UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

SAMUEL MUNIZ SILVA

PROTOCOLO DE COMUNICAÇÃO MICROCONTROLADO POR STM32 PARA O ACIONAMENTO DE ESTUDOS DE BANCO ENVELHECIMENTO DE IGBT'S

FORTALEZA
2025

SAMUEL MUNIZ SILVA

PROTOCOLO DE COMUNICAÇÃO MICROCONTROLADO POR STM32 PARA O ACIONAMENTO DE ESTUDOS DE BANCOS DE ENVELHECIMENTO DE IGBT'S

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologias da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Paulo Peixoto Praça

FORTALEZA

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S583p Silva, Samuel Muniz.
Protocolo de comunicação microcontrolado por STM32 para o acionamento de estudos de banco de envelhecimento de IGBT's / Samuel Muniz Silva. – 2025.
76 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2025.
Orientação: Prof. Dr. Paulo Peixoto Praça.
Coorientação: Prof. Dr. Domenico Sgró.

1. Conversores de potência. 2. IGBT. 3. Relé. 4. Programação em C. I. Título.

CDD 621.3

SAMUEL MUNIZ SILVA

PROTOCOLO DE COMUNICAÇÃO MICROCONTROLADO POR STM32 PARA O ACIONAMENTO DE ESTUDOS DE BANCOS DE ENVELHECIMENTO DE IGBT'S

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em: ___/___/____.

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Peixoto Praça (Orientador)

Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Domenico Sgrò

Universidade Federal do Ceará (UFC)

_____

Prof. Dr.

Universidade Federal do Ceará (UFC)

“If I have seen further, it is by standing on the shoulders of Giants”

- Newton, Isaac (1676)

AGRADECIMENTOS

Primeiramente, gostaria de agradecer aos meus pais, Franscisco Wellington costa da Silva e Aldenira da costa muniz, pela dedicação, amor e conhecimentos dados a mim durante toda minha vida. Sem eles, nada escrito aqui seria possível. Da mesma forma, agradeço Brenna Géssica Nascimento Silva, minha companheira e esposa, cujo apoio e dedicação me levou até esse momento.

Agradeço a todos os companheiros da Universidade Federal do Ceará (UFC) e da *École Centrale de Lyon*, que compartilharam comigo as angústias e dúvidas desse caminho árduo que foi percorrido por nós durante essa jornada de amadurecimento e aprendizado.

Agradeço aos professores da UFC Dr. Luiz Fernando Antunes, Dr. Domenico Sgró, Dr. Paulo Praça e Dr. Demercil de Souza por todo o conhecimento dado e todas as oportunidades oferecidas para minha formação como engenheiro.

Gostaria de agradecer aos meus tutores de estágio de aplicação na França no *Institute SuperGrid*, Sr. François Wallart, gerente de subprograma, e Sr. Diego Velazco, aluno de doutorado no laboratório *Ampère*, por me dar a oportunidade de realizar meu sonho de trabalhar em uma empresa de alta tecnologia para as energias do futuro. Também gostaria de agradecer a toda a equipe do P3 pelo ambiente de trabalho amigável e profissional que me ofereceram.

Gostaria de agradecer aos meus tutores de estágio de fim de estudos na França no *EDF Paris-Saclay*, Sr. Mohammed Amine El Makhroubi, engenheiro de sistemas, e Sra. Elisa Tejeda, doutoranda na EDF, por me dar a oportunidade de realizar meu sonho de trabalhar em uma empresa de alta tecnologia para as energias do futuro. Também gostaria de agradecer a toda a equipe do *Département de systèmes* pelo ambiente de trabalho amigável e profissional que me ofereceram.

RESUMO

O trabalho mostra o desenvolvimento e resultados do projeto de acionamento de barramentos de conversores CC-CA por meio de um protocolo de comunicação microcontrolado pelo módulo de desenvolvimento STM32F334R8, dentro do contexto redes de transmissão em corrente contínua (CC). Primeiramente, será apresentado um resumo do contexto do projeto que deu origem a esse trabalho juntamente com os objetivos que desejam ser atendidos durante o desenvolvimento desse projeto através da utilização dos recursos disponibilizados pelo microcontrolador STM32. Em segundo lugar, serão apresentados os equipamentos utilizados durante a pesquisa, nessa secção serão enumerados e explicados resumidamente com dados técnicos do *datasheet* os equipamentos utilizados para desenvolver tal sistema como controladores, transistores, cabos, chaves de contato mecânico, etc. Em seguida, apresenta-se os objetivos de uma forma mais específica a serem alcançados ao final do projeto. O desenvolvimento do trabalho mostra quais foram as soluções utilizadas para alcançar os objetivos e seus resultados com resumidas explicações dos porquês dos seus êxitos e falhas por meio de validações isoladas das soluções propostas. Como também, serão analisados os resultados para a validação da solução final escolhida, por meio da integração do sistema de comunicação com simulador em tempo real *SpeedGoat*. Uma parte da validação será feita por meio da interface do *debugger* do ambiente de desenvolvimento utilizado, a outra parte da validação será feita por meio do modelo de validação desenvolvido no *Simulink Matlab* pelo engenheiro responsável pelo projeto e última validação será feita para analisar os sinais de saída e entrada por meio da tensão e intervalo de tempo. Finalmente, após a validação de cada uma dessas etapas de validação, conclui-se que o protocolo de comunicação proposto atende as expectativas e cumpri todos os seus objetivos simultâneos de implementar a comunicação entre o controlador mestre e o controlador central por meio do protocolo UART, a detectar falhas por falta de comunicação e o tratamento de mensagens inválida.

Palavras - chaves: conversores de potência; IGBT; relé; STM32; programação em C.

ABSTRACT

This work shows the development and results of a system drive project to study the ageing of a bank of Insulated gate bipolar transistors (IGBT) in the context of developing and improving direct current (DC) transmission lines. Firstly, a summary of the context of the project that gave rise to this work will be presented, together with the objectives that are to be met during the development of this project using the resources made available by the STM32 microcontroller. Secondly, the equipment used during the research will be presented. In this section, the equipment used to develop this system, such as controllers, transistors, cables, mechanical contact switches, etc., will be listed and briefly explained with technical data from the datasheet. This is followed by a more specific presentation of the objectives to be achieved at the end of the project. The development of the work shows which solutions were used to achieve the objectives and their results with brief explanations of the reasons for their successes and failures through isolated validations of the proposed solutions. It will also show the results of validating the final solution chosen, by integrating the system with the *SpeedGoat* real-time simulation system. Part of the validation will be carried out using the debugger interface of the development environment used, the other part of the validation will be carried out using the validation model developed in Simulink Matlab by the engineer responsible for the project and the last validation will be carried out to analyze the output and input signals by means of voltage and time interval. Finally, after the final validation, it is concluded that the proposed communication protocol meets expectations and fulfills all its objectives, defined during its development, for communication between the relevant parties and the detection of faults due to lack of communication or invalid messages.

Keywords: power converters; IGBT; relay; STM32; C programming;

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BMS	*Bit* mais significativo
BSV	*Bit* de sinal de vida
CC	Corrente contínua
CA	Corrente alternada
CPU	*Central Processing Unit*
DMA	*Direct Access Memory*
ECL	École Centrale de Lyon
IDE	*Integrated Development Environment*
IO	*Input and Output*
ISR	*Interrupt service routine*
FIFO	*First In, first out*
FPGA	*Field Programmable Gate Array*
GPIO	*General Purpose Input Output*
HAL	*High Abstraction Layer*
IGBT	*Insulated-Gate Bipolar Transistor*
LED	*Light Emitter Diode*
MOSFET	*Metal–oxide–semiconductor field-effect transistor*
RDR	*Receiving Data Register*
UART	*Universal Asynchronous Receiver Transmitter*
USART	*Universal Synchronous Asynchronous Receiver Transmitter*

LISTA DE SÍMBOLOS

%	Porcentagem
Ω	Ômega
A	Ampére
R	Resistência
C	Capacitância
I	Corrente elétrica
V	Tensão elétrica
Hz	Frequência
π	Número Pi
T	Constante de tempo
L	Indutância

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

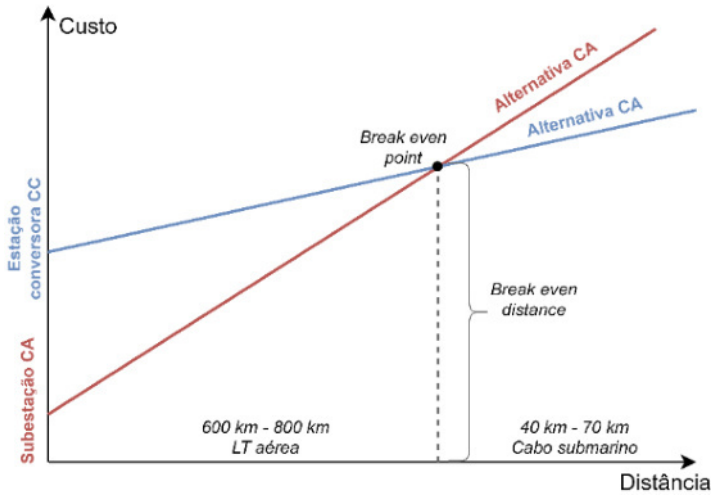
As mudanças climáticas geram um grande impacto na nossa sociedade e meios de produção. Por esse motivo, diversos países buscam realizar uma transição das fontes de energia menos renováveis e liberadores de gás carbônico que fizeram parte do contexto mundial por mais de um século para fontes de energia menos poluentes e mais renováveis como energia eólica e solar.

Um dos grandes desafios na produção de energia elétrica pelos meios mencionados são as perdas do seu processo de extração de uma fonte natural até a chegada do consumidor final. Mais precisamente para o desafio do transporte da energia elétrica desde a unidade produtora até as subestações de média tensão, existe uma grande perca de energia em linhas de distribuição de corrente CA (Corrente Alternada) de longas distâncias na ordem de 1000 Km.

As perdas de energia das linhas CA de longas distância são relacionadas ao efeito corona (descargas elétricas ao redor dos condutores), as perdas relacionadas à reatância indutiva e capacitiva e ao efeito pelicular. Para mitigar esses problemas, é proposta a utilização de linhas de distribuição em corrente CC (Corrente Contínua) devido à ausência de perdas reativas, redução das perdas por efeito Joule e maior capacidade de transmissão por quilômetro.

A figura 1 mostra um gráfico que evidencia uma preferência econômica e logística em utilizar linhas de distribuição CC em longas distâncias.

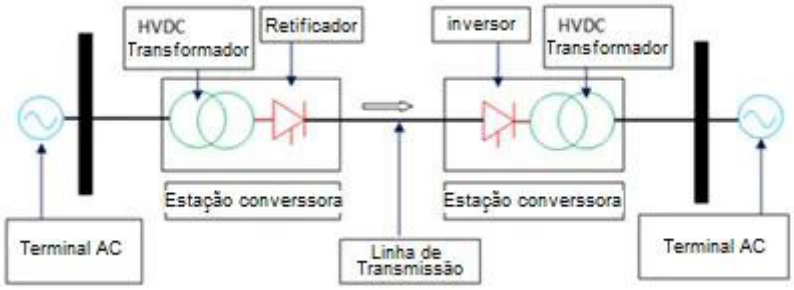
Figura 1 - Comparativo de custo entre linhas de transmissão CC e AC



Fonte: Cepel. Disponível em: https://see.cepel.br/ Acesso em: 24 dez. 2024.

A transmissão de energia por linhas CC é fortemente baseada na utilização de conversores de potência. Em resumo, a energia é extraída da natureza na forma inicial de CA e é retificada por meio de conversores AC/CC para então ser transportada por de linhas CC. Na subestação de média tensão ela é convertida novamente em CA por meio de inversores (conversores CC/CA) para então chegar ao consumidor da mesma forma que uma linha CA, mas com um rendimento maior. A figura 2, ilustra esse processo resumido de conversão das linhas CC.

Figura 2 - Esquema de uma linha de transmissão CC

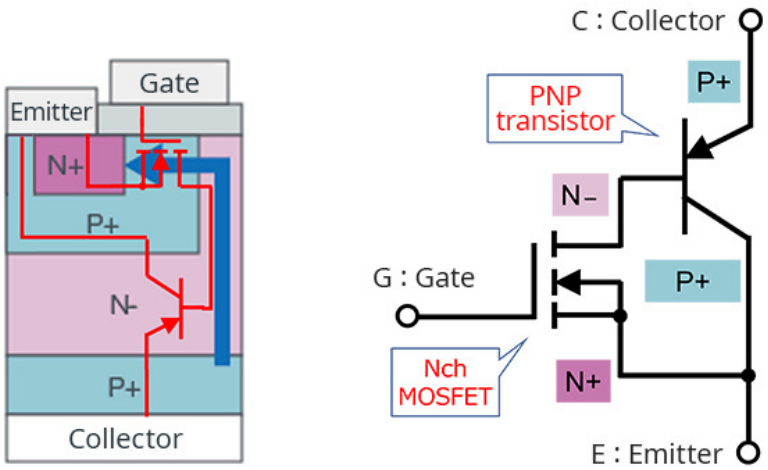


Fonte: ANDRADE, Anderson Luiz de. Hvdc Transmission System - High Voltage Current (Modular Multilevel Converters - MMC).

Nos sistemas de conversão de energia, os IGBTs (*Insulated Gate Bipolar Transistors)* são componentes essenciais para a comutação de altas correntes de forma rápida e eficiente. No entanto, esses dispositivos estão sujeitos a fenômenos de envelhecimento que podem afetar seu desempenho e reduzir sua vida útil, afetando assim a confiabilidade dos conversores de energia.

Os transistores IGBT são dispositivos semicondutores amplamente utilizados em aplicações no campo da eletrônica de potência devido à sua capacidade de combinar as características de alta velocidade de comutação dos transistores controlado por tensão elétrica como os MOSFET (m*etal-oxide semiconductor field-effect transistor*) e as de baixa perda de condução dos transistores controlados por corrente elétrica como os TBJ (transistores bipolares junção). Essa combinação os torna ideais para operar em sistemas de conversão de energia e em outras aplicações do campo da eletrônica de eletrônica de potência. A figura 3 ilustra a construção de um IGBT de canal N como a união de um MOSFET de canal N com um TBJ do tipo PNP.

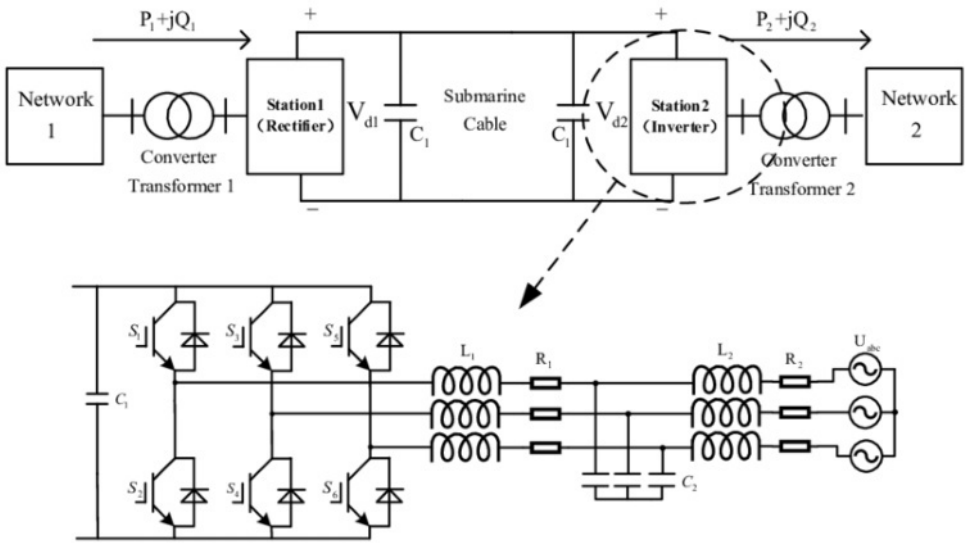
Figura 3 - Junções PN de um transistor IGBT



Fonte: Techweb.com Acesso em: 24 dez. 2024

No contexto das redes HVDC (*High Voltage Direct Current*), os transistores IGBT's desempenham um papel importante dentro dessas redes de transmissão, principalmente nos conversores trifásicos CC/CA. Os IGBTs são então utilizados em conversores de potência que realizam a conversão da energia de CA/CC e de CC/CC. Esses conversores de potência que fazem uso desses componentes para essa aplicação são denominados VSC-HVDC (*Voltage Source Converter High Voltage Direct Current*). A figura 4 ilustra um exemplo topologia de aplicação desenvolvida por Wang et al. (2019) como a utilização de IGBTs em inversores trifásicas para a conversão CC/CA.

Figura 4 - Circuito de linhas de transmissão CC com conversores a IGBT's



Fonte : LI, Lie et al. Microelectronics Reliability, v. 122, p. 114165, 2021

Em geral esses componentes são utilizados principalmente em aplicações como conversão para o processo de Retificação, onde eles são usados para converter a corrente alternada gerada em estações de energia ou redes elétricas em corrente contínua, que é transmitida por longas distâncias através de linhas de transmissão. Da mesma forma no processo inverso na Inversão na estação receptora, onde os IGBTs são usados para a conversão CC/CA, para que possa ser distribuída para as redes de energia locais. Eles também podem ser utilizados em diversas topologias de conversores de CC/CC para controle em malha fechada da tensão da linha de transmissão na saída dos retificadores e na entrada dos inversores, uma vez que podem existir perturbações na rede.

A utilização de IGBTs em HVDC traz diversas vantagens como a comutação independente da rede, o que permite um maior controle e flexibilidade, a capacidade de operar com modulação por largura de pulso (PWM) para reduzir distorções harmônicas, a possibilidade de controlar de forma independente potência ativa e reativa, sua compatibilidade com redes multi-terminais e integração de fontes renováveis. Portanto, IGBTs são mais adequados para sistemas HVDC modernos, baseados em VSC (*Voltage Source Converters*), que exigem maior flexibilidade de controle e integração com fontes renováveis conforme discutido por Boutry et al. (2017).

Com a ampla utilização dos módulos IGBT, a confiabilidade desses módulos torna-se crucial em sistema HVDC, por esse motivo o estudo dos fenômenos de falha e a previsão da vida útil dos IGBTs são de extrema importância para a maximização da eficiência da energia produzida em conversores de potência.

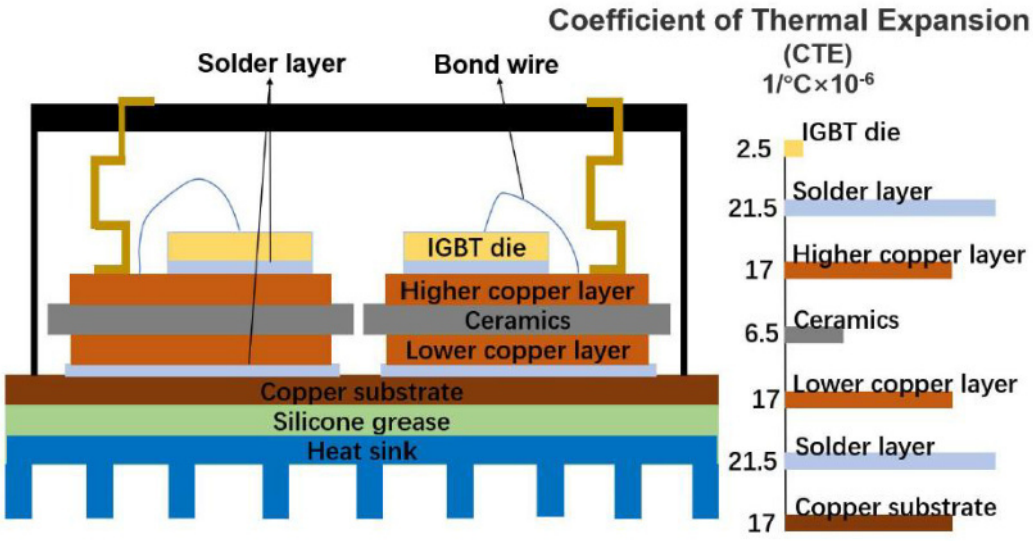
O desenvolvimento de um modelo de vida útil pode contribuir para uma análise mais aprofundada da confiabilidade dos IGBTs. Estimar a vida útil dos IGBTs e orientar a manutenção do sistema são medidas fundamentais para reduzir as falhas e aumentar o tempo efetivo de operação. Um método comum para prever a vida útil do IGBT é o teste de envelhecimento acelerado. Nesse método, o IGBT é submetido a condições de trabalho severas, com correntes e temperaturas de operação elevadas, resultando em um período experimental relativamente curto. Durante a previsão, os dados da amostra são inseridos em uma fórmula de cálculo da vida útil, considerando uma distribuição específica, alguns coeficientes indeterminados e fatores de aceleração, para estimar a vida útil em condições normais de operação.

Segundo Bao e Jiang (2019), o IGBT é composto por várias camadas de diferentes materiais com coeficientes de dilatação diretos. A temperatura de operação do módulo IGBT é

influenciada por vários fatores. À medida que a corrente de carga ou a frequência de comutação aumentam, as temperaturas da junção do IGBT e das diferentes camadas também se elevam. Durante o funcionamento prolongado do IGBT, gera-se uma grande quantidade de calor, o que leva à expansão das camadas estruturais. Como os coeficientes de expansão térmica são diferentes para cada camada, elas se expandem de forma desigual, resultando em deformações, especialmente nas interseções entre dois materiais distintos, onde podem surgir fissuras, cavidades e outras imperfeições que, a longo prazo, podem causar falhas nos módulos IGBT.

Como ilustrado na figura 5, a estrutura de encapsulamento do IGBT e o coeficiente de expansão térmica de cada camada mostram que a diferença entre os coeficientes térmicos dos materiais adjacentes do IGBT e o coeficiente de expansão da camada de solda é a maior, e teoricamente, é onde a deformação causada pelo calor é mais provável de ocorrer. Isso, por sua vez, afeta a vida útil do dispositivo.

Figura 5 - Estrutura interna de transistores IGBT's



Fonte: BAO, Yangjie; JIANG, Quan. 2019 22nd International Conference on Electrical Machines and Systems (ICEMS), 2019.

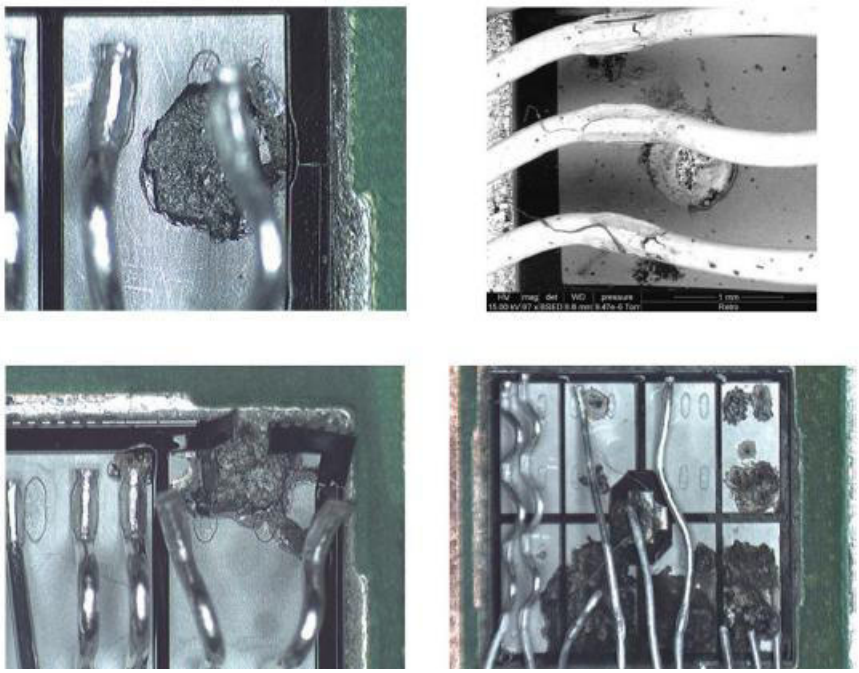
Outro estudo realizado por Smet e Forest (2019) reitera a tese de que os principais efeitos da ciclagem térmica e de energia nos módulos de potência módulos de potência IGBT são bem conhecidos. Esses efeitos são termomecânicos principalmente e dizem respeito principalmente ao conjunto de camadas sob os chips e as conexões. Eles provocam a propagação de fraturas em soldas e nas diferentes camadas de interface dos substratos de cobre

com ligação direta (DBC), mas também geram danos metalúrgicos nas ligações dos fios e na metalização do emissor.

Os conversores de potência usados em aplicações de transporte geralmente passam por modos de ciclagem com diferentes níveis de variação de temperatura e frequência. Em aplicações terrestres ou aéreas, esses conversores operam sob condições de carga intermitente e/ou variável, o que provoca ciclagem de potência. Assim, a faixa de temperatura nos chips normalmente fica entre 30 °C e 60 °C, resultando em vidas úteis superiores a um milhão de ciclos.

Além disso, o módulo de potência pode ser submetido a alguns milhares de ciclos em temperaturas ambientes que variam amplamente. A figura 6 mostra os resultados após os testes de fadiga realizados por Smet e Forest (2019) na solda interna dos IGBTs.

Figura 6 - Desgaste da estrutura interno de transistores IGBT’s



Fonte : SMET, V. et al. Microelectronics Reliability, v. 58, n. 10, 2011.

Portanto, o estudo da fadiga e envelhecimento desses componentes é de extrema importância para a optimização da composição interna dos transistores de tal forma a buscar uma maior eficiência e confiabilidade entre diferentes condições ambientais que variam de -40 °C a -55 °C em regiões frias terrestres e até 120 °C próximo aos motores.

Um outro estudo, conduzido por Li et al. (2021), propõe um novo parâmetro de envelhecimento baseado na teoria cumulativa de fadiga linear de Miner. Esse parâmetro é usado

para caracterizar o estado de envelhecimento do IGBT quando vários modos de falha atuam simultaneamente.

Primeiramente, o fio de ligação e a camada de solda são analisados pelo método de elementos finitos. Em seguida, são analisados de acordo com a carga, o método da curva S-N principal é usado para calcular o acúmulo de fadiga. O maior valor é considerado como o grau de envelhecimento geral do módulo IGBT. Por fim, o método de análise de envelhecimento do IGBT baseado em teoria cumulativa de fadiga linear de Miner sob o modo de falha composta é verificado pela análise de dados.

Esse método não somente pode ser usado para análise de envelhecimento e previsão de vida útil do IGBT, mas também pode prever o modo de falha final do IGBT dentro de um intervalo de erro.

O método considera de forma abrangente os efeitos da concentração de tensão, do modo de carga e da espessura da camada de carga e espessura da camada de material, e tem vantagens significativas na análise de fadiga de estruturas soldadas. A ideia base do método é dividida em 3 partes:

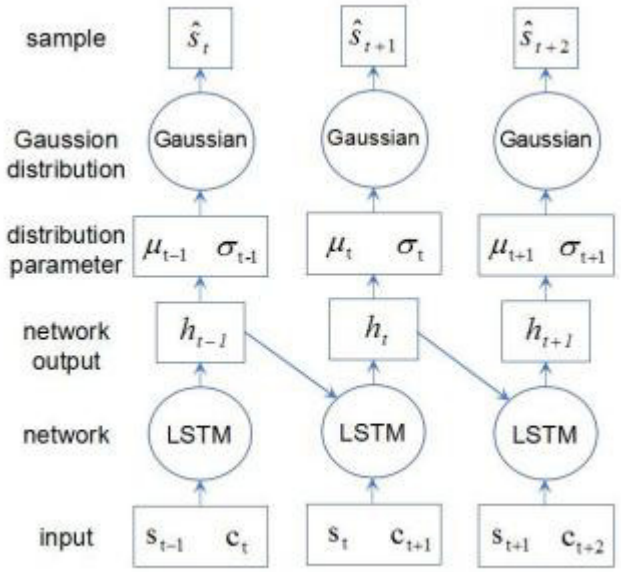
(1) Cálculo da distribuição de tensão da estrutura da superfície de soldagem;
(2) Calcular a distribuição de tensão da estrutura equivalente usando a teoria da mecânica da fratura;
(3) Usar a curva S-N principal para obter a vida útil da camada de solda e do fio de ligação.

Apesar da criação de vários modelos físico que tentam prever o comportamento de transistores IGBT's. A literatura mostra que os modelos baseados na coleta de dados e estruturação de modelos matemáticos vem ganhando e ganhará mais espaço dentro da comunidade científica. Um dos exemplos de estruturação de modelos de dados aplicados a previsão do envelhecimento de IGBT's foi proposto por Ge et al. (2020) pela utilização de um modelo de DeepAR para prever a vida útil restante de módulos IGBT ao combinar as características de chaveamento transiente.

A principal contribuição do trabalho "*RUL Predict of IGBT Based on DeepAR Using Transient Switch Features*" é a combinação de características extraídas durante o funcionamento normal do sistema, que são capturadas durante os períodos de chaveamento transiente do IGBT. O modelo DeepAR, ilustrado pela figura 7, é treinado para identificar padrões nesses dados e prever falhas antes que ocorram, ajudando a otimizar os ciclos de

manutenção. Além disso, a utilização do filtro de Kalman é explorada para melhorar a precisão das predições, com o objetivo de aumentar a confiabilidade do sistema e reduzir os custos associados à manutenção corretiva.

Figura 7 - Esquema de aprendizagem do modelo DeepAR



Fonte: GE, Jianwen et al. Predict of IGBT Based on DeepAR Using Transient Switch Features.

Outro ponto relevante do artigo é a adoção do modelo Hefner, que é aplicado para modelar o comportamento de degradação do IGBT ao longo do tempo. Essa abordagem integrada visa melhorar a precisão das estimativas de RUL e oferece uma ferramenta poderosa para o diagnóstico e manutenção de sistemas de potência eletrônica.

1.2 Projeto e objetivos gerais

Nesse contexto de transmissão CC com transistores IGBTs, o *Institute SuperGrid* é uma empresa francesa instalada na cidade de Villeurbanne (metrópole de Lyon) voltada para o desenvolvimento de linhas de transmissão CC com o objetivo de auxiliar a Europa na transição energética. Uma das linhas de pesquisas do departamento de eletrônica de potência é o estudo e modelização do envelhecimento dos transistores IBGT's por meio de um banco de transistores comandado pelo controlador em tempo real *SpeedGoat*.

Para a implementação do estudo é necessário um sistema de acionamento para permitir o início do teste e fim dos testes. Para isso, esse trabalho tem como objetivo desenvolver a rotina

de comunicação da placa de acionamento, conhecida como “placa relé”, responsável por receber, validar, implementar os comandos enviados e reenviar os estados atuais dos contatos do barramento CC para o *SpeedGoat* que atual como controlador mestre, garantindo a segurança das operações e a integridade dos dados coletados e de todo o equipamento utilizado durante o ensaio.

Para a criação desse protocolo de comunicação, é necessário que esse sistema de acionamento atinja esses objetivos a seguir:

- Receber mensagens com comandos a serem executados;
- Validar a mensagem recebida para evitar acidentes;
- Extrair os comandos contidos na mensagem;
- Ler os estados dos relés e retornar esses estados de forma a garantir a integridade dessas informações;
- Verificar continuamente a integridade da comunicação, a fim de detectar uma falha;
- Realizar essas tarefas no menor tempo possível para evitar o acúmulo de dados.

1.3 Metodologia

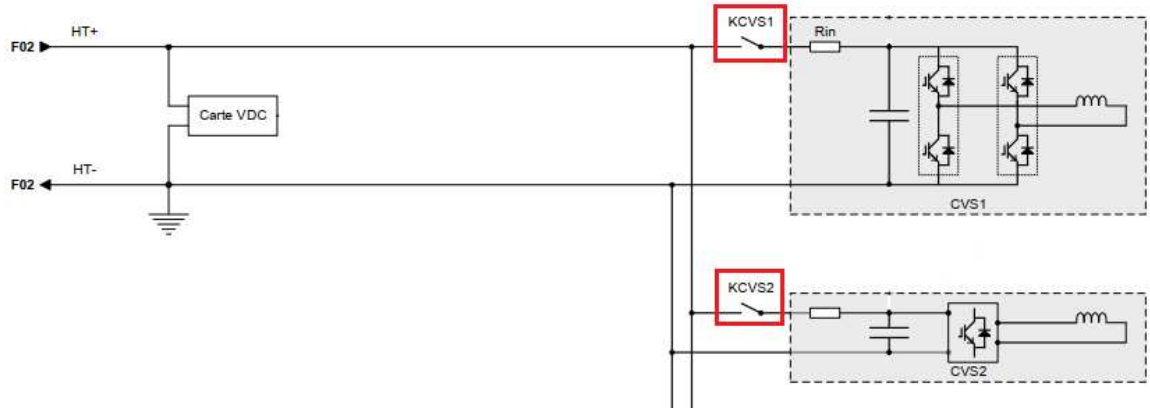
Para atingir esses objetivos, será necessário usar vários recursos da linguagem C aplicados a sistemas incorporados, como a criação de temporizadores para fazer a contagem regressiva do tempo, a execução de comandos por meio da saída digital do controlador, o uso de ADCs (*Analog Digital Converters*) para ler o estado dos relés, o uso de interrupções de temporizador para verificar a conexão entre os dispositivos, compreensão da função dos registros (espaços usados no microcontrolador para armazenar dados ou informações de configuração) em microprocessadores e estudo de *buffers* (espaços de memória usados para armazenar temporariamente dados de informações no microcontrolador) para comunicação a fim de criar protocolos de comunicação mais rápidos, eficientes e seguros.

Após serem realizadas verificações parciais das soluções propostas para os problemas propostos de comunicação e de detecção de erros. Posteriormente, validações gerais devem ser realizadas para atestar a integridade e sincronismo das duas faces da solução final sobre a óptica dos algoritmos e sinais recebidos e enviados.

2 BANCO DE ENVELHECIMENTO DE TRANSISTORES IGBT's

O banco de envelhecimento de transistores IGBT's é dividido em duas topologias para os estudos de envelhecimento. A primeira são os conjuntos de transistores conectados em configuração de inversor monofásico CC/AC ponte H a uma carga indutiva, como mostrar a figura 8. O segundo tipo de topologia são transistores operando como chave ligados ao barramento CC por um filtro RC e a uma carga indutiva na saída. O sistema em maior parte é comandado pelo controlador mestre *SpeedGoat* e possui vários sistemas auxiliares para realizar diversas tarefas como monitorar tensão no barramento CC, monitorar temperatura de junção, etc. Contudo, esse trabalho se dedica ao desenvolvimento da "Placa relé", cujo objetivo é monitorar e controlar as chaves de contato que conectam o barramento CC aos conversores do banco de transistores, como já mostra a figura 8 onde os retângulos vermelhos indicam as chaves de contatos do barramento a serem controladas.

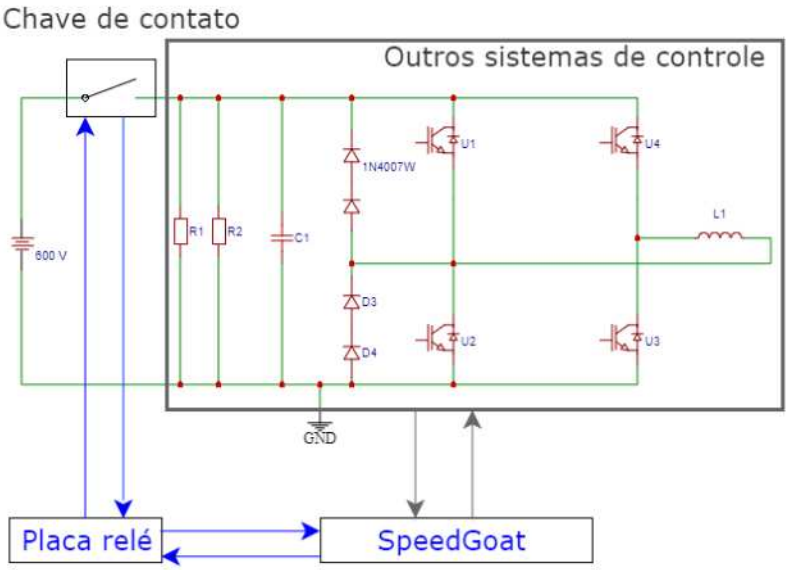
Figura 8 - Esquema do banco de IGBT's e seus contatos de acionamento



Fonte: Institute Supergrid (2021) (Modificada)

A figura 9 exemplifica a relação de dependência entre a placa relé e o *SpeedGoat* para atuar nas chaves de contato de uma instância do banco de transistores. Em resumo, a placa relé é responsável por receber os comandos de abertura e fechamento do barramento por meio de um pacote de mensagem enviado pelo controlador *SpeedGoat* e executá-los, para em seguida, ler os estados dos mesmos e retornar esses dados em forma de mensagem ao controlador central e assim sucessivamente.

Figura 9 - Esquema de relação entre controladores e chaves de contato



Fonte: O próprio Autor

2.1 Transistores IGBTs do banco de envelhecimento

O módulo Infineon FF150R12KE3G foi a escolha de transistor IGBT no estudo do envelhecimento. Um módulo excelente para a aplicação em sistemas HVDC devido a uma série de características técnicas. Como essas apresentadas pela tabela 1.

Tabela 1 - Dados *datasheet* do módulo Infineon FF150R12KE3G

Característica	Valor
Corrente Nominal de operação (Ic)	150 A
Tensão de bloqueio (Vces)	1200 V
Corrente de pico de pulso	300 A
Tensão nominal Vce (Polarização)	2,15 V
Energia de comutação (Polarizado)	8,6 mJ
Energia de comutação (Despolarizado)	3,8 mJ

Fonte: Datasheet Infineon FF150R12KE3G, Infineon technologies (2019)

Essas características são adequadas as redes de transmissão CC, pois são sistemas susceptíveis a surtos e variações nas correntes durante a operação. O módulo FF150R12KE3G possui também um sistema de isolamento robusto que protege os componentes internos, garantindo maior confiabilidade em ambientes de alta tensão. O encapsulamento bem projetado

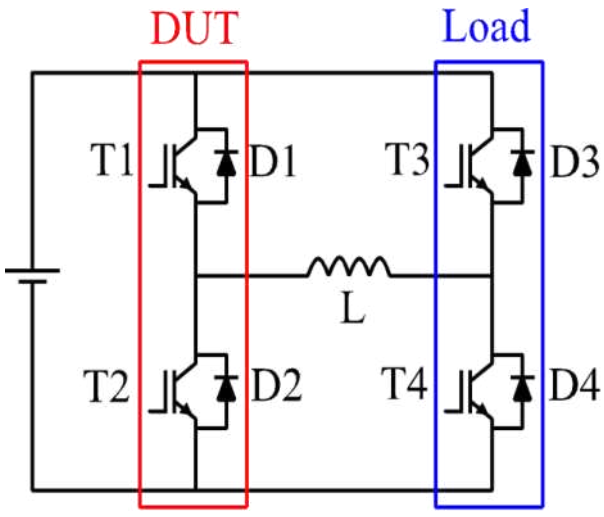
facilita a dissipação de calor e simplifica a instalação em sistemas de refrigeração, como dissipadores de calor e coolers líquidos, assegurando que o módulo opere de maneira estável mesmo sob altas demandas.

Figura 10 - Módulo IGBT Infineon FF150R12KE3G

Fonte: Infineon technologies (2019)

Os testes realizados sobre os IGBT's em uma topologia ponte H com uma carga indutiva L para a implementação de um banco de ensaio de *AC Power Cycling*. É constituída por uma ponte H com uma carga indutiva, como mostra a figura 11. Nesta topologia, apenas um dos ramos será sujeito a ensaios de envelhecimento, o ramo do DUT. O ramo LOAD será utilizado para produzir as formas de onda de corrente necessárias para impor condições de ciclo térmico no ramo DUT. Estes ensaios de envelhecimento são efetuados a uma temperatura de junção constante em torno de 110°C.

Figura 11 - Esquema do conversor do banco de envelhecimento



Fonte: Institute Supergrid (2021)

2.2 Chaves de contato

As chaves de contato possuem o papel de controlar o ensaio, pois são por meio deles que os ensaios do banco de transistores serão iniciados ou interrompidos, ou seja, o barramento DC do banco de transistores será aberto ou fechado. Como trata-se de um contexto de alta potência o banco de IGBT's precisam de contatos com tempo de resposta de menos de 50 ms, para evitar ou mitigar qual dano ao equipamento em uma possível falha, uma corrente de operação na ordem de 30 A e uma tensão de operação na ordem de no mínimo 1600 V. Por esse motivo, foi escolhido o contato com o nome de referência LTC002501*A02, ilustrado pela figura 12.

Figura 12 - LTC002501*A02

Fonte: Widap (2016)

A chave de contato modelo LTC002501*A02 é amplamente utilizada em sistemas elétricos devido à sua confiabilidade e eficiência. Projetada para aplicações de média e alta tensão, destaca-se pela robustez e durabilidade, garantindo desempenho estável em condições extremas. Possui capacidade de manobra sob carga, permitindo operação segura e contínua. De acordo com o datasheet do equipamento, o contato tem a capacidade de conduzir uma corrente contínua de aproximadamente de 30 A em operação nominal. O contator LTC002501*A02 tem as seguintes características mostrado na tabela 2:

Tabela 2 - Dados LTC002501*A02

Característica	Valor
Tensão de alimentação das bobinas V_{DC}	24 – 36 – 48 – 72 - 110
Número de contatos auxiliares	2(1NO + 1NC)
Número de polos	1 Polo NO
Tensão máxima de funcionamento $[V_{DC}/V_{AC}]$	2000V
Tensão de impulso máxima	12kV
Poder de interrupção de corrente DC máximo (t=5ms) [A]	130 A em 900V
Constante de tempo $\tau = \frac{L}{R}\ (ms)$ Fechamento / abertura	$\tau = [25ms; 50ms]$
Potência consumida pela bobina ($T = 20°C$)	32 W

Fonte: Datasheet LTC002501*A02, WIDAP (2016)

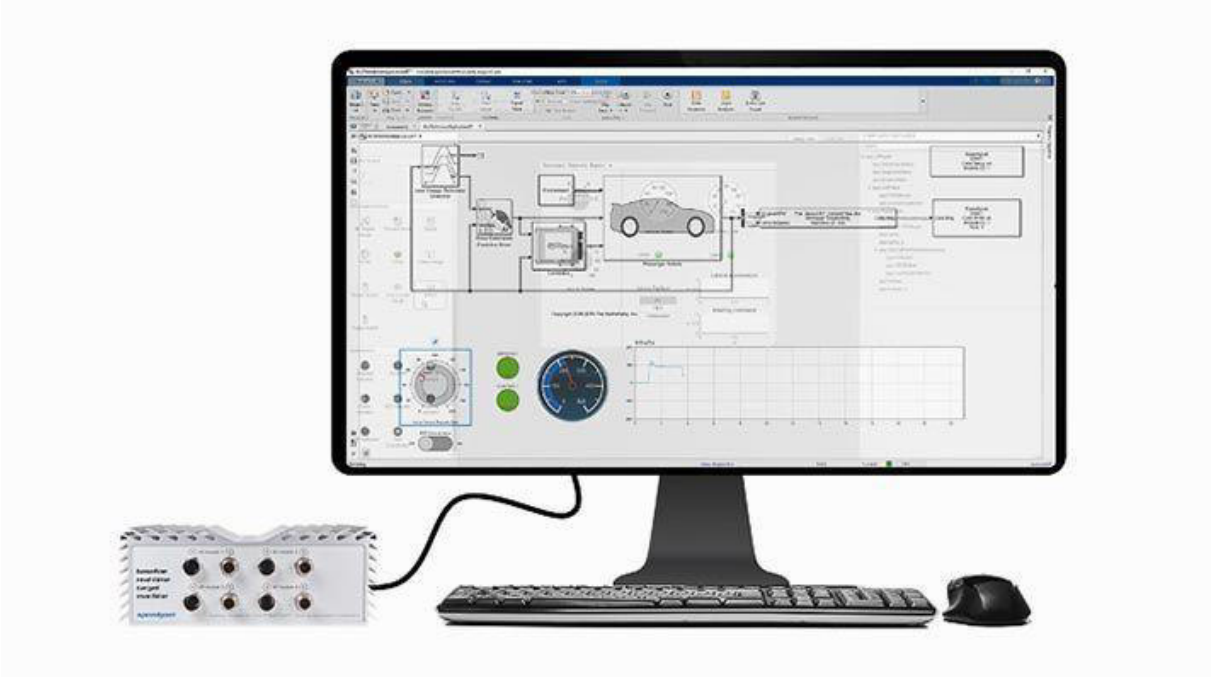
2.3 O controlador mestre *SpeedGoat*

O controlador *SpeedGoat* é um controlador em tempo real responsável pela leitura e controle dos dados de outros periféricos e sensores. O *SpeedGoat* é programado com Simulink/ Matlab para facilitar o desenvolvimento do código gerado automaticamente pelo programa. Algumas das suas características do *SpeedGoat* são:

- 1 - Intel ATOM 1.91 GHz;
- 1 - Placa FPGA Xilinx kintex 325k;
- RAM 2GB;
- SSD 8GB.

Logo, o equipamento demonstra ter um alto poder de processamento em velocidade com um a sua CPU (Intel ATOM 1.91 GHz) e em paralelismo com o FPGA (Xilinx kintex 325k). Apesar da quantidade de memória relativamente pequena de 2GB, essa quantidade já é suficiente para tarefas com poucos threads (sequências de instruções que um computador executa) como é o caso da simulação de modelos físicos.

Figura 13 Interface Simulink para o controle do controlador *SpeedGoat*



Fonte: *SpeedGoat* (2021)

Dessa forma, o principal papel do *SpeedGoat* no sistema de envelhecimento de transistores é simular o sistema físico do banco de IGBT's de forma computacional. Ele utiliza a ferramenta *Mathworks Simulink.* Nessa conjuntura, *o*s softwares da *MathWorks*® são otimizados para garantir o melhor desempenho em tempo real, integração do fluxo de trabalho e usabilidade com as máquinas-alvo *Simulink® Real-Time™* e *SpeedGoat*. O *SpeedGoat* oferece *hardware* projetado para computação em tempo real. Seus sistemas são capazes de executar modelos complexos em tempo real, permitindo que os desenvolvedores interajam com o sistema como se ele fosse o hardware real. Como também, é possível o *hardware SpeedGoat* ser conectado a um controlador físico auxiliar, dessa forma, os desenvolvedores podem testar seus algoritmos em um modelo virtual do sistema. Isso é especialmente útil para validar algoritmos de controle em tempo real sem arriscar o equipamento real, no caso, um conversor CC/CA e seus controladores.

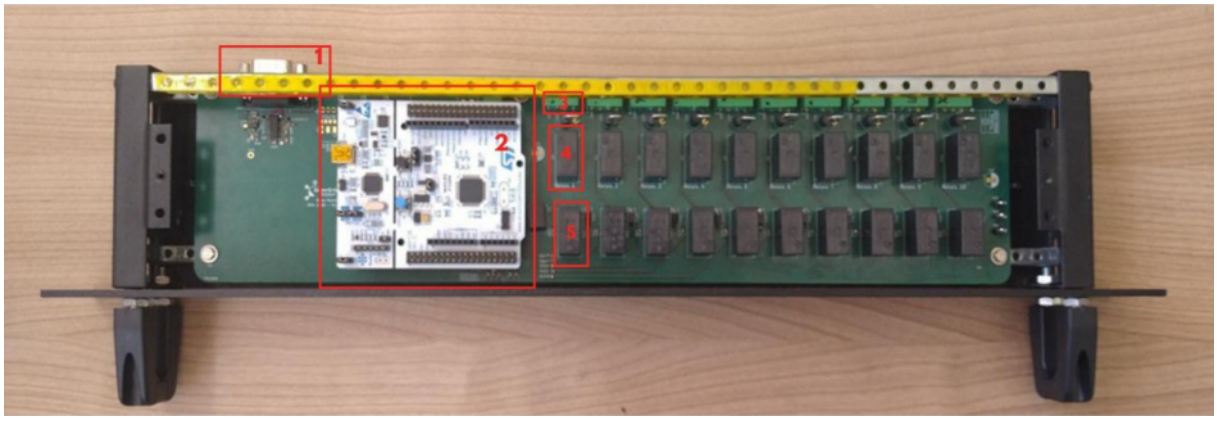
2.4 A placa relé

Como já mencionado anteriormente, a placa Relé é responsável pelo acionamento do barramento CC para dar inícios aos estudos de envelhecimento e pelo monitoramento dos dados trocados entre o seu controlador interno e o controlador mestre (*SpeedGoat*), acionando o estado

de proteção caso ocorra a detecção de um problema com a comunicação. Para isso, ele deve executar as seguintes tarefas:

1. Abrir ou fechar os contatos do barramento;
2. Ler o estado dos contatos;
3. Abrir os contatos em uma falha do controlador mestre *SpeedGoat*;
4. Continuar sua operação caso jumper de comunicação entre ele e o *SpeedGoat* seja desconectado e posteriormente conectado.

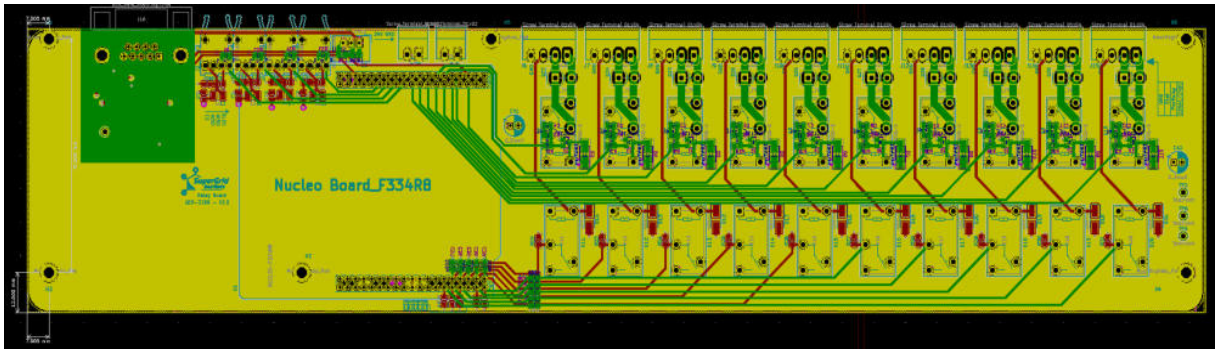
Figura 14 - Placa relé



Fonte: O próprio Autor

A figura 14 destaca 5 elementos que podem ser considerados os atores principais da placa relé. O elemento 1 representa a interface de comunicação RS232 da placa como o módulo *SpeedGoat.* O elemento 2 é a placa de desenvolvimento *NUCLEO* STM32F334R8, responsável pelo processamento de dados recebidos. O elemento 3 são conectores de 4 polos que conectam os relés de comando G5Q-1-EU 24DC a chaves de contato dos barramentos dos conversores de potência do banco de envelhecimento. O elemento 4 é relé de comando G5Q-1-EU 24DC propriamente dito. Finalmente, o elemento 5 são os relés G5Q-1-EU 24DC de releitura dos estados das chaves de contato.

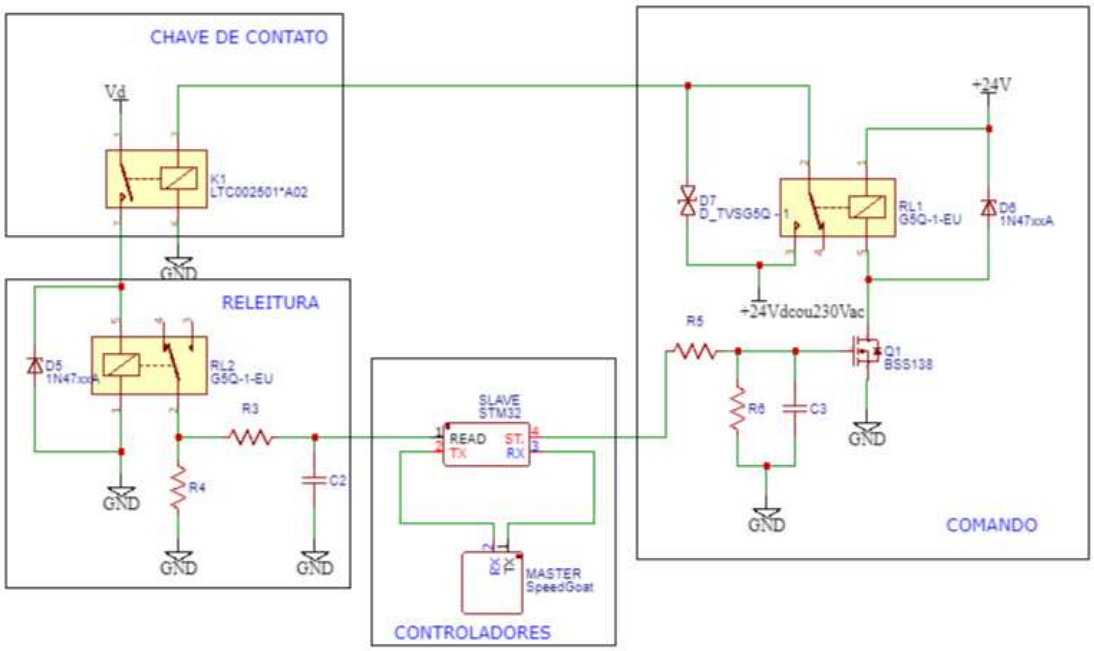
Figura 15 - Esquema das trilhas da Placa Relé



Fonte: O próprio Autor

Para alterar e ler os estados do relé, a placa usa o circuito eletrônico como ilustra a figura 16 é baseado no uso de um relé de comando intermediário de tensão de 24 V para intermediar o sinal binário advindo diretamente do microcontrolador até o controle direto da chave de contato do barramento CC. Do lado esquerdo, o relé da mesma referência para a releitura do estado das chaves de contato do barramento para serem lidos pelo microcontrolador.

Figura 16 Circuito eletrônico de comando e recepção dos reles



Fonte: O próprio Autor

O *SpeedGoat* se comunica com o STM32 por meio do padrão RS-232 (0 V - lógica alta e 12 V - lógica baixa) por meio de um cabo do tipo DB-9. Como o microcontrolador só funciona com uma tensão entre 0 V e 5 V, é necessário realizar a conversão de tensão com o circuito integrado HIN202E.

O STM32 foi programado para receber e enviar um sinal de 5 *bytes* a cada 10 ms por meio do protocolo UART (*Universal Asynchronous Receiver/Transmitter*), a uma velocidade de 19200 *Bauds*/s. Para abrir o relé G5Q-1, o circuito elétrico usa um driver RC para polarizar o MOSFET BSS138. Esse circuito limita os distúrbios perigosos de alta frequência que poderiam danificar o circuito e aumenta a velocidade de polarização do transistor. Além disso, é necessário usar um diodo de roda livre tipo 1N4700A em paralelo com o relé para evitar picos de tensão nas bobinas do relé. No outro lado dos relés G5Q, há um contato para suportar tensões de 240 Vcc e um diodo SCHOTTKY DO - 201 em paralelo para limitar a tensão nesse contato.

Em seguida, as bobinas do relé ativam o contator que alimenta os conversores na bancada de envelhecimento. Por fim, os relés, também do tipo G5Q-1, leem os estados dos contatores. Um filtro RC é colocado entre o contato dos relés de leitura e as entradas do STM32 para atenuar o efeito de salto da abertura e do fechamento dos relés.

Durante o desenvolvimento da placa relé, uma questão importante é a distorção da curva de acionamento e desligamento do relé devido a fenômeno de *bouncing* característico de sistemas eletromecânicos com esses. Para neutralizar esse efeito, a placa foi projetada com um filtro passa baixa do tipo RC para atenuar as altas frequências na tensão sobre o relé produzidas durante o fechamento do contato do relé. O filtro RC fornece uma atenuação ao fechamento do relé da placa, como também a chave de contato, uma vez que ambos estão conectados em cascata.

Considerando que o tempo máximo de acionamento do contato do relé seja de 10ms, a constante de tempo do filtro (τ) deve ser desse mesmo valor. Dessa forma podemos escolher os valores R = 100 $k\Omega$ e C = 100 nF para que $\tau = RC = 10\ ms$.

Uma questão pode surgir durante a análise desse sistema é: Se o *SpeedGoat* é tão poderoso em termo de hardware, por que ele não trata diretamente a abertura e fechamento dos contatos sem o intermédio do STM32? Apesar das dificuldades de saber mais detalhes do projeto devido a política de confidencialidade dos seus idealizados, é possível fazer algumas inferências para esclarecer algumas questões.

Em primeiro lugar, a resposta mais simplista é que, apesar de ele ter um *hardware* robusto e relativamente poderoso para essa simples tarefa, esse poderio pode não ser suficiente para lidar com essas simples tarefas juntamente com as outras complexas tarefas que ele realize como tratar a temperatura, realizar a modulação dos conversores etc. Dessa forma, qualquer atraso ou falha em qualquer uma dessas tarefas por comprometer todas as outras.

A outra resposta por ser apresentada com uma forma de questão de segurança. Em resumo, se, por algum motivo, o *SpeedGoat* falhar, o STM32 serve como uma camada a mais de proteção em termo de *hardware* e de verificação de dados para evitar que qualquer falha de tratamento de dados ou de pane elétrica possa comprometer a integridade do banco de IGBT's. Dessa forma, cria-se uma redundância para a transmissão dos dados de comando.

Portanto, é mais seguro manter essa relação de intermediário entre o controlador escravo (STM32) e mestre (*SpeedGoat*) para evitar uma sobrecarga de processamento sobre o controlador mestre, do que sobrecarregar o controlador mestre em termos de processamento e de uso.

3 DETALHAMENTO DA PLACA RELÉ

Essa secção tem como objetivo continuar a secção anterior em uma intenção de detalhar os elementos mais importantes que compõem a placa relé e levantar algumas questões pertinente sobre a relação entre a placa e o *SpeedGoat.*

3.1 Relés

Os relés referência G5Q-1-EU 24DC, ilustrado pela figura 17, são responsáveis por transmitir para os sinais de comando de abertura e desligamento dos contactores do barramento CC por intermédio dos transistores MOSFET. Esse relé é uma escolha adequada para circuitos controlados por MOSFETs, como o BSS138K que será apresentado mais a frente, devido à sua compatibilidade técnica e versatilidade. Este relé opera com uma tensão de controle de 24Vdc e requer uma corrente de bobina de apenas 16,7 mA, valores que o MOSFET de 3,3 V podem acionar facilmente, já que suporta correntes de dreno de até 200 mA.

Figura 17 - Relé G5Q-1-EU 24DC

Fonte: Farnell.com

Outra vantagem importante é o isolamento galvânico proporcionado pelo relé, que protege o circuito de controle contra interferências ou danos causados por altas tensões ou correntes provenientes da carga. Para garantir um funcionamento seguro e eficiente, foi incluído um diodo em roda-livre em paralelo com a bobina do relé, como uma forma de proteger o MOSFET contra picos de tensão gerados durante a desenergizarão.

As principais características desses relés são mostradas na tabela 3:

Tabela 3 - Dados do relé G5Q-1-EU-24DC

Característica	Valor
Consumo da bobina dos relés	400 mW
Resistencia das bobinas dos relés	1440 Ω
Tensão de corte máximo	277 V_{AC}, 30 V_{DC}
Corrente de corte máxima	10 A_{AC} ou 5 A_{DC}
Resistência de contato	100 $m\Omega$
Tempo máximo para a abertura	10 ms
Tempo máximo para o fechamento	5 ms

Fonte: Datasheet relé G5Q-1-EU-24DC, OMRON (2021)

3.2 Transistores MOSFET do circuito de acionamento dos relés

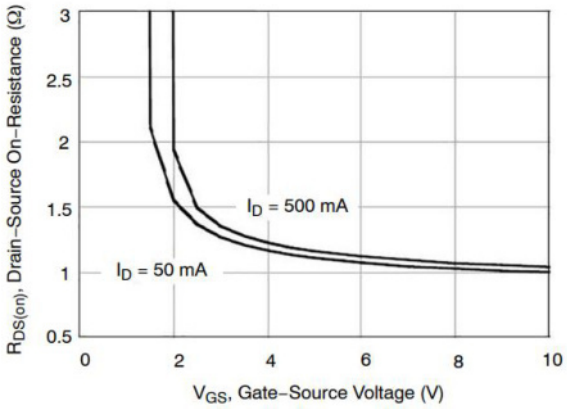
O MOSFET BSS138K ilustrado na figura 18 é uma boa escolha para o controle do relé G5Q-1-EU 24DC. Esse MOSFET de canal N opera eficientemente em baixas tensões de gate (V_{GS}) que varia entre 0,5 V e 1,3 V, tornando-o compatível com níveis de sinal de microcontroladores que utilizam lógicas de 3,3 V ou 5 V.

Figura 18 - MOSFET BSS138K

Fonte: ONSEMI (2022)

Isso garante que o dispositivo seja acionado de forma segura e confiável por um sinal digital. Além disso, como ilustrado na figura 19, o BSS138K apresenta uma baixa resistência de condução resistência de *drain-source* ($R_{DS(on)}$), de aproximadamente 1,2 Ω para uma tensão V_{GS}de 4 V. Isso reduz perdas de potência no componente devido à baixa $R_{DS(on)}$.

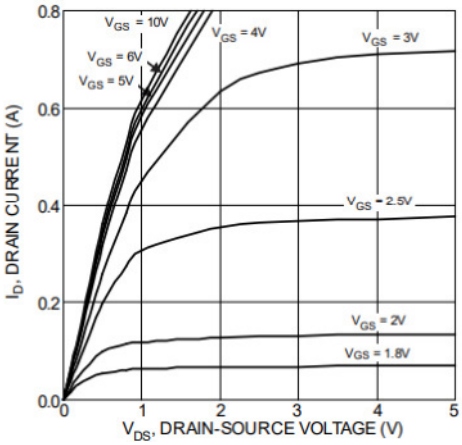
Figura 19 - Resistência *drain-source* por tensão *gate-source* no MOSFET BSS138K



Fonte: Datasheet MOSFET BSS138K ONSEMI (2022)

Como ilustrado pela figura 20, ele possui uma corrente típica de operação em torno de 700 mA para 3 V de V_{GS}, o que é suficiente para acionar o G5Q-1-EU. Segundo o seu datasheet, a tensão máxima de dreno-fonte (V_{DS}) do BSS138K é de 50 V, um valor que oferece ampla margem de segurança em relação à tensão de operação de 24 V do relé. Essa característica garante que o MOSFET possa operar de maneira confiável sem risco de avarias causadas por sobretensões comuns no ambiente de acionamento.

Figura 20 - Corrente de dreno por tensão dreno-source do MOSFET BSS138K



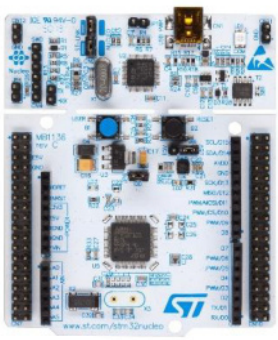
Fonte: Datasheet MOSFET BSS138K ONSEMI (2022)

3.3 A placa de desenvolvimento *Nucleo STM32F334R8*

A placa de desenvolvimento *Nucleo STM32F334R8* é o elemento da placa relé responsável pela comunicação e pela verificação de problemas de verificação. A escolha de utilizar um kit de desenvolvimento no lugar do próprio microcontrolador STM32F334R8

soldado a placa é devido a sua facilidade para o embarque do código fonte por meio de um cabo USB do tipo mini – B e pela presença da unidade de depuração presente nesse módulo que se mostrará valioso durante o desenvolvimento desse projeto.

Figura 21 - Nucleo STM32F334R8



Fonte: STmicroelectronics (2022)

3.3.1 Motivação para a escolha do microcontrolador nessa aplicação

Atualmente, existem uma larga variedade de microcontroladores do mercado dentre os mais populares, temos ATMega, ESP32, Adafruit, STM32, Raspberry pi pico, etc. Diante dessa ampla variedade de opções, uma questão que é levantada é um projeto de eletrônica é a escolha do microcontrolador para a aplicação em um contexto específico. No contexto de energias renováveis que trabalham em altas tensões, os fatores que são mais levados em conta são a confiabilidade e poder de processamento em tempo real. Assim, procura-se um equipamento que tenha uma alta confiabilidade a longos termos e uma alta variedade de opções de programação do seu *Hardware*. Um simples comparativo entre os microcontroladores de maior popularidade é ilustrado pela tabela 4.

Tabela 4 - Comparativo entre os microcontroladores mais populares do mercado

Característica	**STM32F334R8**	**LAUNCHXL-F28379D**	**dsPIC33CH**
Chip	ARM Cortex-M4	C28x	MIPS16e
Arquitetura	32 bits	Dual-core 32 bits	16 bits
Frequência de *Clock*	120 MHz	Até 200 MHz	100 MHz
Memória Flash	64 KB	1 MB (externa)	512 KB
RAM	12 KB	164 KB (SRAM)	72 KB
General purpose Input and Output (GPIO)	37	97	69
Comunicação	UART, I2C, SPI, CAN, LIN	UART, I2C, SPI, CAN, BT	UART, I2C, CAN, SPI
ADC (Conversor A/D)	12 *bit*s, 4 canais	12 *bit*s, 16 canais	10 *bit*s, 6 canais
PWM	3 geradores de PWM	24 canais configuráveis	8 canais
Tensão de Operação	2,0V – 3,6V	3,3V - 5V	5V
Consumo de Energia	Baixo (modos de economia)	Médio	Baixo
Custo	$ 30	$ 60	25

Fonte: O próprio Autor

Apesar de não ser o mais rápido e não ter a maior gama de recursos o STM32 se destaca pela alta precisão, pois ele foi projetado especialmente para aplicações de controle de potência, como inversores e conversores, com suporte para geradores de PWM de alta precisão e ADCs avançados. O núcleo ARM Cortex-M4 é otimizado para processamento em tempo real, sendo excelente para controle e processamento de informação em tempo real. Ele também possui modos de baixo consumo, que podem ser úteis em aplicações de longa duração e dispositivos móveis alimentados por bateria. A comunicação *controller area network* (CAN) e *Local Interconnect Network* (LIN), apesar de não ter sido escolhida para o projeto, permite fácil integração em aplicações automotivas e industriais, um recurso que o Arduino UNO e o ESP32 geralmente não suportam diretamente. Finalmente, o fator mais importante é sua confiabilidade e confiança que é um fator importante para sistemas onde desempenho e segurança são essenciais.

3.3.2 O microcontrolador STM32F334R8

A placa de desenvolvimento *Nucleo STM32F334R8* é uma plataforma de hardware projetada para prototipagem e desenvolvimento de aplicativos embarcados. Equipada com um microcontrolador STM32F334R8 da STMicroelectronics.

O microcontrolador STM32F334R8 pertence à família STM32F3 de microcontroladores baseados em ARM Cortex-M4, conhecida por seu alto desempenho e capacidade de processamento de ponto flutuante.

Um dos destaques da placa *Nucleo STM32F334R8* é sua compatibilidade com o ecossistema de desenvolvimento da *STMicroelectronics*, incluindo o software *CubeIDE*, um conjunto abrangente de bibliotecas de software, exemplos de código e ferramentas de desenvolvimento. A placa oferece diversas opções de conectividade e expansão, incluindo pinos de I/O (entrada/saída) digitais e analógicos, interfaces de comunicação como I2C, SPI e USART.

A placa se destaca pela flexibilidade em aplicações que demandam processamento preciso tempo real, sendo ideal para projetos em áreas como eletrônica de potência, controle de motores e automação industrial. Com seu conjunto avançado de periféricos, incluindo ADCs de alta resolução e temporizadores dedicados, ela é altamente adequada para tarefas que exigem medições rápidas em controle de malha fechada.

Além disso, a placa possui um *debugger*/programador ST-LINK/V2-1 integrado, facilitando o desenvolvimento, o teste e a depuração de aplicações diretamente na placa, sem a necessidade de hardware adicional.

O STM32F334R8 é um microcontrolador de 32 *bits*. Ele é usado neste projeto para controlar e coletar dados da placa de relé. Para fazer isso, ele se comunica com o *SpeedGoat* por meio do protocolo de comunicação UART (*Universal Asynchronous Receiver/Transmitter*) e do padrão RS232.

Esse microcontrolador tem os seguintes recursos principais, como a tabela 5 apresenta:

Tabela 5 - Dados do microcontrolador STM32

Característica	Valor
Frequência máxima de CPU	72 MHz
Frequência máxima de CLOCK	120 MHz
N° de portas input/output	51
Tensão de alimentação	5 V
N° de DMA (*Direct Memory Acess*)	7
N° de portas UART	3
N° de TIMERs	12
Tensão máxima de entrada do ADC	3,6 V

Fonte: Datasheet STM32F334R8, STMicroeletronics (2021)

Para a placa de relé, 10 GPIOs (entradas/saídas de uso geral) são usados no modo de saída para controlar os relés e 10 GPIOs no modo de entrada para ler o status dos interruptores. 4 pinos também são usados para comunicação serial UART1 e UART2. A UART1 é usada para comunicação com o *SpeedGoat*, enquanto a UART2 é usada para depuração com um *Hyper Terminal* em um PC.

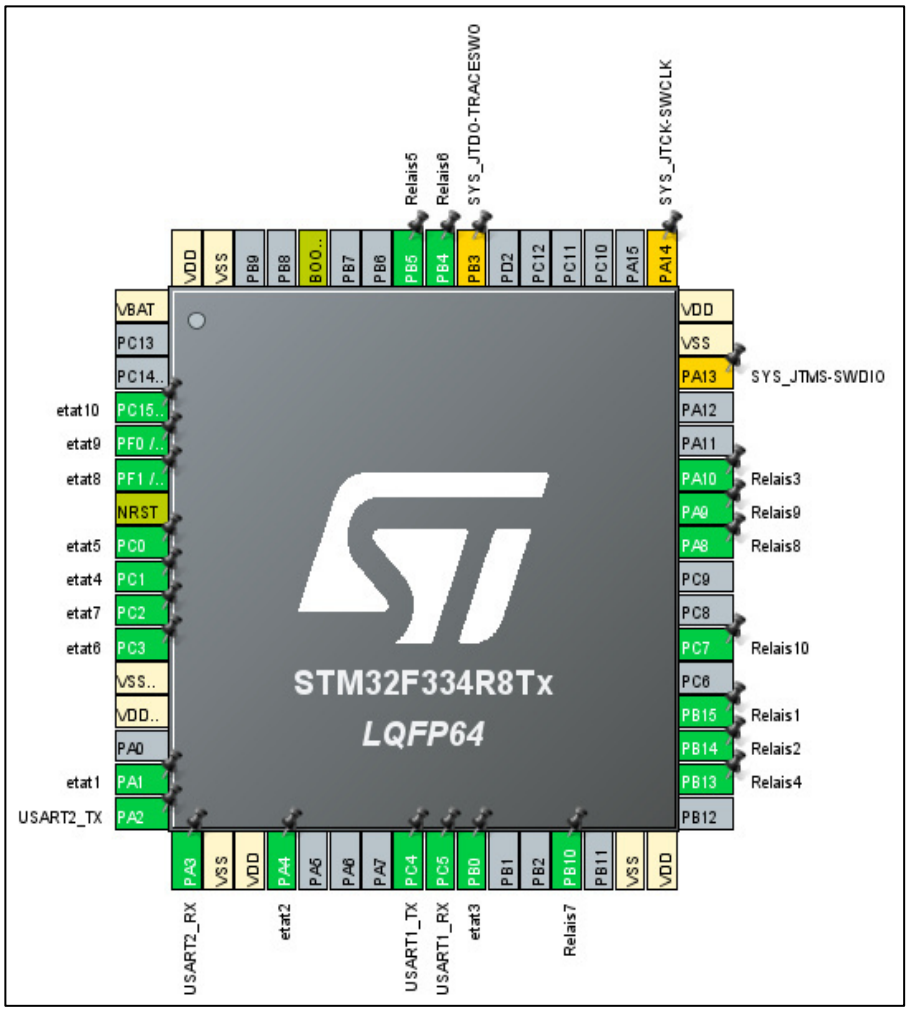
É importante mencionar o fato que para realizar a ativação do módulo debugger da placa, é necessário a ativação de alguns pinos específicos da placa como é o caso dos pinos PB3, PA14 e PA13. O recurso de *debugger* será de grande importância para esse projeto, pois será usado para o monitoramento do fluxo de informação em tempo real. O layout dos pinos no microcontrolador é o seguinte mostrado na tabela 6:

Tabela 6 - Configurações dos pinos do STM32

Pino	Função
PC7	Saída do comando relé 10
PA0	Saída do comando relé 9
PA8	Saída do comando relé 8
PB10	Saída do comando relé 7
PB4	Saída do comando relé 6
PB5	Saída do comando relé 5
PB13	Saída do comando relé 4
PA10	Saída do comando relé 3
PB14	Saída do comando relé 2
PB15	Saída do comando relé 1
PC15	Entrada leitura do estado do relé 10
PF0	Entrada leitura do estado do relé 9
PF1	Entrada leitura do estado do relé 8
PC2	Entrada leitura do estado do relé 7
PB4	Entrada leitura do estado do relé 6
PB5	Entrada leitura do estado do relé 5
PC1	Entrada leitura do estado do relé 4
PB0	Entrada leitura do estado do relé 3
PA4	Entrada leitura do estado do relé 2
PA1	Entrada leitura do estado do relé 1
PA3	Comunicação UART2 recepção
PA2	Comunicação UART2 envio
PC5	Comunicação UART1 recepção
PC4	Comunicação UART1 envio
PB3	Ativação da função de d*ebug*
PA14	Ativação interface cabo serial do *debug*
PA13	Ativação interface cabo serial do *debug*

Fonte: O próprio Autor

Figura 22 - Configuração dos pinos via *CubeIDE*



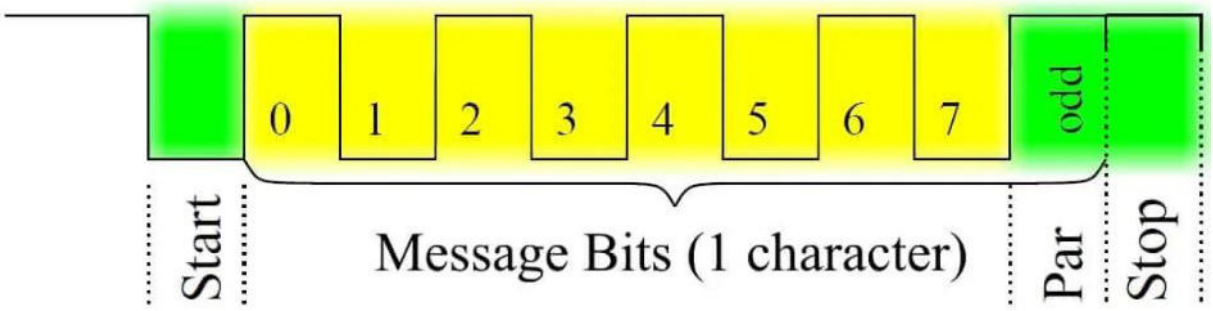
Fonte: O próprio Autor

4 PROTOCOLO DE COMUNICAÇÃO E SEUS OBJETIVOS

4.1 Comunicação UART

A sigla UART significa *universal asynchronous receiver / transmitter* e define um protocolo, ou conjunto de regras, para a troca de dados em série entre dois dispositivos. A UART é muito simples e utiliza apenas dois fios entre o transmissor e o receptor para transmitir e receber em ambas as direções. Ambas as extremidades têm também uma ligação à terra. A comunicação em UART pode ser *simplex* (os dados são enviados apenas numa direção), *half-duplex* (cada lado fala, mas apenas um de cada vez) ou *full-duplex* (ambos os lados podem transmitir simultaneamente). No caso desse projeto é utilizado a configuração *full-duplex*, os dados na UART são transmitidos sob a forma de *bits*, sendo eles start *bit* (*bit* 1 que indica o início da mensagem), dados a serem transmitidos (*bits* 2-9), *bit* de paridade (*bit* 10 que indica se a paridade do número de *bits* da mensagem iguais a 1 é par) e *stop bit* (*bit* 11 indica o fim da mensagem). A figura 23 ilustra o formato da comunicação UART.

Figura 23 - Ilustração do formato do protocolo de comunicação UART

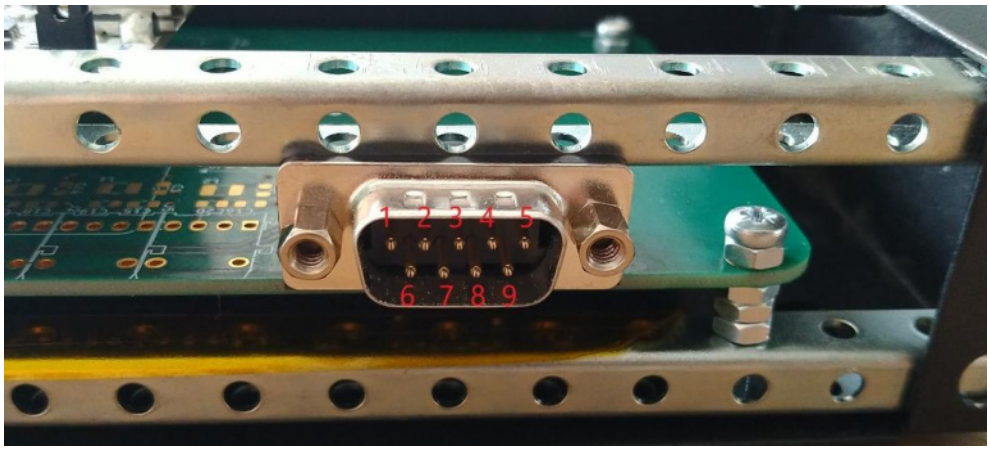


Fonte: netwerkt.wordpress.com. Disponível em: netwerkt.wordpress.com/tag/vhdl/. Acesos em: 24, dez. 2024

4.2 Protocolo de comunicação RS232

Para se comunicar, os módulos usam um conector DB-9 um tipo de conector com 9 pinos, utilizado principalmente em interfaces seriais, como o padrão RS-232, para comunicação entre dispositivos eletrônicos. Embora tenha sido gradualmente substituído por tecnologias mais modernas, como USB e Ethernet, o DB-9 ainda é encontrado em sistemas com aplicações que requerem comunicação serial direta. O layout dos pinos do conector é ilustrado pela tabela 7 e figura 24:

Figura 24 - Conector RS232 da placa relé



Fonte: O próprio Autor

Tabela 7 - Funções dos pinos do conector RS232

Pino	Função
1 (DCD - *Data Carrier Detect*)	Detecta a presença de um sinal portador de dados
2 (RXD - *Receive Data*)	Recebe dados
3 (TXD - *Transmit Data*)	Transmite dados
4 (DTR - *Data Terminal Ready*)	Indica que o terminal está pronto para comunicação
5 (GND - *Ground*)	Terra, referência para os sinais
6 (DSR - *Data Set Ready*)	Indica que o modem está pronto para a comunicação
7 (RTS - *Request to Send*)	Solicita a transmissão de dados
8 (CTS - *Clear to Send*)	Indica que os dados podem ser enviados
9 (RI - *Ring Indicator*)	Indica que há uma chamada recebida (em modems)

Fonte: O próprio Autor

As etapas de comunicação entre esses dispositivos são executadas de forma que o *SpeedGoat* seja o mestre da comunicação e o microcontrolador STM32, o escravo:

- O *SpeedGoat* envia uma mensagem de 5 *bytes* com uma velocidade de 19200 Bauds/s a cada 10 ms em padrão RS232 para o conector DB9 da placa de relés. Essa mensagem de 5 *bytes* contém o comando dos relés da primeira linha da placa;
- O componente HIN211ECBZ transforma o sinal RS 232 em um sinal de 0 a 5 V;
- O microcontrolador STM32 decodifica o sinal UART para abrir ou fechar o relé de acordo com a mensagem;

- O microcontrolador armazena o estado dos relés de leitura em uma variável e codifica uma mensagem de 5 *bytes*. Em seguida, ele envia essa mensagem por meio do protocolo UART a uma velocidade de 19200 Bauds/s para o transceptor HIN211ECBZ, que a converte em RS-232;
- O status dos relés de leitura é enviado para o *SpeedGoat*

4.3 Formato das mensagens trocadas na comunicação

Uma mensagem com exatos 5 *bytes* é enviada a cada 10 ms para realizar a comunicação entre o *STM32* e o *SpeedGoat* e vice-versa. Ela possui o seguinte formato como ilustra a figura 25, de modo que Rx representa o *bit* de estado fechado (1) ou aberto (0) do relé x para cada um dos 10 relés controlados, a mensagem sempre possui os *bytes* 0 e 2 redundantes, assim como os *bytes* 1 e 3, em uma tentativa de detectar possíveis problemas de comunicação. O byte 4 (*Carriage return*) tem um valor constante de 13 (00001101), para identificar o fim da mensagem. Após a validação da mensagem, os *bytes* 0 e 1 são utilizados para transportar os estados dos relés Rx. Além disso, é importante notar, como ilustrado pela figura 25, que os *bits* em preto são estáticos e não devem mudar durante uma operação normal, já aqueles ilustrados em vermelho são dinâmicos e são alterados de acordo com a sua função a ser desempenhada.

Figura 25 - Formato da mensagem de recepção e reenvio

Fonte: O próprio Autor

A mensagem também contém os *bits* de sinal de vida (BSV), que alternam a cada mensagem de 5 *bytes* enviada entre 0 e 1 de forma consecutiva e ininterrupta dentro do período

de 10ms. Esse *bit* monitora a perca de comunicação entre os dois dispositivos, uma vez que a recepção de dois *bits* iguais de mensagens consecutivas indica algum erro de comunicação.

O *bit* de Transmissão não válida (TNV) é pertinente somente para envio do STM32 para o *SpeedGoat*, pois esse *bit* têm o papel de informar ao controlador mestre que a mensagem enviada por ele anteriormente é válida ou inválida, ou seja, se o controlador mestre receber um TSV = 1, isso indica que ocorreu um erro de comunicação com a última mensagem ou a formatação da mensagem estava inválida. Caso contrário, o *bit* TSV deve ser configurado para TSV = 0, o que indica mensagem válida. Esse *bit* é importante para que ambos os lados esteja cientes do bom funcionamento da comunicação.

4.4 Objetivos da comunicação

Em termos de algoritmos e em regime normal de operação, que será especificado mais a frente, o código responsável pela aquisição desses dados deve realizar dentro desse período de envio de 10 ms, sem atraso, sincronizado com o controlador mestre e de forma ininterrupta as seguintes tarefas em sequência:

1. Receber os 5 *bytes* enviados sem perder nenhum único byte pelo protocolo UART;
2. Validar a mensagem recebida para averiguar a ocorrência de erros;
3. Extrair os comandos dentro da mensagem a serem executados nas saídas GPIO;
4. Ler os estados dos relés pelos GPIO's e criar uma mensagem de 5 *bytes* do mesmo formato daquela recebida como mostra a figura 25;
5. Retransmitir a mensagem dos estados dos relés pelo protocolo UART;
6. Verificação contínua da integridade da comunicação, por meio de uma interrupção com período de 10 ms;
7. Entrar em estado de proteção caso o cabo de transmissão de dados seja desconectado e a comunicação e processamento de dados seja restabelecido automaticamente, caso o cabo seja reconectado.

4.5 Configuração de comunicação via *CubeIDE*

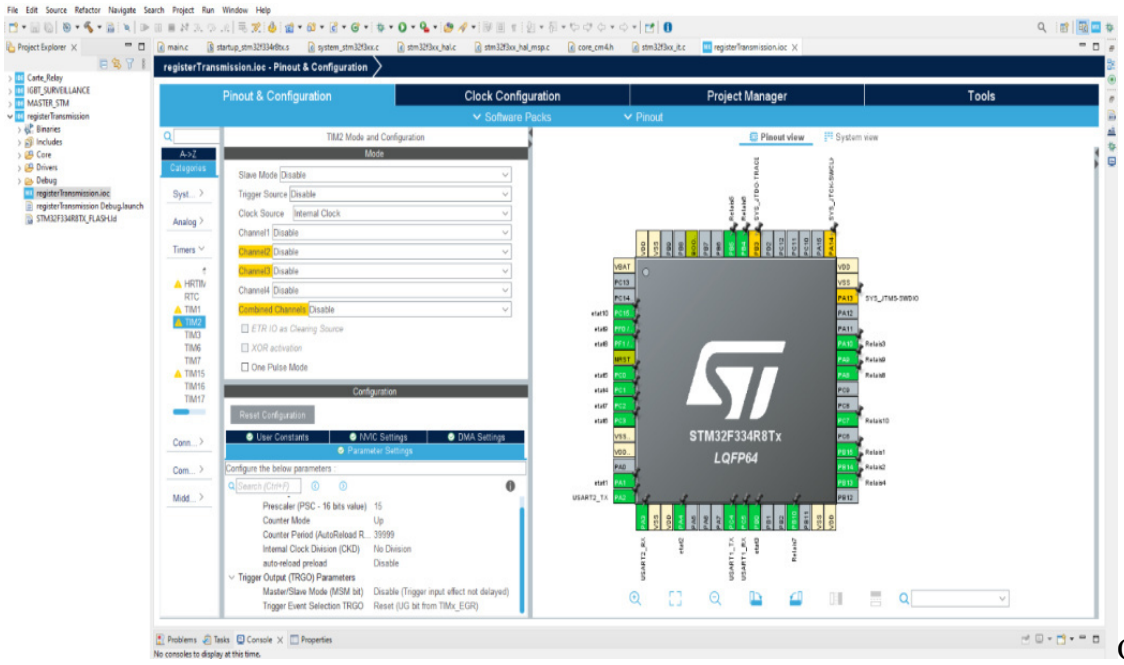
A *CubeIDE* é um ambiente de desenvolvimento integrado (IDE) desenvolvido pela STMicroelectronics, projetado para facilitar o desenvolvimento de aplicações para microcontroladores STM32. Ela oferece ferramentas para edição de código, depuração, e

compilação, integrando recursos como suporte ao STM32CubeMX (configuração de periféricos e geração de código) e bibliotecas otimizadas para esses microcontroladores. É baseada no Eclipse e voltada para simplificar o trabalho com hardware STM32.

A utilização da *CubeIDE*, em comparação com outros *frameworks* como a Arduino IDE e a platformIO, apresenta diversas vantagens para projetos que buscam maior aproveitamento das funcionalidades do hardware. A *CubeIDE* oferece ferramentas específicas para a configuração detalhada do hardware, como configuração de periféricos, *clock*, pinos em uma interface gráfica e intuitiva. Além disso, ela gera automaticamente o código base para inicialização do microcontrolador, otimizando o tempo de desenvolvimento e reduzindo a possibilidade de erros manuais.

Outro benefício significativo é a flexibilidade e o acesso ao HAL (*Hardware Abstraction Layer*) e ao LL (*Low Layer*), que fornecem diferentes níveis de controle do hardware até o seu nível de programação em *bits*. O HAL simplifica o desenvolvimento com APIs de alto nível, enquanto o LL permite maior desempenho em aplicações que demandam eficiência e rapidez. Uma amostra da interface gráfica é ilustrada pela figura 26.

Figura 26 - Interface gráfica de configuração do STM32 via *CubeIDE*



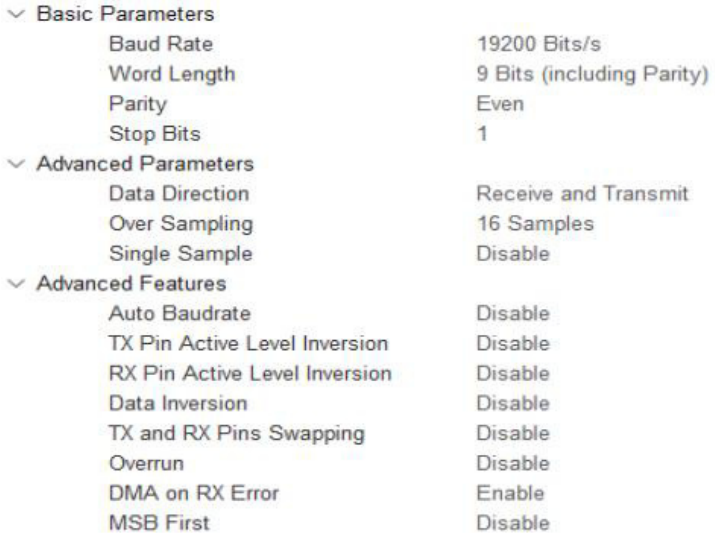
O

Fonte: O próprio Autor

Em relação aos parâmetros básicos de configuração da comunicação na interface gráfica da *CubeIDE*, como ilustrado pela figura 27, foi escolhido a velocidade a transmissão de 19200

bits por segundo para uma mensagem de 8 *bits* (1 *byte*), um *bit* de paridade “par” e um *bit* de “parada” (*stop bit*) enviada pelo protocolo *UART full-duplex* (envio e recepção). O parâmetro *oversampling* refere-se a uma técnica usada em conversores analógico-digitais (ADC) para melhorar a resolução ou a precisão de medições. No caso de "*oversampling = 16 samples*", isso significa que o sistema coleta 16 amostras consecutivas de um sinal e, em seguida, processa essas amostras para obter um único valor representativo.

Figura 27 - Configuração basica de comunicação pelo protocolo UART



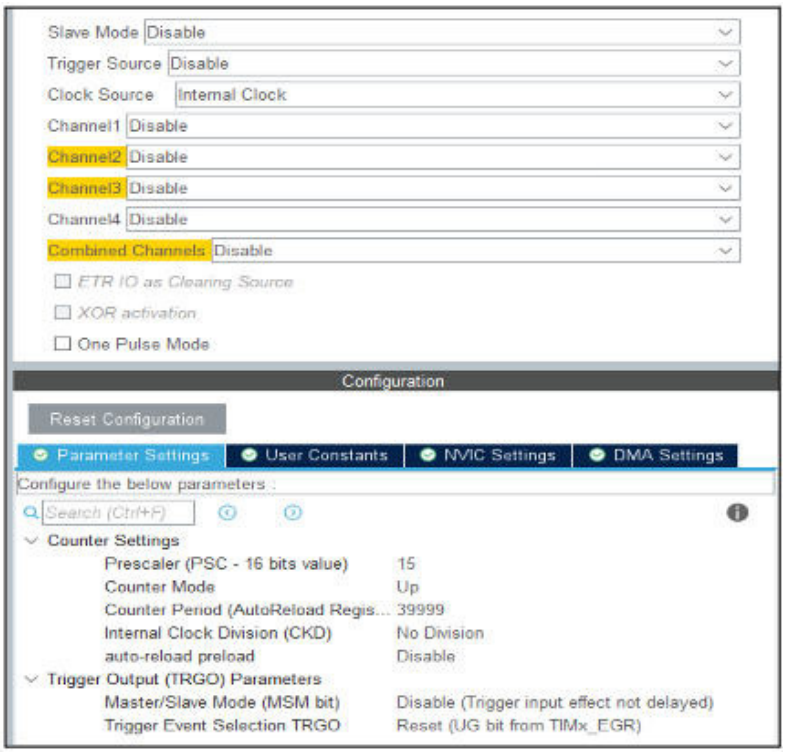
Fonte: O próprio Autor

As configurações fornecidas para o timer em um microcontrolador STM32 determinam como ele irá contar e gerar interrupções, como ilustrado pela figura 28. O *prescaler (PSC)* = 15 divide a frequência do *clock* base do timer por 16 (o valor do *prescaler* é sempre PSC+1) ou seja, com o *clock* interno do STM32 configurado para 64 MHz a frequência efetiva do contador do *timer* será reduzida para 4 MHz. O *Counter Mode (Up)* indica que o contador do timer irá contar de 0 até o valor definido no *counter period (ARR - Auto Reload Register)* é igual a 39999, logo seu valor efetivo é *counter register + 1, ou seja, 40000*. Quando o contador atinge esse valor, ele retorna a zero e pode disparar eventos como interrupções (rotinas que são acionadas fora da rotina principal após o disparo de algum sinal).

Então cada incremento no contador ocorre a cada 0,25 µs (1/4 MHz). O contador atinge o valor 40.000 em 40.000 x 0,25 µs, resultando em um período de 10 ms, o que é o valor definido para o período de verificação de comunicação. O valor de *internal clock division (CKD)* como "sem divisão" significa que o *clock* do contador não sofre divisão adicional além

do *prescaler*. Por fim, com o *autoreload preload* desativado, qualquer modificação no valor do ARR é imediatamente efetivada, o que pode ser útil para alterar o período em tempo real sem esperar o fim de um ciclo de contagem.

Figura 28 - Configuração do TIMER



Fonte: O próprio Autor

5 ALGORTIMOS

Após, um longo trabalho fundamentando as bases do protocolo de comunicação a ser construida, essa secção do trabalho é dedicada a criação dos algoritmos de comunicação e sua aplicação no sistema embarcado.

5.1 Solução 1 de comunicação serial

Em uma primeira tentativa de solução utilizou-se os comandos padrões de comunicação como *HAL_UART_Transmit_IT* e *HAL_UART_Receive,* cujo argumentos utilizados para a comunicação são o tamanho do *buffer* e instância do protocolo de comunicação a ser utilizado, no caso UART1. Dessa forma, é uma simples e clássica forma de abordar o problema.

5.1.1 Handle de transmissão UART

O comando padrão para criar uma instancia de transmissão de dados UART é dado pelo *Handle HAL_UART_Transmit_IT(&huart1, data, sizeof)* que é utilizado para realizar a transmissão de dados pela UART de forma não bloqueante utilizando interrupções no STM32. Isso significa que a função não espera a transmissão de todos os dados para continuar a execução do código. Os seus argumentos são apresentados como:

- *&huart1:*O ponteiro para a estrutura *UART_HandleTypeDef*, que é o *Handle* do periférico *UART*. Essa estrutura contém todas as informações sobre a instância de *UART* que está sendo usada;
- *data*: Ponteiro para o *buffer* que contém os dados a serem transmitidos;
- *sizeof*: Especifica o número de *bytes* que devem ser transmitidos. Como especificado anteriormente, o formato padrão da mensagem de recepção é de 5 *bytes*.

5.1.2 Handle de recepção do UART

O comando padrão para criar uma instancia de transmissão de dados UART é dado pelo *Handle HAL_UART_Receive(&huart1, rxData, sizeof(rxData), timeout)* é usada para receber dados através da comunicação UART de forma bloqueante, ou seja, o código espera até que os dados sejam recebidos ou que o tempo limite (*timeout*) seja atingido. Ela espera receber uma

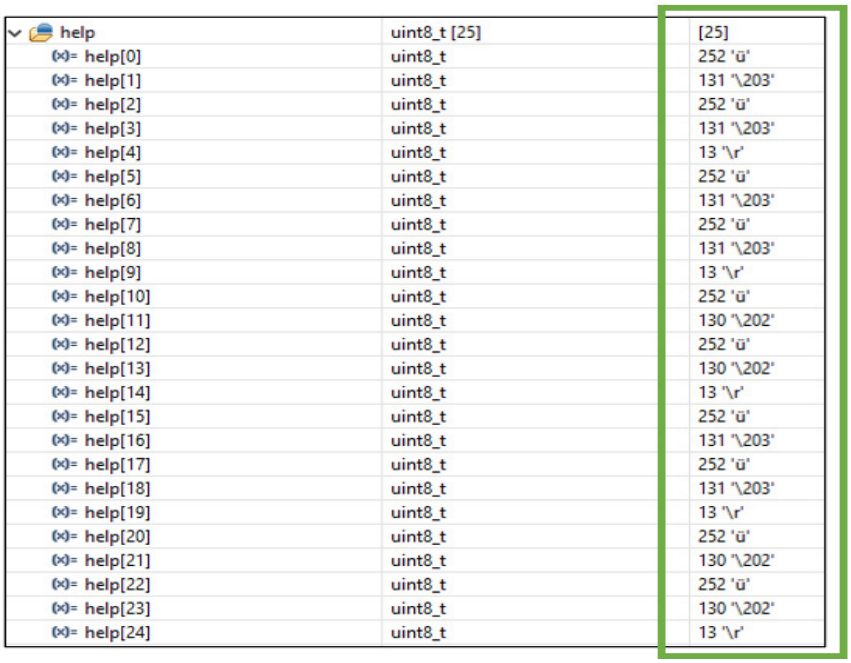
quantidade definida de dados no *buffer rxData* antes de retornar, ou até o timeout especificado. Os seus argumentos são apresentados como:

- *&huart1*: é o ponteiro para a estrutura *UART_HandleTypeDef*, que representa a instância de UART que você está usando. `huart1` é o *Handle* que contém as informações e a configuração da UART, no caso ele faz referência a UART 1;
- *rxData*: é um ponteiro para o *buffer* onde os dados recebidos pela UART serão armazenados;
- *sizeof(rxData)*: indica o tamanho do *buffer* em *bytes*. A função tentará receber exatamente essa quantidade de *bytes* da UART antes de retornar. Como especificado anteriormente, o formato padrão da mensagem de envio é de 5 *bytes*;
- *timeout*: é o tempo limite (timeout) para a operação de recepção, expresso em milissegundos. No caso do projeto, foi escolhido o timeout de 1ms, pois é 10 % do período de transmissão dos dados vindos do *SpeedGoat*, o que possibilita ao STM32 processar e tratar os dados em 9 ms.

5.1.3 Avaliação parcial da solução 1

Essa abordagem se mostrou eficiente no processo de recepção e envio de dados diante das restrições de 10 ms. A figura 29 mostra o fluxo de dados pela interface do *debugger* da *CubeIDE*. Assim, não foi verificada nenhum problema de comunicação entre os controladores mestre e escravo, seja por problema de atraso, dessincronização, mensagem corrompida ou ruído.

Figura 29 - Verificação parcial da recepção do pacote de mensagem recebida



help	uint8_t [25]	[25]
help[0]	uint8_t	252 'ü'
help[1]	uint8_t	131 '\203'
help[2]	uint8_t	252 'ü'
help[3]	uint8_t	131 '\203'
help[4]	uint8_t	13 '\r'
help[5]	uint8_t	252 'ü'
help[6]	uint8_t	131 '\203'
help[7]	uint8_t	252 'ü'
help[8]	uint8_t	131 '\203'
help[9]	uint8_t	13 '\r'
help[10]	uint8_t	252 'ü'
help[11]	uint8_t	130 '\202'
help[12]	uint8_t	252 'ü'
help[13]	uint8_t	130 '\202'
help[14]	uint8_t	13 '\r'
help[15]	uint8_t	252 'ü'
help[16]	uint8_t	131 '\203'
help[17]	uint8_t	252 'ü'
help[18]	uint8_t	131 '\203'
help[19]	uint8_t	13 '\r'
help[20]	uint8_t	252 'ü'
help[21]	uint8_t	130 '\202'
help[22]	uint8_t	252 'ü'
help[23]	uint8_t	130 '\202'
help[24]	uint8_t	13 '\r'

Fonte: O próprio Autor

Contudo, foi-se realizado o teste de retirar os jumpers de comunicação para verificar o cumprimento da tarefa 7 imposta ao STM32 na secção 4.4 e observou-se que, após a reintrodução dos *jumpers* em seus respectivos pinos, o sistema "congelava" e parava todas as suas rotinas, sejam aquelas do *loop* principal, seja aquelas advindas das interrupções.

Dentre os motivos desse erro no sistema, podemos citar:

- Quando você remove os jumpers, a UART pode entrar em um estado inconsistente, como uma condição de *overrun* (quando mais dados são recebidos do que o *buffer* pode armazenar) ou uma interrupção de *framing* (quando os dados recebidos não têm a formatação correta);
- Algumas interrupções não tratadas durante a desconexão dos jumpers, o STM32 pode não conseguir gerenciar corretamente o estado da UART caso o sinal seja interrompido. Analisando o datasheet do equipamento, confirma-se que existe esse comportamento (STM, stm32f334xx datasheet, p.959);
- Quebra da mensagem e não recepção no *stop bit* (*bit* que indica o final da mensagem), *ou seja,* o microcontrolador não sabe qual o final da mensagem e precisa para seu funcionamento para impedir danos ao sistema.

Portanto, esse problema gerado pela solução 1 compromete a integridade do nosso sistema e pode produzir um comando errado nas chaves de contato, o que pode produzir consequências desastrosas. Apesar de que o envio de dados pelo *HAL_UART_Transmit_IT* dessa proposta não mostra ser o entrave, mas sim o recebimento dela *por meio do HAL_UART_Receive.* Dessa forma, é necessário descartar a utilização do *HAL_UART_Receive*, conservar a utilização do *HAL_UART_Transmit_IT* e buscar uma opção que lide melhor com esse problema da desconexão do *jumper*.

5.2 Solução 2 de comunicação serial

Essa solução 2 tenta dar mais atenção para a tarefa 4 da secção 3.1 (Conexão e reconexão do cabo de transmissão de dados). Assim, busca-se uma alternativa que aproxime o programador do hardware para que ele possa ter mais controle sobre o fluxo de informação e o problema gerado pela solução 1.

Nesse contexto, a proposta baseia-se em extrair os dados diretamente do registrador USART_RDR que é constituído de um *buffer* (RDR) entre o barramento interno e o *shift register. O Shift register* é um tipo de circuito digital que é usado para armazenar e manipular dados em forma de *bits*, sendo uma interface direta entre o receptor e o transmissor. Dessa forma, evita-se de tratar qualquer erro de recepção por meio de rotinas pré-definidas pela desenvolvedora do produto, no quais o programador não possui total controle, para uma rotina criada pelo programador que trata desde a recepção até a detecção e tratamento dos erros. Logo, o código proposto pela solução 2 é responsável por duas tarefas.

A primeira é a recepção dos propriamente ditos dados vindos do controlador mestre *SpeedGoat*, por meio de *loop* infinito do tipo "while (1)" que aguarda o sinalizador, ou também chamado *flag*, RXNE indicar que existe um byte recebido pelo *buffer Receive Data register* (RDR) que está disponível para leitura. Assim, os *bytes* serão atualizados organizados dentro desse *buffer* criado e, após ser recebido o byte de fim de mensagem, a mensagem de 5 *bytes* é validada para, finalmente, serem utilizados os comandos contidos dentro dela.

A segunda parte é a detecção de erros, por meio de uma interrupção a cada 10 ms gerada pelo TIMER2, com o objetivo de analisar o comportamento do *bit* de sinal de vida (BSV) e verificar se os dois dispositivos estão se comunicando corretamente.

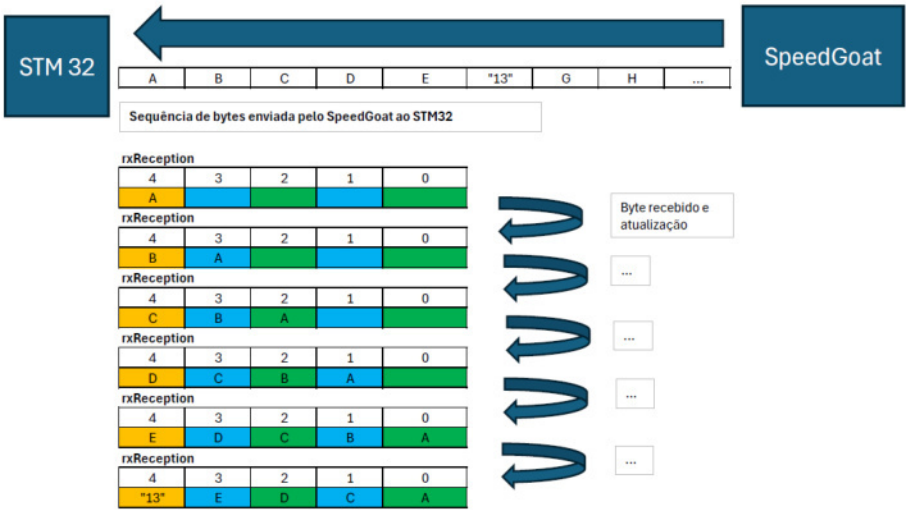
As seguintes variáveis são usadas para executar os algoritmos do código da solução 2 são:

- VAL (tipo ‘int’): Indica se a última mensagem recebida é válida;
- danger (tipo ‘int’): Conta o número de mensagens inválidas recebidas consecutivamente;
- bsv (tipo ‘int’): Variável que armazena o *bit* de sinal de vida (BSV);
- i e j (tipo ‘int’): Variáveis auxiliares;
- life (tipo ‘int’): Armazena o último BSV;
- dead (tipo ‘int’): Conta a cada 10 ms o número de vezes consecutivas sem que o *bit* de sinal de vida seja alterado;
- rxReception[5] (tipo ‘uint8_t’): *buffer* FIFO de 5 *bytes* de recepção.
- command[5] (tipo ‘uint8_t’): Mensagem válida de 5 *bytes* pronta para ser decodificada;
- state[10] (tipo ‘uint8_t’): variável de 10 *bytes* que armazena o estado (fechado ou aberto) dos relés;
- state[5] (tipo ‘uint8_t’): mensagem de 5 *bytes* decodificada pronta para ser enviada do STM32 para o *SpeedGoat*.

5.2.1 Novo algoritmo de recepção da solução 2

Como explicado resumidamente, cada byte enviado pelo *SpeedGoat* será armazenado no *buffer* RDR que sinaliza por meio do sinalizador RXNE (*flag*) que existe um byte de dados recebido e pronto para ser lido. Cada byte enviado pelo *SpeedGoat* é recebido e armazenado em um *buffer First In, First Out* (FIFO) criado pelo programador de 5 *bytes* chamado "*rxReception"*, de tal forma o último byte recebido é sempre armazenado na posição 4 do *buffer* e os outros recebidos anteriormente são deslocados para a próxima posição adjacente, de modo que o byte mais antigo armazenado é sempre excluído, conforme mostrado no diagrama abaixo.

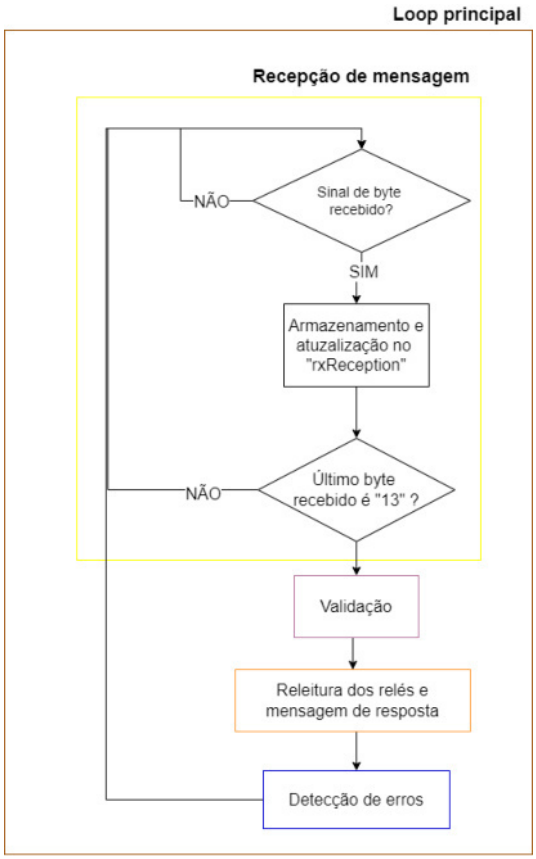
Figura 30 - Exemplo de recepção de mensagem



Fonte: O próprio Autor

Em seguida, o código verifica a cada atualização se o byte 4 possui o valor "13" (*Carriage return* - fim da mensagem). Caso positivo, o *buffer* estará cheio e pronto para ser validado. O diagrama abaixo resume essa parte do algoritmo:

Figura 31 - Esquema de algoritmo de recepção



Fonte: O próprio Autor

5.2.2 Algoritmo de validação e decodificação dos dados da solução 2

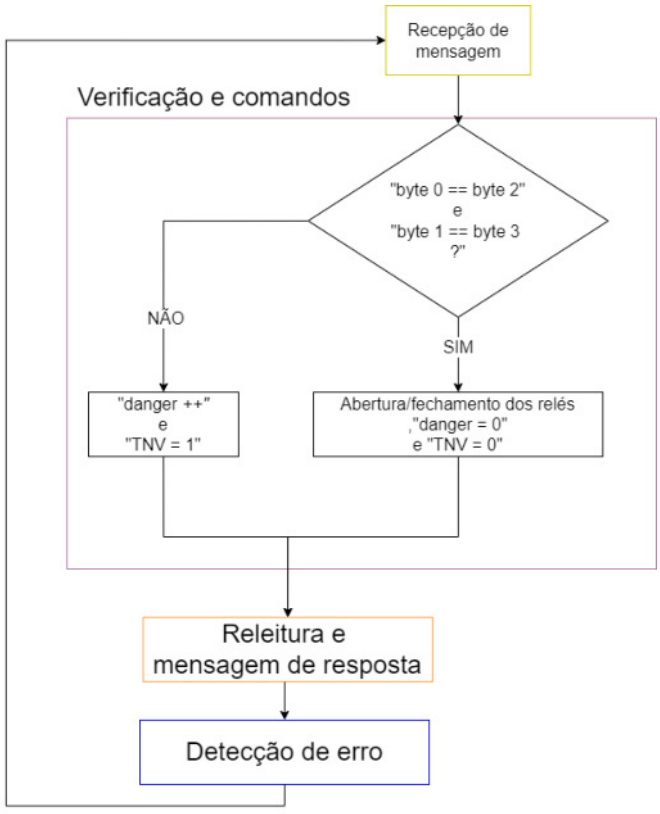
Assim que a mensagem for recebida, o código validará a mensagem para evitar e detectar qualquer problema de comunicação. Para realizar essa tarefa, o código simplesmente verifica se os *bytes* 0/2 e 1/3 são iguais.

Se a validade da mensagem for confirmada, os *bytes* 0 e 1 serão decodificados e os relés 1 a 10 serão fechados se o respectivo *bit* do relé for igual a 1 ou abertos se o respectivo *bit* do relé for igual a 0. Em seguida, o STM32 lerá os estados dos relés e enviará uma mensagem de 5 *bytes* ao *SpeedGoat* com a mesma estrutura da mensagem recebida. Em outras palavras, os *bytes* 0-2 e 1-3 são iguais e o byte 4 tem um conteúdo igual a "13". Os *bytes* 1 e 3 também

contêm um *bit* de sinal de vida (que muda a cada transmissão) e um *bit* que indica se a mensagem recebida foi válida.

A variável de perigo será incrementada em 1 sempre que ocorrer um erro ou será definida como zero se for recebida uma mensagem válida. Assim, se ocorrerem 10 erros consecutivos, o relé nº 1 será aberto de modo a cortar a alimentação de alta tensão da plataforma. A figura 32 a seguir resume o algoritmo de validação.

Figura 32 - Esquema do algoritmo de validação de mensagem



Fonte: O próprio Autor

Para realizar a decodificação dos *bits* em tomadas para abrir ou fechar os relés, é necessário criar um vetor de 10 valores inteiros que armazenará os *bits* que representam o estado a ser executado ao respectivo relé. Para realizar a extração dos *bits*, utilizou-se os operadores *bitwise* “>>”, também conhecido como *left shift* e o operador “&”, operador *AND*, para transportar os *bits* para a posição menos significativa do byte. Finalmente, os valores inteiros das variáveis são convertidos em comandos de abrir e fechar os respectivos relés pela função *HAL_GPIO_WritePin*. A figura 33 ilustra melhor a função *setRelayStates* utilizada para a decodificação.

Figura 33 Função de decodificação da mensagem

```
// Função para configurar os estados dos relés com base na mensagem recebida
void setRelayStates(uint8_t *command) {
    uint8_t relayPins[10] = {GPIO_PIN_15, GPIO_PIN_14, GPIO_PIN_10, GPIO_PIN_13, GPIO_PIN_5,
                             GPIO_PIN_4, GPIO_PIN_10, GPIO_PIN_8, GPIO_PIN_9, GPIO_PIN_7};
    GPIO_TypeDef *relayPorts[10] = {GPIOB, GPIOB, GPIOA, GPIOB, GPIOB,
                                    GPIOB, GPIOB, GPIOA, GPIOA, GPIOC};

    uint8_t relayStates[10] = {0};

    // Extrair os estados dos relés dos bytes da mensagem
    relayStates[0] = (command[0] >> 6) & 0x01;
    relayStates[1] = (command[0] >> 5) & 0x01;
    relayStates[2] = (command[0] >> 4) & 0x01;
    relayStates[3] = (command[0] >> 3) & 0x01;
    relayStates[4] = (command[0] >> 2) & 0x01;
    relayStates[5] = (command[0] >> 1) & 0x01;
    relayStates[6] = command[0] & 0x01;

    relayStates[7] = (command[1] >> 6) & 0x01;
    relayStates[8] = (command[1] >> 5) & 0x01;
    relayStates[9] = (command[1] >> 4) & 0x01;

    // Configurar os pinos dos relés com base nos estados extraídos
    for (int i = 0; i < 10; i++) {
        if (relayStates[i]) {
            HAL_GPIO_WritePin(relayPorts[i], relayPins[i], GPIO_PIN_SET);
        } else {
            HAL_GPIO_WritePin(relayPorts[i], relayPins[i], GPIO_PIN_RESET);
        }
    }
}
```

Fonte: O próprio Autor

5.2.3 Algoritmo de releitura e envio de resposta da solução 2

O algoritmo utilizado para a releitura é baseado nos mesmos princípios utilizados no processo mostrados anteriormente de decodificação da mensagem. Portanto, a função *sendRelayStatus* de envio da mensagem de resposta pode ser resumida como o processo inverso da função decodificação. A figura 34 ilustra melhor a função *sendRelayStatus* utilizada para a releitura dos relés e envio de resposta.

Figura 34 Função modificada utilizado para releitura dos relés e envio de resposta

```
// Função para construir e enviar a mensagem
void sendRelayStatus(void) {
    uint8_t relayPins[10] = {GPIO_PIN_1, GPIO_PIN_4, GPIO_PIN_0, GPIO_PIN_1, GPIO_PIN_0,
                             GPIO_PIN_3, GPIO_PIN_2, GPIO_PIN_1, GPIO_PIN_0, GPIO_PIN_15};
    GPIO_TypeDef *relayPorts[10] = {GPIOA, GPIOA, GPIOB, GPIOC, GPIOC,
                                    GPIOC, GPIOC, GPIOF, GPIOF, GPIOC};

    uint8_t relayStates[10] = {0};
    uint8_t message[5] = {0};

    // Ler o estado dos relés
    for (int i = 0; i < 10; i++) {
        relayStates[i] = HAL_GPIO_ReadPin(relayPorts[i], relayPins[i]);
    }

    // Construir os bytes da mensagem
    message[0] = 0x80 | (relayStates[0] << 6) | (relayStates[1] << 5) | (relayStates[2] << 4) |
                 (relayStates[3] << 3) | (relayStates[4] << 2) | (relayStates[5] << 1) | relayStates[6];

    message[1] = 0x80 | (relayStates[7] << 6) | (relayStates[8] << 5) | (relayStates[9] << 4) |
                 ((TNV & 0x01) << 1) | (0x01 & BSV_state);

    message[2] = message[0];
    message[3] = message[1];
    // Byte fixo conforme especificação
    message[4] = 0x0D;

    // Alternar o estado do BSV para a próxima mensagem
    BSV_state = !BSV_state;

    // Enviar a mensagem pela UART 2
    HAL_UART_Transmit(&huart2, message, 5, HAL_MAX_DELAY);
}
```

Fonte: O próprio Autor

5.2.4 Algoritmo de verificação de conexão da solução 2

O algoritmo geral do controlador escravo STM32 é dividido em 2 tarefas que ocorrem dentro de um mesmo thread (linha de execução de tarefas) sem serem simultâneas: enviar/receber uma mensagem que é gerenciada pela rotina principal do algoritmo já apresentado e a verificação da integridade dessa verificação, por meio de uma interrupção com período de 10 ms e com prioridade sobre a rotina principal.

5.2.4.1 Interrupções em microcontroladores

Interrupções em microcontroladores são mecanismos que permitem a interrupção temporária do fluxo de execução normal de um programa para lidar com eventos externos ou internos que exigem atenção imediata. Quando uma interrupção ocorre, o microcontrolador pausa o que está fazendo, executa uma rotina especial chamada rotina de tratamento de

interrupção ISR (*Interrupt Service Routine*), e depois volta para a tarefa que estava executando antes.

Interrupções são extremamente úteis porque permitem que o microcontrolador responda rapidamente a eventos importantes sem a necessidade de ficar constantemente verificando o status de pinos ou dispositivos (o que seria ineficiente). Elas são muito mais eficientes do que usar "*polling*", onde o processador precisa verificar periodicamente o status de um dispositivo ou condição.

5.2.4.2 Interrupção do tipo TIMER

Uma interrupção do tipo *timer* é um tipo específico de interrupção gerada por um temporizador interno do microcontrolador. O timer é um periférico que conta incrementos baseados ciclo de *clock* do microcontrolador (relógio interno), e quando atinge um valor pré-configurado, ele dispara uma interrupção. Esse tipo de interrupção é amplamente utilizado em sistemas embarcados para realizar tarefas em intervalos regulares de tempo, sem precisar depender do programa principal para controlar o tempo.

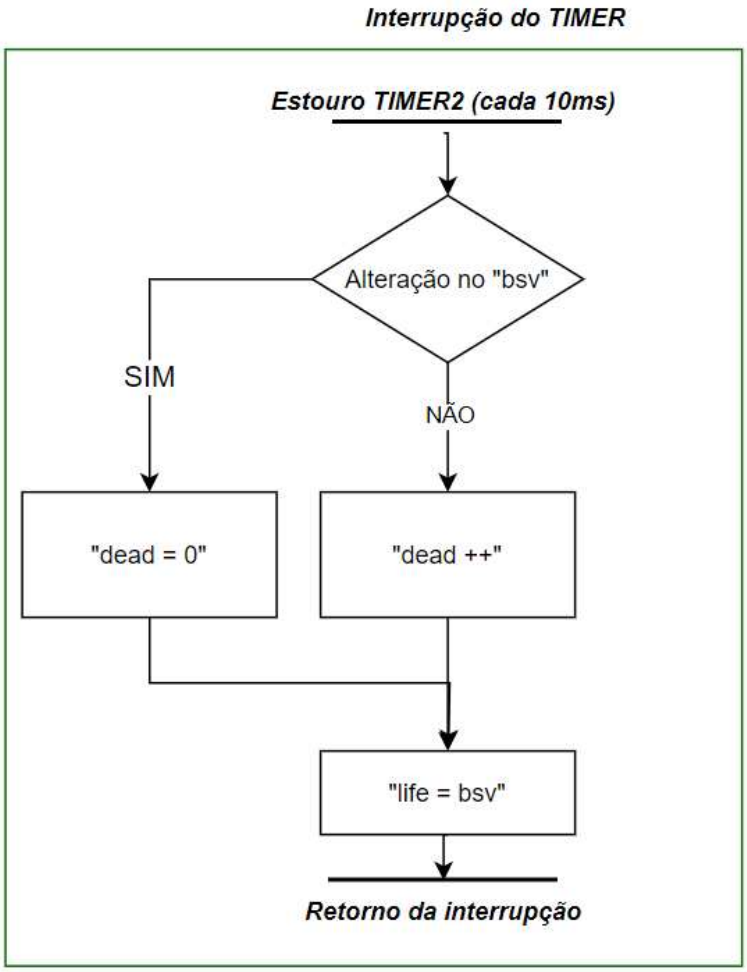
Nesse caso específico, existem 3 *timers* no STM32F334R8. O timer utilizado é o *TIMER2* pré-configurado para disparar uma interrupção a cada 10ms, tendo prioridade sobre a rotina principal, ou seja, independentemente do estado atual da rotina de comunicação, a rotina de tratamento de interrupção *TIMER2* será executada e, após a desativação da *flag* da interrupção, o microcontrolador volta a executar a rotina principal.

5.2.5 Algoritmo da interrupção de verificação de conexão da solução 2

Em resumo, interrupção TIMER2 de período 10 ms, verifica periodicamente se o *bsv* alterna continuamente durante os ciclos entre 1 e 0 ou vice-versa. Quando a interrupção é inicializada, o *bsv* da atual mensagem recebida será comparado com o valor armazenado na variável *life*, que é responsável armazenar o *bit* de sinal de vida a interrupção anterior. Se o valor desse *bit* tiver mudado, isso indica que há uma comunicação em andamento. Caso contrário, a variável *dead* será incrementada em 1 unidade. Se essa falta de alteração ocorrer por 10 interrupções consecutivas, o relé 1 que funciona como uma espécie de chave geral para todos os barramentos CC será aberto por motivos de segurança assim que rotina retornar ao *loop* principal. Caso, a comunicação volte antes de *dead* atingir as 10 unidades, a variável será

zerada e a interrupção continuará normalmente. O diagrama da figura 35 resume essa parte do algoritmo:

Figura 35 - Fluxograma do algoritmo de verificação de comunicação



Fonte: O próprio Autor

5.2.5.1 Resultados parciais algoritmo da interrupção de verificação de conexão

O teste utilizado para verificar o funcionamento parcial da rotina de verificação de conexão foi retirada manualmente do *jumper* de comunicação recepção do *SpeedGoat* para o STM32. Esse teste tem como objetivo verificar o acionamento da abertura do relé geral e o tempo de resposta do sistema diante da desconexão da comunicação.

Para a contagem do tempo entre a percepção da desconexão da comunicação e o acionamento do estado de segurança, utilizou a contagem do TIMER1, um temporizador preciso pré-programado para tempo base de 10 µs. Dentro da rotina principal, o temporizador

inicia a contagem a partir do momento que a variável *dead* tiver o primeiro incremento, devido a uma detecção da queda da comunicação. Assim, a contagem será encerrada a partir do momento que a variável "danger" atingir as 10 unidades e o será mostrado na tela o intervalo de tempo medido pelo temporizador.

Foram feitos 8 testes de detecção de falha e os tempo resultados entre a detecção e a verificação da perda de comunicação. Os resultados parciais são mostrados na tabela 8:

Tabela 8 - Tempo de acionamento do estado de proteção desde a primeira mensagem inválida recebida

Teste	Intervalo de tempo (ms)	Estado de proteção
1	101	SIM
2	101	SIM
3	101	SIM
4	106	SIM
5	101	SIM
6	101	SIM
7	106	SIM
8	106	SIM

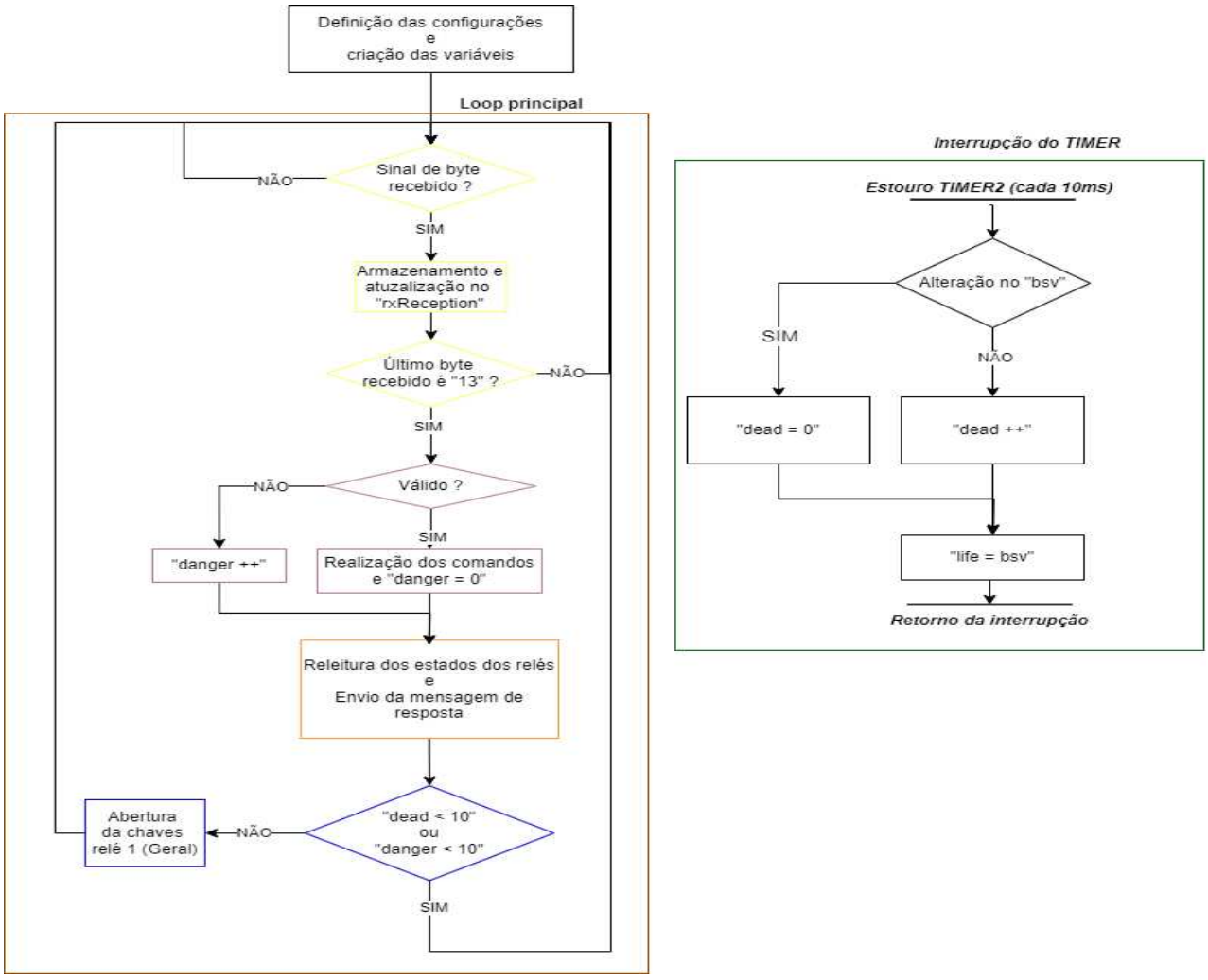
Fonte: O próprio Autor

Portanto, os resultados parciais mostram que o tempo de reação do sistema a queda de conexão é de em média 103 ms, o que é compatível com a expectativa uma vez que a ordem de grandeza de 100 ms é tempo de 10 interrupções do TIMER2 de 10 ms, referente ao acionamento do estado de proteção.

5.2.6 Algoritmo geral da solução 2

A figura 36 ilustra o comportamento geral do algoritmo, integrando o algoritmo de verificação de conexão e o algoritmo de recepção e envio. Como já explicado anteriormente, o algoritmo se resume a definição de configurações, recepção de *bytes*, validação, leitura dos estados dos relés, reenvio de mensagem de estados e análise das variáveis de segurança *dead* e *danger*. Em paralelo o TIMER2 ativa a interrupção de verificação de conexão a cada 10 ms, verifica a mudança do *bsv* e atualiza a variável *dead.*

Figura 36 - Fluxograma geral do algoritmo de comunicação



Fonte: O próprio Autor

Dessa forma, caso algum problema dos dois tipos ocorra, a chave de segurança 1 será aberta para evitar problema. Futuramente se a conexão for restabelecida e a mensagem voltar a não apresentar mais problemas de validação, o comando inserido na própria mensagem será responsável por fechá-la novamente para prosseguir os testes. Caso contrário, o algoritmo fica em espera no *loop* de recepção de *bytes*.

6 VALIDAÇÃO FINAL DA SOLUÇÃO PROPOSTA

Para validar de forma total o protocolo de comunicação entre o controlador mestre e o escravo, é necessário verificar o funcionamento do controlador realizando as tarefas de comunicação e verificação de forma alternada.

Foi realizado um experimento no laboratório para verificar se cada mensagem recebida pelo STM32 era recebida e traduzida corretamente e se a mensagem enviada de volta ao *SpeedGoat* era consistente com os estados dos relés. Para isso, foram realizados 2 tipos de validação: uma em interface de controle construída via Simulink Matlab e outra por meio do *debugger* da *CubeIDE* do STM32. Como já mencionado anteriormente, mensagem de recepção e envio possuem o mesmo formado de 5 *bytes* que foi citada anteriormente com ilustra a figura 25. Assim, para o protocolo de comunicação ser validado, todos os objetivos descritos no tópico 4.4 devem ser atingidos por ambos os lados de comunicação, ou seja, mestre e escravo.

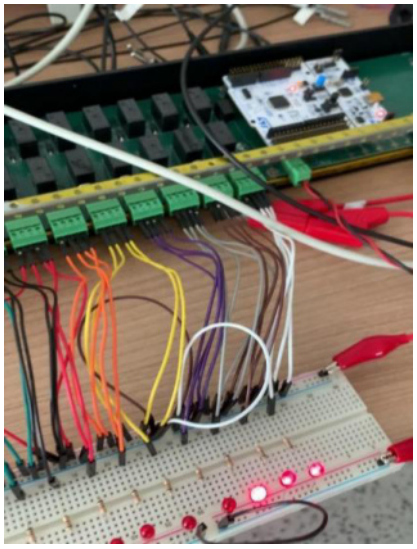
6.1 Validação via Simulink Matlab

No software Simulink, são analisados os valores em tempo real os valores de cada byte enviado e recebido da mensagem, o estado dos relés e os erros detectados durante a validação. Dessa forma, o resumo de cada gráfico analisado é o seguinte:

- *RELAY BOARD 1 - 10*: Status de cada relé na mensagem de retorno;
- B_input_validated: Consistência entre o comando enviado o estado dos relés;
- N_cnt_input1_diff_input2: Validação da formatação mensagem enviada pelo STM32;
- N_cnt_fit_pb_on_TX_message: Validação da formatação mensagem enviada pelo *SpeedGoat*;
- N_cnt_fit_lifesign: Numero de troca de estados do *bit* de sinal de vida em um período de tempo de 150 s;
- N_cnt_fit_message_configuration: Número de mensagens enviadas em um período de 150 s.

Para ajudar a verificar a os estados do GPIO's da placa relé, utilizou-se uma protoboard e 4 jumpers de cores diferentes, um resistor de 150 Ω e um LED vermelho para cada relé. Portanto, o objetivo é observar se cada relé foi fechado ou aberto usando os LEDs.

Figura 37 - Circuito montado para a validação visual do acionamento dos relés



Fonte: O próprio Autor

A mensagem enviada a placa relé pelo *simulink* são de tal forma que os seguintes comandos descritos na tabela 9 devem ser convertidos em estados dos relés.

Tabela 9 - Estados a serem executados com os comandos de teste

Relé	**Estado do relé**
Relé 1	Fechado (Estado 1)
Relé 2	Fechado (Estado 1)
Relé 3	Fechado (Estado 1)
Relé 4	Fechado (Estado 1)
Relé 5	Fechado (Estado 1)
Relé 6	Fechado (Estado 1)
Relé 7	Fechado (Estado 1)
Relé 8	Fechado (Estado 1)
Relé 9	Aberto (Estado 0)
Relé 10	Aberto (Estado 0)

Fonte: O próprio Autor

Assim, após a conexão dos equipamentos e inicialização da simulação, foram obtidos os seguintes resultados na figura 38 em forma de gráficos no *simulink*:

Figura 38 - Resultados da validação pelo Simulink Matlab

Fonte: O próprio Autor

Os resultados da validação via Simulink demonstram que os estados dos relés de 1 a 10 ilustrados pelos gráficos *RELAY_BOARD* são condizentes com os estados esperados da tabela 10 onde o eixo das abscissas representa a quantidade de mensagens enviadas até então. Além disso, durante a validação, não se observou nenhum ruido na manutenção desses estados durante mais de 30 minutos.

Para assegurar a comutabilidade dos estados dos relés, foram realizados testes em que cada estado dos relés de 1 a 10 foi comutado individualmente entre aberto (0) e fechado (1), o que confirmou a individualidade de cada relé em relação aos comandos enviados pelo controlador principal.

Além disso, observa-se em *B_input_validated* (Consistência dos estados com os comandos), *N_cnt_input1_diff_input2* (Validação da formatação mensagem enviada pelo STM32) e *N_cnt_fit_pb_on_TX_message* (Validação da formatação mensagem enviada pelo

SpeedGoat) que todos os sinais registrados permaneceram com valor 1 (validado) e não mostram ruídos ou variações, ou seja, todos os dados foram recebidos, processados e reenviados com sucesso sem a ocorrência de qualquer tipo erro e de forma síncrona, em outras palavras, os relés permaneceram no estado desejado durante os 30 minutos observados.

A variável *N_cnt_fit_lifesign* (Número de troca de estados do *bit* de sinal de vida em um período de tempo de 150 s) ilustra um valor de 15000 trocas do sinal de vida durante os 150 s, o que é condizente com o período de envio de mensagem de 10 ms.

A variável *N_cnt_fit_message_configuration* (Número de mensagens enviadas em um período de 150 s) ilustra um valor de 15000 mensagens enviadas durante os 150 s, o que é condizente com o período de envio de mensagem de 10 ms.

Finalmente, deixou-se validação via Simulink prosseguir por um período de 24 horas para analisar qualquer variação nos estados dos relés. Contudo, não foi observado, após esse período, nenhuma falha de comunicação ou mudança nos estados desejados dos relés. O que contribui para a validação dessa etapa de testes.

6.2 Validação tempo real via *CubeIDE* STM32

Nessa validação, o objetivo é observar o comportamento das variáveis por meio do advento do *debugger* da *CubeIDE*. Diferentemente da validação por simulink que foi utilizada o próprio *SpeedGoat* e a interface de comunicação RS232, essa categoria de validação foi feita com a utilização de um STM32F334R8 auxiliar que envia uma mensagem de 5 *bytes* padrão com os seguintes estados a serem executados como mostra a tabela 10:

Tabela 10 - Estados a serem executados com os comandos de teste da *CubeIDE*

Relé	Estado do relé
1	Fechado (Estado 1)
2	Fechado (Estado 1)
3	Fechado (Estado 1)
4	Fechado (Estado 1)
5	Fechado (Estado 1)
6	Aberto (Estado 0)
7	Aberto (Estado 0)
8	Aberto (Estado 0)
9	Aberto (Estado 0)
10	Aberto (Estado 0)

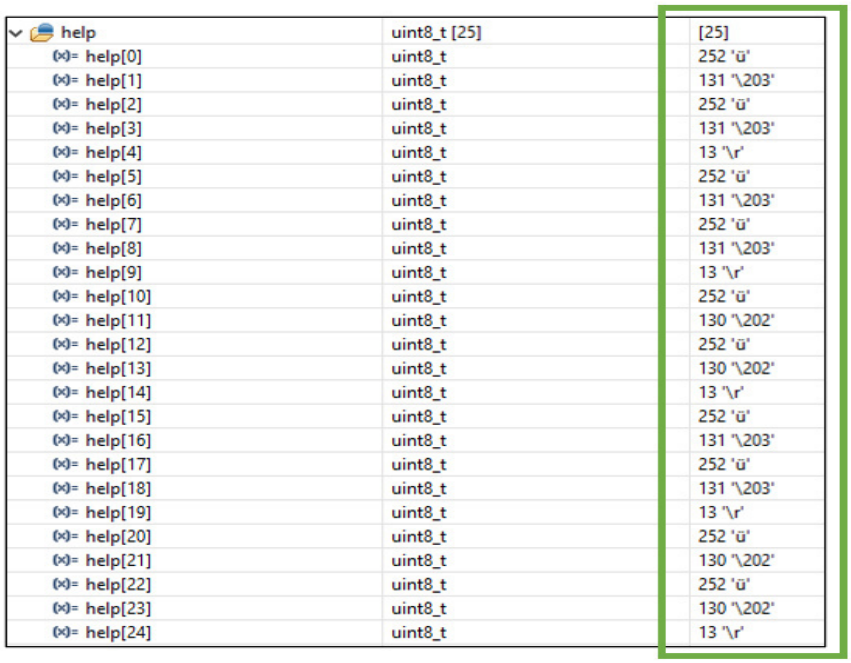
Fonte: O próprio Autor

Com relação aos dados processados pelo Cube Debug IDE, o comportamento de cada variável foi analisado, de modo que cada variável cumpra a seguinte função:

- command [0-4]: 5 *bytes* mensagem recebida (vetor *int*);
- dead: Número de vezes consecutivas que uma mensagem não é recebida (10 ms entre cada verificação);
- danger: Número de vezes consecutivas que uma mensagem recebida é inválida;
- help [0-24]: Amostra da fila de dados recebidos pelo registro RDR.

Após programar o auxiliar STM32 para enviar a mensagem através do protocolo *UART*, o STM32 principal receberá os dados e o fluxo de dados será mostrado na função *debug* da *CubeIDE*. Portanto, os dados mostrados na figura 39 é um recorte da interface:

Figura 39 - Mensagens recebidas em tempo real visualizadas via *debugger*



help	uint8_t [25]	[25]
help[0]	uint8_t	252 'ü'
help[1]	uint8_t	131 '\203'
help[2]	uint8_t	252 'ü'
help[3]	uint8_t	131 '\203'
help[4]	uint8_t	13 '\r'
help[5]	uint8_t	252 'ü'
help[6]	uint8_t	131 '\203'
help[7]	uint8_t	252 'ü'
help[8]	uint8_t	131 '\203'
help[9]	uint8_t	13 '\r'
help[10]	uint8_t	252 'ü'
help[11]	uint8_t	130 '\202'
help[12]	uint8_t	252 'ü'
help[13]	uint8_t	130 '\202'
help[14]	uint8_t	13 '\r'
help[15]	uint8_t	252 'ü'
help[16]	uint8_t	131 '\203'
help[17]	uint8_t	252 'ü'
help[18]	uint8_t	131 '\203'
help[19]	uint8_t	13 '\r'
help[20]	uint8_t	252 'ü'
help[21]	uint8_t	130 '\202'
help[22]	uint8_t	252 'ü'
help[23]	uint8_t	130 '\202'
help[24]	uint8_t	13 '\r'

Fonte: O próprio Autor

Observa-se que os dados dentro do quadrado verde são os bytes do pacote de mensagem armazenados no *buffer* em uma alternância das sequências “252, 131, 252, 131, 13” e “252,

130, 252, 130, 13”. A diferença entre essas duas sequências são os *bytes* 1 e 3 que sempre devem ser iguais dentro da uma mesma mensagem e alternam entre 131 e 130, pois, como esperado, o *bit* de sinal de vida (*bit* menos significativo dos *bytes* 1 e 3) estão alternativamente sendo trocados. Além disso, essa ilustração evidencia a importância do byte 4 ser constante e igual a 13, uma vez que, isso facilita a compreensão entre o final de uma mensagem para o começo de outra.

Uma perguntar pertinente na construção do formato da sequência de 5 *bytes* é: Se um byte com valor é o final de mensagem, o que impede qualquer um dos *bytes* entre 0 e 3 de ter valor também 13 e confundir o início da mensagem? A resposta está no fato desses outros *bytes* serem construídos para jamais terem valor 13, já que o seu BMS (*bit* mais significativo) é sempre a 1 como mostra a figura 40, diferentemente do byte 4 que possui BMS sempre igual a 0.

Figura 40 - *Bit*s de acionamentos dos relés após a validação da mensagem

(x)= (command[1] >> 4) & 0x01	int	0
(x)= (command[1] >> 5) & 0x01	int	0
(x)= (command[1] >> 6) & 0x01	int	0
(x)= (command[0] >> 0) & 0x01	int	0
(x)= (command[0] >> 1) & 0x01	int	0
(x)= (command[0] >> 2) & 0x01	int	1
(x)= (command[0] >> 3) & 0x01	int	1
(x)= (command[0] >> 4) & 0x01	int	1
(x)= (command[0] >> 5) & 0x01	int	1
(x)= (command[0] >> 6) & 0x01	int	1

Fonte: O próprio Autor

Para poder-se analisar os comandos dos relés após a mensagem ser validada e decodificada, utilizou-se o artifício da linguagem C de realizar operações lógicas binárias sobre uma variável. Logo, para diferenciar os *bit*s para cada relés utilizou a seguinte expressão em C:

(command[‘índice do byte a ser analisado’]) >> ’indice do bit naquele byte’ & 0x01

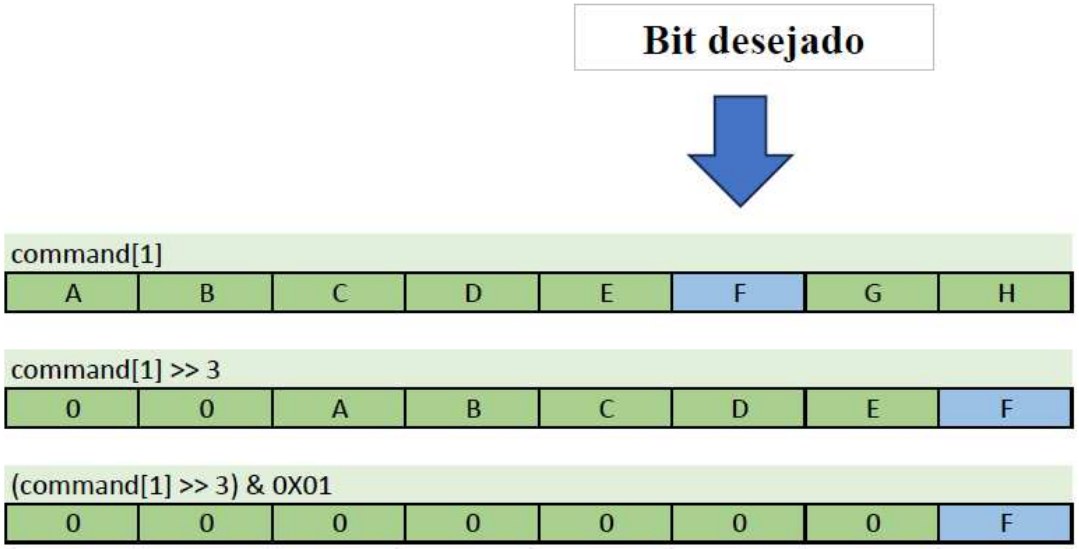
Tal que:

- Operação >> (right shift): Ela desloca os *bit*s de um número para a direita por uma quantidade especificada de posições;
- Operação & (AND): Realizada a operação lógica AND entre duas variáveis;

- 0x01: Essa é uma representação hexadecimal do número 1, forma mais simples de trabalhar em binário em C.

A figura 41 ilustra de uma forma melhor essa transformação para encontrar o *bit* correspondente.

Figura 41 - Ilustração das funções *bitwise* para decodificação dos *bit*s de comando



Fonte: O próprio Autor

Finalmente, analisa-se em tempo real a ocorrencia de erros por meio das varáveis *dead* que representam os erros por falta de comunicação e *danger* que representa os erros por mensagem inválida. Foi-se também inserida uma variável auxiliar *VAL* para contar qualquer tipo de ocorrência dos dois tipos de erros.

Figura 42 - Análise de falhas durante a validação

(x)= dead	int	0
> etat	uint8_t [5]	[5]
(x)= VAL	int	0
aux		Failed to evaluate expression
(x)= danger	int	0

Fonte: O próprio Autor

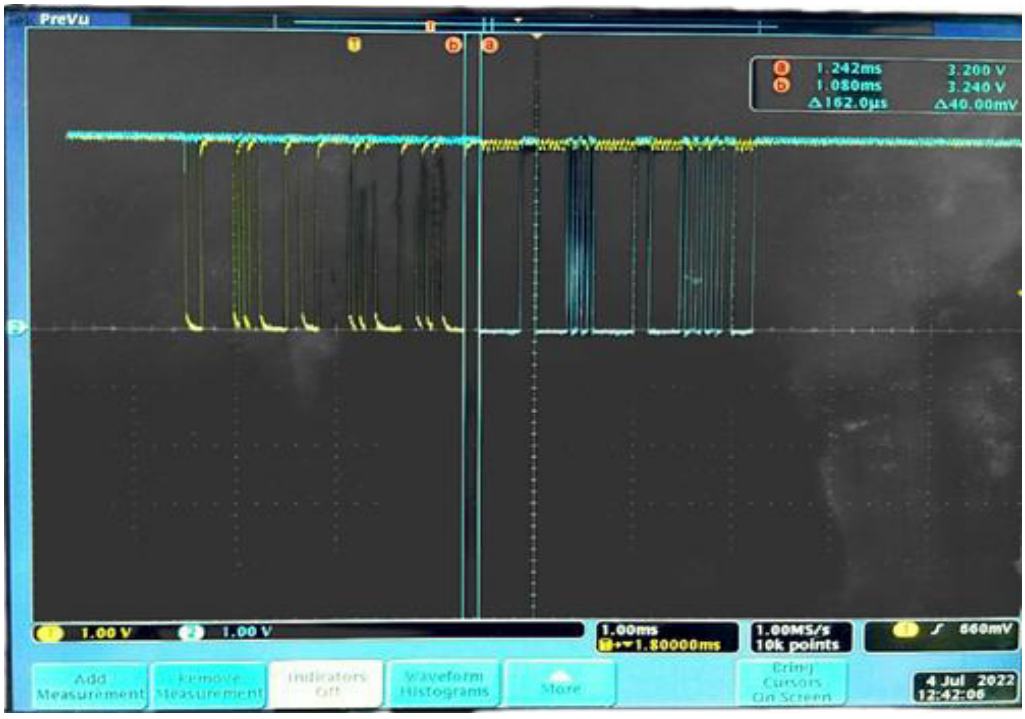
A figura 42 mostra os resultados das variáveis de monitoramento. As variáveis *dead* e *danger* não mostram nenhum erro instantâneo no momento da captura da imagem e *VAL* mostra

que não ocorreu qualquer tipo de erro mesmo que forma pontual, pois seu valor continua 0 mesmo após 1 hora de comunicação contínua. Logo, a validação dessa etapa de testes foi concluída e validada.

6.3 Validação em tempo real via osciloscópio

O osciloscópio de laboratório SuperGrid P3 foi usado para observar o sinal de entrada e saída no STM32 com o objetivo de analisar a tensão de saída da mensagem de recepção (amarelo) e retorno (azul), o sincronismo das mensagens e o atraso entre a recepção da mensagem e o retorno dos estados.

Figura 43 - Medida do intervalo de tempo entre recepção de mensagem e reenvio



Fonte: O próprio Autor

Como pode-se observar na figura 43, a tensão sobre os GPIO's varia entre 3,2 V e 3,24 V o que é um valor adequado tanto para recepção e envio de informação. Além disso, a medição aponta que o intervalo de tempo decorrido entre a recepção e o envio é aproximadamente 162,0 µs, o que é muito excelente comparado ao tempo máximo de 10 ms para realizar essa devolução de informação.

Figura 44 - Tempo entre cada mensagem em osciloscópio

Fonte: O próprio Autor

Finalmente, a figura 44 mostra que o período entre as tarefas de envio e recepção. Dessa forma, é possível ver que cada mensagem é enviada com um intervalo de 10 ms. Não foi observado nenhum ruído que pudesse comprometer a comunicação. Portanto, essa análise de tempo mostra que a escolha de uma transmissão de 19200 Baud/s foi uma ótima escolha par balancear velocidade de transmissão e sincronismo. Logo, a validação dessa etapa de testes foi confirmada.

CONCLUSÃO

Finalmente, pode-se concluir que os resultados mostrados na secção de validação demonstram que o protocolo de comunicação criado é funcional e suficientemente seguro para controlar os relés na bancada de envelhecimento do IGBT para futuros testes.

Os resultados coletados mostraram que o protocolo de comunicação continua de forma contínua, ininterrupta e consistente com os comandos recebidos. Mesmo no caso de uma interrupção na comunicação, o sistema é capaz de restabelecer a comunicação de forma autônoma, sem a ajuda de hardware externo ou da reinicialização do software.

Entretanto, algumas restrições devem ser mantidas para que a comunicação seja mantida de forma satisfatória. A velocidade mínima de comunicação UART deve ser de 19200 Bauds/s, pois abaixo disso pode ser gerado um estouro de dados, comprometendo todo o processo. É importante averiguar a corrente total de saída de Entrada e saída de acordo com o datasheet do STM32, pois segundo o mesmo deve ser de no máximo 80mA.

Além disso, alguns problemas na estrutura do próprio cartão de relé precisam ser corrigidos em uma nova versão do cartão. Primeiramente, a entrada e a saída do sinal RXUART (recepção) do conversor HIN202E estão cruzadas na trilha da placa de circuito impresso. Além disso, a caixa que armazena o cartão não é grande o suficiente para comportar todo a placa relé, o que evidencia a necessidade de refazê-la. Finalmente, existe um problema de isolamento entre a caixa de armazenamento e o cartão, ou seja, caso a superfície de apoio seja condutora, fatores externos podem influenciar erroneamente no sistema.

Em geral, o a *CubeIDE* é um excelente ambiente de programação pois, além de ser uma interface para programação em um alto nível de abstração, permitindo a programação a um nível de baixo nível de abstração com a atuação direta sobre registradores e com suporte para *debugger*.

A nível de programação, o processo de análise de resultados foi dificultado, pois o programador não atribuiu nomes condizente as funções das variáveis, pois essa escolhe de nomes simples facilita a inserção dos nomes nos fluxogramas do código.

Dessa forma, os resultados realizados pela interface *Matlab,* interface *CubeIDE* e pela análise do osciloscópio demonstram que a comunicação entre os dois dispositivos na bancada é assegurada de forma eficaz e não compromete as outras tarefas pertinentes ao controlador mestre.

BIBLIOGRAFIA

[1] MINGXING DU; QINGYI KONG; ZIWEI OUYANG; KEXIN WEI; WILLIAM G. HURLEY **Strategy for Diagnosing the Aging of an IGBT Module by ON-State Voltage Separation**. IEEJ Journal of Industry Applications, v. 7, n. 3, p. 174–182, 2019.

[2] BAO, Y.; JIANG, Q. **Summary of Life Prediction and Failure Analysis of IGBT Modules Based on Accelerated Aging Test**. IEEE Transactions on Power Electronics, v. 31, n. 22, p. 4025-4040, 2019.

[3] SMET, V; FOREST, F; HUSELSTEIN, J; RICHARDEAU, F; KHATIR, Z; LEFEBVRE, S; BERKANI, M. **Ageing and Failure Modes of IGBT Modules in High-Temperature Power Cycling. Microelectronics Reliability**, vol. 58, n. 10, 2011.

[4] WANG, W.; MAHMOUDI, M.; TENG, Y.; GAO, W. **Dynamic Surface Backstepping Control for Voltage Source Converter-High Voltage Direct Current Transmission Grid Side Converter Systems**. IEEE Access, v. 7, p. 56096-56106, 2019.

[5] STMICROELECTRONICS. **STM32F334 Datasheet**. 16 jul. 2018. Disponível em: https://www.st.com/resource/en/datasheet/stm32f334r8.pdf. Acesso em: 8 out. 2024.

[6] HYMEL, S. **Getting Started with STM32 - Introduction to STM32CubeIDE**. 24 jul. 2020. Disponível em: https://www.digikey.com/en/maker/projects/getting-started-with-stm32-introduction-to-stm32cubeide/6a6c60a670c447abb90fd0fd78008697. Acesso em: 8 out. 2024..

[7] STMICROELECTRONICS. **STM32F334xx Advanced Arm®-Based 32-*Bit* MCUs**. Jun. 2020. Disponível em: https://www.st.com/resource/en/reference_manual/dm00093941-stm32f334xx-advanced-arm-based-32-*bit*-mcus-stmicroelectronics.pdf. Acesso em: 8 out. 2024

[8] **STMICROELECTRONICS. Description of STM32F3 HAL and Low-Layer Driver**s. Fev. 2020. Disponível em: https://www.st.com/resource/en/user_manual/um1786-description-of-stm32f3-hal-and-lowlayer-drivers-stmicroelectronics.pdf. Acesso em: 8 out. 2024.

[9] BARR, M. **Programming Embedded Systems: With C and GNU Development Tools**. 2. ed. O'Reilly Media, 2006. ISBN 978-0-596-00983-0.

[10] Arthur Boutry, Cyril Buttay, Dong Dong, Rolando Burgos, Bruno Lefebvre, et al.. **Figures-of-Merit and current metric for the comparison of IGCTs and IGBTs in Modular Multilevel Converters**. EPE'20 ECCE Europe, Sep 2020, Lyon, France.

[11] INFINEON TECHNOLOGIES AG. **IGBT module FF150R12KE3G datasheet**. Neubiberg, Alemanha, 2019. Disponível em: https://www.infineon.com/cms/en/product/power/igbt/igbt-modules/ff150r12ke3g/. Acesso em: 7 nov. 2024.

[12] WIDAP. **Datasheet do switch LTC002501*A02**. Disponível em: <https://www.widap.com/wp-content/uploads/artikel_doc265_72.pdf>. Acesso em: 4 dez. 2024.

[13] ONSEMI. **BSS138K: Small Signal MOSFET. Datasheet**. Disponível em: <https://br.mouser.com/ProductDetail/onsemiFairchild/BSS138K?qs=kDD%2FdQe9TTeCJ7OVuffPnA%3D%3D>. Acesso em: 21 nov. 2024.

[14] OMRON. **G5Q-1-EU: PCB Power Relay. Datasheet**. Disponível em: https://www.omron.com/ecb/products/pdf/en-g5q.pdf. Acesso em: 21 nov. 2024.

[15] LI, Lie; HE, Yigang; WANG, Lei; WANG, Chenyuan; WANG, Chuankun; WU, Xiaoxin. **The aging analysis method of IGBT with composite failure mode based on Miner linear fatigue cumulative theory. Microelectronics Reliability**, v. 122, p. 114165, 2021.ISSN0026-2714.Disponível em: https://www.sciencedirect.com/science/article/pii/S0026271421001311.
Acesso em: 3 dez. 2024. DOI: https://doi.org/10.1016/j.microrel.2021.114165.

[16] GE, Jianwen; HUANG, Yixiang; TAO, Zhiyu; LI, Bingch; XIAO, Dengyu; Yanming, LI; **RUL PRUL Predict of IGBT Based on DeepAR Using Transient Switch Features**. Minhang, Shanghai: Shanghai Jiao Tong University; Yangpu, Shanghai: University of Shanghai for Science and Technology, [s.d.]. Disponível em: <https://doi.org/10.36001/phme.2020.v5i1.1234 >. Acesso em: 04 dez. 2024.