



UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA MECÂNICA

BRUNO HENRIQUE HUFFENBAECHER MARQUES DE OLIVEIRA

**Desenvolvimento de uma rede CAN para um padrão
atômico de frequência portátil**

TRABALHO DE CONCLUSÃO DE CURSO

São Carlos
2016

BRUNO HENRIQUE HUFFENBAECHER MARQUES DE OLIVEIRA

**Desenvolvimento de uma rede CAN para um padrão
atômico de frequência portátil**

Trabalho de Conclusão de Curso, apresentado à Escola de Engenharia de São Carlos, como parte dos requisitos para graduação em Engenharia Mecatrônica

Orientador: Prof. Dr. Daniel Varela Magalhães

São Carlos
2016

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

H898d

Huffenbaecher Marques de Oliveira, Bruno Henrique
Desenvolvimento de uma rede CAN para um padrão
atômico de frequência portátil / Bruno Henrique
Huffenbaecher Marques de Oliveira; orientador Daniel
Varela Magalhães. São Carlos, 2016.

Monografia (Graduação em Engenharia Mecatrônica) --
Escola de Engenharia de São Carlos da Universidade de
São Paulo, 2016.

1. Rede CAN. 2. BeagleBone Black. 3. Padrão de
frequência. 4. Relógio Atômico. 5. Python. I. Título.

Bruno Henrique Huffenbaecher Marques de Oliveira

Desenvolvimento de uma rede CAN para um padrão atômico de frequência portátil

IMPORTANTE: ESSE É APENAS UM TEXTO DE EXEMPLO DE FOLHA DE APROVAÇÃO. VOCÊ DEVERÁ SOLICITAR UMA FOLHA DE APROVAÇÃO PARA SEU TRABALHO NA SECRETARIA DO SEU CURSO (OU DEPARTAMENTO).

Trabalho aprovado. São Carlos , DATA DA APROVAÇÃO:

Prof. Dr. Daniel Varela Magalhães
Orientador

Professor
Convidado 1

Professor
Convidado 2

São Carlos
2016

Agradecimentos

Agradeço primeiramente a minha família, pelo amor e apoio, além de possibilitar meus estudos na USP.

Ao meu orientador, Professor Daniel, pela oportunidade de projeto, orientação e sugestões.

Ao Rodrigo 'Che' Pechoneri, por toda confiança e ajuda no desenvolvimento deste projeto, com seus conhecimentos e ideias e dedicação.

A minha namorada, Raiza, pelo amor, apoio e carinho, acreditando sempre em meu potencial.

A dois grandes amigos, Mateus e Guilherme, por todas as sugestões e ideias de melhorias para o projeto, bem como todos os momentos únicos proporcionados ao longo da trajetória acadêmica.

A todos os professores, que contribuiram diretamente ou indiretamente com minha formação e com a realização dos meus sonhos e de inúmeros alunos que já passaram pela universidade.

Ao LIEPO, por proporcionar a estrutura necessária para o desenvolvimento do projeto, bem como um ambiente descontraído e ao mesmo tempo enriquecedor.

A todos os membros que contribuem diariamente com a comunidade *open source*

A todo grupo de mecatrônica da Escola de Engenharia de São Carlos.

Ao grupo de óptica do Instituto de Física de São Carlos.

A Universidade de São Paulo.

*If I have seen further it is by standing on the
shoulders of Giants - Isaac Newton*

Resumo

OLIVEIRA, B. H. H. M. **Desenvolvimento de uma rede CAN para um padrão atômico de frequência portátil.** 2016. 63f. Trabalho de Conclusão de Curso - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2016

O presente trabalho de conclusão de curso tem por escopo o estudo do protocolo CAN e sua implementação em um padrão atômico de frequência, chamado comumente de relógio atômico. A rede CAN será utilizada como meio de troca de informações entre os módulos que compõem padrão atômico de frequência portátil, sendo neste trabalho projetada especificamente para este fim, atendendo todos os seus requisitos de projeto. Para isso, inicialmente a rede é estabelecida entre os dois controladores CAN nativos da Beaglebone Black, realizando todos os procedimentos necessários para habilitá-los bem como a familiarização com os conceitos do protocolo, os formatos de mensagem e envio de conteúdo. Através da programação em Python, programas são desenvolvidos com o objetivo de atuar diretamente na rede, seja por meio do envio e recebimento de mensagens e do seu pré e pós das mesmas, gerando códigos genéricos para serem adaptados e aplicados aos demais módulos do relógio. Como validação final será testada a comunicação entre a Beaglebone e o microcontrolador Mbed, uma vez que os estes dois hardwares serão efetivamente usados no projeto alvo.

Palavras-Chave: Rede CAN. BeagleBone Black. Python. Relógio Atômico.

Abstract

OLIVEIRA, B. H. H. M. **Development of a CAN protocol to a portable atomic frequency standard** 2016. 63p. Undergraduate Thesis - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2016

The following thesis has as scope the study and implementation of the CAN bus in an atomic frequency standard, also known as atomic clock. The CAN bus will be used as means of exchanging information among the modules that make up the system. As such, it will be projected to this specific application, meeting all its requirements. Initially, the network it is established between the two Beaglebone onboard native CAN controllers, executing all the necessary steps to enable them and also getting familiar with the protocol basic concepts, message standards and sending data. Through Python programming, programs were developed to act directly on the network, not only sending and receiving messages but also pre- and post-processing them, creating generic source codes, which can be adapted and applied to all the atomic clock modules. As a final project validation, the CAN communication between the Beaglebone and de Mbed controller will be tested, since both hardware are used on the target project.

Keywords: CAN bus. Beaglebone Black. Python. Atomic Clock.

Listas de ilustrações

Figura 1 – Diagrama das funções essenciais de um padrão de frequência (malha de controle)	23
Figura 2 – Principais subsistemas que constituem o relógio de átomos frios	24
Figura 3 – Configuração geral do relógio portátil	25
Figura 4 – Arquitetura Central	26
Figura 5 – Arquitetura distribuída	27
Figura 6 – Padrão de mensagem CAN	28
Figura 7 – Barramento CAN	30
Figura 8 – Sinais no Barramento CAN	31
Figura 9 – Beaglebone Black	33
Figura 10 – Interface Cloud9	36
Figura 11 – Transceiver MCP2551	38
Figura 12 – Conversor de Nível Lógico	38
Figura 13 – Microcontrolador LCP1768	39
Figura 14 – Osciloscópio Minipa MO-2025	40
Figura 15 – Circuito Completo	42
Figura 16 – Circuito Completo - Mbed e BeagleBone	44
Figura 17 – Circuito real - Beaglebone e rede CAN	45
Figura 18 – Circuito real - BeagleBone e Mbed	46
Figura 19 – Cape desenvolvida com o circuito CAN	47
Figura 20 – Testes com a CAN raw	47
Figura 21 – Medida do osciloscópio durante a transmissão de mensagens	48
Figura 22 – Testes iniciais com Python	49
Figura 23 – Arquivo de texto utilizado para validação	50
Figura 24 – Execução dos programas em sua versão final	51
Figura 25 – Candump durante o programa txdadosv2.py	52
Figura 26 – Envio por Mbed e recebimento por BBB	52
Figura 27 – Envio por BBB e recebimento por Mbed	53
Figura 28 – Execução dos programas completos	53

Lista de tabelas

Tabela 1 – Relação de comprimento do barramento e taxa de transmissão	31
Tabela 2 – Especificações de Hardware - Beaglebone Black	35
Tabela 3 – Conexão - Pinos e componentes	43

Lista de abreviaturas e siglas

A/D	Analog/Digital
ARM	Advanced RISC Machine
BBB	BeagleBone Black
CAN	Controller Area Network
ECU	Electronic Control Unit
eMMC	Embedded MultimediaCard
GPIO	General Purpose Input/Output
GPMC	General Purpose Memory Controller
GPS	Global Positioning System
HDMI	High Definition Multimedia Interface
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
I/O	Input/Ouput
LCD	Liquid Crystal Display
MMC	MultiMediaCard
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
PCB	Printed Circuit Board
PWM	Pulse-Width Modulation
SD	Secure Digital (cartão)
SPI	Serial Peripheral Interface
SSH	Secure Shell
UART	Universal Asynchronous Receiver/Transmitter

Sumário

1	INTRODUÇÃO	21
1.1	Contexto	21
1.2	Motivação	21
1.3	Objetivos	22
2	REVISÃO BIBLIOGRÁFICA	23
2.1	Padrão de Frequência Atômica	23
2.1.1	Padrão de Frequência compacto	24
2.2	Rede CAN	25
2.2.1	Arquitetura dos sistemas	26
2.2.2	Propriedades da Rede	27
2.2.3	Formato das mensagens	28
2.2.4	O Barramento CAN	29
2.2.5	Controladores e Transceivers CAN	31
3	MATERIAIS E MÉTODOS	33
3.1	BeagleBone Black	33
3.1.1	A interface cloud9	36
3.2	Python	36
3.3	Transceiver CAN	37
3.4	Conversor de nível lógico	38
3.5	Mbed	39
3.6	Osciloscópio	40
3.7	Abordagem de projeto	40
3.7.1	Familiarização com a plataforma	40
3.7.2	Habilitação dos controladores CAN	41
3.7.3	Controladores CAN e Python	41
3.8	Círcuito completo	42
4	RESULTADOS E DISCUSSÕES	45
4.1	Circuitos Reais	45
4.2	Testes com a CAN ‘RAW’	47
4.3	Testes com Python	49
4.4	Testes com Mbed	52
4.5	Criação de artigo	54
5	CONCLUSÃO	55

5.1	Considerações Finais	55
5.2	Trabalhos Futuros	56
	Referências	57
	APÊNDICES	59
	Apêndice A	61

1 Introdução

1.1 Contexto

Medir o tempo, ainda que de forma imprecisa, é algo inerente ao ser humano. Das inúmeras maneiras de se controlar e medir o tempo, os padrões de frequência atômica, mais popularmente conhecidos como relógios atômicos, desempenham a tarefa com exatidão inigualável, e por isso estes equipamentos têm sido utilizados nas mais diversas aplicações, que vão desde navegação (como o GPS) até tecnologias militares(AHMED et al., 2008).

Pesquisas vem sendo desenvolvidas por diversos países para a obtenção de padrões atômicos com precisões ainda melhores que as atuais(MAGALHÃES, 2004). Atualmente, encontra-se em desenvolvimento um padrão de frequência atômica portátil, que tem por objetivo manter as características do modelo atual, ou seja, manter a confiabilidade de exatidão e precisão de curto e longo período de tempo, proporcionando a mobilidade do sistema(PECHONERI, 2013).

O projeto deste sistema móvel conta com uma arquitetura distribuída, composta por diversos subsistemas, responsável pela execução das etapas de medição, acionamento de sensores e processamento de dados. Para o sucesso do projeto, é necessário que haja uma rede de comunicação altamente confiável, que permitirá a troca de informação entre módulos e ao mesmo tempo permita que eles sejam gerenciados por uma interface que também se comunique com externamente com o usuário.

Assim, para que a comunicação seja eficaz, tem-se a necessidade da implementação de uma rede CAN, um barramento de comunicação serial desenvolvido pela BOSCH, inicialmente aplicado à industria automotiva mas atualmente usado amplamente em diversas áreas tecnológicas(SOUSA, 2000).

Outro desafio do relógio de átomos móvel é embarcar todos os subsistemas, de forma a reduzir o espaço físico ocupado por eles. Portanto, além da otimização e compactação dos módulos, é necessário considerar também o computador que gerenciará todos os dados e sistemas e os hardwares de processamento de cada subsistema.

1.2 Motivação

É evidente a necessidade de mensurar o tempo. Com um mundo cada vez mais tecnológico e automatizado, a exatidão se torna indispensável na grande maioria dos

processos e produtos. Atualmente, apenas países desenvolvidos contam com esta tecnologia, e o desenvolvimento nacional de um equipamento portátil expandirá as aplicações dos padrões de frequência, além de trazer benefícios não só para o país, mas também para todo o hemisfério sul.

O design do projeto tem por característica o uso de uma rede compatível com a arquitetura distribuída de seus subsistemas. Portanto, faz-se necessário o uso de uma rede de comunicação que atenda a esses requisitos. A rede CAN é escolhida pois apresenta a robustez e confiabilidade necessárias para a aplicação.

Mais do que simplesmente implementar uma rede de comunicação, este trabalho será a solução utilizada para a comunicação entre os subsistemas do relógio atômico, e por isso, precisa estar alinhado com todas as necessidades e características do projeto.

Portanto, serão necessários estudos acerca do sistema de comunicação e das plataformas embarcadas, o que contribuirá com o avanço pessoal e científico do autor, visto que os temas do projeto são pouco abordados no curso de graduação, desenvolvendo assim novos conhecimentos. No mais, é uma chance única de participar em um projeto de extrema importância e grande potencial para o Brasil.

1.3 Objetivos

O presente trabalho tem como objetivo implementar uma rede CAN com um propósito específico - um projeto de um relógio de átomos frios portátil. Assim, faz-se necessário:

- 1) Habilitar e tornar funcional os controladores CAN presentes na plataforma embarcada BeagleBone Black;
- 2) Desenvolver programas utilizando uma linguagem de programação de alto nível, que permitam a comunicação com a rede CAN, voltados para a aplicação no projeto-alvo;
- 3) Realizar com sucesso a comunicação, através da rede CAN, entre a Beaglebone Black e o microcontrolador Mbed;

2 Revisão Bibliográfica

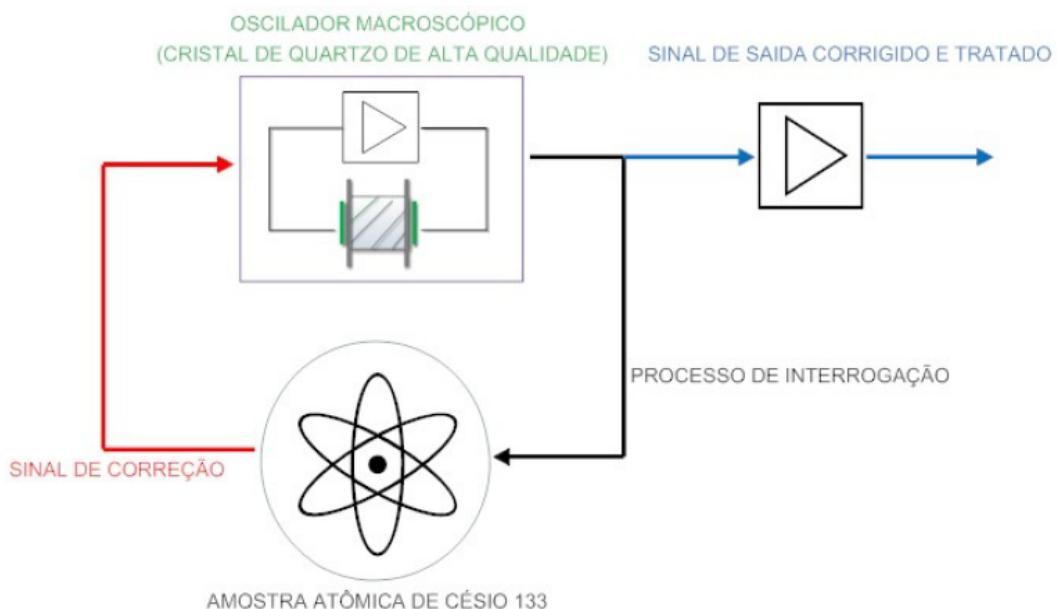
2.1 Padrão de Frequência Atômica

Ainda que não seja obrigatório entender o funcionamento detalhado de um relógio atômico de átomos frios, faz-se necessário compreender seu princípio de funcionamento e sua estrutura, para facilitar o desenvolvimento deste projeto.

Um padrão atômico de frequência fornece um sinal de frequência que está intimamente relacionado ao nível de energia dos átomos. Aplicando um sinal de micro-ondas que esteja ressonante com a frequência de transição do átomo utilizado (como por exemplo o átomo de Césio), é produzido um sinal de resposta que caracteriza o padrão de frequência (PECHONERI, 2013).

Existem diversos processos que são realizados pelo sistema do relógio para condicionar e manipular a amostra atômica e extrair os sinais de frequência necessários. Seu funcionamento ocorre em ciclos, sendo que a cada ciclo ocorre uma correção de sinal, em uma malha de controle, buscando corrigir e tratar o sinal. A figura 1 ilustra este processo.

Figura 1 – Diagrama das funções essenciais de um padrão de frequência (malha de controle)



Fonte: (PECHONERI,2013)

2.1.1 Padrão de Frequência compacto

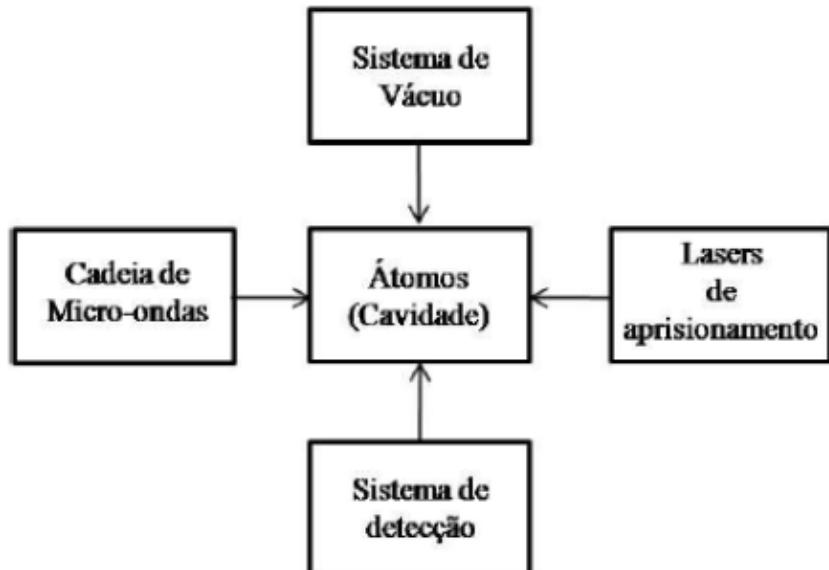
O sistema de um padrão de frequência compacto baseia-se muito no relógio de átomos frios comum, com a diferença de que todos os processos ocorrem dentro de um mesmo espaço, uma cavidade de microondas ressoante com a transição do relógio(MULLER, 2010).

O projeto do relógio atômico portátil visa, além de manter as características e resultados de um padrão de frequência compacto, tornar o equipamento móvel, melhorando sua operacionalidade e robustez expandindo assim seu campo de aplicações(PECHONERI, 2013).

Uma vez que a plataforma será móvel , ela pode ser embarcada juntamente com sua aplicação, facilitando a calibração de sensores e servindo como uma referência de tempo para outros equipamentos. A portabilidade também permite que o equipamento seja deslocado para centros de pesquisa ou industrias nos quais haja necessidade de calibração de equipamentos ou comparações de sinais com outros padrões.

A figura 2 a seguir mostra os principais subsistemas que constituem o relógio.

Figura 2 – Principais subsistemas que constituem o relógio de átomos frios



Fonte: (Muller,2010)

Com o objetivo de redução do volume de seus componentes e equipamentos, foi iniciado um projeto para otimizar os subsistemas, tanto em robustez como em espaço ocupado. Tal mudança criou a necessidade de uma estrutura de controle e monitoramento com arquitetura distribuída, onde cada subsistema possua o funcionamento individual, sendo apenas configurados e monitorados por uma interface de supervisão.

Assim, podemos ver na figura 3 a seguir, a estrutura de controle proposta para o relógio portátil

Figura 3 – Configuração geral do relógio portátil



Fonte: (Pechoneri,2013)

É a partir dessa estrutura que se definiu alguns dos requisitos para este trabalho. A rede responsável pela comunicação será uma rede CAN, escolhida por suas características, entre elas a robustez e confiabilidade. Para gerenciar e configurar os demais subsistemas foi escolhida a plataforma embarcada BeagleBone Black (Beaglebone.org, 2016). Cada subsistema também possui uma unidade lógica responsável por pelo seu controle. No projeto, está unidade será o microcontrolador LCP1768, conhecido como Mbed(MBED, 2016)

Ressalta-se aqui que estas definições foram previamente escolhidas pelo projetista do relógio portátil, cabendo ao autor deste projeto trabalhar para integrar e implementar estas definições, agora requisitos de projeto, conforme os objetivos já definidos.

2.2 Rede CAN

O barramento de comunicação CAN (Controller Area Network), foi desenvolvido pela BOSCH nos anos 80 com o intuito de reduzir consideravelmente o número de condutores utilizados nos sistemas elétricos da indústria automotiva(SOUSA, 2000).

Esta rede foi desenvolvida para ser um sistema de transmissão de mensagens multi-mestre, com velocidades de transmissão de até 1 Mbps. Diferentemente de redes como a Ethernet, ela não transmite blocos grandes de dados de um ponto ao outro. Ao invés disso, são enviadas mensagens curtas, geralmente contendo informações

como parâmetros e dados pontuais, para todas as unidades conectadas à rede(Texas Instruments, 2002).

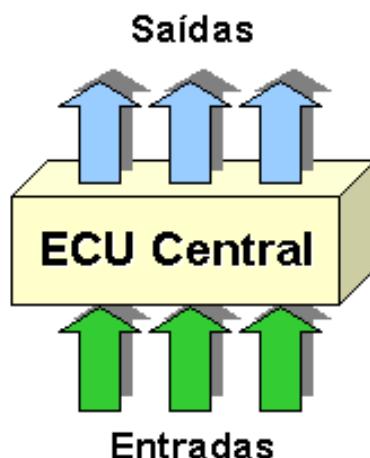
2.2.1 Arquitetura dos sistemas

Atualmente, os sistemas elétricos são compostos por diversas unidades de controle eletrônicos (ECU). Cada uma destas unidades é responsável pelo controle e processamento de funções específicas dentro do sistema completo. Porém é necessário que haja algum tipo de comunicação entre elas para que haja o compartilhamento de informações. A forma que essas unidades de controle são organizadas e interligadas é denominada arquitetura de sistemas(GUIMARÃES, 2003).

Basicamente, existem dois tipos de arquitetura possíveis - A centralizada e a distribuída.

Na centralizada, todos os módulos estão conectados à uma unidade central, que é responsável por receber todas as informações de entrada, processar os dados, e distribuir comandos e ações de saída para as demais unidades.

Figura 4 – Arquitetura Central



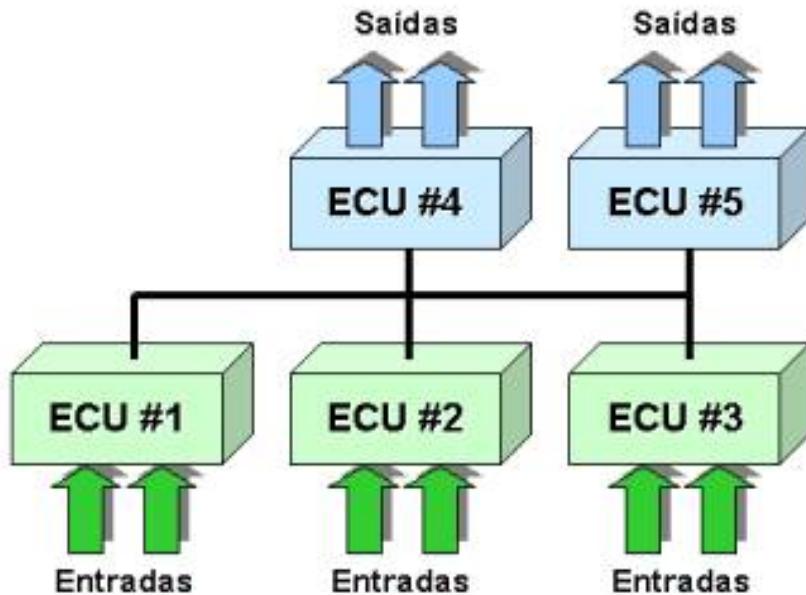
Fonte: (GUIMARÃES,2003)

Este é um tipo de arquitetura mais simples e fácil de ser implementada, porém requer que a unidade central seja capaz de processar todo o volume de dados. Dessa forma, um sistema que possui grande quantidade de sensores, atuadores e etc, gerando um enorme volume de dados, precisará de um módulo central com grande capacidade de processamento. Vale ressaltar também a enorme quantidade de cabos necessários para ligar todos os módulos à unidade central.

Já a arquitetura distribuída apresenta todas as ECU's conectadas entre si, entre um barramento de comunicação comum, compartilhando as informações entre todos os módulos simultaneamente. Não há um processamento central, visto que cada unidade

realiza parte do processamento, compartilhando e recebendo os dados das demais unidades.

Figura 5 – Arquitetura distribuída



Fonte: (GUIMARÃES,2003)

Este tipo de arquitetura apresenta a vantagem de distribuir o processamento, não necessitando de um hardware muito potente para cada unidade, já que o volume de dados a serem processados por unidade é menor. Destaca-se também a redução no número de cabos utilizados para a interligação, já que agora todas as unidades estão conectadas à um barramento comum.

Entretanto, para que a comunicação entre as ECU's seja eficiente é necessário a utilização de um protocolo de comunicação, que dita as regras de comunicação no barramento. Isto pode se tornar um problema, pois de acordo com cada situação específica, a implementação pode tornar-se complexa, deixando o projeto final mais caro.

Por isso, é necessário uma análise detalhada do projeto para que a decisão do tipo de arquitetura trabalhe a favor do mesmo. Para o projeto do padrão de frequência portátil, será utilizado a arquitetura distribuída, pelo fato dos módulos trabalharem com o processamento distribuído, reduzindo a necessidade de hardwares poderosos e que poderiam dificultar a portabilidade do projeto.

2.2.2 Propriedades da Rede

A rede CAN apresenta diversas propriedades. Entre elas, podemos citar as mais relevantes:

- Priorização das mensagens
- Capacidade *multicast* e multi-mestre
- Tempo de latência reduzido
- Sistema de detecção e sinalização de erros
- Retransmissão automática das mensagens em espera
- Simplicidade, baixo custo e robustez (alta tolerância a ruído)

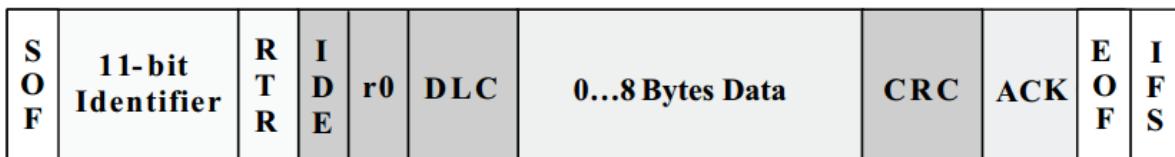
As características listadas fazem com que este protocolo de comunicação seja ideal para sistemas que necessitem de confiabilidade na transmissão das mensagens, uma vez que a rede possui maneiras de detectar e gerenciar os erros de comunicação que possam surgir durante a transmissão, garantindo assim que todas as mensagens sejam enviadas(BOSCH, 1991).

A característica multimestre permite que todas as unidades tenham acesso ao barramento e possam transmitir conteúdo ao mesmo, respeitando a priorização das mensagens, garantindo assim que duas mensagens não sejam transmitidas simultaneamente.

Devido a sua estrutura, que será apresentada nas próximas subseções, é uma rede de baixo custo de implementação e também não exige que as ECU's conectadas à ela possuam conexões diretas com as demais, bastando apenas possuir um controlador CAN conectado ao barramento(Texas Instruments, 2002)

2.2.3 Formato das mensagens

Figura 6 – Padrão de mensagem CAN



Fonte: (TEXAS INSTRUMENTS,2002)

A figura acima ilustra o formato das mensagens que serão transmitidas na rede. O significado e função dos campos das figura 6 são:

- SOF - *Start of Frame* - Bit marca o início da mensagem a ser transmitida
- Identificador de 11bits. O identificador estabelece a prioridade de transmissão da mensagem. Quando menor o valor, maior sua prioridade

- RTR - *Remote Transmission Request* - é o bit que sinaliza quando uma informação é requisitada de outro nó da rede
- IDE - *Identifier Extension* - Bit que sinaliza se a mensagem é transmitida com o identificador de 11 bits ou 29bits(versão extendida)
- r0 - bit de reserva, atualmente sem uso.
- DLC - *Data Length Code* - Armazena o número de bytes de dados que está sendo transmitido
- Data - Campo composto de até 8 bytes, que armazena o conteúdo da mensagem a ser enviada. Os 8 bytes são também referenciados como *frames*
- CRC - *Cyclic Redundance Check* - Contém o numero de bits transmitidos para verificação de erros
- ACK - Também faz parte do sistema de detecção de erros, indicando que a mensagem não possui falhas
- EOF - *End of frame* - Indica o fim da mensagem e também é utilizado para detecção de erros
- IFS - *Interframe Space* - Contém o tempo utilizado pelo controlador CAN para mover um frame recebido corretamente para a posição correta no buffer de mensagem.

Como pode-se notar, a mensagem é composta de alguns mecanismos para detecção e correção de erro, bem como garantir a arbitragem e transmissão de todas as mensagens.

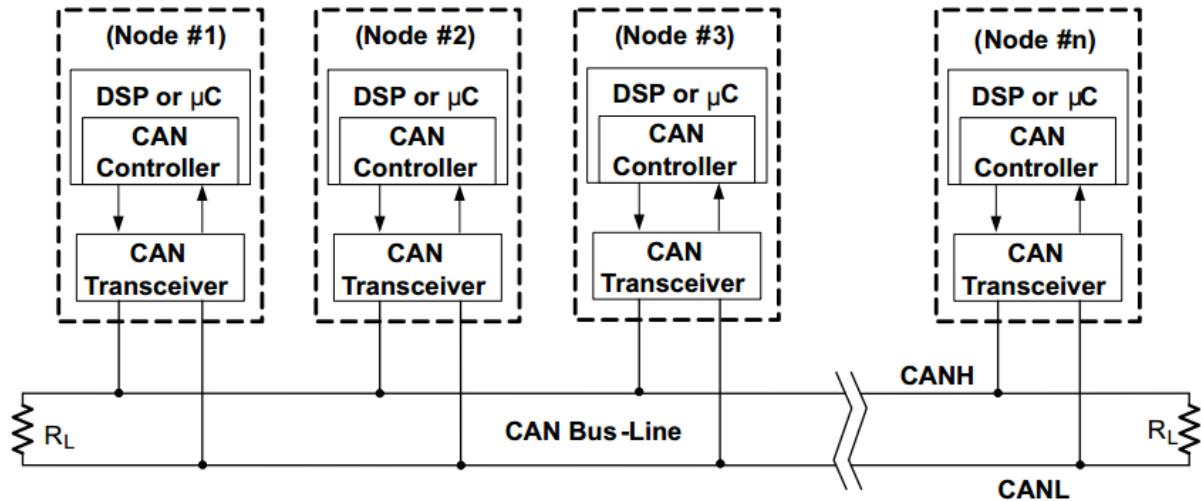
O identificador das mensagens compõem um papel fundamental na rede. Como já citado, ele é utilizado para a priorização das mensagens. Além disso, através do identificador, é possível que cada ECU filtre as mensagens recebidas a partir dele, processando apenas as que são relevantes.

Para isso, é necessário que cada ECU tenha uma espécie de dicionário próprio, contendo a lista de identificadores relevantes e também o conteúdo esperado associado a certo identificador.

2.2.4 O Barramento CAN

Um exemplo do barramento CAN com as ECU's(chamadas aqui de nós da rede) pode ser visto na figura 7:

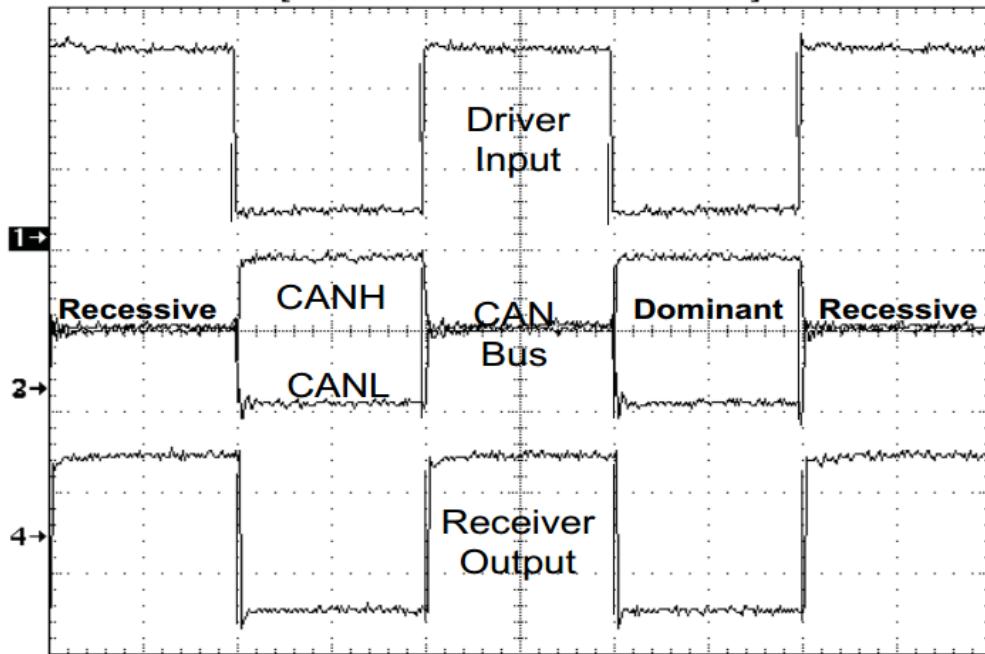
Figura 7 – Barramento CAN



Fonte: (TEXAS INSTRUMENTS, 2002)

O barramento é composto por um par trançado de cabos, denominados CANH(CAN HIGH) e CANL (CAN LOW). Nas extremidades do barramento temos resistores de 120Ohms para manter um nível constante de impedância e evitar a reflexão de sinal entre as vias.

A transmissão de mensagens é feita de forma diferencial no barramento, significando que os sinais de bits altos ou baixo são baseados na diferença de tensão entre as duas vias. Um bit dominante é caracterizado com um diferencial de 2V, enquanto para o bit recessivo não há diferença de tensão entre as duas linhas, como mostrado na figura 8 a seguir.

Figura 8 – Sinais no Barramento CAN

Fonte: (TEXAS INSTRUMENTS,2002)

Apesar do barramento transmitir a taxas de até 1Mb/s, é necessário considerar algumas características e fenômenos envolvidos na transmissão de dados por uma rede. Quando uma rede é muito extensa, os atrasos na propagação de sinais começam a ter efeito significante. Por isso, a taxa de transmissão em uma rede é inversamente proporcional ao seu comprimento(NATALE, 2008). A tabela a seguir mostra uma relação do comprimento do barramento CAN e sua respectiva taxa de transmissão:

Tabela 1 – Relação de comprimento do barramento e taxa de transmissão

Comprimento do barramento	Taxa de transmissão
1Mb/s	25m
500kb/s	100m
250kb/s	250m
125kb/s	500m
10kb/s	5000m

Fonte: (NATALE,2008)

2.2.5 Controladores e Transceivers CAN

Todos os nós são compostos de três elementos - a unidade de processamento, um transceiver e um controlador CAN. Para os usuários iniciantes, é sempre levantada a questão da necessidade do uso do transceiver, ao invés de uma conexão direta do

controlador ao barramento.

Fato é que essas duas unidades possuem funções bem distintas. O controlador CAN é responsável por criar as mensagens, formata-las no padrão do protocolo de comunicação e também por realizar a interface com a ECU.

Já o transceiver possui como função principal fazer a conversão de sinais que representam os bits no barramento CAN para os sinais que os representam no controlador CAN(NATALE, 2008). Dessa forma, o transceiver protege o controlador de cargas excessivas provenientes do barramento.

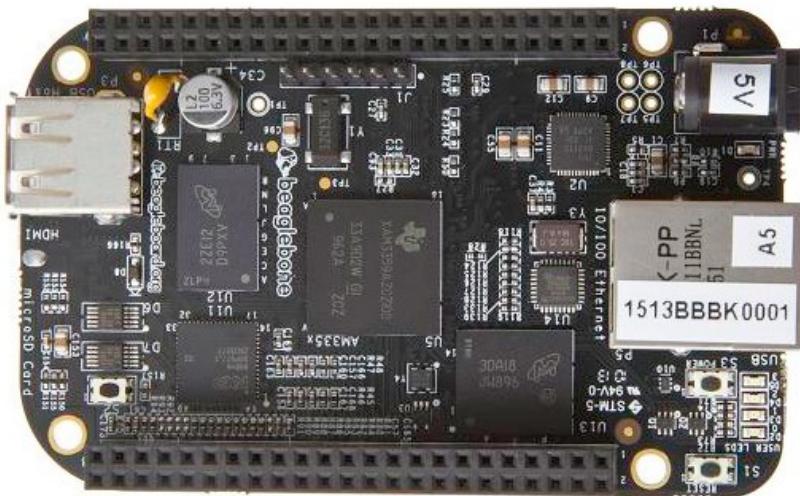
3 Materiais e Métodos

Nesta seção serão apresentados e especificados os materiais e métodos utilizados no projeto, assim como a montagem e construção dos protótipos.

3.1 BeagleBone Black

A Beaglebone Black(Beaglebone.org, 2016) é uma plataforma de desenvolvimento *open-source* de baixo custo (preço sugerido de 55 dólares) voltada para estudantes, desenvolvedores e hobistas. Uma espécie de minicomputador, de tamanho reduzido porém de alto poder computacional. Atualmente é usada em diversas áreas de projetos, principalmente relacionados à área de robótica e automação(Texas Instruments,).

Figura 9 – Beaglebone Black



Fonte: Beagleboard.org

Desenvolvida pela *Texas Instruments*, seu baixo custo, associado à característica *open-source* e ao alto poder computacional fizeram com que o hardware se popularizasse, aumentando as comunidades online - locais onde se reúnem amadores e profissionais para a troca de conhecimento - que facilitaram seu uso e permitiram que projetos mais complexos fossem desenvolvidos.

A plataforma permite a instalação de diversos sistemas operacionais, sendo mais comum o Linux, em suas distribuições Debian, Angstrom e Ubuntu. A tabela 2 a seguir mostra as principais características da BBB:

Tabela 2 – Especificações de Hardware - Beaglebone Black

Processador	AM3358 ARM Cortex-A8
Velocidade do processador	1GHz
Pinos analógicos	7
Pinos digitais	65 (3.3V)
Memória	512MB (800Mhz x 16), 4GB on-board eMMC storage, microSD card slot
USB	mini USB, USB port
Video	micro HDMI, cape add-ons
Áudio	micro HDMI, cape add-ons
Interfaces suportadas	4x UART, 8x PWM, LCD, GPMC, MMC, 2x SPI, 2xI2C, A/D Converter, 2x CAN Bus, 4 Timers

Fonte: BeagleBone.org

A tabela acima nos mostra uma característica importante. A plataforma possui dois controladores CAN nativos, o que facilitará a execução de testes, como será abordado em uma seção posterior.

Para este projeto, foi instalado o sistema operacional Linux em sua distribuição Debian 7.9(Wheezy), disponível para *download* diretamente do site oficial(Beaglebone.org, 2016). A escolha da versão foi dada por preferência do projetista e também pela mesma utilizar como padrão a versão do Kernel 3.8.13, versão necessária para a instalação dos módulos e execução de *overlays* para habilitar os controladores CAN.

A Beaglebone pode ser alimentada através da porta USB ou também utilizando uma fonte externa 5V/1A. A alimentação USB tem a vantagem de também ser uma das formas de comunicação do usuário com a plataforma, que pode ser feita através de um acesso remoto SSH.

Outra vantagem importante é a possibilidade de usar o sistema operacional diretamente no cartão SD. Isso permite a diminuição dos danos causados por usuários inexperientes ou até mesmo danos provenientes de testes avançados. Em caso de falha definitiva, basta apenas gravar a imagem do sistema operacional no cartão novamente, diferente de estar gravada na memória flash, o que poderia resultar em perda definitiva do hardware.

Durante toda a fase de implementação, instalação de bibliotecas e módulos, o sistema operacional foi executado a partir de um cartão de memória MIXZA TOHAOLL 32Gb. Com o projeto já concretizado, o conteúdo do cartão foi passado para a memória

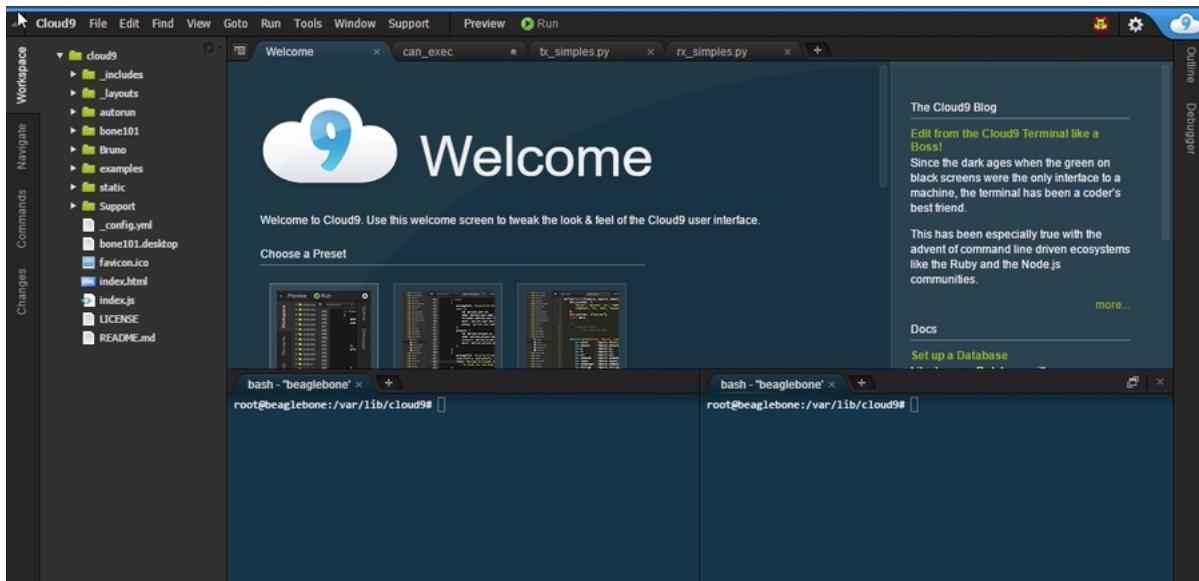
eMMC através de um tutorial desenvolvido por(BARROS, 2015).

O acesso para a comunicação SSH pode ser feito através de programas como o Putty(PUTTY, 2016), que simula um terminal, permitindo ao usuário realizar todo acesso através dele, ou também do cloud9, uma IDE compatível com a BBB.

3.1.1 A interface cloud9

Cloud9(CLOUD9, 2016) é uma IDE com inúmeros recursos, que fornece um ambiente amigável e de fácil manuseio. Ela possui uma integração com a beaglebone, e desta forma, torna a comunicação usuário-plataforma simples e eficiente. A figura 10 ilustra a IDE.

Figura 10 – Interface Cloud9



Fonte: do autor

Através desta IDE é possível enviar e receber arquivos da BBB, programar em diversas linguagens de programação, como Python e C, abrir múltiplas janelas de terminal, visualizando todos os processos de forma mais organizada.

Em suma, ela contém todas as funções de um terminal comum, via Putty, e outras mais.

3.2 Python

Python(PYTHON, 2016) é uma linguagem de programação de auto nível, dinâmica e fácil de ser utilizada, tanto por iniciantes quanto por usuários avançados. Apesar de sua simplicidade, é uma linguagem poderosa, sendo aplicada desde desenvolvimento de websites até aplicações numéricas e científicas.

Criada em 1991, encontra-se atualmente na versão 3.5. Possui uma imensa comunidade de usuários, que desenvolvem bibliotecas para aplicações específicas, além de constantemente aperfeiçoarem a linguagem, com melhorias de performance e atualizações.

Inicialmente, devido à falta de experiência do autor com a linguagem, foi necessário ser feita uma familiarização com a linguagem, suas estruturas, funções básicas, lógica de programação entre outros.

Utilizou-se como guia o livro de(DAWSON, 2010). O livro aborda conteúdos básicos e avançados, suficientes para a compreensão dos conhecimentos necessários para a realização deste projeto.

A versão utilizada é a Python 2.7, por ser o padrão instalado na versão utilizada do sistema operacional escolhido. É uma versão popular entre os usuários, e por isso possui um grande suporte da comunidade.

A principal biblioteca utilizada é a *python-can*(POWELL, 2016). Ela permite a integração do Python com os controladores CAN. Assim, é possível a criação de programas que utilizam os controladores, seja no envio ou recebimento de mensagens da rede CAN, pois a biblioteca possui funções prontas para tal interação.

A instalação e todo o conteúdo técnico da biblioteca pode ser encontrado em(POWELL, 2016). Porém fez-se uso também de exemplos criados por outros usuários nos fóruns da comunidade Python. Em particular destacam-se os exemplos do blog Skpang(PANG, 2016) pela sua didática e simplicidade.

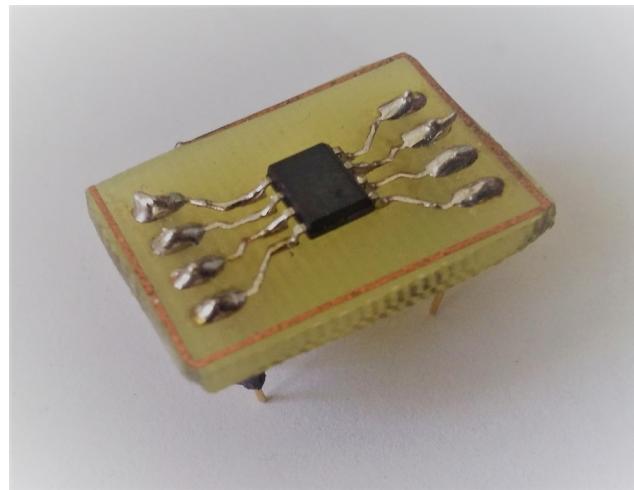
3.3 Transceiver CAN

A interface entre os controladores CAN da Beaglebone e a rede física é feita através do uso de dois transceivers Microchip MCP2551(MICROCHIP, 2010), sendo que cada transceiver é utilizado com um controlador e seus canais de I/O(RX/TX).

A escolha do modelo deu-se a disponibilidade nacional do componente, aliado ao seu baixo custo. Uma característica fundamental que influenciou no projeto mais tarde é o fato deste ser um transceiver 5V, enquanto a BBB trabalha com 3.3V. A solução para este conflito é explicada em seções posteriores.

Os transceiver são soldados em uma placa simples de circuito impresso, como mostrado na figura 11. Essa placa permite o uso de pinos, facilitando o contato e fixação em circuitos maiores, através da utilização de uma protoboard, por exemplo.

Figura 11 – Transceiver MCP2551



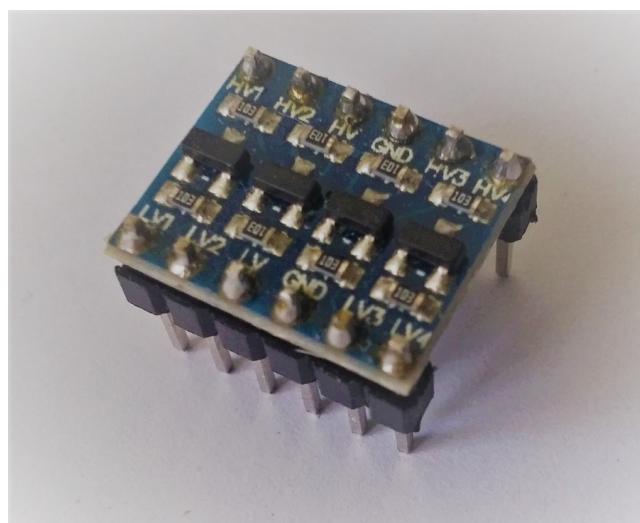
Fonte: Do autor

3.4 Conversor de nível lógico

Devido à escolha do modelo de transceiver, que trabalha com nível lógico de 5V, enquanto a plataforma Beaglebone utiliza 3.3V, faz-se necessário a utilização de conversores de nível lógico de tensão.

No projeto, foi utilizado um módulo pronto, que contém 4 MOSFETs BSS138(FAIRCHILD, 2005), montados de forma conveniente em uma PCB, com pinos acoplados de contato acoplados, facilitando seu uso em projetos. A figura 12 mostra o conversor utilizado no projeto.

Figura 12 – Conversor de Nível Lógico



Fonte: Do autor

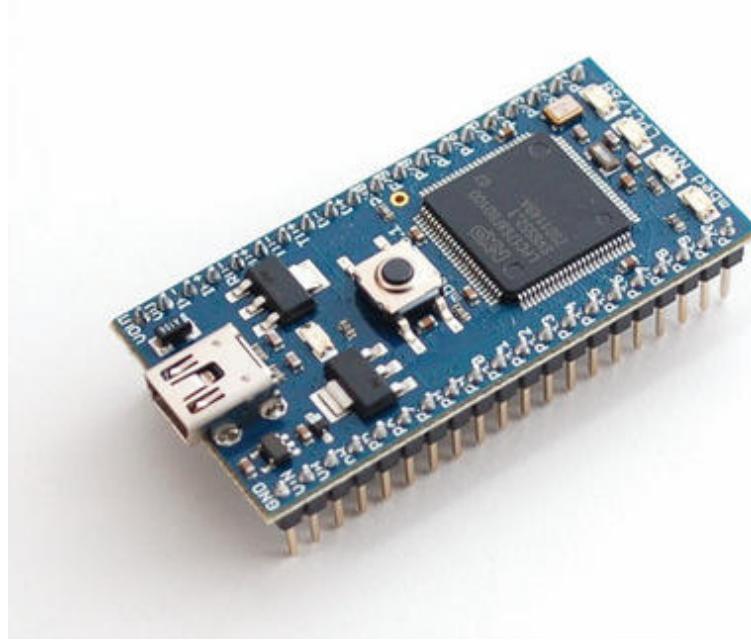
3.5 Mbed

Para a validação final do projeto foi utilizado o microcontrolador LCP1768(NXP, 2015), também conhecido como microcontrolador Mbed. Como já citado anteriormente, esse microcontrolador será responsável por gerenciar os dados dos módulos do relógio atômico, e por tanto, testes de compatibilidade e performance fazem-se necessários para a validação final deste projeto.

Um dos recursos importantes desta plataforma são os controladores CAN que a mesma possui *onboard*, totalmente compatíveis com os padrões e especificações CAN. Ainda é necessário o uso de transceivers CAN entre a rede e os controladores, mas diferente da BBB, aqui utiliza-se 5V como nível lógico alto, e portanto não é necessário o uso de conversores lógicos.

Através da IDE online disponível no próprio site(MBED, 2016) foram criados programas utilizando a linguagem C, baseados na mesma lógica de programação dos programas desenvolvidos em Python, utilizados na Beaglebone. Foi necessário apenas traduzir as sintaxes e funções de uma linguagem para outra.

Figura 13 – Microcontrolador LCP1768



Fonte: <http://blackboxlab.wixsite.com/>

Inicialmente testou-se a conexão BeagleBone - Mbed, através do envio do envio de mensagens simples entre os dois (utilizando o terminal e os comandos *candump*, *cansend* e *cangen*), e posteriormente utilizando as versões finais dos programas desenvolvidos em Python e C.

3.6 Osciloscópio

Para medição dos sinais, no intuito de verificar o funcionamento dos controladores e *transceivers*, foi utilizado um osciloscópio, modelo MO-2025 da Minipa como mostrado na figura 14 abaixo:

Figura 14 – Osciloscópio Minipa MO-2025



Fonte: Minipa.com.br

A utilização do equipamento também facilitou na resolução de problemas, quando era necessário verificar e identificar componentes defeituosos. Isso foi feito percorrendo o circuito e analisando os sinais em cada conexão. As medidas era feitas tomando por referência o sinal *GND(Ground)*.

3.7 Abordagem de projeto

Como citado em seções anteriores, a maioria dos testes funcionais do projeto foi feito utilizando a Beaglebone. As subseções a seguir explicam a abordagem utilizada no projeto.

3.7.1 Familiarização com a plataforma

Antes de utilizar qualquer recurso avançado da Beaglebone Black, foi necessária uma introdução aos conceitos mais básicos da placa. Para isso, utilizou-se dos conhecimentos disponíveis em (SANTOS; PERESTRELO, 2015) e (GRIMMETT, 2013). As duas bibliografias apresentam um conteúdo introdutório e avançado sobre o tema, além da exemplificação didática de aplicações.

3.7.2 Habilitação dos controladores CAN

Apesar da plataforma possuir os dois controladores CAN embarcados, é necessária uma configuração prévia para sua utilização. Isso acontece pois os mesmos pinos usados pelos controladores são compartilhados com outras funções, como GPIO e I2C.

Então, foi feito um *overlay* - uma espécie de sobreposição e reorganização - para os dois drivers CAN e seus respectivos pinos, indicando que os controladores terão prioridade sobre as demais funções. Os *scripts* foram retirados de um blog que aborda sobre o assunto(Embedded Things, 2013), porém algumas das informações estava desatualizadas, sendo necessário fazer pequenas modificações nos códigos e processos.

Também foi necessário a instalação do pacote *can-utils*(ELINUX, 2015), pacote que contém ferramentas e funções para o Linux, que fazem a interface com os drivers CAN.

Desta forma, foi possível então utilizar os controladores sem maiores dificuldades, com base em informações e exemplos do próprio fabricante(Texas Instruments, 2012). Inicialmente, utilizou-se as funções CAN de forma “crua”, ou seja, direto dos terminais do sistema. Esta é uma maneira verificar o funcionamento dos controladores, através do envio e recebimento de mensagens. Porém, ainda que útil, é uma maneira pouco aplicável a projetos, pois a integração com algoritmos que processarão o conteúdo da mensagem se torna complexa.

3.7.3 Controladores CAN e Python

Devido a complexidade de integrar o envio/recebimento das mensagens com o processamento do conteúdo, foi utilizado o suporte que o Python dá para a integração com os drivers CAN.

Assim, como já mencionado na seção 3.2, alinhado ao objetivo deste trabalho e a necessidade do projeto que ele será integrado, os programas criados foram limitados à função de comunicação com a rede, tanto de recebimento como envio de mensagens. Entretanto, os algoritmos escritos permitem fácil integração com as funções que processarão os dados e executarão ações baseadas nos mesmos, que são funções particulares de cada módulo do relógio atômico, não cabendo ao autor a escrita destes algoritmos específicos.

Iniciou-se com a criação de algoritmos simples, que enviavam uma mensagem fixa, enquanto outro programa apenas recebia essa mensagem. Esses programas básicos permitiram que se familiarizasse com os formatos de mensagem, estruturas e tipos de suas variáveis, permitindo o avanço na complexidade dos algoritmos.

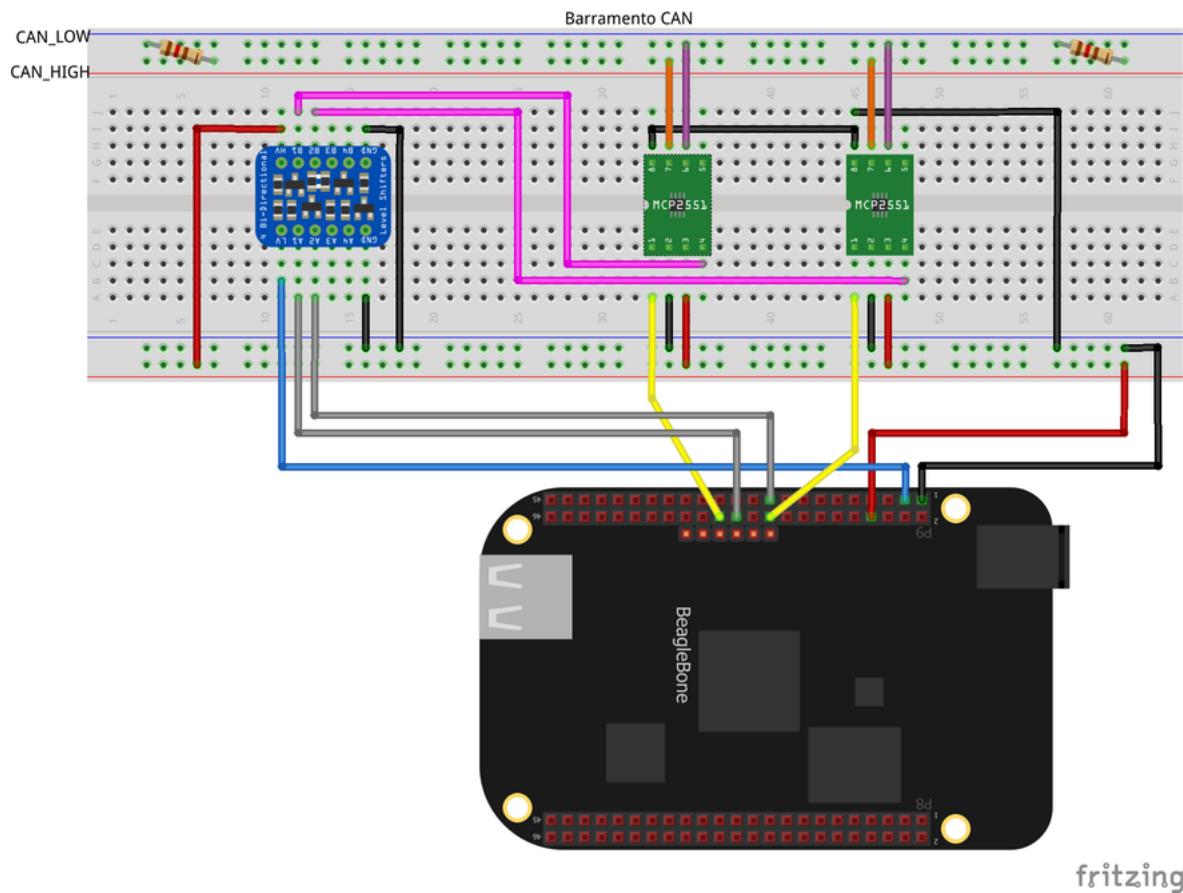
Por fim, foram desenvolvidos 2 programas principais; Um programa responsável por formatar e enviar mensagens na rede e outro pela recepção e pré processamento das informações recebidas. Os programas estão escritos de forma didática e com comentários nos pontos principais, para que se forem necessários pequenos ajustes e modificações, sejam feitos de forma descomplicada.

3.8 Circuito completo

Para que seja possível o teste dos controladores CAN, algoritmos desenvolvidos e testes de validação, é necessário integrar todos os componentes e processos citados anteriormente.

Como já mencionado, a rede CAN precisa de ao menos dois controladores ligados a ela para operar. Assim, se faz necessário integrar os controladores da BBB à rede, e portanto utilizar os transceivers e também os conversores de nível lógico. A figura 15 a seguir ilustra uma representação do circuito, com seus componentes e ligações, desenvolvida utilizando o software livre Fritzing(FRITZING, 2016)

Figura 15 – Circuito Completo



Fonte: Produzido pelo Autor

De uma forma geral, tem-se a rede CAN com seus 2 canais na parte superior da

figura, utilizando as trilhas superiores '+' e '-' (note o uso de dois resistores de 120Ohm em suas extremidades), enquanto a Beaglebone e seus controladores CAN na parte inferior. Ao centro, temos os transceivers fazendo a interface entre eles.

A tabela 3 a seguir relaciona os pinos e conexões dos componentes do circuito:

Tabela 3 – Conexão - Pinos e componentes

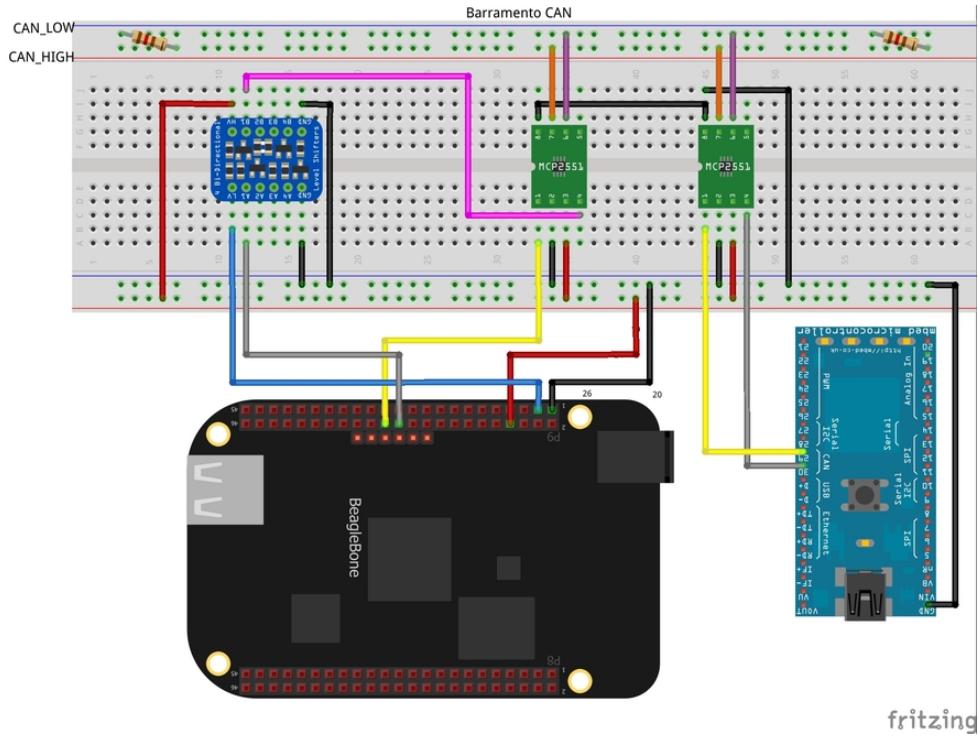
P9_19 (can0_RX)	Saída baixa nº2 do conversor lógico
P9_20 (can0_TX)	Pino 1 do transceiver 2 (Pino TX)
P9_24 (can1_RX)	Saída baixa nº1 do conversor lógico
P9_26 (can1_TX)	Pino 1 do transceiver 1 (Pino TX)
Pino 4 - transceiver 1	Entrada alta nº1 do conversor lógico
Pino 4 - transceiver 2	Entrada alta nº2 do conversor lógico
P9_3 (3.3V)	Referência baixa do conversor lógico
5V	Referência alta do conversor lógico
Pino 7 transceivers 1 e 2	CAN_HIGH (rede CAN)
Pino 6 transceivers 1 e 2	CAN_LOW (rede CAN)
Pino 8 transceivers 1 e 2	GND

Fonte: Criada pelo autor

Os demais pinos são conexões 5V ou GND, baseados nas informações de datasheet dos componentes. As tensões de alimentação são provenientes da BeagleBone, alimentada diretamente de uma porta USB.

Para os testes com o controlador Mbed, utilizamos o circuito a seguir, mostrado na figura 16

Figura 16 – Circuito Completo - Mbed e BeagleBone



Fonte: Do autor

As ligações entre os componentes são semelhantes. Note que agora estamos utilizando apenas um controlador CAN da BBB, no caso o CAN1, juntamente com o controlador CAN da Mbed, representado pelos pinos 29 (RX) e 30 (TX).

Ressalta-se o fato de que os controladores podem ser ligados diretamente ao transceiver, sem a necessidade do conversor de nível lógico. É necessário conectar também o terra (GND) da Mbed com o da BBB, assim todos os componentes possuirão uma tensão de referência comum.

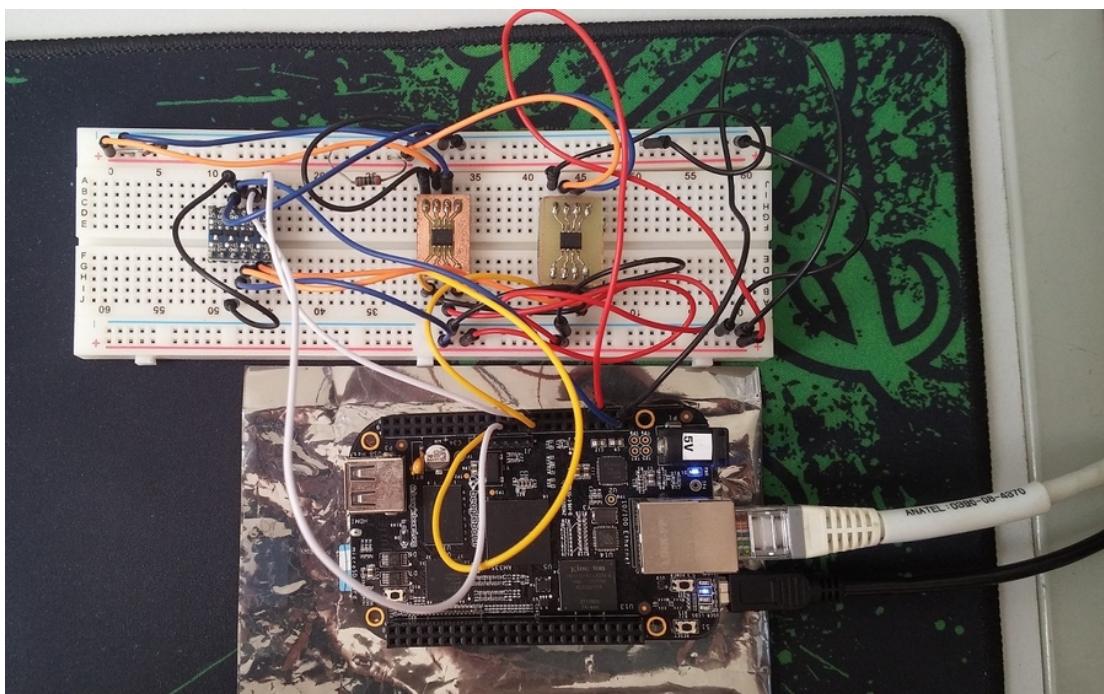
4 Resultados e Discussões

4.1 Circuitos Reais

Com base nas informações técnicas e conhecimentos já descritos nos capítulos anteriores, foram montados circuitos reais. Ainda que na forma experimental, os circuitos serviram para verificar a funcionalidade e eficiência da Beaglebone para a sua função no projeto do relógio de átomos frios, como será evidenciado nas seções a seguir.

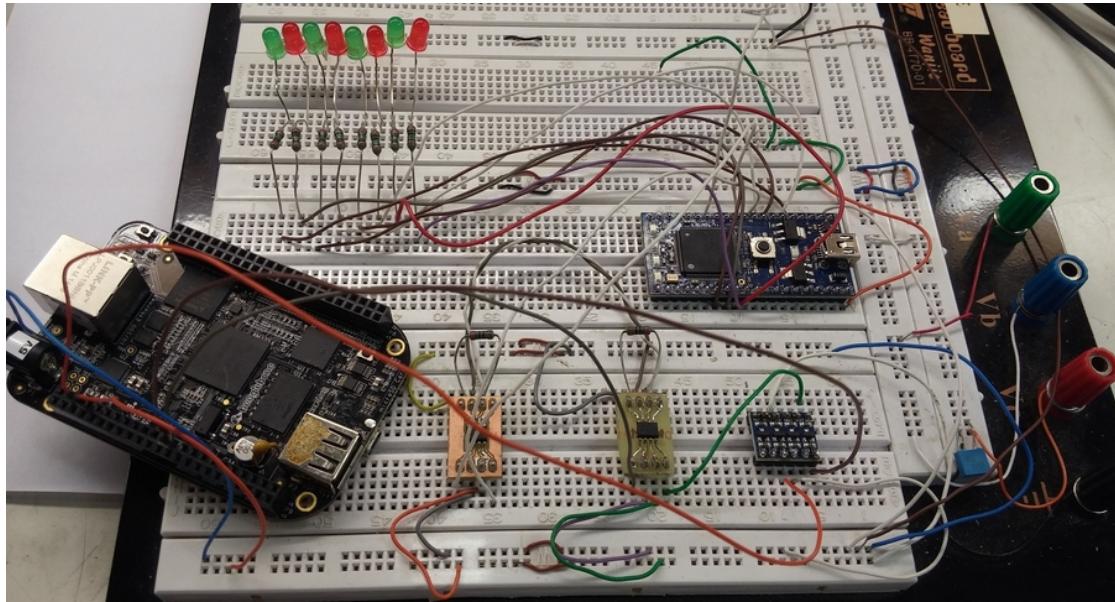
As figuras 17 e 18 a seguir ilustram os dois circuitos criados. O primeiro, com apenas a BeagleBone conectada a rede com seus dois controladores, e o segundo, utilizando a BeagleBone e o controlador Mbed.

Figura 17 – Circuito real - Beaglebone e rede CAN



Fonte: Do autor

Figura 18 – Circuito real - BeagleBone e Mbed



Fonte: Do autor

Apesar do circuito ter sido montado com sucesso, é importante destacar dois problemas que surgiram durante o processo de montagem.

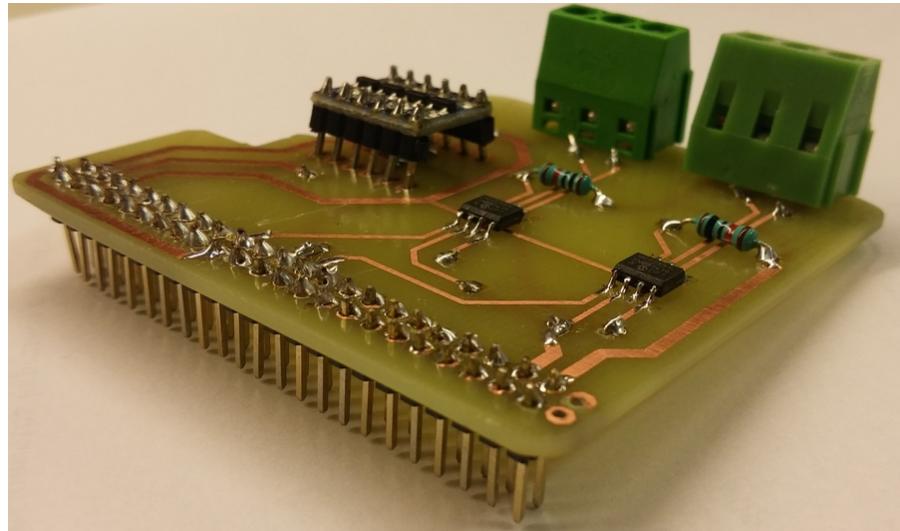
O primeiro, é em relação ao funcionamento e confiabilidade dos *transceivers*. Durante a montagem e testes iniciais foi necessário substituir ao menos uma vez cada componente, pois o mesmo apresentava falha, comprometendo o funcionamento do sistema como um todo.

Acredita-se que a simplicidade e qualidade da placa em que o componente encontra-se soldado, juntamente com o fato de intercambiar as conexões entre a BBB e os *transceivers* a todo momento durante os testes iniciais possam ter gerado o mau funcionamento e comprometimento dos componentes.

O outro problema é em relação a organização dos fios que conectam os componentes; Quando compara-se as figuras 15 e 16 com as figuras do sistema real, o que mais se destaca é a grande quantidade de fios e seus tamanhos. Ainda que totalmente funcional, os fios acabam atrapalhando o usuário visualmente, dificultando a verificação das ligações e até mesmo aferição de sinais para o *debug*.

Por isso, paralelamente está sendo desenvolvido uma ‘cape’ - Como é chamada uma espécie de placa auxiliar que conecta-se ao topo da BBB, com uma função específica - para retirar todos os fios e otimizar o espaço do circuito. Esta *cape* em desenvolvimento será uma placa de circuito impresso, que conterá todos os componentes aqui utilizados - *transceivers* e conversores lógicos - eliminando este problema. O primeiro protótipo desta placa é mostrando na figura 19 a seguir.

Figura 19 – Cape desenvolvida com o circuito CAN



Fonte: Do autor

4.2 Testes com a CAN ‘RAW’

Após montado o circuito da figura 17 foram feitos testes utilizando apenas os terminais e os comandos simples de envio e recebimento de mensagem. A figura 20 ilustra esse teste:

Figura 20 – Testes com a CAN raw

<pre>candump -> beaglebone <- root@beaglebone:/var/lib/cloud9# candump -x -c -tA any (2016-11-19 22:26:39.574069) can0 RX - - 6F8 [8] BC F1 DC 04 FD 67 DC 7A (2016-11-19 22:26:39.574042) can1 TX - - 6F8 [8] BC F1 DC 04 FD 67 DC 7A (2016-11-19 22:26:39.774347) can0 RX - - 313 [8] AD D5 F0 45 25 FE 33 2A (2016-11-19 22:26:39.774317) can1 TX - - 313 [8] AD D5 F0 45 25 FE 33 2A (2016-11-19 22:26:39.974622) can0 RX - - 359 [8] F4 CA 2E 5B 56 80 A6 1D (2016-11-19 22:26:39.974592) can1 TX - - 359 [8] F4 CA 2E 5B 56 80 A6 1D (2016-11-19 22:26:40.174892) can0 RX - - 091 [8] 90 FD 07 22 F8 32 50 52 (2016-11-19 22:26:40.174858) can1 TX - - 091 [8] 90 FD 07 22 F8 32 50 52 (2016-11-19 22:26:40.375195) can0 RX - - 734 [8] B9 D9 72 3D 7D 82 89 6B (2016-11-19 22:26:40.375138) can1 TX - - 734 [8] B9 D9 72 3D 7D 82 89 6B (2016-11-19 22:26:40.575498) can0 RX - - 622 [8] 1C BC 69 55 F6 97 44 04 (2016-11-19 22:26:40.575469) can1 TX - - 622 [8] 1C BC 69 55 F6 97 44 04 (2016-11-19 22:26:40.775750) can0 RX - - 403 [8] 09 20 DC 7E 18 12 8C 02 (2016-11-19 22:26:40.775724) can1 TX - - 403 [8] 09 20 DC 7E 18 12 8C 02 (2016-11-19 22:26:40.976824) can0 RX - - 674 [8] CB 17 87 4A 6D 05 19 32 (2016-11-19 22:26:40.975996) can1 TX - - 674 [8] CB 17 87 4A 6D 05 19 32 (2016-11-19 22:26:41.176289) can0 RX - - 405 [7] 6A 6D F5 2C 18 5F AF (2016-11-19 22:26:41.176257) can1 TX - - 405 [7] 6A 6D F5 2C 18 5F AF (2016-11-19 22:26:41.376563) can0 RX - - 2A5 [7] 3E 5D E3 44 FE E5 92 (2016-11-19 22:26:41.376533) can1 TX - - 2A5 [7] 3E 5D E3 44 FE E5 92 (2016-11-19 22:26:41.576899) can0 RX - - 024 [2] 55 66 (2016-11-19 22:26:41.576863) can1 TX - - 024 [2] 55 66 (2016-11-19 22:26:41.777202) can0 RX - - 2E0 [5] AE 23 1C 0B 21 (2016-11-19 22:26:41.777173) can1 TX - - 2E0 [5] AE 23 1C 0B 21 (2016-11-19 22:26:41.977518) can0 RX - - 2A1 [7] 9E 54 10 27 C3 E8 95 (2016-11-19 22:26:41.977478) can1 TX - - 2A1 [7] 9E 54 10 27 C3 E8 95 (2016-11-19 22:26:42.177779) can0 RX - - 171 [8] B9 80 DA 5A 74 BD 7C 61 (2016-11-19 22:26:42.177751) can1 TX - - 171 [8] B9 80 DA 5A 74 BD 7C 61</pre>	<pre>cangen -> beaglebone <- root@beaglebone:/var/lib/cloud9# cangen can1</pre>
--	---

Fonte: Do autor

Nela, podemos observar no terminal direito o comando *cangen*, que gera e envia mensagens aleatórias , enquanto no terminal direito o comando *candump* mostra na tela todas as mensagens recebidas na rede, informando ainda qual controlador enviou ou recebeu as mensagens. Os parâmetros passados para os comandos são apenas

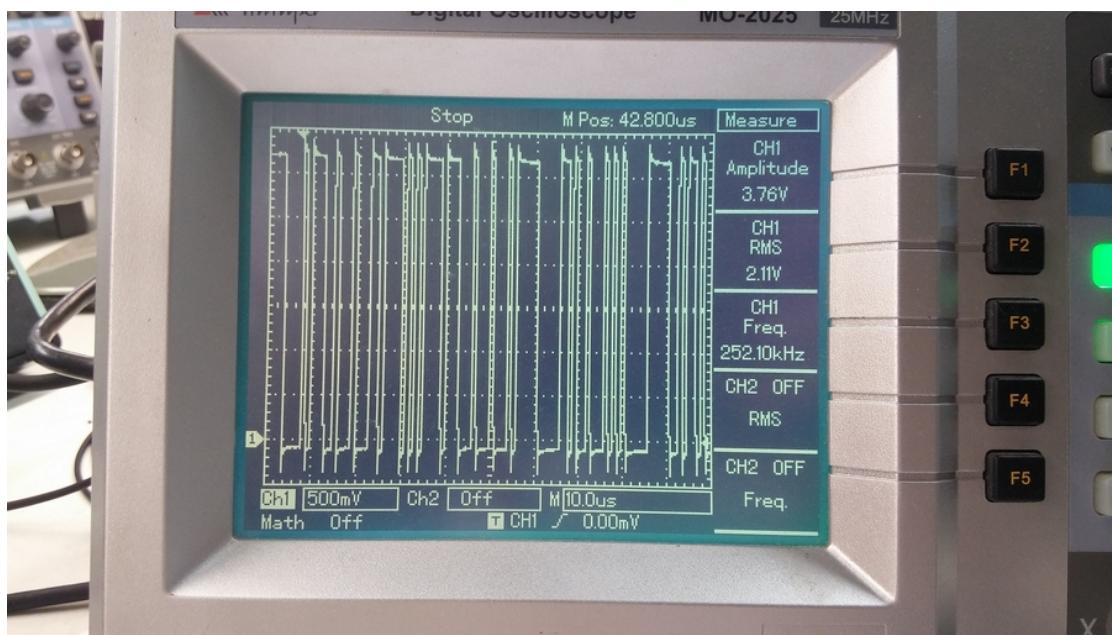
para facilitar o *debug*

Esse resultado valida a instalação dos módulos CAN e o funcionamento dos controladores da BeagleBone, mostrando que os mesmos estão aptos a interagirem com a rede. Dos diversos testes realizados damos destaque a um deles, onde utilizamos um intervalo de envio de 50ms entre cada mensagem gerada, durante um longo período de tempo. Não houve perda de desempenho ou atraso nos envios das mensagens.

Acredita-se que este desempenho é mais que suficiente para as necessidades da aplicação deste projeto.

Além disso, tem-se também a figura 21, resultado da medição do osciloscópio no pino 4 (RX) de um dos *transceivers* durante a execução do comando *cangen*. A informação importante da medida, mais que o próprio valor da tensão, são as oscilações. Elas indicam que as mensagens estão sendo transmitidas na rede.

Figura 21 – Medida do osciloscópio durante a transmissão de mensagens



Fonte: Do autor

Ainda que os testes acima foram realizados com êxito, é importante ressaltar que todo o processo para habilitar os controladores não foi um processo simples, como utilizar outras funções da placa, por exemplo as portas GPIO. De fato, a placa conta com inúmeros recursos, incluindo os controladores CAN, porém usuários não tão avançados podem ter dificuldade no uso de tais recursos.

O sucesso do processo só foi possível com o auxílio de informações compartilhadas pela comunidade aberta da plataforma; Diversos usuários em diferentes fóruns de compartilhamento contribuíram com informações relevantes, relacionadas não só ao processo de habilitação mas também com conteúdo informativo sobre drivers,

particularidades da plataforma e do sistema operacional.

4.3 Testes com Python

De todos os programas gerados durante os testes e protótipos do projeto, destacam-se a importância de 2 pares de programas - os programas iniciais e as versões finais. Os códigos-fonte dos quatro programas podem ser encontrados ao fim deste trabalho no apêndice A. Todos encontram-se comentados de forma detalhada.

Agrupa-se os programas em pares pois cada par é composto de um programa que envia dados (TX) e outros que os receberá (RX). Assim o primeiro par contém os programas *tx_simple.py* e *rx_simple.py*.

O primeiro programa envia periodicamente (a cada 1 segundo) uma mensagem com conteúdo e identificador fixos. Não há nenhum algoritmo complexo no código, o programa se limita apenas a esta tarefa. Inspirou-se no comando *cansend*, que no terminal envia uma mensagem para a rede.

O segundo programa foi inspirado no comando *candump*. Ele tem por função monitorar todas as mensagens recebidas pela rede. Quando uma nova mensagem é recebida, ele verifica se o *id* da mensagem possui um valor esperado e mostra o conteúdo da mensagem. A figura 22 a seguir ilustra o funcionamento simultâneo dos programas.

Figura 22 – Testes iniciais com Python

```

bash - "beaglebone" ✘ Bruno/tcc/tutorial_er ✘ + Stop C Run Ce Command: Bruno/tcc/tutorial_embarca... ... CWD
Recebemos uma mensagem
A mensagem possui um id conhecido!
[0xf5, 0x23, 0x1a]
Recebemos uma mensagem
A mensagem possui um id conhecido!
[0xf5, 0x23, 0x1a]
Recebemos uma mensagem
A mensagem possui um id conhecido!
[0xf5, 0x23, 0x1a]
Recebemos uma mensagem
A mensagem possui um id conhecido!
[0xf5, 0x23, 0x1a]
Recebemos uma mensagem
A mensagem possui um id conhecido!
[0xf5, 0x23, 0x1a]
Recebemos uma mensagem
A mensagem possui um id conhecido!
[0xf5, 0x23, 0x1a]
Recebemos uma mensagem
A mensagem possui um id conhecido!
[0xf5, 0x23, 0x1a]

bash - "beaglebone" ✘ Bruno/tcc/tutorial_er ✘ + Stop C Run Ce Command: Bruno/tcc/tut...
Mensagem enviada
Mensagem enviada
Mensagem enviada
Mensagem enviada

```

Fonte: Do autor

Os programas iniciais permitiram adquirir os conhecimentos básicos da fusão do Python com os drivers CAN. O manuseio dos dados, sejam dos identificadores ou do conteúdo das mensagens e formas de envio e recebimento, ainda que simples, são fundamentais para avançar na complexidade da programação nos permitindo concretizar os objetivos.

A partir de então desenvolveram-se novos programas, com funções adicionais, envolvendo leitura de arquivos, processamento de dados pré e pós envio na rede.

Muitas otimizações de código foram necessárias, bem como ajustes conforme a necessidade da aplicação, resultando assim nas versões finais - os programas *rxdados.py* e *txdadosv2.py*.

O programa *txdadosv2* é o programa responsável pelo envio dos dados na rede. Mais que isso, ele lê o conteúdo de um arquivo de texto e o envia para o barramento. A figura 23 mostra o arquivo de texto utilizado para os testes finais.

Figura 23 – Arquivo de texto utilizado para validação

	0x060	255	128	0	0b1110111	0x4C	0b11101101	86	0x7A
1	0x060	255	128	0	0b1110111	0x4C	0b11101101	86	0x7A
2	32354	5781	4	10005	25123	159875	9867	369	

Fonte: Do autor

O arquivo de texto possui dados no formato definido previamente pelo projeto da aplicação - enviar os parâmetros de configuração para um módulo de saídas digitais. Nele, temos como primeiro elemento da linha inicial o valor que será o id das mensagens, enquanto o restante do conteúdo se refere aos dados que deverão ser enviados. A primeira linha contém os valores de saída das portas lógicas e a segunda refere-se aos tempos em elas manterão esse valor.

Apesar de extenso, o programa de envio segue uma lógica sequencial; Ele lê os dados do arquivo, os processa e os envia na rede. Por exigências de projeto, é necessário que os dados sejam enviados em pares (valor, tempo) com a id da mensagem pré definida no mesmo arquivo.

Assim, temos como exemplo uma mensagem de identificador 60_{16} e conteúdo (255, 32354).

Um dos problemas encontrados para enviar o conteúdo foi o valor a ser enviado, que ultrapassava os limites de protocolo das mensagens. Como já citado anteriormente, as mensagens são compostas por oito grupos de dois bytes cada. Assim, cada posição fica limitada a valores de 0 a 255 (ou 0 a FF, em hexadecimal).

Normalmente, enviaria-se o par (valor,tempo) na mesma mensagem, cada um ocupando um frame na mensagem. Como os valores de tempo são escritos em milissegundos, é razoável esperar que eles facilmente ultrapassarão o valor de uma frame. Por isso, a solução encontrada foi separar os algorismos do tempo, armazenando cada dígito em uma posição. Portanto, em uma mensagem que contém 8 frames, o primeiro contendo o valor da porta lógica e os demais os algorismos do tempo.

Tal estratégia permitiu que valores de tempo, antes limitados à um máximo

de 255ms pudessem ser enviados com até 10^7 ms (cem mil segundos). Para o par exemplificado acima, temos então o seguinte formato de mensagem:

Id de mensagem: 0x60

Conteúdo: [255] [3] [2] [3] [5] [4]

Então, após o par ser processado e a mensagem formatada no modelo descrito, ela é enviada na rede. O processo continua até que todo o conteúdo do arquivo seja enviado. Como fruto dos diversos programas criados anteriormente, a versão final é capaz de trabalhar com valores decimais, hexadecimais e binários, desde que escritos da maneira correta a ser interpretada pela linguagem e pelo algoritmo de programa. Na imagem 23 vemos os números decimais representados pela maneira normal, hexadecimais representados por '0x' antes do valor e binários representados pelo prefixo '0b'. Tal ajuste permite uma maior liberdade do usuário ao gerar os arquivos de texto.

O programa *rxdados.py* recebe as mensagens na rede e as processa. O programa foi criado para receber as mensagens no formato criado pelo programa anterior. Ele tem como output os pares (valor,tempo) recebidos.

Em suma, o programa receberá e processará cada par de uma vez. o valor das saídas lógicas não precisa de nenhum tipo de tratamento. Já o tempo, por estar fragmentado, é necessário agrupar os algarismos da forma original. Para isso foi feita uma função responsável por esta junção. A partir dai, o par de informações é separado em duas variáveis no programa, prontas para serem utilizadas. A figura 24 a seguir exibe a execução simultânea dos programas finais.

Figura 24 – Execução dos programas em sua versão final

The screenshot shows two terminal windows side-by-side. The left window is titled 'python - beaglebone' and shows the command 'root@beaglebone:/var/lib/cloud9/Bruno/tcc/programas# python rxdados.py' followed by the message 'CAN SELECIONADA'. Below this, a list of tuples representing sensor readings is displayed: (255, 32354), (128, 5781), (0, 4), (119, 10005), (76, 25123), (237, 159875), (86, 9867), and (122, 369). The right window is titled 'bash - beaglebone' and shows the command 'root@beaglebone:/var/lib/cloud9/Bruno/tcc/programas# python txdados v2.py' followed by the message 'CAN SELECIONADA' and '---The End---'. Both windows are running on a Beaglebone system with root privileges.

Fonte: Do autor

Note que aparentemente os valores estão diferentes do arquivo txt apresentados. Porém vale lembrar que alguns valores no arquivo estão em uma base numérica diferente, enquanto na figura 24 todos são representados na base decimal.

O programa responsável pelo envio deverá ser usado como template para os programas de cada módulo do relógio atômico. Por ser um programa generalista, que apenas faz o processamento dos dados, ele precisa apenas ser personalizado

conforme a função de cada módulo. Ou seja, deverão ser acrescentadas funções e ações a serem tomadas baseadas no conteúdo recebido.

Para uma verificação e validação adicional, temos também a figura 25, que mostra as mensagens que estão sendo enviadas e recebidas na rede, no momento da execução do programa *txdadosv2.py*

Figura 25 – Candump durante o programa txdadosv2.py

The screenshot shows two terminal windows. The left window, titled 'candump - *beagleb' (partially visible), displays CAN bus traffic with the command 'candump -x -c -tA any'. The right window, titled 'bash - *beaglebone*' (partially visible), shows the execution of 'python txdadosv2.py' and its output, which includes 'CAN SELECIONADA' and '--The End--'.

```

root@beaglebone:/var/lib/cloud9/Bruno/tcc/programas# candump -x -c -tA any
(2016-11-20 00:09:33.116100) can0 RX - - 060 [6] FF 03 02 03 05 04
(2016-11-20 00:09:33.116081) can1 TX - - 060 [6] FF 03 02 03 05 04
(2016-11-20 00:09:34.123843) can0 RX - - 060 [5] 80 05 07 08 01
(2016-11-20 00:09:34.123809) can1 TX - - 060 [5] 80 05 07 08 01
(2016-11-20 00:09:35.127373) can0 RX - - 060 [2] 00 04
(2016-11-20 00:09:35.127339) can1 TX - - 060 [2] 00 04
(2016-11-20 00:09:36.131238) can0 RX - - 060 [6] 77 01 00 00 00 05
(2016-11-20 00:09:36.131204) can1 TX - - 060 [6] 77 01 00 00 00 05
(2016-11-20 00:09:37.133941) can0 RX - - 060 [6] 4C 02 05 01 02 03
(2016-11-20 00:09:37.133903) can1 TX - - 060 [6] 4C 02 05 01 02 03
(2016-11-20 00:09:38.136842) can0 RX - - 060 [7] ED 01 05 09 08 07 05
(2016-11-20 00:09:38.136813) can1 TX - - 060 [7] ED 01 05 09 08 07 05
(2016-11-20 00:09:39.139174) can0 RX - - 060 [5] 56 09 08 06 07
(2016-11-20 00:09:39.139142) can1 TX - - 060 [5] 56 09 08 06 07
(2016-11-20 00:09:40.142984) can0 RX - - 060 [4] 7A 03 06 09
(2016-11-20 00:09:40.142952) can1 TX - - 060 [4] 7A 03 06 09

scalls)
root@beaglebone:/var/lib/cloud9/Bruno/tcc/programas# python txdadosv2.py
CAN SELECIONADA
--The End--
root@beaglebone:/var/lib/cloud9/Bruno/tcc/programas#

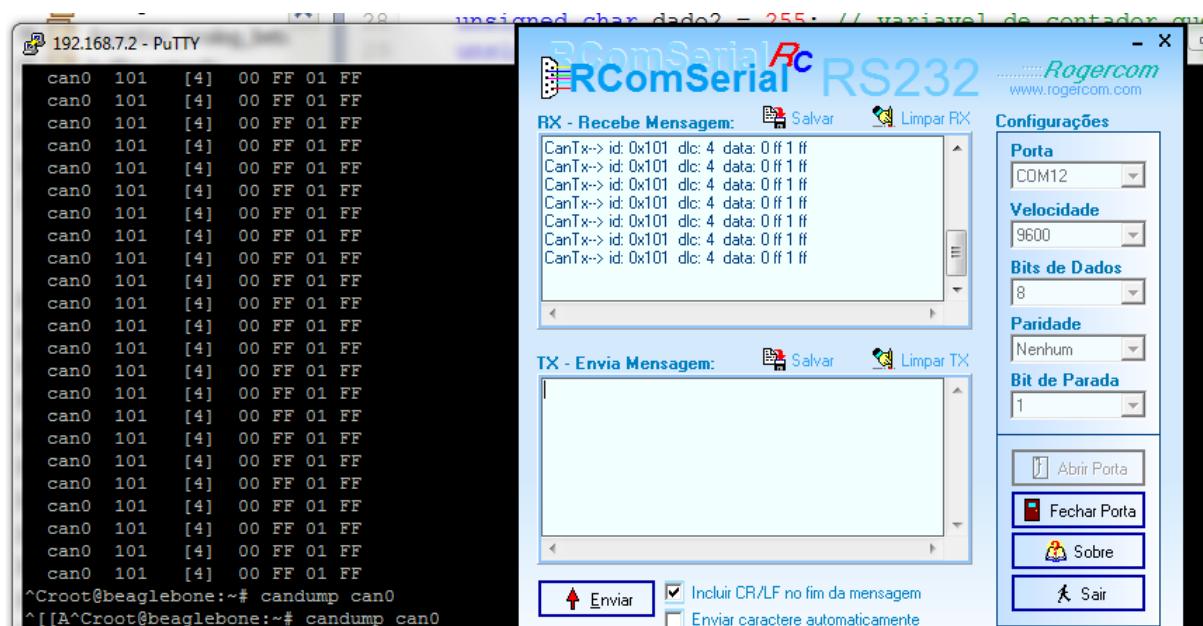
```

Fonte: Do autor

4.4 Testes com Mbed

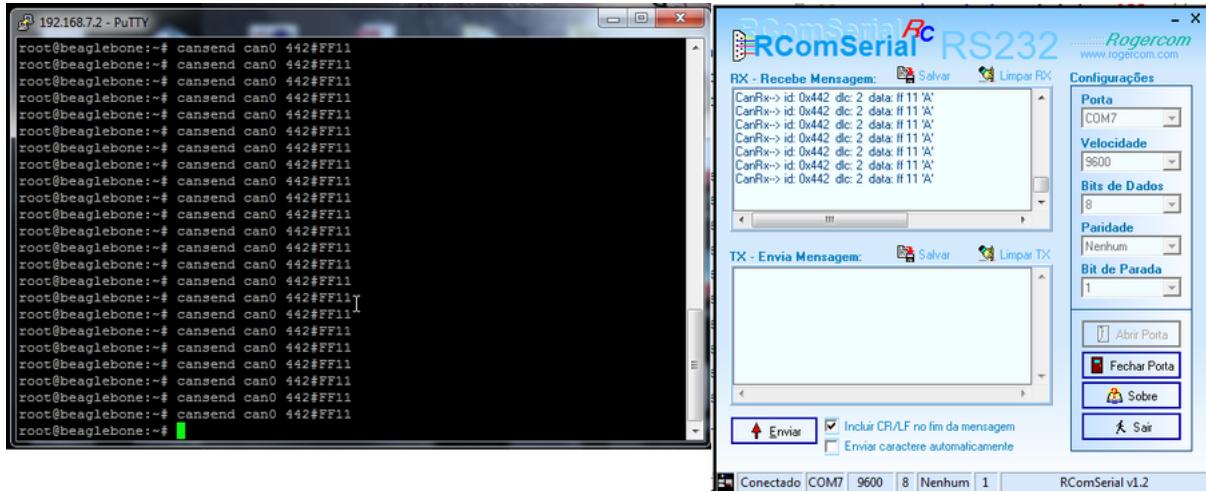
Na Mbed, os resultados também foram promissores. A primeira comunicação, realizada de forma básica e através dos terminais e comandos simples mostrou-se rápida e eficiente. Foram testados os envios e recebimentos de ambas as partes, como mostrados nas figuras 26 e 27 a seguir:

Figura 26 – Envio por Mbed e recebimento por BBB



Fonte: Do autor

Figura 27 – Envio por BBB e recebimento por Mbed



Fonte: Do autor

Após verificado que a comunicação estava funcional, os próximos testes foram realizados para simular o propósito da aplicação, com a Beaglebone enviando as mensagens (através do programa em Python *txdadosv2.py*) e a Mbed recebendo e processando os dados, através da adaptação do programa *rxdados.py*, para a linguagem de programação C.

A BeagleBone envia mensagens pela rede, que são recebidas pela Mbed e separadas em dois vetores, um contendo o valor das saídas lógicas e outro contendo os valores de tempo. Os resultados são mostrados na figura 28.

Figura 28 – Execução dos programas completos

```

root@beaglebone:/var/lib/cloud9/liepo/programas# python txdadosv2.py
CAN SELECCIONADA
--The End--
root@beaglebone:/var/lib/cloud9/liepo/programas# python txdadosv2.py
CAN SELECCIONADA
--The End--
root@beaglebone:/var/lib/cloud9/liepo/programas# [REDACTED]

COM4 - PuTTY
CanRx--> id: 0x60  dlc: 4  data: 7a 3 6 9
for:10
for:100
Dado:3  acm:300 soma:300 pot:2for:10
Dado:6  acm:60 soma:360 pot:1Dado:9 acm:9 soma:369 pot:0soma final:369tempo:36
9ESCREVENDO ARQUIVO
vetor std::value_arq[0]:255
vetor std::value_arq[1]:128
vetor std::value_arq[2]:0
vetor std::value_arq[3]:19
vetor std::value_arq[4]:76
vetor std::value_arq[5]:237
vetor std::value_arq[6]:86
vetor std::value_arq[7]:122
vetor std::time_arq[0]:500
vetor std::time_arq[1]:32354
vetor std::time_arq[2]:5781
vetor std::time_arq[3]:4
vetor std::time_arq[4]:10005
vetor std::time_arq[5]:25123
vetor std::time_arq[6]:159875
vetor std::time_arq[7]:9867
ARQUIVO COMPLETO

```

Fonte: Do autor

Com o sucesso evidenciado pela comunicação entre as duas plataformas, através da rede CAN utilizando os programas desenvolvidos, tem-se a validação final deste projeto.

4.5 Criação de artigo

Com todos os conhecimentos adquiridos durante a execução do projeto, decidiu-se por escrever um artigo, na forma de tutorial, visando compartilhar as informações necessárias para habilitar os controladores CAN na Beaglebone e também demonstrações de utilização com Python.

A motivação principal foi a dificuldade na obtenção de informações relacionadas ao assunto. Novamente, reitare-se que o conteúdo abordando sobre o tema é escasso e muitas vezes impreciso, confuso e desatualizado. O objetivo do artigo criado é ajudar os usuários, principalmente iniciantes, a utilizar o recurso da plataforma.

Esta foi a forma encontrada pelo autor de contribuir com a comunidade *open source*, que ajuda imensamente com inúmeros projetos e pessoas, visando a difusão do conhecimento. Mesmo que não previsto nos objetivos iniciais, é certo que o tutorial é de grande importância para manter vivo o hábito de colaboração destas comunidades.

O artigo será publicado de forma *online*, em um dos websites referência no Brasil sobre eletrônica e sistemas embarcados(EMBARCADOS, 2007) e também no *blog* do LIEPO, website onde os membros do grupo realizam publicações sobre os trabalhos desenvolvidos no laboratório(BLACKBOXLAB, 2016). A publicação do artigo está prevista para a metade de dezembro de 2016.

5 Conclusão

5.1 Considerações Finais

Após toda a elaboração do projeto e execução dos testes de validação, conclui-se que os objetivos foram alcançados. Mais que isso, obtivemos resultados adicionais do que se era esperado.

É fato que, de certa forma, trabalhávamos com restrições (ou pré definições) de requisitos no projeto. Com o hardware já definido, a rede escolhida era necessário que tudo se integrasse. O desafio porém não diminuiu. Foi necessário um estudo avançado no assunto para que fosse possível utilizar o hardware corretamente, elaborando o software compatível com as necessidades de projeto.

O projeto desenvolvido aqui está condizente com o esperado pela aplicação. A plataforma Beaglebone encontra-se configurada de forma padronizada, com todos os arquivos necessários bem como sistema operacional configurado. OS recursos necessários também encontram-se habilitados, disponibilizando-a para ser embarcada no projeto de relógio atômico.

Os programas desenvolvidos resolveram os principais gargalos de projetos, além de serem otimizados para a aplicação. As mensagens e dados estão sendo transferidos de forma eficaz. Mesmo que alternativas simples foram usadas para abordar o problema, verificou-se que elas foram capazes de atingir os objetivos com exatidão.

Em relação as dificuldades encontradas, pode-se dizer que a ausência de informações a respeito dos recursos da Beaglebone Black fazem um pouco de falta, principalmente quando o usuário não posse conhecimentos muito avançados dos sistemas embarcados e operacionais.

De fato, no website oficial da plataforma, bem como na documentação do fabricante, os recursos encontram-se listados. Porém falta informação prática sobre sua utilização. Há sim muito conteúdo sobre sobre os recursos mais simples, mas tratando-se dos mais avançados percebe-se que poderia ser melhor.

Com a criação e publicação do tutorial acredita-se adicionar um pouco mais de informação acerca do assunto. Ainda que o artigo seja um pouco nos detalhes técnicos mais avançados, ele definitivamente permite que qualquer usuário possa usufruir dos recursos em questão sem perder-se no grande volume de informações atualmente disponíveis. Espera-se agora que o tempo seja melhor investido no desenvolvimento de projetos do que na habilitação dos recursos.

Destaca-se novamente a contribuição da comunidade open source. O software e hardware livre permite que os usuários se ajudem na resolução de problemas bem como desenvolvam projetos mais avançados, contribuindo com a difusão de conhecimento.

5.2 Trabalhos Futuros

Sem dúvida, este trabalho foi concluído com êxito. Entretanto, o desenvolvimento deste projeto é apenas dos trabalhos que deverão ser executados para o pleno funcionamento do padrão atômico de frequência móvel.

A nossa abordagem, para o desenvolvimento do software foi sempre focada na comunicação com o módulo de saídas digitais. Porém, ainda há necessidade da comunicação com mais dois módulos importantes: O de entrada analógica e saída analógica.

Ainda que os softwares tenham sido elaborados de maneira genérica, para fácil adaptação, sabe-se que os tipos de dados com que estes módulos trabalharão são ligeiramente diferentes, o que gera a necessidade da criação de programas adicionais para cada módulo.

É fato que de um modo genérico, ainda serão programas que receberão ou enviarão mensagens na rede. Basta apenas que o programador tenha cuidado com as particularidades e necessidades de cada módulo, sem esquecer das especificações e limitações da rede.

Em relação ao hardware, não deverão ser feitas grandes mudanças, já que a Beaglebone continuará se comunicando fisicamente com a rede do mesmo modo anterior. A própria rede permite que novos módulos se conectem sem interferir nas ligações de circuitos anteriores.

Referências

- AHMED, M. et al. The Brazilian time and frequency atomic standards program. *Anais da Academia Brasileira de Ciências*, scielo, v. 80, p. 217 – 252, 06 2008. ISSN 0001-3765. Disponível em: <<http://www.scielo.br/scieloOrg/php/articleXML.php?lang=pt&pid=S0001-37652008000200002>>. Citado na página 21.
- BARROS, C. *Nova imagem na Beaglebone Black*. 2015. Disponível em: <<http://blackboxlab.wixsite.com/home/single-post/2015/07/31/Nova-imagem-na-Beaglebone-Black>>. Acesso em: 08/08/2016. Citado na página 36.
- Beaglebone.org. *BeagleBone Black*. 2016. Disponível em: <<https://beagleboard.org/>>. Acesso em: 15/08/2016. Citado 3 vezes nas páginas 25, 33 e 35.
- BLACKBOXLAB. *BlackBox Lab*. 2016. Disponível em: <<http://blackboxlab.wixsite.com/home>>. Acesso em: 15/11/2016. Citado na página 54.
- BOSCH, R. *CAN Specification - Version 2.0*. [S.I.], 1991. Citado na página 28.
- CLOUD9. *Cloud9*. 2016. Disponível em: <<https://c9.io/>>. Acesso em: 05/11/2016. Citado na página 36.
- DAWSON, M. *Python Programming for the Absolute Beginner*,. 3. ed. [S.I.]: Course Technology, 2010. Citado na página 37.
- ELINUX. *Can-utils*. 2015. Disponível em: <<http://elinux.org/Can-utils>>. Acesso em: 08/08/2016. Citado na página 41.
- EMBARCADOS. *Portal Embarcados*. 2007. Disponível em: <<https://www.embarcados.com.br/>>. Acesso em: 15/11/2016. Citado na página 54.
- Embedded Things. *Enable CANBus on the BeagleBone Black*. 2013. Disponível em: <<http://www.embedded-things.com/bbb/enable-canbus-on-the-beaglebone-black>>. Acesso em: 10/10/2016. Citado na página 41.
- FAIRCHILD. *BSS138*. [S.I.], 2005. Citado na página 38.
- FRITZING. *Fritzing*. 2016. Disponível em: <<http://fritzing.org/download/>>. Acesso em: 10/10/2016. Citado na página 42.
- GRIMMETT, R. *BeagleBone Robotic Projects*. [S.I.]: Packt Publishing, 2013. Citado na página 40.
- GUIMARÃES, A. D. A. UM ROTEIRO DE IMPLEMENTAÇÃO DE UMA REDE CAN (CONTROLLER AREA NETWORK). 2003. Citado na página 26.
- MAGALHÃES, D. V. *Desenvolvimento de uma fountain atômica para utilização como padrão primário de tempo*. 2004. Tese (Doutorado) — Instituto de Física de São Carlos da Universidade de São Paulo. Citado na página 21.

MBED. *Mbed*. 2016. Disponível em: <<https://www.mbed.com>>. Acesso em: 01/11/2016. Citado 2 vezes nas páginas 25 e 39.

MICROCHIP. *MCP2551 - High-Speed CAN Transceiver*. [S.I.], 2010. Citado na página 37.

MULLER, S. T. *Padrão de Frequência Compacto*. 2010. Tese (Doutorado) — Instituto de Física de São Carlos da Universidade de São Paulo. Citado na página 24.

NATALE, M. D. Understanding and using the Controller Area Network. Outubro 2008. Citado 2 vezes nas páginas 31 e 32.

NXP. *LCP1768 Product Datasheet*. [S.I.], 2015. Citado na página 39.

PANG, S. *SK Pang Electronics*. 2016. Disponível em: <<http://skpang.co.uk/blog/>>. Acesso em: 10/10/2016. Citado na página 37.

PECHONERI, R. D. *Sistemas de controle de monitoramento para padrão atômico de frequência de Césio*. 2013. 137 p. Dissertação (Programa de Pós Graduação em Engenharia Mecânica e Área de Concentração em Dinâmica de Máquinas e Sistemas) — Universidade de São Paulo. Citado 3 vezes nas páginas 21, 23 e 24.

POWELL, B. *Python-can Documentation*. 2016. Disponível em: <<https://python-can.readthedocs.io/en/latest/index.html>>. Acesso em: 15/08/2016. Citado na página 37.

PUTTY. *Putty*. 2016. Disponível em: <www.putty.org>. Acesso em: 08/08/2016. Citado na página 36.

PYTHON. *Python*. 2016. Disponível em: <<https://www.python.org>>. Acesso em: 01/11/2016. Citado na página 36.

SANTOS, R.; PERESTRELO, L. *Beaglebone for Dummies*. [S.I.]: John Wiley & Sons, Inc, 2015. Citado na página 40.

SOUSA, J. M. R. P. de. *Aplicação do protocolo CAN a uma rede de comunicações*. 2000. Dissertação (Mestrado) — Faculdade de Engenharia da Universidade do Porto. Citado 2 vezes nas páginas 21 e 25.

Texas Instruments. *BeagleBone Black Development Board*. Disponível em: <<http://www.ti.com/tool/beaglebk>>. Acesso em: 09/08/2016. Citado na página 33.

Texas Instruments. *Introduction to the Controller Area Network (CAN)*. [S.I.], 2002. Citado 2 vezes nas páginas 26 e 28.

Texas Instruments. *AM335X DCAN Driver Guide*. 2012. Disponível em: <http://processors.wiki.ti.com/index.php/AM335X_DCAN_Driver_Guide>. Acesso em: 15/08/2016. Citado na página 41.

Apêndices

Apêndice A

Este apêndice contém o código dos 4 programas principais utilizados neste trabalho.

Código 1 – Programa rx-simples.py

```
#!/usr/bin/python
# coding=UTF-8

### Esse programa monitorar a rede CAN, mostrando todas as mensagens recebidas.
### Tambem verificará se a mensagem possuem um id conhecido ou não

#Importando bibliotecas

import can #Biblioteca do python-can

#Definir qual controlador vamos usar

bus = can.interface.Bus(channel='can0', bustype='socketcan_ctypes') #
    Selecionamos o can0 como o controlador que monitorara 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 mesmo que o id definido
        print("A mensagem possui um id conhecido!")
    else:
        print("A mensagem possui um id desconhecido!")

#O loop a seguir ira 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 percorrer 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
```

Código 2 – Programa tx-simples.py

```

#!/usr/bin/python
# coding=UTF-8

### Esse programa enviara periodicamente uma mensagem para a rede can

#Importando bibliotecas

import can
import time

#Definir qual controlador vamos usar

bus = can.interface.Bus(channel='can1', bustype='socketcan_ctypes') # Selecionamos o can0 como o controlador que monitorara as mensagens

#Definindo um id para nossas mensagens

id = 0x123

#Definindo o conteudo das mensagens
cont = [0xf5,0x23,0x1A]

#Main

while True:

    msg = can.Message(arbitration_id = id,data=cont) #Criando a mensagem que ser enviada na rede
    bus.send(msg) #Enviado para a rede
    print "Mensagem enviada"
    time.sleep(5) #espera 5 segundos antes de enviar a mesma mensagem

```

Código 3 – Programa rx-dados.py

```

#!/usr/bin/python
# coding=UTF-8

#Esse programa simular a mbed que devera receber as mensagens enviadas pelo programa txdados.py, com base nos dados do arquivo txts/dados.txt
#O objetivo aqui e receber a mensagem no padro que ela ser enviada, e obter os dados da forma que eles est o escritos no arquivo
#As mensagens recebidas ter o o padro (valor, [tempo]), sendo que o tempo ser enviado com seus digitos separados,
#permitindo que seja possivel enviar valores maiores que 255(Ex, para um valor de 2000, enviaremos na can 2 0 0 0)
#Ser escrito um c digo para converter o tempo que est separado em digitos para um n mero so

import can
import time
import math

#colocar aqui a id pra validar a mensagem (ou a lista delas)
validation_id = 0x060

```

```
#Sele o do controlador CAN
try:
    bus = can.interface.Bus(channel='can0', bustype='socketcan_ctypes')
    #usa-se socketcan_ctypes pois estamos usando python 2.x
    print("CAN SELECCIONADA \n")
except OSError:
    print("Dispositivo CAN nao encontrado")
    exit()

#Main loop – Processara as mensagens por ordem. S processa a
#proxima assim que a anterior for concluida

def gettime(vetor): #Recebe o vetor com os digitos de tempo, e
#transforma em um unico numero (Ex: [3 0 0 0] => 3000
#   importante destacar que o vetor recebido cont m [valor
#   digitos do tempo] (Ex: [255 3 0 0 0])
#Por isso, s iremos trabalhar com os elementos[1] a [n],
#desconsiderando o elemento [0]
tempo=0
count = len(vetor)-2
for i in range(1,len(vetor)): ##O algoritmo vai multiplicar cada
#digito por 10^potencia referente e somar esse valor
#Ex: [3 0 0 0] => 3*10^3 + 0*10^2
#     + 0*10^1 + 0*10^0 = 3000
    tempo = tempo + vetor[i] * 10**count
    count = count-1
return tempo #Retorna o valor do tempo convertido

### MAIN ####

try: #Esse try/except/finally e opcional. apenas uma forma mais
#elegante de finalizar o programa
    while True:
        message = bus.recv() #aguarda a mensagem ser recebida
        #Verifica se a mensagem de interesse do programa/modulo [
#pela validation id]
        if message.arbitration_id == validation_id:
            valor = message.data[0]
            tempo = gettime(message.data)
            #As variaveis valor e tempo s o respectivamente os pares
            #dos elementos da primera e segunda linha do arquivo de
            #texto
            #Com essas variaveis pode-se continuar o programa a forma
            #que se desejar
            print(valor, tempo)

except KeyboardInterrupt:
    #se for dado um interrupt no programa
    print("Interrupt pressionado — o programa finalizou")
finally:
    print("~~The End~~")
```

Código 4 – Programa tx-dadosv2.py

```

#!/usr/bin/python
# coding=UTF-8

#Esse programa ira ler os dados de um arquivo txt e envia-los pela
#can
#O Desafio aqui ser processar e enviar valores de tempo maiores que
#255
#O arquivo esperado aqui ter o formato:
# 1 linha : message_id [valores]
# 2 linha : [valores de tempo]

#Arquivo refencia: txts/dadosv2.txt

#Modifica es da versao 2:
#O programa agora tamb m e processa numeros binarios escritos no
#txt (Foi criada uma versao nova do txt

import can
import time

#Arquivo que ser lido pelo programa:

nfile = 'txts/dadosv2.txt'

#variaveis

a = []
t = []
msg_data = []

#Selecionando o controlador e tamb m seu tipo (usaremos a can1 para
#enviar)
try:
    bus = can.interface.Bus(channel='can1', bustype='socketcan_ctypes'
                           ) #usa-se socketcan_ctypes pois estamos usando python 2.x
    print("CAN SELECCIONADA \n")
except:
    print("Dispositivo CAN nao encontrado")
    exit()

###ATENCAO — O arquivo txt dever ser formatado no padrao descrito
#no inicio do programa

#Main

try:

    with open(nfile , 'r') as f: #Abre o arquivo. O 'with' garante que
        #o arquivo sera fechado automaticamente

        #Lendo a primeira linha
        data = f.readline()
        vetor = data.split() #o m todo split separa os elementos da
                            #linha(que est o separados por espa o), gerando um vetor
                            #com os elementos

```

```
#separados por virgula
#print("data.split()",vetor) #print do vetor com os
    elementos separados — Descomentar para debug
del a[:] #limpa o vetor a

#O problema que encontrei aqui é que os valores do arquivo
    não são numeros e sim texto. Por isso precisamos
#converte-los em números primeiros

for i in range(len(vetor)): #loop que irá rodar elemento por
    elemento da variável vetor
    try: #Esse try/except irá transformar o texto em
        número, verificando se ele está escrito em hex,
        decimal ou binário
        #Eapsa a conversão adiciona como último elemento
        #do vetor a
        if vetor[i][0:2] == '0b': #Verifica se o valor
            escrito começa com 0b, e portanto conclui-se que
            é binário
            a.append(int(vetor[i],2))
        elif vetor[i][0:2] == '0x': #Se for hexadecimal
            a.append(int(vetor[i],16))
        else: #Se não for nenhum dos casos acima, será
            decimal
            a.append(int(vetor[i])) #armazena na última
            #posição da lista a, sendo que o valor
            #escrito é um decimal
    except:
        print("Os valores do arquivo txt estão com
            formatação incorreta. O programa finalizar")
        exit()

#Aqui temos a primeira linha convertida para
números.

#print("vetor a", a) #print do vetor final da primeira linha
    — Descomentar para debugar

####proxima linha do arquivo:
del t[:]

tempo = f.readline()
tempo = tempo.split()
t = tempo

#print("tempo:", tempo) #Descomentar para debugar — Vetor
    retirado da segunda linha, já separado
#print("t:", t) ###Descomentar para debugar

#Aqui temos as duas linhas do texto lidas e formatadas no
padrão que precisamos
#Agora vamos enviar os pares de valores + tempo pela caixa

msg_id = a[0] ##A id da mensagem sempre o elemento da
primeira linha, que será sempre armazenado na posição a[0]
```

```

# Esse id fixo, ent o ser apenas
definido uma vez

for k in range(1,len(a)): #Esse loop formar e enviar os pares
    (valor,tempo), todos com a mesma id fornecida pelo arquivo
    #Cada itera o do loop envia um par
del msg_data [:]
msg_data.append(a[k]) #Colocando o valor (contido na primeira
    linha) como primeiro elemento do vetor de dados

j = [int(char) for char in t[(k-1)]] #Essa linha converter
    um n mero em uma lista com seus digitos. Ex: 3000 =>
    [3,0,0,0]
                                                #Ele fara isso para um
                                                elemento do vetor t,
                                                extraido da segunda
                                                linha que contem os
                                                tempos

msg_data = msg_data+j #Feita essa conversao acima,
    adicionamos essa lista no vetor de dados
#print("msg_data:", msg_data) ##Descomentar para debugar
msg = can.Message(arbitration_id = msg_id,
    data=msg_data) #Cria a mensagem, utilizando os vetores
    previamente formatados
#O par metro extended_id so limitara a id de 0 a 7FF
bus.send(msg) #envia na rede
time.sleep(1) #sleep (em segundos) para n o enviar tudo de
    uma vez. E opcional usa-lo

#Note que apesar dos valores estarem convertidos em numeros
    decimais, eles sao convertidos automaticamente em hexa
    antes de enviar
#Ja que na realidade sao bytes sendo enviados

except KeyboardInterrupt:
    print("Interrupt pressionado — O programa parou")
finally:
    print("~~The End~~")

```