UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DE CIÊNCIAS EXATAS E ENGENHARIAS
ENGENHARIA ELÉTRICA

GUILHERME RODRIGUES

SISTEMA DE COMUNICAÇÃO DE DISPOSITIVO USB STORAGE COM DIRETÓRIO REMOTO

CAXIAS DO SUL
2022

Guilherme Rodrigues

SISTEMA DE COMUNICAÇÃO DE DISPOSITIVO USB STORAGE COM DIRETÓRIO REMOTO

Trabalho de conclusão de curso apresentado à Área de Ciências Exatas e Engenharias da Universidade de Caxias do Sul como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador:
Prof. Me. Ricardo Leal Costi

CAXIAS DO SUL
2022

Guilherme Rodrigues

SISTEMA DE COMUNICAÇÃO DE DISPOSITIVO USB STORAGE COM DIRETÓRIO REMOTO

Trabalho de conclusão de curso apresentado à Área de Ciências Exatas e Engenharias da Universidade de Caxias do Sul como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador:
Prof. Me. Ricardo Leal Costi

Aprovado em 08 / 07 / 2022

Banca Examinadora

_____
Prof. Me. Ricardo Leal Costi (orientador)
Universidade de Caxias do Sul - UCS

_____
Profa. Me. Andréa Cantarelli Morales
Universidade de Caxias do Sul - UCS

_____
Prof. Me. Bruno Fensterseifer Dias
Universidade de Caxias do Sul - UCS

RESUMO

O presente trabalho propôs o desenvolvimento de um dispositivo para suprir uma demanda de acessar e modificar arquivos que são exibidos em um torno CNC utilizando um computador sem ligação física ao torno. O dispositivo implementado teve como finalidade realizar o elo entre a máquina do usuário que deseja acessar os arquivos e a máquina que os exibe, criando uma rede específica para esta conexão. Foram revistos então alguns conceitos importantes para a definição do projeto deste dispositivo, que opera se comunicando fisicamente a um dos lados e via comunicação sem fio ao outro. Estes conceitos tratam de tecnologias atuais, como o uso da microeletrônica e as comunicações entre dispositivos eletrônicos e as formas para que isto ocorra. A partir disto, foi possível desenvolver um *firmware* e um protótipo para a aplicação, que utilizou uma placa STM32F769 DISCOVERY e um módulo ESP32 para implementar funções de cartão SD, USB, Wi-Fi e servidor FTP.

Palavras-chave: Dispositivo. Comunicação. USB. Wireless. Microcontroladores.

ABSTRACT

The present work proposed the development of a device to supply a demand to access and modify files that are displayed on a CNC lathe using a computer without physical connection to the lathe. The device implemented was intended to create a link between the user's machine that wants to access the files and the machine that displays them, creating a specific network for this connection. We then reviewed some important concepts for defining the design of this device, which operates by physically communicating to one side and wirelessly to the other. These concepts deal with current technologies, such as the use of microelectronics and communications between electronic devices and the ways in which this occurs. From this, it was possible to develop a *firmware* and a prototype for the application, which used a STM32F769 DISCOVERY board and an ESP32 module to implement SD, USB, Wi-Fi and FTP server functions.

Keywords: Device. Communication. USB. Wireless. Micro-controllers.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, que não deixaram faltar apoio e paciência, mas também aos meus familiares e amigos que me deram suporte e incentivo durante a realização deste trabalho e souberam compreender o comprometimento e a dedicação que isto exigiu.

Ao meu professor orientador por toda ajuda, atenção e paciência ao longo do desenvolvimento deste projeto.

A todos meus colegas da UCS e da empresa onde trabalho pelos auxílios fornecidos.

Aos amigos e colegas que se dispuseram a ler meu rascunho e torná-lo um trabalho final.

Aos professores avaliadores da minha banca pela disponibilidade e atenção desprendidas na avaliação deste trabalho.

A todos os professores que fizeram a diferença na minha trajetória desde a escola até aqui, tornando possível concluir este trabalho e encerrar um ciclo bastante importante.

"*Quando achamos a matemática e a física teórica muito difíceis, voltamo-nos para o misticismo.*"
Stephen Hawking

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

bps	*bits por segundo*
dBm	*decibel metro*
GHz	*Giga-Hertz*
GND	*Ground (Terra)*
kbps	*kilobits por segundo*
kB	*kilo-Byte*
m	*metro*
mA	*mili Ampère*
MB	*Mega-Byte*
Mbps	*Megabits por segundo*
V	*Volt*
Vdc	*Volt (Direct Current)*

LISTA DE SIGLAS

ARM	Advanced RISC Machine
CN	Controle Numérico
CNC	Controle Numérico Computadorizado
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DNC	Direct Numerical Control
DSSS	Direct Sequence Spread Spectrum
FAT	File Allocation Table
FHSS	Frequency-Hopping Spread Spectrum
FTP	File Transfer Protocol
I2C	Inter Integrated Circuit
IDE	Integrated Development Environment
IEEE	Institute Of Electrical And Electronics Engineers
IP	Internet Protocol
LAN	Local Area Network
MAC	Media Access Control
MSC	Mass Storage Class
MCU	Microcontroller Unit
NRZI	Non-Return to Zero Inverted
OFDM	Orthogonal Frequency Division Multiplexing
OSI	Open System Interconnection
OTG	On The Go
PC	Personal Computer
PDA	Personal Digital Assistant
RISC	Reduced Instruction Set Computer
RS	Recommended Standard
SD	Secure Digital
SPI	Serial Peripheral Interface
SRAM	Static Random Acess Memory
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous Asynchronous Receiver Transmitter
USB	Universal Serial Bus

UWB	Ultra-Wide-Band
WPAN	Wireless Personal Area Networks

SUMÁRIO

1 INTRODUÇÃO

A confecção de peças usinadas pode ser realizada através do uso de uma máquina CNC (Controle Numérico Computadorizado), que executa os passos de comandos arranjados em um programa CNC. Este programa é escrito em código G e indica o que a máquina deve fazer, podendo ser desenvolvido na própria máquina CNC ou em um computador, utilizando um software para fazê-lo (SANTOS, 2017). Este programa pode ser transportado por uma porta serial ou por um *flash drive* USB (dependendo da máquina) para atualizações. Como cita Costa (2018), é comum existir uma comunicação cabeada entre computador e a máquina utilizando, por exemplo, o padrão RS-232 (a ser estudado neste trabalho).

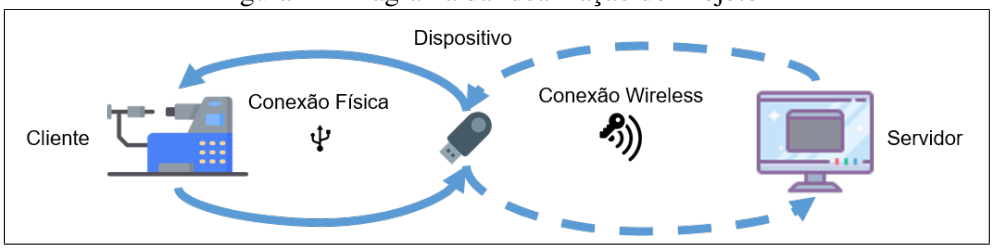
O fato de ser possível realizar atualizações somente por meios físicos acaba gerando alguns problemas como: arquivos desatualizados, incompatibilidade no programa entre máquinas, necessidade de acesso físico à máquina (tempo do operador), entre outros. Como diz Silva et al. (2006), é interessante trazer novas ferramentas para controle e monitoramento *online* das máquinas CNC tradicionais. A importância de sistemas para realizar estas funções vem tendo cada vez mais relevância tanto no meio acadêmico quanto em soluções industriais.

Tendo em vista a relevância das soluções referidas anteriormente, torna-se interessante analisar que as comunicações sem fio estão cada vez mais presentes nos meios de conexão do dia a dia, como Wi-Fi e Bluetooth. Estes tipos de comunicações proporcionam maior mobilidade e portabilidade aos usuários em relação às redes cabeadas. São mais ajustáveis também em relação a estas últimas e de rápida implantação, com menores mudanças em infraestrutura. (SILVA CAETANO, 2021). As características da conectividade sem fio vão de encontro ao interesse em criar soluções remotas para comunicação com as máquinas CNC.

1.1 JUSTIFICATIVA

O trabalho buscou atender à demanda de uma empresa de acessar e ser capaz de modificar determinados arquivos através de uma máquina (Computador - Servidor de Arquivos). Estes mesmos arquivos devem ser exibidos em outra máquina (Torno CNC - Cliente) para que esta última tenha acesso de leitura. Como nem todas as máquinas possuem acesso à rede (*Ethernet*), foi proposta a criação de uma rede sem fio específica como solução para o problema, com o uso de um dispositivo que faça o elo entre ambas. Este dispositivo faz o acesso à rede sem fio e acessa os arquivos do Servidor através dela, devendo estar ligado fisicamente ao Cliente para fornecer acesso ao mesmo a estes arquivos. A Figura 1 ilustra a conceituação inicial do sistema a que foi desenvolvido:

Figura 1 – Diagrama da Idealização do Projeto



Fonte: O autor (2022)

O dispositivo permite que o Cliente acesse os arquivos do Servidor, de forma que a exibição aconteça como se estivessem na própria máquina do Cliente (como um *Flash Drive*, por exemplo). Isto possibilita que sejam feitos backups dos programas e atualizações das demais máquinas com um mesmo programa.

1.2 Objetivo Geral

Desenvolver um *hardware* com comunicação física e sem fio, capaz de transmitir arquivos de programa para uma máquina CNC a partir de um servidor.

1.3 Objetivos Específicos

As ações planejadas para o cumprimento do Objetivo Geral são:

- Desenvolver código de *firmware* para a aplicação;
- Implementar *firmware* em protótipo de *hardware* em uma solução que atenda à aplicação;
- Realizar testes de validação da solução implementada.

1.4 Limitações do Trabalho

Não será abordado o conteúdo dos programas da máquina, somente preocupando-se com a transmissão do mesmo em formato de texto. Também não será realizado projeto de *hardware* para esta aplicação, utilizando somente placas/kits de desenvolvimento para criar um protótipo.

2 FUNDAMENTAÇÃO TEÓRICA

Para o desenvolvimento da solução proposta neste trabalho, é necessário estudar conceitos importantes no desenvolvimento do projeto para ir de encontro à resolução do problema.

2.1 CNC

Controle Numérico (CN) trata-se de uma forma específica de controlar os movimentos de uma máquina, sendo que o posicionamento é a principal variável a ser controlada. Isto se dá pela interpretação direta de instruções codificadas na forma de números e letras, que indicam o que a máquina deve fazer. (ROCHA JUNIOR, 2013)

Segundo Rocha Junior (2013), são três os componentes principais de um sistema de controle numérico: sequência de comandos, unidade de controle e planta eletromecânica.

A sequência de comandos consiste em instruções codificadas que irão ditar o comportamento da máquina em relação à peça que será usinada. A codificação deve ser compatível com a unidade de controle (linguagens como código G ou código M), que decodifica e interpreta a sequência de comandos e define o posicionamento do atuador. Devido à utilização de microprocessadores na unidade de controle, o processo passou a ser computadorizado, ficando conhecido como controle numérico computadorizado (CNC).

De certa forma, o CNC é uma evolução do Controle Numérico e seu principal objetivo é reduzir o tempo de confecção da peça, aumentando a capacidade operacional da máquina. Isto também acaba tornando o processo de usinagem algo mais amigável ao operador. (ROCHA JUNIOR, 2013)

Pode ainda ser citado um outro conceito (menos difundido) que é o controle numérico direto (DNC), que mais tarde foi redefinido como controle numérico distribuído. Enquanto no CNC cada máquina é equipada com o seu próprio processador, no DNC um computador central controla várias máquinas equipadas com computador. Este sistema permite mais capacidade de memória e processamento e oferece flexibilidade. (MARCICANO, 2021)

A elaboração de um programa CNC envolve uma série de informações relacionadas com a geometria da peça, com o tipo de máquina, com as ferramentas disponíveis e ainda todos os fundamentos de usinagem necessários para obtenção do produto com as características desejadas. Normalmente, os comandos utilizam quatro pontos de referência para a posição de ferramentas, denominados: ponto zero da máquina, ponto zero da peça, ponto de referência da ferramenta e ponto de referência da máquina.

Utilizando como exemplo um torno CNC, é utilizada a linguagem de programação G, que é basicamente composta por comandos e valores:

- Velocidade de avanço ou avanço da ferramenta: F;
- Tipo de função *Guidance*: G;

- Tipo de função *Miscelaneous*: M;
- Medidas ou coordenadas: X, Y, Z, I, J, K, R, H, D;
- Número do programa CNC O;
- Rotação do eixo-árvore S;
- Número de ferramenta enderaçada no magazine: T.

Em relação às letras, pode-se indicar:

- D - Diâmetro da ferramenta
- F - Velocidade de avanço da ferramenta
- G - Códigos G (funções “Guidance” de operação)
- H - Comprimento da ferramenta
- I - Centro de arco no eixo x
- J - Centro de arco no eixo y
- K - Centro de arco no eixo z
- M - Códigos M (funções “Miscelaneous” de preparação)
- N - Numeração das linhas do programa (bloco)
- O - Número do programa
- R - Raio de arcos de circunferência
- S - Rotação do eixo-árvore
- T - Seleção de ferramenta
- X - Eixo linear x
- Y - Eixo linear y
- Z - Eixo linear z

Pode-se ter como exemplo de programa CNC a seguinte linha, na qual o comando G01 especifica interpolação linear utilizada para deslocar a ferramenta em trabalho de usinagem da posição até a posição desejada em linha reta:

G01 X/U Z/W A C R F S M

2.2 COMUNICAÇÃO DE DADOS

Para que exista a troca de informação entre dispositivos, é necessário existir uma comunicação. Para que isto ocorra, uma série de pré-requisitos deve ser atendida, tais como velocidade de transmissão/recebimento, sincronização, formatação de bits de dados, entre outros. É necessário também um canal para tal comunicação, sendo o caminho pelo qual a informação irá trafegar. Este caminho pode ser um fio ou então um rádio ou outra fonte de energia radiante (CANZIAN, 2009). Quanto aos tipos de canais:

- *Simplex*: a direção de transmissão possui um único sentido (unilateral);
- *Half-Duplex*: a direção pode ser revertida, ou seja, as mensagens podem ser revertidas, porém, neste caso, nunca ao mesmo tempo;
- *Full-Duplex*: permite a troca de mensagens simultaneamente em ambas as direções.

Na comunicação digital, a representação da informação se dá por meio de bits de dados, que em conjunto (de agrupamento definido pelo protocolo ou aplicação) formam um *byte* e um agrupamento organizado de *bytes* forma um *frame* (mensagem)(CANZIAN, 2009).

2.2.1 Comunicação Serial

A comunicação serial se dá com transferência não simultânea dos bits que fazem parte do *byte*, transmitindo um bit após o outro (SILVEIRA, 1991). Destacam-se os padrões RS-232, RS-485, SPI, I2C e USB.

À taxa de transferência, que é a velocidade com que os dados são enviados em um canal (medido em transições elétricas por segundo), dá-se o nome de *baud rate*. Uma taxa de 9600bps corresponde a uma transferência de 9600 dados por segundo, por exemplo. (CANZIAN, 2009)

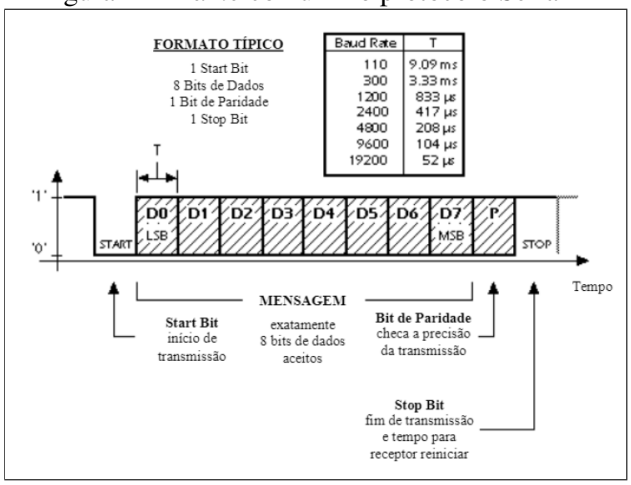
A respeito da transmissão, ela pode ainda ser caracterizada como síncrona ou assíncrona. Normalmente, os dados serializados não são enviados de maneira uniforme através de um canal, mas sim pacotes regulares de informação são enviados seguidos de uma pausa. O receptor deve saber o momento adequado para ler os bits do canal, sabendo quando o pacote começa e o tempo decorrido entre bits. Quando estes itens são atendidos, o receptor está então sincronizado com o transmissor. Falhas na sincronização implicam perda de dados. (CANZIAN, 2009)

Na transmissão síncrona, são utilizados canais separados para transmissão de dados e informação de tempo (pulso de *clock*). De acordo com os pulsos recebidos no canal de tempo, lê os dados e armazena os valores dos bits. Já na transmissão assíncrona, o transmissor e o receptor devem ser previamente configurados para que a comunicação ocorra adequadamente. São utilizados sinais de *clock* interno para cada um, de forma que sejam muito similares entre si. (CANZIAN, 2009)

Nos protocolos de comunicação serial mais comuns, são utilizados pacotes de 10 ou 11 bits, dos quais 8 formam a mensagem. O canal normalmente está em nível lógico alto (1) e recebe

um nível baixo (0) para indicar o *start bit* (início da mensagem). A seguir vêm os 8 bits de dados no *baud rate* especificado, um bit de paridade (opcional) e um *stop bit*. Isto pode ser melhor entendido com a ilustração da Figura 2:

Figura 2 – *Frame* comum no protocolo Serial



Fonte: (CANZIAN, 2009)

Na Figura 2, é ilustrada a visualização do sinal em uma mensagem serial como um gráfico da tensão para cada informação. O sinal mostra os bits como níveis digitais altos para 1 e baixos para 0, sendo a primeira informação um *start bit* de nível baixo, seguido de 7 bits de dados (do menos significativo, ou seja, de menor valor numérico à esquerda para o mais significativo à direita) e de um bit de paridade que pode possuir tanto um nível lógico quanto o outro, e por último de um *stop bit* de nível alto. O tempo T especificado para o pulso de nível lógico digital para cada bit é definido de acordo com o *baud rate* escolhido.

2.2.1.1 RS-232

Dentre os padrões de comunicação serial, destaca-se o RS-232 (RS se refere a *Recommended Standard*). Este protocolo foi desenvolvido com intuito de promover a padronização de uma interface comum para comunicação de dados entre equipamento. O padrão RS-232 especifica as tensões, temporizações e funções dos sinais, um protocolo para troca de informações e as conexões mecânicas. A respeito do *baud rate* dessa comunicação, os valores comuns são de 2400, 4800 e 9600 bps. (CANZIAN, 2009).

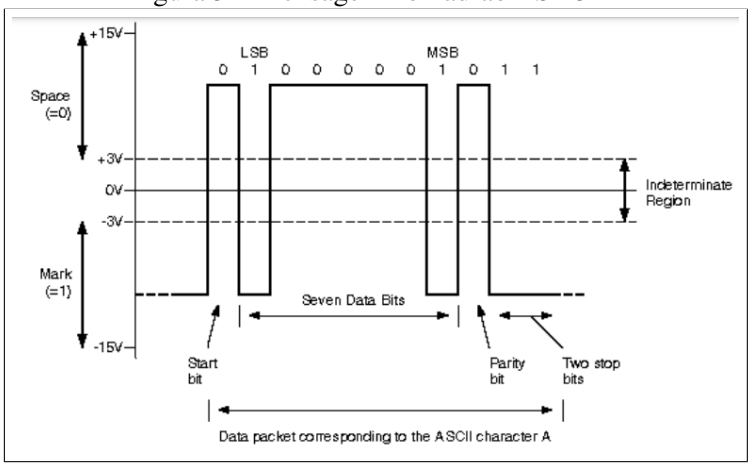
Este padrão diz respeito a uma conexão *Full-Duplex* assíncrona ponto-a-ponto. A base de sua implementação se dá através de três terminais (transmissor, receptor e GND), onde o nível lógico 1 tem o sinal negativo e o nível lógico 0 tem o sinal positivo. Estes sinais são aplicáveis tanto ao terminal transmissor quanto ao receptor, que alternam o nível de tensão para informar dados. A distância máxima (com taxas de transmissão baixas) é de aproximadamente 15 metros, mas para distâncias menores (aproximadamente 1,5m) pode-se atingir taxas de transmissão de

256000 bps (*bits per second*) (CANICEIRO, 2018). Os níveis lógicos são representados por tensões:

- -3V a -15V como Marca = 1 = OFF;
- +3V a +15V como Espaço = 0 = ON;
- Tensões entre -3 V e +3 V são indefinidas.

A Figura 3 ilustra um exemplo de mensagem no padrão RS-232 com os níveis de tensão:

Figura 3 – Mensagem no Padrão RS-232



Fonte: (CANICEIRO, 2018)

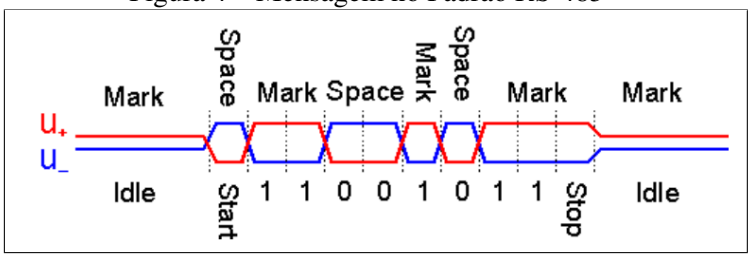
A Figura 3 remete à Figura anterior, porém com sinal em acordo com o padrão RS-232. São exibidos valores de tensão positivos (entre 3 e 15V) para nível lógico 0 e negativos para nível lógico 1 ((entre -3 e -15V). Há um *start bit* de nível lógico 0 e em seguida há os 7 bits de dados, onde é passado como informação para exemplo o caractere 'A', que na tabela ASCII (*American Standard Code for Information Interchange* ou Código Padrão Americano para o Intercâmbio de Informação) tem o valor 41 hexadecimal, que em número binário representa 1000001. Os bits de dados também vão de LSB (*Least Significant Byte* ou Bit menos significativo) à esquerda para MSB (Most Significant Bit ou Bit mais significativo) à direita. Por último há um bit de paridade seguido de dois *stop bits*.

2.2.1.2 RS-485

O RS-485 também se trata de um protocolo assíncrono. Este suporta redes de comunicação multiponto, podendo ter até 32 dispositivos conectados ao mesmo barramento, usando uma linha diferencial equilibrada em par trançado. Este padrão pode ser usado com taxas de transmissão de até 10 Mbps e a distância máxima é de 1200m (para esta distância, só é possível atingir uma taxa de transmissão de no máximo 100 kbps). É constituído por dois terminais diferenciais, A (+) e B (-), com nível de sinal entre -7 V e +12 V e normalmente precisa de uma

resistência de terminação com um valor de impedância correspondente à impedância característica do barramento (valor usual de 120Ω). O objetivo desta resistência é atenuar reflexões, estas que, por sua vez, distorcem os dados transmitidos. Isto permite aumentar a distância e a velocidade de transmissão (CANICEIRO, 2018). A Figura 4 ilustra um exemplo de mensagem no padrão RS-485 com os níveis de tensão:

Figura 4 – Mensagem no Padrão RS-485



Fonte: (CANICEIRO, 2018)

Na Figura 4 é possível ver uma ilustração do sinal de uma mensagem no padrão RS-485, com o mesmo conceito representado na Figura 2, porém é mostrado que a distinção entre os níveis lógicos 1 (*Mark* - Nível positivo) e 0 (*Space* - Nível negativo) se dá pela diferença entre U+ e U-.

2.2.1.3 USB

Atualmente, a maioria dos aparelhos eletrônicos desenvolvidos e projetados para o mercado utiliza portas USB para se comunicar com computadores, capazes de realizar transferências de arquivos, executar aplicações de monitoramento e controle, áudio, internet *wireless*, etc. A comunicação USB é a evolução da comunicação serial RS-232, por possuir um protocolo mais amplo e complexo. Por padrão, toda empresa que vende módulos USB fornece os *drivers* para aplicação (AXELSON, 2009). Segundo Santos e Resende (2009), o USB oferece uma alta velocidade de comunicação sem comprometer a confiabilidade. O custo dos componentes é relativamente baixo e há suporte em todos os sistemas operacionais. Trata-se de uma comunicação serial assíncrona, que, para manter o sincronismo usa a codificação NRZI (*Non-Return to Zero Inverted*). É considerado como um barramento receptor ou de contagem. Quem inicia as transferências de dados é o controlador do *host* (hospedeiro) e todas as transações envolvem a transmissão de até três pacotes. Os pacotes são do tipo *token*, *data* e *handshake*. O *software* do sistema USB no *host* gerencia interações entre os dispositivos USB e o *software* do dispositivo instalado no próprio hospedeiro.

Dentre as principais características do USB estão:

- A máquina (computador) se comporta como um Servidor;
- Podem ser conectados até 127 dispositivos no Servidor;

- Cada cabo USB pode ter até 5 metros de comprimento;
- Para o USB 2.0, a taxa máxima de transferência de dados para o barramento é de 480Mbps;
- O cabo USB possui um par trançado de fios para transmissão dos dados e dois fios para energia, sendo eles um para alimentação (+5Vdc) e um para terra (GND);
- É possível fornecer até 500mA a 5V a partir do computador pelos cabos de energia.

Como explicam Zembovici e Franco(2009), existe apenas um *host* (hospedeiro) no sistema USB. A interface USB para o sistema do hospedeiro refere-se ao seu controlador. O controlador de *host* pode ser implementado combinando *hardware* e *software*. O *host* USB interage com o dispositivo por meio de seu controlador e é responsável por:

- Disponibilizar alimentação para os dispositivos conectados;
- Reconhecer a conexão e desconexão dos dispositivos USB;
- Controlar o fluxo de dados entre ele e os periféricos.

Cada dispositivo USB é responsável pela implementação de subdispositivos que respondam eletricamente às transações. Esses subdispositivos são chamados de *endpoints*. O início de cada transação se dá quando o controlador do *host* envia um pacote, que descreve tipo e direção da transação, endereço do dispositivo e número de *endpoint*. Cada *endpoint* recebe um endereço único, sendo este composto por um número e também pela direção dos dados (envio ou recebimento). Todo dispositivo deve ter um *endpoint* de número zero, o qual faz transações de controle (envia e recebe dados). O *endpoint* de número zero é composto por dois *endpoints*: um de envio e outro de recebimento (com mesmo endereço). São disponibilizados 4 bits para endereçamento dos *endpoints*, o que torna possível um total de 16 (incluindo o de número zero). (SANTOS; RESENDE, 2009)

Em uma transação ocorre a transferência de dados do dispositivo para o *host* ou vice-versa, sendo a direção desta transferência indicada no *token packet*. O emissor da transação então envia um pacote de dados ou indica que não há mais dados a serem transferidos. O destinatário usualmente responde com um *handshake packet*, que indica o sucesso na transferência. Os pacotes do tipo *token* são pacotes de *setup* usados a fim de indicar que o próximo pacote (do tipo *data*) será para escrita no endpoint zero. Podem ser do tipo *in*, indicando que o próximo pacote *data* deve ser lido do *endpoint* de saída ou do tipo *out*, indicando que o próximo pacote *data* deve ser escrito no *endpoint* de entrada. Os pacotes *data* tratam-se dos sinais data1 e data0, que são, de fato, os dados transportados (do *host* ou vice-versa, conforme o *token packet* predecessor). O pacote *handshake* trata de informações como status e confirmações. Podem ter os valores *ack*, *nak* ou *stall*, indicando que o dado foi recebido corretamente, que o *host* ou dispositivo está ocupado ou inoperante no momento ou que há um erro na comunicação, respectivamente. (SANTOS; RESENDE, 2009)

O endereçamento ou enumeração se trata da atividade que identifica os dispositivos ligados ao barramento e atribui endereços únicos a eles. É considerada uma atividade ininterrupta devido ao fato de permitir conexão ou remoção de dispositivos em qualquer instante de tempo. Por este fato, também compreende a detecção e processo de remoção dos periféricos.

2.2.2 Rede *Ethernet*

A *Ethernet* é um padrão de camada física e camada de enlace que opera à 10 Mbps com quadros que possuem tamanho entre 64 e 1518 *bytes*. O endereçamento é feito através de uma numeração única para cada *host* com 6 *bytes*, sendo os primeiros 3 *bytes* para identificação do fabricante e os 3 *bytes* seguintes para o número sequencial da placa. Esta numeração é conhecida como endereço MAC – *Media Access Control*. A subcamada MAC, pertencente à camada 2 da pilha de protocolos OSI (*Open System Interconnection*, que significa Sistemas Abertos de Interconexão), controla a transmissão, a recepção e atua diretamente com o meio físico.

A tecnologia *ethernet*, basicamente, consiste de três elementos: o meio físico, as regras de controle de acesso ao meio e o quadro ethernet. O modo de trasmissão pode ser: *Simplex*, *Half-Duplex* ou *Full-Duplex*.

O modo de transmissão em *half-duplex* requer que apenas uma estação transmita enquanto todas as outras aguardam em silêncio (característica básica de um meio físico compartilhado). O controle deste processo fica a cargo do método de acesso *Carrier Sense Multiple Access with Collision Detection* - CSMA/CD, que busca evitar colisões na transmissão de pacotes, devido ao fato de qualquer estação poder transmitir quando "percebe" o meio livre, podendo ocorrer que duas ou mais estações tentem transmitir simultaneamente. O CSMA/CD pode estar em três estados transmitindo, disputando ou inativo.

2.2.3 Comunicação *Wireless*

As comunicações sem fio se tratam de uma tecnologia em ascendência útil para acessar redes e serviços sem a necessidade de uso de cabos, promovendo flexibilidade e mobilidade. Isto é um benefício em relação às redes cabeadas, eliminando o próprio custo do cabeamento e sendo relativamente simples de desenvolver. O cenário dos principais protocolos *wireless* de curta distância pode ser reduzido aos itens especificados nas normas IEEE 802.15.1, 802.15.3, 802.15.4 e 802.11a/b/g, tratando-se das tecnologias *Bluetooth*, UWB, *ZigBee* e Wi-Fi, respectivamente (WILLIG, 2003).

Silva (2007) define a WPAN (*Wireless Personal Area Network*) como:

> [...]onde se enquadram tecnologias *wireless* de pequeno alcance, entre 10 e 100 metros. Esta é a área de estudo e desenvolvimento normativo do grupo

> de trabalho 15 do IEEE, especializado nos standards WPAN. Esta área compreende as redes sem fios que utilizam dispositivos como os PDA's, PC's ou periféricos, sendo o campo de tecnologias como o *Bluetooth* (responsabilidade do subgrupo IEEE 802.15.1) [...]

2.2.3.1 Wi-Fi (IEEE 802.11)

O padrão IEEE 802.11, conhecido como Wi-Fi, refere-se a uma família de especificações a respeito da tecnologia *wireless* LAN. Os autores Accardi e Dodonov (2012) citam as principais divisões como:

- 802.11a: Opera em frequência entre 5,1 e 5,8GHz e velocidade de 54 Mbps;
- 802.11b: Opera em frequência entre 2,4 e 2,485GHz e velocidade de 11 Mbps;
- 802.11g: Opera em frequência entre 2,4 e 2,485GHz, e velocidade de 54 Mbps;
- 802.11n: Opera em frequência de 2,4 e/ou 5GHz com velocidade de até 600Mbps

Biegelmeyer (2015) complementa, explicando que a divisão 802.11a usa o esquema OFDM (*Orthogonal Frequency Division Multiplexing*) no lugar do FHSS (*Frequency-hopping spread spectrum* ou espectro de difusão em frequência variável) ou DSSS (*Direct Sequence Spread Spectrum* ou sequência direta de espalhamento do espectro) e é incompatível com a 802.11b, devido à sua frequência de trabalho. A divisão 802.11b (*High Rate*), por sua vez, usa somente codificação DSSS, sendo uma extensão do 802.11 que permite um funcionamento semelhante às redes Ethernet. Tem um alcance por volta de sete vezes maior que o 802.11a. O padrão 802.11g é compatível com o 802.11b e o 802.11n é hoje em dia o mais comercializado em produtos eletrônicos.

O padrão Wi-Fi já tem suporte ao protocolo TCP/IP, altamente difundido ao redor do mundo na Internet. Outro fator importante para esta tecnologia é o fato de já existir uma infraestrutura construída, como, por exemplo, em residências, empresas, lugares públicos, etc. Isto torna fácil e versátil sua aplicação. (BIEGELMEYER, 2015)

2.2.4 Protocolo FTP

Conforme Castañeda (2012), o *File Transfer Protocol* (FTP - Protocolo de Transferência de Arquivos) é um protocolo de comunicação para transferir arquivos entre sistemas conectados por meio de uma rede do tipo *Transmission Control Protocol* (TCP - Protocolo de Controle de Transmissão), com base numa arquitetura do tipo cliente-servidor. Um usuário executa um programa cliente que se conecta a outra máquina com um programa servidor e, de conexão estabelecida, podem trocar arquivos de diversos tipos.

Um servidor FTP trata-se de um programa que se executa na máquina servidora, normalmente conectada à internet e seu papel é permitir a troca de informações entre diferentes máquinas. De maneira geral, os servidores não ficam nas máquinas pessoais e o usuário deve usar o protocolo para estabelecer comunicação com o servidor remotamente.

As aplicações mais comuns de servidores FTP são de hospedagem na web, nas quais os clientes podem usar o serviço para fazer upload de suas páginas da web e seus arquivos correspondentes, ou como servidor de backup (backup) de arquivos importantes.

Um cliente FTP usa o protocolo para se conectar a um servidor para transferir arquivos. Os modos de conexão podem ser do tipo Ativo ou Passivo.

No modo ativo, o servidor sempre cria o canal na porta 20, já no cliente o canal será uma porta aleatória maior que 1024. O cliente envia então um comando PORT destinado ao servidor indicando o número de porta, para que o servidor possa abrir uma conexão de dados pela qual os arquivos serão transferidos. Este modo acaba gerando um problema de segurança, pois a máquina cliente estará disposta a aceitar qualquer conexão de entrada em uma porta maior que 1024. A fim de sanar este problema, desenvolveu-se o modo passivo.

No modo passivo, o cliente envia um comando PASV e o servidor indica a porta (maior que 1023) na qual o cliente deve se conectar. O cliente começa então uma conexão da próxima porta de controle para a porta do servidor especificada anteriormente.

2.3 MICROCONTROLADOR

Um microcontrolador (MCU) é um dispositivo com a possibilidade de receber uma programação, que contém instruções de execução de determinadas tarefas. Este dispositivo possui memória e alguns periféricos (digitais e analógicos) com um núcleo de processamento em um mesmo *chip*. É usado em diversos equipamentos, principalmente em sistemas embarcados, que podem ser definidos como equipamentos com *software/firmware* implementado em *hardware* que realizam uma tarefa em tempo real. (NETO, 2019)

O autor ainda ressalta a criação de microcontroladores com arquitetura do tipo ARM (*Advanced RISC Machine*), por possuírem simples design, alta velocidade e grande diversidade de modelos, fabricantes e softwares (IDEs - *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) disponíveis. São microcontroladores que trabalham com barramentos de dados de 32 bits e usam estrutura *Reduced Instruction Set Computer* (RISC). Fabricantes como *STMicroelectronics*, *Freescale*, NXP, *Texas Instruments*, Atmel, Intel, Samsung usam este tipo de arquitetura, que é licenciado pela *ARM Limited*.

2.4 TRABALHOS CORRELATOS

Royo (2017) realizou a implementação de um sistema com comunicação USB e *Ethernet* a fim de realizar integração com uma série de sensores. Esta comunicação serviu para receber da-

dos que foram enviados para uma base de dados. Neste trabalho é vista uma topologia dividida entre sensores, interface e rede local. Os sensores se conectam por comunicação sem fio (utilizando tecnologia *Zigbee*) a um primeiro dispositivo de interface, que por sua vez realiza uma conexão via USB a um segundo dispositivo de interface. Este último se conecta via *Ethernet* à rede local.

Com o projeto de Royo (2017), pode-se verificar que um dos dispositivos trata-se de uma interface *Wireless* USB, tal qual proposto no presente trabalho, porém utilizando diferentes tecnologias e recursos. Isto torna possível entender que há uma demanda para este tipo de aplicação. O autor sugere ainda a criação de um servidor FTP em desenvolvimentos futuros, que vai de encontro ao implementado neste trabalho.

Já Oliveira (2004) fez um estudo de custo benefício com câmeras do tipo *webcam*. Como estas câmeras possuem conexão do tipo USB, utilizou um dispositivo com este tipo de conexão para receber os dados da câmera e enviar a um Servidor FTP, a fim de criar uma espécie de aplicação de monitoramento de câmeras de baixo custo. Contudo, o resultado apresentou resultados como baixa velocidade e baixa resolução. Embora o autor tenha obtido estes resultados, é possível concluir que desde que não sejam arquivos grandes (como capturas de imagem em alta resolução) e em grande quantidade, um servidor FTP é uma maneira possível de transmitir arquivos por conexão remota.

3 METODOLOGIA

Para desenvolver uma solução que atendesse à demanda vista neste trabalho, foi possível aplicar os conceitos revisados em um projeto de dispositivo eletrônico. Este dispositivo tem comunicação USB, comunicação sem fio e processamento e sua alimentação pode ser provida pela própria conexão USB. O processamento é realizado por um microcontrolador e se fez necessário o desenvolvimento de um código em linguagem específica, que possui rotinas de comandos e tarefas a serem interpretadas, processadas e executadas pelo componente. Este código se trata do *firmware* da aplicação.

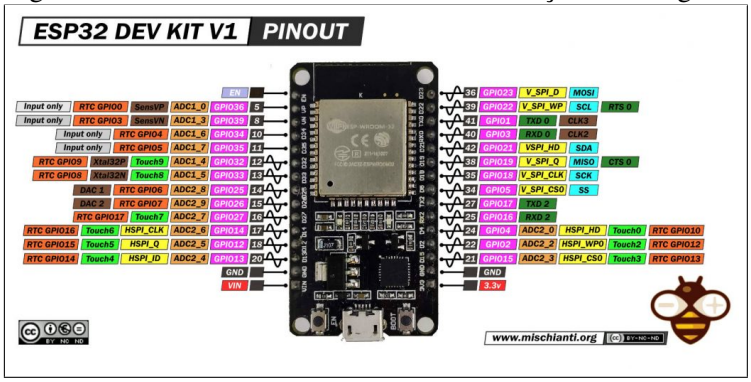
Este capítulo do trabalho apresenta as decisões tomadas antes do desenvolvimento do projeto para que o mesmo atendesse à necessidade exposta na seção 1.1, bem como a escolha realizada dos componentes a serem utilizados e suas especificações.

3.1 CONEXÃO *WIRELESS*

Como forma de acessar os arquivos desejados no servidor de arquivos, foi utilizada uma comunicação *wireless*. Devido às redes Wi-Fi serem uma tecnologia muito comum na indústria e à sua aplicabilidade na transferência de arquivos, este padrão foi utilizado na comunicação sem fio.

Para executar a comunicação Wi-Fi entre o dispositivo e o Servidor, foi utilizado o módulo ESP32 (Espressif Systems), devido à disponibilidade de acesso do autor à placa ESP32 DEVKIT-V1 que possui este chip como base, além de conexões e periféricos adicionais a fim de facilitar o desenvolvimento. Este módulo opera conforme o padrão IEEE 802.11 b/g/n, entre 2,4 e 2,5GHz, com velocidades de até 72,2 Mbps e poder de transmissão de 14dBm (@802.11n). Possui pinos de UART (*Universal asynchronous receiver/transmitter*) para comunicação serial e interligação com o microcontrolador (processamento). A Figura 5 mostra os pinos da placa utilizada e as funções disponíveis em cada um deles:

Figura 5 – Placa ESP32 DEVKIT-V1 e Descrição de Pinagem



Fonte: (MISCHIANTI.ORG, 2022)

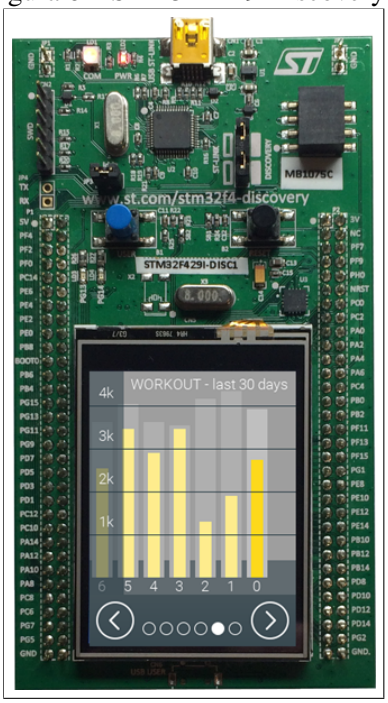
3.2 Conexão Física (USB)

Para disponibilizar os arquivos à máquina CNC, é necessário se comunicar com este último. Esta conexão se deu de forma física/elétrica, pois estas máquinas raramente possuem conexão sem fio. Como o USB é um padrão altamente difundido não só na indústria, como em computadores e equipamentos eletrônicos, este foi o padrão aplicado na conexão física à máquina (Cliente) que deve exibir os arquivos recebidos do Servidor. Para tal, foi necessário definir os componentes de *hardware* utilizados para esta conexão da máquina com o microcontrolador do dispositivo e implementar em *firmware* o controle desta comunicação. Esta conexão também serviu como fonte de energia do circuito do dispositivo, devido ao fornecimento de 5V realizado por este padrão.

3.3 Processamento (Microcontrolador)

Alguns fabricantes fornecem *kits* de desenvolvimentos, que são ferramentas utilizadas a fim de facilitar a criação e teste de aplicações em um componente ou plataforma de *hardware*. Devido à disponibilidade do recurso ao autor, foi inicialmente utilizada a placa STM32F429 DISCOVERY da STMicroelectronics com a finalidade de auxiliar no desenvolvimento do projeto. A Figura 6 mostra o *kit* de desenvolvimento utilizado.

Figura 6 – STM32F429 Discovery



Fonte: (EMBEDDED-WIZARD, 2021)

Esta placa tem como elemento foco o microcontrolador STM32F429ZIT6U e possui recur-

sos de *hardwares* periféricos como: conexão USB, LCD de 2.4", sensor de movimento, botões de interação com o usuários, LEDs de indicação, memória SDRAM de 64 Mbits, conectores ligados aos pinos do microcontrolador, possibilidade de alimentação do circuito por USB, ferramenta de gravação de firmware ou fonte externa, entre outros.

O microcontrolador foi usado com intenção de gerenciar a comunicação USB com a máquina do Cliente, realizar leitura e *"bufferização"* dos dados e comunicação Wi-Fi com a máquina do Servidor. O *chip* STM32F429ZIT6U é compatível com as tarefas necessárias, pois possui diversos recursos, como um *chip* controlador USB 2.0 integrado e pinos de comunicação I2C, SPI, UART e USART, que podem ser utilizados para interligação com o módulo Wi-Fi. Ele possui 2MB de memória Flash e 256+4kB de SRAM e sua alimentação é de 3,3V. Possui 4 portas UART, que podem servir de canal para a comunicação serial implementada microcontrolador e módulo ESP32.

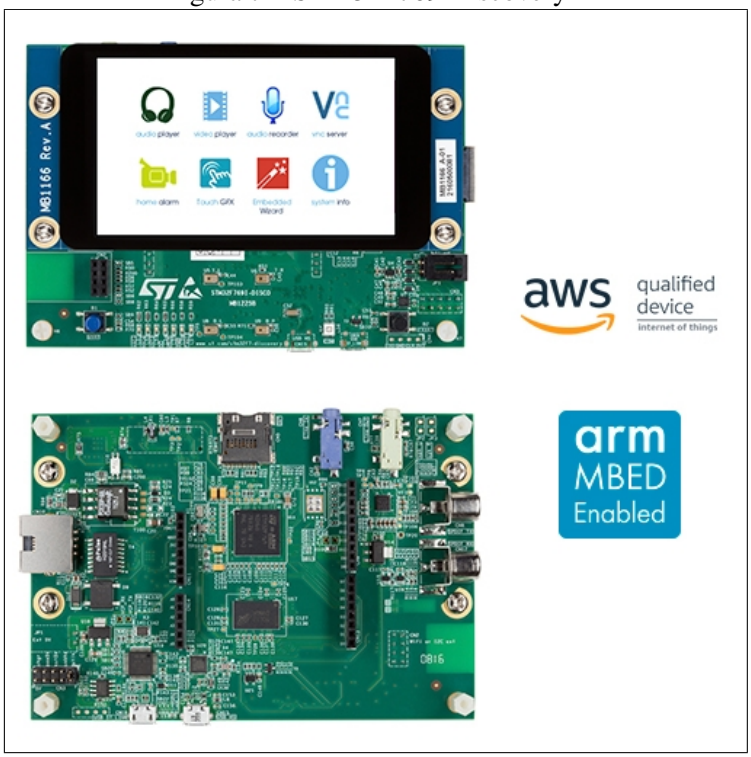
Entretanto, no decorrer do desenvolvimento deste trabalho utilizando a placa STM32F429 DISCOVERY foram encontrados obstáculos no funcionamento. Foi verificado que ao implementar funcionalidades de comunicação USB e realizar a gravação do *firmware* na placa, o Sistema Operacional do computador não reconheceu o dispositivo ao realizar a conexão física via porta USB. Foram realizadas diversas revisões e testes na tentativa de sanar a falha, mas não se obteve sucesso. Este tipo de problema pode ocorrer por uma descompatibilização entre as funções utilizadas e as versões de bibliotecas elencadas pelo software ou então por algum problema de hardware, por exemplo. Encontrou-se inclusive uma reclamação não solucionada a respeito do uso da funcionalidade USB da mesma placa no fórum do fabricante.

A partir do obstáculo encontrado, decidiu-se utilizar, por disponibilidade de acesso do autor à ferramenta, a placa de mesmo fabricante do modelo STM32F769 DISCOVERY. Os recursos desta placa e o microcontrolador principal são de tecnologias superiores às disponíveis na placa STM32F429 DISCOVERY. Ela tem como elemento foco o microcontrolador STM32F769NIH6, com 2 *Mbytes* de memória *flash* e 532 *Kbytes* de memória RAM e possui recursos de *hardwares* periféricos como: conexão USB, LCD touch de 4", entradas e saídas de áudio, microfones, soquetes para ligação com módulo ESP8266 e Arduino, conector de cartão SD, entre outros. A Figura 7 ilustra a nova placa utilizada.

Para o desenvolvimento desta aplicação, foi dado enfoque nos recursos disponíveis de:

- conexão USB: para conectar à Máquina do Cliente;
- conector de cartão de memória SD: para utilizar um cartão como forma de armazenamento de arquivos;
- conectores com pinos de ligação aos módulos UART/USART do microcontrolador: para realizar comunicação com o módulo Wi-Fi utilizado.

Figura 7 – STM32F769 Discovery



Fonte: (STMICROELECTRONICS, 2021)

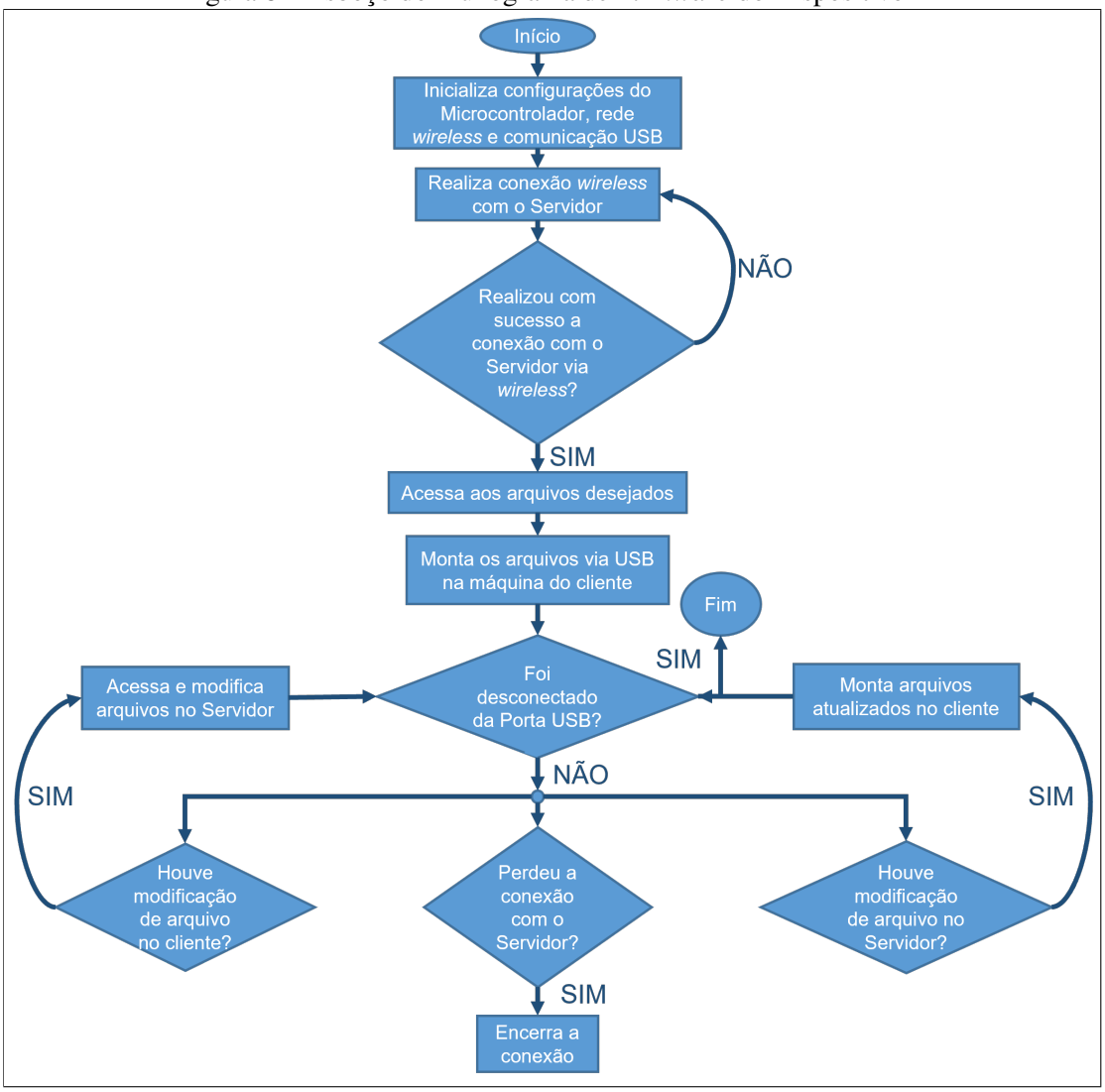
3.4 PRÉ-DEFINIÇÕES DE *FIRMWARE*

Para que o microcontrolador realizasse as tarefas pré-determinadas e o dispositivo pudesse operar corretamente, foi necessário o desenvolvimento de um *firmware*. Este *firmware* foi embarcado no microcontrolador e serviu para controlar a operação do mesmo. O desenvolvimento do *firmware* se deu através da criação de um código fonte em linguagem C. O *firmware* teve que implementar rotinas de:

- Realização de comunicação serial com o módulo Wi-Fi;
- Realização de comunicação USB com a máquina do Cliente (*hardware* já compatível);
- Gerenciamento de leitura e escrita dos arquivos, tanto no Cliente quanto no Servidor;
- Gerenciamento das filas de envio e recebimento dos dados.

A Figura 8 ilustra um fluxograma esboçado para concepção inicial do *firmware* do dispositivo:

Figura 8 – Esboço do Fluxograma de *Firmware* do Dispositivo



Fonte: O Autor (2022)

Como um desenvolvimento de uma aplicação em *firmware* normalmente acontece gradativamente, implementando uma rotina por vez - para que se possa testar e verificar a existência de erros à medida que novos trechos de código são criados - foi interessante o uso de uma ferramenta de versionamento (que pode ser chamada de ferramenta Git). Desta forma foi possível, por exemplo, retornar a uma versão anterior do código que apresentava rotinas funcionais quando uma nova versão estiver apresentando funcionamentos inadequados .

3.5 TESTE

Para conexão com a rede Wi-Fi, foi utilizado um computador com Sistema Operacional *Windows* e foi necessário usar um software que realize conexões com servidores do tipo FTP. Este computador foi responsável por criar, editar e remover arquivos do servidor.

Devido ao fato de um torno CNC também possuir um Sistema Operacional *Windows* em-

barcado, foi possível realizar testes também utilizando um computador, de forma a simular o Cliente do sistema. Este último se comunica com o dispositivo conectado via USB de forma a exibir os arquivos disponibilizados pelo Servidor FTP (refletindo as alterações feitas pela outra máquina).

4 DESENVOLVIMENTO

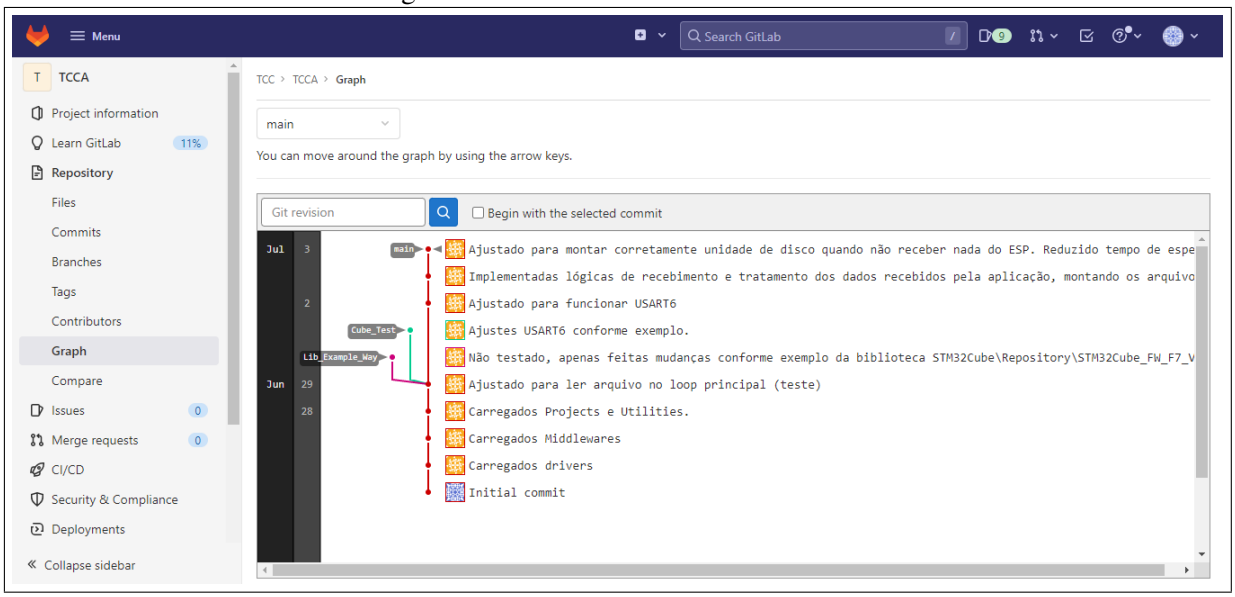
Embora tenha sido necessário escolher as tecnologias de *hardware* para que o dispositivo atendesse à aplicação, o maior desafio deste trabalho foi a implementação de um *firmware* que realizasse o funcionamento adequado sobre estas tecnologias. Aqui neste capítulo foram descritas as análises críticas sobre o projeto, decisões tomadas e os obstáculos encontrados durante o desenvolvimento.

Como o sistema baseia-se em três principais partes, sendo elas a comunicação USB, a comunicação wireless e a integração de ambas em uma lógica de processamento, foi escolhido implementar cada parte em sua vez, optando-se por começar com a interface USB com a máquina do Cliente, para depois realizar a interface Wi-Fi com o Servidor e por fim integrar todo o sistema e implementar o tratamento dos dados de uma ponta à outra.

4.1 CONTROLE DE VERSÕES

Como forma de controlar as versões de *firmware*, foi escolhido o servidor gratuito online *GitLab.com*, que disponibiliza espaço para criação de projetos e repositórios para versionamento. A Figura 9 exibe a página de histórico de alterações em forma de gráfico. Nela pode-se ver à esquerda um menu com os recursos disponíveis para gerenciar o projeto criado e ao centro o histórico de alterações com ramificações.

Figura 9 – Gráfico de Versões no Gitlab

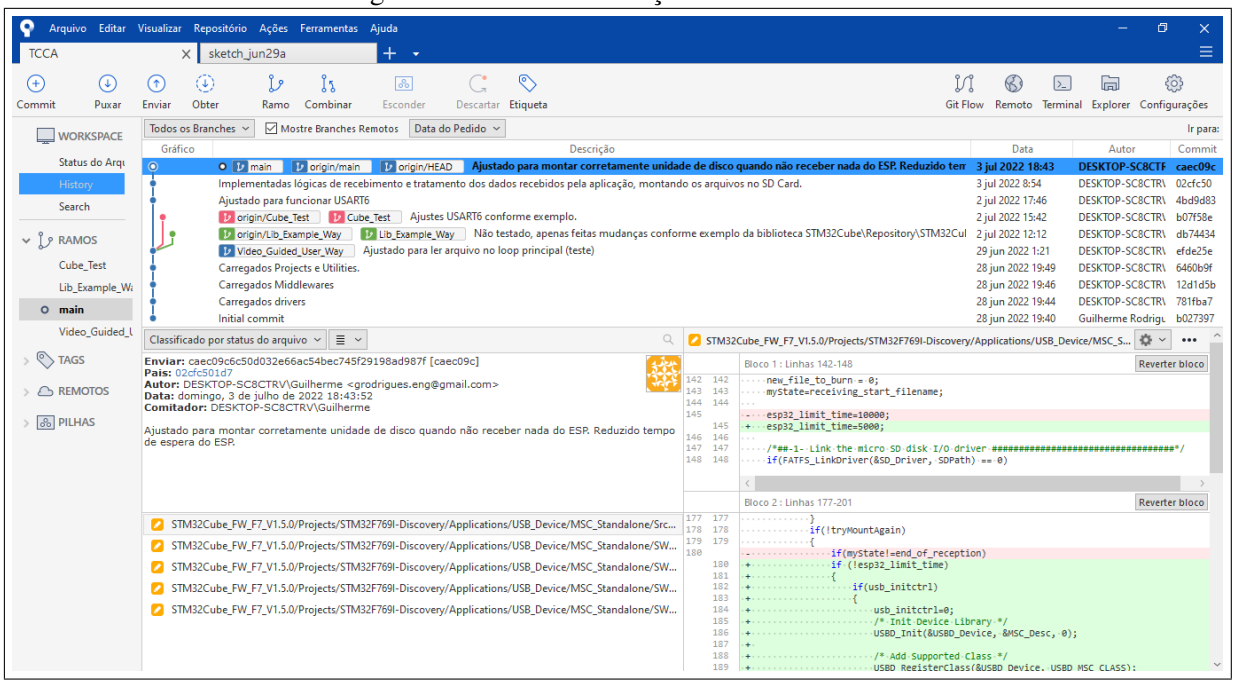


Fonte: O Autor (2022)

Já para uso local (alterações em disco na máquina) é mais prático utilizar um *software* específico que sirva de interface para o usuário e atualize os arquivos no servidor. Para esta

tarefa, foi escolhido o *software Sourcetree*. A Figura 10 ilustra a interface do *software* em uma tela de histórico de alterações para o mesmo projeto exibido no exemplo anterior do servidor *GitLab*. Nesta imagem há à esquerda um menu com recursos de gerenciamento do projeto, na parte superior há uma barra de ferramentas, ao centro o histórico de alterações com ramificações e na parte inferior estão elencados os arquivos alterados em relação à ultima versão.

Figura 10 – Tela de Alterações no Sourcetree



Fonte: O Autor (2022)

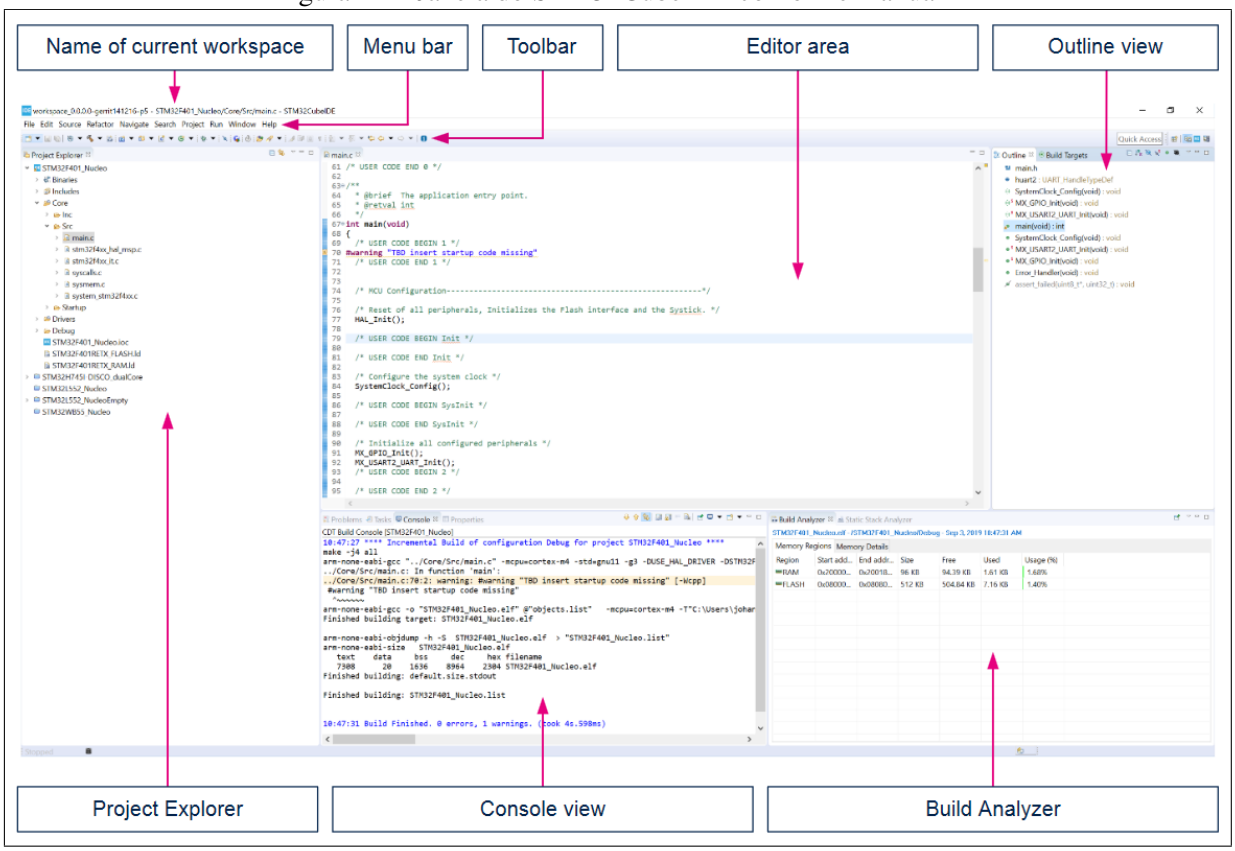
4.2 Implementação da Comunicação USB

A fim de se conectar à máquina do Cliente, sendo a responsável por exibir os arquivos disponibilizados pelo Servidor quando a aplicação estiver concluída, foi utilizada a placa STM32F769 DISCOVERY, que possui um conector do tipo USB micro OTG (*On The Go*) para que seja realizada a conexão com a máquina do cliente via cabo USB. Como a placa possui um *slot*, o armazenamento dos arquivos foi feito na memória de um cartão SD.

Para editar, compilar e gravar um *firmware* nesta placa foi utilizada a IDE *STM32CubeIDE* - Figura 11 - do próprio fabricante do microcontrolador. Esta IDE possui bibliotecas de funções prontas que controlam os recursos de hardware disponíveis em seu mais baixo nível, a fim de que o desenvolvedor tenha mais facilidade ao ter que lidar somente com a lógica de funcionamento do sistema.

Através da interface do software IDE, foi possível habilitar e configurar o funcionamento do recurso de USB na placa, utilizando como auxílio tutoriais disponíveis do fabricante e de outros geradores de conteúdo de programação de microcontroladores.

Figura 11 – Janela do STM32CubeIDE conforme manual



Fonte: (STMICROELECTRONICS, 2021)

Na Figura 11 há acima uma barra de ferramentas, à esquerda a árvore de arquivos do projeto, ao centro o editor com o código fonte selecionado, à direita uma lista das funções do código fonte selecionado e abaixo um gerador de relatório e um analisador de compilações.

Para que o sistema se comportasse como um *flash drive*, conforme proposto neste trabalho, a funcionalidade USB foi direcionada a criar um dispositivo do tipo MSC (*Mass Storage Class* - classe de dispositivo de armazenamento em massa). Para este modelo de placa há um exemplo na biblioteca do fabricante de dispositivo MSC, que foi utilizado como base. As principais bibliotecas para funcionamento da comunicação USB nesta aplicação estão ilustradas na Figura 12:

Figura 12 – Bibliotecas USB

```
/* Includes ------------------------
#include "stm32f7xx_hal.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_msc.h"
#include "usbd_storage.h"
#include "stm32f769i_discovery.h"
#include "stm32f769i_discovery_sd.h"
```

Fonte: O Autor (2022)

A base de bibliotecas estava originalmente na versão mais atual (1.16.2) e apresentou funcionamentos não satisfatórios com o exemplo de aplicação MSC, quando gravada na placa e conectada a um computador, executando de forma adequada somente quando testada na versão 1.5.0. Logo, esta versão foi utilizada como base para desenvolvimento do *firmware* da placa STM32F469 DISCOVERY. O caminho padrão para as bibliotecas no armazenamento do computador é: "C:/Users/User/STM32Cube/Repository".

As funções que devem ser chamadas para inicialização e funcionamento da comunicação USB como MSC estão ilustradas na Figura 13.

Figura 13 – Funções de Inicialização e Configuração USB

```
USBD_StatusTypeDef USBD_Init(USBD_HandleTypeDef *pdev, USBD_DescriptorsTypeDef *pdesc, uint8_t id);
USBD_StatusTypeDef USBD_RegisterClass(USBD_HandleTypeDef *pdev, USBD_ClassTypeDef *pclass);
uint8_t  USBD_MSC_RegisterStorage  (USBD_HandleTypeDef   *pdev,
                                    USBD_StorageTypeDef *fops);
USBD_StatusTypeDef USBD_Start  (USBD_HandleTypeDef *pdev);
```

Fonte: O Autor (2022)

4.3 Implementação das Funções de Manipulação dos Arquivos no Cartão SD

Uma vez que o *firmware* da placa STM32F769 DISCOVERY já estava com rotinas de USB implementadas, foi importado funções de gerenciamento de arquivos para montar o volume no cartão SD, acessar o mesmo e poder ler e escrever arquivos neste volume. Foi utilizado o exemplo de gerenciamento FAT (*File Allocation Table* - Tabela de Alocação de Arquivos) para cartão SD disponível na versão 1.16.2 para uma placa similar (STM32F769I EVAL). Foi necessário estudar o exemplo e importar para o código as funções necessárias, sendo elas as ilustradas na Figura 14:

Figura 14 – Funções FAT para Manipulação dos Arquivos no Cartão SD

```
uint8_t FATFS_LinkDriver(Diskio_drvTypeDef *drv, char *path)
{
  return FATFS_LinkDriverEx(drv, path, 0);
};
FRESULT f_mount (
    FATFS* fs,          /* Pointer to the file system object (NULL:unmount)*/
    const TCHAR* path,  /* Logical drive number to be mounted/unmounted */
    BYTE opt            /* 0:Do not mount (delayed mount), 1:Mount immediately */
);
FRESULT f_open (
    FIL* fp,            /* Pointer to the blank file object */
    const TCHAR* path,  /* Pointer to the file name */
    BYTE mode           /* Access mode and file open mode flags */
);
FRESULT f_write (
    FIL* fp,            /* Pointer to the file object */
    const void *buff,   /* Pointer to the data to be written */
    UINT btw,           /* Number of bytes to write */
    UINT* bw            /* Pointer to number of bytes written */
);
FRESULT f_close (
    FIL *fp     /* Pointer to the file object to be closed */
);
```

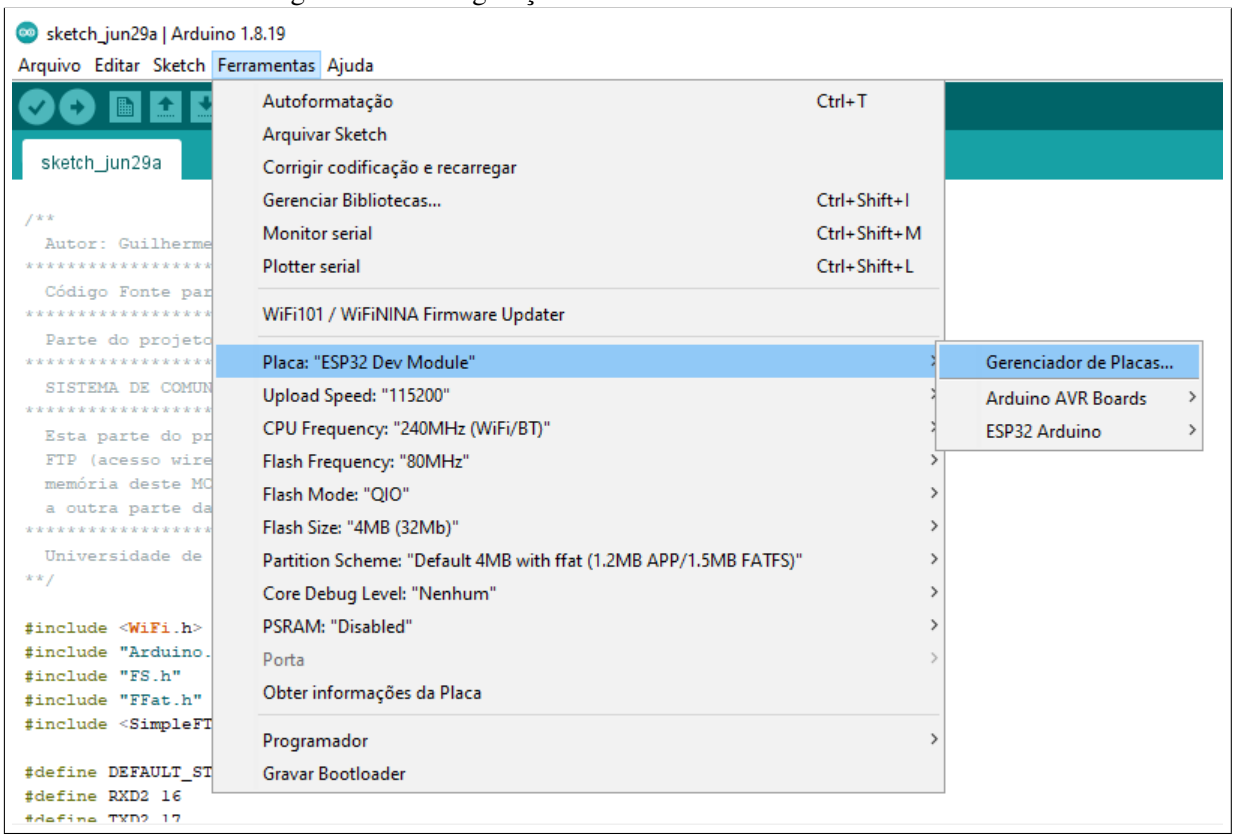
Fonte: O Autor (2022)

4.4 Implementação da Comunicação Wi-Fi

Para que o dispositivo possua uma rede Wi-Fi, foi escolhido o componente ESP32 capaz de realizar esta função (como visto anteriormente). Este componente é um módulo de alta performance que recebe também um *firmware* para realizar operações e tarefas. Para o *firmware* deste componente foi utilizada a linguagem C++, uma evolução da linguagem C.

Pode-se utilizar IDEs como *Visual Studio Code* (da *Microsoft Corporation*) ou então Arduino, desde que seja usado como base as bibliotecas ESP-IDF disponibilizadas pelo fabricante. Optou-se pela IDE Arduino, realizando-se a configuração adequada de placa para uso, conforme pode ser visto na Figura 15:

Figura 15 – Configuração da Placa ESP32 na IDE Arduino



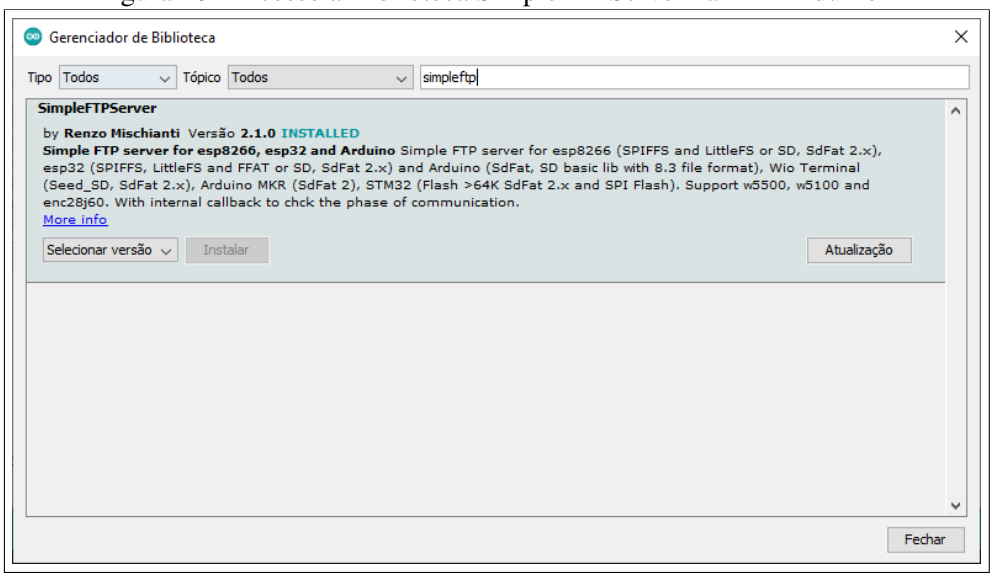
Fonte: O Autor (2022)

O módulo ESP32 deve se conectar a uma rede Wi-Fi e estar disponível para ser acessado através do seu endereço IP, portanto foi utilizada a biblioteca "WiFi.h", que possui a função de inicialização e configuração da conexão Wi-Fi "WiFi.begin(ssid, pass)", onde "ssid"é o nome da rede Wi-Fi na qual o dispositivo estará conectado e "pass"é a senha de segurança desta rede.

Ligado à rede Wi-Fi, o dispositivo executa então um Servidor FTP (ideal para esta aplicação) para manipular os arquivos desejados. Devido ao material disponível de exemplo na internet, escolheu-se utilizar a biblioteca *SimpleFTPServer*, que pode ser baixada no menu exibido na Figura 16. Esta biblioteca possui funções de inicialização, configuração e gerencia-

mento do servidor FTP, apenas é necessário efetuar no setup do código a chamada da função "ftpSrv.begin(user, pass)", onde *user* é o nome de usuário para acesso do cliente e *pass* é a senha de acesso (neste caso utilizaremos os valores "esp32"para ambos). Depois efetua-se a chamada da função "ftpSrv.handleFTP()"no *loop* principal, que fará o gerenciamento deste servidor.

Figura 16 – Acesso à Biblioteca SimpleFTPServer na IDE Arduino



Fonte: O Autor (2022)

O lado positivo de utilizar a biblioteca *SimpleFTPServer* é a autossuficiência da mesma em suas rotinas e a simplicidade para executar o Servidor FTP, enquanto o lado negativo é a dificuldade em fazer alterações na lógica de funcionamento deste servidor, uma vez que são funções já desenvolvidas para operar interagindo entre si dentro da própria biblioteca.

4.5 Implementação das Funções de Manipulação dos Arquivos na Memória Flash do ESP32

Depois de inicializado o servidor FTP, foi declarado a forma de armazenamento dos arquivos utilizados por este servidor. Como não há memória externa na placa utilizada, escolheu-se armazenar na memória *Flash*, através da definição no código fonte utilizando a diretiva "#define DEFAULT_STORAGE_TYPE_ESP32 STORAGE_FFAT". Após realizar esta definição, foi feita a respectiva configuração na IDE, conforme Figura 17.

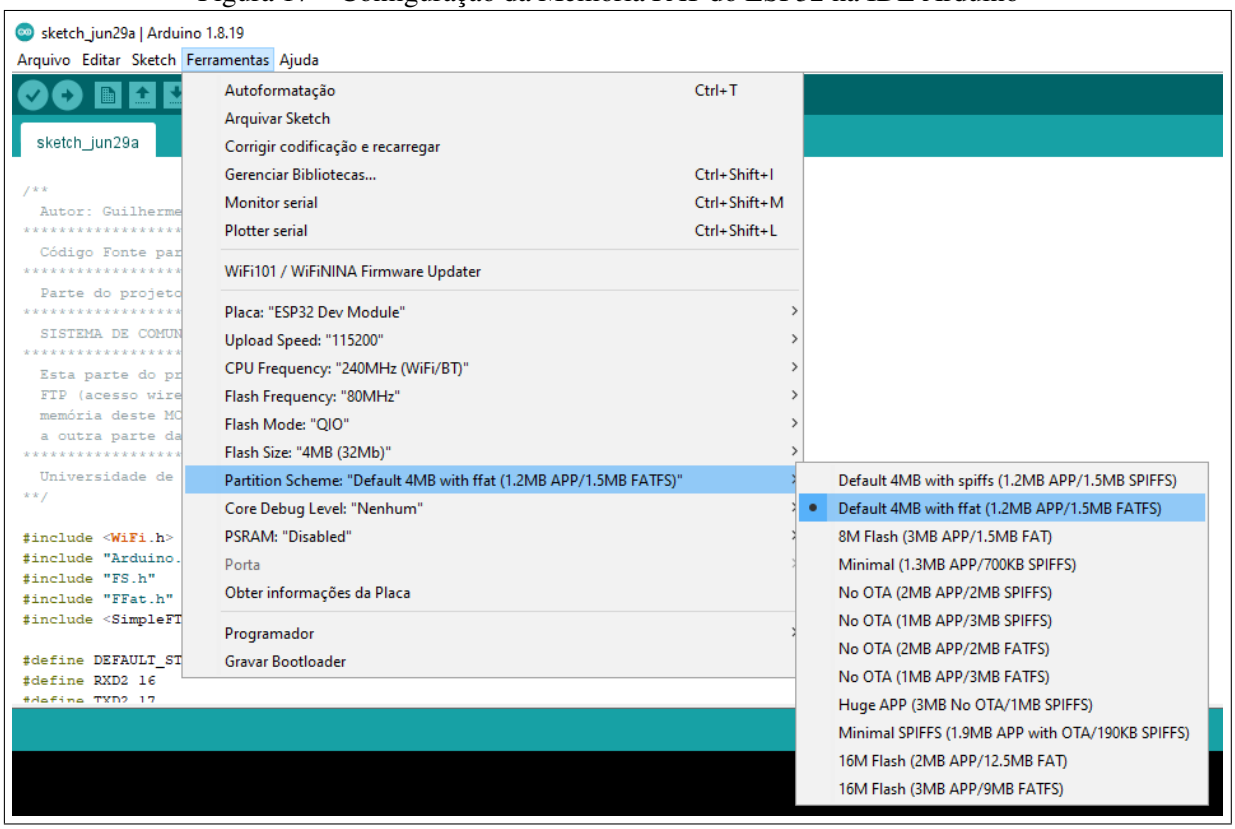
Fez-se a inclusão também das bibliotecas "FS.h"e "FFat.h"no código da aplicação. Estas bibliotecas são importantes para fazer manipulação dos arquivos na memória do ESP32. Foram utilizadas, principalmente, as funções descritas a seguir:

- "FFat.begin()"para criar um sistema de arquivos FAT;
- "FFat.open()"para abrir um arquivo ou diretório no sistema de arquivos FAT;

- "dir.openNextFile()"para informar o próximo arquivo ou diretório na memória;
- File.readString() para ler um arquivo e retornar em caracteres ASCII.

As funções exibidas acima foram importantes na lógica de leitura da memória do ESP32, a fim de acessar estes arquivos e disponibilizá-los posteriormente para o restante do sistema (Cliente).

Figura 17 – Configuração da Memória FAT do ESP32 na IDE Arduino



Fonte: O Autor (2022)

Como a placa ESP32 DEVKIT-V1 possui um gravador de *firmware* embarcado nela, pôde-se gravar facilmente o código gerado através da IDE, via conexão da porta COM USB do PC à placa.

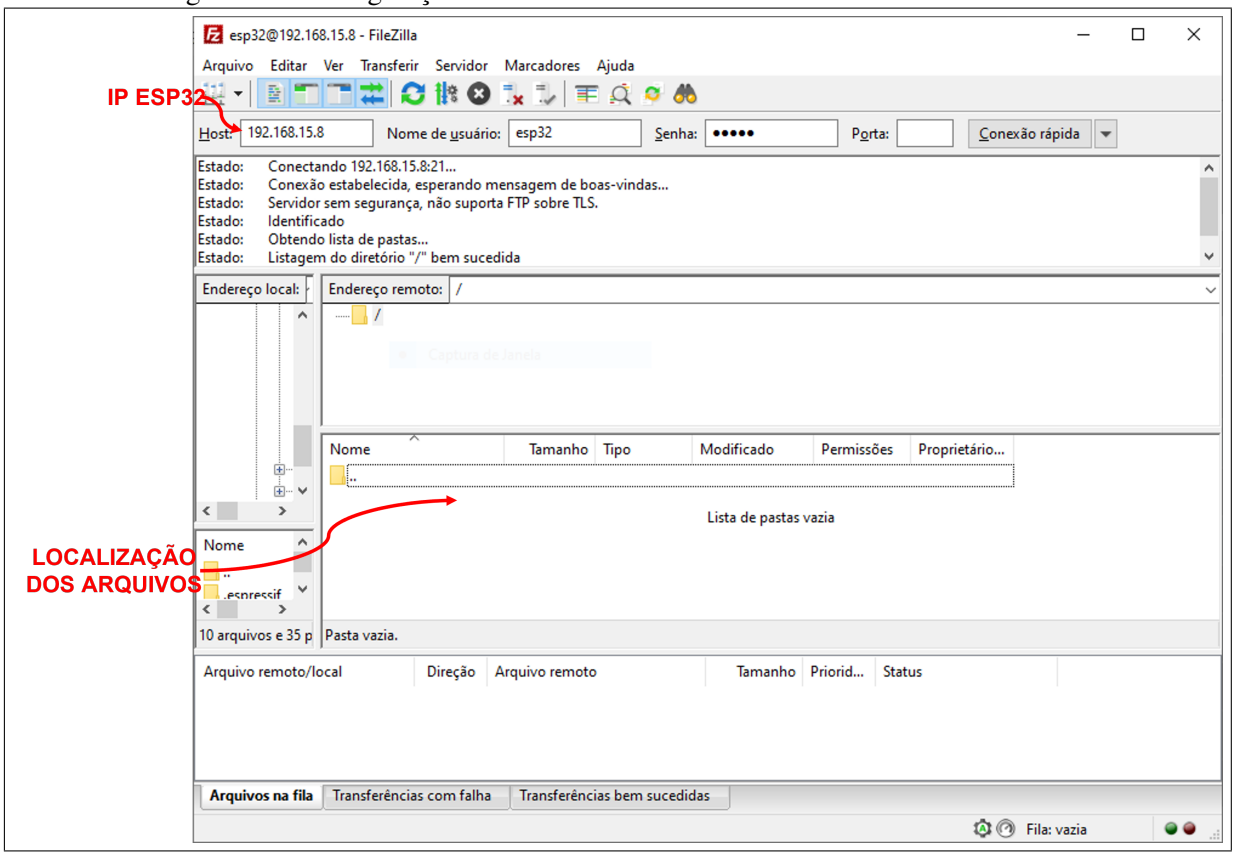
Com os passos já realizados, foi possível realizar testes de conexão, escrita e leitura com o servidor FTP implementado, conforme descrito na próxima seção.

4.6 Conexão ao Servidor FTP

Para testar o servidor FTP implementado, foi utilizado o *software FileZilla*, que realiza conexões como cliente a um servidor FTP da rede (vide configuração de conexão). Este *software* permite fazer carregamento de arquivos para o servidor, editar arquivos que estão no servidor e baixar para a máquina cliente FTP os arquivos que estão no servidor (neste momento é importante ressaltar que o cliente FTP é tratado com o Servidor do sistema proposto neste trabalho).

Pôde-se testar também a exibição correta de arquivos carregados/modificados em uma conexão anterior ao abrir uma nova conexão, garantindo que os dados ficaram salvos na memória do ESP. Na Figura 18 é possível ver a configuração realizada para efetuar a conexão. Na parte superior há uma barra de ferramentas e logo abaixo estão preenchidos os dados para conexão ao ESP32, como o IP do mesmo na rede, usuário e senha. Mais abaixo há um histórico de ações realizadas e por fim a interface com o diretório dos arquivos no servidor FTP.

Figura 18 – Configuração do Software Filezilla Para Acessar o Servidor FTP



Fonte: O Autor (2022)

4.7 Implementação da Comunicação Serial

Como o ESP32 armazena em sua memória os arquivos escritos pelo computador Servidor (via conexão FTP), mas a visualização para a máquina Cliente ocorre somente para os arquivos escritos no cartão SD da placa STM32F469 DISCOVERY, foi necessário implementar uma comunicação serial entre os componentes, para que os arquivos cheguem de uma ponta à outra. Optou-se por utilizar uma conexão serial comum assíncrona com valores padrões entre o módulo ESP32 e o microcontrolador STM32F469. Foram utilizados os pinos de Tx da UART 2 do ESP32 (acesso físico disponível no pino 27/GPIO17 da placa ESP32 DEVKIT-V1) e de Rx da USART6 do STM32F469 (acesso físico disponível no pino D0:CN7/PG9 da placa STM32F469 DISCOVERY).

Definiu-se um *baud rate* de 9600bps, 8 bits de dados, um stop bit e nenhum bit de paridade. O protocolo da comunicação foi escolhido como proprietário (customizado para a aplicação). Cada mensagem conta com um *byte* de cabeçalho de abertura (valores hexadecimais CA para nome de arquivo e DA para conteúdo), um ou dois *bytes* (um para nome de arquivo e dois para conteúdo) com o tamanho dos dados (quantidade n de *bytes* de dados), n *bytes* com os próprios dados (m valor ASCII) e *byte* com um identificador de fim de mensagem (valores hexadecimais C6 para nome de arquivo e D6 para conteúdo). Como a aplicação é voltadas a arquivos simples de texto (programas CNC), foi definido enviar uma mensagem com o nome do arquivo e outra com o conteúdo do mesmo, a fim de facilitar o trabalho no momento da escrita no cartão SD. Uma vez que os dados possuem valor ASCII, foram definidos valores fora desta faixa para os cabeçalhos e identificadores. Na Figura 19 pode-se ver um ilustrativo com o formato das mensagens:

Figura 19 – Formato das Mensagens da Comunicação Serial

Mensagem de nome de arquivo					
Abertura	Tamanho (máx. 255)	Dados			Fim
0xCA	n	Byte 0	...	Byte n-1	0xC6

Mensagem de conteúdo de arquivo						
Abertura	Tamanho (máx. 65535)		Dados			Fim
0xDA	n (MSB)	n (LSB)	Byte 0	...	Byte n-1	0xD6

Fonte: O Autor (2022)

O sistema pode ser resumido em unilateral, ao denominar o ESP32 como o transmissor e o STM32F769 como o receptor da comunicação.

Para efetuar a transmissão pelo ESP32, foi preciso realizar as seguintes configurações em código:

- Inicializar a UART2 com a configuração definida utilizando a função "Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2)";

- Realizar transmissões através das funções "Serial2.write()", para um valor numérico único (um único *Byte*) e "Serial2.print()"para um conjunto de caracteres (múltiplos *Bytes*).

O ESP32 varre então sua memória utilizando as funções citadas na seção 4.5 e envia as mensagens ilustradas na Figura 19 à medida que a leitura dos arquivos vai sendo realizada.

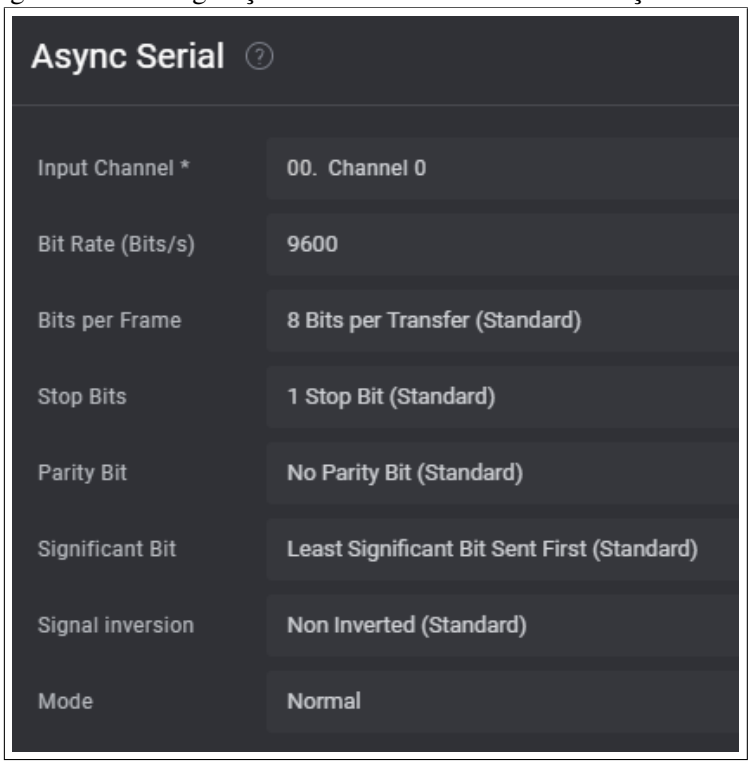
Para checar e validar a transmissão, foi utilizado o software Logic 2, destinado a este tipo de análise com aquisição de sinais de comunicação de dados via conexão de um analisador lógico conectado ao PC (ilustrado na Figura 20).

Figura 20 – Dispositivo Analisador Lógico Utilizado

Fonte: O Autor (2022)

A configuração no *software* para analisar o recebimento está ilustrada na Figura 21, utilizando o Canal CH0. Esta Figura ilustra a selação do canal, do *baud rate* da comunicação, do tamanho de cada *byte*, se há ou não stop bits e bits de paridade, qual a ordem dos bits e se o sinal elétrico é invertido ou não (por padrão sinal alto significa bit com valor 1).

Figura 21 – Configuração dos Parâmetros da Comunicação Serial

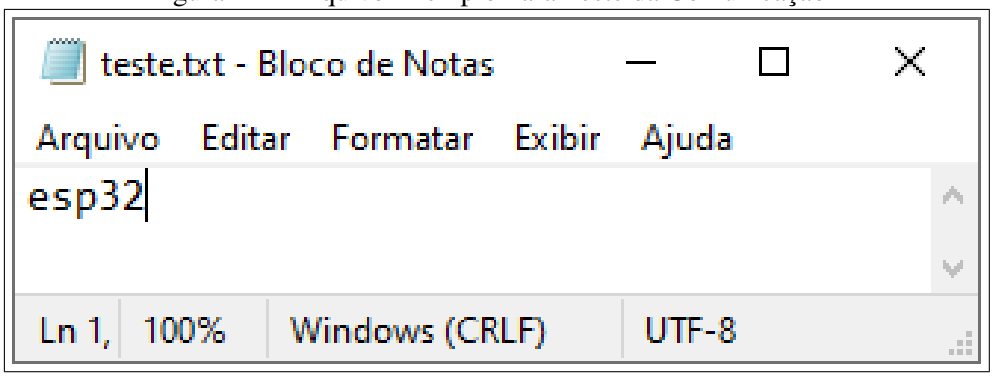


Fonte: O Autor (2022)

Foi utilizado o arquivo "teste.txt", com o texto "esp32"escrito nele. O arquivo está ilustrado

na Figura 22:

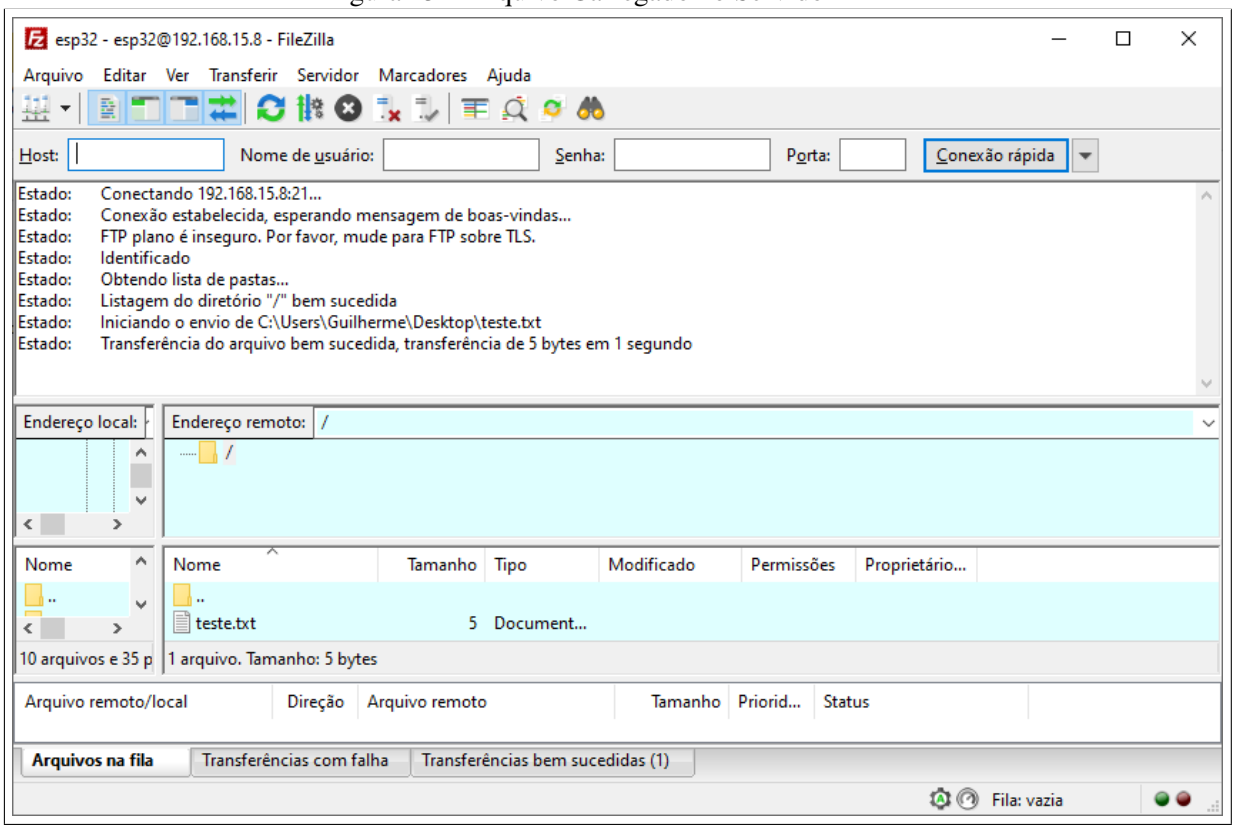
Figura 22 – Arquivo Exemplo Para Teste da Comunicação



Fonte: O Autor (2022)

O arquivo foi carregado no servidor FTP para ser gravado na memória do ESP32, como pode-se observar na figura 23:

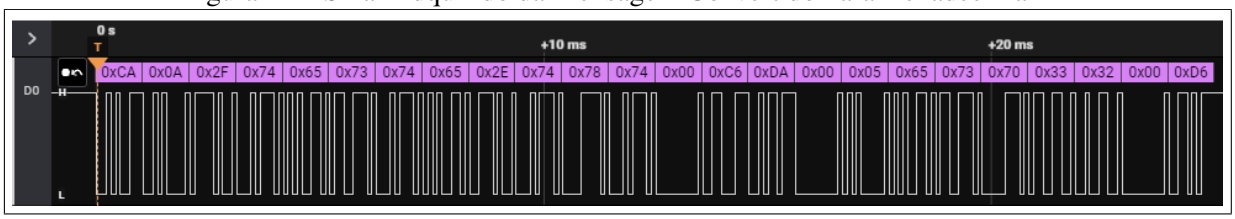
Figura 23 – Arquivo Carregado no Servidor FTP



Fonte: O Autor (2022)

Conectou-se o Canal CH0 do analisador lógico ao pino Tx da UART2 do ESP32 e foi possível verificar êxito na transmissão das mensagens. A Figura 24 exibe a ilustração gráfica do sinal elétrico adquirido, sendo o próprio *software* capaz de decodificar os dados no padrão hexadecimal (exibição acima de cada *byte*).

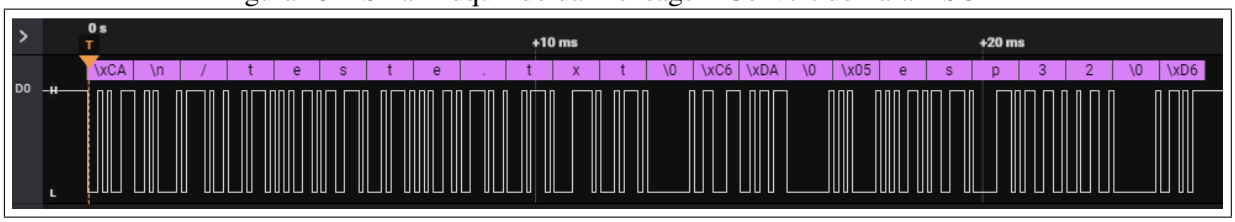
Figura 24 – Sinal Adquirido da Mensagem Convertido Para Hexadecimal



Fonte: O Autor (2022)

Já a Figura 25 exibe os mesmos dados convertidos para o padrão ASCII (podendo-se visualizar o nome e o conteúdo do arquivo):

Figura 25 – Sinal Adquirido da Mensagem Convertido Para ASCII



Fonte: O Autor (2022)

Para o recebimento destes dados na placa STM32F769 DISCOVERY, foi habilitado a USART6, que possui pinos de acesso no conector CN13. Foi utilizada a biblioteca "usart.h", configurando a recepção de dados para funcionar por interrupção, ou seja, executa uma rotina quando receber algum dado no pino Rx. As principais funções chamadas foram:

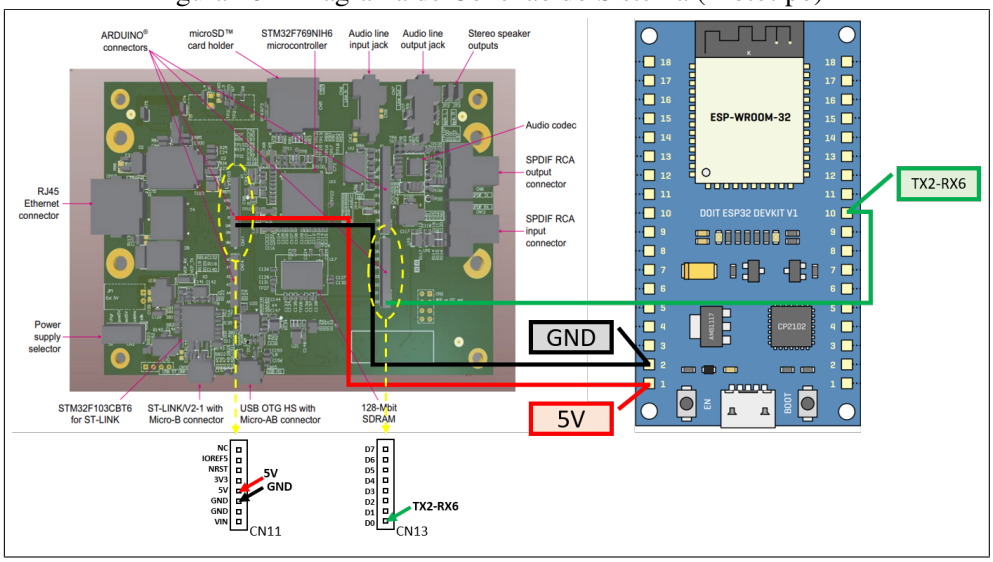
- "MX_USART6_UART_Init()"para inicializar a USART6 com definições do protocolo adotado na transmissão;
- "HAL_StatusTypeDef HAL_UART_Receive_IT()"para habilitar o recebimento por interrupção;
- "HAL_UART_RxCpltCallback()"para executar rotinas/tarefas a cada recebimento de dados.

O método escolhido foi definir o recebimento *byte* a *byte* (*buffer* de tamanho 1), criando uma máquina de estados para verificação de cabeçalhos e identificadores e checagem de dados (se estão de acordo com a quantidade n informada no *byte* de tamanho de dados). Quando a mensagem passa pela checagem, ela é armazenada em um buffer para gravação no cartão SD.

A fim de realizar uma integração do sistema (STM32F769 com ESP32), foi realizada uma conexão elétrica de acordo com o diagrama ilustrado na Figura 26, onde pode-se observar a linha vermelha ligando um barramento de 5V de alimentação e a linha preta ligando um barramento de GND (ambos providos da placa STM32F769) ao ESP32 com a intenção de possuir uma única alimentação do sistema (provida da porta USB). Ainda, há a linha verde que liga o

pino TX2 transmissor da comunicação serial do ESP32 ao pino RX6 receptor do STM32F769. As linhas em amarelo destacam os conectores utilizados na placa DISCOVERY e abaixo estão ilustrados os pinos utilizados e os identificadores destes conectores.

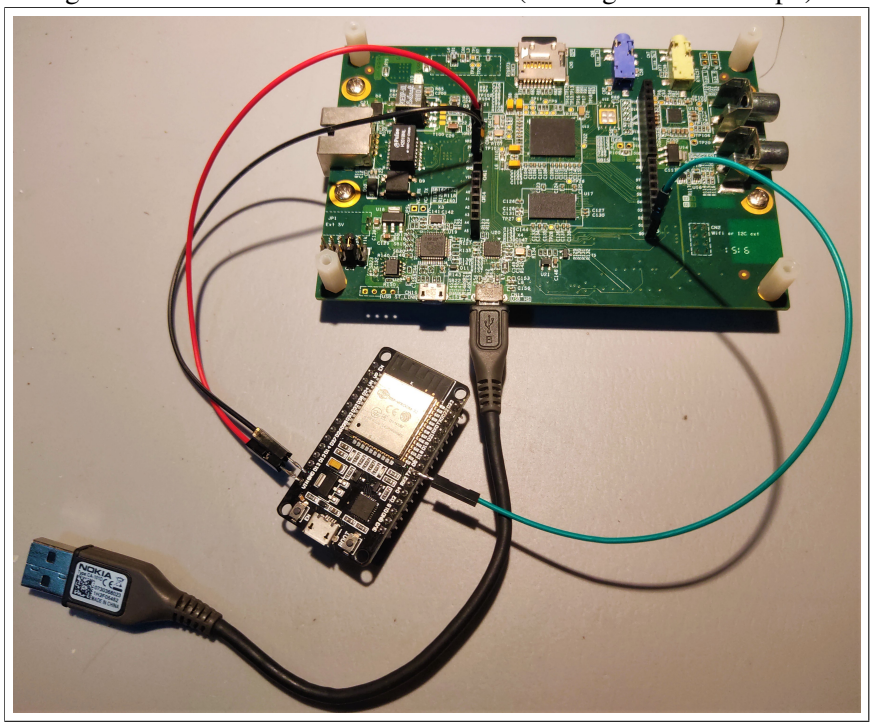
Figura 26 – Diagrama de Conexão do Sistema (Protótipo)



Fonte: O Autor (2022)

Seguindo o diagrama ilustrado na Figura 26, o sistema foi conectado, o que pode ser visualizado na Figura 27 a seguir. Foram utilizados cabos das mesmas cores das linhas ilustradas no diagrama.

Figura 27 – Conexão Prática do Sistema (Montagem do Protótipo)



Fonte: O Autor (2022)

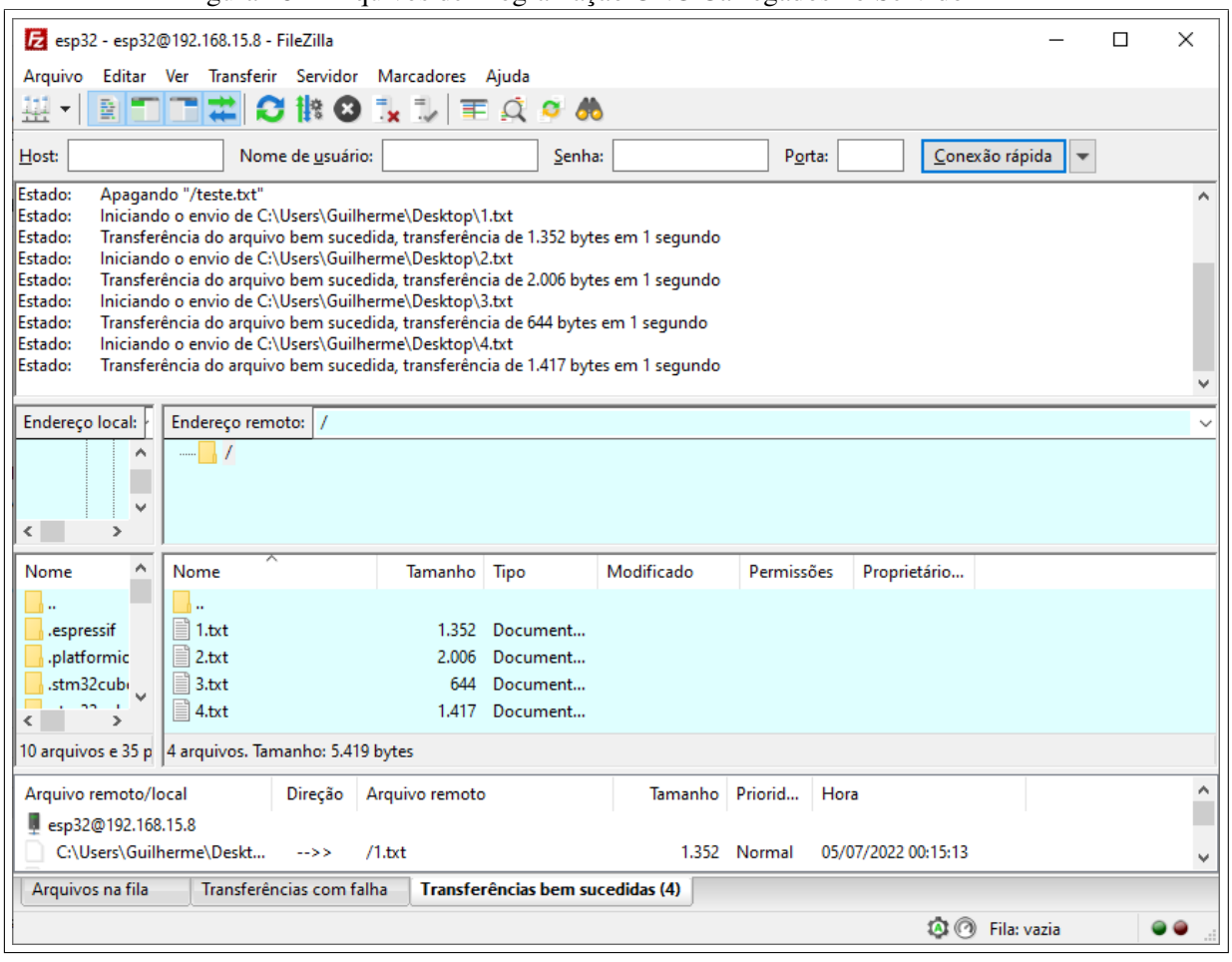
4.8 Implementação da Lógica de Gravação dos Arquivos no Cartão SD

A lógica de gravação no cartão SD dos arquivos recebidos pela comunicação serial ocorre através de uma variável de controle. Esta variável informa que um novo conjunto de mensagens passou na checagem (que há nome e conteúdo de arquivo) e que os dados do buffer que armazena este conteúdo podem ser gravados em um novo arquivo no cartão SD através do uso das funções vistas na seção 4.5. Quando não há mais dados a serem recebidos, o microcontrolador encerra a lógica de gravação e inicializa as funções USB para exibir o dispositivo como um MSC.

5 RESULTADOS

Foi feita a validadação da aplicação conectando o sistema todo à porta USB de uma máquina com Sistema Operacional *Windows*. Em seguida, foi acessado via *wireless* o servidor FTP montado pelo ESP32 através do endereço IP e foram carregados diversos arquivos de texto com exemplos de programas para máquinas CNC. A Figura 28 mostra o servidor com os arquivos:

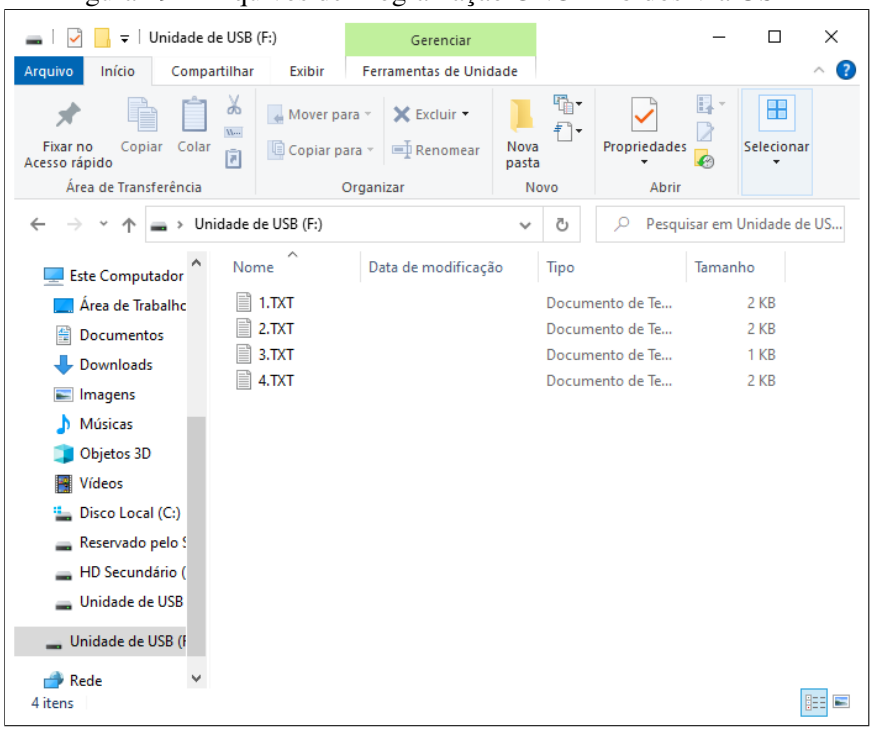
Figura 28 – Arquivos de Programação CNC Carregados no Servidor FTP



Fonte: O Autor (2022)

Após, foi feito o acesso à unidade de disco montada pelo cartão SD na máquina onde está conectado o dispositivo via USB. Foram verificados os arquivos exibidos nesta unidade, conforme mostra a Figura 29:

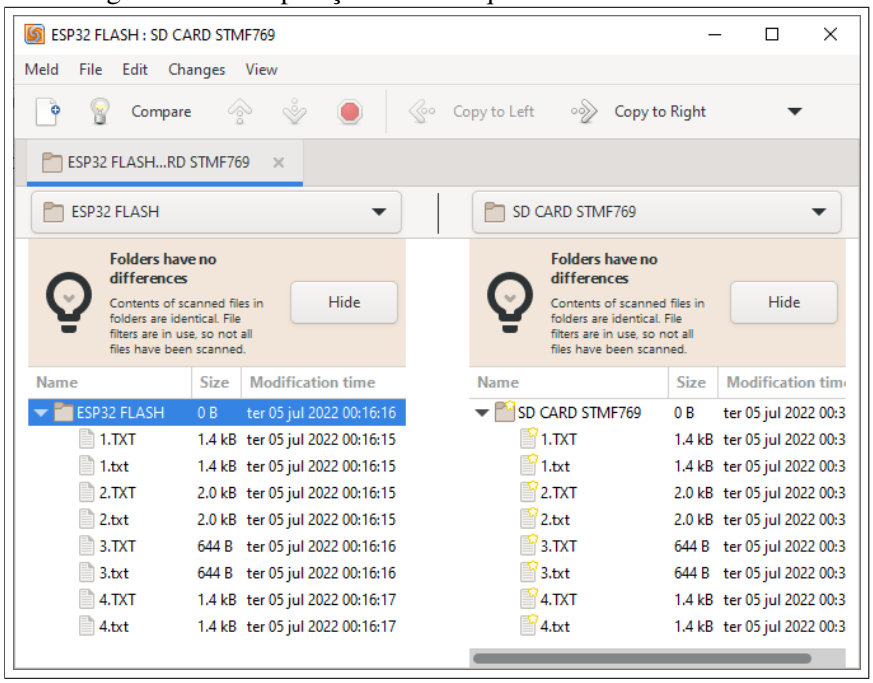
Figura 29 – Arquivos de Programação CNC Exibidos Via USB



Fonte: O Autor (2022)

Em seguida, realizou-se o *download* dos arquivos do servidor FTP e foram copiados os arquivos do cartão SD para duas pastas em disco na mesma máquina. Utilizou-se o *software* de comparação *Meld*. O resultado pode ser visto na Figura 30:

Figura 30 – Comparação Entre Arquivos no Software Meld



Fonte: O Autor (2022)

Pôde-se então observar que os arquivos são idênticos, o que mostra que os arquivos carregados por um computador através de uma conexão *Wireless* para o Servidor FTP criado pelo módulo ESP32 da aplicação foram transmitidos com sucesso para o microcontrolador STMF769, que realizou a escrita no cartão SD alocado na placa DISCOVERY e foi capaz de criar uma funcionalidade do tipo MSC para exibir estes arquivos por meio de uma conexão física USB com uma máquina.

Os códigos fonte principais de cada módulo da aplicação estão nos Apêndices A e B, bem como os links para os arquivos da aplicação completa disponíveis no servidor *Gitlab*.

6 CONSIDERAÇÕES FINAIS

O presente trabalho mostrou que é possível utilizar recursos existentes para construir uma solução que atenda à demanda exposta na seção 1.1. Há diversos caminhos para desenvolver uma aplicação como esta e muitas decisões foram tomadas com base na afinidade do autor com o uso de determinadas tecnologias ou em razão do acesso ao uso das mesmas. Também foi observado neste trabalho que a maior parte do desenvolvimento com o sistema proposto (envolvendo microcontroladores) se deu em torno da criação de códigos de *firmware*. Inclusive, houve obstáculos enfrentados quanto à compatibilidade das bibliotecas de *firmware* fornecidas pelo fabricante de um dos itens utilizados, sendo necessário realizar testes relativos a uma mesma funcionalidade em versões anteriores à instalada por padrão (última versão disponível atualmente).

O dispositivo desenvolvido volta-se para o uso em computadores e máquinas CNC, mas pode ser utilizado ou servir como base para qualquer outra aplicação que tenha as mesmas características, visto que foi baseado em um sistema operacional disponível em computadores.

Como o desenvolvimento foi feito com caráter de protótipo, sugere-se para trabalhos futuros desenvolver uma solução em *hardware* com caráter comercial, inclusive uma aplicação que utilize somente um MCU para controle do sistema, ou então uma aplicação onde ambos MCUs acessem e utilizem uma única memória, já que na aplicação desenvolvida o tamanho de armazenamento fica limitado ao tamanho da *Flash* do ESP32. Sugere-se ainda uma aplicação onde sejam feitas interações bidirecionais ou no sentido contrário (onde seja possível editar arquivos via USB e disponibilizá-los num servidor FTP) do obtido neste trabalho. Também é possível explorar as funções das bibliotecas FTP para realizar ações no dispositivo a cada interação do usuário com o Servidor.

REFERÊNCIAS

ACCARDI, A.; DODONOV, E. Automação residencial: elementos básicos, arquiteturas, setores, aplicações e protocolos. **Tecnologias, Infraestrutura e Software**, [S.l.], p. 156, Nov. 2012.

AXELSON, J. **USB Complete – The developer's guide**. 4. ed. Madison: LakeView Research LLC, 2009. 529 p.

BIEGELMEYER Ânderson. **Internet Of Things – Desenvolvimento e Aplicação de uma Casa Inteligente**. Caxias do Sul, 2015. 78 p.

CANICEIRO, D. V. **GRETDAI – Gateway de recolha e tratamento de dados em ambiente industrial**. Coimbra, 2018. 132 p.

CANZIAN, E. **MINICURSO Comunicação Serial – RS232**. Cotia, São Paulo, Brasil: [s.n.], 2009.

CASTAÑEDA, R. G. **Implementación y ejecución de un protocolo de transferencia de archivos (FTP)**. Pereira, 2012. 43 p.

COSTA, R. D. Transferência automática de programas de máquinas CNC. **XXII Encontro Latino Americano de Iniciação Científica, XVIII Encontro Latino Americano de Pós-Graduação e VIII Encontro de Iniciação à Docência - Universidade do Vale do Paraíba.**, [S.l.], p. 6, 2018.

EMBEDDED-WIZARD. **Getting started with STM: stm32f429 d**iscovery. Disponível em: <https://doc.embedded-wizard.de/getting-started-stm32f429-discovery?v=9.00> Acesso em: 22 de jun. de 2021.

MARCICANO, P. J. P. P. **Introdução ao controle numérico**. Disponível em: <http://sites.poli.usp.br/d/pmr2202/arquivos/aulas/cnc.pdf> Acesso em: 01 de dez. de 2021.

MISCHIANTI.ORG. **DOIT ESP32 DEV KIT v1 high resolution pinout and specs**. Disponível em: <https://www.mischianti.org/2021/02/17/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/> Acesso em: 04 de jul. de 2022.

NETO, S. A. A. **Desenvolvimento de um controlador MIDI com sensores de movimento para aplicação em um projeto audiovisual**. Caxias do Sul, 2019. 89 p.

OLIVEIRA, E. L. de. **Sistema de monitoramento de linhas de produção e serviços através da transmissão de vídeo via internet**. Santa Maria, 2004. 53 p.

ROCHA JUNIOR, P. A. S. da. **Máquina de controle numérico computadorizado**. Belém, 2013. 124 p.

ROYO, M. M. **Desarrollo de una pasarela de datos USB-ethernet con raspberry pi**. Cartagena, 2017. 96 p.

SANTOS, C. A. N. dos; RESENDE, F. P. **Desenvolvimento de interface USB e SPI para eletrocardiografo de alta resolução**. Brasília, 2009. 33 p.

SANTOS, E. P. dos. **CAD/CAM/usinagem CNC integrado a engenharia reversa**. Ilha Solteira, 2017. 55 p.

SILVA, A. T. da. **Módulos de comunicação wireless para sensores**. Porto, 2007. 84 p.

SILVA, C. E. O. da et al. Análise e desenvolvimento de aplicativos de monitoramento remoto em sistemas de manufatura. **Anais do XII ENCITA 2006**, [S.l.], p. 8, 2006.

SILVA CAETANO, G. da. **Implementação de firmware para dispositivo de integração entre controladores de temperatura e sistema em nuvem utilizando conexão wi-fi e protocolo MQTT**. Florianópolis, 2021. 70 p.

SILVEIRA, J. L. da. **Comunicação de dados e sistemas de teleprocessamento**. São Paulo: Makron Books do Brasil Editora Ltda – McGraw-Hill Ltda., 1991.

STMICROELECTRONICS. **STM32CubeIDE user guide**. User Manual, UM2609 - Rev 3, Fevereiro de 2021.

WILLIG, A. **An architecture for wireless extension of Profibus**. Roanoke,: IEEE Int. Conf. Ind. Electron., 2003.

ZEMBOVICI, K. C.; FRANCO, M. G. **Dispositivo para aquisição de sinais e controle digital via USB**. Curitiba, 2009. 92 p.

APÊNDICE A - CÓDIGO FONTE PRINCIPAL STM32F769

listings color

```
/* Includes
   ------------------------------------------------------------------*/
#include "main.h"
#include "usart.h"

/* Private typedef
   -----------------------------------------------------------*/
/* Private define
   -----------------------------------------------------------*/
/* Private macro
   -----------------------------------------------------------*/
/* Private variables
   -----------------------------------------------------------*/
unsigned long esp32_limit_time;
unsigned char leds_sd, usb_initctrl;
unsigned char tryMountAgain;

USBD_HandleTypeDef USBD_Device;
FATFS SDFatFs; /* File system object for SD card logical drive */
FIL MyFile; /* File object */
char SDPath[4]; /* SD card logical drive path */

/* Private function prototypes
   -----------------------------------------------*/
static void SystemClock_Config(void);
static void Error_Handler(void);
static void CPU_CACHE_Enable(void);

#define RXBUFFERSIZE 1
#define DATARXBUFSIZE 4096
#define FILENAMESIZE 256

char aRxBuffer[RXBUFFERSIZE];

uint8_t new_file_to_burn;
uint32_t bufindex;
```

```
uint32_t filedatasize;
uint32_t filenamesize;
char filedata_buffer[DATARXBUFSIZE];
char filename_buffer[FILENAMESIZE];
uint32_t brnfiledatasize;
uint32_t brnfilenamesize;
char brnfiledata_buffer[DATARXBUFSIZE];
char brnfilename_buffer[FILENAMESIZE];

typedef enum
{
  receiving_start_filename,
  receiving_size_filename,
  receiving_data_filename,
  receiving_stop_filename,
  receiving_start_filedata,
  receiving_size_filedatamsb,
  receiving_size_filedatalsb,
  receiving_data_filedata,
  receiving_stop_filedata,
  end_of_reception
} state_t;

state_t myState;

/* Private functions
   -----------------------------------------------------------*/

/**
 * @brief Main program
 * @param None
 * @retval None
 */
int main(void)
{
 FRESULT res;                  /* FatFs function common result code
    */
 uint32_t byteswritten;          /* File write/read counts */

 leds_sd=0;
 tryMountAgain=0;
```

```
  usb_initctrl=1;

  /* Enable the CPU Cache */
  CPU_CACHE_Enable();

  /* STM32F7xx HAL library initialization:
      - Configure the Flash ART accelerator on ITCM interface
      - Configure the Systick to generate an interrupt each 1 msec
      - Set NVIC Group Priority to 4
      - Low Level Initialization
    */
  HAL_Init();

  /* Configure the System clock to have a frequency of 216 MHz */
  SystemClock_Config();

  /* Configure LED1 and LED2 */
  BSP_LED_Init(LED1);
  BSP_LED_Init(LED2);

  MX_USART6_UART_Init();
  /* USER CODE BEGIN 2 */

  HAL_UART_Receive_IT(&huart6, (uint8_t *)aRxBuffer, RXBUFFERSIZE);
  new_file_to_burn = 0;
  myState=receiving_start_filename;

  esp32_limit_time=5000;

  /*##-1- Link the micro SD disk I/O driver
     ##################################*/
  if(FATFS_LinkDriver(&SD_Driver, SDPath) == 0)
  {
//     /* Create FAT volume */
//     res = f_mkfs(SDPath, 0, 4096);
//     if (res != FR_OK)
//     {
//       Error_Handler();
//     }
    /*##-2- Register the file system object to the FatFs module
       ###############*/
```

```
if(f_mount(&SDFatFs, (TCHAR const*)SDPath, 0) != FR_OK)
{
  /* FatFs Initialization Error */
  tryMountAgain = 1;
}
/* Run Application (Interrupt mode) */
while (1)
{
  if(tryMountAgain)
  {
    /*##-2- Register the file system object to the FatFs module
       ##############*/
    if(f_mount(&SDFatFs, (TCHAR const*)SDPath, 0) != FR_OK)
    {
      /* FatFs Initialization Error */
      tryMountAgain = 1;
    }
    else
    {
      tryMountAgain = 0;
    }
  }
  if(!tryMountAgain)
  {
    if (!esp32_limit_time)
    {
      if(usb_initctrl)
      {
        usb_initctrl=0;
        /* Init Device Library */
        USBD_Init(&USBD_Device, &MSC_Desc, 0);

        /* Add Supported Class */
        USBD_RegisterClass(&USBD_Device, USBD_MSC_CLASS);

        /* Add Storage callbacks for MSC Class */
        USBD_MSC_RegisterStorage(&USBD_Device, &USBD_DISK_fops);

        /* Start Device Process */
        USBD_Start(&USBD_Device);
      }
```

```
        }
        else if(myState!=end_of_reception)
        {
          if(new_file_to_burn)
          {
            /* Create and Open a new text file object with write
               access */
            if(f_open(&MyFile, brnfilename_buffer, FA_CREATE_ALWAYS
               | FA_WRITE) == FR_OK)
            {
              /* Write data to the text file */
              res = f_write(&MyFile, brnfiledata_buffer,
                 brnfiledatasize, (void *)&byteswritten);

              if((byteswritten > 0) && (res == FR_OK))
              {
                /* Close the open text file */
                f_close(&MyFile);
              }
            }
            new_file_to_burn=0;
          }
        }
      }
    }
  }
  else
  {
    Error_Handler();
  }
}

/**
  * @brief System Clock Configuration
  *       The system Clock is configured as follow :
  *         System Clock source     = PLL (HSE)
  *         SYSCLK(Hz)              = 216000000
  *         HCLK(Hz)                = 216000000
  *         AHB Prescaler           = 1
  *         APB1 Prescaler          = 4
  *         APB2 Prescaler          = 2
```

```
  *         HSE Frequency(Hz)       = 25000000
  *         PLL_M                   = 25
  *         PLL_N                   = 432
  *         PLL_P                   = 2
  *         PLLSAI_N                = 384
  *         PLLSAI_P                = 8
  *         VDD(V)                  = 3.3
  *         Main regulator output voltage = Scale1 mode
  *         Flash Latency(WS)       = 7
  * @param None
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;

  /* Enable HSE Oscillator and activate PLL with HSE as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_OFF;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 432;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 9;
  if(HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /* Activate the OverDrive to reach the 216 Mhz Frequency */
  if(HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }

  /* Select PLLSAI output as USB clock source */
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_CLK48;
```

```
  PeriphClkInitStruct.Clk48ClockSelection = RCC_CLK48SOURCE_PLLSAIP;
  PeriphClkInitStruct.PLLSAI.PLLSAIN = 384;
  PeriphClkInitStruct.PLLSAI.PLLSAIQ = 7;
  PeriphClkInitStruct.PLLSAI.PLLSAIP = RCC_PLLSAIP_DIV8;
  if(HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /* Select PLL as system clock source and configure the HCLK, PCLK1
     and PCLK2
    clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK |
     RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
  if(HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) !=
     HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief This function is executed in case of error occurrence.
  * @param None
  * @retval None
  */
static void Error_Handler(void)
{
  /* User may add here some code to deal with this error */
  while(1)
  {
  }
}

/**
  * @brief CPU L1-Cache enable.
  * @param None
```

```
  * @retval None
  */
static void CPU_CACHE_Enable(void)
{
  /* Enable I-Cache */
  SCB_EnableICache();

  /* Enable D-Cache */
  SCB_EnableDCache();
}

/**
  * @brief Toggle LEDs to show user input state.
  * @param None
  * @retval None
  */
void Toggle_Leds(void)
{
   static uint32_t ticks;

   if(ticks++ == 100)
   {
     if(leds_sd==2)
     {
       BSP_LED_Toggle(LED1);
       BSP_LED_Off(LED2);
     }
     else if(leds_sd==1)
     {
       BSP_LED_Toggle(LED2);
       BSP_LED_Off(LED1);
     }
     ticks = 0;
   }
}

/**
  * @brief This function handles USART6 global interrupt.
  */
void USART6_IRQHandler(void)
{
```

```
  /* USER CODE BEGIN USART6_IRQn 0 */

  /* USER CODE END USART6_IRQn 0 */
  HAL_UART_IRQHandler(&huart6);
  /* USER CODE BEGIN USART6_IRQn 1 */

  /* USER CODE END USART6_IRQn 1 */
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
{
   switch (myState)
   {
   case end_of_reception:
     break;

   case receiving_start_filename:
     if(aRxBuffer[0]==0xCA)
     {
       esp32_limit_time+=1000;
       myState=receiving_size_filename;
     }
     else if(aRxBuffer[0]==0xC9)
     {
       myState=end_of_reception;
     }
     break;

   case receiving_size_filename:
     filename_buffer[0]='\0';
     if(aRxBuffer[0]<FILENAMESIZE)
     {
       filenamesize = aRxBuffer[0];
       myState=receiving_data_filename;
       bufindex=0;
     }
     else
     {
       myState=receiving_start_filename;
     }
     break;
```

```
case receiving_data_filename:
  if(bufindex<filenamesize)
  {
    if(aRxBuffer[0]=='/')
    {
      filenamesize--;
    }
    else
    {
      filename_buffer[bufindex] = aRxBuffer[0];
      bufindex++;
    }
  }
  else if((bufindex==filenamesize) && (aRxBuffer[0]=='\0'))
  {
    filename_buffer[bufindex] = aRxBuffer[0];
    myState=receiving_stop_filename;
  }
  else
  {
    myState=receiving_start_filename;
  }
  break;

case receiving_stop_filename:
  if(aRxBuffer[0]==0xC6)
  {
    myState=receiving_start_filedata;
  }
  else
  {
    myState=receiving_start_filename;
  }
  break;

case receiving_start_filedata:
  if(aRxBuffer[0]==0xDA)
  {
    myState=receiving_size_filedatamsb;
  }
```

```
    else
    {
      myState=receiving_start_filename;
    }
    break;

  case receiving_size_filedatamsb:
    filedatasize = (aRxBuffer[0] << 8) & 0xFF00;
    myState=receiving_size_filedatalsb;
    break;

  case receiving_size_filedatalsb:
    filedata_buffer[0]='\0';
    filedatasize = filedatasize | (aRxBuffer[0] & 0xFF);
    if(filedatasize<DATARXBUFSIZE)
    {
      myState=receiving_data_filedata;
      bufindex=0;
    }
    else
    {
      myState=receiving_start_filename;
    }
    break;

  case receiving_data_filedata:
    if(bufindex<filedatasize)
    {
      filedata_buffer[bufindex] = aRxBuffer[0];
      bufindex++;
    }
    else if((bufindex==filedatasize) && (aRxBuffer[0]=='\0'))
    {
      myState=receiving_stop_filedata;
    }
    else
    {
      myState=receiving_start_filename;
    }
    break;
```

```
  case receiving_stop_filedata:
    if(aRxBuffer[0]==0xD6)
    {
      new_file_to_burn = 1;
      brnfiledatasize = filedatasize;
      brnfilenamesize = filenamesize;
      memcpy(brnfilename_buffer, filename_buffer,
         brnfilenamesize+1);
      memcpy(brnfiledata_buffer, filedata_buffer,
         brnfiledatasize+1);
    }
    myState=receiving_start_filename;
    break;
  }
  HAL_UART_Receive_IT(&huart6, (uint8_t *)aRxBuffer, RXBUFFERSIZE);
}

#ifdef USE_FULL_ASSERT
/**
  * @brief Reports the name of the source file and the source line
     number
  *       where the assert_param error has occurred.
  * @param file: pointer to the source file name
  * @param line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and
     line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n",
       file, line) */

  /* Infinite loop */
  while (1)
  {
  }
}
#endif
```

Link para acesso à aplicação completa:
<https://gitlab.com/tcc42/tcca>

APÊNDICE B - CÓDIGO FONTE PRINCIPAL ESP32

```
#include <WiFi.h>
#include "Arduino.h"
#include "FS.h"
#include "FFat.h"
#include <SimpleFTPServer.h>

#define DEFAULT_STORAGE_TYPE_ESP32 STORAGE_FFAT
#define RXD2 16
#define TXD2 17

char u_buff[100];
const char* ssid = "user_wifi";
const char* password = "user_password";
int LED_BUILTIN = 2;
FtpServer ftpSrv; //set #define FTP_DEBUG in ESP8266FtpServer.h to
   see ftp verbose on serial

void printDirectory(File dir, int numTabs = 3);
void Tx_New_File(const char* file_name);
void Tx_New_Content(String file_content);

void _callback(FtpOperation ftpOperation, unsigned int freeSpace,
   unsigned int totalSpace)
{
  Serial.print(">>>>>>>>>>>>>>> _callback " );
  Serial.print(ftpOperation);
  /* FTP_CONNECT,
     FTP_DISCONNECT,
     FTP_FREE_SPACE_CHANGE
  */
  Serial.print(" ");
  Serial.print(freeSpace);
  Serial.print(" ");
  Serial.println(totalSpace);

  // freeSpace : totalSpace = x : 360

  if (ftpOperation == FTP_CONNECT) Serial.println(F("CONNECTED"));
```

```
  if (ftpOperation == FTP_DISCONNECT)
     Serial.println(F("DISCONNECTED"));
};
void _transferCallback(FtpTransferOperation ftpOperation, const
    char* name, unsigned int transferredSize)
{
  Serial.print(">>>>>>>>>>>>>>> _transferCallback " );
  Serial.print(ftpOperation);
  /* FTP_UPLOAD_START = 0,
     FTP_UPLOAD = 1,

     FTP_DOWNLOAD_START = 2,
     FTP_DOWNLOAD = 3,

     FTP_TRANSFER_STOP = 4,
     FTP_DOWNLOAD_STOP = 4,
     FTP_UPLOAD_STOP = 4,

     FTP_TRANSFER_ERROR = 5,
     FTP_DOWNLOAD_ERROR = 5,
     FTP_UPLOAD_ERROR = 5
  */
  Serial.print(" ");
  Serial.print(name);
  Serial.print(" ");
  Serial.println(transferredSize);
};

void setup(void)
{
  pinMode (LED_BUILTIN, OUTPUT);
  Serial.begin(115200);
  Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);
  delay(2000);
  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
```

```
  Serial.print(".");
}
Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

/////FTP Setup, ensure FFat is started before ftp; /////////
if (FFat.begin(true))
{
  Serial.println("FFat opened!");

  ftpSrv.setCallback(_callback);
  ftpSrv.setTransferCallback(_transferCallback);

  ftpSrv.begin("esp32", "esp32"); //username, password for ftp. set
     ports in ESP8266FtpServer.h (default 21, 50009 for PASV)
  Serial.println("FTP server started!");
  digitalWrite(LED_BUILTIN, HIGH);
}
else
{
  Serial.println("FFat opened FAIL!!!!!");
}

unsigned int totalBytes = FFat.totalBytes();
unsigned int usedBytes = FFat.usedBytes();
unsigned int freeBytes = FFat.freeBytes();

Serial.println("File sistem info.");

Serial.print("Total space: ");
Serial.print(totalBytes);
Serial.println("byte");

Serial.print("Total space used: ");
Serial.print(usedBytes);
Serial.println("byte");

Serial.print("Total space free: ");
```

```
  Serial.print(freeBytes);
  Serial.println("byte");

  Serial.println();

  // Open dir folder
  File dir = FFat.open("/");
  // Cycle all the content
  printDirectory(dir);
  Serial2.write(0xC9);
}
void loop(void)
{
  ftpSrv.handleFTP(); //make sure in loop you call handleFTP()!!
  // Serial2.write(250);
  // Serial2.print("GUILHERME");
}

void printDirectory(File dir, int numTabs)
{
  while (true)
  {
    File entry = dir.openNextFile();
    if (! entry)
    {
      // no more files
      break;
    }
    else
    {
      Tx_New_File(entry.name());
      File testFile = FFat.open(entry.name(), "r");
      if (testFile)
      {
        Serial.println("Read file content!");
        /**
          File derivate from Stream so you can use all Stream method
          readBytes, findUntil, parseInt, println etc
        */
        String data1 = testFile.readString();
        Serial.println(data1);
```

```
      Tx_New_Content(data1);
      testFile.close();
      delay(200);
     }
     else
     {
      Serial.println("Problem on read file!");
     }
    }
    for (uint8_t i = 0; i < numTabs; i++)
    {
     Serial.print('\t');
    }
    Serial.print(entry.name());
    //   Serial.print("\t Tamanho do nome:");
    //   Serial.print(strlen(entry.name()), DEC);
    if (entry.isDirectory())
    {
     Serial.println("/");
     printDirectory(entry, numTabs + 1);
    }
    else
    {
     // files have sizes, directories do not
     Serial.print("\t\t");
     Serial.println(entry.size(), DEC);
    }
    entry.close();
  }
}

void Tx_New_File(const char* file_name)
{
  unsigned long file_name_size;
  // char user_buff[100];
  // user_buff[0] = 0x01; // 1 em ascii
  // user_buff[1] = 0x01; // linha 1
  // user_buff[2] = '\0';
  //
  // strcat(user_buff, file_name);
  // for (int i = 0; user_buff[i] != '\0'; i++)
```

```
  // {
  //   Serial.print("\nData[");
  //   Serial.print(i, DEC);
  //   Serial.print("] = ");
  //   Serial.print(user_buff[i]);
  // }
  // Serial.print("\n");
  // Serial2.print(user_buff);
  file_name_size = strlen(file_name);
  Serial2.write(0xCA);
  Serial2.write(file_name_size);
  Serial2.print(file_name);
  Serial2.write('\0');
  Serial2.write(0xC6);
}

void Tx_New_Content(String file_content)
{
  unsigned long content_size, content_size_lsb, content_size_msb;
  content_size = file_content.length();
  content_size_lsb = content_size & 0xFF;
  content_size_msb = (content_size >> 8) & 0xFF;
  Serial2.write(0xDA);
  Serial2.write(content_size_msb);
  Serial2.write(content_size_lsb);
  Serial2.print(file_content);
  Serial2.write('\0');
  Serial2.write(0xD6);
}
```

Link para acesso à aplicação completa:
<https://gitlab.com/tcc42/esptcc>