unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

FÁBIO STABILE MONACO

CONTROLADOR PID DE TEMPERATURA BASEADO NO PIC18F4550 E UMA RESISTÊNCIA DE 2 kW

Guaratinguetá
2018

Fábio Stabile Monaco

CONTROLADOR PID DE TEMPERATURA BASEADO NO PIC18F4550 E UMA RESISTÊNCIA DE 2 KW

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estatual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica .

Orientador: Prof^{o} Dr. Samuel E. de Lucema

Guaratinguetá

2018

Monaco, Fábio Stabile
M734c Controlador PID de temperatura baseado no PIC18F4550 e uma resistência de 2 kw / Fábio Stabile Monaco – Guaratinguetá, 2018.
56 f : il.
Bibliografia: f. 43

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2018.
Orientador: Prof. Dr. Samuel E. de Lucena

1. Controladores PID. 2. Microprocessadores. 3. Projetos de engenharia. 4. Controle de temperatura. I.Título.

CDU 621.316.7

Luciana Máximo
Bibliotecária CRB-8/3595

unesp UNIVERSIDADE ESTADUAL PAULISTA
"JULIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

FÁBIO STABILE MONACO

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE
"GRADUADO EM ENGENHARIA ELÉTRICA"

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Profª. Drª. PALOMA MARIA SILVA ROCHA RIZOL
Coordenadora

BANCA EXAMINADORA:

Prof. Dr. SAMUEL E. DE LUCENA
Orientador – UNESP/FEG

Prof. Dr. FERNANDO RIBEIRO FILADELFO
UNESP/FEG

Prof. JOSÉ MARCELO DE ASSIS WENDLING JÚNIOR
UNESP/COTEC

Dezembro de 2018

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus pela minha vida, minha inteligência, minha família e meus amigos,

aos meus pais que sempre confiaram em mim e me proveram de todo apoio moral e financeiro necessários em minha jornada,

ao meu orientador, Profº Dr. Samuel E. de Lucema, que me forneceu todo o apoio necessário desde a escolha do tema até a conclusão do trabalho. O seu interesse e dedicação em orientar me trouxeram grande confiança e motivação para desenvolver o projeto e foram fundamentais para eu conseguir realizar esse trabalho,

aos professores do Departamento de Engenharia Elétrica, que através de seus empenhos e dedicação transmitiram seus conhecimentos, principal fator de meu sucesso.

RESUMO

O presente trabalho é a documentação de um projeto de desenvolvimento de um controlador PID utilizando o microprocessador PIC18F4550, que tem como objetivo realizar o controle de temperatura de uma planta composta por um reservatório de água e uma resistência de 2kW, sendo necessário um controle dinâmico, a fim de absorver possíveis variações do sistema e realizar o controle de forma autônoma e estável ao longo do tempo.

A utilização do controlador PID permite a obtenção de uma resposta mais rápida e minimiza as instabilidades do sistema, resultando em um sistema que tende rapidamente para o equilíbrio.

Neste ensaio, para realizar o controle da temperatura, foi utilizado o sensor de temperatura PT100 e para realizar o controle de potência foi utilizado um TRIAC, que possui como entrada a saída do PIC18F4450 que realiza o controle do ângulo de disparo do TRIAC e assim regula a potência disponibilizada na resistência de 2kW para o aquecimento.

O resultado obtido foi um sistema de controle utilizando o PID, que realiza o controle da temperatura, com o *software* MPLAB X, possuindo uma interface com o usuário, a qual permite a definição da temperatura desejada, trazendo como resultado um erro estacionário inferior a 4%.

O desenvolvimento do presente projeto permitiu uma visualização da construção genérica de um equipamento aplicado às indústrias, sendo que a diferença entre o projeto e um equipamento de utilização real é apenas a escala.

PALAVRAS-CHAVE: PID. PT100. PIC18F4550. TRIAC. MPLAB X.

ABSTRACT

The present work is the documentation of a project that aimed to develop a PID controller using a PIC18F4550 microprocessor. The controller has as its objective to realize the temperature control of a system, which is composed by a water tank and a 2kW resistance. For this reason, a dynamic control was necessary in order to absorb possible system oscillations and realize its control of an autonomous and steady way during the time.

The use of a PID controller allows a quicker answer and minimize system instabilities, leading to a system that tends to balance in a fast way.

In this essay, in order to realize the temperature control, it has been used a PT100 temperature sensor, and in order to realize the power control a TRIAC has been used. This TRIAC has as its input the PIC18F4550's output, that controls the shooting angle and as a consequence regulates the power that will be set available in the resistance for the heating.

The obtained result was a control system that uses the PID, controls the temperature with the MPLAB X's software, having an interface with the user which allows the setting of the desired temperature, having as a result an stationary error lower than 4%.

The development of this project allowed the visualization of the generic construction of an equipment applied to the industries, having as its only difference in comparison to an real use equipment is the scale.

KEYWORDS: PID. PT100. PIC18F4550. TRIAC. MPLAB X.

ÍNDICE DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

TRIAC:	Triode for Alternating Current
Buffer:	Buffer de tensão
Main:	Função principal ou função mãe
Wind up:	Carregar-se
Set Point:	Valor alvo
Rise Time:	Tempo de subida
Overshoot:	Sobresinal
PWM:	Modulação por largura de pulso
Duty cycle:	Ciclo de trabalho
MOSFET:	*Metal Oxide Semiconductor Field Effect Transistor*, ou transistor de efeito de campo metal
Conversor A/D:	Conversor analógico/digital
PT100:	Termoresistência
Kp:	Ganho proporcional
Kd:	Ganho derivativo
Ki:	Ganho integral
W:	Watt
ºC:	Graus Celsius
V:	Volts
A:	Ampère
Hz:	Hertz
s:	Segundo

SUMÁRIO

1 INTRODUÇÃO

Controladores são amplamente utilizados na indústria, nas mais variadas aplicações, como fornos, refrigeradores, ou, ainda, em áreas específicas, a exemplo a óptica e tratamentos térmicos de materiais.

Um exemplo de controlador é o PID, objeto do presente trabalho. Esse controlador calcula um erro entre o valor medido na saída e o valor desejado no processo (*Set Point*). Assim, tenta diminuir o erro apresentado entre a entrada e a saída, ajustando seus valores. É amplamente empregado na indústria, por não exigir um conhecimento preciso do modelo matemático do sistema a ser controlado e, devido que, para sintonizá-lo é necessário apenas variar os ganhos proporcional, integral e derivativo. Assim, caso haja alterações físicas no processo, será relativamente fácil re-sintonizar o controlador. O resultado é um sistema que tende rapidamente para o equilíbrio e permite a redução do *overshoot*, diminuindo assim o tempo de acomodação e evitando descontrole do processo, que poderia causar danos materiais, pessoais e outros.

Importante se faz mencionar que para controlar um processo são necessários os seguintes elementos:

- SENSOR: Utilizado para mesurar a variável que irá ser controlada com a precisão necessária para o determinado processo.
- CONTROLADOR: Pode ser digital ou analógico e deve ser sintonizado ou projetado para realizar o controle das variáveis necessárias para o processo.
- ATUADOR: Equipamento que irá atuar diretamente na variável do processo de acordo com a ação de controle transmitida pelo controlador.

No presente trabalho, estão especificados todos esses elementos. A parte de Controlador é realizada por um PIC 18F4550, a parte de Sensor é realizada utilizado um PT100 (termoresistor) com circuito adicional para garantir sua operação e, por fim, a parte do Atuador é realizada por um ebulidor de 2kW, sendo controlado através de um circuito de potência, que possui como principal elemento um TRIAC, com ângulo de ataque controlado através do Controlador PID, implementado no PIC 18F4550.

1.1 OBJETIVOS

O presente trabalho tem como objetivo desenvolver um controlador PID utilizando microprocessador PIC18f4550 para realizar o controle de temperatura que atenda aos seguintes requisitos:

- Ser controlado digitalmente com capacidade para aquecer o sistema;
- Utilizar o sensor PT100 para mensurar a temperatura;
- Trabalhar em uma faixa de 0 a 100 graus Celsius;
- Possuir a capacidade de controlar a potência fornecida ao atuador (Resistência de 2kW);
- Não possuir sobressinal (*overshoot*) na resposta do sistema, ou seja, não deve ultrapassar o limite de temperatura definida pelo *Set Point.*

1.2 ORGANIZAÇÃO

O trabalho encontra-se distribuído da seguinte forma:

- Primeiro capítulo, composto por introdução e objetivos do projeto.
- Capítulo 2, “Controladores e sistemas de controle”, descreve o controle PID e quais foram as motivações que levaram ao desenvolvimento do presente projeto.
- Capítulo 3, “Descrição do Projeto”, trata, de forma pormenorizada, todos os elementos que foram necessários para a implementação do projeto.
- Capítulo 4, “Dificuldades encontradas”, estão explicados os diversos problemas apresentados durante o desenvolvimento do projeto e quais as soluções aplicadas.
- Capítulo 5, “Aplicações e Resultados Práticos”, encontra-se descrito o algoritmo implementado no PIC 18F4550 para realizar o controlador PID, bem como os resultados obtidos na planta.
- Capítulo 6, “Conclusão”, está apresentada a conclusão sobre todas as etapas do projeto, bem como o aprendizado obtido.

2 CONTROLADOR E SISTEMA DE CONTROLE

Neste capítulo está abordada a descrição de um Sistema de Controle PID digital, suas principais equações e como funciona sua estruturação básica.

Também é abordada a motivação do desenvolvimento do projeto, a fim de ilustrar a sua aplicabilidade na indústria.

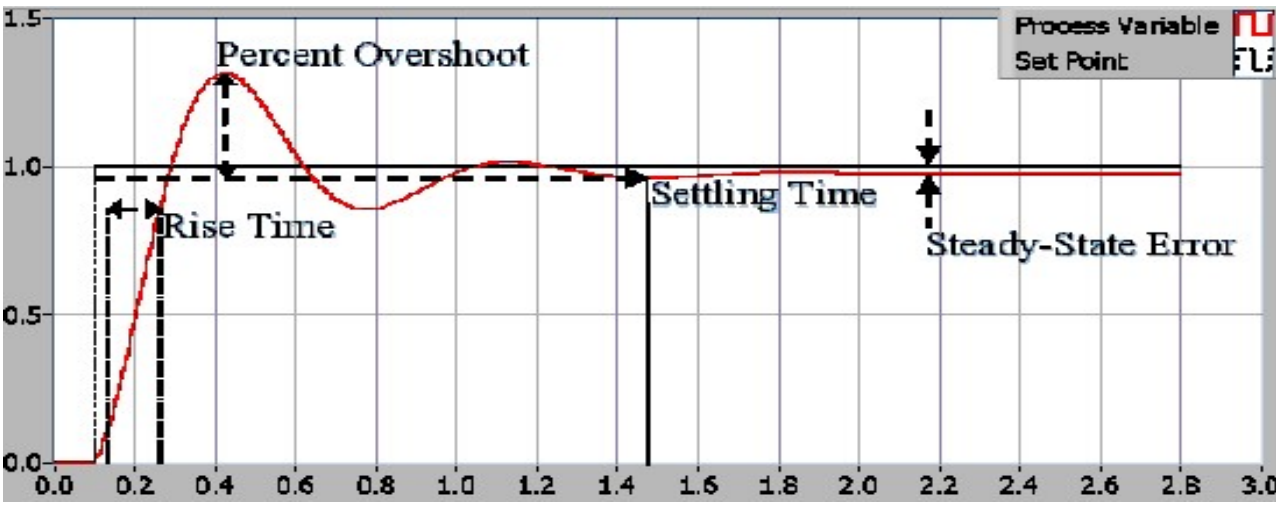
2.1 CONTROLE PID

Como o nome sugere, o algoritmo de controle PID (NATIONAL INSTRUMENTS, 2018) é composto por três coeficientes: proporcional, integral e derivativo, os quais são variados para obter a resposta ideal para a planta.

O controle PID tem como ideia básica ler um sensor e comparar a saída presente no sistema com uma entrada definida pelo usuário (*Set Point*), e a partir do erro obtido, calcular os sinais proporcional, integral e derivativo, para, então, somar os três componentes e definir a nova saída para o atuador.

Para se iniciar o projeto de um controlador PID, deve-se primeiro definir os parâmetros de desempenho do controlador. Para tanto, é aplicada uma função degrau definida como *Set Point* e, em seguida, medida a resposta da variável de processo. A Figura 1, ilustra uma resposta típica de um controlador.

Figura 1: Resposta típica de um controlador



Fonte: NATIONAL INSTRUMENTS, 2018.

A resposta é quantificada pelas características da onda de resposta. O tempo de subida (*Rise Time*) é o tempo que o sistema leva para ir de 10% a 90% do estado estacionário.

O *Percent Overshoot* é o valor que a variável de processo ultrapassa o valor final, expresso como uma porcentagem do valor final.

O *Settling time* é o tempo necessário para a variável do processo chegar dentro de uma determinada porcentagem (normalmente 5%) do valor final.

O *Steady-State* (erro estacionário) é a diferença final entre a variável do processo e o *Set Point*.

Para melhorar a *performance* de um controlador, em função dos parâmetros descritos anteriormente, é necessário ajustar os ganhos proporcional, integral e derivativo, sendo cada um responsável por melhorar a resposta em pontos específicos.

O componente proporcional do PID depende apenas da diferença entre o ponto de ajuste e a variável de processo. Esta diferença é referida como o termo de erro. O ganho proporcional (*Kp*) determina a taxa de resposta de saída para o sinal de erro.

Por exemplo, se o termo de erro tem uma magnitude de 10, um ganho proporcional de 5 produziria uma resposta proporcional de 50. Em geral, aumentando o ganho proporcional, irá aumentar a velocidade da resposta do sistema de controle (*Rise time*).

No entanto, se o ganho proporcional for muito grande, a variável de processo começará a oscilar. Se o *Kp* é aumentado ainda mais, as oscilações ficarão maiores e o sistema ficará instável, podendo oscilar até mesmo de forma incontrolável.

Já o ganho derivativo do PID (*Kd*), fará com que o sistema de controle reaja mais fortemente às mudanças no parâmetro de erro, aumentando a velocidade da resposta global de controle do sistema (*Settling Time*). O ganho derivativo é muito pequeno, pois a derivada de resposta é muito sensível ao ruído no sinal da variável de processo.

Se o sinal de realimentação proveniente do sensor for ruidoso ou o tempo de amostragem para realizar o controle for muito lento, a derivada de resposta pode tornar o sistema de controle instável.

A componente de ganho integral (*Ki*) soma o termo de erro ao longo do tempo. O resultado é que, mesmo um pequeno erro, fará com que a componente integral aumente lentamente. A resposta integral irá aumentando ao longo do tempo a menos que o erro seja zero, portanto, o efeito é o de conduzir o erro de estado estacionário para zero.

2.2 MODELAGEM DO CONTROLE PID DIGITAL

Para modelar o controlador PID digital, deve-se amostrar a equação do PID analógico no domínio do tempo. A Equação 1 do PID analógico é apresentada abaixo.

$$y(t) = K\left(e + \frac{1}{T_i}\int_0^t e(t)dt + T_d e\right) \quad (1)$$

Utilizando as aproximações da derivada pelo método das diferenças finitas e da integral pela integração trapezoidal, define-se a equação linear algébrica genérica aproximada do controlador PID. A transformação analógico-digital da Equação do PID analógico está completamente descrita no artigo publicado pelo Prof. Marcelo Maciel, Maciel (2012).

$$u_k - u_{k-1} = K\left(e_k - e_{k-1} + \frac{T}{T_i}\left[\frac{e_k - e_{k-1}}{2} + e_{k-1}\right] + T_d\,\frac{e_k - 2e_{k-1} + e_{k-2}}{T}\right) \quad (2)$$

Sendo: T: período de amostragem

K: ganho proporcional

Td: ganho derivativo

Ti: ganho integral

u_k: Saída da amostragem Atual

u_{k-1}: Saída da amostragem anterior

ek: erro da amostragem atual

e_{k-1}: erro da amostragem anterior

e_{k-2}:erro de duas amostragens anteriores

2.3 MOTIVAÇÃO PARA O DESENVOLVIMENTO DO PROJETO

Diversas técnicas de controle podem ser empregadas num sistema, contudo, o PID é uma técnica clássica e pode ser aplicada a sistemas com mais de uma entrada (NATIONAL INSTRUMENTS, 2018). Outro ponto importante é que, caso haja uma mudança na planta, não será necessário criar um novo sistema de controle.

Muitos sistemas industriais necessitam ter um controle de temperatura em um fluido, como, por exemplo, no caso da indústria alimentícia.

Respeitadas as devidas proporções, esse projeto poderia ser aplicado a qualquer sistema que necessite de um controle de temperatura com elevada repetibilidade, uma vez que foi necessário realizar o desenvolvimento completo de um sistema de controle, desde o *hardware* necessário para o controle do processo, até o *software* a ser implementado no microcontrolador, e, por fim, fazer a sintonia do PID digital.

Muitos processos industriais são controlados utilizando microcontroladores semelhantes ao utilizado neste trabalho (PIC 18F4550) com rotinas de controle baseadas no PID.

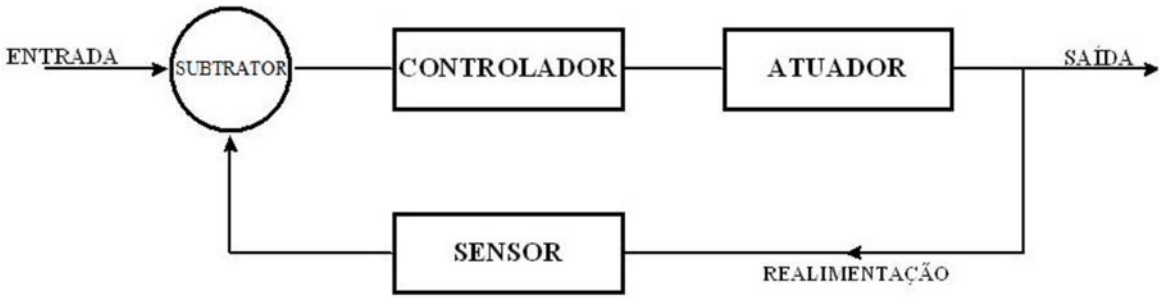
Esse projeto permitiu um aprendizado aprofundado do que é necessário para implementar um controle PID e observar as dificuldades de sua implementação.

Além disso, permitiu, ainda, aplicar uma boa parte dos conhecimentos aprendidos na graduação, abrangendo desde as matérias relacionadas a eletrônica e sistemas micro processados até o curso de controle.

3 DESCRIÇÃO DO PROJETO

O sistema implementado neste projeto pode ser divido conforme o diagrama de blocos apresentado na Figura 2.

Figura 2: Diagrama de blocos do Sistema



Fonte: Produção do próprio autor.

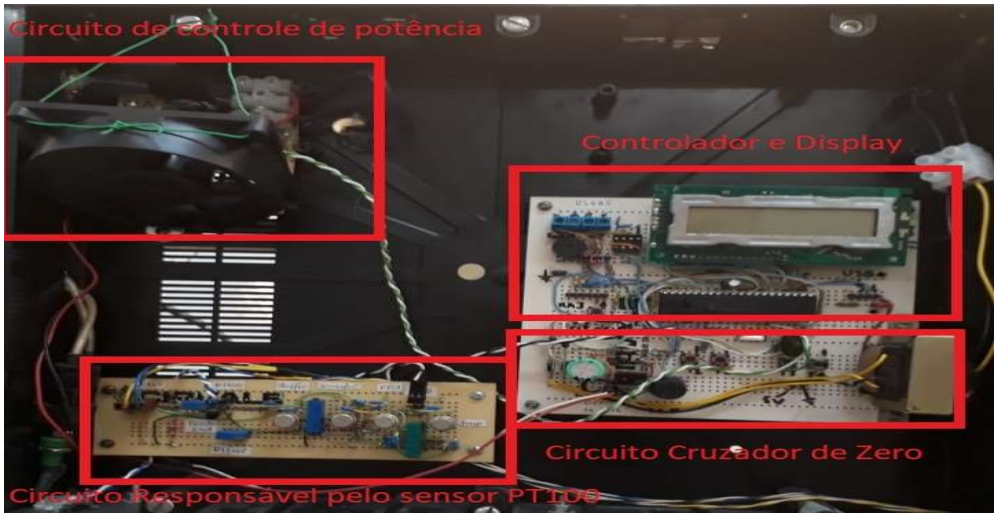
A entrada é composta pela temperatura final desejada para a água. Os papéis de controlador e subtrator são desempenhados pelo microcontrolador PIC 18F4550, que utiliza a entrada definida pelo usuário (*Set Point*) e a compara com o sinal proveniente do sensor.

Este projeto está dividido em duas seções básicas, sendo elas *hardware* e *software*, conforme apresentado a seguir.

3.1 HARDWARE

O *hardware* é dividido nas seguintes subseções: Circuito Cruzador de Zero, Circuito de aquisição de temperatura (responsável pelo sensor PT100), Circuito de controle de potência e a estrutura composta pelo Controlador e *Display*, conforme indicado na Figura 3.

Figura *3*: *Hardware* presente no projeto

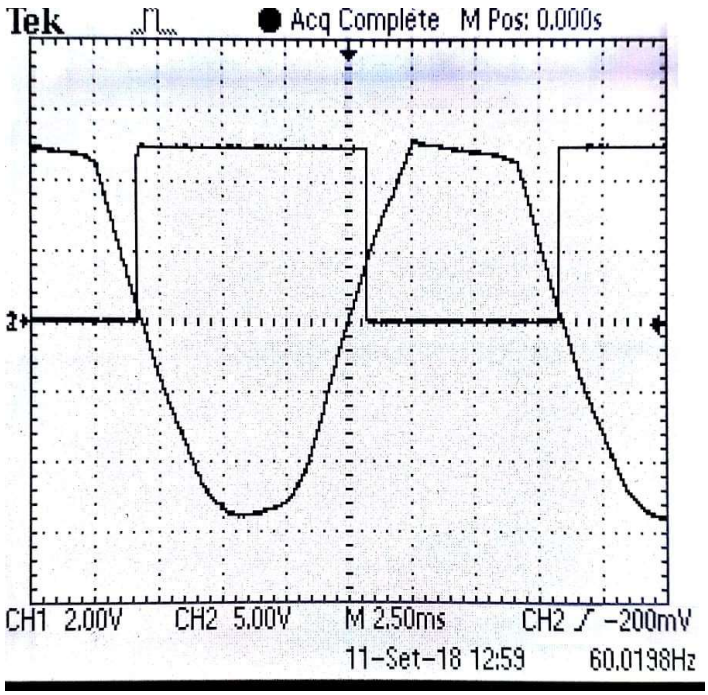


Fonte: Produção do próprio autor.

3.1.1 Circuito Cruzador de Zero

Essa parte do *hardware*, presente no projeto, é responsável por detectar a passagem do semi-ciclo negativo proveniente da rede elétrica e gerar uma onda quadrada síncrona a ela. A Figura 4 apresenta a forma de onda da entrada e saída do circuito cruzador de zero.

Figura *4*: Forma de onda da entrada (CH2) pela saída (CH1) do circuito do cruzador de zero



Fonte: Produção do próprio autor.

O sinal gerado por esse circuito é utilizado pelo controlador para realizar o controle do ângulo de disparo do TRIAC, presente no circuito de controle de potência.

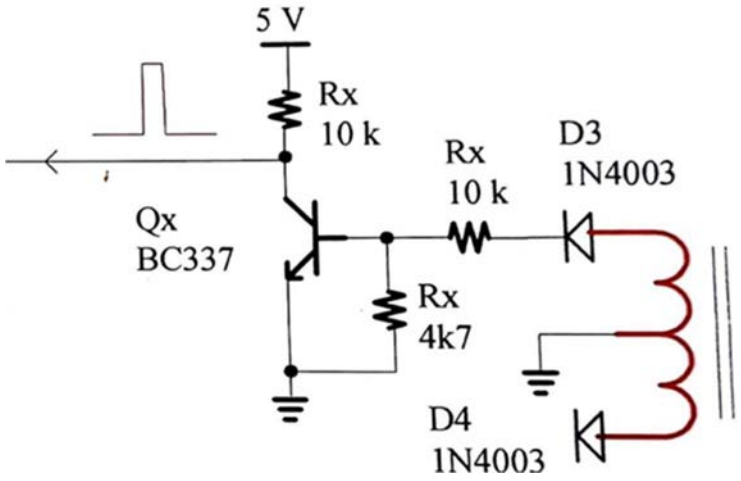
O circuito é composto por um transformador abaixador de tensão (110V para 12 + 12v @300 mA), tendo em um de seus enrolamentos um retificador de meia onda, realizado por um diodo 1N4003. O sinal, então, é utilizado para chavear o transístor (BC337) entre as zonas de corte e saturação.

Durante o semi-ciclo negativo, a tensão entre base e emissor do transistor passa a ser 0 volt, fazendo com que o transistor opere na região de corte e a saída para o microcontrolador seja 5 volts. Já no semi-ciclo positivo (a partir de uma tensão mínima na saída do transformador) a tensão entre base e emissor do transístor atinge o valor e de 0,7 volts, fazendo com que o transistor sature e a saída do cruzador de zero passe a ser 5 volts.

Assim, na saída do cruzador de zero é obtido uma onda quadrada síncrona ao sinal senoidal proveniente da rede elétrica.

O esquemático do cruzador de zero é apresentado na Figura 5.

Figura *5*: Circuito cruzador de zero

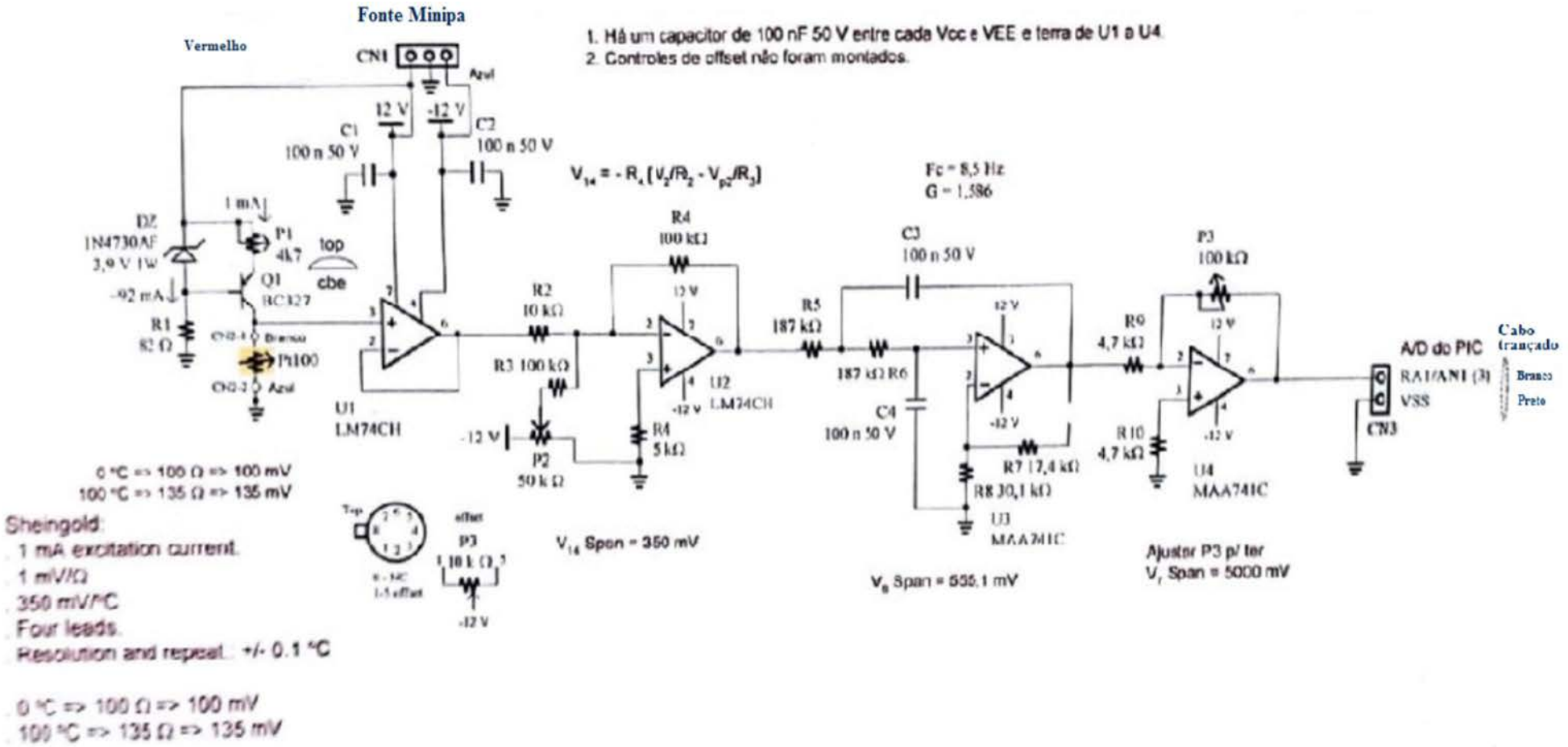


Fonte: Produção do próprio autor.

3.1.2 Circuito de aquisição de temperatura

Para realizar a aquisição da temperatura foi necessário um *hardware* adicional, a fim de permitir a operação do do PT100, que se trata de um termoresistor, onde varia a sua resistência conforme a temperatura em que se encontra. A Figura 6 ilustra esse circuito.

Figura *6*: Circuito responsável pela aquisição da temperatura



Fonte: Produção do próprio autor.

O primeiro elemento constituinte do circuito de medição de temperatura é um espelho de corrente para fornecer uma corrente de 1mA para o PT100 e, assim, permitir a aquisição da temperatura.

O PT100 é uma resistência que varia com a temperatura, sendo que a 0ºC sua resistência é 100 Ω e, por se tratar de um controle de temperatura de até 100ºC, a resistência a esta temperatura é de 135 Ω.

Para evitar o sobrecarregamento do espelho de corrente, foi inserido um *Buffer* (U1) para separar o circuito de medição e a parte de filtragem e amplificação. O próximo estágio (U2) é um somador inversor, onde a sua saída vai variar de acordo com a variação de tensão obtida sobre o PT100.

A menor tensão observada na entrada do estágio U2 é de 100mV e a máxima tensão observada é de 135mV. Assim, ajustou-se o potenciômetro P2 para que, a 0ºC, o somador produza uma saída de 0V, e a 100ºC, uma saída de 350mV.

Após este primeiro estágio de amplificação e ajuste da tensão, foi inserido um circuito para filtragem do sinal (U3), retirando as possíveis interferências ocorridas através do chaveamento do TRIAC (contudo, foi necessário utilizar também outras medidas para as quais será dedicado um capítulo exclusivo neste trabalho). Este filtro está sintonizado com uma frequência de corte de 8,5Hz e, além disso, proporciona um ganho do sinal com fator de multiplicação de 1,586.

Por último, para realizar o ajuste da tensão nos parâmetros adquiríveis do microcontrolador 18F4550, o qual possui um conversor A/D que responde de 0 a 5V, inseriu-se um amplificador inversor (U4) no qual, ajustando o resistor P3, chegou-se a uma saída de 5V a 100ºC e 0V a 0ºC.

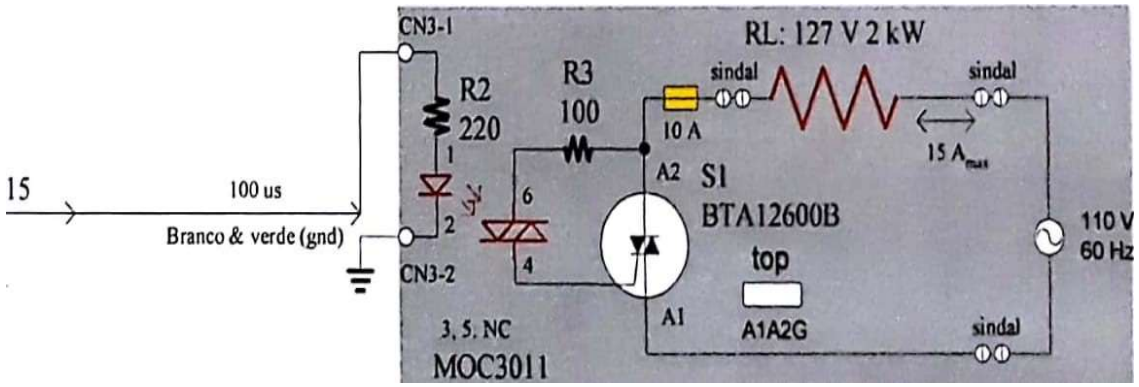
3.1.3 Circuito de Controle de Potência

Para a parte de potência é necessário um circuito que seja capaz de realizar o interfaceamento entre o circuito de potência e o circuito do controlador.

Para isso, foi adicionado um fotoacoplador a fim de isolar eletricamente o circuito de controle e o circuito de potência.

O circuito apresentado no esquemático abaixo (Figura 7) ilustra todo o *hardware* responsável por controlar a parte de potência que possui uma elevada tensão, esse circuito é responsável por realizar a alteração da potência injetada na resistência RL de 2kW a partir do sinal proveniente do PID digital implementado.

Figura 7: Circuito de potência



Fonte: Produção do próprio autor.

Observa-se na Figura 7 que o PIC 18F4550 envia um pulso de 184us para o disparo do TRIAC (superior ao mínimo de 100us) através do Pino 15 do controlador. Esse pino está conectado ao fotoacoplador MOC3011, a fim de se evitar que, caso haja um problema no circuito de potência, o mesmo não venha a ser transmitido ao microcontrolador, acarretando a sua queima. Além disso, para evitar a queima do sistema de potência em caso de um curto circuito, foi inserido um fusível de 10A.

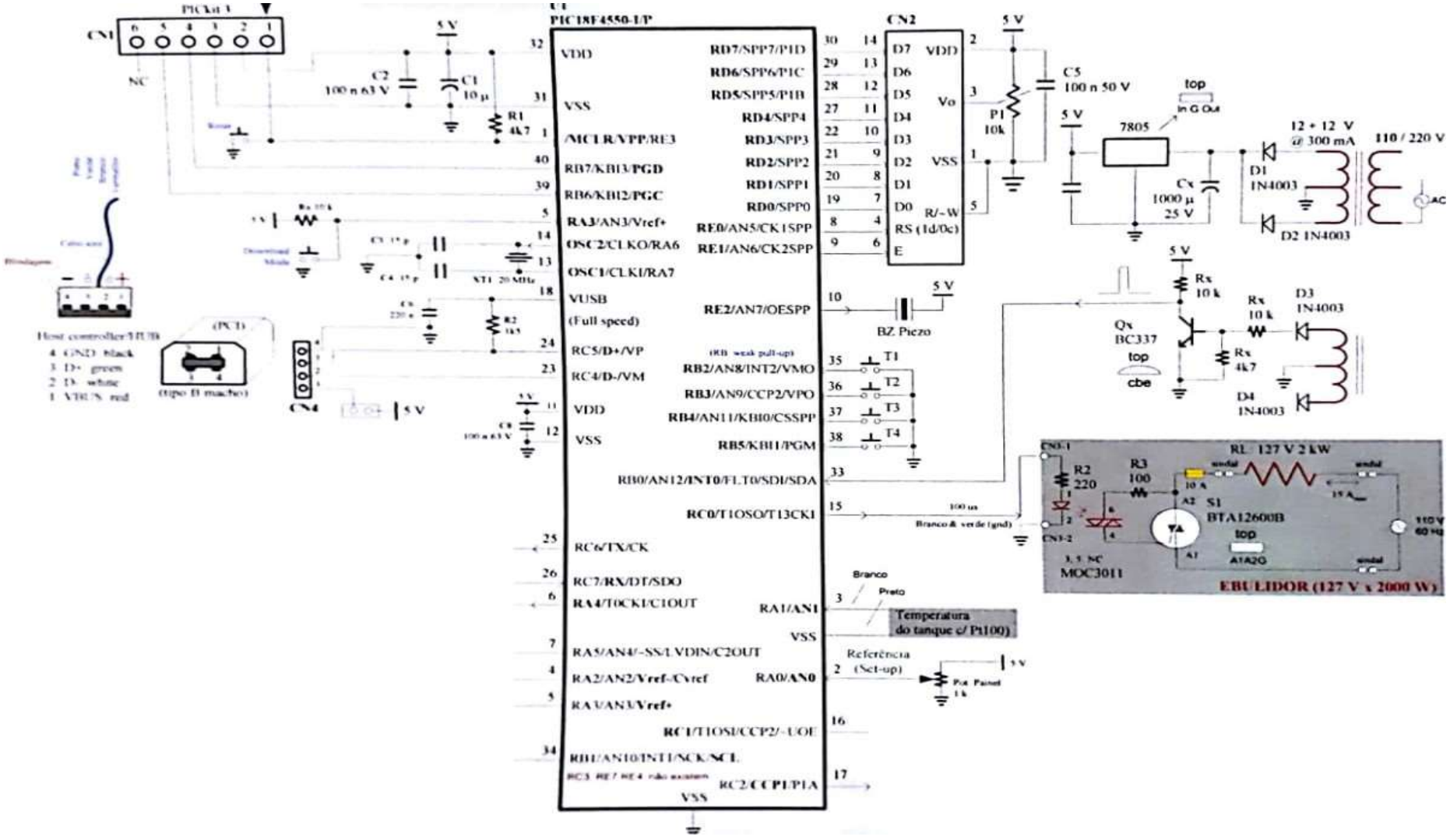
A saída do fotoacoplador está conectada ao pino de *Gate* do TRIAC (BTA12600B) e à entrada do sinal de potência, através de uma resistência de 100 Ω. Ao receber um pulso proveniente do controlador, o fotoacoplador permite que o o sinal de potência seja conduzido ao *gate* realizando o disparo do TRIAC. Controlando o ângulo de disparo do TRIAC é possível controlar a potência aplicada na resistência RL de 2kW (ebulidor).

Nesse projeto, por não ser necessária a utilização de uma potência superior a 1kW, pois trata-se de uma planta de simulação, onde utiliza-se um reservatório de apenas 1,5 litros, é realizado o disparo do TRIAC apenas para meio ciclo do sinal da rede. Portanto, o circuito de potência se comporta como um retificador de meia onda controlado por um TRIAC, a fim de se ajustar com exatidão a potência disponibilizada na resistência de 2kW. Esse ajuste é necessário, pois, através do ajuste da potência transferida à resistência, é possível realizar o controle da temperatura da planta.

3.1.4 Controlador e Display

O microcontrolador utilizado neste projeto, como já dito anteriormente, é o PIC 18F4550 desenvolvido pela Microchip. O diagrama de todos os elementos conectados no PIC são apresentados na Figura 8.

Figura *8*: *Hardware* completo



Fonte: Produção do próprio autor.

A seguir, estão relacionadas as entradas e saídas utilizadas diretamente para o controle da temperatura: as entradas RB2 e RB3 (pinos 35 e 36) para botões; a saída RC0 (pino 15), que realizará o controle do ângulo de ataque do TRIAC; a entrada de interrupção INT0 (pino 33) para receber os pulsos provenientes do cruzador de zero e, assim, realizar o controle do ângulo de disparo do TRIAC, através da saída RC0; a entrada RA0 como o potenciômetro (pino 2), que dará a referência de temperatura que será definida pelo usuário; e a entrada AN1 (pino 3), que receberá o sinal proveniente do circuito do PT100, que fornecerá a atual temperatura em que a água se encontra. Vale lembrar que os pinos 2 e 3 são canais de A/D que realizarão a conversão dos dados analógicos para serem tratados posteriormente pelo controlador.

O último elemento constituído do *hardware* para controle é o *display*, o mesmo está apresentado como item CN2 da Figura 8, necessitando de diversas saídas do controlador. Esse display é utilizado para mostrar ao usuário a atual temperatura da água e o *Set Point* definido para o controlador, o qual buscará atingir tal temperatura.

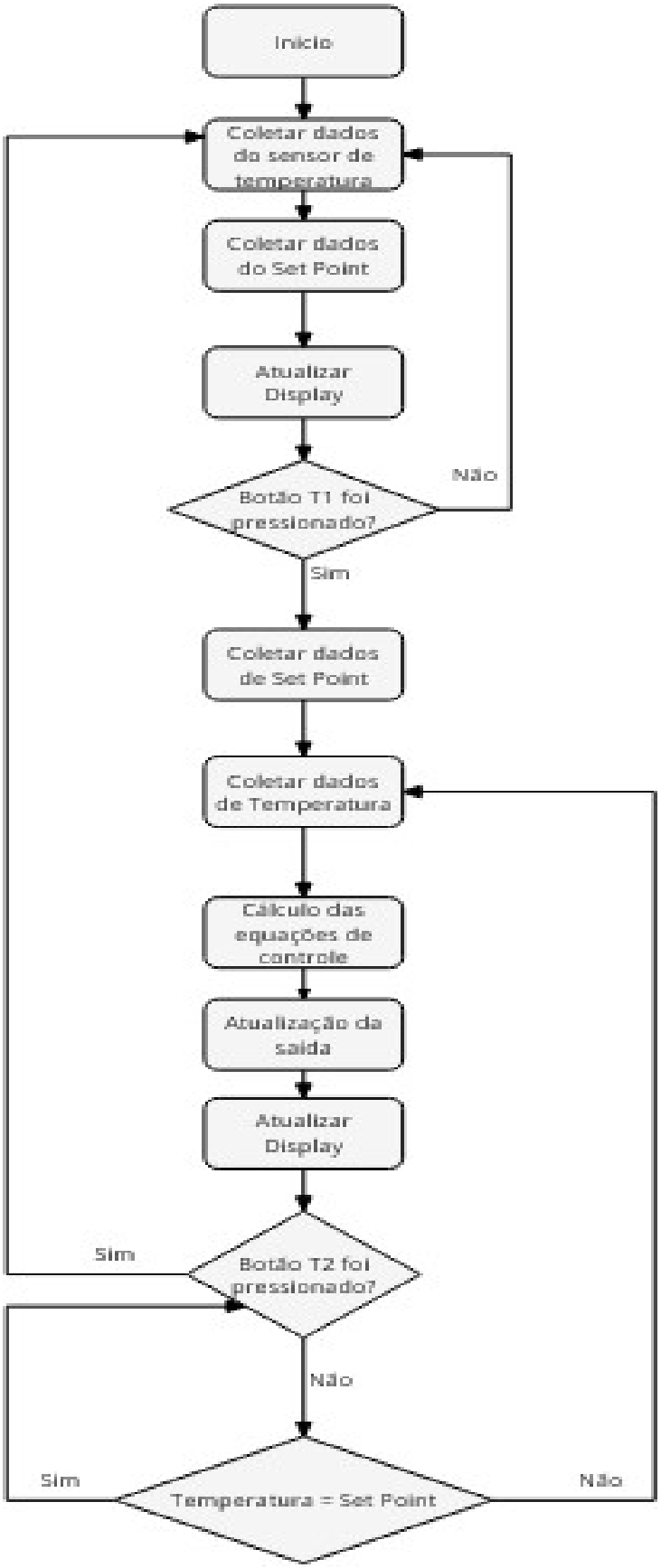
3.2 SOFTWARE

O programa final foi desenvolvido utilizando o *software* MPLAB X (MICROCHIP, 2018) com o compilador MPLAB XC8 C (MICROCHIP, 2018) ambos desenvolvidos pela MICROCHIP e de distribuição gratuita.

Vale ressaltar que esse compilador foi escolhido por possuir uma interface amigável e por ser baseado na linguagem C, uma das matérias que foram ministradas durante a graduação do curso de engenharia na FEG / UNESP.

O programa comentado encontra-se no demonstrado na Figura 9 (fluxograma divido em seções, referentes a cada função desempenhada).

Figura *9*: Fluxograma do *software* implementado no PIC18F4550



Fonte: Produção do próprio autor.

3.2.1 Funções

Dentre os elementos principais presentes no programa estão: as funções referentes ao controle dos caracteres apresentados no *display*, o controle do ângulo de disparo do TRIAC e o controle PID.

O *Setup* inicial é uma das principais "funções" do programa desenvolvido, pois realiza a inicialização das variáveis, do *display*, da interrupção e do temporizador. Dessa forma, ela é crucial para a implementação do projeto, uma vez que dentre seus elementos constituintes estão os cabeçalhos iniciais da estruturação de todos os elementos utilizados no projeto.

A função *Main* possui o corpo principal do programa, e chama as outras funções presentes no programa para serem executadas. É nesta função em que o programa realiza o *looping* (verificação constante dos valores de temperatura atual e *Set Point*), atualizando sempre o *display* e realizando as funções: "LCD_temp" e "LCD_setpoint".

Além disso, verifica-se frequentemente se o botão T1 foi pressionado e, caso positivo, habilita-se a interrupção que trava o valor do *Set Point* e inicia o controlador de temperatura.

Para o controle dos caracteres apresentado no *display* foram necessárias três funções principais: "LCD_temp" e "LCD_setpoint" *e* "*LCD_ini*".

A função "LCD_ini" é responsável por realizar a inicialização do *display* e permitir que o mesmo comece a operar, além de realizar a escrita das frases presentes no *display*, sendo elas: " $TEMPERATURA$ = " e " SET POINT =".

As funções "LCD_temp" e "LCD_setpoint" são responsáveis, respectivamente, por escrever os caracteres de temperatura atual e *Set Point* adquiridos através do conversor A/D.

Essa estruturação foi necessária para otimizar a escrita no *display*, uma vez que não é necessário reescrever as frases, apenas os caracteres referentes a atualização da temperatura e *Set Point* durante a operação do controlador, reduzindo, assim, o tempo de processamento.

Outra função de grande importância é a "atraso_92usX", que gera atrasos múltiplos de 92 us, através da variável de entrada, a qual define o múltiplo a ser utilizado.

Conforme descrito no tópico 3.1.3, é utilizado apenas meio ciclo, o qual foi dividido em 90 valores possíveis de ângulo de disparo, variando de 2 em 2 graus.

Como a senoide presente na rede elétrica possui uma frequência de 60Hz (e, portanto, um período de 1/2 ciclo = 8.33 ms) percebemos que, ao dividir o período por 90, é necessário realizar atrasos de 92 us para gerar a variação em questão.

Entretanto, vale destacar que foram definidos valores mínimos e máximos de ângulo de disparo no controle, a fim de se evitar que o disparo ocorra fora do ciclo esperado, ou desrespeite os limites operacionais apresentados pelo TRIAC.

No caso do controle do ângulo de disparo do TRIAC, o funcionamento ocorre de maneira inversa: caso a saída de controlador seja máxima (ou seja, 90 us), o ângulo de disparo será mínimo (ou seja, 0) e vice-versa.

Essa função de atraso também foi utilizada para realizar a definição da largura do pulso enviado para o disparo do TRIAC, como o pulso mínimo necessário é de 100us, o pulso utilizado foi de aproximadamente 184us. Sendo assim, o valor de entrada para essa função é 2.

3.2.2 Interrupção

Na interrupção foi implementada a estrutura do controlador PID e a escrita dos caracteres da temperatura e *Set Point*.

Para o PID, ocorreu a implementação baseada no cálculo individualizado de cada um dos seus ganhos (Proporcional , Integral e Derivativo), sendo que o tempo de amostragem é calculado a partir da quantidade de pulsos enviados pelo cruzador de zero. Considerando-se que a rede elétrica na qual o equipamento está inserido utiliza uma frequência estável de valor 60Hz e que o tempo de amostragem definido é de 1s, contava-se 60 pulsos provenientes do cruzador de zero. Aí, realizava-se uma nova atualização do ângulo de disparo do TRIAC.

É dentro desta interrupção que se realiza a atualização dos caracteres referentes à temperatura atual durante a operação do controlador e à verificação do botão T2 (que pode ou não ter sido pressionado, e é responsável por desabilitar a interrupção, desligando o controle PID de temperatura, e permitindo que o usuário atualize os valores de *Set Point*).

Destaca-se também que, se o valor de temperatura definida pelo *Set Point* for menor que a temperatura atual do sistema, o controle não irá operar, pois a planta apenas trabalha para o aquecimento - e não para o resfriamento.

Em contrapartida, caso a temperatura seja reduzida de tal maneira que seja inferior ao *Set Point*, e o sistema estiver operando, o controlador irá trabalhar, fazendo a temperatura da água atingir o equilíbrio térmico com o *Set Point* definido.

4 DIFICULDADES ENCONTRADAS

Neste tópico, serão apresentadas todas as principais dificuldades encontradas durante a implantação do projeto, sendo necessárias modificações físicas e/ou de *software* para que o sistema atingisse o objetivo proposto.

4.1 INTERFERÊNCIAS NO CIRCUITO DE MEDIÇÃO DE TEMPERATURA.

O primeiro problema encontrado durante o desenvolvimento do projeto foi o acoplamento capacitivo entre o sensor PT100 e o ebulidor de 2kW.

Ao se iniciar a rotina de controle e o controlador proporcionava a potência máxima no ebulidor devido ao elevado erro entre o *Set Point* e a temperatura do fluido. Com isso, o sensor PT100 se acoplava capacitivamente com o ebulidor e perdia todo o sinal medição.

Em uma das análises realizadas, ao ebulidor entrar em operação, o sinal proveniente do sistema do PT100 reduzia-se a sua amplitude (indicando que o sistema estava a uma temperatura menor que a temperatura real) e fazendo com que o controlador disponibilizasse uma potência maior.

Contudo, com aumento da temperatura, o controlador passava reduzir novamente a potência enviada ao ebulidor e assim o sensor passava a transmitir o valor real da temperatura que era muito superior desligando abruptamente o ebulidor e ainda estando a uma temperatura superior a definida pelo *Set Point*.

Para tentar solucionar esse problema inicialmente foi inserido um filtro passa baixas sintonizado na frequência de 8,5Hz (como descrito anteriormente), mas não solucionou completamente o problema e o mesmo persistia.

Houve uma redução na interferência do sinal de medição, contudo a interferência continuava significativa e prejudicava o controle. Outra tentativa realizada foi a substituição da fonte de alimentação do circuito do sensor que inicialmente o mesmo era alimentado através de uma fonte chaveada.

Essa alimentação foi substituída por uma fonte MINIPA MPC - 303DI a qual é isolada da rede por um transformador. Combinando as duas tentativas houve uma melhora significativa,do problema mas ele ainda persistiria.

A solução definitiva para o problema apresentado foi a combinação das duas tentativas anteriores com o aterramento da carcaça externa do PT100 na fonte de alimentação o que gerou

uma blindagem ainda mais efetiva e permitiu que mesmo que o controlador disponibilizasse a potência máxima no ebulidor (1kW) o sinal do sensor não sofria alteração.

4.2 AQUECIMENTO DO TRIAC

O TRIAC utilizado neste projeto foi o BTA1600B, o qual suporta uma corrente máxima de 12A. Como já descrito anteriormente o mesmo está protegido por um fusível de 10A garantindo que não queime, mesmo que ocorra algum problema de sobrecarga ou curto-circuito.

Contudo, considerando que esteja sendo disponibilizada a potência máxima ao ebulidor (1kW) a uma tensão de 127V a corrente eficaz seria de 7,88A.

Ao se verificar as curvas relacionadas a dissipação térmica do TRIAC e realizar os cálculos relacionados a transferência de calor, verificou-se que seria necessário instalar um dissipador para aumentar a eficiência de troca térmica.

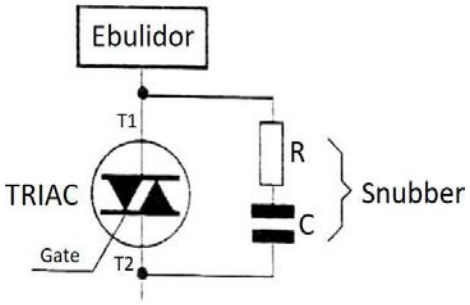
Assim, foi instalado um dissipador no componente e um *cooler* para realizar a ventilação forçada, garantindo que a temperatura do componente estivesse dentro de sua margem operacional, a fim de evitar a sua queima por superaquecimento.

4.3 PROBLEMAS NO CHAVEAMENTO DO TRIAC

Ao se iniciar o chaveamento do TRIAC, o mesmo não respondia corretamente devido aos transientes de alta tensão durante a comutação da carga. Por se tratar de uma resistência em formato de bobina, o ebulidor se comportava como um indutor gerando elevados transientes durante o chaveamento do TRIAC e o mesmo não respondia conforme o esperado.

A solução encontrada foi inserir um circuito denominado *Snubber*, o qual é formado por um capacitor (100 nF) em série com um resistor (330 Ω) ligados entre o principal 1 (T1) e o principal 2 (T2), conforme a Figura 10.

Figura *10*: Circuito de *Snubber*



Fonte: (BRAGA, 2018)

A inserção desse circuito (o qual não consta nas Figuras 6 e 7, devido se tratar de um circuito de auxílio) trouxe inúmeros benefícios à operação do TRIAC, não apenas por reduzir os transientes de alta tensão que ocorrem na comutação de uma carga, mas também por evitar a sobrecarga do TRIAC, uma vez que protege a carga contra transientes de comutação e evita a queima do TRIAC enquanto ele é energizado.

Estes transientes podem causar interferências, como forçar o dispositivo de comutação, podendo provocar sua queima.

O circuito *Snubber* também é utilizado para proteger contatos de relé, absorvendo a energia gerada na comutação de cargas indutivas.

4.4 WINDUP

O problema de *WindUp* é ocasionado por problemas relacionados a parcela integral do PID. Partindo da equação genérica do controlador PID digital descrita no capítulo 2, podemos observar a fonte desse problema.

$$u_k - u_{k-1} = K\left(e_k - e_{k-1} + \frac{T}{T_i}\left[\frac{e_k - e_{k-1}}{2} + e_{k-1}\right] + T_d\,\frac{e_k - 2e_{k-1} + e_{k-2}}{T}\right) \quad (2)$$

A partir da equação apresentada, pode-se perceber que, caso os incrementos posteriores no sinal de controle não contribuam para uma resposta mais rápida do sistema, a integração do erro do sistema fará com que o termo integral alcance valores elevados, sem qualquer efeito sobre a saída da planta.

Assim, o erro terá que assumir valores negativos durante um longo intervalo de tempo, para que o termo integral possa voltar ao valor estacionário esperado, provocando um elevado sobre-sinal e aumentando o tempo de acomodação, tornando-o relativamente

longo. Neste caso, para que o sistema possa ter um desempenho satisfatório, um mecanismo anti *WindUp* deve ser implementado no controlador PID.

Um exemplo de solução (NETO, 2005) para esse tipo de problema é a integração condicional ou método de Anti *WindUp,* que consiste em desligar a ação integral quando o controle está longe do regime permanente. Assim, a ação integral é ativada apenas quando certas condições pré-estabelecidas são satisfeitas, caso contrário o termo integral é mantido constante, ou seja, a entrada do integrador é mantida em zero. É possível realizar esse tipo de inibição das mais variadas formas, como desligar o ganho integral quando sinal de erro for grande. Outra maneira é desligar o integrador somente durante a saturação, contudo, esses métodos possuem como desvantagem o bloqueio do ganho quando o erro for muito grande, podendo aumentar o *Setting time*.

A alternativa mais utilizada (e que foi implementada para solucionar o problema de *WindUp* no projeto) consiste em desligar o integrador quando se encontrar saturado.

Assim, por exemplo, se o controlador está saturado no limite máximo, a ação seria desligada somente enquanto fosse positivo. Entretanto, quando o sinal de erro se tornasse negativo a ação integral voltaria a ser ligada com o intuito de descarregar o integrador.

5 APLICAÇÕES E RESULTADOS PRÁTICOS

No presente tópico será apresentado o desenvolvimento do programa presente no projeto e o método de sintonização do controlador PID.

5.1 UTILIZAÇÃO DO SISTEMA

O projeto foi desenvolvido para permitir que qualquer usuário o utilize de maneira prática e simplificada.

Inicialmente, o usuário deve definir a temperatura desejada utilizando o potenciômetro e pressionar o botão T1 (ilustrado na Figura 8) para iniciar o controlador. O controle irá iniciar-se automaticamente, buscando atingir a temperatura desejada.

O controlador irá manter a temperatura desejada até que seja pressionado o botão T2, o qual desliga o sistema e permite que o usuário modifique o *Set Point* novamente.

Na Figura 9 está ilustrado o fluxograma completo de todas as ações desempenhadas pelo programa desenvolvido para essa aplicação.

5.2 SINTONIZAÇÃO DO CONTROLADOR

A sintonização do controlador foi realizada utilizando o método descrito no artigo "PID without a PhD" (TIM WESCOTT, 2000). Nesse artigo, Tim descreve como realizar a sincronização de uma planta qualquer a partir da observação de sua resposta a uma entrada degrau unitário.

Nesse caso, inicialmente deve-se realizar pequenos ajustes no ganho proporcional, até observar pequenas oscilações na respostas da planta. Em seguida, deve-se utilizar o ganho derivativo (inicialmente cerca de 100 vezes menor) para corrigir os problemas relacionados a essa oscilação. Finalmente, utiliza-se o ganho integral para corrigir o erro estacionário presente no sistema (valor inicial em uma proporção entre o ganho integral e derivativo).

Devido a limitação da resposta do controlador, entre 0 (disponibilidade máxima de potência) e 90 (disponibilidade mínima de potência), foi necessário realizar uma modificação na saída do controlador. Ao iniciar o método descrito por Tim, era observado que os valores eram superiores a 90 em grandes proporções, mesmo em casos onde o erro era pequeno.

Assim, para solucionar esse problema, foi dividido por 100 o sinal que seria enviado para o atuador (circuito de controle de potência), proveniente do controlador PID, a fim de

enquadrar os valores dentro da margem esperada pelo circuito de controle de potência (valores entre 0 e 90, divisão esta aplicada no semi-ciclo do sinal proveniente da rede).

Após realizada a correção, foram aumentados os valores referentes ao ganho proporcional até se identificar a oscilação. No caso, o valor ideal obtido foi 17.

Começando com 0,17 (conforme o descrito no artigo) iniciou-se a análise para a correção do ganho derivativo, mas, neste caso, o valor estava muito maior que o a valor ideal provocando uma instabilidade na planta.

Reduzindo o ganho derivativo, obteve-se uma melhora na resposta do sistema e, em 0,3, foi retirada toda a oscilação presente no sistema.

Solucionada a oscilação da saída, começou-se a inserir gradualmente o ganho integral para retirar o erro de regime estacionário.

Inicialmente, se atribuiu um valor de 2 (uma proporção entre o ganho proporcional e derivativo), o qual foi elevado para se atingir a melhor resposta para o sistema. O valor ideal atingido durante esse processo foi 5.

Em todos os casos, definiu-se 1s como tempo de amostragem, pois trata-se de um sistema de grande inércia, que demanda um longo tempo para a variação de 1ºC na temperatura.

5.3 RESULTADOS OBTIDOS

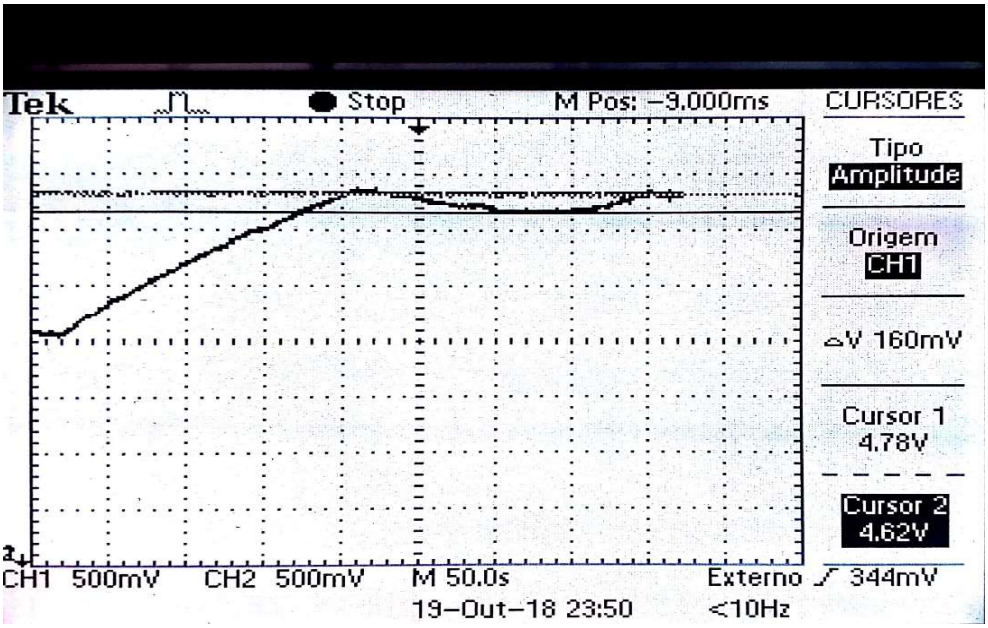
Em muitas aplicações industriais, o desrespeito ao limite da temperatura pode prejudicar todo o processo e acarretar perdas substanciais.

Nesse caso, é preferível que ocorra uma diferença maior entre a temperatura definida (*Set Point*) e a alcançada em relação a ocorrência de uma temperatura superior à delimitada.

Para impedir a ocorrência de um *overshoot* (sobressinal) utilizou-se os parâmetros descritos na secção anterior ($Kp = 17$, $Ki = 5$, $Kd = 0{,}3$).

A Figura 11 apresenta a resposta do sistema para temperatura inicial de 71.6ºC e *Set Point* de 96.6ºC (valores não suportados na exibição do display, mas utilizados no controle).

Figura *11*: Resposta do Sistema

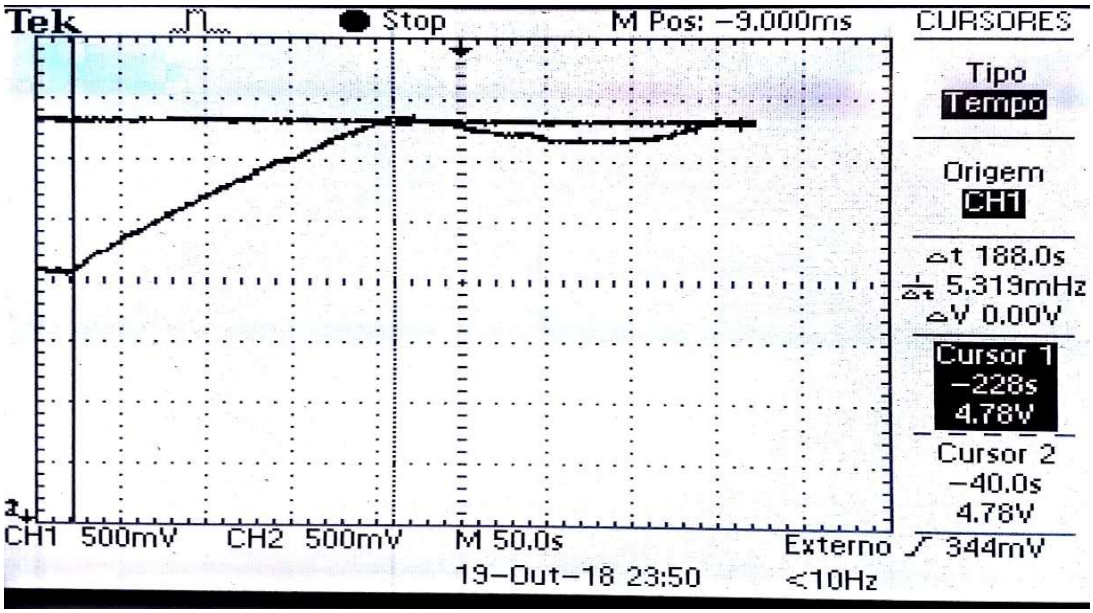


Fonte: Produção do próprio autor.

Observa-se na Figura 11 que o erro estacionário do sistema é de 3,4%, o qual é considerado aceitável na maior parte das aplicações que utilizam o controle de temperatura.

Outro fato perceptível é que o *rise time* (tempo de subida) do sistema está dentro do esperado. A Figura 12, representada abaixo, apresenta visualmente o *rise time*.

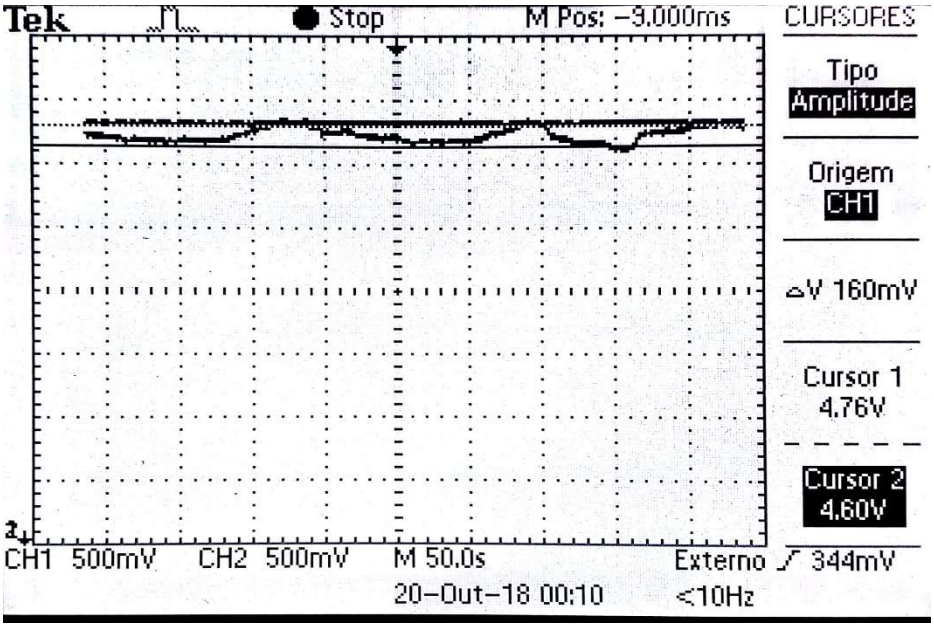
Figura *12*: Análise do tempo de subida



Fonte: Produção do próprio autor.

Para facilitar a visualização da análise do erro estacionário, utilizando o mesmo sistema apresentado anteriormente, deixou-se o controlador operando por um longo período de tempo (maior que 10 minutos) e efetuou-se o registro dos dados da resposta do sistema, conforme a Figura 13.

Figura *13*: Erro estacionário

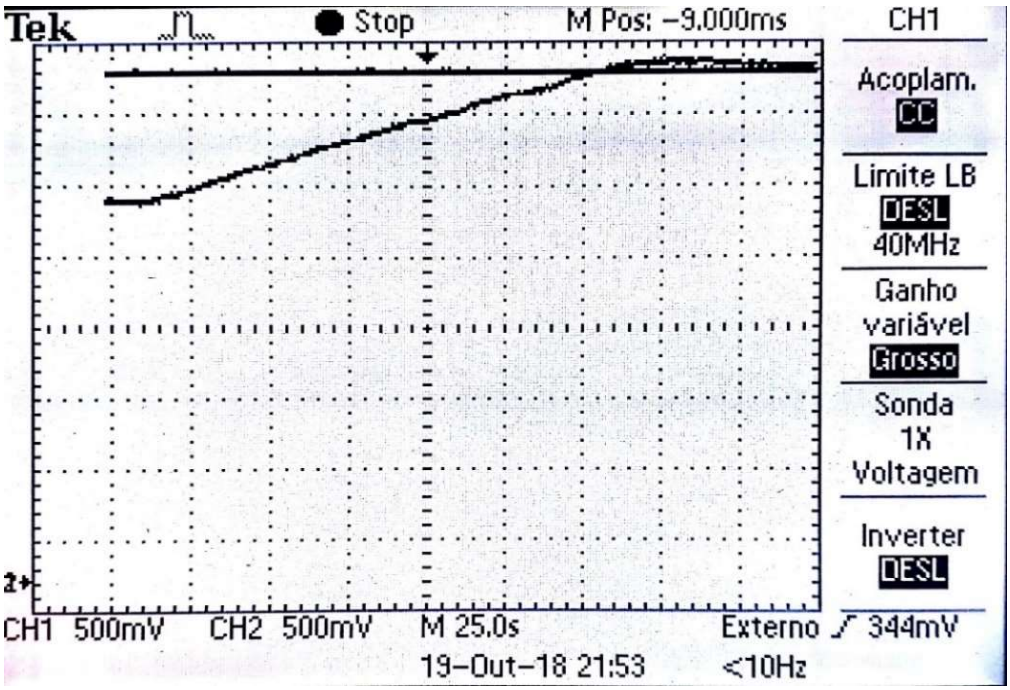


Fonte: Produção do próprio autor.

Com isso, constatou-se que o analisado inicialmente em um curto período de tempo não se alterou, e o erro médio permaneceu em 3,4%.

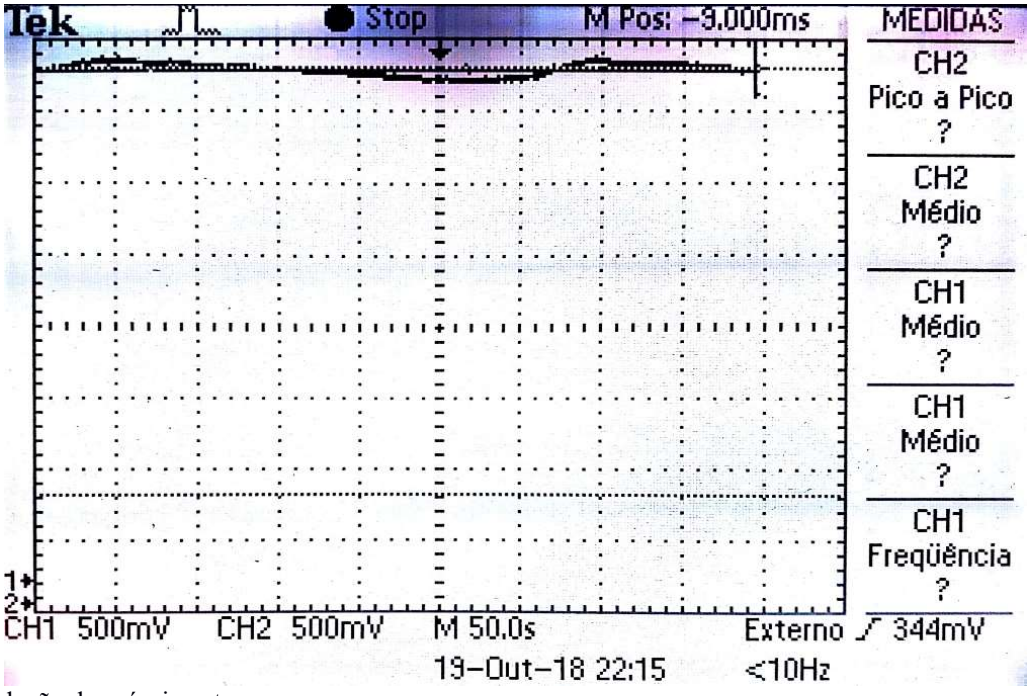
Outro resultado possível para o sistema de controle PID é, conforme as Figuras 14 e 15, a partir dos parâmetros $Kp = 17$, $Ki = 5$ e $Kd = 0{,}5$, com os quais ocorrerá um *overshoot* inicial de 100 mV ou 2ºC (não aceitável no caso em específico, mas que pode ser utilizado em outros sistemas por possuir um erro inferior à 1,5 %).

Figura *14*: Resposta do sistema com Kp = 17, Ki = 5 e Kd = 0,5



Fonte: Produção do próprio autor.

Figura *15*: Erro estacionário com *Kp* = 17, *Ki* = 5 e *Kd* = 0,5

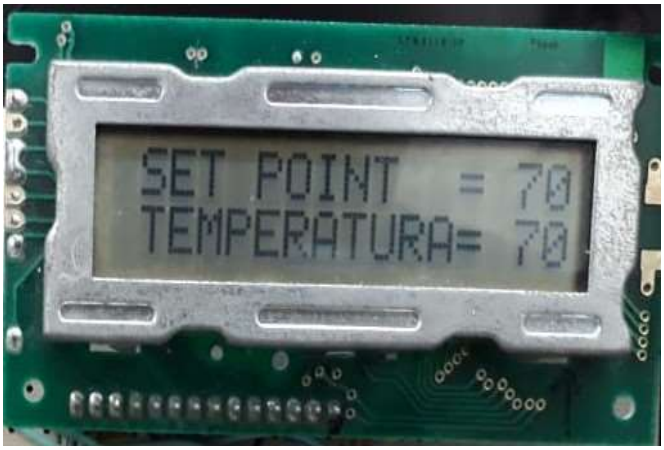


Fonte: Produção do próprio autor.

Na interação com o usuário, o *display* ilustra a temperatura atual e a temperatura definida, sendo que a temperatura atual do sistema é atualizada a cada 5s, pois trata-se de

um sistema cuja atualização não é de alta frequência, não comprometendo, assim, o processamento dos dados. A Figura 16 mostra o *display* em questão.

Figura *16*: Display



Fonte: Produção do próprio autor.

Por apresentar apenas valores inteiros de temperatura, o *display* apresenta valores aproximados da temperatura atual, não tendo a mesma resolução do sistema de controle, que considera variações de 0,1ºC.

Não utilizou-se a mesma resolução para a ilustração no LCD devido à limitação no tempo de processamento dos dados, pois no caso de utilizar tamanha precisão haveria a necessidade de aumentar a taxa de atualização do *display*, comprometendo o sistema de controle.

O microcontrolador PIC18F4550 utiliza um processamento serial que possibilita apenas a execução de uma tarefa por vez, e o processo de cálculo da temperatura, unido à atualização do *display*, necessitariam de mais do que um pulso de interrupção proveniente do cruzador de zero.

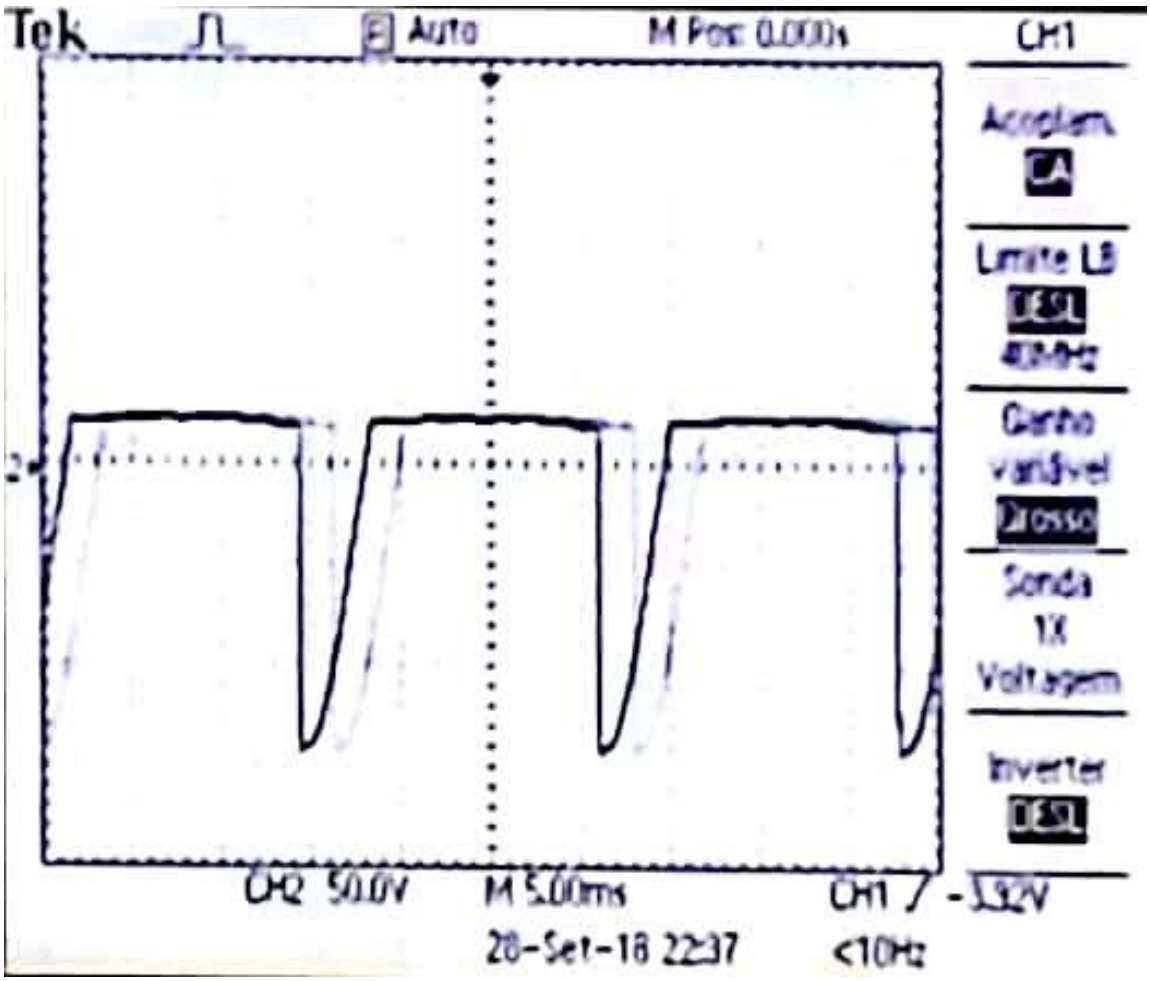
Nesse contexto, existe perda de interrupções, prejudicando o cálculo do tempo das equações de controle e variando o tempo de amostragem, tornando o sistema instável, conforme verificado na prática.

Outro aspecto que é visualizado durante a operação do sistema é a variação do ângulo de disparo do TRIAC durante a operação, conforme mostram as Figuras 17, 18 e 19.

A Figura 17 ilustra a resposta inicial do sistema a partir da aplicação de um degrau unitário na entrada, possuindo o valor máximo de potência sendo disponibilizada a

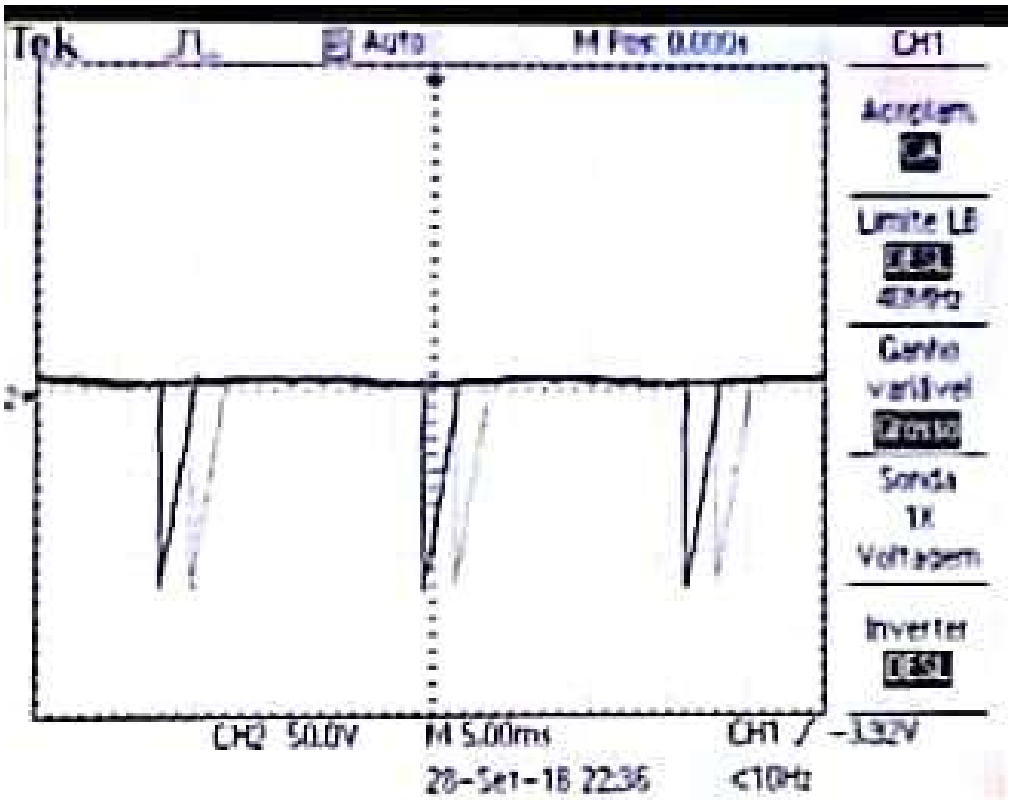
resistência de 2kW. A Figura 18 representa um estado em que o erro é menor e a potência disponibilizada possui valor intermediário. A Figura 19 mostra a situação estacionária.

Figura *17*: Pulso com Intensidade Máxima



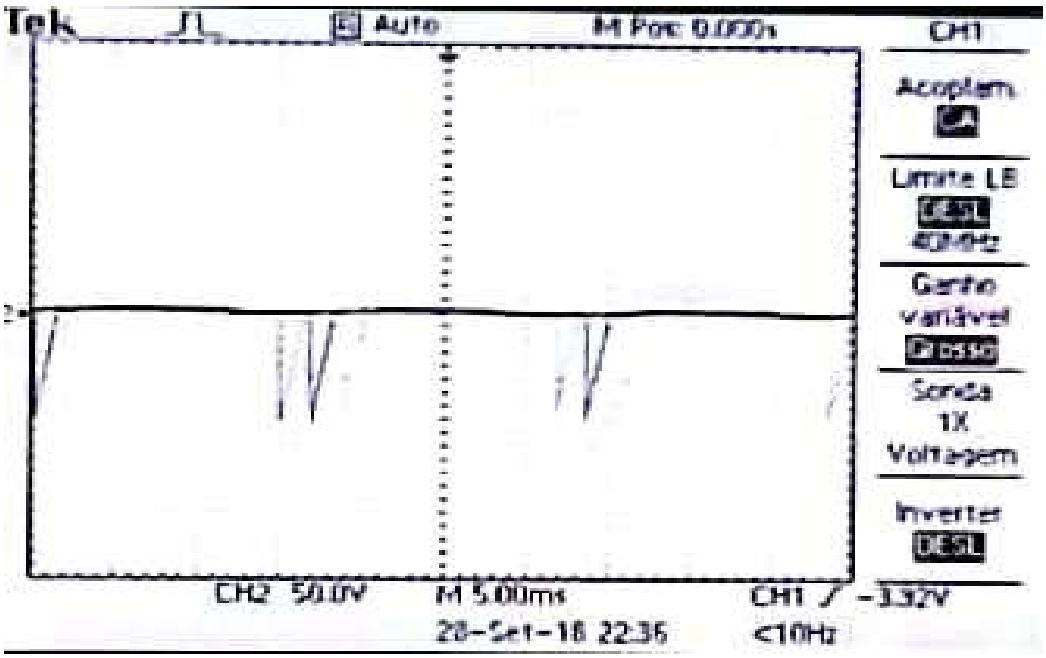
Fonte: Produção do próprio autor.

Figura *18*: Pulso com intensidade Média



Fonte: Produção do próprio autor.

Figura *19*: Pulso com Intensidade Mínima



Fonte: Produção do próprio autor.

Como explicado anteriormente, houve a inserção de limites de intensidade máxima e mínima, definidos na prática, a fim de evitar problemas no acionamento do ebulidor: o limite máximo definido foi 83 e o limite mínimo, 5.

Outro ponto importante analisado é a relação entre o ângulo de disparo do TRIAC e a potência injetada, pois, por se tratar de um sistema de corrente alternada, a relação entre o ângulo de disparo e a potência injetada é não linear.

Utilizou-se inicialmente uma equação para realizar a linearização da potência injetada, contudo, a mesma demandava um grande tempo de processamento, por possuir relações quadráticas.

Devido ao princípio de construção do controlador PID, o qual realiza a variação da saída de forma proporcional ao erro, não foi necessário desprender cálculos mais complexos para realizar a linearização.

Sendo assim, com a variação da temperatura, a potência era proporcionalmente corrigida para atender as necessidades da planta. Portanto, por trabalhar com proporcionalidade, mesmo que os valores não sejam linearmente compatíveis, estão proporcionalmente espaçados; sem gerar, então, grandes discrepâncias no resultado final obtido.

6 CONCLUSÃO

Com o desenvolvimento do presente projeto, foi possível concluir que um sistema de controle digital, implementado utilizando o microcontrolador PIC18F4550, consegue atender com absoluta qualidade o objetivo proposto.

Também observou-se a implementação de um sistema de controle, utilizando o controlador PID, e pode-se verificar como este consegue ser sintonizado de uma maneira simplificada.

Mesmo que ocorram mudanças na planta, é possível corrigi-las apenas modificando os ganhos proporcional, integral e derivativo, não sendo necessárias grandes modificações ou reestruturações do sistema.

A utilização do TRIAC permitiu simplificar o controle de potência, pois, combinado com o cruzador de zero, permitiu controlar efetivamente a potência disponibilizada ao ebulidor.

A combinação do *software* de desenvolvimento (MPLAB X) e o compilador XC8 C se mostraram ferramentas poderosas para a implementação do *software* utilizado.

A versão do compilador atualizada de C18 para XC8 permite utilizar uma linguagem muito simplificada, a qual facilita o desenvolvimento do *software*, trazendo uma linguagem de nível intermediário, tornando a programação mais intuitiva.

É importante destacar que a utilização de equipamentos mais amigáveis ao usuário (como um potenciômetro para definir o *Set Point* e um *display* para facilitar a visualização) permite que o usuário veja o sistema de forma intuitiva, trazendo uma grande familiarização com os equipamentos aplicados na indústria.

A combinação desses sistemas permitiu uma visualização da construção genérica de um equipamento aplicado às indústrias, sendo que a diferença entre o projeto e um equipamento de utilização real é apenas a escala.

As dificuldades apresentadas durante o desenvolvimento do projeto possibilitaram a análise de casos que não foram abordados durante a graduação, como os problemas de *WindUp*, de sincronização dos tempos de processamento, e de acoplamento capacitivo entre sensores e fontes de potência.

Com isso, foi possível obter soluções de problemas similares aos encontrados em aplicações industriais (o ambiente no qual os equipamentos estão possui elevadas interferências eletromagnéticas e os controladores necessitam estar modificados para resistir a possíveis distúrbios aplicados). Os equipamentos devem, então, resistir a tais interferências

eletromagnéticas e estar blindados (fisicamente ou com equações que devem realizar a análise e remoção dos ruídos), a fim de proteger o sistema e garantir a sua perfeita operação.

Para dar continuidade a este projeto, seria possível utilizar a rede UART, já implementada no *hardware* presente na placa (conforme Figura 8), para ajustar remotamente a temperatura desejada no sistema. Assim, o operador não necessitaria se deslocar até o painel do controlador para efetuar tal ajuste (de maneira similar a uma rede supervisória).

7 REFERÊNCIAS

BRAGA, Newton C.. **SNUBBERS (DUV279).** Disponível em: <http://www.newtoncbraga.com.br/index.php/duvidas-dos-internautas/6100>. Acesso em: 11 set. 2018.

HADADE NETO, Antonio. **Técnicas Anti-Windup em estruturas de controle PID, RST e GPC.** 2005. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/102669/223414.pdf?sequence=1>. Acesso em: 11 set. 2018.

INSTRUMENTS, National (Ed.). **PID Theory Explained. 2018.** Disponível em: <http://www.ni.com/white-paper/3782/en/>. Acesso em: 11 set. 2018.

MACIEL, M.; Controle PID com aproximação Digital para utilização no PIC, 2012. Disponível em:< http://microcontrolado.com/controle-pid-no-pic/>. Acesso em: 11 set. 2018.

MICROCHIP (Ed.). **MPLAB X. Version 5.0.** Disponível em: <https://www.microchip.com/mplab/mplab-x-ide>. Acesso em: 11 set. 2018.

MICROCHIP (Ed.). **MPLAB XC8. Version 2.0.** Disponível em: <http://microchipdeveloper.com/xc8:installation>. Acesso em: 11 set. 2018.

WESCOTT, Tim. **PID without a PhD.** 2000. Disponível em: <https://www.embedded.com/design/prototyping-and-development/4211211/PID-without-a-PhD>. Acesso em: 11 set. 2018.

APÊNDICE A – PROGRAMA DESENVOLVIDO

```
////////////////////////////////////////////////////////////////////////////////////////////////////
// Projeto: Sistema de controle de temperatura PID                                                //
// Autor: Fábio Stabile Monaco                                                                    //
// Versão: 1.0                                                                                    //
// Data: 03/08/2018                                                                               //
// CPU: PIC18F4550 (DIP)                                                                          //
// IDE/Compilador/Gravador: MPLAB X, XC8 free, Bootloader Campinas                                //
// Resumo: Controle de temperatura usando um PT100                                                //
//  Sensor eletro-óptico ligado à INT0 permite medição de temperatura.                            //
//  Potenciômetro ligado à AN1 serve como referência de  temperatura.                             //
////////////////////////////////////////////////////////////////////////////////////////////////////

#pragma config PBADEN=OFF

#include <xc.h>
#include <p18f4550.h>

#define rs RE0
#define enable RE1
#define controle 0
#define dado 1
#define Kp  17
#define Ki 5
#define Kd 0.5
#define T 1
#define c1 (K+(K*(Td/T))+(K*(T/(2*Ti))))
#define c2 (-K+(K*(T/2*Ti))-(2*K*(Td/T)))
#define c3 (K*(Td/T))
void interrupt high_priority vetor_int_alta_prioridade();
void interrupt low_priority vetor_int_baixa_prioridade();
```

```
void _atraso_21usX(unsigned int y);
void buzinar();
void LCD_ini();
void LCD_setpoint();
void LCD_temp();


unsigned char Temp[16] = {' ','S','E','T',' ','P','O','I','N','T',' ',' ','=',' '};
unsigned char TempREAL[16] = {' ','T','E','M','P','E','R','A','T','U','R','A','=',' '};
unsigned int
i,j,k,_aux,_aux1,aux=0,aux1=0,lastaux1=0,aux2=0,ek,ek1,uik1,ut,ep,auxdisp,auxdisp1,help=0
,help2=0,p,i,d;
int cont =0,out=0,exit=165,I=0,P=0,D=0;
float erro=0,erro1=0,erro2=0,saida=0,saida1=0;
unsigned char amostra[2], t1uc,t2uc;

////////////////////////////////////////////////////////////////////////////
// Função: void main()                                                    //
//                                                                        //
////////////////////////////////////////////////////////////////////////////
void main(){

   //---------------------------------------------------------------
   // ADC
   ADCON2 = 0b00010110; // Left justified; 16 Tad; Fosc/64.
   ADCON1 = 0b00001101; // Vrefs=Vdd-Vss; Apenas AN0 e AN1 são analógicos.
   ADCON0 = 0b00000100; // Selecionar AN1, ADC off.

   // I/O PORTs
   // Direções conforme esquemático (Módulo sensor de temperatura).

   TRISA = 0b10101111;
   TRISB = 0b11111111;
   TRISC = 0b10111110;
```

```
   TRISD = 0b00000000;
   TRISE = 0b11111000; // Só existem RE3:RE0.

   RE2 = 1;    // 'Neutralizar' buzina e enable (LCD).
   enable = 0;

   //-----------------------------------------------------------------
   LCD_ini();  // Inicialização do LCD de 2 linhas,16 colunas, 8 bits.

   //-----------------------------------------------------------------
   // TIMER 0
   //   Usando interrupção de alta prioridade.

   T1CON = 0b10000100;
   //(7) TMR0ON: Timer0 On/Off Control bit (0: desligado).
   //(6) T08BIT: Timer0 8-bit/16-bit Control bit (0: 16bits).
   //(5) T0CS: Timer0 Clock Source Select bit (0:internal)(RA4 livre).
   //(4) T0SE: Timer0 Source Edge Select bit (0:low to high)
   //(3) PSA: Timer0 Prescaler Assignment bit (1: PS not assigned).
   //(2-0) T0PS2:T0PS0: Timer0 Prescaler Select bits (111->1:256)


   IPEN=1; // Cuidado: default é 0!!!

   //TMR0IP = 0;            // TMR0 overflow interrupt priority: high (1).
                            // INTCON2bits.TMR0IP = 1;
   TMR0IF = 0;             // TMR0 register did not overflow (must be clear in software).
INTCONbits.TMR0IF = 0;
   TMR1IF = 0;             // TMR1 register did not overflow (must be clear in software).
INTCONbits.TMR1IF = 0;
   //TMR0IE = 0;           // Enables the TMR0 overflow interrupt.
                       // INTCONbits.TMR0IE = 1;
   GIEH = 1; /// !!            // Enables all interrupts that have the priority bit
                       //   set (high priority).INTCONbits.GIEH=1;
```

```
//TMR0ON = 1; /// !!!
i=0;
// INT 0
//-----------------------------------------------------------------
INT0IE=0;
INT0IF=0;
INTEDG0=0;
//-----------------------------------------------------------------
RBPU = 0;          //Abilitar Pull-ups

RE2 = 1;
ADON = 1; /// ADC desligado!!!

t1uc = 0;
t2uc = 0;

//*****************************************************************//
// Aguardar interrupção do Timer 0 para: ler a/d e atualizar duty cycle.
while(1){

  if(i==0)
  {
   INT0IE=0;
   //ADCON0bits.CHS = 0b0000;
   CHS0=0;
   GO = 1;
   while(DONE==1);
   aux =(ADRESH);
   //ADCON0bits.CHS = 0b0001;
   CHS0=1;
   GO = 1;
   while(DONE==1);
   aux1 =(ADRESH);
   LCD_setpoint();
```

```
        LCD_temp();
        erro=0;
        erro1=0;
        erro2=0;
        saida=0;
        saida1=0;
       }
        if(RB2 == 0)
       {
       INT0IE=1;
       i=1;
       }
     }

}
// main()
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///
//*********************************************************************************
*//
// FUNÇÕES     FUNÇÕES     FUNÇÕES     FUNÇÕES     FUNÇÕES     FUNÇÕES
//
//*********************************************************************************
**//
////////////////////////////////////////////////////////////////////////////////////////////////////
// Função: void _atraso_92usX(unsigned int y)                                    //
// Entrada:-                                                                     //
// Saída: -                                                                      //
// Descrição: Realizar atraso em múltiplos de 92 us para o controle do           //
//angulo de diparo do TRIAC                                                     //
//   48 MHz / 4 = 12 MHz. 12 MHz / 256 =  46875 Hz => T = 21,33 us.             //
//////////////////////////////////////////////////////////////////x             ////////////////////////////////////

void _atraso_92usX(unsigned int y){
```

```
   unsigned int j=0;
   j=0;
  while(y != j)
  {
   TMR1IF =0;
   TMR1H = 0xFb;
   TMR1L = 0xA8;
   TMR1ON = 1;
   j++;
   while(TMR1IF == 0);
   TMR1ON = 0;
  }
}


//////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////
// Função: void LCD_setpoint()                                                                        //
// Entrada:-                                                                                          //
// Saída: -                                                                                           //
// Descrição: Atualiza os dados referente ao Set  Point atual no display                              //
////////////////////////////////////////////////////////////////////////////////////////////////////////
void LCD_setpoint()
{
   _aux=((aux*0b01100100)/0b11111111);
      cont = 0;
      while(_aux >= 10)
      {
         cont++;
         _aux = (_aux - 10);
      }
      amostra[0] = (char)(cont + 0x30);
```

```
        cont = 0;
        while(_aux >= 1)
        {
           cont++;
           _aux = (_aux - 1);
        }

        amostra[1] = (char)(cont + 0x30);

        cont = 0;

        Temp[14]= amostra[0];
        Temp[15] = amostra[1];
        rs = controle;
        PORTD = (0x80+ t1uc);   // Linhas 1, coluna 1.
        enable = 1;
        _atraso_92usX(2);
        enable = 0;
        _atraso_92usX(2);
        rs = dado;
        PORTD = Temp[t1uc];
        enable = 1;
        _atraso_92usX(2);
        enable = 0;
        _atraso_92usX(2);
        if(t1uc++>14){
              t1uc = 13;

           }
}
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
// Função: void LCD_temp()                                                    //
// Entrada:-                                                                  //
```

```
// Saída: -                                                                    //
// Descrição:Atualiza os dados referente a temperatura atual no display       //
////////////////////////////////////////////////////////////////////////////////////////////////////////
void LCD_temp()
{
      _aux1=((aux1*0b01100100)/0b11111111);
       cont = 0;
       while(_aux1 >= 10)
       {
          cont++;
          _aux1 = (_aux1 - 10);
       }
       amostra[0] = (char)(cont + 0x30);

       cont = 0;
       while(_aux1 >= 1)
       {
          cont++;
          _aux1 = (_aux1 - 1);
       }

       amostra[1] = (char)(cont + 0x30);

       cont = 0;




       TempREAL[14]= amostra[0];
       TempREAL[15] = amostra[1];

       rs = controle;
        PORTD = (0xC0 + t2uc);   // Linhas 2, coluna 1.
       enable = 1;
       //for(j=0;j<5000;j++);
```

```
        _atraso_92usX(2);
        enable = 0;
        //for(j=0;j<10000;j++);
        _atraso_92usX(2);

        rs = dado;
        PORTD = TempREAL[t2uc];
        enable = 1;
        //for(j=0;j<5000;j++);
        _atraso_92usX(2);
        enable = 0;
       // for(j=0;j<10000;j++);
        _atraso_92usX(2);
        //for(j=0;j<60000;j++);
       // for(j=0;j<60000;j++);

        if(t2uc++>14){
             t2uc = 13;
           }
}
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////
// Função: void LCD_ini()                                                                             //
// Entrada:-                                                                                          //
// Saída: -                                                                                           //
// Descrição: Inicializa o display                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////
void LCD_ini(){
   unsigned int j;

   rs = controle;

   PORTD = 0x38;
```

```
    enable = 1;
    for(j=0;j<5000;j++);
    enable = 0;
    for(j=0;j<10000;j++);

    PORTD = 0x0C;
    enable = 1;
    for(j=0;j<5000;j++);
    enable = 0;
    for(j=0;j<10000;j++);

    PORTD = 0x06;
    enable = 1;
    for(j=0;j<5000;j++);
    enable = 0;
    for(j=0;j<10000;j++);

    PORTD = 0x01;
    enable = 1;
    for(j=0;j<5000;j++);


    enable = 0;
    for(j=0;j<10000;j++);
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
//*************************************************************************
**//
// INTERRUPÇÕES    INTERRUPÇÕES       INTERRUPÇÕES       INTERRUPÇÕES
//
//*************************************************************************
**//
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///
// Função: void LCD_ini()
//
// Entrada:-
//
// Saída: -
//
// Descrição: a cada pulso recebido do cruzador de zero inicializa a rotina  do controlador PID
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
void interrupt high_priority vetor_int_alta_prioridade()
{

   if(( INT0IF==1) && (INT0IE==1)) {
      _atraso_92usX(exit);
      RC0 = 1;
      _atraso_92usX(1);
      RC0=0;
      help++;
      help2++;
      if(help==60)
      {
      help =0;
      //ADCON0bits.CHS = 0b0001;
      CHS0=0;
      GO = 1;
      while(DONE==1);
      aux =(ADRESH);
      CHS0=1;
      GO = 1;
```

```
  while(DONE==1);
  aux1 =(ADRESH);
  if(help2 == 120)
  {
  for(k=0;k<5;k++)
  {
     //LCD_temp();
  }
  help2=0;
  }

erro= (float) aux-aux1;

if(erro>0)
{
  P = erro * Kp;
  I += (Ki* erro*T);


  D = (((aux1 - lastaux1)*Kd)/T);

  out = (P + I + D)/100;
  if(out>=84)
  {
     out = 83;
  }
  if(erro < 3)
  {
     I=0;
     P=0;
     D=0;
     out=0;
  }
  if(out < 5)
```

```
        {
           out = 6;
        }



        exit = 90-out;

        }
        }
        INT0IF = 0; // INTCONbits.TMR0IF = 0;

        ///s WriteTimer0(0xEC78);    //Please use HEX. Decimal don't work.
        GIEH =1;    // INTCONbits.GIEH=1
        if(RB3 == 0)
           {
            INT0IE=0;
            i=0;
           }

        }
    }
////////////////////////////////////////////////////////////////////////////////
```