Allan Kollross

MÁQUINA DE FUMAÇA DE BAIXO CUSTO PARA TÚNEL DE VENTO DIDÁTICO

Joinville, SC

2015

Allan Kollross

MÁQUINA DE FUMAÇA DE BAIXO CUSTO PARA TÚNEL DE VENTO DIDÁTICO

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no Centro de Engenharias da Mobilidade da Universidade Federal de Santa Catarina, Campus de Joinville.

Universidade Federal de Santa Catarina – UFSC

Centro de Engenharias da Mobilidade

Bacharelado em Engenharia Mecatrônica

Orientador: Prof. Dr. Anderson Wedderhoff Spengler

Joinville, SC

2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Kollross, Allan
Máquina de fumaça de baixo custo para túnel de vento didático / Allan Kollross ; orientador, Anderson Wedderhoff Spengler - Joinville, SC, 2015.
98 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Joinville. Graduação em Engenharia Mecatrônica.

Inclui referências

1. Engenharia Mecatrônica. 2. Túnel de vento. 3. Instrumentação. 4. Projeto didático. I. , Anderson Wedderhoff Spengler. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Mecatrônica. III. Título.

À todos que são chamados por mim de família.

AGRADECIMENTOS

Agradeço a Deus pela força nos momentos de fraqueza. Sem Ele essa longa jornada não seria possível.

Aos meus pais Adolar e Aglaci pelo incentivo, determinação e ajuda em todos momentos de necessidade. Eles, que no decorrer da vida, me proporcionaram, além de amor e carinho, o conhecimento da integridade, perseverança e a busca sempre em Deus como a força maior para o meu desenvolvimento como ser humano.

À minha amada namorada Ana Carolina, que além de realmente me fazer feliz, contribuiu de forma determinante em todo o percurso de minha vida acadêmica.

Ao meu orientador Anderson, pela disposição, empenho e paciência em todos os momentos necessários.

À todos, meu muito obrigado!

"Seja você quem for, seja qual for a posição social que você tenha na vida, a mais alta ou a mais baixa, tenha sempre como meta muita força, muita determinação e sempre faça tudo com muito amor e com muita fé em Deus, que um dia você chega lá. De alguma maneira você chega lá."

(Ayrton Senna)

RESUMO

Os projetos didáticos estão sendo desenvolvidos por muitos profissionais da educação que notam a importância de concretizar formas diferentes de apresentar os conteúdos abordados na sala de aula. Dada a fase difícil que nosso país atravessa, com poucos recursos para o financiamento de projetos educacionais e também de pesquisas, os conceitos de baixo custo ganham preferência entre os docentes e, se a ideia é realmente interessante e está bem definida, é aceita e desenvolvida por profissionais de todo o país. Nesse contexto, o projeto da máquina de fumaça de baixo custo para o túnel de vento didático que está sendo desenvolvido no Campus Joinville da Universidade Federal de Santa Catarina procura apresentar um protótipo com aspectos construtivos simples, materiais reaproveitados, mas considerando o desenvolvimento tecnológico como requisito básico ao final do trabalho. O projeto buscou a caracterização dos requisitos de operação de cada frente de trabalho criada e o desenvolvimento deles no decorrer de execução do mesmo. Foram feitos ensaios de verificação comprovando o funcionamento da relação entre hardware e do software desenvolvidos para a geração de fumaça, garantindo, efetivamente, a criação do primeiro conceito de gerador de fumaça.

Palavras-chaves: Protótipo, Projeto, Didático, Túnel de Vento, Instrumentação.

ABSTRACT

Educational projects are being developed by many educators that notice the importance to present the knowledge in different ways to students at the classroom. Given the bad moment that our country is passing by, with few resources to finance educational projects and research, the low cost projects gain preference between teachers and, if the idea is really interesting and well defined, it is supported and developed by professionals from all over the country. In this context, the low cost fog machine for a didactic wind tunnel that is being developed on Campus Joinville of Santa Catarina Federal University aims to introduce a prototype with simple construction, using recycled materials, but considering the technological development as a basic requirement till the end of the work. The project aimed to characterize the operational requirements of each mode of operation created and their dynamic characteristics in the course of execution. Verification tests were made checking the relation between hardware and software for the great generation of smoke, ensuring, effectively, the creation of the first concept of smoke generator.

Key-words: Prototype, Project, Didactic, Wind Tunnel, Instrumentation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A/D	Analógico/Digital
I/O	Input/Output
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
LCD	Liquid Crystal Display
VI	Virtual Instruments
UART	Universal Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
CI	Circuito Integrado
PI	Proporcional-Integral
Ω	Resistência Elétrica

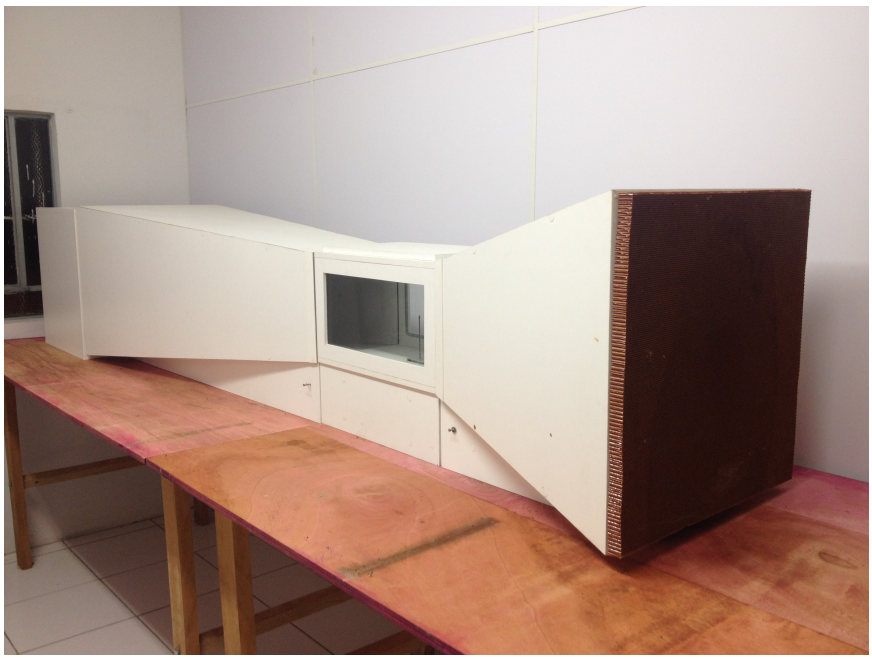
SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de conceitos dentro das instituições de ensino e a geração de conhecimento é uma tarefa realizada diariamente e envolve alunos e professores em assuntos multidisciplinares com temas complexos e bem elaborados. O campus de Joinville da Universidade Federal de Santa Catarina (UFSC) vem participando ativamente na criação de projetos que visam melhorar o aprendizado dos alunos de engenharia, inovando nas áreas de mecânica, elétrica, eletrônica, programação e outras, fortalecendo o nome do curso e da instituição.

O Túnel de Vento Educacional é um entre os projetos desenvolvidos na UFSC em Joinville. Ele tem o objetivo de auxiliar alunos e professores na visualização de fenômenos da aerodinâmica e fornecer condições para realizar tarefas como cálculo das forças envolvidas no escoamento, assim como observar o comportamento do escoamento do ar sobre um objeto. A Figura 1 mostra a estrutura do túnel que está em desenvolvimento.

Figura 1 – Túnel de vento didático da UFSC - campus Joinville.



Fonte: Grupo do projeto, 2015.

Bianchini e Gomes (2007) destacam os benefícios da utilização das ferramentas didáticas de ensino-aprendizagem nas instituições de ensino. Segundo os autores, no modelo de ensino atual, o aluno faz papel de elemento passivo, recebendo informações e repetindo-as, esperando aplicá-las em situações similares. Contudo, os trabalhos baseados em aprendizagem didática estão emergindo e se tornando cada vez mais frequentes,

buscando um rompimento definitivo do modo arcaico de ensino baseado em repetibilidade.

Assim como o trabalho de Bianchini e Gomes (2007), diversos professores de diversos centros educacionais estão motivando os alunos com projetos educacionais e didáticos, sejam esses trabalhos finais de disciplinas, projetos com bolsas de extensão, entre outros.

Já Chiofi e Oliveira (2014) argumenta que a utilização de uma "[...] ferramenta didática possibilita ao processo de ensino e aprendizagem uma aula mais dinâmica, interativa e contextualizada com a realidade dos alunos.". Dessa forma, o túnel de vento didático se encaixa nesse contexto, tornando matérias complexas, tais como as que tratam diretamente do escoamento de fluidos e suas características, mais interessantes sob o ponto de vista de poder planejar uma aula com simulações computacionais e, posteriormente, verificá-las na prática.

1.1 Objetivos

Com isso em mente, este trabalho procura abordar a construção de um dispositivo de geração de fumaça de baixo custo para produzir linhas de fumaça para visualização de escoamento no túnel de vento do campus Joinville da Universidade Federal de Santa Catarina. Esse dispositivo será chamado de "Máquina de Fumaça"e, para concretização dele, os seguintes objetivos devem ser alcançados:

- Objetivos específicos:
  - Conhecimento dos modelos presentes no mercado e modelos desenvolvidos em outros trabalhos;
  - Definição dos dispositivos necessários para tornar possível a geração de fumaça;
  - Projeto, desenvolvimento e construção de um protótipo funcional;
  - Realização de testes;
  - Comparação do custo benefício do projeto.

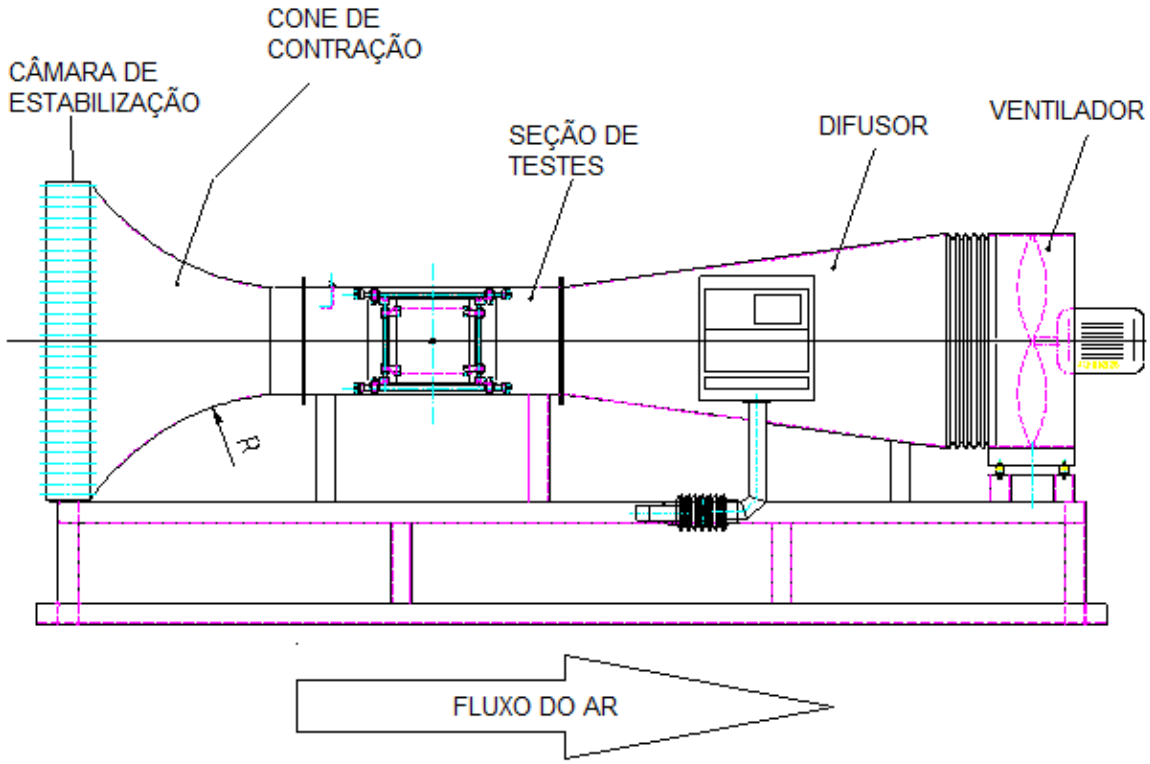
2 REVISÃO TEÓRICA

2.1 Túnel de Vento

Túneis de vento são estruturas utilizadas para analisar os padrões de escoamento dos ventos em torno de objetos sólidos (SINGH, 2008), fornecendo informações importantes para o estudo de problemas aerodinâmicos (COUTINHO, 2014). Pode ser considerado como uma das mais importantes ferramentas nas pesquisas de fenômenos aerodinâmicos em corpos expostos a escoamentos, já que ainda não se consegue reproduzi-los efetivamente em simulações realizadas por computadores (JUNIOR; FERREIRA; LETA, 2012). Os objetos a serem testados são mantidos estáticos na seção de testes, enquanto o ar é propulsado em seus entornos de modo que gere interações entre o fluxo de ar e o corpo de prova.

Para garantir a suavidade do fluxo de ar, Chandler (2011) diz que é necessário incorporar cinco seções básicas: A câmara de estabilização, o cone de contração, a seção de testes, o difusor e a seção de geração de fluxo de ar. A Figura 2 mostra essa divisão.

Figura 2 – Seções típicas de um túnel de vento.

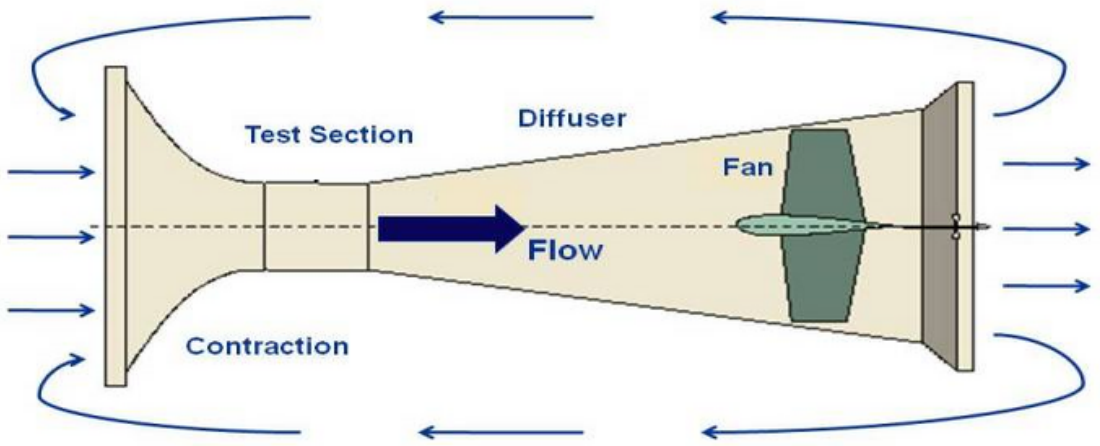


Fonte: (MARTINI et al., 2001).

Há duas configurações principais de túneis de vento. A primeira é chamada de túnel de circuito aberto e caracteriza-se por ter sua entrada e saída abertas para a atmosfera,

conforme a Figura 3.

Figura 3 – Túnel de vento de circuito aberto.



Fonte: (HALL, 2015).

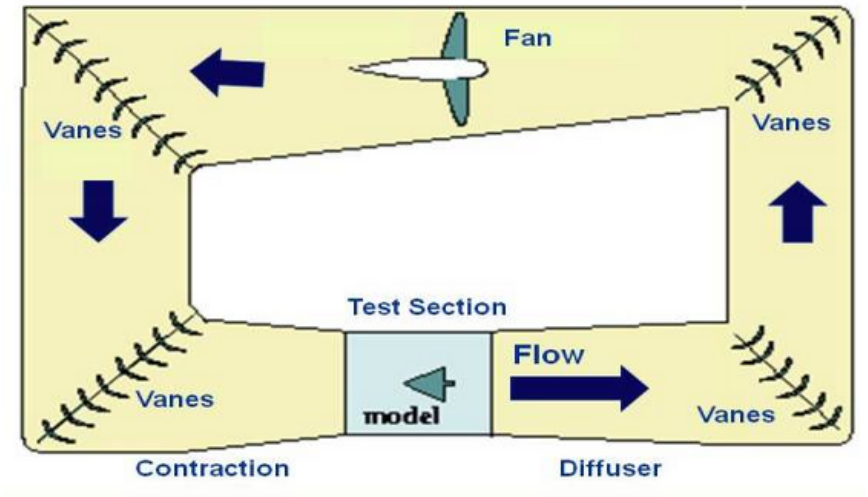
Essa configuração tem suas vantagens e desvantagens. Segundo Hall (2015), da Agência Espacial Americana (NASA), seu baixo custo construtivo e melhor desempenho no design de visualizadores de fluxo, como fumaça, por exemplo, são aspectos vantajosos. Como desvantagem, o documento cita o ruído causado na operação, já que o ventilador está próximo à câmara de testes, o alto custo operacional ocasionado pela contínua aceleração do ar realizada pelo ventilador, e, por fim, baixa qualidade no escoamento do ar através da câmara de testes devido à obstáculos como paredes, mesas, pessoas, produzindo a entrada assimétrica do ar na câmara de suavização. Outro fator que afeta o fluxo são as condições climáticas, visto que ventos e alteração das variáveis do ambiente podem causar grandes mudanças.

A segunda configuração encontrada é com circuito fechado, conforme a Figura 4 está mostrando. Nesse modelo o ar é conduzido da saída da seção de testes novamente para o ventilador através de canais com palhetas de direcionamento. Ao sair do ventilador ele retorna para o cone de contração e passa pela seção de testes, formando assim um ciclo.

De acordo com Hall (2015), túneis de vento de circuito fechado têm como vantagens maior qualidade no escoamento do ar na seção de testes, baixos custos operacionais, já que o ar está continuamente circulando no túnel e o ventilador, em regime permanente, precisa apenas suportar algumas perdas ocasionadas nas paredes e através das palhetas. Dessa forma, não há necessidade de acelera-lo constantemente. Por fim, há também o baixo ruído.

Ainda segundo Hall (2015), as desvantagens dessa arquitetura estão associadas com o alto custo construtivo, maior dificuldade de visualização do escoamento do ar através de fumaça ou outros meios, já que eles se acumulam no circuito e enriquecem o ar com

Figura 4 – Túnel de vento de circuito fechado.



Fonte: (HALL, 2015).

partículas não desejáveis. Também há o aumento da temperatura interna, o qual deve ser compensado utilizando dissipadores de calor ou climatizadores.

Além disso, túneis de vento são classificados de acordo com a velocidade máxima na câmara de testes, a qual se relaciona com a velocidade do som, caracterizando o termo Mach, o qual é a razão entre a velocidade de deslocamento e a velocidade do som. Túneis subsônicos operam com Mach abaixo de 0.8, transônicos entre 0.8 e 1,2, supersônicos entre 1,2 e 5 e hipersônicos conseguem velocidades maiores que 5 vezes a velocidade do som.

2.2 Visualização do escoamento

De acordo com Hall (2015), profissionais que trabalham com aerodinâmica utilizam túneis de vento para testar modelos de novos conceitos de aeronaves e componentes de motores, assim como automóveis, construções civis e, até mesmo, bolas de futebol. Durante um teste, o modelo é colocado na seção de testes do túnel e o ar é forçado a passar por ele. Em alguns testes, existe o interesse de verificar as forças aerodinâmicas produzidas. Já em outros, técnicas de visualização de escoamento são utilizadas para fornecer informações sobre o escoamento em torno do modelo.

Visualização de escoamento é um importante tópico quando se trata de dinâmica dos fluidos, de forma que tem sido assunto de muitas pesquisas ativas todos os anos. Tipicamente, dados de escoamento são gerados em simulações numéricas, tais como dinâmica dos fluidos computacional, e geralmente precisam ser analisados através de visualização física para ganhar melhor entendimento do escoamento. Com o rápido crescimento e desenvolvimento computacional para simulações, a demanda por métodos de visualização mais avançados tem crescido bastante (BRINZA; PRICOP, 2009).

As técnicas de visualização frequentemente utilizadas em túneis de vento são: Fumaça, tufos, feixe de laser, superfície de óleo e fotografia de Schlieren. Hall (2015) afirma que tufos e superfície de óleo são utilizadas para fornecer informações a respeito do estado da camada limite na superfície, particularmente para detectar regiões de separação, assim como de reatamento. Para esclarecer o conceito de camada limite, o autor comenta sua importância mencionando que ela é a fonte tanto de arrasto aerodinâmico quanto de perda da força de sustentação no momento em que ocorre a separação. Engenheiros sempre tentam determinar aonde isso ocorre e quais seus efeitos, para assim, melhorar o projeto da superfície.

De acordo com Smits e Lim (2012), um dos mais importantes atributos considerados em túneis de vento é sua capacidade de gerar fluxo contínuo e não-turbulento. Um ótimo equipamento possui nível de turbulência na ordem de 0,02%, mas, em termos práticos, números na ordem de 5% ainda são aceitáveis, como comentam Dol, Nor e Kamaruzaman (2006) em seu trabalho. Um dos fatores que influenciam na turbulência é o leiaute do túnel. Segundo Smits e Lim (2012), o melhor e mais utilizado leiaute para visualização de escoamento considerando túneis de circuito aberto, é o que possui um elemento que succiona o ar da seção de testes para o difusor. Outra consideração importante está relacionada com a velocidade do escoamento. Os escoamentos aqui descritos são idealizados como incompressíveis, de densidade constante e uniforme. Essas características são vistas em túneis de vento subsônicos, ou seja, de baixa velocidade de escoamento do ar.

Os métodos de visualização chamados ópticos, entendidos neste trabalho pela técnica de feixe de laser e fotografia de Schlieren, são amplamente aplicados em túneis de vento de alta velocidade de escoamento, visto que, como será explicado a seguir, somente podem ser notados quando há alteração na densidade do ar. Para exemplificar a utilização dos métodos de escoamento, os próximos tópicos descreverão algumas características básicas das técnicas de visualização comentadas até alcançar a visualização por fio de fumaça, tópico fundamental deste trabalho.

2.3 Visualização utilizando a técnica de feixe de laser

O método de visualização utilizando a técnica de feixe de laser foi empregado pela primeira vez em 1951 no laboratório de aeronáutica da National Advisory Committee of Aeronautics (NACA). Essa técnica é caracterizada pela inserção de vapor de água ou pequenas partículas no escoamento, dado que, como afirma Hall (2015), se movem juntamente com ele. Considerando que não é possível enxergar esse movimento, um feixe de laser é então introduzido no sistema, perpendicularmente ao escoamento, iluminando o vapor ou partículas, e tornando visível as linhas de fluxo em torno do modelo. A Figura 5 mostra o resultado dessa técnica de visualização.

Uma das considerações feita por Hall (2015) coloca o método de visualização por

Figura 5 – Escoamento em torno da superfície de um objeto utilizando o método feixe de laser.

Fonte: (HALL, 2015).

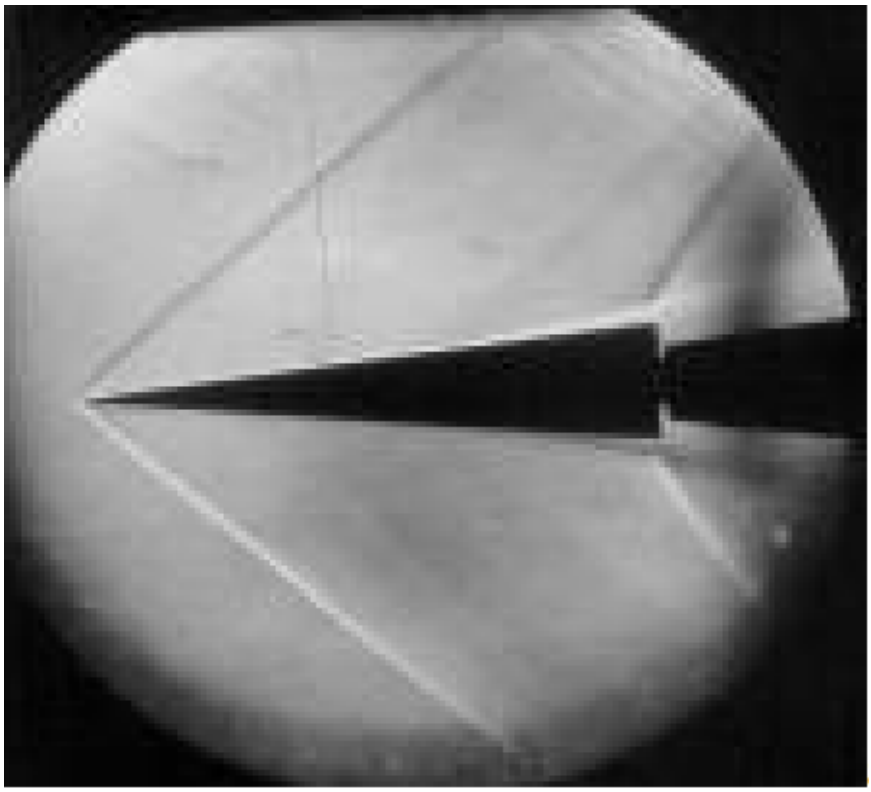
feixe de lasers como um dos melhores já desenvolvidos. Ele explica que isso se deve à praticidade de poder visualizar não só o escoamento, mas também determinar características como a velocidade local dele. Também comenta que uma desvantagem dessa técnica está relacionada com a incapacidade de visualizar áreas que não são iluminadas pelo laser, tal como, segundo a Figura 5, a parte abaixo do modelo.

2.4 Visualização utilizando a técnica fotografia de Schlieren

Ristić (2007b apud ROBERT, 2001) comenta que o método de Schlieren é um método óptico de visualização de escoamento que utiliza a variação da densidade do ar para verificar a deflexão angular de um raio de luz relativo à um meio transparente com não homogeneidades pontuais. Segundo o autor, é um dos métodos mais utilizados em laboratórios de aerodinâmica, dado que é simples e muito útil. A Figura 6 mostra um exemplo de imagem obtida utilizando o método de Schlieren.

Segundo Hall (2015), para ser possível aquisitar imagens como a mostrada na Figura 6 é necessário montar uma estrutura com dois espelhos côncavos, uma fonte de luz de alta luminosidade e dois anteparos. Quando iluminado, os raios de luz encontram gradientes de densidade do ar na seção de testes de modo que a luz é curvada ou refratada. Altos gradientes de densidade são encontrados em ondas de choque, as quais possuem como característica alterações significativas na pressão, temperatura e, consequentemente, na densidade. Os anteparos funcionam como barreiras para excluir a luz curvada da câmera

Figura 6 – Escoamento em torno da superfície de um objeto utilizando o método de Schlieren



Fonte: (HALL, 2015).

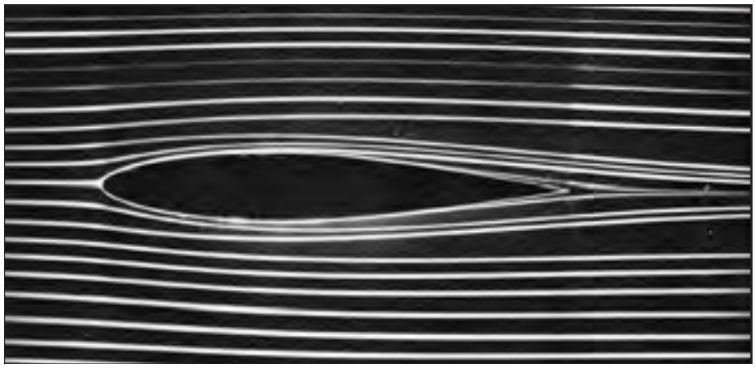
que está gravando a imagem, visto que na refração a luz é desviada para outra direção. Desse modo, a câmera não captura os pontos em que houve desvio, traduzindo isso como pontos mais escuros na imagem.

2.5 Visualização de escoamento pela técnica de fio de fumaça

De acordo com Çengel e Cimbala (2006), linhas de emissão são um conjunto das posições das partículas de fluido que passaram sequencialmente através de um determinado ponto do escoamento. Para visualizá-las, é necessário introduzir uma corrente de fluido visualizador, tal como tinta, no caso de escoamento com água, ou fumaça, no caso de escoamento com ar. Para ilustrar as linhas de emissão, Çengel e Cimbala (2006) apresenta a Figura 7, que é uma imagem real, obtida em laboratório, utilizando um perfil de asa e fios de fumaça para visualização do escoamento em um túnel de vento.

Ristić (2007a) comenta em seu trabalho que apesar da técnica de visualização de escoamento por fumaça ser uma das mais antigas formas de visualização, irá continuar como uma importante ferramenta no estudo da fluido-dinâmica. O autor também comenta sobre a importância dela ao estar aplicada principalmente em túneis de vento de baixa turbulência, mas completa que não há limite de velocidade de escoamento para garantir a

Figura 7 – Linhas de emissão em torno de uma asa.



Fonte: (ÇENGEL; CIMBALA, 2006).

visualização das linhas de fumaça.

Hall (2015) também destaca a importância do método exemplificando que pode ser utilizada para a detecção de vórtices e regiões de separação de escoamento. Além disso, esse método possui a grande vantagem de ser relativamente barato. Como desvantagem, o autor comenta a limitação dessa técnica quando aplicada a escoamentos acima de 500km/h, complementando a citação de Ristić (2007a).

Quanto ao tipo de fumaça que deve entrar na câmara de testes, Ristić (2007a) e Goldstein (1996) afirmam que ela deve ser densa e branca para visibilidade, preferencialmente não tóxica e não corrosiva.

2.6 Trabalhos similares

No trabalho apresentado por Rufato, Poletto e Bartex (2007), os autores comentam que trabalharam na evolução de uma máquina de fumaça anteriormente desenvolvida, na qual foi utilizado vapor d'água como fumaça, mas não foi eficaz pelo fato de condensar no decorrer de sua trajetória. No atual, é destacada a queima de óleo Spindura para a geração de fumaça e, segundo os autores, houve melhora significativa na estabilidade da fumaça.

A estrutura construída para transformar o líquido em vapor é um ebulidor blindado contendo uma resistência elétrica conectada a um acumulador. O funcionamento do sistema é basicamente manual, sendo o ebulidor acionado por corrente alternada provinda da rede elétrica e a dissipação da fumaça acumulada é feita por um cooler controlado com um reostato. Visualmente, busca-se a melhor relação entre a entrada e a saída da fumaça até que o regime laminar seja alcançado.

Já no trabalho de Felicio et al. (2007) é relatado o insucesso na construção de um dispositivo para geração de fumaça. Alguns materiais como tubos de cobre, uma bomba e uma resistência elétrica foram adquiridos e acoplados formando uma estrutura necessária para a geração de fumaça. Ao fazer testes com o dispositivo, foi verificado que o canal

longo entre a resistência e a saída da fumaça não permitiu sua liberação com boa qualidade e o projeto foi arquivado. Para contornar o problema, os autores utilizaram um modelo de máquina comercial para geração de fumaça e, dessa forma, concluíram o trabalho proposto.

Entre os diversos trabalhos disponíveis na literatura que relatam a utilização de fumaça como elemento de visualização de escoamento, são poucos os que tratam efetivamente da construção de uma máquina de fumaça. A solução geralmente aplicada é a compra de dispositivos comerciais.

2.7 Modelos comerciais

Segundo dados encontrados em pesquisas online, os modelos comerciais são divididos em diversas categorias disponíveis para compra. Entre elas se pode citar a potência da resistência de aquecimento, volume do compartimento para armazenar líquido, distância de lançamento da fumaça, resfriamento ativo e outras funcionalidades, como acionamento por controle remoto.

O preço desses modelos varia bastante conforme a melhoria da tecnologia aplicada ao produto. Como exemplo, uma máquina de fumaça simples pode ser encontrada por cerca de R$280,00, enquanto que outra mais complexa só é possível adquirir a partir de R$1400,00.

A característica de resfriamento ativo, encontrada em máquinas de fumaça que possuem um compartimento para a colocação de gelo ou gelo seco, chama a atenção. Esse recurso foi implementado para possibilizar que a fumaça quente seja resfriada e diminua sua densidade, de modo que fique mais pesada que o ar. Em eventos aonde são utilizadas, é interessante que a fumaça fique acumulada no piso após sua geração e seja dissipada no decorrer do tempo.

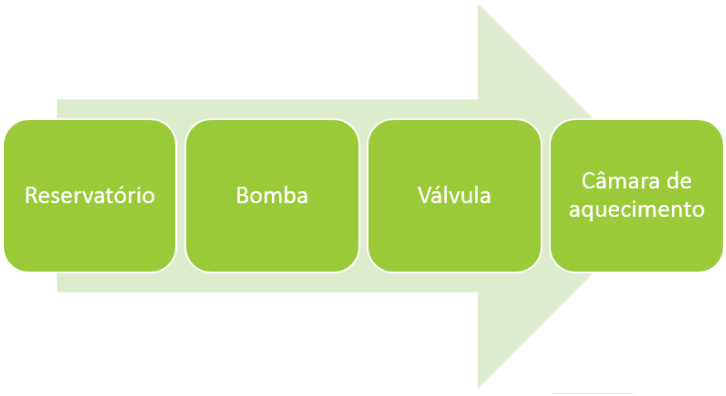
Nas especificações técnicas de modelos comerciais é possível sempre encontrar o comprimento máximo do jato de fumaça. Esses modelos possuem bicos de estrangulamento para que o gás quente e pressurizado na câmara de aquecimento perca pressão e tenha sua temperatura consideravelmente abaixada, permitindo que seja ejetado a distâncias maiores devido ao aumento da densidade.

3 MATERIAIS E MÉTODOS

3.1 Resistência, bomba e bico injetor

As máquinas de fumaça possuem três componentes básicos: Uma bomba, um reservatório e uma câmara de aquecimento. A Figura 8 exemplifica o fluxo básico do fluido, seguindo do reservatório para a câmara de aquecimento.

Figura 8 – Conceito esquemático para geração de fumaça.



Fonte: Autor, 2015.

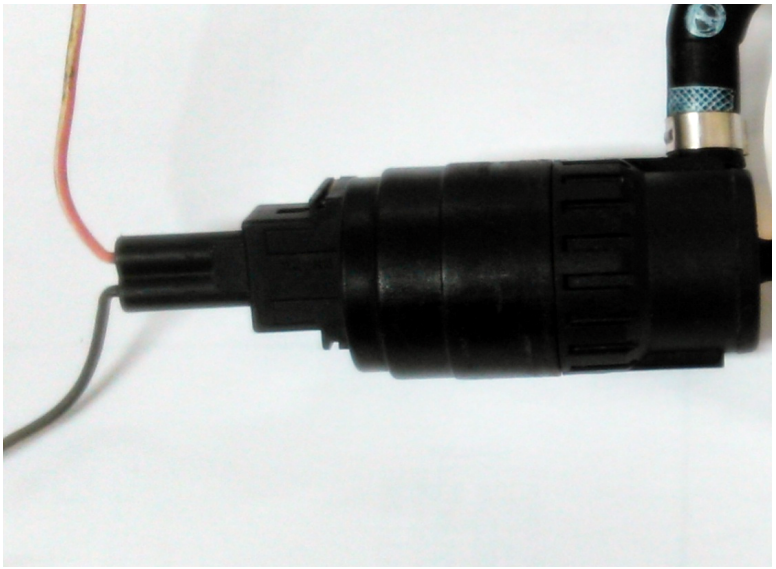
A bomba utilizada no projeto é mostrada na Figura 9. Ela foi adquirida em um centro de peças automotivas usadas, dado que é utilizada em automóveis no bombeamento de água para limpeza do para-brisas. Nenhum dado a respeito de suas características técnicas foi encontrado, mas sabe-se que seu acionamento é realizado com tensão de 12V em CC. Isso permite a modulação da tensão eficaz aplicada de forma que é possível, assim, controlar a vazão de descarga e limitar a quantidade de fluido que será evaporada.

O reservatório utilizado nos testes é um frasco de plástico, capaz de armazenar até 500ml, mostrado na Figura 10. Nas máquinas de fumaça comerciais o reservatório é maior, geralmente em torno de 2l para garantir a autonomia da geração de fumaça por algumas horas.

No caso da resistência, optou-se pelo modelo de fita dado sua facilidade de se adequar na superfície cilíndrica que compõe a câmara de aquecimento. A resistência utilizada é fabricada pela empresa Omega e é composta pela liga metálica Níquel-Cromo, a qual forneceu resistividade de $7,9\Omega$. A resistência já ajusta na câmara de aquecimento é mostrada na Figura 11.

Diferentemente dos modelos comerciais citados, os quais possuem um bico para oferecer resistência à saída fumaça para o ambiente, neste trabalho foi utilizado um bico injetor automotivo para controlar a injeção de fluido na câmara de aquecimento. Dessa

Figura 9 – Bomba automotiva.



Fonte: Autor, 2015.

Figura 10 – Reservatório.

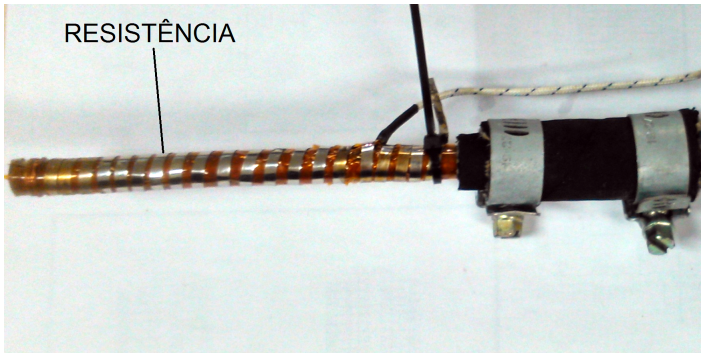
Fonte: Autor, 2015.

forma, além de modular a vazão de fluido com a bomba, o bico efetivamente restringe a passagem de fluido. A Figura 12 mostra o componente já agregado à estrutura.

3.2 Tiva C Series

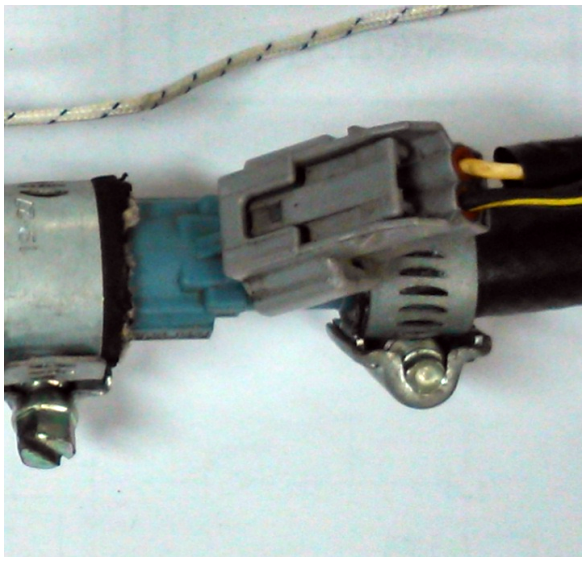
Para realizar o processamento dos comandos e ações que serão descritos, foi utilizado uma LauchPad Tiva C 123GXL, da Texas Instruments, apresentado na Figura 13. Ela

Figura 11 – Estrutura de aquecimento do fluido.



Fonte: Autor, 2015.

Figura 12 – Bico injetor automotivo.

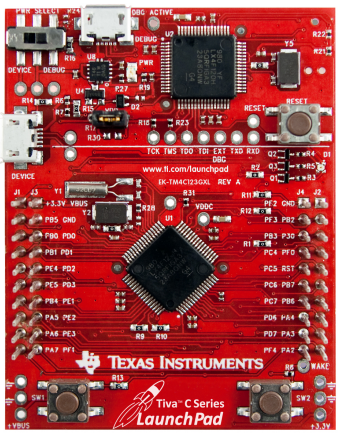


Fonte: Autor, 2015.

possui um microcontrolador ARM Cortex-M4F aliado aos seguintes recursos:

- 256kB Flash, 32kB SRAM, 2kB EEPROM
- Dois módulos CAN (Controller Area Network);
- USB 2.0 Host/Device/OTG + PHY;
- 2 ADCs 12-bit com 2 MSPS;
- 16 saídas PWM;
- 8 UART, 6 I2C, 4 SPI(SSI).
- 6 Timers.

Figura 13 – LauchPad Tiva 123GXL da Texas Instruments.



Fonte: (Texas Instruments, 2013).

Dentre os vários recursos disponíveis na Lauchpad, a implementação do trabalho requisitou a utilização de um módulo UART, dois módulos SPI e dois Timers, para execução periódica de funções. A comunicação através de UART é utilizada na transmissão de dados entre o microcontrolador e o computador. Para gerenciar cada operação, ela conta com interrupções no processamento permitindo exclusividade ao tratamento dos dados no barramento. Os módulos SPI foram implementados para receber dados de temperatura do componente MAX31855, além de comunicar dados a um display.

3.3 MAX31855

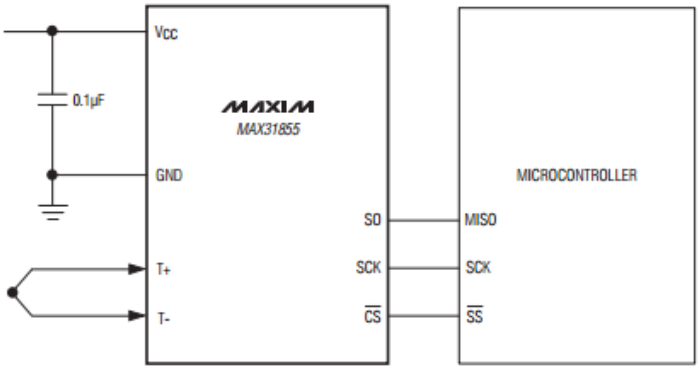
O componente MAX31855 desenvolvido pela MAXIM possui a característica de converter sinais de tensão gerados por termopares em sinais digitais. Além disso, possui integrado um sensor de precisão para medir a temperatura do componente e, internamente, realizar a compensação de junta fria. Seu conversor digital é de 14 bits e a resolução de de externamento dos dados é 0.25 °C. Quanto à precisão da medição, o erro associado no processo de leitura pode chegar até 2 °C quando utilizado termopares tipo T para ler temperaturas entre −270 °C e 400 °C.

Em sua arquitetura interna está implementado um sistema de detecção de erros de medição capaz de verificar quando o termopar está aberto, conectado ao aterramento ou conectado à tensão de alimentação. Para a máquina de fumaça são características indispensáveis, pois problemas na leitura de temperatura podem ser identificados e ações podem ser tomadas antes de ocasionar o descontrole do processo.

O circuito típico de ligação com a alimentação e com o microcontrolador é apresentado na Figura 14. O CI possui oito pinos responsáveis pela alimentação, que suporta tensões entre 3,0V e 3,6V e consome cerca de 0,9mA de corrente, sinal do termopar e

interface serial SPI para leitura dos dados processados.

Figura 14 – Esquema elétrico do CI MAX31855.



Fonte: (MAXIM, 2012).

O controlador interno do componente produz um conjunto de 31 bits para informar a temperatura do termopar, a temperatura do sensor interno e os bits indicando falha. A divisão desses dados é mostrada na Figura 15

Figura 15 – Dados externados pelo CI MAX31855.

	14-BIT THERMOCOUPLE TEMPERATURE DATA				RES	FAULT BIT	12-BIT INTERNAL TEMPERATURE DATA				RES	SCV BIT	SCG BIT	OC BIT
BIT	D31	D30	...	D18	D17	D16	D15	D14	...	D4	D3	D2	D1	D0
VALUE	Sign	MSB 2^{10} (1024°C)	...	LSB 2^{-2} (0.25°C)	Reserved	1 = Fault	Sign	MSB 2^{6} (64°C)	...	LSB 2^{-4} (0.0625°C)	Reserved	1 = Short to V_{CC}	1 = Short to GND	1 = Open Circuit

Fonte: (MAXIM, 2012).

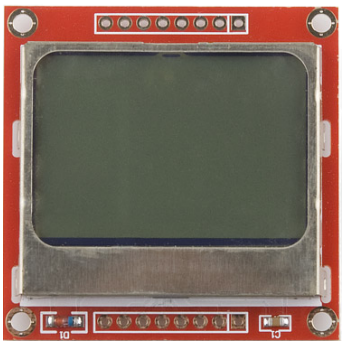
3.4 Display Nokia 5110

O display LCD 5110, mostrado na Figura 16 fabricado pela Nokia é capaz de representar 84x48 pixels, gerenciados por um controlador próprio. Este faz interface com um microcontrolador através de 8 pinos com funcionalidades divididas em alimentação, gerenciamento de dados e reset. A informação para atualização da tela é tratada por SPI através do protocolo fornecido no datasheet do componente.

3.5 Termopar

O conhecimento da temperatura de operação da câmara de aquecimento é determinante para o processo de geração de fumaça, pois é através dela que controlam-se características como densidade e quantidade de fumaça, visto que se pode injetar mais ou menos fluido.

Figura 16 – Display LCD Nokia 5110.



Fonte: (PHILIPS, 2012).

Termopares são dois fios metálicos de diferentes materiais, unidos em uma das extremidades. Seu princípio de operação é baseado na diferença de potencial existente entre o par, a qual possui uma relação com a temperatura de junta (THOMAZINI; ALBUQUERQUE, 2005).

Segundo Thomazini e Albuquerque (2005), a observação do fenômeno que caracteriza os termopares foi notada por pelo menos três pessoas na história, mas é geralmente atribuído à T. J. Seebeck, quando, em 1821, descobriu que em um circuito fechado feito com fios de materiais heterogêneos existe corrente elétrica fuindo da junção T1 para T2 se a temperatura da primeira estiver maior que a da segunda. A Figura 17 ilustra o circuito descrito.

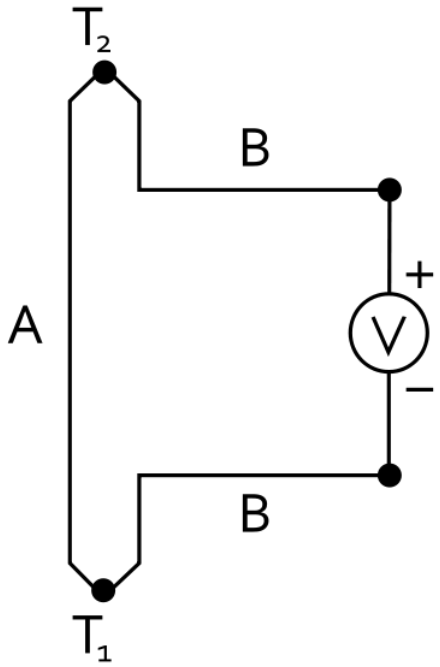
3.6 Fonte de alimentação

As necessidades energéticas do dispositivo são divididas em: alimentação da resistência, alimentação do bico injetor, da bomba e do microcontrolador. Para supri-las, foram utilizadas fontes similares à mostrada na Figura 18. Elas têm a capacidade de fornecer energia a tensão constante ou a corrente constante, numa escala de 0V a 30V continuamente ajustável, e até 3A de corrente de saída.

3.7 Code Composer Studio - CCS

O CCS é um ambiente de desenvolvimento de softwares para linguagem de programação C/C++ dedicado a atender todas as famílias de microcontroladores produzidos pela Texas Instruments. O programa possui recursos realmente interessantes, de forma que oferece ao usuário uma gama de ferramentas para desenvolvimento e depuração, além de uma interface intuitiva para ambos os casos. Entre os recursos disponíveis, está a edição e gerenciamento de arquivos e a depuração de linhas de código, informando rapidamente falhas na implementação. Além disso, ele é oferecido gratuitamente fabricante.

Figura 17 – Experimento realizado por Seebeck.



Fonte: (THOMAZINI; ALBUQUERQUE, 2005).

Figura 18 – Fonte de alimentação utilizada na validação do projeto.

Fonte: (ICEL Manaus, s.d).

3.8 LabVIEW

LabVIEW é uma linguagem de programação gráfica desenvolvida pela empresa National Instruments. Os principais campos de aplicação são a realização de medições, visto

sua versatilidade na comunicação com instrumentos de aquisição de dados e automação. A programação é feita de acordo com o modelo de fluxo de dados e realizada em arquivos chamados "Instrumentos Virtuais"(SOUZA, 2008). Esse software foi utilizado para a geração de uma interface gráfica de controle para a máquina de geração de fumaça.

3.9 Altium Designer

De acordo com IST Sistemas (s.d), "o Altium Designer é uma plataforma ECAD profissional, de alta tecnologia e curva de aprendizagem reduzida, utilizada para elaborar projetos eletrônicos desde os mais simples até os mais complexos, layout de placas de circuito impresso, [...]". Dessa forma, esse software auxilia no desenvolvimento e construção de circuitos eletrônicos.

4 PROJETO DA MÁQUINA DE GERAÇÃO DE FUMAÇA

O projeto da máquina de fumaça envolve a especificação de requisitos para três frentes de trabalho: a estrutura elétrica, estrutura mecânica e estrutura de software. O resultado esperado é que a união desses recursos torne possível a geração controlada de fumaça.

4.1 Projeto elétrico

Os requisitos para o projeto elétrico envolvem o acionamento dos periféricos controlados, o gerenciamento de energia para o microcontrolador e para o display. São eles:

- Necessidade energética de operação da bomba e o bico: 12V e até 2A;
- Necessidade energética de operação da resistência: 12V e até 5A;
- A potência sobre os componentes citados deve ser controlada;
- Tensão de alimentação da Lauchpad e do display LCD: entre 3V e 5V.

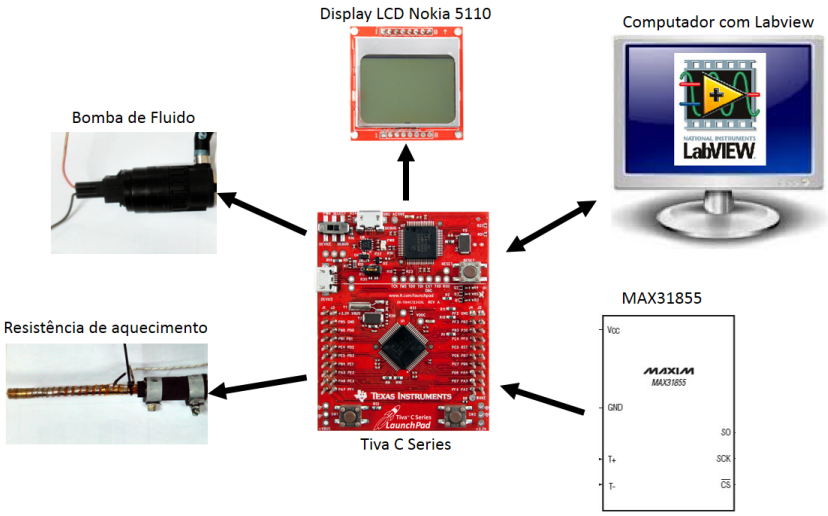
A partir dos requisitos elétricos dos componentes, o circuito do Apêndice A foi desenvolvido. Nele é mostrado que o acionamento dos componentes de potência foi realizado com o auxílio de transistores IRFZ44 do tipo MOSFET, modulando a tensão eficaz fornecida. Eles são acionados pelo driver TC4428CPA, o qual é acionado por um opto acoplador dirigido por um pino do microcontrolador. Este, por sua vez, ativa o gerenciamento de energia por meio da modulação de largura de pulso, conhecida por PWM. Os módulos PWM dedicados à bomba e ao bico serão configurados com frequência de comutação igual a 50Hz, enquanto que o PWM dedicado à resistência será configurado a 10Hz.

O Apêndice A também mostra que a alimentação da Launchpad e dos CI's acoplados à ela é derivada da alimentação fornecida aos componentes de potência. Entretanto, como operam com nível de tensão mais baixo, receberam o regulador de tensão modelo LM7805C, da Texas Instruments.

4.2 Projeto estrutural

Basicamente, o projeto estrutural precisa garantir que todas as peças sejam arranjadas para permitir o fluxo de fluido do reservatório para a câmara de aquecimento, assim como o controle desse processo. Com isso em mente, a Figura 19 foi elaborada e, a partir dela, é possível verificar o relacionamento entre os componentes citados no Capítulo 3.

Figura 19 – Arranjo dos componentes utilizados no projeto.



Fonte: Autor, 2015.

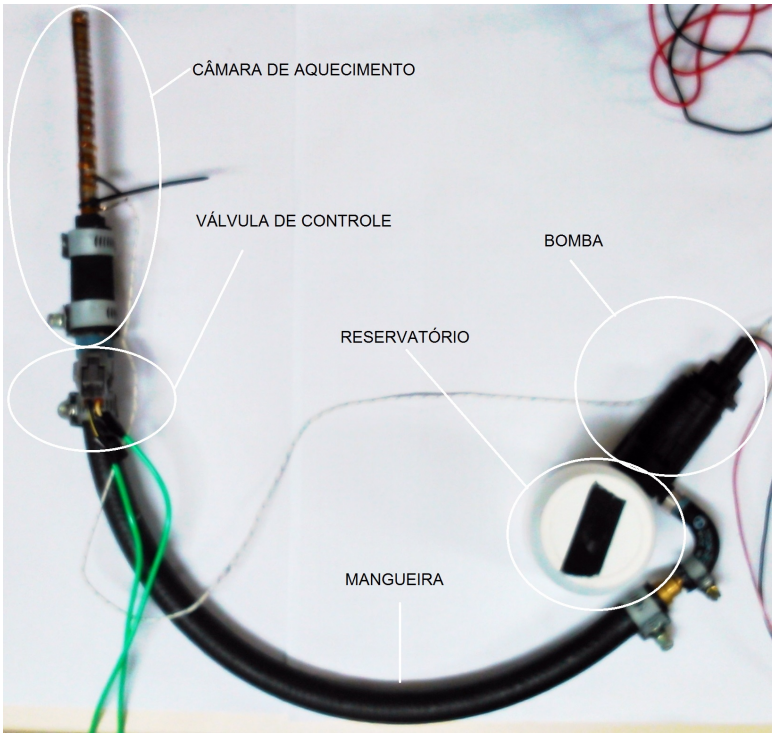
Essa estruturação resultou no conceito apresentado na Figura 20. A resistência, o bico e a bomba estão conectados por uma mangueira permitindo que a bomba retire fluido do reservatório e coloque no canal bloqueado pelo bico injetor. Quando permitida sua passagem, o fluido é lançado sobre a câmara aquecida e, devido à alta temperatura, evapora formando fumaça branca e espessa.

4.3 Projeto do firmware

Os requisitos necessários para o desenvolvimento do software são divididos em duas partes. A primeira referente ao microcontrolador trata do gerenciamento contínuo dos periféricos. A segunda é aplicada à interface gráfica para o computador. Com isso em mente, os requisitos da primeira são:

- Realizar a aquisição contínua da temperatura da resistência;
- Em caso de erro de leitura na temperatura, parar a geração de fumaça até que o problema seja resolvido;
- Possuir três modos de operação organizados da seguinte forma:
  - Modo 1: Geração contínua de fumaça;
  - Modo 2: Geração de fumaça por tempo determinado;
  - Modo 3: Livre acesso às variáveis.
- Gerenciar um byte que represente o status do sistema conforme a seguir:

Figura 20 – Conceito desenvolvido.



Fonte: Autor, 2015.

  - Bit 1: Problema de leitura no termopar;
  - Bit 2: Modo de Operação 1;
  - Bit 3: Modo de Operação 2;
  - Bit 4: Aquecimento da resistência ativado;
  - Bit 5: Bico injetor ativado;
  - Bit 6: Bomba ativada;
  - Bit 7: Valores forçados;
  - Bit 8: Sistema em operação.
- Comunicar ao computador todos os dados citados através de um protocolo.
- Atualizar continuamente o display;

Com relação ao firmware desenvolvido para o computador, ele deve ser capaz de:

- Possuir uma interface simples, funcional e amigável;
- Possuir controle sobre as variáveis de processo, tais como os modos de operação, temperatura e o acionamento dos periféricos;
- Atualizar as informações das variáveis de processo.

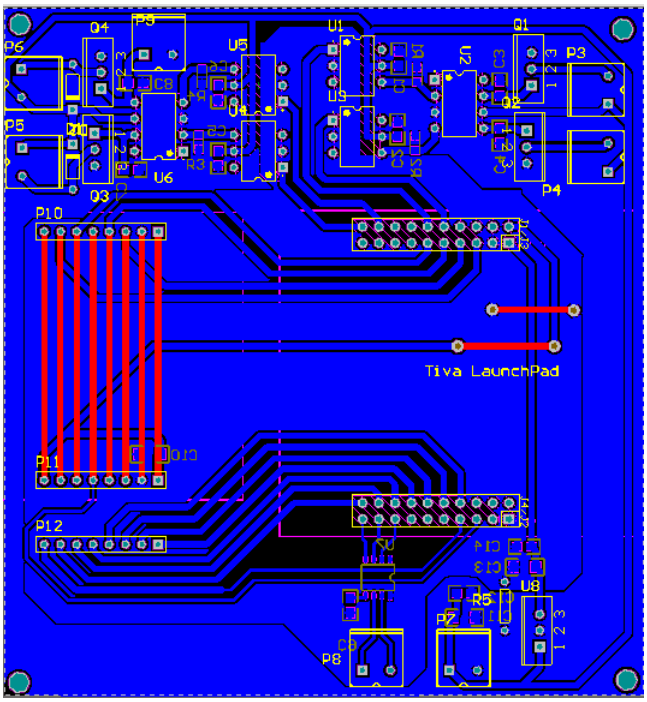
5 DESENVOLVIMENTO DO PROTÓTIPO

A criação do protótipo deve possuir como requisito a integração entre hardware e software, de modo que o hardware comporte o desenvolvimento da parte elétrica juntamente com a parte estrutural. Levando em consideração todos os itens descritos em 4, o desenvolvimento do protótipo foi dividido e estruturado em duas macrocélulas. A primeira reúne os requisitos estruturais e elétricos para a formação do hardware e a segunda agrega as necessidades dos firmwares para compor a parte lógica de operação e criar o software de gerenciamento da máquina de fumaça.

5.1 Hardware

Os requisitos elétricos apresentados na Seção 4.1 permitiram o desenvolvimento do projeto elétrico apresentado no Apêndice A. Este, por sua vez, através do software de geração de placas de circuito impresso Altium, efetivou a geração do layout da placa que comporta os elementos eletrônicos e as saídas de controle, assim como a entrada do sinal elétrico gerado pelo termopar. A Figura 21 mostra o resultado dessa operação.

Figura 21 – Leiaute da placa de circuito impresso constrída.



Fonte: Autor, 2015.

Considerando a descrição realizada na Seção 4.2, onde foi apresentada a forma de conexão entre os componentes que efetivamente geram a fumaça, o hardware do protótipo

se completa quando são realizadas as conexões elétricas entre eles e a placa eletrônica. Adicionalmente, conforme a Figura 20 exemplifica, pode-se considerar o computador como a última parte da estrutura. A estrutura resultante dessa descrição será mostrada no Capítulo 6.

5.2 Software

O desenvolvimento do software foi baseado nos requisitos citados na Seção 4.3, ocasionando o desenvolvimento da lógica de controle do sistema apresentada no Apêndice Apêndice B. A operação do dispositivo é iniciada após a configuração de todos os periféricos controlados. A função main() se responsabiliza por acessar as funções de inicialização e ajuste do clock, do MAX31855, da UART, dos módulos PWM, do display LCD e dos timers periódicos. Com essa etapa concluída, o modo de operação 1 é iniciado. A função de gerenciamento dos modos de operação inicia a execução infinita do programa e sempre está verificando qual é modo de operação ativo no momento.

As funções que gerenciam o funcionamento do dispositivo são baseadas em interrupções de comunicação e de tempo. Da parte temporal, duas rotinas estão implementadas. A primeira delas, executada numa taxa de 10Hz, trata da atualização dos novos dados gerados, enviando sempre os mais recentes para o display e para a interface com o computador. Já a segunda, executada a cada 1 segundo, atualiza um contador de tempo de referência, utilizado no modo de operação 2, além de executar as funções para obtenção de novos dados de temperatura.

A terceira interrupção implementada é destinada à comunicação com um computador. Nesse caso, é uma interrupção do módulo de recepção da UART, ou seja, sempre que são detectados dados no barramento, o microcontrolador se esforça para processá-los.

5.2.1 Modos de operação

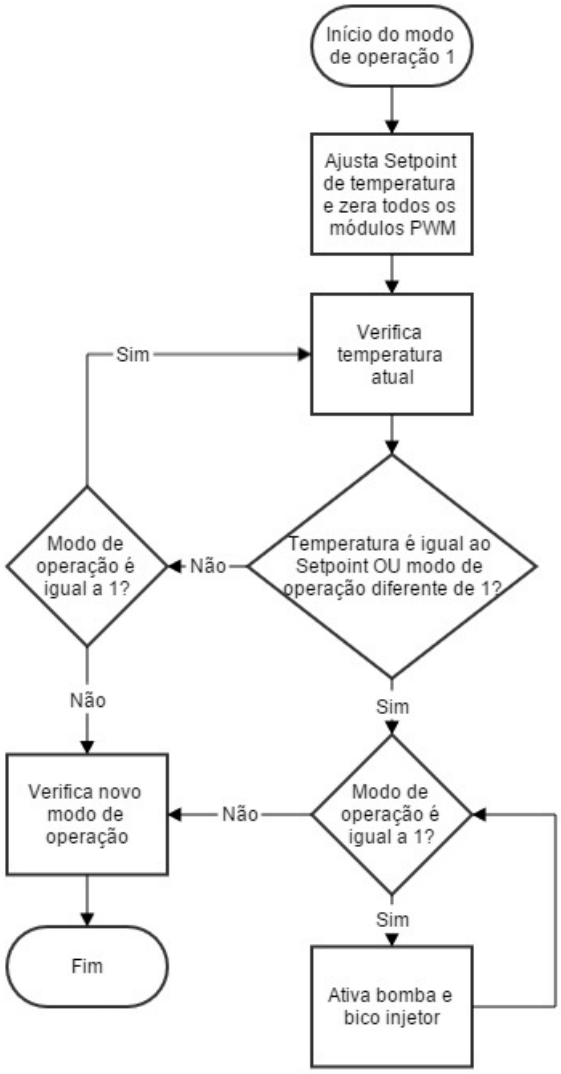
Para oferecer certa liberdade na utilização e em testes com o dispositivo, a máquina de fumaça foi desenvolvida para trabalhar com modos de operação diferenciados, cada qual caracterizando funcionalidades diferentes ao processo. Ao todo são três as opções de trabalho, divididas em: Geração de Fumaça Contínua, Geração de fumaça por tempo determinado e Modo forçado de Atuação.

5.2.2 Modo de operação 1

Quando o operador do dispositivo escolher por "Geração de Fumaça Contínua", o sistema é automaticamente configurado para trabalhar com 90% de razão cíclica da bomba, 100% de razão cíclica do bico injetor e setpoint de temperatura em 160 °C. Esses valores foram pré-determinados durante a fase de testes, quando foi analisado como

cada componente influencia na geração de fumaça. A rotina de operação deste modo foi implementada de acordo com o fluxograma mostrado na Figura 22.

Figura 22 – Fluxograma mostrando a lógica de execução do modo de operação 1.



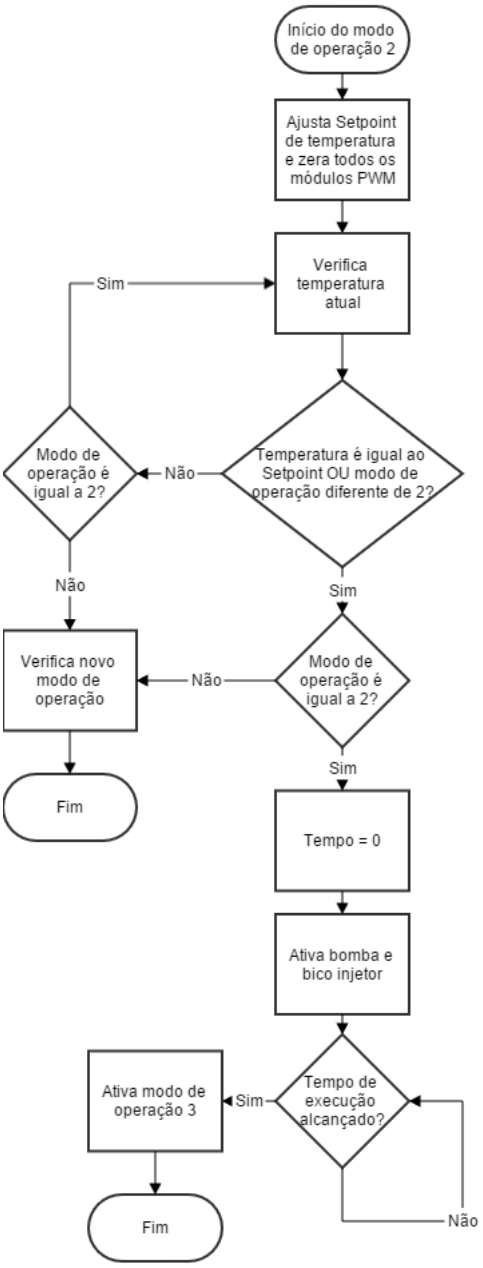
Fonte: Autor, 2015.

De acordo a Figura 22, a rotina do modo de operação 1 inicia forçando todos os atuadores do sistema a serem desligados. Em seguida, a resistência é ligada e o controle de temperatura passa a atuar sobre ela. Para iniciar a geração de fumaça, a temperatura deve, obrigatoriamente, alcançar o valor pré-determinado. Com isso, todos os sistemas são ativados e iniciam a operação normal de geração de fumaça. Para sair desse modo, o usuário precisa escolher outro modo de operação.

5.2.3 Modo de operação 2

A "Geração de Fumaça por Tempo Determinado"foi implementada para fornecer ao usuário fumaça durante um curto e definido intervalo de tempo. Esse modo busca, entre outros motivos, impedir que o dispositivo fique ativado por muito tempo, gerando gastos desnecessários de fluido e de energia. Para exemplificar como o modo é executado, a Figura 23 mostra o fluxograma do processo.

Figura 23 – Fluxograma mostrando a lógica de execução do modo de operação 2.



Fonte: Autor, 2015.

Levando em consideração a Figura 23, se pode notar que o modo de operação 2 é

configurado da mesma forma que o modo 1, já descrito anteriormente. A diferença entre eles está no gerenciamento do tempo, que inicia após os periféricos iniciarem a operação, já com a temperatura estabilizada. Quando o tempo configurado é alcançado, o dispositivo passa a operar no modo de operação 3 automaticamente.

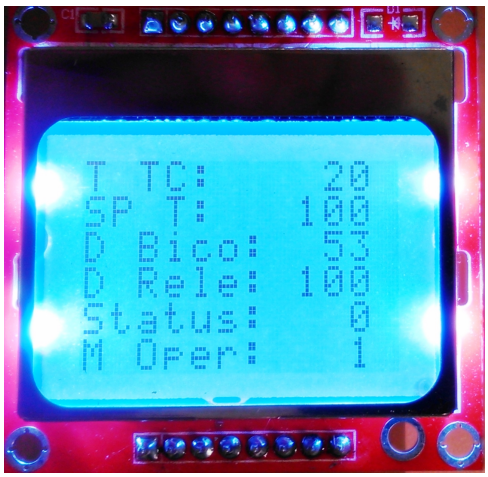
5.2.4 Modo de operação 3

Caso nenhum dos dois modos de trabalho seja escolhido, o usuário tem total autonomia sobre o gerenciamento dos periféricos ao ativar a opção "Modo Forçado de Atuação". Esse modo foi configurado para permitir a geração de novas configurações e, eventualmente, encontrar modos mais eficazes de gerar fumaça.

5.2.5 Comunicação

Os algoritmos de comunicação foram desenvolvidos para permitir a atualização dos dados de processo. A cada ciclo de interrupção dos temporizadores, especialmente do Timer0A, a chance de existirem novos dados de controle é grande. Segundo a o fluxograma mostrado no Apêndice B, a busca e verificação de informações atualizadas acontece nas interrupções do Timer1A. Essa atualização ocorre simultaneamente no display e na interface gráfica. No display ela é realizada com comunicação SPI, sendo os dados colocados no barramento e interpretados pelo controlador do dispositivo. Os dados enviados para ele são exibidos conforme demostrado na Figura 24. Já com o computador, a comunicação acontece via UART, através da porta USB.

Figura 24 – Dados exibidos no display.



Fonte: Autor, 2015

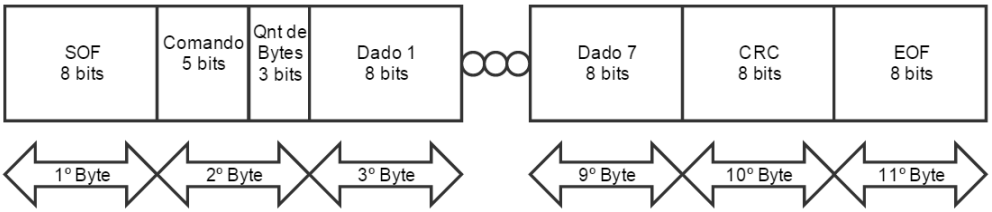
Como mostrado na Figura 24, a informação exibida na tela do display segue a seguinte sequência:

- Temperatura atual, abreviada como "T TC:";

- Temperatura alvo, abreviada como "SP T:";
- DutyCycle aplicado ao bico injetor, abreviado como "D Bico:";
- DutyCycle aplicado à resistência, abreviado como "D Res:";
- Status do sistema, abreviado como "Status:";
- Modo de operação, abreviado como "M Oper:".

Com o objetivo de adicionar robustez na troca de dados entre o microcontrolador e o computador, foi desenvolvido um protocolo de comunicação, o qual tem a forma mostrada na Figura 25. O byte de início de frame foi definido como o valor hexadecimal 0x02. O segundo byte é interpretado dividindo a informação contida nele em cinco bits ([7:3]) mais três bits ([2:0]). Os cinco bits são vistos como o comando da operação desejada, como, por exemplo, a transmissão da temperatura do termopar. Essa estrutura permite o protocolo suportar até 32 comandos independentes. Os três bits restantes comportam o tamanho do dado a ser transmitido, sendo que o limite de dados por transmissão de frame é de 7 bytes.

Figura 25 – Frame de dados enviado a cada troca de informação entre o microcontrolador e o computador.



Fonte: Autor, 2015.

O CRC é um mecanismo criado para empregar alguma lógica de processamento de dados a fim de detectar erros nas trocas de informação e recusar dados corrompidos. Para esse trabalho, a lógica utilizada na geração do byte CRC é a soma de cada byte recebido, desde o início da comunicação, até o último byte de dados, seguida da negação de todos os bits. Do número resultante dessas operações, retiram-se seus últimos 8 bits e define-se esse byte como o valor de checagem, conforme a Equação 5.1 e Equação 5.2.

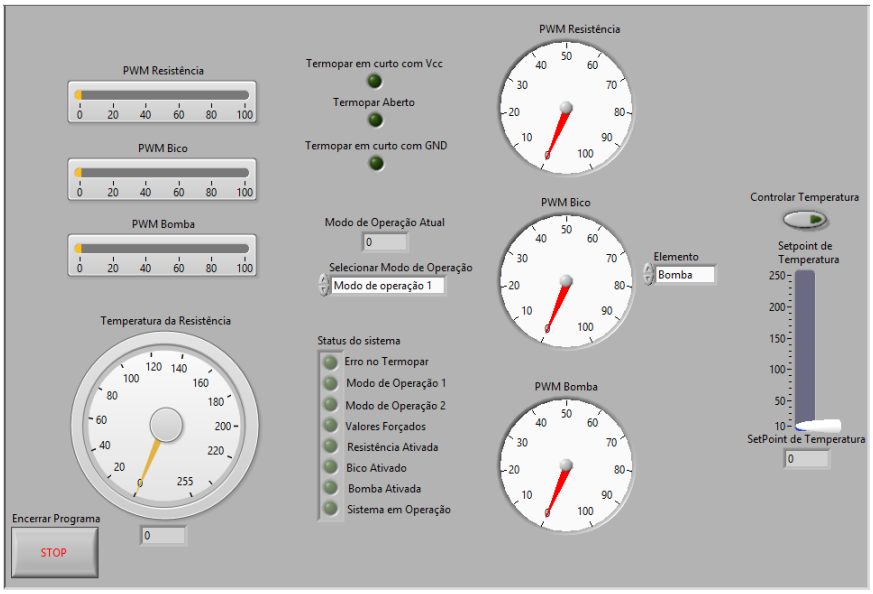
$$CRC = SOF + Byte_2 + Dado_1 + \ldots + Dado_7 \tag{5.1}$$

$$CRC = -CRC \tag{5.2}$$

5.2.6 Interface gráfica

A interface gráfica desenvolvida com o software LabVIEW é mostrada na Figura 26. Ela foi projetada para ter controle sobre as variáveis de processo da máquina de geração de fumaça, ou seja, utilizando o botão "Selecionar Modo de Operação", por exemplo, é possível escolher entre os três modos de operação citados anteriormente e, quando a escolha é realizada, o dado é enviado para o microcontrolador realizar as devidas ações.

Figura 26 – Interface desenvolvida em LabVIEW.



Fonte: Autor, 2015.

O modo de operação 3, como não possui valores pré-definidos, ativa as opções presentes do lado direito da interface, o que permite ao usuário selecionar livremente a potência sobre cada componente, numa escala de 0% a 100% variando a posição do ponteiro nos relógios. Permite também que o controlador de temperatura continue atuando sobre o processo se a opção "Controle de Temperatura"estiver selecionada. Nesse caso, fica indisponível a opção de gerenciamento da potência sobre a resistência, visto que o controlador PI realiza essa função.

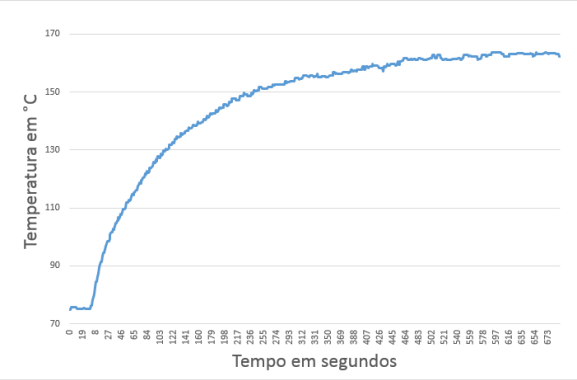
5.2.7 Controlador PI

Com o objetivo cumprir o requisito citado na Seção 4.1 para a temperatura, foi desenvolvido o projeto de um controlador PI e implementado no microcontrolador. Esse procedimento envolveu a identificação da planta, no caso a câmara de aquecimento.

O processo de identificação aconteceu aplicando-se dois pequenos degraus de temperatura em malha aberta até que a resposta do sistema em ambos os casos encontrasse o regime permanente. Como resultado, foi obtido primeiramente a estabilização da tempe-

ratura em 25 °C e, já no segundo degrau, a temperatura alcançou 163, 5 °C, assim como mostra a Figura 27

Figura 27 – Curva de resposta para o sistema em malha aberta.



Fonte: Autor, 2015

A técnica de identificação de sistemas de controle baseadas nos valores iniciais e finais da entrada e da saída do processo, forneceram para o sistema da Figura 27 a planta mostrada na Equação 5.3

$$\frac{y(s)}{u(s)} = \frac{0.05395}{s + 0.008655} \tag{5.3}$$

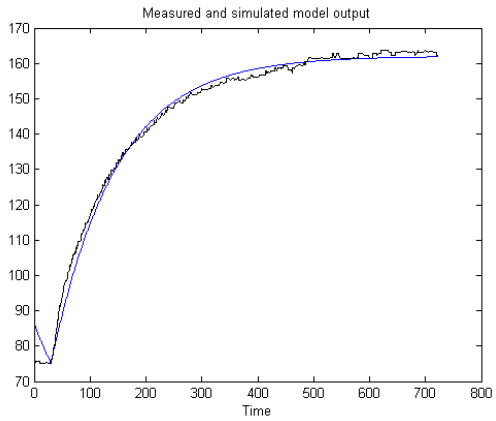
Para comparar o sistema real com o sistema estimado, foi gerada a Figura 28 representando a equivalência entre as duas plantas. A partir da função de transferência da Equação 5.3, é possível calcular os ganhos do controlador PI aplicado ao processo de aquecimento. Tendo em mente que o sistema é lento, o controlador foi projetado para acelerar o sistema para um tempo de resposta fixo em 30s e taxa de amortecimento de 0,7, ocasionando sobrepasso de 13,4%.

Seguindo as especificações anteriores, os valores encontrados para os ganhos proporcional e integral do controlador projetado foram 1,1507 e 0,035982. Na avaliação da resposta do controlador, é possível verificar com a Figura 29 que ele se comporta como o esperado. É importante observar que esse projeto não considerou perturbações no aquecimento, as quais serão geradas quando o fluido é injetado na câmara de aquecimento.

5.3 Custos de desenvolvimento do projeto

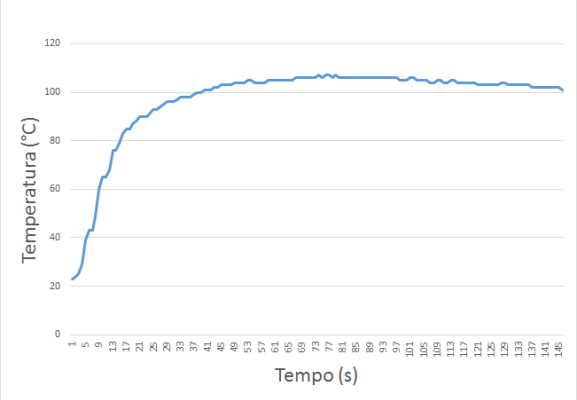
Por se tratar de um trabalho de baixo custo e alto desempenho, optou-se pela compra de materiais que já não estavam mais sendo úteis para a aplicação à qual foram destinados. Por isso, materiais como a bomba, o bico injetor e as mangueiras foram adquiridos de um centro de peças automotivas usadas por um custo relativamente baixo. Os outros componentes relevantes do projeto são descritos na Tabela 1.

Figura 28 – Comparação entre o processo real e estimado.



Fonte: Autor, 2015

Figura 29 – Avaliação do controlador projetado.



Fonte: Autor, 2015

Os componentes tipo CI's fazem parte de amostras grátis enviadas pela Texas Instruments e serão considerados doação para o projeto. Além disso, as fontes de alimentação utilizadas fazem parte da bancada do laboratório e também não serão consideradas na somatória dos valores. Sendo assim e tomando o Custo Total como base de preço para produzir um protótipo da máquina de fumaça, nota-se que ela ainda mais barata se comparada com o modelo comercial mais simples.

Entretanto, a relação custo-benefício dos dois modelos é muito diferente. No caso da máquina comercial, não há como atuar sobre suas características, não há controle da temperatura de evaporação do fluido, nem controle de fluxo de fluido. Esses fatores, por sua vez, colocam o conceito desenvolvido neste trabalho acima do modelo comercial tomado como base.

Tabela 1 – Custos relevantes para o desenvolvimento do projeto

Produto	Custo (R$)
Bico injetor + mangueiras + bomba	R$ 100,00
Abraçadeiras	R$ 4,00
Resistência	R$ 4,00
LauchPad Tiva C	R$ 48,00
Display Nokia	R$ 24,90
Placa de Fenolite	R$ 2,70
Custo Total	R$ 183,60

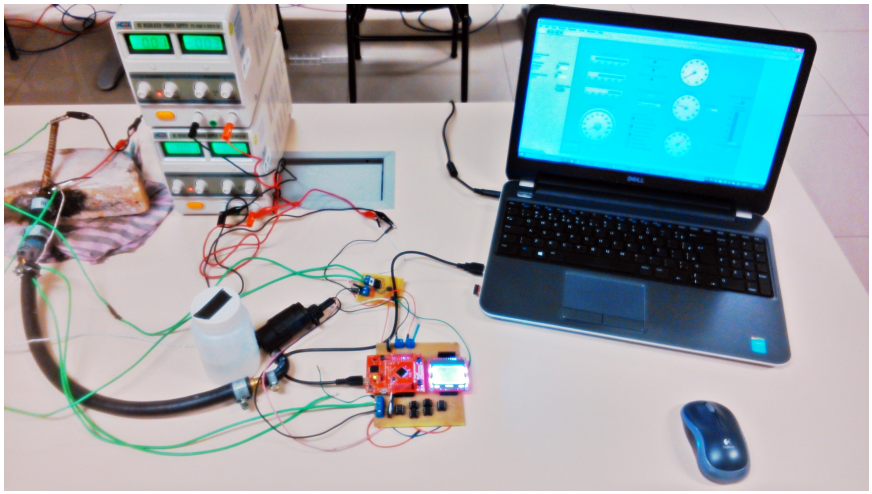
Fonte: Autor, 2015

6 RESULTADOS

Os resultados esperados buscam a análise do comportamento dinâmico do projeto nos modos de operação descritos. Dessa forma, os modos 1 e 2 foram utilizados para aquisição de dados e verificação do comportamento dos componentes durante as etapas de cada processo.

Sendo assim, a realização do projeto descrito até agora resultou no protótipo apresentado na Figura 30. Nessa imagem, ele está montado na bancada do laboratório com as fontes chaveadas alimentando o circuito e um computador executando o programa de monitoramento e controle. Esse arranjo foi utilizado para a realização dos testes de verificação de operação para validação do projeto.

Figura 30 – Estrutura completa para geração de fumaça.



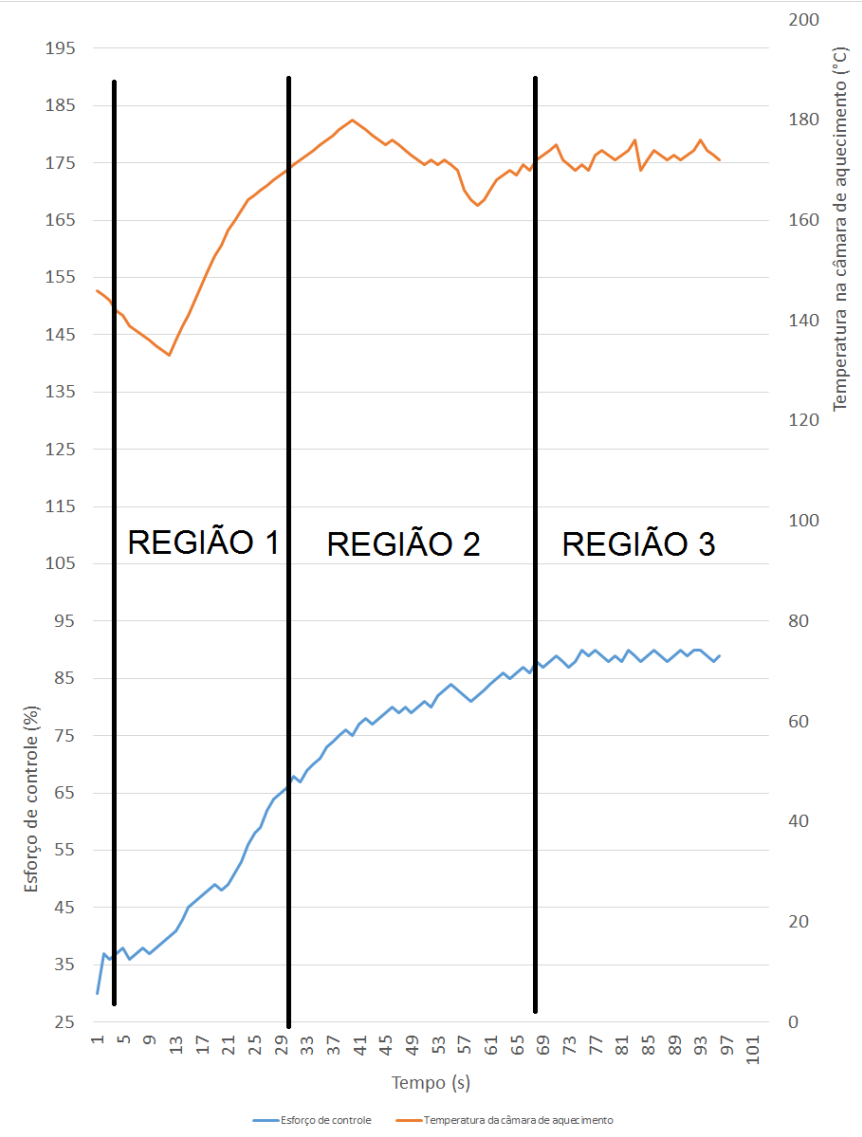
Fonte: Autor, 2015.

O primeiro teste de geração de fumaça utilizando o modo de operação contínuo é mostrado na Figura 31. A curva superior da imagem representa a variação de temperatura ao longo do teste. Já a inferior, o esforço do controlador de temperatura. Nesse teste, nota-se que a temperatura inicial da câmara de aquecimento já estava alta, mas diminuindo.

O início do teste acontece no tempo de 5 segundos, conforme mostra o delimitador da chamada "região 1". Até o tempo de cerca de 30 segundos, o controlador se esforça para esquentar a câmara de aquecimento, que está vazia. O início da operação da máquina de fumaça acontece quando a temperatura atinge 175 °C, ativando a bomba e a válvula de controle, de modo que fluido é injetado para ser evaporado. Nesse momento acontece a diminuição da temperatura da câmara e o esforço constante do controlador para mantê-la aquecida no nível desejado. A partir da região 3 o regime transitório acaba e a fumaça é

gerada continuamente.

Figura 31 – Modo contínuo de funcionamento.

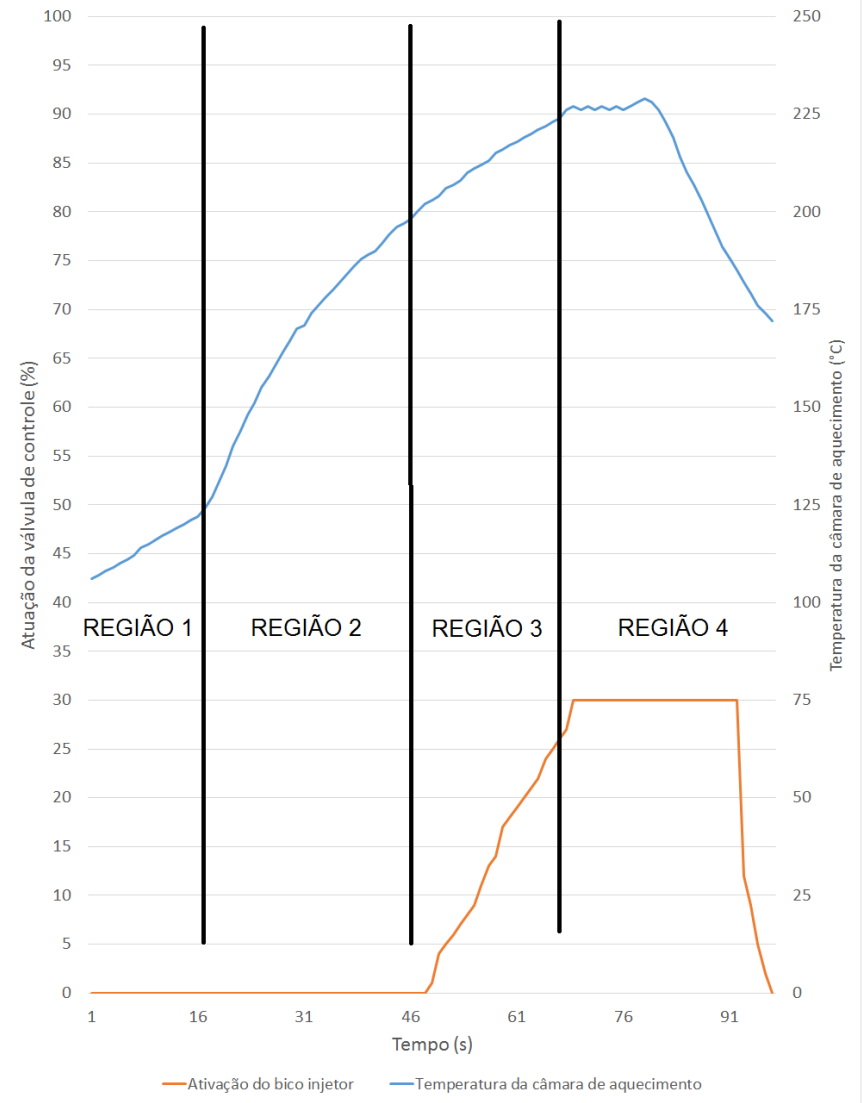


Fonte: Autor, 2015.

No segundo teste efetuado foi verificada a resposta do sistema para o modo de operação 2, no qual o dispositivo opera por determinado tempo gerando a maior quantidade de fumaça. A Figura 32 mostra a dinâmica do funcionamento para o caso em questão. A temperatura inicial na realização do teste foi cerca de 125 °C, indicada pela região 1. A região 2 marca o primeiro regime transitório, no qual o modo de operação é ativado e o controlador de temperatura atua para alcançar o valor programado, nesse caso, 200 °C. Em seguida, inicia o segundo transitório, no qual a válvula é ativada e sua abertura continuamente incrementada, até atingir o limite pré estabelecido de 30%. Na região 4 ocorre efetivamente a atuação do modo de operação 2, sendo a temperatura mantida constante durante todo o intervalo de tempo. Quando o limite é alcançado, a câmara de

aquecimento é desligada, mantendo a atuação da válvula de controle ativa até cerca de 175 °C.

Figura 32 – Modo de funcionamento.



Fonte: Autor, 2015.

Considerando os testes descritos anteriormente, é possível afirmar que as condições propostas no projeto foram atendidas. Analisando os itens críticos desenvolvidos, se pode afirmar que o controlador de temperatura se comportou como esperado, a comunicação entre o microcontrolador e o computador é operacional e o sistema, por sua vez, atua satisfatoriamente aquisitando as variáveis necessárias e controlando todo o conjunto de informações programadas.

7 CONSIDERAÇÕES FINAIS

O desenvolvimento deste trabalho mostrou que é possível a realização de uma máquina de fumaça de baixo custo que opere de maneira eficaz, fornecendo fumaça para avaliações de escoamento sobre objetos no túnel de vento da Universidade Federal de Santa Catarina. Como ele ainda está em desenvolvimento, não foi possível a realização de testes práticos para verificação do comportamento da máquina.

Para trabalhos futuros, sugere-se o acionando da parte elétrica com fontes independentes e isoladas para o fornecimento mais capacidade de corrente para melhorar a dinâmica da câmara aquecimento. Também se pode considerar a construção de mais câmaras de aquecimento dispostas em forma de linha para obter resultados similares à Figura 7.

Além disso, é necessário a construção de uma estrutura que acomode tanto a eletrônica do dispositivo, quanto a estrutura mecânica. É recomendado que a câmara de aquecimento receba isolação térmica de modo que garanta que menos calor seja desperdiçado para o ambiente. Também seria interessante o aumento da capacidade do reservatório para permitir maior tempo de funcionamento.

Levando em consideração a facilidade de obtenção de outras formas de trabalho do dispositivo no modo de operação 3 desenvolvido, espera-se que os usuários explorem incansavelmente as relações entre a injeção de fluido na câmara de aquecimento, a temperatura da mesma e a relação de bombeamento de fluido. Apesar de um ponto bom de trabalho ter sido encontrado, ele pode não ser o melhor que o produto oferece.

Os códigos implementados no microcontrolador para o gerenciamento das características da máquina de fumaça e os arquivos que contém a programação da interface gráfica, estão inseridos no Apêndice C.

REFERÊNCIAS

BIANCHINI, D.; GOMES, F. de S. C. A simulação como ferramenta didática no ensino de engenharia. *XXXV Congresso Brasileiro de Educação em Engenharia COBENGE*, 2007.

BRINZA, I.; PRICOP, M.-V. Methods of flow visualization. *INCAS - BULLETIN*, 2009.

ÇENGEL, Y. A.; CIMBALA, J. M. *Fluid Mechanics: Fundamentals and Applications*. [S.l.]: McGraw-Hill, 2006.

CHANDLER, N. *How Wind Tunnels Work*. 2011. Disponível em: <http://science.howstuffworks.com/wind-tunnel.htm>.

CHIOFI, L. C.; OLIVEIRA, M. R. F. de. O uso das tecnologias educacionais como ferramenta didática no processo de ensino e aprendizagem. *III Jornada de Pesquisas Didáticas*, 2014.

COUTINHO, F. R. *Projeto de um túnel de vento subsônico do tipo soprador*. [S.l.], 2014.

DOL, S. S.; NOR, M. A. M.; KAMARUZAMAN, M. K. An improved smoke-wire flow visualization technique. *4th WSEAS International Conference on Fluid Mechanics and Aerodynamics*, 2006.

FELICIO, G. et al. *Visualização de escoamentos com gases no túnel aerodinâmico*. [S.l.], 2007.

GOLDSTEIN, R. *Fluid Mechanics Measurements*. 2. ed. [S.l.]: CRC Press, 1996.

HALL, N. *Wind Tunnel Index*. 2015. Disponível em: <https://www.grc.nasa.gov/www/k-12/airplane/shortt.html>.

ICEL Manaus. *Manual de instruções da fonte de alimentação modelo PS-4000*. [S.l.], s.d. Disponível em: <http://icel-manaus.com.br/imagens/produtos/PS-4000%20manual.pdf>.

IST Sistemas. *Altium Designer*. [S.l.], s.d. Disponível em: <https://istsistemas.com.br/altium-designer/>.

JUNIOR, D. P. de S.; FERREIRA, G. G.; LETA, F. R. Desenvolvimento de um túnel de vento compacto para simulações didáticas de projetos de engenharia mecânica. *VII Congresso Nacional de Engenharia Mecânica*, 2012.

MARTINI, M. A. et al. *Módulo didático para ensaio de perfis aerodinâmicos - Túnel de vento*. [S.l.], 2001.

MAXIM. *Cold-Junction Compensated Thermocouple-to-Digital Converter*. [S.l.], 2012. Disponível em: <https://www.adafruit.com/datasheets/MAX31855.pdf>.

PHILIPS. *48 x 84 pixels matrix LCD controller/driver*. [S.l.], 2012. Disponível em: <https://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf>.

RISTIĆ, P. E. S. Flow visualisation techniques in wind tunnels part i non optical methods. *Scientific Technical Review, Vol.LVII, No.1*, 2007.

RISTIĆ, P. E. S. Flow visualization techniques in wind tunnels optical methods (part ii). 2007.

ROBERT, A. Schlieren measurement technique. 2001.

RUFATO, E.; POLETTO, R.; BARTEX, S. *Visualização de escoamento de fluidos em túnel de vento*. [S.l.], 2007.

SINGH, K. P. C. *Implementation of a LabVIEW-Based Automated Wind Tunnel Instrumentation System*. [S.l.], 2008.

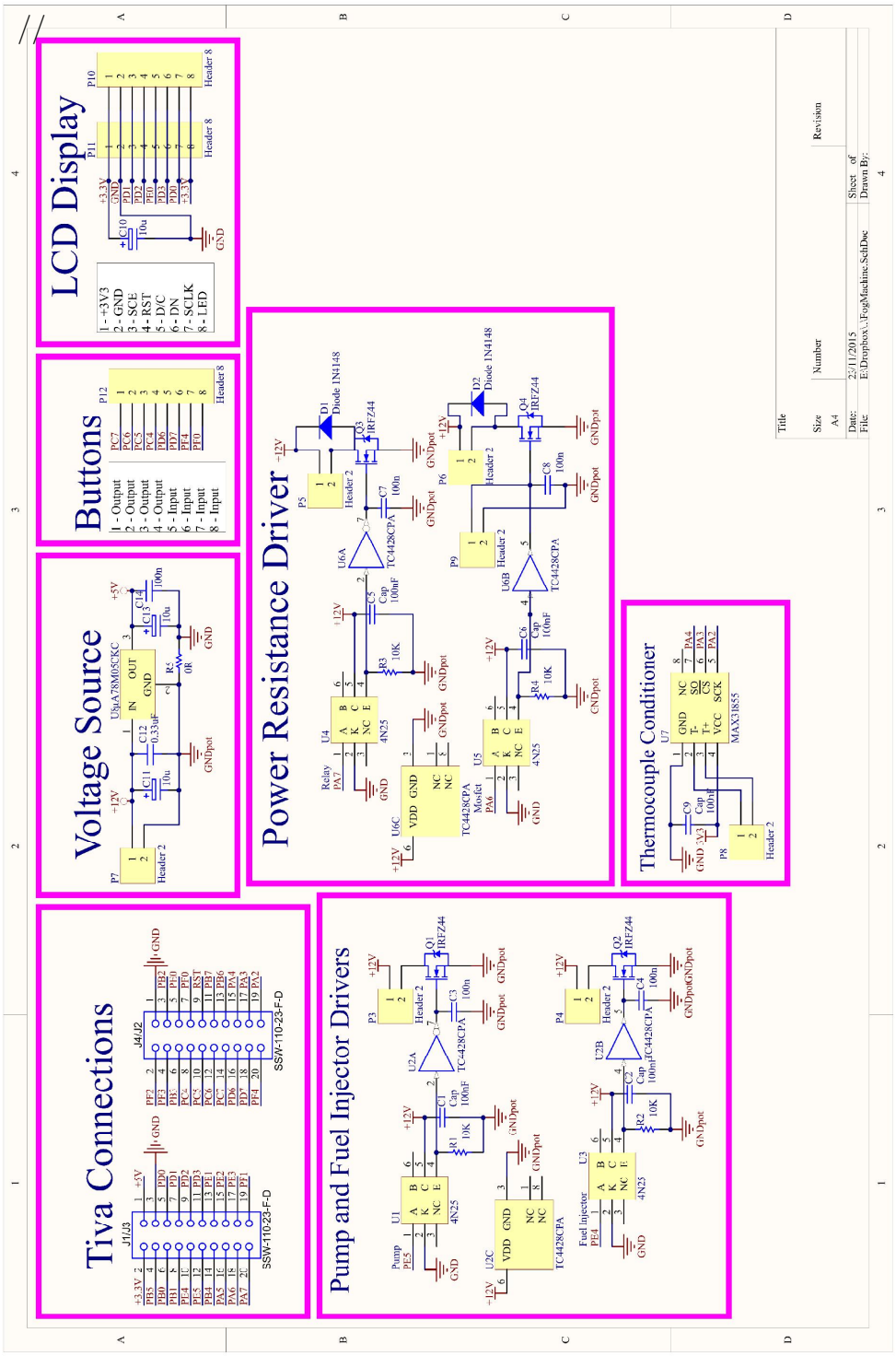
SMITS, A. J.; LIM, T. T. *Flow Visualization: Techniques and Examples*. 2. ed. London: Imperial College Press, 2012.

SOUZA, V. A. *Artigo Sobre LabView*. [S.l.], 2008.

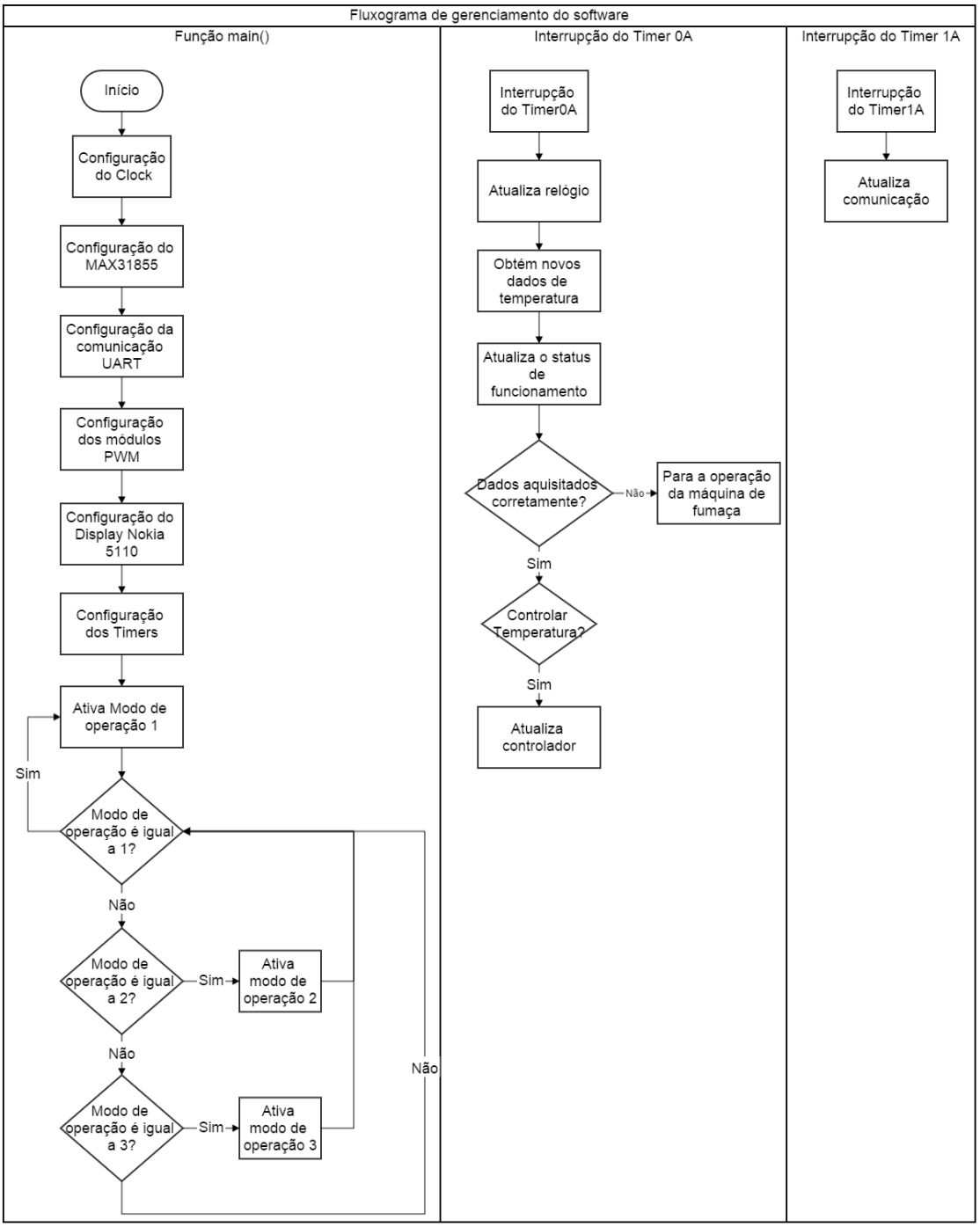
Texas Instruments. *Tiva TM4C123GH6PM Microcontroller*. [S.l.], 2013. Disponível em: <http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf>.

THOMAZINI, D.; ALBUQUERQUE, P. U. B. de. *Sensores industriais: Fundamentos e Aplicações*. 5. ed. São Paulo: Érica, 2005.

Apêndices


LCD Display
Buttons
Voltage Source
Power Resistance Driver
Thermocouple Conditioner
Tiva Connections
Pump and Fuel Injector Drivers
Title
Size A4
Number
Revision
Date: 25/11/2015
File: E:\Dropbox\...\FogMachine.SchDoc
Sheet of
Drawn By:

APÊNDICE B – FLUXOGRAMA DE EXECUÇÃO DO SOFTWARE



APÊNDICE C – CÓDIGOS DESENVOLVIDOS

```c
/*
 * Função main(): Inclui as bibliotecas dos periféricos utilizados no trabalho. A partir dela, todas as configurações são carregadas para possibilitar
 * a perfeita operação do dispostivo gerador de fumaça.
 */

#include "includes.h"
#include "UART.h"
#include "protocol.h"
#include "TIMER.h"
#include "MAX31855.h"
#include "PWM.h"
#include "nokia5110_pub.h"


//uint32_t sysClock = 0;
//uint8_t flagLED = 0;



void UARTIntHandler(void){
    uint32_t ui32Status;
    uint8_t UARTdata;

    // Get the interrrupt status.
    ui32Status = UARTIntStatus(UART0_BASE, true);

    // Clear the asserted interrupts.
    UARTIntClear(UART0_BASE, ui32Status);
    //uint8_t index=0;
    // Loop while there are characters in the receive FIFO.
    while(UARTCharsAvail(UART0_BASE))
   {
        // Read the next character from the UART and write it back to the UART.
        UARTdata = UARTCharGetNonBlocking(UART0_BASE);
        readDataFrame(UARTdata);
   }
}

/*
 * Função: Timer0AIntHandler(void):
 *
 * Definição: Destino da interrupção gerada pelo Timer0A.
 * Atuação: Responsável pela atualização do tempo em segundos e também pela obtenção das variáveis relacionadas à temperatura.
 */
void Timer0AIntHandler(void)
{
    static uint8_t flagLED = 0;
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
    }
    else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
```

```
56      }
57      TimeCounter();
58      UpdateMax();
59  }
60
61  /*
62   * Função: Timer1AIntHandler(void):
63   *
64   * Definição: Destino da interrupção gerada pelo Timer1A.
65   * Atuação: Responsável pela atualização das variáveis de comunicação a cada interrupção.
66   */
67  void Timer1AIntHandler(void)
68  {
69      TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
70      CommunicationUpdate();
71  }
72
73  /*
74   * Função: main(void):
75   *
76   * Definição: Função principal para o início do programa.
77   * Atuação: Inicialização dos periféricos e chamada das funções que suportam o funcionamento do
     programa.
78   */
79  int main(void) {
80
81      SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
82
83      setupMax();
84      UARTInit();
85      PWMInit();
86      inicializa_display();
87      timer0Init();
88      timer1Init();
89      FnOperationMode(1);
90  }
91
```

```c
/*
 * includes.h contém todas as bibliotecas utilizadas no trabalho e é chamada a em cada arquivo header (.h) para carregar todas as configurações
 * necessárias
 */

#ifndef INCLUDES_H_
#define INCLUDES_H_

#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "inc/hw_ssi.h"
#include "driverlib/pwm.h"
#include "driverlib/rom.h"
#include "driverlib/timer.h"
#include "inc/tm4c123gh6pm.h"
#include "driverlib/pin_map.h"

#define FALSE 0
#define TRUE  1
#define OK    0
#define FAIL  1

#endif /* INCLUDES_H_ */

```

```h
#ifndef MAX_UART_DISPLAY_LIBRARY_MAX31855_H_
#define MAX_UART_DISPLAY_LIBRARY_MAX31855_H_

#include "includes.h"

void setupMax();

void getDataMAX(uint8_t pvec[5]);
uint8_t TCtemp(uint8_t *ptemp0, uint8_t *ptemp1);
uint8_t TCtempDec(uint8_t *ptemp1);
uint8_t sensorTemp(uint8_t *ptemp2);
uint8_t sensorTempDec(uint8_t *ptemp3);
uint8_t errorType(uint8_t *ptemp3);
uint8_t bitStatus(uint8_t *ptemp1);

#endif

```

```c
#include "MAX31855.h"

/*
 * Função: setupMax():
 *
 * Definição:
 * Atuação: Executa os comandos necessários para acesar os dados gerados pelo CI MAX31588
 */
void setupMax()
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
    GPIOPinConfigure(GPIO_PA2_SSI0CLK);
    GPIOPinConfigure(GPIO_PA4_SSI0RX);
    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3);
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, 0xFF);
    GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_4);
    SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_1,SSI_MODE_MASTER,
500000, 8);
    SSIEnable(SSI0_BASE);
}

/*
 * Função: interpreta_botao_entrada(int linha, int coluna):
 *
 * Definição:
 * Atuação:
 */
void getDataMAX(uint8_t pvec[5]) {
    uint8_t i;
    uint32_t a;
    for (i = 0; i < 4; i++) {
        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, 0); //CS
        SSIDataPutNonBlocking(SSI0_BASE, 'A');
        while (SSIBusy(SSI0_BASE)) {}
        SSIDataGetNonBlocking(SSI0_BASE, &pvec[i]); // Read the data
        while (SSIDataGetNonBlocking(SSI0_BASE, &a) > 0);
    }
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, 0xFF); //CS
    //Fazer função para atualizar valores na tela
}

/*
 * Função: interpreta_botao_entrada(int linha, int coluna):
 *
 * Definição:
 * Atuação:
 */
uint8_t TCtemp(uint8_t *ptemp0, uint8_t *ptemp1) {
    //return *ptemp0 << 4 | *ptemp1 >> 4;
    return (*ptemp0<<6 | *ptemp1 >> 2) >> 2; // o ultimo >> 2 retira os bits que somam as
casas decimais
}

/*
 * Função: interpreta_botao_entrada(int linha, int coluna):
 *
```

```c
 * Definição:
 * Atuação:
 */
uint8_t TCtempDec(uint8_t *ptemp1) {
    uint8_t auxiliar;
    auxiliar = *ptemp1;
    auxiliar <<= 4;
    auxiliar >>= 6;
    return auxiliar; // o ultimo >> 2 retira os bits que somam as casas decimais
}

/*
 * Função: interpreta_botao_entrada(int linha, int coluna):
 *
 * Definição:
 * Atuação:
 */
uint8_t sensorTemp(uint8_t *ptemp2) {
    return *ptemp2;
}

/*
 * Função: interpreta_botao_entrada(int linha, int coluna):
 *
 * Definição:
 * Atuação:
 */
uint8_t sensorTempDec(uint8_t *ptemp3) {
    uint8_t auxiliar;
    auxiliar = *ptemp3;
    auxiliar >>= 4;
    return auxiliar;
}

/*
 * Função: interpreta_botao_entrada(int linha, int coluna):
 *
 * Definição:
 * Atuação:
 */
uint8_t errorType(uint8_t *ptemp3) { //Seleciona os bits mostradores do tipo de erro que foi
  detectado
    uint8_t auxiliar;
    auxiliar = *ptemp3;
    auxiliar <<= 5;
    return auxiliar >>= 5;
}

/*
 * Função: interpreta_botao_entrada(int linha, int coluna):
 *
 * Definição:
 * Atuação:
 */
uint8_t bitStatus(uint8_t *ptemp1) {
    if (*ptemp1 % 2 == 0) {
        return 0;
```

```
112     } else {
113         return 1; //Parar o programa
114     }
115 }
116
```

```
#ifndef NOKIA5110_H_
#define NOKIA5110_H_

#include "nokia5110_pub.h"

/**************************** Pinagem****************************
// Red SparkFun Nokia 5110 (LCD-10168)
-----------------------------------
Signal        (Nokia 5110) LaunchPad pin
3.3V          (VCC, pin 1) -----> 3v3 power
Ground        (GND, pin 2) -----> GND
SSI0Fss       (SCE, pin 3) -----> PD1
Reset         (RST, pin 4) -----> PD2
Data/Command  (D/C, pin 5) -----> PE0
SSI0Tx        (DN,  pin 6) -----> PD3
SSI0Clk       (SCLK, pin 7) ----> PD0
Backlight     (LED, pin 8) -----> 3v3 power
****************************************************************/

/************************DefiniÃ§Ãµes Gerais**************************/
#define PIN_SCE   7
#define PIN_RESET 6
#define PIN_DC    5
#define PIN_SDIN  4
#define PIN_SCLK  3

#define LCD_COMMAND 0
#define LCD_DATA  1

#define LCD_X     84
#define LCD_Y     48


/************************** Tamanho da tela ***************************/
#define SCREENW     84
#define SCREENH     48

/************************** DefiniÃ§Ãµes de Registradores**************************/
#define DC                      (*((volatile uint32_t *)0x40004100))
#define DC_COMMAND              0
#define DC_DATA                 0x40
#define RESET                   (*((volatile uint32_t *)0x40004200))
#define RESET_LOW               0
#define RESET_HIGH              0x80
#define GPIO_PORTD_DIR_R        (*((volatile uint32_t *)0x40007400))
#define GPIO_PORTD_AFSEL_R      (*((volatile uint32_t *)0x40007420))
#define GPIO_PORTD_DEN_R        (*((volatile uint32_t *)0x4000751C))
#define GPIO_PORTD_AMSEL_R      (*((volatile uint32_t *)0x40007528))
#define GPIO_PORTD_PCTL_R       (*((volatile uint32_t *)0x4000752C))
#define SSI1_CR0_R              (*((volatile uint32_t *)0x40009000))
#define SSI1_CR1_R              (*((volatile uint32_t *)0x40009004))
#define SSI1_DR_R               (*((volatile uint32_t *)0x40009008))
#define SSI1_SR_R               (*((volatile uint32_t *)0x4000900C))
#define SSI1_CPSR_R             (*((volatile uint32_t *)0x40009010))
#define SSI1_CC_R               (*((volatile uint32_t *)0x40009FC8))
#define SSI_CR0_SCR_M           0x0000FF00  // SSI Serial Clock Rate
#define SSI_CR0_SPH             0x00000080  // SSI Serial Clock Phase
```

```
58 #define SSI_CR0_SPO             0x00000040  // SSI Serial Clock Polarity
59 #define SSI_CR0_FRF_M           0x00000030  // SSI Frame Format Select
60 #define SSI_CR0_FRF_MOTO        0x00000000  // Freescale SPI Frame Format
61 #define SSI_CR0_DSS_M           0x0000000F  // SSI Data Size Select
62 #define SSI_CR0_DSS_8           0x00000007  // 8-bit data
63 #define SSI_CR1_MS              0x00000004  // SSI Master/Slave Select
64 #define SSI_CR1_SSE             0x00000002  // SSI Synchronous Serial Port
65 // Enable
66 #define SSI_SR_BSY              0x00000010  // SSI Busy Bit
67 #define SSI_SR_TNF              0x00000002  // SSI Transmit FIFO Not Full
68 #define SSI_CPSR_CPSDVSR_M      0x000000FF  // SSI Clock Prescale Divisor
69 #define SSI_CC_CS_M             0x0000000F  // SSI Baud Clock Source
70 #define SSI_CC_CS_SYSPLL        0x00000000  // Either the system clock (if the
71 // PLL bypass is in effect) or the
72 // PLL output (default)
73 #define SYSCTL_RCGC1_R          (*((volatile uint32_t *)0x400FE104))
74 #define SYSCTL_RCGC2_R          (*((volatile uint32_t *)0x400FE108))
75 #define SYSCTL_RCGC1_SSI0       0x00000010  // SSI0 Clock Gating Control
76 #define SYSCTL_RCGC2_GPIOA      0x00000001  // port A Clock Gating Control
77
78 // ************************** DefiniÃ§Ãµes Nokia5110.h | MORE DEFINES
   **************************
79 #define MAX_X                   84
80 #define MAX_Y                   48
81 #define CONTRAST                0xB1
82
83 // ************************** DefiniÃ§Ãµes PLL | DEFINES PLL **************************
84 #define SYSCTL_RIS_R            (*((volatile uint32_t *)0x400FE050))
85 #define SYSCTL_RIS_PLLLRIS      0x00000040  // PLL Lock Raw Interrupt Status
86 #define SYSCTL_RCC_R            (*((volatile uint32_t *)0x400FE060))
87 #define SYSCTL_RCC_XTAL_M       0x000007C0  // Crystal Value
88 #define SYSCTL_RCC_XTAL_6MHZ    0x000002C0  // 6 MHz Crystal
89 #define SYSCTL_RCC_XTAL_8MHZ    0x00000380  // 8 MHz Crystal
90 #define SYSCTL_RCC_XTAL_16MHZ   0x00000540  // 16 MHz Crystal
91 #define SYSCTL_RCC2_R           (*((volatile uint32_t *)0x400FE070))
92 #define SYSCTL_RCC2_USERCC2     0x80000000  // Use RCC2
93 #define SYSCTL_RCC2_DIV400      0x40000000  // Divide PLL as 400 MHz vs. 200
94 // MHz
95 #define SYSCTL_RCC2_SYSDIV2_M   0x1F800000  // System Clock Divisor 2
96 #define SYSCTL_RCC2_SYSDIV2LSB  0x00400000  // Additional LSB for SYSDIV2
97 #define SYSCTL_RCC2_PWRDN2      0x00002000  // Power-Down PLL 2
98 #define SYSCTL_RCC2_BYPASS2     0x00000800  // PLL Bypass 2
99 #define SYSCTL_RCC2_OSCSRC2_M   0x00000070  // Oscillator Source 2
100 #define SYSCTL_RCC2_OSCSRC2_MO  0x00000000  // MOSC
101
102 #define SYSDIV2 7
103
104 // ************************** EnumeraÃ§Ãµes, Constantes e VariÃ¡veis | ENUMS, CONSTANTS AND
    VARIABLES**************************
105 enum typeOfWrite{
106     COMMAND,                            // the transmission is an LCD command
107     DATA                                // the transmission is data
108 };
109
110 // This table contains the hex values that represent pixels
111 // for a font that is 5 pixels wide and 8 pixels high
112 static const uint8_t ASCII[][5] = {
```

```
113         {0x00, 0x00, 0x00, 0x00, 0x00} // 20
114         ,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
115         ,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
116         ,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #
117         ,{0x24, 0x2a, 0x7f, 0x2a, 0x12} // 24 $
118         ,{0x23, 0x13, 0x08, 0x64, 0x62} // 25 %
119         ,{0x36, 0x49, 0x55, 0x22, 0x50} // 26 &
120         ,{0x00, 0x05, 0x03, 0x00, 0x00} // 27 '
121         ,{0x00, 0x1c, 0x22, 0x41, 0x00} // 28 (
122         ,{0x00, 0x41, 0x22, 0x1c, 0x00} // 29 )
123         ,{0x14, 0x08, 0x3e, 0x08, 0x14} // 2a *
124         ,{0x08, 0x08, 0x3e, 0x08, 0x08} // 2b +
125         ,{0x00, 0x50, 0x30, 0x00, 0x00} // 2c ,
126         ,{0x08, 0x08, 0x08, 0x08, 0x08} // 2d -
127         ,{0x00, 0x60, 0x60, 0x00, 0x00} // 2e .
128         ,{0x20, 0x10, 0x08, 0x04, 0x02} // 2f /
129         ,{0x3e, 0x51, 0x49, 0x45, 0x3e} // 30 0
130         ,{0x00, 0x42, 0x7f, 0x40, 0x00} // 31 1
131         ,{0x42, 0x61, 0x51, 0x49, 0x46} // 32 2
132         ,{0x21, 0x41, 0x45, 0x4b, 0x31} // 33 3
133         ,{0x18, 0x14, 0x12, 0x7f, 0x10} // 34 4
134         ,{0x27, 0x45, 0x45, 0x45, 0x39} // 35 5
135         ,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 36 6
136         ,{0x01, 0x71, 0x09, 0x05, 0x03} // 37 7
137         ,{0x36, 0x49, 0x49, 0x49, 0x36} // 38 8
138         ,{0x06, 0x49, 0x49, 0x29, 0x1e} // 39 9
139         ,{0x00, 0x36, 0x36, 0x00, 0x00} // 3a :
140         ,{0x00, 0x56, 0x36, 0x00, 0x00} // 3b ;
141         ,{0x08, 0x14, 0x22, 0x41, 0x00} // 3c <
142         ,{0x14, 0x14, 0x14, 0x14, 0x14} // 3d =
143         ,{0x00, 0x41, 0x22, 0x14, 0x08} // 3e >
144         ,{0x02, 0x01, 0x51, 0x09, 0x06} // 3f ?
145         ,{0x32, 0x49, 0x79, 0x41, 0x3e} // 40 @
146         ,{0x7e, 0x11, 0x11, 0x11, 0x7e} // 41 A
147         ,{0x7f, 0x49, 0x49, 0x49, 0x36} // 42 B
148         ,{0x3e, 0x41, 0x41, 0x41, 0x22} // 43 C
149         ,{0x7f, 0x41, 0x41, 0x22, 0x1c} // 44 D
150         ,{0x7f, 0x49, 0x49, 0x49, 0x41} // 45 E
151         ,{0x7f, 0x09, 0x09, 0x09, 0x01} // 46 F
152         ,{0x3e, 0x41, 0x49, 0x49, 0x7a} // 47 G
153         ,{0x7f, 0x08, 0x08, 0x08, 0x7f} // 48 H
154         ,{0x00, 0x41, 0x7f, 0x41, 0x00} // 49 I
155         ,{0x20, 0x40, 0x41, 0x3f, 0x01} // 4a J
156         ,{0x7f, 0x08, 0x14, 0x22, 0x41} // 4b K
157         ,{0x7f, 0x40, 0x40, 0x40, 0x40} // 4c L
158         ,{0x7f, 0x02, 0x0c, 0x02, 0x7f} // 4d M
159         ,{0x7f, 0x04, 0x08, 0x10, 0x7f} // 4e N
160         ,{0x3e, 0x41, 0x41, 0x41, 0x3e} // 4f O
161         ,{0x7f, 0x09, 0x09, 0x09, 0x06} // 50 P
162         ,{0x3e, 0x41, 0x51, 0x21, 0x5e} // 51 Q
163         ,{0x7f, 0x09, 0x19, 0x29, 0x46} // 52 R
164         ,{0x46, 0x49, 0x49, 0x49, 0x31} // 53 S
165         ,{0x01, 0x01, 0x7f, 0x01, 0x01} // 54 T
166         ,{0x3f, 0x40, 0x40, 0x40, 0x3f} // 55 U
167         ,{0x1f, 0x20, 0x40, 0x20, 0x1f} // 56 V
168         ,{0x3f, 0x40, 0x38, 0x40, 0x3f} // 57 W
169         ,{0x63, 0x14, 0x08, 0x14, 0x63} // 58 X
```

```
170         ,{0x07, 0x08, 0x70, 0x08, 0x07} // 59 Y
171         ,{0x61, 0x51, 0x49, 0x45, 0x43} // 5a Z
172         ,{0x00, 0x7f, 0x41, 0x41, 0x00} // 5b [
173         ,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c '\'
174         ,{0x00, 0x41, 0x41, 0x7f, 0x00} // 5d ]
175         ,{0x04, 0x02, 0x01, 0x02, 0x04} // 5e ^
176         ,{0x40, 0x40, 0x40, 0x40, 0x40} // 5f _
177         ,{0x00, 0x01, 0x02, 0x04, 0x00} // 60 `
178         ,{0x20, 0x54, 0x54, 0x54, 0x78} // 61 a
179         ,{0x7f, 0x48, 0x44, 0x44, 0x38} // 62 b
180         ,{0x38, 0x44, 0x44, 0x44, 0x20} // 63 c
181         ,{0x38, 0x44, 0x44, 0x48, 0x7f} // 64 d
182         ,{0x38, 0x54, 0x54, 0x54, 0x18} // 65 e
183         ,{0x08, 0x7e, 0x09, 0x01, 0x02} // 66 f
184         ,{0x0c, 0x52, 0x52, 0x52, 0x3e} // 67 g
185         ,{0x7f, 0x08, 0x04, 0x04, 0x78} // 68 h
186         ,{0x00, 0x44, 0x7d, 0x40, 0x00} // 69 i
187         ,{0x20, 0x40, 0x44, 0x3d, 0x00} // 6a j
188         ,{0x7f, 0x10, 0x28, 0x44, 0x00} // 6b k
189         ,{0x00, 0x41, 0x7f, 0x40, 0x00} // 6c l
190         ,{0x7c, 0x04, 0x18, 0x04, 0x78} // 6d m
191         ,{0x7c, 0x08, 0x04, 0x04, 0x78} // 6e n
192         ,{0x38, 0x44, 0x44, 0x44, 0x38} // 6f o
193         ,{0x7c, 0x14, 0x14, 0x14, 0x08} // 70 p
194         ,{0x08, 0x14, 0x14, 0x18, 0x7c} // 71 q
195         ,{0x7c, 0x08, 0x04, 0x04, 0x08} // 72 r
196         ,{0x48, 0x54, 0x54, 0x54, 0x20} // 73 s
197         ,{0x04, 0x3f, 0x44, 0x40, 0x20} // 74 t
198         ,{0x3c, 0x40, 0x40, 0x20, 0x7c} // 75 u
199         ,{0x1c, 0x20, 0x40, 0x20, 0x1c} // 76 v
200         ,{0x3c, 0x40, 0x30, 0x40, 0x3c} // 77 w
201         ,{0x44, 0x28, 0x10, 0x28, 0x44} // 78 x
202         ,{0x0c, 0x50, 0x50, 0x50, 0x3c} // 79 y
203         ,{0x44, 0x64, 0x54, 0x4c, 0x44} // 7a z
204         ,{0x00, 0x08, 0x36, 0x41, 0x00} // 7b {
205         ,{0x00, 0x00, 0x7f, 0x00, 0x00} // 7c |
206         ,{0x00, 0x41, 0x36, 0x08, 0x00} // 7d }
207         ,{0x10, 0x08, 0x08, 0x10, 0x08} // 7e ~
208         //  ,{0x78, 0x46, 0x41, 0x46, 0x78} // 7f DEL
209         ,{0x1f, 0x24, 0x7c, 0x24, 0x1f} // 7f UT sign
210 };
211
212 // ************************** ProtÃ³tipos FunÃ§Ãµes **************************
213 void static lcdwrite(enum typeOfWrite type, uint8_t message);
214 void static lcddatawrite(uint8_t data);
215 void Nokia5110_Init(void);
216 void Nokia5110_OutChar(char data);
217 void Nokia5110_OutString(char *ptr);
218 void Nokia5110_OutUDec(uint16_t n);
219 void Nokia5110_SetCursor(uint8_t newX, uint8_t newY);
220 void Nokia5110_Clear(void);
221 void Nokia5110_DrawFullImage(const uint8_t *ptr);
222 void Nokia5110_PrintBMP(uint8_t xpos, uint8_t ypos, const uint8_t *ptr, uint8_t threshold);
223 void Nokia5110_ClearBuffer(void);
224 void Nokia5110_DisplayBuffer(void);
225 void Nokia5110_ClrPxl(uint32_t i, uint32_t j);
226 void Nokia5110_SetPxl(uint32_t i, uint32_t j);
```

```
227
228 #endif
229
```

```c
#include "nokia5110_priv.h"

/*
 * funções necessárias:
 * lcdwrite
 * Nokia5110_Init
 *
 *
 */


uint8_t Screen[SCREENW*SCREENH/8]; // buffer stores the next image to be printed on the screen
const unsigned char Masks[8]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80}; //usado na funÃ§Ã£o Nokia5110_ClrPxl


void static lcdwrite(enum typeOfWrite type, uint8_t message){
    if(type == COMMAND){
        while(SSIBusy(SSI1_BASE)){}
        GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_0, 0);//GPIO_PORTA_BASE, GPIO_PIN_6, 0
        SSIDataPutNonBlocking(SSI1_BASE, message);
        while(SSIBusy(SSI1_BASE)){};
    } else{
        while((SSI1_SR_R&SSI_SR_TNF)==0){};
        GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_0, 0xFF);
        SSIDataPutNonBlocking(SSI1_BASE, message);
    }
}
void static lcddatawrite(uint8_t data){
    while((SSI1_SR_R&0x00000002)==0){} //SSI0
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_0, 0xFF);//GPIO_PORTA_BASE, GPIO_PIN_6, 0xFF
    SSIDataPutNonBlocking(SSI1_BASE, data);
}

void Nokia5110_Init(void){
    volatile uint32_t delay;

    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI1); //SSI0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);//GPIOA
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);//PARA ATIVAR O D/C
    GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_0); //CONTROLA D/C
    GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_2); //CONTROLA RESET
    GPIOPinConfigure(GPIO_PD0_SSI1CLK); //PA2 SSI0
    GPIOPinConfigure(GPIO_PD1_SSI1FSS); //PA3_SSI0FSS //CHIP SELECT
    GPIOPinConfigure(GPIO_PD3_SSI1TX); //PA5_SSI0TX
    GPIOPinTypeSSI(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_3);//GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_5

    SSIConfigSetExpClk(SSI1_BASE,SysCtlClockGet(),SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 150000, 8);//50000000
    SSIEnable(SSI1_BASE);//SSI0_BASE

    //RESET PARA ZERAR OS REGISTRADORES
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0); //PORTA PIN7
    SysCtlDelay(10); //RESETA O DISPLAY
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0xFF); //PORTA PIN7

```

```
55      lcdwrite(COMMAND, 0x21);              // chip active; horizontal addressing mode (V = 0);
  use extended instruction set (H = 1)
56      lcdwrite(COMMAND, CONTRAST);          // try 0xB1 (for 3.3V red SparkFun), 0xB8 (for 3.3V
  blue SparkFun), 0xBF if your display is too dark, or 0x80 to 0xFF if experimenting
57      lcdwrite(COMMAND, 0x04);              // set temp coefficient
58      lcdwrite(COMMAND, 0x14);              // LCD bias mode 1:48: try 0x13 or 0x14
59
60      lcdwrite(COMMAND, 0x20);              // we must send 0x20 before modifying the display
  control mode
61      lcdwrite(COMMAND, 0x0C);              // set display control to normal mode: 0x0D for
  inverse
62  }
63
64  void Nokia5110_OutChar(char data){
65      int i;
66      lcddatawrite(0x00);
67      for(i=0; i<5; i=i+1){
68          lcddatawrite(ASCII[data - 0x20][i]);
69      }
70      lcddatawrite(0x00);
71  }
72
73  void Nokia5110_OutString(char *ptr){
74      while(*ptr){
75          Nokia5110_OutChar((unsigned char)*ptr);
76          ptr = ptr + 1;
77      }
78  }
79
80  void Nokia5110_OutUDec(uint16_t n){
81      if(n < 10){
82          Nokia5110_OutString("    ");
83          Nokia5110_OutChar(n+'0');
84      } else if(n<100){
85          Nokia5110_OutString("   ");
86          Nokia5110_OutChar(n/10+'0');
87          Nokia5110_OutChar(n%10+'0');
88      } else if(n<1000){
89          Nokia5110_OutString("  ");
90          Nokia5110_OutChar(n/100+'0');
91          n = n%100;
92          Nokia5110_OutChar(n/10+'0');
93          Nokia5110_OutChar(n%10+'0');
94      } else if(n<10000){
95          Nokia5110_OutChar(' ');
96          Nokia5110_OutChar(n/1000+'0');
97          n = n%1000;
98          Nokia5110_OutChar(n/100+'0');
99          n = n%100;
100         Nokia5110_OutChar(n/10+'0');
101         Nokia5110_OutChar(n%10+'0');
102     } else {
103         Nokia5110_OutChar(n/10000+'0');
104         n = n%10000;
105         Nokia5110_OutChar(n/1000+'0');
106         n = n%1000;
107         Nokia5110_OutChar(n/100+'0');
```

```
108         n = n%100;
109         Nokia5110_OutChar(n/10+'0');
110         Nokia5110_OutChar(n%10+'0');
111     }
112 }
113
114 void Nokia5110_SetCursor(uint8_t newX, uint8_t newY){
115     if((newX > 11) || (newY > 5)) return;
116
117     lcdwrite(COMMAND, 0x80|(newX*7));
118     lcdwrite(COMMAND, 0x40|newY);
119 }
120
121 void Nokia5110_Clear(void){
122     int i;
123     for(i=0; i<(MAX_X*MAX_Y/8); i=i+1){
124         lcddatawrite(0x00);
125     }
126     Nokia5110_SetCursor(0, 0);
127 }
128
129 void Nokia5110_DrawFullImage(const uint8_t *ptr){
130     int i;
131     Nokia5110_SetCursor(0, 0);
132     for(i=0; i<(MAX_X*MAX_Y/8); i=i+1){
133         lcddatawrite(ptr[i]);
134     }
135 }
136 uint8_t Screen[SCREENW*SCREENH/8];
137
138 void Nokia5110_PrintBMP(uint8_t xpos, uint8_t ypos, const uint8_t *ptr, uint8_t threshold){
139     int32_t width = ptr[18], height = ptr[22], i, j;
140     uint16_t screenx, screeny;
141     uint8_t mask;
142     // check for clipping
143     if((height <= 0) ||              // bitmap is unexpectedly encoded in top-to-bottom pixel
    order
144             ((width%2) != 0) ||           // must be even number of columns
145             ((xpos + width) > SCREENW) || // right side cut off
146             (ypos < (height - 1)) ||      // top cut off
147             (ypos > SCREENH))           { // bottom cut off
148         return;
149     }
150     if(threshold > 14){
151         threshold = 14;             // only full 'on' turns pixel on
152     }
153     // bitmaps are encoded backwards, so start at the bottom left corner of the image
154     screeny = ypos/8;
155     screenx = xpos + SCREENW*screeny;
156     mask = ypos%8;                // row 0 to 7
157     mask = 0x01<<mask;            // now stores a mask 0x01 to 0x80
158     j = ptr[10];                  // byte 10 contains the offset where image data can be found
159     for(i=1; i<=(width*height/2); i=i+1){
160         // the left pixel is in the upper 4 bits
161         if(((ptr[j]>>4)&0xF) > threshold){
162             Screen[screenx] |= mask;
163         } else{
```

```
164             Screen[screenx] &= ~mask;
165         }
166         screenx = screenx + 1;
167         // the right pixel is in the lower 4 bits
168         if((ptr[j]&0xF) > threshold){
169             Screen[screenx] |= mask;
170         } else{
171             Screen[screenx] &= ~mask;
172         }
173         screenx = screenx + 1;
174         j = j + 1;
175         if((i%(width/2)) == 0){     // at the end of a row
176             if(mask > 0x01){
177                 mask = mask>>1;
178             } else{
179                 mask = 0x80;
180                 screeny = screeny - 1;
181             }
182             screenx = xpos + SCREENW*screeny;
183             // bitmaps are 32-bit word aligned
184             switch((width/2)%4){      // skip any padding
185             case 0: j = j + 0; break;
186             case 1: j = j + 3; break;
187             case 2: j = j + 2; break;
188             case 3: j = j + 1; break;
189             }
190         }
191     }
192 }
193
194 void Nokia5110_ClearBuffer(void){int i;
195     for(i=0; i<SCREENW*SCREENH/8; i=i+1){
196         Screen[i] = 0;              // clear buffer
197     }
198 }
199
200 void Nokia5110_DisplayBuffer(void){
201     Nokia5110_DrawFullImage(Screen);
202 }
203
204
205 void Nokia5110_ClrPxl(uint32_t i, uint32_t j){
206     Screen[84*(i>>3) + j] &= ~Masks[i&0x07];
207 }
208
209 void Nokia5110_SetPxl(uint32_t i, uint32_t j){
210     Screen[84*(i>>3) + j] |= Masks[i&0x07];
211 }
212
```

```c
#include "nokia5110_pub.h"

void inicializa_display(void){
    Nokia5110_Init();
    SysCtlDelay(1000);
    Nokia5110_Clear();
    Nokia5110_ClearBuffer();

    Nokia5110_SetCursor(0,0); //Coluna e linha
    Nokia5110_OutString("T TC:");

    Nokia5110_SetCursor(0,1);
    Nokia5110_OutString("SP T:");

    Nokia5110_SetCursor(0,2);
    Nokia5110_OutString("D Bico:");

    Nokia5110_SetCursor(0,3);
    Nokia5110_OutString("D Res:");

    Nokia5110_SetCursor(0,4);
    Nokia5110_OutString("Status:");

    Nokia5110_SetCursor(0,5);
    Nokia5110_OutString("M Oper:");

}

void displayTemperatureUpdate(uint8_t var)
{
    Nokia5110_SetCursor(7, 0);
    Nokia5110_OutUDec(var);
}

void displaySetpointUpdate(uint8_t var)
{
    Nokia5110_SetCursor(7, 1); //Verificar essa posição da coluna;
    Nokia5110_OutUDec(var);
}

void displayDutyNozzleUpdate(uint8_t var)
{
    Nokia5110_SetCursor(7, 2); //Verificar essa posição da coluna;
    Nokia5110_OutUDec(var);
}

void displayDutyRelayUpdate(uint8_t var)
{
    Nokia5110_SetCursor(7, 3); //Verificar essa posição da coluna;
    Nokia5110_OutUDec(var);
}

void displayStatusUpdate(uint8_t var)
{
    if(var == 1)
    {
        Nokia5110_SetCursor(7, 4); //Verificar essa posição da coluna;
```

```
58          Nokia5110_OutString("ErrTC");
59      }
60      else
61      {
62          Nokia5110_SetCursor(7, 4); //Verificar essa posição da coluna;
63          Nokia5110_OutString("   0");
64      }
65  }
66
67  void displayOperModeUpdate(uint8_t var)
68  {
69      Nokia5110_SetCursor(7, 5); //Verificar essa posição da coluna;
70      Nokia5110_OutUDec(var);
71  }
72
73
74
75
```

```c
/*
 * operation.h
 *
 *  Created on: 02/11/2015
 *      Author: Allan
 */

#ifndef OPERATION_H_
#define OPERATION_H_

#include "includes.h"
#include "UART.h"
#include "nokia5110_pub.h"
#include "protocol.h"
#include "TIMER.h"
#include "MAX31855.h"

#define TEMP_RES_ATIVADA 10 //Indica quanto acima da temperatura ambiente o sistema deve reconhecer que a resistência está funcionando

//Definições do modo de operação 1
#define SP_TEM_MOD_1 165
#define PWM_PUMP_MOD_1 13
#define PWM_NOZZLE_MOD_1 13

//Definições do modo de operação 2
#define SP_TEM_MOD_2 190
#define PWM_PUMP_MOD_2 20
#define PWM_NOZZLE_MOD_2 20
#define RUNNING_TIME 10

//Limite inferior em que se detecta que a resistência do bico aquece de maneira satisfatória
#define HEAT_INF_LIM 15
#define NOZZLE_INF_LIM 10
#define PUMP_INF_LIM 10

#define ValMaxC 1

#define CTCmax ValMaxC
#define CSRmax ValMaxC
#define CErTmax ValMaxC
#define CDuRmax ValMaxC
#define CDuPmax ValMaxC
#define CDuNmax ValMaxC
#define CTSPmax ValMaxC
#define CSSmax ValMaxC
#define COpMmax ValMaxC
#define CSTempMmax ValMaxC
#define CSTempDecMmax ValMaxC
#define CTCDecmax ValMaxC

#define CLEmax 1

void operationVarUpdate(uint8_t UpSp, uint8_t value);
void CommunicationUpdate();
uint8_t ChangeStatus();
void PID(uint8_t temperatura_atual, uint8_t setpoint);
```

```
57 void UpdateMax();
58 void FnOperationMode(uint8_t mode);
59 void delay(uint16_t ms);
60 void labviewComReq();
61
62 #endif /* OPERATION_H_ */
63
```

```c
/*
 * operation.c
 *
 *  Created on: 02/11/2015
 *      Author: Allan
 */

#include "operation.h"

/*
 * Aqui pode ser colocado todas as variáveis do sistema e todas as funções podem reportar para as funções aqui presentes para atualizar as variáveis
 * do projeto. Assim todas as variáveis podem ser consultadas e enviadas para a comunicação com o Labview.
 */

//Variáveis do projeto
//MAX31855
uint8_t mainFrame[4];
uint8_t TemperatureTC=0;
uint8_t TemperatureTCDec=0;
uint8_t TemperatureSR=0;
uint8_t TemperatureSRDec=0;
uint8_t errorTypeReport=0;
uint8_t ifError=0;
uint8_t TemperatureSRdec = 0;
uint8_t CSDecTemp = 0;
//MAX31855 OLD
uint8_t TemperatureTC_OLD=0;
uint8_t TemperatureSR_OLD=0;
uint8_t TemperatureSRDec_OLD=0;
uint8_t errorTypeReport_OLD=0;
uint8_t ifError_OLD=0;
uint8_t TemperatureSRdec_OLD = 0;
uint8_t CSDecTemp_OLD = 0;
uint8_t TemperatureTCDec_OLD=0;

//Duty-Cycles
uint8_t DutyCycleRelay=0;
uint8_t DutyCyclePump=0;
uint8_t DutyCyclenozzle=0;
//Duty-Cycles_OLD
uint8_t DutyCycleRelay_OLD=0;
uint8_t DutyCyclePump_OLD=0;
uint8_t DutyCyclenozzle_OLD=0;

//Operation
uint8_t OperationMode=0;
uint8_t TemperatureSP=0;
uint8_t SystemStatus=0b00000000;
//Operation_OLD
uint8_t OperationMode_OLD=0;
uint8_t TemperatureSP_OLD=0;
uint8_t SystemStatus_OLD=0b00000000;

//PID
uint32_t satMax = 65530, satMin = 50;
```

```
56 float Kp = 1.1507, Ti = 0.035982;
57 float error = 0, integral = 0, error_OLD=0;
58 float PIDValue = 0;
59 uint8_t usePID = 0;
60 uint8_t usePID_OLD = 0;
61 float integral_OLD=0;
62
63 //Outras Variáveis
64 uint32_t loadNom = 100;//uint32_t loadNom = 62499;
65 uint8_t CLE = 0;
66 uint8_t flagStatus=0;
67 uint8_t flagComm_=0;
68
69 /*
70  * Para atualizar essa função repassar:
71  * (SPtemp, 1)
72  * (SPrelay, 2)
73  * (SPnozzle, 3)
74  * (SPpump, 4)
75  */
76 void operationVarUpdate(uint8_t UpSp, uint8_t value) //Essa função atualiza os valores sempre
   que um novo
77 {                                                                                    //
   dado é recebido
78     switch(value)
79     {
80     case 1:
81         TemperatureSP = UpSp;
82         break;
83     case 2:
84         DutyCycleRelay = UpSp;
85         break;
86     case 3:
87         DutyCyclenozzle = UpSp;
88         break;
89     case 4:
90         DutyCyclePump = UpSp;
91         break;
92     case 5:
93         usePID = UpSp;
94         break;
95     case 6:
96         OperationMode = UpSp;
97         //FnOperationMode(OperationMode);
98         break;
99     }
100 }
101
102 uint8_t CTC=0;
103 uint8_t CSR=0;
104 uint8_t CErT=0;
105 uint8_t CDuR=0;
106 uint8_t CDuP=0;
107 uint8_t CDuN=0;
108 uint8_t CTSP=0;
109 uint8_t CSS=0;
110 uint8_t COpM=0;
```

```
111 uint8_t CSTemp=0;
112 uint8_t CSTempDec=0;
113 uint8_t CTCDec = 0;
114
115
116 void CommunicationUpdate()
117 {
118     //Variáveis de temperatura e estado do sensor
119     if(TemperatureTC != TemperatureTC_OLD && CTC <= CTCmax)
120     {
121          TemperatureTC_OLD = TemperatureTC;
122          displayTemperatureUpdate(TemperatureTC);
123          sendDataFrame(2, TemperatureTC);
124          CTC++;
125          return;
126     }
127     if(DutyCycleRelay != DutyCycleRelay_OLD && CDuR <= CDuRmax)
128     {
129         DutyCycleRelay_OLD = DutyCycleRelay;
130          displayDutyRelayUpdate(DutyCycleRelay);
131          sendDataFrame(4, DutyCycleRelay);
132          CDuR++;
133          return;
134     }
135     if(errorTypeReport != errorTypeReport_OLD && CErT <= CErTmax)
136     {
137         errorTypeReport_OLD = errorTypeReport;
138         sendDataFrame(7, errorTypeReport);
139         CErT++;
140         return;
141     }
142     //Variáveis PWM
143     if(DutyCyclePump != DutyCyclePump_OLD  && CDuP <= CDuPmax)
144     {
145         DutyCyclePump_OLD = DutyCyclePump;
146         sendDataFrame(5, DutyCyclePump);
147         CDuP++;
148         return;
149     }
150     if(DutyCyclenozzle != DutyCyclenozzle_OLD && CDuN <= CDuNmax)
151     {
152         DutyCyclenozzle_OLD = DutyCyclenozzle;
153         displayDutyNozzleUpdate(DutyCyclenozzle);
154         sendDataFrame(6, DutyCyclenozzle);
155         CDuN++;
156         return;
157     }
158     if(TemperatureSP != TemperatureSP_OLD && CTSP <= CTSPmax)
159     {
160         TemperatureSP_OLD = TemperatureSP;
161         displaySetpointUpdate(TemperatureSP);
162         sendDataFrame(10, TemperatureSP);
163         CTSP++;
164         return;
165     }
166     if(SystemStatus != SystemStatus_OLD && CSS <= CSSmax)
167     {
```

```
168         SystemStatus_OLD = SystemStatus;
169         sendDataFrame(8, SystemStatus);
170 
171         if(SystemStatus>>7 == 1)
172             displayStatusUpdate(1);
173         else
174             displayStatusUpdate(0);
175         CSS++;
176         return;
177     }
178     if(OperationMode != OperationMode_OLD && COpM <= COpMmax)
179     {
180         OperationMode_OLD = OperationMode;
181         displayOperModeUpdate(OperationMode);
182         sendDataFrame(9, OperationMode);//TODO: Ver qual valor deve ser atualizado em labview
183         COpM++;
184         return;
185     }
186     /*if(TemperatureSR != TemperatureSR_OLD && CSTemp <= CSTempMmax)
187     {
188         TemperatureSR_OLD = TemperatureSR;
189         //displayOperModeUpdate(TemperatureSR);
190         sendDataFrame(3, TemperatureSR);//TODO: Ver qual valor deve ser atualizado em labview
191         CSTemp++;
192         return;
193     }*/
194     /*if(TemperatureSRdec != TemperatureSRdec_OLD && CSTempDec <= CSTempDecMmax)
195     {
196         TemperatureSRdec_OLD = TemperatureSRdec;
197         //displayOperModeUpdate(TemperatureSR);
198         sendDataFrame(11, TemperatureSRdec);//TODO: Ver qual valor deve ser atualizado em
    labview
199         CSTempDec++;
200         return;
201     }*/
202     /*if(TemperatureTCDec != TemperatureTCDec_OLD && CTCDec <= CTCDecmax)
203     {
204         TemperatureTCDec_OLD = TemperatureTCDec;
205         //displayOperModeUpdate(TemperatureSR);
206         sendDataFrame(12, TemperatureTCDec);//TODO: Ver qual valor deve ser atualizado em
    labview
207         CSTemp++;
208         return
209     }*/
210     CTC=0;
211     CErT=0;
212     CDuR=0;
213     CDuP=0;
214     CDuN=0;
215     CTSP=0;
216     CSS=0;
217     COpM=0;
218     //CSTemp=0;
219     //CSTempDec=0;
220     //CTCDec=0;
221 
222     flagComm_++;
```

```
223     if(flagComm_ >10)
224     {
225         sendDataFrame(8, SystemStatus);
226         flagComm_=0;
227     }
228 }
229
230 void labviewComReq()
231 {
232     sendDataFrame(8, SystemStatus);
233     delay(75);
234     sendDataFrame(6, DutyCyclenozzle);
235     delay(75);
236     sendDataFrame(5, DutyCyclePump);
237     delay(75);
238     sendDataFrame(7, errorTypeReport);
239     delay(75);
240     sendDataFrame(9, OperationMode);
241     delay(75);
242     sendDataFrame(2, TemperatureTC);
243     delay(75);
244     sendDataFrame(4, DutyCycleRelay);
245     delay(75);
246     /*sendDataFrame(3, TemperatureSR);
247     delay(100);
248     sendDataFrame(11, TemperatureSRdec);
249     delay(100);
250     sendDataFrame(12, TemperatureTCDec);
251     delay(100);*/
252 }
253
254
255 uint8_t ChangeStatus()
256 {
257     SystemStatus = 0;
258
259     SystemStatus |= ifError<<7; //Problema na leitura do termopar
260
261     if(OperationMode == 1)
262         SystemStatus |= 1<<6; //Modo de operação do 1 sistema
263     else if(OperationMode == 2)
264         SystemStatus |= 1<<5; //Modo de operação do 2 sistema
265     else
266         SystemStatus |= 0<<5;
267         SystemStatus |= 0<<6;
268
269     if(!usePID)
270         SystemStatus |= 1<<4; //Valores são forçados, sem controle de temperatura
271
272     //Verifica a temperatura ambiente e a temperatura da resistência
273     if((TemperatureTC>TemperatureSR+TEMP_RES_ATIVADA) && DutyCycleRelay>HEAT_INF_LIM)
274         SystemStatus |= 1<<3; //Verifica se a resistência está aquecendo
275
276     if(DutyCyclenozzle > NOZZLE_INF_LIM)
277         SystemStatus |= 1<<2; //Retorna se o bico está ativado
278
279     if(DutyCyclePump > PUMP_INF_LIM)
```

```
280         SystemStatus |= 1<<1; //Retorna se a bomba está ativada
281 
282     SystemStatus |= !flagStatus; //Processamento ativado
283     return SystemStatus;
284 }
285 
286 void PID(uint8_t temperatura_atual, uint8_t setpoint)
287 {
288     if(OperationMode != OperationMode_OLD)
289     {
290         error_OLD = 0;
291         integral = 0;
292         integral_OLD = 0;
293         PIDValue = 0;
294         OperationMode_OLD = OperationMode;
295     }
296     uint16_t aux;
297     error = setpoint - temperatura_atual;
298 
299     integral = integral_OLD+Ti*error;
300 
301     integral_OLD = integral;
302     error_OLD = error;
303     PIDValue = Kp*error + integral;// + Kd*deriv;
304 
305     if(PIDValue > 99)
306         PIDValue = 100;
307 
308     aux = PIDValue;
309 
310     SPRelayAjust(aux);
311 }
312 void UpdateMax()
313 {
314     getDataMAX(mainFrame);
315     ifError = bitStatus(&mainFrame[1]);
316     TemperatureTC = TCtemp(&mainFrame[0], &mainFrame[1]);
317     //TemperatureTCDec = TCtempDec(&mainFrame[1]);
318     errorTypeReport = errorType(&mainFrame[3]);
319     //TemperatureSR = sensorTemp(&mainFrame[2]);
320     //TemperatureSRdec = sensorTempDec(&mainFrame[3]);
321     ChangeStatus();
322     //CommunicationUpdate();
323 
324     ifError=0;
325     if(ifError == 1)
326     {
327         SPRelayAjust(0);
328         SPnozzleAjust(0);
329         SPPumpAjust(0);
330     }
331     else
332     {
333         if(usePID)
334             {
335                 PID(TemperatureTC, TemperatureSP);
336             }
```

```
337     }
338 }
339 
340 //Definição dos modos de operação
341 void FnOperationMode(uint8_t mode)
342 {
343     OperationMode = mode;
344     while(1){
345     if(OperationMode == 1)
346     {
347         TemperatureSP = SP_TEM_MOD_1;
348         DutyCyclePump=0;
349         DutyCyclenozzle=0;
350         DutyCycleRelay=0;
351         usePID=1;
352         SPnozzleAjust(DutyCyclenozzle);
353         SPPumpAjust(DutyCyclePump);
354 
355         while((TemperatureTC != SP_TEM_MOD_1) || (OperationMode == 1))
356         {
357 
358         }
359 
360         if(OperationMode == 1)
361         {
362             DutyCyclePump=PWM_PUMP_MOD_1;
363             DutyCyclenozzle=PWM_NOZZLE_MOD_1;
364             SPnozzleAjust(DutyCyclenozzle);
365             SPPumpAjust(DutyCyclePump);
366 
367             while(OperationMode == 1)
368             {
369 
370             }
371         }
372     }
373     else if(OperationMode == 2)
374     {
375         TemperatureSP = SP_TEM_MOD_2;
376         DutyCyclePump=0;
377         DutyCyclenozzle=0;
378         DutyCycleRelay=0;
379         SPnozzleAjust(DutyCyclenozzle);
380         SPPumpAjust(DutyCyclePump);
381         usePID=1;
382 
383         while((TemperatureTC != SP_TEM_MOD_2) || (OperationMode == 2))
384         {
385         }
386 
387         if(OperationMode == 2)
388         {
389             DutyCyclePump=PWM_PUMP_MOD_2;
390             DutyCyclenozzle=PWM_NOZZLE_MOD_2;
391             SPnozzleAjust(DutyCyclenozzle);
392             SPPumpAjust(DutyCyclePump);
393             reCount();
```

```
394
395             while(timeNow() <= RUNNING_TIME)
396             {
397             }
398             OperationMode = 3;
399         }
400     }
401     }
402
403 }
404
405 void delay(uint16_t ms) {
406     uint32_t delay = SysCtlClockGet()/3000;
407     delay*=ms;
408     SysCtlDelay(delay); // SysCtlDelay() leva 3 ciclos para um atraso
409 }
410
```

```c
/*
 * protocol.h
 *
 *  Created on: 28/10/2015
 *      Author: Allan
 */

#ifndef PROTOCOL_H_
#define PROTOCOL_H_
#include "includes.h"
#include "UART.h"
#include "operation.h"

enum
{
    OPERATION_MODE=1, // 1  Escolhe modo de operação
    TURN_PERIPH_OFF,// 2        Aciona relé, bomba e bico
    TEMP_AJUST_SETPOINT,// 3    Ajusta (Aumenta e diminui) setpoint de temperatura
    RELAY_AJUST_PWM, // 4 Ajusta PWM do relé
    PUMP_AJUST_PWM,//5  Ajusta PWM da bomba
    nozzle_AJUST_PWM,// 6   Ajusta PWM dO BICO
    UPDATE_COMMUNICATION,// 7   Ajusta PWM dO BICO
};

enum
{
  LER_STX = 0,
  LER_COMMANDMANAGER,
  LER_dataReceived,
  LER_CRC,
  LER_ETX,
};

#define STX 0x02 //Enviar 2 pelo terminal corresponde a 0x32
#define ETX 0x03

uint8_t readDataFrame(uint8_t dado);
void sendDataFrame(uint8_t command ,uint16_t data);
void managesData (void);

void OperModeAjust(uint16_t modeOper);
void SPTempAjust(uint16_t newSP);
void SPRelayAjust(uint16_t newSP);
void SPnozzleAjust(uint16_t newSP);
void SPPumpAjust(uint16_t newSP);

#endif /* PROTOCOL_H_ */

```

```c
#include "protocol.h"
/*
 * protocol.c
 *
 *  Created on: 28/10/2015
 *      Author: Allan
 */


uint8_t dataReceived[7];
uint8_t crc=0, crcTX=0;
uint8_t protocolState=0, sendingState=0;
uint8_t commandManager=0;
uint8_t flagComm=0;

volatile uint32_t Load = 12499; //Variavel usada como valor de contagem para o modulo PWM
volatile uint32_t Load_Relay = 62499; //Variavel usada como valor de contagem para o modulo PWM

uint8_t tempSP = 0, nozzleSP = 0, relaySP=0, pumpSP=0;


/*
 * Cada novo caracter recebido através da uart é avaliado pela função TrataSerial. O switch é auto alimentado e cada commandManager redireciona para
 * o próximo estado de recebimento de dados. A estrutura do frame é:
 * 1 byte: FSB: Dá início à sequência. Está definido como 0x02
 * 2 byte: Dividido em duas partes: Os 3 primeiros bits dizem quantos bytes serão recebidos e os 5 últimos dizem quais commandManagers devem ser executados
 * Bytes de dados.............
 * 3 byte: CRC
 * 4 byte: EDX: Finaliza a transmissão
 */
uint8_t readDataFrame(uint8_t dado)
{
    uint8_t status = OK;
  static uint8_t dataLength, dataIndex;

  switch(protocolState)
  {
    case LER_STX:
      if (dado == STX)
      {
        crc = dado;
        protocolState = LER_COMMANDMANAGER;
        UARTCharPutNonBlocking(UART0_BASE,1);
      }
      else
          status = FAIL;
    break;

    case LER_COMMANDMANAGER:
      crc += dado;
      commandManager = dado>>3;
      dataLength = dado&0x07;
      dataIndex = 0;
```

```
55        protocolState = (dataLength == 0)?LER_CRC:LER_dataReceived;
56      break;
57
58      case LER_dataReceived:
59        --dataLength;
60        crc = crc+dado;
61        dataReceived[dataIndex] = dado;
62        ++dataIndex;
63        if (dataLength == 0)
64          protocolState = LER_CRC;
65      break;
66
67      case LER_CRC:
68          crc = ~crc;
69        if(crc != dado)
70        {
71            status = FAIL;
72          protocolState = LER_STX;
73        }
74        else
75          protocolState = LER_ETX;
76      break;
77
78      case LER_ETX:
79        if ( dado == ETX )
80        {
81          managesData();
82        }
83        else
84            status = FAIL;
85        protocolState = LER_STX;
86      break;
87    }
88    return (status);
89  }
90
91  void sendDataFrame(uint8_t command ,uint16_t data)
92  {
93      uint8_t data1 = 0, data2 = 0, bytes = 0;
94
95          UARTSendByte(STX); //Faz o envio do byte de identificação
96
97          data1 = (data<<8)>>8;
98          data2 = data>>8;
99          if(data2 >= 1)
100             bytes = 2;
101         else if(data1 >= 1)
102             bytes = 1;
103         else
104             bytes = 0;
105
106         bytes = 2;
107         command <<= 3;
108         command |= bytes;
109         UARTSendByte(command);
110
111         if(bytes == 2)
```

```
112         {
113         UARTSendByte(data2);
114         UARTSendByte(data1);
115         }
116         else if(bytes == 1)
117             UARTSendByte(data1);
118         else
119         {}
120 
121         crcTX = STX+command+data1+data2;
122         UARTSendByte(crcTX);
123 
124         UARTSendByte(ETX); //Faz o envio do byte de identificação
125 }
126 
127 void managesData (void)
128 {
129     uint16_t value=0;
130     value = (dataReceived[1]<<8) | dataReceived[0];
131 
132   switch (commandManager)
133   {
134     case TEMP_AJUST_SETPOINT:
135       SPTempAjust(value);
136       //Enviar informação de SP ajustado?
137     break;
138 
139     case OPERATION_MODE:
140       OperModeAjust(value);
141     break;
142 
143     case PUMP_AJUST_PWM:
144           SPPumpAjust(value);
145     break;
146 
147     case nozzle_AJUST_PWM:
148         SPnozzleAjust(value);
149     break;
150 
151     case RELAY_AJUST_PWM:
152         operationVarUpdate(0, 5);
153         SPRelayAjust(value);
154     break;
155 
156     case UPDATE_COMMUNICATION:
157         labviewComReq();
158     break;
159 
160   }
161 }
162 /*
163  * Para ajustar o SP da temperatura:
164  * Um novo valor deve ser passado
165  * o PWM deve ser ajustado através de um controlador PI PID
166  */
167 void OperModeAjust(uint16_t modeOper)
168 {
```

```
169     operationVarUpdate(modeOper, 6);
170 }
171
172 void SPTempAjust(uint16_t newSP)
173 {
174     tempSP = newSP;
175     operationVarUpdate(tempSP, 1);
176     operationVarUpdate(1, 5);
177 }
178
179 void SPRelayAjust(uint16_t newSP)
180 {
181     relaySP = newSP;
182     if(newSP <= 0)
183         newSP = 25;
184     else if(newSP >= 100)
185         newSP = 65535;
186     else
187             newSP *= 655;
188
189     //newSP = 65535-newSP;
190     PWMPulseWidthSet(PWM1_BASE, PWM_OUT_3, newSP*Load_Relay/65535); //PA7 RELAY
191     operationVarUpdate(relaySP, 2);
192 }
193
194
195 void SPnozzleAjust(uint16_t newSP)
196 {
197     nozzleSP = newSP;
198     if(newSP < 1)
199         newSP = 1;
200     else if(newSP > 99)
201         newSP = 99;
202     newSP *= 655;
203     if(newSP > 65535)
204         newSP = 65535;
205
206     newSP = 65535-newSP;
207     PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, newSP * Load / 65535); //seta a largura do pulso do
    PWM0 (CH1) PE4 ui8AdjustFuel BICO?
208     operationVarUpdate(nozzleSP, 3);
209 }
210
211 void SPPumpAjust(uint16_t newSP)
212 {
213     pumpSP = newSP;
214     if(newSP < 1)
215         newSP = 1;
216     else if(newSP > 99)
217         newSP = 99;
218     newSP *= 655;
219     if(newSP > 65535)
220         newSP = 65535;
221
222     newSP = 65535-newSP;
223     PWMPulseWidthSet(PWM0_BASE, PWM_OUT_5, newSP * Load / 65535); //seta a largura do pulso
    para BOMBA - PE5
```

```
224     operationVarUpdate(pumpSP, 4);
225 }
226
```

```
#define LED_1 GPIO_PIN_1
#define LED_2 GPIO_PIN_2
#define LED_3 GPIO_PIN_3

#define SW1 GPIO_PIN_4
#define SW2 GPIO_PIN_0

#define USER 0 //tipo inicial de aparencia do display

#define TIMER3_VALUE                    (*((volatile uint32_t *)0x40033050))

#define TIMER5_VALUE                    (*((volatile uint32_t *)0x40035050))


#define VMED_MAX_COUNTER    1000    //contador para o qual a vel. med sera zerada
#define VINST_MAX_COUNTER   100     //contador para o qual a vel. inst sera zerada

#define N_AMOSTRAS_AVG 5            //numero de amostras da media para velocidade instantanea

#define PI_2_RAIO 1 //2*pi*raio (em metros)

#define POWER_UP_EL 3
#define POWER_DOWN_EL 4
#define POWER_ON_EL 1
#define POWER_OFF_EL 2

//Define as Unidades que serão colocadas no display
#define U_TEMP 1
#define U_D 2

#define U_RPM 1
#define U_RPS 2
#define U_MPS 3
#define U_KPH 4

#define VMED_TYPE   2
#define VINST_TYPE  1

#define SET     1
#define WRITE   1
#define READ    0
#define UP      2
#define ZSET    3

#define LINHA_1 0
#define LINHA_2 1
#define LINHA_3 2
#define LINHA_4 3

#define COLUNA_1 0
#define COLUNA_2 1
#define COLUNA_3 2
#define COLUNA_4 3

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
```

```
58 #include <stdbool.h>
59
60 #include "inc/tm4c123gh6pm.h"
61 #include "inc/hw_memmap.h"
62 #include "inc/hw_timer.h"
63 #include "inc/hw_types.h"
64 #include "inc/hw_gpio.h"
65 #include "inc/hw_ssi.h"
66 #include "driverlib/sysctl.h"
67 #include "driverlib/interrupt.h"
68 #include "driverlib/gpio.h"
69 #include "driverlib/timer.h"
70 #include "driverlib/uart.h"
71 #include "driverlib/pin_map.h"
72 #include "driverlib/ssi.h"
73 #include "driverlib/pwm.h"
74
```

```
 1 /*
 2  * PWM.h
 3  *
 4  *  Created on: 27/10/2015
 5  *      Author: Allan
 6  */
 7
 8 #ifndef PWM_H_
 9 #define PWM_H_
10
11 #include "includes.h"
12
13 #define GPIO_PA6_M1PWM2         0x00001805
14 #define GPIO_PA7_M1PWM3         0x00001C05
15 #define GPIO_PE5_M0PWM5         0x00041404
16 #define GPIO_PE4_M0PWM4         0x00041004
17
18
19 void PWMInit();
20 void ajustaValores();
21 void excludePWM();
22
23
24
25 #endif /* PWM_H_ */
26
```

```c
#include "PWM.h"

uint16_t ui8AdjustRelay = 25000;
uint16_t ui8AdjustMosfet = 25000;
uint16_t ui8AdjustPump = 25000;
uint16_t ui8AdjustFuel = 25000;

volatile uint32_t PWM_FREQUENCY = 50;
volatile uint32_t PWM_FREQUENCY_Relay = 10;

volatile uint32_t CLOCK = 40000000;
volatile uint32_t ui32Load; //Variavel usada como valor de contagem para o modulo PWM
volatile uint32_t ui32PWMClock; //Clock do PWM
volatile uint32_t ui32Load_Relay,CLOCKPWM; //Variavel usada como valor de contagem para o modulo PWM

void ajustaValores()
{
    //configurações PWM
    ui32PWMClock = CLOCK ; //antes CLOCK/64
    ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1; //ui32Load é a variavel que vai ser usada como valor de contagem. Determina a contagem que será feita dentro do Load register do modulo PWM (frequencia do PWM/frequencia desejada). Além disso, subtrai 1 a cada ciclo até chegar em 0.
    ui32Load_Relay = (ui32PWMClock / PWM_FREQUENCY_Relay) - 1;
}

void PWMInit()
{

    SysCtlPWMClockSet(SYSCTL_PWMDIV_64); //seta a frequencia do PWM para 625kHz (=40MHz/64)

    CLOCK=SysCtlClockGet(); //Essa função apresenta alguns problemas
    CLOCK = SysCtlPWMClockGet();
    CLOCK = 625000; // Com CLOCK = 625000 e sysdiv_5 o PWM de 50Hz funciona
                    //Foi feito teste com CLOCK = 1875000 e sysdiv_1, mas não funcionou


    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); //habilita o módulo PWM1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); //habilita o módulo PWM1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); //habilita o módulo GPIOD (para as saídas PWM dos pinos do portal D)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //habilita o módulo GPIOA (para as saídas PWM dos pinos do portal A)

    GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_6); //configura o pino 6 do portal A como pino PWM de saída
    GPIOPinConfigure(GPIO_PA6_M1PWM2); //configura o pino 6 do portal A para trabalhar no módulo "PWM generator 1" do PWM1

    GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_7); //configura o pino 7 do portal A como pino PWM de saída
    GPIOPinConfigure(GPIO_PA7_M1PWM3); //configura o pino 7 do portal A para trabalhar no módulo "PWM generator 1" do PWM1

    GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_5); //configura o pino 1 do portal F como pino PWM de saída
```

```
47      GPIOPinConfigure(GPIO_PE5_M0PWM5); //configura o pino 1 do portal F para trabalhar no
  módulo "PWM generator 2" do PWM1
48
49      GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_4); //configura o pino 2 do portal F como pino PWM
  de saída
50      GPIOPinConfigure(GPIO_PE4_M0PWM4); //configura o pino 2 do portal F para trabalhar no
  módulo "PWM generator 3" do PWM1
51
52      ajustaValores();
53
54      PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN); //Configura o modulo 1 do PWM
  para o gerador 0 como um contador decrescente
55      PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, ui32Load); //carrega o valor de contagem
56
57      PWMGenConfigure(PWM1_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN); //Configura o modulo 1 do PWM
  para o gerador 1 como um contador decrescente
58      PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, ui32Load_Relay); //carrega o valor de contagem
59
60      PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, ui8AdjustFuel * ui32Load / 65535); //seta a largura
  do pulso do PWM0 (CH1)
61      PWMPulseWidthSet(PWM0_BASE, PWM_OUT_5, ui8AdjustPump * ui32Load / 65535); //seta a largura
  do pulso do PWM1 (CH2)
62      PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, ui8AdjustMosfet * ui32Load_Relay / 65535); //seta a
  largura do pulso do PWM2 (CH3)
63      PWMPulseWidthSet(PWM1_BASE, PWM_OUT_3, ui8AdjustRelay * ui32Load_Relay / 65535); //seta a
  largura do pulso do PWM3 (CH4)
64      //quando a seguinte divisão acontece: ui32Load/65535, isso determina a resolução minima do
  sinal. Quando ui8Adjust é multiplicado o valor é ajustado para a posição desejada. Os valor
  65536 pode ser mudado caso queira aumentar ou diminuir a resolução.
65
66      PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT, true); //modulo 1 do PWM 0 é habilitado como saida
67      PWMOutputState(PWM0_BASE, PWM_OUT_5_BIT, true); //modulo 1 do PWM 1 é habilitado como saida
68      PWMOutputState(PWM1_BASE, PWM_OUT_2_BIT, true); //modulo 1 do PWM 2 é habilitado como saida
69      PWMOutputState(PWM1_BASE, PWM_OUT_3_BIT, true); //modulo 1 do PWM 3 é habilitado como saida
70
71      PWMGenEnable(PWM0_BASE, PWM_GEN_2); //modulo 1 do PWM com gerador 0 é habilitado para rodar
72      PWMGenEnable(PWM1_BASE, PWM_GEN_1); //modulo 1 do PWM com gerador 1 é habilitado para rodar
73  }
74
75  void excludePWM()
76  {
77      PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, 0); //seta a largura do pulso do PWM0 (CH1)
78      PWMPulseWidthSet(PWM0_BASE, PWM_OUT_5, 0); //seta a largura do pulso do PWM1 (CH2)
79      PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, 0); //seta a largura do pulso do PWM2 (CH3)
80      PWMPulseWidthSet(PWM1_BASE, PWM_OUT_3, 0); //seta a largura do pulso do PWM3 (CH4)
81      PWMGenDisable(PWM0_BASE, PWM_GEN_2);
82      PWMGenDisable(PWM1_BASE, PWM_GEN_1);
83  }
84
```

```
 1 #ifndef TIMER_H_
 2 #define TIMER_H_
 3
 4 #include "includes.h"
 5
 6 void timer0Init();
 7 void timer1Init();
 8 void TimeCounter();
 9 uint32_t timeNow();
10 void reCount();
11
12 #endif /* TIMER_H_ */
13
```

```c
#include "TIMER.h"

uint32_t seconds=0;


/*
 * Timer 0A é utilizado para contar o tempo
 * Timer 0B é utilziado para atualizar a leitura da temperatura.
 */
void timer0Init()
{
    uint32_t ui32Period;
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
        TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);

        ui32Period = (SysCtlClockGet() / 1);
        TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);


        IntEnable(INT_TIMER0A);

        TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

        IntMasterEnable();

        TimerEnable(TIMER0_BASE, TIMER_A);

}

/*
 * Timer 1A é utilizado para atualizar a comunicação
 * Timer 1B é utilizado para
 */
void timer1Init()
{
    uint32_t ui32Period;
        //SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        //GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
        TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);

        ui32Period = (SysCtlClockGet() / 10);
        TimerLoadSet(TIMER1_BASE, TIMER_A, ui32Period -1);

        IntEnable(INT_TIMER1A);
        TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
        IntMasterEnable();

        TimerEnable(TIMER1_BASE, TIMER_A);
}

void TimeCounter()
{
```

```
58      seconds++;
59  }
60
61  uint32_t timeNow()
62  {
63      return seconds;
64  }
65
66  void reCount()
67  {
68      seconds = 0;
69  }
70
```

```
 1 #ifndef MAX_UART_DISPLAY_LIBRARY_UART_H_
 2 #define MAX_UART_DISPLAY_LIBRARY_UART_H_
 3
 4 #include "includes.h"
 5
 6 void UARTInit(void);
 7
 8 void UARTIntHandler(void);
 9 void UARTSend(const uint8_t *pui8Buffer, uint32_t ui32Count);
10 void UARTSendByte(uint8_t ui8Byte);
11 void excludeUART();
12
13 #endif
14
```

```c
#include "UART.h"

void UARTInit(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);


    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTIntRegister(UART0_BASE, UARTIntHandler);

    // Configure the UART for 115,200, 8-N-1 operation.
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                             UART_CONFIG_PAR_NONE));

    // Enable the UART interrupt.
    IntEnable(INT_UART0);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
}

void UARTSend(const uint8_t *pui8Buffer, uint32_t ui32Count){
   // Loop while there are more characters to send.
   while(ui32Count--)
   {
       // Write the next character to the UART.
      UARTCharPutNonBlocking(UART0_BASE, *pui8Buffer++);
   }
}

void UARTSendByte(uint8_t ui8Byte)
{
    UARTCharPutNonBlocking(UART0_BASE, ui8Byte);
}



void excludeUART()
{
    UARTDisable(UART0_BASE);
}

```