

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA**

HENRIQUE DA SILVA BENTO

**DESENVOLVIMENTO DE UMA INTERFACE G-CODE EMBARCADA PARA
MÁQUINAS CNC**

FLORIANÓPOLIS, 2023.

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA**

HENRIQUE DA SILVA BENTO

**DESENVOLVIMENTO DE UMA INTERFACE G-CODE EMBARCADA PARA
MÁQUINAS CNC**

Trabalho de Conclusão de Curso submetido
ao Instituto Federal de Educação, Ciência
e Tecnologia de Santa Catarina como parte
dos requisitos para obtenção do título de
Engenheiro em eletrônica.

Orientador:
Prof. Renan Augusto Starke, Dr. Eng

FLORIANÓPOLIS, 2023.

Ficha de identificação da obra elaborada pelo autor.

Bento, Henrique da Silva
**DESENVOLVIMENTO DE UMA INTERFACE G-CODE EMBARCADA
PARA MÁQUINAS CNC** / Henrique da Silva Bento; orientação de
Renan Augusto Starke. - Florianópolis, SC, 2023.
84 p.

Trabalho de Conclusão de Curso (TCC) - Instituto Federal
de Santa Catarina, Câmpus Florianópolis. Bacharelado
em Engenharia Eletrônica. Departamento Acadêmico
de Eletrônica.
Inclui Referências.

**1. Interface. 2. GRBL. 3. G-code. 4. CNC. I. Starke,
Renan Augusto. II. Instituto Federal de Santa Catarina.
III. DESENVOLVIMENTO DE UMA INTERFACE G-CODE EMBARCADA
PARA MÁQUINAS CNC .**

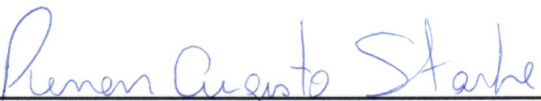
DESENVOLVIMENTO DE UMA INTERFACE G-CODE EMBARCADA PARA
MÁQUINAS CNC

HENRIQUE DA SILVA BENTO

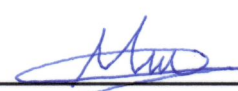
Este trabalho foi julgado adequado para obtenção do título de Engenheiro em Eletrônica e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 15 de dezembro, 2023.

Banca Examinadora:


Prof. Renan Augusto Starke, Dr. Eng.


Prof. Daniel Lohmann, Msc. Eng.


Prof. Matheus Leitzke Pinto, Msc. Eng.

Dedico este trabalho a todos que, de alguma maneira,
contribuíram para minha formação acadêmica.

Agradeço à minha família, professores e amigos
pelo apoio constante e pela significativa colaboração
ao longo desta jornada.

AGRADECIMENTOS

Gostaria expressar minha profunda gratidão aos meus pais, Ailton Bento e Maurina da Silva Bento, por seu inabalável apoio ao longo de todos esses anos. Seu suporte foi fundamental para que eu chegasse ao fim desta jornada acadêmica. À minha namorada, Maria Albertina de Moraes, meu alicerce e companheira incansável durante o desenvolvimento deste TCC, agradeço por seu suporte inestimável. A todos os professores e colegas que integraram essa significativa caminhada na faculdade, meu sincero agradecimento. Cada um de vocês desempenhou um papel crucial na minha formação, e por isso sou profundamente grato. Gostaria de expressar minha sincera gratidão à empresa Metal Protector por seu generoso apoio ao longo da execução deste projeto. Agradeço por fornecer todas as ferramentas necessárias para a realização e conclusão bem-sucedida deste trabalho. O suporte da empresa foi fundamental para o desenvolvimento deste projeto, e estou verdadeiramente grato pela parceria estabelecida. Não posso concluir este agradecimento sem expressar minha profunda gratidão ao Professor Orientador, Dr. Eng. Renan Augusto Starke. Seu conhecimento, orientação e participação ativa em todas as reuniões foram cruciais para o sucesso e conclusão deste projeto. Suas valiosas dicas e *insights* foram fundamentais para minha jornada acadêmica e para o desenvolvimento deste trabalho. Agradeço sinceramente por sua orientação e apoio dedicados.

“A imaginação é mais importante que a ciência, por que a ciência é limitada, ao passo que a imaginação abrange o mundo inteiro.”

Albert Einstein

RESUMO

Este trabalho descreve o projeto e a implementação de uma interface G-Code *Sender* para uma máquina Fresadora ou Laser CNC (Controle Numérico Computadorizado) implementada com *firmware* GRBL. A CNC é uma ferramenta essencial na fabricação automatizada, permitindo a criação precisa e repetível de peças e produtos. O *firmware* GRBL é uma escolha comum para controlar CNCs devido à sua eficiência e capacidade de processar comandos G-Code de maneira ágil e por ser um software de código aberto. O foco deste projeto é proporcionar uma solução que permita a operação autônoma da CNC, sem a necessidade de uma conexão constante com um computador. Para isso, foi desenvolvida uma interface G-Code *Sender* que realiza a leitura e processa os arquivos G-code, além de possuir instruções e configurações de movimentação da CNC diretamente a partir de um dispositivo de armazenamento local, como um cartão SD ou USB. Isso aumenta a flexibilidade e a portabilidade do sistema, eliminando as limitações de conectividade. O projeto foi dividido em várias etapas. Inicialmente, foi realizada uma pesquisa detalhada sobre o *firmware* GRBL, seus requisitos de hardware e protocolo de comunicação. Em seguida, foram selecionados os componentes eletrônicos adequados para a montagem da interface G-Code *Sender*, incluindo o microcontrolador ESP32, a interface de usuário com o *display touchscreen* de 7 polegadas e outros métodos de IHM (Interface Homem Máquina). Uma vez selecionados os componentes, foi realizada a implementação do hardware e do *firmware* da interface. Assim, os resultados finais do projeto serão analisados e demonstrados para verificar a viabilidade e eficácia da controladora *offline*. A CNC foi capaz de executar operações de usinagem a partir dos comandos G-Code armazenados no dispositivo de armazenamento local, sem a necessidade de estar conectada a um computador.

Palavras-chave: Interface G-Code Sender. Automação. Controle Numérico Computadorizado. GRBL. Máquina CNC.

ABSTRACT

This work describes the design and implementation of a G-Code Sender for a Milling or Laser CNC (Computer Numerical Control) machine with GRBL firmware. CNC is an essential tool in automated manufacturing, enabling the precise and repeatable creation of parts and products. GRBL is a common choice for controlling CNCs due to its efficiency and ability to process G-Code commands quickly and it is an open source software. The focus of this project is to provide a solution that allows autonomous operation of the CNC, without needing a constant computer connection. To this end, a G-Code Sender interface was developed that reads and processes G-code files directly from a local storage device such as an SD or USB card, and it can also send moving and settings instructions. This increases system flexibility and portability by eliminating connectivity limitations. The project was divided into several stages. Initially, detailed research was carried out on firmware GRBL its hardware requirements and communication protocol. Next, the appropriate electronic components were selected for assembling the G-Code Sender interface, including the ESP32 microcontroller, the user interface with the 7-inch touchscreen display and other HMI (Human Machine Interface) methods. Once selected, the components, the implementation of the hardware and firmware of the interface will be carried out. The results of the project were analyzed and demonstrated to verify the viability and effectiveness of the offline controller. The CNC is able to perform machining operations from the G-Code commands stored on the local storage device, without the need to be connected to a computer.

Keywords: G-Code Sender interface. Autonomy. Computer Numerical Control. GRBL. CNC machine.

LISTA DE FIGURAS

Figura 1 – Diagrama de blocos CNC	20
Figura 2 – Empresas que fornecem suporte a GRBL	23
Figura 3 – <i>Display</i> Inteligente P80480V70C T07 Proculus	29
Figura 4 – <i>Display</i> Inteligente NX8048T070 Nextion	30
Figura 5 – Layout UnicView AD	32
Figura 6 – Barra de elementos UnicView AD	32
Figura 7 – Diagrama de blocos do sistema	35
Figura 8 – Camadas ESP32	36
Figura 9 – Pinos e configurações da ESP 32-S3	38
Figura 10 – Parte 1 do esquemático final	39
Figura 11 – Parte 2 do esquemático final	40
Figura 12 – Layout PCI	41
Figura 13 – Placa pronta não montada	42
Figura 14 – Placa pronta montada	43
Figura 15 – Representação do Sistema Implementado	44
Figura 16 – Telas da IHM	46
Figura 17 – Interação entre as tarefas para realizar a troca de uma página	48
Figura 18 – Interação entre as tarefas para realizar a atualização dos estados da máquina	49
Figura 19 – Conexão de um dispositivo USB	50
Figura 20 – Projeto da Interface aberto no <i>software</i> Unic View AD	52
Figura 21 – Aba de configurações Unic View AD	53
Figura 22 – Área de controle manual da página inicial	53
Figura 23 – Exemplo de teclado numérico implementado	54
Figura 24 – Elementos equivalentes aos declarados no Código 3.4	55
Figura 25 – Botão para acesso as páginas de configuração do sistema	59
Figura 26 – Diagrama de sequência para realizar a troca de página <i>Display</i>	60
Figura 27 – Fluxograma de funcionamento das funções de comunicação GRBL	62
Figura 28 – Fluxograma de funcionamento sistema USB	66
Figura 29 – Ligações para o teste da Interface G-code <i>Sender</i>	72
Figura 30 – <i>Setup</i> de testes da Interface G-code <i>Sender</i>	73
Figura 31 – <i>Layout</i> desenvolvido no Kicad feito para testes da Interface G-code <i>Sender</i>	74
Figura 32 – <i>Software</i> Flat-cam com a rotina de teste gerada a partir do arquivo de furos	75
Figura 33 – Processo de furação da PCI com a Interface G-code <i>Sender</i> desenvolvida	76

Figura 34 – Furos realizados na PCI ao final dos testes 77

Figura 35 – *Software* bCNC com o G-code da rotina de teste 78

LISTA DE QUADROS

Quadro 1 – Tipos de atuadores e suas vantagens e desvantagens	22
Quadro 2 – Tecnologias selecionadas para o Projeto	33
Quadro 3 – Ligações do ESP32 S3 Dev Kit C 1 v1.1 na PCI	38
Quadro 4 – Tabela de Comandos GRBL	61

LISTA DE TABELAS

Tabela 1 – Tabela com os custos de todas as tecnologias empregadas no projeto 80

LISTA DE ABREVIATURAS E SIGLAS

ADC	<i>Analog to Digital Converter</i> (Conversor Analógico para Digital)
BLE	<i>Bluetooth Low Energy</i> (Bluetooth de baixa energia)
CAD	<i>Computer-Aided Design</i> (Design Assistido por Computador)
CAM	<i>Computer-Aided Manufacturing</i> (Fabricação Assistida por Computador)
CNC	Controle Numérico Computadorizado
ESP-IDF	<i>Espressif IoT Development Framework</i> (Estrutura de desenvolvimento Espressif IoT)
GPIOs	<i>General Purpose Input/Output</i> (Entrada/saída de uso geral)
GRBL	<i>G-code Real-time Behavior Engine</i> (Mecanismo de comportamento em tempo real com código G)
I2C	<i>Inter-Integrated Circuit</i> (Circuito Inter-integrado)
IFSC	Instituto Federal de Santa Catarina
IHM	Interface homem-máquina
IOs	<i>Inputs and outputs</i> (Entradas e saídas)
IoT	<i>Internet of Things</i> (Internet das Coisas)
MIT	Instituto de Tecnologia de Massachusetts
MSC	<i>Mass Storage Class</i> (Classe de armazenamento em massa)
PCI	Placa de Circuito Impresso
RTOS	<i>Real-Time Operating System</i> (Sistema operacional em tempo real)
SDK	<i>Software Development Kit</i> (Kit de desenvolvimento de <i>hardware</i>)
SPI	<i>Serial Peripheral Interface</i> (Interface periférica serial)
UART	<i>Universal Asynchronous Receiver-Transmitter</i> (Receptor-transmissor assíncrono universal)
USART	<i>Universal Synchronous/Asynchronous Receiver-Transmitter</i> (Receptor-transmissor universal síncrono/assíncrono)
VPs	Pontos de Valor

SUMÁRIO

1	INTRODUÇÃO	15
1.1	Justificativa	16
1.2	Definição do Problema	16
1.3	Objetivo Geral	17
1.4	Objetivos Específicos	17
1.5	Estrutura do Trabalho	18
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	CNC (Controle Numérico Computadorizado)	19
2.1.1	Fresadora/Laser	21
2.1.2	Sistema de controle de motores de passo e servo motores	21
2.1.3	Firmware GRBL	22
2.1.4	G-Code	23
2.1.4.1	<i>Estrutura e Funcionamento</i>	24
2.1.4.2	<i>Geração e Edição</i>	24
2.2	Protocolo de Comunicação Serial	25
2.3	ESP32	26
2.3.1	ESP-IDF	27
2.4	FreeRTOS	28
2.5	<i>Display</i> Inteligente	29
2.5.1	Software UnicView AD e conexão com o <i>hardware</i>	31
2.6	Tecnologias do Projeto	32
3	DESENVOLVIMENTO	34
3.1	Metodologia	34
3.1.1	Métodos Aplicados	34
3.2	Projeto da Placa de Circuito Impresso (PCI)	37
3.2.1	Kit de Desenvolvimento ESP32 S3 Dev Kit C 1 v1.1	37
3.2.2	Esquemático, Layout e Modelo Final	39
3.3	Funcionamento do sistema e suas tarefas	43
3.4	Comunicação com o <i>Display</i>	50
3.4.1	<i>Buffer</i> Circular e uso da Biblioteca do Hardware	55
3.4.2	Máquina de Estado para controle das páginas e interação com Elementos	58
3.5	GRBL comunicação	60
3.6	Sistema de Leitura de dispositivos de armazenamento USB	65
4	RESULTADOS	72
4.1	Testes	72
4.2	Custos	79
5	CONSIDERAÇÕES FINAIS	82
5.1	Sugestões para Trabalhos Futuros	83
	REFERÊNCIAS	84

1 INTRODUÇÃO

Desde a Revolução Industrial, a busca por produção mais eficiente levou a evolução da eficácia e funcionalidade das máquinas desenvolvidas e essa busca por melhora de desempenho em algum ponto levou a necessidade e o surgimento das máquinas CNC (TOOL; ENGINEERS; WILSON, 1963), que produzem peças complexas com alta precisão. No entanto, a dependência de conexão com computadores com porta paralela limitou sua autonomia e aumentou custos. O *firmware* GRBL trouxe uma nova possibilidade a esse cenário ao permitir controle ágil e em tempo real através de comandos via comunicação serial (BENGLER, 2014). O GRBL (*G-code Real-time Behavior Engine*) é um *firmware* embarcado que foi desenvolvido para ser executado no microcontrolador ATmega 328P presente na placa de desenvolvimento Arduino UNO. Entretanto, ao longo do tempo, os computadores de mesa, responsáveis por enviar comandos e controlar as máquinas, tornaram-se um obstáculo devido à falta de praticidade em termos de mobilidade, produção em grande escala, custos e outras questões. Diante dessas problemáticas, as novas tecnologias possibilitam o desenvolvimento de uma interface mais acessível, capaz de substituir o computador de mesa de maneira eficaz, porém com uma série de vantagens adicionais.

Para substituir o computador foi proposto a opção de um dispositivo capaz de realizar a leitura dos arquivos G-code e assumir integralmente o controle das configurações do GRBL. O dispositivo descrito é uma interface G-Code *Sender*, para o seu desenvolvimento escolheu-se a família de microcontroladores ESP32 (ESPRESSIF SYSTEMS, 2023c) e o *framework* ESP-IDF (ESPRESSIF SYSTEMS, 2023a). O microcontrolador selecionado será responsável por gerenciar o recebimento e envio de comandos de controle como trocas de tela via comunicação serial com o *display* inteligente de cristal líquido(LCD) *touch*, facilitando a interação usuário-máquina. A comunicação serial segue o padrão do *firmware* GRBL (GRBL, 2021). Outro padrão muito utilizado em máquinas CNC, o G-Code ou também conhecido como RS-274, foi desenvolvido pela EIA (*Electronic Industries Association*) no início da década de 1960 e finalmente padronizado pela ISO em fevereiro de 1980 como RS274D/ISO 6983. É amplamente utilizado por ser um padrão genérico de comunicação e utilizado por várias controladoras, assim garantindo uma boa compatibilidade entre interfaces e máquinas, obtendo uma coordenação precisa dos movimentos da CNC (KRAMER; PROCTOR; MESSINA, 2000).

A escolha da família de microcontroladores ESP32 e a implementação do protocolo serial são fundamentais para o funcionamento eficaz da interface G-Code *Sender*. Os microcontroladores da família ESP32 permitem a utilização de um sistema operacional, além de já contarem com a implementação da leitura de dispositivo flash USB pela fabricante. Isso facilita o desenvolvimento do projeto proposto e abre

possibilidades para futuras funcionalidades, como a integração de uma conexão Wi-Fi para uma interface *online*.

Resumindo, o projeto da interface G-Code *Sender* foca na comunicação entre operador, *firmware* GRBL e CNC, usando o ESP32 e protocolo serial para aprimorar eficiência e autonomia.

1.1 Justificativa

Propõe-se neste trabalho explorar uma abordagem já existente, mas ainda muito hobista e não voltada a indústria: o desenvolvimento de uma interface G-Code *Sender* para CNC, que, embora não execute o *firmware* GRBL diretamente, atua como uma ponte entre o operador e a máquina. Ao enviar comandos para uma placa que executa o *firmware* GRBL e controla a mecânica da CNC, a interface G-Code *Sender* busca emancipar a máquina da necessidade de conexão constante com um computador. Há uma possível redução do custo tornando possível uma produção em maior escala de uma controladora CNC integrada que possibilita ao cliente desenvolver apenas a parte mecânica. Além disso, algumas interfaces ou softwares ligados ao computador tendem a demonstrar uma certa complexidade que dificultam o entendimento do usuário com a máquina, então a possibilidade de desenvolver uma interface simples que atende a todas as necessidades deixando a máquina mais ágil e independente traz um grande ganho produtivo. Assim, esse trabalho justifica ser desenvolvido pelos ganhos que proporciona e pela possibilidade de trazer versatilidade para máquinas já instaladas que trabalham com GRBL trazendo independência, corte de gastos, redução de espaço da máquina, simplificação de operação e versatilidade de transporte entre outros pontos.

1.2 Definição do Problema

O *firmware* GRBL contempla a capacidade de operar por meio de comunicação serial, descartando assim o uso da porta paralela de computadores mais antigos, que era exigido por outras controladoras, e introduzindo novas possibilidades de interface. Entretanto, as aplicações tradicionalmente ainda dependem de computadores de mesa para atuar como a interface dessa comunicação. A justificativa para a abordagem com *desktops* repousa na considerável capacidade computacional que estes oferecem, assim proporcionando facilidade no desenvolvimento de *firmwares* capazes de atender todas as funcionalidades requeridas. Entretanto, a dependência de um computador acarreta uma série de desvantagens, a exemplo das dimensões físicas que tal sistema demanda, a necessidade de periféricos como mouse, teclado e monitor, os custos associados ao conjunto computacional, a complexidade da interface que, frequentemente, não é de caráter amigável, e a comercialização de um produto global,

uma vez que muitas vezes as máquinas CNC são comercializadas acompanhadas de *hardware* destinado à execução no computador, mas esse componente não é fornecido.

A adoção de uma interface G-Code *Sender* incorporada em um microcontrolador surge como uma solução eficiente para simplificar o aparato associado ao controle de máquinas CNC. Essa abordagem oferece uma alternativa financeiramente melhor, em formato compacto, com interface de fácil utilização, mantendo resultados equivalentes aos obtidos por meio de um computador. Em resumo, a escolha de uma interface G-Code *Sender* representa uma alternativa aos desafios do uso de computadores como interface para o *firmware* GRBL. Diante dos benefícios advindos da utilização de uma interface G-Code *Sender* incorporada em um microcontrolador, como pode ser implementado tal dispositivo para atender às necessidades do projeto de modo a representar uma perspectiva inovadora na resolução de problemas, proporcionando uma solução distinta que não compromete a qualidade e o desempenho na operação de máquinas CNC?

1.3 Objetivo Geral

O objetivo geral deste projeto é desenvolver e implementar uma interface G-Code *Sender* para máquinas CNC (Controle Numérico Computadorizado) do tipo Fresadora ou Laser, que empregam o uso do *firmware* de controle de movimentos GRBL. A interface proposta permitirá a operação autônoma da CNC, eliminando a dependência constante de um computador, além de proporcionar uma interface simplificada para interação homem-máquina.

1.4 Objetivos Específicos

- a) Realizar uma pesquisa sobre o GRBL, incluindo seus requisitos de *hardware* e protocolo de comunicação.
- b) Aprofundar conhecimento e desenvolver a interface G-Code *Sender* usando o microcontrolador ESP32 com o *framework* ESP-IDF e o sistema operacional FreeRTOS.
- c) Desenvolver funções para implementar e tratar um protocolo de comunicação entre a controladora, o sistema GRBL e os protocolos do G-Code, permitindo a transmissão eficiente de comandos para a máquina CNC.
- d) Realizar o controle dos três eixos individuais da máquina bem como realizar as configurações do GRBL.
- e) Desenvolver o sistema de funções responsável por processar e enviar comandos G-Code, a partir de um dispositivo de armazenamento local, como um Pen Drive.

- f) Avaliar a viabilidade e eficácia da controladora *offline*, por meio da análise dos resultados obtidos na operação da CNC em comparação com os métodos tradicionais que empregam computadores.
- g) Validar a simplicidade e a praticidade da interface homem-máquina implementada no *display touchscreen*, visando melhorar a usabilidade e a compreensão por parte do operador.
- h) Contribuir para a disseminação de uma solução tecnológica que proporcione autonomia, eficiência e redução de custos no controle de máquinas CNC, explorando as vantagens do *firmware* GRBL e do microcontrolador ESP32.

1.5 Estrutura do Trabalho

Este trabalho concentra-se no desenvolvimento de uma interface para comunicação e controle de uma controladora CNC que utiliza o *hardware* GRBL como seu sistema de controle de movimentos.

O primeiro capítulo oferece uma visão geral do tema abordado, sua justificativa, a definição do problema em foco e os objetivos gerais e específicos do trabalho.

No capítulo seguinte, é apresentada uma pesquisa detalhada sobre o projeto especificado e sua origem, fornecendo um referencial teórico. Explora o contexto das máquinas CNC e suas origens, o *firmware* GRBL, o protocolo G-code, tipos de protocolo de comunicação e seu funcionamento. Também aborda a família de microcontroladores utilizada, especificamente a família ESP32, detalhando todo o seu *framework*. Além disso, discute o sistema operacional FreeRTOS, destacando suas características principais, e explora os *displays* inteligentes, que desempenham um papel crucial no desenvolvimento da interface homem-máquina do projeto.

No terceiro capítulo, é apresentada a metodologia utilizada no trabalho, destacando o tipo de pesquisa e os métodos aplicados. Em seguida, são detalhados os principais componentes do projeto, incluindo a comunicação com o *display*, a interação com o *firmware* GRBL, a leitura e integração com dispositivos USB, e como esses módulos interagem entre si, utilizando o sistema de tarefas implementado pelo FreeRTOS.

No capítulo seguinte são apresentados os resultados obtidos no projeto, comparando-os com ferramentas compatíveis já existentes no mercado. Finalizando, no último capítulo, são apresentadas as conclusões sobre o projeto, incluindo possíveis melhorias para futuras implementações.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, apresenta-se uma exposição detalhada de temas, conteúdos e terminologias essenciais que contribuem para a compreensão deste trabalho. A abordagem visa contextualizar o tema central e elucidar as ferramentas fundamentais que constituem a base para a elaboração do projeto. Serão abordados aspectos pertinentes aos elementos de *software* e *hardware* que possibilitam a concretização do projeto, englobando os respectivos *firmwares* e componentes eletrônicos que viabilizam a execução das funcionalidades propostas.

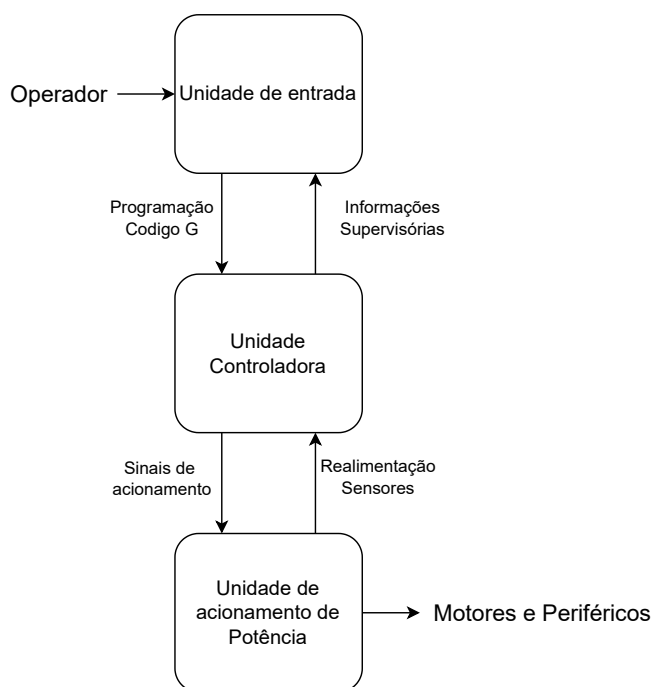
2.1 CNC (Controle Numérico Computadorizado)

As máquinas CNC (Controle Numérico Computadorizado) são dispositivos utilizados na fabricação automatizada para produzir peças com alta precisão e complexidade. Essas máquinas são controladas por computadores e são programadas para seguir instruções específicas, conhecidas como códigos G-Code, para executar operações de corte, fresagem, perfuração, entre outras, em materiais como metal, madeira e plástico. A história da CNC é intrinsecamente ligada ao desenvolvimento tecnológico e à automação da fabricação, promovendo avanços notáveis na indústria. O conceito de controle numérico, o precursor da CNC, teve seus primeiros passos no ano de 1947, quando a necessidade de produzir componentes complexos para aeronaves que demandam uma certa precisão exigiu uma abordagem mais sistemática, a força aérea americana reconheceu nesta ferramenta uma possibilidade de melhorar e ampliar o desenvolvimento da produção de seus aviões assim resolveu entregar a *Parsons Corporation* um contrato para realizar um projeto que se tornaria a primeira máquina CNC (SEAMES, 2001). A ideia era utilizar instruções numéricas para guiar o movimento das ferramentas de usinagem, substituindo a operação manual e sujeita a erros. O ponto de partida histórico ocorreu em 1955, quando *Parsons Corporation* desenvolveu o primeiro protótipo de controle numérico no Instituto de Tecnologia de Massachusetts (MIT) para criar peças complexas de forma automatizada e sendo assim após alguns refinamentos ficou disponível para a indústria (SEAMES, 2001). Entretanto, foi posteriormente que a verdadeira revolução CNC começou a tomar forma. A introdução dos computadores e a capacidade de processamento digital abriram caminho para uma nova geração de máquinas CNC. O desenvolvimento de linguagens de programação específicas para controle numérico, como o G-Code, tornou possível a criação de sequências detalhadas de instruções para guiar o movimento dos eixos. Essa abordagem permitiu maior flexibilidade, precisão e repetibilidade na fabricação. Com o avanço da eletrônica e da microeletrônica nas décadas seguintes, as máquinas CNC se tornaram mais acessíveis e eficientes. A capacidade de controlar múltiplos eixos, combinada com o uso de ferramentas de corte cada vez mais avançadas, abriu

novas possibilidades para a fabricação de componentes complexos em diversas indústrias, desde a automobilística até a aeroespacial. A CNC desempenhou também um papel fundamental na redução de custos e no aumento da produtividade, tornando a produção em massa mais viável.

No cenário atual, a CNC continua a evoluir com avanços tecnológicos como a automação, a robótica e a inteligência artificial. Máquinas CNC altamente sofisticadas podem realizar uma ampla variedade de operações, desde corte e perfuração até usinagem em 5 eixos, impressão 3D e corte a laser. Além disso, a integração com sistemas CAD/CAM (*Computer-Aided Design/Computer-Aided Manufacturing*) permite a criação eficiente de programas de usinagem e a simulação virtual de operações complexas (GENG, 2004). Uma das tendências recentes é a integração da CNC com tecnologias de conectividade, permitindo o monitoramento e controle remoto das máquinas. Além disso, a busca por maior autonomia e independência levou ao desenvolvimento de interface G-Code *Sender*, como a abordagem proposta neste trabalho, que visa proporcionar uma solução mais flexível e portátil para a operação de máquinas CNC. Na Figura 1, é ilustrado o esquema operacional de uma máquina CNC, onde a unidade de entrada se configura como a interface a ser desenvolvida no âmbito deste projeto. Essa interface estabelecerá a comunicação com qualquer unidade controladora capaz de executar o *firmware* GRBL, responsável por coordenar os acionamentos dos mais diversos tipos de motores empregados, culminando na concretização da peça desejada.

Figura 1 – Diagrama de blocos CNC



Fonte: Adaptado de (CARSTENS; CARSTENS, 2015).

2.1.1 Fresadora/Laser

Uma fresadora é uma máquina CNC que realiza operações de usinagem, como cortes e desbastes, em materiais sólidos, geralmente utilizando uma ferramenta de corte rotativa. Já uma máquina laser CNC emprega um feixe de laser para cortar ou gravar materiais. Ambas as máquinas são amplamente utilizadas na indústria de fabricação e prototipagem para criar peças e produtos com alta precisão e qualidade.

2.1.2 Sistema de controle de motores de passo e servo motores

Os motores de passo são dispositivos amplamente empregados em máquinas CNC de pequeno e médio porte para controlar os movimentos dos eixos. Um sistema de controle de motores de passo consiste em eletrônica e *software* que converte comandos do controlador em pulsos elétricos que movimentam o motor de passo em incrementos precisos. Isso permite um controle preciso sobre o posicionamento e movimentação da máquina CNC. Já para máquinas CNCs maiores e que exigem um torque maior, o uso de servo motor é uma tecnologia amplamente empregada para controle preciso em projetos. Embora inicialmente associado a aplicações especiais que demandam controle preciso de torque, velocidade e posição, sua crescente disponibilidade e redução de custos o tornaram uma alternativa vantajosa em comparação com motores de indução, atuadores hidráulicos e pneumáticos. Essa mudança no cenário ampliou significativamente a adoção de servo motores em uma variedade de contextos industriais, oferecendo maior precisão e desempenho em sistemas automatizados (SILVEIRA, 2016). O Quadro 1 demonstra alguns tipos de motores utilizados na indústria e suas principais vantagens e desvantagens assim cada motor se adequa a um tipo de projeto.

Quadro 1 – Tipos de atuadores e suas vantagens e desvantagens

Tipo de Acionamento		Pontos Fortes	Pontos Fracos
Dispositivo Mecânico	Dispositivo Hidráulico / Pneumático	– Estrutura simples – Preço competitivo	– Baixa acurácia – Pode produzir vibração – Pode poluir o ambiente
Dispositivo Eletromecânico	Motor de Passo	– Controlador simples – Preço competitivo	– Altos ruídos e vibrações – Não possui tamanhos elevados – Não se aplica a alta precisão
	Motor de Indução	– Preço competitivo – Alta precisão no controle – Torques altos e Força motriz elevada – Ruídos e vibrações baixos – Excelente manutenção – Alta velocidade e precisão no controle – Sem conversão mecânica	– Preço um pouco elevado
	Servo Motor		
	Motor Linear		– Preço elevado

Fonte: (SILVEIRA, 2016).

2.1.3 Firmware GRBL

O *firmware* GRBL é um elemento crucial no ecossistema das máquinas CNC. Desenvolvido para operar como um *firmware* intermediário entre a máquina física e as instruções de controle, o GRBL desempenha um papel fundamental ao interpretar comandos G-Code em tempo real e coordenar os movimentos dos eixos da CNC. Sua arquitetura eficiente desenvolvida em C altamente otimizado e capacidade de processamento rápido o tornam uma escolha amplamente adotada por operadores e fabricantes de máquinas CNC e possui um tipo de licença GPLv3 (GRBL, 2021). A Figura 2 abaixo demonstra algumas empresas que já fornecem suporte para o GRBL.

Figura 2 – Empresas que fornecem suporte a GRBL



Fonte: (GRBL, 2021).

Segundo o GRBL (2021), o mesmo opera como um *firmware* embarcado que foi inicialmente desenvolvido para microcontroladores AVR. Teve como foco o Atmega328P com a plataforma Arduino por sua popularidade e diversos recursos que a família AVR oferece. Entretanto, atualmente suporta diversos microcontroladores e placas controladoras. Estas devem traduzir os códigos G-code em sequências de pulsos elétricos enviados aos motores de passo ou servomotores. Esses pulsos direcionam os movimentos da máquina, controlando a velocidade, a direção e a posição dos eixos X, Y e Z, e possivelmente eixos adicionais em máquinas de maior complexidade.

Sua operação em tempo real é um dos pontos fortes do GRBL, permitindo uma execução precisa das instruções de movimento enquanto mantém um fluxo contínuo de operação. A comunicação serial é frequentemente utilizada para enviar os comandos G-code ao *firmware* GRBL, permitindo uma integração eficiente com interfaces de usuário, como computadores e interfaces capazes de realizar o envio do G-Code por comunicação serial.

2.1.4 G-Code

O G-code é uma linguagem de programação amplamente utilizada na indústria de fabricação e usinagem para controlar máquinas CNC. Ele desempenha um papel crucial ao definir as instruções e comandos que orientam os movimentos e operações executadas pela máquina. O G-code é composto por uma série de códigos alfanuméricos que representam movimentos, velocidades, posicionamentos e outras ações necessárias para a produção de peças com precisão e eficiência (CHAMPLAIN; WOODCOCK, 2015).

2.1.4.1 Estrutura e Funcionamento

Cada instrução G-code é composta por uma letra seguida de um número, formando um comando específico. Por exemplo, o comando G01 indica um movimento linear a uma velocidade constante, enquanto o G02 e o G03 são usados para movimentos circulares em sentido horário e anti-horário, respectivamente. Além disso, há códigos adicionais que podem modificar parâmetros como velocidade de avanço, posição inicial, profundidade de corte e compensação de ferramentas (CHAMPLAIN; WOODCOCK, 2015). O Código 2.1 representa um exemplo de uso de G-code.

Código 2.1 – Exemplo de G-Code para um círculo de 50mm de diametro

```
1 G21 ; Definir unidades para milímetros
2 G90 ; Definir posicionamento absoluto
3 G1 X0 Y0 F1000 ; Movimento rápido para o ponto inicial
4
5 ; Iniciar o movimento circular
6 G2 X25 Y0 I25 J0 F1000 ; G2 é para movimento circular no sentido horá
  rio
7 G2 X0 Y0 I0 J25 F1000 ; Continuar o círculo
8 G2 X-25 Y0 I0 J-25 F1000 ; Continuar o círculo
9 G2 X0 Y0 I-25 J0 F1000 ; Continuar o círculo
10
11 ; Retornar ao ponto inicial
12 G1 X0 Y0 F1000 ; Movimento rápido de volta ao ponto inicial
13
14 G90 ; Voltar ao posicionamento absoluto
```

Fonte: Elaboração própria.

As instruções G-code são empregadas em uma variedade de processos de usinagem e fabricação, incluindo fresagem, torneamento, corte a laser, impressão 3D e muito mais. Sua versatilidade reside na capacidade de controlar uma ampla gama de movimentos e operações, permitindo a criação de peças com geometrias complexas e detalhadas.

Além do G-Code padrão, muitas máquinas CNC utilizam extensões ou variantes específicas de fabricantes ou modelos. Essas extensões podem adicionar funcionalidades adicionais, como controles específicos da máquina, ciclos automáticos e ajustes de compensação. Portanto, é importante estar ciente das nuances do G-Code específico para a máquina que está sendo usada.

2.1.4.2 Geração e Edição

A criação manual do G-code pode ser uma tarefa complexa e detalhada, exigindo um entendimento profundo das capacidades da máquina e das operações a

serem realizadas. No entanto, muitos softwares de CAD/CAM são capazes de gerar automaticamente o G-Code a partir de modelos 3D e configurações definidas pelo usuário. Essa abordagem reduz significativamente a possibilidade de erros humanos e agiliza o processo de programação. Diversos softwares desempenham um papel crucial na geração de código G, adaptando-se a uma variedade de contextos industriais e de fabricação, abrangendo máquinas e configurações diversas. Para a impressão 3D, o Cura (ULTIMAKER, 2023) se destaca como uma escolha gratuita, oferecendo recursos avançados de fatiamento e personalização. Outra opção nesse domínio é o Simplify3D (SIMPLIFY3D, 2023), embora seja um *software* proprietário. Na área de fabricação de placas de circuito impresso (PCI), o FlatCAM (CARAM, 2023) é uma ferramenta especializada, permitindo a importação de arquivos Gerber para gerar layouts completos de PCB. Esses softwares atendem a demandas específicas, proporcionando opções versáteis para uma variedade de aplicações industriais.

2.2 Protocolo de Comunicação Serial

A comunicação serial é uma forma de transmitir dados entre dispositivos eletrônicos por meio de um canal de comunicação unidirecional ou bidirecional. Essa abordagem utiliza uma sequência de bits para transmitir informações, sendo amplamente aplicada em uma variedade de contextos, desde comunicação entre computadores até controle de dispositivos periféricos, como microcontroladores e sensores (DAKOL, 2023). Em termos de funcionamento na comunicação serial, os bits são transmitidos sequencialmente, um após o outro, em uma linha de transmissão. Geralmente, há dois sinais dedicados para a comunicação: um para transmitir os dados (TX) e outro para receber os dados (RX). Essa transmissão ocorre em alta velocidade, podendo ser síncrona com a sincronização sendo mantida por meio de um *clock* compartilhado entre os dispositivos. No contexto de comunicação serial, é possível manter uma abordagem assíncrona, onde o *clock* é implicitamente definido. Nesse cenário, a sincronização entre os dispositivos envolvidos é alcançada garantindo que os *baud rates* (taxas de transmissão) sejam idênticos. Além disso, são utilizados bits de início e de parada para delimitar o tamanho dos pacotes de dados transmitidos. Essa configuração assíncrona oferece flexibilidade, embora exija uma concordância rigorosa quanto à taxa de transmissão para garantir uma comunicação eficaz.

Existem diversos padrões e protocolos para a comunicação serial, o principal para esse trabalho é o tipo UART (*Universal Asynchronous Receiver-Transmitter*). Trata-se de um protocolo de comunicação de hardware que emprega comunicação serial assíncrona, caracterizada pela ausência de um sinal de clock para sincronizar os bits transmitidos pelo dispositivo transmissor com o dispositivo receptor. Além disso, destaca-se que a velocidade de transmissão é configurável, conferindo flexibilidade ao

processo de comunicação (PEÑA; LEGASPI, 2020).

A comunicação serial é amplamente empregada em uma variedade de aplicações, incluindo:

- Microcontroladores: utilizam comunicação serial para se comunicar com outros dispositivos, como sensores, módulos de comunicação sem fio e *displays*.
- Eletrônica industrial: é usada em automação industrial para controlar máquinas, sensores, atuadores, rede *Modbus*, *Fieldbus*, CAN(*Controller Area Network*) *Open*, *DeviceNet*.
- Comunicação sem fio: Protocolos como *Bluetooth* e *Zigbee* usam a comunicação serial para transmitir dados entre dispositivos.
- Computadores: barramentos PCI Express (AJANOVIC, 2009) e SATA (GARCIA, 2009) para conexão de dispositivos de armazenamento.

A comunicação serial é uma tecnologia fundamental que permite a troca eficiente de dados entre dispositivos eletrônicos. As vantagens em relação a protocolos paralelos antes usados em máquinas CNC, os protocolos seriais têm uma linha de *clock* para sincronização, já os protocolos paralelos exigem uma sincronização precisa dos bits transmitidos entre os dispositivos. Isso pode resultar em desafios adicionais na implementação do *firmware* e *hardware*. A comunicação serial oferece uma maneira confiável de transmitir informações em uma ampla gama de aplicações, desempenhando um papel crucial na conectividade e automação da tecnologia moderna.

2.3 ESP32

O ESP32 (ESPRESSIF SYSTEMS, 2023c) é uma família de microcontroladores desenvolvidos pela empresa chinesa Espressif Systems. Ele destaca-se por suas capacidades de conectividade *Wi-Fi* e *Bluetooth*, além de oferecer um conjunto de recursos que o tornam adequado para uma ampla gama de aplicações, desde projetos de Internet das Coisas (IoT) até sistemas embarcados complexos.

Algumas das características que destacam o ESP32 são:

- Dual-core: pode possuir dois núcleos de processamento, o que permite a execução de tarefas paralelas, melhorando o desempenho e a eficiência do sistema.
- Conectividade *Wi-Fi* e *Bluetooth*: uma das maiores vantagens do ESP32 é a capacidade de se conectar a redes *Wi-Fi* e dispositivos *Bluetooth*, permitindo a comunicação sem fio em ambientes IoT.

- Baixo consumo de energia: é projetado para operar com eficiência energética, tornando-o adequado para aplicações alimentadas por bateria ou energia solar.
- Ampla variedade de interfaces: há uma variedade de interfaces de *hardware*, incluindo UART, I2C, SPI, GPIO e ADC, permitindo a conexão com uma ampla gama de sensores, dispositivos e periféricos.
- Segurança: há suporte a recursos de segurança, como criptografia e autenticação, garantindo a proteção dos dados em trânsito.

2.3.1 ESP-IDF

O ESP-IDF (*Espressif IoT Development Framework*) (ESPRESSIF SYSTEMS, 2023a) é o conjunto de ferramentas de desenvolvimento oficial fornecido pela Espressif Systems para programar e desenvolver aplicativos para o ESP32 e outros dispositivos Espressif. O ESP-IDF inclui bibliotecas, *drivers* e exemplos que simplificam o processo de criação de *software* para o microcontrolador.

Alguns componentes e características que integram o *framework* ESP-IDF.

- SDK (*Software Development Kit*): O SDK é a base do ESP-IDF e inclui as bibliotecas e ferramentas essenciais para o desenvolvimento de aplicativos.
- RTOS (Sistema Operacional de Tempo Real): O ESP-IDF inclui o FreeRTOS como sistema operacional de tempo real que facilita a programação de tarefas paralelas e auxilia a execução confiável de operações concorrentes.
- Ferramentas de compilação: O ESP-IDF vem com ferramentas de compilação e depuração que facilitam a criação e o teste de código para o ESP32.
- Exemplos e documentação: O ESP-IDF oferece uma variedade de exemplos e documentação detalhada que auxiliam os desenvolvedores a compreender os recursos e as capacidades do microcontrolador.
- Suporte a hardware: O ESP-IDF oferece suporte a muitos periféricos e dispositivos, simplificando a integração do *hardware* com o *software*.

O ESP32 em conjunto com o ESP-IDF oferece uma plataforma integrada para o desenvolvimento de projetos IoT e sistemas embarcados. Sua capacidade de conectividade, recursos de baixo consumo de energia e facilidade de desenvolvimento o tornam uma escolha popular para uma variedade de aplicações, desde sensores e sistemas de automação residencial e industrial.

2.4 FreeRTOS

O FreeRTOS é um sistema operacional de tempo real, de código aberto projetado para aplicações embarcadas que exigem operação e algumas funcionalidades em tempo real, ou seja, garantem a execução da tarefa dentro do período delimitado para a sua execução. Ele oferece um ambiente para o desenvolvimento de sistemas que precisam lidar com tarefas simultâneas e concorrentes, onde a pontualidade no cumprimento dos períodos de execução e a confiabilidade são fundamentais (FREERTOS, 2023). O FreeRTOS é amplamente utilizado em uma variedade de indústrias, incluindo automação industrial, eletrônicos de consumo, sistemas de controle e dispositivos IoT. Algumas características e conceitos do FreeRTOS.

- **Tarefas:** Permite criar várias tarefas independentes, cada uma com sua própria pilha e contexto. Isso permite que diferentes partes de um sistema sejam tratadas como tarefas separadas, simplificando o desenvolvimento e a manutenção do código.
- **Escalonamento prioritário:** O sistema de escalonamento do FreeRTOS é baseado em prioridades, permitindo que as tarefas com maior prioridade sejam executadas antes das tarefas com menor prioridade. Isso é crucial para garantir que tarefas críticas sejam concluídas em tempo hábil.
- **Tempo real:** É projetado para sistemas de tempo real, onde a precisão no cumprimento de prazos é essencial. Isso torna o FreeRTOS adequado para aplicações onde a pontualidade é crucial, como automação industrial e sistemas de controle.
- **Semáforos e filas:** Oferece mecanismos como semáforos e filas para sincronização e comunicação entre tarefas. Isso permite que as tarefas compartilhem recursos e informações de maneira segura e organizada.
- **Gestão de memória:** Inclui um mecanismo de gerenciamento de memória que permite a alocação e liberação eficiente de recursos de memória.
- **Portabilidade:** Projetado para ser portátil, o que significa que ele pode ser facilmente adaptado para funcionar em diferentes arquiteturas e plataformas.
- **Documentação abundante:** Possui documentação detalhada e exemplos que auxiliam os desenvolvedores a entenderem e utilizarem eficazmente o sistema.
- **Licença open source:** É distribuído sob uma licença de código aberto MIT, o que permite que desenvolvedores utilizem, modifiquem e distribuam o sistema de acordo com suas necessidades.

O FreeRTOS é utilizado em uma variedade de aplicações, tais como:

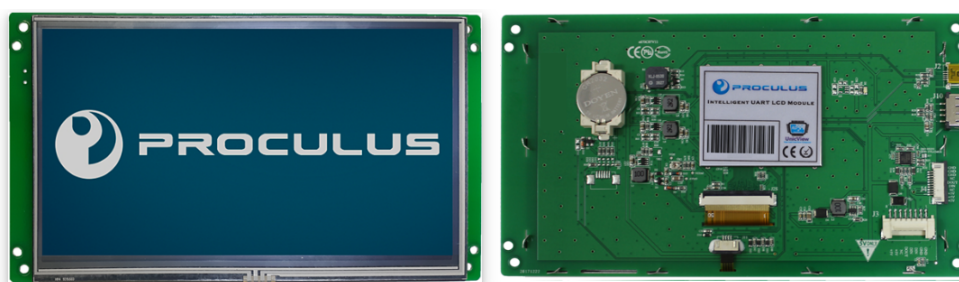
- Automóveis: Em sistemas de controle de veículos, desde sistemas de direção assistida até sistemas de entretenimento.
- Dispositivos IoT: Em sensores, atuadores e dispositivos conectados à Internet, onde a escalabilidade e o gerenciamento de tarefas são essenciais.
- Indústria: Para automação industrial, controle de máquinas e sistemas de monitoramento.
- Eletrônicos de Consumo: Em produtos como dispositivos de áudio, eletrodomésticos inteligentes e *wearables*.
- Telecomunicações: Para o gerenciamento de redes e sistemas de comunicação.
- Aeroespacial: Em sistemas embarcados para controle de aeronaves e satélites.

O FreeRTOS tem uma abordagem modular, recursos de escalonamento prioritário e ampla adoção na indústria o tornam uma escolha popular para projetos que exigem controle de tarefas concorrentes.

2.5 Display Inteligente

Displays inteligentes referem-se a telas ou painéis que incorporam tecnologia avançada para oferecer recursos além da simples exibição de informações (MELLO, 2023). Eles frequentemente incluem capacidades de interação, conectividade com outros dispositivos e processamento de dados. Esses *displays* podem ser encontrados em uma variedade de dispositivos até equipamentos industriais e sistemas de automação. Essa categoria de produtos conta com diversos fabricantes no mercado, sendo a Proculus um deles. A representação da Proculus no Brasil é feita pela empresa Viktor Vision, que disponibiliza uma ampla seleção de *displays* inteligentes diretamente no país. A Figura 3 demonstra um exemplar desta fabricante.

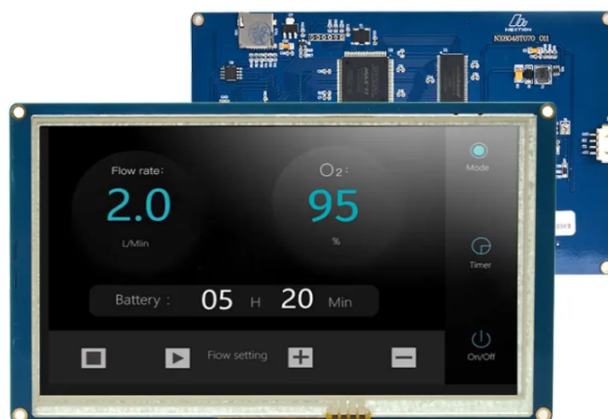
Figura 3 – Display Inteligente P80480V70C T07 Proculus



Fonte: (VIKTOR VISION, 2023b).

Outro fabricante significativo é a ITEAD, que oferece uma diversidade de produtos nesse segmento, embora não tenha distribuição oficial no Brasil. Um dos seus produtos é demonstrado na Figura 4.

Figura 4 – Display Inteligente NX8048T070 Nextion



Fonte: (ITEAD, 2023).

Alguns dos recursos comuns de *displays* inteligentes são:

- **Toque Interativo:** Muitos *displays* inteligentes são sensíveis ao toque, permitindo aos usuários interagir diretamente com a interface. Isso é amplamente usado em quiosques de autoatendimento e em sistemas de controle industrial.
- **Conectividade:** *Displays* inteligentes frequentemente incluem opções de conectividade, como *Wi-Fi*, *Bluetooth* ou cabos, permitindo que se comuniquem com outros dispositivos e acessem informações ou serviços online.
- **Processamento Interno:** Alguns *displays* inteligentes têm capacidade de processamento integrado, permitindo que executem aplicativos, exibam informações em tempo real e até realizem tarefas complexas sem a necessidade de um dispositivo externo.
- **Personalização e Configuração:** A maioria dos *displays* inteligentes permite personalização da interface e configuração de acordo com as preferências do usuário ou os requisitos da aplicação.
- **Aplicações Diversas:** *Displays* inteligentes são usados em várias indústrias, incluindo automação industrial, saúde, varejo, entretenimento, educação e muito mais.

Os *displays* inteligentes podem ser usados em diversas áreas que necessitam de uma interface que facilite a interação homem-máquina, algumas aplicações em potencial são:

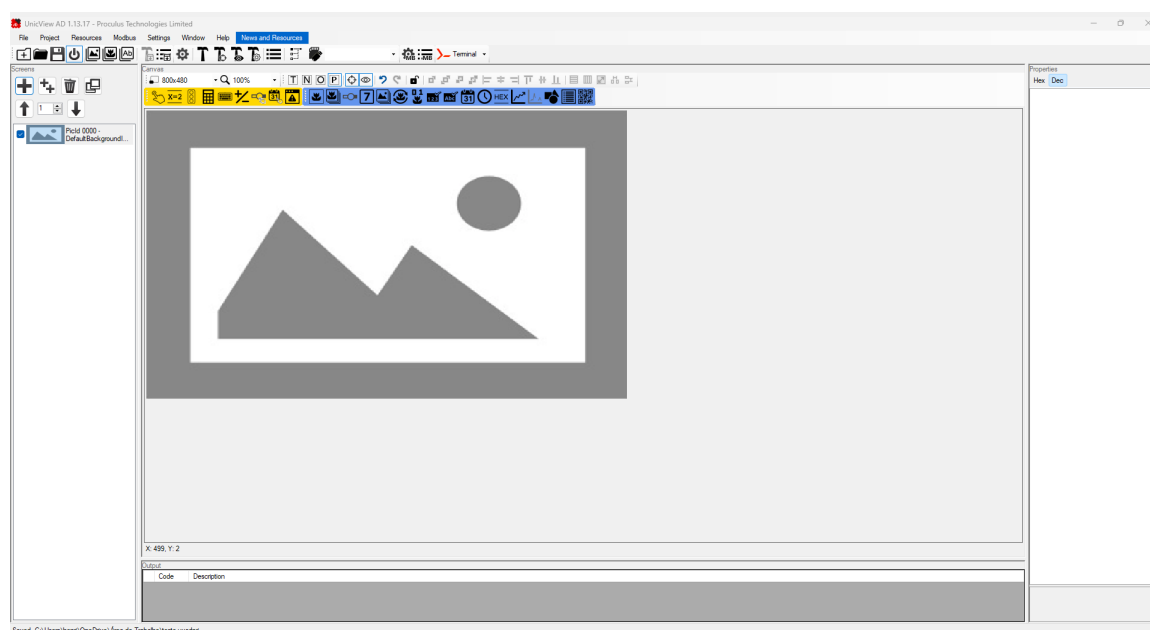
- Automação Industrial: podem ser usados para controlar e monitorar processos industriais, exibindo informações de produção, status de máquinas e permitindo ajustes em tempo real.
- Sistemas de Entretenimento: estão presentes em TVs, monitores e dispositivos de *streaming*, proporcionando interatividade, acesso a aplicativos e exibição de conteúdo.
- Setor de Saúde: podem ser usados em equipamentos médicos para exibir informações de pacientes, resultados de exames e até mesmo permitir interações com profissionais de saúde.
- Setor Automotivo: em veículos oferecem informações de navegação, entretenimento, diagnóstico do veículo e muito mais.
- Quiosques de Informação: são usados em locais públicos para fornecer informações interativas aos usuários, como mapas, horários de transporte público e guias turísticos.
- Educação: podem ser usados em ambientes educacionais para criar experiências de aprendizado interativas e envolventes.

Percebe-se que os *displays* inteligentes fornecem interatividade, conectividade e funcionalidades avançadas para uma variedade de aplicações em várias indústrias.

2.5.1 Software UnicView AD e conexão com o *hardware*

O *display* Proculus inclui um *software* que gerencia e programa todo o aspecto gráfico e funcional do *display*. A Figura 5 ilustra o *software* UnicView AD na sua versão 1.13.17 (VIKTOR VISION, 2023a) sem abrir nenhum projeto.

Figura 5 – Layout UnicView AD



Fonte: Elaboração própria.

Ele contém vários elementos e funcionalidades, cada um com características de funcionamento específicas, associadas a um endereço denominado Pontos de Valor (VPs) (VIKTOR VISION, 2017a), que representa a posição inicial do valor do elemento na memória RAM do *display*. A Figura 6 representa a barra com os elementos funcionais em amarelo e os elementos gráficos em azul.

Figura 6 – Barra de elementos UnicView AD



Fonte: Elaboração própria.

A barra de elementos (VIKTOR VISION, 2017b) proporciona a capacidade de adicionar componentes ao projeto do *display*. Esses elementos dividem-se em dois grupos distintos: os elementos funcionais, que se destinam à interação com a tela sensível ao toque, e os elementos gráficos, cuja função principal é ler os valores de seus VPs e exibí-los no visor do *display*.

2.6 Tecnologias do Projeto

A partir do referencial teórico elaborado, foi construída uma tabela que resume as tecnologias selecionadas para a execução do projeto. O Quadro 2 apresenta de forma detalhada todas as tecnologias escolhidas, acompanhadas das justificativas que fundamentam a decisão por cada uma delas.

Quadro 2 – Tecnologias selecionadas para o Projeto

Tecnologia	Justificativa
ESP32-S3	Selecionado por ter interface USB nativa sendo um diferencial entre os outros chips da família ESP.
Kit de Desenvolvimento ESP32 S3 Dev Kit C 1 v1.1	Facilita o desenvolvimento por possuir alguns circuitos úteis já implementados.
ESP-IDF versão 5.0.2	Possui uma vasta gama de bibliotecas e implementações de interfaces validadas.
Espressif IDE versão 2.9.1	Baseado no <i>firmware</i> eclipse, tendo boa interface e encurtando a curva de aprendizagem do uso do framework ESP-IDF.
Display P80480V70C_T07+ (Proculus)	Possui representação no Brasil tendo acesso a suporte, garantia e uma biblioteca de funções em C. Além de possuir um <i>software</i> voltado para o desenvolvimento da parte gráfica.
GRBL versão 1.1	Software Open Source responsável por receber instruções G-code traduzindo-as para movimentação dos motores.
Arduino UNO	Placa de desenvolvimento que utiliza ATMEGA 328P responsável por executar o <i>firmware</i> GRBL.

Fonte: Elaboração própria.

Esses conceitos e tecnologias formam a base teórica essencial para o desenvolvimento da interface G-Code *Sender* para máquinas CNC, conforme descrito no projeto do TCC. Cada um desses elementos desempenha um papel fundamental na criação de um sistema eficiente, autônomo e de fácil utilização para controlar máquinas CNC, melhorando a versatilidade e a acessibilidade da tecnologia CNC na indústria de fabricação automatizada.

3 DESENVOLVIMENTO

Neste capítulo, aborda-se o desenvolvimento do projeto, discutindo sua estrutura de pesquisa e os métodos empregados. Cada módulo que compõe o projeto será detalhadamente explorado, incluindo suas especificações de desenvolvimento e funcionamento. Este capítulo oferece uma visão abrangente do processo de desenvolvimento realizado para alcançar os objetivos do projeto.

3.1 Metodologia

A condução deste trabalho seguiu uma metodologia baseada em princípios de pesquisa aplicada, exploratória, descritiva e bibliográfica.

A pesquisa aplicada teve como objetivo primordial a resolução de problemas práticos e a aplicação de conhecimento teórico para a solução de desafios reais. A pesquisa exploratória, conforme delineada por (GIL, 2002), tem por finalidade aprofundar a compreensão de um problema, tornando-o mais claro e formulando hipóteses. Essa etapa, caracterizada por sua flexibilidade, permitiu considerar múltiplos aspectos relacionados ao tema em estudo. Frequentemente, envolveu revisões bibliográficas, conversas com especialistas que possuíam experiência prática no problema e análise de exemplos ilustrativos que auxiliaram na compreensão do tema. Por outro lado, a pesquisa descritiva tem como foco principal a análise e descrição das características de um determinado fenômeno, bem como o estabelecimento das relações entre variáveis relevantes.

3.1.1 Métodos Aplicados

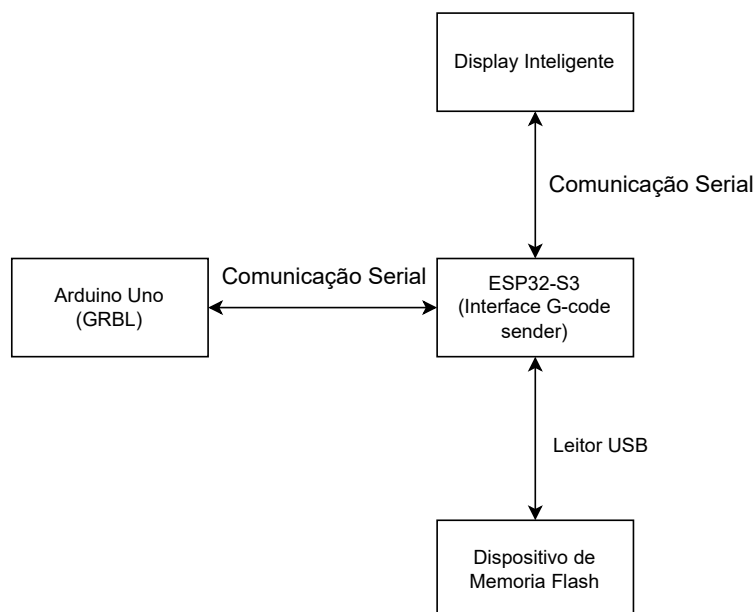
O desenvolvimento deste projeto seguiu uma série de etapas que incluíram o planejamento, pesquisa de requisitos, seleção de tecnologias e a implementação.

Este trabalho tem como foco a implementação uma interface para o *firmware* GRBL que é a unidade de controle da CNC. Para realizar o desenvolvimento da interface G-Code *Sender*, o primeiro passo foi delinear o escopo do trabalho, identificando as funcionalidades essenciais que a interface G-Code *Sender* deve abranger para garantir um bom funcionamento da unidade de controle. Com base nesse escopo, foi realizada uma pesquisa detalhada referente aos requisitos e as tecnologias disponíveis para atendê-los. Essa fase forneceu o referencial teórico, com foco nas tecnologias que compõem o projeto.

A escolha das tecnologias a serem utilizadas foi realizada considerando, principalmente, a sinergia com o GRBL e a arquitetura global do projeto. Uma das possibilidades para executar o *firmware* GRBL é o Arduino Uno que utiliza o microcontrolador Atmega328. O Arduino é muito acessível comercialmente e por isso foi utilizado para

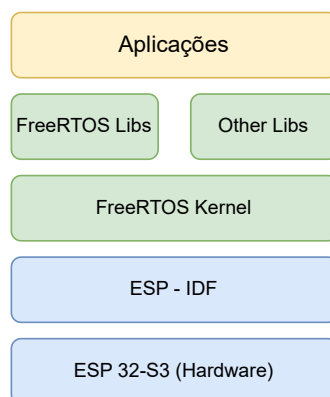
ser a unidade controladora recebendo os comandos através de comunicação serial. A Figura 7 apresenta o diagrama de blocos do sistema que será implementado, no qual a placa Arduino UNO incorpora o GRBL. A aplicação desenvolvida em C para a interface G-code *sender* será integrada a um microcontrolador da família ESP32, estabelecendo comunicação tanto com um *display* inteligente quanto realizando a leitura de um dispositivo de memória flash.

Figura 7 – Diagrama de blocos do sistema



Fonte: Elaboração própria.

Como citado anteriormente, utilizou-se a família de microcontroladores ESP32 para embarcar a aplicação desenvolvida, visto que já possui implementado todas as interfaces e periféricos desejados. Além disso, alguns modelos desta família possuem a interface USB já implementada, fundamental para a leitura de arquivos G-Code. Dentre essas opções, foi escolhida o módulo ESP32-S3-DevKitC-1 (ESPRESSIF SYSTEMS, 2023b), uma vez que oferece duas unidades de processamento, conectividade *Wi-Fi* e *Bluetooth Low Energy* (BLE), uma variedade de exemplos da própria fabricante e uma comunidade de usuários ativa. Quanto ao *software*, o ambiente de desenvolvimento foi utilizado a Espressif-IDE (versão 2.9.1), que é o ambiente integrado de desenvolvimento para o ESP-IDF na versão 5.0.2. Foi utilizado o FreeRTOS como sistema operacional para viabilizar a utilização de múltiplas tarefas, ou seja, permitir que diferentes lógica do programa cooperem de forma paralela, buscando simplificar a estrutura lógica do programa em contraste com uma aplicação de tarefa única, na qual todas as operações ocorrem de maneira sequencial. As camadas de funcionamento e implementação do projeto são demonstradas na Figura 8 com toda a programação da aplicação desenvolvida em linguagem C.

Figura 8 – Camadas ESP32

Fonte: Elaboração própria.

A parte de interface homem-máquina (IHM) foi implementada por meio do *display* inteligente P80480V70C_T07+ da empresa *Proculus*, com uma comunicação serial assíncrona com o microcontrolador ESP32-S3. Essa escolha simplificou a representação gráfica e proporcionou alta qualidade de imagem e funcionalidade. O uso dessa tecnologia facilitou a seleção e integração desta solução. O *display* inteligente utiliza o *software* UnicView AD que permitiu o desenvolvimento de um projeto gráfico personalizado, incorporando elementos gráficos, feitos no *software* Corel Draw (COREL CORPORATION, 2023), e de toque (*touchscreen*) que simplificam a interação do usuário. Isso também aliviou a carga de processamento do microcontrolador central, pois o *display* possui um sistema embarcado próprio para essa finalidade.

Posteriormente, realizaram-se testes abrangentes e específicos para garantir o funcionamento adequado de cada componente do projeto, incluindo os protocolos de comunicação serial, a interface USB, o acionamento de pinos de entrada/saída (GPIOs) e o uso do sistema operacional FreeRTOS para a aplicação. A implementação foi realizada seguindo exemplos fornecidos pela Espressif System, que foram adaptados para se adequar às necessidades do projeto.

Para minimizar interferências e problemas de conexão, uma Placa de Circuito Impresso (PCI) protótipo foi desenvolvida no *software* Proteus, com interfaces prontas para atender ao projeto. Com o protótipo físico montado, a implementação do projeto foi dividida em quatro partes principais:

- Integração do *display* inteligente: esta fase envolveu a configuração dos parâmetros de comunicação serial e o recebimento de informações através de um *buffer* circular no recebimento da ESP32-S3.
- Integração do GRBL: foi desenvolvida uma biblioteca que implementa o protocolo de comunicação do GRBL.

- Desenvolvimento da interface gráfica do *display*: foco na criação de uma interface que permite o controle das funções do GRBL e na integração com a parte de comunicação.
- Implementação da leitura de pendrives: a capacidade de ler e executar arquivos G-code diretamente de dispositivos de armazenamento USB.

Com a implementação concluída, foram realizados testes práticos para validar o funcionamento, sendo feitas comparações dos resultados entre o *software* bCNC (BCNC, 2022) e a interface feita. A validação final resultou na conclusão bem-sucedida do primeiro protótipo.

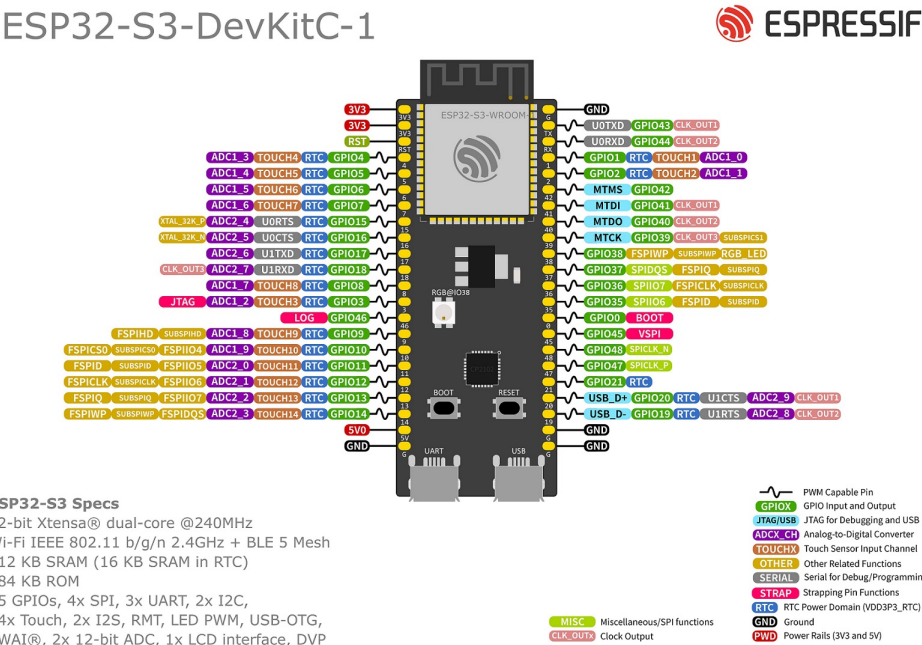
3.2 Projeto da Placa de Circuito Impresso (PCI)

O objetivo da placa de circuito impresso (PCI) é integrar eletricamente todas as tecnologias envolvidas no projeto, proporcionando menor interferência de possíveis problemas de conexões ou ruídos externos. Além disso, esta etapa visa criar o primeiro protótipo para validação. O projeto da PCI foi concebido em torno da placa de desenvolvimento ESP32-S3-DevKitC-1. Este módulo é acoplado na PCI em desenvolvimento, contribuindo para maior praticidade e agilidade no desenvolvimento. A PCI abrange as seguintes conexões: ligação com o *display* serial, alimentação por meio de um conector, conexão serial para o GRBL, um conector USB, 10 entradas opto-acopladas, acessos por conectores às demais *input/output*-I/Os e uma entrada direta de alimentação para o *display*, suportando até 15 V para entregar em sua saída para o circuito do microcontrolador uma tensão de 5V.

3.2.1 Kit de Desenvolvimento ESP32 S3 Dev Kit C 1 v1.1

Este kit de desenvolvimento utiliza o módulo ESP32-S3-WROOM-1 da Espressif e disponibiliza a maioria dos pinos de entrada e saída em dois conectores. Possui duas entradas USB, uma que dá acesso ao periférico USB integrado do chip e a uma funcionalidade integrada de depuração JTAG, e outra que possibilita a conexão serial para a gravação de códigos projetados e relatórios de depuração segundo Espressif Systems (2023b).

Figura 9 – Pinos e configurações da ESP 32-S3



Fonte: (ESPRESSIF SYSTEMS, 2023b).

As ligações entre o ESP32 S3 Dev Kit C 1 v1.1 e a placa de interface são detalhadas no Quadro 3, abrangendo todas as conexões relevantes.

Quadro 3 – Ligações do ESP32 S3 Dev Kit C 1 v1.1 na PCI

Pinos	Sinal
GPIO1	TX1(GRBL)
GPIO2	RX1(GRBL)
GPIO de 3-11 e GPIO 48	Entradas Optoacopladas
GPIO 17	TX(Display)
GPIO 18	RX(Display)
GPIO 19	USB D+
GPIO 20	USB D-
5 Vin	Ligada na saída do regulador de tensão da PCI
GND	Referência da placa da interface

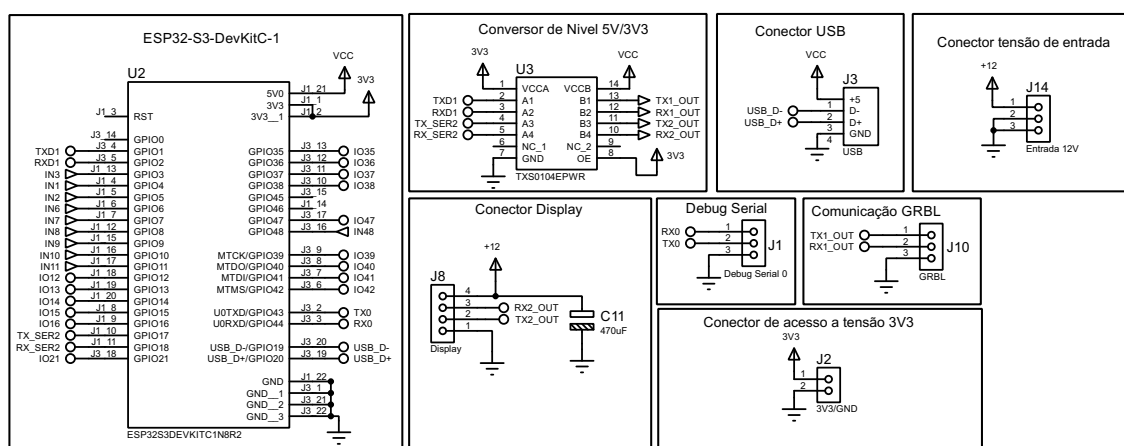
Fonte: Elaboração própria.

É importante ressaltar que os pinos GPIO0, GPIO45 e GPIO46 não devem ser utilizados pois estão relacionados à memória e a sinalizações de modos de *boot*. As demais conexões ficam ligadas diretamente a conectores para futura utilização, embora, até o momento, não tenham um uso específico. Com as conexões definidas, pode-se proceder ao design da PCI.

3.2.2 Esquemático, Layout e Modelo Final

A PCI foi elaborada utilizando o *software* Proteus. O esquemático foi dividido em duas partes. A primeira parte compreende oito blocos, que englobam a origem dos sinais e a parte de comunicação. A Figura 10 demonstra os 8 blocos como um todo. O bloco mais crucial é o que se refere a placa de desenvolvimento ESP32-S3-DevKitC-1, uma vez que concentra todas as conexões ao *chip* de controle e a parte de alimentação de 5V. Outro bloco de grande importância é o conversor de nível, que é vital para adequar as tensões de comunicação, visto que o *display* inteligente e a placa Arduino UNO operam em 5V, enquanto a ESP32S3 funciona em 3,3V. Isso é feito para garantir um funcionamento adequado, dado que os limites das entradas/saídas (IOs) da ESP32S3 são de 3,3V. Os seis blocos restantes representam a ligação dos sinais aos conectores que proporcionam o acesso aos sinais desejados, com destaque para o capacitor C11 no bloco do conector do *display*, que está ligado à alimentação para evitar oscilações de tensão que possam prejudicar o funcionamento.

Figura 10 – Parte 1 do esquemático final

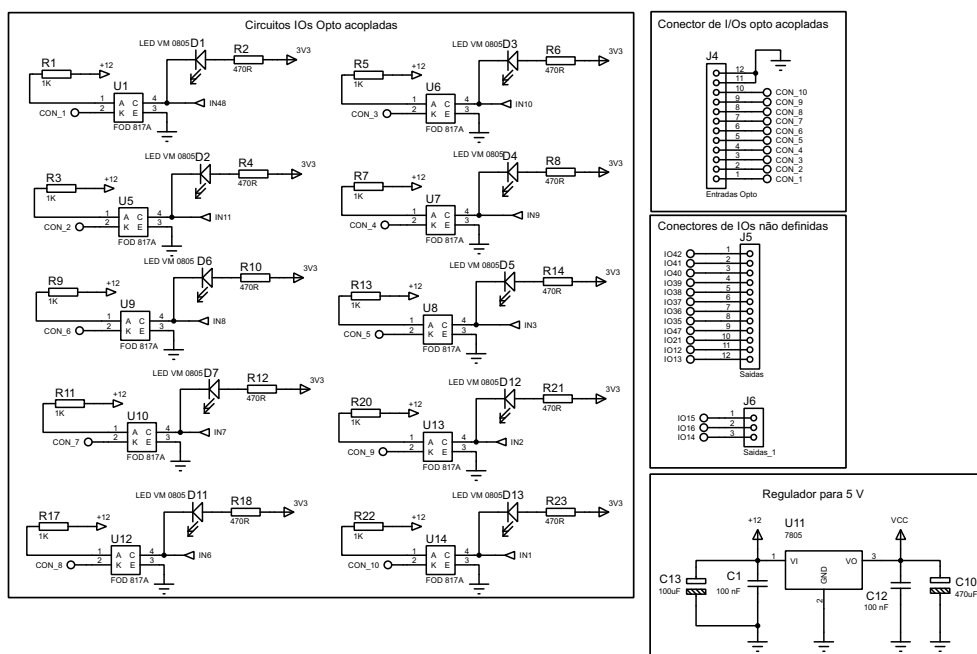


Fonte: Elaboração própria.

A segunda parte do esquemático abrange quatro blocos ilustrados pela Figura 11 demonstra-os. O bloco do regulador de 5V alimenta exclusivamente o microcontrolador. Foi selecionado o regulador linear LM7805 (STMICROELECTRONICS, 2018), dada a facilidade de uso, disponibilidade, baixa corrente consumida pelo microcontrolador e baixa queda de tensão gerada sobre o mesmo sendo a máxima esperada de 12V. O regulador linear funciona basicamente regulando a sua tensão e corrente de saída dissipando a energia em formato de calor. Sendo a quantidade dessa a diferença da energia de entrada com a de saída, a perda no regulador esperada é pequena em relação as perdas e tensões do sistema de acionamento de motores, assim não tendo problemas com perdas de calor. Os capacitores, tanto na entrada

quanto na saída deste bloco, atuam como filtros de baixa e alta frequência. O bloco “Circuitos IOs Optoacopladas” foi incorporado prevendo acionamentos externos de sensores que operem com tensões de 24V sendo esta uma tensão comum na indústria, utilizando entradas com optoacopladores FOD817 (ONSEMI, 2019). Esses circuitos com optoacopladores operam de modo que, ao receberem um sinal de nível baixo, fecham o circuito de entrada acendendo o LED interno que ativa o fototransistor na saída. Isso também gera um sinal baixo na entrada do microcontrolador. Os dois blocos restantes são conectores que dão acesso às IOs disponíveis, com destaque para os pontos de GND para serem utilizados para acionar as entradas optoacopladas, como chaves de fim de curso, por exemplo.

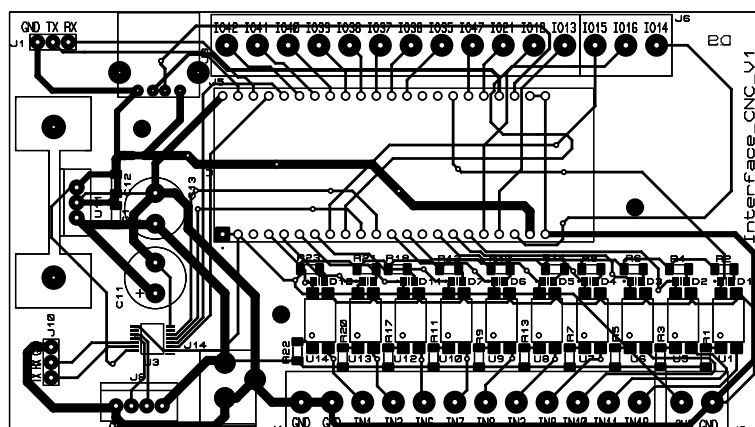
Figura 11 – Parte 2 do esquemático final



Fonte: Elaboração própria.

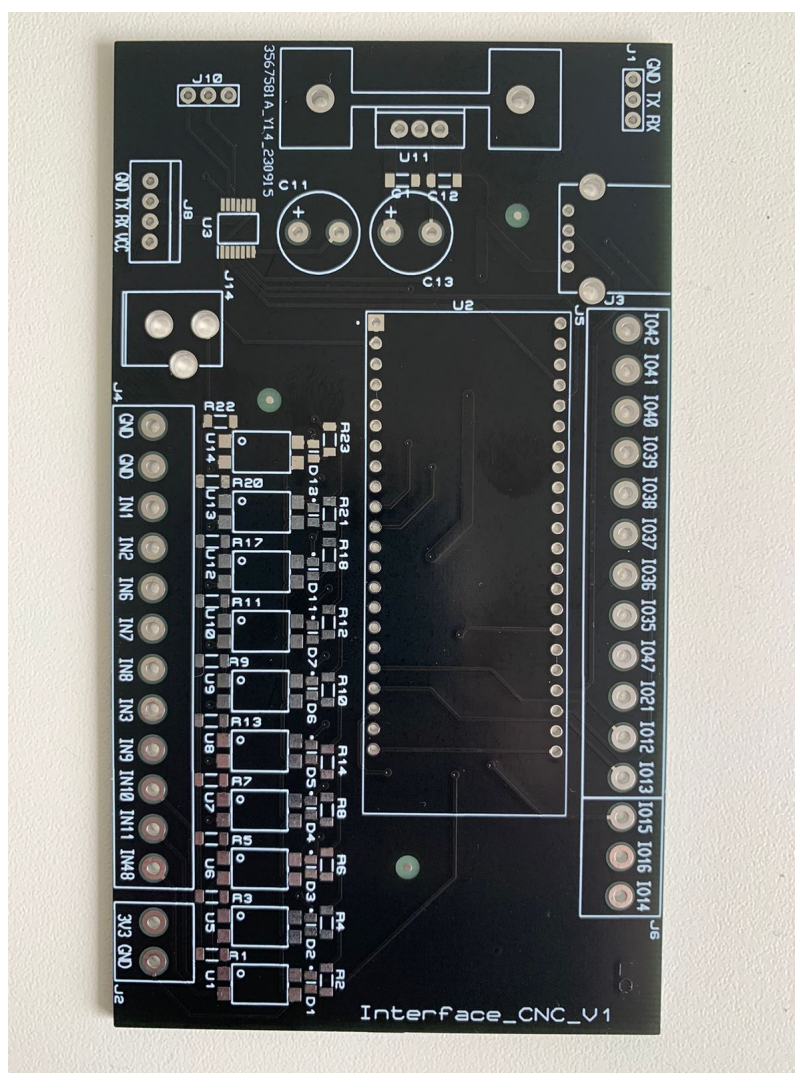
Com o esquemático definido, foi possível avançar para o projeto do *layout* da PCI. O *layout* teve tem resultado final demonstrado na Figura 12.

Figura 12 – Layout PCI



Fonte: Elaboração própria.

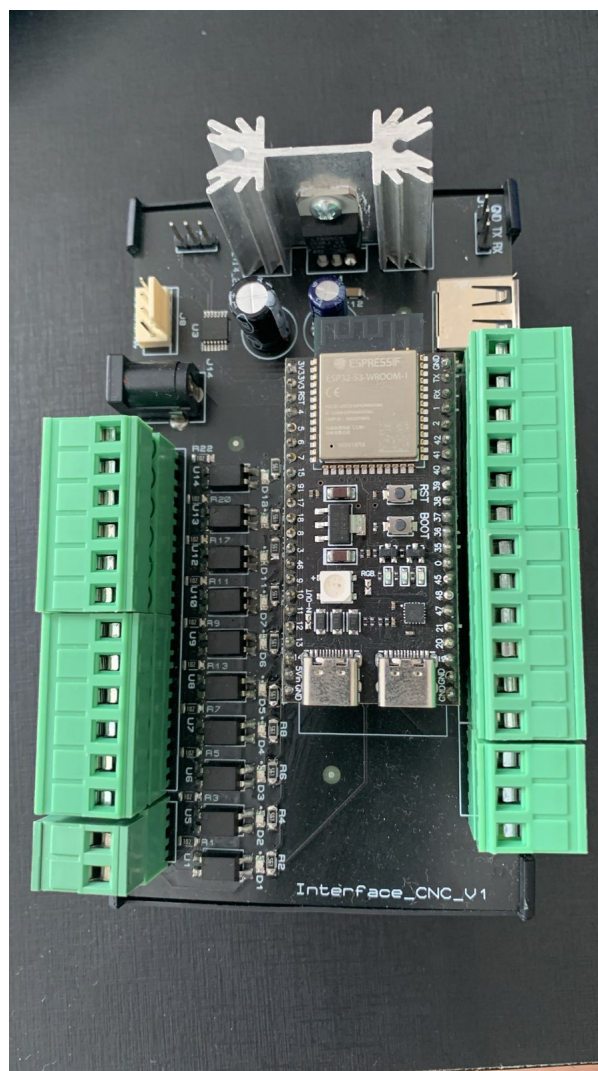
Um dos objetivos é permitir que esta placa seja acoplada a um painel industrial por meio de trilhos DIN, comumente utilizados na indústria. Portanto, a largura da placa foi limitada a 70 mm para permitir o encaixe adequado aos conectores do trilho. Após a conclusão do design, a PCI foi fabricada pela empresa chinesa JLC PCB, o resultado é demonstrado na Figura 13.

Figura 13 – Placa pronta não montada

Fonte: Elaboração própria.

Ao receber-se o lote piloto, procedeu-se à montagem e realização de testes das conexões validando todos os blocos. Como a placa de circuito impresso foi confeccionada por terceiros, foi feita uma bateria de testes elétricos para garantir o seu bom funcionamento e assegurar que nenhum erro de fabricação prejudique o desenvolvimento. Foi testado primeiramente a parte de alimentação conectando uma fonte de tensão de 15V e verificando a tensão de saída é entregue de acordo com os 5V esperados, assim sendo possível evitar problemas com sobretensão. Posteriormente cada circuito de acionamento de maneira individual foi testado colocando uma chave fim de curso que ativasse os circuitos opto-acoplados. Com a aprovação nos testes, o primeiro protótipo foi concluído. A PCI pronta, montada e equipada com os conectores para o trilho DIN é exibida na Figura 14.

Figura 14 – Placa pronta montada

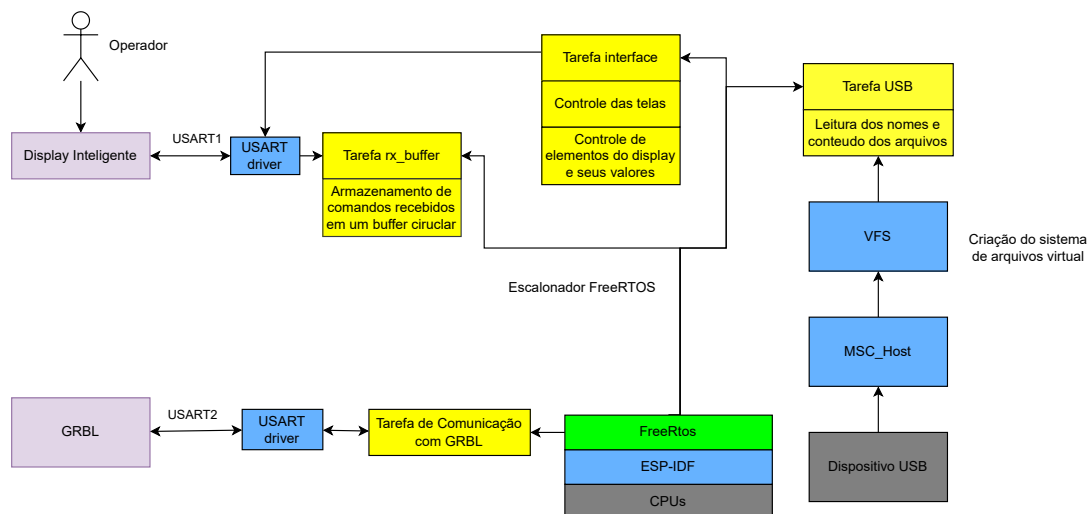


Fonte: Elaboração própria.

3.3 Funcionamento do sistema e suas tarefas

Como citado anteriormente, o projeto foi implementado utilizando o sistema operacional FreeRTOS em conjunto com o *framework* ESP-IDF, responsável pelos *drivers* e aspectos de nível mais baixo como periféricos do projeto. O sistema desenvolvido é representado de maneira geral na Figura 15, evidenciando dois componentes externos em lilás sendo estes o GRBL e o *display* inteligente. Os componentes em cinza são hardwares, no caso as CPUs advém do chip ESP32-S3, responsável por embarcar toda a aplicação desenvolvida. Em azul, destaca-se os itens que compõem o *framework* ESP-IDF onde no caso do sistema se tem o *driver* da comunicação serial, o sistema de arquivos virtual e o sistema de hospedagem de dispositivos de *Mass Storage Class* (MSC). E em amarelo estão as tarefas principais que compõem o sistema e abaixo destas estão suas principais responsabilidades e funcionalidades.

Figura 15 – Representação do Sistema Implementado



Fonte: Elaboração própria.

A aplicação é estruturada em três núcleos. O primeiro núcleo engloba a parte de interface homem-máquina, centrada no *display* inteligente, que recebe e envia informações para o microcontrolador através da porta Serial 1. Este núcleo compreende duas tarefas principais, fundamentais para grande parte de seu funcionamento. O segundo núcleo é dedicado à comunicação com o GRBL, realizada pela porta Serial 2, com uma tarefa principal encarregada de gerenciar a comunicação com o mesmo. O terceiro e último núcleo trata da conexão do dispositivo USB e da criação de seu sistema de arquivos virtual. Ele inclui também uma tarefa que acessa esse sistema de arquivos virtual e executa operações de leitura. A integração desses três núcleos permite a gestão eficaz de todas as funcionalidades necessárias para a interface G-Code *Sender*, resultando em um sistema funcional e que atende as necessidades estabelecidas.

O sistema do projeto é implementado com um sistema operacional através de tarefas. A criação das tarefas e a definição de suas propriedades desempenham um papel crucial no bom funcionamento do sistema como um todo. A definição das prioridades das tarefas foi realizada de maneira empírica, visando estabelecer uma fluidez eficaz do sistema do ponto de vista do usuário. A declaração das tarefas é apresentada no trecho de Código 3.1.

Código 3.1 – Declaração das tarefas do sistema

```

16
17 //Tarefa de recebimento de informações do display
18 xTaskCreate(&rx_task, "uart_rx_task", 1024 * 2, NULL,
19             configMAX_PRIORITIES, NULL);
20 //Tarefa de controle das telas do display e seus elementos

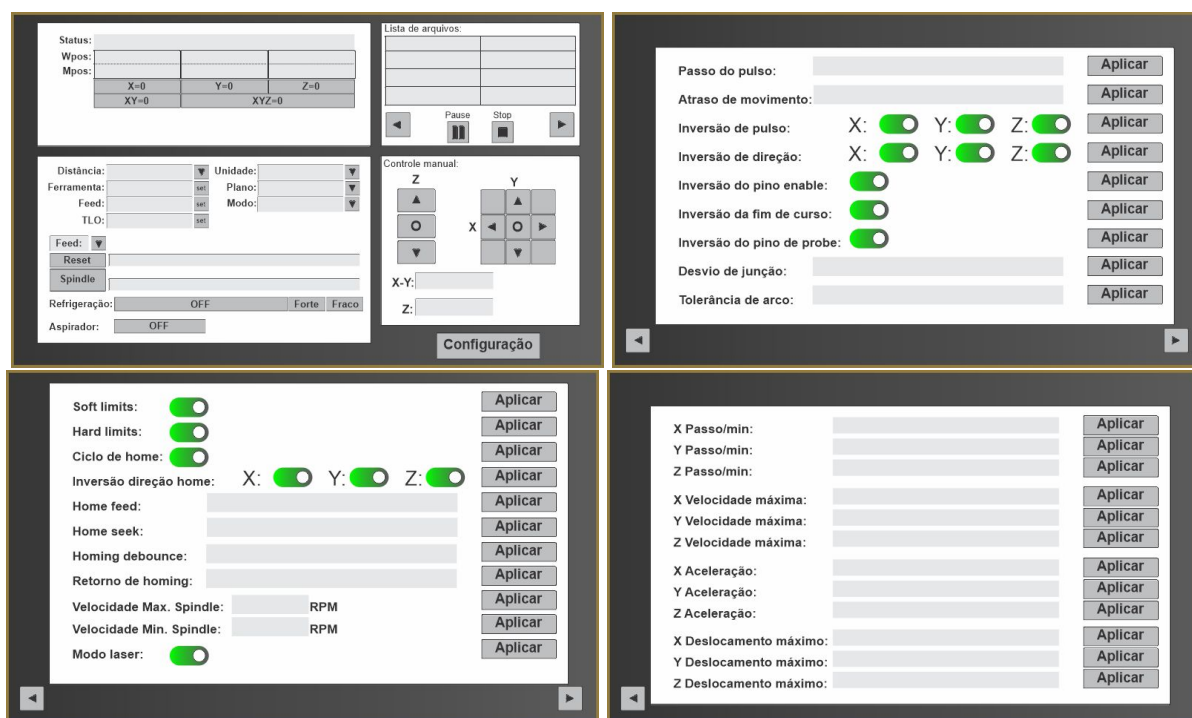
```

```
21 xTaskCreate(&interface_task, "interface_task", 1024 * 4, NULL,  
    configMAX_PRIORITIES - 1, NULL);  
22  
23 //Tarefa de Envio e Rebimento de comandos com o GRBL  
24 xTaskCreate(&grbl_communication_task, "grbl_com_task", 1024 * 4, NULL,  
    configMAX_PRIORITIES - 2, NULL);  
25  
26 //Tarefa de solicitação de atualização de estados do GRBL  
27 xTaskCreate(&att_states, "att_state", 1024, NULL, configMAX_PRIORITIES  
    - 2, NULL);  
28  
29 //Tarefa de controle de dispositivos USB  
30 xTaskCreate(&usb_task, "usb_task", 1024 * 4, NULL,  
    configMAX_PRIORITIES - 3, NULL);
```

Fonte: Elaboração própria.

O primeiro núcleo do sistema requer a maior prioridade de execução para oferecer uma melhor experiência ao usuário, envolvendo o controle da comunicação com o *display* e a gestão das telas. As funcionalidades e controles da aplicação foram divididas em quatro telas, sendo a primeira tela composta por uma variedade de controles e ações como movimentação individual do eixo, posicionamento da máquina, lista de arquivos caso um dispositivo USB esteja conectado, entre outras. As telas subsequentes destinam-se principalmente a ajustes de configuração do GRBL. A Figura 16 ilustra as quatro telas.

Figura 16 – Telas da IHM



Fonte: Elaboração própria.

Os usuários têm a liberdade de interagir com cada elemento nas telas e navegar entre elas. Todo o sistema de resposta às interações do usuário com o *display* é processado pelo microcontrolador. O usuário espera uma resposta ágil e fluida do *display* ao trocar de tela ou selecionar alguma função. Para isso, existem duas tarefas principais: a “rx_task” e a “interface_task”. A “rx_task” é encarregada de executar a função de recebimento e armazenamento dos comandos em um *buffer* circular. A tarefa “interface_task” controla uma máquina de estados, chamando funções correspondentes a cada tela implementada. Em outras palavras, o microcontrolador verifica se ocorreu alguma interação com o *display* em determinada tela e, caso tenha ocorrido, identifica o elemento e processa a informação recebida de acordo. Ao trocar de tela, o estado da máquina é alterado, assim como os elementos que serão verificados. A tela do *display* sempre corresponde ao estado atual da máquina de estados da tarefa. Esta tarefa conduz verificações da comunicação serial, possibilitando que outras tarefas a coloquem em modo de suspensão. Além disso, algumas das funções empregadas nesta tarefa envolvem o uso da função *delay*, proporcionando a viabilidade para que outras tarefas assumam o controle. O Código 3.2 a seguir demonstra a tarefa “interface_task”.

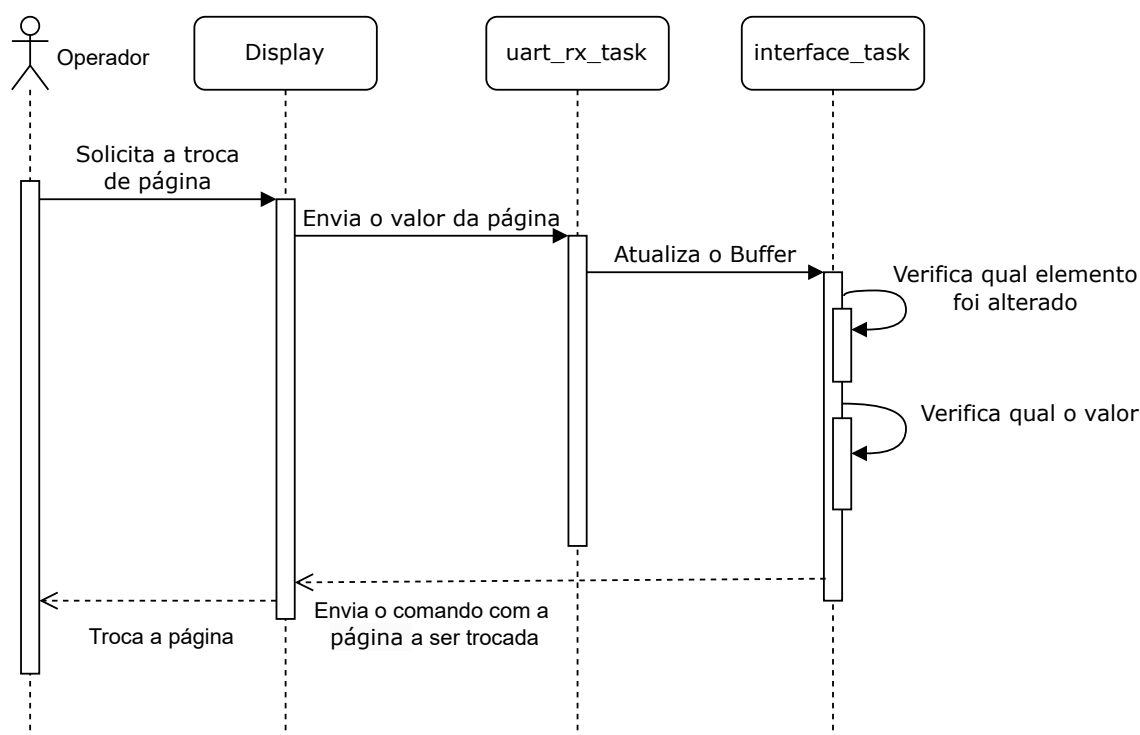
Código 3.2 – Tarefa de controle das páginas do display

```
32 void interface_task(void *arg) {
33     //Inicia a variavel de estado para a pagina inicial
```

```
34     pageid = 0;
35     //Envia para o display o valor da página inicial
36     lcm_change_pic_id(pageid);
37     //Inicia algumas varaveis do display
38     lcm_var_write(&feed_rate, 100000);
39     lcm_var_write(&jog_xy, 10000);
40     lcm_var_write(&jog_z, 10000);
41
42     //Loop infinito da máquina de estados
43     while (1) {
44         switch (pageid) {
45             case 0:
46                 tela_controle();
47                 break;
48             case 1:
49                 tela_config1();
50                 break;
51             case 2:
52                 tela_config2();
53                 break;
54             case 3:
55                 tela_config3();
56                 break;
57             default:
58                 //lcm_change_pic_id(erro_page);
59                 break;
60         }
61     }
62 }
```

Fonte: Elaboração própria.

Essa tarefa controla as páginas e os elementos do *display* garantindo que as funções de controle das páginas sejam implementadas e verificadas de maneira eficiente para evitar atrasos ou lentidão no *software*. As funções que são executadas em cada estado implementam o controle das suas respectivas páginas da Figura 16. Essas funções são estruturadas colocando todos os elementos presentes naquela página. De acordo com o elemento que o operador interage, o microcontrolador faz o tratamento da ação. Assim, uma função verifica se algum elemento foi ativado, caso tenha sido, verifica qual é e trata da maneira adequada. A Figura 17 demonstra a interação entre as tarefas para a troca de uma página.

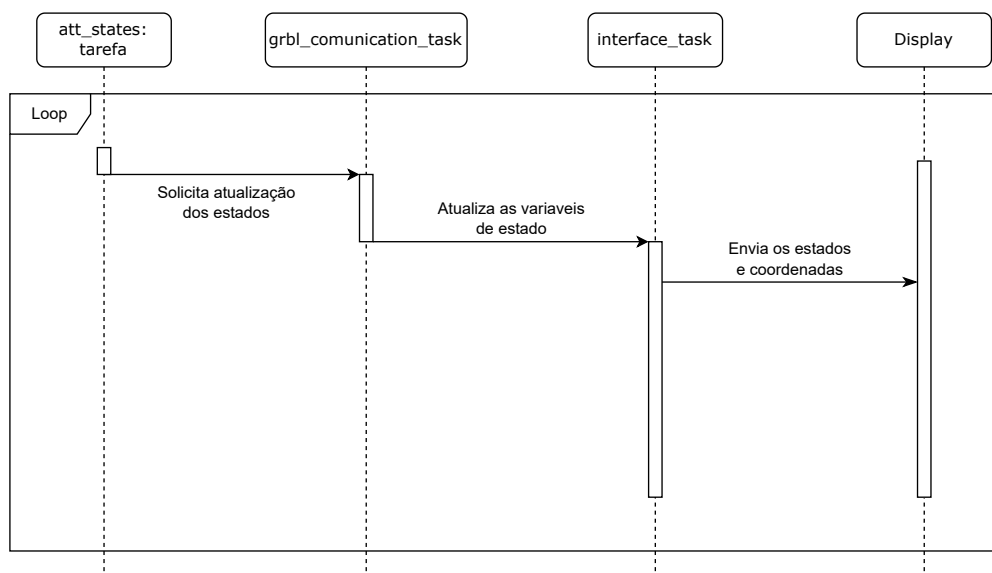
Figura 17 – Interação entre as tarefas para realizar a troca de uma página

Fonte: Elaboração própria.

O segundo núcleo tem uma prioridade de execução um pouco menor, este é focado na comunicação com o GRBL, onde tem uma tarefa que recebe os comandos que devem ser enviados para a placa Arduino Uno que embarca o GRBL. Esta tarefa não somente é responsável por enviar esses comandos, também é responsável por receber e tratar da maneira adequada as respostas do GRBL de acordo com o comando enviado. A “grbl_comunication_task” é essa tarefa. Ela é uma tarefa consumidora de uma fila de mensagens *First In, First Out* (FIFO) que é populada pelas outras tarefas do sistema, por exemplo, se a tarefa de interface recebe do *display* que uma alteração de configuração no GRBL deva ser realizada, esta envia através de uma fila de mensagens a configuração que deve ser alterada e seu valor. A tarefa responsável pela comunicação irá ler a fila de mensagens. Como é uma fila FIFO, ela irá executar por ordem de chegada. Ao enviar o comando, todo o protocolo de comunicação com o GRBL até a sinalização de conclusão do recebimento é realizado. Caso não possua nenhum comando válido, a tarefa fica suspensa. Sua prioridade precisa ser ajustada de maneira a garantir uma resposta ágil mas sem interferir na fluidez da IHM. A atualização constante do *status* de funcionamento da máquina e de suas coordenadas é crucial no projeto para se saber qual a posição da ferramenta e o estado do *software* GRBL. Por isso, uma tarefa chamada “att_states” foi desenvolvida para, a cada 2.5 segundos, enviar uma solicitação de atualização dos estados da máquina para o GRBL. Essa

tarefa possui a mesma prioridade da tarefa de envio de comandos ao GRBL. A Figura 18 demonstra a interação das tarefas ao acontecer a solicitação de atualização do status e coordenadas do GRBL.

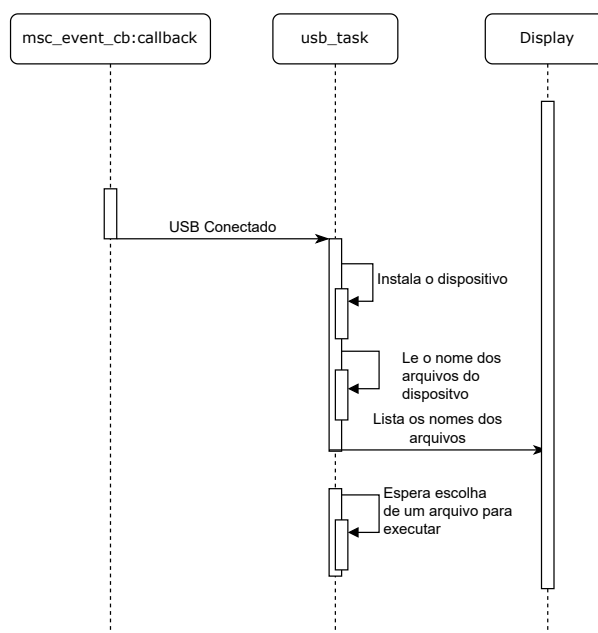
Figura 18 – Interação entre as tarefas para realizar a atualização dos estados da máquina



Fonte: Elaboração própria.

O terceiro núcleo possui as tarefas com a prioridade mais baixa, a “usb_task” e “handle_usb_events”, que realiza operações com dispositivos USB. Dado que a interação com dispositivos USB é esporádica, atribuir-lhe uma prioridade menor não compromete o bom funcionamento do sistema e mantém as funções essenciais com a fluidez necessária. Esse núcleo fica responsável por detectar quando um dispositivo USB é conectado ou desconectado. Para saber se um dispositivo foi conectado ou desconectado, o sistema de “Hospedagem MSC” possui uma função de *callback* chamada de “msc_event_cb” que verifica os eventos de conexão e desconexão. Quando um dispositivo USB é conectado, faz-se sua instalação implementando o sistema de arquivos virtuais para possibilitar a leitura dos arquivos presentes na unidade. Assim, esse núcleo também é responsável pela leitura dos nomes dos arquivos G-Code que podem ser abertos e executados pelo sistema para realizar uma operação de usinagem, gravação ou corte a laser através do envio de linhas de G-code. A interação entre as tarefas em uma conexão de um dispositivo USB é demonstrado através da Figura 19.

Figura 19 – Conexão de um dispositivo USB



Fonte: Elaboração própria.

3.4 Comunicação com o *Display*

A comunicação com o *display* é totalmente realizada via comunicação serial pela porta serial 1. O processo inicial consiste em configurar e ajustar os parâmetros da comunicação serial *Full-Duplex* de maneira que o *display* seja capaz de se comunicar com a interface. O projeto do *display* desenvolvido no *software* UnicView AD, este vem pré-configurado com uma taxa de transmissão (*baudrate*) de 115200 e 1 bit de parada, porém pode ser alterado (VIKTOR VISION, 2017b). Para poder testar esses parâmetros, o primeiro passo foi a implementação de uma comunicação básica com o *display* inteligente. Inicialmente, verificou-se os parâmetros da comunicação serial se estavam adequados, realizando um simples recebimento de uma *string* e o envio de um comando de troca de página. Com o *display* alimentado e as conexões adequadas entre o pinos de recepção e transmissão das duas partes foi possível confirmar o funcionamento básico da comunicação. Com esse teste feito, o *display* possui um protocolo de comunicação específico que deve ser recebido e analisado para se adquirir a informação do mesmo. Entretanto a fabricante fornece uma biblioteca desenvolvida em C (VIKTOR VISION, 2020), assim evitando o desenvolvimento do zero de funções de recebimento desses comandos. Esta biblioteca facilita o estabelecimento da comunicação com o *display* e interação com os elementos do mesmo, tendo funções de envio de comandos também que possuem um protocolo. Foi utilizada essa biblioteca na comunicação para simplificar toda essa interpretação de comandos e simplificar o desenvolvimento dessa comunicação, já que este possui um protocolo específico. Este protocolo é demonstrado no Código 3.3, este também pode ser analisado em detalhes

na página 45 em (VIKTOR VISION, 2017b).

Código 3.3 – Protocolo de comunicação dos *displays* Proculus.

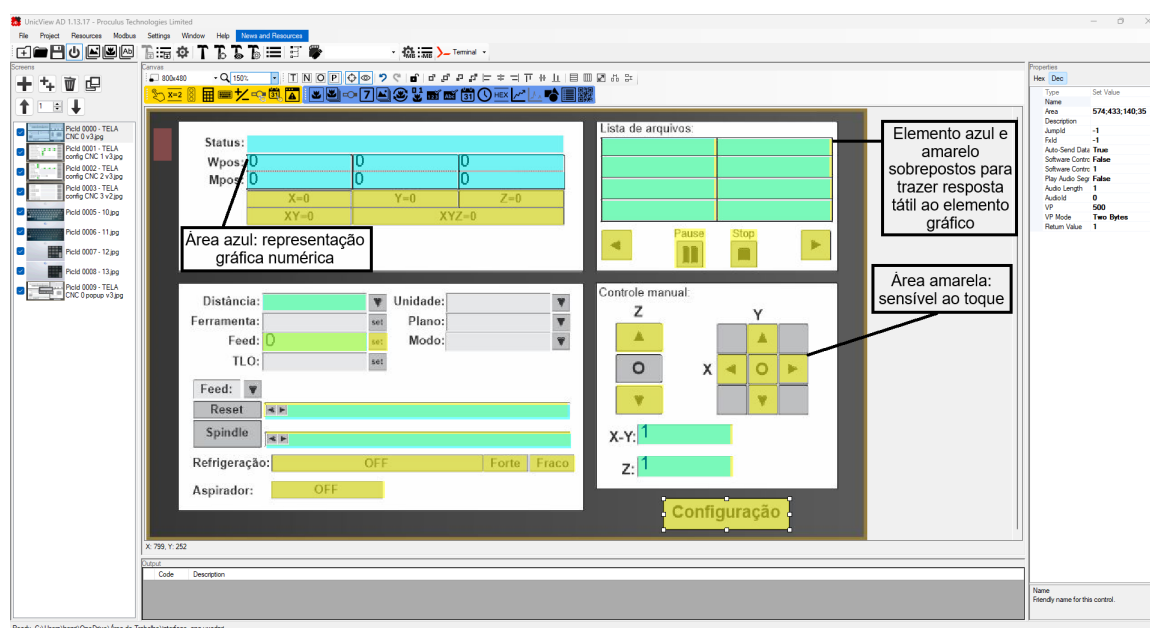
```
63 <Frame Header H> <Frame Header L> <Byte Count> <Command> [<Data>..] [<
    CRC H> <CRC L>]
```

Fonte: (VIKTOR VISION, 2017b).

Uma das funções desta biblioteca é, por exemplo, disponibilizar os valores enviados por determinado elemento do *display* armazenando em sua variável correspondente na aplicação desenvolvida em C. O *display* possui um projeto exemplo disponível em (VIKTOR VISION, 2020), e para validar a biblioteca, carregou-se o exemplo, testando uma série de elementos que o *display* continha. Com esse teste bem-sucedido, foi possível confirmar como êxito a comunicação padrão entre as partes. Além disso, o *display* por ser acompanhado do *software* Unic View AD (VIKTOR VISION, 2023a), que lida com a parte gráfica, dispõe de tecnologia embarcada, permitindo todo o desenvolvimento, estruturação e configuração gráfica. O projeto gráfico gerado pode ser transferido para o *display* através de um pendrive USB. Assim, com o projeto gravado no *display* e as ligações feitas da maneira correta segundo (VIKTOR VISION, 2017a), um microcontrolador recebe e processa as informações de cada elemento de forma apropriada.

Ao abrir o projeto gráfico no UnicView AD, as telas são exibidas conforme a Figura 20. Na barra à esquerda, estão listadas todas as páginas implementadas no projeto. Os retângulos em amarelo representam os elementos funcionais, que respondem exclusivamente ao toque do usuário realizando ações específicas. Os retângulos em azul representam os elementos gráficos que só podem ser modificados por um elemento funcional ou por um comando do microcontrolador, atualizando o valor de seu VP. Esses elementos não possuem interação direta com o usuário e o espaço que ocupam na figura indica onde a informação que representam será exibida. Os elementos funcionais e gráficos podem sobrepor-se como é demonstrado na Figura 20. Para listar os arquivos que serão lidos de um dispositivo USB é colocado elementos *text display* para mostrar o texto dos nomes e se coloca também o elemento *set value* para que no momento em que o usuário selecionar o arquivo, realizar o envio do valor correspondente para o microcontrolador. Os elementos funcionais não podem ser sobrepostos pois estes utilizam da área alocada para ser sua zona de ativação sensível ao toque, portanto, para não haver conflitos, não devem compartilhar essa área de ativação.

Figura 20 – Projeto da Interface aberto no software Unic View AD



Fonte: Elaboração própria.

Cada um desses elementos gráficos e funcionais demonstrados anteriormente no referencial teórico na seção 2.5.1 disponibilizam uma série de opções de configuração que permitem uma adaptação precisa ao projeto (VIKTOR VISION, 2017a). Por exemplo, no caso do elemento “set value”, tem-se a capacidade de mudar o valor do seu VP ao ser ativado. Esses elementos tem a configuração “Auto-Send-Data” que estabelece se o *display* deve transmitir via serial a atualização do valor do VP automaticamente para o microcontrolador ou se deve esperar por um comando de atualização (*pooling*). A configuração “software control” possibilita ao microcontrolador o poder ativar através de uma comando serial a funcionalidade do elemento, no caso do elemento “set value” ele simularia como se o usuário tivesse pressionado a área. Outra possibilidade de configuração é especificar a quantidade de bytes ou bits que um VP deve ocupar na memória do *display*.

A configuração dessas opções é crucial para garantir que os elementos funcionem da maneira desejada. A Figura 21 exemplifica a aba de configuração para o elemento mencionado, onde foi definido que o *display* deve enviar automaticamente o comando serial para o microcontrolador ao ser pressionado, nesse caso o valor para o seu VP é zero e o VP ocupa dois bytes na memória.

Figura 21 – Aba de configurações Unic View AD

Properties	
Type	Set Value
Name	
Area	193;244;133;95
Description	
JumpId	-1
FxId	-1
Auto-Send Data	True
Software Contro	False
Software Contro	1
Play Audio Segr	False
Audio Length	1
Audiold	0
VP	0
VP Mode	Two Bytes
Return Value	0

Fonte: Elaboração própria.

Essas configurações permitem a personalização dos elementos garantindo que eles atendam às necessidades específicas do projeto e funcionem em conformidade com os requisitos do sistema. Essa abordagem possibilita a combinação de elementos funcionais e gráficos, permitindo a criação de funcionalidades complexas. Por exemplo, é possível utilizar um elemento funcional, como uma caixa de entrada numérica (*numeric input*), para compartilhar o mesmo VP de valor com um elemento gráfico, como uma área de exibição de números (*numeric display*). Essa combinação é demonstrada na Figura 22 onde é utilizada essa combinação para realizar a entrada da distância para realizar um movimento individual “jog” dos eixos X, Y e Z da máquina CNC.

Figura 22 – Área de controle manual da página inicial

Controle manual:

Z		Y
▲		▲
○	X	○
▼		▼
X-Y: 1		
Z: 1		

Fonte: Elaboração própria.

Essa combinação entre os elementos funcionais e gráficos é fundamental, pois permite que o elemento funcional *numeric input* atualize o VP com os dados inseridos pelo usuário através de um teclado numérico que pode ser criado pelo desenvolvedor. A Figura 23 demonstra um exemplo de teclado numérico. O *numeric input* invoca o teclado numérico através das configurações que podem ser feitas

dizendo qual página do projeto possui o teclado e qual a área do mesmo. Em resposta, o elemento gráfico *numeric display* recebe e reflete essas alterações na tela do *display*. Além disso, o microcontrolador pode manipular o VP, possibilitando a atualização e modificação dos números exibidos no visor de acordo com a lógica do sistema. Essa integração eficaz entre os elementos funcionais e gráficos proporciona uma experiência para o usuário flexível e dinâmica no projeto do *display*.

Figura 23 – Exemplo de teclado numérico implementado



Fonte: Elaboração própria.

Cada VP abrange normalmente dois bytes. Dependendo do elemento, é possível configurar os VPs de 1 bit até oito bytes, permitindo que um elemento ocupe um ou vários VPs, dependendo do tipo de elemento. Cada elemento criado no *software* UnicView AD, seja gráfico ou funcional, requer a criação de uma variável correspondente no *software* desenvolvido em C para o microcontrolador. O Código 3.4 a seguir demonstra a declaração de algumas variáveis feitas que levam em consideração o tipo de variável, seu VP e o tipo do mesmo.

Código 3.4 – Declaração de variáveis do *display*

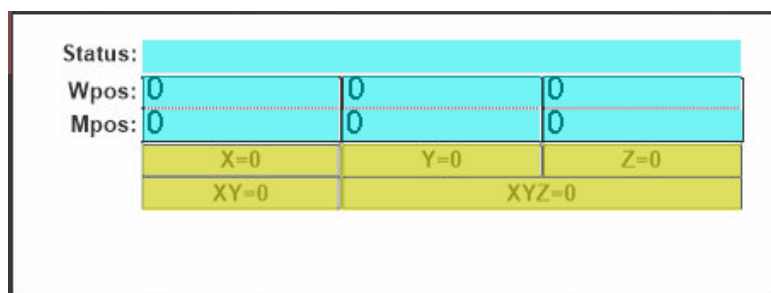
```
64 //Posições da maquina de cada eixo
65 lcm_var_parameters_t Mpos_X = { 12, FOUR_BYTES };
66 lcm_var_parameters_t Mpos_Y = { 16, FOUR_BYTES };
67 lcm_var_parameters_t Mpos_Z = { 20, FOUR_BYTES };
68 //Status atual da maquina
69 lcm_string_parameters_t Status = { 24, 10 };
70 //Botão responsavel por zerar a posição de um ou mais eixos
```

```
71 lcm_var_parameters_t zera_param = { 35, TWO_BYTES };
```

Fonte: Elaboração própria.

A Figura 24 mostra os elementos equivalentes para essas variáveis declaradas. Dessa forma, o *display* pode enviar uma atualização de valor específica de um VP, assim o microcontrolador recebe, armazena e posteriormente lê e processa adequadamente.

Figura 24 – Elementos equivalentes aos declarados no Código 3.4



Fonte: Elaboração própria.

3.4.1 Buffer Circular e uso da Biblioteca do Hardware

Para garantir a integridade das informações entre os elementos estabelecidos no *display* e no microcontrolador, foi implementado um *buffer* circular, desenvolvido com base em um exemplo disponibilizado por (ESPRESSIF SYSTEM, 2023a). Este *buffer* circular deve armazenar todos os comandos recebidos pelo *display*. A tarefa "rx_task" é a responsável por executar essa função que recebe um comando, verifica seu tamanho buscando o byte do protocolo que informa o tamanho da mensagem e o copia para o *buffer*. Caso o *buffer* não tenha espaço suficiente para armazenar o comando completo, ele faz uma cópia parcial para a parte disponível e armazena a parte restante no início do *buffer*, justificando o nome "*buffer* circular". A função de recebimento é demonstrada no Código 3.5 sendo este implementado em C com as funções particulares do *framework* ESP-IDF.

Código 3.5 – Função de recebimento e armazenamento de informação no buffer circular a partir da comunicação serial.

```
72 void receive_on_buffer(void) {
73     uint8_t info[3]; // Array para armazenar informações de pacotes
74
75     // Receba os primeiros 3 bytes (informações do pacote)
76     const int rxBytes = receiveData(RX_TASK_TAG, info, 3);
77
78     if (rxBytes > 0) {
79         // Atualiza a última posição antes de copiar novos dados
```

```
80     oldPos = newPos;
81
82     // Extraia packetSize do terceiro byte (assumindo little-endian)
83     packet.packetSize = (int) info[2];
84
85     // Se a posição atual + o novo tamanho dos dados for maior
86     // que o buffer principal
87     if (oldPos + (3 + (packet.packetSize)) > RX_BUF_SIZE)
88     {
89         // Descubra quanto espaço resta no buffer principal
90         datatocopy = RX_BUF_SIZE - oldPos;
91         if (datatocopy < 3) {
92
93             memcpy((uint8_t*) lcm_rx_ring_buffer + oldPos,
94                    (uint8_t*) info,
95                    datatocopy);
96
97             oldPos = 0;
98             // Copie os dados restantes
99             memcpy((uint8_t*) lcm_rx_ring_buffer,
100                   (uint8_t*) info + datatocopy,
101                   (3 - datatocopy));
102
103             receiveData(RX_TASK_TAG,
104                        (uint8_t*) lcm_rx_ring_buffer + (3 - datatocopy),
105                        packet.packetSize);
106             // Atualizar a posição
107             newPos = (packet.packetSize + 3 - datatocopy);
108         } else {
109             memcpy((uint8_t*) lcm_rx_ring_buffer + oldPos,
110                    (uint8_t*) info,
111                    3);
112
113             receiveData(RX_TASK_TAG,
114                        (uint8_t*) lcm_rx_ring_buffer + (3 + oldPos),
115                        (datatocopy - 3));
116             oldPos = 0;
117
118             receiveData(RX_TASK_TAG,
119                        (uint8_t*) lcm_rx_ring_buffer,
120                        (packet.packetSize - datatocopy + 3));
121             // Atualizar a posição
122             newPos = (packet.packetSize + datatocopy - 3);
123         }
124     } else {
125         memcpy((uint8_t*) lcm_rx_ring_buffer + oldPos, (uint8_t*) info,
126               3);
```

```

126     receiveData(RX_TASK_TAG,
127                (uint8_t*) lcm_rx_ring_buffer + oldPos + 3,
128                (packet.packetSize));
129         // Atualizar a posição
130         newPos = ((3 + (packet.packetSize)) + oldPos);
131     }
132 }
133 /* se a posição atual + o novo tamanho dos dados for menor que o
134    buffer principal
135 */
136 }

```

Fonte: Elaboração própria.

As informações contidas no *buffer* são lidas por uma função “*lcm_read_byte()*” da biblioteca fornecida pelo fabricante do *display* em Viktor Vision (2020). Ela foi alterada somente para acessar o *buffer* circular e saber sua posição atual. O Código 3.6 demonstra a função citada.

Código 3.6 – Função de leitura de dados do display a partir *buffer* circular.

```

137 uint16_t lcm_read_byte() // Função de leitura da serial utilizada pela
138                          biblioteca UniconViewAD.lcm
139 {
140     if (newPos != lcm_rx_ring_buffer_received_index) {
141         uint8_t data = read_lcm_ring_buffer(
142             lcm_rx_ring_buffer_received_index);
143         ++lcm_rx_ring_buffer_received_index;
144         if (RX_BUF_SIZE == lcm_rx_ring_buffer_received_index) {
145             lcm_rx_ring_buffer_received_index = 0;
146         }
147         return data;
148     } else {
149         return LCM_EMPTY_RESPONSE;
150     }
151 }

```

Fonte: (VIKTOR VISION, 2020).

Essa função desempenha um papel fundamental no processamento de comandos recebidos, permitindo a atualização dos valores das variáveis correspondentes a partir de um comando recebido. Isso viabiliza a utilização de outras funções para verificar se houve alguma atualização nos dados. A função mais comumente empregada nesse contexto é “*lcm_var_available()*”, a qual requer que seja especificada a variável a ser monitorada. Ela realiza uma verificação para determinar se ocorreu

alguma atualização nos dados. Em caso afirmativo, a função retorna o valor 1; caso contrário, retorna 0. Esta função é demonstrada no Código 3.7.

No caso de detecção de uma atualização, o valor recém-atualizado pode ser obtido utilizando a função `lcm_var_get_data()`. Essa função realiza a leitura do valor associado à variável fornecida, permitindo que ele seja processado de acordo com as necessidades da aplicação. Esse mecanismo de monitoramento e leitura de valores atualizados possibilita a integração eficaz de comandos recebidos no fluxo de trabalho da aplicação. A seguir tem o Código 3.7 que utiliza esse mecanismo de verificação de variável, nesse caso verifica se há uma troca de página, e se for o caso, para qual página deve-se ir.

Código 3.7 – Rotina para verificação de elemento e leitura de seu valor

```
150 //Verifica se chegou alguma atualização do valor da variavel
151 if (lcm_var_available(&button_pag)) {
152 //Le o valor da variavel que corresponde a qual pagina do display
    deve ser trocada
153     pageid = lcm_var_get_data(&button_pag);
154 //Muda a página do display de acordo com o valor recebido
155     lcm_change_pic_id(pageid);
156 }
```

Fonte: Elaboração própria.

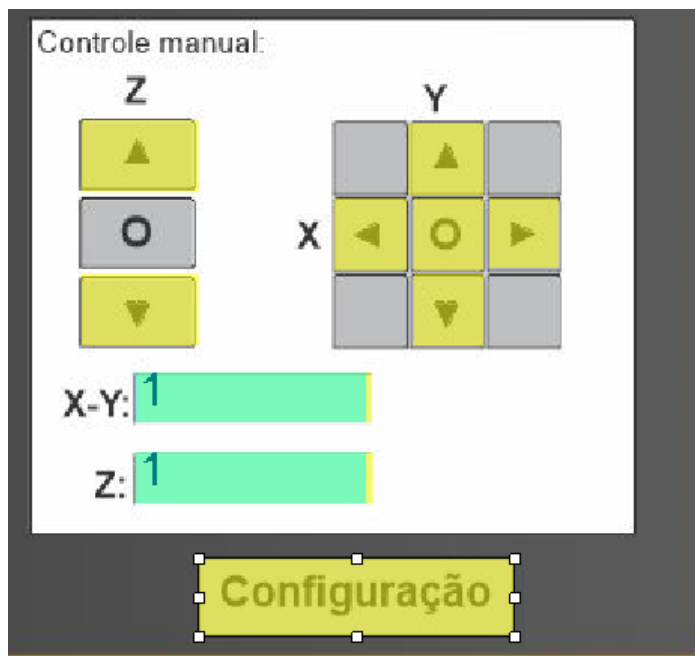
Para validar essas funções, simulou-se o envio de mensagens com o protocolo padrão do *display* de maneira periódica para ocupar completamente o *buffer*. Isso permitiu verificar se havia algum problema com a função diante dos possíveis casos de *overflow* do *buffer*, assegurando que ele conseguisse executar a rotação da posição de recebimento sem prejudicar nenhuma informação. Com o *buffer* validado, foi realizado o teste final, executando o exemplo fornecido pela fabricante com o uso do *buffer* desenvolvido, concluindo assim os testes de funcionamento de toda a comunicação com o *display*.

3.4.2 Máquina de Estado para controle das páginas e interação com Elementos

Para otimizar o sistema, o processo de verificação de variáveis que correspondem a elementos do *display* e suas respostas são gerenciados por uma máquina de estados. Essa máquina de estados é dividida em páginas que correspondem às mesmas do *display* estas foram demonstradas anteriormente na Figura 16, as funções da máquina de estado são alinhadas com os elementos das páginas criadas e desenhadas no *software* gráfico. Por exemplo, a página inicial mostrada anteriormente possui um botão para troca de página destacado na Figura 25, ao se pressionar o

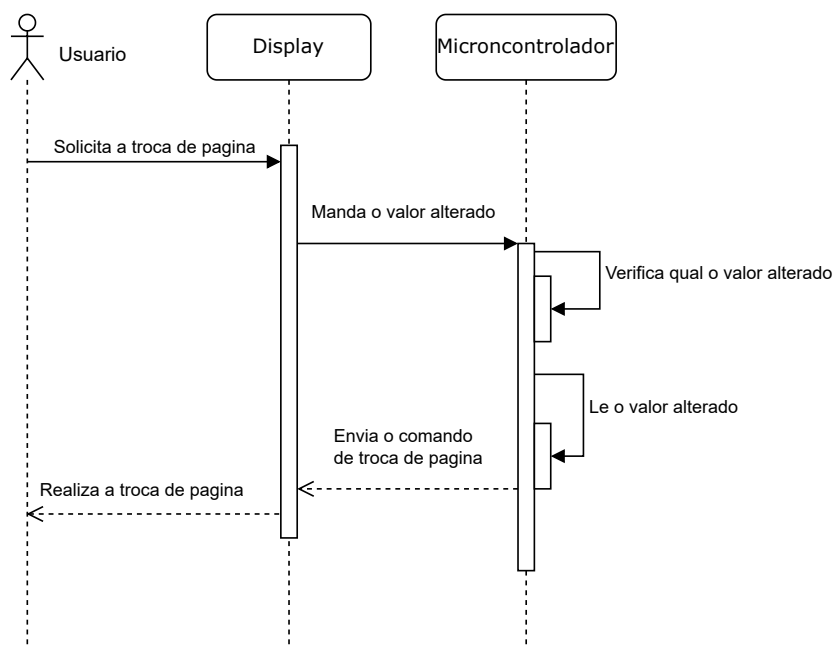
botão “Configuração”, realiza-se uma troca para a página de configurações e dando acesso a outras duas páginas que contém as principais configurações do GRBL.

Figura 25 – Botão para acesso as páginas de configuração do sistema



Fonte: Elaboração própria.

Cada estado tem uma função que representa a sua página correspondente, ou seja, contém e verifica apenas as variáveis que representam os elementos da página que está sendo usada naquele momento, garantindo um funcionamento suave e eficaz do sistema. Isso permite que cada estado e sua página correspondente interajam somente com os elementos correspondentes. Sempre que há uma troca de página, essa ação é realizada pelo microcontrolador, o que resulta na atualização da máquina de estados. A solicitação da troca de página é iniciada pelo *display*, porém, a troca e a realização desta interação é realizado pelo microcontrolador, assegurando uma interação adequada entre o *display* e o microcontrolador mantendo uma sincronização entre estado atual e página atual. Isso garante um tempo de resposta eficaz e uma experiência fluida para o usuário, o que promove uma boa experiência de uso mantendo também o controle e sincronização do sistema. A seguir é demonstrada na Figura 26 as interações realizadas na troca de página entre o *display* e o microcontrolador.

Figura 26 – Diagrama de sequência para realizar a troca de página *Display*

Fonte: Elaboração própria.

Com toda a comunicação implementada, é fundamental verificar a fluidez qualitativa da interface gráfica verificando a velocidade da troca de páginas e as demais interações do *display*. A avaliação da fluidez do sistema foi realizada por meio de comparações com *softwares* executados em computadores de mesa, levando em consideração o desempenho geral em termos de fluidez de uso. Apesar das limitações inerentes ao microcontrolador utilizado, a fluidez alcançada foi considerada satisfatória, especialmente quando comparada a sistemas em computadores com capacidade de processamento significativamente superior.

3.5 GRBL comunicação

Como citado anteriormente a comunicação com o GRBL é efetuada através de comunicação serial com protocolo específico. Boa parte da implementação baseou-se na documentação (GRBL, 2022), que orienta sobre a implementação de uma interface de maneira geral. Compreendendo o protocolo e os parâmetros de comunicação fornecidos pela documentação, estabeleceram-se funções para enviar comandos específicos e tratar as informações de retorno do GRBL. Os comandos de configuração do GRBL geralmente começam com o caractere “\$” e possuem uma sequência definida com base no comando desejado. Outros comandos podem ser compostos por um único caractere. Independentemente do tipo de comando, é essencial que todos eles terminem com os caracteres de finalização “CR” ou “LF”, pois o GRBL confirma o término do comando com esses caracteres. Além dos comandos G-code e

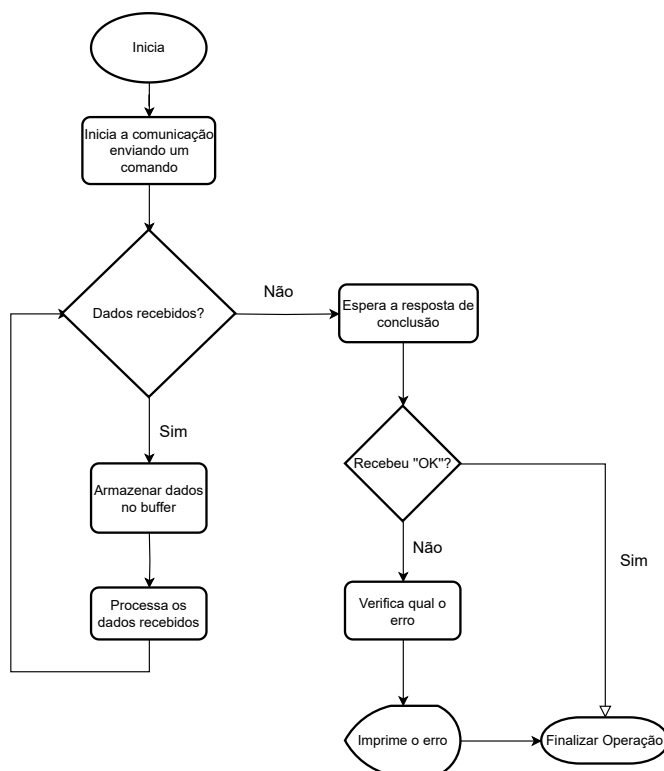
M-code, o GRBL possui uma série de comandos próprios estes são demonstrados no Quadro 4.

Quadro 4 – Tabela de Comandos GRBL

Código	Descrição
\$\$	Exibir configurações do Grbl.
\$x=val	Altere a configuração Grbl x para val.
\$#	Veja os parâmetros do GCode.
\$G	Veja o estado do analisador GCode.
\$C	Alternar modo de verificação Gcode
\$H	Executar ciclo de retorno
\$J=gcode	Execute o movimento de corrida.
\$X	Mate o estado de bloqueio de alarme.
\$I	Ver informações da versão
\$N	Ver código de inicialização salvo
\$Nx=line	Salvar linha GCode de inicialização (x = 0 ou 1) São executados em um reset.
RST =	Restaura as configurações do Grbl para os padrões.
\$RST=#	Apaga os deslocamentos WCS G54-G59 e as posições G28/30 armazenadas na EEPROM.
\$RST=*	Limpe e carregue todos os dados da EEPROM.
\$SLP	Ative o modo de suspensão.
Ctrl-x	Reinicialização suave
?	Consulta de relatório de status.
~	Início/retomar do ciclo a partir de suspensão de alimentação, porta ou pausa de programa.
!	Feed Hold – Pare todo o movimento.

Fonte: Adaptação de (Sainsmart, 2022).

Para enviar esses comandos personalizados ao GRBL, um protocolo específico é adotado, incluindo o envio, recebimento e conclusão da comunicação. O diagrama da Figura 27 ilustra o protocolo padrão que representa o funcionamento básico de todas as funções necessárias para alcançar as funcionalidades desejadas.

Figura 27 – Fluxograma de funcionamento das funções de comunicação GRBL

Fonte: Elaboração própria.

Seguindo a lógica descrita acima, torna-se possível enviar e receber todos os comandos desejados para o GRBL. Uma biblioteca de comunicação foi desenvolvida com base nessa lógica, implementando os comandos necessários para o funcionamento da interface. Após a implementação da biblioteca de funções de comunicação, a próxima etapa envolve a configuração da comunicação com o GRBL.

A comunicação com o GRBL deve ser configurada de acordo com os padrões do GRBL, os quais podem ser alterados se assim for desejado, considerando limitações do microcontrolador ou outros fatores pertinentes. O código a seguir demonstra a variável que configura a comunicação serial do ESP32S3 com o GRBL.

Código 3.8 – Configuração da Uart

```

157 const uart_config_t uart_config = {
158     .baud_rate = 115200,
159     .data_bits = UART_DATA_8_BITS,
160     .parity = UART_PARITY_DISABLE,
161     .stop_bits = UART_STOP_BITS_1,
162     .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
163     .source_clk = UART_SCLK_DEFAULT, };
  
```

Fonte: Elaboração própria.

Os padrões de comunicação serial do GRBL podem ser personalizados no arquivo “config.h” do projeto (GRBL, 2021). Para gerenciar essa comunicação uma tarefa chamada “*grbl_communication_task()*” é encarregada apenas do controle da comunicação e acesso à porta de comunicação serial. Esta tarefa possui uma fila de mensagens, que é um mecanismo do FreeRTOS para permitir a comunicação entre duas ou mais tarefas. As tarefas produtivas populam a fila de mensagens, enquanto a tarefa consumidora lê e esvazia a fila. A tarefa consumidora é responsável por receber e ler uma mensagem da fila e processá-la adequadamente, chamando a função da biblioteca desenvolvida correspondente ao comando a ser executado. O Código 3.9 exemplifica essa tarefa.

Código 3.9 – Tarefa de envio de comandos GRBL

```
164 void grbl_communication_task(void *arg) {
165     GrblData update;
166     //Ao iniciar a aplicação faz a leitura das configurações atuais do
167     //GRBL
168     att_conf = 1;
169     //Envia um feed rate padrão
170     send_to_grbl("F100\r\n");
171
172     while (1) {
173
174         //Verifica se de ser feita a leitura das configurações do GRBL
175         if (att_conf == 1) {
176             grbl_get_config();
177             configs_info = get_configs();
178             att_conf = 0;
179         }
180         //Verifica se tem algum item na fila para ser processado
181         if (xQueueReceive(message_queue, &update,
182             portMAX_DELAY) == pdTRUE) {
183
184             // Processa os dados da estrutura recebido na fila e executa os
185             // comando de acordo
186             switch (update.selec_func) {
187                 case 1:
188                     grbl_set_config(update.parametro, update.value1);
189                     break;
190                 case 2:
191                     ESP_LOGI(TAG, "Tenta o jog");
192                     grbl_jog(update.value1, update.value2, update.value3);
193                     break;
194                 case 3:
195                     grbl_set_axis_zero(update.value3);
196                     break;
```

```
196     case 4:
197         if (xSemaphoreTake(data_mutex, portMAX_DELAY)) {
198             grbl_get_state();
199             xSemaphoreGive(data_mutex);
200         }
201         break;
202     default:
203         break;
204 }
205 }
206 }
207 }
```

Fonte: Elaboração própria.

Essa fila de mensagens é criada de acordo com o número máximo de mensagens que se deseja receber, no caso do projeto foi estipulado trinta e duas mensagens. O tamanho da mensagem a ser recebida também impacta na criação da fila pois impacta na área de memória a ser alocada. No caso do projeto desenvolvido, foi estabelecido uma estrutura de dados para ser o padrão de mensagens da fila. Esta estrutura possui cinco parâmetros, em que o parâmetro “selec_func” é responsável por selecionar o tipo de comando a ser enviado assim selecionando a função correspondente a esse da biblioteca desenvolvida. De acordo com o comando selecionado determina-se quais parâmetros devem ser lidos. A estrutura de dados é demonstrada abaixo.

Código 3.10 – Estrutura para envio de comando para o GRBL

```
208 typedef struct {
209     uint8_t selec_func;
210     char *parametro;
211     float value1;
212     float value2;
213     uint8_t value3;
214 } GrblData;
```

Fonte: Elaboração própria.

A função “grbl_get_state()” é a única função que apresenta restrição de funcionamento, pois duas ou mais tarefas podem acessar a mesma variável. Portanto, é utilizado um *mutex* para garantir acesso singular aos dados. Cada vez que o sistema deseja enviar um comando ao GRBL, envia-se a estrutura de dados para a fila de mensagens, contendo o comando a ser executado e os parâmetros a serem passados. O Código 3.11 ilustra o envio de uma mensagem à fila.

Código 3.11 – Exemplo de envio de comando para o GRBL

```
215 // Enviar o comando diretamente:
216 // grbl_set_config("12",
217 lcm_var_get_data(&arc_tolerance));
218
219 Update_data.selec_func = 1;
220 Update_data.value1 = lcm_var_get_data(&arc_tolerance);
221 Update_data.parametro = "12";
222 xQueueSend(message_queue, &Update_data, 100 / portTICK_PERIOD_MS);
```

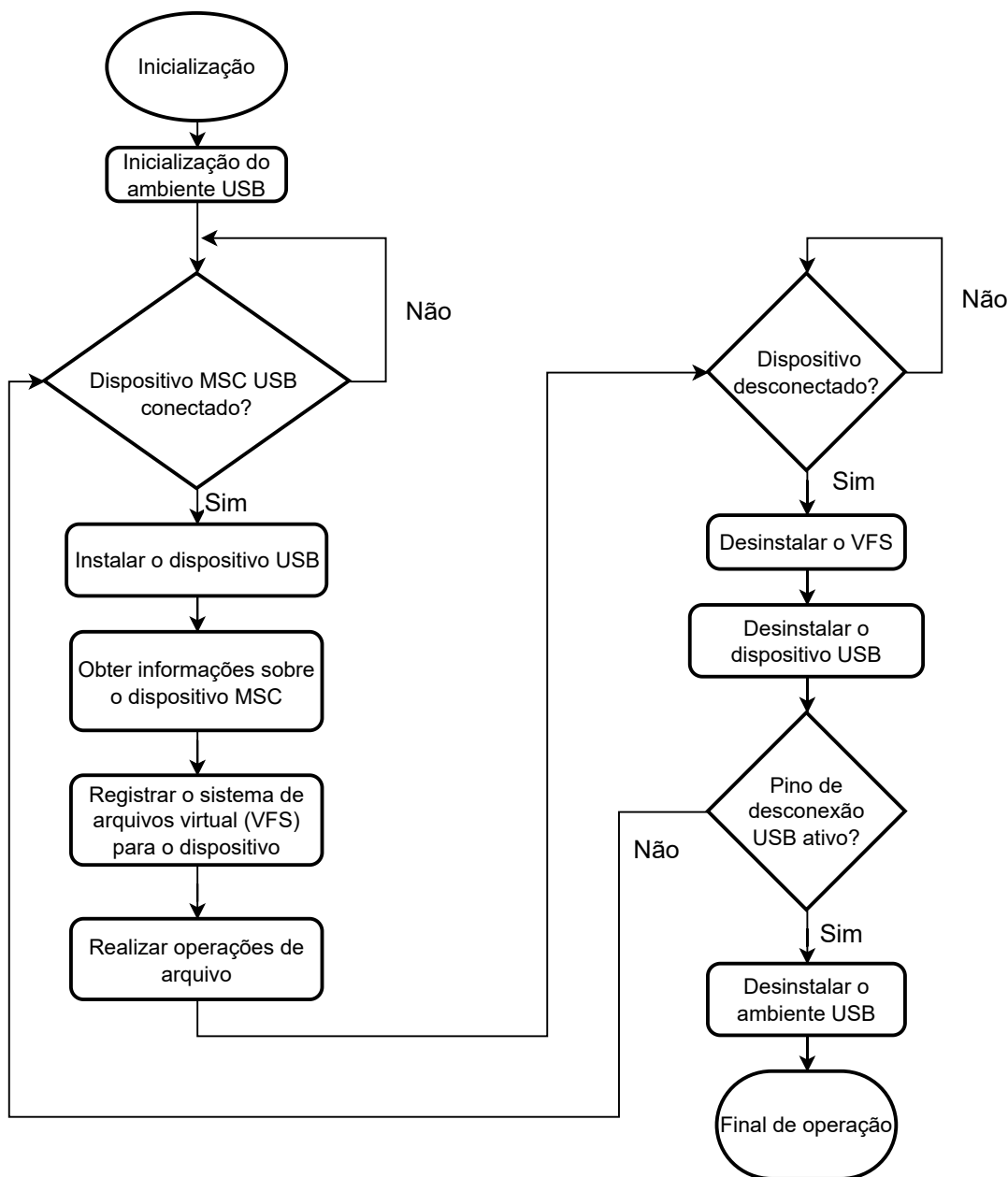
Fonte: Elaboração própria.

A implementação da fila de mensagens é fundamental para o envio e processamento de comandos. Com essa estrutura implementada, a comunicação com o GRBL é completamente funcional. Para poder verificar o funcionamento como um todo inicialmente, utilizando uma placa Arduino UNO, conectaram-se os pinos de recepção e transmissão desta à ESP32-S3. Com a conexão estabelecida e os parâmetros configurados corretamente, iniciaram-se os testes, enviando comandos simples, como solicitar a versão do GRBL, para verificar se a mensagem esperada com a versão correta era recebida. Com o sucesso dos testes iniciais, procederam para testes mais complexos, como a recepção das configurações do GRBL, permitindo validar se as configurações foram alteradas corretamente. Uma das primeiras funções implementadas foi o recebimento dessas configurações. Como a maioria dos comandos do GRBL envia uma mensagem de “OK” para sinalizar a execução correta, ou uma mensagem de erro em caso contrário, desenvolveram-se funções que verificam e imprimem mensagens de erro quando necessário. Realizou-se testes práticos em uma placa Arduino UNO que executa o *software* GRBL, enviando comandos de movimento e troca de padrões de movimento para validar a comunicação.

3.6 Sistema de Leitura de dispositivos de armazenamento USB

O sistema de leitura de dispositivos de armazenamento USB, como unidades *flash* foi baseado no exemplo (ESPRESSIF SYSTEM, 2023b), e foi adaptado para a aplicação final. Este exemplo opera detectando a conexão de um dispositivo USB e foi adaptado para se realizar a abertura de uma pasta específica do dispositivo. Se a abertura da pasta for bem sucedida são listados até 32 arquivos para disponibilizá-los ao usuário. Embora o sistema seja compacto, ele atende a todas as necessidades da aplicação. Uma biblioteca com funções desenvolvidas foi criada para executar esse sistema. O diagrama da Figura 28 demonstra o funcionamento base do sistema, ilustrando os passos que o sistema executa para a operação mencionada anteriormente.

Figura 28 – Fluxograma de funcionamento sistema USB



Fonte: Elaboração própria.

O primeiro passo é uma função de inicialização, na qual são definidas as configurações do sistema. A função é demonstrada no Código 3.12. Neste sistema USB, há um pino para informar o fim da conexão sua configuração e feita no início da função no Código citado. Além disso, é utilizada uma função do FreeRTOS chamada *Event Groups*, por meio da qual é criado um grupo de bits que pode representar uma série de eventos importantes relacionados ao sistema USB e seus dispositivos. A configuração do sistema de *Host* USB define o nível de interrupção para os eventos.

Código 3.12 – Função de inicialização do sistema USB

```
223 void usb_config_init(void) {
```

```

224 BaseType_t task_created;
225 //Definição das configurações do pino que sinaliza o desligamento
do sistema USB
226 const gpio_config_t input_pin = {
227     .pin_bit_mask = BIT64(USB_DISCONNECT_PIN),
228     .mode = GPIO_MODE_INPUT,
229     .pull_up_en = GPIO_PULLUP_ENABLE, };
230 //Verificação da correta criação
231 ESP_ERROR_CHECK(gpio_config(&input_pin));
232
233 //Criação de um Grupo de bits que sinalizam algum evento
234 usb_flags = xEventGroupCreate();
235 assert(usb_flags);
236
237 //Configuração do nível das interrupções do sistema USB Host
238 const usb_host_config_t host_config = { .intr_flags =
ESP_INTR_FLAG_LEVEL3 };
239 ESP_ERROR_CHECK(usb_host_install(&host_config));
240
241 //Criação da task para lidar com os eventos do sistema USB
242 task_created = xTaskCreate(handle_usb_events,
243     "usb_events",
244     2048 * 4,
245     NULL,
246     configMAX_PRIORITIES - 23,
247     NULL);
248
249 assert(task_created);
250
251 //Criação da função de callback do sistema Host MSC
252 const msc_host_driver_config_t msc_config =
253 { .create_background_task = true,
254     .task_priority = 5,
255     .stack_size = 2048 * 4,
256     .callback = msc_event_cb, };
257
258 ESP_ERROR_CHECK(msc_host_install(&msc_config));
259 }

```

Fonte: Elaboração própria.

Em seguida no final do Código 3.12, é criada uma tarefa para lidar com esses eventos, responsável por realizar verificações e sinalizar a conclusão das ações. A implementação desta tarefa é demonstrada no Código 3.13.

Código 3.13 – Função que lida com os eventos do hospedeiro USB

```

260 // Lida com eventos comuns da biblioteca host USB
261 void handle_usb_events(void *args) {

```

```

262
263 while (1) {
264
265     uint32_t event_flags;
266
267     usb_host_lib_handle_events(portMAX_DELAY, &event_flags);
268
269     // Liberar dispositivos assim que o registro de todos os clientes
270     for cancelado
271     if (event_flags & USB_HOST_LIB_EVENT_FLAGS_NO_CLIENTS) {
272         usb_host_device_free_all();
273         xEventGroupSetBits(usb_flags, HOST_NO_CLIENT);
274     }
275     // Dê o semáforo ready_to_uninstall_usb para indicar essa
276     biblioteca USB Host
277     // pode ser desinicializado e encerrar esta tarefa
278     if (event_flags & USB_HOST_LIB_EVENT_FLAGS_ALL_FREE) {
279         xEventGroupSetBits(usb_flags, HOST_ALL_FREE);
280     }
281 }
282 vTaskDelete(NULL);
283 }

```

Fonte: Adaptado de (ESPRESSIF SYSTEM, 2023b).

O sistema de leitura de dispositivos de armazenamento em massa (*Mass Storage Class* - MSC) é configurado para lidar com eventos de conexão e desconexão dos dispositivos. A função de *callback* responsável por verificar e processar esses eventos é demonstrada no Código 3.14.

Código 3.14 – Função de callback para os eventos de dispositivos MSC

```

282 void msc_event_cb(const msc_host_event_t *event, void *arg) {
283
284     if (event->event == MSC_DEVICE_CONNECTED) {
285         ESP_LOGI(TAG, "MSC device connected");
286         // O endereço do dispositivo USB obtido é colocado após eventos da
287         aplicação
288         xEventGroupSetBits(usb_flags,
289             DEVICE_CONNECTED | (event->device.address << 4));
290
291         usb_status = 1;
292     } else if (event->event == MSC_DEVICE_DISCONNECTED) {
293
294         xEventGroupSetBits(usb_flags, DEVICE_DISCONNECTED);
295
296         ESP_LOGI(TAG, "MSC device disconnected");

```

```
297
298 //Apaga os nomes dos arquivos
299 ESP_LOGI(TAG, "Apaga os nomes dos arquivos");
300 for (int k = 0; k < 8; k++) {
301     lcm_string_write(nome_arqs[k], (uint8_t*) "", 1);
302 }
303 usb_status = 3;
304 }
305 }
```

Fonte: Adaptado de (ESPRESSIF SYSTEM, 2023b).

Essa função monitora se um dispositivo USB foi conectado ou desconectado e sinaliza para as demais tarefas do sistema por meio de bits de evento e *flags* o que permite o gerenciamento dos eventos de conexão ou desconexão do dispositivo.

Com essas funcionalidades implementadas, é estabelecida a tarefa “control_usb” esta que possui uma maquina de estados onde seu primeiro estado aguarda por um dispositivo válido ser conectado. Quando um dispositivo é conectado, ele é instalado no sistema e as operações são iniciadas. Após a conexão, o sistema troca de estado e aguarda a seleção do arquivo por parte do usuário e executa as ações desejadas. Ao ser desconectado e feita a troca de estado novamente, e o dispositivo é desinstalado, indicando o sucesso da operação. A tarefa que realiza esse processo é demonstrada no Código 3.15.

Código 3.15 – Função de controle de ações de dispositivos de memória flash

```
306 void control_usb(void) {
307     switch (usb_status) {
308     case 1:
309         //Aguarda um do dispositivo de memoria flash ser conectado
310         uint8_t device_address = wait_for_msc_device();
311         if (device_address != 0) {
312             //Realiza a instalação do dispositivo
313             ESP_ERROR_CHECK(msc_host_install_device(device_address, &
314 msc_device));
315
316             msc_host_print_descriptors(msc_device);
317
318             ESP_ERROR_CHECK(msc_host_get_device_info(msc_device, &info));
319
320             print_device_info(&info);
321
322             ESP_ERROR_CHECK(msc_host_vfs_register(msc_device,
323 "/usb", &mount_config,
324 &vfs_handle));
325             //Realiza a leitura do nomes dos arquivos presentes no
```

```
dispositivo USB
325     if (xSemaphoreTake(names_mutex, portMAX_DELAY)) {
326         file_operations();
327         xSemaphoreGive(names_mutex);
328     }
329     usb_status = 2;
330 }
331 break;
332 case 2:
333 //Aguarda a seleção de um arquivo para ser executado
334     select_file(directory);
335     vTaskDelay(1000 / portTICK_PERIOD_MS);
336     break;
337 case 3:
338 //Apaga os nomes dos arquivos
339     if (xSemaphoreTake(names_mutex, portMAX_DELAY)) {
340         memset(file_names, 0, sizeof(file_names));
341         xSemaphoreGive(names_mutex);
342     }
343 //Realiza a desinstalação do dispositivo de memoria flash
344     ESP_LOGI(TAG, "Uninstall Device");
345     xEventGroupClearBits(usb_flags, READY_TO_UNINSTALL);
346     ESP_ERROR_CHECK(msc_host_vfs_unregister(vfs_handle));
347     ESP_ERROR_CHECK(msc_host_uninstall_device(msc_device));
348     usb_status = 0;
349     break;
350 default:
351     vTaskDelay(100 / portTICK_PERIOD_MS);
352     break;
353 }
354 }
```

Fonte: Elaboração própria.

Funções auxiliares também foram criadas para realizar a leitura, escrita e criação de arquivos em diretórios específicos, seguindo uma aplicação básica desenvolvida em C. Neste projeto específico, foram implementadas funções para ler os nomes de 32 arquivos em uma determinada pasta e exibir esses nomes no *display*. Ao selecionar um dos arquivos, o sistema o abre e envia linha por linha para o *software* GRBL executar os comandos G-code.

Para validar toda a implementação realizada foram feitos alguns testes. Para a leitura do USB, validou-se inicialmente o exemplo em Espressif System (2023b), que serviu de base para o desenvolvimento. Esse exemplo envia informações do dispositivo para o monitor serial e cria um arquivo de texto no pendrive com a frase “*Hello Word*”. Após verificar que o sistema reconheceu corretamente o dispositivo e criava o arquivo

esperado, o código foi modificado para atender à aplicação deste trabalho. Em seguida, testou-se a leitura de um arquivo selecionado pelo usuário, verificando os comandos G-code através de um monitor serial enviados pela interface e validando as informações recebidas de confirmações do GRBL.

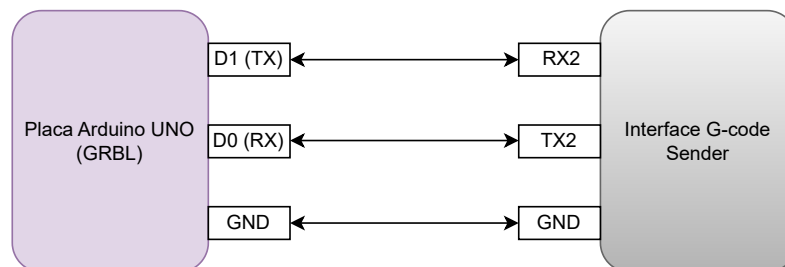
4 RESULTADOS

Neste capítulo serão apresentados os resultados alcançados do desenvolvimento do projeto e o processo de teste das funções implementadas. Além disso, serão abordados os procedimentos adotados para realizar testes práticos do sistema como um todo, juntamente com possíveis comparações com tecnologias concorrentes do mercado.

4.1 Testes

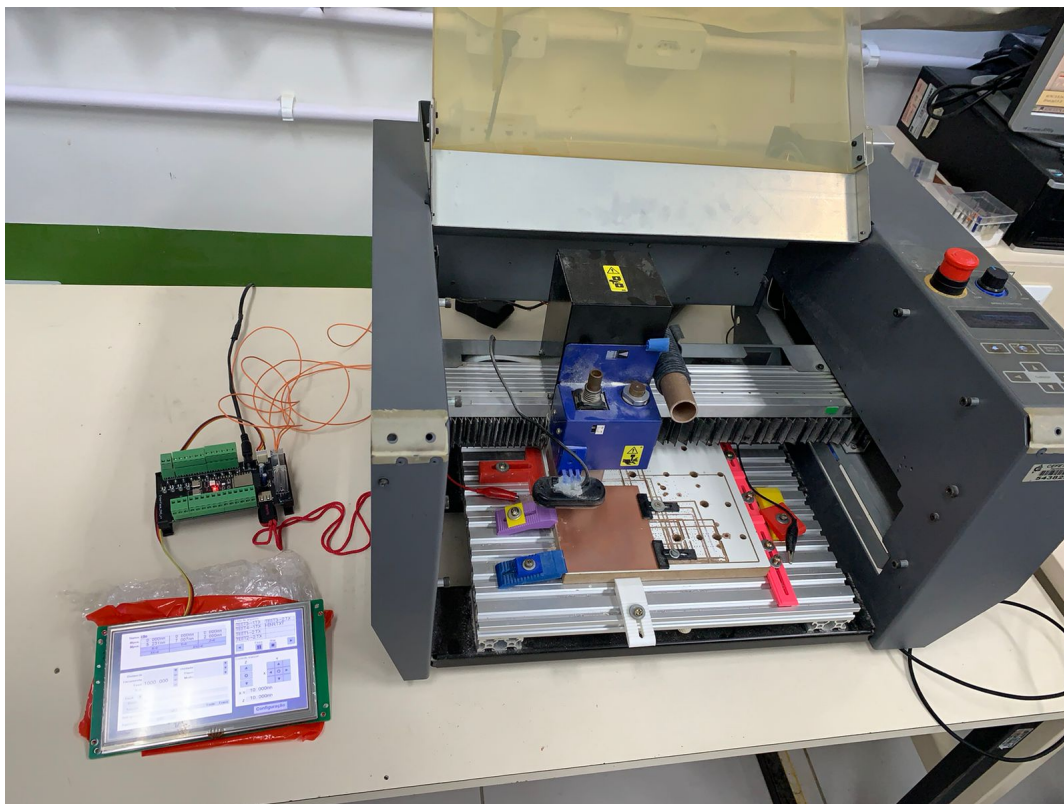
Para a realização dos testes, utilizou-se uma máquina CNC do Laboratório de Pesquisa Avançada em Eletrônica (LPAE) do IFSC que atualmente opera com o *software* GRBL embarcado em uma placa Arduino Uno sendo esta gerenciada pelo *software* bCNC rodando em uma Computador de mesa. Para adaptar essa máquina ao propósito dos testes com a interface desenvolvida, foi necessário estabelecer uma comunicação serial, seguindo a configuração apresentada na Figura 29. Com a ligação estabelecida, foram conduzidos testes iniciais, acessando a tela de configurações e verificando as configurações pré-existentes da máquina.

Figura 29 – Ligações para o teste da Interface G-code Sender



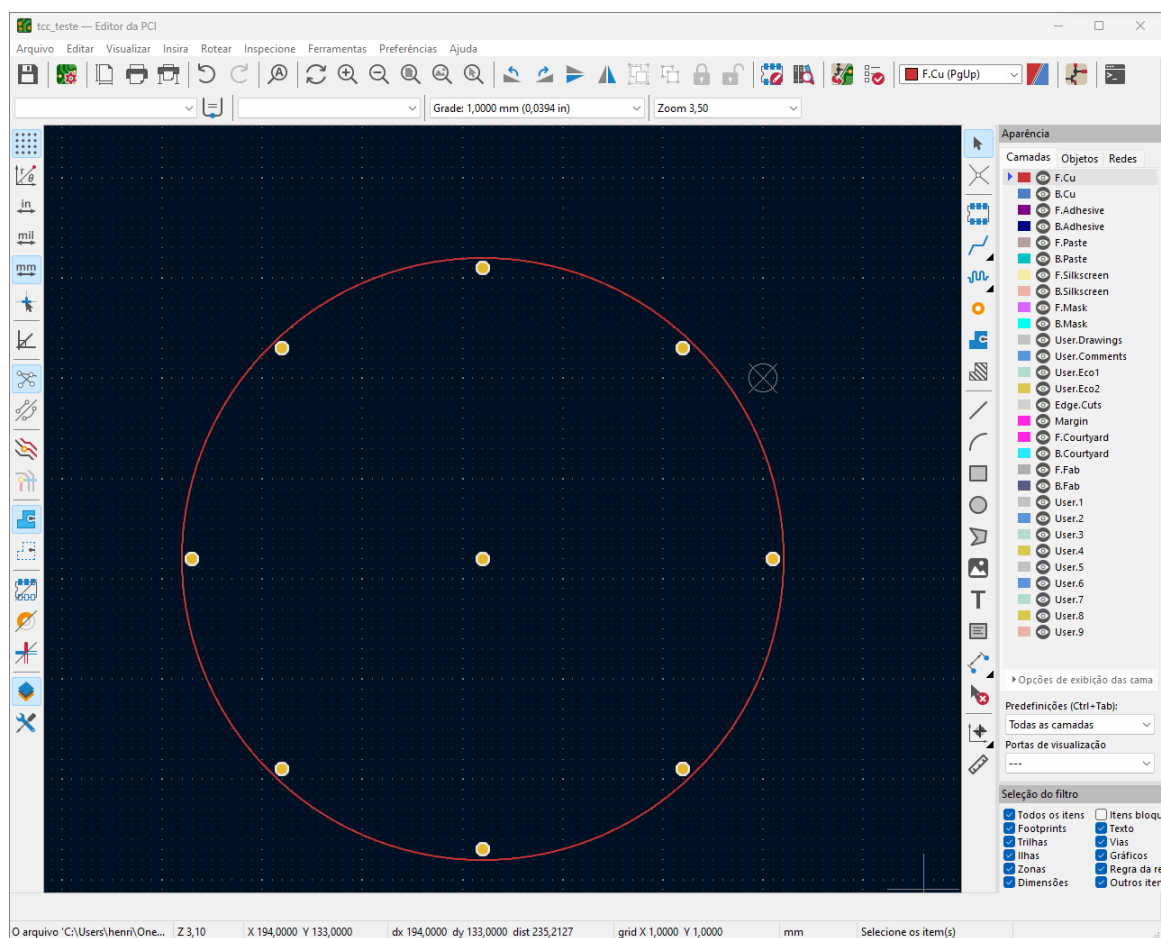
Fonte: Elaboração própria.

Posteriormente, realizou-se um teste prático ao alterar uma das configurações, neste caso, o sentido do eixo X. Para verificar a eficácia dessa alteração, solicitou-se que a máquina executasse um movimento no sentido positivo do eixo X antes e depois da mudança. Como os movimentos ocorrem em sentidos opostos um no sentido negativo do eixo e outro no sentido positivo, indicou que a configuração foi realizada com sucesso. O cenário de teste foi montado conforme demonstrado na Figura 30.

Figura 30 – Setup de testes da Interface G-code Sender

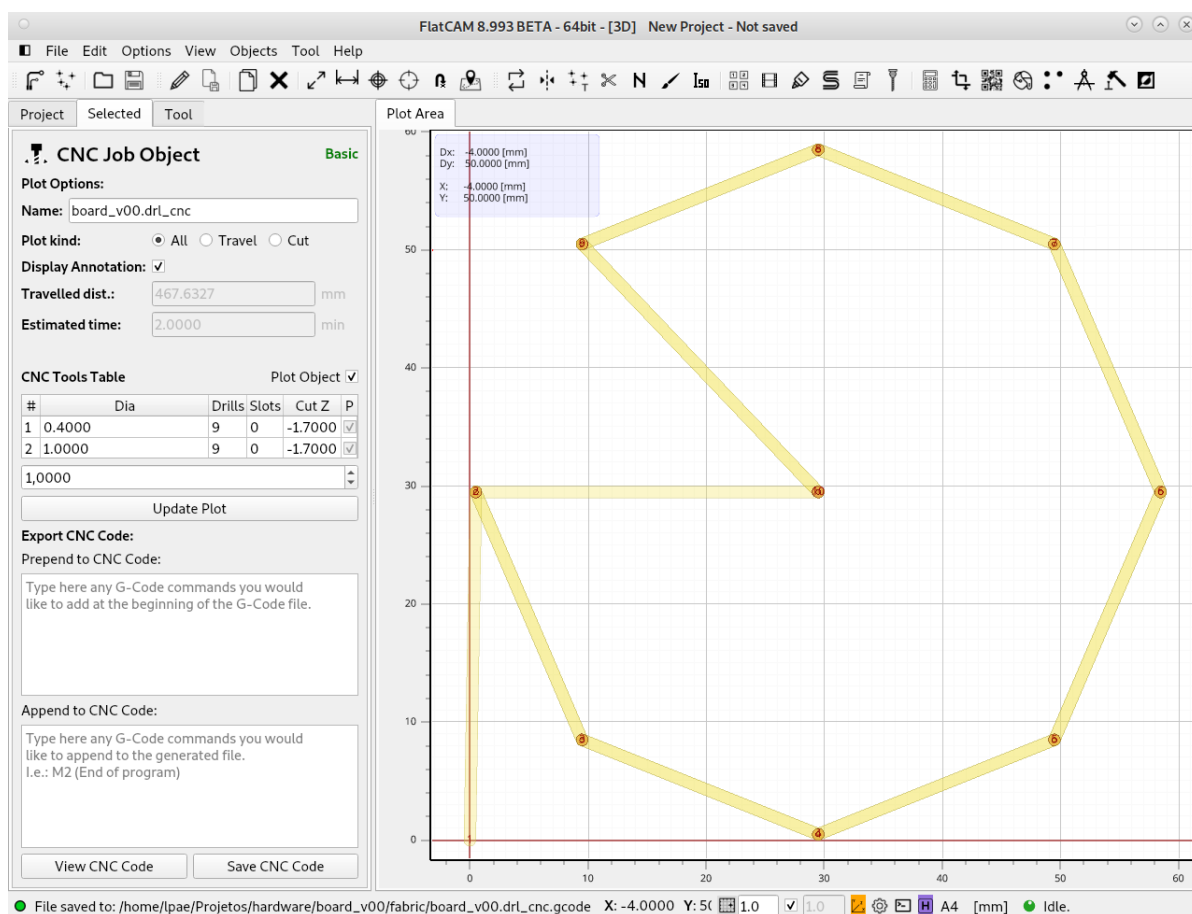
Fonte: Elaboração própria.

Com as configurações estabelecidas e ajustadas, gerou-se, por meio do *software* KiCad, um *layout* de PCB contendo um círculo de raio de 30 mm com furos ao longo de sua extensão, conforme ilustrado na Figura 31.

Figura 31 – Layout desenvolvido no Kicad feito para testes da Interface G-code Sender

Fonte: Elaboração própria.

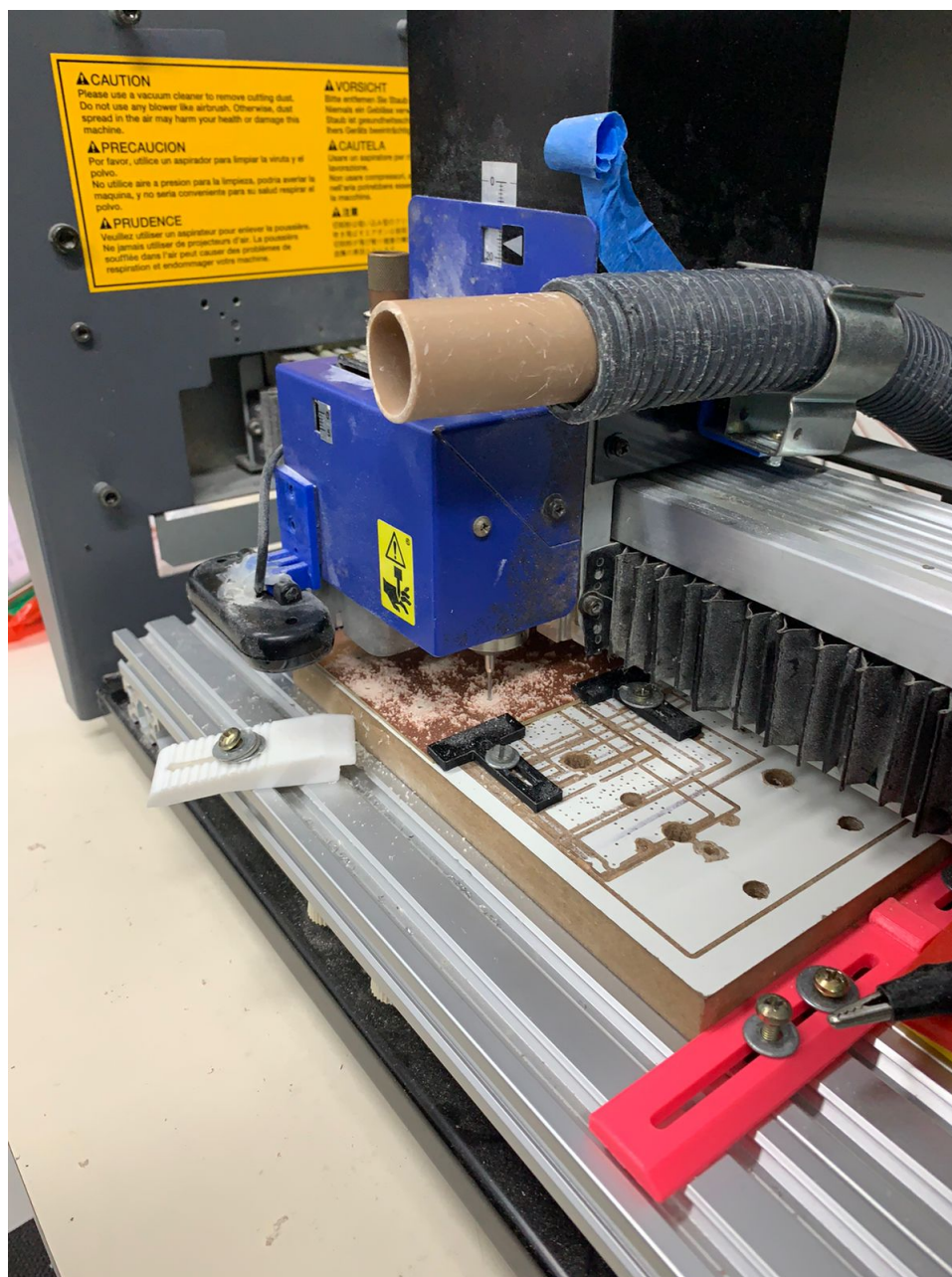
A partir desse *layout*, foram gerados os arquivos de furos Excellon no próprio KiCad. Esses arquivos foram então processados no *software* FlatCAM para a geração do arquivo G-Code, como ilustrado na Figura 32. O arquivo G-code foi transferido para uma unidade *flash* USB e inserido na interface desenvolvida. Assim, foi possível testar a mesma rotina tanto com a interface G-Code *Sender* desenvolvida quanto com o *software* bCNC, proporcionando parâmetros de comparação.

Figura 32 – Software Flat-cam com a rotina de teste gerada a partir do arquivo de furos

Fonte: Elaboração própria.

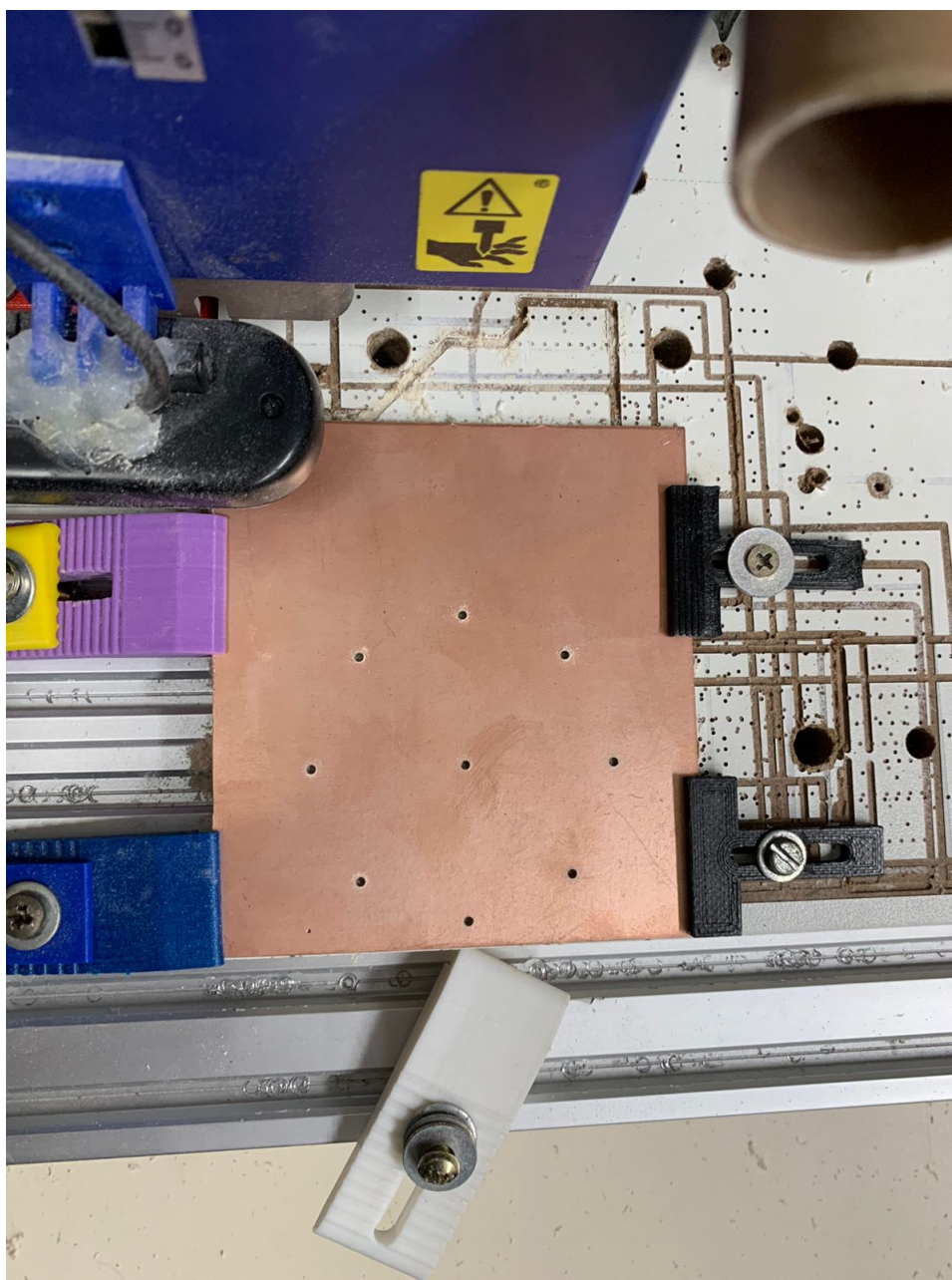
Um teste para validar o sistema foi realizado executando o arquivo G-Code dos furos tanto pelo bCNC quanto pela interface embarcada desenvolvida. Inicialmente posicionou-se uma placa de circuito impresso, conforme mostrado na Figura 30, e em seguida, definiu-se a posição zero dos três eixos. A execução pelo bCNC realizou todos os furos corretamente, e, ao final, foi solicitado o retorno para o ponto zero dos três eixos.

Posteriormente, para confirmar a funcionalidade da Interface G-code *Sender* desenvolvida, foi realizada a execução da mesma rotina, definindo a posição inicial dos eixos igual à do bCNC e executando o mesmo arquivo. A Figura 33 demonstra o processo de furação, onde a máquina CNC está realizando o processo de furação de uma PCI.

Figura 33 – Processo de furação da PCI com a Interface G-code *Sender* desenvolvida

Fonte: Elaboração própria.

Com a correta execução dos furos no mesmo local, foi possível confirmar a funcionalidade da interface desenvolvida, que obteve sucesso em executar todo o G-Code corretamente. A Figura 34 demonstra a placa ao final dos testes realizados pelos dois métodos. É possível observar que a furação feita pelos dois métodos ocorreu no mesmo local, validando o quesito de precisão de movimentação da Interface G-code *Sender* desenvolvida.

Figura 34 – Furos realizados na PCI ao final dos testes

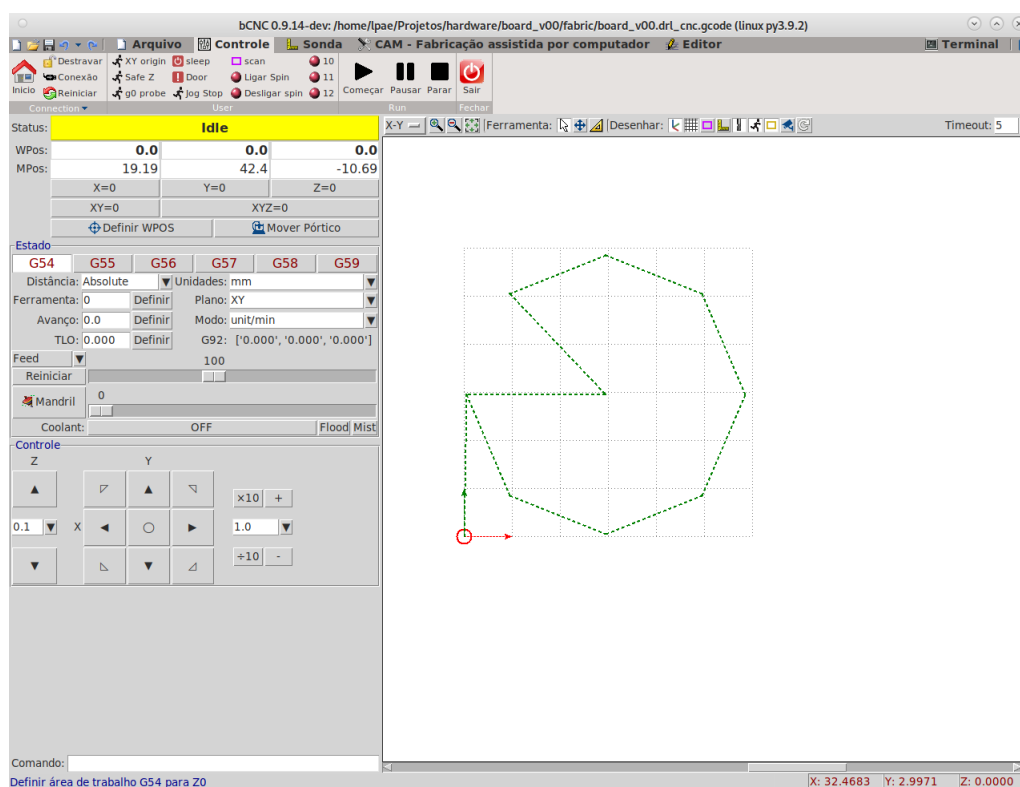
Fonte: Elaboração própria.

Comparando a interface G-code *Sender* desenvolvida com softwares de computador, observa-se que o tempo de início da rotina é ligeiramente maior, sugerindo potencial para melhorias, especialmente no processo de leitura de arquivos. Em uma análise direta do tempo de execução de um arquivo entre a interface desenvolvida e o bCNC, foi notada uma diferença no tempo de início de execução, algo na casa dos segundos, dependendo do tempo para executar o processo este pode ser considerado um tempo quase irrelevante. Como o bCNC, o arquivo é aberto previamente, este aguarda apenas o comando para dar início ao envio do G-code da rotina selecionada. Em contraste, a interface desenvolvida aguarda a seleção do arquivo e posteriormente

executa a abertura do mesmo dando início ao envio dos comandos do mesmo.

Ao término da execução, o bCNC apresenta a vantagem de possuir a funcionalidade de retorno automático para a posição zero dos três eixos. Outras vantagens notáveis do bCNC incluem a visualização prévia do trajeto completo da fresa, com o desenho exibido antecipadamente e um indicador mostrando a posição atual da fresa no desenho, oferecendo uma funcionalidade útil ilustrada na Figura 35. Além disso, o bCNC conta com sondagem de nível, crucial para processos de usinagem em PCI, onde pequenas variações no desnivelamento podem impactar significativamente o resultado final. O *software* também permite a edição do G-code para ajustes e oferece uma função gráfica para visualizar o impacto das alterações feitas. Adicionalmente, o bCNC apresenta uma função de alinhamento por câmera, uma técnica valiosa em máquinas CNC, utilizando uma câmera para capturar imagens da área de trabalho e facilitar o alinhamento automático ou assistido por imagem.

Figura 35 – Software bCNC com o G-code da rotina de teste



Fonte: Elaboração própria.

No entanto, dado que esses softwares contam com uma plataforma de maior capacidade computacional e passam por um processo de maturação e ajustes mais extenso, o projeto desenvolvido continuará evoluindo para incorporar novas funcionalidades. Os softwares G-code *Sender* projetados para computadores de mesa oferecem uma gama mais ampla de funcionalidades, especialmente no domínio da visão computacional. Isso se deve à maior capacidade de processamento disponível em

computadores de mesa em comparação com o microcontrolador utilizado neste projeto. As interfaces para *desktop* ainda mantêm uma vantagem significativa em termos de funcionalidades e capacidades para execuções detalhadas.

Por fim, para simular e avaliar o sistema em condições reais, o sistema permaneceu ativo por 48 horas, realizando interações esporádicas e executando arquivos. Durante esse período, não foram observados travamentos ou perda de funcionalidade no sistema.

4.2 Custos

Após avaliação dos resultados dessas comparações, realizou-se um levantamento de custos do *hardware*. Neste projeto, os dois itens principais são o módulo de comando da interface G-code *Sender* ou seja a PCI que embarca a tecnologia de controle da mesma e o *display* inteligente que realiza toda a parte de IHM. Na tabela 1 demonstra todos os custos dos itens usados para montar a PCI e compor o projeto.

Tabela 1 – Tabela com os custos de todas as tecnologias empregadas no projeto

Quantidade	Referencia	Valor	Valor unitário (R\$)	Total (R\$)
2	C1,C12	100 nF	0,05378	0,10756
2	C10,C11	470uF	0,15656	0,31312
1	C13	100uF	0,08205	0,08205
10	R1,R3,R5,R7 ,R9,R11,R13, R17,R20,R22	1K	0,01248	0,1248
10	R2,R4,R6,R8 ,R10,R12,R14 ,R18,R21,R23	470R	0,01248	0,1248
10	U1,U5,U6,U7 ,U8,U9,U10 ,U12,U13,U14	FOD817A	0,27558	2,7558
1	U2	ESP32S3 DEVKITC1N8R2	118,9	118,9
1	U3	TXS0104EPWR	8,34	8,34
1	U11	7805	0,48926	0,48926
10	D1,D2,D3,D4 ,D5,D6,D7, D11,D12,D13	LED VM 0805	0,08396	0,8396
4	J4, J5	Conector Borne Stlz 6 vias	0,996	3,984
1	J6	Conector Borne Stlz 3 vias	0,5036	0,5036
1	J2	Conector Borne Stlz 2 vias	0,2756	0,2756
1	J3	USB	1,15312	1,15312
2	J1,J10	Barra dourada 3 vias	0,39	0,78
1	J14	Jack J4 PCI 90º	0,50975	0,50975
1	Dissipador U11	HS 3512-30	1,41	1,41
1	Display Inteligente	P80480V70C_T07+	660	660
1	Circuito impresso		26,716	26,716
1	Fonte 5V 2.4A	KD-301C	35	35
	Total de custo			862,40906

Fonte: Elaboração própria.

O item com maior custo é o *display* inteligente. Ele incorpora tecnologia embarcada e é um produto importado com um alto valor agregado. Outro componente que pode ser alterado é a placa de desenvolvimento do ESP32 que possui um valor elevado. É possível adquirir o chip diretamente reduzindo os custos. Contudo, o custo do projeto protótipo está dentro das expectativas, apresentando potencial para se tornar

um produto viável. Alternativamente, é possível obter melhores preços para o *display* considerando tamanhos menores ou adquirindo em maior quantidade, o que resultaria em uma proposta mais competitiva no mercado.

Em comparação com o valor de um computador de mesa e seus periféricos, estimados em uma faixa de 1500 reais a 2500, o projeto proposto oferece uma relação custo-benefício mais atrativa. No entanto, é importante ressaltar que o projeto possui algumas limitações de uso, sendo os *softwares* para computadores de mesa mais versáteis principalmente quando são necessários recursos computacionais mais robustos como o nivelamento, alinhamento e outras funções mais avançadas.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi concebida uma interface G-code *Sender* implementada em um microcontrolador ESP32-S3 com o propósito de simplificar e otimizar a operação de máquinas CNC. Essa abordagem visa melhorar o envio de rotinas G-code para controladoras que utilizam o *software* GRBL. Após a definição das tecnologias que comporiam o projeto, todos os três módulos planejados, incluindo os módulos de IHM, comunicação com o *software* GRBL e módulo USB, foram implementados.

O desenvolvimento da interface G-Code *Sender* para máquinas CNC, com base no *firmware* GRBL, alcançou os objetivos propostos. Uma pesquisa abrangente sobre o GRBL, seus requisitos de *hardware* e protocolos de comunicação, forneceu a base necessária para o projeto. Utilizando o microcontrolador ESP32 com o *framework* ESP-IDF e o sistema operacional FreeRTOS, a interface foi desenvolvida, incluindo funções essenciais para implementar e tratar protocolos de comunicação eficientes entre a controladora e o sistema GRBL, bem como os protocolos do G-Code. O controle individual dos três eixos da máquina, as configurações do GRBL e o sistema de funções para processar e enviar comandos G-Code foram implementados com sucesso. A viabilidade e eficácia da controladora offline foram avaliadas, comparando os resultados obtidos na operação da CNC com métodos tradicionais que utilizam computadores. Além disso, a simplicidade e praticidade da interface homem-máquina implementada no *display touchscreen* foram validadas, contribuindo para a disseminação de uma solução tecnológica que oferece autonomia, eficiência e redução de custos no controle de máquinas CNC, aproveitando as vantagens do *firmware* GRBL e do microcontrolador ESP32.

Ao longo do desenvolvimento, deparamo-nos com desafios, principalmente relacionados às peculiaridades da comunicação com o GRBL. A adequação do protocolo de comunicação demandou diversos testes, visto que o formato dos comandos varia dependendo do tipo enviado. Por outro lado, a implementação da parte do *display* transcorreu de forma mais tranquila, uma vez que a biblioteca fornecida e o software gráfico abstraem a complexidade do protocolo de comunicação, cabendo ao desenvolvedor apenas o tratamento das informações recebidas e enviadas. Quanto ao módulo USB, o exemplo utilizado como base já contemplava as funções fundamentais para a aplicação, sendo necessário principalmente adaptar as operações com arquivos e incorporar as funcionalidades específicas do projeto. Após a conclusão de cada módulo, testes individuais foram realizados para validar o funcionamento, possibilitando a subsequente integração e validação do sistema como um todo. Um teste crucial consistiu na execução de um arquivo contendo uma rotina G-code que realizava uma série de furos em um círculo de raio de 30 mm. A comparação dos resultados desse teste com o bCNC evidenciou a necessidade de melhorias e funcionalidades adicionais

para aprimorar a qualidade da usinagem. Mas comprovou a eficácia de todas as funcionalidades propostas no início do trabalho e trazendo margem para novas melhorias e próximos objetivos para o projeto.

Este trabalho apresenta uma nova possibilidade de interface G-code *Sender* que pode proporcionar uma redução de custos em comparação com outras alternativas disponíveis, especialmente para empresas que buscam desenvolver equipamentos em maior escala. Além disso, oferece uma oportunidade para entusiastas desenvolverem soluções semelhantes, eliminando a dependência exclusiva de computadores de mesa e permitindo o uso de máquinas CNC com o GRBL.

5.1 Sugestões para Trabalhos Futuros

Com a base do projeto estabelecida, há espaço para incorporar novas funcionalidades que aprimorem a qualidade e a funcionalidade da interface. Recursos como nivelamento automático da ferramenta, visualização prévia da usinagem, acompanhamento em tempo real da posição da ferramenta e um sistema de *probe* são algumas das possíveis extensões. Essas e outras melhorias são propostas para futuros trabalhos com o intuito de posicionar essa interface como concorrente em qualidade e confiabilidade com outras G-code *Senders* disponíveis no mercado atual.

O projeto será continuado com o objetivo de refinamento e transformação em um produto completo, possivelmente através de parcerias com empresas investidoras interessadas em introduzir um produto inovador tanto para uso interno quanto como uma potencial oferta para outras indústrias.

REFERÊNCIAS

- AJANOVIC, J. **PCI Express 3.0 Overview**. 2009. Disponível em: https://old.hotchips.org/wp-content/uploads/hc_archives/hc21/1_sun/HC21.23.1.SystemInterconnectTutorial-Epub/HC21.23.131.Ajanovic-Intel-PCleGen3.pdf. 26
- BCNC. **bcNC**. 2022. Disponível em: <https://github.com/vlachoudis/bCNC/tree/master>. Acesso em: 08 nov 2023. 37
- BENGLER. **GRBL**. 2014. Disponível em: <http://bengler.no/grbl>. Acesso em: 2023-11-10. 15
- CARAM, J. P. **FlatCAM: 2D Computer-Aided PCB Manufacturing**. 2023. Disponível em: <https://github.com/Denvi/FlatCAM>. Acesso em: 2023-10-29. 25
- CARSTENS, S. F.; CARSTENS, T. A. **Projeto, Construção e Funcionamento de uma Máquina CNC com Plataforma Livre - Arduino**. 2015. Disponível em: <http://joinville.ifsc.edu.br/~bibliotecajoi/arquivos/tcc/mecind/156169.pdf>. Acesso em: 2023-11-09. 20
- CHAMPLAIN, J.; WOODCOCK, J. **Understanding Gcode Commands**. Image Engraving, 2015. [Online]. Disponível em: <https://www.picengrave.com/Pic%20Programs%20Page/PDF%20Files/misc/Understanding%20Gcode.pdf>. 23, 24
- COREL CORPORATION. **CorelDRAW**. 2023. Disponível em: <https://www.coreldraw.com/br/>. Acesso em: 2023-11-14. 36
- DAKOL. **O que é Comunicação Serial?** 2023. Disponível em: [Online]. Disponível em: <https://www.dakol.com.br/comunicacao-serial/#:~:text=O%20que%20%C3%A9%20Comunica%C3%A7%C3%A3o%20serial,canal%20de%20comunica%C3%A7%C3%A3o%20ou%20barramento>. 25
- ESPRESSIF SYSTEM. **UART Asynchronous Example with Separate Receive and Transfer Tasks**. 2023. Disponível em: https://github.com/espressif/esp-idf/tree/master/examples/peripherals/uart/uart_async_rxtxtasks. Acesso em: 2023-11-10. 55
- ESPRESSIF SYSTEM. **USB Mass Storage Class example**. 2023. Disponível em: <https://github.com/espressif/esp-idf/tree/master/examples/peripherals/usb/host/msc>. Acesso em: 2023-11-10. 65, 68, 69, 70
- ESPRESSIF SYSTEMS. **ESP-IDF Programming Guide**. 2023. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/index.html>. Acesso em: 08 nov 2023. 15, 27
- ESPRESSIF SYSTEMS. **ESP32-S3-DevKitC-1 v1.1**. 2023. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1.html>. Acesso em: 08 nov 2023. 35, 37, 38
- ESPRESSIF SYSTEMS. **ESP32 Technical Reference Manual**. 2023. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf. Acesso em: 08 nov 2023. 15, 26

- FREERTOS. **FreeRTOS**. 2023. Disponível em: <https://www.freertos.org/>. Acesso em: 2023-10-29. 28
- GARCIA, M. S. **Um Overview sobre Interfaces de Discos Rígidos: ATA, SATA, SCSI, FC e SAS**. 2009. Disponível em: <https://www.ic.unicamp.br/~ducatte/mo401/1s2009/T2/089057-t2.pdf>. 26
- GENG, H. (Ed.). **Manufacturing Engineering Handbook**. 2nd. ed. New York: McGraw-Hill Education, 2004. 20
- GIL, A. C. **Como elaborar projetos de pesquisa. 4. ed.** [S.l.]: [S.l.]: Editora Atlas S.A, 2002. 41 p. 34
- GRBL. **GRBL**. 2021. Disponível em: <https://github.com/gnea/grbl>. Acesso em: 08 nov 2023. 15, 22, 23, 63
- GRBL. **Grbl v1.1 Interface**. 2022. Disponível em: <https://github.com/gnea/grbl/wiki/Grbl-v1.1-Interface>. Acesso em: 08 nov 2023. 60
- ITEAD. **ITEAD**. 2023. Disponível em: <https://itead.cc/>. Acesso em: 2023-11-10. 30
- KRAMER, T.; PROCTOR, F.; MESSINA, E. **The NIST RS274NGC Interpreter - Version 3**. NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD, 2000. Disponível em: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=823374. 15
- MELLO, M. **Display inteligente: o que é, função, tipos e vantagens**. 2023. Disponível em: <https://www.makerhero.com/blog/display-inteligente/>. Acesso em: 2023-11-10. 29
- ONSEMI. **FOD817 - 4-Pin DIP Phototransistor Optocouplers**. 2019. Disponível em: https://br.mouser.com/datasheet/2/308/1/FOD814_D-2313902.pdf. Acesso em: 2023-11-10. 40
- PEÑA, E.; LEGASPI, M. G. **Uart: A hardware communication protocol understanding universal asynchronous receiver/transmitter**. *Visit Analog*, v. 54, n. 4, 2020. 26
- Sainsmart. **GRBL V1.1 Quick Reference**. 2022. Disponível em: <https://www.sainsmart.com/blogs/news/grbl-v1-1-quick-reference>. Acesso em: 2023-11-09. 61
- SEAMES, W. S. **Computer Numerical Control: Concepts and Programming**. Stanford: Cengage Learning, 2001. 19
- SILVEIRA, C. B. **Servo Motor: Veja como Funciona e Quais os Tipos**. 2016. Disponível em: <https://www.citisystems.com.br/servo-motor/>. Acesso em: 2023-11-10. 21, 22
- SIMPLIFY3D. **Simplify3D**. 2023. Disponível em: <https://www.simplify3d.com/>. Acesso em: 2023-10-29. 25
- STMICROELECTRONICS. **Datasheet - L78 - Positive voltage regulator ICs**. 2018. Disponível em: <https://br.mouser.com/datasheet/2/389/l78-1849632.pdf>. Acesso em: 2023-11-10. 39

TOOL, A. S. of; ENGINEERS, M.; WILSON, F. W. **Numerical Control in Manufacturing**. New York: McGraw-Hill, 1963. 15

ULTIMAKER. **Ultimaker Cura**. 2023. Disponível em. Disponível em: <https://ultimaker.com/software/ultimaker-cura/>. Acesso em: 2023-10-29. 25

VIKTOR VISION. **UnicView AD v1.6.0 Development Guide 1.0**. 2017. Disponível em. Disponível em: <https://proculustechologies.com/software-resources/UnicView/UnicView%20AD/UnicView%20AD%20Development%20Guide.pdf>. Acesso em: 2023-10-29. 32, 51, 52

VIKTOR VISION. **UnicView AD v1.6.0 User Guide 1.0**. 2017. Disponível em. Disponível em: <https://proculustechologies.com/software-resources/UnicView/UnicView%20AD/UnicView%20AD%20User%20Guide.pdf>. Acesso em: 2023-10-29. 32, 50, 51

VIKTOR VISION. **Displays Proculus com microcontrolador PIC**. 2020. Disponível em. Disponível em: <https://victorvision.com.br/blog/displays-proculus-com-microcontrolador-pic/>. Acesso em: 2023-10-29. 50, 51, 57

VIKTOR VISION. **UnicView AD: Software de Interfaces Gráficas**. 2023. Disponível em. Disponível em: <https://victorvision.com.br/solucoes/unicview-ad/>. Acesso em: 2023-10-29. 31, 51

VIKTOR VISION. **Viktor Vision**. 2023. Disponível em. Disponível em: <https://victorvision.com.br/>. Acesso em: 2023-10-29. 29