UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA – EST

DERIK ADAN DO NASCIMENTO MAIA

PROTÓTIPO DE BAIXO CUSTO DE ANALISADOR RS-232 SÍNCRONO E PROTOCOLO HDLC PARA LINHAS DE RADIODETERMINAÇÃO

Manaus
2022

DERIK ADAN DO NASCIMENTO MAIA

PROTÓTIPO DE BAIXO CUSTO DE ANALISADOR RS-232 SÍNCRONO E PROTOCOLO HDLC PARA LINHAS DE RADIODETERMINAÇÃO

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Eletrônica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro Eletrônico.

Orientador: Fábio de Sousa Cardoso, Dr.

Manaus

2022

Universidade do Estado do Amazonas – UEA
Escola Superior de Tecnologia - EST

Banca Avaliadora composta por: *Data da defesa: 21/10/2022.*
Prof. Fábio de Sousa Cardoso, Dr *(Orientador)*
Prof. Jozias Parente de Oliveira, Dr
Prof. Daniel Guzmán del Río, Dr

CIP – Catalogação na Publicação

Maia, Derik Adan do Nascimento

Protótipo de baixo custo de analisador RS-232 síncrono e protocolo HDLC para linhas de radiodeterminação / Derik Adan do Nascimento Maia; [orientado por] Fábio de Sousa Cardoso. – Manaus: 2022.
83 p.: il.

Trabalho de Conclusão de Curso (Graduação em Engenharia Eletrônica). Universidade do Estado do Amazonas, 2022.

1. RS-232. 2. HDLC. 3. Bit Stuffing. 4. Microcontroladores. 5. Protocolos de comunicação. 6. Sistemas embarcados. I. Cardoso, Fábio de Sousa.

DERIK ADAN DO NASCIMENTO MAIA

PROTÓTIPO DE BAIXO CUSTO DE ANALISADOR RS-232 SÍNCRONO E PROTOCOLO HDLC PARA LINHAS DE RADIODETERMINAÇÃO

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Eletrônica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro em Eletrônica.

Nota obtida: _____ (_____)

Aprovada em _____/_____/_____.

Área de concentração: Sistemas Embarcados, Protocolos de comunicação.

BANCA EXAMINADORA

_____
Orientador: Fábio de Sousa Cardoso, Dr.

_____
Avaliador: Jozias Parente de Oliveira, Dr.

_____
Avaliador: Daniel Guzmán del Río, Dr.

Manaus 2022

DEDICATÓRIA

A todos aqueles que me proporcionaram a oportunidade de nascer nesse mundo, ser amado, cuidado e educado, especialmente em memória de minha querida mãe, Ednilze Nascimento, e de meu grande amigo e mentor profissional, Jorge Lozano, dedico este trabalho generosamente, como singela forma de gratidão.

AGRADECIMENTOS

Aos meus pais e familiares que sempre encorajaram minhas decisões. A todos os professores que dispuseram de seu tempo, paciência e conhecimento para me orientar no caminho do saber. À minha amada esposa Alyne Maia e ao meu filho Luiz Maia, pela compreensão de minha ausência para realização desta pesquisa. Aos amigos Ricardo Matheus, Rafael Castro, Karolayne Martins, Antônio Pereira, Mário Oliveira e Estevão Assis, sempre solícitos a me apoiar diversas vezes durante a formação acadêmica. Ademais, aos irmãos de farda, pelas instruções, conselhos e exemplos morais no cumprimento do dever com a Pátria.

“Quando você perder, não perca também a lição”.

Dalai Lama

RESUMO

O presente trabalho tem como objetivo o desenvolvimento de um protótipo de dispositivo eletrônico de baixo custo para avaliação de sinais RS-232, a fim de inspecionar indicadores de controle de fluxo, sincronismo e conteúdo de quadros do protocolo HDLC na camada de enlace de dados. A aplicação da proposta visa estimular o desenvolvimento de alternativas para equipamentos de inspeção de enlaces de telecomunicações, por meio de sistemas embarcados, no âmbito nacional. Tal proposta permite contribuir para acelerar a inspeção e a manutenção de linhas de radiodeterminação no contexto do Sistema de Controle do Espaço Aéreo Brasileiro, de modo a proporcionar menor tempo de indisponibilidade dos sensores de vigilância radar. Isto otimizaria a acurácia na projeção da posição de aeronaves em evolução no espaço aéreo, e, por sua vez, favoreceria menor risco de incidentes aéreos, que, quando ocorrem, podem trazer prejuízos financeiros e perda de vidas humanas. Primeiramente é apresentada a fundamentação teórica através da revisão da literatura, que conceitua as principais tecnologias de telecomunicações empregadas nos enlaces envolvidos, além de teorias e ferramentas usadas no projeto de sistemas embarcados. Em seguida, são apresentadas as etapas e os materiais necessários à construção de uma placa de circuito impresso responsável por equalizar os níveis de tensão utilizados em RS-232 e TTL, utilizando componentes discretos e de fácil aquisição no mercado nacional. Posteriormente, demonstra-se a aplicação de uma técnica de detecção e de remoção de bits por software, além de classificação, agrupamento e envio dos dados recebidos para um dispositivo móvel através de um transceptor *bluetooth*. Como resultado do trabalho, são apresentados os testes e os dados coletados na validação do protótipo em relação à equalização dos níveis de tensão e da faixa de operação dos sinais de sincronismo. Também são apresentados dados comparativos entre os obtidos pelas ferramentas disponíveis localmente e os do protótipo desenvolvido. Por fim, analisam-se os dados obtidos para validação da hipótese sobre a possibilidade de desenvolver um dispositivo que atenda aos requisitos de inspeção de sinais da camada de enlace de dados em linhas de radiodeterminação.

Palavras-chaves: RS-232. HDLC. Bit Stuffing. Microcontroladores. Protocolos de comunicação. Sistemas embarcados.

ABSTRACT

This work aims to develop a prototype of a low-cost electronic device for evaluating RS-232 signals, in order to inspect indicators of flow control, synchronization and content from HDLC protocol frames in the data link layer. The proposed application seeks to stimulate the development of alternatives for inspection equipments of telecommunications links, through embedded systems at the national level. This proposal contributes to speeding up the inspection and maintenance of radiodetermination lines in the context of the Brazilian Airspace Control System, in order to provide less downtime of radar surveillance sensors. It would optimize the accuracy in projecting the position of evolving aircraft in the airspace, and, in turn, would contribute to a lower risk of aircraft incidents, which, when they occur, can cause financial losses and loss of human lives. First, the theoretical foundation is presented with a literature review, which conceptualizes the main telecommunications technologies used in the links involved, as well as theories and tools used in the design of embedded systems. Next, the steps and materials necessary for the construction of a printed circuit board responsible for equalizing the voltage levels used in RS-232 and TTL are presented, using discrete components that are easily acquired in the national market. Subsequently, an application for the detection and the removal of bits by software is demonstrated, as well as classification, grouping and sending of these received data to a mobile device through a bluetooth transceiver. For the work's results, the tests and data collected in the validation of the prototype are presented in relation to the equalization of voltage levels and the operating range of the synchronism signals. Comparative data between those obtained by the locally available tools and the developed prototype are also presented. Finally, the data obtained are analyzed to validate the hypothesis about the possibility of developing a device that meets the requirements for inspecting signals from the data link layer on radiodetermination lines.

Keywords: RS-232. HDLC. Bit Stuffing. Microcontrollers. Communication protocols. Embedded systems.

LISTA DE FIGURAS

LISTA DE SIGLAS

ASTERIX	All Purpose Structured Eurocontrol Surveillance Information Exchange
ASIC	Application-specific integrated circuit
CAN	Contoller Area Network
CINDACTA IV	Quarto Centro Integrado de Defesa Aérea e Controle de Tráfego Aéreo
CRC	Cyclic Redundancy Check
CTS	Clear to Send
DSU	Unidade de Serviço de Dados
ECD	Equipamento de Comunicação de Dados
EIA	Electronic Industries Association
ETC	External Transmitter Clock
ETD	Equipamento Terminal de Dados
FCS	Frame Check Sequence
FRN	Field Reference Number
GPIO	General Purpose Input/Output
HDLC	High-Level Data Link Control
LAPB	Link Access Procedure Balanced
LLC	Controle de *Link* Lógico
LSB	Least Significant Bit
MAC	Controle de Acesso ao Meio
MSB	Most Significant Bit
OSI	Open Systems Interconnection
PCB	Printed Circuit Board
PDU	Protocol Data Unit
RC	Receiver Clock
RTS	Request to Send
SDU	Service Data Unit
SISCEAB	Sistema de Controle do Espaço Aéreo Brasileiro
TBJ	Transistor Bipolar de Junção
TC	Transmitter Clock
TTL	Transistor-Transistor Logic
USART	Universal Synchronous Asynchronous Receiver Transmitter
WAN	Wide Area Network

SUMÁRIO

INTRODUÇÃO

O controle e a defesa de tráfego aéreo brasileiro dependem de diversos sensores espalhados por todo território nacional. Muitos desses sensores são radares fixos terrestres que enviam seus dados por canalizações de enlaces satelitais.

Somente no contexto do Quarto Centro Integrado de Defesa Aérea e Controle de Tráfego Aéreo (CINDACTA IV), as canalizações de voz e dados, que servem de suporte à prestação do serviço de controle de tráfego aéreo e de defesa aérea na região amazônica, compreendem aproximadamente 60% da área territorial brasileira. Essas canalizações têm como objetivo interligar os equipamentos de telecomunicações instalados nos sítios remotos ao CINDACTA IV (BRASIL, 2019).

Ao receber essas canalizações, as interfaces de dados para processamento obedecem a determinados protocolos de comunicação nas camadas físicas e de enlace de dados. Dispositivos para avaliar os sinais de controle de nível lógico, bem como analisadores de protocolo, são de difícil aquisição devido à sua obsolescência, alto preço ou escassez no mercado nacional.

Esta pesquisa visa testar a possibilidade de criar um dispositivo eletrônico para captura de sinais padrão RS-232 síncrono, portátil e de baixo custo, a fim de inspecionar indicadores de controle de fluxo de dados, leitura e decomposição de quadros da camada de enlace de dados. Em virtude disso, faz-se necessário projetar e montar um protótipo capaz de receber como entrada sinais coletados em campo no padrão EIA RS-232 síncrono, através de conector DB-25. Em seguida, processar quadros do protocolo *High-Level Data Link Control* (HDLC) e, por intermédio de conexão sem fio, apresentar como saídas as informações de sinais de controle de fluxo, integridade e conteúdo de quadros em interface gráfica para aplicativo móvel.

Tal proposta permite contribuir para acelerar a inspeção e a manutenção de linhas de radiodeterminação no contexto do Sistema de Controle do Espaço Aéreo Brasileiro (SISCEAB), de modo a proporcionar menor tempo de indisponibilidade dos sensores de vigilância radar. Isto otimizaria a acurácia na projeção da posição de aeronaves em evolução no espaço aéreo, e, por sua vez, favoreceria menor risco de incidentes aéreos, que, quando ocorrem, podem trazer prejuízos financeiros e perda de vidas humanas.

Para implementação da pesquisa, serão utilizados conhecimentos adquiridos no decorrer acadêmico do curso de Engenharia Eletrônica, principalmente nas disciplinas de Construção Eletrônica, Eletrônica Aplicada, Sistemas Operacionais Embarcados e de Tempo Real, Redes sem Fio e Microcontroladores.

Para o desenvolvimento do protótipo, será feita uma revisão teórica nas áreas de camadas física e de enlace de dados, projeto de circuitos eletrônicos para acoplar as entradas do microcontrolador à interface serial RS-232, desenvolvimento de programas para microcontroladores STM32, comunicação de redes de área pessoal tipo *bluetooth* e interfaces para dispositivos móveis *Android*.

Este trabalho está dividido em quatro capítulos, os quais são: referencial teórico, materiais e métodos, implementação do projeto e análise dos resultados obtidos

O primeiro capítulo aborda o referencial teórico, onde são tratados os assuntos supracitados que estão relacionados ao projeto.

No segundo capítulo, são mostrados os materiais utilizados para a implementação da pesquisa, bem como o método proposto. Serão apresentadas as etapas seguidas na implementação do protótipo, de forma sequencial, começando pela pesquisa e aprofundamento teórico nos assuntos referenciados, seguido do projeto de leiaute e simulação da placa de circuito impresso que serve de base ao protótipo, ensaios de validação da placa de circuito impresso produzida, desenvolvimento e integração do programa embarcado projetado e, por fim, testes de validação final do protótipo como um todo.

No terceiro capítulo, é apresentada a descrição detalhada do protótipo que foi implementado. Primeiramente, serão apresentados os módulos que constituem o protótipo, os quais são: condicionamento dos sinais elétricos da camada física, processamento de quadros da camada de enlace de dados e transferência de dados processados através de interface *bluetooth* para apresentação em aplicativo móvel. São descritas as ferramentas e tecnologias utilizadas para a implementação de cada módulo que constitui este protótipo. Para isso, serão descritos os procedimentos e ambientes de desenvolvimento para desenho e simulação de esquemático de circuitos eletrônicos, geração de placas de circuito impresso, programação de *firmware* para microcontrolador STM32, programação de aplicativos para dispositivos móveis e, por fim, a validação do protótipo de sistema.

No quarto capítulo, são apresentados os resultados obtidos com a implementação do trabalho proposto. Serão comentadas as análises dos resultados baseadas no referencial teórico e no conhecimento adquirido durante todo o processo de pesquisa e implementação do projeto.

Por fim, na conclusão, será retomado o que foi proposto como hipótese em comparação com os resultados obtidos, de modo a atestar a aplicabilidade da solução proposta. Serão relatados os desafios encontrados no decorrer da implementação e sugestões para trabalhos futuros.

1 REFERENCIAL TEÓRICO

Diante do esclarecimento da formulação do problema, da hipótese a ser verificada e dos objetivos estabelecidos na introdução, faz-se necessário, nesse momento, definir diante do conhecimento científico as características das tecnologias que operam de fundo à proposta apresentada. Para isso, este capítulo explora, sem esgotar, os referenciais teóricos dos assuntos que orbitam o tema.

Isso se deve ao fato de ser indispensável o conhecimento teórico para emprego correto e eficiente dos meios que contribuirão para a validação da hipótese. Neste caso, são abordadas as tecnologias e especificações de operação de enlace físico e lógico, processamento de dados por microcontroladores e de interface com dispositivos móveis para propagação das informações tratadas.

1.1 PADRÃO EIA RS-232

No início da década de 1960, o comitê de padrões *Electronic Industries Association* (EIA) desenvolveu um padrão de interface para equipamentos de comunicação de dados, o qual no ano de 1969, foi definido como padrão RS-232C. Naquela época, as comunicações de dados significavam apenas a troca de dados digitais entre um computador *mainframe* localizado centralmente e um terminal de computador remoto, ou possivelmente entre dois terminais sem um computador envolvido. Esses dispositivos eram conectados por linhas telefônicas de voz e, consequentemente, exigiam um modem em cada extremidade para a tradução do sinal. Embora simples no conceito, as muitas possibilidades de erro de dados que ocorrem durante uma transmissão de dados por meio de um canal analógico exigem um projeto relativamente complexo. Desta maneira, um padrão era necessário para garantir uma comunicação confiável e, em segundo lugar, permitir a interconexão de equipamentos produzidos por diferentes fabricantes, promovendo assim os benefícios da produção em massa e da concorrência. Uma das versões mais usadas mundialmente, EIA RS-232E, foi introduzida em 1991, a qual especifica tensões de operação, temporizações e funções de sinais, além de um protocolo para troca de informações e conectores mecânicos (SALCIUNAS, 1991).

Sendo assim, o padrão EIA RS-232 é um conjunto de especificações que define as funções de interface de circuito e suas atribuições de pinos dos conectores correspondentes para transferência de dados entre o Equipamento Terminal de Dados (ETD ou, do original inglês, DTE) e o Equipamento de Comunicação de Dados (ECD ou, do original inglês, DCE).

Operações *full-duplex* ou *half-duplex* são suportadas para modos de transmissões síncronas, que exigem um sinal de *clock* fornecido por uma das linhas da interface ou transmissões assíncronas em velocidades de até 20 kbps (SALCIUNAS, 1991).

Em resumo, o padrão define características elétricas como níveis de tensão, taxa de sinalização, comportamento de curto-circuito e carga máxima da capacitância. Determina também características mecânicas da interface, conectores e identificação dos pinos, além de funções de cada circuito no conector da interface. Estabelece ainda subconjuntos padrões de circuitos de interface para aplicações de telecomunicação.

Entretanto, o padrão não define codificação, enquadramento dos caracteres no fluxo de dados (bits por caractere, bits de início e parada, paridade), protocolos para detecção de erros, algoritmos para compressão de dados, taxas de bit para transmissão (apesar de limitar o padrão a taxas de até 20 kbits por segundo) ou o fornecimento de energia para dispositivos externos.

Quando terminais eletrônicos começaram a ser usados, eram projetados para serem intercambiáveis com aparelhos teletipo. A terceira revisão deste padrão (chamada de RS-232C), publicada em 1969, foi criada em parte para adequar-se às características elétricas destes dispositivos e também suportavam RS-232. Este padrão foi originalmente usado para conectar um teletipo (equipamento eletromecânico de comunicação assíncrona) a um modem. Deste modo, foi utilizado em diversos tipos de comunicação remota, especialmente por modems. Posteriormente, microcomputadores pessoais e outros equipamentos começaram a utilizar este padrão para comunicação com equipamentos já existentes. Quando a empresa IBM lançou microcomputadores com uma porta RS-232, esta interface tornou-se realmente onipresente. Por muitos anos o padrão para comunicação serial em quase todos os microcomputadores era algum tipo de porta RS-232 e continuou sendo utilizado em grande escala até o fim dos anos 90. Durante esse tempo, essa foi a maneira padrão para a conexão de modems (STRANGIO, 1997).

Sobre as aplicações desse padrão, Fey e Gauer (2020, p. 193) dizem que “a RS-232 é usada tanto para transferência de dados em circuitos assíncronos como também para enlaces síncronos tais como o SDLC, HDLC, *Frame Relay* e o X.25”.

1.1.1 Pinagem, conectores e sinais

O padrão RS-232 define os níveis de tensão que correspondem aos níveis lógicos zero e um para a transmissão de dados e as linhas de sinal de controle. Os sinais válidos situam-se na faixa de -15 a -3 e +3 a +15 volts. A faixa de -3 a +3 volts não é um nível RS-232 válido. O padrão especifica uma tensão máxima de circuito aberto de 25 volts. Os drivers e receptores

RS-232 devem ser capazes de resistir a curto-circuito indefinido com o aterramento ou a qualquer nível de tensão de até ± 25 volts (GRODZIK, [2012]).

O ETD possui um conector DB-25 macho e utiliza 22 dos 25 pinos disponíveis para sinais e terra. Já o equipamento na extremidade próxima da conexão (a interface da linha telefônica), chamado de dispositivo ECD (geralmente um modem), possui um conector DB-25 fêmea e utiliza os mesmos 22 pinos disponíveis para sinais e terra. O cabo que liga os dispositivos ETD e ECD é um cabo paralelo direto sem cruzamentos ou auto conexões nos conectores.

Muitas das 22 linhas de RS-232 pertencem a conexões em que o dispositivo ECD é um modem e são usadas somente quando o protocolo de software as emprega. Para qualquer dispositivo ECD que não seja um modem, ou quando dois dispositivos ETD estão diretamente ligados, são necessárias poucas linhas de sinal. Além do fato de muitas das linhas serem para canais secundários projetados, que são usados apenas em condições muito específicas. Esses canais secundários fornecem o gerenciamento do modem remoto, permitindo alterar as taxas de transmissão em tempo real, solicitar a retransmissão se um erro de paridade for detectado e outras funções de controle. Tais canais secundários, quando usados, são normalmente configurados para operar em taxas de transmissão muito baixas em comparação com o canal primário, de modo a garantir confiabilidade no caminho de controle.

Os sinais de temporização do transmissor e do receptor (pinos 15, 17 e 24) são usados apenas para um protocolo de transmissão síncrona. Para o protocolo assíncrono padrão de 8 bits, os sinais de temporização externos são desnecessários.

Nomes de sinais que implicam uma direção, como *Transmit Data* e *Receive Data*, são nomeados do ponto de vista do dispositivo ETD. Se o padrão RS-232 fosse seguido à risca, esses sinais também teriam o mesmo nome para o mesmo número do pino no lado do ECD. No entanto, isso não é feito na prática pela maioria dos engenheiros, provavelmente pela falta de convenção de qual dispositivo será ETD ou ECD. Como resultado, os nomes dos sinais sensíveis à direção são alterados no lado do ECD para refletir sua direção de acionamento no mesmo (STRANGIO, 1997).

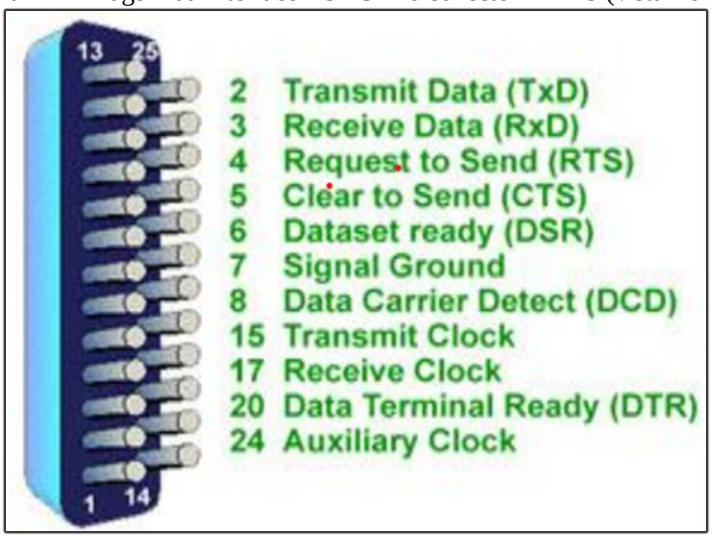
As funções dos sinais RS-232 podem ser subdivididas em seis categorias:

a) terra do sinal e blindagem;
b) canais de comunicação primária: intercâmbio de dados e sinais de controle de fluxo;
c) canais de comunicação secundária: controle do modem remoto, solicitações de retransmissão em caso de erros e controle sobre a configuração do canal primário;

d) *status* do modem e sinais de controle: *status* do modem e pontos de verificação intermediários à medida que o canal de voz do telefone é estabelecido;
e) sinais de temporização: se um protocolo síncrono for usado, esses sinais fornecem informações de tempo para o transmissor e o receptor, que podem operar em diferentes taxas de transmissão; e
f) sinais de teste de canal: antes que os dados sejam trocados, o canal pode ser testado quanto à sua integridade e a taxa de transmissão pode ser ajustada automaticamente para a taxa máxima que o canal pode suportar.

Nesta pesquisa, que objetiva a captura de dados em linhas de radiodeterminação, são usados apenas as categorias de terra e sinais de blindagem, canais de comunicação primária, *status* do modem e sinais de temporização. A Figura 1 apresenta a pinagem dos sinais utilizados para enlace de dados síncrono com controle de fluxo por *hardware* e até três configurações de sinais de temporização, frequentemente usados para esse tipo de aplicação, através do conector DB-25. Como mencionado anteriormente, os números são destacados a partir do ponto de vista do dispositivo ETD. O Quadro 1 mostra a correlação entre pinagem e funções dos circuitos mais utilizados em comunicações síncronas.

Figura 1 – Pinagem da interface RS-232 no conector DB-25 (vista frontal).



Fonte: (FEY; GAUER, 2020, p. 194).

Quadro 1 – Correlação entre pinagem, função do circuito e sinais estabelecidos no padrão RS-232 mais utilizado.

Número do pino	Função	Sinal de dados de		Sinais de controle de		Sinal de temporização de	
		ECD	ETD	ECD	ETD	ECD	ETD
1	Protective ground / shield						
2	Transmitted Data (TxD)		X				
3	Received Data (RxD)	X					
4	Request to Send (RTS)				X		
5	Clear to Send (CTS)			X			
6	DCE Ready (DSR)			X			
7	Signal ground / Common Return						
8	Received Line Signal Detector (CD)			X			
9	Reserved for data set testing						
10	Reserved for data set testing						
11	Unassigned						
12	Secondary Received Line Signal Detector (SCD)			X			
13	Secondary Clear to Send (SCTS)			X			
14	Secondary Transmitted Data (STxD)		X				
15	Transmitter Signal Element Timing (TC)					X	
16	Secondary Received Data (SRxD)	X					
17	Receiver Signal Element Timing (RC)					X	
18	Local Loopback (LL)						
19	Secondary Request to Send (SRTS)				X		
20	DTE Ready (DTR)				X		
21	Remote Loopback (RL)			X			
22	Ring Indicator (RI)			X			
23	Data Signal Rate Selector			X	X		
24	Transmitter Signal Element Timing (ETC)						X
25	Test Mode (TM)			X			

Fonte: adaptado de (SALCIUNAS, 1991).

1.1.2 Especificações elétricas

Um circuito de intercâmbio é definido como um circuito que permite a troca de dados entre o ETD e o ECD. O RS-232 descreve quatro categorias de circuitos de intercâmbio que se aplicam geralmente para todos os sistemas. Eles são circuitos de aterramento ou retorno comum, dados, controle e temporização. Vinte e dois circuitos são especificados por funções possíveis e três não estão atribuídos (SALCIUNAS, 1991, p. 2).

O padrão RS-232 inclui uma referência de aterramento comum no pino 7, frequentemente ligado ao pino 1, e uma blindagem circular que envolve todos os condutores do cabo. As tensões de dados, temporização e sinal de controle são medidas em relação a esse terra

comum. O RS-232 não pode ser usado em aplicações onde os equipamentos nas extremidades opostas da conexão devam ser isolados eletricamente.

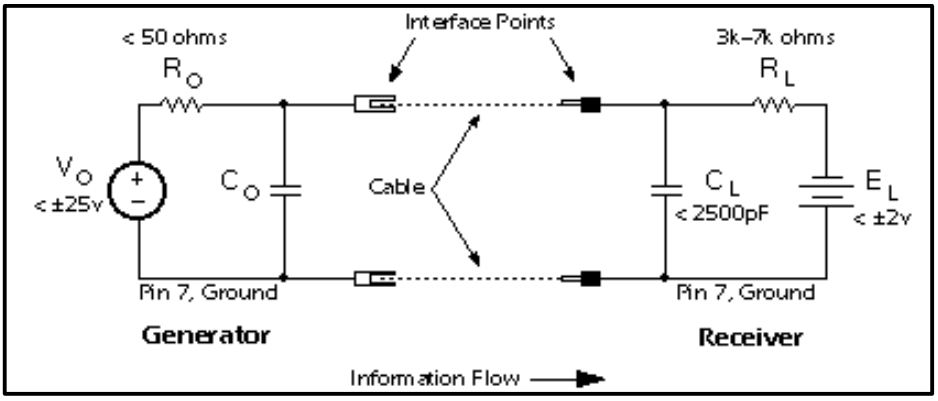
Os dados transmitidos são representados pela condição de marca (*mark*) para binário um (‘1’ lógico) e a condição de espaço (*space*) para binário zero (‘0’ lógico). Um sinal de dados em um circuito de intercâmbio está na condição de marca quando a tensão é mais negativa do que -3 volts com em relação ao sinal de terra. Quando a tensão do sinal de dados é mais positiva que +3 volts em relação ao pino de terra, o sinal de dados está na condição de espaço. A área entre -3 e +3 volts é a região de transição. Dentro da região de transição, o estado do sinal não é definido. Em circuitos de troca de tempo ou controle, o sinal é considerado desligado quando a tensão no ponto de interface é mais negativa do que - 3 volts com respeito ao pino de terra. É considerado ligado se esta tensão é mais positiva do que + 3 volts. Não há funções definidas para tensões na região de transição entre -3 e +3 volts.

A tensão do driver de circuito aberto V_O, com respeito ao pino de terra, não deve exceder 25 volts. O potencial no ponto de interface não deve ser menor que 5 volts nem maior que 15 volts em magnitude quando a resistência do terminador R_L está entre 3 kΩ e 7 kΩ, e a tensão do terminador em aberto E_L é zero. Além disso, a capacitância de derivação efetiva C_L, associada ao terminador, não deve exceder 2500 pF no ponto de interface (SALCIUNAS, 1991, p. 7).

Há ainda algumas características determinadas pelo padrão, como temporizações de transição de nível dos sinais, no entanto, para construção do módulo de equalização dos sinais elétricos da camada física para *Transistor-Transistor Logic* (TTL), não há necessidade de preocupar-se com eles, dado que já são controlados pelos dispositivos ECD.

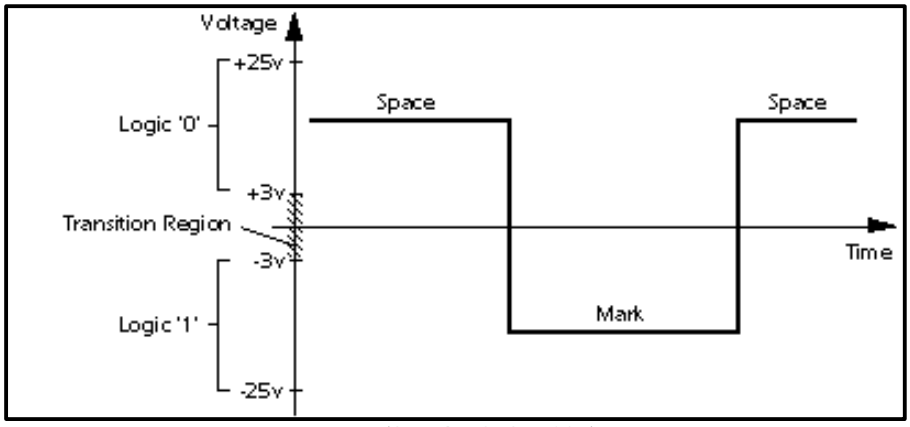
Todos os sinais de intercâmbio que entram na transição devem prosseguir para o estado de sinal oposto. A direção da tensão não deve mudar enquanto na região de transição. O tempo necessário para um sinal de controle passar através da região de transição não deve exceder um milissegundo. O tempo necessário para um sinal de dados ou de temporização passar pela região de transição não deve exceder um milissegundo ou 4 por cento da duração normal de um elemento de sinal nesse circuito, o que for menor. A taxa instantânea máxima de tensão não deve exceder 30 volts por microssegundo. A Figura 2 apresenta os elementos básicos da especificação elétrica do padrão EIA RS-232. A Figura 3 demonstra os níveis lógicos e suas nomenclaturas.

Figura 2 – Elementos da especificação elétrica EIA RS-232.



Fonte: (STRANGIO, 1997).

Figura 3 – Níveis lógicos atribuídos às faixas de tensão RS-232.



Fonte: (STRANGIO, 1997).

1.1.3 Equalização lógica RS-232 para TTL

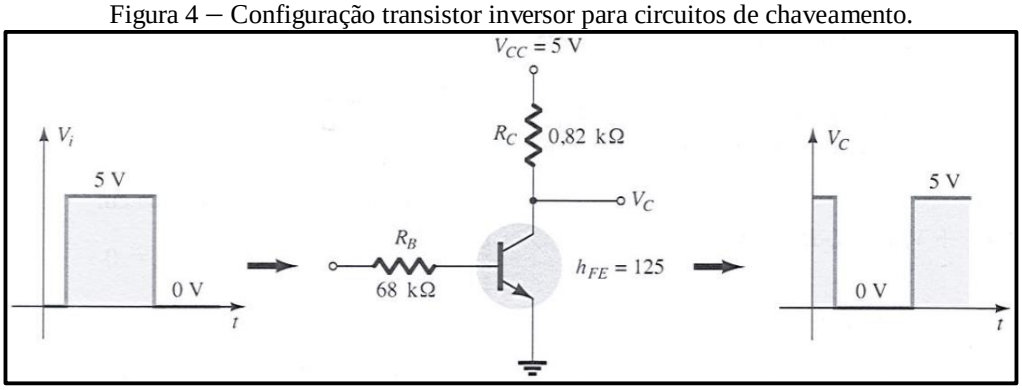
Devido às tensões de trabalho diferentes das entradas do microcontrolador, que atendem aos padrões TTL e CMOS, é necessário projetar um circuito que compatibilize os níveis lógicos de tensão (CIPELLI; MARKUS; SANDRINI, 2007). Nesse sentido, propõe-se desenvolver um circuito que atenda as características elétricas de entrada e que seja compatível com a tecnologia TTL. Dessa forma, para Boylestad e Nashelsky (2013, p. 186), "os transistores podem ser utilizados como chaves em computadores e aplicações de controle". Assim sendo, pode-se utilizar a configuração da Figura 4 como circuito de entrada para definir a saída operando na faixa de 0 a 5 volts.

Para esta configuração, a tensão de saída V_C é oposta à tensão aplicada no terminal de entrada. Boylestad e Nashelsky (2013, p. 186) resumem o princípio de funcionamento do circuito da seguinte forma:

> [...] a única fonte CC é conectada ao coletor ou circuito de saída, e para aplicações em computação, é tipicamente igual à magnitude do nível "alto" do sinal aplicado – no caso, 5 V. O resistor R_B garantirá que a tensão total aplicada de 5 V não apareça através da junção base-emissor. Também definirá o valor e I_B para a condição "ligado" (BOYLESTAD; NASHELSKY, 2013, p. 186).

Ainda sobre o assunto, Boylestad e Nashelsky (2013, p. 186) comentam que "um projeto apropriado para que o transistor atue como um inversor exige que o ponto de operação alterne do corte para a saturação ao longo da reta de carga".

Figura 4 – Configuração transistor inversor para circuitos de chaveamento.



Fonte: (BOYLESTAD; NASHELSKY, 2013, p. 186).

O padrão RS-232 estabelece que a impedância de entrada de um circuito deva estar entre 3 kΩ e 7 kΩ. Considera-se então o valor médio, 5 kΩ. Desta maneira, admite-se que a corrente I_B sobre o resistor R_B com a tensão de entrada *Vi*, tipicamente de magnitude 12 volts, seja definida pela (Equação 1.

$$I_B = \frac{V_i - 0{,}7\ V}{R_B} = \frac{12\ V - 0{,}7\ V}{5\ \text{k}\Omega} \cong 2\ mA \qquad \text{(Equação 1)}$$

Uma característica relevante para esta fase do projeto é a utilização de componentes que não causem atraso de operação maior que o período de comutação do sinal a ser recebido. Em respeito ao teorema da amostragem de Nyquist-Shannon, sabe-se que "a frequência de amostragem tem que ser maior que o dobro da largura da faixa unilateral do sinal no tempo contínuo" (DINIZ; SILVA; NETTO, 2014, p. 38). Nesse caso, refere-se à capacidade de o transistor poder operar o chaveamento das zonas de corte e saturação a taxas compatíveis a 20 kbits por segundo. Sendo assim, o período *t* de transição de estado deve ser inferior a:

$$f = 2 * 20000 = 40000 = \frac{1}{t} \Rightarrow t = \frac{1}{40000} = = 25\ \mu s \qquad \text{(Equação 2)}$$

Uma opção popular no mercado que atenda a essas características é o Transistor Bipolar de Junção (TBJ) 2N2222, comumente utilizado como elemento de chaveamento para circuitos digitais, por possuir temporizações nominais de comutação na ordem de nanossegundos.

1.1.4 Transmissão síncrona

Conforme Zanco (2007, p. 285), na transmissão síncrona, os bits transmitidos são sincronizados com pulsos de *clock* (temporização ou relógio), sendo necessárias pelo menos duas linhas de transmissão, uma para a temporização e outra para os dados. Vários padrões de comunicação serial síncrona foram desenvolvidos e são ainda hoje amplamente utilizados na comunicação de dados. Pode-se destacar entre eles o I^2C, o SPI, o *Microwire Serial Interface* e ainda *Universal Synchronous Asynchronous Receiver Transmitter* (USART), o qual é objeto desta pesquisa.

É comum utilizar os termos *master* (mestre) e *slave* (escravo) numa transmissão síncrona. Normalmente o mestre é aquele que impõe o sinal de temporização. As sinalizações observadas em campo nesta pesquisa podem ser oferecidas tanto pelo ECD quanto pelo ETD. Neste caso, capturaremos todos os sinais de temporização primários, sejam eles:

a) *Transmitter Signal Element Timing* (também chamado de *Transmitter Clock*, ou TC): relevante quando o dispositivo ECD é um modem e está operando com um protocolo síncrono. O modem gera este sinal de temporização para controlar a taxa na qual os dados são enviados em TxD (pino 2) do dispositivo ETD para o dispositivo ECD. Com relação ao modo de operação, a transição lógica ‘1’ para lógica ‘0’ (tensão negativa para tensão positiva) causa uma transição correspondente para o próximo elemento de dados na linha de dados transmitidos. O modem gera este sinal continuamente, exceto quando está executando funções de diagnóstico interno;
b) *Receiver Signal Element Timing* (também chamado de *Receiver Clock, ou RC*): este sinal é semelhante ao TC descrito acima, exceto que fornece informações de tempo para o receptor ETD; e
c) *Transmitter Signal Element Timing* (também chamado *External Transmitter Clock* ou ETC): os sinais de temporização são fornecidos pelo dispositivo ETD para uso

> por um modem. Este sinal é utilizado somente quando TC e RC não estão em uso. A transição lógica ‘1’ para lógica ‘0’ (tensão negativa para tensão positiva) indica o centro de tempo do elemento de dados. Os sinais de temporização serão fornecidos sempre que o ETD for ligado, independentemente de outras condições de sinal (STRANGIO, 1997).

A referência única no tempo entre o transmissor e o receptor é chamada de sincronismo e é estabelecido no início da transmissão de cada mensagem por meio de caracteres especiais, campos no *frame* de dados e até mesmo através de linhas específicas de comunicação. Com relação a isso, Fey e Gauer (2020) esclarecem que

> nas comunicações síncronas, os dados não são enviados em *bytes* individuais, mas como frames de grandes blocos de dados. Tamanhos de frames variam de alguns *bytes* para 1500 *bytes* em redes locais padrão *Ethernet* ou para 4096 *bytes* na maioria dos sistemas *Frame Relay*. O relógio, *clock* ou sincronismo é embutido na codificação de fluxo de dados, ou provido em linhas de sincronismo separadas, de tal maneira que o remetente e o receptor sempre estão em sincronização durante uma transmissão de *frame*. Os *frames* mais modernos são construídos conforme a estrutura de *frame* baseado no protocolo de nível 2 denominado de HDLC (*High-Level Data Link Control*) ou protocolo similar (FEY; GAUER, 2020, p. 221).

1.2 PROTOCOLOS DE COMUNICAÇÃO

Os protocolos de comunicação da camada de enlace de dados são responsáveis pela implementação de mecanismos de controle de fluxo e de controle de erros durante a troca de dados entre transmissor e receptor. Um protocolo é definido como um conjunto de regras e procedimentos que disciplinam a comunicação de dados entre dispositivos, visando a troca de informações de modo ordenado e sem erros, sendo necessários para iniciar, manter e concluir uma comunicação entre dois pontos em um sistema de comunicação. Portanto, um protocolo é composto de uma relação lógica entre códigos (sintaxe), de significados organizados que contextualizam a mensagem (semântica) e de uma temporização disponível aos dispositivos que desejam se comunicar. Desta forma, a sintaxe de um protocolo define os conjuntos de bits divididos em campos. Por sua vez, a semântica define o significado exato dos bits dentro dos campos. Já a temporização define a relação entre a faixa dos bits dentro dos campos e as pausas entre reconhecimentos dos mesmos (FEY; GAUER, 2020, p. 227).

1.2.1 Modelo OSI

A ISO 7498 é um padrão para interconexão de sistemas abertos para sistemas de processamento de informação. O termo *Open Systems Interconnection* (OSI) implica que diferentes sistemas que seguem o padrão estarão aptos a se comunicarem entre si e utiliza uma arquitetura de sete camadas que incluem ponto de acesso ao serviço, protocolos, conexões, entre outros. O mais baixo nível, a camada física (*Physical Layer*), conecta-se com o meio físico enquanto o mais alto nível, a camada de aplicação, conecta-se com o software aplicativo.

Os protocolos ativam um sinal elétrico de um dispositivo para interagir com uma entidade correspondente na mesma camada em outro dispositivo. O conceito de camadas de software (*layers*) separa os módulos de um software em camadas, cada uma com sua contribuição para a execução do mesmo. Este modelo é dividido em sete camadas hierárquicas, onde cada uma usa as funções da própria ou da camada anterior para esconder a complexidade e transparecer de modo simples as operações ao usuário, seja ele um programa ou uma outra camada (FEY; GAUER, 2020, p. 150).

As camadas são empilhadas na seguinte ordem:

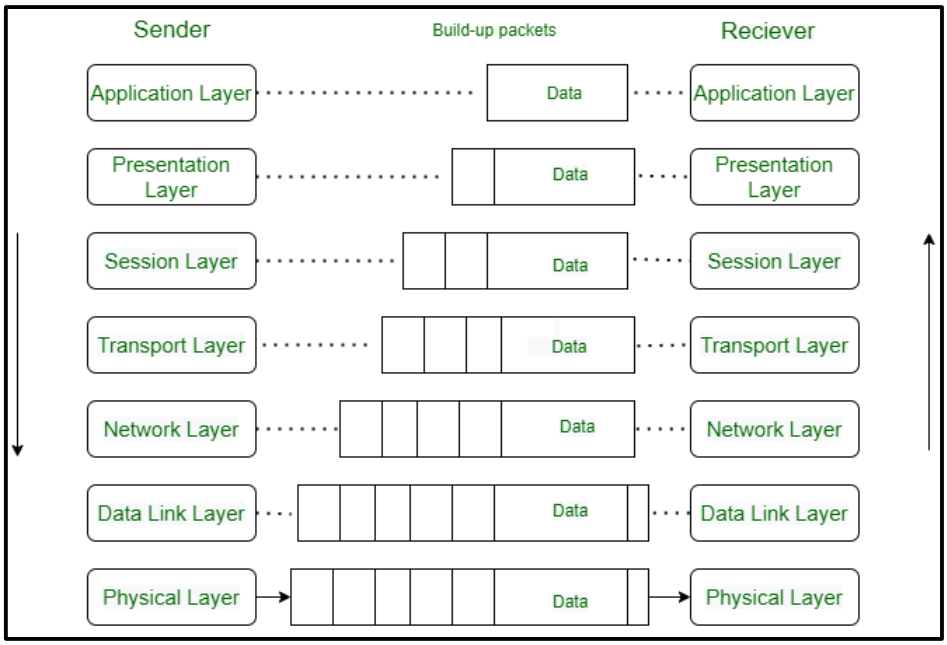
a) aplicação;
b) apresentação;
c) sessão;
d) transporte;
e) rede;
f) enlace de dados; e
g) física.

Interagindo através da transmissão de Unidade de Protocolo de Dados (do inglês *Protocol Data Unit* ou PDU), o bloco de dados é dividido em três partes: cabeçalho (*header*), carga útil ou Unidade de Serviço de Dados (do inglês *Service Data Unit* ou SDU) e rodapé (*trailer*). A SDU é uma unidade específica de dados que foram passados de uma camada OSI para uma camada inferior e que esta ainda não encapsulou em uma PDU.

A PDU é uma camada de N e o SDU uma camada de N-1. Com efeito, a SDU é a carga útil de uma dada PDU. Isto é, o processo de alteração de um SDU a uma PDU é constituído por um processo de encapsulamento, realizada pela camada inferior. Todos os dados contidos no SDU ficam encapsulado dentro do PDU. Deste modo, a camada inferior adiciona cabeçalhos ou rodapés para a SDU, transformando-a numa PDU. Os cabeçalhos ou rodapés adicionados fazem parte do processo utilizado para tornar possível a obtenção de dados de uma fonte para

um destino. A Figura 5 representa a dinâmica dos PDUs enquanto tramitam por meio das camadas (PINHEIRO, 2004).

Figura 5 – Representação da construção de PDUs de acordo com o modelo OSI.



Fonte: (ACERVO LIMA, 2021).

1.2.2 Protocolos síncronos orientados a bit

A maioria dos protocolos atuais possui a característica de uma comunicação síncrona, por transmitirem blocos de caracteres. Em redes locais, vários protocolos síncronos são conhecidos, tais como o *ethernet* (CSMA/CD) e o *token passing*. Já no contexto das redes *Wide Area Network* (WAN), são amplamente conhecidos os protocolos: SDLC, HDLC, PPP e *Frame Relay*. Outra característica dos protocolos decorre da forma como são orientados conforme o tratamento das suas funções. Eles podem ser orientados a bit ou a *byte*. Nos protocolos orientados a *byte*, existe um conjunto de caracteres convencionados para desempenhar determinadas funções. Nos protocolos orientados a bit, essas funções são desempenhadas por conjuntos de bits que têm significado para algumas camadas da arquitetura, considerando que estas foram desenvolvidas de acordo com um determinado modelo e organizadas em camadas que realizam uma função bem definida. Os protocolos orientados a bit não utilizam caracteres específicos e os campos de informação, endereço e controle são tratados em nível de bit. São diferenciados dos orientados a *byte* (caractere) por serem *half* ou *full-duplex*, independentes dos códigos, permitindo blocos de tamanho maior. São exemplos típicos SDLC, BDLC, HDLC e X.25.

A técnica utilizada para administrar uma linha multiponto é que só um secundário pode estar transmitindo num determinado momento. Se dois ou mais secundários transmitirem ao mesmo tempo, há interferência entre eles e os dados de ambos são perdidos. Por causa disto, transmissões dos secundários são controladas pelo primário, ou seja, um secundário só pode transmitir quando permitido pelo primário, além de transmitir apenas um número limitado de *frames* antes de devolver o controle ao primário. As estações secundárias têm modems ou Unidades de Serviços de Dados (DSU) que são capazes de comutar a portadora para ligado ou desligado. A portadora é controlada pelo sinal de *Request to Send* (RTS) enviado do terminal para o modem. O secundário eleva o RTS quando é permitido transmitir pelo primário. O modem comuta a portadora para ligado. Pode haver uma demora curta de alguns milissegundos para a portadora estabilizar e então o modem eleva o sinal *Clear to Send* (CTS), no sentido modem para o terminal. O equipamento secundário então transmite. Quando a transmissão está completa, o secundário derruba o sinal RTS, o modem derruba o sinal CTS e então comuta a portadora para desligado ao mesmo tempo. Desse modo, o próximo secundário pode transmitir, se permitido pelo equipamento primário (FEY; GAUER, 2020). A Figura 6 demonstra um diagrama de tempo que relaciona as sinalizações supracitadas.

Figura 6 – Diagrama de portadora controlada por RTS/CTS.

Fonte: (FEY; GAUER, 2020, p. 232).

1.2.3 Protocolo HDLC

A camada de enlace de dados divide-se em Controle de *Link* Lógico (LLC), responsável pelo controle de aceitação de mensagem, e Controle de Acesso ao Meio (MAC), responsável por manter e oferecer condições básicas para operação, controlando o acesso ao meio físico, detectando erros e desencapsulando mensagens (LUGLI; SANTOS, 2009).

Baseado no protocolo SDLC da IBM, em 1979, a OSI criou o protocolo HDLC visando a padronização de um protocolo orientado a bit para transmissão de dados síncrona *half* ou *full-duplex*, podendo operar em linhas comutadas ou permanentes, ponto a ponto ou multiponto.

Este padrão corresponde à camada de enlace de dados e é responsável pela transmissão de dados livres de erros entre nós da rede. Sua PDU é o quadro, internacionalmente conhecido como *frame*. Um quadro é conceitualmente uma sequência de campos de endereço, controle, informação e código verificador de erro, delimitado por sinalizadores de abertura e fechamento (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2002, p. 5). A Figura 7 demonstra o formato do quadro do protocolo HDLC da OSI.

Figura 7 – Formato do *frame* HDLC.

Fonte: (FEY; GAUER, 2020, p. 234).

1.2.3.1 Inserção de bits

O tipo de enquadramento empregado pelo protocolo HDLC utiliza o método de inserção de bit, através do uso de *flags* iniciais e finais. Sobre o assunto, Tanenbaum e Wetherall (2011) esclarecem que

> cada quadro começa e termina com um padrão de bits especial, 01111110, ou 0x7E em hexadecimal. Sempre que encontra cinco valores 1 consecutivos nos dados, a camada de enlace de dados do transmissor automaticamente insere um bit 0 no fluxo de bits que está sendo enviado. Ao ver cinco bits 1 consecutivos sendo recebidos, seguidos por um bit 0, o receptor automaticamente remove o bit 0 (TANENBAUM; WETHERALL, 2011, p. 125).

O trabalho realizado por Høyer e Kvamme (2004), permite realizar o processo de bit *de-stuffing* (a remoção de bits inseridos pelo transmissor) sem o auxílio de um *hardware* dedicado. O bit *de-stuffing* é normalmente realizado por *hardware* diretamente nas portas seriais ou por algum hardware adicional na entrada de dados do receptor. No entanto, quando a manipulação de *flags* e bit *stuffing* é realizada por software, é difícil determinar um código eficiente. A razão para isso é que os processadores digitais de sinal (do inglês Digital Signal Processors ou DSP) e as unidades centrais de processamento (do inglês Central Process Unit ou CPU) são projetados para trabalhar eficientemente com *bytes* (8 bits), *words* (16 bits) e *long words* (32 bits), e não com bits únicos. A CPU normalmente precisa usar várias instruções para lidar com cada bit (HØYER; KVAMME, 2004, p. 2).

A *flag* inicial é um único *byte* que indica o início e o fim do *frame* e consiste de uma sequência binária 01111110. O campo endereço normalmente contém o endereço de uma estação secundária e é vazio para *links* ponto a ponto. O formato do campo controle varia em função do tipo de *frame* enviado. Há três tipos de *frame* HDLC, sendo eles:

a) *Information* (I-frame) – Transporta dados;
b) *Supervisory* (S-frame) – Transporta comandos e respostas; e
c) *Unnumbered* (U-frame) – Transporta sequência de comandos adicionais.

1.2.3.2 Detecção de erros

O campo de dados, naturalmente, contém os dados úteis que se deseja transmitir. Já o campo *Frame Check Sequence* (FCS) é necessário para detecção de erros e usa algoritmo de 16 bits (2 *byte*s) ou 32 bits (4 *byte*s). A FCS é um tipo de código verificador de quadros de redundância cíclica (*Cyclic Redundancy Check* - CRC), projetado para detectar pacotes de dados em erro no receptor, verificando a integridade do bloco de dados através de uma técnica de polinômios geradores matriciais. Cada pacote é codificado por códigos de comprimento $n \leq 2^m - 1$. Os códigos mais comuns têm m = 12, 16 ou 32, onde n o tamanho do comprimento e m o grau do polinômio gerador (CAVALCANTE, 2016, p. 162).

A Tabela 1 apresenta a relação de cada campo de um quadro HDLC e seus comprimentos em bits.

Tabela 1 - Relação de comprimentos em bits dos campos de um *frame* HDLC.

Nome do campo	Comprimento em bits
Flag inicial	8
Endereço	8
Controle	8 ou 16
Dados	Variável ou zerado em alguns *frames*
FCS	16 ou 32
Flag final	8

Fonte: o próprio autor.

A FCS é calculada para todo o comprimento do respectivo quadro, excluindo os campos de *flag* de abertura, o próprio FCS e quaisquer outros elementos de sincronismo (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2002, p. 12). A FCS é gerada quando um quadro é transmitido (antes de qualquer preenchimento de bits) e é verificada quando um quadro é recebido. Se um ou mais bits foram modificados durante a transmissão

devido a interferências do meio físico, a verificação da FCS falha e, nesse caso, todo o quadro é descartado. O mecanismo de uso da FCS permite apenas que se verifique a integridade do quadro e não a sua recuperação, que, nesse caso, deve ser enviado novamente ao receptor.

O HDLC *Link Access Procedure Balanced* (LAPB), uma das implementações do HDLC, é um protocolo síncrono orientado a bit que provê transparência de dados completa em uma operação ponto a ponto *full-duplex*. Ele suporta uma ligação *peer to peer*, no qual nenhum equipamento nas pontas desempenha o papel da estação mestre permanente. O HDLC LAPB é um protocolo muito eficiente, pois um mínimo de *overhead* é exigido para assegurar o controle de fluxo, detecção de erro e recuperação do mesmo. Se os dados estão fluindo em ambas as direções (*full-duplex*), os próprios quadros de dados levam todas as informações exigidas para assegurar a integridade de dados (FEY; GAUER, 2020).

Sobre as características de uso do HLDC, Fey e Gauer (2020) comentam ainda que

> o conceito de uma janela de frame (conceito semelhante ao *windows sizing* do TCP/IP) é usado para se poder enviar frames múltiplos antes da confirmação da parte receptora que o primeiro *frame* foi recebido corretamente. Isto significa que dados podem continuar fluindo em situações onde pode haver longo atraso de tempo de '*turnaround'*, sem paradas para esperar por um reconhecimento. Este tipo de situação acontece, por exemplo, em comunicação por satélite. Os tamanhos de janela variam, mas é tipicamente de 7 *frames* para a maioria das linhas terrestres e de até 128 *frames* para ligações por satélite (FEY; GAUER, 2020, p. 235),

justificativa para tal protocolo ser usado para os enlaces de radiodeterminação no contexto do SISCEAB, onde boa parte desses enlaces é por satélite e provê boa recuperação em ambientes com muita interferência no meio, além do fato de não ser necessário o estabelecimento de um equipamento mestre permanente.

O procedimento para uso da FCS é baseado nas seguintes suposições (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2002, p. 95):

a) os k bits de dados a serem verificados pela FCS podem ser representados por um polinômio G(x), como por exemplo o $G(x) = x^5 + x^3 + 1$, que representa 101001;
b) os campos de endereço, controle e informação (se existir no quadro) são representados pelo polinômio G(x);
c) o primeiro bit após a *flag* de abertura é o bit mais significativo de G(x) com o propósito de gerar a FCS, independente da representação real dos campos de endereço, controle e informação; e
d) há um polinômio gerador P(x) de grau 16, tendo a forma $G(x) = x^{16} + x^{12} + x^5 + 1$.

A FCS é definida como o complemento de um resto R(x) obtido da divisão módulo 2 de $x^{16}\,G(x) + x^k(x^{15} + x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1)$ pelo gerador polinomial P(x). Desse modo:

$$\frac{x^{16}G(x) \; + \; x^k \; (x^{15} \; + \; x^{14} \; + \cdots + \; x \; + \; 1)}{P(x)} = Q(x) + \frac{R(x)}{P(x)} \qquad \text{(Equação 3)}$$

A (Equação 3) demonstra que a multiplicação de G(x) por x^{16} corresponde a um deslocamento de G(x) de 16 casas, fornecendo o espaço de 16 bits para a FCS. A adição de x^k ($x^{15} + x^{14} + ... + x + 1$) a $x^{16}G(x)$, é fornecida para proteção contra a perda de sinalizadores iniciais, que podem não ser detectáveis se o restante inicial for zero. "A complementação de R(x), pelo transmissor, na conclusão da divisão garante que a mensagem recebida, sem erros, resultará em um único resto diferente de zero no receptor" (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2002, p. 96, tradução nossa).

No transmissor, a FCS é adicionada ao $x^{16}\,G(x)$, resultando em uma mensagem M(x) de tamanho n, onde $M(x) = x^{16}\,G(x) + FCS$. No receptor, a mensagem recebida M(x) é multiplicada por x^{16}, adicionada a $x^n\,(x^{15} + x^{14} + ... + x + 1)$ e dividida por P(x).

$$\frac{x^{16}[x^{16}G(x) + FCS] + x^n\,(x^{15} + x^{14} + \cdots + x + 1)}{P(x)} = Qr(x) + \frac{Pr(x)}{P(x)} \qquad \text{(Equação 4)}$$

Se a transmissão é livre de erro, o resto Rr(x) será 0001 1101 0000 1111. Dessa forma, Rr(x) é o resto da divisão:

$$\frac{x^{16}L(x)}{P(x)}, \qquad L(x) = \; x^{15} \; + x^{14} + \ldots + x \; + \; 1 \qquad \text{(Equação 5)}$$

Nota-se ainda que FCS = R(x) = L(x) + R(x). A (Equação 4, para o resíduo do receptor FCS, é usada para mostrar que inverter a FCS no receptor retorna o registrador de verificação para zero, onde L(x), definido pela (Equação 5), é o dado original a ser transmitido e Rr(x) é o conteúdo residual do registrador FCS. Assim sendo, evidencia-se a (Equação 6) (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2002, p. 96, tradução nossa).

$$\frac{x^{16}L(x)}{P(x)} = Q(x) + \frac{Rr(x)}{P(x)} \qquad \text{(Equação 6)}$$

Fisicamente, a adição x^{16} L(x) de é alcançada ao inverter a FCS.

1.2.4 Protocolo ASTERIX

O protocolo *All Purpose Structured Eurocontrol Surveillance Information Exchange* (ASTERIX) é um padrão estabelecido pela EUROCONTROL que atua nas camadas de apresentação e aplicação do modelo OSI. O ASTERIX é um conjunto de documentos que define a implementação de baixo nível, em bit, de um formato de dados usado para trocar informações relacionadas à vigilância e a outras aplicações de gerenciamento de tráfego aéreo. O protocolo ASTERIX é projetado para meios de comunicação com largura de banda limitada, por isso, segue regras que permitem transmitir todas as informações necessárias, com a menor carga de dados possível (EUROCONTROL, 2021, p. 13).

O protocolo ASTERIX é definido em 256 categorias de três grupos:

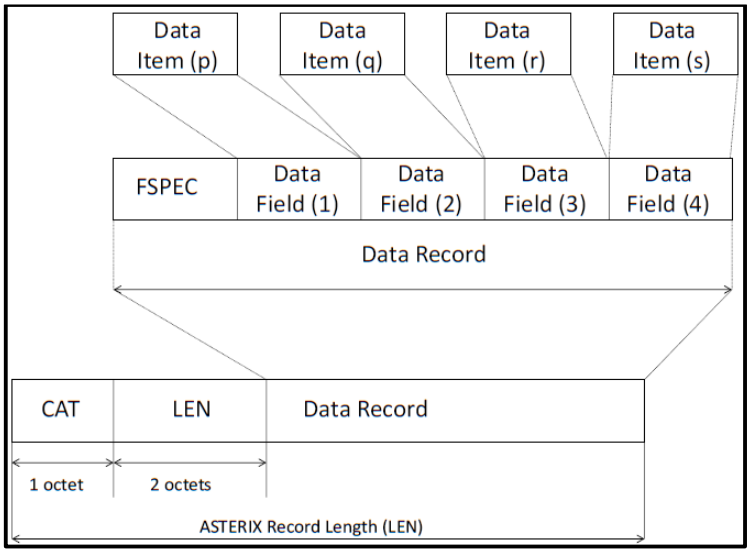
a) *Standard* – Uso civil e militar para as categorias da faixa de 0 a 127;
b) *Special* – Exclusivamente militar para as categorias da faixa de 128 a 240; e
c) *Non-standard* – Uso civil e militar para customizações da faixa de 241 a 255.

Os dados trocados pelo meio de comunicação entre os diferentes usuários são organizados em categorias que determinam seu tipo e seu formato a ser aplicado. Esta classificação de categorias padronizadas é aplicada por todos os usuários do ASTERIX com o objetivo de permitir a fácil identificação e o posterior tratamento dos dados, bem como de facilitar o envio dos dados para a tarefa de aplicação apropriada na unidade receptora. Para cada categoria é definido um catálogo de itens de dados. As solicitações que envolvam a troca de informações de uma determinada categoria fazem uso exclusivamente dos itens de dados de um dos catálogos. Em casos não padronizados, essa informação precisa ser trocada através do campo de finalidade especial. Cada item de dados recebe uma referência inequívoca, chamada de *Field Reference Number* (FRN), que identifica este item no respectivo catálogo (EUROCONTROL, 2021, p. 20).

A Figura 8 demonstra a organização da estrutura de dados do protocolo ASTERIX. O campo CAT, conhecido como *Data Category*, indica a categoria do bloco. O campo LEN, chamado de *Record Length Indicator*, representa o tamanho total do bloco de dados, incluindo os campos CAT e LEN. O mecanismo de extensão FSPEC sinaliza a presença ou a ausência de

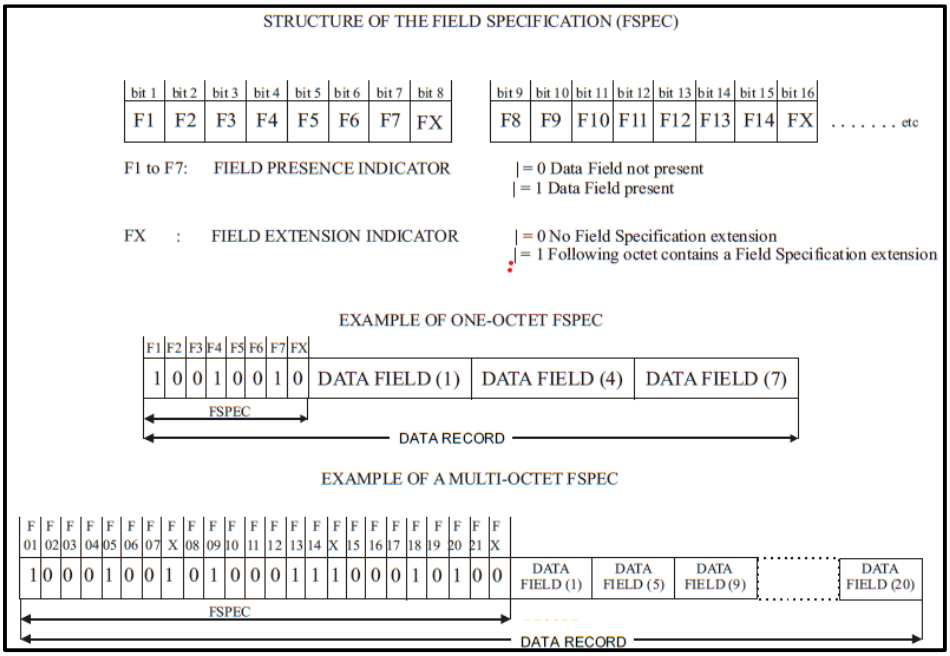
um *Data Field* e seu comprimento mínimo é de um octeto. Ele é usado quando um *Data Field* com FRN maiores que sete precisam ser transmitidos, definindo o bit menos significativo (LSB) como um, até que um octeto seja encontrado com o LSB definido como zero. O LSB no campo FSPEC é chamado de *Field Extension Indicator* (FX) (EUROCONTROL, 2021, p. 26). A Figura 9 apresenta como é estabelecida a organização dos campos em um registro de dados do protocolo ASTERIX.

Figura 8 – Estrutura de gravação de dados do protocolo ASTERIX.



Fonte: (EUROCONTROL, 2021, p. 25).

Figura 9 – Organização do sequenciamenteo de campos do protocolo ASTERIX.



Fonte: (EUROCONTROL, 2021, p. 38).

Cada *Data Item* está relacionado a uma *User Application Profile* (UAP), que é uma coleção unívoca para cada categoria de aplicação de mensagens ASTERIX. Essa coleção define quais *Data Item* catalogados serão usados, seu comprimento, sua atribuição aos campos de dados e quaisquer requisitos específicos padronizados para a transmissão e interpretação das mensagens. Uma UAP é única para cada categoria de mensagem. A Figura 10 mostra um exemplo de UAP de categoria genérica nnn.

Figura 10 – Exemplo de uma UAP de categoria nnn.

FRN	Data Item	Information	Length
1	Innn/010	Data Source Identifier	2
2	-	Spare	-
3	Innn/015	Service Identification	1
4	Innn/070	Time Of Track Information	3
5	Innn/105	Calculated Track Position (WGS-84)	8
6	Innn/100	Calculated Track Position (Cartesian)	6
7	Innn/185	Calculated Track Velocity (Cartesian)	4
FX	-	Field extension indicator	-
8	Innn/210	Calculated Acceleration (Cartesian)	2
9	Innn/060	Track Mode 3/A Code	2
10	Innn/245	Target Identification	7
11	-	Spare	-
12	-	Spare	-
13	-	Spare	-
14	-	Spare	-
FX	-	Field extension indicator	-

Fonte: (EUROCONTROL, 2021, p. 21).

1.3 MICROCONTROLADOR STM32

A família de microcontroladores STM32 da STMicroelectronics é projetada para aplicações de tempo real com baixo consumo de energia, processamento digital de sinais e conectividade. São microcontroladores de 32 bits e arquitetura *ARM® Cortex®-M*. Para Brown (2018, p. 13), essa família de microcontroladores "oferece uma estrutura para criação de uma vasta quantidade de sistemas embarcados, desde simples *dongles* alimentados à bateria até sistemas complexos de tempo real como helicópteros auto pilotados".

A grande vantagem dos microcontroladores é integrar, no mesmo componente, as diversas interfaces de entrada e saída, além de funcionalidades como memória e circuitos osciladores. Interfaces de entrada analógica também são comuns (OLIVEIRA, 2017). A escolha de uma plataforma baseada neste microcontrolador se deu após ponderar algumas características como arquitetura, consumo, periféricos, velocidade e capacidade de

processamento, tamanho e encapsulamento. O MCU STM32F103C8T6 em um projeto utilizando uma *Printed Circuit Board* (PCB) azul é conhecido como *Blue Pill*.

A placa *Blue Pill* foi escolhida devido aos seguintes fatores:

a) custo baixo;
b) disponibilidade (encontrado facilmente no mercado nacional);
c) capacidade avançada; e
d) formato pequeno da PCB.

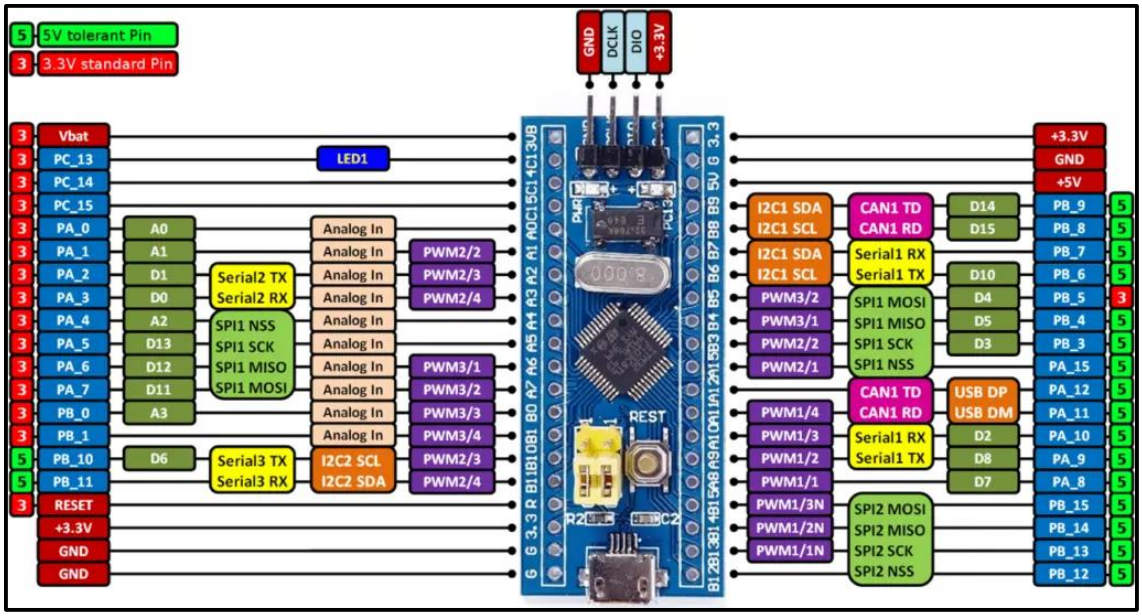
A *Blue Pill* é uma boa alternativa de baixo custo para estudantes e entusiastas que gostam de explorar a plataforma *ARM Cortex-M3*. O dispositivo está prontamente disponível e é extremamente capaz. Seu formato pequeno dentro de uma placa de circuito impresso permite que terminais sejam soldados em suas bordas e conectados diretamente em uma *protoboard* ou matriz de contatos. O baixo custo permite que você possua vários dispositivos para projetos envolvendo comunicações *Contoller Area Network* (CAN), por exemplo (SILVA, 2020).

Os periféricos incluídos consistem em:

a) 4 portas GPIO de 16 bits (a maioria é tolerante a 5 volts);
b) 3 x USART;
c) 2 x controladores I2C;
d) 2 x controladores SPI;
e) 2 x ADC (conversor analógico para digital);
f) 2 x DMA (controladores de acesso direto à memória);
g) 4 x temporizadores;
h) Temporizadores *Watchdog*;
i) 1 x controlador USB;
j) 1 x controlador CAN;
k) 1 x gerador CRC;
l) RAM estática de 20K;
m) Memória *Flash* 64K (ou 128K); e
n) CPU *ARM Cortex M3*, *clock* máximo de 72 MHz.

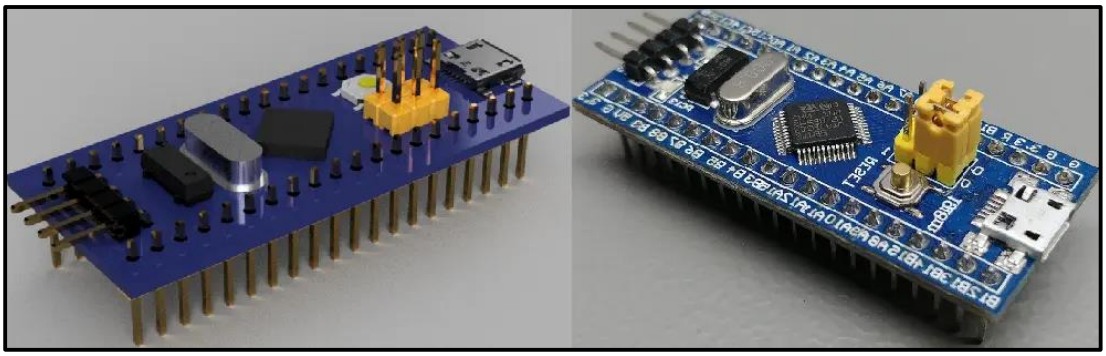
A Figura 11 e a Figura 12 ilustram, respectivamente, a pinagem disponível na *Blue Pill* e o dispositivo físico comercialmente encontrado.

Figura 11 – Pinagens oferecidas pela plataforma *Blue Pill*.



Fonte: (SILVA, 2020).

Figura 12 – Placa *Blue Pill* com chip STM32F103C8T6.



Fonte: (SILVA, 2020).

1.3.1 Drivers HAL

A camada de *driver Hardware Abstraction Layer* (HAL) fornece um conjunto simples e genérico de várias instâncias de interfaces de programação de aplicativos (API) para interagir com a camada superior (aplicativos, bibliotecas e pilhas).

As APIs do *driver* HAL são divididas em duas categorias: APIs genéricas, que fornecem funções comuns e genéricas para toda a série STM32 e APIs de extensão, que incluem funções específicas e personalizadas para uma determinada linha de microcontrolador. Os *drivers* HAL incluem um conjunto completo de APIs prontas para uso que simplificam a implementação do aplicativo do usuário. Por exemplo, os periféricos de comunicação contêm APIs para inicializar e configurar o periférico, gerenciar transferências de dados, lidar com interrupções ou acesso direto à memória e gerenciar erros de comunicação (STMICROELECTRONICS, 2020, p. 1).

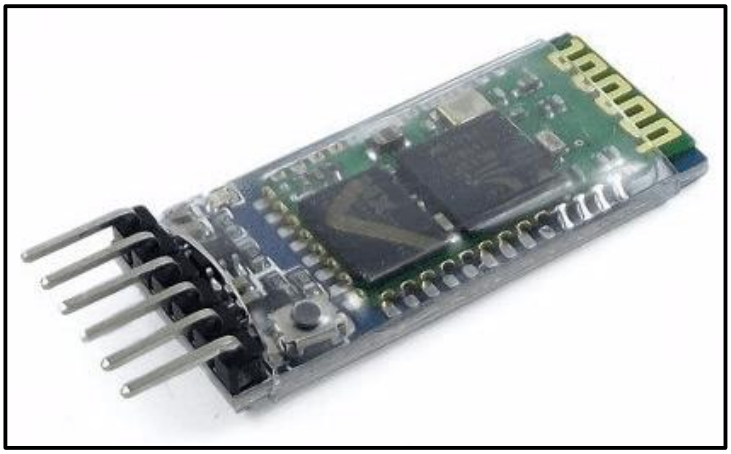
Os *drivers* HAL são orientados a recursos. Por exemplo, as APIs de temporizadores são divididas em várias categorias seguindo as funções de interrupção, como temporizador básico, captura e modulação de largura de pulso. Também implementa a detecção de falhas em tempo de execução verificando os valores de entrada de todas as funções. Essa verificação dinâmica aumenta a robustez do *firmware*. A detecção em tempo de execução também é adequada para o desenvolvimento e depuração de aplicativos do usuário.

1.4 COMUNICAÇÃO SEM FIO

"O protocolo *bluetooth*, definido pelo padrão IEEE 802.15.1 foi o primeiro protocolo *Wireless Personal Area Network* (WPAN) a se tornar comercial e se popularizar" (OLIVEIRA, 2017, p. 38). Em uma época em que as conexões *WiFi* e 3G ainda não eram a realidade da maioria dos usuários, essa tecnologia se tornou bastante útil, principalmente para troca de arquivos entre dispositivos e ligação com periféricos como fones de ouvido ou teclados e mouses (OLIVEIRA, 2017, p. 39).

Nesta pesquisa, essa tecnologia será aplicada para envio de informações entre o protótipo que estará acoplado na interface serial DB-25 e o dispositivo móvel que conterá a interface com o usuário, por meio do transceptor HC-05. Este transceptor permitirá uma conexão sem fio com o dispositivo móvel, através do uso de uma das interfaces USART do STM32. A Figura 13 apresenta o módulo HC-05 como é comercialmente difundido.

Figura 13 – Módulo transceptor HC-05.

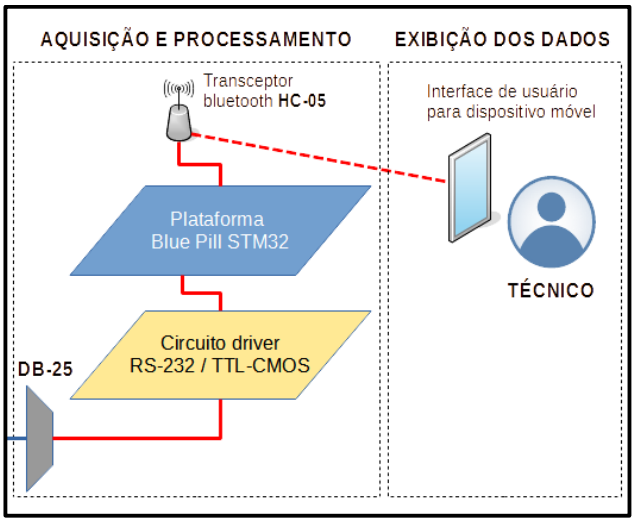


Fonte: (OLIVEIRA, 2016).

2 MATERIAIS E MÉTODOS

Neste capítulo são apresentados o método proposto, materiais e ferramentas utilizados para a implementação do projeto. As etapas de implementação são apresentadas conforme a entrada, processamento e saída de dados, tendo como ponto de partida a equalização dos sinais RS-232 para TTL, de modo a possibilitar a leitura dos sinais de comunicação serial ao microcontrolador STM32, passando pelo processamento dos quadros HDLC pelo programa embutido no microcontrolador, o envio de mensagens por *bluetooth* no formato *JavaScript Object Notation* (JSON) e, por fim, a apresentação dos quadros HDLC na tela do aplicativo para dispositivo móvel *Android*. A Figura 14 apresenta em módulos os componentes do protótipo.

Figura 14 – Modelo conceitual do protótipo.



Fonte: o próprio autor.

2.1 MÉTODO PROPOSTO

Na primeira etapa, foi desenvolvido um circuito eletrônico para compatibilidade entre os níveis lógicos RS-232 e TTL, de modo que a entrada de comunicação serial do microcontrolador STM32, pudesse perceber os sinais de espaço e marca parametrizados dentro da faixa de tensão de trabalho do padrão TTL, ou seja, de 0 a 5 volts. Este circuito eletrônico também serve para acomodar a plataforma *Blue Pill*, o módulo HC-05, selecionar os canais que serão percebidos pela placa e ainda indicar por meio de *leds* se há tráfego de dados em cada pinagem das entradas.

Na segunda etapa, foi escrito o programa para STM32 capaz de receber os dados seriais, identificar características de *baudrate* e do estado lógico de pinagens de controle de fluxo, assim como verificar a validade dos quadros através de processo de bit *de-stuffing* e da decodificação do FCS de cada quadro HDLC. Essas informações são então convertidas em mensagens de texto para posterior envio ao dispositivo por meio do transceptor HC-05.

Na terceira etapa, foi desenvolvido o aplicativo que recebe mensagens do módulo HC-05 através da interface *bluetooth* do dispositivo móvel. Essas mensagens contêm os campos de interesse dos quadros HDLC e as características do meio físico, sendo tratadas e apresentadas na tela do usuário.

2.2 MATERIAIS UTILIZADOS

Para montagem de um circuito inicial para gravação e testes de conectividade sem fio, foram necessários:

1 *protoboard* de 400 pinos.
1 módulo *Blue Pill* (STM32F103C8T6).
1 módulo *Bluetooth* HC-05.
1 gravador ST-Link v2.
1 multímetro digital Hikari HM-1000.
Cabos para conexões (cabos *jumper*).

Para a implementação do projeto foram usadas as seguintes ferramentas:
Software Proteus Design Suite 8.10 para *Windows* 10 de 64 bits.
Software STM32CubeIDE 1.9.0 para *Windows* 10 de 64 bits.
Software Android Studio Bumblebee | 2021.1.1 *Patch* 2 para *Windows* 10 de 64 bits.
VirtualBench All-in-One Instrument da *National Instruments*.
Osciloscópio Tektronix TDS 3032C de dois canais, 300 MHz a 2,5 GS/s.
Ferro de solda de 30 watts.
Ponta para ferro de solda em formato de fenda de 30 watts.
Tubo de solda estanho de 1,0 mm.
Alicate de corte.

Para a confecção do protótipo e testes, foram necessários os seguintes materiais:
1 caixa de derivação cbox-ob 150mm x 110mm x 70mm opaca.

4 parafusos 3mm x 30mm.

4 parafusos 3mm x 10mm.

12 porcas sextavadas 3mm.

1 placa de circuito impresso do protótipo.

1 módulo *Blue Pill* (STM32F103C8T6).

1 módulo *Bluetooth* RS232 HC-05.

1 fonte alimentação de 5,0 volts a 1,0 ampère.

2 circuitos integrados *buffer* 74LS07.

1 barra gráfica de leds verde.

1 barra gráfica de leds vermelha.

10 transistores NPN 2N2222

10 diodos 1N4148.

10 resistores 1 kΩ (1/4W).

10 resistores 4,7 kΩ (1/4W).

10 resistores 10 kΩ (1/4W).

2 conectores DB-25 fêmea soldáveis.

1 módulo *dip switch* de 10 vias de 180 Graus.

2 soquetes de 14 pinos.

1 soquete de 40 pinos.

1 borne de 2 polos.

2 barras de pinos de 12 vias.

1 barra de soquete de 6 vias.

1 barra de soquete de 12 vias.

Cabos para conexões (cabos *jumper*).

3 IMPLEMENTAÇÃO DO PROJETO

Neste capítulo são apresentados os procedimentos de forma detalhada para o desenvolvimento do protótipo, o qual é composto do projeto de um circuito que converte as tensões elétricas do padrão RS-232 para o padrão TTL, além de exibir quais sinais estão habilitados na entrada por meio de barras gráficas de leds. Em seguida, a elaboração de um programa que executa a leitura dos quadros HDLC no microcontrolador STM32, verifica os sinais de controle de fluxo e valida e organiza os campos dos quadros para enviar ao transceptor HC-05 em formato de texto. Por fim, é demonstrado o desenvolvimento do aplicativo que exibirá as informações enviadas pelo protótipo.

São apresentados nesse capítulo:

a) projeto de PCB condicionadora de sinais RS-232/TTL;
b) montagem dos componentes na PCB e testes de bancada;
c) elaboração do programa para captura dos sinais de controle do meio de comunicação e interpretação dos quadros HDLC no *STM32*; e
d) elaboração do aplicativo para apresentação dos quadros HDLC e sinais de controle.

3.1 PROJETO DE PCB PARA EQUALIZAÇÃO RS-232/TTL

No ambiente onde as avaliações dos canais de radiodeterminação são executadas, fatores como mobilidade, praticidade, robustez e confiabilidade são, além de desejáveis, relevantes para inspecionar as características desses canais com agilidade e rigor necessários.

Em virtude disso, foi estabelecido um primeiro conceito usando componentes discretos, de fácil acesso no mercado nacional e de manuseio através de ferramentas de prototipagem.

Respeitando os padrões RS-232 e TTL, o circuito deve ter como características elétricas:

a) impedância de entrada: de 3 kΩ a 7 kΩ;
b) deve operar a taxas até 20 kbits ciclos por segundo;
c) tensão de alimentação dos circuitos: 5,0 V a no máximo 1,0 A; e
d) corrente mínima de dreno na saída: 16 mA.

Através do módulo *ISIS* do *Proteus Design Suite* foi possível elaborar um equalizador das faixas de tensão RS-232 para TTL. Um resistor RB de 4,7 kΩ serve de impedância de entrada do circuito e de elemento polarizador da corrente de base do transistor. Nessa configuração, emissor comum, o transistor opera nas regiões de corte e saturação, provocando o chaveamento da corrente I_C. No coletor do transistor, o resistor RC deve suportar a corrente

I_C no estado de saturação e deve servir de resistor de *pull-up* no estado de corte, estabelecendo a tensão para nível lógico alto. Na saturação, a corrente máxima suportada pelo transistor é definida pela (Equação 7).

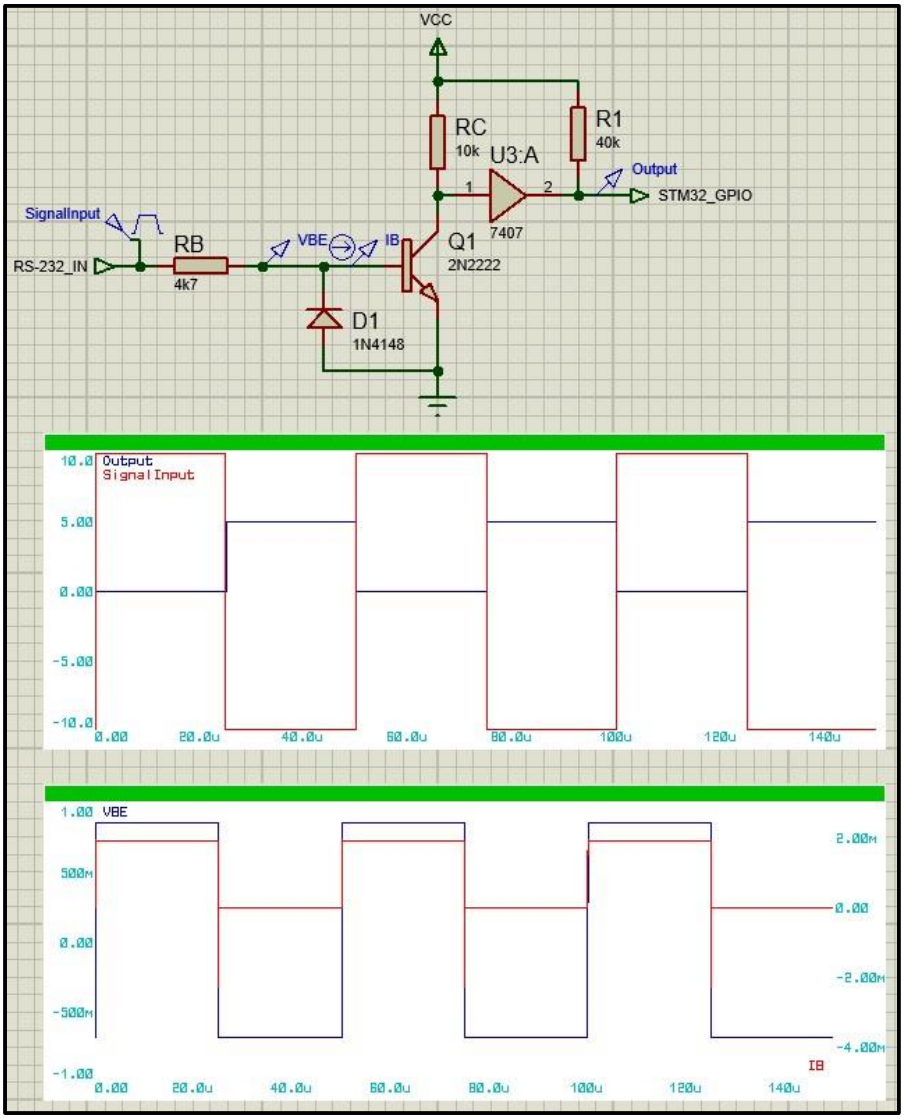
$$I_E = I_C = \frac{VCC}{RC} = \frac{5\ V}{10\ k\Omega} = 500\ \mu A \qquad \text{(Equação 7)}$$

Ao coletor do transistor, foi adicionado um circuito integrado 7407, *buffer*, responsável por garantir a impedância de saída do circuito chaveador transistorizado conforme os padrões TTL. Este circuito integrado possui seis buffers independentes, o que permite manter agrupado um número significativo de entradas em um pequeno espaço. É importante salientar que este circuito é compatível com o 7406, *buffer* inversor, pois tem as mesmas pinagens de entradas, saídas e alimentações. Desse modo, é possível intercambiar essa saída para inversão das tensões de entrada ao microcontrolador, a depender da necessidade.

O *datasheet* do transistor 2N2222 informa que a tensão máxima reversa da junção base-emissor é de 6 V. Para proteção dessa junção, foi inserido ao circuito um diodo retificador 1N4148, recomendado para circuitos de chaveamento de alta velocidade por possui período de comutação máxima de apenas 4 ns.

Os resistores foram dimensionados conforme os valores comerciais e como teste de simulação foi configurada uma entrada pulsada de -12 V a +12 V, operando a 50% em cada nível, a uma taxa de 19200 ciclos por segundo. O resistor R1 foi inserido na simulação para emular o efeito do resistor de *pull-up* interno da porta GPIO do microcontrolador. A Figura 15 apresenta o circuito concebido, seguindo as características supracitadas e gráficos simulando a entrada e a saída por 150 µs, para verificar o comportamento das tensões e correntes de operação.

Figura 15 – Concepção do condicionamento RS-232 para TTL.



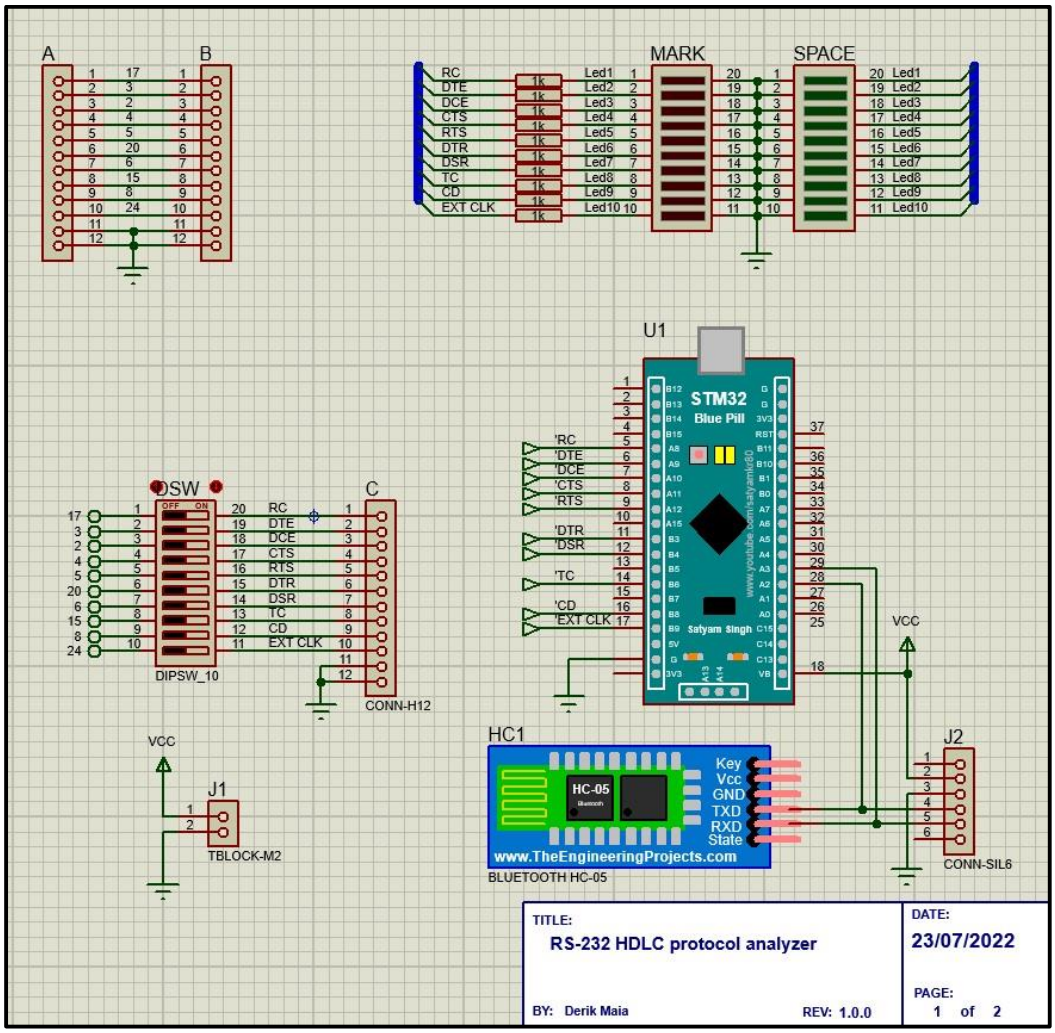
Fonte: o próprio autor.

A placa de equalização também deve servir para acomodar os outros dispositivos, entradas, conexões e chaves de seleção. Nesse sentido, foi desenhado o esquemático da Figura 16 que demonstra o esquemático elaborado para dispor as interfaces de entrada, as chaves seletoras para os diferentes sinais de entrada, os indicadores luminosos, os módulos de processamento e comunicação de dados, bem como a entrada da fonte de alimentação. Mais detalhes sobre o esquemático completo podem ser encontrados no Apêndice A**Erro! Fonte de referência não encontrada.**.

A Figura 17 apresenta o modelo 3D concebido para a PCB modelada utilizando a ferramenta ARES. O projeto levou em consideração que a maioria dos fabricantes de PCBs, de modo a incentivar empresas de pequeno porte, permitem a fabricação de placas em custo

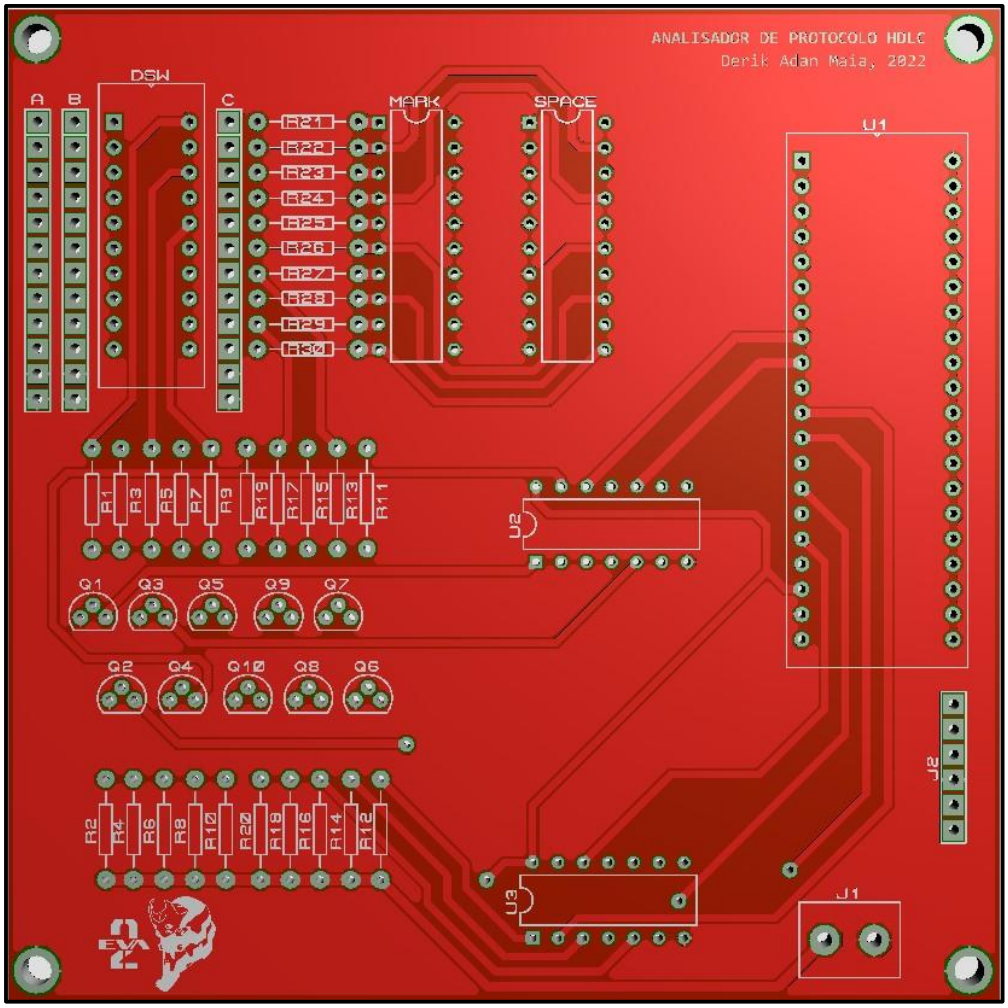
reduzido para circuitos de até 100 mm por 100 mm em baixo volume de produção, normalmente com pedido mínimo de cinco unidades.

Figura 16 – Esquemático das interfaces e módulos do protótipo.



Fonte: o próprio autor.

Figura 17 – Modelo 3D da PCB equalizadora RS-232/TTL sem os componentes.



Fonte: o próprio autor.

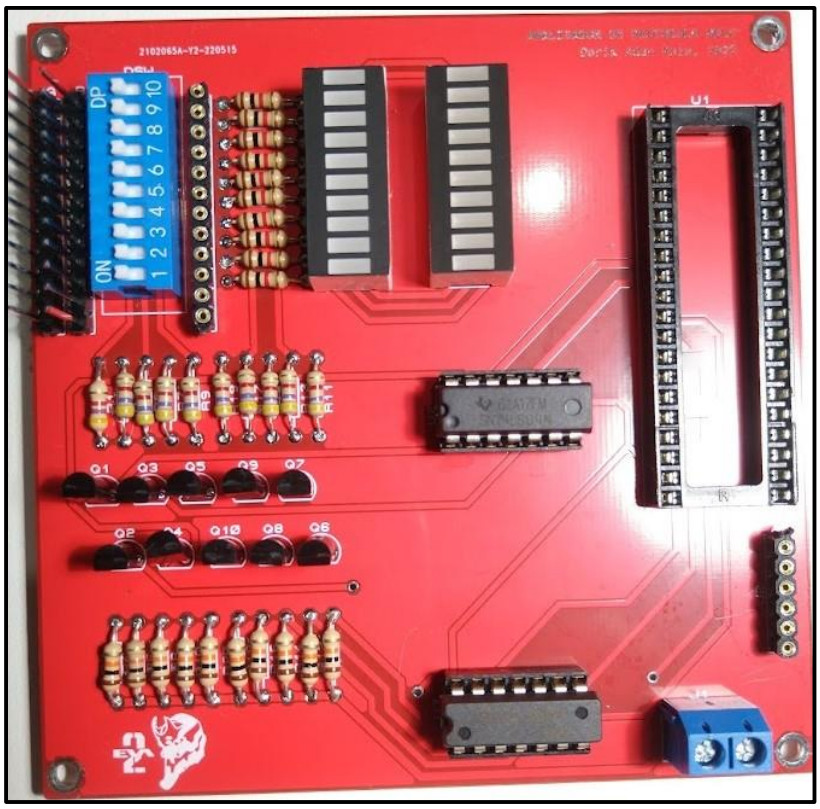
3.2 MONTAGEM DOS COMPONENTES NA PCB E TESTES DE BANCADA

Nesta etapa, foram realizadas as soldagens dos componentes e soquetes na PCB, assim como a fixação dos módulos *Blue Pill*, transceptor HC-05, circuitos integrados e barras de leds.

Os soquetes foram concebidos para facilitar a substituição dos circuitos integrados em caso de queima, sem a necessidade de dessoldagem.

Em seguida, os componentes foram fixados em seus respectivos soquetes, e a fonte de alimentação foi parafusada no borne de alimentação. A Figura 18 exibe o trabalho de soldagem e montagens dos componentes essenciais a placa condicionadora de sinais RS-232/TTL, ainda sem a *Blue Pill* e o módulo HC-05 afixados.

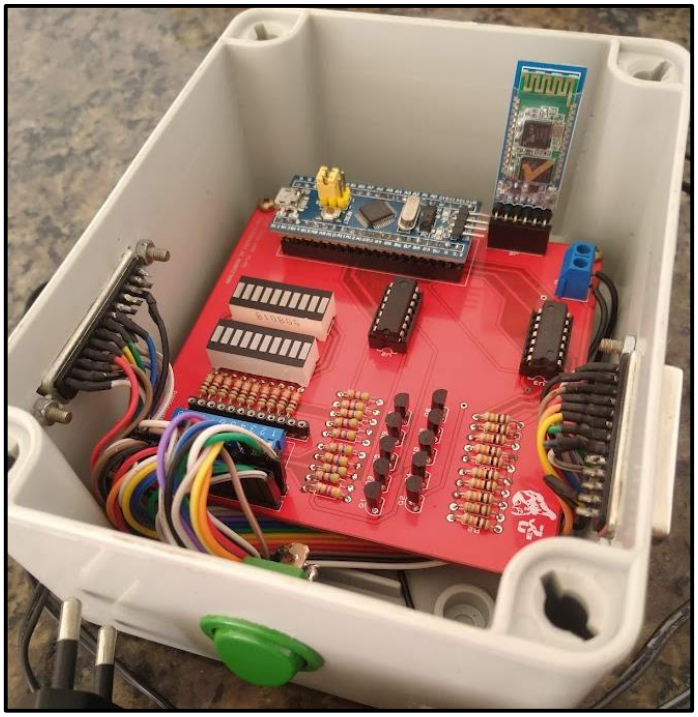
Figura 18 – Soldagem dos componentes, soquetes e fixação de módulos na PCB.



Fonte: o próprio autor.

Em seguida à soldagem dos componentes eletrônicos, foi instalada a placa em uma caixa plástica para receber as entradas por meio de conectores DB-25, suportar e proteger a placa, mecânica e eletricamente. Foram realizados furos na caixa para passagem de parafusos de fixação da placa e para posicionar os conectores de entrada. A Figura 19 apresenta a montagem da placa na caixa plástica, com os conectores DB-25 fêmea e botão liga-desliga montados.

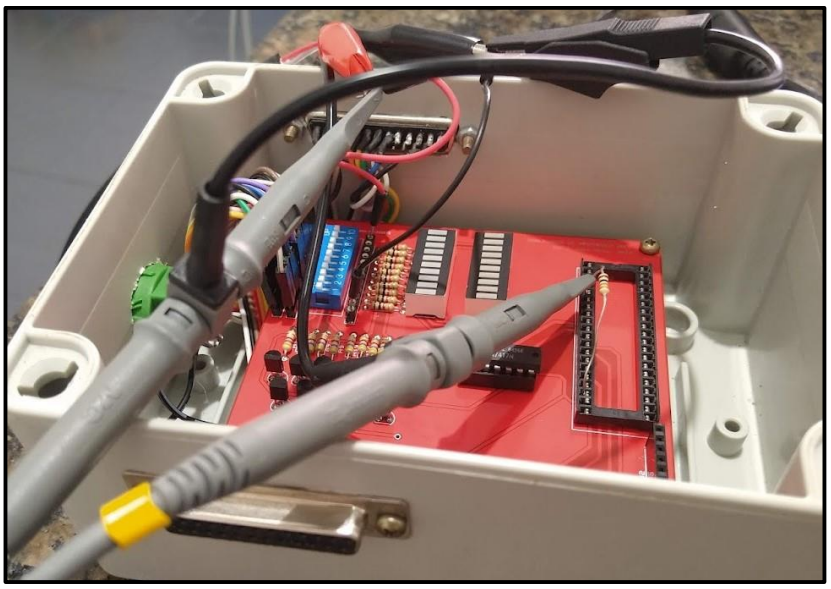
Figura 19 – Placa montada na caixa com conectores de entrada e botão liga-desliga.



Fonte: o próprio autor.

E por fim, testes na placa foram feitos com o auxílio do equipamento *NI VirtualBench All-in-One* como mostram a Figura 20 e a Figura 21. A Figura 22, demonstra a emulação de uma onda quadrada com *duty cicle* de 50% e amplitude de 20 V (gráfico em amarelo) e sua captura dentro da faixa de tensão TTL (gráfico em vermelho). Para emulação do *pull-up* interno do microcontrolador, foi conectado um resistor de 10 kΩ.

Figura 20 – Testes funcionais entre a entrada RS-232 e a saída TTL.

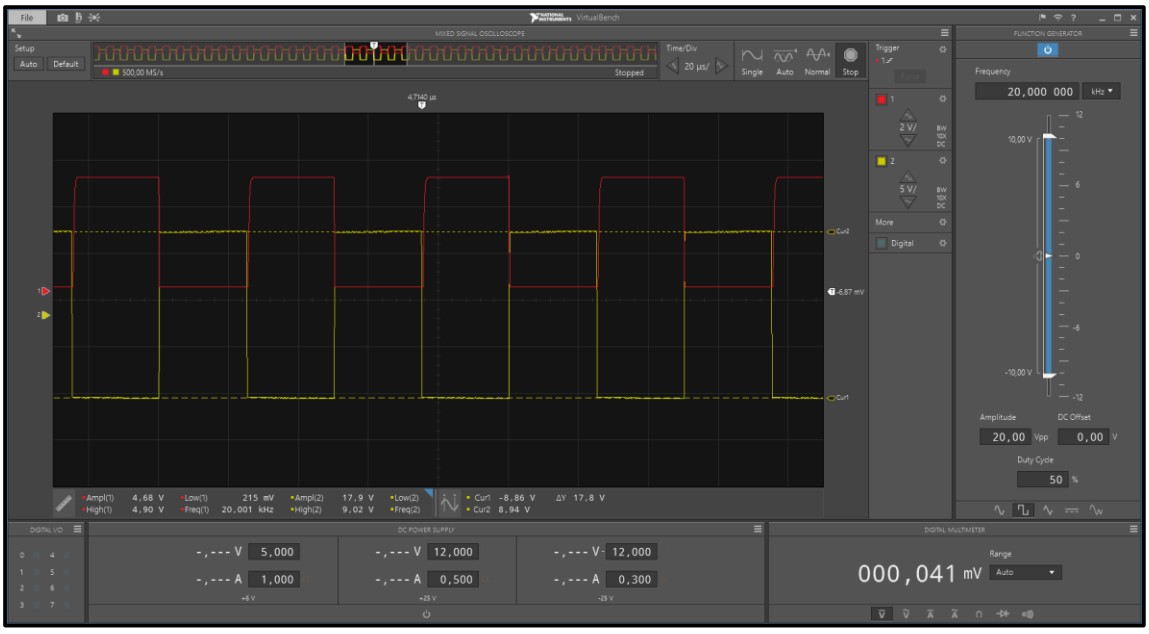


Fonte: o próprio autor.

Figura 21 – Uso do *NI VirtualBench All-in-One* para testes funcionais do protótipo.

Fonte: o próprio autor.

Figura 22 – Emulação de sinal de entrada RS-232 e captura da saída convertida para TTL.



Fonte: o próprio autor.

3.3 CAPTURA DE SINAIS E TRATAMENTO DOS QUADROS HDLC NO STM32

Nesta etapa, foi elaborado o programa responsável por receber os quadros HDLC pela interface DB-25, realizar a separação dos campos do quadro e verificar o FCS. Ainda nesta etapa, os dados tratados são organizados no formato JSON em uma estrutura chave e valor, para posterior envio pelo transceptor HC-05 ao dispositivo móvel.

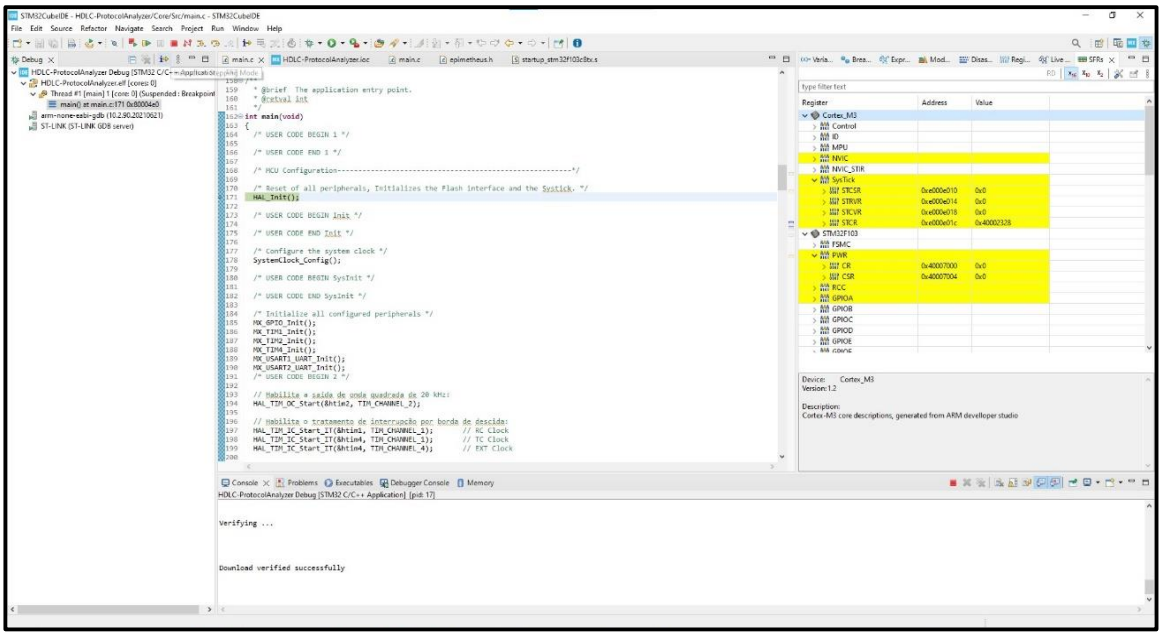
O *STM32CubeIDE* é uma ferramenta de desenvolvimento integrada completa, que faz parte do ecossistema de software *STM32Cube*. É uma plataforma de desenvolvimento C/C++

avançada com configuração de periféricos, geração de código, compilação de código e recursos de depuração para microcontroladores e microprocessadores STM32. Ele é baseado na estrutura Eclipse®/CDT™ e na cadeia de ferramentas *GNU C Compiler* (GCC) para o desenvolvimento e *GNU Debuger* (GDB) para a depuração. Ele permite a integração das centenas de *plugins* existentes que complementam os recursos do Eclipse® IDE.

Esta ferramenta integra as funcionalidades de configuração e criação de projetos do STM32 do *STM32CubeMX* para oferecer uma experiência de ferramenta completa e economizar tempo de instalação e desenvolvimento. Após a seleção de um microcontrolador ou microprocessador STM32 vazio, é realizada a pré-configuração a partir da seleção de uma placa ou da seleção de um exemplo, então o projeto é criado e o código de inicialização gerado. A qualquer momento durante o desenvolvimento, o usuário pode retornar à inicialização e configuração dos periféricos ou *middleware* (programa que está sendo gerado para o microcontrolador) e regenerar o código de inicialização sem impacto no código do usuário.

O *STM32CubeIDE* inclui analisadores de compilação e pilha que fornecem informações úteis sobre o status do projeto e os requisitos de memória. Esta IDE também inclui recursos de depuração padrão e avançados, incluindo visualizações de registros de núcleo de CPU, memórias e registros de periféricos, bem como observação variável ao vivo, interface *Serial Wire Viewer* ou analisador de falhas. A Figura 23 apresenta os principais dados exibidos na perspectiva de *debug*, durante a depuração de parte do código.

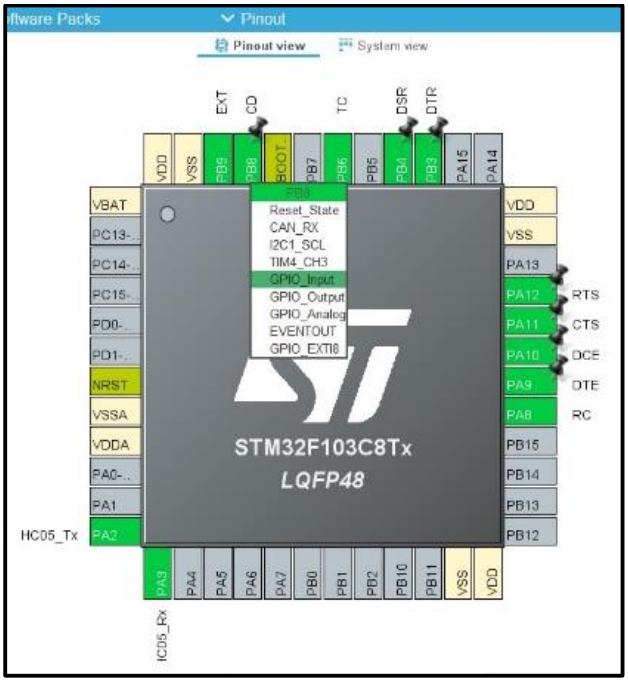
Figura 23 – Demonstração da IDE em perspectiva de depuração do *STM32CubeIDE*.



Fonte: o próprio autor.

Para constatar o estado lógico dos pinos de sinais de comunicação RS-232, os pinos CTS, RTS, DSR, DTR e CD são configurados como entradas digitais (GPIO_Input), conforme a Figura 24. Além disso, os pinos de sinais de temporização (RC, TC e EXT CLK) são configurados como entradas de captura (Input Capture) para canais de temporizadores (*timers*).

Figura 24 – Configuração de pinos de entrada no STM32CubeIDE.



Fonte: o próprio autor.

Dois periféricos são usados no projeto, a USART 1 e a USART 2, onde ambos são configurados em modo assíncrono. Na USART 1 configura-se um *baudrate* inicial de 9600 bits/s, que pode ser modificado depois da captura dos sinais de temporização. Por meio da USART 1, especificamente do pino DCE (PA10), os quadros serão recebidos e tratados. Na USART 2, o *baudrate* é definido a 115200 bits/s, sendo esta USART a responsável por transferir os dados recebidos e tratados no transceptor HC-05.

Para realizar o tratamento de quadros HDLC, controladores de comunicação HDLC são largamente implementados através de soluções de circuito integrado de aplicação específica (do inglês, *Application-specific integrated circuit* ou ASIC). Esses controladores são projetados utilizando ferramentas de descrição de *hardware*, inicialmente em dispositivos *Field Programmable Gate Array* (FPGA), já que técnicas de processamento de dados em paralelo, período de processamento de instrução e arranjo e acesso à memória são muito mais eficientes em ASICs do que em programas escritos em linguagens de baixo ou médio nível para microcontroladores.

Sobre o assunto, Sridevi e Reddy (2012) comentam que

> a programação de software de procedimentos HDLC é flexível e pode ser usada em muitas aplicações HDLC diferentes por simples modificação. No entanto, os programas consomem muito dos recursos do processador e do tempo de execução. [...] Ao adotar a tecnologia de processamento de hardware, os dispositivos FPGA podem ser programados repetidamente. Portanto, para uma implementação mais rápida, vários pesquisadores projetaram e desenvolveram controladores HDLC de acordo com os requisitos do sistema básico de comunicação usando FPGA. (SRIDEVI; REDDY, 2012, p. 1, tradução nossa).

Ainda sobre o assunto, Høyer e Kvamme (2004) afirmam que

> é possível trabalhar com bits únicos em processadores de uso geral, mas a CPU normalmente precisa usar várias instruções para lidar com cada bit. [...] quando a manipulação de *flag* e bit *stuffing* é realizada por software, há sérios problemas para escrever código eficiente. [...] também é o caso quando *flag* e bit *stuffing* são implementados e consomem uma parte significativa da capacidade da CPU (HØYER; KVAMME, 2004, p. 2, tradução nossa).

Entretanto, visando o menor custo de implementação, sugere-se que a solução de tratamento dos quadros HDLC seja implementada diretamente pelo elemento processador do protótipo, o microcontrolador STM32F103C8T6. Nesse sentido, o trabalho de Høyer e Kvamme (2004) proporciona a elaboração de um programa que interprete uma cadeia de bits recebidos pela interface de entrada e, por meio de tabelas, caracterize essa cadeia como um sinal de *flag*, *abort* ou bit a ser inserido ou removido. Tal solução pode ser usada tanto para transmissão quanto para recepção, porém este protótipo resume-se em escopo a apenas receber e interpretar dados. Dessa forma, o programa será elaborado de modo a receber uma entrada de cadeia de bits, chamada de *input buffer*, identificar uma determinada sequência previamente mapeada, que caracteriza o tipo padrão de caractere especial, e fornecer uma saída equivalente ao dado originalmente enviado pelo transmissor antes do processo de inserção de bits, chamada de *output buffer*. Os autores também informam que "O receptor HDLC buscará os dados preenchidos com bits do *input buffer*, descompactará os dados e armazenará os dados no *output buffer*. O receptor também indicará quando detectar uma sequência *FLAG* ou *ABORT*" (HØYER; KVAMME, 2004, p. 16, tradução nossa).

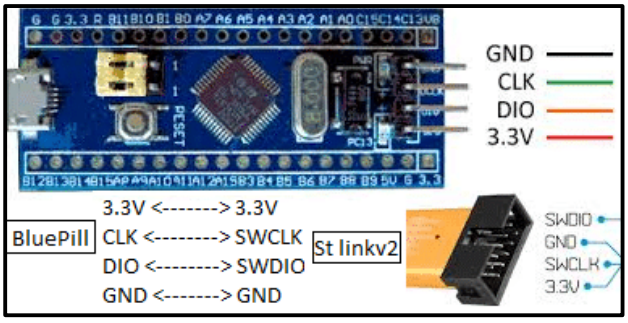
Primeiramente, um *byte* do *input buffer* é avaliado por vez, sendo então partido entre os quatro bits mais significativos (*nibble* do MSB) e os quatro bits menos significativos (*nibble* do LSB). Estes dois grupos são avaliados para reconhecer um padrão que indicará se a sequência representa dados, *flag* ou *abort*. Após esse reconhecimento, duas tabelas são consultadas para

identificar quantos bits serão removidos do *input buffer* e se serão transferidos ao *output buffer* ou serão identificados como caracteres de controle. O número de ‘1’ lógicos consecutivos então é consultado, indicando assim um determinado padrão. O Apêndice B apresenta em detalhes o diagrama do programa interpretador de quadros HDLC e o Apêndice C descreve o código do projeto a ser importado para compilação.

Ao compilar o programa, é necessário criar um arquivo binário de extensão .hex que possa ser carregado na memória da Blue Pill. Para configurar a geração desse arquivo, acessa-se no STM32CubeIDE o menu “*Project -> Properties -> C/C++ Build: Settings -> Tools Settings* : *MCU Post build outputs*” e marca-se a opção “*Convert to Intel Hex file (-O ihex)*”. Essa configuração permite que o artefato .hex possa ser encontrado no diretório *debug* ao compilar o projeto sem erros.

Para a gravação do programa na *Blue Pill*, é necessário conectar o gravador ST-Link v2 à *Blue Pill* conforme a Figura 25. A interface USB do gravador então é conectada ao computador e cabos do tipo *jumper* podem ser usados para conectar os terminais do gravador e da *Blue Pill*. Em seguida, é necessário carregar o arquivo por meio do programa STM32CubeProgrammer e, por fim, realizar a gravação.

Figura 25 – Configuração conexão do gravador ST-Link v2 à *Blue Pill*.



Fonte: (EMBARCADOS, 2019).

3.4 PROGRAMA DO ANDROID STUDIO

O *Android Studio* é um Ambiente de Desenvolvimento Integrado para a criação de aplicativos de uso exclusivo do sistema operacional *Android*. Essa ferramenta permite uma programação que abrange diferentes modelos de *smartphones* ou *tablets*, sendo adaptável para diversas especificidades, como tamanho de tela e versionamento do sistema operacional (DIMARZIO, 2017, p. 32). Como base para a codificação a ser utilizada nesta pesquisa, foi utilizado um projeto de código aberto desenvolvido especificamente para habilitar uma conexão

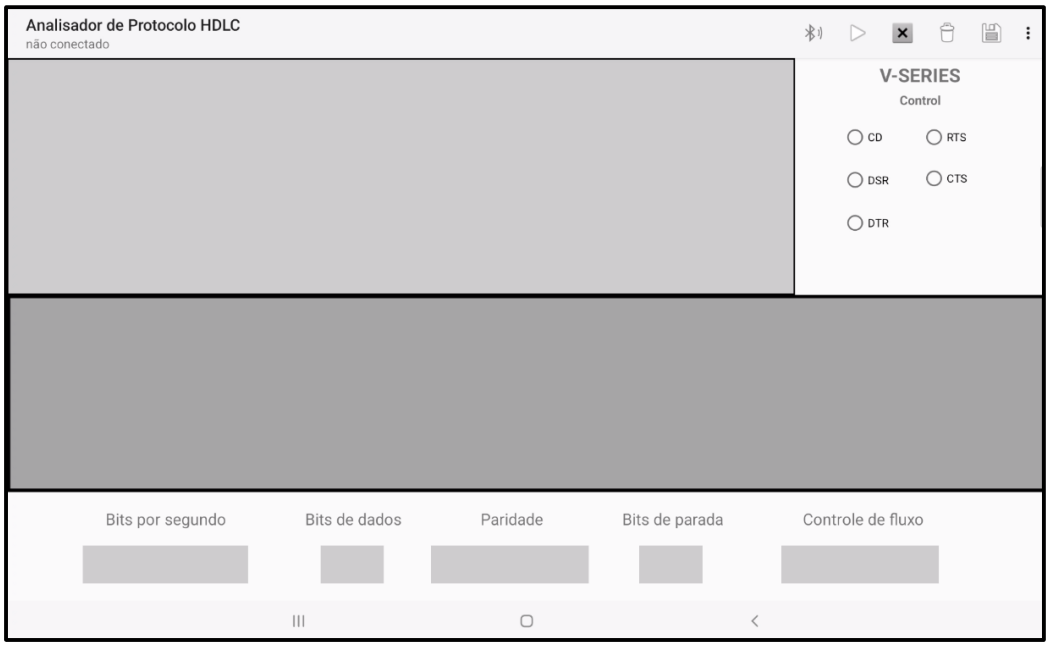
do tipo *bluetooth* entre o STM32 e o *Android*, utilizando o módulo HC-05 para possibilitar essa troca de informações.

Os códigos do Apêndice D apresentam as quatro classes principais utilizadas no código desenvolvido:

a) *BluetoothChatFragment.java*, que controla a conexão *bluetooth* iniciada entre os aparelhos, assim como habilita os botões para o início, a interrupção e o salvamento dos dados a serem recebidos;
b) *BluetoothChatService.java*, que inicia ou finaliza a conexão *bluetooth*, também gerenciando se essa está ativa ou inativa;
c) *DeviceListActivity.java*, que lista os possíveis aparelhos pareáveis via *bluetooth*, adicionando novos aparelhos encontrados ou mantendo aqueles que foram pareados anteriormente; e
d) *MainActivity.java*, que é a classe principal e responsável pela execução do aplicativo como um todo.

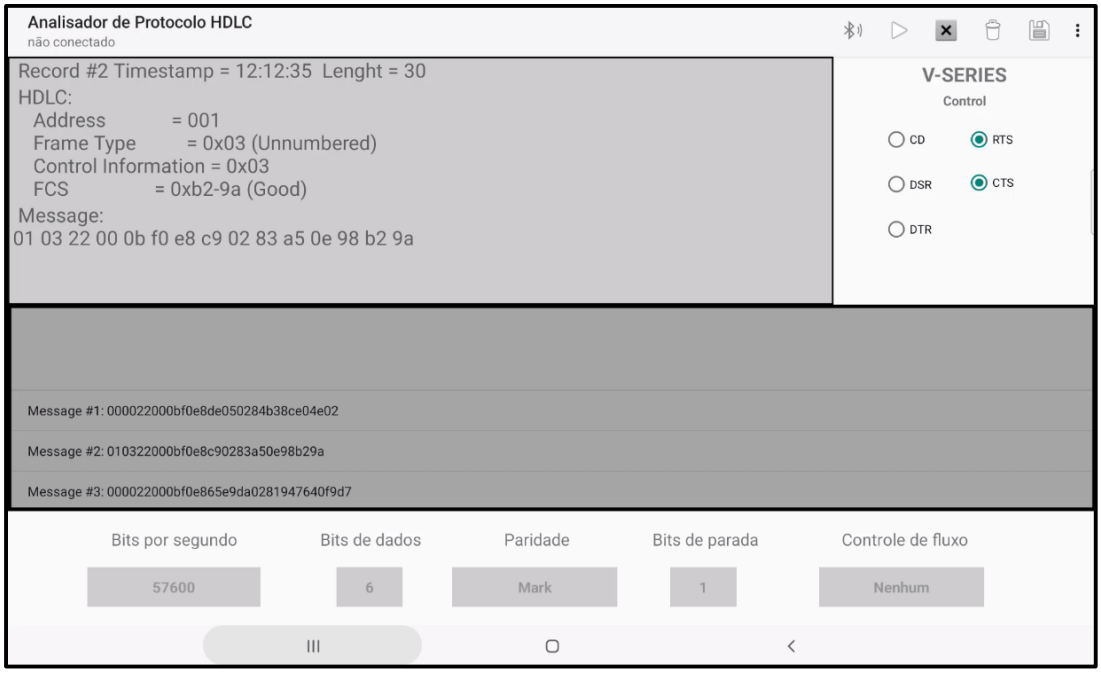
Outras classes foram desenvolvidas especificamente para o *layout* utilizado no projeto. A Figura 26 ilustra a interface do usuário formatada para um *tablet* Samsung Galaxy Tab S7+, porém o *Android Studio* permite que a visualização ocorra em diferentes modelos através de uma ferramenta que proporciona responsividade ao tamanho da tela desejada. A Figura 27 ilustra essa interface preenchida com os dados recebidos.

Figura 26 – Interface de Usuário do *Android Studio*.



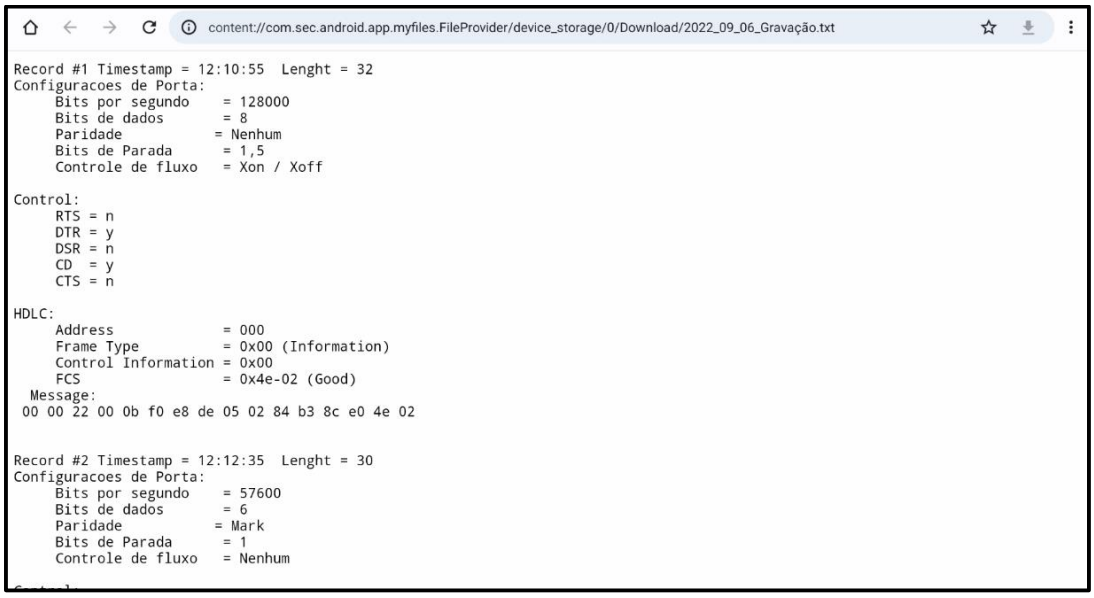
Fonte: o próprio autor.

Figura 27 – Interface de Usuário do *Android Studio* com os dados recebidos.



Fonte: o próprio autor.

Figura 28 – Arquivo .txt salvo com as informações recebidas.



Fonte: o próprio autor.

Os dados recebidos são interpretados através de campos de um arquivo do tipo JSON com os principais campos esperados em um analisador de protocolo HDLC. Uma vez que o recebimento de dados seja interrompido, é possível que o usuário salve um arquivo do tipo TXT para armazenar essas informações, conforme a Figura 28.

4 RESULTADOS OBTIDOS

Este capítulo está separado em quatro seções onde são mostrados os resultados referentes aos testes de tensões de operação das saídas da placa equalizadora, a verificação da captura dos sinais de controle de fluxo e de temporização RS-232, tratamento e envio do conteúdo de quadros HDLC e, por fim, a avaliação do código que implementa a técnica de remoção de bits dos quadros recebidos.

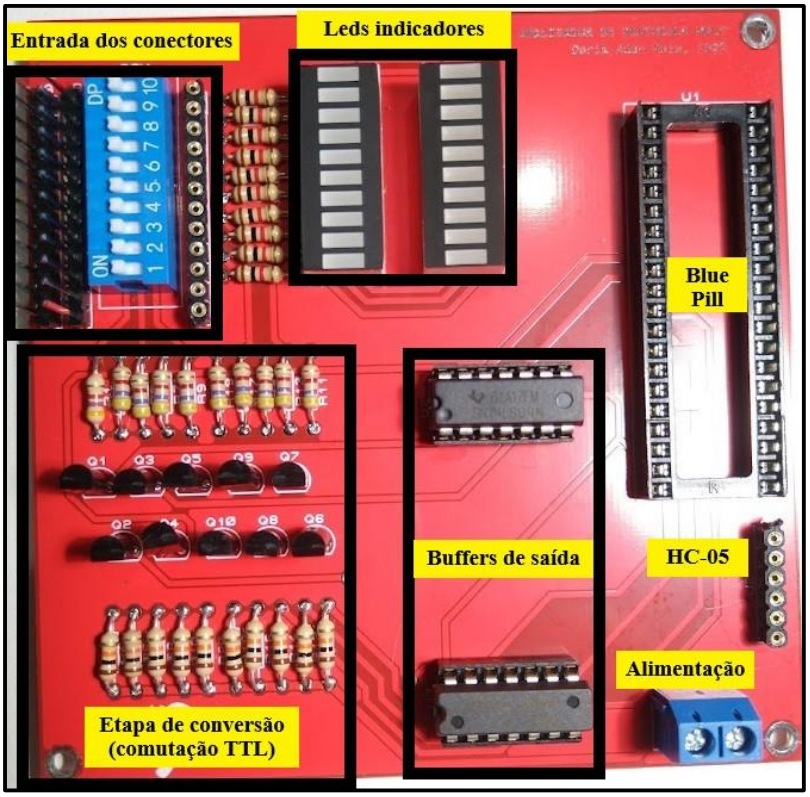
4.1 OPERAÇÃO DA PLACA EQUALIZADORA

Nesta seção são apresentadas as partes que compõem a placa equalizadora RS-232 para TTL do protótipo desenvolvido e os testes realizados ao expor este a um ambiente de campo, recebendo os estímulos da camada física por meio dos conectores DB-25. Inicialmente, são medidas as tensões elétricas e as formas de onda de cada pinagem na entrada da placa equalizadora. Em seguida, são realizadas as mesmas medidas em cada pino que vai à entrada das GPIO da placa *Blue Pill*.

Para realizar a aquisição de sinais e dados da camada de enlace, foi formulada uma placa que promove a interface com os dispositivos de comunicação através de *headers* ligados por cabos *jumper* aos concectores DB-25 do protótipo. Como evidenciado pela Figura 29, a placa possui receptáculos de acesso para tomadas de medições de tensão e chaves *dip switch* para a seleção de pinos de entrada. Em seguida, *leds* indicam os estados lógicos RS-232 de cada pino avaliado. A próxima etapa ocorre com a conversão através de comutação TTL, por intermédio de resistores, transistores e diodos de proteção. Por conseguinte, *buffers* TTL regulam as correntes de saída em detrimento da impedância de entrada das portas GPIO do microcontrolador. Ainda há também os soquetes para acoplamento do dispositivo *Blue Pill* e do transceptor HC-05, além do borne de alimentação, onde a fonte reguladora de tensão é conectada.

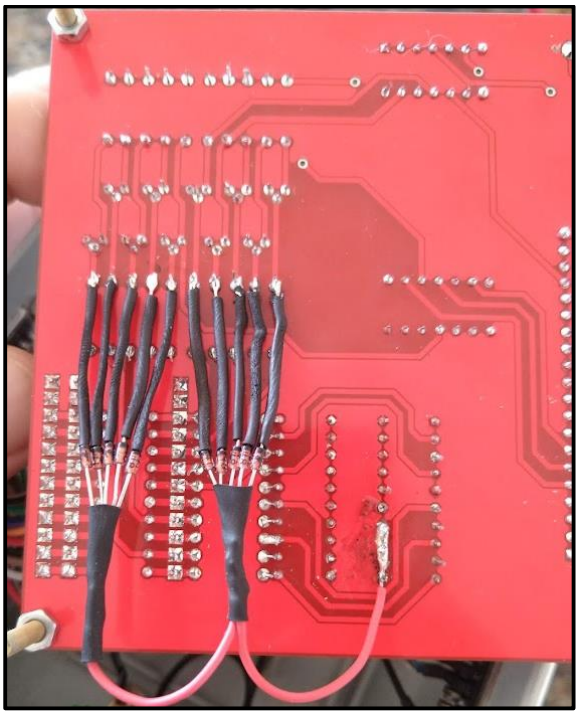
Durante o processo de concepção da placa equalizadora, não havia a necessidade do uso de diodos de proteção da junção base-emissor para os transistores 2N2222. Conforme apresentado pela Figura 30, esses diodos foram soldados posteriormente pela camada inferior da placa, devidamente isolados. Esses diodos previnem que a junção base-emissor de cada transistor não sofra tensões superiores ao recomendado em seus *datasheets*. Essa tensão limite é de 6 volts. Essa adaptação foi realizada no esquemático.

Figura 29 – Partes componentes da placa equalizadora RS-232/TTL.



Fonte: o próprio autor.

Figura 30 – Adição de diodos de proteção da junção base-emissor do 2N2222.



Fonte: o próprio autor.

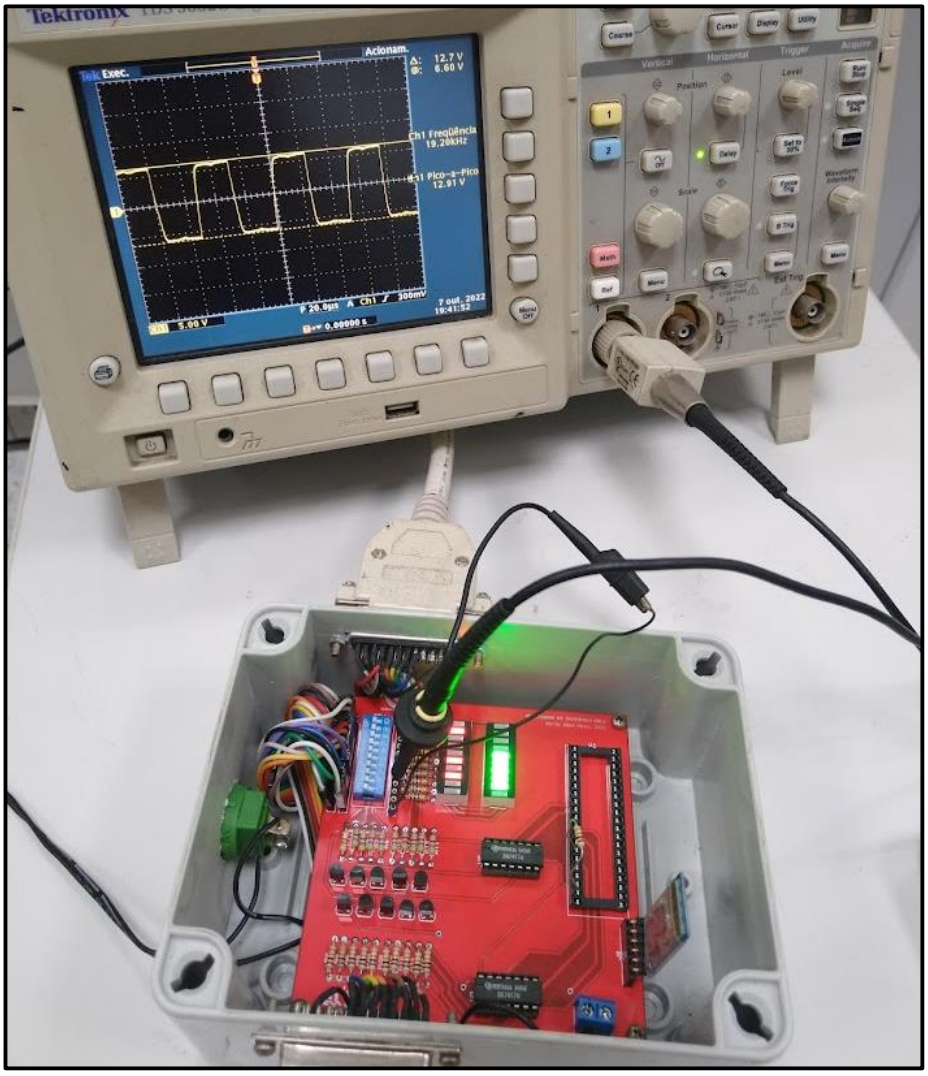
4.2 VERIFICAÇÃO DOS SINAIS RS-232

Nesta etapa são realizadas medições por meio de um osciloscópio que compara os sinais de entrada RS-232 e a resposta nos padrões de tensão TTL para uso nas GPIO da *Blue Pill*.

Na Figura 31, um cabo de linha de radiodeterminação é conectado a uma das entradas do protótipo e são avaliadas as características de frequência e os valores pico a pico, especificamente, do pino de sincronismo RC.

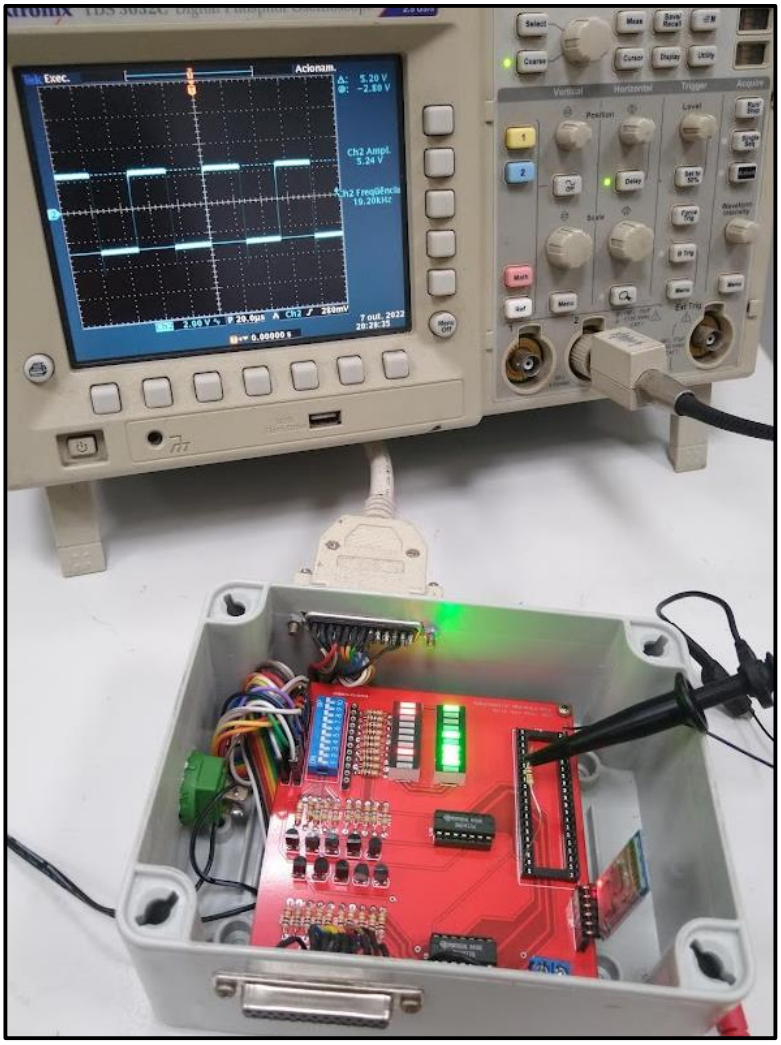
Na Figura 32, são avalidas as características de frequência e os valores pico a pico, do mesmo pino de sincronismo RC, entretanto, na saída do *buffer* que vai ao soquete da *Blue Pill*.

Figura 31 – Verificação dos sinais recebidos pela placa equalizadora.



Fonte: o próprio autor.

Figura 32 – Verificação dos sinais tratados pela placa equalizadora.



Fonte: o próprio autor.

A Tabela 2 apresenta a relação entre as pinagens e as medidas de cada pino de entrada e de saída tomadas nesta etapa. Os campos marcados com "-" são grandezas inaplicáveis à respectiva pinagem e tais leituras são correspondentes à apenas uma amostra de enlace de dados. O intuito desta etapa é averiguar se alguma das sinalizações não pode ser medida devido a limitações na construção da placa ou de algum componente eletrônico. Ainda é possível notar que as sinalizações de temporização e sincronismo (RC, TC e EXT) não têm qualquer prejuízo ou interferência que impeça a aquisição do *baudrate*.

Tabela 2 – Comparativo das medidas de entrada e saída da placa equalizadora RS-232.

Número do pino	Função	Entrada (Conectores DB-25)			Saída (Soquete da Blue Pill)		
		Frequência [kHz]	Amplitude [V]		Frequência [kHz]	Amplitude [V]	
			Min	Max		Min	Max
1	Protective ground / shield	-	0	0	-	0	0
2	Transmitted Data (DTE)	-	-6,1	6,5	-	0,1	5,2
3	Received Data (DCE)	-	-6,3	6,4	-	0,3	5,2
4	Request to Send (RTS)	-	-	-	-	-	-
5	Clear to Send (CTS)	-	-	8,3	-	-	5,1
6	DCE Ready (DSR)	-	-	6,8	-	-	5,2
7	Signal ground	-	-	-	-	-	-
8	Signal Detector (CD)	-	-	-	-	-	-
15	Transmitter Timing (TC)	19,200	-6,1	6,6	19,200	0,2	5,2
17	Receiver Timing (RC)	19,200	-6,1	6,5	19,200	0,2	5,2
20	DTE Ready (DTR)	-	-6,2	-	-	-	5,1
24	External Transm. Timing (EXT)	19,200	-6,3	6,5	19,200	0,2	5,2

Fonte: o próprio autor.

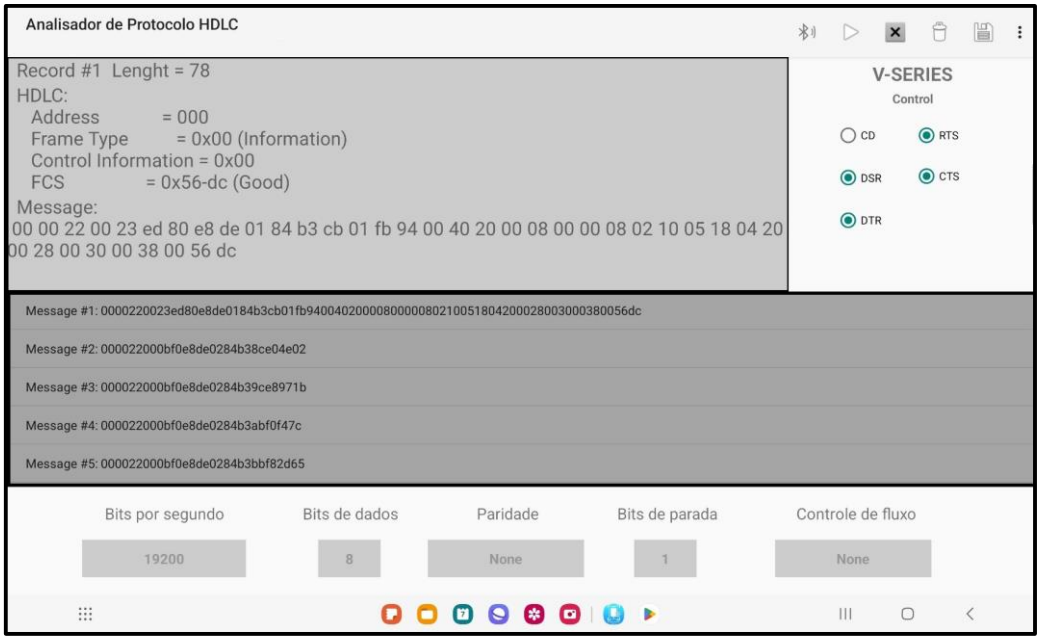
4.3 TRATAMENTO E ENVIO DOS QUADROS HDLC

Nesta etapa são verificados os resultados da leitura e da interpretação dos quadros HDLC enviados ao dispositivo móvel para inspeção pelo aplicativo *android*. Após a ligação de cabos e a alimentação do protótipo, o *led* indicador de estado do transceptor HC-05 começa a piscar, indicando que está disponível para pareamento. Com isso, o usuário pode abrir o aplicativo e clicar no botão de seleção de dispositivo *bluetooth*.

Nesse momento, o *led* indicador de estado do HC-05 passa a piscar mais lentamente, demonstrando que o dispositivo está pareado. A partir de então, o usuário deve clicar no botão de *play* e a classe responsável por receber os dados provenientes da *Blue Pill* organiza as informações de estados lógicos, pinos de sincronismo e quadros disponíveis em um arquivo no formato chave e valor JSON. Esse arquivo é lido e suas informações são apresentadas na tela, conforme a Figura 33, que apresenta o conteúdo dos dados tratados, os pinos de controle de fluxo, a taxa de transmissão em bits por segundo e a validação do campo FCS de cada quadro recebido.

Em cinza escuro, é possível visualizar uma fila de mensagens em formato de lista. À medida que mais quadros são recebidos, a fila de mensagens acumula mais itens e cada um dos objetos da lista é selecionável para visualizar seus dados em detalhes na área superior do aplicativo.

Figura 33 – Aplicativo recebendo e tratando os dados recebidos pela interface com HC-05.



Fonte: o próprio autor.

4.4 COMPARATIVO ENTRE ANALISADORES

Equipamentos analisadores de protocolos de enlace de dados têm sido cada vez menos comuns no mercado, devido principalmente a uma forte mudança na forma como dispositivos de telecomunicações para transmissões de longas distâncias são gerenciados. Geralmente, há uma tendência no uso de ativos de redes de computadores, tais como servidores e roteadores gerenciáveis por TCP/IP, que, além de cumprirem suas atividades fins, utilizam das camadas de redes e de transporte para monitorar e relatar seus enlaces de comunicação. Esses dispositivos contam com acesso remoto e interfaces *web* para alteração de configurações e emissão de relatórios e estatísticas.

No entanto, ainda é comum que empresas e órgãos que operam enlaces de comunicação abrangentes utilizem analisadores de protocolos da camada de enlace de dados, sendo a maior parte desses dispositivos fabricada por grandes empresas de instrumentação eletrônica. Ainda é possível encontrar à venda dispositivos como o *Agilent* J2300D *Wan Internet Advisor*, uma solução de analisador de protocolo que concentra um painel de seleção e inspeção de pinagem de interfaces seriais. Este dispositivo pode capturar e analisar dados de diversos padrões da camada física, além de um vasto leque de opções de protocolos de camada de enlace de dados. Sem dúvida, esses dispositivos são muito robustos, eficientes e concentram uma repleta gama de possibilidades para a análise de diferentes padrões de enlaces de dados.

Figura 34 – Oferta de analisador de protocolo HDLC no mercado internacional.

Fonte: https://www.ebay.com/p/663699190.

Quanto ao valor dos componentes utilizados para realizar a compatibilidade de níveis de tensão e montagem do protótipo, o Quadro 2 esclarece que são necessários R$ 311,19 para a confecção do protótipo frente aos R$ 388,85 (com frete de R$ 624,93) para a compra de um analisador no mercado internacional, conforme Figura 34.

Quadro 2 – Relação de materiais para confecção do protótipo.

Item	Descrição	Quantidade	Valor unitário	Subtotal
1	Caixa de derivação cbox-ob 150mm x 110mm x 70mm	1	R$ 43,60	R$ 43,60
2	Parafusos 3 mm x 30 mm	4	R$ 0,30	R$ 1,20
3	Porcas sextavadas 3 mm	12	R$ 0,10	R$ 1,20
4	Placa de circuito impresso do protótipo	1	R$ 79,60	R$ 79,60
5	Módulo *Blue Pill* (STM32F103C8T6)	1	R$ 45,00	R$ 45,00
6	Módulo Bluetooth RS232 HC-05.	1	R$ 48,00	R$ 48,00
7	Fonte alimentação de 5,0 volts a 1,0 ampère.	1	R$ 19,00	R$ 19,00
8	Circuitos integrados buffer 74LS07.	2	R$ 1,50	R$ 3,00
9	Barra gráfica de leds verde.	1	R$ 9,80	R$ 9,80
10	Barra gráfica de leds vermelha.	1	R$ 10,20	R$ 10,20
11	Transistores NPN 2N2222	10	R$ 0,50	R$ 5,00
12	Diodos 1N4148.	10	R$ 0,30	R$ 3,00
13	Resistores 1 kΩ (1/4W)	10	R$ 0,50	R$ 5,00
14	Resistores 4,7 kΩ (1/4W).	10	R$ 0,50	R$ 5,00
15	Resistores 10 kΩ (1/4W)	10	R$ 0,50	R$ 5,00
16	Conectores DB-25 fêmea soldáveis.	2	R$ 3,52	R$ 7,04

17	Módulo dip switch de 10 vias de 180 Graus.	1	R$ 3,57	R$ 3,57
18	Soquetes de 14 pinos.	2	R$ 0,44	R$ 0,88
19	Soquete de 40 pinos.	1	R$ 1,20	R$ 1,20
20	Borne de 2 polos.	1	R$ 0,80	R$ 0,80
21	Barras de pinos de 12 vias.	2	R$ 1,95	R$ 3,90
22	Barra de soquete de 6 vias.	1	R$ 0,70	R$ 0,70
23	Barra de soquete de 12 vias.	1	R$ 1,50	R$ 1,50
24	Cabos para conexões (cabos jumper).	1	R$ 8,00	R$ 8,00
			Total	**R$ 311,19**

Fonte: o próprio autor.

Entretanto, é necessário estabelecer categorias para comparação dos dois dispositivos diante dos valores supracitados, sendo possível afirmar que uma solução de sistema embarcado viável, utilizando materiais e métodos similares a esta pesquisa, pode oferecer uma proposta de confecção de um dispositivo analisador de protocolo com um custo menor do que uma solução industrial mais robusta e completa.

CONCLUSÃO

Esta pesquisa apresentou, no capítulo de referencial teórico, os principais fundamentos que embasam os requisitos e as tecnologias empregadas para a construção do protótipo, que visa testar a possibilidade de criar um dispositivo eletrônico para captura de sinais padrão RS-232 síncrono, portátil e de baixo custo, a fim de inspecionar indicadores de controle de fluxo de dados, leitura e decomposição de quadros da camada de enlace de dados.

Durante a concepção do projeto foram investigadas as plataformas mais destacadas de microcontroladores utilizadas recentemente para prototipagem de sistemas embarcados. A plataforma *Blue Pill*, que conta com o microcontrolador STM32F103C8T6, foi escolhida devido a seu baixo custo e versatilidade, além de tamanho reduzido e possibilidade de trabalhar com taxas de 72 MHz de *clock*. Para compatibilidade das tensões de operação entre a interface de entrada RS-232 e o microcontrolador, foi desenvolvida uma placa equalizadora de RS-232 para TTL. Esta placa também serve de base para o acomplamento de conectores, chaves seletoras, leds indicadores e soquete para o dispositivo de comunicação sem fio. Na etapa de aquisição e processamento de quadros HDLC foi desenvolvido um programa *firmware* para o STM32. Este programa implementa uma técnica eficiente de remoção de bits por software e detecta por meio do campo FCS se o quadro recebido é íntegro. Os dados processados são então enviados a um aplicativo em dispositivo móvel. Este último, foi criado para apresentar os dados ao usuário e permitir a gravação de arquivos de relatório com os dados recebidos.

No capítulo de resultados obtidos, é possível constatar a viabilidade do uso do protótipo ao fim a que se destina, pois demonstrou ser capaz de obter os dados necessários para inspeção dos sinais de enlace de dados das tecnologias utilizadas em linhas de radiodeterminação. Outrossim, apesar de garantir a operação em um nicho específico, o protótipo mostra-se mais barato que as soluções comerciais disponíveis no mercado, além de ser mais portátil e usar de componentes discretos que facilitam sua confecção e sua manutenção.

Sugere-se, como melhoria, implementar o tratamento de dados para o protocolo ASTERIX pelo aplicativo *Android*, exibindo assim campos do datagrama, o que pode ser interessante na análise de dados radar brutos. Além disto, pode-se avaliar a possibilidade de utilizar componentes eletrônicos menores, como SMD, com o intuito de reduzir o custo de fabricação e o tamanho do protótipo.

Com base nos resultados obtidos, verifica-se então que o protótipo de baixo custo é capaz de capturar sinais de padrão RS-232, validar os dados obtidos e apresentá-los ao usuário através de um aplicativo para dispositivos móveis sem fio.

REFERÊNCIAS

ACERVO LIMA. **Como a comunicação acontece usando o modelo OSI**. 2021. Disponivel em: <https://acervolima.com/como-a-comunicacao-acontece-usando-o-modelo-osi/>. Acesso em: 20 fev. 2022.

BOYLESTAD, R. L.; NASHELSKY, L. **Dispositivos eletrônicos e teoria de circuitos**. 11. ed. São Paulo: Pearson Education do Brasil, 2013.

BRASIL. Ministério da Defesa. **CINDACTA IV conclui a atualização da sua infraestrutura de enlaces de voz e dados**. 2019. Disponivel em: <https://www.decea.mil.br/?i=midia-e-informacao&p=pg_noticia&materia=cindacta-iv-conclui-a-atualizacao-da-sua-infraestrutura-de-enlaces-de-voz-e-dados>. Acesso em: 22 jun. 2021.

BROWN, G. **Descobrindo o Microcontrolador STM32**. Tradução de Amanda F. V. Martins; Felippe Gaede, *et al.* Espírito Santo: Universidade Federal do Espírito Santo, 2018. Disponivel em: <https://docplayer.com.br/157622913-Descobrindo-o-microcontrolador-stm32.html>. Acesso em: 20 fev. 2022.

CAVALCANTE, P. C. S. **Teoria da Informação e Codificação:** Notas de aula. Manaus: [s.n.], 2016. 199 p. Disponivel em: <https://docs.google.com/viewer?a=v&pid=sites&srcid=dWVhLmVkdS5icnxwY3Njc2l0ZXxneDo2ZWZkMzg0NTEyNzQ4ZmU5>. Acesso em: 25 fev. 2022.

CIPELLI, A. M. V.; MARKUS, O.; SANDRINI, W. **Teoria e Desenvolvimento de Projetos de Circuitos Eletrônicos**. 23. ed. São Paulo: Érica, 2007.

DIMARZIO, J. F. **Beginning Android Programming with Android Studio**. 4. ed. Indianapolis: John Wiley & Sons, Inc., v. 1, 2017.

DINIZ, P. S. R.; SILVA, E. A. B. D.; NETTO, S. L. **Processamento digital de sinais:** projeto e análise de sistemas. Porto Alegre: Bookman, 2014. 976 p.

EMBARCADOS. STM32CubeIDE: Primeiros passos e CMSIS Core com GPIO. **Embarcados**. 2019. Disponivel em: <https://embarcados.com.br/stm32cubeide-primeiros-passos-e-cmsis-core-com-gpio/>. Acesso em: 26 jul. 2022.

EUROCONTROL. **EUROCONTROL Specification for Surveillance Data Exchange - Part 1: All Purpose Structured EUROCONTROL Surveillance Information Exchange (ASTERIX)**. EUROCONTROL. [S.l.]. 2021.

FEY, A. F.; GAUER, R. R. **Fundamentos de Telecomunicações e Comunicação de Dados.** 2. ed. Caxias do Sul: [s.n.], 2020. 266 p. Acesso em: 31 jan. 2022.

GRODZIK, R. **RS232 Cookbook**. [s.l.]: [s.n.], [2012]. Disponivel em: <https://www.amazon.com/dp/B007BV0OX4>. Acesso em: 15 fev. 2022.

HØYER, F.; KVAMME, A. **A method and an apparatus for bit stuffing and a corresponding method and apparatus for bit de-stuffing**. WO2004056023A1, 1 jul. 2004.

Disponivel em: <https://patents.google.com/patent/WO2004056023A1/en>. Acesso em: 20 ago. 2022.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Information technology - Telecommunications and information exchange between systems - High-level data link control (HDLC) procedures**. ISO Copyright Office. Geneva. 2002.

LUGLI, A. B.; SANTOS, M. M. **Sistemas Fieldbus para automação industrial**. São Paulo: Érica, 2009. ISBN 978-85-365-0249-6.

OLIVEIRA, S. **Internet das Coisas com ESP8266, Arduino e Raspberry Pi**. São Paulo: Novatec, 2017. ISBN 978-85-7522-581-3.

OLIVEIRA, W. **Comunicação entre Módulos Bluetooth HC-05 e HC-06**. 2016. Disponivel em: <https://www.embarcados.com.br/modulos-bluetooth-hc-05-e-hc-06/>. Acesso em: 27 fev. 2022.

PINHEIRO, J. M. S. Projeto de redes. **O Modelo OSI**. 22 nov. 2004. Disponivel em: <https://www.projetoderedes.com.br/artigos/artigo_modelo_osi.php>. Acesso em: 07 jul. 2022.

SALCIUNAS, A. V. **Electronic Industries Association (EIA) RS-232-C Interface Standard**. Delran N. J.: McGraw-Hill, 1991. Disponivel em: <http://bitsavers.trailing-edge.com/pdf/datapro/communications_standards/2740_EIA_RS-232-C.pdf>. Acesso em: 10 fev. 2022.

SILVA, I. **Conheça a placa Blue Pill (STM32F103C8T6)**. 2020. Disponivel em: <https://www.embarcados.com.br/blue-pill-stm32f103c8t6/>. Acesso em: 26 fev. 2022.

SRIDEVI, M.; REDDY, P. S. Design And Implementation Of Hdlc Protocol On Fpga. **International Journal of Engineering Research and Applications (IJERA)**, Vol. 2, Set. 2012. Disponivel em: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.640.8584&rep=rep1&type=pdf>. Acesso em: 30 ago. 2022.

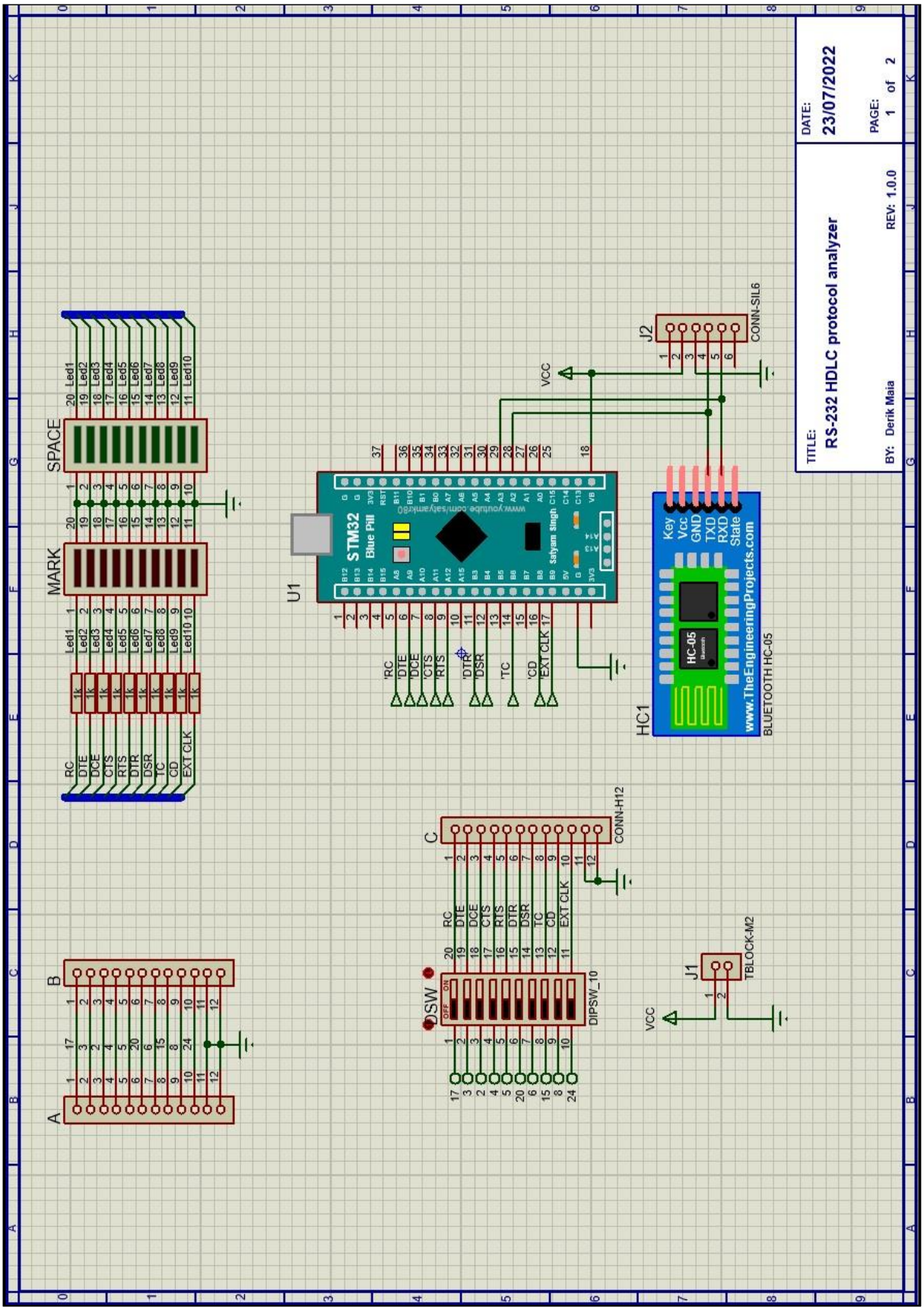
STMICROELECTRONICS. **Description of STM32F1 HAL and low-layer drivers**. 2020. Disponivel em: <https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf>. Acesso em: 20 jul. 2022.

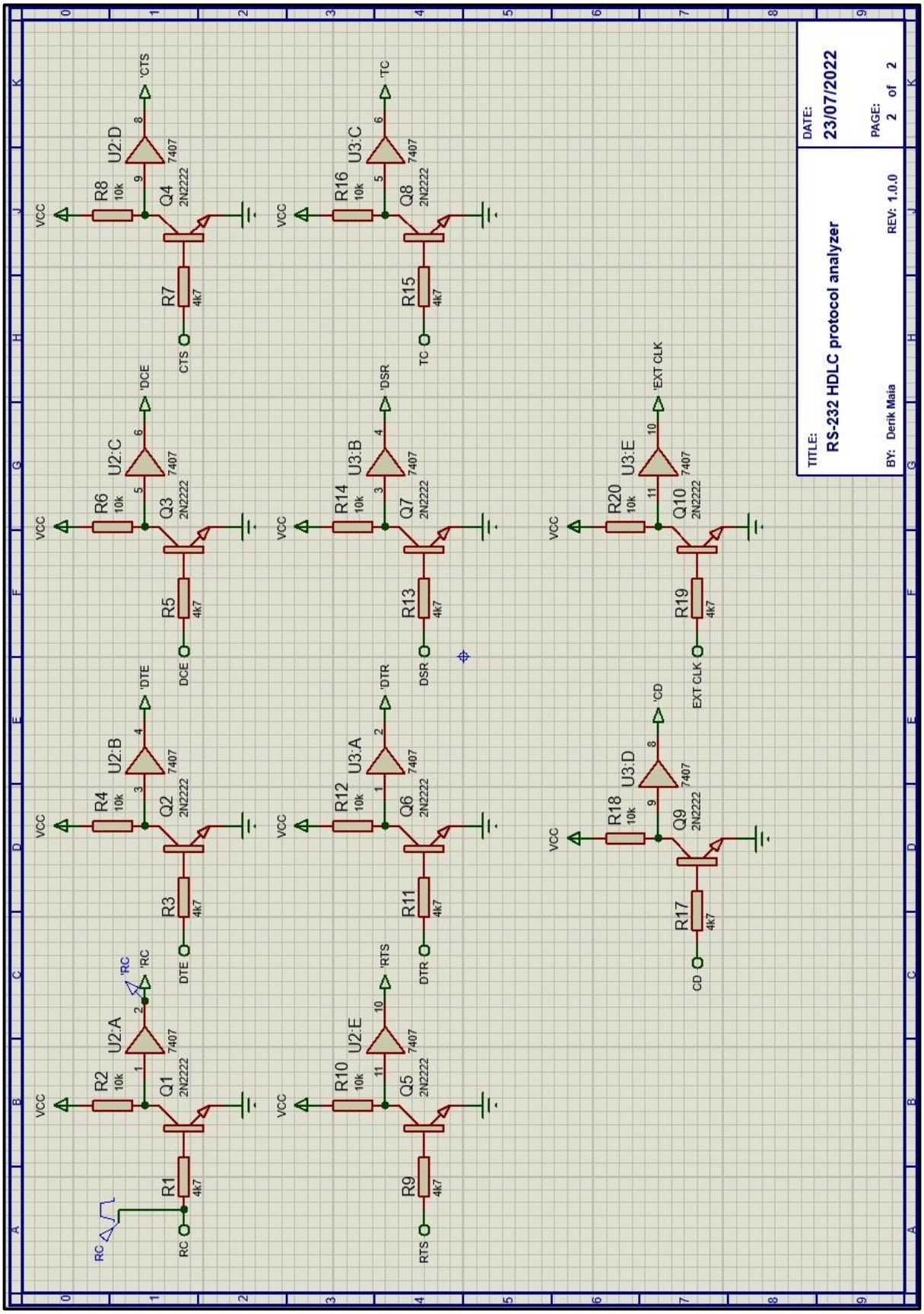
STRANGIO, C. **The RS-232 Standard**. Lexington: CAMI Research Inc., 1997. Disponivel em: <https://mil.ufl.edu/3744/docs/RS232_standard_files/RS232_standard.html>. Acesso em: 31 jan. 2022.

TANENBAUM, A. S.; WETHERALL, D. **Redes de computadores**. Tradução de Daniel Vieira. 5. ed. São Paulo: Pearson, 2011.

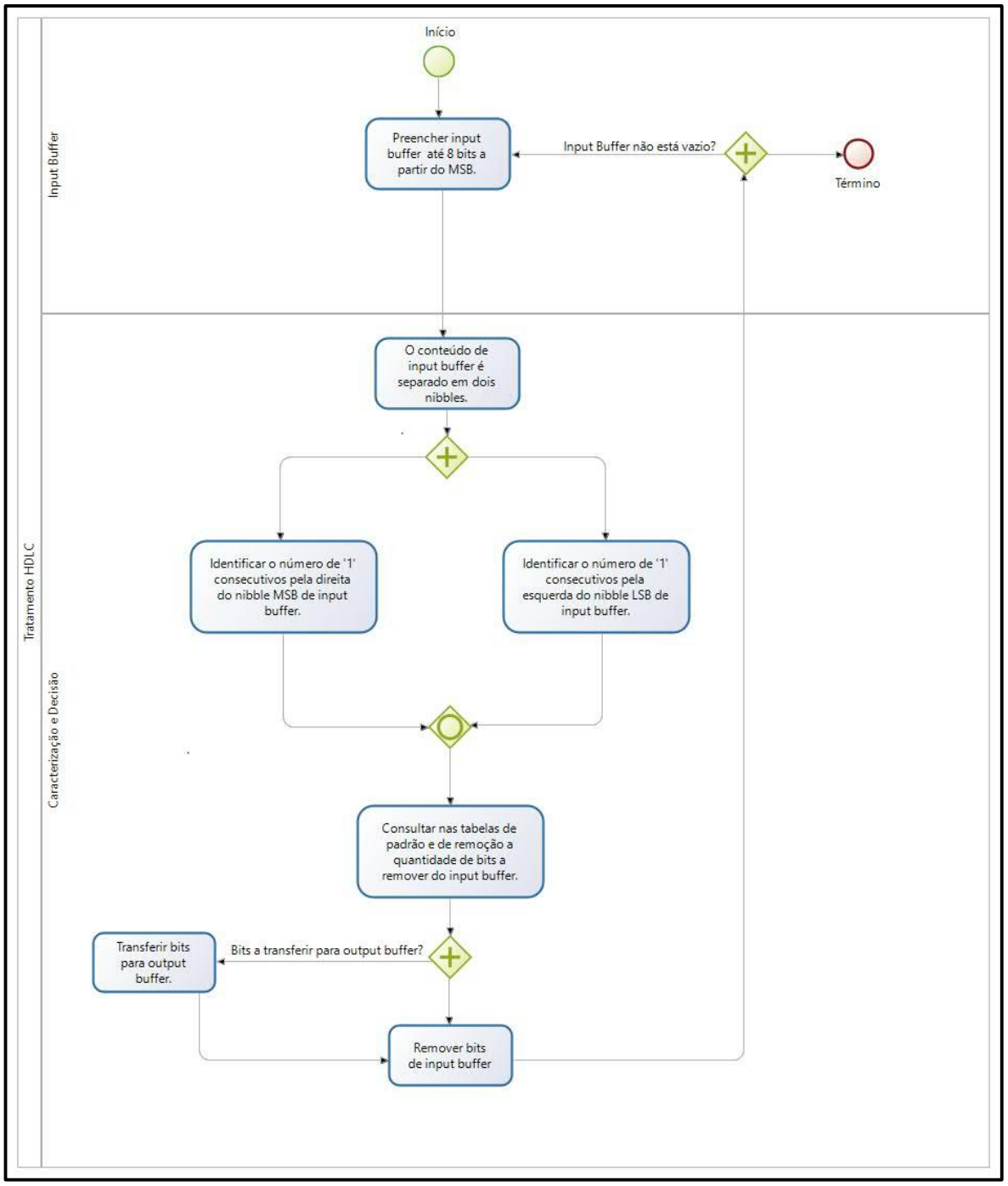
ZANCO, W. D. S. **Microcontroladores PIC16F628A/648A:** uma abordagem prática e objetiva. 2. ed. São Paulo: Érica, 2007. ISBN 978-85-365-0059-1.

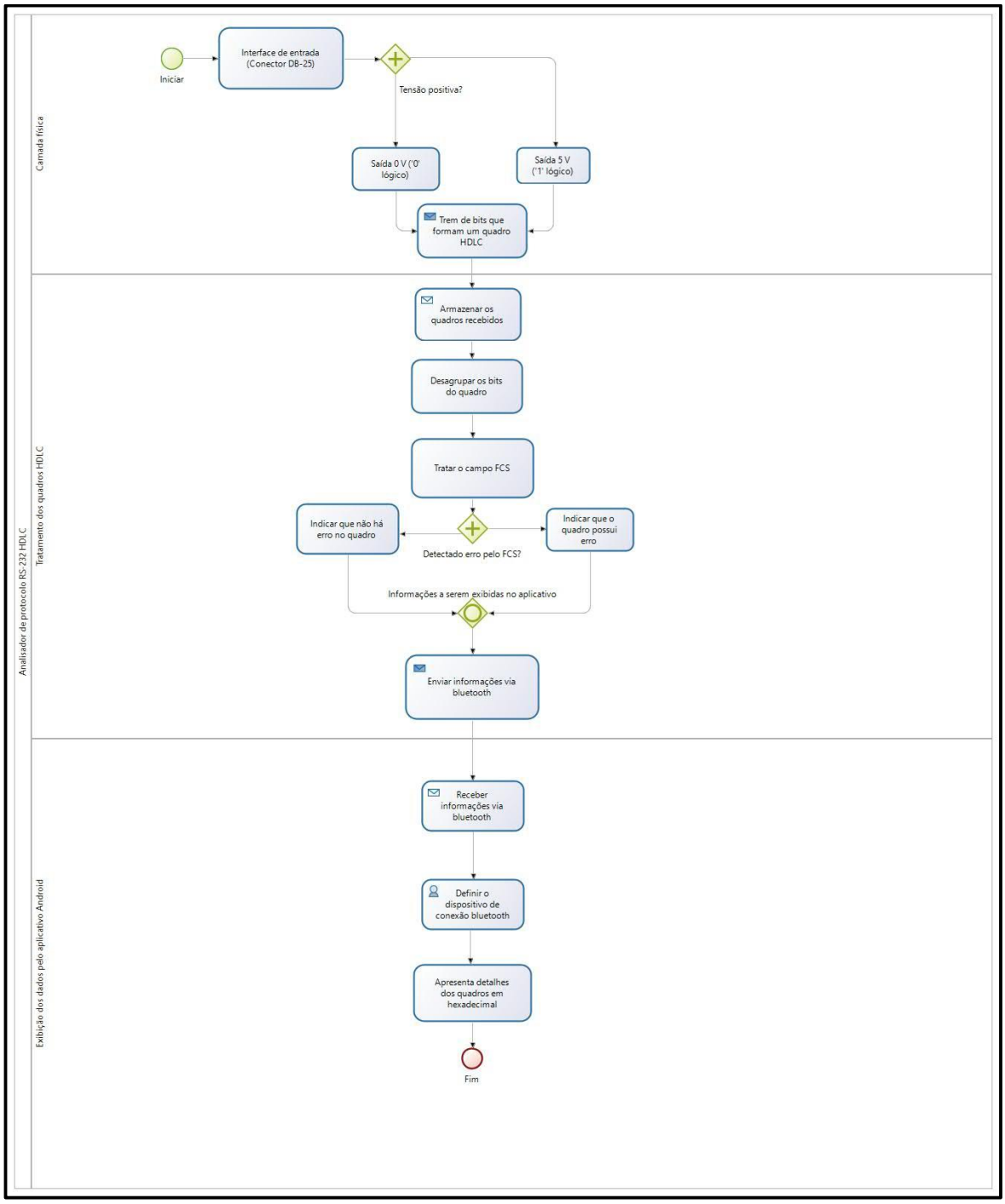
APÊNDICE A – ESQUEMÁTICOS DA PLACA EQUALIZADORA RS-232/TTL




TITLE:
RS-232 HDLC protocol analyzer
DATE:
23/07/2022
BY: Derik Maia
REV: 1.0.0
PAGE:
2 of 2
VCC
R1
4k7
R2
10k
Q1
2N2222
U2:A
7407
RC
RTS
R3
4k7
R4
10k
Q2
2N2222
U2:B
7407
DTE
R5
4k7
R6
10k
Q3
2N2222
U2:C
7407
DCE
R7
4k7
R8
10k
Q4
2N2222
U2:D
7407
CTS
R9
4k7
R10
10k
Q5
2N2222
U2:E
7407
DTR
R11
4k7
R12
10k
Q6
2N2222
U3:A
7407
DSR
R13
4k7
R14
10k
Q7
2N2222
U3:B
7407
TC
R15
4k7
R16
10k
Q8
2N2222
U3:C
7407
CD
R17
4k7
R18
10k
Q9
2N2222
U3:D
7407
EXT CLK
R19
4k7
R20
10k
Q10
2N2222
U3:E
7407

APÊNDICE B – DIAGRAMAS DE COMPOSIÇÃO DO PROJETO




Analisador de protocolo RS-232 HDLC
Camada física
Iniciar
Interface de entrada (Conector DB-25)
Tensão positiva?
Saída 0 V ('0' lógico)
Saída 5 V ('1' lógico)
Trem de bits que formam um quadro HDLC
Tratamento dos quadros HDLC
Armazenar os quadros recebidos
Desagrupar os bits do quadro
Tratar o campo FCS
Indicar que não há erro no quadro
Detectado erro pelo FCS?
Indicar que o quadro possui erro
Informações a serem exibidas no aplicativo
Enviar informações via bluetooth
Exibição dos dados pelo aplicativo Android
Receber informações via bluetooth
Definir o dispositivo de conexão bluetooth
Apresenta detalhes dos quadros em hexadecimal
Fim

APÊNDICE C – PROGRAMA DECODIFICADOR HDLC

```
/**

******************************************************
  *   UNIVERSIADE DO ESTADO DO AMAZONAS
  * ESCOLA SUPERIOR DE TECNOLOGIA

******************************************************
  * PROTÓTIPO DE BAIXO CUSTO DE ANALISADOR RS-
232
  * SÍNCRONO E PROTOCOLO HDLC PARA LINHAS DE
  *    RADIODETERMINAÇÃO
  *
  * Acadêmico: Derik Maia.
  * Manaus-AM, Outubro de 2022.

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

/* Private includes ----------------------------------------------------
---*/
/* USER CODE BEGIN Includes */
#include "mylibrary.h"
#include <stdio.h>
#include <stdlib.h>
#include "epimetheus.h"
/* USER CODE END Includes */

/* Private variables ---------------------------------------------------
---*/
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim4;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
char BTMessage[1024];
char StrBuffer[256];

uint16_t  Tm1Ch1[2] = {0,0},
    Tm4Ch1[2] = {0,0},
    Tm4Ch4[2] = {0,0},
    Freqs[3] =  {0,0,0};
uint8_t set[3] =  {0,0,0};
typedef struct RS232_status
{
char   CTS,
    RTS,
    DTR,
    DSR,
    CD,
    TC[6],
    RC[6],
    EXT[6];
} RS232_status;

typedef struct HDLC_frame
{
uint16_t FrameID;
char  FrameType,
    Address,
    Control,
    Data[256],
    FCS[2],
    FCSInfo;
} HDLC_frame;

uint8_t HDLCInput[4096];
uint8_t InputBuffer,
   Msb4,
   Lsb4;
uint16_t AccInputBuffer = 0;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------
------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM4_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------
----*/
/* USER CODE BEGIN 0 */
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef
*htim)
{
 if (htim==&htim1)
 {
  if (set[0]==2)
  {
   set[0]=0;
   if ((Tm1Ch1[1]-Tm1Ch1[0]) > 0)
Freqs[0]=7200000/(Tm1Ch1[1]-Tm1Ch1[0]);
   if (Freqs[0] >= 4700 && Freqs[0] <= 4900) Freqs[0] = 4800;
   else if (Freqs[0] >= 9500 && Freqs[0] <= 9700) Freqs[0] =
9600;
   else if (Freqs[0] >= 19100 && Freqs[0] <= 19300) Freqs[0] =
19200;
   else if (Freqs[0] >= 38300 && Freqs[0] <= 38500) Freqs[0] =
38400;
   else Freqs[0] = 0;
  }
  if (set[0]==1)
  {
   Tm1Ch1[1]=HAL_TIM_ReadCapturedValue(&htim1,
TIM_CHANNEL_1);
   set[0]=2;
  }
  if (set[0]==0)
  {
   Tm1Ch1[0]=HAL_TIM_ReadCapturedValue(&htim1,
TIM_CHANNEL_1);
   set[0]=1;
  }

 } else
  if (htim==&htim4)
 {
  if (htim->Channel == 1)
  {
   if (set[1]==2)
   {
    set[1]=0;
    if ((Tm4Ch1[1]-Tm4Ch1[0]) > 0)
Freqs[1]=7200000/(Tm4Ch1[1]-Tm4Ch1[0]);
    if (Freqs[1] >= 4700 && Freqs[1] <= 4900) Freqs[1] = 4800;
    else if (Freqs[1] >= 9500 && Freqs[1] <= 9700) Freqs[1] =
9600;
    else if (Freqs[1] >= 19100 && Freqs[1] <= 19300) Freqs[1]
= 19200;
    else if (Freqs[1] >= 38300 && Freqs[1] <= 38500) Freqs[1]
= 38400;
```

```
    else Freqs[1] = 0;
   }
   if (set[1]==1)
   {
    Tm4Ch1[1]=HAL_TIM_ReadCapturedValue(&htim4,
TIM_CHANNEL_1);
    set[1]=2;
   }
   if (set[1]==0)
   {
    Tm4Ch1[0]=HAL_TIM_ReadCapturedValue(&htim4,
TIM_CHANNEL_1);
    set[1]=1;
   }
  } else if (htim->Channel == 8)
  {
   if (set[2]==2)
   {
    set[2]=0;
    if ((Tm4Ch4[1]-Tm4Ch4[0]) > 0)
Freqs[2]=7200000/(Tm4Ch4[1]-Tm4Ch4[0]);
    if (Freqs[2] >= 4700 && Freqs[2] <= 4900) Freqs[2] = 4800;
    else if (Freqs[2] >= 9500 && Freqs[2] <= 9700) Freqs[2] =
9600;
    else if (Freqs[2] >= 19100 && Freqs[2] <= 19300) Freqs[2]
= 19200;
    else if (Freqs[2] >= 38300 && Freqs[2] <= 38500) Freqs[2]
= 38400;
    else Freqs[2] = 0;
   }
   if (set[2]==1)
   {
    Tm4Ch4[1]=HAL_TIM_ReadCapturedValue(&htim4,
TIM_CHANNEL_4);
    set[2]=2;
   }
   if (set[2]==0)
   {
    Tm4Ch4[0]=HAL_TIM_ReadCapturedValue(&htim4,
TIM_CHANNEL_4);
    set[2]=1;
   }
  }
 }
}

int main(void)
{
 /* USER CODE BEGIN 1 */
 RS232_status Status = {'n','n','n','n','n',"","",""};

 /* USER CODE END 1 */

 /* MCU Configuration--------------------------------------------------
-------*/

 /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
 HAL_Init();

 /* USER CODE BEGIN Init */

 /* USER CODE END Init */

 /* Configure the system clock */
 SystemClock_Config();

 /* USER CODE BEGIN SysInit */

 /* USER CODE END SysInit */

 /* Initialize all configured peripherals */
 MX_GPIO_Init();
 MX_TIM2_Init();
 MX_USART2_UART_Init();
 MX_USART1_UART_Init();
 MX_TIM4_Init();
 MX_TIM1_Init();
 /* USER CODE BEGIN 2 */
 HAL_Delay(200);
 HAL_GPIO_WritePin(BluetoothReset_GPIO_Port,
BluetoothReset_Pin, SET);
 HAL_Delay(1000);
 memset(buffer, 0, sizeof(buffer));
 HAL_TIM_Base_Start_IT(&htim2);
 __HAL_UART_ENABLE_IT(&huart2, UART_IT_RXNE);

 HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_1);   // RC
Clock
 HAL_TIM_IC_Start_IT(&htim4, TIM_CHANNEL_1);   // TC
Clock
 HAL_TIM_IC_Start_IT(&htim4, TIM_CHANNEL_4);   //
EXT Clock

 /* USER CODE END 2 */

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  HAL_Delay(800);
  Status.CTS = HAL_GPIO_ReadPin(GPIOA, CTS_Pin);
  Status.RTS = HAL_GPIO_ReadPin(GPIOA, RTS_Pin);
  Status.DTR = HAL_GPIO_ReadPin(GPIOB, DTR_Pin);
  Status.DSR = HAL_GPIO_ReadPin(GPIOB, DSR_Pin);
  Status.CD =  HAL_GPIO_ReadPin(GPIOB, CD_Pin);
  sprintf(StrBuffer, "%d", Freqs[0]);
  strcpy(Status.RC, StrBuffer);
  sprintf(StrBuffer, "%d", Freqs[1]);
  strcpy(Status.TC, StrBuffer);
  sprintf(StrBuffer, "%d", Freqs[2]);
  strcpy(Status.EXT, StrBuffer);

  // Resets RS-232 input baudrate from RC pin.
  HAL_UART_DeInit(&huart1);
  if (Status.RC == 0 && Status.TC == 0 && Status.EXT == 0){
   huart1.Init.BaudRate = atoi((Status.RC == 0)? Status.RC :
Status.TC);
  }
  HAL_UART_Init(&huart1);

  HAL_UART_Receive(&huart1, HDLCInput,
sizeof(HDLCInput), 1000);

  //Tratamento HDLC
  InputBuffer = HDLCInput[0];
  Msb4 = (InputBuffer & 0xF0) >> 4;
  Lsb4 = InputBuffer & 0x0F;

  InputBuffer = HDLCInput [j];
  AccInputBits = 8;
  do {
 Lsb4 = SplitByte(InputBuffer,0);
 Msb4 = SplitByte(InputBuffer,1);
 RemoveBits =
REMOVE_BITS_TABLE[LSBConsecOnesinMSB4(Msb4)][M
SBConsecOnesinLSB4(Lsb4)];
      Pattern = PATTERN_TYPE_TABLE
[LSBConsecOnesinMSB4(Msb4)][MSBConsecOnesinLSB4(Ls
b4)];
      printf("\nPattern: %x\n", Pattern);
      switch (Pattern)
      {
         case 0: //NORMAL
            if (Flag==0) break;
            OutputBuffer = GetNBitsFromMSB(InputBuffer,
RemoveBits);
            for (OutputBufferGap=RemoveBits; OutputBufferGap
> 0; OutputBufferGap--)
```

```
            {
               k = AccOutputBits/8;
               OUTPUT[k] |= GetNBitsFromMSB(OutputBuffer,
1) >> AccOutputBits%8;
               OutputBuffer <<= 1;
               AccOutputBits++;
            }
            break;
         case 1: // BIT_STUFF
            if (Flag==0) break;
            OutputBuffer = GetNBitsFromMSB(InputBuffer,
RemoveBits-1);
            for (OutputBufferGap=RemoveBits; OutputBufferGap
> 0; OutputBufferGap--)
            {
               k = AccOutputBits/8;
               OUTPUT[k] |= GetNBitsFromMSB(OutputBuffer,
1) >> AccOutputBits%8;
               OutputBuffer <<= 1;
               AccOutputBits++;
            }
            break;
         case 2: // FLAG
            Flag++;
            FlagOutputIndex = k;
            if (k > FlagOutputIndex)
            {
               // Compor mensagem (conteudo do frame
               HDLCFrame HdlcFrame;
               HdlcFrame.FrameID = Flag;
               HdlcFrame.Address = OUTPUT[FlagOutputIndex];
               HdlcFrame.ControlInfo =
OUTPUT[FlagOutputIndex+1];
               for (unsigned char i=0;
i<=sizeof(HdlcFrame.Information);i++)
               {
                  HdlcFrame.Information[i] =
OUTPUT[FlagOutputIndex+2+i];
               }
               HdlcFrame.FCS[0] = OUTPUT[k-1];
               HdlcFrame.FCS[1] = OUTPUT[k];
            }
            break;
         case 3: // ABORT
            Abort++;
         default:
            //Nothing
            break;
      }

      for (InputBufferGap=RemoveBits; InputBufferGap > 0;
InputBufferGap--)
      {
         j = AccInputBits/8;
         InputBuffer <<= 1;
         InputBuffer |=  SubsetBits(HDLCInput[j], 7-
AccInputBits%8, 1);
         AccInputBits++;
      }
      } while (j < sizeof(HDLCInput));

   HDLC_frame HDLCFrame;

   // Monta mensagem JSON
   /*
   {
     "address": "00",
     "frame_type": "00",
     "control_information": "00",
     "fcs": "4e02",
     "message": "000022000bf0e8de050284b38ce04e02",
   "fcs_info": "Good",
     "control_CD": "y",
     "control_DSR": "n",
     "control_DTR": "y",
     "control_RTS": "n",
     "control_CTS": "n",
     "bits_segundo": "19200",
     "bits_dados": "8",
     "paridade": "None",
     "bits_parada": "1",
     "controle_fluxo": "None"
     }
   */
   unsigned int i = 0,
         j = sizeof(Data)-1;

   for (i = 0; i <= j; i++)
   {
    Data[i] = Data1[i];
    if (Data1[j] == '\0') j--;
   }

   for (i = 0; i <= sizeof(Data); i++)
   {
    if (i == 0) HDLCFrame.Address = Data[i];
    else if (i == 1) HDLCFrame.Control = Data[i];
    else if (i >=2 && i < j-1) HDLCFrame.Data[i-2] = Data[i];
    else if (i >=j-1 && i <= j) HDLCFrame.FCS[i+1-j] = Data[i];
   }
   HDLCFrame.FrameType = 0;

   HDLCFrame.FCSInfo = 'y';
   strcpy(BTMessage, "{\"address\":\"");
   sprintf(StrBuffer, "%02X", HDLCFrame.Address);
   strcat(BTMessage, StrBuffer);
   strcat(BTMessage, "\",\"frame_type\":\"");
   sprintf(StrBuffer, "%02X", HDLCFrame.FrameType);
   strcat(BTMessage, StrBuffer);
   strcat(BTMessage, "\",\"control_information\":\"");
   sprintf(StrBuffer, "%02X", HDLCFrame.Control);
   strcat(BTMessage, StrBuffer);
   strcat(BTMessage, "\",\"fcs\":\"");
   for (i = 0; i < sizeof(HDLCFrame.FCS); i++)
   {
    sprintf(StrBuffer, "%02X", HDLCFrame.FCS[i]);
    strcat(BTMessage, StrBuffer);
   }
   strcat(BTMessage, "\",\"message\":\"");
   for (i = 0; i <= j-2; i++)
   {
    sprintf(StrBuffer, "%02X", HDLCFrame.Data[i]);
    strcat(BTMessage, StrBuffer);
   }
   strcat(BTMessage, "\",\"fcs_info\":\"");
   strcat(BTMessage, HDLCFrame.FCSInfo);
   strcat(BTMessage, "\",\"control_CD\":\"");
   strcat(BTMessage, (Status.CD)?"y":"n");
   strcat(BTMessage, "\",\"control_DSR\":\"");
   strcat(BTMessage, (Status.DSR)?"y":"n");
   strcat(BTMessage, "\",\"control_DTR\":\"");
   strcat(BTMessage, (Status.DTR)?"y":"n");
   strcat(BTMessage, "\",\"control_RTS\":\"");
   strcat(BTMessage, (Status.RTS)?"y":"n");
   strcat(BTMessage, "\",\"control_CTS\":\"");
   strcat(BTMessage, (Status.CTS)?"y":"n");
   strcat(BTMessage, "\",\"bits_segundo\":\"");
   strcat(BTMessage, Status.RC);
   strcat(BTMessage, "\",\"transmit_clock\":\"");
   strcat(BTMessage, Status.TC);
   strcat(BTMessage, "\",\"external_clock\":\"");
   strcat(BTMessage, Status.EXT);
   strcat(BTMessage, "\",\"external_clock\":\"");
   strcat(BTMessage, Status.EXT);
   strcat(BTMessage, "\",\"bits_dados\":\"");
   strcat(BTMessage, huart2.Init.WordLength);
   strcat(BTMessage, "\",\"paridade\":\"");
   strcat(BTMessage, huart2.Init.Parity);
   strcat(BTMessage, "\",\"bits_parada\":\"");
   strcat(BTMessage, huart2.Init.StopBits);
   strcat(BTMessage, "\",\"controle_fluxo\":\"");
   strcat(BTMessage, huart2.Init.HwFlowCtl);
```

```
    strcat(BTMessage, "}");
    // Envia mensagem pela interface bluetooth
    HAL_UART_Transmit(&huart1, BTMessage,
sizeof(BTMessage), 1000);
    HAL_Delay(500);

  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified
parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue =
RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource =
RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 10-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_IC_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity =
TIM_INPUTCHANNELPOLARITY_FALLING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC,
TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 720;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 1999;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2,
&sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2,
&sMasterConfig) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief TIM4 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM4_Init(void)
{

  /* USER CODE BEGIN TIM4_Init 0 */

  /* USER CODE END TIM4_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM4_Init 1 */

  /* USER CODE END TIM4_Init 1 */
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 10-1;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 65535;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_IC_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity =
TIM_INPUTCHANNELPOLARITY_FALLING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC,
TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity =
TIM_INPUTCHANNELPOLARITY_RISING;
  if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC,
TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM4_Init 2 */

  /* USER CODE END TIM4_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
```

```
  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin,
GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(BluetoothReset_GPIO_Port,
BluetoothReset_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOTest_GPIO_Port, GPIOTest_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pins : LED_Pin BluetoothReset_Pin */
  GPIO_InitStruct.Pin = LED_Pin|BluetoothReset_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : GPIOTest_Pin */
  GPIO_InitStruct.Pin = GPIOTest_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOTest_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : CTS_Pin RTS_Pin */
  GPIO_InitStruct.Pin = CTS_Pin|RTS_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : DTR_Pin DSR_Pin CD_Pin */
  GPIO_InitStruct.Pin = DTR_Pin|DSR_Pin|CD_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}
```

```
/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL
error return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source
line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name
and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n",
file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

APÊNDICE D – APLICATIVO ANDROID

BluetoothChatFragment.java

```
package com.example.android.bluetoothchat;

import android.app.ActionBar;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentActivity;

import com.example.android.common.logger.Log;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Controle do Bluetooth para comunicar com outros aparelhos
 */
public class BluetoothChatFragment extends Fragment {

    private static final String TAG = "BluetoothChatFragment";

    private static final int
REQUEST_CONNECT_DEVICE_SECURE = 1;
    private static final int
REQUEST_CONNECT_DEVICE_INSECURE = 2;
    private static final int REQUEST_ENABLE_BT = 3;
    private String receiveBuffer = "";
    private String mConnectedDeviceName = null;
    private StringBuffer mOutStringBuffer;
    private BluetoothAdapter mBluetoothAdapter = null;
    private BluetoothChatService mChatService = null;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
        mBluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
        FragmentActivity activity = getActivity();
        if (mBluetoothAdapter == null && activity != null) {
            Toast.makeText(activity, "Bluetooth is not available",
Toast.LENGTH_LONG).show();
            activity.finish();
        }
    }

    @Override
    public void onStart() {
        super.onStart();
        if (mBluetoothAdapter == null) {
            return;
        }
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent,
REQUEST_ENABLE_BT);
        } else if (mChatService == null) {
            setupChat();
        }
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (mChatService != null) {
            mChatService.stop();
        }
    }

    @Override
    public void onResume() {
        super.onResume();

        if (mChatService != null) {
            if (mChatService.getState() ==
BluetoothChatService.STATE_NONE) {
                mChatService.start();
            }
        }
    }

    @Override
    public void onViewCreated(View view, @Nullable Bundle
savedInstanceState) {

    }

    private void setupChat() {
        Log.d(TAG, "setupChat()");

        FragmentActivity activity = getActivity();
        if (activity == null) {
            return;
        }
        mChatService = new BluetoothChatService(activity,
mHandler);
        mOutStringBuffer = new StringBuffer();
    }

    private void ensureDiscoverable() {
        if (mBluetoothAdapter.getScanMode() !=

BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOV
ERABLE) {
            Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERAB
LE);

discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCO
VERABLE_DURATION, 300);
            startActivity(discoverableIntent);
        }
    }

    private void setStatus(int resId) {
        FragmentActivity activity = getActivity();
        if (null == activity) {
            return;
        }
        final ActionBar actionBar = activity.getActionBar();
        if (null == actionBar) {
```

```
            return;
        }
        actionBar.setSubtitle(resId);
    }

    private void setStatus(CharSequence subTitle) {
        FragmentActivity activity = getActivity();
        if (null == activity) {
            return;
        }
        final ActionBar actionBar = activity.getActionBar();
        if (null == actionBar) {
            return;
        }
        actionBar.setSubtitle(subTitle);
    }

    private void messageHandler()
    {

    }

    private final Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            FragmentActivity activity = getActivity();
            switch (msg.what) {
                case Constants.MESSAGE_STATE_CHANGE:
                    switch (msg.arg1) {
                        case
BluetoothChatService.STATE_CONNECTED:
                            setStatus(getString(R.string.title_connected_to,
mConnectedDeviceName));
                            break;
                        case
BluetoothChatService.STATE_CONNECTING:
                            setStatus(R.string.title_connecting);
                            break;
                        case BluetoothChatService.STATE_LISTEN:
                        case BluetoothChatService.STATE_NONE:
                            setStatus(R.string.title_not_connected);
                            break;
                    }
                    break;
                case Constants.MESSAGE_WRITE:
                    break;
                case Constants.MESSAGE_READ:
                    byte[] readBuf = (byte[]) msg.obj;
                    String readMessage = new String(readBuf, 0,
msg.arg1);
                    receiveBuffer += readMessage;
                    if(receiveBuffer.contains("\n")) {
                        receiveBuffer = receiveBuffer.substring(0,
receiveBuffer.length() - 1);
                        messageHandler();
                        receiveBuffer = "";
                    }
                    break;
                case Constants.MESSAGE_DEVICE_NAME:
                    mConnectedDeviceName =
msg.getData().getString(Constants.DEVICE_NAME);
                    if (null != activity) {
                        Toast.makeText(activity, "Conectado a "
                                + mConnectedDeviceName,
Toast.LENGTH_SHORT).show();
                    }
                    break;
                case Constants.MESSAGE_TOAST:
                    if (null != activity) {
                        Toast.makeText(activity,
msg.getData().getString(Constants.TOAST),
                                Toast.LENGTH_SHORT).show();
                    }
                    break;
            }
        }
    };

    public void onActivityResult(int requestCode, int resultCode,
Intent data) {
        switch (requestCode) {
            case REQUEST_CONNECT_DEVICE_SECURE:
                if (resultCode == Activity.RESULT_OK) {
                    connectDevice(data, true);
                }
                break;
            case REQUEST_CONNECT_DEVICE_INSECURE:
                if (resultCode == Activity.RESULT_OK) {
                    connectDevice(data, false);
                }
                break;
            case REQUEST_ENABLE_BT:
                if (resultCode == Activity.RESULT_OK) {
                    setupChat();
                } else {
                    Log.d(TAG, "Blutooth não habilitado");
                    FragmentActivity activity = getActivity();
                    if (activity != null) {
                        Toast.makeText(activity,
R.string.bt_not_enabled_leaving,
                                Toast.LENGTH_SHORT).show();
                        activity.finish();
                    }
                }
        }
    }

    private void connectDevice(Intent data, boolean secure) {
        Bundle extras = data.getExtras();
        if (extras == null) {
            return;
        }
        String address =
extras.getString(DeviceListActivity.EXTRA_DEVICE_ADDRE
SS);
        BluetoothDevice device =
mBluetoothAdapter.getRemoteDevice(address);
        mChatService.connect(device, secure);
    }

    @Override
    public void onCreateOptionsMenu(@NonNull Menu menu,
MenuInflater inflater) {
        inflater.inflate(R.menu.bluetooth_chat, menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        MainActivity activity = (MainActivity) getActivity();

        switch (item.getItemId()) {
            case R.id.button_PLAY: {
                Toast.makeText(activity, "Iniciando aquisição de
dados...", Toast.LENGTH_SHORT).show();
                activity.arrayAdapter.clear();
                activity.arrayAdapter.addAll(activity.record);
                return true;
            }
            case R.id.button_STOP: {
                Toast.makeText(activity, "Interrompendo aquisição de
dados...", Toast.LENGTH_SHORT).show();
                return true;
            }
            case R.id.button_CLEAR: {
                Toast.makeText(activity, "Limpando dados...",
Toast.LENGTH_SHORT).show();
                activity.arrayAdapter.clear();
                activity.textViewMessage.setText(null);
                activity.textViewHDLC.setText(null);
                activity.textViewRecord.setText(null);
                activity.editTextBitsegundo.setText(null);
```

```
            activity.editTextBitDados.setText(null);
            activity.editTextParidade.setText(null);
            activity.editTextBitParada.setText(null);
            activity.editTextControleFluxo.setText(null);
            activity.switchRTS.setChecked(false);
            activity.switchDTR.setChecked(false);
            activity.switchDSR.setChecked(false);
            activity.switchCD.setChecked(false);
            activity.switchCTS.setChecked(false);
            return true;
        }
        case R.id.button_SAVE: {
            Toast.makeText(activity, "Salvando dados...",
Toast.LENGTH_SHORT).show();
            SimpleDateFormat formatter = new
SimpleDateFormat("yyyy_MM_dd");
            String recording;
            String status_control;

            File logFile = new
File("/storage/emulated/0/Download/" + formatter.format(new
Date())
                    + "_Gravação.txt");

            if (!logFile.exists()){ try { logFile.createNewFile();}
                catch (IOException e) { e.printStackTrace();}
            }

            try {
                new FileWriter(logFile,false).close();
                BufferedWriter buf = new BufferedWriter(new
FileWriter(logFile,true));

                int i = 0;
                while (i< activity.record.toArray().length) {
                    Object object =
activity.listView.getItemAtPosition(i);
                    String str_i = "";
                    String fcs_i = "(Good)";
                    String frame_type_i = null;

                    if (activity.frame_type.get(i).equals("00")) {
                        frame_type_i = "(Information)";
                    } else if (activity.frame_type.get(i).equals("02")){
                        frame_type_i = "(Supervisory)";
                    } else if (activity.frame_type.get(i).equals("03")){
                        frame_type_i = "(Unnumbered)";
                    }

                    for (int j=0; j<=(activity.message.get(i).length()-
2);j=j+2){
                        str_i = str_i + " " +
activity.message.get(i).substring(j,j+2);
                    }

                    status_control = "Record #" + (i+1) + "
Timestamp = " +
                            activity.timestamp.get(i) + "  Lenght = " +
                            activity.message.get(i).length() +
"\nConfiguracoes de Porta:" + "\n     Bits por segundo    = " +
                            activity.bits_segundo.get(i) + "\n     Bits de
dados      = " +
                            activity.bits_dados.get(i) + "\n     Paridade
= " +
                            activity.paridade.get(i) + "\n     Bits de
Parada      = " +
                            activity.bits_parada.get(i) + "\n     Controle
de fluxo   = " +
                            activity.controle_fluxo.get(i)
+"\n\nControl:"+ "\n     RTS = " +
                            activity.control_RTS.get(i) + "\n     DTR = "
+
                            activity.control_DTR.get(i) + "\n     DSR = "
+
                            activity.control_DSR.get(i) + "\n     CD  = "
+
                            activity.control_CD.get(i) + "\n     CTS = " +
                            activity.control_CTS.get(i) + "\n";

                    recording = "HDLC:"
                            + "\n     Address            = 0" +
activity.address.get(i)
                            + "\n     Frame Type         = 0x" +
activity.frame_type.get(i)
                            + " " + frame_type_i
                            + "\n     Control Information = 0x" +
activity.control_information.get(i)
                            + "\n     FCS                = 0x" +
activity.fcs.get(i).substring(0,2)
                            + "-" + activity.fcs.get(i).substring(2,4)
                            + " " + fcs_i + "\n  Message:\n" + str_i +
"\n\n";

                    buf.append(status_control);
                    buf.newLine();
                    buf.append(recording);
                    buf.newLine();
                    i++;
                }
                buf.flush();
                buf.close();

                Toast.makeText(activity, "Dados salvos em " +
logFile.getAbsolutePath()
                        + ".", Toast.LENGTH_SHORT).show();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return true;
        }
        case R.id.secure_connect_scan: {
            Intent serverIntent = new Intent(getActivity(),
DeviceListActivity.class);
            startActivityForResult(serverIntent,
REQUEST_CONNECT_DEVICE_SECURE);
            return true;
        }
        case R.id.discoverable: {
            ensureDiscoverable();
            return true;
        }
    }
    return false;
  }
}
```

BluetoothChatService.java

```
package com.example.android.bluetoothchat;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;

import com.example.android.common.logger.Log;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

/**
 * Gerenciamento das conecções Bluetooth com outros aparelhos
 */
public class BluetoothChatService {
    private static final String TAG = "BluetoothChatService";
```

```
    private static final String NAME_SECURE =
"BluetoothChatSecure";
    private static final String NAME_INSECURE =
"BluetoothChatInsecure";
    private static final UUID MY_UUID_SECURE =
            UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");
    private static final UUID MY_UUID_INSECURE =
            UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private AcceptThread mSecureAcceptThread;
    private AcceptThread mInsecureAcceptThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;
    private int mNewState;
    public static final int STATE_NONE = 0;
    public static final int STATE_LISTEN = 1;
    public static final int STATE_CONNECTING = 2;
    public static final int STATE_CONNECTED = 3;

    public BluetoothChatService(Context context, Handler
handler) {
        mAdapter = BluetoothAdapter.getDefaultAdapter();
        mState = STATE_NONE;
        mNewState = mState;
        mHandler = handler;
    }

    private synchronized void updateUserInterfaceTitle() {
        mState = getState();
        Log.d(TAG, "updateUserInterfaceTitle() " + mNewState +
" -> " + mState);
        mNewState = mState;

mHandler.obtainMessage(Constants.MESSAGE_STATE_CHA
NGE, mNewState, -1).sendToTarget();
    }

    public synchronized int getState() {
        return mState;
    }

    public synchronized void start() {
        Log.d(TAG, "start");

        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        if (mSecureAcceptThread == null) {
            mSecureAcceptThread = new AcceptThread(true);
            mSecureAcceptThread.start();
        }
        if (mInsecureAcceptThread == null) {
            mInsecureAcceptThread = new AcceptThread(false);
            mInsecureAcceptThread.start();
        }

        updateUserInterfaceTitle();
    }

    public synchronized void connect(BluetoothDevice device,
boolean secure) {
        Log.d(TAG, "connect to: " + device);

        if (mState == STATE_CONNECTING) {
            if (mConnectThread != null) {
                mConnectThread.cancel();
                mConnectThread = null;
            }
        }
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        mConnectThread = new ConnectThread(device, secure);
        mConnectThread.start();
        updateUserInterfaceTitle();
    }

    public synchronized void connected(BluetoothSocket socket,
BluetoothDevice
            device, final String socketType) {
        Log.d(TAG, "connected, Socket Type:" + socketType);

        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        if (mSecureAcceptThread != null) {
            mSecureAcceptThread.cancel();
            mSecureAcceptThread = null;
        }
        if (mInsecureAcceptThread != null) {
            mInsecureAcceptThread.cancel();
            mInsecureAcceptThread = null;
        }

        mConnectedThread = new ConnectedThread(socket,
socketType);
        mConnectedThread.start();
        Message msg =
mHandler.obtainMessage(Constants.MESSAGE_DEVICE_NA
ME);
        Bundle bundle = new Bundle();
        bundle.putString(Constants.DEVICE_NAME,
device.getName());
        msg.setData(bundle);
        mHandler.sendMessage(msg);
        updateUserInterfaceTitle();
    }

    public synchronized void stop() {
        Log.d(TAG, "stop");

        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        if (mSecureAcceptThread != null) {
            mSecureAcceptThread.cancel();
            mSecureAcceptThread = null;
        }
        if (mInsecureAcceptThread != null) {
            mInsecureAcceptThread.cancel();
            mInsecureAcceptThread = null;
        }
        mState = STATE_NONE;
        updateUserInterfaceTitle();
    }

    public void write(byte[] out) {
        ConnectedThread r;
```

```
        synchronized (this) {
            if (mState != STATE_CONNECTED) return;
            r = mConnectedThread;
        }
        r.write(out);
    }

    private void connectionFailed() {
        Message msg =
mHandler.obtainMessage(Constants.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(Constants.TOAST, "Unable to connect
device");
        msg.setData(bundle);
        mHandler.sendMessage(msg);
        mState = STATE_NONE;
        updateUserInterfaceTitle();

        BluetoothChatService.this.start();
    }

    private void connectionLost() {
        Message msg =
mHandler.obtainMessage(Constants.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(Constants.TOAST, "Device connection
was lost");
        msg.setData(bundle);
        mHandler.sendMessage(msg);
        mState = STATE_NONE;
        updateUserInterfaceTitle();

        BluetoothChatService.this.start();
    }

    private class AcceptThread extends Thread {
        private final BluetoothServerSocket mmServerSocket;
        private String mSocketType;

        public AcceptThread(boolean secure) {
            BluetoothServerSocket tmp = null;
            mSocketType = secure ? "Secure" : "Insecure";

            try {
                if (secure) {
                    tmp =
mAdapter.listenUsingRfcommWithServiceRecord(NAME_SEC
URE,
                            MY_UUID_SECURE);
                } else {
                    tmp =
mAdapter.listenUsingInsecureRfcommWithServiceRecord(
                            NAME_INSECURE,
MY_UUID_INSECURE);
                }
            } catch (IOException e) {
                Log.e(TAG, "Socket Type: " + mSocketType +
"listen() failed", e);
            }
            mmServerSocket = tmp;
            mState = STATE_LISTEN;
        }

        public void run() {
            Log.d(TAG, "Socket Type: " + mSocketType +
                    "BEGIN mAcceptThread" + this);
            setName("AcceptThread" + mSocketType);

            BluetoothSocket socket;

            while (mState != STATE_CONNECTED) {
                try {
                    socket = mmServerSocket.accept();
                } catch (IOException e) {
                    Log.e(TAG, "Socket Type: " + mSocketType +
"accept() failed", e);
                    break;
                }

                if (socket != null) {
                    synchronized (BluetoothChatService.this) {
                        switch (mState) {
                            case STATE_LISTEN:
                            case STATE_CONNECTING:
                                connected(socket,
socket.getRemoteDevice(),
                                        mSocketType);
                                break;
                            case STATE_NONE:
                            case STATE_CONNECTED:
                                try {
                                    socket.close();
                                } catch (IOException e) {
                                    Log.e(TAG, "Could not close unwanted
socket", e);
                                }
                                break;
                        }
                    }
                }
            }
            Log.i(TAG, "END mAcceptThread, socket Type: " +
mSocketType);

        }

        public void cancel() {
            Log.d(TAG, "Socket Type" + mSocketType + "cancel "
+ this);
            try {
                mmServerSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "Socket Type" + mSocketType + "close()
of server failed", e);
            }
        }
    }

    private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;
        private String mSocketType;

        public ConnectThread(BluetoothDevice device, boolean
secure) {
            mmDevice = device;
            BluetoothSocket tmp = null;
            mSocketType = secure ? "Secure" : "Insecure";

            try {
                if (secure) {
                    tmp =
device.createRfcommSocketToServiceRecord(
                            MY_UUID_SECURE);
                } else {
                    tmp =
device.createInsecureRfcommSocketToServiceRecord(
                            MY_UUID_INSECURE);
                }
            } catch (IOException e) {
                Log.e(TAG, "Socket Type: " + mSocketType +
"create() failed", e);
            }
            mmSocket = tmp;
            mState = STATE_CONNECTING;
        }

        public void run() {
            Log.i(TAG, "BEGIN mConnectThread SocketType:" +
mSocketType);
            setName("ConnectThread" + mSocketType);
            mAdapter.cancelDiscovery();
```

```
            try {
                mmSocket.connect();
            } catch (IOException e) {
                try {
                    mmSocket.close();
                } catch (IOException e2) {
                    Log.e(TAG, "unable to close() " + mSocketType +
                            " socket during connection failure", e2);
                }
                connectionFailed();
                return;
            }

            synchronized (BluetoothChatService.this) {
                mConnectThread = null;
            }
            connected(mmSocket, mmDevice, mSocketType);
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect " + mSocketType + "
socket failed", e);
            }
        }
    }

    private class ConnectedThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket, String
socketType) {
            Log.d(TAG, "create ConnectedThread: " + socketType);
            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) {
                Log.e(TAG, "temp sockets not created", e);
            }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
            mState = STATE_CONNECTED;
        }

        public void run() {
            Log.i(TAG, "BEGIN mConnectedThread");
            byte[] buffer = new byte [100_000_000];
            int bytes;

            while (mState == STATE_CONNECTED) {
                try {
                    bytes = mmInStream.read(buffer);

mHandler.obtainMessage(Constants.MESSAGE_READ, bytes, -
1, buffer)
                            .sendToTarget();
                } catch (IOException e) {
                    Log.e(TAG, "disconnected", e);
                    connectionLost();
                    break;
                }
            }
        }

        public void write(byte[] buffer) {
            try {
                mmOutStream.write(buffer);

mHandler.obtainMessage(Constants.MESSAGE_WRITE, -1, -1,
buffer)
                        .sendToTarget();
            } catch (IOException e) {
                Log.e(TAG, "Exception during write", e);
            }
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }
}
```

DeviceListActivity.java

```
package com.example.android.bluetoothchat;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;

import com.example.android.common.logger.Log;

import java.util.Set;

/**
 * Listagem de aparelhos pareados e detectados
 */
public class DeviceListActivity extends Activity {

    private static final String TAG = "DeviceListActivity";
    public static String EXTRA_DEVICE_ADDRESS =
"device_address";
    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter<String> mNewDevicesArrayAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);


requestWindowFeature(Window.FEATURE_INDETERMINAT
E_PROGRESS);
        setContentView(R.layout.activity_device_list);
        setResult(Activity.RESULT_CANCELED);
        Button scanButton = findViewById(R.id.button_scan);
        scanButton.setOnClickListener(new
View.OnClickListener() {
            public void onClick(View v) {
                doDiscovery();
                v.setVisibility(View.GONE);
            }
        });

        ArrayAdapter<String> pairedDevicesArrayAdapter =
                new ArrayAdapter<>(this, R.layout.device_name);
        mNewDevicesArrayAdapter = new ArrayAdapter<>(this,
R.layout.device_name);
```

```
        ListView pairedListView =
findViewById(R.id.paired_devices);
        pairedListView.setAdapter(pairedDevicesArrayAdapter);

pairedListView.setOnItemClickListener(mDeviceClickListener);
        ListView newDevicesListView =
findViewById(R.id.new_devices);

newDevicesListView.setAdapter(mNewDevicesArrayAdapter);

newDevicesListView.setOnItemClickListener(mDeviceClickLis
tener);

        IntentFilter filter = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
        this.registerReceiver(mReceiver, filter);
        filter = new
IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISH
ED);
        this.registerReceiver(mReceiver, filter);

        mBtAdapter = BluetoothAdapter.getDefaultAdapter();
        Set<BluetoothDevice> pairedDevices =
mBtAdapter.getBondedDevices();

        if (pairedDevices.size() > 0) {

findViewById(R.id.title_paired_devices).setVisibility(View.VIS
IBLE);
            for (BluetoothDevice device : pairedDevices) {
                pairedDevicesArrayAdapter.add(device.getName() +
"\n" + device.getAddress());
            }
        } else {
            String noDevices =
getResources().getText(R.string.none_paired).toString();
            pairedDevicesArrayAdapter.add(noDevices);
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();

        if (mBtAdapter != null) {
            mBtAdapter.cancelDiscovery();
        }
        this.unregisterReceiver(mReceiver);
    }

    private void doDiscovery() {
        Log.d(TAG, "doDiscovery()");

        setProgressBarIndeterminateVisibility(true);
        setTitle(R.string.scanning);

findViewById(R.id.title_new_devices).setVisibility(View.VISIB
LE);
        if (mBtAdapter.isDiscovering()) {
            mBtAdapter.cancelDiscovery();
        }
        mBtAdapter.startDiscovery();
    }

    private AdapterView.OnItemClickListener
mDeviceClickListener
            = new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> av, View v, int
arg2, long arg3) {
            mBtAdapter.cancelDiscovery();
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);
            Intent intent = new Intent();
            intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
            setResult(Activity.RESULT_OK, intent);
            finish();
        }
    };

    private final BroadcastReceiver mReceiver = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                if (device != null && device.getBondState() !=
BluetoothDevice.BOND_BONDED) {
                    mNewDevicesArrayAdapter.add(device.getName()
+ "\n" + device.getAddress());
                }
            } else if
(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(
action)) {
                setProgressBarIndeterminateVisibility(false);
                setTitle(R.string.select_device);
                if (mNewDevicesArrayAdapter.getCount() == 0) {
                    String noDevices =
getResources().getText(R.string.none_found).toString();
                    mNewDevicesArrayAdapter.add(noDevices);
                }
            }
        }
    };
}
```

MainActivity.java

```
package com.example.android.bluetoothchat;


import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.RadioButton;
import android.widget.TextView;

import androidx.fragment.app.FragmentTransaction;

import
com.example.android.common.activities.SampleActivityBase;
import com.example.android.common.logger.Log;
import com.example.android.common.logger.LogWrapper;
import
com.example.android.common.logger.MessageOnlyLogFilter;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;

public class MainActivity extends SampleActivityBase {

    public static final String TAG = "MainActivity";

    String json = null;

    ArrayList<String> record = new ArrayList<>();
    ArrayList<String> address = new ArrayList<>();
    ArrayList<String> frame_type = new ArrayList<>();
    ArrayList<String> control_information = new ArrayList<>();
```

```
    ArrayList<String> fcs = new ArrayList<>();
    ArrayList<String> message = new ArrayList<>();
    ArrayList<String> timestamp = new ArrayList<>();
    ArrayList<String> control_CD = new ArrayList<>();
    ArrayList<String> control_DSR = new ArrayList<>();
    ArrayList<String> control_DTR = new ArrayList<>();
    ArrayList<String> control_RTS = new ArrayList<>();
    ArrayList<String> control_CTS = new ArrayList<>();
    ArrayList<String> bits_segundo = new ArrayList<>();
    ArrayList<String> bits_dados = new ArrayList<>();
    ArrayList<String> paridade = new ArrayList<>();
    ArrayList<String> bits_parada = new ArrayList<>();
    ArrayList<String> controle_fluxo = new ArrayList<>();

    JSONObject obj;
    JSONArray userArray;

    ListView listView;
    TextView textViewRecord;
    TextView textViewHDLC;
    TextView textViewMessage;
    EditText editTextBitsegundo;
    EditText editTextBitDados;
    EditText editTextParidade;
    EditText editTextBitParada;
    EditText editTextControleFluxo;
    RadioButton switchCD;
    RadioButton switchDSR;
    RadioButton switchDTR;
    RadioButton switchRTS;
    RadioButton switchCTS;
    ArrayAdapter<String> arrayAdapter;

    private boolean mLogShown;

     @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        listView               = findViewById(R.id.QUADRO_lista);
        textViewRecord         =
findViewById(R.id.text_Quadro_Detalhe_Record);
        textViewHDLC           =
findViewById(R.id.text_Quadro_Detalhe_HDLC);
        textViewMessage        =
findViewById(R.id.text_Quadro_Detalhe_Message);
        editTextBitsegundo     = findViewById(R.id.bitssegundo);
        editTextBitDados       = findViewById(R.id.bitsdados);
        editTextParidade       = findViewById(R.id.paridade);
        editTextBitParada      = findViewById(R.id.bitsparada);
        editTextControleFluxo  =
findViewById(R.id.controlefluxo);
        switchCD               = findViewById(R.id.switch_CD);
        switchDSR              = findViewById(R.id.switch_DSR);
        switchDTR              = findViewById(R.id.switch_DTR);
        switchRTS              = findViewById(R.id.switch_RTS);
        switchCTS              = findViewById(R.id.switch_CTS);

        try {
            obj = new JSONObject(loadJSONFromAsset());
            userArray = obj.getJSONArray("HDLC");
            for (int i =0; i < userArray.length(); i++){
                record.add("Message #" + (i+1) + ": " +
userArray.getJSONObject(i).getString("message"));

address.add(userArray.getJSONObject(i).getString("address"));

frame_type.add(userArray.getJSONObject(i).getString("frame_t
ype"));

control_information.add(userArray.getJSONObject(i).getString(
"control_information"));
                fcs.add(userArray.getJSONObject(i).getString("fcs"));
message.add(userArray.getJSONObject(i).getString("message"))
;

timestamp.add(userArray.getJSONObject(i).getString("timestam
p"));

control_CD.add(userArray.getJSONObject(i).getString("control
_CD"));

control_DSR.add(userArray.getJSONObject(i).getString("contro
l_DSR"));

control_DTR.add(userArray.getJSONObject(i).getString("contro
l_DTR"));

control_RTS.add(userArray.getJSONObject(i).getString("contro
l_RTS"));

control_CTS.add(userArray.getJSONObject(i).getString("contro
l_CTS"));

bits_segundo.add(userArray.getJSONObject(i).getString("bits_s
egundo"));

bits_dados.add(userArray.getJSONObject(i).getString("bits_dad
os"));

paridade.add(userArray.getJSONObject(i).getString("paridade"))
;

bits_parada.add(userArray.getJSONObject(i).getString("bits_par
ada"));

controle_fluxo.add(userArray.getJSONObject(i).getString("contr
ole_fluxo"));
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }

        arrayAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1);
        listView.setAdapter(arrayAdapter);

        listView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View
view, int position, long id) {

                Object object = listView.getItemAtPosition(position);
                String message_info = "";
                String fcs_info = "(Good)";
                String frame_info = null;

                textViewRecord.setText("  Record #" + (position+1) +
" Timestamp = " + timestamp.get(position)
                        + "  Lenght = " + message.get(position).length()
                );

                if (frame_type.get(position).equals("00")) {
                    frame_info = "(Information)";
                } else if (frame_type.get(position).equals("02")){
                    frame_info = "(Supervisory)";
                } else if (frame_type.get(position).equals("03")){
                    frame_info = "(Unnumbered)";
                }

                for (int i=0; i<=(message.get(position).length()-
2);i=i+2){
                    message_info = message_info + " " +
message.get(position).substring(i,i+2);
                }

                textViewHDLC.setText("  HDLC:"
```

```
                + "\n     Address            = 0" +
address.get(position)
                + "\n     Frame Type         = 0x" +
frame_type.get(position)
                + " " + frame_info
                + "\n     Control Information = 0x" +
control_information.get(position)
                + "\n     FCS                = 0x" +
fcs.get(position).substring(0,2)
                + "-" + fcs.get(position).substring(2,4)
                + " " + fcs_info);
            textViewMessage.setText("  Message:\n" +
message_info);


editTextBitsegundo.setText(bits_segundo.get(position));
            editTextBitDados.setText(bits_dados.get(position));
            editTextParidade.setText(paridade.get(position));
            editTextBitParada.setText(bits_parada.get(position));

editTextControleFluxo.setText(controle_fluxo.get(position));

            if (control_CD.get(position).equals("y")){
               switchCD.setChecked(true);
            } else if (control_CD.get(position).equals("n")){
               switchCD.setChecked(false);
            }

            if (control_CTS.get(position).equals("y")){
               switchCTS.setChecked(true);
            } else if (control_CTS.get(position).equals("n")){
               switchCTS.setChecked(false);
            }

            if (control_DSR.get(position).equals("y")){
               switchDSR.setChecked(true);
            } else if (control_DSR.get(position).equals("n")){
               switchDSR.setChecked(false);
            }

            if (control_DTR.get(position).equals("y")){
               switchDTR.setChecked(true);
            } else if (control_DTR.get(position).equals("n")){
               switchDTR.setChecked(false);
            }

            if (control_RTS.get(position).equals("y")){
               switchRTS.setChecked(true);
            } else if (control_RTS.get(position).equals("n")){
               switchRTS.setChecked(false);
            }
         }
      });

      if (savedInstanceState == null) {
         FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();
         BluetoothChatFragment fragment = new
BluetoothChatFragment();
         transaction.replace(R.id.sample_content_fragment,
fragment);
         transaction.commit();
      }
   }

   private String loadJSONFromAsset() {
      try {
         InputStream is = getAssets().open("jsonformatter.json");
         int size = is.available();
         byte[] buffer = new byte[size];
         is.read(buffer);
         is.close();
         json = new String(buffer, "UTF-8");
      } catch (IOException e) {
         e.printStackTrace();
         return null;
      }
      return json;
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.main, menu);
      return true;
   }

   @Override
   public boolean onPrepareOptionsMenu(Menu menu) {
      return super.onPrepareOptionsMenu(menu);
   }

   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      return super.onOptionsItemSelected(item);
   }

   @Override
   public void initializeLogging() {
      LogWrapper logWrapper = new LogWrapper();
      Log.setLogNode(logWrapper);
      MessageOnlyLogFilter msgFilter = new
MessageOnlyLogFilter();
      logWrapper.setNext(msgFilter);
      Log.i(TAG, "Ready");
   }
}
```