

FUNDAÇÃO UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS TECNOLÓGICAS
ENGENHARIA ELÉTRICA

ADRIEL WIPPEL

**SISTEMA PARA CONTROLE DE TEMPERATURA E UMIDADE EM ESTUFA
UTILIZANDO MICROCONTROLADOR ARDUINO UNO E MÓDULOS REMOTOS
ATTINY NA CONFIGURAÇÃO MESTRE-ESCRAVO VIA PROTOCOLO DE
COMUNICAÇÃO I²C**

BLUMENAU

2020

ADRIEL WIPPEL

**SISTEMA PARA CONTROLE DE TEMPERATURA E UMIDADE EM ESTUFA
UTILIZANDO MICROCONTROLADOR ARDUINO UNO E MÓDULOS REMOTOS
ATTINY NA CONFIGURAÇÃO MESTRE-ESCRAVO VIA PROTOCOLO DE
COMUNICAÇÃO I²C**

Trabalho de Conclusão de Curso apresentado ao
Curso de Engenharia Elétrica da Universidade
Regional de Blumenau como requisito parcial para
a obtenção do título de Bacharel em Engenharia
Elétrica.

Orientador: Prof. André Luiz Schlingmann, MSc.

BLUMENAU

2020

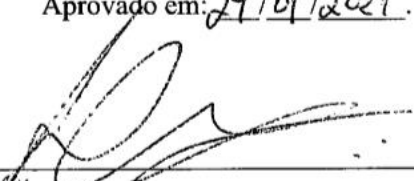
ADRIEL WIPPEL

**SISTEMA PARA CONTROLE DE TEMPERATURA E UMIDADE EM ESTUFA
UTILIZANDO MICROCONTROLADOR ARDUINO UNO E MÓDULOS REMOTOS
ATTINY NA CONFIGURAÇÃO MESTRE-ESCRAVO VIA PROTOCOLO DE
COMUNICAÇÃO I²C**

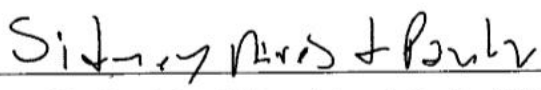
Trabalho de Conclusão de Curso apresentado ao
Curso de Engenharia Elétrica da Universidade
Regional de Blumenau como requisito parcial para
a obtenção do título de Bacharel em Engenharia
Elétrica.

Orientador: Prof. André Luiz Schlingmann, MSc.

Aprovado em: 29/01/2021.



Presidente: Prof. André Luiz Schlingmann, Me. - Orientador, FURB

Membro: Prof. Luiz Alberto Koehler, Dr., FURB

Membro: Eng. Sidney Aires de Paula, FURB

AGRADECIMENTOS

Gostaria de agradecer imensamente à minha família pela oportunidade e incentivo aos meus estudos, meu pai Otávio por ter despertado em mim desde pequeno a curiosidade, a vontade de aprender e compreender o mundo, minha mãe por ter me ensinado sobre disciplina e persistência, e ao meu irmão, por ter tornado esta jornada mais leve e trazido bom-humor sempre.

Agradeço a todos os professores e colegas de curso que estiveram junto comigo durante esta caminhada, me trazendo sempre mais conhecimento, novos olhares sobre vários assuntos e uma paixão ainda maior pela área de estudo.

Também quero agradecer aos meus colegas de trabalho, com os quais passo grande parte do meu dia, posso afirmar que aprendi muito com eles e que são uma fonte de inspiração, além de a possibilidade de aplicar meus conhecimentos na prática e resolver problemas reais através do meu conhecimento, ter feito meu amor pela área crescer a cada dia.

Nós somos uma maneira do Cosmos conhecer a si mesmo.

[CARL SAGAN, 1980]

RESUMO

Para realizar o controle do “microclima” de uma estufa, existem diversas opções disponíveis. A proposta de trabalho, é realizar este controle, utilizando microcontroladores de baixo custo, porém com grande versatilidade, permitindo controle individual de áreas específicas dentro do ambiente. Para isto, foi empregado um microcontrolador Arduino UNO, como central de processamento e entrada de parâmetros, e microcontroladores AtTiny85, em módulos remotos que capturam os dados, enviam à central e executam os comandos recebidos. Esta comunicação é realizada através do protocolo de comunicação I²C, onde a placa UNO atua como mestre e os módulos AtTiny85 como escravos. Para inserção dos parâmetros desejados, é utilizado um terminal serial. As leituras realizadas pelo sistema são dos valores de temperatura ambiente, umidade do ar e umidade do solo, executando controle de temperatura através de aquecedores, controle de umidade do ar através de chuva artificial, e umidade do solo através de irrigação.

Palavras-chave: Controle de estufa. Arduino. Comunicação serial I²C.

ABSTRACT

To control the “microclimate” of a greenhouse, several options are available. This paper proposal is to carry out this control, using low cost microcontrollers, but with great versatility, allowing individual control of specific areas within the environment. For this, an Arduino UNO microcontroller was used, as a central processing and to input parameters, and AtTiny85 microcontrollers, in remote modules that capture the data, send it to the central and execute the received commands. This communication is carried out through the I²C communication protocol, where the UNO board acts as a master and the AtTiny85 modules as slaves. To insert the desired parameters, a serial terminal is used. The the data collected by the system are the values of ambient temperature, air humidity and soil moisture, executing temperature control through heaters, air humidity control through artificial rain, and soil moisture through irrigation.

Keywords: Greenhouse control. Arduino. I²C serial communication.

LISTA DE ILUSTRAÇÕES

Figura 1 - Modelo de estufa (greenhouse).	13
Figura 2 - Arduino UNO.....	14
Figura 3 - Placa ATtiny85.	15
Figura 4 - Sensor DHT11.	16
Figura 5 - Sensor de umidade de solo.....	17
Figura 6 - Placa de interface do sensor de umidade de solo.	18
Figura 7 - Válvula hidráulica.	20
Figura 8 - Monitor Serial exibindo status dos módulos (ATtiny).	24
Figura 9 - Fluxograma do código Mestre.	26
Figura 10 - Pinagem da placa ATtiny85.....	29
Figura 11 - Diagrama eletrônico ATtiny.	30
Figura 12 - Modelo de layout para estufa.	31
Figura 13 - Modelo 3D do aquecedor.....	32
Figura 14 - Disposição da tubulação de irrigação.	33
Figura 15 - Vista em corte do método de irrigação de solo.....	34
Figura 16 - Aspecto final da montagem.	35
Figura 17 - Leitura de temperatura variando após aquecimento.	36
Figura 18 - Valores parametrizados para o Módulo 2.	37

SUMÁRIO

1 INTRODUÇÃO.....	11
1.1 PERGUNTA HIPÓTESE	11
1.2 OBJETIVO	11
1.2.1 Objetivo geral.....	11
1.2.2 Objetivos específicos	12
1.3 JUSTIFICATIVA	12
2 REVISÃO DE LITERATURA	12
2.1 ESTUFA 12	
2.2 PLANTAS E SUAS NECESSIDADES	13
2.3 MICROCONTROLADORES	14
2.3.1 ATtiny85	15
2.4 SENSOR DE UMIDADE E TEMPERATURA	16
2.5 SENSOR DE UMIDADE DE SOLO	16
2.6 PROTOCOLO DE COMUNICAÇÃO I ² C.....	18
2.7 VÁLVULAS HIDRÁULICAS	19
3 DESENVOLVIMENTO.....	21
3.1 METODOLOGIA.....	21
3.2 VISÃO GERAL DO SISTEMA	21
3.3 ALGORITMO	23
3.3.1 Código de programação Arduino UNO – Mestre	23
3.3.2 Código de programação ATtiny85 – Escravo	27
3.4 ENTRADA DOS PARÂMETROS DESEJADOS	27
3.5 CIRCUITO ELETRÔNICO DO MÓDULO (ATTINY).....	28
3.6 CIRCUITO ELETRÔNICO CONTROLADOR (ARDUINO)	30
3.7 LAYOUT DA ESTUFA	30
3.8 RESPOSTA DO SISTEMA.....	32
3.9 CONTROLE DE TEMPERATURA	32

3.10 CONTROLE DE UMIDADE DO AR.....	33
3.11 CONTROLE DE UMIDADE DO SOLO.....	33
4 CONCLUSÃO E RECOMENDAÇÕES.....	35
REFERÊNCIAS	38
APÊNDICE A – DIAGRAMA DO CIRCUITO.....	39
APÊNDICE B – CÓDIGO DE PROGRAMAÇÃO DO MESTRE	40
APÊNDICE C – CÓDIGO DE PROGRAMAÇÃO DO ESCRAVO	58
APÊNDICE D – MANUAL DE UTILIZAÇÃO.....	60
APÊNDICE E – LISTA DE MATERIAIS	69
ANEXO A – LAYOUT PLACA ARDUINO UNO.....	70

1 INTRODUÇÃO

O cultivo de plantas sempre esteve associado ao desenvolvimento da humanidade, pois um dos maiores marcos da história humana foi o desenvolvimento da agricultura, que possibilitou o surgimento das civilizações. Isto foi um processo longo e trabalhoso, principalmente devido à impossibilidade do controle do clima. Entretanto, atualmente desenvolvemos novas tecnologias, que possibilitam a criação de microclimas específicos para um grande tipo de cultivos, são as estufas.

O desenvolvimento das estufas possibilitou um avanço na agricultura, permitindo o cultivo de diversas plantas em ambientes que não possuem as características próprias para seu desenvolvimento. Isto se deve ao fato de que nelas é possível haver o controle de fatores fundamentais para as plantas, como temperatura, umidade e irrigação. Tendo em vista a necessidade deste controle, houve o propósito de criar um sistema baseado em microcontroladores para a criação de um sistema que monitore e atue para que os valores dos fatores principais sejam controlados.

A implementação de um sistema de controle de temperatura, umidade de ar e irrigação para estufas, vem de encontro com a aplicação de diversos conceitos estudados durante o curso de Engenharia Elétrica, pois está baseado em eletrônica analógica e digital, automação e programação. O emprego de microcontroladores torna o projeto bastante versátil e “limpo”, necessitando apenas de alguns poucos componentes periféricos.

Foi adotado o propósito de implementação prática do circuito, para fins de comprovação empírica do seu funcionamento.

1.1 PERGUNTA HIPÓTESE

Como projetar um sistema de monitoramento e controle de uma estufa, que seja fácil de ser implementado e operado, de baixo custo e versátil.

1.2 OBJETIVO

A seguir serão apresentados os objetivos a serem alcançados através deste trabalho de conclusão de curso.

1.2.1 Objetivo geral

O objetivo deste trabalho é desenvolver um circuito eletrônico comandado por microcontroladores que monitore e controle elementos fundamentais para o funcionamento de uma estufa (temperatura, umidade do ar, irrigação, entre outros), que possa ser aplicado tanto

para estufas de grande porte através de acoplamento de módulos, quanto em pequenas estufas que poderão ser utilizadas em âmbito domiciliar, sendo de implementação fácil e rápida.

1.2.2 Objetivos específicos

- a) Projetar um circuito eletroeletrônico gerenciado por um microcontrolador que acione os dispositivos necessários para controle de umidade e temperatura;
- b) Desenvolver um código de programação baseado em Arduino, que realize a leitura de sensores de umidade e temperatura e execute as funções de controle necessárias;
- c) Estabelecer uma comunicação serial (I²C) entre o microcontrolador principal e os módulos endereçáveis, e;
- d) Montagem de protótipo do circuito para validação prática.

1.3 JUSTIFICATIVA

Com o aumento pela demanda de alimentos mais saudáveis e de menor impacto para o meio-ambiente (como os alimentos orgânicos por exemplo), a produção de vegetais em estufa, acaba sendo uma ótima opção, pois inibe a entrada de insetos, vermes ervas daninhas entre outros, tornando desnecessário o uso de pesticidas e outros compostos químicos. Outro ponto é a possibilidade de implantação deste sistema em pequenas estufas, que as pessoas poderão ter em suas casas para a produção do próprio alimento, já que a proposta é que ele seja de baixo custo e instalação simplificada.

2 REVISÃO DE LITERATURA

Esta seção destina-se a apresentar os fundamentos teóricos das partes e componentes relevantes para a aplicação do projeto proposto, a fim de clarificar a compreensão do sistema, até por leitores que não estejam familiarizados com todos os conceitos que servem como base para o desenvolvimento deste trabalho de conclusão de curso.

2.1 ESTUFA

Para se desenvolver um controle de umidade e temperatura de uma estufa, é necessário entender sua finalidade, construção e seu funcionamento. Por este motivo, este tópico destina-se a apresentar os conceitos e componentes básicos de uma estufa.

Uma estufa é uma estrutura cuja principal função é a protecção[sic] das plantações contra ameaças, como por exemplo é o caso do vento e das chuvas fortes. Podemos ainda definir uma estufa como uma estrutura coberta com

material transparente à radiação solar. As estufas são utilizadas em culturas protegidas permitindo no seu interior, o trabalho de pessoas e máquinas. (MOREIRA, 2019).

De acordo com Hewitt (2019), as estufas são construídas com um vidro que é opaco a radiação infravermelha, portanto quando a luz visível o atravessa e é absorvida pelo chão e este reemite infravermelho, assim, essa radiação ficaria retida dentro da estufa.

Segundo Attenborough (2020), com o uso de estufas, é possível produzir até dez vezes mais alimentos na mesma área, utilizando menos água, menos fertilizantes, menos pesticidas e emitindo menos carbono.

Figura 1 - Modelo de estufa (*greenhouse*).



Fonte: 3D Warehouse (2020).

2.2 PLANTAS E SUAS NECESSIDADES

Como o objetivo de uma estufa é o cultivo de plantas, principalmente legumes e verduras, cabe aqui um breve estudo relacionado às necessidades existentes para manter as plantas saudáveis através do controle da umidade e temperatura sobre elas.

As plantas precisam de luz do Sol, água e ar para crescer. São incapazes de se locomover, e suas células têm paredes rígidas constituídas de um material resistente, a celulose. Além disso, elas fabricam o próprio alimento através de um processo denominado fotossíntese. Para isso, utilizam energia do Sol, água e dióxido de carbono (também chamado de gás carbônico). (BRITANICA ESCOLA, 2020).

Para suprir estas necessidades, um sistema que monitore os níveis de temperatura e umidade e os regule de acordo com o demandado pelas plantas, mostra-se bastante eficiente, para que estas tenham um bom desenvolvimento e crescimento.

2.3 MICROCONTROLADORES

Os microcontroladores tornaram os circuitos eletrônicos muito mais versáteis, pois podem cumprir diferentes funções, sem a necessidade de alterações no circuito periférico, tornando a criação de projetos eletrônicos muito mais fácil e barata.

[...] microcontroladores são, essencialmente, processadores autônomos, destinados a serem *embarcados* em algum dispositivo eletrônico que não é definitivamente um computador. Eles são baratos e fáceis de usar, e permitem que você coloque a inteligência de um computador em praticamente qualquer coisa eletrônica. (HOROWITZ, 2017. p.1053).

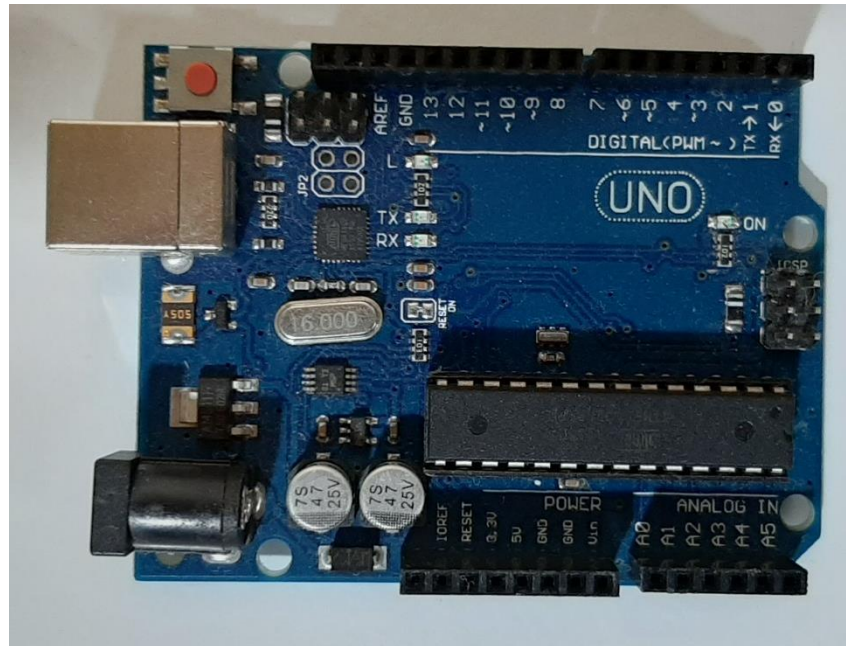
Existem diversas marcas e modelos disponíveis no mercado, para este projeto, foi adotado o *Arduino UNO*, um microcontrolador da família AVR da empresa Atmel, sendo programado através da interface (IDE) Arduino.

O Arduino é uma placa de desenvolvimento baseada em um microcontrolador Atmel AVR. Possui conectores de entrada e saída para controle de dispositivos e um circuito que facilita sua conexão com um computador via porta USB. Foi criada em 2005 por professores que desejavam fornecer aos alunos uma plataforma de fácil desenvolvimento e aprendizado de eletrônica e desenvolvimento de software. (BAUERMEISTER, 2018).

Os microcontroladores Arduino possuem um baixo custo e grande versatilidade, que aliados à sua plataforma aberta fornece uma ampla gama de aplicações e facilidade para encontrar projetos e bibliotecas com funções específicas.

A placa Arduino UNO possui uma porta para comunicação serial, que suporta além de outros protocolos, o UART e o I²C. Ela conta ainda com vinte portas (interfaces de saída física de sinal - GPIO), sendo que dessas, seis podem ser utilizadas para leitura de valores analógicos de tensão entre 0 e 5V e outras seis portas podem ser utilizadas como saída PWM.

Figura 2 - Arduino UNO.



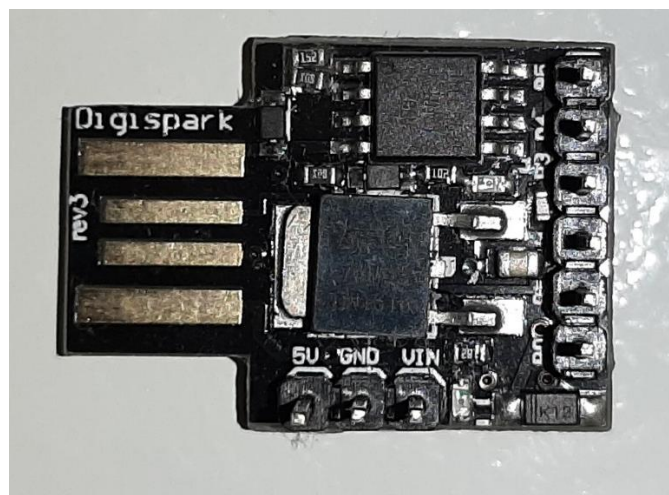
Fonte: Autor (2020).

2.3.1 ATtiny85

No projeto também é utilizado a placa de desenvolvimento *ATtiny85* da empresa *Digispark*, também baseado em microcontrolador AVR.

Esta placa possui seis pinos de GPIO interface para comunicação I²C, além de quatro dos seis pinos de GPIO poderem ser utilizados como entrada analógica.

Figura 3 - Placa ATtiny85.



Fonte: Autor (2020).

Esta placa possui conector USB já embutido (lado esquerdo), tornando-a muito prática para gravação do código através de um computador.

2.4 SENSOR DE UMIDADE E TEMPERATURA

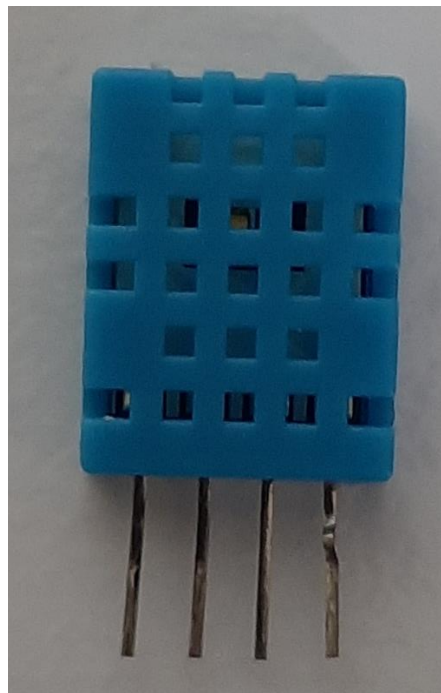
O sensor empregado para a leitura de temperatura e umidade, foi o DHT11, por ser compacto e de baixo custo.

O sensor DHT11 inclui um componente medidor de umidade e um componente NTC para temperatura, ambos conectados a um controlador de 8-bits. O interessante neste componente é o protocolo usado para transferir dados entre o MCU e DHT11, pois as leituras do sensor são enviadas usando apenas um único fio de barramento. (THOMSEN, 2013).

Os dados deste sensor são então recebidos pelo terminal DATA e podem ser lidos utilizando-se uma biblioteca específica para Arduino.

A sequência de pinos, da esquerda para a direita é: Pino 1 - +VCC, Pino 2 – DATA (dados), Pino 3 – não conectado, Pino 4 – GND.

Figura 4 - Sensor DHT11.



Fonte: Autor (2020).

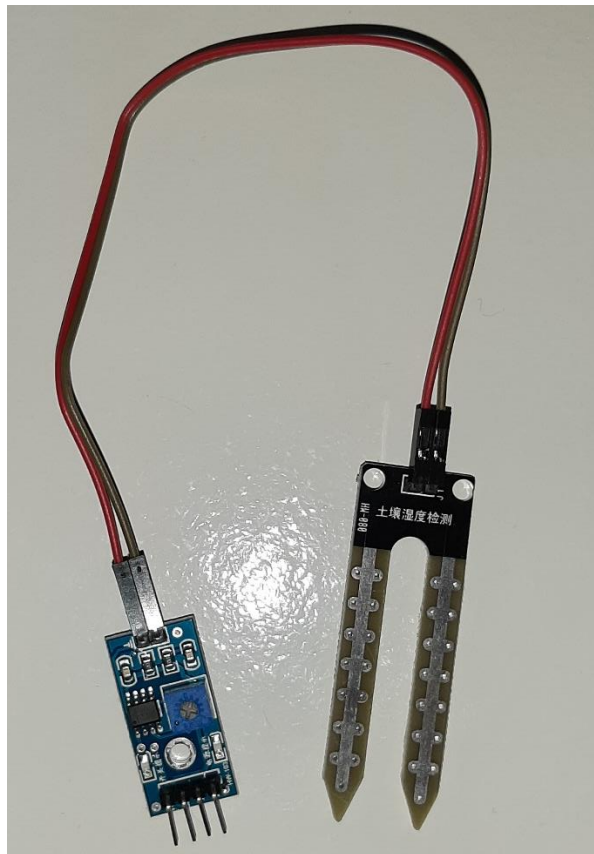
2.5 SENSOR DE UMIDADE DE SOLO

O sensor de umidade do solo consiste em 2 partes: uma sonda que entra em contato com o solo, e um pequeno módulo contendo um chip comparador LM393, que vai ler os dados que vêm do sensor e enviá-los para o

microcontrolador [...]. Como saída, temos um pino D0, que fica em nível 0 ou 1 dependendo da umidade, e um pino de saída analógica (A0), que possibilita monitorar com maior precisão usando uma porta analógica do microcontrolador. (THOMSEN, 2018).

Em resumo, este sensor é formado por dois terminais que são enterrados no solo, e conforme a umidade do solo varia, a resistência elétrica entre as varetas também muda, alterando a tensão lida na entrada analógica do microcontrolador. Quando a sonda está com os terminais secos, sem umidade, a tensão de saída A0 é de 5 volts, quando o sensor está imerso em um líquido de alta condutividade, a saída de tensão em A0 é de 0 volts (umidade de 100%).

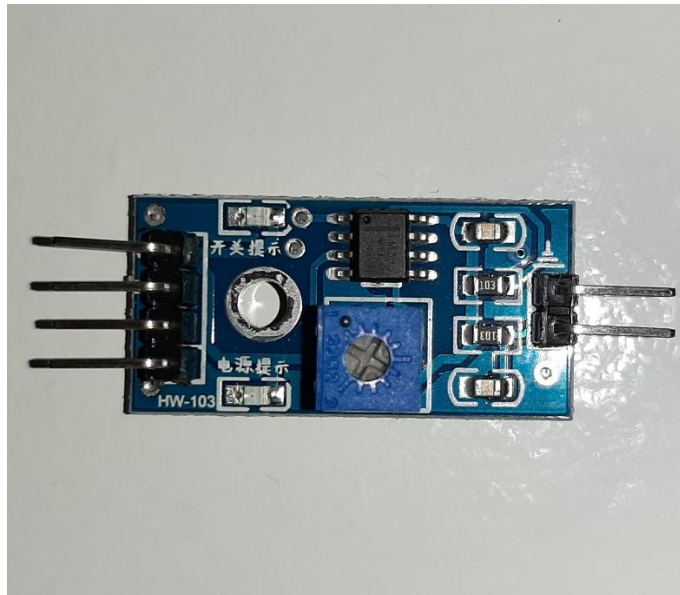
Figura 5 - Sensor de umidade de solo.



Fonte: Autor (2020).

O dispositivo que realiza a conversão do valor de resistência das varetas em uma tensão analógica, é a placa representada na Figura 6, ela possui um *trimpot* e um comparador LM393.

Figura 6 - Placa de interface do sensor de umidade de solo.



Fonte: Autor (2020).

Os dois pinos à direita são o ponto onde a sonda que será introduzida no solo é ligada. Nos pinos da esquerda, de baixo para cima tem-se: A0 (saída de tensão analógica de 0 a 5V), D0 (saída digital que fica em nível alto – 5V – quando a umidade está abaixo do regulado no trimpot), GND e por fim VCC.

2.6 PROTOCOLO DE COMUNICAÇÃO I²C

Um dos objetivos do circuito, é que ele seja de fácil implementação e bastante versátil, para isso, foi empregado um *Arduino UNO* na função “mestre” e os módulos “escravos” serão os microcontroladores ATtiny85, para isto, é necessário estabelecer uma comunicação entre a central (*Arduino UNO*) e os módulos remotos (ATtiny85). Assim, foi adotado o protocolo I²C, pela possibilidade de endereçamento de cada unidade remota, possibilitando a criação de várias áreas com controle independente dentro da estufa.

O barramento de interface serial *Inter-Integrated-Circuit* (IIC, I2C ou I²C) foi criado pela Philips, para a comunicação entre chips. [...] ele usa apenas 2 fios, que constituem no barramento conectado a todos os CIs escravos [...] o endereçamento é enviado (em primeiro lugar) na mesma linha que os dados são enviados ou recebidos; o barramento é “*half-duplex*” – isto é, os dados podem se mover apenas um sentido de cada vez (o sentido é especificado por um bit seguinte ao endereço); embora I²C seja uma arquitetura mestre-escravo (como SPI), qualquer dispositivo no barramento pode se tornar mestre quando

o mestre atual renuncia ao controle (pelo envio de um bit de fim que termina a sua sessão com um escravo particular). (HOROWITZ, 2017. p.1034).

Ainda de acordo com Horowitz (2017), no barramento I²C temos uma função para cada fio do par que o compõe, um deles é a linha de *clock* (SCL) e o outro a linha de dados (SDA). A linha SCL é ativada pelo mestre, já a linha SDA que é bidirecional é ativada pelo mestre para especificar o endereço do escravo (7 bits), informa também o sentido de transferência utilizando 1 bit, após isso o escravo transmite um bit de confirmação (ACK), então finalmente os bytes contendo os dados são transmitidos do escravo ao mestre, ou vice-versa, a velocidade de transmissão é controlada pelo mestre. A transmissão é finalizada quando o mestre envia um bit de fim, após o último byte tiver sido transferido.

Uma grande vantagem deste protocolo, é um menor custo com estrutura, já que emprega apenas dois fios.

2.7 VÁLVULAS HIDRÁULICAS

Para o controle do fluxo de água para a irrigação são empregadas válvulas hidráulicas, utilizadas de dois métodos, uma para irrigação central (criando uma espécie de “chuva artificial”) e outras de menor calibre, instaladas de forma dispersa em vários pontos estratégicos, para uma irrigação pontual direta no solo.

[...] as válvulas hidráulicas, são equipamentos desenvolvidos para realizar o controle dos sistemas hidráulicos industriais. Basicamente, todas as válvulas hidráulicas têm funções parecidas que são de reduzir, aumentar, fechar ou manter a pressão em determinados pontos. (HIDRAUTEC, 2020).

Figura 7 - Válvula hidráulica.



Fonte: Autor (2020).

As válvulas empregadas possuem especificações de tensão e corrente superiores à capacidade das portas de saída dos microcontroladores, devido a isto, é necessário o emprego de interfaces como relés e/ou transístores para “acoplar” e adequar os valores.

3 DESENVOLVIMENTO

Aqui serão apresentados os métodos e componentes utilizados para a elaboração do trabalho, bem como detalhes do circuito, seu princípio de funcionamento e resultados obtidos.

3.1 METODOLOGIA

Este trabalho é baseado em método qualitativo, no que diz respeito ao estudo de fundamentação teórica e métodos já conhecidos, aplicados e documentados na literatura técnica. É baseado também em métodos empíricos e testes de hipóteses, tendo como objetivo a execução do projeto proposto.

3.2 VISÃO GERAL DO SISTEMA

O sistema consiste basicamente em 4 partes interligadas:

- Arduino UNO: Tem a função de centralizar as informações e direcionar comandos para os módulos. Para isto, ele é interligado aos módulos (serão explicados no tópico abaixo), através do protocolo de comunicação I²C, na configuração mestre-escravo. O Arduino UNO foi configurado para atuar como “mestre”, ele abre uma requisição de dados para cada módulo em sua rede de comunicação (um módulo por vez), através do endereço do “escravo” desejado, ao receber os dados do módulo em questão, ele analisa os dados, processa a informação e envia comandos de volta, para que o escravo execute a ação necessária (ligar ou desligar uma válvula de irrigação por exemplo). No mestre, também ficarão gravados os parâmetros desejados para o sistema, como por exemplo umidade e temperatura desejados.

- ATtiny: Serão os módulos instalados em locais estratégicos pela estufa, contendo vários pontos de medição de temperatura, umidade do ar e umidade do solo, para, quando requisitado pelo mestre, envie essas informações. Outra função dos módulos ATtiny, será de executar as instruções enviadas pelo mestre (ligar e desligar válvulas por exemplo).

- Sensores: Os sensores empregados são o DHT11, para leitura da temperatura e umidade do ar; e, sensor de umidade de solo, que consiste em um par de varetas onde é aplicada uma tensão elétrica, e o valor da tensão de saída varia de acordo com a resistência do solo (que é relacionada à umidade presente).

- Atuadores: Para executar a função desejada, são utilizados alguns atuadores com funções distintas. Dois tipos de válvulas hidráulicas compõem o circuito, uma é destinada à irrigação geral de uma área definida dentro da estufa, a fim de gerar uma “chuva artificial” no espaço necessário; a outra, é uma válvula menor, que será utilizada para a irrigação no solo, injetando água diretamente no recipiente onde estão as plantas. Também serão utilizados

aquecedores, com o objetivo de gerar um aumento da temperatura interna quando demandado, como por exemplo no inverno.

O APÊNDICE A mostra um digrama da planta, indicando as conexões e sentido de fluxo dos dados entre os componentes.

O funcionamento se dá pela seguinte forma: Cada módulo (microcontrolador ATtiny) tem acoplado um sensor DHT11 (temperatura e umidade do ar), sensor de umidade de solo e duas válvulas hidráulicas. Através da porta D3 do ATtiny, são recebidos os valores de temperatura e umidade do ar enviados pelo sensor DHT11, quando requisitado pelo mestre, este valor é enviado através da rede I²C e guardado em uma variável do tipo “int” chamada “h”, após isso é então exibido através da porta serial. Este dado também é comparado com o valor de umidade do ar desejado que foi inserido no programa, e caso o valor de umidade lido esteja abaixo do parametrizado, o mestre envia ao escravo (também através do protocolo I²C) um comando para ligar a válvula de irrigação do respectivo módulo. O processo de controle de temperatura é muito semelhante, tendo como diferença principal que o valor lido pelo sensor DHT11 é armazenado na variável “t”.

A leitura do sensor de umidade de solo é feita através do pino analógico A0 (“senSolo” no código de programação) da placa ATtiny. Este valor é então armazenado na variável “umiSolo” e enviados ao mestre via I²C, mediante requisição. O valor recebido é armazenado no mestre em uma variável também chamada “umiSolo”. O valor lido pela porta analógica A0 usa um *byte* de memória, ou seja, seu valor pode variar de 0 a 255, sendo 0 quando a tensão detectada no pino for igual a 0 volts, e, 255 quando a tensão na porta for de 5V. Para adequar esses valores na faixa de 0 a 100 (0 para 0% de umidade e 100 indicando umidade máxima) é utilizada a função “map” (linha 119 do código do mestre) que condiciona o valor recebido, dentro da faixa desejada e o armazena novamente na mesma variável “umiSolo”, o valor de umidade de solo é então comparado com o desejado (previamente parametrizado pelo usuário) e envia comandos para ligar ou desligar a válvula de irrigação de solo de acordo com a necessidade.

A ação de controle que é enviada aos módulos (linhas 132 a 154 no programa do mestre), é feita através de Operadores de Comparação e Operadores Booleanos, o resultado é então atribuído à um número decimal que tem um significado binário. Os números são 0 (00 em binário), 1 (01 em binário), 2 (10 em binário) e 3 (11 em binário), esse valor binário indica o estado das portas D1 e D4 do controlador ATtiny, sendo D1 (válvula de irrigação tipo “chuveiro”) o *bit0* e, D4 (válvula de irrigação do solo) o *bit1*. O mesmo protocolo de codificação

é utilizado na programação dos módulos (linhas 57 a 75 no programa do escravo), onde o número decimal recebido é utilizado para definir o estado das saídas D1 e D4.

Tabela 1 - Relação entre o valor enviado pelo mestre via I²C e estado físico das portas no módulo ATtiny.

Valor Decimal	Valor Binário	Porta digital D4	Porta digital D1
0	00	DESLIGADA	DESLIGADA
1	01	DESLIGADA	LIGADA
2	10	LIGADA	DESLIGADA
3	11	LIGADA	LIGADA

Fonte: Autor (2020).

3.3 ALGORITMO

O código de programação empregado foi criado através da IDE Arduino, ela baseia-se em linguagem de programação “C++” orientada a objeto e adaptada para microcontroladores. Ela foi escolhida por sua versatilidade e facilidade de implementação, devidas principalmente pelas bibliotecas de código e o vasto número de documentos e informações disponíveis na internet. Foram utilizados dois códigos distintos, um para o Arduino UNO (mestre) e outro para a placa ATtiny (escravo), cada um com suas bibliotecas, funções e comandos específicos. Fez-se necessária a utilização de bibliotecas específicas para algumas funções do programa. Para estabelecer o protocolo de comunicação I²C, foi empregada a biblioteca “Wire” no Arduino UNO (mestre) e da biblioteca “TinyWireS” no ATtiny (escravo), neste, também foi utilizada a biblioteca “TinyDHT” para obtenção de dados do sensor DHT11 através do controlador ATtiny. Outra biblioteca empregada foi “EEPROM” no microcontrolador mestre, para que os valores desejados de temperatura e umidade sejam armazenados em uma memória não volátil, ou seja, mesmo com o circuito desligado, os valores atribuídos às variáveis serão mantidos.

Os códigos de programação completos estão nos APÊNDICES B e C para o Arduino UNO e ATtiny85 respectivamente.

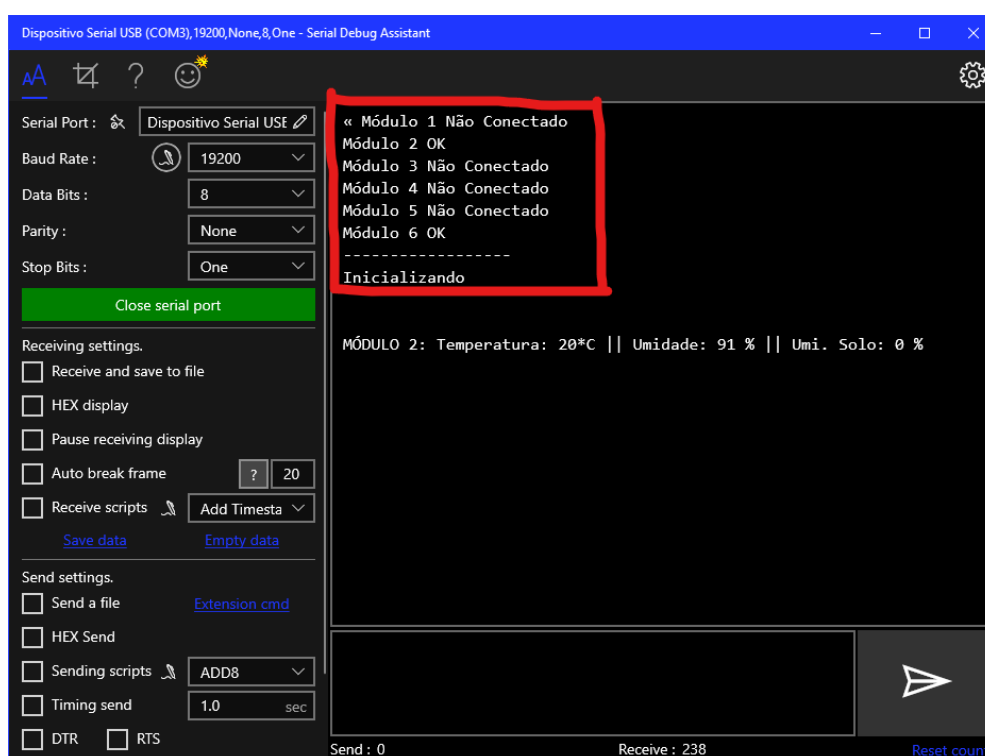
3.3.1 Código de programação Arduino UNO – Mestre

Nas primeiras linhas do código do código (linha 1 até 43), são declaradas as variáveis utilizadas, incluídas as bibliotecas que serão utilizadas e definidos nomes para as portas do microcontrolador.

A função “void setup()” (linhas 45 à 98) é executada apenas uma vez pelo microcontrolador, ela serve para definição dos parâmetros iniciais, como por exemplo habilitar a comunicação I²C (linha 47) e a comunicação serial (linha 48). Na linha 49 do código, o endereço inicial é definido como zero. Esta variável “endereco” é utilizada para que o programa

realize uma varredura de todos os módulos remotos (escravos) conectados na rede I²C, sem a necessidade da criação de código para cada endereço de escravo, economizando assim várias linhas no programa. Entre as linhas 51 e 68, são definidos os bytes utilizados na EEPROM e quais variáveis estão atribuídas a eles. Nas linhas 72 até 78, os pinos de GPIO são definidos como saídas, para que possam ser utilizados para o acionamento de periféricos no circuito. A função contida nas linhas 83 a 93 servem para realizar uma varredura inicial de todos os endereços conectados na rede I²C e exibir o status de cada um no monitor serial, como na figura abaixo:

Figura 8 - Monitor Serial exibindo status dos módulos (ATtiny).



Fonte: Autor (2020).

No caso da Figura 8, pode-se verificar que os únicos módulos encontrados pelo Mestre através do barramento I²C foram os com endereço 2 e 6.

À partir da linha 100 do código, inicia-se o “void loop()” ou loop principal que é executado indefinidamente enquanto o microcontrolador estiver ligado. Aqui também a primeira instrução é definir o endereço inicial como zero, para iniciar a varredura dos módulos disponíveis através do protocolo I²C.

As linhas 105, 106, 107 e 108, criam “arrays” para armazenar os parâmetros de umidade do ar, umidade do solo, temperatura e pinos digitais, respectivamente. Através do *array*, é possível utilizar uma única variável, porém com várias posições de memória, podendo realizar uma varredura concatenada com cada endereço dos escravos que estão sendo consultados, tendo

o mesmo efeito de reduzir enormemente o número de linhas de código necessárias. Por exemplo, para armazenar os parâmetros de temperatura inseridos pelo usuário, é utilizado o *array* “setTemp” (linha 107 do código), que possui sete variáveis dentro dele, a primeira é o valor zero, pois este valor de endereço não é utilizado por nenhum módulo. O segundo é a variável “setTemp1” que é utilizado apenas quando o endereço do escravo consultado é igual a 1. O a segunda variável é “setTemp2” que é utilizada somente quando a comunicação é estabelecida com o escravo que possui o endereço 2, e assim sucessivamente.

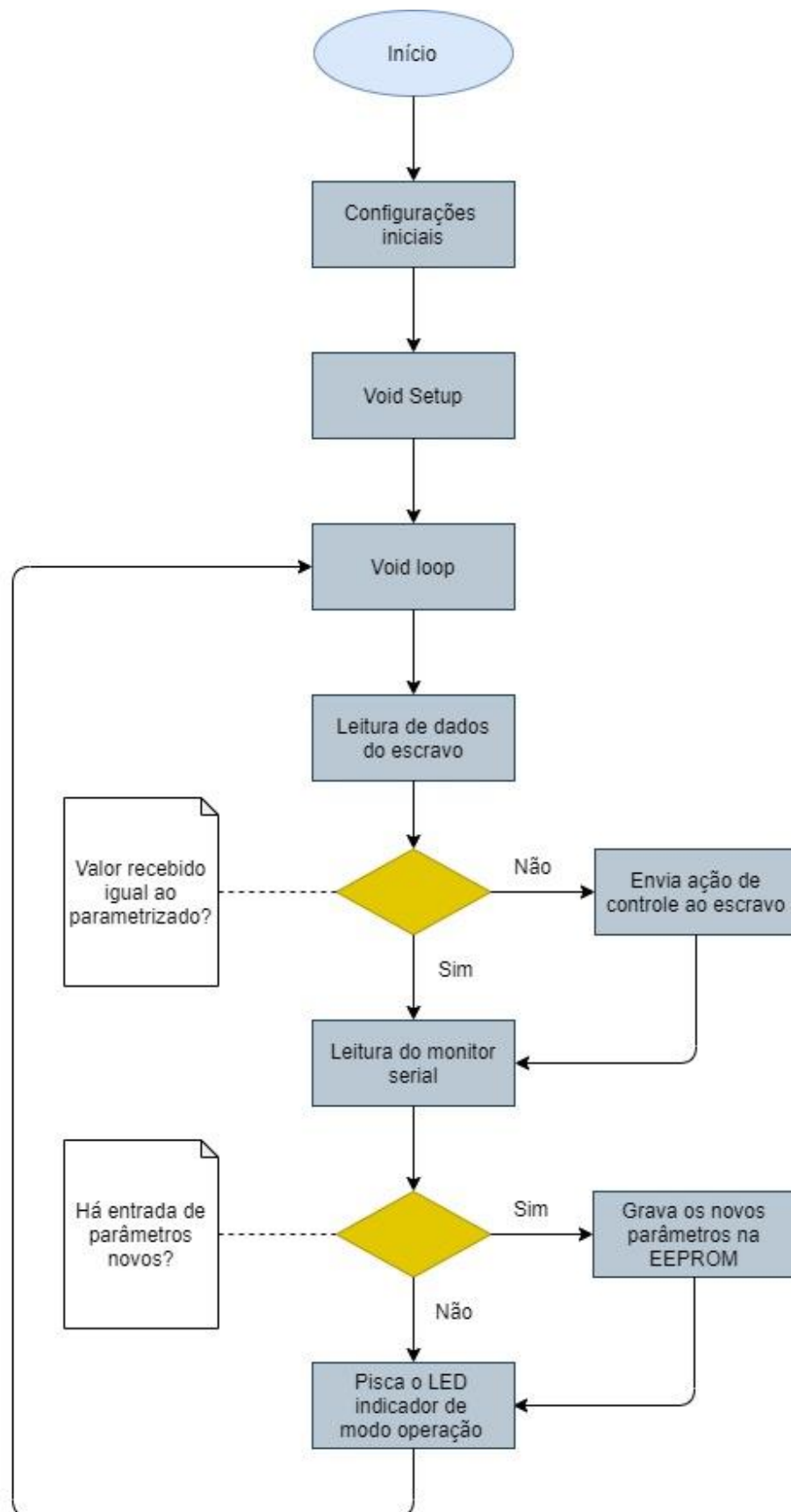
A função “while (endereço < 6)” na linha 110, é finalizada apenas na linha 647. Ela é utilizada para gerar uma requisição a cada módulo ATtiny conectado que tenha endereço de 1 a 6, pois o comando da linha 111 faz com que a cada ciclo, o próximo endereço (próximo escravo) seja consultado. Na linha 113 o mestre envia uma requisição de 3 *bytes* ao escravo com endereço em questão.

A linha 115 é muito importante, pois apenas executará os próximos comandos, caso o escravo responda às requisições do mestre, poupando tempo de ciclo caso ele não esteja conectado ao barramento de comunicação. Isto também evita que o monitor serial exiba valores nulos para os módulos que não estejam conectados na rede. Dentro deste comando (linhas 115 até 161) são realizadas as funções de leitura das informações enviadas pelos módulos remotos (ATtiny), estes dados são então enviados ao terminal serial para serem exibidos. E é também aqui que os valores de umidade de ar, umidade de solo e temperatura são comparados com os valores parametrizados, e as ações de controle são enviadas (ligar ou desligar válvulas e/ou aquecedores).

As linhas de código 165 até 639 são responsáveis pela entrada dos parâmetros desejados pelo usuário, elas executam a leitura dos textos enviados através do monitor serial, e compara com as funções programadas, como por exemplo a função “Config” (linha 179). Esta função em específico, faz com que o programa entre no modo de configuração e consulta de parâmetros, o permanece neste modo, até o monitor serial receber o comando “FimConfig” (linha 180). Enquanto o controlador estiver no modo de configuração, um LED indicativo ficará aceso (linha 181) quando o usuário sair do modo configuração, digitando o comando “FimConfig”, este mesmo LED mudará para um status intermitente, ficará piscando (linhas 642, a 645), indicando que o controlador está no modo de operação.

As linhas de código 652 até 665 estão fora do loop principal, elas são funções específicas para leitura e gravação de dados na EEPROM do Arduino UNO, que são “chamadas” pelo código principal quando necessário (linhas 205 e 546 por exemplo).

Figura 9 - Fluxograma do código Mestre.



Fonte: Autor (2020).

3.3.2 Código de programação ATtiny85 – Escravo

Assim como no código de programação do mestre, as primeiras linhas do programa servem para declaração de variáveis, inclusão de bibliotecas e definição dos nomes dos pinos de GPIO. Para o ATtiny85, são as linhas 1 até 19.

Entre as linhas 21 e 33, está o “void setup” loop de configurações que é executado apenas uma vez quando o microcontrolador é ligado. Nele inicia-se a comunicação na rede I²C (linha 23) utilizando o endereço especificado do módulo na linha 3. A linha 27 inicia a biblioteca para leitura do sensor DHT11.

Entre as linhas 35 e 51, está o loop principal, que é executado repetidamente. Nele, existe um comando para verificar se há alguma requisição do mestre (linha 39), as próximas linhas são responsáveis pela leitura dos valores dos sensores. A linha de código 41 realiza a leitura do valor da porta analógica A0 e armazena esse valor na variável “umiSolo”. As linhas 43 e 44 fazem a leitura de umidade do ar e temperatura, e armazena os valores nas variáveis “h” e “t” respectivamente.

A partir da linha 53 até o fim do código, está a função de incementação que faz a leitura dos valores enviados pelo mestre no barramento de comunicação, este valor é armazenado na variável “data” e de acordo com o valor recebido, é executada a ação de ativar ou desativar as portas digitais (esta relação já foi apresentada na Tabela 1).

3.4 ENTRADA DOS PARÂMETROS DESEJADOS

A inserção dos parâmetros desejados (temperatura, umidade e umidade do solo), é feita através de um terminal serial instalado em um computador (Figura 8), obedecendo uma listagem de comandos a serem digitados neste terminal. A velocidade de comunicação da porta serial (baud rate) utilizada neste programa no Arduino é de 19200 bps.

Para entrar-se no modo de configuração, é necessário digitar o comando “Config” e enviá-lo através do monitor serial. Há um LED indicativo que pisca durante a execução do programa e muda seu estado para “aceso” quando o modo de configuração é habilitado. Com a função de configuração ativada, é possível verificar os valores já inseridos no sistema, para isso basta ser digitado e enviado o comando “VerParametros” (é necessário observar as letras maiúsculas e minúsculas e utilizar o comando exatamente da forma apresentada). No monitor serial será exibido o valor de temperatura, umidade do ar e do solo, configurados para todos os módulos. Para envio dos parâmetros desejados, utilizam-se três caracteres, o primeiro para indicar qual a função de controle deseja-se atribuir, as opções são:

- T: Para inserir a temperatura desejada (em graus celsius);
- U: Para inserir a umidade do ar desejada (umidade relativa em percentual);
- S: Para inserir a umidade do solo desejada (umidade em valor percentual).

O segundo caractere é a letra “M” indicando a palavra “módulo”. O terceiro caractere é o endereço do módulo que se deseja configurar, partindo do endereço 1 até 6.

Por exemplo para configuração de temperatura do módulo 3, o comando a ser enviado será TM3 (temperatura módulo 3), para configuração da umidade do solo do módulo 5, o comando será SM5 (umidade do solo módulo 5) e assim por diante. Quando o comando é recebido, uma mensagem é exibida informando qual o parâmetro e módulo será alterado. Então basta digitar o valor desejado e enviá-lo pelo monitor serial. Uma nova mensagem é exibida com o novo valor atribuído ao parâmetro desejado.

As variáveis no código para entrada destes parâmetros são: setUmiAr1, setUmiAr2, setUmiAr3, setUmiAr4, setUmiAr5 e setUmiAr6, para leitura da umidade de ar desejada para cada módulo remoto. Estes valores são armazenados em um array, para que seu valor possa variar de 1 a 6, fazendo a varredura em cada endereço de módulo disponível. Também são utilizadas as variáveis setUmiAr1, setUmiAr2, setUmiAr3, setUmiAr4, setUmiAr5 e setUmiAr6, que também são armazenadas em um array para que o valor numérico varie de acordo com o endereço do módulo. O mesmo procedimento é feito para os parâmetros de temperatura, utilizando as variáveis setTemp1, setTemp2, setTemp3, setTemp4, setTemp5, setTemp6. Cada valor é então guardado em dois endereços da EEPROM (dois bytes de memória), e são consultados pelo programa quando necessário.

O APÊNDICE , serve como um manual de utilização para o usuário, e contém detalhes dos comandos disponíveis para entrada de parâmetros bem como consulta do status do sistema e parâmetros gravados.

3.5 CIRCUITO ELETRÔNICO DO MÓDULO (ATTINY)

A placa de desenvolvimento ATtiny, precisa ser alimentada com uma tensão de 5 volts em corrente contínua, porém ela já possui um regulador de tensão para a faixa de 7 à 35V embutido possibilitando a utilização de uma fonte de 12Vcc no projeto.

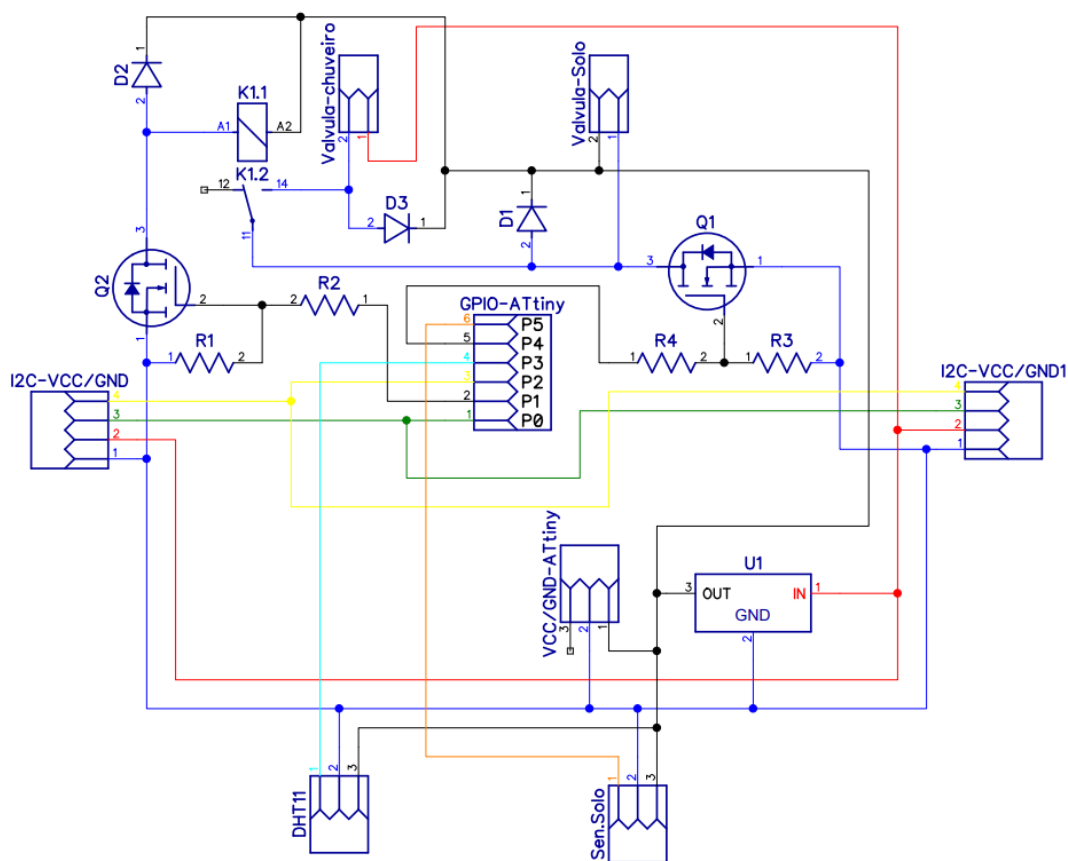
Figura 10 - Pinagem da placa ATtiny85.



Fonte: Arduino e Cia (2016).

Na Figura 10, é apresentada a nomeação de cada terminal do controlador ATtiny85, onde podemos verificar que os pinos D0 (P0 no diagrama eletrônico) e D2 (P2 no diagrama eletrônico) são utilizados para interface de comunicação I²C. O terminal D1 (P1 no diagrama eletrônico) é responsável pelo acionamento da válvula de irrigação principal (tipo chuveiro), instalada no teto da estufa. A porta D3 (P3 no diagrama eletrônico) é ligado ao terminal DATA do sensor DHT11, onde recebe os dados de temperatura e umidade. D4 (P4 no diagrama eletrônico) é responsável pelo acionamento da válvula de irrigação diretamente no solo através de mangueiras furadas e enterradas. Por fim, D5 (A0 ou P5 no diagrama eletrônico) é utilizado para a leitura do valor analógico de tensão enviado pelo sensor de umidade de solo.

Figura 11 - Diagrama eletrônico ATtiny.



Fonte: Autor (2020).

3.6 CIRCUITO ELETRÔNICO CONTROLADOR (ARDUINO)

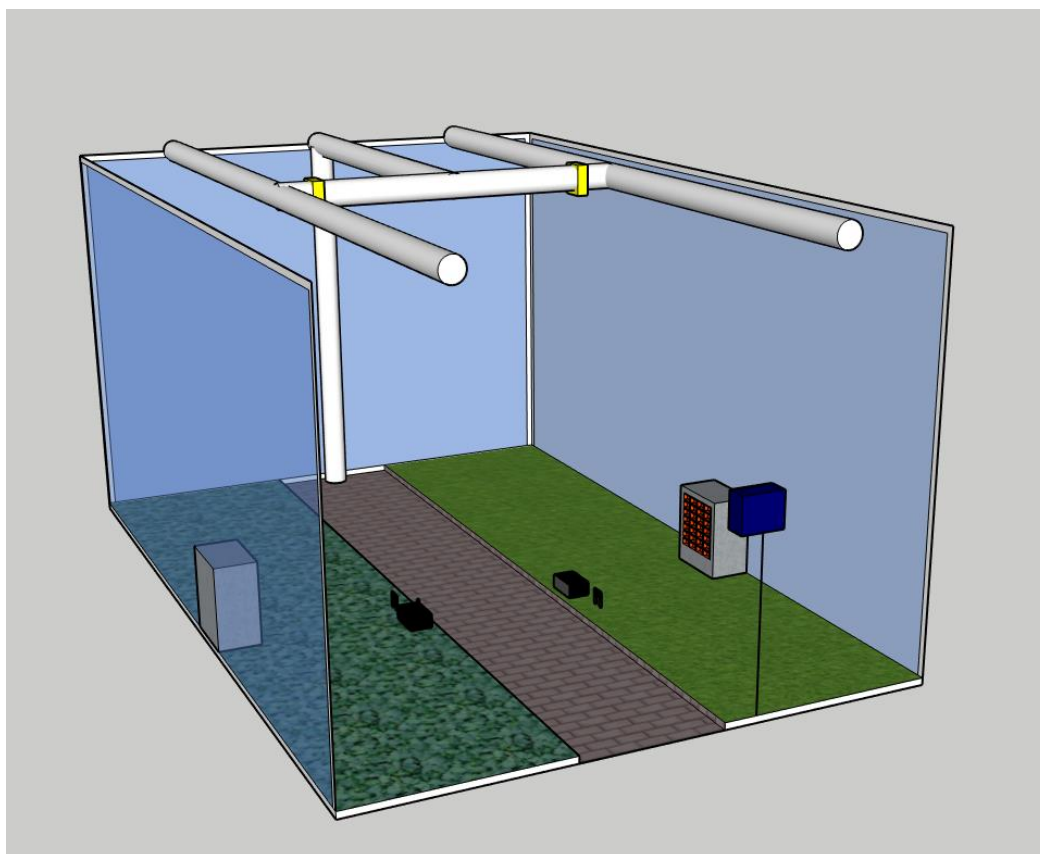
A placa de desenvolvimento Arduino UNO, utiliza o microcontrolador ATMEGA328P da *Atmel*®. Ela possui uma porta USB que pode ser utilizada através de um cabo, possui também entrada para alimentação de 6 a 20V em corrente contínua.

Um layout da placa com suas pinagens é apresentado no ANEXO A . Para o circuito em questão, foram utilizados os pinos SCL e SDA para o protocolo de comunicação I²C entre o Arduino UNO e os módulos. O pino de GND também é interligado ao negativo da alimentação, para que todos os componentes do circuito tenham a mesma referência de potencial elétrico. Os terminais 2, 3, 4, 5, 6 e 7, são utilizados para o acionamento dos aquecedores instalados em locais próximos aos seus respectivos módulos. Devido à alta corrente e tensão demandada pelos aquecedores, são utilizados relés como interface.

3.7 LAYOUT DA ESTUFA

O objetivo do circuito controlador, é que ele seja versátil e possa ser utilizado em vários tipos e modelos de estufas diferentes, utilizando um número maior ou menor de módulos remotos de acordo com as necessidades. Porém para título de demonstração, esta secção apresenta um dos layouts possíveis de aplicação do sistema.

Figura 12 - Modelo de layout para estufa.



Fonte: Autor (2020).

Na Figura 12, é apresentado um layout sugerido, utilizando dois módulos independentes, criando assim a possibilidade de cultivo de dois tipos distintos de plantas, com necessidade de temperatura e umidade diferentes, isto é possível criando áreas específicas dentro da mesma, isoladas com paredes de vidro. A estrutura montada na parte superior, é a tubulação responsável pela irrigação em forma de “chuva” que servirá como controle da umidade do ar. Em amarelo, são as duas válvulas (uma para cada módulo) que permitem ou bloqueiam o fluxo de água pela tubulação. Em cinza são os aquecedores, responsáveis pelo controle de temperatura. A caixa em azul à direita, é o controlador principal (onde se encontra o Arduino UNO). As duas caixas pretas, uma à esquerda e outra à direita do corredor, são os módulos remotos, onde estão alojados os microcontroladores ATtiny85.

Este layout leva em conta uma estufa com dimensões 4 X 6m, totalizando 24m², e utilizando-se dois módulos remotos (como já abordado anteriormente, o sistema suporta até 6

módulos/escravos simultâneos). Porém, devido a sua versatilidade pode ser instalado em estufas muito menores, um pouco maiores que um aquário por exemplo, que poderão ficar em uma sala de um apartamento.

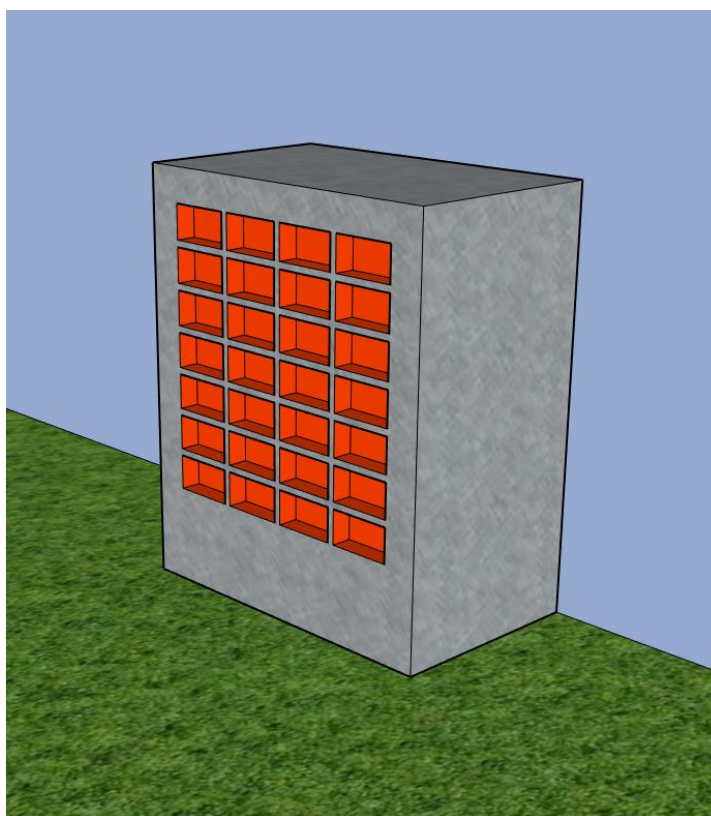
3.8 RESPOSTA DO SISTEMA

Como o tamanho físico de uma estufa é relativamente grande, e os componentes a serem controlados (temperatura e umidade) não mudam seus valores de forma abrupta, não é necessário que o controlador tenha uma resposta extremamente rápida, por isso a ação de controle adotada foi um simples “ligado-desligado”, pois a própria dinâmica do código de programação responde lentamente, gerando uma janela de atuação, evitando assim que os atuadores fiquem ligando e desligando a todo momento, mesmo próximo do limiar de operação.

3.9 CONTROLE DE TEMPERATURA

O controle de temperatura se dará através de aquecedores específicos para ambientes de estufas. A Figura 13 ilustra um esboço do aquecedor que pode ser utilizado. Além dos resistores para aquecimento, este equipamento conta com ventoinhas para a circulação de ar quente, aumentando a área de abrangência.

Figura 13 - Modelo 3D do aquecedor.

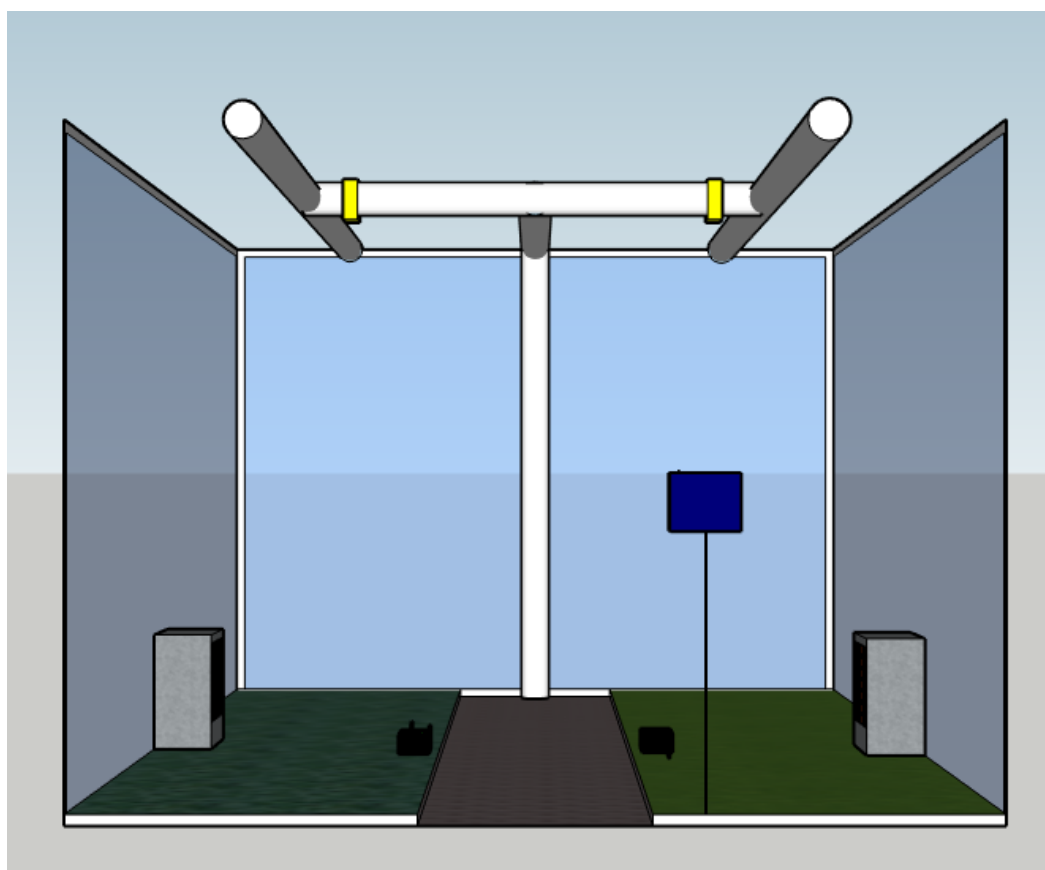


Fonte: Autor (2020).

3.10 CONTROLE DE UMIDADE DO AR

Para a regulação da umidade de ar no ambiente da estufa, adotou-se o emprego de tubulações de água, que tem seu fluxo controlado através de válvulas. No caso apresentado da figura abaixo, existe um cano central de onde vem o fluxo de água, há então uma conexão tipo “T” que o divide para que sejam criadas duas áreas de irrigação, uma à esquerda e outra à direita.

Figura 14 - Disposição da tubulação de irrigação.



Fonte: Autor (2020).

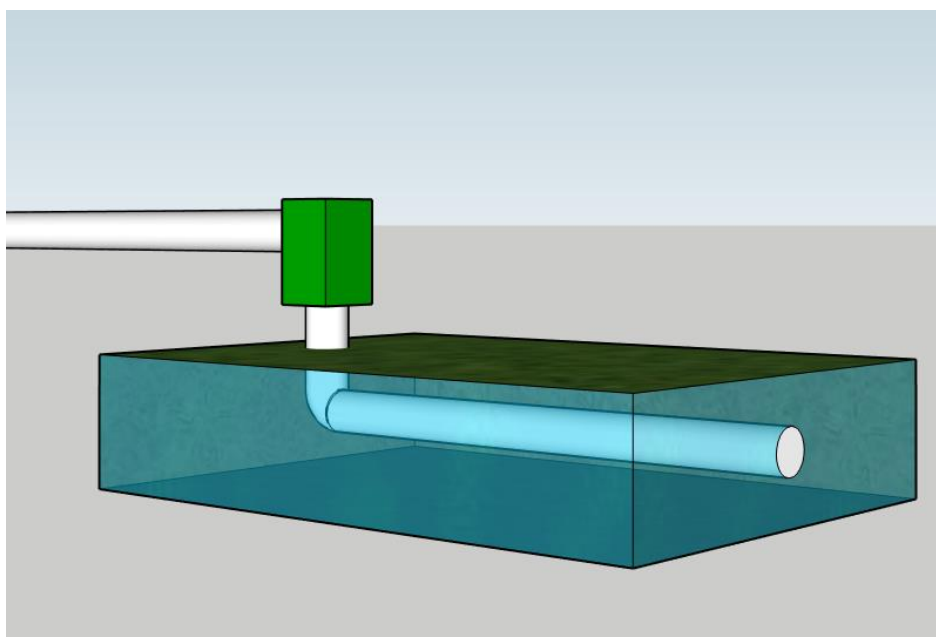
O fluxo de água parte do solo pela tubulação (em branco), em direção ao teto da estufa, e após isso, é liberado ou interrompido pelas válvulas (destacadas em amarelo na figura).

3.11 CONTROLE DE UMIDADE DO SOLO

Como algumas plantas podem ter necessidades distintas para umidade do ar e do solo, é utilizado um segundo sistema de irrigação, enquanto o primeiro apresentava uma irrigação geral, umidificando todo o ambiente, este segundo método é muito mais direcionado, promovendo uma irrigação direta nas raízes das plantas, para tal, uma mangueira perfurada é

enterrada no solo e logo acima (em verde) é instalada a válvula de controle de fluxo, conforme a ilustração abaixo:

Figura 15 - Vista em corte do método de irrigação de solo.



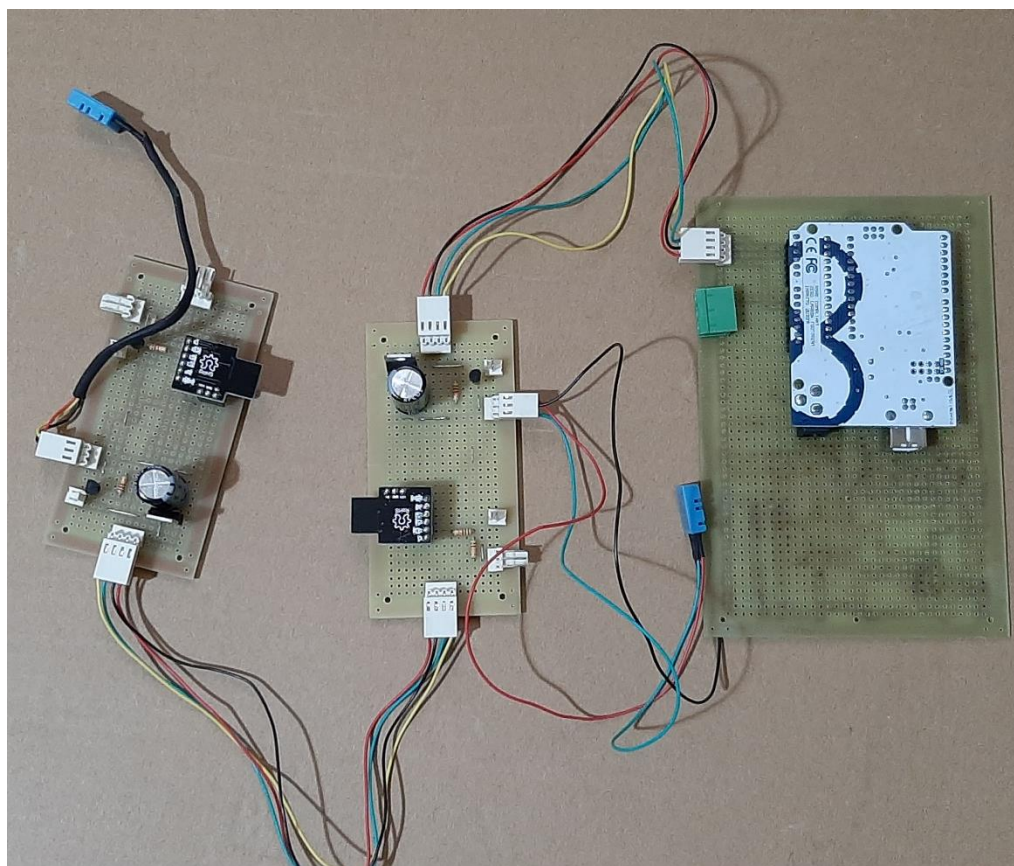
Fonte: Autor (2020).

Utilizando-se este método, é possível criar canais de irrigação independentes provenientes por exemplo de tanques, que podem conter determinados tipos de nutrientes, sais minerais etc. cada um específico para o melhor desenvolvimento da espécie cultivada.

4 CONCLUSÃO E RECOMENDAÇÕES

Após a finalização dos algoritmos e testes preliminares em protoboard, foi realizada a montagem do circuito em placa de fenolite ilhada, a fim de diminuir os possíveis ruídos e mau-contatos. O aspecto final da montagem está na figura abaixo.

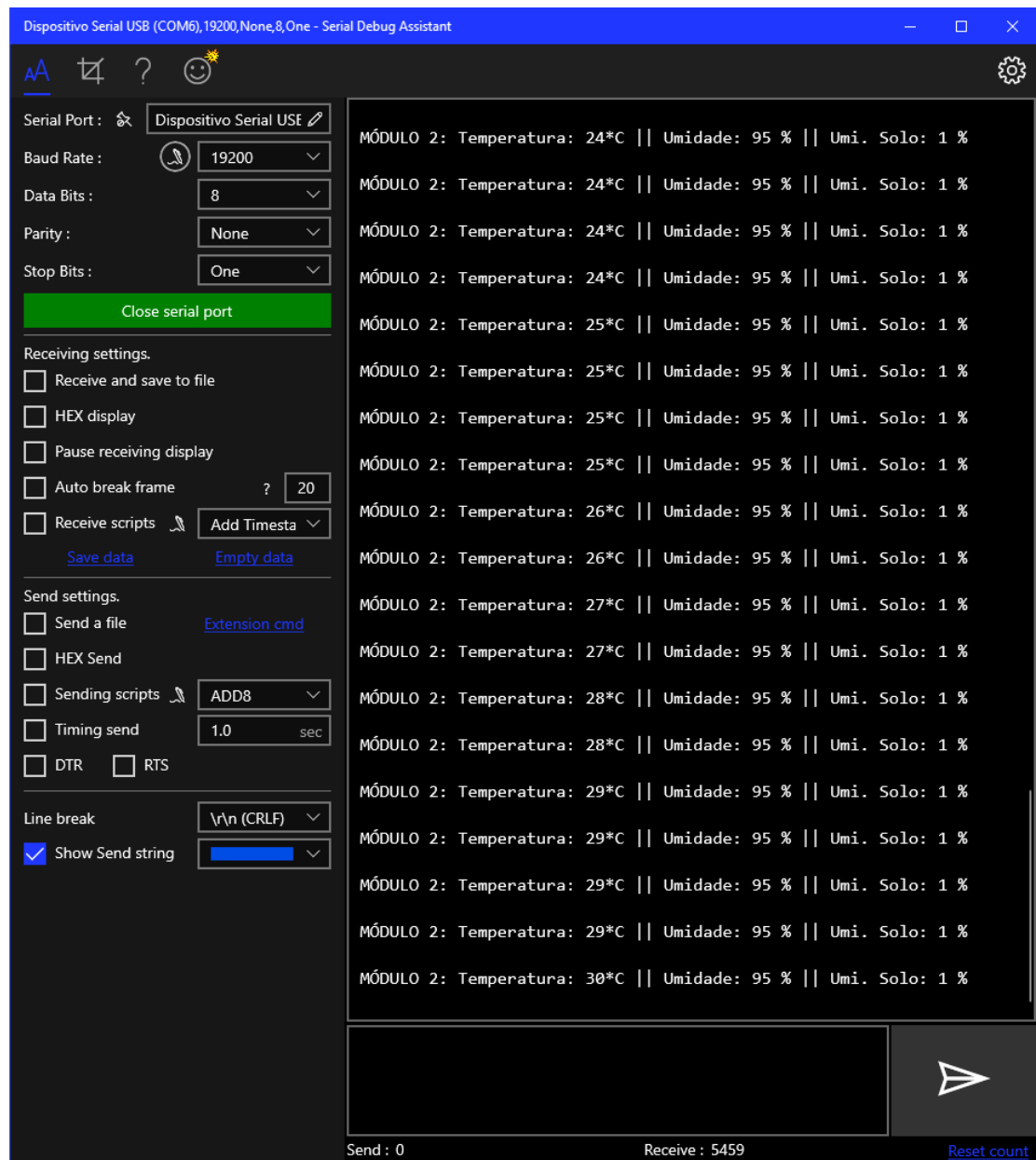
Figura 16 - Aspecto final da montagem.



Fonte: Autor (2020).

Com o circuito montado, foi possível realizar os testes e constatar que o sistema respondeu conforme o esperado, quando os parâmetros configurados eram atingidos, as funções de controle iniciavam a atuação ou cessavam, de acordo com as necessidades. Como as variáveis a serem controladas são de temperatura e umidade, elas demandam um período relativamente longo para variar, desta forma, a resposta do sistema se apresentou estável nas aplicações práticas também, pois não houve ligamentos e desligamentos rápidos e/ou intermitentes, que pudessem causar danos aos componentes eletroeletrônicos.

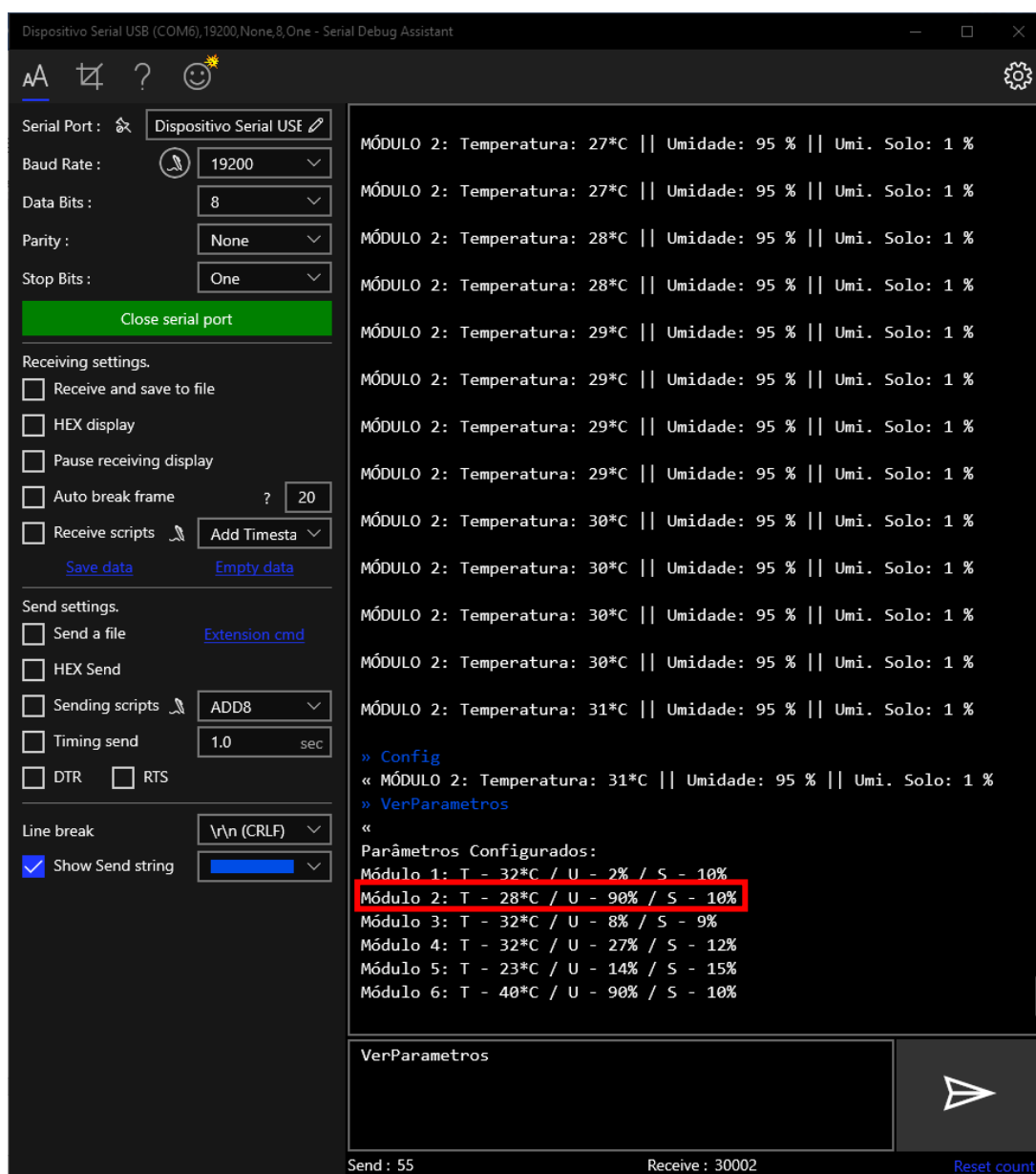
Figura 17 - Leitura de temperatura variando após aquecimento.



Fonte: Autor (2020).

A Figura 17, demonstra a variação de temperatura lida pelo Módulo 2, variando proporcionalmente com seu aumento. A ação de controle é executada na temperatura de 28°C, conforme parametrizado na Figura 18.

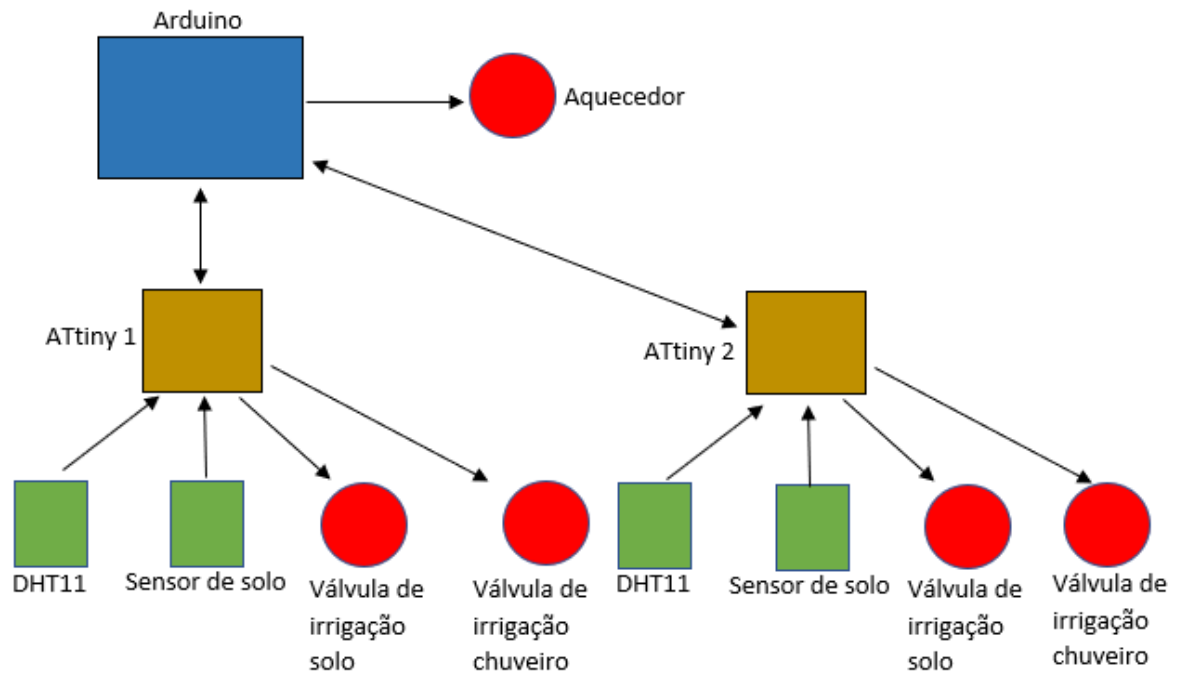
Figura 18 - Valores parametrizados para o Módulo 2.



Como recomendações de melhorias, seria possível integrar o sistema à rede de internet, tornando o Arduino UNO um *webservice*, o que possibilitaria o acompanhamento e configuração de parâmetros de maneira remota.

REFERÊNCIAS

- MOREIRA, Rosa. **Tipos de estufas para a agricultura: tipos de geometrias ou configuração.** A Cientista Agrícola. 2019. Disponível em: <https://acientistaagricola.pt/tipos-de-estufas-consoante-a-forma/>. Acesso em 06 jun. 2020.
- HEWITT, Paul G.. **Fundamentos da Física Conceitual.** Ed. Bookman. 2019
- ATTENBOROUGH, David. **David Attenborough e Nosso Planeta.** Netflix 2020.
- BRITANNICA ESCOLA. **Planta.** Disponível em: <https://escola.britannica.com.br/artigo/planta/482227>. Acesso em 11 jun. 2020.
- HOROWITZ, Paul. **A arte da eletrônica: circuitos eletrônicos e microeletrônica.** 3ª ed. Editora Bookman – Porto Alegre, RS. 2017.
- BAUERMEISTER, Giovanni. **Primeiros passos com Arduino.** Blog FELIPEFLOP. 2018. Disponível em: <https://www.filipeflop.com/blog/primeiros-passos-com-arduino/>. Acesso em 11 jun. 2020.
- THOMSEN, Adilson. **Monitorando Temperatura e Umidade com o sensor DHT11.** Blog FELIPEFLOP. 2013. Disponível em: <https://www.filipeflop.com/blog/monitorando-temperatura-e-umidade-com-o-sensor-dht11/>. Acesso em 09 maio 2020.
- THOMSEN, Adilson. **Monitore sua planta usando Arduino.** Blog FELIPEFLOP. 2018. Disponível em: <https://www.filipeflop.com/blog/monitore-sua-planta-usando-arduino/>. Acesso em 11 jun. 2020.
- HIDRAUTEC Equipamentos Hidráulicos. **Válvulas Hidráulicas.** Disponível em: <https://www.hidrautec.com.br/valvulas-hidraulicas>. Acesso em: 11 jun. 2020.
- 3D WAHREHOUSE. Sketchup. Disponível em: <https://3dwarehouse.sketchup.com/user/81736696-2904-42e4-be9b-4df9283661bc/Hermes>. Acesso em 27 jun. 2020.
- ARDUINO E CIA. **Como programar a Digispark ATtiny85 com IDE Arduino.** Arduino e Cia, 2016. Disponível em: <https://www.arduinoecia.com.br/digispark-attiny85-ide-arduino/>. Acesso em 16 de maio 2020.

APÊNDICE A – DIAGRAMA DO CIRCUITO

APÊNDICE B – CÓDIGO DE PROGRAMAÇÃO DO MESTRE

Abaixo encontra-se o código desenvolvido na IDE do Arduino para a placa Arduino UNO que foi utilizado na função “mestre” no protocolo de comunicação I²C (mestre-escravo).

```

001.#include <EEPROM.h> // inclui a biblioteca EEPROM
002.#define espacoEEPROM 1000 // define o espaço de 1000 bytes para a
EEPROM
003.
004.void EEPROMWriteInt(int address, int value); // Função para gravar na
EEPROM (ocupa 2 Bytes)
005.int EEPROMReadInt(int address); // Função para leitura do valor na
EEPROM
006.
007.String textoRecebido = ""; // cria um campo do tipo string para leitura
do terminal serial
008.unsigned long delay1 = 0; // cria um delay para leitura do terminal
009.
010.#include <Wire.h> // inclui a biblioteca para comunicação I2C
011.
012.int setUmiAr1; // cria a variável que armazena valor de umidade do ar
para o módulo remoto 1
013.int setUmiSolo1; // cria a variável que armazena valor de umidade do
solo para o módulo remoto 1
014.int setTemp1; // cria a variável que armazena valor de temperatura para
o módulo remoto 1
015.
016.int setUmiAr2; // cria a variável que armazena valor de umidade do ar
para o módulo remoto 2
017.int setUmiSolo2; // cria a variável que armazena valor de umidade do
solo para o módulo remoto 2
018.int setTemp2; // cria a variável que armazena valor de temperatura para
o módulo remoto 2
019.
020.int setUmiAr3; // cria a variável que armazena valor de umidade do ar
para o módulo remoto 3
021.int setUmiSolo3; // cria a variável que armazena valor de umidade do
solo para o módulo remoto 3
022.int setTemp3; // cria a variável que armazena valor de temperatura para
o módulo remoto 3
023.
024.int setUmiAr4; // cria a variável que armazena valor de umidade do ar
para o módulo remoto 4
025.int setUmiSolo4; // cria a variável que armazena valor de umidade do
solo para o módulo remoto 4
026.int setTemp4; // cria a variável que armazena valor de temperatura para
o módulo remoto 4
027.
028.int setUmiAr5; // cria a variável que armazena valor de umidade do ar
para o módulo remoto 5
029.int setUmiSolo5; // cria a variável que armazena valor de umidade do
solo para o módulo remoto 5
030.int setTemp5; // cria a variável que armazena valor de temperatura para
o módulo remoto 5
031.
032.int setUmiAr6; // cria a variável que armazena valor de umidade do ar
para o módulo remoto 6
033.int setUmiSolo6; // cria a variável que armazena valor de umidade do
solo para o módulo remoto 6

```



```

034.int setTemp6; // cria a variável que armazena valor de temperatura para
o módulo remoto 6
035.
036.#define T1 2 // define a porta digital 2 como T1
037.#define T2 3 // define a porta digital 3 como T2
038.#define T3 4 // define a porta digital 4 como T3
039.#define T4 5 // define a porta digital 5 como T4
040.#define T5 6 // define a porta digital 6 como T5
041.#define T6 7 // define a porta digital 7 como T6
042.
043.int endereco; // cria variável que armazena endereço do escravo
044.
045.void setup() // loop de configurações
046.{
047.  Wire.begin(); // inicia a comunicação I²C
048.  Serial.begin(19200); // inicia a porta serial
049.  endereco = 0; // define valor inicial do endereço do escravo para ser
incrementado na função da linha "80"
050.
051.  setTemp1 = EEPROMReadInt(1); // reserva os bytes 1 e 2 para o valor
desejado de temperatura do módulo 1
052.  setUmiAr1 = EEPROMReadInt(3); // reserva os bytes 3 e 4 para o valor
desejado de umidade do ar do módulo 1
053.  setUmiSolo1 = EEPROMReadInt(5); // reserva os bytes 5 e 6 para o
valor desejado de umidade do solo do módulo 1
054.  setTemp2 = EEPROMReadInt(7); // reserva os bytes 7 e 8 para o valor
desejado de temperatura do módulo 2
055.  setUmiAr2 = EEPROMReadInt(9); // reserva os bytes 9 e 10 para o valor
desejado de umidade do ar do módulo 2
056.  setUmiSolo2 = EEPROMReadInt(11); // reserva os bytes 11 e 12 para o
valor desejado de umidade do solo do módulo 2
057.  setTemp3 = EEPROMReadInt(13); // reserva os bytes 13 e 14 para o
valor desejado de temperatura do módulo 3
058.  setUmiAr3 = EEPROMReadInt(15); // reserva os bytes 15 e 16 para o
valor desejado de umidade do ar do módulo 3
059.  setUmiSolo3 = EEPROMReadInt(17); // reserva os bytes 17 e 18 para o
valor desejado de umidade do solo do módulo 3
060.  setTemp4 = EEPROMReadInt(19); // reserva os bytes 19 e 20 para o
valor desejado de temperatura do módulo 4
061.  setUmiAr4 = EEPROMReadInt(21); // reserva os bytes 21 e 22 para o
valor desejado de umidade do ar do módulo 4
062.  setUmiSolo4 = EEPROMReadInt(23); // reserva os bytes 23 e 24 para o
valor desejado de umidade do solo do módulo 4
063.  setTemp5 = EEPROMReadInt(25); // reserva os bytes 25 e 26 para o
valor desejado de temperatura do módulo 5
064.  setUmiAr5 = EEPROMReadInt(27); // reserva os bytes 27 e 28 para o
valor desejado de umidade do ar do módulo 5
065.  setUmiSolo5 = EEPROMReadInt(29); // reserva os bytes 29 e 30 para o
valor desejado de umidade do solo do módulo 5
066.  setTemp6 = EEPROMReadInt(31); // reserva os bytes 31 e 32 para o
valor desejado de temperatura do módulo 6
067.  setUmiAr6 = EEPROMReadInt(33); // reserva os bytes 33 e 34 para o
valor desejado de umidade do ar do módulo 6
068.  setUmiSolo6 = EEPROMReadInt(35); // reserva os bytes 35 e 36 para o
valor desejado de umidade do solo do módulo 6
069.
070.  delay(200); // delay de 200 milissegundos
071.
072.  pinMode(LED_BUILTIN, OUTPUT); // define o pino 13 como saída
073.  pinMode(T1, OUTPUT); // define o pino T1 como saída
074.  pinMode(T2, OUTPUT); // define o pino T2 como saída

```

```

075. pinMode(T3, OUTPUT); // define o pino T3 como saída
076. pinMode(T4, OUTPUT); // define o pino T4 como saída
077. pinMode(T5, OUTPUT); // define o pino T5 como saída
078. pinMode(T6, OUTPUT); // define o pino T6 como saída
079.
080. while (endereco < 6) { // enquanto o valor do endereço for menor que
6
081.     endereco = endereco + 1; // ao valor do endereço é somado 1
(variável é zerada na linha "49")
082.
083.     Wire.requestFrom(endereco, 3); // solicita via I²C ao endereço do
escravo 3 bytes de informação
084.     if (Wire.available()) { // caso o módulo esteja disponível
085.         Serial.print("Módulo "); // escreve na porta serial
086.         Serial.print(endereco); // escreve o endereço do módulo que
respondeu ao mestre
087.         Serial.println(" OK"); // escreve "OK" na porta serial e pula uma
linha, indicando que o módulo está ativo
088.     } else { // senão, se o módulo requisitado não responder
089.         Serial.print("Módulo "); // escreve na porta serial
090.         Serial.print(endereco); // escreve o endereço do módulo que não
respondeu ao mestre
091.         Serial.println(" Não Conectado"); // escreve "Não conectado" na
porta serial e pula uma linha, indicando que o módulo não está ativo
092.     }
093. }
094. Serial.println("-----"); // escreve na porta serial uma
linha de separação
095. Serial.println("Inicializando"); // escreve "Inicializando" na porta
serial
096. delay(2000); // delay de 2000 milissegundos
097. Serial.println(" "); // escreve um campo vazio no monitor serial e
pula uma linha
098.}
099.
100.void loop() // loop principal
101.{
102.
103.     endereco = 0; // inicia com endereço 0 do escravo para ser
incrementado na função da linha "110"
104.     Serial.println(" "); // pula uma linha no monitor serial
105.     int setUmiAr[7] = {0, setUmiAr1, setUmiAr2, setUmiAr3, setUmiAr4,
setUmiAr5, setUmiAr6}; // cria um array para umidade do ar
106.     int setUmiSolo[7] = {0, setUmiSolo1, setUmiSolo2, setUmiSolo3,
setUmiSolo4, setUmiSolo5, setUmiSolo6}; // cria um array para umidade do
solo
107.     int setTemp[7] = {0, setTemp1, setTemp2, setTemp3, setTemp4,
setTemp5, setTemp6}; // cria um array para temperatura
108.     int porta[7] = {0, T1, T2, T3, T4, T5, T6}; // cria um array para as
ports de controle de temperatura
109.
110.     while (endereco < 6) { // enquanto o valor do endereço for menor que 6
111.         endereco = endereco + 1; // ao valor do endereço é somado 1
(variável é zerada na linha "103")
112.
113.         Wire.requestFrom(endereco, 3); // solicita via I²C ao endereço do
escravo 3 bytes de informação
114.
115.         while (Wire.available()) { // enquanto o módulo esteja disponível
116.             int t = Wire.read(); // variável "t" recebe o valor do primeiro
byte (temperatura lida pelo DHT11)

```

```

117.     int h = Wire.read(); // variável "h" recebe o valor do segundo
byte (umidade do ar lida pelo DHT11)
118.     int umiSolo = Wire.read(); // variável "umiSolo" recebe o valor
do terceiro byte (umidade do solo lida pelo sensor de solo)
119.     umiSolo = map(umiSolo, 0, 255, 100, 0); // função que condiciona
o valor bruto (0 a 255) em valor percentual (0 a 100%)
120.     Serial.print("MÓDULO "); // escreve na porta serial
121.     Serial.print(endereco); // escreve o endereço do módulo
122.     Serial.print(": Temperatura: "); // escreve na porta serial
123.     Serial.print(t); // escreve o valor de temperatura recebido
124.     Serial.print("*C |"); // escreve na porta serial
125.     Serial.print("| Umidade: "); // escreve na porta serial
126.     Serial.print(h); // escreve o valor de umidade do ar recebido
127.     Serial.print(" % |"); // escreve na porta serial
128.     Serial.print("| Umi. Solo: "); // escreve na porta serial
129.     Serial.print(umiSolo); // recebe o valor de umidade de solo
recebido
130.     Serial.println(" %"); // escreve na porta serial e pula uma linha
131.
132.     if ( h >= setUmiAr[endereco] && umiSolo >= setUmiSolo[endereco])
{ // se o valor de umidade de ar lido for maior ou igual ao parametrizado,
e, o valor da umidade do solo for maior ou igual ao parametrizado
133.         Wire.beginTransmission(endereco); // inicia transmissão via I²C
ao escravo com endereço
134.         Wire.write(0); // envia o valor 0 ao escravo (00 em binário,
pinos 1 e 4 do módulo desligados)
135.         Wire.endTransmission(); // finaliza a transmissão
136.     }
137.
138.     if ( h < setUmiAr[endereco] && umiSolo >= setUmiSolo[endereco]) {
// se o valor de umidade de ar lido for menor que parametrizado, e, o valor
da umidade do solo for maior ou igual ao parametrizado
139.         Wire.beginTransmission(endereco); // inicia transmissão via I²C
ao escravo com endereço
140.         Wire.write(1); // envia o valor 1 ao escravo (01 em binário,
pino 1 do módulo ligado e pino 4 desligado)
141.         Wire.endTransmission(); // finaliza a transmissão
142.     }
143.
144.     if ( h >= setUmiAr[endereco] && umiSolo < setUmiSolo[endereco]) {
// se o valor de umidade de ar lido for maior ou igual ao parametrizado, e,
o valor da umidade do solo for menor que parametrizado
145.         Wire.beginTransmission(endereco); // inicia transmissão via I²C
ao escravo com endereço
146.         Wire.write(2); // envia o valor 2 ao escravo (10 em binário,
pino 1 do módulo desligado e pino 4 ligado)
147.         Wire.endTransmission(); // finaliza a transmissão
148.     }
149.
150.     if ( h < setUmiAr[endereco] && umiSolo < setUmiSolo[endereco]) {
// se o valor de umidade de ar lido for menor que parametrizado, e, o valor
da umidade do solo for menor que parametrizado
151.         Wire.beginTransmission(endereco); // inicia transmissão via I²C
ao escravo com endereço
152.         Wire.write(3); // envia o valor 3 ao escravo (11 em binário,
pinos 1 e 4 do módulo ligados)
153.         Wire.endTransmission(); // finaliza a transmissão
154.     }
155.     if (t < setTemp[endereco]) { // se a temperatura lida for menor
do que o parametrizado

```

```

156.         digitalWrite(porta[endereco], HIGH); // porta digital do módulo
é ativada
157.     } else { // senão
158.         digitalWrite(porta[endereco], LOW); // porta digital é
desativada
159.     }
160.     delay(100); // delay de 100 milissegundos
161. }
162.
163. // ----- Entrada de Parâmetros -----
164.
165. char caracter; // cria uma variável do tipo caractere
166.
167. if (Serial.available()) { // se a porta serial estiver disponível
168.     caracter = Serial.read(); // variável "caracter" recebe o que foi
digitado no monitor serial
169.     textoRecebido += caracter; // "textoRecebido" recebe a informação
de "caracter"
170.     delay1 = millis(); // delay
171. }
172.
173. if (((millis() - delay1) > 10) && (textoRecebido != "")) { // caso
o delay for maior que 10 milissegundos e texto recebido for diferente de ""
(vazio)
174.     textoRecebido = ""; // "textoRecebido" recebe "" (vazio) para
iniciar nova leitura
175. }
176.
177. // *****
178.
179. if (textoRecebido == "Config") { // se "textoRecebido" é igual a
"Config" (entra no modo de configuração de parâmetros de temperatura e
umidade desejados
180.     while (textoRecebido != "FimConfig") { // enquanto
"textoRecebido" for diferente de "FimConfig"
181.         digitalWrite(LED_BUILTIN, HIGH); // pino 13 fica em nível alto
(LED configurado para ser indicativo da função de configuração de
parâmetros)
182.         if (Serial.available()) { // se a porta serial estiver
disponível
183.             caracter = Serial.read(); // variável "caracter" recebe o que
foi digitado no monitor serial
184.             textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
185.             delay1 = millis(); // delay
186.         }
187.
188.         if (((millis() - delay1) > 10) && (textoRecebido != "")) { //
caso o delay for maior que 10 milissegundos e texto recebido for diferente
de "" (vazio)
189.             textoRecebido = ""; // "textoRecebido" recebe "" (vazio) para
iniciar nova leitura
190.         }
191.         // -----
192.         if (textoRecebido == "TM1") { // se "textoRecebido" é igual a
"TM1"
193.             while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
194.                 if (Serial.available()) { // se a porta serial estiver
disponível

```

```

195.          caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
196.          textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
197.          delay1 = millis(); // delay
198.      }
199.
200.      if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
201.          Serial.println(" "); // pula uma linha no monitor serial
202.          Serial.print("Temperatura Módulo 1: "); // escreve no
monitor serial
203.          setTemp1 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
204.          Serial.println(setTemp1); // porta serial escreve valor
atribuído à variável
205.          EEPROMWriteInt(1, setTemp1); // grava o valor atribuído
na EEPROM
206.          textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
207.      }
208.  }
209.  }
210.  // -----
211.  if (textoRecebido == "UM1") { // se "textoRecebido" é igual a
"UM1"
212.      while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
213.          if (Serial.available()) { // se a porta serial estiver
disponível
214.              caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
215.              textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
216.              delay1 = millis(); // delay
217.          }
218.
219.          if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
220.              Serial.println(" "); // pula uma linha no monitor serial
221.              Serial.print("Umidade Módulo 1: "); // escreve no monitor
serial
222.              setUmiAr1 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
223.              Serial.println(setUmiAr1); // porta serial escreve valor
atribuído à variável
224.              EEPROMWriteInt(3, setUmiAr1); // grava o valor atribuído
na EEPROM
225.              textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
226.          }
227.      }
228.  }
229.  // -----
230.  if (textoRecebido == "SM1") { // se "textoRecebido" é igual a
"SM1"
231.      while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"

```

```

232.         if (Serial.available()) { // se a porta serial estiver
disponível
233.             caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
234.             textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
235.             delay1 = millis(); // delay
236.         }
237.
238.         if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
239.             Serial.println(" "); // pula uma linha no monitor serial
240.             Serial.print("Umidade de Solo Módulo 1: "); // escreve no
monitor serial
241.             setUmiSolo1 = textoRecebido.toInt(); // converte a
variável caracter em valor inteiro
242.             Serial.println(setUmiSolo1); // porta serial escreve
valor atribuído à variável
243.             EEPROMWriteInt(5, setUmiSolo1); // grava o valor
atribuído na EEPROM
244.             textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
245.         }
246.     }
247. }
248. // -----
249.     if (textoRecebido == "TM2") { // se "textoRecebido" é igual a
"TM2"
250.         while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
251.             if (Serial.available()) { // se a porta serial estiver
disponível
252.                 caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
253.                 textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
254.                 delay1 = millis(); // delay
255.             }
256.
257.             if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
258.                 Serial.println(" "); // pula uma linha no monitor serial
259.                 Serial.print("Temperatura Módulo 2: "); // escreve no
monitor serial
260.                 setTemp2 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
261.                 Serial.println(setTemp2); // porta serial escreve valor
atribuído à variável
262.                 EEPROMWriteInt(7, setTemp2); // grava o valor atribuído
na EEPROM
263.                 textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
264.             }
265.         }
266.     }
267. // -----
268.     if (textoRecebido == "UM2") { // se "textoRecebido" é igual a
"UM2"

```

```

269.         while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
270.             if (Serial.available()) { // se a porta serial estiver
disponível
271.                 caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
272.                 textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
273.                 delay1 = millis(); // delay
274.             }
275.
276.             if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
277.                 Serial.println(" "); // pula uma linha no monitor serial
278.                 Serial.print("Umidade Módulo 2: "); // escreve no monitor
serial
279.                 setUmiAr2 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
280.                 Serial.println(setUmiAr2); // porta serial escreve valor
atribuído à variável
281.                 EEPROMWriteInt(9, setUmiAr2); // grava o valor atribuído
na EEPROM
282.                 textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
283.             }
284.         }
285.     }
286.     // -----
287.     if (textoRecebido == "SM2") { // se "textoRecebido" é igual a
"SM2"
288.         while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
289.             if (Serial.available()) { // se a porta serial estiver
disponível
290.                 caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
291.                 textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
292.                 delay1 = millis(); // delay
293.             }
294.
295.             if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
296.                 Serial.println(" "); // pula uma linha no monitor serial
297.                 Serial.print("Umidade de Solo Módulo 2: "); // escreve no
monitor serial
298.                 setUmiSolo2 = textoRecebido.toInt(); // converte a
variável caracter em valor inteiro
299.                 Serial.println(setUmiSolo2); // porta serial escreve
valor atribuído à variável
300.                 EEPROMWriteInt(11, setUmiSolo2); // grava o valor
atribuído na EEPROM
301.                 textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
302.             }
303.         }
304.     }
305.
306.     // -----

```

```

307.         if (textoRecebido == "TM3") { // se "textoRecebido" é igual a
"TM3"
308.             while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
309.                 if (Serial.available()) { // se a porta serial estiver
disponível
310.                     caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
311.                     textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
312.                     delay1 = millis(); // delay
313.                 }
314.
315.                 if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
316.                     Serial.println(" "); // pula uma linha no monitor serial
317.                     Serial.print("Temperatura Módulo 3: "); // escreve no
monitor serial
318.                     setTemp3 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
319.                     Serial.println(setTemp3); // porta serial escreve valor
atribuído à variável
320.                     EEPROMWriteInt(13, setTemp3); // grava o valor atribuído
na EEPROM
321.                     textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
322.                 }
323.             }
324.         }
325.         // -----
326.         if (textoRecebido == "UM3") { // se "textoRecebido" é igual a
"UM3"
327.             while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
328.                 if (Serial.available()) { // se a porta serial estiver
disponível
329.                     caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
330.                     textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
331.                     delay1 = millis(); // delay
332.                 }
333.
334.                 if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
335.                     Serial.println(" "); // pula uma linha no monitor serial
336.                     Serial.print("Umidade Módulo 3: "); // escreve no monitor
serial
337.                     setUmiAr3 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
338.                     Serial.println(setUmiAr3); // porta serial escreve valor
atribuído à variável
339.                     EEPROMWriteInt(15, setUmiAr3); // grava o valor atribuído
na EEPROM
340.                     textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
341.                 }
342.             }
343.         }

```



```

344.          // -----
345.          if (textoRecebido == "SM3") { // se "textoRecebido" é igual a
"SM3"
346.              while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
347.                  if (Serial.available()) { // se a porta serial estiver
disponível
348.                      caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
349.                      textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
350.                      delay1 = millis(); // delay
351.                  }
352.
353.                  if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
354.                      Serial.println(" "); // pula uma linha no monitor serial
355.                      Serial.print("Umidade de Solo Módulo 3: "); // escreve no
monitor serial
356.                      setUmiSolo3 = textoRecebido.toInt(); // converte a
variável caracter em valor inteiro
357.                      Serial.println(setUmiSolo3); // porta serial escreve
valor atribuído à variável
358.                      EEPROMWriteInt(17, setUmiSolo3); // grava o valor
atribuído na EEPROM
359.                      textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
360.                  }
361.              }
362.          }
363.          // -----
364.          if (textoRecebido == "TM4") { // se "textoRecebido" é igual a
"TM4"
365.              while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
366.                  if (Serial.available()) { // se a porta serial estiver
disponível
367.                      caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
368.                      textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
369.                      delay1 = millis(); // delay
370.                  }
371.
372.                  if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
373.                      Serial.println(" "); // pula uma linha no monitor serial
374.                      Serial.print("Temperatura Módulo 4: "); // escreve no
monitor serial
375.                      setTemp4 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
376.                      Serial.println(setTemp4); // porta serial escreve valor
atribuído à variável
377.                      EEPROMWriteInt(19, setTemp4); // grava o valor atribuído
na EEPROM
378.                      textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
379.                  }
380.              }

```

```

381.         }
382.         // -----
383.         if (textoRecebido == "UM4") { // se "textoRecebido" é igual a
"UM4"
384.             while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
385.                 if (Serial.available()) { // se a porta serial estiver
disponível
386.                     caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
387.                     textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
388.                     delay1 = millis(); // delay
389.                 }
390.
391.                 if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
392.                     Serial.println(" "); // pula uma linha no monitor serial
393.                     Serial.print("Umidade Módulo 4: "); // escreve no monitor
serial
394.                     setUmiAr4 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
395.                     Serial.println(setUmiAr4); // porta serial escreve valor
atribuído à variável
396.                     EEPROMWriteInt(21, setUmiAr4); // grava o valor atribuído
na EEPROM
397.                     textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
398.                 }
399.             }
400.         }
401.         // -----
402.         if (textoRecebido == "SM4") { // se "textoRecebido" é igual a
"SM4"
403.             while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
404.                 if (Serial.available()) { // se a porta serial estiver
disponível
405.                     caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
406.                     textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
407.                     delay1 = millis(); // delay
408.                 }
409.
410.                 if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
411.                     Serial.println(" "); // pula uma linha no monitor serial
412.                     Serial.print("Umidade de Solo Módulo 4: "); // escreve no
monitor serial
413.                     setUmiSolo4 = textoRecebido.toInt(); // converte a
variável caracter em valor inteiro
414.                     Serial.println(setUmiSolo4); // porta serial escreve
valor atribuído à variável
415.                     EEPROMWriteInt(23, setUmiSolo4); // grava o valor
atribuído na EEPROM
416.                     textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
417.                 }

```

```

418.         }
419.     }
420.     // -----
421.     if (textoRecebido == "TM5") { // se "textoRecebido" é igual a
"TM5"
422.         while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
423.             if (Serial.available()) { // se a porta serial estiver
disponível
424.                 caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
425.                 textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
426.                 delay1 = millis(); // delay
427.             }
428.
429.             if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
430.                 Serial.println(" "); // pula uma linha no monitor serial
431.                 Serial.print("Temperatura Módulo 5: "); // escreve no
monitor serial
432.                 setTemp5 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
433.                 Serial.println(setTemp5); // porta serial escreve valor
atribuído à variável
434.                 EEPROMWriteInt(25, setTemp5); // grava o valor atribuído
na EEPROM
435.                 textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
436.             }
437.         }
438.     }
439.     // -----
440.     if (textoRecebido == "UM5") { // se "textoRecebido" é igual a
"UM5"
441.         while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
442.             if (Serial.available()) { // se a porta serial estiver
disponível
443.                 caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
444.                 textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
445.                 delay1 = millis(); // delay
446.             }
447.
448.             if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
449.                 Serial.println(" "); // pula uma linha no monitor serial
450.                 Serial.print("Umidade Módulo 5: "); // escreve no monitor
serial
451.                 setUmiAr5 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
452.                 Serial.println(setUmiAr5); // porta serial escreve valor
atribuído à variável
453.                 EEPROMWriteInt(27, setUmiAr5); // grava o valor atribuído
na EEPROM
454.                 textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura

```

```

455.         }
456.     }
457. }
458. // -----
459. if (textoRecebido == "SM5") { // se "textoRecebido" é igual a
"SM5"
460.     while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
461.         if (Serial.available()) { // se a porta serial estiver
disponível
462.             caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
463.             textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
464.             delay1 = millis(); // delay
465.         }
466.
467.         if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
468.             Serial.println(" "); // pula uma linha no monitor serial
469.             Serial.print("Umidade de Solo Módulo 5: "); // escreve no
monitor serial
470.             setUmiSolo5 = textoRecebido.toInt(); // converte a
variável caracter em valor inteiro
471.             Serial.println(setUmiSolo5); // porta serial escreve
valor atribuído à variável
472.             EEPROMWriteInt(29, setUmiSolo5); // grava o valor
atribuído na EEPROM
473.             textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
474.         }
475.     }
476. }
477. // -----
478. if (textoRecebido == "TM6") { // se "textoRecebido" é igual a
"TM6"
479.     while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
480.         if (Serial.available()) { // se a porta serial estiver
disponível
481.             caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
482.             textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
483.             delay1 = millis(); // delay
484.         }
485.
486.         if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
487.             Serial.println(" "); // pula uma linha no monitor serial
488.             Serial.print("Temperatura Módulo 6: "); // escreve no
monitor serial
489.             setTemp6 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
490.             Serial.println(setTemp6); // porta serial escreve valor
atribuído à variável
491.             EEPROMWriteInt(31, setTemp6); // grava o valor atribuído
na EEPROM

```

```

492.         textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
493.     }
494. }
495. }
496. // -----
497.     if (textoRecebido == "UM6") { // se "textoRecebido" é igual a
"UM6"
498.         while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
499.             if (Serial.available()) { // se a porta serial estiver
disponível
500.                 caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
501.                 textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
502.                 delay1 = millis(); // delay
503.             }
504.
505.             if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
506.                 Serial.println(" "); // pula uma linha no monitor serial
507.                 Serial.print("Umidade Módulo 6: "); // escreve no monitor
serial
508.                 setUmiAr6 = textoRecebido.toInt(); // converte a variável
caracter em valor inteiro
509.                 Serial.println(setUmiAr6); // porta serial escreve valor
atribuído à variável
510.                 EEPROMWriteInt(33, setUmiAr6); // grava o valor atribuído
na EEPROM
511.                 textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
512.             }
513.         }
514.     }
515. // -----
516.     if (textoRecebido == "SM6") { // se "textoRecebido" é igual a
"SM6"
517.         while (textoRecebido != "fim") { // enquanto "textoRecebido"
for diferente de "fim"
518.             if (Serial.available()) { // se a porta serial estiver
disponível
519.                 caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
520.                 textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
521.                 delay1 = millis(); // delay
522.             }
523.
524.             if (((millis() - delay1) > 10) && (textoRecebido != "")) {
// caso o delay for maior que 10 milissegundos e texto recebido for
diferente de "" (vazio)
525.                 Serial.println(" "); // pula uma linha no monitor serial
526.                 Serial.print("Umidade de Solo Módulo 6: "); // escreve no
monitor serial
527.                 setUmiSolo6 = textoRecebido.toInt(); // converte a
variável caracter em valor inteiro
528.                 Serial.println(setUmiSolo6); // porta serial escreve
valor atribuído à variável

```

```

529.          EEPROMWriteInt(35, setUmiSolo6); // grava o valor
atribuído na EEPROM
530.          textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
531.      }
532.  }
533.  }
534.
535.      // -----
536.      if (textoRecebido == "VerParametros") { // se "textoRecebido" é
igual a "VerParametros"
537.          if (Serial.available()) { // se a porta serial estiver
disponível
538.              caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
539.              textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
540.              delay1 = millis(); // delay
541.          }
542.
543.          if (((millis() - delay1) > 10) && (textoRecebido != "")) { //
caso o delay for maior que 10 milissegundos e texto recebido for diferente
de "" (vazio)
544.              Serial.println(" "); // pula uma linha no monitor serial
545.              Serial.println("Parâmetros Configurados: "); // escreve no
monitor serial e pula uma linha
546.              setTemp1 = EEPROMReadInt(1); // variável recebe o valor
guardado na EEPROM
547.              Serial.print("Módulo 1: T - "); // escreve no monitor
serial
548.              Serial.print(setTemp1); // escreve o valor da variável no
monitor serial
549.              Serial.print("*C / U - "); // escreve no monitor serial
550.              setUmiAr1 = EEPROMReadInt(3); // variável recebe o valor
guardado na EEPROM
551.              Serial.print(setUmiAr1); // escreve o valor da variável no
monitor serial
552.              Serial.print("% / S - "); // escreve no monitor serial
553.              setUmiSolo1 = EEPROMReadInt(5); // variável recebe o valor
guardado na EEPROM
554.              Serial.print(setUmiSolo1); // escreve o valor da variável
no monitor serial
555.              Serial.println("%"); // escreve no monitor serial e pula
uma linha
556.              setTemp2 = EEPROMReadInt(7); // variável recebe o valor
guardado na EEPROM
557.              Serial.print("Módulo 2: T - "); // escreve no monitor
serial
558.              Serial.print(setTemp2); // escreve o valor da variável no
monitor serial
559.              Serial.print("*C / U - "); // escreve no monitor serial
560.              setUmiAr2 = EEPROMReadInt(9); // variável recebe o valor
guardado na EEPROM
561.              Serial.print(setUmiAr2); // escreve o valor da variável no
monitor serial
562.              Serial.print("% / S - "); // escreve no monitor serial
563.              setUmiSolo2 = EEPROMReadInt(11); // variável recebe o
valor guardado na EEPROM
564.              Serial.print(setUmiSolo2); // escreve o valor da variável
no monitor serial

```

```

565.          Serial.println("%"); // escreve no monitor serial e pula
uma linha
566.          setTemp3 = EEPROMReadInt(13); // variável recebe o valor
guardado na EEPROM
567.          Serial.print("Módulo 3: T - "); // escreve no monitor
serial
568.          Serial.print(setTemp3); // escreve o valor da variável no
monitor serial
569.          Serial.print("*C / U - "); // escreve no monitor serial
570.          setUmiAr3 = EEPROMReadInt(15); // variável recebe o valor
guardado na EEPROM
571.          Serial.print(setUmiAr3); // escreve o valor da variável no
monitor serial
572.          Serial.print("% / S - "); // escreve no monitor serial
573.          setUmiSolo3 = EEPROMReadInt(17); // variável recebe o
valor guardado na EEPROM
574.          Serial.print(setUmiSolo3); // escreve o valor da variável
no monitor serial
575.          Serial.println("%"); // escreve no monitor serial e pula
uma linha
576.          setTemp4 = EEPROMReadInt(19); // variável recebe o valor
guardado na EEPROM
577.          Serial.print("Módulo 4: T - "); // escreve no monitor
serial
578.          Serial.print(setTemp4); // escreve o valor da variável no
monitor serial
579.          Serial.print("*C / U - "); // escreve no monitor serial
580.          setUmiAr4 = EEPROMReadInt(21); // variável recebe o valor
guardado na EEPROM
581.          Serial.print(setUmiAr4); // escreve o valor da variável no
monitor serial
582.          Serial.print("% / S - "); // escreve no monitor serial
583.          setUmiSolo4 = EEPROMReadInt(23); // variável recebe o
valor guardado na EEPROM
584.          Serial.print(setUmiSolo4); // escreve o valor da variável
no monitor serial
585.          Serial.println("%"); // escreve no monitor serial e pula
uma linha
586.          setTemp5 = EEPROMReadInt(25); // variável recebe o valor
guardado na EEPROM
587.          Serial.print("Módulo 5: T - "); // escreve no monitor
serial
588.          Serial.print(setTemp5); // escreve o valor da variável no
monitor serial
589.          Serial.print("*C / U - "); // escreve no monitor serial
590.          setUmiAr5 = EEPROMReadInt(27); // variável recebe o valor
guardado na EEPROM
591.          Serial.print(setUmiAr5); // escreve o valor da variável no
monitor serial
592.          Serial.print("% / S - "); // escreve no monitor serial
593.          setUmiSolo5 = EEPROMReadInt(29); // variável recebe o
valor guardado na EEPROM
594.          Serial.print(setUmiSolo5); // escreve o valor da variável
no monitor serial
595.          Serial.println("%"); // escreve no monitor serial e pula
uma linha
596.          setTemp6 = EEPROMReadInt(31); // variável recebe o valor
guardado na EEPROM
597.          Serial.print("Módulo 6: T - "); // escreve no monitor
serial

```

```

598.          Serial.print(setTemp6); // escreve o valor da variável no
monitor serial
599.          Serial.print("*C / U - "); // escreve no monitor serial
600.          setUmiAr6 = EEPROMReadInt(33); // variável recebe o valor
guardado na EEPROM
601.          Serial.print(setUmiAr6); // escreve o valor da variável no
monitor serial
602.          Serial.print("% / S - "); // escreve no monitor serial
603.          setUmiSolo6 = EEPROMReadInt(35); // variável recebe o
valor guardado na EEPROM
604.          Serial.print(setUmiSolo6); // escreve o valor da variável
no monitor serial
605.          Serial.println("%"); // escreve no monitor serial e pula
uma linha
606.
607.          textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
608.          }
609.      }
610.      //-----
611.      if (textoRecebido == "Status") { // se "textoRecebido" é igual
a "Status"
612.          if (Serial.available()) { // se a porta serial estiver
disponível
613.              caracter = Serial.read(); // variável "caracter" recebe o
que foi digitado no monitor serial
614.              textoRecebido += caracter; // "textoRecebido" recebe a
informação de "caracter"
615.              delay1 = millis(); // delay
616.          }
617.
618.          if (((millis() - delay1) > 10) && (textoRecebido != "")) { //
caso o delay for maior que 10 milissegundos e texto recebido for diferente
de "" (vazio)
619.              endereco = 0; // variável endereço recebe valor 0
620.              Serial.println(" "); // pula uma linha no monitor serial
621.              while (endereco < 6) { // enquanto endereço menor que 6
622.                  endereco = endereco + 1; // ao valor do endereço é somado
1 (variável é zerada na linha "619")
623.
624.                  Wire.requestFrom(endereco, 3); // solicita via I²C ao
endereço do escravo 3 bytes de informação
625.                  if (Wire.available()) { // caso o módulo esteja
disponível
626.                      Serial.print("Módulo "); // escreve na porta serial
627.                      Serial.print(endereco); // escreve o endereço do módulo
que respondeu ao mestre
628.                      Serial.println(" OK"); // escreve "OK" na porta serial
e pula uma linha, indicando que o módulo está ativo
629.                  } else { // senão, se o módulo requisitado não responder
630.                      Serial.print("Módulo "); // escreve na porta serial
631.                      Serial.print(endereco); // escreve o endereço do módulo
que não respondeu ao mestre
632.                      Serial.println(" Não Conectado"); // escreve "Não
conectado" na porta serial e pula uma linha, indicando que o módulo não
está ativo
633.                  }
634.              }
635.              textoRecebido = ""; // "textoRecebido" recebe "" (vazio)
para iniciar nova leitura
636.          }

```



```

637.     }
638. }
639. }
640. // ----- Fim da entrada de parâmetros -----
641.
642.     digitalWrite(LED_BUILTIN, HIGH); // acende o LED indicador de
funcionamento (código rodando)
643.     delay(100); // delay de 100 milissegundos
644.     digitalWrite(LED_BUILTIN, LOW); //apaga o LED indicador de
funcionamento
645.     delay(100); // delay de 100 milissegundos
646.
647. }
648.}
649.
650.// ----- Loop EEPROM -----
651.
652.void EEPROMWriteInt(int address, int value) { // função para escrita de
variáveis do tipo inteiro na EEPROM
653.    byte hiByte = highByte(value); // maior byte recebe o valor
654.    byte loByte = lowByte(value); // menor byte recebe o valor
655.
656.    EEPROM.write(address, hiByte); // escreve na EEPROM o valor da
variável do maior byte (mais significativo)
657.    EEPROM.write(address + 1, loByte); // escreve na EEPROM um endereço
após o anterior, o valor da variável do menor byte (menos significativo)
658.}
659.
660.int EEPROMReadInt(int address) { // função para leitura de variáveis
do tipo inteiro na EEPROM
661.    byte hiByte = EEPROM.read(address); // maior byte recebe o valor lido
da EEPROM
662.    byte loByte = EEPROM.read(address + 1); // menor byte recebe o valor
do endereço EEPROM após o anterior
663.
664.    return word(hiByte, loByte); // une os dois bytes como "word" e
retorna o valor
665.}

```

APÊNDICE C – CÓDIGO DE PROGRAMAÇÃO DO ESCRAVO

Abaixo encontra-se o código desenvolvido na IDE do Arduino para o módulo ATtiny que foi utilizado na função “escravo” no protocolo de comunicação I²C (mestre-escravo).

```

001.#include <TinyDHT.h> // inclui a biblioteca para o sensor DHT11
002.
003.#define I2C_SLAVE_ADDRESS 0x2 // configura o endereço do escravo
004.
005.#include <TinyWireS.h> // inclui a biblioteca para comunicação I2C
006.
007.
008.#define DHTPIN 3 // define o pino A3 para receber os dados do sensor
DHT11
009.
010.#define DHTTYPE DHT11 // define o tipo de sensor utilizado
011.
012.DHT dht(DHTPIN, DHTTYPE); // seta a função de leitura do sensor DHT de
acordo com as configurações
013.
014.float data; // dado recebido do mestre
015.
016.
017.#define senSolo A0 // define a entrada analógica A0 como sensor de
umidade de solo
018.
019.int umiSolo; // variável que armazena o valor da umidade
020.
021.void setup() // loop de configurações
022.{
023.  TinyWireS.begin(I2C_SLAVE_ADDRESS); // inicia a rede I2C
024.  TinyWireS.onReceive(receiveEvent); // habilita o evento de
recebimento de dados
025.  // TinyWireS.onRequest(requestEvent);
026.
027.  dht.begin(); // inicia a biblioteca DHT
028.
029.  pinMode(1, OUTPUT); // define pino A1 como saída
030.  pinMode(4, OUTPUT); // define pino A4 como saída
031.  digitalWrite(1, LOW); // coloca pino A1 em nível baixo
032.  digitalWrite(4, LOW); // coloca pino A4 em nível baixo
033.}
034.
035.void loop() // loop principal
036.{
037.
038.
039.  TinyWireS_stop_check(); // instrução da biblioteca <TinyWireS.h> para
aguardar informações do mestre
040.
041.  umiSolo = analogRead(senSolo); // faz a leitura de tensão na entrada
analógica A0 e a armazena na variável "umiSolo"
042.
043.  int8_t h = dht.readHumidity(); // instrução da biblioteca <TinyDHT.h>
para leitura da umidade do sensor DHT11 e a armazena na variável "h"
044.  int16_t t = dht.readTemperature(); // instrução da biblioteca
<TinyDHT.h> para leitura da temperatura do sensor DHT11 e a armazena na
variável "t"
045.

```

```
046.
047. TinyWireS.send(t); // envia o valor de temperatura ao mestre
048. TinyWireS.send(h); // envia o valor de umidade do ar ao mestre
049. TinyWireS.send(umiSolo); // envia valor da umidade do solo ao mestre
050.
051.}
052.
053.void receiveEvent(uint8_t howMany) { // loop para receber instruções do
mestre
054.
055.  data = TinyWireS.receive(); // inicia o recebimento dos dados e
armazena na variável "data"
056.
057.  if (data == 0) { // caso a variável data for igual a 0 (00 em
binário)
058.    digitalWrite(1, LOW); // pino A1 nível baixo
059.    digitalWrite(4, LOW); // pino A4 nível baixo
060.  }
061.
062.  if (data == 1) { // caso a variável data for igual a 1 (01 em
binário)
063.    digitalWrite(1, HIGH); // pino A1 nível alto
064.    digitalWrite(4, LOW); // pino A4 nível baixo
065.  }
066.
067.  if (data == 2) { // caso a variável data for igual a 2 (10 em
binário)
068.    digitalWrite(1, LOW); // pino A1 nível baixo
069.    digitalWrite(4, HIGH); // pino A4 nível alto
070.  }
071.
072.  if (data == 3) { // caso a variável data for igual a 1 (11 em
binário)
073.    digitalWrite(1, HIGH); // pino A1 nível alto
074.    digitalWrite(4, HIGH); // pino A4 nível alto
075.  }
076.
077.}
```

APÊNDICE D – MANUAL DE UTILIZAÇÃO

Visão Geral

Esta secção aborda a operação e utilização do sistema de controle de temperatura, onde são definidos os componentes e a forma de introduzir e consultar os parâmetros desejados de operação.

O controlador de estufa é constituído de cinco partes principais:

- 1 – Fonte de alimentação;
- 2 – Controlador principal;
- 3 – Módulos remotos;
- 4 – Sensores;
- 5 – Atuadores.

A seguir são apresentadas as características e funções de cada componente do controlador.

1 – A fonte de alimentação tem sua tensão de saída de 12Vcc e capacidade de corrente de 5A, sua função é suprir a necessidade de energia elétrica ao circuito.

2 – O controlador principal é onde todos os dados e parâmetros são armazenados e processados, para depois tomar a ação de controle necessária para que os valores desejados sejam alcançados.

3 – Os módulos remotos são instalados na estufa nos pontos onde há interesse de se medir valores de temperatura, umidade do solo e do ar, eles são os responsáveis por receber a leitura dos sensores e enviá-los ao controlador principal.

4 – Dois sensores são utilizados, o sensor DHT11, que lê os valores de temperatura e umidade do ar, e o sensor de umidade de solo, que consiste em duas hastes introduzidas no solo e realiza a leitura da umidade.

5 – Os atuadores são os componentes que irão realizar a ação desejada, são dois tipos de válvulas hidráulicas, uma para irrigação direta no solo, outra tipo chuveiro, para chuva artificial. Também fazem parte dos atuadores, “resistências” elétricas que tem por função aquecer o ambiente interno da estufa.

Acoplando os Módulos

O sistema comporta até 6 (seis) módulos simultaneamente, cada um com um sensor DHT11, um sensor de umidade de solo, uma válvula para irrigação de solo e uma válvula de irrigação tipo “chuveiro”. Cada módulo tem um endereço específico (de 1 a 6) que é

reconhecido automaticamente pelo controlador principal, bastando apenas conectar cada módulo em seu respectivo conector.

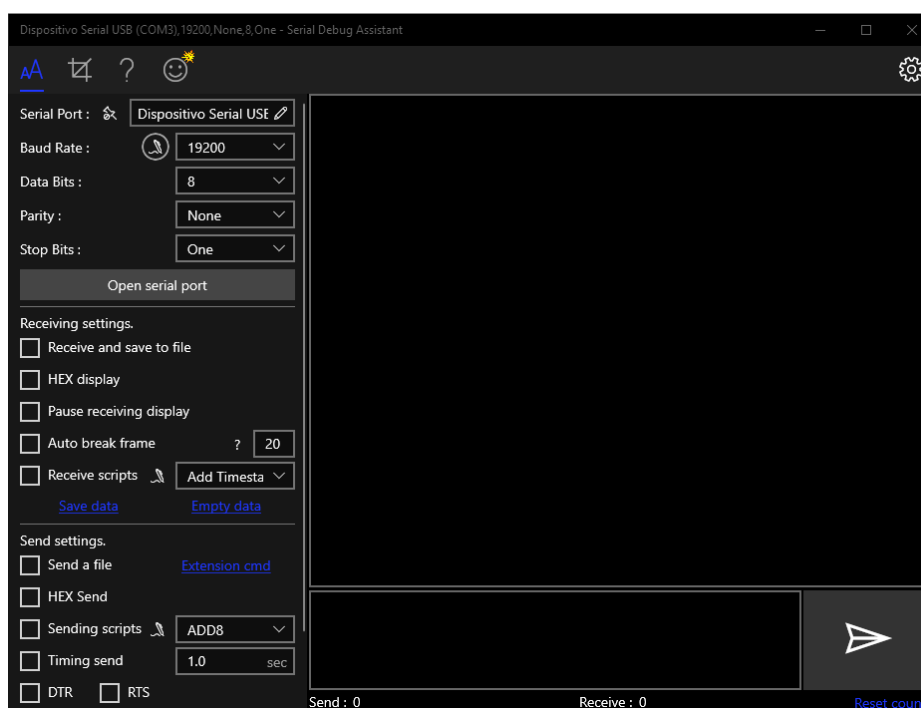
Conectando o controlador ao computador

Para que seja possível inserir os parâmetros desejados, é necessário utilizar um monitor serial, que será responsável por estabelecer a comunicação entre o PC e a placa de controle, para isso, deve-se baixar o aplicativo “Serial Debug Assistant” disponível gratuitamente na *Microsoft Store*®. Após o download, basta conectar o cabo USB no controlador principal e em uma porta USB do computador e abrir o aplicativo.

Configurando o aplicativo “Serial Debug Assistant”

Com o aplicativo Serial Debug Assistant aberto, deve-se primeiramente configurar qual a porta USB utilizada, para isso, em “Serial Port” é necessário clicar em “Dispositivo Serial USB” e escolher entre COM1, COM2, COM3, etc. Após isso, em “Baud Rate” é preciso selecionar 19200, que é a velocidade de comunicação entre o computador e o controlador. Com as configurações finalizadas, basta clicar em “Open serial port” que a comunicação será iniciada e os valores dos módulos ativos já serão exibidos.

Imagem 1- Visão geral do Serial Debug Assistant

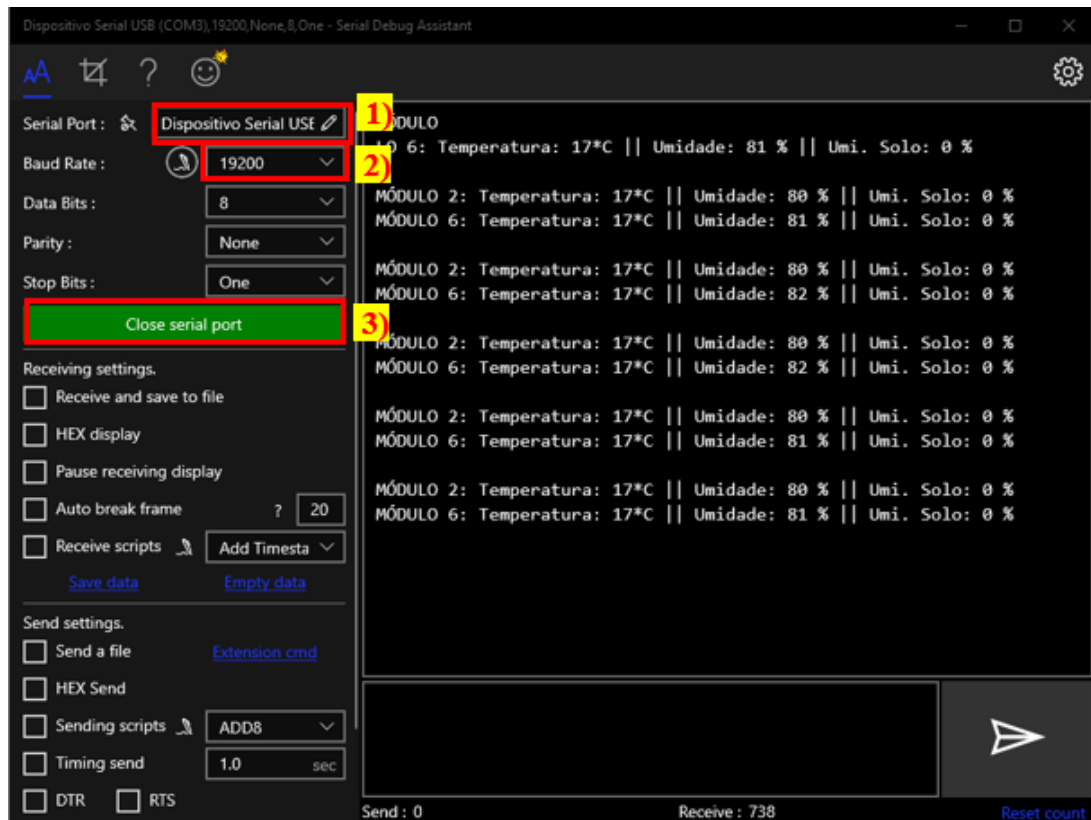


Etapas para configurar o aplicativo:

- 1) Selecionar a porta USB (COM1, COM2, COM3, etc.)

- 2) Configurar Baud Rate para 19200
- 3) Clicar em “Open serial port” o campo ficará na cor verde e mudará o texto para “Close serial port” então o terminal começará a ler e exibir as informações dos módulos caso já estejam instalados.

Imagem 2 - Monitor serial configurado

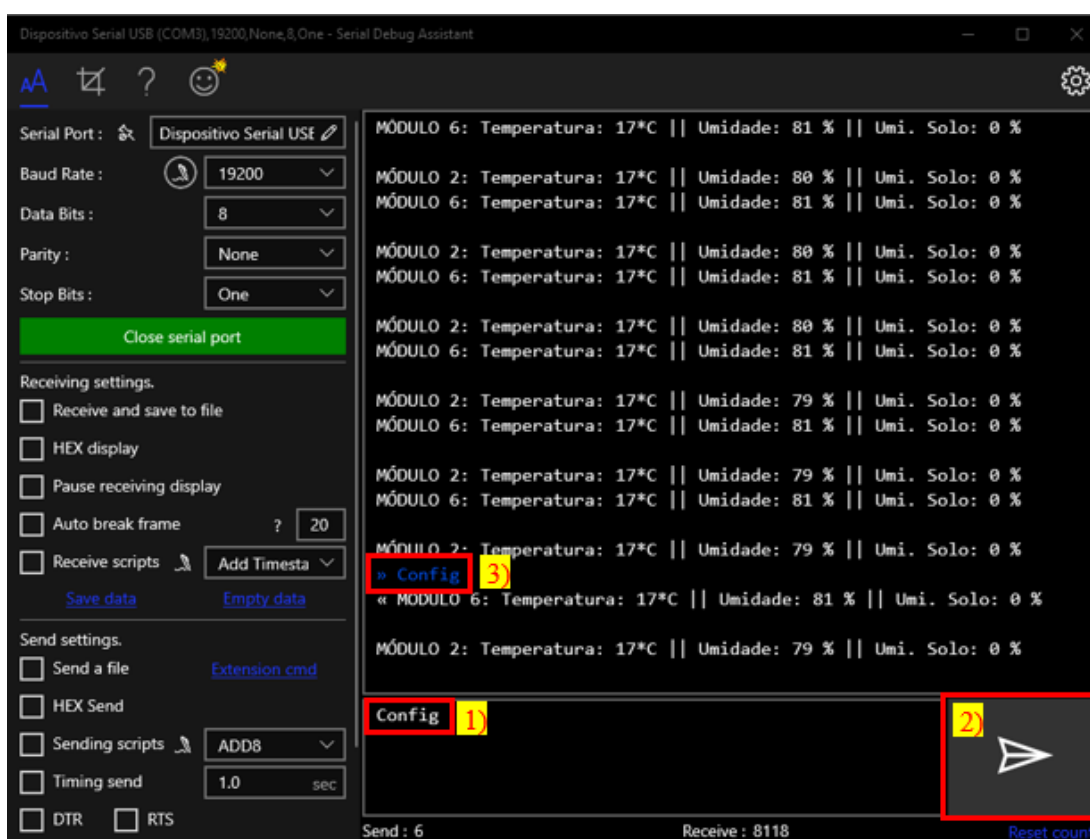


Entrando no modo de configuração

Sempre que desejar alterar algum parâmetro, consultar os parâmetros já configurados, ou checar o status dos módulos remotos, é necessário entrar no modo configuração. Para tal, é necessário enviar o comando “Config” através do monitor serial.

- 1) Digitar “Config” no monitor serial – Atenção para letras maiúsculas e minúsculas.
- 2) Enviar o comando clicando no botão de envio (a tecla Esc também realiza esta função).
- 3) Quando o comando é enviado, ele é exibido no monitor serial.

Imagem 3 - Função Configurações

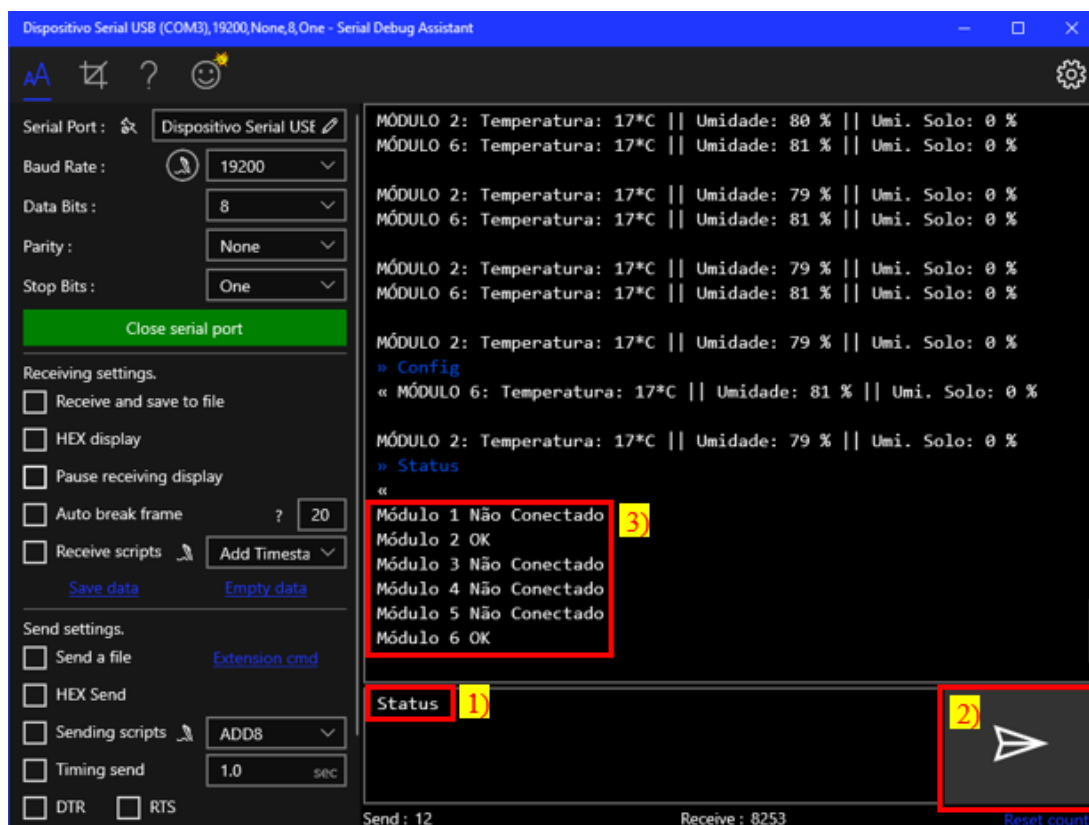


Quando o modo configuração está ativo, o LED indicador de funcionamento, que fica piscando enquanto o controlador está no modo de operação, muda seu status e passa a ficar aceso.

Verificando o status dos módulos remotos

Uma vez no modo de configuração, é possível verificar o status de todos os módulos, para isto basta enviar o comando “Status” através do monitor serial.

Imagem 4 - Comando Status



No exemplo acima, apenas os módulos 2 e 6 estão conectados ao controlador central.

Enviando parâmetros

É possível configurar de forma independente, os parâmetros de temperatura, umidade e umidade do solo, para cada um dos módulos remotos existentes. Abaixo se encontra a lista de comandos disponíveis:

- TM1 – Configura a temperatura desejada para o módulo 1
- UM1 – Configura a umidade do ar desejada para o módulo 1
- SM1 – Configura a umidade do solo desejada para o módulo 1
- TM2 – Configura a temperatura desejada para o módulo 2
- UM2 – Configura a umidade do ar desejada para o módulo 2
- SM2 – Configura a umidade do solo desejada para o módulo 2
- TM3 – Configura a temperatura desejada para o módulo 3
- UM3 – Configura a umidade do ar desejada para o módulo 3
- SM3 – Configura a umidade do solo desejada para o módulo 3
- TM4 – Configura a temperatura desejada para o módulo 4
- UM4 – Configura a umidade do ar desejada para o módulo 4

SM4 – Configura a umidade do solo desejada para o módulo 4

TM5 – Configura a temperatura desejada para o módulo 5

UM5 – Configura a umidade do ar desejada para o módulo 5

SM5 – Configura a umidade do solo desejada para o módulo 5

TM6 – Configura a temperatura desejada para o módulo 6

UM6 – Configura a umidade do ar desejada para o módulo 6

SM6 – Configura a umidade do solo desejada para o módulo 6

Para inserir estes parâmetros, basta enviar um dos comandos acima pelo monitor serial. Uma mensagem indicando qual parâmetro será alterado, será exibido, após isto basta digitar o valor desejado, e enviá-lo pelo monitor serial, uma nova mensagem é exibida com o valor inserido. Ao finalizar a configuração, é necessário enviar o comando “fim” pela porta serial.

Por exemplo, para configurar a temperatura do módulo 2:

- 1) Digite o comando “TM2” no terminal serial;
- 2) Envie o comando
- 3) Uma mensagem será exibida, com o parâmetro que será alterado
- 4) Digite o valor que deseja
- 5) Envie o comando
- 6) Será exibida uma mensagem com o valor configurado
- 7) Enviar o comando “fim” para sair da configuração de TM2.

Imagem 5 - Configuração de temperatura

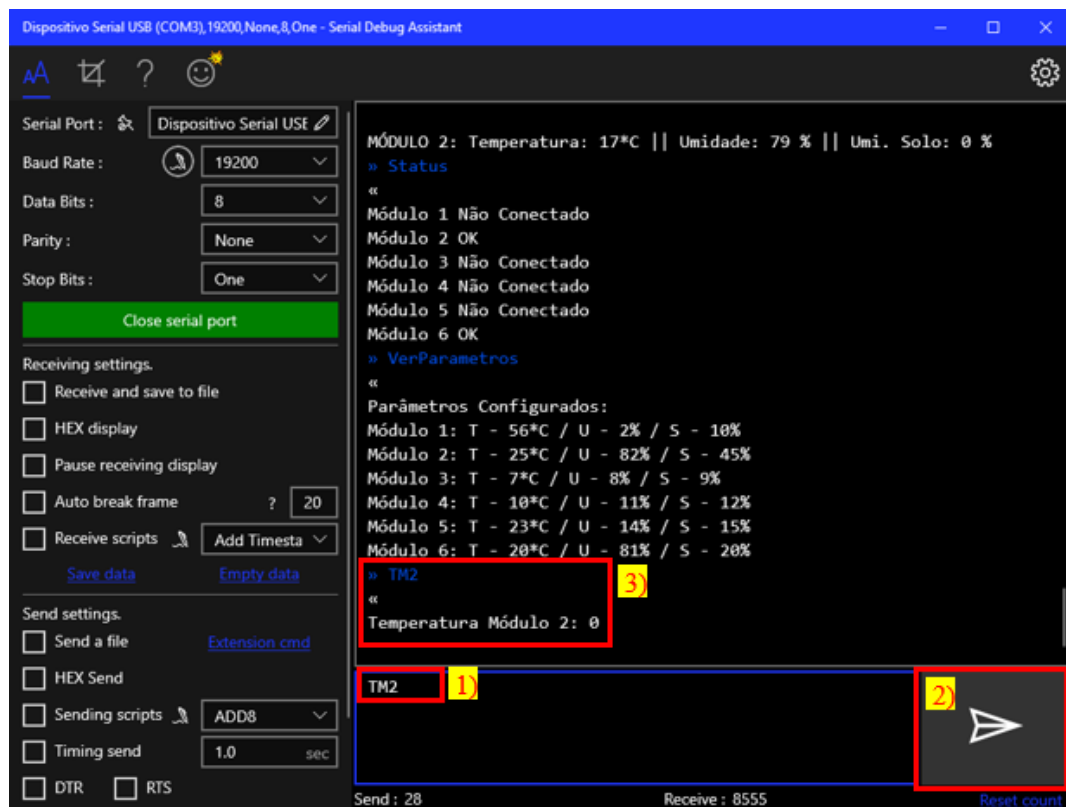


Imagem 6 - Envio do valor desejado

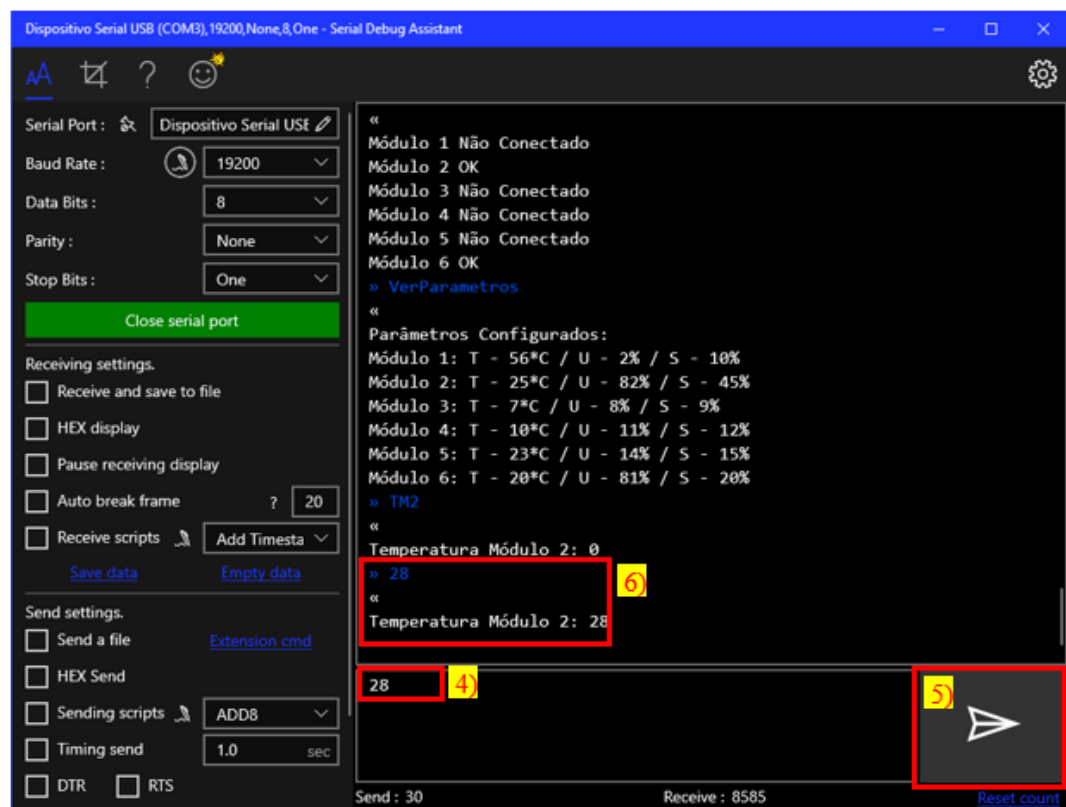
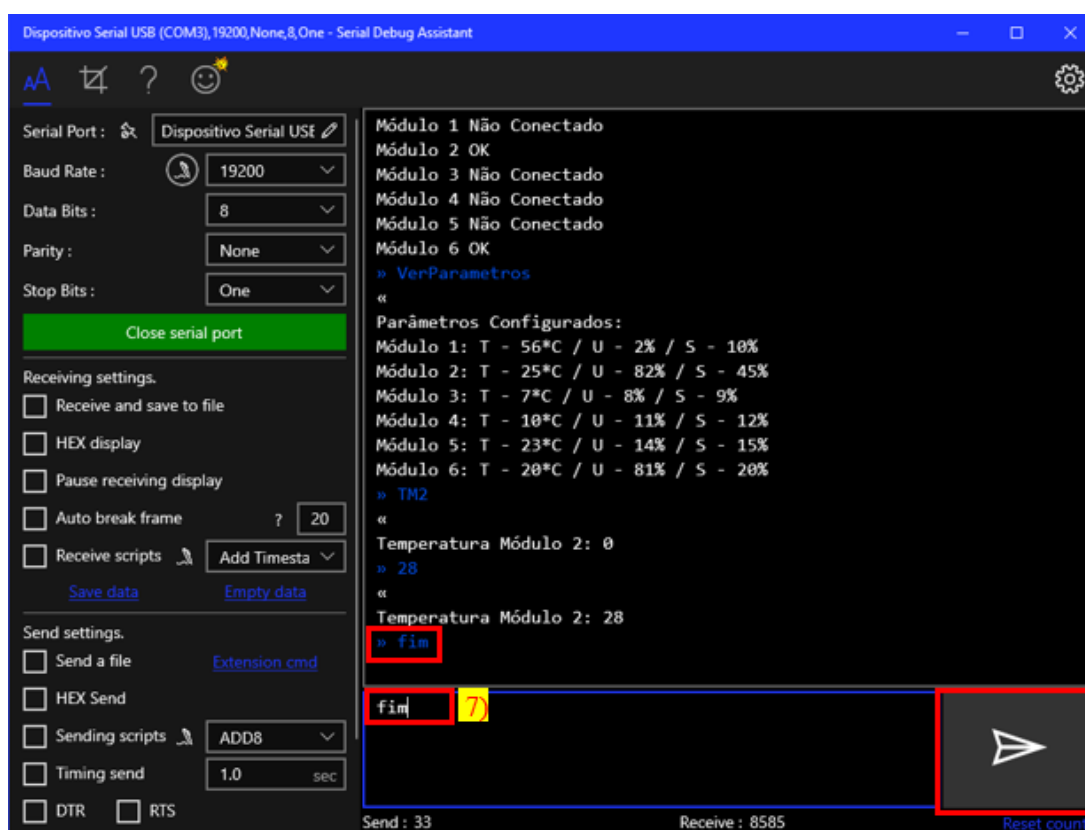


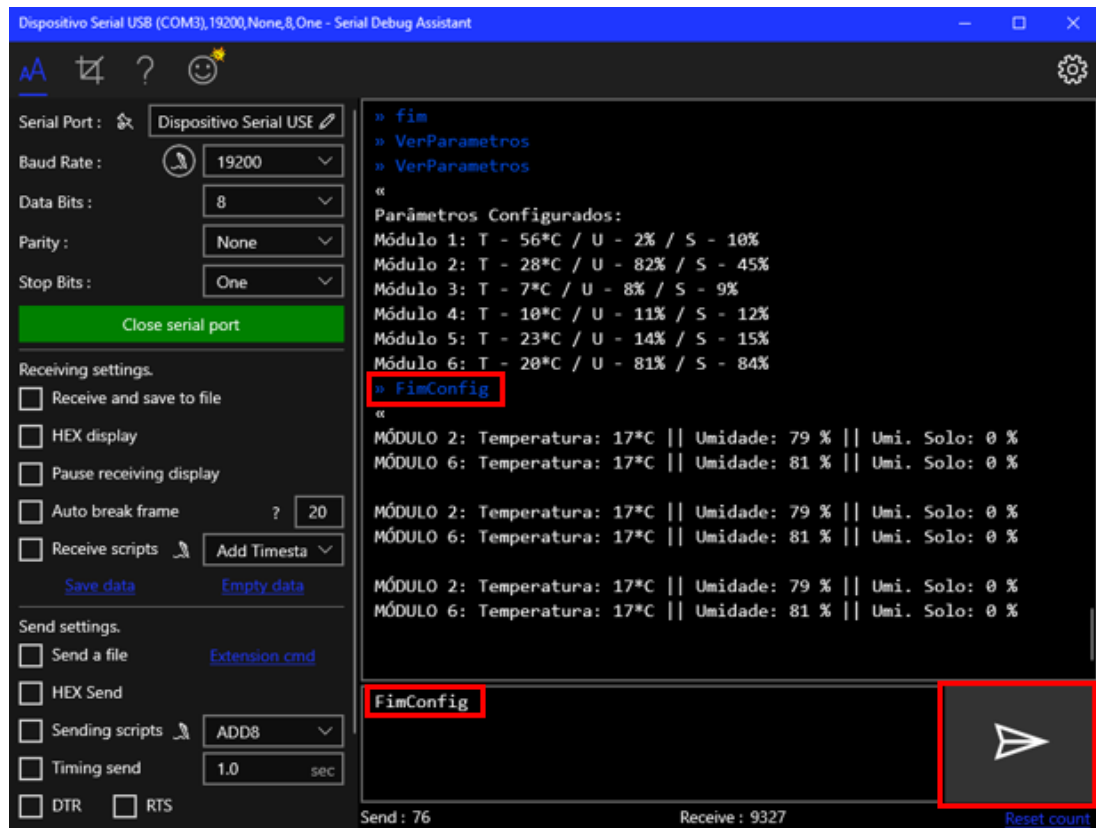
Imagem 7 - Fim da configuração de TM2



Saindo do modo de configuração:

Após todos os parâmetros desejados terem sido configurados pelo usuário, é necessário sair do modo de configuração, para que o controlador entre no modo de operação. Para isto, basta enviar o comando “FimConfig” através do monitor serial (atenção para maiúsculas e minúsculas). O controlador voltará para o modo “Operação” e começará a exibir os valores dos módulos.

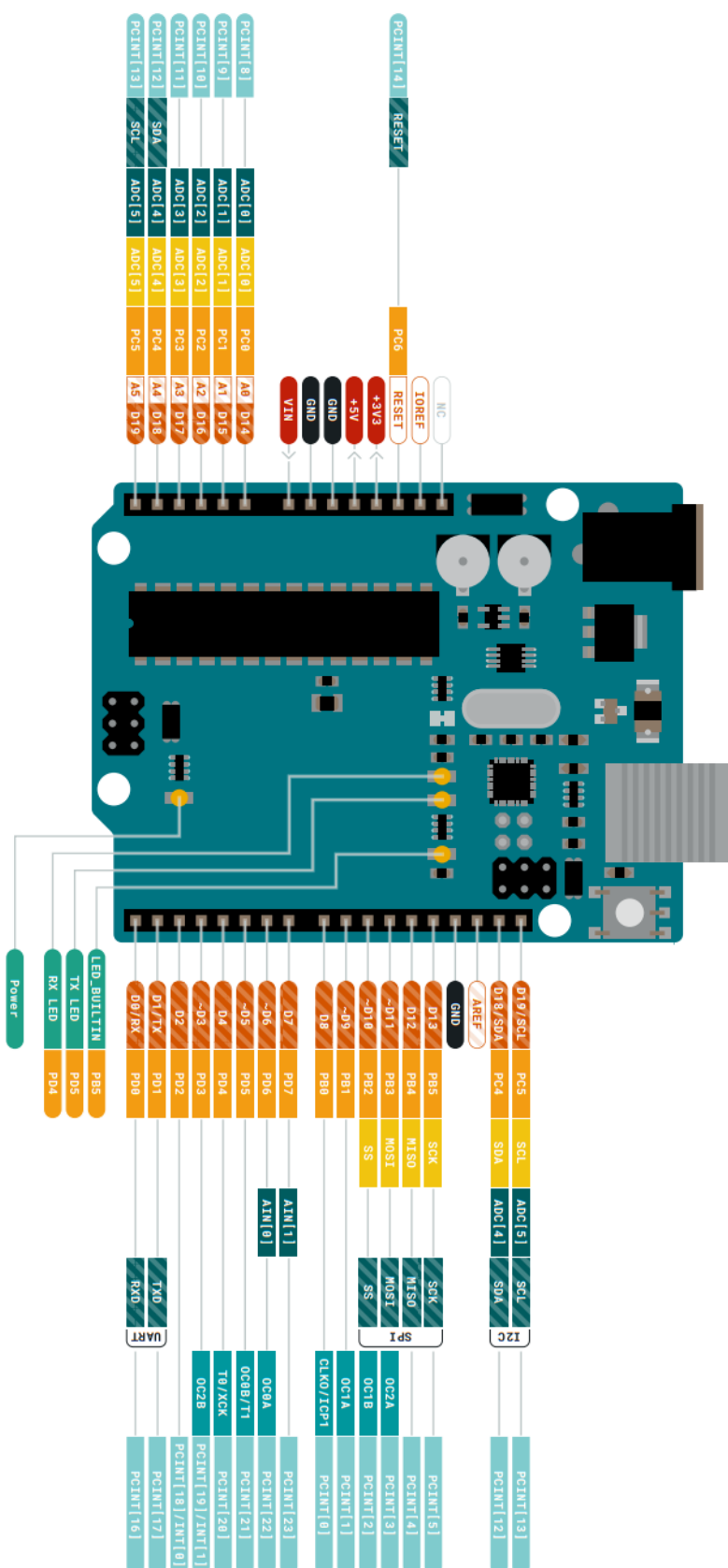
Imagem 8 - Saindo do modo Configuração



APÊNDICE E – LISTA DE MATERIAIS

01	Arduino UNO
06	AtTiny85
12	Transistor 2N7000
12	Resistor 10K Ω
12	Resistor 1K Ω
12	Conector 3 pinos
12	Conector 4 pinos
7	Placa fenoliteilhada

ANEXO A – LAYOUT PLACA ARDUINO UNO



Disponível em: https://content.arduino.cc/assets/Pinout-UNOrev3_latest.pdf.