CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Jordana Caires Carvalho

PROPOSTA DE ESTRUTURA PARA SISTEMA DE CONTROLE E SUPERVISÃO DE UMA BANCADA DE GERAÇÃO DE ENERGIA ELÉTRICA HÍBRIDA DE PEQUENO PORTE UTILIZANDO BEAGLEBONE BLACK E REDE CAN.

Timóteo

2018

Jordana Caires Carvalho

PROPOSTA DE ESTRUTURA PARA SISTEMA DE CONTROLE E SUPERVISÃO DE UMA BANCADA DE GERAÇÃO DE ENERGIA ELÉTRICA HÍBRIDA DE PEQUENO PORTE UTILIZANDO BEAGLEBONE BLACK E REDE CAN.

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Rodrigo Gaiba de Oliveira

Timóteo

2018

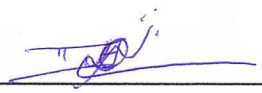
Jordana Caires Carvalho

PROPOSTA DE ESTRUTURA PARA SISTEMA DE CONTROLE E SUPERVISÃO DE UMA BANCADA DE GERAÇÃO DE ENERGIA ELÉTRICA HÍBRIDA DE PEQUENO PORTE UTILIZANDO BEAGLEBONE BLACK E REDE CAN.

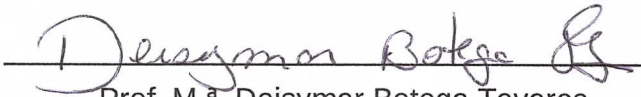
Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, campus Timóteo, como requisito parcial para obtenção do título de Engenheira de Computação.

Trabalho aprovado. Timóteo, 14 de dezembro de 2018:

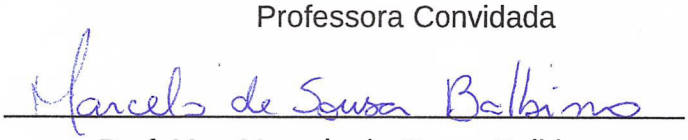
Prof. D.r. Rodrigo Gaiba de Oliveira
Orientador



Prof. D.r. Elder de Oliveira Rodrigues
Professor Convidado



Prof. M.ª. Deisymar Botega Tavares
Professora Convidada



Prof. M.e. Marcelo de Sousa Balbino
Professor Convidado

Timóteo
2018

Dedico ao avanço da educação no Brasil
e ao futuro mais sustentável.

Agradecimentos

Agradeço à Deus pela força e sabedoria nos diversos momentos deste trabalho.

Agradeço aos brasileiros por terem proporcionado a realização deste trabalho em duas instituições públicas de ensino (CEFET-MG Campus Timóteo e UNIFEI Campus Itabira).

Agradeço ao orientador Rodrigo Gaiba de Oliveira (CEFET-MG Campus Timóteo).

Agradeço aos meus colaboradores da UNIFEI Campus Itabira: Geovane Luciano dos Reis, João Lucas da Silva, Luís Eduardo Guedes de Oliveira Soares e Waner Wodson Aparecido Gonçalves Silva.

Agradeço aos meus colaboradores João Marcos Martins da Costa Cota e Leonardo Filipe Rodrigues Ribeiro, que graduaram em Engenharia de Computação pelo CEFET-MG Campus Timóteo.

Agradeço aos professores Juliano de Almeida Monte-Mor e Willian Gomes de Almeida (UNIFEI Campus Itabira).

Agradeço à professora Deisymar Botega Tavares e ao professor Elder de Oliveira Rodrigues (CEFET-MG Campus Timóteo).

Agradeço ao Rafael Mario da Silva, mestrando da UNIFEI, pela paciência com o Code Composer Studio.

Agradeço a todos os pesquisadores mencionados nas referências e professores ao longo de minha vida, por diminuírem minha ignorância.

Agradeço meus pais e o meu irmão.

Agradeço aos estudantes de curso pela contribuição para minha formação pessoal e acadêmica.

Agradeço as pessoas que indiretamente contribuíram para o êxito deste trabalho.

A ordem de agradecimento não desmerece e não engrandece ninguém. Sou muito grata a cada pessoa que me ajudou neste trabalho por todo acolhimento, apoio, compreensão, esforço e paciência.

Cada pessoa que tive a oportunidade de compartilhar um pouco de seu conhecimento deixou uma marca neste trabalho e em mim.

” Se você quer algo que nunca teve,
você precisa estar disposto a fazer algo que nunca fez.”
Thomas Jefferson

Resumo

A geração de energia elétrica por fontes renováveis tem obtido crescimento, principalmente por meio de geração de pequeno porte. O controle e supervisão dessa geração é necessário para que o comando coordenado dos conversores de potência que realizam a interface entre a fonte renovável e a rede elétrica, não tenham a produção independente sem monitoramento e acionamento manual. Notando esse cenário no laboratório de geração de energia elétrica (LGEE) na UNIFEI Campus Itabira, propos-se uma estrutura de um sistema que faça o acoplamento de uma bancada de um sistema de geração de energia elétrica híbrida de pequeno porte, podendo atuar como um modelo a ser seguido para o controle das demais bancadas de geração do LGEE. Mediante conversas, observações e levantamento de alguns requisitos procurou-se entender as necessidades dos usuários do laboratório. Através do dispositivo embarcado Beaglebone Black (BBB), rede CAN, processador digital de sinais (DSP) e um sistema interno e externo criou-se a estruturação, como uma forma de centralizar os anseios requeridos para controle e supervisão. Apesar deste estudo ter um caráter de planejamento e prototipação, uma vez que se faz necessário melhorias, validações e expansão do modelo proposto para as demais bancadas, percebeu-se e relatou-se alguns pontos relevantes para a continuidade do estudo.

Palavras-chave: Beaglebone Black, Rede CAN, Energia Renovável, Sistema de controle.

Abstract

The generation of electric energy by renewable sources has been growing, mainly by means of small generation. The control and supervision of this generation is necessary so that the coordinated control of the power converters that perform the interface between the renewable source and the electric grid, do not have independent production without monitoring and manual actuation. Noting this scenario in the laboratory of electric power generation (LGEE) in the UNIFEI Campus Itabira, a structure of a system was proposed that coupling a bench of a hybrid electric power generation system of small size, being able to act as a model to be followed for the control of the other LGEE generation benches. Through conversations, observations and survey of some requirements, we tried to understand the needs of the users of the laboratory. Through the Beaglebone Black (BBB) embedded device, CAN network, digital signal processor (DSP) and an internal and external system, structuring was created as a way to centralize the requirements for control and supervision. Although this study has a character of planning and prototyping, once it is necessary to improve, validate and expand the model proposed for the other benches, it was noticed and reported some points relevant to the continuity of the study.

Keywords: Beaglebone Black, CAN network, Renewable Energy, Control System.

Lista de ilustrações

Lista de abreviaturas e siglas

BBB	Beaglebone Black
CCS	Code Composer Studio
CLG	Conversor do Lado da Geração
CLR	Conversor do Lado da Rede
DSP	*Digital Signal Processor*
CAN	*Controller Area Network*
CEFET-MG	Centro Federal de Educação Tecnológica de Minas Gerais
GD	Geração Distribuída
LGEE	Laboratório de Geração de Energia Elétrica
PHC	Pequena Central Hídrica
UNIFEI	Universidade Federal de Itajubá
SHGEER	Sistema Híbrido de Geração de Energia Elétrica Renovável Fotovoltaico-Eólico
SO	Sistema Operacional
SSH	*Secure Shell*

Sumário

1 Introdução

A energia elétrica é obtida por diversos meios, podendo ser renováveis ou não. A matriz energética brasileira é constituída em torno de 70% pela geração das hidrelétricas (FILHO, 2017). Este meio renovável é de alto impacto ambiental, se comparada com outras alternativas.

Pela alta utilização desta geração de energia, o sistema elétrico está submetido a "aleatoriedade do recursos hidrológicos, tornando viáveis as usinas termelétricas somente quando os reservatórios devem estar esgotados"(DA SILVA, 2016, tradução própria livre, p.334). Tal situação ocasionou em 2001, uma crise energética em que o racionamento se fez necessário. Diante de tal cenário, o aumento das percentagens da diversificação dos outros meios da matriz é requerido.

Em 2012, através da resolução normativa 482 a ANEEL (Agência Nacional de Energia Elétrica), autorizou o consumidor da concessionária de energia a tornar-se produtor por fonte renovável, como eólica e solar (ANEEL, 2012). Garantiu-se assim, o incentivo ao setor de energia renovável, consequentemente em geração de pequeno porte, e "o uso produtivo da energia poderá estar associado à formas de gestão que possa vir a facilitar o serviço da concessionária nessas áreas remotas"(PINHO et al., 2008, p.17). Nesse sentido, o conceito de geração distribuída (GD) expandiu-se.

A GD conforme PINHO et al. (2008, p.2) explica, trata-se de recursos de pequeno porte de forma integrada ou isolada, que beneficiam consumidores diretos ou indiretos. Ou seja, a geração distribuída conforme o próprio termo, é a produção de energia elétrica por meio de qualquer tipo de equipamento de pequeno porte, que consiga prover um retorno positivo ao seu usuário (forma isolada) ou a quem a recebe, como por exemplo a concessionária (forma integrada).

BARBOSA FILHO (2014) cita que:

> A GD oferece inúmeras vantagens ao setor elétrico, visto que a disposição da unidade de geração próxima a carga permite a diminuição das perdas associadas ao transporte de energia elétrica, além de uma maior diversificação das tecnologias empregadas para produção de energia, e assim sua escolha pode ser realizada em função dos requerimentos específicos da carga ou da disponibilidade dos recursos energéticos locais (RODRIGUES, 2000, apud OLADE, 2011)

Sendo mais específico para o cenário brasileiro:

> No Brasil, as tendências para o incremento da geração distribuída decorrem de diversas causas, como:

> desejo dos consumidores de reduzir o custo do suprimento de energia elétrica e de melhorar a confiabilidade desse suprimento, face ao aumento dos preços aplicados pelas concessionárias e às deficiências das mesmas; em particular;
>
> reestruturação institucional do setor elétrico;
>
> disponibilidade crescente do gás natural para geração, em virtude do aumento da oferta tanto de origem nacional como externa, da construção de gasodutos para transporte e do desenvolvimento das redes de distribuição (OLADE, 2011)
>
> crescente aumento e aperfeiçoamento de tecnologias para aproveitamento de energia a partir de fontes renováveis, em destaque para solar e eólica;
>
> o baixo valor econômico da venda de energia, obtido através de leilões de energia para fontes renováveis;
>
> políticas públicas de incentivo ao mercado de energia solar, que colocou o Brasil em destaque quanto ao aproveitamento da solar térmica;
>
> conscientização dos problemas ambientais, promovendo soluções que tendem a reduzir os impactos ambientais da geração, dentre as quais as que permitem melhor aproveitamento da energia proveniente de combustíveis quer fósseis quer da biomassa; e da diminuição da utilização e construção de grandes redes de distribuição;
>
> progresso da tecnologia eletrônica e consequente redução nos custos de sistemas de controle, de processamento e de transmissão de dados, viabilizando a operação de sistemas elétricos cada vez mais complexos (INEE, 2001).

Entretanto, o custo de implantação de modelos de produção de pequeno porte é alto inicialmente, devido aos componentes não serem de fabricação nacional. Para tentar maximizar a captação de recursos e justificar os investimentos, sistemas híbridos são uma possibilidade que está sendo explorada.

> Define-se esse sistema como sendo aquele que utiliza mais de uma fonte de energia que, dependendo da disponibilidade dos recursos, deve gerar e distribuir energia elétrica, de forma otimizada e com custos mínimos, a uma determinada carga ou a uma rede elétrica, isolada ou conectada a outras redes [barbosa, 2006]. Dada a possibilidade de uma fonte suprir a falta temporária de outra, esse tipo de sistema tem capacidade de operar com menor risco de interrupção.(...). (PINHO et al., 2008, p.171).

1.1 Motivação

Segundo estudos, "não há uma base de dados que aponte o número de sistemas híbridos implantados e em funcionamento no mundo"(PINHO et al., 2008,p.172), e "não há políticas bem definidas de incentivos para o emprego desses sistemas"(PINHO et al., 2008, p.175).

Para suprir esse déficit de informações e fomento, é importante documentar informações para os envolvidos na área, uma vez que não há este controle(PINHO et al., 2008); avançar no domínio das tecnologias; encontrar soluções que diminuam custos; e inovar em diferentes aspectos.

Diante deste pontos, viabiliza-se como uma oportunidade para o meio acadêmico e industrial, aprofundar as pesquisas em energias renováveis, especificamente nas híbridas.

Vislumbrando contribuir com esse setor, o laboratório de Geração de Energia da Universidade Federal de Itajubá (UNIFEI) Campus Itabira desenvolve pesquisa na área de geração híbrida (fotovoltaica, eólica e pequena central hídrica - pch).

A utilização de fontes diversificadas na mesma rede elétrica traz desafios, entre eles o comando coordenado dos conversores de potência, que fazem interface entre a fonte renovável e a rede elétrica. Entretanto, a produção independente sem monitoramento, e com acionamento manual, torna inviável avançar os estudos.

Portanto, é inevitável um meio que organize e apresente esses dados monitorados e acionados, de forma visual e analítica, em conjunto com a automatização dos processos de controle, para as pessoas envolvidas com os projetos no laboratório.

Neste cenário, é necessário o desenvolvimento de um sistema que seja capaz de integrar conversores de potência, usuários do sistema, e a concessionária de energia.

1.2 Objetivos

O objetivo geral desse trabalho é criar uma proposta de um sistema que faça o acoplamento de uma bancada de um sistema de geração de energia elétrica híbrida de pequeno porte.

Objetivam-se mais especificamente:

1. Estudo e uso de uma rede de comunicação industrial (CAN).
2. Estudo e uso de uma plataforma embarcada (Beaglebone Black).
3. Estudo e uso de frameworks que possibilitem a comunicação externa (React) e interna (Flask) dos dados provindos do dispositivo embarcado (Beaglebone Black).
4. Estudo e uso de um hardware que seja capaz de enviar e captar geração de sinais (DSP).
5. Estudo e montagem do circuito integrado entre a rede de comunicação, placa embarcada e dispositivo de sinais elétricos.
6. Estabelecer uma rede de comunicação entre os materiais utilizados, com a estruturação de um protocolo para recebimento e envio de controle.
7. Desenvolver um protótipo de sistema em um computador embarcado que receba os dados, e envie os acionamentos para os componentes conectados em rede.
8. Desenvolver um protótipo de um sistema web do lado do usuário que permita visualizar os componentes de bancada e gráficos das variáveis controladas.

1.3 Estrutura textual

Este trabalho de conclusão de curso é estruturado nos seguintes capítulos: no capítulo 1 é apresentada a introdução, motivação e objetivos gerais e específicos; no capítulo 2 é

apresentado os fundamentos históricos, teóricos e metodológicos; no capítulo 3 é apresentado com mais detalhes os fundamentos teóricos e estado da arte; no capítulo 4 é apresentada a bancada para se elaborar a proposta, a proposta em si com os materiais utilizados; no capítulo 5 são apresentados os resultados; e por fim, no capítulo 6 são apresentadas as conclusões e considerações finais, principais contribuições, limitações e indicadas algumas direções para trabalhos futuros.

2 Materiais e métodos

A Engenharia da Computação é uma área do conhecimento que abrange a maioria das outras áreas, como uma forma de suporte e rapidez nas rotinas desempenhadas, e de forma interna, contempla diferentes estudos.

Este trabalho contempla redes de computadores, engenharia elétrica e desenvolvimento de software. Caracteriza-se no estilo de pesquisa aplicada segundo ANDER-EGG, 1978, p.33 citado por MARCONI E LAKATOS, 2012:

- Caracteriza-se por ser de interesse prático;
- Objetiva a aplicação dos resultados na prática;
- Meta: utilização dos conhecimentos existentes e dos resultados da pesquisa para a solução de problemas reais.

2.1 Materiais

- 1 dispositivo embarcado (Beaglebone Black (BBB)).
- 1 cartão micro SD de 4GB (pode ser de mais quantidade de memória).
- 1 "imagem"(distribuição, instalação) do Debian 9 LXQT para o BBB (disponível no site oficial do dispositivo).
- 2 módulos seriais Can Bus Tja1050 (*Transceivers*).
- 1 *Digital Signal Processor* (DSP) [Processador Digital de Sinal em tradução livre].
- 2 resistores de 120 ohms (pode-se fazer associação em paralelo ou em série com outros resistores até completar 120 ohms).
- 1 *protoboard* ou placa de circuito impresso.
- Cabos (*jumpers*) para conexões entre componentes.
- 1 conversor bidirecional de 5V para 3.3V.
- Linguagem de programação Python.
- Linguagem de programação React JS.
- Terminal de comando do sistema operacional instalado dentro do Beaglebone Black.

2.2 Métodos

Os procedimentos metodológicos deste trabalho foram:

- Procurar e utilizar trabalhos corelatos.
- Procurar e utilizar informações em sites oficiais dos materiais utilizados.
- Procurar e utilizar os conhecimentos transmitidos pelos envolvidos na autoria do trabalho.
- Realizar pequenos testes para validar códigos e componentes.
- Incrementar código e componentes a medida que os testes tinham resultados positivos.

3 Fundamentos teóricos e estado da arte

Habitualmente nas áreas de engenharia, espera-se que as soluções sejam eficientes, robustas, e de baixo custo. Sendo assim, determinou-se que as ferramentas empregadas neste projeto atenderiam tais requisitos na medida do possível. Para que isso acontecesse, um software gratuito ou de desenvolvimento próprio para verificar e visualizar as ações e variáveis da bancada de controle foi necessário. A este tipo de sistema, chamamos de supervisório.

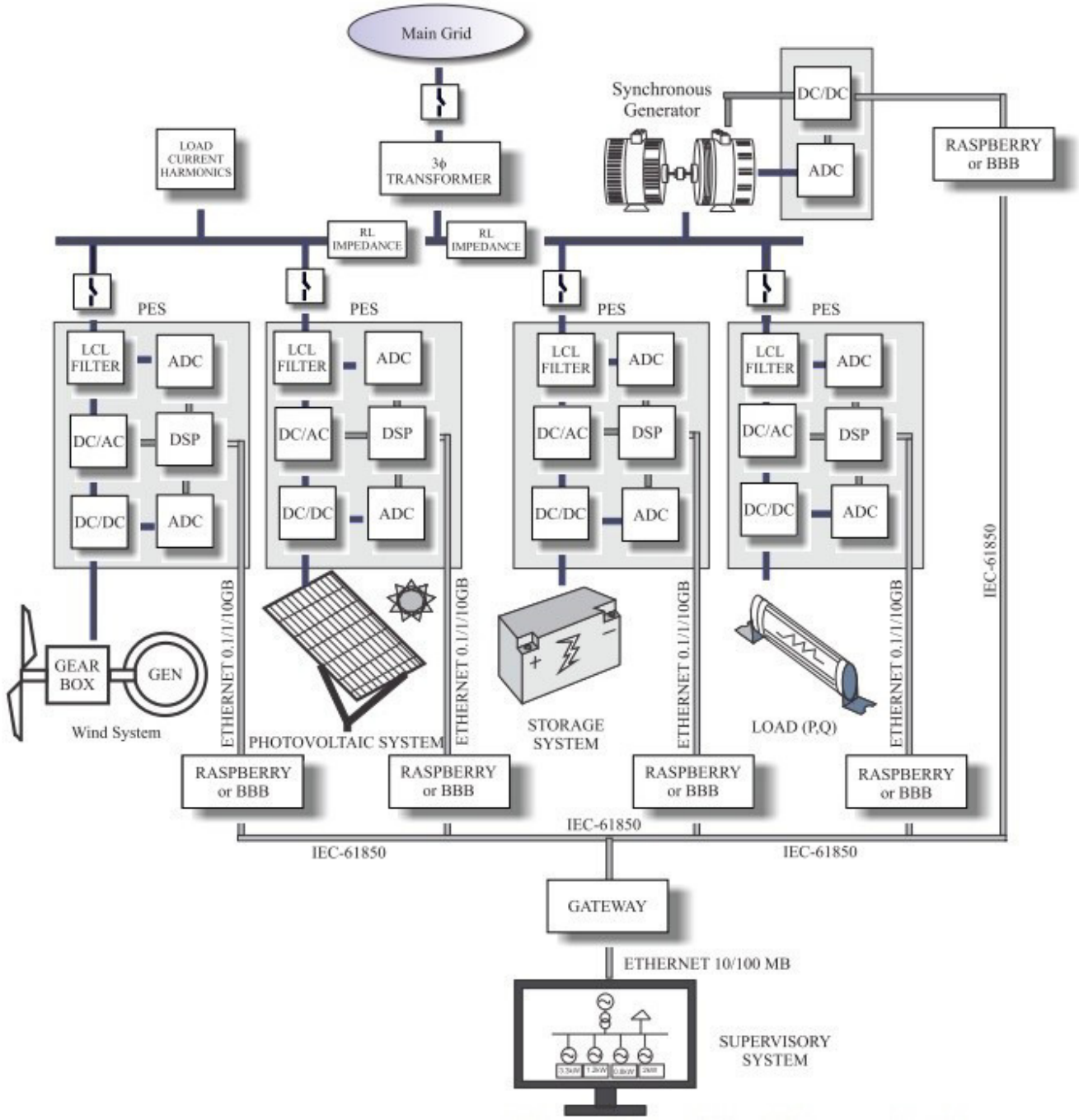
Um supervisório, segundo URZÊDA(2006, p.15), "tem por objetivo ilustrar o comportamento de um processo através de figuras e gráficos, tornando-se assim, uma interface objetiva entre um operador e o processo".

De acordo com URZÊDA(2006, p. 18, apud OGATA, 1997)

> o software supervisório deve ser visto como o conjunto de programas gerados e configurados no software básico de supervisão, implementando as estratégias de controle e supervisão com telas gráficas de interface homem-máquina (IHM) que facilitam a visualização do contexto atual, a aquisição e tratamento de dados do processo e a gerência de relatórios e alarmes.
>
> A visualização de dados consiste na apresentação das informações através de interfaces homem-máquina geralmente acompanhados por animações, de modo a simular a evolução do estado dos dispositivos controlados na instalação dos edifícios. Os sistemas SCADA permitem visualizar os dados recolhidos, além de previsões e tendências de processo com base em valores recolhidos e valores parametrizados pelo operador, bem como com gráficos e relatórios relativos a dados atuais e históricos.
>
> Os alarmes são classificados por níveis de prioridade em função de sua gravidade, sendo reservada a maior prioridade para os alarmes relacionados com questões de segurança. Através da informação proveniente de login, os sistemas SCADA identificam e localizam os operadores, de modo a filtrar e encaminhar os alarmes em função de suas áreas de competência e responsabilidade.
>
> Os sistemas SCADA guardam em arquivos os logs (informação) relativa a todos os alarmes gerados, de modo a permitir que posteriormente se proceda uma análise mais detalhada das circunstâncias que estiveram na sua origem.
>
> Para atingir níveis aceitáveis de tolerância às falhas é usual a existência de informação redundantes na rede de máquinas backup situadas dentro e fora das instalações. Assim sempre que se verifique uma falha num computador o controle das operações seja transferido automaticamente para outro computador que possua todos os seus dados espelhados do computador que estava funcionando até então, para que não haja interrupção significativa.

Em mais alto nível de visualização, a proposta de solução assemelha-se a de RESTREPO-ZAMBRANO et al.(2016), observado pela Figura 1. O supervisório comunica-se com os dispositivos por meio da rede Ethernet, intermediado por Beaglebone Black ou Raspberry, e controlados por módulos DSP.

Figura 1 – Diagrama de componentes elétricos

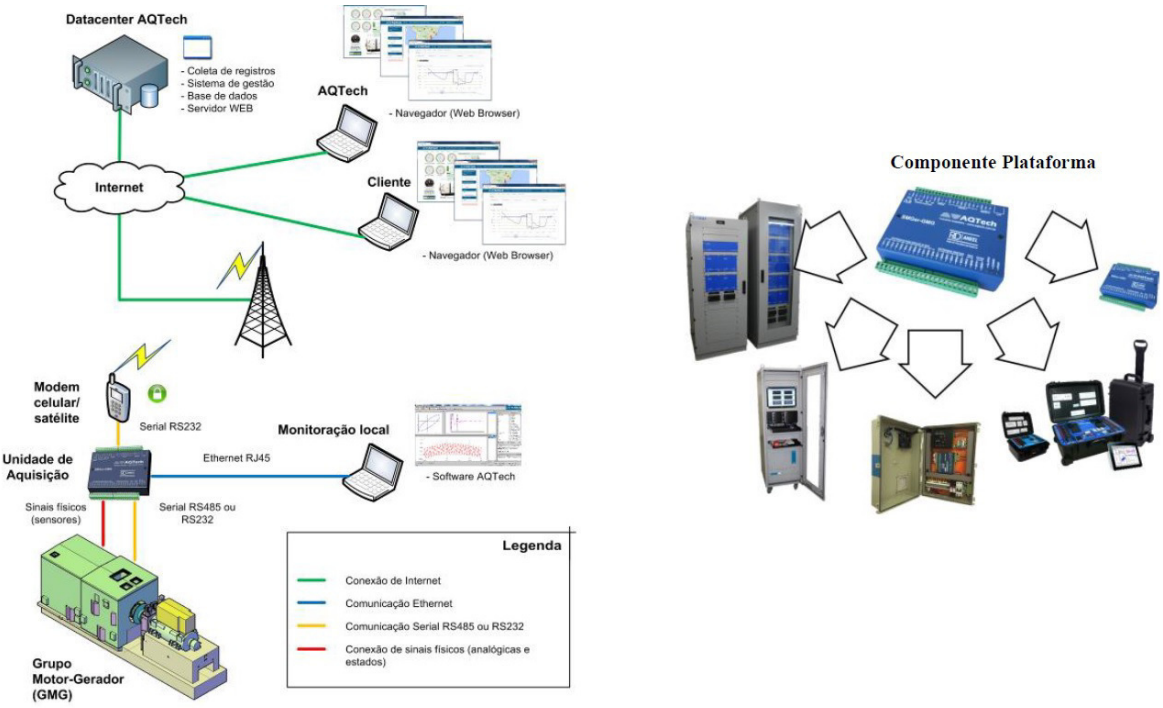


Fonte: RESTREPO-ZAMBRANO et al., 2016, p.5

FERREIRA et al.(2015), também se utiliza de estratégia semelhante em seu artigo intitulado "Monitoramento Online como Ferramenta para Otimização da Manutenção de Geradores: Uma Tecnologia 100% Nacional em Arquitetura Distribuída", desenvolveu-se o monitoramento dos conversores através do hardware SMGer-GMG, e software embarcado utilizando a linguagem de programação C++. A figura 2 ilustra a topologia empregada. Percebe-se pela figura, que o autor utilizou-se dos conceitos de um supervisório ao coletar e registrar os dados por meio do Datacenter da AQTech, realizar a monitoração local com auxílio de software e

sensores.

Figura 2 – Arquitetura do sistema de monitoração remota de GMGs e componente plataforma.



Fonte: FERREIRA(2015).

MANUR et al.(2017) detalha em "Simple electric utility platform: A hardware/software solution for operating emergent microgrids", os elementos para criação de uma plataforma para microgrids com uso do Beaglebone Black com tecnologia móvel (celular/smartphone). A figura 3 representa os processos embutidos no trabalho no HEMApp(a) e o resultado através de um smartphone pelo HEMText(b).

BERTELLI (2015) desenvolveu um "Sistema de Controle e Supervisão da Estação de Processos MPS PA via Plataforma BeagleBone", em que desenvolveu-se

> (...) um sistema de automação para a estação de processos MPS PA, utilizando a plataforma de desenvolvimento BeagleBone Black. O sistema supervisório adotado foi o software livre ScadaBR, com a troca de dados feita via Modbus Serial. A programação do sistema embarcado, para realizar o controle da planta e aquisição e conversão de dados para o formato Modbus, foi feita na linguagem Python. Para o interfaceamento da BeagleBone com a planta, fez-se necessário o projeto de um circuito de conexão.

Após a realização do estudo de todos os componentes requisitados, BERTELLI(2015) elaborou o sistema supervisório adequado a cada situação. A figura 4 ilustra uma das telas produzidas, neste caso, trata-se de uma referência no tanque de 12cm.

FEDRIZZI e SORIA(2015) em "Application of a single-board computer as a lowcost pulse generator", utiliza o Beaglebone Black como um gerador de pulso e utiliza a linguagem de programação Python para o controle da geração, em que

Um computador embarcado open-source BeagleBone Black (BBB) foi implementado como um gerador de pulso totalmente programável de baixo custo. O gerador de pulsos usa o subsistema do BBB de Unidade de Tempo Real Programável (PRU) para alcançar uma determinação temporal resolução de 5 ns, um jitter de RMS de 290 ps e uma estabilidade de base de tempo na ordem de 10 ppm.
Uma estrutura de software baseada em Python também foi desenvolvida para simplificar o uso do gerador de pulso. (FEDRIZZI e SORIA, 2015, tradução própria).

Figura 3 – HEMApp (a) e HEMText(b)

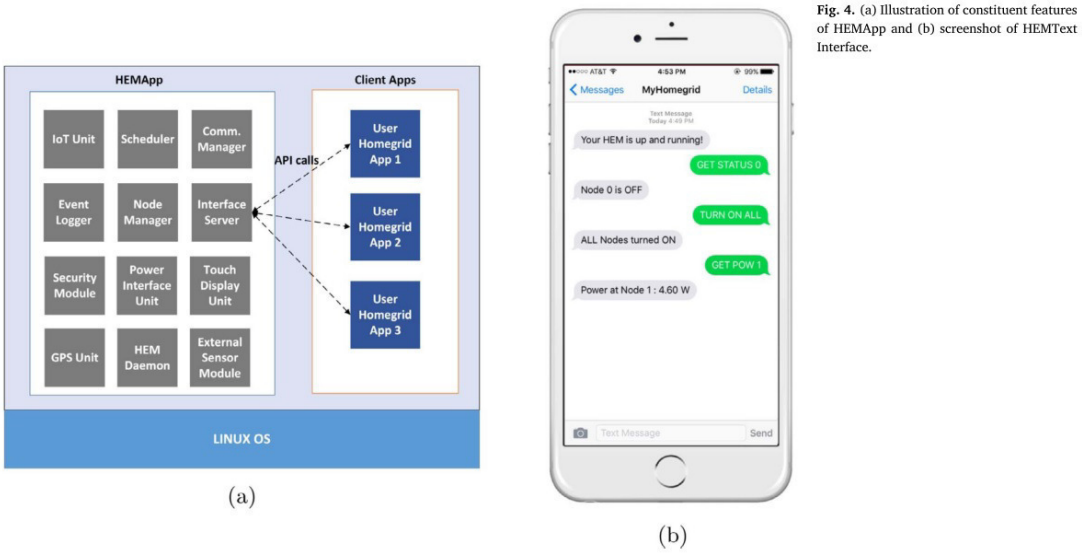
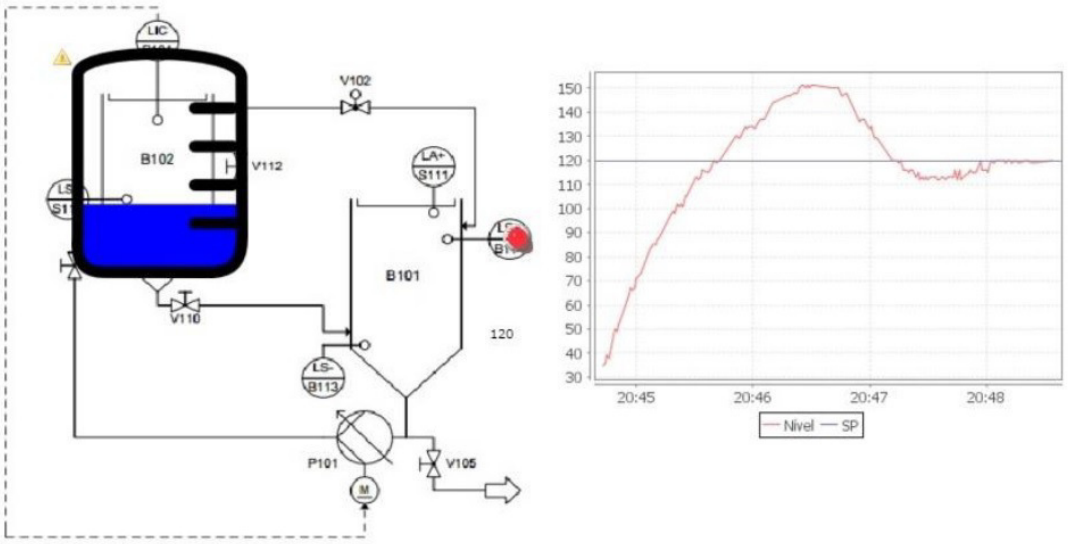


Fig. 4. (a) Illustration of constituent features of HEMApp and (b) screenshot of HEMText Interface.

Fonte: MANUR et al.(2017).

Figura 4 – Sistema supervisório com ScadaBr para referência de 12cm



Fonte: BERTELLI(2015).

4 Caracterização da rede de geração híbrida (UNIFEI), composição embarcada e estrutura formada

Este capítulo descreve quais materiais foram necessários para a elaboração e prática do trabalho. São citadas as suas características e potenciais de uso. Bem como é exposto o cenário em que são/serão utilizados.

4.1 Rede de geração selecionada e suas características

Por meio de reuniões com os docentes responsáveis pelo Laboratório de Geração de Energia Elétrica (LGEE), da UNIFEI Campus Itabira, apresentou-se a situação de que há a produção independente da geração de energia elétrica híbrida (fotovoltaica, eólica e pequena central hídrica - pch). Neste laboratório há cinco bancadas com dispositivos de controle de operações, para os diferentes tipos de geração.

Em cada bancada, as chaves de acionamento de cada DSP são feitas de forma manual. Assim, um usuário tem de operar e controlar unitariamente os dispositivos, não sendo possível atuar de forma paralela, tão pouco remota. No tocante do controle coordenado dos conversores de potência, também presentes nas bancadas, que fazem interface entre a fonte renovável e a rede elétrica, ainda não é presente um sistema de gerenciamento de energia que seja capaz de integrar conversores de potência, usuários do sistema, e a concessionária de energia.

Diante deste cenário foi identificado que o passo a passo das rotinas têm de ser embutida em um sistema de controle (supervisório), de forma simples e amigável, permitindo ao usuário do sistema apenas acessar e clicar nas opções de ações disponíveis, sem ter conhecimento prévio dos processos que devem ser desempenhados previamente. Além disso, espera-se que este supervisório informe sobre as condições dos mecanismos, apresentando em tempo real seus valores, e possíveis erros.

Para que haja a construção e correto funcionamento do futuro supervisório a ser criado, percebeu-se por parte dos envolvidos que, a comunicação entre as bancadas teria de ser realizada. Entretanto, como processo inicial, haveria de ter antes, uma estrutura para automatizar a rotina usada para acionar os componentes presentes, conforme figura 5.

A fim de limitar este trabalho, o objetivo é criar uma proposta de uma estrutura do sistema de controle que faça o acoplamento da bancada do sistema de geração de energia elétrica híbrida de pequeno porte, como um modelo a ser seguido para o controle das demais bancadas posteriormente.

A bancada eleita do sistema híbrido de pequeno porte (geração eólica e solar), foi resultado do colaborador Geovane Luciano dos Reis, como trabalho de mestrado ("Projeto e construção de um conversor monofásico em ponte h multicelular entrelaçado para geração fotovoltaica e eólica de pequeno porte"). Dos Reis afirma que

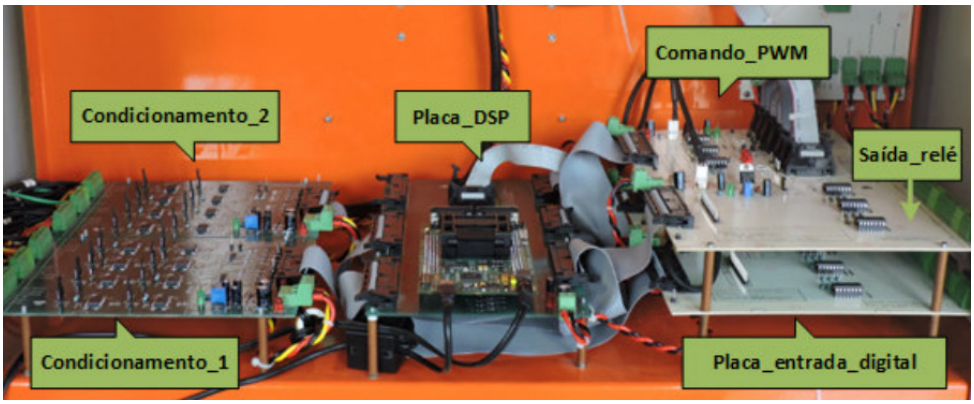
> O sistema híbrido fotovoltaico-eólico tem maior confiabilidade para fornecer energia de forma ininterrupta do que uma dessas fontes de forma individual. Com base no estudo prévio realizado, optou-se pela construção do conversor monofásico híbrido em ponte H multicelular entrelaçado para geração fotovoltaica e eólica de pequeno porte(...). Este pode ser uma opção para levar energia elétrica em localidades remotas na operação (Off-grid), restando apenas o armazenamento em baterias para períodos de intermitência da geração das fontes renováveis ou para aplicação em sistemas (On-grid) já consolidados. (DOS REIS, 2017, p.17).

Após estudada essa confiabilidade, o autor da dissertação configurou a montagem conforme a figura 6, "no telhado do quarto andar do prédio Nikola Tesla"(DOS REIS, 2017, p.24). Estão representados os 40 panéis fotovoltáicos de 255 Wp da empresa Yingli solar, divididos em dois *arrays* com 20 unidades cada, e uma turbina eólica de eixo vertical Razec 266, agregados as três etapas do processo.

O Sistema Híbrido de Geração de Energia Elétrica Renovável Fotovoltaico-Eólico (SHGEER), é composto por três etapas que podem ser observadas na figura 7 e alocadas na bancada representada pela figura 8 em que:

> a primeira é o Conversor do Lado da Geração (CLG) que é composto de três conversores do tipo Boost que são responsáveis por extrair a máxima potência dos dois arranjos de painéis fotovoltaicos e também do gerador síncrono de imã permanente da turbina eólica e injetá-la no Link-CC. A segunda é o Link-CC que é responsável pela interface entre os conversores do lado da geração que entregam corrente contínua para o mesmo e o Conversor do Lado da Rede (CLR) que representa a terceira etapa sendo composto de um inversor monofásico de quatro braços entrelaçados através de indutores acoplados denominados.(DOS REIS, 2017, p.23).

Figura 5 – Módulo de Controle do SHGEER em que está presente o DSP para esta bancada.

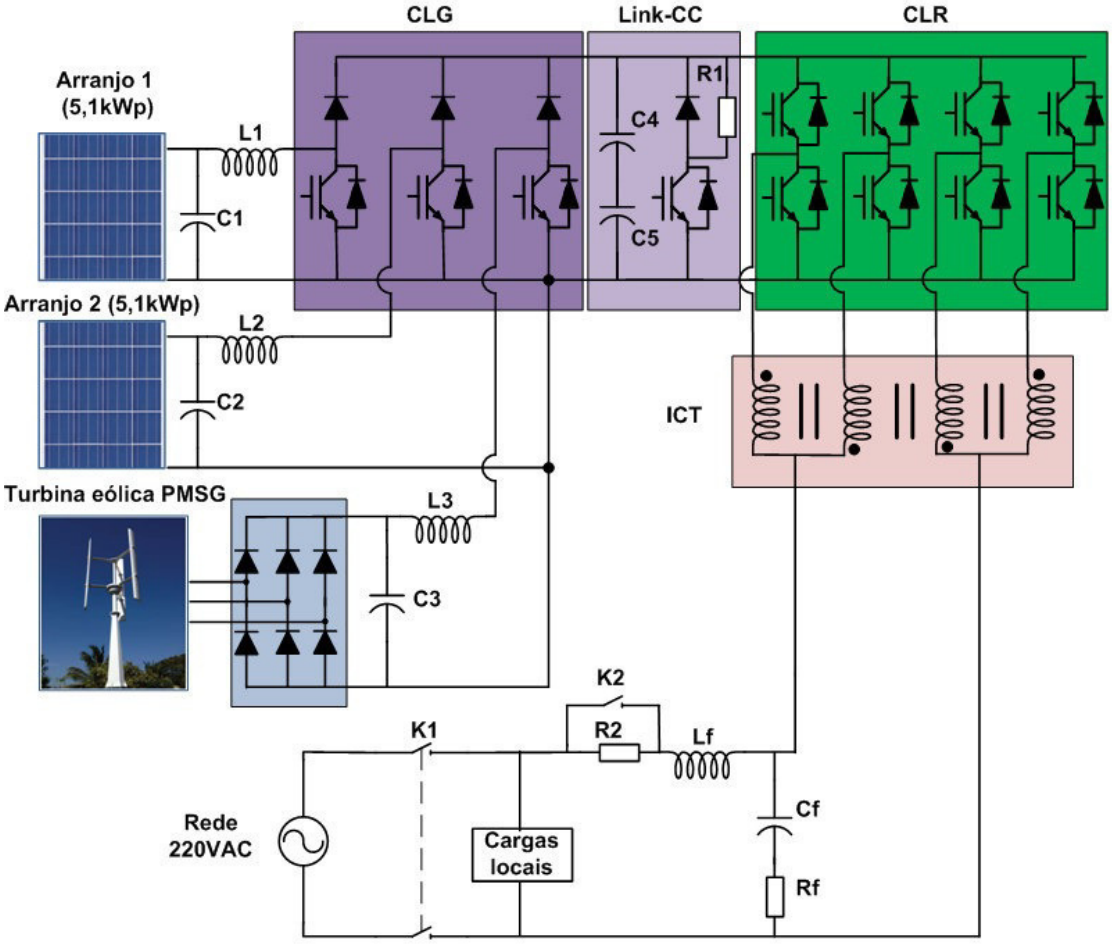


Fonte: DOS REIS, 2017, p.29.

Figura 6 – Painéis solares e turbina eólica de pequeno porte.

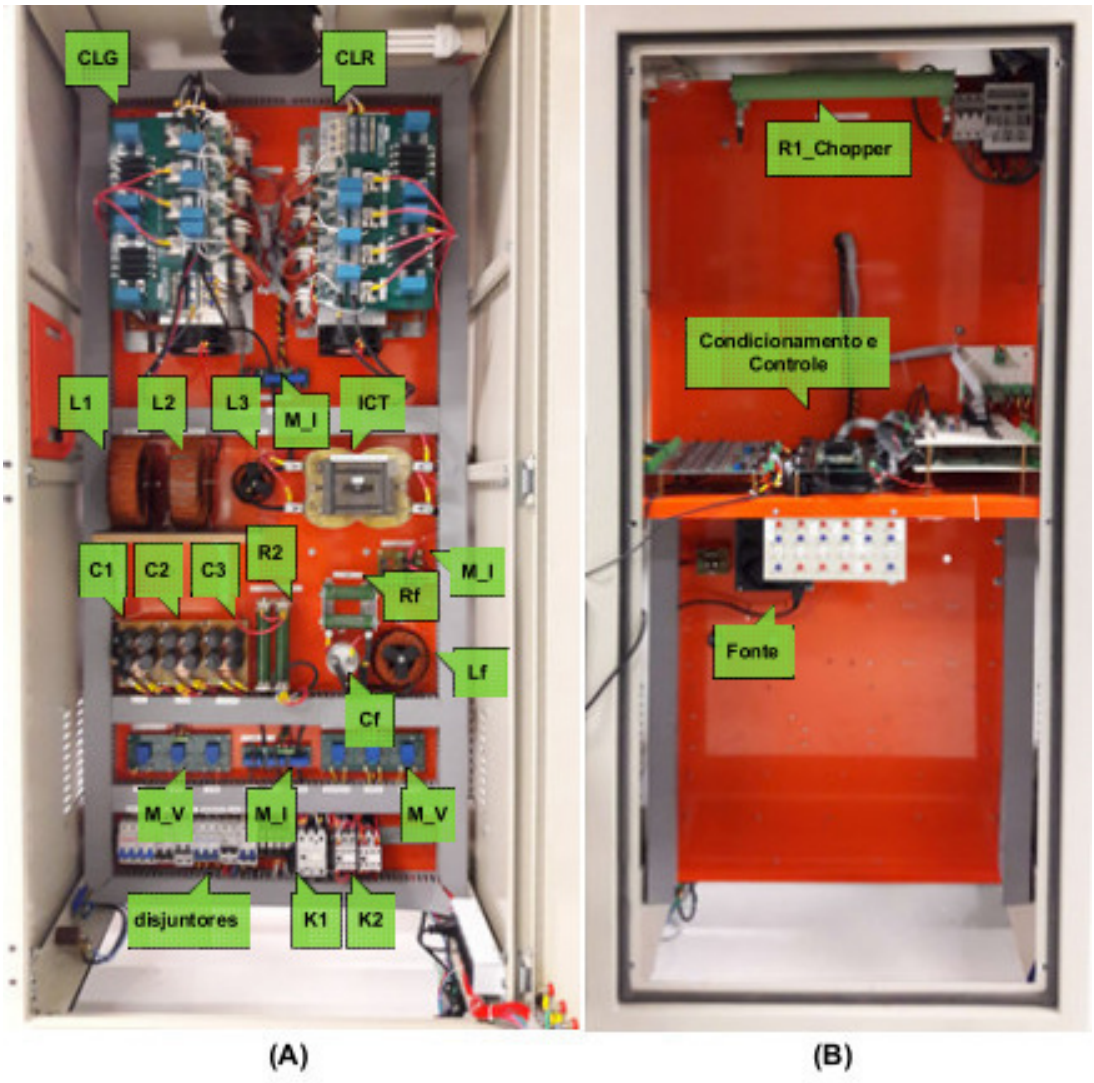
Fonte: DOS REIS, 2017, p.25.

Figura 7 – Diagrama de componentes elétricos



Fonte: DOS REIS, 2017, p.23.

Figura 8 – (A) Parte de potência e (B) Controle do painel do SHGEER.

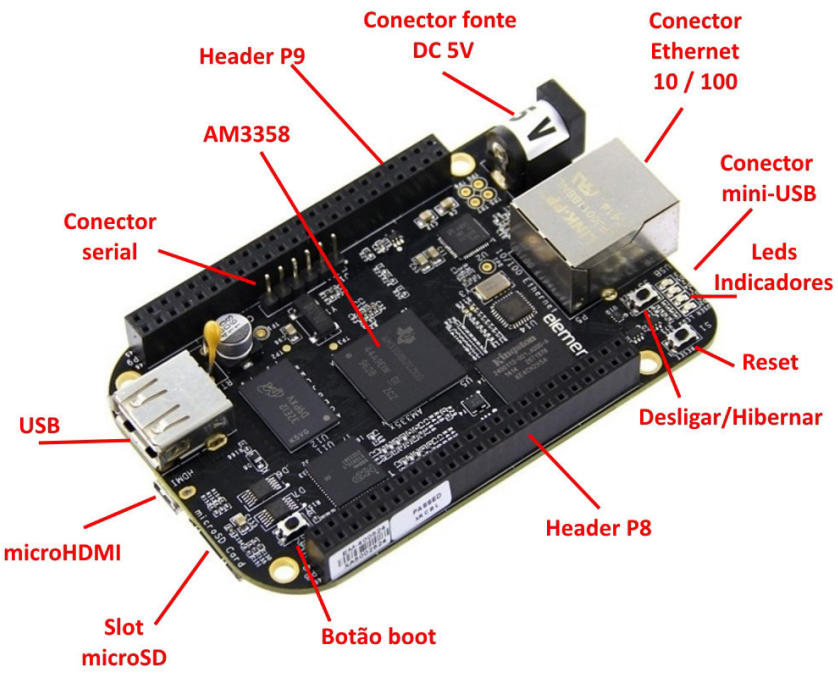


Fonte: DOS REIS, 2017, p.28.

4.2 Composição embarcada utilizada

Para criar a proposta de solução, houve uma estratégia que consistiu no levantamento dos recursos já possuídos, bem como o trabalho feito pelos docentes e terceiros no LGEE. O Beaglebone Black (BBB), foi disponibilizado pela UNIFEI Campus Itabira, sendo "um computador embarcado de baixo custo, com tamanho de um cartão de crédito baseado no sistema TI Sitara AM335x ARM Cortex-A8"(FEDRIZZI e SORIA, 2015, p.2, tradução própria). Possui interfaceamento direto "com até 26 pinos de entrada e saída que podem ser usados para fins de controle. Cada pino de saída pode ser alternado em um ciclo de clock"(FEDRIZZI e SORIA, 2015, p.2, tradução própria).

Figura 9 – Beagle Bone Black



Fonte: THOMSEN(2014).

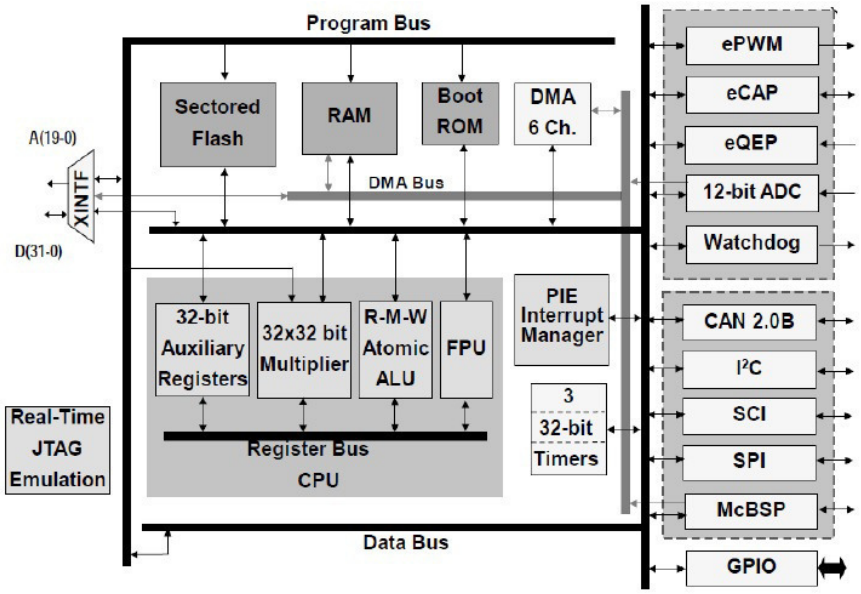
Para armazenar o sistema operacional no Beaglebone Black decidiu-se por um cartão de memória micro SD. Optou-se por essa utilização pela facilidade de remoção de imagem do sistema em situação de falhas, ao invés de salvar na memória flash do dispositivo [BBB]. O sistema operacional adotado foi o Debian 9 LXQT, por ser a versão mais nova e completa, de acordo com o período temporal de realização deste trabalho.

Como identificado na seção anterior, cada bancada possui um DSP, que "é um dispositivo programável, que detêm seu próprio código de instruções"(NUNES; ALBUQUERQUE), sendo responsável pelo controle dos conversores de potência. Todas as bancadas já utilizam o dispositivo, com as propriedades de possuir o

> Uso de DMA (DirectMemoryAccess)
> Operações aritméticas especiais tais como as MACs (Multiply Accumulate) para realizar a Transformada Rápida de Fourier (FFT, do inglês Fast Fourier Transform);
> Memória de Programa separada da memória de dados (Arquitetura Harvard)
> Unidades de ponto flutuante, nos dispositivos mais modernos.
> Os DSP possuem uma arquitetura especializada para reduzir o número de instruções e operações necessárias de um algoritmo de processamento de sinais. Os DSP se diferenciam dos microcontroladores por possuírem suporte as seguintes operações específicas de algoritmos de processamento de sinais, tais como [18]:
> Convolução de Sinais;
> Correlação de Sinais (medida de similaridade entre dois sinais);
> Transformada Discreta de Fourier (DFT, do inglês Discrete Time Fourier);
> Power Spectrum (decompõe o sinalem suas componentes de frequência mostrando a distribuição de potência no espectro de frequência);
> Filtros Digitais- utilizam a alta capacidade de processamento do DSP para realizar cálculos numéricos nas amostras com o intuito de filtrar certas frequências. (VIEIRA, 2013, p.8).

O DSP utilizado neste trabalho é o modelo TMS320f28335 da empresa Texas Instruments, cujo o diagrama de blocos está ilustrado na figura 10, em módulo sem acoplamento na figura 11, e com acoplamento na figura 12.

Figura 10 – Estrutura de blocos do DSP



Fonte: Frank Bormann e Texas Instruments

Figura 11 – DSP 28335

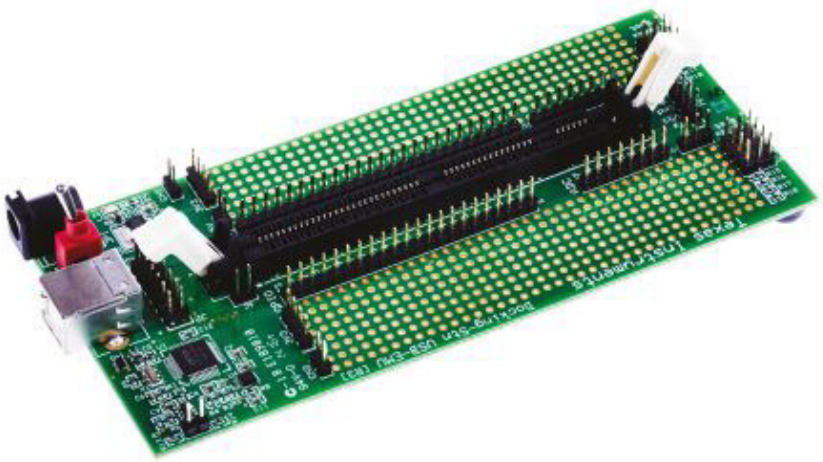


Figura 12 – Kit Delfino da Texas Instruments para encaixar o módulo contendo o DSP.

A CAN (*Controller Area Network*) foi objeto de interesse por parte dos responsáveis do LGEE, visando sua simplicidade e custo-benefício por ser "um barramento serial para interligar dispositivos em rede"(MAURICI, 2005).

> Como detalhado por Livani, Kaiser e Jia [10], CAN possui facilidades que são muito desejadas na área da computação embutida, como tolerância a EMI*, prioridade de mensagens, recuperação de falhas, entre outras.
> Uma rede CAN pode interligar até 2032 dispositivos, sendo que o limite prático é de aproximadamente 110 dispositivos [5], cada um destes é tratado como um nó da rede. (...) Cada nó ligado a este link serial é capaz de ouvir, simultaneamente a outros nós, os dados transmitidos na rede. A escrita, porem, é uma operação permitida somente para um dispositivo por vez.
> (...)
> CAN trabalha de modo semelhante a ethernet comum, mas ao invés de corrigir colisões de transmissão fazendo com que os dois nós em conflito parem de transmitir, a rede CAN usa um arbítrio de comparação binária (figura 2.1) para definir a prioridade das mensagens e decide qual será enviada. Quanto menor o valor associado maior a prioridade. Os bits que trafegam na rede recebem uma denominação de dominante e recessivo, um bit dominante representa o valor lógico 0 e o recessivo, o valor lógico 1 [1](figura 2.2).

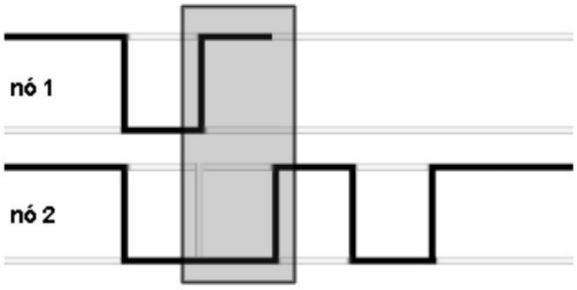


Figura 2.1: Exemplo do arbítrio sendo aplicado durante uma transmissão entre dois nós

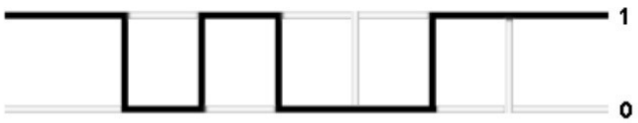
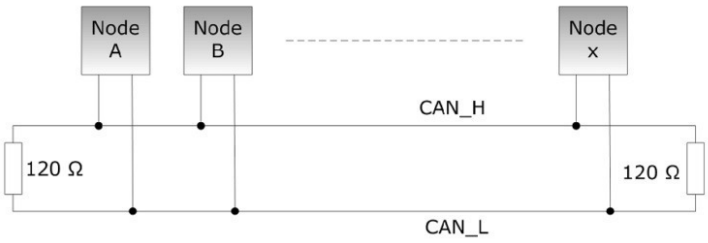


Figura 2.2: Estados lógicos do barramento em uma transmissão

Na figura 13 temos a representação do barramento CAN em que o bit 1 é considerado o bit menos significativo e o bit 0 o bit mais significativo. "Para gerar um bit dominante é necessário que a tensão em CAN_H seja cerca de 3,5V e a tensão em CAN_L seja de 1,5V (padrão ISO 11898)"(FERREIRA et al., 2015).

Figura 13 – Barramento CAN

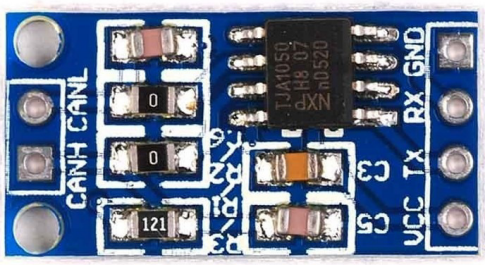


Fonte: FERREIRA et al.(2015).

Para montar a rede CAN, são necessários dois resistores de 120 Ohm cada (conforme 13), dois módulos seriais Can Bus Tja1050 e um conversor bidirecional, já que a tensão vinda do Beaglebone Black é de 3.3V.

Os módulos seriais Can Bus Tja1050 atuam como *transceivers*: enviando e recebendo os sinais do barramento CAN. Na figura 14, nota-se no lado esquerdo os canais de *high* (CANH) e *low* (CANL), em que serão captadas as diferenças de tensão na CAN. No lado direito, há o canal VCC e GND para energia e aterramento respectivamente, e os canais de sinais de transmissão (TX) e recebimento (RX).

Figura 14 – Transceiver



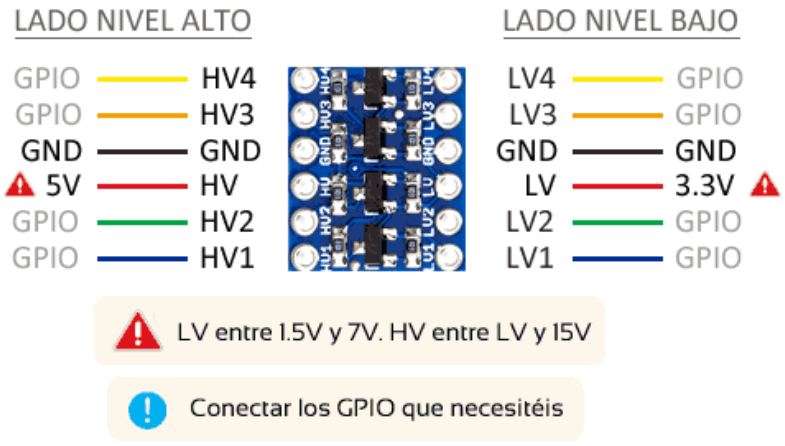
Fonte: Desconhecida.

O transceiver trabalha na faixa de 5V e o nível lógico do BBB é de 3.3V. Assim, para conseguir atingir o patamar dos transceivers, usa-se o conversor de nível lógico. Este dispositivo representado pela figura 15 possui os pinos centrais HV (*High Voltage*), GND e LV *Low Voltage*, sendo as tensões de referência.

Os pinos TX0 e TX1 são bidirecionais, convertendo sinais de 3.3V para 5V em ambos os sentidos. Os pinos RX1 e RX0 são unidirecionais: o sinal de 5V entra no pino RX1 e sai 3.3V no pino RX0.

A linguagem de programação Python foi a mais usada nos trabalhos co-relatos e neste, utilizada de duas formas: como Webservice pelo microframework Flask e como suporte a rede CAN pela biblioteca python-can. Webservice é a forma de acesso que uma aplicação irá se comunicar pela rede, conseguindo realizar a integração e compatibilidade entre diferentes aplicações.

Figura 15 – Conversor bidirecional



Fonte: LLAMAS, 2018.

Python foi criada em 1991 por Guido Van Rossum e semelhante a linguagem ABC, "é orientada a objetos, interpretada e interativa". Possui "controle de bloco por indentação sem a necessidade de símbolos delimitadores e códigos bastante legíveis". Suporta as bibliotecas em C/C++(LOBO, 2007). De acordo com a comunidade PYTHON BRASIL, pode-se "criar programas que funcionam no modo texto, tanto interativos como servidores (ou daemons)", e interfaces com "a interface nativa do seu sistema, ou então utilizando Tk, GTk, Qt, wxWidgets e tantas outras".

Assim, de forma a realizar a integração com o circuito da CAN e ter um modo de comunicação pelo Beaglebone Black

> A biblioteca python-can oferece suporte à rede da área do controlador para o Python, fornecendo abstrações comuns a diferentes dispositivos de hardware e um conjunto de utilitários para enviar e receber mensagens em um barramento CAN.
>
> python-CAN executa qualquer onde Python é executado; de computadores de alta potência com CAN comercial para dispositivos usb diretamente para dispositivos de baixa energia com Linux, como BeagleBone ou RaspberryPi.
>
> Mais concretamente, alguns exemplos de uso da biblioteca:
>
> Registro passivo do que ocorre em um barramento CAN. Por exemplo, monitorando um veículo comercial usando sua porta OBD-II. Teste de hardware que interage via CAN. Os módulos encontrados em carros modernos, motos, barcos e até cadeiras de rodas tiveram componentes testados a partir de Python usando esta biblioteca.
> Protótipos de novos módulos de hardware ou algoritmos de software em loop.
> Interaja facilmente com um ônibus existente.
> Criando módulos virtuais para protótipo da comunicação CAN bus.
> (PYTHON SOFTWARE FOUNDATION, tradução própria).

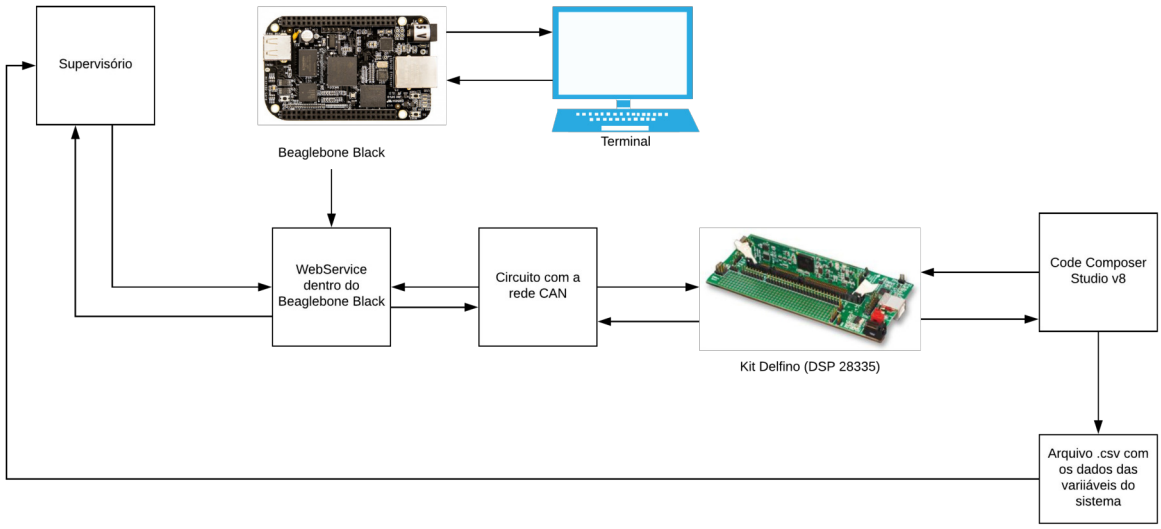
Para realizar a construção inicial do supervisório, referente a bancada selecionada como início do processo de automatização, decidiu-se pela tecnologia React. Esta é uma biblioteca JavaScript para criar telas para sistemas, de forma declarativa, eficiente e flexível,

concedendo criar complexas interfaces de usuário de pequenas e isoladas peças, chamadas "componentes"(React, tradução própria).

4.3 Proposta de estrutura do sistema de controle

Através da literatura e recursos disponíveis, verificou-se e decidiu-se que o sistema embarcado (BBB) e o processador digital de sinais (DSP) deveriam ser utilizados. Isto, a fim de criar um sistema supervisório web com o auxílio das linguagens Python e React em comunicação com a rede CAN, como um meio viável e centralizador, tanto de informações de controle das variáveis do laboratório, quanto de operação. Sendo assim, através da orientação do coorientador Leonardo Ribeiro, criou-se a estrutura de dispositivos e tecnologias em que para fins visuais e explicativos, a figura 16 ilustra os componentes de hardware e software utilizados.

Figura 16 – Solução proposta neste trabalho.



Fonte: Própria autoria.

Na figura 16, observamos que há uma comunicação bidirecional entre o computador externo e o BBB através do protocolo SSH (*Secure Shell*). O Beagle possui um Webservice que recebe e envia mensagens para o barramento CAN (circuito), em que este se comunica também de forma bidirecional com o DSP 28335.

Os sinais enviados para o DSP são interpretados por arquivos criados no Code Composer Studio (CCS v8) em que é possível receber e enviar comandos para o barramento, bem como analisar os envios para acionar componentes do próprio dispositivo.

Os arquivos na linguagem C codificados no CCS, possuem uma função para gerar um arquivo do tipo planilha de extensão .csv, em que são armazenados os dados das variáveis enquanto o sistema está sendo executado.

O supervisório carrega este arquivo de tempos em tempos para gerar os gráficos das

grandezas manipuladas pela bancada e possui os botões de acionamento para a rede CAN.

O Beaglebone Black foi selecionado pela compatibilidade com a rede CAN, com a função de ser o computador embarcado a receber os dados de comunicação desta, e por

> Diferentemente de outras placas de desenvolvimento de sistemas desenvolvimento com especificações semelhantes, a BeagleBone foi desenvolvida primariamente como uma ferramenta de prototipagem rápida e desenvolvimento de projetos eletrônicos devido ao seu hardware open source e grande quantidade de interfaces I/O. Seus 92 pinos conectores a tornam um computador adequado para coleta de dados e controle de dispositivos e equipamentos.
> (...)
> A plataforma é distribuída com o sistema operacional Debian Linux e bibliotecas em C, C++ e Python, dando a BeagleBone versatilidade e performance para uma grande gama de aplicações industriais(BERTELLI, 2015, p.29).

Ressalta-se que, antes de instalar um sistema operacional (SO) no Beagle, se o cabos não vieram de fábrica, é necessário um cabo que consiga conduzir 5V(como os cabos de celulares da década de 2000) ou um cabo com porta mini USB em uma ponta e a outra USB comum. Um cartão SD com capacidade mínima de 4GB é necessário para gravar a imagem do SO. É recomendado gravar neste dispositivo primeiro, pois, caso algo dê errado, basta troca-lo ou substituir o sistema. Em caso de trocar o sistema, o usuário deverá adquirir um gravador de cartão SD, zelando para não perder os dados já inseridos. Para maiores detalhes, visitar o endereço oficial do fabricante (http://beagleboard.org/getting-started), figura 17.

Figura 17 – Página inicial do Beaglebone Black

Fonte: Site oficial do Beaglebone Black

4.3.1 Instalação da plataforma Beaglebone Black

Algumas observações e considerações sobre a instalação do sistema embarcado:

- Apesar de haver outras distribuições de SOs, é aconselhável instalar a distribuição Debian do site oficial apresentado (https://beagleboard.org/latest-images). Há dois tipos de imagem: para Internet das Coisas (IoT - Internet of Things) e a LXQt (The Lightweight Qt Desktop Environment - O ambiente de trabalho leve do Qt, em tradução livre, sendo Qt é um framework multiplataforma para desenvolvimento de interfaces gráficas em C++ criado pela empresa norueguesa Trolltech.) Optou-se pela distribuição 9.4 LXQt por ser a mais completa e a mais nova. Na figura 18 está marcado na cor roxa a distribuição utilizada. Importante frisar a leitura da distribuição adequada ao dispositivo selecionado para uso dentre as várias disponíveis nesta página e depois verificar via terminal qual a versão do kernel, em que neste caso é o 4.4.

- Como o Beagle foi conectado a um computador utilizando Ubuntu 18.04 (pressupõe acesso autorizado por meio de senha, ambiente previamente configurado e demais detalhes realizados que extrapolem este trabalho), instalou-se via terminal o programa Etcher (<https://etcher.io/>), gravador de cartão SD. As instruções necessárias estão em <https://www.edivaldobrito.com.br/etcher-no-linux/> . Deve-se estar atento a todas as etapas do processo, apenas retirando o gravador de cartão quando autorizado. Em caso da gravação estar demorando mais de um minuto para começar a informar a porcentagem de quanto está faltando para começar o processo com a imagem, é sábio desinstalar e reinstalar o programa, já que o cinco minutos são suficientes para realizar toda a operação, já com o cartão em mãos.

- Após inserir o cartão na porta correspondente do beagle, deve-se pressionar o botão próximo desta porta até conectar-se o cabo de energia (utilizou-se o da porta mini USB acoplada ao BBB e a porta USB comum a um computador) e as luzes dos 4 LEDs acenderem, conforme a figura 19. Esse procedimento é feito para carregar a imagem Debian direto do cartão. Todas as vezes que o dispositivo for ligado, essa ação deverá ocorrer, exceto em caso de *reboot*.

- O BBB deverá estar totalmente ligado e sinalizado no computador após 15 segundos. Na porta USB correspondente do computador, aparecerá o nome e na área de trabalho aparecerá um *beagle* com o nome dado na configuração da gravação da imagem no Etcher. Utilizando distribuições recentes, não há a necessidade de instalar driver para o computador acessar via rede o dispositivo. Porém, pode-se realizar por garantia. Assim, o acesso oferecido pelo Beagle é através de um servidor DHCP (IP 192.168.7.1 ou 192.168.6.1), reservando o IP 192.168.7.2 ou 192.168.6.2 para manipulação. Digitando em um browser Google Chrome ou Mozilla o IP 192.168.7.2 (caso utilizado neste trabalho), a mesma página oficial a qual seguimos para estes passos iniciais deverá aparecer.

Figura 18 – Distribuições Debian oficiais

See the Getting Started guide and the community wiki page for hints on loading these images.

Recommended Debian Images

Stretch IoT (without graphical desktop) for BeagleBone and PocketBeagle via microSD card

- Debian 9.5 2018-10-07 4GB SD IoT
  image for PocketBeagle, BeagleBone, BeagleBone Black, BeagleBone Black Wireless, BeagleBone Blue, SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, element14 BeagleBone Black Industrial, Arrow BeagleBone Black Industrial and Mentorel BeagleBone uSomIQ - more info - sha256sum: 52363c654b7a1187656b08c5686af7564c956d6c60c7df5bf4af098f7df395e0

Stretch LXQT (with graphical desktop) for BeagleBone via microSD card

- Debian 9.5 2018-10-07 4GB SD LXQT
  image for PocketBeagle, BeagleBone, BeagleBone Black, BeagleBone Black Wireless, BeagleBone Blue, SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, element14 BeagleBone Black Industrial, Arrow BeagleBone Black Industrial and Mentorel BeagleBone uSomIQ - more info - sha256sum: 5dde8099b9a540ef85f5c35b4bd6e56410f7a499a5fb02a96d918d67d8769f5e

Stretch for BeagleBoard-X15 via microSD card

- Debian 9.5 2018-10-07 4GB SD LXQT image for BeagleBoard-X15 - more info -
  sha256sum: db3f2b6a1305de715d33c1cb330ae51fbefe15379d4bd797ed3c1aa685b9729d

Stretch for BeagleBoard via microSD card

- Debian 9.5 2018-10-07 4GB SD LXQT image for BeagleBoard, BeagleBoard-xM - more info -
  sha256sum: 2a29626ab7c20890109a0eea4ea6e88e4e31d01d8a447b38eaac5953d8eb9ece

To turn these images into eMMC flasher images, edit the /boot/uEnv.txt file on the Linux partition on the microSD card and remove the '#' on the line with 'cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh'. Enabling this will cause booting the microSD card to flash the eMMC. Images are no longer provided here for this to avoid people accidentally overwriting their eMMC flash.

For testing, flasher and other Debian images, see elinux.org/Beagleboard:BeagleBoneBlack_Debian and debian.beagleboard.org/images.

"Alpha" Testing Debian images

- Debian Testing (Buster) 2018-06-17 4GB SD IoT image for PocketBeagle, BeagleBone, BeagleBone Black, BeagleBone Black Wireless, BeagleBone Blue, SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, element14 BeagleBone Black Industrial, Arrow BeagleBone Black Industrial and Mentorel BeagleBone uSomIQ - more info -
  sha256sum: 9ed014714643f911582a8869c4477b9bd8548332e60641e60dd21e0b76ac3ffd
- Debian Testing (Buster) 2018-06-17 4GB SD IoT image for BeagleBoard-X15 - more info -
  sha256sum: f74f4e743ad02201ef8d96dc9e0dd30b904c20d6ead08b8537a91caaa1c5a55d

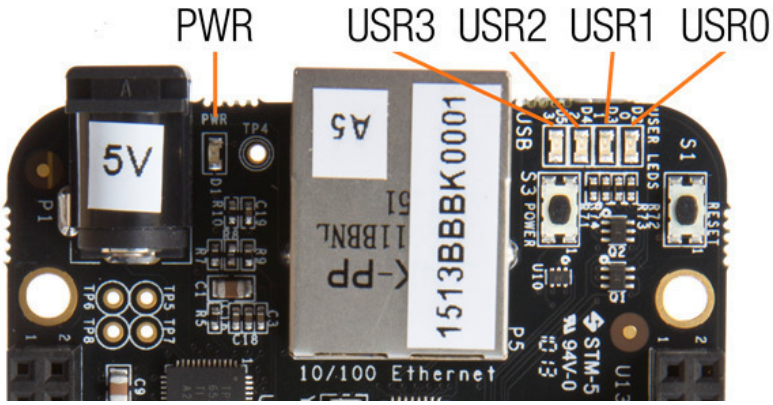
Older Debian images

BeagleBone compatibles via microSD card (without flashing the eMMC)

- Debian 9.5 2018-08-30 4GB SD IoT image for PocketBeagle, BeagleBone, BeagleBone Black, BeagleBone Black Wireless, BeagleBone Blue, SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, element14 BeagleBone Black Industrial, Arrow BeagleBone Black Industrial and Mentorel BeagleBone uSomIQ - more info - sha256sum: e591fd74e63545aa86ce602ef2bacfada535994e8de278c1ef5707501ea78aa5
- Debian 9.4 2018-06-17 4GB SD LXQT image for PocketBeagle, BeagleBone, BeagleBone Black, BeagleBone Black Wireless, BeagleBone Blue, SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, element14 BeagleBone Black Industrial, Arrow BeagleBone Black Industrial and Mentorel BeagleBone uSomIQ - more info - sha256sum: 25ba1710f80ae74903571bf657ebccef3c3a403e604bf06e788113463f3009fb
- Debian 9.4 2018-06-17 4GB SD IoT image for PocketBeagle, BeagleBone, BeagleBone Black, BeagleBone Black Wireless, BeagleBone Blue, SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, element14 BeagleBone Black Industrial, Arrow BeagleBone Black Industrial and Mentorel BeagleBone uSomIQ - more info - sha256sum: 8f6b9b7e38e4a0f0f979b6281982a16ce09c572224b586283eedff5e205f7673

Fonte: Site oficial do Beaglebone Black

Figura 19 – LEDs disponíveis na placa.

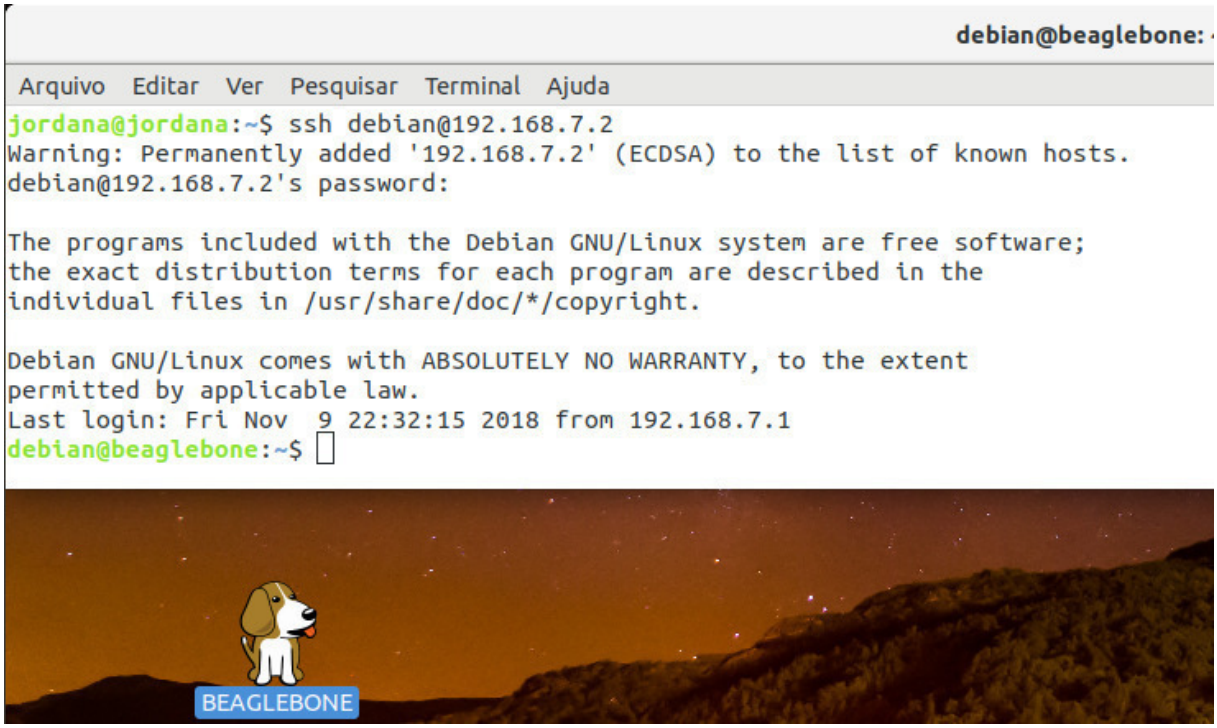


Fonte: Site oficial do Beaglebone Black

Como o sistema operacional Debian instalado e acesso a placa realizado, tem-se a possibilidade de escolha ao sistema por duas vias: no computador colocar o programa Putty (sudo apt install putty) configurando com IP 192.168.7.2 ou abrir o terminal, inserindo

o comando *ssh debian@192.168.7.2*. A senha de acesso por padrão em ambos os casos é *temppwd*. Preferencialmente, o segundo método foi utilizado conforme demonstrado na figura 20 e com o ícone do Beagle para melhor esclarecimento do parágrafo anterior.

Figura 20 – Acesso via terminal do computador em que o BBB está conectado via USB.



Fonte: Própria autoria.

4.3.2 Configuração da rede no Beaglebone Black

O acesso para a internet dentro do Beagle é possibilitado com a instalação de pacotes via dois terminais. Um terminal do dispositivo e outro do computador. Seguindo orientações do Elementz team (2014) deve-se atuar como usuário *root* em ambos. Como usuário Debian, basta digitar *sudo su* e a senha *temppwd*. Como usuário Ubuntu 18.04 LTS, também digitar *sudo su* para se tornar *root* e a senha de acesso.

No terminal do BBB (somente o comando):

- *ifconfig usb0 192.168.7.2*
- *route add default gw 192.168.7.1*

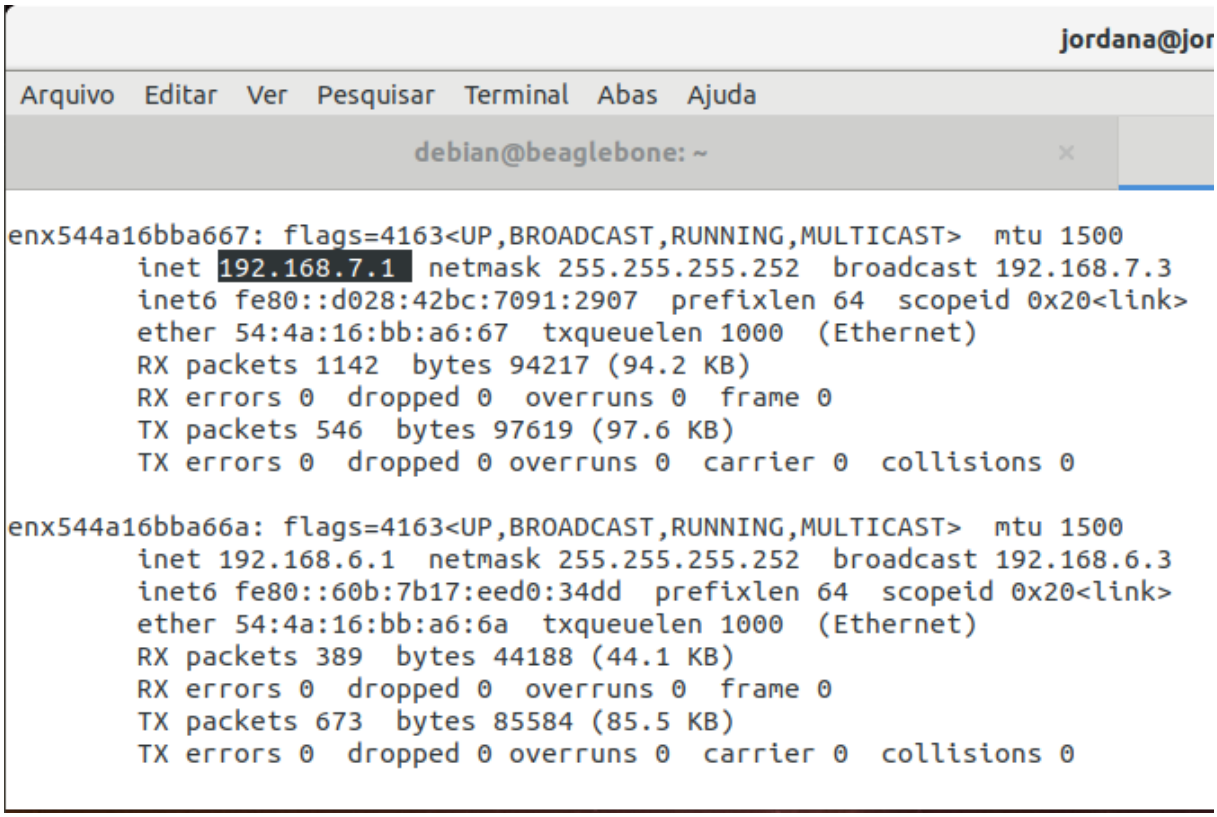
No terminal do Ubuntu (somente o comando):

- *ifconfig eth5 192.168.7.1*
- *route add default gw 192.168.7.1*
- *iptables –table nat –append POSTROUTING –out-interface wlan0 -j MASQUERADE*

- *iptables –append FORWARD –in-interface eth5 -j ACCEPT*
- *echo 1 > /proc/sys/net/ipv4/ip_forward*

eth5 é o nome como o Ubuntu recebeu o BBB e wlan0 é como o Ubuntu está sendo conectado pela internet, em um caso hipotético. Para descobrir as correspondências reais adequadas, no terminal do computador digite *ifconfig*, como em 21. O IP 192.168.7.1 está relacionado ao enx544a16bba667 sendo este correspondente ao eth5. Quanto ao wlan0, este também estará listado a sua representação conforme o meio adotado para realizar a conexão de internet no Ubuntu (cabeada ou wireless).

Figura 21 – Comando *ifconfig* no terminal do computador.



Fonte: Própria autoria.

No terminal do Beagle digite *echo "nameserver 8.8.8.8»> /etc/resolv.conf* para questão de DNS, *ping 8.8.8.8* para verificar conexão e *ping www.google.com.br* para verificar se a resolução de DNS foi aceita. Ao digitar os comandos com *ping* espere por 3 segundos e pressione Ctrl+C, verificando se os pacotes foram transmitidos com sucesso. Em caso negativo, há as seguintes possibilidades:

- Realizar todas as etapas de configuração de rede novamente, atentando-se aos nomes reais dos dispositivos (caso do eth5 e wlan0).
- Nos terminais fazer uso do comando *sudo service networking restart* para reiniciar a rede.

- Reiniciar o Beagle (*sudo reboot*) e realizar mais uma vez as configurações.
- Usar no terminal do BBB *sudo systemctl restart systemd-resolved*.
- Configurar o firewall e portas do Ubuntu, conforme <https://www.digitalocean.com/community/tutorials/how-to-setup-a-firewall-with-ufw-on-an-ubuntu-and-debian-cloud-server>.
- Testar as configurações em outra rede de conexão. Essa última tentativa é a mais crítica, indicando uma erro de acesso da própria rede externa. Provavelmente por algum item do roteador ou do provedor de acesso. Este caso ocorreu neste trabalho e foi de difícil identificação, levando a trocas desnecessárias de imagens do Debian.

Com a internet funcionando dentro do dispositivo baixamos todos os pacotes necessários para continuar o trabalho, a serem descritos neste texto. Enfatiza-se digitar os comandos de rede sempre que o uso será requerido.

4.3.3 Biblioteca Python-CAN 3.0

Para uso da biblioteca python-can 3.0 é necessário que a rede CAN esteja habilitada no Beaglebone Black. De posse do conhecimento anterior, via terminal (somente o comando):

- Algumas operações precisam de privilégios root: *sudo su*
- Instalar o can-utils:
  git clone https://github.com/linux-can/can-utils.git
  cd can-utils
  ./autogen.sh
  ./configure
  make
  make install
- Ativar o overlay da interface can: *nano /boot/uEnv.txt*
- Alterar a linha:
  dtb_overlay=/lib/firmware/<file8>.dtbo
  Para:
  dtb_overlay=/lib/firmware/BB-CAN1-00A0.dtbo
- Adicionar os drivers can:
  echo can » /etc/modules
  echo can-dev » /etc/modules
  echo can-raw » /etc/modules
- Adicionar a interface can0:
  nano /etc/network/interfaces

- Inserir as linhas abaixo, no final do arquivo:
  allow-hotplug can0
  iface can0 can static
  bitrate 500000
  up /sbin/ip link set can0 up
- Reiniciar o BBB: *sudo shutdown -r now*

Para confirmar os passos anteriores, basta digitar *ifconfig* quando o Beagle reiniciar e verificar se há listado "can0"ou "can1"conforme o canal selecionado para habilitar. Caso o BBB não tenha rede, é possível baixar o pacote no computador externo e enviar pelo terminal via scp:

scp -r /pasta-local/ user@domain:/pasta-remota/

Exemplo

scp -r /home/jordana/Documentos/can-utils debian@192.168.7.2:/home/can/ docs-can-utils

Para realizar estes comandos é necessário estar como *root* e não ter espaços excedentes no comando.

Todas as vezes em que a rede CAN for necessária (independente de dar *reboot* no BBB) é necessário habilita-la através de:

sudo ifconfig can0 down
sudo ip link set can0 up type can bitrate 500000 loopback on
sudo ifconfig can0 up
cangen can0

Lembrando que "can0"foi o canal da CAN utilizado neste trabalho, ao invés de "can1".

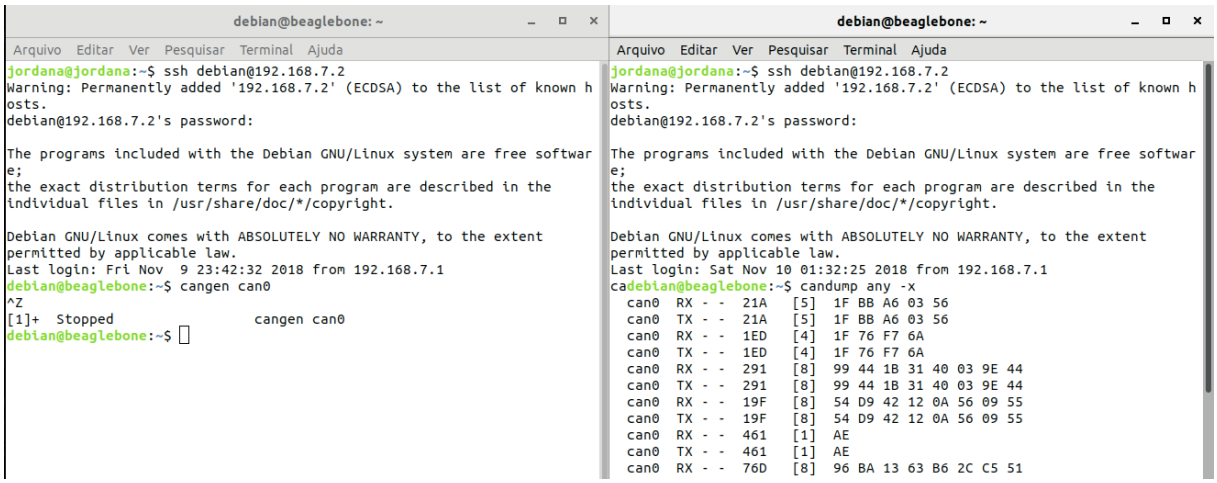
Após, a python-can é instalada com o comando *pip install python-can*. Caso o pacote *python* não esteja presente: *sudo apt-get install python3.5* (3.5 é a versão do python, havendo a 2.7 também) e para o pacote *pip*: *sudo apt-get install python-pip*.

Para testar o circuito da CAN (próxima seção) e outros procedimentos, segundo OLIVEIRA(2017), basta digitar em um terminal *cangen can0* ou *cangen can1* para o envio de mensagens aleatórias, figura 22. Em outro, *candump any -x* para mostrar as "mensagens de todos os controladores. O parâmetro –x dará os detalhes de qual controlador enviou o recebeu a mensagem."

Caso o resultado seja parecido com a figura 22, digite *sudo nano rx.py* (sudo para atuar como root, nano para escrever em arquivo, rx o nome do arquivo e .py a extensão do arquivo, que neste caso é python).

Ao aparecer o modo escrita no terminal, coloque o código de acordo com o Anexo A - Algoritmo rx.py e Anexo B - Algoritmo tx.py. Segundo Oliveira(2017) e comprovado neste trabalho, com a rede CAN ligada de forma correta, estes scripts em Python irão enviar e receber mensagens do barramento.

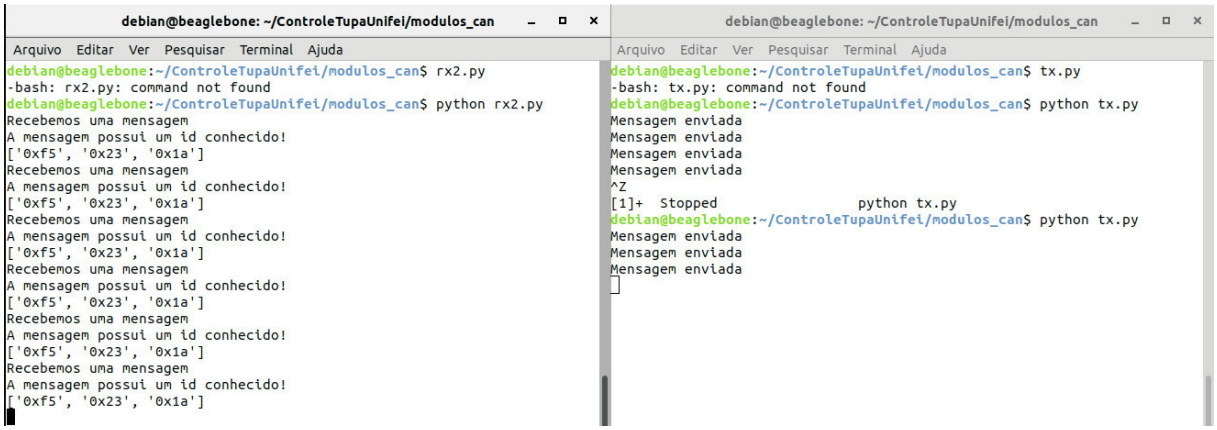
Figura 22 – Comandos *cangen* e *candump* em diferentes terminais.



Fonte: Própria autoria.

rx.py irá receber, verificar e informar a mensagem do script também em Python de *tx.py*. Enquanto que *tx.py* irá enviar de 5 em 5 segundos uma mensagem definida para ser reconhecida por *rx.py*. Na figura 23, podemos observar a execução dos arquivos em diferentes terminais, sendo que não é possível chama-los diretamente, devendo colocar *python* com o nome do arquivo para que este possa ser executado.

Figura 23 – Execução dos scripts *rx.py* e *tx.py* em diferentes terminais.



Fonte: Própria autoria.

4.3.4 Rede CAN

A CAN é "um barramento serial para interligar dispositivos em rede"(MAURICI, 2005, p.5). Sendo que

> cada nó ligado a este link serial é capaz de ouvir, simultaneamente a outros nós, os dados transmitidos na rede. A escrita, porem, é uma operação permitida somente para um dispositivo por vez"(MAURICI, 2005, p.6).

Por tais características, ser amplamente utilizada na indústria, e atender aos requisi-

tos de comunicação, optou-se pela rede CAN, tendo em vista que deve ser comportável ou compatívelcom as demais soluções empregadas.

Para montar o circuito da CAN são necessários os módulos seriais Can Bus Tja1050 e o conversor bidirecional. Na integração da solução, o Beaglebone Black e o Kit Delfino contendo o DSP 28335.

A Figura 24 mostra os pinos a serem conectados para realizar a comunicação entre o circuito e o BBB.

Figura 24 – Conexões do Beaglebone Black.

Pino	Função
P9_1	GND
P9_2	GND
P9_3	DC_3.3V
P9_4	DC_3.3V
P9_5	VDD_5V (fonte externa)
P9_6	VDD_5V (fonte externa)
P9_7	SYS_5V (fonte USB)
P9_19	dcan0_rx
P9_20	dcan0_tx
P9_24	dcan1_rx
P9_26	dcan1_tx

Fonte: Própria autoria.

A Figura 25 mostra as ligações entre o conversor bidirecional, DSP e BBB para realizar o funcionamento da rede CAN.

Figura 25 – Conexões do circuito do barramento CAN. Fonte: Própria autora.

Pino	Conversor Bidirecional
GND1	GND2 : Conversor Bidirecional
HV1	RX1: Transceiver 1
LV1	P_31 (TX): DSP
HV2	TX1: Transceiver 1
LV2	P_30 (RX): DSP
HV	P9_7 (SYS_5V): BBB ou 5V: DSP
LV	P9_3 ou P9_4 (3.3V): BBB ou 3.3V: DSP
HV3	RX: Transceiver 2
LV3	P9_24 (RX): BBB
HV4	TX: Transceiver 2
LV4	P9_26 (TX): BBB

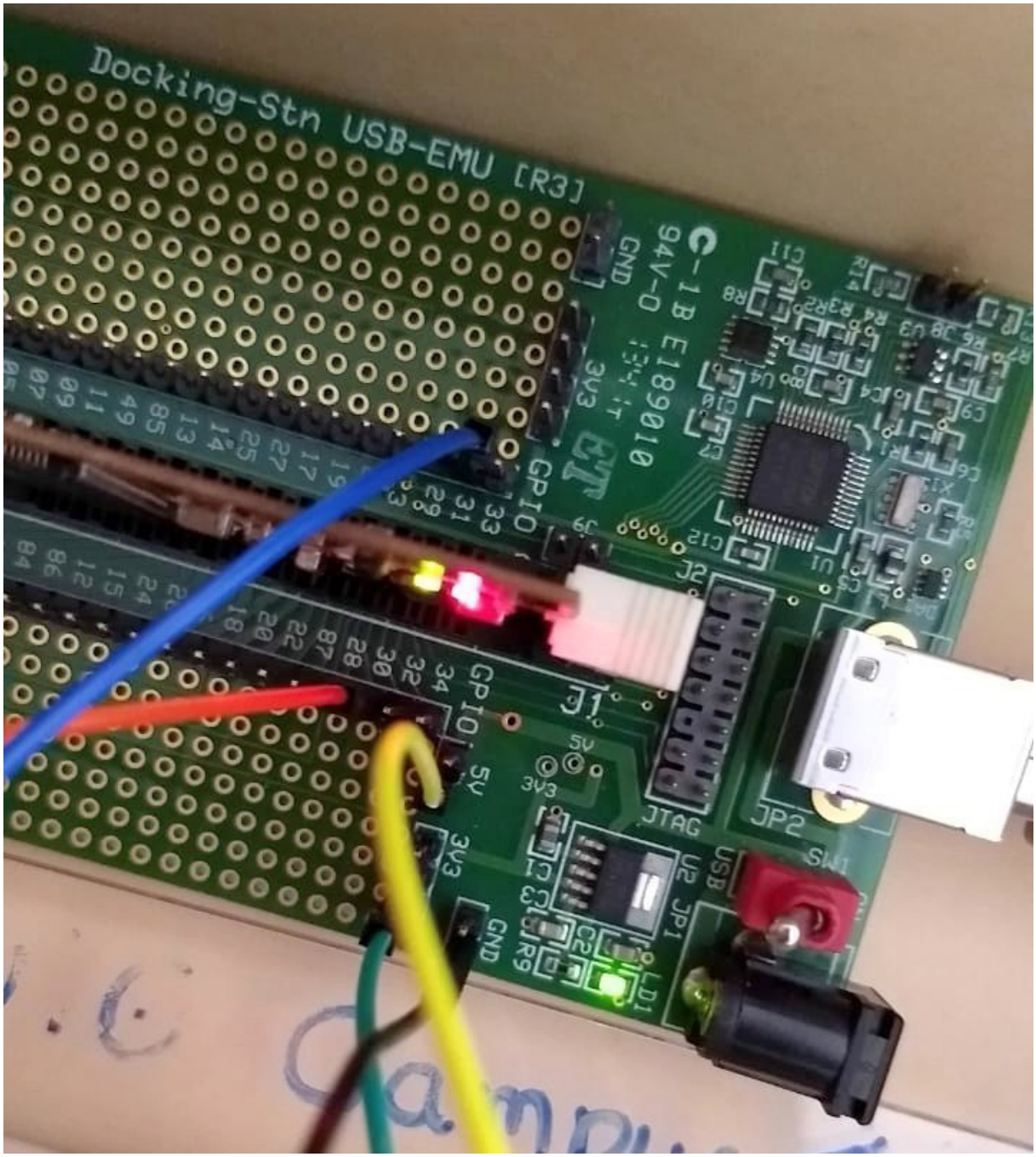
Fonte: Própria autoria.

Os pinos 30 e 31 do kit Delfino contendo o DSP TMF28335, correspondem ao RX e TX do barramento CAN, conectados pelos fios laranja e azul em 26.

Na figura 27 os fios brancos são das ligações RX e TX dos transceivers no conversor bidirecional (elemento à esquerda), os pretos do GND, os amarelos do VCC (5V) e verde da tensão de 3.3V do DSP.

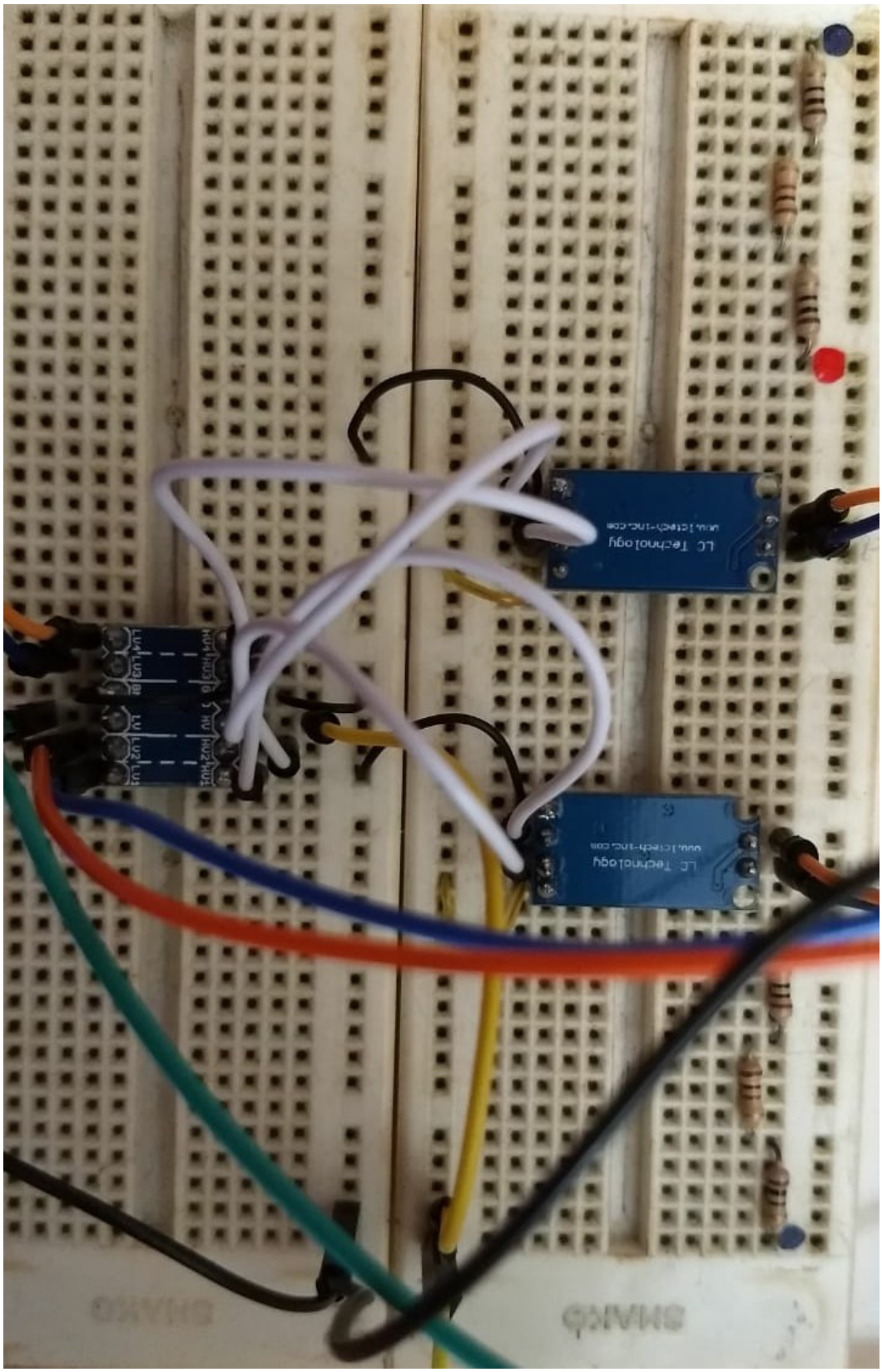
A montagem completa do circuito está na figura 28, em que é possível ver os três dispositivos conectados e ligados.

Figura 26 – Ligação nos pinos 30 e 31 do kit Delfino.



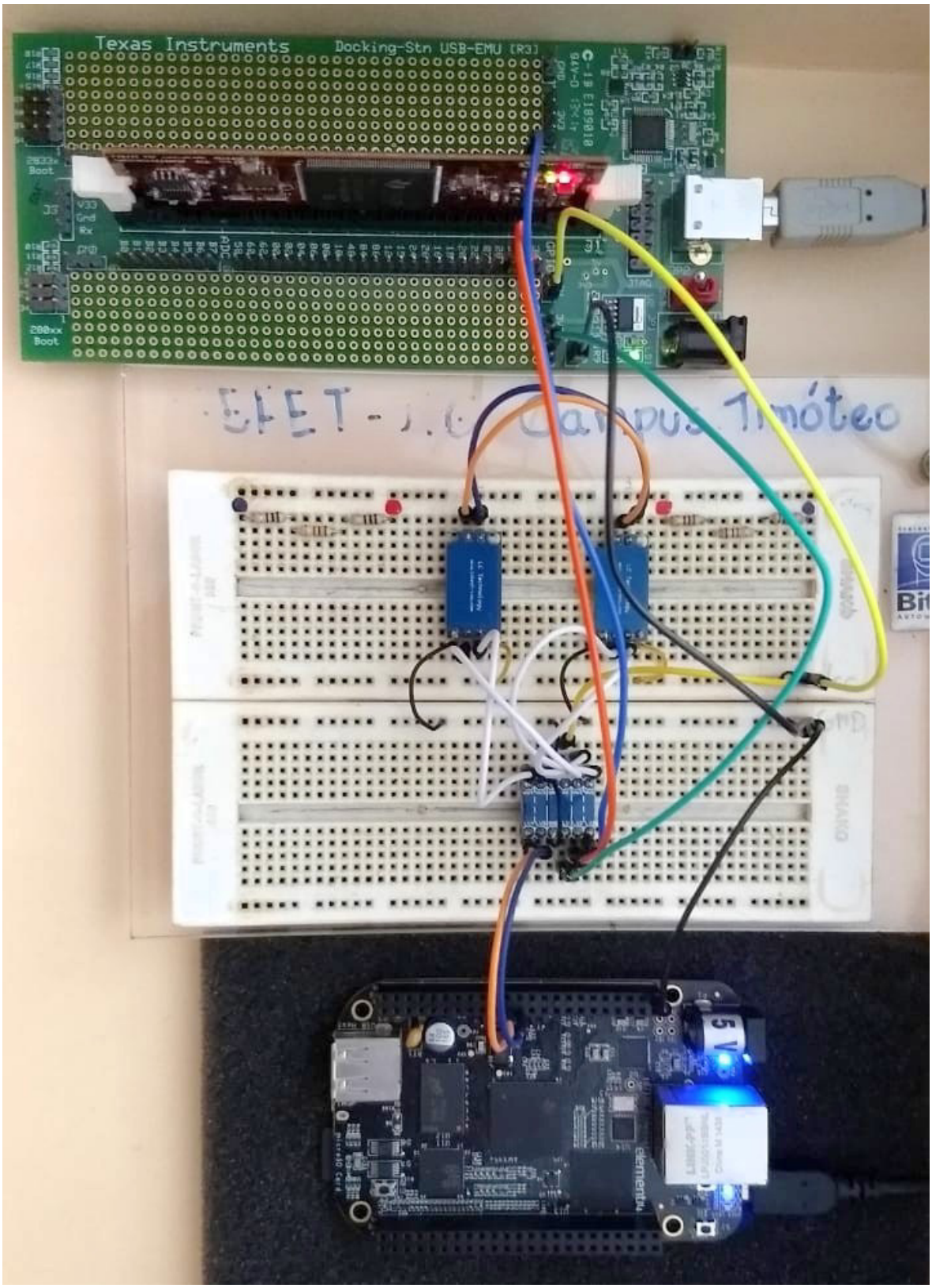
Fonte: Própria autoria.

Figura 27 – Circuito contendo o conversor bidirecional e os transceivers.



Fonte: Própria autoria.

Figura 28 – Montagem completa do circuito com o DSP e o Beaglebone Black.



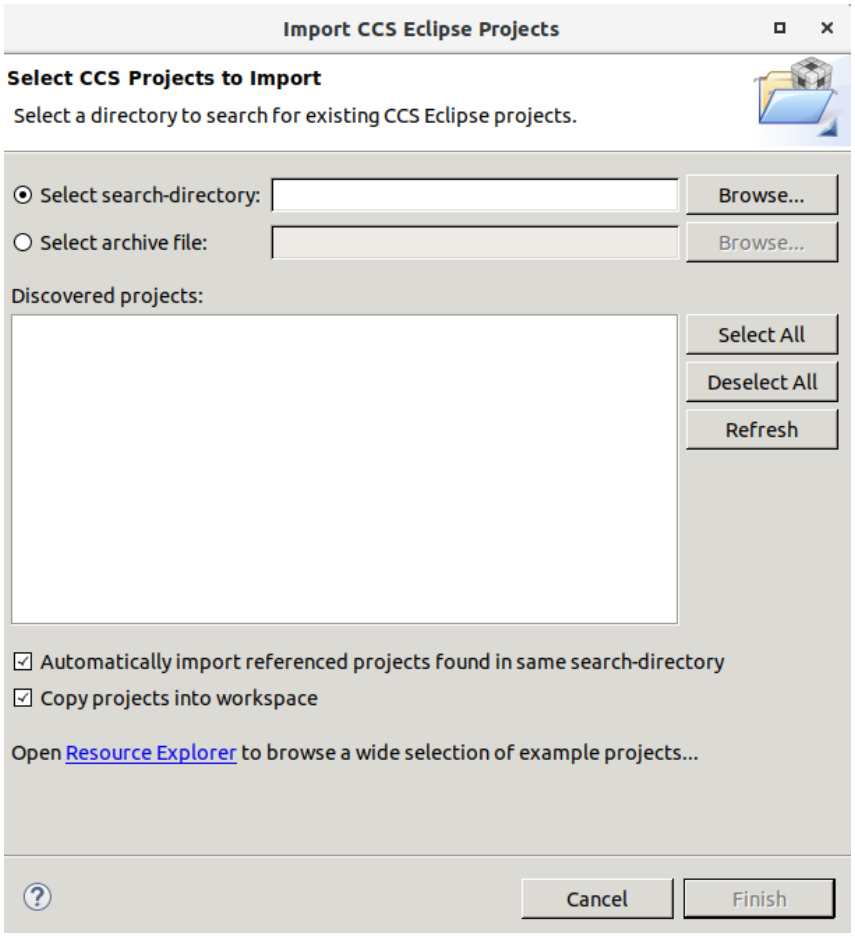
Fonte: Própria autoria.

4.3.5 *Digital Signal Processor* (DSP)

Para executar os comandos pertinentes ao TMSF28335, é necessário a sua manipulação por meio de codificação no software Code Composer Studio. A versão contemplada foi a 8.2.0, mas há outras disponíveis em <http://processors.wiki.ti.com/index.php/Download_CCS>. No momento da instalação, verificar os dispositivos suportados e outros itens é fundamental como um dos requisitos da boa execução dos arquivos.

Ressalta-se a importância de incluir a biblioteca correspondente ao dispositivo escolhido, bem como a sua versão correta a fim de abrangir eventuais codificações em versões anteriores do software. Se houve importação dos arquivos do projeto, como na figura 29 é benéfico a marcação dos itens de referenciar e copiar depedências para o local atual, sendo este não aconselhável ser a área de trabalho do sistema operacional. Demais especificidades da instalação e importação de projeto ocorridas neste trabalho podem ser observadas de acordo com o Apêndice A.

Figura 29 – Modelo de importação do projeto.



Fonte: Própria autoria.

Do trabalho desenvolvido pelos colaboradores, em específico por Luís Eduardo Guedes de Oliveira Soares responsável por acoplar e implementar o código da rede CAN na bancada, a parte a ser diretamente utilizada está demostrada no algoritmo no Anexo C referente ao arquivo "Tupa.c". Soares sabendo que os códigos no Beaglebone são de 8 bytes estruturou que o bit mais significativo refere-se a categorização das grandezas de medição e alarme, dessa forma:

- Tensão: 00.00.00.03.00.00.00.XX
- Corrente: 00.00.00.02.00.00.00.XX
- Potência: 00.00.00.01.00.00.00.XX
- Alarmes: 00.00.00.04.00.00.00.XX

E os bits menos significativos representam os valores que essas grandezas apresentam no momento da medição. Para a criação do padrão para o envio de mensagens através do Beaglebone Black, segue-se o mesmo princípio como é esclarecido pelo autor:

> Bloco de Envio de mensagem:
> root@beaglebone: cansend can1 000000FF00.00.00.03.00.00.00.01
>
> Da mensagem acima a parte cansend can1 é responsável por solicitar a informação via Beaglebone conectado à porta CAN1, uma vez que vá fazer leitura somente, este bloco não será necessário implementar no Code Composer visto que poderá ser feita a leitura direta apenas atribuindo o valor da variável. A parte em verde foi definida no codecomposer como código de envio de mensagem logo este endereço diz que alguma mensagem será enviada. A parte em azul é responsável por definir qual variável será lida (tensão, corrente, potencia) e a parte em laranja diz em qual elemento será realizado esta leitura(Tensão na fase 1, Corrente na carga, Tensão no barramento). O bloco
>
> rxcan1 = ECanaMboxes.MBOX1.MDL.all;(menos significativos)
> rxcan2 = ECanaMboxes.MBOX1.MDH.all;(mais significativos)
> responsável por ler o conjunto de bytes no programa CCS que será utilizado no DSP. Desta forma pensou-se em fazer de forma que a primeira parte indicasse qual leitura seria feita e na segunda parte define-se qual o equipamento seria mensurado. Exemplo:
> Quando a sequência de rxcan1=3 que são os primeiros bytes. O programa entra para fazer leitura de tensão e quem vai definir tensão em qual ponto será lido é o rxcan2 que receberá valores de 1, 2, 3,etc. Baseado nas variáveis do sistema que irão definir qual local ler.

A separação da parte mais significativa da menos significativa por meio de um algoritmo pode ser observado pelo Anexo C pelas estruturas condicionais switch..case e if para identificar as variáveis do sistema à serem manipuladas. No último caso de verificação, houve a pré definição de "LER_TENSAO"(linha 32 do código) sendo a parte mais significativa representando o bloco contendo o valor 3, como na parte azul do comando de envio exemplificado pelo autor, em que a mesma além de receber a mensagem da CAN, envia uma resposta pelo barramento. Importante notar no Anexo C que falta a contemplação de algumas variáveis da bancada, mas uma vez testado e validado, a expansão para as demais, bem como as demais bancadas torna-se viável.

5 Resultados alcançados

Os resultados deste trabalho podem ser classificados em resultados finais e resultados ao longo do processo.

Os resultados ao longo do processo caracterizam-se por todas as configurações e instalações realizadas para atingir os resultados finais. Dessa forma, destacam-se a ambientação no sistema embarcado, o circuito e comunicação da rede CAN e o desenvolvimento em conjunto do DSP via Code Composer Studio.

No que refere-se aos resultados finais, como relatado nos capítulos anteriores, as codificações do sistema se dividiram em duas: uma dentro do Beaglebone Black e a outra em um computador externo. O BBB consome os dados via Flask através do pacote *pip* (sudo apt-get install python-pip) e do framework *sudo pip install flask*, necessários para o código no Apêndice B (Algoritmo bancada.py).

Este algoritmo é um exemplo da interface de acesso aos comandos CAN para o circuito através do BBB. É esta porta de comunicação que permite com que o site encontre um meio de enviar comandos para a CAN por meio do site no computador externo. Para executar somente este algoritmo, é necessário salva-lo como um arquivo (sudo nano nomeDoArquivo.py) e chama-lo no terminal através do python (python nomeDoArquivoDeExecucao.py. Nota-se que para fazer a conexão com a parte da CAN em questão, o endereço deve ser acompanhado de "/app/tcan"(digite http://192.168.7.2:5000/app/tcan no navegador do computador externo).

No site do lado externo do BBB, ou seja, no computador em que está conectado, iniciou-se os trabalhos pelo framework DJango. Entretanto, para as finalidades propostas, apesar de também utilizar Python e ter um maior suporte da biblioteca interativa para gráficos Plot.ly, decidiu-se migrar para o React Js pela leveza, baixa complexidade de código e maior foco nas telas para o usuário.

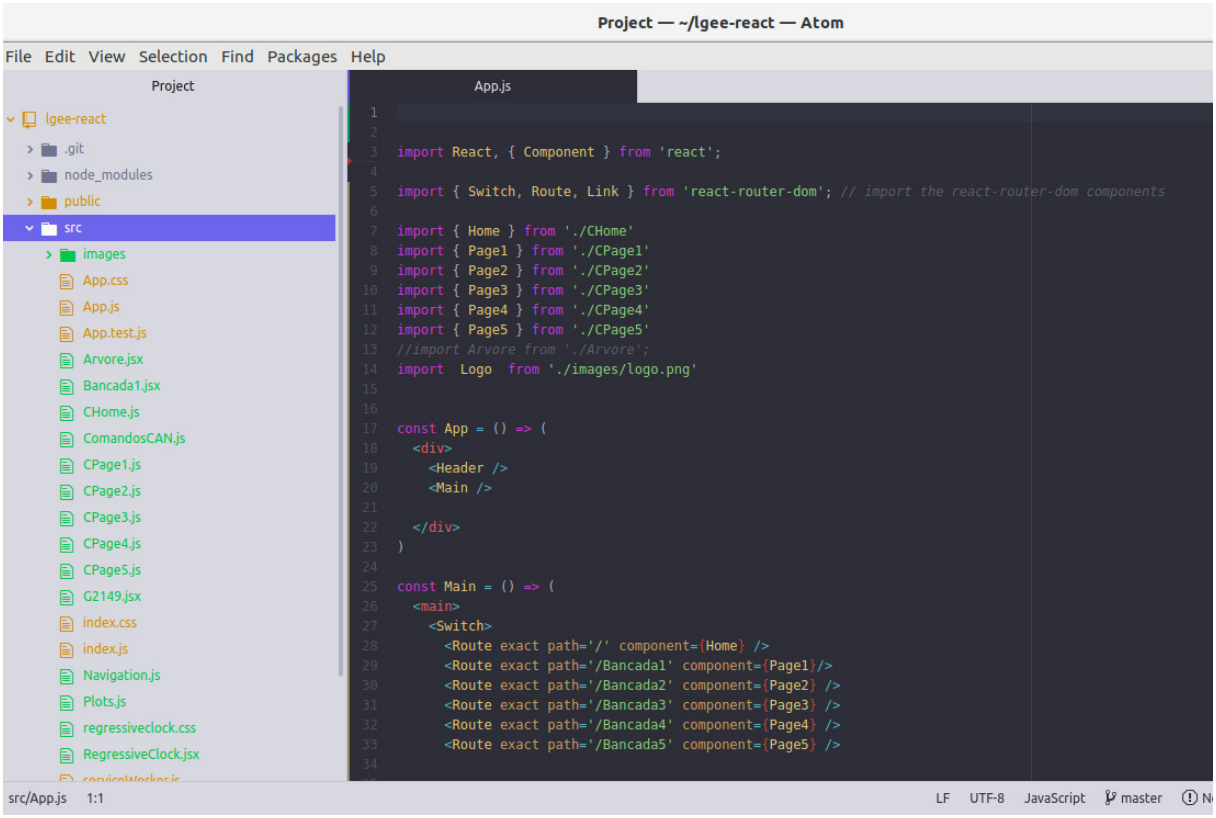
O pacote do React é instalado pelos comandos (digitar apenas os comandos como sudo *sudo su*):

- *npm install -g create-react-app*
- *create-react-app nomeDaAplicação*
- *cd nomeDaAplicação/*

O Bootstrap foi adicionado ao diretório da aplicação pelo terminal (*npm install –save react react-dom && npm install –save react-bootstrap*) e no arquivo "index.js"pelo import: *import '../node_modules/bootstrap/dist/css/bootstrap.min.css'* para personalizar os itens de tela. O comando *npm start* dentro do diretório da aplicação, a inicializa pela porta 3000 (http://localhost:3000/ no navegador).

A figura 30 demonstra que as páginas de apresentação do projeto foi completamente criado por meio de arquivos JavaScript, e dentro deles é possível adicionar estruturas em HTML, respeitando o propósito da tecnologia React.

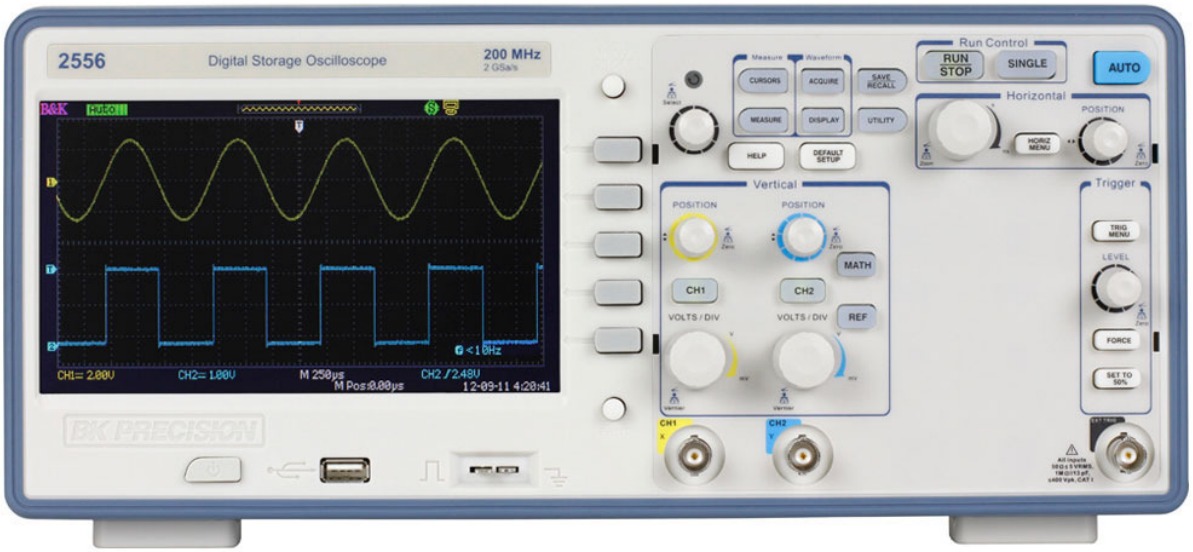
Figura 30 – Estrutura do projeto em React.



Fonte: Própria autoria.

A fim de simular o comportamento de um oscilóscopio, representado pela figura 31, como ver gráficos das variáveis, visualizar mais de um gráfico (diferentes canais [CH1 e CH2]), criar uma zona de aumento [*zoom*] em um/nos gráficos, observar variáveis em tempo real, salvar gráfico, entre outras funções, a biblioteca Plot.ly foi empregada tanto no projeto inicial em DJango (Python) quanto em React (JavaScript). Há várias bibliotecas de gráficos documentadas e mais populares entre os desenvolvedores de software, entretanto, esses efeitos só são contemplados em sua totalidade pelo Plotly. Um exemplo de sub gráfico com as funcionalidades de salvar o gráfico como imagem de extensão png, *zoom* em uma área, arrastar uma área do gráfico [*pan*), caixa de seleção, laço de seleção, mais e menos *zoom* em todo o gráfico, auto escala, voltar as coordenadas para o gráfico original, pontilhar e mostrar as coordenadas X e Y dos pontos no gráfico ao se passar o mouse sobre eles podem ser verificadas pela figura 32 lado superior direito. Ao acessar <https://plot.ly/~LGEE.UNIFEI.Itabira/0/#/> é possível notar os recursos de forma mais prática e certificar-se das operações disponíveis também do lado superior direito.

Figura 31 – Exemplo de um osciloscopio digital.



Fonte: BK Precision.

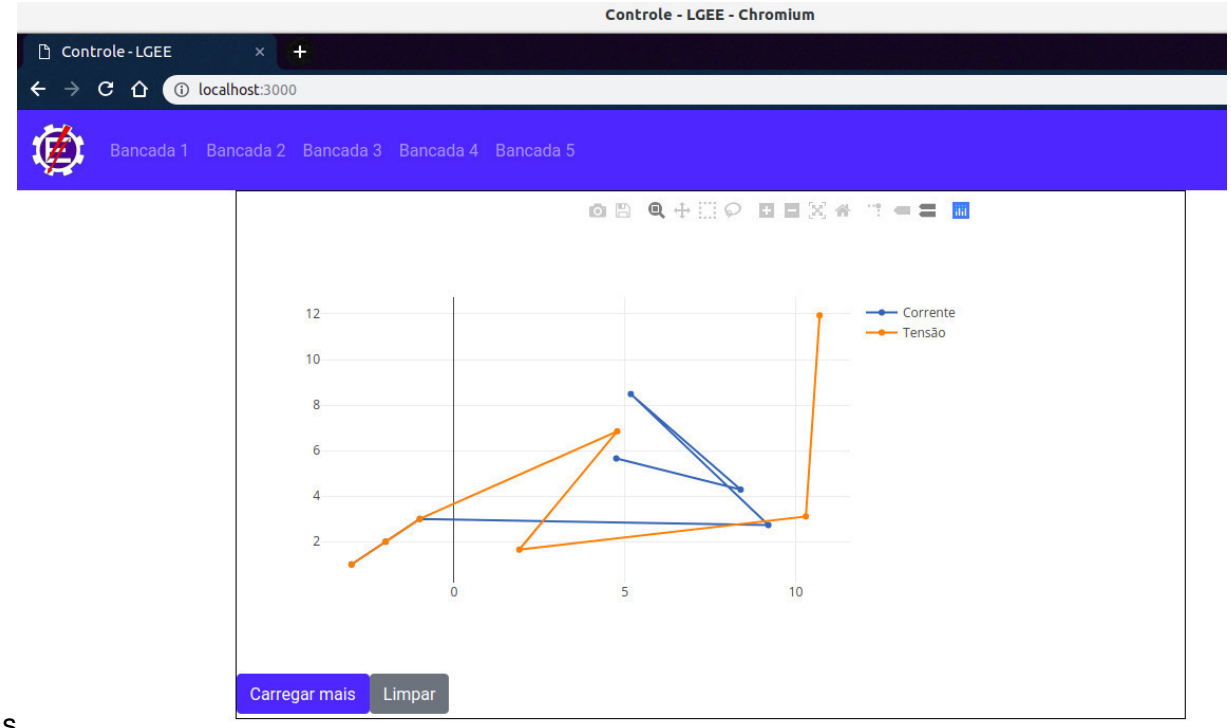
Figura 32 – Exemplo de sub gráficos no Plotly.

Fonte: Plotly.

A diferença é que em React por utilizar JavaScript, para criar os gráficos em tempo real foi mais desafiador se comparado com o Plotly Python devido a pouca documentação, pouca utilização da comunidade (que migrou para outras plataformas de gráfico como o Chart JS) e falta de exemplos explicativos. Baseando-se no gráfico em tempo real em Plotly Python pro-

duzido ao utilizar-se o DJango como exemplo direcionador dos parâmetros a serem mantidos, para criar o gráfico da figura 33, o colaborador João Marcos Cota criou e conduziu sua estratégia de resolução para a questão do efeito de tempo real nos gráficos, testando e observando o funcionamento cíclico do React. O algoritmo pode ser contemplado no Anexo D e explicado em detalhes em <https://medium.com/@jmmccota/plotly-react-and-dynamic-data-d40c7292dbfb>

Figura 33 – Gráfico em tempo real do Plotly em React.



s

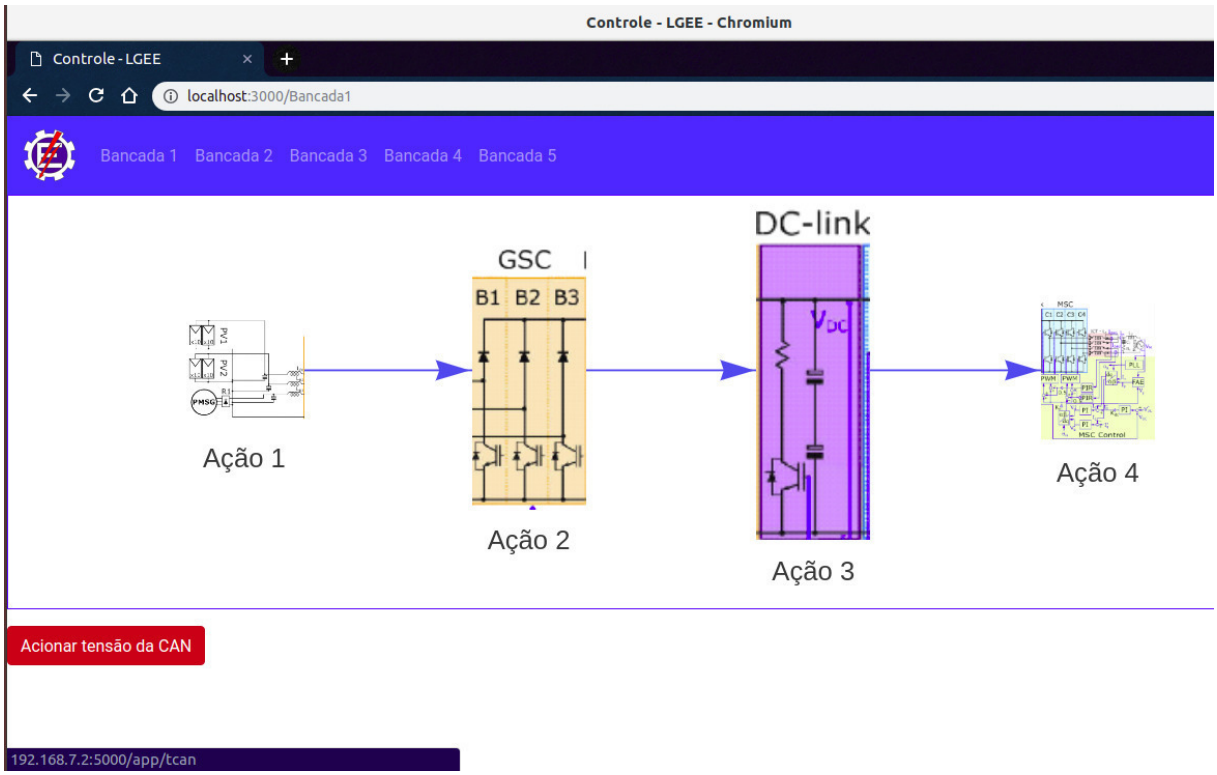
Fonte: Própria autoria.

A figura 33 representa o protótipo funcional da página inicial de todo o sistema do supervisório. Observa-se que na parte superior estão já preparados de forma estrutural no código o recebimento da expansão para as outras bancadas do laboratório (Bancada 1, 2, 3, 4 e 5). A intenção dessa tela é esquematizar um teste prático do Plotly na geração de um gráfico em tempo real, mesmo que não seja da bancada selecionada, a fim de validar a sua escolha para o projeto. Além disso, uma vez feito para uma bancada e testado, a sua aplicação torna-se plausível e pode ter como exemplo o modelo de visualização como na figura 32, em que é apresentado vários gráficos em uma só tela com o adicional de ter mais de uma variável em um mesmo gráfico. Ou seja, supondo-se um cenário em que cada bancada requeira no mínimo 3 variáveis em um mesmo gráfico, isso corresponde que em uma única tela, podemos ver 12 variáveis separadas por bancadas (3 variáveis no gráfico em verde, 3 variáveis no gráfico em vermelho, 3 variáveis no gráfico em azul e 3 variáveis no gráfico amarelo). É importante ressaltar que, cada gráfico tem a possibilidade de haver um número de variáveis e comportamento independente uns dos outros, bastando apenas uma personalização na codificação.

A biblioteca Vis (no terminal *npm install vis*) de interface gráfica de grafos foi utiliza e modificada, semelhante ao trabalho do coorientador, para os fins deste trabalho como mostra a

figura 34 e através do link \<https://science-tree.firebaseapp.com/#/person/la6329553180774321>, como uma ferramenta de apresentação do diagrama da bancada selecionada, uma vez que o requisito de projeto é dar a oportunidade ao usuário que manipula a bancada apenas realizar seus controles, sem haver a necessidade de saber todas as rotinas de funcionamento envolvidas. Entretanto, ao invés de ter-se botões como "Acionar tensão da CAN", botões devidamente próximos dos blocos do diagrama das bancadas é desejável nas próximas prototipações. Nota-se também que há no canto inferior esquerdo, o endereço 192.168.7.2:5000/app/tcan correspondente ao botão de "Acionar tensão da CAN"e que é referente a chamada da função "tcan"do Apêndice B.

Figura 34 – Gráfico em tempo real do Plotly em React.



Fonte: Própria autoria.

6 Considerações Finais

Este trabalho procurou contribuir com a área de energias renováveis no tocante ao comando coordenado dos conversores de potência que realizam a interface entre a fonte renovável e a rede elétrica, sendo a produção independente feita sem monitoramento e acionamento manual. Diante dessa motivação, o objetivo geral foi criar uma proposta de um sistema que faça o acoplamento de uma bancada de um sistema de geração de energia elétrica híbrida de pequeno porte, podendo atuar como um modelo a ser seguido para o controle das demais bancadas de geração do laboratório de geração de energia elétrica (LGEE) da UNIFEI Campus Itabira, o qual foi estudado para a realização do projeto.

Tendo em vista este cenário, o campo de conhecimento da Engenharia da Computação pode atuar como uma ferramenta de aplicação prática, levando seus conceitos de desenvolvimento de software, protocolo de comunicação e sistema embarcado no processo de construção do projeto. Dessa forma, possível realizar a estrutura para que o sistema interno e externo ao Beaglebone Black comunicasse com o dispositivo de sinal (DSP) e com o circuito da rede CAN para uma bancada do LGEE no intuito de realizar um protótipo inicial referente aos requisitos que a mesma demanda.

Reforça-se portanto a importância do projeto por ter uma motivação na área que possui deficiências na parte de supervisão e controle, bem como a utilização atuante da parte de Engenharia da Computação ao ser capaz de possuir meios para propor soluções para tal.

6.1 Propostas de continuidade

Espera-se que haja continuidade, tendo como sugestões:

- Adaptar a interface de iteração com o usuário.
- Planejar a codificação das demais bancadas para haver a comunicação de todas em conjunto pelo barramento CAN e DSPs para verificar se há suporte suficiente em termos de velocidade de operação e correto funcionamento com todas as bancadas funcionando em conjunto.
- Realizar o levantamento de requisitos das funcionalidades das outras bancadas.
- Analisar os valores coletados das variáveis das bancadas identificando a qualidade da energia, funcionamento dos equipamentos e prevendo as médias de geração em relação a diminuição do custo na conta da concessionária.
- Emitir relatórios para visualização condensada das variáveis.

Referências

LLAMAS, Luis. Adaptar 3.3V a 5V (y viceversa) em arduino com level shifther. Disponível em: <https://www.luisllamas.es/arduino-level-shifter/>.

Tutorial: Intro to React – React. Disponível em: <https://reactjs.org/tutorial/tutorial.html>. Acesso em: 6 dez. 2018.

OLIVEIRA, Bruno. Aplicação de rede CAN com BeagleBone Black e Python - Embarcados. 2017. Disponível em: <https://www.embarcados.com.br/can-com-beaglebone-black-e-python/>. Acesso em: 26 nov. 2018.

REACT. Tutorial: Intro To React. Disponível em: <https://reactjs.org/tutorial/tutorial.html#functional-components>. Acesso em: 26 nov. 2018.

Módulo Serial Can Bus Tja1050 Arduino Pic Raspberry - R$ 26,50 em Mercado Livre. Disponível em: <https://produto.mercadolivre.com.br/MLB-967776740-modulo-serial-can-bus-tja1050-arduino-pic-raspberry-_JM?quantity=1>. Acesso em: 26 nov. 2018.

BASE2. Você conhece a diferença entre os APIs e Web Services? | Base2. Disponível em: <http://www.base2.com.br/2016/06/08/voce-conhece-a-diferenca-entre-os-apis-e-web-services/>. Acesso em: 26 nov. 2018.

SMYTH, Patrick. Creating Web APIs with Python and Flask. 2018. Disponível em: <https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask>. Acesso em: 26 nov. 2018.

python-can — python-can 3.0.0 documentation. Disponível em: <https://python-can.readthedocs.io/en/3.0.0/>. Acesso em: 26 nov. 2018.

DigitalOcean. How To Setup a Firewall with UFW on an Ubuntu and Debian Cloud Server. Disponível em: <https://www.digitalocean.com/community/tutorials/how-to-setup-a-firewall-with-ufw-on-an-ubuntu-and-debian-cloud-server>. Acesso em: 19 nov. 2018.

ELEMENTZ TEAM. Sharing internet using Network-Over-USB in BeagleBone Black | Random Codes - Elementz Tech Blog. Disponível em: <https://elementztechblog.wordpress.com/2014/12/22/sharing-internet-using-network-over-usb-in-beaglebone-black/>. Acesso em: 19 nov. 2018.

The Flask Mega-Tutorial Part XXIII: Application Programming Interfaces (APIs) - miguelgrinberg.com. Disponível em: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-xxiii-application-programming-interfaces-apis>. Acesso em: 9 nov. 2018.

Designing a RESTful API with Python and Flask - miguelgrinberg.com. Disponível em: <https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask>. Acesso em: 9 nov. 2018.

FERREIRA, Rodrigo. REST: Princípios e boas práticas - Blog da Caelum: desenvolvimento, web, mobile, UX e Scrum. Disponível em: <http://blog.caelum.com.br/rest-principios-e-boas-praticas/>. Acesso em: 8 nov. 2018.

CAN BUS: IMPLEMENTAÇÃO. Disponível em: <http://www.alexag.com.br/CAN_Bus_Parte_3.html>. Acesso em: 5 nov. 2018.

CAN BUS: CONCEITUAÇÃO. Disponível em: <http://www.alexag.com.br/CAN_Bus_Parte_2.html>. Acesso em: 5 nov. 2018.

ARQUITETURAS ELETRO-ELETRÔNICAS: CONCEITUAÇÃO. Disponível em: <http://www.alexag.com.br/CAN_Bus_Parte_1.html>. Acesso em: 5 nov. 2018.

MICROCHIP. Introduction to Controller Area Network (CAN). . [S.l: s.n.], [S.d.]. Disponível em: <https://www.microchip.com/stellent/groups/SiteComm_sg/documents/DeviceDoc/en558265.pdf>. Acesso em: 5 nov. 2018.

NASCIMENTO, Lucas de Camargo. PROTOCOLO DE COMUNICAÇÃO CAN E SUAS APLICAÇÕES NA INDÚSTRIA AUTOMOBILÍSTICA. . Itatiba: [s.n.], 2006. Disponível em: <http://lyceumonline.usf.edu.br/salavirtual/documentos/1598.pdf>. Acesso em: 21 out. 2018.

PROTOCOLO DE COMUNICAÇÕES CAN 3.1 SISTEMAS EM REDE vs vs vs vs SISTEMAS CENTRALIZADOS. . [S.l: s.n.], [S.d.]. Disponível em: <https://web.fe.up.pt/~ee99058/projecto/pdf/Can.pdf>. Acesso em: 21 out. 2018.

CAN communication tutorial, using simulated CAN bus — sgframework 0.2.3 documentation. Disponível em: <https://sgframework.readthedocs.io/en/latest/cantutorial.html>. Acesso em: 19 out. 2018.

DE SOUZA, Paulo Vitor; HERRERA, Christian. Estudo e Elaboração de uma Rede CAN para Aplicação em um Sistema Automotivo. . [S.l: s.n.], 2013. Disponível em: <http://www.divinopolis.cefetmg.br/wp-content/uploads/sites/8/2017/02/TCC-Paulo-Vitor-de-Souza.pdf>. Acesso em: 19 out. 2018.

CAN BUS: CONCEITUAÇÃO. Disponível em: <http://www.alexag.com.br/CAN_Bus_Parte_2.html>. Acesso em: 19 out. 2018.

Rede CAN - Sistema de transmissão de dados em rede entre os módulos de controles veiculares. Disponível em: <https://www.oficinabrasil.com.br/noticia/tecnicas/rede-can-sistema-de-transmissao-de-dados-em-rede-entre-os-modulos-de-controles-veiculares>. Acesso em: 19 out. 2018.

Rede CAN - O que é e como funciona - Dicas. Disponível em: <http://dicas.webautomotivo.com.br/rede-can-o-que-e-e-como-funciona/>. Acesso em: 19 out. 2018.

SN65HVD233-HT. . [S.l: s.n.], 2008. Disponível em: <http://www.ti.com/lit/ds/symlink/sn65hvd233-ht.pdf>. Acesso em: 30 jul. 2018.

RENEW ENERGIA. Painel Business Intelligence – Crescimento do mercado de GD no Brasil – Dados ANEEL | RENEW ENERGIAS RENOVÁVEIS LTDA. Disponível em: <http://renewenergia.com.br/portalbianeel/>. Acesso em: 14 dez. 2017.

THOMSEN, Adilson. Introdução ao BeagleBone Black - FilipeFlop. 2014. Disponível em: <https://www.filipeflop.com/blog/beaglebone-black-rev-c/>. Acesso em: 5 dez. 2017.

TMS320x2833x, 2823x System Control and Interrupts Reference Guide. 2007. Disponível em: <http://www.ti.com/lit/ug/sprufb0d/sprufb0d.pdf>. Acesso em: 4 dez. 2017.

Silicon Errata. 2007. Disponível em: <http://www.ti.com/lit/er/sprz272k/sprz272k.pdf>. Acesso em: 4 dez. 2017.

PYTHON BRASIL. Perguntas Frequentes SobrePython - PythonBrasil. Disponível em: <https://wiki.python.org.br/PerguntasFrequentes/SobrePython#Como_se_pronuncia_Python.3F>. Acesso em: 4 dez. 2017.

NUNES, Rafael Astuto Arouche; DE ALBUQUERQUE, Marcelo Portes. Digital Signal Processing - Processador Digital de Sinais - DSP - CBPF/MCT - CAT. Disponível em: <http://www.cbpf.br/~rastuto/>. Acesso em: 4 dez. 2017.

PYTHON SOFTWARE FOUNDATION. Python-Can — Python-Can 2.0.0-Alpha.2 Documentation. Disponível em: <https://python-can.readthedocs.io/en/latest/#>. Acesso em: 4 dez. 2017.

LOBO, Marcelo Programação Orientada a Objetos. 2007. Disponível em: <https://www.devmedia.com.br/introducao-ao-desenvolvimento-web-com-python/6552>. Acesso em: 2 dez. 2017.

ZHANG, Yang et al. Design of DSP-based high-power digital solar array simulator. 19 dez. 2013, [S.l.]: International Society for Optics and Photonics, 19 dez. 2013. p. 90460I. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2037268>. Acesso em: 2 dez. 2017.

VIEIRA, Leonardo Pereira. Guia de Utilização do Controlador Digital de Sinal TMS320F28335. 2013. 69 f. Universidade Federal de Juiz de Fora, 2013. Disponível em: <http://www.ufjf.br/prh-pb214/files/2014/01/Leonardo-Guia-de-utilização-do-controlador-digital-de-sinal-TMS320F28335.pdf>. Acesso em: 2 dez. 2017.

MAURICI, Alessandro Barreiros. Suporte a Redes CAN para Aplicações Embarcadas. 2005.

DOS REIS, Geovane Luciano. Projeto e construção de um conversor monofásico em ponte h multicelular entrelaçado para geração fotovoltaica e eólica de pequeno porte. 2017. 165 f. Universidade Federal de Minas Gerais, 2017. Disponível em: <https://ppgee.ufmg.br/defesas/1389M.PDF>. Acesso em: 30 nov. 2017.

LOBACCARO, Gabriele; CARLUCCI, Salvatore; LÖFSTRÖM, Erica. A review of systems and technologies for smart homes and smart grids. Energies, v. 9, n. 5, p. 1–33, 2016.

SÁENZ, Jacobo et al. Open and Low-Cost Virtual and Remote Labs on Control Engineering. IEEE Access, v. 3, p. 805–814, 2015.

RESTREPO-ZAMBRANO, José Alex et al. Experimental framework for laboratory scale microgrids. Revista Facultad de Ingeniería Universidad de Antioquia, n. 81, p. 9–23, 2016. Disponível em: <http://aprendeenlinea.udea.edu.co/revistas/index.php/ingenieria/article/view/26436>.

FILHO, Fernando Coelho et al. Resenha Energética Brasileira - Exercício de 2016. . [S.l: s.n.], 2017. Disponível em: <www.mme.gov.br>. Acesso em: 16 nov. 2017.

MANUR, Ashray; VENKATARAMANAN, Giri; SEHLOFF, David. Simple electric utility platform: A hardware/software solution for operating emergent microgrids. Applied Energy, 2017. Disponível em: <https://ac.els-cdn.com/S0306261917309467/1-s2.0-S0306261917309467-main.pdf?_tid=32c08c50-cacc-11e7-95c2-00000aacb360&acdnat=1510836583_35b9ee882f510948593bacdf705f7bed>. Acesso em: 16 nov. 2017.

FEDRIZZI, Marcus; SORIA, Julio. Application of a single-board computer as a low-cost pulse generator. Measurement Science and Technology, v. 26, n. 9, p. 095302, 2015. Disponível em: <http://stacks.iop.org/0957-0233/26/i=9/a=095302?key=crossref.6fd5d69cac81de4307a4a748f76d44b1>. Acesso em: 16 nov. 2017.

DA SILVA, Rodrigo Côrrea; DE MARCHI NETO, Ismael; SEIFERT, Stephan Silva. Electricity supply security and the future role of renewable energy sources in Brazil. Renewable and Sustainable Energy Reviews, v. 59, p. 328–341, 2016. Disponível em: <http://dx.doi.org/10.1016/j.rser.2016.01.001>.

DE MORAIS, Luciano Cardoso. Estudo sobre o panorama da energia elétrica no Brasil e Tendências futuras. 2015. 128 f. Universidade Estadual Paulista, 2015. Disponível em: <repositorio.unesp.br/bitstream/handle/11449/132645/000852309.pdf?sequence=1>. Acesso em: 12 nov. 2017.

GUERRA, José Baltazar Salgueirinho Osório de Andrade et al. Future scenarios and trends in energy generation in Brazil: Supply and demand and mitigation forecasts. Journal of Cleaner Production, v. 103, p. 197–210, 2015. Disponível em: <https://ac.els-cdn.com/S095965261401021X/1-s2.0-S095965261401021X-main.pdf?_tid=0964ce36-c7f0-11e7-ab1c-00000aacb360&acdnat=1510522122_7caf17e7fcf2e7599eaad8875e37afb9>. Acesso em: 12 nov. 2017.

BERTELLI, Guilherme Pereira Marchioro. Sistema de Controle e Supervisão da Estação de Processos MPS PA via Plataforma BeagleBone Sistema de Controle e Supervisão da Estação de Processos MPS PA via Plataforma BeagleBone. 2015.

GMBH, Robert Bosch. F2833x Controller Area Network Introduction. p. 1–56, [S.d.].

FERREIRA, Mauro Pacheco et al. Monitoramento Online como Ferramenta para Otimização da Manutenção de Geradores: Uma Tecnologia 100% Nacional em Arquitetura Distribuída. Xi Simpósio De Automação De Sistemas Elétricos, p. 1–10, 2015. Disponível em:

<http://www.aqtech.com.br/casos_artigos/AQTech_Informe_Tecnico_XI_SIMPASE_2015.pdf>. Acesso em: 10 nov. 2017.

URZÊDA, Claiton Cesar De. Software scada como plataforma para a racionalização inteligente de energia elétrica em automação predial. 2006. Universidade Federal de Brasília, 2006. Disponível em: <http://repositorio.unb.br/handle/10482/3285>. Acesso em: 10 nov. 2017.

ANEEL. Resolução Normativa n° 481 de 17 de Abril de 2012.[S.l: s.n.]. Disponível em: <http://www.aneel.gov.br/cedoc/ren2012481.pdf>.

BARBOSA FILHO, Wilson Pereira; AZEVEDO, Abílio César soares De. Geração Distribuída : Vantagens E Desvantagens. II Simpósio de Estudos e Pesquisas em Ciências Ambientais na Amazônia, v. 1, p. 1–11, 2014. Disponível em: <http://www.feam.br/images/stories/arquivos/mudnacaclimatica/2014/artigo_gd.pdf>. Acesso em: 16 ago. 2017.

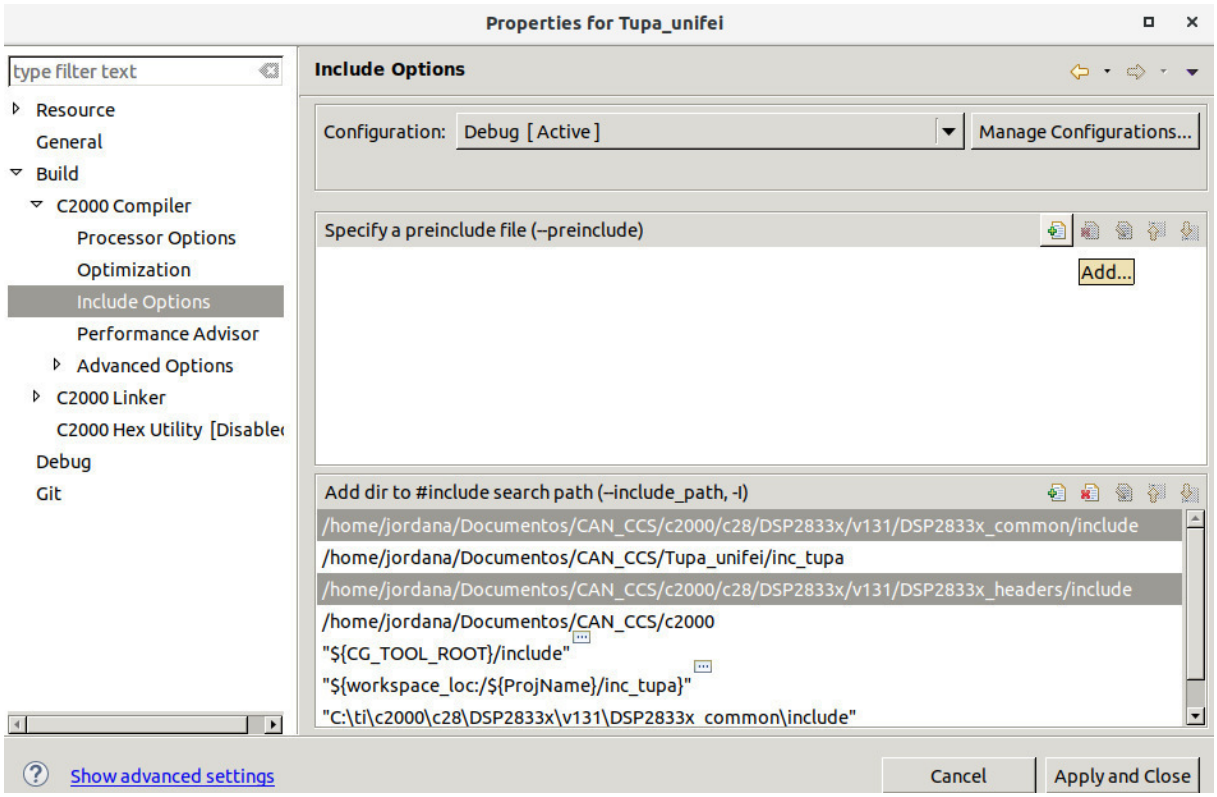
PINHO, João Tavares et al. Sistemas Híbridos: Soluções Energéticas para a Amazônia. p. 396, 2008. Disponível em: <http://luzparatodos.mme.gov.br/luzparatodos/downloads/Solucoes_Energeticas_para_a_Amazonia_Hibrido.pdf>. Acesso em: 14 ago. 2017.

Apêndices

APÊNDICE A – Instalação e importação de projeto no Code Composer Studio 8.2.0

A inclusão das bibliotecas dá-se clicando com o lado direito do mouse sobre o projeto, escolhendo a última opção "Properties/Propriedades". Ao abrir a nova janela como na imagem 35, seguir por *Build > C200 Compiler > Include Options* e clicar sobre o ícone de adicionar (primeiro item do lado direito, identificado por "Add..."). A inclusão dos arquivos da pasta *common*, *headers* e da pasta c200 por completa é necessária. É recomendado que a pasta de inclusão esteja no mesmo diretório do projeto. As inclusões podem ser verificadas nos itens em cinza da imagem.

Figura 35 – Inclusão das bibliotecas no projeto do CCS v8.2.0.

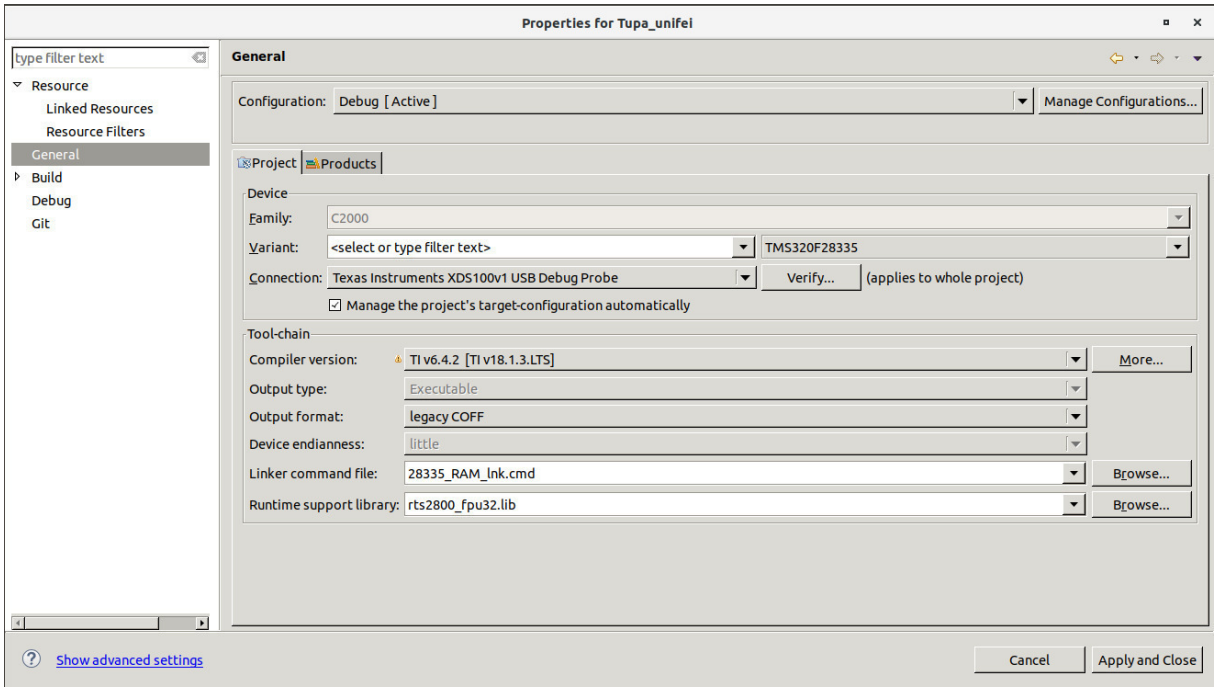


Fonte: Própria autoria.

Identificar e configurar as propriedades do dispositivo em que consta o 28335, são realizadas na mesma janela de "Properties/Propriedades", na opção "General". A fim de apurar se houve a correta seleção, o botão "Verify"deve ser pressionado em retorno positivo ou negativo para tal. Em caso negativo, note se o kit está ligado, encaixado corretamente, com cabo

e tensão adequada, ou se há outras possibilidades em tela que o atendam melhor no Code Composer.

Figura 36 – Configuração do dispositivo no Code Composer Studio.

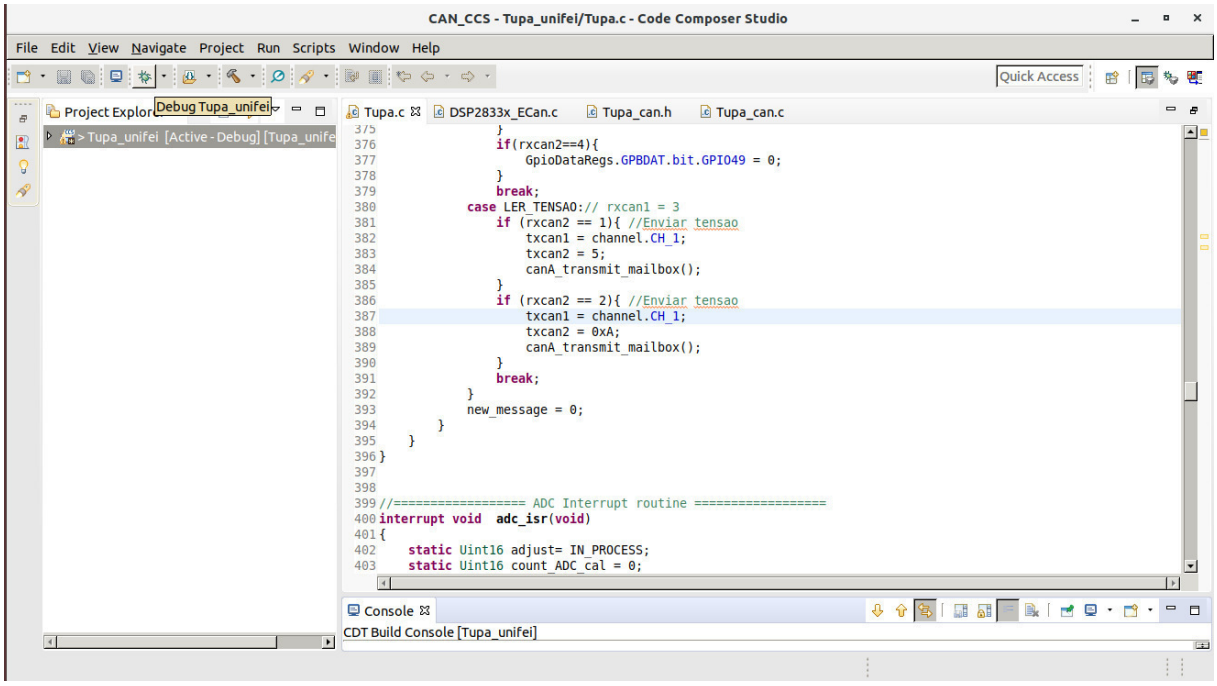


Fonte: Própria autoria.

Após a etapa de ajustes o projeto pode ser compilado e executado por meio do botão com o ícone de um inseto, identificado de forma sobressalente no canto superior esquerdo com a etiqueta em amarelo de "Debug" na figura 37.

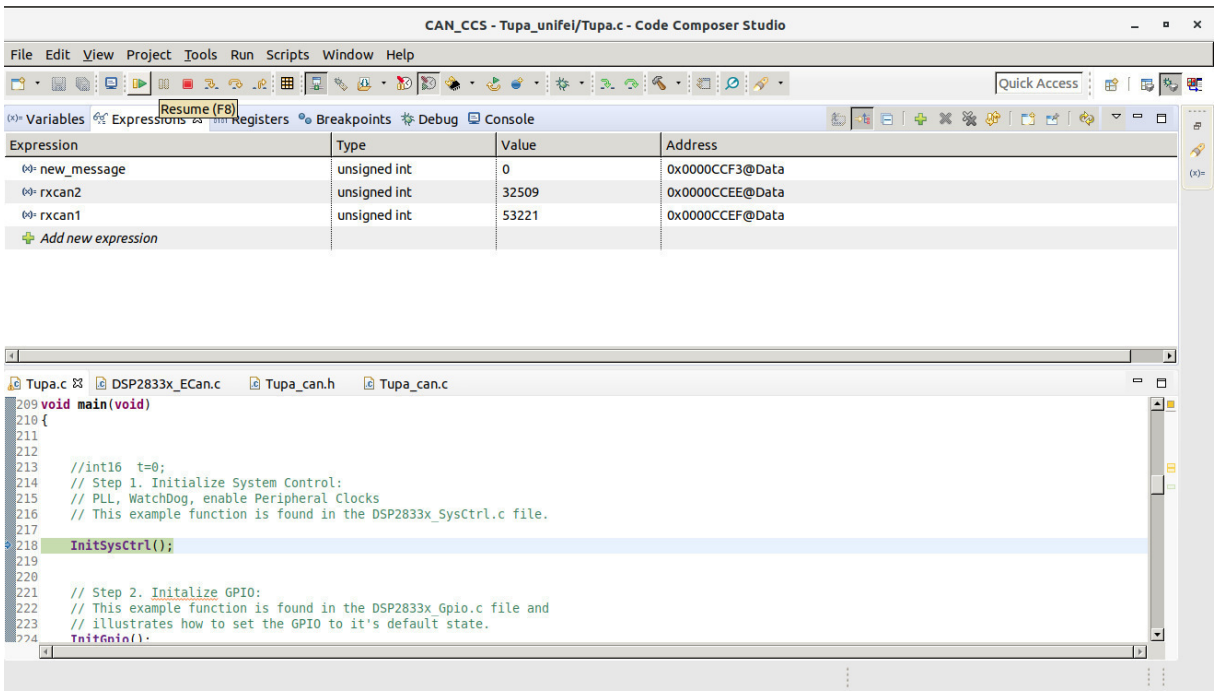
Em modo de execução, para que o projeto de fato faça isso, o ícone de uma seta em verde, identificado de forma sobressalente no canto superior esquerdo com a etiqueta em amarelo de "Resume(F8)" na figura 38 deve ser pressionado. Nota-se que para visualizar variáveis ou expressões, a aba "Expressions"(área de tabela com cores altenadas em branco e cinza na 38) pode ser utilizada.

Figura 37 – Área de trabalho do Code Composer pronto para execução.



Fonte: Própria autoria.

Figura 38 – Code Composer com o código em modo de pré execução com as variáveis e expressões escolhidas para visualização.



Fonte: Própria autoria.

APÊNDICE B – Algoritmo bancada.py

```
# -*- coding: utf-8 -*-
#!flask/bin/python
#!/usr/bin/python
from flask import Flask,jsonify, abort, request, make_response, url_for
from flask_httpauth import HTTPBasicAuth
app = Flask(__name__, static_url_path = )
auth = HTTPBasicAuth()
import can
import time
bus = can.interface.Bus(channel='can0', bustype='socketcan_native')
id = 255#
#Definindo o conteudo das mensagens
cont = [3,1]
@app.route('/')
def index():
    return "Controle - Tupá - Laboratório de Geração de Energia Elétrica"
@auth.get_password
def get_password(username):
    if username == 'lgee':
        return 'lgee'
    return None
@auth.error_handler
def unauthorized():
    return make_response(json    ify(  'Erro 401': 'Acesso não autorizado'  ), 403)
@app.errorhandler(400)
def not_found(error):
    return make_response(json    ify(  'Erro 400': 'Má requisição'  ), 400)
@app.errorhandler(404)
def not_found(error):
    return make_response(json    ify(  'Erro 404': 'Página não encontrada'  ), 404)
# TESTE CAN - BOTÃO ----------------------------------------------------
@app.route('/app/tcan', methods = ['GET'])
#@auth.login_required
def tcan():
    msg = can.Message(arbitration_id = id,data=cont)
#Criando a mensagem que será enviada na rede
    bus.send(msg) #Enviado para a rede
    print("_________ Mensagem enviada_______")
    return jsonify(  'tasks': map(make_public_task, tasks)  )
#--------------------------------------------------------------------
```

```
tasks = [
{
    'id': 1,
    'title': Bancada 1,
    'description': Geração eólica,
    'done': False
    },
    {
    'id': 2,
    'title': Bancada 2,
    'description': Geração solar,
    'done': False
    }
]
def make_public_task(task):
new_task = {}
for field in task:
    if field == 'id':
        new_task['uri'] = url_for('get_task', task_id = task['id'], _external = True)
    else:
        new_task[field] = task[field]
return new_task
@app.route('/todo/api/v1.0/tasks', methods = ['GET'])
@auth.login_required
def get_tasks():
    return jsonify( { 'tasks': map(make_public_task, tasks) } )
@app.route('/todo/api/v1.0/tasks/<int:task_id>', methods = ['GET'])
@auth.login_required
def get_task(task_id):
task = filter(lambda t: t['id'] == task_id, tasks)
    if len(task) == 0:
        abort(404)
    return jsonify(  'task': make_public_task(task[0])  )
@app.route('/todo/api/v1.0/tasks', methods = ['POST'])
@auth.login_required
def create_task():
    if not request.json or not 'title' in request.json:
        abort(400)
    task = {
        'id': tasks[-1]['id'] + 1,
        'title': request.json['title'],
        'description': request.json.get('description', ),
        'done': False
    }
    tasks.append(task)
    returnjsonify(  'task': make_public_task(task)  ), 201
```

```
@app.route('/todo/api/v1.0/tasks/<int:task_id>', methods = ['PUT'])
@auth.login_required
def update_task(task_id):
    task = filter(lambda t: t['id'] == task_id, tasks)
    if len(task) == 0:
        abort(404)
    if not request.json:
        abort(400)
    if 'title' in request.json and type(request.json['title']) != unicode:
    abort(400)
    if 'description' in request.json and type(request.json['description']) is not unicode:
        abort(400)
    if 'done' in request.json and type(request.json['done']) is not bool:
        abort(400)
    task[0]['title'] = request.json.get('title', task[0]['title'])
    task[0]['description'] = request.json.get('description', task[0]['description'])
    task[0]['done'] = request.json.get('done', task[0]['done'])
    return jsonify( 'task': make_public_task(task[0]) )
@app.route('/todo/api/v1.0/tasks/<int:task_id>', methods = ['DELETE'])
@auth.login_required
def delete_task(task_id):
    task = filter(lambda t: t['id'] == task_id, tasks)
    if len(task) == 0:
        abort(404)
    tasks.remove(task[0])
    return jsonify( 'result': True )
    if __name__ == "__main__":
        app.run(debug=True, host='0.0.0.0')
```

Anexos

ANEXO A – Algoritmo rx.py

Algoritmo 1: RX.PY

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
# Esse programa monitorará a rede CAN, mostrando todas as mensagens recebidas.
# Também verificará se a mensagem possuem um id conhecido ou não
#Importando bibliotecas
import can  Biblioteca do python-can
import time
print("Recebi RX")
#Definir qual controlador vamos usar
bus = can.interface.Bus(channel='can0', bustype='socketcan_native')
#Selecionamos o can0 como o controlador que monitorará as mensagens
#Definindo um id conhecido
id = 0x123
#Main
while True:
        message = bus.recv()
        #Fica esperando receber mensagem no barramento CAN
        print("Recebemos uma mensagem")
             if message.arbitration_id == id:
            #verifica se o id da mensagem é o mesmo que o id definido
                  print("A mensagem possui um id conhecido!")
             else:
                  print("A mensagem possui um id desconhecido!")
                  #O loop a seguir irá formatar o conteudo do vetor para
                  que possa ser apresentado na tela da forma de hexadecimal
        s=[]
        for i in range(len(message.data)):
       #loop que percorrerá todo o vetor message.data
             s.append(hex(message.data[i]))
             #Convertendo o valor em hexadecimal e adicionando ao fim do vetor s
        print(s)
        return(s)
```

ANEXO B – Algoritmo tx.py

Algoritmo 2: TX.PY

```
1  #-*- coding: utf-8 -*-
2  # Esse programa enviará periodicamente uma mensagem para a rede can
3  #Importando bibliotecas
4  import can
5  import time
6  #Definir qual controlador vamos usar
7  bus = can.interface.Bus(channel='can0', bustype='socketcan_native')
8  #Selecionamos o can0 como o controlador que monitorará as mensagens
9  #Definindo um id para nossas mensagens
10 id = 0x123
11 #Definindo o conteudo das mensagens
12 cont = [0xf5,0x23,0x1A]
13 #Main
14 while True:
15         msg = can.Message(arbitration_id = id,data=cont)
16         #Criando a mensagem que será enviada na rede
17         bus.send(msg) #Enviado para a rede
18         print("Mensagem enviada")
19         time.sleep(5)
20         #espera 5 segundos antes de enviar a mesma mensagem
```

ANEXO C – Algoritmo tupa.c

Algoritmo 3: TUPA.C

```
1  while(1){ //================== GSI on and off control ==================
2      if(new_message){
3        switch(rxcan1){
4          case 1:
5               if(rxcan2==1){
6                  GpioDataRegs.GPADAT.bit.GPIO9 = 1;
7               }
8               if (rxcan2==2){
9                  GpioDataRegs.GPADAT.bit.GPIO11 = 1;
10              }
11              if (rxcan2==3){
12                 GpioDataRegs.GPBDAT.bit.GPIO34 = 1;
13              }
14              if(rxcan2==4){
15                 GpioDataRegs.GPBDAT.bit.GPIO49 = 1; //entradas_red.Vrede;
16              }
17                 break;
18           case 2:
19              if(rxcan2==1){
20                 GpioDataRegs.GPADAT.bit.GPIO9 = 0;
21              }
22              if (rxcan2==2){
23                 GpioDataRegs.GPADAT.bit.GPIO11 = 0;
24              }
25              if (rxcan2==3){
26                 GpioDataRegs.GPBDAT.bit.GPIO34 = 0;
27              }
28              if(rxcan2==4){
29                 GpioDataRegs.GPBDAT.bit.GPIO49 = 0;
30              }
31              break;
```

```
1  [1]
2              case LER_TENSAO: // rxcan1 = 3
3                 if (rxcan2 == 1){ //Enviar tensao
4                    txcan1 = channel.CH_1;
5                    txcan2 = 5;
6                    canA_transmit_mailbox();
7                 }
8                 if (rxcan2 == 2){ //Enviar tensao
9                    txcan1 = channel.CH_1;
10                   txcan2 = 0xA;
11                   canA_transmit_mailbox();
12                }
13                break;
14             }
15             new_message = 0;
16          }
17       }
18    }
```

ANEXO D – Algoritmo Plotly React

Algoritmo 4: PLOTLY EM JAVASCRIPT

```
1  import React from 'react';
2  import Plot from 'react-plotly.js';
3  export default class PlotLyGraphic extends React.Component {
4    state = {
5      line1: {
6        x: [-3, -2, -1],
7        y: [1, 2, 3],
8        name: 'Line 1'
9      },
10     line2: {
11       x: [1, 2, 3],
12       y: [-3, -2, -1],
13       name: 'Line 2'
14     },
15     layout: {
16       datarevision: 0,
17     },
18       revision: 0,
19   }
20     rand = () => parseInt(Math.random() * 10, 10);
21     increaseGraphic = () => {
22       const { line1, line2, layout } = this.state;
23       line1.x.push(this.rand());
24       line1.y.push(this.rand());
25       line2.x.push(this.rand());
26       line2.y.push(this.rand());
27       this.setState( revision: this.state.revision + 1 );
28       layout.datarevision = this.state.revision + 1;
29     }
```

```
1  [1]
2                render() {
3                  return (
4                    <div>
5                    <Plot
6                    data={[
7                      this.state.line1,
8                      this.state.line2,
9                      ]}
10                     layout={this.state.layout}
11                     revision={this.state.revision}
12                     graphDiv="graph"
13                     />
14                     </div>
15                   );
16                 }
17               }
```