CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
DE MINAS GERAIS - CAMPUS-IV / ARAXÁ

Curso de Engenharia de Automação Industrial

Controle proporcional/integral por software da temperatura da água do chuveiro elétrico com temporização

Alessandro Hermógenes da Silva

Araxá

2012

Alessandro Hermógenes da Silva

Controle proporcional/integral por software da temperatura da água do chuveiro elétrico com temporização

Projeto de pesquisa apresentado ao Curso de Engenharia de Automação Industrial do CEFET-MG, como requisito para a obtenção do título de graduação.

Orientador: Professor Henrique José Avelar

Araxá

2012

MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS-IV - ARAXÁ
COORDENAÇÃO DE ENGENHARIA DE AUTOMAÇÃO INDUSTRIAL

TRABALHO DE CONCLUSÃO DE CURSO

ATA DE DEFESA

ATA da defesa de Trabalho de Conclusão de Curso de Engenharia de Automação Industrial do(a) aluno(a) _____.

Ás _____ horas do dia ____ do mês de _____ de _____, reuniu-se no Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG/CAMPUS-IV - ARAXÁ, a Comissão Examinadora de Trabalho de Conclusão de Curso para julgar, na condição de exame final e como parte dos requisitos para a obtenção do Título de Engenheiro(a) de Automação Industrial o trabalho intitulado_____

_____

_____

Abrindo a sessão, o(a) Presidente(a) da Comissão, Prof(a). _____

_____, após dar ciência aos presentes sobre o teor das Normas Regulamentares para Trabalhos de Conclusão de Cursos do CEFET-MG, passou a palavra ao(a) aluno(a) para apresentação de seu trabalho. Seguiu-se a arguição pelos examinadores, com a respectiva defesa do(a) candidato(a). Logo após, a Comissão se reuniu, sem a presença do(a) aluno(a) e do público, para julgamento e expedição do resultado final. Foram atribuídas indicações de nota entre 0 e 100 e conceito Aprovado = A ou Reprovado = R.

Prof(a). _____ (Orientador/a) conceito: ____ nota: ____/100

Prof(a). _____ conceito: ____ nota: ____/100

Prof(a). _____ conceito: ____ nota: ____/100

Pela média das notas registradas acima, o(a) aluno(a) foi considerado(a) _____, com nota final igual a _____/100, resultado informado publicamente pelo Presidente da Comissão. O(a) aluno(a) abaixo assinado declara que o trabalho ora identificado e apresentado é da sua autoria material e intelectual, excetuando eventuais elementos tais como passagens de texto, citações, figuras e datas, desde que as mesmas tenham identificadas claramente a fonte original, explicitando as autorizações obtidas dos respectivos autores, quando necessárias. Declara ainda, neste âmbito, não estar violando quaisquer direitos de terceiros.

_____	_____	_____
Nome do Aluno(a)	Assinatura	Data

Nada mais havendo a registrar eu, Presidente(a) da Comissão, lavro a presente ATA, que será assinada por mim e pelos demais membros da Banca. Araxá, _____ de _____ de 20____.

Dedico à minha esposa Andrea,
ao meu filho Felipe
e à minha mãe Belma.

AGRADECIMENTOS

Agradeço a Deus sobre todas das coisas, pois é ele quem nos dá forças para seguir em frente e trilhar novos caminhos na busca do sucesso, tanto profissional como das nossas realizações pessoais.

Ao professor Henrique pela dedicação e conhecimento que tem nos dado no decorrer deste longo curso.

Aos laboratoristas de eletrônica Gilberto e José Afonso que sempre me ajudaram com boa vontade e atenção nas dúvidas pertinentes à execução do projeto.

Não sei o que possa parecer aos olhos do mundo, mas aos meus pareço apenas ter sido como um menino brincando à beira-mar, divertindo-me com o fato de encontrar de vez em quando um seixo mais liso ou uma concha mais bonita que o normal, enquanto o grande oceano da verdade permanece completamente por descobrir à minha frente.

Isaac Newton

RESUMO

Tendo em vista o grande avanço que a eletrônica e a informática têm-nos oferecido e pela grande necessidade atual de um desenvolvimento sustentável sem desperdícios, elaboramos um projeto para controle de temperatura da água e do tempo de uso de um chuveiro elétrico doméstico, através da utilização de um número mínimo de componentes eletrônicos de baixo custo, visando a economia de água e de energia elétrica. O controlador foi elaborado com a utilização do microcontrolador *PIC18F4550* conectado diretamente ao módulo de potência, constituído apenas por TRIAC e DIAC, através de uma de suas portas I/O e fazendo uso de um controle proporcional/integral programado em linguagem C.

Palavras chave: controle de temperatura, economia de energia, microcontrolador, *PIC*.

ABSTRACT

Given the great progress that electronics and computing have offered us and the great current need for sustainable development without waste, designed a project to control water temperature and time using a domestic electric shower, using a minimum number of low cost electronic components, aimed at saving water and electricity. The controller was designed using the *PIC18F4550* microcontroller connected directly to the power module, constituted only by TRIAC and DIAC, through one of its I/O ports and making use of a control proportional/integral programmed in C language.

Keywords: temperature control, energy saving, microcontroller, PIC.

LISTA DE FIGURAS

LISTA DE TABELAS

SIGLAS E ABREVIATURAS

AD - Analógico-Digital

CISC - Complex Instruction Set Computer (Computador com Conjunto Complexo de Instruções)

CPU - Central Processing Unit (Unidade Central de Processamento)

DIAC - Diode for Alternating Current (Diodo para Corrente Alternada)

EC - External Oscilator (Oscilador Externo)

EEPROM - Electrically Erasable Programmable Read Only Memory (Memória Somente Leitura Apagável Eletricamente)

EPROM - Erasable Programmable Read Only Memory (Memória Somente Leitura Apagável)

HS - High Speed (Alta velocidade)

I/O - Input/Output (Entrada/Saída)

intRC - Internal Resistor and Capacitor (Resistor e Capacitor internos)

ISR - Interrupt Service Routine (Rotina de serviço de Interrupção)

LCD - Liquid Crystal Display (Visor de Cristal Líquido)

MCLR - Master Clear Reset

MCU - Microcontroller Unit (Unidade Microcontroladora)

PIC - Programmable Interface Controller (Controlador de Interface Programável)

PROM - Programmable Read Only Memory (Memória Somente Leitura Programável)

PV - Process Variable (Variável de Processo)

PWM - Pulse Width Modulation (Modulação por Largura de Pulso)

RAM - Ramdom Acess Memory (Memória de Acesso Aleatório)

RISC - Reduced Instruction Set Computer (Computador com Conjunto de Instruções Reduzido)

ROM - Read Only Memory (Memória Somente Leitura)

SCR - Silicon Controlled Rectifier (Retificador Controlado de Silício)

SP - Set-Point (Valor desejado)

Tamb - Temperatura ambiente

TRIAC - Triode for Alternating Current (Triodo para Corrente Alternada)

USB - Universal Serial Bus (Barramento Serial Universal)

XT - Xtal

LISTA DE SÍMBOLOS

°C - grau Celsius

μ - micro (10^{-6})

K - quilo (10^{3})

M - mega (10^{6})

m - mili (10^{-3})

Δ - delta

SUMÁRIO

1 INTRODUÇÃO

O controle automático de temperatura e temporização para chuveiro elétrico apresenta-se como uma opção efetiva com relação ao consumo excessivo de energia elétrica e água no banho. Por meio desta pesquisa, pretendemos criar uma forma eficaz de economia de energia elétrica e de água potável no uso do chuveiro elétrico.

Vivemos um momento singular no planeta. Por um lado, o avanço da indústria, da tecnologia e da velocidade na transmissão de informações. Por outro lado, o esgotamento de recursos não renováveis em função do aumento e aceleração da produção. Crescer conciliando progresso e sustentabilidade é o nosso grande desafio atual.

A produção de energia através da força da água, processo iniciado com a revolução industrial, propiciou ao ser humano uma infinidade de benefícios, dentre eles, o banho quente por meio do chuveiro elétrico. Paralelo a isto, vieram os problemas: impactos ambientais causados pelo alagamento de áreas para produção de energia elétrica, comprometendo flora e fauna locais e a liberação de gás metano, um dos vilões do efeito estufa, na decomposição de matéria orgânica realizada pelas bactérias nas represas hidroelétricas.

Será possível um desenvolvimento, mantendo o conforto e, ao mesmo tempo, amenizar nossos impactos negativos sobre a biosfera?

As pessoas, geralmente, preferem um banho quente e demorado. Porém, no que diz respeito à economia, a temperatura da água e tempo de uso do chuveiro são dois fatores que ampliam demasiadamente o consumo de energia elétrica residencial.

Analisando o primeiro fator, podemos observar que em um banho quente acima do ideal, ou seja, com uma temperatura da água em níveis acima de um valor ótimo,

estaremos desperdiçando energia elétrica e, abaixo deste valor ótimo, perderemos o conforto térmico desejado.

Em relação ao segundo fator, além do consumo elevado de água potável, que é uma preocupação ambiental crescente nos últimos anos, com um banho demorado também temos um considerável aumento do consumo de energia elétrica impulsionado pela alta potência necessária aos chuveiros elétricos residenciais para aquecer a água nesses valores desejáveis.

Com a facilidade de informações disponíveis, através dos meios de comunicação que temos hoje, fica evidente que as pessoas sabem da necessidade de economizar este recurso natural tão precioso e reduzir o gasto de energia elétrica, de modo a garantir a sustentabilidade no planeta. No entanto, entre teoria e prática, encontramos uma distância preenchida pela necessidade excessiva de conforto e comodismo.

Sendo assim, e com o grande avanço que a eletrônica vem nos oferecendo através de microcontroladores, sensores de baixo custo, atuadores eletrônicos etc., podemos elaborar circuitos de controle para que o uso da energia elétrica seja otimizado em um chuveiro elétrico.

A presente proposta, portanto, utilizou o microcontrolador *PIC18F4550* em conjunto com um sensor linear de temperatura, o *LM35* e uma chave de corrente alternada denominada *TRIAC*, de uma forma que o circuito eletrônico do controlador ficou reduzido em número de componentes pelo uso mais exigente dos recursos do *PIC*, reduzindo, portanto, a quantidade de circuitos externos.

Acreditamos que, com a utilização deste controlador, reduziremos o consumo de energia elétrica e de água nas residências, pois, sendo o banho um hábito diário das pessoas em muito contribui para o desperdício. Assim daremos a nossa parcela de contribuição para um melhor aproveitamento dos recursos naturais do planeta, que é dever de todos.

Deste modo, com o intuito de demonstrar a funcionalidade do projeto aqui apresentado, montamos um protótipo do circuito controlador aplicado a um chuveiro residencial, no qual foram feitas adaptações para colocação do sensor de temperatura e da chave de ativação do circuito. Porém, antes disso, toda a programação do circuito de comando, circuito de potência e circuitos sensores, foi avaliada através de simulações em softwares e através de circuitos reais montados em *protoboard*.

Para a determinação dos valores, que foram usados como parâmetros do controle automático com base na temperatura da água ambiente, foi feita uma pesquisa de campo, onde determinamos estatisticamente quais os valores ideais de temperatura e tempo de banho em ocasiões em que a exigência por conforto e economia se apresentam mais necessários. Além disso, a pesquisa serviu para averiguar a tese de que as pessoas reconhecem a necessidade de economia de água e energia elétrica. Sendo assim, para analisar a aceitação desta proposta, a pesquisa de campo foi realizada entre jovens e adultos, com sua tabulação conforme APÊNDICE A.

O trabalho se desenvolverá em quatro capítulos. Primeiramente, no capítulo 2 será tratada a fundamentação teórica necessária ao entendimento dos recursos utilizados no projeto, bem como a escolha do tipo de controle a ser empregado.

O capítulo 3 mostrará o desenvolvimento do projeto no que diz respeito à implementação do protótipo real, para demonstração do sistema de controle proposto. Também serão feitos os testes finais e a análise dos resultados obtidos.

O capítulo 4 apresentará uma análise conclusiva de todos os objetivos propostos por esta pesquisa para verificação da viabilidade de construção de um controlador conforme descrito.

Finalmente, o capítulo 5 apresentará sugestões para projetos futuros que poderão melhorar o seu desempenho e ampliar sua aplicação para ambientes que já tenham outro recurso de aquecimento de água.

2 FUNDAMENTOS TEÓRICOS

2.1 O funcionamento do controlador

Tendo como princípio a economia de energia elétrica e de água potável, o protótipo do controlador tem por objetivo manter a temperatura da água do chuveiro em um valor pré-estabelecido, sem que seja necessária a alteração do nível de vazão de água mediante atuação na torneira de alimentação. Desta forma, e mantendo esta vazão em um nível baixo o suficiente para realizar o banho, será usada uma menor parcela de energia elétrica, já que a massa de água a ser aquecida é proporcional à vazão mássica em um dado intervalo de tempo, o que também leva a uma economia de água. Além disso, com uma temporização ativada, estaremos limitando o tempo de banho, que é o principal fator de consumo excessivo de energia elétrica.

O protótipo do controlador apresenta dois modos de funcionamento: o modo automático e o modo manual.

O primeiro modo é baseado em pesquisa estatística que determinou o valor ideal em que a temperatura da água deve ser controlada, tomando como base a temperatura da água no ambiente onde está instalado o chuveiro. Para isso, foi utilizado o sensor linear de temperatura *LM35* ligado a uma das portas analógicas do microcontrolador *PIC18F4550*, conforme suas especificações de aplicações típicas, constantes em seu *data-sheet,* para melhorar sua tolerância a cargas capacitivas provenientes de fontes eletromagnéticas, como relés, transmissores de rádio, motores a escovas e transientes de SCRs, “onde sua fiação pode agir como uma antena de recepção e suas junções internas podem atuar como retificadores”, conforme Nacional Semiconductor[1]. Neste modo, o valor da temperatura controlada aparece sob a indicação "T.Autom." no *LCD*.

[1] Nacional Semiconductor, 2000, p.7.

Já no modo manual o usuário poderá ajustar a temperatura desejada a ser controlada através de um potenciômetro (resistor variável), e a mesma será mostrada no *LCD* sob a indicação "T.Manual".

Em ambos os modos, assim que o usuário abrir a torneira para iniciar o banho, a pressão que a água exerce dentro do chuveiro, além de ligar o seu circuito elétrico convencional, acionará também uma microchave liga-desliga devidamente posicionada, que indicará ao microcontrolador *PIC18F4550* o acionamento do chuveiro (presença de água) através de uma de suas interrupções externas. Isto iniciará o controle para a temperatura ajustada e, assim que for atingida, a temporização do banho também será iniciada, caso esteja ativada. Nesta situação, o usuário poderá ativar uma pausa na temporização bastando fechar a torneira e, após algum tempo, reabri-la para terminar o banho, sem prejuízo na temporização. Caso o usuário já tenha terminado o banho enquanto a pausa estiver ativada, será cronometrado um tempo para que o circuito desligue automaticamente e inicie o modo de economia de energia do *PIC* (modo *sleep*).

Nas duas situações em que o banho termina, seja por tempo esgotado ou por desligamento automático, além de iniciar o modo *sleep*, o chuveiro somente poderá ser usado novamente se o circuito de controle for reiniciado através de um interruptor com chave, mostrado na FIG. 1, em que a chave pode ser retirada. Isto é muito útil para que os pais façam um controle de uso do chuveiro pelos filhos, por exemplo.

Figura 1 - Interruptor elétrico com chave
Fonte: www.google.com.br.

Para maior comodidade do usuário, este interruptor deverá ser usado também para ativação/desativação da temporização. Sendo assim, quando acionado, será mostrada no *LCD* a mensagem "T." assim que a temperatura ajustada para o

controle for atingida, indicando que a temporização está ativada. Caso contrário, não será mostrada esta mensagem no *LCD*.

O circuito de controle também entra no modo de economia de energia (modo *sleep*) assim que for atingido um tempo pré-estabelecido, na situação em que o mesmo foi energizado, mas que a chave de início não foi acionada (chuveiro não está sendo usado). Isto é particularmente útil quando a energia retornar após um eventual desligamento da rede elétrica.

Este controlador poderá ser utilizado em residências que possuem um aquecimento solar de água e que será útil em duas situações distintas: primeiro quando faltar água quente devido ao tempo nublado, incapacitando o reservatório de suprir a demanda; segundo para aquecer a água ainda fria, que esteve parada na tubulação, até a chegada da água quente do aquecedor solar, economizando uma parcela de água que seria descartada por não atender à temperatura desejada pelo usuário. Neste caso, quando a água quente do aquecedor solar atingir o chuveiro e já estiver na temperatura desejada, o controlador não mais aquecerá a resistência elétrica e, portanto, não consumirá energia elétrica. Porém, caso a água do aquecedor estiver mais quente que o desejado, o usuário terá de atuar na torneira de água fria para fazer o controle. Isto pode ser resolvido utilizando uma válvula de controle de vazão para a água fria, que seria uma implementação futura para o projeto em questão.

2.2 A programação do *PIC*

Neste item introduziremos alguns conceitos teóricos e configurações internas para realizar a programação do *PIC18F4550* dentro das especificações técnicas fornecidas pelo seu fabricante, a *MICROCHIP*®.

2.2.1 Os microcontroladores

Os microcontroladores (*MCU* ou *µC*) são dispositivos que possuem internamente uma *CPU* (*Central Process Unit* - Unidade Central de Processamento), memória de dados, memória de programa e periféricos (portas de entrada e saída, conversores A/D e D/A, temporizadores, interrupções, módulos de comunicação etc.) e estão presentes na maioria dos aparelhos eletrônicos modernos. Fazemos uso destes aparelhos (celulares, MP3s, impressoras, TVs e muitos outros) diariamente.

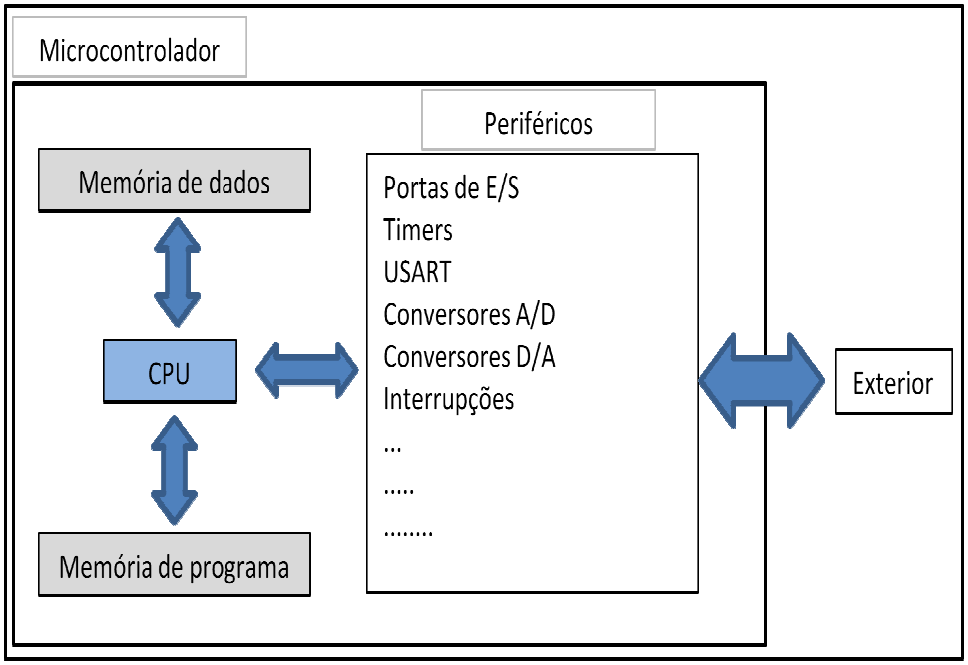

Figura 2 - Diagrama simplificado de um microcontrolador
Fonte: MIYADAIRA, 2010, p.21.

Estes dispositivos são classificados em duas arquiteturas de construção: *Harvard* e *Von-Neumann*. A primeira é caracterizada por conter dois barramentos distintos para as memórias de dados e de programa, possuindo, portanto, um melhor fluxo de informação em comparação à outra arquitetura, que utiliza apenas um barramento.

A memória de programa (memória em que o programador armazena o seu *firmware*, ou seja, o programa) usada nos *MCUs* são não-voláteis, ou seja, que não perdem o conteúdo quando o circuito não está alimentado eletricamente, e em geral são dos tipos:

- *ROM* (*Read Only Memory*) - é uma memória que possui um conteúdo gravado pelo fabricante e que não pode ser apagado. Esta armazena as instruções do Microcontrolador e pode ser apenas lida.
- *EPROM* (*Erasable Programmable Read Only Memory*) - memória que pode ter o conteúdo apagado pelo usuário através de uma janela de quartzo, expondo-a a luz ultravioleta e, depois, reprogramada.
- *EEPROM* (*Electrically Erasable Programmable Read Only Memory*) - memória que pode ter seu conteúdo apagado eletricamente. Nesta operação os *bytes* são manipulados um por vez.
- *OTP* (*One Time Programmable*) ou *PROM* - memória que pode ser gravada apenas uma vez. Possui menor custo, mas dever ser usada somente quando o programa não necessitar mais de alterações.
- *FLASH* - é um tipo especial da *EEPROM*. Sua operação de apagamento é feita em blocos, portanto mais rápida. É o tipo de memória usada na fabricação dos *pendrives*, muito comuns hoje em dia.

Já a memória de dados é volátil e armazena as variáveis e constantes do sistema durante o seu funcionamento. Ela perde seu conteúdo quanto o circuito for desligado. É definida como *RAM* (*Random Access Memory*).

Um conjunto importante presente em todos os Microcontroladores são os pinos *I/O* (*input/output*), que são os pinos de entrada e saída, respectivamente, responsáveis pela comunicação com o mundo exterior. É através destes pinos que o *MCU* pode

receber dados (de sensores, por exemplo) e enviar dados (para acionar um motor, ou acender uma lâmpada, por exemplo). Eles são configuráveis via software para ter a função de entrada ou de saída.

Outro fator que deve ser levado em consideração nos *MCUs*, é a velocidade de processamento, que está relacionada diretamente com a frequência de *clock* estabelecida durante a programação e através de um circuito oscilador interno ou externo. O uso de circuito oscilador interno é comum quando não há necessidade de precisão no valor do *clock*, mas de acordo com MIYADAIRA[2] "se uma comunicação serial for necessária, é desejável que um cristal de quartzo seja empregado para garantir a fidelidade do sinal de *clock*, resultando maior confiabilidade na transmissão/recepção dos dados."

Devemos salientar que quanto maior for o *clock* escolhido para a frequência de trabalho do *MCU*, maior será o seu consumo de energia.

2.2.2 O microcontrolador *PIC18F4550*

Para o projeto do controlador apresentado foi utilizado o microcontrolador *PIC18F4550*, pertencente à família 18F dos dispositivos fabricados pela *MICROCHIP*. Este microcontrolador é baseado na arquitetura *Harvard* e seu conjunto de instruções é do tipo *RISC* (*Reduced Instruction Set Computer* – Computador com conjunto de instruções reduzido), que, ao contrário da tecnologia *CISC* (*Complex Instruction Set Computer* – Computador com conjunto complexo de instruções), possui circuitos mais simples e pode trabalhar em frequências maiores, segundo Guia do Hardware[3]. Esta unidade, por exemplo, pode trabalhar em frequências de até 48MHz (executando 12 milhões de instruções por segundo).

O microcontrolador *PIC18F4550* é um dispositivo de 8 bits com memória de programa (tipo *FLASH*) de 32Kbytes e memória *RAM* de 2Kbytes. Possui *clock*

[2] MIYADAIRA, 2010, p.23.
[3] Guia do Hardware, 2007.

interno de 8MHz, mas, para o projeto, foi utilizado um cristal oscilador externo de 20MHz. Possui 40 pinos dos quais temos as portas de E/S (entrada e saída), onde fazemos toda a comunicação com os circuitos que integram o controlador (circuito de potência, *LCD*, chaves, *LEDs* indicadores, sensores de temperatura etc.).

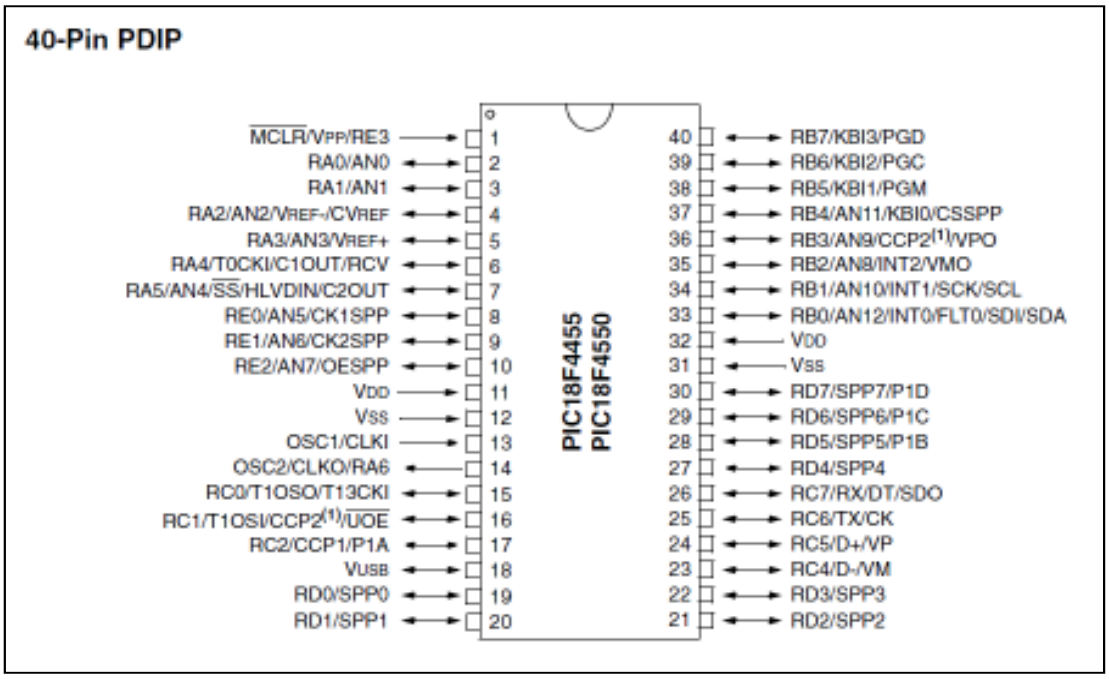

Figura 3 - Pinagem do *MCU PIC18F4550*
Fonte: Nacional Semiconductor, 2000, p.4.

2.2.3 Frequência de trabalho do *PIC18F4550*

Este microcontrolador é dotado da tecnologia *PIPELINE*, em que uma instrução é executada ao mesmo tempo em que outra é carregada no registro de instrução (*IR*) para a próxima execução. Isto garante que uma instrução seja executada em um único ciclo de máquina e aumente a velocidade de processamento do mesmo.

Ciclo de máquina é o tempo que o *PIC* gasta para executar uma instrução e é determinado pela frequência do *clock* (F_{osc}):

$$F_{maq} = \frac{F_{osc}}{4}$$

(1)

Assim, para um cristal oscilador de 20MHz, teremos uma frequência de máquina (F_{maq}) de 5MHz e, consequentemente, um ciclo de máquina (T_{maq}) de 0,2μs. Isto corresponde a 5 milhões de instruções por segundo.

Porém, para uma configuração completa da frequência em que o *PIC* irá trabalhar temos que levar em consideração alguns fatores: oscilador utilizado, funções do oscilador para a USB, além da escolha da própria frequência de máquina a estabelecer.

Primeiramente, devemos escolher qual o tipo de oscilador utilizar: se externo ou interno. Para o externo existem três modos de operação, que são os modos HS, XT e EC. O modo HS (*High Speed*) possui maior nível de condução e é designado para cristais com frequências superiores a 4MHz, porém apresenta maior consumo de energia. O modo XT (*Xtal*) é designado para cristais de 1 a 4MHz e com consumo moderado de energia. E o modo EC (*External Oscilator*) é para uso com um circuito oscilador externo. Já para uso do oscilador tipo interno, temos o modo intRC (*Internal Resistor and Capacitor*), que é uma implementação física de um circuito RC dentro do *PIC,* presente em alguns modelos e que fornece uma frequência de 31KHz, e outra fonte interna que fornece um *clock* de 8MHz. Neste projeto foi escolhido o oscilador externo, através de um cristal oscilador de 20MHz. Isto porque ele irá trabalhar com comunicação serial via interface *USB*.

Além disso, quando é utilizada a comunicação *USB*, devemos satisfazer as frequências impostas por esta interface, que são de 48MHz para *full-speed* ou 6MHz para *low-speed*. Isto pode ser conseguido através de um cristal de 48MHz como oscilador primário ou através da ativação do bloco de 96MHz PLL dividido por dois para se conseguir, por exemplo, a frequência *full-speed*. Para o primeiro caso, basta definir o modo para HS (modo HS com PLL desabilitado) e escolher a frequência de oscilação (clock do *PIC* ou oscilador primário ou F_{osc}), através da diretiva de configuração de parâmetros do *PIC, FUSE CPUDIV1, CPUDIV2, CPUDIV3 ou CPUDIV4,* (F_{osc} dividida por 1, 2, 3 ou 4, respectivamente), que finalmente será dividida por quatro, obtendo-se a frequência interna de trabalho do *PIC* (F_{maq}), de acordo com a equação 1, citada anteriormente. Para o segundo caso, definimos o

modo para HSPLL (modo HS com PLL habilitado), o *FUSE PLL* de acordo com o cristal utilizado para obtermos os 4MHz, gerador dos 96MHz divididos por 2 pelo *FUSE USBDIV* para a USB (48MHz), e o *FUSE CPUDIV1, CPUDIV2, CPUDIV3 ou CPUDIV4,* (F_{osc} proveniente do bloco PLL dividida por 2, 3, 4 ou 6, respectivamente) de acordo com a frequência de máquina desejada, que poderá ser de 12, 8, 6 ou 4MHz, da mesma forma que no primeiro caso. Devemos ficar atentos ao *FUSE USBDIV* que determina de onde vem o *clock* para a *USB*. Caso esteja em 1 (divisão por um), o clock será proveniente do oscilador primário (cristal) que deverá ser de 48MHz para trabalhar em *full-speed*. Caso esteja em 2 (divisão por 2), o *clock* será proveniente do bloco de 96MHz PLL dividido por 2. O diagrama da FIG. 4 mostra o caminho seguido para a definição da configuração do oscilador para o *PIC18F4550*.

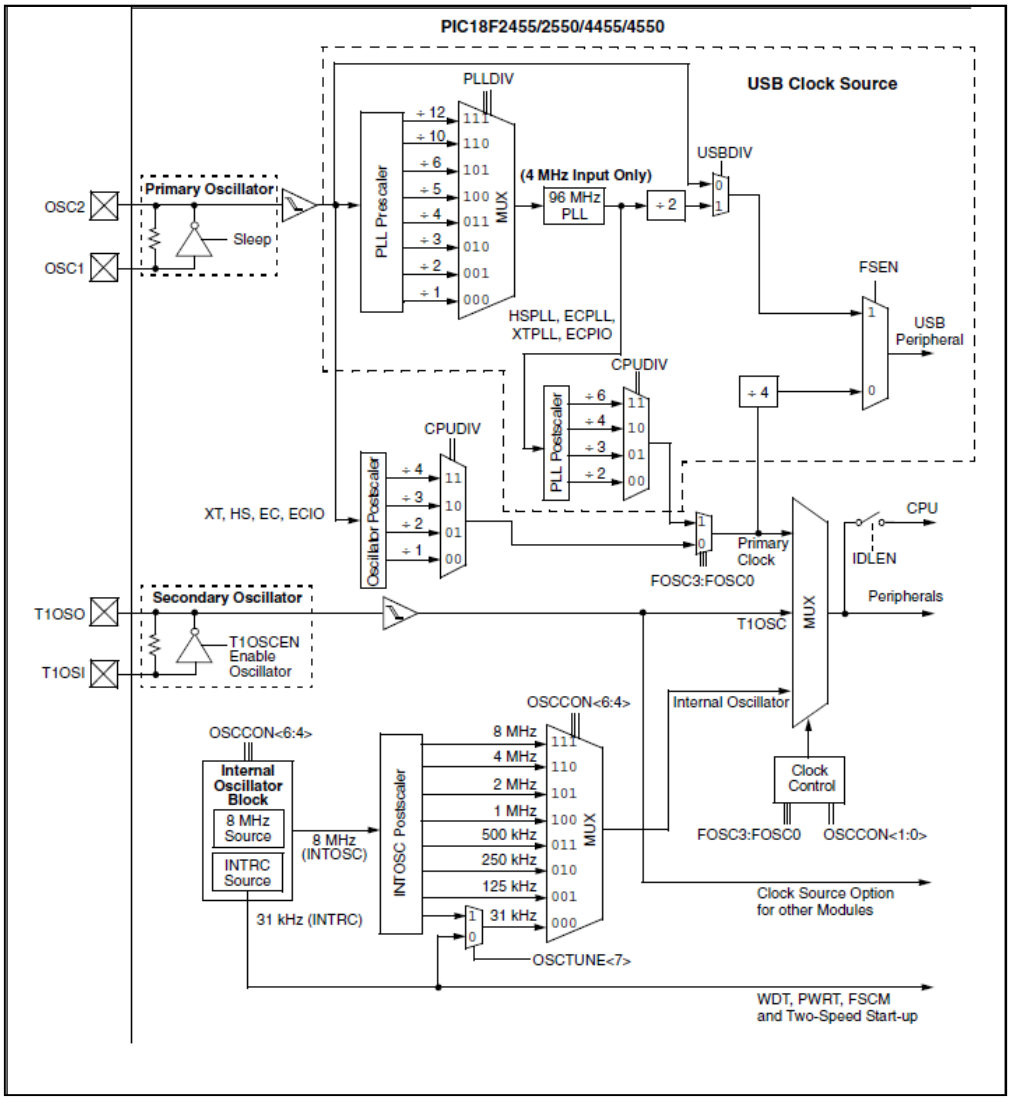

Figura 4 - Diagrama de *clock* do microcontrolador *PIC18F4550*
Fonte: Nacional Semiconductor, 2000, p.4.

Para que o cristal funcione adequadamente, é recomendada pelo fabricante a utilização de um par de capacitores com valores de acordo com o modo e a frequência do cristal utilizado (veja TAB. 1). Sendo assim, para um cristal de 20MHz no modo HS, devemos utilizar dois capacitores de 15pF (pico farad) cada um, conforme esquema da FIG. 5.

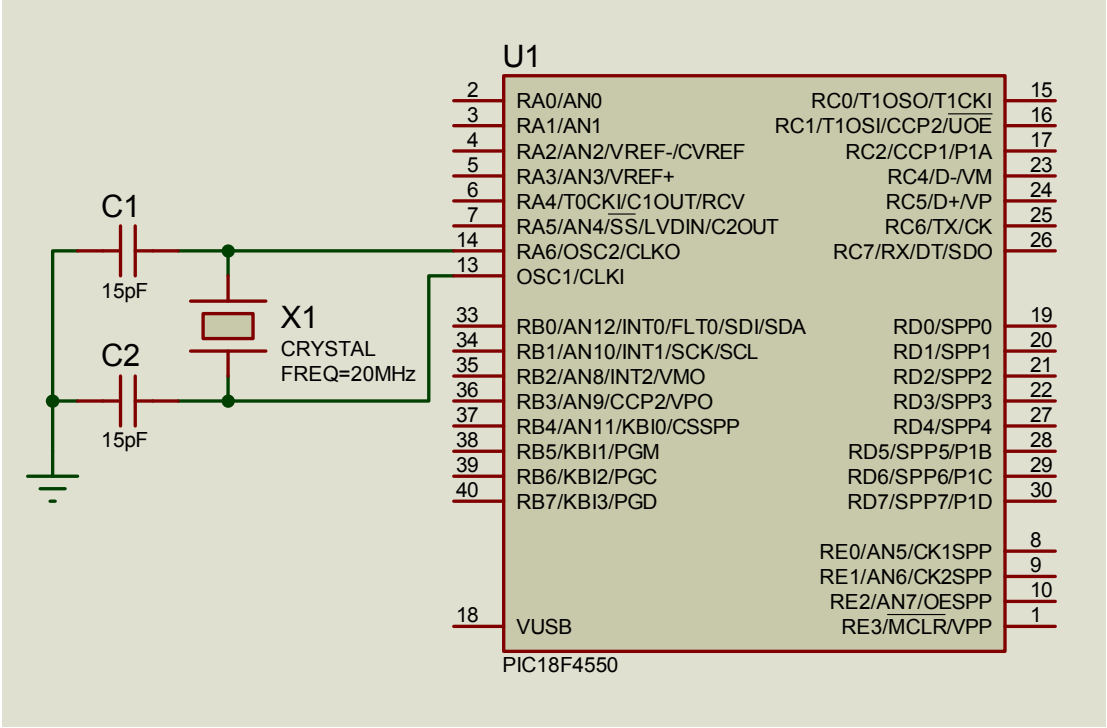

Figura 5 - Ligação do Cristal de 20MHz para modo HS
Fonte: do autor.

Tabela 1 - Cristal x Capacitor

Oscilador Tipo	Frequência do cristal	Capacitores (C1 e C2)
XT	4MHz	27pF
HS	4MHz	27 pF
	8MHz	22 pF
	20MHz	15 pF

Fonte: MIYADAIRA, 2010, p.155.

A seguir mostramos as linhas de comando utilizadas na programação para definição dos *FUSES* que determinarão a frequência de trabalho do *PIC* utilizado no projeto.

Lembrando que a programação foi realizada utilizando o compilador *PIC-C* da *CCS Inc*.

```
#FUSES HSPLL          //High Speed Crystal/Resonator with PLL enabled
#FUSES PLL5           //Divide By 5(20MHz oscillator input)
#FUSES CPUDIV1        //No System Clock Postscaler
#FUSES USBDIV         //USB clock source comes from PLL divide by 2
```

Desta forma, como a frequência do clock (F_{osc}) ficou definida em 48MHz com a ativação do bloco PLL, a frequência de máquina (F_{maq}) para o projeto é de 12MHz. Assim, o *PIC* executa 12 milhões de instruções por segundo.

2.2.4 Gerenciamento de energia

O *MCU PIC18F4550*, assim como outros modelos de microcontroladores, podem trabalhar em três estados de gerenciamento de energia. O modo *RUN* é padrão do *PIC*, onde a *CPU* e os periféricos estão ligados. Neste modo, o microcontrolador está executando todas as funções para as quais ele foi programado e, obviamente, é o modo em que há o pleno consumo de energia elétrica. O modo *IDLE* é o modo em que a *CPU* fica desligada, porém seus periféricos continuam ativados. Neste modo, o *PIC* consome apenas 5,8μA. E, por último, o modo *sleep*, em que ambos (*CPU* e periféricos) são desativados assim que o programa encontra a instrução para entrar neste modo, consumindo tipicamente 0,1μA.

Estes modos são particularmente úteis quando o circuito é alimentado por baterias, maximizando o seu tempo de carga. Já para o projeto em questão, que é alimentado pela rede elétrica, o uso do modo *sleep,* conforme apresentado anteriormente, garante um consumo mínimo de energia elétrica por parte do circuito de comando, quando o mesmo não está realizando o controle. Não se justifica um consumo de energia mesmo pequeno, mas que pode ser reduzido, quando o chuveiro não está em uso. Isto seria caracterizado como desperdício. Assim, além da ativação do modo *sleep*, foi implementado um circuito de comando que, através do próprio

software, desliga o *LCD*. Este, quando ligado, consome 2mA somados aos 130mA para luz de fundo (*backlight*).

Estando o *PIC* em modo *sleep*, existem duas formas para que ele retorne em seu estado normal de trabalho: por meio de um reinício (*reset*), que é gerado por oito eventos especiais (*Power-on Reset*, *Brown-out Reset*, *Reset* por operação normal através do pino *MCLR*, *Reset* por modo de gerenciamento de energia através do pino *MCLR*, *Watchdog Timer*, *Instrução de Reset*, *Stack full Reset* e *Stack underflow Reset*) e por meio de uma interrupção. Para o controlador do projeto proposto, o modo *sleep* será interrompido através do acionamento da mini-chave posicionada dentro do chuveiro, que provoca uma interrupção externa. Também é possível através de um *reset* por operação do pino *MCLR*, mas somente quando o usuário dispõe da chave conforme descrito na seção 1.1 deste capítulo.

2.2.5 Conversão analógica/digital

Um dos pontos mais importantes do projeto para este controlador de temperatura é a aquisição dos dados provenientes dos sensores *LM35*. São usados dois deles: um para medir a temperatura da água que entra (que será considerada a temperatura ambiente inicial) e o outro para medir a temperatura da água que sai do chuveiro, a chamada variável de processo, ou PV, que é a que desejamos controlar. Além destes valores, é necessário também saber qual a temperatura desejada pelo usuário. Este valor é o chamado *set-point* (SP), que será obtido através de um resistor variável ou através da linearização de uma curva, quando o chuveiro estiver no modo automático.

Para fazer a aquisição destes valores utilizamos o periférico conversor AD presente no *PIC*, onde a tensão analógica proporcional à temperatura sofrerá uma conversão para um valor digital (representação binária). Esse valor binário (8 ou 10 bits) é diretamente proporcional à faixa estabelecida pela tensão de referência, fixada em um de seus pinos.

A resolução do conversor AD utilizada neste projeto é de 10 bits. Sendo assim, temos 1024 (2^{10}) valores binários distintos para representar a faixa de temperatura analógica obtida através do sensor *LM35* e do potenciômetro.

Para uma tensão de alimentação do *PIC* em 5 volts, e fazendo desta a tensão de referência para o conversor AD, teríamos o seguinte resultado:

$$V_{bit} = \frac{V_{referência}}{2^{resolução} - 1} \tag{2}$$

$$V_{bit} = \frac{5000mV}{2^{1024} - 1}$$

$$V_{bit} = \frac{5000mV}{1023}$$

$$V_{bit} = 4{,}887585mV$$

Ou seja, a cada 4,887585mV de variação de tensão proveniente do sensor ou do potenciômetro, o conversor AD assume um valor de 0 a 1023. O valor do sinal analógico pode ser obtido pela equação 3.

$$V_{sinal\_analógico} = V_{bit} * Valor_{conversor} \tag{3}$$

Onde, $Valor_{conversor}$ é o valor de 0 a 1023 do conversor AD.

No projeto em questão foi utilizada uma tensão de referência para o conversor AD do *PIC* de 0,45 volts, de forma a aproveitar toda a faixa dos dez bits de conversão (1023 valores). Assim, a faixa de 0 a 0,45 volts corresponde linearmente à faixa de temperatura de operação considerada para o chuveiro de 0 a 45°C[4], ou seja, para uma variação de tensão de 0 a 0,45 volts na entrada do conversor AD, teremos um sinal digital correspondente de 0 a 45°C. Esta faixa é para trabalho tanto do sensor *LM35*, cuja resolução é de 10mV/°C e faixa de 2 a 150°C, quanto para o potenciômetro de ajuste do *set-point* manual, que terá seu valor máximo fixado no valor da soma entre a temperatura da água de entrada no chuveiro com a variação ΔT alcançada pelo chuveiro, para a vazão de água estabelecida.

[4] veja considerações sobre a potência do chuveiro nas seções 2.3 e 3.3

2.2.5.1 Tempo de aquisição e tempo de conversão

Tempo de aquisição é o tempo necessário para carregar o canal AD selecionado, garantindo a carga do capacitor C_{hold} com o nível de tensão presente na entrada do conversor (grandeza analógica a ser medida). Após este tempo, o capacitor C_{hold} é desconectado da entrada iniciando o processo de conversão desta tensão em um valor digital. Neste ponto temos então o início do tempo de conversão.

Para o *PIC18F4550*, alimentado por 5 volts, segue que:

$$T_{\min\ \_aq} = T_{AMP} + T_C + T_{COFF} \tag{4}$$

Onde,

T_{AMP} é o tempo de estabilização do amplificador (0,2μs para o *PIC18F4550*)

T_C é o tempo de carregamento do capacitor C_{hold}.

$$T_C = C_{hold}\,(R_{IC} + R_{SS} + R_S) ln(1/2048) \tag{5}$$

T_{COFF} é o coeficiente de temperatura dado por:

$$T_{COFF} = (T_{AMB} - 25℃) * 0.02\mu s/℃ \tag{6}$$

Assim, segundo MIYADAIRA[5], para uma temperatura ambiente de 50°C, por exemplo, o tempo mínimo de aquisição seria de 1,75μs. Considerando esta afirmativa, podemos estabelecer um tempo de espera de 2μs após a seleção de cada canal AD a ser lido para que as três grandezas analógicas sejam adquiridas a partir da entrada. Esse tempo seria para uma temperatura T_{AMB} aproximada de 65°C, que dificilmente seria atingida no ambiente em questão.

Tempo de conversão é o tempo para a conversão analógico-digital para cada bit e deve estar entre 0,7μs e 25μs. Vale a equação 7.

[5] MIYADAIRA, 2010, p.268.

$$T_{AD} = \frac{1}{F_{osc}/16}$$

(7)

Assim, para uma frequência de oscilação (F_{osc}) de 20MHz, teremos um T_{AD} de 0,8µs e para uma resolução de 10 bits do conversor AD, teremos um T_{AD} total de 8µs.

Portanto, o tempo total para que as três variáveis usadas neste controlador (Tamb, SP e PV) assumam seus valores, seria de, aproximadamente, 30µs.

2.2.6 Módulo de interrupção

As interrupções nos *MCUs* são paradas na execução normal do programa para que seja executada uma rotina de maior prioridade, a *ISR* (Rotina de Serviço de Interrupção), designada para uma determinada ação. Após terminada, o processador retorna ao ponto onde o programa estava quando a interrupção foi gerada.

As fontes geradoras de interrupção podem ser externas, como a mudança do estado de um pino de entrada, ou interna, pela ocorrência de eventos como *TIMER*, conversão analógica/digital, comunicação *USB* etc..

Existem também os modos de tratamento de interrupções, em que pode ser definida a prioridade de cada uma delas. No modo-padrão, a prioridade não é levada em consideração. Assim, quando ocorre uma interrupção, a execução é desviada para a sua *ISR*, que é executada até o seu final. Quando uma ordem de prioridade é definida através de uma diretiva do sistema, o desvio do programa ocorre para a interrupção de mais alta prioridade durante a execução de uma interrupção de menor prioridade.

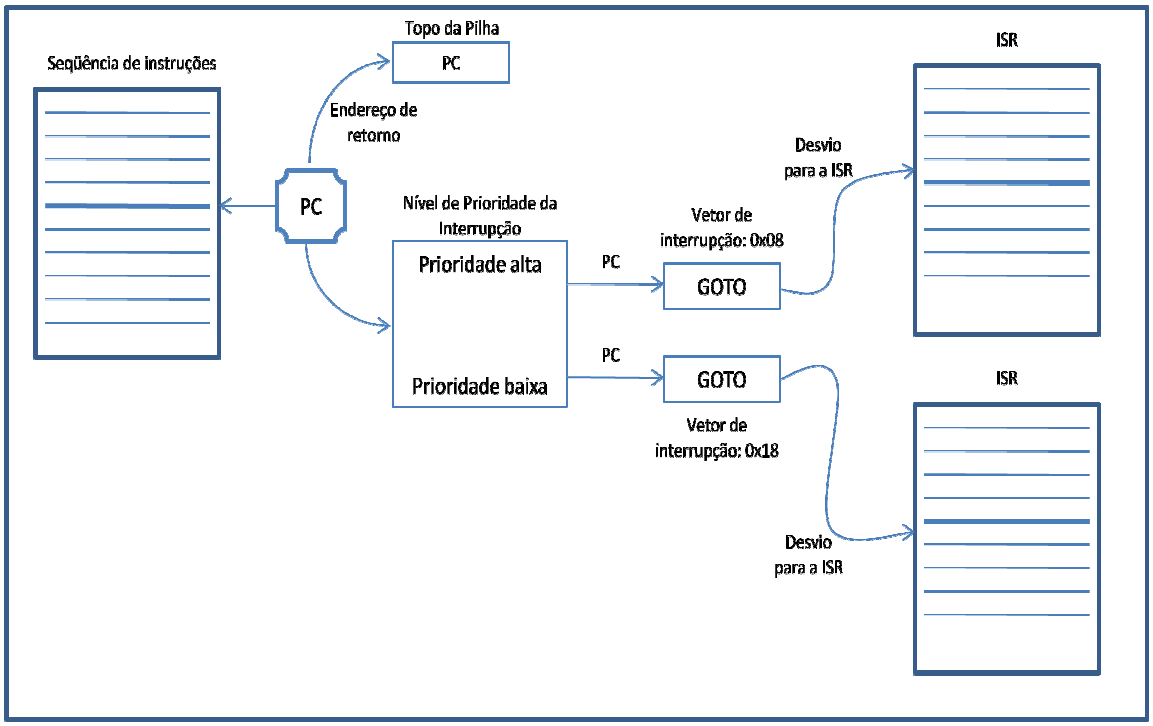

Figura 6 - Comportamento com prioridade de interrupção
Fonte: MIYADAIRA, 2010, p.202.

Para este projeto, foram habilitadas as interrupções externas 1 e 2 (*int_ext* e *int_ext1*) e as interrupções para os temporizadores (*TIMERs)* 0 (zero), 1, 2 e 3, além da interrupção da interface *USB*, que está dentro do código do protocolo *USB_CDC*, com a seguinte ordem de prioridade:

#priority int_EXT1, int_TIMER0, int_TIMER1, int_TIMER3, int_EXT, int_TIMER2, int_USB

Desta forma, a sequência acima determina que a interrupção externa 1 (*int_EXT1*) é a de maior prioridade.

Na seção seguinte será detalhada a função e a configuração feita para cada interrupção utilizada na programação do controlador de temperatura deste projeto.

2.2.6.1 Geração dos pulsos para o controle

As interrupções do *MCU PIC18F4550* desempenham juntas, neste projeto, um papel muito importante na geração do sinal de controle do ângulo de fase, que, aplicado à senoide da rede elétrica, faz a variação da tensão na carga. Sem o uso delas, por exemplo, seria necessário utilizar o integrado *TCA785*. Este, através de um sinal *PWM* proveniente do *PIC*, faria o controle do ângulo de fase. Com isso, podemos reduzir parte do circuito externo que seria necessário para obter o mesmo resultado.

Isto foi feito através da ativação de interrupções de forma sequencial e temporizada, gerando pulsos de duração definida em uma das portas *I/O* do *PIC*. Tais pulsos serão enviados ao circuito de potência, composto basicamente por *DIAC* e *TRIAC*, que são os componentes eletrônicos atuantes na variação da tensão aplicada na resistência elétrica do chuveiro.

Primeiramente, vamos considerar a interrupção externa *int_ext* trabalhando em função da frequência de 60Hz (*hertz* - ciclos por segundo, período T = 16,67ms) da rede elétrica, sendo chamada a cada passagem por zero da mesma através de um circuito externo detector de zero. Assim, esta interrupção ocorre numa frequência de 120Hz, ou seja, a cada 8,33ms (milisegundos), aproximados. Esta interrupção é utilizada para contagem de tempo usado para temporização de entrada do modo de economia de energia: o modo *sleep*. Além disso, ela define qual será o tempo em que o pulso deverá acontecer na porta *I/O* (tempo Tx), a partir do início de uma passagem por zero da rede elétrica, tempo este que é definido na função controle proporcional/integral, de acordo com os parâmetros do controlador e das variáveis em questão (temperatura da água no ambiente, temperatura da água e *set-point*). Lembramos que o pulso deve ocorrer tanto no semiciclo positivo, quanto no semiciclo negativo da senoide.

Neste ponto, já haverá um valor de Tx, calculado pela função controle, a ser temporizado pelo *TIMER 1* (*int_timer1)*, que é habilitado para iniciar sua contagem dentro da interrupção externa *int_ext*. O *TIMER 1*, por sua vez, quando ocorre, gera o pulso na porta I/O do *PIC* por um período de tempo Tp (tempo do pulso)

estabelecido e contado pelo temporizador *TIMER 0* (*int_timer0*). Dentro da própria *ISR* do *TIMER 1*, ocorre a sua desabilitação, para que seja possível repetir a contagem deste tempo, além de habilitar a interrupção *TIMER 0*, que fará o desligamento do pulso, quando esta ocorrer. A interrupção *TIMER 0* também é desabilitada dentro de sua própria *ISR*.

Os tempos para as interrupções *TIMER 0* e *TIMER 1* foram estabelecidos levando-se em consideração o tempo gasto pelo *PIC* para executar a função controle e fazer a aquisição/conversão de dados do conversor analógico-digital (tempo de aquisição e tempo de conversão), além do tempo gasto pela *PIC* na execução das linhas de código até a geração do pulso. Sendo assim, o tempo até o início do pulso, chamado Tx e temporizado por *TIMER 1*, não é menor que 100µs (microsegundos), a partir da passagem por zero. O tempo de duração do pulso, chamado Tp e temporizado por *TIMER 0*, possui o valor de 80µs. Desta forma, resta um tempo, antes do final do primeiro semiciclo da senoide, de aproximados 53,3µs. Isto para que a duração do pulso não atinja a próxima passagem por zero da rede, o que afetaria o próximo semiciclo. Estes tempos estão representados na FIG. 7.

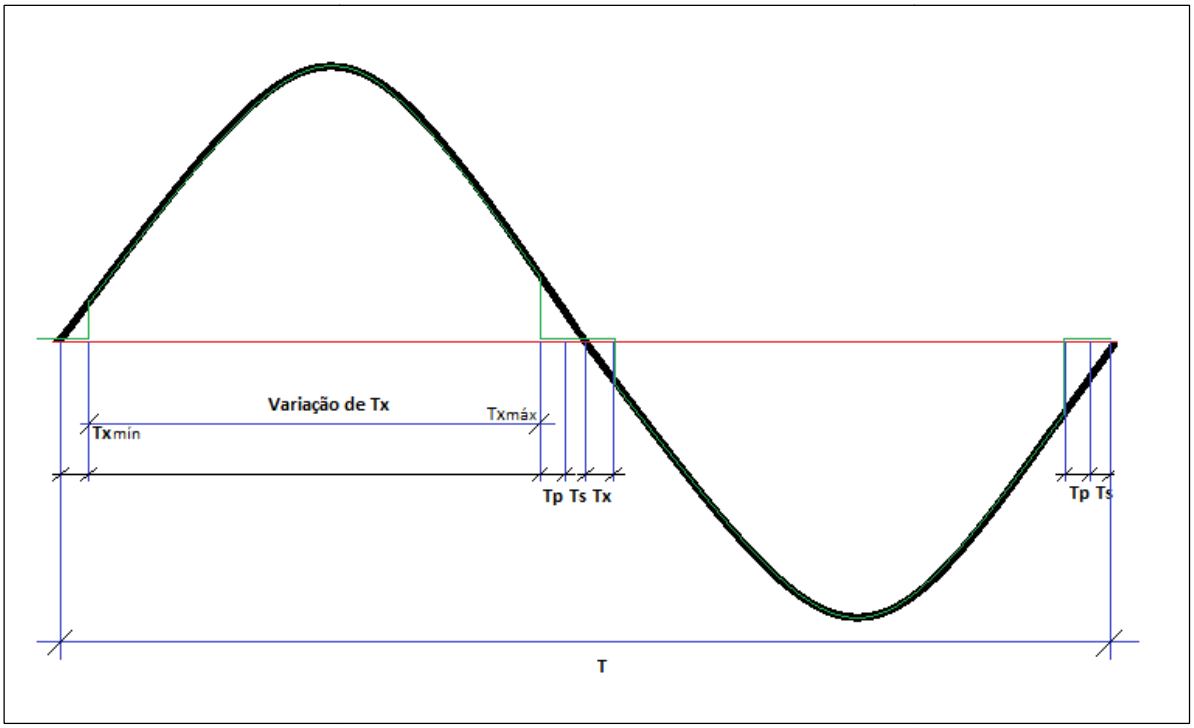

Figura 7 - Representação da senoide da rede com as temporizações
Fonte: do autor.

Com isso, dentro deste intervalo de 100µs a 8,10ms, que são o mínimo e o máximo tempo de aproveitamento do semiciclo da senoide, teremos uma tensão máxima um pouco menor que a tensão da rede e uma tensão mínima um pouco superior a 0 (zero) volts, respectivamente.

Os valores para os temporizadores foram obtidos da seguinte forma, considerando a frequência de máquina (F_{maq}) do *PIC* de 12MHz:

T é o período de oscilação do temporizador e é dado pela equação 8.

$$T = \frac{1}{\frac{12x10^6}{prescaler}} \tag{8}$$

O tempo de *overflow* t (tempo em que é gerada a interrupção), é dado pela equação 9.

$$t = T * 2^{n^{\circ} bits} \tag{9}$$

O *prescaler* pode ser definido como o número de vezes em que o evento deve ocorrer antes que o registro para *overflow* seja incrementado. Por exemplo, o evento pode ocorrer a cada duas bordas de subida do sinal de *clock*, ao invés de uma.

TIMER 0

$$T = \frac{1}{12x10^6/1} = 83{,}3333x10^{-9}s$$

$$t = T * 65536 = 5{,}461310^{-3}s$$

para contar 80µs, aplica-se regra de três simples, onde:

t --> 65536
80µs --> X
X = 960

Portanto, devemos fazer: *set_timer0(65536-960)*.

TIMER 1

$$T = \frac{1}{\frac{12x10^6}{8}} = 6666{,}67x10^{-9}s$$

$$t = T * 65536 = 43{,}690610^{-3}s$$

Neste caso temos que calcular dois valores: um para 100μs e um para 8,10ms. Assim, pela regra de três, temos: X_1 = 150 e X_2 = 12150, respectivamente.

Desta forma, o valor para a função *set_timer1(65536-X_n)* deve estar dentro desta faixa, que é obtida pela função controle.

O procedimento para os demais temporizadores (*TIMER 2* e *TIMER 3*) é o mesmo, lembrando que o primeiro é de apenas 8 bits (256) e foi definido com um bloco *prescaler* de 16 e um bloco *postscaler* de 16, e o segundo é de 16 bits e um bloco *prescaler* de 8. O bloco *postscaler* funciona como um contador de sinais para que o *flag bit* de interrupção seja habilitado. Basta fazer o valor de *t* multiplicado pelo bloco *postscaler* para obter o novo valor de *overflow*. O bloco *prescaler* está presente em todos os *TIMERs*, mas o *postscaler*, de acordo com MIYADAIRA[6], está presente apenas no *TIMER 2* e *TIMER 4*.

2.3 Considerações para o controle automático

Tendo como objetivo elaborar um controlador de temperatura em que o usuário possa simplesmente ligar a água do chuveiro e tomar seu banho sem precisar se preocupar em regular a temperatura da água, foi implementado o modo de controle automático. Sendo assim, o controlador toma por base a temperatura da água que

[6] MIYADAIRA, 2010, p.228.

está chegando ao chuveiro, proveniente do reservatório (caixa d'água), para, automaticamente, gerar um valor de *set-point* (valor desejado para a água do banho), através de valores pré-estabelecidos vindos de análise dos dados da pesquisa de campo, conforme APÊNDICE A. Para o protótipo apresentado aqui, estes valores foram estabelecidos em uma vazão mínima e constante de água.

Desta forma, considerando que a temperatura da água a ser aquecida segue um valor próximo à temperatura em que se encontra o ambiente, que em dias de calor está em valores mais elevados e em dias de frio possui valores bem abaixo de 20°C, uma curva ideal para estabelecer o *set-point* automático poderia ser como no gráfico da FIG. 8.

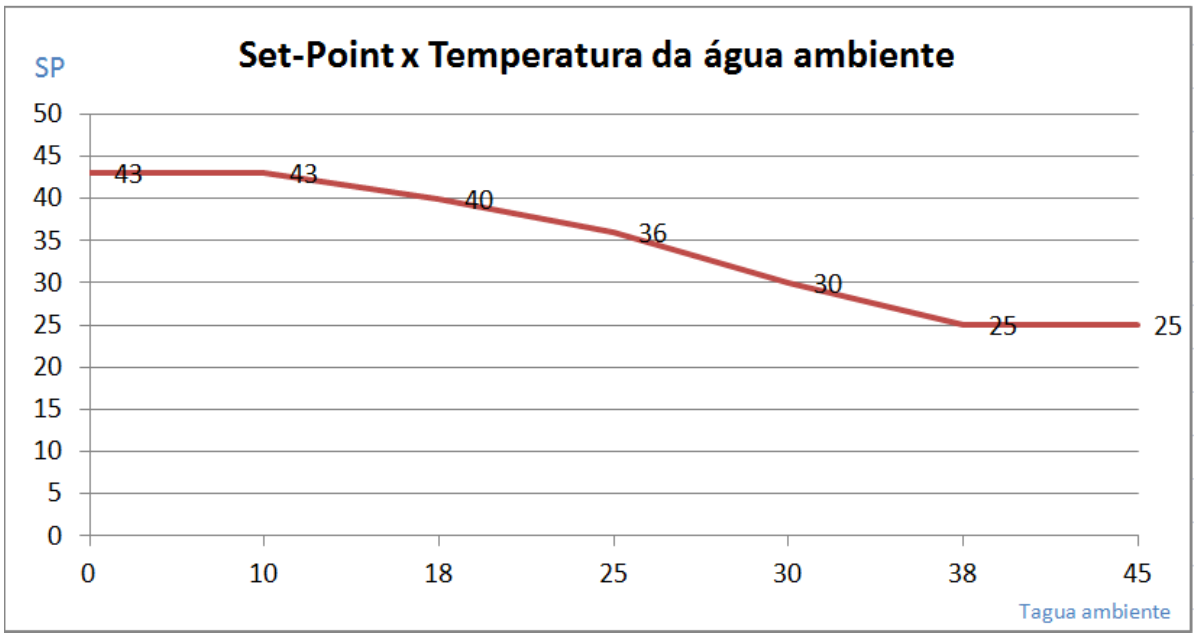

Figura 8 - Curva ideal para o set-point automático
Fonte: do autor.

Neste gráfico, vemos que quanto mais baixa a temperatura da água no ambiente, maior é o valor de temperatura para a água do banho, até um máximo de 43°C, conforme preferência dos usuários em pesquisa anexa e levando em consideração a sensibilidade térmica humana, descrita adiante. Para temperaturas intermediárias da água no ambiente entre 10°C e 38°C, o *set-point* varia linearmente entre os valores 43°C e 25°C, respectivamente, também conforme preferência dos usuários.

Segundo SMANIOTO[7], o corpo humano, depois das atividades que realiza, até mesmo as atividades orgânicas, necessita trocar calor com o ambiente para que não o acumule. Porém, se o ambiente estiver em temperatura maior ou igual ao corpo humano, haverá dificuldade em se realizar esta troca. Por isso, o corpo humano possui, na pele, dois tipos de receptores de temperatura, um com sensibilidade entre 23°C e 28°C e outro entre 38°C e 43°C. São eles que fazem com que a pessoa sinta frio ou calor. Sendo assim, para temperaturas da água no ambiente acima de 38°C, foi estabelecido o valor de *set-point* em 25°C, pois "... é a temperatura que os seres humanos nus começam a sentir frio", de acordo com Trilhas e Rumos[8], e, também por essa premissa, foi estabelecido o *set-point* de 43°C para a máxima temperatura da água do banho, em ambientes mais frios.

No entanto, para se conseguir uma variação de temperatura da água (ΔTm) de 43°C, teríamos que dispor de um chuveiro de potência mais elevada, e consequentemente, também de um TRIAC de maior potência para suportar toda a corrente elétrica necessária a essa variação. Mesmo para uma variação pequena de temperatura, o ideal é um chuveiro de maior potência, pois caso o tempo para se atingir a temperatura desejada seja muito alto, tornaria inviável o projeto, devido ao alto consumo de água. Na construção do protótipo deste controlador será utilizado um chuveiro cuja potência nominal escolhida é de 3400 watts. Este chuveiro consegue elevar a temperatura da água por uma variação máxima ΔTm de, aproximadamente, 16°C em um tempo em torno de 1 minuto e para uma vazão de 2 l/min (litros por minuto) de água (veja valores para resposta ao degrau na próxima seção). Porém, nada impede a construção de um protótipo com componentes e um chuveiro de maiores potências para melhores resultados e, até mesmo, para uma variação de 43°C.

Outro ponto importante a ser considerado é que não devemos usar toda a variação de temperatura obtida (ΔTm), para que seja possível um controle mais rápido e eficiente. Desta forma, foi estabelecido, para este protótipo, um ΔT de 12°C.

[7] SMANIOTO, 1991.
[8] Trilhas e Rumos, 2007.

Sendo assim, a curva para o *set-point* automático para o chuveiro utilizado no protótipo é a apresentada na FIG. 9.

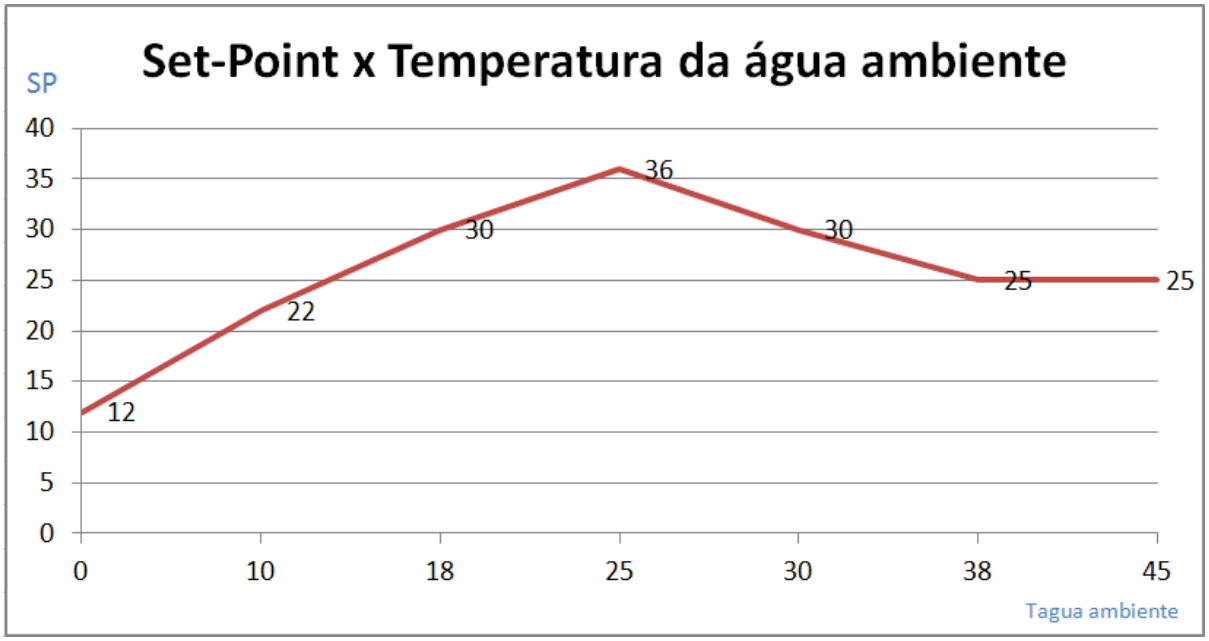

Figura 9 - Curva do set-point automático para o protótipo
Fonte: do autor.

Esta curva foi linearizada para obtermos as equações de conversão de valores do set-point, através de código em linguagem C utilizada na programação do *PIC*. O código fonte se encontra no APÊNDICE B.

2.4 O sistema de controle

Nesta seção mostraremos com foi realizado o processo de sintonia do controlador PI utilizando a regra de Ziegler-Nichols, através de seu primeiro método, que é o de resposta ao degrau unitário realizado em teste de malha aberta. O segundo método, que é feito em malha fechada, consiste em aumentar o ganho proporcional gradativamente até encontrar um valor Ku (ganho último) e Pu (período último), cuja resposta do sistema é oscilatória com amplitude constante. Este método não foi usado aqui pela dificuldade em encontrar esta oscilação e também por não ser um

método muito utilizado na indústria, segundo CAMPOS[9]. Tanto o primeiro quanto o segundo métodos de sintonia são úteis quando os modelos matemáticos da planta são desconhecidos, porém podem também ser aplicados em modelos conhecidos. A aplicação do primeiro método, neste caso, se justifica pelo fato de que a planta em questão é do tipo estável e pode ser aproximada por um sistema de primeira ordem. No entanto, esses métodos fornecem valores estimados dos parâmetros, sendo, portanto, um ponto de partida para uma sintonia fina, conforme OGATA[10].

2.4.1 Resposta ao degrau unitário

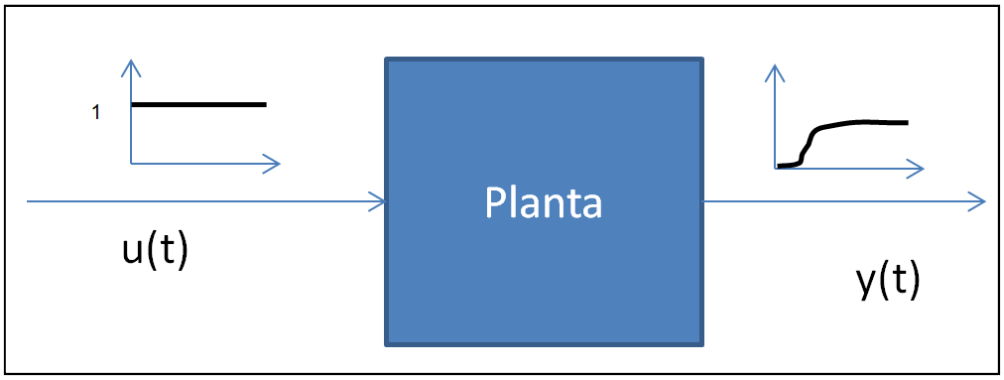

Figura 10 - Resposta ao degrau unitário de uma planta
Fonte: OGATA, 2006. p.559.

O método de resposta ao degrau consiste em aplicar um valor na saída do controlador e observar a resposta da planta, como mostra a FIG. 10 representativa. Desta forma, alimentando o chuveiro com uma vazão mínima de água, em torno de 2 litros/minuto, e aplicando a tensão máxima da rede elétrica em sua resistência, durante um intervalo de tempo, foram obtidos os dados para a construção de um gráfico com o seguinte aspecto, mostrado na FIG. 11.

[9] CAMPOS, 2006. p.49.
[10] OGATA, 2006.

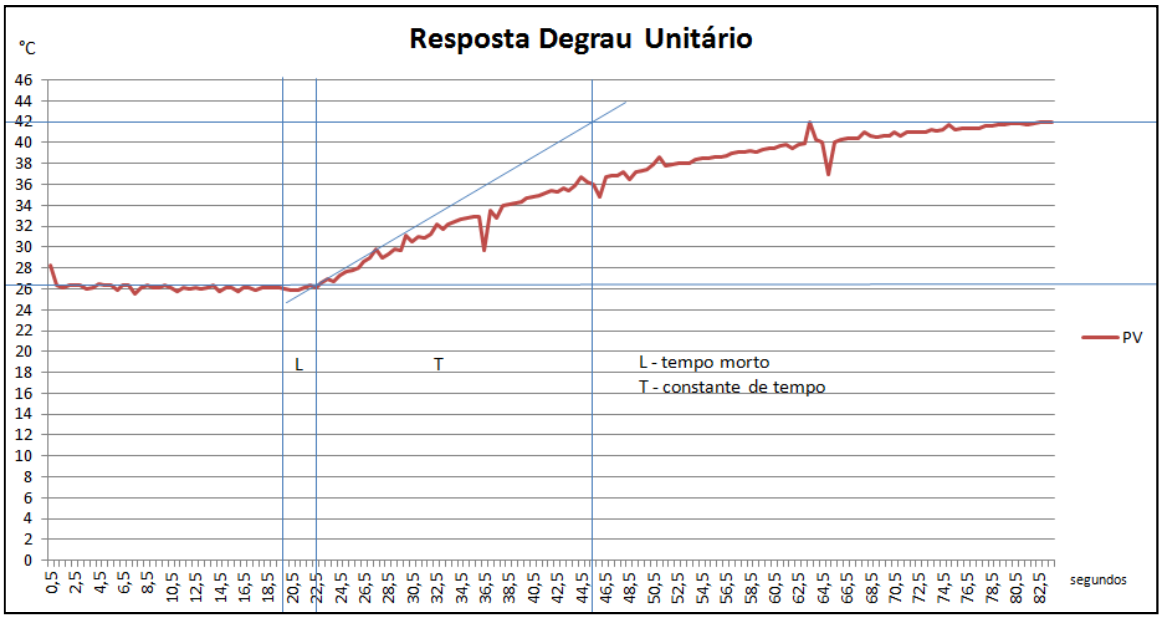

Figura 11 - Curva de resposta ao degrau unitário da planta
Fonte: do autor.

Estes dados foram recebidos através da comunicação USB implementada no protótipo, a uma taxa de 500ms.

De acordo com Ogata[11], "se a planta não possui integradores nem pólos complexos conjugados dominantes, então esta curva de resposta ao degrau unitário pode ter o aspecto de um S", conforme podemos ver na FIG. 11, e, portanto, podemos aplicar o método citado.

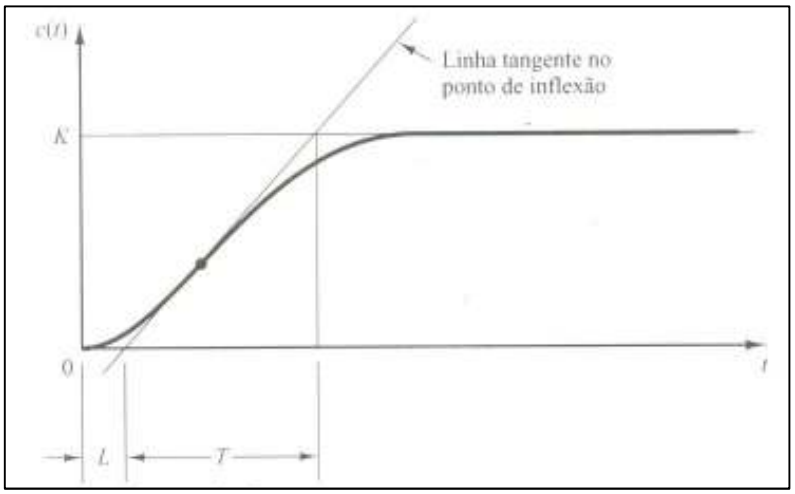

Figura 12 - Ponto de inflexão de uma curva
Fonte: OGATA, 2006. p.559.

[11] OGATA, 2006. p.558.

Traçando uma reta tangente ao ponto de inflexão da curva de resposta ao degrau, conforme mostra o exemplo da FIG. 12, obtemos os parâmetros T (constante de tempo) e L (tempo morto) da planta e sua função de transferência pode ser aproximada por um sistema de primeira ordem com um atraso de transporte, conforme equação 10 abaixo.

$$G_p(s) = \frac{Y(s)}{U(s)} = \frac{ke^{-Ls}}{Ts + 1} \tag{10}$$

O parâmetro k é o ponto de saturação da curva, também chamado de ganho da planta. O parâmetro L mede o atraso da planta, que é o tempo em que a planta demora a responder ao degrau aplicado. O parâmetro T é o tempo de subida da planta até 63% do ponto de saturação.

Os valores obtidos para T e L, foram, respectivamente, 23 segundos e 2,5 segundos e, considerando a vazão mínima estabelecida, o parâmetro k teria o valor de 16°C, porém estabelecemos este valor em 12°C, conforme consideração feita anteriormente no item 2.3.

Fazendo uso da aproximação de Padé (equação 11) de primeira ordem, onde,

$$e^{-x} = \frac{1}{1 + x} \tag{11}$$

E, aplicando-a na equação 10, obtemos a função de transferência da planta em malha aberta (equação 12).

$$G_p(s) = \frac{k}{LTs^2 + (L + T)s + 1} \tag{12}$$

Sendo assim, substituindo os valores obtidos, temos a equação 13.

$$G_p(s) = \frac{12}{57{,}5s^2 + 25{,}5s + 1}$$

(13)

2.4.2 O controlador PI

Para CAMPOS[12], das diversas malhas de controle usadas na indústria, como refinarias, plantas químicas e de papel, 97% usam o algorítmo PID devido à simplicidade no ajuste de seus parâmetros e a estar disponível em quase todos os equipamentos de controle.

O objetivo do algorítimo PID é trabalhar em cima do erro entre a variável controlada (PV) e o valor do set-point (SP), de modo a eliminá-lo. Para isso, existem três módulos desse algorítimo, o proporcional, o integral e o derivativo, que são combinados para gerar os controladores tão usados nas indústrias: P (proporcional), PI (proporcional e integral), PD (proporcional e derivativo) e PID (proporcional, integral e derivativo).

O controlador PI gera uma saída que é proporcional ao erro e proporcional à integral desse erro. Considerando o controlador PI paralelo clássico, em que o fator proporcional Kp multiplica também o termo integral, temos a equação 14.

$$G_c(s) = Kp\left(1 + \frac{1}{T_i s}\right)$$

(14)

O fator $\frac{1}{T_i}$ é o ganho integral e representa o número de repetições por segundo, ou seja, o valor de Ti significa o tempo em que a ação integral repetirá a ação proporcional, com a tendência de saturar a saída do controlador e eliminar o erro de regime permanente, inerente ao controle puramente proporcional. Essa saturação pode gerar, na prática, sobrevalor (*overshoot*) e oscilações, porém, em controladores digitais, é facilmente resolvida com o conceito do "*reset windup*". Para

[12] CAMPOS, 2006.

isto, basta limitar o valor da soma da variação desejada para a saída com o valor atual, em 100% do valor da saída.

Nos controladores digitais, normalmente é utilizada a implementação do algorítimo em velocidade, que garante a facilidade do *reset windup.* Assim, a equação para o controlador PI paralelo clássico ficará conforme a equação 15.

$$G_c(s) = Kp\left(1 + \frac{1}{T_i s}TA\right) \tag{15}$$

O termo TA é a taxa de amostragem do controlador digital e o termo $\frac{1}{T_i}$ é igual à constante integral Ki.

2.4.3 A sintonia do controlador PI

A FIG. 13 mostra o diagrama de um sistema em malha fechada.

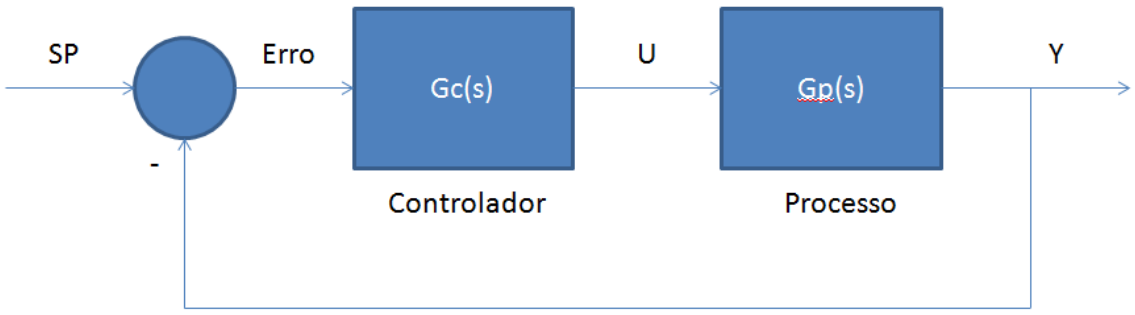

Figura 13 - Diagrama de blocos de um sistema em malha fechada
Fonte: CAMPOS, 2006. p.45.

Desta forma, a saída do controlador é dada pela equação 16.

$$U(s) = G_c(s)xE(s) \tag{16}$$

E a saída do processo pela equação 17.

$$Y(s) = Gp(s)xU(s)$$

(17)

Sabemos que o erro é a diferença entre o *set-point* e a variável de processo, portanto:

$$E(s) = SP(s) - Y(S)$$

(18)

Sendo assim, a função de transferência do sistema em malha fechada é como na equação 20.

$$\frac{Y(s)}{SP(s)} = \frac{Gp(s)G_c(s)}{1 + Gp(s)G_c(s)}$$

(20)

Vemos que os pólos desta função são as raízes do seu denominador e, para que o sistema seja estável, deverão ter parte real negativa. Logo, percebemos que esta estabilidade depende também dos parâmetros de sintonia do controlador C(s).

Existem vários critérios de desempenho para a sintonia do controlador, como o menor sobrevalor (ou *overshoot*), a razão de declínio, o menor tempo de assentamento, o menor tempo de ascensão etc., além da robustez que garante que os pólos da função de transferência tenham sempre a parte real negativa para qualquer modelo dinâmico do processo (Gp(s)), segundo CAMPOS[13].

A partir de então, podemos realizar os cálculos dos valores para os parâmetros Kp e Ti, através da tabela proposta por Ziegler e Nichols, em 1943.

Tabela 2 - Sintonia segundo Ziegler e Nichols

Controlador	Kp	Ti	Td
P	T/(K*L)	-	-
PI	0.9*T/(K*L)	3.33*L	-
PID	1.2*T/(K*L)	2*L	0.5*L

Fonte: CAMPOS, 2006. p.51.

[13] CAMPOS, 2006.

Nesta tabela, o parâmetro K (ganho do processo) é a variação em percentual da saída pela entrada, com valores normalizados (0-100%) de acordo com o *range* do equipamento *(faixa de atuação)*, dada pela equação 21.

$$K = \frac{\Delta y(\%)}{\Delta u(\%)} \tag{21}$$

Porém algumas considerações, segundo CAMPOS[14], devem ser observadas antes de usarmos a tabela para os cálculos. Primeiro foi considerado que Ziegler e Nichols utilizavam simulações computacionais com PID paralelo clássico para obter o método; segundo, que o fator de incontrolabilidade (L/T) deve estar entre 0,1 e 0,3 para melhor desempenho (tempo morto não significativo) e; terceiro, que o método foi feito para controladores analógicos. Então devemos aumentar o tempo morto de um valor igual à metade do período de amostragem (L' = L +TA/2) para controladores digitais, quando a taxa de amostragem for considerável.

Segundo CORREA[15], “em geral, quando o valor do tempo morto é comparável com o valor da constante de tempo pode-se dizer que a malha é particularmente difícil de ser controlada”; mas “se o tempo morto for inferior a 20% da constante de tempo, podemos utilizar o controlador PI, levando a zero a constante derivativa”. Isto justifica o uso do controle apenas proporcional/integral neste projeto, já que o valor do tempo morto é de 10,87% da constante de tempo.

O ganho do processo é de K=1, pois a variação na saída da planta foi de 100% (ΔPV=16°C) para uma variação na saída do controlador também de 100% (ΔMV=115 volts), na resposta ao degrau.

Aplicando a TAB. 2 na equação do controlador PI paralelo clássico (equação 15), obteremos a equação 23, que nos mostrará seus pólos e zeros.

$$G_c(s) = Kp\left(1 + \frac{1}{T_i s}TA\right) \tag{22}$$

[14] CAMPOS, 2006.
[15] CORREA, 2002. p.102.

$$G_c(s) = 0{,}9\frac{T}{L}\left(1 + \frac{TA}{\frac{L}{0{,}3}s}\right)$$

Obtemos:

$$G_c(s) = \frac{0{,}9\frac{T}{L}\left(s + \frac{0{,}3TA}{L}\right)}{s}$$

(23)

A equação 23 mostra que o controlador PI paralelo clássico tem um pólo na origem e um zero em:

$$s = -\frac{0{,}3TA}{L}$$

(24)

Assim, aplicando os valores obtidos da resposta ao degrau na TAB. 2, considerando que a taxa de amostragem de 50ms estabelecida no *PIC* é um valor considerável em relação ao valor de 2,5 segundos do tempo morto, fazendo, então L' = L+(TA/2), e que o fator de incontrolabilidade (L'/T) é de 0,1097, que está dentro da faixa para este método, obtivemos os seguintes valores para os parâmetros do controlador PI:

$$Kp = 8{,}198$$

e

$$Ti = 8{,}408 \quad \text{ou} \quad Ki = 0{,}119$$

Pela equação 24 temos, portanto, um zero em:

$$s = -0{,}00594$$

Aplicando valores na equação 23, obtemos a Gc(s) do controlador.

$$G_c(s) = \frac{8{,}198(s + 0{,}00594)}{s}$$

(25)

Multiplicando a equação 13 (função de transferência da planta) pela equação 25, conforme equação 20, obtemos a função de transferência do sistema em malha fechada (equação 26).

$$\frac{Y(s)}{SP(s)} = \frac{98{,}376s + 0{,}58434344}{57{,}5s^3 + 25{,}5s^2 + s} \tag{26}$$

Como dito anteriormente, estes valores são um ponto de partida para uma sintonia fina e devemos realizar observações do processo para melhorar seu comportamento.

3 DESENVOLVIMENTO DO PROJETO

3.1 Os circuitos em simulação virtual

Nesta seção serão detalhadas as partes principais do circuito do controlador, que é composto pelo circuito de comando, circuito detector de zero da rede elétrica, circuito de potência da carga, circuito de alimentação e o circuito sensor.

Todos estes circuitos foram desenvolvidos e testados através do *software* simulador *ISIS Professional Release 7.8 SP2 with Advanced Simulation*, da *Labcenter Eletronics*©. Assim foi possível analisar o funcionamento de cada componente do controlador antes de testá-los em circuitos reais.

3.1.1 Circuito de comando

O circuito de comando é composto principalmente pelo *MCU PIC18F4550*, que, conforme descrito no item 1.2, é o responsável por toda a lógica de funcionamento do controlador de temperatura proposto. Através dele são realizados os cálculos para a geração dos pulsos de controle da tensão na carga, a aquisição/conversão dos dados provenientes dos sensores e de referência do usuário, controle de todas as indicações luminosas, gerenciamento da comunicação *USB* para envio dos dados de análise do controle, todas as temporizações envolvidas no processo pelas interrupções e monitoramento das chaves externas de comando.

A FIG. 14 mostra o circuito de comando elaborado no software simulador ISIS, com seus respectivos elementos externos. Nela podemos ver as entradas PV, SPManual e Tagua, conectadas às entradas analógicas do *PIC* e provenientes do circuito sensor. Temos a entrada *INT0* no pino 33 que é proveniente do circuito de detecção de zero e a saída PULSO no pino 15, onde o pulso gerado pela função controle segue para o circuito de potência. Além disso, temos os *leds* indicadores de estado

de funcionamento, as chaves de comando ligadas às portas *I/O* por resistores de *pull-up*, o *LCD* 2x16 com oito vias de comunicação, o resistor ajustável (*trimpot*) conectado ao pino 5 (VRef+) para 0,45 volts de tensão de referência para o módulo de conversão AD, a chave com o *led* do *BOOTLOADER* e a comunicação *USB*.

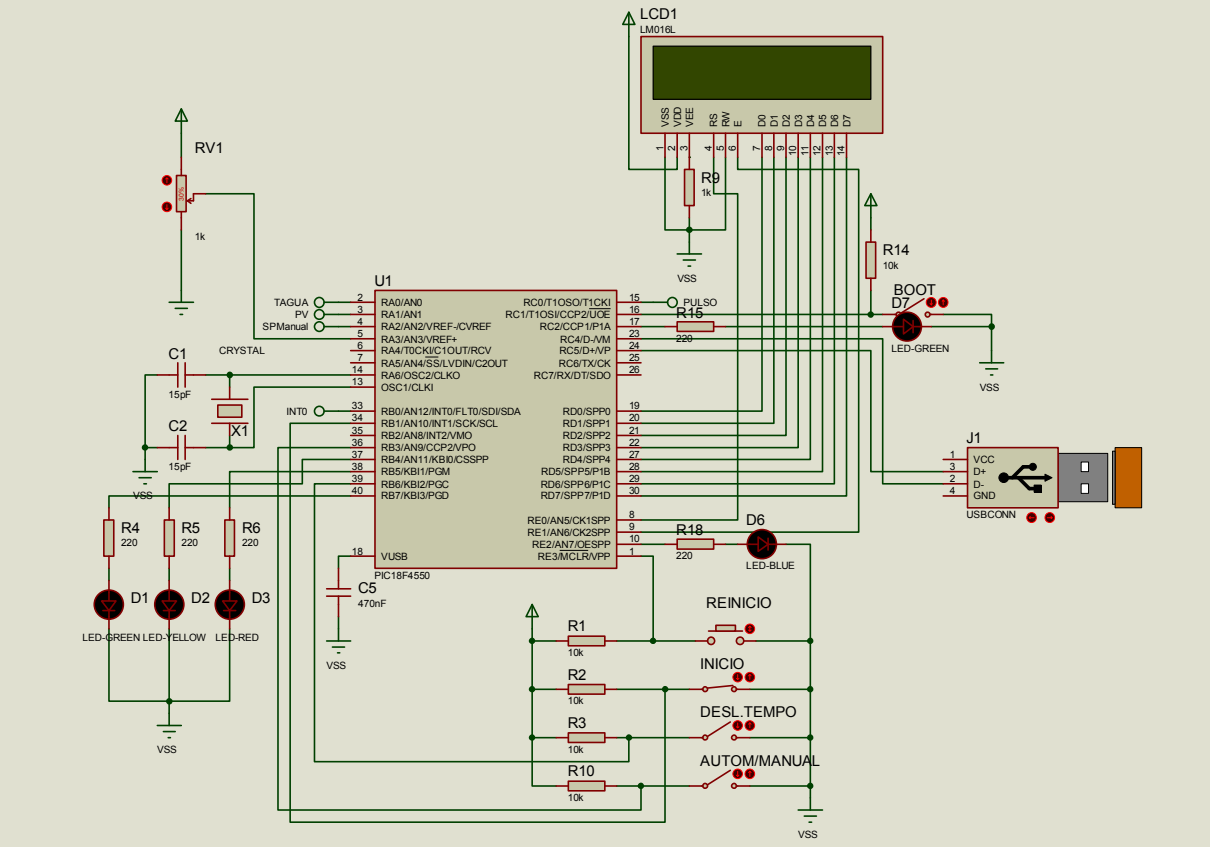

Figura 14 - Circuito de comando
Fonte: do autor.

3.1.2 Circuito detector de zero

Para que fosse possível a ativação do pulso no momento certo dentro do semiciclo da senoide (tempo Tx) e realizar um controle preciso do ângulo de fase, foi necessário implementar um circuito capaz de perceber cada instante em que o sinal da rede elétrica passa pelo ponto de zero volt. Uma forma simples foi utilizando o integrado *LM741*, um amplificador operacional, que, na configuração mostrada na FIG. 15, gera uma onda quadrada positiva, em que as bordas de subida e descida coincidem com as passagens por zero da rede. Isto é suficiente para gerar a

interrupção externa *int_ext*, onde está conectada a saída deste circuito. A FIG. 16 mostra a onda quadrada (em azul) com seus respectivos pontos de subida e descida de borda.

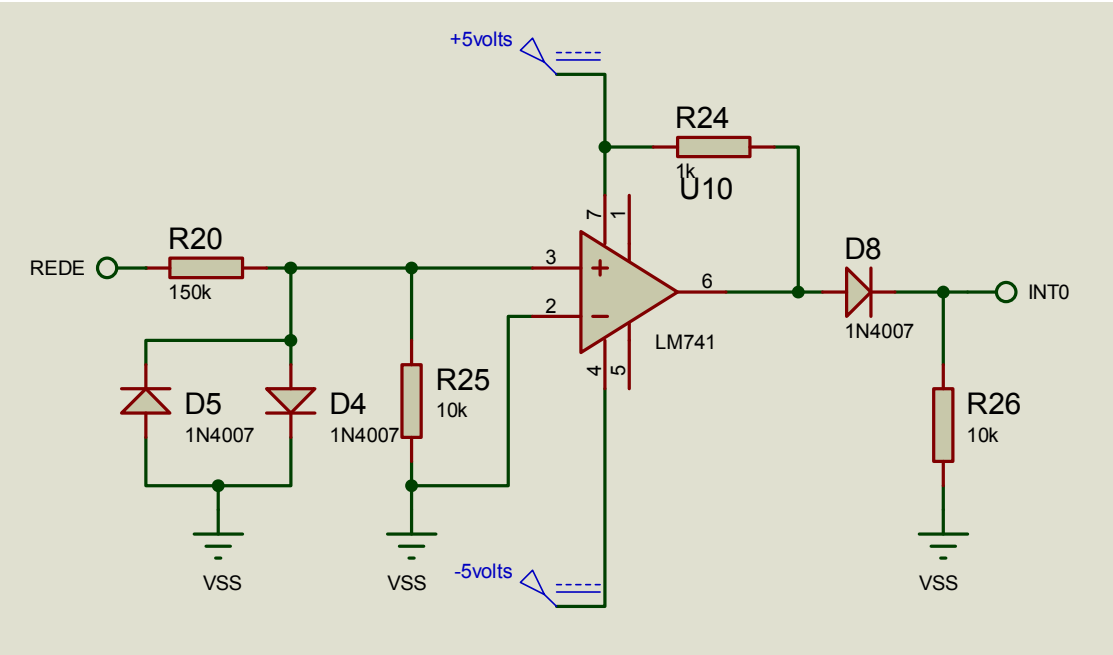

Figura 15 - Circuito para detecção de zero da rede elétrica
Fonte: do autor.

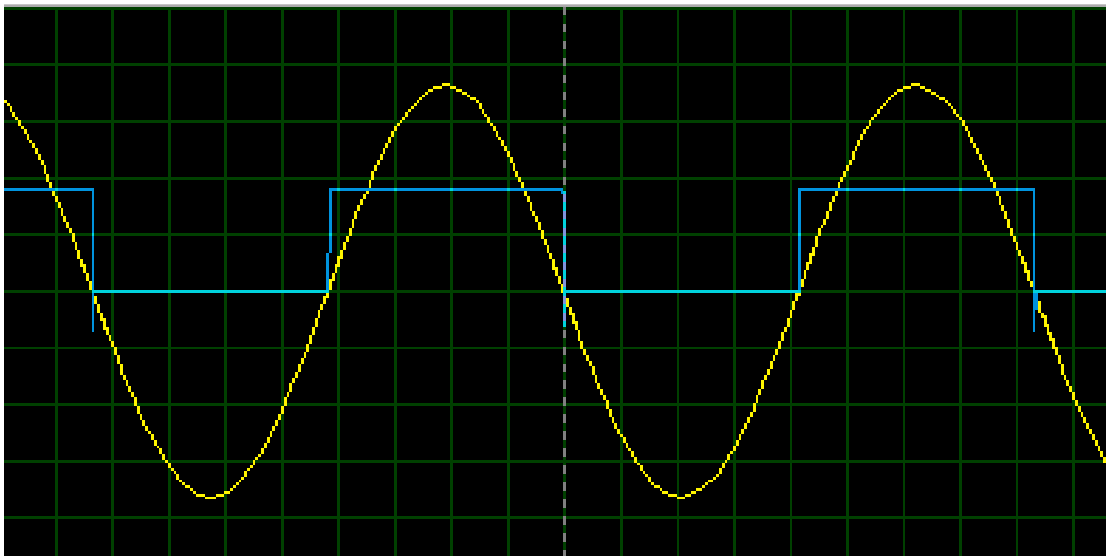

Figura 16 - Senoide e da forma de onda quadrada da detecção de zero
Fonte: do autor.

3.1.3 Circuito de potência da carga

O pulso gerado pela porta *I/O* polariza a base do transistor *NPN* BC548, que ativa o *led* interno do integrado *MOC3023*. Neste momento é gerada a corrente necessária para disparar o *TRIAC BTA41-600B*, através do *opto-diac* interno, que permanecerá neste estado até que a corrente de manutenção entre seus terminais caia a zero novamente. Isto ocorrerá na próxima passagem por zero da rede elétrica e, assim, novo pulso é gerado para o próximo semiciclo da senoide.

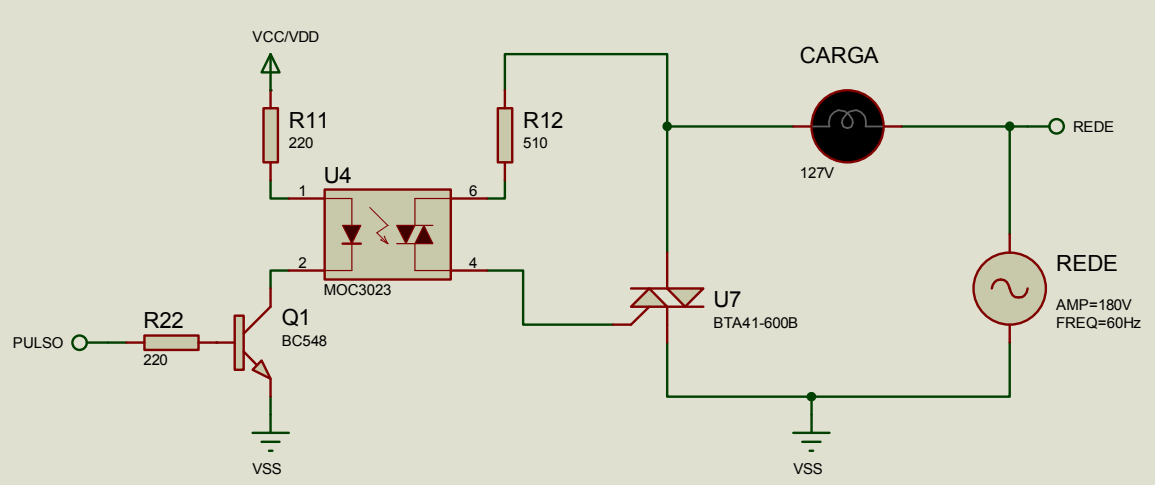

Figura 17 - Circuito de potência para controle da tensão na carga
Fonte: do autor.

A FIG. 18 mostra o momento onde o pulso (em vermelho) é gerado para um dado valor de Tx e a forma de onda da tensão aplicada no *TRIAC* (em verde).

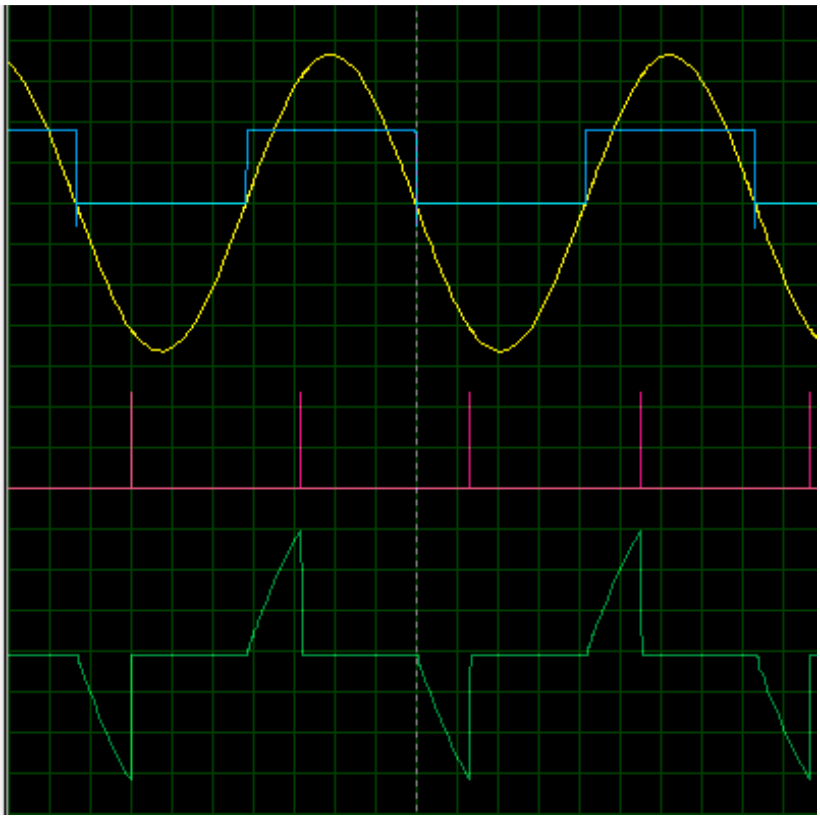

Figura 18 - Forma de onda do Pulso e da tensão no TRIAC
Fonte: do autor.

3.1.4 Circuito sensor

Na aquisição dos dados, utilizados pela função controle (temperatura da água de saída do chuveiro e temperatura da água no ambiente), foram utilizados resistores variáveis para gerar a faixa de tensão de 0 a 0,45 volts, através de um divisor de tensão. O valor de 0,45 volts é a tensão de referência para o módulo de conversão A/D trabalhar na faixa de temperatura de 0 a 45°C provenientes do sensor LM35 e do potenciômetro de ajuste do *set-point*. Esta é a faixa de temperatura de trabalho do chuveiro para o protótipo do controlador em questão.

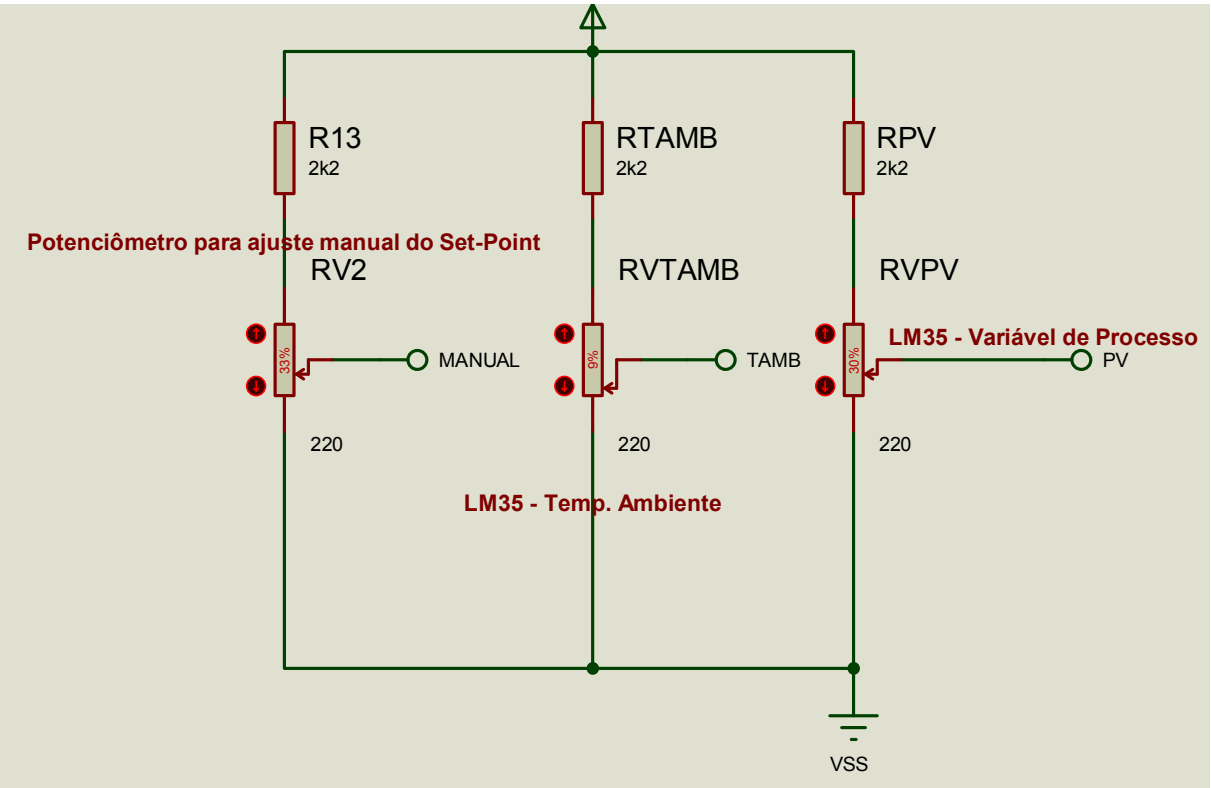

Figura 19 - Simulação dos sensores em ambiente virtual
Fonte: do autor.

3.1.5 Circuito de alimentação

Com a finalidade de evitar ruídos indesejáveis, é essencial a construção de uma fonte de alimentação confiável. Para isso, foi utilizado o regulador de tensão *LM7805*, que estabiliza a tensão de alimentação para o *PIC* em 5 volts.

Neste projeto, foi preciso também que a fonte de alimentação fosse simétrica de +5 e -5 volts, necessários para alimentar o integrado *LM741,* utilizado no circuito detector de zero. Isto foi conseguido com um diodo *zener* de 5,1 volts para a elaboração de um circuito de terra virtual, conforme mostra a FIG. 20.

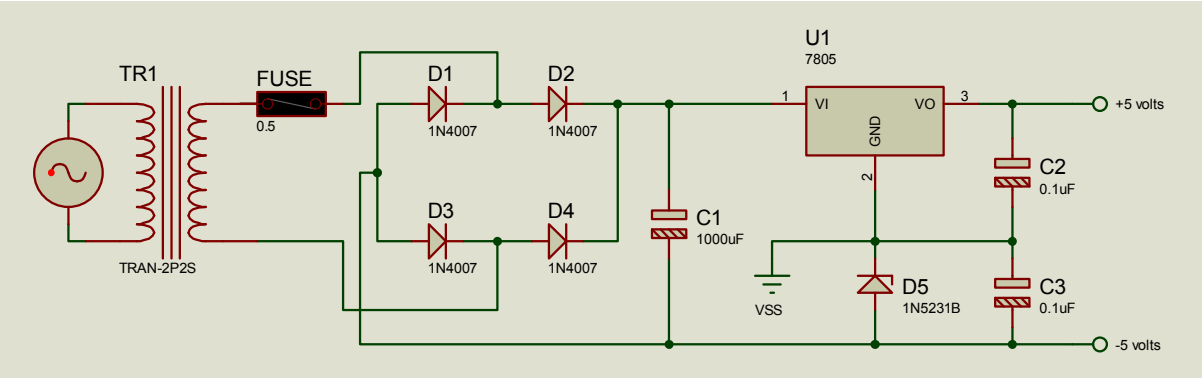

Figura 20 - Fonte de alimentação simétrica por terra virtual
Fonte: do autor.

Além disso, o uso de capacitor de desacoplamento próximo aos pinos de alimentação do *PIC* é de grande valia para estabilizar a tensão e reduzir eventuais ruídos. Para isso, foram utilizados dois capacitores de 100nF cada um, que devem ser posicionados próximos aos pinos Vdd e Vss do *PIC*, na placa de circuito impresso a ser confeccionada.

A FIG. 21 e a FIG. 22 mostram os circuitos implementados no simulador *ISIS* e utilizados para a confecção das placas de circuito impresso.

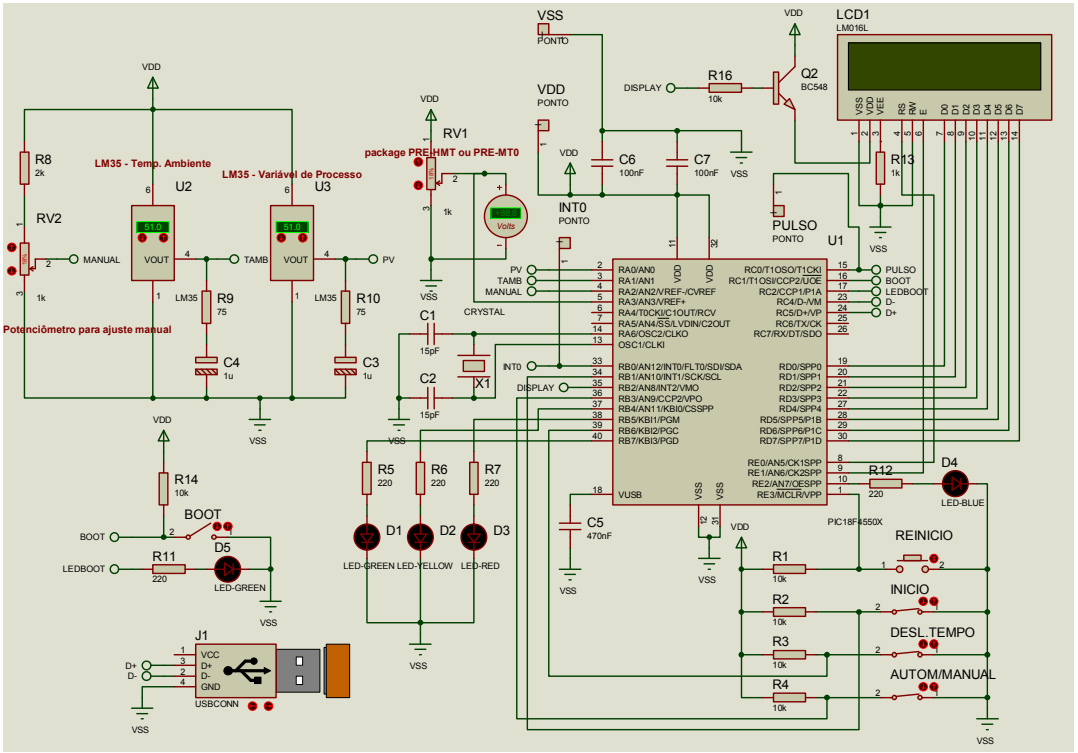

Figura 21 - Circuito para confecção de placas - parte 1
Fonte: do autor.

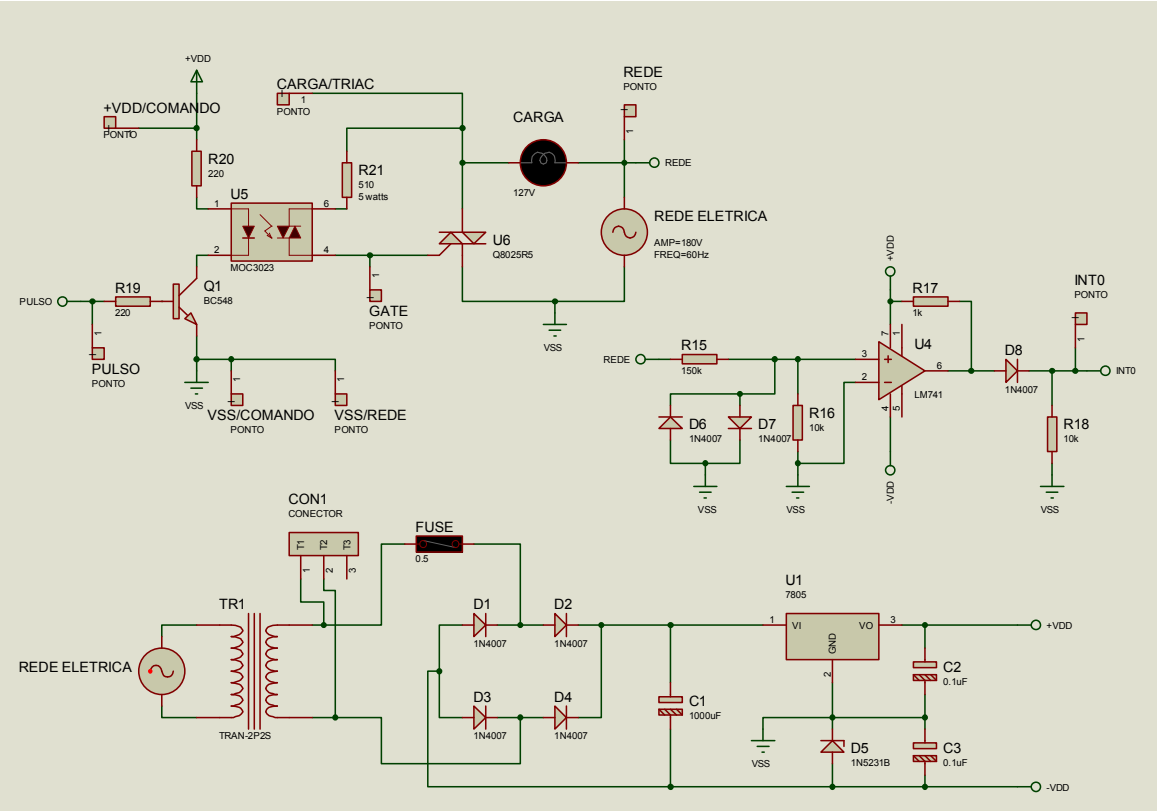

Figura 22 - Circuito para confecção de placas - parte 2
Fonte: do autor.

3.2 Os circuitos em simulação real

Após a simulação em software, foi montado o circuito com componentes reais para simulação em *protoboard*, comprovando os resultados obtidos virtualmente. Esta simulação foi importante para testar a lógica de funcionamento das chaves de comando, visualizar os valores analógicos adquiridos e convertidos dos sensores *LM35* e potenciômetro através do *LCD*, testar o funcionamento do pulso enviado ao *GATE* do *TRIAC*, visualizando o controle da tensão na carga através da luminosidade de uma lâmpada incandescente, que representou a resistência do chuveiro elétrico.

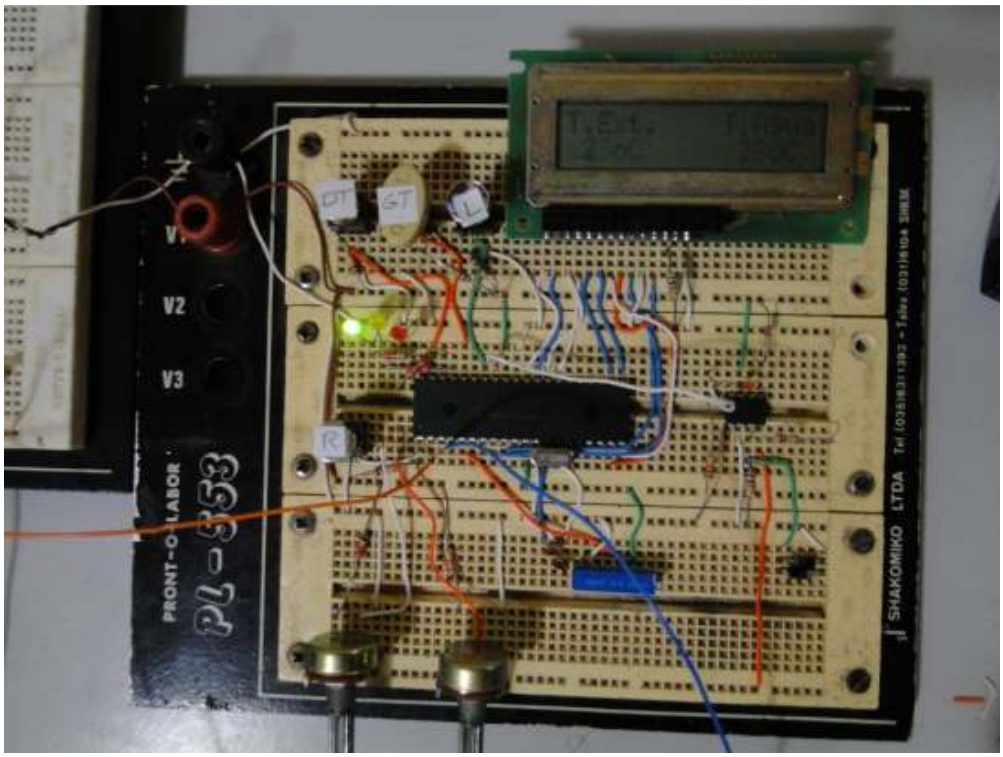

Figura 23 - Circuito de comando em protoboard
Fonte: do autor.

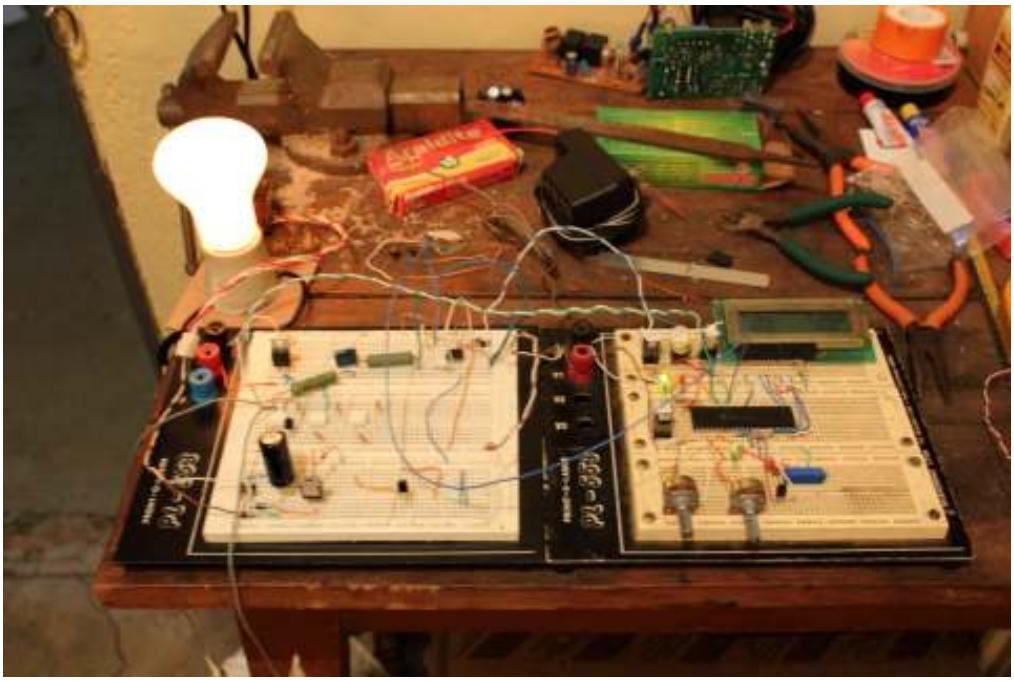

Figura 24 - Circuito do controlador em protoboard
Fonte: do autor.

3.3 Adaptações no chuveiro

Foi utilizado um modelo comercial de chuveiro elétrico, da marca *FAME*, que possui uma seleção manual de quatro temperaturas: super quente, quente, morna e fria, de potências 5400 watts, 3400 watts, 2000 watts e desligado, respectivamente. Um projeto de controlador de temperatura exige que se use um atuador de alta potência, para que o objetivo seja atingido com facilidade e rapidez. No entanto, considerando o uso do *TRIAC BTA41-600B* de 40 ampéres de corrente máxima e para o teste de funcionamento do protótipo do projeto, foi utilizada somente a potência de 3400 watts, onde temos em torno de 27 ampéres máximos. Assim, para uso com a potência de 5400 watts, o que daria uma maior variação Δ°C de temperatura para o controle, devemos utilizar um *TRIAC* que suporte correntes acima de 50 ampéres. Deste modo, foi isolado o trecho de resistência do chuveiro, de modo a aproveitar somente aquela potência.

Neste chuveiro, também foi adaptada a chave (FIG. 25) para iniciar o controle da temperatura pelo controlador. Ela foi fixada de modo a ser acionada quando o diafragma do chuveiro se movimenta de forma ascendente, com a pressão exercida pela passagem da água quando o usuário abre a torneira, como pode ser visto na FIG. 27. A posição do sensor *LM35*, que faz a medição da temperatura da água (PV), também pode ser visto nas figuras seguintes. Ele teve seus terminais elétricos devidamente isolados do contato com a água e posicionado através de um furo feito do topo para o interior do chuveiro, de modo que seu encapsulamento ficasse em contado direto com a água.

Além disso, foi fixado um LM35 para medir a temperatura da água de entrada no chuveiro, conforme mostra a FIG. 29.

Figura 25 - Chave de início
Fonte: do autor.

Figura 26 - O LM35 em encapsulamento TO-92
Fonte: www.google.com.br.

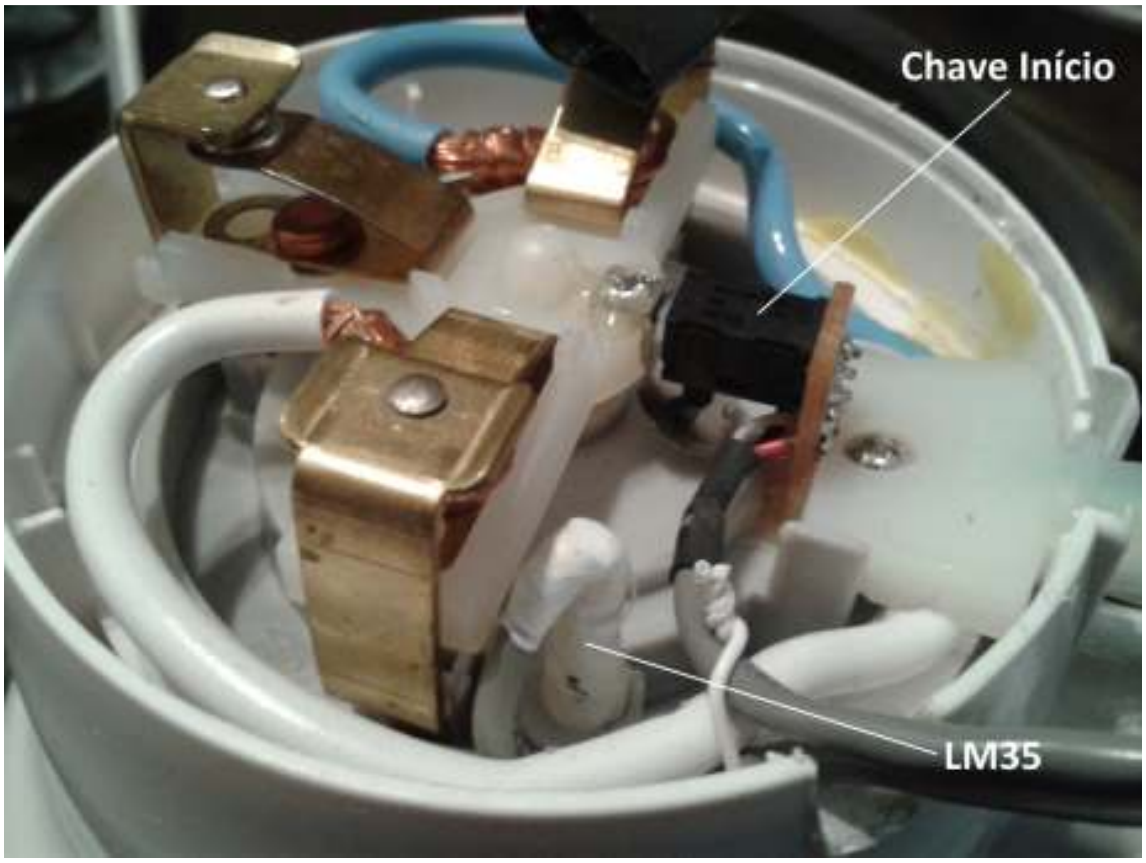

Figura 27 - Posicionamento da chave e do sensor no chuveiro
Fonte: do autor.

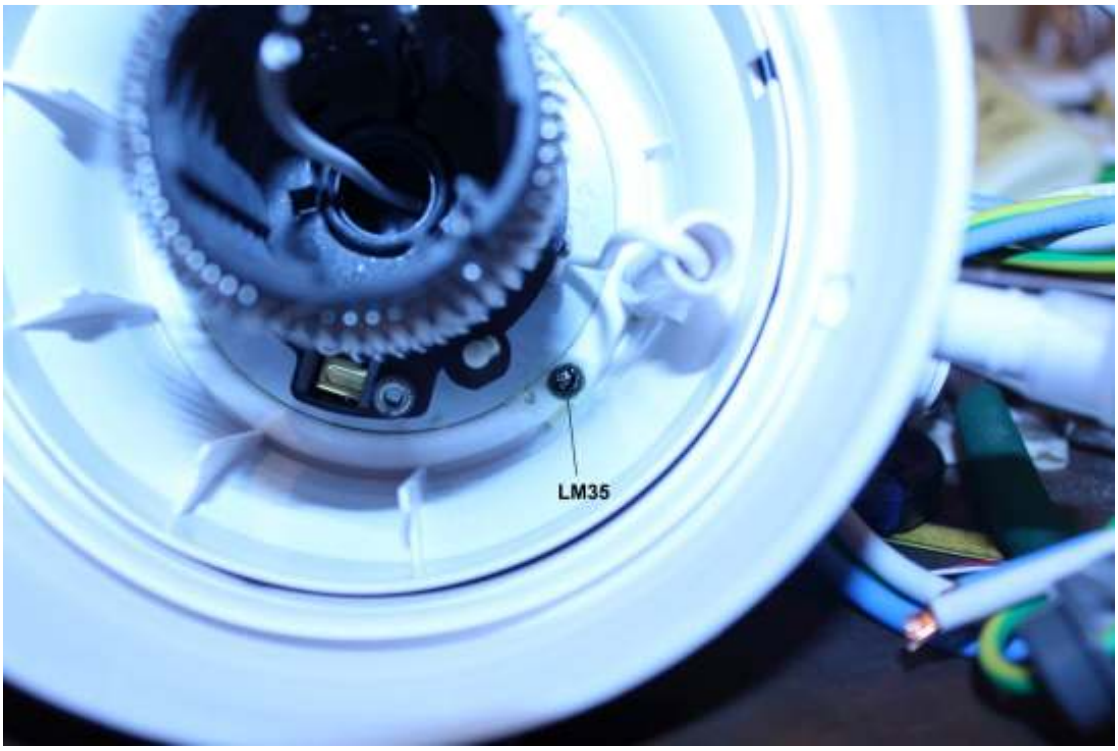

Figura 28 - Interior do chuveiro
Fonte: do autor.

Figura 29 - Fixação do sensor da temperatura da água de entrada
Fonte: do autor.

3.4 Montagem do protótipo

O circuito eletrônico para o protótipo foi montado em um conjunto composto por duas partes em circuito impresso: a fonte de alimentação simétrica com terra virtual, junto com o circuito de potência e o circuito detector de zero, e o circuito de comando, onde está o *PIC*, o *LCD*, o sensor de temperatura da água no ambiente e o potenciômetro de ajuste do *set-point*. A FIG. 30 mostra o conjunto, que também contém a lâmpada incandescente para a realização de testes.

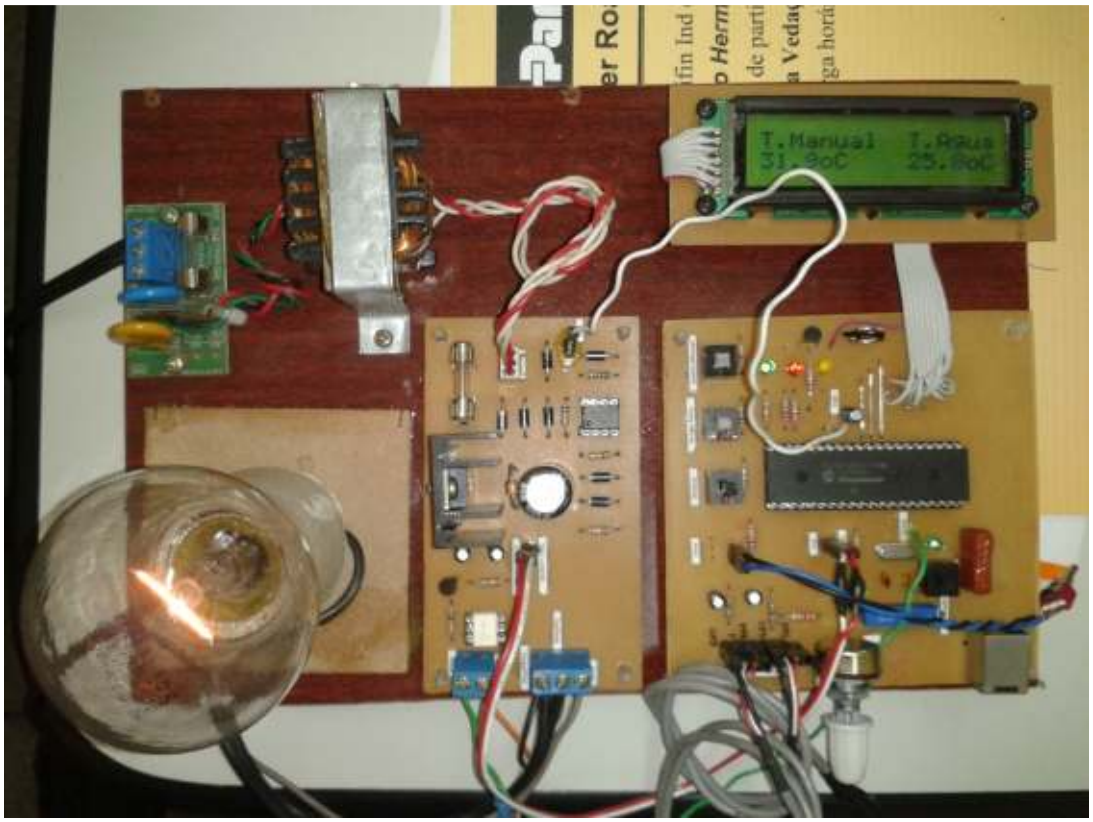

Figura 30 - Protótipo do circuito do controlador
Fonte: do autor.

A segunda parte do protótipo é composta pelo chuveiro, que teve as adaptações feitas conforme seção anterior, e o atuador *TRIAC BTA41-600B* (FIG. 31). Este atuador, por trabalhar diretamente com a alta corrente de alimentação do chuveiro, necessita de dissipação para o calor gerado pelo efeito *joule* (fenômeno de transformação de energia elétrica em energia térmica) durante o funcionamento do controlador. Sendo assim, foi utilizado na entrada de água do chuveiro, um cano de alumínio, onde foi fixado o atuador, devidamente isolado de contato elétrico através de placa de mica e envolto em pasta térmica para melhor dissipação do calor, conforme mostra a FIG. 32.

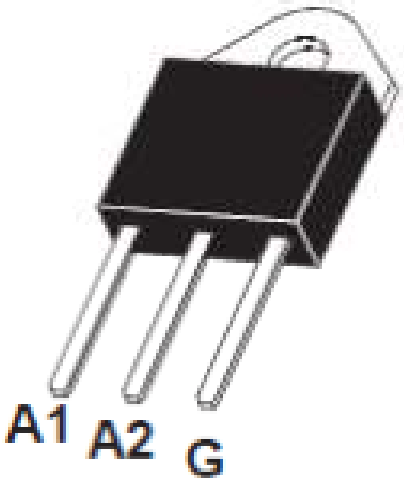

Figura 31 - *TRIAC BTA41-600B*
Fonte: www.google.com.br.

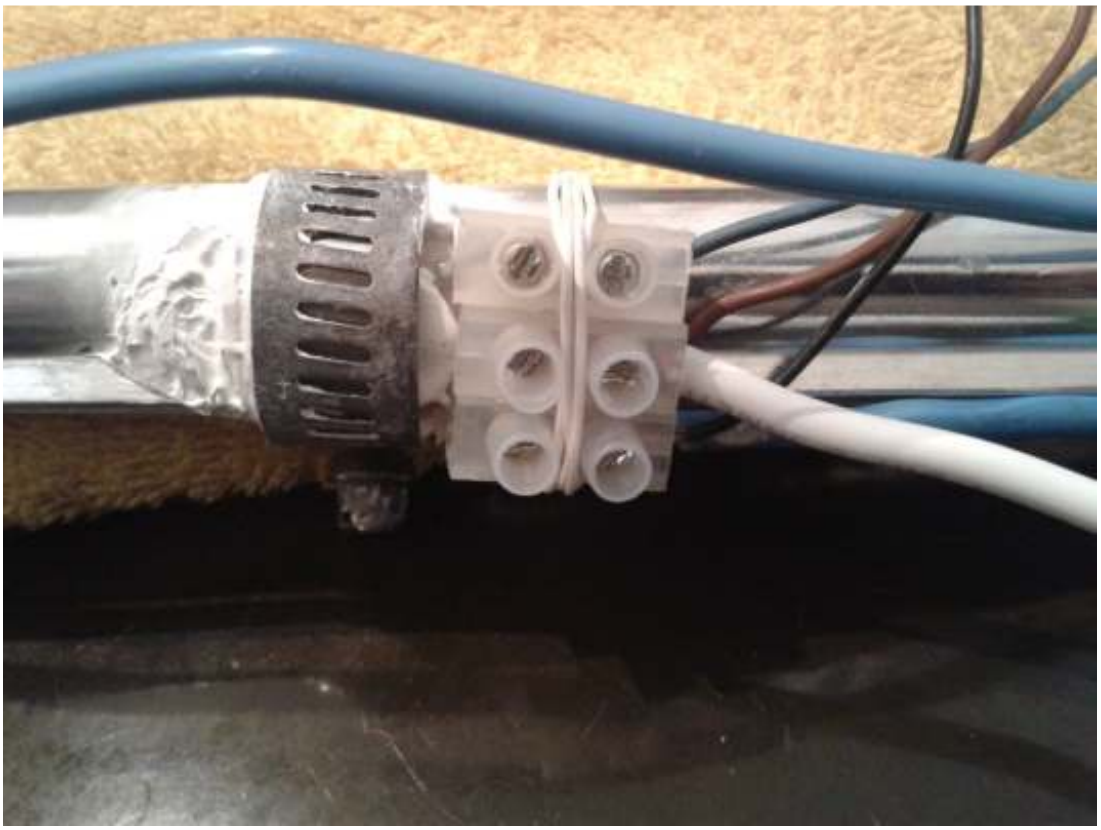

Figura 32 - Fixação do atuador (TRIAC) para dissipação do calor
Fonte: do autor.

Segundo seu fabricante, a *STMicroelectronics*[16], o calor gerado pelo *TRIAC BTA41-600B* segue o gráfico da FIG. 33.

[16] STMicroelectronics, 2001.

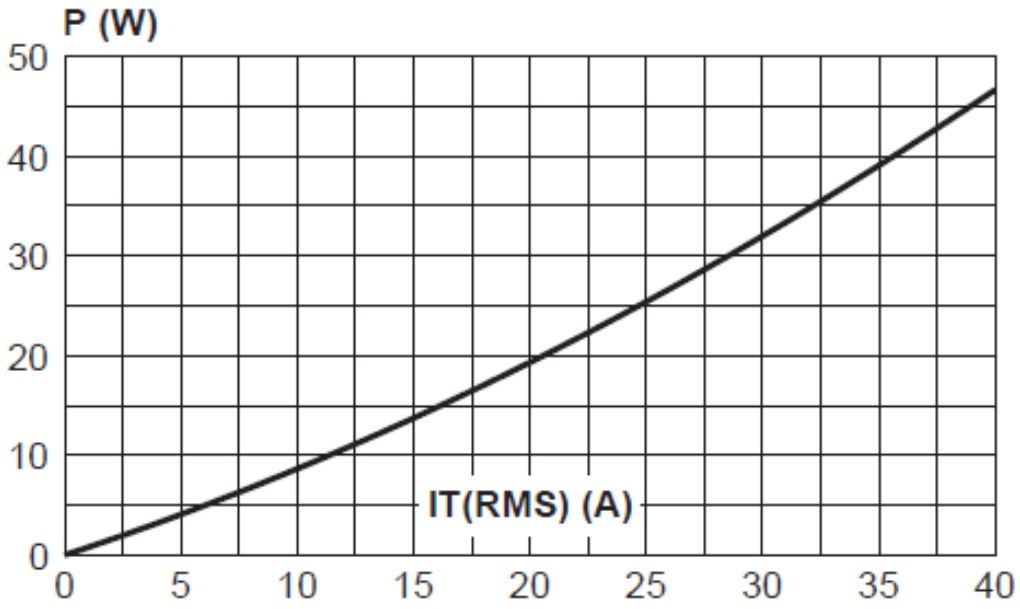

Figura 33 - Máxima dissipação de potência pela corrente RMS no TRIAC
Fonte: STMicroelectronics, 2001. p.3.

Sendo assim, trabalhando na potência do chuveiro escolhida para o projeto (3400 watts), onde a corrente é de aproximadamente 27 ampéres, será gerado um máximo de 30 watts de calor por efeito *joule*. Este valor é apenas 0,88% da potência utilizada pelo chuveiro. Assim, considerando as mesmas condições de vazão da água e o ganho de temperatura com a potência de 3400 watts[17], se aplicarmos uma potência de 30 watts no chuveiro, o ganho de temperatura seria, sem considerar as perdas, de apenas 0,16ºC, aproximadamente. Esta variação pré-aquece a água de entrada no chuveiro, conforme fixação do TRIAC mostrada na FIG. 32, não sendo desperdiçada para o ambiente, porém, não representa um valor significativo.

3.5 Comunicação para aquisição dos dados

O protótipo deste controlador possui a implementação de uma comunicação serial, emulada através de uma porta *USB*, utilizando o protocolo *USB-CDC*. Com ela, foi possível a aquisição de dados pertinentes ao controle PI para a elaboração de gráficos, úteis na sintonia do controlador.

[17] Vazão = 2,0 litros/min e ΔTm = 19ºC, veja seção 2.4 Sistema de controle.

Para isso, foi ativado o módulo de comunicação *USB* do *PIC18F4550,* utilizando as portas D+ e D- (pinos 24 e 23, respectivamente) e as bibliotecas *usb.h* e *usb_cdc.h* necessárias, contidas no compilador *PIC*-C da *CCS Inc.*. A FIG. 34 mostra as ligações para a comunicação *USB*, usada também para o recurso do modo *BOOTLOADER*, que nos possibilita gravar o *firmware* (programa do controlador, por exemplo) no *PIC* sem a necessidade de retirá-lo do circuito. Este modo foi muito útil para gravar várias vezes as alterações nos parâmetros do programa sempre que fosse necessário testar, bastando, para isso, conectar o cabo *USB*, ativá-lo através das chaves *Reset* e *Boot* e carregar o arquivo hexadecimal gerado pelo compilador. A gravação é feita através do programa *PDFSUSB* da *Microchip*. Porém, antes disso, devemos usar um gravador de PIC (*PIC-Kit 2*, por exemplo) uma primeira vez, para gravar o código do *BOOTLOADER*, também disponibilizado pela *Microchip*.

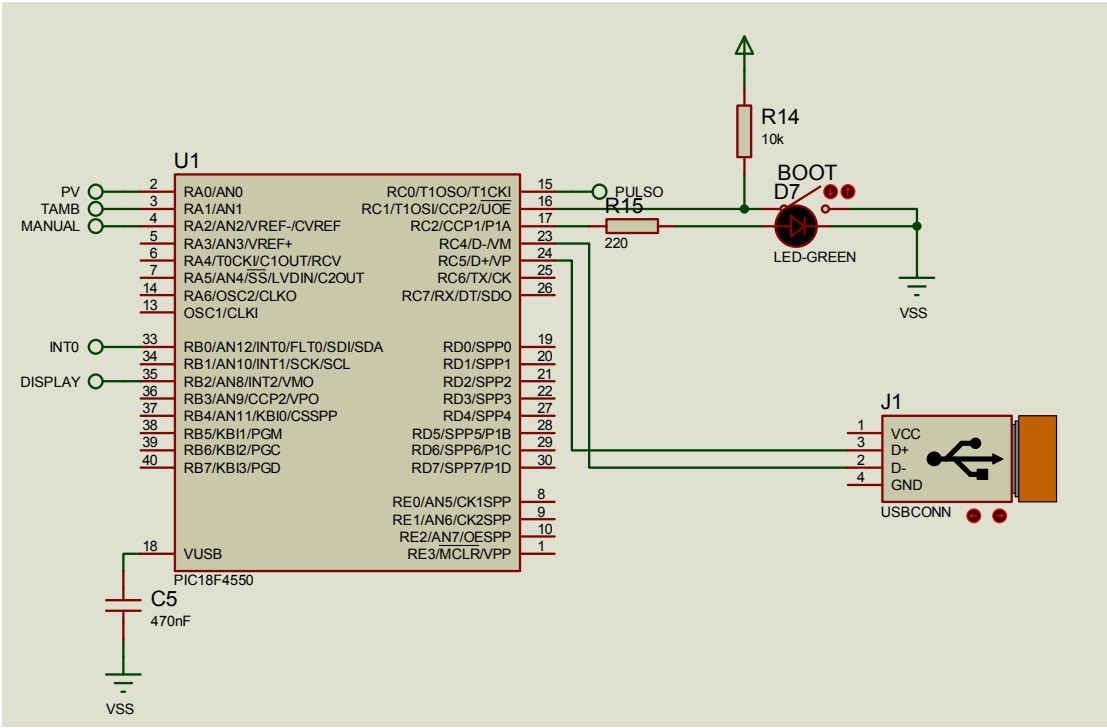

Figura 34 - Comunicação USB
Fonte: do autor.

Com a ajuda do TIMER 3, é feita uma temporização de 500ms para o envio dos dados pela *USB*. Assim, os dados tempo do processo, temperatura da água no ambiente (Tamb), set-point (SP), temperatura da água (PV) e tensão aplicada na carga (TC), são enviados a cada 0,5 segundos, conforme o trecho de código da função enviar_dados() seguinte.

```
void enviar_dados()
{
        printf(usb_cdc_putc_fast,"%f%c%f%c%f%c%f%c%f%c",
                contatempoenviototal,' ',Tamb,' ',SP,' ',PV,' ',TC,'@');                    }
```

Para a recepção dos dados, foi elaborado um programa em *Visual C#* (código fonte no APÊNDICE C) que, através de uma função de leitura contínua da porta *COM* associada ao controlador pelo sistema operacional do microcomputador, armazena os dados enviados durante a execução do controle pelo *PIC*. Estes dados são tratados e usados para gerar os gráficos para análise do controle. As FIG. 35 e FIG. 36 mostram as telas do programa em *Visual C#* para a recepção dos dados e envio de alguns comandos ao *PIC*.

Figura 35 - Tela para conexão com o controlador
Fonte: do autor.

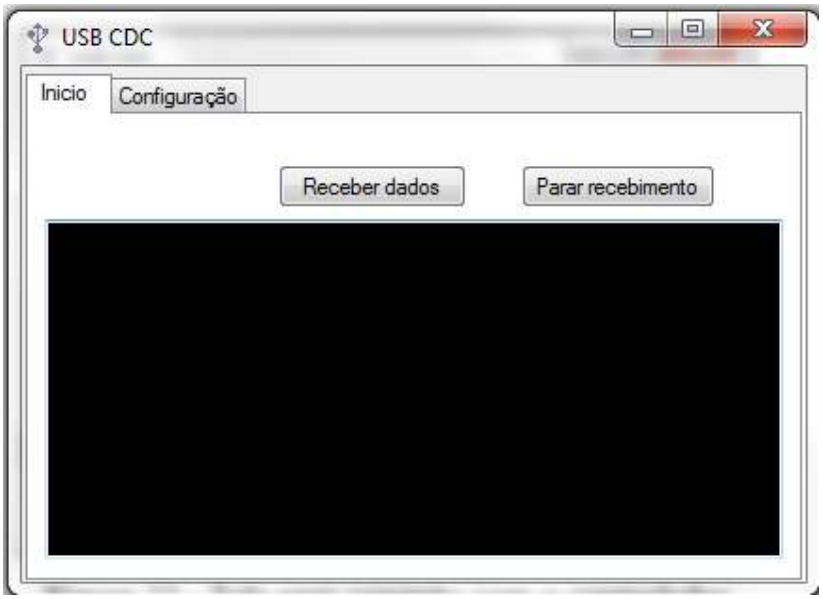

Figura 36 - Tela de recebimento dos dados
Fonte: do autor.

No programa do *PIC* devem ser estabelecidos os parâmetros para a comunicação serial com a diretiva #use rs232 ("parâmetros"). Para este projeto foram definidos da seguinte forma:

```
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
```

Assim, no programa de recepção em *Visual C#*, deve-se tomar o cuidado em configurar estes mesmos parâmetros para que seja possível o estabelecimento da comunicação entre o microcomputador e o *PIC*.

3.6 Testes e análise dos resultados

Os testes para o controle em malha fechada foram realizados conforme montagem do protótipo e adaptações no chuveiro apresentados no item 2 e utilizando uma vazão fixa de 2 litros por minuto de água.

Desta forma, os parâmetros calculados no item 2.4, através da resposta ao degrau unitário e sintonia, foram aplicados ao controlador PI paralelo clássico.

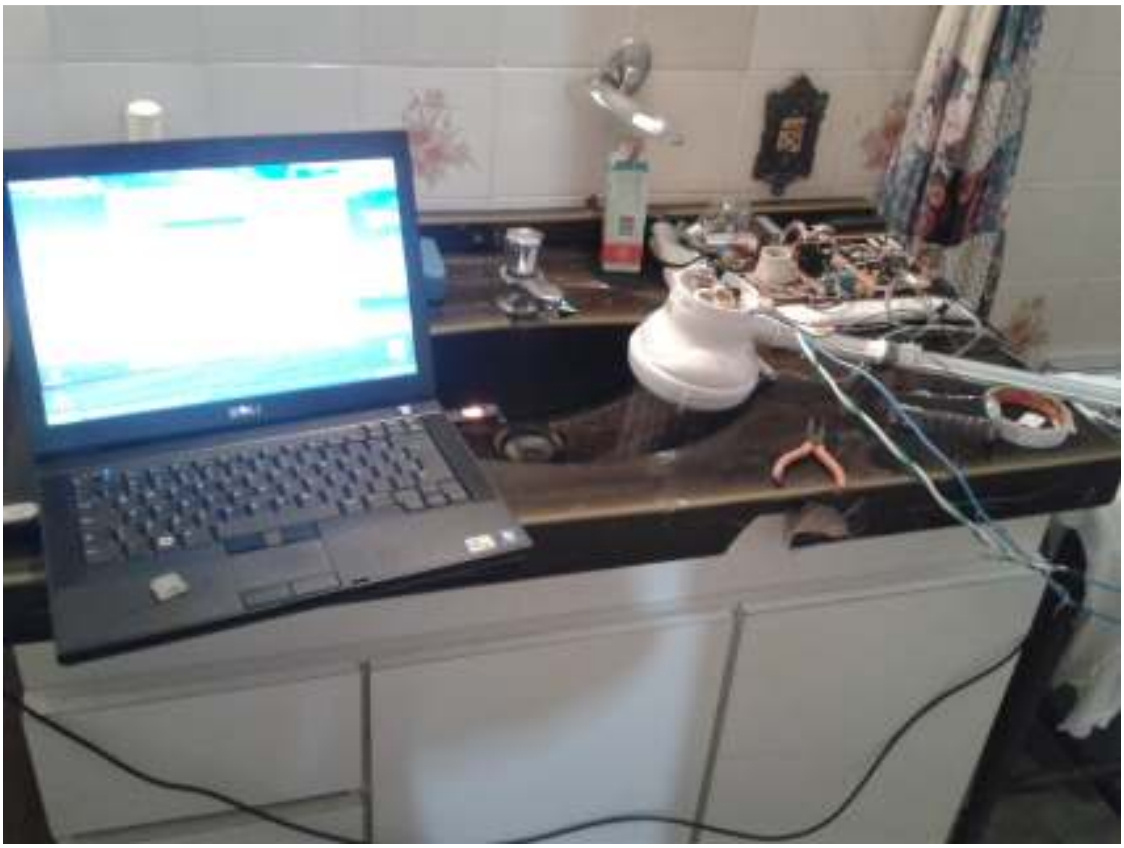

Figura 37 - Montagem do protótipo para aquisição dos dados
Fonte: do autor.

O primeiro teste realizado utilizou os valores de Kp e Ki conseguidos diretamente do método de Ziegler e Nichols, que são, respectivamente, 8,198 e 0,119. Porém, como estamos utilizando um controlador digital, devemos levar em conta a taxa de amostragem em que o controlador executará o termo integral do controle PI. Sendo assim, aplicamos ao programa do PIC a equação 15 do item 2.4.2, através da função controle() em linguagem C, conforme pode ser visto no código apresentado no APÊNDICE B.

O resultado obtido com este teste pode ser visto no gráfico da FIG. 38.

Vemos por este gráfico que com o *set-point* em 37,44°C obtivemos um tempo de ascensão (ou subida) Ts, que é o tempo em que a temperatura da água de entrada atinge o *set-point*, de 24,14 segundos, um sobrevalor (ou *overshoot*) Ov, que é o valor máximo da temperatura atingida pela água, de 12,18% do valor do *set-point*, e um tempo de assentamento (ou acomodação) Ta, que é o tempo em que o desvio em regime permanente é menor que 5% entre o valor da temperatura da água e o

set-point, de, aproximadamente, 69 segundos, sendo o desvio Dv de 2,27%. Tais valores foram obtidos de acordo com as equações que seguem abaixo.

$$Ov = \frac{Tmáx - SP}{SP} 100 \quad [\%] \quad Ov = \frac{42 - 37{,}44}{37{,}44} 100 = 12{,}18\% \tag{27}$$

$$Ts = pelo\ gráfico \quad [s] \quad Ts = 24{,}14\ s \tag{28}$$

$$Dv = \frac{SP - Tempd}{SP} 100 \quad [\%] \quad Dv = \frac{37{,}44 - 36{,}59}{37{,}44} 100 = 2{,}27\% \quad Ta = pelo\ gráfico = \ 69\ s \tag{29}$$

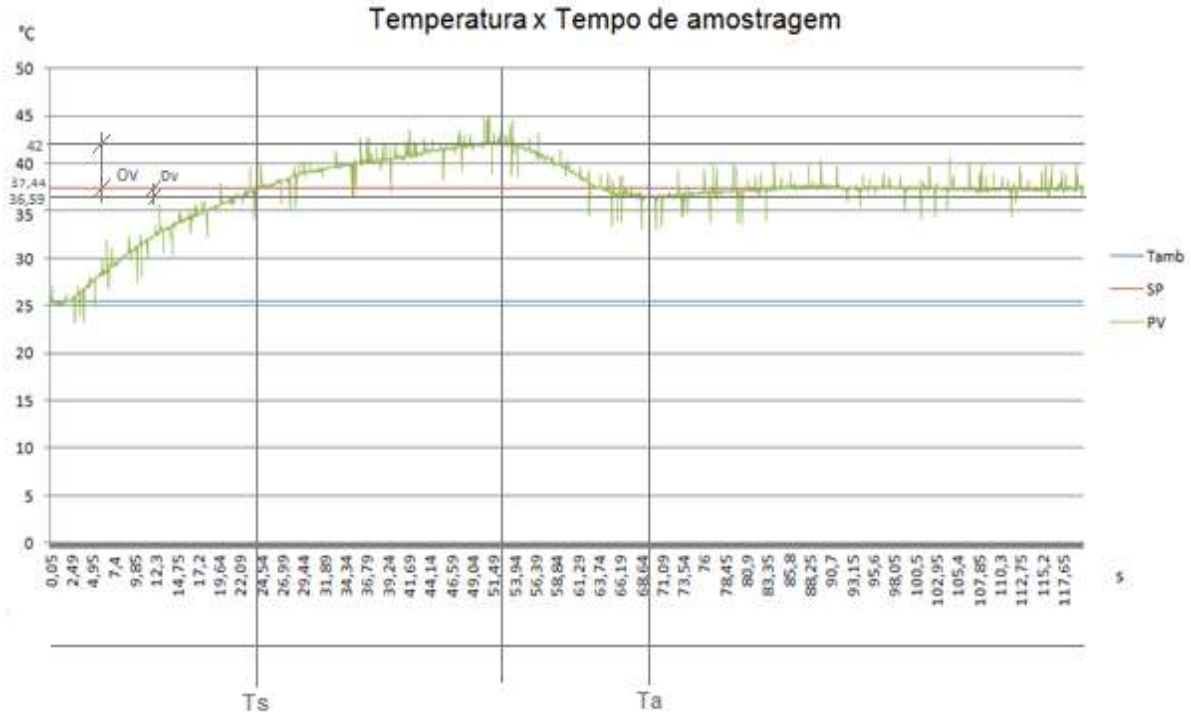

Figura 38 - Valores obtidos pelo método Ziegler e Nichols
Fonte: do autor.

Para melhorar os resultados e conseguir um menor Ts, um menor Ov e um menor Ta, seguimos conforme exemplificação de OGATA[18] para realizar uma sintonia fina depois de observações do processo.

Desta forma, movemos o zero do controlador PI para s = -0,0015 e dobramos o valor da constante proporcional Kp para 16,396.

[18] OGATA, 2006. p.562.

Assim, a equação para o controlador ficou da seguinte forma (equação 30):

$$G_c(s) = \frac{16{,}396(s + 0{,}0015)}{s} \tag{30}$$

E a equação para o sistema em malha fechada (equação 31):

$$\frac{Y(s)}{SP(s)} = \frac{196{,}752s + 0{,}295128}{57{,}5s^3 + 25{,}5s^2 + s} \tag{31}$$

Observamos que o valor do módulo de s é o valor de Ki multiplicado pela taxa de amostragem TA, conforme mostra a equação 23.

Aplicando estes novos valores ao controlador PI do protótipo conseguimos os seguintes resultados, como mostra o gráfico da FIG. 39.

$Ov = \frac{Tmáx - SP}{SP} 100 = \frac{39{,}80 - 37{,}88}{37{,}88} 100 = 5\%$

$Ts = pelo\ gráfico = 20\ s$

$Dv = \frac{Tempd - SP}{SP} 100 = \frac{39{,}52 - 37{,}88}{37{,}88} 100 = 4{,}32\% < 5\%,$

logo $Ta = pelo\ gráfico = 30\ s$

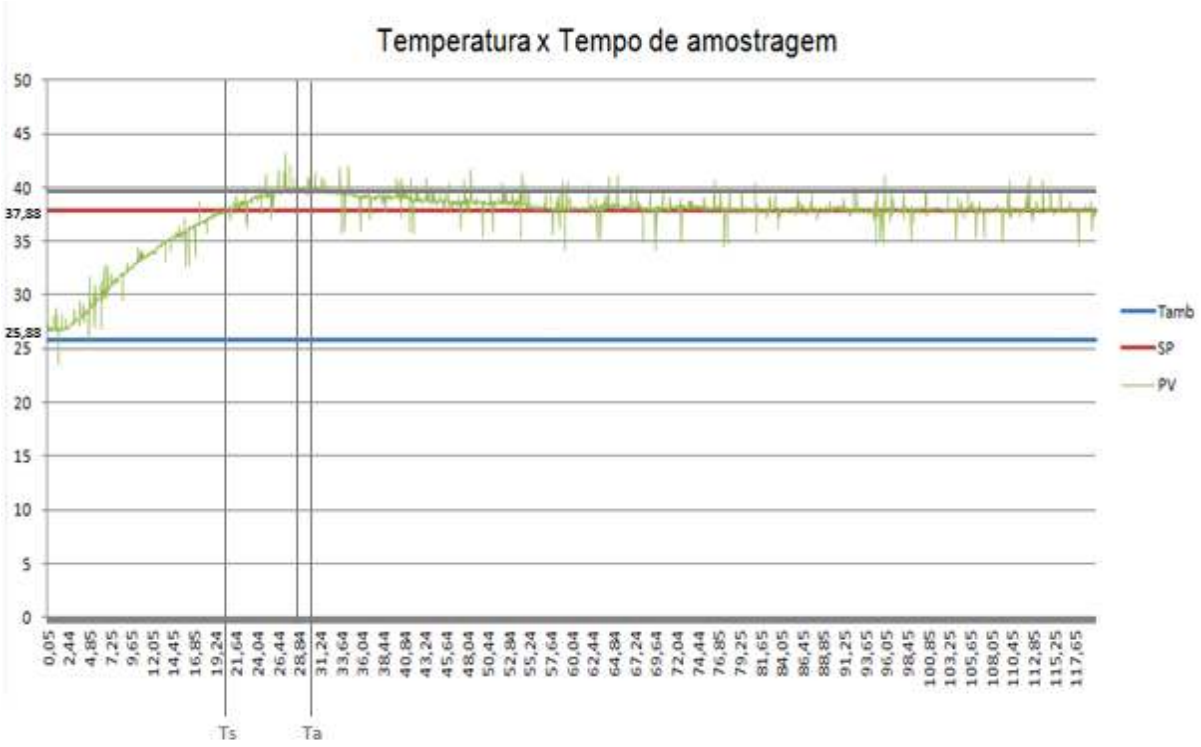

Figura 39 - Valores obtidos por sintonia fina
Fonte: do autor.

Conforme pode ser visto nos valores, obtivemos uma melhora significativa no controle.

No ambiente onde foi testado o protótipo há instalado um sistema de aquecimento solar que gasta em torno de 1 minuto e meio para entregar a água em uma temperatura de 38°C na saída do chuveiro. Sendo assim, se o usuário espera até que a água atinja este valor antes de começar o banho e considerando a vazão de 2 litros por minuto, seriam desperdiçados 3 litros de água. Com o controlador instalado e considerando esta mesma temperatura, o tempo gasto seria de 30 segundos (tempo de assentamento do controlador). Desta forma, a perda de água seria, aproximadamente, 66% menor.

Este percentual, logicamente, será diferente para outros ambientes, pois dependerá do comprimento da tubulação que liga o chuveiro ao reservatório do aquecedor solar. De qualquer forma representará uma economia de água e energia elétrica.

O tempo de banho pode ser fixado em um valor entre 5 e 10 minutos, conforme resultado da pesquisa (Questão 9). Porém, para uma máxima economia de água e energia elétrica, o ideal é fixar em 5 minutos. Neste caso, o consumo de água seria em torno de 10 litros de água por banho.

Devemos lembrar que para uma efetiva economia de água e um controle ótimo, devemos fixar a vazão de água em um menor valor, através de uma torneira para este fim.

4 CONCLUSÃO

Analisando os resultados obtidos nos testes em malha fechada do controlador proposto e o resultado da pesquisa de campo realizada para estabelecer um controle automático, pudemos perceber que é perfeitamente possível realizar uma economia de água potável e energia elétrica, conciliando a consciência das pessoas no assunto à ajuda da tecnologia disposta a nosso favor.

Com este projeto de pesquisa, percebemos a possibilidade de expandir o uso do controlador para uso em conjunto com um aquecedor solar, representando de fato uma economia de água pela redução do tempo total do banho.

Vimos que a utilização do calor gerado pelo circuito de potência para um pré-aquecimento da água não representou um ganho significativo de temperatura, apesar de que não está sendo perdido para o ambiente e sim dissipado na água através da tubulação de entrada.

E vimos também a possibilidade de construção do controlador utilizando uma menor quantidade de componentes no circuito de potência, tornando-o mais simples, através de um maior aproveitamento dos recursos do microcontrolador para a geração do sinal de controle.

5 TRABALHOS FUTUROS

Fica como sugestão para trabalhos futuros:

- aplicação do controlador em modelo de chuveiro mais potente.
- implementação de válvula de controle de vazão para a água fria, quando o controlador for usado juntamente com um sistema de aquecimento solar.
- medição e controle da vazão da água através da diferença de temperatura entre a água de entrada e de saída do chuveiro.
- realização de uma sintonia fina dos parâmetros do controlador através do uso de métodos computacionais.

REFERÊNCIAS BIBLIOGRÁFICAS

BOYLESTAD, Robert L.; NASHELSKY, Louis. **Dispositivos Eletrônicos e teoria de circuitos**. 8. ed. São Paulo: Pearson Prentice Hall. 2004. 672 p.

CAMPOS, Mario Cesar M. Massa de; TEIXEIRA, Herbert C. G. **Controles típicos de equipamentos e processos industriais**. 1. ed. São Paulo: Edgard Blücher. 2006. 396 p.

CORREA, Paulo Vicente. **Fundamentos de Instrumentação e Controle**. 2002. 163 p. Disponível em: <http://www.adjutojunior.com.br/controle/fundamentos_ instrumentacao_controle.pdf>. Acessado em: 01.11.2012

Guia do Hardware. **Processadores RISC x Processadores CISC**. 2007. Disponível em: <http://www.hardware.com.br/artigos/risc-cisc/>. Acessado em: 27.09.2012.

MIYADAIRA, Alberto Noboru. **Microcontroladores PIC18: Aprenda e programe em linguagem C**. 1. ed. São Paulo: Érica. 2010. 400 p.

Nacional Semiconductor. **LM35 Precision Centigrade Temperature Sensors**. 2000. 13 p. Disponível em: <https://www.national.com/ds/LM/LM35.pdf>. Acessado em 25.02.2012.

OGATA, Kasuhiko. **Engenharia de Controle Moderno**. 4.ed. São Paulo: Pearson Prentice Hall. 2003. 788 p.

PEREIRA, Fábio. **Microcontroladores PIC: Programação em C**. 7. ed. São Paulo: Érica. 2009. 358 p.

SMANIOTO, Thiago R. **Calor**. Revista Super Interessante. 47. ed. 1991. Disponível em: <http://super.abril.com.br/saude/sentimos-calor-insuportavel-quando-temperatura-externa-atinge-mais-36-graus-c-se-media-normal-nosso-corpo-esta-36.shtml>. Acessado em: 26.10.2012.

STMicroelectronics. **BTA40 and BTA/BTB41 Series 40A TRIACS**. 2001. 6 p. Disponível em: <www.datasheetcatalog.com>. Acessado em: 23.10.2012.

Teoria de controle PID. Disponível em: <http://pt.scribd.com/doc/16955622/Teoria-de-Controle-PID>. Acessado em: 20.02.2012.

Teoria do TRIAC. Disponível em: <http://www.sonelec-musique.com/electronique_theorie_triac.html>. Acessado em: 10.04.2012.

Trilhas e Rumos. **Considerações sobre o frio**. 2011. Disponível em: <http://www.trilhaserumos.com.br/canada/arquivos/consideracoes-sobre-o-frio>. Acessado em: 27.10.2012.

APÊNDICE A - PESQUISA DE CAMPO

Para a obtenção dos dados referentes à preferência de temperatura da água para o banho, considerando a temperatura em que se encontra o ambiente, foi realizada uma pesquisa entre funcionários, professores e alunos do Campus-IV/CEFET-MG, Colégio Dom Bosco e AC Agromercantil. Estes dados foram usados para determinação de valores que serão assumidos pelo *set-point* do controlador, quando o mesmo estiver no modo automático de controle da temperatura.

Seguem abaixo, portanto, as perguntas que foram feitas aos usuários, e, logo em seguida, o resultado da pesquisa, conforme tabulação.

Tabulação

Tabela 3 - Temperatura ideal da água no inverno na versão dos entrevistados.

Questão 1	f	%
Mais alta	32	43,24
Morna	7	9,46
Entre a morna e a quente	35	47,30
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Tabela 4 - Temperatura ideal da água no verão na versão dos entrevistados.

Questão 2	f	%
Mais alta	2	2,70
Mais baixa	32	43,24
Morna	37	50,00
Entre a morna e a quente	3	4,05
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Tabela 5 - Forma usual de controle de vazão da água de acordo com os entrevistados.

Questão 3	f	%
Nunca, deixo sempre toda aberta	8	10,81
Nunca, deixo sempre quase fechada	2	2,70
Sempre regulo	58	78,38
Abro sempre pela metade	6	8,11
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Tabela 6 - Desconforto térmico quando em contado com a água muito quente em dias muito frios segundo os entrevistados.

Questão 4	f	%
Sim, mas prefiro assim	16	21,62
Sim, por isso não o faço	21	28,38
Não	33	44,59
Não sei	4	5,41
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Tabela 7 - Hábito de esperar o aquecimento da água antes de iniciar o banho de acordo com os entrevistados.

Questão 5	f	%
Sim, mas somente quando está muito frio	36	48,65
Sim, sempre espero	30	40,54
Não, nunca espero	8	10,81
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Tabela 8 - Preferência pela temperatura da água em dias amenos segundo os entrevistados.

Questão 6	f	%
Bem quente	3	4,05
Morna pra quente	25	33,78
Melhor só morna	38	51,35
Mais fria que morna	8	10,81
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Tabela 9 - Consciência do gasto de água em banhos demorados de acordo com os entrevistados.

Questão 7	f	%
Totalmente	34	45,95
Parcialmente	30	40,54
Não sei sobre esse assunto	10	13,51
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Tabela 10 - Consciência da importância de economia de água potável segundo os entrevistados.

Questão 8	f	%
Totalmente	57	77,03
Parcialmente	16	21,62
Não sei sobre esse assunto	1	1,35
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Tabela 11 - Tempo de banho segundo os entrevistados.

Questão 9	f	%
Menos de 5 minutos	7	9,46
Entre 5 e 10 minutos	52	70,27
Mais de dez minutos	15	20,27
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Tabela 12 - Utilização de um controlador de tempo de banho e de temperatura da água de acordo com os entrevistados.

Questão 10	f	%
Sim	53	71,62
Não	21	28,38
Total	74	100,00

Fonte: Dados coletados da pesquisa.

Analisando as respostas da pesquisa, vemos que a grande maioria (90,54%) prefere uma temperatura da água mais quente (Questão 1), mesmo que isto represente um desconforto ou não, mostrado na Questão 4.

No verão, 50% preferem a temperatura morna da água e 43,24% já preferem a temperatura da água mais baixa (Questão 2). Temperatura morna da água está entre 30°C e 35°C.

Na Questão 3, 78,38% sempre regulam a vazão da água para obter melhor temperatura. Isto mostra o interesse pelo conforto térmico.

Na Questão 5, 40,59% das pessoas sempre esperam a água esquentar antes de entrar nela e 48,65% o fazem somente quando está muito frio. Isto comprova o desperdício de água quando se usa somente o aquecimento solar.

Pela Questão 6, vemos que as pessoas têm preferência por água morna, mesmo em dias com temperatura amena.

As respostas das Questões 7 e 8 mostram que a grande maioria tem consciência da quantidade de água gasta no banho e a necessidade de sua economia. Sendo assim, trocariam o chuveiro convencional por um temporizado e com controle de temperatura (Questão 10).

Estas análises foram feitas considerando que as pessoas não sabem o valor em °C em que se encontra a água do banho.

APÊNDICE B - CÓDIGO FONTE EM LINGUAGEM C PARA O *PIC18F4550*

Arquivo *.c

```
#include "I:\Curso Engenharia\11 periodo\TCC\Controle automatico de temperatura para chuveiro
elétrico\3 - Circ+PropInt+BootLoader+ComunUSBCDC\Codigo para ccswinxp\codigoccsxp.h"
#bit IDLEN = 0xFD3.7   //bit para configuração do modo de gerenciamento de energia do PIC
                       //em 0, modo SLEEP, CPU e Periféricos desligados. Consumo de 0,1uA.
                       //em 1, modo IDLE, CPU desligada, mas periféricos funcionando.
                       //Consumo de 5,8uA.
                       //O modo configurado é chamado pela instrução sleep();
#include <usb_cdc.h>
#include <usb.h>

#priority int_EXT1,int_TIMER0,int_TIMER1,int_TIMER3,int_EXT,int_TIMER2,int_USB

unsigned int nslave,nfuncao;     //variáveis para receber comandos via USB
int rec[2];

unsigned int32 cont=0;           //variável contador para o Timer2 (temporização).
int op=4;                        //variável indicadora das 4 posições possíveis das chaves.
int16 cont3=0;                   //variável para contagem de tempo para alternar display.
unsigned int32 cont4=0;          //variável para contagem de tempo de pausa prolongada.
int16 contatempoenvio=0;         //variável para contagem de tempo para envio de dados.
float contatempoenviototal=0;    //variável acumuladora do tempo de envio.
int16 tempocontrole=0;
int16 piscaled=0;

int flagtempo=0;                 //flag de término de temporização. Exige reinício.
int flagtempo1=0;                //
int flagtempo2=0;                //
int flagtimer=1;                 //
int flaglimpa1=0;                //flags de sinalizações diversas
int flaglimpa2=0;                //
int flaglimpa3=0;                //
int flagpausa=0;                 //
int flagpausa1=0;                //flag para evitar desligar interrupção externa
int flagpausa2=0;                //flag indicativo de que está em pausa (se 0, não está)
int flagpausa3=0;                //flag indicativo de pausa prolongada (para entrar em sleep)
int flaginicio=0;                //flag indicativo de estado inicial (Ligue o chuveiro!)
int flagedge=0;                  //flag para inversão de borda.
int flagliga=1;                  //flag indicativo de estado da chave liga
int flagalternadisplay=0;        //flag para alternar mostragem de valores de temperatura no display.
int flagcontinua=0;              //flag para continuar o programa após passagem por zero.
int flagenvio=0;                 //flag para habilitar ou desabilitar envio de dados
int flagtempoenvio=0;            //flag para tempo de envio de dados.
int flagtempocontrole=0;         //flag para possibilitar execução da função controle()

int32 valoradc0=0;               //variáveis para armazenar valores da conversão AD
int32 valoradc1=0;
int32 valoradc2=0;

float vtempo1,vtempo2;           //variáveis para verificação da temperatura desejada
//float deltaTmax=16;            //valor da variação máxima de temperatura atingida pelo chuveiro
float deltaT=12;                 //valor da variação considerada para o controle
float Taguaamb=0, SP=0, PV=0, MV=0, TC=0;  //variáveis para calculo do controle
float MVp=0, MVi=0, erro=0, erroa=0, Kp=0, Ki; //variáveis para o erro e ganho proporcional
```

```
float Tx;                       //variável que conta tempo para o pulso
int32 Valor_Set;                //variável para atribuir valor do Timer 1

void controle()
{
  if (!input(pin_b3))       //chave de seleção automático/manual
  {
    if (Taguaamb<=10)
    {
      SP = (Taguaamb+12);
    }
    if ((Taguaamb>10)&&(Taguaamb<=18))
    {
      SP = (Taguaamb+12);
    }
    if ((Taguaamb>18)&&(Taguaamb<=25))
    {
      SP = ((0.71428*Taguaamb)+18.14285);
    }
    if ((Taguaamb>25)&&(Taguaamb<=30))      //Linearização da curva para set-point Automático.
    {
      SP = (-Taguaamb+60);
    }
    if ((Taguaamb>30)&&(Taguaamb<=38))
    {
      SP = ((-0.625*Taguaamb)+48.75);
    }
    if (Taguaamb>38)
    {
      SP = 25;
    }
  }
  else
  {
    if (SP<=Taguaamb) SP=Taguaamb;                  // para limitar o valor do set-point manual
    if (SP>=(Taguaamb+deltaT)) SP=(Taguaamb+deltaT);
                        // para limitar o valor do set-point manual no valor máximo para controle
                        // A variação deltaT é de 12ºC
  }

  //Controle Proporcional/Integral

  PV = ((100*((valoradc0*0.45)/1023))-0.8);       //valor de tensão correspondente ao valor da
                                                  //temperatura da água.
                                                  // -0.8 pra corrigir diferença de valores
  erro = (SP-PV);    //o erro varia entre -150 e +150,  na faixa de 0v a 1,5v (0 a 150 graus Celsius)

  Kp = 16.396;
  Ki = 0.0015;              //Ki*TA  para TA=50ms (controlador digital)

  MVp = ((Kp*erro));

  MVi = ( ((Kp*Ki*(erro+erroa))) + MVi );  //PI paralelo clássico
  //MVi = ( ((Ki*(erro+erroa))) + MVi );      //PI paralelo tradicional
  MV = (115 - (MVp + MVi));      //aumentar temperatura significa diminuir Tx, diminuir MV.
                                 //ação indireta
  erroa = erro;

  if (MV>=115) MV = 115;         //reset-windup
  if (MV<=0) MV = 0;
```

```
  }
  if (MV<=0)
  TC = -MV+115;                    //valor da tensão na carga (0 a 115 volts)

  Tx = ((MV*8.1)/115);             //Ajuste para um Tx máximo de 8,10ms

  //valores para Fint=12MHz (Fosc=48MHz - Planta)
  //Ti=100us Tp+Ts=233us
  //150 equivale a 100us
  //12150 equivale a 8.1ms
  Valor_Set = ((Tx*12150)/8.1); //Valor para set_timer1(65536-Valor_Set);
                                //para que o timer1 conte até no máximo 8,10ms
  if (Valor_Set<=150) Valor_Set=150;   //Valor mínimo para que haja um tempo de 100us
                                       //entre a interrupção de passagem por zero e o término dos
                                       //cálculos (conversão AD e controle)
  if (Valor_Set>=12150) Valor_Set=12150;       //Valor máximo para que haja um tempo de
                                               //Ts=53,3us antes do próximo pulso
                                               //Considerando: Ts = (T/2 ms - 8,1000ms) - Ti
                                               //(Ti=tempo do inicio=100us) - Tp  (Tp=tempo de
                                               //pulso=80us)
                                               //Ts = 233,3us - 100us - 80us = 53,3us ( Ts=Tempo
                                               //de sobra depois do término do pulso e antes da
                                               //próxima interrupção)
}

void ler_adcs()
{
  set_adc_channel(0);  //0,45v --> 1023   // usando tesão de referência de 045 volts no pino 5 do PIC
  delay_us(2);
  valoradc0 = read_adc();

  set_adc_channel(1);
  delay_us(2);
  valoradc1 = read_adc();

  set_adc_channel(2);
  delay_us(2);
  valoradc2 = read_adc();
}

#int_TIMER0
void  TIMER0_isr(void)           //interrupção do timer zero para desligar o pulso em c0
{
  output_low(pin_c0);
  disable_interrupts(INT_TIMER0);
}

#int_TIMER1
void  TIMER1_isr(void)
{
  if ((flagpausa1)||(flaginicio)||(op==3)||(op==4)) output_low(pin_c0);  //se em pausa, ou estado inicial,
                                                    //ou desligado (tempo esgotado) ou ligado (tempo
                                                    //esgotado), não emite pulso em c0.
  else output_high(pin_c0);
  disable_interrupts(INT_TIMER1);
  set_timer0(64576);    // habilita Timer 0 para contar 80us,  que é a duração do pulso.
  enable_interrupts(INT_TIMER0);
}
```

```
#int_TIMER2
void  TIMER2_isr(void)
{
  cont = cont+1;                    //aqui é a temporização do banho.
  if (cont==54931) //5 minutos   //183,1065863 corresponde a 1 segundo.
  {
    flagtempo=1;                    //sinalização de tempo esgotado. Exige reinicio.
    cont=0;
  }
}

#int_TIMER3
void  TIMER3_isr(void)            //timer 3 configurado para contar 25 ms de overflow
{
    tempocontrole = tempocontrole+1;  //tempo de 50ms para executar ler_adcs().
    if (tempocontrole==2)
    {
      ler_adcs();
      flagtempocontrole=1;
      tempocontrole = 0;                    //flag para reiniciar contagem
    }

    if (op==1)
    {
      contatempoenvio = contatempoenvio + 1; //envio de dados a cada 0,05 segundo pela porta USB
      if (contatempoenvio==2)
      {
        flagtempoenvio=1;
        contatempoenvio=0;
      }
    }
    else
    {
      flagtempoenvio=0;
      contatempoenvio=0;                    //para zerar valores quando a chave estiver desativada.
      contatempoenviototal=0;
    }

    cont3 = cont3+1;
    if (cont3==120)       //contagem de 3 segundos para alternar o display entre Taguaamb e SP
    {                     //1 ciclo = 25ms
      flagalternadisplay=!flagalternadisplay;
      cont3 = 0;
    }

    if ((flagpausa2)||(flaginicio)||(op==3)||(op==4))   //se em pausa, ou estado inicial, ou desligado
                                                        //(tempo esgotado) ou ligado (tempo esgotado),
                                                              //iniciar contagem de tempo para modo sleep.
    {                                                         //3 seg -- 120 ciclos  (5 min=300seg)
      cont4=cont4+1;              //contagem de 5 min
      if (cont4==12000) //5 minutos. Este tempo é o tempo para pausa (chuveiro desligado)
      {
        disable_interrupts(INT_EXT);
        flagpausa3=1;
        cont4 = 0;
      }
    }
    else cont4=0;
    set_timer3(28036+get_timer3());     //Somando o valor atual do timer3 para compensar
                                        //o tempo de processamento.
```

```
}

#int_EXT
void  EXT_isr(void)
{
  set_timer1(65536-Valor_Set); //configura o timer1 para contar até o valor de Tx
  enable_interrupts(INT_TIMER1);

  flagcontinua=1;        //flag para continuar o programa. (para sair do loop infinito while;)

  piscaled = piscaled+1;
  if (piscaled==60)
  {                                          //loop para piscar led com a passagem por zero da rede
    output_toggle(pin_e2);
    piscaled=0;
  }

  flagedge=!flagedge;
  if (flagedge) ext_int_edge(H_TO_L);  //inversão de borda para interrupção.
  else ext_int_edge(L_TO_H);
}

#int_EXT1
void  EXT1_isr(void)     //Chave de inicio conectada na interrupção externa 1.
                         //Possibilita entrada em modo Sleep quando o chuveiro não está em uso.
{
  flagliga = input(pin_b1);
  if (!flagliga) ext_int_edge(1,L_TO_H);      //Lógica para possibilitar pausa na temporização.
  else ext_int_edge(1,H_TO_L);
}

/************************************************************************
 *                          Saídas                                      *
 ************************************************************************/

#define rs pin_e0          //sinaliza recepção de dados ou instruções do LCD-pino rs
#define enable pin_e1   //definindo com o nome enable o pin_e1(enable do LCD)

/************************************************************************
 *            Rotina que envia um COMANDO para o LCD                    *
 ************************************************************************/

void comando_lcd(int caracter)
{
  output_low(rs);          //rs = 0 seleciona o envio de um comando
  output_d(caracter);    //portd = caracter - carrega o portb com o caracter
  output_high(enable);  //enable = 1 - gera pulso no enable
  delay_us(4);
  output_low(enable);   //enable = 0 - desce o pino de enable
  delay_us(160);
}

/************************************************************************
 *              Rotina que envia um DADO para o LCD                     *
 ************************************************************************/

void escreve_lcd(int caracter)
{
  output_high(rs);         //rs = 1 - seleciona o envio de dado
  output_d(caracter);    //portb = caracter - carrega o portb com o caracter
```

```
  output_high(enable);  //enable = 1 - gera pulso no enable
  delay_us(4);
  output_low(enable);   //enable = 0 - desce o pino de enable

  delay_us(160);
}

/************************************************************************
 *                    Função para limpar o LCD                          *
 ************************************************************************/

void limpa_lcd()
{
  comando_lcd(0x01);  //limpa lcd
  delay_ms(8);
}

/************************************************************************
 *                Inicialização do Display de LCD                       *
 ************************************************************************/

void inicializa_lcd()
{
  comando_lcd(0x30);     // envia comando para inicializar display
  delay_ms(16);
  comando_lcd(0x30);     // envia comando para inicializar display
  delay_us(4000);
  comando_lcd(0x30);     // envia comando para inicializar display
  comando_lcd(0x38);     // liga o display, sem cursor e sem blink
  limpa_lcd();           // limpa lcd
  comando_lcd(0x0c);     // display sem cursor
  comando_lcd(0x06);     // desloca cursor para a direita
}

/************************************************************************
 *                        Tela Principal                                *
 ************************************************************************/

void tela_principal()
{
  if (!input(pin_b3))
  {
    if (flagalternadisplay==1)
    {
      if (!input(pin_b3))
      {
        comando_lcd(0x80);                  // posiciona o cursor na linha 0, coluna 1
        printf (escreve_lcd,"T.Autom.  ");  // imprime mensagem no lcd
      }
      else
      {
        comando_lcd(0x80);                  // posiciona o cursor na linha 0, coluna 1
        printf (escreve_lcd,"T.Manual");    // imprime mensagem no lcd
      }
    }
    else
    {
        comando_lcd(0x80);                  // posiciona o cursor na linha 0, coluna 1
        printf (escreve_lcd,"TAg.Amb.  ");  // imprime mensagem no lcd
    }
```

```
  }
  else
  {
     comando_lcd(0x80);                    // posiciona o cursor na linha 0, coluna 1
     printf (escreve_lcd,"T.Manual");      // imprime mensagem no lcd
  }

  comando_lcd(0x8a);
  printf (escreve_lcd,"T.Agua");
  comando_lcd(0xc4);                 // posiciona o cursor na linha 1, coluna 5
  printf (escreve_lcd,"oC");         // imprime mensagem no lcd
  comando_lcd(0xce);
  printf (escreve_lcd,"oC");
}

void mostrar_temperaturas()  //Função para mostrar valores de Temperatura no LCD
{
  //1023       -->  150oC
  //valoradc() -->  temperatura
  if (!input(pin_b3))
  {
    if (flagalternadisplay==1)          //lógica para alternar mostragem de temperaturas no display
    {
      comando_lcd(0xc0);
      printf(escreve_lcd,"%3.1f",SP);
    }
    else
    {
      comando_lcd(0xc0);
      printf(escreve_lcd,"%3.1f",Taguaamb);
    }
  }
  else
  {
      comando_lcd(0xc0);
      printf(escreve_lcd,"%3.1f",SP);
  }

  comando_lcd(0xca);
  printf(escreve_lcd,"%3.1f",PV);}

/*************************************************************************
 *                Configurações do Microcontrolador                      *
 *************************************************************************/

void main()
{
  setup_adc_ports(AN0_TO_AN2|VSS_VREF);
  setup_adc(ADC_CLOCK_INTERNAL);
  setup_psp(PSP_DISABLED);
  setup_spi(SPI_SS_DISABLED);
  setup_wdt(WDT_OFF);
  setup_timer_0(RTCC_INTERNAL);
  setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);
  setup_timer_2(T2_DIV_BY_16,255,16);
  setup_timer_3(T3_INTERNAL|T3_DIV_BY_8);
  set_timer3(28036);   //para contar 25 ms , em Fint=12Mhz
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  ext_int_edge(L_TO_H);
```

```
ext_int_edge(1,H_TO_L);
enable_interrupts(INT_EXT1);

enable_interrupts(GLOBAL);

set_tris_a (0b11001111);
set_tris_b (0b01001011);
set_tris_c (0b10000000);   //c0=pulso(saída), c6=TX (saída), c7=Rx (entrada)
set_tris_d (0b00000000);
set_tris_e (0b00001000);

IDLEN = 0;               //em 0, modo SLEEP; em 1, modo IDLE
output_high(pin_b2);  //pino 35 (base do transistor que liga/desliga o Vcc do display)
delay_ms(1000);
inicializa_lcd();

output_low(pin_b5);
output_low(pin_b6);  // zerando as saídas (leds)
output_low(pin_b7);
output_low(pin_c0);  // mantendo porta de pulso (c0) em baixo.
output_low(pin_e2);

flagliga = input(pin_b1);
if (flagliga) ext_int_edge(1,H_TO_L);   //para gerar interrupção quando o botão liga for pressionado.
else ext_int_edge(1,L_TO_H);

ler_adcs();                        //Para ler a temperatura ambiente pela primeira vez.
Taguaamb = ((100*((valoradc1*0.45)/1023))-0.8);
SP = ((100*((valoradc2*1.5)/1023))-0.75);
usb_init_cs();       //inicia usb sem ficar esperando em loop infinito até conectar à bus.
usb_cdc_init();     // Inicia protocolo cdc

flagenvio=1;       //para iniciar envio quando a chave liga for ativada.

enable_interrupts(INT_EXT);
enable_interrupts(INT_TIMER3);

while (true)
{
  usb_task();    //para sempre verificar a conexão, quando se usa o usb_int_cs()
  if ((!flagliga)&&(!flagtempo)) op=1;
  if ((flagliga)&&(!flagtempo))  op=2;
  if ((flagliga)&&(flagtempo))   op=3;
  if ((!flagliga)&&(flagtempo))  op=4;

  switch (op)
  {
    case 1:  //ligado e contando tempo
    {

        if (flagtempocontrole==1)      //loop para tempo de amostragem de aprox. 50 ms
        {
          controle();
          flagtempocontrole=0;   //flag para possibilitar loop if a cada 50ms aprox.
        }
        flaginicio=0;      //não está no estado inicial (desligado)
        flagpausa1=0;  //em 0 permite emissão de pulso em c0.
        flaglimpa1=0;
        flaglimpa3=0;
        if (!flaglimpa2) limpa_lcd(); //limpar LCD apenas uma vez enquanto estiver no caso 1
```

```
            flaglimpa2=1;
            tela_principal();
            vtempo1 = SP-(SP*0.05);         // variável para verificação se a temperatura ja atingiu o
                                            //desejável e iniciar a temporização
            vtempo2 = SP+(SP*0.05);
            flagtimer = input(pin_b6);      // verificação do botão de liga/desliga temporização
            if ( (flagtimer) && ((PV>vtempo1)&&(PV<vtempo2)) && ((!flagtempo1)||(flagtempo2)) )
            {
              enable_interrupts(INT_TIMER2);
              comando_lcd(0xc7);
              printf(escreve_lcd,"T.");             // condição em que a temporização está ligada
              flagtempo1=1;
            }
            else if (!flagtimer)
            {
              disable_interrupts(INT_TIMER2);
              comando_lcd(0xc7);                    // condição em que a temporização está desligada
              printf(escreve_lcd,"  ");
              flagtempo2=1;
            }

            output_high(pin_b7);  //led verde aceso
            output_low(pin_b4);   //led amarelo apagado
            output_low(pin_b5);   //led vermelho apagado

            mostrar_temperaturas();

            if (usb_cdc_kbhit(1))   //verifica se chegou um novo dado no buffer USB.
            {
              rec[0]=usb_cdc_getc();
              nslave = rec[0];
              if (nslave==1) //verifica se o endereço do slave é igual a 1
              {
                rec[1]=usb_cdc_getc(); //recebe o segundo dado do buffer USB.
                nfuncao = rec[1];
                if (nfuncao==3)
                {
                  flagenvio=1;

                }
                if (nfuncao==2)
                {
                  flagenvio=0;
                  contatempoenviototal=0;
                }
              }
            }
            if ((flagenvio)&&(flagtempoenvio))
            {
              contatempoenviototal = contatempoenviototal+0.05;
              printf(usb_cdc_putc_fast,"%f%c%f%c%f%c%f%c%f%c",contatempoenviototal,'
',Taguaamb,' ',SP,' ',PV,' ',TC,'@');                   //envia os dados para a USB
              flagtempoenvio=0;
            }

            flagpausa=1;       //flag para que seja possível dar pausa no chuveiro.
            flagpausa2=0;      //flag indicativo de que está em pausa (se 0, não está)
          }
          break;
          case 2:  //desligado ou em pausa.
```

```
{
  if (!flagpausa)   //situação em que o chuveiro está desligado. Situação inicial
  {
    flaginicio=1; //indica que está no estado inicial
    flagpausa2=0; //indica que não está em pausa
    disable_interrupts(INT_TIMER2);  //desabilitar contagem de tempo (temporização)
    flaglimpa2=0;
    flaglimpa3=0;
    if (!flaglimpa1) limpa_lcd();
    flaglimpa1=1;

    if (!input(pin_b3))  //se em modo automático ...
    {
      if (flagalternadisplay==1)
      {
        comando_lcd(0x80);
        printf(escreve_lcd,"Modo     LIGUE A");
        comando_lcd(0xC0);
        printf(escreve_lcd,"Autom.    AGUA! ");
      }
      else
      {
        comando_lcd(0x80);
        printf(escreve_lcd," Temp.Agua.Amb. ");    //Considerando que o sensor está medido a
                                                   //temperatura da água que está na
                                                   //temperatura ambiente
        ler_adcs();
        Taguaamb = ((100*((valoradc1*0.45)/1023))-0.8);
        comando_lcd(0xc0);
        printf(escreve_lcd,"     %3.1foC       ",Taguaamb);
      }
    }


    else  //... se em modo manual ...
    {
      if (flagalternadisplay==1) //lógica para alternar mostragem de temperaturas no display.
      {
        comando_lcd(0x80);
        printf(escreve_lcd,"Modo     LIGUE A");
        comando_lcd(0xC0);
        printf(escreve_lcd,"Manual    AGUA! ");

      }
      else
      {
        comando_lcd(0x80);
        printf(escreve_lcd," Ajuste T.Manual");
        ler_adcs();
        Taguaamb = ((100*((valoradc1*0.45)/1023))-0.8);
        SP = ((100*((valoradc2*0.45)/1023))-0.75);

        if (SP<=Taguaamb) SP=Taguaamb;  //para limitar o valor do set-point manual
        if (SP>=(Taguaamb+deltaT)) SP=(Taguaamb+deltaT);  //para limitar o valor do set-point
                                                          //manual no valor máximo para controle
        comando_lcd(0xc0);
        printf(escreve_lcd,"     %3.1foC       ",SP);
      }
    }
```

```
   output_low(pin_c0);  //desligando pulso
   output_low(pin_b7);  //led verde apagado
   output_low(pin_b4);  //led amarelo apagado
   output_high(pin_b5); //led vermelho aceso

   if (flagpausa3)          //caso em situação inicial prolongada
   {
     output_low(pin_e2);   //desligando led da Int0

     output_low(pin_c0);   //desligando pulso
     output_low(pin_b7);   //led verde apagado
     output_low(pin_b4);   //led amarelo apagado
     output_low(pin_b5);   //led vermelho apagado

     usb_detach();

     disable_interrupts(INT_TIMER2);
     disable_interrupts(INT_TIMER1); //desabilitando todas as interrupções, exceto int_ext1
     disable_interrupts(INT_TIMER0);

     output_low(pin_b2);  //pino 35 (base do transistor que liga/desliga o Vcc do display)
     output_d(0x00);       //necessário para desligar completamente o display
     output_e(0x00);
     sleep();
     output_high(pin_b2);  //pino 35 (base do transistor que liga/desliga o Vcc do display)
     delay_ms(1000);
     flagcontinua=0;
     enable_interrupts(INT_EXT);
     while (!flagcontinua);  //continuar programa somente depois de um zero na rede.
     inicializa_lcd();
     flagpausa3=0;
   }
}
else      //flagpausa em 1 (chave ligada). Chave de inicio pressionada enquanto chuveiro
          //ligado, aciona pausa.
{
   flaginicio=0;  //indica que não está no estado inicial. Em pausa.
   flagpausa2=1;  //indica que está em pausa.
   disable_interrupts(INT_TIMER2);  //desabilitar contagem de tempo (temporização)
   flagpausa1=1;  //em 1 inibe emissão de pulso em c0.
   flagtempo1=0;  //zerando para possibilitar reinicio da contagem de tempo quando retornar ao
                  //caso 1 (ligado em contando tempo)
   flaglimpa2=0;
   flaglimpa3=0;
   if (!flaglimpa1) limpa_lcd();
   flaglimpa1=1;
   comando_lcd(0x85);
   printf(escreve_lcd,"Pausa!");
   comando_lcd(0xc2);
   printf(escreve_lcd,"Religue agua.");

   output_low(pin_c0);   //desligando pulso
   output_low(pin_b7);   //led verde apagado
   output_high(pin_b4);  //led amarelo aceso
   output_low(pin_b5);   //led vermelho apagado

   if (flagpausa3)      //caso de pausa prolongada
   {
     flagtempo=1;
```

```
        output_low(pin_e2);   //desligando led da Int0

        output_low(pin_c0);   //desligando pulso
        output_low(pin_b7);   //led verde apagado
        output_low(pin_b4);   //led amarelo apagado
        output_low(pin_b5);   //led vermelho apagado

        usb_detach();

        disable_interrupts(INT_TIMER2);
        disable_interrupts(INT_TIMER1);    //desabilitando todas as interrupções, exceto int_ext1
        disable_interrupts(INT_TIMER0);

        output_low(pin_b2);  //pino 35 (base do transistor que liga/desliga o Vcc do display)
        output_d(0x00);       //necessário para desligar completamente o display
        output_e(0x00);
        sleep();
        output_high(pin_b2);  //pino 35 (base do transistor que liga/desliga o Vcc do display)
        delay_ms(1000);
        flagcontinua=0;
        enable_interrupts(INT_EXT);
        while (!flagcontinua);  //continuar programa somente depois de zero na rede
        inicializa_lcd();
        flagpausa3=0;
        flaglimpa1=0;
      }
    }
  }
  break;
  case 3:  //desligado e tempo esgotado
  {
    disable_interrupts(INT_TIMER2);   //desabilitar contagem de tempo (temporização)
    flaglimpa2=0;
    flaglimpa3=0;
    if (!flaglimpa1) limpa_lcd();
    flaglimpa1=1;

    comando_lcd(0x81);
    printf(escreve_lcd,"Tempo esgotado");
    comando_lcd(0xc4);
    printf(escreve_lcd,"Reinicie");

    flagpausa=0;
    flagpausa2=0;      //flag indicativo de que está em pausa (se 0, não está)

    output_low(pin_c0);  //desligando pulso (redundância)
    output_low(pin_b7);  //led verde apagado
    output_low(pin_b4);  //led amarelo apagado
    output_high(pin_b5); //led vermelho aceso

    if (flagpausa3)      //caso de tempo esgotado
    {
      flagtempo=1;

      output_low(pin_e2);   //desligando led da Int0

      output_low(pin_c0);   //desligando pulso
      output_low(pin_b7);   //led verde apagado
      output_low(pin_b4);   //led amarelo apagado
      output_low(pin_b5);   //led vermelho apagado
```

```
      usb_detach();

      disable_interrupts(INT_TIMER2);
      disable_interrupts(INT_TIMER1);       //desabilitando todas as interrupções, exceto int_ext1
      disable_interrupts(INT_TIMER0);

      output_low(pin_b2);  //pino 35 (base do transistor que liga/desliga o Vcc do display)
      output_d(0x00);       //necessário para desligar completamente o display
      output_e(0x00);
      sleep();
      output_high(pin_b2);  //pino 35 (base do transistor que liga/desliga o Vcc do display)
      delay_ms(1000);
      flagcontinua=0;
      enable_interrupts(INT_EXT);
      while (!flagcontinua);  //continuar programa somente depois de detectado zero na rede.
      inicializa_lcd();
      flagpausa3=0;
    }
}
  break;
  case 4:  //ligado e tempo esgotado
  {
    disable_interrupts(INT_TIMER2); //desabilitar contagem de tempo (temporização)
    flaglimpa2=0;
    flaglimpa3=0;
    if (!flaglimpa1) limpa_lcd();
    flaglimpa1=1;

    comando_lcd(0x81);
    printf(escreve_lcd,"Tempo esgotado");
    comando_lcd(0xc4);
    printf(escreve_lcd,"Reinicie");

    flagpausa=0;
    flagpausa2=0;      //flag indicativo de que não está em pausa (desligado).

    output_low(pin_c0);  //desligando pulso (redundância)
    output_low(pin_b7);  //led verde apagado
    output_low(pin_b4);  //led amarelo apagado
    output_high(pin_b5); //led vermelho aceso

    if (flagpausa3)      //caso de tempo esgotado
    {
      flagtempo=1;

      output_low(pin_e2);   //desligando led da Int0

      output_low(pin_c0);   //desligando pulso
      output_low(pin_b7);   //led verde apagado
      output_low(pin_b4);   //led amarelo apagado
      output_low(pin_b5);   //led vermelho apagado

      usb_detach();

      disable_interrupts(INT_TIMER2);
      disable_interrupts(INT_TIMER1);       //desabilitando todas as interrupções, exceto int_ext1
      disable_interrupts(INT_TIMER0);

      output_low(pin_b2);  //pino 35 (base do transistor que liga/desliga o Vcc do display)
```

```
            output_d(0x00);      //necessário para desligar completamente o display
            output_e(0x00);      //
            sleep();
            output_high(pin_b2);  //pino 35 (base do transistor que liga/desliga o Vcc do display)
            delay_ms(1000);
            flagcontinua=0;
            enable_interrupts(INT_EXT);
            while (!flagcontinua);  //continuar programa somente depois de detectado zero na rede
            inicializa_lcd();
            flagpausa3=0;
          }
        }
        break;
      }
    }
}
```

Arquivo *.h

```
#include <18F4550.h>
#device adc=10

//Código para remapear o inicio do programa e os vetores de interrupção.
//Necessário para gravar através do BootLoader
#build (reset=0x800, interrupt=0x808)
#org 0x000, 0x7ff {}

#FUSES NOWDT                 //No Watch Dog Timer
#FUSES WDT128                //Watch Dog Timer uses 1:128 Postscale
#FUSES HSPLL                 //High Speed Crystal/Resonator with PLL enabled
#FUSES NOPROTECT             //Code not protected from reading
#FUSES NOBROWNOUT            //No brownout reset
#FUSES BORV20                //Brownout reset at 2.0V
#FUSES NOPUT                 //No Power Up Timer
#FUSES NOCPD                 //No EE protection
#FUSES STVREN                //Stack full/underflow will cause reset
#FUSES NODEBUG               //No Debug mode for ICD
#FUSES NOLVP                 //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOWRT                 //Program memory not write protected
#FUSES NOWRTD                //Data EEPROM not write protected
#FUSES IESO                  //Internal External Switch Over mode enabled
#FUSES FCMEN                 //Fail-safe clock monitor enabled
#FUSES PBADEN                //PORTB pins are configured as analog input channels on RESET
#FUSES NOWRTC                //configuration not registers write protected
#FUSES NOWRTB                //Boot block not write protected
#FUSES NOEBTR                //Memory not protected from table reads
#FUSES NOEBTRB               //Boot block not protected from table reads
#FUSES NOCPB                 //No Boot Block code protection
#FUSES MCLR                  //Master Clear pin enabled
#FUSES LPT1OSC               //Timer1 configured for low-power operation
#FUSES NOXINST               //Extended set extension and Indexed Addressing mode disabled
#FUSES PLL5                  //Divide By 5(20MHz oscillator input)
#FUSES CPUDIV1               //No System Clock Postscaler
#FUSES USBDIV                //USB clock source comes from PLL divide by 2
#FUSES VREGEN                //USB voltage regulator enabled
#FUSES ICPRT                 //ICPRT enabled

#use delay(clock=20000000)
#use fast_io(ALL)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
```

APÊNDICE C - CÓDIGO FONTE DO PROGRAMA EM C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace SimpleSerial
{
    public partial class Form1 : Form
    {
        // Declaração de Variáveis
        string RX;  //variável que armazena o dado recebido pela USB (string)
        byte[] tra = new byte[2]; //variável para envio de dados pela USB
        string[] rec = new string[4];

        public Form1()
        {
            InitializeComponent();
            buttonStart.Enabled = true;
            buttonStop.Enabled = false;     //inicialização de botões
            textBox1.ReadOnly = true;
        }
        //função para conectar à porta com estabelecida pelo usuário
        private void buttonStart_Click(object sender, EventArgs e)
        {
        // Altere para sua COM. Veja no gerenciador de dipositivos qual a disponível
            bool error = false;
            serialPort1.PortName = PortName.Text;
            serialPort1.BaudRate = Convert.ToInt16(BaudRate.Text);

            if (!serialPort1.IsOpen)
            {
                try
                {
                    // Abrir a porta
                    serialPort1.Open(); //tenta conectar ...
                }
                catch (UnauthorizedAccessException) { error = true; }
                catch (Exception) { error = true; }
                // ... se não for possível, mostra msg de erro.
                if (error) MessageBox.Show(this, "Não foi possível abrir a porta COM.
Está em uso, foi removida ou não está disponível.", "COM Port não disponível",
MessageBoxButtons.OK, MessageBoxIcon.Stop);

                if (serialPort1.IsOpen)
                {
                    buttonStart.Enabled = false;
                    buttonStop.Enabled = true; //tratamento de botões caso conectado.
                    textBox1.ReadOnly = false;
                }
            }
        }
        //função para desfazer conexão
        private void buttonStop_Click(object sender, EventArgs e)
        {
            bool error = false;
```

```
            if (serialPort1.IsOpen)
            {
                try
                {
                    // fechar a porta
                    serialPort1.Close();  //tenta desfazer conexão ...
                }
                catch (UnauthorizedAccessException) { error = true; }
                catch (Exception) { error = true; }
                //... caso não seja possível, mostra msg de erro.
                if (error) MessageBox.Show(this, "Não foi possível fechar a porta COM.
Já foi removida ou não está disponível.", "COM Port não disponível",
MessageBoxButtons.OK, MessageBoxIcon.Stop);

                buttonStart.Enabled = true;
                buttonStop.Enabled = false;
                textBox1.ReadOnly = true;
            }
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            bool error = false;
            if (serialPort1.IsOpen)
            {
                try
                {
                    // Fechar a porta
                    serialPort1.Close();  //tenta desfazer conexão ...
                }
                catch (UnauthorizedAccessException) { error = true; }
                catch (Exception) { error = true; }
                //... caso não seja possível, mostra msg de erro.
                if (error) MessageBox.Show(this, "Não foi possível fechar a porta COM.
Já foi removida ou não está disponível.", "COM Port não disponível",
MessageBoxButtons.OK, MessageBoxIcon.Stop);
            }

        }
        //função para logica de acendimento dos LED's no Aplicativo
        private void DisplayText(object sender, EventArgs e)
        {
            textBox1.AppendText(RX); // = RX; // (Convert.ToString(rec[0]));
        }
        //função de recebimento do dado
        private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            RX = serialPort1.ReadExisting();
            this.Invoke(new EventHandler(DisplayText));//invoca a função DisplayText()
        }

        private void button1_Click_1(object sender, EventArgs e)
        {
            tra[0] = 1;
            tra[1] = 3;

            bool error = false;
            if (serialPort1.IsOpen)
            {
                try
                {
```

```
                    // Fechar a porta
                    serialPort1.Write(tra, 0, 2);  //tenta enviar 13 para a porta USB
                }
                catch (UnauthorizedAccessException) { error = true; }
                catch (Exception) { error = true; }
                //... caso não seja possível, mostra msg de erro.
                if (error) MessageBox.Show(this, "Não foi possível enviar os dados à
porta COM.  Já foi removida ou não está disponível.", "COM Port não disponível",
MessageBoxButtons.OK, MessageBoxIcon.Stop);
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            tra[0] = 1;
            tra[1] = 2;

            bool error = false;
            if (serialPort1.IsOpen)
            {
                try
                {
                    // Fechar a porta
                    serialPort1.Write(tra, 0, 2);  //tenta enviar 13 para a porta USB
                }
                catch (UnauthorizedAccessException) { error = true; }
                catch (Exception) { error = true; }
                //... caso não seja possível, mostra msg de erro.
                if (error) MessageBox.Show(this, "Não foi possível enviar os dados à
porta COM.  Já foi removida ou não está disponível.", "COM Port não disponível",
MessageBoxButtons.OK, MessageBoxIcon.Stop);
            }
        }
    }
}
```