ESCOLA NAVAL

talant de bien faire

Ricardo Alexandre Inácio dos Santos Caldeira Chaves

Traçador de Curvas para Transístores com Interface para Dispositivos Móveis

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Engenheiros Navais, ramo de Armas e Eletrónica

Alfeite

2021

ESCOLA NAVAL

talant de bien faire

Ricardo Alexandre Inácio dos Santos Caldeira Chaves

Traçador de Curvas para Transístores com Interface para Dispositivos Móveis

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Engenheiros Navais, ramo de Armas e Eletrónica

Orientação de: Vítor Manuel Rodrigues Viegas
Coorientação de: Bruno Duarte Damas

O Aluno Mestrando	O Orientador	O Coorientador
Caldeira Chaves		
Ricardo Alexandre Inácio dos Santos Caldeira Chaves	Vítor Manuel Rodrigues Viegas	Bruno Duarte Damas
ASPOF EN-AEL	Professor Doutor	Professor Doutor

Alfeite

2021

Resumo

Um traçador de curvas é um equipamento de instrumentação eletrónica que adquire e traça as curvas características de tensão e de corrente aos terminais de um componente eletrónico, como díodos e transístores. Os resultados são apresentados em forma de tabelas de valores e de gráficos X-Y, que representam as curvas que caracterizam o funcionamento próprio do componente, conhecidas por "curvas características".

O traçador de curvas clássico é imprescindível num laboratório de eletrónica, principalmente em âmbito académico, pois permite aos alunos, de forma rápida e intuitiva, visualizar na prática conceitos nucleares nas áreas de eletrónica e medidas elétricas. Tratando-se de instrumentos dispendiosos, pretendeu-se com este trabalho desenvolver uma alternativa de baixo custo.

O equipamento criado trata-se de um dispositivo portátil de baixo custo que replica o funcionamento de um traçador de curvas, capaz de caracterizar transístores de junção bipolar, do tipo NPN e do tipo PNP, e transístores de efeito de campo de semicondutor óxido-metálico, de canal N e canal P. O dispositivo assenta num microcontrolador Arduino, um circuito de condicionamento de sinal e um circuito de alimentação, integrados numa única placa de circuito impresso. A interface com o utilizador é feita através de uma aplicação compatível com dispositivos móveis de sistema operativo Android, que comunica com o microcontrolador via ligação *Bluetooth*. A aplicação permite iniciar a ligação com o dispositivo, selecionar o tipo de transístor e de ensaio a realizar, visualizar os resultados do ensaio e exportar os dados para plataformas de terceiros.

Palavras-chave: Traçador de curvas; Transístor; Arduino; Dispositivo móvel; *Bluetooth*.

Abstract

A curve tracer is an electronic measurement device which plots current-voltage characteristics of electronic components, such as diodes and transistors. The results are presented in table form or plotted as X-Y graphs, which represent the specific curves that characterize the function of the given component, known as "characteristic curves".

The curve tracer is essential in any electronics lab, even more for academic purposes, as it enables students to visualize nuclear concepts about electronics and electric measures, in a fast and intuitive manner. As curve tracers are expensive devices, it was the intention with this project to develop a low-cost solution.

The developed solution is a low-cost portable device which replicates the function of a curve tracer, capable of characterizing type NPN and type PNP bipolar junction transistors and channel N and channel P metal-oxide-semiconductor field-effect transistors. The device is composed of an Arduino microcontroller, a signal conditioning circuit and a power supply circuit, all integrated in a single printed circuit board. The user interface is provided by an Android mobile device compatible app. The app communicates with the microcontroller via Bluetooth and enables the user to initiate the connection with the portable device, select the type of transistor and type of curve to run the test on and to visualize and export the test results to third-party platforms.

Keywords: Curve Tracer; Transistor; Arduino; Mobile Device; Bluetooth.

Índice

Resumo ..... *V*

Abstract ..... *VII*

Índice ..... *IX*

Índice de Figuras ..... *XI*

Lista de Abreviaturas, Siglas e Acrónimos ..... *XIII*

1 Introdução *1*

1.1 Motivação **1**

1.2 Objetivo da Dissertação **2**

1.3 Estrutura da Dissertação **2**

2 Enquadramento Teórico *3*

2.1 O Transístor **3**

2.2 Curvas Características de um Transístor Bipolar **9**

2.2.1 Curva de Entrada 11

2.2.2 Curva de Transferência 12

2.2.3 Curva de Saída 12

2.3 Curvas Características de um Transístor MOSFET **14**

3 Desenvolvimento *17*

3.1 Arquitetura Geral **17**

3.2 Hardware **18**

3.2.1 Circuito de Condicionamento de Sinal 19

3.2.2 Circuito de Alimentação 22

3.2.3 Placa de Circuito Impresso 23

3.2.4 Protótipo Final 25

3.3 Software **28**

3.3.1 Microcontrolador Arduino 28

3.3.2 Aplicação para Dispositivos Android 30

4 Ensaios Experimentais *37*

4.1 Transístores Bipolares NPN 37

4.2 Transístores Bipolares PNP 40

4.3 Transístores MOSFET 43

5 Conclusão 45

6 Bibliografia 47

7 Apêndices 49

A. Placa de Circuito Impresso 49

A.1. Face Superior 50

A.2. Face Inferior 51

A.3. Lista dos Componentes 52

A.4. Disposição dos Componentes 53

B. Código Fonte: Arduino 55

C. Código Fonte: Android Studio 67

C.1. AndroidManifest.xml 67

C.2. FirstActivity.java 68

C.3. MainActivity.java 72

C.4. DeviceListAdapter.java 85

C.5. Strings.xml 86

C.6. Provider_paths.xml 86

C.7. Activity_first.xml 86

C.8. Activity_main.xml 87

C.9. Devicelistadapter_layout.xml 89

C.10. Dropdown_layout.xml 90

C.11. Spinner_layout.xml 90

C.12. Sharebutton_layout.xml 90

Índice de Figuras

Figura 1. Constituição de uma junção PN. 4
Figura 2. Formação da zona de depleção numa junção PN. 4
Figura 3. Construção e simbologia de transístores bipolares NPN e PNP. 5
Figura 4. Polarização de transístor bipolar NPN na zona ativa. 6
Figura 5. Estrutura física de transístor MOSFET de canal N. 7
Figura 6. Canal criado entre a fonte e dreno de transístor NMOS. 8
Figura 7. Montagem em emissor comum de um transístor bipolar NPN. 9
Figura 8. Representação gráfica da curva de entrada de um transístor. 11
Figura 9. Representação gráfica da família de curvas de saída 13
Figura 10. Perfil do canal induzido devido à tensão dreno-fonte. 15
Figura 11. Curva de saída teórica de transístores MOSFET de canal N. 16
Figura 12. Diagrama de blocos da arquitetura geral do sistema de medida. 17
Figura 13. Circuito de condicionamento de sinal. 19
Figura 14. Circuito de polarização de transístores BJT NPN 20
Figura 15. Circuito de polarização de transístores BJT PNP. 22
Figura 16. Diagrama do circuito de alimentação. 23
Figura 17. Esquema do circuito de condicionamento de sinal. 24
Figura 18. Face superior da PCB. 24
Figura 19. Face inferior da PCB. 24
Figura 20. Conector do DUT. 25
Figura 21. Placa de circuito impresso finalizada. 26
Figura 22. PCB e suporte da bateria fixados na caixa 26
Figura 23. Protótipo final - interior. 27
Figura 24. Protótipo final - exterior. 27
Figura 25. Fluxograma do funcionamento das funções específicas do Arduino. 30
Figura 26. Fluxograma do funcionamento da aplicação móvel. 32
Figura 27. Janela de solicitação para ativação do Bluetooth. 33
Figura 28. Página inicial da aplicação. 33
Figure 29. Página principal da aplicação. 34
Figura 30. Spinner para seleção do tipo de transístor. 35

Figura 31. Spinner para seleção do tipo de curva característica..... 35
Figura 32. Visualização gráfica dos resultados do ensaio. 36
Figura 33. Janela para exportação dos resultados do ensaio..... 36
Figura 34. Ensaio de transístor NPN: curva característica de entrada..... 37
Figura 35. Ensaio de transístor NPN: curva característica de transferência. 38
Figura 36. Ensaio de transístor NPN: curvas características de saída..... 39
Figura 37. Ensaio de transístor PNP: curva característica de entrada. 40
Figura 38. Ensaio de transístor PNP: curva característica de transferência..... 41
Figura 39. Ensaio de transístor PNP: curvas características de saída. 42
Figura 40. Curva de saída de transístor MOSFET de canal N..... 43
Figura 41. Curva de saída de transístor MOSFET de canal P. 44

Lista de Abreviaturas, Siglas e Acrónimos

AMPOP	Amplificador Operacional
BJT	*Bipolar Junction Transistor*
CSV	*Comma Separated Values*
DAC	*Digital-to-Analog Converter*
DC	*Direct Current*
DUT	*Device Under Test*
I2C	*Inter-Integrated Circuit*
IDE	*Integrated Development Environment*
MAC	*Media Access Control*
MOSFET	*Metal-Oxide-Semiconductor Field-Effect Transistor*
PWM	*Pulse Width Modulation*
SMU	*Source Measure Unit*

1 Introdução

1.1 Motivação

Um traçador de curvas é um equipamento de instrumentação eletrónica que adquire e traça as curvas características de tensão e de corrente aos terminais de um componente eletrónico, como díodos e transístores. Os resultados são apresentados em forma de tabelas de valores e de gráficos X-Y, que representam as curvas que caracterizam o funcionamento próprio do componente, conhecidas por "curvas características" [1].

O traçador de curvas é um instrumento muito útil num laboratório de eletrónica, pois permite conhecer as características dos transístores de forma fácil, rápida e intuitiva. É um equipamento que facilita, quer a manutenção de circuitos eletrónicos existentes, quer o desenho e construção de novos circuitos. Em ambiente académico permite aos alunos validar e consolidar conceitos nucleares nas áreas de eletrónica e medidas elétricas.

Atualmente, já não é comum encontrar traçadores de curvas clássicos, sendo a sua funcionalidade aproximadamente replicada por unidades de alimentação e medida (*source measure unit*, SMU) [2] com software próprio para esse efeito. Contudo, os traçadores de curvas continuam a ser populares, devido à sua simplicidade de utilização face às SMU [3]. Quer os traçadores de curvas clássicos, quer as SMU, são instrumentos dispendiosos, com modelos de entrada a custarem alguns milhares de euros. Para além disso, em contexto de ensino à distância, a portabilidade dos instrumentos de medida ganha especial relevo, algo que este tipo de equipamentos não consegue proporcionar.

Face ao exposto, pretende-se construir um traçador de curvas para transístores destinado a equipar o laboratório de eletrónica da Escola Naval, que seja portátil e acessível em termos económicos, consolidando, através da prática, os conhecimentos e competências adquiridos ao longo do percurso académico.

1.2 Objetivo da Dissertação

Com esta dissertação pretende-se desenvolver um sistema constituído por um dispositivo portátil e uma aplicação para dispositivos móveis que replique a funcionalidade de um traçador de curvas tradicional. O dispositivo portátil deverá ser de pequenas dimensões e de baixo custo, e deverá permitir extrair as curvas características de transístores. A aplicação deverá funcionar como interface gráfica com o utilizador ligando-se ao dispositivo portátil através de *Bluetooth*. A aplicação deverá ser compatível com o sistema operativo Android e deverá permitir a escolha do transístor a ensaiar, a seleção do tipo de curva, a visualização gráfica dos resultados do ensaio e a exportação dos dados para outros suportes.

1.3 Estrutura da Dissertação

A estrutura deste documento procura refletir o processo de desenvolvimento do trabalho realizado.

O capítulo 2 (Enquadramento Teórico), explica alguns conceitos importantes para a compreensão do trabalho, nomeadamente o que é um transístor e como funciona, e como se obtêm as suas curvas características.

O capítulo 3 (Desenvolvimento) descreve o trabalho desenvolvido, divido em três subcapítulos, um primeiro explicativo da arquitetura geral implementada, um segundo dedicado ao hardware, e um terceiro dedicado ao software.

O capítulo 4 (Ensaios Experimentais) analisa e discute os resultados obtidos nos ensaios experimentais do dispositivo desenvolvido.

Finalmente, o capítulo 5 (Conclusão) apresenta as conclusões e lições aprendidas com a realização deste trabalho, bem como sugestões de desenvolvimento e implementação futuros.

2 Enquadramento Teórico

Este capítulo visa explicar alguns conceitos base para contextualizar o restante trabalho. Encontra-se dividido em três subcapítulos, um sobre o funcionamento dos transístores e outros dois sobre as suas curvas características.

2.1 O Transístor

Existem diferentes tipos de transístores, dependendo da sua forma de funcionamento. No âmbito deste trabalho, serão abordados os transístores de junção bipolar (*bipolar junction transistor*, BJT) nas suas duas configurações, NPN e PNP, e transístores de efeito de campo de semicondutor óxido-metálico (*metal-oxide-semiconductor field-effect transistors*, MOSFET), de canal N e canal P.

Um transístor de junção bipolar é um componente eletrónico semicondutor de três terminais, identificados como base, emissor e coletor. O funcionamento do transístor assenta na combinação de duas junções PN, que se passa a explicar nos parágrafos que se seguem.

Os semicondutores são normalmente feitos de silício, elemento semicondutor com quatro eletrões na camada de valência. Consegue-se um semicondutor do tipo N ao adicionar impurezas pentavalentes, num processo denominado de dopagem, ao semicondutor de silício puro. Estas impurezas com cinco eletrões na camada de valência, normalmente fósforo, recombinam-se com os átomos de silício deixando um eletrão livre, que, estando fracamente ligado ao seu átomo, constituí um transportador de carga elétrica negativa. Os semicondutores do tipo P são conseguidos da mesma forma, ao dopar o semicondutor de silício com impurezas trivalentes, como o boro. A recombinação dos elementos cria "espaços vazios" prontos a receber eletrões, referidos como lacunas, que podem ser interpretadas como transportadores de carga elétrica positiva.

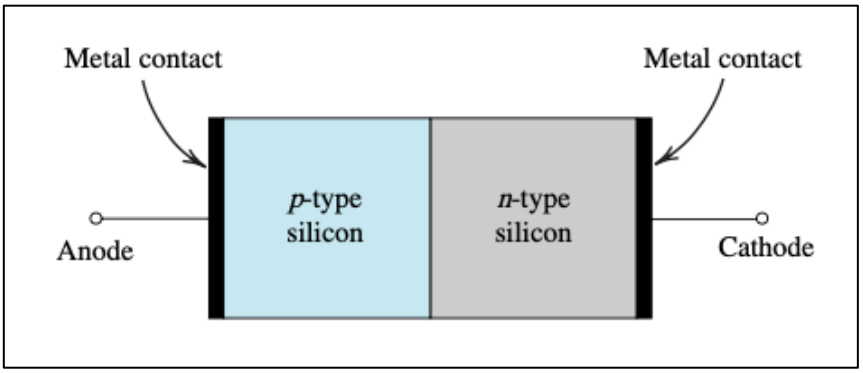


Figura 1. Constituição de uma junção PN.[1]

Ao se juntar os dois tipos de semicondutores, P e N, conforme é mostrado na Figura 1, na zona fronteira, existe a recombinação de eletrões e lacunas devido ao fenómeno de difusão. Na zona próxima da junção, os eletrões livres do semicondutor tipo N transitam para o semicondutor do tipo P, ocupando as lacunas existentes. Quando um eletrão abandona o seu átomo na região N, este fica com carga positiva. Por sua vez, quando um átomo na região P aceita este eletrão, este fica com carga negativa. Assim, são criadas cargas fixas em ambos os lados da junção, que dão origem a uma barreira de potencial, de aproximadamente 0,7 V, denominada zona de depleção, que se opõe ao trânsito de carga elétrica em ambos os sentidos, conforme a Figura 2.

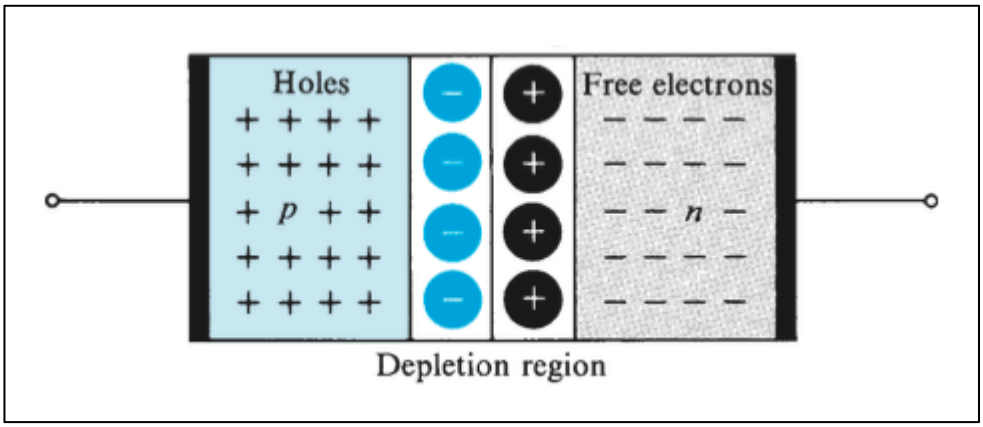


Figura 2. Formação da zona de depleção numa junção PN.[2]

[1] Fonte: retirado de [4].
[2] Fonte: retirado de [4].

Ao ser aplicada uma tensão superior a 0,7 V aos terminais da junção, com a região P positiva em relação à região N, as cargas elétricas atravessam a junção, pois o potencial aplicado externamente atrai os eletrões da região N e repele as lacunas da região P, estreitando a zona de depleção. Ao se aplicar uma tensão com a polaridade inversa, existe o alargamento da zona de depleção que inibe o trânsito das cargas elétricas. Este é o princípio de funcionamento de uma junção PN, que é a base do díodo, um componente eletrónico que é bom condutor de corrente elétrica quando diretamente polarizado, e mau condutor quando inversamente polarizado.

Como referido, um transístor bipolar consiste na combinação de duas junções PN, sendo tipo NPN quando combinadas de forma à região P ser comum, e PNP quando é comum a região N, conforme a Figura 3.

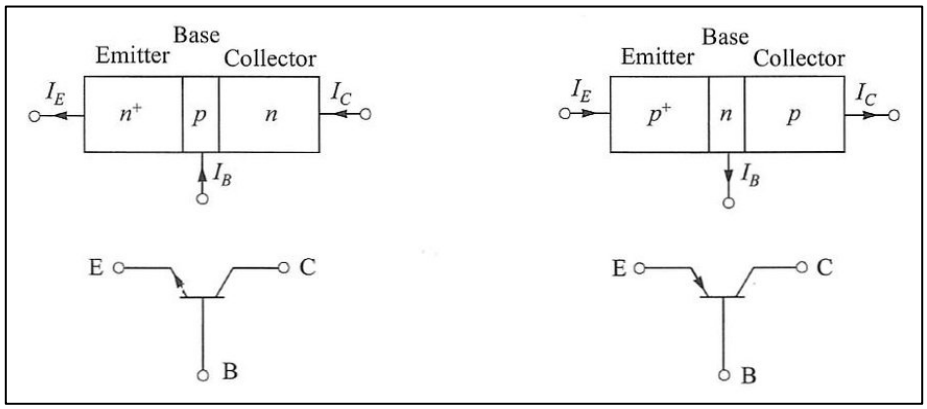


Figura 3. Construção e simbologia de transístores bipolares NPN e PNP.[3]

A seguinte explicação do funcionamento do transístor bipolar supõe um transistor bipolar NPN polarizado na zona ativa, isto é, que a junção base-emissor (BE) se encontra diretamente polarizada por uma tensão V_{BE}, e a junção base-coletor (BC) está inversamente polarizada por uma tensão V_{CB}, conforme a Figura 4.

[3] Fonte: retirado de https://www.researchgate.net/figure/Symbols-and-nomenclatures-of-a-n-p-n-transistor-and-b-p-n-p-transistor-I-E-I-B-and_fig3_259283223

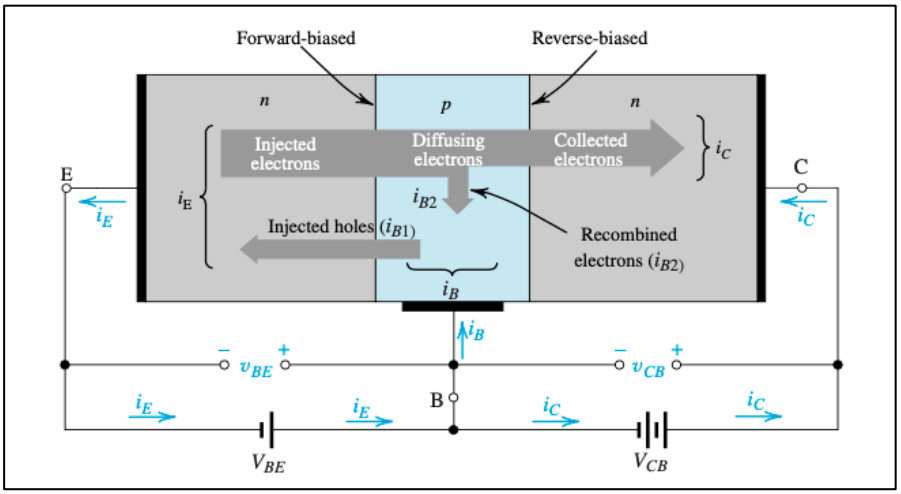


Figura 4. Polarização de transístor bipolar NPN na zona ativa.[4]

Ao estar diretamente polarizada, a zona de depleção da junção BE estreita, dando origem à passagem de eletrões e lacunas entre o emissor e a base do transístor. Contudo, o emissor N é fortemente dopado enquanto a base P é levemente dopada, o que significa que apenas alguns eletrões provenientes do emissor se recombinam com as lacunas existentes na base, dando origem à corrente de base do transístor (I_B). Os restantes eletrões, que são a maioria, difundem-se pela base, que é estreita, e facilmente chegam à zona de depleção da junção BC, onde o campo elétrico criado pela tensão V_{CB} os atrai até ao coletor, dando origem à corrente de coletor (I_C). A corrente de emissor (I_E) é a soma de I_B e I_C, já que as correntes de entrada e de saída do transístor têm que ser iguais.

Os transístores bipolares PNP funcionam da mesma forma, com a diferença de que as tensões de polarização são invertidas e as correntes têm sentidos opostos, sendo os primários transportadores de carga as lacunas, em vez dos eletrões.

[4] Fonte: retirado de [4].

Os transístores MOSFET são outro tipo de transístores que, embora o seu princípio de funcionamento seja diferente dos bipolares, o seu funcionamento como bloco é muito semelhante. A explicação que se segue refere-se a um transistor MOSFET de canal N, também referido como NMOS.

A Figura 5 demonstra a estrutura física de um transistor NMOS. Este tipo de transístor tem quatro terminais e consiste num substrato de semicondutor do tipo P, no qual são integradas duas regiões de semicondutor do tipo N, que representam dois terminais do componente, a fonte (*source*, S) e o dreno (*drain*, D). Sobre a superfície do substrato, entre as regiões da fonte e do dreno, é colocada uma camada de dióxido de silício que é um isolador elétrico. Sobre esta camada é inserido um contacto metálico constituindo outro terminal, a porta (*gate*, G). Embora exista ainda um quarto terminal, o corpo (*body*, B), cuja ligação é feita ao substrato do NMOS, normalmente não é considerado para efeitos de polarização do transístor, e é ligado diretamente à fonte do NMOS. A explicação do funcionamento do transístor NMOS que se segue pressupõe esta ligação, e refere-se ao transístor como um componente de três terminais, sendo estes a fonte, a porta e o dreno.

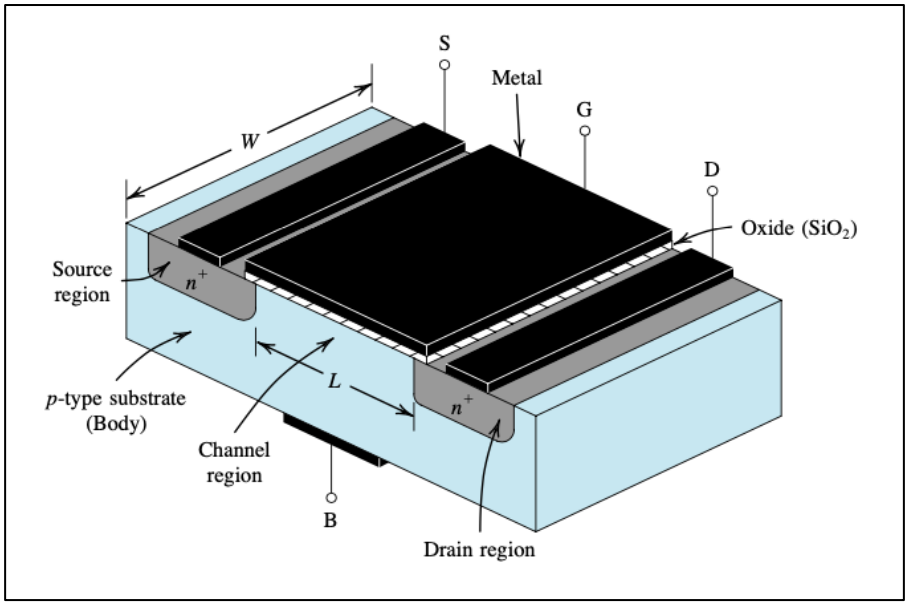


Figura 5. Estrutura física de transístor MOSFET de canal N.[5]

[5] Fonte: retirado de [4].

A estrutura do transístor cria duas junções PN, entre o substrato e a fonte e o substrato e o dreno. Quando não existe tensão aplicada na porta do transístor, estas junções funcionam como dois díodos em série em direções opostas, o que inibe a condução de corrente entre o dreno e a fonte quando é aplicada uma tensão entre estes terminais (*drain-to-source*, V_{DS}).

Assumindo uma ligação à massa na fonte do NMOS, ao ser aplicada uma tensão positiva na porta, tensão porta-fonte (*gate-to-source*, V_{GS}), as lacunas presentes no substrato são repelidas da zona junto à porta do transístor, ao mesmo tempo que são atraídos para esta zona os eletrões livres. Esta concentração de transportadores de carga entre a fonte e o dreno cria um canal entre estes dois terminais, que permite a condução de corrente elétrica. Não sendo aplicada uma tensão V_{DS}, conforme demonstra a Figura 6, o canal induzido tem uma espessura uniforme ao longo da sua extensão.

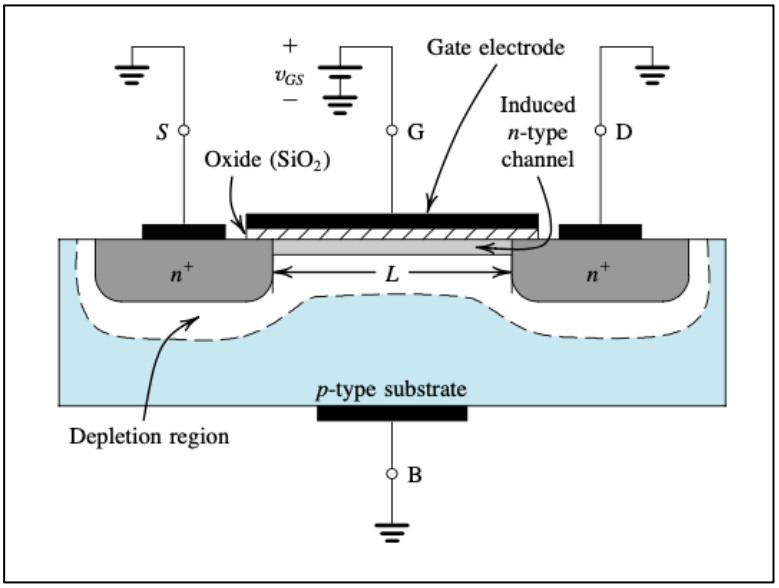


Figura 6. Canal criado entre a fonte e dreno de transístor NMOS.[6]

O canal induzido é do tipo N, o que identifica o transístor MOSFET como sendo de canal N. Os transístores MOSFET de canal P, ou PMOS, apresentam um

[6] Fonte: retirado de [4].

substrato do tipo N e as regiões da fonte e do dreno do tipo P, que dão origem a um canal do tipo P quando polarizado.

Ao valor de tensão V_{GS} necessário para que exista a indução do canal condutor, e por sua vez condução de corrente entre a fonte e o dreno, dá-se o nome de tensão de *threshold* (V_t), e é positivo para transístores NMOS e negativo para PMOS. Este parâmetro é análogo à tensão necessária para polarizar diretamente a junção base-emissor nos transístores bipolares.

Como bloco funcional, os transístores bipolares e MOSFET podem ser tratados de forma semelhante, considerando os terminais coletor, base e emissor análogos aos terminais dreno, porta e fonte, respetivamente. A principal diferença entre estes tipos de transístores reside no facto de a corrente de porta dos transístores MOSFET ser virtualmente nula, comparativamente com a corrente de base dos transístores bipolares. Esta característica torna os transístores MOSFET ideais para implementações em lógica digital, que é onde são amplamente utilizados.

2.2 Curvas Características de um Transístor Bipolar

Uma das montagens mais comuns do transístor bipolar é com o emissor comum, montagem representada na Figura 7.

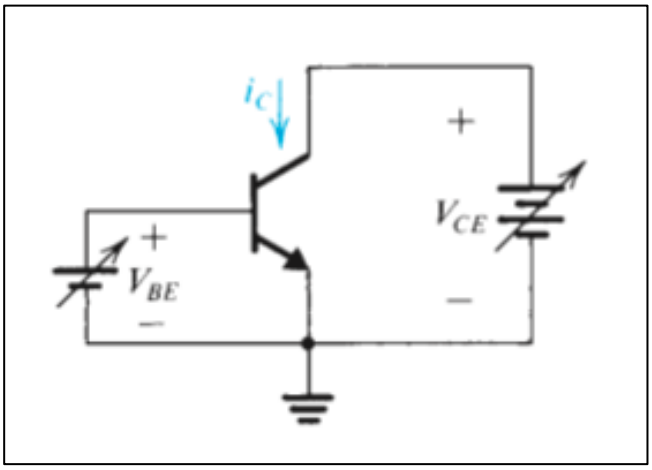


Figura 7. Montagem em emissor comum de um transístor bipolar NPN.[7]

[7] Fonte: retirado de [4].

Nesta configuração, estando o transístor NPN polarizado na zona ativa, a sua junção BE funciona como um díodo, e a corrente de emissor segue uma relação exponencial dada pela seguinte equação:

(1) $I_E = I_S\, e^{V_{BE}/V_T}$.

onde I_s representa a corrente inversa de saturação e V_T é a tensão térmica, uma constante dependente da temperatura da junção.

Dado que as correntes de entrada e de saída do transístor têm de ser iguais tem-se a equação 2:

(2) $I_E = I_B + I_C$.

Como já referido, a maioria dos eletrões emitidos pelo emissor chegam ao coletor, com a exceção de uma pequena percentagem que dá origem à corrente de base. Pode-se então inferir que a corrente de base é uma fração da corrente de coletor, conforme a seguinte equação:

(3) $I_B = \frac{I_C}{\beta}$.

onde β representa o ganho de corrente do transístor, que em transístores bipolares tem valores típicos entre 50 e 200 [4]. Com as equações 2 e 3 obtém-se a equação 4:

(4) $I_E = \frac{\beta+1}{\beta} I_C$.

que para valores típicos de β, verifica que a corrente de emissor é aproximadamente igual à corrente de coletor. A equação 4 pode ser reescrita da seguinte forma:

(5) $I_C = \alpha I_E$.

onde α representa o rendimento do transístor e tem a seguinte relação com β:

(6) $\beta = \frac{\alpha}{1-\alpha}$.

O rendimento do transístor depende de vários fatores de fabrico. A equação 6 demonstra que pequenas variações de α implicam grandes variações de β, assim, transístores do mesmo tipo, com rendimentos semelhantes, podem apresentar valores de ganho muito diferentes.

2.2.1 Curva de Entrada

A curva de entrada é a representação da corrente de base em função da tensão base-emissor, ou seja, $I_B(V_{BE})$, com V_{CE} constante.

Com base nas equações 1 e 3, obtém-se a seguinte relação exponencial que caracteriza a corrente de base:

(7) $I_B \approx \frac{1}{\beta} \times I_S \, e^{V_{BE}/V_T}$.

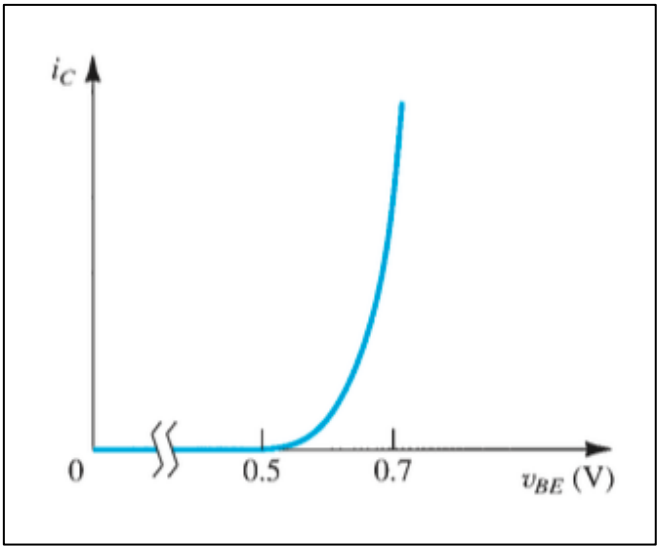


Figura 8. Representação gráfica da curva de entrada de um transístor bipolar.[8]

[8] Fonte: retirado de [4].

A curva característica representada na Figura 8 demonstra o funcionamento da junção BE, onde a condução de corrente depende da tensão aplicada aos seus terminais. Com o aumentar da tensão, a zona de depleção estreita, o que aumenta a corrente de coletor. A característica de entrada não varia significativamente com V_{CE}, daí ser uma curva única.

2.2.2 Curva de Transferência

A curva de transferência é a representação da corrente de coletor em função da corrente de base, ou seja, $I_C(I_B)$, com V_{CE} constante. Com base na equação 3, obtém-se a seguinte relação proporcional:

(8) $I_C = \beta I_B$.

A representação gráfica desta curva característica é uma reta, cujo declive indica o valor de β. Por não variar significativamente com V_{CE}, a característica de transferência também é única.

2.2.3 Curva de Saída

A curva de saída é a representação da corrente de coletor em função da tensão coletor-emissor, ou seja, $I_C(V_{CE})$, para um valor de V_{BE} fixo. Como foi demonstrado, I_B depende de V_{BE} e I_C depende de I_B, assim, para diferentes valores de V_{BE} têm-se diferentes curvas de saída, que em conjunto representam a família de curvas de saída do transístor, conforme a representação gráfica da Figura 9.

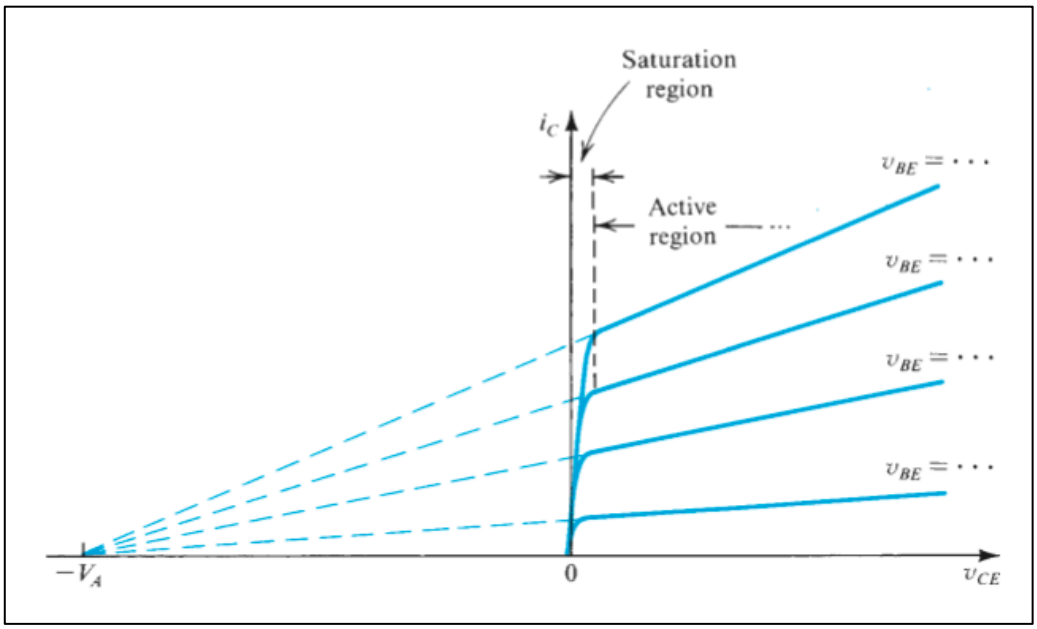


Figura 9. Representação gráfica da família de curvas de saída.[9]

Seria expectável que a corrente de coletor não dependesse de V_{CE}, visto que I_C depende somente da quantidade de eletrões emitidos pelo emissor e da recombinação dos mesmos com as lacunas existentes na base do transístor, o que daria origem a curvas horizontais na representação gráfica. Contudo, o aumentar da tensão coletor-emissor faz aumentar a zona de depleção da junção BC, por estar inversamente polarizada. Este alargamento da zona de depleção resulta no estreitamento da base e na redução do número de lacunas disponíveis, o que implica que existe um maior número de eletrões provenientes do emissor a chegar ao coletor, fazendo assim aumentar I_C.

Este fenómeno é conhecido por "efeito de Early" e leva a que as curvas convirjam para um ponto comum, representado no gráfico da Figura 9 por V_A. O parâmetro V_A representa a "tensão de Early" e é uma característica própria de cada transístor.

[9] Fonte: retirado de [4].

2.3 Curvas Características de um Transístor MOSFET

Como previamente referido, uma das principais características dos transístores MOSFET é o facto de a corrente de porta ser praticamente nula, pelo que não é lógico considerar as suas características de entrada e de transferência.

Os próximos parágrafos visam explicar a dinâmica por detrás da curva de saída de transístores MOSFET, ou seja, as curvas obtidas para a corrente de dreno em função da tensão dreno-fonte para tensões porta-fonte fixas, $I_D(V_{DS}) \mid_{V_{GS}}$.

Assumindo um transístor MOSFET de canal N polarizado por uma tensão V_{GS} de valor superior à sua tensão de *threshold*, é induzido o canal entre a fonte e o dreno do transístor. A diferença entre V_{GS} e V_t, ou seja, o excesso de tensão necessária para induzir o canal é referida como tensão de *overdrive* (V_{OV}), conforme a equação 9:

$$V_{GS} = V_t + V_{OV} \tag{9}$$

Ao ser aplicada uma tensão V_{DS}, a tensão entre a porta e os pontos ao longo da extensão do canal deixa de ser constante, sendo o seu valor máximo junto à fonte e mínimo junto ao dreno. A tensão junto à fonte é simplesmente V_{GS}, mas como existe diferença de potencial entre os extremos do canal, imposta por V_{DS}, junto ao dreno, o valor da tensão é dado pela diferença entre estas tensões, como demonstra a equação 10:

$$V_{GD} = V_{GS} - V_{DS} = V_t + V_{OV} - V_{DS} \tag{10}$$

Visto que a espessura do canal depende do valor de V_{GS}, a sua espessura deixa de ser uniforme ao longo da sua extensão, e apresenta um perfil como o demonstrado na Figura 10.

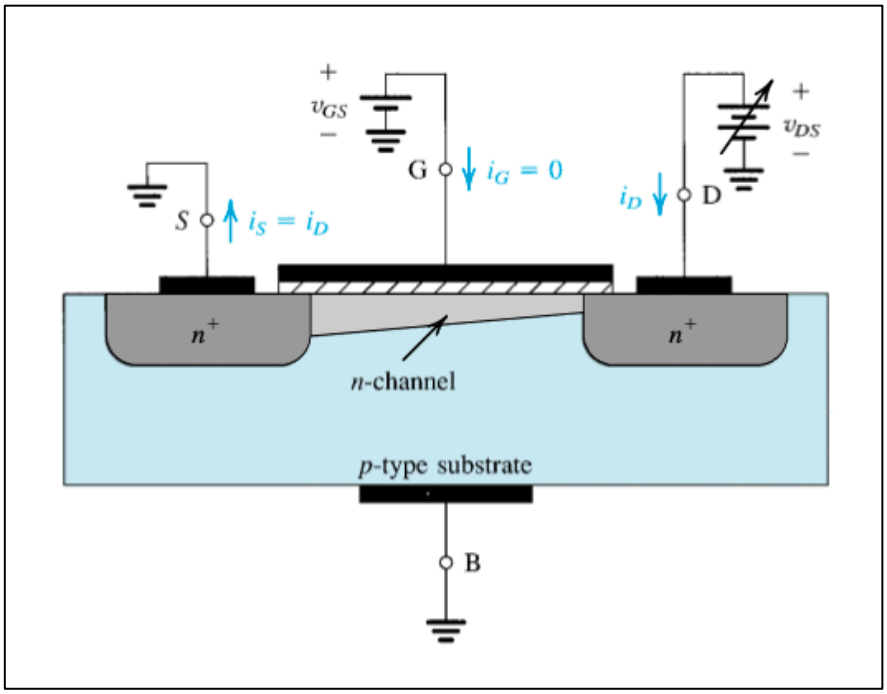


Figura 10. Perfil do canal induzido devido à tensão dreno-fonte.[10]

Continuando a aumentar V_{DS}, o canal vai estreitando cada vez mais junto ao dreno do transístor. Na situação limite, em que o valor da tensão V_{DS} iguala V_{OV}, ou seja, a tensão V_{GD} é igual a V_t, implica que não existe diferença de potencial suficiente entre a porta e o dreno para existir canal, sendo a sua espessura mínima junto ao dreno. A partir deste ponto, em que o canal se encontra em *pinch-off*, mesmo aumentando o valor de V_{DS}, a espessura do canal não é afetada. Nesta situação, continua a existir corrente entre a fonte e o dreno, pois os eletrões provenientes da fonte são acelerados pelo canal e atravessam a zona de depleção criada, até ao dreno.

Esta forma de funcionamento dos transístores MOSFET traduz-se na curva característica de saída representada na Figura 11.

[10] Fonte: retirado de [4].

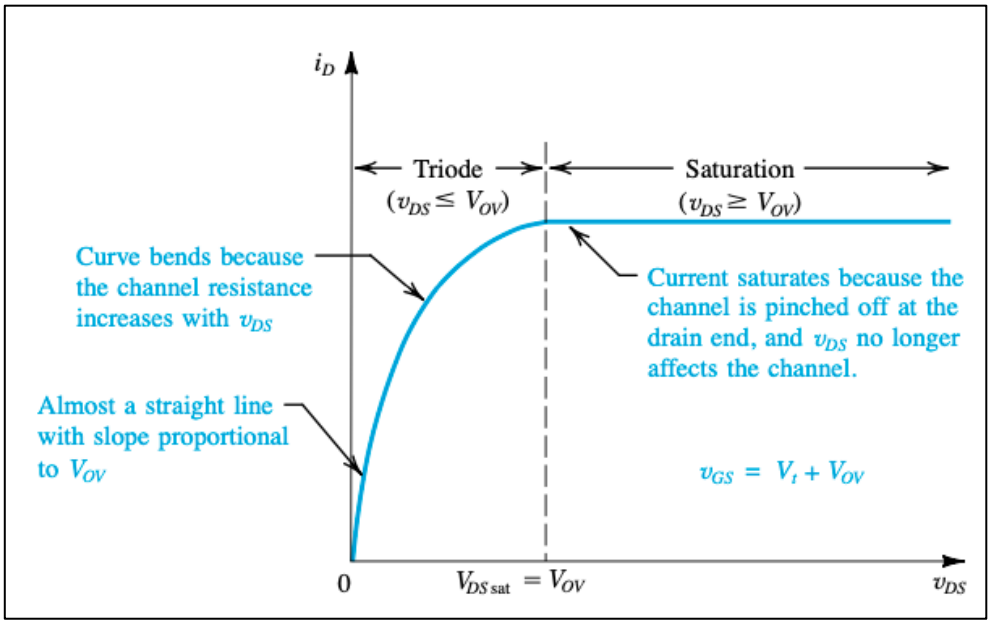


Figura 11. Curva de saída teórica de transístores MOSFET de canal N.[11]

Para valores de V_{DS} baixos, a corrente de dreno (I_D) é proporcional à tensão de *overdrive*. Para valores de V_{DS} maiores, o estreitamento do canal faz com que a resistência do mesmo aumente de forma proporcional, assim, a resistência do canal é proporcional a V_{DS}. O aumento da resistência do canal faz diminuir I_D, explicando a curvatura da representação gráfica. Quando V_{DS} iguala V_{OV}, o transístor entra em saturação, o que explica a horizontalidade da curva característica nesta região. A valor da tensão de saturação ($V_{DS\,sat}$) é dado pela seguinte equação:

(11) $V_{DS\,sat} = V_{OV} = V_{GS} - V_t$

[11] Fonte: retirado de [4].

3 Desenvolvimento

Como já referido, pretendeu-se com este trabalho desenvolver um traçador de curvas de baixo custo com interface gráfica disponibilizada via aplicação móvel para dispositivos Android. Neste capítulo é abordado o processo de desenvolvimento do projeto nas suas várias vertentes. A secção 3.1 refere-se ao sistema desenvolvido como um todo, evidenciando como as diferentes partes se relacionam entre si. Na secção 3.2 aborda-se o hardware desenvolvido, desde o circuito de condicionamento de sinal até ao protótipo final. Finalmente, o software desenvolvido é abordado na secção 3.3.

3.1 Arquitetura Geral

Neste trabalho adotou-se a arquitetura apresentada na Figura 12. Esta estrutura permite segmentar o projeto nas suas diferentes vertentes: desenho do circuito de condicionamento de sinal, desenho do circuito de alimentação, programação do microcontrolador e programação do dispositivo móvel.

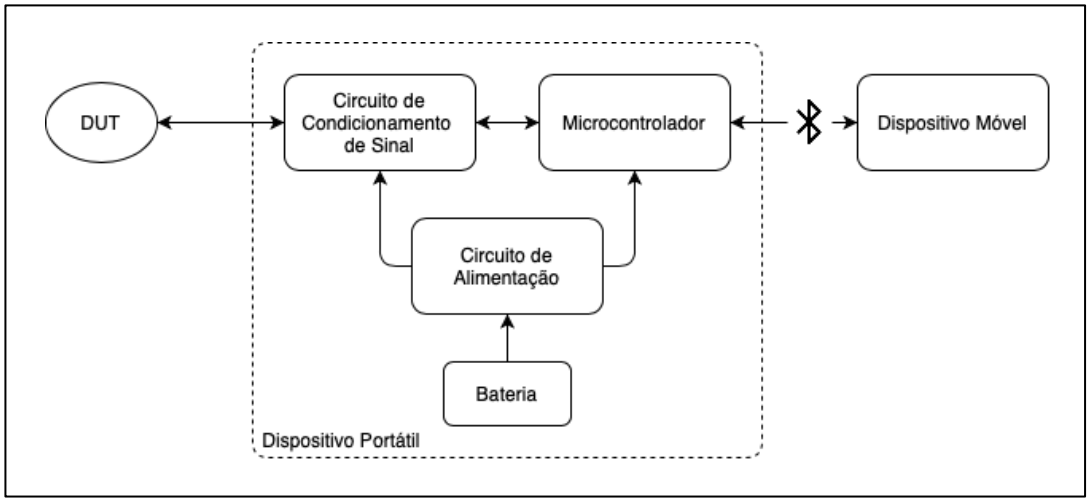


Figura 12. Diagrama de blocos da arquitetura geral do sistema de medida.

O DUT (*device under test*) é o transístor cujas curvas características se pretendem extrair. O microcontrolador, constituído por um Arduino Nano e dois conversores digital-analógico (*digital-to-analog converter*, DAC), tem por missão estimular o transístor, posicionando-o num determinado ponto de funcionamento (PF), e medir os valores de tensão e corrente aos seus terminais, a partir dos quais são desenhadas as curvas características do transístor. O Arduino utiliza um módulo de comunicação *Bluetooth* HC-05 para comunicar com o dispositivo móvel que trata da interface com o utilizador.

Entre o DUT e o Arduino insere-se um circuito de condicionamento de sinal para polarizar o DUT e para converter as correntes em tensões e as tensões negativas em positivas, dado que o Arduino não suporta a medição direta de correntes elétricas nem de tensões negativas.

O circuito de alimentação fornece as tensões necessárias para o circuito de condicionamento de sinal e para o Arduino a partir de uma bateria de 9 V.

A bateria, o circuito de alimentação, o circuito de condicionamento de sinal, e o Arduino estão integrados numa placa de circuito impresso única, representada pela linha a tracejado no diagrama de blocos da Figura 12, e que constitui o referido "dispositivo portátil".

O dispositivo móvel serve como interface gráfica com o utilizador. Foi desenvolvida uma aplicação que permite o controlo das funcionalidades do Arduino e a visualização e exportação dos dados dos ensaios realizados.

O trabalho desenvolvido em cada um destes blocos base é descrito mais detalhadamente nas próximas secções deste capítulo.

3.2 Hardware

Esta secção é dedicada ao hardware desenvolvido no âmbito desta dissertação. Está dividido em quatro partes que refletem o processo de desenvolvimento do circuito de condicionamento de sinal, do circuito de alimentação, da placa de circuito impresso e a montagem do protótipo final.

3.2.1 Circuito de Condicionamento de Sinal

O circuito de condicionamento de sinal adotado encontra-se representado na Figura 13. Este circuito permite a extração das curvas características de transístores bipolares NPN e PNP, e transístores NMOS e PMOS, assumindo a correspondência entre terminais previamente referida. Para implementar o circuito utilizou-se o integrado LM324N, que disponibiliza quatro amplificadores operacionais (AMPOP) e resistências com tolerância de 5% e ¼ W. Os conversores digital-analógico são distribuídos pela Adafruit e têm a referência MCP4725.

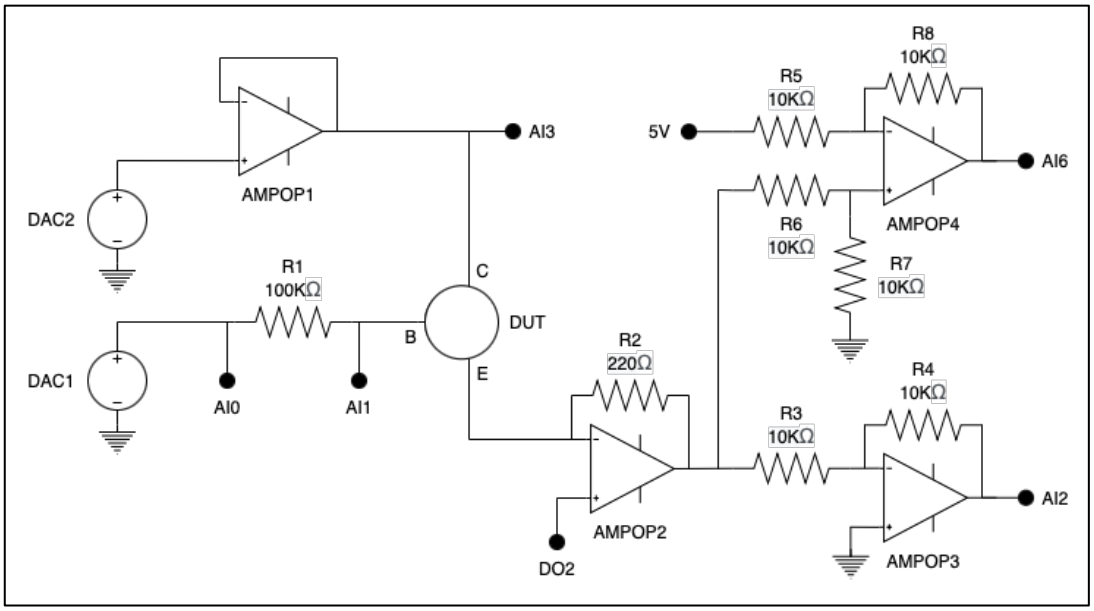


Figura 13. Circuito de condicionamento de sinal.

Os terminais deste circuito identificados como AI (*analog input*) são ligados aos respetivos terminais analógicos do Arduino, e permitem a medição de tensões para o cálculo das grandezas das curvas características. O terminal DO2 (*digital output*) é ligado ao respetivo terminal digital do Arduino e permite fornecer a este terminal tensões de 5 V ou 0 V (ligação à massa).

Os DACs, representados pelas fontes de tensão DAC1 e DAC2, são configurados pelo Arduino via protocolo I2C [5], e permitem a injeção de sinal no circuito para a polarização do DUT. Embora o Arduino seja capaz de injetar sinais analógicos diretamente no circuito através de saídas PWM (*pulse width modulation*),

optou-se pela utilização de DACs por ser mais simples e permitir a geração de sinais com maior qualidade. As saídas PWM do Arduino estão limitadas por software a uma resolução de 8 bits [6], ou seja, 256 valores diferentes, o que é limitativo já que os conversores analógico-digital para medição de tensões são de 10 bits. Os DACs utilizados permitem uma resolução de 12 bits, correspondente a 4096 valores diferentes, o que permite maximizar a resolução das leituras. Além disso, as saídas PWM necessitariam de filtros passa-baixo para alisar os sinais em tensão, o que aumentaria a complexidade do circuito de condicionamento.

O circuito para a polarização dos transístores bipolares NPN e NMOS encontra-se representado na Figura 14 e pressupõe DO1 = 0 V (*LOW*).

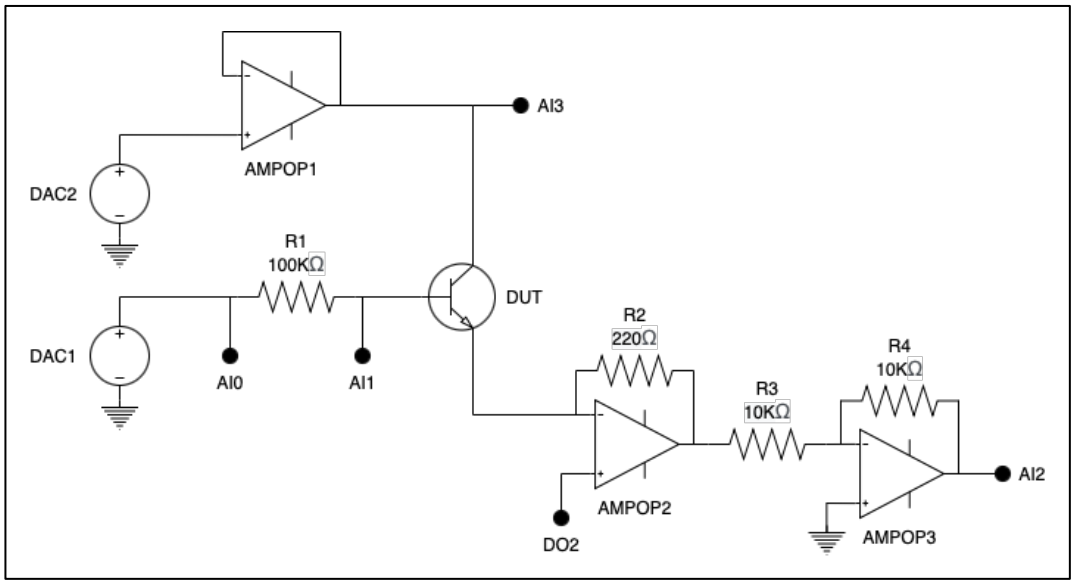


Figura 14. Circuito de polarização de transístores BJT NPN e NMOS.

O DAC1 polariza diretamente a junção base-emissor do transístor fazendo-o conduzir mais ou menos consoante o valor da tensão aplicada. A resistência R1 serve para amostrar a corrente de base. O DAC2 serve para impor uma tensão V_{CE} no transístor desde que que DO2 = 0 V (*LOW*), como se pressupõe. Colocando o terminal DO2 à massa (0 V) consegue-se criar uma massa virtual no emissor do transístor e, ao mesmo tempo, converter a corrente de emissor em tensão através da montagem implementada com o AMPOP2. A tensão de saída do AMPOP2, que é negativa, é invertida pela montagem inversora de ganho unitário constituída pelo AMPOP3 e

entregue ao Arduino. O circuito suporta correntes de emissor entre 0 e 23 mA, que dão origem a uma tensão entre 0 e 5 V em AI2, que é precisamente a gama de operação do Arduino. O AMPOP1, que implementa uma montagem de ganho unitário, funciona como *buffer*, de forma a não sobrecarregar o DAC1 e aumentar a capacidade de fornecimento de corrente ao DUT.

Com este circuito calcula-se a corrente de base do DUT através da equação 3 utilizando as medições nos terminais AI0 e AI1. As tensões base-emissor e coletor-emissor são medidas diretamente nos terminais AI1 e AI3, respetivamente.

(3) $I_B = \frac{AI0 - AI1}{R_1}$.

A corrente de emissor é dada pela relação do conversor corrente-tensão implementado pelo AMPOP2:

(4) $I_E = \frac{AI2}{R_2}$.

A corrente de coletor é dada pela diferença entre a corrente de emissor e a corrente de base:

(5) $I_C = I_E - I_B$.

O circuito para polarização de transístores PNP e PMOS encontra-se representado na Figura 15 e pressupõe DO2 = +5 V (*HIGH*).

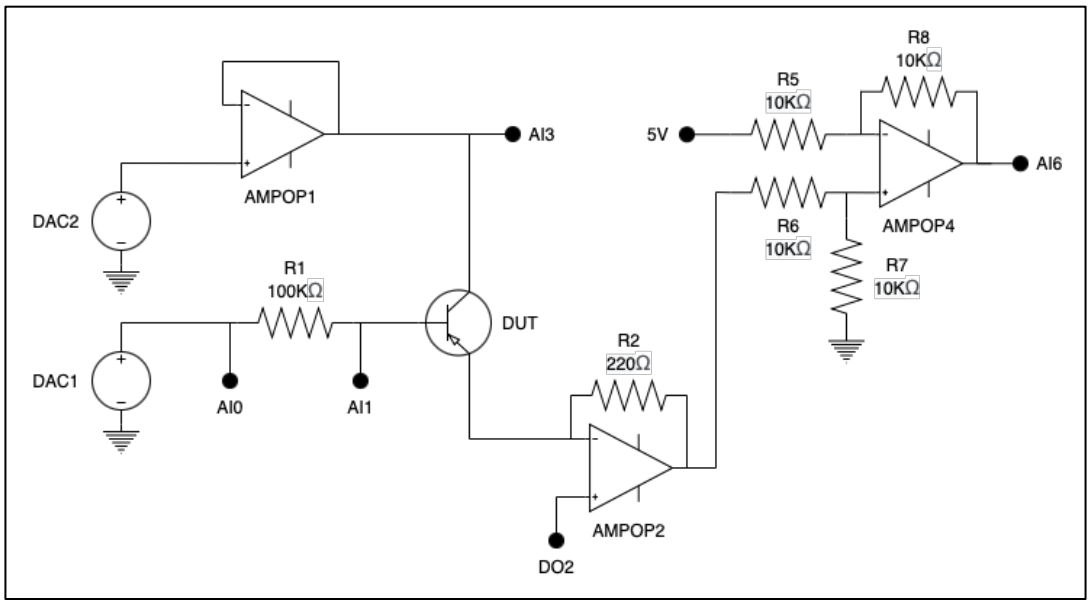


Figura 15. Circuito de polarização de transístores BJT PNP e PMOS.

Ao contrário do que acontece com os transístores bipolares NPN e NMOS, as tensões criadas no terminal de saída do AMPOP2 são positivas, daí não ser necessário uma montagem inversora. Contudo, colocar o terminal DO2 a *HIGH* implica adicionar um *offset* de +5 V na tensão de saída do AMPOP2, dando origem a tensões entre +5 e +10 V que não são admitidas pelo Arduino. Para resolver este problema, acrescentou-se uma montagem diferencial de ganho unitário baseada no AMPOP4 que neutraliza (subtrai) este *offset*. Com este arranjo, correntes de emissor entre 0 e 23 mA dão origem a tensões entre 0 e +5 V.

O cálculo das grandezas das curvas características é feito da mesma forma que para os transístores bipolares NPN e NMOS, com a exceção da corrente de emissor que é calculada com a medição do terminal AI6, conforme a equação 6.

(6) $$I_E = \frac{AI6}{R_2} .$$

3.2.2 Circuito de Alimentação

O esquema de alimentação elétrica encontra-se representado na Figura 16.

A alimentação necessária a todos os componentes é fornecida por um conversor DC-DC (*direct current*) DD1718PA fabricado pela Eletechsup. Este conversor é alimentado por uma pilha de 9 V recarregável e disponibiliza tensões simétricas de ±15 V, utilizadas para alimentar os AMPOPs do circuito de condicionamento de sinal. O Arduino, que admite tensões de alimentação entre +7 e +12 V, é alimentado diretamente pelos +9 V fornecidos pela bateria. Os DACs e módulo *Bluetooth* HC-05, ambos com tensões de operação de +5 V, são alimentados pelo Arduino, que disponibiliza um terminal próprio para este efeito. A massa é comum a todo o circuito.

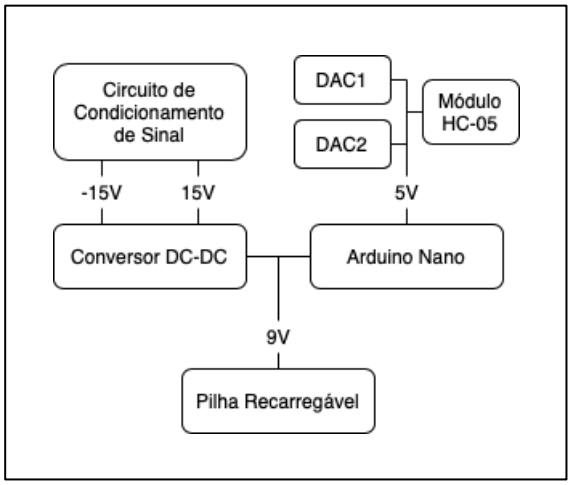


Figura 16. Diagrama do circuito de alimentação.

3.2.3 Placa de Circuito Impresso

O desenho da placa de circuito impresso (*printed circuit board*, PCB) foi feito com recurso ao software Eagle da AutoDesk. Esta plataforma permite o desenho da PCB com base no esquema do circuito eletrónico. Nesse sentido, desenhou-se o circuito de condicionamento de sinal conforme a Figura 17.

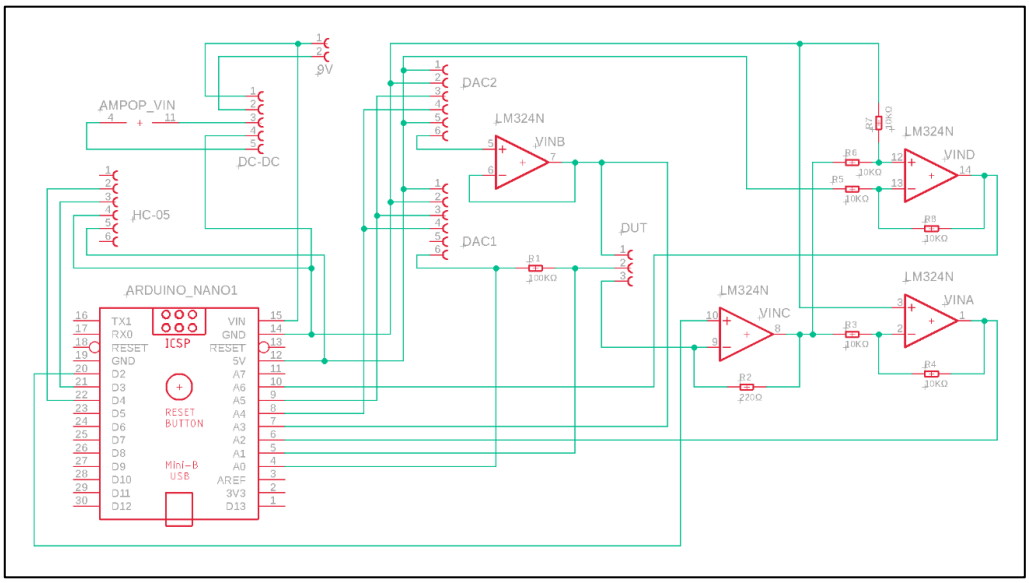


Figura 17. Esquema do circuito de condicionamento de sinal.

Com base neste esquema gerou-se uma PCB base, sobre a qual se trabalhou com o objetivo de organizar os componentes e as pistas de forma eficiente. Utilizou-se uma placa de face dupla e pistas de 0,5 mm de forma a simplificar e reduzir o custo de fabrico. O desenho final da PCB é demonstrado na Figura 18 e Figura 19.

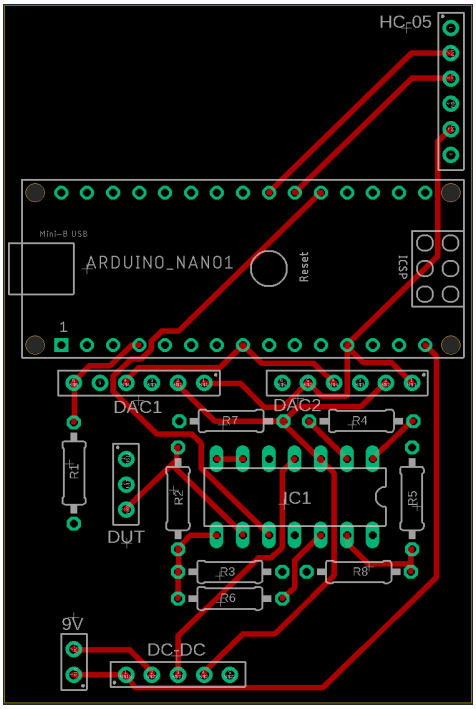


Figura 18. Face superior da PCB.

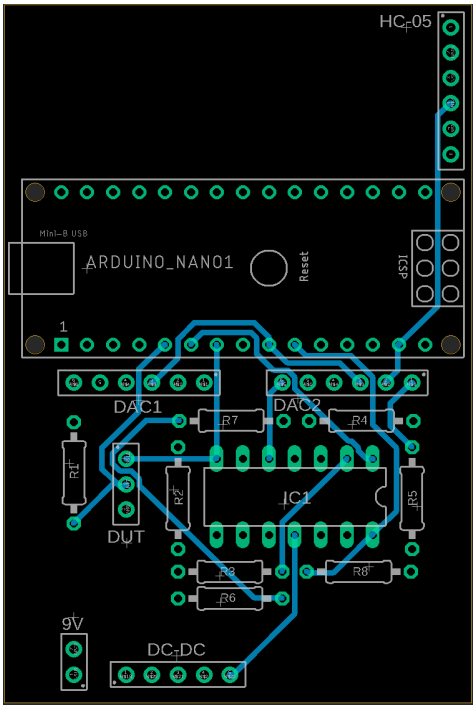


Figura 19. Face inferior da PCB.

As máscaras de ambas as faces da PCB em escala 1:1 encontram-se em apêndice (apêndice A), juntamente com a lista e disposição dos componentes.

3.2.4 Protótipo Final

O protótipo final foi construído agregando todas as partes referidas nas secções anteriores. Na placa de circuito impresso soldaram-se suportes para conectar o Arduino Nano, o módulo *Bluetooth* HC-05, os DACs, o conversor DC-DC de 15 V e o circuito integrado LM324. Soldaram-se diretamente na placa as resistências e as ligações do suporte da bateria e do conector onde se coloca o DUT. Criou-se o conector demonstrado na Figura 20, de forma a simplificar a colocação do DUT no dispositivo, visto que os transístores disponíveis comercialmente apresentam diferentes combinações dos seus terminais.

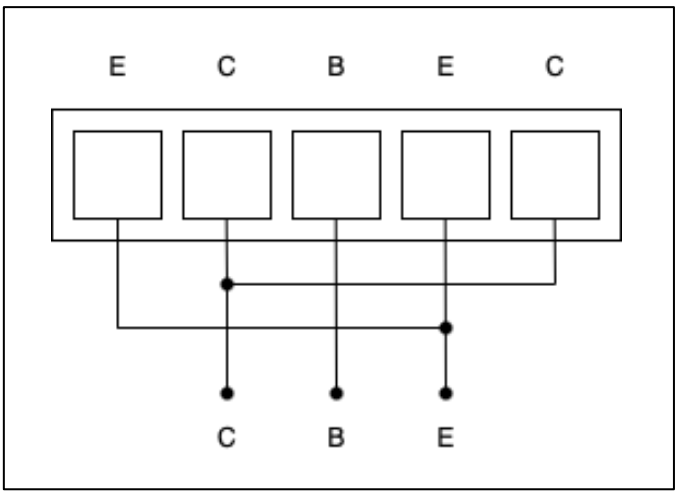


Figura 20. Conector do DUT.

A Figura 21 demonstra a placa de circuito impresso finalizada. Fizeram-se quatro furos nas extremidades para simplificar a fixação da placa com o uso de parafusos de plástico, com o objetivo de não danificar a placa.

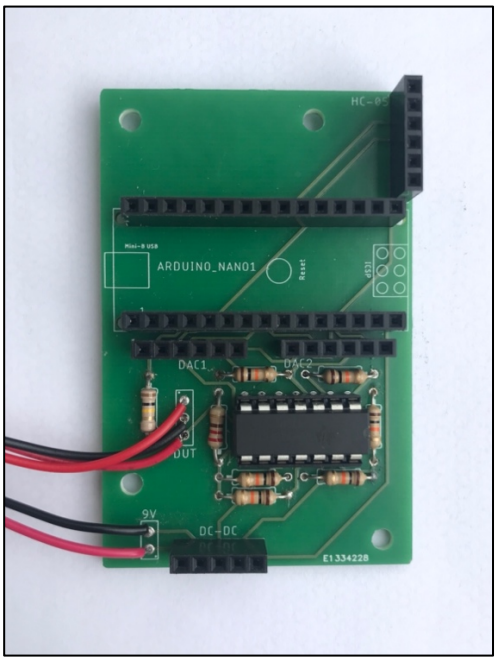


Figura 21. Placa de circuito impresso finalizada.

O suporte da bateria e os encaixes dos parafusos da PCB foram fixados no interior de uma caixa utilizando resina epóxi, que garante a boa fixação destes componentes. A Figura 22 mostra a fixação da placa de circuito impresso e do suporte da bateria no interior da caixa.

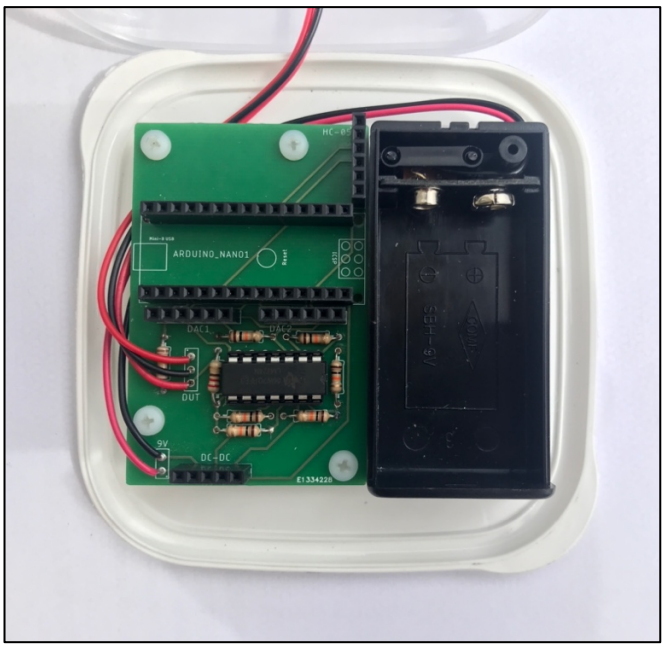


Figura 22. PCB e suporte da bateria fixados na caixa.

O protótipo finalizado encontra-se demonstrado na Figura 23, o interior, e na Figura 24, o exterior. Os terminais do conector do DUT encontram-se identificados, para transístores bipolares e MOSFET.

Figura 23. Protótipo final - interior.

Figura 24. Protótipo final - exterior.

3.3 Software

Esta secção é dedicada ao software desenvolvido no âmbito deste trabalho, estando divido em duas partes. A primeira parte refere-se à programação do microcontrolador Arduino, utilizando o ambiente Arduino IDE, e a segunda parte diz respeito ao desenvolvimento da aplicação para dispositivos móveis Android, com recurso ao ambiente Android Studio.

3.3.1 Microcontrolador Arduino

A programação do microcontrolador Arduino foi feita com recurso ao ambiente de desenvolvimento Arduino IDE [7]. Esta plataforma suporta a linguagem de programação C++ e é utilizada para a escrita e *upload* de código para o microcontrolador.

Na função "setup()", declaram-se e inicializam-se os canais de comunicação série: a porta série nativa do Arduino para *debug*, e a porta série "BT" através dos terminais D3 (Rx) e D4 (Tx) para comunicação via *Bluetooth* com o dispositivo móvel, através do módulo HC-05 e fazendo uso da biblioteca nativa "SoftwareSerial.h" [8]. Declara-se também os terminais analógicos do Arduino a serem utilizados como *input* para aquisição de dados, e o terminal digital a ser utilizado como *output*.

Dada a forma de funcionamento do ambiente Arduino, implementou-se uma arquitetura de máquina de estados baseada em duas variáveis de controlo que, conforme o seu valor, despoletam determinadas funções. Estas variáveis indicam o tipo de transístor a ser testado e o tipo de curva a ser extraída. O dispositivo, ao ser iniciado, encontra-se num estado de *standby*, verificando continuamente se chegam dados pela porta série *Bluetooth*, dados esses que alteram o valor das variáveis de controlo. Ao serem recebidos dados, o valor das variáveis de controlo é alterado e é iniciada a função "selectFunc()", que utiliza uma estrutura *switch-case* para iniciar a função adequada conforme o valor das variáveis.

Conforme o tipo de transístor e curva característica, é iniciada uma das seguintes funções:

- “curvaEntradaNPN()”;
- “curvaTransferenciaNPN()”;
- “curvaSaidaNPN()”;
- “curvaEntradaPNP()”;
- “curvaTransferenciaPNP()”;
- “curvaSaidaPNP()”;
- “curvaSaidaNMOS()”;
- “curvaSaidaPMOS()”.

Cada uma destas funções extrai a curva característica específica. Todas elas seguem a mesma arquitetura, sendo constituídas por um ciclo *for* (dois no caso das curvas características de saída), que, de forma incremental, em cada iteração polariza o DUT, mede os valores de tensão nos seus terminais, efetua os cálculos das correntes e tensões relevantes para a curva característica, e envia os dados via porta série *Bluetooth* para o dispositivo móvel, conforme o fluxograma da Figura 25. Os valores de tensão são medidos ao chamar a função “measureInputs()”, que adquire múltiplos valores para o mesmo PF e calcula a sua média.

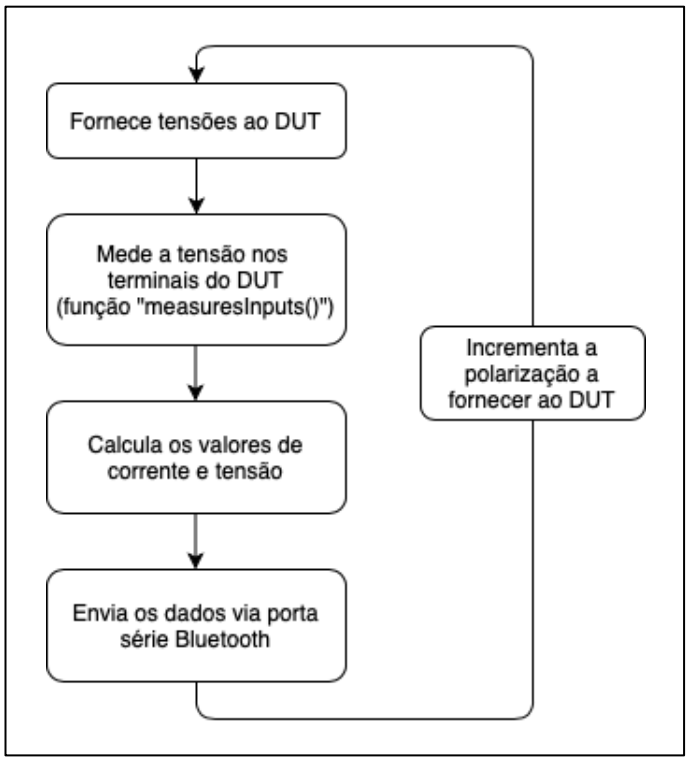


Figura 25. Fluxograma do funcionamento das funções específicas do Arduino.

A configuração de ambos os DACs é feita com recurso à biblioteca "Adafruit_MCP4725.h" [9]. Esta biblioteca, disponibilizada pelo próprio fabricante, permite o controlo da tensão fornecida pelos DAC por via de funções próprias, fazendo uso do protocolo I2C através dos terminais do Arduino específicos para o efeito, A4 (SDA) e A5 (SCL).

O código fonte desenvolvido encontra-se em apêndice (apêndice B).

3.3.2 Aplicação para Dispositivos Android

A programação da aplicação móvel foi feita com recurso à plataforma Android Studio [10], desenvolvida pela Google. Trata-se do ambiente de desenvolvimento integrado mais amplamente utilizado para a programação de dispositivos móveis compatíveis com o sistema operativo Android. Tendo suporte para várias linguagens de programação, optou-se por programar em Java.

Numa fase inicial, foi importante definir a estrutura e os requisitos da aplicação a ser desenvolvida. Consideraram-se relevantes os seguintes requisitos:

- Capacidade para iniciar a ligação *Bluetooth*, incluindo encontrar e emparelhar com novos dispositivos;
- Capacidade para escolha do tipo de curva característica;
- Capacidade para escolha do tipo de transístor;
- Capacidade de visualização gráfica das curvas características;
- Indicação do progresso durante a extração de curvas características;
- Capacidade de exportar os dados para plataformas de armazenamento em *cloud* e via email;
- Inferface gráfica simples e intuitiva.

Quanto à estrutura, com base nos requisitos, concebeu-se o modelo representado pelo fluxograma da Figura 26.

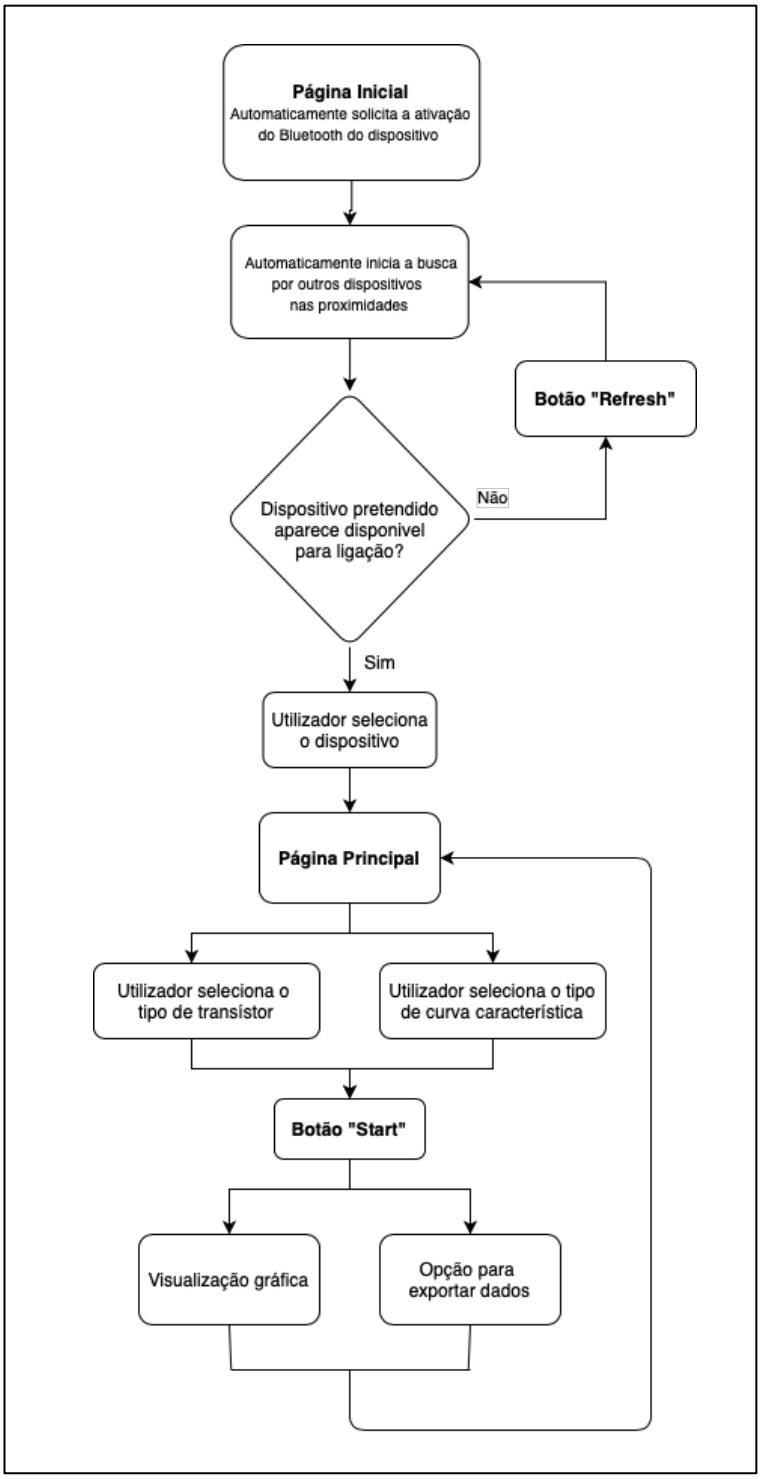


Figura 26. Fluxograma do funcionamento da aplicação móvel.

Ao abrir a aplicação, caso o *Bluetooth* do dispositivo móvel não se encontre ligado, é automaticamente solicitada a ativação do mesmo, conforme a Figura 27. Ao aceitar a ativação do *Bluetooth*, a aplicação começa imediatamente a procurar

outros dispositivos *Bluetooth* disponíveis nas proximidades, disponibilizando os nomes e endereços MAC dos dispositivos encontrados, conforme a Figura 28. Caso o dispositivo pretendido não apareça, o utilizador tem a possibilidade de reiniciar a procura utilizando o botão *refresh.* O utilizador, ao identificar o nome do dispositivo portátil, cujo módulo *Bluetooth* HC-05 foi configurado como sendo "CurveTracer", seleciona o mesmo, iniciando a ligação *Bluetooth* com o dispositivo. Caso seja a primeira ligação a este dispositivo, é também efetuado o processo de emparelhamento (*pairing*).

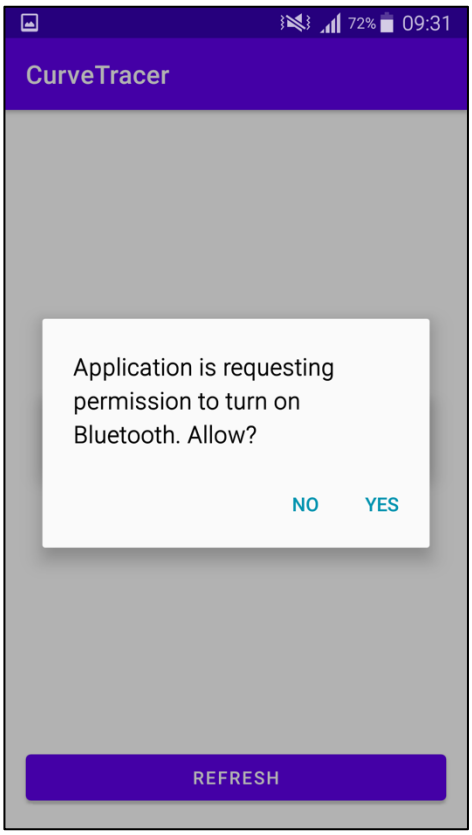


Figura 27. Janela de solicitação para ativação do Bluetooth.

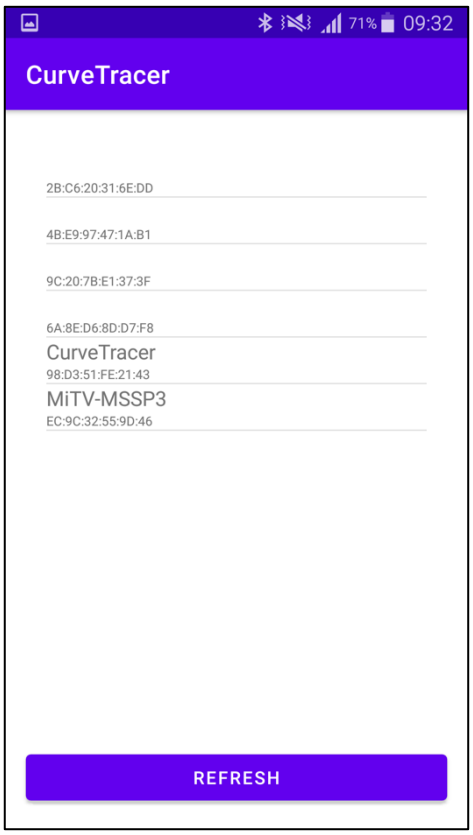


Figura 28. Página inicial da aplicação.

Concluída a ligação com sucesso, a aplicação exibe a sua página principal, representada na Figura 29.

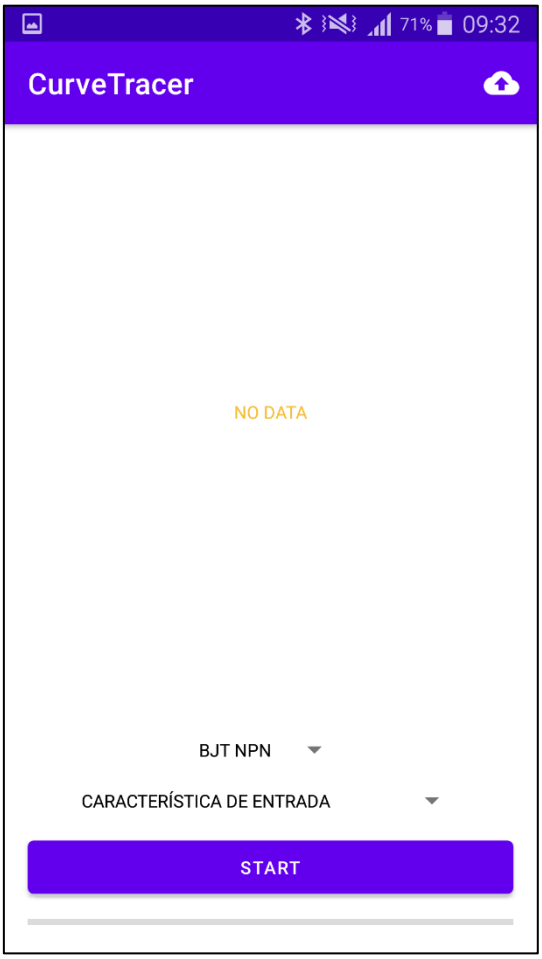


Figura 29. Página principal da aplicação.

Nesta página são disponibilizados dois *spinners*, um para a escolha do tipo de transístor a ser caracterizado, tendo as opções demonstradas na Figura 30, e outro para o tipo de ensaio a realizar, conforme a Figura 31. Como os transístores MOSFET apresentam apenas característica de saída, o tipo de ensaio é automaticamente selecionado quando o utilizador seleciona este tipo de transístor. No canto superior direito é exibido o botão para a exportação dos resultados do ensaio, que nesta fase, devido a ainda não existirem dados a exportar, não é executável. Na parte inferior da página encontra-se o botão de *start*, que inicia o ensaio, e a barra de progresso, que permite a visualização do progresso durante o ensaio.

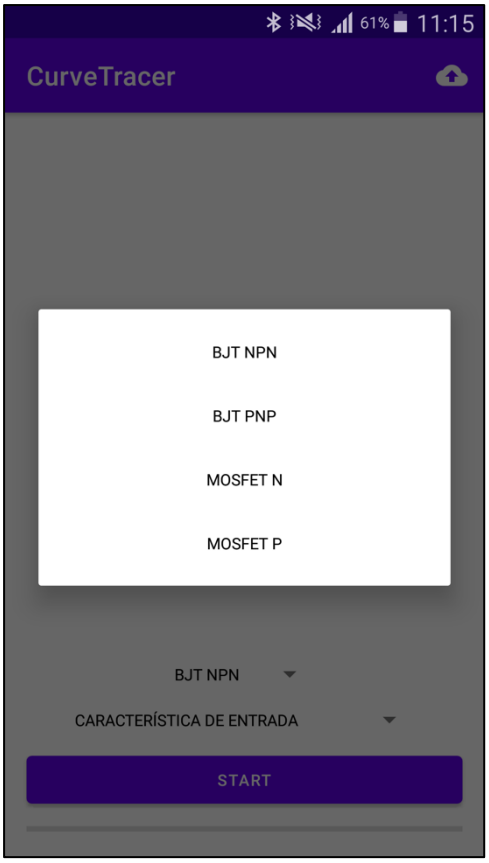


Figura 30. Spinner para seleção do tipo de transístor.

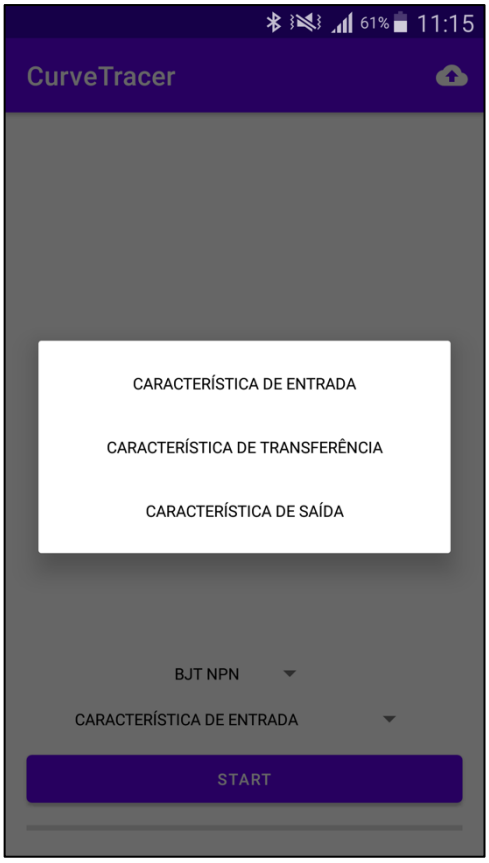


Figura 31. Spinner para seleção do tipo de curva característica.

Terminado o ensaio, os resultados são visualizados graficamente na parte superior da página. Os valores de ganho e de tensão de Early de transístores bipolares são calculados automaticamente, com base nos declives das curvas características, e indicados no canto inferior direito do gráfico, conforme demonstra a Figura 32. Nesta fase, é permitida a exportação dos dados através do botão para o efeito previamente referido, sendo apresentada a janela apresentada na Figura 33. O ficheiro exportado é de formato CSV e as plataformas admitidas para a exportação, sendo de terceiros, dependem do fabricante do dispositivo e das aplicações em si instaladas. O código fonte da aplicação encontra-se em apêndice (apêndice C).

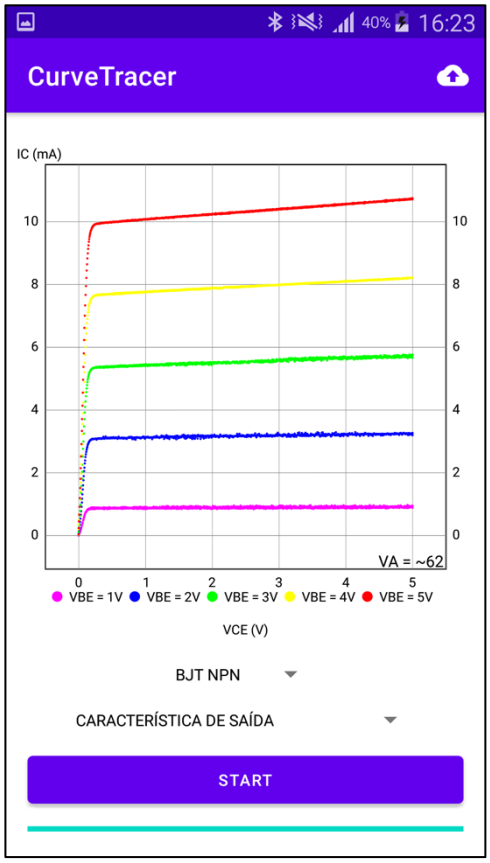


Figura 32. Visualização gráfica dos resultados do ensaio.

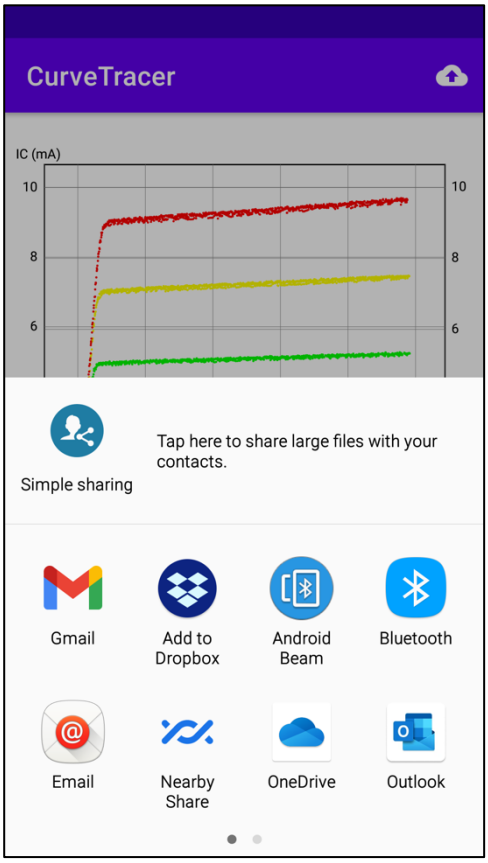


Figura 33. Janela para exportação dos resultados do ensaio.

4 Ensaios Experimentais

O presente capítulo contempla os resultados dos ensaios experimentais realizados com o objetivo de aferir o funcionamento do traçador de curvas. Foram realizados ensaios com transístores bipolares NPN e PNP, e com transístores MOSFET de canal N e canal P.

4.1 Transístores Bipolares NPN

Utilizou-se o transístor 2N2222A para os ensaios experimentais de transístores bipolares NPN.

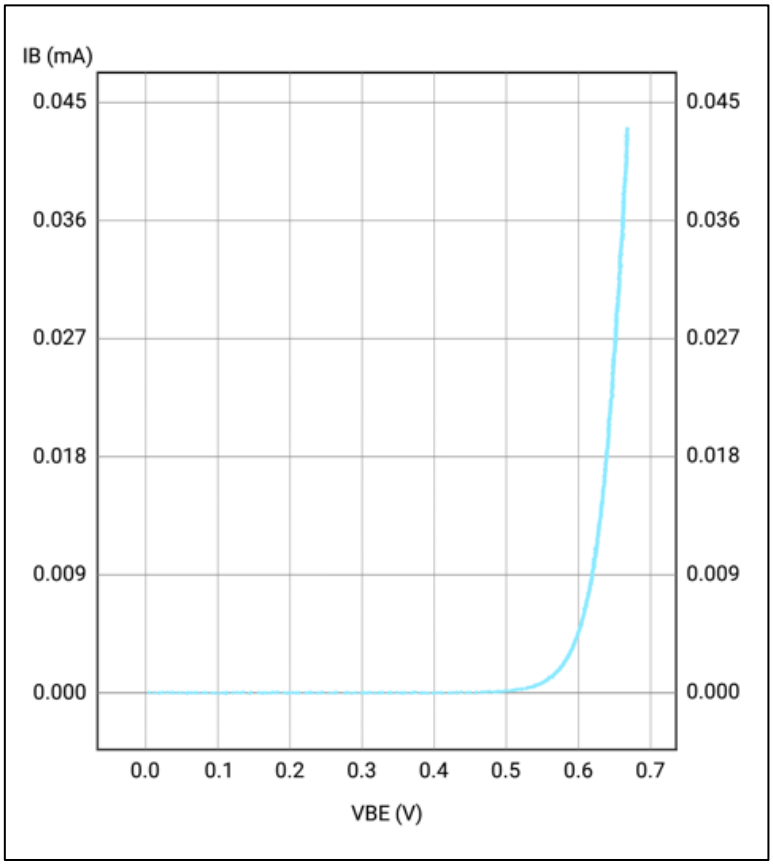


Figura 34. Ensaio de transístor NPN: curva característica de entrada.

A Figura 34 demonstra a representação gráfica dos resultados obtidos no ensaio da curva característica de entrada. Verifica-se que a curva obtida vai ao encontro do esperado, seguindo a mesma relação exponencial da curva teórica prevista no capítulo 2.

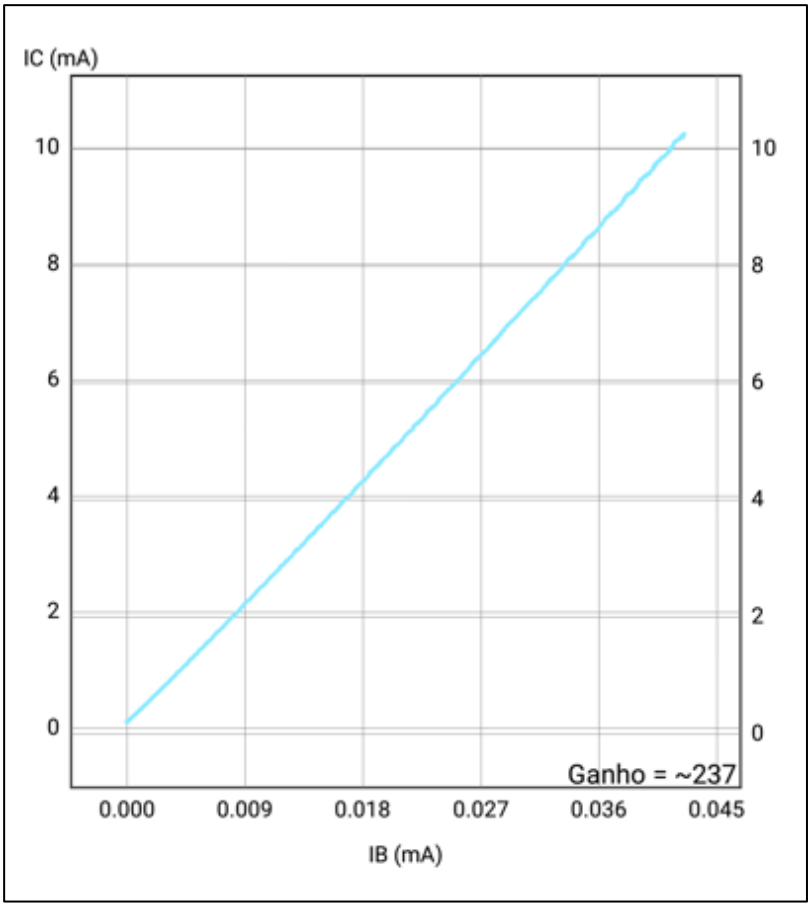


Figura 35. Ensaio de transístor NPN: curva característica de transferência.

Os resultados obtidos no ensaio da curva característica de transferência estão representados na Figura 35. Verifica-se que a curva respeita as previsões teóricas, e que o ganho de 237, calculado a partir do declive da reta, é consistente com o valor típico destes transístores.

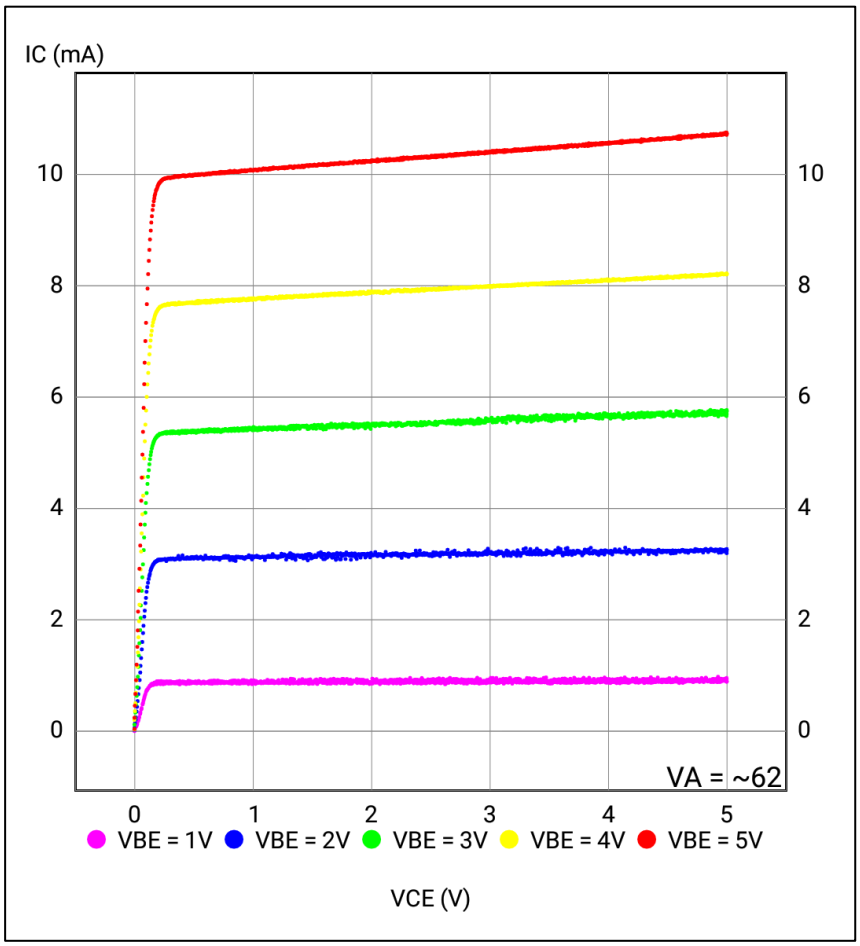


Figura 36. Ensaio de transístor NPN: curvas características de saída.

A Figura 36 mostra os resultados do ensaio experimental das curvas características de saída. Verifica-se mais uma vez que as curvas obtidas são coerentes com as curvas teóricas previstas no capítulo 2. O efeito de Early é claramente visível na representação gráfica, sendo o valor de V_A aproximadamente 62 V, calculado com base no declive da curva correspondente à tensão base-emissor de 5 V, onde o efeito é mais pronunciado.

4.2 Transístores Bipolares PNP

Para os ensaios experimentais de transístores bipolares PNP utilizou-se o transístor 2N2907A. Este transístor é complementar ao 2N2222A, na medida em que tem características semelhantes, mas simétricas.

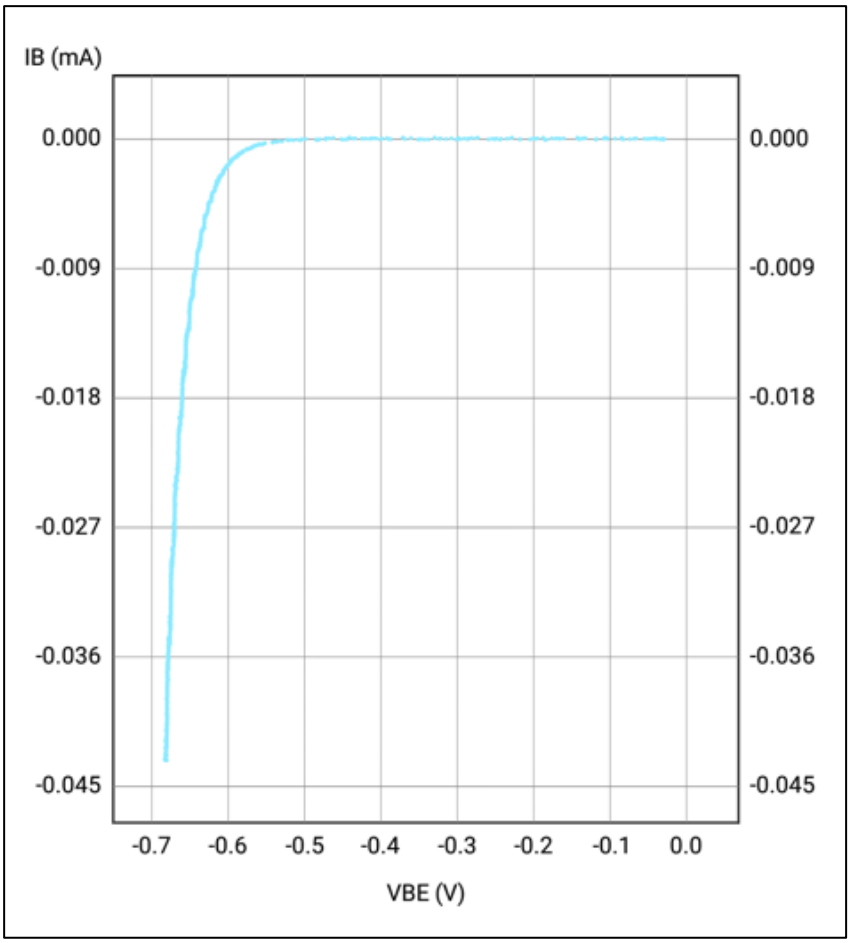


Figura 37. Ensaio de transístor PNP: curva característica de entrada.

A Figura 37, que representa os resultados do ensaio para a curva característica de entrada, demonstra que, apesar de simétrica, a curva é praticamente idêntica à obtida para transístores NPN, o que é esperado por serem transístores complementares.

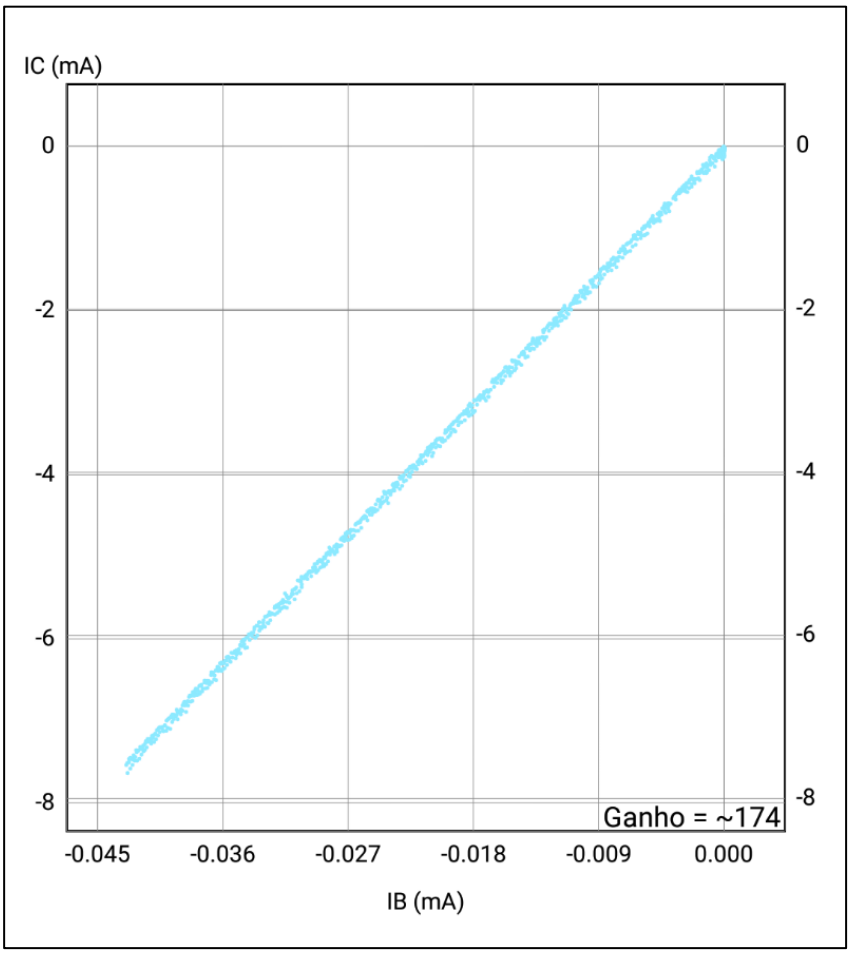


Figura 38. Ensaio de transístor PNP: curva característica de transferência.

Os resultados do ensaio para a curva característica de transferência, representados graficamente na Figura 38, seguem a relação proporcional esperada, e o ganho aproximado calculado, de 174, enquadra-se nos valores esperados.

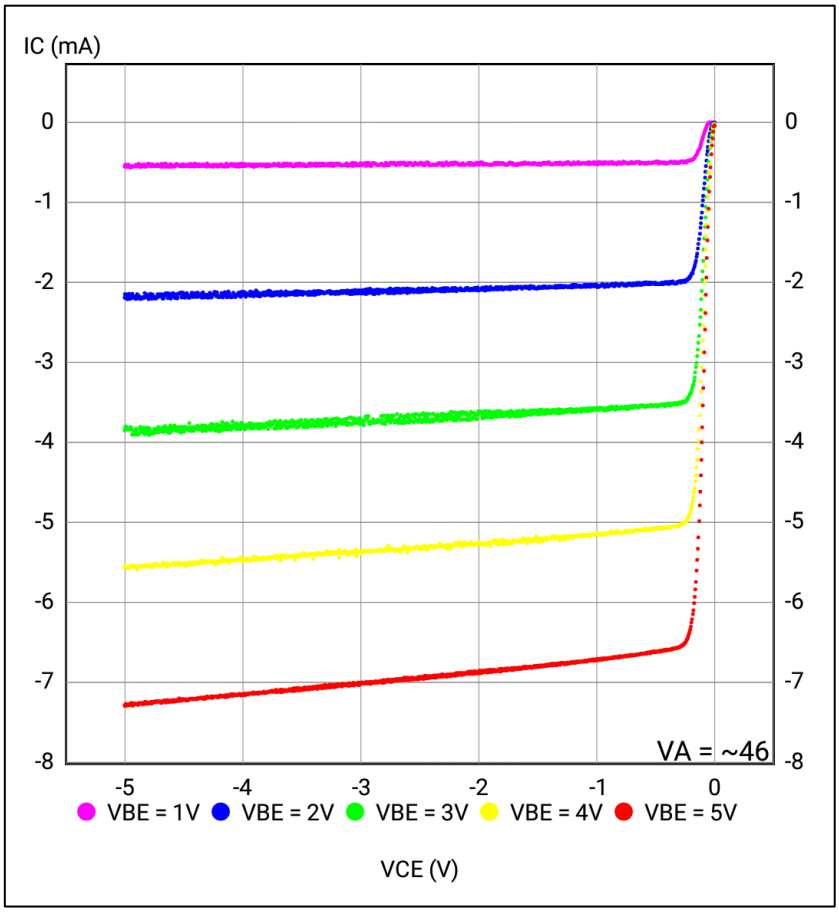


Figura 39. Ensaio de transístor PNP: curvas características de saída.

A Figura 39 representa os resultados do ensaio para as curvas características de saída. A família de curvas obtida, sendo simétrica ao obtido para transístores bipolares NPN, vai ao encontro do esperado, e uma vez mais, é claramente visível o efeito de Early, sendo a tensão de Early aproximadamente 46 V.

4.3 Transístores MOSFET

Para os ensaios experimentais de transístores MOSFET foram utilizados os transístores BS170 (NMOS) e BS250 (PMOS). Optou-se por configurar o dispositivo de forma a limitar o intervalo de valores de V_{GS} entre 2 e 3 V, com o objetivo de permitir a visualização de um maior número de curvas dentro dos limites funcionais do dispositivo.

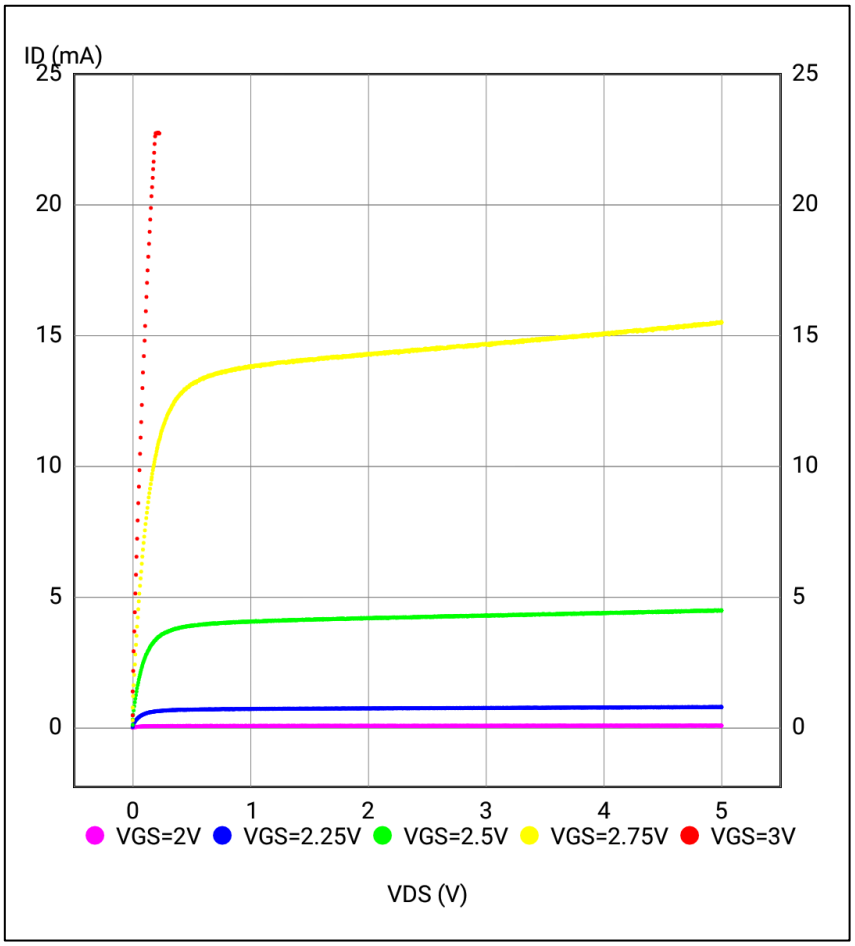


Figura 40. Curva de saída de transístor MOSFET de canal N.

A Figura 40 representa os resultados obtidos do ensaio do transístor NMOS. Verifica-se que para V_{GS} entre 2,25 e 2,75 V as curvas seguem o padrão teórico. A curva para V_{GS} de 3 V entra em saturação pois os valores da corrente de dreno ultrapassam os limites do circuito de condicionamento. Para V_{GS} de 2 V o transístor

não se encontra em condução, pois a sua tensão de *threshold* é superior a este valor, daí esta curva ser nula.

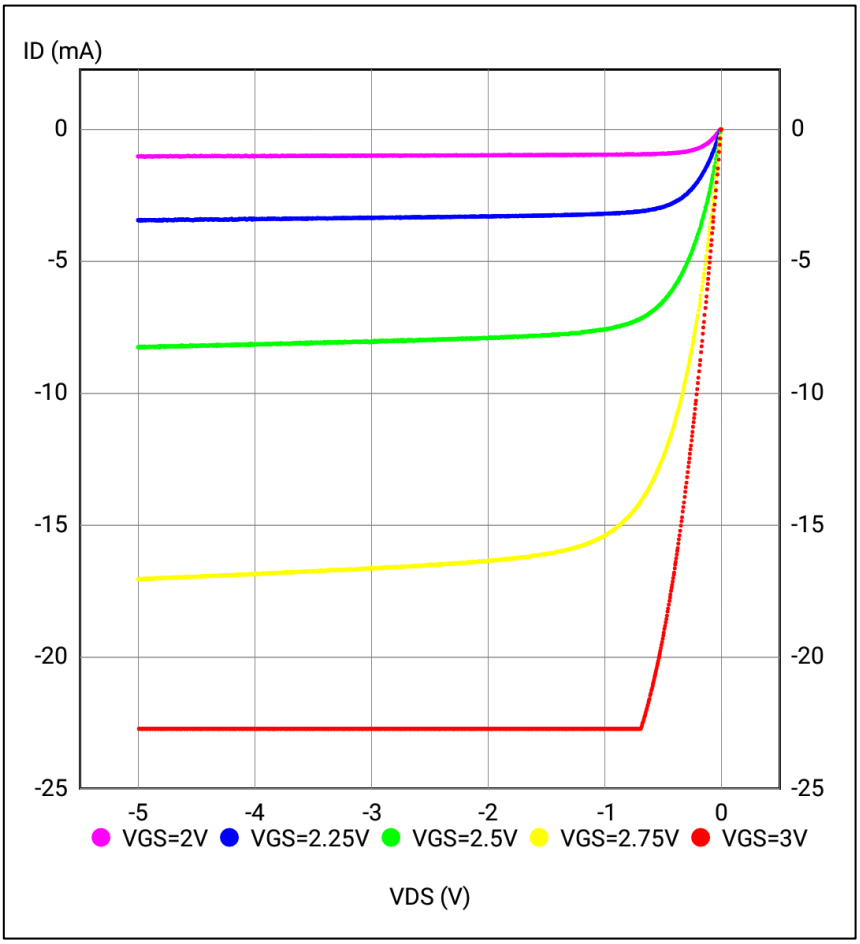


Figura 41. Curva de saída de transístor MOSFET de canal P.

Os resultados do ensaio experimental do transístor PMOS encontram-se representados na Figura 41, e verificam o esperado. A curva para V_{GS} de 3 V entra em saturação, devido a ultrapassar os limites funcionais do dispositivo.

5 Conclusão

Pretendeu-se com esta dissertação desenvolver um sistema portátil e de baixo custo que replicasse a funcionalidade de um traçador de curvas. Deveria ser capaz de extrair as curvas características de transístores bipolares e MOSFET, apresentar os resultados graficamente e permitir a exportação dos respetivos dados. Com base nos requisitos, desenvolveu-se um dispositivo portátil de pequenas dimensões e uma aplicação compatível com dispositivos móveis de sistema operativo Android. A aplicação, que estabelece ligação via *Bluetooth* com o dispositivo portátil, funciona como interface com o utilizador, permitindo a seleção do tipo de transístor e curva a ensaiar e a visualização e exportação dos resultados do ensaio. Considera-se que o trabalho foi bem-sucedido, apresentando resultados satisfatórios para o objetivo pretendido. O dispositivo equipará o laboratório de eletrónica da Escola Naval, onde permitirá aos alunos a visualização na prática de conceitos nucleares das áreas da eletrónica e medidas elétricas. Dada a sua portabilidade, este dispositivo é também uma mais-valia em contexto de ensino à distância, permitindo aos alunos não perder o contacto com a vertente prática da sua formação.

O desenvolvimento deste trabalho permitiu cimentar as competências e conhecimentos adquiridos no percurso académico da Escola Naval, quer pela sua vertente multidisciplinar, quer pela sua componente prática. Adicionalmente, permitiu ganhar competências na área da programação de aplicações móveis que poderão ser úteis em projetos de desenvolvimento tecnológico futuros.

Com base no trabalho realizado e resultados obtidos, considera-se que existem aspetos onde é possível ampliar as capacidades do dispositivo desenvolvido. Nesse sentido, seguem-se algumas sugestões de desenvolvimento futuro:

- Estender a aplicação a dispositivos móveis de sistema operativo iOS, de forma a ampliar o universo de compatibilidade do dispositivo;

- Implementar uma funcionalidade de indicação do nível de carga da bateria;

- Ampliar os limites funcionais do dispositivo:

    - via hardware, redesenhando o circuito de condicionamento de sinal para integrar resistências variáveis nas montagens com AMPOPs, de forma a permitir a configuração dos seus ganhos, o que possibilita maiores valores de corrente no circuito respeitando a gama de leitura do microcontrolador;

    - via software, implementando uma funcionalidade em que o dispositivo, na extração das curvas características de saída, com base nos valores de ganho ou tensão de *threshold* dos transístores, configura automaticamente os valores de V_{BE} ou V_{GS} de forma aos valores de corrente no circuito não atingirem a saturação.

6 Bibliografia

[1] https://www.globalspec.com/learnmore/test_measurement/electrical_testing_equipment/curve_tracers, acedido em março de 2021

[2] D. Mercer, "What is a Source Measurement Unit or SMU?", *Analog Dialogue*, dezembro 2017.
Disponível em: https://www.analog.com/en/analog-dialogue/studentzone/studentzone-december-2017.html

[3] J. Gorley, "What's the Difference Between a Classic Curve Tracer and SMU with Curve-Tracer Software?", *Electronic Design*, fevereiro 2020.
Disponível em: https://www.electronicdesign.com/technologies/test-measurement/article/21123551/whats-the-difference-between-a-classic-curve-tracer-and-smu-with-curvetracer-software

[4] A. S. Sedra e K. C. Smith, *Microelectronic Circuits*, 6ª Edição, EUA: Oxford University Press, 2009.

[5] S. Afzal, "I2C Primer: What is I2C? (Part 1)", *Analog Devices*.
Disponível em: https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html

[6] https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/, acedido em março de 2021

[7] https://www.arduino.cc/en/software, acedido em março de 2021

[8] https://www.arduino.cc/en/Reference/SoftwareSerial, acedido em março de 2021

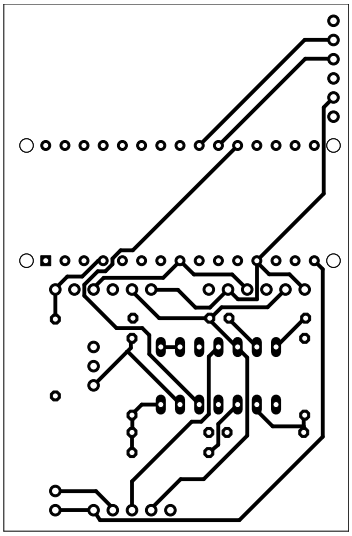
[9] https://github.com/adafruit/Adafruit_MCP4725, acedido em março de 2021

[10] https://developer.android.com/guide, acedido em março de 2021

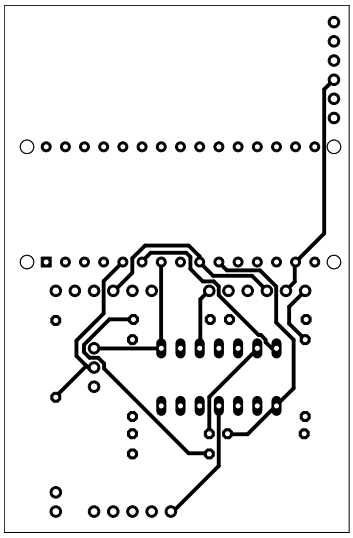
7 Apêndices

A. Placa de Circuito Impresso

A.1. Face Superior



A.2. Face Inferior

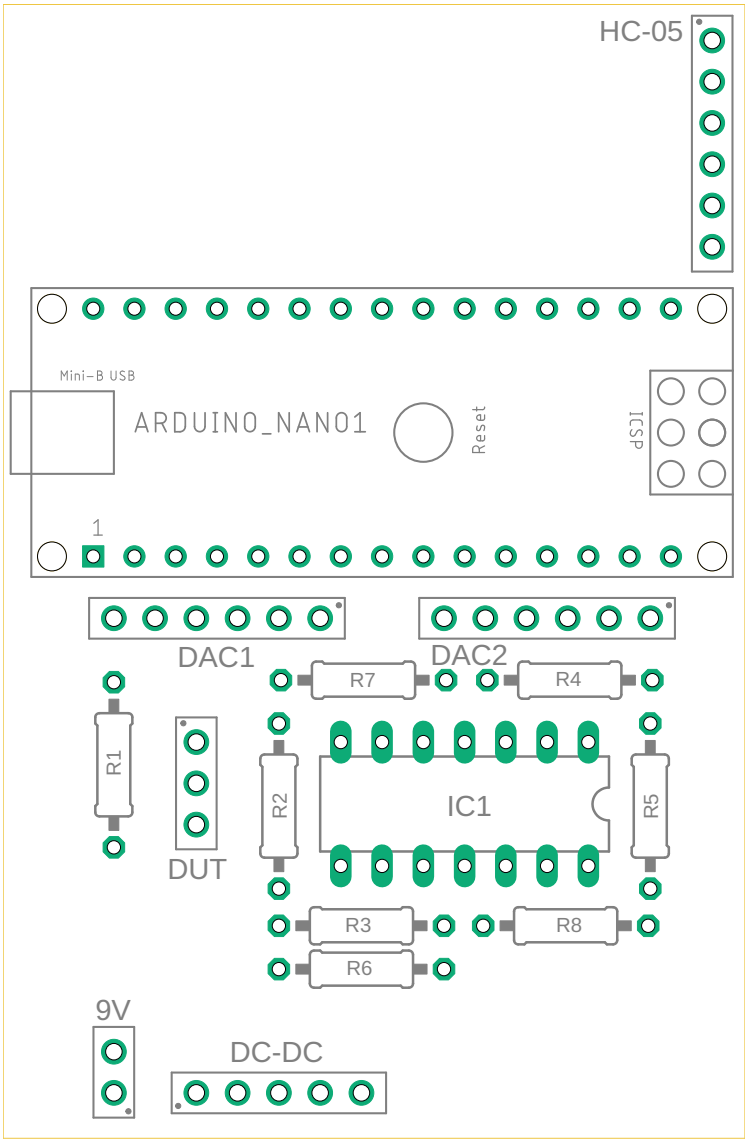


A.3. Lista dos Componentes

- Módulo *Bluetooth* HC-05 (ZS-040)
- Microcontrolador Arduino Nano V3

- R1 100KΩ
- R2 220Ω
- R3 10KΩ
- R4 10KΩ
- R5 10KΩ
- R6 10KΩ
- R7 10KΩ
- R8 10KΩ
- IC1 LM324N
- DAC1 Adafruit MCP4725
- DAC2 Adafruit MCP4725
- DC-DC Eletechsup DD1718PA 15V

A.4. Disposição dos Componentes



B. Código Fonte: Arduino

```
#include <Wire.h>
#include <SoftwareSerial.h>
#include <Adafruit_MCP4725.h>

SoftwareSerial BT(3,4); // cria porta série através dos pins 3(Rx) e
4(Tx)

Adafruit_MCP4725 dac;

char state = 'w'; // w-waiting, e-entrada, t-transferência, s-saída
char dutType = 'n'; // n-npn, p-pnp, x-nmos, y-pmos
char rcvByte;
boolean newData = false;

const float nr_samples = 10.00; //número de amostras por PF

const float R1 = 100.00; // valor em KOhm para resultado ser em mA
const float R2 = 0.22;

float meanAI0, meanAI1, meanAI2, meanAI3, meanAI6;

void setup()
{
  Serial.begin(115200); // porta série nativa do Arduino

  BT.begin(115200); // porta série via Bluetooth

  pinMode(A0,INPUT);
  pinMode(A1,INPUT);
  pinMode(A2,INPUT);
  pinMode(A3,INPUT);
  pinMode(A6,INPUT);

  pinMode(2,OUTPUT);

  dac.begin(0x62); // DAC1
  dac.setVoltage(0,false);

  dac.begin(0x63); // DAC2
  dac.setVoltage(0,false);

  digitalWrite(2,LOW);

  Serial.println("ARDUINO READY");
}


void readDataBT()
{
  if (BT.available() > 0)
  {
    rcvByte = BT.read();
    newData = true;
  }
}
```

```
void measureInputsNPN()
{
  float AI0, AI1, AI2, AI3;
  float samplesAI0 = 0;
  float samplesAI1 = 0;
  float samplesAI2 = 0;
  float samplesAI3 = 0;

  for (byte i = 1; i <= nr_samples; i++)
  {
      AI0 = analogRead(A0) * (5.00/1023.00);
      AI1 = analogRead(A1) * (5.00/1023.00);
      AI2 = analogRead(A2) * (5.00/1023.00);
      AI3 = analogRead(A3) * (5.00/1023.00);

      samplesAI0 += AI0;
      samplesAI1 += AI1;
      samplesAI2 += AI2;
      samplesAI3 += AI3;
  }

  meanAI0 = samplesAI0/nr_samples;
  meanAI1 = samplesAI1/nr_samples;
  meanAI2 = samplesAI2/nr_samples;
  meanAI3 = samplesAI3/nr_samples;
}


void measureInputsPNP()
{
  float AI0, AI1, AI3, AI6;
  float samplesAI0 = 0;
  float samplesAI1 = 0;
  float samplesAI3 = 0;
  float samplesAI6 = 0;

  for (byte i = 1; i <= nr_samples; i++)
  {
      AI0 = analogRead(A0) * (5.00/1023.00);
      AI1 = analogRead(A1) * (5.00/1023.00);
      AI3 = analogRead(A3) * (5.00/1023.00);
      AI6 = analogRead(A6) * (5.00/1023.00);

      samplesAI0 += AI0;
      samplesAI1 += AI1;
      samplesAI3 += AI3;
      samplesAI6 += AI6;
  }

  meanAI0 = samplesAI0/nr_samples;
  meanAI1 = samplesAI1/nr_samples;
  meanAI3 = samplesAI3/nr_samples;
  meanAI6 = samplesAI6/nr_samples;
}
```

```
void curvaEntradaNPN()
{
  Serial.println("curvaEntradaNPN(): started");

  float VBE, IB;

  digitalWrite(2,LOW);

  dac.begin(0x63); // DAC2
  dac.setVoltage(4095,false); // 5V

  Serial.println();
  Serial.println("     VBE            IB");
  Serial.println();

  dac.begin(0x62); // DAC1

  for (int i = 0; i <= 4095; i += 4)
  {
    dac.setVoltage(i,false);

    delay(1);

    measureInputsNPN();

    VBE = meanAI1;
    IB = (meanAI0-meanAI1)/R1;

    byte* byteVBE = (byte*) &VBE;
    byte* byteIB = (byte*) &IB;

    BT.write(byteVBE,4);
    BT.write(byteIB,4);

    Serial.print(VBE,DEC);
    Serial.print(" , ");
    Serial.println(IB,DEC);
  }

  dac.begin(0x62); // DAC1
  dac.setVoltage(0,false);

  dac.begin(0x63); // DAC2
  dac.setVoltage(0,false);

  Serial.println("curvaEntradaNPN(): finished");
}


void curvaTransferenciaNPN()
{
  Serial.println("curvaTransferenciaNPN(): started");

  float IB, IC, IE;

  digitalWrite(2,LOW);

  dac.begin(0x63); // DAC2
  dac.setVoltage(4095,false); // 5V
```

```
  Serial.println();
  Serial.println("     IB              IC");
  Serial.println();

  dac.begin(0x62); // DAC1

  for (int i = 0; i <= 4095; i += 4)
  {
    dac.setVoltage(i,false);

    delay(1);

    measureInputsNPN();

    IB = (meanAI0-meanAI1)/R1;
    IE = meanAI2/R2;
    IC = IE - IB;

    byte* byteIB = (byte*) &IB;
    byte* byteIC = (byte*) &IC;

    BT.write(byteIB,4);
    BT.write(byteIC,4);

    Serial.print(IB,DEC);
    Serial.print(" , ");
    Serial.println(IC,DEC);
  }

  dac.begin(0x62); // DAC1
  dac.setVoltage(0,false);

  dac.begin(0x63); // DAC2
  dac.setVoltage(0,false);

  Serial.println("curvaTransferenciaNPN(): finished");
}


void curvaSaidaNPN()
{
  Serial.println("curvaSaidaNPN(): started");

  float IB, IC, IE, VCE;

  int valuesVBE[5] = {820, 1638, 2458, 3276, 4095}; // corresponente a
VBE = 1V, 2V, 3V, 4V e 5V

  digitalWrite(2,LOW);

  for (byte i = 0; i <= 4; i++) // VBE
  {
    dac.begin(0x62);
    dac.setVoltage(valuesVBE[i],false);

    delay(1);

    Serial.println();
```

```
    Serial.print("Curva para VBE = ");
    Serial.print(i+1);
    Serial.println(" V");

    Serial.println();
    Serial.println("     VCE            IC");
    Serial.println();

    for (int i = 0; i <= 4095; i += 4) // VCE
    {
      dac.begin(0x63);
      dac.setVoltage(i,false);

      delay(1);

      measureInputsNPN();

      IB = (meanAI0-meanAI1)/R1;
      IE = meanAI2/R2;
      IC = IE - IB;
      VCE = meanAI3;

      byte* byteVCE = (byte*) &VCE;
      byte* byteIC = (byte*) &IC;

      BT.write(byteVCE,4);
      BT.write(byteIC,4);

      Serial.print(VCE,DEC);
      Serial.print(" , ");
      Serial.println(IC,DEC);
    }
  }

  dac.begin(0x62); //DAC1
  dac.setVoltage(0,false);

  dac.begin(0x63); //DAC2
  dac.setVoltage(0,false);

  Serial.println("curvaSaidaNPN(): finished");
}


void curvaEntradaPNP()
{
  Serial.println("curvaEntradaPNP(): started");

  float IB, VBE;

  Serial.println();
  Serial.println("     VBE            IB");
  Serial.println();

  digitalWrite(2,HIGH);

  dac.begin(0x63); // DAC2
  dac.setVoltage(0,false);
```

```
  dac.begin(0x62); // DAC1

  for (int i = 4095; i >= 0; i -= 4)
  {
    dac.setVoltage(i,false);

    delay(1);

    measureInputsPNP();

    VBE = meanAI1 - 5;
    IB = (meanAI0-meanAI1)/R1;

    byte* byteVBE = (byte*) &VBE;
    byte* byteIB = (byte*) &IB;

    BT.write(byteVBE,4);
    BT.write(byteIB,4);

    Serial.print(VBE,DEC);
    Serial.print(" , ");
    Serial.println(IB,DEC);
  }

  dac.begin(0x62); // DAC1
  dac.setVoltage(0,false);

  digitalWrite(2,LOW);

  Serial.println("curvaEntradaPNP(): finished");
}


void curvaTransferenciaPNP()
{
  Serial.println("curvaTransferenciaPNP(): started");

  float IB, IC, IE;

  Serial.println();
  Serial.println("     IB              IC");
  Serial.println();

  digitalWrite(2,HIGH);

  dac.begin(0x63); // DAC2
  dac.setVoltage(0,false);

  dac.begin(0x62); // DAC1

  for (int i = 4095; i >= 0; i -= 4)
  {
    dac.setVoltage(i,false);

    delay(1);

    measureInputsPNP();

    IB = (meanAI0-meanAI1)/R1;
```

```
    IE = -meanAI6/R2;
    IC = IE + IB;

    byte* byteIB = (byte*) &IB;
    byte* byteIC = (byte*) &IC;

    BT.write(byteIB,4);
    BT.write(byteIC,4);

    Serial.print(IB,DEC);
    Serial.print(" , ");
    Serial.println(IC,DEC);
  }

  dac.begin(0x62); // DAC1
  dac.setVoltage(0,false);

  digitalWrite(2,LOW);

  Serial.println("curvaTransferenciaPNP(): finished");
}


void curvaSaidaPNP()
{
  Serial.println("curvaSaidaPNP(): started");

  float IB, IC, IE, VCE;

  int valuesVBE[5] = {3276, 2458, 1638, 820, 0}; // corresponente a
[4,3,2,1,0]V, ou seja, VBE = [1,2,3,4,5]V

  digitalWrite(2,HIGH);

  for (byte i = 0; i <= 4; i++) // VBE
  {
    dac.begin(0x62); // DAC1
    dac.setVoltage(valuesVBE[i],false);

    delay(1);

    Serial.println();
    Serial.print("Curva para VBE = ");
    Serial.print(i+1);
    Serial.println(" V");

    Serial.println();
    Serial.println("     VCE             IC");
    Serial.println();


    for (int i = 4095; i >= 0; i -= 4) // VCE
    {
      dac.begin(0x63); // DAC2
      dac.setVoltage(i,false);

      delay(1);

      measureInputsPNP();
```

```
      IB = (meanAI0-meanAI1)/R1;
      IE = -meanAI6/R2;
      IC = IE + IB;
      VCE = meanAI3 - 5;

      byte* byteVCE = (byte*) &VCE;
      byte* byteIC = (byte*) &IC;

      BT.write(byteVCE,4);
      BT.write(byteIC,4);

      Serial.print(VCE,DEC);
      Serial.print(" , ");
      Serial.println(IC,DEC);
    }
  }

  dac.begin(0x62); // DAC1
  dac.setVoltage(0,false);

  dac.begin(0x63); // DAC2
  dac.setVoltage(0,false);

  digitalWrite(2,LOW);

  Serial.println("curvaSaidaPNP(): finished");
}


void curvaSaidaNMOS()
{
  Serial.println("curvaSaidaNMOS(): started");

  float IG, ID, IS, VDS;

  int valuesVGS[5] = {1638, 1843, 2048, 2252, 2458}; // corresponente
a VGS = 2V, 2.25V, 2.5V, 2.75V e 3V

  digitalWrite(2,LOW);

  for (byte i = 0; i <= 4; i++) // VBE
  {
    dac.begin(0x62);
    dac.setVoltage(valuesVGS[i],false);

    delay(1);

    Serial.println();
    Serial.print("Curva para VGS = ");
    Serial.print(valuesVGS[i]/4095*5);
    Serial.println(" V");

    Serial.println();
    Serial.println("     VDS            ID");
    Serial.println();

    for (int i = 0; i <= 4095; i += 4) // VDS
    {
```

```
      dac.begin(0x63);
      dac.setVoltage(i,false);

      delay(1);

      measureInputsNPN();

      IS = meanAI2/R2;
      ID = IS;
      VDS = meanAI3;

      byte* byteVDS = (byte*) &VDS;
      byte* byteID = (byte*) &ID;

      BT.write(byteVDS,4);
      BT.write(byteID,4);

      Serial.print(VDS,DEC);
      Serial.print(" , ");
      Serial.println(ID,DEC);
    }
  }

  dac.begin(0x62); //DAC1
  dac.setVoltage(0,false);

  dac.begin(0x63); //DAC2
  dac.setVoltage(0,false);

  Serial.println("curvaSaidaNMOS(): finished");
}


void curvaSaidaPMOS()
{
  Serial.println("curvaSaidaPMOS(): started");

  float IG, ID, IS, VDS;

  int valuesVGS[5] = {2458, 2252, 2048, 1843, 1638}; // corresponente
a [3, 2.75, 2.5, 2.25, 2]V, ou seja, VGS = [2, 2.25, 2.5, 2.75, 3]V

  digitalWrite(2,HIGH);

  for (byte i = 0; i <= 4; i++) // VBE
  {
    dac.begin(0x62); // DAC1
    dac.setVoltage(valuesVGS[i],false);

    delay(1);

    Serial.println();
    Serial.print("Curva para VGS = ");
    Serial.print(5-valuesVGS[i]/4095*5);
    Serial.println(" V");

    Serial.println();
    Serial.println("     VDS            ID");
    Serial.println();
```

```
    for (int i = 4095; i >= 0; i -= 4) // VDS
    {
      dac.begin(0x63); // DAC2
      dac.setVoltage(i,false);

      delay(1);

      measureInputsPNP();

      IS = -meanAI6/R2;
      ID = IS;
      VDS = meanAI3 - 5;

      byte* byteVDS = (byte*) &VDS;
      byte* byteID = (byte*) &ID;

      BT.write(byteVDS,4);
      BT.write(byteID,4);

      Serial.print(VDS,DEC);
      Serial.print(" , ");
      Serial.println(ID,DEC);
    }
  }

  dac.begin(0x62); // DAC1
  dac.setVoltage(0,false);

  dac.begin(0x63); // DAC2
  dac.setVoltage(0,false);

  digitalWrite(2,LOW);

  Serial.println("curvaSaidaPMOS(): finished");
}


void selectFunc()
{
  if (state == 'e' && dutType == 'n')
  {
    curvaEntradaNPN();
    state = 'w';
  }

  else if (state == 't' && dutType == 'n')
  {
    curvaTransferenciaNPN();
    state = 'w';
  }

  else if (state == 's' && dutType == 'n')
  {
    curvaSaidaNPN();
    state = 'w';
  }
  else if (state == 'e' && dutType == 'p')
```

```
  {
    curvaEntradaPNP();
    state = 'w';
  }

  else if (state == 't' && dutType == 'p')
  {
    curvaTransferenciaPNP();
    state = 'w';
  }

  else if (state == 's' && dutType == 'p')
  {
    curvaSaidaPNP();
    state = 'w';
  }

  else if (dutType == 'x') // NMOS
  {
    curvaSaidaNMOS();
    state = 'w';
  }

  else if (dutType == 'y') // PMOS
  {
    curvaSaidaPMOS();
    state = 'w';
  }
}

void loop()
{
  readDataBT();

  if (newData == true)
  {
    if (rcvByte == 'n' || rcvByte == 'p' || rcvByte == 'x' || rcvByte
== 'y')
    {
      dutType = rcvByte;
    }

    else
    {
      state = rcvByte;

      selectFunc();
    }

    newData = false;
  }
}
```

C. Código Fonte: Android Studio

C.1. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tese">

    <uses-feature android:name="android.hardware.bluetooth" />

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
/>
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.TESE">

        <activity android:name=".MainActivity"
            android:screenOrientation="portrait"/>

        <activity android:name=".FirstActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

       <provider
           android:authorities="com.example.tese.fileprovider"
           android:name="androidx.core.content.FileProvider"
           android:grantUriPermissions="true"
           android:exported="false">
           <meta-data
               android:name="android.support.FILE_PROVIDER_PATHS"
               android:resource="@xml/provider_paths" />
       </provider>

    </application>

</manifest>
```

C.2. FirstActivity.java

```
package com.example.tese;

import androidx.appcompat.app.AppCompatActivity;

import android.Manifest;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Build;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ListView;

import java.util.ArrayList;

public class FirstActivity extends AppCompatActivity implements
AdapterView.OnItemClickListener {

    private static final String TAG = "MainActivity";

    BluetoothAdapter bluetoothAdapter;
    BluetoothDevice bluetoothDevice;

    Button refreshButton;
    ListView listView;

    public ArrayList<BluetoothDevice> foundBTDevices;
    public DeviceListAdapter deviceListAdapter;

    int ACTION_REQUEST_MULTIPLE_PERMISSION = 1;

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

        int permissionsCheck;

        // Verifica a versão do sistema operativo caso seja necessário
solicitar permissões
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {

            permissionsCheck =
this.checkSelfPermission("Manifest.permission.ACCESS_FINE_LOCATION");
            permissionsCheck +=
this.checkSelfPermission("Manifest.permission.ACCESS_COARSE_LOCATION")
;
            permissionsCheck +=
this.checkSelfPermission("Manifest.permission.BLUETOOTH_ADMIN");
```

```
            permissionsCheck +=
this.checkSelfPermission("Manifest.permission.BLUETOOTH");

            if (permissionsCheck != 0) {
                this.requestPermissions(new
String[]{Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.ACCESS_COARSE_LOCATION,
Manifest.permission.BLUETOOTH_ADMIN, Manifest.permission.BLUETOOTH},
ACTION_REQUEST_MULTIPLE_PERMISSION);
            }
        }

        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        foundBTDevices = new ArrayList<>();

        refreshButton = findViewById(R.id.connectButton);
        listView = findViewById(R.id.myListView);

        refreshButton.setOnClickListener(view -> findDevices());

        enableBT();

        findDevices();

        listView.setOnItemClickListener(FirstActivity.this);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(enablingBroadcast);
        unregisterReceiver(findingBroadcast);

        bluetoothAdapter.cancelDiscovery();
    }

    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int
i, long l) {

        bluetoothAdapter.cancelDiscovery();

        String deviceName = foundBTDevices.get(i).getName();
        String deviceAddress = foundBTDevices.get(i).getAddress();

        bluetoothDevice = foundBTDevices.get(i);

        // Gera um novo Intent para mudar de atividade, passando os
dados de identificação do dispositivo Bluetooth
        Intent changeActivity = new Intent(FirstActivity.this,
MainActivity.class);
        changeActivity.putExtra("bluetoothDeviceName", deviceName);
        changeActivity.putExtra("bluetoothDeviceAddress",
deviceAddress);
        startActivity(changeActivity);
    }

    // Caso o Bluetooth do dispositivo se encontre desligado, solicita
a sua ativação
```

```
    private void enableBT() {
        Log.d(TAG, "enableBT: Started");

        IntentFilter intentFilter = new
IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
        registerReceiver(enablingBroadcast, intentFilter);

        if(bluetoothAdapter == null) {
            Log.d(TAG, "enableBT: Device does not have Bluetooth
capabilities");

            finish();
        }

        if(!bluetoothAdapter.isEnabled()) {
            Log.d(TAG, "enableBT: Enabling Bluetooth...");

            Intent enableBTIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivity(enableBTIntent);
        }
    }

    // Inicia ou reinicia a procura por outros dispositivos Bluetooth
    public void findDevices() {
        Log.d(TAG, "findDevices: Started");

        IntentFilter intentFilter = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
        registerReceiver(findingBroadcast, intentFilter);

        if (bluetoothAdapter.isDiscovering()) {

            bluetoothAdapter.cancelDiscovery();

            bluetoothAdapter.startDiscovery();
            Log.d(TAG, "findDevices: Looking for bluetooth devices");
        }

        if(!bluetoothAdapter.isDiscovering()) {

            bluetoothAdapter.startDiscovery();
            Log.d(TAG, "findDevices: Looking for bluetooth devices");
        }
    }

    // Escuta o sistema Android em busca de mensagens de alteração do
estado do Bluetooth do dispositivo
    private final BroadcastReceiver enablingBroadcast = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            final String action = intent.getAction();

            if(action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)) {

                final int state =
intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
```

```
BluetoothAdapter.ERROR);

                switch(state) {
                    case BluetoothAdapter.STATE_OFF:
                        Log.d(TAG, "onReceive: STATE OFF");
                        break;
                    case BluetoothAdapter.STATE_TURNING_OFF:
                        Log.d(TAG, "onReceive: STATE TURNING OFF");
                        break;
                    case BluetoothAdapter.STATE_ON:
                        Log.d(TAG, "onReceive: STATE ON");

                        findDevices();

                        break;
                    case BluetoothAdapter.STATE_TURNING_ON:
                        Log.d(TAG, "onReceive: STATE TURNING ON");
                        break;
                }
            }
        }
    };

    // Escuta o sistema Android em busca de mensagens relativas à
descoberta de dispositivos Bluetooth disponíveis
    private final BroadcastReceiver findingBroadcast = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            final String action = intent.getAction();

            if(action.equals(BluetoothDevice.ACTION_FOUND)) {

                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

                Log.d(TAG, "DeviceFound: "+ device.getName() +" :
"+device.getAddress());

                foundBTDevices.add(device);

                deviceListAdapter = new DeviceListAdapter(context,
R.layout.devicelistadapter_layout, foundBTDevices);
                listView.setAdapter(deviceListAdapter);
            }
        }
    };
}
```

C.3. MainActivity.java

```
package com.example.tese;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.FileProvider;

import android.annotation.SuppressLint;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.graphics.Color;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.Spinner;
import android.widget.TextView;

import com.github.mikephil.charting.charts.ScatterChart;
import com.github.mikephil.charting.components.Description;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.ScatterData;
import com.github.mikephil.charting.data.ScatterDataSet;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.ArrayList;
import java.util.UUID;


public class MainActivity extends AppCompatActivity implements
AdapterView.OnItemSelectedListener {

    private static final String TAG = "SecondActivity";

    private static final UUID SPP_UUID = UUID.fromString("00001101-
```

```
0000-1000-8000-00805F9B34FB");

    BluetoothAdapter bluetoothAdapter;
    BluetoothDevice bluetoothDevice;
    BluetoothSocket bluetoothSocket;

    Button startButton;
    TextView xLabel, yLabel;
    Spinner curveSpinner, typeSpinner;

    ScatterChart scatterView;
    ScatterDataSet scatterDataSet;
    ScatterData scatterData;
    Description description;
    Legend legend;
    XAxis xAxis;
    YAxis yAxis;
    ArrayList<Entry> entriesArray;

    InputStream inStream;
    OutputStream outStream;

    String curveType, transistorType, deviceName, deviceAddress;
    StringBuilder exportData;

    byte[] dataArray, xBytesArray, yBytesArray;
    ByteBuffer dataBuffer;

    int rcvBytes, totalBytes;

    float xPointMax, yPointMax, xPointMin, yPointMin, xPointA,
yPointA, xPointB, yPointB;

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

        Log.d(TAG, "SecondActivity: Started");

        IntentFilter intentFilter = new
IntentFilter(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
        registerReceiver(bondingBroadcast, intentFilter);

        // Recebe os dados de identificação do dispositivo Bluetooth
        Intent receivingIntent = getIntent();
        deviceName =
receivingIntent.getStringExtra("bluetoothDeviceName");
        deviceAddress =
receivingIntent.getStringExtra("bluetoothDeviceAddress");

        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        bluetoothDevice =
bluetoothAdapter.getRemoteDevice(deviceAddress);

        startButton = findViewById(R.id.startButton);
        xLabel = findViewById(R.id.xLabel);
        yLabel = findViewById(R.id.yLabel);
        description = new Description();
```

```
        description.setTextSize(12);
        scatterView = findViewById(R.id.scatterView);
        scatterView.setNoDataText("NO DATA");
        scatterView.setDescription(description);
        scatterView.setDrawBorders(true);
        legend = scatterView.getLegend();
        legend.setForm(Legend.LegendForm.CIRCLE);
        legend.setXEntrySpace(5);
        xAxis = scatterView.getXAxis();
        xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
        xAxis.setDrawLabels(true);
        yAxis = scatterView.getAxisLeft();
        yAxis.setDrawLabels(true);

        typeSpinner = findViewById(R.id.typeSpinner);
        ArrayAdapter<CharSequence> typeSpinnerAdapter =
ArrayAdapter.createFromResource(this, R.array.typeSpinnerOptions,
R.layout.spinner_layout);

typeSpinnerAdapter.setDropDownViewResource(R.layout.dropdown_layout);
        typeSpinner.setAdapter(typeSpinnerAdapter);
        typeSpinner.setOnItemSelectedListener(this);

        curveSpinner = findViewById(R.id.curveSpinner);
        ArrayAdapter<CharSequence> curveSpinnerAdapter =
ArrayAdapter.createFromResource(this, R.array.curveSpinnerOptions,
R.layout.spinner_layout);

curveSpinnerAdapter.setDropDownViewResource(R.layout.dropdown_layout);
        curveSpinner.setAdapter(curveSpinnerAdapter);
        curveSpinner.setOnItemSelectedListener(this);

        ConnectThread connectThread = new ConnectThread();
        connectThread.start();

        startButton.setOnClickListener(view -> {

            GettingDataThread gettingDataThread = new
GettingDataThread();
            gettingDataThread.start();

        });
    }

    // Inicia a Thread para conexão com o dispositivo Bluetooth
    private class ConnectThread extends Thread {

        public void run() {
            Log.d(TAG, "ConnectThread: run(): started");

            if (Build.VERSION.SDK_INT >
Build.VERSION_CODES.JELLY_BEAN_MR2) {

                Log.d(TAG, "ConnectThread: run(): Pairing to: " +
deviceName + " : " + deviceAddress);

                int bondState = bluetoothDevice.getBondState();

                // Verifica se o dispositivo Bluetooth já se encontra
```

```
emparelhado
                if (bondState == BluetoothDevice.BOND_BONDED) {

                    connectToDevice(bluetoothDevice);

                } else {

                    bluetoothDevice.createBond();
                }
            }
        }
    }

    // Thread responsável pela aquisição de dados
    public class GettingDataThread extends Thread {

        ProgressBar progressBar = findViewById(R.id.progressBar);

        @SuppressLint("SetTextI18n")
        public void run() {
            Log.d(TAG, "GettingDataThread: run(): started");

            progressBar.setIndeterminate(false);
            progressBar.setProgress(0);

            // Indica ao Arduino o tipo de curva e de transístor
            OutStreamWrite(transistorType);
            OutStreamWrite(curveType);

            if (curveType.equals("e") || curveType.equals("t")) {

                totalBytes = 8192;

            } else if (curveType.equals("s")){

                totalBytes = 40960;
            }

            progressBar.setMax(totalBytes);

            dataArray = new byte[totalBytes];
            dataBuffer = ByteBuffer.wrap(dataArray);
            dataBuffer.clear();

            rcvBytes = 0;
            xPointMax = 0;
            xPointMin = 0;
            yPointMax = 0;
            yPointMin = 0;

            int nrBytes;
            byte[] buffer = new byte[32];

            // Recebe os dados do Arduino enquanto existirem
            while (true) {

                try {

                    if (inStream.available() > 0) {
```

```
                        nrBytes = inStream.read(buffer);

                        dataBuffer.put(buffer, 0, nrBytes);

                        rcvBytes += nrBytes;

                        progressBar.incrementProgressBy(nrBytes);

                        Log.d(TAG, "GettingDataThread: rcvBytes: " +
rcvBytes);

                        if (rcvBytes >= totalBytes) {

                            break;
                        }
                    }

                } catch (IOException e) {
                    Log.d(TAG, "InputStream: IOException: " +
e.getMessage());
                    break;
                }
            }

            dataBuffer.rewind();

            plotGraph();

            runOnUiThread(() -> {

                switch (curveType) {
                    case "e":
                        xLabel.setText("VBE (V)");
                        yLabel.setText("IB (mA)");
                        break;
                    case "t":
                        xLabel.setText("IB (mA)");
                        yLabel.setText("IC (mA)");
                        break;
                    case "s":

                        if (transistorType.equals("n") ||
transistorType.equals("p")) {

                            xLabel.setText("VCE (V)");
                            yLabel.setText("IC (mA)");

                        } else if (transistorType.equals("x") ||
transistorType.equals("y")) {

                            xLabel.setText("VDS (V)");
                            yLabel.setText("ID (mA)");
                        }
                        break;
                }

                scatterView.invalidate();
            });
```

```
        }
    }

    // Inicia a ligação com o dispositivo e abre a porta série de
comunicação
    public void connectToDevice(BluetoothDevice device) {

        Log.d(TAG, "connectToDevice(): started, trying to connect");

        bluetoothSocket = null;

        try {

            bluetoothSocket =
device.createInsecureRfcommSocketToServiceRecord(SPP_UUID);
            bluetoothSocket.connect();

            Log.d(TAG, "connectToDevice(): Connection successful");

            outStream = bluetoothSocket.getOutputStream();
            inStream = bluetoothSocket.getInputStream();

        } catch (IOException e) {
            Log.d(TAG, "connectToDevice(): Connection failed: " +
e.getMessage());
        }
    }

    // Converte os dados recebidos em bytes para valores númericos
    public void bytesToEntries() {
        Log.d(TAG, "bytesToEntries: started");

        xBytesArray = new byte[4];
        yBytesArray = new byte[4];

        entriesArray = new ArrayList<>();

        for (int i = 1; i <= 1024; i++) {

            int length = 4;

            dataBuffer.get(xBytesArray, 0, length);
            dataBuffer.get(yBytesArray, 0, length);

            float xPoint =
ByteBuffer.wrap(xBytesArray).order(ByteOrder.LITTLE_ENDIAN).getFloat()
;
            float yPoint =
ByteBuffer.wrap(yBytesArray).order(ByteOrder.LITTLE_ENDIAN).getFloat()
;

            // Pontos para cálculo das margens
            if (xPoint > xPointMax) {
                xPointMax = xPoint;
            } else if (yPoint > yPointMax) {
                yPointMax = yPoint;
            } else if (xPoint < xPointMin) {
                xPointMin = xPoint;
            } else if (yPoint < yPointMin) {
```

```
                yPointMin = yPoint;
            }

            // Pontos para cálculo do ganho e tensão de Early
            if (!curveType.equals("e")) {

                if (i == 200) {
                    xPointA = xPoint;
                    yPointA = yPoint;
                } else if (i == 922) {
                    xPointB = xPoint;
                    yPointB = yPoint;
                }
            }

            exportData.append(xPoint).append(" ,
").append(yPoint).append("\n");

            entriesArray.add(new Entry(xPoint,yPoint));
            Log.d(TAG, "plotGraph(): dataPoint: " + xPoint + " : " +
yPoint);
        }
    }

    // Representa os dados em gráficos X-Y
    public void plotGraph() {
        Log.d(TAG, "plotGraph(): started");

        scatterData = new ScatterData();
        exportData = new StringBuilder();

        switch (curveType) {

            case "e":

                bytesToEntries();

                scatterDataSet = new ScatterDataSet(entriesArray, "");

scatterDataSet.setScatterShape(ScatterChart.ScatterShape.CIRCLE);
                scatterDataSet.setScatterShapeSize(5);
                legend.setEnabled(false);
                description.setText("");
                scatterView.setDescription(description);

                scatterData.addDataSet(scatterDataSet);

                break;

            case "t":

                int transistorGain = 0;

                bytesToEntries();

                if (transistorType.equals("n")) {

                    transistorGain = (int) ((yPointB - yPointA) /
(xPointB - xPointA));
```

```
                } else if (transistorType.equals("p")) {

                    transistorGain = (int) ((-yPointB + yPointA) / (-
xPointB + xPointA));
                }

                scatterDataSet = new ScatterDataSet(entriesArray, "");

scatterDataSet.setScatterShape(ScatterChart.ScatterShape.CIRCLE);
                scatterDataSet.setScatterShapeSize(5);
                legend.setEnabled(false);
                description.setText("Ganho = ~" + transistorGain);
                scatterView.setDescription(description);

                scatterData.addDataSet(scatterDataSet);

                break;

            case "s":

                int earlyVoltage = 0;

                if (transistorType.equals("n") ||
transistorType.equals("p")) {

                    for (int i = 1; i <= 5; i++) {

                        switch (i) {
                            case 1:
                                exportData.append("VBE =
1V").append("\n");
                                break;
                            case 2:
                                exportData.append("\n").append("VBE =
2V").append("\n");
                                break;
                            case 3:
                                exportData.append("\n").append("VBE =
3V").append("\n");
                                break;
                            case 4:
                                exportData.append("\n").append("VBE =
4V").append("\n");
                                break;
                            case 5:
                                exportData.append("\n").append("VBE =
5V").append("\n");
                                break;
                        }

                        bytesToEntries();

                        switch (i) {
                            case 1:
                                scatterDataSet = new
ScatterDataSet(entriesArray, "VBE = 1V");

scatterDataSet.setColor(Color.MAGENTA);
```

```
                                break;
                            case 2:
                                scatterDataSet = new
ScatterDataSet(entriesArray, "VBE = 2V");
                                scatterDataSet.setColor(Color.BLUE);
                                break;
                            case 3:
                                scatterDataSet = new
ScatterDataSet(entriesArray, "VBE = 3V");
                                scatterDataSet.setColor(Color.GREEN);
                                break;
                            case 4:
                                scatterDataSet = new
ScatterDataSet(entriesArray, "VBE = 4V");
                                scatterDataSet.setColor(Color.YELLOW);
                                break;
                            case 5:
                                scatterDataSet = new
ScatterDataSet(entriesArray, "VBE = 5V");
                                scatterDataSet.setColor(Color.RED);
                                break;
                        }


scatterDataSet.setScatterShape(ScatterChart.ScatterShape.CIRCLE);
                        scatterDataSet.setScatterShapeSize(5);

                        scatterData.addDataSet(scatterDataSet);
                    }

                    float m = ((yPointB - yPointA) / (xPointB -
xPointA));

                    if (transistorType.equals("n")) {

                        earlyVoltage = (int) (yPointB/m - xPointB);

                    } else if (transistorType.equals("p")) {

                        earlyVoltage = (int) -(yPointB/m - xPointB);
                    }

                    description.setText("VA = ~" + earlyVoltage);
                    scatterView.setDescription(description);

                } else if (transistorType.equals("x") ||
transistorType.equals("y")) {

                    for (int i = 1; i <= 5; i++) {

                        switch (i) {
                            case 1:
                                exportData.append("VGS =
2V").append("\n");
                                break;
                            case 2:
                                exportData.append("\n").append("VGS =
2.25V").append("\n");
                                break;
```

```
                                case 3:
                                    exportData.append("\n").append("VGS =
2.5V").append("\n");
                                    break;
                                case 4:
                                    exportData.append("\n").append("VGS =
2.75V").append("\n");
                                    break;
                                case 5:
                                    exportData.append("\n").append("VGS =
3V").append("\n");
                                    break;
                            }

                            bytesToEntries();

                            switch (i) {
                                case 1:
                                    scatterDataSet = new
ScatterDataSet(entriesArray, "VGS=2V");

scatterDataSet.setColor(Color.MAGENTA);
                                    break;
                                case 2:
                                    scatterDataSet = new
ScatterDataSet(entriesArray, "VGS=2.25V");
                                    scatterDataSet.setColor(Color.BLUE);
                                    break;
                                case 3:
                                    scatterDataSet = new
ScatterDataSet(entriesArray, "VGS=2.5V");
                                    scatterDataSet.setColor(Color.GREEN);
                                    break;
                                case 4:
                                    scatterDataSet = new
ScatterDataSet(entriesArray, "VGS=2.75V");
                                    scatterDataSet.setColor(Color.YELLOW);
                                    break;
                                case 5:
                                    scatterDataSet = new
ScatterDataSet(entriesArray, "VGS=3V");
                                    scatterDataSet.setColor(Color.RED);
                                    break;
                            }


scatterDataSet.setScatterShape(ScatterChart.ScatterShape.CIRCLE);
                            scatterDataSet.setScatterShapeSize(5);

                            scatterData.addDataSet(scatterDataSet);
                        }

                        description.setText("");
                        scatterView.setDescription(description);
                    }

                    legend.setEnabled(true);

                    break;
```

```
        }

        float xMargin = 0;
        float yMargin = 0;

        if (transistorType.equals("n") || transistorType.equals("x"))
{ // NPN ou NMOS
            xMargin = (float) .1 * xPointMax;
            yMargin = (float) .1 * yPointMax;
        } else if (transistorType.equals("p") ||
transistorType.equals("y")) { // PNP ou PMOS
            xMargin = (float) -.1 * xPointMin;
            yMargin = (float) -.1 * yPointMin;
        }

        xAxis.setAxisMaximum(xPointMax + xMargin);
        xAxis.setAxisMinimum(xPointMin - xMargin);
        yAxis.setAxisMaximum(yPointMax + yMargin);
        yAxis.setAxisMinimum(yPointMin - yMargin);

        scatterView.setData(scatterData);
    }

    // Método para enviar dados pela porta série Bluetooth
    public void OutStreamWrite(String text) {

        try {

            outStream.write(text.getBytes());
            Log.d(TAG, "outStream: sent bytes: " + text);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

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


    @Override
    public void onDestroy() {
        super.onDestroy();
        unregisterReceiver(bondingBroadcast);

        bluetoothAdapter.cancelDiscovery();
    }

    // Cria um ficheiro txt com os dados recebidos e possibilita a sua
exportação
    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {

        Log.d(TAG, "menu: SAVE ICON selected");

        try {
```

```
            FileOutputStream fileOutputStream =
openFileOutput("CurveTracerData.csv", MODE_PRIVATE);
            fileOutputStream.write(exportData.toString().getBytes());
            fileOutputStream.close();

            Context context = getApplicationContext();
            File file = new File(getFilesDir(),
"CurveTracerData.csv");

            Uri path = FileProvider.getUriForFile(context,
"com.example.tese.fileprovider", file);

            Intent fileIntent = new Intent(Intent.ACTION_SEND);
            fileIntent.setType("text/csv");
            fileIntent.putExtra(Intent.EXTRA_SUBJECT, "Curve Tracer
Data");

fileIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
            fileIntent.putExtra(Intent.EXTRA_STREAM, path);

            startActivity(Intent.createChooser(fileIntent, "Save
File"));

        } catch (Exception e) {
            e.printStackTrace();
        }

        return super.onOptionsItemSelected(item);
    }

    // Escuta o sistema Android em busca de mensagens de alteração do
estado de emparelhamento com o dispositivo Bluetooth
    private final BroadcastReceiver bondingBroadcast = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            final String action = intent.getAction();


if(action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)) {

                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

                if (device.getBondState() ==
BluetoothDevice.BOND_BONDED) {
                    Log.d(TAG, "bondingBroadcast: BOND_BONDED");

                    connectToDevice(bluetoothDevice);
                }
                if (device.getBondState() ==
BluetoothDevice.BOND_BONDING) {
                    Log.d(TAG, "bondingBroadcast: BOND_BONDING");
                }
                if (device.getBondState() ==
BluetoothDevice.BOND_NONE) {
                    Log.d(TAG, "bondingBroadcast: BOND_NONE");
```

```
                }
            }
        }
    };

    // Converte a opção selecionada nos spinners na respetiva variável
de controlo
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view,
int i, long l) {

        int itemPosition = adapterView.getSelectedItemPosition();
        String item = adapterView.getItemAtPosition(i).toString();

        if (adapterView.getId() == R.id.curveSpinner) {

            if (itemPosition == 0) { // Curva de entrada
                curveType = "e";
            } else if (itemPosition == 1) { // Curva de transferência
                curveType = "t";
            } else if (itemPosition == 2) { // Curva de saída
                curveType = "s";
            }
            Log.d(TAG, "curveSpinner.OnItemSelected: itemPosition: " +
itemPosition +": item: " + item + ": state: " + curveType);

        } else if (adapterView.getId() == R.id.typeSpinner) {

            if (itemPosition == 0) { // BJT NPN
                transistorType = "n";
            } else if (itemPosition == 1) { // BJT PNP
                transistorType = "p";
            } else if (itemPosition == 2) { // MOSFET N
                transistorType = "x";
                curveSpinner.setSelection(2);
                curveType = "s";
            } else if (itemPosition == 3) { // MOSFET P
                transistorType = "y";
                curveSpinner.setSelection(2);
                curveType = "s";
            }

            Log.d(TAG, "typeSpinner.OnItemSelected: itemPosition: " +
itemPosition + ": item: " + item + ": state: " + transistorType);
        }
    }

    // Opções default dos spinners
    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {

        if (adapterView.getId() == R.id.curveSpinner) {

            curveType = "e";

        } else if (adapterView.getId() == R.id.typeSpinner) {

            transistorType = "n";
        }
```

```
    }
}
```

C.4. DeviceListAdapter.java

```
package com.example.tese;

import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import java.util.ArrayList;

public class DeviceListAdapter extends ArrayAdapter<BluetoothDevice> {

    private LayoutInflater layoutInflater;
    private ArrayList<BluetoothDevice> foundBTDevices;
    private int viewResourceID;

    public DeviceListAdapter(Context context, int tvResourceID,
ArrayList<BluetoothDevice> devices) {
        super(context, tvResourceID, devices);
        this.foundBTDevices = devices;
        layoutInflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        viewResourceID = tvResourceID;
    }

    public View getView(int position, View convertView, ViewGroup
parent) {
        convertView = layoutInflater.inflate(viewResourceID, null);

        BluetoothDevice device = foundBTDevices.get(position);

        if (device != null) {
            TextView deviceName =
convertView.findViewById(R.id.tvDeviceName);
            TextView deviceAddress =
convertView.findViewById(R.id.tvDeviceAddress);

            if (deviceName != null) {
                deviceName.setText(device.getName());
            }
            if (deviceAddress != null) {
                deviceAddress.setText(device.getAddress());
            }
        }

        return convertView;
```

```
    }
}
```

C.5. Strings.xml

```
<resources>

    <string name="app_name">CurveTracer</string>
    <string name="connectButton">REFRESH</string>
    <string name="startButton">START</string>

    <string-array name="curveSpinnerOptions" >
        <item>CARACTERÍSTICA DE ENTRADA</item>
        <item>CARACTERÍSTICA DE TRANSFERÊNCIA</item>
        <item>CARACTERÍSTICA DE SAÍDA</item>
    </string-array>

    <string-array name="typeSpinnerOptions" >
        <item>BJT NPN</item>
        <item>BJT PNP</item>
        <item>MOSFET N</item>
        <item>MOSFET P</item>
    </string-array>

</resources>
```

C.6. Provider_paths.xml

```
<?xml version="1.0" encoding="utf-8"?>

<paths>
    <files-path
        name="CurveTracerData.csv"
        path="."/>
</paths>
```

C.7. Activity_first.xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    tools:context=".FirstActivity">

    <ListView
        android:id="@+id/myListView"
        android:layout_width="match_parent"
        android:layout_height="550dp"
        android:layout_marginStart="32dp"
        android:layout_marginLeft="32dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="32dp"
        android:layout_marginRight="32dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/connectButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="16dp"
        android:text="@string/connectButton"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

C.8. Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/startButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="8dp"
        android:text="@string/startButton"
```

```
        android:textSize="12sp"
        app:layout_constraintBottom_toTopOf="@+id/progressBar"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent" />

    <Spinner
        android:id="@+id/curveSpinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:spinnerMode="dialog"
        app:layout_constraintBottom_toTopOf="@+id/startButton"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent" />

    <Spinner
        android:id="@+id/typeSpinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:spinnerMode="dialog"
        app:layout_constraintBottom_toTopOf="@+id/curveSpinner"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent" />

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="10dp"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="16dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent" />

    <com.github.mikephil.charting.charts.ScatterChart
        android:id="@+id/scatterView"
        android:layout_width="match_parent"
        android:layout_height="350dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="24dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/xLabel"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text=""
        android:textColor="@color/black"
        android:textSize="10sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/scatterView" />

    <TextView
        android:id="@+id/yLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:textColor="@color/black"
        android:textSize="10sp"
        app:layout_constraintStart_toStartOf="@+id/scatterView"
        app:layout_constraintTop_toTopOf="@+id/scatterView" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

C.9. Devicelistadapter_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myListView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/tvDeviceName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textSize="16sp" />

    <TextView
        android:id="@+id/tvDeviceAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textSize="10sp" />
</LinearLayout>
```

C.10. Dropdown_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>

<CheckedTextView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    style="?android:attr/spinnerDropDownItemStyle"
    android:singleLine="true"
    android:layout_width="match_parent"
    android:layout_height="?attr/dropdownListPreferredItemHeight"
    android:ellipsize="marquee"
    android:textSize="12sp"
    android:textAlignment="center"
    android:gravity="center" />
```

C.11. Spinner_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>

<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="12sp"
    android:textColor="#000000"
    android:padding="5dip"
    tools:ignore="RtlHardcoded" />
```

C.12. Sharebutton_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/saveIcon"
        android:title="SAVE"
        android:icon="@drawable/upload_icon"
        app:showAsAction="always" />

</menu>
```