de controle, obtendo o controlador no tempo discreto (domínio "z") [26].

Neste contexto existem diversos métodos para a segunda etapa do redesenho digital, como o método de Tustin, equivalente Zero-Order Holder (ZOH), método de Euler e o método de pólo-zero emparelhado [26].

2.2.1. Método de Tustin

Este método de discretização entre sistemas contínuos e discretos apresenta a melhor correspondência entre sistemas de tempo contínuo e de tempo discreto. O plano s é totalmente mapeado dentro do círculo de raio unitário. A aproximação entre o domínio "s" e o domínio "z" é feita utilizando a equação (1) [27].

$$z = e^{sT_s} \approx \frac{1 + \frac{sT_s}{2}}{(1 - \frac{sT_s}{2})} \tag{1}$$

Onde T_s é a frequência de amostragem do sistema.

2.2.2. Equação à Diferenças

Para implementação das equações discretizadas em microcontroladores é necessária a utilização da equação de diferenças oriunda da transformada Z inversa das funções de transferência.

Um exemplo de obtenção da equação de diferenças é descrito conforme as equações (2) a (7), onde n é o estado atual da variável de interesse, e n-1 é o estado anterior.

$$\frac{Y(z)}{U(z)} = \frac{B_0 z + B_1}{z - A_1} \tag{2}$$

$$\frac{Y(z)}{U(z)} = \frac{B_0 z + B_1}{z - A_1} * \frac{z^{-1}}{z^{-1}} = \frac{B_0 + B_1 z^{-1}}{1 - A_1 z^{-1}} \tag{3}$$

$$Y(z)(1 - A_1 z^{-1}) = U(z)(B_0 + B_1 z^{-1}) \tag{4}$$

$$Y(z) - A_1 z^{-1} y(z) = B_0 U(z) + B_1 z^{-1} U(z) \tag{5}$$

$$Y(z) = B_0 U(z) + B_1 z^{-1} U(z) + A_1 z^{-1} Y(z) \tag{6}$$

$$y[n] = A_1 y[n-1] + B_0 u[n] + B_1 u[n-1] \tag{7}$$

Onde A_1, B_0 e B_1 são constantes.

2.3. MICROCONTROLADOR C2000 F2837XD

O C2000™ Delfino™ F2837xD da Texas Instruments é um microcontrolador com dois processadores de 32 bits de ponto fixo mais uma unidade de ponto flutuante. O processador também conta com uma unidade matemática que realiza operações trigonométricas comuns como senos, cossenos, raízes quadradas e arcotangentes [28].

Cada núcleo é idêntico e possui acesso à sua própria RAM local e memória flash, além de uma RAM compartilhada globalmente. A comunicação entre os dois núcleos da CPU (CPU1 e CPU2) é feita através de um módulo IPC (Comunicações entre Processadores). Além disso, ambos os núcleos têm acesso a um conjunto comum de periféricos analógicos e de controle altamente integrados [29]. A Tabela 1 exibe as principais características do F2837xD.

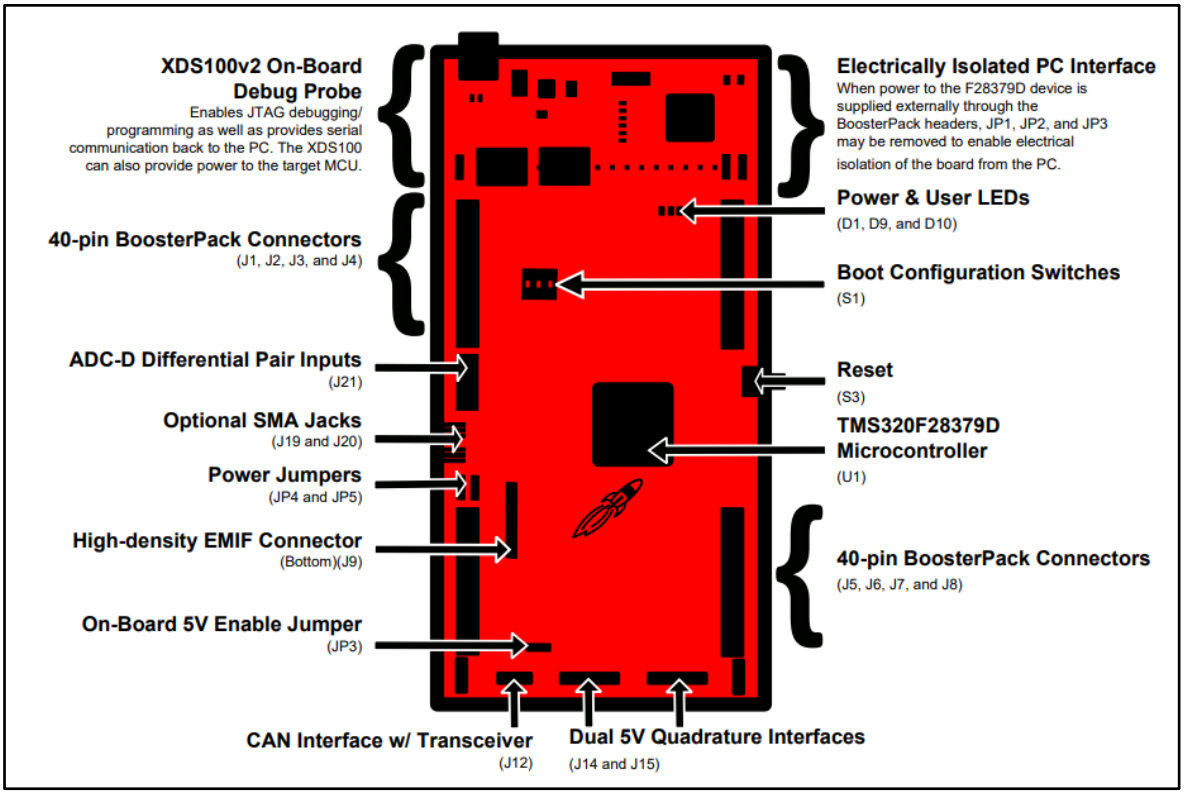
Tabela 1. Principais características do F2837xD.

Característica	Descrição
CPU	Dual-Core C28x + CLA (Control Law Accelerator)
Frequência do Clock	Até 200 MHz
Memória Flash	Até 1 MB
Memória RAM	Até 204 KB
Periféricos de Comunicação	SCI, SPI, I2C, CAN, UART
Conversores	ADCs de 16-bit, DACs de 12-bit
PWM (ePWM)	Módulos Enhanced PWM com resolução de alta precisão
Capture Modules (eCAP)	Módulos de captura avançada
Interfaces Externas	GPIOs, JTAG
Interrupções	Módulo de Expansão de Interrupção Periférica (PIE)
Timers	Timers de propósito geral e timers C28x
Proteção	ECC (Error-Correcting Code) na memória flash e RAM
Controladores	Controlador DMA (Direct Memory Access)
Recursos Especiais	Modo de baixo consumo, recursos de segurança, entre outros

Fonte: Compilado de [30].

A Plataforma de Desenvolvimento C2000™ LAUNCHXL-F28379D é uma placa completa de desenvolvimento e possui todos os componentes de hardware necessários para criar aplicativos baseados nos microcontroladores F2837xD [30], a Figura 6 exibe as características gerais da placa.

Figura 6. Características Gerais da Placa de Desenvolvimento C2000™ LAUNCHXL-F28379D



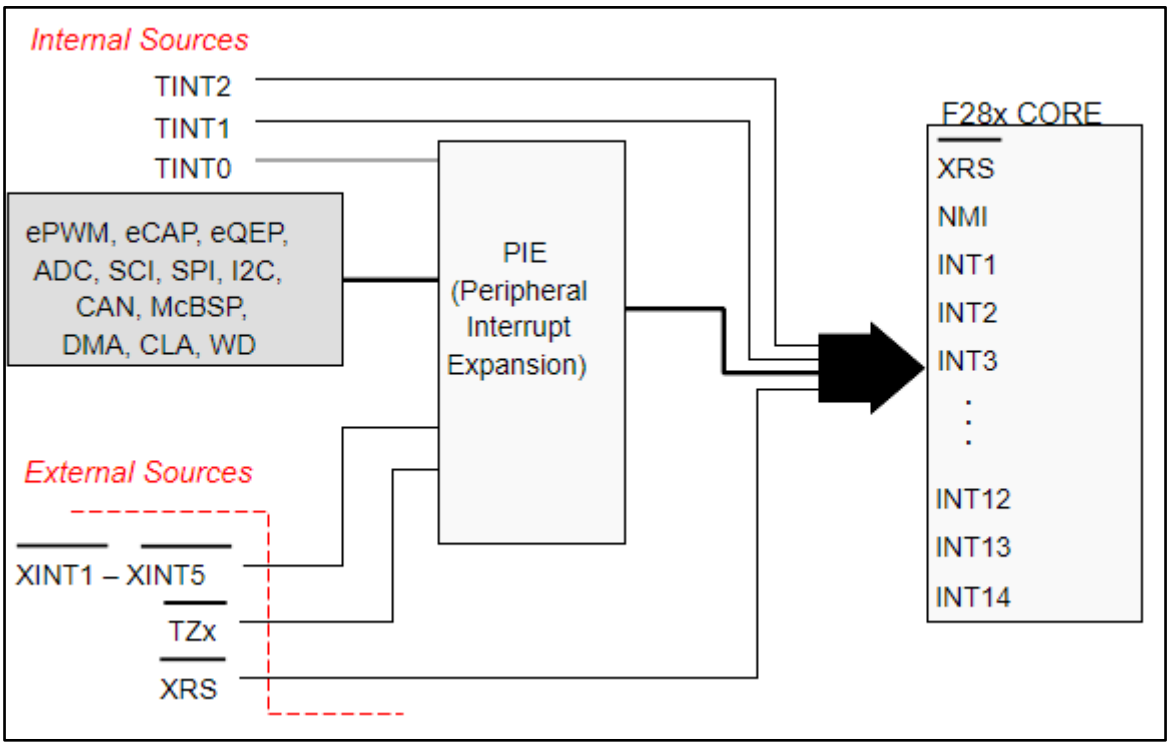
Fonte: [30]

2.3.1. Interrupções

Uma interrupção é um sinal que faz a CPU pausar sua execução e mudar para uma rotina de serviço de interrupção (ISR). A CPU C28x tem 14 linhas de interrupção. INT13 e INT14 estão ligadas aos temporizadores 1 e 2. As outras 12 linhas se conectam a sinais periféricos pelo módulo PIE, que permite multiplexar até 16 interrupções por linha e suporta 192 fontes de interrupção [30].

A Figura 7 exibe um diagrama das interrupções, além dos protocolos de comunicação possuírem a função de interrupção, existem temporizadores (TINT) e fontes externas (XINT), que podem desencadear uma interrupção no código [29].

Figura 7. Diagrama de Interrupções.



Fonte: [29]

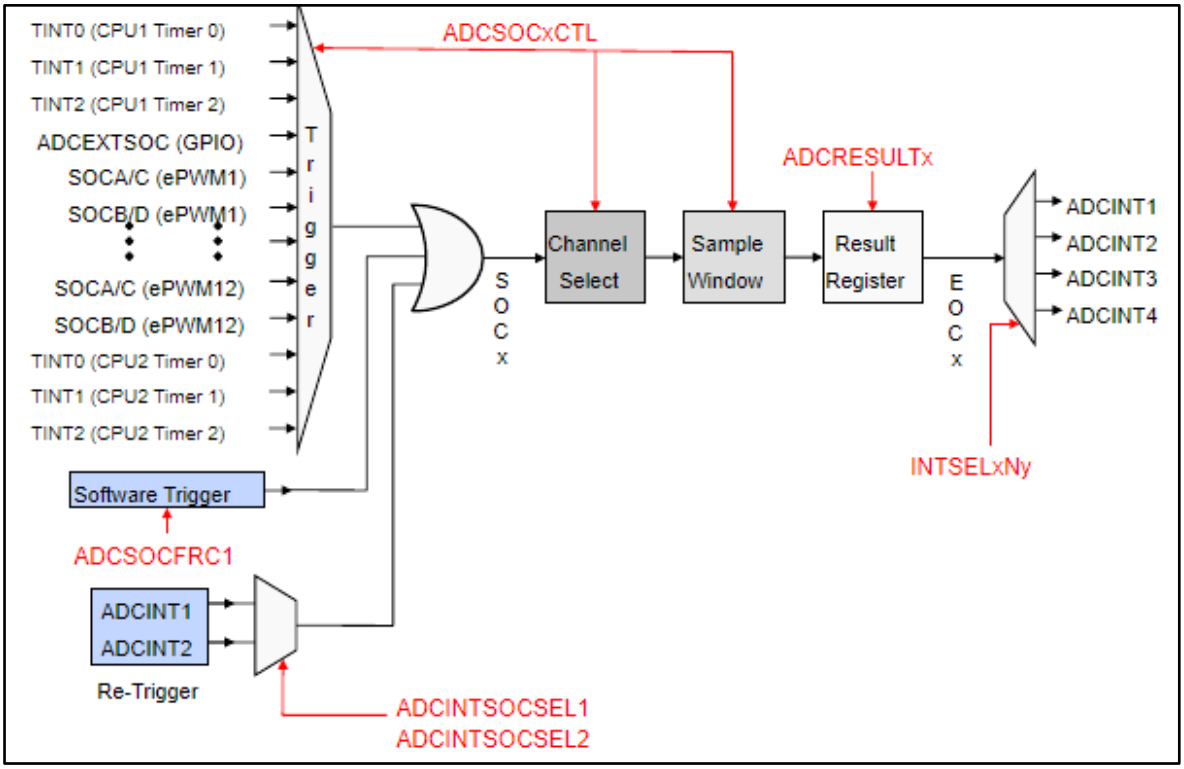
Um módulo de Expansão de Interrupção Periférica (Peripheral Interrupt Expansion - PIE) multiplexa até dezesseis interrupções periféricas em cada uma das doze linhas de interrupção da CPU. São 12 grupos (INT1 a INT12) com um registrador de 16 bits cada, que mapeiam todos os recursos de interrupção.

2.3.2. Conversor Analógico Digital (ADC)

O F28379D possui 4 ADCs independentes de 16 bits/12 bits, que podem ser acessados pelas duas CPUs. A resolução de cada ADC pode ser selecionada por software [30]. Existem diversos sinais que podem ser utilizados como gatilhos para iniciar uma conversão do ADC, como temporizadores, módulos PWM, fontes externas e o próprio código [29].

A Figura 8 exibe o fluxo de um único início de conversão de um canal ADC, os registradores estão exibidos com o texto em vermelho.

Figura 8. Fluxo de conversão ADC - Desenvolvimento C2000™ LAUNCHXL-F28379D.



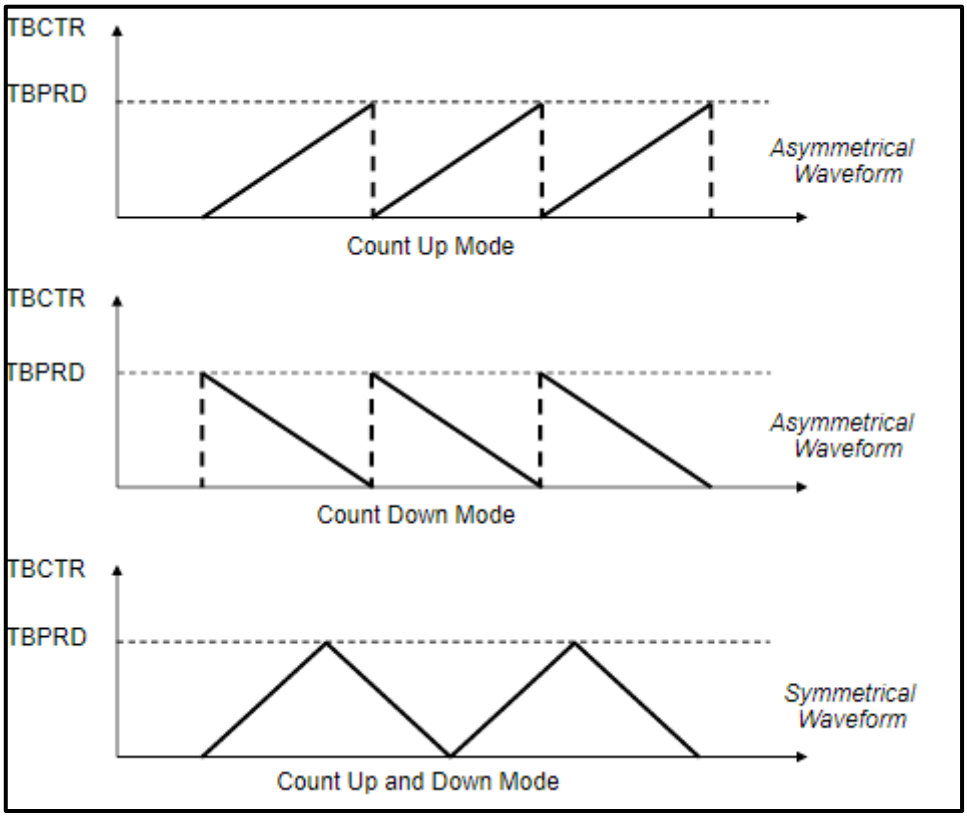
Fonte: [29]

2.3.3. Modulação por Largura de Pulso (PWM)

Cada módulo ePWM é idêntico e possui duas saídas PWM, a EPWMxA e a EPWMxB. As ondas PWM geradas estão disponíveis nas saídas dos pinos GPIO. Além disso, o módulo ePWM pode gerar sinais de início de conversão ADC e produzir interrupções para o bloco PIE [29]. Em sistemas conversores eletrônicos de potência esta estratégia é muito utilizada para a sincronização e redução de aquisição de ruídos de chaveamento.

O ePWM possui três modos de contadores de tempo: up mode, down mode, e up and down mode. A Figura 9 exibe os módulos de contagem.

Figura 9. Modos de Contagem do ePWM.



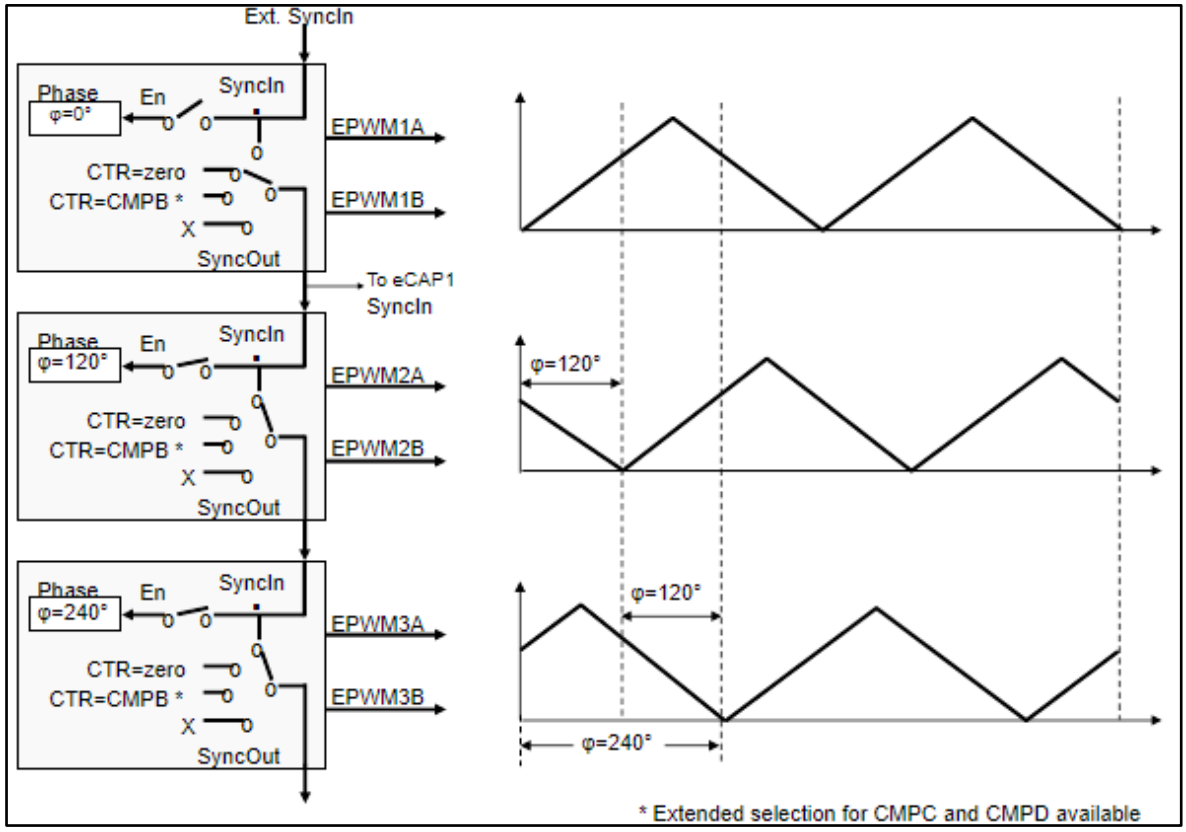
Fonte: [29]

A sincronização permite que vários módulos ePWM funcionem juntos como um único sistema. Essa sincronização se baseia em um sinal de sincronização, no contador de base de tempo igual a zero ou no contador de base de tempo igual ao registro de comparação B. Além disso, a forma de onda pode ser deslocada em fase [29]. A Figura 10 exibe um sistema de sincronização de fases do ePWM.

A saída do PWM é gerada ao comparar os registradores CMPA e CMPB com a onda contadora de tempo. A Figura 11 exibe um exemplo de sinal de saída com modulação independente para os PWM-A e B [29].

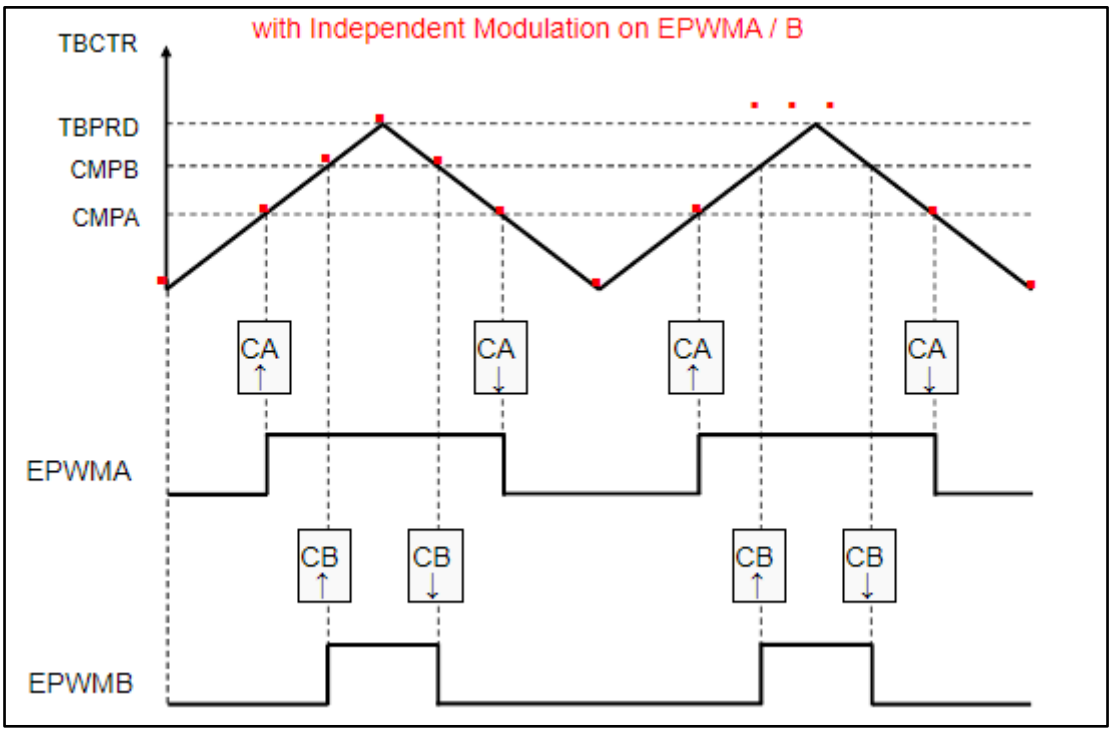
É atribuído o valor “alto” para a saída quando a onda triangular atinge o valor do registrador CMP na borda de subida, e o valor “baixo” quando atinge novamente o valor do registrador, mas na borda de descida.

Figura 10. Sincronização de fases do ePWM.



Fonte: [29]

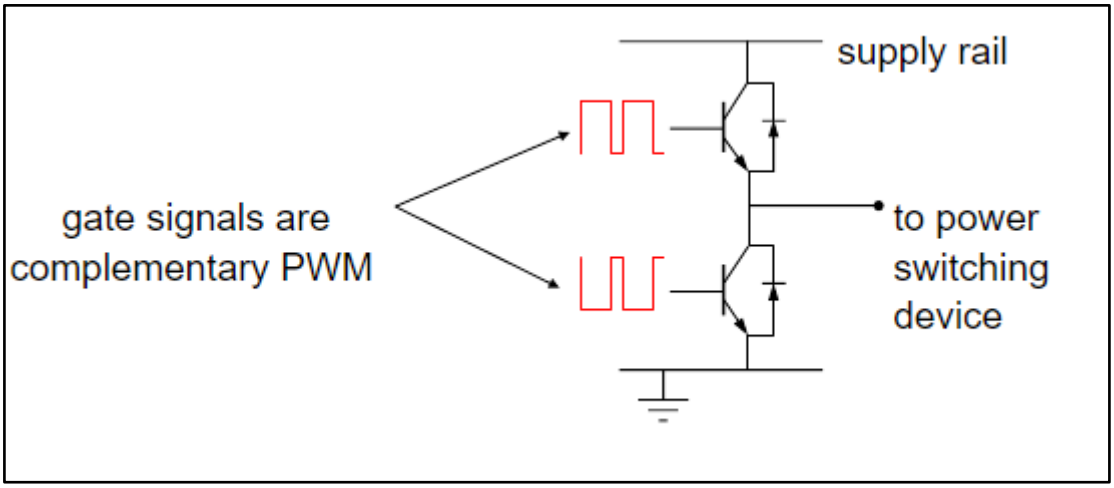
Figura 11. Forma de onda do ePWM, no modo Up-Down.



Fonte: [29]

O ePWM conta ainda com um submódulo de dead-band que atribui um delay nos sinais do PWM, disponibilizando tempo hábil para a abertura dos interruptores de potência, visto que as chaves eletrônicas fecham mais rapidamente que abrem, prevenindo um curto-circuito. Esse módulo suporta programação independente da borda de subida e borda de descida [29]. A Figura 12 exibe a motivação para o submódulo de dead-band que, quando configurado, permite que o PWM-B funcione de forma complementar ao PWM-A.

Figura 12. Dead-band e PWM Complementar.



Fonte: [29]

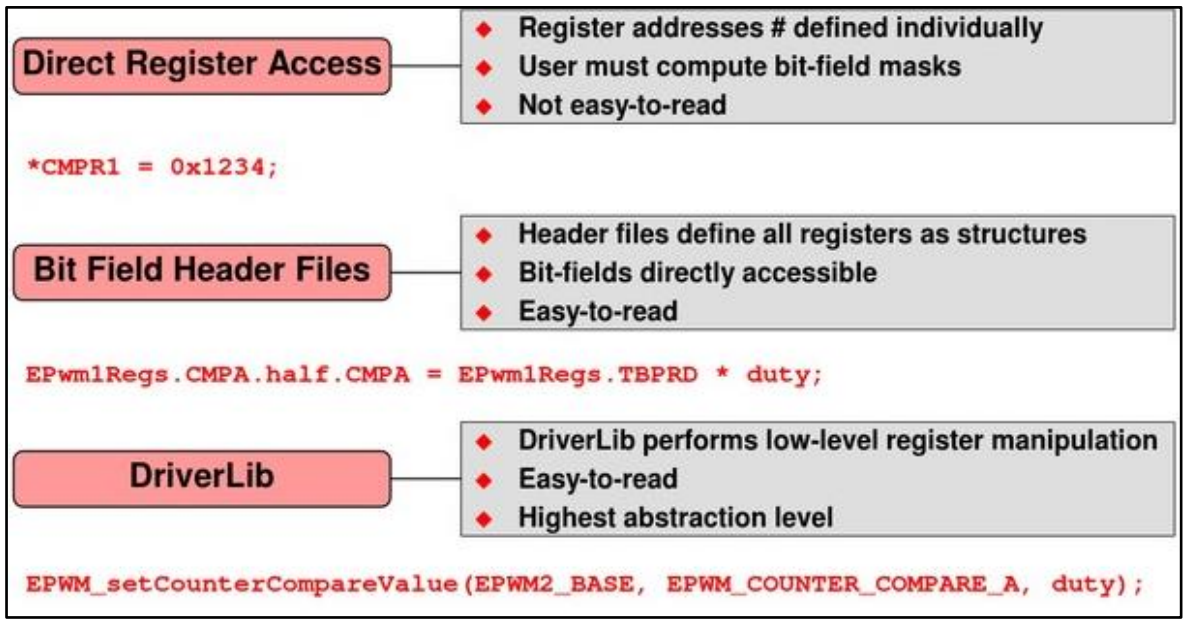
2.4. Linguagens de Programação e Ambientes de Desenvolvimento

O Code Composer Studio™ (CCS) é um ambiente de desenvolvimento integrado (IDE) para as famílias de processadores embarcados da Texas Instruments (TI). O CCS inclui um conjunto de ferramentas para desenvolver e depurar aplicações embarcadas. Ele conta com compiladores para cada família de dispositivos da Texas Instruments, editor de código-fonte, ambiente de construção de projetos, depurador, analisador, simulador, dentre outras [29].

Existem diferentes níveis de programação que oferecem diferentes níveis de abstração. O nível mais alto é o *DriverLib*, que são funções em C que configuram automaticamente os campos de bits dos registradores necessários ao funcionamento do algoritmo. O segundo nível é chamado de *Bit Field Header Files*, que são estruturas em C que permitem o acesso aos registradores como um todo ou por bits e campos de bits, e são modificados sem necessidade de mascaramento. O acesso direto ao registro é o nível mais baixo, onde o usuário define e acessa os endereços dos registros em C ou assembly [29].

A Figura 13 exibe um quadro de comparação dos modos de programação dos registradores ao configurar o valor do CMPA do ePWM.

Figura 13. Comparação dos Modos de Programação de Registradores.



Fonte: [29]

2.5. Rede de Comunicação

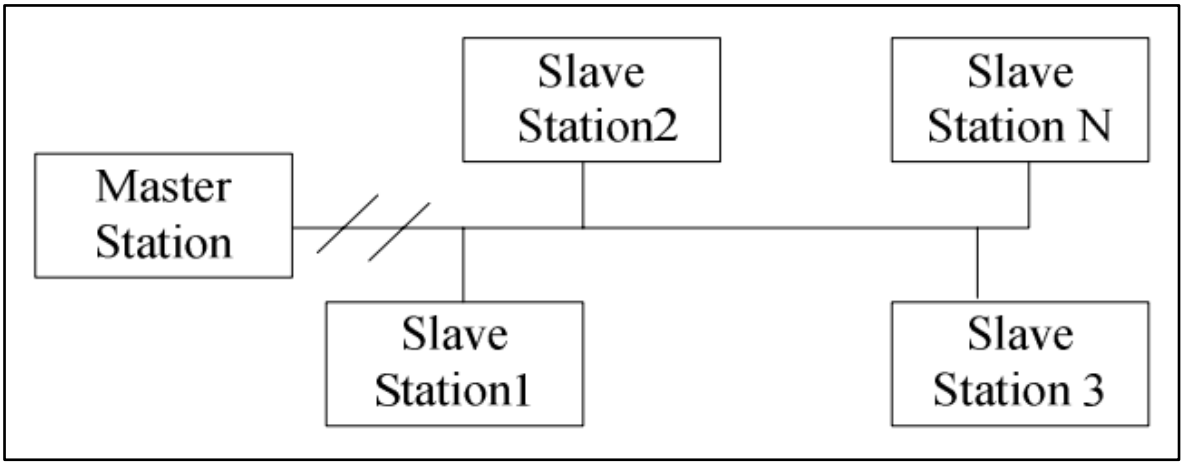
Tendo enfoque na segurança e no próprio funcionamento da aplicação, é comum observar equipamentos com sistemas de controle remoto [31]. Neste contexto se verifica a necessidade de sistemas e de protocolos de comunicação para o tráfego adequado das informações. Sistemas estes que podem ser com fio ou rede sem fio.

2.5.1. Protocolo RS485

O protocolo RS485 é um padrão baseado nos protocolos RS422 e RS232 estabelecidos pela EIA. Neste padrão, foram adicionadas capacidades de comunicação multiponto e fluxo de informação bidirecional. Este permite a conexão de múltiplos transmissores simultaneamente, melhorando a capacidade de transmissão, proteção contra conflitos e expandindo o alcance do modo comum do barramento (bus). Ele utiliza transmissão balanceada e recepção diferencial, proporcionando alta resistência a interferências [31], ponto forte e necessário em sistemas contendo elementos chaveados.

O sistema de comunicação é composto por um nó mestre e vários nós escravos. O nó mestre verifica constantemente se o nó escravo precisa se comunicar. Se necessário, concede o controle do bus aos escravos até que os dados sejam completamente enviados. O dispositivo mestre inicia a transmissão e o dispositivo escravo responde conforme os dados fornecidos pelo mestre. O mestre pode se comunicar individualmente com os escravos ou transmitir dados de forma geral. Se os dispositivos escravos desejarem se comunicar entre si, isso deve ser feito através do dispositivo mestre [31]. A Figura 14 exibe um esquema de comunicação entre mestre e escravo.

Figura 14. Comunicação em Rede entre Mestre e Escravo.



Fonte: [31]

2.6. CONECTIVIDADE COM ESP32

Desenvolvido pela Espressif Systems, o ESP32 é um microcontrolador de baixo custo com capacidades integradas de WiFi e Bluetooth. Seu núcleo baseia-se no Xtensa® Dual-Core 32-bit LX6 microprocessors, permitindo operações eficientes e multitarefa [32].

O ESP32 é compatível com os padrões IEEE 802.11b/g/n, permitindo conexões em faixas de 2,4 GHz. Ele suporta várias formas de segurança, incluindo WPA2. O chip também apresenta modos WiFi como estação (STA), ponto de acesso (AP) e ambos simultaneamente (STA/AP) [33].

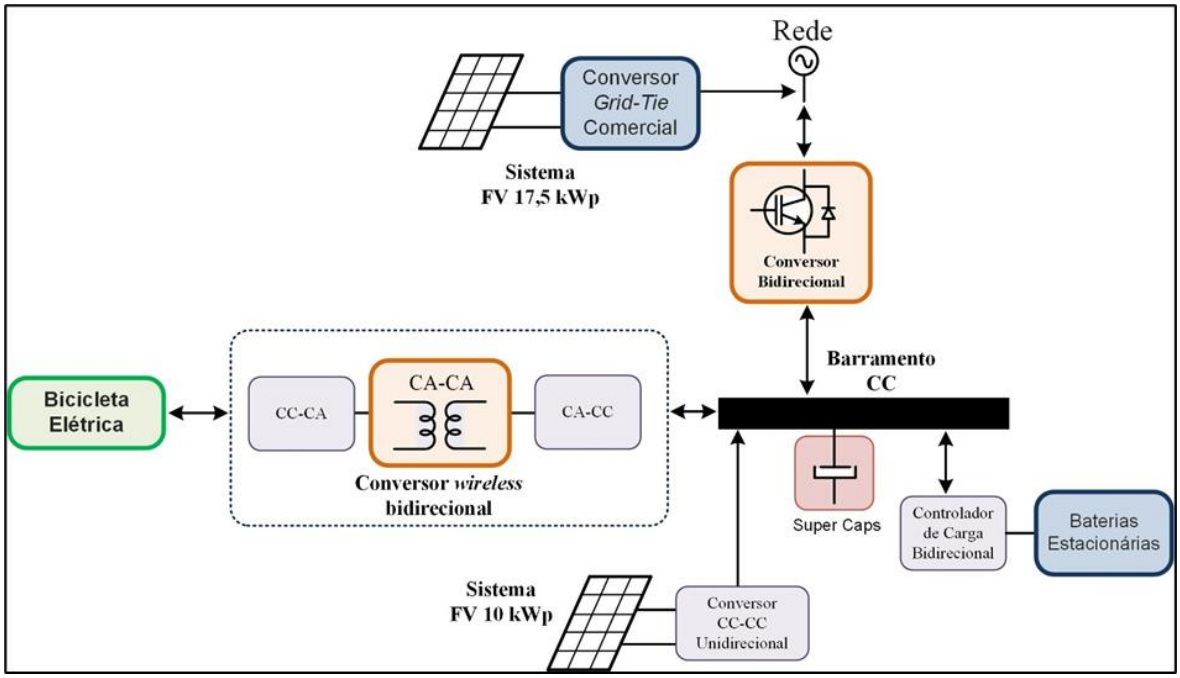
3. METODOLOGIA

3.1. SELEÇÃO E CARACTERIZAÇÃO DO HARDWARE

A Tabela 4 exibe as características do conversor bidirecional do eletroposto do Projeto de P&D Desenvolvimento de Sistema Nacional de Recarga Rápida de Bicicletas e veículos Elétricos para Aplicações V2G (Vehicle to Grid), exibido na Figura 15. O projeto dos elementos físicos foi derivado de [34], tendo esta dissertação o foco no desenvolvimento do firmware de controle e da comunicação, conforme já salientado.

Característica	Valor
Tensão no Barramento CC	400 - 500 V
Modelo do Inversor Comercial	SPCIT-2000-60-15
Potência	~11,5 kVa em 220V
Frequência de Chaveamento (fs)	15 kHz
Frequência de Amostragem	30 kHz
Quantidade de Inversores	2 em Paralelo
Quantidade de Fases	3
Indutância do Filtro L	650 µH
Capacitor do Barramento CC	16,8 mF

Figura 15. Configuração do Eletroposto Experimental.

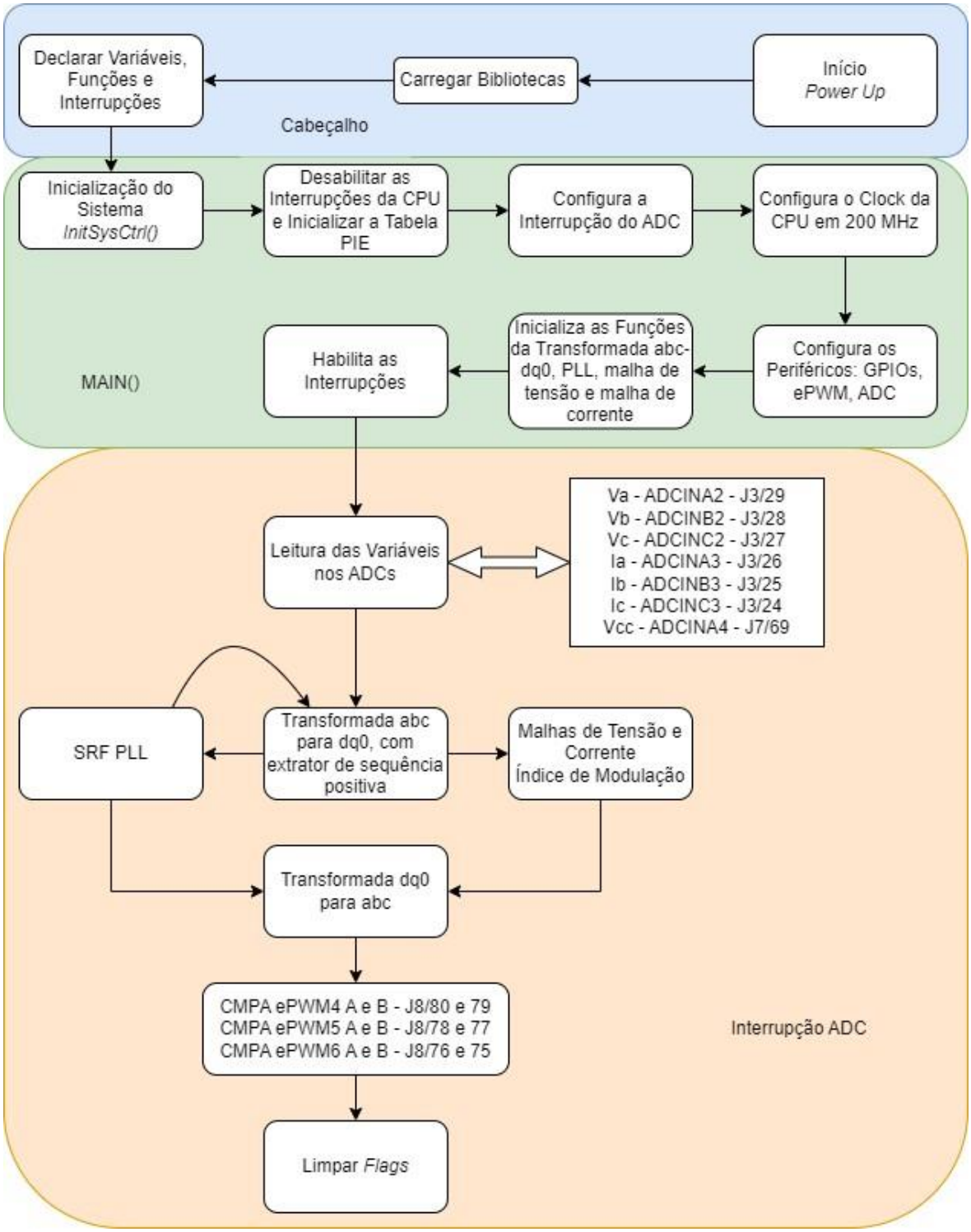


Fonte: Projeto de Pesquisa e Desenvolvimento: Desenvolvimento de Sistema Nacional de Recarga Rápida de Bicicletas e Veículos Elétricos para Aplicações V2G (Vehicle-to-Grid).

3.2. IMPLEMENTAÇÃO DOS ALGORITMOS DE CONTROLE

A Figura 16 exibe o diagrama de fluxo do código de controle e funcionamento para a geração dos pulsos de PWM que comandam os interruptores de potência do conversor bidirecional, de forma a garantir a conexão segura e a injeção de potência na rede de distribuição.

Figura 16. Diagrama de fluxo do código de controle e funcionamento.

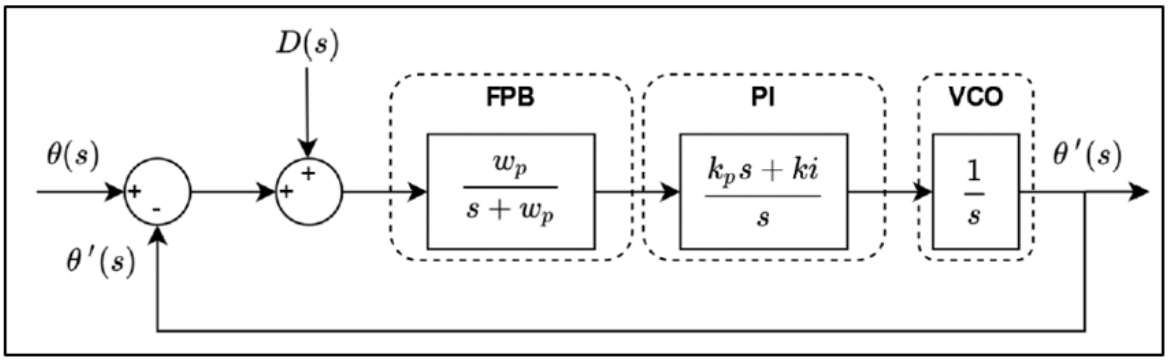


Fonte: O Próprio Autor.

3.2.1. Controle para a Sincronização com a Rede

O *phase-locked-loop* (PLL) é o estado da arte para a sincronização de conversores com uma rede elétrica existente. A Figura 17 exibe o diagrama de blocos que compõe um PLL básico. O primeiro bloco é o detector de fase (*Phase Detector* - PD). Este é responsável por gerar um erro entre as fases do sinal de saída do PLL (v') e o sinal de entrada (v), a referência do algoritmo. O laço de filtro (LF) é o segundo bloco que possui a função de atenuar as componentes alternadas de alta frequência geradas pelo detector de fase através de um compensador proporcional integral (PI) ou de um filtro passa-baixas (FPB). O último bloco é o oscilador controlado por tensão (*Voltage Controlled Oscillator* - VCO) que gera um sinal sincronizado com o sinal de entrada [35].

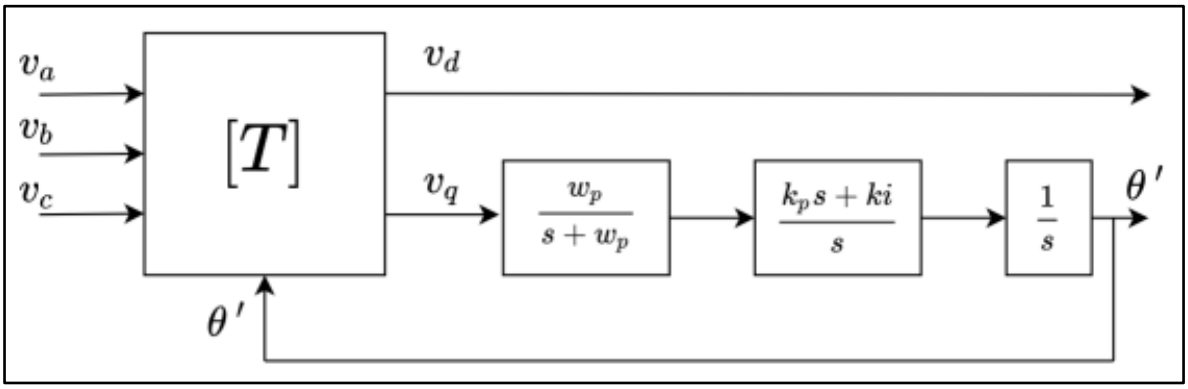
Figura 17. Diagrama de Blocos de um PLL Básico Linearizado.



Fonte: [35]

Por se tratar de um sistema trifásico, é necessário um ajuste ao algoritmo PLL para operar em um sistema trifásico. O SRF-PLL (*Synchronous Reference Frame Phase-Locked Loop*) é uma alternativa. Este utiliza as coordenadas d e q, obtidas pelas transformadas de Clarke e Park que transformam o sistema trifásico em um sistema estacionário de duas coordenadas, possibilitando a utilização da estrutura do PLL básico. A referência do PLL se torna o eixo q. Quando esta obtém valor nulo, o PLL está travado em fase. A Figura 18 exibe a estrutura de um SRF-PLL.

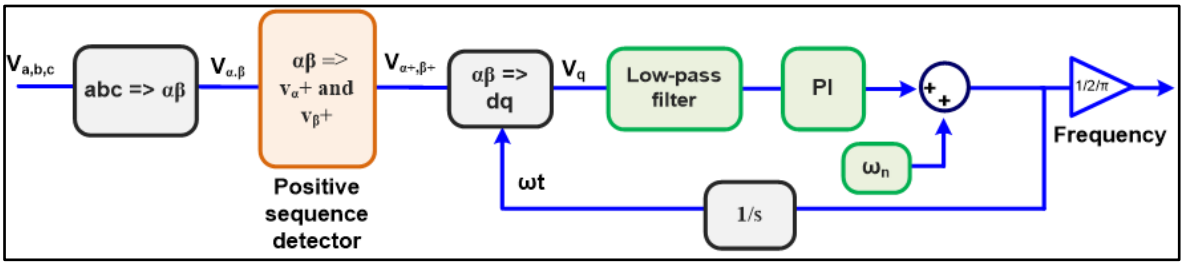
Figura 18. Diagramas de Blocos de um SRF-PLL.



Fonte: [36]

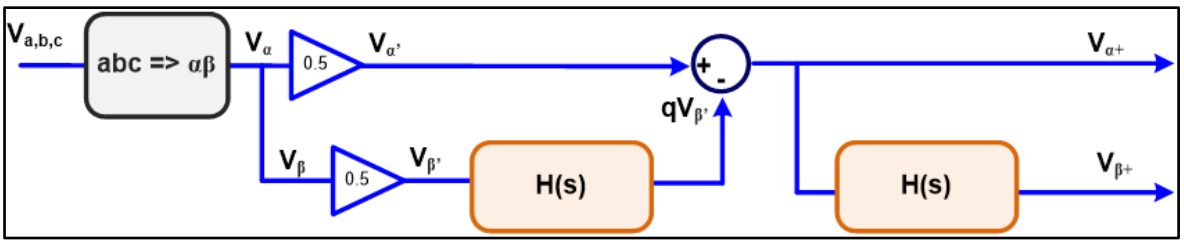
Pensando em aprimorar a resposta da frequência de saída do PLL, foi proposto um extrator de sequência positiva nos sinais α e β da transformada de Clarke para minimizar possíveis distorções nos sinais medidos, ou erros de medição por parte do conversor analógico para digital do DSP. A principal influência verificada foi o valor CC (offset) nas formas de onda do sistema de condicionamento que reduzia a assertividade do SRF-PLL. A implementação da topologia proposta é indicada pela Figura 19, e como é feita a extração da sequência positiva pela Figura 20, onde H(s) é um filtro passa-tudo com fase de 90º. A adição de wn entra como uma alimentação direta para melhoria de desempenho.

Figura 19. SRF-PLL com Extrator de Sequência Positiva.



Fonte: [37]

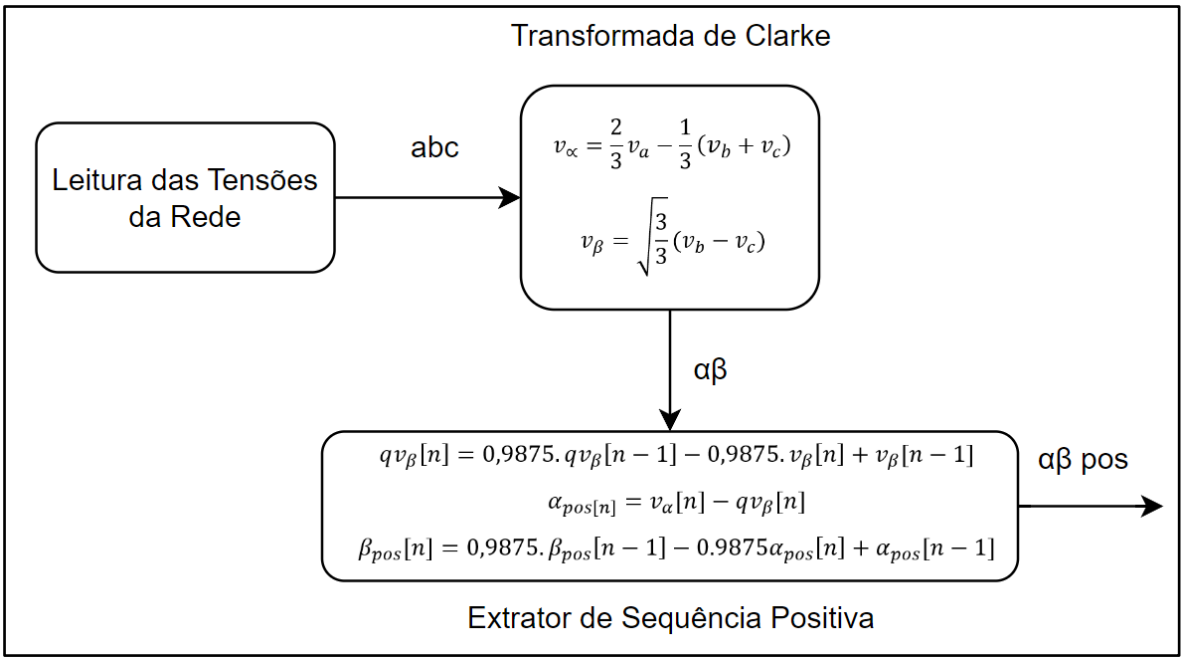
Figura 20. Extrator de Sequência Positiva.



Fonte: [37]

O diagrama da Figura 21 exibe o fluxo de implementação do extrator de sequência positiva.

Figura 21. Fluxo de Implementação do Extrator de Sequência Positiva.



Fonte: O Próprio Autor

Onde [n] representa o estado atual da variável, e [n-1] o estado anterior.

O projeto do controlador do PLL foi obtido a partir de [37]. Tendo como entrada um coeficiente de amortecimento de 0,9 e uma frequência de cruzamento de ganho em 6Hz assim, obteve-se kp = 67,86 e Ki = 1421,3. A frequência de corte do filtro passa-baixa é de 36 Hz, sendo 6 vezes maior que a frequência de cruzamento de ganho do PLL, para não influenciar em sua dinâmica.

As equações foram discretizadas utilizando o método de Tustin (bilinear).

A função contínua (s) do filtro passa-baixas do PLL é apresentada em (8). Em (9) é exibida sua forma discreta (z) e, em 10 a equação de diferenças para implementação discreta.

$$FPB_{PLL}(s) = \frac{361{,}9}{s + 361{,}9} \tag{8}$$

$$FPB_{PLL}(z) = \frac{0{,}0068008z + 0{,}005984}{z - 0{,}988} \tag{9}$$

$$FPB_{PLL}(n) = 0{,}988.FPB_{PLL}(n-1) + 0{,}0068008u(n) + 0{,}005984u(n-1) \tag{10}$$

Pensando em melhorar a resposta da detecção de fase em termos de estabilização e rejeição de ruídos, optou-se pela utilização de um filtro de segunda ordem, implementando dois filtros passa-baixas de primeira ordem em cascata. Ambos os filtros sintonizados na mesma frequência de corte.

A função contínua do PI, em (s) do PLL é apresentada em (11), em (12) é exibida sua forma discreta (z), e em 13 a equação de diferenças.

$$PI_{PLL}(s) = \frac{150{,}8s + 9475}{s} \tag{11}$$

$$PI_{PLL}(z) = \frac{150{,}9z - 150{,}6}{z - 1} \tag{12}$$

$$PI_{PLL}(n) = PI_{PLL}(n-1) + 150{,}9u(n) - 150{,}6u(n-1) \tag{13}$$

A função contínua (s) do VCO do PLL é apresentada em (14), em (15) é exibida sua forma discreta (z), e em (16) a equação de diferenças.

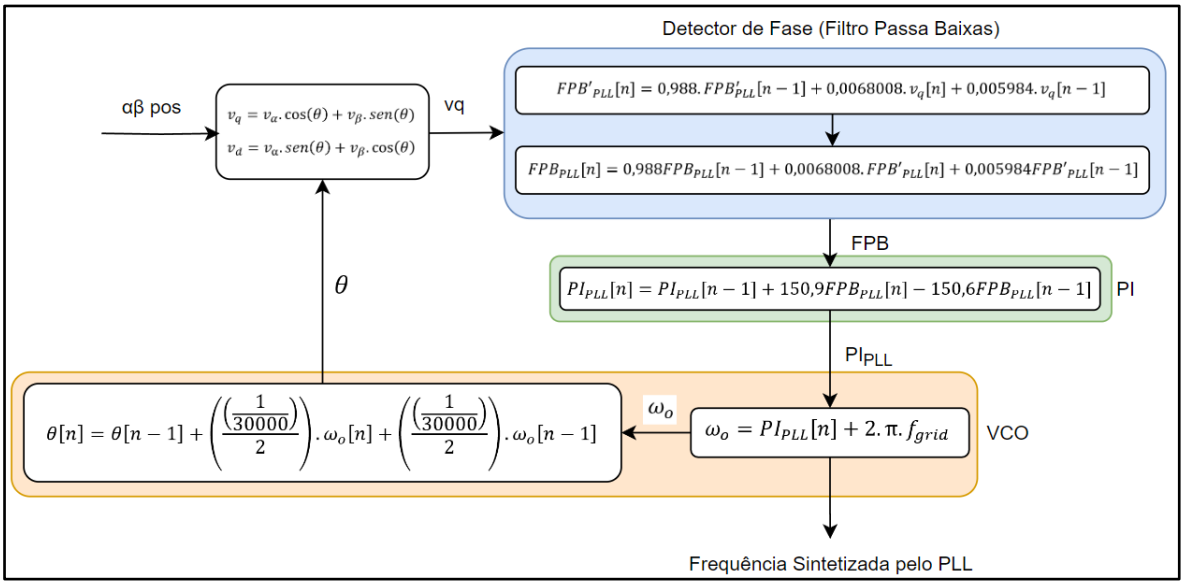
$$VCO_{PLL}(s) = \frac{1}{s} \tag{14}$$

$$VCO_{PLL}(z) = \frac{1{,}667x10^{-5}z + 1{,}667x10^{-5}}{z - 1} \tag{15}$$

$$VCO_{PLL}(n) = VCO_{PLL}(n-1) + 1{,}667x10^{-5}u(n) + 1{,}667x10^{-5}u(n-1) \quad (16)$$

A Figura 22 exibe o fluxo de implementação do SRF-PLL. A entrada do sistema é a saída do extrator de sequência positiva, realimentado com o ângulo theta gerado pelo algoritmo do PLL. Onde fgrid é a frequência esperada da rede em Hz.

Figura 22. Fluxo de Implementação do SRF-PLL.



Fonte: O Próprio Autor

3.2.2. Malhas de Tensão e Corrente

O controle de potência ativa é feito através do controle da tensão do barramento CC do conversor. Este gera a referência de corrente de pico, que será sincronizada via PLL, para a malha de corrente. A obtenção destes controladores não foi o foco deste trabalho, sendo obtido em colaboração com pesquisadores do projeto V2G ANEEL. A função contínua (s) do PI da malha de tensão é apresentada em (17), em (18) é exibida sua forma discreta (z), e em (19) é exibida a equação à diferenças para implementação desta malha.

$$id_{ref}(s) = \frac{3{,}423s + 115{,}3}{s} \quad (17)$$

$$id_{ref}(z) = \frac{3{,}425z - 3{,}421}{z - 1} \tag{18}$$

$$id_{ref}(n) = id_{ref}(n-1) + 3{,}425.Vcc_{err}(n) - 3{,}421.Vcc_{err}(n-1) \tag{19}$$

A função contínua (s) do PI da malha de corrente é apresentada em (20), em (21) é exibida sua forma discreta (z), e em (22) é exibida a equação à diferenças.

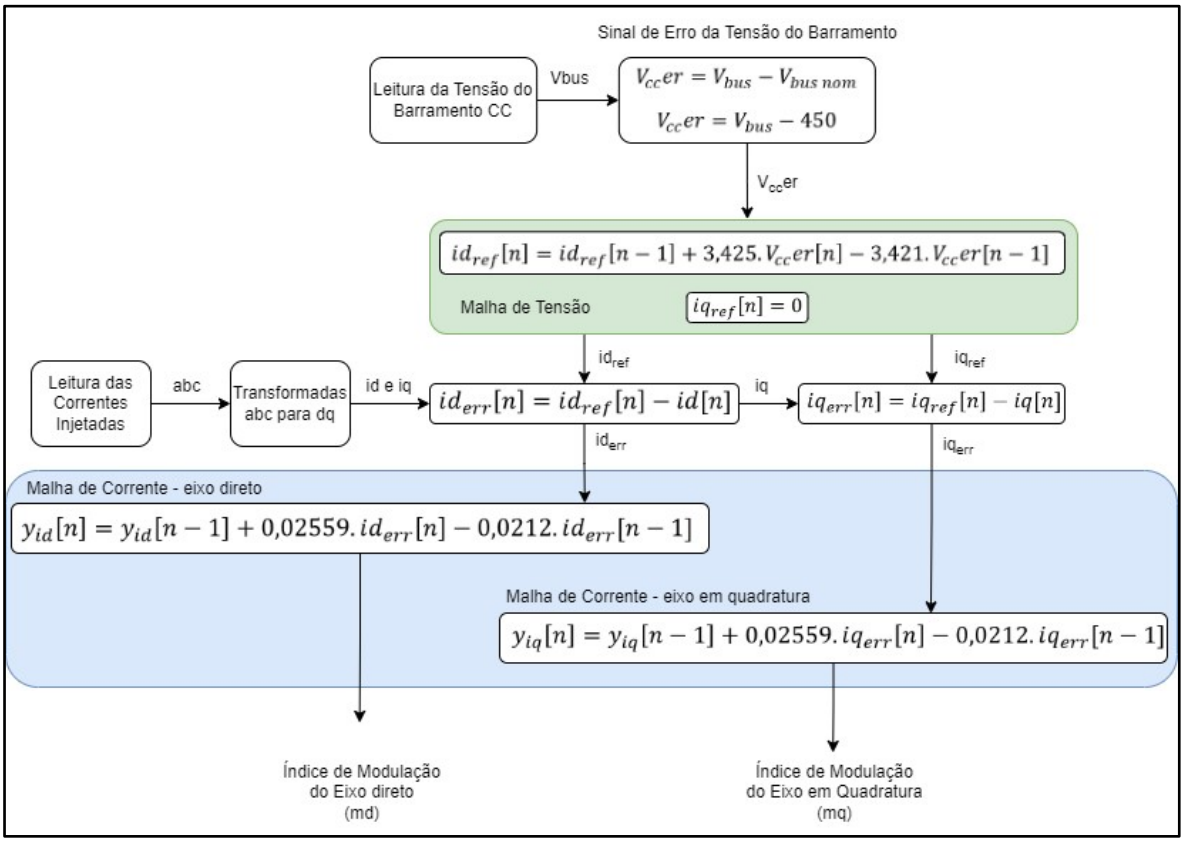
$$PI_{mc}(s) = \frac{0{,}0234s + 131{,}6}{s} \tag{20}$$

$$PI_{mc}(z) = \frac{0{,}02559z - 0{,}02121}{z - 1} \tag{21}$$

$$PI_{mc}(n) = PI_{MC}(n-1) + 0{,}02559u(n) - 0{,}02121u(n-1) \tag{22}$$

A Figura 23 exibe o fluxo de implementação das malhas de tensão e corrente. O sistema possui como entradas a tensão do barramento CC e as correntes injetadas pelo conversor bidirecional. A partir daí, são produzidos os índices de modulação md e mq, que são transformados para os eixos abc com as transformadas inversas de Clarke e Park, utilizando o ângulo *theta* do algoritmo do SRF-PLL. Neste momento, não há injeção de potência reativa, sendo, portanto, iqref = 0. Alterações nesta variável fazem o controle da injeção ou absorção de reativos. Uma malha adicional, pode ser inserida, para dado um valor de potência reativa de referência (Qref) e da potência reativa mensurada, calcular o valor necessário para iqref.

Figura 23. Fluxo de Implementação das Malhas de Tensão e Corrente.



Fonte: O Próprio Autor.

3.2.3. Comunicação

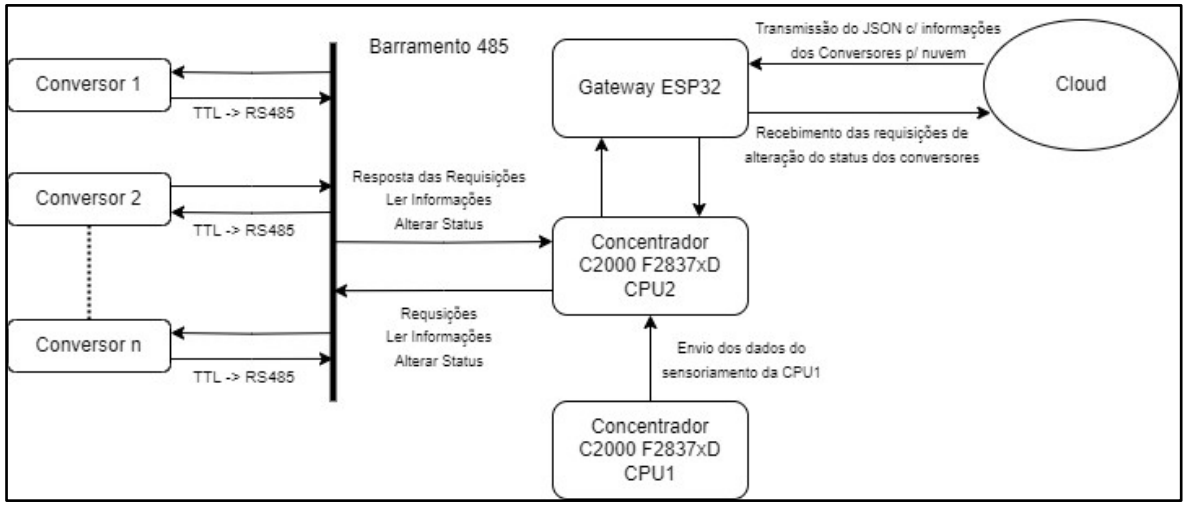
A Figura 24 exibe um diagrama da comunicação dos conversores do eletroposto experimental. Os conversores de carregamento dos veículos elétricos, do sistema de energia solar fotovoltaica e do controlador de carga das baterias estacionárias se comunicam com a CPU2 do F2837xD via barramento RS485. O DSP possui duas CPUs, a CPU1 e CPU2. A CPU 1 é responsável pelo controle do inversor bidirecional de conexão à rede e a CPU 2, do mesmo DSP, é responsável pela comunicação. Na estratégia de comunicação, o DSP concentra as informações de todos os conversores do sistema, antes dos dados serem disponibilizados. O *baudrate* utilizado foi de 115200.

O DSP se comunica então com um ESP32 para o envio dos dados para a nuvem. Na nuvem existe um algoritmo de controle, que não é escopo deste trabalho, que responde o status dos conversores, devendo o concentrador enviar de volta para os respectivos conversores se estes devem permanecer ligados ou desligados de acordo com a estratégia de

controle do algoritmo, que pode buscar maximizar a receita ou o estado de carga das baterias. Este algoritmo está em desenvolvimento em outra frente de trabalho. Resultados preliminares podem ser observados em [38].

A CPU1, além de possuir todos os dados de tensão, corrente e potência do conversor bidirecional, também monitora a rede elétrica. Assim, a CPU1 operando em sincronia com a CPU2, envia os dados das leituras de tensões e correntes da rede elétrica para a CPU2, que, por sua vez, também envia para a nuvem.

Figura 24. Fluxo da Comunicação entre Conversores.



A comunicação é feita por requisições, que são disparadas periodicamente para todos os nós da rede. É função do concentrador o controle das requisições de leitura a serem enviadas para cada um dos nós da rede. Cada requisição é individual para cada conversor, sendo diferenciada no pacote através do campo CONV_ID.

A Tabela 2 exibe o pacote de dados enviado pelo concentrador em forma de requisição.

Tabela 2. Pacote de Dados de Requisição Enviado aos Nós do Barramento.

Posição	Valor	Descrição
0	‘[‘ (0x5B)	Start Byte
1	CONV_ID	ID do Conversor
2	0x01 (CONV_REQ_CMD)	Comando para leitura de informações do conversor.

3	‘]’ (0x5D)	End Byte
4	0x0D	Quebra de Linha ‘/r’
5	0x0A	Retorno de carro ‘/n’

Fonte: O Próprio Autor.

A resposta do conversor para a requisição em questão possui o pacote conforme exibido na Tabela 3.

Tabela 3. Pacote de Dados de Resposta do Conversor.

Posição	**Valor**	**Descrição**
0	‘[‘ (0x5B)	Start Byte
1	CONV_ID	ID do Conversor
2	0x01 (CONV_REQ_CMD)	Comando para leitura de informações do conversor.
3	Informação 1 - 4 Bytes em Little Endian podem ser alocados aqui. É possível alocar um float, ou um long, de acordo com a escolha do desenvolvedor do conversor.	Campo para Informação 1.
4		
5		
6		
7	Informação 2 - 4 Bytes em Little Endian podem ser alocados aqui. É possível alocar um float, ou um long, de acordo com a escolha do desenvolvedor do conversor.	Campo para Informação 2.
8		
9		
10		
11	Informação 2 - 4 Bytes em Little Endian podem ser alocados aqui. É possível alocar um float, ou um long, de acordo com a escolha do desenvolvedor do conversor.	Campo para Informação 2.
12		
13		
14		
15	‘]’ (0x5D)	End Byte

Fonte: O Próprio Autor.

O pacote de dados enviado do ESP32 para o concentrador, que é replicado ao Barramento 485, para a alterações dos status dos conversores, está exibido na Tabela 4.

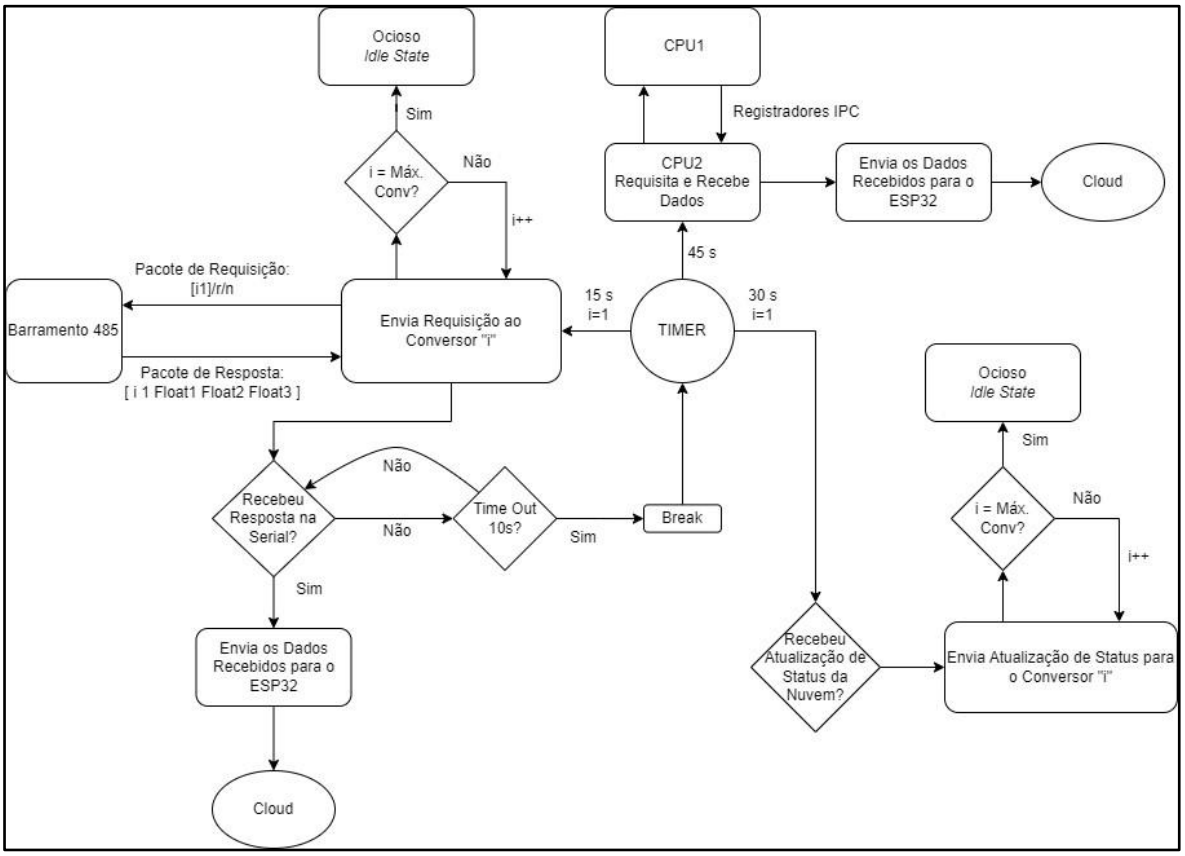
Tabela 4. Pacote de Dados de Alteração de Status.

Posição	**Valor**	**Descrição**
0	‘[‘ (0x5B)	**Start Byte**
1	CONV_ID	ID do Conversor
2	0x02 (STATUS_REQ_CMD)	Comando para alteração de status.
3	Novo Estado (0x01, 0x02, 0x03…)	Novo estado do conversor. Valores a definir ainda, mas sugiro: - 0x00 Desligar - 0x01 Ligar - 0x02 Carga
4	‘]’ (0x5D)	End Byte
5	0x0D	Quebra de Linha ‘/r’
6	0x0A	Retorno de carro ‘/n’

Fonte: O Próprio Autor.

A Figura 25 exibe o fluxo de comunicação do sistema.

Figura 25. Fluxo de Comunicação do Sistema.

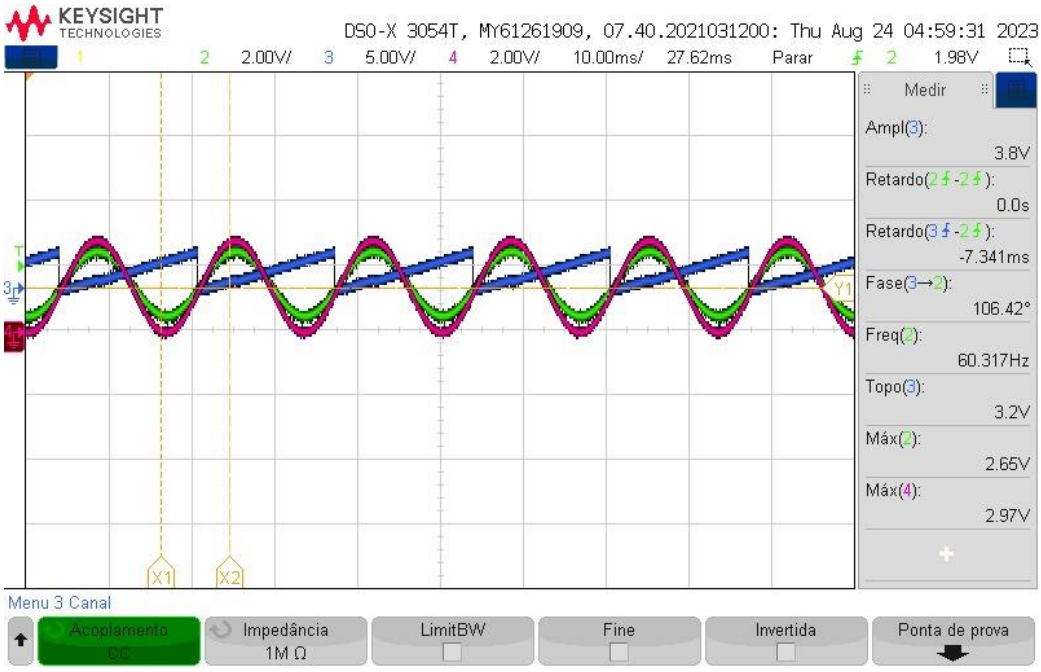


Fonte: O Próprio Autor.

CONCLUSÕES

A Figura 26 exibe a sincronização entre a rede e o sinal de saída do algoritmo SRF-PLL. O canal 1, em magenta (CH1), representa a fase A da rede elétrica na saída do condicionamento de sinais, o canal 2, em verde (CH2), a resposta senoidal gerada pelo PLL obtida via DAC (Digital-to-Analog Converter) interno ao DSP, e o canal 3, em azul (CH3), mostra o ângulo theta de saída da execução do código do PLL, demonstrando a correta sincronização.

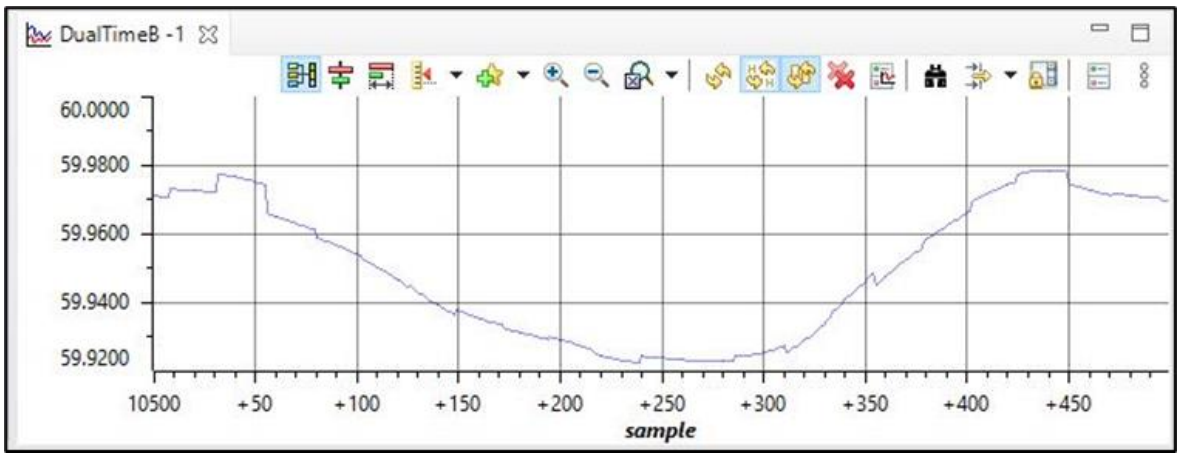
Figura 26. Resultados da implementação do SRF-PLL.



Fonte: O Próprio Autor.

A frequência gerada pelo algoritmo PLL é exibida na Figura 27. Observa-se que há uma oscilação considerável em torno de um valor médio, indicando a necessidade de melhorias no algoritmo. Esta tela foi capturada diretamente da interface de programação do DSP, no Code Composer. Foi proposto e implementado um extrator de sequência positiva na componente beta da transformada de Clarke.

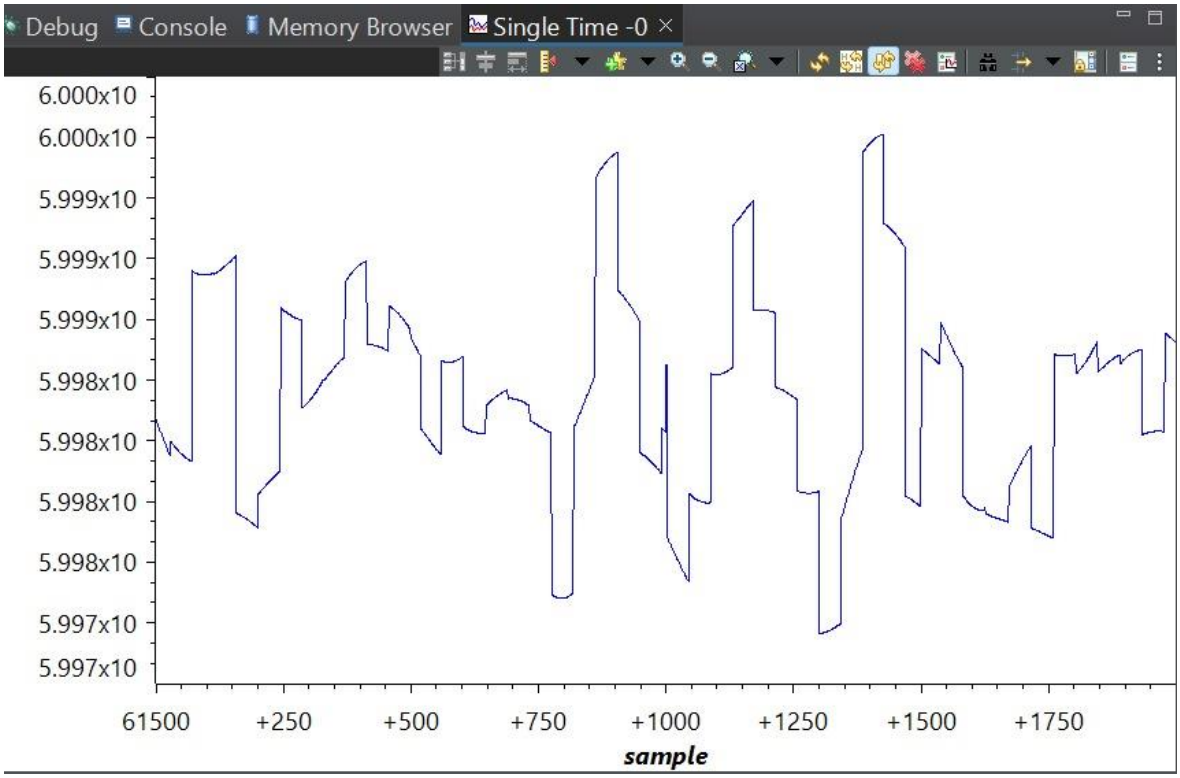
Figura 27. Frequência do PLL Antes da Extração de Sequência Positiva.



Fonte: Próprio Autor.

Na Figura 28 observa-se a resposta da frequência de saída do PLL, com o extrator de sequência positiva, apresentando melhora considerável com relação ao resultado da Figura 27.

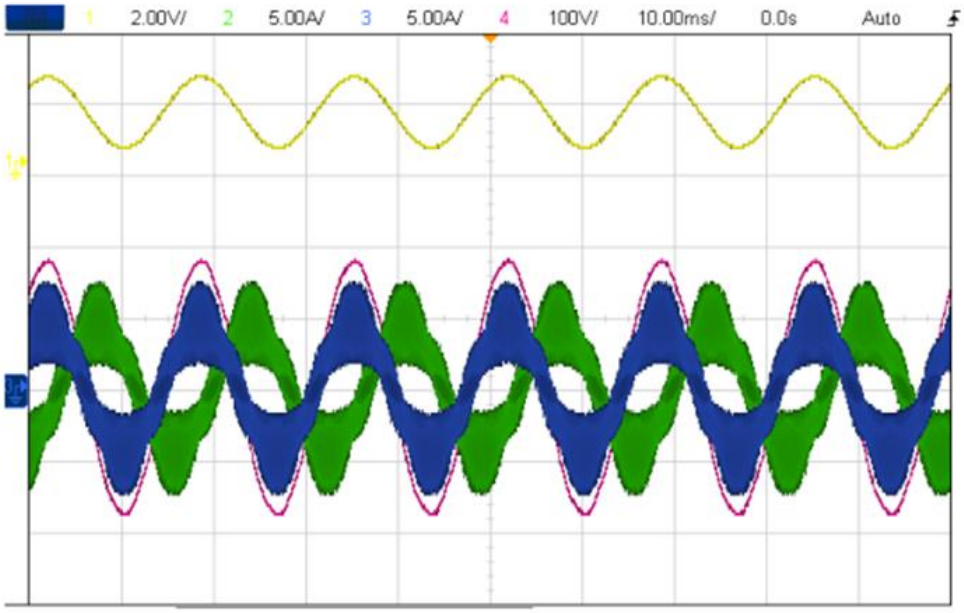
Figura 28. Frequência Obtida pelo PLL após Inserção da Extração de Sequência Positiva.



Fonte: Próprio Autor.

A Figura 29 exibe um teste de funcionamento em malha fechada do inversor bidirecional alimentando, em primeiro momento, um conjunto de cargas resistivas, mas em sincronia com a rede de distribuição – emulando um estágio de pré-conexão. Os canais 2 e 3 (azul e verde) exibem as formas de ondas das correntes de duas fases (A e B) do inversor. Em rosa (canal 4), é exibida a forma de onda da fase A da rede, em contraste com o canal 1, que possui a forma de onda de sincronismo. Destaca-se que a baixa corrente circulando no sistema produz o ripple observado. O sistema opera em 1,2 kW o que demonstra a viabilidade do firmware desenvolvido.

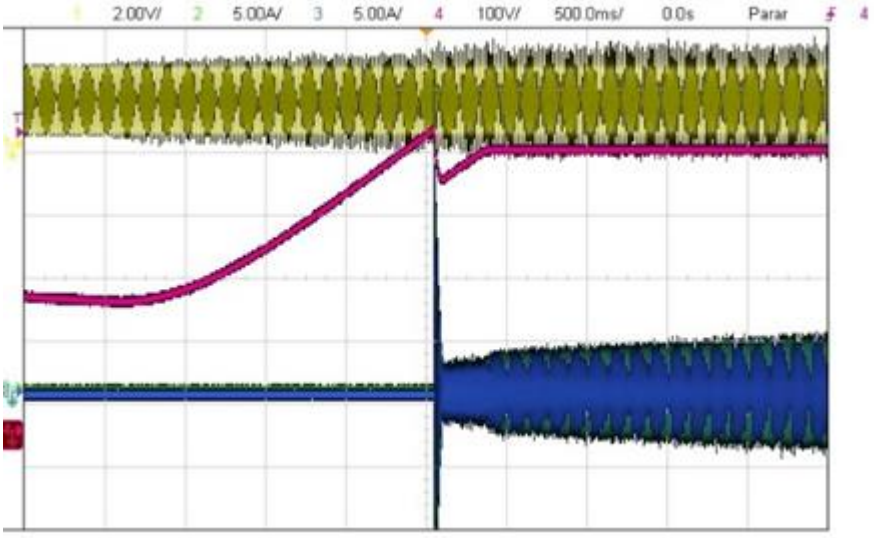
Figura 29. Formas de onda para a potência de 1,2 kW.



Fonte: O Próprio Autor.

Na Figura 30 é possível verificar o instante de conexão do conversor à rede elétrica, que ocorre exatamente após a tensão no barramento CC atingir 450V. Nesta figura, em amarelo tem-se a tensão de referência, em magenta, a tensão no barramento CC e, em verde e azul, as correntes injetadas na rede. O sistema apresenta overshoot de correntes no instante da conexão, mas de valor menor do que o nominal do inversor, que logo cessa e, passa a injetar potência de maneira estável.

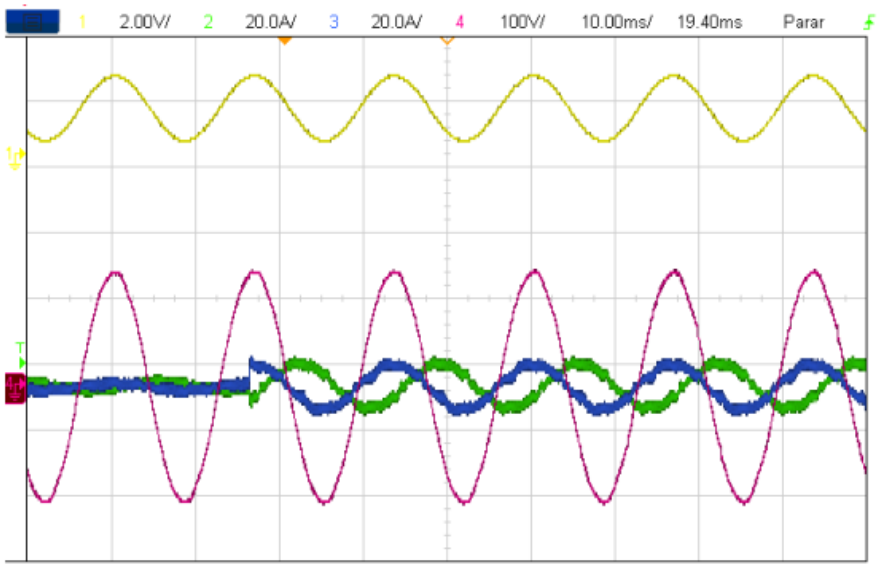
Figura 30. Resultados experimentais para o instante de conexão do conversor à rede.



Fonte: O Próprio Autor.

Resultado para uma variação de potência injetada na rede pode ser verificado na Figura 31. O degrau de corrente é de Id=2A para Id=8A, demonstrando a estabilidade do controle de corrente e, a qualidade do firmware desenvolvido.

Figura 31. Resultados experimentais para um degrau de corrente de referência.

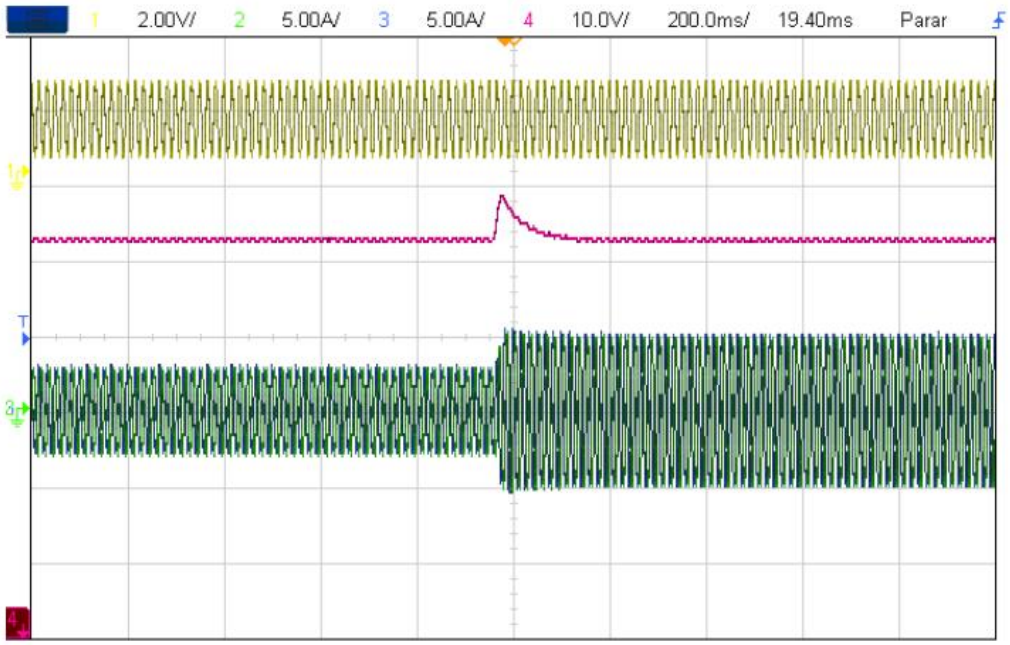


Fonte: O Próprio Autor.

Neste mesmo contexto, de degrau de potência, pode-se observar a variação da tensão no barramento CC, devido a um aumento de potência (inserida via aumento da corrente do barramento CC por uma fonte externa). Observa-se, em magenta, o controle

adequado da tensão no barramento CC, estabilizando a tensão no patamar da tensão de referência, como observado pela Figura 32.

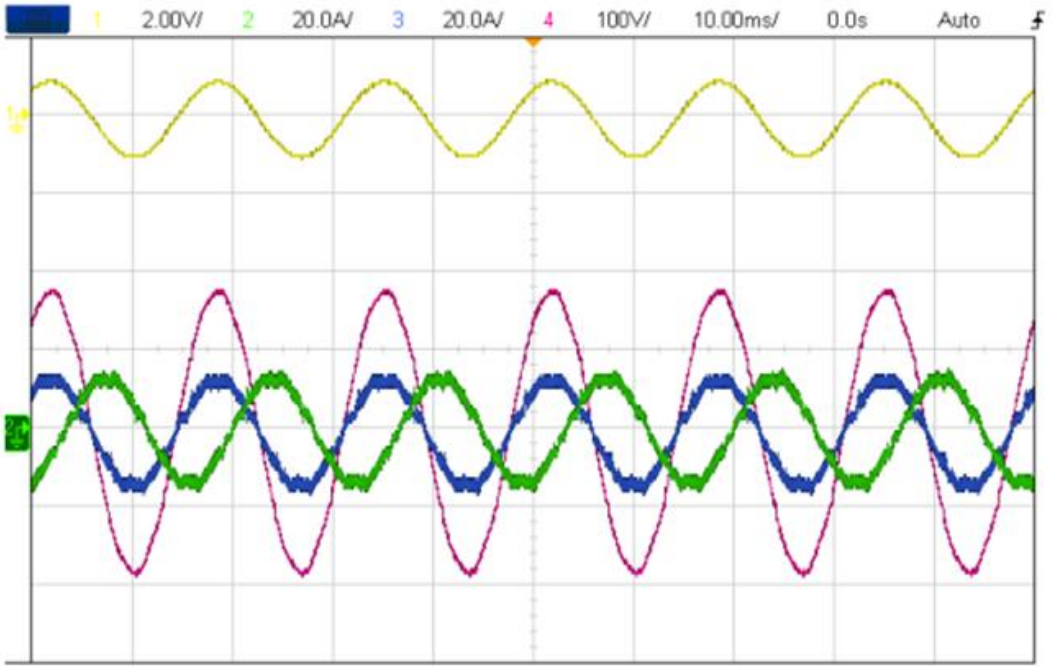
Figura 32. Resultados experimentais para um aumento da disponibilidade de energia no barramento CC.



Fonte: O Próprio Autor.

Em conexão com a rede elétrica, injetando-se aproximadamente 4 kW, verificam-se as formas de onda de interesse e, em regime permanente. As correntes (em azul e verde) possuem distorção harmônica da ordem de 5%, dentro das normas de conexão (Figura 33).

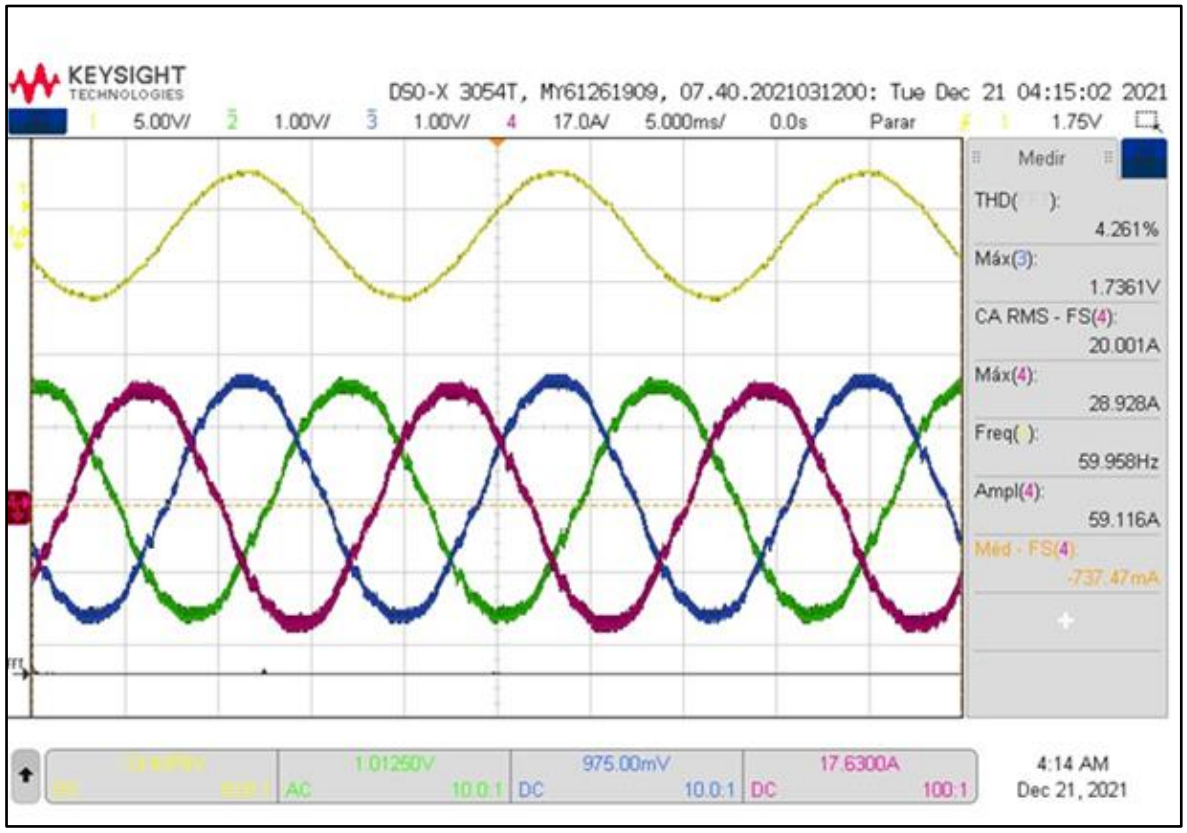
Figura 33. Resultados em malha fechada do conversor e em conexão à rede.



Fonte: O Próprio Autor.

Na Figura 34 pode-se observar as três correntes injetadas na rede com potência próxima a 8 kW. Esta potência foi limitada pelos equipamentos presentes no laboratório de eletrônica de potência do PPGEE à época dos experimentos. A distorção harmônica total (DHT) de corrente é de 4,26%, valor abaixo dos 5% estabelecidos por normativas internacionais, mesmo considerando um valor de potência reduzido em relação à potência nominal do eletroposto (20 kW). Nesta Figura, em amarelo, tem-se a tensão de referência do PLL, em magenta, azul e verde, as correntes da rede.

Figura 34. Correntes Injetadas para uma Potência de 8kW.



Fonte: O Próprio Autor.

A Figura 35 exibe os dados recebidos em um servidor local, enviados pelo ESP32. Os “n” conversores foram simulados utilizando algoritmos computacionais. Desta forma, validou-se o envio de requisições e recebimento de respostas com dados pelo DSP, o envio desses dados ao gateway ESP32, e o envio desses dados à um servidor.

Figura 35. Dados Recebidos em um Servidor Local.

Fonte: O Próprio Autor.

Este trabalho é fruto de um projeto de pesquisa, desenvolvimento e inovação em mobilidade elétrica, intitulado: Desenvolvimento de Sistema Nacional de Recarga Rápida de Bicicletas e Veículos Elétricos para Aplicações V2G (Vehicle-to-grid), fomentado pela ANEEL.

Em colaboração com os demais membros do projeto, foram projetados todos os circuitos de potência, sensoriamento e condicionamento de sinais, além dos algoritmos de controle e funcionamento de um inversor bidirecional conectado à rede elétrica. Conversor este, o principal da microrrede que compõe o eletroposto.

Esta dissertação teve como foco a implementação do firmware do conversor trifásico bidirecional e a comunicação deste com os demais conversores do eletroposto laboratorial.

Para a realização do controle digital, as equações em tempo contínuo foram discretizadas através do método de Tustin (bilinear), pois apresenta boa correspondência entre os sistemas em tempo contínuo e discreto ao se utilizar adequada frequência de amostragem. Após isso, foram derivadas as equações à diferenças de cada bloco controlador.

O microcontrolador utilizado na implementação é o C2000™ Delfino™ F2837xD da Texas Instruments, que possui excelente capacidade de processamento em duas CPUs individuais, além de uma série de recursos para a implementação do controle e modulação de conversores de potência, como unidade de cálculos trigonométricos e registradores para configuração de dead-time.

Foram implementados o algoritmo de sincronização e uma proposta de extrator de sequência positiva, a malha de controle de tensão e de corrente, para, assim, obter os adequados pulsos PWM que ativam os interruptores do conversor.

A viabilidade do firmware foi avaliada com o sistema operando, inicialmente, em baixa potência (1,2 kW) e, com carga resistiva. Na sequência, apresentou estabilidade em no instante de conexão. Adicionalmente, o inversor apresentou boa estabilidade frente ao degrau de potência injetada, de 2 A para 8 A. Também foi observada manutenção da tensão no barramento CC após o degrau de potência, validando o controle de tensão, que mantém a tensão regulada em torno de 450 V.

Ao injetar 4 kW de potência, o inversor apresentou boas formas de onda, com distorção harmônica dentro dos limites estabelecidos pelo IEEE (5%). Ao aumentar a potência injetada para 8kW, a distorção harmônica de corrente foi da ordem de 4,26%, ainda melhor que o resultado anterior.

Quanto à comunicação, foram validados os algoritmos de envio de requisições e recebimento de repostas, pelo DSP. Além disso, o envio dessas informações ao ESP32 e posteriormente à nuvem foi validado através de um servidor local.

O inversor bidirecional conectado à rede apresentou resultado satisfatório em baixas potências, ressalta-se que o conversor foi projetado para operar em 20kW, e devido à limitações dos equipamentos presentes no laboratório só foi validado em potência de até 8 kW. Mesmo em potência reduzida, este apresentou resultados dentro dos limites de qualidade de energia estabelecidos pelo IEEE.

Para futuros trabalhos, prevê-se a integração do funcionamento deste conversor com um algoritmo de controle online, para que a decisão de quanta potência injetar ou receber

venha de um centralizador na nuvem. Também são necessários testes em potências maiores e em condições de exaustão e falhas.

REFERÊNCIAS

[1] LIU, Zhu et al. Monitoring global carbon emissions in 2021. Nature Reviews Earth & Environment, v. 3, n. 4, p. 217-219, 2022.

[2] Ge, M., Friedrich, J., & Vigna, L. (2023, March 9). 4 gráficos para entender as emissões de gases de efeito estufa por país e por setor. WRI Brasil. https://www.wribrasil.org.br/noticias/4-graficos-para-entender-emissoes-de-gases-de-efeito-estufa-por-pais-e-por-setor

[3] OUTLOOK, Annual Energy et al. Energy information administration. Department of Energy, v. 92010, n. 9, p. 1-15, 2010.

[4] AHMAD, Fareed et al. Optimal location of electric vehicle charging station and its impact on distribution network: A review. Energy Reports, v. 8, p. 2314-2333, 2022.

[5] N. Adnan, S. M. Nordin, M. A. B. Bahruddin, and M. Ali, How trust can drive forward the user acceptance to the technology? In-vehicle technology for autonomous vehicle, Transportation Research Part A: Policy and Practice, vol. 118, pp. 819-836, Dec. 2018.

[6] RAJENDRAN, Gowthamraj et al. A comprehensive review on system architecture and international standards for electric vehicle charging stations. Journal of Energy Storage, v. 42, p. 103099, 2021.

[7] H. Takanashi, Y. Sato, Y. Kaneko, S. Abe, T. Yasuda, A large air gap 3 kW wireless

power transfer system for electric vehicles, in: Proceedings of the IEEE Energy

Conversion Congress and Exposition (ECCE), IEEE, 2012, pp. 269–274.

[8] Wang, D.; Locment, F.; Sechilariu, M. Modelling, Simulation, and Management Strategy of an Electric Vehicle Charging Station Based on a DC Microgrid. Appl. Sci. 2020, 10, 2053.

[9] Wang, R.; Sun, Q.; Qin, D.; Li, Y.; Li, X.; Wang, P. Steady-state stability assessment of AC-busbar plug-in electric vehicle charging station with photovoltaic. J. Mod. Power Syst. Clean Energy 2020, 8, 884–894.

[10] JANG, Yohan et al. Grid-Connected Inverter for a PV-Powered Electric Vehicle Charging Station to Enhance the Stability of a Microgrid. Sustainability, v. 13, n. 24, p. 14022, 2021.

[11] Xing, Y.Q.; Jin, J.X.; Wang, Y.L.; Du, B.X.; Wang, S.C. An electric vehicle charging system using an SMES implanted smart grid. IEEE Trans. Appl. Supercond. 2016, 26, 1–4.

[12] Van Der Kam, M.; van Sark, W. Smart charging of electric vehicles with photovoltaic power and vehicle-to-grid technology in a microgrid; a case study. Appl. Energy 2015, 152, 20–30.

[13] Z. Wang and S. Wang, "Grid Power Peak Shaving and Valley Filling Using Vehicle-to-grid Systems", IEEE Transactions on Power Delivery, vol. 28, no. 3, p. 1822-1829, 2013.

[14] S. Jung, M. Yoon, J. Suh, "Optimal Frequency Regulation V2g Control With Dod Of Ev Battery", International Journal of Recent Technology and Engineering, vol. 8, no. 2S6, p. 75-78, 2019.

[15] JANG, Yohan et al. Grid-Connected Inverter for a PV-Powered Electric Vehicle Charging Station to Enhance the Stability of a Microgrid. Sustainability, v. 13, n. 24, p. 14022, 2021.

[16] Nsengiyaremye, J., Bikash, P. C., Begovic, M. M. (2021). Low-cost Communication-assisted Line Protection For Multi-inverter Based Microgrids. IEEE Transactions on Power Delivery, 6(36), 3371-3382. https://doi.org/10.1109/tpwrd.2020.3039176

[17] Song, Z., & Chen, W. (2019). Novel DC–AC inverter based on phase-shift shoot-through controlled dual-active-bridge and high-frequency pulse DC link. IET Power Electronics, 12(14), 3842-3851.

[18] Qian, H., Lai, J.J., Zhang, J., & Yu, W. (2010). High-efficiency bidirectional AC-DC converter for energy storage systems. 2010 IEEE Energy Conversion Congress and Exposition, 3224-3229.

[19] M. H. Rashid, “Eletrônica de Potência, Dispositivos, Circuitos e Aplicações,”

Tradução de L. Abramowicz, 2014.

[20] Hobraiche, J., Vilain, J., Macret, P., Patin, N. (2009). A New Pwm Strategy To Reduce the Inverter Input Current Ripples. IEEE Transactions on Power Electronics, 1(24), 172-180. https://doi.org/10.1109/tpel.2008.2006357

[21] Holmes, D. G., & Lipo, T. A. (2003). Pulse width modulation for power converters: principles and practice (Vol. 18). John Wiley & Sons.

[22] Hart, D. W. (2016). Eletrônica de potência: análise e projetos de circuitos. McGraw Hill Brasil.

[23] Microchip Technology Inc, “Pulse Width Modulation (PWM)”, Developer Help - Skills.

[24] W. Wu, Y. Liu, Y. He, H. S.-H. Chung, M. Liserre, and F. Blaabjerg, “Damping methods for resonances caused by lcl-filter-based current-controlled grid-tied power inverters: An overview,” IEEE Transactions on Industrial Electronics, vol. 64, no. 9, pp. 7402–7413, 2017.

[25] Nise, N. S., & da Silva, F. R. (2002). Engenharia de sistemas de controle (Vol. 3). LTC.

[26] Zhai, G., Norisada, T., Imae, J., & Kobayashi, T. (2012). An extension of generalized bilinear transformation for digital redesign. International Journal of Innovative Computing, Information and Control, 8(6), 4071-4081.

[27] The MathWorks, Inc (n.d.). Continuous-Discrete Conversion Methods. https://www.mathworks.com/help/control/ug/continuous-discrete-conversion-methods.html

[28] TMS320F2837xD Dual-Core Delfino Microcontrollers Technical Reference Manual, Rev. I, Texas Instruments, 2013, rev. 2019.

[29] TMS320F2837xD Microcontroller Workshop Guide and Lab Manual, Rev. 2, Texas Instruments, 2018.

[30] LAUNCHXL-F28379D Overview User's Guide, Rev. I, Texas Instruments, 2016, rev. 2019.

[31] Jia, H. J., & Guo, Z. H. (2010, November). Research on the technology of RS485 over Ethernet. In 2010 International Conference on E-Product E-Service and E-Entertainment (pp. 1-3). IEEE.

[32] Maier, A., Sharp, A., & Vagapov, Y. (2017, September). Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things. In 2017 Internet Technologies and Applications (ITA) (pp. 143-148). IEEE.

[33] Espressif Systems (2023, August 2). ESP32-H2 Technical Reference Manual https://www.espressif.com/en/support/documents/technical-doc.

[34] VOLPATO, A. S. ; SOUZA, M. A. ; BALTA, F. M. ; BATISTA, E. A. ; GODOY, RUBEN ; BRITO, MOACYR A. G. DE . Interleaved Bidirectional DC-AC Converter for Electric Vehicle Charging Station. In: 14th IEEE/IAS International Conference on Industry Applications, 2021, São Paulo. INDUSCON. São Paulo: IEEE, 2021. v. 1. p. 1-6.

[35] R. Teodorescu, M. Liserre e P. Rodriguez, Grid converters for photovoltaic and wind power systems, John Wiley & Sons, 2011.

[36] S. Golestan, M. Monfared, F. D. Freijedo, and J. M. Guerrero, "Design and tuning of a modified power-based pll for single-phase grid-connected power conditioning systems," IEEE Transactions on Power Electronics, vol. 27, no. 8, pp. 3639–3650, 2012.

[37] DE BRITO, MOACYR AURELIANO GOMES; CARVALHO, A. A. ; Godoy, R. B. ; VOLPATO, A. S. ; PEREIRA, L. F. S. C. ; BATISTA, E. A. . A New Positive-Sequence Detector Phase-Locked Loop Algorithm for DC Offset Rejection. In: XVI Congresso Brasileiro de Eletrônica de Potência, 2021, João Pessoa. COBEP 2021. João Pessoa: IEEE, 2021. v. 1. p. 1-5.

[38] DA SILVA, LUCAS G.; DE ANDRADE, NICHOLAS D. ; GODOY, RUBEN B. ; DE BRITO, MOACYR A. G. ; MADDALENA, EMILIO T. . Differential Evolution and Fuzzy-Logic-Based Predictive Algorithm for V2G Charging Stations. Applied Sciences-Basel, v. 13, p. 5921, 2023.

APÊNDICE A – TRABALHOS PUBLICADOS

PEREIRA, L. F. S. C.; CARVALHO, A. A.; PEREZ, J. L.; BATISTA, E. A.; Godoy, R. B.; BRITO, MOACYR A. G. DE. Design of FOPI Controller for the Frequency Fixed Orthogonal Signal Generator (FFOSG) Based PLL through Genetic Algorithm. In: 14th IEEE/IAS International Conference on Industry Applications, 2021, São Paulo. INDUSCON. São Paulo: IEEE, 2021. v. 1. p. 1-7.

DE BRITO, MOACYR AURELIANO GOMES; CARVALHO, A. A.; Godoy, R. B.; VOLPATO, A. S.; PEREIRA, L. F. S. C. ; BATISTA, E. A. . A New Positive-Sequence Detector Phase-Locked Loop Algorithm for DC Offset Rejection. In: XVI Congresso Brasileiro de Eletrônica de Potência, 2021, João Pessoa. COBEP 2021. João Pessoa: IEEE, 2021. v. 1. p. 1-5.

APENDICE B – CONFIGURAÇÃO DO AMBIENTE DE PROGRAMAÇÃO

As figuras A1 e A2 demonstram a criação de um novo projeto apenas com o arquivo main.c.

Figura A1 - Criando um novo projeto no Code Composer Studio.

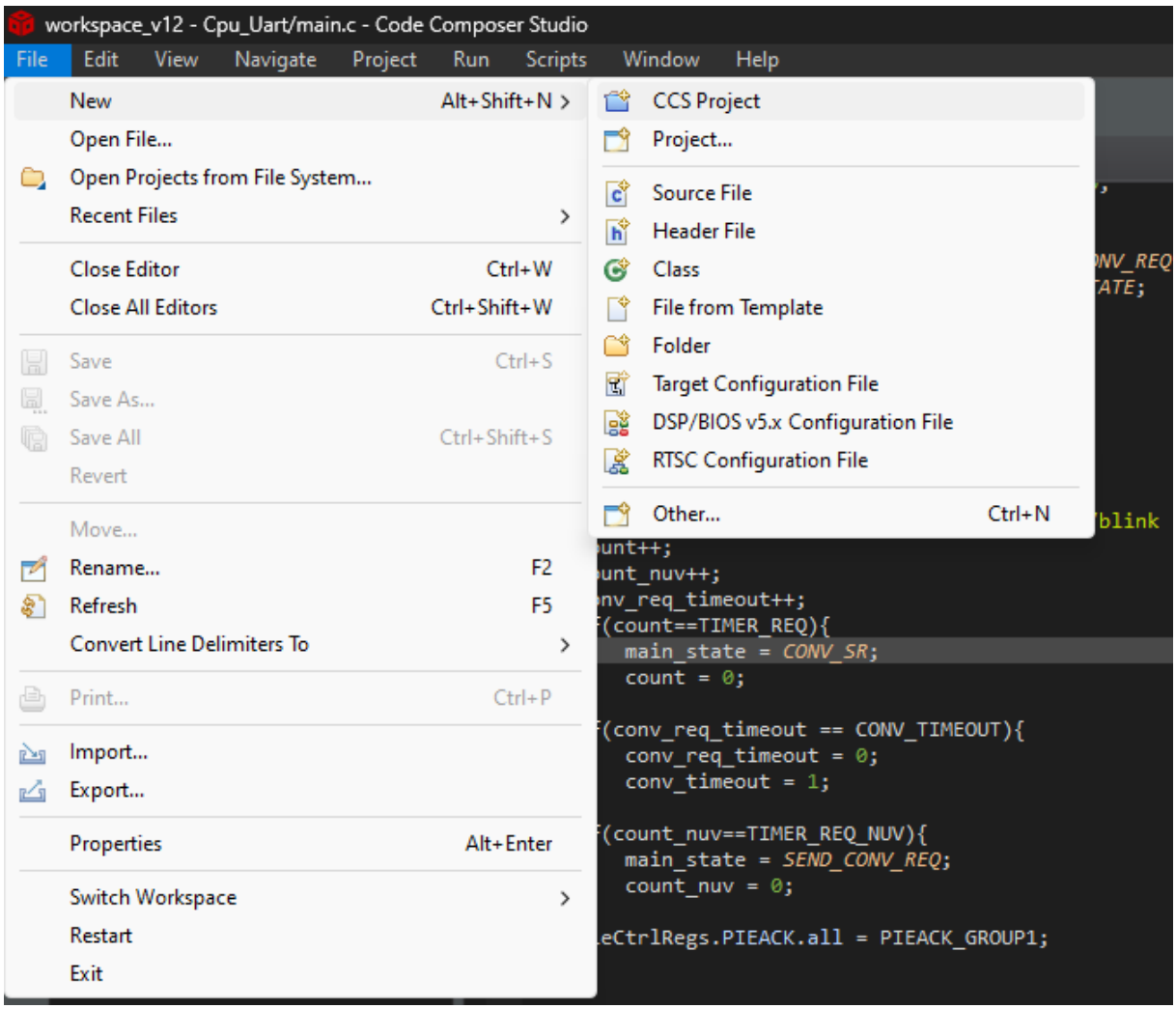
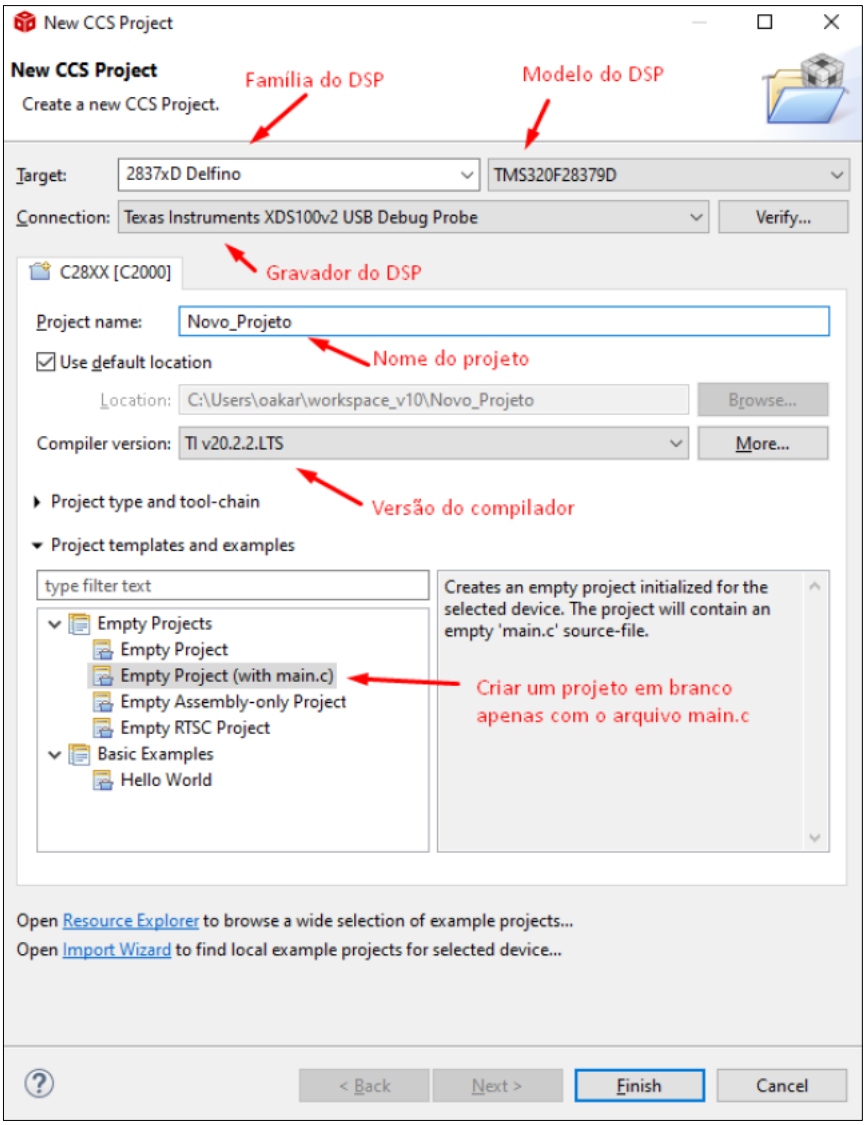


Figura A2 - Configurações de criação do novo projeto.



Após criado, o projeto precisa de algumas configurações, conforme as figuras A3, A4 e A5. Para acessar o menu de configurações, botão direito do mouse em cima do projeto e *properties*.

Figura A3 - Configuração de gravação do código.

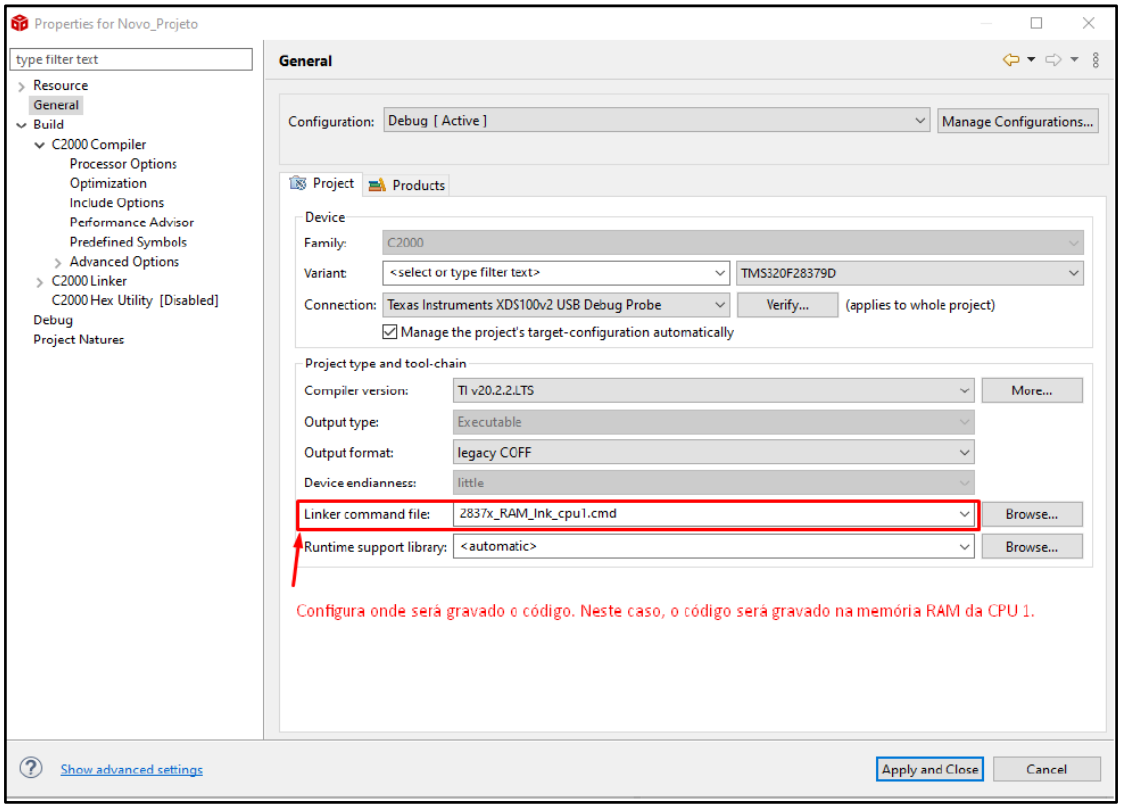


Figura A4 - Inserindo endereços de bibliotecas (1).

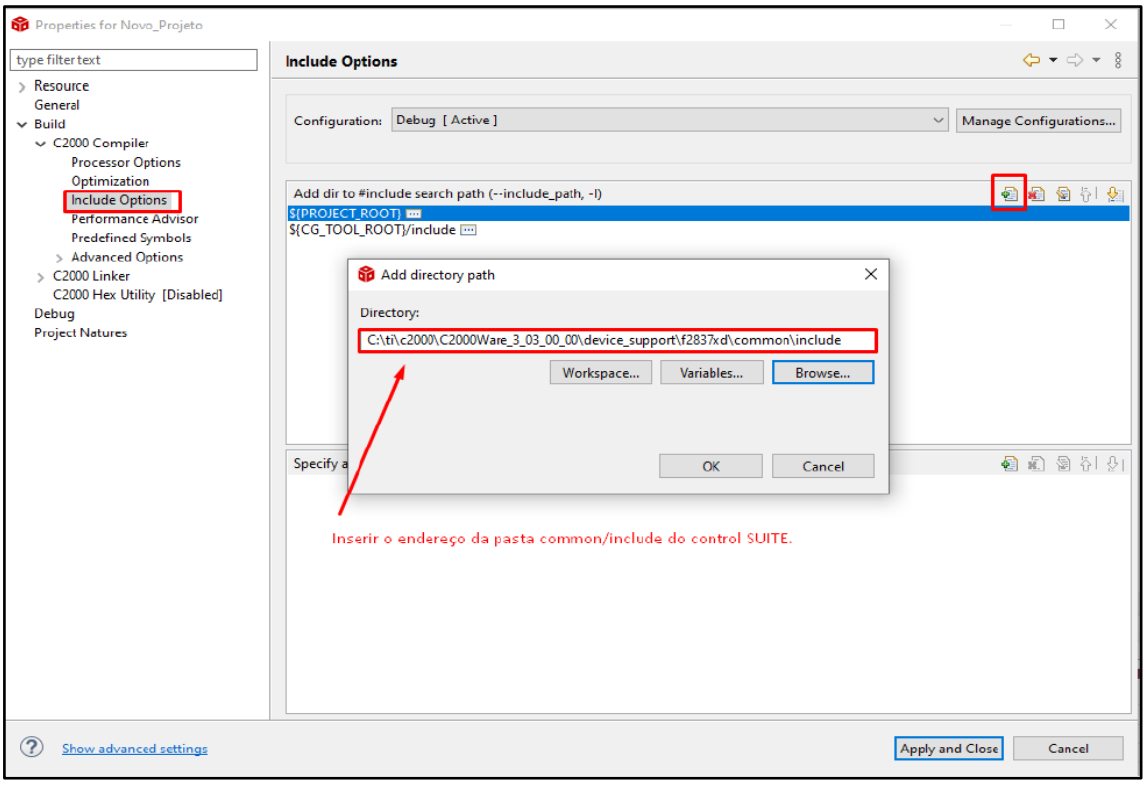
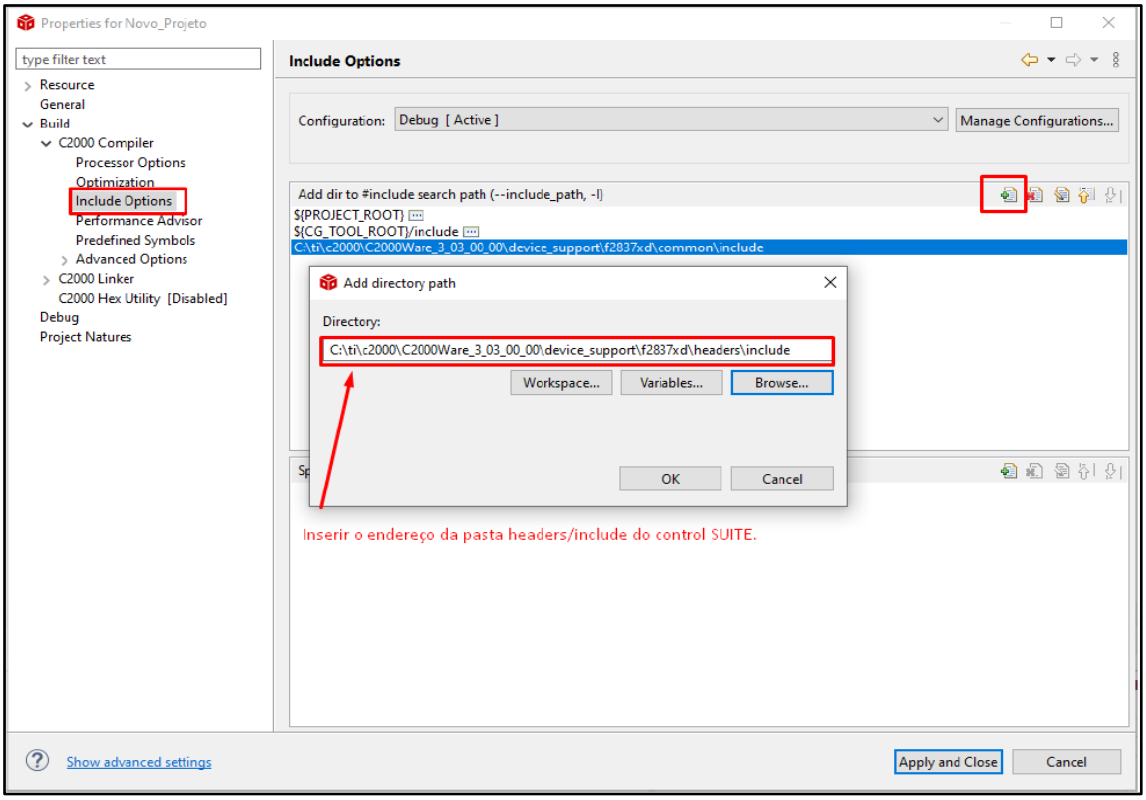


Figura A5 - Inserindo endereço de bibliotecas (2).



Feita a configuração do endereço das bibliotecas, é necessário agora inserir as próprias bibliotecas que serão utilizadas nos códigos. Para isto, clicar com o botão direito do mouse no projeto, e em seguida em Add Files...

Agora navegue até a pasta ...\device_support\f2837xd\common\source e insira os seguintes arquivos:

• F2837xD_CodeStartBranch.asm

• F2837xD_DefaultISR.c

• F2837xD_Gpio.c

• F2837xD_PieCtrl.c

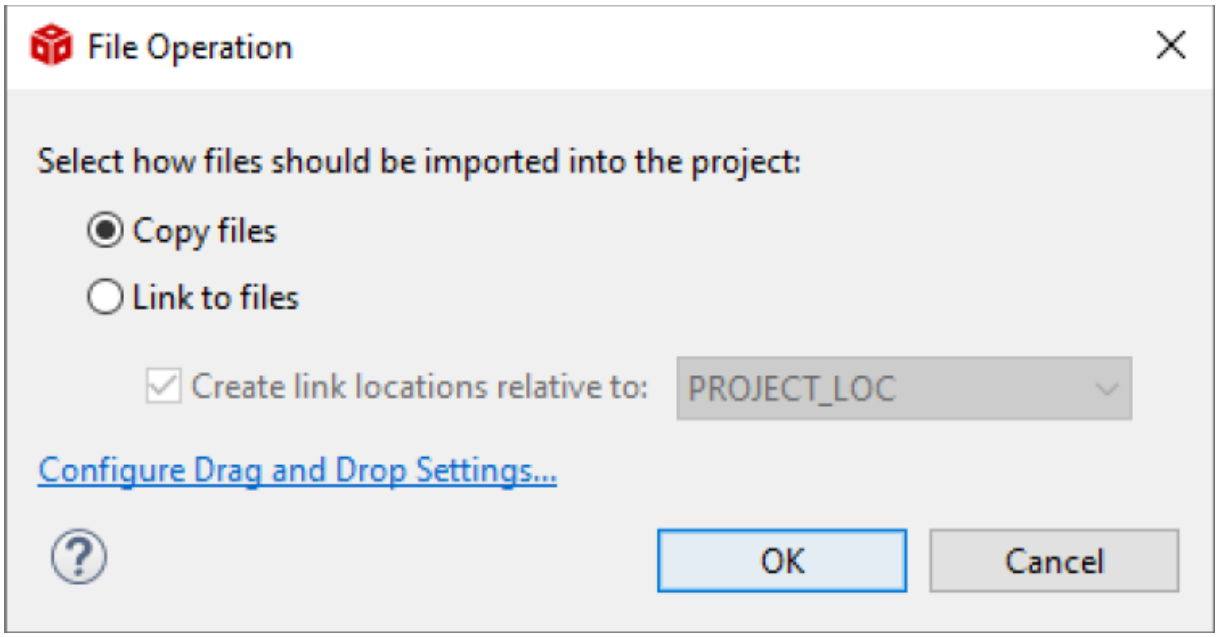
• F2837xD_PieVect.c

• F2837xD_SysCtrl.c

• F2837xD_usDelay.asm

Será exibida uma janela com as opções de copiar as bibliotecas para a pasta do projeto ou apenas direcionar as bibliotecas ao projeto, conforme Figura A6.

Figura A6 - Janela após seleção das bibliotecas.



Insira também o arquivo "F2837xD_GlobalVariableDefs.c", localizado na pasta ...\device_support\f2837xd\headers\source.

Insira o arquivo "F2837xD_Headers_nonBIOS_cpu1.cmd", localizado na pasta "...\device_support\f2837xd\headers\cmd".

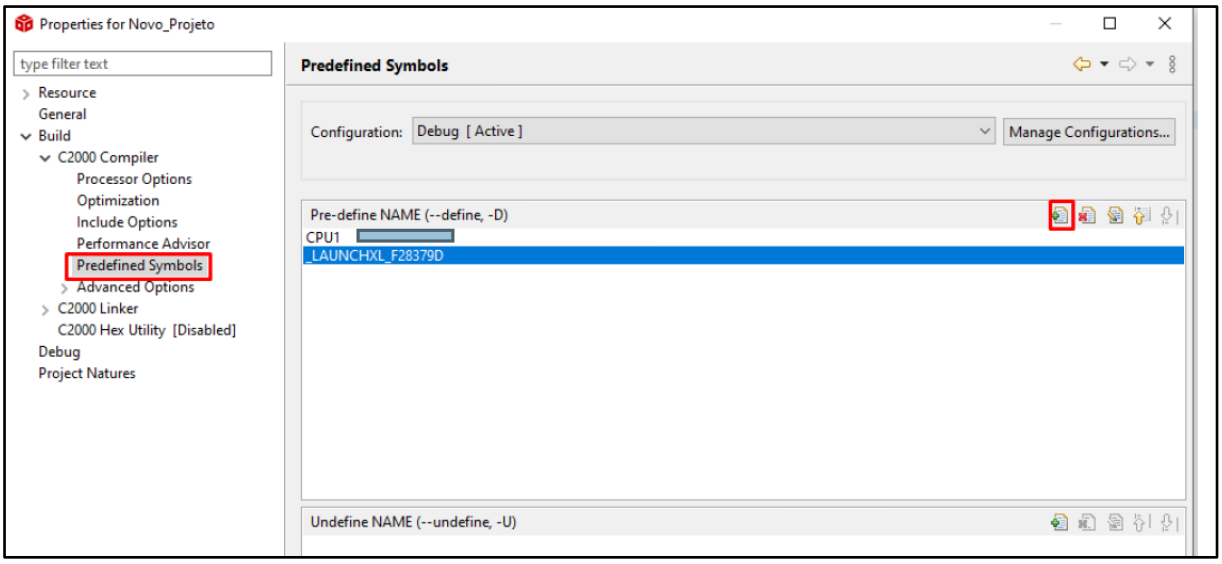
Agora é preciso configurar outros parâmetros no projeto; para isso, abrir as configurações novamente (botão direito do mouse no projeto, properties". Vá até o menu Predefined Symbols e insira, no campo Pre-Define NAME (--define, _D), os seguintes itens:

• CPU1;

• _LAUNCHXL_F28379D

Essas diretivas vão indicar que é a CPU1 que será utilizada e define o CLOCK da CPU1 como 200 MHz, respectivamente. Caso se omita a diretiva _LAUNCHXL_F28379D, o clock da CPU1 será de 100 MHz. A figura A7 mostra a janela para adicionar as diretivas.

Figura A7 - Diretivas de configuração.



APÊNDICE C - CÓDIGOS

```
/*main.c
 * Inversor bidirecional
 */

#include "ABC_DQ0.h"
#include "Peripheral_Setup.h"
#include "math.h"
#include "SPLL_3PH_SRF_F.h"
#include "Voltage_Loop.h"
#include "Current_Loop.h"
#include "DQ0_ABC_F.h"
#include "stdio.h"
#include "main.h"

uint32_t count = 0, index= 0;   //Variáveis de contagem

float32 vrede = 0, phase = 0, ampl = 1;
float32 Va=0, Vb=0, Vc=0, Ia=0, Ib=0, Ic=0, Vcc=0;
float32 dataToCpu2[7]={0,0,0,0,0,0,0};

SPLL_3ph_SRF spll;              //Estrutura do PLL
ABC_DQ0_STRUCT abcdq0;           //Estrutura da transformada abc - dq0
DQ0_ABC_F dq0abc;
VL_VARIABLES vl;                //Estrutura da malha de tensão
CL_VARIABLES cl;                //Estrutura da malha de corrente

float32 *padc2 =  &abcdq0.alpha_pos_v[0]; //ponteiros para o plot
float32 *padc1 =  &Va;           //ponteiros para o plot

#define BUFFER_SIZE 8
static unsigned char buffer_tx[] = {'a', 'b', 'c', 'd', 'e', 'f', '\n', '\r'};
unsigned char buffer_rx[] = {0, 0, 0, 0, 0, 0, 0, 0, 0};

unsigned char buffer_ind = 0, tx_buffer_send = 0, rx_buffer_ready = 0;
float val1 = 0;

__interrupt void isr_cpu_timer0(void);
__interrupt void isr_adc(void);
__interrupt void isr_uart_rx(void);
void send_buffer_tx(void);
void send_buffer_tx(void);


int main(void){
    InitSysCtrl();                          // Initialize System Control:
    DINT;                                   // Disable CPU interrupts
    InitPieCtrl();                                  // Initialize the PIE control
registers to their default state
        IER = 0x0000;                           // Disable CPU interrupts
    IFR = 0x0000;                           // Clear all CPU interrupt flags:
```

```
    InitPieVectTable();                     // Initialize the PIE vector table

    EALLOW;
    CpuSysRegs.PCLKCR0.bit.CPUTIMER0 = 1;   //Habilita timer0 da CPu
    PieVectTable.SCIA_RX_INT = &isr_uart_rx;
    PieVectTable.TIMER0_INT = &isr_cpu_timer0;//configura interrupção de timer
    PieVectTable.ADCA1_INT = &isr_adc;      //Configura interrupção de ADC.

    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;      //Timer 0
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;      //ADC
    PieCtrlRegs.PIEIER9.bit.INTx1 = 1;      // SCIA RX

    EDIS;
    IER |= (M_INT1|M_INT9);

    InitCpuTimers();
    ConfigCpuTimer(&CpuTimer0, 200, 1000000);
    CpuTimer0Regs.TCR.all = 0x4001;

    Setup_GPIO();                               //Chama as funções de configuração
dos periféricos
    Setup_ePWM();
    Setup_ADC();
    Setup_DAC();
    Setup_UART();
    initIPC();


      GpioDataRegs.GPADAT.bit.GPIO24 = 1;

    SPLL_3ph_SRF_init(60,deltaT, &spll);    //Inicializa a função do PLL.
    ABC_DQ0_INIT(&abcdq0);            //Inicializa a função da transformada.
    Voltage_Loop_init(&vl);                   //Inicializa a função da malha de
tensão.
    Current_Loop_init(&cl);

    EINT;                                   // Enable Global interrupt INTM
    ERTM;                                     // Enable Global realtime interrupt
DBGM
    GpioDataRegs.GPBDAT.bit.GPIO34 = 0;     //Desliga o led 34
    GpioDataRegs.GPADAT.bit.GPIO31 = 0;     //liga o led 31
    while(1){

    }
}

//interrupção de tempo
__interrupt void isr_cpu_timer0(void){
    GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;

    for(int i = 0; i < 7; i++) {
            IPCMtoCSetBits(IPC_FLAG0, *((uint32_t*) &dataToCpu2[i]), 0xFFFFFFFF,
0);
            // Aguarde até que a CPU2 leia o dado
            while(IPCGetStatus(IPC_FLAG0) != 0);
        }
```

```
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

}

//interrupção do ADC
__interrupt void isr_adc(void){
    GpioDataRegs.GPADAT.bit.GPIO14 = 1;

    GpioDataRegs.GPADAT.bit.GPIO24  =  (!GpioDataRegs.GPADAT.bit.GPIO16)  ?  1  :
GpioDataRegs.GPADAT.bit.GPIO24;

    Va =  0.0005*((int)AdcaResultRegs.ADCRESULT0 - 2048); //ACAINA2 - PIN J3/29
    Vb =  0.0005*((int)AdcbResultRegs.ADCRESULT2 - 2048); //ACAINB2 - PIN J3/28
    Vc =  0.0005*((int)AdccResultRegs.ADCRESULT4 - 2048); //ACAINB2 - PIN J3/27
    Ia =  0.0005*((int)AdcaResultRegs.ADCRESULT1 - 2048); //ACAINA3 - PIN J3/26
    Ib =  0.0005*((int)AdcbResultRegs.ADCRESULT3 - 2048); //ACAINB3 - PIN J3/25
    Ic =  0.0005*((int)AdccResultRegs.ADCRESULT5 - 2048); //ACAINB3 - PIN J3/24
    Vcc = 0.0005*((int)AdcaResultRegs.ADCRESULT6 - 2048); //ACAINA4 - PIN J7/69

    abcdq0.a_v = Va;
    abcdq0.b_v = Vb;
    abcdq0.c_v = Vc;
    abcdq0.a_i = Ia;
    abcdq0.b_i = Ib;
    abcdq0.c_i = Ic;
    ABC_DQ0_V_FUNC(&abcdq0);
    ABC_DQ0_I_FUNC(&abcdq0);
    PROTECOES(&abcdq0, &spll, &cl, &vl);
    abcdq0.cosine = (float32)__cos(spll.theta[0]);
    abcdq0.sine = (float32)__sin(spll.theta[0]);

    spll.v_q[0] = abcdq0.q_v;

    if(enableControl==1){
        SPLL_3ph_SRF_FUNC(&spll);
        Voltage_Loop_FUNC(&vl);
    }

    cl.relay=vl.relay;
    cl.Id=abcdq0.d_i;
    cl.Iq=abcdq0.q_i;
    cl.Id_ref=vl.yid_ref[0];
    cl.Iq_ref=vl.yiq_ref[0];

    if(enableControl==1){
    Current_Loop_FUNC(&cl);
    }

    dq0abc.cosine=(float32)__cos(spll.theta[0]);
    dq0abc.sine = (float32)__sin(spll.theta[0]);
    dq0abc.d=cl.Md;
    dq0abc.q=cl.Mq;

    if(enableControl==1){
    DQ0_ABC_F_FUNC(&dq0abc);
```

```
    }

    if(enableControl==1){
    EPwm4Regs.CMPA.bit.CMPA    =        (uint16_t)((    dq0abc.a    +    1.0)    *
((float)(EPwm4Regs.TBPRD>>1)) ); //PWM4 A/B PINJ8 80/79
    EPwm5Regs.CMPA.bit.CMPA    =        (uint16_t)((    dq0abc.b    +    1.0)    *
((float)(EPwm5Regs.TBPRD>>1)) ); //PWM5 A/B PINJ8 78/77
    EPwm6Regs.CMPA.bit.CMPA    =        (uint16_t)((    dq0abc.c    +    1.0)    *
((float)(EPwm6Regs.TBPRD>>1)) ); //PWM6 A/B PINJ8 76/75
    }

    dataToCpu2[0]=Vcc;
    dataToCpu2[1]=Va;
    dataToCpu2[2]=Vb;
    dataToCpu2[3]=Vc;
    dataToCpu2[4]=Ia;
    dataToCpu2[5]=Ib;
    dataToCpu2[6]=Ic;

    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;      //clear INT1 flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
    GpioDataRegs.GPADAT.bit.GPIO14 = 0;
}

void initIPC() {
    // Habilitar IPC Interrupt para a CPU1
    IpcRegs.IPCCTL.bit.IPCEN = 1;

    // Limpar todas as flags
    IpcRegs.IPCFLG.all = 0;

    // Limpar todos os pedidos e respostas
    IpcRegs.IPCACK.all = 0xFFFFFFFF;
}
```

CPU1 – ABC_DQ0.c

```
#include "ABC_DQ0.h"
#include "math.h"
#include "SPLL_3PH_SRF_F.h"
#include "Voltage_Loop.h"
#include "Current_Loop.h"


float32 Vmin = 0;
float32 Imin = 0;
float32 Vmax = 240;
float32 Imax = 5;
```

```
float32 enableControl = 1, deltaT = 1/30000;

float32 Vap[3], Vbp[3], Vcp[3], Iap[3], Ibp[3], Icp[3];
float32 Varms[20], Vbrms[20], Vcrms[20], Iarms[20], Ibrms[20], Icrms[2];
int rmsIndex = 0;
int counter = 0;
int errorCode = 0;
int errorCounter = 0;
int errorFlag = 0;

//*********** Structure Init Function ****//
void ABC_DQ0_INIT(ABC_DQ0_STRUCT *v){
                                                                    v->alpha_v=0;
    v->beta_v[0]=0;
    v->beta_v[1]=0;
                                                                       v->sine=0;
                                                                     v->cosine=0;
                                                                        v->z_v=0;
                                                                        v->d_v=0;
                                                                        v->q_v=0;
                                                                        v->a_v=0;
                                                                        v->b_v=0;
                                                                        v->c_v=0;
                                                                   v->qvbeta_v[0]=0;
                                                                   v->qvbeta_v[1]=0;
                                                                v->alpha_pos_v[0]=0;
                                                                v->alpha_pos_v[1]=0;
                                                                 v->beta_pos_v[0]=0;
                                                                 v->beta_pos_v[1]=0;

    v->alpha_i=0;
    v->beta_i[0]=0;
    v->beta_i[1]=0;
    v->z_i=0;
    v->d_i=0;
    v->q_i=0;
    v->a_i=0;
    v->b_i=0;
    v->c_i=0;
    v->qvbeta_i[0]=0;
    v->qvbeta_i[1]=0;
    v->alpha_pos_i[0]=0;
    v->alpha_pos_i[1]=0;
    v->beta_pos_i[0]=0;
    v->beta_pos_i[1]=0;

    Vap[0]=0;
    Vbp[0]=0;
    Vcp[0]=0;
    Iap[0]=0;
    Ibp[0]=0;
    Icp[0]=0;
    Vap[1]=0;
    Vbp[1]=0;
    Vcp[1]=0;
    Iap[1]=0;
```

```
    Ibp[1]=0;
    Icp[1]=0;
    Vap[2]=0;
    Vbp[2]=0;
    Vcp[2]=0;
    Iap[2]=0;
    Ibp[2]=0;
    Icp[2]=0;

}

//*********** Function Definition ********//
void ABC_DQ0_V_FUNC(ABC_DQ0_STRUCT *v){

    v->alpha_v=(0.6666666667)*(v->a_v-0.5*(v->b_v+v->c_v));
    v->beta_v[0]=(0.57735026913)*(v->b_v-v->c_v);
    v->z_v=0.57735026913*(v->a_v+v->b_v+v->c_v);
    v->alpha_v=v->alpha_v*0.5;
    v->beta_v[0]=v->beta_v[0]*0.5;

    v->qvbeta_v[0]=(0.9875*v->qvbeta_v[1])-(0.9875*v->beta_v[0])+(v->beta_v[1]);
    v->alpha_pos_v[0]=(v->alpha_v)-(v->qvbeta_v[0]);
    v->beta_pos_v[0]=(0.9875*v->beta_pos_v[1])-(0.9875*v->alpha_pos_v[0])+(v-
>alpha_pos_v[1]);

    v->q_v=(v->alpha_pos_v[0]*v->cosine)+(v->beta_pos_v[0]*v->sine);
    v->d_v=(v->alpha_pos_v[0]*-v->sine)+(v->beta_pos_v[0]*v->cosine);

    v->beta_v[1]=v->beta_v[0];
    v->qvbeta_v[1]=v->qvbeta_v[0];
    v->alpha_pos_v[1]=v->alpha_pos_v[0];
    v->beta_pos_v[1]=v->beta_pos_v[0];
}

void ABC_DQ0_I_FUNC(ABC_DQ0_STRUCT *i){

    i->alpha_i=(0.6666666667)*(i->a_i-0.5*(i->b_i+i->c_i));
    i->beta_i[0]=(0.57735026913)*(i->b_i-i->c_i);
    i->z_i=0.57735026913*(i->a_i+i->b_i+i->c_i);
    i->alpha_i=i->alpha_i*0.5;
    i->beta_i[0]=i->beta_i[0]*0.5;

    i->qvbeta_i[0]=(0.9875*i->qvbeta_i[1])-(0.9875*i->beta_i[0])+(i->beta_i[1]);
    i->alpha_pos_i[0]=(i->alpha_i)-(i->qvbeta_i[0]);
    i->beta_pos_i[0]=(0.9875*i->beta_pos_i[1])-(0.9875*i->alpha_pos_i[0])+(i-
>alpha_pos_i[1]);

    i->q_i=(i->alpha_pos_i[0]*i->cosine)+(i->beta_pos_i[0]*i->sine);
    i->d_i=(i->alpha_pos_i[0]*-i->sine)+(i->beta_pos_i[0]*i->cosine);

    i->beta_i[1]=i->beta_i[0];
    i->qvbeta_i[1]=i->qvbeta_i[0];
    i->alpha_pos_i[1]=i->alpha_pos_i[0];
    i->beta_pos_i[1]=i->beta_pos_i[0];
}
```

CPU1 – ABC_DQ0.h

```
#ifndef ABC_DQ0
#define ABC_DQ0

#include "SPLL_3PH_SRF_F.h"
#include "Voltage_Loop.h"
#include "Current_Loop.h"

typedef float float32;

extern float32 enableControl;
extern float32 deltaT;

//*********** Structure Definition ********//
typedef struct{
    float32 a_v;
    float32 b_v;
    float32 c_v;
                                                    float32 alpha_v;
                                                  float32 beta_v[2];
                                                       float32 sine;
                                                     float32 cosine;
                                                        float32 d_v;
                                                        float32 q_v;
                                                        float32 z_v;
                                                 float32 qvbeta_v[2];
                                               float32 alpha_pos_v[2];
                                                float32 beta_pos_v[2];
    float32 a_i;
    float32 b_i;
    float32 c_i;
    float32 alpha_i;
    float32 beta_i[2];
    float32 d_i;
    float32 q_i;
    float32 z_i;
    float32 qvbeta_i[2];
    float32 alpha_pos_i[2];
    float32 beta_pos_i[2];
}ABC_DQ0_STRUCT;

//*********** Function Declarations *******//
void ABC_DQ0_INIT(ABC_DQ0_STRUCT *v);
void ABC_DQ0_V_FUNC(ABC_DQ0_STRUCT *v);
void ABC_DQ0_I_FUNC(ABC_DQ0_STRUCT *i);
void   PROTECOES(ABC_DQ0_STRUCT   *p,   SPLL_3ph_SRF   *spll,   CL_VARIABLES   *cl,
VL_VARIABLES *vl);
void disablePWM(void);
void enablePWM(void);

//*********** Macro Definition ***********//
```

```
#define ABC_DQ0_POS_F_MACRO(v)


                                                                    \
v.alpha=(0.6666666667)*(v.a          -          0.5*(v.b          +          v.c));
                                                                    \
                                                 v.beta=(0.57735026913)*(v.b - v.c);

                                                                    \
                                               v.z =0.57735026913*(v.a + v.b + v.c);

                                                                    \
                                                  v.d=v.alpha*v.cos + v.beta*v.sin;

                                                                    \
                                                 v.q=-v.alpha*v.sin + v.beta*v.cos;


#endif /* ABC_DQ0 */
```

CPU1 - DQ0_ABC.c

```
#include "DQ0_ABC_F.h"

//*********** Structure Init Function ****//
void DQ0_ABC_F_init(DQ0_ABC_F *v)
{
                                                                        v->a=0;
                                                                        v->b=0;
                                                                        v->c=0;
                                                                    v->alpha=0;
                                                                     v->beta=0;
                                                                        v->z=0;
                                                                        v->d=0;
                                                                        v->q=0;
}

//*********** Function Definition ********//
void DQ0_ABC_F_FUNC(DQ0_ABC_F *v)
{
                                          v->alpha = v->d*v->cosine - v->q*v->sine;
                                          v->beta  = v->d*v->sine + v->q*v->cosine;
                                                      v->a = v->alpha + 0.5*v->z;
                               v->b = -0.5*v->alpha + 0.8660254*v->beta + 0.5*v->z;
                               v->c = -0.5*v->alpha - 0.8660254*v->beta + 0.5*v->z;
}
```

CPU1 - DQ0-ABC.h

```
#ifndef DQ0_ABC_F_H
#define DQ0_ABC_F_H
```

```
typedef float float32;
//*********** Structure Definition ********//
typedef struct{
                                                                    float32 a;
                                                                    float32 b;
                                                                    float32 c;
                                                                  float32 alpha;
                                                                   float32 beta;
                                                                   float32 sine;
                                                                 float32 cosine;
                                                                    float32 d;
                                                                    float32 q;
                                                                    float32 z;
}DQ0_ABC_F;

//*********** Function Declarations *******//
void DQ0_ABC_F_init(DQ0_ABC_F *v);
void DQ0_ABC_F_FUNC(DQ0_ABC_F *v);

//*********** Macro Definition ***********//
#define DQ0_ABC_F_MACRO(v)


                                                                        \
                                                    v.alpha = v.d*v.cos - v.q*v.sin;

                                                                        \
                                                    v.beta  = v.d*v.sin + v.q*v.cos;

                                                                        \
                                                            v.a = v.alpha + 0.5*v.z;


                                                                        \
                                          v.b = -0.5*v.alpha + 0.8660254*v.beta + 0.5*v.z;\
                                           v.c = -0.5*v.alpha - 0.8660254*v.beta + 0.5*v.z;


#endif /* DQ0_ABC_F_H_ */
```

CPU1 – Peripheral_Setup.c

```
/*
 * Peripheral_Setup.c
 *
 *  Created on: 23 de jul de 2020
 *      Author: waner
 */
#include "Peripheral_Setup.h"
#define TRIG_SEL_ePWM1_SOCA 0x05

void Setup_GPIO(void)
{
    // pg 959 Table Mux, pg 965 Registers and  spruhm8i.pdf - Technical reference
```

```
    EALLOW;
    // LED 31 A, 2
    // LED 34 B, 1
    GpioCtrlRegs.GPAGMUX2.bit.GPIO31 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0;

    GpioCtrlRegs.GPBGMUX1.bit.GPIO34 = 0;
    GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;

    GpioCtrlRegs.GPAPUD.bit.GPIO31 = 1;
    GpioCtrlRegs.GPBPUD.bit.GPIO34 = 1;

    GpioCtrlRegs.GPADIR.bit.GPIO31 = 1;
    GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;

    GpioCtrlRegs.GPBCSEL1.bit.GPIO34 = GPIO_MUX_CPU2;
    GpioCtrlRegs.GPACSEL4.bit.GPIO31 = GPIO_MUX_CPU1;

    //PWM 1A e 1B
    GpioCtrlRegs.GPAGMUX1.bit.GPIO0 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO0 = 1;

    GpioCtrlRegs.GPAGMUX1.bit.GPIO1 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO1 = 1;

    //PWM 4A e 4A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO6 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO6 = 1;

    GpioCtrlRegs.GPAGMUX1.bit.GPIO7 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO7 = 1;

    //PWM 5A e 5A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO8 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO8 = 1;

    GpioCtrlRegs.GPAGMUX1.bit.GPIO9 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO9 = 1;

    //PWM 6A e 6A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO10 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO10 = 1;

    GpioCtrlRegs.GPAGMUX1.bit.GPIO11 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO11 = 1;

    //GPIO Out 14, 15, 24
    GpioCtrlRegs.GPAGMUX1.bit.GPIO14 = 0;
```

```
    GpioCtrlRegs.GPAMUX1.bit.GPIO14 = 0;
    GpioCtrlRegs.GPAPUD.bit.GPIO14 = 1;
    GpioCtrlRegs.GPADIR.bit.GPIO14 = 1;
    GpioCtrlRegs.GPACSEL2.bit.GPIO14 = GPIO_MUX_CPU1CLA;

    GpioCtrlRegs.GPAGMUX1.bit.GPIO15 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO15 = 0;
    GpioCtrlRegs.GPAPUD.bit.GPIO15 = 1;
    GpioCtrlRegs.GPADIR.bit.GPIO15 = 1;
    GpioCtrlRegs.GPACSEL2.bit.GPIO15 = GPIO_MUX_CPU1;

    // Start PWM
    GpioCtrlRegs.GPAGMUX2.bit.GPIO24 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO24 = 0;
    GpioCtrlRegs.GPAPUD.bit.GPIO24 = 1;
    GpioCtrlRegs.GPADIR.bit.GPIO24 = 1;
    GpioCtrlRegs.GPACSEL4.bit.GPIO24 = GPIO_MUX_CPU1;

    //GPIO 16 input
    GpioCtrlRegs.GPAPUD.bit.GPIO16 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO16 = 0;

    //GPIO 42 e 43 USCI-A
    GpioCtrlRegs.GPBGMUX1.bit.GPIO42 = 3;
    GpioCtrlRegs.GPBMUX1.bit.GPIO42 = 3;
    GpioCtrlRegs.GPBPUD.bit.GPIO42 = 1;
    GpioCtrlRegs.GPBDIR.bit.GPIO42 = 1;
    GpioCtrlRegs.GPBCSEL2.bit.GPIO42 = GPIO_MUX_CPU1;

    GpioCtrlRegs.GPBGMUX1.bit.GPIO43 = 3;
    GpioCtrlRegs.GPBMUX1.bit.GPIO43 = 3;
    GpioCtrlRegs.GPBPUD.bit.GPIO43 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO43 = 0;
    GpioCtrlRegs.GPBCSEL2.bit.GPIO43 = GPIO_MUX_CPU1;
    GpioCtrlRegs.GPBQSEL1.bit.GPIO43 = 3;           // Asynch input
    EDIS;

}

void Setup_ePWM(void)
{
    EALLOW;
    CpuSysRegs.PCLKCR2.bit.EPWM1 = 1;               //Habilita o EPWM1.
    CpuSysRegs.PCLKCR2.bit.EPWM4 = 1;               //Habilita o EPWM4.
    CpuSysRegs.PCLKCR2.bit.EPWM5 = 1;               //Habilita o EPWM5.
    CpuSysRegs.PCLKCR2.bit.EPWM6 = 1;               //Habilita o EPWM6.


    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;     //Desabilita o clock de TODOS PWM.

    EPwm1Regs.TBPRD = 1667; //Configura a freq. do PWM. (200.000.000/(2*30.000))
    EPwm1Regs.CMPA.bit.CMPA = EPwm1Regs.TBPRD >> 1; //Configura o Duty Cycle do
PWM (aqui está em 50%)

    EPwm1Regs.TBPHS.bit.TBPHS = 0;    //Configura a fase do PWM (aqui está em 0)
```

```
    EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; //Sincronização do PWM. Pode sincronizar um PWM com outro
    EPwm1Regs.TBCTR = 0x0000;                     // Clear counter
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Configura o modo de contagem do PWM, influencia na freq.
    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;

    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;   // Load registers every ZERO
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO_PRD;
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;   // Load registers every ZERO
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO_PRD;

    EPwm1Regs.AQCTLA.bit.PRD = AQ_NO_ACTION;
    EPwm1Regs.AQCTLA.bit.ZRO = AQ_NO_ACTION;
    EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;          // set actions for EPWM1A
    EPwm1Regs.AQCTLA.bit.CAD = AQ_SET;

    EPwm1Regs.DBCTL.bit.POLSEL = DB_DISABLE;     // Active Hi complementary
    EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
    EPwm1Regs.DBFED.bit.DBFED = 0;              // FED = 20 TBCLKs
    EPwm1Regs.DBRED.bit.DBRED = 0;              // RED = 20 TBCLKs

    //Trigger ADC
    EPwm1Regs.ETSEL.bit.SOCAEN = 1;               // Enable SOC on A group
    EPwm1Regs.ETSEL.bit.SOCASEL = ET_CTR_PRDZERO; // Dispara ADC no topo
    EPwm1Regs.ETPS.bit.SOCAPRD = ET_1ST;          // Trigger on every event

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // ePWM4

    EPwm4Regs.TBPRD = 3333;                       // Set timer period
    //EPwm2Regs.CMPA.bit.CMPA = EPwm2Regs.TBPRD >> 1;
    EPwm4Regs.TBPHS.bit.TBPHS = 0;                // Phase is 0
    EPwm4Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;
    EPwm4Regs.TBCTR = 0x0000;                     // Clear counter
    EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up/down
    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;

    EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;   // Load registers every ZERO
    EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO_PRD;
    EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;   // Load registers every ZERO
    EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO_PRD;

    EPwm4Regs.AQCTLA.bit.PRD = AQ_NO_ACTION;
    EPwm4Regs.AQCTLA.bit.ZRO = AQ_NO_ACTION;
    EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR;          // set actions for EPWM1A
    EPwm4Regs.AQCTLA.bit.CAD = AQ_SET;

    EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;     // Active Hi complementary
    EPwm4Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
    EPwm4Regs.DBFED.bit.DBFED = 100;              // FED = 20 TBCLKs
```

```
    EPwm4Regs.DBRED.bit.DBRED = 100;                // RED = 20 TBCLKs

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~
            // ePWM5
    EPwm5Regs.TBPRD = 3333;                         // Set timer period
    //EPwm2Regs.CMPA.bit.CMPA = EPwm2Regs.TBPRD >> 1;
    EPwm5Regs.TBPHS.bit.TBPHS = 0;                  // Phase is 0
    EPwm5Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;
    EPwm5Regs.TBCTR = 0x0000;                       // Clear counter
    EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count up/down
    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;

    EPwm5Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;     // Load registers every ZERO
    EPwm5Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO_PRD;
    EPwm5Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;     // Load registers every ZERO
    EPwm5Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO_PRD;

    EPwm5Regs.AQCTLA.bit.PRD = AQ_NO_ACTION;
    EPwm5Regs.AQCTLA.bit.ZRO = AQ_NO_ACTION;
    EPwm5Regs.AQCTLA.bit.CAU = AQ_CLEAR;            // set actions for EPWM1A
    EPwm5Regs.AQCTLA.bit.CAD = AQ_SET;

    EPwm5Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;       // Active Hi complementary
    EPwm5Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;  // enable Dead-band module
    EPwm5Regs.DBFED.bit.DBFED = 100;                // FED = 20 TBCLKs
    EPwm5Regs.DBRED.bit.DBRED = 100;                // RED = 20 TBCLKs

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~
    // ePWM5
    EPwm6Regs.TBPRD = 3333;                         // Set timer period
    //EPwm2Regs.CMPA.bit.CMPA = EPwm2Regs.TBPRD >> 1;
    EPwm6Regs.TBPHS.bit.TBPHS = 0;                  // Phase is 0
    EPwm6Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;
    EPwm6Regs.TBCTR = 0x0000;                       // Clear counter
    EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  // Count up/down
    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;

    EPwm6Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;     // Load registers every ZERO
    EPwm6Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO_PRD;
    EPwm6Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;     // Load registers every ZERO
    EPwm6Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO_PRD;

    EPwm6Regs.AQCTLA.bit.PRD = AQ_NO_ACTION;
    EPwm6Regs.AQCTLA.bit.ZRO = AQ_NO_ACTION;
    EPwm6Regs.AQCTLA.bit.CAU = AQ_CLEAR;            // set actions for EPWM1A
    EPwm6Regs.AQCTLA.bit.CAD = AQ_SET;

    EPwm6Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;       // Active Hi complementary
    EPwm6Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;  // enable Dead-band module
    EPwm6Regs.DBFED.bit.DBFED = 100;                // FED = 20 TBCLKs
```

```
    EPwm6Regs.DBRED.bit.DBRED = 100;                // RED = 20 TBCLKs

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~

    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;
}

void Setup_ADC(void)
{
    Uint16 acqps;
    // determine minimum acquisition window (in SYSCLKS) based on resolution
    if (ADC_RESOLUTION_12BIT == AdcaRegs.ADCCTL2.bit.RESOLUTION)
        acqps = 14;                             // 75ns
    else
        // resolution is 16-bit
        acqps = 63;                             // 320ns

    EALLOW;
    CpuSysRegs.PCLKCR13.bit.ADC_A = 1;
    AdcaRegs.ADCCTL2.bit.PRESCALE = 14;          // set ADCCLK divider to /4
    AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS  =  1;  //  Set  pulse  um  ciclo  antes  do
resultado
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;          // power up the ADC

    CpuSysRegs.PCLKCR13.bit.ADC_B = 1;
    AdcbRegs.ADCCTL2.bit.PRESCALE = 14;          // set ADCCLK divider to /4
    AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS  =  1;  //  Set  pulse  um  ciclo  antes  do
resultado
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;          // power up the ADC

    CpuSysRegs.PCLKCR13.bit.ADC_C = 1;
    AdccRegs.ADCCTL2.bit.PRESCALE = 14;          // set ADCCLK divider to /4
    AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
    AdccRegs.ADCCTL1.bit.INTPULSEPOS  =  1;  //  Set  pulse  um  ciclo  antes  do
resultado
    AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;          // power up the ADC

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

    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2;
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = acqps;   //sample window is 15 SYSCLK cycles
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL  =  TRIG_SEL_ePWM1_SOCA;  //trigger  on  ePWM2
SOCA

    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 3;
    AdcaRegs.ADCSOC1CTL.bit.ACQPS = acqps;
    AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = TRIG_SEL_ePWM1_SOCA;

    AdcaRegs.ADCSOC6CTL.bit.CHSEL = 4;
    AdcaRegs.ADCSOC6CTL.bit.ACQPS = acqps;
    AdcaRegs.ADCSOC6CTL.bit.TRIGSEL = TRIG_SEL_ePWM1_SOCA;
```

```
    AdcbRegs.ADCSOC2CTL.bit.CHSEL = 2;
    AdcbRegs.ADCSOC2CTL.bit.ACQPS = acqps;
    AdcbRegs.ADCSOC2CTL.bit.TRIGSEL = TRIG_SEL_ePWM1_SOCA;

    AdcbRegs.ADCSOC3CTL.bit.CHSEL = 3;
    AdcbRegs.ADCSOC3CTL.bit.ACQPS = acqps;
    AdcbRegs.ADCSOC3CTL.bit.TRIGSEL = TRIG_SEL_ePWM1_SOCA;

    AdccRegs.ADCSOC4CTL.bit.CHSEL = 2;
    AdccRegs.ADCSOC4CTL.bit.ACQPS = acqps;
    AdccRegs.ADCSOC4CTL.bit.TRIGSEL = TRIG_SEL_ePWM1_SOCA;

    AdccRegs.ADCSOC5CTL.bit.CHSEL = 3;
    AdccRegs.ADCSOC5CTL.bit.ACQPS = acqps;
    AdccRegs.ADCSOC5CTL.bit.TRIGSEL = TRIG_SEL_ePWM1_SOCA;

    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 0x01;  // 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

    EDIS;
}
```

CPU1 – Peripheral_Setup.H

```
/*
 * Peripheral_Setup.h
 *
 *  Created on: 29 de set de 2020
 *      Author: Artur
 */

#ifndef PERIPHERAL_SETUP_H_
#define PERIPHERAL_SETUP_H_
#include "F28x_Project.h"

void Setup_GPIO (void);
void Setup_ePWM(void);
void Setup_ADC(void);
void Setup_DAC(void);
void Setup_UART(void);

#define CPU_FREQ        200E6
#define LSPCLK_FREQ     CPU_FREQ/4
#define SCI_FREQ        115200
#define SCI_PRD         (LSPCLK_FREQ/(SCI_FREQ*8))-1

#endif /* PERIPHERAL_SETUP_H_ */
```

CPU1 – SPLL_3PH_SRF_F.c

```
#include "SPLL_3PH_SRF_F.h"
float32 freqAdc = 30000;
```

```
//*********** Structure Init Function ****//
void   SPLL_3ph_SRF_init(float32   Grid_freq,   float32   DELTA_T,   SPLL_3ph_SRF
*spll_obj)
{
    spll_obj->v_q[0]=(float32)(0.0);
    spll_obj->v_q[1]=(float32)(0.0);

    spll_obj->ylf[0]=(float32)(0.0);
    spll_obj->ylf[1]=(float32)(0.0);
    spll_obj->ylf[2]=(float32)(0.0);
    spll_obj->ylf[3]=(float32)(0.0);

    spll_obj->ypi[0]=(float32)(0.0);
    spll_obj->ypi[1]=(float32)(0.0);

    spll_obj->fo=(float32)(0.0);
    spll_obj->wo[0]=(float32)(0.0);
    spll_obj->wo[1]=(float32)(0.0);
    spll_obj->fn=(float32)(Grid_freq);

    spll_obj->theta[0]=(float32)(0.0);
    spll_obj->theta[1]=(float32)(0.0);

    // coeficientes das equações de diferenças
    spll_obj->lpf_coeff.B0_pi=(float32)(151);
    spll_obj->lpf_coeff.B1_pi=(float32)(-150.6);
    spll_obj->lpf_coeff.A1_pi=(float32)(1.0);
    spll_obj->lpf_coeff.B0_lf=(float32)(0.005996);
    spll_obj->lpf_coeff.B1_lf=(float32)(0.005996);
    spll_obj->lpf_coeff.A1_lf=(float32)(0.988);

    spll_obj->delta_T=(float32)(1/freqAdc);
}

//*********** Function Definition ********//
void SPLL_3ph_SRF_FUNC(SPLL_3ph_SRF *spll_obj)
{
    //---------------------------------//
    // Loop Filter                     //
    spll_obj->ylf[0]=(spll_obj->lpf_coeff.A1_lf*spll_obj->ylf[1])   +   (spll_obj-
>lpf_coeff.B0_lf*spll_obj->v_q[0])     +     (spll_obj->lpf_coeff.B1_lf*spll_obj-
>v_q[1]);
    spll_obj->ylf[2]=(spll_obj->lpf_coeff.A1_lf*spll_obj->ylf[3])   +   (spll_obj-
>lpf_coeff.B0_lf*spll_obj->ylf[0])     +     (spll_obj->lpf_coeff.B1_lf*spll_obj-
>ylf[1]);


    //---------------------------------//
    // PI                              //
    spll_obj->ypi[0]=(spll_obj->ypi[1])    +    (67.88*spll_obj->ylf[2])    +    (-
67.84*spll_obj->ylf[3]);

    spll_obj->ypi[0]=(spll_obj->ypi[0]>(float32)(62.8))?(float32)(62.8):spll_obj-
>ypi[0];
    spll_obj->ypi[0]=(spll_obj->ypi[0]<(float32)(-62.8))?(float32)(-
62.8):spll_obj->ypi[0];
```

```
    spll_obj->ypi[1]=spll_obj->ypi[0];
    spll_obj->ylf[1]=spll_obj->ylf[0];
    spll_obj->ylf[3]=spll_obj->ylf[2];
    spll_obj->v_q[1]=spll_obj->v_q[0];

    //---------------------------------//
    // VCO                             //
    //---------------------------------//
    spll_obj->wo[0]=spll_obj->ypi[0]+(2*3.1415926*spll_obj->fn);
    spll_obj->fo=spll_obj->wo[0]/(2*3.1415926);

   spll_obj->theta[0]=spll_obj->theta[1]    +    (spll_obj->delta_T/2)*spll_obj-
>wo[0]+(spll_obj->delta_T/2)*spll_obj->wo[1];

    spll_obj->wo[1]=spll_obj->wo[0];

    if(spll_obj->theta[0] > (float32)(2*3.1415926)){
        spll_obj->theta[0]=spll_obj->theta[0] - (float32)(2*3.1415926);
    }
    spll_obj->theta[1]=spll_obj->theta[0];
}
```

CPU1 – SPLL_3PH_SRF_F.h

```
#ifndef SPLL_3ph_SRF_H_
#define SPLL_3ph_SRF_H_
#include "F2837xD_device.h"

//typedef float float32;

//*********** Structure Definition ********//
typedef struct
{
    float32 B1_pi;          //Coeficientes do PI
    float32 B0_pi;
    float32 A1_pi;
    float32 B1_lf;          //Coeficientes do LF
    float32 B0_lf;
    float32 A1_lf;
} SPLL_3ph_SRF_LPF_COEFF;

typedef struct
{
    float32 v_q[2];         //referência q
    float32 ylf[4];         // Saída do filtro
    float32 ypi[2];         //saída do PI
    float32 wo[2];
    float32 fo;             // output frequency of PLL
    float32 fn;             //nominal frequency
    float32 theta[2];       //ângulo da rede
    float32 delta_T;        //inverso da freq. de operação do PLL
    SPLL_3ph_SRF_LPF_COEFF lpf_coeff;
} SPLL_3ph_SRF;
```

```
//*********** Function Declarations *******//
void SPLL_3ph_SRF_init(float32 Grid_freq, float32 DELTA_T, SPLL_3ph_SRF *spll);
void SPLL_3ph_SRF_FUNC(SPLL_3ph_SRF *spll_obj);

//*********** Macro Definition **********//
#define                                              SPLL_3ph_SRF_MACRO(spll_obj)
\
    /*update        v_q[0]        before        calling        the        routine*/
\
    /*                                Loop                                  Filter*/
\
    spll_obj.ylf[0]=spll_obj.ylf[1]  +  (spll_obj.lpf_coeff.B0_pi*spll_obj.v_q[0])
+ (spll_obj.lpf_coeff.B1_pi*spll_obj.v_q[1]);  \
    spll_obj.ylf[1]=spll_obj.ylf[0];
\
    spll_obj.v_q[1]=spll_obj.v_q[0];
\

spll_obj.ylf[0]=(spll_obj.ylf[0]>(float32)(200.0))?(float32)(200.0):spll_obj.ylf[
0];                                                  \
    /*  VCO                                                                    */
\
    spll_obj.fo=spll_obj.fn                    +                   spll_obj.ylf[0];
\
    spll_obj.theta[0]=spll_obj.theta[1]                                          +
((spll_obj.fo*spll_obj.delta_T)*(float32)(2*3.1415926));
\
    if(spll_obj.theta[0]                 >                 (float32)(2*3.1415926))
\
        spll_obj.theta[0]=spll_obj.theta[0]       -       (float32)(2*3.1415926);
\
    spll_obj.theta[1]=spll_obj.theta[0];
\


#endif /* SPLL_3ph_SRF_F_H_ */
```

CPU1 – Voltage_Loop.c

```
#include "Voltage_Loop.h"

int Vbus_nom = 450;        //Tensão desejada no barramento CC
int relayStatus = 0;      //Relé on ou Off
float satInf = 1;          //Saturação inferior para id_ref.
float satSup = 5;          //Saturação superior para id_ref.

void Voltage_Loop_init (VL_VARIABLES*vl)
{
    vl->Vc_bus=(float32)(450);

    vl->Vc_err[0]=(float32)(0.0);
    vl->Vc_err[1]=(float32)(0.0);
```

```
    vl->yid_ref[0]=(float32)(0.0);
    vl->yid_ref[1]=(float32)(0.0);

    vl->yiq_ref[0]=(float32)(0.0);
    vl->yiq_ref[1]=(float32)(0.0);

    vl->relay=(float32)(0.0);

    vl->id_coeff.B1_id=(float32)(-3.421);
    vl->id_coeff.B0_id=(float32)(3.425);
    vl->id_coeff.A1_id=(float32)(1);

}

void Voltage_Loop_FUNC (VL_VARIABLES*vl)
{
    //update Vc_bus[0] before calling the routine
    //---------------------------------//
    // Relay_signal                    //
    //---------------------------------//
     relayStatus=(vl->Vc_bus>=450)?1:0;          //se Vc_bus > 450, relayStatus =
1; se não, relayStatus = 0;
     vl->relay=relayStatus;
    //---------------------------------//
    // id_ref                          //
    //---------------------------------//
    vl->Vc_err[0]=Vbus_nom-(vl->Vc_bus);        //Sinal de erro.
    vl->Vc_err[0]=vl->Vc_err[0]*relayStatus;    //Relé ligado/desligado
    vl->Vc_err[0]=vl->Vc_err[0]*(-1);           //ganho negativo

    vl->yid_ref[0]=(vl->id_coeff.A1_id*vl->yid_ref[1])+(vl->id_coeff.B0_id*vl-
>Vc_err[0])+(vl->id_coeff.B1_id*vl->Vc_err[1]);     //PI
    vl->Vc_err[1]=vl->Vc_err[0];
    vl->yid_ref[1]=vl->yid_ref[0];

    vl->yid_ref[0]=(vl->yid_ref[0]>satSup)?satSup:vl->yid_ref[0]; //Se yid_ref >
satSup, yid_ref = satSup, se não yid_ref=yid_ref;
    vl->yid_ref[0]=(vl->yid_ref[0]<satInf)?satInf:vl->yid_ref[0]; //Se yid_ref <
satInf, yid_ref = satInf, se não yid_ref=yid_ref;
    //---------------------------------//
    // iq_ref                          //
    //---------------------------------//
    vl->yiq_ref[0] = 0;

}
```

CPU1 – Voltage_Loop.h

```
#ifndef Voltage_Loop_H_
#define Voltage_Loop_H_

typedef float float32;

typedef struct
```

```
{
    float32 B1_id;          //Coeficientes do ramo para calcular id_ref
    float32 B0_id;
    float32 A1_id;
} VL_COEFF;

typedef struct
{
    float32 Vc_bus;           //Tensão no barramento CC
    float32  Vc_err[2];           //Erro entre tensão desejada no barramento CC
(Vbus_nom) e a tensão atual no barramento
    float32 yid_ref[2];       //referência id
    float32 yiq_ref[2];       //referência iq
    float32 relay;
    VL_COEFF id_coeff;

} VL_VARIABLES;

void Voltage_Loop_init (VL_VARIABLES*vl_obj);
void Voltage_Loop_FUNC (VL_VARIABLES*vL_obj);
#endif /* Voltage_Loop_H_ */
```

CPU1 – Current_Loop.c

```
#include "Current_Loop.h"


void Current_Loop_init (CL_VARIABLES*cl){
    cl->Iq=(float32)(0.0);
    cl->Iq_err=(float32)(0.0);
    cl->Iq_relay[0]=(float32)(0.0);
    cl->Iq_relay[1]=(float32)(0.0);
    cl->yiq[0]=(float32)(0.0);
    cl->yiq[1]=(float32)(0.0);
    cl->Mq=(float32)(0.0);
    cl->Iq_ref=(float32)(0.0);

    cl->Id=(float32)(0.0);
    cl->Id_err=(float32)(0.0);
    cl->Id_relay[0]=(float32)(0.0);
    cl->Id_relay[1]=(float32)(0.0);
    cl->yid[0]=(float32)(0.0);
    cl->yid[1]=(float32)(0.0);
    cl->Md=(float32)(0.0);
    cl->Id_ref=(float32)(0.0);

    cl->Vbus=(float32)(450);
    cl->relay=(float32)(0.0);

    cl->cl_coeff.A1_iq=(float32)(1);
    cl->cl_coeff.B0_iq=(float32)(0.02559);
    cl->cl_coeff.B1_iq=(float32)(-0.02119);

    cl->cl_coeff.A1_id=(float32)(1);
```

```
    cl->cl_coeff.B0_id=(float32)(0.02559);
    cl->cl_coeff.B1_id=(float32)(-0.02119);

}

void Current_Loop_FUNC(CL_VARIABLES*cl){
    cl->relay=(cl->Vbus>=450)?1:0;

    cl->Iq_err=(cl->Iq_ref)-(cl->Iq);
    cl->Iq_relay[0]=(cl->Iq_err)*(cl->relay);
    cl->yiq[0]=(cl->cl_coeff.A1_iq*cl->yiq[1])+(cl->cl_coeff.B0_iq*cl-
>Iq_relay[0])+(cl->cl_coeff.B1_iq*cl->Iq_relay[1]);
    cl->Mq=(cl->yiq[0]>1)?1:cl->yiq[0];
    cl->Mq=(cl->yiq[0]<-1)?(-1):cl->yiq[0];


    cl->Id_err=(cl->Id_ref)-(cl->Id);
    cl->Iq_relay[0]=(cl->Iq_err)*(cl->relay);
    cl->yid[0]=(cl->cl_coeff.A1_id*cl->yid[1])+(cl->cl_coeff.B0_id*cl-
>Id_relay[0])+(cl->cl_coeff.B1_id*cl->Id_relay[1]);
    cl->Md=(cl->yid[0]>1)?1:cl->yid[0];
    cl->Md=(cl->yid[0]<-1)?(-1):cl->yid[0];
}
```

CPU1 – Current_Loop.h

```
#ifndef Current_Loop_H_
#define Current_Loop_H_

typedef float float32;

typedef struct
{
    float32 B1_iq;
    float32 B0_iq;
    float32 A1_iq;
    float32 B1_id;
    float32 B0_id;
    float32 A1_id;
} CL_COEFF;

typedef struct
{
    float32 Iq;
    float32 Iq_err;
    float32 Iq_relay[2];
    float32 yiq[2];
    float32 Mq;
    float32 Id;
    float32 Id_err;
    float32 Id_relay[2];
    float32 yid[2];
    float32 Md;
    float32 Iq_ref;
    float32 Id_ref;
    float32 relay;
```

```
    float32 Vbus;
    CL_COEFF cl_coeff;

} CL_VARIABLES;

void Current_Loop_init (CL_VARIABLES*vl_obj);
void Current_Loop_FUNC (CL_VARIABLES*vL_obj);
#endif /* Current_Loop_H_ */
```

CPU2 – Main.c

```
#include "peripheral_setup.h"
#include "stdio.h"
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "stdint.h"

typedef enum MAIN_STATE {
    IDLE_STATE,
    CPU1_DATA_SR,
    CONV_SR,
    SEND_CONV_REQ,
}MAIN_STATE;

char main_state = IDLE_STATE;

#define BUFFER_SIZE 15
#define TAM_REQ 6
#define TAM_REQ_NUV 7
#define NUV_REQ_CMD 2
#define CONV_REQ_CMD 1
#define MAX_CONV 2
#define TIMER_REQ 15
#define TIMER_REQ_NUV 30
#define CONV_TIMEOUT 10


__interrupt void isr_cpu_timer0(void);
__interrupt void uartc_rx(void);
__interrupt void uartb_rx(void);

void conv_req (char id);
void conv_req_nuv (char id,char state);
void uartb_tx(char *s, int size_s);
void uartc_tx(char *s, int size_s);
char req_read(char* data, char id);
void initIPC();

uint32_t conv_req_timeout = 0;
uint32_t count = 0, count_nuv=0;

char Vwf[100];
char VA[16];        //VARIÁVEL TENSÃO
float VB = 230.5;  //DADO DE TENSÃO
```

```
unsigned char buffer_tx[] = "FAM13 - Inversor Bidirecional\r\n";
unsigned char buffer_rx[BUFFER_SIZE];
unsigned char buffer_rx_esp[MAX_CONV+2];
char ATRST[]="AT\r\n";

volatile unsigned char buffer_ind = 0, tx_buffer_send = 0, rx_buffer_ready = 0,
rx_esp_buffer_ready = 0, conv_timeout = 0, rxb_ind=0;
float val1 = 0;

char enviar = 0, recebido_inv = 0; //flag da com
char nuv_req[MAX_CONV+2];

int main(void)
{
    InitSysCtrl();                          // Initialize System Control:
        DINT;                                   // Disable CPU interrupts
        InitPieCtrl();                              // Initialize the PIE control
registers to their default state
        IER = 0x0000;                           // Disable CPU interrupts
        IFR = 0x0000;                           // Clear all CPU interrupt flags:
        InitPieVectTable();                         // Initialize the PIE vector
table

        EALLOW;
        CpuSysRegs.PCLKCR0.bit.CPUTIMER0 = 1;

        PieVectTable.SCIC_RX_INT = &uartc_rx;
        PieVectTable.SCIB_RX_INT = &uartb_rx;
        PieVectTable.TIMER0_INT = &isr_cpu_timer0;
        //PieVectTable.IPC1_INT=&isr_IPC1;

        PieCtrlRegs.PIEIER1.bit.INTx7 = 1;      // Timer 0
        PieCtrlRegs.PIEIER9.bit.INTx1 = 1;      // SCIA RX
        PieCtrlRegs.PIEIER9.bit.INTx3 = 1;      // SCIB RX
        PieCtrlRegs.PIEIER8.bit.INTx5 = 1;      // SCIC RX

        EDIS;

        IER |= (M_INT1|M_INT9|M_INT8);

        InitCpuTimers();
        ConfigCpuTimer(&CpuTimer0, 200, 1000000);
        CpuTimer0Regs.TCR.all = 0x4001;

        Setup_GPIO();
        Setup_UART();
        initIPC();

        EINT;                                   // Enable Global interrupt INTM
        ERTM;                                       // Enable Global realtime
interrupt DBGM

        GpioDataRegs.GPADAT.bit.GPIO31 = 1;
        GpioDataRegs.GPBDAT.bit.GPIO34 = 0;
```

```
        memset(nuv_req,0x01,MAX_CONV);

           while(1){
               switch(main_state){
               case IDLE_STATE:
                   break;
               case CPU1_DATA_SR:
                   float receivedData[7];
                       for(int i = 0; i < 7; i++) {
                           while(IPCGetStatus(IPC_FLAG0) == 0);
                           receivedData[i] = *((float*) &(IpcRegs.IPCRECVADDR));
                           IpcRegs.IPCACK.all = IPC_FLAG0;
                       }

                   break;
               case CONV_SR:;
                   int conv_id;
                   for(conv_id = 1; conv_id <= MAX_CONV; conv_id++){
                       conv_req(conv_id);
                       conv_timeout = 0;
                       conv_req_timeout = 0;
                       while(rx_buffer_ready != 1){
                           if(conv_timeout==1){
                               break;
                           }
                       }
                       if(rx_buffer_ready == 1){
                           char data_to_send[BUFFER_SIZE];
                           if(req_read(data_to_send, conv_id))
                           {
                               uartb_tx(data_to_send,BUFFER_SIZE);
                           }
                           rx_buffer_ready = 0;
                       }
                  }
                   if(main_state == CONV_SR){
                       main_state = IDLE_STATE;
                   }
                   break;

               case SEND_CONV_REQ:;
                   if(rx_esp_buffer_ready==1){
                       int conv_id;
                       for(conv_id = 1; conv_id <= MAX_CONV; conv_id++){
                           conv_req_nuv(conv_id, nuv_req[conv_id+1]);
                           DELAY_US(1000000);
                           conv_timeout = 0;
                           conv_req_timeout = 0;
                       }
                       rx_esp_buffer_ready=0;
                       if(main_state == SEND_CONV_REQ){
                           main_state = IDLE_STATE;
                       }
                   }
                   break;
        }
```

```
    }
}

__interrupt void isr_cpu_timer0(void){
    GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;   //blink Led
    count++;
    count_nuv++;
    conv_req_timeout++;
    if(count==TIMER_REQ){
        main_state = CONV_SR;
        count = 0;
    }
    if(conv_req_timeout == CONV_TIMEOUT){
        conv_req_timeout = 0;
        conv_timeout = 1;
    }
    if(count_nuv==TIMER_REQ_NUV){
        main_state = SEND_CONV_REQ;
        count_nuv = 0;
    }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

void conv_req (char id){
    char req_conv[TAM_REQ]= {'[',id, CONV_REQ_CMD,']',0x0d,0x0a};
    uartc_tx(req_conv, TAM_REQ);
}

void conv_req_nuv (char id,char state){
    char req_conv[TAM_REQ_NUV]= {'[',id, NUV_REQ_CMD, state,']',0x0d,0x0a};
    uartc_tx(req_conv, TAM_REQ_NUV);
}

//USANDO UARTA PARA TESTES -> MUDAR PARA UARTB
void uartb_tx(char *s, int size_s){
    int i;
    for(i = 0; i < size_s; i++){
        //while (ScibRegs.SCIFFTX.bit.TXFFST != 0){}
        while(!ScibRegs.SCICTL2.bit.TXRDY){
        }
        ScibRegs.SCITXBUF.all = s[i];
    }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

void esp_req_read(char *s, int size_s){
}

void uartc_tx(char *s, int size_s){
    int i;
    for(i = 0; i < size_s; i++){
        //while (SciaRegs.SCIFFTX.bit.TXFFST != 0){}
        while(!ScicRegs.SCICTL2.bit.TXRDY){
        }
        ScicRegs.SCITXBUF.all = s[i];
    }
```

```
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

char req_read(char* data, char id){
    memcpy(data, buffer_rx, BUFFER_SIZE);
    if (data[1]==id){
        return 1;
    }
    else {return 0;
    }
}

//RECEBE DO ESP
__interrupt void uartb_rx(void){
    buffer_rx_esp[rxb_ind++] = ScibRegs.SCIRXBUF.all;
    if(rxb_ind==MAX_CONV+2){
        memcpy(nuv_req,buffer_rx_esp,MAX_CONV+2);
        rx_esp_buffer_ready = 1;
        rxb_ind=0;
    }
    ScibRegs.SCIFFRX.bit.RXFFOVRCLR = 1;         // Clear Overflow flag
    ScibRegs.SCIFFRX.bit.RXFFINTCLR = 1;            // Clear Interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}

//RECEBE DOS CONVERSORES
__interrupt void uartc_rx(void){
    buffer_rx[buffer_ind++] = ScicRegs.SCIRXBUF.all;  // Read data
    if(buffer_ind==BUFFER_SIZE){
        rx_buffer_ready = 1;
        buffer_ind=0;
    }
    ScicRegs.SCIFFRX.bit.RXFFOVRCLR = 1;         // Clear Overflow flag
    ScicRegs.SCIFFRX.bit.RXFFINTCLR = 1;         // Clear Interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP8;
}

void initIPC() {
    // Habilitar IPC Interrupt para a CPU2
    IpcRegs.IPCCTL.bit.IPCEN = 1;

    // Limpar todas as flags
    IpcRegs.IPCFLG.all = 0;

    // Limpar todos os pedidos e respostas
    IpcRegs.IPCACK.all = 0xFFFFFFFF;
      }
```

CPU2 – Peripheral_setup.c

```
/* peripheral_setup.c
 *
 *  Created on: 15 de set de 2021
 *      Author: oakar
 */
```

```
#include "peripheral_setup.h"

#define TRIG_SEL_ePWM1_SOCA 0x05
#define TRIG_SEL_ePWM8_SOCA 0x05


void Setup_GPIO(void){
    // pg 959 Table Mux, pg 965 Registers and  spruhm8i.pdf - Technical reference
    EALLOW;
    // LED 31 A, 2
    // LED 34 B, 1
    GpioCtrlRegs.GPAGMUX2.bit.GPIO31 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0;

    GpioCtrlRegs.GPBGMUX1.bit.GPIO34 = 0;
    GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;

    GpioCtrlRegs.GPAPUD.bit.GPIO31 = 1;
    GpioCtrlRegs.GPBPUD.bit.GPIO34 = 1;

    GpioCtrlRegs.GPADIR.bit.GPIO31 = 1;
    GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;

    GpioCtrlRegs.GPBCSEL1.bit.GPIO34 = GPIO_MUX_CPU2;
    GpioCtrlRegs.GPACSEL4.bit.GPIO31 = GPIO_MUX_CPU1;

    //GPIO 42 e 43 USCI-A
    GpioCtrlRegs.GPBGMUX1.bit.GPIO42 = 3;
    GpioCtrlRegs.GPBMUX1.bit.GPIO42 = 3;
    GpioCtrlRegs.GPBPUD.bit.GPIO42 = 1;
    GpioCtrlRegs.GPBDIR.bit.GPIO42 = 1;
    GpioCtrlRegs.GPBCSEL2.bit.GPIO42 = GPIO_MUX_CPU1;

    GpioCtrlRegs.GPBGMUX1.bit.GPIO43 = 3;
    GpioCtrlRegs.GPBMUX1.bit.GPIO43 = 3;
    GpioCtrlRegs.GPBPUD.bit.GPIO43 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO43 = 0;
    GpioCtrlRegs.GPBCSEL2.bit.GPIO43 = GPIO_MUX_CPU1;
    GpioCtrlRegs.GPBQSEL1.bit.GPIO43 = 3;

    //GPIO 18(TX) e 19(RX) USCI-B
    GpioCtrlRegs.GPAGMUX2.bit.GPIO18 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 2;
    GpioCtrlRegs.GPAPUD.bit.GPIO18 = 1;
    GpioCtrlRegs.GPADIR.bit.GPIO18 = 1;
    GpioCtrlRegs.GPACSEL3.bit.GPIO18 = GPIO_MUX_CPU1;

    GpioCtrlRegs.GPAGMUX2.bit.GPIO19 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 2;
    GpioCtrlRegs.GPAPUD.bit.GPIO19 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO19 = 0;
    GpioCtrlRegs.GPACSEL3.bit.GPIO19 = GPIO_MUX_CPU1;

    //GPIO 56(TX) e 139(RX) USCI-C
```

```
    GpioCtrlRegs.GPBGMUX2.bit.GPIO56 = 1;
    GpioCtrlRegs.GPBMUX2.bit.GPIO56 = 2;
    GpioCtrlRegs.GPBPUD.bit.GPIO56 = 1;
    GpioCtrlRegs.GPBDIR.bit.GPIO56 = 1;
    GpioCtrlRegs.GPBCSEL4.bit.GPIO56 = GPIO_MUX_CPU1;

    GpioCtrlRegs.GPEGMUX1.bit.GPIO139 = 1;
    GpioCtrlRegs.GPEMUX1.bit.GPIO139 = 2;
    GpioCtrlRegs.GPEPUD.bit.GPIO139 = 0;
    GpioCtrlRegs.GPEDIR.bit.GPIO139 = 0;
    GpioCtrlRegs.GPECSEL2.bit.GPIO139 = GPIO_MUX_CPU1;

    EDIS;
}


void Setup_UART(void){
    // pg 2279 spruhm8i.pdf - Technical reference
    SciaRegs.SCICCR.all = 0x0007;              // 1 stop bit,  No loopback, no
parity,8 char bits,
    SciaRegs.SCICCR.bit.SCICHAR = 0x07;      // SCI character length from one to
eight           bits
    SciaRegs.SCICTL1.all = 0x0003;            // enable TX, RX, internal SCICLK,
Disable RX ERR, SLEEP, TXWAKE
    SciaRegs.SCICTL2.bit.TXINTENA = 0;
    SciaRegs.SCICTL2.bit.RXBKINTENA = 1;
    SciaRegs.SCIHBAUD.all = 0x0000;
    SciaRegs.SCILBAUD.all = SCI_PRD;
    SciaRegs.SCIFFTX.all = 0xC022;
    SciaRegs.SCIFFRX.all = 0x0021;
    SciaRegs.SCIFFCT.all = 0x00;
    SciaRegs.SCICTL1.all = 0x0023;     // Relinquish SCI from Reset
    SciaRegs.SCIFFTX.bit.TXFIFORESET = 1;
    SciaRegs.SCIFFRX.bit.RXFIFORESET = 1;

    ScicRegs.SCICCR.all = 0x0007;
    ScicRegs.SCICCR.bit.SCICHAR = 0x07;
    ScicRegs.SCICTL1.all = 0x0003;
    ScicRegs.SCICTL2.bit.TXINTENA = 0;
    ScicRegs.SCICTL2.bit.RXBKINTENA = 1;
    ScicRegs.SCIHBAUD.all = 0x0000;
    ScicRegs.SCILBAUD.all = SCI_PRD;
    ScicRegs.SCIFFTX.all = 0xC022;
    ScicRegs.SCIFFRX.all = 0x0021;
    ScicRegs.SCIFFCT.all = 0x00;
    ScicRegs.SCICTL1.all = 0x0023;
    ScicRegs.SCIFFTX.bit.TXFIFORESET = 1;
    ScicRegs.SCIFFRX.bit.RXFIFORESET = 1;

    ScibRegs.SCICCR.all = 0x0007;
    ScibRegs.SCICCR.bit.SCICHAR = 0x07;
    ScibRegs.SCICTL1.all = 0x0003;
    ScibRegs.SCICTL2.bit.TXINTENA = 0;
    ScibRegs.SCICTL2.bit.RXBKINTENA = 1;
    ScibRegs.SCIHBAUD.all = 0x0000;
    ScibRegs.SCILBAUD.all = SCI_PRD;
```

```
    ScibRegs.SCIFFTX.all = 0xC022;
    ScibRegs.SCIFFRX.all = 0x0021;
    ScibRegs.SCIFFCT.all = 0x00;
    ScibRegs.SCICTL1.all = 0x0023;
    ScibRegs.SCIFFTX.bit.TXFIFORESET = 1;
    ScibRegs.SCIFFRX.bit.RXFIFORESET = 1;
}
```

CPU2 – Peripheral_Setup.h

```
/*
 * Peripheral_Setup.h
 *
 *  Created on: 23 de jul de 2020
 *      Author: waner
 */

#ifndef PERIPHERAL_SETUP_H_
#define PERIPHERAL_SETUP_H_
#include "F28x_Project.h"

void Setup_GPIO(void);
void Setup_UART(void);

#define CPU_FREQ        200E6
#define LSPCLK_FREQ     CPU_FREQ/4
#define SCI_FREQ        115200
#define SCI_PRD         (LSPCLK_FREQ/(SCI_FREQ*8))-1

#endif /* PERIPHERAL_SETUP_H_ */
```