

**Universidade do Minho**  
Escola de Engenharia

## **PROJETO INTEGRADOR EM TELECOMUNICAÇÕES E INFORMÁTICA**

# **TRANSFERÊNCIA DE INFORMAÇÃO POR CANAL ÓTICO**

Grupo 2

Inês Barreira Marques - a84913@alunos.uminho.pt

Miguel Chaves Moreira - a77314@alunos.uminho.pt

Rui Filipe Ribeiro Freitas - pg47639@alunos.uminho.pt

Tiago João Pereira Ferreira - pg47692@alunos.uminho.pt

2 de junho de 2022

# Índice

<b>1</b>	<b>Introdução</b>	<b>4</b>
<b>2</b>	<b>Planeamento do projeto</b>	<b>5</b>
2.1	Planeamento temporal . . . . .	5
2.2	Tecnologias/Ferramentas necessárias . . . . .	6
2.2.1	Ao nível do hardware . . . . .	6
2.2.2	Ao nível do software . . . . .	7
<b>3</b>	<b>Fundamentos</b>	<b>8</b>
3.1	Arquitetura geral do sistema . . . . .	8
3.2	Hardware Específico . . . . .	9
3.2.1	Fototransístor . . . . .	9
3.2.2	Fotodiodo . . . . .	10
3.3	Métodos de acesso múltiplo . . . . .	11
3.3.1	CDMA . . . . .	12
<b>4</b>	<b>Desenvolvimento</b>	<b>13</b>
4.1	Modelo de Camadas . . . . .	13
4.2	Emissor . . . . .	14
4.2.1	Circuito . . . . .	14
4.2.2	Aplicação . . . . .	16
4.2.3	ESP32 . . . . .	18
4.3	Recetor . . . . .	22
4.3.1	Circuito . . . . .	22
4.3.2	Aplicação . . . . .	23
4.3.3	ESP32 . . . . .	25
<b>5</b>	<b>Testes e Discussão de Resultados</b>	<b>28</b>
5.1	Distância máxima . . . . .	28
5.2	Testes do tempo de transferência . . . . .	30
5.3	Testes do tamanho do payload . . . . .	32
<b>6</b>	<b>Conclusão</b>	<b>33</b>

## **Lista de Figuras**

1	Arquitetura geral do sistema.	8
2	Fototransistor.	9
3	Fotodiodo.	10
4	Métodos de acesso múltiplo e o seu recurso característico.	11
5	Métodos de acesso múltiplo.	11
6	CDMA.	12
7	Modelo de Camadas.	13
8	Círculo emissor.	14
9	Fluxograma da aplicação do emissor.	16
10	Trama Emissor.	16
11	Aplicação Emissor.	17
12	Aplicação Emissor - Ficheiro.	17
13	Fluxograma do programa do ESP32 do emissor.	18
14	Trama Data.	19
15	Conexão TCP/IP - ESP32.	20
16	Receção e criação de pacotes - ESP32.	21
17	Envio de um byte - ESP32.	21
18	Círculo recetor.	22
19	Fluxograma do recetor.	23
20	Aplicação do recetor.	24
21	Ficheiro recebido.	24
22	Fluxograma do programa do ESP32 - Recetor.	25
23	Receção de um byte - ESP32.	26
24	Tratamento de um byte - ESP32.	27
25	Teste distância.	28
26	Visualização do sinal a 32 cm.	29
27	Visualização do sinal a 40 cm.	29

28	Função packet_creation() - 1. . . . .	35
29	Função packet_creation() - 2. . . . .	36

## **Lista de Tabelas**

1	Planeamento temporal . . . . .	5
2	Tecnologias ao nível do hardware . . . . .	6
3	Tecnologias a nível de software. . . . .	7
4	Testes dos tempos de transferência. . . . .	30
5	Testes do tamanho do <i>payload</i> . . . . .	32

# **1 Introdução**

Atualmente existem inúmeros espaços, interiores e exteriores, cuja iluminação é feita com base em luminárias LED, que usam díodos emissores de luz com o objetivo de gerar radiação na zona do espetro visível. Estes LEDs são utilizados também para transmitir informação através de modulações da intensidade luminosa.

O presente relatório apresenta as tarefas desenvolvidas no contexto do trabalho semestral. A realização deste teve como objetivo a construção de um modelo de um sistema com luminárias LED de modo a conseguir transmitir informação sobre um local em que estas estão instaladas. De modo a ser possível construir este modelo, foi necessário criar um receptor capaz de receber a informação transmitida num dado local e reproduzi-la ao utilizador.

De modo a sermos capazes de cumprir com os objetivos deste projeto semestral foi necessário pôr em prática conhecimentos adquiridos noutras unidades curriculares assim como adquirir novos conhecimentos na área das comunicações óticas, modulações e codificações, deteção e correção de erros, entre outras.

## 2 Planeamento do projeto

### 2.1 Planeamento temporal

Numa fase inicial do projeto, o grupo realizou um planeamento temporal com o intuito de delinear tarefas e limites para as mesmas de forma a que fosse possível ter em atenção as etapas já realizadas, o que ainda tínhamos para fazer e ainda o tempo restante para realizar as etapas necessários ao desenrolar do trabalho. A tabela seguinte corresponde ao planeamento realizado no início do projeto. Apesar de algumas tarefas não terem sido realizadas a tempo, através deste planeamento foi possível o grupo guiar-se relativamente ao que já estava feito e ao que faltava fazer.

**Tabela 1:** Planeamento temporal

Fase	Intermédia 1	Intermédia 2	Final
<b>Tarefas</b>	<ul style="list-style-type: none"><li>• Definir o modelo de camadas</li><li>• Compreensão de conceitos</li><li>• Comunicação entre PCs</li><li>• Elaboração da Apresentação da Fase Intermédia 1</li></ul>	<ul style="list-style-type: none"><li>• Montagem do driver e frontend</li><li>• Desenvolvimento de código do ESP</li><li>• Aplicação do CDMA</li><li>• Elaboração da Apresentação da Fase Intermédia 2</li></ul>	<ul style="list-style-type: none"><li>• Correção das fases anteriores</li><li>• Testes e discussão de resultados</li><li>• Elaboração do relatório final</li><li>• Elaboração da Apresentação da Fase Final</li></ul>
<b>Datas</b>	<ul style="list-style-type: none"><li>• <b>Entrega da apresentação:</b> 11 de Março</li><li>• <b>Apresentação:</b> 18 de Março</li></ul>	<ul style="list-style-type: none"><li>• <b>Entrega da apresentação:</b> 1 de Abril</li><li>• <b>Apresentação:</b> 8 de Abril</li></ul>	<ul style="list-style-type: none"><li>• <b>Entrega do Relatório final:</b> 1 de junho</li><li>• <b>Apresentação:</b> 3 de junho</li></ul>

## 2.2 Tecnologias/Ferramentas necessárias

Em qualquer projeto são necessárias certas tecnologias/ferramentas que facilitem o trabalho e ao mesmo tempo que aumentem a produtividade. Para este trabalho em específico foi necessário recorrer a tecnologias específicas tanto ao nível do hardware como ao nível do software, de modo a cumprir com os objetivos propostos.

### 2.2.1 Ao nível do hardware

No que toca à parte do hardware deste trabalho as tecnologias por nós utilizadas estão presentes na tabela seguinte.

**Tabela 2:** Tecnologias ao nível do hardware

Ferramenta	Quantidade	Descrição
Computador portátil	2	Desenvolvimento de código, elaboração do relatório, pesquisa e testes.
Módulo ESP32	2	Responsável por realizar a transferência e receção dos dados.
Fotodiode	1	Utilizado para envio dos dados.
Fototransistor	1	Utilizado para a receção dos dados provenientes do Fotodiode.
Transistor 2n2222	1	Responsável pelo controlo da tensão e da corrente que passam no led
Circuito tl084	1	Utilizado para amplicar o nosso sinal.
Condensadores	2	Utilizado para filtrar a nossa onda.
Resistências	várias	Responsáveis pelo controlo da corrente.
Cabos	vários	Responsáveis pela ligação dos componentes.

### 2.2.2 Ao nível do software

Em relação ao software que foi utilizado na execução do nosso projeto este passa maioritariamente pelas aplicações apresentadas na tabela 2. Nesta realçamos as mais importantes e fundamentais para a comunicação entre o grupo e desenvolvimento do trabalho.

**Tabela 3:** Tecnologias a nível de software.

LOGO	DESCRIÇÃO	UTILIZAÇÃO
	<b>Overleaf</b>	Realização e sincronização do relatório do projeto
	<b>Arduino IDE</b>	Software para a conexão com a placa ESP32 e comunicação com o mesmo
	<b>Visual Studio Code</b>	Execução e compilação do código
	<b>Discord</b>	Comunicação entre o grupo
	<b>Visual Paradigm</b>	Implementação de fluxogramas para a elaboração do projeto
	<b>GitHub</b>	Sincronização do código da aplicação

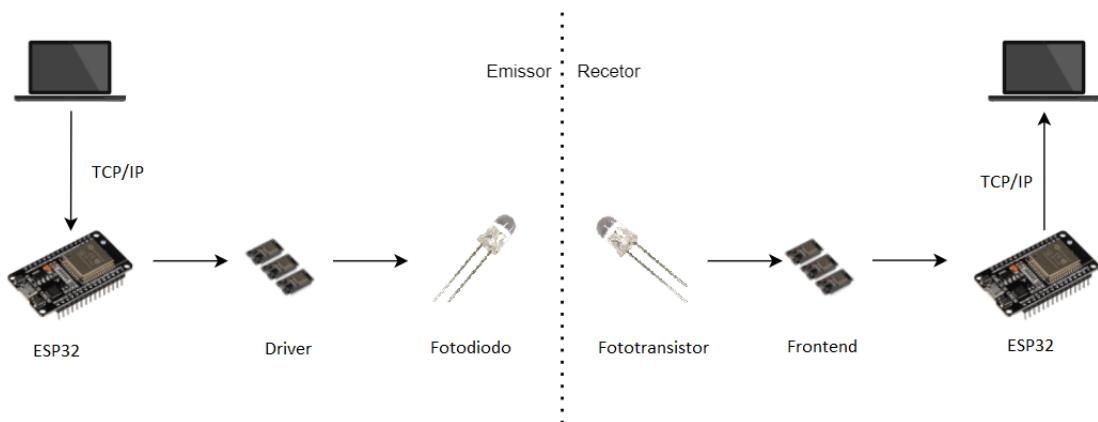
### 3 Fundamentos

Nesta secção do relatório é apresentada a parte mais teórica da realização deste projeto com o desenho da arquitetura geral a implementar assim como a divisão deste projeto nas devidas camadas e respetivas tarefas.

#### 3.1 Arquitetura geral do sistema

A nossa arquitetura está dividida em duas partes, a do emissor e do recetor. Do lado do emissor é executada a aplicação num PC que gera dados que são enviados ao ESP32 através de uma ligação TCP/IP para mais tarde serem difundidos através do fotodiodo para o recetor. O driver tem como função interligar o ESP32 com o dispositivo emissor de luz.

Do lado do recetor temos um fototransístor que recebe os dados provenientes do emissor. É implementado à custa de hardware específico para transformar o sinal ótico num sinal elétrico e software a executar no ESP32. Por fim, temos um PC do lado do recetor que recebe e visualiza a informação difundida através do canal ótico.



**Figura 1:** Arquitetura geral do sistema.

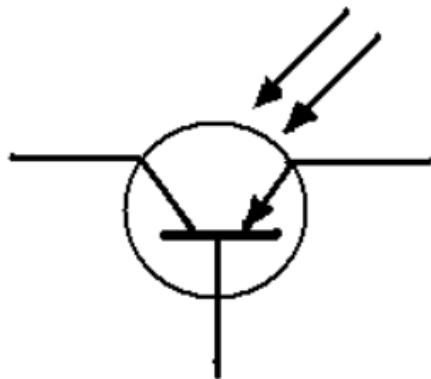
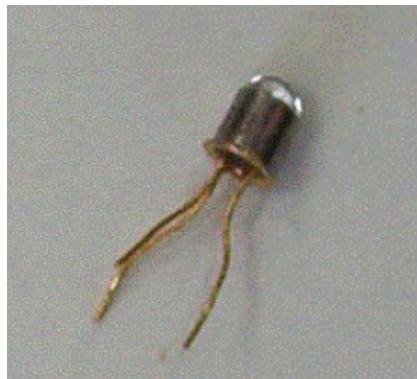
Considera-se que o canal é todo e qualquer espaço, interior, em que exista linha de vista entre o dispositivo emissor de luz e o fototransístor.

## 3.2 Hardware Específico

Nesta subsecção é apresentado em detalhe as características e forma de funcionamento das 2 tecnologias de hardware mais importantes na realização deste projeto, o fototransistor e o fotodíodo, pois é destes que é enviado e recebido o feixe ótico.

### 3.2.1 Fototransistor

Um fototransistor é um transistor bipolar encorporado numa proteção transparente que permite que a luz possa atingir a base coletora da junção. O fototransistor funciona de maneira similar a um fotodiodo, apresentando uma sensibilidade à luz muito maior.[b1] Estes são uma comutação eletrónica e componente de amplificação de corrente que depende da exposição à luz para operar. Quando a luz incide na junção, a corrente contrária flui que é proporcional à luminosidade. Os fototransistores funcionam por luz em vez de corrente elétrica podendo transformar essa energia luminosa recebida em energia elétrica. São transístores com o terminal da base exposto, em vez de enviar corrente para a base, os fotões de luz que incidem ativam o transistore. Isso acontece porque um fototransistor é feito de um semicondutor bipolar e concentra a energia que passa por ele.[b2]

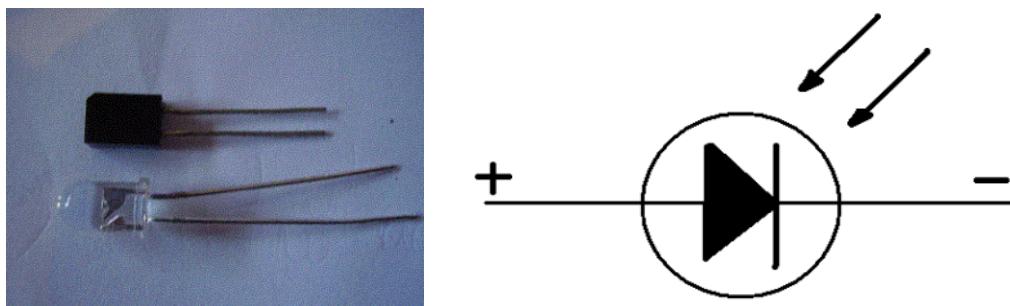


**Figura 2:** Fototransistor.

### 3.2.2 Fotodiodo

Os Fotodiodos são dispositivos eletrónicos feitos de um material semicondutor. Possuem uma junção semicondutora, que tem a propriedade de variar a resistência elétrica em função da intensidade da luz que nela incide (número de fotões). Quando a luz incide na junção, a resistência no sentido oposto ao fluxo normal cai de forma abrupta, o que permite que exista um fluxo de corrente em ambos os sentidos.[b5]

O aumento da corrente no sentido proibido permite detetar a luz incidente e pode ser relacionado com a intensidade luminosa que atinge a junção. Alguns fotodiodos funcionam como um diodo emissor de luz. A extremidade menor do diodo é o terminal do cátodo, enquanto que a extremidade mais longa do diodo é o terminal do ânodo. Na condição de polarização direta, a corrente convencional flui do ânodo para o cátodo, seguindo a seta no símbolo do diodo.[b4]



**Figura 3:** Fotodiodo.

### 3.3 Métodos de acesso múltiplo

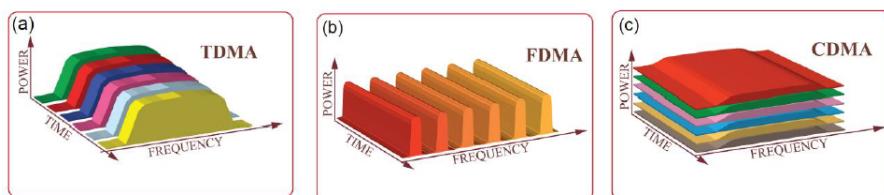
Um método de acesso múltiplo é uma técnica que é usada na partilha de recursos limitados entre vários utilizadores. Cada método aproveita como vantagem certas características de um sistema de comunicação de modo a dividir e alocar a largura de banda disponível. A seguinte tabela demonstra quais os métodos existentes assim como qual o recurso que cada um utiliza.

Multiple Access Method	Resource Characteristic
TDMA	Time
WDMA	Optical Wavelength
SCMA (FDMA)	Electrical Frequency
CDMA	Code

**Figura 4:** Métodos de acesso múltiplo e o seu recurso caraterístico.

Através da observação da tabela podemos observar que o método TDMA utiliza o tempo como caraterística, o WDMA utiliza o comprimento de onda ótico, o FDMA que utiliza a frequência elétrica e o CDMA código, mais especificamente códigos de espalhamento.

É importante realçar as principais diferenças entre estes métodos demonstrados na figura seguinte em que podemos retirar informação relativamente à utilização da largura de banda por parte destes. Resumidamente podemos observar que no TDMA figura 5(a) os utilizadores partilham a mesma frequência em diferentes espaços de tempo, no FDMA figura 5(b) em que diferentes bandas de frequência são atribuídas a diferentes fluxos de dados e no CDMA figura 5(c) em que a totalidade do espetro é utilizada para codificar a informação de todos os utilizadores em que estes são distinguidos pelos seus códigos de espalhamento.



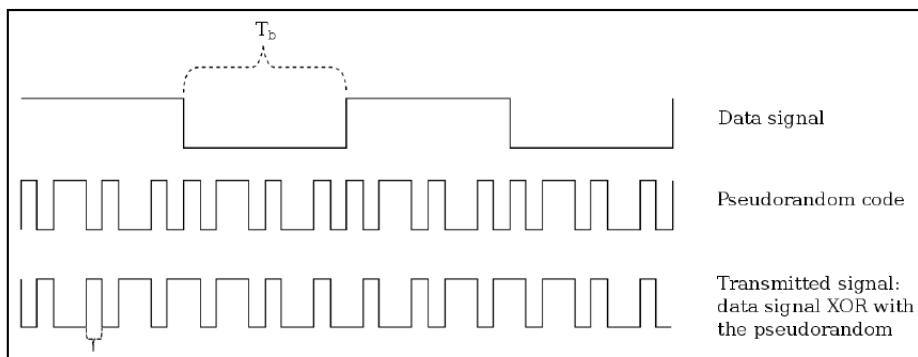
**Figura 5:** Métodos de acesso múltiplo.

### 3.3.1 CDMA

O CDMA (Code Division Multiple Access) é uma tecnologia de múltiplo acesso em que todos os utilizadores podem transmitir na mesma banda de frequência ao mesmo tempo, utilizando uma técnica de espalhamento do espetro.

Como cada par transmissor-recetor tem um único código de pseudo-ruído para evitar interferência e garantir privacidade, isto faz com que haja um grande espaço para transferência de dados.

Na figura 6 podemos observar o que acontece quando pretendemos transmitir uma mensagem. Na primeira linha encontra-se o sinal da informação a transmitir, na segunda o código de pseudo-ruído que corresponde ao sinal e por fim a realização do XOR entre estes 2 sinais resultando assim um sinal transmitido privado e eficaz.



**Figura 6:** CDMA.

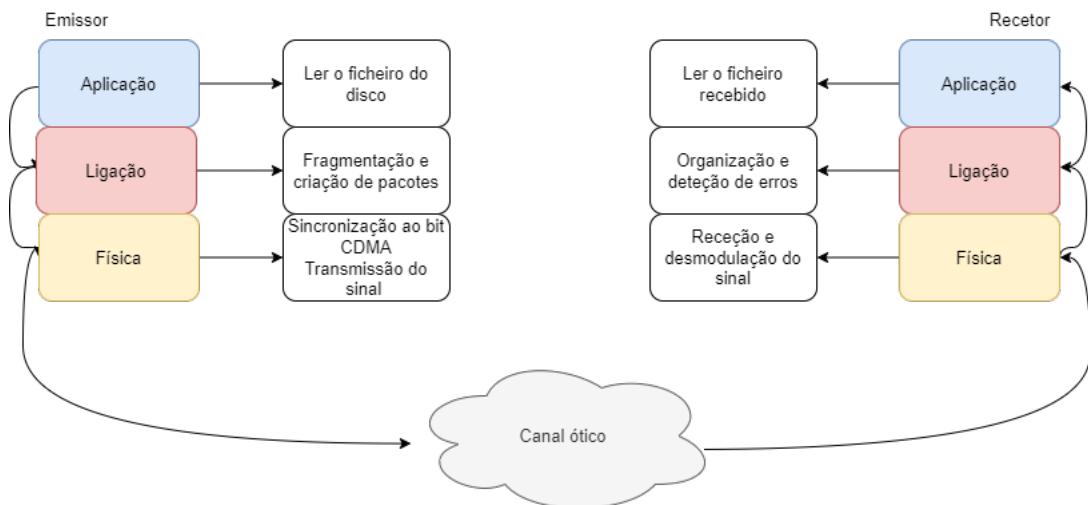
Se o sinal recebido não for dirigido para o utilizador, a correlação será muito pequena pelo que este não será aceite, isto porque o código usado por cada utilizador é diferente. De forma que os sinais sejam diferentes, são utilizados códigos de Walsh pelo facto de serem ortogonais, permitindo assim que estes sejam o mais distintos possível uns dos outros.

## 4 Desenvolvimento

Nesta secção do relatório é apresentada a parte mais prática do projeto onde estão as decisões tomadas na realização dos códigos das várias camadas.

### 4.1 Modelo de Camadas

Para a realização deste projeto, foi necessário criar um modelo de camadas que atendesse a todas as necessidades do trabalho. Foi por isso desenhado um modelo com 3 camadas, sendo estas a camada de Aplicação, Ligação e a Física. Como sugerido pelos docentes foi realizada uma abordagem *top-down*, isto é, começando pela implementação das camadas de nível superior e avançando depois para as camadas de nível inferior.



**Figura 7:** Modelo de Camadas.

No lado do emissor, na camada de Aplicação, é lido um ficheiro em memória para ser transmitido. Esta camada comunica com a camada abaixo, camada de Ligação, através do protocolo TCP/IP. Na camada de Ligação é realizada a fragmentação do ficheiro e a criação de pacotes para posterior envio. Esta camada comunica com a

camada física através de software. Na camada Física é feita a sincronização ao bit, é aplicado o CDMA e a transmissão do sinal por via do canal ótico.

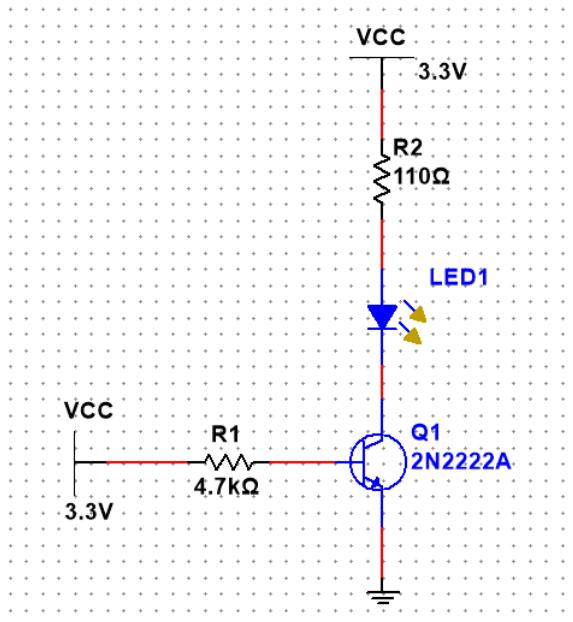
No lado do receptor, o sinal é recebido do canal ótico, pela camada Física, e é feita a desmodulação do mesmo. O sinal desmodulado, é enviado para a camada de Ligação onde vai ser reorganizado o ficheiro e onde é feita a deteção de erros, terminada esta etapa é enviado para a camada de aplicação onde vai ocorrer a leitura do ficheiro.

## 4.2 Emissor

De seguida é explicado todo o processo que foi realizado para implementar tanto o código da aplicação como da placa ESP32 do lado do emissor.

### 4.2.1 Circuito

Na figura seguinte é representado um esquema do circuito realizado pelo grupo.



**Figura 8:** Circuito emissor.

De seguida apresentamos os cálculos que o grupo realizou de forma a descobrir as resistências necessárias a usar no circuito do emissor. Na equação (1) está apresentado o cálculo da resistência do coletor em que foi utilizado o valor de 20 mA para a corrente, valor retirado do datasheet do módulo ESP32.

$$V_{cc} - R_c \times I_c - V_{led} = 0 \Leftrightarrow 3.3 - R_c I_c - 1.2 = 0 \Leftrightarrow R_c = \frac{2.1}{20mA} \Leftrightarrow R_c = 105\Omega \quad (1)$$

No cálculo da corrente Ib apresentado em (2), foi usada a corrente máxima Ic, que é a corrente máxima do LED e o ganho(Beta), em que ambos os valores foram retirados dos datasheets do fotodíodo e do transístor, respetivamente.

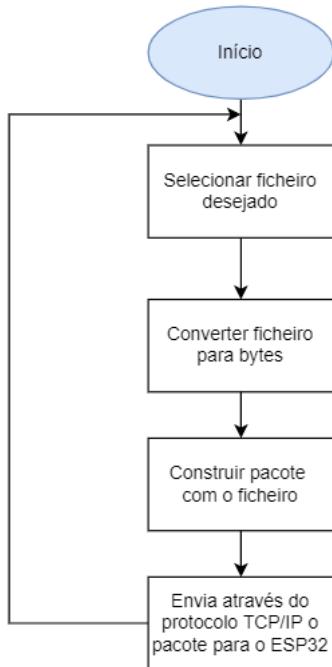
$$I_b = \frac{I_c}{\beta} \Leftrightarrow I_b = \frac{50mA}{75} \quad (2)$$

Na equação (3) encontram-se os cálculos efetuados para a resistência da base. Foi usado como I a corrente típica encontrada no datasheet do fotodíodo e uma voltagem de 3.3V.

$$V_{cc} - R_b I_b - V_{be} = 0 \Leftrightarrow 3.3 - R_b I_b - 0.6 = 0 \Leftrightarrow R_b = \frac{3.3 - 0.6}{6.67 \times 10^{-4}} \Leftrightarrow R \geq 4K\Omega \quad (3)$$

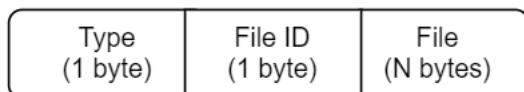
#### 4.2.2 Aplicação

Relativamente à aplicação do emissor, este é um programa bastante simples e o seu fluxograma é apresentado na figura seguinte.



**Figura 9:** Fluxograma da aplicação do emissor.

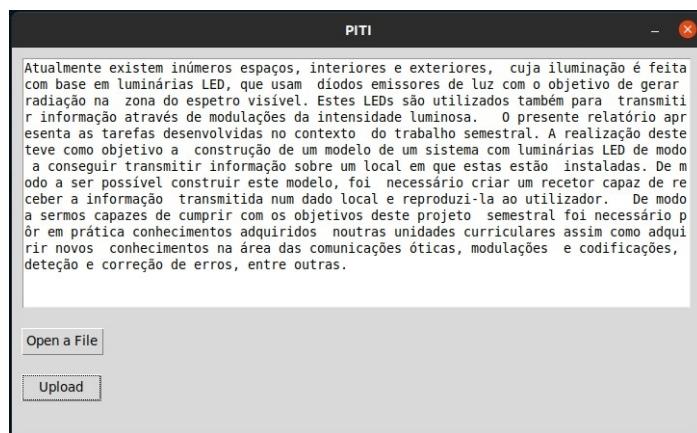
Através da observação do fluxograma anterior podemos então observar o funcionamento da aplicação no lado do emissor em que este começa por selecionar o ficheiro desejado através de uma interface gráfica simples. Posteriormente, o programa faz a leitura binária do ficheiro pretendido e cria um pacote, como o demonstrado na figura abaixo.



**Figura 10:** Trama Emissor.

A trama do emissor, tal como é possível ver na figura 10, possui três campos, o primeiro campo é o tipo, que ocupa 1 byte, e que é utilizado para indicar ao ESP32 qual o pacote que está a receber. O segundo campo, que ocupa também 1 byte, contém o ID do ficheiro que identifica o ficheiro que está a ser enviado. O terceiro e último campo é o conteúdo do ficheiro e ocupa o tamanho total do ficheiro.

Após o utilizador correr a aplicação é apresentada a janela da próxima figura em que o utilizador tem 2 botões para carregar, 1 para escolher qual ficheiro quer enviar e outro onde realiza o envio para a placa ESP32.



**Figura 11:** Aplicação Emissor.

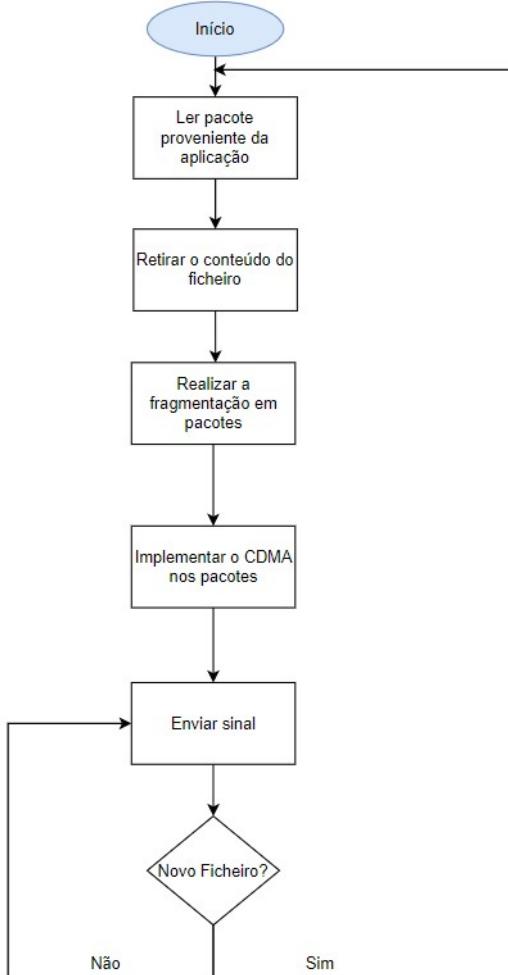
O ficheiro enviado é apresentado de seguida como prova do conteúdo recebido na subsecção do recetor.

```
msg.txt
1 Atualmente existem inúmeros espaços, interiores e exteriores,
2 cuja iluminação é feita com base em luminárias LED, que usam
3 diodos emissores de luz com o objetivo de gerar radiação na
4 zona do espetro visível. Estes LEDs são utilizados também para
5 transmitir informação através de modulações da intensidade luminosa.
6
7 O presente relatório apresenta as tarefas desenvolvidas no contexto
8 do trabalho semestral. A realização deste teve como objetivo a
9 construção de um modelo de um sistema com luminárias LED de modo
10 a conseguir transmitir informação sobre um local em que estas estão
11 instaladas. De modo a ser possível construir este modelo, foi
12 necessário criar um recetor capaz de receber a informação
13 transmitida num dado local e reproduzi-la ao utilizador.
14
15 De modo a sermos capazes de cumprir com os objetivos deste projeto
16 semestral foi necessário pôr em prática conhecimentos adquiridos
17 noutras unidades curriculares assim como adquirir novos
18 conhecimentos na área das comunicações óticas, modulações
19 e codificações, deteção e correção de erros, entre outras.
```

**Figura 12:** Aplicação Emissor - Ficheiro.

#### 4.2.3 ESP32

Relativamente ao ESP32 do lado do emissor, este contém um programa mais complexo e o seu fluxograma é apresentado na figura seguinte.



**Figura 13:** Fluxograma do programa do ESP32 do emissor.

Como é possível observar neste fluxograma é um pouco mais complexo do que o anterior, isto porque é na placa ESP32 onde é realizada a fragmentação em pacotes e a implementação do CDMA, que são processos complexos de implementar. Inicialmente é realizada a leitura do pacote enviado pela aplicação com a respetiva remoção dos dados do ficheiro a ser enviado. Com os dados do ficheiro é realizada a

fragmentação em pacotes sendo posteriormente implementado o CDMA a cada pacote. Terminada a implementação do CDMA, o sinal vai ser enviado constantemente com os vários pacotes. O envio contínuo do sinal apenas é interrompido quando é recebido um novo ficheiro. A trama utilizada para o envio de dados do ESP32 do lado do emissor para o ESP32 do lado do receptor é a demonstrada na figura seguinte.

Start Frame Delimiter (1 bytes)	File ID (1 byte)	Packet type (1 byte)	Packet Number (1 byte)	Payload Size (1 byte)	Payload (N byte)	Checksum (1 bytes)
---------------------------------------	---------------------	-------------------------	---------------------------	--------------------------	---------------------	-----------------------

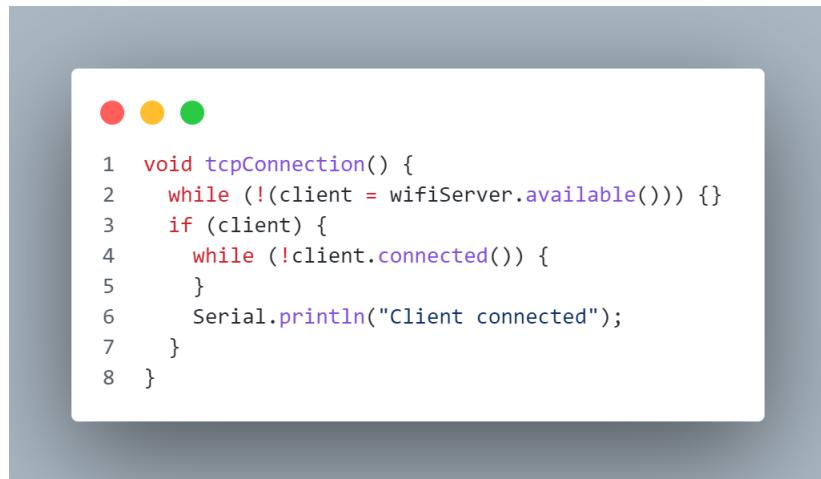
**Figura 14:** Trama Data.

Os campos utilizados na trama foram os seguintes:

- **Start Frame Delimiter** - O SFD é um byte(10101011) que indica o início de um pacote;
- **File ID** - Indica o ID do ficheiro a ser enviado;
- **Packet type** -Indica o tipo de pacote ( 1 - Start Packet, 2 - Normal Packet, 3 - Final Packet);
- **Packet Number** - Indica o número do pacote;
- **Payload Size** - Contém o tamanho do payload;
- **Payload** - Contém o conteúdo do ficheiro;
- **Checksum** - Usado para a deteção de erros.

O código desenvolvido no ESP32 está dividido em três partes essenciais, a conexão TCP/IP que comunica com a aplicação, onde recebe o ficheiro a ser transmitido, a criação dos pacotes a serem enviados pelo link ótico e o próprio envio.

A imagem seguinte representa a secção do código onde é estabelecida a conexão TCP/IP com a aplicação do emissor. Como se pode verificar o ESP32 fica à espera de receber uma conexão.



**Figura 15:** Conexão TCP/IP - ESP32.

Depois de estabelecida a conexão, o próximo passo é receber o ficheiro para ser dividido em pacotes, este processo é demonstrado na figura abaixo. Como se pode observar, o ficheiro recebido é passado como argumento para a função *packet\_creation()*. Como o próprio nome indica, esta função tem como objetivo criar os pacotes necessário para o envio do ficheiro. O código desta função encontra-se no anexo A.

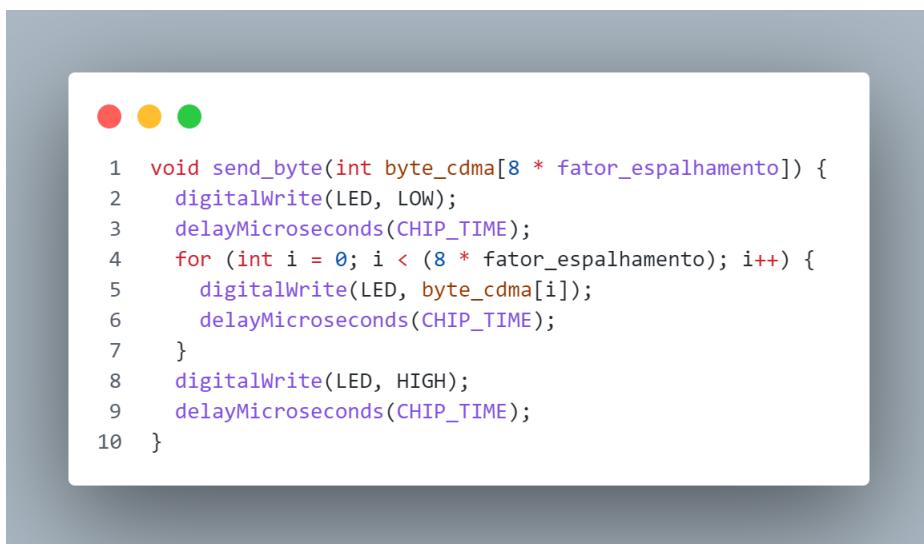


```
● ● ●

1 void loop() {
2   tcpConnection();
3   String packet_received;
4   packet_received = client.readString();
5   packet_received.remove(0, 2);
6   Serial.println(packet_received);
7   Serial.println("Packet received:");
8   Serial.println(packet_received.length());
9   byte byteArray[packet_received.length()];
10  packet_received.getBytes(byteArray, packet_received.length() + 1);
11  packet_creation(byteArray, packet_received.length());
12 }
```

**Figura 16:** Re却是 e criação de pacotes - ESP32.

Depois de criados os pacotes é então efetuado o CDMA ao byte e envio do mesmo. O envio é efetuado através da função send\_byte(), figura 17, onde consoante o valor do chip é colocada o pino no estado HIGH ou LOW durante o CHIP\_TIME.



```
● ● ●

1 void send_byte(int byte_cdma[8 * fator_espalhamento]) {
2   digitalWrite(LED, LOW);
3   delayMicroseconds(CHIP_TIME);
4   for (int i = 0; i < (8 * fator_espalhamento); i++) {
5     digitalWrite(LED, byte_cdma[i]);
6     delayMicroseconds(CHIP_TIME);
7   }
8   digitalWrite(LED, HIGH);
9   delayMicroseconds(CHIP_TIME);
10 }
```

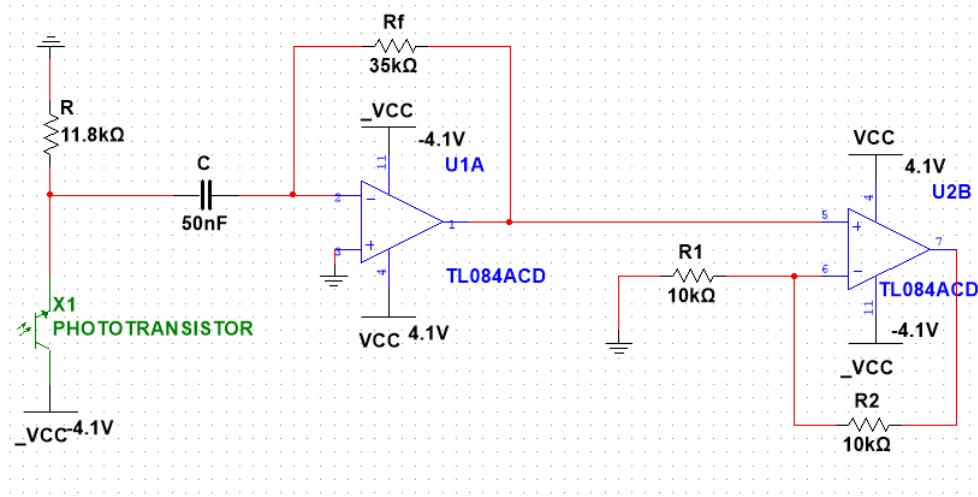
**Figura 17:** Envio de um byte - ESP32.

## 4.3 Recetor

De seguida é explicado todo o processo que foi realizado para implementar tanto o código da aplicação como da placa ESP32 do lado do recetor.

### 4.3.1 Circuito

Na figura abaixo está representado um esquema do circuito que foi projetado pelo grupo.



**Figura 18:** Circuito recetor.

Na equação (3), está apresentando o cálculo da resistência de feedback. Pretende-se que a tensão de saída Ampop seja 3.3 V, e foi usado como I uma corrente mínima de 0.1 mA.

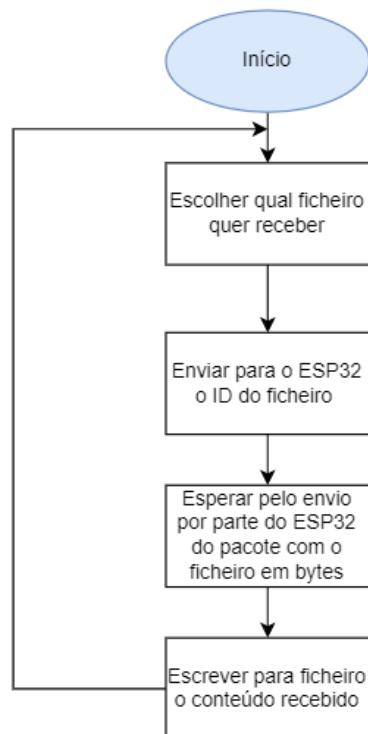
$$R_1 = \frac{V_o}{I} \Leftrightarrow R_1 = \frac{3.3V}{0.1mA} \Leftrightarrow R_1 = 33k\Omega \quad (4)$$

A equação (4) foi usada para calcular os valores teóricos para a resistência R, usando condensadores presentes na caixa de componentes e atribuindo uma frequência de corte que fosse no mínimo metade da frequência da onda, no entanto, os valores teóricos não estavam a ter os resultados pretendidos na onda e por isso, foram obtidos experimentalmente.

$$f_c = \frac{1}{2\pi R_2 C_1} \quad (5)$$

#### 4.3.2 Aplicação

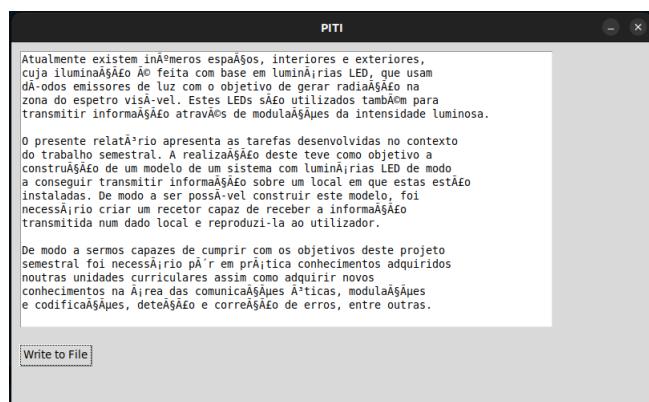
Relativamente à aplicação do recetor, o fluxograma é apresentado na figura abaixo.



**Figura 19:** Fluxograma do recetor.

Inicialmente, o utilizador vai escolher o ficheiro que pretende receber através da aplicação. Em seguida envia para o ESP32 o ID do ficheiro pretendido e aguarda o envio por parte do ESP32 do pacote com o ficheiro em bytes e escreve para um ficheiro o conteúdo recebido. Terminando estes passos, o recetor fica novamente disponível para receber novos ficheiros.

De seguida é apresentado o ecrã recebido pelo recetor quando este recebe o ficheiro proveniente do ESP32.



**Figura 20:** Aplicação do recetor.

Com o ficheiro recebido o utilizador tem a opção de escrever o conteúdo num ficheiro output como demonstrado na figura que se segue. Comparando este ficheiro ao apresentado na subsecção do emissor é possível constatar que o ficheiro foi enviado e recebido com sucesso sem erros.

```

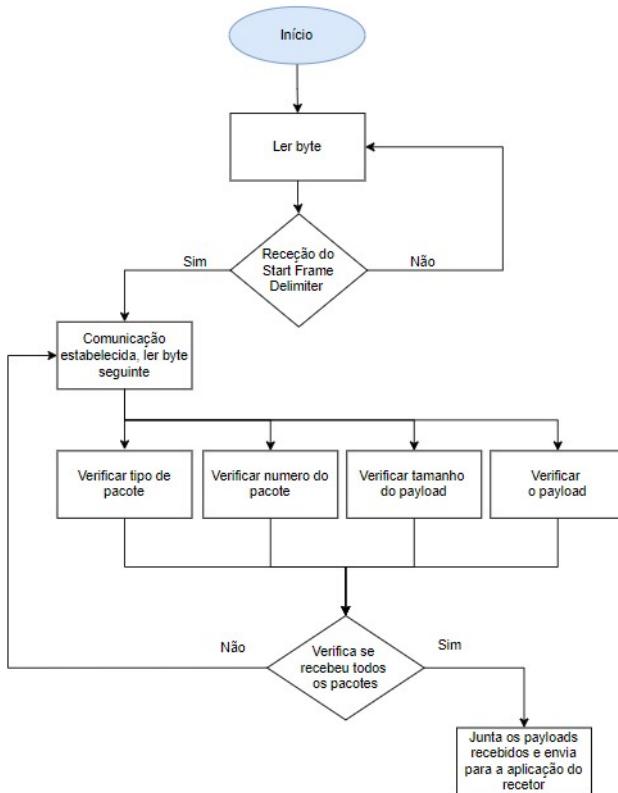
msg.txt
1 Atualmente existem inúmeros espaços, interiores e exteriores,
2 cuja iluminação é feita com base em luminárias LED, que usam
3 diodos emissores de luz com o objetivo de gerar radiação na
4 zona do espetro visível. Estes LEDs são utilizados também para
5 transmitir informação através de modulações da intensidade luminosa.
6
7 O presente relatório apresenta as tarefas desenvolvidas no contexto
8 do trabalho semestral. A realização deste teve como objetivo a
9 construção de um modelo de um sistema com luminárias LED de modo
10 a conseguir transmitir informação sobre um local em que estas estão
11 instaladas. De modo a ser possível construir este modelo, foi
12 necessário criar um recetor capaz de receber a informação
13 transmitida num dado local e reproduzi-la ao utilizador.
14
15 De modo a sermos capazes de cumprir com os objetivos deste projeto
16 semestral foi necessário pôr em prática conhecimentos adquiridos
17 noutras unidades curriculares assim como adquirir novos
18 conhecimentos na área das comunicações óticas, modulações
19 e codificações, deteção e correção de erros, entre outras.

```

**Figura 21:** Ficheiro recebido.

#### 4.3.3 ESP32

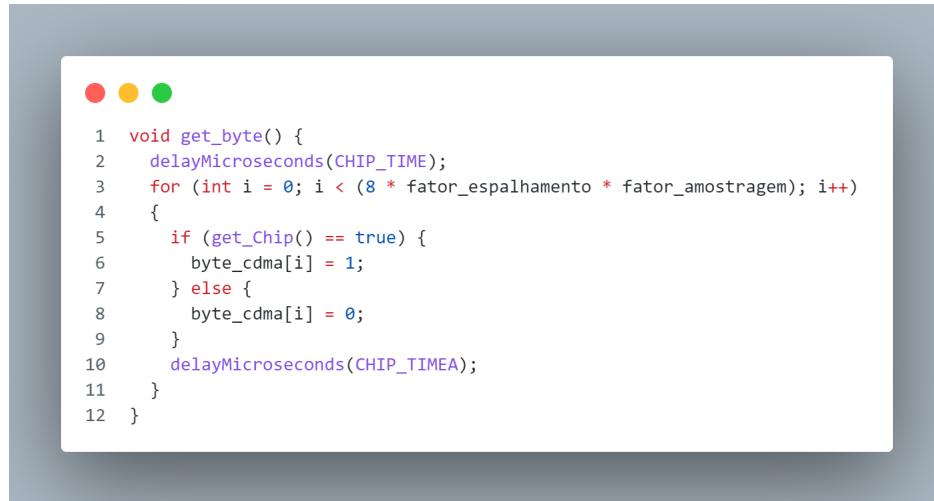
Relativamente ao ESP32 do lado do recetor, na figura abaixo está apresentado um fluxograma que explica o funcionamento do mesmo.



**Figura 22:** Fluxograma do programa do ESP32 - Recetor.

Inicialmente o ESP32 do recetor está sempre a ler 1 byte e fazer a verificação se esse byte se trata do Start Frame Delimiter (SFD), se o byte não se tratar do SFD o recetor volta a ler 1 byte. Quando o byte lido for igual ao SFD, a comunicação entre o recetor e emissor é estabelecida e em seguida são lidos os bytes seguintes. É verificado o tipo de pacote, o número de pacote, o tamanho do *payload* e o conteúdo do *payload*. Depois de feita a verificação destes campos da trama é feita a verificação da receção de todos os pacotes. Caso não tenham sido recebidos todos os pacotes, volta ao passo em que é estabelecida a ligação. Caso todos os pacotes tenham sido recebidos, junta-se todos os *payloads* recebidos e em seguida são enviados para a aplicação do recetor.

Do código implementado no ESP32, faz sentido destacar a função *get\_byte()*, figura 23, em que esta tem como objetivo ler os *chips* recebidos para serem descodificados, resultando deste processo um byte.



```
● ● ●  
1 void get_byte() {  
2     delayMicroseconds(CHIP_TIME);  
3     for (int i = 0; i < (8 * fator_espalhamento * fator_amostragem); i++)  
4     {  
5         if (get_Chip() == true) {  
6             byte_cdma[i] = 1;  
7         } else {  
8             byte_cdma[i] = 0;  
9         }  
10        delayMicroseconds(CHIP_TIMEA);  
11    }  
12 }
```

**Figura 23:** Recepção de um byte - ESP32.

O byte descodificado é passado então como argumento para função *verify\_byte()*, figura 24, em que é nesta onde se encontra todo o tratamento do byte recebido. Consoante o tipo de byte recebido são efetuados processos diferentes. Em primeiro lugar é esperado o byte do tipo SFD e só depois é feito o tratamento de cada byte que pertence a uma trama.

```

1 int verify_byte(byte byte_recebido) {
2     if (byte_recebido == 0b10101011) {
3         byte_pacote = 1;
4         return 1;
5     }if ( byte_pacote == 1) {
6         packet_id = byte_recebido;
7         if (packet_id == file_id) {
8             return 2;
9         }
10    } if ( byte_pacote == 2) {
11        packet_type = byte_recebido;
12        return 3;
13    }if ( byte_pacote == 3) {
14        packet_number = byte_recebido;
15        return 4;
16    }if ( byte_pacote == 4 ) {
17        packet_payload_size = byte_recebido;
18        return 5;
19    }if ( byte_pacote == 5 ) {
20        payload[aux_payload] = byte_recebido;
21        aux_payload++;
22        if (aux_payload == packet_payload_size) {
23            return 6;
24        } else {
25            return 5;
26        }
27    }
28    if ( byte_pacote == 6 ) {
29        byte crc_array[packet_payload_size];
30        for (int i = 0; i < packet_payload_size; i++) {
31            crc_array[i] = payload[i];
32        }
33        packet_checksum = crcx::crc8(crc_array, packet_payload_size);

```

**Figura 24:** Tratamento de um byte - ESP32.

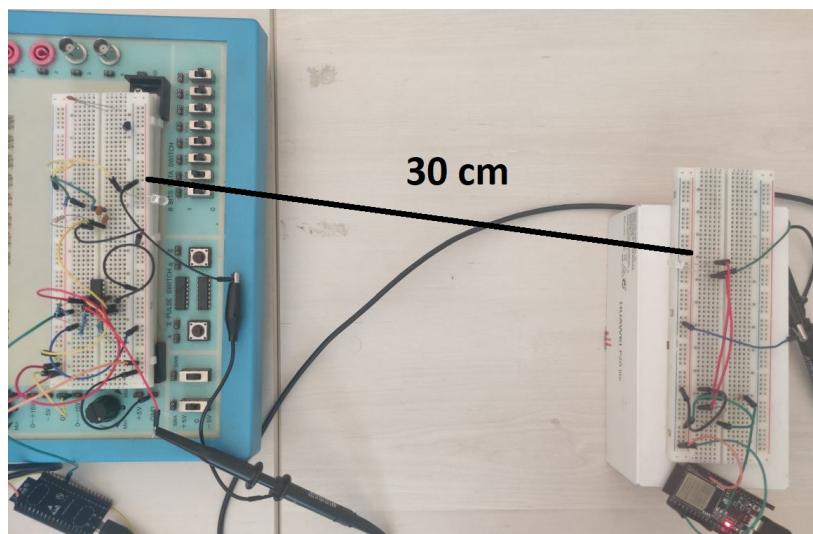
## 5 Testes e Discussão de Resultados

De modo a testar a solução implementada pelo grupo foi necessário realizar um conjunto de testes que demonstrasse os resultados relativos às duas métricas de desempenho principais, o alcance (distância máxima entre o emissor de luz e o fotodetector) e o tempo mínimo para a transmissão da mensagem entre o emissor e o receptor(sem erros).

### 5.1 Distância máxima

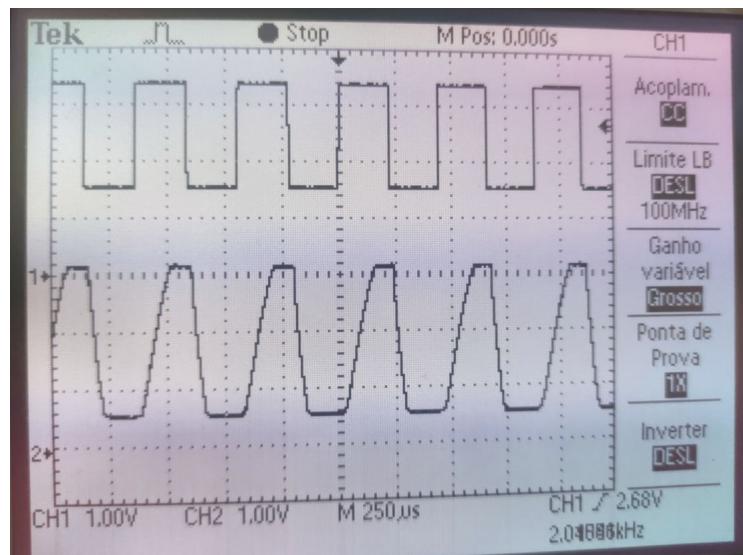
De modo a conseguir realizar os testes para tirar o tempo mínimo na transmissão dos dados foi necessário primeiramente descobrir qual a distância máxima em que era possível enviar informação com a menor percentagem de erro. A distância atingida foi de aproximadamente 52 centímetros quando o tempo do bit era alto pois tinha uma frequência baixa e a onda era bem visível. No entanto o grupo decidiu optar por melhorar a velocidade de envio comprometendo assim a distância, que passou dos 52 cm para os 32 cm.

De seguida é apresentada uma imagem com a distância atingida pelo grupo quando a comunicação era efetuada sem erros.



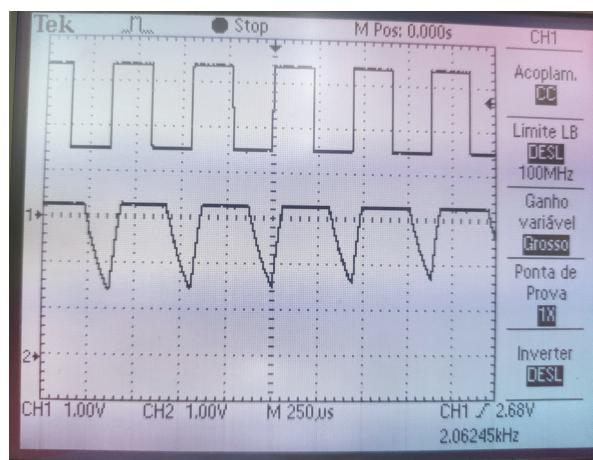
**Figura 25:** Teste distância.

Na próxima figura é possível observar 2 ondas, sendo que a de cima corresponde à onde que sai do emissor e a de baixo a onda que é retirada no recetor quando estes se encontram a 32 cm.



**Figura 26:** Visualização do sinal a 32 cm.

Após ser aumentada a distância para os 40 cm é possível observar a onda do recetor identificada na figura 27 que ficava tão distorcida que já não era possível receber a informação sem erros.



**Figura 27:** Visualização do sinal a 40 cm.

## 5.2 Testes do tempo de transferência

De modo a perceber quanto tempo levava a transmitir um ficheiro do emissor para o recetor foi necessário proporcionar um ambiente com certos parâmetros iguais como a quantidade de luz incidente na sala, medida em lux em que  $1\text{lux} = 1\text{lumen}/m^2$ , a distância entre o emissor e recetor e o tamanho do ficheiro, entre outros. De seguida são apresentados os valores utilizados para a realização dos testes.

Luz ambiente	255 lux
Distância entre o emissor e o recetor	32 cm
Tamanho do ficheiro	1123 bytes
Tamanho do <i>payload</i>	64 bytes
Tempo do bit	220 $\mu s$

Com os valores anteriores devidamente estipulados foram realizados testes alterando os valores do fator de espalhamento e de amostragem de modo a ser possível concluir quais seriam os melhores valores para estes parâmetros que permitissem um baixo tempo de transmissão, métrica crucial para o bom funcionamento da nossa solução. De seguida é apresentada uma tabela com os valores obtidos da realização dos testes.

**Tabela 4:** Testes dos tempos de transferência.

Fator de Espalhamento	Fator de Amostragem	Tempo de Transferência (s)	Débito Útil (bps)	Débito Bruto (bps)
2	4	6.63	1507.088989	1651.88537
3	4	9.3	1074.408602	1177.634409
4	4	11.19	892.9401251	978.7310098
5	3	14.33	697.2784368	764.2707606
6	3	16.16	618.3168317	677.7227723

De acordo com a tabela anterior é possível observar que nos primeiros 3 testes, onde foi utilizado um fator de amostragem de 4, os tempos de transferência eram tão melhores quanto menor fosse o fator de espalhamento, pois eram enviados menos bits, o que poderia aumentar a probabilidade de erro mas caso não aconteceu. Após isso foram realizados 2 testes com um menor fator de amostragem mas aumentando o fator de espalhamento cujos resultados tendem a piorar visto o número de bits enviados serem superiores relativamente aos anteriores. Com a obtenção do tempo de transferência e como o ficheiro era o mesmo em todos os testes, foi possível também obter tanto o débito útil como o bruto em que ao observar a tabela é possível retirar que para um fator de espalhamento de 2 e fator de amostragem de 4 é possível enviar 1651 bps em que 1507 correspondem ao *payload*.

Relativamente à taxa de transmissão ou *bit rate* da solução implementada esta é calculada tendo em conta a seguinte expressão com R a indicar a taxa de transmissão e Tb o tempo do bit.

$$R = \frac{1}{T_b} \quad (6)$$

Substituindo na expressão anterior o Tb pelos 220 microsegundos é possível obter uma taxa de transmissão de aproximadamente 4545 bps em que esta taxa quando comparada com o débito é substancialmente maior. Isto acontece porque apesar dos termos serem parecidos, enquanto que o *bit rate* mede a quantidade de informação que um canal consegue transmitir por unidade de tempo, o débito bruto (*throughput*) mede a quantidade de informação que é recebida com sucesso num canal por unidade de tempo. Alguns fatores que podem afetar esta diferença de valores é o facto de nem todos os bits enviados serem recebidos, a presença de *overheads* presentes na comunicação, entre outros.

### 5.3 Testes do tamanho do payload

Após termos a melhor distância possível e o menor tempo de transmissão do ficheiro foram realizados testes para ver se o facto de aumentar ou diminuir o tamanho do *payload* poderia ter influência no tempo de transmissão. Os valores obtidos da realização dos testes foram os apresentados a seguir.

**Tabela 5:** Testes do tamanho do *payload*.

Tamanho do <i>payload</i>	Tempo de Transferência (s)	Taxa de transmissão/pacote (s)
32	17.3	0.066880067
64	16.16	0.123200123
128	16.05	0.235840236

Da tabela anterior é possível concluir que através da alteração do tamanho do *payload*, o tempo de transmissão não é suficientemente afetado para que seja notada uma grande diferença, apesar de ser possível observar que quanto maior for o pacote, o tempo de transferência tende a descer. Isto acontece porque ao aumentar o tamanho do pacote o número de bits total de cabeçalho é diminuído.

## **6 Conclusão**

Após a conclusão do projeto podemos afirmar que nos sentimos satisfeitos com o resultado e com o nosso desempenho uma vez que cumprimos com os objetivos propostos. O facto de termos elaborado um planeamento antes da realização do projeto ajudou na coordenação e organização do que cada elemento do grupo tinha de fazer.

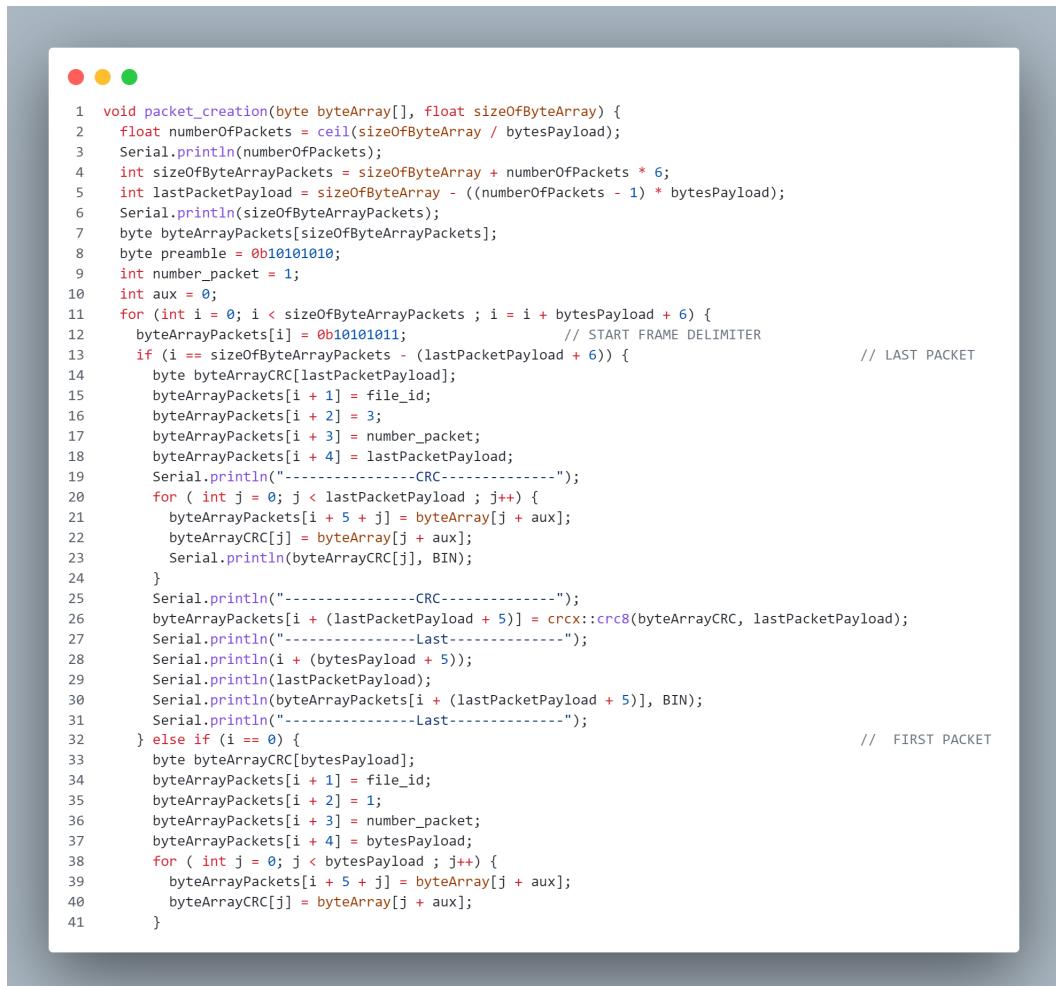
O grupo teve algumas dificuldades na montagem do circuito receptor, um pouco por causa de alguns componentes serem obtidos experimentalmente, o que nos demorou algum tempo para aperfeiçoar. Teve dificuldades também na sincronização do código após a implementação do CDMA e também com o tempo que demorava a enviar o ficheiro de 1000 bytes pelo canal ótico, no entanto estes problemas foram superados com o auxílio dos professores e com alguma pesquisa feita pelo grupo.

Concluindo, os elementos do grupo estão satisfeitos com o trabalho que foi realizado, bem como com a aprendizagem que foi adquirida com a realização do mesmo, salientando o que foi aprendido na área das comunicações óticas, modulação e implementação de sinais e deteção e correção de erros.

## **Referências**

- [1] Phototransistor basics. (n.d.). Phototransistor Basics. Retrieved May 2022, from <https://jf-parede.pt/phototransistor-basics>
- [2] SENSORES-Fototransistores. (n.d.). Sensores. Retrieved May 16, 2022, from <https://ppgenfis.if.ufrgs.br/mef004/20061/Cesar/SENSORES-Fototransistor.html>
- [3] Fototransistor. (2021, October 8). Wikipedia. Retrieved May 16, 2022, from <https://pt.wikipedia.org/wiki/Fototrans%C3%ADstor>
- [4] What is photodiode. (n.d.). What Is Photodiode. Retrieved May 2022, from <https://jf-parede.pt/what-is-photodiode>
- [5] SENSORES-Fotodiodos. (n.d.). Sensores. Retrieved May 2022, from <https://ppgenfis.if.ufrgs.br/mef004/20061/Cesar/SENSORES-Fotodiodo.html>

## A packet\_creation()



The image shows a screenshot of the Arduino IDE. A code editor window is open, displaying the `packet_creation()` function. The code is written in C++ and uses the `Serial` library for communication. The function takes a byte array and a float size as parameters. It calculates the number of packets needed, prints it to the serial port, and then creates a packet structure. The packet starts with a frame delimiter (0b10101011), followed by file\_id, number\_packet, lastPacketPayload, and a series of data bytes. A CRC is calculated and appended at the end. The code handles both first and subsequent packets in the sequence.

```
1 void packet_creation(byte byteArray[], float sizeOfByteArray) {
2     float numberPackets = ceil(sizeOfByteArray / bytesPayload);
3     Serial.println(numberPackets);
4     int sizeOfByteArrayPackets = sizeOfByteArray + numberPackets * 6;
5     int lastPacketPayload = sizeOfByteArray - ((numberPackets - 1) * bytesPayload);
6     Serial.println(sizeOfByteArrayPackets);
7     byte byteArrayPackets[sizeOfByteArrayPackets];
8     byte preamble = 0b10101010;
9     int number_packet = 1;
10    int aux = 0;
11    for (int i = 0; i < sizeOfByteArrayPackets ; i = i + bytesPayload + 6) {
12        byteArrayPackets[i] = 0b10101011; // START FRAME DELIMITER
13        if (i == sizeOfByteArrayPackets - (lastPacketPayload + 6)) { // LAST PACKET
14            byte byteArrayCRC[lastPacketPayload];
15            byteArrayPackets[i + 1] = file_id;
16            byteArrayPackets[i + 2] = 3;
17            byteArrayPackets[i + 3] = number_packet;
18            byteArrayPackets[i + 4] = lastPacketPayload;
19            Serial.println("-----CRC-----");
20            for ( int j = 0; j < lastPacketPayload ; j++) {
21                byteArrayPackets[i + 5 + j] = byteArray[j + aux];
22                byteArrayCRC[j] = byteArray[j + aux];
23                Serial.println(byteArrayCRC[j], BIN);
24            }
25            Serial.println("-----CRC-----");
26            byteArrayPackets[i + (lastPacketPayload + 5)] = crcx::crc8(byteArrayCRC, lastPacketPayload);
27            Serial.println("-----Last-----");
28            Serial.println(i + (bytesPayload + 5));
29            Serial.println(lastPacketPayload);
30            Serial.println(byteArrayPackets[i + (lastPacketPayload + 5)], BIN);
31            Serial.println("-----Last-----");
32        } else if (i == 0) { // FIRST PACKET
33            byte byteArrayCRC[bytesPayload];
34            byteArrayPackets[i + 1] = file_id;
35            byteArrayPackets[i + 2] = 1;
36            byteArrayPackets[i + 3] = number_packet;
37            byteArrayPackets[i + 4] = bytesPayload;
38            for ( int j = 0; j < bytesPayload ; j++) {
39                byteArrayPackets[i + 5 + j] = byteArray[j + aux];
40                byteArrayCRC[j] = byteArray[j + aux];
41            }
42        }
43    }
44}
```

**Figura 28:** Função packet\_creation() - 1.

```

● ● ●

1     byteArrayPackets[i + (bytesPayload + 5)] = crcx::crc8(byteArrayCRC, bytesPayload); // NORMAL PACKET
2 } else {
3     byteArrayCRC[bytesPayload];
4     byteArrayPackets[i + 1] = file_id;
5     byteArrayPackets[i + 2] = 2;
6     byteArrayPackets[i + 3] = number_packet;
7     byteArrayPackets[i + 4] = bytesPayload;
8     for ( int j = 0; j < bytesPayload ; j++) {
9         byteArrayPackets[i + 5 + j] = byteArray[j + aux];
10        byteArrayCRC[j] = byteArray[j + aux];
11    }
12    byteArrayPackets[i + (bytesPayload + 5)] = crcx::crc8(byteArrayCRC, bytesPayload);
13 }
14 number_packet = number_packet + 1;
15 aux = aux + bytesPayload;
16
17 }
18 int y = 0;
19 for (int i = 0; i < 8 * fator_espalhamento; i++) {
20     if (y == tamanho_codigo) {
21         y = 0;
22     }
23     codigo_tamanho[i] = codigo[y];
24     y = y + 1;
25 }
26
27 int codigo_amostrado[8 * fator_espalhamento * fator_amostragem];
28 int auxx = 0;
29 int k = 0;
30 for (int i = 0; i < 8 * fator_espalhamento; i++) {
31     for (int t = 0; t < fator_amostragem; t++) {
32         codigo_amostrado[auxx + t] = codigo_tamanho[i];
33     }
34     auxx = auxx + fator_amostragem;
35 }
36 while(true){
37     for (int i = 0; i < sizeOfByteArrayPackets; i++) {
38         int aux2 = 0;
39         for (int r = 0; r < 8; r++) {
40             int bitvar = bitRead(byteArrayPackets[i], 7 - r);
41             if (bitvar == 0) {
42                 bitvar = -1;
43             }
44             for (int j = 0; j < fator_espalhamento * fator_amostragem; j++) {
45                 byte_cdma[j + aux2] = bitvar * codigo_amostrado[j + aux2];
46             }
47             aux2 = aux2 + fator_espalhamento * fator_amostragem;
48         }
49         for (int i = 0; i < 8 * fator_espalhamento * fator_amostragem; i++) {
50             if (byte_cdma[i] == -1) {
51                 byte_cdma[i] = 0;
52             }
53         }
54         send_byte(byte_cdma);
55     }
56 }
57 }
```

**Figura 29:** Função packet\_creation() - 2.