

**RELATÓRIO**

**Redes de Computadores**

**Trabalho Prático nº1 – Protocolo de Ligação de Dados**

*Turma 1*

Luís Afonso Sampaio Oliveira - up201707229@fe.up.pt

Pedro Miguel Ribeiro Alves – up201707234@fe.up.pt

Ricardo França Domingues Cardoso – up201604686@fe.up.pt

*Outubro de 2019*

Índice

[1. Sumário 4](#_Toc23884057)

[2. Introdução 4](#_Toc23884058)

[3. Arquitetura 4](#_Toc23884059)

[4. Estrutura do Código 5](#_Toc23884060)

[llfunctions 5](#_Toc23884061)

[Funções principais da camada de ligação: 5](#_Toc23884062)

[Variáveis Globais: 5](#_Toc23884063)

[Macros Principais: 5](#_Toc23884064)

[msg\_state\_machine 6](#_Toc23884065)

[Função da máquina de estados: 6](#_Toc23884066)

[Estados: 6](#_Toc23884067)

[Macros Principais: 6](#_Toc23884068)

[app\_package\_handling 6](#_Toc23884069)

[Funções principais da camada da aplicação: 6](#_Toc23884070)

[Macros Principais: 7](#_Toc23884071)

[noncanonical 8](#_Toc23884072)

[Funções principais da camada da aplicação: 8](#_Toc23884073)

[writenoncanonical 8](#_Toc23884074)

[Funções principais da camada da aplicação: 8](#_Toc23884075)

[5. Casos de Uso Principais 8](#_Toc23884076)

[Interface 8](#_Toc23884077)

[Sequência da transmissão de dados 8](#_Toc23884078)

[6. Protocolo de Ligação Lógica 9](#_Toc23884079)

[llopen 9](#_Toc23884080)

[llwrite 9](#_Toc23884081)

[llread 10](#_Toc23884082)

[llclose 10](#_Toc23884083)

[7. Protocolo de Aplicação 11](#_Toc23884084)

[buildControlPackage 11](#_Toc23884085)

[buildDataPackage 11](#_Toc23884086)

[createReceivedFile 11](#_Toc23884087)

[sendMessage 12](#_Toc23884088)

[fileDataReading 12](#_Toc23884089)

[8. Validação 12](#_Toc23884090)

[9. Eficiência do Protocolo de Ligação de Dados 13](#_Toc23884091)

[Variação da Capacidade de Ligação 13](#_Toc23884092)

[Variação do Tempo de Propagação (T\_prop) 13](#_Toc23884093)

[Variação do tamanho dos pacotes de dados 14](#_Toc23884094)

[Variação de FER 14](#_Toc23884095)

[10. Conclusões 15](#_Toc23884096)

[11. Anexo I - Código Fonte 15](#_Toc23884097)

[llfunctions.h 15](#_Toc23884098)

[llfunctions.c 21](#_Toc23884099)

[msg\_state\_machine.h 32](#_Toc23884100)

[msg\_state\_machine.c 33](#_Toc23884101)

[app\_package\_handling.h 35](#_Toc23884102)

[app\_package\_handling.c 37](#_Toc23884103)

[noncanonical.c 41](#_Toc23884104)

[writenoncanonical.c 46](#_Toc23884105)

[12. Anexo II – Tabelas 51](#_Toc23884106)

[Variação da Capacidade de Ligação 51](#_Toc23884107)

[Variação do Tempo de Propagação (T\_prop) 52](#_Toc23884108)

[Variação do tamanho dos pacotes de dados 53](#_Toc23884109)

[Variação de FER 54](#_Toc23884110)

[BCC1 54](#_Toc23884111)

[BCC2 55](#_Toc23884112)

# Sumário

  Este relatório foi efetuado no âmbito da unidade curricular de Redes de Computadores, de modo a complementar o primeiro trabalho laboratorial cujo foco é a transferência de dados.

Como conclusão, todos os objetivos propostos foram cumpridos, resultando numa aplicação totalmente funcional e capaz de transferir dados sem perdas.

# Introdução

O principal objetivo do trabalho é implementar um protocolo de ligação de dados de acordo com o guião previamente fornecido e desenvolver uma aplicação que utilize este protocolo como base para a transferência de ficheiros entre computadores através de um cabo série.

Este relatório é um suporte à aplicação desenvolvida e tem como funcionalidade explicar e analisar os vários componentes do trabalho e os seus resultados, unindo a componente prática à teórica. Isto passa pela discussão dos seguintes tópicos:

* **3. Arquitetura**

Exposição dos blocos funcionais e interfaces.

* **4. Estrutura do** **Código**

Discussão sobre a API, estruturas de dados utilizadas e funções de destaque.

* **5. Casos de uso principais**

Identificação dos casos de uso e análise das principais sequências de chamada de funções.

* **6. Protocolo de ligação lógica**

Identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.

* **7. Protocolo de aplicação**

Identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.

* **8. Validação**

Descrição dos testes efetuados com apresentação quantificada dos resultados.

* **9. Eficiência do protocolo de ligação de dados**

Caraterização estatística da eficiência do protocolo.

* **10. Conclusão**

Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

# 3. Arquitetura

  O projeto encontra-se dividido em dois blocos principais, sendo estes o bloco do emissor, responsável pelo envio do ficheiro, e o bloco do recetor, responsável pelo envio do mesmo.

Existe também uma divisão entre a camada do protocolo de transmissão de dados e a camada da aplicação, que é comum a todos os elementos do trabalho.

# 4. Estrutura do Código

  O código está distribuído por cinco ficheiros de código. Os ficheiros **“llfunctions.c”** e **“msg\_state\_machine.c”** tratam do Protocolo de Ligação de Dados, contendo o primeiro as funções especificadas no guião e as funções necessárias à sua implementação, e o segundo a definição da máquina de estados utilizada na leitura de tramas. O ficheiro **“app\_package\_handling.c”** contém as funções relativas ao Protocolo da Aplicação de Teste. O ficheiro “**noncanonical.c**” é responsável pelas funções do recetor e “**writenoncanonical.c**” é responsável pelas funções do emissor, e ambos os ficheiros recorrem às funções do Protocolo de Ligação de Dados e do Protocolo da Aplicação de Teste. Existe também, para cada ficheiro, um *header file*, onde estão declaradas as funções necessárias e constantes importantes.

## llfunctions

### Funções principais da camada de ligação:

* **llopen** – envia a trama de supervisão SET e recebe a trama UA, se for chamada pelo emissor; recebe a trama de supervisão SET e envia a trama UA, se for chamada pelo recetor;
* **llwrite** – faz stuffing das tramas I e envia-as;
* **llread** – recebe tramas I e recupera as tramas originais realizando destuffing;
* **llclose –** envia a trama de supervisão DISC, recebe DISC e envia UA, se for chamada pelo emissor; recebe a trama de controlo DISC, envia DISC e recebe UA, se for chamada pelo recetor.
* **serialReadControl2 –** lê o serial port caráter a caráter até a sequência desejada ser encontrada ou um alarme ser accionado. Utilizado para determinar a receção de tramas.

### Variáveis Globais:

* **numAlarms** – contador do número de alarmes realizados, inicializado a 0;
* **alarmFlag** – inicializada a FALSE, passa a TRUE sempre que um alarme é acionado;
* **oldtio, newtio –** structs termios com as definições da porta série;
* **curr\_ns –** número sequencial da trama a enviar, inicializado a 0.

### Macros Principais:

* **MAX\_RETR –** Número máximo de tentativas de reenvio;
* **TIMEOUT –** Número de segundos de cada alarme;
* **BAUDRATE –** Capacidade da Ligação;
* **D\_MAX\_SIZE –** Número máximo de bytes das tramas I;

## msg\_state\_machine

### Função da máquina de estados:

**stateMachineOpen –** altera o estado da máquina de estados conforme o caráter recebido e o estado atual. Utilizada para ler tramas de controlo e o header das tramas I.

### Estados:

* **START\_S –** Aguardando uma flag delimitadora que dê início à trama;
* **FLAG\_RCV –** Recebeu a flag com sucesso;
* **A\_RCV –** Recebeu o Campo de Endereço com sucesso;
* **C\_RCV –** Recebeu o Campo de Controlo com sucesso;
* **BCC\_OK –** Recebeu o BCC com sucesso;
* **STOP\_S –** Terminou o processamento com sucesso.

### Macros Principais:

* **F\_FLAG –** flag delimitadora das tramas;
* **A1\_ -** Campo de endereço em Comandos enviados pelo Emissor e Respostas enviadas pelo Receptor;
* **A2\_ -** Campo de endereço em Comandos enviados pelo Receptor e Respostas enviadas pelo Emissor;
* **C\_SET –** Campo de controlo do Comando SET;
* **C\_I(n) –** Campo de controlo da trama I, com n = Ns;
* **C\_DISC –** Campo de controlo do Comando DISC;
* **C\_UA –** Campo de controlo da Resposta UA;
* **C\_RR(n) –** Campo de controlo da Resposta RR, com n = Nr;
* **C\_REJ(n) –** Campo de controlo da Resposta REJ, com n = Nr;
* **ESC\_OCT –** Octeto de Escape, utilizado no stuffing da trama I;
* **FLAG\_STUFF –** Octeto utilizado no stuffing da ocorrência da Flag nos dados ou no bcc dos dados na trama I;
* **ESC\_STUFF –** Octeto utilizado no stuffing da ocorrência do Octeto de Escape nos dados ou no bcc dos dados na trama I;

## app\_package\_handling

### Funções principais da camada da aplicação:

* **buildControlPackage –** Constrói um pacote de controlo;
* **buildDataPackage –** Constrói um pacote de dados;
* **readStartPackage –** Lê o pacote de controlo *start* e guarda a informação recebida.
* **readPackage –** Lê um pacote e determina se se trata de um pacote de dados ou do pacote de controlo *endI,* processando a informação recebida se for de dados.
* **stateMachineApp –** altera o estado da máquina de estados conforme o estado atual. Utilizada para identificar o estado da aplicação;
* **displayCompletion –** mostra o progresso da transferência.

### Macros Principais:

* **C\_DATA –** Campo de Controlo do pacote de dados;
* **C\_START –** Campo de Controlo do pacote *start;*
* **C\_END –** Campo de Controlo do pacote *end;*
* **T\_SIZE –** campo de *type* do tamanho do ficheiro em formato *tlv*;
* **T\_NAME –** Campo de *type* do nome do ficheiro em formato *tlv*;
* **PACKAGE\_DATA\_SIZE –** número máximo de bytes que podem ser enviados em cada pacote de dados;

## noncanonical

### Funções principais da camada da aplicação:

* **isFileSizeExpected –** compara o tamanho do ficheiro recebido e o tamanho do ficheiro original;
* **createReceivedFile –** verifica se já existe um ficheiro com o mesmo nome e cria o ficheiro;
* **main –** base da camada da aplicação, visto que gere todo o processo nesta camada e chama as funções da camada de ligação.

## writenoncanonical

### Funções principais da camada da aplicação:

* **sendMessage –** envia o pacote pelo serial port, chamando **llwrite**, e determina se é necessário reenviar;
* **fileDataReading –** lê dados do ficheiro e constrói um pacote de dados. Muda o estado da aplicação se chegar ao fim do ficheiro.
* **setupControlPackage –** determina o tamanho do ficheiro, constrói o pacote de controlo e muda o estado da aplicação.
* **main –** base da camada da aplicação, visto que gere todo o processo nesta camada e chama as funções da camada de ligação.

# 5. Casos de Uso Principais

Os casos de uso da aplicação são a interface, que permite que o transmissor escolha o ficheiro a enviar, e a transferência do ficheiro por porta série, entre dois computadores, o transmissor e o recetor.

## Interface

O utilizador, de modo a iniciar a aplicação, deve inserir um conjunto de argumentos. No caso do emissor o utilizador deve inserir a porta de série (ex: **/dev/ttyS0**), e o nome do ficheiro a transmitir (ex: **pinguim.gif**). No caso do recetor o utilizador necessita apenas de introduzir a porta série.

## Sequência da transmissão de dados

1. Transmissor escolhe o ficheiro a enviar;
2. Configuração da ligação entre os computadores;
3. Estabelecimento da ligação;
4. Envio da informação por parte do emissor;
5. Receção da informação por parte do recetor;
6. Armazenamento da informação recebida pelo recetor num ficheiro com um nome igual, se não existir um ficheiro com igual nome, ou com um sufixo adequado, se já existir;
7. Terminação da ligação.

# 6. Protocolo de Ligação Lógica

As escritas no serial port são feitas trama a trama, no entanto a leitura é feita caráter a caráter.

## llopen

int llopen(char \*port, int type);

Esta função estabelece a ligação entre o emissor e o recetor. Como tal, a abertura da porta de série e respetiva configuração é também realizada no início desta função, de modo a evitar a repetição de código.

No emissor, esta função envia a trama de controlo SET e ativa um alarme, que é desativado após receber uma resposta (UA). Se não receber uma resposta dentro de um determinado intervalo de tempo, SET é reenviado. Este mecanismo de retransmissão é repetido até um número máximo de vezes, terminando o programa se esse número for atingido sem obter nenhuma resposta.

No recetor, a função aguarda a chegada de uma trama de controlo SET, enviando uma resposta UA após a sua receção. Para enviar tanto UA como SET é utilizada a função **sendControlMessage.** Esta função tem como argumentos o Campo de Controlo e o Campo de Endereço das tramas de Supervisão/Não Numeradas, permitindo o envio de qualquer trama destes tipos.

Na receção de UA e SET é utilizada a função **serialReadControl1**, versão mais simples de **serialReadControl2**, que utiliza uma máquina de estados de modo a registar mais facilmente a ocorrência da trama desejada. A função termina quando se encontra a sequência desejada ou com a ocorrência de um alarme, o que permite interromper a leitura do serial port para reenviar tramas. A função recebe como argumento um Campo de Controlo e um Campo de Endereço, permitindo ler ambas as tramas de supervisão utilizadas em **llopen**.

## llwrite

int llwrite(int fd, uint8\_t \*buffer, int length);

Esta função é responsável pelo stuffing e envio das tramas realizados pelo emissor.

É utilizado um número de sequência, **Ns** no emissor, **Nr** no emissor, para sincronizar o emissor com o recetor.

Primeiro é realizado o *framing* da mensagem, isto é, acrescenta-se o cabeçalho do Protocolo de Ligação à mensagem (o cálculo do BCC2 é realizado pela função **dataBCC).** É realizado o *stuffing*, quer da mensagem quer do BCC2, recorrendo à função **byteStuffing**. Após o stuffing a mensagem é enviada.

O envio da trama I possui o mesmo mecanismo de retransmissão que o envio de SET utilizado em **llopen.** Após o envio da trama é accionado um alarme que é desativado após a receção de uma resposta RR ou REJ. Caso o alarme seja atingido, a trama é reenviada até um número máximo de vezes. Se receber um REJ é registada a necessidade de reenviar a mensagem. A verificação do tipo de resposta é realizada através da função **serialReadControl2,** que, tendo como argumentos o Campo de Endereço e dois Campos de Controlo (C de REJ e C de RR), utiliza uma máquina de estados até atingir o estado final, ao encontrar a flag de delimitadora que determina o fim da trama, ou um alarme ser accionado.

**Ns** apenas é alterado após a receção de uma resposta RR com **Nr** diferente, permanecendo igual no caso de reenvio, time-out e ocorrência de alarm.

## llread

int llread(int fd, uint8\_t \*buffer);

Esta função é responsável pela receção e destuffing das tramas por realizadas pelo emissor.

O *header* é validado pela função **serialReadControl2,** que recebe como argumento um Campo de Endereço e dois possíveis Campos de Controlo (**C** quando **Ns** é 1 e **C** quando **Ns** é 0). Após receber um *header* válido, a função **readIFrame** processa o resto da mensagem, realizando o destuffing e verificando o BCC2. Caso o Campo de Controlo corresponda a um **Ns** igual ao **Nr** atual, significa que a informação já foi recebida e se trata de um duplicado, que é descartado, e envia-se uma resposta RR. Caso contrário, se houver erros nos dados, a trama também é descartada, enviando uma resposta REJ. Se não houver erros nos dados e o número de sequência for o esperado, a mensagem é processada.

De modo a permitir o teste da eficiência com a variação de FER, é chamada a função **hasError,** utilizada para simular a ocorrência de erros no BCC1 e no BCC2, com probabilidades constantes e independentes, definidas antes de o programa correr.

## llclose

int llclose(int fd, int type);

Esta função trata da terminação entre o emissor e o recetor, tal como da restauração da configuração da porta de série e o fecho desta.

No emissor, é enviada a trama de Supervisão DISC com utilizando a função **sendControlMessage** e aguardada a receção de outro DISC pela função **serialReadControl1**, sendo utilizado o sistema de retransmissão previamente mencionado. Após a receção do outro DISC é enviado um UA recorrendo, novamente, a **sendControlMessage**.

No recetor é aguardado um DISC, recorrendo a **serialReadControl1**. Após a receção deste DISC envia-se um DISC, utilizando a função **sendControlMessage,** e espera-se um UA, utilizando a função **serialReadControl1**, com o uso do sistema de retransmissão.

# 7. Protocolo de Aplicação

O protocolo de aplicação implementado está implementado da seguinte forma:

* O envio dos pacotes START e END indicam, respetivamente, o início e fim da transferência do ficheiro. Ambos contém o nome e tamanho do ficheiro enviado.
* O ficheiro é lido progressivamente pelo emissor, conforme vai enviando os dados, lendo o máximo possível de bytes a cada iteração. O número de bytes lidos não pode ultrapassar o número máximo atribuído ao campo de dados dos pacotes de dados
* Cada fragmento de dados é encapsulado com um header que contém o número de sequência, em módulo 255, do pacote e o tamanho do fragmento.
* O ficheiro é construído progressivamente pelo recetor, conforme vai recebendo os fragmentos de dados.

## buildControlPackage

unsigned int buildControlPackage(uint8\_t control\_field, unsigned int file\_size, uint8\_t \*file\_name, int file\_name\_length, uint8\_t \*dest);

Esta função cria um pacote START ou END, colocando-o no apontador *dest.* Recebe como argumentos o campo de controlo, para identificar o pacote pretendido, o nome do ficheiro e o tamanho deste. Este pacote será enviado pela função **llwrite**, pertencente ao protocolo de ligação.

## buildDataPackage

unsigned int buildDataPackage(unsigned int data\_size, uint8\_t \*data, uint8\_t \*dest);

Esta função cria um pacote de dados. Recebe como argumentos os dados, o seu tamanho e um apontador para armazenar o pacote construído. O header é então produzido, sendo constituído por:

* Campo de Controlo C, que identifica o pacote como sendo de dados;
* Número de Sequência N, em módulo 255;
* Os octetos L2 e L1, que representam o tamanho K do ficheiro

(K = L2 \* 256 + L1);

## createReceivedFile

int createReceivedFile(char \*file\_name)

Esta função verifica se o ficheiro a ser recebido pelo recetor já existe. Caso exista, tenta, sequencialmente, nomes alternativos até alcançar um que não esteja a ser utilizado. As alternativas são construídas acrescentando um sufixo ao nome do ficheiro (ex: pinguim(1).gif ). A função também distingue nomes que contenham extensões ou não, de modo a colocar o sufixo de forma adequada.

## sendMessage

int sendMessage(int bytes\_written, int fd, uint8\_t \*data\_package);

Esta função envia o pacote já criado pelo serial port, retornando TRUE se for necessário reenviá-lo, FALSE se não for, e terminando o programa se ocorrer time-out.

## fileDataReading

int fileDataReading(int fd, uint8\_t \*data\_package, enum appState \*state)

Esta função lê bytes do ficheiro até atingir o número máximo permitido por pacote ou até chegar ao fim do ficheiro. Se não houver mais bytes para ler, altera o estado da aplicação, senão constrói um pacote com os dados recebidos.

# 8. Validação

De modo a testar a aplicação desenvolvida, foram realizados os seguintes testes:

* Envio de ficheiros de diversos tamanhos;
* Geração de curto circuito durante o envio de um ficheiro;
* Interrupção da ligação por alguns segundos, aquando do envio do ficheiro;
* Envio de um ficheiro com variação na percentagem de erros simulados;
* Envio de um ficheiro com variação do tamanho do pacote;
* Envio de um ficheiro com variação das capacidades de ligação (Baudrate);

Todos os testes foram concluídos com sucesso.

# 9. Eficiência do Protocolo de Ligação de Dados

## Variação da Capacidade de Ligação

De acordo com os testes realizados, a eficiência do protocolo diminui com o aumento do baudrate.

Gráfico 1 - Eficiência com variação do baudrate

## Variação do Tempo de Propagação (T\_prop)

O aumento do tempo de propagação leva a uma diminuição da eficiência. O transporte de cada pacote provoca um atraso, que, acumulado, não é negligível, como comprova o gráfico.

Gráfico 2 - Eficiência com variação do tempo de propagação

## Variação do tamanho dos pacotes de dados

O gráfico abaixo demonstra que do aumento do tamanho dos pacotes de dados resulta o aumento da eficiência. Um maior número de dados transportados significa um menor número de tramas enviadas, o que faz com que o programa execute mais rapidamente.

Gráfico 3 - Eficiência com variação do tamanho dos pacotes de dados

## Variação de FER

O gráfico comprova que a ocorrência de erros no BCC1 e no BCC2 afetam negativamente a eficiência do programa. No caso de erros no BCC1, o efeito é severo, pois o recetor não responde, levando o emissor a aguardar um número previamente definido de segundos antes de reenviar a informação. Isto atrasa bastante a execução, e se a probabilidade de erros no BCC1 for alta, há a possibilidade de o recetor receber vários pacotes inválidos seguidos, o que pode provocar um time-out no emissor. No caso de erros no BCC2, o atraso é menor, visto que o programa reenvia imediatamente a trama.

Gráfico 4 - Eficiência com variação de FER

# 10. Conclusões

O tema do trabalho é o protocolo de ligação de dados e consiste em implementar um sistema de comunicação de dados entre dois sistemas ligados por um *serial port*. Este sistema deve ser fiável, garantindo o envio de dados sem erros.

Também foi desenvolvida uma aplicação, de modo a testar o protocolo realizado, sendo independente deste. Esta independência foi alcançada utilizando funções especiais, que serviram de interface protocolo-aplicação. Assim, na camada de ligação de dados não há qualquer processamento dos pacotes de dados a serem transportados na camada da aplicação, e na camada da aplicação não são conhecidos os detalhes da implementação do protocolo de ligação de dados. Apenas é conhecida a forma de acesso ao sistema implementado.

Em conclusão, o trabalho foi concluído com sucesso, cumprindo todos os objetivos, e permitiu aprofundar os conhecimentos teóricos e práticos dos temas abordados.

# 11. Anexo I - Código Fonte

## llfunctions.h

*#ifndef* LLFUNCTIONS\_H

*#define* LLFUNCTIONS\_H

*#include* "msg\_state\_machine.h"

*#include* <termios.h>

*#define* BIT(n) (0x01 << (n))

*#define* TRANSMITTER 0

*#define* RECEIVER 1

*#define* COMMAND 0

*#define* REPLY 1

*#define* FALSE 0

*#define* TRUE 1

*#define* MAX\_RETR 3

*#define* TIMEOUT 3

*#define* BAUDRATE B38400

*//ll return values*

*#define* TIMEOUT\_RET -1

*#define* RESEND\_RET -2

*//I FRAME data max size*

*#define* D\_MAX\_SIZE         255

*#define* D\_STUFFED\_MAX\_SIZE D\_MAX\_SIZE \* 2

*//I frame max size*

*#define* HEAD\_SIZE                4

*#define* F\_TAIL\_SIZE              2

*#define* F\_STUFFED\_TAIL\_SIZE      3

*#define* F\_STUFFED\_SIZE           HEAD\_SIZE + D\_STUFFED\_MAX\_SIZE + F\_STUFFED\_TAIL\_SIZE

*//I frame return values*

*#define* DATA\_ERROR -1

*//reply types*

*#define* RR  1

*#define* REJ 2

*//discard frame*

*#define* DISCARD -1

*/\*\**

*\* @brief Indicates the alarm call through a global flag and increments the alarm counter*

*\* @param sig signal identifier.*

*\*/*

void alarmHandler(int sig);

*/\*\**

*\* @brief Assembles the control frame and sends it to the given file descriptor*

*\**

*\* @param fd     The file descriptor.*

*\* @param C      Frame control field.*

*\* @param A      Frame adress field.*

*\*/*

void sendControlMessage(int fd, uint8\_t C, uint8\_t A);

*/\*\**

*\* @brief Assembles the I frame and sends it to the given file descriptor. Handles stuffing, head and tail building.*

*\**

*\* @param fd     The file descriptor.*

*\* @param c\_num  I Frame control field. Can either be C\_I(0) or C\_I(1).*

*\* @param info   Information to be transferred, not yet stuffed.*

*\* @param info\_length   Length of information (number of bytes/octets)*

*\* @return int   Number of bytes written (after stuffing).*

*\*/*

int sendICommand(int fd, int c\_num, uint8\_t \*info, unsigned int info\_length);

*/\*\**

*\* @brief Reads the I frame from the file descriptor and retrieves the message it carries. Handles destuffing, tail reception and validation.*

*\**

*\* @param fd     The file descriptor.*

*\* @param buffer Where the message will be stored (unstuffed).*

*\* @return int   Number of bytes read (after destuffing), or DATA\_ERROR if the data is not valid.*

*\*/*

int readIFrame(int fd, uint8\_t \*buffer);

*/\*\**

*\* @brief Assembles I frame's Data BCC, obtained through the exclusive OR of all the data's octets.*

*\**

*\* @param info         The Data.*

*\* @param info\_length  Data's length, in bytes.*

*\* @return uint8\_t     Octet obtained through the XOR of all data's octets*

*\*/*

uint8\_t dataBCC(uint8\_t \*info, unsigned int info\_length);

*/\*\**

*\* @brief Receives a message and builds the corresponding stuffed message*

*\**

*\* @param msg         The Data.*

*\* @param msg\_length  Data's length, in bytes.*

*\* @param stuffed\_msg Where the stuffed message will be stored.*

*\* @return int        Size of stuffed message.*

*\*/*

int byteStuffing(uint8\_t \*msg, unsigned int msg\_length, uint8\_t \*stuffed\_msg);

*/\*\**

*\* @brief Receives a stuffed message and rebuilds the original message*

*\**

*\* @param stuffed\_msg Stuffed message.*

*\* @param msg\_length  Data's length, in bytes.*

*\* @param destuff     Where the original message will be stored*

*\* @return int        Size of original message.*

*\*/*

int destuffing(uint8\_t \*stuffed\_msg, unsigned int msg\_length, uint8\_t \*destuff);

*/\*\**

*\* @brief Reads from the file descriptor until it receives the ending flag or exceeds the message max size. Handles the*

*\* retrieval of the I frame's half after the tail (data + head) for ulterior processing.*

*\**

*\* @param fd          The file descriptor.*

*\* @param buffer      Where the received data will be stored.*

*\* @return int        Number of bytes read, a negative value if it fails or exceeds the max size.*

*\*/*

int getFrame(int fd, uint8\_t \*buffer);

*/\*\**

*\* @brief Compares the received Data BCC with a BCC calculated with the received data to check if there were transmission errors.*

*\**

*\* @param msg     The Data.*

*\* @param length  Data's length, in bytes.*

*\* @param BCC     BCC read from the serial port, used in error checking.*

*\* @return int    0 if the BCCs differ (due to a transmission error), other value if they're the same.*

*\*/*

int checkBCC(uint8\_t \*msg, unsigned length, uint8\_t BCC);

*/\*\**

*\* @brief Opens the serial port and sets the signal handler, then attempts to establish the connection.*

*\* a) type == TRANSMITTER*

*\**

*\* Sends the SET and waits for the UA reply. When SET is sent, an alarm is called, and if the reply is not received*

*\* within TIMEOUT seconds it resends the SET command. It keeps re-sending until it receives the reply or has attempted*

*\* MAX\_RETR times.*

*\**

*\* b) type == RECEIVER*

*\* Awaits the SET command and sends the UA reply when it receives it.*

*\**

*\* @param port    The Port to be opened.*

*\* @param type    App type indentifier (TRANSMITTER or RECEIVER).*

*\* @return int    Returns a file descriptor if it sucessfully establishes the connection, 0 if there's an error when acessing the serial*

*\* port.*

*\* a) returns TIMEOUT\_RET if it times out while sending the SET command*

*\*/*

int llopen(char \*port, int type);

*/\*\**

*\* @brief Sends a message through the serial port. Uses Ns to synchronize with the receiver.*

*\* Sends the I frame and waits for the reply. When SET is sent, an alarm is called, and if the reply is not received*

*\* within TIMEOUT seconds it resends the SET command. It keeps re-sending until it receives the reply or has attempted*

*\* MAX\_RETR times.*

*\* @param fd      File descriptor.*

*\* @param buffer  Message to be sent through the serial port.*

*\* @param length  Message's size.*

*\* @return int    Returns the number of bytes written on a sucessful transmittion, RESEND\_RET if the same data has to be sent again*

*\* or TIMEOUT\_RET if it times out.*

*\*/*

int llwrite(int fd, uint8\_t \*buffer, int length);

*/\*\**

*\* @brief Receives a message from the serial port. Uses Nr to synchronize with the transmitter.*

*\* Receives the I frame. If the data is ok and carries the expected Ns then replies with an RR reply;*

*\* if the data is a duplicate (the Ns is different than expected) it also sends the RR reply.*

*\* If the data has errors it sends the REJ reply.*

*\* @param fd      File descriptor.*

*\* @param buffer  Where the received message will be stored.*

*\* @param length  Message's size.*

*\* @return int    Returns the number of bytes written on a sucessful transmittion,*

*\* DISCARD if the data has errors or is a duplicate.*

*\*/*

int llread(int fd, uint8\_t \*buffer);

*/\*\**

*\* @brief Attempts to terminate the connection.*

*\* a) type == TRANSMITTER*

*\**

*\* Sends the DISC command and waits for the DISC reply. When DISC is sent, an alarm is called, and if the reply is not received*

*\* within TIMEOUT seconds it resends the DISC command. It keeps re-sending until it receives the reply or has attempted*

*\* MAX\_RETR times. If it receives the DISC reply, it sends an UA reply to the receiver.*

*\**

*\* b) type == RECEIVER*

*\* Awaits the DISC command and sends the DISC reply when it receives it.*

*\* It then waits for the UA reply. When DISC is sent, an alarm is called, and if the reply is not received*

*\* within TIMEOUT seconds it resends DISC. It keeps re-sending until it receives the reply or has attempted*

*\* MAX\_RETR times.*

*\* @param port    The Port to be opened.*

*\* @param type    App type indentifier (TRANSMITTER or RECEIVER).*

*\* @return int    Returns 1 if sucessful, TIMEOUT\_RET if it times out.*

*\*/*

int llclose(int fd, int type);

*/\*\**

*\* @brief Reads the serial until either a control frame or I frame's header with the specified A and C fields has been found.*

*\* It uses a state machine to process the received bytes.*

*\* It can search for two C's at the same time, testing each one in a different state machine.*

*\* @param fd      File descriptor.*

*\* @param c       Desired control field.*

*\* @param a       Desired adress field.*

*\* @param c2      Another desired control field, used when there are two possible values for C.*

*\* @param headerI Determines if it's looking for a control frame or an I frame's header. If TRUE, it stops at the BCC,*

*\* otherwise looks for the end flag.*

*\* @return int    Returns 1 if it found the desired sequence, while using c; 2 if it did so while using c2;*

*\* FALSE if it didn't find anything when the alarm was called, -1 if there was an error reading.*

*\*/*

int serialReadControl2(int fd, uint8\_t c, uint8\_t a, uint8\_t c2, int headerI);

*/\*\**

*\* @brief serialReadControl version which can only search for a control frame and test a single C octet.*

*\*/*

int serialReadControl1(int fd, uint8\_t c, uint8\_t a);

*/\*\**

*\* @brief initializes the port*

*\* \*/*

void initPort(int fd);

*#endif*

## llfunctions.c

*#include* "llfunctions.h"

*#include* <sys/types.h>

*#include* <sys/stat.h>

*#include* <fcntl.h>

*#include* <unistd.h>

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <string.h>

*#include* <termios.h>

*#include* <stdint.h>

*#include* <signal.h>

*#include* <errno.h>

int numAlarms = 0;

int alarmFlag = FALSE;

struct termios oldtio, newtio;

int curr\_ns = 0;

void alarmHandler(int sig)

{

*if* (sig == SIGALRM)

    {

        numAlarms++;

        alarmFlag = TRUE;

        printf("alarm nº %d called\n", numAlarms);

    }

}

void sendControlMessage(int fd, uint8\_t C, uint8\_t A)

{

    uint8\_t msg[5];

    msg[0] = F\_FLAG;

    msg[1] = A;

    msg[2] = C;

    msg[3] = A ^ C;

    msg[4] = F\_FLAG;

    write(fd, msg, 5);

}

int sendICommand(int fd, int c\_num, uint8\_t \*info, unsigned int info\_length)

{

    uint8\_t msg[F\_STUFFED\_SIZE];

    uint8\_t unstuffed\_bcc;

    msg[0] = F\_FLAG;

    msg[1] = A1\_;

    msg[2] = C\_I(c\_num);   *//C*

    msg[3] = A1\_ ^ msg[2]; *//BCC 1*

*// for (unsigned int i = 0; i < info\_length; i++)*

*// {*

*//     printf("%x", info[i]);*

*// }*

*// printf("\n");*

    unsigned int header\_size = 4;

    unsigned int stuffed\_msg\_size = byteStuffing(info, info\_length, msg + header\_size); *//stuffs the message*

    unstuffed\_bcc = dataBCC(info, info\_length); *//BCC 2, XOR of all the data octets*

    unsigned int stuffed\_bcc\_size = byteStuffing(&unstuffed\_bcc, 1, msg + header\_size + stuffed\_msg\_size); *//stuffs the bcc*

    msg[stuffed\_msg\_size + header\_size + stuffed\_bcc\_size] = F\_FLAG;

    unsigned int tail\_size = stuffed\_bcc\_size + 1;

    unsigned int frame\_size = header\_size + stuffed\_msg\_size + tail\_size;

*// for (unsigned int i = 0; i < frame\_size; i++)*

*// {*

*//     printf("%x", msg[i]);*

*// }*

*// printf("\n");*

*return* write(fd, msg, frame\_size);

}

int readIFrame(int fd, uint8\_t \*buffer)

{

    uint8\_t stuffed\_msg\_w\_tail[D\_STUFFED\_MAX\_SIZE + F\_STUFFED\_TAIL\_SIZE];

    uint8\_t unstuffed\_msg\_w\_tail[D\_MAX\_SIZE + F\_TAIL\_SIZE];

    int stuffed\_msg\_size = getFrame(fd, stuffed\_msg\_w\_tail);

*// printf("STUFFED\n");*

*// for (unsigned int i = 0; i < stuffed\_msg\_size; i++)*

*// {*

*//     printf("%x", stuffed\_msg\_w\_tail[i]);*

*// }*

*// printf("\n");*

*if* (stuffed\_msg\_size <= 0)

    {

*return* DATA\_ERROR;

    }

    int unstuffed\_msg\_size = destuffing(stuffed\_msg\_w\_tail, stuffed\_msg\_size, unstuffed\_msg\_w\_tail) - F\_TAIL\_SIZE; *//destuffs message*

*if* (unstuffed\_msg\_size <= 0)

    {

*return* DATA\_ERROR;

    }

    uint8\_t bcc = unstuffed\_msg\_w\_tail[unstuffed\_msg\_size]; *//retrieves BCC from tail*

*// printf("UNSTUFFED\n");*

*// for (unsigned int i = 0; i < unstuffed\_msg\_size + F\_TAIL\_SIZE; i++)*

*// {*

*//     printf("%x", unstuffed\_msg\_w\_tail[i]);*

*// }*

*// printf("\n");*

    int is\_bcc\_valid = checkBCC(unstuffed\_msg\_w\_tail, unstuffed\_msg\_size, bcc); *//compares the received BCC and the BCC calculated with the received data*

*//printf("is bcc valid? %d\n", is\_bcc\_valid);*

*//error simulation*

*if* (hasError(BCC2ERR))

        is\_bcc\_valid = 0;

*if* (is\_bcc\_valid)

    {

        memcpy(buffer, unstuffed\_msg\_w\_tail, unstuffed\_msg\_size);

*return* unstuffed\_msg\_size;

    }

*else*

    {

*return* DATA\_ERROR;

    }

}

uint8\_t dataBCC(uint8\_t \*info, unsigned int info\_length)

{

    uint8\_t bcc = info[0];

*for* (unsigned i = 1; i < info\_length; i++)

    {

        bcc = bcc ^ info[i];

    }

*return* bcc;

}

*//builds stuffed msg, returns size of msg after stuffing*

int byteStuffing(uint8\_t \*msg, unsigned int msg\_length, uint8\_t \*stuffed\_msg)

{

    unsigned j = 0;

*for* (unsigned i = 0; i < msg\_length; i++)

    {

        uint8\_t curr\_char = msg[i];

*//printf("%x\n", curr\_char);*

*if* (curr\_char == F\_FLAG)

        {

            stuffed\_msg[j++] = ESC\_OCT;

            stuffed\_msg[j++] = FLAG\_STUFF;

        }

*else* *if* (curr\_char == ESC\_OCT)

        {

            stuffed\_msg[j++] = ESC\_OCT;

            stuffed\_msg[j++] = ESC\_STUFF;

        }

*else*

        {

            stuffed\_msg[j++] = curr\_char;

        }

*//printf("%d - size\n", j);*

    }

*return* j;

}

*//builds a destuffed message. returns size of destuffed msg*

int destuffing(uint8\_t \*stuffed\_msg, unsigned int msg\_length, uint8\_t \*destuffed\_msg)

{

    unsigned j = 0;

*// printf("DESTUFFING\n");*

*for* (unsigned i = 0; i < msg\_length; i++)

    {

        uint8\_t curr\_char = stuffed\_msg[i];

*// printf("%x", curr\_char);*

*if* (curr\_char == ESC\_OCT)

        {

            i++;

            uint8\_t next\_char = stuffed\_msg[i];

*// printf("%x", next\_char);*

*if* (next\_char == FLAG\_STUFF)

            {

                destuffed\_msg[j] = F\_FLAG;

            }

*else* *if* (next\_char == ESC\_STUFF)

            {

                destuffed\_msg[j] = ESC\_OCT;

            }

        }

*else*

        {

            destuffed\_msg[j] = curr\_char;

        }

        j++;

    }

*// printf("\n");*

*// printf("DESTUFFING ENDED\n");*

*return* j;

}

int getFrame(int fd, uint8\_t \*buffer)

{

    int res = 0;

    uint8\_t curr\_char;

*while* (!alarmFlag)

    { */\* loop for input \*/*

*if* (res >= D\_STUFFED\_MAX\_SIZE + F\_STUFFED\_TAIL\_SIZE)

*return* -2;

*if* (read(fd, &curr\_char, 1))

        {

            buffer[res] = curr\_char;

*// printf("%x", buffer[res]);*

            res++;

        }

*else*

        {

*return* -1;

        }

*if* (curr\_char == F\_FLAG && res > 0)

*break*;

    }

*// printf("\n");*

*// printf("RES - %d\n\n", res);*

*return* res;

}

int checkBCC(uint8\_t \*msg, unsigned length, uint8\_t bcc)

{

    uint8\_t data\_bcc = msg[0];

*for* (unsigned i = 1; i < length; i++)

    {

        data\_bcc = data\_bcc ^ msg[i];

    }

*return* (data\_bcc == bcc);

}

int llopen(char \*port, int type)

{

    int fd;

    fd = open(port, O\_RDWR | O\_NOCTTY);

*if* (fd < 0)

*return* 0;

    initPort(fd);

    signal(SIGALRM, alarmHandler);

    siginterrupt(SIGALRM, 1);

*if* (type == TRANSMITTER)

    {

*//sprintf(buf, "%lx", frame);*

*do*

        {

            alarmFlag = FALSE;

            printf("Sending SET command - attempt nº %d\n", numAlarms + 1);

            sendControlMessage(fd, C\_SET, A1\_);

            alarm(TIMEOUT);

*if* (serialReadControl1(fd, C\_UA, A1\_) > 0)

            {

                alarm(0);

                printf("Received UA reply\n");

*break*;

            }

        } *while* (alarmFlag && numAlarms < MAX\_RETR);

    }

*else*

    {

*if* (serialReadControl1(fd, C\_SET, A1\_) > 0)

        {

            printf("Received SET command\n");

            sendControlMessage(fd, C\_UA, A1\_);

            printf("Sent UA reply\n");

        }

    }

    alarmFlag = FALSE;

*if* (numAlarms == MAX\_RETR)

    {

        numAlarms = 0;

*return* TIMEOUT\_RET;

    }

*else*

    {

        numAlarms = 0;

*return* fd;

    }

}

int llwrite(int fd, uint8\_t \*buffer, int length)

{

*//used to synch the transmitter and receiver*

    int curr\_nr = (curr\_ns + 1) % 2;

    unsigned bytes\_written = 0;

    int reply\_type = 0;

    printf("%d - CURR NS\n", curr\_ns);

*do*

    {

        alarmFlag = FALSE;

        printf("Sending I frame - attempt nº %d\n", numAlarms + 1);

        bytes\_written = sendICommand(fd, curr\_ns, buffer, length);

        alarm(TIMEOUT);

        reply\_type = serialReadControl2(fd, C\_RR(curr\_nr), A1\_, C\_REJ(curr\_nr), FALSE);

*if* (reply\_type > 0)

        {

*//Received a valid reply*

            alarm(0);

*break*;

        }

    } *while* (alarmFlag && numAlarms < MAX\_RETR);

    alarmFlag = FALSE;

    numAlarms = 0;

*if* (reply\_type == RR)

    {

*//The I frame was sucessfully received*

        printf("Received RR reply\n");

        curr\_ns = (curr\_ns + 1) % 2;

*return* bytes\_written;

    }

*else* *if* (reply\_type == REJ)

    {

*//The I frame was either received with errors or a duplicate*

        printf("Received REJ reply\n");

*return* RESEND\_RET;

    }

*else*

    {

        printf("Timed out while sending I frame\n");

*return* TIMEOUT\_RET;

    }

}

int llread(int fd, uint8\_t \*buffer)

{

    int curr\_nr = (curr\_ns + 1) % 2;

    printf("%d - CURR NS\n", curr\_ns);

    int command\_type = serialReadControl2(fd, C\_I(curr\_ns), A1\_, C\_I(curr\_nr), TRUE);

    printf("COMMAND\_TYPE HEADER RESULT = %d\n", command\_type);

    int msg\_size = DISCARD;

*// sleep(1);*

*if* (command\_type > 0)

    {

        msg\_size = readIFrame(fd, buffer);

        printf("msg size = %d\n", msg\_size);

*if* (command\_type == 1)

        {

*if* (msg\_size >= 0)

            {

                printf("Data is ok, sent RR reply\n");

                sendControlMessage(fd, C\_RR(curr\_nr), A1\_);

                curr\_ns = (curr\_ns + 1) % 2;

            }

*else*

            {

                printf("Error in data, Sent REJ reply\n");

                msg\_size = DISCARD;

                sendControlMessage(fd, C\_REJ(curr\_nr), A1\_);

            }

        }

*else* *if* (command\_type == 2)

        {

            printf("Frame is duplicate, sent RR reply\n");

            msg\_size = DISCARD;

            sendControlMessage(fd, C\_RR(curr\_ns), A1\_);

        }

    }

*return* msg\_size;

}

int llclose(int fd, int type)

{

*if* (type == TRANSMITTER)

    {

*do*

        {

            alarmFlag = FALSE;

            printf("Sending DISC command - attempt nº %d\n", numAlarms + 1);

            sendControlMessage(fd, C\_DISC, A1\_);

            alarm(TIMEOUT);

*if* (serialReadControl1(fd, C\_DISC, A2\_) > 0)

            {

                alarm(0);

                printf("Received DISC command\n");

                sendControlMessage(fd, C\_UA, A2\_);

                printf("Sent UA reply\n");

*break*;

            }

        } *while* (alarmFlag && numAlarms < MAX\_RETR);

    }

*else*

    {

*if* (serialReadControl1(fd, C\_DISC, A1\_) > 0)

        {

            printf("Received DISC command\n");

*do*

            {

                alarmFlag = FALSE;

                printf("Sending DISC command - attempt nº %d\n", numAlarms + 1);

                sendControlMessage(fd, C\_DISC, A2\_);

                alarm(TIMEOUT);

*if* (serialReadControl1(fd, C\_UA, A2\_) > 0)

                {

                    alarm(0);

                    printf("Received UA reply\n");

*break*;

                }

            } *while* (alarmFlag && numAlarms < MAX\_RETR);

        }

    }

    alarmFlag = FALSE;

*if* (tcsetattr(fd, TCSANOW, &oldtio) == -1)

    {

        perror("tcsetattr");

        exit(-1);

    }

    close(fd);

*if* (numAlarms == MAX\_RETR)

    {

        numAlarms = 0;

*return* TIMEOUT\_RET;

    }

*else*

    {

        numAlarms = 0;

*return* 1;

    }

}

int serialReadControl2(int fd, uint8\_t c, uint8\_t a, uint8\_t c2, int headerI)

{

    enum state curr\_state = START\_S;

    enum state curr\_state2 = START\_S;

    uint8\_t curr\_byte;

*while* (curr\_state != STOP\_S && curr\_state2 != STOP\_S && !alarmFlag)

    { */\* loop for input \*/*

*// if (headerI)*

*//     printf("curr\_byte = %x, curr\_state1 = %d, curr\_state2 = %d\n", curr\_byte, curr\_state, curr\_state2);*

*if* (read(fd, &curr\_byte, 1) < 0)

        {

*if* (errno != EINTR)

*return* -1;

        }

        stateMachineOpen(&curr\_state, &curr\_byte, c, a, headerI);

*if* (c2 != 0x0F)

        {

            stateMachineOpen(&curr\_state2, &curr\_byte, c2, a, headerI);

        }

    }

*if* (curr\_state == STOP\_S)

*return* 1;

*else* *if* (curr\_state2 == STOP\_S)

*return* 2;

*else*

*return* FALSE;

}

int serialReadControl1(int fd, uint8\_t c, uint8\_t a)

{

*return* serialReadControl2(fd, c, a, 0x0F, FALSE);

}

void initPort(int fd)

{

*if* (tcgetattr(fd, &oldtio) == -1)

    { */\* save current port settings \*/*

        perror("tcgetattr");

        exit(-1);

    }

    bzero(&newtio, sizeof(newtio));

    newtio.c\_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;

    newtio.c\_iflag = IGNPAR;

    newtio.c\_oflag = 0;

*/\* set input mode (non-canonical, no echo,...) \*/*

    newtio.c\_lflag = 0;

    newtio.c\_cc[VTIME] = 0; */\* inter-character timer unused \*/*

    newtio.c\_cc[VMIN] = 1;  */\* blocking read until 5 chars received \*/*

*/\**

*VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a*

*leitura do(s) pr�ximo(s) caracter(es)*

*\*/*

    tcflush(fd, TCIOFLUSH);

*if* (tcsetattr(fd, TCSANOW, &newtio) == -1)

    {

        perror("tcsetattr");

        exit(-1);

    }

    printf("New termios structure set\n");

}

## msg\_state\_machine.h

*/\* State Machine for messages communications \*/*

*#ifndef* MSG\_STATE\_MACHINE\_H

*#define* MSG\_STATE\_MACHINE\_H

*#include* <stdint.h>

*/\*\*/*

*#define* F\_FLAG 0x7E

*/\* 00000011 (0x03) em Comandos enviados pelo Emissor e Respostas enviadas pelo Receptor \*/*

*#define* A1\_ 0X03

*/\* 00000001 (0x01) em Comandos enviados pelo Receptor e Respostas enviadas pelo Emissor \*/*

*#define* A2\_ 0X01

*//comandos, n = 0/1*

*#define* C\_SET 0x03

*#define* C\_I(n)   ((n) << (6))

*#define* C\_DISC 0x0B

*//respostas, n = 0/1*

*#define* C\_UA 0x06

*#define* C\_RR(n)    (((n) << (7)) | 0x05)

*#define* C\_REJ(n)   (((n) << (7)) | 0x01)

*//stuffing*

*#define* ESC\_OCT     0x7D

*#define* FLAG\_STUFF  0x5E

*#define* ESC\_STUFF   0x5D

*/\*\*/*

*//*

*#define* BCC1ERR 0

*#define* BCC2ERR 0

enum state {START\_S, FLAG\_RCV, A\_RCV, C\_RCV, BCC\_OK, STOP\_S};

void stateMachineOpen(enum state \*state, uint8\_t \*received\_char, uint8\_t c, uint8\_t a, int headerI);

int hasError(int error\_chance);

*#endif //MSG\_STATE\_MACHINE\_H*

## msg\_state\_machine.c

*/\* State Machine for messages communications \*/*

*#include* "msg\_state\_machine.h"

*#include* <stdio.h>

*#include* <stdlib.h>

*#define* FALSE 0

*#define* TRUE 1

int hasError(int error\_chance)

{

    int has\_error = 0;

*if* (error\_chance > 0)

    {

        int random\_number = (rand() % 100) + 1;

        printf("RANDOM NUMBER - %d\n", random\_number);

        has\_error = (error\_chance >= random\_number);

        printf("ERRORI - %d\n", has\_error);

    }

*return* has\_error;

}

uint8\_t get\_bcc(uint8\_t A, uint8\_t C, int headerI)

{

*if* (headerI)

    {

*if* (hasError(BCC1ERR))

*return* A & C;

    }

*return* A ^ C;

}

void stateMachineOpen(enum state \*state, uint8\_t \*received\_char, uint8\_t c, uint8\_t a, int headerI){

*switch* (\*state){

*//flag*

*case* START\_S:

*if* (\*received\_char == F\_FLAG)

                \*state = FLAG\_RCV;

*//printf("START - RCV = %x\n", \*received\_char);*

*break*;

*case* FLAG\_RCV:

*if* (\*received\_char == a)

                \*state = A\_RCV;

*else* *if* (\*received\_char != F\_FLAG)

                \*state = START\_S;

*//printf("FLAG - RCV = %x\n", \*received\_char);*

*break*;

*case* A\_RCV:

*if* (\*received\_char == c)

                \*state = C\_RCV;

*else* *if* (\*received\_char == F\_FLAG)

                \*state = 1;

*else* \*state = START\_S;

*//printf("A - RCV = %x\n", \*received\_char);*

*break*;

*case* C\_RCV:

*if* (\*received\_char == get\_bcc(a, c, headerI))

            {

*if* (headerI != TRUE)

                    \*state = BCC\_OK; *//if it is a control frame, the BCC is not the last octet*

*else*

                    \*state = STOP\_S; *//if it is an I header, the BCC is the last octet*

            }

*else* *if* (\*received\_char == F\_FLAG)

                \*state = FLAG\_RCV;

*else*

                \*state = START\_S;

*//printf("C - RCV = %x\n", \*received\_char);*

*break*;

*case* BCC\_OK:

*if* (\*received\_char == F\_FLAG)

                \*state = STOP\_S;

*else* \*state = START\_S;

*//printf("BCC - RCV = %x\n", \*received\_char);*

*break*;

*default*:

*break*;

    }

}

## app\_package\_handling.h

*#ifndef* APP\_PACKAGE\_HANDLING\_H

*#define* APP\_PACKAGE\_HANDLING\_H

*#include* <stdint.h>

*//C*

*#define* C\_DATA     0x01

*#define* C\_START    0x02

*#define* C\_END      0x03

*//control TLV*

*#define* T\_SIZE     0x00

*#define* T\_NAME     0x01

*#define* PACK\_MAX\_SIZE  255

*#define* PACKAGE\_H\_SIZE 4

*#define* PACKAGE\_DATA\_SIZE PACK\_MAX\_SIZE - PACKAGE\_H\_SIZE

*//end package returns*

*#define* INVALID\_END\_PACKAGE -1

*#define* VALID\_END\_PACKAGE   -2

*#define* INVALID\_PACKAGE     -3

*//file*

*#define* FILE\_NAME\_MAX\_SIZE     255

*#define* EXTENSION\_MAX\_SIZE     16

*#define* ORIGINAL\_FILE\_MAX\_SIZE PACK\_MAX\_SIZE - 10

*//START: start of file data sending, DATA: sending file data, END: Ending Program, STOP: Program has ended.*

enum appState {START, DATA, END, STOP\_};

*/\*\**

*\* @brief Changes the state to the next*

*\*/*

void stateMachineApp(enum appState \*state);

*/\*\**

*\* @brief Builds a TLV format message*

*\* @param type     T.*

*\* @param length   L.*

*\* @param value    V.*

*\* @param dest     Where the TLV message will be stored.*

*\* @return int     Size of tlv message (in bytes).*

*\*/*

unsigned int writeTlv(uint8\_t type, uint8\_t length, uint8\_t \*value, uint8\_t \*dest);

*/\*\**

*\* @brief Builds a Control Package, which is made of a Control Field and*

*\* two TLV messages, containing the file size and file name*

*\* @param dest     Where the Control Package will be stored.*

*\* @return int     Size of Control Package (in bytes).*

*\*/*

unsigned int buildControlPackage(uint8\_t control\_field, unsigned int file\_size, uint8\_t \*file\_name, int file\_name\_length, uint8\_t \*dest);

*/\*\**

*\* @brief Builds a Data Package, which is made of a Control Field, a sequence number (N),*

*\* two octets which contain the data's size, (L1 and L2), and the data.*

*\* @param dest     Where the Data Package will be stored.*

*\* @return int     Size of Data Package (in bytes).*

*\*/*

unsigned int buildDataPackage(unsigned int data\_size, uint8\_t \*data, uint8\_t \*dest);

*/\*\**

*\* @brief Checks if the received end\_package is valid (has the same TLV data as the start\_package)*

*\* @return int     TRUE if is valid, FALSE if it isn't.*

*\*/*

int isEndPackage(uint8\_t \*package, uint8\_t \*start\_package, unsigned int start\_package\_length);

*/\*\**

*\* @brief Reads the Start Package, checking the control field and retrieving the values sent in TLV format*

*\* @param package       Received Package.*

*\* @param package\_size  Received Package's size (in bytes).*

*\* @param file\_size     Where the file size will be stored.*

*\* @param file\_name     Where the file name will be stored.*

*\* @return int          Positive value if it's a start package, negative otherwise*

*\*/*

int readStartPackage(uint8\_t \*package, unsigned int package\_size, uint8\_t \*file\_size, uint8\_t \*file\_name);

int readDataPackage(uint8\_t \*package, uint8\_t \*dest, uint8\_t expected\_N);

int readPackage(uint8\_t \*package, uint8\_t \*dest, uint8\_t expected\_N, uint8\_t \*start\_package, unsigned int start\_package\_length);

void display\_completion(unsigned int file\_size, unsigned int curr\_size);

*#endif*

## app\_package\_handling.c

*#include* <unistd.h>

*#include* <stdlib.h>

*#include* <string.h>

*#include* <stdio.h>

*#include* "app\_package\_handling.h"

*#define* FALSE 0

*#define* TRUE 1

void stateMachineApp(enum appState \*state)

{

*switch* (\*state)

    {

*case* START:

        \*state = DATA;

*break*;

*case* DATA:

        \*state = END;

*break*;

*case* END:

        \*state = STOP\_;

*break*;

*default*:

*break*;

    }

}

unsigned int writeTlv(uint8\_t type, uint8\_t length, uint8\_t \*value, uint8\_t \*dest)

{

    dest[0] = type;

    dest[1] = length;

    memcpy(dest + 2, value, length);

*return* length + 2;

}

unsigned int buildControlPackage(uint8\_t control\_field, unsigned int file\_size, uint8\_t \*file\_name, int file\_name\_length, uint8\_t \*dest)

{

    dest[0] = control\_field;

    unsigned int total\_package\_size = 1;

*//tlv file size*

*//printf("SIZE OF FILE SIZE - %x\n", sizeof(file\_size));*

    total\_package\_size += writeTlv(T\_SIZE, sizeof(file\_size), (uint8\_t \*) &file\_size, dest + total\_package\_size);

*//printf("FILE SIZE - %x\n", file\_size);*

*//tlv file name*

    total\_package\_size += writeTlv(T\_NAME, file\_name\_length + 1, file\_name, dest + total\_package\_size);

*// for (int i = 0; i < total\_package\_size; i++){*

*//     printf("%x ", dest[i]);*

*// }*

*// printf("\n");*

*return* total\_package\_size;

}

unsigned int buildDataPackage(unsigned int data\_size, uint8\_t \*data, uint8\_t \*dest)

{

    static uint8\_t sequence\_number\_N = 0;

    dest[0] = C\_DATA;

    dest[1] = sequence\_number\_N;

    uint8\_t l1 = data\_size % 256;

    uint8\_t l2 = (data\_size - l1) / 256;

    dest[2] = l2;

    dest[3] = l1;

    memcpy(dest + PACKAGE\_H\_SIZE, data, data\_size);

    sequence\_number\_N = (sequence\_number\_N + 1) % 256; *//update sequence number*

*return* PACKAGE\_H\_SIZE + data\_size;

}

int isEndPackage(uint8\_t \*package, uint8\_t \*start\_package, unsigned int start\_package\_length)

{

*for* (unsigned int i = 1; i < start\_package\_length; i++)

    {

*// printf("pack: %x vs start pack: %x\n", package[i], start\_package[i]);*

*if* (package[i] != start\_package[i])

*return* FALSE;

    }

*return* TRUE;

}

int readStartPackage(uint8\_t \*package, unsigned int package\_size, uint8\_t \*file\_size, uint8\_t \*file\_name)

{

*// printf("C = %x = %x?\n", package[0], C\_START);*

*if* (package[0] != C\_START)

*return* -1;

*//printf("package size - %d\n", package\_size);*

    unsigned int curr\_pack\_byte = 1;

*while* (curr\_pack\_byte < package\_size)

    {

        uint8\_t type = package[curr\_pack\_byte++];

*//printf("%x | ", package[curr\_pack\_byte - 1]);*

        uint8\_t length = package[curr\_pack\_byte++];

*//printf("%x | ", package[curr\_pack\_byte - 1]);*

*for* (unsigned i = 0; i < length; i++)

        {

*if* (type == T\_SIZE)

                file\_size[i] = package[curr\_pack\_byte++];

*else* *if* (type == T\_NAME)

                file\_name[i] = package[curr\_pack\_byte++];

*//printf("%x | ", package[curr\_pack\_byte - 1]);*

        }

*//printf("\n");*

    }

*//printf("\n");*

*return* curr\_pack\_byte;

}

int readDataPackage(uint8\_t \*package, uint8\_t \*dest, uint8\_t expected\_N){

    uint8\_t read\_sequence\_N = package[1];

    printf("Received Sequence N: %d\n", read\_sequence\_N);

*if* (expected\_N != read\_sequence\_N){

*return* -1;

    }

    uint8\_t l2 = package[2];

    uint8\_t l1 = package[3];

    int data\_length = 256 \* l2 + l1;

*for* (int i = 0; i < data\_length; i++)

    {

        dest[i] = package[i + 4];

*// printf("%x", dest[i]);*

    }

*// printf("\n");*

*return* data\_length;

}

int readPackage(uint8\_t \*package, uint8\_t \*dest, uint8\_t expected\_N, uint8\_t \*start\_package, unsigned int start\_package\_length)

{

*if* (package[0] == C\_DATA)

    {

        printf("Received Data Package\n");

*return* readDataPackage(package, dest, expected\_N);

    }

*else* *if* (package[0] == C\_END)

    {

*if* (isEndPackage(package, start\_package, start\_package\_length))

        {

            printf("Received Valid End Package\n");

*return* VALID\_END\_PACKAGE;

        }

*else*

        {

            printf("Received Invalid End Package\n");

*return* INVALID\_END\_PACKAGE;

        }

    }

    printf("Received Invalid Package\n");

*return* INVALID\_PACKAGE;

}

void display\_completion(unsigned int file\_size, unsigned int curr\_size){

    float file\_in\_packages = (file\_size / ((unsigned ) PACKAGE\_DATA\_SIZE));

    int percentage = ((curr\_size / file\_in\_packages) \* 100);

    int squares = percentage / 5;

    printf("%d%% completed => [ ", percentage);

*for* (int i = 0; i < 20; i++){

*if* (i < squares)

           printf("+ ");

*else* printf("- ");

    }

    printf("]\n");

}

## noncanonical.c

*/\*Non-Canonical Input Processing\*/*

*#include* <sys/types.h>

*#include* <sys/stat.h>

*#include* <fcntl.h>

*#include* <termios.h>

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <string.h>

*#include* <unistd.h>

*#include* <errno.h>

*#include* "llfunctions.h"

*#include* "app\_package\_handling.h"

*#include* <sys/time.h>

*#include* <time.h>

*#define* \_POSIX\_SOURCE 1 */\* POSIX compliant source \*/*

volatile int STOP = FALSE;

int isFileSizeExpected(int fd, unsigned int file\_size){

  struct stat st;

  fstat(fd, &st); *//retrieves size of file*

*return* (st.st\_size == file\_size);

}

int getLastCharPos(char \*str, char character){

  unsigned int str\_size = strlen(str);

*for* (int i = str\_size - 1; i >= 0; i--){

*if* (str[i] == character)

*return* i;

  }

*return* -1;

}

*//creates a file with the requested name, if it doesn't already exist. Creates a copy otherwise. Returns file descriptor*

int createReceivedFile(char \*file\_name){

  int file;

  int i = 1;

  char extension[EXTENSION\_MAX\_SIZE];

  char file\_name\_no\_extension[FILE\_NAME\_MAX\_SIZE];

  int last\_dot\_pos = getLastCharPos(file\_name, '.');*//position of last '.'*

*if* (last\_dot\_pos > 0)*//if it has an extension*

  {

    strcpy(extension, file\_name + last\_dot\_pos); *//get extension*

    memcpy(file\_name\_no\_extension, file\_name, last\_dot\_pos); *//get file name without extension*

    file\_name\_no\_extension[last\_dot\_pos] = '\0';

  }

*else* strcpy(file\_name\_no\_extension, file\_name);

*while* ((file = open(file\_name, O\_WRONLY | O\_CREAT | O\_EXCL | O\_APPEND, 0777)) < 0)

  {

*if* (errno == EEXIST)

    {

*if* (last\_dot\_pos > 0) *//if it has extension*

        sprintf(file\_name, "%s(%d)%s", file\_name\_no\_extension, i, extension);

*else*

        sprintf(file\_name, "%s(%d)", file\_name\_no\_extension, i);

*if* (strlen(file\_name) < FILE\_NAME\_MAX\_SIZE){

*//if the size does not exceed the maximum file*

        i++;

      }

*else* {

        perror("file name too big");

        exit(-1);

      }

    }

*else*

    {

      perror("error on creating file");

      exit(-1);

    }

  }

*return* file;

}

int main(int argc, char \*\*argv)

{

  int fd; *//, c, res = 0;*

  unsigned char buf[D\_MAX\_SIZE];

*/\*unsigned char bufo[14] = {FLAG\_STUFF, 0x02, 0x20, 0x40, ESC\_OCT, F\_FLAG, F\_FLAG, ESC\_OCT, 0x20, 0x30, 0x10, ESC\_OCT, 0x20, 0x30};*

*unsigned char bufon[3] = {0x0, 0x0E, 0x70};*

*sendICommand(1, bufon, 3);\*/*

*if* ((argc < 2) ||

      ((strcmp("/dev/ttyS0", argv[1]) != 0) &&

       (strcmp("/dev/ttyS1", argv[1]) != 0) &&

       (strcmp("/dev/ttyS2", argv[1]) != 0) &&

       (strcmp("/dev/ttyS4", argv[1]) != 0)))

  {

    printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");

    exit(1);

  }

*/\**

*Open serial port device for reading and writing and not as controlling tty*

*because we don't want to get killed if linenoise sends CTRL-C.*

*\*/*

  fd = llopen(argv[1], RECEIVER);

*if* (fd == 0)

  {

    perror(argv[1]);

    exit(-1);

  }

*//serialRead(fd, buf);*

*// while (1)*

*// {*

*//   llread(fd, buf);*

*// }*

*//clock test*

  struct timeval start, stop;

  gettimeofday(&start, NULL); *// get initial time-stamp*

*//random erros generation*

  srand(time(NULL));

*/\**

*O ciclo WHILE deve ser alterado de modo a respeitar o indicado no gui�o*

*\*/*

  int bytes\_read;

  enum appState state = START;

  uint8\_t data\_package[D\_MAX\_SIZE];

  uint8\_t start\_package[D\_MAX\_SIZE];

  int s\_pack\_size;

  char file\_name[FILE\_NAME\_MAX\_SIZE];

  unsigned int file\_size;

  int file = -1;

  uint8\_t expected\_N = 0;

  int number\_of\_N\_groups = 0; *//incremented each time N reaches 255 (N is mod 256) (each group corresponds to 256 packets)*

*while* (STOP == FALSE)

  {

    bytes\_read = llread(fd,data\_package); *//get message from serial port*

*if* (bytes\_read != DISCARD) *//if the data is new and has no errors*

    {

*//read the serial port message and store it in data\_package*

*if* (state == START)

      {

        s\_pack\_size = readStartPackage(data\_package, bytes\_read, (uint8\_t \*) &file\_size, (uint8\_t \*) file\_name);

*if* (s\_pack\_size > 0)

        {

          file = createReceivedFile(file\_name); *//create the file to be received*

          memcpy(start\_package, data\_package, s\_pack\_size); *//store the start package for future validation*

          stateMachineApp(&state); *//start reading data*

        }

      }

*else* *if* (state == DATA)

      {

        int read\_result = readPackage(data\_package, buf, expected\_N, start\_package, s\_pack\_size);

*if* (read\_result > 0){

*//is valid data package*

*if* (write(file, buf, read\_result) < 0)

          {

            perror("writing to file");

            exit(2);

          }

*//*

          display\_completion(file\_size, expected\_N + 256 \* number\_of\_N\_groups);

*if* (expected\_N == 255)

            number\_of\_N\_groups++;

          expected\_N = (expected\_N + 1) % 256;

*//*

        }

*else* *if* (read\_result == VALID\_END\_PACKAGE)

          stateMachineApp(&state); *//move on to END phase*

*else* {

          printf("UNEXPECTED/INVALID PACKAGE\n");

        }

      }

*if* (state == END)

      {

        stateMachineApp(&state);

      }

    }

*if* (state == STOP\_)

      STOP = TRUE;

  }

  gettimeofday(&stop, NULL); *// get final time-stamp*

*if* (llclose(fd, RECEIVER) < 0)

  {

    printf("Timed out while closing port\n");

  }

*if* (!isFileSizeExpected(file, file\_size))

  {

    printf("Received file size is different than expected!!!\n");

  }

  close(file);

  double seconds = (double)(stop.tv\_sec - start.tv\_sec);

  double useconds = (seconds \* 1000000) + stop.tv\_usec - (start.tv\_usec);

  double t\_s = useconds / 1000000;

*// subtract time-stamps and*

*// multiply to get elapsed*

*// time in ns*

  printf("%f, time elapsed\n", t\_s);

*return* 0;

}

## writenoncanonical.c

*/\*Non-Canonical Input Processing\*/*

*#include* <sys/types.h>

*#include* <sys/stat.h>

*#include* <fcntl.h>

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <unistd.h>

*#include* <string.h>

*#include* "llfunctions.h"

*#include* "app\_package\_handling.h"

*#define* MODEMDEVICE "/dev/ttyS1"

*#define* \_POSIX\_SOURCE 1 */\* POSIX compliant source \*/*

volatile int STOP = FALSE;

int sendMessage(int bytes\_written, int fd, uint8\_t \*data\_package)

{

  int resend = FALSE;

*if* (bytes\_written > 0)

  {

    int write\_result = llwrite(fd, data\_package, bytes\_written);

*if* (write\_result == RESEND\_RET)

      resend = TRUE;

*else* *if* (write\_result == TIMEOUT\_RET)

    {

      printf("TIMED OUT\n");

      exit(2);

    }

*// printf("message sent - %d bytes\n", write\_result);*

  }

*return* resend;

}

int fileDataReading(int fd, uint8\_t \*data\_package, enum appState \*state)

{

  uint8\_t buf[PACKAGE\_DATA\_SIZE];

  int bytes\_read = read(fd, buf, PACKAGE\_DATA\_SIZE);

*// for (unsigned int i = 0; i < bytes\_read; i++)*

*// {*

*//   printf("%x", buf[i]);*

*// }*

*// printf("\n");*

*if* (bytes\_read < 0)

  {

    perror("file reading");

    exit(1);

  }

*else* *if* (bytes\_read == 0)

  {

*//if it reaches the end of the file, go to the END state*

    stateMachineApp(state);

  }

*else*

  {

    bytes\_read = buildDataPackage(bytes\_read, buf, data\_package);

  }

*return* bytes\_read;

}

int setupControlPackage(int fd, uint8\_t \*data\_package, uint8\_t c, enum appState \*state, char \*file\_name, unsigned int name\_size)

{

  struct stat st;

  fstat(fd, &st); *//retrieves size of file*

  int bytes\_written = buildControlPackage(c, st.st\_size, (uint8\_t \*) file\_name, name\_size, data\_package);

  stateMachineApp(state); *//if Start, goes to DATA state; if END, goes to stop state*

*return* bytes\_written;

}

int main(int argc, char \*\*argv)

{

  int fd; *//, c, res;*

  unsigned char buf[D\_MAX\_SIZE];

*//int i, sum = 0, speed = 0;*

*if* ((argc < 3) ||

      ((strcmp("/dev/ttyS0", argv[1]) != 0) &&

       (strcmp("/dev/ttyS1", argv[1]) != 0) &&

       (strcmp("/dev/ttyS2", argv[1]) != 0)))

  {

    printf("Usage:\tnserial SerialPort FileName\n\tex: nserial /dev/ttyS1 pinguim.gif\n");

    exit(1);

  }

*//file name and respective size*

  unsigned file\_name\_length = strlen(argv[2]);

*if* (file\_name\_length > ORIGINAL\_FILE\_MAX\_SIZE ){

    printf("File name is too big\n");

    exit(4);

  }

*//open file to be read*

  int file = open(argv[2], O\_RDONLY, 0777);

*if* (file < 0)

  {

    perror("error on opening requested file");

    exit(-1);

  }

*//*

*/\**

*Open serial port device for reading and writing and not as controlling tty*

*because we don't want to get killed if linenoise sends CTRL-C.*

*\*/*

  fd = llopen(argv[1], TRANSMITTER);

*if* (fd == 0)

  {

    perror(argv[1]);

    exit(-1);

  }

*else* *if* (fd == TIMEOUT\_RET)

  {

    printf("Timed out while opening port\n");

    exit(2);

  }

*/\**

*O ciclo FOR e as instru��es seguintes devem ser alterados de modo a respeitar*

*o indicado no gui�o*

*\*/*

  int resend = FALSE;

  int bytes\_written;

  enum appState state = START;

  uint8\_t data\_package[D\_MAX\_SIZE];

*while* (STOP == FALSE)

  {

*if* (!resend)

    {

*//if the previous message was received sucessfully, build the next one*

*if* (state == START)

      {

        bytes\_written = setupControlPackage(file, data\_package, C\_START, &state, argv[2], file\_name\_length);

      }

*else* *if* (state == DATA)

      {

        bytes\_written = fileDataReading(file, data\_package, &state);

*//printf("%d - BYTES WRITTEN\n", bytes\_written);*

      }

*else* *if* (state == END)

      {

        bytes\_written = setupControlPackage(file, data\_package, C\_END, &state, argv[2], file\_name\_length);

      }

    }

    resend = sendMessage(bytes\_written, fd, data\_package);

*if* (!resend && state == STOP\_) *//if it reached the end and the END Package has been received sucessfully*

      STOP = TRUE;

  }

*//close*

*if* (llclose(fd, TRANSMITTER) < 0)

  {

    printf("Timed out while closing port\n");

  }

  close(file);

*return* 0;

}

# 12. Anexo II – Tabelas

## Variação da Capacidade de Ligação

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C |  | T(s) | R(bit/s) | S(R/C) | S MEDIO |
| 1200 |  | 97,705888 | 898,042091 | 0,748368 | 0,74837 |
| 1200 |  | 97,706033 | 898,040759 | 0,748367 |  |
| 1200 |  | 97,705034 | 898,049941 | 0,748375 |  |
| 1200 |  | 97,705932 | 898,041687 | 0,748368 |  |
|  |  |  |  |  |  |
| 1800 |  | 65,139124 | 1347,024562 | 0,748347 | 0,748345 |
| 1800 |  | 65,139052 | 1347,026051 | 0,748348 |  |
| 1800 |  | 65,139755 | 1347,011514 | 0,74834 |  |
| 1800 |  | 65,139332 | 1347,020261 | 0,748345 |  |
|  |  |  |  |  |  |
| 2400 |  | 50,222229 | 1747,114808 | 0,727965 | 0,727965 |
| 2400 |  | 50,222272 | 1747,113313 | 0,727964 |  |
| 2400 |  | 50,222326 | 1747,111434 | 0,727963 |  |
| 2400 |  | 50,222001 | 1747,12274 | 0,727968 |  |
|  |  |  |  |  |  |
| 4800 |  | 25,115359 | 3493,639092 | 0,727841 | 0,727814 |
| 4800 |  | 25,115475 | 3493,622956 | 0,727838 |  |
| 4800 |  | 25,115449 | 3493,626572 | 0,727839 |  |
| 4800 |  | 25,11893 | 3493,142423 | 0,727738 |  |
|  |  |  |  |  |  |
| 9600 |  | 12,561549 | 6985,125799 | 0,727617 | 0,727614 |
| 9600 |  | 12,56166 | 6985,064076 | 0,727611 |  |
| 9600 |  | 12,561392 | 6985,213104 | 0,727626 |  |
| 9600 |  | 12,561814 | 6984,978443 | 0,727602 |  |
|  |  |  |  |  |  |
| 19200 |  | 6,284633 | 13961,674453 | 0,727171 | 0,727149 |
| 19200 |  | 6,284739 | 13961,438971 | 0,727158 |  |
| 19200 |  | 6,284883 | 13961,119085 | 0,727142 |  |
| 19200 |  | 6,285007 | 13960,843639 | 0,727127 |  |
|  |  |  |  |  |  |
| 38400 |  | 3,146729 | 27884,193396 | 0,726151 | 0,726528 |
| 38400 |  | 3,219702 | 27252,21154 | 0,709693 |  |
| 38400 |  | 3,108139 | 28230,397675 | 0,735167 |  |
| 38400 |  | 3,108417 | 28227,872901 | 0,735101 |  |

|  |  |  |
| --- | --- | --- |
| **Total bytes** | | 10968 |
| **Total bits** |  | 87744 |
| **Package Size** | | 255 |

## Variação do Tempo de Propagação (T\_prop)

|  |  |  |
| --- | --- | --- |
| **Total bytes** | | 10968 |
| **Total bits** |  | 87744 |
| **Package Size** | | 255 |
| **Baudrate ( C )** | | 38400 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T\_prop(s) | T(s) | R(bit/s) | S(R/C) | S MEDIO |
| 0 | 3,146729 | 27884,193396 | 0,726151 | 0,699662 |
| 0 | 3,219702 | 27252,21154 | 0,709693 |  |
| 0 | 3,146633 | 27885,044109 | 0,726173 |  |
| 0 | 3,589213 | 24446,584808 | 0,63663 |  |
|  |  |  |  |  |
| 1 | 45,226076 | 1940,119678 | 0,050524 | 0,050523 |
| 1 | 45,22889 | 1939,99897 | 0,050521 |  |
| 1 | 45,229982 | 1939,952132 | 0,05052 |  |
| 1 | 45,221825 | 1940,302055 | 0,050529 |  |
|  |  |  |  |  |
| 2 | 90,238982 | 972,351395 | 0,025322 | 0,025321 |
| 2 | 90,235119 | 972,393021 | 0,025323 |  |
| 2 | 90,234355 | 972,401254 | 0,025323 |  |
| 2 | 90,262077 | 972,102603 | 0,025315 |  |

## Variação do tamanho dos pacotes de dados

|  |  |  |
| --- | --- | --- |
| **Total bytes** | | 10968 |
| **Total bits** |  | 87744 |
| **Baudrate ( C )** | | 38400 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Package S | T(s) | R(bit/s) | S(R/C) | S MEDIO |
| 40 | 4,434645 | 19786,025713 | 0,515261 | 0,514873 |
| 40 | 4,439087 | 19766,226704 | 0,514745 |  |
| 40 | 4,439213 | 19765,665671 | 0,514731 |  |
| 40 | 4,439003 | 19766,600743 | 0,514755 |  |
|  |  |  |  |  |
| 80 | 3,622802 | 24219,927007 | 0,630727 | 0,630751 |
| 80 | 3,622316 | 24223,176553 | 0,630812 |  |
| 80 | 3,622537 | 24221,698771 | 0,630773 |  |
| 80 | 3,623013 | 24218,516467 | 0,630691 |  |
|  |  |  |  |  |
| 120 | 3,367995 | 26052,295208 | 0,678445 | 0,678514 |
| 120 | 3,367383 | 26057,030044 | 0,678568 |  |
| 120 | 3,367639 | 26055,04925 | 0,678517 |  |
| 120 | 3,367598 | 26055,366466 | 0,678525 |  |
|  |  |  |  |  |
| 160 | 3,245112 | 27038,820232 | 0,704136 | 0,704012 |
| 160 | 3,245848 | 27032,689146 | 0,703976 |  |
| 160 | 3,245888 | 27032,356015 | 0,703968 |  |
| 160 | 3,245885 | 27032,380999 | 0,703968 |  |
|  |  |  |  |  |
| 200 | 3,169515 | 27683,730792 | 0,72093 | 0,720952 |
| 200 | 3,169441 | 27684,37715 | 0,720947 |  |
| 200 | 3,169723 | 27681,914161 | 0,720883 |  |
| 200 | 3,169009 | 27688,151091 | 0,721046 |  |
|  |  |  |  |  |
| 255 | 3,146729 | 27884,193396 | 0,726151 | 0,726528 |
| 255 | 3,219702 | 27252,21154 | 0,709693 |  |
| 255 | 3,108139 | 28230,397675 | 0,735167 |  |
| 255 | 3,108417 | 28227,872901 | 0,735101 |  |

## Variação de FER

|  |  |  |
| --- | --- | --- |
| **Total bytes** | | 10968 |
| **Total bits** |  | 87744 |
| **Package Size** | | 255 |
| **Baudrate ( C )** | | 38400 |

### BCC1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BCC1 | T(s) | R(bit/s) | S(R/C) | S MEDIO |
| 0 | 3,146729 | 27884,193396 | 0,726151 | 0,699662 |
| 0 | 3,219702 | 27252,21154 | 0,709693 |  |
| 0 | 3,146633 | 27885,044109 | 0,726173 |  |
| 0 | 3,589213 | 24446,584808 | 0,63663 |  |
|  |  |  |  |  |
| 2 | 3,809237 | 23034,534212 | 0,599858 | 0,322536 |
| 2 | 7,49916 | 11700,510457 | 0,304701 |  |
| 2 | 10,543155 | 8322,366502 | 0,216728 |  |
| 2 | 13,532164 | 6484,10705 | 0,168857 |  |
|  |  |  |  |  |
| 4 | 10,609489 | 8270,332341 | 0,215373 | 0,219478 |
| 4 | 16,698142 | 5254,716363 | 0,136842 |  |
| 4 | 7,595569 | 11551,998277 | 0,300833 |  |
| 4 | 10,161645 | 8634,822413 | 0,224865 |  |
|  |  |  |  |  |
| 6 | 10,854174 | 8083,89473 | 0,210518 | 0,178769 |
| 6 | 20,007201 | 4385,620957 | 0,114209 |  |
| 6 | 23,148625 | 3790,462716 | 0,09871 |  |
| 6 | 7,835044 | 11198,916049 | 0,291638 |  |
|  |  |  |  |  |
| 8 | 13,931043 | 6298,451595 | 0,164022 | 0,139019 |
| 8 | 14,028543 | 6254,676626 | 0,162882 |  |
| 8 | 19,992404 | 4388,866892 | 0,114293 |  |
| 8 | 19,890456 | 4411,361911 | 0,114879 |  |
|  |  |  |  |  |
| 10 | 19,313945 | 4543,038722 | 0,118308 | 0,158478 |
| 10 | 16,947312 | 5177,458231 | 0,13483 |  |
| 10 | 7,802289 | 11245,930521 | 0,292863 |  |
| 10 | 25,99233 | 3375,765081 | 0,087911 |  |

### BCC2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BCC2 | T(s) | R(bit/s) | S(R/C) | S MEDIO |
| 0 | 3,146729 | 27884,193396 | 0,726151 | 0,699662 |
| 0 | 3,219702 | 27252,21154 | 0,709693 |  |
| 0 | 3,146633 | 27885,044109 | 0,726173 |  |
| 0 | 3,589213 | 24446,584808 | 0,63663 |  |
|  |  |  |  |  |
| 2 | 4,370563 | 20076,132068 | 0,522816 | 0,535086 |
| 2 | 4,074575 | 21534,51587 | 0,560795 |  |
| 2 | 4,574175 | 19182,475528 | 0,499544 |  |
| 2 | 4,100937 | 21396,085821 | 0,55719 |  |
|  |  |  |  |  |
| 4 | 4,586252 | 19131,962221 | 0,498228 | 0,506427 |
| 4 | 4,824915 | 18185,605342 | 0,473583 |  |
| 4 | 4,2982 | 20414,126844 | 0,531618 |  |
| 4 | 4,375056 | 20055,514718 | 0,522279 |  |
|  |  |  |  |  |
| 6 | 4,720032 | 18589,704477 | 0,484107 | 0,462508 |
| 6 | 4,990139 | 17583,478136 | 0,457903 |  |
| 6 | 4,830078 | 18166,166261 | 0,473077 |  |
| 6 | 5,253535 | 16701,896913 | 0,434945 |  |
|  |  |  |  |  |
| 8 | 4,418142 | 19859,932071 | 0,517186 | 0,46872 |
| 8 | 5,32522 | 16477,065736 | 0,42909 |  |
| 8 | 4,303248 | 20390,179697 | 0,530994 |  |
| 8 | 5,746857 | 15268,171802 | 0,397609 |  |
|  |  |  |  |  |
| 10 | 4,85291 | 18080,697973 | 0,470852 | 0,463948 |
| 10 | 4,537821 | 19336,152748 | 0,503546 |  |
| 10 | 5,016693 | 17490,406529 | 0,455479 |  |
| 10 | 5,364897 | 16355,206819 | 0,425917 |  |

|  |  |  |
| --- | --- | --- |
| **Total bytes** | | 10968 |
| **Total bits** |  | 87744 |
| **Package Size** | | 255 |
| **Baudrate ( C )** | | 38400 |