UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
ENGENHARIA MECATRÔNICA

MARIANA DIAS MARTINS

CONTROLADOR LÓGICO PROGRAMÁVEL DE CÓDIGO INTERPRETADO

Joinville
2024

MARIANA DIAS MARTINS

CONTROLADOR LÓGICO PROGRAMÁVEL DE CÓDIGO INTERPRETADO

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

Orientador: Dr. Dalton Luiz Rech Vidor

Joinville
2024

Aos meus pais e avós.

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus e aos meus guias por me mostrarem o caminho e me levarem por esta jornada. Agradeço também:

À Universidade Federal de Santa Catarina, em especial ao Curso de Engenharia Mecatrônica e todo o seu corpo docente, pela oportunidade de atuar como aluna.

Ao meu orientador, professor Dalton Luiz Rech Vidor pela amizade e companheirismo, mas principalmente pela confiança atribuída a mim. Obrigada por ter acreditado em mim em todos os momentos.

Aos meus pais, cujo amor, apoio e encorajamento me guiaram através de todas as etapas da minha jornada acadêmica. Sua crença inabalável em mim e no meu potencial foram a força por trás das minhas conquistas.

Por fim, expresso minha gratidão aos amigos e colegas de classe, por todas as memórias e risadas. Juntos, enfrentamos desafios, celebramos sucessos e crescemos como indivíduos.

Este trabalho é o resultado do apoio coletivo que recebi e representa não apenas o meu esforço, mas também a confiança depositada em mim por aqueles que me cercam.

RESUMO

Com o avanço da automação, os CLPs se tornaram essenciais na indústria, oferecendo versatilidade e eficiência ao substituir a lógica cabeada a relé, reduzindo custos e facilitando mudanças nas linhas de montagem. No entanto, sua exclusividade limita seu uso em ambientes educacionais, tornando necessária a busca por alternativas acessíveis. Diante dessa problemática, surge a proposta de investigar microcontroladores como alternativas de baixo custo para emular CLPs em sistemas de automação industrial. Sendo assim, esta pesquisa tem como objetivo geral avaliar a viabilidade dessa abordagem, considerando desempenho, confiabilidade e escalabilidade em cenários simulados. Para isso, foi utilizado o microcontrolador Arduino UNO, programado com um código interpretado que lê instruções do usuário armazenadas em um cartão de memória e produz saídas adequadas às entradas simuladas. Os resultados obtidos mostram que é viável implementar a emulação de um CLP com microcontroladores, atendendo ao objetivo proposto e demonstrando o potencial dessa abordagem como solução prática e acessível.

Palavra-chave: CLP; automação industrial; Arduino; código interpretado.

ABSTRACT

With the advancement of automation, Programmable Logic Controllers (PLCs) have become essential in the industry, offering versatility and efficiency by replacing relay-based wired logic, reducing costs, and facilitating changes on assembly lines. However, their exclusivity limits their use in educational environments, necessitating the search for accessible alternatives. In light of this challenge, the proposal arises to investigate microcontrollers as low-cost alternatives to emulate PLCs in industrial automation systems. Therefore, this research aims to evaluate the feasibility of this approach, considering performance, reliability, and scalability in simulated scenarios. To achieve this, the Arduino UNO microcontroller was used, programmed with interpreted code that reads user instructions stored on a memory card and produces outputs corresponding to simulated inputs. The obtained results show that it is possible to implement PLC emulation with microcontrollers, meeting the proposed objective and demonstrating the potential of this approach as a practical and accessible solution.

Keywords: PLC; industrial automation; Arduino UNO; structured code.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CLP - Controlador Lógico Programável
CPU - Central Processing Unit
NA - Naturalmente Aberto
NF - Naturalmente Fechado
TON - Time On Delay
TOFF - Time OFF Delay
IoT - Internet of Things

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da automação, a evolução dos equipamentos que a conduzem tornou-se imprescindível. Entre esses dispositivos, destaca-se o Controlador Lógico Programável (CLP), que se tornou um dos principais protagonistas nas indústrias devido à versatilidade e poder de processamento. Os primeiros CLPs surgiram no final da década de 1960, impulsionados pela necessidade da General Motors de mudar a lógica de controle dos painéis de comando durante alterações na linha de montagem, mudanças essas que consumiam tempo e geravam custos significativos (Antonelli, 1998).

A introdução dos CLPs possibilitou eliminar a dependência dos relés nas lógicas de comando, substituindo-os por software, tornando a instalação e o funcionamento mais confiável que as estruturas físicas, otimizando espaços e facilitando a identificação de adversidades na linha de montagem. Esse avanço trouxe ganhos significativos para a indústria como redução de custos de produção, aumento de produtividade, diminuição de perdas e falhas no ciclo produtivo e, portanto, o aumento dos lucros e evolução das empresas (Nogueira; Novaes, 2015).

Atualmente, a crescente demanda por bens de consumo e serviços e, consequentemente, a busca incessante na otimização dos processos continua, evidenciando cada vez mais a importância da automação nesse contexto. No entanto, tecnologias de automação, como os CLPs, costumam ser dispendiosas e de propriedade exclusiva, o que por vezes torna inviável sua exploração e utilização em contextos educacionais, como salas de aulas ou laboratórios universitários. Nessas situações, é comum recorrer apenas a simuladores, o que resulta na ausência da componente prática no processo de ensino, como a montagem de um painel de controle, por exemplo.

Diante desse cenário, surge a necessidade de reformular as tecnologias para atender a demanda por otimização. Utilizando uma memória programável, é possivel implementar, num microcontrolador, um código que interprete funções específicas, como lógica, sequenciamento, temporização, contagem e aritmérica, utilizando módulos de entradas e saídas, de maneira semelhante ao que um CLP tradicional faria para controlar máquinas e processos (Capelli, 2007).

Sendo assim, este trabalho apresenta uma pesquisa que explora o microcontrolador Arduino UNO como uma alternativa ao CLP comum para controle industrial, avaliando seu desempenho, confiabilidade e escalabilidade em cenários simulados.

1.1 OBJETIVOS

Para contornar a problemática de alto custo de um CLP afim de democratizar seu uso em ambientes educacionais, propõe-se os seguintes objetivos.

1.1.1 Objetivo Geral

Investigar a viabilidade de utilizar a placa Arduino UNO como alternativa educacional de custo reduzido, para emular um Controlador Lógico Programável em sistemas de automação industrial.

1.1.2 Objetivos Específicos

- Propor uma arquitetura para aplicação do projeto;
- Desenvolver um código interpretado capaz de ler, pelo menos, quatro entradas digitais e fornecer quatro saídas digitais;
- Criar uma linguagem estruturada a partir de códigos a ser lida pelo microcontrolador;
- Demonstrar a aplicabilidade do CLP que utiliza o Arduino UNO em cenários industriais.

2 CONTROLADOR LÓGICO PROGRAMÁVEL

O Controlador Lógico Programável (CLP) é um equipamento eletrônico, digital, facilmente programado. Em linhas gerais, o CLP assemelha-se a um computador onde é possível inserir um programa para controlar e monitorar dispositivos de saídas, de acordo com parâmetros enviados por dispositivos de entrada. Os programas desenvolvidos são totalmente personalizáveis, compostos por uma variedade de instruções ou funções específicas, como lógica, sequenciamento, temporização, contagem e aritmética. Essa flexibilidade torna o CLP um dispositivo dinâmico, aplicável em diversos sistemas automáticos para acionamento e/ou monitoramento de máquinas e processos (Ribeiro, 2005).

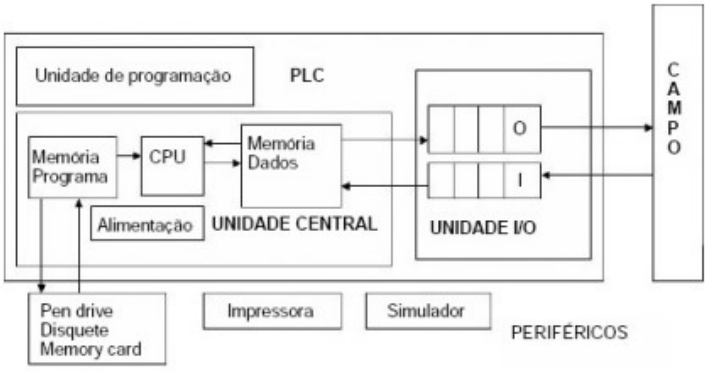
Neste capítulo são apresentadas as características e funcionalidades de um CLP, além de serem descritos os requisitos tanto de software quanto de hardware. São apresentadas, ainda, metodologias para desenvolvimento e implementação de CLPs.

2.1 ARQUITETURA

O hardware de um CLP é constituído por três partes fundamentais: unidade central de processamento, unidade I/O (entradas/saídas) e unidade de programação. A unidade central organiza as funções de controle e guarda o programa, a unidade I/O liga a unidade de programação com os dispositivos de campo a ser controlados e a unidade de programação é a interface homem/máquina para a escrita do programa do CLP (Prudente, 2011).

A Figura 1 ilustra o esquema de blocos do hardware de um CLP e demonstra também o percurso das informações.

Figura 1 – Hardware do controlador lógico programável.



Fonte: Prudente (2011, p. 52).

Na Figura 1, observa-se ainda as três unidades fundamentais da unidade central: a fonte de alimentação, a memória e o microprocessador. O suporte eletrônico

que inclui o microprocessador é comumente chamado Central Processing Unit (CPU) e é responsável pelo funcionamento lógico de todos os circuitos. Segundo Fávero (2013) as características da CPU são:

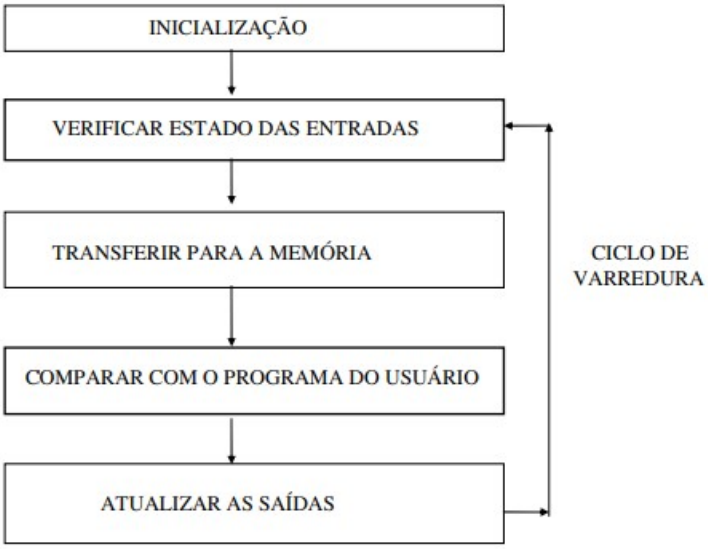
- Microprocessadores ou microcontroladores de 8 ou 64 bits;
- Endereçamento de memória até 64 Mega Bytes, podendo chegar a vários gigabytes dependendo do dispositivo;
- Velocidades de CLOCK entre 20MHz e 4GHz;
- Manipulação de dados decimais, octais, hexadecimais, até ponto flutuante.

A CPU faz a leitura dos sinais de entrada, processando as informações provenientes de botões, interruptores de posição, sensores ou transdutores. Quando há uma variação de sinal, a CPU responde elaborando novos dados conforme uma sequência predefinida, denominada programa. Esses dados geram sinais na amplitude adequada na saída, os quais controlam os dispositivos atuadores, possibilitando o movimento de válvulas pneumáticas, motores elétricos, entre outros, conforme a sequência programada (Prudente, 2011).

2.2 PRINCIPIO DE FUNCIONAMENTO

A partir da descrição sucinta da estrutura de um CLP, torna-se igualmente crucial entender a forma com que o dispositivo lê e processa as informações que chegam à unidade central. O esquema da Figura 2 apresenta o detalhamento da ordem sequencial pela qual a CPU realiza todas as suas operações, conhecida como ciclo de scan ou varredura.

Figura 2 – Ciclo de Scan.



Fonte: Antonelli (1998, p. 4).

Antonelli (1998) explica cada etapa do ciclo de varredura ou ciclo de scan. Começando pela inicalização que corresponde ao momento em que CLP é ligado e deve executar uma série de verificações e funções pré-programadas, entre as quais:

- Verificar o funcionamento eletrônico da CPU, memórias e circuitos auxiliares;
- Verificar a configuração interna e comparar com os circuitos instalados (entradas e saídas);
- Verificar o estado dos interruptores ou botões no painel de controle do CLP (RUN/STOP ou PROG);
- Desativar todas as saídas;
- Verificar a existência de um programa;
- Emitir um aviso de erro caso algum dos itens listados falhe.

Em seguida, o CLP lê cada uma das entradas, verificando se alguma foi acionada. Quando alguma entrada sofre uma alteração, o estado da entrada é armazenado numa região da memória chamada Memória Imagem das Entradas, que será consultada pelo CLP no decorrer do processamento do programa do usuário em COMPARAR COM O PROGRAMA DO USUÁRIO. Após consultar a Memória Imagem das Entradas, o CLP atualiza o estado da Memória Imagem das Saídas, de acordo com as instruções definidas pelo usuário em seu programa. Por fim, o CLP atualiza as interfaces ou módulos de saída com o valor contido na Memória das Saídas e inicia um novo ciclo de varredura (Antonelli, 1998).

Ciclos de scan mais modernos, como os descritos por Oliveira (2017), podem incorporar processos adicionais, como a comunicação e o *housekeeping*. A comunicação trata da comunicação do CLP com dispositivos externos, incluindo terminais de programação, computadores ou outros CLPs conectados em redes industriais. Enquanto o *housekeeping* é responsável pela gestão de memória, atualização de temporizadores e registros internos, bem como outras tarefas administrativas. Essas inclusões ampliam a funcionalidade e a integração do sistema, mas também aumentam sua complexidade operacional.

Ao tempo que demora para executar o ciclo de scan dá-se o nome de tempo de scan e está frequentemente associado ao tamanho do programa. Se o tempo de scan é longo, a entrada poderá passar do estado (0), desativada, ao estado (1), ativada, e novamente ao estado (0) enquanto o CLP se encontra num estado diferente daquele que foi lido na entrada. Isso implica que o CLP pode não detectar a variação do estado da entrada e, consequentemente, não desempenhará as instruções que deveria. Em geral, os CLPs possuem instruções para monitorar o tempo necessário para realizar um ciclo de varredura.

Caso o tempo de scan seja longo demais é possível bloquear o CLP. Este pode dispor de um temporizador interno denominado watch-dog, ajustado em torno de

150 ms, cuja função é parar automaticamente o ciclo se a varredura ultrapassar limite do tempo estabelecido. Essa medida é adotada para prevenir a geração de ciclos de programas repetitivos, como loops causados por mau funcionamento, conforme mencionado por Prudente (2011).

2.3 MÓDULOS OU INTERFACES DE ENTRADA E SAÍDA

Os módulos ou interfaces de Entrada são circuitos utilizados para adequar eletricamente os sinais de entrada para que possam ser processados pela CPU do CLP (Antonelli, 1998). As entradas permitem receber informações sobre o processo e são os pontos de ingresso dos sinais provenientes de botoeiras, contatos de relés, sensores, encoders e todos os tipos de dispositivos usados para monitorar o processo e fornecer alguma informação ao CLP.

As entradas podem ser digitais ou analógicas e ainda categorizadas como internas ou externas. As entradas externas são aquelas por onde entrará o sinal enviado por um sensor ao CLP e as entradas internas são aquelas que recebem sinal de um componente interno do CLP, como um contato de um temporizador utilizado para ligar um outro componente interno ou uma saída externa (Silva, 2015).

Por outro lado, os Módulos ou Interfaces de Saída adequam eletricamente os sinais vindos do microprocessador para que se possa atuar nos circuitos controlados, pelos quais o CLP pode alimentar uma carga, por exemplo. Assim como as entradas, as saídas também podem ser do tipo digitais ou analógicas e, ainda, externas ou internas. As saídas externas são aquelas por onde o CLP envia um sinal elétrico para um componente externo, enquanto que uma saída interna pode ser a bobina de um *FLAG* ou de um temporizador interno (Silva, 2015). Neste caso, *FLAG* seria um registro interno que sinaliza um evento ou alteração tanto em uma bobina quanto numa determinada operação realizada.

2.3.1 Digitais

As entradas ou saídas digitais estão relacionadas a sinais discretos, ou seja, que só oferecem dois estados possíveis, nível alto (1) ou nível baixo (0), ligado e desligado, respectivamente. Além disso, os sinais do tipo digital podem ser relativos ao tipo de alimentação do sensor, podendo operar em corrente contínua (24 VCC) ou em corrente alternada (110 ou 220 VCA) (Prudente, 2011). Assim, pode ser notado que o sinal digital não se baseia apenas na condição física de contato aberto ou fechado, mas sim no nível de tensão passada pelo ponto de contato.

2.3.2 Analógicas

As entradas ou saídas analógicas são aquelas que recebem sinais contínuos no tempo e que podem assumir qualquer valor entre o mínimo e o máximo valor de trabalho da entrada, são grandezas físicas como temperatura, pressão, velocidade ou aceleração. O tratamento desses sinais é notavelmente mais trabalhoso do que o dos sinais digitais, sobretudo quando se quer precisão de transmissão. Neste caso, é preciso recorrer a memórias de 16 ou 32 bits (Prudente, 2011).

Uma informação importante a respeito das entradas analógicas é a sua resolução, normalmente medida em bits. Quanto maior for o número de bits de uma entrada analógica melhor será a representação dessa grandeza. Antonelli (1998) dá o exemplo de que uma placa de entrada analógica de 0 a 10 VCC com uma resolução de 8 bits permite uma sensibilidade de 39,2 mV, enquanto que a mesma faixa com uma resolução de 12 bits permite uma sensibilidade de 2,4 mV e uma de 16 bits permite uma sensibilidade de 0,2 mV.

2.4 FUNÇÕES DE DESEMPENHO OPERACIONAL

O CLP foi projetado para operar em ambiente industrial suportando variações de temperatura, umidade e distúrbios elétricos para que, por meio do processamento de entradas e saídas, conseguisse substituir os quadros de relés e pudesse flexibilizar o controle das máquinas e processos. Sendo assim, Ribeiro (2005) elenca as funções mínimas que um CLP deve ser capaz de desempenhar:

- Relé básico;
- Temporização no ligamento, desligamento, retentivo ou não, com base e tempo de 1 e 0,1 s;
- Contador crescente ou decrescente;
- Transferência de blocos;
- Transmissão por exceção de mudança de status;
- Lógica booleana (AND, OR, NOT);
- Operações matemáticas (soma, subtração, multiplicação, divisão, raiz);
- Sequenciadores;
- Comparadores (maior, menor, maior ou igual, menor ou igual, igual, não igual);
- Linearizadores;
- Calculo matemático de ponto flutuante para a correção da vazão devida a pressão e temperatura;
- Integração de vazão instantânea durante intervalo de tempo;
- Filtro de sinais analógicos.

Além disso, é importante considerar alguns métodos de proteção de dados, autodiagnóstico e protocolos de comunicação.

2.4.1 Proteção dos dados

Todo CLP deve registrar em sua memória os valores das entradas e saídas, que devem ser acessíveis tanto localmente pelo terminal de programação como remotamente por uma interface de supervisão e controle (Ribeiro, 2005). Entretanto, em caso de falta de energia elétrica, o sistema operacional deve automaticamente garantir a alimentação das memórias por uma bateria de backup para evitar a perda de dados (Prudente, 2011).

2.4.2 Autodiagnóstico

Ao final de cada scan, o CLP efetua uma execução de autoteste ou diagnóstico sobre o funcionamento do seu circuito interno. Se ocorrer qualquer anomalia, o sistema operacional emite um sinal de alarme para advertir o operador de qualquer mau funcionamento, desabilitando automaticamente as saídas (Prudente, 2011). Compreende-se por mau funcionamento qualquer anomalia no microprocessador, erro na sintaxe do programa do usuário, tempo de scan anormal, nível de bateria de backup insuficiente, anomalias na memória ou na exportação de dados.

Cabe ainda ao autodiagnóstico notificar o sistema da repartida automática após uma falha na alimentação.

2.5 LINGUAGENS DE PROGRAMAÇÃO

Como o CLP veio para substituir relés eletromecânicos, é mais comumente programado usando lógica de relés. As cinco linguagens que são encontradas geralmente em programas de CLP e que são padronizadas pela norma IEC 61131-3 são: Texto Estruturado (Strutured Text ou ST) e Lista de Instruções (Instruction List ou IL) sendo classificadas como linguagens textuais. Diagrama Ladder (LD) e Diagramas de Blocos Funcionais (Function Block Diagram ou FBD) classificadas como linguagens gráficas. Linguagem de Diagrama Sequencial (Sequential Flow Chart ou SFC) classificada como linguagem gráfica, porém permite a adição de textos (ICE, 2013).

2.5.1 Ladder

Devido à sua semelhança com representações físicas dos circuitos elétricos, os diagramas ladder de relés são facilmente entendidos por muitas pessoas não instrumentistas. Eles usam símbolos intuitivos, como contatos e bobinas, que correspondem a elementos comuns em sistemas elétricos e mecânicos, facilitando a compreensão das conexões e interações entre os dispositivos do sistema.

Cada linha e bloco do programa ladder representam operações e decisões lógicas que controlam o funcionamento das saídas e entradas do sistema. Os blocos mais comuns são: contatos normalmente abertos (NA) ou normalmente fechados (NF), bobinas, blocos de SET e RESET, temporizadores e contadores, além das operações lógicas AND (e), OR (ou) ou NOT (não).

Contatos normalmente abertos ou normalmente fechados são usados para representar a condição de entradas como sensores ou interruptores e determinam se a saída deve ser ativada ou não. Um contato NA permite a passagem de energia quando é acionado, enquanto um contato NF conduz energia quando não está atuado (Mendes, 2021).

As bobinas, por sua vez, representam as saídas do sistema, como motores, válvulas e luzes, e podem ser associadas a comandos de SET ou RESET. O bloco SET ativa a saída e a mantém ligada mesmo na ausência do sinal de entrada, já o bloco RESET desativa a saída enquanto o sinal de entrada está presente e, mesmo que a entrada cesse, a saída permanece desligada (Mendes, 2021).

Os temporizadores controlam o tempo de acionamento de uma saída e os tipos mais comuns são o Time ON Delay (TON) e o Time OFF Delay (TOFF). O TON inicia a contagem ao detectar um sinal de entrada e ativa a saída após o tempo configurado. Por outro lado, o TOFF desliga a saída após um intervalo definido e caso o sinal de entrada retorne antes do término do tempo, o temporizador reinicia a contagem (Mendes, 2021).

Por último, o contador contabiliza pulsos elétricos ou eventos no sistema. Ele inicia em zero e incrementa a cada pulso de entrada, ativando a saída ao atingir um número predefinido de contagens.

Ao combinar esses elementos, é possível criar lógicas complexas que garantem o controle adequado dos processos industriais. Assim, o programa ladder integra os dispositivos físicos ao controle lógico, garantindo que a interação entre entradas e saídas ocorra de forma eficiente e confiável.

3 METODOLOGIA

A condução desta pesquisa seguirá uma abordagem sistemática e estruturada visando alcançar os objetivos propostos. Dado que o sistema deve emular um Controlador Lógico Programável (CLP) e sua forma de operação, torna-se necessário incorporar alguns elementos cruciais para o seu funcionamento. Estes incluem o microcontrolador Arduino, responsável por interpretar o programa armazenado no cartão de memória contendo o código do usuário; uma memória interna ao microcontrolador para registrar os estados das entradas e saídas; as próprias entradas e saídas; e, adicionalmente, uma interface com o usuário e um banco de dados externo ao sistema (Prudente, 2011).

Com esse propósito, será desenvolvido um código interpretado capaz de simular as funções principais de um CLP, que será orientado para a leitura de pelo menos quatro entradas digitais e fornecimento de quatro saídas digitais. Paralelamente, será implementada uma linguagem de estruturação que permita uma programação intuitiva e eficiente do código do usuário. Por fim, o sistema será submetido a experimentos controlados nos quais variáveis de entrada conhecidas serão aplicadas, permitindo a avaliação da congruência das saídas com os resultados esperados (GIL, 2008).

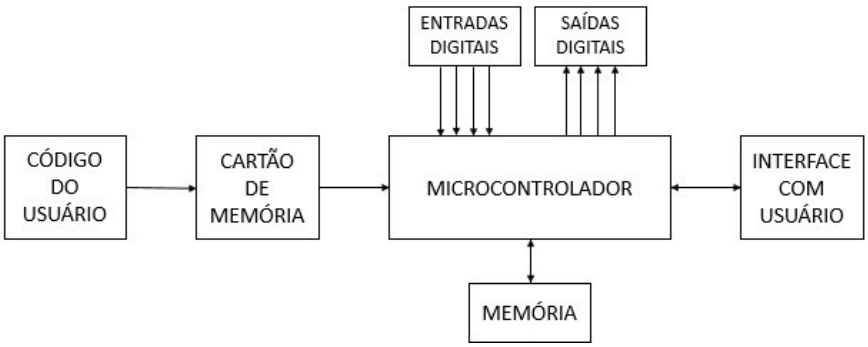
Este ultimo passo é crucial para garantir a viabilidade do sistema emulado e avaliar o seu desempenho e confiabilidade, além de atestar se o mesmo poderia ser usado em ambientes educacionais para simular cenários industriais simples.

Nesse sentido, a primeira fase do estudo irá abranger a elaboração de uma proposta de arquitetura de hardware, na qual os principais componentes do sistema emulador de CLP serão delineados. Posteriormente, será realizado um estudo sobre o código interpretado, com o objetivo de detalhar o raciocínio por trás do fluxo de leitura das entradas e processamento das funções lógicas, explicando como ler e criar a lógica do código do usuário. Por fim, a última etapa consistirá no desenvolvimento e na implementação do código no microcontrolador escolhido, consolidando a integração entre hardware e software.

3.1 HARDWARE

Esta seção incluirá a integração da unidade de processamento com os demais módulos, especificando o fluxo dos dados, conforme ilustrado na Figura 3.

Figura 3 – Diagrama de Blocos.



Fonte: Autor (2024).

O hardware desenvolvido para o projeto foi projetado para ser funcional e de fácil implementação, utilizando componentes acessíveis e compatíveis com a plataforma Arduino. Sendo assim, os componentes apresentados na Figura 3 serão replicados pelos componentes:

- Arduino Uno, escolhido como o microcontrolador principal devido à sua versatilidade e ampla compatibilidade com sensores e periféricos, será responsável por processar as entradas, executar a lógica programada e controlar as saídas;
- Quatro botões comuns configurados para atuar como as entradas do sistema;
- Quatro LEDs configurados para simular as saídas do sistema, fornecendo uma representação visual clara do estado do sistema em resposta às entradas acionadas;
- Cartão Micro SD Transcend de 512 MB conectado ao módulo SD Card como meio de armazenamento físico, tanto do código interpretado quanto dos dados gerados pelo programa.
- Módulo Micro SD Card utilizado para leitura e escrita de dados no cartão SD;

Para garantir o funcionamento adequado da configuração, é essencial assegurar as conexões corretas entre o módulo MicroSD e o microcontrolador. O módulo MicroSD possui seis pinos principais, que possibilitam sua comunicação com o Arduino através do protocolo SPI, são eles:

- VCC, pino de alimentação, deve ser conectado ao pino 5V do Arduino;
- GND, pino de aterramento, conectado ao pino GND do Arduino;
- Master In Slave Out (MISO), saída SPI, responsável pelo envio de dados;
- Master Out Slave In (MOSI), entrada SPI, responsável pelo recebimento de dados;
- Serial Clock (SCK), recebe pulsos de clock do Arduino para sincronizar a transmissão de dados;

- Chip Select (CS), pino de controle utilizado para selecionar o módulo como dispositivo ativo no barramento SPI.

A seguir, na Tabela 1, são apresentadas as conexões específicas entre os pinos do módulo MicroSD e os pinos do Arduino, considerando a utilização da placa Arduino UNO:

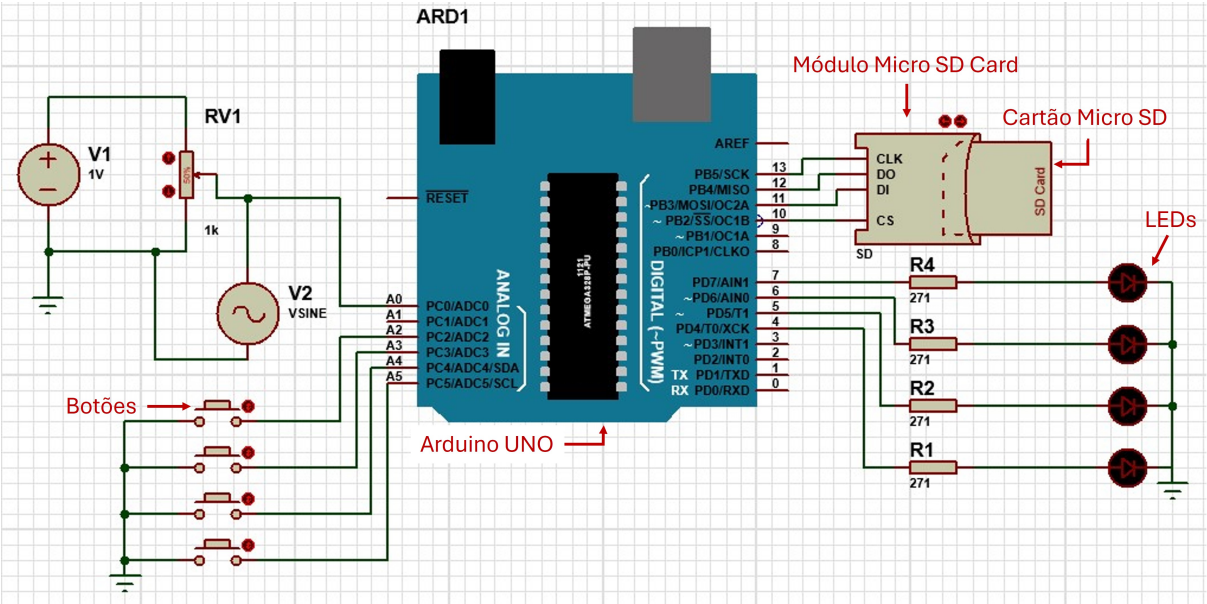
Tabela 1 – Conexões entre o Módulo MicroSD e o Arduino.

Módulo MicroSD	Arduino UNO
VCC	5V
GND	GND
MISO	12
MOSI	11
SCK	13
CS	10

Fonte: Autor (2024).

O diagrama representado na Figura 4 detalha as conexões feitas para integrar todos os componentes.

Figura 4 – Diagrama Esquemático do Hardware.



Fonte: Autor (2024).

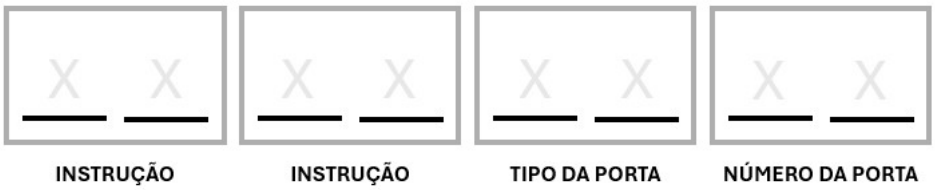
Adicionalmente, para assegurar o correto funcionamento da leitura e gravação no cartão MicroSD, é fundamental realizar a formatação do cartão em FAT16 ou FAT32, já que a maioria das bibliotecas e softwares que interagem com cartões SD foram projetadas para trabalharem com esses sistemas de arquivos. Isso porque o sistema FAT possui uma estrutura de diretórios simples, facilitando o armazenamento e a leitura em dispositivos com pouca capacidade de processamento e memória.

3.2 CÓDIGO INTERPRETADO

Após a conclusão do hardware e da formatação do cartão SD, foi necessário desenvolver o código interpretado. Para isso, a lógica de escrita foi baseada na tabela ASCII. Embora o uso de siglas de caracteres fosse mais intuitivo para escrita e visualização no arquivo, optou-se pela utilização de números devido à memória RAM do Arduino UNO limitada a 2048 bytes (Arduino José Bagur, 2023).

Cada linha escrita no arquivo TXT foi projetada para representar uma instrução e corresponder a um tipo de contato baseado na lógica ladder. Assim, definiu-se que cada par de dígitos representaria uma letra na tabela ASCII. Enquanto os sufixos numéricos (com até quatro dígitos) indicariam a instrução a ser executada, os dois dígitos subsequentes especificariam o tipo de porta que receberia o comando (entrada ou saída - IN ou OUT) e os dois últimos dígitos identificariam o número da porta associada à instrução, conforme ilustrado na Figura 5.

Figura 5 – Estrutura Usual da Linha de Instrução.



Fonte: Autor (2024).

Esse formato, com os dígitos da porta no final do número interpretado, facilita a reestruturação do código caso seja necessário expandi-lo para incluir mais portas lógicas, já que essa estrutura permite isolar os dois últimos dígitos com facilidade, caso necessário. Por exemplo, na instrução hipotética 827311, que traduz os caracteres R, I e 11 (Read In 11) para ler a entrada 11, é possível identificar a entrada de forma isolada, conforme ilustrado:

$$Porta = 827311 - 827300 = 11 \tag{1}$$

Algumas das instruções mais utilizadas estão listadas na Tabela 2, onde 'XX' representa o número da porta associada à instrução, podendo variar de 1 a 4 no caso de entradas e saídas, ou de 1 a 8 no caso de portas lógicas do tipo OU.

Tabela 2 – Relação Ação Lógica e Código Interpretado

Ação Lógica	Código Interpretado	Visualização com Caracteres
Início do programa	**0**	0
Nova linha	**787600**	New Line 00
Lê entrada	**8273XX**	Read In XX
Lê entrada NF	**-8273XX**	-Read In XX
Lê saída	**8279XX**	Read Out XX
Escreve na saída	**8779XX**	Write Out XX
Set bobina	**838779XX**	Set Write Out XX
Reset bobina	**828779XX**	Reset Write Out XX
TON	**847978**	T O N
TOFF	**847970**	T O F
Abre OU	**40**	(
Fecha OU	**41**	)
Guarda OU	**4041XX**	( ) XX

Fonte: Autor (2024).

Como pode ser observado na Tabela 2, existem algumas instruções específicas ou momentos do código que divergem dessa estruturação padrão, pois necessitam ser organizadas em blocos, um exemplo disso são os timers. No projeto, o programa foi estruturado para incluir apenas um TON e um TOFF, que são representados de forma a compor o bloco mostrado na Figura 6.

Figura 6 – Bloco Lógico TON e TOFF.

```
2   827301
3   847978 /TON
4   60000  /Tempo em milisegundos
5   7901   /Tipo (Out) e numero (01) da porta que vai ser ligada
6   787600
7   827302
8   847970 /TOF
9   60000  /Tempo em milisegundos
10  7901   /Tipo (Out) e numero (01) da porta que vai ser desligada
```

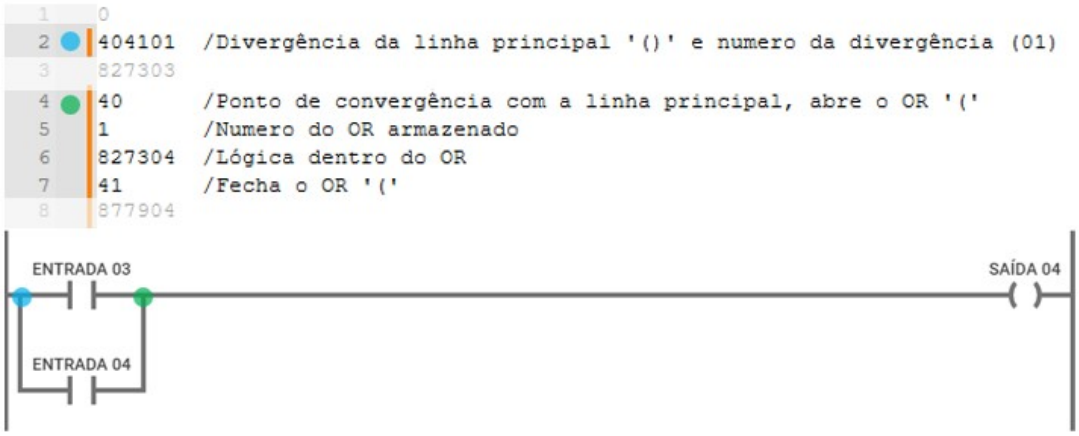
Fonte: Autor (2024).

Outro elemento da lógica ladder que precisa ser estruturado em forma de bloco é o OR. O OR precisa ser acionado em duas instâncias: quando surge e diverge da linha principal de comandos e quando faz a intersecção e converge de volta à linha principal.

No primeiro caso, utiliza-se a instrução '()XX', que indica que o valor lógico acumulado até aquele ponto deve ser armazenado em uma variável auxiliar para consultas futuras. Como é possível criar até 8 laços OR (por limitação do projeto) no programa, essa variável auxiliar também armazena a posição 'XX' de quando esse OR surgiu, essa posição é sempre por ordem de varredura.

Por sua vez, a intersecção do OR com a linha principal de comandos segue a estruturação apresentada no bloco da Figura 7. Essa estrutura permite a formação de cascatas de blocos OR, garantindo mais um nível de flexibilidade e complexidade ao programa do usuário.

Figura 7 – Bloco Lógico OR.



Fonte: Autor (2024).

Além dos blocos de temporizadores e da lógica OR, existe ainda o código de início, representado por ’0’, para sinalizar o início da execução do arquivo; e o código de nova linha ’787600’, para indicar que o valor lógico acumulado deve ser resetado para 1, garantindo, de forma semelhante a um diagrama ladder, que cada nova linha comece com um sinal ativo.

Finalmente, visando garantir a leitura correta do arquivo TXT, é essencial que não haja espaços entre os números ou caracteres extras, como o ’\r’ (retorno de carro). Por isso, utilizou-se o Notepad++ para converter o arquivo para o formato UNIX, que usa apenas o ’\n’ (nova linha) no final de cada linha.

3.3 CÓDIGO ARDUINO

A implementação do CLP foi projetada para integrar funcionalidades essenciais, como a leitura do cartão SD, a decodificação do código interpretado e a execução dos comandos lógicos armazenados dentro do cartão no formato TXT, a fim de emular um ciclo scan.

3.3.1 Bibliotecas e Variáveis Globais

Para alcançar esses objetivos, o desenvolvimento do sistema fez uso de três bibliotecas principais: SPI.h, SD.h e arduino-timer.h, com todo o código elaborado na IDE Arduino.

A biblioteca SPI.h fornece funções prontas que facilitam a configuração e gerenciamento da comunicação entre o microcontrolador e os dispositivos periféricos por meio do protocolo Serial Peripheral Interface (SPI). Protocolo este, amplamente adotado por sua velocidade e simplicidade na conexão de múltiplos dispositivos em um único barramento (Arduino, 2024).

Já a biblioteca SD.h foi empregada para o gerenciamento de arquivos e dados no cartão MicroSD. Ela oferece uma interface de alto nível para leitura e escrita, permitindo tanto o acesso ao código interpretado quanto o armazenamento dos registros de operação (Arduino, 2024).

Por fim, a biblioteca arduino-timer.h foi utilizada para criar timers de código não bloqueante. Diferente dos métodos tradicionais baseados em delays, essa biblioteca possibilita a realização de tarefas temporizadas de maneira assíncrona, garantindo alta responsividade e permitindo que o sistema execute múltiplas operações eficientemente e em tempo real (Contreras, 2023).

Essa biblioteca se baseia em dois métodos principais: *.tick()* e *.at()*. O *.tick()* deve ser chamado no loop principal e é responsável por atualizar os temporizadores e verificar se alguma tarefa precisa ser executada. Já o *.at()* permite agendar funções em intervalos específicos, assegurando sua execução periódica e precisa. Essa abordagem elimina bloqueios no fluxo de execução, tornando o sistema mais estável.

Além das bibliotecas adequadas para comunicação e controle temporal, o gerenciamento eficiente da memória e das variáveis é fundamental para o bom desempenho do programa. Nesse sentido, é necessário reservar um espaço de memória que replique a Memória Imagem das Entradas e das Saídas de um CLP, com variáveis armazenadas fora do loop principal. No Arduino, essa memória é implementada por meio de variáveis globais, que armazenam os estados das entradas e saídas, permitindo que o CLP as consulte continuamente durante o processamento do programa do usuário (Antonelli, 1998). Aqui também devem ser criadas outras variáveis que não podem sofrer reinicialização pelo loop principal, como, por exemplo, os timers.

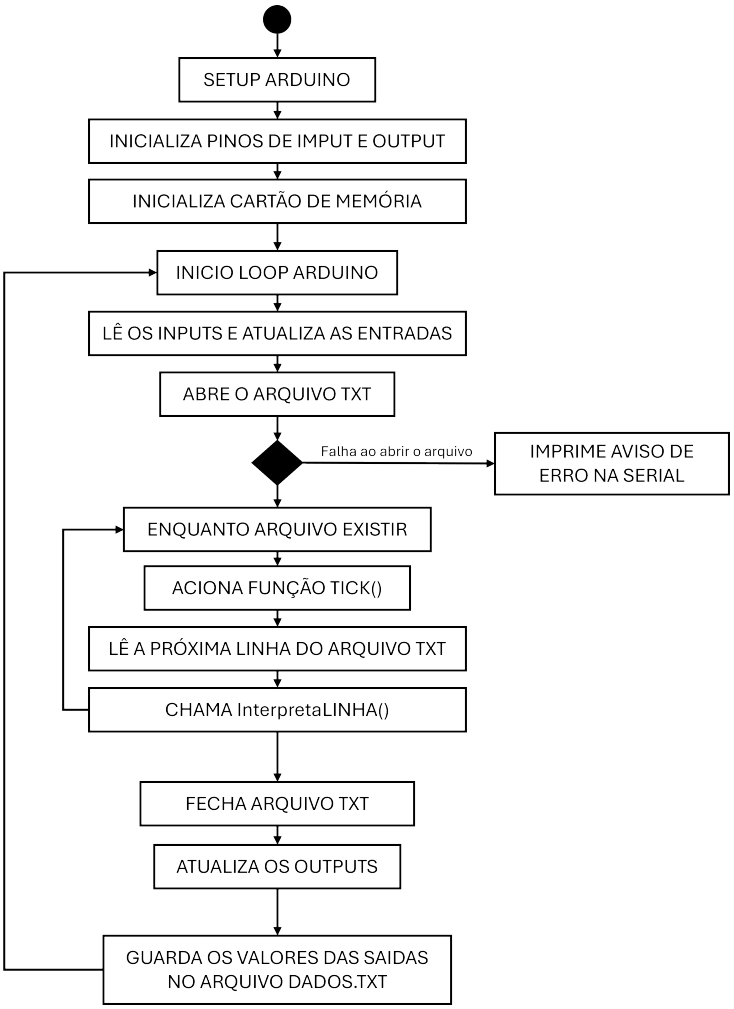
As inicializações tanto das bibliotecas, quanto das variáveis globais são feitas na seção principal do código Arduino, denominado 'Codigo_CLP.ino', que coordena a operação adequada do sistema.

3.3.2 Codigo_CLP.ino

O 'Codigo_CLP.ino' executa uma varredura similar ao ciclo de scan de um CLP. Após realizar o setup inicial, o código entra no loop principal onde, a cada iteração, o sistema realiza a leitura das entradas com o comando *digitalRead()*, executa o processamento da lógica e atualiza as saídas com o comando *digitalWrite()* de acordo com os resultados obtidos.

A execução do código é representada no fluxograma da Figura 8, que ilustra a interação do sistema.

Figura 8 – Fluxograma Codigo_CLP.ino.



Fonte: Autor (2024).

O processo é cíclico, repetindo-se continuamente, o que garante que o sistema esteja sempre monitorando as entradas e ajustando as saídas conforme as condições do código interpretado armazenado no cartão SD.

Nesse processo o programa abre o arquivo armazenado no cartão através da função *SD.open(Arquivo)*, onde 'Arquivo' é o nome do TXT no cartão SD que será lido linha a linha pela função *readStringUntil('\n').toInt()*. Cada linha é então passada para a função *InterpretaLINE* que irá fazer a interpretação daquela instrução e irá retornar o valor acumulado no sinal lógico (0 ou 1) para atualizar a variável *ValorLINHA*. Quando o documento TXT chega ao fim, o arquivo é fechado com *close()*, as saídas são atualizadas e seus valores são salvos no cartão de memória.

3.3.3 Codigo_CLP.h

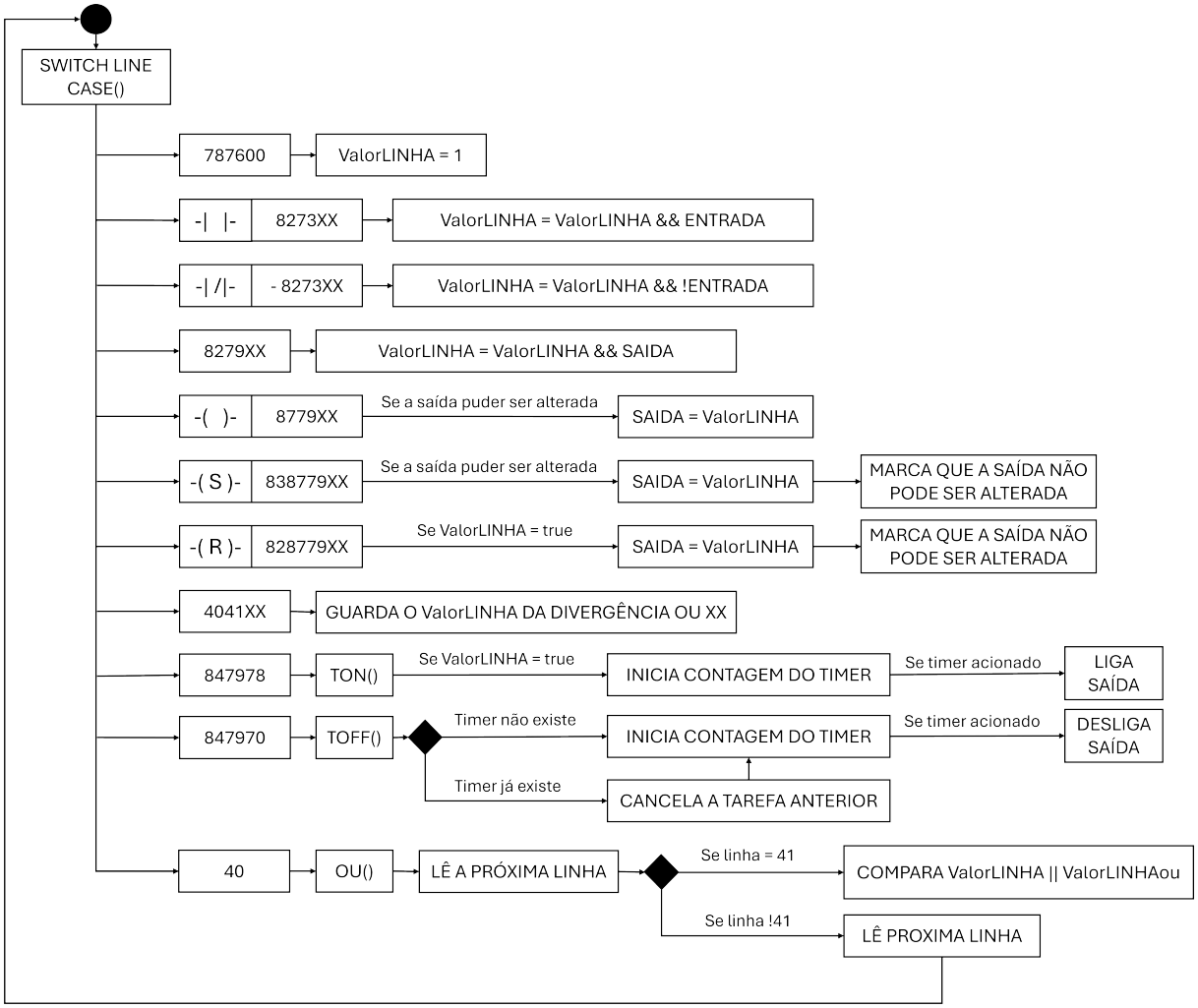
Diferente do 'Codigo_CLP.ino', o 'Codigo_CLP.h' implementa a lógica de interpretação dos comandos armazenados no arquivo TXT, através das funções:

- void InicializaCartaoSD(int CS);
- void SalvaTXT(File &myFile);
- bool InterpretaLINE(long line, File &myFile, bool &ValorLINHA);
- bool OU(File &myFile);
- bool taskCallBack(const long *Saida);
- void TON(File &myFile, bool &ValorLINHA);
- void TOFF(File &myFile);

A lógica do programa gira em torno da função principal *InterpretaLINE()*, que processa cada linha do arquivo TXT e determina o comportamento código com base no valor lido. A lógica é estruturada em um *switch case* que identifica diferentes comandos, como leitura de entradas e saídas, escrita nas saídas, configuração de temporizadores (TON e TOFF) e operações lógicas como OR.

A dinâmica do código *InterpretaLINE()* é representada no fluxograma da Figura 9.

Figura 9 – Fluxograma Função *InterpretaLINE()*



Fonte: Autor (2024).

A função *InterpretaLINE()* avalia o valor da linha e, dependendo do caso, ajusta o valor lógico acumulado *ValorLINHA*, ativa ou desativa saídas, ou agenda tarefas temporizadas com o auxílio de timers. Além disso, funções auxiliares como TON, TOFF e OU complementam a lógica, permitindo manipulações temporais e operações booleanas mais complexas. As variáveis globais das saídas são manipuladas e atualizadas no decorrer do código, assegurando que as operações lógicas considerem sempre o estado mais recente do sistema.

4 RESULTADOS

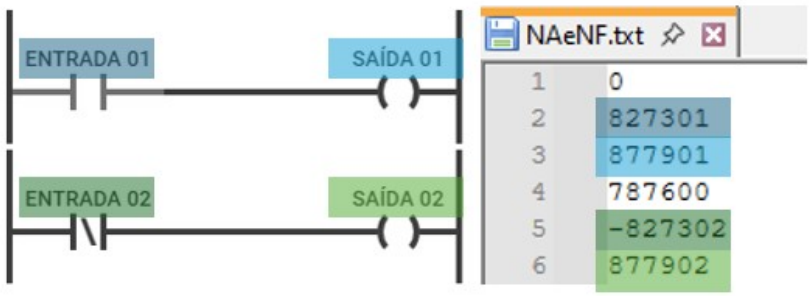
Antes de começar com o desenvolvimento das interações lógicas de um CLP, o primeiro passo foi verificar a funcionalidade de escrita e leitura no cartão SD. Durante essa etapa, foram identificados problemas relacionados à formatação do arquivo TXT, que apresentava caracteres extras quando salvo diretamente pelo Bloco de Notas do Windows. Para resolver isso, foi então utilizado o Notepad++ para converter o arquivo para o formato UNIX, garantindo que cada linha fosse finalizada apenas com '\n'. Essa padronização foi necessária, pois a função principal de leitura, readStringUntil(), processa cada linha até encontrar o caractere de nova linha.

Com o cartão de memória funcionando corretamente, foi possível começar o desenvolvimento e teste dos blocos operacionais e da lógica entre eles. Além disso, após finalizada a implementação do código, também foram exploradas maneiras de otimizar o tempo de resposta do programa.

4.1 TESTES BLOCOS LÓGICOS

Os testes dos blocos lógicos começaram pelos elementos mais básicos: os contatos normalmente abertos e fechados, verificando se o sinal elétrico era corretamente direcionado às saídas. Para isso, foi elaborado um arquivo TXT com a lógica ladder apresentada na Figura 10, que teve o comportamento esperado, como mostrado na Figura 11.

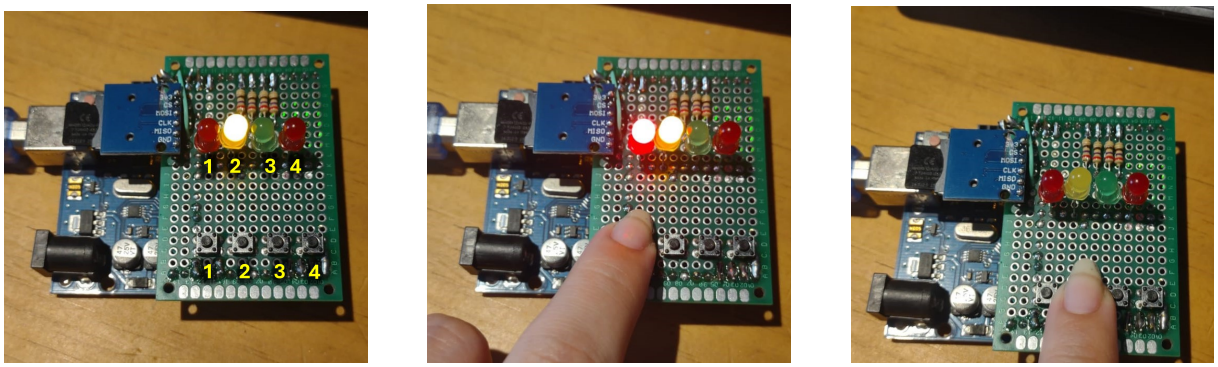
Figura 10 – Texto Interpretado para Contatos NA e NF.



Fonte: Autor (2024).

Na Figura 11 estão representadas: as saídas no seu estado normal sem nenhuma entrada acionada, Figura 11a; a saída 01 ativada após pressionar o botão 01, Figura 11b; e a saída 02 desativada após pressionar o botão 02 de contato NF, Figura 11c.

Figura 11 – Resultados do Teste dos Contatos NA e NF.



(a) Nenhuma Entrada Acionada.

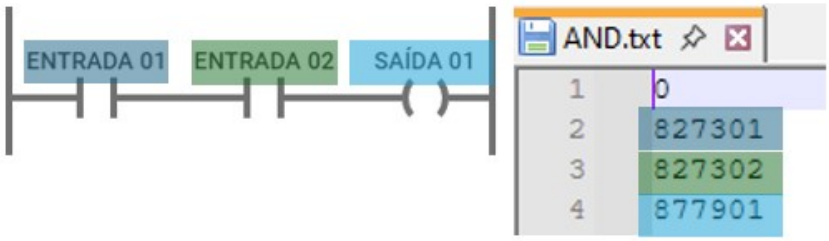
(b) Apenas Entrada 01 Acionada.

(c) Apenas Entrada 02 Acionada.

Fonte: Autor (2024).

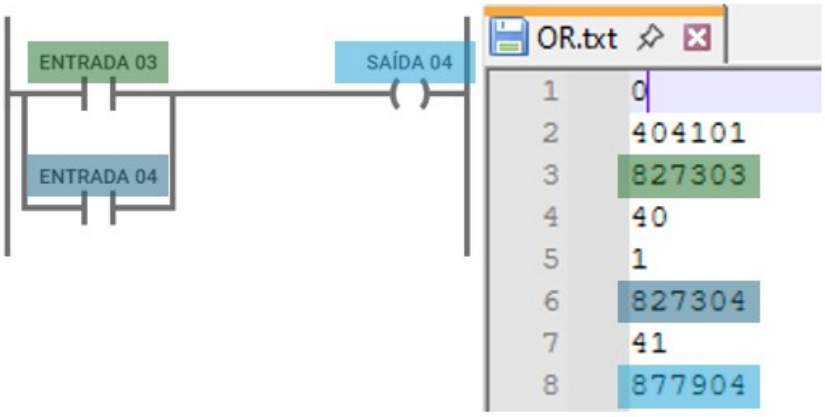
Com o funcionamento da lógica básica confirmado, foram implementadas as operações AND e OR. Os circuitos usados para estes testes estão ilustrados nas Figuras 12 e 13, respectivamente.

Figura 12 – Texto Interpretado para Operação Lógica AND.



Fonte: Autor (2024).

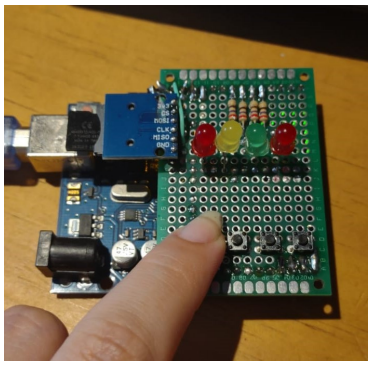
Figura 13 – Texto Interpretado para Operação Lógica OR.



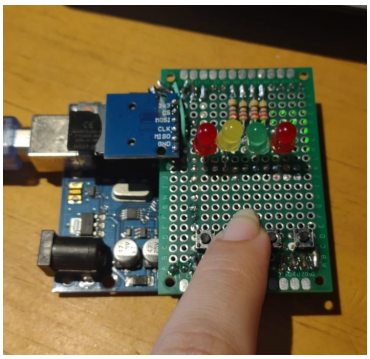
Fonte: Autor (2024).

O acionamento de uma saída pela lógica AND resultou no comportamento apresentado na Figura X, dentro do esperado.

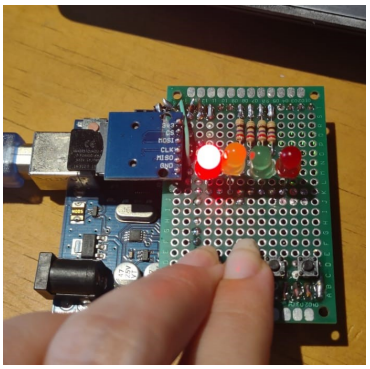
Figura 14 – Resultados do Teste da Operação Lógica AND.



(a) Apenas Entrada 01 Acionada.



(b) Apenas Entrada 02 Acionada.

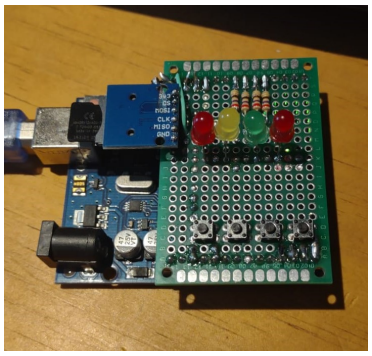


(c) Ambas Entradas Acionadas.

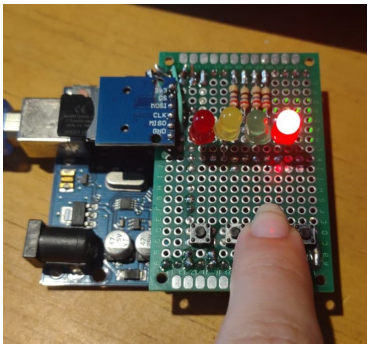
Fonte: Autor (2024).

Já o acionamento pela lógica OR também apresentou o comportamento esperado, como pode ser observado na Figura X.

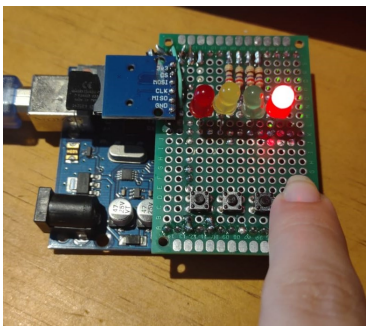
Figura 15 – Resultados do Teste da Operação Lógica OR.



(a) Nenhuma Entrada Acionada.



(b) Apenas Entrada 01 Acionada.

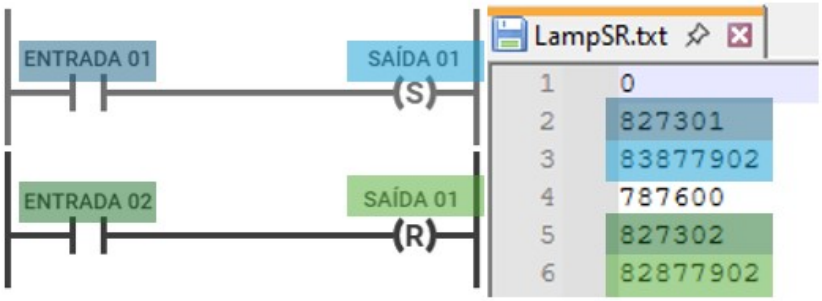


(c) Apenas Entrada 02 Acionada.

Fonte: Autor (2024).

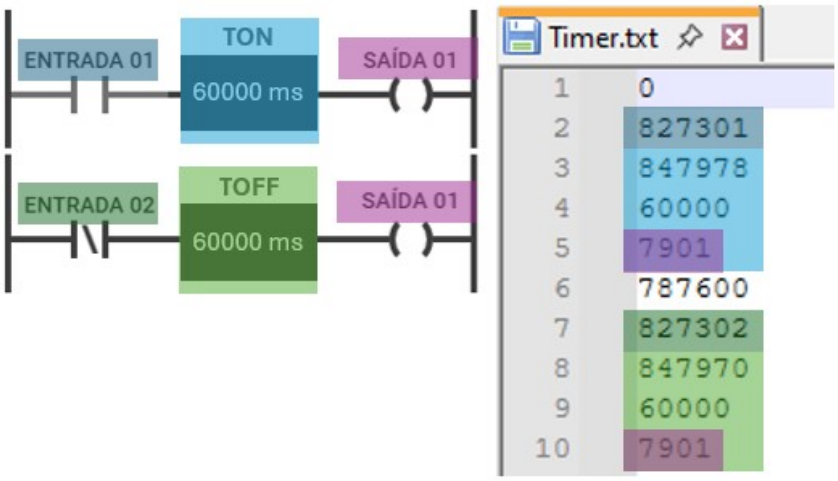
Além disso, foram desenvolvidos códigos para testar as funcionalidades de SET e RESET das bobinas, bem como os temporizadores. Os circuitos utilizados nesses testes estão representados na Figura 16 e Figura 17.

Figura 16 – Texto Interpretado para Bobinas Set/Reset.



Fonte: Autor (2024).

Figura 17 – Texto Interpretado para Bloco Temporizador.



Fonte: Autor (2024).

Como descrever textualmente o funcionamento dos circuitos é um desafio, foram gravados vídeos demonstrando cada teste realizado. Esses vídeos foram compilados em uma playlist, disponível através do link (Martins, 2024).

4.2 ANÁLISE DO TEMPO DE EXECUÇÃO DO CICLO DE SCAN

Com o funcionamento da lógica estabelecido, passou-se a analisar o tempo necessário para acionar as saídas e o consumo de memória RAM. Foi constatado que o uso excessivo de memória comprometia a execução de programas mais complexos e, em alguns casos, resultava em travamentos.

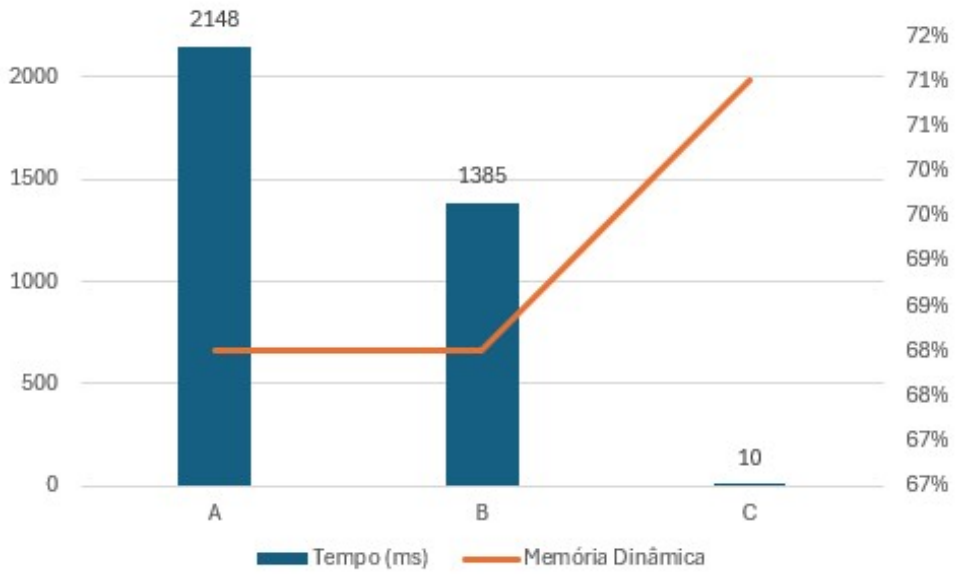
Inicialmente, foi abordado o problema do alto consumo de memória RAM, que chegava a 89% devido ao sketch. Grande parte desse consumo, cerca de 30%, estava relacionada às strings exibidas no terminal para monitorar o fluxo das interações lógicas e os sinais elétricos. A solução foi armazenar todas as strings estáticas na memória flash, reduzindo significativamente o uso de RAM.

No entanto, durante a análise do desempenho temporal, verificou-se que o excesso de impressões no terminal aumentava o tempo do ciclo de scan em 763 ms, tornando inviável o uso contínuo desse recurso. Assim, o programa foi ajustado para imprimir apenas as informações essenciais no terminal durante a fase de setup.

Visando melhorar ainda mais o tempo de execução, foi analisada a possibilidade de reduzir o número de interações de leitura do arquivo TXT. Uma das hipóteses levantadas consistia em ler o arquivo inteiro e armazenar os dados em uma variável auxiliar uma unica vez, diminuindo o uso da biblioteca SD.h. Essa abordagem mostrou-se eficaz, pois diminuiu significativamente o tempo necessário para a leitura do arquivo, otimizando o ciclo de scan do programa.

As medições de tempo e execução do ciclo foram realizadas utilizando a função *millis()* do Arduino e o mesmo circuito usado nos testes da operação lógica AND já mostrado na Figura 12. Os resultados dessas medições, tanto em termos de tempo quanto do consumo de memória RAM, estão resumidos no gráfico apresentado na Figura 18.

Figura 18 – Análise de Tempo de Execução do Ciclo de Scan.



Fonte: Autor (2024).

No gráfico, o programa A corresponde ao código com as impressões de string sendo feitas na Serial do Arduino; o programa B corresponde ao código com as leituras feitas durante o ciclo scan; e o programa C corresponde ao código com as leituras sendo feitas e armazenadas uma única vez durante o setup.

Pelo gráfico, observa-se que os ajustes realizados resultaram em uma melhoria significativa no tempo de execução, especialmente ao reduzir o número de interações com o arquivo TXT por meio da biblioteca SD.h. No entanto, no caso C, houve um aumento no uso da memória RAM devido à necessidade de criar mais variáveis

globais para armazenar uma cópia do código interpretado no próprio Arduino. Esse aumento no consumo de memória ressalta a importância de avaliar como as otimizações afetam o equilíbrio entre os recursos de hardware disponíveis e o desempenho do software, bem como a relevância de priorizar um desses aspectos em detrimento do outro dependendo da aplicação.

No contexto educacional, onde o objetivo principal é demonstrar conceitos e processos de forma didática, a prioridade é garantir que o sistema funcione de maneira previsível e permita a observação clara de seus resultados. Nesses casos, o tempo de resposta não precisa ser extremamente rápido, pois o foco deve estar na precisão lógica e na consistência dos resultados. Assim, mesmo que o uso de memória RAM seja elevado ou o tempo do ciclo de scan seja maior, isso pode ser aceitável.

Por outro lado, em aplicações que exigem desempenho em tempo real, como sistemas industriais ou automação crítica, qualquer aumento significativo no tempo de resposta pode inviabilizar o sistema. Nesses cenários, a redução do tempo de execução se torna imprescindível, mesmo que isso implique em um consumo maior de memória.

4.3 LIMITAÇÕES

É importante notar que o uso excessivo de memória pode comprometer a estabilidade do sistema, especialmente em dispositivos com recursos limitados, como o Arduino. Embora o projeto tenha mostrado resultados relevantes, existem alguns limites técnicos e práticos que precisam ser considerados.

Um dos aspectos críticos é a memória do Arduino em relação ao tamanho do arquivo TXT usado para armazenar o programa. O cartão SD pode conter programas muito maiores do que a memória RAM e Flash do Arduino conseguem suportar. Isso pode resultar em erros diversos, como:

- Travamentos do sistema devido ao esgotamento da memória RAM;
- Falhas no ciclo de scan, onde a lógica não é executada corretamente;
- Corrupção dos dados, caso a memória seja sobrecarregada sem proteção adequada;
- Perda de variáveis temporárias, que podem comprometer a execução das operações lógicas.

Um possível limite para o tamanho do arquivo TXT poderia ser definido por meio da análise das capacidades das memórias RAM e Flash do microcontrolador e pela quantidade de variáveis necessárias para armazenar o estado do sistema. Por exemplo, ao usar buffers adicionais ou estruturas temporárias, é crucial considerar o espaço necessário para cada variável e os recursos de memória adicionais.

Outro problema surge na implementação das operações lógicas OR. Alguns testes, como o acionamento do bloco OR em cenários de abrir e fechar, não foram realizados. Isso dificulta a identificação de possíveis falhas nesse cenário e compromete a confiabilidade das operações lógicas em aplicações práticas. Por exemplo, um usuário poderia abrir um bloco OR sem fechá-lo adequadamente, o que tornaria o comportamento do programa imprevisível e instável.

Essa fragilidade na implementação está diretamente relacionada à natureza do código interpretado baseado em números, que não é tão intuitivo e visual. Essa abordagem complexa pode facilitar erros de escrita e dificultar a interpretação da lógica implementada. Dado o foco educacional do projeto, essa questão representa uma barreira significativa para os alunos, pois demanda o aprendizado da estrutura e da interpretação do código em um nível mais técnico e de baixo nível antes que eles possam realmente interagir com a ferramenta de forma prática e efetiva.

Nesse cenário, seria ideal ter um programa ou dispositivo periférico que traduzisse as instruções lógicas para o formato de código interpretado de baixo nível. Essa solução facilitaria o aprendizado e o uso do sistema, tornando a experiência educacional mais prática e acessível, além de melhorar a robustez e a confiabilidade do programa.

5 CONCLUSÃO

Este trabalho teve como objetivo principal desenvolver um dispositivo lógico programável de código interpretado utilizando o Arduino UNO, buscando validar até certo ponto a aplicação de microcontroladores no controle industrial. Além disso, também se buscou demonstrar seu potencial como uma alternativa educacional acessível, adequada para utilização no ambiente acadêmico da Universidade Federal de Santa Catarina (UFSC).

O desenvolvimento do dispositivo lógico programável utilizando o Arduino UNO demonstrou ser uma solução viável e eficaz para a execução de lógica ladder em um ambiente de baixo custo. Os testes realizados validaram a capacidade do sistema de interpretar comandos armazenados em um arquivo TXT no cartão MicroSD, traduzindo-os para operações lógicas e temporais em tempo real. O uso de bibliotecas como SPI.h, SD.h e arduino-timer.h permitiu integrar de maneira eficiente a leitura, interpretação e execução de comandos, além de garantir a responsividade necessária para emular um ciclo de scan, similar ao de CLPs industriais. Esse sistema apresentou flexibilidade para manipular diferentes tipos de entradas e saídas, incluindo operações OR e temporizadores TON e TOFF, evidenciando sua robustez e potencial para aplicações educacionais.

Apesar das limitações inerentes ao Arduino UNO, como sua memória RAM reduzida, sua utilização mostrou que a aplicação pode ser facilmente adaptada para outros microcontroladores que utilizem linguagens semelhantes. Inclusive este trabalho abre caminho para diversas possibilidades de expansão em aplicações futuras. Caso haja a possibilidade de usar um microcontrolador mais potente, seria viável implementar um maior número de entradas e saídas, além de incorporar alocações dinâmicas no código, tornando-o mais genérico e adaptável. Essa abordagem eliminaria a necessidade de modificações manuais no código para adicionar ou remover periféricos, ampliando significativamente sua flexibilidade.

Além disso, o sistema poderia ser estendido para incluir o controle e leitura de sensores, como os de temperatura, expandindo suas funcionalidades para aplicações industriais e domésticas de monitoramento. Outra possibilidade seria o desenvolvimento de uma ferramenta que traduzisse automaticamente textos de alto nível para o formato estruturado utilizado neste trabalho, explorando conceitos de transformação de linguagens de alto para baixo nível. Ainda, adicionalmente, a integração com uma interface web possibilitaria o monitoramento em tempo real dos estados das entradas e saídas, além de permitir seu controle remoto, agregando valor ao sistema e expandindo suas aplicações no contexto de Internet das Coisas (IoT).

Em suma, o desenvolvimento projeto mostrou a viabilidade e o potencial dos

microcontroladores no aprendizado de conceitos de automação além de ser uma base sólida para futuras expansões, oferecendo possibilidades de adaptação a outras placas e integrações com sensores e interfaces externas.

REFERÊNCIAS

ANTONELLI, P. L. **Introdução aos Controladores Lógicos Programáveis (CLPs)**, Apostila de Curso de CLP Básico, CEETPES, 1998.

ARDUINO. **SPI**. 2024. Disponível em: https://docs.arduino.cc/language-reference/en/functions/communication/SPI/.

ARDUINO JOSé BAGUR, T. C. **Arduino Memory Guide**. 2023. Disponível em: https://docs.arduino.cc/learn/programming/memory-guide/.

ARDUINO, S. **SD**. 2024. Disponível em: https://docs.arduino.cc/libraries/sd/.

CAPELLI, A. **CLP Controladores lógicos programáveis na prática**. São Paulo: Antena, 2007.

CONTRERAS, M. **arduino-timer - library for delaying function calls**. 2023. Disponível em: https://github.com/contrem/arduino-timer.

BORTOLI FáVERO, E. M. de. **Organização e Arquitetura de Computadores**. Curitiba: e-Tec Brasil, 2013.

GIL, A. C. **Métodos e técnicas de pesquisa social.** São Paulo: Atlas, 2008.

INTERNATIONAL ELECTROTECHNICAL COMMISSION. **IEC 61131-3**: Programmable controllers – part 3: Programming languages. [S.l.], 2013. v. 3º ed.

MARTINS, M. D. **Testes TCC Mariana Dias Martins**. 2024. Disponível em: https://www.youtube.com/playlist?list=PLPDzjjCsA_PWf09QY_uphP-V7DWFa97V_.

MENDES, R. M. **Programação de CLPs. Métodos e Técnicas**. São Paulo: Scienza, 2021.

NOGUEIRA, A. L.; NOVAES, F. F. Arduino como interface em automação industrial: capacidades e limitações. *In*: CISTI, 17 a 20 de junho de 2015, Aveiro, Portugal. **Sistemas e Tecnologias de Informação**. 2015. p. 199–201. Disponível em: https://web.s.ebscohost.com/ehost/pdfviewer/pdfviewer?vid=0&sid=1edfa197-516e-4b88-9463-e4afb5b0836a%40redis. Acesso em: 8 nov. 2023.

OLIVEIRA, A. D. de. IntroduÇÃo aos controladores lÓgicos programÁveis. Apostila Curso Técnico de Eletromecânica, CEFET, Pelotas, RS, p. 31, 2017.

PRUDENTE, F. **Automação Industrial PLC: Teoria e Aplicações**. Rio de Janeiro, RJ: LTC — Livros Técnicos e Científicos, 2011.

RIBEIRO, M. A. **Automação Industrial**. Salvador, BA: Tek Treinamento Consultoria, 2005.

SILVA, G. P. da. **PLC – Controladores Lógicos Programáveis**, SENAI – CFPOC, Pouso Alegre„ p. 12, 2015.