

UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

E DE COMPUTAÇÃO

Sistema Automatizado de Produção de Cerveja,

Monitorado e Controlado Remotamente

Autor: Leonardo Graboski Veiga

Orientador: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2015

Leonardo Graboski Veiga

Sistema Automatizado de Produção de Cerveja, Monitorado e Controlado Remotamente

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica

ORIENTADOR: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2015

Página com a ficha catalográfica (em página par).

página com a folha de aprovação (página ímpar).

Dedicatória

Dedico este trabalho aos meus pais.

Leonardo Graboski Veiga.

Agradecimentos

Tenho convicção de que quem somos pode ser sintetizado pela soma das interações sociais que acumulamos com o passar dos anos. É por isso que agradeço a todos que passaram por minha vida — cada um de vocês é parte daquilo que hoje sou. Não obstante, farei alguns agradecimentos direcionados às pessoas que não são cometas, mas sim pedaços de rocha eternos. Os planetas do meu sistema solar.

Agradeço à minha família pelo suporte incondicional: aos meus genitores Dirce Graboski e Gustavo Veiga, pela vida e por tudo aquilo que se sucedeu desde então; à minha avó Sofia Karachinski e o meu avô Victor Graboski, que me ensinaram que as virtudes e a inteligência de um ser humano vão muito além do estudo; aos meus padrinhos Luis Fernando Busnardo e Cinderela Busnardo, com os quais me diverti e aprendi um bocado; às minhas tias Inez Graboski e Glaci Edwiges Graboski, que foram minhas segundas mães durante a maior parte da minha vida; aos meus primos e primas André Luiz Graboski, Felippe Busnardo, Ana Paula Busnardo e Maria Fernanda Busnardo e; à esposa do meu pai, Elisabete Veiga.

Agradeço aos meus amigos de Curitiba pelas inúmeras situações que compartilhamos: Albert Faria, Pedro Henrique Bertolin, Douglas Auchewski, Eduardo Henrique Fasolin, Isabela Deolindo, Lucas Lupatini, Pamella Prohmann, Massimo Karly, Moacir Costa Filho, Lucas Vieira e Paulo Jacques Filho, dentre vários outros não menos importantes. Desde os tempos do Santa Maria e, posteriormente da UTFPR, até os dias de hoje, nunca aceitaram ver senão alegria em meus olhos e sempre estiveram dispostos a encarar uma boa aventura.

Agradeços aos meus amigos de São Carlos pela caminhada que fizemos juntos ao longo destes anos de faculdade: Rafael Hellman, com quem dividi moradia; Rafael Oishi, Vitor Martins e Gabriel Moreira, da Rep Hot Chili Peppers, minha segunda casa sancarlense; Renato Nunes Moraes e Renata de Camilo, pelo duradouro companheirismo e; a todos os outros amigos e colegas de classe que compartilharam experiência e conhecimento me prol de atingir um objetivo coletivo almejado por todos, chamado graduação.

Agradeço aos meus professores: em especial ao meu professor orientador Prof. Dr. Evandro Luis Linhari Rodrigues, que me incentivou a sempre buscar o melhor e me apresentou ao incrível mundo do Linux embarcado.

E a todos os outros que porventura eu tenha esquecido por um lapso de memória, mas não por falta de merecimento.

Também agradeço àqueles que contribuíram para o resultado final que é este trabalho de conclusão de curso.

Leonardo Graboski Veiga.

*"O sucesso é ir de fracasso em fracasso
sem perder o entusiasmo."*

Winston Churchill

Resumo

Este trabalho consiste na implementação de um sistema elétrico automatizado de produção de cerveja, baseado na plataforma de Linux embarcado BeagleBone Black. Foi desenvolvida uma interface de usuário para prover comunicação entre o operador e o sistema pela internet, enquanto o controle do processo de brassagem foi implementado em Node.js. O subsistema *PRU-ICSS* da BeagleBone foi utilizado para o controle de temperatura do processo, por meio de um controlador PID cujo atuador utilizado foi um resistor de potência. O sistema foi capaz de produzir 20 litros de cerveja sem intervenção do operador desde após a adição de maltes até o resfriamento do mosto cervejeiro.

Palavras-Chave: BeagleBone, Black, PID, PRU, node, controlador, brassagem, cerveja, automatizado, SBC.

Abstract

This work consists in the implementation of an automated electric brewing system, based in the BeagleBone Black embedded Linux platform. A user interface was developed to provide web communication between the operator and the system, while the brewing process control was deployed in Node.js. The BeagleBone subsystem *PRU-ICSS* was used to control the process temperature, by running a PID controller with a power resistor as the actuator. The system was capable of making 20 litres of beer without the operator's intervention since after the adding of the malts until the wort chilling.

Keywords: BeagleBone, Black, PID, PRU, node, controller, brewing, beer, automated, SBC.

Lista de Figuras

1.1	Número de cervejarias norte-americanas por categoria.	32
1.2	Cronograma do Trabalho de Conclusão de Curso	36
2.1	Programa de temperaturas típico de uma brassagem.	41
2.2	Dispositivos de mistura de grãos à brassagem.	43
2.3	Esquema de panela de brassagem industrial moderna.	44
2.4	Esquema de panela de mistura do mosto.	45
2.5	Diferentes configurações de caldeirões de fervura.	47
2.6	Arranjo para recuperação de energia.	49
2.7	Sistemas de clarificação.	50
2.8	Esquema de funcionamento do trocador de calor de placas.	51
2.9	BeagleBone Black e principais componentes	53
2.10	Diagrama de blocos de alto nível da BeagleBone Black	55
2.11	Configuração padrão dos pinos da BeagleBone Black	56
2.12	Diagrama de blocos do subsistema PRU-ICSS	57
2.13	Pinos da BeagleBone Black acessíveis pela PRU-ICSS	58
2.14	Fluxo de desenvolvimento de software da PRU	61
2.15	Sinal senoidal em função do tempo	71
2.16	Representação esquemática do diodo polarizado diretamente	71
2.17	Definição do circuito equivalente linear, usando-se segmentos de reta para aproximar a curva característica	72
2.18	Notações e símbolos utilizados para a configuração base-comum	73
2.19	Símbolos do JFET e do MOSFET	74
2.20	Obtendo a curva de transferência das curvas de dreno de um JFET	75
2.21	Arquitetura interna do DS18B20	77
2.22	Mapa de memória do DS18B20	77

2.23 Registradores de temperatura do DS18B20	77
4.1 Representação esquemática da estrutura mecânica	84
4.2 Sistema CIP básico	86
4.3 Diagrama conceitual das conexões mecânicas	88
4.4 Diagrama da panela de mostura e conexões	89
4.5 Detalhes da vedação da panela de mostura	89
4.6 Projeto da estrutura metálica	90
4.7 Dimensões da estrutura de adição de lúpulos	92
4.8 Corpo da estrutura de adição de lúpulos	92
4.9 Tampas da estrutura de adição de lúpulos	93
4.10 Esquema de ligação das varetas à tampa de correr	93
4.11 Estrutura de adição de lúpulos	94
4.12 Últimas imagens pré-compiladas disponíveis para a BBB	96
4.13 Configuração do <i>Putty</i> para acesso SSH via USB	97
4.14 Tentativa de login bem sucedida pelo <i>Putty</i>	97
4.15 Acesso à BBB via SSH	98
4.16 Leitura do sensor DS18B20 pelo terminal	101
4.17 Estrutura de diretórios da aplicação da BBB	102
4.18 Ambiente completo do IDE <i>Cloud9</i>	106
4.19 Criação de repositório no <i>GitHub</i>	107
4.20 Fazendo git commit após mudanças incrementais	108
4.21 Arquivo CSV com registro de temperaturas gerado em Python	110
4.22 Gráfico com 7200 pontos gerado em Python	112
4.23 Geração de gráfico em Python múltiplas vezes para obter o tempo médio	113
4.24 Exemplo de uso do módulo <i>Debug</i>	115
4.25 Resultado de chamada de função sem criação de escopo x com criação de escopo	120
4.26 Representação em fluxograma do algoritmo de implementação da função <i>getTemperatureReading</i>	126
5.1 Página inicial da UI	133
5.2 Página de apresentação da UI	134
5.3 Gráfico dinâmico de temperatura em função do tempo	135

5.4	Gráfico estático do histórico de temperaturas registradas	135
5.5	Menu de tarefas da UI	136
5.6	Gerenciador de receitas da UI	136
5.7	Editor de receitas da UI	137
5.8	Gerenciador de início da produção	137
5.9	Interface de acompanhamento e ajustes da brassagem - console Javascript com <i>feedback</i> do servidor para uma sequência de comandos executados . . .	138
5.10	Aviso de página em desenvolvimento	138
6.1	Distribuição temporal das macrotarefas do semestre	139

Listas de Tabelas

1.1	Atividades propostas do Trabalho de Conclusão de Curso	35
2.1	Características de diferentes sistemas de fervura.	48
2.2	Especificações Gerais da BeagleBone Black	54
2.3	Mapa de Memória Local da PRU	59
2.4	Mapa de Memória Global da PRU	60
2.5	Descrição dos campos do registrador de controle de <i>pads</i>	65
2.6	Lista de pinos do <i>header</i> P8 e seus respectivos modos de funcionamento . . .	66
2.7	Lista de pinos do <i>header</i> P9 e seus respectivos modos de funcionamento . . .	66
3.1	Lista de materiais que compõem o núcleo do projeto	79
3.2	Lista de materiais utilizados na confecção do subsistema mecânico	80
3.3	Lista de componentes eletrônicos diversos empregados na confecção de PCBs e cabeamento do sistema	81
3.4	Lista de equipamentos, instrumentos e <i>softwares</i> de suporte ao desenvolvimento do projeto	82
3.5	Lista insumos cervejeiros para produção de cerveja do tipo <i>Blond Ale</i>	82
4.1	Dimensões elementos mecânicos relevantes para o projeto da altura da estrutura metálica	90
4.2	Lista de diretórios e arquivos da aplicação da BBB	103
4.3	Estatísticas referentes ao tempo de geração de gráfico na BBB, para 7200 pontos em Python	113
4.4	Correspondência entre o valor numérico e o significado dos códigos usados para registro da brassagem	129
II.1	Registradores do módulo de controle	207

Listas de Códigos-fonte

4.1	Passos para configuração do Wi-Fi	98
4.2	Configuração para reset do Wi-Fi após o boot	98
4.3	Instalação e configuração do NTP	99
4.4	Servidores brasileiros do NTP	99
4.5	Instalação do DTC	100
4.6	Fragmento de device tree para o DS18B20	100
4.7	Compilação de <i>device tree</i>	100
4.8	Reinicialização do servidor web Apache	101
4.9	Primeira tentativa de instalação do <i>Cloud9 IDE core</i> na BBB	104
4.10	Instalação bem sucedida do <i>Cloud9 IDE core</i> na BBB	105
4.11	Execução do <i>Cloud9</i>	105
4.12	Instalação do <i>Git</i> e cópia do repositório do <i>GitHub</i> para a BBB	107
4.13	Primeiro <i>git commit</i>	108
4.14	Função de <i>log</i> da temperatura em Python	109
4.15	Script para gravação das leituras de temperatura em Python	110
4.16	Instalação da biblioteca Matplotlib	110
4.17	Instalação do ecossistema SciPy	111
4.18	Importação das bibliotecas para geração de gráfico em Python	111
4.19	Instalação e/ou atualização do Node.js	113
4.20	Criação do diretório <i>/var/www/node_modules</i>	114
4.21	Instalação do módulo <i>Debug</i>	114
4.22	Exemplo de uso do módulo <i>Debug</i>	114
4.23	Instalação do Octalbonescript	116
4.24	Modificações feitas ao template de <i>device tree</i> do Octalbonescript	116
4.25	Criação de objetos para os elementos do sistema em Node	117

4.26	Uso de clausura para criação de escopo	119
4.27	Instalação dos módulos necessários para implementar o servidor web em Node.js	120
4.28	Configuração do PHP-express	121
4.29	Configuração do servidor web com o <i>framework</i> Express.js	121
4.30	Exemplo de uso dos argumentos <i>req</i> e <i>res</i> definidos pelo Express.js	123
4.31	Declaração do objeto <i>environmentVariables</i>	127
4.32	Uso da função <i>JSON.stringify</i> para codificar um objeto em string JSON	128
B.1	Módulo com as funções para <i>log</i> da temperatura	149
B.2	Script para plotagem de gráfico	152
C.1	Código-fonte raiz da aplicação em Node.js	155
C.2	Módulo de gerenciamento de GPIO e PWM	157
C.3	Módulo de roteamento do servidor web	162
C.4	Módulo de registros e verificações em geral	165
C.5	Código-fonte raiz da aplicação em Node.js	172
D.1	/etc/network/interfaces	185
D.2	w1.dts	186

Siglas

ABV	<i>Alcohol by Volume</i> - Porcentagem do volume de álcool na cerveja
API	<i>Application Programming Interface</i> - Interface de Programação de Aplicação
BBB	BeagleBone Black
BJT	<i>Bipolar Junction Transistor</i> - Transistor Bipolar de Junção
BK	<i>Brewing Kettle</i> - Panela de Fervura
CI	Círculo Integrado - equivalente a IC (<i>Integrated Circuit</i>) e muitas vezes referido como <i>chip</i> .
CIP	<i>Clean in Process</i> - Limpeza no Local, é a limpeza automatizada dos equipamentos
CRC	<i>Cyclic Redundancy Check</i> - Verificação de Redundância Cíclica
DDP	Diferença de potencial
DT	<i>Device Tree</i> - Uma estrutura de dados em árvore usada para descrever <i>hardware</i>
DTC	<i>Device Tree Compiler</i> - Compilador de <i>Device Tree</i>
FET	<i>Field Effect Transistor</i> - Transistor de Efeito de Campo
GPIO	<i>General Purpose Input/Output</i> - Entrada/Saída de Propósito Geral
GUI	<i>Graphical User Interface</i> - Interface Gráfica de Usuário
HID	<i>Human Interface Device</i>
HLT	<i>Hot Liquor Tank</i> - Tanque de Água Quente
IBU	<i>International Bitterness Unit</i> - Unidade de Amargor Internacional
IDE	<i>Integrated Development Environment</i> - Ambiente de Desenvolvimento Integrado
IEP	<i>Industrial Ethernet Peripheral</i> - Periférico Ethernet Industrial
IoT	<i>Internet of Things</i> - Internet das Coisas
LSB	<i>Least Significant Bit</i> - Bit Menos Significativo
MOSFET	<i>Metal-Oxide Semiconductor Field Effect Transistor</i> - Transistor de Efeito de Campo Metal-Óxido Semicondutor
LT	<i>Lauter Tun</i> - Panela de Filtragem

MSB	<i>Most Significant Bit</i> - Bit Mais Significativo
MT	<i>Mash Tun</i> - Panela de Mostura
NPM	<i>Node Package Manager</i> - Gerenciador de Pacotes do Node
OG	<i>Original Gravity</i> - Gravidade Original. É a densidade do mosto após o cozimento dos grãos
PID	<i>Proportional-Integral-Derivative-Controller</i> - Controlador Proporcional Integral Derivativo
PCB	<i>Printed Circuit Board</i> - Placa de Circuito Impresso. Embora PCI seja a abreviação em português, o termo PCB foi escolhido para uso no intuito de não confundir o leitor com o barra
REST	<i>Representational State Transfer</i> - Transferência de Estado Representacional
RISC	<i>Reduced Instruction Set Computer</i> - Computador com um conjunto reduzido de instruções
SaaS	<i>Software as a Service</i> - Software como um Serviço
SO	<i>Sistema Operacional</i> - também conhecido como OS (<i>Operating System</i>)
SoC	<i>System on Chip</i> - Sistema em um Chip
SBC	<i>Single Board Computer</i> - Computador de Placa Única
SDK	<i>Software Development Kit</i> - Kit de Desenvolvimento de Software
SSH	<i>Secure Shell</i> - Protocolo de comunicação entre computadores
UI	<i>User Interface</i> - Interface de Usuário
VCS	<i>Version Control System</i> - Sistema de Controle de Versão

Sumário

1	Introdução	31
1.1	Motivação	31
1.2	Objetivos	33
1.2.1	Cronograma	34
1.3	Justificativas/relevância	36
1.4	Organização do Trabalho	36
2	Embasamento Teórico	39
2.1	Processo de Produção de Cerveja	39
2.2	Estrutura mecânica	42
2.2.1	Panela de mostura	43
2.2.2	Caldeirão de fervura	45
2.2.3	Adição de lúpulos	49
2.2.4	Clarificação e resfriamento do mosto	49
2.3	BeagleBone Black	52
2.3.1	<i>Programmable Real-time Unit – PRU-ICSS</i>	56
2.3.2	<i>Device Tree</i>	61
2.4	Programação em Node.js (Javascript)	67
2.4.1	<i>Node Package Manager - NPM</i>	68
2.4.2	<i>MEAN Stack</i>	69
2.4.3	Servidor Express.js	69
2.5	Circuitos de Interface	70
2.5.1	Sinais	70
2.5.2	Dispositivos Semicondutores	71
2.5.3	Sensor de temperatura DS18B20	75

2.6	Sistema de controle de temperatura	78
2.6.1	Controlador PID	78
2.6.2	Interface com sensores e atuadores	78
3	Materiais	79
3.1	Núcleo do projeto	79
3.2	Sistema mecânico	79
3.3	Componentes eletrônicos	80
3.4	Equipamentos, instrumentos, <i>softwares</i> e miscelânea	81
3.5	Insumos cervejeiros	82
4	Métodos	83
4.1	Estrutura mecânica	83
4.1.1	Funcionamento da estrutura	83
4.1.2	Dimensionamento da parte funcional	86
4.1.3	Estrutura metálica de suporte	89
4.2	Estrutura de adição de lúpulos	91
4.3	Configuração da BeagleBone Black	95
4.3.1	Ajustes de rede para uso do adaptador Wi-Fi/USB	97
4.3.2	Data/hora	99
4.3.3	Sensor DS18B20	100
4.3.4	Webserver Apache	101
4.4	Organização do diretório da aplicação	101
4.5	IDE e sistema de controle de versão	104
4.5.1	<i>Cloud9</i>	104
4.5.2	<i>Git/GitHub</i>	106
4.6	Geração de gráfico e registro de temperatura em Python	108
4.6.1	Registro de temperatura	108
4.6.2	Gráfico de temperatura	110
4.7	Aplicação <i>server-side</i> em Node.js	113
4.7.1	Acesso a GPIO e PWM	115
4.7.2	Servidor Express.js	120
4.7.3	Registros e verificações em geral	124
4.7.4	Controle do processo de brassagem	129

4.7.5	Simulação do controle do processo de brassagem	131
4.8	Interface de usuário	131
4.8.1	Tratamento de formulário em PHP	131
4.9	Circuitos de interface entre a BBB e sensores/atuadores	132
4.9.1	Acionamentos de potência	132
4.9.2	Detector de <i>zero-crossing</i>	132
4.9.3	Detector de nível de líquido	132
4.10	Sistema de controle de temperatura	132
4.10.1	Resistores de aquecimento	132
4.10.2	Controlador PID	132
5	Resultados e Discussões	133
5.1	Interface de usuário	133
6	Conclusão ou Conclusões	139
A	Códigos-fonte da interface de usuário	147
B	Scripts Python	149
B.1	Módulo com as funções para <i>log</i> da temperatura	149
B.2	Script para plotagem de gráfico	152
C	Códigos-fonte Node.js	155
C.1	Código-fonte raiz da aplicação em Node.js	155
C.2	Módulo de gerenciamento de GPIO e PWM	157
C.3	Módulo de roteamento do servidor web	162
C.4	Módulo de registros e verificações em geral	165
C.5	Módulo de controle do processo de brassagem	172
D	Arquivos de configuração do sistema	185
D.1	Arquivo de configuração de rede	185
D.2	Device Tree para o DS18B20	186
E	Cálculo de OG e IBU de uma receita de cerveja	189
E.1	Estimação do IBU para uma receita de 20l	190

F Análise do tempo de leitura do sensor DS18B20 em Python	193
I Diagrama de conexões elétricas da BeagleBone Black	195
II Registradores do Módulo de Controle	207
III Válvula Danfoss EV210BD 032U3620	213
IV Bobina Danfoss 042N7550	221
V Bomba centrífuga Topsflo B08H121006	225

Capítulo 1

Introdução

O objetivo deste Trabalho de Conclusão de Curso é o projeto de um sistema elétrico automatizado de produção de cerveja, monitorado e controlado remotamente. A plataforma de Linux embarcado BeagleBone Black foi escolhida para ser o núcleo de processamento de dados, cuja linguagem de programação escolhida foi o Javascript em conjunto com o interpretador Node.js; comunicação via *web* desenvolvida em HTML, CSS, Javascript e PHP e; controle do processo, de tal modo que foi necessário projetar circuitos de interface entre a placa e os sensores e atuadores do sistema.

1.1 Motivação

Há algumas décadas começou nos Estados Unidos da América um movimento espontâneo no qual diversos cidadãos norte-americanos começaram a produzir cervejas por conta própria. Estes foram denominados *homebrewers*, ou cervejeiros caseiros, e eles criaram um mercado de insumos e equipamentos para brassar - ou seja, produzir cerveja - em pequenas quantidades e de forma artesanal, utilizando-se de processos e ferramentas improvisados para tal [1].

Apesar das ferramentas de produção rudimentares empregadas no início do movimento de produção caseira de cerveja — e que são utilizadas até hoje — percebeu-se que a qualidade do resultado final podia ser tão boa quanto a de cervejas comerciais de alta qualidade. O movimento cervejeiro cresceu e a partir dele nasceram diversas microcervejarias: segundo dados da *Brewers Association*, em 1994 existiam 192 microcervejarias e 329 *brewpubs* (bares que produzem a própria cerveja) nos Estados Unidos, enquanto em 2014 este número cresceu para 1871 microcervejarias e 1412 *brewpubs* [2], indicando uma tendência de mercado no que diz respeito às cervejas artesanais: a figura 2.8 ilustra os números que apoiam esta tendência.

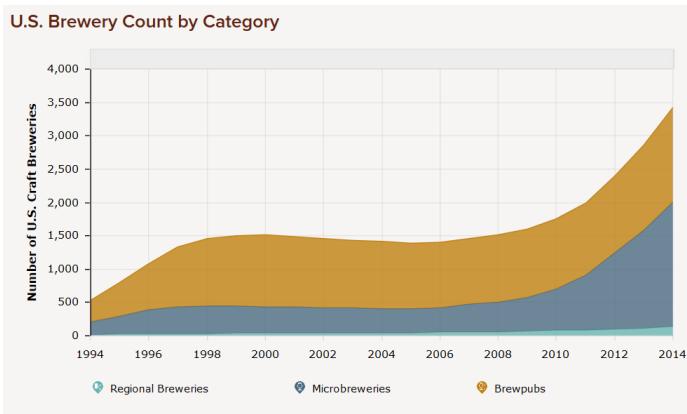


Figura 1.1: Número de cervejarias norte-americanas por categoria.

Fonte: Site da Brewers Association¹

No Brasil, o movimento cervejeiro artesanal caseiro é considerado por algumas fontes um mercado incipiente, porém com bom ritmo de crescimento [3], devido em grande parte à tendência de mercado em relação às microcervejarias. Desde 1995, com a fundação da cervejaria Dado Bier, no estado do Rio Grande do Sul, que alega ser a primeira microcervejaria do Brasil [4], seguida pela cervejaria Colorado, do estado de São Paulo, fundada em 1996 [5], o movimento cervejeiro cresceu e, a partir do início da década de 2010 se popularizou, com estimativa de que existam atualmente cerca de 200 microcervejarias no país [3]. Esta busca por diferenciação do consumo é refletida também no aumento do coeficiente de penetração de importações neste setor, que indica a quantidade percentual de importações: este cresceu de 0,02% em 2005 para 0,21% em 2011 [3]. Cabe ressaltar que a indústria da cerveja no Brasil, como um todo, corresponde a 2% do PIB do país, representando um faturamento anual de R\$70 bilhões [6], portanto o mercado de artesanais, que representa cerca de 1% do total, corresponde a um faturamento de R\$700 milhões, com prognóstico de crescimento para 2% em até dez anos [7].

Além disto, existem sistemas de produção automática de cerveja que são comercializados em outros países, a exemplo do PicoBrew [8], dos Estados Unidos, cuja ideia é similar à de uma cafeteira automática e do Williams Warn [9], da Nova Zelândia, que é um projeto compacto mais parecido com o equipamento de um produtor caseiro, apesar de completamente automatizado.

A motivação para este trabalho, portanto, surgiu a partir da constatação de que o mercado de cervejas artesanais é crescente e rentável, tanto no que diz respeito a fornecer equipamentos para produtores caseiros, quanto ao fornecimento de equipamentos e tecnologia para

¹Disponível em <https://www.brewersassociation.org/statistics/number-of-breweries/>

micro e pequenas cervejarias. Outro ponto que cabe ressaltar é que a maioria das indústrias nacionais fornecedoras de equipamentos para o mercado cervejeiro – dentre elas Mec Bier Microcervejarias, Bierking Equipamentos, Biermatik Comercial Ltda., Cervejando.com e Mybeer, dentre outros – oferece, na melhor das hipóteses, soluções semi-automatizadas para operação presencial, a despeito das soluções automáticas encontradas em outros países, o que motivou a automação mais complexa e remota deste trabalho. A motivação final deste projeto não é a criação de um produto, mas sim a compreensão e aplicação de tecnologias à área de automação do processo de produção de cerveja.

1.2 Objetivos

Para melhor entender os objetivos propostos, uma breve introdução ao processo de produção de cerveja é necessária, visando a identificação dos processos que podem ser automatizados. As principais etapas são: moagem dos maltes - moagem dos grãos maltados de cevada; mosturação - cozimento dos maltes seguindo rampas e patamares específicos de temperatura; filtragem do mosto - recirculação do líquido (mosto) por um filtro, com o objetivo de evitar que ele apresente material particulado; lavagem dos grãos - processo opcional no qual, após separar o mosto dos grãos, estes são lavados para extrair o máximo possível dos seus açúcares fermentáveis; fervura do mosto - processo de fervura do mosto por um tempo predeterminado e adição dos lúpulos ao longo do processo; *whirlpool* - separação das proteínas coaguladas e material particulado do líquido, geralmente feito por centrifugação; resfriamento - diminuição abrupta da temperatura do mosto a fim de evitar sua contaminação por bactérias, utilizando um *chiller*, também conhecido como trocador de calor; adição das leveduras e fermentação - processo que transforma os açúcares fermentáveis da cerveja em álcool e gás carbônico; maturação - processo de envase ou consumo do barril - o envase seguido de uma refermentação da garrafa serve para carbonatar a cerveja, enquanto ao consumir diretamente do barril, é preciso ter um equipamento que possibilite a carbonatação forçada da mesma [1].

Conhecido o processo de produção, é possível entender o que se espera obter do trabalho: projeto de um sistema embarcado multiprocessado que automatiza o processo de produção de cerveja desde o cozimento do mosto até seu resfriamento. A moagem dos grãos e as etapas subsequentes ao resfriamento não estão inclusas no escopo deste projeto, uma vez que a soma de toda a automação seria impraticável em função do orçamento e do tempo disponíveis. Com relação ao controle e monitoramento, o microcontrolador ARM do sistema deve executar

uma distribuição de Linux embarcado, sobre a qual será implementado um servidor web que hospede a interface de usuário e também parte do código de controle do sistema. A interface possibilita ao usuário adicionar, editar e excluir receitas; iniciar, monitorar e modificar *on-the-fly* o processo automático de produção de cerveja; apresentar estatísticas do sistema e; oferecer opções de configuração ao usuário.

Os objetivos gerais e específicos do projeto são:

- Implantação de distribuição Debian do Linux no sistema BeagleBone Black
- Projeto de interfaceamento entre sensores diversos e o sistema BeagleBone Black
- Projeto, aquisição dos componentes e montagem do sistema mecânico
- Levantamento das plantas térmicas a serem controladas
- Projeto dos controladores do sistema de temperatura e implementação no subsistema PRU-ICSS
- Projeto do acionamento dos resistores de potência
- Projeto e implementação do programa de controle de acionamento de válvulas e bombas, e adição de ingradientes
- Desenvolvimento da interface web do sistema
- Detecção e tomada de decisões em caso de interrupção no fornecimento de energia
- Teste e validação do funcionamento do sistema completo
- Verificação organoléptica subjetiva da qualidade da cerveja, objetivando a comprovação do funcionamento do sistema

1.2.1 Cronograma

O cronograma previsto para a realização deste projeto contém as atividades e suas respectivas alocações de tempo. As atividades são listadas na tabela 1.1 e o cronograma é exposto na figura 1.2.

Tabela 1.1: Atividades propostas do Trabalho de Conclusão de Curso

Atividade	Descrição
1	Instalação do Debian na memória eMMC da BeagleBone Black
2	Acesso aos terminais de E/S da BeagleBone Black
3	Instalação e uso de módulo WI-FI USB
4	Interfaceamento e uso de sensores de temperatura DS18B20
5	Projeto da parte mecânica
6	Aquisição das peças e montagem da parte mecânica
7	Dimensionamento e aquisição dos resistores de aquecimento
8	Projeto, teste e interfaceamento do circuito de acionamento das bombas de recirculação e válvulas solenóide
9	Teste de funcionamento da parte mecânica, com controle manual
10	Projeto do acionamento dos resistores de potência, utilizando relé de estado sólido
11	Projeto, teste e interfaceamento de sistema de detecção de vazão
12	Projeto dos sistemas de adição controlada de insumos
13	Levantamento das plantas térmicas do sistema
14	Projeto dos controladores PID digitais e implementação no subsistema PRU-ICSS
15	Desenvolvimento da interface web para cadastro de receitas de cerveja
16	Desenvolvimento da interface web para acompanhamento do processo de brasagem
17	Desenvolvimento da interface web para controle e ajustes do processo de brasagem <i>on-the-fly</i>
18	Desenvolvimento da interface web para acesso a estatísticas das produções anteriores
19	Teste completo do sistema
20	Pesquisa de avaliação organoléptica da cerveja produzida, para validação do sistema
21	Redação da monografia

	jul/15	ago/15	set/15	out/15	nov/15	dez/15	jan/16	fev/16	mar/16	abr/16	mai/16	jun/16
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												
21												

Figura 1.2: Cronograma do Trabalho de Conclusão de Curso

1.3 Justificativas/relevância

Este trabalho objetiva a aplicação de conhecimentos e técnicas – além da busca pelo estado da arte – nas áreas de sistemas de controle, circuitos elétricos e eletrônicos, instrumentação, microcontroladores e sistemas digitais. Conhecimentos estes obtidos durante o curso de graduação em Engenharia Elétrica - Ênfase em Eletrônica.

Outro foco é o aprendizado de novos conhecimentos, majoritariamente nas áreas de Linux embarcado, desenvolvimento de aplicações *web* e engenharia mecânica.

Este trabalho é importante por criar métodos de automação para uma indústria representativa no mercado nacional e, no que diz respeito ao desenvolvimento de um produto, ao incentivo da população ao consumo consciente de cervejas — bandeira levantada pelas microcervejarias artesanais nacionais, com o lema "beba menos, beba melhor".

1.4 Organização do Trabalho

Este trabalho está distribuído em XXX capítulos, incluindo esta introdução, dispostos conforme a descrição que segue:

Capítulo 2: Descreve

Capítulo 3: Discorre sobre

Capítulo 4: Apresenta

Capítulo 2

Embasamento Teórico

Neste capítulo, será apresentado o embasamento teórico utilizado para o desenvolvimento deste Trabalho de Conclusão de Curso, abordando desde o processo de produção de cerveja e os equipamentos utilizados até os sistemas de controle e plataforma de implementação da interface de usuário, dentre outros tópicos.

2.1 Processo de Produção de Cerveja

O primeiro passo para o desenvolvimento do controle de automatização da produção de cerveja é o entendimento dos seus processos e particularidades. Esta seção tem o objetivo de apresentar uma introdução à fabricação de cerveja, o que implica em significativas simplificações e resumos dos processos apresentados. Para maiores informações sobre o tema, recomenda-se a consulta do material bibliográfico de referência.

A produção começa com a escolha e **moagem** dos maltes, produzidos a partir de cereais selecionados, sendo a cevada o cereal mais amplamente utilizado – embora outros grãos, como trigo, centeio e aveia também possam ser empregados [10]. O grão malteado é a fonte principal de açúcares fermentáveis utilizado na fabricação de cerveja e é produzido a partir da germinação parcial do cereal, sendo que diferentes processos e variedades de grãos resultam em diferentes qualidades de maltes. Estes, por sua vez, conferem características únicas aos diversos estilos de cerveja [1, 10]. O método de moagem do malte é escolhido com base nos métodos de brassagem e separação de mosto a serem empregados [10], e.g. em uma moagem na qual a cama de grãos formada no processo serve como um filtro para separação do mosto, deve-se preservar a casca, utilizando uma moagem grossa e de maceração, evitando Trituração.

Com os maltes moídos, segue-se para a **brassagem**, que consiste na mistura de determinada quantidade de água quente aos maltes, possibilitando a extração e diluição dos seus açúcares fermentáveis. Enquanto um extrato em água fria tem rendimento na ordem de 15-22%, o HWE (*hot water extract*) – extrato em água quente, chega à ordem de 75-83% devido à atividade enzimática catalisadora [10].

O processo de brassagem pode ser realizado de várias maneiras, dentre elas [10]:

- (a) infusão simples, na qual os maltes são cozidos a uma temperatura fixa, quase isotérmica, utilizado tradicionalmente por cervejarias britânicas. É feita em uma tina de brassagem, na qual tanto o processo de extração dos açúcares quanto a separação do extrato do mosto são realizados. O cozimento se dá a uma temperatura na ordem de 63°C a 67°C, durante 30 minutos a duas horas e meia. O líquido da panela (extrato) é re-circulado para que as partículas sólidas sejam separadas deste e, por isso, é importante que a moagem dos grãos seja grossa, já que estes assentam na tina e servem como filtro. Após a separação do extrato, a cama de grãos formada é lavada com um spray de água quente, processo denominado *sparging*;
- (b) decocção, na qual a moagem dos grãos é mais fina e os maltes utilizados são pouco modificados: devido à moagem fina, os grãos podem ser bombeados ou misturados. Este método usa três recipientes: um recipiente de mistura da mosturação, um recipiente de decocção e um dispositivo de separação do extrato do mosto. Neste processo retira-se uma quantidade de líquido, que está a uma temperatura inicial de 35°C, e este é fervido e adicionado novamente ao restante. Após a mistura, a temperatura do mosto deve ser de cerca de 50°C e deve permanecer assim por um período de tempo. O processo é repetido outras duas vezes, para obter as temperaturas de 65°C e 76°C respectivamente;
- (c) dupla infusão; e
- (d) maceração escalonada, infusão por temperaturas programadas ou infusão por degraus de temperatura é um processo que tem substituído os outros sistemas, em função de sua praticidade e economia de energia, podendo economizar de 30 a 50% com relação a um programa de decocção similar. A extração de açúcares é feita de forma análoga ao processo de infusão simples, com a diferença de que as temperaturas do processo são controladas em rampas e patamares. A filtragem do extrato do mosto pode ser feita tanto utilizando a técnica de cama de grãos do processo de infusão simples quanto

a filtragem direta da decocção. A figura 2.1 apresenta o exemplo de três programas de temperatura de brassagem - de cima para baixo, os gráficos da figura se referem a brassagens por degraus, decocção simples e decocção dupla. No processo por degraus, o aumento de temperatura entre os patamares deve ser da ordem de 1°C/min.

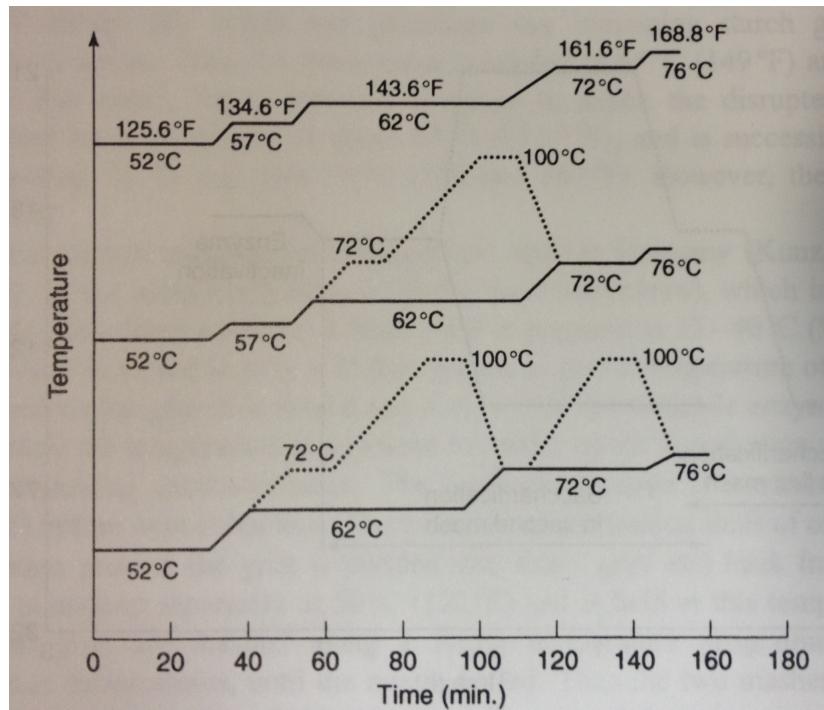


Figura 2.1: Programa de temperaturas típico de uma brassagem.

Fonte: BRIGGS (2011)

Após a **filtragem** do extrato do mosto, este é transferido para um caldeirão de **fervura**, processo no qual são adicionados os lúpulos e que leva cerca de uma hora, mas que pode durar mais tempo dependendo da receita [10], a exemplo das cervejas comerciais *90 Minute IPA* e *120 Minute IPA*, da cervejaria Dogfish Head, cujo tempo de fervura dura 90 minutos e 120 minutos, respectivamente [11, 12]. Lúpulo é uma flor em formato de cone e é utilizado para conferir amargor e aroma à cerveja, além de ser um ótimo conservante natural. Pode ser adicionado ao mosto em flores, *pellets* (pastilhas prensadas) ou extrato. Quando adicionado no início da fervura, contribui para o amargor da cerveja, por meio da isomerização de ácidos alfa que ocorre em função da alta temperatura. Em contrapartida, compostos aromáticos voláteis são perdidos neste processo e, por isso, também é adicionada uma quantidade de lúpulos ao final da fervura, para contribuir com o aroma [1, 10].

Além de conferir amargor à cerveja, por meio da adição de lúpulos, a fervura também é responsável pela coagulação de proteínas indesejáveis, assepsia do líquido, evaporação e con-

sequente redução do seu volume, mudanças no sabor da cerveja e evaporação de compostos voláteis indesejáveis [10].

A seguir, o mosto fervido deve ser **resfriado** rapidamente até 26°C para evitar oxidação, contaminação e criação de compostos orgânicos que introduzem sabores indesejados [1]. O processo de resfriamento é realizado com o emprego de um trocador de calor, denominado *chiller*. Também devem ser separadas e desprezadas as proteínas coaguladas e os restos de lúpulos decorrentes da fervura. Este processo geralmente é denominado *whirlpool*, em função de sua natureza, na qual o mosto fervido é centrifugado e os compostos indesejados se acumulam no centro e no fundo da panela de fervura [10]. O último passo é aerar ou até mesmo oxigenar o mosto, para que as leveduras possam se reproduzir corretamente no início da **fermentação** [10].

A levedura deve ser adicionada ao mosto resfriado e oxigenado o mais rápido possível para evitar contaminações, e o recipiente de fermentação comumente é selado, evitando a entrada de ar [10]. O processo de fermentação e os subsequentes processos de **maturação** e **envase** não serão detalhados, pois sua automação não é parte do escopo deste trabalho.

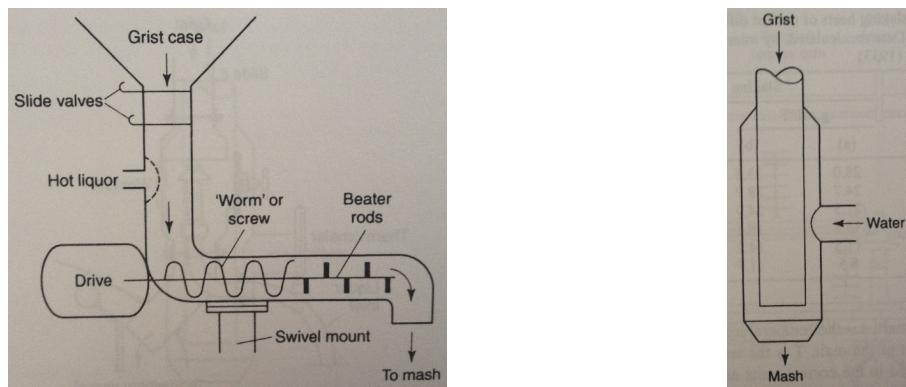
2.2 Estrutura mecânica

O processo de brassagem, que consiste no cozimento dos maltes para extração dos açúcares fermentáveis e filtração do líquido resultante, conhecido como mosto, é realizado em equipamentos específicos para estas tarefas. Em função das diferentes técnicas de brassagem e tecnologias desenvolvidas ao longo do tempo, diversos tipos e arranjos de equipamentos foram desenvolvidos [10].

Atualmente existe uma convergência nos métodos de fabricação de cervejas, motivada principalmente por questões financeiras, cujo objetivo do equipamento é maximizar a produção. Outros motivos para o desenvolvimento e aprimoramento de equipamentos são a necessidade de sempre produzir uma cerveja com as mesmas características, ou seja, padronizar a produção e, também, a preocupação crescente com redução do uso de energia e água e a redução da produção de efluentes [10]. Ainda assim, equipamentos antigos continuam em uso, seja pela impossibilidade de reproduzir a cerveja em equipamentos mais modernos ou pelo custo elevado da atualização das plantas de produção.

Na adição dos grãos à panela de mostura, o processo de mistura destes à água é importante para que a brassagem seja eficiente, uma vez que os grãos mal misturados podem formar

aglutinados que impedem a extração dos açúcares. Para evitar este inconveniente, dispositivos mecânicos foram desenvolvidos, conforme exposto na figura 2.2: em (a) a água e os grãos são misturados em um fluxo constante, determinado pela velocidade de giro de uma rosca sem fim e; em (b) a água é adicionada aos maltes em um ângulo tangente, de tal forma que um vórtex a mistura aos grãos. A temperatura da água também deve ser controlada, para evitar que proteínas sejam inativadas e seu volume inicial é predeterminado conforme a receita [10].



(a) Dispositivo com rosca sem fim

(b) Dispositivo de mistura por vórtex

Figura 2.2: Dispositivos de mistura de grãos à brassagem.

Fonte: BRIGGS (2011)

2.2.1 Panela de mostura

A panela de mostura, ou MT (*mash tun*), é o dispositivo mais simples para a preparação do mosto, uma vez que nela ocorre a extração dos açúcares, a filtração do extrato do mosto e a lavagem dos grãos [10]. A figura 2.3 apresenta uma configuração comum de MT, usada atualmente.

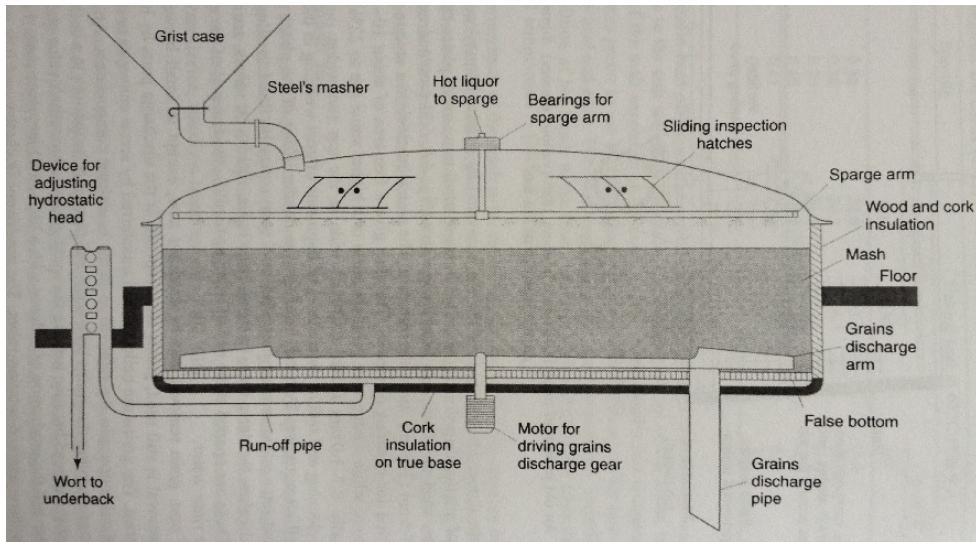


Figura 2.3: Esquema de panela de brassagem industrial moderna.

Fonte: BRIGGS (2011)

As MTs possuem seção transversal circular e atualmente são feitas de aço inoxidável, revestidas com material isolante térmico. Embora originalmente fossem abertas, atualmente são cobertas para evitar perda de calor e espalhamento do vapor de água pelo ambiente da brassagem. O fundo da MT é coberto por um fundo falso, que fica a alguns centímetros acima do fundo original e cuja função é atuar como filtro, em conjunto com a cama de grãos. Este fundo falso consiste de uma ou mais chapas de metal com rasgos da ordem de 0,7-1,0mm, que representam uma área livre para a drenagem de cerca de 12% da área total, ou uma trama com fios de metal cuja área livre chega a 22% [10].

O descarte dos grãos é feito através de um tubo, com a ajuda de uma pá que varre o fundo da panela ou um jato de ar comprimido. Métodos de remoção dos grãos descontinuados são a remoção manual (que ainda existe em pequenas instalações) e o enxágue dos grãos com bombeamento para um tanque anexo – este não mais usado em função da necessidade de mais água e posterior tratamento desta, que resulta em custos adicionais, além da perda do valor comercial dos grãos a serem descartados. Com relação à limpeza, as MTs são construídas com suporte à limpeza CIP (limpeza no local), inclusive na região entre o fundo da panela e o fundo falso [10].

A água da lavagem dos grãos, ou *sparging*, é borrifada por um cano ou mais canos suspensos acima da cama de grãos, conforme ilustrado na figura 2.3. Estes canos são conhecidos como braços de *sparging* e geralmente giram no eixo em função da força da água, o que só é possível devido aos furos no tubo serem feitos na vertical, especialmente para este propósito. Outro aspecto da construção dos braços de *sparging* é que, à medida que se aproxima do

centro da panela, os furos são feitos a uma distância menor entre si, possibilitando que a água seja igualmente distribuída sobre a cama de grãos [10]. O extrato do mosto é coletado por tubos no fundo da panela.

No caso da brassagem feita pelo processo de decocção, brassagem dupla ou infusão por temperatura controlada, diferentes recipientes de brassagem são utilizados. A figura 2.4 é um exemplo de tina de mistura (não confundir com mostura, embora a panela seja utilizada no processo de mostura) usada para brassagem de temperatura controlada. Nestes casos, a moagem dos maltes é mais fina e estes são constantemente misturados por uma pá no fundo da panela. Mesmo dentre estes sistemas os equipamentos diferem entre si e, diferente da MT, nestes sistemas é preciso utilizar outro recipiente para separar o extrato dos grãos. O separador pode ser uma tina de lavagem, conhecida também como LT (*lauter tun*), ou pode ser um filtro de mosto. Embora a LT e o filtro de mosto sejam amplamente utilizados na indústria, não serão detalhados neste trabalho, já que a abordagem prática adotada não inclui seu uso.

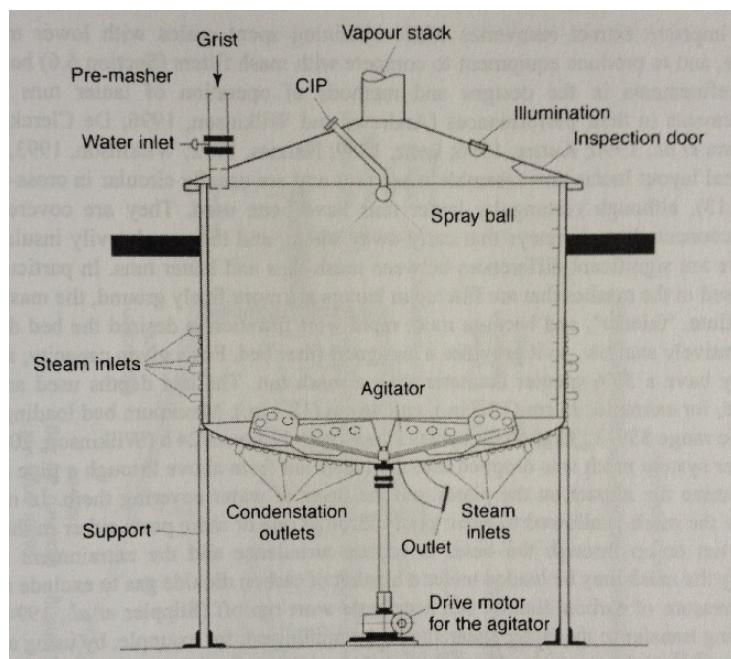


Figura 2.4: Esquema de panela de mistura do mosto.

Fonte: BRIGGS (2011)

2.2.2 Caldeirão de fervura

A fervura é o processo no qual o extrato do mosto é fervido com a adição de lúpulos em um caldeirão de fervura. Em inglês o termo utilizado para este caldeirão (*copper*, que significa cobre) lembra o fato de que inicialmente era comum o emprego de tinas de cobre, dada a

facilidade de moldagem que este metal permite, a alta condutividade térmica e a aparência atrativa dos caldeirões [10]. Este também é usualmente referido como BK, ou *brewing kettle* (caldeirão de fervura). Por muito tempo o estudo do processo de fervura foi negligenciado por ser considerado simples, mas à medida que a questão da economia de energia veio à pauta, estudos oriundos desta necessidade mostraram que o processo é mais complicado do que foi inicialmente considerado [10].

Como existe uma lista de objetivos que devem ser cumpridos durante o processo de fervura, o projeto do equipamento é essencial para que estes sejam atingidos e, em um grau mais avançado, a maior economia de energia possa ser obtida. O primeiro objetivo – que não está necessariamente em ordem de importância – é a evaporação de água e consequente concentração do mosto, com taxas de evaporação inicialmente na faixa de 10% do volume por hora e que foram reduzidas com o desenvolvimento tecnológico das grandes indústrias, já que o custo de evaporação da água é caro em termos de demanda energética. O segundo objetivo importante da fervura é esterilizar o mosto, ou pelo menos matar formas vegetativas de microrganismos – ainda que esporos possam sobreviver ao processo. A terceira função do processo é a evaporação de compostos voláteis indesejados, o que resultou em um desafio tecnológico para diminuir os tempos de fervura e quantidade de água evaporada sem que os compostos voláteis deixassem de ser eliminados efetivamente [10].

Dentre as mudanças que ocorrem no mosto durante a fervura, podem ser notadas criações, adições e transformações de substâncias químicas, como a dispersão de resinas e óleos do lúpulo no mosto e a isomerização de ácidos- α (transformação dos ácidos- α presentes no lúpulo em isômeros que conferem características de amargor à bebida), além da desnaturação e formação de coágulos de proteínas – processo que é favorecido por uma fervura rigorosa e prolongada – que virão a formar o chamado *trub*, resultado da decantação destas proteínas. Este processo de decantação pode ser acelerado por meio da adição de substâncias como gel de sílica ou musgo irlandês, conhecido como *irish moss* [10].

Historicamente o aquecimento dos BKs era feito de forma direta, com a queima de combustíveis sólidos. Embora estes não sejam empregados atualmente, ainda existem BKs que são aquecidos diretamente, por meio da queima de óleos ou gases. Ainda assim, atualmente o maior elemento de aquecimento utilizado é o vapor d'água. Embora estes sistemas sejam mecanicamente mais complexos, o aquecimento a vapor reduz a quantidade de calor aplicada por unidade de área, o que evita a caramelização do mosto [10]. Na figura 2.5 são apresentadas três configurações de BKs, sendo (a) e (b) por meio de aplicação direta de vapor à

superfície do recipiente e (c) utilizando um aquecedor interno, também a vapor.

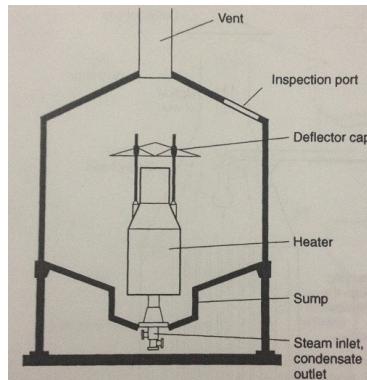
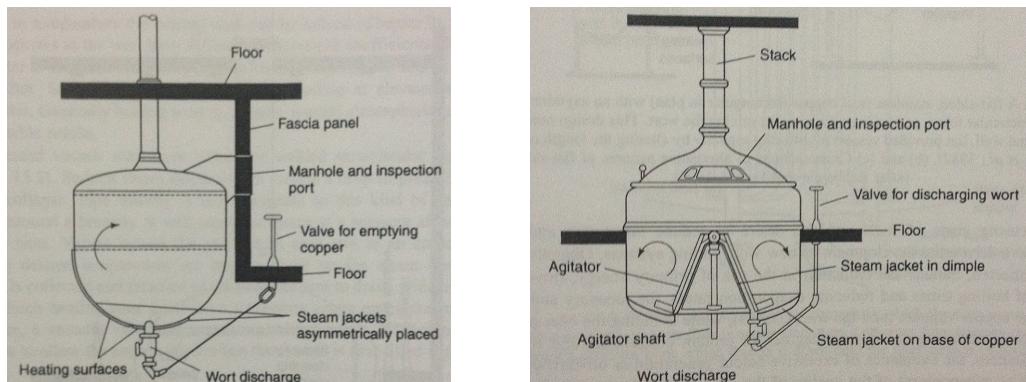


Figura 2.5: Diferentes configurações de caldeirões de fervura.

Fonte: BRIGGS (2011)

O projeto destes sistemas mecânicos são feitos com o objetivo de evitar que o mosto seja caramelizado ou descaracterizado de alguma forma e de modo a economizar energia. Com isso surgiram em função do tempo sistemas pressurizados e de aquecimento externo do líquido, dentre outras configurações diversas. Todos os sistemas modernos e eficientes de fervura são fechados e se aproveitam da pressurização de algum modo para reduzir o custo energético do processo de fervura, que se dá pela redução do tempo de fervura possibilitada pela elevação da temperatura do mosto. Contudo é preciso atentar-se ao fato de que temperaturas muito altas podem afetar negativamente a produção da cerveja e, portanto, valores típicos de temperatura empregados pela indústria chegam a até 104°C, com exceções¹[10]. A tabela 2.1 apresenta alguns sistemas de fervura utilizados na indústria e suas características com relação a temperatura, tempo de fervura e evaporação do mosto. Note-se que mesmo para um tipo específico de fervura, diversas configurações de equipamento podem ser adota-

das.

Tabela 2.1: Características de diferentes sistemas de fervura.

Fonte: adaptado de BRIGGS(2011)

Sistema de aquecimento	Temperatura (°C)	Tempo de fervura (min.)	Evaporação (%)
Panela de "alta performance"	100	120-150	12-16
Aquecedores internos/externos, com contrapressão	102-103	60-80	8
Fervura de baixa pressão	103-104	55-65	6-7
Fervura dinâmica de baixa pressão	103-104	45-50	4,5-5
Fervura de alta temperatura / alta pressão	130-140	2,5-3	6-8
Aquecimento de película fina	100	35-40	4-4,7

Embora o objetivo principal deste trabalho não seja a eficiência energética do sistema, neste parágrafo são apresentadas algumas considerações acerca do tema. Uma vez que o aquecimento de água une todas as partes do processo de produção de cerveja, não é realista acreditar que a economia de energia pode ser aplicada exclusivamente ao processo de fervura, que é o mais custoso em termos energéticos, portanto automação de controles, equipe bem capacitada, boa conservação do edifício, *designs* eficientes, dentre outros fatores, são importantes para a economia de energia. No tocante à fervura em específico, geralmente são utilizados sistemas que recuperam energia de vapor e/ou água quente para reuso nas diversas etapas do processo de produção de cerveja [10]. Um exemplo é o sistema da figura 2.6, que utiliza energia do vapor de aquecimento utilizado na fervura para posterior aquecimento do mosto e pré-aquecimento de outra fervura. Para que isto seja possível, é essencial que a cervejaria realize consecutivas produções, uma vez que a energia é transferida para a produção seguinte.

¹O sistema de alta temperatura / alta pressão (140°C) é pouco utilizado e testes práticos mostram que os resultados em termos de qualidade da cerveja são questionáveis [10]

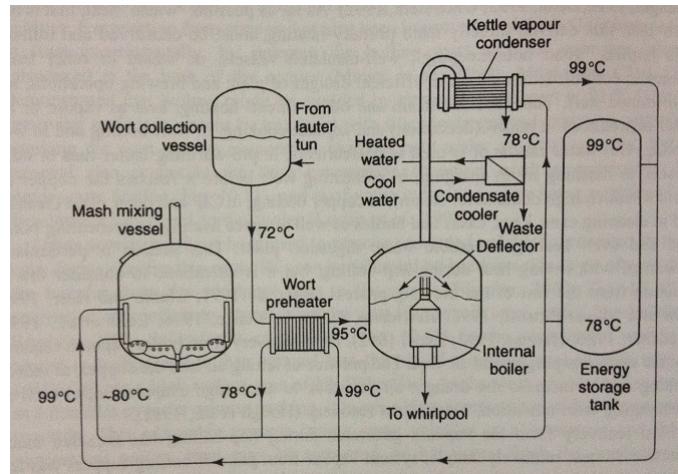


Figura 2.6: Arranjo no qual vapor do caldeirão de fervura é condensado e calor é recuperado e armazenado como água quente em um tanque de gradiente de temperatura.

Fonte: BRIGGS (2011)

2.2.3 Adição de lúpulos

É comum realizar a adição de lúpulos manualmente em pequenas cervejarias, porém deve-se observar que a adição de lúpulos pode ser uma tarefa perigosa, pois o mosto pode subitamente vazar. Também, em adições tardias de lúpulo, ocorre a entrada de ar no BK, o que faz com que sistemas pressurizados não possam ser utilizados sem as devidas considerações e/ou modificações no equipamento [10].

A adição de lúpulos geralmente não é satisfatória quando são utilizadas flores, por isso pastilhas ou extratos são geralmente utilizados. Em grandes cervejarias a maior parte do desafio está no fato de que os lúpulos devem ser armazenados em ambientes com temperatura controlada, que geralmente ficam longe do BK, além de serem armazenados em caixas ou tanques. Além do sistema automático de transporte destes pacotes, em caso de BKs com pressurização, é preciso utilizar sistemas de câmaras de compressão para adicionar os lúpulos à panela [10].

2.2.4 Clarificação e resfriamento do mosto

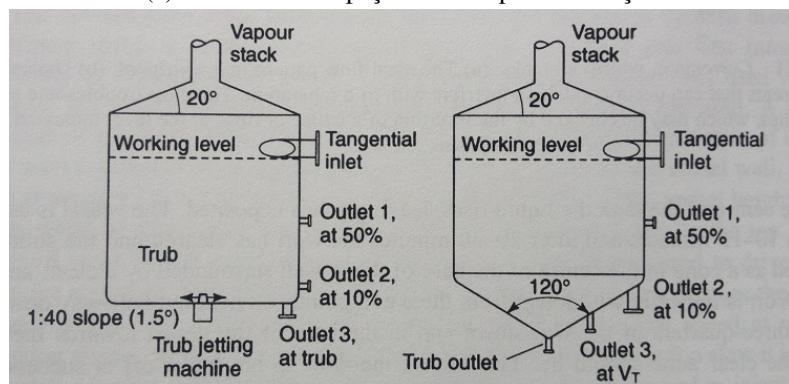
Quando o processo de fervura é finalizado, o mosto deverá estar claro, ou seja, com aparência brilhante. Ainda assim, é preciso remover o *trub*² e os restos de lúpulos utilizados no processo, já que o *trub* contribui com compostos sulfurosos e alcoóis indesejados, além de tornar a cerveja turva – o que pode ou não ser um problema [13]. Diversos equipamentos

²O *trub* descrito neste capítulo é referente ao *hot break*, ou processo de fervura. Neste trabalho não será abordado o *trub* decorrente do *cold break*, ou resfriamento do mosto, que começa a ser formado em temperaturas próximas a 70°C

foram criados com este propósito, seja para mostos com adição de lúpulos em flores, que facilitam a extração dos compostos indesejáveis, quanto para mostos feitos com pastilhas e extratos de lúpulo. Há sistemas chamados *hop back* empregados na filtragem de mostos com lúpulos em flor e que lembram MTs, nos quais o mosto é filtrado por um fundo falso, com elemento filtrador sendo a cama de lúpulos que decanta para o fundo da panela – inclusive há pequenas cervejarias que utilizam a MT para realizar este método de clarificação [10].

Embora existam outros sistemas de clarificação, o empregado mais largamente na indústria atual é a técnica de *whirlpool* (redemoinho, em tradução livre) na qual mosto quente é injetado tangencialmente em um recipiente cilíndrico, a uma velocidade fixa, sendo a altura de injeção do mosto variável em função de diferentes projetos. Sistemas de gravidade geralmente não são suficientes e bombas ou sistemas projetados para utilizar o efeito de termossifão são utilizadas. As correntes de circulação do mosto decorrentes desta injeção tangencial fazem com que o material particulado fique concentrado no fundo e no centro do recipiente, possibilitando o escoamento controlado do líquido e a consequente separação do *trub* [10]. Na figura 2.7 são ilustrados sistemas de separação por decantação (a) e por *whirlpool* (b).

(a) Sistema de serapação do *trub* por decantação



(b) Sistemas de separação do *trub* por *whirlpool*, com fundo plano e arredondado

Figura 2.7: Sistemas de clarificação.

Fonte: BRIGGS (2011)

Após o processo de clarificação deve começar a fermentação e, para que as leveduras possam ser inoculadas no mosto, este deve ser resfriado até a temperatura ideal de operação destes microorganismos, que está tipicamente na faixa 15-22°C para leveduras do tipo *ale* e 6-12°C para leveduras do tipo *lager*. O resfriamento rápido do mosto deve ocorrer, pois assim reações químicas decorrentes da fervura são interrompidas e a chance de contaminação do mosto é reduzida [10].

Para isto, um *cooler* (resfriador ou trocador de calor) é utilizado. Dentre os modelos mais largamente empregados encontra-se o *cooler* vertical, no qual o mosto passa por dentro de um arranjo de tubos finos, enquanto na parte externa destes água fria é circulada em contra-fluxo. Ainda assim, o modelo de *cooler* mais popular é do trocador de calor de placas, devido ao seu tamanho compacto, versatilidade e eficiência. As placas possuem padrões de desenho de tal forma que, quando são compactadas, elas formam canais pelos quais o mosto é passado em contra-fluxo a um agente resfriador (água fria, por exemplo). Estes trocadores de calor são projetados para que o escoamento seja turbulento e, consequentemente, a troca de calor seja mais eficiente. Quando um agente resfriador diferente da água é utilizado, a manutenção do sistema deve ser reforçada, já que vazamentos implicam na contaminação do mosto [10]. A figura apresenta um esquema de *cooler* de placas.

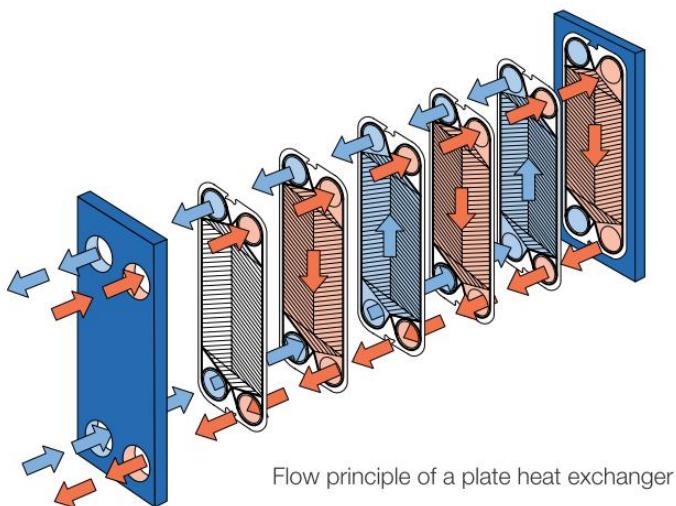


Figura 2.8: Esquema de funcionamento do trocador de calor de placas.
Fonte: Site da Offshore Energy Today³

Por fim, antes de inocular a levedura, é importante que o mosto seja suficientemente

³Disponível em <http://www.offshoreenergytoday.com/wp-content/uploads/2012/03/Alfa-Laval-to-Supply-Plate-Heat-Exchangers-for-Brazilian-Offshore-Platforms.jpg>

aerado ou oxigenado. Sem isso, a levedura não terá um ambiente adequado à sua reprodução e a atenuação do mosto – transformação de açúcares e oxigênio em álcool e gás carbônico – não será completada. Diferenças de solubilidade do mosto e dos tipos de levedura podem influenciar na quantidade de oxigênio necessária para um bom andamento da fermentação, sendo que para obter os níveis adequados, em escala industrial a oxigenação geralmente é forçada [10, 1]. Na produção de cerveja artesanal, a aeração geralmente é feita agitando o mosto resfriado ou com a ajuda de um sistema de aeração de aquário [1].

2.3 BeagleBone Black

BBB é uma placa de sistema embarcado desenvolvida para a comunidade de código aberto (*open-source*), iniciantes e quaisquer pessoas interessadas em um sistema equipado com um processador ARM Cortex-A8 32-bits de baixo custo, seja para teste de conceito ou mesmo uso pessoal/profissional. Embora o sistema tenha sido desenvolvido com um número reduzido de funcionalidades, de modo a proporcionar uma boa experiência de uso, esta placa não é uma plataforma completa de desenvolvimento nem é voltada para o desenvolvimento de um produto específico, mas sim para a experimentação e aprendizado nas áreas de *software* e *hardware* [14]. A figura 2.9 apresenta a placa e seus principais componentes, em conjunto com a tabela 2.2, onde estão contidas as especificações gerais de hardware e a figura 2.10, na qual está contido um diagrama de blocos do sistema, de alto nível de abstração. O diagrama esquemático do sistema pode ser obtido no Anexo I.

É imprescindível atentar para o fato de que a BBB possui diversas revisões de *hardware*, sendo que a versão utilizada neste trabalho é a **revisão C**.

A combinação do processador ARM Cortex-A8 com o projeto de *hardware* da placa possibilita o uso de um sistema operacional embarcado – geralmente uma distribuição Linux – que fornece facilidades de programação, a exemplo de uma IDE acessada pelo navegador da internet chamada Cloud9 em combinação com uma biblioteca de acesso ao hardware em *server-side* Javascript (Node.js), possibilitando a prototipagem rápida de soluções embarcadas de produtos voltados ao mundo real. À medida que o desenvolvedor do sistema se torna mais experiente, é possível desenvolver soluções mais complicadas em diversas linguagens de programação, utilizando-se para isto do sistema operacional Linux embarcado na placa. [15]. Além disso, projetos como o do carro de controle remoto via web, desenvolvido por [15] com o intuito de confirmar as capacidades de placa e seu potencial uso no ensino de

sistemas embarcados, reforçam a escolha da BBB como plataforma para este projeto.

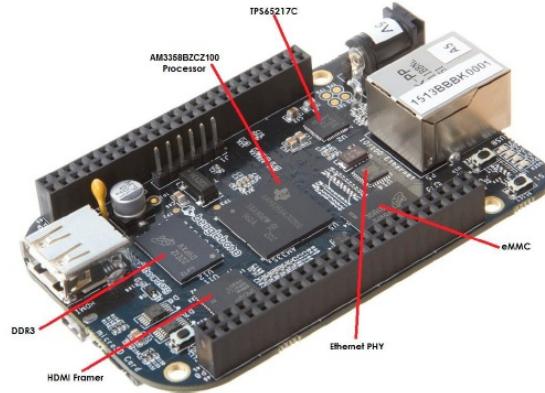


Figura 2.9: BeagleBone Black e principais componentes.

Fonte: COLEY (2014)

- **Sitara AM3358BZCZ100** - SoC (*system on chip*)
- **Micron 512MB DDR3L ou Kingston 512MB DDR3** - memória RAM
- **TPS65217C PMIC** - controlador de alimentação dos diferentes componentes do sistema
- **SMSC Ethernet PHY** - interface física à rede Ethernet
- **Micron eMMC** - memória não volátil MMC de 4GB
- **HDMI Framer** - controlador para uso com display HDMI ou DVI-D

Tabela 2.2: Especificações Gerais da BeagleBone Black.

Fonte: adaptado de COLEY(2014)

	Especificação
Processador	Sitara AM3358BZCZ100, 1GHz, 2000 MIPS
Motor Gráfico	SGX530 3D, 20M polígonos/s
Memória SDRAM	512MB DDR3L 800MHz
Memória Flash	4GB, 8bit MMC embarcada
CI Gerenciador de Alimentação	TPS65217C + regulador adicional (linear)
Suporte a <i>debug</i>	Interface serial, Conector CTI JTAG opcional de 20 pinos
Fonte de Alimentação	mini USB, conector DC ou 5VDC na barra de pinos
PCB	8,64cm x 5,33cm (3,4"x 2,1") - 6 layers
Leds Indicadores	1 para alimentação; 2 para Ethernet; 4 acessíveis ao usuário
USB 2.0 Client	Acesso à USB0 via miniUSB
USB 2.0 Host	Acesso à USB1, soquete tipo A, 500mA LS/FS/HS
Serial	Acesso à UART0 via <i>header</i> de 6 pinos, TTL 3.3V
Ethernet	10/100 RJ45
SD/MMC	Conector microSD, 3,3V
Chaves	Botões push de reset, boot e alimentação
Saída de vídeo	16b HDMI, 1280x1024 (MAX), 1024x768, 1280x720, 1440x900, 1920x1080@24Hz
Audio	Via interface HDMI, estéreo
Conectores de expansão	Alimentação 5V, 3,3V e VDD_ADC(1,8V) 3,3V para todos os sinais de E/S McASP0, SPI, I2C, até 69 pinos de GPIO, LCD, GPMC, MMC1, MMC2, 7 entradas para conversor A/D (1,8V MAX), 4 timers, 4 UART, CAN, PWM via hardware, interrupção XDMA
Peso	39,68g (1,4oz)
Consumo@5VDC	Ocioso - 280mA Carregando página web - 430mA O consumo varia conforme o desempenho

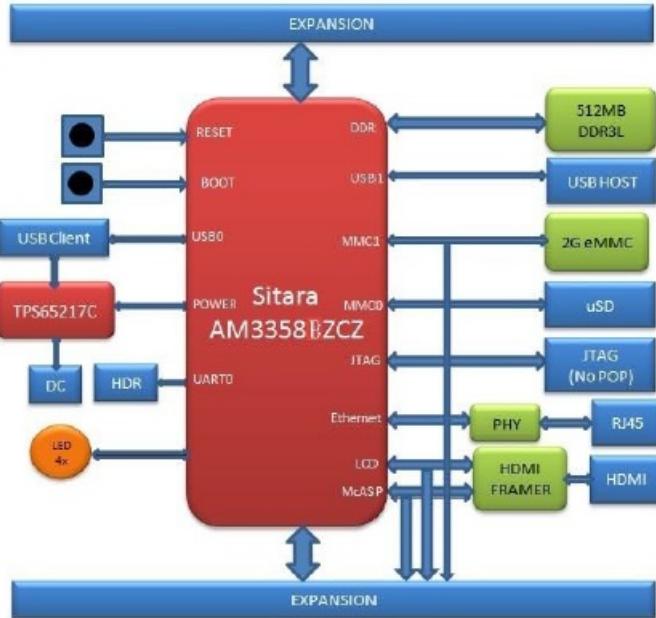


Figura 2.10: Diagrama de blocos de alto nível da BeagleBone Black.

Fonte: COLEY (2014)

Cada pino de GPIO digital da BBB possui até 7 modos diferentes de operação, que são configurados utilizando uma ferramenta do Linux chamada *Device Tree*. A figura 2.11 apresenta a configuração padrão dos pinos, ou seja, quando é usada a distribuição Angstrom do Linux, pré-compilada e que é fornecida com a placa. Com relação aos pinos de GPIO, é importante notar que sua lógica opera com níveis de tensão de 0V e 3,3V para os níveis baixo e alto, respectivamente e portanto aplicar uma tensão negativa ou superior a 3,3V pode danificar permanentemente a placa [16].

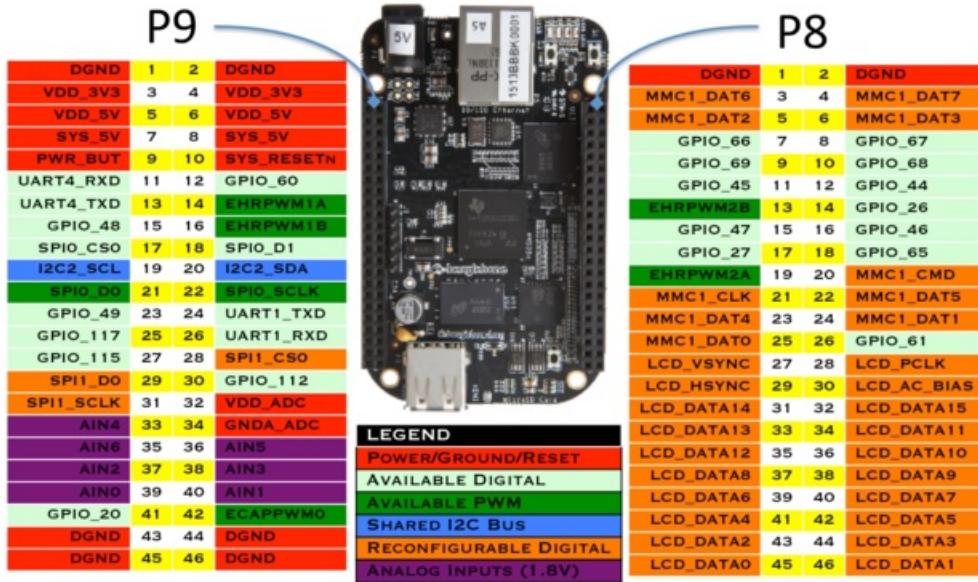


Figura 2.11: Configuração padrão dos pinos da BeagleBone Black.

Fonte: Site oficial da Fundação BeagleBoard.org⁴

2.3.1 Programmable Real-time Unit – PRU-ICSS

A PRU-ICSS – *Programmable Real-Time Unit Subsystem and Industrial Communications Subsystem* (Unidade Programável de Tempo-Real e Subsistema de Comunicação Industrial, em tradução livre) é um subsistema do processador AM3358 que equipa a BBB, que é fabricado pela *Texas Instruments* mas não tem suporte oficial da mesma [14]. Esta unidade consiste de dois núcleos RISC de 32 bits, uma memória compartilhada de 12kB, uma memória de dados e uma memória de programa para cada núcleo, ambas com 8kB de capacidade e periféricos internos – além da possibilidade de acessar todos os eventos, pinos e recursos do SoC AM3358 por meio de uma interface OCP, que confere a este subsistema grande flexibilidade. Cada núcleo da PRU pode operar independentemente ou de maneira sincronizada, dependendo de como o *firmware* para eles é escrito [17]. Na figura é ilustrado o diagrama de blocos deste sistema.

⁴Disponível em <http://beagleboard.org/Support/bone101>

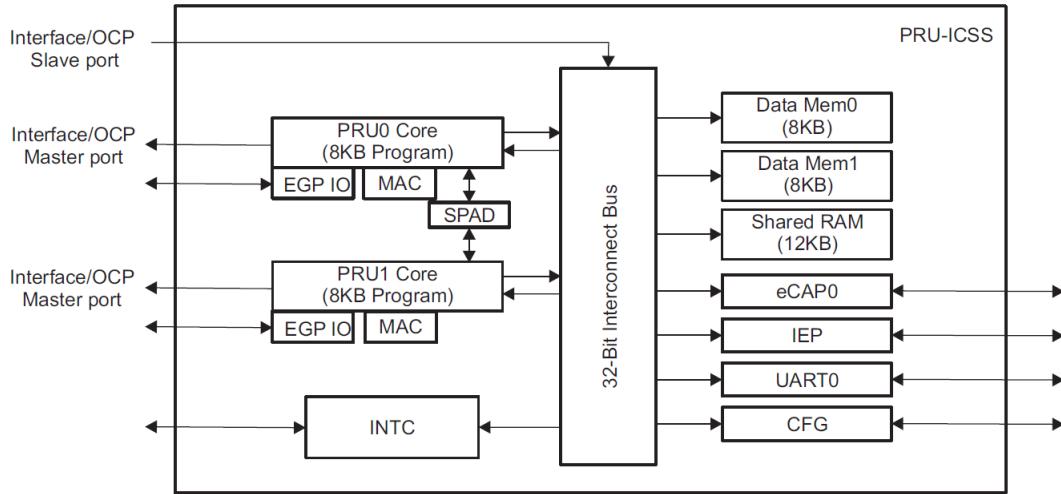


Figura 2.12: Diagrama de blocos do subsistema PRU-ICSS.

Fonte: TEXAS INSTRUMENTS (2015)

Os núcleos da PRU foram otimizados para realizar tarefas com requisitos severos de tempo-real e, portanto, tem um *clock* de 200MHz e um sistema de instruções RISC no qual todas elas são executadas em um ciclo de *clock*, propositalmente sem suporte a *pipelining*. Cada núcleo tem 31 registradores, cuja utilização vai de propósito geral a indexação e controle de GPIO, acessíveis a nível de bit, byte, *halfword* (16 bits), *word* (32 bits) ou por ponteiro [17]. Uma vez que cada instrução pode ser executada em 5ns e o acesso a GPIO é feito por meio de uma única instrução de acesso a registradores, isto significa que é possível obter uma resolução de chaveamento determinístico de no máximo 5ns, com dados obtidos na prática, por [18], de *jitter RMS* de 290ps e estabilidade de frequência na ordem de 10 ppm .

Uma vez que as diversas funcionalidades do módulo PRU são acessadas por meio de mapeamento da memória de dados, é preciso saber a faixa de endereço destas. Aqui deve ser feita a distinção entre acesso à memória local da PRU, que compreende os endereços de 0x0000_0000 a 0x0007_FFFF, e o acesso à memória global, cujos endereços começam em 0x0008_0000. Também é importante notar que as memórias internas dos módulos podem ser acessados com o endereço global de memória, mas isso reduz significativamente o tempo de acesso, já que o sinal é roteado para fora da PRU antes de ser recebido [17]. Na tabela 2.3 é apresentado o mapa de memória local e na tabela 2.4 é apresentado o mapa de memória global. Também há uma tabela de constantes gravada em hardware, com valores de uso recorrente, para economizar instruções de programação e registradores, nos quais estes endereços precisariam ser carregados se não estivessem disponíveis em hardware [17].

Embora cada PRU suporte 30 canais de entrada e 32 canais de saída no modo direto

[17], a BBB roteia somente 25 pinos do SoC ao conector de expansão. Destes, 10 pinos são exclusivos da PRU0 (8 saídas ou 9 entradas) e 15 pinos para a PRU1 (13 saídas ou 14 entradas). Eventualmente, alguns pinos pré-configurados para outras funções devem ser reconfigurados antes que possam ser utilizados [14]. Os pinos da BBB acessíveis pela PRU podem ser obtidos na figura 2.13.

P9			P8		
DGND	1	2	DGND	1	2
VDD_3V3	3	4	VDD_3V3	3	4
VDD_5V	5	6	VDD_5V	5	6
SYS_5V	7	8	SYS_5V	7	8
PWR_BUT	9	10	SYS_RESETN	9	10
GPIO_30	11	12	GPIO_60		
GPIO_31	13	14	GPIO_50		
GPIO_48	15	16	GPIO_51		
GPIO_5	17	18	GPIO_4		
I2C2_SCL	19	20	I2C2_SDA		
GPIO_3	21	22	GPIO_2		
GPIO_49	23	24	GPIO_15		
PRUO_7	25	26	PRU1_16 IN		
PRUO_5	27	28	PRUO_3		
PRUO_1	29	30	PRUO_2		
PRUO_0	31	32	VDD_ADC		
AIN4	33	34	GND_ADC		
AIN6	35	36	AIN5		
AIN2	37	38	AIN3		
AIN0	39	40	AIN1		
PRUO_6	41	42	PRUO_4		
DGND	43	44	DGND		
DGND	45	46	DGND		

Figura 2.13: Pinos da BeagleBone Black acessíveis pela PRU-ICSS.

Fonte: Site oficial da Fundação BeagleBoard.org⁵

⁵Disponível em <http://beagleboard.org/Support/bone101>

Tabela 2.3: Mapa de Memória Local da PRU.
Fonte: adaptado de TEXAS INSTRUMENTS(2014)

Endereço Inicial	PRU0	PRU1
0x0000_0000	Mem. dados da PRU0	Mem. dados da PRU1
0x0000_2000	Mem. dados da PRU1	Mem. dados da PRU0
0x0001_0000	Mem. dados compartilhada	Mem. dados compartilhada
0x0002_0000	INTC	INTC
0x0002_2000	Controle da PRU0	Controle da PRU0
0x0002_2400	Reservado	Reservado
0x0002_4000	Controle da PRU1	Controle da PRU1
0x0002_4400	Reservado	Reservado
0x0002_6000	CFG	CFG
0x0002_8000	UART0	UART0
0x0002_A000	Reservado	Reservado
0x0002_C000	Reservado	Reservado
0x0002_E000	IEP	IEP
0x0003_0000	eCAP0	eCAP0
0x0003_2000	Reservado	Reservado
0x0003_2400	Reservado	Reservado
0x0003_4000	Reservado	Reservado
0x0003_8000	Reservado	Reservado
0x0004_0000	Reservado	Reservado
0x0008_0000	System OCP_HP0	System OCP_HP1

As interfaces de interrupção e de GPIO estão contidas nos registradores R30 e R31: o registrador R31 é utilizado tanto para interface dos pinos configurados como entrada quanto para leitura e/ou geração de eventos de interrupção, enquanto o registrador R30 retorna status ou muda o valor dos pinos configurados como saída, para leitura e escrita, respectivamente. Embora a entrada possa ser configurada nos modos de captura paralela de 16 bits e trem de pulsos de 28 bits, estes modos de operação não serão detalhados, já que seu uso não está contido no escopo deste projeto. O mesmo é válido para a saída, que além do modo direto, pode ser configurada para transmitir um trem de pulsos [17].

Outra característica da PRU é a existência de um IEP - *Industrial Ethernet Peripheral* (Periférico Ethernet Industrial, em tradução livre), composto de um *timer* para Ethernet industrial com 8 eventos de comparação e uma porta digital de E/S. O *timer* Ethernet industrial é simplesmente um temporizador de 32 bits e, portanto, pode ser utilizado para uso geral. É um contador positivo, cujo valor dos incrementos pode ser programado na faixa de 1-16, com compensador de uso opcional. Os 8 registradores comparadores permitem a criação de eventos cada vez que o valor do comparador corresponde ao do temporizador [17].

Tabela 2.4: Mapa de Memória Global da PRU.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014)

Endereço de Offset	PRU-ICSS
0x0000_0000	Mem. dados da PRU0
0x0000_2000	Mem. dados da PRU1
0x0001_0000	Mem. dados compartilhada
0x0002_0000	INTC
0x0002_2000	Controle da PRU0
0x0002_2400	Debug da PRU0
0x0002_4000	Controle da PRU1
0x0002_4400	Debug da PRU1
0x0002_6000	CFG
0x0002_8000	UART0
0x0002_A000	Reservado
0x0002_C000	Reservado
0x0002_E000	IEP
0x0003_0000	eCAP0
0x0003_2000	Reservado
0x0003_2400	Reservado
0x0003_4000	IRAM da PRU0
0x0003_8000	IRAM da PRU1
0x0004_0000	Reservado

No que diz respeito ao desenvolvimento de *firmware* para a PRU, há um conjunto de instruções RISC em assembly, suportado por um conjunto de ferramentas que ajudam a gerar o arquivo binário a ser carregado na memória de programa. Alguns exemplos são o *assembler*, *archiver* e o *linker* [19]. Outra ferramenta de desenvolvimento proporcionada pela Texas Instruments é um compilador C/C++, que facilita a programação em níveis de abstração elevados [20]. A figura 2.14 apresenta um diagrama de blocos que ilustra os passos a serem executados desde compilar o código C/C++ até carregar o arquivo executável na memória de programa da PRU.

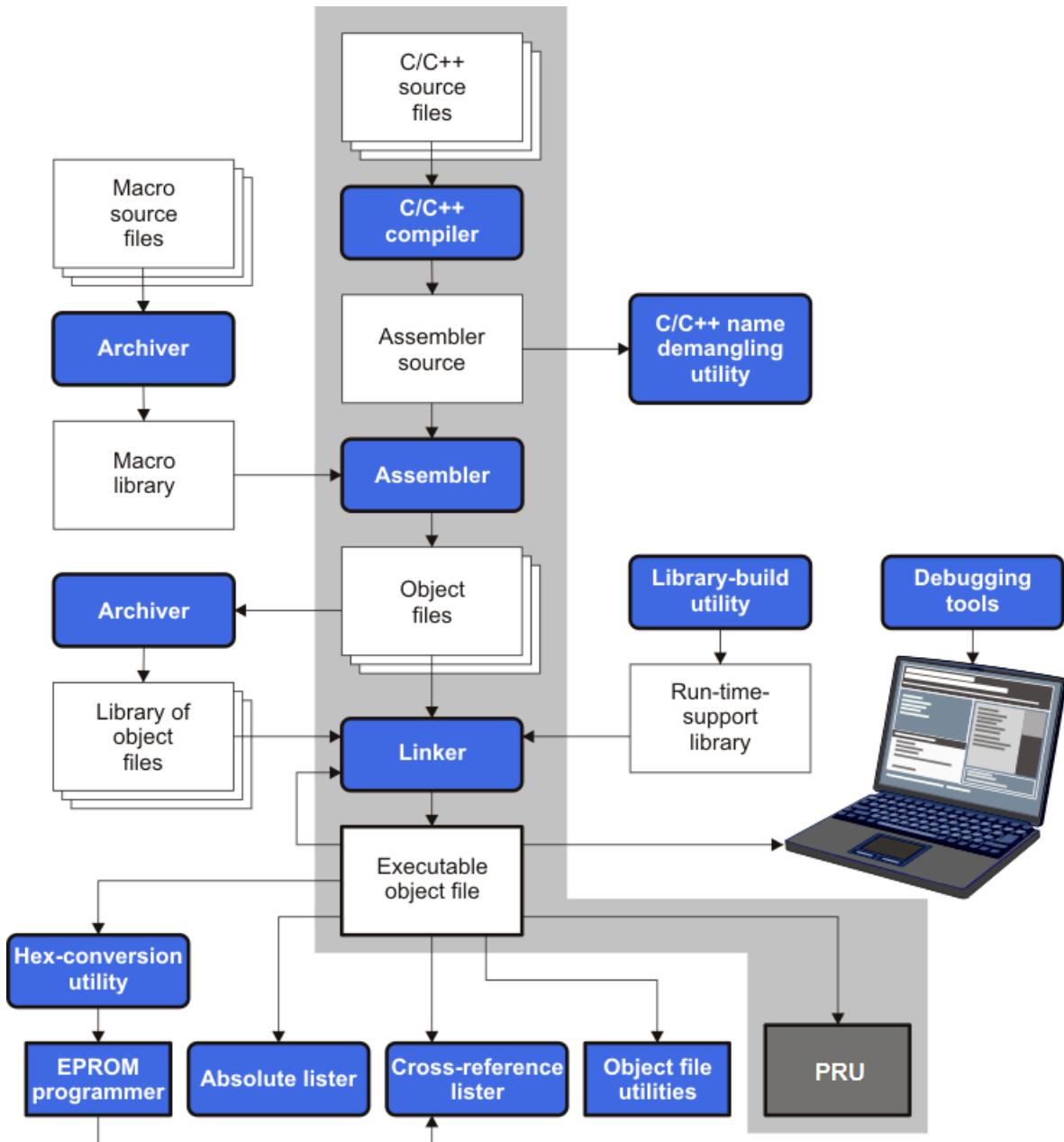


Figura 2.14: Fluxo de desenvolvimento de software da PRU.

Fonte: TEXAS INSTRUMENTS

2.3.2 Device Tree

A *Device Tree* ou *DT* é basicamente uma estrutura de dados utilizada para descrever *hardware*, cujo objetivo é possibilitar que as características de *hardware* de um sistema específico sejam passadas para o SO durante o *boot*, ao invés de estas informações ficarem gravadas no SO [21], o que possibilita que um *driver* do Kernel seja usado em configurações diferentes de *hardware* sem a necessidade de modificações em seu código-fonte. Com relação

aos sistemas equipados com processadores ARM, a necessidade de uso da DT surgiu devido ao grande número de novas placas, que tornou muito difícil manter uma versão do Kernel para cada sistema. Seu objetivo não é modificar o sistema em tempo de execução, porém no caso da BBB, foi criada uma solução chamada *Device Tree Overlay* (ou sobreposição) para executar esta função, facilitando a configuração de GPIO, dentre outras funções, e uso da DT pelo usuário final [22], fatores que são de interesse de aprendizes da área.

Uma DT é uma estrutura em árvore, composta de nós e propriedades. Uma abordagem para entender o seu funcionamento é por meio de um exemplo de sobreposição específico para a BBB. Abaixo é apresentado um exemplo de sobreposição de DT específico da BBB que apresenta os principais conceitos para o uso desta ferramenta:

```

1  /*
2  * Copyright (C) 2013 CircuitCo
3  *
4  * Virtual cape for UART1 on connector pins P9.24 P9.26
5  *
6  * This program is free software; you can redistribute it and/
7  * or modify
8  * it under the terms of the GNU General Public License
9  * version 2 as
10 * published by the Free Software Foundation.
11 */
12 /dts-v1/;
13 /plugin/;
14 /
15 compatible = "ti,beaglebone", "ti,beaglebone-black";
16
17 /* identification */
18 part-number = "BB-UART1";
19 version = "00A0";
20
21 /* state the resources this cape uses */
22 exclusive-use =
23     /* the pin header uses */
24     "P9.24",           /* uart1_txd */
25     "P9.26",           /* uart1_rxd */
26     /* the hardware ip uses */
27     "uart1";
28
29 fragment@0 {
30     target = <&am33xx_pinmux>;
31     __overlay__ {
32         bb_uart1_pins: pinmux_bb_uart1_pins {
33             pinctrl-single,pins = <
34             /* P9.24 uart1_txd.uart1_txd MODE0 OUTPUT (TX) */
35             0x184 0x20
36             /* P9.26 uart1_rxd.uart1_rxd MODE0 INPUT (RX) */
37             0x180 0x20
38             >;
39         };
40     };
41 };
42
43 fragment@1 {
44     target = <&uart2>; /* really uart1 */
45     __overlay__ {
46         status = "okay";
47         pinctrl-names = "default";
48         pinctrl-0 = <&bb_uart1_pins>;
49     };
50 };
51 };

```

Neste código, na linha 14 é definida a plataforma à qual esta DT se refere. Com isso, ficam declaradas as funcionalidades de *hardware* do sistema. Na sequência, a identificação

mostra quais sobreposições de DT estão carregadas e o nome do arquivo compilado deve ser igual ao nome nela atribuído; a versão deve ser 00A0 para a BBB. Entre as linhas 20 e 26 são listados os recursos de *hardware* utilizados por esta sobreposição, que impedem que outras sobreposições que usem os mesmos recursos sejam carregadas; no caso deste exemplo, os pinos referentes à UART1, assim como o próprio dispositivo UART1 são utilizados. A seguir, os fragmentos descrevem qual dispositivo terá suas configurações sobrepostas. No fragmento 0, é o multiplexador dos pinos da BBB, compatível com o *driver pinctrl-single* - os dois valores hexadecimais utilizados para cada pino são o *offset* do pino com relação ao seu registrador e a configuração do pino. Já no fragmento 1 é configurada a interface UART1. Por enquanto, serão introduzidas somente as questões referentes à configuração do multiplexador, uma vez que seu conhecimento é necessário para o uso dos pinos no modo GPIO.

Para determinar o valor hexadecimal de *offset* do pino ao qual se deseja aplicar uma configuração, primeiro é preciso obter seu nome a partir do seu número na barra de pinos, conforme ilustrado na figura 2.11, consultando as tabelas 2.6 e 2.7, nas quais o nome do pino é referente ao **modo 0**, que também é o nome do pino no manual do SoC [14, 17]. A seguir, obtém-se o valor hexadecimal do registrador referente ao pino, na tabela II.1 presente no anexo II, adaptada do manual do SoC, e subtrai-se 0x800 de seu valor original.

Com relação à configuração do pino, este aceita os parâmetros de ajuste de *slew-rate*, ativação do *buffer* de entrada, ajuste e ativação do *pull-up/pull-down* interno e seleção do modo de operação do pino. Note-se que, com relação ao buffer de entrada, ao menos um blog da internet refere-se a este bit como sendo um bit de seleção para entrada **ou** saída [23], porém o manual indica que quando o bit está setado, o pino pode ser usado tanto como entrada quanto saída [17]. A tabela 2.5 apresenta os campos do registrador de controle de E/S, assim como a descrição de cada campo e os valores aceitos.

Tabela 2.5: Descrição dos campos do registrador de controle de *pads*.

Fonte: adaptado de TEXAS INSTRUMENTS(2014)

Bit	Campo	Valor	Descrição
31-7	Reservado		Reservado. Leitura retorna 0
6	SLEWCTRL	0 1	Seleciona entre <i>slew-rate</i> rápido ou lento 0 - rápido 1 - lento
5	RXACTIVE	0 1	Ativa modo de entrada para o pino 0 - somente saída 1 - Receptor ativado. Entrada ou saída.
4	PULLTYPESEL	0 1	Seleção de pull-up/pull-down 0 - pull-down 1 - pull-up
3	PULLUDEN	0 1	Habilita pull-up/pull-down 0 - habilitado 1 - desabilitado
2-0	MUXMODE	0-7	Seleção de funcionalidade do pino multiplexado

Tabela 2.6: Lista de pinos do *header P8* e seus respectivos modos de funcionamento
fonte: COLEY

REF: BBONEBLK_SRM BeagleBone Black System Reference Manual Rev C.1							
Table 12. Expansion Header P8 Pinout							
PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4
1,2	R9	GPIO1_6	gpmc.ad6	mmc1.dat6		GND	
3	GPIO1_13	gpmc.ad7					
4	GPIO1_2	gpmc.ad8					
5	GPIO1_2	gpmc.dat2	mmc1.dat2				
6	T8	GPIO1_3	gpmc.ad9	mmc1.dat3			
7	R7	TIMER4	gpmc.ad10	mmc1.dat4			
8	T7	TIMER5	gpmc.ad11	mmc1.dat5			
9	T6	TIMER5	gpmc.be0h_ce0	timers5			
10	U6	TIMER6	gpmc.wen	timers6			
11	R12	GPIO1_13	gpmc.ad13	lcd.datas16	mmc2.dat1	eQEP2B_in	prt1.psu.psu.r30_15
12	U12	GPIO1_2	gpmc.ad14	lcd.datas17	mmc2.dat2	mcasp0_axr2	prt1.psu.psu.r30_14
13	T10	EHRPWM2B	gpmc.ad9	lcd.datas22	mmc1.dat1	mmc2.dat5	gpio113
11	T11	GPIO1_26	gpmc.ad10	lcd.datas21	mmc1.dat2	mmc2.dat6	gpio112
13	U13	GPIO1_26	gpmc.ad15	lcd.datas21	mmc1.dat3	eH PWM2_trigzone_in	gpio112
16	U14	GPIO1_14	gpmc.ad16	lcd.datas17	mmc1.dat4	eQEP2_strobe	prt1.psu.psu.r30_13
17	V9	GPIO1_27	gpmc.ad11	lcd.datas20	mmc1.dat3	mmc2.dat7	prt1.psu.psu.r30_14
18	V12	GPIO2_2	gpmc.clk_mux0	lcd.memory_clk	mmc2.clk	mcasp0_fsr	gpio111
19	U10	UART3_CS1N	gpmc.ad17	lcd.datas20	mmc2.dat4	eH PWM2A	gpio112
20	V9	GPIO1_31	gpmc.cs1	gpmc_clk	mmc1.clk		
21	U9	GPIO1_30	gpmc.ad5	mmc1.dat5			
22	V9	GPIO1_5	gpmc.ad6	mmc1.dat6			
23	U9	GPIO1_30	gpmc.ad7	mmc1.dat7			
24	V7	GPIO1_1	gpmc.ad1	mmc1.dat8			
25	U7	GPIO1_0	gpmc.ad0	mmc1.dat0			
26	V6	GPIO1_29	gpmc.cs0	gpmc_a0			
27	V6	GPIO1_29	gpmc_a0	gpmc_a0			
28	V8	GPIO2_24	lcd.pick	gpmc_a0			
29	R5	GPIO2_23	lcd.hsync	gpmc_a0			
30	R6	GPIO2_23	lcd.ac_bias_en	gpmc_a11			
31	V1	UART3_CS1N	gpmc.ad18	eQEP1_index	mcasp0_axr1	uart5_rxd	uart5_ctsn
32	T5	UART3_CS1N	gpmc_be0h	lcd.datas15	mcasp0_axr0	uart5_txm	uart5_ctsn
33	V3	UART4_CS1N	gpmc_a19	eQEP1_strobe	mcasp0_axr0x2	mcasp0_axr3	gpio0111
34	U3	UART4_CS1N	gpmc_be1h	eQEP1_in	mcasp0_txr	mcasp0_axr3	gpio0109
35	V2	UART3_CS1N	gpmc_a15	lcd.datas13	mcasp0_axr0	uart3_txm	uart3_ctsn
36	U3	UART3_CS1N	gpmc_be1h	eQEP2_index	mcasp0_axr0	mcasp0_axr2	uart3_ctsn
37	U1	UART3_RXD	gpmc_a16	lcd.datas10	mcasp0_axr1	uart3_txm	uart2_ctsn
38	U1	UART3_RXD	gpmc_be1h	eH PWM1_index	mcasp0_axr0	uart3_txm	uart2_ctsn
39	T3	GPIO2_24	gpmc_a0	lcd.datas6	eQEP2_index	uart5_txm	uart2_ctsn
40	T4	GPIO2_13	gpmc_a7	lcd.datas7	eQEP2_strobe	prt1.edio_data_out7	prt1.psu.psu.r30_6
41	T1	GPIO2_13	gpmc_a4	lcd.datas4	eQEP2A_in	prt1.psu.psu.r30_7	prt1.psu.psu.r31_7
42	T2	GPIO2_10	gpmc_a5	lcd.datas5	eQEP2B_in	prt1.psu.psu.r30_4	prt1.psu.psu.r31_4
43	R3	GPIO2_8	gpmc_a2	lcd.datas2	mmc2.clk	prt1.psu.psu.r30_2	prt1.psu.psu.r31_2
44	R4	GPIO2_9	gpmc_a3	lcd.datas3	eH PWM0_index	prt1.psu.psu.r30_3	prt1.psu.psu.r31_3
45	R1	GPIO2_6	gpmc_a0	lcd.datas0	eH PWM2A	prt1.psu.psu.r30_0	prt1.psu.psu.r31_0
46	R2	GPIO2_7	gpmc_a1	lcd.datas1	eH PWM2B	prt1.psu.psu.r30_1	prt1.psu.psu.r31_1

Tabela 2.7: Lista de pinos do *header P9* e seus respectivos modos de funcionamento
fonte: COLEY

REF: BBONEBLK_SRM BeagleBone Black System Reference Manual Rev C.1							
Table 13. Expansion Header P9 Pinout							
PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4
10	A10	UART4_RXD	gpmc.w0t0	mcasp0_crs	mmc1.cmd	mmc1.vdd	mmc1.vdd
11	T17	UART4_RXD	gpmc_be1n	mcasp0_dat0	mmc1.dat0	mmc1.vdd	mmc1.vdd
12	U18	GPIO1_28	gpmc_be1n	mcasp0_dat1	mmc1.dat1	mmc2.vdd	mmc2.vdd
13	U17	UART4_RXD	gpmc_wpn	mcasp0_dat2	mmc1.dat2	mmc2.vdd	mmc2.vdd
14	U14	EHRPWM1B	gpmc_a10	mcasp0_dat3	mmc1.dat3	mmc2.vdd	mmc2.vdd
15	B13	GPIO1_16	gpmc_a11	mcasp0_dat4	mmc1.dat4	mmc2.vdd	mmc2.vdd
16	T14	EHRPWM1B	gpmc_a3	mcasp0_dat5	mmc1.dat5	mmc2.vdd	mmc2.vdd
17	A16	D21_SCL	gpmc_be1h	mcasp0_dat6	mmc1.dat6	mmc2.vdd	mmc2.vdd
18	U19	D21_SDA	gpmc_be1h	mcasp0_dat7	mmc1.dat7	mmc2.vdd	mmc2.vdd
19	D17	D22_SDA	uart1_rxn	mcasp0_rxn	I2C2.SCL	I2C2.SDA	I2C2.SDA
20	D18	D22_SDA	uart1_rxn	mcasp0_rxn	I2C2.SCL	I2C2.SDA	I2C2.SDA
21	B17	UART2_RXD	sp1_d0	uart2_txd	I2C2.SCL	eH PWM0	EMU3_mu1
22	U19	UART2_RXD	sp1_d1	uart2_txd	I2C2.SCL	eH PWM0	EMU3_mu1
23	V14	GPIO1_17	gpmc_a1	gpmc_rxn	mcasp0_dat0	gpmc_a17	eH PWM0_sync
24	D15	UART1_RXD	uart1_rxn	mcasp0_dat1	mcasp0_dat1	mcasp0_axr1	mcasp0_axr1
25	A14	GPIO3_20	mcasp0_be1h	mcasp0_be1h	mcasp0_axr1	mcasp0_axr1	mcasp0_axr1
26	U20	U20_PWDMD	mcasp0_be1h	mcasp0_be1h	mcasp0_axr2	mcasp0_axr2	mcasp0_axr2
27	C13	GPIO1_19	mcasp0_be1h	eQEP0_be1h	mcasp0_axr3	mcasp0_axr3	mcasp0_axr3
28	C12	SP1_CS0	mcasp0_be1h	eH PWM0_index	mcasp0_axr3	mcasp0_axr3	mcasp0_axr3
29	B13	SP1_D0	mcasp0_be1h	eH PWM0	sp1_d0	mcasp0_axr1	mcasp0_axr1
30	D19	SP1_D1	mcasp0_be1h	eH PWM0	sp1_d1	mcasp0_axr1	mcasp0_axr1
31	A13	SP1_CS1	mcasp0_be1h	eH PWM0	sp1_d0	mcasp0_axr1	mcasp0_axr1
32					VADC		
33		C8			AIN4		
34					AIN5		
35		A8			AIN6		
36		B8			AIN5		
37		B9			AIN5		
38		A7			AIN5		
39		B6			AIN0		
40		C7			AIN1		
41#	D14	GLITCH2	xdm1_slient_pst	ICIN	ICIN	timer1_mux1	EMU1_mu0
D13	GPIO1_20	xdm1_slient_pst	ICIN	mcasp0_axr1	mcasp0_axr1	mcasp0_axr1	mcasp0_axr1
C18	GPIO1_7	eCAP1_in_PWM0_out	uart3_txd	sp1_d1	mcasp1_in_PWM2_out	prt1.psu.psu.r30_3	prt1.psu.psu.r31_3
42#P	B12	GPIO1_18	mcasp0_addr	eQEP0A_in	mcasp0_axr2	mcasp0_axr2	mcasp0_axr2
43-46					GND		

*GPIO3_21 is also the 24.576MHz clock input to the processor to enable HDMI audio. To use this pin the oscillator must be disabled.

2.4 Programação em Node.js (Javascript)

Node.js é um ambiente em tempo de execução recente, criado em 2009 [24], que permite a execução no servidor de programas escritos em Javascript. Para tal, este ambiente utiliza um motor implementado pelo Google para seu navegador Chrome — o V8, que compila o Javascript em código de máquina e não *bytecode* ou interpretação *on-the-fly*, o que torna a execução de Javascript extremamente rápida [25], embora seja importante lembrar que o *overhead* inerente de lingaugens de tipagem dinâmica custa o tempo de processamento de resolução do tipo de variável utilizada [26].

Node, devido à natureza assíncrona de I/O do Javascript, implementa um suporte nativo à programação orientada a eventos, ainda que seja executado em uma única *thread*, o que o torna um diferencial com relação às outras lingaugens de programação mais populares, que exigem do desenvolvedor a implementação de processos de *multithreading* para atingir paralelismo de processamento [27]. Ainda, o estilo de programação em Node é orientado a *callbacks*, que nada mais é do que passar como argumento uma função que será executada assim que um evento acabar.

Historicamente, o desenvolvimento de *multithreading* surgiu da necessidade de serviços de rede que possibilitassem múltiplas comunicações em paralelo, o que é conhecido como múltiplas fontes de I/O e assim será referenciado neste trabalho — isto ocorreu devido ao fato de linguagens tradicionais de programação, mais antigas, focarem na implementação de aplicações operadas por um único ser humano enquanto, com o desenvolvimento da internet, esta realidade mudou drasticamente, exigindo processamento de I/O massivo [28]. Em processadores com diversos núcleos, isso significa processamento em paralelo, enquanto em processadores com um único núcleo é implementada uma multiplexação temporal que permite tal técnica. O fato mais notável das implementações de *multithreading* é a complexidade do código, que não é trivial e geralmente dá origem a códigos mal estruturados e repletos de variáveis globais, difíceis de manter [27, 28].

Javascript, que nasceu como uma linguagem para execução no cliente, tornou-se a espinha dorsal de aplicações HTML modernas e, recentemente, ficou muito popular dentre aplicações no servidor. Especificamente em Node, o comportamento assíncrono é a regra geral, devido à orientação a eventos. Isto leva programadores acostumados a linguagens síncronas como C a terem dificuldades iniciais de programação: o uso de *callbacks* de evento é fundamental para estruturar a ordem com que certas tarefas são executadas, sempre após o final do evento

pai [28]. É importante salientar que este tipo de implementação ocorre de forma transparente ao usuário mas, na verdade, nada mais é do que um *loop* principal cujo objetivo é cuidar de todas as chamadas a funções. Outro ponto notável relacionado ao Node é que, para executar aplicações em múltiplos núcleos, é preciso executar múltiplas instâncias, o que pode ser gerenciado com bibliotecas de suporte [27, 28]. Note-se também que os termos *orientado a eventos* e *assíncrono* são equivalentes [28].

Para ilustrar o comportamento assíncrono do Node, no código a seguir não há garantia alguma de que a função executada na segunda linha receba um parâmetro diferente de nulo (Null) [28]:

```
1 var botaoStatus = leBotao();
2 imprimeValorBotao(botaoStatus); //nao ha garantia de que
   leBotao ja acabou de ser executada
```

A maneira correta para executar este tipo de código seria usando uma função de callback, como descrito no trecho de código abaixo, embora esta não seja a única maneira de fazê-lo. Observe-se que a função é declarada como um parâmetro de *leBotao* e não tem nome — é uma função anônima. Embora isto seja prático, do ponto de vista de manutenção de código e *debug* é uma prática que deve ser evitada, uma vez que pode originar uma situação de dificuldade de manutenção de código conhecida como *callback hell*, ou *inferno de callbacks* [29, 30].

```
1 leBotao(function(botaoStatus) {
2     imprimeValorBotao(botaoStatus); //so acontece depois que o
      botao eh lido
3 }) ;
```

2.4.1 Node Package Manager - NPM

O *Node Package Manager*, ou simplesmente NPM, é não somente um gerenciador de pacotes como também é um repositório de pacotes de terceiros e um padrão para definição de dependências. Os pacotes ficam em um registro público e podem ser gerenciados por uma ferramenta própria por linha de comando. O uso do NPM não é obrigatório, mas à medida que aplicações mais complexas são desenvolvidas é quase mandatório seu uso, pois assim é possível usar módulos prontos para executar diversas tarefas com facilidade. Uma vantagem do NPM é que os módulos podem ser instalados localmente, restringindo-os ao diretório do projeto e, portanto, garantindo certo nível de segurança. [28]. Abaixo é apresentado um exemplo no qual o pacote fictício *meuPacote* é instalado para a versão mais recente adicionada ao repositório do NPM:

```
1 npm install meuPacote@latest
```

Outro exemplo de uso do NPM é a definição de dependências de um projeto e posterior possibilidade de instalação de todas elas de uma só vez. Para isto, um arquivo no formato JSON descreve estas dependências e outras informações do projeto:

```
1 {
2   "name" : "meuPacote",
3   "version" : "1.0.0",
4   "dependencies" : {
5     "debug" : "0.3.x",
6     "nano" : "*",
7     "request" : ">0.2.0"
8   }
9 }
```

E a instalação das dependências se dá da seguinte maneira, assumindo que o presente diretório de trabalho é o mesmo do projeto:

```
1 npm install
```

2.4.2 MEAN Stack

MEAN é uma pilha de desenvolvimento, de código aberto, para aplicações web e provê um conjunto de quatro ferramentas que, juntas, são a base necessária para a criação de aplicações tanto de servidor quanto do cliente. Um exemplo de pilha de desenvolvimento para aplicações web muito disseminado é a pilha LAMP. Voltando à MEAN, cada uma das 4 letras deste acrônimo representa uma das ferramentas [31]:

- **MongoDB** - banco de dados orientado a objetos
- **Express.js** - framework para criação de servidor web e roteamento
- **Angular.js** - framework para aplicações web
- **Node.js** - base da aplicação do servidor

Note-se que toda a pilha é baseada na linguagem Javascript, o que permite ao desenvolvedor de uma solução completa uma curva de aprendizado mais rápida, pois menos tempo é empregado aprendendo a sintaxe e o paradigma de programação e mais tempo dedicado à funcionalidade da aplicação em si [31].

2.4.3 Servidor Express.js

Express é um framework de *middleware* para implementação de aplicações web no servidor, incluindo mas não limitado a roteamento e operações HTTP, a exemplo dos métodos GET

e POST. Embora o Node apresente um módulo dedicado a HTTP, a proposta do Express é simplificar seu uso e evitar que o mesmo trabalho seja realizado múltiplas vezes e por várias pessoas diferentes [31].

Dentre as facilidades proporcionadas pelo node está a fácil implementação de diferentes respostas para diferentes tipos de requisições baseadas no método HTTP e a renderização dinâmica de documentos HTML [28]. O tratamento de requisições baseadas no método HTTP, possibilitado pelo Express, também é conhecido como RESTful API, que é uma API baseada no REST (*Representational State Transfer*). Este por sua vez é um modelo para serviços web que implementa as operações HTTP POST, GET, PUT e DELETE, que são mapeadas como operações básicas de banco de dados: criar, ler, atualizar e deletar [31].

2.5 Circuitos de Interface

No âmbito deste trabalho, circuitos de interface são circuitos eletrônicos que possibilitam a interação entre a plataforma BeagleBone Black e os sensores e atuadores do sistema mecânico. O projeto adequado destes circuitos é essencial não somente para que o projeto funcione corretamente, mas também para evitar que componentes do sistema, a exemplo da BBB, sejam danificados. Nesta seção, será abordada a teoria essencial para o projeto destes, desde conceitos relacionados à teoria de sinais, passando por alguns elementos básicos da eletrônica e finalizando com as características dos sensores e atuadores empregados.

2.5.1 Sinais

Sinais são funções matemáticas que guardam informações acerca de fenômenos físicos, a exemplo da temperatura de um elemento em função do tempo ou a representação do som de um instrumento musical. Na figura 2.15 é apresentado um período de um sinal senoidal puro. Uma vez que, para obter informações a partir de um sinal, é preciso processá-lo, as mais diversas grandezas físicas devem ser convertidas para grandezas manipuláveis pelo sistema de processamento — que no caso de circuitos eletrônicos é comumente a tensão elétrica, embora outras grandezas também sejam utilizadas [32]. Para que a conversão de diferentes grandezas físicas em sinais elétricos seja possível, são empregados dispositivos sensores e transdutores.

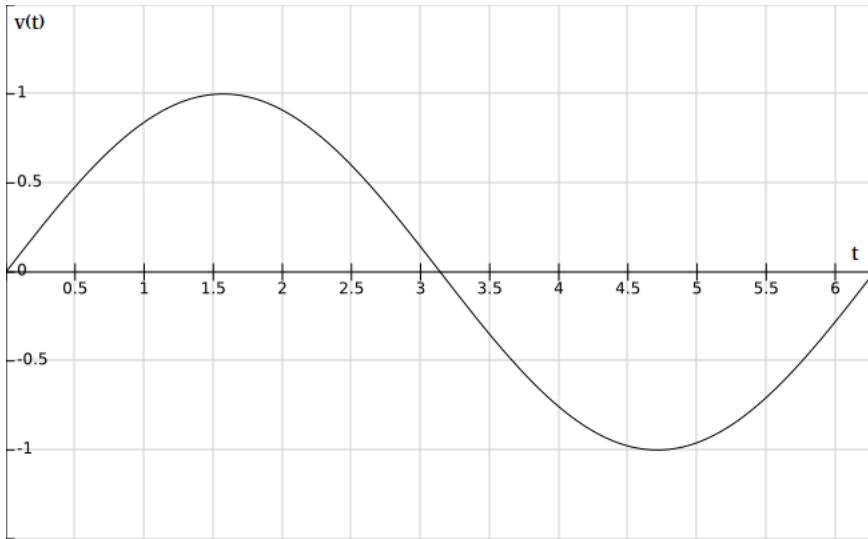


Figura 2.15: Sinal senoidal em função do tempo

2.5.2 Dispositivos Semicondutores

Diodos são os elementos eletrônicos mais simples, constituídos de dois terminais que estão conectados a uma junção PN, ou seja, a uma junção formada pelo contato entre dois cristais semicondutores dopados com impurezas de polaridades opostas [33]. O efeito prático desta junção é a capacidade de retificação: quando uma DDP positiva é aplicada aos terminais do diodo conforme exposto na figura 2.16, a corrente elétrica flui pelos terminais do dispositivo e diz-se que o diodo está polarizado diretamente; caso a DDP aplicada seja invertida, a corrente elétrica deixa de fluir pelo dispositivo [32]. Este pode ser considerado o diodo ideal, que é o modelo mais simplificado deste elemento e não leva em consideração nenhuma característica real do mesmo.

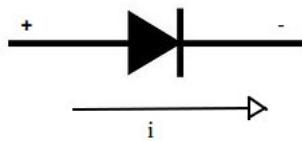


Figura 2.16: Representação esquemática do diodo polarizado diretamente

O diodo ideal é descrito pela equação 2.1, na qual I_s é a corrente de saturação reversa, $k = 11600/\eta$, com η igual a 1 para o germânio e 2 para o silício e T_k é a temperatura em kelvin. Muitas vezes, esta curva é aproximada por um modelo conhecido como *circuito equivalente linear*, composto de um diodo em série com uma fonte de tensão e um resistor, que definem o limiar de condução e o nível de resistência do dispositivo quando está conduzindo [34]. A figura 2.17 ilustra um exemplo de definição do circuito equivalente a partir da re-

presentação real. Outra modelagem é conhecida como *modelo para pequenos sinais*, no qual é fixado um ponto de operação em torno do qual a excursão do sinal aplicado ao circuito é pequena, conhecido como ponto de polarização ou **ponto quiescente**. Neste modelo, a região exponencial é aproximada linearmente por um resistor cujo valor é o inverso da inclinação da reta tangente ao ponto de polarização [32]. Cabe salientar que o avanço das ferramentas de computação tornaram possível o cálculo dos mais diversos circuitos eletrônicos de forma rápida e exata, relegando portanto os modelos simplificados a ferramentas de ensino ou para esboço de circuitos simples [34], desde que os modelos empregados nas simulações sejam fiéis ao comportamento real dos componentes [32].

$$I_d = I_s \epsilon^{kV_d/T_k} - I_s \quad (2.1)$$

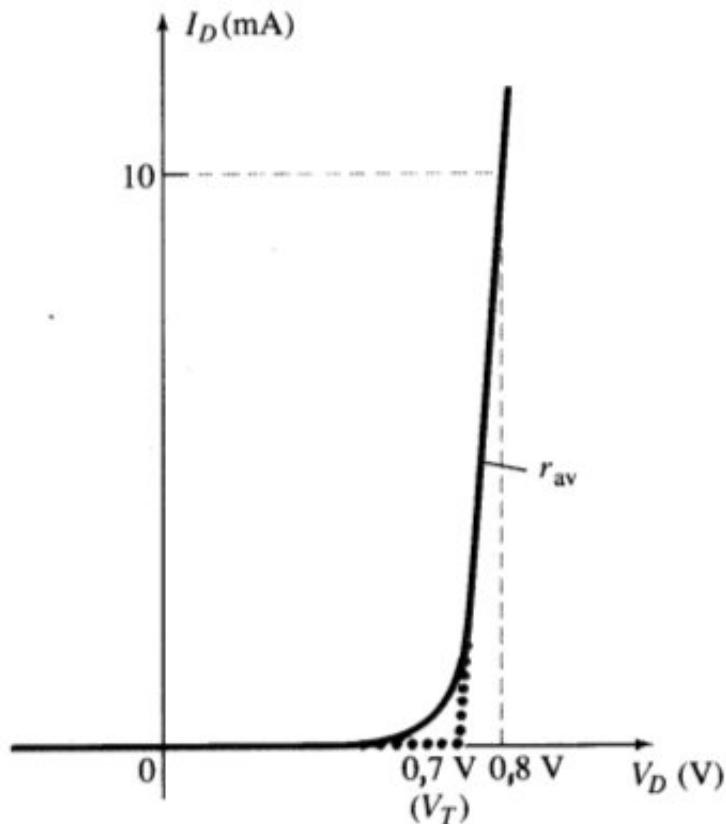


Figura 2.17: Definição do circuito equivalente linear, usando-se segmentos de reta para aproximar a curva característica
Fonte: BOYLESTAD (2011)

A partir do conhecimento obtido com a introdução ao diodo e sua junção PN, será agora abordado o transistor bipolar de junção, também conhecido com BJT. De forma análoga ao

diodo, este dispositivo é constituído da junção de três cristais semicondutores dopados, podendo ser uma junção do tipo NPN ou PNP: ambos funcionam de forma complementar, sendo que o tipo NPN conduz majoritariamente elétrons, enquanto o tipo PNP conduz majoritariamente lacunas. Na figura 2.18 são apresentados os sentidos de corrente e os símbolos dos transistores PNP e NPN e, aplicando a lei das correntes de Kirchhoff, obtém-se a relação entre as correntes de emissor, coletores e base do transistor, descritas pela equação 2.2 [34]. Note-se que é usado o sentido real da corrente e não o convencional, comumente adotado em cálculos.

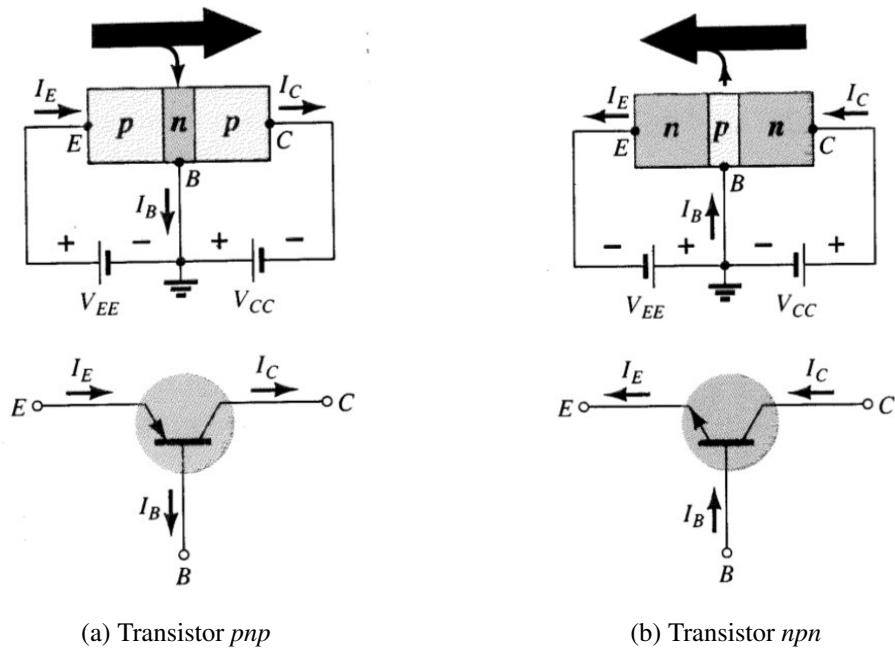


Figura 2.18: Notações e símbolos utilizados para a configuração base-comum.

Fonte: BOYLESTAD (2011)

$$I_e = I_c + I_b \quad (2.2)$$

Para finalizar a introdução ao transistor bipolar, é apresentada uma relação simplificada entre as correntes de coletores I_c e de base I_b na equação 2.3, uma vez que esta é uma relação que permite a realização de cálculos rápidos para esboçar circuitos com transistores BJT. Cabe salientar que a validade desta relação se dá somente quando o transistor está corretamente polarizado e na região de condução. Outro fato que deve ser levado em conta é que o valor de β depende das características de construção de cada BJT e é um valor cujo desvio padrão é elevado mesmo se comparados dois transistores reais do mesmo modelo [34]. Ainda que o conteúdo relativo a transistores BJT seja muito mais extenso e detalhado, cabe ao leitor

consultar as referências bibliográficas [34] e [32] para mais detalhes acerca do tema.

$$I_c = \beta \cdot I_b \quad (2.3)$$

Existe outro tipo de transistor conhecido como transistor de efeito de campo ou FET. Embora este seja um dispositivo semelhante ao BJT no que diz respeito à aplicação, também existem inúmeras diferenças, dentre as quais a mais notável é o fato de que o BJT é controlado a corrente, enquanto o FET é controlado a tensão. Outra diferença importante é o fato de que transistores FET podem ser de canal *n* ou canal *p*, ou seja, são transistores unipolares — que conduzem somente elétrons ou somente lacunas, respectivamente. Sendo um dispositivo controlado a tensão, a impedância de entrada é altíssima, muitas vezes considerada infinita em casos práticos [34]. Uma subclasse destes dispositivos muito popular é o MOSFET - popularmente empregado como chave eletrônica na área de circuitos integrados, mas também adotado para chaveamento de dispositivos de alta potência em função da sua baixa impedância quando ligado [32]. O símbolo dos transistores JFET e MOSFET são apresentados na figura 2.19. Os terminais do dispositivo são nomeados *fonte* (*source*), *dreno* (*drain*) e *porta* (*gate*).

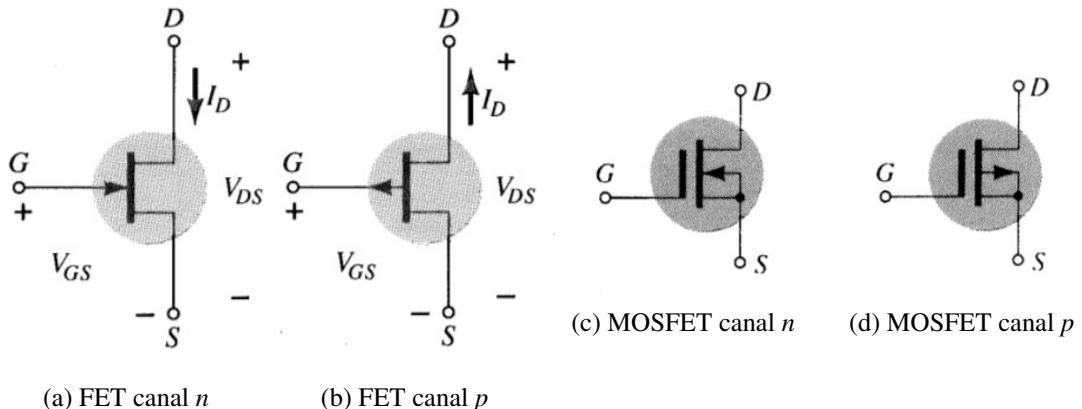


Figura 2.19: Símbolos do JFET e do MOSFET.

Fonte: BOYLESTAD (2011)

Os transistores do tipo FET/MOSFET são também conhecidos como resistores controlados por tensão, o que ajuda a entender seu funcionamento: assumindo que há uma DDP entre os terminais de *dreno* e *fonte* V_{ds} , a resistência à passagem de corrente por estes terminais é controlada pela tensão aplicada à *porta* com relação à *fonte* V_{gs} do dispositivo. Quanto mais próximo de zero é a tensão, maior é o valor da resistência, para os dispositivos FET e MOSFET intensificação [34]. Este comportamento é ilustrado na figura 2.20 e a equação

simplificada 2.4 rege o comportamento de I_d em função de V_{gs} , na qual I_{dss} e V_p são os pontos notáveis no gráfico da figura 2.20 [34].

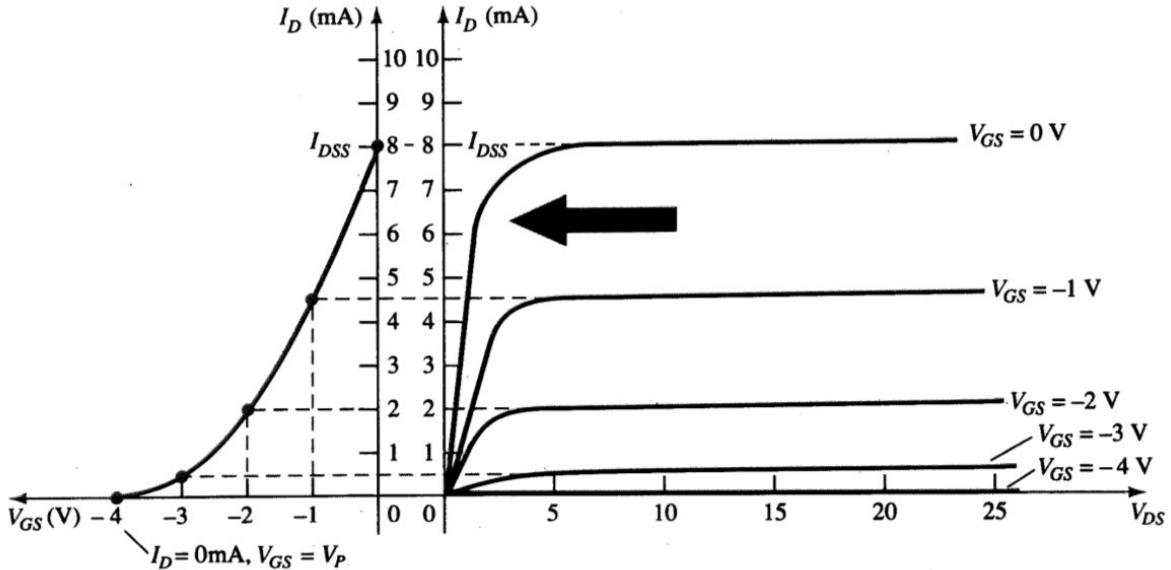


Figura 2.20: Obtendo a curva de transferência das curvas de dreno

Fonte: BOYLESTAD (2011)

$$I_d = I_{dss} \left(1 - \frac{V_{gs}}{V_p} \right)^2 \quad (2.4)$$

De maneira análoga ao BJT, tanto o FET quanto suas variações, aperfeiçoamentos e aplicações possuem uma teoria vasta, que não cabe a este trabalho detalhar. Para o aprofundamento no tema, recomenda-se a leitura da bilbiografia — em especial de [34] e [32].

2.5.3 Sensor de temperatura DS18B20

O sensor de temperatura DS18B20, projetado pela Maxim, é um sensor digital que utiliza somente um pino de dados para comunicação com o dispositivo *master*, cujas regras são estabelecidas pelo protocolo proprietário de comunicação *1-Wire®*[35].

1-Wire é um protocolo de comunicação serial *half-duplex* que utiliza uma linha para transmissão de dados, além da referência. É um protocolo de comunicação *master/slave*, ou seja, um dispositivo *master* inicia a comunicação e os dispositivos *slave* somente respondem aos seus comandos. No caso deste protocolo específico, todo dispositivo possui um identificador de 64 bits único, dentro do qual um byte indica o tipo do dispositivo. Devido ao seu modo de funcionamento, quando a linha de dados é desconectada, os dispositivos *slave* entram em

estado de *reset*, o que favorece seu uso em aplicações de contato. Enquanto a maioria destes dispositivos tem uma faixa de alimentação de 2,80-5,25V e não possui pino para alimentação [36] – utilizando um sistema de alimentação parasita – o sensor DS18B20 tem a possibilidade de alimentação parasita ou não e sua faixa tolerada para funcionamento é de 3-5,5V [35].

Uma vez que este protocolo de comunicação foi concebido para operações nas quais os dispositivos *master* e *slave* estão fisicamente muito próximos, alguns cuidados devem ser observados ao utilizar linhas de transmissão de longas distâncias para manter a boa performance do sistema [37]. Como o estudo da Maxim acerca das implicações do uso de linhas de longa distância negligencia este aspecto para redes com poucos dispositivos e cabeamento curto (menor que 10m), os cuidados de projeto das redes 1-wire não serão aqui especificados, deixando um aviso para trabalhos futuros que visem aplicações de longas distâncias ou com número elevado de dispositivos, a exemplo de uma microcervejaria de porte industrial.

Com relação a aspectos específicos do sensor de temperatura, este apresenta precisão de 0,5°C para a faixa de temperaturas de -10°C a +85°C, e de 2,0°C para a faixa completa de operação, de -55°C a +125°C. Quanto à resolução, esta pode ser programada de 9 a 12 bits, sendo 8 bits para a parte inteira e 1 a 4 bits decimais. O custo da melhor resolução é o aumento no tempo de conversão de uma leitura para valor digital, variando de 93,75ms a 750ms para os casos extremos [35].

A arquitetura interna do CI é apresentada na figura 2.21. Nota-se que o dispositivo, por utilizar o protocolo *1-Wire*, tem uma ROM de 64 bits para identificação [36], além de uma região de memória de rascunho, que consiste de uma memória SRAM de 64 bits, dividida em 8 regiões de 1 byte, para promover a interface com as funcionalidades do CI: sensor de temperatura, alarmes programáveis para estouro de valores mínimo e máximo das medições, registrador de configuração da resolução e gerador de código CRC de 8 bits [35]. A figura 2.22 apresenta o mapa de memória do DS18B20 – nota-se que os registradores de configuração e alarme são copiados para uma memória não volátil (EEPROM) após configurados pelo *master*, retendo seus valores mesmo em caso de corte na alimentação. Outra informação útil obtida a partir da figura 2.22 é o valor da temperatura lido após uma queda de alimentação, que é de +85°C até que o dispositivo esteja pronto para uso [35].

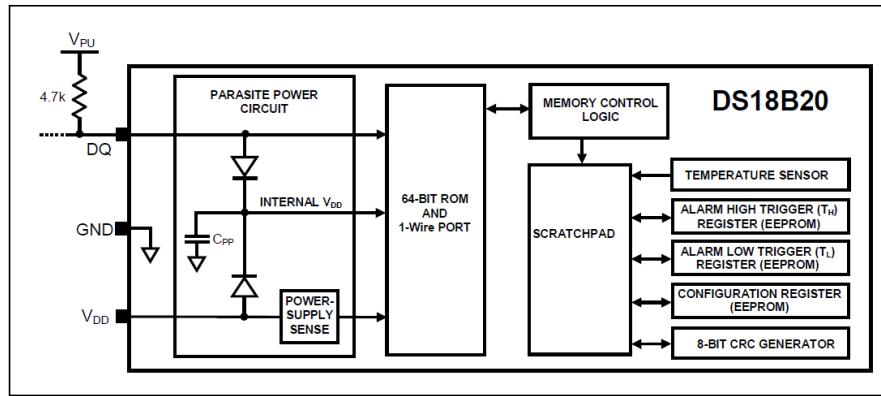


Figura 2.21: Arquitetura interna do DS18B20
Fonte: MAXIM INTEGRATED

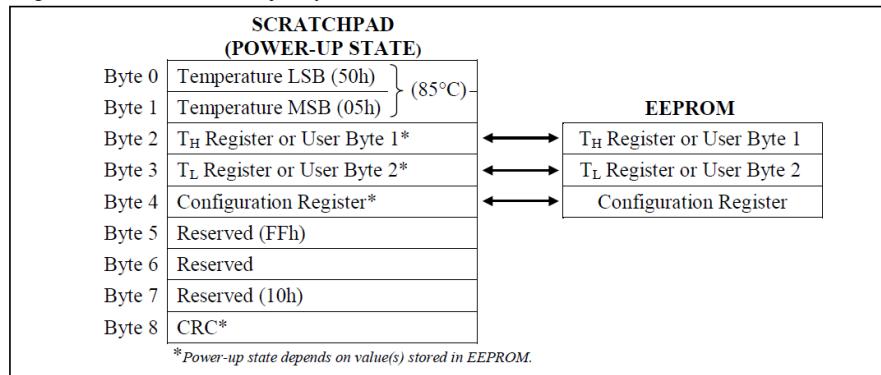


Figura 2.22: Mapa de memória do DS18B20.
Fonte: MAXIM INTEGRATED

As leituras de temperatura são feitas em dois bytes, LSB e MSB, representados na figura 2.23, e que juntos são uma representação em complemento de dois, sendo que os 5 bits mais significativos são redundantes e, para resoluções menores do que 12 bits, o valor dos bits menos significativos é indefinido (1 bit indefinido para 11 bits de resolução; 2 bits indefinidos para 10 bits de resolução, etc).

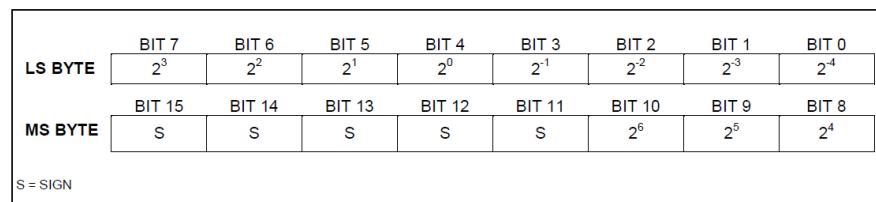


Figura 2.23: Registradores de temperatura do DS18B20.
Fonte: MAXIM INTEGRATED

O Kernel do Linux possui *device drivers* que suportam parcialmente o uso do sensor DS18B20, portanto não há a necessidade de descrever em detalhes o protocolo *1-Wire*. O

device driver wl-gpio controla o barramento (*master*) *1-wire* utilizando a API GPIO para controlar a linha, sendo que a porta utilizada é especificada utilizando dados específicos da placa. O *device driver wl-therm* suporta algumas famílias de sensores de temperatura da Maxim (*slave*), dentre elas a DS18*20, fazendo conversões básicas de temperatura e fornecendo-as ao sistema por meio do FS (*File System* ou Sistema de Arquivos). O resultado da abertura e leitura do arquivo correspondente a um sensor, é a conversão de temperatura pelo *driver* e posterior fornecimento dos dados recebidos pela placa, checagem do CRC e temperatura em °C/1000. Este driver não suporta conversão de temperatura simultânea de múltiplos sensores nem redução da precisão de conversão, portanto só são possíveis leituras de 12 bits. Em caso de alimentação parasita, somente um sensor pode estar conectado à linha por vez e, caso a placa ofereça suporte a *pull-up* interno, o driver é capaz de usá-lo [38].

2.6 Sistema de controle de temperatura

2.6.1 Controlador PID

2.6.2 Interface com sensores e atuadores

Capítulo 3

Materiais

Este capítulo lista os materiais empregados no desenvolvimento deste projeto.

3.1 Núcleo do projeto

Os materiais utilizados no núcleo do mesmo estão contidos na tabela 3.1:

Tabela 3.1: Lista de materiais que compõem o núcleo do projeto

Descrição	Quantidade
BeagleBone Black Rev. C	1
Fonte de tensão 5V/2A	1
Cartão µSD 4GB	1
Adaptador Wi-Fi USB Ralink RT5370	1
Sonda de aço inoxidável à prova d'água com sensor DS18B20	1

3.2 Sistema mecânico

Os materiais utilizados na construção da parte mecânica do sistema estão listados na tabela 3.2:

Tabela 3.2: Lista de materiais utilizados na confecção do subsistema mecânico

Descrição	Quantidade
Itens diversos	
Caldeirão de alumínio n.º36 (32 litros)	2
Resistor helicoidal com duas voltas, 110V/2200W	2
Fundo falso de aço inoxidável circular $\phi=36\text{cm}$	1
Bomba centrífuga magnética 12VDC Topsflo TL-B08H-12-1006	2
Trocador de calor de contra-fluxo, tubo de alumínio de 7m	1
Tecido mussalina x 1,5m (centímetros)	30
Tubulação	
Mangueira silicone 3/8"(metros)	1
Mangueira cristal 3/8"(metros)	2
Mangueira cristal 1/2"(metros)	7
Tubo schedule S40 1/2" aço inoxidável (metros)	3
Valvulas e acessorios	
Válvulas e acessórios	
Válvula de retenção portinhola 1/2" aço inoxidável	2
Válvula esfera latão, passagem plena 1/2"	2
Válvula solenóide 1/2"12V plástico uso geral	1
Válvula solenóide Danfoss 1/2"EV210BD 032U3620	3
Bobina Danfoss 15W 12VDC 042N7550	3
Conexões e acessórios	
Espigão de latão com redução 1/2- 3/8"	3
Bico de engate rápido de plástico 1/2"	1
Engate rápido de plástico 1/2"	1
Abraçadeira 3/8"	4
Abraçadeira 1/2"	2
Luva sextavada 1/2"latão	1
Arruela aço inoxidável 1/2"	4
Anel de vedação de silicone 1/2"	4
Contra porca aço inoxidável 1/2"	2
Cotovelo fêmea 90° 1/2" aço inoxidável	1
Cotovelo M/F 90° 1/2" aço inoxidável	1
Tee fêmea 90° 1/2" aço inoxidável	4
Niple duplo sextavado 1/2" aço inoxidável	9
Espigão sextavado 1/2" aço inoxidável	4

3.3 Componentes eletrônicos

Os componentes eletrônicos utilizados para confecção dos circuitos de interface com a BBB, acionamentos de potência e circuitos diversos estão listados na tabela 3.3:

Tabela 3.3: Lista de componentes eletrônicos diversos empregados na confecção de PCBs e cabeamento do sistema

Componente	Modelo/valor/descrição	Quantidade
Placa de acionamentos de potência		
Capacitor cerâmico	100nF	1
Capacitor poliéster	330nF	1
Resistor	2,2kΩ 1/4W 5%	10
Resistor	22kΩ 1/4W 5%	10
Optoacoplador	4N25 DIP	7
Regulador de tensão	LM7805 TO220	1
Transistor MOSFET	IRF540N TO220	7
Díodo retificador	1N4007	6
Conecotor para alimentação	J4	2
Conecotor header	10 vias	1
Borne KRE	2 vias	6
Porta fusível	25mm 6A(MAX)	1
Fusível de vidro	20AG 5A 20mm	1
Cabeamento		
Conecotor latch	10 vias	1
Cabo flat (metros)	10 vias	2
Componentes diversos		
Fonte de tensão	12V/5A	1
Relé de estado sólido	Fotek SSR-25 DA (25A)	2
Cabo de cobre (metros)	4mm seção transversal	6
Tomada	20A	1
Adaptador T	padrão NEMA	1
Terminal anel	pré-isolado M6	4
Plug tomada	20A padrão NBR 14136	1
Servo Motor	TowerPro MG995	1

3.4 Equipamentos, instrumentos, softwares e miscelânea

Os equipamentos, instrumentos de medição e demais objetos e/ou softwares de suporte ao desenvolvimento do projeto estão listados na tabela 3.4.

Tabela 3.4: Lista de equipamentos, instrumentos e *softwares* de suporte ao desenvolvimento do projeto

Descrição	Detalhes
Instrumentos de medição	
Osciloscópio Agilent DSO-X 2002A	70MHz - 2 canais
Multímetro Minipa ET-2110	
Softwares	
MATLAB	versão
FreeCAD	v. 0.15
MatterControl	v. 1.3.0 + plugin CNCBrasil
Cura	v. 15.06.03
Putty	
Diversos	
Impressora 3D	CNCBrasil Standard
Filamento para impressão 3D	ABS 1,75mm
Controle de versão	
GitHub	https://github.com/leograbafinal_paper_tcc
Filamento para impressão 3D	ABS 1,75mm

3.5 Insumos cervejeiros

A descrição dos insumos cervejeiros utilizados para confecção da receita e posterior teste do sistema pode ser encontrada na tabela 3.5.

Tabela 3.5: Lista insumos cervejeiros para produção de cerveja do tipo *Blond Ale*

Descrição	Quantidade
Malte Pilsen (kg)	4,5
Malte Munich (kg)	0,5
Lúpulo Hallertauer Tradition (g)	50
Levedura desidratada Safale S-04 (Tipo Ale inglesa) sachê	1
Sanitizante Iodophor diluído em água (ml/l)	0,8

Capítulo 4

Métodos

Este capítulo descreve todos os procedimentos de projeto e testes, dentre outros, empregados visando a obtenção do resultado final deste trabalho. Os procedimentos aqui empregados são embasados na teoria apresentada no capítulo 2. Cabe salientar que, embora este seja um projeto de integração entre *mecânica*, *hardware* e *software*, os esforços foram concentrados na área de desenvolvimento de *softwares*, desde o *firmware* da PRU-ICSS integrado ao servidor Apache posteriormente migrado para Node.js, até a aplicação de usuário, escrita em diversas linguagens, de programação ou não — a exemplo de HTML e CSS.

4.1 Estrutura mecânica

Para o projeto da parte mecânica, em primeiro lugar foi concebida uma configuração de sistema baseada tanto na literatura disponível quanto em projetos de sistemas disponíveis para aquisição no mercado: um sistema de duas panelas análogas à MT e ao BK, sendo que durante o processo de cozimento do mosto, o BK pudesse ser utilizado como HLT e, portanto, fez-se necessário o projeto de um sistema de recirculação entre as duas panelas. Observava-se que esta proposta representa um híbrido entre um sistema de 3 panelas tradicional - que consiste de uma panela para mosturação, uma para lavagem dos grãos, ou sparging, e uma para fervura; e o sistema popularizado pela Braumeister, composto de 1 panela, com recirculação. Esta abordagem permite a economia financeira e simplificação mecânica do sistema de três panelas com o benefício do *sparging* inexistente no sistema de uma panela.

4.1.1 Funcionamento da estrutura

Na figura 4.1 é apresentada uma representação esquemática da parte funcional.

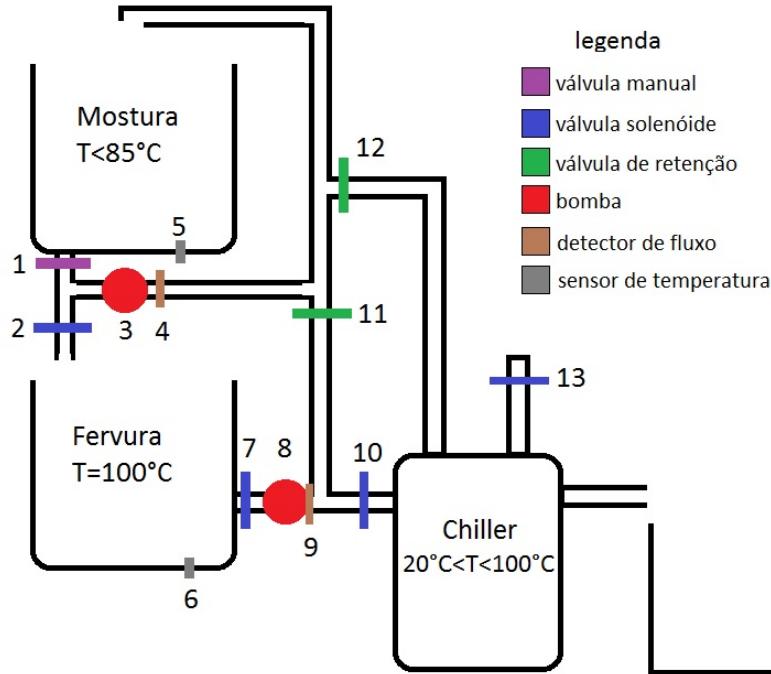


Figura 4.1: Representação esquemática da estrutura mecânica

Na panela superior ou MT, rotulada como *Mostura*, os grãos são adicionados após o aquecimento inicial da água:

- A válvula 1 é uma válvula manual do tipo esfera, com passagem plena. Durante a operação do equipamento ela deve ficar sempre aberta, portanto seu uso é realizado somente em caso de emergências.
 - Após a adição dos grãos, a bomba 3 é ligada para que o líquido da panela seja recirculado. As válvulas de retenção 11 e 12 não permitem que o líquido passe por elas, portanto sobra somente um caminho possível para este, que é ascender na tubulação até entrar novamente na MT. Note-se que não há válvula automatizada na saída da MT, uma vez que o mosto fica em recirculação constante ao longo do processo de cozimento.
 - Após a mostura a bomba 3 é desligada e a válvula solenóide 2 é aberta, escoando o líquido para a panela de fervura BK. Simultaneamente, a válvula solenóide 7 e a bomba 8 são ativadas, fazendo com que a água de lavagem presente na BK/HLT flua pela tubulação até a MT, iniciando o processo de *sparging*. Durante um tempo predefinido, o mosto e a água de lavagem recirculam pelas duas panelas e se misturam.
 - Durante a fervura, as válvulas 2 e 7 são fechadas. Neste período a MT deve ser esvaziada, já que ela será usada posteriormente para armazenamento da água de esfriamento do mosto — água usada para limpeza do sistema e que reduz a produção de efluentes

do mesmo.

- Após a fervura, as válvulas solenóides 7 e 10 são ativadas, permitindo o escoamento do líquido para o trocador de calor. Simultaneamente a válvula solenóide 13 é aberta e água fria circula em contra-fluxo para resfriar o mosto. Esta água sai quente do trocador de calor e passa pela válvula de retenção 12, preenchendo a parte superior da tubulação e sendo armazenada na MT para posterior limpeza do sistema.
- O mosto que sai resfriado do *chiller* cai no balde de fermentação. Neste processo o líquido entra em contato com o ar, o que é não somente desejável como essencial para o sucesso da fermentação, portanto esta é a última parte do processo automatizada. Uma solução futura e que permite automação é o uso de um aerador em conjunto com um tanque de fermentação selado.

Para o bom funcionamento do sistema proposto, algumas condições devem ser atendidas:

- Se o volume total de líquido nas duas panelas ao final da mostura exceder o volume da BK, ela transbordará. Por este motivo é importante que exista um sistema de detecção de transbordo. Ainda assim, deve-se requisitar que o usuário do sistema calcule corretamente as proporções da receita, pois apesar do sistema automático anti-transbordo, ocorrerá perda de insumos devido ao acúmulo de mosto inutilizável na MT em caso de erro.
- São necessários filtros de material particulado para o bom funcionamento das válvulas e bombas [39].
- Os dispositivos mecânicos (válvulas, bombas, tubulação, dentre outros) e eletrônicos (sensores e atuadores) em contato com o sistema mecânico devem ser apropriados às altas temperaturas impostas pela natureza do processo [39, 40].
- As panelas, tendo como referência seu fundo, não devem ser alinhadas concêntricas, já que isto torna a adição de lúpulos e o processo de manutenção do equipamento desajeitados.
- A automação da técnica de *whirlpool* não é contemplada nesta configuração de sistema. Se o usuário a considera necessária, ele deve se encarregar de executá-la manualmente.
- A única válvula com pressão suficiente para ser servo-operada é a 13, assumindo-se que a pressão incidente sobre ela seja maior do que 0,5bar, portanto as outras devem ser de operação direta [39, 40].
- A limpeza automatizada, ou CIP, não é contemplada nesta configuração do sistema, devido à alta complexidade [41], conforme indica a figura 4.2, e ao custo de implemen-

tação proibitivo para este trabalho. Não obstante, é recomendado ao usuário do sistema que faça a limpeza do mesmo tão rápido quanto possível após o final da produção.

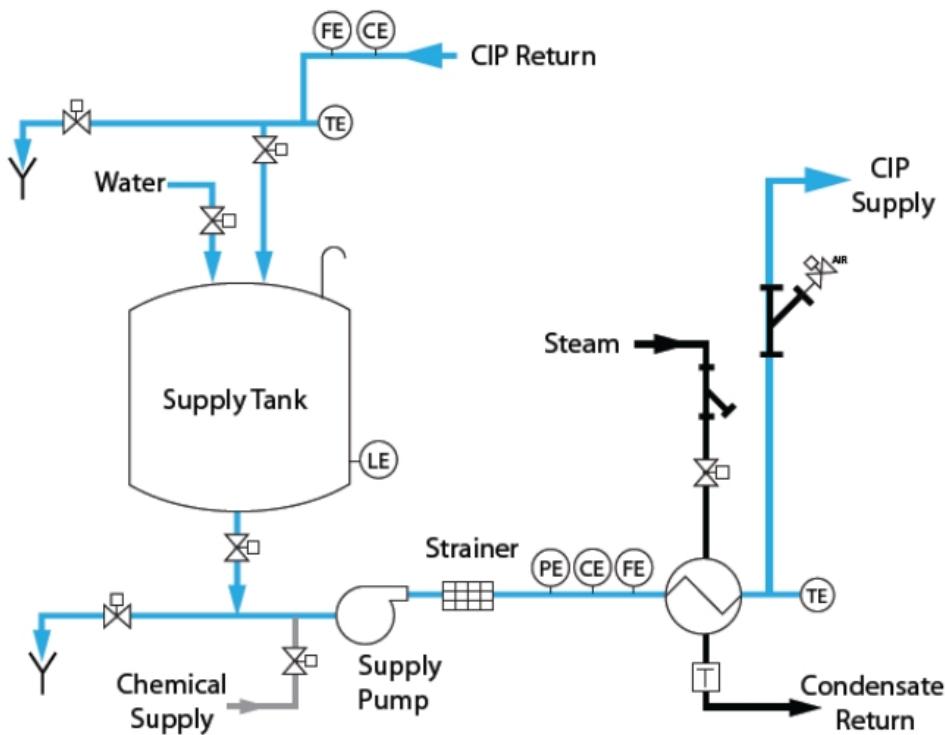


Figura 4.2: Sistema CIP básico.
Fonte: ROSE e MONTGOMERY (2010)

4.1.2 Dimensionamento da parte funcional

Com a filosofia de operação do sistema mecânico definida, é possível realizar o dimensionamento deste sistema. A primeira consideração a ser feita é que os materiais e métodos aqui empregados não seguem nenhuma norma que permita o uso deste equipamento para fabricação de cerveja para venda — tal escolha é feita em função do alto custo de um sistema completamente dentro das normas e também pelo fato de este ser um trabalho cujo foco é a automação e o acesso remoto.

Não obstante, o material da tubulação escolhido foi o aço inoxidável AISI304. Já que o volume de líquido a ser trabalhado é pequeno, menor do que 40 litros, e a pressão de trabalho não é maior do que a pressão das bombas escolhidas posteriormente, foi decidido o uso de tubulação de diâmetro de 1/2" (21,34mm de diâmetro externo) e espessura da parede no padrão Schedule 40 (2,77mm de espessura). Embora esta espessura seja superdimensionada para a presente aplicação, o mecânico responsável pela montagem do sistema a requisitou

para que fosse possível fazer rosca sem danificar a integridade dos tubos. As conexões, seguindo a escolha dos tubos, também são de aço inoxidável de 1/2".

As ligações entre tubos, conexões e outros componentes do sistema são ligações rosqueadas. Este é um meio de ligação antigo, porém de baixo custo, fácil execução e usualmente empregadas em tubulações de diâmetro nominal pequeno, menor do que 4"[42]. Para vedação foi escolhido o Teflon, uma vez que é um material de baixo custo e alta disponibilidade e o padrão de rosca adotado neste projeto é o BSP, baseado na norma ISO. Cabe salientar que, embora as ligações rosqueadas sejam permitidas sob certas circunstâncias na prática, elas são relegadas a instalações de baixa responsabilidade [42].

As panelas foram escolhidas no material de alumínio, em função do custo proibitivo do aço inoxidável. São caldeirões padrão utilizados no ramo de hotelaria e restaurantes e disponíveis em vários volumes. As duas panelas utilizadas neste trabalho tem capacidade para 32 litros e diâmetro do fundo de 36cm, portanto sua especificação no mercado é *caldeirão de alumínio n. 36*. Tal volume, 60% superior ao da produção máxima aconselhada, é necessário devido ao volume dos grãos, à quantidade da água de lavagem e às perdas por evaporação que exigem uso de um volume de água inicial superior ao nominal.

O parâmetro inicial escolhido para seleção das válvulas solenóide foi a temperatura máxima de operação, cuja fonte de dados de operação é fornecida pelos fabricantes. Em seguida, foi escolhida uma vedação adequada: os dois tipos mais comuns de vedação com suporte a temperatura de pelo menos 100°C são o EPDM e FKM (etileno-propileno-dieno e fluoreto de vinilideno, respectivamente) [39]. Estes materiais possuem uma tabela de compatibilidade de fluidos cuja classificação pode ser satisfatória, boa, duvidosa, insatisfatória e desconhecida — tanto para cerveja quanto para o mosto, ambas as vedações são classificadas como satisfatórias, embora para água a classificação do FKM seja somente boa [43]. Por fim, um parâmetro que foi verificado antes da escolha final das válvulas é a posição de operação, que varia conforme os modelos do fabricante [39]. Com base nos parâmetros supracitados, foi decidido usar:

- Válvula solenóide 1/2"12V plástico uso geral para água de resfriamento do trocador de calor.
- Válvula solenóide Danfoss 1/2"EV210BD 032U3620 para demais válvulas.
- Bobina Danfoss 15W 12VDC 042N7550 para acionamento das solenóides Danfoss.

Informações técnicas sobre a válvula EV210BD 032U3620 estão no anexo III e sobre a bobina 042N7550 no anexo IV.

As bombas de recirculação foram escolhidas com base na temperatura de operação e grau alimentício. Posteriormente, tanto a vazão quanto a perda de carga do sistema foram estimadas para verificar se a bomba escolhida seria adequada: o modelo Topsflo B08H-12-1006 tem capacidade máxima de vazão de 10l/min e carga expressa em coluna de água de 6m. Como a capacidade máxima do sistema é de 20l e idealmente a temperatura deve subir à taxa de 1°C/min, além de que a altura do sistema é menor do que 2m e as perdas de carga foram consideradas desprezíveis para este caso, foi decidido que a bomba escolhida é adequada ao projeto. No V encontra-se a folha de dados técnicos do dispositivo.

Quanto às conexões, a figura 4.3 apresenta um diagrama conceitual do sistema, contendo todas as peças necessárias para a montagem do mesmo:

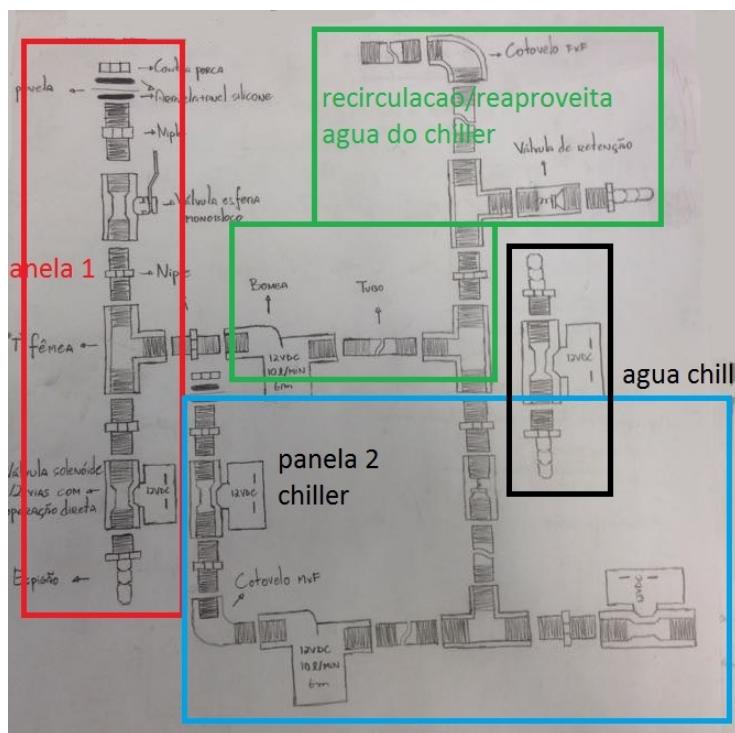
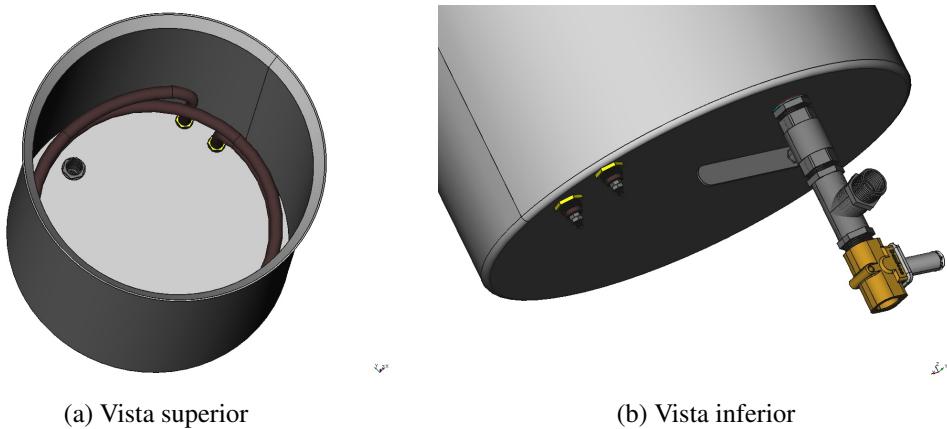


Figura 4.3: Diagrama conceitual das conexões mecânicas

Na figura 4.4 é apresentado um diagrama da MT construído no software FreeCAD, à qual estão conectados o resistor de potência e um niple com vedação. Também estão presentes na figura a válvula esfera manual, um *tee* fêmea, uma válvula Danfoss EV210B e três niples de conexão entre estes componentes. O modelo da válvula foi obtido a partir do website da Danfoss e importado, enquanto a modelagem dos outros componentes foi desenvolvida na plataforma FreeCAD. Na figura 4.5 são apresentadas em detalhe as conexões do resistor e do niple à panela e, por isto, a estrutura da mesma foi omitida. Os itens em amarelo são contra-porcas, em azul arruelas e, em vermelho anéis de vedação (*o-rings*).



(a) Vista superior (b) Vista inferior

Figura 4.4: Diagrama da panela de mostura e conexões

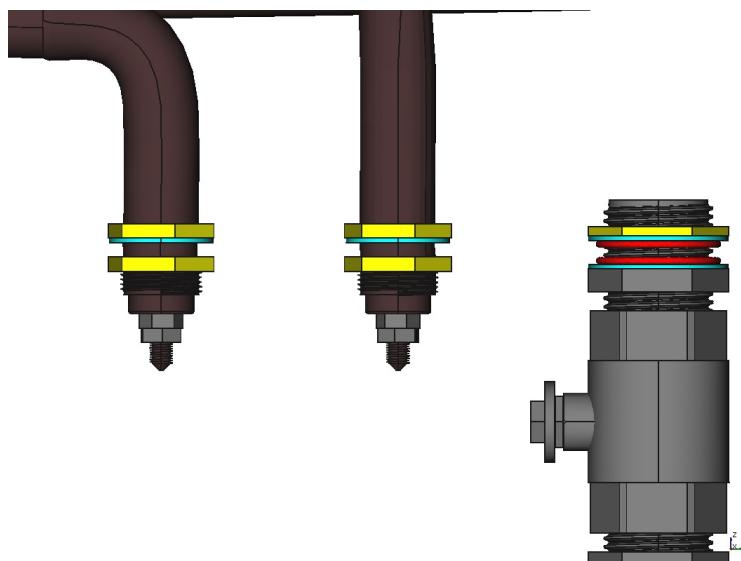


Figura 4.5: Detalhes da vedação da panela de mostura

4.1.3 Estrutura metálica de suporte

Parte importante do projeto é o dimensionamento da estrutura metálica de suporte às panelas, já que esta é responsável não somente pelo alojamento do sistema mecânico como também pela facilidade de manutenção. Para definição da altura das panelas, foram observadas primeiramente as alturas das panelas, da estrutura de adição de lúpulos (que será detalhada à parte na seção 4.2) e do *chiller*. Na sequência, foi estimado um espaço de manobra entre as duas panelas e uma altura mínima do chão, com base na figura 4.1. Todas as medidas citadas estão dispostas na tabela 4.1.

Tabela 4.1: Dimensões elementos mecânicos relevantes para o projeto da altura da estrutura metálica

Descrição	Altura (cm)
Panela BK	30,0
Panela MT	30,0
<i>Chiller</i>	30,0
Estrutura dos lúpulos aberta	20,0
Vão livre entre panelas	20,0
Vão do <i>chiller</i> ao BK	10,0
Espaço entre MT e topo	10,0

Somando as alturas medidas e estimadas, obteve-se que a estrutura metálica deve ter 1,5m de altura. A largura deve ser de 36,5cm para acomodar as panelas somente com uma pequena folga e o comprimento escolhido é de 70cm, o que permite alinhar as panelas de maneira não concêntrica, conforme abordado na seção 4.1 — este valor é suficiente, já que o diâmetro somado das duas panelas é de 72cm e deve haver ao menos uma intersecção para permitir o escoamento por gravidade da MT ao BK. Na figura 4.6 é apresentado o projeto da estrutura já com as prateleiras para acomodar as panelas. Note-se que foram escolhidas cantoneiras de 2"x 1/8"para a construção deste equipamento.

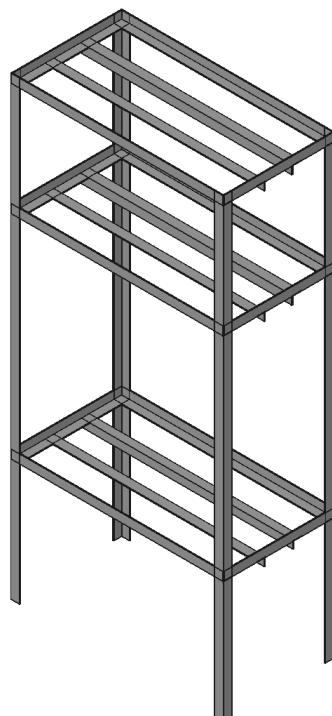


Figura 4.6: Projeto da estrutura metálica

4.2 Estrutura de adição de lúpulos

Devido à necessidade de adicionar lúpulos à receita em proporções e tempos predefinidos, foi necessário o projeto de uma estrutura automática de adição de lúpulos. Em função das inúmeras possibilidades de combinações de lúpulos e temporizações, a escolha natural do projeto foi uma estrutura que permita esta flexibilidade — uma caixa com oito compartimentos separados para adição sequencial destes insumos. Com base em um pacote de lúpulos em *pellets* de 50g, estimou-se que seu volume é de 160cm^3 e, sabendo-se que uma receita de 20l dificilmente utiliza uma quantidade maior do que 400g de lúpulos, a decisão tomada foi a de que cada um dos 8 compartimentos deveria ter os 160cm^3 necessários para acomodar 50g. Para confirmar que 400g é uma quantidade razoável, foi seguido o cálculo de IBU de [1], detalhado no apêndice E.

Fixando a altura do compartimento em 10,0cm foram escolhidos valores de comprimento e largura de 16,0cm e 8,0cm respectivamente, assumindo inicialmente que as paredes das divisórias tem espessura desprezível. A equação para volume de um prisma e a escolha das dimensões são demonstradas em 4.1:

$$V = l \cdot w \cdot h = V_{50g} \cdot 8 \quad (4.1a)$$

$$= l \cdot w \cdot 10 = 160 \cdot 8$$

$$l \cdot w \cdot 10 = 160 \cdot 8, \text{ fixando } l = 16\text{cm} \quad \therefore$$

$$w = 160 / 16 = 10\text{cm} \quad e \quad (4.1b)$$

$$w_{compartimento} = l / 8 = 16 / 8 = 2\text{cm} \quad (4.1c)$$

Posteriormente aos cálculos de 4.1, durante o projeto no software FreeCAD, observou-se que a assumpção de espessura desprezível é falsa e um retrabalho nas medidas do compartimento levou às dimensões externas finais de 18,0cm x 8,4cm x 10,0cm; as paredes externas tendo espessura de 3,0mm e as internas de 2,0mm. A figura 4.7 esboça as dimensões externas da caixa e a figura 4.8 apresenta o corpo da estrutura.

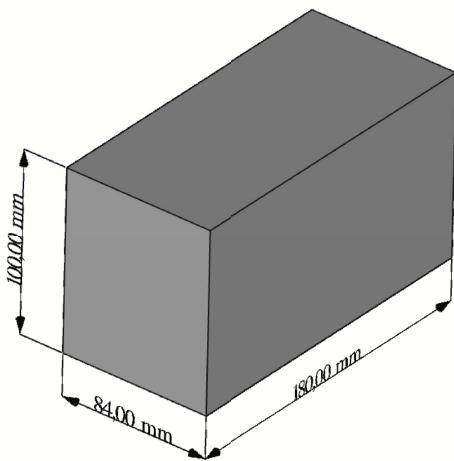
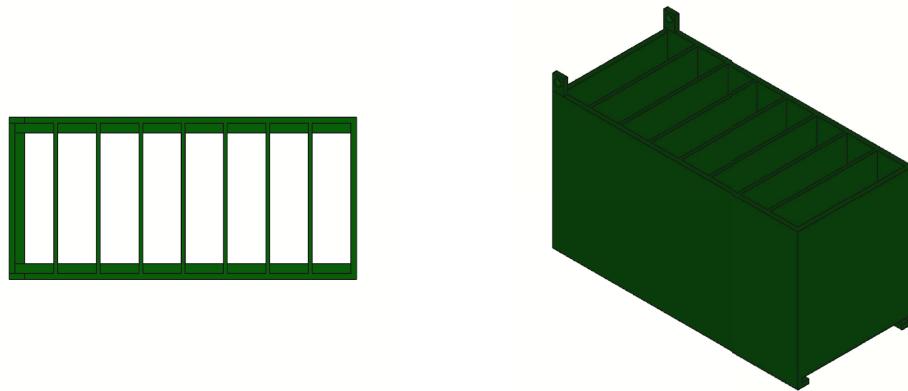


Figura 4.7: Dimensões da estrutura de adição de lúpulos

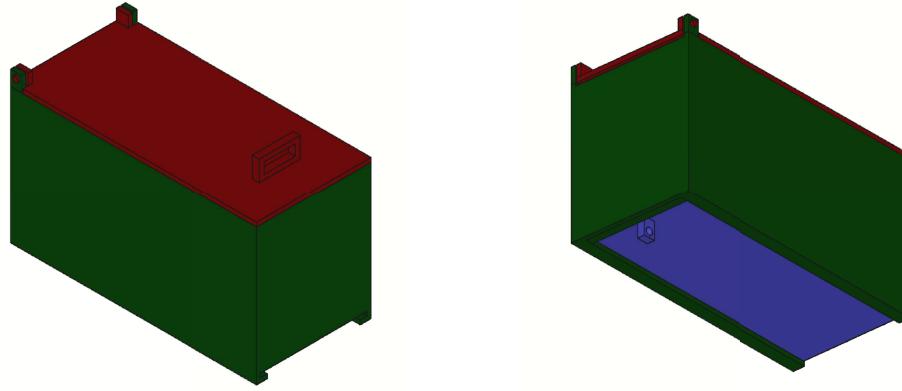


(a) Vista superior

(b) Vista axiométrica

Figura 4.8: Corpo da estrutura de adição de lúpulos

Foram projetadas duas portinholas: uma na face superior da caixa, para reabastecimento manual dos lúpulos, com uma alça e fixada na caixa por dois pinos, de modo que a abertura desta tampa é um movimento de rotação e; uma na face inferior, para adição automática, de correr, de modo que a abertura é um movimento de translação; ambas estão ilustradas na figura 4.9.



(a) Tampa manual para reabastecimento (b) Tampa para adição automática dos lúpulos

Figura 4.9: Tampas da estrutura de adição de lúpulos

Duas varetas de comprimento l_1 e l_2 são ligadas ao encaixa da tampa de correr, representada em azul na figura 4.9b, e a elas é acoplado um servo-motor cujo ângulo de operação θ define a abertura — ou deslocamento — da tampa, definido por d . A figura 4.10 esquematiza as ligações.

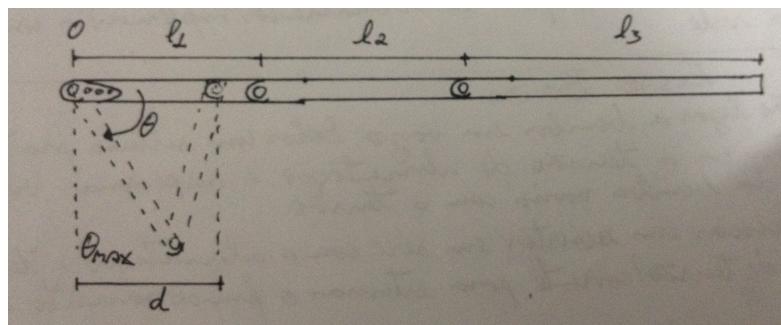


Figura 4.10: Esquema de ligação das varetas à tampa de correr

Sendo o comprimento da tampa igual a l_3 , a condição em 4.2 deve ser satisfeita para que a excursão máxima de abertura da caixa possa ser atingida. É possível demonstrar com o auxílio da Lei dos Cossenos, em 4.3 que $l_1 = l_2 = l$ para que a mínima abertura da caixa possa ser atingida.

$$l_1 + l_2 \geq l_3 \quad (4.2)$$

$$l_2^2 = d^2 + l_1^2 - 2dl_1 \cdot \cos(\theta) \quad (4.3)$$

se $\theta = 90^\circ \rightarrow d = 0 \therefore$

$$l_2^2 = l_1^2 \rightarrow l_2 = l_1 = l \quad (4.4)$$

A taxa máxima de deslocamento em função do ângulo do servo motor é expressa em 4.5:

$$\frac{dl_3}{d\theta} = \frac{2l}{90^\circ} \quad (4.5)$$

Entretanto, quanto maior o valor desta taxa, menor é a resolução de abertura da caixa, portanto obtem-se que o menor valor possível de l , a partir de 4.2 é $l = l_3/2$. Assim sendo, a taxa de deslocamento calculada a partir de 4.5 é de $2mm/\circ$.

Embora o valor teórico de θ seja 90° , na prática, devido ao encaixe entre tampa e vareta não ser na extremidade, este valor é reduzido. Tal fato não deve ser negligenciado, uma vez que isto pode forçar a operação do servo-motor. Sabe-se pelo projeto executado no FreeCAD que a distância da borda ao encaixe da tampa é de 2,3cm portanto, com o auxílio da Lei dos Cossenos, obtém-se que $\theta_{max} = 82^\circ$. Quanto ao valor mínimo não há preocupação, já que a abertura da caixa terá uma excursão reduzida em 2,3cm com relação ao seu comprimento. As varetas não foram calculadas novamente pois julgou-se uma boa prática essa redução, que evita que a tampa caia da caixa quando em máxima excursão. A figura 4.11 apresenta o esquema completo de construção da estrutura de adição de lúpulos.

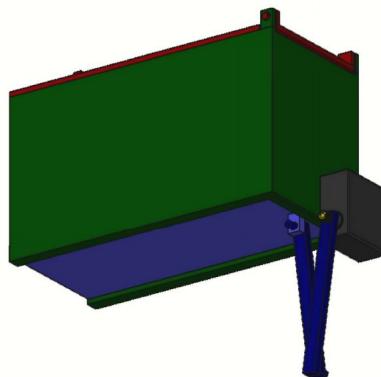


Figura 4.11: Estrutura de adição de lúpulos

4.3 Configuração da BeagleBone Black

O primeiro passo para o desenvolvimento do projeto foi a configuração da BeagleBone Black, que é o coração deste sistema. Para tal, foi escolhido o procedimento de gravar a distribuição Debian pré-compilada do sistema operacional Linux na memória de armazenamento eMMC de 4GB da placa. Esta distribuição foi escolhida em função da facilidade de obtenção de suporte em fóruns da internet, artigos e livros, dentre outros, além de ser o SO oficial da BBB [44].

NOTA: Na ocasião da realização deste procedimento, havia uma diferenciação entre a imagem fornecida para uso diretamente a partir do cartão µSD e a imagem para gravação no eMMC, porém agora o procedimento adotado pela fundação BeagleBoard.org é fornecer somente uma imagem pronta para uso diretamente do cartão µSD e, no caso de o usuário decidir gravar a distribuição no eMMC, o arquivo uEnv.txt na partição de *boot* do cartão µSD deve ser editado conforme instruções antes de ser inserido na BBB.

Tanto a última versão estável do Debian para BBB (v. 7.9), conhecida como *Wheezy* e a última versão em desenvolvimento (v. 8.2), conhecida como *Jessie*, quanto a versão utilizada neste projeto (v. 7.5) podem ser obtidas a partir do site oficial da BBB [45], conforme ilustrado na figura 4.12. A instalação da versão 7.5 se resume a:

1. Baixar a imagem Debian 7.5 para eMMC do site beagleboard.org/latest-images
2. Descomprimir a imagem
3. Gravar a imagem em um cartão µSD formatado (não basta copiar, é preciso utilizar um software que grava a imagem corretamente. Neste caso foi utilizado o *Win32 Disk Imager*)
4. Inserir o cartão µSD na BBB desligada
5. Pressionar o botão de boot e energizar a placa com ele ainda pressionado
6. Esperar um dos leds de status acender antes de largar o botão
7. Esperar os 4 leds de status ficarem continuamente acesos (durante o processo de gravação eles trabalharão de maneira intermitente)
8. Desenergizar a BBB e retirar o cartão µSD

BeagleBoard.org Latest Firmware Images

Download the latest firmware for your BeagleBoard, BeagleBoard-xM, BeagleBone, BeagleBone Black, Seeed BeagleBone Green or Arrow BeagleBone Black Industrial

See the [Getting Started guide](#) and the [official wiki page](#) for hints on loading these images.

Recommended Debian Images

Wheezy for BeagleBone, BeagleBone Black and Seeed BeagleBone Green via microSD card

- [Debian 7.9 \(BeagleBone, BeagleBone Black, Seeed BeagleBone Green - 4GB SD\) 2015-11-12 - more info - sha256sum: f6e67ba01f159d202c655f5e429c3a6c2398123bc3d38d54846c597275d277](#)

Jessie for BeagleBone, BeagleBone Black, Seeed BeagleBone Green and Arrow BeagleBone Black Industrial via microSD card

- [Debian 8.2 \(BeagleBone, BeagleBone Black, Seeed BeagleBone Green, Arrow BeagleBone Black Industrial - 4GB SD\) 2015-11-12 - more info - sha256sum: 47142693655004c733fb00abe7820f0a6b7dc02fb5a5f03a5d099ebc79b6c](#)

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 http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

Older Debian images

BeagleBone and BeagleBone Black via microSD card (without flashing the eMMC)

- [Debian 7.8 \(BeagleBone, BeagleBone Black - 4GB SD\) 2015-03-01 - more info - bittorrent - md5: c848627722b7a5f7bc89791cc8949e3b](#)
- [Debian 7.5 \(BeagleBone, BeagleBone Black - 2GB SD\) 2014-05-14 - more info - bittorrent - md5: 35877ce21e8ed0eb1bdcc6819ad71c317](#)
- [BeagleBone Black \(eMMC flasher\)](#)
- [Debian 7.5 \(BeagleBone Black - 2GB eMMC\) 2014-05-14 - more info - bittorrent - md5: 74615fb680afbf252c034d3807c9b4ae](#)



Figura 4.12: Últimas imagens pré-compiladas disponíveis para a BBB.

Fonte: Adaptado do site oficial da Fundação BeagleBoard.org¹

Após este procedimento, a BBB estará pronta para uso a partir da memória interna eMMC. A verificação do funcionamento do sistema foi feita conectando a placa a um computador instalado com Windows 7 e conectado à internet, via interface USB e posterior acesso através do software de código aberto para acesso via SSH *Putty*, disponível para *download* no site oficial putty.org [46] (qualquer software similar que possibilite o acesso via SSH pode ser utilizado para este fim). Para isso, é preciso também instalar no computador o driver que dá acesso à BBB via USB, disponível em beagleboard.org/getting-started. O acesso via SSH se dá utilizando o endereço de IP **192.168.7.2**, usuário: **debian** e senha: **debian** [16]. A figura 4.13 mostra o ambiente de configuração do *Putty* e a figura 4.14 apresenta a tela após uma tentativa de login bem sucedida.

¹Disponível em <http://beagleboard.org/latest-images>

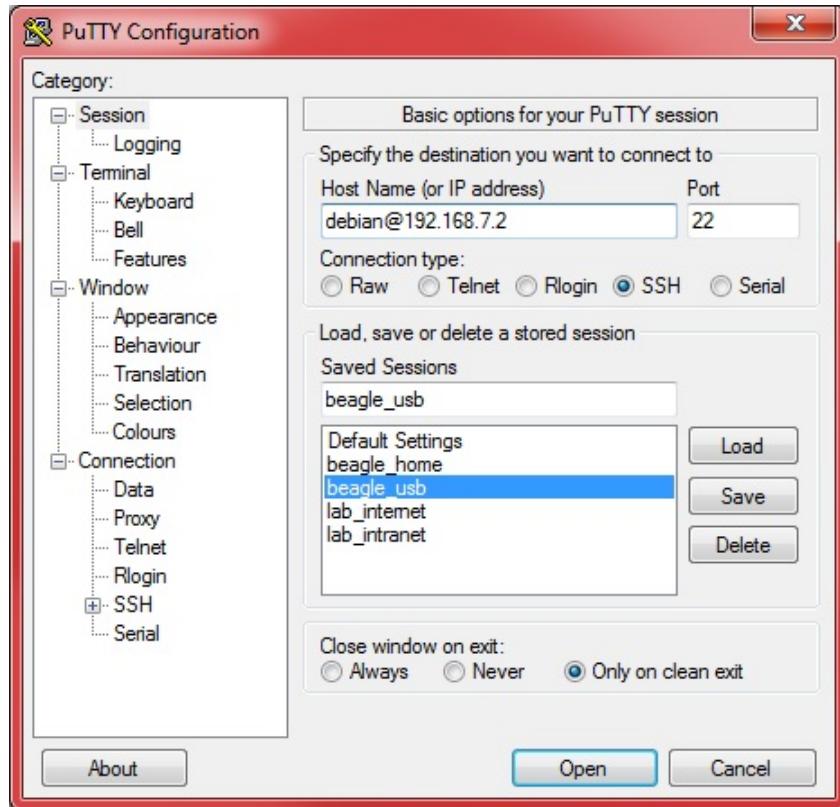


Figura 4.13: Configuração do *Putty* para acesso SSH via USB

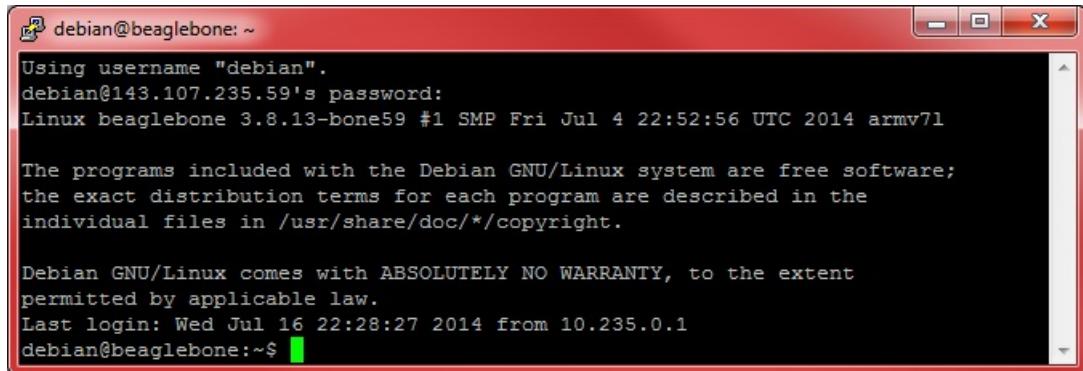


Figura 4.14: Tentativa de login bem sucedida pelo *Putty*

4.3.1 Ajustes de rede para uso do adaptador Wi-Fi/USB

Para utilizar o adaptador Wi-Fi USB, em primeiro lugar este foi conectado à BBB. Em seguida, no terminal, o dispositivo foi listado para confirmar que o sistema estava identificando-o corretamente e ativado, permitindo assim obter a lista de redes Wi-Fi próximas. Com isso, foi possível obter os dados da rede Wi-Fi de interesse, para posterior edição do arquivo */etc/network/interfaces*, responsável pelas configurações de rede do sistema – este pode ser obtido no apêndice D. O código 4.1 ilustra a sequência de comandos executados:

```

1 lsusb # verifica se o dispositivo USB foi detectado
2 sudo ifconfig wlan up # ativa a interface de rede USB
3 sudo iwlist scan # lista as redes Wi-Fi disponíveis
4
5 # Identificar os valores da rede de interesse, e.g.
6 # ESSID:"Nome_da_rede" -> nome da rede
7 # IE: IEEE 802.11i/WPA2 Version 1 -> encriptação
8 # E então é modificado o arquivo de configuração
9
10 # abre o arquivo para edição
11 sudo nano /etc/network/interfaces
12 # Após modificar o arquivo e salvá-lo:
13
14 sudo ifup wlan0 # ativa a interface de rede Wi-Fi
15 sudo ifdown eth0 # desativa a interface Ethernet

```

Código-fonte 4.1: Passos para configuração do Wi-Fi

No arquivo */etc/network/interfaces*, as configurações utilizadas entre as linhas 19 e 35 do código são referentes à configuração do Wi-Fi para configuração da BBB com endereço de IP estático 192.168.1.155, permitindo o acesso à plataforma remotamente pela internet e não somente dentro da intranet. Observe-se que o acesso a este IP da intranet se dá pelo IP externo 143.107.xxx.xxx, que é o endereço do departamento da engenharia elétrica da USP de São Carlos. Por motivos de segurança este será omitido no presente trabalho.

Depois de todo este procedimento, o dispositivo ainda não estava funcionando após a operação de *reboot*, portanto foi utilizado um *script* de reset do Wi-Fi executado durante o *boot*, cujas instruções de uso, obtidas em <https://learn.adafruit.com/setting-up-wifi-with-beaglebone-configuration>, estão descritas no código-fonte 4.2. A figura 4.15 indica sucesso de conexão via SSH usando uma máquina com sistema operacional Linux Ubuntu 14.04 LTS.

```

1 git clone https://github.com/adafruit/wifi-reset.git #
  download do script
2 cd wifi-reset/ #entra no diretório do script
3 chmod +x install.sh #permissão de execução para o script
4 sudo ./install.sh #executa o script

```

Código-fonte 4.2: Configuração para reset do Wi-Fi após o boot

```

leonardo@grabaDev:~/final_paper_tcc$ ssh debian@143.107.235.59 -p 8585
debian@143.107.235.59's password:
Linux beaglebone 3.8.13-bone59 #1 SMP Fri Jul 4 22:52:56 UTC 2014 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Apr 23 22:51:13 2016 from 191.189.99.112
debian@beaglebone:~$ 

```

Figura 4.15: Acesso à BBB via SSH

4.3.2 Data/hora

Para a configuração automática de data e hora da BBB, foi escolhido o uso do protocolo NTP – *Network Time Protocol* ou Protocolo de Tempo para Redes – cujo objetivo é sincronizar os relógios de dispositivos conectados a uma rede a partir de fontes precisas [47]. No Brasil, a confiabilidade deste protocolo é responsabilidade do Observatório Nacional (ON), responsável legal por garantir a Hora Legal Brasileira, em conjunto com o Núcleo de Informação e Coordenação do Ponto BR, administrador e operador do domínio ".br"[48, 49].

Primeiramente foi realizada a instalação do protocolo, seguida da configuração do arquivo */etc/ntp.conf* e da modificação do arquivo que indica a *timezone*, sendo este um link simbólico para o verdadeiro arquivo. Os comandos executados estão descritos no código-fonte 4.3:

```

1 sudo apt-get install ntp #instala o NTP
2 sudo nano /etc/ntp.conf #abre o arquivo de configuração para
   edição
3 sudo mv /etc/localtime /etc/localtime-old #backup do arquivo
   antigo da timezone
4 sudo ln -s /usr/share/zoneinfo/America/Sao_Paulo /etc/
   localtime #cria link simbólico para arquivo da nova
   timezone

```

Código-fonte 4.3: Instalação e configuração do NTP

Na edição do arquivo */etc/ntp.conf*, a única modificação feita é a substituição dos servidores de hora padrão para os servidores brasileiros, listados no endereço <http://www.pool.ntp.org/zone/br> – no final de cada endereço foi adicionada a instrução *iburst*, conforme descrito no website oficial do protocolo (<http://support.ntp.org/bin/view/Support/ConfiguringNTP>) e que, embora não esteja claro seu funcionamento, é essencial para que o horário seja corretamente configurado a cada *reboot*. O trecho de código-fonte 4.4 indica as modificações no arquivo */etc/ntp.conf* adequando aos servidores brasileiros:

```

1 # You do need to talk to an NTP server or two (or three).
2 #server ntp.your-provider.example
3
4 # pool.ntp.org maps to about 1000 low-stratum NTP servers.
   Your server will
5 # pick a different set every time it starts up. Please
   consider joining the
6 # pool: <http://www.pool.ntp.org/join.html>
7 server 0.br.pool.ntp.org iburst
8 server 1.br.pool.ntp.org iburst
9 server 2.br.pool.ntp.org iburst
10 server 3.br.pool.ntp.org iburst

```

Código-fonte 4.4: Servidores brasileiros do NTP

4.3.3 Sensor DS18B20

Uma vez que o sensor de temperatura DS18B20 já é suportado pelo Kernel e incluído como *driver* na distribuição padrão da BBB, somente foi necessário criar uma sobreposição de *device tree* indicando em que pino o sensor está ligado. Tanto o mux quanto o OCP do SoC precisaram ser configurados e, para a compilação do arquivo texto em binário, foi necessária a instalação do DTC, descrita no código-fonte 4.5:

```
1 sudo wget -c https://raw.githubusercontent.com/RobertCNelson/tools/
    master/pkgs/dtc.sh
2 sudo chmod +x dtc.sh
3 ./dtc.sh
```

Código-fonte 4.5: Instalação do DTC

O arquivo com a sobreposição da *device tree* é descrito na seção D.2, porém o código-fonte 4.6 apresenta um trecho relevante deste, no qual observa-se a configuração do multiplexador do SoC. Note-se que esta configuração significa que o *pull-up* interno de 10kda BBB está ativado, eliminando a necessidade de um resistor externo. O valor mínimo de *pull-up* segundo a folha de dados do sensor é de 4,7k, mas não foi observada perda de desempenho do mesmo na presente configuração.

```
1 bb_wl_pins: pinmux_bb_wl_pins {
2     pinctrl-single,pins = <0x70 0x37>;
3 }
```

Código-fonte 4.6: Fragmento de device tree para o DS18B20

A compilação da sobreposição é executada com o comando presente no trecho de código 4.7. Adicionar -00A0"ao nome do arquivo de saída DTBO é fundamental. Note-se também que o arquivo DTBO é copiado para o diretório no qual ficam todas as sobreposições de *device tree*. Para carregar a sobreposição no boot, é preciso adicionar a linha *echo wl > /sys/devices/bone_capemgr.*/slots* ao arquivo */etc/rc.local*.

```
1 sudo dtc -O dtb -o wl-00A0.dtbo -b 0 -@ wl.dts
2 sudo cp wl-00A0.dtbo /lib/firmware
```

Código-fonte 4.7: Compilação de *device tree*

Após o reboot e com o sensor conectado à BBB, é possível obter o valor da temperatura realizando a leitura do arquivo */sys/devices/wl_bus_master1/28-*wl_slave*, conforme ilustrado na figura 4.16. Observe-se que na figura ao invés do asterisco foi utilizado o identificador único deste sensor específico — quando é utilizado um asterisco, ele é substituído por todos os possíveis nomes de sensores, portanto é útil para leitura de diversos sensores de uma vez. Para saber quantos dispositivos com o protocolo *1-wire* estão conectados à

mesma linha e assim descobrir os números referentes a eles, é possível ler o arquivo `/sys/devices/w1_bus_master1/w1_master_slaves` ou então verificar o conteúdo do diretório `/sys/devices/w1_bus_master1/`.

```
debian@beaglebone:~$ cat /sys/devices/w1_bus_master1/28-000004ee8ded/w1_slave
c1 01 4b 46 7f ff 0f 10 38 : crc=38 YES
c1 01 4b 46 7f ff 0f 10 38 t=28062
```

Figura 4.16: Leitura do sensor DS18B20 pelo terminal

4.3.4 Webserver Apache

Embora o Apache venha instalado na distribuição padrão da imagem do Debian para a BBB, é preciso configurar a porta que ele usa para conexões, predefinida pelas configurações de rede do departamento de engenharia elétrica. A porta será omitida por segurança, mas na descrição do processo de configuração aqui proposta será usado o valor 8080, padrão para diversos protocolos de internet, inclusive o HTTP.

Há dois arquivos que devem ser editados para modificar as configurações do Apache. No arquivo `/etc/apache2/sites-enabled/000-default` a primeira linha do arquivo deve ser modificada para `<VirtualHost *:8080>` e no arquivo `/etc/apache2/ports.conf` as linhas `NameVirtualHost *:8080` e `Listen 8080` devem ser adicionadas ou modificadas, caso já existam. Para que as configurações tenham efeito, o servidor deve ser reiniciado com o comando descrito na caixa de código-fonte 4.8:

```
1 sudo service apache2 restart
```

Código-fonte 4.8: Reinicialização do servidor web Apache

4.4 Organização do diretório da aplicação

Dado o nível de complexidade a que chegou o desenvolvimento da aplicação de controle do sistema, foi necessário adotar um sistema de divisão de arquivos por função. Dentre os tipos de arquivos há principalmente códigos-fonte, imagens e registros. O objetivo desta seção é explicar a organização adotada.

Cabe primeiramente observar que há duas grandes frentes de projeto com relação a software: a aplicação do servidor e a aplicação do cliente. Uma vez que há momentos em que ambas estão entrelaçadas, foi decidido por não separar os arquivos em pastas *cliente* e *servidor*, mas somente classificar cada arquivo como pertencente principalmente a uma destas duas frentes.

A figura 4.17 apresenta a estrutura reduzida de diretórios do projeto, contendo somente os diretórios cujos arquivos foram desenvolvidos neste projeto, enquanto a tabela 4.2 lista todos os diretórios e arquivos relevantes ao projeto, assim como os descreve brevemente e os distingue entre *cliente* ou *servidor*.

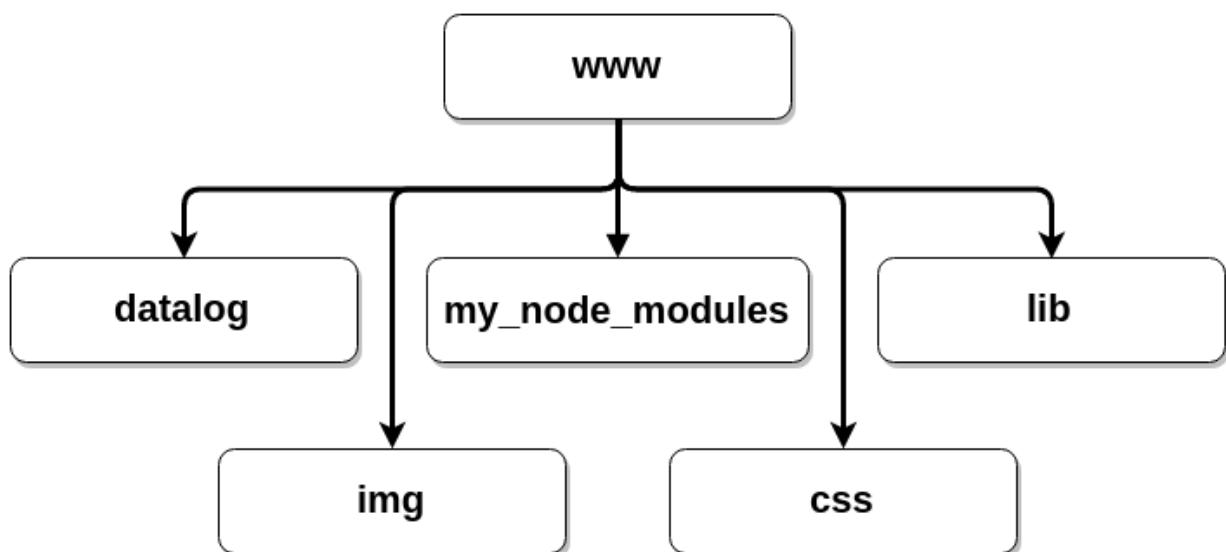


Figura 4.17: Estrutura de diretórios da aplicação da BBB

Tabela 4.2: Lista de diretórios e arquivos da aplicação da BBB

Diretório/arquivo	Descrição	Frente
www	diretório raiz	
about.php	página de descrição do projeto	cliente
control.php	GUI de controle do sistema	cliente
controle.js	arquivo raiz da aplicação do servidor	servidor
dyn_graph2.html	gráfico dinâmico de linha/área	cliente
index.html	página inicial - contém somente uma imagem clicável	cliente
listrecipe.php	gerenciador de receitas - permite criar, editar e excluir	cliente
newrecipe.php	GUI de edição de receitas	cliente
settings.php	GUI de configurações não referentes à brassagem	cliente
startrecipe.php	escolha da receita e início da brassagem	cliente
stats.php	estatísticas diversas de produções anteriores	cliente
www/css	configurações de estilo da GUI	
buttons.css	estiliza alguns botões da GUI	cliente
config.css	configurações gerais de estilo da GUI	cliente
form.css	estilo do formulário de edição de receitas	cliente
www/datalog	registros e backup de dados em memória não volátil	
backup.log	registro detalhado da última brassagem	servidor
default.csv	registro de temperaturas e tempos do DS18B20	servidor
instant.csv	última leitura do DS18B20 da MT	servidor
instant_bk.csv	última leitura do DS18B20 do BK	servidor
lockfile	arquivo que define se há uma brassagem em execução	servidor
www/img	imagens geradas para a GUI	
beer2.png	favicon	cliente
beer_compiler.png	imagem da página inicial da GUI	cliente
tplot.png	gráfico estático da temperatura em função do tempo	cliente
www/my_node_modules	módulos em Node.js desenvolvidos neste projeto	
ctrl.js	implementação do algoritmo de controle da brassagem	servidor
gpio_cfg.js	configuração e controle de GPIO e PWM	servidor
log_check_misc.js	verificação de integridade de receita. Leitura e registros de informações referentes à brassagem	servidor
routes.js	rotas do servidor Express.js	servidor
www/lib	módulos requisitados por códigos-fonte do cliente	
deletereceipt.php	deleta uma receita do servidor	servidor
header.js	auxiliar de criação do cabeçalho da GUI	cliente
header.php	criação do cabeçalho da GUI	cliente
listreceipt.php	lê conteúdo ou deleta receita do servidor	servidor
newreceipt.php	organiza e salva automaticamente a GUI de edição de receitas	cliente / servidor
newreceipt.php	salva ou carrega receita para a GUI de edição de receitas	servidor
previewreceipt.php	lê conteúdo essencial da receita e devolve para a GUI	cliente / servidor
log	registros de erros de aplicações da BBB	
recipes	receitas de cerveja	
node_modules	módulos do Node.js instalados por meio do NPM	
d3-3.4.10	biblioteca Javascript de visualização de dados	
rickshaw	API de geração de gráficos baseada na biblioteca D3.js	
tmp	diretório para testes e arquivos temporários	

4.5 IDE e sistema de controle de versão

Para facilitar o desenvolvimento e a documentação de *softwares*, duas ferramentas foram adotadas além do acesso ao terminal via SSH: *Cloud9*, cujo uso é descrito na seção 4.5.1 e *Git/GitHub*, na seção 4.5.2.

4.5.1 *Cloud9*

Cloud9 é um IDE do tipo SaaS, ou seja, um software que está na nuvem e cujo acesso é feito por meio de um navegador de internet. Embora o serviço padrão seja hospedado nos próprios servidores da *Cloud9 IDE Inc.*, uma SDK é disponibilizada, na qual está incluso o núcleo do projeto como código-aberto, livre para uso não comercial e desenvolvido em Node.js, possibilitando portanto a hospedagem um serviço do *Cloud9* localmente, que é o objetivo aqui proposto [50]. Esta é uma nítida vantagem para uma plataforma *headless* — sem periféricos do tipo HID — e com acesso à rede, a exemplo da BBB neste projeto, pois permite a programação direta no dispositivo por meio de uma interface gráfica via web.

Tanto o núcleo quanto as informações para instalação podem ser encontradas no repositório do projeto do *GitHub* (<https://github.com/c9/core>). A primeira tentativa de instalação realizado na BBB é apresentado na caixa de código-fonte 4.9:

```
1 cd ~/
2 git clone git://github.com/c9/core.git c9sdk
3 cd c9sdk
4 scripts/install-sdk.sh
```

Código-fonte 4.9: Primeira tentativa de instalação do *Cloud9 IDE core* na BBB

Ao tentar rodar a aplicação apontando para o diretório */var/www* como diretório de projeto, ou seja, o diretório raiz ao qual o *Cloud9* tem acesso, não foi possível acessar os arquivos uma vez que o diretório em si pertence ao usuário *www-data* e os arquivos pertencem a *root*.

Ao rodar a aplicação como administrador, outros erros ocorreram, o que levou à tentativa de instalar outro script presente no endereço <https://github.com/c9/install> do repositório do *Cloud9*, que permite a conexão da IDE a um servidor SSH. Ainda assim, os erros persistiram, o que levou à abertura de uma questão em <https://github.com/c9/core/issues/143>, na qual está descrito o processo adotado para solucionar o problema. Na caixa de código-fonte 4.10 é apresentado o processo de instalação executado com sucesso:

```

1 cd ~/
2 wget --no-check-certificate https://raw.githubusercontent.com
   /c9/install/master/install.sh
3 sudo bash install.sh

```

Código-fonte 4.10: Instalação bem sucedida do *Cloud9 IDE core* na BBB

Note-se que muitas tentativas foram executadas para obter o resultado final esperado, por isso em caso de tentativa de reprodução do processo de instalação aqui descrito é recomendado tentar usar somente o processo da caixa de código 4.10 e, em caso de falha, partir para a etapa descrita anteriormente. Ainda assim, outro processo de instalação mais direto e que deve ser tentado prioritariamente ao aqui descrito, mas que não foi testado, está descrito em <https://cloud9-sdk.readme.io/docs/running-the-sdk>.

Para rodar a aplicação no background e continuar rodando mesmo após desconectar da sessão SSH, é preciso executar o servidor do *Cloud9* como root usando o comando descrito na caixa 4.11. O parâmetro *-l* indica o endereço de IP do servidor, *-p* a porta, *-a* usuário e senha e *-w* o diretório de trabalho. Também são feitas redireções das saídas *stdout* e *stderr* para os arquivos */var/www/log/cl9.out* e */var/www/log/cl9.err*, respectivamente.

```

1 sudo su #precisa ser root
2 cd /home/debian #pasta na qual esta instalado o Cloud9
3 nohup node c9sdk/server.js -l 143.107.xxx.xxx -p xxxx -a
   usuario:senha -w /var/www/ > /var/www/log/cl9.out 2> /var/
   www/log/cl9.err < /dev/null &

```

Código-fonte 4.11: Execução do *Cloud9*

Ao realizar o acesso pelo browser usando o IP e porta selecionados, ou seja, digitando *IP:porta* na URL do navegador, o resultado é apresentado na figura 4.18. Note-se que o ambiente apresentado é um *preset* de IDE completa, mas que é completamente personalizável, além de existirem outros *presets*, dentre os quais um modo minimalista no qual somente o editor de texto ocupa a tela, bom para momentos de desenvolvimento intensivo de um arquivo específico.

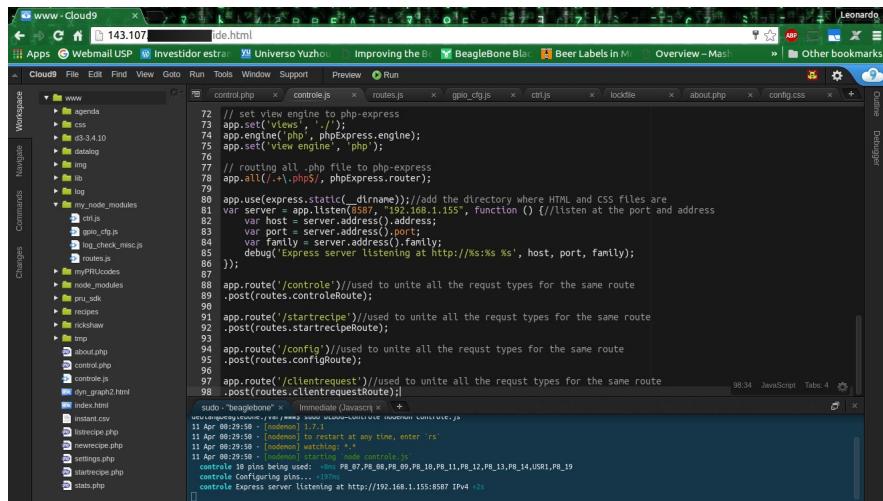


Figura 4.18: Ambiente completo do IDE *Cloud9*

4.5.2 *Git/GitHub*

Git é um DVCS (sistema de controle de versão distribuído) ou SCM (gerenciador de códigos-fonte), ou seja, uma ferramenta cuja finalidade é gerenciar o desenvolvimento de *software*. Um dos principais objetivos de um sistema como este é a documentação das mudanças feitas nos arquivos do projeto ao longo do seu desenvolvimento, o que permite facilmente reverter arquivos ou mesmo o projeto inteiro para um estado anterior, comparar mudanças em função do tempo, verificar quem foi a última pessoa a modificar um arquivo e possivelmente introduzir um bug, dentre outros recursos. E o mais importante, executar todas estas funções de maneira leve e fácil [51].

Esta ferramenta foi escolhida dentre muitas outras DVCS disponíveis por ser de código-aberto e extremamente popular, não somente pelas suas qualidades mas também pelo fato de ser a DCVS usada para o controle de versão do Kernel do Linux, sendo que o *Git* foi criado pela própria comunidade de manutenção do Kernel, incluindo Linus Torvalds [51]. Embora a referência bibliográfica consultada sobre este assunto possua mais de 500 páginas, demonstrando o nível de complexidade que a ferramenta pode atingir, neste projeto somente o essencial será abordado — o que significa compreender o uso de alguns comandos básicos.

Com relação ao *GitHub*, este nada mais é do que um serviço de hospedagem de repositórios *Git*. Uma de suas grandes vantagens é a popularidade, o que o faz ser uma fonte repleta de recursos de código-aberto. No que diz respeito a este trabalho, dentre suas vantagens estão o fato de o serviço facilitar o processo de *backup* de todo o trabalho realizado no âmbito de *software*, prover uma estrutura que permite a replicação do projeto com poucos coman-

dos e também o uso deste serviço para a redação da presente monografia sem a preocupação de ter que levar os arquivos fisicamente a diversas localidades ou depender de serviços de armazenamento de propósito geral, que não apresentam as vantagens de um DVCS.

Para configurar uma conta no github, o primeiro passo foi a realização do cadastro em <https://github.com>, seguida da criação de um repositório online, conforme ilustrado na figura 4.19. Após isto, o repositório foi clonado na BBB e os arquivos relevantes do sistema foram copiados para ele. Observe-se que esta abordagem foi improvisada devido ao fato de arquivos de configuração, dentre outros, encontrarem-se em repositórios específicos do sistema, já que o mais adequado seria que todos os arquivos do projeto fossem criados na pasta do *Git*. Para instalar, configurar o *Git* e clonar o repositório recém criado do *GitHub* para a BBB, foram executados os comandos descritos na caixa de código-fonte 4.12.

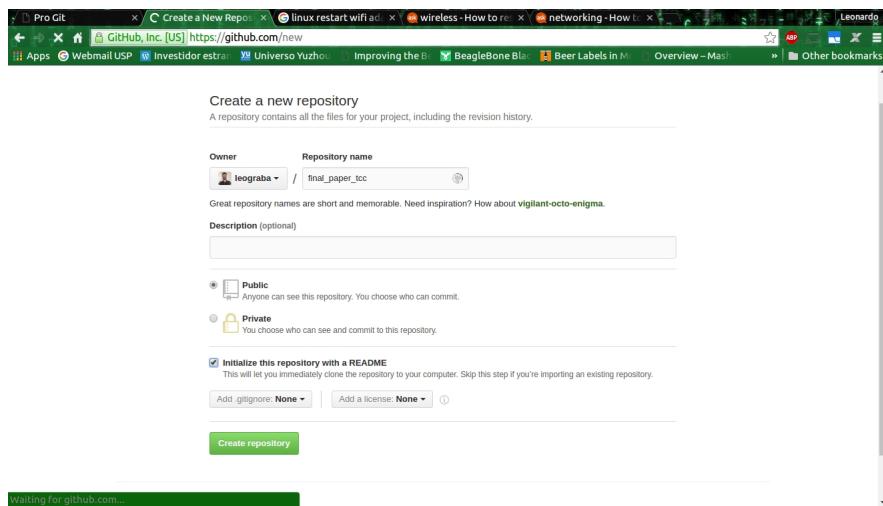


Figura 4.19: Criação de repositório no *GitHub*

```

1 sudo apt-get install git
2 git configure --global user.name "Nome"
3 git configure --global user.email email@exemplo.com
4 cd ~
5 git clone https://github.com/leograb/final_paper_tcc.git

```

Código-fonte 4.12: Instalação do *Git* e cópia do repositório do *GitHub* para a BBB

Após isto o diretório *final_paper_tcc* do *GitHub* foi clonado. Todos os arquivos relevantes ao projeto foram copiados para ele e, subsequentemente, adicionados ao *Git*: sempre que um arquivo é adicionado ou modificado em um repositório *Git*, ele é ignorado a menos que seja executado o comando *git add arquivo* — isto permite flexibilidade, principalmente com relação a arquivos que não devem ser adicionados ao repositório, como arquivos auxiliares gerados por compiladores, por exemplo. Após cada mudança, para torná-la efetiva é preciso executar o comando *git commit -m "mensagem para registro"*, que além da data e hora da

modificação, permite adicionar uma mensagem para identificar o que foi modificado desde o último commit. Por fim, foi preciso enviar as mudanças do repositório local para o *GitHub*. O processo de adicionar arquivos novos/modificados, fazer *commit* nas mudanças e atualizar a origem remota está descrito no código-fonte 4.13 e foi usado sempre que mudanças foram feitas ao projeto, desde implementações de funcionalidades a correções de *bugs*. Um exemplo de *commit* após mudanças incrementais é apresentado na figura 4.20.

```
1 git add .
2 git commit "Primeiro commit"
3 git push origin master
```

Código-fonte 4.13: Primeiro *git commit*

```
debian@beaglebone:~/final_paper_tcc$ git status
# On branch develop
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   www/controle.js
#
no changes added to commit (use "git add" and/or "git commit -a")
debian@beaglebone:~/final_paper_tcc$ git add www/controle.js
debian@beaglebone:~/final_paper_tcc$ git commit -m "Melhorias incrementais ao controle da brassagem"
[develop d149273] Melhorias incrementais ao controle da brassagem
 1 file changed, 2 insertions(+), 2 deletions(-)
debian@beaglebone:~/final_paper_tcc$ git push origin develop
Username for 'https://github.com': leogveiga@gmail.com
Password for 'https://leogveiga@gmail.com@github.com':
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 397 bytes, done.
Total 4 (delta 3), reused 0 (delta 0)
To https://github.com/leogveiga/final_paper_tcc.git
  deae892..d149273  develop -> develop
debian@beaglebone:~/final_paper_tcc$
```

Figura 4.20: Fazendo *git commit* após mudanças incrementais

4.6 Geração de gráfico e registro de temperatura em Python

Para o armazenamento da temperatura do sensor em função do tempo, assim como a geração de um gráfico com o histórico de temperaturas para o usuário do sistema, foram escritos dois *scripts* em Python, sendo a saída do *script* de *log* a entrada do *script* de geração do gráfico — embora ambos possam ser executados independentemente, a geração do gráfico precisa de um arquivo contendo o histórico de temperaturas.

4.6.1 Registro de temperatura

Para o arquivo do *log*, foram primeiramente escritas algumas funções:

- **tread()** — lê o arquivo referente ao sensor de temperatura e retorna deste somente a temperatura formatada em graus celsius.

- **tprint_all(tcelsius)** — recebe temperatura em °C e imprime com formatação HTML em °C, °F, K e °R; pode ser usada em conjunto com a função tread()
- **tprint_all_terminal(tcelsius)** — idêntica à função tprint_all(), porém imprime com formatação para o terminal do Linux
- **tlog(file = "/var/www/default.csv")** — cria/adiciona a um arquivo de log no formato CSV a temperatura medida e a data/hora correspondente no formato *Unix epoch* — que é definido como o número de segundos passados desde 1 de janeiro de 1970 não considerando segundos bissextos — em intervalo de amostragem definido na função, até que o script seja interrompido de alguma maneira. Em função de erros de leitura esporádicos do sensor, se a temperatura lida for menor ou igual a zero, esta é descartada. Imprime a temperatura lida a cada amostragem. O tempo de amostragem é definido pela soma do tempo de leitura do arquivo (inerente ao sistema) com o tempo ocioso definido nesta função: para que o tempo de amostragem seja o mais próximo possível de 1s, foi realizado um estudo de caso, descrito no apêndice F, e com isso obteve-se o valor de 0,2202s ocioso definido nesta função. Além disto, nesta função é chamada *tlog_instant* que registra em um arquivo separado a última leitura válida de temperatura e tempo.
- **tlog_instant(temperature, epoch, file = "/var/www/datalog/instant.csv")** — registra a última leitura válida de temperatura e tempo em um arquivo, criando ou sobrescrevendo o mesmo.
- **tlog_test(file = "/var/www/datalog/default.csv")** — função que gera um arquivo de *log* com rampas e degraus de temperatura, para uso em simulações posteriores do funcionamento do sistema, descritas na seção 4.7.5.

Sendo a função *tlog* a mais relevante, esta pode ser obtida na caixa de código-fonte 4.14. O script completo está registrado no código-fonte B.1 do apêndice B. Ainda assim, as funções por si só não são executadas sozinhas, portanto foi criado um script auxiliar em Python que chama a função *tlog*, descrito no código-fonte 4.15.

```

1 def tlog(file = "/var/www/datalog/default.csv"):
2     """salva temperatura e Unix Time em .csv"""
3     tsample = 0.2202 #amostra a cada ~1 segundo
4     buff_temp = tread()#guarda o último valor lido
5     exist = os.path.isfile(file)
6     with open(file, 'a', 1) as log:
7         if exist is False:#se vai criar o arquivo agora
8             log.write("temperatura,data\n")
9         while True: #loop infinito
10            temp_celsius = tread()
11            epoch = time.time()#lê a data/hora do sistema
12            if temp_celsius >= 0:#evita leitura errada

```

```

13     log.write("%f,%f\n" % (temp_celsius, epoch))
14     tlog_instant(temp_celsius, epoch)
15     time.sleep(tsample) #espera n segundos
16     print "registrando temperatura!", temp_celsius

```

Código-fonte 4.14: Função de *log* da temperatura em Python

```

1 import temp
2 temp.tlog()
3 # para executar no terminal, basta executar:
4 # sudo python log.py

```

Código-fonte 4.15: Script para gravação das leituras de temperatura em Python

Na figura 4.21 é apresentada uma amostra de arquivo CSV gerado pelo script 4.15, e que valida o funcionamento da aplicação.

```

temperatura,data
20.200000,1404790916.416406
20.400000,1404790917.429680
20.600000,1404790918.437585
20.800000,1404790919.444913
21.000000,1404790920.452805
21.200000,1404790921.456841
"default.csv" [readonly] 12921 lines, 361934 characters

```

Figura 4.21: Arquivo CSV com registro de temperaturas gerado em Python

4.6.2 Gráfico de temperatura

Com o script que gera um registro da temperatura medida pelo sensor DS18B20 em função do tempo, o próximo passo é a análise dos dados coletados. Uma forma fazê-lo é plotando um gráfico. Para fazê-lo em Python, foi decidido usar as bibliotecas *Matplotlib* e *NumPy*, integrantes do ecossistema *SciPy*.

Matplotlib é uma biblioteca desenvolvida especificamente para a plotagem 2D de figuras estáticas em scripts Python, no Python shell (similar ao ambiente interpretador de softwares proprietários como MATLAB e Mathematica), em aplicações de servidores e *toolkits* com GUI [52]. Para instalar esta biblioteca, basta executar o comando descrito na caixa de código-fonte 4.16.

```
1 sudo apt-get install python-matplotlib
```

Código-fonte 4.16: Instalação da biblioteca Matplotlib

NumPy é um pacote em Python que fornece suporte a arranjos e matrizes multidimensionais, além de diversas funções matemáticas de suporte a estes formatos de dados. É amplamente empregado nas áreas de computação científica, engenharia e matemática [53]. Este

pacote é parte integrante do ecossistema *SciPy*, cuja instalação é apresentada na caixa de código-fonte 4.17.

```
1 sudo apt-get install python-scipy
```

Código-fonte 4.17: Instalação do ecossistema SciPy

Desenvolvimento do gráfico

Com todas as ferramentas necessárias instaladas, pôde ser iniciado o desenvolvimento do script que gera o gráfico da temperatura em função do tempo. A abordagem inicial foi criar um script que gera um arquivo PDF do gráfico a partir do exemplo encontrado na página http://matplotlib.org/examples/pylab_examples/multipage_pdf.html, modificando este script para as necessidades do projeto.

Como o desenvolvimento foi realizado sem uma interface gráfica, referida como *backend*, foi preciso explicitar a saída como um renderizador no lugar deste, conforme indicado no código-fonte 4.18, que documenta a importação das bibliotecas/funções utilizadas no script. Note-se que a mudança da saída deve ocorrer antes da importação da biblioteca *numpy* para que não ocorra erro.

```
1 import time
2 import matplotlib
3 matplotlib.use('AGG') #muda a output para um renderer ao inves
        de backend
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from scipy import interpolate
```

Código-fonte 4.18: Importação das bibliotecas para geração de gráfico em Python

Na única função do script, *graph_gen(file = '/var/www/default.csv', graph = '/var/www/t-plot.png')*, é explicitado que a entrada é um arquivo CSV e a saída um arquivo PNG. Nesta função o arquivo CSV é lido linha por linha em um *loop*, sendo a primeira linha do arquivo ignorada em função de ela não conter dados, conforme ilustrado na figura 4.21. Ao longo das iterações deste *loop*, a data/hora do primeiro e último registro de temperatura são usadas para gerar um título para o gráfico e, a cada iteração os dados são decodificados, o que se resume basicamente a separar o que há antes e depois da vírgula presente na linha lida do arquivo CSV.

Na parte da função referente à plotagem do gráfico, além de configurações triviais como cor de fundo, tamanho de fontes e legendas, dentre outros, é notável o ajuste da escala do tempo, para o qual foi pensado um registro de temperatura contínuo e não somente o registro

de uma produção de cerveja: para até duas horas de registro a escala é impressa em minutos; entre duas horas e um dia e meio de registros, em horas; entre um dia e meio e três meses de registros, em dias; e acima disto em meses. Após isto, o gráfico é plotado, salvo no formato PNG com densidade de pontos por polegada de 150dpi, e fechado para liberar a memória do sistema.

O script que chama a função *graph_gen* é descrito no código-fonte ??, presente no mesmo arquivo em que está a função. Note-se que o gráfico é gerado periodicamente a cada 300 segundos e o tempo necessário para executar tal tarefa é computado e impresso na saída padrão do Python. O arquivo completo que contém a função *graph_gen* está documentado no código-fonte B.2 do apêndice B.

Na figura 4.22 é apresentado um gráfico gerado na BBB a partir de 7200 pontos amostrados, ou seja, 120 minutos. Este tempo foi escolhido uma vez que dificilmente o cozimento do mosto ou a fervura dura mais do que isso, portanto provando que a implementação é factível para o uso a que se destina. A tabela 4.3 apresenta informações estatísticas sobre o tempo de computação necessário para a geração do gráfico. A figura 4.23 apresenta o processo de verificação do número de pontos registrados e o posterior processo de coleta dos dados que deram origem à tabela 4.3.

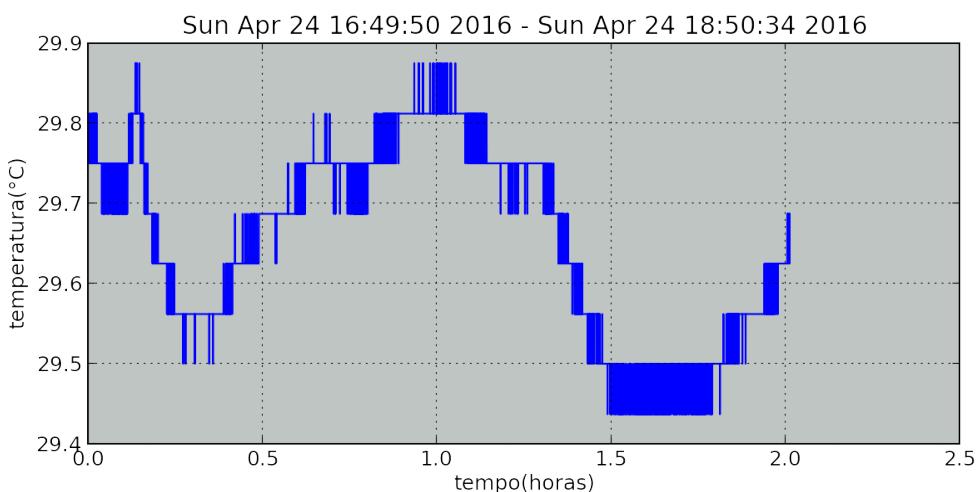


Figura 4.22: Gráfico com 7200 pontos gerado em Python

Tabela 4.3: Estatísticas referentes ao tempo de geração de gráfico na BBB, para 7200 pontos em Python

Descrição	Valor
N.º de amostras	30
Média (s)	2,106
Máximo (s)	2,068
Mínimo (s)	2,149
Desvio padrão (s)	0,023

```
debian@beaglebone:~/brewing$ wc -l /var/www/datalog/default.csv
7201 /var/www/datalog/default.csv
debian@beaglebone:~/brewing$ sudo python grafico.py
tempo para gerar grafico (s) = 2.127
tempo para gerar grafico (s) = 2.068
tempo para gerar grafico (s) = 2.136
tempo para gerar grafico (s) = 2.077
tempo para gerar grafico (s) = 2.116
tempo para gerar grafico (s) = 2.083
tempo para gerar grafico (s) = 2.085
tempo para gerar grafico (s) = 2.126
tempo para gerar grafico (s) = 2.118
tempo para gerar grafico (s) = 2.117
tempo para gerar grafico (s) = 2.085
tempo para gerar grafico (s) = 2.099
tempo para gerar grafico (s) = 2.113
tempo para gerar grafico (s) = 2.087
tempo para gerar grafico (s) = 2.115
tempo para gerar grafico (s) = 2.090
tempo para gerar grafico (s) = 2.138
tempo para gerar grafico (s) = 2.091
tempo para gerar grafico (s) = 2.094
tempo para gerar grafico (s) = 2.126
tempo para gerar grafico (s) = 2.095
tempo para gerar grafico (s) = 2.139
tempo para gerar grafico (s) = 2.080
tempo para gerar grafico (s) = 2.149
tempo para gerar grafico (s) = 2.089
tempo para gerar grafico (s) = 2.093
tempo para gerar grafico (s) = 2.145
tempo para gerar grafico (s) = 2.095
tempo para gerar grafico (s) = 2.120
tempo para gerar grafico (s) = 2.082
```

Figura 4.23: Geração de gráfico em Python múltiplas vezes para obter o tempo médio

4.7 Aplicação *server-side* em Node.js

Nesta seção será detalhado o uso da linguagem de programação Javascript no ambiente de interpretação Node.js. O escopo da aplicação desta linguagem no projeto se aplica, mas não é limitado a: acesso ao *hardware*, incluindo barramentos de propósito geral (GPIO) e periféricos como PWM; criação de um servidor web minimalista; e controle do processo de produção de cerveja. Node.js já vem instalado na distribuição padrão do Debian utilizada neste projeto, porém é possível instalar ou atualizar o ambiente por meio do gerenciador de pacotes *apt-get*, conforme descrito na caixa de código-fonte 4.19:

```
1 | sudo apt-get install nodejs
```

Código-fonte 4.19: Instalação e/ou atualização do Node.js

Com isto, o gerenciador de pacotes do Node chamado NPM (Node Package Manager) também é instalado no sistema. Neste projeto, o desenvolvimento em Node foi legado do desenvolvimento prévio em PHP e, por isto, tudo foi realizado no diretório `/var/www`, cujo dono é o usuário `www-data`. Em função disto, foi criada manualmente a pasta `/var/www/node_modules` e o dono foi trocado para o usuário `debian` (enquanto sob condições normais este processo seria automático). Isto foi feito por questões de segurança, uma vez que os módulos do Node são autocontidos no diretório do projeto, o que desencoraja a instalação global na maioria dos casos. O processo de criação do diretório é descrito no trecho de código 4.20:

```
1 cd /var/www
2 sudo mkdir node_modules
3 sudo chown debian:debian node_modules
```

Código-fonte 4.20: Criação do diretório `/var/www/node_modules`

Além disto, foi instalado o módulo `Debug` do Node, que ajuda no desenvolvimento da aplicação da seguinte maneira: ele encapsula a função `console.log`, análoga ao `printf` da linguagem C, de tal maneira que as mensagens só são impressas na tela se uma variável do terminal estiver configurada durante a execução do código. Além disto, diferentes grupos de mensagens de `debug` podem ser configurados, permitindo flexibilidade e também é adotado um esquema de cores que facilita a compreensão das mensagens de `debug`. Para ilustrar o uso deste módulo, na caixa 4.21 é descrita a instalação do módulo; o trecho de código-fonte 4.22 descreve como usar o módulo em uma aplicação de Node e a figura 4.24 apresenta o resultado do uso do módulo no exemplo, para as seguintes situações: não imprime mensagens de `debug`; imprime grupos de mensagens de `debug` e; imprime todas as mensagens de `debug` disponíveis.

```
1 npm install debug
```

Código-fonte 4.21: Instalação do módulo `Debug`

```
1 "use strict";
2
3 var dbg_gpio = require('debug')('gpio');
4 var dbg_pwm = require('debug')('pwm');
5
6 console.log("Inicio exemplo:");
7 for(var i = 0; i < 3; i++) {
8     switch(i) {
9         case 0:
10             dbg_gpio("Debug situacao gpio");
11             break;
```

```

12   case 1:
13     dbg_pwm("Debug situacao pwm");
14     break;
15   case 2:
16     dbg_gpio("Debug ambos gpio");
17     dbg_pwm("Debug ambos pwm");
18     break;
19   }
20 }
```

Código-fonte 4.22: Exemplo de uso do módulo *Debug*

```

leonardo@grabaDev:~/final_paper_tcc/tmp$ node testenode.js
Inicio exemplo:
leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=gpio node testenode.js
Inicio exemplo:
  gpio Debug situacao gpio +0ms
  gpio Debug ambos gpio +2ms
leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=pwm node testenode.js
Inicio exemplo:
  pwm Debug situacao pwm +0ms
  pwm Debug ambos pwm +2ms
leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=gpio,pwm node testenode.js
Inicio exemplo:
  gpio Debug situacao gpio +0ms
  pwm Debug situacao pwm +3ms
  gpio Debug ambos gpio +1ms
  pwm Debug ambos pwm +0ms
leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=* node testenode.js
Inicio exemplo:
  gpio Debug situacao gpio +0ms
  pwm Debug situacao pwm +3ms
  gpio Debug ambos gpio +2ms
  pwm Debug ambos pwm +0ms
```

Figura 4.24: Exemplo de uso do módulo *Debug*

Neste capítulo deve-se assumir que todas as operações foram realizadas no diretório */var/www*, salvo indicações em contrário. Note-se também que, no início de todos os códigos-fonte completos, presentes no apêndice C, é usada a declaração "**use-strict**", que evita o uso de potenciais armadilhas do Javascript, como usar inadvertidamente uma variável global sem declará-la, dentre outras armadilhas.

4.7.1 Acesso a GPIO e PWM

O acesso às características de *hardware* da BBB necessárias para o controle das válvulas, bombas, resistores de potência e servo-motor do sistema é feito por meio do módulo de código-aberto *Octalbonescript*, que consiste de um *fork* (ou derivação) do módulo oficial da BBB de acesso a hardware *Bonescript*. Sua escolha se deve ao fato de que esta biblioteca

era mais completa e estável durante o desenvolvimento deste projeto. A instalação desta biblioteca está descrita na caixa 4.23. A documentação da API deste módulo foi obtida em <https://github.com/theoctal/octalbonescript/wiki>.

```
1 npm install octalbonescript
```

Código-fonte 4.23: Instalação do Octalbonescript

Uma vez que este módulo compila uma sobreposição de *device tree* a partir de um template fornecido junto com a instalação, foi necessário modificá-lo para acomodar a descrição de hardware do sensor DS18B20. Para tal, foi investigado que o comportamento da biblioteca *Octalbonescript* com relação à *device tree* é o seguinte: se a sobreposição já está carregada não é feita nova compilação do fragmento nem tentativa de carregar a sobreposição, portanto foi necessário resetar a BBB toda vez que novos testes eram realizados. O template fica no diretório *node_modules/octalbonescript/dts/* e o nome do arquivo modificado é *OBS_UNIV_template.dts*. As modificações realizadas, assim como as linhas do código modificadas estão descritas na caixa de código-fonte 4.24. Linhas que começam com "X" foram apagadas, as outras foram adicionadas.

```
1 [389]          /* P9_11 (THE ONE-WIRE DS18B20) */
2 [390]          bb_w1_pins: pinmux_bb_w1_pins {
3 [391]              pinctrl-single,pins = <0x070 0x37>; };
4 X[392]          /* Mode 7, Pull-Up, RxActive */
5 X[393]          P9_11_gpio_pin: pinmux_P9_11_gpio_pin {
6 X[394]              pinctrl-single,pins = <0x070 0x2F>; };
7 X[395]          /* Mode 7, RxActive */
8 X[396]          P9_11_gpio_pu_pin: pinmux_P9_11_gpio_pu_pin {
9 X[397]              pinctrl-single,pins = <0x070 0x37>; };
10 X[398]          /* Mode 7, Pull-Up, RxActive */
11 X[399]          P9_11_gpio_pd_pin: pinmux_P9_11_gpio_pd_pin {
12 X[400]              pinctrl-single,pins = <0x070 0x27>; };
13 X[401]          /* Mode 7, Pull-Down, RxActive */
14 X[402]          P9_11_uart_pin: pinmux_P9_11_uart_pin {
15 X[403]              pinctrl-single,pins = <0x070 0x36>; };
16 X[404]          /* Mode 6, Pull-Up, RxActive */
17 X[405]          onewire@0 {
18 X[406]              compatible = "w1-gpio";
19 X[407]              status = "okay";
20 X[408]              pinctrl-names = "default";
21 X[409]              pinctrl-0 = <&bb_w1_pins>;
22 X[410]              pinctrl-1 = <&P9_11_gpio_pin>;    gpios = <&
23 X[411]                  gpio1 30 0>;
24 X[412]              pinctrl-2 = <&P9_11_gpio_pu_pin>;
25 X[413]              pinctrl-3 = <&P9_11_gpio_pd_pin>;
26 X[414]              pinctrl-4 = <&P9_11_uart_pin>;
27 X[415]          };
28 X[416]          P9_11 {
29 X[417]              gpio-name = "P9_11";
30 X[418]              gpio = <&gpio1 30 0>;
31 X[419]              input;
32 X[420]              dir-changeable;
```

```
27 | X [1237]      };
```

Código-fonte 4.24: Modificações feitas ao template de *device tree* do Octalbonescript

Com esta configuração da *device tree*, foi preciso parar de carregar o fragmento descrito na seção 4.3.3, pois ao tentar usar o *Octalbonescript* passou a ocorrer um erro do tipo **EEXIST: file already exists**, que indica em um âmbito geral que um arquivo que deveria ser criado já existe.

A partir deste ponto, foi desenvolvido um módulo nomeado *gpio_config* responsável pelo gerenciamento de GPIO e PWM, baseado no Octalbonescript. Foram criados objetos para cada elemento do sistema, usados para guardar a identificação do pino de GPIO de cada objeto, o estado e a configuração de cada pino. Para isto, uma função auxiliar foi criada. Também foi criado um objeto que reúne todos os objetos dos pinos, para facilitar a configuração inicial e também manter controle sobre o status do sistema durante seu funcionamento. Uma vez que em Node.js as variáveis são todas ponteiros, ao atribuir uma variável a outra o que se está fazendo é apontar as duas variáveis para o mesmo endereço de memória. Na caixa de código-fonte 4.25 é apresentada a função de criação dos objetos, a declaração dos elementos aquecedores e a união destes elementos, a título de exemplo. A descrição completa das declarações de objetos pode ser encontrada no código-fonte C.2 no apêndice C.

```
1  var mash_heat = module.exports.mash_heat = new PinObjectIO("P8_13");
2  var boil_heat = module.exports.boil_heat = new PinObjectIO("P8_14");
3  var heaters = module.exports.heaters = collect(mash_heat,
4                                              boil_heat);
5
6  // I/O pins
7  function PinObjectIO(pinId){ //function to create pin object,
8    should recieve pin ID
9    if(pinId){ //if the variable is passed to function or not
10      empty
11      this.id = pinId; this.state = b.LOW; this.cfg = b.OUTPUT;
12    }
13  }
14
15 function collect(){ //function concat objects
16  var ret = {};//the new object
17  var len = arguments.length;//the total number of objects
18  passed to collect
19  for (var i=0; i<len; i++) { //do it for every object passed
20    for (var p in arguments[i]) { //iterate the i-eth object
21      passed
22      if (arguments[i].hasOwnProperty(p)) { //whenever there
23        is a property
```

```

21         ret[p] = arguments[i][p]; //add the property to the
22         new object
23     }
24 }
25 return ret;
26 }
```

Código-fonte 4.25: Criação de objetos para os elementos do sistema em Node

Além disto, foi criado um objeto que guarda o status do sistema: o número total de pinos a serem configurados, o número de pinos configurados corretamente, o número de pinos como GPIO, PWM, analógicos e interrupções. Quanto às funções do módulo, estas são:

- **changeStatusIO(pin, val)** — recebe o nome do pino, e.g. *mash_heat*, e o valor (*true* ou *false* para GPIO e numérico para PWM). Ajusta o valor do pino conforme passado para a função, assim como trata de atualizar as variáveis de controle dos pinos.
- **getSystemStatus()** — retorna o valor/estado de todos os pinos do sistema, sendo 0 ou 1 para GPIO e numérico para PWM.
- **pinsConfiguration()** — configura todos os pinos declarados como entrada, saída ou PWM. Em caso de erro de configuração de algum pino é impressa uma mensagem com o auxílio do módulo *debug*, portanto cabe ao administrador do sistema testar se a configuração está funcionando no momento da instalação do equipamento ou quando julgar necessário.
- **ioTest()** — função de teste. Implementa um algoritmo que seta os pinos de GPIO do sistema de tal forma sequencial, ou seja, chamando esta função em um *loop* faz com que um efeito cascata seja produzido, caso os pinos estejam ligados a LEDs.

Na função *pinsConfiguration*, é preciso notar que dentro do laço de repetição *for* é chamada uma função anônima para a configuração de cada pino. Isto é necessário pois, se todas as operações fossem realizadas diretamente dentro do laço, o comportamento assíncrono do Node geraria inconsistências na execução do código. Assim sendo, o objetivo da função anônima é criar um escopo que permite a execução assíncrona do código sem que a referência passada para esta função anônima se degenerasse. Uma vez que esta função precisa receber o valor e não o ponteiro da variável de iteração do laço, tanto o pino a ser configurado quanto o seu índice (variável de iteração) no laço de repetição são passados com o auxílio do método *call()*, que neste caso copia o pino como o elemento *this* e o índice como o primeiro argumento da função, criando um novo escopo por meio de uma clausura. Sem isso, antes

que a primeira invocação da função anônima acabasse, o valor de `i`, por exemplo, já seria `all_io_objects.length` e não zero, que seria o valor esperado.

Para tornar a compreensão desta necessidade mais clara, a caixa de código-fonte 4.26 apresenta um exemplo no qual duas funções com um laço `for` são chamadas: a primeira delas imprime o valor da variável de iteração para todas as chamadas do laço imediatamente e com um atraso de 100ms, mas sem precauções com relação ao comportamento assíncrono do Node; já a segunda função utiliza o método descrito no parágrafo anterior. Observa-se na figura 4.25 que sem tomar precauções, na chamada com atraso, é impresso o último valor da variável de iteração do laço em todas as iterações. Isto deixa claro a necessidade de criar um novo escopo, assim como demonstra que esta é uma fonte de erros para programadores inexperientes e/ou descuidados.

```

1  'use-strict';
2  var debug1 = require('debug')('debug1');
3  var debug2 = require('debug')('debug2');
4
5  function pinConfiguration() {
6      for(var i = 0; i < 3; i++) {
7          setTimeout(function() { debug1('atraso: ' + i
8              ); }, 100);
9          debug1('imediato: ' + i);
10     }
11 }
12
13 function pinConfiguration2() {
14     for(var i = 0; i < 3; i++) {
15         (function () {
16             var i2 = this;
17             setTimeout(function() { debug2('
18                 atraso: ' + i2); }, 100);
19         }).call(i);
20         debug2('imediato: ' + i);
21     }
22 }
23 pinConfiguration2();
24 pinConfiguration();
```

Código-fonte 4.26: Uso de clausura para criação de escopo

```
leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=* node testeclosure.js
debug2 imediato: 0 +0ms
debug2 imediato: 1 +3ms
debug2 imediato: 2 +2ms
debug1 imediato: 0 +0ms
debug1 imediato: 1 +1ms
debug1 imediato: 2 +0ms
debug2 atraso: 0 +96ms
debug2 atraso: 1 +2ms
debug2 atraso: 2 +0ms
debug1 atraso: 3 +2ms
debug1 atraso: 3 +0ms
debug1 atraso: 3 +0ms
```

Figura 4.25: Resultado de chamada de função sem criação de escopo x com criação de escopo

4.7.2 Servidor Express.js

Mesmo com o servidor Apache já configurado na BBB, quando foi decidido o uso de Node.js como linguagem do servidor o Apache foi substituído pelo *Framework* Express, dados os benefícios apontados na seção 2.4. Para isto, a consulta à documentação da API deste framework (<http://expressjs.com/pt-br/4x/api.html>) foi a base de desenvolvimento do servidor.

Outros dois módulos auxiliares foram empregados em conjunto com o Express: o módulo *Body-parser*, cuja função é a interpretação de dados codificados em JSON pelo cliente; e o módulo *PHP-express*, que permite a interpretação de arquivos PHP pelo servidor. Este último módulo foi empregado devido a algumas funcionalidades desenvolvidas em PHP em uma fase inicial do projeto, portanto suportando a aplicação legada. Reescrever os códigos PHP é uma opção, porém em função do tempo necessário para tal tarefa, esta foi a solução adotada. Na caixa 4.27 é documentada a instalação dos módulos.

```
1 npm install express
2 npm install body-parser
3 npm install php-express
```

Código-fonte 4.27: Instalação dos módulos necessários para implementar o servidor web em Node.js

O código-fonte no qual é configurado o servidor é o */var/www/controle.js*, que é também o arquivo raiz da aplicação em Node, e está disponível no apêndice C.1. É interessante observar a configuração do PHP-express, descrita no trecho de código-fonte 4.28, no qual observa-se que logo ao adicionar o módulo ao código é passado o caminho do PHP, que deve ser previamente instalado no sistema. Também é notável o fato de que os arquivos PHP são identificados por meio de uma expressão regular, cuja função à qual ela é aplicada direciona estes arquivos para o *engine* do PHP.

```

1 var phpExpress = require('php-express')({ // assumes php is
2   in your PATH
3   binPath: 'php'
4 });
5 // set view engine to php-express
6 app.set('views', './');
7 app.engine('php', phpExpress.engine);
8 app.set('view engine', 'php');
9
10 // routing all .php file to php-express
11 app.all(/.+\.php$/, phpExpress.router);

```

Código-fonte 4.28: Configuração do PHP-express

Com relação à configuração do Express, três pontos devem ser notados: o diretório no qual é executada a aplicação é configurado como o diretório raiz do servidor, seja para servir arquivos HTML e CSS ou arquivos de mídia e apoio da aplicação; o endereço de IP e porta são especificados de modo a permitir acesso à BBB pela internet e; são configuradas rotas para diferentes categorias de requisições do cliente. O fragmento de código-fonte 4.29 apresenta os três pontos discutidos neste parágrafo. Note-se que é adicionado o módulo *routes*, que foi escrito separadamente somente para facilitar a compreensão do código. Ele define o comportamento das rotas.

```

1 var express = require("express");
2 var app = express();
3 var routes = require('./my_node_modules/routes.js');
4
5 app.use(express.static(__dirname)); //add the directory where
6 // HTML and CSS files are
7 var server = app.listen(8587, "192.168.1.155", function () {
8   //listen at the port and address
9   var host = server.address().address;
10  var port = server.address().port;
11  var family = server.address().family;
12  debug('Express server listening at http://%s:%s %s', host,
13    port, family);
14});
15
16 app.route('/controle') //used to unite all the request types
17 // for the same route
18 .post(routes.controlRoute);
19
20 app.route('/startrecipe') //used to unite all the request types
21 // for the same route
22 .post(routes.startrecipeRoute);
23
24 app.route('/config') //used to unite all the request types for
25 // the same route
26 .post(routes.configRoute);
27
28 app.route('/clientrequest') //used to unite all the request
29 // types for the same route

```

23 | .post(routes.clientrequestRoute);

Código-fonte 4.29: Configuração do servidor web com o framework Express.js

Quanto ao módulo que define o comportamento das rotas do servidor, seu código-fonte é apresentado na caixa de código-fonte do apêndice C.3. Ele consiste de quatro funções — uma para cada rota:

- **controleRoute(req, res)** — realiza o controle direto de GPIO e PWM. Faz uso do módulo *gpio_cfg*, descrito na seção 4.7.1.
 - Muda o valor de um terminal, conforme especificado pelo cliente.
 - Retorna para o cliente o status de todos os pinos da aplicação, independente da sua configuração.
- **startRecipeRoute(req, res)** — executa diversas funções relacionadas à produção de cerveja, inclusive iniciar o processo de controle da mesma, conforme a requisição enviada pelo cliente.
 - Retorna a lista de receitas cadastradas no servidor.
 - Verifica integridade da receita, ou seja, se ela contém informações suficientes para que uma produção seja iniciada.
 - Realiza uma série de verificações e começa uma brassagem, se aprovado.
 - Verifica se já existe alguma receita em andamento.
 - Retorna erro para o cliente se nenhuma requisição é válida.
- **configRoute(req, res)** — ajusta ou responde com a data e hora da BBB, conforme a solicitação do cliente.
- **clientrequestRoute(req, res)** — função auxiliar do processo de mostura, usada para sinalizar que um novo degrau de temperatura deve começar.

Dentre as particularidades de implementação deste módulo, observa-se o uso dos argumentos *req* e *res* em todas as funções, uma vez que estes são padronizado pela API do Express para requisições do tipo POST feitas pelo cliente. O argumento *req* contém todos os dados enviados pelo cliente no padrão HTTP POST, que é parte da transferência de estado representacional, conhecida como REST. Uma vez que o módulo Express está em conformidade com as restrições REST, ele é considerado um serviço *RESTful*. Quanto ao argumento *res*, este contém o objeto responsável por enviar a resposta do servidor para o cliente após o processamento da requisição, que é conseguida por meio método *send*. É importante observar

que, neste projeto, todas as requisições do cliente e respostas do servidor são feitas usando a codificação de dados JSON.

O exemplo expresso na caixa de código-fonte 4.30 demonstra o uso dos argumentos *req* e *res* descritos nesta seção. Foi assumido que são recebidos os dados *comando* e *valor*. Se o comando for "inverter", é chamada a função que inverte um botão, cuja resposta pode ser *erro1* ou *sucesso*. Ainda, se o comando for diferente de "inverter", a resposta é *erro2*.

```

1 function processaRequisicaoPost(req, res) {
2   var comando = req.body.comando;
3   var botao = req.body.valor;
4
5   if(comando == "inverter") {
6     inverteBotao(botao, function(err) {
7       if (err) res.send('{"resposta":"erro1"}'); return;
8       res.send('{"resposta":"sucesso"}');
9     });
10  }
11  else res.send({ "resposta": "erro2" });
12}

```

Código-fonte 4.30: Exemplo de uso dos argumentos *req* e *res* definidos pelo Express.js

Também é importante entender o uso dos módulos *fs* e *child_process* incluídos por padrão no Node.js e cuja documentação da API pode ser obtida no site do Node.js (<https://nodejs.org/dist/latest-v4.x/docs/api/>). O módulo *fs* implementa o acesso ao sistema de arquivos, com funções de leitura e escrita a arquivos, mudança de dono ou modo de execução, criação e leitura de diretórios, dentre outras. Quanto à leitura e escrita de arquivos, esta pode ser realizada de forma assíncrona ou síncrona — usuários que ainda estão no processo de aprendizado de orientação a eventos tendem a usar a versão síncrona, porém isto deve ser evitado, uma vez que faz o bloqueio da execução da aplicação enquanto o arquivo não acabar de ser acessado.

Já o módulo *child_process* permite a criação de processos filhos do Node, que são executados independentemente deste: basicamente é possível executar uma linha de comando ou script do *shell* independente do Node. Neste módulo, foi utilizada a função *exec* para ajustar a data e hora da BBB após receber uma requisição do cliente, por meio dos comandos **date -s** e **hwclock -w**, que ajusta a data e hora do sistema e sincroniza com o RTC do sistema, respectivamente. A implementação desta funcionalidade pode ser verificada na função *configRoute* do código-fonte C.3, encontrado no apêndice C.

4.7.3 Registros e verificações em geral

Para as funções de registro, ou *log*, e de verificações, além de eventuais funções de miscelânea, foi escrito um módulo separado, cujo código-fonte está documentado na caixa C.4 do apêndice C. AS funções e suas respectivas descrições são apresentadas:

- **startTemperatureLogging()** — inicia o script Python descrito na seção 4.6.1, por meio de um processo filho do Node.
 - Tenta deletar o arquivo que contém a última leitura de temperatura (*/var/www/datalog/instant.csv*) antes de iniciar o script em Python.
 - Define o comportamento caso o script pare de ser executado: imprime uma mensagem de debug.
- **stopTemperatureLogging()** — aborta a execução do script Python iniciado por *startTemperatureLogging*.
- **getTemperatureReading(logFilePath, logReadHandler, callback)** — lê a mais recente amostragem de temperatura, que está salva no arquivo especificado pelo argumento *logFilePath*. Além da descrição aqui apresentada, o funcionamento desta função está descrito no algoritmo da figura 4.26
 - Recebe como argumento uma função de callback padrão, ou seja, cujo primeiro argumento é o possível erro e o segundo argumento é o valor de temperatura obtido em caso de sucesso.
 - O argumento *logReadHandler* é um objeto externo à função e que deve guardar o valor da última leitura de temperatura, as últimas 5 *timestamps* (registros de data/hora) e a contagem de erros de leitura de temperatura referentes a um sensor específico.
 - Se há algum problema na leitura do arquivo que deve conter o registro da última temperatura lida, a variável de erro é incrementada e, se a contagem exceder 180 erros, a mensagem de erro da função de *callback* indica que muitos erros foram detectados. Note-se que 180 é um valor quase que arbitrário que pode representar 3 minutos de leituras consecutivas erradas ou 180 erros esporádicos de leitura do arquivo e cuja escolha foi realizada pois representa 5% das amostras contidas em uma hora e que, por sua vez, é aceito como o tempo mínimo de uma brassagem ou fervura. Os testes não foram levados à exaustão, mas há espaço para a realização de ensaios para determinação de um valor mais adequado.

- Em caso de leitura do arquivo com sucesso, se a formatação do valor lido estiver errada, imprime erro, incrementa a variável de erro e, no caso de a contagem exceder 180, seta a mensagem de erro da função de *callback* indicando que muitos erros foram detectados.
 - Se o valor lido for consistente, atualiza o objeto *logReadHandler*. Compara as últimas 5 *timestamps* obtidas e incrementa a variável de erro se todas elas forem coincidentes — isto indica primariamente que o sensor de temperatura está mal conectado, ou mesmo desconectado em caso de erros consecutivos. Por isto a mensagem de erro da *callback* retorna a possibilidade de problemas com o sensor. No caso de a contagem de erros exceder 180, a mensagem de erro da função de *callback* indica que muitos erros foram detectados ao invés de avisar sobre o sensor.
 - Por fim, se a leitura do registro de temperatura passar por todas as checagens, é retornado nulo na mensagem de erro e o valor de temperatura como segundo argumento.
- **checkRecipeIntegrity(recipe, path, res)** — Verifica a integridade da receita selecionada pelo usuário antes de permitir o início de uma brassagem. Os argumentos recebidos pela função são o nome da receita, o diretório de receitas a ser verificado e o objeto que contém a resposta da requisição AJAX feita pelo cliente, respectivamente.
 - Se não consegue ler o arquivo da receita, responde com erro ao cliente.
 - Caso contrário, lê o arquivo *lockfile* para verificar se já existe uma brassagem em andamento. Se não consegue ler o *lockfile* ou ele indica que há uma brassagem em andamento, também responde com erro.
 - Verifica a consistência da receita com o seguinte critério: há campos estritamente necessários para a produção e que impedem o início da receita se estiverem em branco, campos que geram avisos mas permitem o início de uma receita após o usuário se declarar ciente e campos indiferentes para o sistema. São os campos estritamente necessários: água de brassagem, tempo da fervura, temperatura inicial de brassagem, primeiro degrau de temperatura, tempo do primeiro degrau, malte 1, quantidade do malte 1, lúpulo 1, quantidade do lúpulo 1 e tempo de adição do lúpulo 1. Os campos que geram avisos são: nome da receita, estilo, levedura, água de sparging e temperatura de sparging. Todos os outros campos são indiferentes à verificação de consistência da receita.

- Se tudo deu certo, uma variável global é setada para registro e uso posterior.

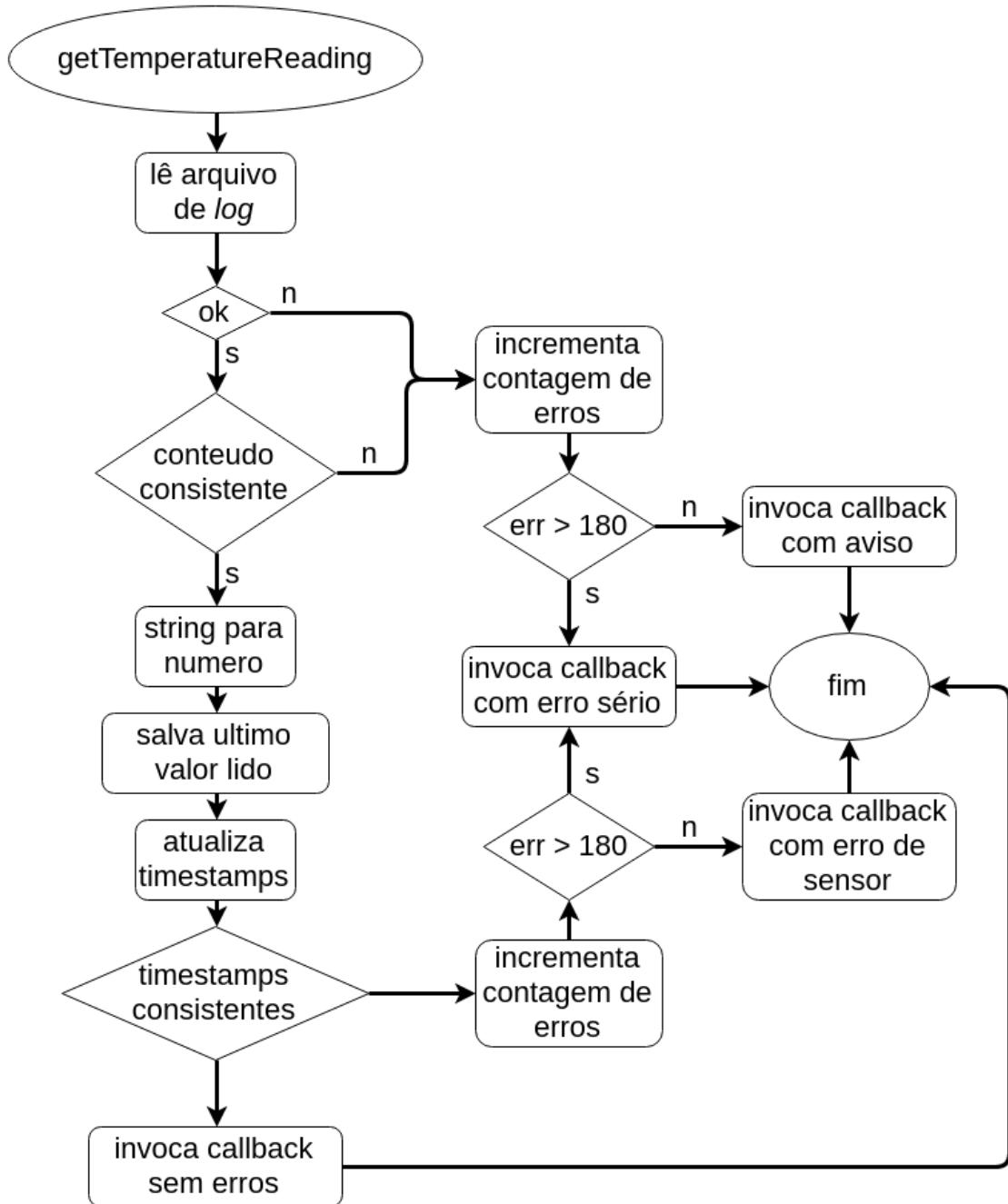


Figura 4.26: Representação em fluxograma do algoritmo de implementação da função `getTemperatureReading`

- `logToFile(message, code, notableTimestamp)` — função que realiza registro do status da brassagem sempre que invocada. Basicamente, ela salva a variável global `environmentVariables` em um arquivo de texto.
 - O argumento `message` pode conter uma string com uma mensagem explicativa com, por exemplo, a situação ou estágio da brassagem em que ocorreu o registro,

mas pode também ser deixado em branco. O argumento *code* tem basicamente o mesmo propósito, mas é um número predefinido para cada situação possível da brassagem, que permite que o arquivo de registro possa ser interpretado com maior facilidade; por este motivo, recomenda-se veementemente que não seja deixado em branco. A tabela 4.4 apresenta a correspondência entre os valores do argumento *code* e seus significados.

- O argumento *notableTimestamp* é usado para indicar quando uma situação notável ocorreu, como por exemplo o início da brassagem, o final de uma rampa ou degrau de temperatura, o início da fervura, etc. Para todas as outras situações deve ser deixado em branco, sendo que as situações notáveis são predefinidas.
- O objeto *environmentVariables*, declarado no código-fonte raiz da aplicação em Node, que foi abordado na seção 4.7.2 e pode ser obtido no apêndice C, também é descrito na caixa 4.31. Ele é salvo no arquivo de *log* no formato JSON e para isto é empregada a função *JSON.stringify* conforme descrito na linha de código-fonte exibida na caixa 4.32.
- **sendRecipeNames(path, res)** — responde para o cliente usando o objeto *res* com todas as receitas disponíveis no diretório passado como argumento *path* para a função.
 - É preciso notar que, quando uma receita é deletada, a única modificação realizada é que a extensão do arquivo *.recipe* é modificada para *.recipe.del*, assim é fácil recuperar a receita caso o usuário a tenha deletado por engano. Por isso nesta função, além de ler os nomes dos arquivos presentes no diretório e excluir a extensão, foi preciso excluir os arquivos deletados.

```

1 //global variables that should also be saved to a backup file
2 //periodically
3 global.environmentVariables = {
4   warn: "", //if not empty holds some warning message
5   code: "", //tells the same as msg, but as an index, easier
6   //to check programatically
7   tmpMT: "", //mash tun temperature
8   tmpMTsetp: "", //mash tun current setpoint
9   tmpBK: "", //brewing kettle temperature, also the "hot
10  //liquor tank" for sparging
11  tmpBKsetp: "", //brewing kettle/hot liquor tank current
12  //setpoint
13  timeLeft: "", //helping variable to tell the client the time
14  //left for the step rests or the boil, etc
15  readyForNextStep: false, //set whenever the system is ready
16  //for the next step
17  auto: true, //whether the process control is running
18  //automatically or there is human intervention

```

```

12     processFail: false, //flag is set if the process fails
13         irreversibly
14     msg: "", //holds some explanatory message
15     timestamps:{ //notable timestamps
16         start: "", //epoch time of the first request to start a
17             recipe
18         startHeating: "", //epoch time of the start to heat the
19             mash water
20         finishHeating: "", //epoch for the finish of the heating
21             of mash water
22         //start of the ramps
23         sRamp0: "", sRamp1: "", sRamp2: "", sRamp3: "", sRamp4: ""
24             , sRamp5: "", sRamp6: "", sRamp7: "",
25         //finish of the ramps
26         fRamp0: "", fRamp1: "", fRamp2: "", fRamp3: "", fRamp4: ""
27             , fRamp5: "", fRamp6: "", fRamp7: "",
28         startDrain: "", //start of the sparging process, when the
29             mash is parcially drained to the BK
30         startSparge: "", //here the recirculation pump starts
31             working
32         heatingBoil: "", //started to heat the wort after sparging
33         boilStart: "", //time when temperature is near enough
34             boiling (>96°C)
35         boilFinishScheduled: "", //time when the boil is scheduled
36             to finish
37         curr: "", //epoch time of the current variables state
38     },
39     ioStatus: gpioCfg.all_io, //also records the IO status
40     okToStart: false, //true if a recipe is ok enough to start
41         a production
42     recipe: "" //recipe name
43 };

```

Código-fonte 4.31: Declaração do objeto *environmentVariables*

```

1 dataToSave = JSON.stringify(global.environmentVariables) + "\n";

```

Código-fonte 4.32: Uso da função *JSON.stringify* para codificar um objeto em string JSON

Tabela 4.4: Correspondência entre o valor numérico e o significado dos códigos usados para registro da brassagem

Valor	Significado
0	requisição para início da brassagem
1	produção iniciada
2	esquentando água da mostura
3	esperando adição dos grãos
4	rampa da mostura em execução
5	degrau da mostura em execução
6	<i>sparging</i> em execução
7	transbordamento da MT
8	esquentando mosto para a fervura
9	fervendo o mosto
10	lúpulo adicionado
11	
11	
11	
11	
11	

4.7.4 Controle do processo de brassagem

O módulo responsável pelo controle do processo de brassagem é o *ctrl.js*, cujo código-fonte está disponível no apêndice C, na caixa de código-fonte C.5. É importante ressaltar que, embora o comportamento do Node.js seja assíncrono o processo de brassagem é sequencial, o que implica no uso de diversas funções de *callback* aninhadas. No intuito de evitar o *inferno de callbacks*, apresentado na seção 2.4, este processo sequencial foi quebrado em algumas funções que implementam a funcionalidade do módulo e outras auxiliares. Seu funcionamento e aspectos chave são discutidos na presente seção.

A primeira função do processo de controle é a *startMashingProcess(recipe, res, lockFile, recipesPath, callback)*, invocada a partir de uma requisição do cliente e, portanto, sua chamada está presente no módulo referente às rotas do Express, discutido na seção 4.7.2. Ela recebe como parâmetros o nome da receita iniciada, o objeto de resposta da requisição AJAX, o caminho para o arquivo *lockfile*, o caminho para o diretório das receitas e uma função de *callback*. Em primeiro lugar é verificada a flag global *global.environmentVariables.okToStart* que indica se a receita está pronta para ser executada — esta flag é setada previamente pela função *checkRecipeIntegrity*, descrita na seção 4.7.3 e chamada durante o processo de interação pré-brassagem do usuário com o sistema. Depois disto, também é verificado se o

nome da receita passado como argumento é idêntico ao da variável global de controle *global.environmentVariables.recipe*.

Caso as verificações iniciais sejam positivas, é escrito o valor "1" no arquivo *lockfile*, indicando daqui para a frente que há uma brassagem em andamento. O conteúdo da receita é lido em uma variável e o *log* de temperatura é iniciado, ou seja, o script Python é invocado como um processo filho do Node. Também é feito o *log* inicial do processo de brassagem. Se tudo ocorreu sem erros, o servidor envia uma resposta para o cliente indicando sucesso, chama a função de *callback* com a variável erro nula e invoca a função *heatMashWater*, que dá prosseguimento ao controle do processo. Em caso de erro em alguma das situações descritas, é enviada uma resposta de erro para o cliente e a função de *callback* é invocada com a variável erro indicando o erro ocorrido. Note-se que a função de *callback* não tem importância no que diz respeito ao controle, mas serve somente para retornar o status da tentativa de início da produção.

A função *heatMashWater(recipeContents)* recém chamada, dá prosseguimento ao controle da brassagem. Como o seu nome sugere, ela é responsável pelo aquecimento da água da brassagem até que ela atinja a temperatura inicial, mantendo este *setpoint* até que o usuário adicione os maltes à MT. O único argumento que a função recebe é o conteúdo da receita, a partir do qual é obtido o *setpoint*. Inicialmente a bomba de recirculação é ligada e é feito um *log* da brassagem que anota a timestamp notável de inicio de aquecimento.

São iniciados dois laços de repetição: um que faz um *log* da situação da brassagem a cada 5 segundos e outro que é repetido a cada 1 segundo e que lê a temperatura da água e chama a função auxiliar *updateHeatingConditions* para controlar o elemento aquecedor da MT. No *callback* da *updateHeatingConditions* fica definido que, se esta é a primeira vez que a temperatura desejada é atingida, o laço de repetição do *log* é atualizado para refletir a mudança de situação de "esquentando água da brassagem" para "esperando adição dos maltes". Se não é a primeira vez que é atingido o *setpoint*, a *callback* não faz nada. Em resumo, após a temperatura atingir o valor desejado, ela é mantida até que o usuário adicione os maltes.

Concomitantemente, o laço de controle de temperatura da função *heatMashWater* verifica a flag global *global.environmentVariables.readyForNextStep* que só é setada após o usuário indicar, por meio da GUI, que ele adicionou os maltes. Quando isto acontece, a flag é zerada e os laços de *log* e de controle são interrompidos. Por fim é chamada a função de controle de rampas e degraus de temperatura, mas antes de entrar nos seus detalhes de funcionamento, será explicada a função auxiliar *updateHeatingConditions(vessel, setp, temperature,*

callback), uma vez que já foi feito uso dela.

Os argumentos que ela recebe possuem uma interpretação bem direta: *vessel* é o recipiente a ser controlado, ou seja, pode ser a MT ou o BK; *setp* e *temperature* são respectivamente a temperatura desejada e a atual e; *callback* é a função executada sempre que a temperatura é maior ou igual ao valor do *setpoint*, e cujo argumento passado para ela é nulo. O controle do elemento aquecedor é implementado da seguinte forma: quando a temperatura está abaixo de 70% do *setpoint*, o aquecimento fica ligado 100% do tempo; quando a temperatura está entre 70% e 90% do *setpoint*, o aquecimento fica ligado 66% do tempo; quando a temperatura está entre 90% e 100% do *setpoint*, o aquecimento fica ligado 33% do tempo e; acima disto o aquecimento fica desligado. Cabe ressaltar que a vantagem desta função auxiliar não é o sistema de controle de aquecimento, mas sim o fato de que este é auto-contido, ou seja, para quaisquer implementações de controle de temperatura (PID, histerese, direto), basta modificar esta função, sem alteração do resto do código. Outra condição que deve ser atendida é que esta função precisa ser chamada periodicamente dentro de um laço de repetição, uma vez que ela somente atualiza o valore da saída (elemento aquecedor) baseada nas entradas (temperatura e *setpoint*).

4.7.5 Simulação do controle do processo de brassagem

4.8 Interface de usuário

4.8.1 Tratamento de formulário em PHP

A interface de usuário foi desenvolvida para acesso via navegador da internet. Dois servidores foram utilizados em conjunto: o servidor Apache, que é um servidor *web* robusto e de fácil implementação e o *framework Express.js*, para *Node.js*. As linguagens utilizadas para a implementação da aplicação foram: a linguagem de marcação HTML, a linguagem de folhas de estilo CSS, a linguagem de programação *client-side* interpretada Javascript em conjunto com a biblioteca *crossbrowser* jQuery e o método AJAX, a linguagem de programação *server-side* interpretada PHP e, a linguagem de programação *server-side* Javascript interpretada pela aplicação *Node.js*.

Parte da implementação foi feita por meio de acesso ao terminal via SSH e parte via *Cloud9*, que é uma IDE online. A estrutura da aplicação desenvolvida é apresentada:

- Página inicial

- Página de apresentação
- Gráficos de temperatura dinâmico e estático (página somente para versão de demonstração)
- Menu de seleção de tarefas
 - Gerenciador de receitas de cerveja
 - * Editor de receitas
 - Gerenciador de início da produção (em desenvolvimento)
 - Acompanhamento e possibilidade de ajustes da brassagem (em desenvolvimento)
 - Estatísticas (a ser desenvolvida)
 - Opções e ajuste de configurações do sistema (a ser desenvolvida)

Os códigos desenvolvidos e comentados podem ser consultados no apêndice D e os resultados são descritos na seção 5.1.

4.9 Circuitos de interface entre a BBB e sensores/atuadores

4.9.1 Acionamentos de potência

4.9.2 Detector de *zero-crossing*

4.9.3 Detector de nível de líquido

4.10 Sistema de controle de temperatura

4.10.1 Resistores de aquecimento

4.10.2 Controlador PID

Capítulo 5

Resultados e Discussões

Neste capítulo são apresentados e discutidos os resultados obtidos a partir dos métodos descritos na seção 4.

5.1 Interface de usuário

Nesta seção são apresentados os resultados do desenvolvimento da interface de usuário.

A página inicial é composta de uma imagem com uma área clicável no centro, sobre a etiqueta *Start Here*, que leva à página de apresentação. O resultado visual de seu desenvolvimento pode ser verificado na figura 5.1.



Figura 5.1: Página inicial da UI

A página de apresentação contém uma breve descrição do projeto e de seu objetivo, além dos botões que levam: à interface onde estão os gráficos de temperatura e; ao menu de seleção de tarefas. O resultado visual de seu desenvolvimento pode ser verificado na figura 5.2.

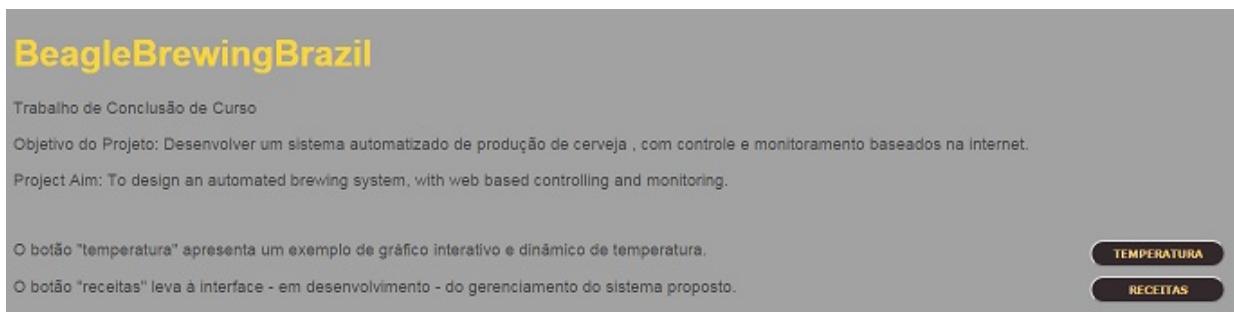


Figura 5.2: Página de apresentação da UI

Na página que plota os gráficos de temperatura em função do tempo é possível ler o valor atual da temperatura do sensor DS18B20. Também é nesta página que encontra-se o gráfico dinâmico e interativo, que é atualizado a cada aproximadamente 1 segundo – o tempo de atualização não é fixo em função da natureza de operação do interpretador do Javascript e o tempo de amostragem da temperatura não é fixo em função do sistema operacional escolhido não ser de tempo real. Neste gráfico, apresentado na figura 5.3 é possível:

- Selecionar as variáveis plotadas
- Escolher a plotagem do gráfico em linha ou área
- Empilhar ou sobrepor os gráficos (no caso da plotagem em área)
- Unir os pontos amostrados em degrau, linha ou realizar uma interpolação cardinal
- Suavizar a plotagem empregando um filtro de média móvel
- Escolher o número de pontos plotados, de maneira a obter o registro dos últimos: 5 minutos, 30 minutos ou 1 hora de amostragens
- Ajuste fino do número de pontos plotados, por meio da escolha dos limites inicial e final, utilizando a barra sob o gráfico
- Leitura precisa do valor de qualquer ponto plotado e da data/hora da amostragem, colocando o cursor sobre o local do gráfico no qual se deseja realizar a leitura

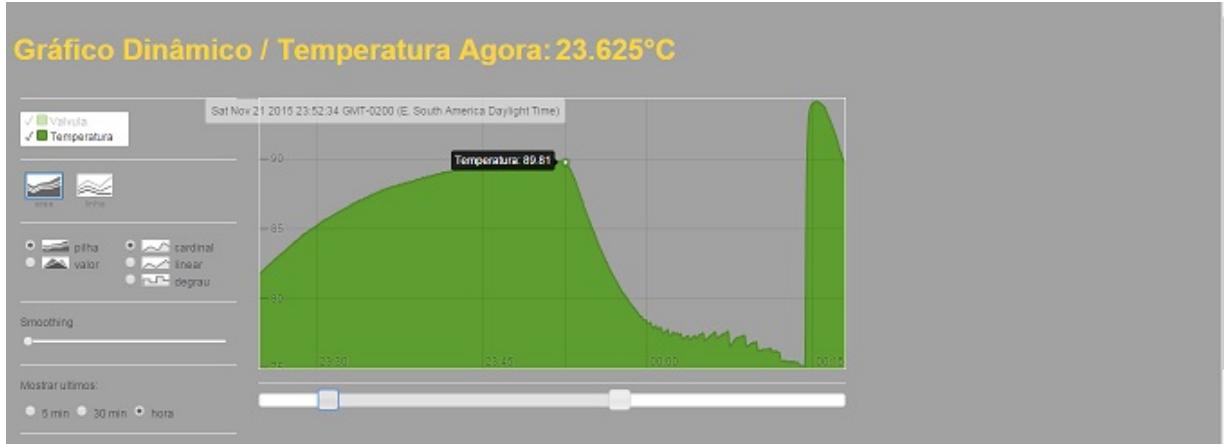


Figura 5.3: Gráfico dinâmico de temperatura em função do tempo
Ajuste fino e cursor sobre o gráfico sendo aplicados

Ainda na página que plota os gráficos de temperatura, é plotado um gráfico estático do histórico de todos os pontos registrados no arquivo de *log*. O resultado visual de seu desenvolvimento pode ser verificado na figura 5.4.

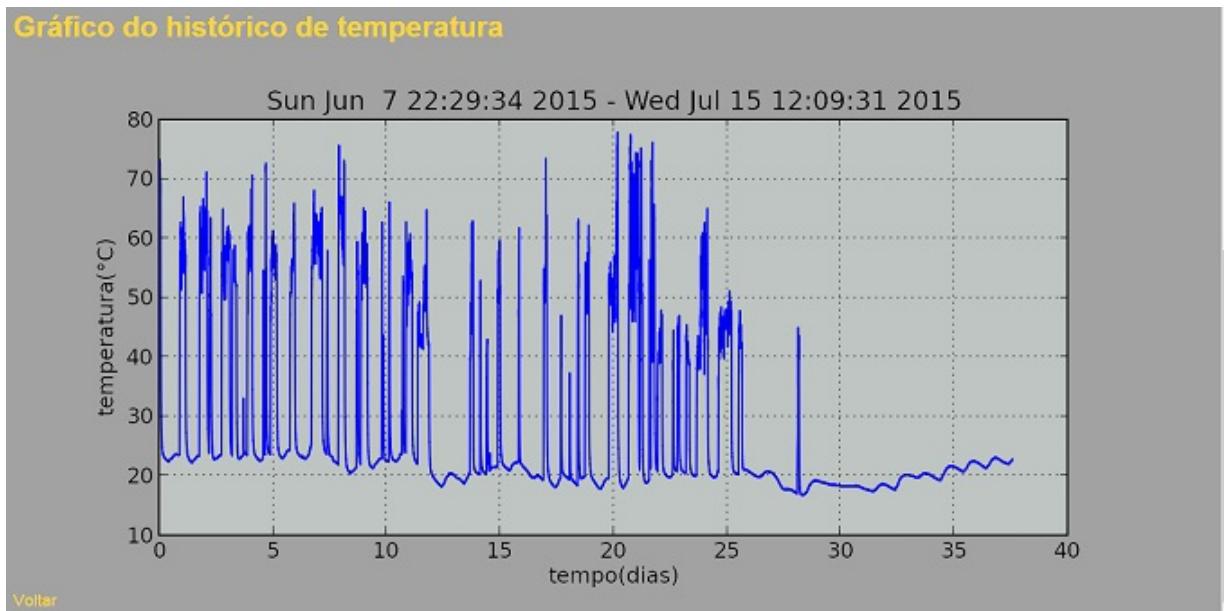


Figura 5.4: Gráfico estático do histórico de temperaturas registradas

O menu de tarefas, ilustrado na figura 5.5 é basicamente uma página com links para o gerenciador de receitas, o gerenciador de início da produção, o acompanhamento e ajustes da brassagem e as estatísticas. A página de configurações e opções não foi implementada em tempo.

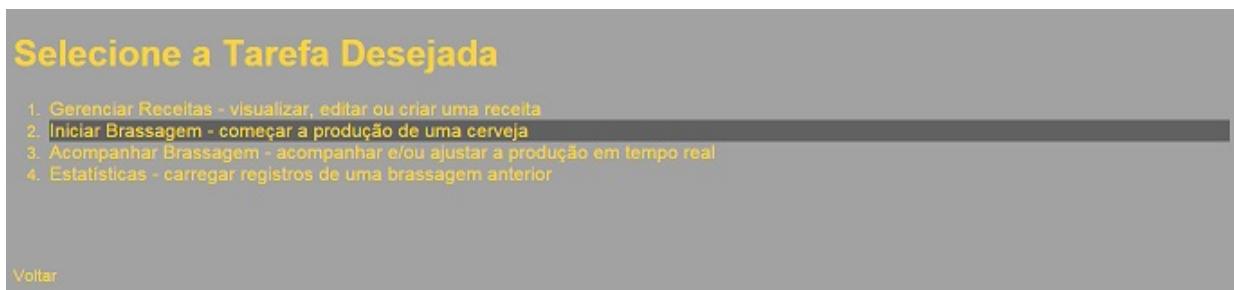


Figura 5.5: Menu de tarefas da UI

O gerenciador de receitas lista e permite acesso a todas as receitas cadastradas no sistema, possibilita exclusão (com a opção de desfazer para exclusão acidental) e criação de nova receita. Quando o cursor é colocado sobre uma receita cadastrada, é apresentado um *preview* para que o usuário tenha uma realimentação rápida de seu conteúdo. A figura 5.6 ilustra esta interface em uso, com o cursor sobre a receita *TheMightyMightyIPA* e seu *preview*, além da receita *Exemplo Kolsch* excluída e com possibilidade de recuperação.

Figura 5.6: Gerenciador de receitas da UI
Cursor sobre a receita *TheMightyMightyIPA* e receita *Exemplo Kolsch* excluída

O editor de receitas apresenta caixas de entrada para o usuário adicionar as características desejadas da cerveja, sendo que a cada malte, lúpulo e temperatura adicionado, aparece automaticamente um campo extra para estes itens, limitado a 8 entradas para cada. Só foi implementado um botão de voltar pois há uma funcionalidade de salvamento automático – o aviso no canto superior direito da tela mostra o status do salvamento da receita: no caso da figura 5.7, que ilustra a implementação do editor, nota-se que a mensagem de status avisa o usuário que há um campo essencial para a produção da cerveja que não foi preenchido.

Crie ou edite sua receita!

Receita salva, mas há campos necessários não preenchidos.

Nome da Receita:	bla
Estilo da Cerveja:	bla *
Levedura:	bla *
Água - mosturação(t):	*
Água - sparging (t):	1
Fervura do mosto(min):	75 *
Maltes	
Malte 1:	1 Quantidade(kg): 1 *
Malte 2:	Quantidade(kg):
Lúpulos	
Lúpulo 1:	1 Quantidade(g): 1 Tempo de adição(min): 1 *
Lúpulo 2:	Quantidade(g): Tempo de adição(min):
Cozimento do mosto	
Temperatura inicial:	1 *
Temperatura 1:	1 Tempo(min): 1 *
Temperatura 2:	54 Tempo(min):

[Voltar](#)

Figura 5.7: Editor de receitas da UI
Receita *bla* parcialmente preenchida com aviso de campos essenciais em branco

O gerenciador de início da produção apresenta ao usuário um aviso sinalizando de que tudo deve estar devidamente verificado antes do início da produção. Também apresenta uma caixa de seleção da receita a ser produzida. O resultado visual de seu desenvolvimento pode ser verificado na figura 5.8.

Iniciar Brassagem

⚠ Antes de iniciar uma receita, certifique-se de que o equipamento está devidamente limpo e sanitizado, e de que a água e ingredientes estão a postos para a produção.

Exemplo American Stout *

[Voltar](#)

Figura 5.8: Gerenciador de início da produção

O acompanhamento e possibilidade de ajustes da brassagem implementa um controle manual do LED1 da BBB, bombas, válvulas, resistores de potência e servo-motor do sistema, com *feedback* visual para o usuário. Na figura 6.1 é apresentada a UI e o console do Javascript, no qual aparecem as respostas do servidor para a seguinte sequência de comandos: ativação da bomba do mosto, ativação da válvula da água, valor do ângulo do servo-motor setado para 87° e, ativação seguida desligamento do aquecedor de fervura.



Figura 5.9: Interface de acompanhamento e ajustes da brassagem - console Javascript com *feedback* do servidor para uma sequência de comandos executados

A página de estatísticas apresenta somente um aviso de página em desenvolvimento, conforme ilustrado na figura 5.10.



Figura 5.10: Aviso de página em desenvolvimento

Capítulo 6

Conclusão ou Conclusões

A partir dos métodos descritos no capítulo ?? e resultados apresentados no capítulo 5, é possível notar que estes não são compatíveis com o cronograma previsto, introduzido na seção 1.2.1. Isto é função do planejamento do trabalho, documentado no próprio cronograma – a redação da monografia teve começo previsto para janeiro de 2016, por isso a presente monografia parcial está incompleta em função de adaptações realizadas no planejamento e que não conseguiram reservar tempo suficiente para documentar todas as tarefas realizadas ao longo do semestre. Na figura é apresentado um gráfico de pizza no qual todas as macrotarefas do semestre e a respectiva parcela de dedicação dada a cada tarefa podem ser examinadas.

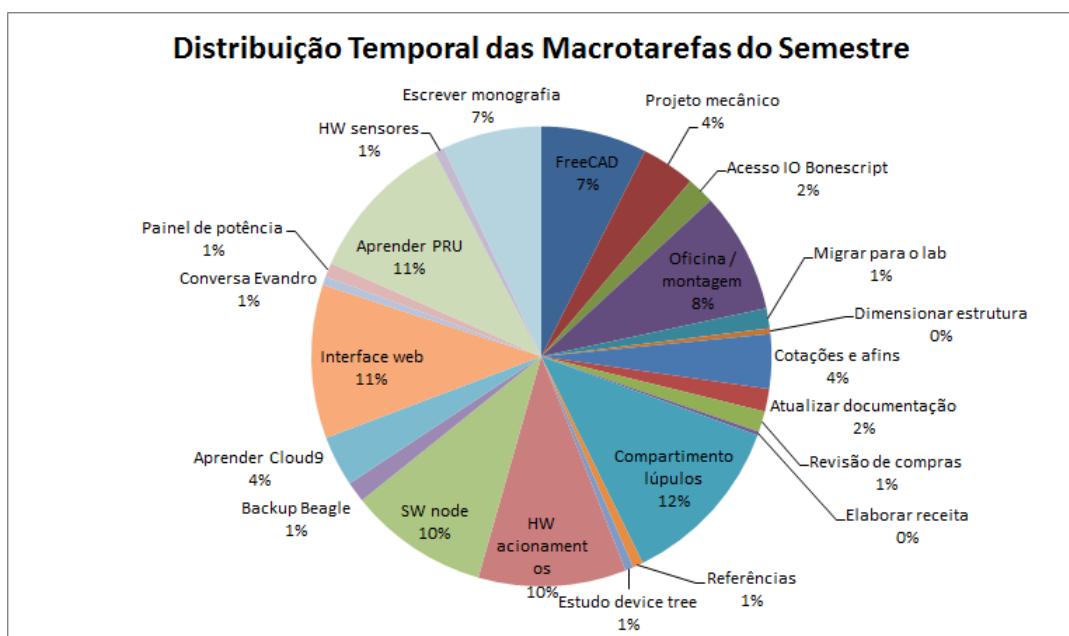


Figura 6.1: Distribuição temporal das macrotarefas do semestre

Com relação aos métodos e resultados documentados, quase que exclusivamente a respeito da interface de usuário, percebe-se que esta foi bem estruturada, com as evidências de que funcionalidades complexas e uma padronização visual foram implementadas. Ainda assim, é possível perceber que há espaço para melhorias e finalização de tarefas: a página com os gráficos de temperatura ainda é somente uma demonstração, mas não contribui como funcionalidade do sistema; a interface de início de brassagem não implementa funcionalidade que permita o início efetivo de uma brassagem e; a interface de estatísticas nem começou a ser planejada.

Em compensação, outros aspectos da UI foram bem sucedidos: as páginas introdutórias são funcionais; o menu de seleção de tarefas é bem organizado e auto-descritivo; a interface de gerenciamento e edição de receitas funciona perfeitamente dentro das especificações iniciais e é totalmente voltada ao usuário, utilizando conceitos de *desing thinking* e ações que podem ser desfeitas [54, 55] e; o controle manual do sistema é simples, intuitivo e funcional.

Trabalhos futuros

Isso é para o Relatório Final de defesa.....

Referências Bibliográficas

- [1] J.J. PALMER. *How to Brew*. Brewers Publications. Terceira edição, 2006.
- [2] BREWERS ASSOCIATION. *Number of Breweries*. Disponível em: <https://www.brewersassociation.org/statistics/number-of-breweries/>, Acesso em: 12 de novembro de 2015.
- [3] O. CERVIERI JÚNIOR et al. *O Setor de Bebidas no Brasil*. BNDES Setorial, Rio de Janeiro, n. 40, p. [93]-129, set. 2015.
- [4] REVISTA DA CERVEJA. n.1, mai. 2012.
- [5] REVISTA DA CERVEJA. n.2, jul. 2012.
- [6] ASSOCIAÇÃO BRASILEIRA DA INDÚSTRIA DA CERVEJA. *Anuário 2014*. Disponível em: <http://www.cervbrasil.org.br/arquivos/anuariofinal2014.pdf>, Acesso em: 13 de novembro de 2015.
- [7] ASSOCIAÇÃO BRASILEIRA DE BEBIDAS. *Categorias*. Disponível em: <http://www.abrabe.org.br/categorias/>, Acesso em: 18 de novembro de 2015.
- [8] PICOBREW INC. *Pico Brew*. Disponível em: <https://www.picobrew.com/>, Acesso em: 18 de novembro de 2015.
- [9] WILLIAMSWARN NZ LTD. *The WilliamsWarn Personal Brewery*. Disponível em: <http://www.williamswarn.com/The-WilliamsWarn#.VkyEqnarTIU>, Acesso em: 18 de novembro de 2015.
- [10] D.E. BRIGGS et al. *Brewing - Science and Practice*. Woodhead Publishing Limited. Terceira edição, 2011.
- [11] A. TIERNEY-JONES. *1001 Cervejas para Beber Antes de Morrer*. Sextante, 2011.

- [12] DOGFISH HEAD. *120 Minute IPA*. Disponível em: <http://www.dogfish.com/brews-spirits/the-brews/occassional-rarities/120-minute-ipa.htm>, Acesso em: 23 de novembro de 2015.
- [13] A. FODOR. *How to Remove Trub*. Brew Your Own, v. 3, n. 12, dez. 1997. Disponível em: <http://byo.com/stories/issue/item/890-how-to-remove-trub>, Acesso em: 14 de dezembro de 2015.
- [14] G. COLEY. *BeagleBone Black System Reference Manual*. Revision C1, mai. 2014. Disponível em: https://github.com/CircuitCo/BeagleBone-Black/blob/master/BBB_SRM.pdf?raw=true, Acesso em: 02 de dezembro de 2015.
- [15] H.W.; WOLTMAN B.D. HE, N.; HUANG. *The Use of BeagleBone Black Board in Engineering Design and Development*. The 2014 ASEE North Midwest Section Conference, Iowa City, IA, out. 2014.
- [16] J. LUMME. *BeagleBone Home Automation*. Packt Publishing, 2013.
- [17] TEXAS INSTRUMENTS. *AM335x SitaraTMProcessors Technical Reference Manual*. Revision fev. 2015. Disponível em: <http://www.ti.com/lit/ug/spruh731/spruh731.pdf>, Acesso em: 02 de dezembro de 2015.
- [18] J. FEDRIZZI, M.; SORIA. *Application of a single-board computer as a low cost pulse generator*. Measurement Science and Technology, v. 26, n. 9, set. 2015. Disponível em: <http://arxiv.org/abs/1504.07024>, Acesso em: 02 de dezembro de 2015.
- [19] TEXAS INSTRUMENTS. *PRU assembly language tools v2.1*. Disponível em: <http://www.ti.com/lit/ug/spruhv6a/spruhv6a.pdf>, Acesso em: 29 de março de 2016.
- [20] TEXAS INSTRUMENTS. *PRU optimizing C/C++ compiler v2.1*. Disponível em: <http://www.ti.com/lit/ug/spruhv7a/spruhv7a.pdf>, Acesso em: 29 de março de 2016.
- [21] Devicetree.org. *Device Tree Usage*. Disponível em: http://www.devicetree.org/Device_Tree_Usage, Acesso em: 21 de dezembro de 2015.
- [22] J. COOPER. *Introduction to the BeagleBone Black Device Tree*. Disponível em: <https://learn.adafruit.com/>

introduction-to-the-beaglebone-black-device-tree?view=all, Acesso em: 21 de dezembro de 2015.

- [23] D. MOLLOY. *Beaglebone: GPIO Programming on ARM Embedded Linux*. mai. 2012. Disponível em: https://www.youtube.com/watch?v=wui_wU1AeQc, Acesso em: 28 de dezembro de 2015.
- [24] S. TILKOV, S.; VINOSKI. *Node.js: Using JavaScript to Build High-Performance Network Programs*. 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), Lincoln/NE, p. 86-89, nov. 2015. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7426643>, Acesso em: 19 de abril de 2016.
- [25] ZÁKOVÁ K. BOSÁK, T. *Node.js Based Remote Control of Thermo-optical Plant*. 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV), Bangkok - Tailândia, p. 209-213, fev. 2015. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7087293>, Acesso em: 19 de abril de 2016.
- [26] T. OGASAWARA. *Workload Characterization of Server-Side JavaScript*. 2014 IEEE International Symposium on Workload Characterization (IISWC), Raleigh/NC, p. 13-21, out. 2014. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6983035>, Acesso em: 19 de abril de 2016.
- [27] S. TILKOV, S.; VINOSKI. *Node.js: Using JavaScript to Build High-Performance Network Programs*. IEEE Internet Computing, p. 80-83, nov. 2010. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5617064>, Acesso em: 19 de abril de 2016.
- [28] P. TEIXEIRA. *Professional Node.js - Building Javascript Based Scalable Software*. John Wiley Sons, Inc., 2013.
- [29] callbackhell.com. *Callback Hell - A Guide to Writing Asynchronous JavaScript Programs*. Disponível em: <http://callbackhell.com/>, Acesso em: 19 de abril de 2016.
- [30] S. ROBINSON. *Avoiding Callback Hell in Node.js*. Stack Abuse, jul. 2015. Disponível em: <http://stackabuse.com/avoiding-callback-hell-in-node-js/>, Acesso em: 19 de abril de 2016.

- [31] T. OGASAWARA. *Workload Characterization of Server-Side JavaScript*. 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milão - Itália, p. 280-285, dez. 2015. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7389066>, Acesso em: 20 de abril de 2016.
- [32] K.C. SEDRA, A.S; SMITH. *Microelectronic Circuits*. New York : Oxford University Press. Sétima edição, 2015.
- [33] P.R. VERONESE. *Junções*. SEL-EESC-USP, 2012.
- [34] L. BOYLESTAD, R.; NASHELSKY. *Dispositivos Eletrônicos e Teoria de Circuitos*. LTC - Livros Técnicos e Científicos Editora S.A. Oitava Edição, 2011.
- [35] MAXIM INTEGRATED. *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*. 2008. Disponível em: <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>, Acesso em: 02 de dezembro de 2015.
- [36] B. LINKE. *Overview of 1-Wire Technology and Its Use*. jun. 2008. Disponível em: <http://pdfserv.maximintegrated.com/en/an/AN1796.pdf>, Acesso em: 02 de dezembro de 2015.
- [37] MAXIM INTEGRATED. *Guidelines for Reliable Long Line 1-Wire Networks*. set. 2008. Disponível em: <http://pdfserv.maximintegrated.com/en/an/AN148.pdf>, Acesso em: 02 de dezembro de 2015.
- [38] THE LINUX KERNEL ARCHIVES. *Index of /doc/Documentation/wl*. Disponível em: <https://www.kernel.org/doc/Documentation/wl/>, Acesso em: 03 de dezembro de 2015.
- [39] DANFOSS. *How to Use Solenoid Valves*. Disponível em: http://www.danfoss.com/NR/rdonlyres/CDF180FA-4AFE-46AF-99A1-4E20A4FC1418/0/ICPS600A402_1_oms_solenoid_valves_how_to_use.pdf, Acesso em: 22 de abril de 2016.
- [40] JEFFERSON AUTOMAÇÃO. *Válvulas Solenóides - Informação de Engenharia*. Disponível em: <http://www.jefferson.ind.br/images/download/Valvula-solenoide.pdf>, Acesso em: 22 de abril de 2016.
- [41] E. ROSE, T.; MONTGOMERY. *Brewery CIP Automation Systems*. MBAA District NW Fall Meeting, Portland/OR, out. 2010. Disponível em:

http://www.mbaa.com/districts/northwest/events/documents/2010_10_08brewerycipautomation.pdf, Acesso em: 22 de abril de 2016.

- [42] F. FERRAZ. *Meios de Ligação de Tubos - Conexões de Tubulação - Válvulas Industriais.* IFBA, abr. 2009. Disponível em: http://docente.ifb.edu.br/paulobaltazar/lib/exe/fetch.php?media=apostila_tubulacao_ifba.pdf, Acesso em: 22 de abril de 2016.
- [43] IPEX. *EPDM FKM Chemical Resistance Guide.* IPEX Incorporated Canadá. Primeira Edição, 2009.
- [44] THE BEAGLEBOARD.ORG FOUNDATION. *Debian.* Disponível em: <http://beagleboard.org/project/debian/>, Acesso em: 03 de dezembro de 2015.
- [45] THE BEAGLEBOARD.ORG FOUNDATION. *BeagleBoard.org Latest Firmware Images.* Disponível em: <http://beagleboard.org/latest-images>, Acesso em: 03 de dezembro de 2015.
- [46] putty.org. *Download Putty.* Disponível em: <http://www.putty.org>, Acesso em: 03 de dezembro de 2015.
- [47] NTP.BR. *Saiba mais sobre o NTP.* Disponível em: <http://ntp.br/>, Acesso em: 21 de dezembro de 2015.
- [48] NIC.BR. *Núcleo de Informação e Coordenação do Ponto BR.* Disponível em: <http://nic.br/>, Acesso em: 21 de dezembro de 2015.
- [49] Divisão Serviço da Hora. *Divisão Serviço da Hora - DSHO.* Disponível em: <http://pcdsh01.on.br/>, Acesso em: 21 de dezembro de 2015.
- [50] Cloud9 IDE Inc. *License for the SDK and Packages.* Disponível em: <https://cloud9-sdk.readme.io/docs/the-licenses-for-cloud9-sdk-and-packages>, Acesso em: 25 de abril de 2016.
- [51] STRAUB B. CHACON, S. *Pro Git - Everything You Need to Know About Git.* Apress Media LLC, Segunda Edição, 2014. Disponível em: <https://git-scm.com/book/en/v2>, Acesso em: 25 de abril de 2016.
- [52] J.D. HUNTER. *Matplotlib A 2D graphics environment.* Computing In Science Engineering, v. 9, n. 3, p. 90-95, IEEE COMPUTER SOC, 2007.

- [53] *scipy.org. NumPy.* Disponível em: <http://www.numpy.org/>, Acesso em: 24 de abril de 2016.
- [54] J. KOLKO. *Design Thinking Comes of Age.* Harvard Business Review, Boston, p. 66-71, set. 2015. Disponível em: <https://hbr.org/2015/09/design-thinking-comes-of-age>, Acesso em: 04 de dezembro de 2015.
- [55] A. RASKIN. *Never Use a Warning When You Mean Undo.* A List Apart, jul. 2007. Disponível em: <http://alistapart.com/article/neveruseawarning>, Acesso em: 04 de dezembro de 2015.
- [56] Brewer's Friend. *Beer Styles - IBU Chart Graph.* Disponível em: <http://www.brewersfriend.com/2009/01/24/beer-styles-ibu-chart-graph-bitterness-range/>, Acesso em: 23 de abril de 2016.

Apêndice A

Códigos-fonte da interface de usuário

Códigos em elaboração.

Apêndice B

Scripts Python

B.1 Módulo com as funções para *log* da temperatura

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 #importa os módulos que serão usados no script
5 import time
6 import os.path
7 #import graficos_png as graph
8 #import datetime
9 #import wiringpi2 as wpi
10
11 def tread():
12     """lê temperatura do DS18B20 e retorna em celsius"""
13     tfile = open("/sys/bus/w1/devices/28-000004ee8ded/w1_slave")
14     tfile.readline()
15     temp_raw = tfile.readline()
16     tfile.close()
17     temp_raw = temp_raw.split(" ")[9]
18     temp_raw = float(temp_raw[2:])
19     return temp_raw/1000
20
21 def tprint_all(tcelsius):
22     """imprime a temperatura em diversas escalas no formato
23         HTML"""
24     print "Temperatura:<br />",
25     print "%2f &degC<br />" % tcelsius, #celsius
26     print "%2f &degF<br />" % (tcelsius*1.8+32), #fahrenheit
27     print "%2f K<br />" % (tcelsius+273.15), #kelvin
28     print "%2f &degR<br />" % (tcelsius*1.8+32+459.67), #
29     rankine
30
31 def tprint_all_terminal(tcelsius):
32     """imprime a temperatura em diversas escalas
33         formatado para o terminal"""
34     print "Temperatura:"
35     print "%2f" % tcelsius, u"\u00B0C"#celsius

```

```

33     print "%.2f" % (tcelsius*1.8+32), u"\u00B0F"#
34         fahrenheit
35     print "%.2f K" % (tcelsius+273.15)#kelvin
36     print "%.2f" % (tcelsius*1.8+32+459.67), u"\u00B0R"#
37         rankine
38
39 def tlog(file = "/var/www/datalog/default.csv"):
40     """salva temperatura e Unix Time em .csv """
41     #arquivo padrão CSV com cabecalho
42
43     #file = raw_input("digite o nome do arquivo e.g. log-data\n")
44     #file += ".log"
45     #tsample = float(raw_input("digite tempo de amostragem em
46     #segundos: "))
47
48     #os comandos acima foram ignorados para poder usar o nohup
49     #que não recebe input)
50     tsample = 0.2202 #amostra a cada x segundos
51     #amostras = 5000#numero de amostras a serem coletadas
52     buff_temp = tread()#guarda o último valor lido
53     exist = os.path.isfile(file)
54     #buffer = open(file,"a")#mesma função da linha abaixo,
55     #porem menos recomendada
56     with open(file, 'a', 1) as log:#desse jeito, o arquivo será
57         fechado
58     #mesmo que haja uma excessao, diferente de usar file.close
59     #()
60     #o arg. 1 indica que o buffer antes de escrever no arquivo
61     #eh 1 linha
62     if exist is False:#se vai criar o arquivo agora
63         log.write("temperatura,data\n")
64     while True: #loop infinito
65         #while amostras > 0:#coleta o numero de amostras
66         #amostras = amostras - 1#decrementa variavel de
67             controle
68         temp_celsius = tread()
69         epoch = time.time()#lê a data/hora do sistema
70         #nowis = time.ctime(epoch)#converte para string
71         if temp_celsius >= 0:#evita leitura errada
72             #essa leitura errada é intermitente e causa
73                 desconhecida
74             log.write("%f,%f\n" % (temp_celsius, epoch))
75             tlog_instant(temp_celsius, epoch)#escreve arquivo com
76                 ultima leitura
77             #graph.graph_gen()# atualiza gráfico da temperatura
78             #escreve temperatura e data/hora no .txt
79             time.sleep(tsample)#espera n segundos
80             print "registrando temperatura!",temp_celsius
81
82
83
84 def tlog_instant(temperature, epoch, file = "/var/www/datalog
85 /instant.csv"):
86     """salva última temperatura e Unix Time em arquivo .csv"""
87     with open(file, 'w', 1) as log:#sobrescreve o arquivo toda
88         vez

```

```

77 #temperature = tread()#lê a temperatura
78 #epoch = int(time.time())#lê o Unix Time do sistema
79 log.write("temperatura,data\n")
80 log.write("%f,%f\n" % (temperature, epoch))
81
82 def tlog_test(file = "/var/www/datalog/default.csv"):
83     """salva temperatura e Unix Time em .csv"""
84     #arquivo padrão CSV com cabecalho
85     tsample = 1 #valor de amostragem para teste
86     exist = os.path.isfile(file)
87     with open(file, 'a', 1) as log:#desse jeito, o arquivo será
88         fechado
89     if exist is False:#se vai criar o arquivo agora
90         log.write("temperatura,data\n")
91     temp_celsius = 25.00#somente para teste
92     temp_sparge = 20.00#somente para teste
93     ramp_section = 0#sequencia de funcoes que gera as rampas/
94         degraus
95     while True: #loop infinito
96         if ramp_section == 0:#aquece medio ate 29 graus
97             temp_celsius += 0.2
98             if temp_celsius >= 29:
99                 ramp_section = 1
100            elif ramp_section == 1:#aquece devagar ate 30 graus
101                temp_celsius += 0.1
102                if temp_celsius >= 30:
103                    ramp_section = 2
104                elif ramp_section == 2:#espera usuario adicionar maltes
105                    time.sleep(15)#30 segundos
106                    ramp_section = 3
107                elif ramp_section == 3:#aquece medio ate 35 graus
108                    temp_celsius += 0.2
109                    if temp_celsius >= 35:
110                        ramp_section = 4
111                    elif ramp_section == 4:#retrai aquecimento ate 33 graus
112                        temp_celsius -= 0.2
113                        if temp_celsius <= 33:
114                            ramp_section = 5
115                        elif ramp_section == 5:#degrau 1,
116                            aquece sparge durante 1 minuto
117                            temp_sparge += 0.5
118                            if temp_sparge >= 50:
119                                ramp_section = 6
120                            elif ramp_section == 6:#aquece rapido ate 42 graus
121                                temp_celsius += 0.3
122                                if temp_celsius >= 42:
123                                    ramp_section = 7
124                                elif ramp_section == 7:#aquece medio ate 59 graus
125                                    temp_celsius += 0.2
126                                    if temp_celsius >= 59:
127                                        ramp_section = 8
128                                    elif ramp_section == 8:#aquece devagar ate 60 graus
129                                        temp_celsius += 0.1
130                                        if temp_celsius >= 60:
131                                            ramp_section = 9
132
133 else:#temperatura de fervura
134     temp_sparge += 0.3

```

```
132 epoch = time.time()#lê a data/hora do sistema
133 log.write("%f,%f\n" % (temp_celsius, epoch))
134 tlog_instant(temp_celsius, epoch)#escreve arquivo com
135         ultima leitura
136 tlog_instant(temp_spurge, epoch, "/var/www/datalog/
137         instant_bk.csv")#somente para teste por enquanto
138 #escreve temperatura e data/hora no .txt
139 time.sleep(tsample)#espera n segundos
140 print "registrando temperatura!",temp_celsius,
141         temp_spurge
```

Código-fonte B.1: Módulo com as funções para *log* da temperatura

B.2 Script para plotagem de gráfico

```

35     title += time.ctime(float(last_line[1])) #pega a data/hora
36         da ultima linha
37     #nao da pra fazer slice do tipo [n:], porque nao exclui o
38         \n do fim da linha
39
40     #plota o grafico da temperatura
41     fig = plt.figure(figsize=(9, 4)), dpi=300)
42     ax = fig.add_subplot(111) #subplot p/ mudar cor de fundo
43     ax.set_axis_bgcolor('#BEC5C2') #muda cor do fundo do plot
44     plt.grid() #plota o grafico com grid ligado
45     #se tratando de salvar a figura, o dpi so faz diferenca no
46         savefig
47
48     #ajusta a escala do tempo baseado no tempo total de
49         amostragem dos dados
50     size_x = int(float(ttimes[len(ttimes)-1]))-float(ttimes[0])
51         #tempo final menos tempo inicial
52     if size_x < 7200:#ate duas horas, escala em minutos
53         for tms in ttimes:#itera pelo array do eixo-x
54             x_scale.append((float(tms)-float(ttimes[0]))/60) #
55                 transforma em minutos
56             plt.xlabel('tempo(minutos)')#imprime label indicando
57                 minutos
58     elif size_x < 129600:#ate um dia e meio, escala em horas
59         for tms in ttimes:#itera pelo array do eixo-x
60             x_scale.append((float(tms)-float(
61                 ttimes[0]))/(60*60))#transforma em
62                 horas
63             plt.xlabel('tempo(horas)')#imprime label indicando horas
64     elif size_x < 7776000:#ate tres meses, escala em dias
65         for tms in ttimes:#itera pelo array do eixo-x
66             x_scale.append((float(tms)-float(
67                 ttimes[0]))/(60*60*24))#transforma em
68                 horas
69             plt.xlabel(u'tempo(dias)')#imprime label indicando dias
70     else:#caso contrario, escala em meses
71         for tms in ttimes:#itera pelo array do eixo-x
72             x_scale.append((float(tms)-float(
73                 ttimes[0]))/(60*60*24*30))#
74                 transforma em horas
75             plt.xlabel(u'tempo(meses -> 30 dias por m\N{COMBINING ACUTE ACCENT}')#imprime label indicando meses
76
77     #gerando interpolacao
78     #tvalues = np.array(tvalues)
79     #x_smooth = np.linspace(x_scale.min(), x_scale.max(), 10)
80     #y_smooth = spline(x_scale, tvalues, x_smooth)
81     #tck = interpolate.splrep(x_scale, tvalues)
82     #print tck
83
84     #plotando os graficos
85     plt.plot(x_scale, tvalues, '-', color='b')
86     #plt.plot(ttimes, tvalues, '-', color='b')
87
88     #trabalha as legendas
89     # tiks = ax.get_xticks().tolist()#le o eixo das legendas
90     # size_tks = len(tiks)#ve quantas posicoes tem pra legenda

```

```

78 | # for lgd in tiks:#itera essas posicoes
79 |     legenda.append(x_scale(size_tks))
80 | tiks[:]=['bla','ble','bli','blo','blu','bal','bel','bil','
81 |     bol','bul']
82 | # print tiks[1]
83 | ax.set_xticklabels(tiks)
84 | #ax.axis_date()
85 | #plt.tight_layout(pad=4.0, w_pad=0.5, h_pad=1.0)
86 | #plt.plot(x_smooth, y_smooth, '-', color='r')
87 | plt.title(title)
88 | plt.ylabel(u'temperatura (\u00B0C)')
89 | plt.savefig(graph, dpi=150, facecolor='none')#='#97D9CC')
90 | plt.close()
91 |
92 | while True:
93 |     delta_t = time.time()#guarda tempo inicial
94 |     graph_gen()
95 |     delta_t = time.time() - delta_t#calcula tempo total
96 |     print 'tempo para gerar grafico (s) = %.3f' %delta_t
         time.sleep(300)

```

Código-fonte B.2: Script para plotagem de gráfico

Apêndice C

Códigos-fonte Node.js

C.1 Código-fonte raiz da aplicação em Node.js

```

1 "use strict";
2 //load modules
3 var bodyParser = require('body-parser');
4 var express = require("express");
5 var app = express();
6 var debug = require('debug')('controle');
7 var pru = require("node-pru-extended");
8 var fs = require("fs");
9 var exec = require("child_process").exec;
10 var phpExpress = require('php-express')({ // assumes php is
    in your PATH
11 // must specify options hash even if no options provided!
12   binPath: 'php'
13 });
14 //load modules written for this application
15 var gpioCfg = require('./my_node_modules/gpio_cfg.js');
16 var ctrl = require('./my_node_modules/ctrl.js');
17 var log = require('./my_node_modules/log_check_misc.js');
18 var routes = require('./my_node_modules/routes.js');
19
20 //global variables that should also be saved to a backup file
21 //periodically
22 global.environmentVariables = {
23   warn: "", //if not empty holds some warning message
24   code: "", //tells the same as msg, but as an index, easier
25   // to check programatically
26   tmpMT: "", //mash tun temperature
27   tmpMTsetp: "", //mash tun current setpoint
28   tmpBK: "", //brewing kettle temperature, also the "hot
29   // liquor tank" for sparging
30   tmpBKsetp: "", //brewing kettle/hot liquor tank current
31   // setpoint
32   timeLeft: "", //helping variable to tell the client the time
33   // left for the step rests or the boil, etc
34   readyForNextStep: false, //set whenever the system is ready
35   // for the next step

```

```

30     auto: true, //whether the process is running automatically
31     or there is human intervention
32   processFail: false, //flag is set if the process fails
33     irreversibly
34   msg: "", //holds some explanatory message
35   timestamps:{ //notable timestamps
36     start: "", //epoch time of the first request to start a
37       recipe
38     startHeating: "", //epoch time of the start to heat the
39       mash water
40     finishHeating: "", //epoch for the finish of the heating
41       of mash water
42   //start of the ramps
43   sRamp0: "", sRamp1: "", sRamp2: "", sRamp3: "", sRamp4: ""
44     , sRamp5: "", sRamp6: "", sRamp7: "",
45   //finish of the ramps
46   fRamp0: "", fRamp1: "", fRamp2: "", fRamp3: "", fRamp4: ""
47     , fRamp5: "", fRamp6: "", fRamp7: "",
48   startDrain: "", //start of the sparging process, when the
49     mash is parcially drained to the BK
50   startSparge: "", //here the recirculation pump starts
51     working
52   heatingBoil: "", //started to heat the wort after sparging
53   boilStart: "", //time when temperature is near enough
54     boiling (>96°C)
55   boilFinishScheduled: "", //time when the boil is scheduled
56     to finish
57   curr: "", //epoch time of the current variables state
58 },
59   ioStatus: gpioCfg.all_io, //also records the IO status
60   okToStart: false, //true if a recipe is ok enough to start
61     a production
62   recipe: "" //recipe name
63 };
64 */
65 //Trying to access and/or use the PRU
66 console.log(pru);
67 pru.init(); //initialize the communications
68 //pru.loadDatafile(0, "/var/www/myPRUcodes/data.bin");
69 pru.execute(0, "/var/www/myPRUcodes/pisca_text.bin", 0);
70 //console.log(pru.getSharedRAM());
71 pru.exit(0); //force the PRU code to terminate
72 */
73
74 // Pin configuration
75 debug(gpioCfg.ioStatus.total + " pins being used: ", gpioCfg.
76   all_io_pins.toString());
77 debug("Configuring pins...");
```

gpioCfg.pinsConfiguration();

//Using Express to create a server

app.use(bodyParser.urlencoded({ //to support URL-encoded bodies, MUST come before routing

extended: **true**

}));

// set view engine to php-express

app.set('views', './');

```

74 app.engine('php', phpExpress.engine);
75 app.set('view engine', 'php');
76
77 // routing all .php file to php-express
78 app.all(/.+\.php$/, phpExpress.router);
79
80 app.use(express.static(__dirname)); //add the directory where
81     // HTML and CSS files are
82 var server = app.listen(8587, "192.168.1.155", function () {
83     //listen at the port and address
84     var host = server.address().address;
85     var port = server.address().port;
86     var family = server.address().family;
87     debug('Express server listening at http://%s:%s %s', host,
88           port, family);
89 });
90
91 app.route('/controle') //used to unite all the request types
92     // for the same route
93     .post(routes.controlRoute);
94
95 app.route('/startrecipe') //used to unite all the request types
96     // for the same route
97     .post(routes.startrecipeRoute);
98
99 app.route('/config') //used to unite all the request types for
100    // the same route
101    .post(routes.configRoute);
102
103 app.route('/clientrequest') //used to unite all the request
104    // types for the same route
105    .post(routes.clientrequestRoute);

```

Código-fonte C.1: Código-fonte raiz da aplicação em Node.js

C.2 Módulo de gerenciamento de GPIO e PWM

```

1 "use strict";
2
3 //var m = require('../controle.js');//not sure it is needed
4 // to add the requester to share variables
4 var b = require('octalbonescript');
5 var debug = require('debug')('gpio');
6
7 //Variables
8 var led = module.exports.led = new PinObjectIO("USR1");
9 var mash_pump = module.exports.mash_pump = new PinObjectIO(
10     "P8_07");
11 var boil_pump = module.exports.boil_pump = new PinObjectIO(
12     "P8_08");
11 var pumps = module.exports.pumps = {mash_pump:mash_pump,
13     boil_pump:boil_pump};
12
13 var mash_valve = module.exports.mash_valve = new PinObjectIO(
14     "P8_09");

```

```

14 | var boil_valve = module.exports.boil_valve = new PinObjectIO(
15 |   "P8_10");
16 | var chill_valve = module.exports.chill_valve = new
17 |   PinObjectIO("P8_11");
18 | var water_valve = module.exports.water_valve = new
19 |   PinObjectIO("P8_12");
20 | var valves = module.exports.valves = { mash_valve:mash_valve,
21 |   boil_valve:boil_valve,
22 |   chill_valve:chill_valve, water_valve:water_valve};
23 |
24 | //for servo, use 0.0325 < duty < 0.11 to protect servo
25 | //integrity
26 | var servo_pwm = module.exports.servo_pwm = {id:"P8_19", state
27 |   :{duty:0.11, freq:50}, cfg:b.ANALOG_OUTPUT}; //state of PWM
28 | //is duty-cycle and frequency
29 | //var servo_pwm = {id:"P8_19", state:{duty:0.11, freq:60}};///
30 | //this is for the SSR test
31 |
32 | var all_io = module.exports.all_io = collect(pumps, valves,
33 |   heaters, {led:led}, {servo_pwm:servo_pwm});
34 | //debug(all_io);
35 | var all_io_objects = module.exports.all_io_objects = Object.
36 |   keys(all_io); //get all the keys, because cannot access
37 |   object by index, e.g all_io[2]
38 | var all_io_pins = module.exports.all_io_pins = [];//all the
39 |   pins used as an array
40 | for (var i = 0; i < all_io_objects.length; i++){//get the
41 |   pins one by one
42 |   all_io_pins[i] = all_io[all_io_objects[i]].id; //and add to
43 |   the array
44 | }
45 |
46 | var ioStatus = module.exports.ioStatus = {cfgok:0, gpio:0,
47 |   pwm:0, analog:0, interrupt:0, total:all_io_objects.length,
48 |   //helping object
49 |   //some functions just to exercise the use of methods
50 |   newGpio:    function(){ this.cfgok++; this.gpio++; },
51 |   newPwm:     function(){ this.cfgok++; this.pwm++; },
52 |   newAnalog:  function(){ this.cfgok++; this.analog
53 |     ++; },
54 |   newInterrupt: function(){ this.cfgok++; this.analog
55 |     ++; },
56 |   gpio2pwm:   function(){ this.gpio--; this.pwm++; },
57 |   gpio2interrupt: function(){ this.gpio--; this.
58 |     interrupt++; },
59 |   pwm2gpio:   function(){ this.pwm--; this.gpio++; },
60 |   pwm2interrupt: function(){ this.pwm--; this.
61 |     interrupt++; },
62 |   interrupt2gpio: function(){ this.interrupt--; this.
63 |     gpio++; },
64 | }
```

```

47     interrupt2pwm: function() { this.interrupt--; this.
48     }
49
50 //functions
51 module.exports.changeStatusIO = function (pin, val) {
52   if(val == "true"){//if button is checked
53     all_io[pin].state = b.HIGH;//turn corresponding pin HIGH
54     b.digitalWriteSync(all_io[pin].id, all_io[pin].state);
55     debug("Pin " + pin + " turned HIGH");
56   }
57   else if(val == "false"){//if button is unchecked
58     all_io[pin].state = b.LOW;//turn corresponding pin LOW
59     b.digitalWriteSync(all_io[pin].id, all_io[pin].state);
60     debug("Pin " + pin + " turned LOW");
61   }
62   else{//if it is not a button
63     if(pin == "servo_pwm"){//check if this is the PWM
64       servo_pwm.state.duty = 0.0325 + (0.0775/180)*val;//turn
65       degree to duty-cycle
66       b.analogWrite(servo_pwm.id, servo_pwm.state.duty,
67         servo_pwm.state.freq, function(err_wr){//set PWM
68         if(err_wr){//if anything goes wrong
69           debug(err_wr);//print the error
70         }
71         else{//otherwise
72           debug("Servo angle set to: " + val);//print the
73           angle the PWM was set to
74         }
75       });
76     }
77   }
78 }
79
80 module.exports.getSystemStatus = function () {
81   var all_io_status = {};//object with all the pin pairs key:
82   value
83   for(var i = 0; i < all_io_objects.length; i++){//get the
84     pair key:value pin by pin
85     all_io_status[all_io_objects[i]] = all_io[all_io_objects[i]].state;
86   }
87   return(all_io_status);
88 }
89
90 // I/O pins
91 function PinObjectIO(pinId){//function to create pin object,
92   should receive pin ID
93   if(pinId){//if the variable is passed to function or not
94     empty
95     this.id = pinId; this.state = b.LOW; this.cfg = b.OUTPUT;
96   }
97   else{//if no variable is passed or passed empty
98     debug("No variable passed to create pin object");
99   }
100 }
101
102 module.exports.pinsConfiguration = function () {
103 }
```

```

96  for (var i = 0; i < all_io_objects.length; i++) { //set all
97    pins as outputs
98    (function (pinIndex) { //need to create a scope for the
99      current pin variable, because it is asynchronous
100     debug("      pin passed is " + this + "; pin index passed
101       is " + pinIndex); //this" is the first parameter
102       passed -> the current pin
103     b.pinMode(this, all_io[all_io_objects[pinIndex]].cfg,
104       function (err, pin) { //configure and callback function
105         if (err) //if by the end of executing pinMode function,
106           there is an error
107           console.error(err.message); //then the error is
108           printed
109         else { //initial state, everyone LOW because HIGH state
110           means ON
111           if (all_io[all_io_objects[pinIndex]].cfg == b.OUTPUT
112             ) { //if pin is configured as digital output
113               debug('      pin ' + pin + ' ready[OUTPUT], ' +
114                 ioStatus.cfgok+1) + "/" + ioStatus.total + "
115                 pins configured");
116               b.digitalWriteSync(pin, all_io[all_io_objects[
117                 pinIndex]].state); //state is LOW because of
118                 initial values
119               //ioStatus.cfgok++;
120               ioStatus.newGpio(); //indicates one more pin is
121                 configured as gpio
122               if (ioStatus.cfgok == ioStatus.total) { //if all
123                 pins are already configured
124                 //interval = setInterval(function () { ioTest () },
125                   5000); //then start the blinking test very
126                   slow
127                   debug(ioStatus.gpio + " pins as GPIO, " +
128                     ioStatus.pwm + " as PWM, " +
129                     ioStatus.analog + " as ANALOG, " + ioStatus.
130                     interrupt + " as INTERRUPT");
131                   }
132                 }
133               else if (all_io[all_io_objects[pinIndex]].cfg == b.
134                 ANALOG_OUTPUT) { //if pin is configured as PWM
135                 debug('      pin ' + pin + ' ready[PWM], ' +
136                   ioStatus.cfgok+1) + "/" + ioStatus.total + "
137                   pins configured");
138                 b.analogWrite(servo_pwm.id, servo_pwm.state.duty,
139                   servo_pwm.state.freq, function (err_wr) { //try
140                     to configure
141                     if (err_wr) { //if error
142                       debug(err_wr);
143                     }
144                   else { //if ok
145                     ioStatus.newPwm(); //indicates one more pin is
146                     configured as PWM
147                     if (ioStatus.cfgok == ioStatus.total) { //if all
148                       pins are already configured
149                       //interval = setInterval(function () { ioTest
150                         () }, 5000); //then start the blinking
151                         test very slow
152                         debug(ioStatus.gpio + " pins as GPIO, " +
153                           ioStatus.pwm + " as PWM, " +
154                           ioStatus.analog + " as ANALOG, " +
155                           ioStatus.interrupt + " as INTERRUPT");
156                         }
157                       }
158                     }
159                   }
160                 }
161               }
162             }
163           }
164         }
165       }
166     }
167   }
168 }
```

```

125             ioStatus.analog + " as ANALOG, " +
126             ioStatus.interrupt + " as INTERRUPT");
127         }
128     });
129 }
130 }
131
132 //everyone is output
133 //get all the object keys as vector and use it to access
134 //all the object keys
135 //).call(all_io[all_io_objects[i]].id,i); //passes pin as "
136 //this" and index as pinIndex, only works properly in
137 //strict mode
138 }
139 };
140
141 module.exports.ioTest = function () {
142     /*This function do some blinking thing, it should be called
143      inside setInterval(),like:
144      setInterval(ioTest, interval); //period = interval*IOpins =
145      500ms*9 = 4.5s*/
146     var on_count = 0; //number of io turned b.HIGH
147     var last_on; //number of last io b.HIGH
148
149     for(var i = 0; i < all_io_objects.length; i++){ //check all
150         if(all_io[all_io_objects[i]].state == b.HIGH){ //if HIGH
151             on_count++; //one more io HIGH
152             last_on = i; //this is the last io HIGH
153         }
154
155         if(on_count == 1){ //if only one IO HIGH
156             if(last_on == 8){ //if last pin is the one HIGH
157                 //debug('Activating pin 0');
158                 all_io[all_io_objects[last_on]].state = b.LOW; //turn it
159                 LOW
160                 all_io[all_io_objects[0]].state = b.HIGH; //and next one
161                 HIGH, which means the first
162                 b.digitalWriteSync(all_io[all_io_objects[last_on]].id,
163                     all_io[all_io_objects[last_on]].state);
164                 b.digitalWriteSync(all_io[all_io_objects[0]].id, all_io
165                     [all_io_objects[0]].state);
166             }
167             else{ //if any pin but the last is the one HIGH
168                 //debug('Activating pin ' + (last_on+1));
169                 all_io[all_io_objects[last_on]].state = b.LOW; //turn it
170                 LOW
171                 all_io[all_io_objects[last_on+1]].state = b.HIGH; //and
172                 next one HIGH
173                 b.digitalWriteSync(all_io[all_io_objects[last_on]].id,
174                     all_io[all_io_objects[last_on]].state);
175                 b.digitalWriteSync(all_io[all_io_objects[last_on+1]].id
176                     , all_io[all_io_objects[last_on+1]].state);
177             }
178         }
179     }
180     else{ //if there is more than one IO HIGH
181         for(i = 1; i < all_io_objects.length; i++) {

```

```

168     all_io[all_io_objects[i]].state = b.LOW; //turn all but
169     first LOW
170     b.digitalWriteSync(all_io[all_io_objects[i]].id, all_io
171         [all_io_objects[i]].state);
172     }
173     all_io[all_io_objects[0]].state = b.HIGH; //turn first
174     HIGH
175     b.digitalWriteSync(all_io[all_io_objects[0]].id, all_io[
176         all_io_objects[0]].state);
177     }
178     }
179     function collect() {//function concat objects
180         var ret = {};//the new object
181         var len = arguments.length;//the total number of objects
182             passed to collect
183         for (var i=0; i<len; i++) {//do it for every object passed
184             for (var p in arguments[i]) {//iterate the i-eth object
185                 passed
186                 if (arguments[i].hasOwnProperty(p)) {//whenever there
187                     is a property
188                     ret[p] = arguments[i][p];//add the property to the
189                         new object
190                 }
191             }
192         }
193         return ret;
194     }

```

Código-fonte C.2: Módulo de gerenciamento de GPIO e PWM

C.3 Módulo de roteamento do servidor web

```

1 'use strict';
2
3 var debug = require('debug')('routes');
4 var exec = require('child_process').exec;
5 var fs = require('fs');
6 var gpioCfg = require('./gpio_cfg.js');
7 var log = require('./log_check_misc.js');
8 var ctrl = require('./ctrl.js');
9
10 module.exports.controlRoute = function (req, res) {
11     var serverResponse = {command:req.body.command, btn:req.
12         body.btn, val:req.body.val};
13     var command = req.body.command, pin = req.body.btn, val =
14         req.body.val;
15
16     if(command == "pinSwitch"){//if command passed by client is
17         to switch a pin configuration
18         gpioCfg.changeStatusIO(pin, val);//change the IO status
19             according to the data received
20         res.send(serverResponse);//echo the received data to the
21             server
22     }

```

```

18  else if(command == "getStatus") {
19      global.environmentVariables.ioStatus = gpioCfg.all_io; // 
20          all of the pins status
21      res.send(global.environmentVariables);
22  }
23
24 module.exports.startrecipeRoute = function (req, res) {
25     var serverResponse = {resp:"success"};
26     var command = req.body.command;
27     debug("Command: " + command);
28     var recipesPath = "./recipes"; //path to the recipes
29         directory
30     var bkpFile = "./datalog/backup.log"; //path to the backup
31         logs file
32     var lockFile = "./datalog/lockfile"; //file that tells if
33         recipe is running
34     var recipeName ;
35     if(command == "getRecipes"){//if command passed by client
36         is to get the recipe names
37         log.sendRecipeNames(recipesPath, res); //get it and return
38             to the client
39     }
40     else if(command == "startRequest"){//if the client wants to
41         start a recipe
42         recipeName = req.body.recipe + ".recipe"; //get the recipe
43             name
44         log.checkRecipeIntegrity(recipeName, recipesPath, res);
45     }
46     else if(command == "startRecipe"){//start the recipe;
47         should only be requested after the "startRequest"
48         command
49         recipeName = req.body.recipe + ".recipe"; //recipe name
50             sent from the client
51         ctrl.startMashingProcess(recipeName, res, lockFile,
52             recipesPath, function(err, nextStep){
53             if(err){
54                 debug("Ops, something wrong. It's so annoying that
55                     the error isn't explained here, isn't it?");
56                 return;
57             }
58             debug(nextStep); //here the next step, i.e. controlling
59                 the mash steps, should be called
60         }); //start the production
61     }
62     else if(command == "inProgress"){//checks if there is a
63         recipe running
64         fs.readFile(lockFile, function(err, data){
65             if(err){
66                 serverResponse.resp = "false"; //assumes the error
67                     means no recipe is in progress
68                 res.send(serverResponse);
69             }
70             else{
71                 if(+data == 1){ //if there is a recipe in progress
72                     serverResponse.resp = "true";
73                     fs.readFile(bkpFile, "utf-8", function(err, data){
74

```

```

59         if(err){//if contents of log could not be
60             retrieved
61             serverResponse.recipe = "unknown";//not the
62             best thing to do
63             res.send(serverResponse);
64         }
65     else{
66         var lines = data.trim().split('\n');//separate
67             line by line
68             debug(lines.slice(-1)[0]);
69             var lastLine = lines.slice(-1)[0];//get the
70             last log line
71             var properties = JSON.parse(lastLine);//JSON to
72             object
73             properties.recipe = properties.recipe.replace(".
74                 .recipe", ""); //remove the file extension
75             properties.recipe = properties.recipe.replace(/_
76                 /g, " ");//replace underscores with spaces
77             serverResponse.recipe = properties.recipe;//
78                 send recipe name to the client
79             debug(serverResponse);
80             res.send(serverResponse);
81         }
82     }
83     else{
84         serverResponse.resp = "false";
85         res.send(serverResponse);
86     }
87 }
88 };
89 module.exports.configRoute = function (req, res) {
90     var request = req.body.request;
91     var nd = req.body.newdate;
92     var serverResponse = {datetime:""};
93     if(request == "datetime"){//send the system time/date to
94         the client
95         serverResponse.datetime = Date();
96         res.send(serverResponse);
97     }
98     else if(request == "setdatetime"){//change the system time/
99         date
100        exec('sudo date -s ' + nd + ' | hwclock -w',//execute
101            shell command
102        function(error, stdout, stderr) {
103            debug('stdout: ' + stdout);
104            debug('stderr: ' + stderr);
105            if (error !== null) {//if new date wasn't correctly set
106                debug('exec error: ' + error); //print error
107            }
108        }
109    }
110 }
111 
```

```

105     serverResponse.datetime = "error";//send fail to
106     client
107     res.send(serverResponse);
108 } //if date was correctly set, send ok to client
109 debug("date changed to: " + Date(nd)); //print
110 serverResponse.datetime = "ok";
111 res.send(serverResponse);
112 }
113 );
114 }
115 };
116
117 module.exports.clientrequestRoute = function(req, res) {
118     var command = req.body.command;
119     var serverResponse = {resp:"success"};
120     debug("Client request: " + command);
121     if(command == "startMashRamp"){//set flag to tell ctrl.
122         startMashingProcess() to go to the next step
123         global.environmentVariables.readyForNextStep = true;
124         res.send(serverResponse);
125     }
126     else{
127         serverResponse.resp = "fail";
128     }
129 };

```

Código-fonte C.3: Módulo de roteamento do servidor web

C.4 Módulo de registros e verificações em geral

```

1 'use strict';
2
3 var debug = require('debug')('log');
4 var fs = require('fs');
5 var spawn = require("child_process").spawn;
6
7 //variables
8 var temperatureLogHandler;//variable to handle the python "log.py" script
9
10 module.exports.startTemperatureLogging = function () {
11     //starts the python script that logs temperature to file
12     //sudo kill $(ps aux | grep log.py | grep -v grep | awk '{ print $2}') //to stop the process from terminal
13     fs.unlink("./datalog/instant.csv", function(err){//try to
14         delete the old last saved value
15         if(err){//this may happen if the old log was already
16             deleted
17             debug("file could not be deleted!"); //probably nothing
18             to worry about
19         }
20     //start the logging process anyway
21 };

```

```

18     temperatureLogHandler = spawn("python", ["/home/debian/
19         brewing/log_test.py"]); //starts to log
20     debug("Temperature logging started!");
21     temperatureLogHandler.on('close', function(code, signal) {
22         debug("Temperature logging stopped!");
23     });
24   );
25
26 module.exports.stopTemperatureLogging = function() {
27   debug("Stopping the temperature logging!");
28   temperatureLogHandler.kill('SIGHUP'); //kill the process
29   and stop logging
30 };
31 module.exports.getTemperatureReading = function (logFilePath,
32   logReadHandler, callback){
33   // var logReadHandler = {
34   //   lastValidTemperature: 0,//the last valid temperature
35   //   reading (defaults to zero, not the better solution)
36   //   lastReadingsTimestamp: [0, 0, 0, 0, 0, 0],//last 5
37   //   timestamps, to check if readings are going ok
38   //   errorCount: 0//counts the file reading errors
39   // };
40   fs.readFile(logFilePath, "utf-8", function(err, data){ ///
41     gets the most recent temperature reading
42     var parsedData ;//variable to hold the relevant data (
43       temperature and its timestamp)
44     var callbackError;
45     if(err){ //could not read temperature
46       logReadHandler.errorCount++; //increment the errorCount
47       variable
48       if(logReadHandler.errorCount >= 180){ //180 arbitrarily
49         chosen - it is 5% of 1 hour logging readings
50         //oh my god thigs are bad! User, you must take over
51         from here
52         callbackError = "Too many temperature reading errors,
53           something may be going awry!";
54         debug("Too many temperature reading errors, something
55           may be going awry!");
56         callback(err, null);
57       }
58     } else{
59       callbackError = "Single wrong reading, nothing to
60         worry yet.";
61       debug("Single wrong reading, nothing to worry yet.");
62       callback(err, null);
63     }
64   } //if temperature reading from file was successful
65   parsedData = data.trim().split('\n').slice(-1)[0].split
66   (',');
67   if((parsedData[0] == "temperature") || (typeof
68     parsedData[0] != "string") || isNaN(+parsedData[0]))
69   { //wrong reading beacuse of wrong logging
70     debug("Wrong temperature reading: " + parsedData[0]);
71     parsedData[0] = logReadHandler.lastValidTemperature;
72     //then use the last valid temperature reading

```

```

59     logReadHandler.errorCount++; //increment the
60         errorCount variable
61     if(logReadHandler.errorCount >= 180){ //180
62         arbitrarily chosen - it is 5% of 1 hour logging
63         readings
64         //oh my god thigs are bad! User, you must take over
65         from here
66         callbackError = "Too many temperature reading
67             errors, something may be going awry!";
68         debug("Too many temperature reading errors,
69             something may be going awry!");
70         callback(err, null);
71     }
72 } else{
73     callbackError = "Single wrong reading, nothing to
74         worry yet.";
75     debug("Single wrong reading, nothing to worry yet.");
76     callback(err, null);
77 }
78 } else{
79     parsedData[0] = +parsedData[0]; //string to number
80     logReadHandler.lastValidTemperature = +parsedData[0];
81         //save the last valid temperature reading
82 }
83 logReadHandler.lastReadingsTimestamp[4] = parsedData
84     [1]; //updates last temperature reading timestamp
85 for(var i = 0; i < 4; i++){ //updates the older
86     timestamps also
87     logReadHandler.lastReadingsTimestamp[i] =
88         logReadHandler.lastReadingsTimestamp[i+1];
89 }
90 if(logReadHandler.lastReadingsTimestamp[0] >=
91     logReadHandler.lastReadingsTimestamp[4]){ //if the
92     reading is the same within 4s
93     //hey user check this, because something may be going
94     awry!
95     callbackError = "Log not updating, check temperature
96         probe connection!";
97     debug("Log not updating, check temperature probe
98         connection!");
99     logReadHandler.errorCount++; //increment the
100        errorCount variable
101 if(logReadHandler.errorCount >= 180){ //180
102         arbitrarily chosen - it is 5% of 1 hour logging
103         readings
104         //oh my god thigs are bad! User, you must take over
105         from here
106         callbackError = "Too many temperature reading
107             errors, something may be going awry!";
108         debug("Too many temperature reading errors,
109             something may be going awry!");
110     }
111     callback(err, null);
112 }
113 else{//then we can act in order to get to the
114     temperature setpoint

```

```

93     //callback second argument is the temperature
94     callback(null, parsedData[0]); //successfully got the
95         temperature reading
96     }
97   );
98 };
99
100 module.exports.checkRecipeIntegrity = function (recipe, path,
101   res) {
102   var serverResponse = {resp: "success", warn: "", err: ""};
103   //tells the client if everything is ok
104   var recipeContents ;
105   global.environmentVariables.recipe = recipe; //save the
106   recipe name
107   fs.readFile(path + "/" + recipe, function(err,
108     recipeFileContents){
109     var okToStartFlag = 1;
110     if(err){//if file contents could not be retrieved
111       serverResponse.resp = "couldntReadFile"; //tells the
112         client
113       res.send(serverResponse);
114     }
115     else{//if file was successfully read
116       fs.readFile("./datalog/lockfile", function(err,
117         lockFileContents){
118         if(err){//if lockfile could not be read for some
119           reason
120           serverResponse.resp = err;
121           //res.send(serverResponse);
122           okToStartFlag = 0; //unable to start the recipe
123           debug("couldn't read lockfile, unable to start
124             recipe.");
125           return;
126         }
127         if(+lockFileContents == 0){ //if there is no recipe in
128           progress
129           debug("ready to rock!");
130           recipeContents = recipeFileContents.toString("UTF8")
131             .split("\n"); //split contents to array
132           for(var i = 0; i < recipeContents.length; i++){ //
133             get only the relevant data
134             recipeContents[i] = recipeContents[i].split(' ')
135               [1];
136             if(!recipeContents[i]){//if some recipe line is
137               blank
138               switch(i){ //and this line is important, then:
139                 case 0://recipe name not set (warning)
140                   serverResponse.warn += "nome da receita; ";
141                   break;
142                 case 1://beer style not set (warning)
143                   serverResponse.warn += "estilo; ";
144                   break;
145                 case 2://beer yeast not set (warning)
146                   serverResponse.warn += "levedura; ";
147                   break;
148                 case 3://mash water not set (error)
149               }
150             }
151           }
152         }
153       }
154     }
155   }
156 }
```

```

136         serverResponse.err += "água de brassagem; "
137         ;
138         global.environmentVariables.okToStart =
139             false;
140         okToStartFlag = 0;
141         break;
142     case 4://sparging water not set (warning)
143         serverResponse.warn += "água de sparging; "
144         ;
145         break;
146     case 5://sparging water temperature not set (
147         warning)
148         serverResponse.warn += "temperatura de
149             sparging; ";
150         break;
151     case 6://boiling time not set (error)
152         serverResponse.err += "tempo da fervura; ";
153         global.environmentVariables.okToStart =
154             false;
155         okToStartFlag = 0;
156         break;
157     case 7://mash initial temperature not set (
158         error)
159         serverResponse.err += "temperatura inicial
160             de brassagem; ";
161         global.environmentVariables.okToStart =
162             false;
163         okToStartFlag = 0;
164         break;
165     case 8://mash first step not set (error)
166         serverResponse.err += "primeiro degrau de
167             temperatura; ";
168         global.environmentVariables.okToStart =
169             false;
170         okToStartFlag = 0;
171         break;
172     case 24://malt 1 not set (error)
173         serverResponse.err += "malte 1; ";
174         global.environmentVariables.okToStart =
175             false;
176         okToStartFlag = 0;
177         break;
178     case 25://malt 1 quantity not set (error)
179         serverResponse.err += "quantidade do malte
180             1; ";
181         global.environmentVariables.okToStart =
182             false;
183         okToStartFlag = 0;
184         break;
185     case 40://hop 1 not set (error)
186         serverResponse.err += "lúpulo 1; ";

```

```

178         global.environmentVariables.okToStart =
179             false;
180         okToStartFlag = 0;
181         break;
182     case 41://hop 1 quantity not set (error)
183         serverResponse.err += "quantidade do lúpulo
184             1; ";
185         global.environmentVariables.okToStart =
186             false;
187         okToStartFlag = 0;
188         break;
189     case 42://hop 1 adding time not set (error)
190         serverResponse.err += "tempo de adição do
191             lúpulo 1; ";
192         global.environmentVariables.okToStart =
193             false;
194         okToStartFlag = 0;
195         break;
196     }
197 }
198 if(okToStartFlag){//if recipe can be started
199     global.environmentVariables.okToStart = true;//
200         add this info to the global variables
201     }
202     logToFile("request to start production", 0, "start"
203         );
204     res.send(serverResponse); //answer to the client (
205         may have error msg or not)
206     debug(serverResponse);
207 }
208 else{//there is a recipe in progress, cannot start
209     //okToStartFlag = 0;
210     serverResponse.resp = "recipe in progress. Unable
211         to start mashing";
212     res.send(serverResponse); //answer error to the
213         client
214     }
215 }
216 }
217 }
218 }
219 var logToFile = module.exports.logToFile = function (message,
220     code, notableTimestamp){
221     var d = new Date();
222     var logTime = d.getTime();
223     var dataToSave ;
224     var logFile = "./datalog/backup.log";//path to the backup
225         logs file
226     global.environmentVariables.msg = message;//explanatory
227         message to be logged
228     global.environmentVariables.code = code;//code referring to
229         the message
230     global.environmentVariables.timestamps.curr = logTime;//
231         logs the request to start recipe timestamp
232     if(notableTimestamp in global.environmentVariables.
233         timestamps){//checks if variable is declared

```

```

220     debug(notableTimestamp + " is declared. Logging notable
221         timestamp");
222     global.environmentVariables.timestamps[notableTimestamp]
223         = logTime;
224 }
225 //global.environmentVariables.ioStatus = gpioCfg.all_io;//
226 // all of the pins status
227 //the above commented line may not be needed since the main
228 // script attributes one variable to the other
229 //and so one variable points to the other, not needing to
230 // refresh it. *They are pointers, not values!
231 dataToSave = JSON.stringify(global.environmentVariables) +
232     "\n";
233 debug(global.environmentVariables.msg + " - code: " +
234     global.environmentVariables.code + " - auto: " + global.
235     environmentVariables.auto);
236 if(code == 0){//if it is the first line to be logged,
237     // overwrite log file
238     fs.writeFile(logFile, dataToSave, function(err){//
239         // overwrite previous backup
240         if(err){//if was unable to log to the file
241             debug("could not write to the backup file!");//well
242                 my friend, you're on your own!
243         }
244         else{
245             debug("backup log started!");
246         }
247     });
248 }
249
250 module.exports.sendRecipeNames = function (path, res){//try
251     // to read files in directory
252     fs.readdir(path, function(err, files){
253         var deletedIndexes = new Array();// variable that holds
254             the indexes of the deleted recipes
255         var serverResponse = {resp:"success"};
256         if(err){//if something is wrong
257             debug(err);//print the error
258             serverResponse.resp = "error";
259             serverResponse.recipes = err;
260         }
261         else{
262             for(var i = 0; i < files.length; i++){//iterate the
263                 array of names

```

```
261     if(files[i].indexOf(".del") > 0){//if the server  
262         reads a deleted file  
263         deletedIndexes.push(i);  
264     }  
265     files[i] = files[i].replace(".recipe", ""); //remove  
266         the file extension  
267     files[i] = files[i].replace(/_/g, " "); //replace  
268         underlines with spaces  
269 }  
270 for(i = (deletedIndexes.length)-1; i >= 0; i--){//  
271     iterate the array of deleted recipes  
272     //debug("index: " + deletedIndexes[i]);  
273     files.splice(deletedIndexes[i],1); //deletes the file  
274         name from the array  
275 }  
276 debug("deleted recipes indexes: " + deletedIndexes);  
277 serverResponse.recipes = files;  
278 }  
279 res.send(serverResponse); //send the recipes if successful  
280     , otherwise sends the error  
281 }); //get it and return to the client  
282 };
```

Código-fonte C.4: Módulo de registros e verificações em geral

C.5 Módulo de controle do processo de brassagem

```
1 'use strict';
2
3 var debug = require('debug')('ctrl');
4 var fs = require('fs');
5 var b = require('octalbonescript');
6 var gpioCfg = require('./gpio_cfg.js');
7 var log = require('./log_check_misc.js');
8
9 module.exports.startMashingProcess = function (recipe, res,
10   lockFile, recipesPath, callback) {
11   /*Starts the mashing process, checking if everything is in
12    order, starting
13    the temperature logging and the heating of the mash water.
14    The callback function is passed as callback to the
15    heatMashWater function*/
16   var serverResponse = {resp:"success"};
17   var recipeContents ;
18   if(global.environmentVariables.okToStart){//first check is
19     if the recipe is ok to start
20     if(recipe == global.environmentVariables.recipe){//recipe
21       name should match also
22       fs.writeFile(lockFile, 1, function(err){//write to
23         lockfile telling there is a recipe in progress
24         if(err){
25           serverResponse.resp = "could not write to lockfile"
26           ;
27           res.send(serverResponse);
28           callback(serverResponse.resp,null);
29         }
30       });
31     }
32   }
33 }
```

```

22     return;
23   }
24   log.logToFile("production starting", 1); //log to file
25   fs.readFile(recipesPath + "/" + global.
26   environmentVariables.recipe, function(err, data){
27     if(err){ //if file contents could not be retrieved
28       serverResponse.resp = "couldn't read recipe file"
29       ; //tells the client
30       res.send(serverResponse);
31       callback(serverResponse.resp,null);
32       return;
33     } //if file was successfully read
34     log.startTemperatureLogging();
35     recipeContents = data.toString("UTF8").split("\n");
36     //split contents to array
37     for(var i = 0; i < recipeContents.length; i++){ ///
38       //get only the relevant data
39       recipeContents[i] = recipeContents[i].split(' ')
40       [1];
41     }
42     serverResponse.resp = "success"; //tells the client
43     //everything went alright
44     res.send(serverResponse);
45     heatMashWater(recipeContents); //start to heat the
46     //mash water
47     //The callback is called when the heat starts, not
48     //after it finishes
49     callback(null,"Started to heat mash water");
50   });
51 });
52 }
53 else{ //if recipe name doesn't match the one from "
54   startRequest"
55   serverResponse.resp = "failed";
56   res.send(serverResponse);
57   callback(serverResponse.resp,null);
58 }
59
60 function heatMashWater(recipeContents){
61   //heats the mashing water to the starting setpoint
62   var mashSetpoint = recipeContents[7];
63   var instantPath = "./datalog/instant.csv";
64   var lockFile = "./datalog/lockfile"; //file that tells if
65   //recipe is running
66   var reachedSetpoint = false; //var set to true the first
   time the checkpoint is reached
   var logReadHandler = {
     lastValidTemperature: 0, //the last valid temperature
     reading (defaults to zero, not the better solution)
     lastReadingsTimestamp: [0, 0, 0, 0, 0], //last 5
     timestamps, to check if readings are going ok
   }
}

```

```

67     errorCount: 0 //counts the file reading errors
68   };
69   global.heatingPower = 2;
70   global.environmentVariables.tmpMTsetp = mashSetpoint; //  

    stores the setpoints
71   gpioCfg.changeStatusIO("mash_pump", "true"); //turn the  

    recirculation pump on
72   log.logToFile("heating mash water", 2, "startHeating"); //  

    log to file, first time get notable timestamp
73   var logTimer = setInterval(function() { //logs the heating  

    process every ~5s
74     log.logToFile("heating mash water", 2); //log to file
75   }, 5000);
76   var readTmpTimer = setInterval(function() { //read  

    temperature every second
77     debug("Executing temperature control loop[1]");
78     log.getTemperatureReading(instantPath, logReadHandler,
      function(err, temperature){
79       if(err){
80         debug("something went wrong");
81         return;
82     }
83     updateHeatingConditions("MT", mashSetpoint, temperature
      , function(){
84       if(!reachedSetpoint){ //if it is the first time the  

        setpoint is reached
85         debug('Heating of the mash water finished. Now the  

          user must add the grains.');
86         debug("Current temperature = " +
        temperature + "\n"
        + "Reading errors = " +
        logReadHandler.errorCount);
87         reachedSetpoint = true; //holds this information
88         clearInterval(logTimer); //stop the logging
89         log.logToFile("waiting for grains", 3, "  

          finishHeating"); //log to file at least once
90         logTimer = setInterval(function() { //logs that
          system is waiting for user input
91           log.logToFile("waiting for grains", 3); //log to
            file
92         }, 5000);
93       }
94       //while the temperature is above the setpoint, do
95       //nothing - just wait for the user
96     });
97   });
98   //this if must be outside the updateHeatingConditions()  

   call, otherwise recursive behavior mess it up
99   if(global.environmentVariables.readyForNextStep){ //wait
     for the temperature reach setpoint and user add grains
100   global.environmentVariables.readyForNextStep = false; //  

     prepare the flag for the next step request
101   clearInterval(logTimer); //stop the "waiting for grains"  

     logging
102   clearInterval(readTmpTimer); //stop the temperature
     adjusting loop
103   rampControl(recipeContents);
104 }

```

```

105     }, 1000);
106     //just do the next things after the user added the grains
107 }
108
109 function rampControl(recipeContents){
110     var lockFile = "./datalog/lockfile"; //file that tells if
111     // recipe is running
112     var instantPath = "./datalog/instant.csv";
113     var instantBK = "./datalog/instant_bk.csv";
114     var currentSetpointPointer = 1;
115     var finishedRamp = false;
116     var mashSetpoint = recipeContents[8]; //get the first
117     // setpoint
118     var spargeSetpoint = recipeContents[5];
119     var spargeSet = false; // only set this if the sparge
120     // temperature is set
121     var stepTime = 60*1000*recipeContents[9]; //get the first
122     // step rest time in ms
123     global.environmentVariables.tmpMTsetp = mashSetpoint; //
124     // stores the setpoints
125     global.environmentVariables.tmpBKsetp = spargeSetpoint;
126     var logReadHandler = {
127         lastValidTemperature: 0, //the last valid temperature
128         // reading (defaults to zero, not the better solution)
129         lastReadingsTimestamp: [0, 0, 0, 0, 0], //last 5
130         // timestamps, to check if readings are going ok
131         errorCount: 0 //counts the file reading errors
132     };
133     var logReadHandlerBK = {
134         lastValidTemperature: 0, //the last valid temperature
135         // reading (defaults to zero, not the better solution)
136         lastReadingsTimestamp: [0, 0, 0, 0, 0], //last 5
137         // timestamps, to check if readings are going ok
138         errorCount: 0 //counts the file reading errors
139     };
140     if(spargeSetpoint){ //if there is a sparge temperature
141         spargeSet = true; //then assumes there will be sparging
142         // water to heat
143     }
144     fs.writeFile(lockFile, 0, function(err){ //release system to
145         // start new production (for testing purpose only)
146         if(err) return;
147     });
148     debug("Ramp control started!");
149     log.logToFile("mash ramp in progress", 4, "sRamp0"); //log
150     // immediately and hold notable timestamp
151     var logTimer = setInterval(function(){//logs that the mash
152         // ramp is in progress
153         log.logToFile("mash ramp[" + currentSetpointPointer + "]"
154             // in progress", 4); //log to file
155     }, 5000);
156     var readTmpTimer = setInterval(function (){//read
157         // temperature every second
158         debug("Executing temperature control loop[2]");
159         if(global.environmentVariables.readyForNextStep &&
160             currentSetpointPointer < 8){ //if there are still
161             mashing steps

```

```

145     debug("setpoint pointer: " + currentSetpointPointer); //  

146     cannot enter here when currentSetpointPointer == 1  

147     global.environmentVariables.readyForNextStep = false; //  

148     wait for the step to finish before checking next  

149     step  

150     if(recipeContents[8 + 2*currentSetpointPointer]) { //and  

151         the value is not empty  

152         mashSetpoint = recipeContents[8 + 2*  

153             currentSetpointPointer]; //get the new setpoint  

154         stepTime = 60*1000*recipeContents[9 + 2*  

155             currentSetpointPointer]; //get the next step rest  

156             time in ms  

157         global.environmentVariables.tmpMTsetp = mashSetpoint;  

158         //stores the setpoints  

159         debug("Starting next ramp");  

160         clearInterval(logTimer); //stop logging step to log  

161         ramp  

162         log.logToFile("mash ramp in progress", 4, "sRamp" +  

163             currentSetpointPointer); //log immediately and hold  

164             notable timestamp  

165         logTimer = setInterval(function() { //logs that the  

166             mash ramp is in progress  

167             log.logToFile("mash ramp[" + currentSetpointPointer  

168                 + "] in progress", 4); //log to file  

169             }, 5000);  

170     }  

171     else{  

172         //gets here for all the lasting setpoints ( //  

173         currentSetpointPointer <= 7)  

174         global.environmentVariables.readyForNextStep = true;  

175         //ready for the sparging/boil  

176         debug("End of the mash ramps/steps process " +  

177             currentSetpointPointer);  

178     }  

179     currentSetpointPointer++; //increment to point to the  

180     next setpoint  

181 }  

182 else if(global.environmentVariables.readyForNextStep) { //  

183     start the next step, which is sparging or boil  

184     global.environmentVariables.readyForNextStep = false; //  

185     prepare for the next step  

186     gpioCfg.changeStatusIO("mash_heat", "false"); //just to  

187     make sure, turn off heating elements  

188     gpioCfg.changeStatusIO("boil_heat", "false"); //just to  

189     make sure, turn off heating elements  

190     gpioCfg.changeStatusIO("mash_pump", "false"); //also  

191     shutdown the recirculation pump  

192     clearInterval(readTmpTimer); //stop everything, for now  

193     at least  

194     clearInterval(logTimer);  

195     if(spargeSet){ //if there will be sparging, then start  

196         it  

197         spargingControl(recipeContents); //start the sparging  

198             process  

199     }  

200     else{ //goes straight to the boil, without sparging  

201         debug("No sparging: starting the boil process.");  

202         theBoil(recipeContents);  

203     }

```

```

178     }
179 }
180 else { //temperature control code, no need to execute when
181     it is getting the next setpoint
182     log.getTemperatureReading(instantPath, logReadHandler,
183         function(err, temperature) { //read current
184             temperature
185             if(err) {
186                 debug("something went wrong");
187                 return;
188             }
189             updateHeatingConditions("MT", mashSetpoint,
190                 temperature, function() { //control temperature for
191                     current setpoint
192                     if (!finishedRamp) { //if the ramp just reached its
193                         setpoint
194                         //log and save the notable timestamp fRamp
195                         finishedRamp = true;
196                         clearInterval(logTimer); //stop logging ramp to
197                         log step
198                         log.logToFile("mash step[" +
199                             currentSetpointPointer + "] in progress", 5, "
200                             fRamp" + currentSetpointPointer);
201                         logTimer = setInterval(function() { //logs that the
202                             mash ramp is in progress
203                             log.logToFile("mash step[" +
204                                 currentSetpointPointer + "] in progress", 5)
205                                 ; //log to file
206                             }, 5000);
207                         debug('Ramp finished, keeping temperature for the
208                             duration of the step.');
209                         debug("Current temperature = " +
210                             temperature + "\n"
211                             + "Reading errors = " +
212                             logReadHandler.errorCount);
213                     }
214                     if((global.environmentVariables.timestamps.curr -
215                         global.environmentVariables.timestamps["fRamp" +
216                             currentSetpointPointer]) > stepTime) {
217                         //if the step rest is finished
218                         finishedRamp = false;
219                         global.environmentVariables.readyForNextStep =
220                             true; //if setpoint is reached
221                         debug('Step finished, starting next ramp or going
222                             to sparging.');
223                         debug("Current temperature = " +
224                             temperature + "\n"
225                             + "Reading errors = " +
226                             logReadHandler.errorCount);
227                     }
228                     else { //if it is in the middle of a step, heat the
229                         sparging water and tell the user the step time
230                         left
231                         global.environmentVariables.timeLeft = //in
232                             minutes (not round) = total time - elapsed
233                             time
234                         (stepTime - (global.environmentVariables.
235                             timestamps.curr - global.environmentVariables.

```

```

        timestamps["fRamp" + currentSetpointPointer])) /60000;
210    if(spargeSet){//only heat sparging water if there
      is sparging water to heat
211    log.getTemperatureReading(instantBK,
      logReadHandlerBK, function(err, temperature)
      {//read current temperature
212    if(err){
      debug("something went wrong");
      return;
213    }
      updateHeatingConditions("BK", spargeSetpoint,
      temperature, function(){//heat the
      sparging water
214    });
      });
215    });
216  });
217  });
218  });
219  });
220  });
221  });
222  });
223  });
224  },1000);
225}
226
227function spargingControl(recipeContents){
228  //function heavily based in timing. The time it takes for
  the water to flow is pretty predictable.
229  //The time for the mash to drain depends on the recipe,
  hence the mash tun overflow watching device.
230  //Because the pump flow is higher then the drain flow, the
  pump must be controlled to adjust.
231  //The pump must start some time after the drain, so the
  grain bed is almost uncovered by the mash.
232  //Also, in case of overflow, there must be a predefined
  time or time function to wait until the recirculation
  restarts.
233  var logMsg = { notableTimestamp:"startDrain",//if there is
    the need to log a notable timestamp, put its name here
234    message:"sparging in progress",//log message must
    be put here
235    code:6//code identifier
236  };
237  var mashTunOverflow = false;
238  var finishedFirstDrain = false;//should modify to use
    setTimeout instead of setInterval
239  var finishedSpargeTime = false;//should modify to use
    setTimeout instead of setInterval
240  var justAnotherFlag = false;//terrible thing to name a flag
    like this! Hue Hue BR trolling myself from the future
241  //the time setpoints are based on the volume of water used
    in mashing and sparging
242  var drainTimeSetp = 500*recipeContents[3];//time in ms that
    should wait before showering with sparge water
243  var spargeTimeSetp = 3000*recipeContents[4];//time in ms
    that should wait before finishing the process
244  var overflowTimeSetp = 30000;//time to wait before pumping
    the sparge water again

```

```

245  var realSpargeTime = 0; //approximately the time that the
246    recirculation pump is working in ms
247
248  debug("Starting the sparging process");
249  log.logToFile(logMsg.message, logMsg.code, logMsg.
250    notableTimestamp); //log the startDrain notable timestamp
251  var logTimer = setInterval(function() { //log periodically
252    if(logMsg.notableTimestamp){ //if a notable timestamp is
253      set
254      log.logToFile(logMsg.message, logMsg.code, logMsg.
255        notableTimestamp); //log with notable timestamp
256      logMsg.notableTimestamp = ""; //clear the notable
257      timestamp so it doesn't overwrite
258    }
259  } ,5000); //every half second? really? it can be done every 5
260    s without major concerns!
261
262  var overflowTimer = setInterval(function() { //watch
263    periodically for MT overflow
264    if(!mashTunOverflow && global.environmentVariables.
265      ioStatus["led"].state == b.HIGH){ //if MT is about to
266      overflow
267      debug("Overflow detected, pumping on hold");
268      mashTunOverflow = true; //set overflow flag
269      gpioCfg.changeStatusIO("boil_pump", "false"); //stop
270      pumping
271      logMsg.message = "mash tun overflow"; logMsg.code = 7;
272      //log to file that the mash tun is almost
273      overflowing
274      setTimeout(function() { //wait some time before turning
275        the recirculation pump on again
276        mashTunOverflow = false;
277        //gpioCfg.changeStatusIO("boil_pump", "true"); //this
278        is risky since the pump could be already off
279        before overflow
280        debug("Fake overflow OFF for testing purposes");
281        gpioCfg.changeStatusIO("led", "false"); //fake
282        overflow for testing purposes
283        }, overflowTimeSetup);
284      }
285    } ,500);
286
287  gpioCfg.changeStatusIO("mash_valve", "true"); //open the
288    drain valve
289  var spargeTimer = setTimeout(function() { //wait until the
290    first mash drain is finished before spraying sparge
291    water
292    debug("Finished the first drain");
293    logMsg.message = "sparging in progress"; logMsg.code = 6;
294    //log the notable startSparge timestamp
295    logMsg.notableTimestamp = "startSparge";
296    gpioCfg.changeStatusIO("boil_valve", "true"); //open the
297    recirculation valve

```

```

280     gpioCfg.changeStatusIO("boil_pump", "true");//get the
          pump to run, consider some method to control the flow
          rate
281     spargeTimer = setInterval(function(){//must wait for the
          sparge time in a loop because of the overflow check
282     if(!mashTunOverflow){//if there is no overflow
          logMsg.message = "sparging in progress"; logMsg.code
          = 6;//this is just needed after an overflow
284     gpioCfg.changeStatusIO("boil_pump", "true");//this
          is just needed after an overflow
285     realSpargeTime += 500;//add approximately the loop
          time
286     if(realSpargeTime == 500){//fake an overflow to check
          code
287     debug("Fake overflow ON for testing purposes : " +
          realSpargeTime);
288     gpioCfg.changeStatusIO("led", "true");//fake
          overflow for testing purposes
289   }
290   if(realSpargeTime > spargeTimeSetp){//if the running
          time is enough
291     debug("Stopping the sparging process. Waiting for
          the mash to completely drain");
292     gpioCfg.changeStatusIO("boil_valve", "false");//
          close the recirculation valve
293     gpioCfg.changeStatusIO("boil_pump", "false");//stop
          pumping
294     clearInterval(overflowTimer);//from now on there is
          no more overflow risk
295     clearInterval(spargeTimer);//also stop the time
          increment loop
296     setTimeout(function(){//wait for all the mash to
          drain to the BK
297       debug("mash drained, sparging process finished");
298       gpioCfg.changeStatusIO("mash_valve", "false");//
          close the drain valve
299       gpioCfg.changeStatusIO("boil_valve", "false");//
          for security check
300       gpioCfg.changeStatusIO("boil_pump", "false");//
          for security check
301       clearInterval(logTimer);//stop periodic logging
302       //for now, I think there is no problem to call
          the boil here cause the only nested
          //functions are setInterval and setTimeout
303       theBoil(recipeContents);//maybe it is better to
          call this outside the loops?
304     }, 2.5*spargeTimeSetp);
305   }
306 }
307 }
308 }, 500);
309 },drainTimeSetp);
310 }
311
312 function theBoil(recipeContents){
313   //Straightforward part of the process, no need to do a lot
     of checks, etc
314   //something important is remember the user to remove the
     grains from the MT

```

```

315 //because the chill water will be stored there
316 var boilFlag = false; //set when the boil starts
317 var endBoilTime; //hold the timestamp of when the boil must
318   end
319 var instantBK = "./datalog/instant_bk.csv";
320 var logReadHandlerBK = {
321   lastValidTemperature: 0, //the last valid temperature
322     reading (defaults to zero, not the better solution)
323   lastReadingsTimestamp: [0, 0, 0, 0, 0], //last 5
324     timestamps, to check if readings are going ok
325   errorCount: 0 //counts the file reading errors
326 };
327 debug("Heating wort until it boils");
328 log.logToFile("heating wort to boil", 8, "heatingBoil"); //
329   log the process start
330 var logTimer = setInterval(function() { //log while waiting
331   for the boil
332     log.logToFile("heating wort to boil", 8); //log
333       periodically
334     }, 5000);
335 gpioCfg.changeStatusIO("boil_heat", "true"); //full power
336 var readTmpTimer = setInterval(function() { //read
337   temperature every second
338   var hopAddSetp; //timestamp of when to add the next hop
339   log.getTemperatureReading(instantBK, logReadHandlerBK,
340     function(err, temperature){
341       if(err){
342         debug("something went wrong");
343         return;
344       }
345       if(temperature > 96 && !boilFlag){ //if it is near
346         enough boiling
347         debug("boiling started! Burn, baby, burn!");
348         clearInterval(logTimer); //stop logging code 7
349         log.logToFile("boiling wort", 9, "boilStart"); //log
350           the process start
351         logTimer = setInterval(function() { //log the boil
352           log.logToFile("boiling wort", 9); //log periodically
353         }, 5000);
354         endBoilTime = 60*1000*recipeContents[6] + global.
355           environmentVariables.timestamps.curr; //end of
356             boiling
357         global.environmentVariables.timestamps.
358           boilFinishScheduled = endBoilTime;
359         for(var hopPointer = 0; hopPointer < 8; hopPointer++){
360           //check all possible hop addition times
361           hopAddSetp = 60*1000*recipeContents[42 + 3*
362             hopPointer]; //time from the beginning in ms
363           if(hopAddSetp){ //if there is a next hop to add
364             //this way, if the hops are out of order of
365               addition, it doesn't matter
366             debug("Hop " + recipeContents[40 + 3*hopPointer]
367               + "scheduled to be added in " + hopAddSetp +
368                 " ms");
369             setTimeout(function() { //then add it when it is
370               time
371               debug("Adding hop " + recipeContents[40 + 3*
372                 hopPointer]);

```

```

353         controlHopCompartment(hopPointer + 1);
354         log.logToFile("added hop [" + hopPointer + "]",
355                         10); //log periodically
356     }, hopAddSetp);
357 }
358 }
359 });
360 if(global.environmentVariables.timestamps.curr >
361     endBoilTime){//end of the boil
362     debug("end of the boil");
363     gpioCfg.changeStatusIO("boil_heat", "false"); //shutdown
364     the BK heater
365     controlHopCompartment(0); //close the hops compartment
366     log.stopTemperatureLogging(); //for testing purposes
367 }, 1000);
368 }
369 function controlHopCompartment(pos){ //control the servo to
370     open the hops compartment until requested position
371     //maybe it's better to close everytime a hop is added,
372     think about it!
373     var angle = pos*11.25; //11.25 degree per compartment, if
374     pos=0, close everything
375     gpioCfg.changeStatusIO("servo_pwm", angle); //change the
376     compartment opening
377 }
378
379 function updateHeatingConditions(vessel, setp, temperature,
380     callback){
381     //This function must be called in a loop, every loop
382     interation
383     if(vessel == "mash_heat" || vessel == "MT" || vessel == "mash tun"){
384         vessel = "mash_heat"; //if wants to heat the mash tun
385         global.environmentVariables.tmpMT = temperature; //update
386         the mash tun temperature
387         if(global.environmentVariables.ioStatus["boil_heat"].state == b.HIGH){ //security check
388             gpioCfg.changeStatusIO("boil_heat", "false"); //turn off
389             the other heating element
390             debug("Security shutdown of the BK heater");
391         }
392     }
393     else if(vessel == "boil_heat" || vessel == "BK" || vessel
394         == "brewing kettle" || vessel == "copper"){
395         vessel = "boil_heat"; //if wants to heat the brewing
396         kettle
397         global.environmentVariables.tmpBK = temperature; //update
398         the brewing kettle temperature
399         if(global.environmentVariables.ioStatus["mash_heat"].state == b.HIGH){ //security check
400             gpioCfg.changeStatusIO("mash_heat", "false"); //turn off
401             the other heating element
402             debug("Security shutdown of the MT heater");
403         }
404     }

```

```

393     else//if vessel is unknown
394         callback("unknown vessel");
395         return;//don't even try to heat
396     }
397     debug("Heating power variable: " + global.heatingPower);
398     if(temperature < 0.7*setp){//if temperature is less than
399         0.7 of the setpoint
400         debug(vessel + ": 100% heating power");
401         gpioCfg.changeStatusIO(vessel, "true");//give 100% power
402             to the heating resistor
403     }
404     else if(temperature < 0.9*setp){//if temperature between
405         0.7 and 0.9 of the setpoint
406         debug(vessel + ": 66% heating power");
407         if(global.heatingPower == 0){//give 66% power to the
408             heating resistor
409             gpioCfg.changeStatusIO(vessel, "false");// 1/3 of the
410                 time off
411             global.heatingPower = 2;// heatingPower goes from 0 to
412                 2
413     }
414     else{
415         gpioCfg.changeStatusIO(vessel, "true");// 2/3 of the
416             time on
417         global.heatingPower--;
418     }
419     else if(temperature < setp){//if temperature between 0.9
420         and 1.0 of the setpoint
421         debug(vessel + ": 33% heating power");
422         if(global.heatingPower == 0){//give 33% power to the
423             heating resistor
424             gpioCfg.changeStatusIO(vessel, "true");// 1/3 of the
425                 time on
426             global.heatingPower = 2;// heatingPower goes from 0 to
427                 2
428     }
429     else{
430         gpioCfg.changeStatusIO(vessel, "false");// 2/3 of the
431             time off
432         global.heatingPower--;
433     }
434     else//in this case, the temperature got to the setpoint
435     debug(vessel + ": 0% heating power");
436     gpioCfg.changeStatusIO(vessel, "false");// turn the
437         heating element off
438     callback(null);//choose what to do
439 }

```

Código-fonte C.5: Código-fonte raiz da aplicação em Node.js

Apêndice D

Arquivos de configuração do sistema

D.1 Arquivo de configuração de rede

```

1 # This file describes the network interfaces available
2 # on your system and how to activate them.
3 # For more information, see interfaces(5).
4
5 # The loopback network interface
6 auto lo
7 iface lo inet loopback
8
9 # The primary network interface
10 #auto eth0
11 #iface eth0 inet dhcp
12 # Example to keep MAC address between reboots
13 #hwaddress ether DE:AD:BE:EF:CA:FE
14
15 # The secondary network interface
16 #auto eth1
17 #iface eth1 inet dhcp
18
19 # WiFi Example
20 auto wlan0
21 allow-hotplug wlan0
22 iface wlan0 inet static
23 #allow-hotplug wlan0
24 #auto lo
25 #iface lo inet loopback
26 #auto eth0
27 #iface eth0 inet static
28 address 192.168.1.155
29 netmask 255.255.252.0
30 network 192.168.1.0
31 gateway 192.168.1.1
32 #pre-up ifconfig eth0 hw ether 00:01:02:03:05:14
33 dns-nameservers 143.107.225.6 143.107.182.2 8.8.8.8
34 wpa-ssid      "nome_da_rede"
35 wpa-psk       "senha1"
36 #iface wlan0 inet dhcp
37 #      wpa-ssid "outra_rede"
```

```

38 #      wpa-psk    "senha2"
39
40 # Ethernet/RNDIS gadget (g_ether)
41 # ... or on host side, usbnet and random hwaddr
42 # Note on some boards, usb0 is automatically
43 # setup with an init script
44 iface usb0 inet static
45 address 192.168.7.2
46 netmask 255.255.255.0
47 network 192.168.7.0
48 gateway 192.168.7.1

```

Código-fonte D.1: /etc/network/interfaces

D.2 Device Tree para o DS18B20

```

1 /*
2 * Copyright (C) 2012 Texas Instruments Incorporated - http://
3 * www.ti.com/
4 *
5 * This program is free software; you can redistribute it and/
6 * or modify
7 * it under the terms of the GNU General Public License
8 * version 2 as
9 * published by the Free Software Foundation.
10 *
11 * Modified by Russell Senior from the weather cape's DTS
12 * file.
13 * Minor formatting by C W Rose.
14 * Modified by Leonardo Graboski Veiga to change the DS18B20
15 * pin and pin configuration
16 */
17 /dts-v1/;
18 /plugin/;

19 /
20 {
21     compatible = "ti,beaglebone", "ti,beaglebone-black";
22     part-number = "BB-W1";
23     version = "00A0";
24
25     exclusive-use = "P9.11";
26
27     fragment@0 {
28         target = <&am33xx_pinmux>;
29         __overlay__ {
30             bb_w1_pins: pinmux_bb_w1_pins {
31                 pinctrl-single,pins = <0x70 0x37>;
32             };
33         };
34     };
35
36     fragment@1 {
37         target = <&ocp>;
38         __overlay__ {
39             onewire@0 {

```

```
35         status          = "okay";
36         compatible     = "w1-gpio";
37         pinctrl-names  = "default";
38         pinctrl-0       = <&bb_w1_pins>;
39
40         gpios          = <&gpiol 30 0>;
41     };
42 };
43 };
44 }
```

Código-fonte D.2: w1.dts

Apêndice E

Cálculo de OG e IBU de uma receita de cerveja

Neste apêndice é apresentado o cálculo da gravidade original (OG) e do amargor (IBU) de uma receita de cerveja, baseado em [1]. Em primeiro lugar, é importante salientar que o cálculo da OG e do IBU dependem fortemente de dados do equipamento de produção de cerveja e/ou de constantes levantadas empiricamente, o que significa que este roteiro deve ser visto como um guia e, portanto, na necessidade de obter dados exatos, medições feitas por laboratórios especializados são necessárias.

Primeiramente é preciso calcular as unidades de ácidos alfa, que é um meio para quantificar a contribuição de amargor independentemente do tipo de lúpulo. Para isto, o peso em onças é multiplicado pela porcentagem de ácidos alfa, medida em laboratório e fornecida pelo fabricante. A fórmula para cálculo de AAU é apresentada na equação E.1. Outra aplicação interessante é quando uma variação específica de lúpulo tem sua porcentagem de ácidos alfa diferente em anos diferentes: neste caso, sabendo a AAU da receita original, é fácil recalcular a massa necessária para adequar a receita às novas especificações.

$$AAU = w \cdot aa \quad (\text{E.1})$$

Outro parâmetro de padronização de uma cerveja é a gravidade original, que nada mais é do que a densidade do mosto após o cozimento dos grãos. Ela é baseada nos pontos de contribuição (*pts*) de cada malte, ou seja, uma transformação numérica da gravidade original do malte, que quantifica a contribuição de açúcares do malte em questão para o mosto e indicada em g/cm^3 . O valor da gravidade original do malte é fornecido pelo fabricante geralmente

em pontos, mas a equação E.2 expressa o cálculo a partir do valor da densidade. Com isso é possível calcular a gravidade original do mosto, conforme descrito na equação E.3, na qual n é o número de maltes empregados na receita, η é a eficiência do equipamento, ou capacidade de extração de açúcares fermentáveis dos grãos para o mosto e V é o volume final de cerveja. Observe-se que o volume é expresso em galões e a massa em onças.

$$pts_{malte} = (1 - OG_{malte}) \cdot 1000 \quad (\text{E.2})$$

$$OG = \left\{ \left[\sum_n (pts_n \cdot w_n) \right] \cdot \frac{\eta}{1000 \cdot V} \right\} + 1 \quad (\text{E.3})$$

A utilização é um dos fatores mais críticos para o cálculo adequado da quantidade de IBU da cerveja. Ela é uma função da OG e do tempo de fervura. Uma vez que seu valor é fortemente dependente do vigor da fervura, da química do mosto e de outros parâmetros difíceis de quantificar, valores empíricos de constantes são empregados como um ponto de estimativa de IBU — embora com o tempo o operador do equipamento deva fazer um ajuste fino destas constantes para aumentar a precisão do cálculo. O cálculo da utilização u é apresentado no conjunto de equações E.6.

$$u = f(OG) \cdot f(t), \quad \text{onde :} \quad (\text{E.4a})$$

$$f(OG) = 1,65 \cdot 0,000125^{(OG-1)} \quad (\text{E.4b})$$

$$f(t) = \frac{1 - e^{-0,04t}}{4,15} \quad (\text{E.4c})$$

Finalmente, é possível calcular a quantidade de amargor da cerveja usando a fórmula E.5, na qual V é o volume final da produção expresso em galões e 74,89 é o fator de conversão de onças por galão para miligramas por litro, unidade do IBU.

$$IBU = \frac{AAU \cdot u \cdot 74,89}{V} \quad (\text{E.5})$$

E.1 Estimação do IBU para uma receita de 20l

Para saber se 400g de lúpulos são suficientes para uma receita de 20 litros, foi feito um cálculo hipotético considerando situações extremas: densidade original alta ($1,120 g/cm^3$), tempo de

fervura alto (120min), lúpulo nobre (baixo teor de ácidos alfa = 5%):

$$AAU = (400 \cdot 0,035274) \cdot 5,0 = 70,55$$

$$f(OG) = 1,65 \cdot 0,000125^{(1,12-1,00)} = 0,56$$

$$f(t) = \frac{1 - e^{-0,04 \cdot 120}}{4,15} = 0,24$$

$$u = 0,56 \cdot 0,24 = 0,135$$

$$IBU = \frac{70,55 \cdot 0,135 \cdot 74,89}{20 \cdot 0,264} = 135mg/l$$

Considerando o guia do BJCP (Beer Judge Certification Program), que é adotado por profissionais ao redor de todo o mundo para participarem de competições de cervejas como juízes, os dois estilos de cerveja com maior amargor são *Imperial India Pale Ale* e *American Barley Wine*, ambos com limite superior de 120 IBU [56] e o livro *How to Brew* que descreve somente um estilo com mais de 120 IBU — justamente o *American Barley Wine* — é razoável assumir que raramente será necessário operar a estrutura de adição de lúpulos em sua capacidade máxima.

Apêndice F

Análise do tempo de leitura do sensor DS18B20 em Python

Anexo I

Diagrama de conexões elétricas da BeagleBone Black

REV	Description	DATE	BY
A4A	Initial production Release.	11/19/2012	GC
A5	On the initial production release the processors were to be found incorrect as supplied by TI. Parts while marked AM3359 were actually AM3352. This revision uses the correct parts.	1/2/2013	GC
D	<ul style="list-style-type: none"> 1. Deleted R29-R44 from the LCD lines. 2. Added 47pf capacitors C156-C173 to LCD data lines to ground. 3. Changed schematic revision to A5A. 4. Changed a few footprints after PCB update for above changes. 5. Added access point for the battery function of the TPS65217C. 6. Added Ferrite beads in series with LED power and 5V power rail of the USB host connector. Required to pass FCC/CE testing due to noise emissions on that pin. 7. Added power button to enable sleep, wakeup, power down and power up features on the system. 8. Added Modification to add 100K ohm resistor to ground to prevent crosstalk when serial cable is not plugged in. 		
A5A	<ul style="list-style-type: none"> 1. Added 100K pulldown on J1 pin 4 to prevent crosstalk when serial cable is not connected into PCB layout. 2. Changed the LED resistors to 4.7kΩ to lower the brightness. 	2/8/2013	GC
C	<ul style="list-style-type: none"> 1. Changed R46, R47, R48 to 0 ohms. 2. Changed R45 to 22 Ohms. <p>Change was made due to production failures on some boards due to differences in impedances.</p>		
A5B	<ul style="list-style-type: none"> 1. Moved the enable for the VDD_3V3B regulator to VDD_3V3A rail. Change was made to reduce the delay between the ramp up of the 3.3V rails. 2. Added a AND gate to the SYS_RESETn circuitry. There is a small change that on power up the nRESETn signal on the processor may go high, causing the SYS_RESETn signal to go HI before it should. This change reinforces the reset with the PORZn reset signal. 3. Added optional zero ohm resistor to tie GND_OSCO to system ground. 	5/2/2013	GC
A5C	<ul style="list-style-type: none"> 1. Added optional zero ohm resistor to tie GND_OSC1 to system ground. 2. Changed C106 to a 1uF capacitor. 3. Changed C24 to a 2.2uF capacitor. 4. Made R8 installed and R9 not installed. 	6/12/2013	GC
A6	<ul style="list-style-type: none"> 1. Changed the processor to the AM3358BZCZ100. 	12/13/2013	GC
B	<ul style="list-style-type: none"> 1. Increased the eMMC from 2GB to 4GB. 	1/20/2014	GC
C	<ul style="list-style-type: none"> 1. Increased the eMMC from 2GB to 4GB. 	3/21/2014	GC

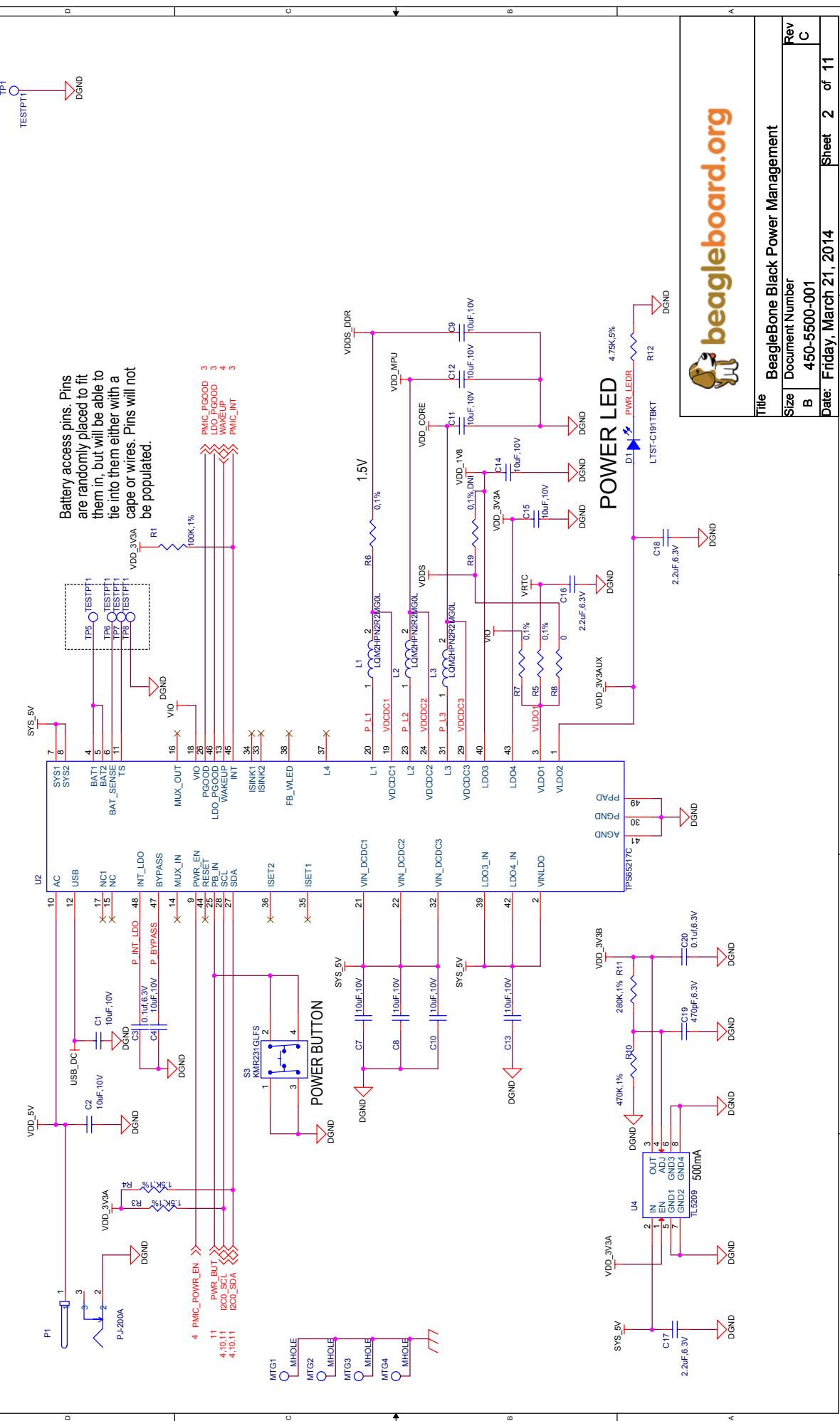
This schematic is *NOT SUPPORTED* and DOES NOT constitute a reference design. Only "community" support is allowed via resources at BeagleBoard.org/discuss.

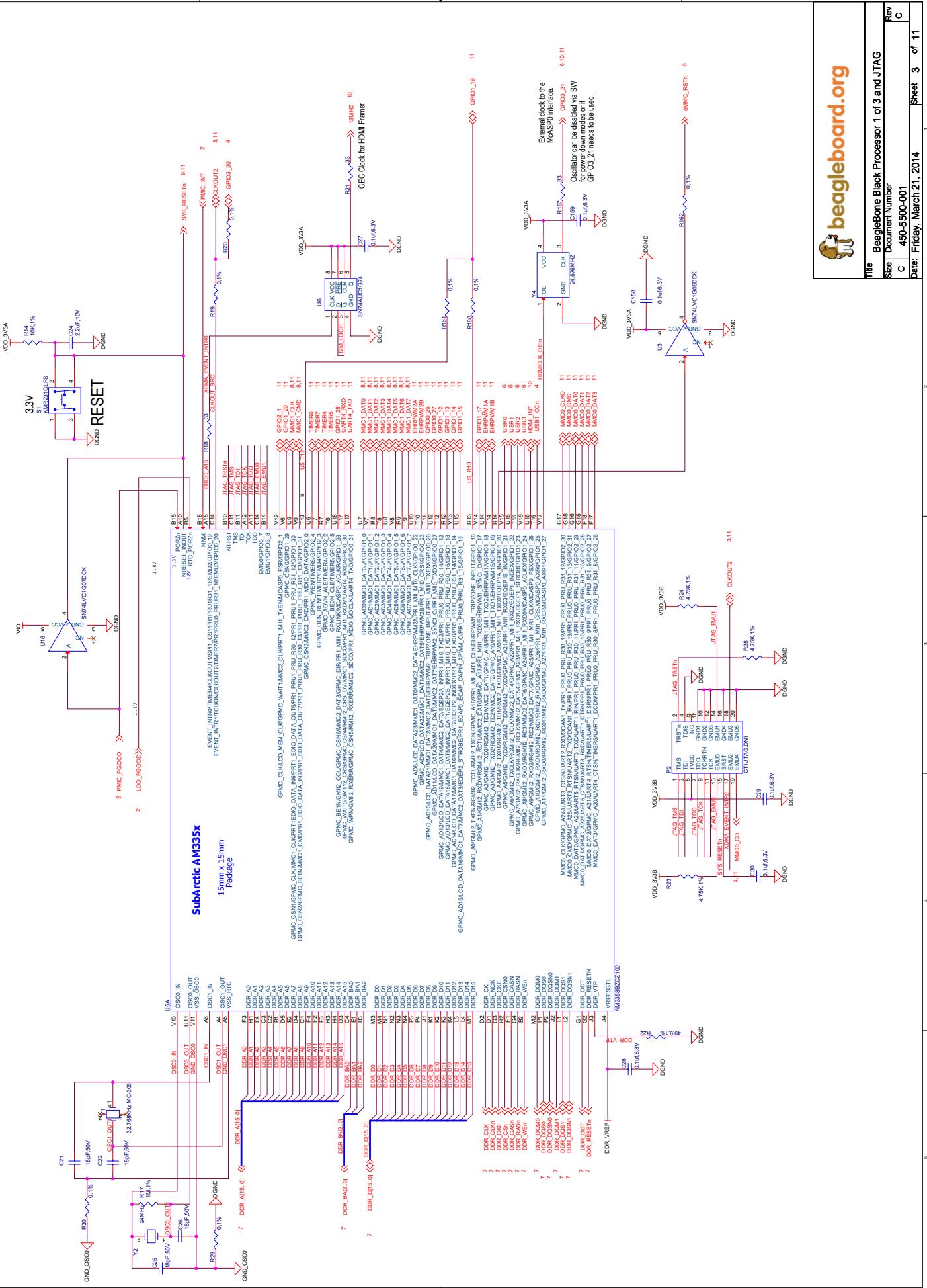
THERE IS NO WARRANTY FOR THIS DESIGN ' TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE DESIGN "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE DESIGN IS WITH YOU. SHOULD THE DESIGN PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

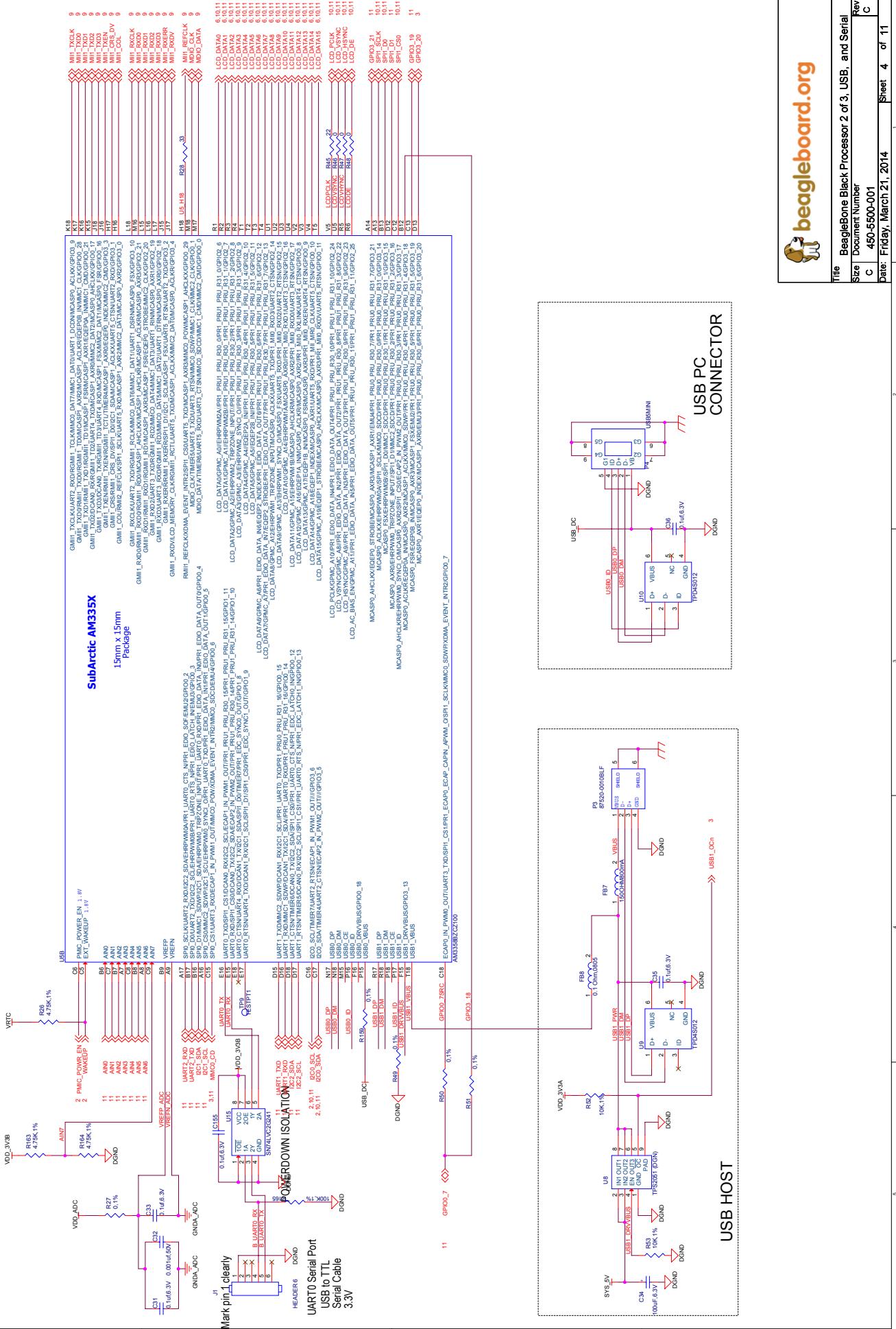
PAGE NO.	SCHEMATIC PAGE
1	COVER PAGE
2	POWER MANAGEMENT
3	PROCESSOR 1 OF 3, JTAG HEADER
4	PROCESSOR 2 OF 3, UAB PORTS
5	PROCESSOR 3 OF 3
6	LED, CONFIGURATION AND BUTTON
7	DDR3 MEMORY
8	eMMC FLASH
9	10/100 ETHERNET
10	HDMI FRAMER
11	EXP CONN, USD

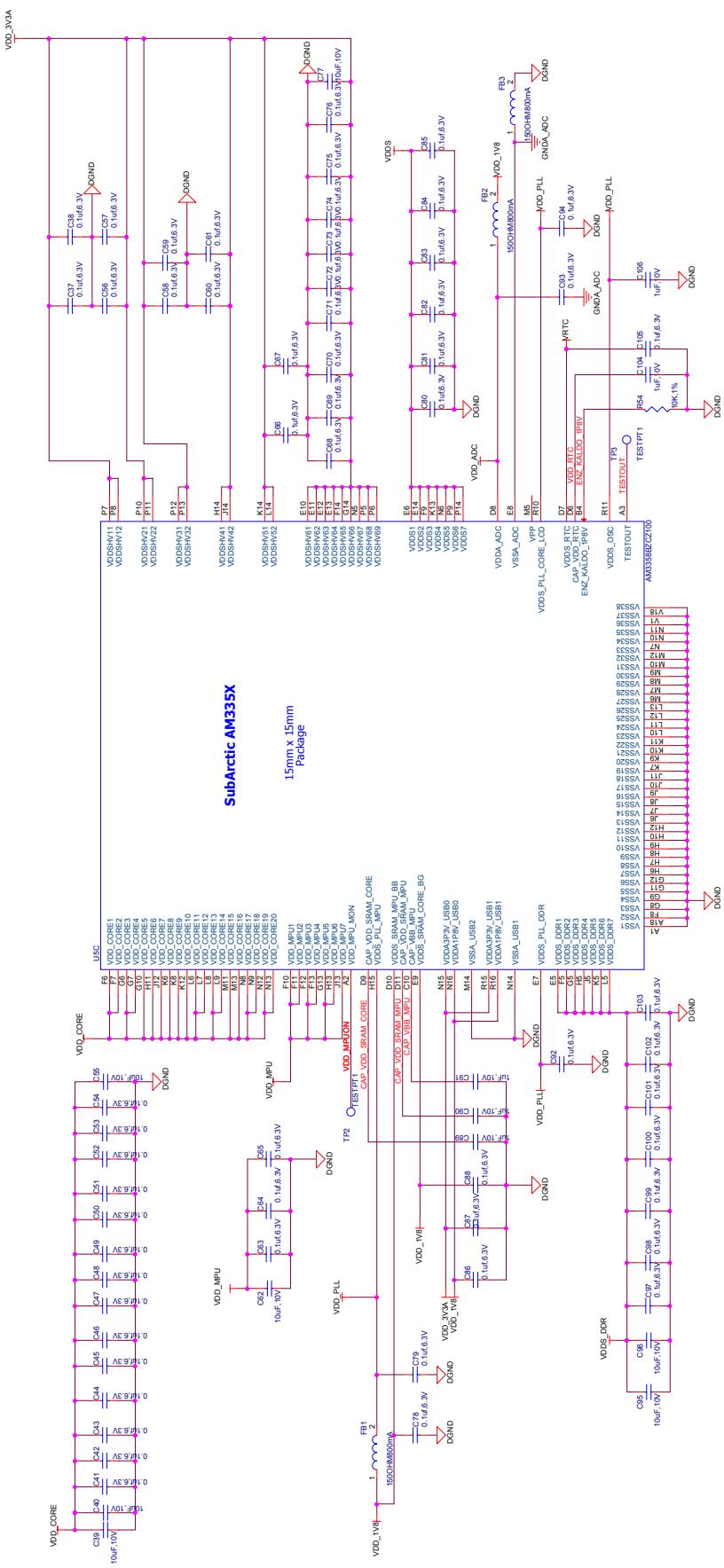
NOTE: PCB Revision for this board is Rev B6

	beagleboard.org
Title	BeagleBone Black Cover Page
Size	Document Number
B	450-5500-001
Date:	Friday, March 21, 2014
Rev	C
Sheet	1 of 11





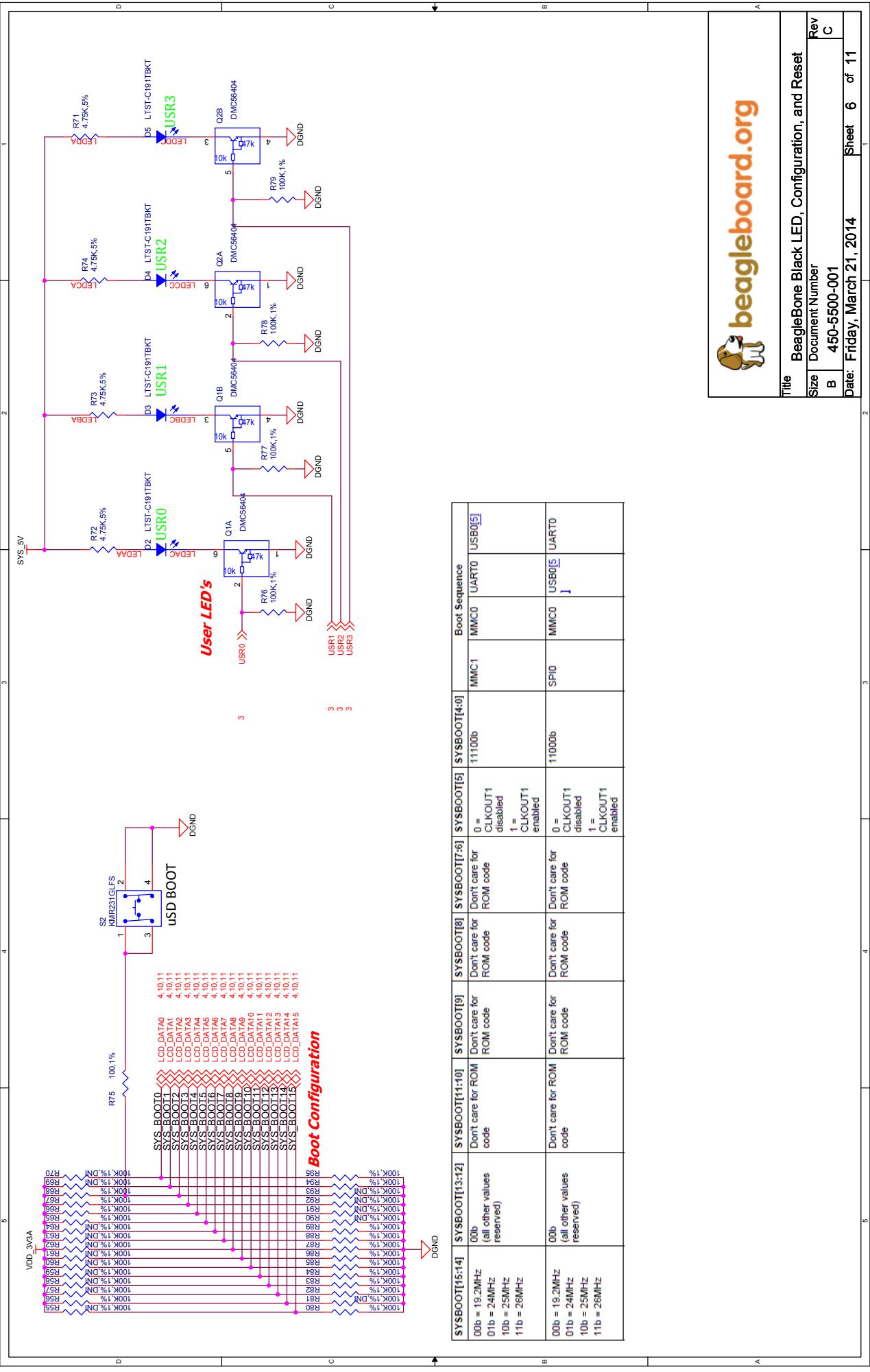




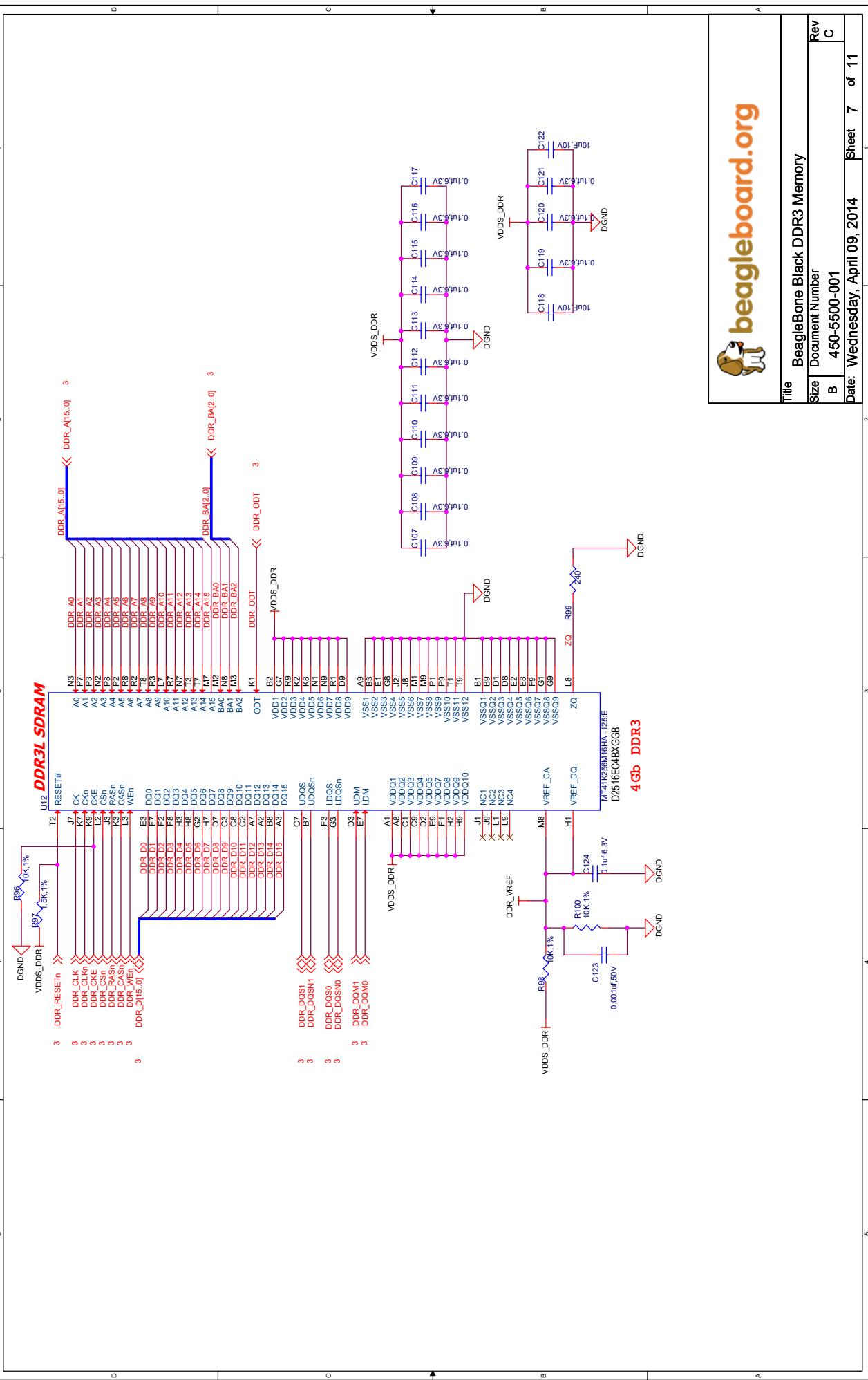
BeagleBone Black Processor 3 of 3

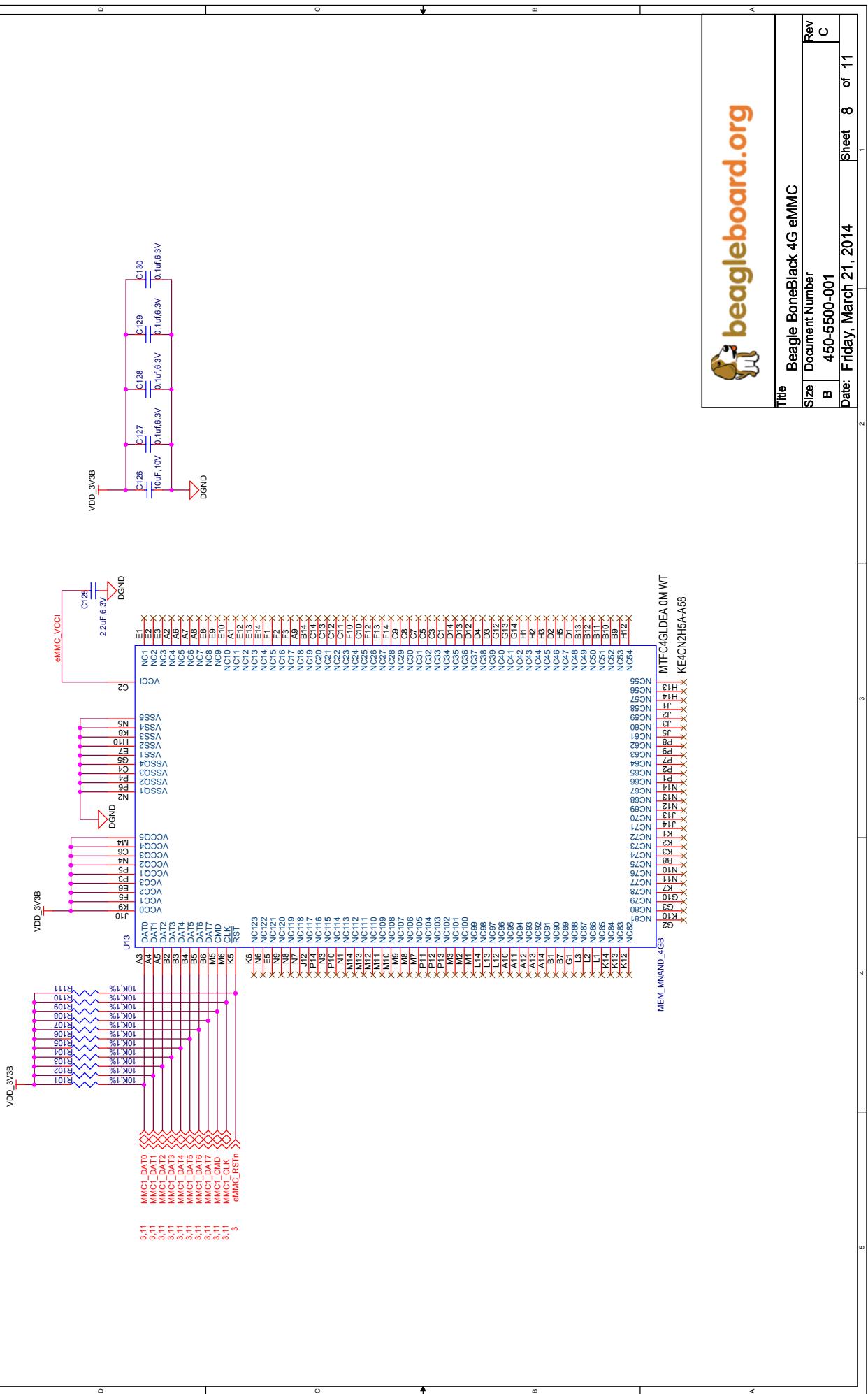
BeagleBoze Black Processor Rev

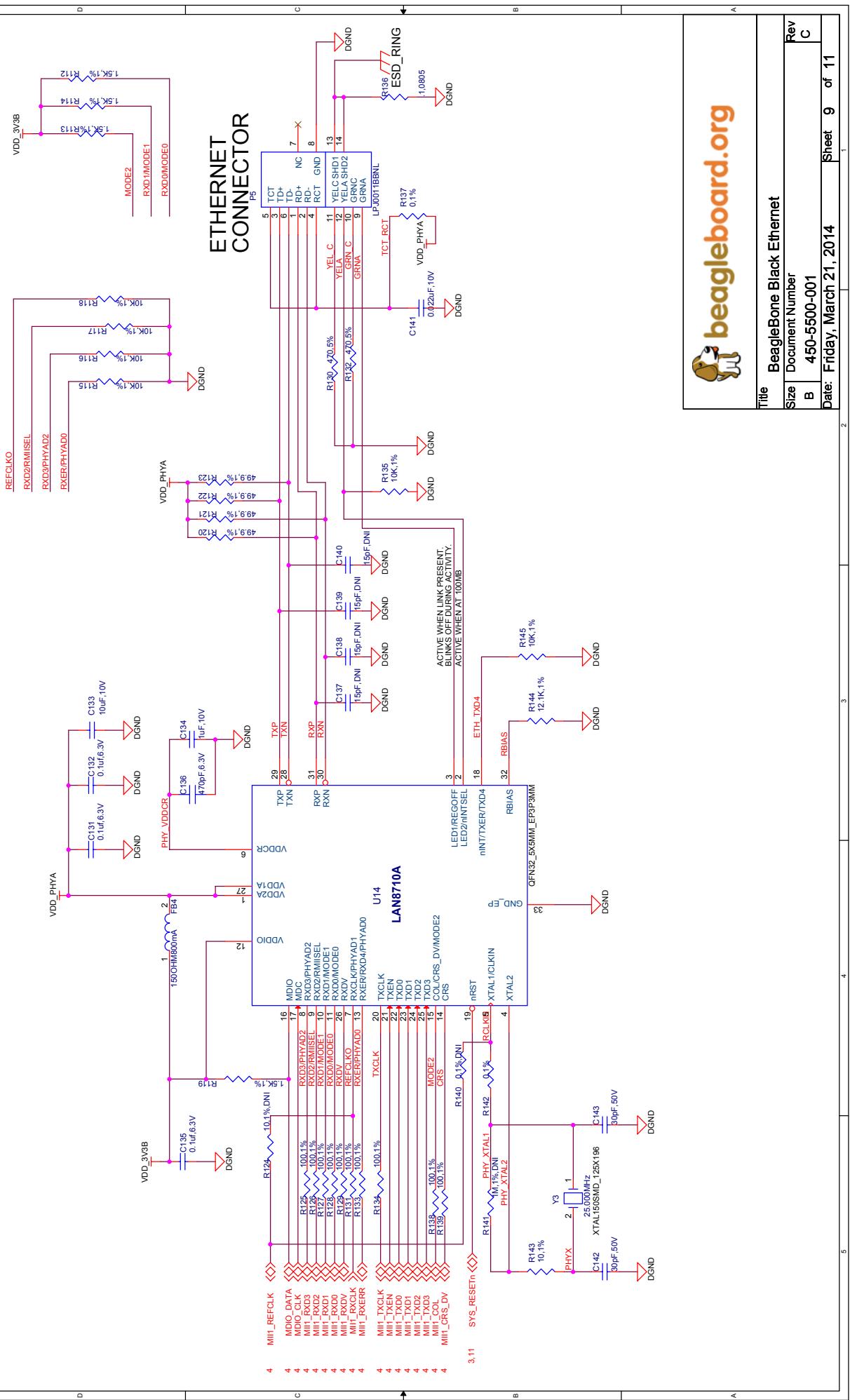
1

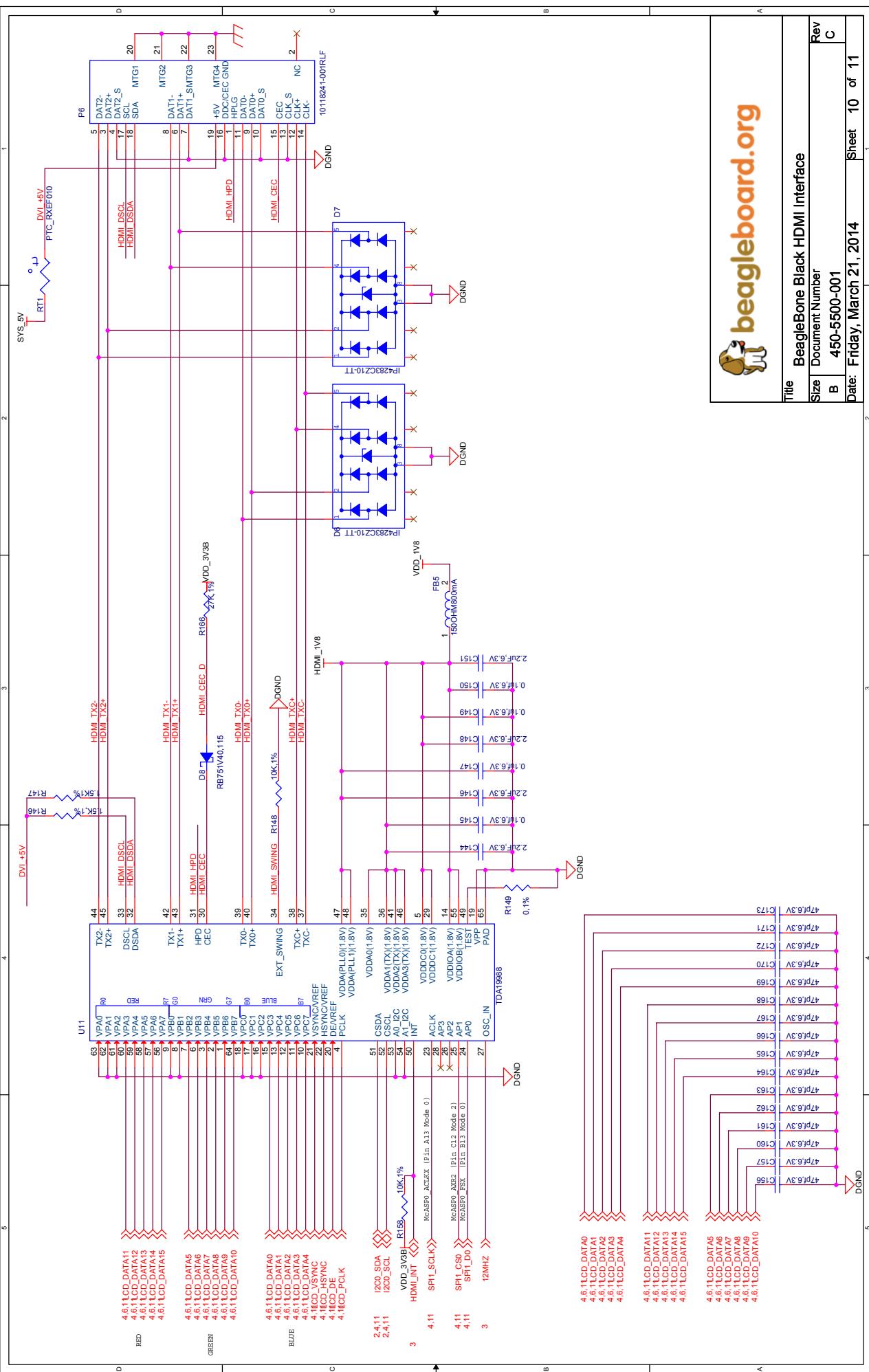


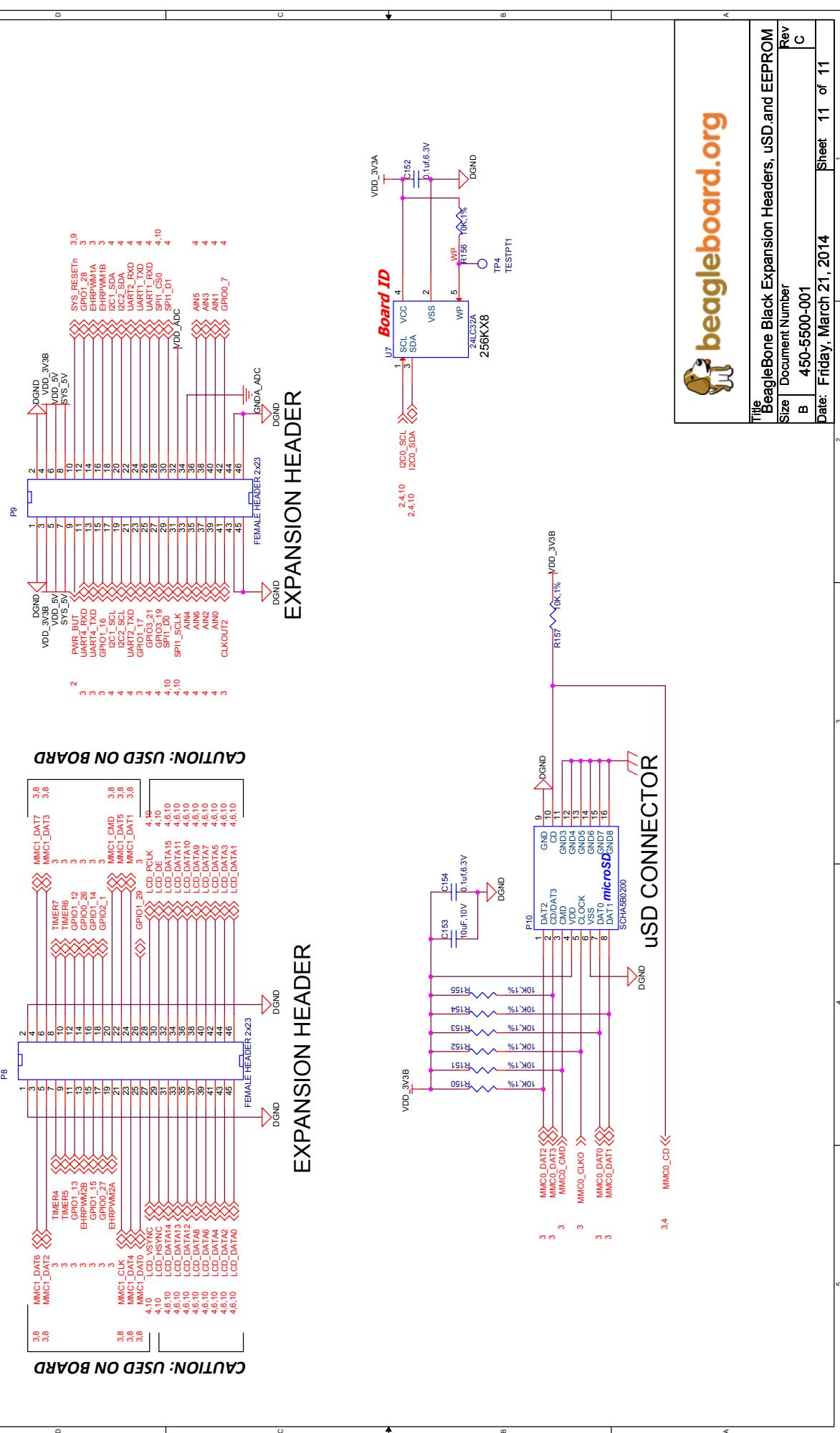
Title		BeagleBone Black LED, Configuration, and Reset	
Size	Document Number	Rev	C
B	450-5500-001		
Date:	Friday, March 21, 2014	Sheet 6 of 11	1











Anexo II

Registradores do Módulo de Controle

Tabela II.1: Registradores do módulo de controle.

Fonte: adaptado de TEXAS INSTRUMENTS(2014)

Offset	Acrônimo
800h	conf_gpmc_ad0
804h	conf_gpmc_ad1
808h	conf_gpmc_ad2
80Ch	conf_gpmc_ad3
810h	conf_gpmc_ad4
814h	conf_gpmc_ad5
818h	conf_gpmc_ad6
81Ch	conf_gpmc_ad7
820h	conf_gpmc_ad8
824h	conf_gpmc_ad9
828h	conf_gpmc_ad10
82Ch	conf_gpmc_ad11
830h	conf_gpmc_ad12
834h	conf_gpmc_ad13
838h	conf_gpmc_ad14
83Ch	conf_gpmc_ad15
840h	conf_gpmc_a0
844h	conf_gpmc_a1

Tabela II.1: Registradores do módulo de controle.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014) (continuação)

848h	conf_gpmc_a2
84Ch	conf_gpmc_a3
850h	conf_gpmc_a4
854h	conf_gpmc_a5
858h	conf_gpmc_a6
85Ch	conf_gpmc_a7
860h	conf_gpmc_a8
864h	conf_gpmc_a9
868h	conf_gpmc_a10
86Ch	conf_gpmc_a11
870h	conf_gpmc_wait0
874h	conf_gpmc_wpn
878h	conf_gpmc_ben1
87Ch	conf_gpmc_csn0
880h	conf_gpmc_csn1
884h	conf_gpmc_csn2
888h	conf_gpmc_csn3
88Ch	conf_gpmc_clk
890h	conf_gpmc_advn_ale
894h	conf_gpmc_oen_ren
898h	conf_gpmc_wen
89Ch	conf_gpmc_ben0_cle
8A0h	conf_lcd_data0
8A4h	conf_lcd_data1
8A8h	conf_lcd_data2
8ACh	conf_lcd_data3
8B0h	conf_lcd_data4
8B4h	conf_lcd_data5

Tabela II.1: Registradores do módulo de controle.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014) (continuação)

8B8h	conf_lcd_data6
8BCh	conf_lcd_data7
8C0h	conf_lcd_data8
8C4h	conf_lcd_data9
8C8h	conf_lcd_data10
8CCh	conf_lcd_data11
8D0h	conf_lcd_data12
8D4h	conf_lcd_data13
8D8h	conf_lcd_data14
8DCh	conf_lcd_data15
8E0h	conf_lcd_vsync
8E4h	conf_lcd_hsync
8E8h	conf_lcd_pclk
8ECh	conf_lcd_ac_bias_en
8F0h	conf_mmc0_dat3
8F4h	conf_mmc0_dat2
8F8h	conf_mmc0_dat1
8FCh	conf_mmc0_dat0
900h	conf_mmc0_clk
904h	conf_mmc0_cmd
908h	conf_mmi1_col
90Ch	conf_mmi1_crs
910h	conf_mmi1_rx_er
914h	conf_mmi1_tx_en
918h	conf_mmi1_rx_dv
91Ch	conf_mmi1_txd3
920h	conf_mmi1_txd2
924h	conf_mmi1_txd1

Tabela II.1: Registradores do módulo de controle.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014) (continuação)

928h	conf_mmi1_txd0
92Ch	conf_mmi1_tx_clk
930h	conf_mmi1_rx_clk
934h	conf_mmi1_rxd3
938h	conf_mmi1_rxd2
93Ch	conf_mmi1_rxd1
940h	conf_mmi1_rxd0
944h	conf_rmmi1_ref_clk
948h	conf_mdio
94Ch	conf_mdc
950h	conf_spi0_sclk
954h	conf_spi0_d0
958h	conf_spi0_d1
95Ch	conf_spi0_cs0
960h	conf_spi0_cs1
964h	conf_ecap0_in_pwm0_out
968h	conf_uart0_ctsn
96Ch	conf_uart0_rtsn
970h	conf_uart0_rdx
974h	conf_uart0_tdx
978h	conf_uart1_ctsn
97Ch	conf_uart1_rtsn
980h	conf_uart1_rdx
984h	conf_uart1_tdx
988h	conf_i2c0_sda
98Ch	conf_i2c0_scl
990h	conf_mcasp0_aclkx
994h	conf_mcasp0_fsx

Tabela II.1: Registradores do módulo de controle.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014) (continuação)

998h	conf_mcasp0_axr0
99Ch	conf_mcasp0_ahclkx
9A0h	conf_mcasp0_aclkx
9A4h	conf_mcasp0_fsr
9A8h	conf_mcasp0_axr1
9ACh	conf_mcasp0_ahclkx
9B0h	conf_xdma_event_intr0
9B4h	conf_xdma_event_intr1
9B8h	conf_warmrstn
9C0h	conf_nnnmi
9D0h	conf_tms
9D4h	conf_tdi
9D8h	conf_tdo
9DCh	conf_tck
9E0h	conf_trstn
9E4h	conf_emu0
9E8h	conf_emu1
9F8h	conf_RTC_pwrctrlstn
9FCh	conf_pmic_power_en
A00h	conf_ext_wakeup
A04h	conf_RTC_kaldo_enn
A1Ch	conf_usb0_drvvbus
A34h	conf_usb1_drvvbus

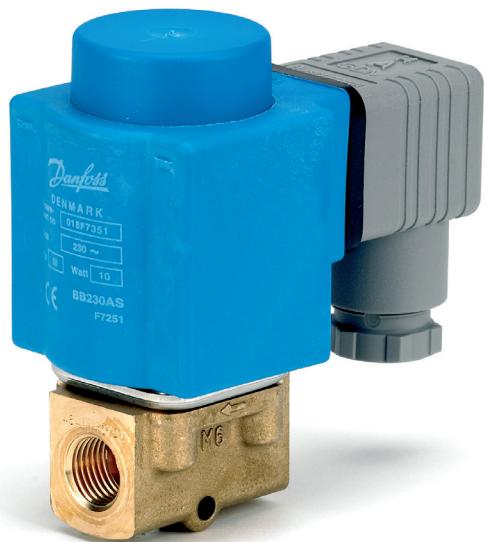
Anexo III

Válvula Danfoss EV210BD 032U3620

Ficha técnica

Válvulas solenoides de 2/2 vias de operação direta

Tipo EV210B



A EV210B cobre uma ampla linha de válvulas solenoides de 2/2 vias de operação direta para uso universal.

A EV210B é uma válvula muito robusta com alto desempenho e pode ser utilizada em todos os tipos e condições de trabalho, até mesmo nas mais adversas aplicações industriais, tais como controle e fechamento.

Características e versões:

- Para água, óleo, ar comprimido e meios neutros similares.
- Faixa de fluxo: 0 – 8 m³/h
- Pressão diferencial: 0 – 30 bar
- Temperatura do meio: -30 a 140 °C
- Temperatura ambiente: Até 80 °C
- Grau de proteção: Até IP67
- Conexões de rosca: G 1/8 – G 1
- DN 1.5 – 25
- Viscosidade: Até 50 cSt
- A válvula pode ser usada para vácuo
- Versão EV210B em latão para água, óleo, ar comprimido e meios neutros similares
- Versão EV210 em aço inoxidável para líquidos e gases neutros e agressivos.
- Também disponível com conexão NPT.

**Corpo da válvula em latão
EV210B, NF**


Conex. ISO 228/1	Material de vedado cão	Diâmetro do orifício	Valor kv [m³/h]	Pressão diferencial mín. a máx. [bar]/ tipo de bobina ²⁾							Temperatura do meio mín. a máx. [°C]	Código
				BA 9 [W c.a]	BA 15 [W c.c]	BD 15 [W c.a]	BB 10 [W c.a]	BB 18 [W c.c]	BG 12 [W c.a]	BG 20 [W c.c]		
G 1/8	EPDM ¹⁾	1.5	0.08	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	-30 - 120	032U5701
	FKM		0.08	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	-10 - 100	032U5702
	FKM	2.0	0.15	0 - 30	0 - 20	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	-10 - 100	032U5704
	EPDM ¹⁾	3.0	0.30	0 - 15	0 - 9	0 - 24	0 - 20	0 - 13	0 - 30	0 - 25	-30 - 120	032U5705
	FKM		0.30	0 - 15	0 - 9	0 - 24	0 - 20	0 - 13	0 - 30	0 - 25	-10 - 100	032U5706
G 1/4	FKM	1.5	0.08	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	-10 - 100	032U3629
	EPDM ¹⁾	2.0	0.15	0 - 30	0 - 20	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	-30 - 120	032U5707
	FKM		0.15	0 - 30	0 - 20	0 - 30	0 - 30	0 - 30	0 - 30	0 - 30	-10 - 100	032U5708
	EPDM ¹⁾	3.0	0.30	0 - 15	0 - 9	0 - 24	0 - 20	0 - 13	0 - 30	0 - 25	-30 - 120	032U5709
	FKM		0.30	0 - 15	0 - 9	0 - 24	0 - 20	0 - 13	0 - 30	0 - 25	-10 - 100	032U5710
	EPDM ¹⁾	4.5	0.55	0 - 8	0 - 3.5	0 - 12	0 - 10	0 - 4.5	0 - 13	0 - 9	-30 - 120	032U3600
	FKM		0.55	0 - 8	0 - 3.5	0 - 12	0 - 10	0 - 4.5	0 - 13	0 - 9	-10 - 100	032U3601
	EPDM ¹⁾	6.0	0.70	0 - 2.5	0 - 1.0	0 - 3.3	0 - 4.0	0 - 2.0	0 - 6	0 - 4.5	-30 - 120	032U3602
	FKM		0.70	0 - 2.5	0 - 1.0	0 - 3.3	0 - 4.0	0 - 2.0	0 - 6	0 - 4.5	-10 - 100	032U3603
G 3/8	EPDM ¹⁾	3.0	0.30	0 - 15	0 - 9	0 - 24	0 - 20	0 - 13	0 - 30	0 - 25	-30 - 120	032U3642
	FKM		0.30	0 - 15	0 - 9	0 - 24	0 - 20	0 - 13	0 - 30	0 - 25	-10 - 100	032U3643
	EPDM ¹⁾	4.5	0.55	0 - 8	0 - 3.5	0 - 12	0 - 10	0 - 4.5	0 - 13	0 - 9	-30 - 120	032U3605
	FKM		0.55	0 - 8	0 - 3.5	0 - 12	0 - 10	0 - 4.5	0 - 13	0 - 9	-10 - 100	032U3606
	EPDM ¹⁾	6.0	0.70	0 - 2.5	0 - 1.0	0 - 3.3	0 - 4.0	0 - 2.0	0 - 6	0 - 4.5	-30 - 120	032U3607
	FKM		0.70	0 - 2.5	0 - 1.0	0 - 3.3	0 - 4.0	0 - 2.0	0 - 6	0 - 4.5	-10 - 100	032U3608
	EPDM ¹⁾	8.0	1.00	0 - 1.5	0 - 0.5	0 - 2.0	0 - 2.0	0 - 1.2	0 - 3	0 - 2.5	-30 - 120	032U3609
	FKM		1.00	0 - 1.5	0 - 0.5	0 - 2.0	0 - 2.0	0 - 1.2	0 - 3	0 - 2.5	-10 - 100	032U3610
	EPDM ¹⁾	10.0	1.50	0 - 0.8	0 - 0.3	0 - 1.1	0 - 1.2	0 - 0.6	0 - 1.6	0 - 1.3	-30 - 120	032U3611
	FKM		1.50	0 - 0.8	0 - 0.3	0 - 1.1	0 - 1.2	0 - 0.6	0 - 1.6	0 - 1.3	-10 - 100	032U3612
	EPDM ¹⁾	15.0	2.50	0 - 0.25	-	0 - 0.4	0 - 0.3	0 - 0.15	0 - 0.45	0 - 0.4	-30 - 120	032U3613
	FKM		2.50	0 - 0.25	-	0 - 0.4	0 - 0.3	0 - 0.15	0 - 0.45	0 - 0.4	-10 - 100	032U3614
G 1/2	EPDM ¹⁾	8.0	1.00	0 - 1.5	0 - 0.5	0 - 2.0	0 - 2.0	0 - 1.2	0 - 3	0 - 2.5	-30 - 120	032U3615
	FKM		1.00	0 - 1.5	0 - 0.5	0 - 2.0	0 - 2.0	0 - 1.2	0 - 3	0 - 2.5	-10 - 100	032U3616
	EPDM ¹⁾	10.0	1.50	0 - 0.8	0 - 0.3	0 - 1.1	0 - 1.2	0 - 0.6	0 - 1.6	0 - 1.3	-30 - 120	032U3617
	FKM		1.50	0 - 0.8	0 - 0.3	0 - 1.1	0 - 1.2	0 - 0.6	0 - 1.6	0 - 1.3	-10 - 100	032U3618
	EPDM ¹⁾	15.0	2.85	0 - 0.25	-	0 - 0.4	0 - 0.3	0 - 0.15	0 - 0.45	0 - 0.4	-30 - 120	032U3619
	FKM		2.85	0 - 0.25	-	0 - 0.4	0 - 0.3	0 - 0.15	0 - 0.45	0 - 0.4	-10 - 100	032U3620
G 3/4	EPDM ¹⁾	20.0	4.50	-	-	-	0 - 0.28	0 - 0.12	0 - 0.4	0 - 0.35	-30 - 120	032U3621
	FKM		4.50	-	-	-	0 - 0.28	0 - 0.12	0 - 0.4	0 - 0.35	-10 - 100	032U3622
G 1	EPDM ¹⁾	25.0	8.00	-	-	-	0 - 0.25	0 - 0.09	0 - 0.35	0 - 0.2	-30 - 120	032U3623
	FKM		8.00	-	-	-	0 - 0.25	0 - 0.09	0 - 0.35	0 - 0.2	-10 - 100	032U3624

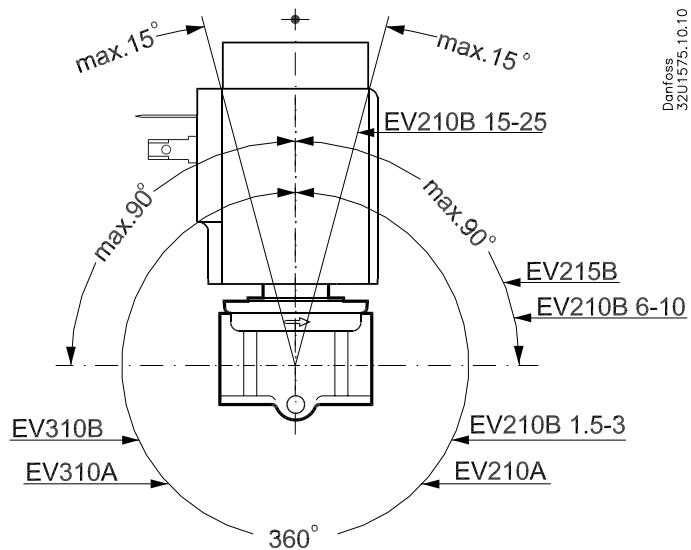
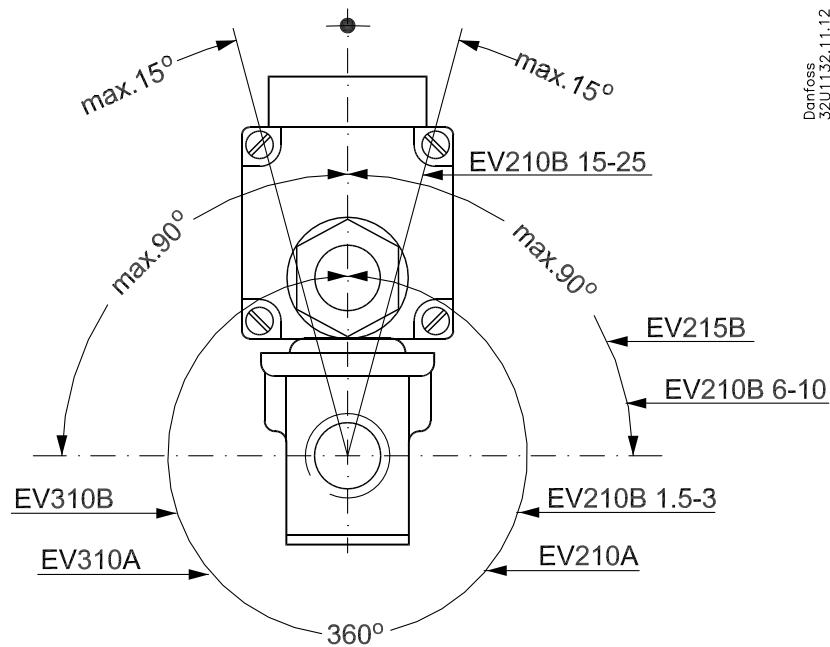
¹⁾ Vapor de baixa pressão de 140 °C / 3.6 bar, orifício DN 1.5 - 4.5.

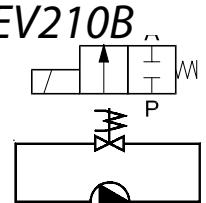
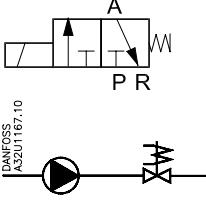
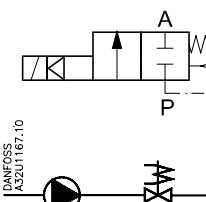
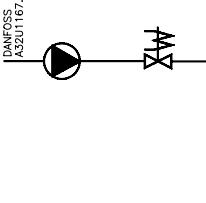
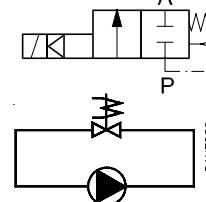
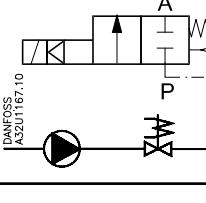
- Vapor de baixa pressão: DN 1.5 - 3 Usar bobina tipo BB ou BG.

DN 4.5 Usar bobina tipo BG.

²⁾ A faixa de pressão pode ser estendida para usar em um vácuo grosso, normalmente até 99% de vácuo (10 mbar), dependendo da aplicação.

Direct-operated valves



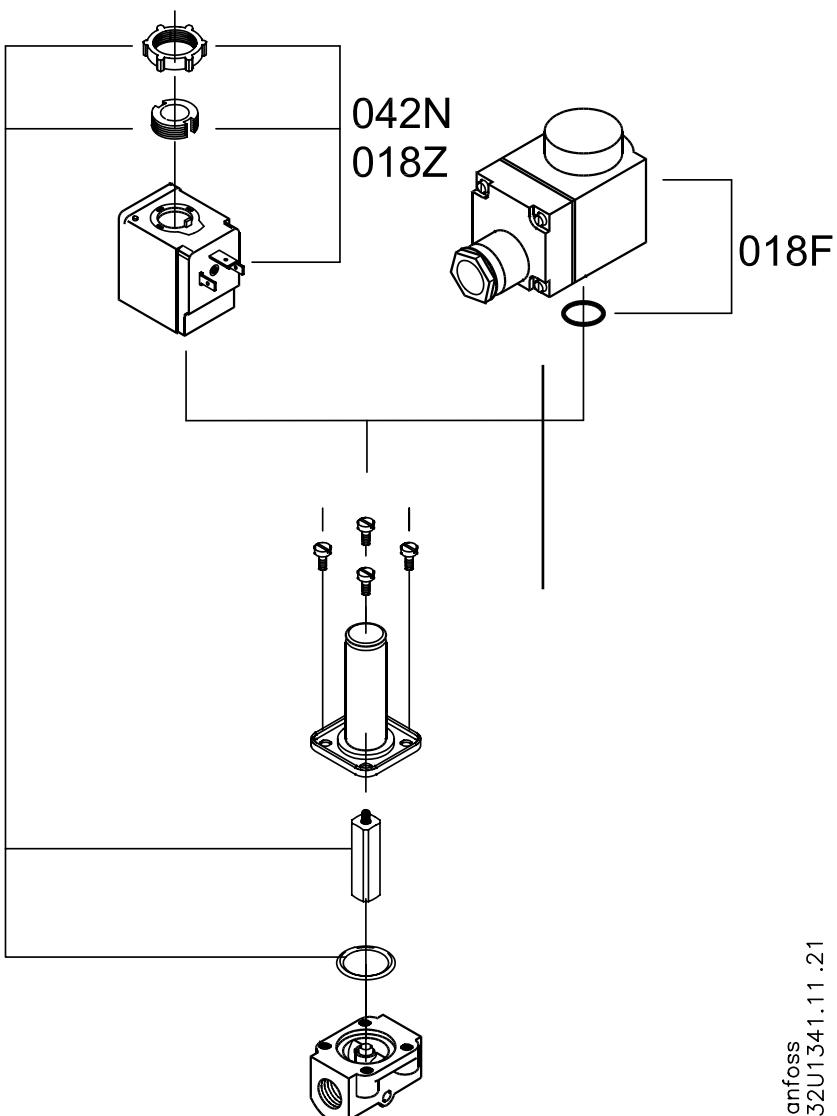
	Medium			
	Air and neutral gases	Water	Oil	Steam
EV210B   <small>DANFOSS A32U1166.10</small>	X	X	X	
EV310B   <small>DANFOSS A32U1167.10</small>	X	X	X	
EV220B   <small>DANFOSS A32U1167.10</small>   <small>DANFOSS A32U1166.10</small>	X	X	X	
EV250B   <small>DANFOSS A32U1166.10</small>	X	X	X	X
EV225B   <small>DANFOSS A32U1167.10</small>				X

For other characteristics please see the valve selection guide

Characteristics		Description
Connection [ISO 228/1]	Function	
G 3/8" - G 1"	NC/NO	EV210B covers a wide range of direct-operated 2/2-way solenoid valves for universal use. EV210B is a real robust valve program with high performance and can be used in all kind of tough working conditions.
G 1/8" - G 3/8"	NC/NO	EV310B is a direct-operated 3/2-way solenoid valve. The valve is especially used in connection with air-operated valves to allow air supply and air relief for the air actuator.
G 1/4" - G 1"	NC/NO	EV220B 6-22 is a direct servo-operated 2/2-way solenoid valve program. This program is especially for OEM applications demanding a robust solution and moderate flow rates.
G 1/2" - G 2"	NC/NO	EV220B 15-50 is a universal indirect servo-operated 2/2-way solenoid valve program. Valve body in brass, dezincification resistant brass and stainless steel ensures that a broad variety of applications can be covered.
G 3/8" - G 1"	NC	EV250B with assisted lift is especially to use in closed circuits with low differential pressure, but demanding moderate flow rates. Valve body in DZR brass ensures a long life, even in connection with aggressive steam media.
G 1/4" - G 1"	NC	The EV225B design is based on a PTFE diaphragm and valve body in dezincification resistant brass, ensuring high reliable function and long life even in connection with contaminated steam.

Spare parts set for EV210B NC

The spare parts set contains locking button and nut for coil, armature with valve plate and spring, and O-rings.



EPDM¹⁾ versions

Type	Code no.
EV210B 1.5 - 4.5	032U6000
EV210B 6,8,10	032U2006

FKM¹⁾ versions

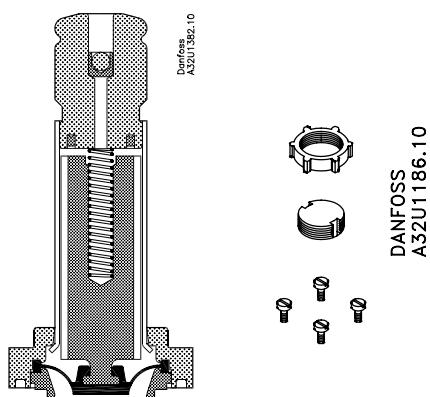
Type	Code no.
EV210B 1.5 - 4.5	032U2003
EV210B 6,8,10	032U2011

¹⁾ See page 20 for description of seal materials

Danfoss
A32U1341.11.21

Isolating diaphragm kit for EV210B 1.5-4.5 NC and EV220B 15-50 NC

Avoids build up of contaminates that can block movement of the armature. Permits use of more aggressive media that would normally attack the armature. Gel filled; guarantees operation after long periods of inactivity.



Seal material	Code no.
EPDM ¹⁾	042U1009
FKM ¹⁾	042U1010

¹⁾ See page 20 for description of seal materials

Anexo IV

Bobina Danfoss 042N7550

Data sheet

Solenoid coils



Danfoss solenoid valves and coils are usually ordered separately to allow maximum flexibility, enabling you to select a valve and coil combination to best suit your needs. The Danfoss coil program consists of both the easy-to-handle Clip-On system and traditional coils with threaded fastener. Also, with approvals such as EEx/ATEX and UL, we offer a wide range of application specific coils for e.g. steam or hazardous areas.

Features

- Encapsulated coils with long operating life, even under extreme conditions
- Standard coils for AC or DC
- Standard coils from 12 V to 400 V, 50, 60 or 50/60 Hz
- Coils can be fitted without use of tools
- Coils can only be removed with use of tools
- Standard coils available with:
 - Cable plugs
 - Industrial plugs
 - Terminal box
 - 3 core cable
 - Junction box
 - Conduit hub

BA, High performance coils



- Ambient temperature: Up to 40 °C
- IP00 version with DIN 43650 A spade connectors
- IP20 version with protective cap
- IP65 version with mounted cable plug

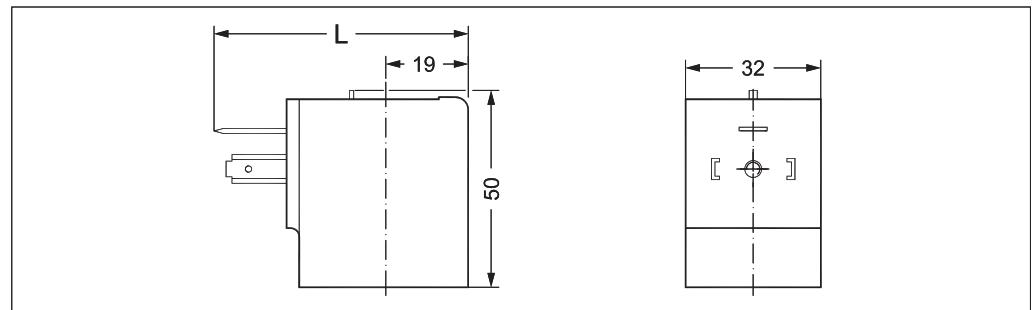
Coil type	Supply voltage		Frequency [Hz]	Power consumption holding [W]	Code no.
	[V AC]	[V DC]			
BA024A	24	–	50	9	042N7508
BA048A	48	–	50	9	042N7510
BA115A	115	–	50	9	042N7512
BA230A	220 – 230	–	50	9	042N7501
BA240A	240	–	50	9	042N7502
BA400A	380 – 400	–	50	9	042N7504
BA024B	24	–	60	9	042N7520
BA115B	115	–	60	9	042N7522
BA220B	220	–	60	9	042N7523
BA012D	–	12	–	15	042N7550
BA024D	–	24	–	15	042N7551

Technical data

Design	In accordance with VDE 0580		
Voltage variation	220/380 V AC	-15%, +10%	
	230/400 V AC	-10%, +6%	
	Other AC coils with NC valve	-15%, +10%	
	Other AC coils with NO valve and all DC	±10%	
Power consumption, cut in	39 VA AC coils only		
Insulation of coil windings	Class H according to IEC 85		
Connection	Spade connector in accordance with DIN 43650 form A		
Enclosure, IEC 529	IP00 with spade connec. IP20 with protective cap, IP65 with cable plug		
Ambient temperature	Max. 40 °C		
Duty rating	Continuous		
Plug type	Cable plug		

Dimensions and weight

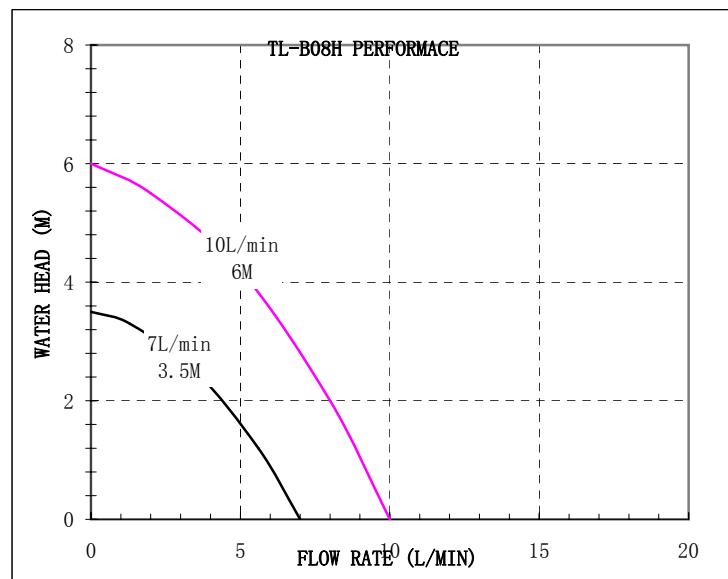
Type	L without cable plug [mm]	L with protective cap [mm]	L with cable plug [mm]	Weight [kg]
BA	54	71	79	0.16



Anexo V

Bomba centrífuga Topsflo B08H121006

TOPSFLO / TL-B08H



Model	Product Code	Max Flow Rate (L/M)	Rated Voltage (DC)	Current (A)	Max Water Head (M)	Power (W)
TL-B08H (100°C)	TL-B08H-12-0703	7	12VDC	0.9	3.5	11
	TL-B08H-12-1006	10	12VDC	1.6	6	20
	TL-B08H-24-0703	7	24VDC	0.5	3.5	12
	TL-B08H-24-1006	10	24VDC	0.8	6	20
can customize specification of TL-B08H-24-1208, but which only for ≤70°C liquid only						
TL-B08H/PV (100°C)	TL-B08H/PV-12/24-1006	10	12V (8V~26V)	0.8	6	20
	special design low starting, can be powered directly by solar panels, apply for both 12V and 24V voltage					
pump itself meet: I/H/F (high temperature 100°C / food grade) can be customized: S (submersible) ONLY for liquid ≤60°C, ambient temperature≤40°C						

Important Note:

TOPSFLO not confirmation also not recommend the TL-B08H or H/PV pump used for Solar Hot Water System Circulation, for the pump is belong to normal plastic circulation pump (not with specializing enhanced design) which can not meet 10bar system pressure.

TOPSFLO also not responsible for any quality problems happened in short time or years due to usage of this pump for Solar Hot Water System Circulation which usually with system pressure requirements to 10bar, also with all materials requirements should above 200°C,etc

