UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA - CCET
DEPARTAMENTO DE ENGENHARIA ELÉTRICA - DEE

Eduardo Santucci Roston

Sistema de Controle Automatizado de Baixo Custo para Irrigação

SÃO CARLOS -SP
2021

Eduardo Santucci Roston

Sistema de Controle Automatizado de Baixo Custo para Irrigação

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia Elétrica da Universidade Federal de São Carlos, para obtenção do título de bacharel em Engenharia Elétrica.

Orientador: Prof.Dr. Celso Ap.de França

São Carlos-SP
2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia - CCET
Programa de Graduação em Engenharia Elétrica

Comissão avaliadora

Membros da comissão examinadora que avaliou e aprovou a Defesa do Trabalho de Conclusão de Curso do candidato Eduardo Santucci Roston, realizada em 02/12/2021:

Prof. Dr. Celso Aparecido de França
Instituição: Universidade Federal de São Carlos

Prof. Dr. Helder Vinícius Avanço Galeti
Instituição: Universidade Federal de São Carlos

Prof. Dr. Osmar Ogashawara
Instituição: Universidade Federal de São Carlos

DEDICATÓRIA

Dedico o presente trabalho ao meu pai, que sempre me deu excelentes conselhos e ensinamentos com sua vasta experiencia na área acadêmica e a minha mãe, que sempre me deu suporte incondicional durante toda essa árdua jornada de desenvolvimento acadêmico e profissional.

Ao Prof. Dr. Celso Aparecido de França pela mentoria, oportunidade de desenvolvimento do tema e principalmente a dedicação e disponibilidade de ajudar, tornando o presente trabalho possível.

RESUMO

Sistemas de irrigação automatizados, sejam eles para fins comerciais ou residenciais, são amplamente implementados com o objetivo de automatizar uma tarefa recorrente, no caso rega de plantas e plantações, e para o uso consciente dos recursos hídricos. Este tipo de sistema é muito predominante em países mais desenvolvidos, como os EUA, porém o preço elevado para sua instalação acaba se tornando uma barreira financeira para muitos, especialmente em países mais pobres como o Brasil. O presente trabalho visa o desenvolvimento de um dos principais componentes utilizados neste tipo de sistema, o controlador de irrigação, responsável por coordenar as rotinas de irrigação assim como comutar o fluxo de água para diferentes áreas de irrigação, porém com o diferencial de ser um controlador Open-Source e Open-Hardware de baixo custo. Este controlador tem o objetivo de ser uma alternativa de maior custo benefício, se comparado a concorrentes comerciais, visando o acesso de um maior público a este tipo de tecnologia. O controlador em questão é um sistema embarcado baseado na plataforma Open-Source Arduino, utilizando o microcontrolador ATmega328p. O mesmo é composto por duas placas de circuito impresso (PCI) operando de forma conjunta, sendo elas o “Módulo de Controle”, responsável pelo controle do sistema e automatização das rotinas de irrigação, e o “Módulo de Acionamento”, responsável por comutar as válvulas solenoide de modo a permitir o fluxo de água. Ao final do desenvolvimento deste trabalho verificou-se que o objetivo principal foi atingido, o controlador desenvolvido apresenta um custo total significativamente menor do que de similares comerciais sem sacrificar as principais funções oferecidas por soluções de preços mais elevados.

Palavras-chave: Sistema embarcado. Sistema de irrigação. Irrigação automatizada. Eficiência hídrica.

ABSTRACT

Low Cost Automated Irrigation Control System

AUTHOR: Eduardo Santucci Roston
ADVISOR: Prof. Dr. Celso Aparecido de França

Automated irrigation systems, whether for commercial or residential purposes, are widely implemented with the aim of automating a recurrent task, in this case watering plants and crops, and for the smart use of water resources. This type of system is very prevalent in more developed countries, such as the USA, but the high price for its installation ends up becoming a financial barrier for many, especially in poorer countries like Brazil. The present work aims to develop one of the main components used in this type of system, the irrigation controller, responsible for coordinating the irrigation routines as well as switching the water flow to different irrigation areas, but with the difference of being an Open-Source and Open-Hardware low-cost controller. This controller has the objective of being a more cost-effective alternative, if compared to commercial competitors, aiming at the access of a larger public to this type of technology. The controller in question is an embedded system based on the Open-Source Arduino platform, using the ATmega328p microcontroller. It is composed of two printed circuit boards (PCB) operating together, the "Control Module", responsible for system control and automation of irrigation routines, and the "Driver Module", responsible for switching solenoid valves to allow water flow. At the end of the development of this work, it was verified that the main objective was reached, the developed controller presents a total cost significantly lower than that of commercial similar ones without sacrificing the main functions offered by higher price solutions.

Keyword: Embedded system. Irrigation system. Automated irrigation. Water efficiency.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

CI – Circuito Integrado
DIY – Do-It-Yourself (Faça-Você-Mesmo)
EUA – Estados Unidos da América
IDE – Integrated Development Environment
LCD – Liquid Crystal Display (Display de Cristal Líquido)
LDR – Light Dependent Resistor (Resistor Variável Conforme Incidência De Luz)
LED – Light Emitting Diode (Diodo Emissor de Luz)
PCB – Printed Circuit Board
PCI – Placa de Circuito Impresso
PWM – Pulse Width Modulation
RTC – Real Time Clock
SCL – Serial Clock (Clock Serial)
SDA – Serial Data (Dados Seriais)
TBJ – Transistor Bipolar de Junção
TTL – Transistor Transistor Logic

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A irrigação, seja ela para fins de plantio de subsistência, como plantações de insumos, ou para fins decorativos, como jardins, está presente na rotina do ser humano desde o começo da civilização agrícola. Com o passar do tempo, novas técnicas de irrigação foram desenvolvidas visando aprimorar técnicas já existentes de modo a aumentar a produtividade de plantações e para facilitar o ato da irrigação em si.

De acordo com Mantovani et al (2013), existem diversos tipos de irrigação como: Irrigação por Superfície, Aspersão, Micro aspersão, Gotejamento, Subsuperficial, Sulco, entre outros. O presente trabalho irá focar na irrigação por aspersão, no qual diversos aspersores, geram uma cortina de gotículas de água de modo a abranger diversas áreas.

Graças ao advento de novas tecnologias, em especial o transistor, o ato de regar plantações e jardins se tornou algo automatizado eletronicamente nos tempos modernos, não necessitando mais da atenção constante do ser humano para garantir a rega efetiva e também garantindo o uso consciente dos recursos hídricos.

Atualmente existem diversas soluções disponíveis no mercado para sistemas de irrigação automatizados. Tais sistemas podem ser voltados para irrigação de grandes áreas de plantio, como em fazendas e gramados de estádios, ou em áreas menores, como jardins residenciais e comerciais, assim como pequenas hortas.

Um problema enfrentado por pessoas que buscam automatizar a rega de sua propriedade residencial é o alto valor inicial exigido para a implementação do sistema, o que pode representar uma barreira financeira para muitos.

Para uma pequena área de irrigação, como um jardim, o valor pode chegar a 1000 reais apenas para a aquisição de equipamentos como mangueiras, aspersores e conectores, sem contar com o valor do controlador em si, que pode variar de 500 a 800 reais (em 2021) para um simples de 4 estações (número de válvulas solenoides que podem ser conectadas). O controlador programável é um item essencial e necessário, pois é responsável por coordenar as rotinas de irrigação e comutar válvulas hídricas eletrônicas de forma automática.

Com auxílio das áreas da eletrônica, sistemas digitais e sistemas embarcados,

o presente trabalho visa o desenvolvimento de um sistema embarcado, baseado na plataforma Open-Source Arduino, fazendo uso do microcontrolador ATmega328p da fabricante de semicondutores Atmel.

1.2 OBJETIVO GERAL

O objetivo geral deste trabalho foi desenvolver um sistema embarcado de controle para a automatização do ato de irrigação de múltiplas zonas (cada zona de irrigação pode ser compreendida como uma área abrangida pelos aspersores conectados à mesma válvula solenoide, no qual o fluxo de água pode ser controlado de forma independente das demais) e que seja eficiente, simples de usar pelo usuário e que apresente as principais funções dos controladores de irrigação comerciais, porém com o diferencial de ser uma solução DIY (Faça-Você-Mesmo), Open-Source e Open Hardware a um custo muito mais acessível do que os produtos comerciais já disponíveis no mercado. Em suma, desenvolver um controlador programável para irrigação a um preço competitivo, porém sem sacrificar as principais funções disponíveis nos controladores comerciais de preços mais elevados.

1.3 OBJETIVOS ESPECÍFICOS

Desenvolver e confeccionar duas placas de circuito impresso (PCI) para serem utilizadas de forma conjunta. A primeira PCI é o controlador propriamente dito, referenciado como Módulo de Controle, contendo o microcontrolador, tela, botões, LEDs e demais componentes eletrônicos para seu correto funcionamento. A segunda PCI, referenciada como Módulo de Acionamento, é responsável por comutar até 5 relés de forma independente e que poderá ser conectada em cascata a demais Módulos de Acionamento para aumentar o número de relés, e consequentemente o número de válvulas solenoide, algo que é conveniente para expandir o número de zonas de irrigação sem a necessidade de compra de outro Módulo de Controle.

1.4 JUSTIFICATIVA

O tema estudado se encontra em um período em que o Brasil sofre sazonalmente com secas e escassez hídrica, o que acaba por tornar o uso consciente dos recursos hídricos cada vez mais necessário.

Sistemas de irrigação automatizados são boas soluções para minimizar o desperdício hídrico, já que apenas as zonas de irrigação especificas são regadas, desta maneira não desperdiçando este recurso em áreas desnecessárias. O tempo pré-determinado de rega para cada zona também auxilia a evitar o desperdício.

A irrigação automatizada, além de eliminar o tempo necessário para realizar esta atividade manualmente, auxilia no uso consciente da água, porém o custo elevado de controladores comerciais inibe e desestimula seu uso para a maioria das pessoas, portanto uma alternativa de maior custo benefício se faz necessária para a disseminação desta tecnologia.

1.5 ESTRUTURA DO TRABALHO

O trabalho está dividido em cinco capítulos. Este capítulo inicial contextualiza e justifica o trabalho. No segundo capítulo é apresentada a fundamentação teórica sobre o assunto obtida através da revisão bibliográfica. O terceiro capítulo mostra o desenvolvimento e a implementação do trabalho. No capítulo quatro são relatados e comentados os resultados obtidos. Por fim, as conclusões finais são apresentadas no capítulo cinco.

2 FUNDAMENTAÇÃO TEÓRICA E REVISÃO DE LITERATURA

Durante a primeira etapa do trabalho final de graduação foi desenvolvido um estudo de revisão bibliográfica dos conceitos que posteriormente foram trabalhados no projeto, tais como: agricultura irrigada, irrigação por aspersão, sistemas embarcados e sistemas automáticos de irrigação.

Também é apresentado os componentes e tecnologias que foram utilizados durante o desenvolvimento do trabalho assim como uma breve introdução teórica sobre cada um deles. No capítulo 3, é apresentado como todos esses componentes interagem entre si para formar o controlador do sistema de irrigação automático.

O embasamento teórico foi essencial para introduzir conhecimentos e competências necessárias para o desenvolvimento do trabalho.

2.1 AGRICULTURA IRRIGADA

A agricultura possui um papel fundamental no desenvolvimento de qualquer civilização, logo diferentes métodos de irrigação foram sendo desenvolvidos e implementados ao longo dos anos como forma de otimizar a eficiência da produção, garantir o plantio mesmo nos períodos de seca graças ao uso consciente dos recursos hídricos e possibilitar a rega de áreas de cultivo maiores.

De acordo com Mantovani et al (2013) a agricultura irrigada se tornou uma excelente estratégia para otimizar a produção mundial de alimentos, assim como proporcionar o desenvolvimento sustentável no campo, garantindo o uso consciente dos recursos hídricos. A agricultura irrigada se torna cada vez mais necessária, já que atualmente mais da metade da população mundial depende de alimentos produzidos em áreas irrigadas.

> No passado, a utilização da irrigação era uma opção técnica de aplicação de água que visava principalmente à luta contra a seca. Atualmente, a irrigação, no foco do agronegócio, insere-se em um conceito mais amplo de agricultura irrigada, sendo uma estratégia para aumento da produção, produtividade e rentabilidade da propriedade agrícola de forma sustentável, preservando o meio ambiente e criando condições para manutenção do homem no campo,

> através da geração de empregos permanentes e estáveis.(Mantovani et al, 2013, p. 13).

2.2 IRRIGAÇÃO POR ASPERSÃO

Atualmente, existem diversos métodos de irrigação, tais como: Irrigação por Superfície, Irrigação Localizada, Aspersão, Micro aspersão, Gotejamento, Subsuperficial, Sulco, entre outros. O presente trabalho irá focar na irrigação por aspersão, no qual diversos aspersores, geram uma cortina de gotículas de água de modo a abranger diversas áreas.

De acordo com Mantovani et al (2013) o método de irrigação por aspersão, além de ser o mais utilizado na atualidade, é caracterizado pelo fato de que quando a água é aspergida pelos aspersores é gerado uma cortina de gotículas de água de modo a simular uma chuva fina que cai sob as plantas.

> O sistema de aspersão convencional é o sistema básico de irrigação por aspersão, do qual derivam todos os demais, e caracteriza-se pelo uso de tubulações móveis de engate rápido ou fixo e enterrado, irrigando normalmente áreas pequenas ou médias. (Mantovani et al, 2013, p. 111).

De acordo com Mantovani et al (2013) os sistemas de irrigação por aspersão convencionais possuem flexibilidade para sofrerem adaptações em seu sistema visando a economia no uso da mão de obra, melhoria na eficiência de irrigação e adequação às distintas situações de campo. Os sistemas mais comuns possuem algumas classificações, tais como: Portátil, Semiportátil, Fixo, Malha, Canhão hidráulico e Mangueira.

Neste trabalho será focado o sistema de irrigação por aspersão do tipo Malha, que é caracterizado por ser um sistema fixo, no qual as tubulações que levam a água até os aspersores são enterradas, além de apresentar um aspersor por malha.

2.3 COMPONENTES DE UM SISTEMA DE IRRIGAÇÃO POR ASPERSÃO

De acordo com Mantovani et al (2013) os principais componentes utilizados para o método de irrigação por aspersão convencional são: sistema de bombeamento, tubulações, acessórios e aspersores. Os acessórios são os mesmos utilizados em outros sistemas, alguns exemplos são: curvas, reduções, tampão final, entre outros.

Neste trabalho, não foi utilizado um sistema de bombeamento para aumentar a pressão da água que chega aos aspersores, já que o sistema de irrigação abordado é voltado para uso em áreas pequenas e residenciais como jardins, hortas e pequenas plantações. Para este objetivo, a pressão de água proveniente da torneira é suficiente para cumprir o objetivo. A utilização de um sistema de bombeamento se faz necessária em situações em que a pressão da água original não é suficiente para o correto funcionamento dos aspersores, já que cada tipo de aspersor possui uma pressão hídrica mínima para operar de forma satisfatória dentro dos requisitos do fabricante.

O sistema de irrigação por aspersão do tipo malha, por ser um sistema fixo, possui as suas tubulações, tubos PVC ou mangueiras, enterradas no solo, com apenas o aspersor para fora que se movimenta automaticamente com a pressão da água, de modo a abranger uma área de cobertura maior.

> O sistema tipo malha é todo fixo, com as tubulações enterradas, muito utilizado em outros países e recentemente redescoberto no Brasil. Nesse sistema funciona, por sua vez, somente um aspersor por linha lateral (malha) em vez de vários, como é feito no sistema convencional fixo, permitindo, em função disso, menores diâmetros nas linhas laterais (1/2”, ¾” e 1”), o que minimiza a utilização de mão de obra. Nesse caso, o que se movimenta não é a linha lateral e sim o aspersor. (Mantovani et al, 2013, p. 115).

De acordo com Mantovani et al (2013) é importante considerar que o sistema de irrigação por aspersão do tipo malha vem sendo utilizado cada vez mais não somente em áreas pequenas, mas também em áreas médias e grandes devido a sua adaptabilidade em diferentes situações de campo, custo competitivo e facilidades operacionais.

2.4 SISTEMAS DE IRRIGAÇÃO AUTOMATIZADA

Novas tecnologias possibilitaram o desenvolvimento de sistemas de irrigação completamente automatizados, no qual o controlador de irrigação, após configurado com as rotinas de irrigação, executa a rega de forma automática e rotineira.

Em um primeiro momento esses controladores de irrigação eram direcionados apenas aos agricultores e comércios devido ao preço elevado deste tipo de sistema, porém com o advento de novas tecnologias de desenvolvimento de hardware e software de código aberto, como a plataforma Arduino, diversas soluções DIY (Do It Yourself – Faça você mesmo) começaram a emergir, tornando possível o acesso de um maior público a este tipo de sistema, especialmente aqueles que buscam automatizar a irrigação de suas propriedades residências sem querer arcar com o custo elevado de controladores profissionais que são voltados a áreas extensas.

De acordo com Dhatri et al (2019), a utilização da plataforma Arduino e componentes facilmente acessíveis ao público, como válvulas solenoide, display de LCD, relés e sensores de umidade, são uma boa solução para aqueles que desejam criar um sistema de irrigação automático a baixo custo para sua residência.

> A low cost Arduino based Automatic Irrigation system using Soil moisture sensor is presented in this paper in which the soil moisture sensor gives it’s output depending on the conditions of the soil and later with the help of Arduino it get’s worked. (Dhatri et al, 2019, p. 1).

O sistema de irrigação desenvolvido por Dhatri et al (2019) cumpre sua função, o mesmo consegue realizar a rega de forma automatizada das plantas com base no nível de umidade do solo, monitorado pelo sensor de umidade, porém a solução encontrada pelo autor para automatizar a irrigação apresenta alguns problemas que devem ser tratados. Primeiramente, o sistema desenvolvido necessita que o usuário detenha conhecimentos em eletrônica e programação básica, o que restringe seu uso por parte daqueles que não conhecem. Esse conhecimento se faz necessário quando se considera que as rotinas de irrigação só podem ser alteradas com a alteração do código base do microcontrolador, já que não foi implementado a possibilidade de ajustes da rotina, por meio de uma interface homem máquina. Segundo, o número de plantas que podem ser regadas com esse sistema é bastante reduzido já que Dhatri

(2019) faz uso de um módulo relé comercial de apenas um canal, ou seja, apenas uma válvula solenoide pode ser comutada por vez, tornando a rega de áreas médias e grandes inviável. Terceiro, não foi desenvolvida nenhuma PCB customizada que englobe todos os componentes utilizados, forçando o usuário a construir por conta própria o sistema, algo que pode ser uma barreira intelectual por aqueles que não possuem conhecimento em eletrônica e sistemas embarcados, novamente reduzindo o número de pessoas que poderiam usufruir desta tecnologia.

Levando em consideração os problemas previamente apontados com o design de Dhatri et al (2019) algumas modificações e aprimoramentos foram feitos de modo a aumentar o público alvo para o uso deste tipo de tecnologia de automação residencial, assim como a elaboração de uma abordagem mais “user friendly” (facilidade que o usuário tem em utilizar o sistema), ou seja, sem a necessidade de o usuário ter conhecimentos técnicos em eletrônica para a utilização do sistema.

Com o intuito de permitir a rega automatizada para áreas de pequeno, médio e grande porte, foi desenvolvido um módulo de acionamento que contém cinco relés e que pode ser conectado em cascata a outros módulos de acionamento de modo a possibilitar ao usuário o aumento do número de zonas de irrigação, essa abordagem é um aprimoramento significativo se comparado ao módulo relé comercial que contém apenas 1 relé, como o utilizado por Dhatri et al (2019).

Um outro fator a ser considerado é a facilidade que um usuário teria para utilizar o sistema de irrigação, diferentemente do sistema desenvolvido por Dhatri et al (2019), que requer conhecimentos de eletrônica e programação básica por parte do usuário, foi desenvolvido um controlador com uma interface inspirada nos controladores de irrigação comerciais, de modo a possibilitar seu uso por pessoas que não dominam essas áreas técnicas.

2.5 MICROCONTROLADOR

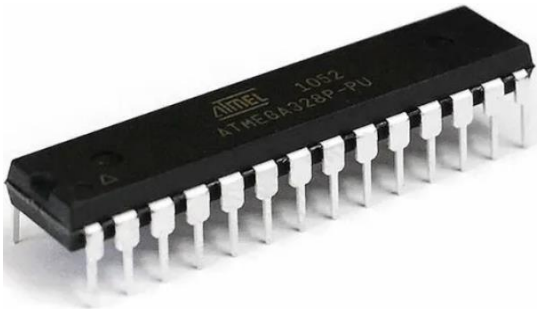
Como descrito por Scherz et al (2013) o microcontrolador é basicamente um computador em um chip, porém sem a presença de um teclado, monitor e mouse. Este dispositivo conta com uma CPU (Central Processing Unit), ROM (Read Only

Memory), RAM (Random Access Memory), portos para comunicação serial e outros componentes dependendo do microcontrolador. Estes dispositivos podem ser utilizados para uma infinidade de produtos e dispositivos eletrônicos que necessitam realizar tarefas repetitivas entre outras aplicações. Atualmente a maior fabricante de microcontroladores é a empresa americana Microchip que em 2016 adquiriu a Atmel, outro gigante no mundo dos microcontroladores.

> The microcontroller is essentially a computer on a chip. It contains a processing unit, ROM, RAM, serial communications ports, ADCs, and so on. In essence, a microcontroller is a computer, but without the monitor, keyboard, and mouse. These devices are called microcontroller because they are small (micro) and because they control machines, gadgets, and so on. (Scherz et al, 2013, p. 859).

O microcontrolador escolhido para o presente trabalho é o ATmega328p, Figura 1, da fabricante de semicondutores Atmel (adquirida pela Microchip). As razões que culminaram em sua escolha são: robustez, vasta documentação, grande popularidade em projetos eletrônicos, baixo custo, extremamente versátil, quantidade significativa de memória RAM e ROM para a aplicação em sistemas embarcados, além de estar integrado em diversas placas Arduino. As especificações do ATmega328p podem ser vistas no Apêndice A.

Figura 1 – Microcontrolador ATmega328p no encapsulamento DIP



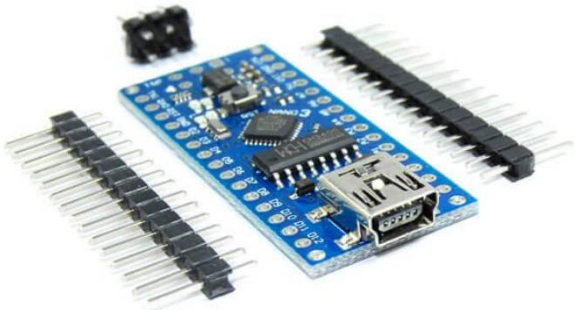
Autoria: https://www.easytronics.com.br/atmega328p-pu

2.6 ARDUINO NANO

Como um dos objetivos do presente trabalho é o de desenvolver um sistema

embarcado, que não somente seja de baixo custo, mas que também possa ser alterado de maneira simples pelo usuário, foi escolhida a placa de desenvolvimento Arduino Nano (5V), Figura 2, que possui vasta documentação e suporte da plataforma Arduino.

Figura 2 – Placa de desenvolvimento Arduino Nano com ATmega328p



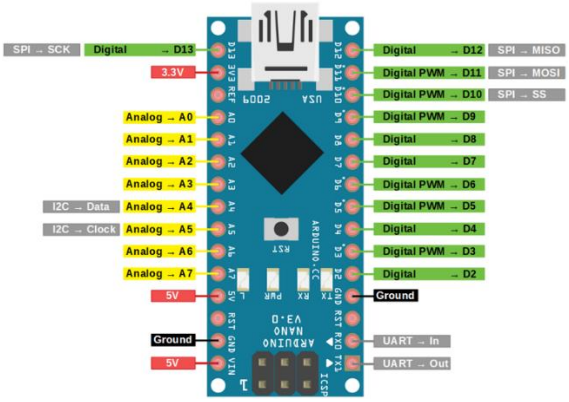
Autoria: https://www.adrobotica.com/produto/arduino-nano-v3-0-atmega328p-ch340g/

Como mencionado por Monk (2012), a placa de desenvolvimento Arduino Nano é excelente para projetos em eletrônica, devido ao fato de ser uma das placas Arduino mais baratas disponíveis no mercado e porque o seu design compacto é compatível com as conexões de *Bread Boards*, tornando a prototipagem de circuitos embarcados mais eficiente e simplificada.

O Arduino Nano possui a vantagem de ter um custo menor do que a soma dos valores de todos os componentes necessários para se utilizar o ATmega328p se comprados separadamente, já que o mesmo necessita de componentes extras para seu correto funcionamento. Esta placa também conta com um conversor USB para Serial já embutido na mesma, não necessitando de adaptadores externos, facilitando a sua programação e diminuindo o custo total do sistema.

Esta placa é voltada para a aprendizagem e desenvolvimento de projetos em eletrônica que envolvem microcontroladores da família AVR da Atmel, além de contar com especificações técnicas adequadas para o projeto (conforme Apêndice B). Ela possui o microcontrolador ATmega328p previamente programado com um bootloader desenvolvido pela plataforma Arduino como forma de facilitar a sua programação. A Figura 3 apresenta um diagrama referente a todos os pinos presentes nesta placa.

Figura 3 – Diagrama descritivo dos pinos da placa Arduino Nano



Autoria: https://lobodarobotica.com/blog/arduino-nano-pinout/

De acordo com Banzi (2011), a programação do microcontrolador pode ser efetuada através do software IDE fornecido pela plataforma Arduino. É através dele que códigos podem ser desenvolvidos e editados utilizando uma variante da linguagem de programação C/C++, o mesmo também é responsável pela compilação do código criado e, com auxílio de um cabo USB, carregado no microcontrolador. Atualmente existem diversas bibliotecas desenvolvidas para auxiliar no desenvolvimento de código para essa plataforma assim como diversos periféricos como módulos, sensores e displays.

Os projetos e esquemas de hardwares são distribuídos sob a licença Creative Commons Attribution Share-Alike 2.5. O código fonte para o IDE e a biblioteca de funções da placa são disponibilizadas sob a licença GPLv2 e hospedadas pelo projeto Google Code.

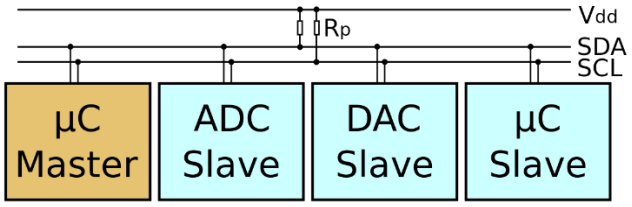
2.7 COMUNICAÇÃO I2C

De acordo com Mendonça (2021), o protocolo de comunicação I2C (Inter-Integrated Circuit) é um barramento serial do tipo “Barramento Multimestre” desenvolvido pela Philips. Esse tipo de comunicação é utilizado para conectar diversos periféricos de baixa velocidade a um sistema embarcado, placa mãe ou telefone celular. O nome significa Circuito Inter-integrado e é comumente referenciado como I2C ou I²C.

Desde o dia 1 de outubro de 2006, o protocolo I2C pode ser implementado sem a necessidade de se pagar nenhuma taxa de licenciamento, contudo, algumas taxas ainda são exigidas para obtenção de endereços escravos I2C.

A Figura 4 exemplifica um barramento I2C, onde existe um dispositivo “Mestre” (amarelo) e 3 dispositivos “Escravos” (azul). A comunicação I2C funciona com apenas 2 duas linhas bidirecionais, uma linha de dados e uma linha de clock, ambas necessitam de um resistor de pull-up para assegurar a robustez da comunicação. As tensões típicas utilizadas são de +5V ou 3,3V, porém outras tensões são permitidas dependendo do sistema. O modelo de referência I2C tem um espaço de endereçamento de 7-bit ou de 10-bit. Normalmente barramentos I2C tem velocidade de 100 kbits/s no modo padrão e de 10 kbits/s no modo de baixa velocidade, mas frequências mais baixas de clock também são permitidas.

Figura 4 – Barramento I2C com 1 mestre e 3 dispositivos escravos



Autoria: https://pt.wikipedia.org/wiki/I%C2%B2C

Em teoria são permitidos até 127 periféricos conectados a um mesmo barramento de 7 bits respeitando a capacitância máxima de 400 pF no barramento e a resistência equivalente dos resistores de pull-up nas linhas SDA (Dados Seriais (Serial Data)) e SCL (Clock Serial (Serial Clock)), caso seja usado módulos que já contenham resistores de pull-up (Req > 1,53KOhm para +5V e Req > 967 Ohms para +3,3V), ou seja, na prática o número pode ser bem menor do que 127.

Pinos para comunicação I2C no Arduino Nano

Embora a placa de desenvolvimento Arduino Nano tenha a função de comunicação I2C não se pode utilizar quaisquer pinos digitais para esta aplicação, sendo necessário utilizar os pinos reservados A4 para o sinal de SDA e A5 para SCL.

2.8 INTERRUPÇÕES

Como descrito por Monk (2012) *interrupts* (interrupções em inglês) são eventos externos que temporariamente suspendem a execução do programa principal para que uma parte específica do código seja executada.

O microcontrolador irá interromper a rotina em execução ao perceber que uma interrupção foi acionada, em seguida irá executar o bloco de código correspondente a interrupção e, ao término da mesma, retornará a rotina original a partir do ponto de onde havia parado.

A função de interrupção é extremamente importante para a programação de microcontroladores pelo fato de que a maioria deles não consegue executar mais de uma tarefa ao mesmo tempo.

Normalmente o microcontrolador executa cada tarefa em determinada ordem e fica “preso” nessa tarefa até completá-la, porém em certos cenários é importante que seja possível interromper qualquer tarefa que o microcontrolador esteja executando para que faça outra de maior prioridade.

Observação: A função Interrupção só é possível graças a uma característica do hardware do ATmega328p, portanto não é possível utilizar qualquer pino para essa aplicação. Na placa Arduino Nano os pinos digitais 2 (INT0) e 3 (INT1) podem ser utilizados com a função de *interrupt*.

2.9 DISPLAY DE CRISTAL LÍQUIDO - LCD

O display de cristal líquido, mostrado na Figura 5, é um componente comumente utilizado com microcontroladores, permitindo uma interface visual entre homem e máquina (HMI em inglês) por meio da visualização gráfica de caracteres exibidos no display.

Figura 5 – Display de Cristal Líquido 16x2

Autoria: https://www.usinainfo.com.br/display-arduino/display-lcd-16x2-com-fundo-verde-2954.html

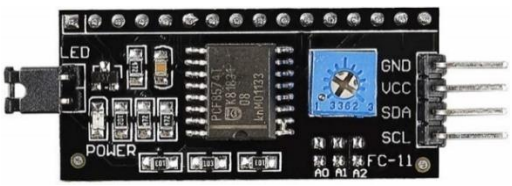
De acordo com Scherz et al (2013) displays de cristal líquido são uma boa solução para aplicações que requerem baixo consumo energético, já que o mesmo utiliza de fontes luminosas externas para a visualização dos caracteres, porém possui a desvantagem de o tempo de resposta para escrita de novos caracteres ser consideravelmente alto, entre 40 a 100 milissegundos, desvantagem essa não significativa para a aplicação do mesmo no controlador de irrigação.

Os números 16x2, encontrado no nome de diversos modelos de LCD, indicam que o mesmo possui 16 colunas por 2 linhas de caracteres que podem ser exibidos ao mesmo tempo no display.

Os módulos de LCD 16x2, em sua maioria, contam um controlador, HD44780 desenvolvido pela Hitachi, que permite a comunicação simplificada entre display e microcontrolador.

A comunicação entre Display e microcontrolador pode ser paralela (4 ou 8 bits) ou serial (I2C) com auxílio de um adaptador, mostrado na Figura 6. Devido ao número limitado de portas digitais em microcontroladores é comum o uso do protocolo de comunicação I2C, já que o mesmo necessita de menos conexões.

Figura 6 – Módulo serial I2C para display LCD



Autoria: https://tecdicas.com/como-ligar-um-display-lcd-16x2-i2c-no-arduino-uno/

O módulo I2C para LCD, Figura 6, conta com 3 pinos, A0, A1 e A2, que podem ser utilizados para alterar o endereço do display caso o mesmo endereço esteja sendo compartilhado com outros dispositivos no barramento I2C. A Tabela 1 mostra as configurações possíveis para alterar o endereço do módulo.

Tabela 1 – Lista de endereços

Endereço	A0	A1	A2
0x20	0	0	0
0x21	1	0	0
0x22	0	1	0
0x23	1	1	0
0x24	0	0	1
0x25	1	0	1
0x26	0	1	1
0x27	1	1	1

Autoria própria

Disposição dos caracteres na tela

Com relação a disposição dos caracteres, a configuração desse tipo de LCD pode ser comparada à de um display que possui 40 colunas por 2 linhas, em que cada [] representa 1 caractere:

[]
[]

Dessa forma em um LCD 16x2 são necessários 40 caracteres para que o 41˚ caractere seja mostrado automaticamente no LCD, por tanto é necessário levar isso em consideração na programação.

Sempre considere um LCD 16x2 como um LCD 40x2 em que apenas os 16 primeiros caracteres estão visíveis na tela.

Em um LCD 10x4 por exemplo é necessário notar que a mesma configuração de 40 colunas por 2 linhas também se aplica, porém de uma forma menos intuitiva, já

que as linhas são dispostas de forma não usual:

[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] → 1˚ Linha de texto;
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] → 3˚ Linha de texto;
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] → 2˚ Linha de texto;
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] → 4˚ Linha de texto;

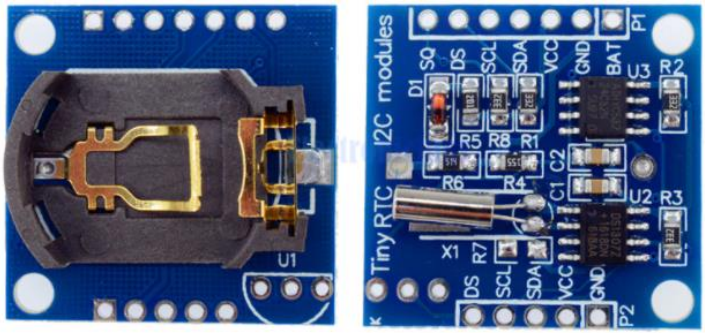
2.10 REAL TIME CLOCK – DS1307

No artigo de Koprda et al (2015) os autores fazem uso de um módulo RTC em conjunto com o microcontrolador como forma de garantir que o sistema de irrigação mantenha a sua precisão temporal durante longos intervalos de tempo, garantindo a rega efetiva e precisa de acordo com a data e hora pré estabelecidos, tudo isso com a vantagem de ser um módulo de baixo consumo energético.

> The circuit DS1302-0902A4 (Figure 3a) was used as a module of real time. The module is able to remember and compensate time (leap year etc.) with an accuracy of seconds until 2100. It is connected to an hour battery and is able to work with the consumption of less than 1μW. It requires connection of at least five wires while two of them serve to connect and the microcontroller communicates through three signal wires SCLK, I/O, CE. (Koprda et al (2015), p. 230).

O módulo RTC (Real Time Clock), Figura 7, é baseado no circuito integrado DS1307, através da comunicação I2C o módulo pode fornecer ao microcontrolador dados referentes a data, como dia da semana, dia do mês, mês, ano e horário, como horas, minutos e segundos, nos formatos de 12 ou 24 horas. Meses com menos de 31 dias e anos bissextos são ajustados automaticamente. As especificações do módulo RTC podem ser vistas no Apêndice C.

Figura 7 – Módulo RTC - DS1307



Autoria: https://pt.aliexpress.com/item/2037925259.html

Uma bateria de lítio (LIR2032) pode ser instalada no módulo para garantir que os dados sejam preservados mesmo sem alimentação externa, e é acionada automaticamente em caso de falta de energia no módulo.

Este módulo ainda conta com a possibilidade de instalação pelo usuário de um sensor de temperatura (DS18B20) em sua PCB (Placa de Circuito Impresso). As informações do sensor de temperatura podem ser acessadas pelo pino DS do módulo.

A Tabela 2 apresenta todos os pinos que estão disponíveis no módulo RTC assim como sua função.

Tabela 2 – Pinos presentes no RTC DS1307

Pinos	Função
SQ	Sinal de Onda Quadrada
DS	Device Select
SCL	Serial Clock - I2C
SDA	Serial Data - I2C
VCC	5V ou 3.3V
GND	Ground
BAT	Nível de carga da bateria

Autoria própria

Detalhamento das funções de cada pino:

- SQ
    - O pino SQ pode ser programado como uma saída de dreno aberto de uso geral ou como uma saída de onda quadrada. A programação é feita através de um registro de controle (palavra de 1 byte).
    - Se o bit SQWE (bit 4 do registro de controle) é 0, então o nível do pino é controlado pelo bit OUT (bit 7 do registro de controle) do registrador de controle 7.
    - Se o bit SQWE for 1, uma onda quadrada é derivada da divisão da frequência do cristal de 32768Hz. Dependendo da configuração dos bits RS0 e RS1, pode-se obter frequências de 1 Hz, 4096 Hz, 8192 Hz e 32768 Hz.
    - Como o pino SQ é uma saída de dreno aberto, é necessário a utilização de um resistor externo de pull-up de 4700 Ohms no sinal.
- DS
    - Utilizado para verificar a temperatura ambiente caso o sensor de temperatura DS18B20 esteja presente no módulo.
- SCL
    - Utilizado na comunicação I2C entre módulo e microcontrolador. O mesmo deve ser conectado ao pino de Serial Clock presente no microcontrolador.
- SDA
    - Utilizado na comunicação I2C entre módulo e microcontrolador. O mesmo deve ser conectado ao pino de Serial Data presente no microcontrolador.
- BAT
    - Utilizado para verificar o nível de tensão da bateria presente no módulo.

2.11 REGISTRADOR DE DESLOCAMENTO – 74HC595

De acordo com Scherz et al (2013) o registrador de deslocamento é um dispositivo que pode ser utilizado para aplicações em que dados precisam ser retidos temporariamente, copiados e/ou deslocados em bits para a esquerda ou direita. O registrador de deslocamento 74HC595 é um dispositivo CMOS Si-gate de alta velocidade composto por uma fila de flip-flops interconectados e é pino compatível com dispositivos Schottky TTL de baixa potência (LSTTL).

> Data words traveling through a digital system frequently must be temporarily held, copied, and bit-shifted to the left or to the right. A

> device that can be used for such applications is the shift register. A shift register is contructed from a row of flip-flops connected so that digital data can be shifted down the row either in a left or right direction. (Scherz et al (2013), p. 802).

O 74HC595 é um registrador de deslocamento serial de 8 estágios com um registrador de armazenamento e saídas de 3 estados. Os registradores têm relógios separados. O mesmo possui uma entrada serial de 8 bits e saída serial ou paralela de 8 bits (conforme Apêndice D).

Os dados são deslocados nas transições positivas da entrada de clock do registrador de deslocamento (SHCP). Os dados em cada registrador são transferidos para o registrador de armazenamento a cada transição positiva do pino de entrada de clock do registrador (STCP - storage register clock input). Se os dois clocks estiverem conectados, o registrador de deslocamento sempre estará um pulso de clock à frente do registrador de armazenamento.

O registrador de deslocamento tem uma entrada serial (DS - serial input) e uma saída padrão serial (Q7S - serial standard output) para conexões em cascata. Também é fornecido um pino de reset assíncrono (active LOW) para todos os 8 estágios de registro de deslocamento. O o registrador de armazenamento tem 8 saídas paralelas de driver de barramento de 3 estados. Dados no registrador de armazenamento aparecem na saída sempre que a entrada de habilitação de saída (OE - Output Enable) está BAIXA.

O registrador de deslocamento 74HC595 é um componente comum em muitos projetos envolvendo microcontroladores por possibilitar a expansão do número de saídas digitais que outrora seria um fator limitante dos microcontroladores para algumas aplicações.

2.12 TRANSDUTOR PIEZOELÉTRICO – BUZZER

O Buzzer, também conhecido como transdutor piezoelétrico, é um dispositivo de áudio capaz de gerar diferentes tons através do efeito piezoelétrico. Piezo eletricidade é a capacidade de alguns cristais gerarem tensão elétrica por resposta a uma pressão mecânica, o contrário também acontece, ou seja, ao se aplicar uma

diferença de potencial neste tipo de cristal o mesmo irá transformar a energia elétrica em energia mecânica dessa forma gerando vibrações que, dependendo da frequência da oscilação, pode ser ouvida.

De acordo com Scherz (2013) estes dispositivos são ótimos para aplicações que envolvem a necessidade de avisos sonoros como em alarmes, também estão presentes em alguns tipos de sensores, alarme de relógios e timers, confirmação da entrada do usuário, como ao pressionar uma tecla ou um botão, entre outras aplicações.

Estes dispositivos, por funcionarem através do efeito piezoelétrico, necessitam receber um sinal alternado, em PWM (Pulse Width Modulation), por exemplo, para que o cristal possa vibrar constantemente a aplicação de uma tensão contínua não fará que o mesmo mantenha a sua vibração, por consequência, não gerando som, a menos que o próprio dispositivo buzzer já possua um oscilador interno.

Diversos tons musicais simples podem ser criados por meio de um sinal PWM, conectado ao buzzer, ao se alterar a frequência do sinal de entrada. A Tabela 3 apresenta algumas das notas geradas por um buzzer ao se aplicar um sinal PWM em diferentes frequências.

Tabela 3 – Frequências relativas a cada nota

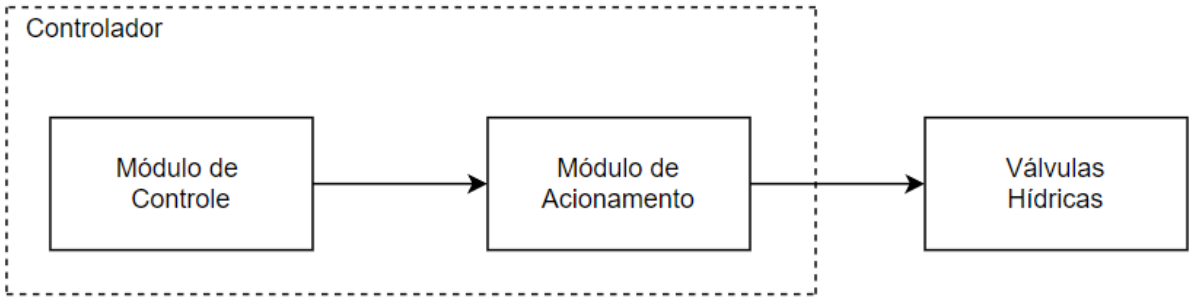
Tom	Frequência (Hz)
Do (C)	262
Re (D)	294
Mi (E)	330
Fa (F)	349
Sol (G)	392
Lá (A)	440
Si (B)	494

Autoria Própria

3 DESENVOLVIMENTO DO TRABALHO

O diagrama de blocos, ilustrado na Figura 8, representa o controlador de irrigação desenvolvido, note que o controlador em si é composto por duas PCIs que são utilizadas de forma conjunta. O Módulo de Controle é responsável pela automação do sistema e interface com o usuário, enquanto o Módulo de Acionamento é responsável por comutar as válvulas hídricas através dos relés disponíveis na placa.

Figura 8 – Diagrama de blocos do sistema

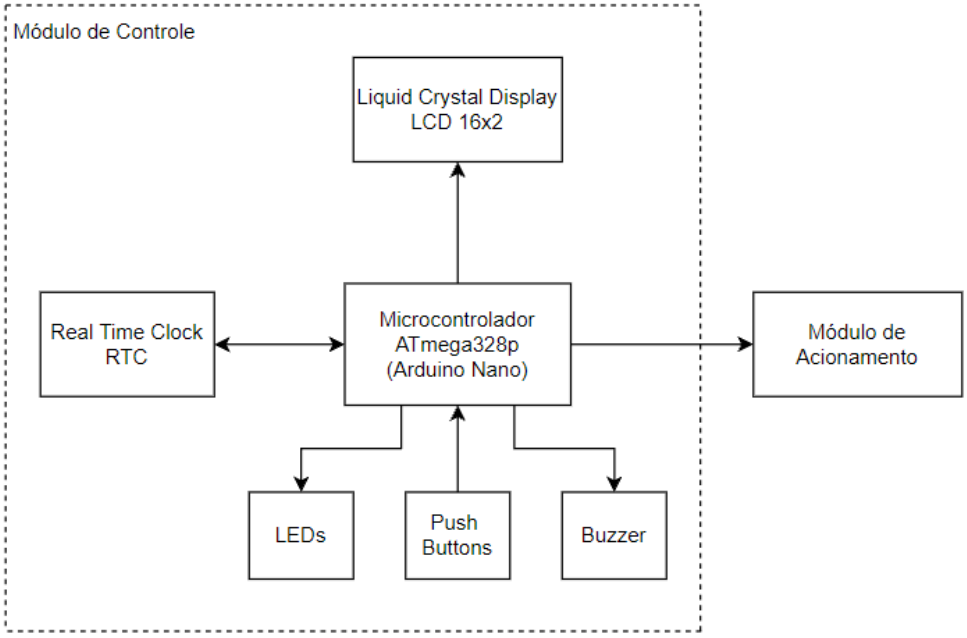


Autoria Própria

3.1 MÓDULO DE CONTROLE

O Módulo de Controle, representado pela Figura 9, é responsável por gerenciar todas as rotinas de irrigação que foram pré-programadas no microcontrolador. Com auxílio de um display de cristal líquido de 16x2 caracteres, LEDs indicativos, buzzer e botões táteis ocorre a interação entre controlador e usuário. Por sua vez, o Módulo de Controle se comunica com o Módulo de Acionamento para comutar o fluxo de água de forma cadenciada.

Figura 9 – Diagrama de blocos do Módulo de Controle



Autoria Própria

Componentes do Módulo de Controle:

- 1x Arduino Nano (Microcontrolador: ATmega328p - Atmel);
- 1x Display de cristal líquido 16x2;
- 1x Módulo Real Time Clock;
- 3x LEDs;
- 7x Botões Táteis;
- 1x Buzzer;
- Componentes auxiliares como resistores, capacitores e conectores.

Conexões entre Arduino Nano e Display

O display de cristal líquido de 16x2 caracteres foi escolhido por seu baixo custo, fácil utilização com microcontroladores e como forma intuitiva para navegação das configurações do controlador pelo usuário. No trabalho foi utilizado o protocolo de comunicação I2C, pois o mesmo necessita de apenas dois pinos para a comunicação, um pino de clock (SCL - Serial Clock) e um de dados (SDA - Serial Data), porém possui a desvantagem de precisar de um adaptador I2C para paralelo.

A utilização da comunicação I2C se faz necessária por dois motivos: Primeiro, o número de entradas e saídas do microcontrolador é limitada, logo a comunicação paralela reduziria ainda mais o número de pinos restantes para os demais componentes; Segundo, esse protocolo de comunicação é utilizado no módulo RTC, também em uso no trabalho, logo ambos os componentes, RTC e Display, podem utilizar o mesmo barramento de comunicação.

A Tabela 4 representa as conexões entre o microcontrolador Arduino Nano e o Módulo I2C - LCD 16x2.

Tabela 4 – Conexões entre Arduino Nano e módulo I2C - LCD 16x2

Pino Arduino Nano	Módulo I2C - LCD 16x2
VCC	VCC (+5V)
GND	GND
A4	SDA - Serial Data
A5	SCL - Serial Clock

Autoria Própria

Conexões entre Arduino Nano e Módulo Real Time Clock

Tendo em vista que as rotinas de irrigação do controlador são coordenadas com base em períodos de tempo e horários pré definidos pelo usuário, é de extrema importância que o microcontrolador possa saber a data e hora exata do dia, tornando o módulo RTC, um componente indispensável para essa aplicação, ainda mais se considerado que o microcontrolador ATmega328p não é capaz de manter a contagem de tempo de forma precisa por longos períodos de tempo.

O módulo RTC escolhido para essa aplicação é o DS1307, facilmente encontrado para compra em sites de produtos eletrônicos e seu preço baixo auxilia em manter o preço final do controlador reduzido. O módulo RTC DS1307 faz uso do protocolo de comunicação I2C, necessitando apenas de dois pinos do microcontrolador para a comunicação. A Tabela 5 mostra as conexões entre Arduino Nano e Módulo RTC.

Tabela 5 – Conexões entre microcontrolador e módulo I2C - LCD 16x2

Pino Arduino Nano	Módulo RTC DS1307
VCC	VCC (+5V)
GND	GND
A4	SDA - Serial Data
A5	SCL - Serial Clock

Autoria Própria

Este módulo também possui a possibilidade de instalação de uma bateria recarregável LIR2032. A instalação dessa bateria possibilita ao módulo continuar a contagem de tempo mesmo se a fonte de alimentação do controlador for interrompida, dessa maneira garantindo que o microcontrolador possa ser resetado ou desconectado com a garantia de que ao se inicializar a data e hora estarão corretos, consequentemente, tornando desnecessário que o usuário ajuste a data e hora no controlador toda vez que o mesmo for desconectado da fonte de alimentação.

Conexões entre Arduino Nano e demais componentes

O Módulo de Controle conta com sete botões táteis, 4 deles dispostos em um formato de D-Pad (similar a controles de vídeo games) para auxiliar o usuário na navegação dos menus no display de LCD, dois deles permitem o usuário as ações de “Confirmar” e “Retornar” enquanto os outros dois permitem incrementar e decrementar os valores das variáveis do controlador. Os demais três botões possuem funções diversas, dois deles foram configurados como *presets* de irrigação e o terceiro botão está conectado a um dos pinos de interrupção do microcontrolador e possui a função de “parar” a rotina de irrigação, mesmo se estiver em andamento, essa é uma medida de segurança para caso a irrigação deva ser interrompida por algum motivo pelo usuário.

Os LEDs são utilizados como uma forma de indicativo luminoso para o usuário e a cor de cada um representa uma mensagem. O LED vermelho ficará acesso caso haja algum problema que demanda a atenção do usuário, o LED verde significa que tudo está “OK”, e por fim, o LED azul ficara acesso para indicar que a irrigação está em andamento, alertando que pelo menos uma das zonas de irrigação está sendo

irrigada pelos aspersores.

O Buzzer foi escolhido para fornecer ao usuário um feedback auditivo, seja durante o momento da configuração das rotinas de irrigação ou para efeitos de alarme de modo a indicar algum problema ou mal funcionamento.

A Tabela 6 mostra as conexões entre Arduino Nano e demais componentes como LEDs, botões táteis e Buzzer utilizados no Módulo de Controle.

Tabela 6 – Conexões entre Arduino Nano e demais componentes

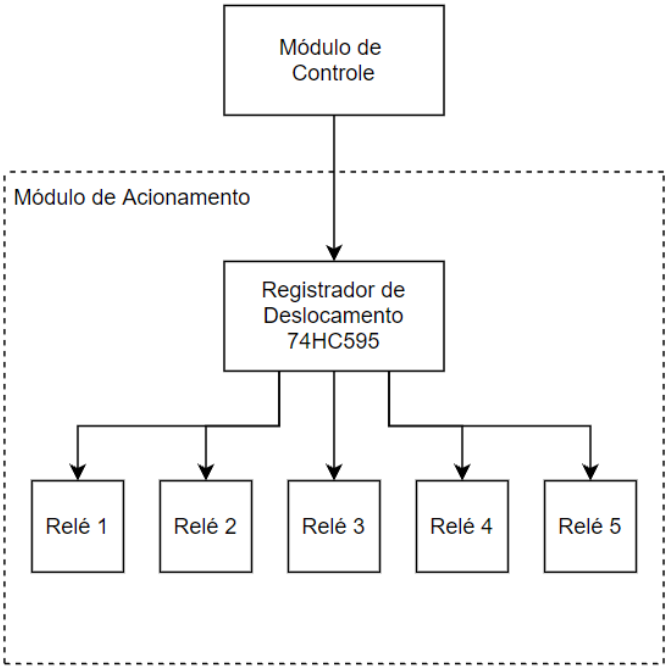
Pino - Arduino	Componente
D2	Sensor (interrupt)
D3	Botão - STOP
D4	Botão - B
D5	Botão - A
D6	LED Vermelho
D7	LED Azul
D8	LED Verde
D9	Buzzer
A2	Botão - DOWN
A3	Botão - ENTER
A4	SDA
A5	SCL
A6	Botão - RETURN
A7	Botão - UP

Autoria Própria

3.2 MÓDULO DE ACIONAMENTO

O Módulo de Acionamento, representado pela Figura 10, conta com um registrador de deslocamento 74HC595 e cinco relés. É através desse módulo que ocorre a comutação dos relés e, por consequência, a comutação das válvulas hídricas para permitir o fluxo de água de forma ordenada para os aspersores.

Figura 10 – Diagrama de blocos do Módulo de Acionamento



Autoria Própria

Componentes do Módulo de Acionamento:

- 1x Registrador de deslocamento 74HC595;
- 5x Relés;
- 5x Leds;
- 5x Transistores bipolares de junção npn;
- Componentes auxiliares como diodos, resistores, capacitores e conectores.

A utilização de um registrador de deslocamento se faz necessário devido ao fato de o número de saídas digitais do microcontrolador ser limitada. A utilização deste componente também permite adicionar a funcionalidade de expansão do número de zonas de irrigação, já que vários Módulos de Acionamento podem ser conectados em cascata, sendo necessário apenas uma alteração no código base do microcontrolador. Essa funcionalidade propicia uma redução de custos por parte dos usuários que necessitam de mais de 5 zonas de irrigação, já que não é necessário a aquisição de outro Módulo de Controle, pois diversos Módulos de Acionamento podem ser conectado ao mesmo Módulo de Controle.

O Módulo de Acionamento é conectado ao Módulo de Controle através de

cinco conexões de controle e dados, além de Vcc e Gnd, apresentado na Tabela 7. Com isso, o microcontrolador consegue se comunicar com o registrador de deslocamento enviando os dados correspondentes a quais zonas de irrigação devem estar ativadas ou desativadas.

Tabela 7 – Conexões entre Arduino Nano e Registrador de Deslocamento

Pino Arduino Nano	Pino 74HC595	Função
D13	13	$\overline{OE}$(enablePin)
D10	14	DS (dataPin)
D11	11	SH_CP (clockPin)
D12	12	$\overline{MR}$ (clearPin)
A0	10	ST_CP (latchPin)
GND	8	GND
VCC(+5V)	16	VCC(+5V)

Autoria Própria

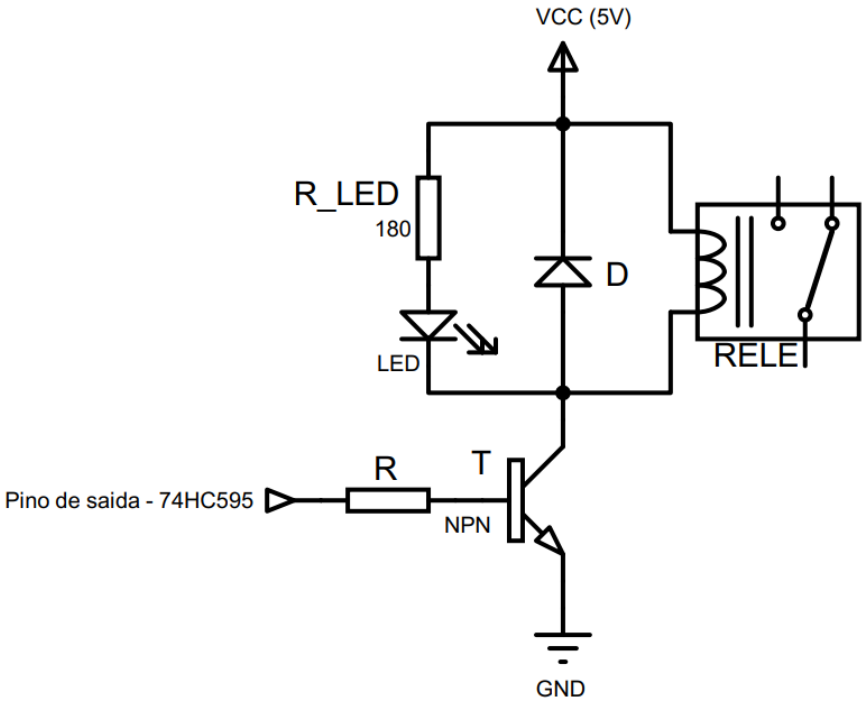
Conexões entre 74HC595 e Relés

Como mostrado na Figura 10, os relés são conectados às saídas do registrador de deslocamento, porém essa conexão não pode ser direta, tendo em vista que os pinos de saída do CI 74HC595 não podem suprir uma corrente de dreno de mais de 40mA, corrente essa não suficiente para energizar a bobina do relé. A solução encontrada para este problema foi a adição de um transistor bipolar de junção do tipo npn por relé, de modo que o sinal do pino de saída do CI esteja conectado a base do transistor, portanto ao receber um sinal lógico HIGH (+5V) o transistor, permite a passagem de corrente energizando a bobina do relé e ao receber um sinal lógico LOW (0V) o transistor não fica polarizado, impedindo a comutação do relé.

Um outro fator importante a ser considerado é que a tensão, assim como a corrente, gerada nos terminais da bobina durante o colapso do campo magnético, ocasionado no momento de seu desligamento, pode ser prejudicial ao circuito de controle, que geralmente opera com tensões bem mais baixas como 5V ou 3,3V. Portanto, se faz necessário a adição de um diodo de roda livre por relé para redirecionar essa corrente para que ela retorne e se dissipe na própria bobina. A Figura 11 exemplifica um circuito envolvendo o acionamento de um relé de dois

contatos por um dos pinos do registrador de deslocamento.

Figura 11 – Esquemático acionamento relé



Autoria Própria

Cálculo do valor do resistor

O valor da resistência R, mostrado na Figura 11, pode ser calculado com base nas especificações dos componentes escolhidos, como transistor, led e relé.

Primeiramente foi verificado, com auxílio de um multímetro, que a resistência da bobina é de aproximadamente 90Ω. Com essa informação e a tensão de energização da bobina (+5V) foi possível calcular a corrente mínima necessária para energizar a bobina e, consequentemente, comutar o relé:

$$V = I * R \rightarrow I = \frac{V}{R} \rightarrow I = \frac{5V}{90\Omega} \rightarrow I = 55{,}5mA$$

Um led azul está conectado em paralelo aos terminais do relé para indicar o estado atual do mesmo. O led, por sua vez, está conectado em série a um resistor de

180Ω e é alimentado pelos mesmos 5V. Considerando uma queda de tensão de 3V no led, foi calculado a corrente fornecida ao led:

$$V = I * R \rightarrow I = \frac{V}{R} \rightarrow I = \frac{5V - 3V}{180\Omega} \rightarrow I = 11{,}1mA$$

De acordo com o esquemático da Figura 11 a corrente de energização da bobina e a corrente que atravessa o led são as mesmas que passam pelo coletor do transistor, portanto temos que Ic = 55,5mA + 11,1mA = 66,6mA. Com auxílio do datasheet do transistor npn 2n2222a temos que β=100, com isso foi possível calcular o valor da corrente de base Ib do transistor:

$$I_c = \beta * I_b \rightarrow I_b = \frac{I_c}{\beta} \rightarrow I_b = \frac{66{,}6mA}{100} \rightarrow I_b = 0{,}67mA$$

Como calculado, o valor mínimo para a corrente de base é de 0,67mA para comutar o relé e acender o led, portanto podemos escolher um valor de resistor que garanta uma corrente de base entre 0,67mA e 40mA (máxima corrente de dreno dos pinos de saída do CI 74HC595). Para este cálculo foi considerado uma tensão de +5V no pino de saída do CI assim como uma tensão Vbe ≈ 0,7V.

Considerando um resistor de 1KΩ, temos uma corrente de base de 4,3mA. Corrente essa, suficiente para comutar o relé e acender o led de status.

$$V = I_b * R \rightarrow I_b = \frac{V}{R} \rightarrow I_b = \frac{(5 - 0{,}7)V}{1000\Omega} \rightarrow I_b = 4{,}3mA$$

Portanto, R = 1KΩ.

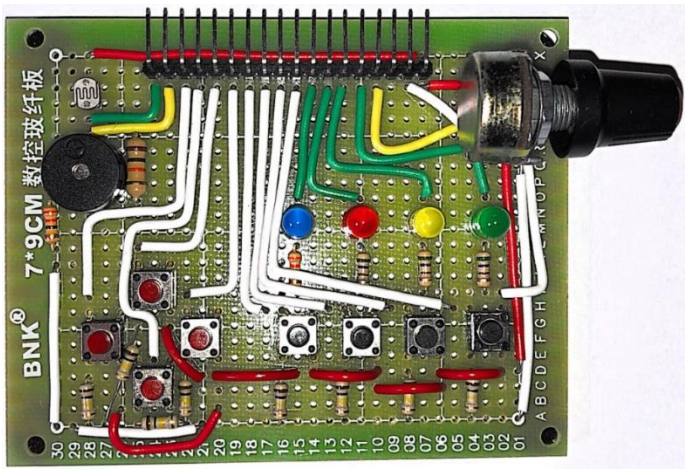
3.3 PROTÓTIPOS

De modo a simplificar o processo de desenvolvimento de hardware e software, assim como minimizar possíveis erros provenientes de mau contato e interferência devido a grande quantidade de conexões (indutância parasita causada por jumpers na Bread Board), diversos protótipos foram construídos em placas de prototipagem.

Protótipo 1 - Placa de Interface com o Usuário

O objetivo desta placa foi auxiliar no desenvolvimento do controlador final referente a interação entre homem e controlador de irrigação, verificando quais componentes são realmente necessários de modo que a placa controladora seja intuitiva e de fácil entendimento pelo usuário. A vista superior da placa é mostrada na Figura 12.

Figura 12 – Protótipo de interface com o usuário



Autoria Própria

Componentes do Protótipo 1:

- 1x Sensor LDR (Light Dependent Resistor);
- 1x Buzzer;
- 1x Potenciômetro de 100K Ohms;
- 4x LEDs;

- 8x Botões Táteis.

Essa placa marca o início do desenvolvimento do controlador de irrigação, como, em um primeiro momento, não se tinha conhecimento de como a interface entre homem máquina seria implementada, diversos componentes foram soldados e seus terminais conectados ao header superior para fácil conexão entre placa e microcontrolador.

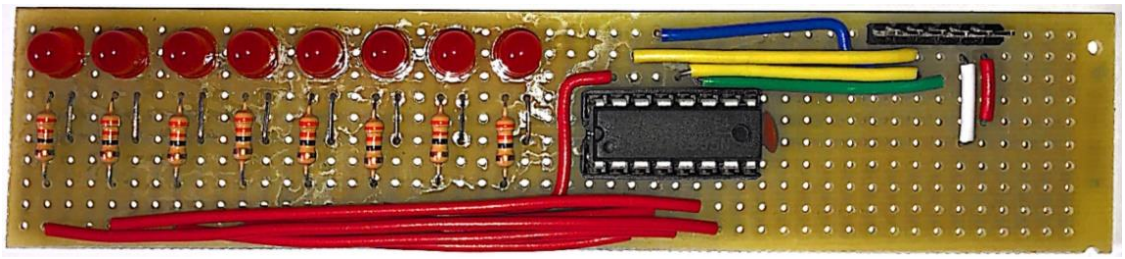
Os botões da placa são utilizados para auxiliar o usuário na navegação de menus, exibidos no display, e para configurar o controlador. Os LEDs servem como indicativo luminoso para diferentes ações como informar o status do controlador (Verde informa que o controlador está pronto para ser usado e o Azul para informar que a irrigação está em andamento) e para avisos de alerta (Vermelho para informar sobre problemas).

Durante a fase de desenvolvimento, verificou-se a não necessidade dos componentes LDR, potenciômetro e LED amarelo, portanto os mesmos não foram incorporados na versão final. Estes componentes não tinham uma funcionalidade específica no projeto apenas foram adicionados na placa do Protótipo 1 como uma garantia de que seria fácil testa-los caso fosse aprimorado no projeto posteriormente.

Protótipo 2 - Placa de Testes do Registrador de Deslocamento 74HC595

O objetivo desta placa foi auxiliar no desenvolvimento do Módulo de Acionamento referente ao entendimento do funcionamento do registrador de deslocamento 74HC595. Seguindo as informações técnicas presentes no datasheet do componente foi confeccionada a placa de testes apresentada na Figura 13, no qual as saídas paralelas do registrador de deslocamento foram conectadas a LEDs como forma de indicar o estado (ON ou OFF) de cada pino da saída de dados e para simular os relés que posteriormente seriam conectados ao CI. Um header macho de 6 pinos foi utilizado para facilitar a conexão entre protótipo e microcontrolador. As conexões entre o registrador de deslocamento e header é apresentado na Tabela 8.

Figura 13 – Protótipo do Módulo de Acionamento



Autoria Própria

Tabela 8 – Função de cada pino do header macho

Pino	Função
1	$\overline{OE}$
2	DS (dataPin)
3	ST_CP (latchPin)
4	SH_CP (clockPin)
5	GND
6	VCC(+5V)

Autoria Própria

Componentes do Protótipo 2:

- 1x Registrador de deslocamento 74HC595;
- 8x LEDs conectados ao registrador;
- 8x Resistores;
- 1x capacitor de 10uF.

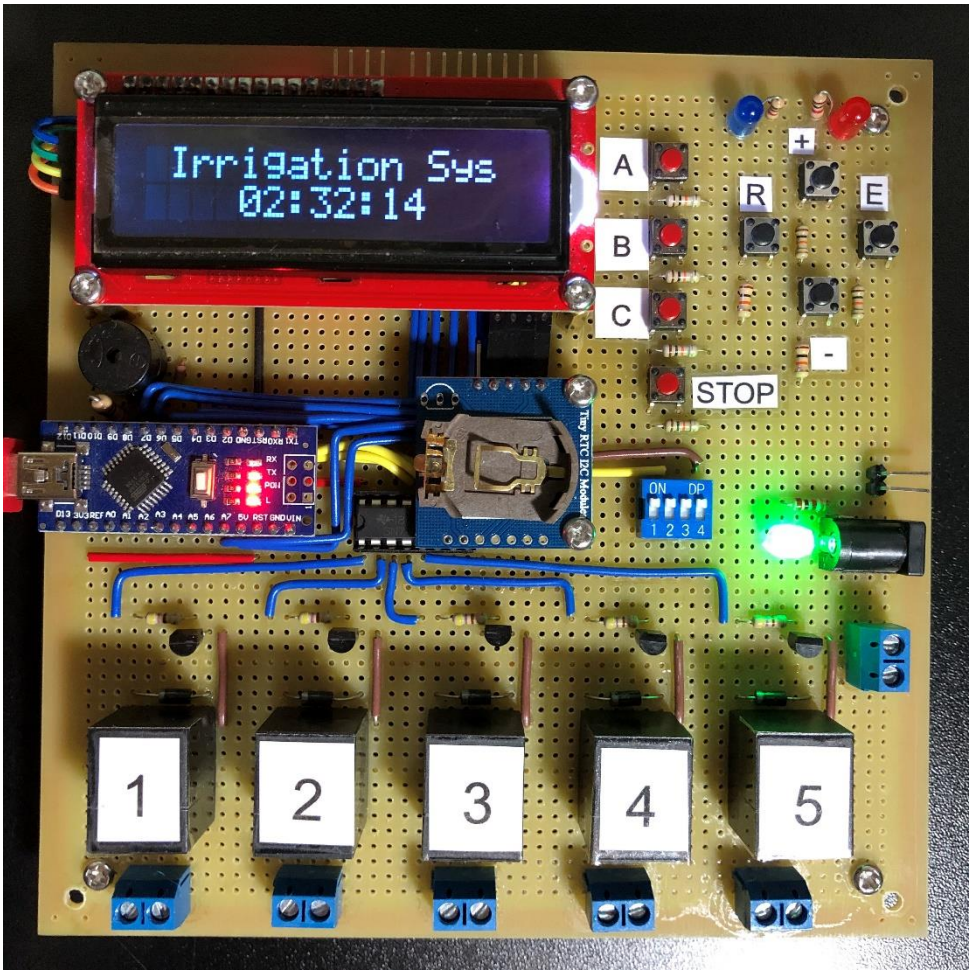
Durante a fase de testes os pinos de dados foram conectados ao Arduino Nano e uma série de bits foram mandados à placa de modo a verificar o funcionamento e se o mesmo se comportava como previsto. Após o término dos testes verificou-se que os bits mandados pelo microcontrolador correspondiam com a ordem dos LEDs acesos, comprovando o funcionamento do CI e a comunicação entre protótipo e microcontrolador.

Protótipo 3 - Controlador Completo

Esse protótipo foi criado com o objetivo de se ter uma visão melhor de como seria o produto final além de auxiliar no desenvolvimento do código em C/C++ que comanda as funcionalidades do sistema embarcado.

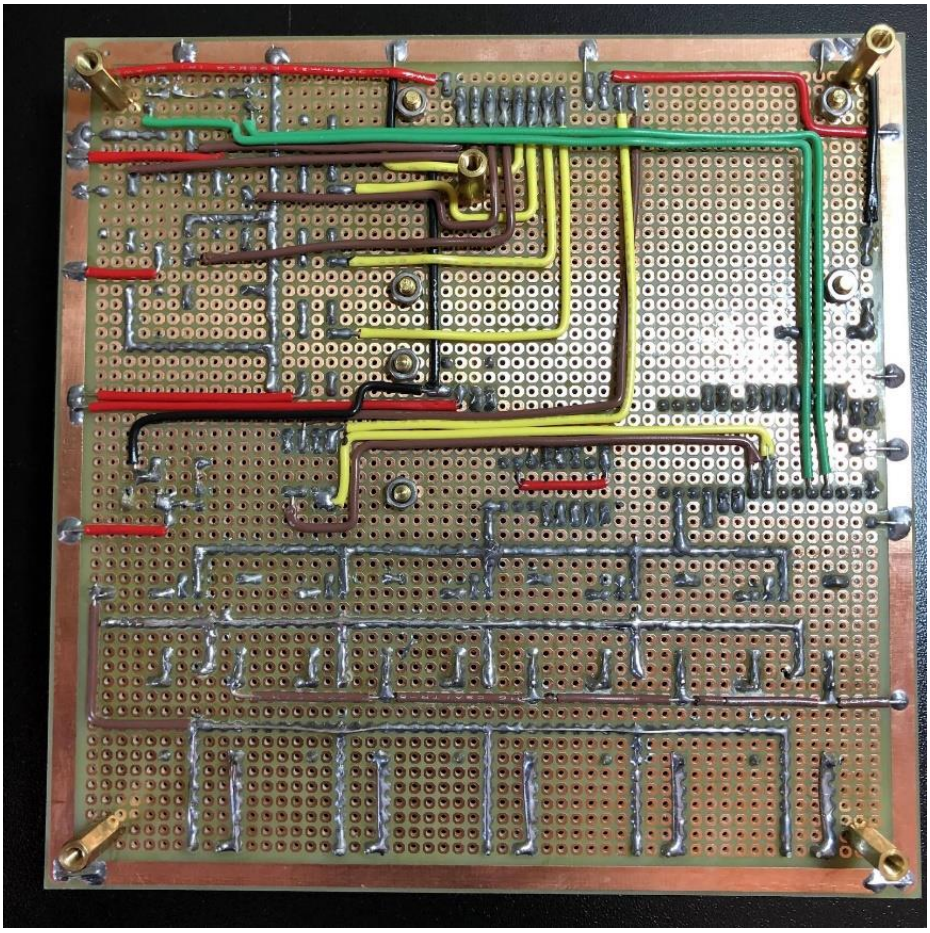
A placa foi soldada manualmente durante um período de 16 horas. A soldagem das vias e dos componentes reduz significativamente a possibilidade de mau contato e interferência (indutância e capacitância parasitas), comuns em circuitos montados na Bread Board. A Figura 14 mostra a vista superior da placa, enquanto a Figura 15 a vista inferior.

Figura 14 – Vista superior do protótipo do controlador completo



Autoria Própria

Figura 15 – Vista inferior do protótipo do controlador completo



Autoria Própria

Componentes do Controlador Completo:

- 1x Arduino Nano 5V;
- 1x Display LCD 16x2 - I2C;
- 1x Módulo RTC (Real Time Clock) - I2C;
- 1x Registrador de deslocamento 74HC595;
- 1x Buzzer;
- 5x Relés;
- 5x Transistores npn 2N2222;
- 5x Diodos 4007;
- 1x DIP Switch 4 canais;
- 8x Push Buttons;
- 3x LEDs;
- 6x Conectores de 2 vias;

- 1x DC Barrel Power Jack;
- 16x Resistores de valores variados;
- 1x Placa de fenolite perfurada de 15cm x 15 cm.

Embora o controlador final seja composto por duas PCIs, sendo o Módulo de Controle e Módulo de Acionamento, por motivos de simplicidade para os testes de bancada foi confeccionado uma placa de desenvolvimento que contém todos os componentes soldados na mesma placa. Esse protótipo auxiliou na verificação da escolha e disposição física dos componentes.

Por motivos de teste de bancada e *debugging*, essa placa também conta com um DIP Switch de 4 conexões conectado ao barramento de dados I2C que foi utilizado para os testes de comunicação do LCD e RTC. Através deste DIP Switch foi possível escolher a qual componente o microcontrolador estaria conectado, assim como permitir a comunicação de ambos ao mesmo tempo no barramento I2C ou de nenhum.

Durante a fase de testes, foi verificado o correto funcionamento da comunicação entre todos os componentes presentes na placa e microcontrolador, comprovando que todas as conexões estavam de acordo. Essa placa também foi utilizada para dar início a programação em C/C++ do sistema embarcado.

Com a fase de testes terminada, foi possível dar início ao desenvolvimento da versão final do controlador de irrigação. Para tal, foi utilizado o software Proteus 8 para a criação do esquemático elétrico final assim como as PCIs do Módulo de Controle e Acionamento.

3.4 CONTROLADOR FINALIZADO

Após a realização dos testes de bancada efetuados nos protótipos e a comprovação de seu funcionamento, o desenvolvimento da versão final do Controlador de Irrigação foi iniciado.

O controlador final é composto por duas PCIs funcionando de forma conjunta, sendo o Módulo de Controle, responsável pelas rotinas de irrigação e interface com

o usuário, e o Módulo de Acionamento, responsável por comutar as válvulas hídricas conectadas aos relés da mesma.

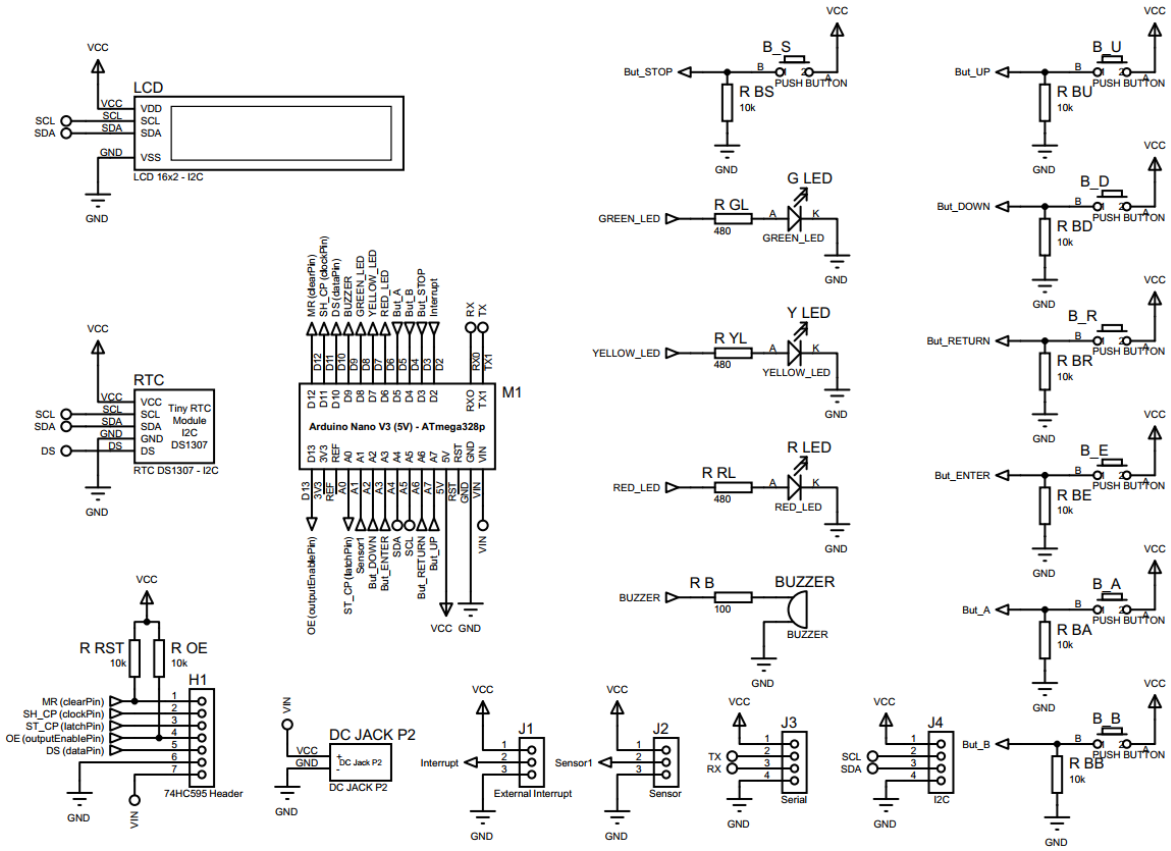
Com auxílio do software Proteus 8 foi desenvolvido as versões finais dos circuitos de ambos os módulos assim como suas respectivas PCIs. Durante a fase de desenvolvimento do hardware do controlador foram criadas diversas bibliotecas de componentes para o módulo RTC, LCD 16x2, botões, Arduino Nano e conector DC no Proteus 8 tendo em vista que esses componentes não estavam disponíveis no software básico. Enquanto que para a programação do sistema embarcado foi utilizado a IDE Arduino v1.8.9 de 64bits para Windows.

De modo a reduzir ao máximo as dimensões físicas da PCI, e consequentemente o tamanho do controlador completo, foi utilizado um design de PCI que utiliza duas camadas de cobre para as vias de ligação entre componentes.

Módulo de Controle

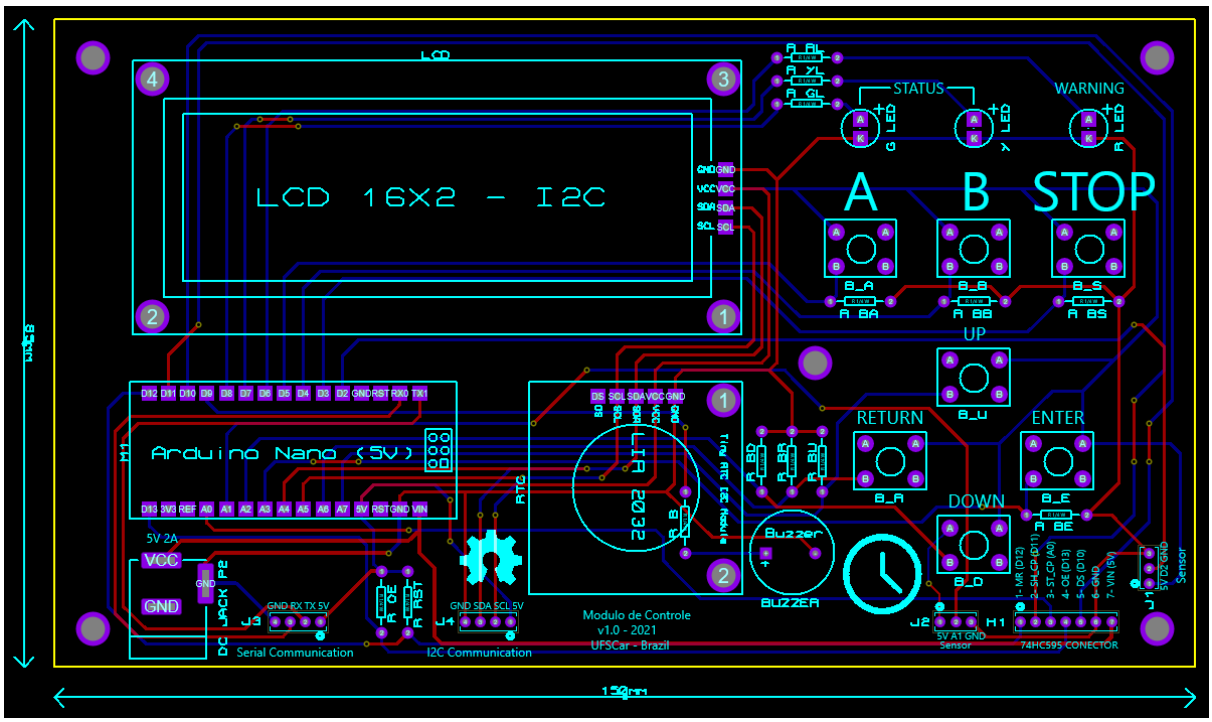
Com auxílio do software Proteus 8 foi desenvolvido o esquemático do circuito do Módulo de Controle assim como o layout da PCI, mostrados, nas Figuras 16 e 17, respectivamente. As Figuras 18 e 19 mostram a renderização 2D da PCI e a renderização 3D do circuito montado, respectivamente.

Figura 16 – Esquemático do circuito do Módulo de Controle



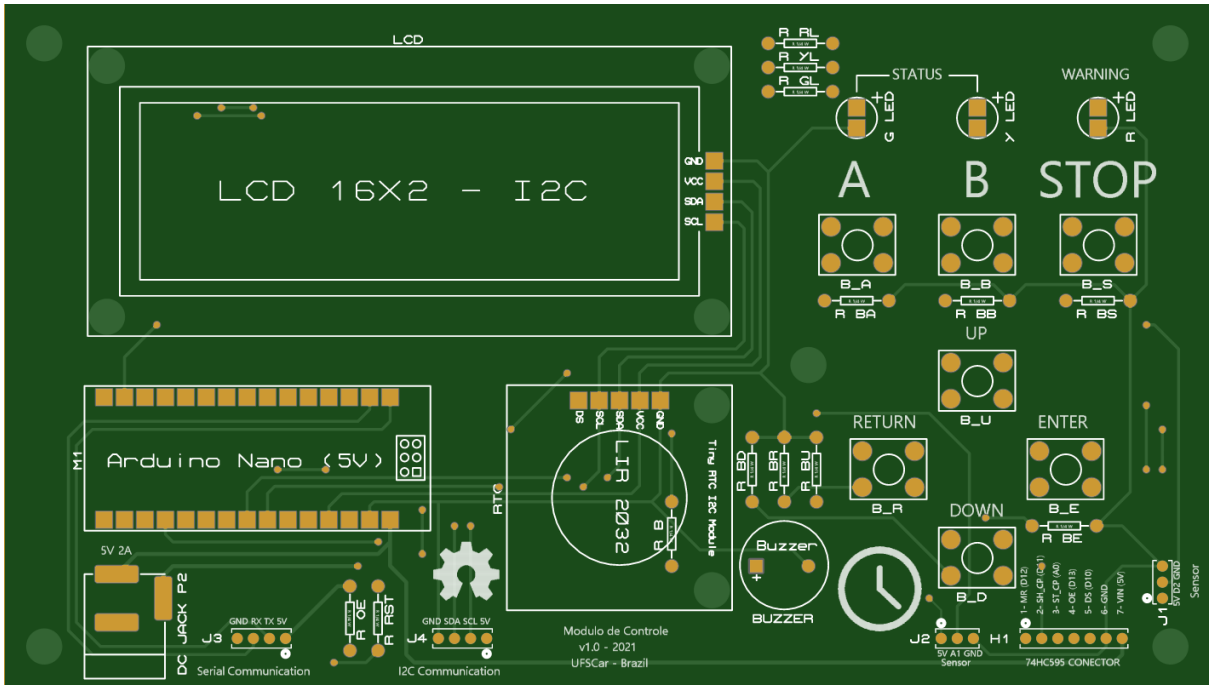
Autoria Própria

Figura 17 – PCI do Módulo de Controle



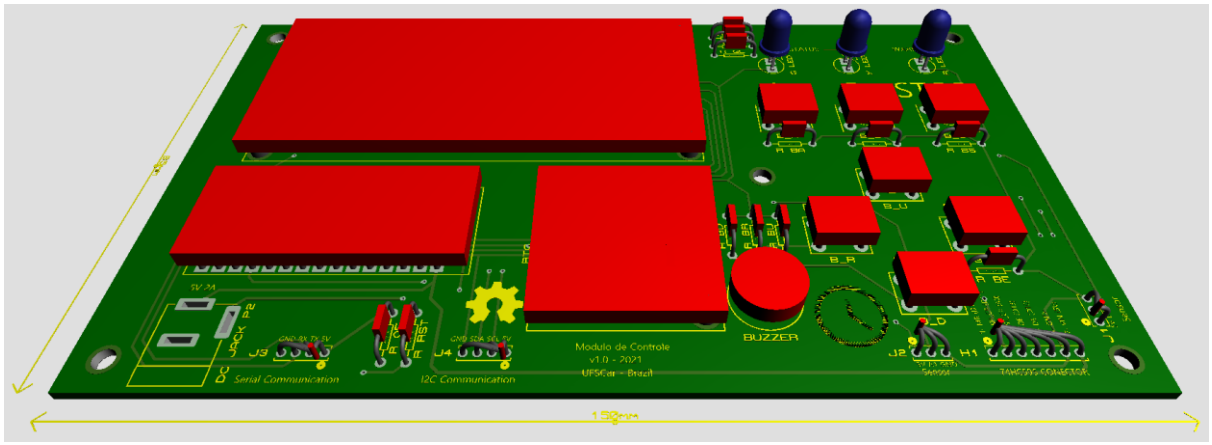
Autoria Própria

Figura 18 – Vista Superior 2D da PCI do Módulo de Controle



Autoria Própria

Figura 19 – Vista Superior 3D do Módulo de Controle



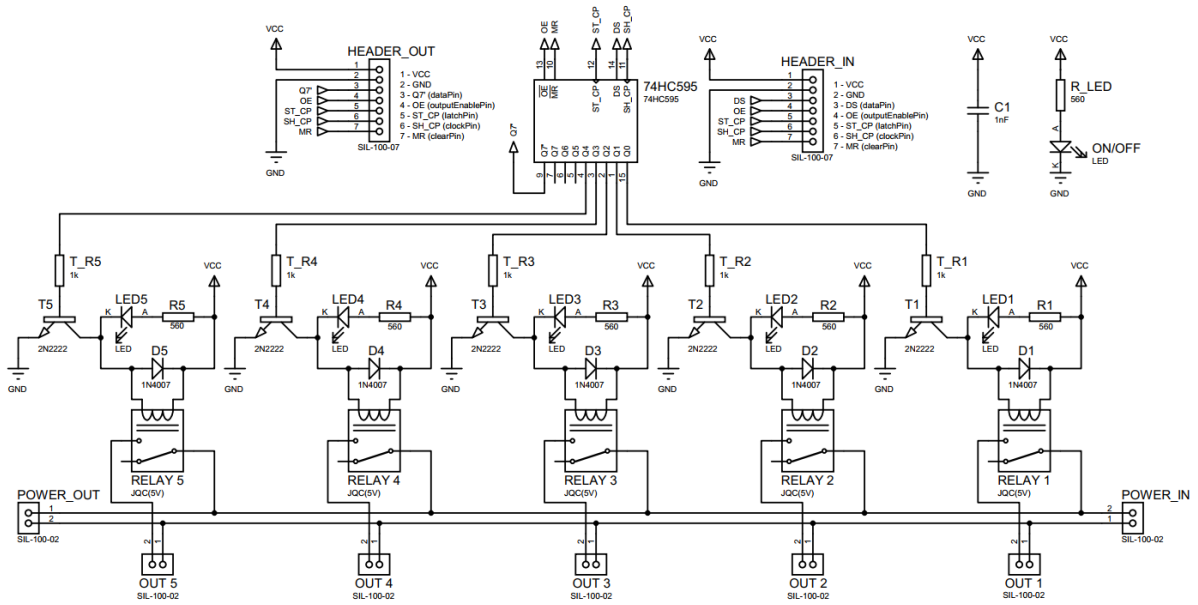
Autoria Própria

A placa de circuito impressa desenvolvida para o Módulo de Controle possui a dimensão física de 85mm x 150 mm. Ao longo do desenvolvimento da PCI alguns recursos extras foram adicionados na mesma, tais como: 1x conector para comunicação Serial, 1x conector para comunicação I2C, 1x conector para se adicionar um sensor que utilize apenas 1 pino digital (Pino D2) e 1x conector para se adicionar um sensor que utilize apenas 1 pino analógico (Pino A1).

Módulo de Acionamento

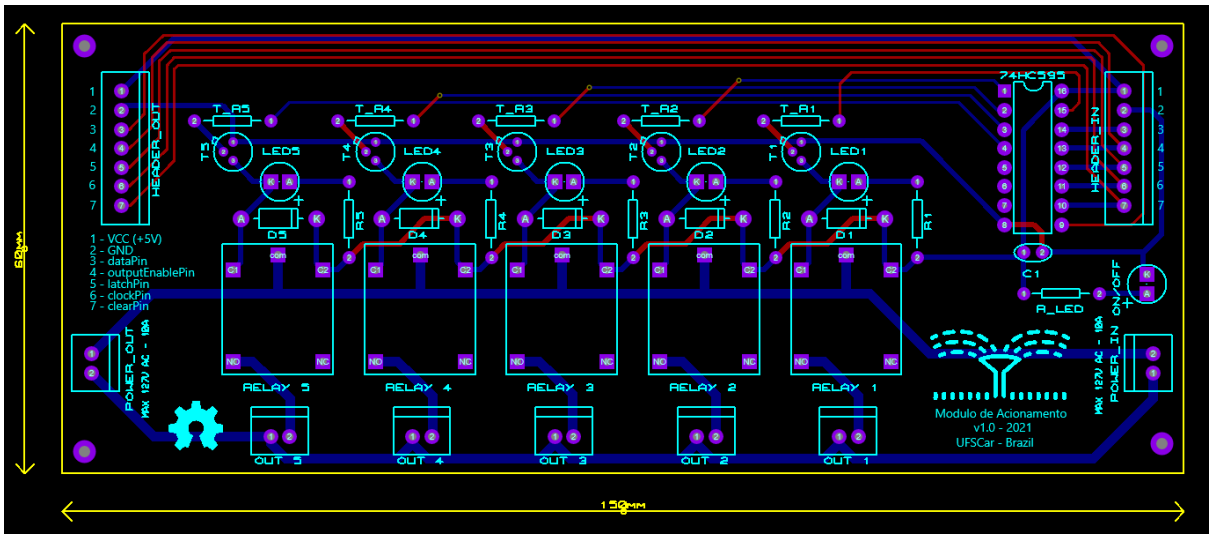
Com auxílio do software Proteus 8 foi desenvolvido o esquemático do circuito do Módulo de Acionamento assim como o layout da PCI, mostrados, nas Figuras 20 e 21, respectivamente. As Figuras 22 e 23 mostram a renderização 2D da PCI e a renderização 3D do circuito montado, respectivamente.

Figura 20 – Esquemático do circuito do Módulo de Acionamento



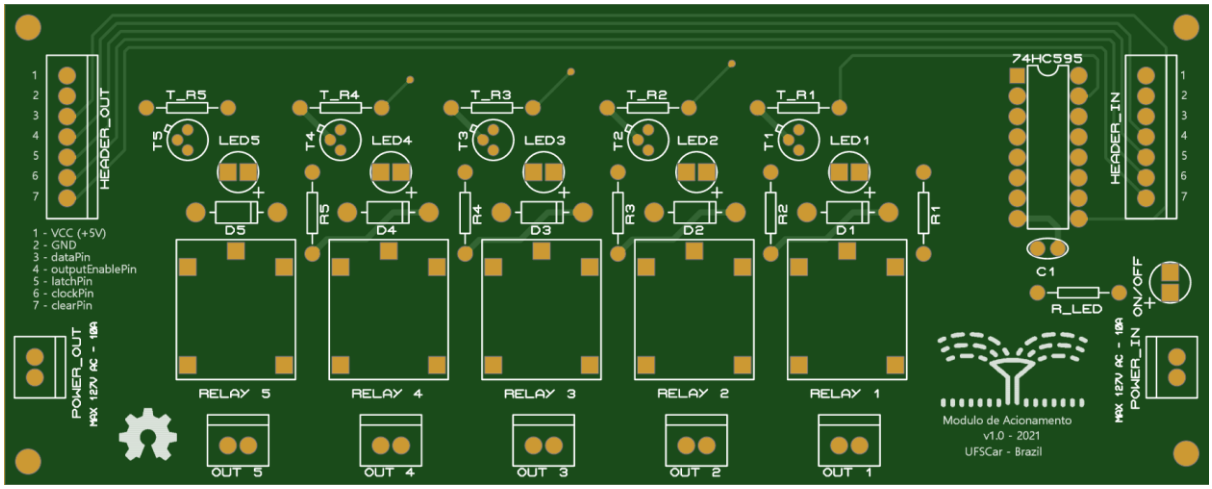
Autoria Própria

Figura 21 – PCI do Módulo de Acionamento



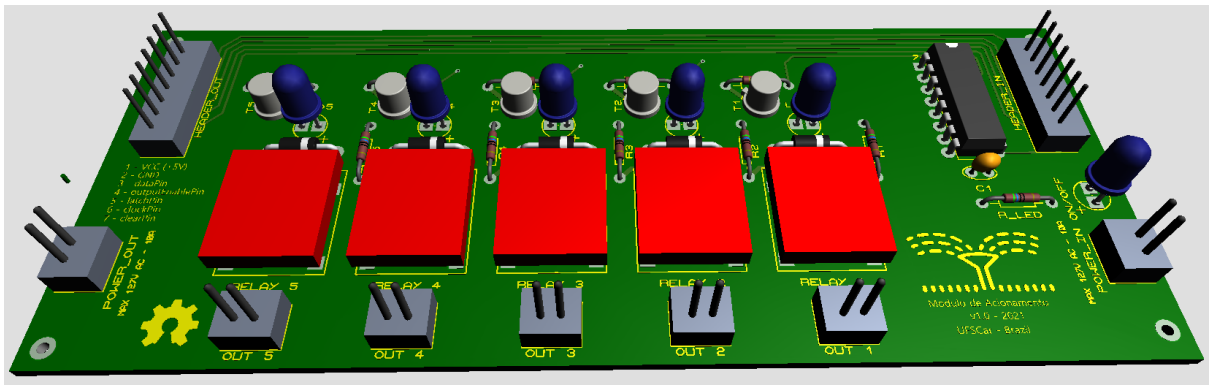
Autoria Própria

Figura 22 – Vista Superior 2D da PCI do Módulo de Acionamento



Autoria Própria

Figura 23 – Vista Superior 3D do Módulo de Acionamento



Autoria Própria

A placa de circuito impressa desenvolvida para o Módulo de Acionamento possui a dimensão física de 60mm x 150 mm. Ao longo do desenvolvimento da PCI foi inserido a funcionalidade de visualizar, por meio de LEDs, o status de cada relé, ou seja, o LED ficara aceso quando o relé estiver ativado e desligado quando o relé estiver desativado.

Embora existam módulos relé comerciais disponíveis no mercado, optou-se pelo desenvolvimento de um customizado tendo em vista que, graças ao registrador de deslocamento, diversos Módulos de Acionamento podem ser conectados em cascata de forma a ampliar o número de zonas de irrigação e essa funcionalidade não é encontrada em módulos relé comerciais.

3.5 DISTRIBUIÇÃO HÍDRICA

Um dispositivo de distribuição de água, Figura 24, foi desenvolvido com o propósito de disponibilizar quatro saídas de água através de uma única fonte. Esse tipo de adaptador hídrico se faz necessário quando o usuário pretende utilizar diversas zonas de irrigação independentes, porém não possui o número necessário de pontos de água no local. Através desse dispositivo o usuário consegue comutar a vazão hídrica de até quatro zonas de irrigação, podendo ser expandido para mais zonas caso necessário, fazendo uso de apenas um ponto de água.

Figura 24 – Dispositivo de distribuição hídrica

Autoria Própria

Componentes do dispositivo de distribuição hídrica:

- 3x Conectores tipo T fêmea de ½";
- 4x Conectores macho-macho de ½";
- 4x Adaptadores macho-fêmea de ½" para ¾";
- 4x Válvulas solenoide 127V AC de ¾";
- 1x Cotovelo 90° fêmea-fêmea de ½";

- 4x Adaptadores de ¾” para mangueira;
- 1x Adaptador de ½” para mangueira.

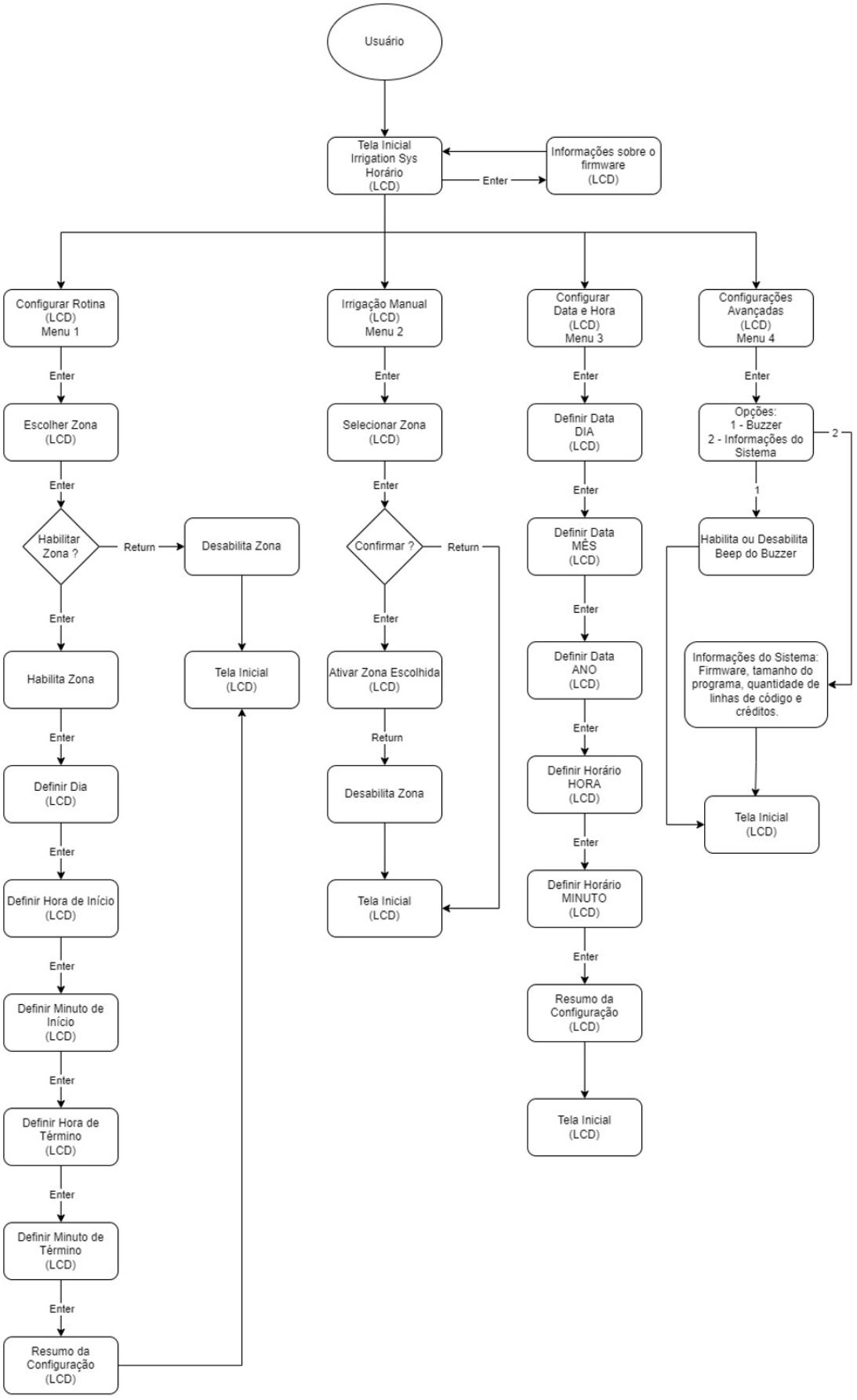
A desvantagem deste tipo de adaptador provém do fato de que todas as zonas de irrigação fazem uso da mesma fonte de água, portanto, caso todas as n zonas de irrigação estejam ativadas ao mesmo tempo, a pressão hídrica proveniente de um único ponto de água será distribuída entre essas n zonas. A pressão hídrica é um fator importante a ser considerado em um projeto de irrigação, já que os aspersores necessitam de uma pressão mínima para o seu correto funcionamento. Uma solução para este problema é ativar uma zona de irrigação por vez, desta maneira a pressão hídrica não será substancialmente afetada, em contrapartida o tempo total da rotina de irrigação será maior. Outra solução, porém mais custoso para o usuário, é a aquisição de uma bomba hídrica para aumentar a pressão da vazão de água.

3.6 PROGRAMAÇÃO

O desenvolvimento do código foi realizado no IDE Arduino versão 1.8.9 para Windows 64bits, diversas bibliotecas como LiquidCrystal_I2C.h, Wire.h e RTClib.h foram utilizadas na programação para otimização do código.

A Figura 25 apresenta um fluxograma de alto nível de como o controlador irá atuar durante a fase de configuração realizada pelo usuário. De modo a facilitar a interação entre usuário e controlador, foi desenvolvido um sistema de Menus e Submenus que separam as diversas funções do controlador.

Figura 25 – Fluxograma



Autoria Própria

Como apresentado na Figura 25, as configurações do controlador são baseadas em um sistema de Menus, no qual o usuário pode entrar em cada menu separadamente e configurar o mesmo. Os menus foram criados de tal forma que todas as informações relevantes sejam mostradas na própria tela do controlador, não necessitando assim de um manual de instruções para configura-lo.

A navegação nos menus e as configurações são feitas utilizando-se quatro botões, “Up” e “Down” para aumentar ou diminuir o valor exibido e “Enter” e “Return”, para selecionar/confirmar uma ação ou retornar/cancelar, respectivamente.

Os botões A e B, que podem ser vistos nas Figuras 16, 17 e 18, foram configurados para iniciar uma rotina de irrigação rápida:

- Ao se pressionar o botão “A” a Zona 1 será regada durante o intervalo pré definido de 30 minutos;
- Ao se pressionar o botão “B” a Zona 1 será regada durante o intervalo pré definido de 60 minutos.

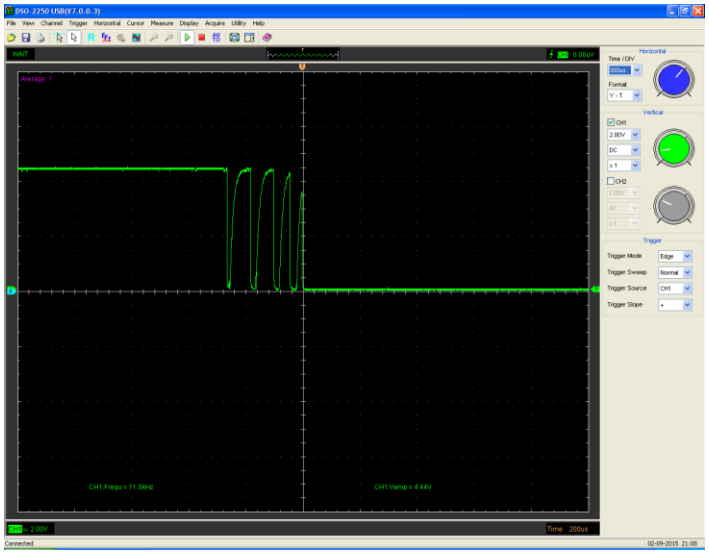
O botão “STOP”, que pode ser visto nas Figuras 16, 17 e 18, está conectado ao pino de interrupção do Arduino Nano, portanto, ao ser pressionado, o mesmo irá cancelar quaisquer rotinas de irrigação que estejam em andamento. Essa funcionalidade é uma medida de segurança do controlador, já que erros, problemas ou imprevistos podem acontecer, logo se faz necessário uma maneira de interromper o fluxo de água nos aspersores com rapidez.

Debouncing dos botões táteis

Durante a programação, um fator importante que foi considerado é o efeito de *bouncing* que ocorre ao se pressionar um botão tátil. Embora este tipo de interruptor seja muito útil para diversas aplicações, o mesmo apresenta um problema que deve ser tratado para o correto funcionamento do circuito. O efeito de *Bouncing*. Ao se pressionar um botão táctil os contatos metálicos se encostam para fechar o circuito, porém, devido a maleabilidade do metal, esse contato não é instantâneo. O que ocorre é que ao se pressionar o botão os contatos “batem” um no outro repetidamente, gerando vários sinais de ALTO e BAIXO durante um período curto de tempo, este fenômeno é chamado de *Bouncing*.

Normalmente, os interruptores demoram alguns microssegundos a alguns milissegundos para fechar completamente. Isso significa que em termos de lógica digital é como se o interruptor estivesse fisicamente abrindo e fechando inúmeras vezes durante esse curto espaço de tempo. A Figura 26 demonstra esse efeito.

Figura 26 – Exemplo do efeito de bouncing registrado ao se pressionar o botão



Autoria: https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/

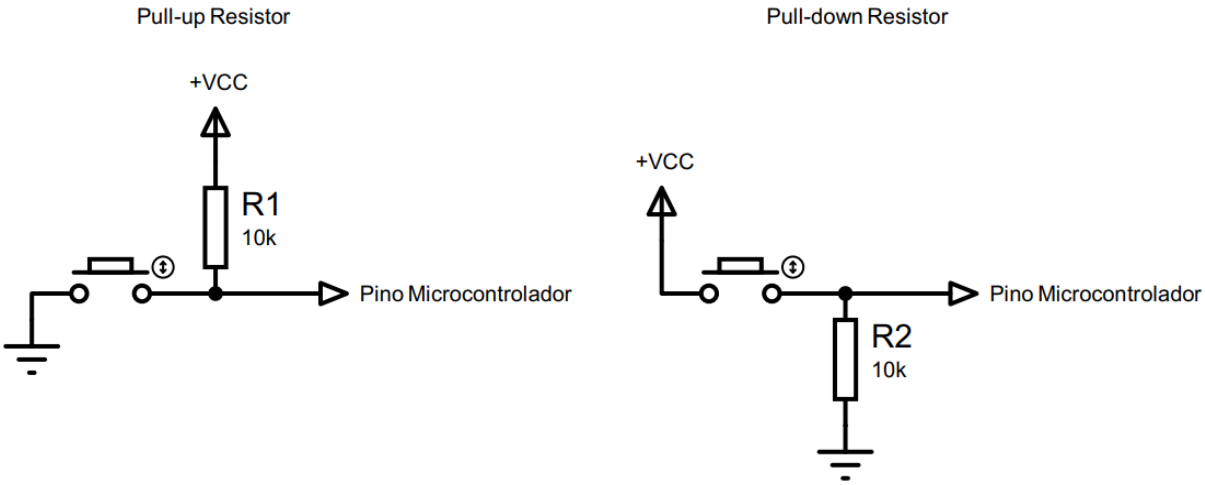
Como o botão táctil serve de meio de interação entre usuário e máquina, a máquina está constantemente “checando” se existe uma mudança no sinal de entrada em seus pinos de entrada digital, porém, devido ao *bouncing*, o microcontrolador pode registrar inúmeros sinais de ALTO e BAIXO, mesmo que o usuário tenha pressionado o botão uma única vez.

De modo a impedir que ocorra esses falsos pressionamentos é necessário fazer uso de alguma técnica de *debouncing* para atenuar esse efeito. O *debouncing* pode ser feito por meio de hardware ou software. Considerando que um dos objetivos do presente trabalho é o fator custo benefício, foi escolhido o método de *debouncing* por software, já que este método pode ser implementado no código do programa sem custo.

Debouncing - Software

Mesmo com o debouncing sendo implementado através de software se faz necessário incluir um resistor de pull-up ou de pull-down, mostrado na Figura 27, nos terminais do botão táctil para impedir que ruídos no pino de entrada do microcontrolador causem um falso disparo.

Figura 27 – Exemplo de botão táctil com resistores de pull-up e pull-down



Autoria Própria

Uma forma eficiente para leitura do estado de múltiplos botões em um projeto envolvendo microcontroladores é a criação de uma função que recebe como argumento o pino no qual determinado botão está conectado e retorna, na lógica booleana, um estado de True caso o botão foi pressionado ou False caso o botão não tenha sido pressionado.

A função booleana criada também ignora o período em que o bouncing ocorre ao realizar uma nova leitura após 10 ms. O código de exemplo dessa função pode ser encontrado no Apêndice E.

O firmware do microcontrolador ocupa um total de 17818 bytes de memória interna de um máximo de 30720 bytes, o que equivale a 58% do espaço total disponível. Ou seja, a adição de novas funcionalidades em versões futuras é uma possibilidade. O código completo do firmware do microcontrolador pode ser conferido no Apêndice F.

4 RESULTADOS

4.1 ANÁLISE DE CUSTO

Como um dos objetivos do presente trabalho é que o Controlador de Irrigação seja o mais acessível possível, o custo baixo foi um fator a se considerar durante a escolha dos componentes. Após uma pesquisa efetuada em novembro de 2021 em sites de venda de componentes eletrônicos, como mostrado na Tabela 9, foi calculado o valor aproximado para a produção do Controlador de Irrigação completo.

Tabela 9 – Custo Total do Controlador de Irrigação (novembro de 2021)

Módulo de Controle				
Quantidade	**Componente**	**Valor Unitário**	**Valor Total**	**Link**
1	Arduino Nano (5V)	R$ 57,00	R$ 57,00	https://www.baudaeletronica.com.br/placa-nano-r3.html
1	Display LCD 16x2	R$ 22,61	R$ 22,61	https://www.baudaeletronica.com.br/display-lcd-16x2-azul.html
1	Módulo I2C para LCD	R$ 10,00	R$ 10,00	https://www.baudaeletronica.com.br/modulo-adaptador-i2c-para-display-lcd.html
1	Módulo RTC DS1307	R$ 14,15	R$ 14,15	https://www.baudaeletronica.com.br/real-time-clock-rtc-ds1307-sem-bateria.html
1	Buzzer	R$ 2,17	R$ 2,17	https://www.baudaeletronica.com.br/buzzer-5v-sem-oscilador-interno.html
4	LEDs	R$ 0,24	R$ 0,96	https://www.baudaeletronica.com.br/led-difuso-5mm-azul.html
7	Push Button	R$ 0,24	R$ 1,68	https://www.baudaeletronica.com.br/chave-tactil-6x6x4-3mm-4-terminais.html
14	Resistores Diversos	R$ 0,10	R$ 1,40	https://www.baudaeletronica.com.br/resistor-1k1-1-1-4w.html
1	Capacitor Cerâmico 100nF	R$ 0,13	R$ 0,13	https://www.baudaeletronica.com.br/capacitor-ceramico-100nf-50v.html
1	Conector DC Jack P4	R$ 1,00	R$ 1,00	https://www.baudaeletronica.com.br/jack-p4-para-pcb-2-1-5x5mm.html
1	PCI	R$ 12,00	R$ 12,00	https://jlcpcb.com/
Custo Total				**R$ 123,10**

Módulo de Acionamento				
Quantidade	**Componente**	**Valor Unitário**	**Valor Total**	**Link**
1	CI 74HC595	R$ 2,18	R$ 2,18	https://www.baudaeletronica.com.br/circuito-integrado-74hc595-shift-register.html
5	Relés 5V	R$ 4,04	R$ 20,20	https://www.baudaeletronica.com.br/rele-5v-2-posicoes-250v-10a.html
5	Transistores npn 2n2222a	R$ 0,37	R$ 1,85	https://www.baudaeletronica.com.br/transistor-npn-2n2222.html
5	Diodos 1N4007	R$ 0,24	R$ 1,20	https://www.baudaeletronica.com.br/diodo-1n4007.html
5	LEDs	R$ 0,24	R$ 1,20	https://www.baudaeletronica.com.br/led-difuso-5mm-azul.html
7	Conectores de 2 vias	R$ 1,39	R$ 9,73	https://www.baudaeletronica.com.br/borne-2-polos-kf-301-2t.html
1	PCI	R$ 10,00	R$ 10,00	https://jlcpcb.com/
Custo Total				**R$ 46,36**

Custo Total do Controlador de Irrigação	**R$ 169,46**

Autoria Própria

De modo a verificar se o preço final do controlador de irrigação atingiu o seu objetivo de possuir um preço baixo, foi realizada uma pesquisa de preço em novembro de 2021 de modo a determinar o preço médio de um controlador de irrigação com funcionalidades parecidas com o desenvolvido. A Tabela 10 apresenta os resultados.

Tabela 10 – Pesquisa de preço de controladores comerciais (novembro de 2021)

Controladores Comerciais			
Controlador	**Marca**	**Preço**	**Link**
Controlador Rain Bird Esp Rzx-e 4 Estações	Rain Bird	R$ 622,00	https://www.mercadolivre.com.br/
Controlador de Irrigação 4 estação Kenntech AquaQuattro	AquaQuattro	R$ 636,90	https://www.americanas.com.br/
Controlador Hunter X Core 401 Interno 4 Setores	Hunter	R$ 650,00	https://www.paiolverde.com.br/
Programador Rps46 - 6 Zonas	KRain	R$ 559,00	https://www.americanas.com.br/
Média de Preço			**R$ 629,45**

Autoria Própria

Como apresentado pela Tabela 9, o custo total para esse controlador é de aproximadamente 170 reais. Preço este, significativamente baixo quando se comparado aos concorrentes comerciais, que apresentam, em média, um custo de 630 reais. Com isso, o Controlador de Irrigação desenvolvido neste trabalho apresenta um custo 73% menor do que a média da concorrência sem sacrificar as principais funcionalidades esperadas de um controlador de irrigação profissional.

É importante ressaltar que esse custo reduzido se deve ao fato do controlador desenvolvido ser uma solução DIY, logo o custo dos impostos sobre produtos industrializados, mão de obra e outros fatores não estão sendo considerados.

4.2 FUNCIONALIDADES DO CONTROLADOR

O Controlador de Irrigação desenvolvido conta com todas as funcionalidades básicas presentes nos controladores comerciais são elas:

- Configuração de múltiplas rotinas de irrigação com base nos dias e horários especificados pelo usuário;
- Tempos de irrigação ajustados com incrementos de +/- 10 minutos;
- Irrigação manual, no qual o usuário pode especificar um tempo especifico em que a zona será irrigada;
- Armazenamento da data e hora do controlador, mesmo em casos de perda de energia, com auxílio do módulo RTC (caso o usuário adicione uma bateria no módulo);
- Display de LCD para facilitar interação entre usuário e controlador.

4.3 VANTAGENS EM RELAÇÃO AOS CONTROLADORES COMERCIAIS

O Controlador de Irrigação desenvolvido possui o diferencial de apresentar uma série de vantagens e funcionalidades que não são encontrados em alternativas comerciais de maior custo. Essas funcionalidades foram sendo introduzidas durante as diversas iterações do sistema durante a fase de desenvolvimento de modo a fornecer ao usuário final um produto modular, de fácil uso e adaptável as necessidades do indivíduo. Essas vantagens são listadas a baixo:

- **Sistema Open-Source e Open Hardware**
    - Todo o hardware e software desenvolvidos para o controlador são disponibilizados sem custo ou licença a todos aqueles que desejam utiliza-lo, modifica-lo e ou revende-lo baseado no design original.
- **Maior número de zonas de irrigação**
    - O Controlador básico desenvolvido apresenta um total de 5 zonas de irrigação, em contrapartida, os controladores comerciais básicos apresentam de 1 a 4 zonas de irrigação, enquanto os modelos comerciais de maior valor podem apresentar de 6 a 8 zonas.
- **Modularidade**
    - O design do hardware do Controlador de Irrigação desenvolvido foi feito com foco em modularidade, ao ser construído em duas PCBs, uma para o Módulo de Controle e uma para o Módulo de Irrigação, o usuário tem a possibilidade de trocar o Módulo de Acionamento por outro, caso haja falha ou defeito no mesmo, sem a necessidade de trocar o Módulo de Controle, além de possibilitar ao mesmo o desenvolvimento de Módulos de Acionamento customizados para fins específicos. De forma análoga o usuário pode trocar o Módulo de Controle por outro e utilizar o mesmo Módulo de Acionamento.
- **Zonas de Irrigação Expansíveis**
    - Controladores de Irrigação comerciais possuem um número fixo de zonas de irrigação, ou seja, caso o usuário necessite de mais zonas o mesmo irá precisar comprar outros controladores, o que irá gerar um custo ainda maior. O Módulo de Acionamento, por ser baseado no circuito integrado 74HC595, apresenta a funcionalidade de poder ser

conectado em cascata a outros Módulos de Acionamento de modo a expandir o número de zonas sem a necessidade de comprar outro controlador.

- **Baseado na plataforma Arduino**
    - 100% do controlador desenvolvido é baseado na plataforma de desenvolvimento Arduino, que apresenta vasta documentação na internet e fácil uso, isso possibilita usuários que tenham certo conhecimento em programação em ajustar o firmware do controlador de acordo com seus requisitos, caso o firmware básico não seja suficiente ou para adicionar novas funcionalidades.
- **Comunicação Serial**
    - O Módulo de Controle apresenta uma conexão disponível ao usuário para comunicação serial (pinos RX e TX do Arduino Nano) entre o controlador e outros periféricos que utilizem o mesmo protocolo de comunicação. Essa comunicação pode ser útil a usuários mais avançados como uma ferramenta para debugging ou para possibilitar a adição de novas funcionalidades no controlador.
- **Comunicação I2C**
    - O Módulo de Controle apresenta uma conexão disponível ao usuário para comunicação I2C (pinos A4 e A5 do Arduino Nano). O controlador faz uso de dois componentes, RTC e LCD, que fazem uso da comunicação I2C, porém outros componentes que aceitem esse protocolo podem ser adicionados pelo usuário caso desejado.
- **Sensores**
    - O Módulo de Controle apresenta uma conexão disponível ao usuário para a adição de um sensor genérico que faça uso de um pino digital (pino D2 do Arduino Nano). O pino D2 foi escolhido por também apresentar a funcionalidade de ser usado como uma interrupção;
    - O Módulo de Controle também conta com uma conexão para o pino analógico A1 do Arduino Nano, possibilitando ao usuário a utilização de um sensor, por exemplo um sensor de umidade básico.

De forma geral, o Controlador de Irrigação foi desenvolvido com o usuário em mente, de modo que o mesmo fique livre para realizar modificações no firmware do mesmo e adicionar novas funcionalidades para se adequar as suas necessidades. Essas conexões diretas ao microcontrolador não são encontradas em controladores comerciais, mas foram adicionadas para tornar o controlador o mais versátil possível ao usuário.

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÕES

Ao longo do desenvolvimento do trabalho diversos protótipos foram criados para auxiliar na versão final do Controlador de Irrigação. Esse período de prototipagem se mostrou significativamente útil no desenvolvimento, já que diversas funções e componentes foram sendo alterados durante o processo.

Após finalizar as placas de circuito impresso de ambos os módulos, foi verificado, pelas dimensões físicas, que o controlador desenvolvido apresenta tamanho similar aos concorrentes, assim como uma interface com o usuário similar.

Através da análise de custo e pesquisa de mercado foi verificado que o objetivo do controlador de irrigação de apresentar um bom custo benefício ao usuário final foi atingido, ainda mais se os componentes forem adquiridos em sites de e-commerce chines.

Com isso, o Controlador de Irrigação desenvolvido é de fato uma boa alternativa aos controladores comerciais, tanto em custo como em funcionalidades. O fato de o mesmo apresentar um valor de custo 73% menor do que os comerciais, possibilita o acesso deste tipo de tecnologia à um maior público.

5.2 TRABALHOS FUTUROS

O sistema de irrigação desenvolvido poderia ser aprimorado ao se substituir o microcontrolador Arduino Nano por um ESP32 da Espressif Systems, dessa maneira adicionando a funcionalidade de Wi-Fi e Bluetooth no projeto, permitindo a conexão do controlador na rede ou no celular do usuário, além de que o mesmo possui uma memória interna maior, possibilitando que códigos mais complexos possam ser desenvolvidos para o controlador.

Alguns tipos de sensores poderiam ser incluídos no controlador. Um sensor de umidade pode verificar a umidade da terra e com base nessa leitura o controlador realiza a rega caso o solo atinja um nível de seca elevado. Um sensor de vazão

também poderia ser incluído como forma de verificar a vazão da água pelas tubulações e caso a vazão seja muito baixa o controlador desligaria o sistema e reportaria como um problema ao usuário, indicando possíveis vazamentos.

REFERÊNCIAS

BANZI, Massimo. **Getting Started with Arduino**. 2ª Edição. Estados Unidos da America: O'Reilly, 2011. 118 páginas.

DHATRI P V S, Divya; Pachiyannan, M; Swaroopa Rani, J.K.; Pravallika, G. A Low-Cost Arduino based Automatic Irrigation System using Soil Moisture Sensor: Design and Analysis. **2019 2nd International Conference on Signal Processing and Communication (ICSPC-2019)**, Coimbatore, INDIA, 978-1-7281-1849-9, março 2019.

KOPRDA, Stefan; BALOGH, Zoltan; HRUBÝ, Dusan; TURCÁNI, Milan. Proposal of the irrigation system using low-cost Arduino system as part of a smart home. **SISY 2015 • IEEE 13th International Symposium on Intelligent Systems and Informatics**, Subotica, Serbia, 978-1-4673-9388-1, setembro 2015.

MANTOVANI, Everardo Chartuni; BERNARDO, Salassier; PALARETTI, Luiz Fabiano. **Irrigação Princípios e Métodos.** 3ª Edição. Brasil: Editora UFV, 2009. 355 páginas.

MENDONÇA, Hélio Sousa. **SPI e I2C**. Disponível em: <https://paginas.fe.up.pt/~hsm/docencia/comp/spi-e-i2c/>. Acesso em: 12 jul. 2021.

MONK, Simon. **Programming Arduino**: Getting Started with Sketches. 1ª Edição. Estados Unidos da America: The McGraw-Hill Companies, 2012. 162 páginas.

SCHERZ, Paul; MONK, Simon. **Practical Electronics for Inventors**. 3ª Edição. Estados Unidos da America: The McGraw-Hill Companies, 2013. 1014 páginas.

APÊNDICE A – Microcontrolador ATmega328p

Especificações - Microcontrolador ATmega328p

- Microcontrolador AVR de 28 pinos;
- Memória Flash de Programa: 32 kbytes;
- Memória EEPROM: 1 kbytes;
- Memória SRAM: 2 kbytes;
- Pinos de I/O: 23;
- Timers: Dois de 8-bits / Um de 16-bits;
- Conversor A/D: 10-bit Seis Canais;
- PWM: Seis Canais;
- RTC: Sim, com um oscilador externo;
- MSSP: SPI e I^2C;
- USART: Sim;
- Oscilador Externo: Até 20MHz.

APÊNDICE B – Arduino Nano

Especificações - Arduino Nano

- Microcontrolador: ATmega328p;
- Arquitetura: AVR;
- Tensão de operação: 5 V;
- Memória Flash: 32 KB no qual 2 KB são usados pelo bootloader;
- Memória SRAM: 2 KB;
- Velocidade de Clock: 16 MHz;
- Pinos de entrada analógica: 8;
- Memória EEPROM: 1 KB;
- Corrente contínua por pino de I/O: 40 mA;
- Tensão de entrada (Vin): 7 a 12 V;
- Pinos de I/O Digital: 22 (6 suportam PWM);
- Pinos capazes de gerar sinais em PWM: 6;
- Consumo de Energia: 19 mA;
- Tamanho da PCB: 18 x 45 mm;
- Peso: 7 g.

APÊNDICE C – Real Time Clock (DS1307)

Especificações DS1307

- O relógio em tempo real (RTC) conta segundos, minutos, Horas, Data do Mês, Mês, Dia do Semana e ano com ano bissexto Compensação válida até 2100
- Interface I2C;
- Tensão de Operação: 5V;
- Menos que 500nA de corrente quando operando com uma bateria;
- Memória SVRAM: 56bytes;
- Modos de operação: Fonte externa ou bateria;
- Programmable square wave output pin;
- Pino de saída de onda quadrada programável.

APÊNDICE D – Registrador de Deslocamento – 74HC595

Especificações – 74HC595

- Entrada serial de 8-bit;
- Saída serial ou paralela de 8-bit;
- Registrador de armazenamento com saídas de 3 estados;
- Registrador de deslocamento com liberação direta (direct clear);
- frequência de deslocamento de 100 MHz (típico);
- Proteção ESD:
- HBM JESD22-A114F excede 2.000 V;
- MM JESD22-A115-A excede 200 V;
- Especificado de -40C a +85C e de -40C a +125C.

APÊNDICE E – Exemplo de Código para Debouncing

```
#define button 8                                  // Botão conectado ao pino D8
int buttonPress = 0;
int count = 0;

void setup() {
  pinMode(button, OUTPUT);
  Serial.begin(9600);
}

void loop() {
   buttonPress = debounceButtonPress(button);
  if(buttonPress == 1)
  {
    count ++;
    Serial.print("BANG! - Contagem: ");
    Serial.println(count);
    delay(150);
    }
  }
boolean debounceButtonPress(int buttonPin)   // Função de debounce
{
  boolean oldState = 0;
  boolean state = digitalRead(buttonPin);
  if(oldState != state)
  {
    delay(10);                                 //Atraso de 10ms para não registrar o bouncing
    state = digitalRead(buttonPin);            //Nova leitura do estado do botão
  }
  return state;
}
```

APÊNDICE F – Firmware do Controlador de Irrigação

Observação: O código a seguir foi desenvolvido com o protótipo do controlador em mente e suas respectivas conexões, para utiliza-lo com o Módulo de Controle é necessário atualizar o valor dos pinos do Arduino Nano de acordo com as Tabela 6 e 7 de conexões entre microcontrolador e componentes na seção “Arduino Pinout Conections” do código.

```
// ---------- Header ----------
// Irrigation Control Program
// Author: Eduardo Roston
// Institution UFSCar
// Date v1.0: 17 of november 2021
// Date v1.1: 19 of november 2021
// Date v1.2: 21 of november 2021
// Date v1.3: 22 of november 2021
// Date v1.4: 23 of november 2021
// Date v1.5: 24 of november 2021
// Date v1.6: 25 of november 2021
// Date v1.7: 26 of november 2021
// Date v1.8: 26 of november 2021
// Date v1.9: 26 of november 2021
// This is an Open Source program.

//----------------------------------------------------------------

//---------- Libraries ----------
#include<Wire.h>
#include<LiquidCrystal_I2C.h>   //Biblioteca do LCD + Módulo I2C
#include "RTClib.h"             //Biblioteca Módulo RTC

//---------- General Global Variables ----------
const int z1 = 2;
const int z2 = 4;
const int z3 = 8;
const int z4 = 16;
const int z5 = 32;
int flag_butA = 1;
int flag_butB = 1;
int horaButA = 0;
int minutoButA = 0;
int horaButB = 0;
int minutoButB = 0;

//---------- checkZoneAlarm Function Global Variables ----------
int zone = 0;
int diaZone = 0;
int horaInicial = 0;
int minutoInicial = 0;
int horaFinal = 0;
int minutoFinal = 0;
int flagZone = 0;
```

```
// Set the timer period in minutes for butA and butB
int setTimerA = 30;
int setTimerB = 60;

//---------- Buzzer Global Variables ----------
int freq = 3000;
int enable = 0; // Enable (1) or disable (0) buzzer beep

//---------- RTC Global Variables ----------
int dia = 1;
int mes = 1;
int ano = 2021;
int hora = 0;
int minuto = 0;
int segundo = 0;

//---------- Menu Global Variables ----------
int menu = 0;
int zoneNumber[] = {0,0,0,0,0};
int auxZoneNum = 1;
int enableZoneNumber[] = {0,0,0,0,0};

//char *zoneDays[] = {"S T Q Q S S D", "S Q S", "S D"};
char *zoneDay[] = {"Domingo", "Segunda", "Terca", "Quarta","Quinta", "Sexta", "Sabado"};
int auxDay = 0;
int auxHour = 0;
int auxMinute = 0;
int auxPeriod = 0;

//----- Debugging -----
int valor = 0;

//---------- Arduino Pinout Conections ----------
#define butStop 2    //Connected to Button Stop
#define greenLed 3   //Connected to green LED
#define but3 4       //Connected to Button 3
#define butB 5       //Connected to Button B
#define butA 6       //Connected to Button A
#define butDown 7    //Connected to Button Down
#define butReturn 8  //Connected to Button Return
#define buzzer 9     //Connected to Buzzer
#define blueLed A0   //Connected to blue LED
#define redLed A1    //Connected to red LED
#define butUp A6     //Connected to Button Up
#define butEnter A7  //Connected to Button Enter
#define enablePin 10 //Connected to output enable, pin 13 on 74HC595
#define dataPin 11   //Connected to DS, pin 14 on 74HC595
#define clockPin 12  //Connected to SH_CP, pin 11 on 74HC595
#define latchPin 13  //Connected to ST_CP, pin 12 on 74HC595
#define clearPin A2  //Connected to Master Reset, pin 10 on 74HC595
```

```
//---------- Initialization ----------

LiquidCrystal_I2C lcd (0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
 // Define o LCD assim como o endereço I2C do Módulo, //no caso 0x27

RTC_DS1307 rtc;

char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"};

void setup() {

//---------- pinMode Setup ----------

//----- Push Buttons -----
  pinMode(butUp, INPUT);
  pinMode(butDown, INPUT);
  pinMode(butReturn, INPUT);
  pinMode(butEnter, INPUT);
  pinMode(butA, INPUT);
  pinMode(butB, INPUT);
  pinMode(but3, INPUT);
  pinMode(butStop, INPUT);

//----- LEDs -----
  pinMode(greenLed, OUTPUT);
  pinMode(blueLed, OUTPUT);

//----- Buzzer -----
  pinMode(buzzer, OUTPUT);

//----- 74HC595 -----
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(enablePin, OUTPUT);
  pinMode(clearPin, OUTPUT);

//----- RTC DS1307 -----

// Chcks if RTC Module is connected
if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }
// Checks if RTC Module has lost power
  if (!rtc.isrunning()) {
    Serial.println("RTC lost power, lets set the time!");


// ---------- RTC Initial Setup DATE&TIME ----------
// Following line sets the RTC with an explicit date & time
```

```
// for example to set January 27 2017 at 12:56 you would call:
// rtc.adjust(DateTime(2017, 1, 27, 12, 56, 0));
// Will adjust to this preset time and date only if RTC looses power
    int day_ = 12;
    int month_ = 9;
    int year_ = 2021;
    int hour_ = 22;
    int minute_ = 0;
    int seconds_ = 0;
    rtc.adjust(DateTime(year_, month_, day_, hour_, minute_, seconds_));
  }

// ---------- LCD Start Setup ----------

  lcd.begin(16, 2); //Inicializa o LCD
  lcd.clear();  //Limpa a tela

// ---------- Serial Communication Start Setup ----------

  Serial.begin(9600); //Opens the serial COM and sets bound rate to 9600 bps

// ---------- Shift Register Start Setup ----------
//Ao inicializar o programa, todas as zonas são desligadas
  digitalWrite(enablePin, HIGH);
  digitalWrite(clearPin, LOW);

// ---------- Interrupt Start Setup ----------
attachInterrupt(digitalPinToInterrupt(butStop), disableZones, HIGH);

// -------------------- Main Setup --------------------

// Disable all zones on startup

disableZones();

// ---------- Initializing System ----------
  lcd.print("Iniciando");
  lcd.setCursor(9,0);
  lcd.print(".");
  delay(200);
  lcd.print(".");
  delay(200);
  lcd.print(".");
  delay(200);
  lcd.print(".");
  delay(200);
  lcd.print(".");
  delay(200);
  lcd.print(".");
  delay(200);
  lcd.print("OK");
  delay(1500);
  lcd.clear();
```

```
} // End of Void Setup


// --------------------- void loop() ---------------------

void loop() {

// Update Menu Screens

updateMenu();

// Check if any Irrigation routines are enabled, if so,
//it will enable and disable the corresponding irrigation zone

checkZoneAlarm(zone, diaZone, horaInicial, minutoInicial, horaFinal, minutoFinal, flagZone);
DateTime now = rtc.now();
Serial.print("now.day() = ");
Serial.println(now.day());
Serial.print("now.dayOfTheWeek() = ");
Serial.println(now.dayOfTheWeek());


// ---------- Menu Selection ----------
  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    lcd.clear();
    menu++;
    updateMenu();
    delay(100);
}

  if(debounceButtonPress(butDown)){
    lcd.clear();
    menu--;
    updateMenu();
    delay(100);
}


// ---------- Configuração Botão A ----------
  if(debounceButtonPress(butA)){
    lcd.clear();
    lcd.home();
    lcd.print(" Ativar Zona 1");
    lcd.setCursor(0,1);
    lcd.print("por 30 minutos ?");

    while(flag_butA == 1){
    if(debounceButtonPress(butReturn)){
      flag_butA = 0;
      lcd.clear();
      break;
```

```
}
if(analogRead(butEnter) > 1000){
  buzzerPress(freq,enable);
  digitalWrite(blueLed, HIGH);
  lcd.clear();
  lcd.home();
  lcd.print(" Zona 1 Ativada");
  activateZone(2);
  lcd.setCursor(0,1);
  lcd.print("I>");
  DateTime now = rtc.now();
  horaButA = now.hour();
  minutoButB = now.minute();

  lcd.setCursor(2,1);
  if(now.hour() < 10){lcd.print("0");}
  lcd.print(now.hour(), DEC);
  lcd.print(":");
  lcd.setCursor(5,1);
  if(now.minute() < 10){lcd.print("0");}
  lcd.print(now.minute(), DEC);
  lcd.setCursor(9,1);
  lcd.print("T>");

  int minutoTimer = 0;
  int segundoTimer = 0;

  // Add Time Period

  while(flag_butA == 1){
    // 30 minute Timer on Display
         int j = 0;
         int k = 1;
    for(int i = 0; i < (setTimerA*60); i++){

      segundoTimer = 59 - j;
      minutoTimer = setTimerA - k;
      j++;

      if(j == 60){
        j = 0;
        k++;
      }

      if(k > setTimerA){
        k = setTimerA;
      }

      lcd.setCursor(11,1);
      if(minutoTimer < 10){
        lcd.setCursor(11,1);
        lcd.print("0");
```

```
          lcd.setCursor(12,1);
         }

         lcd.print(minutoTimer);
         lcd.setCursor(13,1);
         lcd.print(":");
         if(segundoTimer < 10){
          lcd.setCursor(14,1);
          lcd.print("0");
         }
         lcd.print(segundoTimer);
         delay(1000);

         if(debounceButtonPress(butReturn)){
         flag_butA = 0;
         disableZones();
         digitalWrite(blueLed, LOW);
         break;
        }
        } // End for();

        disableZones();
        flag_butA = 0;
        break;

        if(debounceButtonPress(butReturn)){
         disableZones();
         flag_butA = 0;
         digitalWrite(blueLed, LOW);
         break;
        }
      } //End while()2
    }
    } // End while()1

    flag_butA = 1;
    menu = 1;
    lcd.clear();
} // End But A Menu


// ---------- Configuração Botão B ----------

  if(debounceButtonPress(butB)){
    lcd.clear();
    lcd.home();
    lcd.print(" Ativar Zona 1");
    lcd.setCursor(0,1);
    lcd.print("por 60 minutos ?");

    while(flag_butB == 1){
    if(debounceButtonPress(butReturn)){
```

```
  flag_butB = 0;
  lcd.clear();
  break;
 }
 if(analogRead(butEnter) > 1000){
  buzzerPress(freq,enable);
  digitalWrite(blueLed, HIGH);
  lcd.clear();
  lcd.home();
  lcd.print(" Zona 1 Ativada");
  activateZone(2);
  lcd.setCursor(0,1);
  lcd.print("I>");
  DateTime now = rtc.now();
  horaButA = now.hour();
  minutoButB = now.minute();

  lcd.setCursor(2,1);
  if(now.hour() < 10){lcd.print("0");}
  lcd.print(now.hour(), DEC);
  lcd.print(":");
  lcd.setCursor(5,1);
  if(now.minute() < 10){lcd.print("0");}
  lcd.print(now.minute(), DEC);
  lcd.setCursor(9,1);
  lcd.print("T>");

  int minutoTimer = 0;
  int segundoTimer = 0;

  // Add Time Period

  while(flag_butB == 1){
    // 30 minute Timer on Display
         int j = 0;
         int k = 1;
    for(int i = 0; i < (setTimerB*60); i++){

      segundoTimer = 59 - j;
      minutoTimer = setTimerB - k;
      j++;

      if(j == 60){
        j = 0;
        k++;
      }

      if(k > setTimerB){
        k = setTimerB;
      }

      lcd.setCursor(11,1);
```

```
          if(minutoTimer < 10){
            lcd.setCursor(11,1);
            lcd.print("0");
            lcd.setCursor(12,1);
          }

          lcd.print(minutoTimer);
          lcd.setCursor(13,1);
          lcd.print(":");
          if(segundoTimer < 10){
            lcd.setCursor(14,1);
            lcd.print("0");
          }
          lcd.print(segundoTimer);
          delay(1000);

          if(debounceButtonPress(butReturn)){
          flag_butB = 0;
          disableZones();
          digitalWrite(blueLed, LOW);
          break;
        }
        } // End for();
        disableZones();
        flag_butB = 0;
        break;

        if(debounceButtonPress(butReturn)){
          disableZones();
          flag_butB = 0;
          digitalWrite(blueLed, LOW);
          break;
        }
      } //End while()2
    }
    } // End while()1

    flag_butB = 1;
    menu = 1;
    lcd.clear();
} // End But B Menu



//----- Menu 1 - Easter Egg (System Firmware Information) -----
  if(analogRead(butEnter) > 1000 && menu == 1){
    buzzerPress(freq,enable);
    lcd.clear();
    lcd.home();
    lcd.print("  Firmware v1.9");
    lcd.setCursor(0,1);
    lcd.print("1508 LinesOfCode");
```

```
    delay(4000);
    lcd.clear();
    lcd.home();
  }



//----- Menu 2 - Configurar Rotina -----
  if(analogRead(butEnter) > 1000 && menu == 2){
    digitalWrite(redLed, HIGH);
    buzzerPress(freq,enable);
    delay(200);

    // ----- Set Zone -----
    lcd.clear();
    lcd.home();
    lcd.print("Selecionar  Zona");
    lcd.setCursor(0,1);
    lcd.print("     Zona ");
    int zone = 1;
    lcd.setCursor(10,1);
    lcd.print(zone);

    while(analogRead(butEnter) < 1000){

      if(analogRead(butUp) > 1000){
        buzzerPress(freq,enable);
        zone++;
        if(zone > 5){ zone = 5;}
        lcd.setCursor(10,1);
        lcd.print(zone);
        delay(200);
      }

      if(debounceButtonPress(butDown) == HIGH){
        zone--;
        if(zone < 1){ zone = 1;}
        lcd.setCursor(10,1);
        lcd.print(zone);
        delay(200);
      }

    } // End while()1

    lcd.clear();
    lcd.home();
    lcd.print("Ativar Zona ");
    lcd.print(zone);
    lcd.print(" ?");
    lcd.setCursor(0,1);
    lcd.print("  Press Enter");
  //  int flagZone = 0;
```

```
int flagMenu = 1;
int flagExit = 1;
delay(200);
while(flagMenu){
  if(analogRead(butEnter) > 1000){
    buzzerPress(freq,enable);
    flagZone = 1;
    flagMenu = 0;
    delay(200);
}
  if(debounceButtonPress(butReturn) == HIGH){

    lcd.clear();
    lcd.home();
    lcd.print("     Zona ");
    lcd.print(zone);
    lcd.setCursor(0,1);
    lcd.print("  Desabilitada");
    delay(5000);
    digitalWrite(redLed, LOW);
    flagZone = 0;
    flagMenu = 0;
    flagExit = 0;
    break;
  }
} // End while()2

    int flagMenu20 = 1;
    lcd.clear();
    lcd.home();
    lcd.print("Zona ");
    lcd.print(zone);
    lcd.print("  Ativada!");
    lcd.setCursor(0,1);
    lcd.print("  Press Enter");

while(flagMenu20 && flagExit){
  if(analogRead(butEnter) > 1000){
    buzzerPress(freq,enable);
    flagMenu20 = 0;
    delay(200);
  }

  if(debounceButtonPress(butReturn) == HIGH){
    digitalWrite(redLed, LOW);
    break;
  }
} // End while()2


    // ---------- Set Day ----------
    int flagMenu21 = 1;
```

```
    lcd.clear();
    lcd.home();
    lcd.print(" Configurar dia");
    lcd.setCursor(0,1);
    lcd.print("    >");
    lcd.print(zoneDay[diaZone]);

 while(flagMenu21 && flagExit){

  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    diaZone++;
    if(diaZone > 6){ diaZone = 6;}
    lcd.clear();
    lcd.home();
    lcd.print(" Configurar dia");
    lcd.setCursor(0,1);
    lcd.print("    >");
    lcd.print(zoneDay[diaZone]);
    delay(200);
  }

  if(debounceButtonPress(butDown) == HIGH){
    diaZone--;
    if(diaZone < 0){ diaZone = 0;}
    lcd.clear();
    lcd.home();
    lcd.print(" Configurar dia");
    lcd.setCursor(0,1);
    lcd.print("    >");
    lcd.print(zoneDay[diaZone]);
    delay(200);
  }

  if(analogRead(butEnter) > 1000){
    buzzerPress(freq,enable);
    flagMenu21 = 0;
    delay(200);
  }
 } // End while()3

  // ----- Set horario de inicio - Horas -----
  int flagMenu22 = 1;
 // int horaInicial = 0;

  lcd.clear();
  lcd.home();
  lcd.print("Horario Inicio");
  lcd.setCursor(0,1);
  lcd.print("Hora >");
  if(horaInicial < 10){lcd.print("0");}
```

```
 lcd.print(horaInicial);

 while(flagMenu22 && flagExit){

  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    horaInicial++;
  if(horaInicial > 23){ horaInicial = 23;}
    lcd.clear();
    lcd.home();
    lcd.print("Horario Inicio");
    lcd.setCursor(0,1);
    lcd.print("Hora >");
    if(horaInicial < 10){lcd.print("0");}
    lcd.print(horaInicial);
    delay(200);
}

  if(debounceButtonPress(butDown) == HIGH){
    horaInicial--;
  if(horaInicial < 0){ horaInicial = 0;}
    lcd.clear();
    lcd.home();
    lcd.print("Horario Inicio");
    lcd.setCursor(0,1);
    lcd.print("Hora >");
    if(horaInicial < 10){lcd.print("0");}
    lcd.print(horaInicial);
    delay(200);
  }

  if(analogRead(butEnter) > 1000){
    buzzerPress(freq,enable);
    flagMenu22 = 0;
    delay(200);
  }

 } // End while()4

 // ----- Set horario de inicio - Minutos -----
 int flagMenu23 = 1;
//int minutoInicial = 0;

 lcd.clear();
 lcd.home();
 lcd.print("Horario Inicio");
 lcd.setCursor(0,1);
 lcd.print("Minuto >");
 if(minutoInicial < 10){lcd.print("0");}
 lcd.print(minutoInicial);

 while(flagMenu23 && flagExit){
```

```
  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    minutoInicial++;
  if(minutoInicial > 59){ minutoInicial = 59;}
    lcd.clear();
    lcd.home();
    lcd.print("Horario Inicio");
    lcd.setCursor(0,1);
    lcd.print("Minuto >");
    if(minutoInicial < 10){lcd.print("0");}
    lcd.print(minutoInicial);
    delay(200);
 }

  if(debounceButtonPress(butDown) == HIGH){
    minutoInicial--;
  if(minutoInicial < 0){ minutoInicial = 0;}
    lcd.clear();
    lcd.home();
    lcd.print("Horario Inicio");
    lcd.setCursor(0,1);
    lcd.print("Minuto >");
    if(minutoInicial < 10){lcd.print("0");}
    lcd.print(minutoInicial);
    delay(200);
  }

  if(analogRead(butEnter) > 1000){
    buzzerPress(freq,enable);
    flagMenu23 = 0;
    delay(200);
  }

 } // End while()5

 // ----- Set horario de Termino - Horas -----
 int flagMenu24 = 1;
// int horaFinal = 0;

 lcd.clear();
 lcd.home();
 lcd.print("Horario Termino");
 lcd.setCursor(0,1);
 lcd.print("Hora >");
 if(horaFinal < 10){lcd.print("0");}
 lcd.print(horaFinal);

 while(flagMenu24 && flagExit){

  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
```

```
    horaFinal++;
  if(horaFinal > 23){ horaFinal = 23;}
    lcd.clear();
    lcd.home();
    lcd.print("Horario Termino");
    lcd.setCursor(0,1);
    lcd.print("Hora >");
    if(horaFinal < 10){lcd.print("0");}
    lcd.print(horaFinal);
    delay(200);
}

  if(debounceButtonPress(butDown) == HIGH){
    horaFinal--;
  if(horaFinal < 0){ horaFinal = 0;}
    lcd.clear();
    lcd.home();
    lcd.print("Horario Termino");
    lcd.setCursor(0,1);
    lcd.print("Hora >");
    if(horaFinal < 10){lcd.print("0");}
    lcd.print(horaFinal);
    delay(200);
  }

  if(analogRead(butEnter) > 1000){
    buzzerPress(freq,enable);
    flagMenu24 = 0;
    delay(200);
  }

 } // End while()6

 // ----- Set horario de inicio - Minutos -----
 int flagMenu25 = 1;
// int minutoFinal = 0;

 lcd.clear();
 lcd.home();
 lcd.print("Horario Termino");
 lcd.setCursor(0,1);
 lcd.print("Minuto >");
 if(minutoFinal < 10){lcd.print("0");}
 lcd.print(minutoFinal);

 while(flagMenu25 && flagExit){

  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    minutoFinal++;
  if(minutoFinal > 59){ minutoFinal = 59;}
    lcd.clear();
```

```
    lcd.home();
    lcd.print("Horario Termino");
    lcd.setCursor(0,1);
    lcd.print("Minuto >");
    if(minutoFinal < 10){lcd.print("0");}
    lcd.print(minutoFinal);
    delay(200);
}

  if(debounceButtonPress(butDown) == HIGH){
    minutoFinal--;
  if(minutoFinal < 0){ minutoFinal = 0;}
    lcd.clear();
    lcd.home();
    lcd.print("Horario Termino");
    lcd.setCursor(0,1);
    lcd.print("Minuto >");
    if(minutoFinal < 10){lcd.print("0");}
    lcd.print(minutoFinal);
    delay(200);
  }

  if(analogRead(butEnter) > 1000){
    buzzerPress(freq,enable);
    flagMenu25 = 0;
    delay(200);
  }

} // End while()7

// ----- Resumo da Configuração -----

// Show Initial and End Time
lcd.clear();
lcd.home();
lcd.print("Inicio:    ");
if(horaInicial < 10){lcd.print("0");}
lcd.print(horaInicial);
lcd.print(":");
if(minutoInicial < 10){lcd.print("0");}
lcd.print(minutoInicial);
lcd.setCursor(0,1);
lcd.print("Fim:       ");
if(horaFinal < 10){lcd.print("0");}
lcd.print(horaFinal);
lcd.print(":");
if(minutoFinal < 10){lcd.print("0");}
lcd.print(minutoFinal);
if(flagExit == 1){
delay(4000);
digitalWrite(redLed, LOW);
lcd.clear();
```

```
      } else{
        lcd.clear();
        digitalWrite(redLed, LOW);
      }
      menu = 1;
} // End Menu 2



//----- Menu 3 - Irrigação Manual -----
  if(analogRead(butEnter) > 1000 && menu == 3){
    digitalWrite(redLed, HIGH);
    buzzerPress(freq,enable);

    lcd.clear();
    lcd.home();
    lcd.print("Selecionar  Zona");
    lcd.setCursor(0,1);
    lcd.print("     Zona ");
    int zone = 1;
    lcd.setCursor(10,1);
    lcd.print(zone);
    delay(200);

    while(analogRead(butEnter) < 1000){

      if(analogRead(butUp) > 1000){
        buzzerPress(freq,enable);
        zone++;
        if(zone > 5){ zone = 5;}
        lcd.setCursor(10,1);
        lcd.print(zone);
        delay(200);
      }

      if(debounceButtonPress(butDown) == HIGH){
        zone--;
        if(zone < 1){ zone = 1;}
        lcd.setCursor(10,1);
        lcd.print(zone);
        delay(200);
      }

    } // End while()1

    lcd.clear();
    lcd.home();
    lcd.print("Ativar Zona ");
    lcd.print(zone);
    lcd.print(" ?");
    lcd.setCursor(0,1);
    lcd.print("  Press Enter");
```

```
    int flagMenu31 = 1;

    while(flagMenu31){
      if(analogRead(butEnter) > 1000){
        buzzerPress(freq,enable);
        flagMenu31 = 0;
        delay(200);
      }

      if(debounceButtonPress(butReturn) == HIGH){
        digitalWrite(redLed, LOW);
        break;
      }
    } // End while()2
    int flagMenu3 = 1;

    while(flagMenu3){
      if(analogRead(butEnter) > 1000){
        buzzerPress(freq,enable);
        lcd.clear();
        lcd.home();
        lcd.print("Zona ");
        lcd.print(zone);
        lcd.print("  Ativada!");
        lcd.setCursor(0,1);
        lcd.print("  Press Return");

        activateZone(convertZoneNum(zone));
        digitalWrite(blueLed, HIGH);
        digitalWrite(redLed, LOW);
        delay(200);
      }

      if(debounceButtonPress(butReturn) == HIGH){
        lcd.clear();
        disableZones();
        digitalWrite(blueLed, LOW);
        digitalWrite(redLed, LOW);
        flagMenu3 = 0;
        break;
      }
   } // End while()3
       menu = 1;
} // End Menu 3



//----- Zone activation Demo -----
// This routine will cycle thru all the zones with a binary count
// to check if they are working (for debugging purposes only)
```

```
if(debounceButtonPress(but3) == HIGH){
  for (int j = 0; j < 256; j++) {
    activateZone(j);
    delay(250);
    if(debounceButtonPress(butReturn) == HIGH){
      break;
    }
  }
  disableZones();
}



//----- Menu 4 - Configurar Data e Hora do RTC -----
  if(analogRead(butEnter) > 1000 && menu == 4){
    buzzerPress(freq,enable);

    // Set Day
    digitalWrite(redLed, HIGH);
    lcd.clear();
    lcd.home();
    lcd.print("Config Dia:");
    lcd.setCursor(12,0);
    lcd.print(dia);
    delay(150);

    while(analogRead(butEnter) < 1000){
      if(analogRead(butUp) > 1000){
        buzzerPress(freq,enable);
        dia++;
        if(dia > 31){ dia = 31;}
        lcd.setCursor(12,0);
        lcd.print("  ");
        lcd.setCursor(12,0);
        lcd.print(dia);
        delay(200);
      }

      if(debounceButtonPress(butDown) == HIGH){
        dia--;
        if(dia < 1){ dia = 1;}
        lcd.setCursor(12,0);
        lcd.print("  ");
        lcd.setCursor(12,0);
        lcd.print(dia);
        delay(200);
      }
    } // End while()

    // Set Month
    lcd.clear();
    lcd.home();
```

```
lcd.print("Config Mes:");
lcd.setCursor(12,0);
lcd.print(mes);
delay(150);

while(analogRead(butEnter) < 1000){
  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    mes++;
    if(mes > 12){ mes = 12;}
    lcd.setCursor(12,0);
    lcd.print("  ");
    lcd.setCursor(12,0);
    lcd.print(mes);
    delay(200);
  }

  if(debounceButtonPress(butDown) == HIGH){
    mes--;
    if(mes < 1){ mes = 1;}
    lcd.setCursor(12,0);
    lcd.print("  ");
    lcd.setCursor(12,0);
    lcd.print(mes);
    delay(200);
  }
} // End while()

// Set Year
lcd.clear();
lcd.home();
lcd.print("Config Ano:");
lcd.setCursor(12,0);
lcd.print(ano);
delay(150);

while(analogRead(butEnter) < 1000){
  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    ano++;
    lcd.setCursor(12,0);
    lcd.print("  ");
    lcd.setCursor(12,0);
    lcd.print(ano);
    delay(200);
  }

  if(debounceButtonPress(butDown) == HIGH){
    ano--;
    if(ano < 2021){ ano = 2021;}
    lcd.setCursor(12,0);
    lcd.print("  ");
```

```
    lcd.setCursor(12,0);
    lcd.print(ano);
    delay(200);
  }
} // End while()

// Set Hour
lcd.clear();
lcd.home();
lcd.print("Config Hora:");
lcd.setCursor(13,0);
lcd.print(hora);
delay(150);

while(analogRead(butEnter) < 1000){
  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    hora++;
    if(hora > 23){ hora = 23;}
    lcd.setCursor(13,0);
    lcd.print("  ");
    lcd.setCursor(13,0);
    lcd.print(hora);
    delay(200);
  }

  if(debounceButtonPress(butDown) == HIGH){
    hora--;
    if(hora < 0){ hora = 0;}
    lcd.setCursor(13,0);
    lcd.print("  ");
    lcd.setCursor(13,0);
    lcd.print(hora);
    delay(200);
  }
} // End while()

// Set Minute
lcd.clear();
lcd.home();
lcd.print("Config Minuto:");
lcd.setCursor(14,0);
lcd.print(minuto);
delay(150);

while(analogRead(butEnter) < 1000){
  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    minuto++;
    if(minuto > 59){ minuto = 59;}
    lcd.setCursor(14,0);
    lcd.print("  ");
```

```
      lcd.setCursor(14,0);
      lcd.print(minuto);
      delay(200);
     }

     if(debounceButtonPress(butDown) == HIGH){
      minuto--;
      if(minuto < 0){ minuto = 0;}
      lcd.setCursor(14,0);
      lcd.print("  ");
      lcd.setCursor(14,0);
      lcd.print(minuto);
      delay(200);
     }
    } // End while()

    lcd.clear();
    lcd.home();
    lcd.print("Data: ");
    lcd.print(dia);
    lcd.print("/");
    lcd.print(mes);
    lcd.print("/");
    lcd.print(ano);
    lcd.setCursor(0,1);
    lcd.print("Hora: ");
    if(hora < 10){
    lcd.print("0");
    }
    lcd.print(hora);
    lcd.print(":");
    if(minuto <10){lcd.print("0");}
    lcd.print(minuto);
    delay(5000);
    lcd.clear();
   //Configura o RTC e sai do if final
   rtc.adjust(DateTime(ano, mes, dia, hora, minuto, segundo));
    digitalWrite(redLed, LOW);
    menu = 1;
  } // End Menu 4



//----- Menu 5 - Configuracoes Avancadas do Controlador -----
  if(analogRead(butEnter) > 1000 && menu == 5){
    buzzerPress(freq,enable);

    // Display Settings

    lcd.clear();
    lcd.home();
    lcd.print(">Buzzer");
```

```
lcd.setCursor(0,1);
lcd.print(" System Info");
delay(150);
int subMenu1 = 0;

while(analogRead(butEnter) < 1000){
  if(analogRead(butUp) > 1000){
    buzzerPress(freq,enable);
    subMenu1--;
    if(subMenu1 < 0){subMenu1 = 0;}
      lcd.home();
      lcd.print(" ");
      lcd.setCursor(0,1);
      lcd.print(" ");

      delay(150);
  }

  if(debounceButtonPress(butDown) == HIGH){
    subMenu1++;
    if(subMenu1 > 1){subMenu1 = 1;}
      lcd.home();
      lcd.print(" ");
      lcd.setCursor(0,1);
      lcd.print(" ");

  }

  if(debounceButtonPress(butReturn) == HIGH){
    lcd.clear();
    subMenu1 = 2; // Diferent value than 0 or 1, in this case will make it exit the submenu
    break;
  }

switch(subMenu1){
  case 0:
  lcd.home();
  lcd.print(">Buzzer");
  lcd.setCursor(0,1);
  lcd.print(" System Info");
  subMenu1 = 0;
  break;

  case 1:
  lcd.home();
  lcd.print(" Buzzer");
  lcd.setCursor(0,1);
  lcd.print(">System Info");
  subMenu1 = 1;
  break;

} // End switch()
```

```
} // End while()1


if(subMenu1 == 0){

    lcd.clear();
    lcd.home();
    lcd.print("Ativar Buzzer ?");
    lcd.setCursor(0,1);
    lcd.print(">S/N");
    delay(200);

while(analogRead(butEnter) < 1000){

      if(analogRead(butUp) > 1000){
        buzzerPress(freq,enable);
        lcd.setCursor(1,1);
        lcd.print(">   ");
        lcd.setCursor(1,1);
        lcd.print("SIM");
        enable = 1;
        delay(200);
      }

      if(debounceButtonPress(butDown) == HIGH){
        lcd.setCursor(1,1);
        lcd.print(">   ");
        lcd.setCursor(1,1);
        lcd.print("NAO");
        enable = 0;
        delay(200);
      }

    } // End while()

        lcd.clear();
        lcd.home();
        lcd.print("     Buzzer");
        lcd.setCursor(0,1);
        lcd.print("  Configurado!");
        delay(2000);
        digitalWrite(redLed, LOW);
        lcd.clear();
        lcd.home();
} // End if() subMenu1 = 0


  if(subMenu1 == 1){

    lcd.clear();
    lcd.home();
```

```
        lcd.print(" Firmware v1.9");
        lcd.setCursor(0,1);
        lcd.print("1508 LinesOfCode");
        delay(3500);

        lcd.clear();
        lcd.home();

        lcd.print("S:   17850 bytes");
        lcd.setCursor(0,1);
        lcd.print("GV:   1384 bytes");
        delay(3500);

        lcd.clear();
        lcd.home();

        lcd.print("Author: E.Roston");
        lcd.setCursor(0,1);
        lcd.print("  UFSCar 2021");
        delay(3500);



      } // End if() subMenu1 = 1
        lcd.clear();
        digitalWrite(redLed, LOW);
  } // End Menu 5


} // End void loop()



//---------------------- User Functions ----------------------

//---------- Convert Zone ----------

int convertZoneNum(int z){

  int zoneNum = 0;

  switch(z){

    case 0:
    zoneNum = 0;
    break;

    case 1:
    zoneNum = 2;
    break;
```

```
    case 2:
    zoneNum = 4;
    break;

    case 3:
    zoneNum = 8;
    break;

    case 4:
    zoneNum = 16;
    break;

    case 5:
    zoneNum = 32;
    break;

  }
  return zoneNum;
}

// ---------- Activate Zone ----------

// Z1 = 2; Z2 = 4; Z3 = 8; Z4 = 16; Z5 = 32;
 void activateZone(int z) {
  digitalWrite(clearPin, HIGH);
  digitalWrite(latchPin, LOW); // Keep the latch pin LOW while data is transmitting
  shiftOut(dataPin, clockPin, MSBFIRST, z); //transmite o valor de z, a começar pelo bit menos
significativo
  digitalWrite(latchPin, HIGH); //retorna o pino latch para high para sinalizar ao chip que esse não
precisa mais esperar por informação
  digitalWrite(enablePin, LOW); //Enable register output
}


// ---------- Disable Zones ----------

// This function will disable the output of all the zones

void disableZones(){
//  digitalWrite(clearPin, LOW);
    digitalWrite(clearPin, HIGH);
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, MSBFIRST, 0);
    digitalWrite(latchPin, HIGH);
    digitalWrite(enablePin, HIGH);
}


// ---------- Button Debouncing ----------
// This function will debounce any button it receives as argument

boolean debounceButtonPress(int buttonPin)
```

```
{
  boolean oldState = 0;

  boolean state = digitalRead(buttonPin);
  if(oldState != state)
  {
    delay(10); //Atraso de 10ms para não registrar o bouncing
    state = digitalRead(buttonPin); //Nova leitura do estado do botão
    buzzerPress(freq,enable);
    delay(100);
  }

  return state;
}


// ---------- Initial Screen - Title and Time ----------

void showTitleScreen(){

// Get current time
  DateTime now = rtc.now();

// Show Title

  lcd.home();
  lcd.print(" Irrigation Sys");
  lcd.setCursor(0,1);

  // Show Time
  lcd.setCursor(4,1);
  if(now.hour() < 10){
    lcd.setCursor(4,1);
    lcd.print("0");
  }
  lcd.print(now.hour(),DEC);
  lcd.setCursor(6,1);
  lcd.print(":");
  lcd.setCursor(7,1);
  if(now.minute() < 10){lcd.print("0");}
  lcd.print(now.minute(),DEC);
  lcd.setCursor(9,1);
  lcd.print(":");
  lcd.setCursor(10,1);
  if(now.second() < 10) {
    lcd.setCursor(10,1);
    lcd.print("0");
    }
  lcd.print(now.second(),DEC);
}
```

```
// ---------- System Menu ----------

void updateMenu(){

  switch(menu){
    case 0:
      menu = 1;
      break;

    case 1:
      showTitleScreen();
      break;

    case 2:
      lcd.home();
      lcd.print("    Configurar");
      lcd.setCursor(0,1);
      lcd.print("      Rotina");
      break;

    case 3:
      lcd.home();
      lcd.print("    Irrigacao");
      lcd.setCursor(0,1);
      lcd.print("      Manual");
      break;

    case 4:
      lcd.home();
      lcd.print("    Configurar");
      lcd.setCursor(0,1);
      lcd.print("  Data e Hora");
      break;

    case 5:
      lcd.home();
      lcd.print(" Configuracoes");
      lcd.setCursor(0,1);
      lcd.print("    Avancadas");
      break;

    case 6:
      menu = 5;
      break;
  }
}


// ---------- Show Current Time on LCD ----------

void showTimeLCD(){
```

```
        DateTime now = rtc.now();
        lcd.setCursor(4,1);
        if(now.hour() < 10){
          lcd.setCursor(4,1);
          lcd.print("0");
        }
        lcd.print(now.hour(),DEC);
        lcd.setCursor(6,1);
        lcd.print(":");
        lcd.setCursor(7,1);
        lcd.print(now.minute(),DEC);
        lcd.setCursor(9,1);
        lcd.print(":");
        lcd.setCursor(10,1);
        if(now.second() < 10) {
          lcd.setCursor(10,1);
          lcd.print("0");
          }
        lcd.print(now.second(),DEC);
}


// ---------- Show Current Time on LCD (Just Hours and Minutes) ----------

void showTimeLCDHM(){

        DateTime now = rtc.now();
        lcd.setCursor(2,1);
        if(now.hour() < 10){
          lcd.setCursor(2,1);
          lcd.print("0");
        }
        lcd.print(now.hour(),DEC);
        lcd.setCursor(4,1);
        lcd.print(":");
        lcd.setCursor(5,1);
        lcd.print(now.minute(),DEC);
}


// ---------- Buzzer Single Beep for Button Press ----------

void buzzerPress(int buzzFreq, int buzzEnable){
  if(buzzEnable){
    tone(buzzer,buzzFreq, 100);
  }
}


// ---------- CheckZoneAlarm ----------
// This function will check if a specific zone should be enabled or disabled at a specific DATE&TIME
```

```
void checkZoneAlarm(int zona_, int dia_, int horaInicial_, int minutoInicial_, int horaFinal_, int
minutoFinal_, int zoneFlag_){

  // Get current time
  DateTime now = rtc.now();

  // Enable specific zone at specific Date&Time
  if(now.hour() == horaInicial_ && now.minute() == minutoInicial_ && now.dayOfTheWeek() == dia_
&& zoneFlag_ == 1 ){
        activateZone(convertZoneNum(zona_));
}

  // Disable specific zone at specific Date&Time
  if(now.hour() == horaFinal_ && now.minute() == minutoFinal_ && now.dayOfTheWeek() == dia_ &&
zoneFlag_ == 1 ){
        disableZones();
  }
}
```