UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUIS FELIPE SOTO LAURO
LUIZ AUGUSTO BERNARDI

UNIDADE DE TELEMETRIA, ARMAZENAMENTO E VISUALIZAÇÃO DE DADOS EMBARCADO EM UM CARRO DE FÓRUMLA SAE

CURITIBA
2021

LUIS FELIPE SOTO LAURO
LUIZ AUGUSTO BERNARDI

UNIDADE DE TELEMETRIA, ARMAZENAMENTO E VISUALIZAÇÃO DE DADOS EMBARCADO EM UM CARRO DE FÓRUMLA SAE

Telemetry, storage and display unit embedded in a Formula Student Car

Trabalho de conclusão de curso de graduação apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia Eletrônica do Departamento de Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Hugo Vieira Neto.
Coorientador: Prof. Dr. Bruno Sens Chang.

CURITIBA
2021

LUIS FELIPE SOTO LAURO
LUIZ AUGUSTO BERNARDI

UNIDADE DE TELEMETRIA, ARMAZENAMENTO E VISUALIZAÇÃO DE DADOS EMBARCADO EM UM CARRO DE FORMULA SAE

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 17/Novembro/2021

_____

Bruno Sens Chang
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Luis Fernando Copetti
Mestrado
Universidade Tecnológica Federal do Paraná

_____

Guilherme de Santi Peron
Doutorado
Universidade Tecnológica Federal do Paraná

CURITIBA

2021

Dedico este trabalho àqueles que me guiaram nesta jornada.

- Luis Felipe Soto Lauro

Dedico este trabalho àqueles cuja confiança em mim me permitiu chegar tão longe.

- Luiz Augusto Bernardi

AGRADECIMENTOS

Ao nosso orientador, professor Dr. Hugo Vieira Neto (*in memoriam*), por ter acreditado neste projeto e em nosso potencial para cumpri-lo, pela sua dedicação e altruísmo com o partilhar do conhecimento. Somos privilegiados pelos tantos momentos em que a sua luz iluminou o nosso caminho.

Ao nosso co-orientador, professor Dr. Bruno Sens Chang, que com grande carinho aceitou o desafio de nos guiar na reta final deste projeto. Agradecemos pela sensibilidade e respeito pelo trabalho aqui desenvolvido, pela presteza e assertividade nas suas orientações.

À equipe Fórmula UTFPR, em especial ao Lucas Franco, capitão do subsistema de Elétrica e Eletrônica embarcada. Agradecemos por abrirem o espaço para o nosso projeto e por oferecer o suporte da equipe para o modelagem do barramento CAN.

Aos nossos pais, que por vezes sentiram a nossa ausência enquanto nos dedicávamos a este projeto. Sua compreensão e incentivo foram determinantes para que aqui chegássemos.

À Francielle Tavares Bernardi e Gabriel Mendes Simoni, pela dedicação com que nos auxiliaram na finalização deste projeto, sem a sua ajuda essa jornada teria sido mais difícil.

Aos nossos colegas de curso de Engenharia Eletrônica, em especial àqueles que se tornaram amigos.

O que redes de trens, canais e estradas foram outrora, redes de comunicação, informação e computadorização são hoje.

(**KREISKY, Bruno**)

Chanceler da Austria 1970 - 1983

Estamos nos aproximando do estágio em que os problemas que precisamos resolver se tornarão insolviveis sem computadores. Eu não temo os computadores. Eu temo a falta deles.

(**ASIMOV, Isaac**)

RESUMO

LAURO, Luis Felipe Soto; BERNARDI, Luiz Augusto. **Unidade de Telemetria, Armazenamento e Visualização de dados embarcada em um carro de Fórmula SAE**. 2021. 116 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) – Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

A Associação dos Engenheiros Automobilísticos promove desde 1981 a competição de *Fórmula SAE*, proporcionando a jovens engenheiros a oportunidade de desenvolver um projeto complexo e multidisciplinar: o desenvolvimento de um carro de competição. A busca por ferramentas e métodos adequados de teste e validação é fundamental para o desenvolvimento de projetos complexos. Neste trabalho, desenvolvemos um sistema de telemetria para apoiar o teste e suportar o processo de validação de um carro de Fórmula SAE. Utilizamos uma placa de desenvolvimento STM32F469I-DISCOVERY como unidade de processamento embarcada e um par de rádios da série CC13XX da Texas Instruments para fornecer um meio de comunicação sem fio com uma estação remota, onde um aplicativo *desktop* baseado em Qt foi usado como interface gráfica de usuário (GUI). Após validar a ferramenta proposta em um cenário relevante, concluímos que ela foi capaz de coletar, registrar e exibir (local e remotamente) dados relevantes sobre o desempenho do carro. Esta ferramenta é de grande relevância para o desenvolvimento contínuo do projeto, especialmente para o monitoramento e validação de todos os diferentes subsistemas que o compõem.

Palavras-chave: Sistema embarcado. Eletrônica automotiva. Telemetria. Fórmula SAE.

ABSTRACT

LAURO, Luis Felipe Soto; BERNARDI, Luiz Augusto. **Telemetry, Storage and display unit embedded in a Formula Student car.**. 2021. 116 p. Bachelor Thesis (Bachelor's Degree in Electronic Engineering) – Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

The Society of Automotive Engineers has been promoting the *Formula SAE* competition since 1981 and gives young engineers the opportunity to take part in a complex and multidisciplinary project: the development of a competition car. Finding suitable tools and methods that enable validation through testing is fundamental to developing such complex projects. In this thesis, we develop a telemetry system to support the testing and validation of a Formula SAE monoseat car. We used a STM32F469I-DISCOVERY development board as an embedded processing unit and two CC13XX series radios from Texas Instruments to enable wireless communication with a remote station. To provide a user interface for data analysis, we developed a desktop application using the Qt framework. After validating the proposed tool in a relevant scenario, we found that it was able to collect, register and display (both locally and remotely) relevant data about the vehicle's performance. This is of great importance for the continuous development of the project, especially for the monitoring and validation of all of its subsystems.

Keywords: Embedded system. Automotive Electronics. Telemetry. Formula SAE.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

SIGLAS

ABS	*Acrilonitrila Butadieno Estireno*
ACK	*Acknowledge*
CRC	*Cyclic Redundancy Check*
CSMA/CD	*Carrier Sense Multiple Access Collision Detection*
DBC	*DataBase Container*
DLC	*Data Length Code*
DMA	*Direct Memory Access*
DSI	*Display Serial Interface*
ECPA	*Esporte Clube Piracicabano de Automobilismo*
ECU	Unidade de Controle Eletrônico, do inglês *Electronic Control Unit*
EoF	*End of Frame*
F1	*Fórmula 1*
IDE	Ambiente Integrado de Desenvolvimento, do inglês *Intregrated Development Environment*
IDE	*Identifier Extension*
IFS	*Interframe Space*
IoT	Internet das coisas, do inglês *Internet of Things*
ISM	*Industrial, Scientific and Medical*
LCD	*Liquid Crystal Display*
LNA	*Amplificador de baixo ruído, do inglês Low Noise Amplifier*
MDE	Modelo Digital de Elevação
MMC	*Multimedia Card*
MVC	*Model-View-Controller*
MVP	*Model-View-Presenter*
PA	*Amplificador de Potência, do inglês Power Amplifier*
PDF	Formato de Documento Portátil, do inglês *Portable Document Format*
RF	*Rádio Frequência*
RSSI	Indicação de Força do Sinal Recebido, do inglês *Received Signal Strength Indicator*
RTC	*Real Time Clock*
RTOS	Sistema Operacional em Tempo Real, do inglês *Real-time Operating System*
RTR	*Remote Transmission Request*

SD	*Secure Digital*
SDHC	*Secure Digital High Capacity*
SDK	*Software Development Kit*
SDSC	*Secure Digital Standard Capacity*
SDUC	*Secure Digital Ultra Capacity*
SDXC	*Secure Digital Extended Capacity*
SO	Sistema Operacional
SoF	*Start of Frame*
TRL	Nível de Prontidão Tecnológica, do inglês *Technology Readiness Level*
UTFPR	Universidade Tecnológica Federal do Paraná
VCU	Unidade de Controle Veicular, do inglês *Vehicle Control Unit*

ACRÔNIMOS

ADAS	*Advanced Driver-Assistance*
BIOS	*Basic Input/Output System*
CAN	Controller Area Network
DIP	Encapsulamento Duplo em Linha , do inglês *Dual In-line Package*
DRAM	*Dynamic Random-access Memory*
FIA	Federação Internacional de Automobilismo, do francês *Fédération Internationale de l'Automobile*
FIEP	Federação das Indústrias do Estado do Paraná
GUI	Interface Gráfica do Usuário, do inglês *Graphical User Interface*
HAL	Camada de Abstração de *hardware*, do inglês *Hardware Abstraction Layer*
IPPUC	Instituto de Pesquisa e Planejamento Urbano de Curitiba
ISO	Organização Internacional de Normalização, do inglês *International Standardization Organization*
LIDAR	*Light Detection And Ranging*
LIN	Local Interconnetc Network
LoS	Linha de visão, do inglês *Line of Sight*
MOST	Media Oriented Systems Transport
OSI	*Open System Interconnection*
RAM	*Random-access Memory*
SAE	Sociedade de engenheiros automotivos, do inglês *Society of Automotive Engineers*
SoC	Sistema em um *chip*, do inglês *System on Chip*
SRAM	*Static Random-access Memory*

SUMÁRIO

1 INTRODUÇÃO

A Sociedade de engenheiros automotivos, do inglês *Society of Automotive Engineers* (SAE), fomenta desde 1976 eventos estudantis para a promoção de competições relacionadas a área de mobilidade. O primeiro evento deste tipo foi a competição de protótipos Mini-BAJA SAE, sediada na Universidade da Carolina do Sul. A modalidade BAJA SAE viria a se tornar sinônimo de competição de estudantes de engenharia nas décadas seguintes. (MEANS, 2001) Esta modalidade, que empresta o nome da corrida mexicana de veículos *off-road* (Baja 1000) desafia grupos de estudantes a projetar e construir um protótipo automotivo robusto, capaz de suportar as provas e testes de resistência aplicados em um ambiente similar a um Rally.

A modalidade Fórmula SAE foi introduzida em 1981, como modalidade de contraposição à BAJA SAE, fomentando o desenvolvimento de protótipos automotivos para competirem dentro das pistas. Na primeira edição do evento, sediada na Universidade do Texas, quatro equipes foram inscritas e apresentaram seus protótipos. Após 16 edições, em 1996, a competição regional norte-americana contabilizou mais de 75 equipes inscritas, comprovando o sucesso do evento (CASE, 1996).

Atualmente, a competição é organizada em etapas regionais (Alemanha, Austrália, Brasil, Inglaterra e Itália) e uma etapa global nos Estados Unidos, onde são reunidas as equipes vencedoras das etapas locais. O Brasil faz parte do circuito global de competições desde o ano de 2004.

Nesta modalidade, os estudantes são solicitados a aplicar na prática os conhecimentos adquiridos em sala de aula a fim de desenvolver o projeto completo de um carro do tipo Fórmula (monoposto, de cockpit aberto, desenvolvido para competições em autódromos). Os projetos desenvolvidos são avaliados por uma equipe de especialistas associados à SAE, que os submetem a testes estáticos e dinâmicos durante os três dias do evento.

O objetivo da competição é oferecer ao estudante o contato com um projeto desafiador, multi-disciplinar desenvolvido em equipe, assemelhando-se com o ambiente do mercado de trabalho das grandes empresas do ramo automotivo. Para o aluno, trata-se de uma oportunidade de ganhar visibilidade frente às grandes empresas, agregando ao seu currículo conhecimento técnico relevante para a área automotiva e fortalecendo

suas habilidades interpessoais. Para as empresas, é um ambiente fértil para a captura de novos talentos, já familiarizados com os fundamentos da engenharia automobilística.

A Universidade Tecnológica Federal do Paraná (UTFPR) campus Curitiba é uma instituição de presença constante nas competições de Fórmula SAE. A primeira participação foi em 2007, na 3ª Competição de Fórmula SAE, representada pela equipe UTFPRacing. Em 2009 a equipe, até então formada somente por alunos do curso de Engenharia Mecânica, foi reformulada, passando a se chamar Equipe TPR e abrindo vagas para alunos dos demais cursos da instituição (Design, Engenharia Elétrica, etc.). A partir de 2011, a Equipe Imperador de Baja SAE incorporou a equipe de Fórmula SAE, destacando-se como uma das poucas equipes a participar simultaneamente das duas categorias.

Ao final de 2013, no entanto, as dificuldades financeiras impuseram à equipe a decisão de congelar o projeto do protótipo de Fórmula SAE em detrimento do protótipo Baja SAE (BARATA, 2014). Com isso, a universidade ficou sem uma equipe de Fórmula SAE até o ano de 2016, quando um grupo de alunos dos cursos de Engenharia Mecânica, Elétrica e Eletrônica se uniram para estabelecer a Equipe Fórmula UTFPR, que representa, desde então, a instituição. A Figura 1 apresenta o protótipo EK-304 da equipe Fórmula UTFPR.

Figura 1 – Protótipo EK-304.

Fonte: Equipe Fórmula UTFPR.

Nestes mais de 13 anos de participação nas competições nacionais, o melhor resultado registrado para a universidade foi a 10ª colocação na competição de 2012 (CLASSIFICACAO. . . , 2012). Desde 2018 (ano de estreia da atual equipe Fórmula UTFPR na competição) a equipe evolui o seu desempenho, entretanto, o aumento da

competitividade da categoria, ilustrado principalmente pelo crescimento no número de equipes competidoras e pela qualidade dos protótipos apresentados, elevam o desafio de figurar entre as melhores equipes de Fórmula SAE a cada nova edição da competição, exigindo projetos cada vez mais complexos e multidisciplinares.

A complexidade inerente da integração de diversos subsistemas mecânicos (direção, transmissão, suspensão, etc.) no projeto automotivo mostrou-se um ambiente propício para o desenvolvimento de ferramentas de eletrônica embarcada, sobretudo no campo do sensoriamento eletrônico. No projeto automotivo de veículos de passeio, a integração de ferramentas eletrônicas deu-se, inicialmente, pela necessidade de controlar parâmetros da injeção de motores a combustão em tempo real. Em 1968 a Volkswagen integrou ao projeto do VW1600 o Bosch D-Jetronic, um novo dispositivo eletrônico de controle de injeção de combustível substituto ao usual carburador. (SENOUCI; ZITOUNI, 2016)

A partir de então, houve um crescimento exponencial do número de sistemas eletrônicos embarcados nos projetos automotivos, seja em substituição a elementos puramente mecânicos, eletro-mecânicos, hidráulicos ou ainda na introdução de novas funcionalidades de conforto, segurança e assistência ao piloto. Hoje, o projeto automotivo de um sedã de luxo apresenta, por exemplo, uma complexa estrutura de comunicação eletrônica que possibilita que mais de 70 Unidade de Controle Eletrônico, do inglês *Electronic Control Unit* (ECU) troquem informações para o monitoramento de até 250 sinais eletrônicos, desde velocidade e rotação do motor, até dados de sensores radar e de câmeras para aplicações de direção autônoma. (NAVET; SIMONOT-LION, 2009). A Figura 2 apresenta o leque de subsistemas envolvidos no projeto automotivo contemporâneo.

No campo dos esportes a motor, a categoria de *Fórmula 1* (F1) destaca-se como a ponta da lança no que diz respeito à pesquisa e desenvolvimento de tecnologias para aumentar o desempenho, confiabilidade e (mais recentemente) diminuir a emissão de gases nocivos ao meio ambiente. Diferente dos interesses e aplicações da indústria automobilística, a eletrônica embarcada aplicada aos esportes a motor é aplicada fundamentalmente no monitoramento em tempo real da performance do veículo, em detrimento das aplicações de conforto ou assistência ao piloto. A aplicação de monitoramento remoto dos parâmetros de performance do veículo é chamada telemetria.

Figura 2 – Subsistemas do projeto automotivo.

Fonte: Maurer e Winner (2013).

Introduzida inicialmente pela equipe Mclaren de Fórmula Indy em 1975, a primeira aplicação de monitoramento era capaz de monitorar 14 parâmetros da performance do veículo, porém apenas registrando localmente, tornando necessária a parada do carro na garagem para a descarga das informações registradas.(McLaren. . . , 2016) Somente em 1991 seria possível o envio de dados em tempo real do carro para a garagem da equipe (SULTANA, 2020).

Nos anos seguintes houve avanços na quantidade de parâmetros monitorados e na tecnologia de transmissão em tempo real. Na primeira metade da década de 1990 foram introduzidos uma variedade de recursos de eletrônica embarcada, tais como transmissão automática, controle de tração, suspensão ativa e telemetria bidirecional. A figura 3 mostra o centro de operações de telemetria da equipe Ferrari de F1 para o acompanhamento do desempenho dos protótipos durante um final de semana de competições.

Muitos dos recursos de eletrônica embarcada foram banidos da F1 pela Federação Internacional de Automobilismo, do francês *Fédération Internationale de l'Automobile* (FIA) ao longo dos últimos anos, como uma iniciativa para a diminuição dos custos da categoria e para recuperar o fator humano na disputa do campeonato. A categoria de Fórmula SAE, no entanto, não apresenta nenhuma restrição de funcionalidade para os recursos de eletrônica embarcada.

Atualmente, dado o avanço na tecnologia de eletrônica embarcada, é correto afirmar que o veículo de competição é um gerador de dados por excelência. Como tal,

Figura 3 – Central de telemetria da Scuderia Ferrari de F1 em meados de 2000.

Fonte: Tremayne (2009).

a utilização de aplicações de telemetria tem se tornado indispensável para o desenvolvimento do seu projeto, qualquer seja a sua categoria. Essas ferramentas permitem aos projetistas a validação da integração do projeto, bem como o teste rápido e interativo das configurações e ajustes mecânicos. Do ponto de vista do desenvolvimento contínuo, atuam ainda como elo de realimentação, fornecendo aos projetistas uma base de dados sólida para a tomada de decisão em projetos subsequentes.

1.1 OBJETIVOS GERAIS

Este trabalho objetiva o projeto e construção de um sistema de telemetria, armazenamento e visualização de dados de modo a atender as necessidades da equipe Fórmula UTFPR no contexto de competições de Fórmula SAE. A finalidade deste sistema é a constituição de uma ferramenta de integração e validação dos protótipos desenvolvidos pela equipe, tal que esta seja capaz de se tornar mais competitiva e aprimorar o seu projeto.

1.2 OBJETIVOS ESPECÍFICOS

- Permitir que o piloto acompanhe os parâmetros de desempenho do veículo construído pela equipe, em tempo real.

- Permitir que o time de engenharia acompanhe os parâmetros de desempenho do veículo remotamente, em tempo real e *a posteriori*.
- Construir um sistema com Nível de Prontidão Tecnológica, do inglês *Technology Readiness Level* (TRL) 6, isto é, um modelo demonstrador em ambiente relevante (WILLIAMSON; BEASLEY, 2011) (ISO, 2015).
- Desenvolver um sistema aberto para expansões futuras.

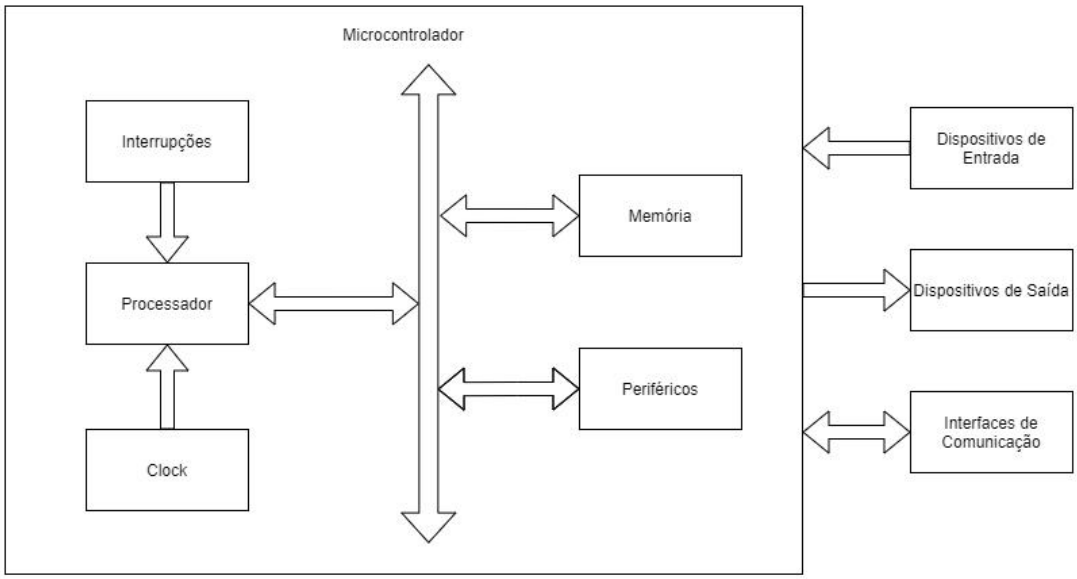
2 FUNDAMENTAÇÃO TEÓRICA

2.1 SISTEMAS EMBARCADOS

Um sistema embarcado é um sistema computadorizado construído especificamente para sua aplicação e está usualmente sujeito a restrições de *hardware* e *software*, como: baixo custo, pequena dimensão e baixo consumo de energia (WHITE, 2012). Além disso, estão sujeitos a padrões de qualidade mais altos visto que são frequentemente utilizados em aplicações críticas, tais como computadores de bordo de locomotivas e aeronaves.

Apesar de sistemas embarcados serem projetados com uma aplicação em mente, podem ser resumidos através dos componentes apresentados na Figura 4.

Figura 4 – Arquitetura geral de um sistema embarcado.



Fonte: Autoria própria.

Uma subcategoria de extrema importância são os sistemas em tempo real, que estão sujeitos a restrições de tempo, sendo necessário que eventos, internos ou externos, sejam tratados dentro de prazos previamente definidos. Os sistemas em tempo real podem ser classificados em dois grupos:

- **Não-Críticos**: sistemas onde falhas em atender os prazos são aceitáveis.

- **Críticos**: sistemas que devem operar de forma totalmente determinística e falhas em atender os prazos são totalmente inaceitáveis e correspondem a uma pane absoluta do sistema.

2.1.1 Sistema Operacional de Tempo Real

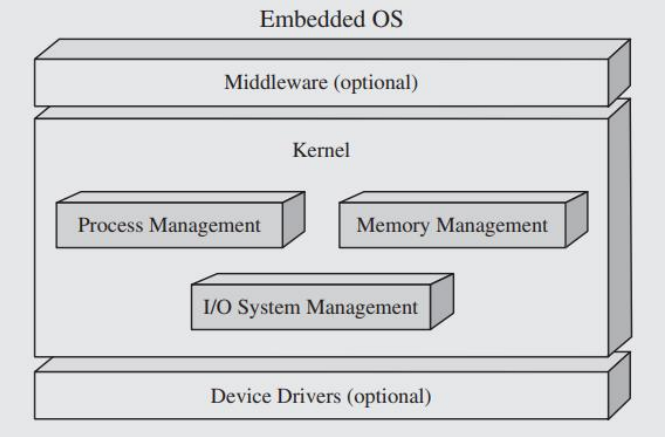
Antes de mais nada é interessante definirmos o que é, especificamente, um Sistema Operacional (SO). Um SO é um conjunto de bibliotecas de *software* que atende duas demandas em sistemas embarcados:

- Abstração de *hardware* para o *software* de aplicação executado pelo SO, permitindo uma menor dependência de *hardware*.
- Gerenciamento de *hardware* e recursos de *software* de forma a garantir que o sistema opere de forma eficiente.

Como apresentado na Figura 5, sistemas operacionais possuem diversos componentes, entretanto o único essencial é o *kernel*, responsável por fornecer as principais funcionalidades presentes em um SO:

- Gerenciamento de processos
- Gerenciamento de memória
- Gerenciamento de interfaces de entrada e saída

Figura 5 – Modelo genérico de um sistema operacional



Fonte: Noergaard (2013).

O SO gerencia e executa o *software* de aplicação através de tarefas, também denominadas processos, onde cada uma delas armazena todo o contexto de execução da mesma, como programa executado, valores dos registradores e *stack*. Além disso, alguns sistemas operacionais permitem que apenas uma tarefa exista, sendo necessário que todo o *software* embarcado seja executado dentro de uma única tarefa. Sistemas deste tipo são ditos monotarefas. Outros sistemas operacionais permitem que o *software* seja modularizado em múltiplas tarefas, sendo denominados multitarefas.

No contexto de sistemas operacionais multitarefas é necessário que se forneçam políticas e mecanismos para que o tempo do processador seja alocado entre as diferentes tarefas, causando assim a impressão de que todas as tarefas do sistema estão sendo executadas concorrentemente. Tal distribuição de tempo é alcançada através da comunicação, escalonamento e sincronização entre as tarefas. Do mesmo contexto surge a necessidade das outras duas principais funcionalidades, visto que a memória e os periféricos devem ser compartilhados entre as tarefas sistematicamente para que a aplicação opere de forma adequada e confiável (NOERGAARD, 2013).

Um Sistema Operacional em Tempo Real, do inglês *Real-time Operating System* (RTOS), além de possuir as funcionalidades tradicionais, oferece suporte ao desenvolvimento de sistemas em tempo real de forma simples e confiável (MARWEDEL, 2018). Desta forma, o RTOS permite que aplicações sejam criadas para operar de forma determinística e dentro de prazos firmemente estipulados. Para alcançar esse objetivo, os sistemas operacionais em tempo real possuem as características apresentadas abaixo (WANG, 2017):

- Latência mínima no atendimento de interrupções.
- Tarefas de maior prioridade devem causar a preempção de tarefas de menor prioridade a qualquer momento.
- Algoritmos de escalonamento avançados para atender às restrições temporais de sistemas em tempo real.
- Regiões críticas, onde ocorrem interações entre tarefas através de comunicação ou compartilhamento de recursos, devem ser as mais curtas possíveis diminuindo o risco de prejudicarem a preempvitidade do sistema.

Um exemplo de RTOS é o FreeRTOS, criado e mantido pela *Real Time Engineers Ltd.*. Este SO é um sistema de código aberto e nível comercial, voltado para microcontroladores e pequeno microprocessadores, que prioriza o baixo consumo de energia, a confiabilidade e a facilidade de uso (MASTERING. . . , 2016). Dentre as vantagens de uso do FreeRTOS, destacam-se:

- RTOS gratuito e sem taxas
- Qualidade e robustez são controladas pelo desenvolvedores
- Possui suporte dos desenvolvedores e da comunidade
- Fornece abstração de temporização
- Totalmente orientado à eventos
- Portado para diversos compiladores e arquiteturas

Por fornecer apenas o essencial para a construção de sistemas em tempo real, o FreeRTOS possui suporte à extensão de suas funcionalidades através de bibliotecas, permitindo que seja estruturado apenas com as funcionalidades necessárias para a aplicação desenvolvida. Sendo assim o FreeRTOS se mostra a plataforma ideal para o desenvolvimento do SnifferCAN do projeto.

Outro exemplo é o TI-RTOS, um RTOS desenvolvido pela Texas Instruments para aplicações em microcontroladores e microprocessadores comercializados por esta fabricante. Este sistema é apresentado como uma base sólida para o desenvolvimento de aplicações embarcadas em uma variedade de famílias de dispositivos, tais como os rádios para comunicação sem fio utilizados no projeto, tornando o desenvolvimento e manutenção mais simples (INSTRUMENTS, 2016).

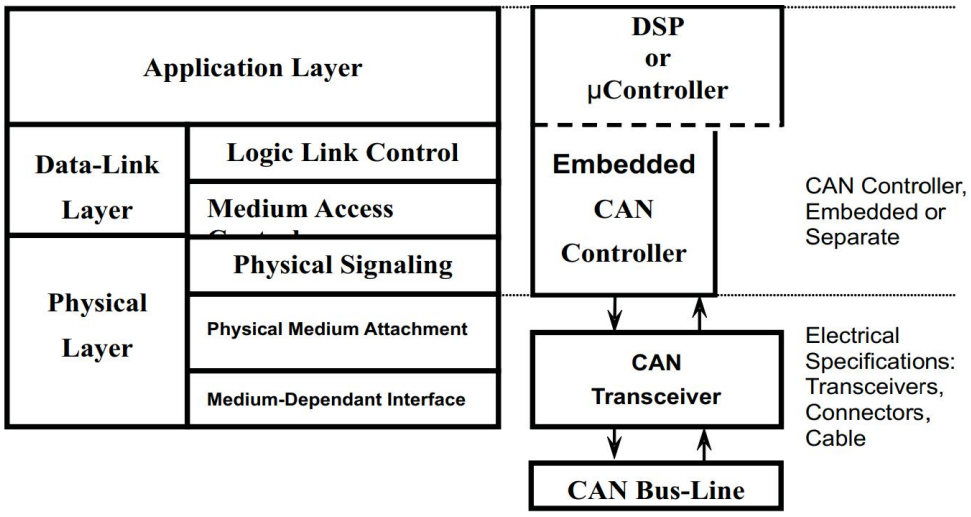
2.2 CAN

O barramento Controller Area Network (CAN) foi lançado na segunda metade da década de 1980 pela Robert Bosch GmbH, com intuito de substituir a fiação cada vez mais complexa nos projetos automotivos, fornecendo meios de comunicação entre dispositivos eletrônicos embarcados em veículos com alta imunidade à interferência elétrica e habilidade de autodiagnóstico (CORRIGAN, 2016). A arquitetura proposta

foi submetida para padronização internacional no início dos anos 1990, sendo logo publicada pela Organização Internacional de Normalização, do inglês *International Standardization Organization* (ISO).

O protocolo CAN (ISO-11898) é um protocolo multi-mestre baseado em mensagens, desenvolvido em conformidade com o modelo *Open System Interconnection* (OSI) e destinado para aplicações com requisitos de transmissão assíncrona de dados em alta velocidade. Apesar de estar em conformidade com o modelo OSI, o protocolo CAN só define a camada Física e a camada de Enlace, como representado na Figura 6.

Figura 6 – Arquitetura padronizada do protocolo CAN ISO-11898.



Fonte: Adaptado de Corrigan (2016).

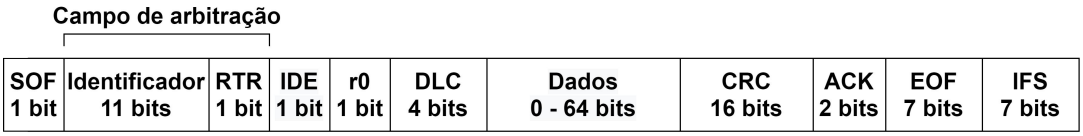
Uma característica importante do protocolo CAN é que este possui duas versões: a padrão, com identificador de mensagem de 11-bit, e a estendida, com identificador de mensagem de 29-bit. Entretanto, este texto foca na versão padrão, visto que é a versão utilizada no dispositivo construído.

Outra característica importante para a compreensão do protocolo é a dominância dos bits no barramento, onde o bit ’0’ é identificado como dominante e o bit ’1’ como recessivo, de modo que quando ambos os estados são aplicados no barramento simultaneamente o mesmo possuirá apenas o estado lógico do bit ’0’. Sendo uma característica crucial para a priorização e detecção de colisões de mensagens.

Uma mensagem do protocolo CAN padrão é subdividida em onze campos como apresentado na Figura 7.

Cada campo da mensagem é detalhado abaixo (CORRIGAN, 2016):

Figura 7 – Mensagem protocolo CAN padrão.



Fonte: Autoria própria.

- *Start of Frame* (SoF): Bit dominante utilizado para indicar o início da mensagem e sincronizar os nós.
- Identificador: Identificador único de mensagem e que estabelece a prioridade da mesma, quanto menor o valor do campo, maior a prioridade.
- *Remote Transmission Request* (RTR): Requisição de dados sobre uma mensagem, quando recessivo sinaliza uma requisição remota.
- *Identifier Extension* (IDE): Indica o uso (valor recessivo) ou não (valor dominante) de extensão no identificador.
- r0: Campo reservado.
- *Data Length Code* (DLC): Quantidade de *bytes* no campo de dados, possui valor entre 0 e 8.
- Dados: Dados da aplicação enviados na mensagem.
- *Cyclic Redundancy Check* (CRC): Soma de verificação dos dados transmitidos para detecção de erros.
- *Acknowledge* (ACK): Confirmação de recebimento de mensagem. É dominante caso ao menos um nó tenha recebido a mensagem livre de erros.
- *End of Frame* (EoF): Marca o final da mensagem.
- *Interframe Space* (IFS): Espaçamento entre mensagens.

Dada a estrutura da mensagem apresentada na Figura 7, existem quatro tipos distintos de mensagens que podem ser transmitidas em um barramento CAN:

- *Data Frame*: Transmissão de dados.
- *Remote Frame*: Requisição de dados.

- *Error Frame*: Indicação de erro na mensagem recebida anteriormente.
- *Overload Frame*: Requisição de maior atraso entre mensagens para processamento das mensagens já recebidas.

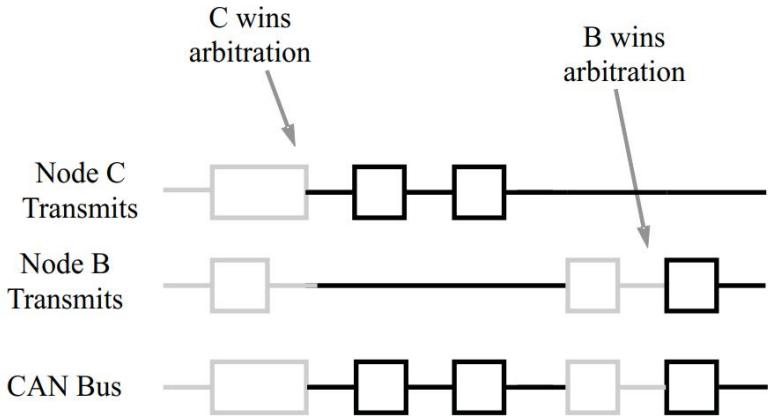
Por se tratar de um protocolo baseado em mensagens, não existe o conceito de endereçamento e as mensagens são enviadas em modo *broadcast* para todos os nós presentes no barramento (PAZUL, 2005).

Dessa forma, é responsabilidade de cada nó participante da rede a verificação das mensagens recebidas (através do campo identificador) para a definição da relevância, e então, o seu descarte ou processamento. Tal característica traz uma grande flexibilidade para o protocolo, visto que novos nós podem ser incluídos no barramento sem que os demais precisem tomar conhecimento ou serem alterados.

Do ponto de vista do controle de acesso ao meio compartilhado, o protocolo CAN é um protocolo *Carrier Sense Multiple Access Collision Detection* (CSMA/CD), o que significa que todos os nós presentes no barramento devem aguardar por um período de inatividade no barramento antes de tentaram enviar uma mensagem, de modo que passado este período, todos possuem oportunidades iguais de transmitir (PAZUL, 2005). Sendo assim, se faz necessário o processo de arbitração responsável pela detecção e resolução de colisões para determinar qual nó manterá o uso do barramento, sendo utilizado o campo de arbitração, como mostrado na Figura 7, e a característica de dominância dos bits para tal tarefa.

Dessa forma, mensagens com campo de arbitração de valor menor são mais prioritárias e ganham o acesso ao barramento e, então, todo o processo de detecção e resolução de colisões é concluído ainda durante a transmissão do campo de arbitração. Além disso, outro fator importante para que o processo de arbitração funcione adequadamente, é que cada nó, durante a transmissão, monitore o barramento para verificar se o estado atual corresponde ao que está sendo enviado. Caso corresponda, o nó deve continuar transmitindo, caso contrário, deve cessar o envio e aguardar até que o barramento se torne inativo novamente. Portanto, devido à natureza única dos identificadores, apenas um nó continuará transmitindo após a arbitração, sem que a mensagem sofra atrasos, ou seja, corrompida. A Figura 8 apresenta um exemplo de como a arbitração opera no barramento CAN.

Figura 8 – Arbitração no barramento CAN.



Fonte: Corrigan (2016).

É importante ressaltar que o campo de arbitração é estruturado de tal forma que a mensagem do tipo *Data Frame* seja mais prioritária do que a mensagem do tipo *Remote Frame* para o mesmo identificador de mensagem.

2.2.1 DBC

DataBase Container (DBC) é um formato de arquivo texto que descreve de forma estruturada a comunicação em uma rede CAN (Vetor Informatik GmbH, 2007), sendo possível definir a temporização, os nós e as mensagens presentes na rede. Dentre essas características, a que torna o DBC extremamente flexível é a descrição de mensagens, pois permite que o campo de dados de uma mensagem, como representado na Figura 7, seja subdividido em sinais, permitindo definir parâmetros como o nome, a máscara de bits, o intervalo, a unidade, a escala e o *offset* do sinal. Tal funcionalidade permite que a rede seja descrita nos mínimos detalhes de forma clara, o que simplifica muito o processo de criação e manutenção. Além disso, é possível utilizá-lo em ferramentas de monitoramento e análise e até mesmo para o desenvolvimento da própria ECU, por meio do *parse* dinâmico ou estático do arquivo. Assim, é possível que o *software* de aplicação da ECU foque apenas no processamento da mensagem e deixe a cargo de uma interface bem definida o processo de conversão.

2.3 INTERFACES DE USUÁRIO E INTERFACES GRÁFICAS

Interfaces de usuário nada mais são do que as formas de interação com pessoas disponibilizadas em sistemas computadorizados. Podem ser divididas em dois grupos: as interfaces de saída, utilizadas para apresentação de informações ao usuário, e as interfaces de entrada, utilizadas para recebimento de informações do usuário. Exemplos conhecidos dessas interfaces são mouses, telas e teclados (GALITZ, 2007).

Interfaces gráficas são atualmente as principais interfaces de usuário utilizadas e estão presentes em todas as situações do nosso cotidiano. São componentes de uma aplicação utilizados para que os usuários possam visualizar e interagir com as funcionalidades disponibilizadas pela aplicação através de um dispositivo apontador, como um dispositivo eletrônico equivalente à mão humana ou até mesmo um que interpreta movimentos da mão humana (GALITZ, 2007). Estas interfaces apresentam uma série de vantagens quando comparadas a interfaces puramente baseadas em texto, como:

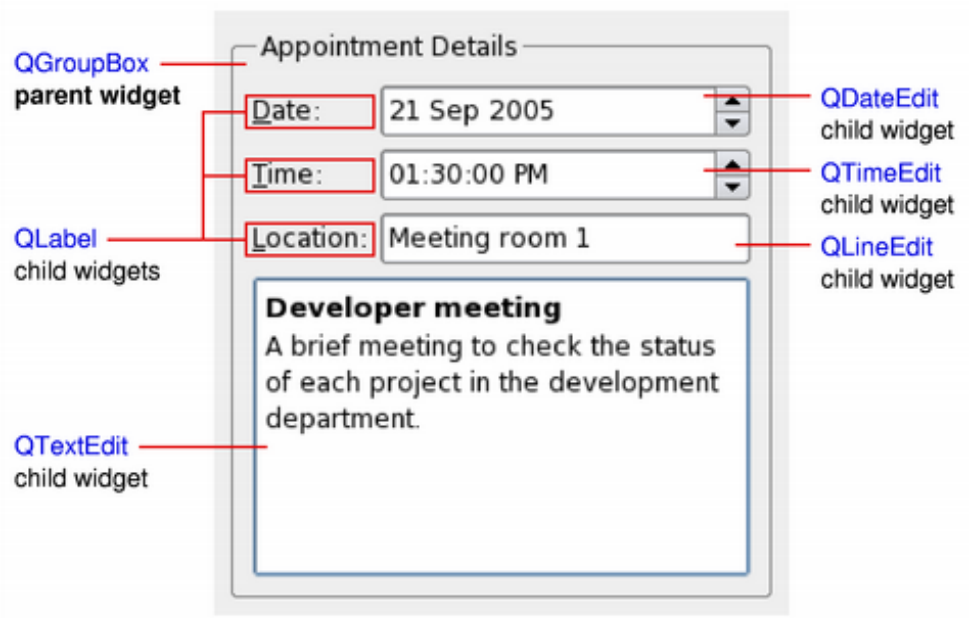
- Mais simples de aprender e compreender
- Mais fáceis de lembrar
- Utilização mais natural ao ser humano
- Fornecem maior contexto durante a utilização

2.3.1 Qt

Qt é um *framework cross-platform* de código aberto desenvolvido em C++ para criação de aplicações gráficas embarcadas, móveis e *desktop*. Lançado em 1995 e hoje mantido pela Qt Company, o *framework* atualmente tem suporte a diversos sistemas operacionais como Windows, derivados Unix, Android, iOS e OS X (ABOUT. . . , 2019). Além do extenso suporte, também possibilita a criação de aplicações que apresentam ótimo desempenho, o que a torna uma escolha excelente para aplicações comerciais como televisores, equipamentos médicos, computadores de bordo de carros, entre outras aplicações gráficas embarcadas (SHERRIFF, 2018).

Aplicações gráficas Qt são orientadas à eventos e construídas através de *widgets*, sendo os elementos primários para a criação de interfaces e utilizados para criar todos os outros elementos presentes na aplicação, como apresentação e entrada de dados (COMPANY, 2021). A Figura 9 apresenta um exemplo de interface criada com o *framework* Qt.

Figura 9 – Exemplo de aplicação gráfica Qt.



Fonte: Company (2021).

Entretanto, o *framework* não se limita em fornecer suporte a criação de interfaces gráficas e integra diversos outros módulos tais como banco de dados, gerenciamento de redes de computadores, processamento de imagens, geração de Formato de Documento Portátil, do inglês *Portable Document Format* (PDF), entre muitos outros. Além dos módulos oficiais, a comunidade também oferece uma infinidade de módulos que enriquecem ainda mais o Qt.

O *framework* disponibiliza, além das bibliotecas em C++, diversas ferramentas que o tornam ainda mais poderoso, tais como o Qt Creator, um Ambiente Integrado de Desenvolvimento, do inglês *Intregrated Development Environment* (IDE), que agrega diversas funcionalidades importantes durante a criação de aplicações gráficas, o Qt Designer e o Qt Quick, *softwares* utilizados para projetar interfaces de usuário, e o Qt Linguist, uma ferramenta para gerenciamento de traduções na aplicação. E todas essas funcionalidades não se limitam ao C++, pois o *framework* Qt já foi portado para diversas outras linguagens como Rust, Java e Python.

2.3.2 TouchGFX

TouchGFX, também denominado X-Cube-TouchGFX, é um pacote X-Cube utilizado para criação de aplicações gráficas para dispositivos baseados em microcontroladores e microprocessadores STM32. O pacote é composto por três componentes principais (STMICROELECTRONICS, 2021):

- TouchGFX *Designer*: ferramenta para criação da parte visual de interfaces gráficas.
- TouchGFX *Generator*: extensão CubeMX para configuração de *hardware* e *software* de dispositivos STM32, criação e gerenciamento de projetos de interfaces gráficas.
- TouchGFX *Engine*: framework C++ utilizado para controle e manipulação da aplicação gráfica.

O pacote possibilita a criação de interfaces gráficas de forma ágil e simples, gerando aplicações de alto desempenho, visto que são totalmente otimizadas para dispositivos da família STM32.

Aplicações gráficas TouchGFX seguem o padrão arquitetural *Model-View-Presenter* (MVP), uma derivação do padrão *Model-View-Controller* (MVC). As três classes deste padrão são detalhas abaixo (STMICROELECTRONICS, 2021):

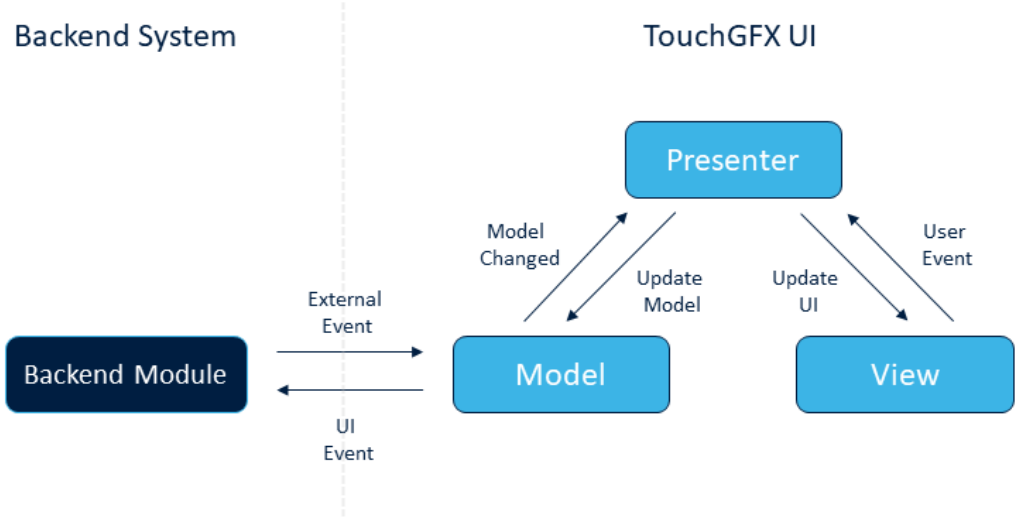
- *Model*: representa os dados presentes na interface gráfica.
- *View*: estrutura visual da interface gráfica.
- *Presenter*: atua sobre as outras duas classes, coletando dados e formatando-os para exibição.

Este padrão possui diversos benefícios, sendo o principal deles a separação de responsabilidades entre as classes, de modo que cada classe realiza uma única tarefa bem definida. Desta forma, a aplicação se torna muito mais simples, facilitando o processo de testes e manutenção.

Contudo, o padrão arquitetural só se aplica para a interface gráfica TouchGFX, sendo necessária a construção de um canal de comunicação entre a parte gráfica e o

restante da aplicação, denominado *backend system* no contexto do TouchGFX, para a transmissão de eventos de ambos os lados. Para tal é fornecida uma função que é executada a cada ciclo do TouchGFX na classe *Model*, nesta deve ser implementado o protocolo de comunicação entre a parte gráfica e o *backend*. Esta característica permite total liberdade na escolha do protocolo, possibilitando que o *backend system* seja executado em outra tarefa ou processador, por exemplo. A Figura 10 apresenta a estrutura de uma aplicação gráfica TouchGFX genérica.

Figura 10 – Estrutura de aplicação gráfica TouchGFX.



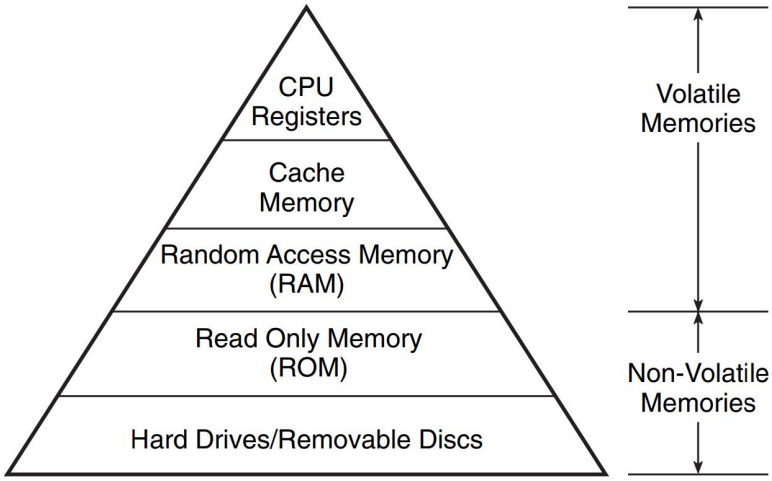
Fonte: STMicroelectronics (2021).

2.4 MEMÓRIA E ARMAZENAMENTO

As memórias de computador são dispositivos eletrônicos utilizados para armazenamento, permanente ou temporário, de dados binários, possibilitando que sejam realizadas operações de leitura e, caso a memória permita, escrita (MAINI, 2007). Uma memória, independente do tipo, pode ser descrita simplificadamente como um conjunto de entidades de armazenamento endereçáveis. Estas entidades são denominadas *words* e são formadas por bits ou, no contexto de memórias, células. É importante notar que a quantidade de *words* em um dispositivo de memória e a quantidade de células em uma *word* variam de memória para memória. A Figura 11 apresenta as categorias de dispositivos de memória tipicamente presentes em um sistema computadorizado.

Memórias podem ser classificadas em dois grupos com relação à persistência

Figura 11 – Hierarquia de memória em sistemas computadorizados.



Fonte: Maini (2007).

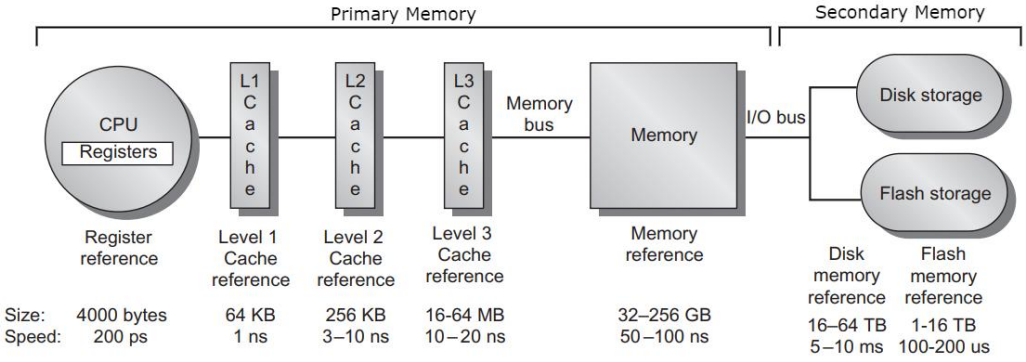
dos dados: voláteis e não voláteis. Memórias voláteis precisam de alimentação para manter os dados e utilizam tecnologias como: *Static Random-access Memory* (SRAM) e *Dynamic Random-access Memory* (DRAM). Já as memórias não voláteis são capazes de armazenar dados sem necessidade de energia elétrica e utilizam tecnologias como: *NAND Flash* e disco rígido magnético (HENNESSY, 2019).

Além disso, também podem ser classificadas com relação à forma de uso: memória primária e memória secundária.

A memória primária é utilizada para execução de programas, mantendo instruções e dados de execução, e para a inicialização do sistema, armazenando o *Basic Input/Output System* (BIOS) e os programas de inicialização do sistema. Como são utilizadas para execução, costumam apresentar tempos de acesso pequenos, o que melhora o desempenho do sistema.

Já a memória secundária, ou armazenamento secundário, é utilizada para armazenamento persistente em massa de dados e programas. Apresentam capacidades de armazenamento muito maiores do que as memórias primárias e possuem baixo custo, em contrapartida, possuem tempos de resposta na ordem de microssegundos, para dispositivos de estado sólido, e milissegundos, para dispositivos magnéticos e óticos. A Figura 12 apresenta as capacidades e tempos de resposta típicos dos diversos tipos de memória presentes em um sistema computadorizado.

Figura 12 – Hierarquia, capacidade e tempos de resposta de memórias.



Fonte: Adaptada de Hennessy (2019).

2.4.1 Secure Digital Card

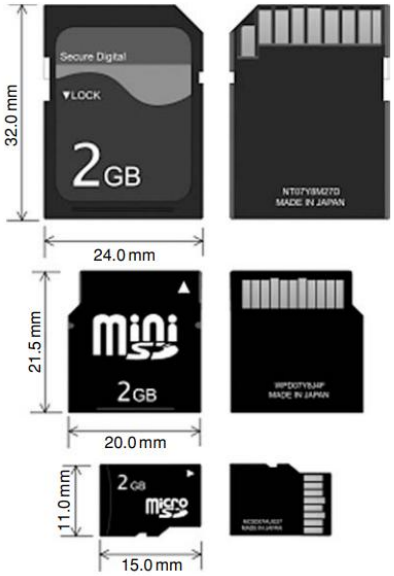
Secure Digital (SD) *cards* são dispositivos de armazenamento não volátil de estado sólido criados pela Panasonic, SanDisk e Toshiba em 2000 com base na tecnologia *Multimedia Card* (MMC) (IBRAHIM, 2010). São desenvolvidos com tecnologia NAND e focam especificamente em características como: grande capacidade de armazenamento, segurança e alto desempenho. Possuem mecanismos de proteção de conteúdo, autenticação e algoritmo de cifra para proteção contra uso ilegal do conteúdo do dispositivo, além de suportar outros sistemas de segurança padronizados, como o ISO-7816, permitindo que o mesmo seja utilizado como dispositivo de identificação eletrônica, por exemplo (ASSOCIATION, 2020).

SD *cards* estão presentes em quatro padrões de capacidade (CAPACITY. . . , 2021):

- *Secure Digital Standard Capacity* (SDSC): capacidade até 2GB utilizando sistema de arquivos FAT12 e FAT16.
- *Secure Digital High Capacity* (SDHC): capacidade entre 2GB e 32GB utilizando sistema de arquivos FAT32.
- *Secure Digital Extended Capacity* (SDXC): capacidade entre 32GB e 2TB utilizando sistema de arquivos exFAT.
- *Secure Digital Ultra Capacity* (SDUC): capacidade entre 2TB e 32TB utilizando sistema de arquivos exFAT.

Além disso, também são disponibilizados em três tamanhos distintos, como apresentado na Figura 13.

Figura 13 – Tipos de SD *Cards*.



Fonte: Ibrahim (2010).

Estas características tornaram os SD *cards* a principal escolha no mercado quando se trata de armazenamento em massa removível, tornando o processo de armazenar e acessar dados muito mais rápido e flexível.

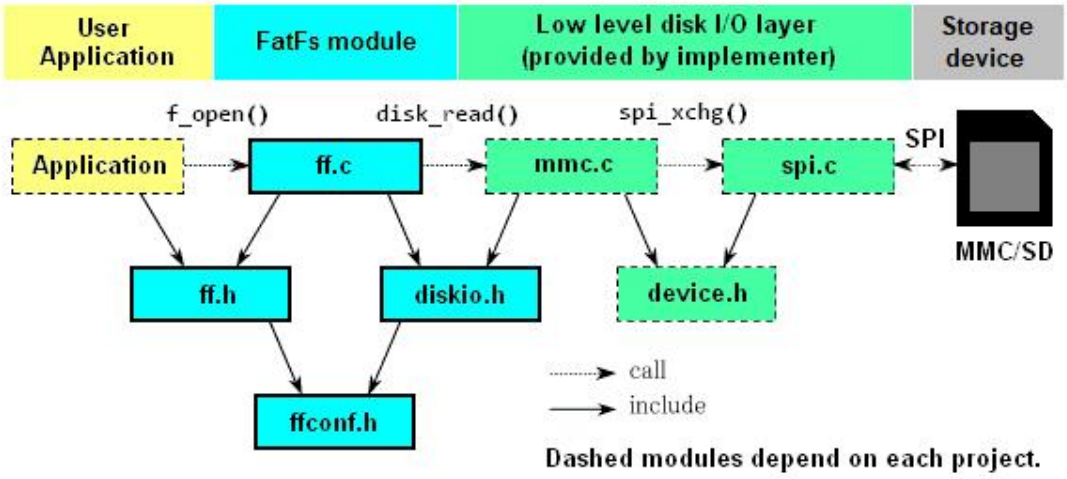
2.4.2 FatFS

FatFS é um sistema de arquivos FAT/exFAT *open source* gratuito voltado para sistemas embarcados, desenvolvido em conformidade com o ANSI C e independente de plataforma, sendo possível integrá-lo em diversas arquiteturas de microcontroladores (FatFs. . . , 2021). Além disso, o FatFS é totalmente isolado do *driver* de disco, como apresentado na Figura 14, tornando possível utilizá-lo com uma diversidade de dispositivos de armazenamento e protocolos de comunicação com o disco, sendo necessário apenas que o *driver* do dispositivo em questão seja fornecido com uma interface conhecida.

Entre as principais características do FatFS, destacam-se:

- Baixa utilização de memória *Random-access Memory* (RAM)
- *Thread safe* em RTOSs

Figura 14 – Arquitetura de aplicação com FatFS.



Fonte: FatFs... (2021).

- Independente de plataforma
- Configurável para diversos discos e protocolos de comunicação

Dessa forma, o sistema FatFS permite que a aplicação embarcada utilize a memória de armazenamento não volátil de forma simples e confiável, sem que seja necessário interagir diretamente com o *hardware* e permitindo que os dados gerados sejam armazenados de forma estruturada.

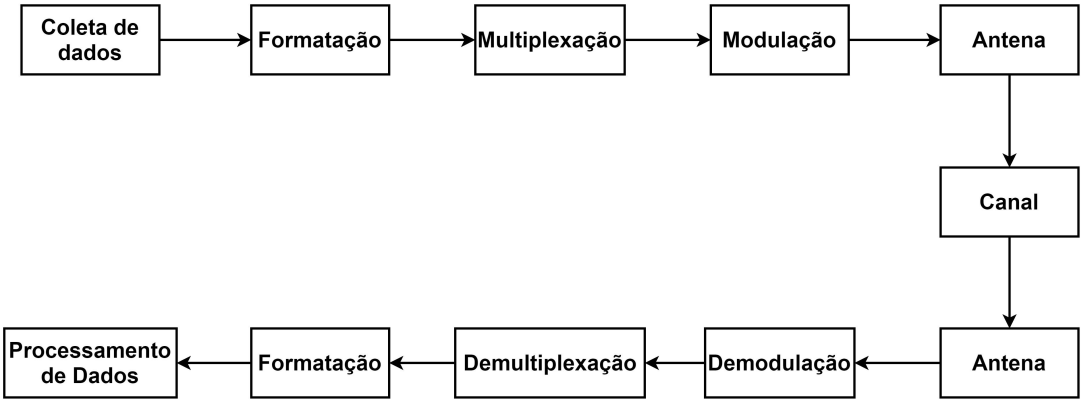
2.5 TELEMETRIA

Sistemas de telemetria são casos especiais de sistemas de comunicação, usualmente digitais, utilizados para coletar os dados de interesse na aplicação no transmissor, que costuma ser de difícil acesso, e transmiti-los para o receptor, onde os dados podem ser tratados e processados de forma simples (CARDEN *et al.*, 2002). A Figura 15 apresenta a arquitetura geral de um sistema de telemetria.

Cada componente presente na Figura 15 é detalhado abaixo:

- Coleta de Dados: coleta os dados de interesse da aplicação.
- Formatação: adequação dos dados coletados para processamento digital.
- Multiplexação/Demultiplexação: permite o compartilhamento do canal de transmissão por dois ou mais pares receptor-transmissor.

Figura 15 – Arquitetura geral de um sistema de telemetria.



Fonte: Autoria própria.

- Modulação/Demodulação: adequação da mensagem ao canal de transmissão, atendendo as características do mesmo.
- Antena: responsável por irradiar/receber o sinal no canal.
- Canal: meio físico no qual ocorre a transmissão dos sinais.
- Processamento de Dados: ponta da comunicação responsável por tratar os dados de interesse.

Sistemas de telemetria são empregados em diversas aplicações, tais como carros de competição e agricultura de precisão. Nos últimos anos, no contexto da Internet das coisas, do inglês *Internet of Things* (IoT), os sistemas de telemetria tem ganhado cada vez mais espaço, permitindo que dados de sensores coletados por pequenos dispositivos, limitados em processamento e capacidade energética, sejam transmitidos a dispositivos de maior capacidade para o seu processamento e publicação. Este modelo de aplicação torna-se cada vez mais comum a medida em que se desenvolvem novas e melhores tecnologias de comunicação, em especial aquelas integradas aos dispositivos de baixo consumo.

As aplicações de telemetria permitem a construção de sistemas mais robustos e estáveis, visto que facilitam o acompanhamento e registro da sua atividade de interesse, possibilitam o cálculo de métricas para avaliação de parâmetros funcionais, identificam falhas e gargalos de forma ágil, tornando objetivos os processos de gerenciamento de qualidade e as tomadas de decisão. Além disso, com o emprego de aplicações de

telemetria há a possibilidade de agir sobre o processo de interesse em tempo real, aplicando medidas corretivas, por exemplo.

Dentre as tecnologias de comunicação sem fio, destacam-se em aplicações de telemetria aquelas que operam nas bandas *Industrial, Scientific and Medical* (ISM) do espectro abaixo de 1 GHz, denominadas Sub-GHz, em especial as faixas de 315 *MHz*, 433 *MHz* e 915 *MHz*. Tais tecnologias apresentam grandes vantagens quando utilizadas em sistemas embarcados, visto que costumam apresentar baixo custo, longos alcances e baixo consumo (KEY. . . , 2017). A Figura 16 apresenta uma comparação entre a pilha TI 15.4, a implementação proprietária da Texas Instruments dos padrões IEEE 802.15.4e/g, e diversas outras tecnologias de comunicação sem fio.

Figura 16 – Comparação de tecnologias sem fio.

Features & specifications	Bluetooth® Classic	Bluetooth Low Energy	Zigbee	Thread	Wi-Fi	Proprietary Sub-1 GHz / 2.4 GHz
Range	Up to 100 m	Up to 200 m or 400 m w LR	Up to 200 m[1]	Up to 200 m	Up to 200 m	Up to 1600 m
Frequency	2.4 GHz	2.4 GHz	2.4 GHz	2.4 GHz	2.4 GHz 5 GHz	Sub-1 GHz 2.4G Hz
PHY throughput	Up to 3 Mbps	Up to 2 Mbps	Up to 250 kbps	Up to 250 kbps	Up to 72 Mbps	500 kbps (Sub-1) 2 Mbps (2.4 GHz)
Network type	Peer-to-peer, Star	Peer-to-peer, Star, Broadcast	Mesh	Mesh	Peer-to-peer, Star	Peer-to-peer, Star, Mesh
Network size	8	30	500+	350+	250	1000+
Battery type	Single-AA	Coin-cell	Coin-cell & energy harvesting	Coin-cell	Double-AA	Coin-cell

Fonte: Wireless. . . (2020).

Em especial, a TI 15.4 oferece uma série de vantagens, tais como:

- Maiores alcances
- Baixa interferência, visto que as bandas são menos congestionadas
- Baixo consumo de energia
- Diversas possibilidades de topologia de rede
- Suporte a grande número de nós na rede

Entretanto, há de se destacar que tais vantagens surgem com o uso de taxas de transmissão muito menores do que aquelas praticadas por dispositivos que operam na faixa de 2,4 *GHz*, sendo ideais para aplicações com baixas taxas de transferência de dados. Ainda, é necessário definir e implementar um protocolo da camada superior

(aplicação) para a comunicação entre os nós, visto que esta camada fica de fora das especificações do protocolo.

Outro detalhe importante com relação à utilização de bandas ISM para o desenvolvimento de aplicações de comunicação sem fio é que estas não são sujeitas a mesma legislação de forma global. Assim, produtos desenvolvidos nessas bandas podem ter problemas ao serem homologados, a depender das normas que se aplicam localmente.

3 REVISÃO DA LITERATURA

3.1 COMUNICAÇÃO AUTOMOTIVA

Nos últimos anos, houve um aumento exponencial do número de funcionalidades de eletrônica embarcada na indústria automotiva. Funções como assistência de navegação, sistemas de segurança ativos (ABS, controle de tração, controle de estabilidade), assistências de pilotagem, entre tantas outras, são cada vez mais comuns até mesmo em modelos de entrada destinados a mercados emergentes. Segundo (LEEN; HEFFERNAN, 2002), na primeira década do século XXI, o custo de eletrônica embarcada em modelos de luxo já superava 23% do seu custo total. O aumento das funcionalidades de eletrônica embarcada que possibilitou com que estas e tantas outras funcionalidades se popularizassem, trouxe também os desafios da integração de sistemas com arquiteturas cada vez mais complexas.

Tipicamente, sistemas embarcados automotivos são divididos em diferentes domínios funcionais (do inglês *functional domains*), cada qual com suas características e limitações. Tais domínios são definidos por (MAURER; WINNER, 2013) como esferas de conhecimento, de influência e de ação nos quais um ou mais sistemas são concebidos. O autor lista os domínios funcionais tradicionalmente aceitos para a sistematização do projeto automobilístico:

- Propulsão (do inglês *powertrain*): envolve as funcionalidades de gerenciamento do motor e distribuição de potência mecânica. Em geral, possui requisitos críticos para prazos de processamento.
- Chassi (do inglês *chassis*): envolve os sistemas que afetam a dinâmica e controle veicular, tais como direção, suspensão e freios. Este domínio é criticamente acoplado à segurança do veículo, visto que seus elementos afetam diretamente o comportamento veicular e as respostas às entradas do piloto.
- Carroceria (do inglês *body*): envolve as funcionalidades do veículo que não estão relacionadas ao controle da sua dinâmica, tais como iluminação e sinalização, limpadores de para-brisa, comandos de janelas, bancos e etc. Em geral, elementos deste domínio não estão sujeitos a limitações restritas de prazos de

processamento ou ainda fortemente acoplados à segurança veicular, visto que não estão diretamente na cadeia de controle do veículo.

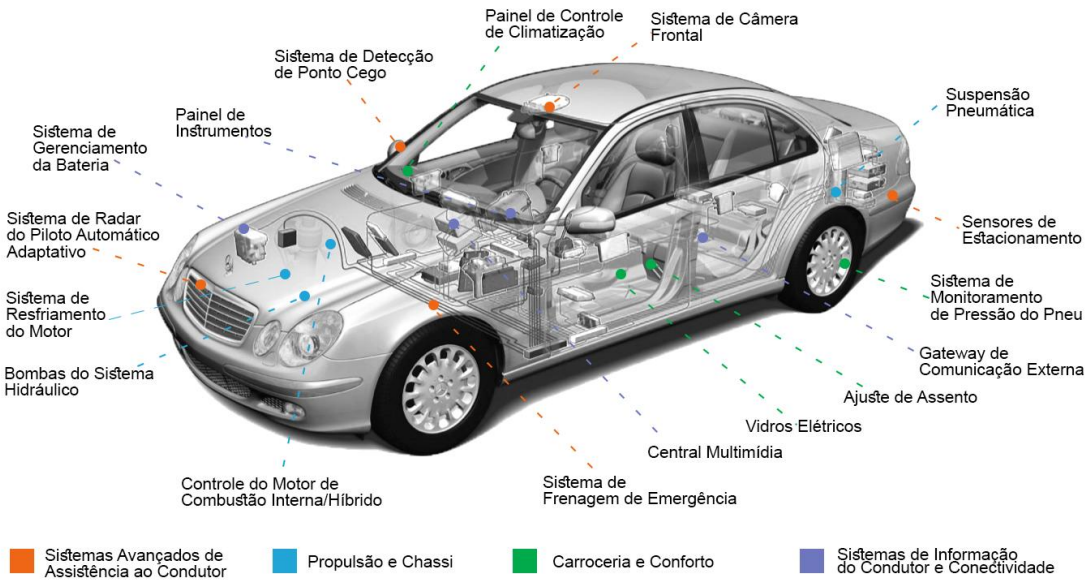
- Interface Homem-Máquina (do inglês *Human-Machine Interface (HMI)*): envolve as funcionalidades de interação do motorista e passageiros com as inúmeras funcionalidades embarcadas no veículo, sobretudo relacionadas ao conforto e entretenimento dos ocupantes.

- Diagnóstico e Telemática (do inglês *diagnostics and telematics*): inclui tanto os sistemas que suportam a troca de informações (dados) entre o veículo e outros entes (infraestrutura, veículos, pedestres, dispositivos, etc.) como os sistemas dedicados a acessar e relatar informações para o diagnóstico dos diversos sistemas embarcados no veículo.

Recentemente, com a introdução de sistemas de auxílio avançado de direção (*Advanced Driver-Assistance* (ADAS)) que requerem múltiplos sensores tais como radares, câmeras e *Light Detection And Ranging* (LIDAR) além dos diversos atuadores que implementam a automatização das tarefas de pilotagem, novos domínios funcionais têm sido acrescentados ao projeto automotivo, sobretudo relacionados às funcionalidades de segurança e proteção veicular (do inglês *safety and security*). A Figura 17 ilustra um conjunto de funcionalidades embarcadas no projeto automotivo e seus correspondentes domínios funcionais.

Um dos principais desafios da indústria automotiva nas últimas décadas tem sido o desenvolvimento de métodos e ferramentas que permitam organizar, diminuir custos e tornar mais eficiente a comunicação entre os elementos de eletrônica embarcada em cada domínio funcional. Do ponto de vista histórico, até o início da década de 1990 a comunicação entre os incipientes sistemas eletrônicos era feita "ponto-a-ponto"(topologia de comunicação em que os nós são conectados entre si sem níveis hierárquicos nem arquitetura definida). Contudo, com o amadurecimento e consequente aumento dos sistemas eletrônicos embarcados, esta solução torna-se inviável devido aos problemas de peso, custos, complexidade e confiabilidade relacionadas ao cabeamento e conexões de múltiplos entes nesta topologia (NAVET; SIMONOT-LION, 2009).

Figura 17 – Funcionalidades automotivas e seus domínios



Fonte: Autoria própria.

O barramento CAN surge, então, do desenvolvimento de uma alternativa que permitisse a multiplexação de dados em um mesmo barramento compartilhado, a descentralização do processamento e a padronização de regras e protocolos de comunicação.

Do ponto de vista taxonômico, as redes de comunicação automotiva são classificadas pelo comitê de redes veiculares e comunicação de dados da SAE conforme as velocidades de transmissão e domínios funcionais de aplicação. A Tabela 1 apresenta um resumo dessa classificação apontando também os principais representantes de cada classe de rede de comunicação automotiva.

Tabela 1 – Classificação de redes embarcadas automotivas.

Classe	Taxa de transferência	Domínios funcionais de aplicação	Principais representantes
Classe A	Abaixo de 10 kbps	Carroceria: Baixo nível	LIN
Classe B	Entre 10 kbps e 125 kbps	Carroceria: Não-diagnóstico Carroceria: Não-críticos	*Single Wire CAN* CAN 2.0
Classe C	Entre 125kbps e 1Mbps	Propulsão: Críticos	*High Speed CAN*
Classe D	Acima de 1Mbps	Propulsão e Chassis: Tempo real Carroceria: Segurança dos ocupantes Interface homem-máquina	FlexRay *Safety-by-wire* MOST

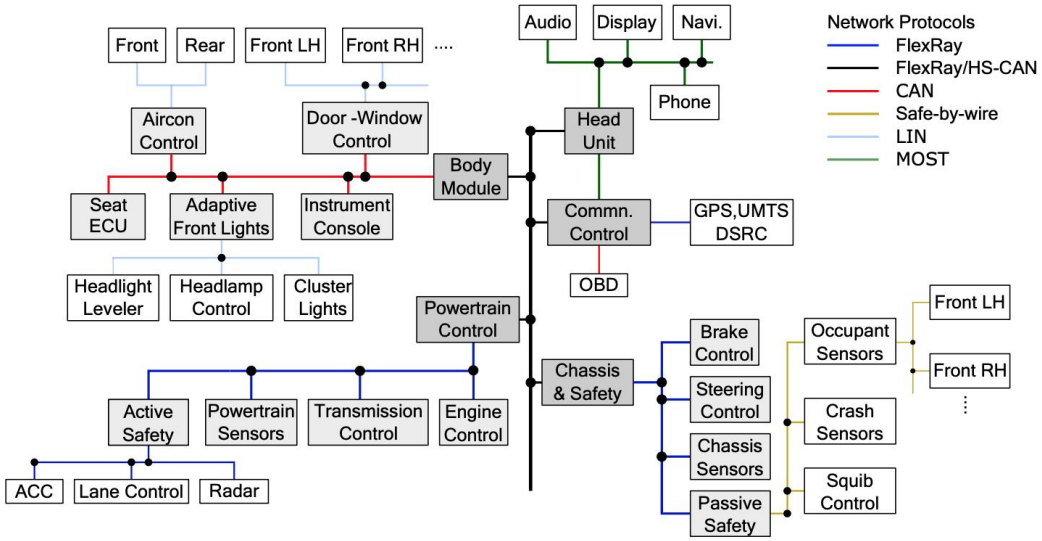
Fonte: Shanker (2016).

Para além da classificação das redes de comunicação automotivas conforme

a sua taxa de transferência, fatores como a topologia suportada (anel, barramento, estrela, ponto-a-ponto), alcance e meio físico de transmissão são importantes para a determinação da rede de comunicação de escolha para uma determinada aplicação.

Segundo (NAVET; SIMONOT-LION, 2009), é comum que o projeto de um automóvel moderno compreenda diversas redes de comunicação, particionadas entre os diferentes domínios funcionais e interconectadas por dispositivos *gateways*. Os *gateways* fornecem interfaces para múltiplas redes como CAN, LIN, MOST, entre outras, a depender dos requisitos funcionais do domínio funcional de cada aplicação. A Figura 18 ilustra a arquitetura das redes de comunicação no projeto de um automóvel moderno.

Figura 18 – Arquiteturas de comunicação típicas em um automóvel moderno



Fonte: Shanker (2016).

Do ponto de vista da evolução no desenvolvimento dos protocolos e redes de comunicação automotivos, segundo (NAVET; SIMONOT-LION, 2009), o desenvolvimento de novas funcionalidades principalmente relacionadas a proteção dos ocupantes (controle de acionamento de *air bags*, detecção de colisão, etc.), tais como *"Safe-by-Wire plus"*, devem estimular o desenvolvimento de novos protocolos de comunicação em um futuro próximo.

3.2 ELETRÔNICA EMBARCADA EM COMPETIÇÕES SAE

Da mesma forma como na indústria automobilística, a inclusão de elementos de eletrônica embarcada também se consolidou no contexto das competições promovidas pela SAE, sobretudo nos protótipos de Fórmula SAE. Neste contexto, as funcionalidades de eletrônica embarcada dedicam-se especialmente ao monitoramento de parâmetros de desempenho do veículo. Para tanto, um número crescente de sensores tem sido embarcado nos protótipos de Fórmula SAE, o que estimulou também o desenvolvimento de arquiteturas de comunicação para o seu suporte. O protocolo CAN destaca-se como a solução mais difundida para o desenvolvimento de tais arquiteturas de comunicação.

Em (LOPES, 2007) são listadas algumas das características do protocolo CAN que o posicionam como a solução de escolha para aplicações com restrições de tempo crítico e segurança, tais como o monitoramento de um veículo de Fórmula SAE:

- Taxa de transmissão configurável (entre 50kbps a 1Mbps);
- Mecanismo de controle de acesso CSMA/CD com detecção de arbitragem não-destrutiva de dados;
- Meio físico composto de apenas dois fios;
- Flexibilidade para adicionar e remover dispositivos da rede;
- Protocolo multi-mestre, em que todos os dispositivos se alternam nos estados mestre e escravo durante a troca de mensagens.

No quadro nacional, entre os principais trabalhos relacionados ao uso do barramento CAN como base para a criação de uma arquitetura de rede de comunicação automotiva aplicada no contexto das competições promovidas pela SAE, destacam-se as publicações de (NUNES, 2016), (NASCIMENTO JUNIOR; SOUZA, 2014), (LEITE, 2012) e (ROSSO, 2017).

Em (NASCIMENTO JUNIOR; SOUZA, 2014) é documentado o desenvolvimento pioneiro de uma arquitetura eletrônica embarcada em veículos Baja SAE no Brasil, aplicada ao protótipo da equipe Víbora do Instituto Militar de Engenharia. A arquitetura desenvolvida contempla 3 ECUs de sensoriamento (temperatura, nível de combustível e velocidade da roda) e uma interface de apresentação dos dados, através

de um *display TFT 320x240*. A plataforma de desenvolvimento escolhida é baseada na arquitetura PIC, sendo utilizada a família PIC18F para as ECUs de sensoriamento e um microcontrolador dsPIC33 para a interface gráfica.

Em (NUNES, 2016), um sistema composto por três ECUs é proposto para a aquisição e exibição das informações de velocidade, rotação do motor, temperatura, tensão da bateria e nível de combustível de um veículo Baja SAE, desenvolvido pela equipe Car-Kará da Universidade Federal do Rio Grande do Norte. Esse trabalho concentra-se principalmente no desenvolvimento da instrumentação e da arquitetura do barramento CAN, limitado à coleta e exibição de dados locais. Ainda, do ponto de vista das ferramentas de *hardware* para a construção da solução proposta, os autores lançam mão da plataforma de desenvolvimento Arduino (ATmega 2560) para a rápida prototipação, sacrificando a eficiência e a viabilidade financeira da expansão em escala da solução proposta.

Em (LEITE, 2012), um módulo eletrônico composto por sensores de aceleração, giroscópios e receptor GPS foi desenvolvido e implantado no barramento de comunicação CAN de um veículo de Fórmula SAE da equipe *Solid Edge EESC-USP* da Universidade de São Paulo para a aquisição das informações de posicionamento e deslocamento do veículo. Além disso, através de um dispositivo *sniffer* (coletor de informações) uma aplicação baseada na plataforma *LabVIEW* foi criada para a visualização em um dispositivo *desktop* dos dados coletados, através de uma interface serial RS232. Do ponto de vista do sistema final, a escolha da plataforma *LabVIEW* implica na necessidade de gerenciamento de licenças para a utilização da aplicação de visualização dos dados coletados, ao mesmo tempo em que a utilização de uma interface cabeada como meio de comunicação entre o veículo e a aplicação de visualização de dados dificulta o controle das informações coletadas em tempo real durante as atividades da competição. Para o desenvolvimento da aplicação embarcada, os autores lançam mão da família de microcontroladores PIC18F, que apesar de possuírem controlador CAN integrado e Encapsulamento Duplo em Linha , do inglês *Dual In-line Package* (DIP), que favorece a sua montagem em placas de prototipação, possuem ultrapassada arquitetura 8-bits que impõe limitações na precisão de saída dos dados calculados na unidade de processamento.

Em (ROSSO, 2017), um controlador de aceleração eletrônico foi desenvolvido

seguindo os requisitos levantados pela equipe Fórmula CEM de Fórmula SAE da Universidade Federal de Santa Catarina. Para tanto, um sensor acoplado ao pedal de aceleração foi conectado à plataforma de desenvolvimento Texas Instruments Hercules RM4 (ARM Cortex-R4) para controlar o acionamento do TBI (do inglês *Throttle Body Injector*), através de uma rede de comunicação CAN embarcada no veículo. Este trabalho mostra o uso de ferramentas atuais de desenvolvimento para a solução proposta, além de extensiva discussão a respeito dos requisitos de documentação, de *hardware*, de *software* e de desempenho para o desenvolvimento de funcionalidades de eletrônica embarcada *safety-critical* para veículos de Fórmula SAE.

No contexto internacional, a utilização de arquiteturas de eletrônica embarcada encontra-se mais consolidada no ambiente das competições estudantis organizadas pela SAE. Assim, a literatura mais recente reúne trabalhos que partem de uma infra-estrutura de eletrônica embarcada avançada e cujo foco, portanto, trata da adição de componentes e funcionalidades igualmente avançadas, tais como o desenvolvimento de sistemas de telemetria (ELLIOT, 2012), integração de sensores não-triviais (VISCONTI *et al.*, 2019), ou ainda extensivas discussões a cerca de tópicos avançados de engenharia de sistemas e de requisitos (TAN, 2011).

Dentre estes trabalhos, destacam-se as publicações de (BRAUNE, 2014) e (EICH, 2016), dois projetos para o desenvolvimento de soluções de telemetria para veículos de Fórmula SAE.

Em (BRAUNE, 2014), é exposto o desenvolvimento de um sistema de telemetria para a equipe *AMZ* de Fórmula SAE da Universidade Tecnológica Federal de Zurique (ETH Zürich). A solução proposta incorpora, além da transmissão de parâmetros funcionais do veículo, também um canal de vídeo para a transmissão de imagens capturadas por uma câmera GoPro embarcada. O autor apresenta um breve estudo sobre os temas *Link Budget* e *Path Loss* apresentando o compromisso entre diferentes frequências de portadora para a transmissão de dados sem fio. Para o desenvolvimento da solução proposta, são utilizadas ferramentas criadas especificamente para a criação de sistemas embarcados automotivos, tais como a Unidade de Controle Veicular, do inglês *Vehicle Control Unit* (VCU) ES910 e ES921 (ETAS). Do ponto de vista prático, uma série de testes foram projetados e executados para garantir a conformidade do sistema com os requisitos levantados, conforme as práticas modernas para o

desenvolvimento de ECUs. Entretanto, a utilização de sistemas comerciais de aquisição de dados automotivos, torna a solução final dependente de onerosas licenças para o seu uso.

Em (EICH, 2016), um extenso e detalhado processo de levantamento de requisitos e de engenharia de sistemas é conduzido para o projeto de um sistema de telemetria bi-direcional aplicado ao protótipo de Fórmula SAE da equipe *municHMotorsport* da Universidade de Ciências Aplicadas de Munique. Este trabalho apresenta a solução proposta do ponto de vista da engenharia de sistemas, focando especialmente no processo de levantamento de requisitos, baseando-se no guia de práticas descrito em (LIBRARY; DEFENSE, 2016), e nos testes de validação da solução apresentada, em detrimento da exposição detalhada do desenvolvimento do projeto de *hardware* e *software*. Sendo assim, justifica-se que as ferramentas de desenvolvimento (sistema embarcado, antenas, etc.) escolhidas para solução proposta sejam, na sua maioria, componentes *off-the-shelf*, isto é, módulos de propósito geral que se adéquam para o desenvolvimento da aplicação desejada, dado o enfoque do autor no projeto e análise da aplicação enquanto sistema.

4 MATERIAL E MÉTODOS

Este capítulo trata da identificação e análise do problema relatado pela equipe Fórmula UTFPR e da apresentação da solução proposta, em particular os materiais escolhidos e o planejamento para sua elaboração.

4.1 ANÁLISE DO PROBLEMA

O protótipo mais recente da equipe Fórmula UTFPR (EK-304) possui um projeto de eletrônica embarcada primitivo, dispondo apenas de um conjunto de elementos essenciais para a condução do carro. Apesar disso, a equipe possui um *roadmap* para o desenvolvimento de uma arquitetura de comunicação automotiva completa que permita a inclusão de um número maior de sensores nos seus protótipos. Este planejamento de longo prazo foi definido pela equipe tendo em vista o projeto de protótipos cada vez mais competitivos, utilizando os dados coletados por uma rede de sensores eletrônicos para identificar tanto falhas de projeto, bem como oportunidades para o aumento de performance.

A primeira entrega do planejamento da arquitetura de comunicação automotiva é o projeto de um barramento de comunicação baseado no protocolo CAN 2.0, que será embarcada no protótipo atualmente em desenvolvimento (EK-305). O projeto do barramento de comunicação prevê o suporte para as unidades apresentadas na Tabela 2.

Tabela 2 – ECUs presentes no barramento projetado.

ECU	Descrição
Transmissão	Responsável pelo envio de informação de posição de marcha.
Suspensão Dianteira	Responsável pelo envio das informações de aceleração e posicionamento angular (giroscópio) nos três eixos em cada braço da suspensão dianteira.
Injeção Eletrônica	Responsável pelo envio das informações de temperatura do ar e do líquido refrigerante, posição do corpo do acelerador (TPS), pressão de escapamento (MAP), rotação do motor e eficiência na queima de combustível (Lambda).
Suspensão Traseira	Responsável pelo envio das informações de aceleração e posicionamento angular (giroscópio) nos três eixos em cada braço da suspensão traseira, além da velocidade das rodas traseiras.

Fonte: Autoria própria.

A escolha da equipe pela implementação do barramento CAN como meio físico para o desenvolvimento de uma arquitetura de comunicação embarcada permite ainda

que a equipe tenha flexibilidade para a adição de novos componentes no futuro, tais como de temperatura de freios, temperatura de pneus, nível de combustível, entre outros.

Entretanto, apenas a introdução de elementos geradores de dados no veículo não resolve o problema de escassez de informações *per si*. É necessário que se disponha de ferramentas que permitam o processamento e a análise dos dados gerados para que, então, sejam transformados em *informação relevante*. Neste cenário, surge a necessidade de um sistema capaz de disponibilizar os dados gerados no veículo para que seja analisado, em tempo real e *a posteriori*, o desempenho do protótipo.

4.2 PROPOSTA DE SOLUÇÃO

A primeira etapa para o projeto da solução proposta parte do estudo do contexto do problema, isto é, a identificação de detalhes e fatores limitantes que possam impactar o projeto diretamente, permitindo, desta forma, que a solução projetada tenha mais chances de atender as necessidades da equipe.

A característica mais marcante e perceptível do sistema idealizado é sua composição de duas partes: a primeira embarcada no veículo, para aquisição dos dados de interesse da equipe, e a segunda remota ao veículo, responsável por receber os dados e apresentá-los adequadamente. Consequentemente, identificou-se que as duas partes citadas são mapeadas para dois agentes com necessidades distintas: o piloto, embarcado no veículo, com interesse nos dados que possam ajudá-lo a tirar o melhor desempenho possível do veículo durante a competição; e a equipe técnica, instalada remotamente, que visa coletar dados que possam ajudá-la em identificar possíveis melhorias no protótipo além de acompanhar o comportamento do mesmo durante a competição para a tomada de decisões estratégicas.

Como comentado anteriormente, a equipe já havia realizado um esforço inicial para a implantação do barramento CAN para a comunicação embarcada de seus protótipos. Entretanto, após análise da modelagem do barramento implementado, verificou-se que o barramento não havia sido modelado de forma ideal e documentada. Para tal análise, todos os parâmetros e mensagens trafegados no barramento foram elencados, assim como o período nos quais os mesmos são gerados e seu respectivo tamanho, o resultado desta análise está presente no Apêndice B. Após a conclusão do

estudo do barramento, os pontos apresentados abaixo foram identificados:

- Cada ECU que compõe o sistema foi mapeada para um ID CAN e um identificador presente no *payload* foi criado para indicar qual informação específica da ECU estava sendo transmitida, uma abordagem não convencional, visto que usualmente os identificadores do barramento CAN são orientados à mensagem, indicando unicamente a informação transmitida.
- As mensagens não foram priorizadas corretamente, o que dificulta a inclusão adequada de novas mensagens.
- A estrutura das mensagens não foi projetada de forma convencional e consistente, onde um mesmo ID não indicava unicamente uma informação, o que dificulta o processamento das mesmas.

Dessa forma, foi necessário readequar o barramento CAN visando mitigar os problemas identificados acima. Para descrever formalmente a nova estrutura da rede, foi o utilizado o formato DBC, já apresentado na subseção 2.2.1, que além de permitir que o barramento seja descrito de forma estruturada, permite que o mesmo seja modelado de forma robusta e coesa, simplificando a manutenção e inclusão de nós e mensagens na rede.

Tendo conhecimento dos agentes interessados e de todas as informações presentes no barramento CAN, se faz necessário determinar quais informações são exigidas por cada agente. Como já discutido, a equipe técnica deseja acompanhar o desempenho do protótipo em tempo real e realizar análises *a posteriori*, desta forma seria benéfico que todas as informações trafegadas no barramento CAN estivessem disponíveis. Já o piloto, tem interesse apenas no subconjunto de informações ideal que possam auxiliá-lo na condução do protótipo, neste contexto determinamos, junto a equipe Fórmula UTFPR, o conjunto apresentado na Tabela 3, derivado da análise apresentada no Apêndice B.

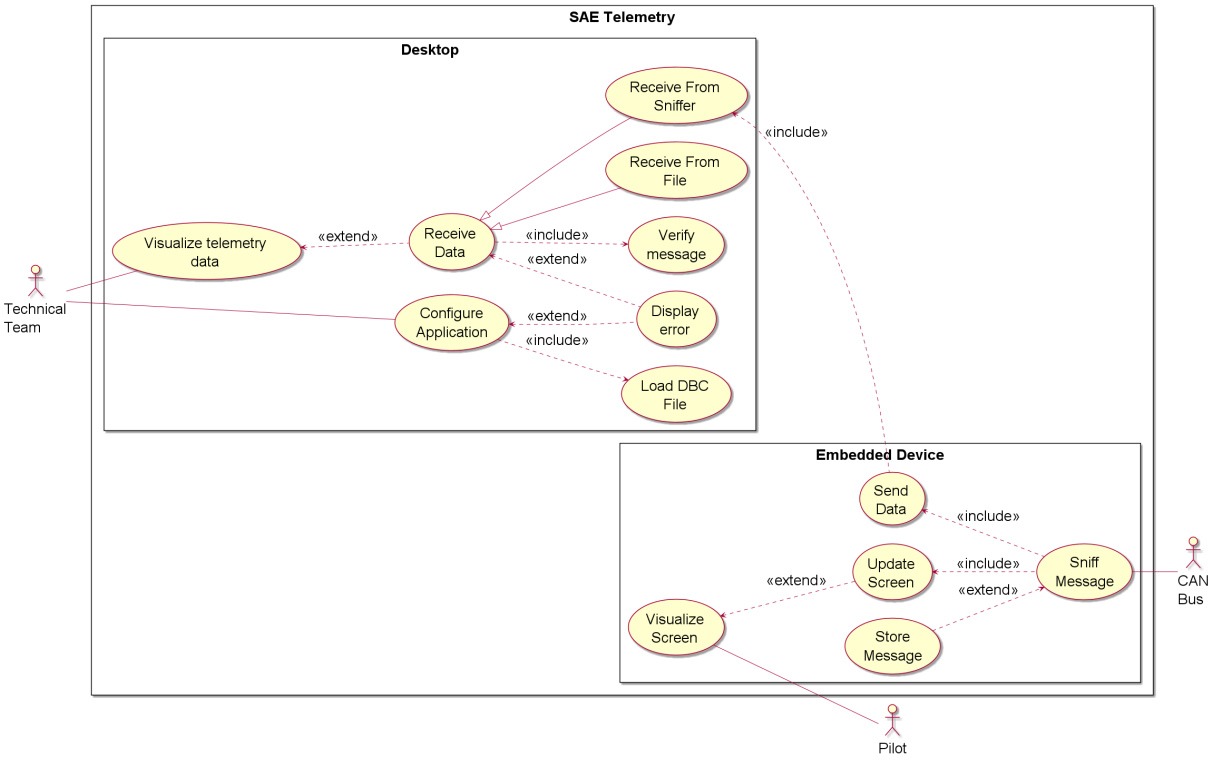
Considerando as informações apresentadas acima, o diagrama de caso de uso apresentado na Figura 19 foi desenhado para descrever as funcionalidades do sistema.

Tabela 3 – Informações de interesse do piloto.

Módulo CAN	Parâmetro	Prioridade
Injeção	Temperatura do líquido de arrefecimento	Alta
Injeção	Rotação do motor	Alta
Injeção	Temperatura do ar no coletor de admissão	Média
Traseiro	Velocidade	Alta
Traseiro	Nível de combustível	Baixa
Traseiro	Temperatura do óleo	Média
Traseiro	Marcha engatada	Alta

Fonte: Autoria própria.

Figura 19 – Diagrama de caso de uso do sistema.



Fonte: Autoria própria.

4.3 MODELAGEM DO SISTEMA

Com base no que foi apresentado na seção anterior, os requisitos apresentados no Apêndice A foram desenvolvidos e, para atendê-los, propõem-se um sistema composto por três grandes subsistemas, apresentado na Tabela 4 e ilustrado na Figura 20.

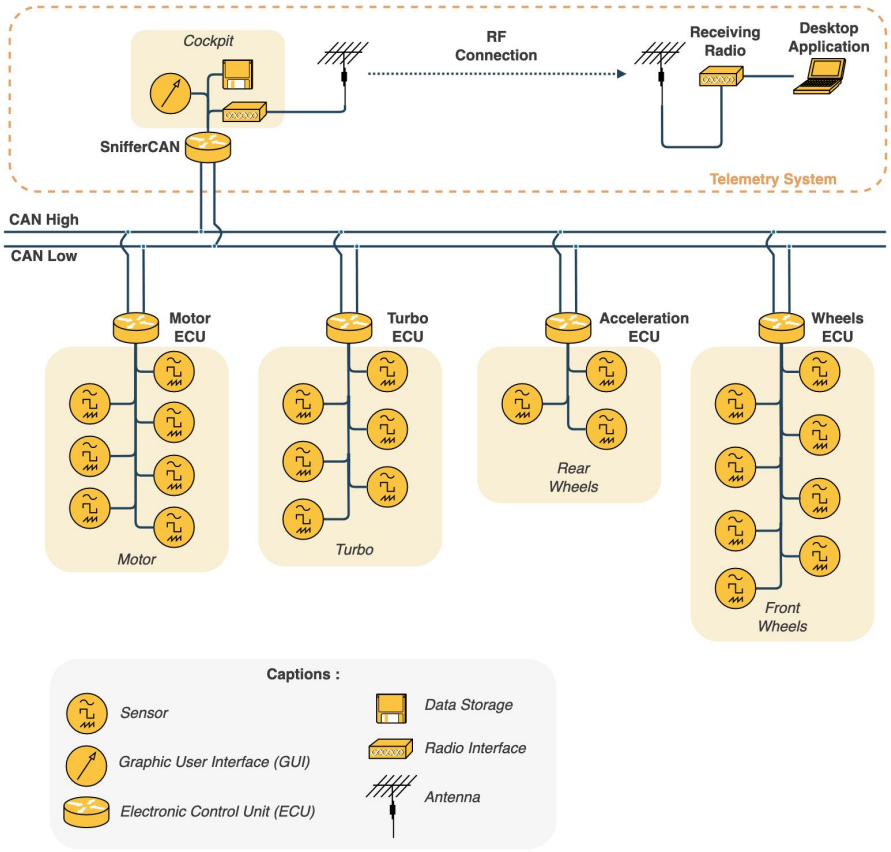
Nas seções a seguir, todos os subsistemas presentes serão discutidos, apresentando os detalhes de projeto e funcionamento de cada um deles.

Tabela 4 – Resumo da delegação de requisitos de sistema.

Componente	Descrição	Requisitos implementados
SnifferCAN	Unidade embarcada responsável pela captura, armazenamento local em uma unidade de memória persistente e transmissão dos dados.	REQ1 REQ2 REQ3 REQ4 REQ6 REQ11 REQ12
Rádio Receptor	Interface de comunicação sem fio responsável pelo recebimento dos dados enviados pelo SnifferCAN de forma remota.	REQ3 REQ12
Aplicação Desktop	Processamento e apresentação dos dados recebidos em tempo real ou via arquivo de registro de mensagens capturados.	REQ5 REQ7 REQ8 REQ9 REQ10 REQ12

Fonte: Autoria própria.

Figura 20 – Arquitetura de alto nível do sistema.



Fonte: Autoria própria.

4.4 SNIFFERCAN

O SnifferCAN é o subsistema embarcado no veículo responsável pela captura, visualização e armazenamento local das mensagens capturadas no barramento CAN, além da transmissão sem fio para uma unidade remota.

Nas próximas subseções serão apresentadas os detalhes das implementações de *hardware* e *software* que tornam possível a construção deste subsistema.

4.4.1 Hardware

Os requisitos endereçados ao SnifferCAN, como apresentados na Tabela 4, implicam em uma série de funcionalidades e itens de *hardware* necessários para atendê-los, a Tabela 5 apresenta o conjunto identificado para a implementação do escopo definido pelos requisitos delegados ao subsistema.

Tabela 5 – Resumo dos requisitos de *hardware* - SnifferCAN.

Funcionalidade do subsistema	Item de Hardware
Coleta de dados do barramento	Controlador CAN *Transceiver* CAN
Registro em mídia não volátil	Controlador SDHC Conector de cartão SD
Exibição de dados embarcada	Controlador LCD *Display* LCD
Transmissão de dados para base fixa	Modulador RF Amplificador RF Antena
Detecção de erros na transmissão remota	Unidade de cálculo CRC

Fonte: Autoria própria.

Com o objetivo de simplificar o desenvolvimento do sistema final, optou-se pela utilização de placas de desenvolvimento (*development boards*), que tornam dispensáveis as etapas de projeto e fabricação de placa de circuito impresso para acomodação dos elementos de *hardware*, além de agregarem uma série de recursos e facilidades para as etapas de prototipação do sistema. Esta escolha de projeto está alinhada com a proposta de desenvolvimento de uma solução compatível com um TRL de grau 6, para o qual se espera que um modelo representativo em termos de formato, configuração e função seja desenvolvido para demonstrar em ambiente relevante (integrado) as funções críticas e o desempenho de um sistema projetado. Neste sentido, o desenvolvimento de uma plataforma de *hardware* sob medida para o sistema idealizado excede o

escopo deste trabalho.

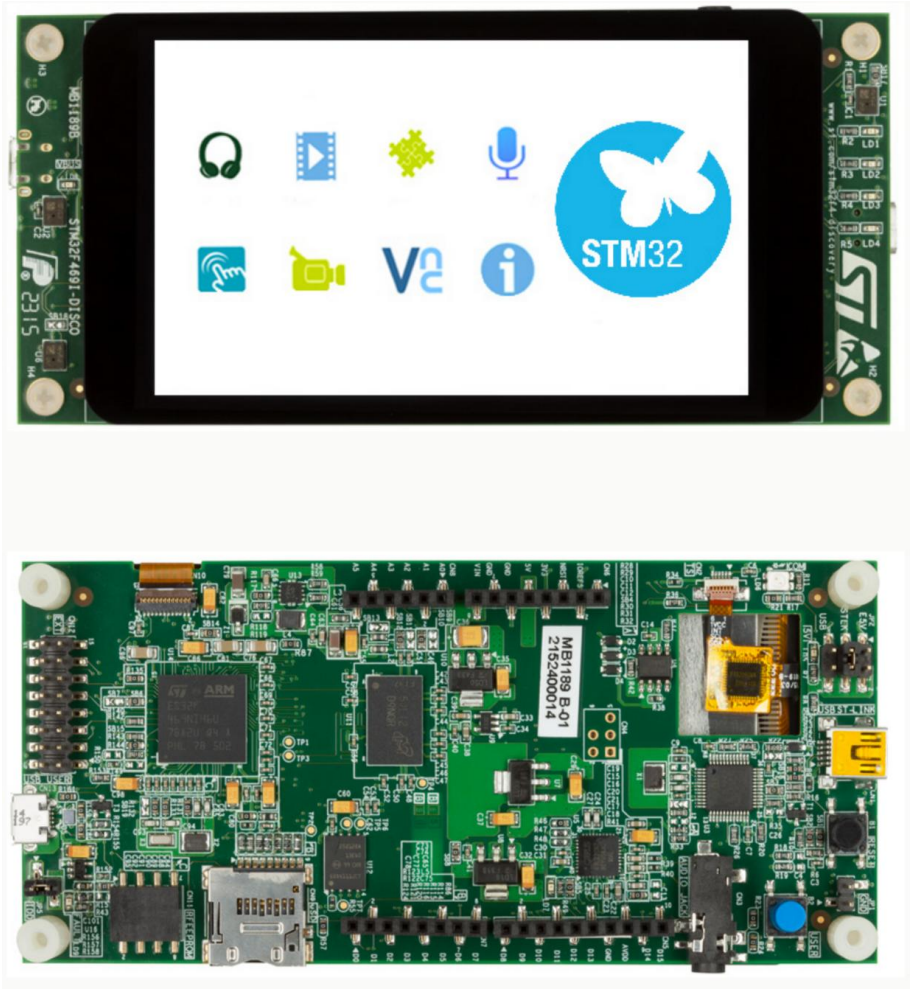
Entretanto, encontrar uma solução única de *hardware* pronta e disponível em mercado *off-the-shelf* que compreenda todos os elementos listados na Tabela 5 é uma tarefa complicada, dada a diversidade de elementos solicitados. Em geral, plataformas de desenvolvimento que integram os componentes que permitem a *transmissão de dados para a base fixa* não integram também elementos que permitam a *exibição de dados embarcada*, ou ainda deixam de incluir elementos que permitam a *coleta de dados do barramento*. Da mesma forma, outras plataformas que integram os elementos necessários para a implementação de um ou mais requisitos, por vezes omitem outros elementos de *hardware* necessários para a implementação dos demais requisitos. A solução encontrada para este impasse foi a adoção de uma arquitetura de *hardware* distribuída, composta por um conjunto de elementos de *hardware* prontos e disponíveis em mercado, ao invés de uma única plataforma de desenvolvimento.

A família de microcontroladores ARM Cortex-M (e**M**bedded) oferece uma variedade de núcleos de 32 bits interoperáveis (isto é, que permitem a portabilidade de *software* entre si) de baixo custo e com boa eficiência energética, adequados para o desenvolvimento de uma gama de aplicações embarcadas tais como dispositivos IoT, controle de motores, dispositivos Interface Gráfica do Usuário, do inglês *Graphical User Interface* (GUI), sistemas de controle industrial, sistemas automotivos, etc. Segundo pesquisa internacional (EE Times, 2019), processadores de 32 bits estão presentes em mais de 60% dos projetos de aplicações embarcadas atuais. Ainda, segundo a mesma pesquisa, a família de processadores ARM Cortex-M é apontada como favorita para o desenvolvimento de novos projetos, em detrimento a outras arquiteturas como PIC32, AVR32. Por ser uma escolha de mercado, optamos pela procura de placas de desenvolvimento com núcleos ARM Cortex-M.

A plataforma STM32F469DISCOVERY é uma plataforma de desenvolvimento criada para a prototipagem de aplicações GUI, baseada no microcontrolador ARM Cortex-M4 STM32F469, que dispõe de um *display* TFT-LCD de 4 polegadas com resolução de 800x400 *pixels*, além de interfaces de áudio, *debugger* integrado, etc. A plataforma é apresentada na figura Figura 21.

Do ponto de vista dos elementos de *hardware* necessários para a implementação dos requisitos do sistema, quer seja pelos recursos internos ao núcleo ou por

Figura 21 – Plataforma de desenvolvimento STM32F469DISCOVERY.



Fonte: Adaptado de STMicroelectronics (2020).

periféricos incluídos no projeto da placa de desenvolvimento, esta plataforma oferece uma grande cobertura dos itens necessários para o desenvolvimento do subsistema SnifferCAN, entre eles:

- *Controlador bxCAN*: O controlador CAN 2.0A/B integrado ao núcleo diminui a carga de processamento do núcleo, dado que gerencia de forma autônoma o envio e a recepção de mensagens, inclusive a filtragem de mensagens relevantes de acordo com uma lista de IDs pré-configurada.
- *Controlador LTDC e Display Serial Interface (DSI) Host*: O controlador LTDC integrado ao núcleo diminui a carga de processamento do núcleo ao alimentar diretamente o *display* gráfico através de operações DMA para a leitura do *frame-buffer*. Além disso, a presença de uma interface DSI permite a comunicação serial

em alta velocidade com *displays Liquid Crystal Display* (LCD) de alta resolução utilizando um número reduzido de pinos.

- *Controlador SD*: Controlador SD 2.0 integrado ao núcleo para a comunicação em alta velocidade com dispositivos de armazenamento removíveis. O controlador diminui a carga de processamento do núcleo, dado que oferece a transmissão de dados através de métodos do tipo *Direct Memory Access* (DMA).
- *Interfaces UART/USART*: Controlador para implementação de interfaces de comunicações seriais síncronas e assíncronas *full duplex* integrado ao núcleo, que permite flexibilidade de comunicação com periféricos e recursos externos, tal como o componente Rádio Transmissor neste projeto. Ainda, interfaces UART se constituem como um meio prático para a criação de interfaces de *debug* e comunicação com dispositivos externos em baixas e médias velocidades.
- *Relógio Real Time Clock (RTC)*: A unidade de temporização de tempo real integrada ao núcleo provê uma forma precisa de rotulação de tempo para as mensagens e eventos registrados no barramento. Este elemento é especialmente importante para a criação da funcionalidade de *replay* dos dados salvos na unidade de memória embarcada no veículo.

A plataforma STM32F469DISCOVERY, no entanto, não oferece nenhum recurso de *hardware* para a implementação de funcionalidades de comunicação sem fio, de modo a atender completamente os requisitos listados na Tabela 5. Sistema em um *chip*, do inglês *System on Chip*s (SoCs) desenvolvidos para aplicações sem fio oferecem (em um único *chip*) uma solução de *hardware* completa, integrando tanto microcontroladores e periféricos tradicionais (*timers*, conversores AD, etc.) como também componentes dedicados à transmissão sem fio (PLLs, moduladores e demoduladores e etc.). Por esta razão, neste projeto, optamos pelo uso de um SoC para a implementação das funcionalidades de comunicação sem fio delegadas ao subsistema SnifferCAN.

A escolha de um SoC para a implementação da funcionalidade de transmissão sem fio está diretamente associada a própria tecnologia e protocolo de transmissão de dados, uma vez que, não incomum, cada SoC dedica-se a implementação de uma tecnologia de transmissão de dados. Alguns exemplos disponíveis no mercado são: STM32WLE5 (LoRA), CC13xx (TI-SubGHz), ATA8529 (Sigfox), ESP8266 (Wi-Fi), etc.

Diante da grande variedade de tecnologias de transmissão distintas disponíveis em soluções prontas no mercado, levou-se em conta os seguintes aspectos característicos da aplicação de interesse para a sua definição:

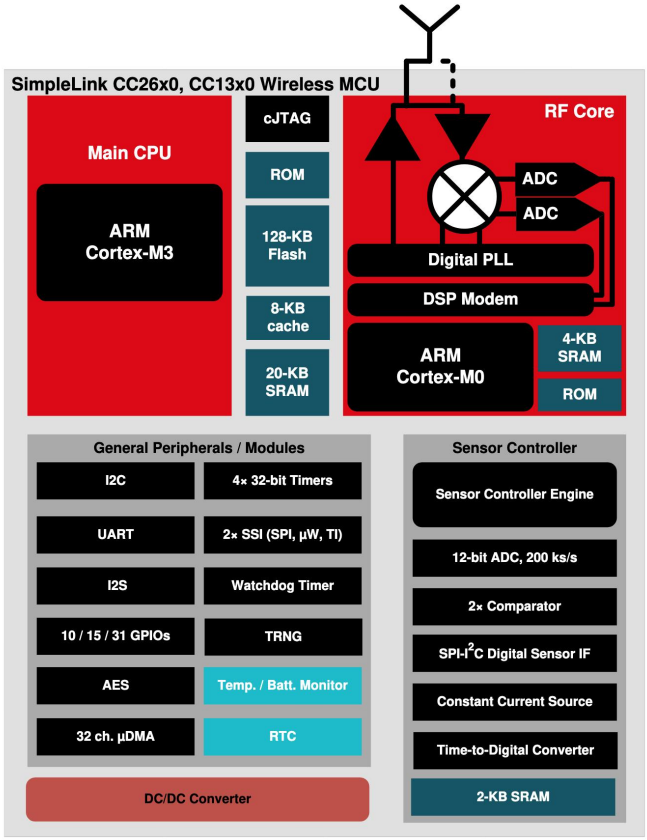
- *Diminuto volume de dados*: A comunicação sem fio deve suportar tão somente o envio de dados do subsistema SnifferCAN para o Rádio Receptor. Este tráfego é sempre unidirecional e limita-se às informações coletadas no barramento CAN embarcado no veículo, cuja taxa de transmissão máxima é de 125 Kbps (Apêndice A).
- *Grande alcance*: A transmissão sem fio deve garantir a integridade dos dados para uma distância entre o SnifferCAN e o Rádio receptor de até 300 m (Apêndice A).
- *Arquitetura ponto-a-ponto*: A tecnologia de transmissão sem fio deve suportar a transmissão de dados ponto-a-ponto entre dois elementos, SnifferCAN e Rádio Receptor (Apêndice A). Preferencialmente, a tecnologia de transmissão deve permitir a adição de novos elementos no futuro.

Optou-se pela utilização de componentes da família CC13xx, da Texas Instruments, que oferece opções de SoCs baseados em microcontroladores ARM Cortex-M para a implementação comunicação sem fio. O diagrama em blocos do SoC CC13x0 é apresentado na Figura 22.

A família de componentes CC13xx faz parte da plataforma Texas Instruments SimpleLink™ que consiste em um amplo portfólio de produtos destinado ao desenvolvimento de aplicações sem fio. Cada produto da plataforma compartilha com seus pares um mesmo *Software Development Kit* (SDK), o que permite grande portabilidade de código entre eles. Ainda, o SDK da plataforma SimpleLink™ oferece a implementação pronta de diversas pilhas de protocolos consagrados, tais como Zigbee, Bluetooth, Thread, Wi-Fi, além da implementação de protocolos proprietários Texas Instruments, tanto para faixa Sub-GHz como para a faixa de 2,4 GHz. A Figura 23 apresenta algumas características da plataforma SimpleLink™.

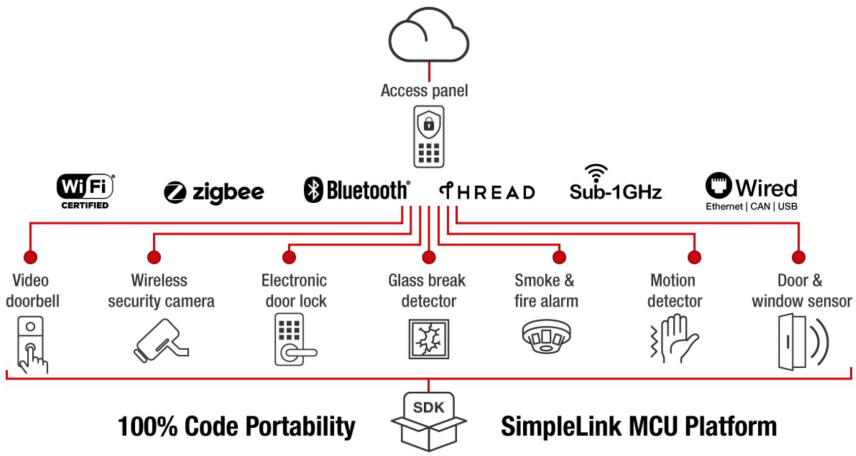
Neste projeto, utilizou-se os componentes SimpleLink™ CC1310 e CC1352 operando em frequências ISM Sub-GHz e implementando o protocolo de comunicação proprietário Texas Instruments Sub-GHz para obter boa performance no alcance de transmissão e no consumo de energia, ao custo de um volume de dados limitado a

Figura 22 – Diagrama de blocos do SoC CC13x0.



Fonte: Adaptado de Texas Instruments (2020).

Figura 23 – Plataforma Texas Instruments SimpleLink™



Fonte: Texas Instruments (2018b).

500 Kbps. A Figura 16 apresenta um comparativo resumido entre as tecnologias de transmissão suportadas pelos componentes da família Texas Instruments SimpleLink™.

Novamente, para acelerar o desenvolvimento da aplicação de transmissão sem fio, optou-se pela utilização de uma plataforma de desenvolvimento para o SoC

CC1310. A plataforma LAUNCHXL-CC13-90 é uma plataforma de desenvolvimento para o SoC CC1310 que, além das funções de *debug* e infraestrutura básica de *hardware* (alimentação, circuitos de *clock* e etc.), disponibiliza ainda uma interface (*front-end*) de *Rádio Frequência* (RF) baseada no circuito integrado CC1190. Esta interface de RF implementa amplificadores RF (*Amplificador de Potência, do inglês Power Amplifier* (PA) e*Amplificador de baixo ruído, do inglês Low Noise Amplifier* (LNA)) para melhorar a eficiência e estender o alcance do enlace de comunicação sem fio. Utilizou-se a plataforma LAUNCHXL-CC13-90, apresentada na Figura 24, na composição do subsistema SnifferCAN.

Figura 24 – Plataforma de desenvolvimento LAUNCHXL-CC13-90.

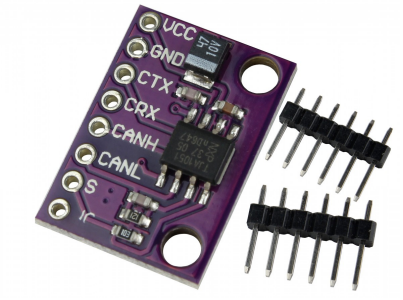
Fonte: Adaptado de Texas Instruments (2018a).

A plataforma LAUNCHXL-CC13-90 é utilizada no subsistema SnifferCAN como uma espécie de *modem*, permitindo que dados enviados do processador central (STM32F469DISCOVERY) sejam transmitidos sem fio. A conexão de *hardware* entre o processador central e o *gateway* é realizada por uma interface UART de 460.800 bps, suficiente para suportar o volume de dados entre os módulos.

Para completar os elementos de *hardware* necessários para o desenvolvimento do subsistema SnifferCAN, é necessário adicionar um *transceiver* CAN, visto que apesar de o núcleo STM32F469 disponibilizar um controlador CAN 2.0 A/B, a plataforma

não integra um *transceiver* para a necessária conversão entre o sinal de terminação única (*single-ended*) para o sinal diferencial implementado na camada física do protocolo. Para tanto, optou-se pela utilização de uma *breakout board*, isto é, uma placa de circuito impresso que oferece a infra-estrutura básica (capacitores de desacoplamento e circuitos de polarização) e acesso facilitado aos pinos de um determinado circuito integrado. A *breakout board* utilizada é apresentada na Figura 25.

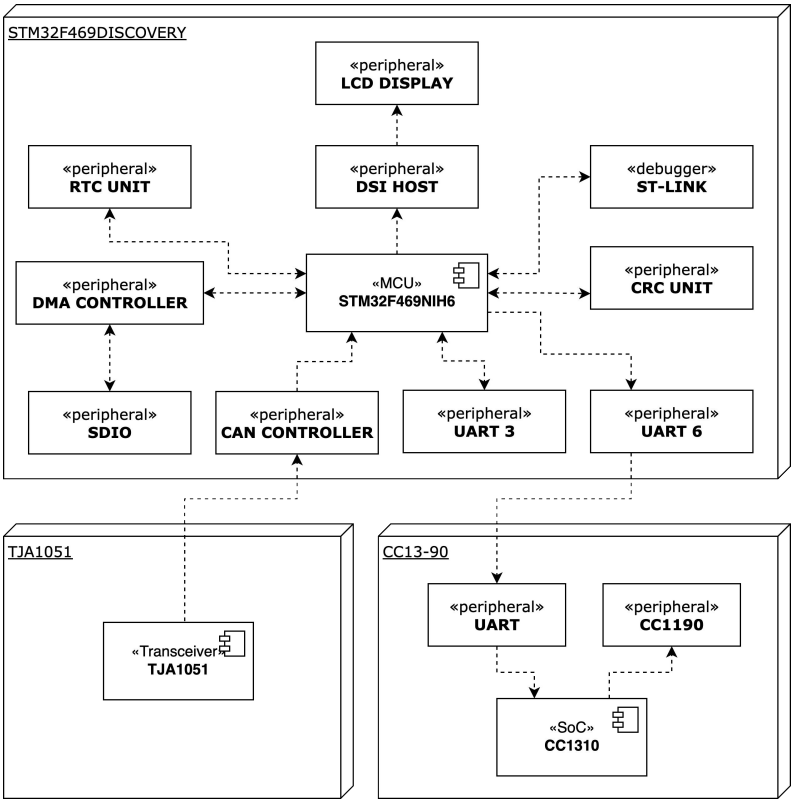
Figura 25 – *Breakout board* para o *transceiver* CAN TJA1051



Fonte: Adaptado de UsinaInfo (2021).

O conjunto completo de elementos de *hardware* integrados no subsistema SnifferCAN é apresentado na Figura 26.

Figura 26 – Diagrama de *hardware* completo do subsistema SnifferCAN.



Fonte: Autoria própria.

Por fim, para viabilizar a instalação do SnifferCAN no veículo, foi projetado um invólucro de modo a reduzir a um único volume o conjunto de elementos que compõem o subsistema, sem prejuízo da exposição dos elementos de interface com o usuário e com o veículo. O *software* Google SketchUp foi utilizado para o projeto e prototipação virtual do invólucro, sendo necessária a modelagem de cada um dos elementos do subsistema (placas de desenvolvimento, *breakout boards*, conectores, etc.). A documentação fornecida pelas fabricantes em (STMicroelectronics, 2021) e (Texas Instruments, 2021) foi utilizada como referência para a criação de modelos simplificados das duas plataformas de desenvolvimento. Os demais componentes foram modelados tomando as suas medidas com o auxílio de um paquímetro. Após modelado, o invólucro foi fabricado utilizando o processo de impressão 3D com filamentos em *Acrilonitrila Butadieno Estireno* (ABS), material que confere boa resistência mecânica e térmica, sem prejuízo no peso total do subsistema. O projeto do invólucro é apresentado na Figura 27.

Figura 27 – Projeto do invólucro do subsistema SnifferCAN

Fonte: Autoria própria.

4.4.2 Firmware

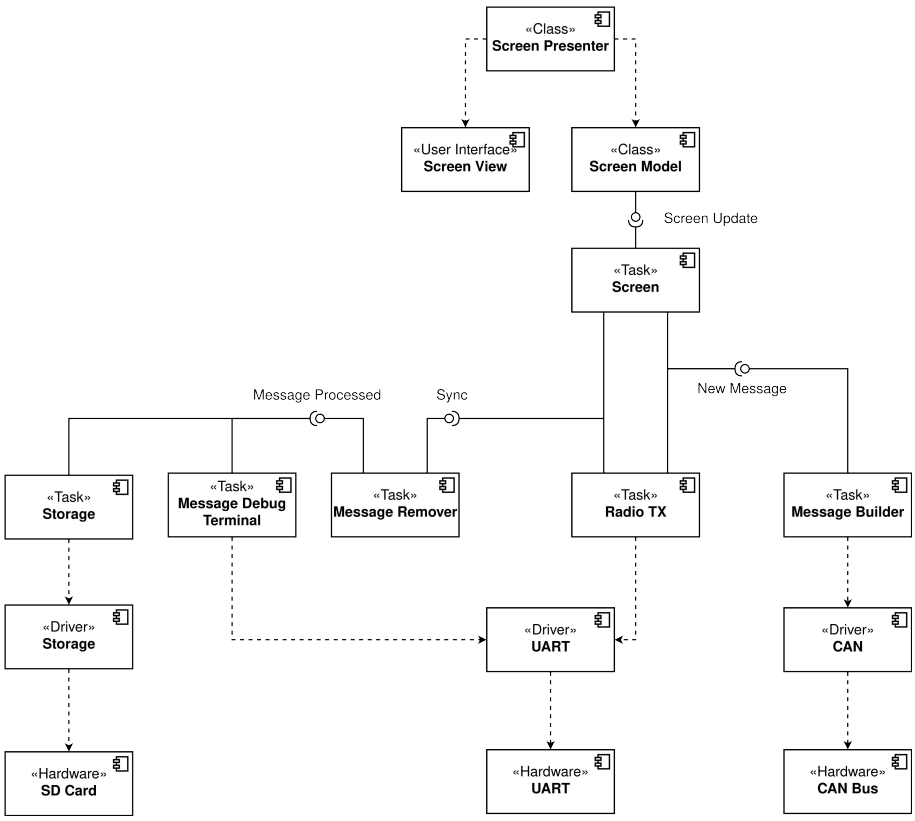
A arquitetura de *hardware* para o subsistema SnifferCAN discutida na subseção 4.4.1 e ilustrada na Figura 26, apresenta características de um sistema distribuído e assimétrico. Tais sistemas são caracterizados pela capacidade de multiprocessamento, utilizando múltiplas unidades independentes de processamento (neste caso microcontroladores) conectadas por uma interface de comunicação, para permitir o processamento colaborativo de dados. No caso específico do SnifferCAN, uma configuração diretor-executor (do inglês *leader-performer*), em que o microcontrolador STM32F469 assume o papel de *líder*, realizando grande parte das tarefas do processamento do sistema, tais como monitoramento do barramento CAN, armazenamento de dados na unidade de memória persistente e gerenciamento da GUI; e o microcontrolador CC1310 assume o papel de *executor*, atuando sob a coordenação do *líder* apenas quando a transmissão de mensagens por meio sem fio é solicitada. Do ponto de vista do *firmware*, sua arquitetura deve, também, refletir a hierarquia entre os microcontroladores.

O diagrama de componentes para o *firmware* embarcado no microcontrolador STM32F469 é apresentado na Figura 28.

Em linhas gerais, o *firmware* embarcado no microcontrolador STM32F469 é composto por 6 tarefas, cada uma responsável pelo controle de um aspecto específico do sistema, desde a leitura de mensagens no barramento CAN até a exibição dos dados relevantes na interface gráfica, além da coordenação do envio de dados sem fio. Abaixo encontra-se um resumo das tarefas implementadas no *firmware* do microcontrolador STM32F469:

- *Message Builder*: recepção e conversão da mensagem CAN em uma mensagem de aplicação[1].
- *Radio TX*: envio da mensagem de aplicação[1] para o microcontrolador CC1310 via interface serial, para que seja transmitida sem fio até a estação técnica da equipe.
- *Screen*: processamento das mensagens de aplicação[1] para exibição de dados relevantes ao piloto na tela embarcada.

Figura 28 – Diagrama de componentes para o *firmware* do SnifferCAN.



Fonte: Autoria própria.

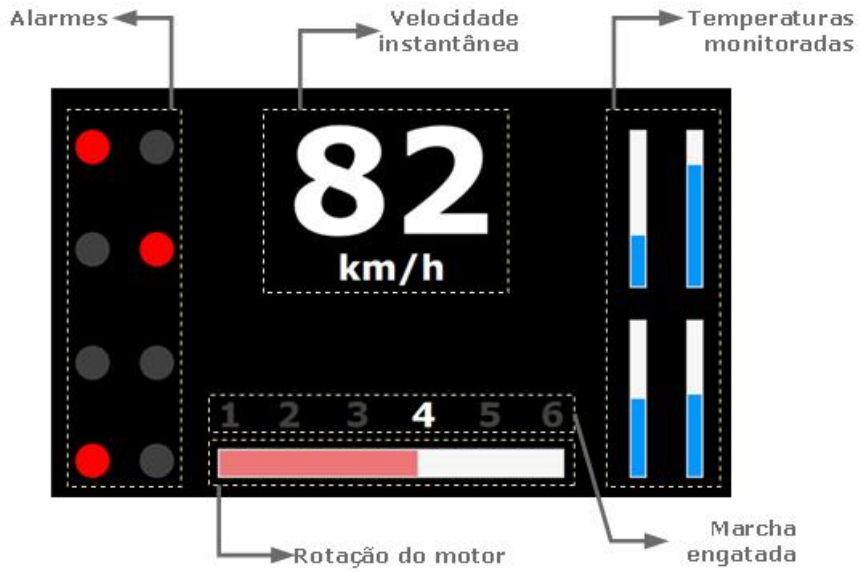
- *Storage*: armazenamento das mensagens de aplicação[1] em formato binário no SD *card*, para garantir persistência dos dados coletados no veículo.
- *Message Remover*: responsável pelo gerenciamento das filas de mensagens de aplicação[1], consumidas pelas demais tarefas.
- *Message Debug Terminal*: implementa funcionalidades de *debug*, permitindo a visualização das mensagens de aplicação [1] em um terminal serial, além da configuração de data e hora do módulo RTC.

O projeto da tela embarcada levou em consideração as informações de interesse do piloto apresentadas na Tabela 3, além disso a tela também conta com LEDs virtuais para a apresentação de futuros alarmes, caso seja do interesse da equipe. O resultado do projeto é ilustrado na Figura 29.

A Camada de Abstração de *hardware*, do inglês *Hardware Abstraction Layer* (HAL) fornecida pela STMicroelectronics (STM32Cube) oferece valiosos recursos de

[1] Chamamos de mensagem de aplicação o produto da decodificação e conformação de uma mensagem CAN de acordo com a estrutura de dados apresentada no Apêndice C.

Figura 29 – GUI para visualização de dados embarcados



Fonte: Autoria própria.

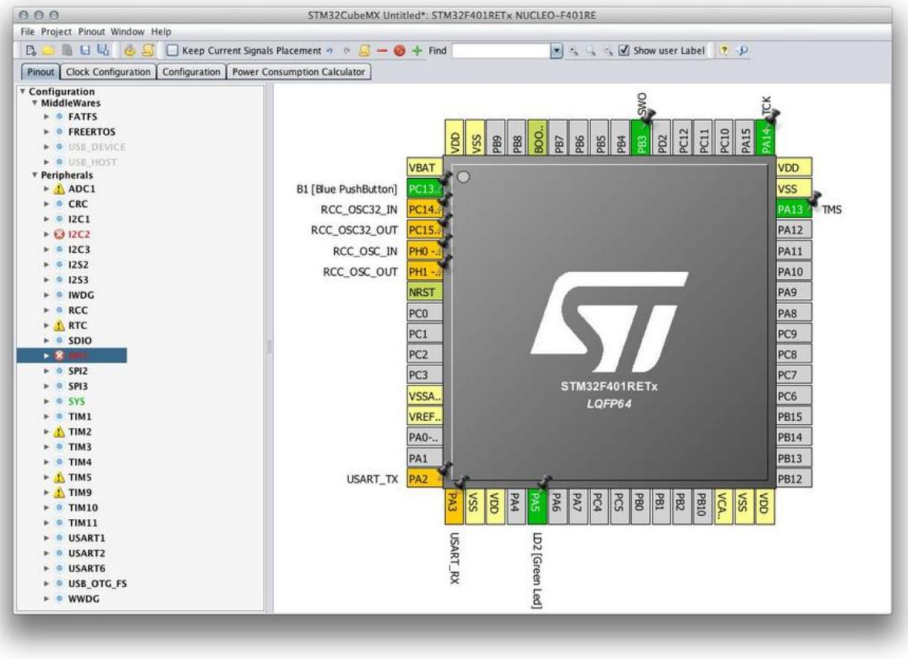
abstração que não só aceleram o processo de configuração inicial dos periféricos de *hardware*, mas também fornece uma série de *middlewares*, tais como RTOS, bibliotecas USB, pilhas TCP/IP, entre outros, além de permitir elevado grau de portabilidade de código entre microcontroladores da família STM32.

Neste projeto utilizou-se a ferramenta STM32Cube para a configuração dos periféricos UART, , SD, PLL, CAN, RTC e DSI. Além disso, empregou-se a ferramenta para a configuração do FreeRTOS, sistema operacional de tempo real utilizado para o gerenciamento das tarefas em execução no microcontrolador. A tela de configuração da ferramenta STM32Cube é apresentada na Figura 30.

Com relação ao fluxo de dados e sincronismo das tarefas apresentado na Figura 31, é possível agrupar as tarefas em duas categorias: tarefas críticas e não-críticas. As tarefas críticas devem ser atendidas quanto antes e não podem sofrer interferência de tarefas mais demoradas. Neste grupo estão as tarefas *Screen* e *UART Raw TX*. Já as tarefas não-críticas não precisam ser executadas imediatamente após a recepção de uma mensagem CAN e, por isso, possuem prazo mais elástico. Neste grupo estão as tarefas *Storage* e *Message Debug Terminal*.

Neste cenário, a tarefa *Message Remover* age como um intermediador entre as duas categorias, sincronizando as tarefas críticas e, posteriormente, disponibilizando

Figura 30 – Interface de configuração da ferramenta STM32Cube.



Fonte: Noviello (2018).

as mensagens processadas para as tarefas não-críticas.

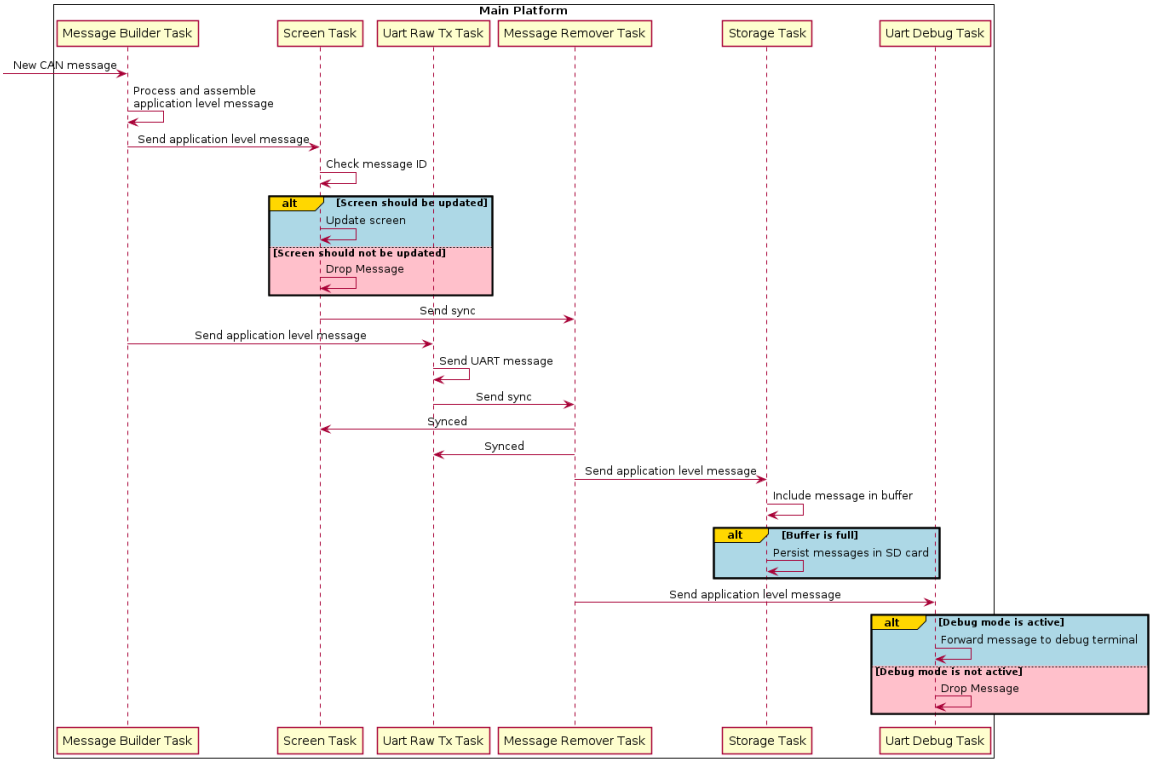
Ainda, outra peculiaridade do projeto é que a tarefa *Message Debug Terminal* permanece inativa no microcontrolador STM32F469 até que se receba uma mensagem de inicialização na interface serial dedicada ao *debug*. Somente então a tarefa é ativada e passa a decodificar comandos externos, que incluem tanto a configuração externa do módulo RTC como a impressão (*echo*) das mensagens de aplicação em tempo real.

O diagrama de atividades de cada tarefa descrita na Figura 31 é apresentado em detalhes no Apêndice D.

O *firmware* embarcado no microcontrolador CC1310 é modesto, dado o seu papel de *executor* na arquitetura diretor-executor implementada no SnifferCAN. A sua atribuição limita-se a atender as solicitações enviadas pelo microcontrolador STM32F469 (*diretor*) para a transmissão sem fio de mensagens de aplicação. Assim, o microcontrolador CC1310 executa apenas duas tarefas, em um arranjo muito semelhante ao *problema produtor-consumidor*[1]: o recebimento de mensagens de aplicação na interface (enviada pelo microcontrolador STM32F469) e o envio destas mensagens via comunicação sem fio. A arquitetura do firmware embarcado na plataforma TEXAS é estabelecida no diagrama de componentes apresentado pela Figura 32.

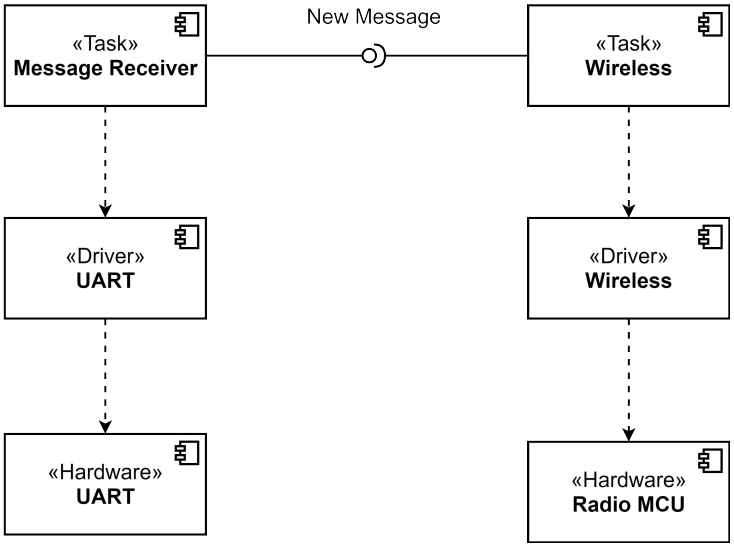
Do ponto de vista de configuração dos periféricos de *hardware* necessários

Figura 31 – Diagrama de sequência das tarefas executadas no microcontrolador STM32F469.



Fonte: Autoria própria.

Figura 32 – Diagrama de componentes para o *firmware* do rádio transmissor.

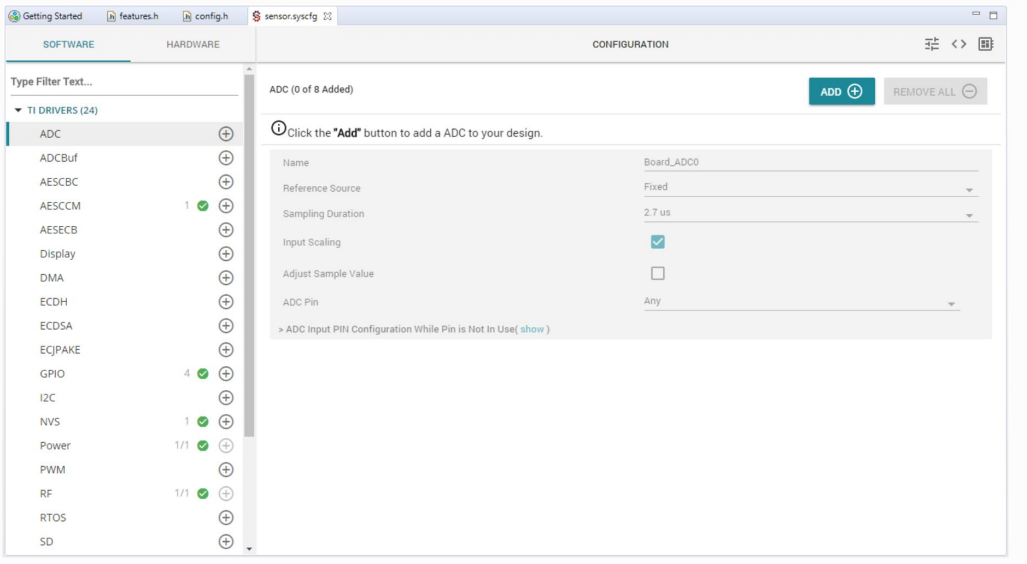


Fonte: Autoria própria.

para o suporte às tarefas de recebimento de mensagens via e transmissão sem fio de dados, a fabricante Texas Instruments oferece uma interface gráfica (*sysconfig*) para a configuração de dispositivos suportados pela HAL SimpleLink. A ferramenta *sysconfig* permite tanto a configuração de periféricos como interfaces (UART, GPIO, ADC e etc.)

como a a configuração de pilhas de protocolos (BLE, Zigbee, TI 15.4, etc.). Neste projeto empregou-se a ferramenta *sysconfig* para a configuração inicial do dispositivo, configurando tanto a pilha TI 15.4 utilizada para a comunicação sem fio, bem como os parâmetros a interface UART para comunicação com o microcontrolador STM32F469. A tela de configuração da ferramenta *sysconfig* é apresentada na Figura 33. Para o gerenciamento das tarefas em execução no microcontrolador CC1310, empregou-se o RTOS TI-RTOS (também chamado SYS/BIOS), sistema operacional fornecido e suportado pela fabricante Texas Instruments.

Figura 33 – Interface de configuração da ferramenta *sysconfig*.



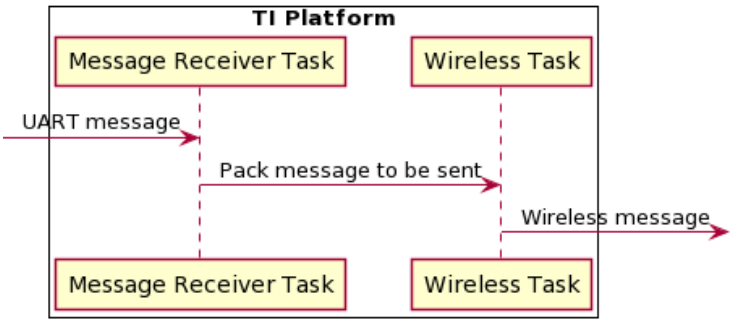
Fonte: Texas Instruments (2019b).

O fluxo de dados e sincronismo das tarefas implementadas no microcontrolador CC1310 é apresentado na Figura 34. A tarefa *message receiver* configura o periférico UART e o monitora continuamente em busca de pacotes enviados pelo microcontrolador STM32F469. Cada pacote recebido é incluído em uma fila de mensagens para que seja consumido pela tarefa *wireless TX*. A tarefa *wireless TX* realiza a configuração inicial do periférico RF quando o sistema é ligado e, após isso, somente volta a ser executada enquanto houverem entradas na fila de mensagens para serem consumidas, garantindo

[1] O *problema produtor-consumidor* é um exemplo clássico de sincronização entre processos. Nele, são descritos dois processos que compartilham um buffer de tamanho fixo. A atribuição do produtor é a geração e escrita indefinida de dados no buffer, enquanto o consumidor deve consumir tais dados, removendo-os do buffer, um de cada vez. O problema surge em assegurar que o produtor não adicione dados em um buffer cheio e nem o consumidor consuma um buffer vazio. Para solucionar o problema produtor-consumidor é necessário lançar mão de técnicas de sincronização de processos, tais como semáforos, filas de mensagens e etc.

a sincronização entre os processos e evitando o *problema produtor-consumidor*.

Figura 34 – Diagrama de sequência das tarefas executadas no microcontrolador CC1310.



Fonte: Autoria própria.

4.5 RÁDIO RECEPTOR

O rádio receptor atua como interface de recepção para a aplicação *desktop*, capturando as mensagens transmitidas no canal de comunicação sem fio e enviando-as, através de interface UART de 460.800 bps, para o computador onde a aplicação *desktop* está sendo executada.

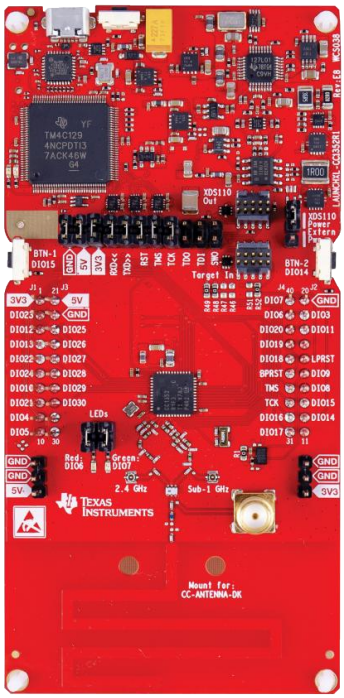
Assim como para o SnifferCAN, a Tabela 4 endereça os requisitos associados com o rádio receptor e, essencialmente, estipula apenas uma funcionalidade principal para o subsistema: recepção de dados sem fio. Levando em consideração tal característica e as decisões de projeto feitas na seção anterior, torna-se natural a escolha de uma placa de desenvolvimento da família SimpleLink™TI CC13xx, em particular a plataforma LAUNCHXL-1352R1, apresentada na figura Figura 35. Esta plataforma baseia-se no SoC CC1352R, um poderoso SoC para a implementação de aplicações de conectividade sem fio com suporte a múltiplos protocolos, dentre eles: Wi-Fi®, BLE, TI 15-4, Zigbee, etc.

4.5.1 Firmware Rádio Receptor

Tendo em vista que o rádio receptor age como contraparte do rádio transmissor, o *firmware* embarcado é igualmente simples, como apresentado no diagrama de componentes da Figura 36, e conta essencialmente com duas tarefas:

- *Wireless*: responsável por receber as os dados transmitidos no canal de comuni-

Figura 35 – Plataforma de desenvolvimento LAUNCHXL-CC1352R1.



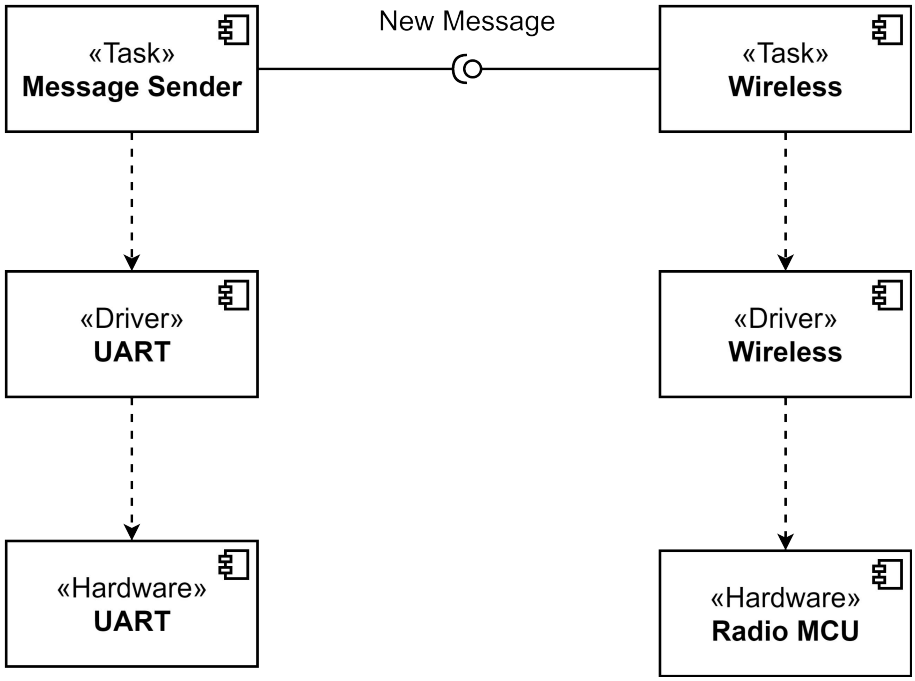
Fonte: Adaptado de Texas Instruments (2019a).

cação sem fio.

- *Message Sender*: responsável por tratar as mensagens recebidas pela tarefa *Wireless* e enviá-las para a aplicação *desktop* através da interface UART.

Além de atuarem como contrapartes, é importante lembrar que fazem parte da mesma família de microcontroladores, desta forma o *firmware* do rádio receptor foi desenvolvido utilizando as mesmas aplicações e ferramentas empregadas no rádio transmissor e apresentadas na subseção 4.4.2.

Figura 36 – Diagrama de componentes do rádio receptor.



Fonte: Autoria própria.

4.6 APLICAÇÃO DESKTOP

Como já discutido durante o desenvolvimento da solução neste capítulo, a aplicação *desktop* visa atender as necessidades da equipe técnica respeitando os requisitos estabelecidos para o subsistema na Tabela 4.

Para evitar que a equipe tenha gastos com a aquisição de *softwares* ou licenças, a aplicação foi implementada utilizando o *Qt*, um *framework* para desenvolvimento de aplicações gráficas, tanto *desktop* como embarcadas, utilizado amplamente em soluções comerciais, como computadores de bordo e interfaces gráficas de televisores inteligentes. Além de ser totalmente gratuito para uso, o *framework* apresenta diversas vantagens na elaboração da aplicação *desktop*, entre elas:

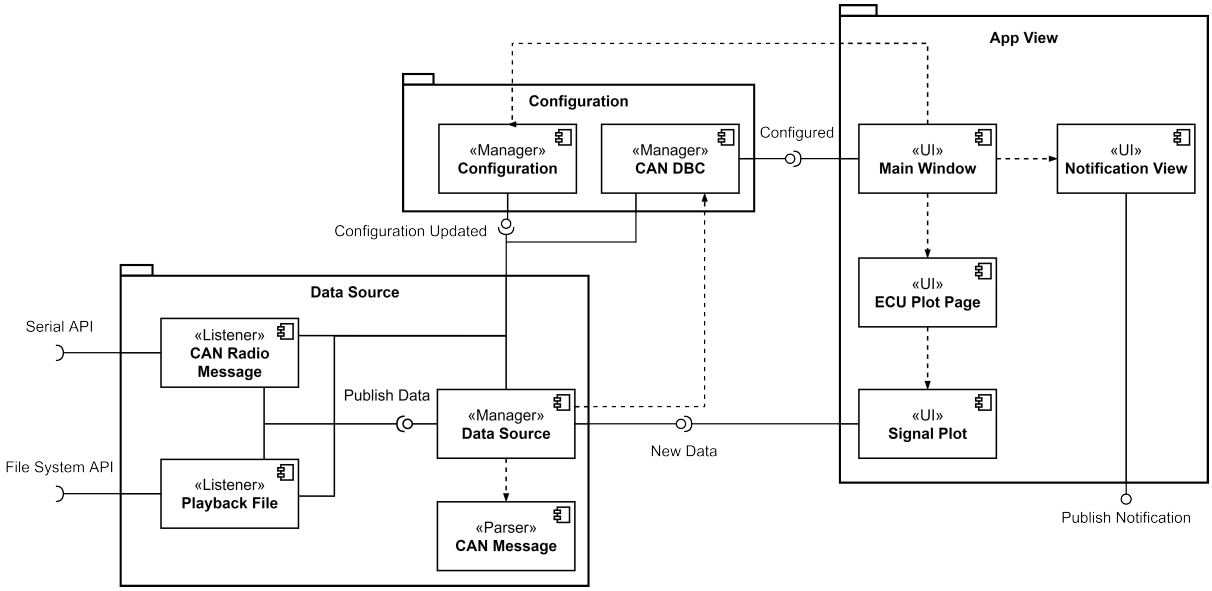
1. Utilizado de forma ampla comercialmente, sendo uma ferramenta relevante no mercado e amplamente validada.

2. Possui uma diversidade de bibliotecas já inclusas, permitindo que funcionalidades sejam implementadas tendo como base *software* testado e confiável.

3. *Framework* multiplataforma, permitindo que seja portado para outras plataformas de forma simples.

Para atender todos os requisitos solicitados, a arquitetura presente na Figura 37 foi idealizada e contempla os pacotes apresentados na Tabela 6.

Figura 37 – Diagrama de componentes da aplicação desktop.



Fonte: Autoria própria.

Tabela 6 – Pacotes presentes na aplicação *desktop*.

Pacote	Descrição
Data Source	Responsável pelo controle da fonte de dados que atualiza a interface gráfica.
App View	Responsável pela interface gráfica da aplicação, tanto dos dados como dos possíveis notificações para o usuário.
Configuration	Responsável por gerenciar as configurações feitas pelo usuário na aplicação.

Fonte: Autoria própria.

Cada pacote será descrito em detalhes nas próximas seções, descrevendo de forma detalhada o funcionamento e decisões de projeto de cada um dos componentes presentes no sistema.

4.6.1 Configuração da Aplicação

O pacote *Configuration*, apresentado na Figura 37, gerencia os parâmetros de operação da aplicação controlados pelo usuário, permitindo que o mesmo adapte a aplicação de acordo com suas necessidades através de um interface simples. Como já apresentado na Figura 37, o pacote conta com dois componentes para atender a funcionalidade descrita: *Configuration Manager* e *CAN DBC Manager*.

O componente *Configuration Manager* é o principal responsável pela configuração do sistema, controlando a interface de usuário e gerindo os parâmetros

configurados. O diagrama de atividade utilizado para modelar tal componente está presente no Apêndice E, mas, essencialmente, o mesmo permite que usuário informe três características globais da aplicação:

- Arquivo DBC que descreve o barramento dimensionado pela equipe.
- Modo de operação do sistema.
- Parâmetros de configuração específicos de cada um dos modos de operação.

A Tabela 7 apresenta os modos de operação disponíveis na aplicação e os parâmetros configurados para cada um dos modos.

Tabela 7 – Modos de operação e parâmetros de configuração.

Modo de Operação	Parâmetros
Desabilitado	
Tempo Real	Porta do Rádio Receptor *Baud Rate* do Rádio Receptor
Playback	Arquivo de registro de mensagens

Fonte: Autoria própria.

Já o componente *CAN DBC Manager* atua como um facilitador dentro da aplicação, sendo responsável por interpretar o arquivo DBC, configurado no componente *Configuration Manager*, e servir a informação do barramento de forma simples e estruturada aos demais componentes, sem que precisem acessar o arquivo diretamente. Para tal, o mesmo fornece as funcionalidades listadas abaixo para toda a aplicação:

- Descrição dos nós presentes no barramento.
- Descrição das mensagens presentes em um único nó do barramento.
- Descrição dos sinais presentes para uma mensagem em específico.
- Decodificação dos sinais presentes em uma mensagem.

Para implementar a capacidade de interpretação de arquivos DBC no componente *CAN DBC Manager*, foi utilizada a biblioteca *dbcppp* (XR3B0RN, 2021), livre para uso e focada unicamente em fornecer funcionalidade de *parse* de arquivos DBC com alto desempenho, além disto a mesma foi amplamente testada e validada, permitindo que a aplicação construída se torne mais robusta e confiável.

4.6.2 Recepção de Dados

O pacote *Data Source*, apresentado na Figura 37, gerencia as interfaces de recepção, validação das mensagens recebidas e tratamento de erros nas mensagens. Entretanto, o pacote foi idealizado de forma a atender não somente a funcionalidade descrita, como também os pontos apresentados a seguir:

- Separação entre as etapas de recepção de dados em cada uma das interfaces e o processo de *parse* das mensagens.
- Facilidade de inclusão de novas interfaces de recepção.
- Uma única interface confiável para emissão das mensagens recebidas pela aplicação.

Dessa forma, o mesmo foi estruturado como já apresentado na Figura 37 e conta com quatro componentes. O componente *Data Source Manager*, modelado pelo diagrama de atividade presente no Apêndice E, é o único responsável pela processo de *parse* e validação de mensagens recebidas pela aplicação e opera como interface para emissão de mensagens. Além disso, o componente também controla a propagação das mensagens dos componentes de recepção de acordo com o modo de operação configurado pelo usuário, portanto apenas as mensagens recebidas da interface configurada são emitidas e as demais são descartadas. Por sua vez, o componente *CAN Message Parser* age como suporte para o *Data Source Manager* e implementa o processo de conversão da mensagem de aplicação serializada para o formato desserializado, apresentado no Apêndice C.

Já os componentes *CAN Radio Message Listener* e *Playback File Listener*, modelados pelos diagramas de atividade apresentados no Apêndice E, são receptores específicos para as interfaces disponíveis no sistema atualmente e, portanto, implementam apenas os processos associados a recepção e validação de dados nas interfaces em questão.

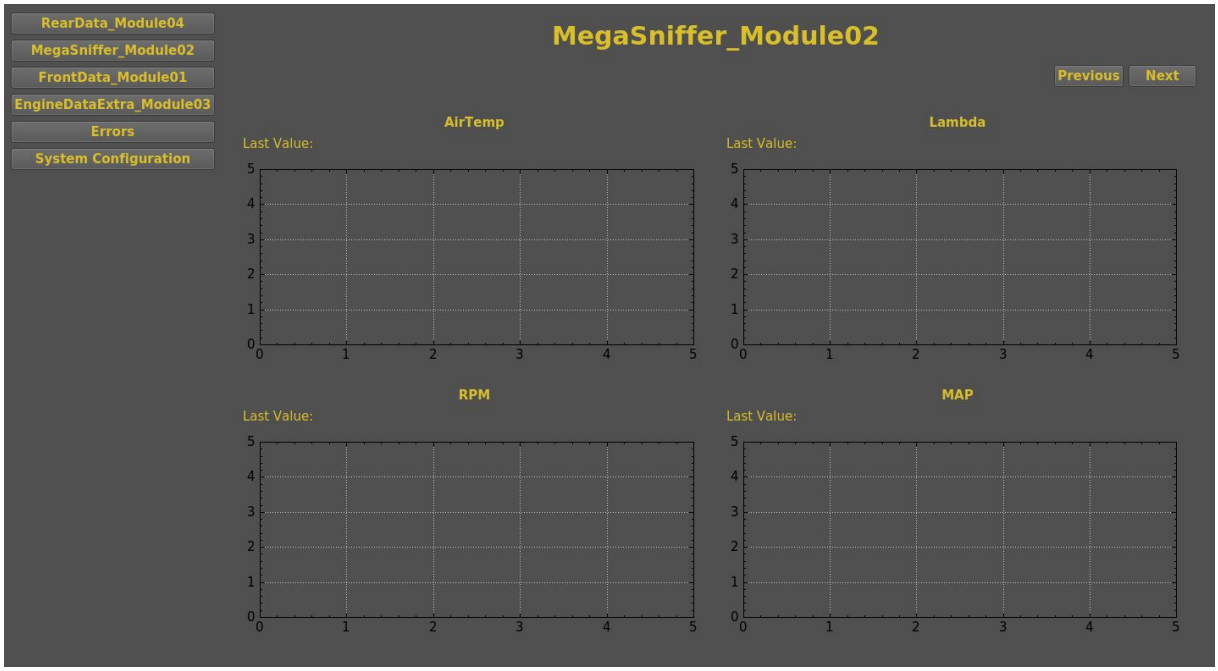
Sendo assim, fica claro que caso seja do interesse da equipe incluir uma nova forma de recepção de mensagens a aplicação, basta que um novo componente seja criado e que este se conecte a interface disponibilizada pelo componente *Data Source Manager*, enviando a mensagem no formato especificado pela interface.

4.6.3 Visualização

O pacote *App View*, apresentado na Figura 37, gerencia a interface gráfica para interação com o usuário, incluindo tarefas como exibição das informações recebidas do barramento CAN e de notificações. Como já apresentado no diagrama, o pacote foi projetado com quatro componentes, sendo o *Main Window UI* o principal deles e os demais atuando como componentes secundários.

O componente *Main Window UI* opera como ponto de entrada da aplicação e gerenciador do pacote, tratando os eventos de modificação do arquivo DBC e alterando dinamicamente a interface de exibição de dados, composta pelos componentes *ECU Plot Page UI* e *Signal Plot UI*, para que corresponda com a nova estrutura do barramento. Para tal, o *Main Window UI* executa uma instância do *ECU Plot Page UI* para cada ECU presente no barramento recebido e o último por sua vez apresenta um *Signal Plot UI* para cada sinal presente na ECU em questão. A operação de cada um dos componentes é apresentada no Apêndice E e a Figura 38 apresenta como a aplicação exibe os sinais disponibilizados pela ECU.

Figura 38 – Apresentação dos sinais disponibilizados pela ECU.



Fonte: Autoria própria.

O componente *Notication View UI* por sua vez opera como interface unificada para a apresentação de notificações na aplicação e desta forma qualquer outro com-

ponente presente na aplicação pode solicitar que uma notificação seja exibida, sendo assim a Figura 37 exibe a interface sem nenhuma conexão para simplificação do diagrama.

4.7 VERSIONAMENTO DE CÓDIGO

Um aspecto essencial para o desenvolvimento de *software* atualmente é o versionamento do mesmo, visto que os sistemas estão se tornando cada vez mais complexos e são compostos por diversos subsistemas. Plataformas de gerenciamento de repositórios de código facilitam o rastreamento de mudanças, permitindo que cada alteração tenha uma motivo claro e seja associada com um responsável. Tal gerenciamento, leva a aplicações com maior qualidade e mais robustas, pois a mesma é construída incrementalmente e cada alteração é registrada no gerenciador e, caso seja problemática, pode ser removida, retornando a aplicação a um estado estável e funcional.

Desta forma, é extremamente interessante que uma solução de versionamento seja utilizada para controlar o *software* desenvolvido durante a construção do sistema proposto, tanto para a aplicação como para as ferramentas de teste. Para tal, foi escolhido o GitLab, um gerenciador de repositório de *software* gratuito baseado em git. Além de permitir o gerenciamento de *software*, também conta com uma ferramenta para gerenciamento de tarefas, permitindo que alterações no repositório sejam associadas à tarefas específicas e só ocorram dada uma necessidade real conhecida. A Tabela 8 apresenta todos repositórios de *software* disponíveis.

Tabela 8 – Repositórios de software.

Aplicação	Link do Repositório
Firmware SAE CAN Sniffer	https://gitlab.com/tcc-sae/sae-cansniffer/
Firmware Rádio Transmissor	https://gitlab.com/tcc-sae/sae-can-rf-transmitter/
Firmware Rádio Receptor	https://gitlab.com/tcc-sae/sae-can-rf-transmitter/
SAE HMI App	https://gitlab.com/tcc-sae/sae-hmi-app/
Scripts de teste	https://gitlab.com/tcc-sae/sae-cansniffer/-/tree/master/Tools/

Fonte: Autoria própria.

5 TESTES E RESULTADOS

O principal resultado deste trabalho é o desenvolvimento de um protótipo para a coleta, armazenamento local e transmissão de dados embarcado em um carro de Fórmula SAE. Este protótipo é construído de forma a demonstrar, em cenário relevante, a viabilidade da aplicação proposta, conforme o nível 6 da escala de prontidão tecnológica tal como discutido em (ISO, 2015).

No processo de desenvolvimento de produtos, o nível 6 da escala de disponibilidade de tecnologia (TRL-6) é o nível em que já é possível avaliar quantitativamente o desempenho das tecnologias que compõem o produto, principalmente levando em consideração as peculiaridades que caracterizam o cenário de aplicação do produto.

A avaliação dos resultados deste trabalho foi realizada submetendo o protótipo desenvolvido a uma série de testes projetados para validar os requisitos de sistema apresentados no Apêndice A. Durante a aplicação dos testes, empregou-se ferramentas de simulação para fornecer dados de entrada ao sistema, modelados de acordo com os dados reais disponíveis no ambiente da aplicação (veículo de Fórmula SAE). Da mesma forma, outros aspectos fundamentais do cenário de aplicação foram levados em consideração, como o alcance da transmissão sem fio e a velocidade de deslocamento de um veículo de Fórmula SAE.

As próximas seções detalham os procedimentos dos testes realizados e apresentam os resultados obtidos.

5.1 TESTES EM CAMPO

Os testes em campo foram projetados para expor o protótipo construído em cenários semelhantes ao ambiente real de operação. Neste sentido, foram levados em consideração os aspectos fundamentais que caracterizam um veículo de Fórmula SAE e seu ambiente operacional, ou seja, o ambiente das competições da Fórmula SAE Brasil, para modelar um cenário de aplicação relevante.

As subseções a seguir apresentam as considerações feitas para a modelagem do contexto testes.

5.1.1 Cenário de aplicação de testes

Historicamente, desde a sua criação, o evento da Fórmula SAE no Brasil tem sido realizado no *Esporte Clube Piracicabano de Automobilismo* (ECPA), no interior do estado de São Paulo, local que também é sede das competições da categoria Baja SAE. A estrutura *"on-road"* do ECPA contempla uma pista de kart com 912 m de extensão, uma pista para automóveis com mais de 2000 m de extensão, além da infraestrutura de boxes, arquibancada, ambulatório entre outras estruturas de apoio aos eventos e competições de automobilismo.

Um sistema de telemetria, tal como qualquer sistema de comunicação sem fio, sofre grande influência do ambiente externo. O desempenho do sistema deve ser previsto tendo em vista a distância e altura relativa ao solo dos pontos de comunicação, o relevo, a quantidade e as características das obstruções na Linha de visão, do inglês *Line of Sight* (LoS), entre outros fatores.

Empregou-se a ferramenta QGIS para o levantamento das características fundamentais do ECPA, utilizando os dados de imagem georreferenciada do google para renderizar a imagem de satélite do local, e dados do radar PALSAR embarcado no satélite ALOS para obter um Modelo Digital de Elevação (MDE) de alta resolução. Através do MDE obtido é possível extrair o mapa de linhas de nível, com resolução de 5 m. O resultado do levantamento realizado é apresentado na Figura 39.

O levantamento das características fundamentais do cenário de aplicação real sugere que os testes devem ser executados em um local com ampla área aberta, sem obstruções na linha de visada e que permita a circulação controlada veículos para a execução de testes dinâmicos. A sede da Federação das Indústrias do Estado do Paraná (FIEP) em Curitiba possuí tais características, oferecendo condições seguras para a execução dos testes. Para validar a escolha do local de aplicação dos testes, utilizou-se novamente a ferramenta QGIS para o levantamento das características do local. Empregou-se o mapa de linhas de nível com resolução de 1 metro fornecido pelo Instituto de Pesquisa e Planejamento Urbano de Curitiba (IPPUC) para a avaliação da topologia do local. A Figura 40 mostra o resultado da avaliação do local de testes e a localização dos pontos de interesse.

O circuito planejado para a realização do teste dinâmico foi dividido em setores

Figura 39 – Esporte Clube Piracicabano de Automobilismo.

Fonte: Autoria própria.

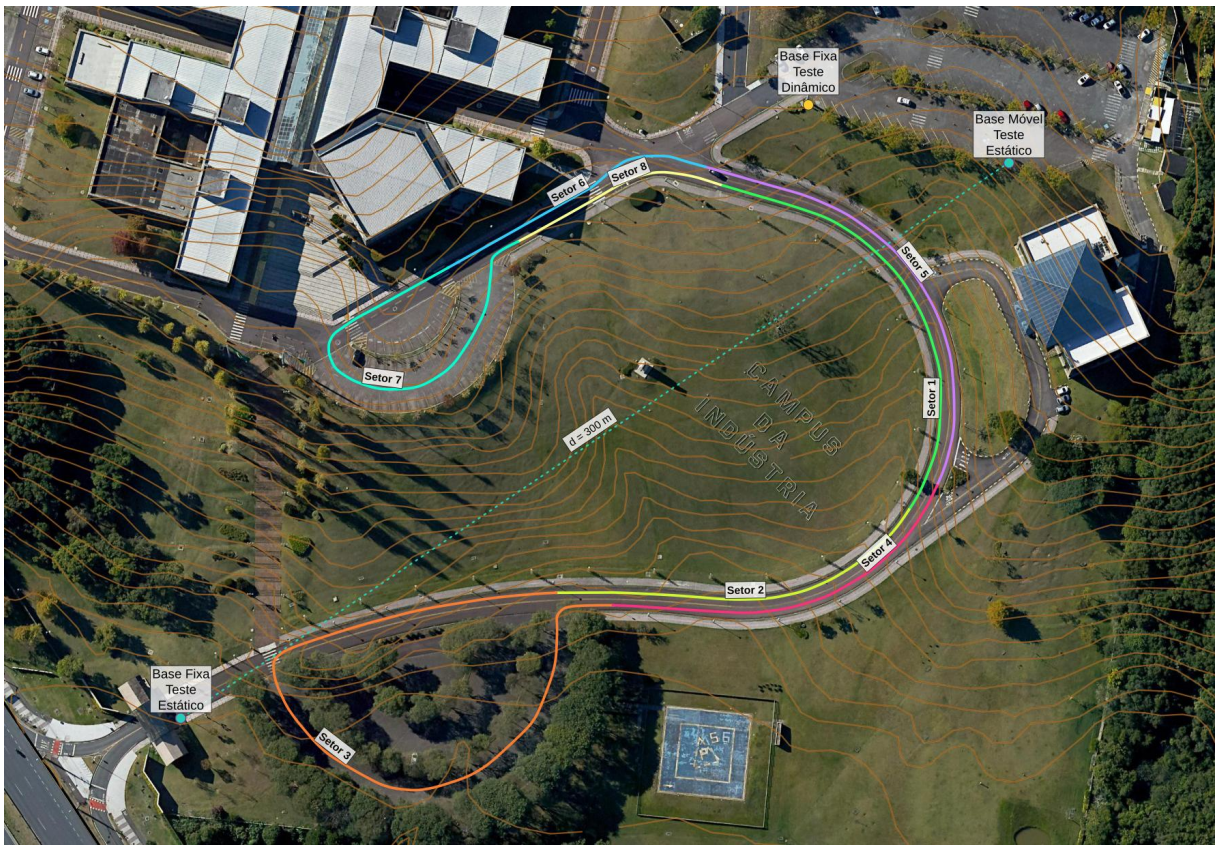
de acordo com as características relevantes para a transmissão sem fio, tais como velocidade relativa, qualidade da LoS e distância entre as bases. A Tabela 9 caracteriza cada setor deste circuito.

Tabela 9 – Características do circuito projetado para o teste dinâmico.

Setor	Velocidade	Linha de visada	Distância entre as bases
1	Alta	Ótima	Baixa
2	Média	Ruim	Média
3	Baixa	Ruim	Alta
4	Alta	Ruim	Média
5	Alta	Ótima	Baixa
6	Alta	Ótima	Baixa
7	Alta	Boa	Média
8	Baixa	Ótima	Baixa

Fonte: Autoria própria.

Figura 40 – Ambiente de testes - FIEP.



Fonte: Autoria própria.

5.1.2 Duração dos testes

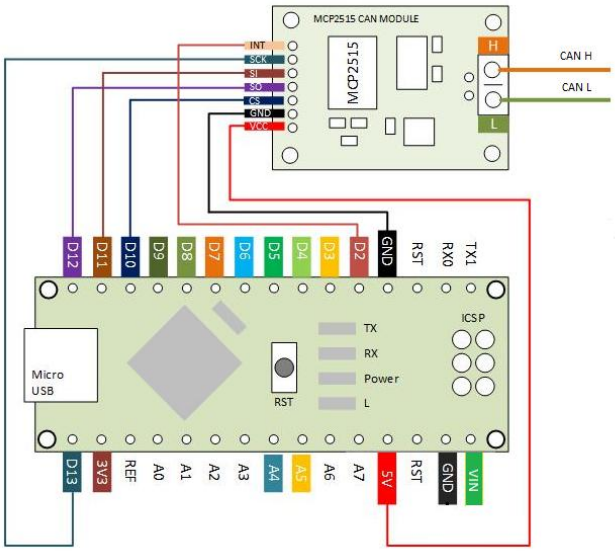
A duração dos testes foi calculada levando em conta a etapa mais longa da competição, determinada em (SAE, 2021). A prova *Endurance* possui um percurso total de 22 km e tem o traçado do circuito planejado para que a velocidade média do veículo esteja no intervalo entre 48 e 57 km/h. Assim, a duração da prova foi estimada entre 24 minutos e 28 minutos. Para cobrir o pior caso, optou-se por testes com duração total de 30 minutos.

5.1.3 Ocupação do barramento CAN

A determinação da máxima taxa de ocupação do barramento suportada pelo sistema dentro do limite imposto pelo **REQ1** foi feita com o auxílio de um simulador de barramento CAN, composto por um dispositivo Arduino Nano montado em conjunto com o módulo CAN MCP2515, como ilustrado na figura Figura 41. Este dispositivo foi programado para gerar mensagens similares àquelas presentes no barramento CAN

de um veículo de Fórmula SAE, com a vantagem de total controle na sua temporização. Além disso, o simulador desenvolvido oferece grande mobilidade, já que não depende de qualquer outro hardware externo para o seu funcionamento.

Figura 41 – Dispositivo simulador de barramento CAN.



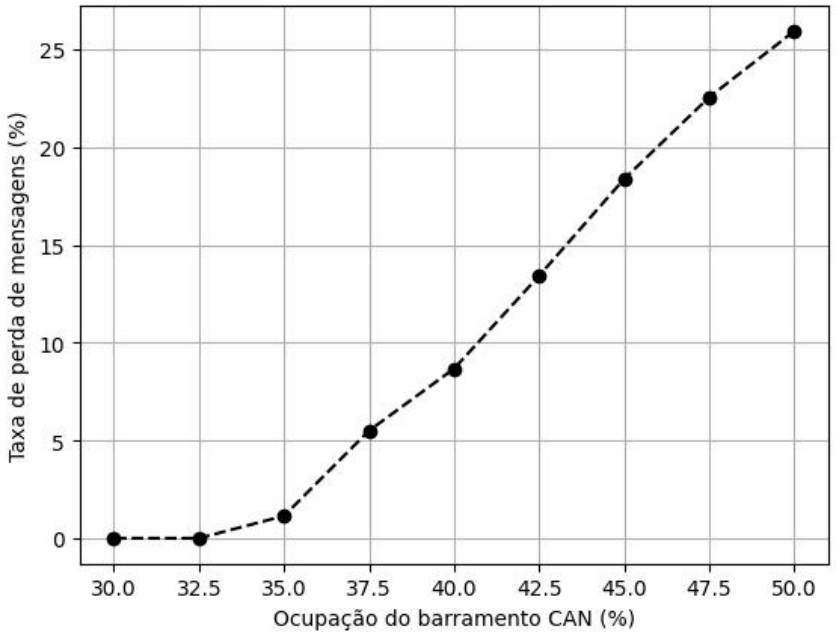
Fonte: Adaptado de Dmitry (2021).

A determinação da taxa de perda global de mensagens para uma dada ocupação do barramento foi realizada em bancada, considerando o envio de lotes de 50.000 mensagens entre o simulador de barramento e o SnifferCAN. A variação da taxa de ocupação do barramento foi controlada pelo intervalo de tempo entre as mensagens enviadas, calibrado com o auxílio de um osciloscópio Keysight DSOX 1102G com módulo de decodificação CAN. A curva resultante dos testes é apresentada na Figura 42.

Com o resultado obtido, e considerando o **REQ1**, fica claro que a máxima taxa de ocupação para o SnifferCAN operar adequadamente é de 40%. Entretanto, tendo em vista que o teste foi realizado em condições ideais, empregou-se uma margem de segurança para que, ao ser submetido a condições mais críticas, o sistema continue operando dentro dos limites estabelecidos pelo **REQ1**.

Assim, definiu-se a taxa de ocupação de 32,5% a ser utilizada durante os testes, implicando em uma taxa de perda de mensagens nula em bancada.

Figura 42 – Taxa de perda global de mensagens em função da ocupação do barramento CAN.



Fonte: Autoria própria.

5.1.4 Teste de simulação estática

O objetivo deste teste é a avaliação do desempenho do sistema quando submetido aos parâmetros mais restritivos modelados em **REQ1** e **REQ3** (Apêndice A), sem levar em conta fatores dinâmicos, tais como deslocamentos do transmissor ou de obstáculos ao longo da LoS. Para tal, o procedimento de teste abaixo foi elaborado:

1. Configurar o simulador de barramento CAN para a máxima taxa de ocupação.
2. Conectar o rádio receptor ao desktop que será utilizado como base fixa.
3. Executar a aplicação desktop.
4. Posicionar a base móvel a uma distância de 300 metros da base fixa.
5. Conectar o simulador de barramento CAN à base móvel.
6. Iniciar o envio de mensagens através do simulador de barramento CAN.
7. Coletar as informações de mensagens perdidas e Indicação de Força do Sinal Recebido, do inglês *Received Signal Strength Indicator* (RSSI) durante o tempo de duração do teste.

O teste foi realizado em três baterias de igual duração, conforme estipulado na subseção 5.1.2, amostrando os dados de interesse em dois momentos: ao atingir 50% e 100% da duração total da bateria. O resultado deste teste é apresentado na Tabela 10.

Tabela 10 – Resultados do teste de simulação estática.

Bateria	$\overline{RSSI}$ [dBm]	Tempo [$minutos$]	Taxa de perda de mensagens [$\%$]
1	-80,71	15	5,06
		30	4,56
2	-81,59	15	3,38
		30	3,37
3	-80,40	15	4,92
		30	6,98

Fonte: Autoria própria.

5.1.5 Teste de simulação dinâmica

O objetivo deste teste é a avaliação do desempenho do sistema em um cenário de aplicação semelhante ao ambiente real de operação de um veículo de Fórmula SAE. Neste teste são consideradas variáveis dinâmicas, como a aceleração e o deslocamento relativo entre as bases e a alteração da qualidade da LoS. Além disso, este teste expõe a unidade construída a interferências mecânicas, tais como vibrações e acelerações. O procedimento para a execução deste teste é descrito a seguir:

1. Configurar o simulador de barramento CAN para a máxima taxa de ocupação.
2. Conectar o rádio receptor ao desktop que será utilizado como base fixa.
3. Executar a aplicação desktop.
4. Conectar o simulador de barramento CAN à base móvel.
5. Embarcar a base móvel em um veículo
6. Iniciar o envio de mensagens através do simulador de barramento CAN.
7. Percorrer o circuito de testes com a base móvel embarcada no veículo.
8. Coletar as informações de mensagens perdidas e RSSI durante o tempo de duração do teste.

O teste foi dividido em três baterias, cada uma composta por 5 voltas no circuito apresentado na Figura 40, totalizando aproximadamente o limite de tempo apresentado em subseção 5.1.2.

As taxas de perda de mensagens obtidas para cada uma das baterias do teste são apresentadas na Tabela 11.

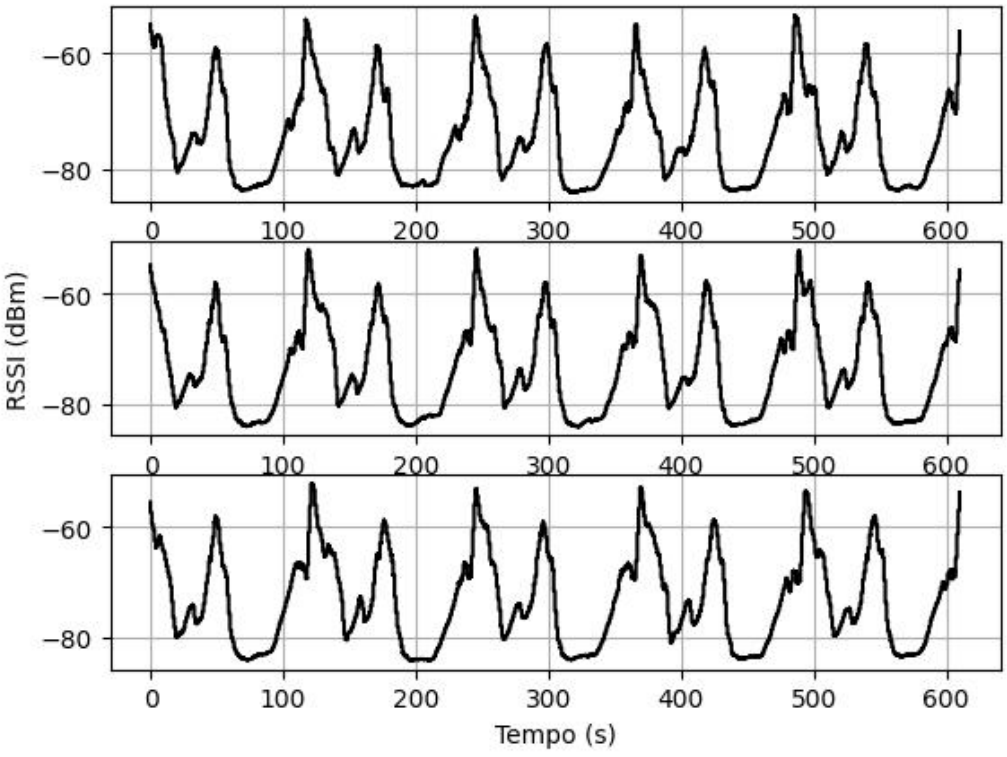
Tabela 11 – Taxas de perda de mensagens do teste de simulação dinâmica.

Bateria	Taxa de perda de mensagens [$\%$]	$\overline{RSSI}$ [dBm]
1	35,14	-69,84
2	40,00	-69,04
3	39,38	-68,95

Fonte: Autoria própria.

O comportamento do RSSI para cada uma das baterias é apresentado na Figura 43.

Figura 43 – Baterias do teste de simulação dinâmica.

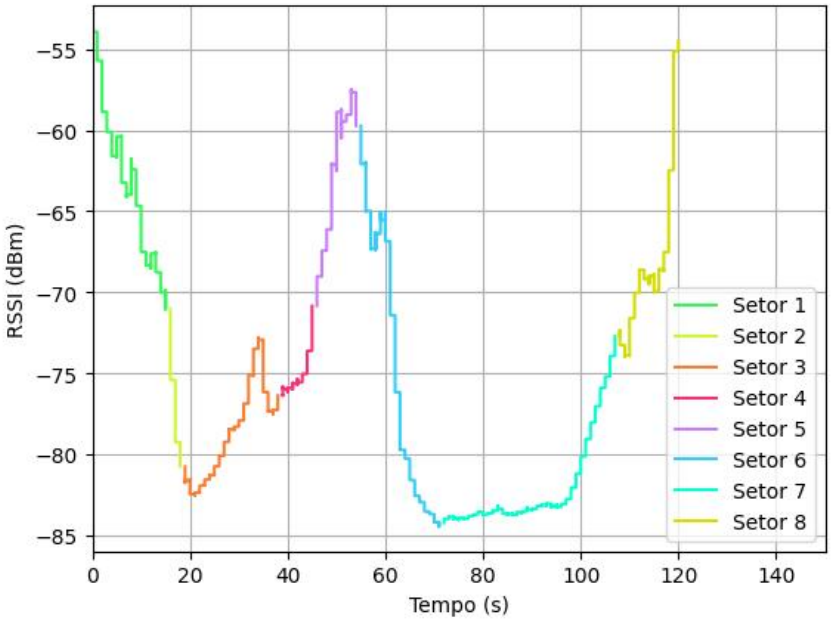


Fonte: Autoria própria.

A Figura 44 apresenta em detalhes o comportamento do RSSI em cada um dos setores do circuito de teste.

Ao expor o sistema ao cenário proposto para o teste dinâmico, o mesmo apresentou, como pode-se verificar na Tabela 11, resultados que extrapolam os limites

Figura 44 – Volta do teste de simulação dinâmica.



Fonte: Autoria própria.

propostos pelo **REQ1**. Entretanto, tal comportamento não reflete a robustez do sistema construído, uma vez que este apresenta bom desempenho nos setores 1 e 5, que contam com altas velocidades e ótima LoS, e nos testes estáticos, que apresentam grandes distâncias e boa LoS. A partir destas considerações, pode-se inferir que o fator limitante é a degradação da LoS, como pode ser observado nos setores onde o sistema apresenta RSSI abaixo do valor médio do circuito, como nos setores 2 e 3, devido ao perfil de elevação e vegetação da locação para os testes, e os setores 6 e 7, dado o fluxo constante de pedestres e veículos. Portanto este resultado indica a necessidade do bom posicionamento da base fixa, de modo a manter a melhor LoS possível e permitir que o sistema opere com taxas de perdas mais baixas.

5.2 TESTE DE ARMAZENAMENTO

O objetivo deste teste é a validação, em bancada, da capacidade de armazenamento das mensagens lidas do barramento CAN em mídia não volátil pelo período de tempo estipulado pelo **REQ2** quando exposto à altas taxas de ocupação do barramento. O procedimento de teste é apresentado abaixo:

1. Configurar o simulador de barramento CAN para a máxima taxa de ocupação.

2. Conectar o rádio receptor ao desktop que será utilizado como base fixa.
3. Executar a aplicação desktop.
4. Conectar o simulador de barramento CAN à base móvel.
5. Iniciar o envio de mensagens através do simulador de barramento CAN.
6. Coletar as informações de total de mensagens transmitidas e total de mensagens armazenadas durante o tempo de duração do teste.

O teste foi realizado em três baterias de igual duração e os dados de interesse foram amostrados ao final de cada uma das baterias. Os resultados obtidos são apresentado na Tabela 12.

Tabela 12 – Resultados do teste de armazenamento.

Bateria	Mensagens coletadas	Mensagens armazenadas (esperado)	Mensagens armazenadas (real)	Taxa de perda [%]
1	581.599	581.550	580.400	0,20
2	585.570	585.550	584.450	0,19
3	578.867	578.850	578.000	0,15

Fonte: Autoria própria.

5.3 TESTE DE REPLAY

O objetivo deste teste é a validação simultânea das funcionalidades de sinalização de falhas no barramento e importação de dados na aplicação desktop, exigidos pelos **REQ6** e **REQ7** apresentados no Apêndice A.

Para realização deste teste foi implementado um *script* gerador de arquivos de registro em Python, idênticos aos gravados pelo SnifferCAN em mídia não volátil durante a sua operação normal. Desta forma, é possível a injeção controlada de mensagens de erro no barramento CAN, combinadas com mensagens válidas. O SnifferCAN possui um conjunto de 23 identificadores para erros ocorridos no barramento CAN.

Neste teste, foi gerado um arquivo de registro contento cada um dos 23 possíveis identificadores de erro no barramento CAN, incluindo-os a cada 10 mensagens válidas, com uma mensagem por segundo, totalizando 230 mensagens. O procedimento detalhado do teste em questão é apresentado abaixo:

1. Utilizar o *script* para gerar um arquivo binário contendo tanto mensagens bem sucedidas como mensagens de erros.
2. Executar a aplicação desktop e configurá-la para o modo replay.
3. Importar o arquivo de registro gerado pelo *script*.
4. Iniciar a reprodução das mensagens.

Os erros identificados pela aplicação desktop durante a realização do teste são apresentados na Figura 45.

Figura 45 – Resultados do teste de replay.

Errors

Timestamp	Error
31/10/2021 15:40:06	Occurred an error in message 10 - error code 1
31/10/2021 15:40:16	Occurred an error in message 20 - error code 2
31/10/2021 15:40:26	Occurred an error in message 30 - error code 3
31/10/2021 15:40:36	Occurred an error in message 40 - error code 4
31/10/2021 15:40:46	Occurred an error in message 50 - error code 5
31/10/2021 15:40:56	Occurred an error in message 60 - error code 6
31/10/2021 15:41:06	Occurred an error in message 70 - error code 7
31/10/2021 15:41:16	Occurred an error in message 80 - error code 8
31/10/2021 15:41:26	Occurred an error in message 90 - error code 9
31/10/2021 15:41:36	Occurred an error in message 100 - error code 10
31/10/2021 15:41:46	Occurred an error in message 110 - error code 11
31/10/2021 15:41:56	Occurred an error in message 120 - error code 12
31/10/2021 15:42:06	Occurred an error in message 130 - error code 13
31/10/2021 15:42:16	Occurred an error in message 140 - error code 14
31/10/2021 15:42:26	Occurred an error in message 150 - error code 15
31/10/2021 15:42:36	Occurred an error in message 160 - error code 16
31/10/2021 15:42:46	Occurred an error in message 170 - error code 17
31/10/2021 15:42:56	Occurred an error in message 180 - error code 18
31/10/2021 15:43:06	Occurred an error in message 190 - error code 19
31/10/2021 15:43:16	Occurred an error in message 200 - error code 20
31/10/2021 15:43:26	Occurred an error in message 210 - error code 21
31/10/2021 15:43:36	Occurred an error in message 220 - error code 22
31/10/2021 15:43:46	Occurred an error in message 230 - error code 23

Fonte: Autoria própria.

5.4 TESTE DE VALIDAÇÃO DOS DADOS TRANSMITIDOS SEM FIO

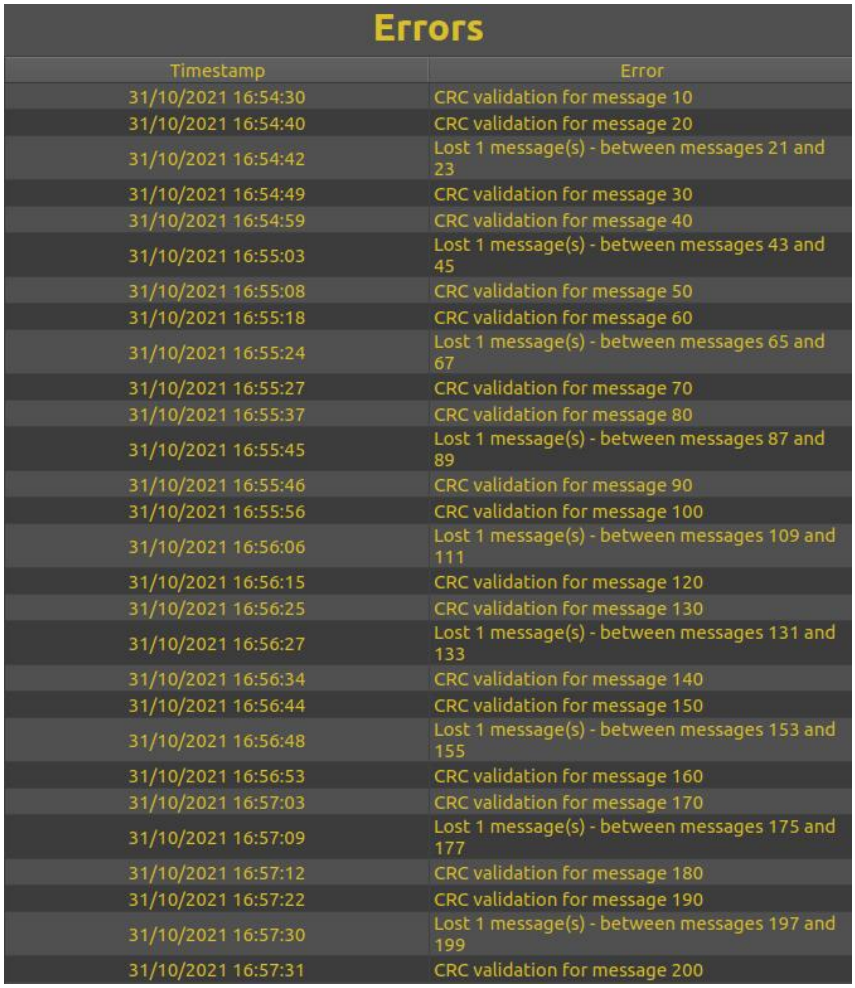
O objetivo deste teste é a verificação das funcionalidades de validação dos dados recebidos via canal de comunicação sem fio da aplicação desktop, conforme

exigido pelos **REQ9** e **REQ10** apresentados no Apêndice A.

Para tanto, desenvolveu-se um *script* gerador de mensagens de aplicação, idênticas àquelas recebidas do rádio receptor durante a operação normal do sistema. Nestas mensagens, injetou-se de forma controlada valores incorretos para os campos CRC e *sequence number* de forma a simular erros de corrupção e perda de mensagens, respectivamente. Foram geradas 200 mensagens para a reprodução do teste, a uma taxa de uma mensagem por segundo, com a injeção de 1 erro de corrupção de dados a cada 10 mensagens e 1 mensagem perdida a cada 22 mensagens. O procedimento do teste é detalhado a seguir:

1. Utilizar o *script* para gerar um conjunto de mensagens que simula erros de corrupção e perda na transmissão sem fio.
2. Executar a aplicação desktop e configurá-la para o modo replay.
3. Importar o arquivo de registro gerado pelo *script*.
4. Iniciar a reprodução das mensagens.

Figura 46 – Resultados do teste de validação dos dados transmitidos sem fio.



Errors

Timestamp	Error
31/10/2021 16:54:30	CRC validation for message 10
31/10/2021 16:54:40	CRC validation for message 20
31/10/2021 16:54:42	Lost 1 message(s) - between messages 21 and 23
31/10/2021 16:54:49	CRC validation for message 30
31/10/2021 16:54:59	CRC validation for message 40
31/10/2021 16:55:03	Lost 1 message(s) - between messages 43 and 45
31/10/2021 16:55:08	CRC validation for message 50
31/10/2021 16:55:18	CRC validation for message 60
31/10/2021 16:55:24	Lost 1 message(s) - between messages 65 and 67
31/10/2021 16:55:27	CRC validation for message 70
31/10/2021 16:55:37	CRC validation for message 80
31/10/2021 16:55:45	Lost 1 message(s) - between messages 87 and 89
31/10/2021 16:55:46	CRC validation for message 90
31/10/2021 16:55:56	CRC validation for message 100
31/10/2021 16:56:06	Lost 1 message(s) - between messages 109 and 111
31/10/2021 16:56:15	CRC validation for message 120
31/10/2021 16:56:25	CRC validation for message 130
31/10/2021 16:56:27	Lost 1 message(s) - between messages 131 and 133
31/10/2021 16:56:34	CRC validation for message 140
31/10/2021 16:56:44	CRC validation for message 150
31/10/2021 16:56:48	Lost 1 message(s) - between messages 153 and 155
31/10/2021 16:56:53	CRC validation for message 160
31/10/2021 16:57:03	CRC validation for message 170
31/10/2021 16:57:09	Lost 1 message(s) - between messages 175 and 177
31/10/2021 16:57:12	CRC validation for message 180
31/10/2021 16:57:22	CRC validation for message 190
31/10/2021 16:57:30	Lost 1 message(s) - between messages 197 and 199
31/10/2021 16:57:31	CRC validation for message 200

Fonte: Autoria própria.

5.5 MASSA TOTAL DO DISPOSITIVO EMBARCADO

Este experimento consiste na validação da massa total do dispositivo, conforme a limitação imposta pelo **REQ11** apresentado no Apêndice A. A verificação foi feita com o auxílio de uma balança SF-400, com capacidade máxima de até 10 kg e resolução de 1 g. A massa total do dispositivo embarcado é de 251 g, conforme mostra a Figura 47.

Figura 47 – Massa total do dispositivo embarcado

Fonte: Autoria própria.

5.6 CUSTOS

A restrição de custos imposta pelo **REQ12** apresentada no Apêndice A foi verificada através da cotação de preços dos componentes disponíveis em mercado nos principais fornecedores, como Mouser e Digikey, e fabricantes das partes que constituem o sistema. A cotação dos componentes foi tomada em Dólar para diminuir a variação temporal dos preços, tendo em vista a instabilidade do Real. Ainda, não foram considerados custos extras, tais como frete e impostos, pois dependem diretamente da localização e legislação no momento da compra.

O custo de equipamentos de uso geral, tal como o computador para a execução da aplicação de visualização de dados na base fixa, não foram levados em conta para o custo do sistema.

A Tabela 13 apresenta o custo médio de cada componente necessário para a construção do sistema, bem como o custo total do sistema.

Tabela 13 – Custos da construção do sistema.

Componente	Subsistema	Preço (US$)
STM32F469DISCOVERY	SnifferCAN	59,00
LAUNCHXL-CC13-90	SnifferCAN	39,00
Transceiver CAN TJA1051	SnifferCAN	2,93
Conversor ajustável de tensão LM317	SnifferCAN	2,45
Cartão MicroSD	SnifferCAN	3,29
Invólucro[1]	SnifferCAN	10,00
Conector M12 de 5 vias (painel)	SnifferCAN	7,06
Antena 915 MHz	SnifferCAN	6,38
LAUNCHXL-CC1352R1	Rádio receptor	39,99
Antena 915 MHz	Rádio receptor	6,38
Total		**176,48**

Fonte: Autoria própria.

[1] Visto que o invólucro foi fabricado sob-demanda, o custo do mesmo representa o valor pago para a sua fabricação.

6 CONCLUSÕES E PERSPECTIVAS

Este trabalho propõe o desenvolvimento de uma unidade de telemetria, armazenamento e visualização de dados para a utilização no protótipo de Fórmula SAE desenvolvido pela equipe Fórmula UTFPR. Esperamos com isso, disponibilizar para a equipe uma ferramenta de acompanhamento do desempenho do protótipo, permitindo o refinamento contínuo do projeto e um consequente aumento do aproveitamento durante as competições.

Entretanto, para viabilizar a integração deste trabalho com o projeto da equipe Fórmula UTFPR, foram determinadas uma série de requisitos de sistemas apresentados no Apêndice A e validados através dos métodos apresentados no Capítulo 5.

Do ponto de vista estático, o sistema atendeu aos requisitos propostos, conforme apresentado na subseção 5.1.4. Observamos, ainda, a possibilidade de expansão da taxa limite de ocupação do barramento CAN, visto que a taxa média de perda de mensagens do sistema se manteve inferior a 50% do limite estipulado pelo **REQ1**.

Do ponto de vista dinâmico, conforme apresentado na subseção 5.1.5, ao expor o sistema a um cenário com o deslocamento da base móvel, não foi possível observar relação direta entre a velocidade de deslocamento e um aumento na perda de mensagens devido ao efeito doppler, em especial quando comparamos os valores de RSSI dos setores mais rápidos (1 e 5) apresentados na Figura 44 com o valor de RSSI médio do circuito, apresentado na Tabela 11. Entretanto, notamos que a redução da LoS exerceu papel fundamental para o aumento da taxa de perda de mensagens. Isso pode ser notado nos setores 2 e 3, devido ao perfil de elevação do terreno, e nos setores 6 e 7, dado o fluxo de pedestres e veículos. Embora as taxas de perda de mensagens apresentadas na Tabela 11 não atendam ao **REQ1**, consideramos que este não seja um indicativo negativo do desempenho do sistema, mas um reflexo da locação para o teste dinâmico. Conforme apresentado na Figura 40, o circuito do teste possui terreno acidentado, regiões cercadas por vegetação e tráfego constante de pedestres e veículos, fatores que impactam diretamente na qualidade de LoS. O resultado apresentado na Tabela 10 para o teste estático corrobora com essa premissa, em que dada uma LoS ótima, mesmo o sistema operando com níveis de RSSI muito menores do que o RSSI médio obtido no teste dinâmico (devido a distância entre as

bases), as taxas de perda de mensagens se mantiveram abaixo do limite estipulado pelo **REQ1**. Neste sentido, fica clara a importância da determinação da posição ótima de instalação da base fixa para garantir o desempenho ideal do sistema.

Apesar do sucesso com relação ao funcionamento geral, o sistema projetado apresentou problemas para cumprir os requisitos **REQ2** e **REQ12**.

Com relação ao **REQ2**, não há margem para nenhuma taxa de perda de mensagens no armazenamento local embarcado no veículo, por menor que seja. Entretanto, durante os testes identificamos uma perda de 0,2% no armazenamento de mensagens em mídia não volátil, taxa essa que impede a validação do requisito, apesar de considerarmos um resultado aceitável para a aplicação.

Já com relação ao **REQ12**, os custos para construção do sistema ficaram 17,65% acima do projetado no requisito. Apesar deste valor representar um aumento considerável, podemos atribuí-lo a utilização estrita de componentes *off-the-shelf*, que em contrapartida permitem também expansões no projeto com reduzida adição de novos componentes. Tais expansões podem incluir novas funcionalidades de acordo com os interesses da equipe, como por exemplo o desenvolvimento de comunicação bi-direcional, canal de áudio ou ainda a migração para 2,4 GHz.

Independente do resultado positivo do projeto, no decorrer do mesmo superamos uma série de dificuldades que valem ser ressaltadas. A primeira, e principal delas, foi a integração dos ambientes de desenvolvimento fornecidos pela STMicroelectronics, que são distribuídos de forma isolada e não contam com uma documentação detalhada para a sua utilização, tornando o processo por vezes confuso e demorado. Da mesma forma, a documentação fornecida pela Texas Instruments para a camada física do protocolo de comunicação sem fio.

Além disso, tivemos dificuldades para atingir a robustez no sistema, em especial quando submetido a altas taxas de ocupação no barramento CAN. Neste sentido, foi necessário reavaliar os parâmetros de priorização das tarefas no RTOS e a temporização das atividades bloqueantes.

Por fim, cabe ainda ressaltar que devido a pandemia do Covid-19 declarada pela OMS, este projeto foi inevitavelmente desenvolvido de forma remota, nos obrigando a fazer concessões de escopo, tal como a não-execução de testes em conjunto com a equipe.

6.1 TRABALHOS FUTUROS

Tendo em vista que o projeto foi desenvolvido com o intuito de atender as necessidades iniciais da equipe Fórmula UTFPR na implantação de um sistema de telemetria, o seu escopo foi concentrado no desenvolvimento dos componentes fundamentais que suportam tal aplicação. Assim, é importante considerar o universo de novas funcionalidades e oportunidades de melhoria de desempenho e robustez que se abrem a partir do desenvolvimento deste trabalho.

Do ponto de vista das possíveis melhorias no sistema, a mais marcante delas é o aumento da taxa de ocupação máxima do barramento CAN suportada pelo Sniffer-CAN. Essa melhoria pode ser atingida através da redução do tamanho da mensagem de aplicação, passando de uma estrutura de dados para um agrupamento binário, por exemplo. Esta alteração garante um *overhead* muito menor para as mensagens de aplicação, diminuindo consideravelmente a ocupação dos canais de transmissão e, consequentemente, liberando espaço para um fluxo maior de dados na entrada do sistema. De forma semelhante, a troca da interface de comunicação entre a STM32 e o rádio transmissor por uma alternativa mais veloz, tal como um barramento SPI.

Por outro lado, a inclusão de novas funcionalidades pode incluir o desenvolvimento de comunicação bi-direcional entre os agentes do sistema permitindo, por exemplo, desde a implementação de uma cada de transporte completa, até a configuração remota de parâmetros de desempenho do veículo. Por fim, é possível considerar a implantação de um canal de voz para a comunicação em tempo real entre o piloto e a equipe de engenheiros nos boxes.

REFERÊNCIAS

ABOUT Qt - Qt Wiki. 2019. Disponível em: https://wiki.qt.io/About_Qt#References.

ASSOCIATION, SD Card. **Physical Layer Simplified Specification**. SD Card Association, 2020. Disponível em: https://www.sdcard.org/downloads/pls/pdf?p=Part1_Physical_Layer_Simplified_Specification_Ver8.00.jpg&f=Part1_Physical_Layer_Simplified_Specification_Ver8.00.pdf&e=EN_SS1_8.

BARATA, Juliano. **Equipe Imperador UTFPR: Fórmula SAE, Baja... diversidade é isso aí!** 2014. Disponível em: https://flatout.com.br/equipe-imperador-utfpr-formula-sae-baja-diversidade-e-isso-ai16446/.

BRAUNE, Nils. **Telemetry Unit for a Formula StudentRace Car**. 2014. Disponível em: https://pub.tik.ee.ethz.ch/students/2013-HS/SA-2013-67.pdf.

CAPACITY (SD/SDHC/SDXC/SDUC) | SD Association. 2021. Disponível em: https://www.sdcard.org/developers/sd-standard-overview/capacity-sd-sdhc-sdxc-sduc/.

CARDEN, Frank; JEDLICKA, Russell P.; HENRY, Robert. **Telemetry systems engineering**. *[S.l.]*: Artech House, 2002. (Artech House telecommunications library). ISBN 978-1-58053-257-0.

CASE, Dean E. Formula SAE - Competition History 1981 - 1996. *In*: . *[s.n.]*, 1996. p. 962509. Disponível em: https://www.sae.org/content/962509/.

CLASSIFICACAO Final Formula SAE 2012. São Paulo, 2012. 1 p. Disponível em: http://saebrasil.org.br/wp-content/uploads/2020/03/FSAE-2012-Classifica%C3%A7%C3%A3o-Final.pdf.

COMPANY, Qt. **Qt Widgets 5.15.3**. 2021. Disponível em: https://doc.qt.io/qt-5/qtwidgets-index.html.

CORRIGAN, Steve. **Introduction to the Controller Area Network (CAN)**. Texas Instruments, 2016. Disponível em: https://www.ti.com/lit/pdf/sloa101.

Dmitry. **Hardware:**. 2021. Original-date: 2016-03-13T09:33:18Z. Disponível em: https://github.com/autowp/arduino-mcp2515.

EE Times. **Embedded Markets Study (2019)**. 2019. Disponível em: https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf.

EICH, Florian. Bidirectional CAN bus telemetry on a Formula Student/SAE car. p. 114, 2016.

ELLIOT, Jock. Telemetry helps Formula SAE team close the loop on design. **SAE Off-Highway Engineering**, p. 3, 2012. Disponível em: http://articles.sae.org/11264/.

FatFs - Generic FAT Filesystem Module. 2021. Disponível em: http://elm-chan.org/fsw/ff/00index_e.html.

GALITZ, Wilbert O. **The essential guide to user interface design: an introduction to GUI design principles and techniques**. 3rd ed. ed. *[S.l.]*: Wiley Pub, 2007. OCLC: ocm76792111. ISBN 978-0-470-05342-3.

HENNESSY, John L. **Computer Architecture: A Quantitative Approach**. 6. ed. *[S.l.]*: Morgan Kaufmann Publishers, 2019. ISBN 978-0-12-811905-1.

IBRAHIM, Dogan. **SD card projects using the PIC microcontroller**. *[S.l.]*: Newnes/Elsevier, 2010. OCLC: ocn298783710. ISBN 978-1-85617-719-1.

INSTRUMENTS, Texas. TI-RTOS 2.20 user's guide. p. 133, 2016.

ISO. **ISO16290**. 2015.

KEY Priorities for Sub-GHz Wireless Deployment. Silicon Labs, 2017. Disponível em: https://www.silabs.com/documents/public/white-papers/Key-Priorities-for-Sub-GHz-Wireless-Deployments.pdf.

LEEN, G.; HEFFERNAN, D. Expanding automotive electronic systems. **Computer**, v. 35, n. 1, p. 88–93, jan. 2002. ISSN 00189162. Disponível em: http://ieeexplore.ieee.org/document/976923/.

LEITE, Leandro de Brito. MÓDULO SENSOR DE MOVIMENTO E POSICIONAMENTO INTEGRADO À REDE CAN. São Paulo, p. 60, 2012.

LIBRARY, Space Science; DEFENSE, D.D. **Systems engineering fundamentals**. CreateSpace Independent Publishing Platform, 2016. ISBN 978-1-5377-0346-6. Disponível em: https://books.google.com.br/books?id=XcwXvgAACAAJ.

LOPES, Wellington Carlos. **Análise de desempenho do protocolo CAN para aplicação na área agrícola utilizando redes de Petri coloridas**. jun. 2007. Tese (Mestrado em Manufatura) — Universidade de São Paulo, São Carlos, jun. 2007. Disponível em: http://www.teses.usp.br/teses/disponiveis/18/18145/tde-13062008-110448/.

MAINI, Anil K. **Digital Electronics**. John Wiley & Sons, Ltd, 2007. ISBN 978-0-470-51052-0 978-0-470-03214-5. Disponível em: http://doi.wiley.com/10.1002/9780470510520.

MARWEDEL, Peter. **Embedded System Design**. Springer International Publishing, 2018. (Embedded Systems). ISBN 978-3-319-56043-4 978-3-319-56045-8. Disponível em: http://link.springer.com/10.1007/978-3-319-56045-8.

MASTERING the FreeRTOSTM Real Time Kernel: A Hands-On Tutorial Guide. 2016. Disponível em: https://www.freertos.org/fr-content-src/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf.

MAURER, Markus; WINNER, Hermann (Ed.). **Automotive Systems Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36454-9 978-3-642-36455-6. Disponível em: http://link.springer.com/10.1007/978-3-642-36455-6.

McLaren Racing - A brief history of computing in Formula 1. 2016. Disponível em: https://www.mclaren.com/racing/team/a-brief-history-of-computing-in-F1-1052199/.

MEANS, Kenneth H. **History of Mini Baja East**. 2001. Disponível em: http://students.sae.org/competitions/bajasae/history.pdf.

NASCIMENTO JUNIOR, Danternei Lucas Do; SOUZA, Verônica Luiza Ladeira De. **Implementação da eletrônica embarcada na viatura baja do Exército Brasileiro**. 2014. Tese (Doutorado) — Instituto militar de Tecnologia, Rio de Janeiro, 2014. Disponível em: https://bdex.eb.mil.br/jspui/bitstream/123456789/7946/1/PFC%20Danternei%20-%20Ver%c3%b4nica%20aC-19561.pdf.

NAVET, Nicolas; SIMONOT-LION, Françoise (Ed.). **Automotive embedded systems handbook**. Boca Raton: CRC Press, 2009. (Industrial information technology series). OCLC: ocn231680179. ISBN 978-0-8493-8026-6.

NOERGAARD, Tammy. **Embedded systems architecture: a comprehensive guide for engineers and programmers**. Second edition. *[S.l.]*: Elsevier/Newnes, 2013. ISBN 978-0-12-382196-6.

NOVIELLO, Carmine. **"Mastering STM32."A step-by-step guide to the most complete ARM Cortex-M platform, using a free and powerful development environment based on Eclipse and GCC**. 0.18. ed. *[S.l.]*: Leanpub, 2018.

NUNES, Tomaz Filgueira. Telemetria de um veículo Baja SAE através de rede CAN. p. 49, 2016.

PAZUL, Keith. **Controller Area Network (CAN) basics**. Microchip, 2005. Disponível em: https://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en011694.

ROSSO, Marco Antônio Zolett. **Desenvolvimento do controle eletrônico do acelerador**. 2017. Tese (Doutorado) — Universidade Federal de Santa Catarina, Joinville, 2017.

SAE. **Formula SAE Rules 2020**. 2021. Disponível em: https://fsaeonline.com/cdsweb/gen/DownloadDocument.aspx?DocumentID=1b6bda52-48d0-4286-931d-c9418165fd3e.

SENOUCI, B; ZITOUNI, R. FPGA based Networked Embedded Systems Design and Prototyping: Automobile oriented Applications. **physical Systems**, p. 6, 2016.

SHANKER, Shreejith. **Enhancing automotive embedded systems with FPGAs**. 2016. Tese (Doutorado) — Nanyang Technological University, 2016. Disponível em: http://hdl.handle.net/10356/67064.

SHERRIFF, Nicholas. **Learn Qt 5: build modern, responsive cross-platform desktop applications with Qt, C++, and QML**. *[S.l.: s.n.]*, 2018. OCLC: 1026402279. ISBN 978-1-78847-885-4.

STMicroelectronics. **Discovery kit with STM32F469NI MCU**. *[S.l.]*, 2020. 30 p. Disponível em: https://www.st.com/resource/en/user_manual/dm00218846-discovery-kit-with-stm32f469ni-mcu-stmicroelectronics.pdf.

STMICROELECTRONICS. **Model-View-Presenter Design Pattern | TouchGFX Documentation**. 2021. Disponível em: https://support.touchgfx.com/docs/development/ui-development/software-architecture/model-view-presenter-design-pattern.

STMicroelectronics. **STM32F4DISCOVERY - CAD Resources**. 2021. Disponível em: https://www.st.com/en/evaluation-tools/stm32f4discovery.html#cad-resources.

STMICROELECTRONICS. **What is TouchGFX? | TouchGFX Documentation**. 2021. Disponível em: https://support.touchgfx.com/docs/introduction/what-is-touchgfx.

SULTANA, Author Jason. **Data acquisition and telemetry in Formula 1**. 2020. Disponível em: https://formulaoneinsights.com/data-acquisition-and-telemetry-in-formula-1/.

TAN, Frank Yi. Development of Electric Formula SAE Real-time Telemetry Software. p. 101, 2011.

Texas Instruments. **LAUNCHXL-CC13-90 User Manual**. 2018. Disponível em: https://www.ti.com/lit/ug/swau104a/swau104a.pdf?ts=1629935054986&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FLAUNCHXL-CC13-90.

Texas Instruments. **SimpleLink™ Microcontroller Platform**. 2018. Disponível em: https://www.ti.com/lit/ml/swat002b/swat002b.pdf?ts=1629766722868&ref_url=https%253A%252F%252Fwww.google.com%252F.

Texas Instruments. **CC1352R1 LaunchPad Quick Start Guide**. 2019. Disponível em: https://www.ti.com/lit/ml/swru525e/swru525e.pdf.

Texas Instruments. **Get started with SysConfig — SimpleLink™ CC13x2 / CC26x2 SDK Thread User's Guide 1.02.04.00 documentation**. 2019. Disponível em: https://software-dl.ti.com/simplelink/esd/simplelink_cc13x2_26x2_sdk/2.40.00.81/exports/docs/thread/html/sysconfig/sysconfig.html.

Texas Instruments. **CC13x0, CC26x0 SimpleLink™ Wireless MCU - Technical Reference Manual**. 2020. Disponível em: https://www.ti.com/lit/ug/swcu117i/swcu117i.pdf?ts=1629832908239&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCC2650MODA.

Texas Instruments. **LAUNCHXL-CC13-90 Design Files**. 2021. Disponível em: https://www.ti.com/tool/LAUNCHXL-CC13-90#design-files.

TREMAYNE, David. **The science of Formula 1 design: expert analysis of the anatomy of the modern Grand Prix car**. 3rd ed. ed. Sparkford, NR Yeovil, Somerset, U.K. : Newbury Park, Calif: Haynes Pub. ; Haynes North America, 2009. ISBN 9781844257188.

USINAINFO. **Módulo CAN BUS Arduino TJA1051**. 2021. Disponível em: https://www.usinainfo.com.br/outros-modulos-arduino/modulo-can-bus-arduino-tja1051-5448.html.

Vetor Informatik GmbH. **DBC File Format Documentation**. 2007. Disponível em: http://read.pudn.com/downloads766/ebook/3041455/DBC_File_Format_Documentation.pdf.

VISCONTI, Paolo; SBARRO, B; FAZIO, R de; LAY-EKUAKILLE, A. Design and Testing of an Electronic Control System Based on STM X-Nucleo Board for Detection and Wireless Transmission of Sensors Data Applied to a Single-Seat Formula SAE Car. **INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS**, v. 65, n. 4, p. 671–678, 2019. Disponível em: http://ijet.ise.pw.edu.pl/index.php/ijet/article/view/10.24425-ijet.2019.130248/627.

WANG, K.C. **Embedded and Real-Time Operating Systems**. Springer International Publishing, 2017. ISBN 978-3-319-51516-8 978-3-319-51517-5. Disponível em: http://link.springer.com/10.1007/978-3-319-51517-5.

WHITE, Elecia. **Making embedded systems: design patterns for great software**. 1. ed. ed. *[S.l.]*: O'Reilly, 2012. OCLC: 815881891. ISBN 978-1-4493-0214-6.

WILLIAMSON, Roy; BEASLEY, Jon. Automotive technology and manufacturing readiness levels: a guide to recognised stages of development within the automotive industry. **URN 11/672**, p. 7, 2011. Disponível em: https://www.smmt.co.uk/wp-content/uploads/sites/2/Automotive-Technology-and-Manufacturing-Readiness-Levels.pdf.

WIRELESS Connectivity Technology Selection Guide. Texas Instruments, 2020. Disponível em: https://www.ti.com/lit/sg/swat016/swat016.pdf?ts=1610049873146&ref_url=https%253A%252F%252Fwww.google.com%252F.

XR3B0RN. **dbcppp**. 2021. Disponível em: https://github.com/xR3b0rn/dbcppp.

APÊNDICES

APÊNDICE A – REQUISITOS DE SISTEMA

Tabela 14 – Requisitos de sistema

Requisito	Descrição	Prioridade
REQ1	**Coleta de dados do barramento:** O sistema deve operar com a máxima ocupação possível do barramento CAN cuja taxa de perda de mensagens é inferior a 10%, considerando um período de 30 minutos de operação.	Alta
REQ2	**Registro em mídia não-volátil:** O sistema deve ser capaz de armazenar localmente os dados coletados no carro em formato binário, correspondentes a pelo menos 30 minutos de testes, considerando um barramento com as mesmas condições definidas no requisito **REQ1**	Alta
REQ3	**Transmissão de dados para base remota:** O sistema deve ser capaz de transmitir, em tempo real, os dados coletados no veículo para uma base remota, via comunicação sem fio, distante em até 300 m do veículo, considerando o mesmo barramento definido no requisito **REQ1**.	Alta
REQ4	**Exibição de dados embarcada:** O sistema deve fornecer uma interface gráfica local para a exibição de dados coletados no veículo.	Alta
REQ5	**Exibição de dados remotos:** O sistema deve fornecer uma aplicação gráfica que permita a visualização dos dados coletados remotamente, em tempo real.	Alta
REQ6	**Sinalização de falhas no barramento:** O sistema deve ser capaz de sinalizar eventos de falhas ocorridos no meio físico (barramento), tais como *stuff-bit error*, *ack error* e *form error*.	Alta
REQ7	**Importação de dados:** A aplicação gráfica definida no requisito **REQ5** deve suportar a importação de dados gravados localmente no veículo em mídia não-volátil, permitindo a reconstrução dos eventos em formato (*replay*).	Média
REQ8	**Importação de configurações:** A aplicação gráfica definida no requisito **REQ5** deve ser capaz de importar arquivos de modelagem de barramento CAN (DBC) para a identificação automática dos rótulos e escalas das mensagens exibidas.	Baixa
REQ9	**Detecção de erros na transmissão remota:** A aplicação gráfica definida no requisito **REQ5** deve ser capaz de identificar mensagens corrompidas durante a transmissão sem fio.	Alta
REQ10	**Detecção de perda de mensagens na transmissão remota:** A aplicação gráfica definida no requisito **REQ5** deve ainda ser capaz de identificar a perda de mensagens em tempo real.	Média
REQ11	**Restrição de peso:** A unidade embarcada no veículo deve possuir peso total inferior a 300g.	Média
REQ12	**Restrição de custo:** O custo total do sistema deve ser inferior a USD 150,00	Média

Fonte: Autoria própria.

APÊNDICE B – PARÂMETROS MONITORADOS

Tabela 15 – Sensores e parâmetros monitorados pela equipe Fórmula UTFPR

Módulo CAN	Parâmetro	Val. Min.	Val. Max.	Unidade	Largura [bits]	Resolução	Prioridade	Mensagens [s^{-1}]	Deadline [s]
Dianteiro	Aceleração lateral (Frontal)	-4	4	m/s^2	16	122E-6	Média	50	0,02
Dianteiro	Aceleração longitudinal (Frontal)	-4	4	m/s^2	16	122E-6	Média	50	0,02
Dianteiro	Aceleração lateral (CG)	-4	4	m/s^2	16	122E-6	Média	50	0,02
Dianteiro	Aceleração longitudinal (CG)	-4	4	m/s^2	16	122E-6	Média	50	0,02
Dianteiro	Ângulo de direção	-225	225	°	8	1,75	Baixa	25	0,04
Dianteiro	Posição pedal freio	0	35	°	8	0,13	Baixa	25	0,04
Dianteiro	Tensão da bateria	0	18	V	8	0,07	Baixa	25	0,04
Dianteiro	Corrente da bateria	-15	30	A	8	0,17	Baixa	25	0,04
Turbo	Temperatura do ar no restritor	0	100	°C	8	0,39	Média	50	0,02
Turbo	Temperatura do ar na entrada do turbo	0	200	°C	8	0,78	Alta	100	0,01
Turbo	Temperatura do ar na saída do turbo	0	200	°C	8	0,78	Alta	100	0,01
Turbo	Pressão do ar no restritor	0,5	3	bar	8	0,009	Média	50	0,02
Turbo	Pressão do ar no turbo	0,5	3	bar	8	0,009	Alta	100	0,01
Turbo	Pressão do ar na saída do turbo	0,5	3	bar	8	0,009	Alta	100	0,01
Injeção	Temperatura do líquido de arrefecimento	0	100	°C	8	0,39	Alta	100	0,01
Injeção	Posição da borboleta (TPS)	0	90	°	8	0,35	Alta	100	0,01
Injeção	Rotação do motor	0	13000	rpm	8	50,78	Alta	100	0,01
Injeção	Pressão do ar no coletor	0,5	3	bar	8	0,009	Alta	100	0,01
Injeção	Temperatura do Ar no coletor de admissão	0	100	°C	8	0,39	Média	50	0,02
Injeção	Volume de Oxigênio pós-combustão	0	5	V	8	0,02	Alta	100	0,01
Injeção	Velocidade na saída do câmbio	0	13000	rpm	8	50,78	Média	50	0,02
Traseiro	Aceleração lateral (Traseira)	-4	4	m/s^2	16	122E-6	Média	50	0,02
Traseiro	Aceleração longitudinal (Traseira)	-4	4	m/s^2	16	122E-6	Média	50	0,02
Traseiro	Nível de combustível	0	10	L	8	0,04	Baixa	25	0,04

Fonte: Autoria própria.

APÊNDICE C – ESTRUTURA DA MENSAGEM DE APLICAÇÃO

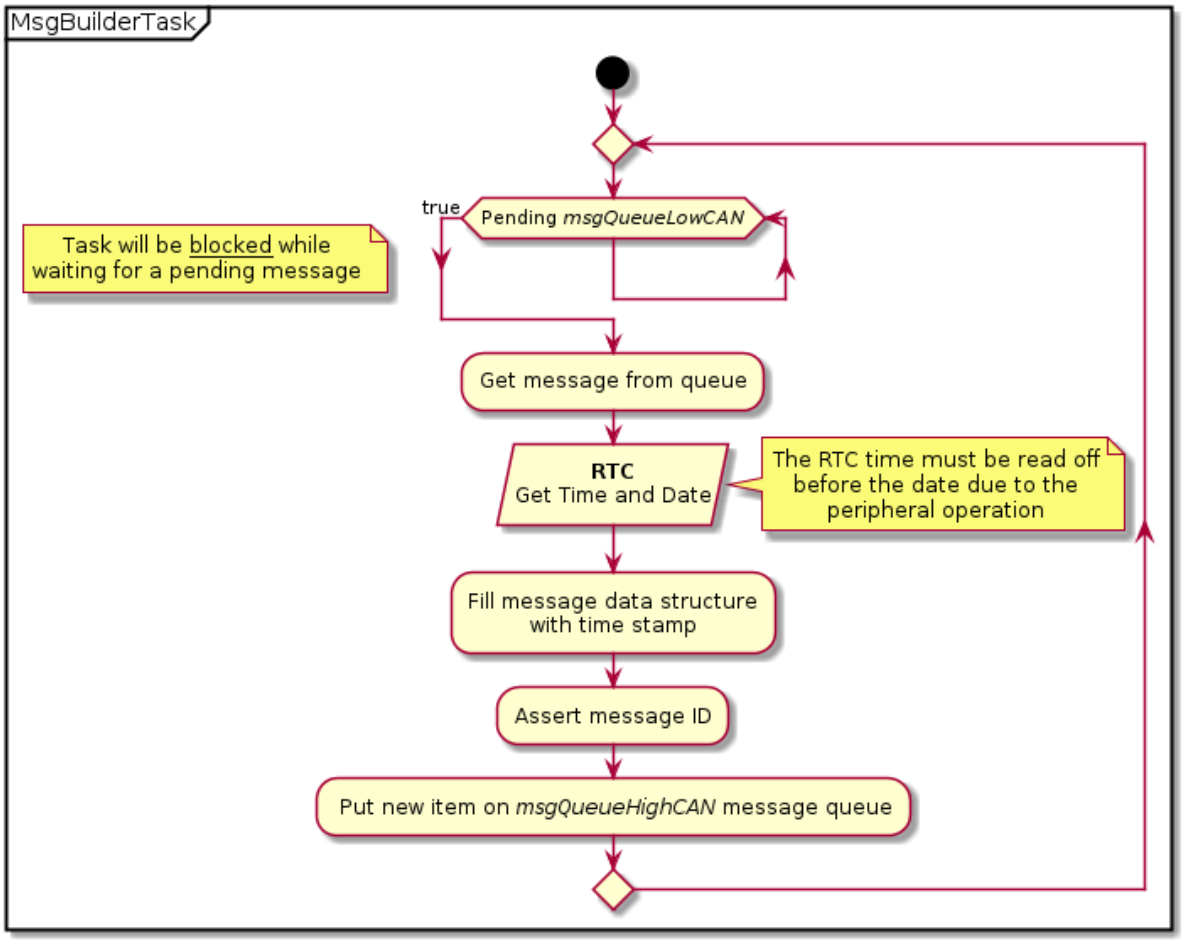
Tabela 16 – Campos presentes na mensagem de aplicação do sistema

Campo	Tipo	Largura [bits]
SeqNumber	unsgined int	32
Date	unsgined int	8
Month	unsgined int	8
Year	unsgined int	8
Hours	unsgined int	8
Minutes	unsgined int	8
Seconds	unsgined int	8
SubSeconds	unsgined int	8
canId	unsgined int	32
RTR	unsgined int	8
IDE	unsgined int	8
DLC	unsgined int	8
CANPayload	Array com 8 posições de unsgined int	64
ErrorCode	unsgined int	16
CRCValue	unsgined int	32

Fonte: Autoria própria.

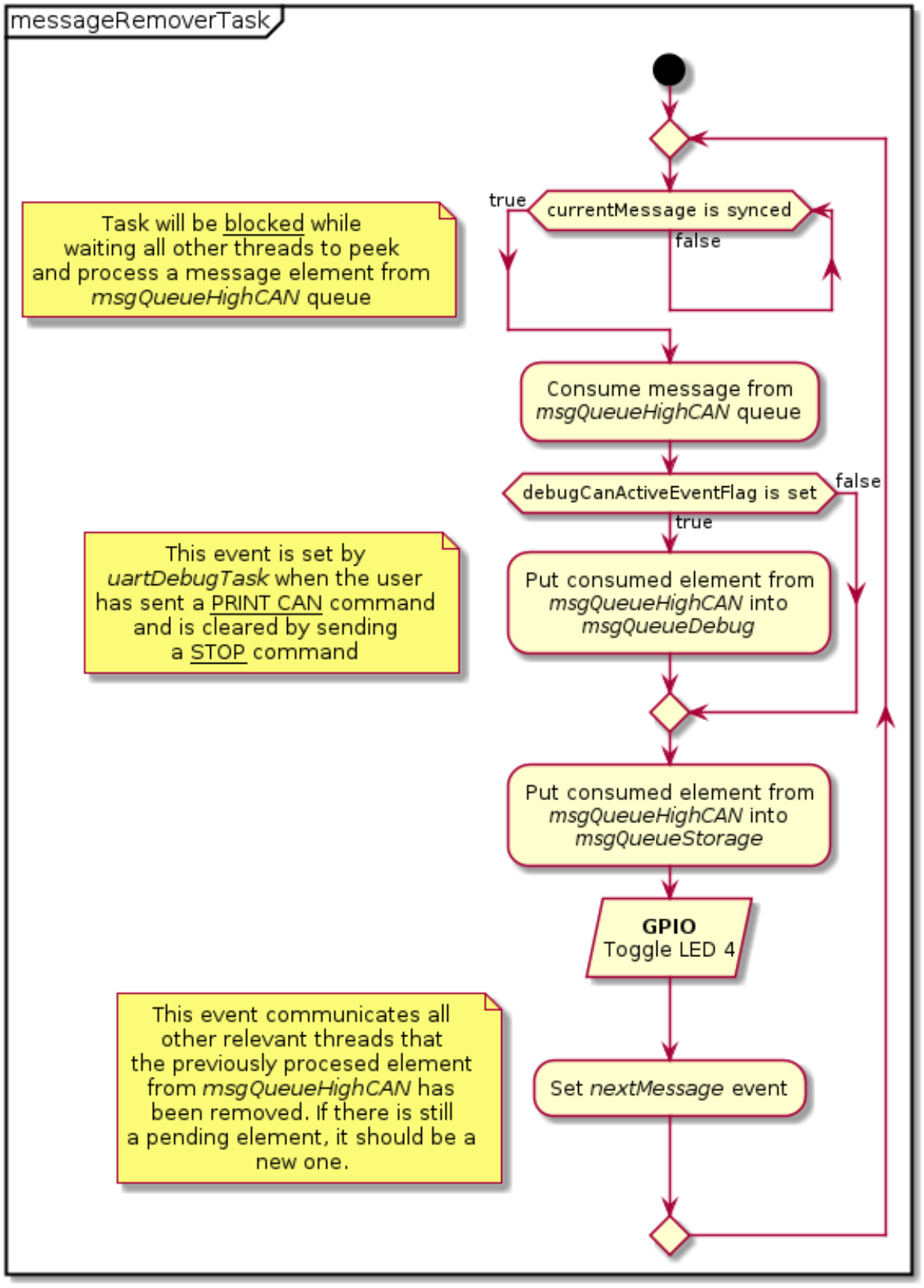
APÊNDICE D – DIAGRAMAS DE ATIVIDADE DO SNIFFERCAN

Figura 48 – Diagrama de atividade da tarefa *Message Builder*.



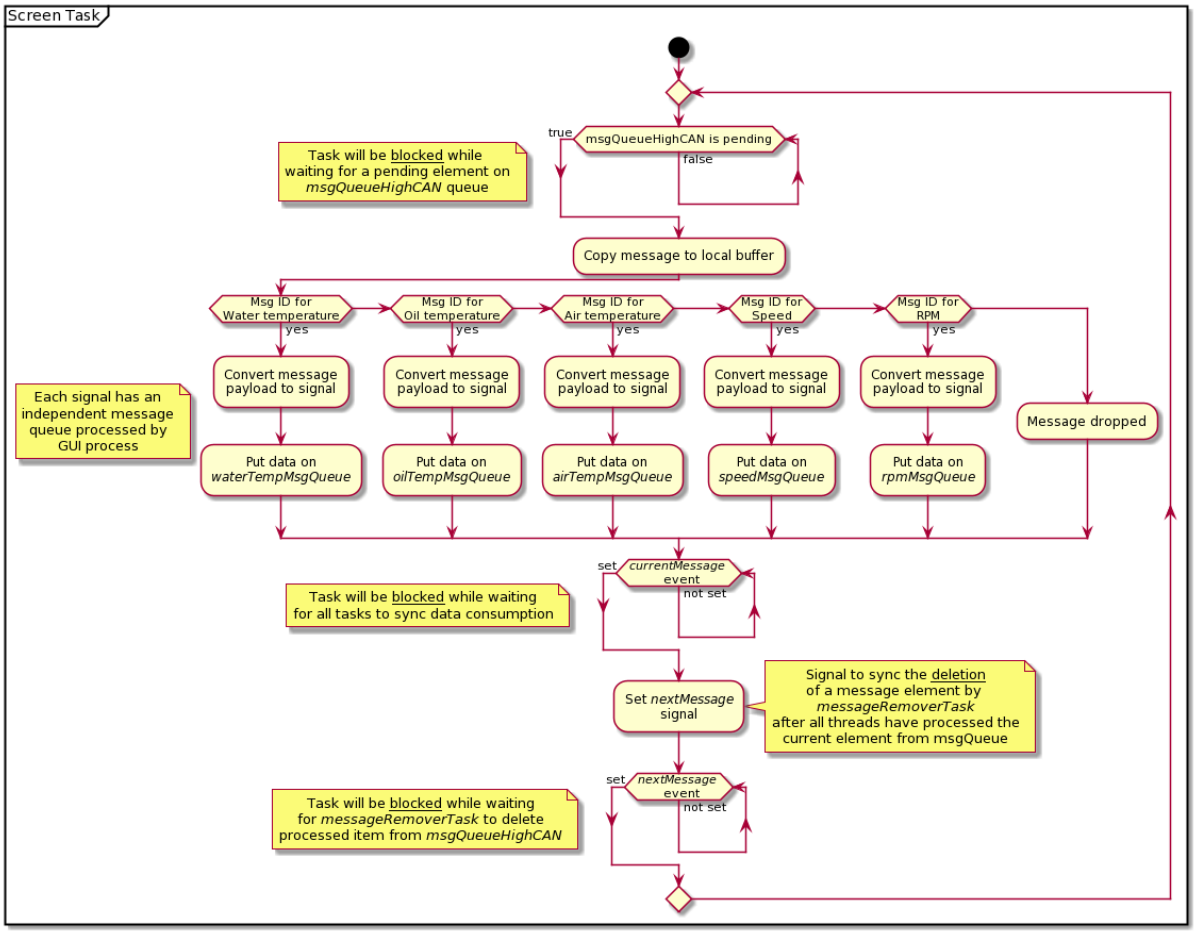
Fonte: Autoria própria.

Figura 49 – Diagrama de atividade da tarefa *Message Remover*.



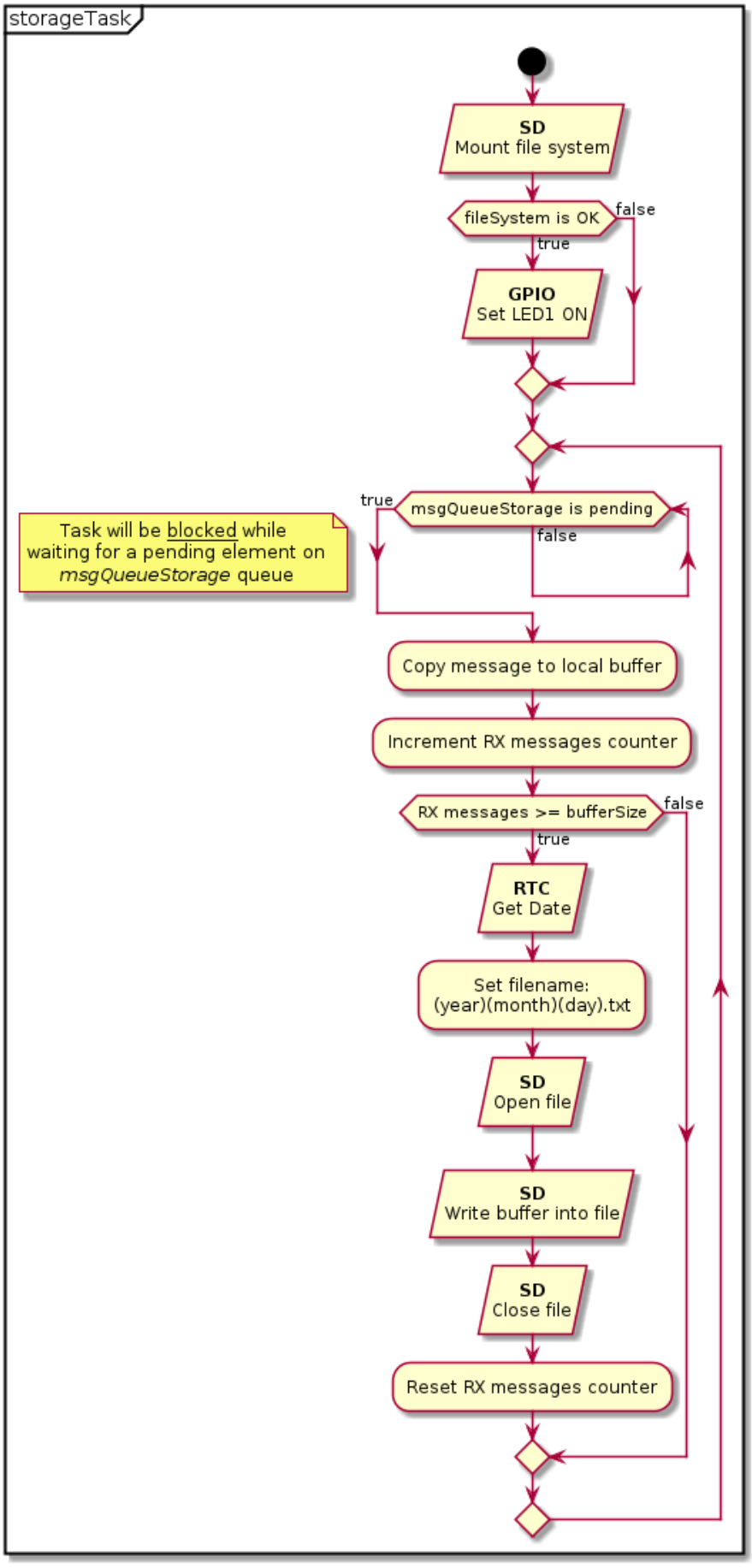
Fonte: Autoria própria.

Figura 50 – Diagrama de atividade da tarefa *Screen*.



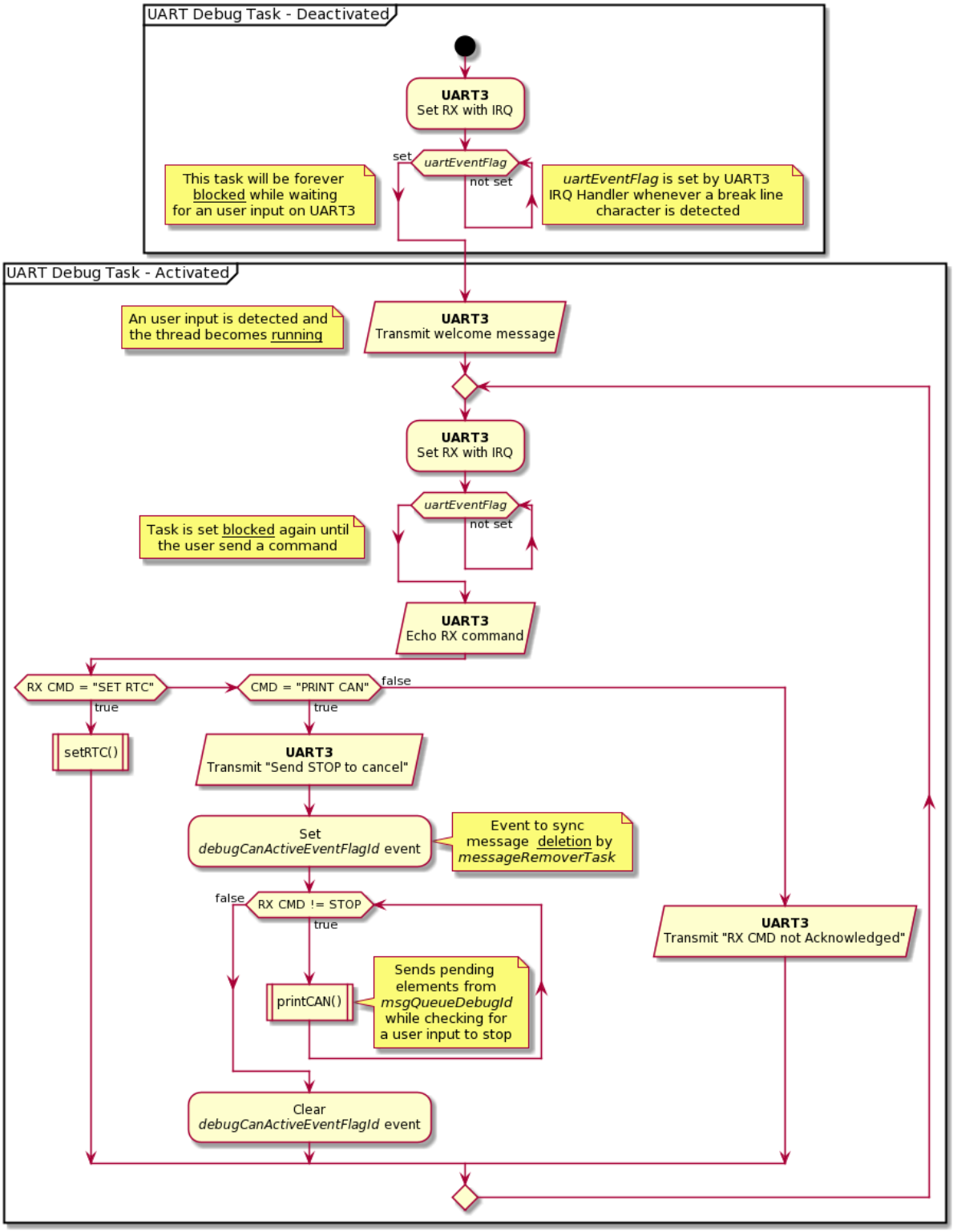
Fonte: Autoria própria.

Figura 51 – Diagrama de atividade da tarefa *Storage*.



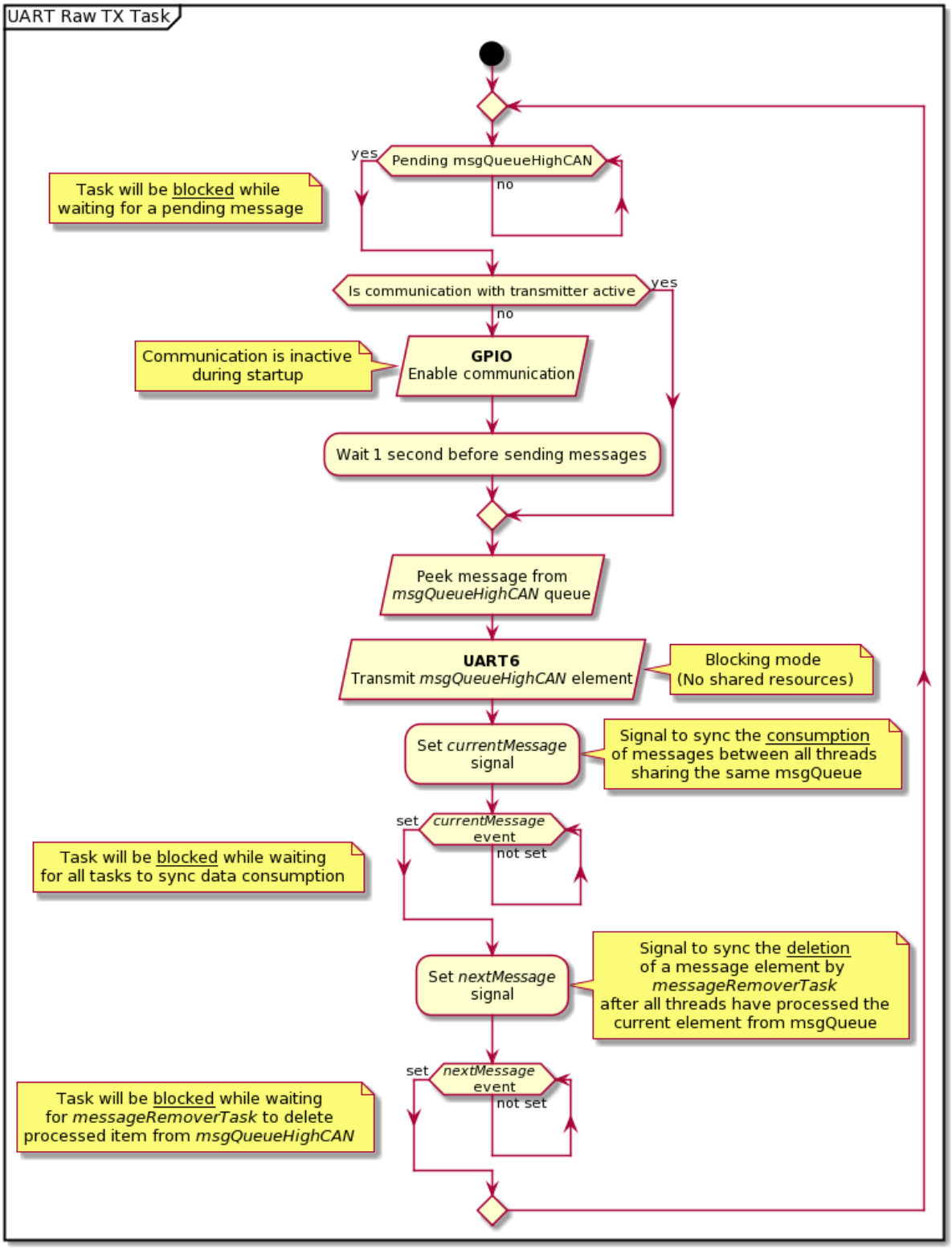
Fonte: Autoria própria.

Figura 52 – Diagrama de atividade da tarefa *UART Debug*.



Fonte: Autoria própria.

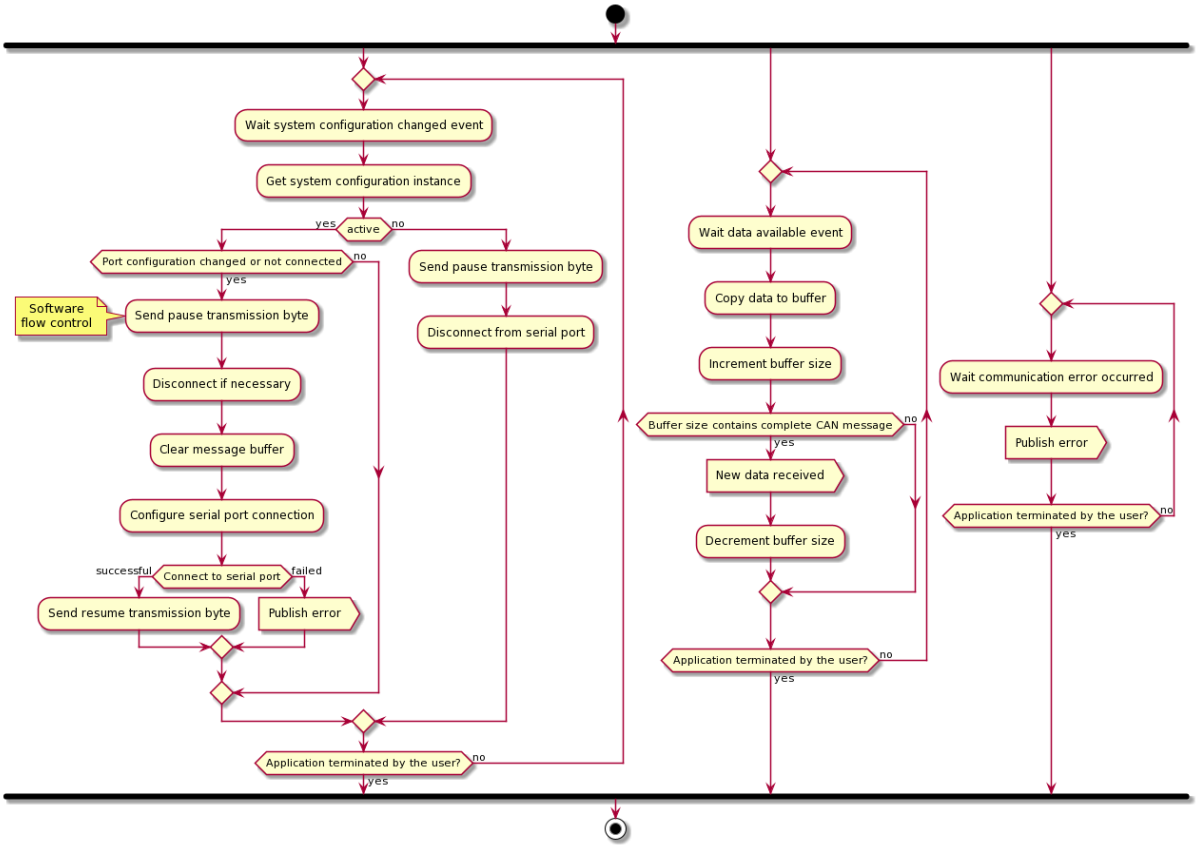
Figura 53 – Diagrama de atividade da tarefa *UART Raw TX*.



Fonte: Autoria própria.

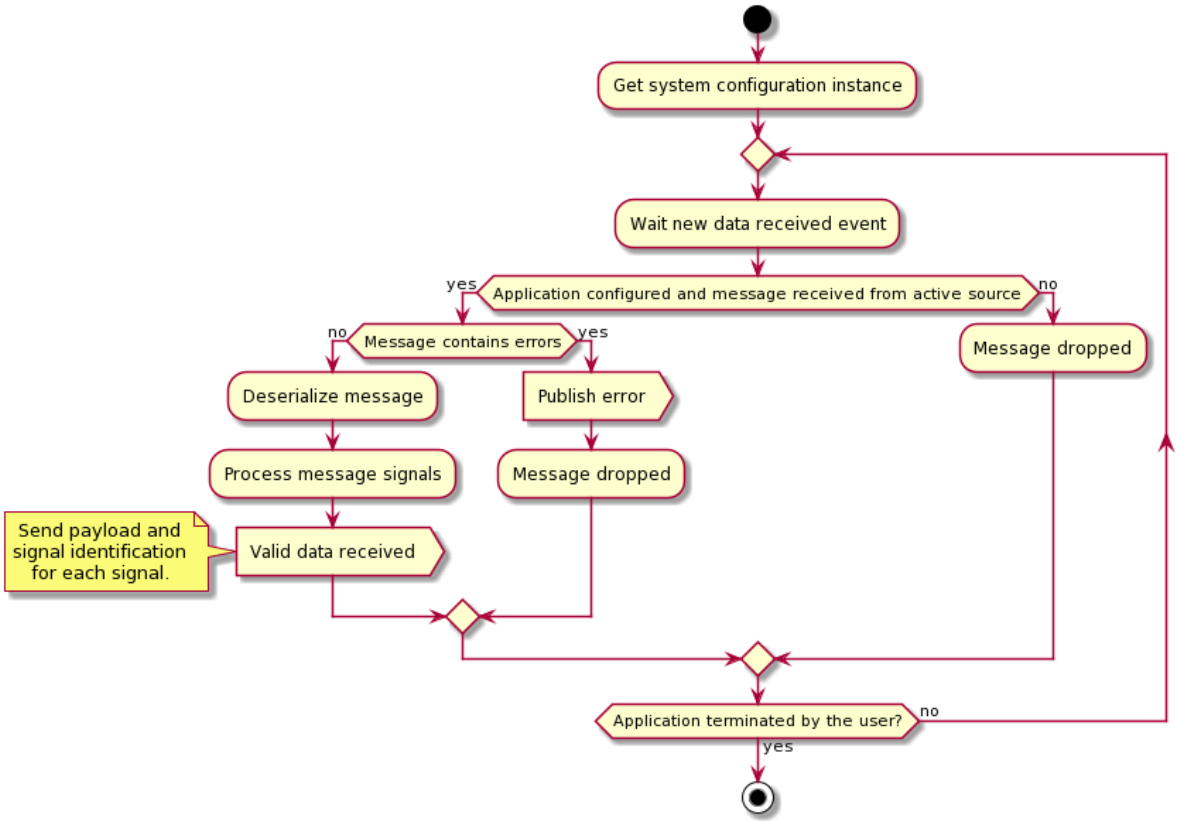
APÊNDICE E – DIAGRAMAS DE ATIVIDADE DA APLICAÇÃO DESKTOP

Figura 54 – Diagrama de atividade do componente *CAN Listener*.



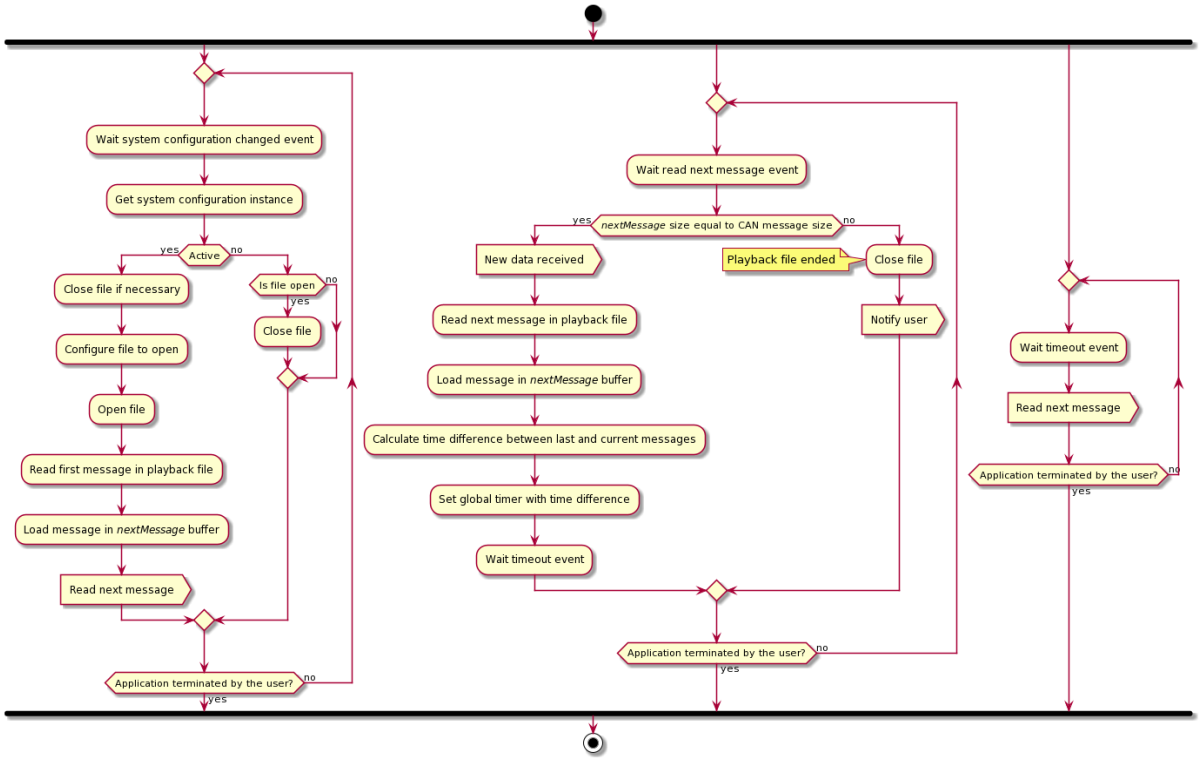
Fonte: Autoria própria.

Figura 55 – Diagrama de atividade do componente *Data Source Manager*.



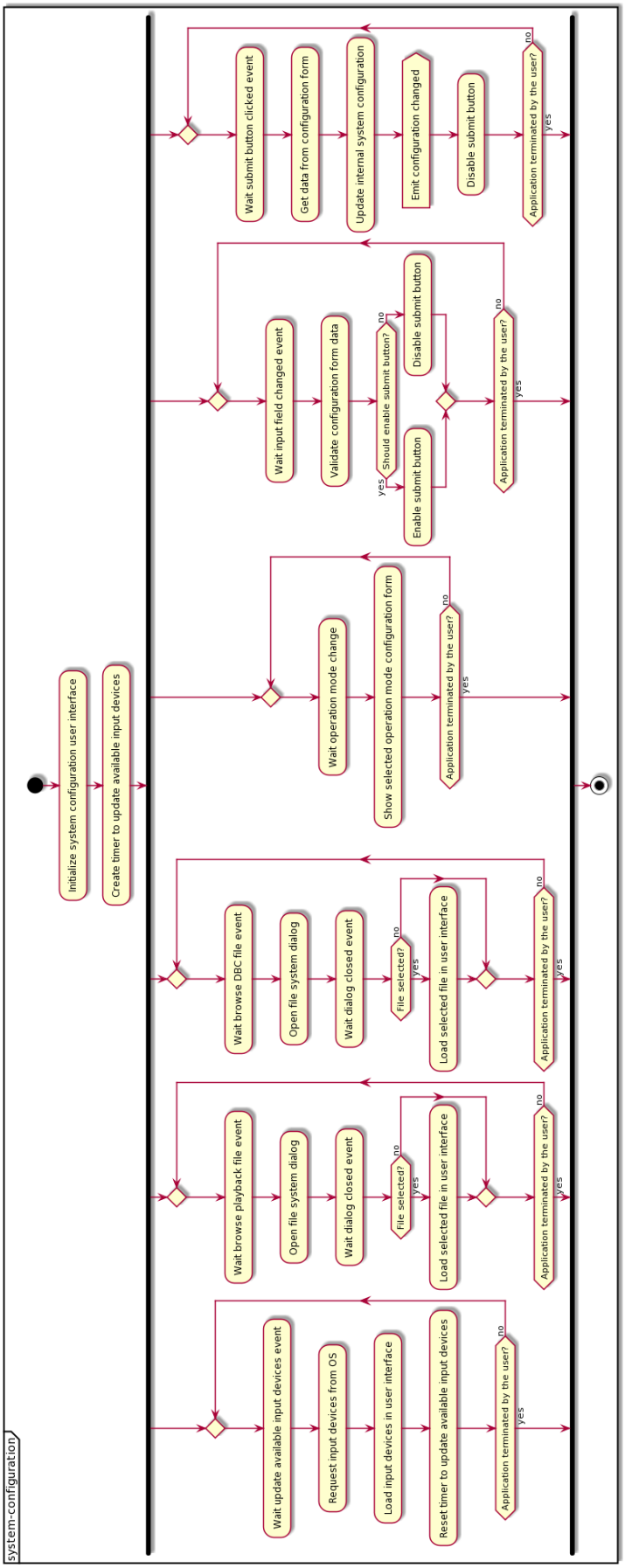
Fonte: Autoria própria.

Figura 56 – Diagrama de atividade do componente *Playback Listener*.



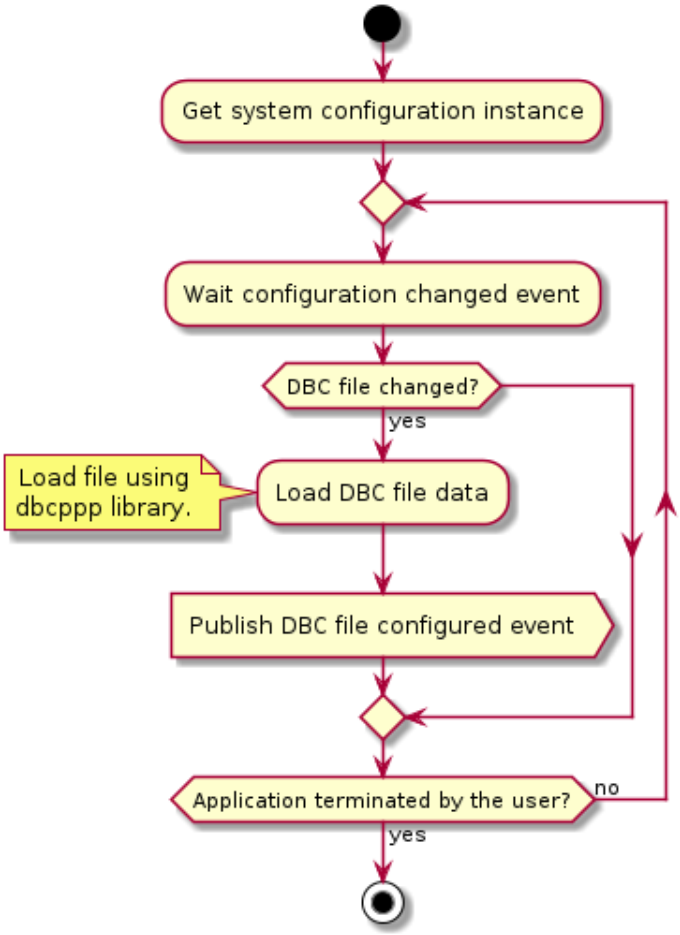
Fonte: Autoria própria.

Figura 57 – Diagrama de atividade do componente *System Configuration*.



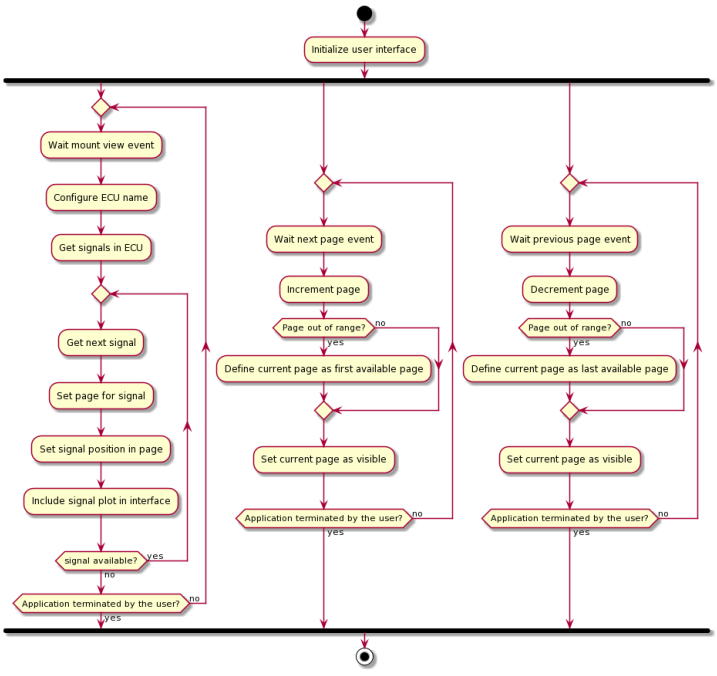
Fonte: Autoria própria.

Figura 58 – Diagrama de atividade do componente *CAN DBC Manager*.



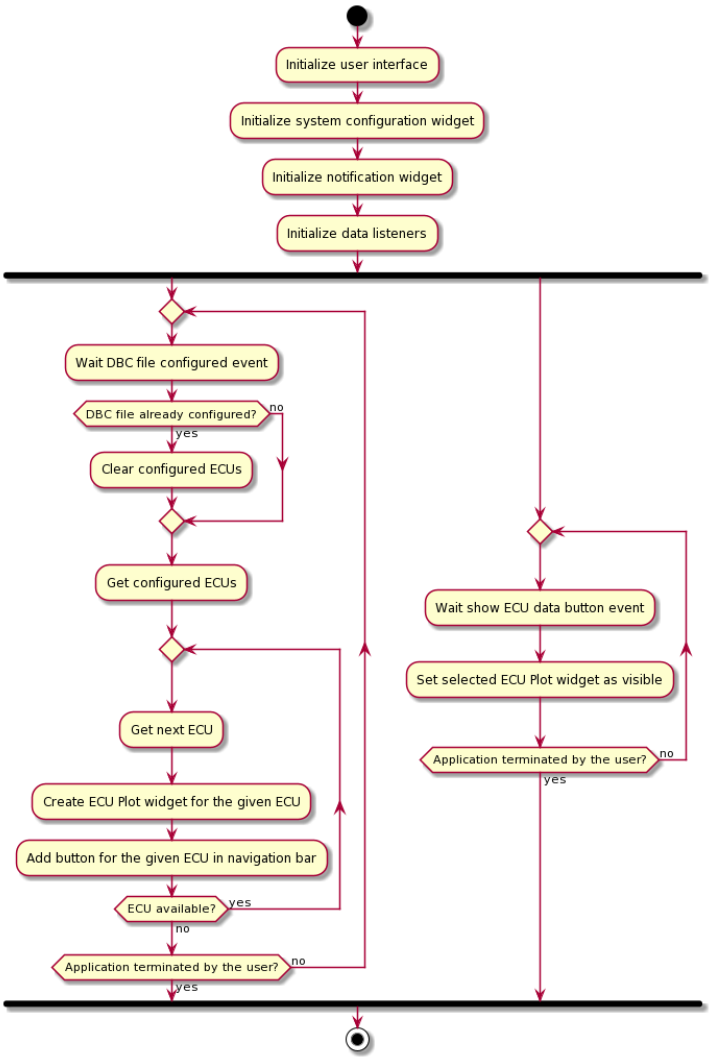
Fonte: Autoria própria.

Figura 59 – Diagrama de atividade do componente *ECU Plot UI*.



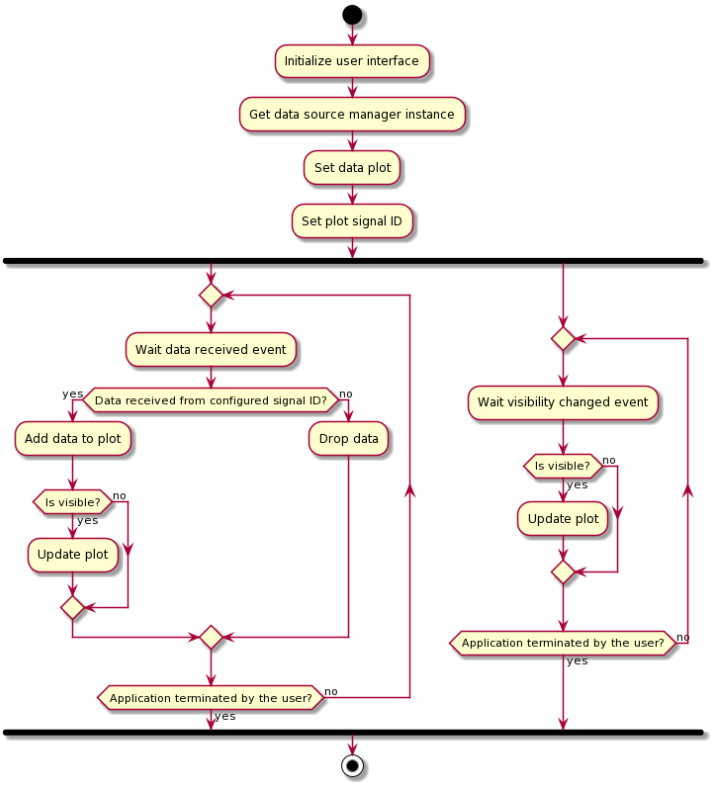
Fonte: Autoria própria.

Figura 60 – Diagrama de atividade do componente *Main Window UI*.



Fonte: Autoria própria.

Figura 61 – Diagrama de atividade do componente *Signal Plot UI*.



Fonte: Autoria própria.

ÍNDICE REMISSIVO