



Sistemas Embebidos para Computação Física e IoT

Volume 1:
Introdução e Conceitos Básicos

RUI POLICARPO DUARTE

Projeto de Sistemas Embebidos para Computação Física e IoT

Volume 1: Introdução e Conceitos Básicos

Rui Polcarpo Duarte, PhD, Eng

24 de outubro de 2023

Copyright © 2023, Rui Polcarpo Duarte

Todos os direitos reservados. Sem a prévia autorização por escrito do autor, esta obra não pode ser reproduzida, no todo ou em parte, por meio de gravação ou por qualquer processo mecânico, fotográfico ou eletrónico, nem ser introduzida numa base de dados, difundida ou de qualquer forma copiada para uso público ou privado, além do uso legal como breve citação em artigos e críticas.

1ª Edição: outubro de 2023

ISBN: 978-989-33-5358-5



Imagem da capa: Foto da placa de desenvolvimento CoolRunner II por Rui Polcarpo Duarte

O Autor

Rui Policarpo Duarte é licenciado em Engenharia de Sistemas de Telecomunicações e Electrónica pelo Instituto Superior de Engenharia de Lisboa, mestre em Engenharia Eletrónica e de Computadores pelo Instituto Superior Técnico, e doutorado em Engenharia Elétrica e Eletrónica pelo Imperial College London. Atualmente leciona a disciplina de Sensores e Atuadores no Instituto Superior de Engenharia de Lisboa. No passado lecionou várias disciplinas relacionadas com Sistemas Embebidos, nomeadamente Arquiteturas de Computadores, Projeto de Sistemas Digitais, Eletrónica I, Análise de Circuitos, Redes de Computadores e Sistemas Operativos. Tem ainda participado em vários projetos de investigação nacionais e internacionais relacionados com sistemas embebidos e reconfiguráveis.

Agradecimentos

O autor agradece a ajuda dos Professores Alberto Cunha (IST) e Carlos Carvalho (ISEL) na discussão e revisão de vários assuntos apresentados neste livro.

Prefácio

OS avanços recentes na tecnologia de semicondutores têm permitido inovar em diferentes âmbitos de aplicação da computação para uso pessoal, e na automação de atividades domésticas, científicas e industriais. A maioria destas aplicações envolve a interação com o espaço envolvente, através da sensorização do meio e controlando atuadores que o alteram. A disponibilidade de sistemas cada vez mais pequenos, complexos e com maior capacidade de processamento de algoritmos complexos levam ao desenvolvimento de novos sistemas com maior autonomia, precisão, e inteligência.

O elevado nível de integração existente nalguns componentes electrónicos atuais e a existência de plataformas de desenvolvimento (hardware/software) fáceis de usar permite iniciar o desenvolvimento de sistemas embebidos usando poucos recursos e uma abordagem de baixa complexidade para que sejam fácil de entender de uma forma abrangente.

Para realizar algumas destas ideias, é necessário não só ter conhecimentos a nível da ligação dos diferentes tipos de componentes, como também a sua programação.

Este livro introduz diversos conceitos base e apresenta métodos usados habitualmente em projeto de sistemas embebidos. O público alvo são os alunos do ensino secundário via tecnológica, ensino superior das áreas de Engenharia, e outros profissionais que tenham alguns conhecimentos de eletrónica e programação em C/C++/Java.

Este livro começa por introduzir conceitos base relacionados com a tecnologia e para depois mostrar como integrar diferentes tipos de módulos quer sejam sensores, atuadores, e interfaces, e a sua programação, como forma de criar outros sistemas mais complexos. Além disso, é fornecido o código-fonte para permitir a experimentação com um vasto conjunto de componentes, procurando facilitar a tarefa de criar um sistema embebido.

Este livro encontra-se organizado em 9 capítulos:

- Capítulo I: Introdução - Introduz os sistemas embebidos, como são caracterizados, as suas aplicações, e as ferramentas usadas no seu desenvolvimento, com focus na plataforma Arduino;
- Capítulo II: Conceitos Base - Apresenta conceitos base usados ao longo do livro sobre electrónica e programação, necessários para o entendi-

mento dos sistemas embebidos;

- Capítulo III: Interfaces - Introduz os tipos de interface mais habituais para ligação de componentes e módulos a sistemas embebidos, incluindo as ligações físicas e a sua programação. Este capítulo apresenta também sistemas de comunicações para comunicação em sistemas heterogéneos, e suporte para ligação à Internet;
- Capítulo IV: Sensores - Trata dos diferentes tipos de sensores mais comuns, destinados a obter medidas do meio físico ambiente e traduzi-los em sinais eléctricos para serem processados. Explicam-se os princípios de funcionamento, as respectivas ligações eléctricas à unidade de processamento e a sua programação;
- Capítulo V: Atuadores - Aborda os diferentes tipos de atuadores, para produção de diferentes manifestações físicas, as respectivas ligações eléctricas e a sua programação;
- Capítulo VI: Conceitos Avançados - Apresenta conceitos avançados empregues em sistemas com outros requisitos, nomeadamente tolerância a falhas, resposta em tempo real, controlo e métodos matemáticos;
- Capítulo VII: Projetos - Estudo e projecto de alguns sistemas embebidos para demonstração de conceitos apresentados anteriormente através da realização de algumas aplicações típicas.
- Capítulo VIII: Exercícios - Conjunto de desafios propostos para o leitor resolver.

Conteúdo

O Autor	iii
Agradecimentos	v
Resumo	vii
Lista de Figuras	xiii
Lista de Tabelas	xv
Lista de Programas Arduino	xvii
Nomenclatura	xix
1 Introdução	1
1.1 Sistemas Embebidos	1
1.2 Computação Ciber-Física	4
1.3 Internet das Coisas	5
1.4 Plataforma Arduino	7
1.4.1 Placas Arduino - Hardware	9
1.4.2 Shields Arduino	12
1.4.3 Ambiente de Desenvolvimento - Software	14
1.4.4 Instalação	15
1.5 Primeiro Sketch Arduino	17
1.6 Outros Exemplos	19
1.7 Ferramentas	19
1.7.1 Terminal de Texto	20
1.7.2 Traçador Gráfico	23
2 Conceitos Básicos	25
2.1 Grandezas Físicas	25
2.2 Eletrônica	26
2.2.1 Tensão, Resistência e Corrente	26
2.2.2 Sinais, Tempo, Frequência e Espectro	30
2.2.3 Analógico e Digital	31

2.2.4	Elementos de um Circuito	35
2.2.5	Componentes Eletrônicos	41
2.2.6	Desenvolvimento de Circuitos	54
2.2.7	Verificação de Circuitos	61
2.3	Programação	66
2.3.1	C/C++ e Arduino	66
2.3.2	Tipos e Estruturas de Dados	68
2.3.3	Funções Especiais Arduino	70
2.3.4	Programar para o Mundo Exterior	71
2.3.5	Rotinas de Atendimento de Interrupções	72
2.3.6	Bibliotecas Arduino	73
2.3.7	Avaliação de Desempenho e Recursos Usados	75
A	Anexos	91
A.1	Representações Numéricas	91
A.1.1	Binário	91
A.1.2	Hexadecimal	91
A.2	Tabela ASCII	92
A.3	Armazenamento de Informação	93
A.4	Códigos de Terminal ANSI	94
A.5	Convenções de Programação	96
A.6	Segurança e Boas Práticas	96
A.7	Referência Rápida de C++	99
A.8	Erros Frequentes	102
A.8.1	Erros em Arduino	102
A.8.2	Erros em Programação	103
A.8.3	Erros em Electrónica	105
A.9	Referências	106
A.10	Fabricantes de Placas de Circuito Impresso	106

Lista de Figuras

1.1	Exemplo de um sistema embebido miniaturizado para captura e transmissão de imagens médicas.	2
1.2	Diagrama de blocos de um sistema embebido, com componentes e periféricos mais comuns.	3
1.3	Ecossistema dos sistemas ciber-físicos.	4
1.4	Arquitetura base de um sistema embebido genérico com entradas para sensores e saídas para atuadores.	5
1.5	Ecossistema dos sistemas IoT.	6
1.6	Cota global dos tipos de aplicações em IoT.	7
1.7	Logotipo Arduino	7
1.8	Esquema de ligação de um Arduino Mega a um computador e um conjunto de LEDs.	8
1.9	Componentes de uma placa Arduino Uno R3	9
1.10	a) Arduino Uno, b) Genuino 101, c) Arduino Beetle, d) Arduino Nano, e) Arduino Lilypad, f) Arduino Mega 2560 e g) Arduino Due.	10
1.11	Placas STM32 Nucleo446 da ST Microelectronics (esquerda) e Chipkit uc32 da Trenz Electronics (direita) com conetores compatíveis com Arduino.	12
1.12	Placas FPGA Intel DE10-Nano (esquerda) e Xilinx Pynq-Z2 (direita).	12
1.13	a) Motor-Shield, b) Multi-Shield, c) WiFi-Shield, d) Ethernet-Shield, e) LCD Keypad-Shield, e f) TFT-Shield.	14
1.14	Compilação e programação do exemplo Blink para uma placa Arduino Uno.	16
1.15	Ligação de um Arduino Uno a um computador via cabo USB.	17
1.16	Configuração do porto de comunicações da placa Arduino Uno.	18
1.17	Botões do menu do ambiente Arduino para compilar o programa (esquerda) e programar a placa Arduino (direita).	19
1.18	Compilação e programação do exemplo Blink para uma placa Arduino Uno.	20
1.19	Serial Monitor	22
1.20	Demonstração da aplicação para cifrar uma mensagem introduzida pelo utilizador com a cifra de César.	23

1.21	Serial Plotter	24
2.1	Onda sinusoidal com variação de 1000 ciclos (1 kHz) por segundo.	31
2.2	Exemplo do espectro (em cima) e espectrograma (em baixo).	32
2.3	Sinal analógico (esquerda) e digital (direita).	33
2.4	Processador Intel Pentium MMX (ASIC), à esquerda, e um dispositivo reconfigurável CPU+FPGA, à direita.	35
2.5	Exemplo de um circuito básico com um gerador, carga, condutores, e aparelhos de medida.	36
2.6	Esquema de uma fonte de alimentação com transformador e retificador.	38
2.7	Fonte de alimentação de bancada regulável.	39
2.8	Esquema de alimentação de um Arduino por energia solar.	40
2.9	Circuitos de transmissão e recepção de energia sem fios.	40
2.10	Condutores e cabos de diferentes tipos para ligação entre sistemas embebidos e outros periféricos.	41
2.11	Aspetto de diferentes tipos de componentes eletrónicos.	43
2.12	Associação de resistências em série (esquerda) e em paralelo (direita).	43
2.13	Código de cores das resistências.	44
2.14	Caderno com amostras de todas as resistências de uma série de fabrico (esquerda). Detalhe das resistências de 68 Ω (direita).	45
2.15	Símbolo e aspeto de condensadores fixos, polarizados e variáveis.	46
2.16	Associação de condensadores em série (esquerda) e em paralelo (direita).	47
2.17	Vários tipos de bobinas.	48
2.18	Montagem de um diodo num circuito em sentido direto (esquerda) e inverso (direita).	49
2.19	Símbolo do díodo retificador, e exemplos de vários tipos de diodos com diversos encapsulamentos.	50
2.20	Símbolos elétricos dos diferentes tipos de transistores, da esquerda para a direita: BJT tipo-N, BJT tipo-P, MOSFET canal-N e MOSFET canal-P.	50
2.21	Primeiro processador da Intel de 1971, o 4004 funcionava com um barramento de 4 bits (esquerda), e um processador Core-i9 a 64 bits de 2019 (direita).	53
2.22	Arquitetura do microcontrolador ATmega328 da Atmel/Microchip.	53
2.23	Arquitetura do microcontrolador ATmega328 da Atmel/Microchip.	54

2.24	Antes das atuais placas de prototipagem os circuitos eram realizados em tábuas de pão (<i>breadboard</i>), à esquerda. Ao centro e à direita as atuais <i>breadboards</i> para prototipagem de circuitos eletrónicos, com as respetivas ligações internas.	55
2.25	Aplicação Fritzing em uso para o desenho de esquemas elétricos (esquerda) e PCBs (direita).	56
2.26	Exemplos de <i>footprints</i> SMD (não estão à escala).	57
2.27	Vista da face superior de uma PCB de duas camadas após fabricação.	58
2.28	Exemplo de uma placa de circuito impresso de 4 camadas. . .	59
2.29	Faces superior e inferior de uma placa de circuito impresso com demonstração da capacidade de fabricação, espaçamento e espessura das pistas e tamanho dos furos.	60
2.30	Estação de soldadura (esquerda) e rolo de solda (direita). . . .	61
2.31	Exemplo de soldadura (esquerda) e dessoldadura (direita) de um terminal de um componente numa PCB.	62
2.32	Multímetro de uso geral (esquerda) e multímetro para periféricos USB (direita).	63
2.33	Osciloscópio digital de 4 canais (esquerda), e ponta de prova de osciloscópio e respetivos acessórios (direita).	64
2.34	Circuito de adaptação do conector áudio/mic do telemóvel para ponta de osciloscópio.	65
2.35	Demonstração da aplicação Android para funcionar como um osciloscópio digital usando a entrada de microfone do telemóvel. . .	65
2.36	Esquema de ligações dos conectores <i>Shield</i> Arduíno.	72
2.37	Tempo despendido com as instruções de E/S.	78
2.38	Evolução da ocupação das zonas de <i>Heap</i> e <i>Stack</i> , até ser esgotada a memória dinâmica disponível.	82
2.39	Relação entre as várias identidades trigonométricas.	83
A.1	Vários exemplos de manga termo-retrátil de diferentes cores e tamanhos (esquerda), e exemplo de soldaduras protegidas com dois pedaços de manga termo-retrátil (direita).	98
A.2	Medida da resistência máxima de um potenciómetro com valor diferente do especificado ($10\text{ k}\Omega$).	99
A.3	Exemplo de organização das pastas de um projeto.	104

Lista de Tabelas

1.1	Placas Arduino e as suas características.	11
1.2	Descrição dos pinos de E/S disponíveis na maioria das placas Arduino e Shields.	13
1.3	Lista de exemplos que acompanham a instalação do ambiente de desenvolvimento Arduino.	21
2.1	Quadro-resumo das 7 unidades fundamentais do Sistema Internacional.	26
2.2	Quadro-resumo das grandezas físicas derivadas mais relevantes para os sistemas embebidos.	27
2.3	Prefixos mais usados na representação de quantidades de unidades SI.	28
2.4	Exemplo de codificação da informação relativa ao tipo de sangue com dois bits.	32
2.5	Resumo das tabelas de verdade das funções lógicas elementares NOT, AND, OR e XOR.	34
2.6	Lista dos componentes eletrônicos mais usados.	42
2.7	Tabela com a correspondência entre as cores das riscas das resistências e o seu valor.	44
2.8	Tipos básicos de dados.	85
2.9	Tipos de variáveis e o espaço que ocupam em memória	86
2.10	Lista de funções disponibilizadas na biblioteca Arduino. . . .	87
2.11	Funções matemáticas implementadas na biblioteca Arduino. .	88
2.12	Lista de funções de E/S disponibilizadas na biblioteca Arduino.	88
2.13	Lista de bibliotecas existentes na instalação do IDE Arduino. .	89
2.14	Relação entre os ângulos das diferentes identidades trigonométricas em relação aos valores do seno no primeiro quadrante.	89
2.15	Tempo e memória de programa necessários para o cálculo do seno dos ângulos entre 0° e 90°	89
A.1	Convensões de nomes mais comuns.	97

Lista de Programas

1.1	Código do exemplo de demonstração do Arduino “blink”	18
1.2	Programa de demonstração do terminal Arduino	21
2.1	Conteúdo do ficheiro <code>main.cpp</code> no IDE Arduino	67
2.2	Tipos de variáveis definidos pelo Arduino	68
2.3	Listagem do Sketch <code>varTypes</code> para mostrar no terminal de texto a ocupação em memória de cada tipo de variável	68
2.4	Sketch de demonstração da utilização de classes para definição do componente LED.	70
2.5	<code>interruptDemo.ino</code>	73
2.6	Exemplo de utilização das biblioteca gráfica Adafruit.	75
2.7	<code>timeCount.ino</code>	76
2.8	Sketch para medição dos tempos das operações de E/S.	77
2.9	Demonstração da corrupção do <i>Stack</i> num Arduino.	80
2.10	Resultado da compilação do Sketch para corrupção do Stack .	81
2.11	Resultado da execução do Sketch para corrupção do Stack . .	82
2.12	Demonstração da utilização de constantes em memória não volátil.	83
A.1	Demonstração de programação num Sketch Arduino	99

Nomenclatura

Acronimos e Abreviaturas

ALU	Arithmetic Logic Unit
ASAP	As Soon As Possible
ASIC	Application Specific Integrated Circuit
CAD	Computer-Aided Design
CPU	Central processing Unit
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FPGA	Field-Programable Gate Array
FSM	Finite State Machine
GPU	Graphics Processing Unit
HDL	Hardware Description Languages
I/O	Input/Output
LUT	Look-Up Table
MAC	Multiply Accumulate
MSb	Most Significant bit
NaN	Not a Number
RAM	Random Access Memory
ROM	Read Only Memory
VHDL	Very-high-speed integrated circuit Hardware Description Language

List of Symbols

$O(n)$ Complexity order n

1

Introdução

Antes de existirem sistemas embebidos, os sistemas electrónicos eram realizados com componentes electrónicos discretos, cada um com uma funcionalidade distinta e estática. Com a miniaturização oferecida pelas novas tecnologias de fabrico de semicondutores, os sistemas embebidos surgiram como o resultado de integração de vários componentes electrónicos numa única solução mais complexa e programável, onde o mesmo sistema pode servir para realizar aplicações diferentes.

Este capítulo dedica-se à apresentação dos sistemas embebidos¹, e dos conceitos mais importantes envolvidos na especificação, realização e programação de sistemas embebidos, bem como explicação do princípio de funcionamento dos componentes electrónicos mais comuns.

1.1 Sistemas Embebidos

Um sistema embebido, por definição, é um sistema computacional dedicado ao processamento de informação proveniente de sensores num determinado ambiente ou objeto, e sua modificação através de atuadores. Designam-se por sistemas embebidos uma vez que integram um sistema computacional com vários componentes, ou módulos, para realizar uma solução integrada.

Os sistemas embebidos habitualmente são sistemas de alta disponibilidade, baixo consumo e baixo custo, e podem ser usados para realizar aplicações

¹Atualmente, embora existam vários sinónimos válidos, não existe consenso nas comunidades académica e profissional sobre a nomenclatura a adotar em Português. A nomenclatura “sistemas embebidos” foi escolhida pelo autor por se aproximar do inglês “Embedded Systems”.

CAPÍTULO 1. INTRODUÇÃO

que vão desde o simples temporizador de ciclos de uma máquina de lavar até elementos de um sistema aviónico ou controladores de membros prostéticos estimulados por impulsos nervosos. A figura 1.1 mostra uma câmara miniatura para realização de exames de endoscopia parcialmente desmontada. Neste figura é visível o sensor de imagem, as pilhas com autonomia para alimentar o circuito durante 10 horas, a antena para transmitir a imagem recebida para um recetor na proximidade do paciente, e os 4 LEDs para iluminar a zona a ser filmada. A imagem é passada do sensor para o circuito transmissor 434 MHz, onde a antena se encontra acoplada, através de uma fita com vários condutores. À direita encontra-se um LED de 5 mm para servir de escala. Todo o sistema é do tamanho de um comprimido (32,3 mm comprimento e 11,6 mm de diâmetro) e pesa 2,9 gramas.

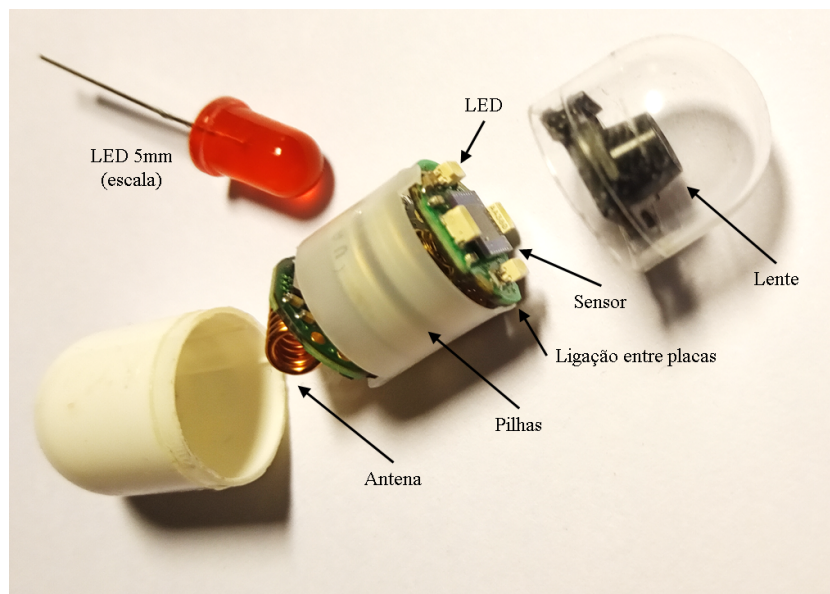


Figura 1.1: Exemplo de um sistema embebido miniaturizado para captura e transmissão de imagens médicas.

Tipicamente, o sistema embebido é baseado num Microcontrolador - *Micro-Controller Unit* (MCU) e em periféricos para suporte de funções específicas. O MCU é constituído por componentes que no passado eram componentes discretos que teriam de ser ligados numa placa de circuito impresso, nomeadamente:

- *Central Processing Unit* (CPU) - processador genérico. É o “cerebro” do sistema embebido e é responsável pelo processamento das instruções do programa residente em memória;
- *Random-Access Memory* (RAM) - memória para armazenar os dados a processar (variáveis), ex. *Static Random Access Memory* (SRAM) e *Dynamic Random Access Memory* (DRAM);

- *Read-Only Memory* (ROM) - memória que armazena o programa a executar (instruções), ex. FLASH e Electrically Erasable Programmable Read-Only Memory (EEPROM);
- *Analog-to-Digital Converter* (ADC) - unidade para conversão de sinais analógicos em digitais, usada para ligar sensores analógicos;
- *Digital to Analog Converter* (DAC) - unidade para conversão de sinais digitais em analógicos para ligar atuadores análogos;
- unidades para interligar e outros sub(sistemas) integram vários controladores de barramentos, por exemplo I2C, CAN, e SPI;
- unidades dedicadas a realizar funções em *hardware* que não são triviais em *software*, nomeadamente o temporizadores (*Timers*) para contagem precisa de tempo, e controlador de acesso direto à memória (*Direct Memory Access* (DMA)) para acelerar transferência de dados dos periféricos para a memória.

A figura 1.2 ilustra o diagrama de blocos de um sistema embebido genérico, composto pelos blocos habitualmente integrados num MCU, nomeadamente o CPU, DMA, RAM, ADC e DAC. Mais adiante será mostrado como os diversos componente funcionam e são usados.

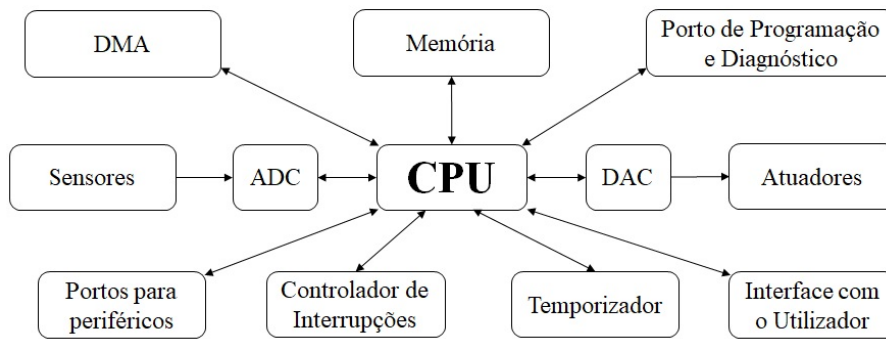


Figura 1.2: Diagrama de blocos de um sistema embebido, com componentes e periféricos mais comuns.

A escolha da plataforma para a realização de um sistema embebido depende dos seguintes parâmetros:

- energia consumida (e autonomia): se o sistema for alimentado a pilhas ou bateria, quando mais energia consumir, menor será o tempo da sua autonomia;
- espaço de memória de programa e dados: o sistema deve ter memória suficiente para ser capaz de reter os dados a processar, bem como o programa com os algoritmos a executar;

- periféricos e interfaces: o número de pinos de Entrada/Saída (E/S) a ligação dos componentes, ou sub-sistemas externos;
- velocidade de funcionamento, ou frequência de relógio do CPU: limita o tempo de execução dos algoritmos. Um CPU que funcione a uma frequência elevada demora menos tempo a executar o mesmo número de instruções, mas tem maior consumo de energia;
- facilidade de utilização: importa conseguir utilizar o sistema para realizar a solução pretendida sem ser necessário um conhecimento profundo sobre a matéria.

1.2 Computação Ciber-Física

A noção de computação ciber-física integra computação e processos físicos. Este tipo de sistemas pode ser visto como um ecossistema onde sistemas embebidos controlam os processos físicos usando atuadores, com base em processamento realizado sobre dados obtidos através de sensores.

A computação física é uma evolução do sistema embebido, fornecendo uma visão holística entre os processos físicos, comunicações, programas e circuitos electrónicos. A figura 1.3 mostra exemplos de alguns domínios de aplicação, nomeadamente: saúde (implantes, robótica para cirurgia remota e administração de medicamentos autónoma), transportes (gestão de frotas e optimização de rotas) e infraestruturas (monitorização em tempo-real de consumo e distribuição de energia, edifícios inteligentes).

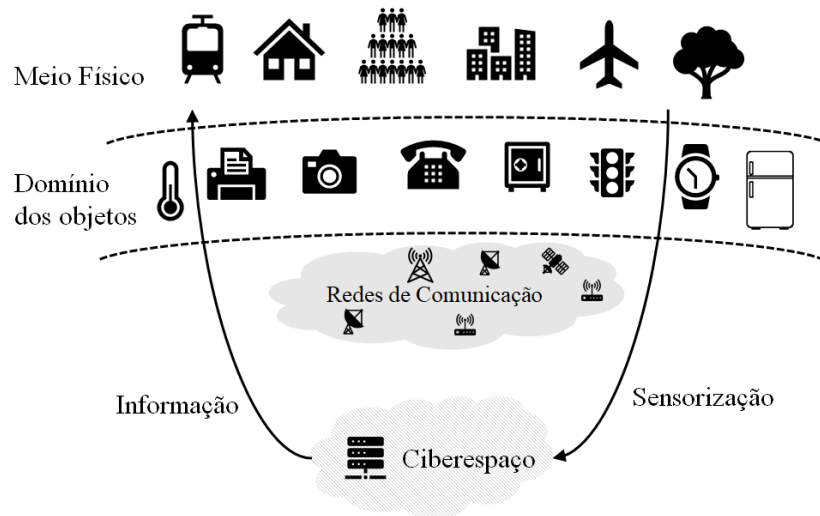


Figura 1.3: Ecossistema dos sistemas ciber-físicos.

Um dos grandes desafios da computação física passa pela modelação do sistema a realizar tendo em conta as limitações físicas e computacionais,

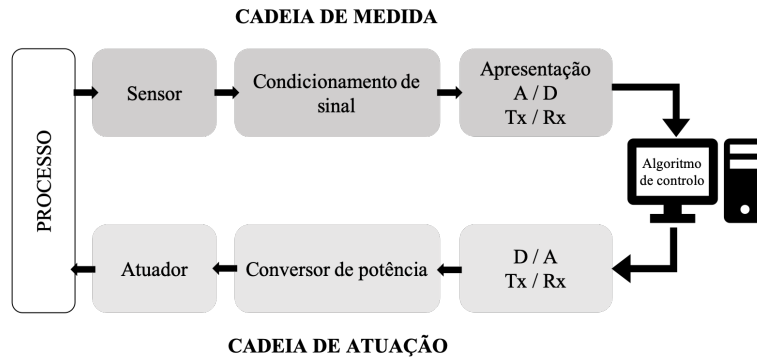


Figura 1.4: Arquitetura base de um sistema embebido genérico com entradas para sensores e saídas para atuadores.

uma vez que os resultados do processamento dependem do tipo de sensores usados e do tempo de processamento do sistema. Por exemplo, um sistema com pouca capacidade de processamento demorará mais tempo a produzir um resultado, ou a reagir a variações medidas nos sensores, pelo que não pode ser usado em sistemas que requerem atuação rápida, como condução autónoma. Da mesma forma, um sistema com sensores com pouca precisão nunca vai conseguir atingir a precisão necessária em sistemas produção de peças para aviões, análise de amostras médicas ou auxílio em cirurgia.

1.3 Internet das Coisas

A expressão Internet das Coisas deriva de Internet das Coisas - *Internet of Things* (IoT)² e serve para designar sistemas embebidos que foram pensados para estarem permanentemente ligados à Internet e fornecerem serviços Web, páginas da Internet e acesso a bases de dados distribuídas.

Aliando os sistemas embebidos às comunicações, e com a facilidade de se obter ligação à Internet, foi possível criar dispositivos electrónicos capazes de realizar diferentes funções com base em serviços facilmente disponibilizados pela Internet.

Nesta área existem diversos ramos de investigação em constante expansão, nomeadamente:

- arquiteturas de computadores de baixo consumo e com tolerância a falhas,
- sensores de baixo consumo e alta precisão,

²É discutível se a expressão “Coisas para a Internet” (*Things for the Internet*) seria mais adequada do que “Internet das Coisas”

CAPÍTULO 1. INTRODUÇÃO

- sistemas operativos de tempo-real, fiáveis e com baixo consumo de energia.

A figura 1.5 mostra alguns tipos de aplicações que se enquadram no panorama IoT.

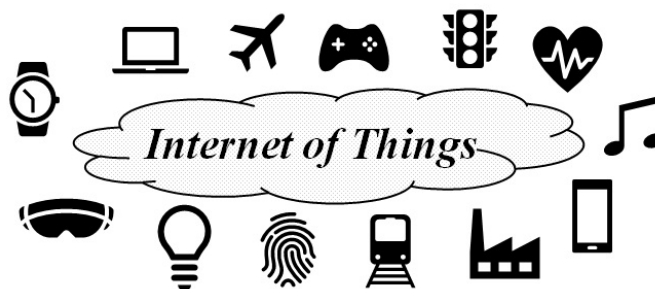


Figura 1.5: Ecossistema dos sistemas IoT.

As aplicações IoT mais frequentes são destinadas à quantificação do espaço e indivíduos:

- Domésticas: video-vigilância, automação e monitorização de espaços para casas inteligentes, assistência a idosos e incapacitados;
- Industriais: automação e monitorização de atividades de produção de bens e agricultura;
- Comerciais: gestão de meios de transporte, monitorização da saúde de pessoas na realização de tarefas laborais, e registo de indicadores de bem-estar;
- Infraestruturas: monitorização e gestão de consumo energético e monitorização do ambiente.

A figura 1.6 mostra a taxa global de projects nos diferentes segmentos IoT, num estudo realizado em 2016³.

Os sistemas IoT são simples, limitados e projetado à medida dos requisitos mínimos de funcionamento, e por isso não possuem capacidade de processamento para tratamento dos dados adquiridos, enviado os dados pela Internet para servidores que agregam os dados gerados por vários sensores.

O IoT está ligado a outras áreas, tais como *Big Data* e segurança informática. A *Big Data* está dedicada ao tratamento de grandes volumes de dados para providenciar uma melhor compreensão sobre um determinado espaço e população, que não seria fácil, ou possível, de obter de outra forma.

³<https://iot-analytics.com/>

Segmento IoT	Cota [%]	Segmento IoT	Cota [%]
 Indústria conectada	22	 Agricultura inteligente	6
 Cidades inteligentes	20	 Edifícios conectados	5
 Energia inteligente	13	 Saúde conectada	5
 Carros conectados	13	 Retalho inteligente	4
 Outros	8	 Cadeias de abastecimento inteligentes	4

Figura 1.6: Cota global dos tipos de aplicações em IoT.

Ao transferir dados pela Internet é necessário ter em atenção a segurança dos dados pois muitas vezes os sistemas embebidos não possuem as atualizações necessárias para garantir que o sistema oferece a melhor segurança e privacidade. Sistemas como *Smart Watches* e dispensadores automáticos de medicação requerem um nível de segurança e privacidade que a maioria dos sistemas embebidos ligados à Internet não consegue oferecer.

1.4 Plataforma Arduino

O Arduino é uma família de sistemas embebidos mais adotados atualmente, e é indicado para iniciar a aprendizagem em sistemas embebidos, uma vez que oferece um conjunto de sistemas de desenvolvimento acessíveis, com baixo custo, de fácil programação e personalização, e permitem inter-operabilidade com vários sistemas. A figura 1.7 mostra o logotipo que identifica os componentes da família Arduino.



Figura 1.7: Logotipo Arduino

Um Arduino é baseado no microcontrolador que é ligado a outros componentes através de pinos de E/S existentes na sua placa de circuitos via fios ou placas de expansão (*shields*). Um Arduino é programado através de um computador, onde são escritos programas (conhecidos como “Rascunhos” derivado do Inglês “Sketch”) em linguagem C/C++, contendo as instruções a executar. Diferentes placas Arduino possuem diferentes MCUs, sendo o AVR da Microchip (herdado da Atmel) o mais usado.

CAPÍTULO 1. INTRODUÇÃO

A plataforma Arduino tornou-se bastante atrativa devido a 5 aspectos fundamentais:

1. Normalização do suporte físico - as placas de desenvolvimento oferecem as mesmas ligações para expansão do sistema sempre na mesma localização. A vantagem é fazer o desenvolvimento de apenas uma placa de expansão que é compatível com várias placas Arduino;
2. Ambiente de desenvolvimento integrado - o mesmo ambiente serve para escrever o programa a executar no sistema embebido com suporte para as várias placas Arduino. Permite ainda a reutilização de (partes de) programas e bibliotecas mesmo em placas com características diferentes;
3. Domínio de utilização público (*open-source*) - o seu uso não está obrigado ao pagamento de licenças (*royalties*);
4. Abstração dos detalhes habituais na programação de MCUs - ao assumir algumas opções e valores por omissão, e ocultar alguns detalhes está a facilitar o desenvolvimento por parte de leigos em sistemas embebidos. A desvantagem é a perda de algumas funcionalidades específicas ou eficiência;
5. Reprogramação - um Arduino pode ser usado para a realização de vários sistemas diferentes. A possibilidade de reprogramar o sistema permite a correcção de erros ou adicionar funcionalidades no seu programa *a posteriori*.

A figura 1.8 ilustra a ligação de Arduino Mega a um computador via USB para fornecer alimentação e programação, e também a um LED cujo funcionamento é descrito no programa transferido para o Arduino.

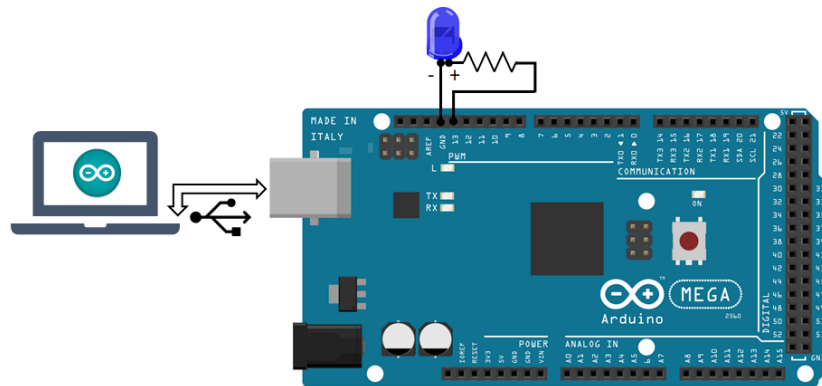


Figura 1.8: Esquema de ligação de um Arduino Mega a um computador e um conjunto de LEDs.

1.4.1 Placas Arduino - Hardware

Existem diversas placas da família Arduino, usadas para fins diferentes, dependendo dos requisitos do sistema. A principal diferença entre as diferentes placas é o MCU, o que se traduz em circuitos com diferentes tamanhos, números de pinos de E/S, velocidades de processamento e periféricos de comunicação.

Das diferentes placas, o Arduino Uno é a placa mais comum. A figura 1.9 mostra o esquema de um Arduino Uno, e os seus componentes. A placa possui no centro um MCU que é programado através da interface USB. O MCU depois de programado é capaz de controlar quaisquer periféricos que sejam ligados nos 4 blocos de conectores de E/S, localizados na extremidade da placa (em cima e em baixo). A placa possui também uma ficha de alimentação, para que possa funcionar de forma autónoma sem ser necessário estar ligado a um computador. Os conetores ICSP servem para programar diretamente o MCU e o controlador de comunicação USB, mas não necessitam de ser usados normalmente. Existe um botão de RESET na placa que permite reiniciar o programa armazenado no MCU.

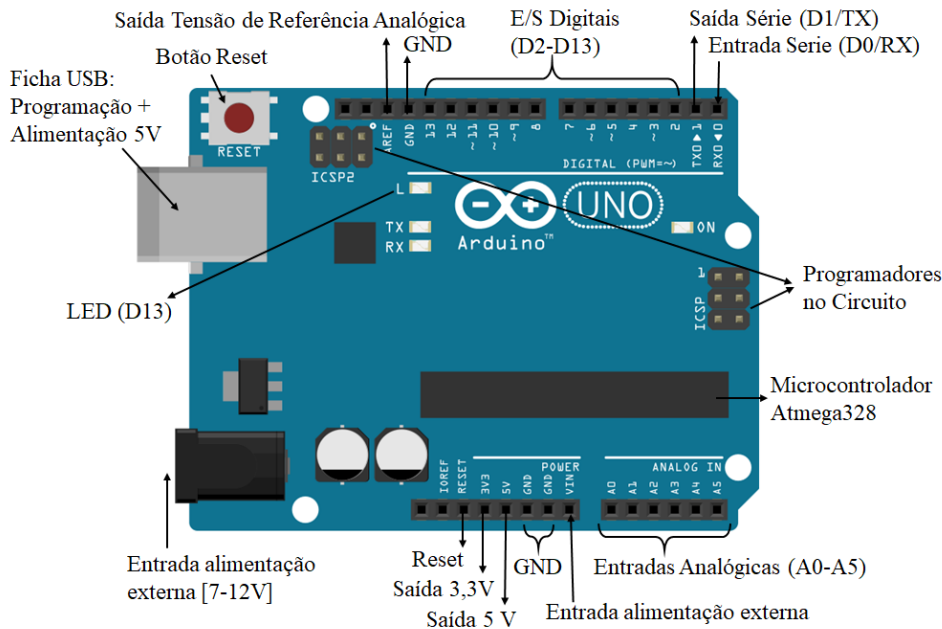


Figura 1.9: Componentes de uma placa Arduino Uno R3

A figura 1.10 apresenta vários sistemas de desenvolvimento da família Arduino, e compatíveis, nomeadamente: a) Uno, b) Genuino 101, c) Beetle/Leonardo, d) Nano, e) Lilypad, f) Mega e g) Due. De salientar que a localização dos conectores de alimentação, comunicação USB, e os conectores para placas de expansão são idênticos em várias placas. A tabela 1.1 mostra

CAPÍTULO 1. INTRODUÇÃO

algumas das principais características das placas Arduino.

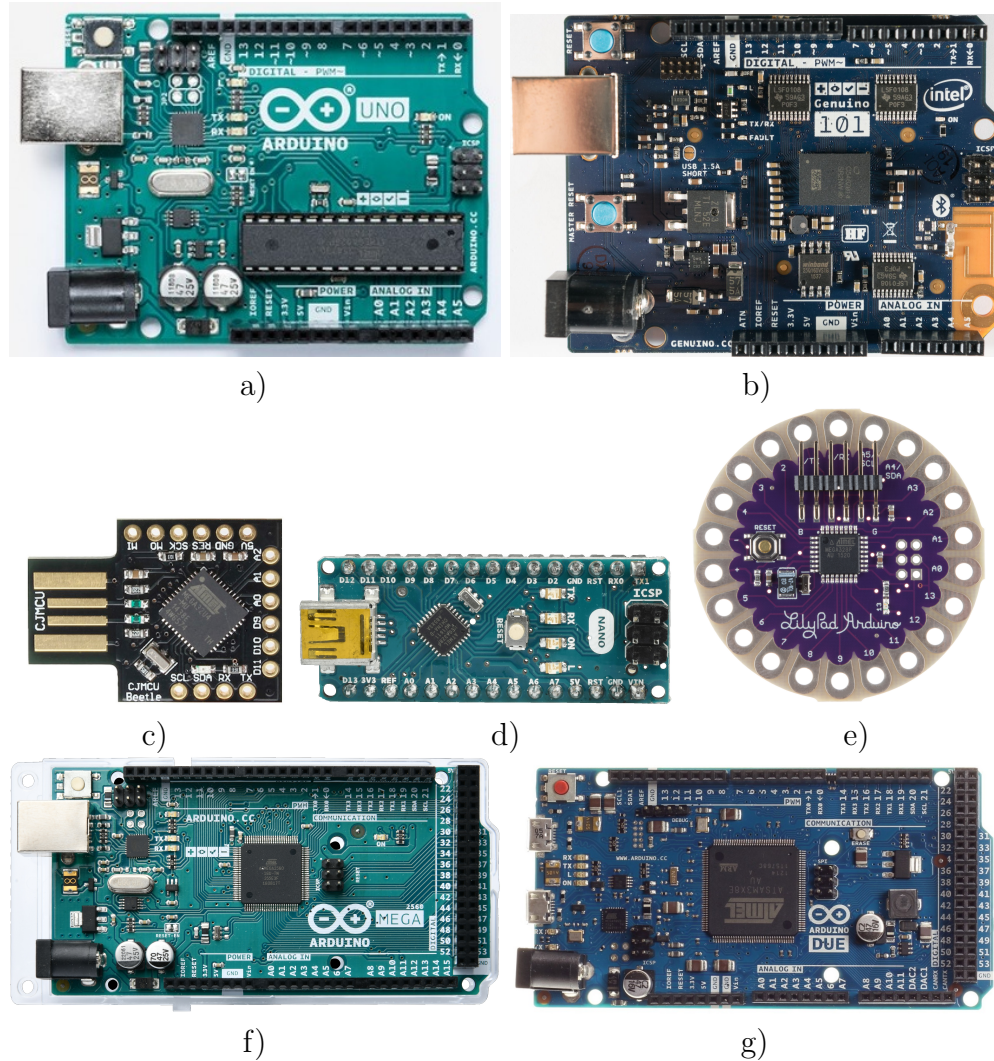


Figura 1.10: a) Arduino Uno, b) Genuino 101, c) Arduino Beetle, d) Arduino Nano, e) Arduino Lilypad, f) Arduino Mega 2560 e g) Arduino Due.

O MCUs dos Arduínos vêm pré-programados de fábrica com um *bootloader*, que faz com que o ambiente de desenvolvimento consiga detetar a sua presença, e transferir o programa compilado para a sua memória. As placas compatíveis Arduino possuem um conector *In-Circuit Serial Programming* (ICSP) para se poder programar o *bootloader* no MCU de uma nova placa.

Para além das placas Arduino “originais” existem também sistemas compatíveis, que são capazes de funcionar com outras placas e *Shields* Arduino. Alguns destes sistemas são baseados em MCUs de diversos fabricantes, ou tecnologia *Field-Programmable Gate Array* (FPGA)⁴, capaz de atingir níveis

⁴são dispositivos lógicos reconfiguráveis usados para a realização de sistemas digitais

1.4. PLATAFORMA ARDUÍNO

Arduíno	MCU	Observações
Uno	ATmega328P (CLK 16 MHz, SRAM 1 kB, FLASH 32 kB, 14 E/S digitais e 6 entradas analógicas)	normalmente utilizado como sistema de iniciação pois oferece uma boa relação entre desempenho, número de pinos E/S, e inúmeros exemplos disponíveis
101	Intel Curie (CLK 32 MHz, SRAM 24 kB, FLASH 196 kB, 14 E/S digitais e 6 entradas analógicas)	placa semelhante ao Uno mas com suporte para WiFi
Beetle / Leonardo	Atmega32U4: CLK 16 MHz, SRAM 2,5 kB, FLASH 32 kB, 20 E/S digitais e 12 entradas analógicas	o MCU permite ser programado como periférico USB, podendo emular um teclado ou rato USB
Nano	ATmega328: (CLK 16 MHz, SRAM 1 kB, FLASH 16 kB, 14 E/S digitais e 8 entradas analógicas)	placa com dimensões reduzidas, e menos pinos disponíveis para ligar periféricos
Lilypad	ATmega168V ou ATmega328V (CLK 8 MHz, SRAM 1kB, FLASH 16 kB,)	placa para sistemas vestíveis, onde os terminais são ilhas maiores para permitir coser à roupa com linha condutora
Mega	ATmega2560 (CLK 16 MHz, SRAM 8 kB, FLASH 256 kB, 54 E/S digitais e 16 entradas analógicas)	Placa maior, com mais pinos de E/S e 4 UARTs
Due	ATSAM3X8E (CLK 84 MHz, SRAM 96 kB, FLASH 512 kB, 54 E/S digitais e 12 entradas analógicas)	Placa com as mesmas dimensões que o Arduíno Mega e as 4 UARTs, mas com o um processador mais rápido

Tabela 1.1: Placas Arduíno e as suas caraterísticas.

CAPÍTULO 1. INTRODUÇÃO

de desempenho muito superiores aos MCUs habituais nos Arduínos. A figura 1.11 mostra dois exemplos de sistemas de desenvolvimento de MCUs da ST (esquerda) e Microchip (direita). A figura 1.12 mostra duas placas com FPGAs da Altera/Intel (esquerda) e Xilinx (direita). Em todas as placas é fácil reconhecer o conjunto de conectores de expansão Arduino. Uma placa que não exista na lista de placas do ambiente Arduino poderá ser adicionada posteriormente através do utilitário gestor de placas.

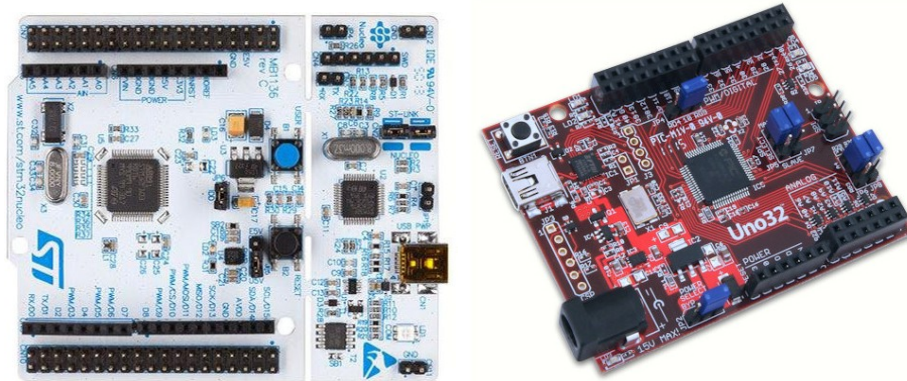


Figura 1.11: Placas STM32 Nucleo446 da ST Microelectronics (esquerda) e Chipkit uc32 da Trenz Electronics (direita) com conectores compatíveis com Arduino.

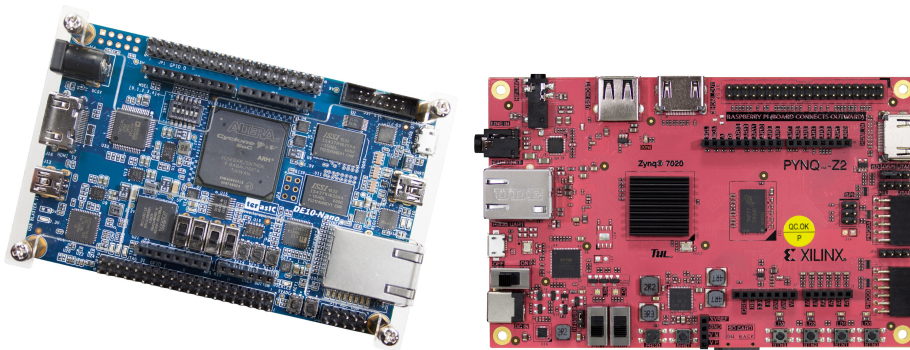


Figura 1.12: Placas FPGA Intel DE10-Nano (esquerda) e Xilinx Pynq-Z2 (direita).

1.4.2 Shields Arduino

As placas de expansão para os Arduínos são conhecidas como *Shields* (escudos) e servem para dotar um Arduino de funcionalidade que não lhe é nativa.

altamente paralelizáveis e especializados usados na aceleração de algoritmos, por exemplo redes neuronais e bitcoins. Alguns dispositivos FPGA incorporam um processador ARM dual-core

Os *Shields* foram pensados para serem empilhados sobre os conectores laterais, tendo as ligações normalizadas para efeitos de compatibilidade, tal como ilustrado na figura 1.13. Esta normalização permite criar placas de expansão para uma determinada finalidade, mas que podem ser utilizadas em noutros Arduínos. A única condição é que tanto a placa como o *Shield* Arduino tenham os mesmos pinos disponíveis. A tabela descreve a funcionalidade dos pinos existentes num Arduino Uno.

Pino	Descrição	Pino	Descrição
Reset	re-iniciação do Arduino	D0 / RX	Pino de entrada e saída digital também usado para recepção de dados do porto série assíncrono
3.3V	Tensão de alimentação gerada internamente pelo Arduino	D1 / TX	Pino de entrada e saída digital também usado para transmissão de dados do porto série assíncrono
5V	Tensão de alimentação gerada internamente pelo Arduino	D2-13	Pinos de entrada e saída digitais
GND	Referência ou “massa” da alimentação	PWM (~)	Pinos marcados com ~ podem ser usados para geração de sinais reguláveis no tempo via pinos digitais

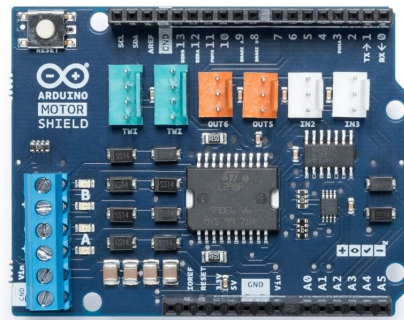
Tabela 1.2: Descrição dos pinos de E/S disponíveis na maioria das placas Arduino e Shields.

A vantagem da utilização de Shields é ter circuitos de baixa/média complexidade montados e testados, poupando o tempo e o esforço a de realizar e testar várias ligações. A figura 1.13 apresenta várias placas de expansão que podem ser empilhadas em placas compatíveis com Arduino Uno:

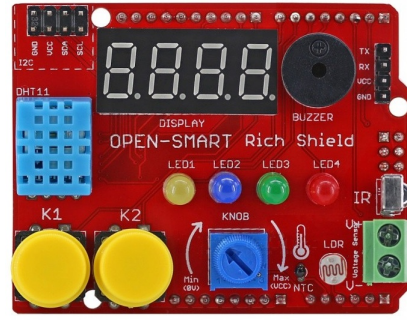
- a) Motor-Shield - Placa de adaptação para ligação de motores elétricos;
- b) Multi-Shield - Placa com vários sensores e atuadores, nomeadamente: temperatura e humidade, luminosidade, posição, botões de pressão, LEDs, mostrador de 7 segmentos, *beeper* e receptor de infravermelhos;
- c) WiFi-Shield - Placa que permite ligação a uma rede sem fios WiFi.
- d) Ethernet-Shield - Placa que permite ligação a uma rede com fios do tipo Ethernet;

CAPÍTULO 1. INTRODUÇÃO

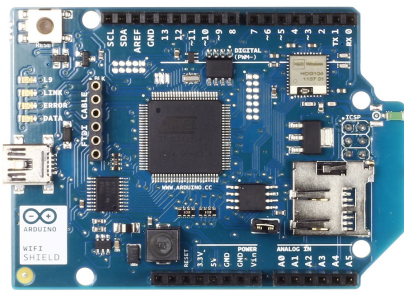
- e) LCD Keypad-Shield - Placa com um mostrador alfanumérico de duas linhas e 16 caracteres por linha e 6 botões de pressão;
- f) TFT-Shield - Placa que incorpora um ecrã gráfico com sensor tátil;



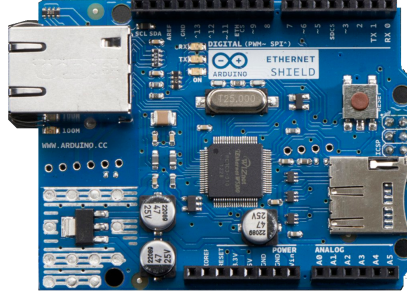
a)



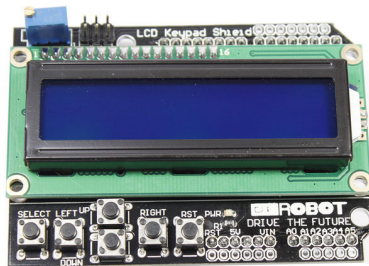
b)



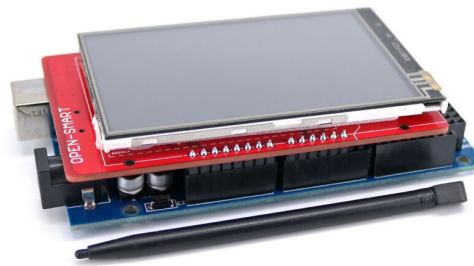
c)



d)



e)



f)

Figura 1.13: a) Motor-Shield, b) Multi-Shield, c) WiFi-Shield, d) Ethernet-Shield, e) LCD Keypad-Shield, e f) TFT-Shield.

1.4.3 Ambiente de Desenvolvimento - Software

Para que uma plataforma genérica cumpra a funcionalidade para que foi pensada é necessário que seja programada. Os programas escritos para os Arduínos são conhecidos como *Sketch* (esboço). O programa, ou código-fonte, é escrito numa linguagem de programação (C++ no Arduino) e compilado

antes de ser usado para programar o MCU. Estas funções são suportadas através de um ambiente de desenvolvimento.

Os ambientes de desenvolvimento são também conhecidos como IDE (*Integrated Development Environment* - ambiente de desenvolvimento integrado), e numa única aplicação permitem escrever o programa, usar bibliotecas pré-instaladas, compilar, executar na plataforma alvo, observar a saída do sistema, e depurar erros existentes.

O Arduino oferece um ambiente de programação simplificado, em relação ao de outros sistemas MCUs, mas igual para as diferentes placas. Este ambiente possui uma interface minimalista, que abstrai muitos detalhes oferecidos pelas ferramentas tradicionais de programação de MCUs. Este ambiente de desenvolvimento oferece a vantagem de se poder usar o mesmo programa C/C++ para programar MCUs de famílias diferentes.

O IDE do Arduino oferece as seguintes funcionalidades:

1. Editor - para escrita de programas;
2. Compilador - para tradução das instruções do programa em códigos interpretáveis pelo MCU;
3. Programador - para programação do MCU com o programa compilado;
4. Ferramentas - para depuração e visualização dos dados enviados pelo Arduino para o computador.

Para facilitar a programação de periféricos, *Shields* e funções complexas, existem bibliotecas pré-instaladas para a programação de funções complexas, por exemplo funções trigonométricas, e configuração de componentes habitualmente usados em *Shields*, nomeadamente mostradores Liquid Crystal Display (LCD), controladores de rede e motores. A maioria das bibliotecas inclui exemplos de demonstração através de *Sketches* prontos a programar.

1.4.4 Instalação

O ambiente de desenvolvimento (IDE) Arduino está disponível para os sistemas operativos Windows, Linux, e MacOS e pode ser obtido na página Web oficial, em: <https://www.arduino.cc/en/Main/Software>. Ainda sem a placa Arduino ligada ao computador, deve-se executar o programa de instalação, usando todas as opções por omissão (*next*→*next*...). Os *drivers* “Adafruit” e “Arduino” devem ser também instalados. O programa deverá ficar instalado na pasta: C:/Program Files (x86)/Arduino. Para colocar em execução o IDE Arduino deve-se clicar no símbolo do Arduino no ambiente de trabalho do computador, ou procurando pela aplicação com o nome “Arduino”.

CAPÍTULO 1. INTRODUÇÃO

Quando em execução, a janela principal do IDE mostra um conjunto de menus e botões no topo, uma área de edição de texto dedicada à escrita de programas, e uma área de apresentação de mensagens no fundo a preto, ilustrado na figura 1.18.

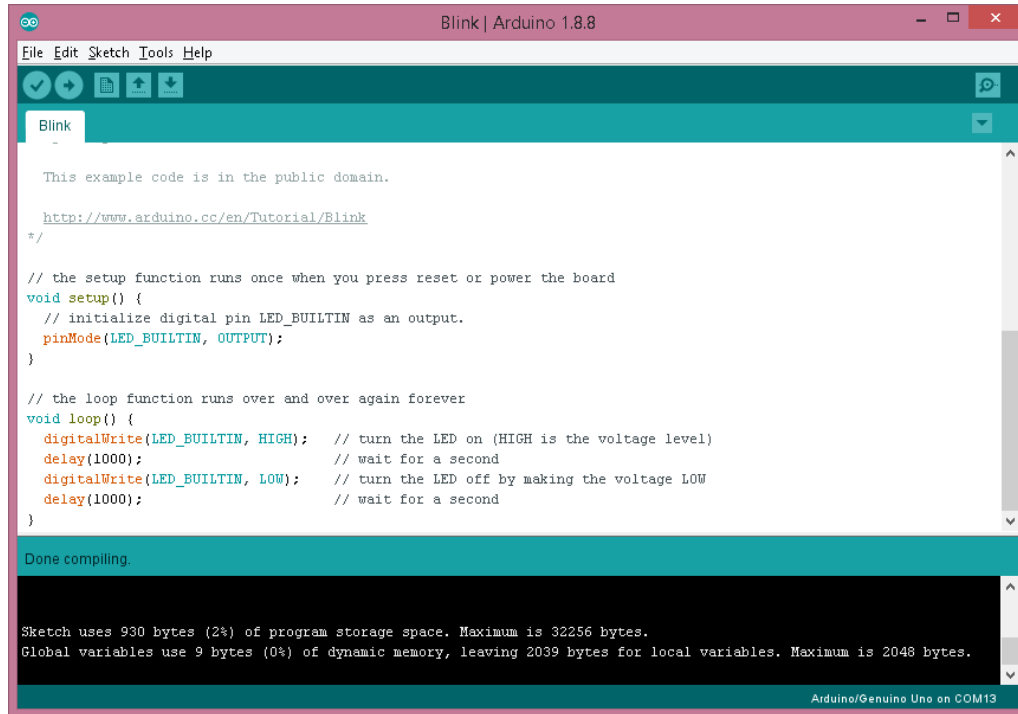


Figura 1.14: Compilação e programação do exemplo Blink para uma placa Arduino Uno.

Para testar que o IDE e a placa Arduino funcionam em pleno deve-se ligar o Arduino ao computador. A figura 1.15 ilustra a ligação de um Arduino Uno, sendo o processo semelhante para os outros sistemas Arduino:

1. Ligar uma extremidade do cabo USB a um Arduino (ex. Uno);
2. Ligar a outra extremidade do cabo USB a uma ficha USB do computador.
3. O LED verde da placa deve acender, e o LED amarelo poderá piscar momentaneamente.

No caso da placa Arduino Due, que possui duas fichas USB-micro, deve-se ligar o conector à ficha próxima da ficha de alimentação externa e seleccionar o *Programming Port*. Antes de programar a placa deve configurar o porto de comunicação entre o Arduino e o computador:

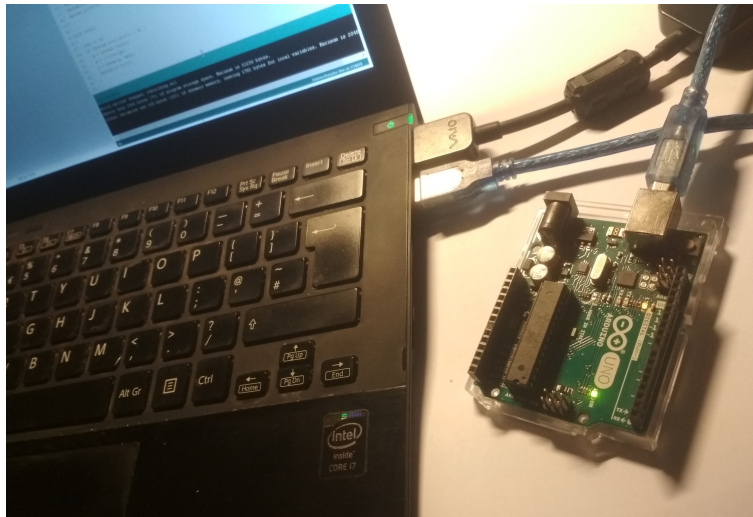


Figura 1.15: Ligação de um Arduino Uno a um computador via cabo USB.

1. Configurar qual a placa Arduino em uso - Menu: **Tools**→**Board**→**Arduino/Genuino Uno**;
2. Configurar qual o porto de comunicação - Menu: **Tools**→**Port**→**COM13 (Arduino/Genuino Uno)**;

A figura 1.16 ilustra a configuração da placa e do seu porto. Se apenas existir uma placa original Arduino ligada, é provável que o ambiente de desenvolvimento a identifique ou detete automaticamente.

O identificador do porto varia entre placas, computadores e sistemas operativos, pelo que pode ser necessário identificar o porto adequado. No caso de existirem vários sistemas ligados por portos USB, pode ser mais fácil desligá-los momentaneamente para descobrir qual o porto do Arduino. No sistema operativo Windows, o “Gestor de Dispositivos” mostra quais os portos disponíveis (COM*).

É possível escolher o idioma do IDE Arduino no menu das preferências bem como ajustar o tamanho da fonte e ativar a visualização dos números de linha no editor de texto.

1.5 Primeiro Sketch Arduino

Para verificar que a instalação foi bem sucedida e tudo funciona em pleno, pode-se experimentar um programa de demonstração que vem incluído na instalação do Arduino. Neste caso, para “ver” um Arduino a funcionar não é ainda necessário escrever um programa.

A escolha para primeira interação com a plataforma é o “blink”. Menu: **File**→**Examples**→**01.Basics**→**Blink**

CAPÍTULO 1. INTRODUÇÃO

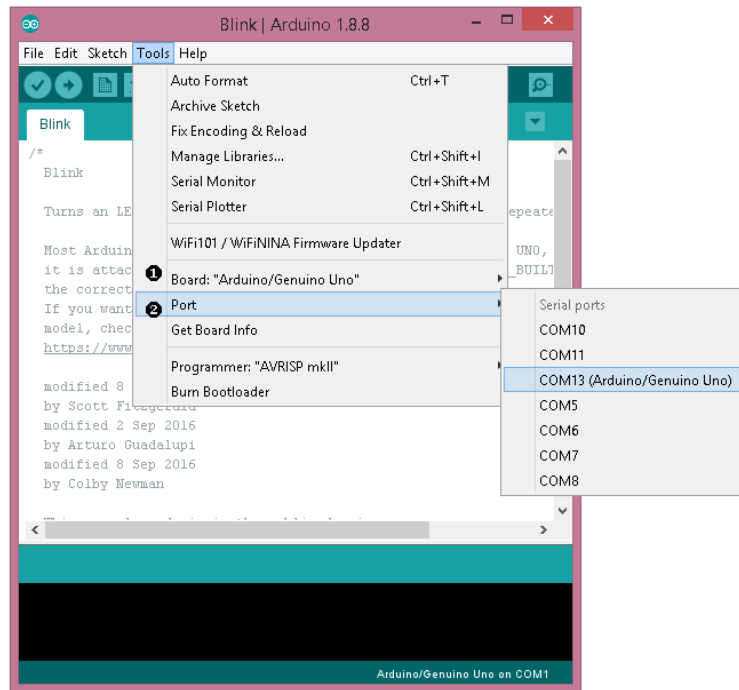


Figura 1.16: Configuração do porto de comunicações da placa Arduino Uno.

Listagem do Sketch `proj/blink/blink.ino`:

Listagem 1.1: Código do exemplo de demonstração do Arduino “blink”

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);                     // wait for a second
}
```

Antes de programar o sistema Arduino, é necessário usar o compilador para traduzir o texto escrito pelo programador para linguagem máquina (os 0s e 1s que fazem o CPU funcionar). Se não existirem erros no programa, pode-se programar o Arduino. A compilação é sempre realizada antes da programação. As acções de compilação e programação são feitas usando os comandos ilustrados na figura 1.17.

- Compilar: menu **Sketch**→**Verify/Compile**, ou teclas de atalho **CTRL+R**.
- Programar: Menu **Sketch**→**Upload**, ou teclas de atalho **CTRL+U**.

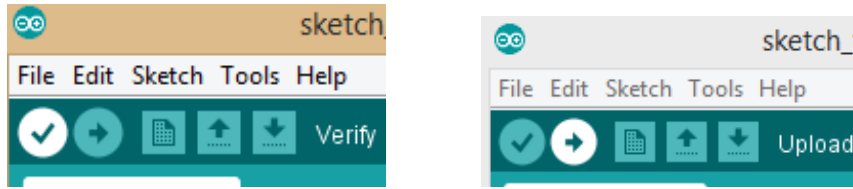


Figura 1.17: Botões do menu do ambiente Arduino para compilar o programa (esquerda) e programar a placa Arduino (direita).

Após a compilação com sucesso, é exibida a taxa de ocupação de memória com as instruções e as variáveis do programa para o Arduino escolhido. Para o exemplo “blink”, no Arduino Uno, o resultado de compilação é o seguinte:

```
Sketch uses 930 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.
Maximum is 2048 bytes.
```

O resultado da compilação deve ser sempre verificado para garantir que o programa não tem erros ou não está a ocupar mais recursos do que os disponíveis no sistema escolhido. Caso sejam necessários mais recursos e a funcionalidade seja imprescindível, deve-se procurar outro sistema que ofereça mais recursos. Por exemplo, o Arduino Mega 2560 que oferece um tamanho de programa máximo de 253952 bytes e 8192 bytes de dados. A figura 1.18 ilustra o aspecto do IDE Arduino após a execução destes dois passos:

1.6 Outros Exemplos

O ambiente de desenvolvimento Arduino traz com a instalação vários exemplos prontos a correr. Alguns dos exemplos necessitam de componentes ou periféricos ligados nos pinos de E/S do Arduino para poderem funcionar. Os exemplos podem ser encontrados através do seguinte Menu: **File**→**Examples**. Os tipos de exemplos disponíveis estão indicados na tabela 1.3. Existem ainda exemplos destinados à demonstração da utilização de outras placas ou bibliotecas e que podem ser instalados posteriormente, por exemplo, a biblioteca gráfica Adafruit.

1.7 Ferramentas

O ambiente de desenvolvimento Arduino inclui várias ferramentas destinadas a facilitar a interação do utilizador/programador com o sistema Arduino para depuração, visualização e interface/console de texto. Estas ferramentas encontram-se no menu “Tools”:

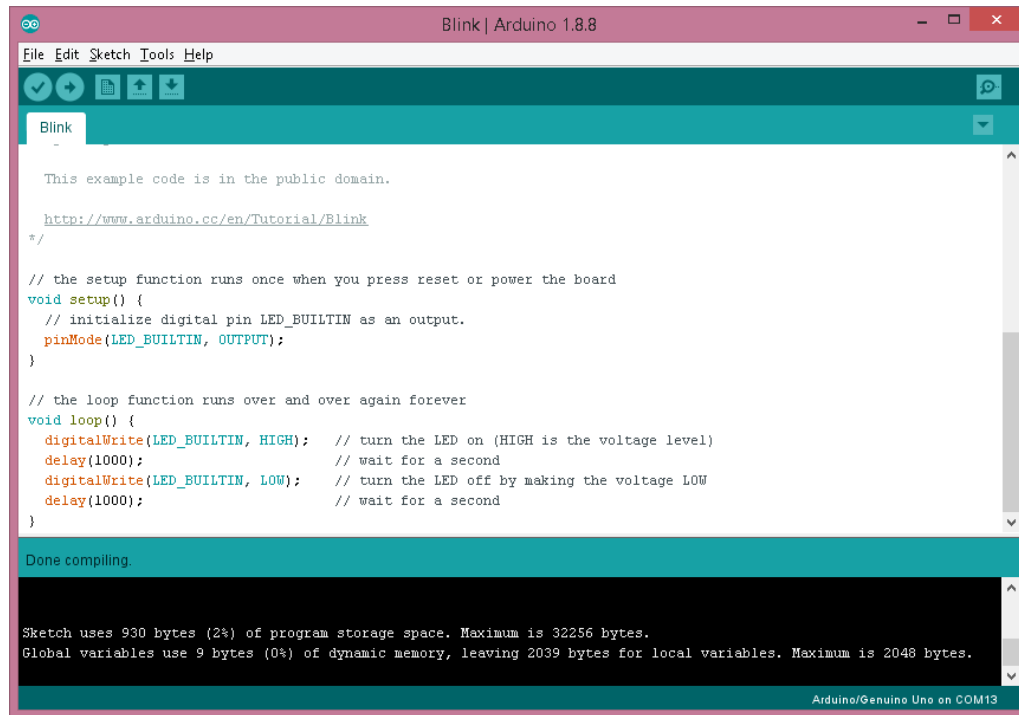


Figura 1.18: Compilação e programação do exemplo Blink para uma placa Arduino Uno.

- Serial Monitor - Atalho CTRL-SHIFT-M - Terminal de texto para leitura e introdução de texto. Útil para fornecer indicação da leitura de valores ou execução de uma determinada instrução ou função.
- Serial Plotter - Atalho CTRL-SHIFT-L - Traçador gráfico para representar de forma gráfica os valores enviados pelo Arduino.

1.7.1 Terminal de Texto

O terminal para recepção e transmissão de texto é conhecido como *Serial Monitor*, e mostra os caracteres enviados pelo Arduino, através do cabo USB. O terminal funciona com base em caracteres de texto (ASCII)⁵, passados através de um porto de comunicações interno série assíncrono conhecido como UART, ligado ao porto USB da placa Arduino. No lado do computador, o Arduino aparece como um dispositivo de comunicação série (COM* em Windows e /dev/ttyACM* em Linux). A programação do porto para transmissão do texto é feita através de um elemento chamado **Serial**, sendo necessário que tanto o Arduino e o computador usem a mesma configuração (ex. *baud rate*), sob pena de não ser possível reconhecer o texto enviado. Os detalhes da sua programação são explicados no capítulo IV deste livro.

⁵ver Anexos

Exemplos	Descrição
Basics	manipulação básica dos pinos de E/S digitais e analógicos
Digital	aplicações para diferentes configuração e utilização dos pinos de E/S digitais
Analog	aplicações para diferentes configuração e utilização dos pinos de E/S digitais
Communication	demonstração de diferentes funcionalidades de utilização do canal série do Arduino
Control	demonstração de formas de realização de ciclos de execução num programa Arduino
Sensors	demonstração de alguns sensores de movimento, nomeadamente acelerómetros e detectores de vibrações
Display	demonstração de mostradores baseados em filas e matrizes de LEDs
Strings	exemplificação de diferentes operações possíveis sobre <i>Strings</i> , ou sequências de caracteres
USB	emulação de um dispositivo USB para simular a operação de um teclado ou de um rato num computador
StarterKit BasicKit	vários exemplos que acompanham o Starter Kit Arduino
ArduinoISP	permite usar a plataforma Arduino como um programador de MCUs da família AVR

Tabela 1.3: Lista de exemplos que acompanham a instalação do ambiente de desenvolvimento Arduino.

O *Serial Monitor* encontra-se sob o menu: **Tools**→**Serial Monitor**, ou através do atalho: **CTRL+SHIFT+M**, e dá também a possibilidade de indicar o tempo em que cada linha foi recebida (*Show Timestamp*). A figura 1.19 ilustra o terminal de texto em funcionamento, à direita, ao mostrar as mensagens gerados pelo Sketch ao fundo.

A listagem 1.2 serve para demonstrar o funcionamento do Arduino a receber caracteres introduzidos pelo utilizador e enviar o carater correspondente à cifra de César⁶ para o terminal. A demonstração da aplicação encontra-se na figura 1.20. Para testar este Sketch o terminal de texto deve estar configurado para funcionar a 9600 baud.

Listagem 1.2: Programa de demonstração do terminal Arduino

```
/* ..Cesar Cipher:.
```

⁶Cifra baseada na substituição de cada carater da mensagem original por outro deslocado N posições no alfabeto, onde N é a chave da cifra usada para cifrar e recuperar a mensagem original. A mensagem original é obtida deslocando as mesmas N posições mas no sentido contrário.

CAPÍTULO 1. INTRODUÇÃO

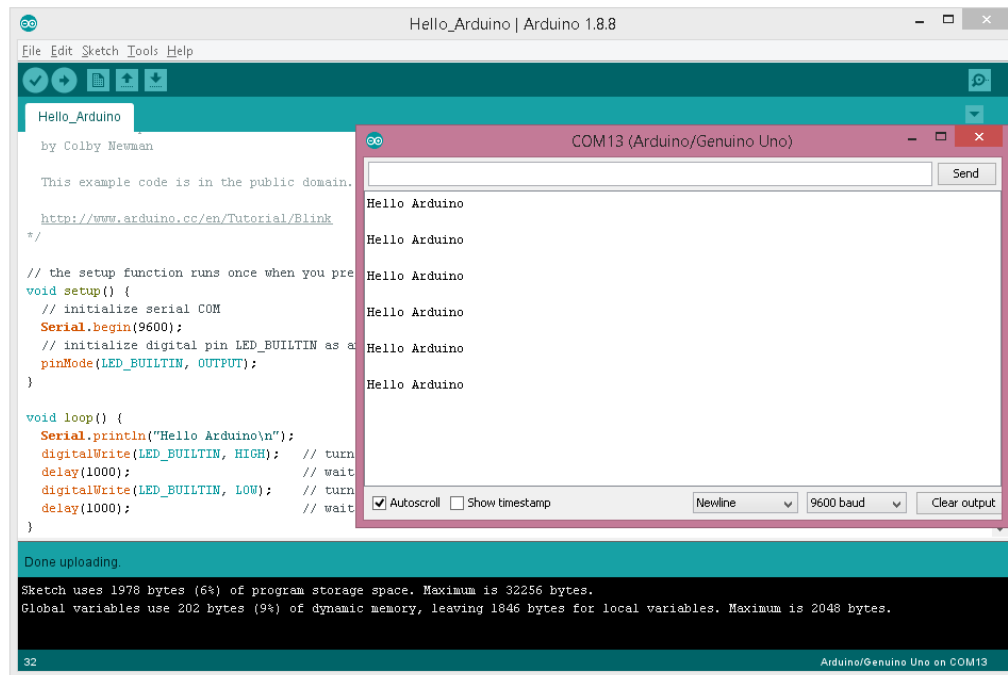


Figura 1.19: Serial Monitor

```
* receives string from the user and prints the ciphered message  
* in the serial monitor  
*/  
  
char key;  
  
char cesar(char plain, char key)  
{  
    char aux;  
    if((plain >= 'a') && (plain <= 'z')) {  
        aux = plain + key;  
        if(aux < 'a') {  
            aux += 'z' - 'a';  
        }  
        else if(aux > 'z') {  
            aux += 'a' - 'z';  
        }  
    }  
    else {  
        aux = plain;  
    }  
    return aux;  
}  
  
void setup()  
{  
    Serial.begin(9600);  
    while (!Serial) ; // wait for serial port to connect.  
    randomSeed(analogRead(0));  
    key = random(0, 20) - 10; // key between -10 and 10  
    Serial.println("Welcome to Cesar Cipher.");  
    Serial.print("Key: ");  
    Serial.println(key, DEC);  
    Serial.println("Please enter the messages to cipher...");  
}
```

```

}

void loop()
{
  char c, p;
  if (Serial.available() > 0) {
    p = Serial.read();
    p = tolower(p);
    c = cesar(p, key);
    Serial.print(c);
  }
}

```

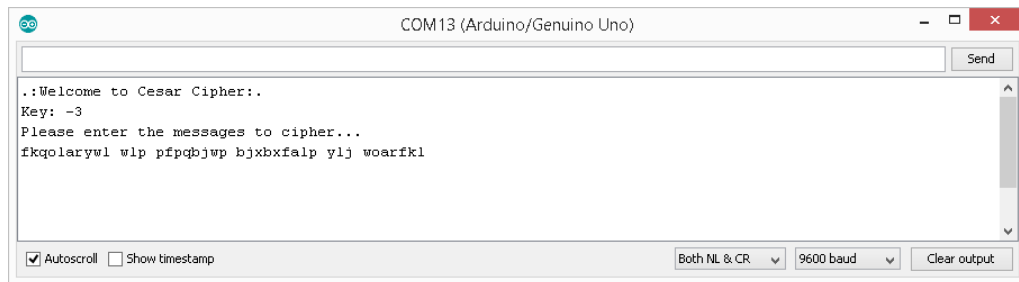


Figura 1.20: Demonstração da aplicação para cifrar uma mensagem introduzida pelo utilizador com a cifra de César.

1.7.2 Traçador Gráfico

Para as aplicações que geram muitos valores numéricos pode ser mais fácil perceber a variação dos dados visualizando-os num gráfico do que ler uma lista de valores. O traçador gráfico (*Serial Plotter*) permite facilitar a leitura e interpretação de valores numéricos ao longo do tempo. Para que os valores sejam apresentados é necessário que o Arduino envie pelo porto série uma linha de texto contendo o valor decimal pretendido, por exemplo `Serial.println("123");`.

Esta ferramenta suporta vários traços em simultâneo, sendo os valores enviados numa linha e separados por uma virgula (.). Quando a aplicação Arduino escrever valores na porta série, o *Serial Plotter*, produz um gráfico correspondente aos valores recebidos em que:

- o valor determina a posição no eixo vertical;
- o eixo horizontal corresponde ao instante de tempo do valor (incrementado automaticamente a cada nova amostra).

O traçador gráfico encontra-se no menu: **Tools**→**Serial Plotter**, ou através da combinação de teclas de atalho: **CTRL+SHIFT+L**. O *Serial Plotter* recebe os valores via porta série, pelo que a velocidade de transmissão vai limitar quanto tempo o Arduino irá ficar dedicado apenas à comunicação.

CAPÍTULO 1. INTRODUÇÃO

Quanto mais rápida a comunicação e menor a quantidade de informação a transmitir, menor será o tempo ocupado pelo Arduino a apresentar um novo ponto no gráfico. Tal como no caso do terminal de texto, é necessário ter a mesma configuração no porto série do Arduino e do computador. A figura 1.21 exemplifica o funcionamento do traçador gráfico.

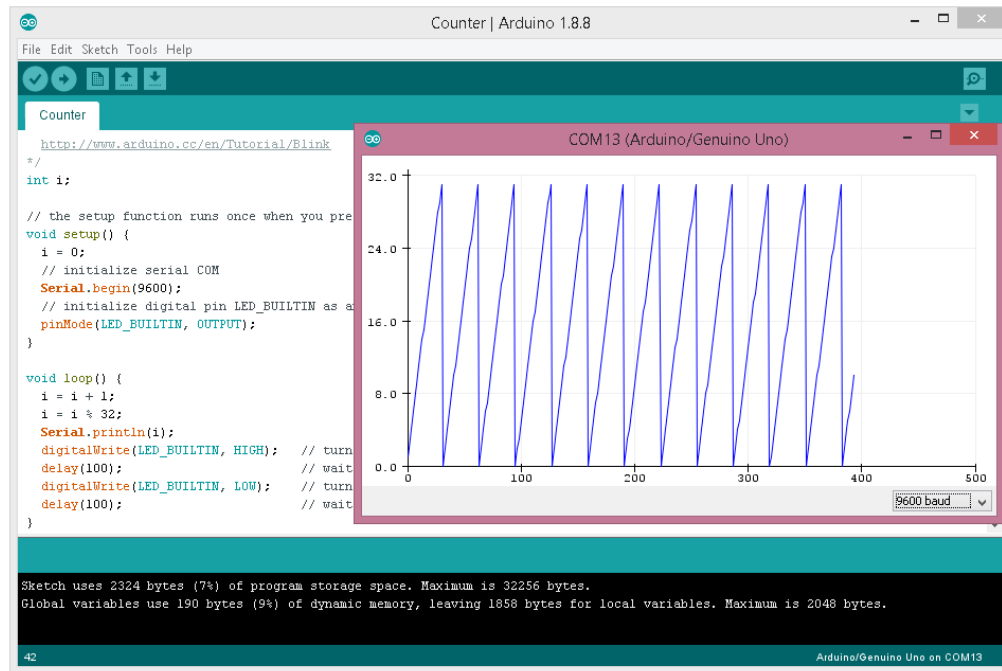


Figura 1.21: Serial Plotter

2

Conceitos Básicos

Este capítulo aborda os conceitos-chave em eletrónica e programação para conseguir entender e realizar diferentes tipos de sistemas embebidos.

O capítulo começa por explicar as grandezas físicas envolvidas em electrónica, apresenta os componentes electrónicos mais comuns e o seu princípio de funcionamento. São dadas também noções sobre o desenvolvimento e prototipagem de sistemas embebidos.

A segunda metade deste capítulo é dedicada a rever conceitos de programação em C++ e explicadas as extensões introduzidas na plataforma Arduíno, nomeadamente como ter um sistema embebido a interagir com periféricos externos e usar bibliotecas criadas por terceiros. O capítulo termina com noções de avaliação de desempenho e recursos usados.

2.1 Grandezas Físicas

À medida que os materiais e as suas propriedades foram sendo descobertos ao longo do tempo, houve necessidade de serem quantificados em medidas de grandezas. Para garantir que as medidas são entendidas e obtidas da mesma forma por todos, foi criado o Sistema Internacional de unidades - do francês *Système International* (SI). Este sistema de unidades está baseado em sete unidades fundamentais, e vinte e duas unidades derivadas. O sistema SI é adoptado praticamente em todo o Mundo, excepto nos Estados Unidos da América. A eletrónica baseia todos os seus princípios de funcionamento nas propriedades físicas dos seus componentes, daí ser importante conhecer quais as grandezas envolvidas para se criarem circuitos que se comportem de forma esperada.

CAPÍTULO 2. CONCEITOS BÁSICOS

A tabela 2.1 mostra as unidades SI fundamentais, juntamente com a medida, abreviatura e o símbolo usado. A tabela 2.2 resume as principais grandezas físicas mais relevantes no âmbito de sistemas embebidos, incluindo a abreviatura pelas quais são conhecidas e a respetiva unidade.

Medida	Abrev.	Unidade	Símbolo
Comprimento	L	Metro	m
Massa	M	grama	g
Tempo	T	Segundo	s
Corrente elétrica	I	Ampere	A
Temperatura	Θ	Kelvin	K
Quantidade de substância	N	Mole	mol
Intensidade luminosa	J	Candela	cd

Tabela 2.1: Quadro-resumo das 7 unidades fundamentais do Sistema Internacional.

Como as grandezas físicas podem ter valores muito grandes ou muito pequenos, o SI definiu também um conjunto de prefixos métricos, para facilitar a escrita e leitura de valores com muitos zeros. Os prefixos são símbolos que representam potências de base 10 (10^x). Por exemplo, um comprimento de 1000 m (10^3 m) pode ser representado por 1 *km*, tal como uma milésima parte de um metro, um milímetro, 1 *mm* (10^{-3} m), correspondente a 0,001 *m*. A tabela 2.3 mostra os prefixos SI mais usados, bem como a relação de grandeza. Todas as grandezas são representadas com um carater existente no teclado alfanumérico, exceto o μ . Para facilitar a sua escrita em sistemas que não aceitam símbolos usa-se o *u*, pois é o carater mais parecido com o símbolo μ , ficando a faltar a “perna”.

2.2 Eletrónica

As próximas seções são dedicadas à teoria e funcionamento de circuitos eletrónicos, e à introdução aos componentes mais comuns em circuitos eletrónicos: qual o seu aspeto físico, como funcionam, quais as grandezas envolvidas e como se podem relacionar com outros componentes. Esta seção também apresenta as ferramentas usadas no desenvolvimento de circuitos.

2.2.1 Tensão, Resistência e Corrente

Em eletrónica as três grandezas mais importantes num circuito são:

- Tensão (V) - Em analogia à canalização, a tensão pode ser vista como a quantidade de água congelada numa pequena fatia de um cano. Para interligar vários componentes é necessário que eles respeitem a “largura

Medida	Abrev.	Nome	Símbolo	Relação
Ângulo (plano)	θ	Radiano	<i>rad</i>	
Ângulo sólido	Ω	Esterradiano	<i>sr</i>	
Frequência	f	Hertz	Hz	T^{-1}
Força / peso	F	Newton	N	
Pressão / deformação	P	Pascal	Pa	N/m^2
Energia / trabalho / calor	W	Joule	J	$N.m$
Potência / fluxo radiante	P	Watt	W	J/s
Carga elétrica ou quantidade de eletricidade	Q	Coulomb	C	
Tensão / diferença de potencial	U	Volt	V	W/A
Capacidade	C	Farad	F	C/V
Resistência, impedância, reactância	R	Ohm	Ω	V/A
Condutância elétrica	G	Siemens	S	Ω^{-1}
Fluxo magnético	F	Weber	Wb	$V.s$
Densidade do fluxo magnético	B	Tesla	T	Wb/m^2
Indutância	L	Henry	H	Wb/A
Temperatura relativa a 273.15K	T	graus celsius	$^{\circ}C$	
Fluxo luminoso	Φ_v	Lumen	lm	$cd.sr$
Intensidade Luminosa	Iv	Candela	<i>cd</i>	
Iluminância	Ev	Lux	lx	lm/m^2

Tabela 2.2: Quadro-resumo das grandezas físicas derivadas mais relevantes para os sistemas embebidos.

máxima” para poderem funcionar adequadamente. Se o raio de dois canos for diferente, a canalização não vai funcionar corretamente. Muitos componentes e circuitos eletrônicos são fabricados para funcionarem com uma determinada tensão, se for usada outra tensão inferior eles podem não funcionar, se for superior ao especificado, os componentes vão queimar. Na maioria dos sistemas embebidos, onde é necessário interligar vários componentes, é habitual usar-se sempre a mesma tensão para todos os componentes, tal como uma canalização usa o mesmo “diâmetro de canos”. Em eletrônica a tensão é definida como a diferença de potencial entre dois pontos, e pode ser medida por exemplo nos terminais de uma pilha. Dois pontos ligados por um fio condutor têm uma diferença de potencial nula (curto-circuito). A tensão é medida em Volt (V).

CAPÍTULO 2. CONCEITOS BÁSICOS

Nome do Prefixo	Símbolo	Equivalencia	Base 10
Tera	T	1 000 000 000 000	10^{12}
Giga	G	1 000 000 000	10^9
Mega	M	1 000 000	10^6
Quilo	k	1 000	10^3
Mili	m	0,001	10^{-3}
Micro	μ ou u	0,000 001	10^{-6}
Nano	n	0,000 000 001	10^{-9}
Pico	p	0,000 000 000 001	10^{-12}
Femto	f	0,000 000 000 000 001	10^{-15}

Tabela 2.3: Prefixos mais usados na representação de quantidades de unidades SI.

- Resistência (R) - Equivalente a um estrangulamento de uma mangueira de água. Quanto maior for o aperto, maior é a resistência, e menos o circuito “flui”. Por vezes é necessário colocar elementos para limitar que o fluxo exerça demasiada pressão nos restantes componentes. Em eletrónica as resistências são usadas para provocar uma queda de tensão entre dois pontos, ou limitar a corrente que percorre um circuito. A resistência é medida em Ohm (Ω)
- Intensidade da corrente elétrica (I) - O conceito de corrente elétrica é equivalente à quantidade de água que passa num cano num determinado instante (caudal). Um cano quando ligado a uma bomba de água, sem qualquer estrangulamento, ou resistência, fornece o máximo de água possível pela bomba. Mas, se existir algum elemento que cause atrito, por exemplo um cano entupido, uma torneira fechada ou uma mangueira apertada, o caudal de água vai ser menor do que sem qualquer impedimento. Tal como o caudal de água numa secção do cano, em eletrónica, a corrente também tem uma direcção, ou sentido, tendo-se convencionado que circula do terminal positivo do gerador, pelo resto do circuito até ao terminal negativo do gerador. A corrente define o consumo de um componente ou circuito eletrónico em funcionamento e é medida em Ampere (A).

Estas três grandezas relacionam-se através da Lei de Ohm.

$$R = \frac{V}{I}[\Omega] \quad (2.1)$$

$$V = R \times I[V] \quad (2.2)$$

$$I = \frac{V}{R}[A] \quad (2.3)$$

$$(2.4)$$

A lei de Ohm é vista como uma simplificação válida para a corrente contínua, e precedeu as leis de Maxwell, que regem o funcionamento de circuitos elétricos em corrente alternada. A tensão e a corrente podem ser observadas num circuito em funcionamento através de aparelhos de medida:

- Tensão - medida com um voltímetro (ou multímetro em modo voltímetro) ligado em paralelo com 2 nós do circuito a medir. Nos sistemas embebidos este valor ronda entre os 0-5 V;
- Corrente - medida com um amperímetro (ou multímetro em modo amperímetro) ligado em série, ou intercalado, na malha do circuito a medir. Nos sistemas embebidos este valor ronda os 0-2 A;

Estas medidas têm polaridade, ou seja, se os terminais do aparelho de medida forem trocados, a medida terá valor negativo. A resistência é medida com o circuito desligado. Se se pretender medir a resistência de um componente, é necessário retirá-lo do circuito para que os demais componentes não influenciem a medida.

Considerando uma fonte de alimentação de 5 V com uma resistência de 1 M Ω , a corrente que circula no circuito é $5\text{ V}/10^6\ \Omega = 5\ \mu\text{A}$. Se a resistência for 1000 vezes menor, ou seja, de 1 k Ω , a corrente é $5\text{ V}/1000\ \Omega = 5\ \text{mA}$. Se a resistência for apenas 1 Ω (resistência dos condutores), a corrente que percorre o circuito é $5\text{ V}/1\ \Omega = 5\ \text{A}$.

Outra grandeza usada para caracterizar elementos e circuito eletrónicos é a potência (fornecida e consumida). A potência é medida em Watt e é derivada de outras duas grandezas físicas: tensão e intensidade da corrente. A relação entre a potência, a tensão e a corrente num determinado elemento é: $P = V \times I$ [W]

Recuperando a analogia da canalização, a potência corresponde ao volume de água fornecida num instante de tempo (como se o cano ficasse congelado). Se houver alta pressão vai haver muita água numa fatia do cano. Se a pressão for muito baixa, a água será menor que no caso anterior. É possível ter o mesmo volume de água com canos largos e estreitos, mas o cano estreito necessita de ter água com mais pressão do que o cano mais largo. De forma semelhante, em eletrónica a mesma potência pode ser obtida com pouca tensão e muita corrente, ou muita tensão e pouca corrente. No caso de sistemas embebidos as potências envolvidas são normalmente baixas (< 5 W), exceto quando é necessário controlar cargas ou atuadores que consumam mais corrente ou funcionem com tensões maiores, como por exemplo lâmpadas, motores elétricos, ou outros componentes eletro-mecânicos que necessitem de exercer força. Os Arduínos são alimentados normalmente a 5 V e internamente funcionam a 5 V ou 3,3 V. Sem quaisquer periféricos ligados consomem cerca de 50 mA (± 10 mA).

Os nomes das grandezas físicas foram atribuídos em reconhecimento do

trabalho desenvolvido por vários cientistas notáveis pelo seu trabalho na área, nomeadamente:

- Alessandro Volta (1745-1827) - tensão,
- André-Marie Ampère (1775-1836) - corrente,
- Georg Simon Ohm (1789-1854) - impedância,
- James Watt (1736-1819) - potência,
- Michael Faraday (1791-1867) - capacidade,
- Joseph Henry (1797-1878) - indutância,
- Nikola Tesla (1856-1943) - corrente alternada e motor elétrico,
- James Clerk Maxwell (1831-1879) - fluxo magnético.

2.2.2 Sinais, Tempo, Frequência e Espectro

Um sinal é uma representação que relaciona duas grandezas: tensão e tempo. Uma lanterna a pilhas tem um brilho constante durante bastante tempo, e por isso, é um sinal contínuo. No entanto, existem sinais que variam ao longo do tempo em sistemas eletrônicos, tais como: áudio (música e voz), vídeo, ondas de Rádio Frequência (RF), e sinais biológicos tais como o batimento cardíaco ou impulsos nervosos.

Os sinais variantes no tempo são ainda diferenciados em periódicos e não periódicos. Por exemplo, o fornecimento da rede elétrica é feito com uma onda sinusoidal com uma amplitude de 220 V que oscila 50 vezes por segundo (tempo de voltar à posição inicial), ou seja, a onda repete-se a cada 20 ms. A frequência de um sinal periódico é medida em Hertz (Hz). Se um sinal oscilar 1000 vezes durante 1 segundo, significa que o sinal tem uma frequência de 1 kHz. O recíproco da frequência é o período da onda, eq. 2.5, ou seja, o tempo que ela demora a completar um ciclo, neste caso é 1 μ s:

$$T = \frac{1}{f}[\text{seg}] \quad (2.5)$$

Para o exemplo anterior, um sinal com uma frequência de 1 kHz, tem um período de 1 μ s, ilustrado na figura 2.1. Esta onda tem uma amplitude de 1 V, o que significa que tem 2 V_{pp} (pico-a-pico = diferença entre o valor máximo e mínimo), e oscila em torno de 0 V (deslocamento ou *offset*). Os sinais periódicos são também caracterizados pela forma da sua onda: sinusoidal, rectangular ou triangular.

É possível ter um sinal composto por várias ondas em frequências diferentes sobrepostas. Exemplo disso é o sinal áudio de uma música tocada por uma

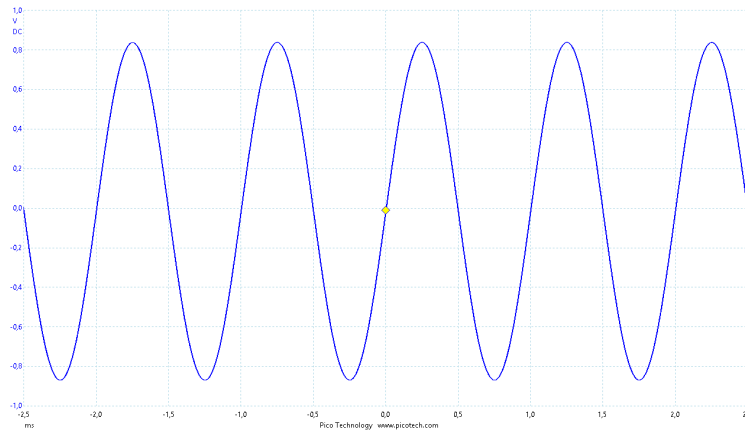


Figura 2.1: Onda sinusoidal com variação de 1000 ciclos (1 kHz) por segundo.

orquestra ou banda rock, que habitualmente tem a mistura dos vários tons produzidos por vários instrumentos musicais em simultâneo. Numa música, é fácil distinguir sons graves (baixas frequências normalmente produzidas pelo bombo da bateria, contra-baixo e pela viola-baixo) dos sons agudos (altas frequências produzidas pela guitarra elétrica instrumentos de sopro, violinos e pelos pratos da bateria).

A forma de visualizar as frequências presentes num sinal é através de um diagrama de espetro. Este diagrama relaciona no eixo horizontal os valores das frequências, enquanto que o eixo vertical apresenta a amplitude do sinal para cada frequência. O espetrograma contém a representação do espetro ao longo do tempo. Este gráfico relaciona frequência, amplitude e tempo, onde a intensidade é representada por uma cor, e o eixo vertical passa a representar o tempo. Ambos os gráficos, espetro e espetrograma, encontram-se na figura 2.2.

O conceito de espetro é relevante para aplicações onde é necessário identificar quais as frequências existentes num sinal para as poder amplificar ou anular, por exemplo, para dar ênfase a um instrumento na música, ou suprimir o “zumbido” causado pelos 50 Hz da rede elétrica do sinal gerado por um sensor. A largura de banda de um sinal é definida como a diferença entre as frequências máxima e mínima de um sinal. Os limites são definidos num diagrama de espetro observando o intervalo correspondente aos pontos onde o sinal tem -3 dB (ou 70.7%) do seu valor máximo.

2.2.3 Analógico e Digital

Um sinal elétrico (analógico) evolui ao longo do tempo tal como qualquer outra grandeza da natureza, é o fenómeno contínuo, ou seja, não varia instantaneamente e, entre dois valores, tem uma quantidade de valores infinitos, o equivalente a uma função contínua em matemática. Na natureza encontra-

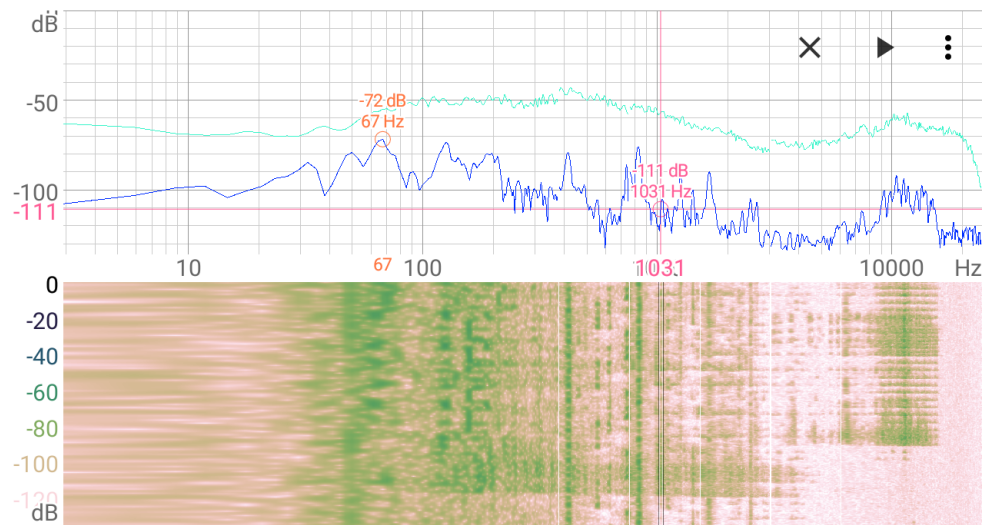


Figura 2.2: Exemplo do espectro (em cima) e espectrograma (em baixo).

Tipo de Sangue	bit 1	bit 0	Num
A	0	0	0
B	0	1	1
AB	1	0	2
O	1	1	3

Tabela 2.4: Exemplo de codificação da informação relativa ao tipo de sangue com dois bits.

mos várias analogias com sinais analógicos: intensidade da luz do sol ao longo do dia, variação da temperatura, e deslocamento ao longo de um percurso. Eles podem variar muito rapidamente mas nunca “saltam” instantaneamente de valor.

Um sinal digital é uma representação usada em computadores, e é baseado em 2 estados distintos, 0 e 1, servindo para codificar uma unidade fundamental de informação (bit). Por exemplo, com um bit pode-se ter a informação sobre a ocorrência de chuva: 0 = não chove, 1 = chove. Para codificar mais do que dois valores podem-se associar vários bits. Exemplo de uma variável com quatro valores é o tipo de sangue: A, B, AB ou O. Neste caso a codificação da informação pode ser feita da forma apresentada na tabela 2.4 com dois bits. Podem-se juntar tantos bits quantos necessários para codificar a informação necessária. Como a representação binária ocupa mais dígitos é habitual usar a base hexadecimal para representar os valores com menos dígitos. Os computadores atuais funcionam com base em palavras de 64 bits.

Um sinal digital tem correspondência a um dos dois extremos de potencial

do sinal para indicar o valor do bit. O 0 corresponde a GND e o 1 a VCC. A figura 2.3 ilustra a representação de um sinal analógico (topo) e um sinal digital (baixo). O eixo horizontal representa o tempo, e o eixo vertical a tensão em cada instante de tempo.

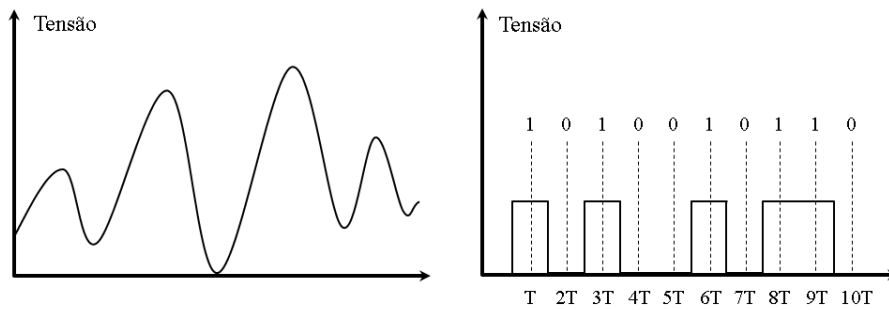


Figura 2.3: Sinal analógico (esquerda) e digital (direita).

Circuitos Analógicos

Os circuitos analógicos funcionam numa gama de tensões capaz de suportar a variação dos seus sinais em cada etapa do circuito, desde o sensor até ao atuador. Um circuito analógico é caracterizado por ter os componentes ligados de forma a realizarem uma função matemática, em que os argumentos da função são os sinais analógicos à entrada, e os resultados correspondem aos valores de tensão analógica na saída.

Um exemplo de um circuito analógico é o amplificador áudio que recebe na entrada um sinal de muita baixa amplitude de uma fonte de som (microfone ou guitarra) e, ao longo de várias etapas de amplificação vai aumentar o nível de corrente que a carga (colunas) podem consumir, que pode ir desde poucos Watts, para um rádio, até milhares de Watts para concertos em estádios.

Os circuitos analógicos são mais sensíveis a atenuações e perturbações do meio, nomeadamente flutuações da tensão de alimentação e interferências parasitas. O equipamento áudio é um bom exemplo disso, sendo o equipamento de alta-fidelidade projectado com bastante cuidado.

Hoje em dia a eletrónica analógica a nível industrial é realizada em circuitos integrados, sendo necessárias ferramentas de desenvolvimento avançadas para especificar e simular o circuito antes de o realizar, bem como laboratórios bastantes sofisticados com “salas limpas” capazes de fabricar os circuitos integrados. Em Portugal existem algumas “salas limpas” em centros de investigação.

Circuitos Digitais

Para facilitar o processamento de informação em formato digital foram desenvolvidas portas lógicas, que realizam operações sobre valores 0 e 1 (GND e VCC), distanciando-se das limitações dos circuitos analógicos, e possibilita a interoperação entre sistemas realizados em tecnologias diferentes. Inicialmente os circuitos digitais, ou lógicos, eram realizados com portas lógicas combinatórias discretas NOT, AND, OR, e XOR, cujo funcionamento está descrito na tabela 2.5, e registos para guardar (ou fixar) os valores colocados nas suas entradas. As portas lógicas e os registos eram combinados para realizar um conjunto de operações sobre os bits na sua entrada, que podem ter valores 0 ou 1.

Para garantir que todos os elementos lógicos funcionam de forma determinística e coerente, são sincronizados com um sinal de relógio, como um timoneiro com um tambor num barco a remos, dá a indicação, através de pulsos, para que todos os registos do circuito guardem os valores lógicos nas suas entradas e os deixem fixos na suas saídas até à chegada do próximo pulso.

A	NOT	A	B	AND	A	B	OR	A	B	XOR
0	1	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	1	0	1	1
		1	0	0	1	0	1	1	0	1
		1	1	1	1	1	1	1	1	0

Tabela 2.5: Resumo das tabelas de verdade das funções lógicas elementares NOT, AND, OR e XOR.

A complexidade crescente dos circuitos lógicos levou ao desenvolvimento de linguagens de especificação (ex. Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL) e Verilog) e ferramentas próprias (ex. Cadence, Modelsim, Vivado, Quartus). Os circuitos digitais são fabricados em tecnologia semicondutora de silício de forma permanente - Application-Specific Integrated Circuit (ASIC), ou reconfigurável - FPGA.

Os ASICs são dispositivos que uma vez fabricados mantêm a sua funcionalidade para sempre, ao passo que as FPGAs podem ser reconfiguradas para realizarem outro circuito lógico. Os processadores dos computadores são um exemplo de um dispositivo ASIC e são os dispositivos digitais mais fabricados, figura 2.4 à esquerda. Existem dispositivos FPGA que combinam as duas tecnologias ao incluir um CPU junto à lógica reconfigurável, tal como o da figura 2.4 à direita.

Um CPU é um sistema digital síncrono que executa instruções a partir de um sinal de relógio ou *clock*. A capacidade de processamento de um processador é tão maior quanto mais instruções são executadas por unidade de tempo,

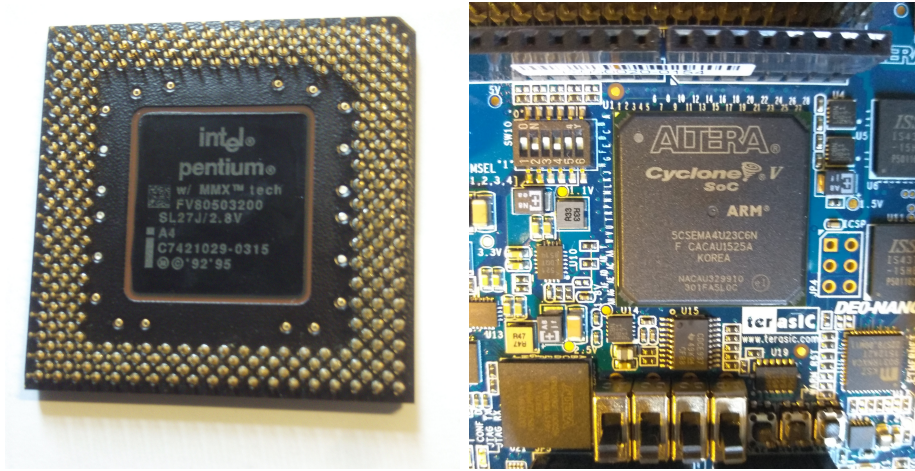


Figura 2.4: Processador Intel Pentium MMX (ASIC), à esquerda, e um dispositivo reconfigurável CPU+FPGA, à direita.

normalmente proporcional ao número de unidades de processamento no sistema e ao número de ciclos de relógio por segundo. A frequência máxima de funcionamento é especificada pelo fabricante e não deve ser ultrapassada (*overclocking*), sob pena do circuito deixar de funcionar corretamente ou ficar destruído.

2.2.4 Elementos de um Circuito

Nos circuitos eletrônicos existem três tipos de componentes fundamentais: geradores, cargas e condutores de energia elétrica. Os geradores são os produtores de energia elétrica no circuito, as cargas são os consumidores, e os condutores servem para transportar energia pelo circuito, sendo feitos de metal bom condutor elétrico, e revestidos de material isolante para proteger de ligações acidentais (curto-circuitos). Num circuito existe pelo menos um gerador e uma ou mais cargas, ligadas por diversos condutores, tal como ilustrado na figura 2.5.

Só existe transferência de energia elétrica entre os componentes do circuito quando este se encontra “fechado”, ou seja, todos os seus elementos estão interligados por condutores. Caso contrário diz-se que o circuito está “aberto”, pois não existe circulação de energia entre o gerador e os consumidores. O princípio de conservação da energia indica que a potência que o gerador fornece é a mesma que o elemento consumidor absorve, juntamente com a potência dissipada nos condutores.

Retomando a comparação de um circuito eletrônico a uma canalização, todas as ligações têm de estar fechadas para poder funcionar corretamente. O volume de água fornecido pela rede a um circuito de rega é igual à água irrigada pelos diversos aspersores.

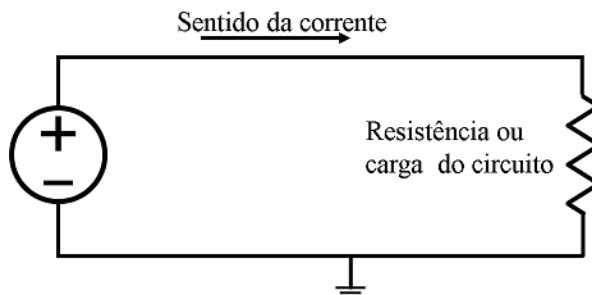


Figura 2.5: Exemplo de um circuito básico com um gerador, carga, condutores, e aparelhos de medida.

Os condutores elétricos não são perfeitos, ou seja, têm perdas e apresentam sempre alguma oposição (resistência) à passagem da corrente elétrica através da libertação de calor (efeito de Joule). Quanto mais comprido o condutor, maiores são as perdas. Normalmente para pequenos circuitos a funcionarem com baixas correntes as perdas são desprezáveis.

Geradores

Os geradores são a fonte de energia ao circuito e são caracterizados pela tensão gerada nos seus terminais e pela corrente máxima capazes de produzir. Existem 2 tipos de geradores: Corrente Alternada (CA), *Alternate Current* (AC), e Corrente Contínua (CC), *Drive Continuous* (DC). Corrente contínua significa que o sentido da corrente se mantém inalterado ao longo do tempo, ao passo que a corrente alternada troca de sentido periodicamente. Os geradores CC possuem um terminal positivo e outro negativo, e quando em funcionamento, a corrente circula do terminal positivo, atravessando todos os componentes do circuito até ao terminal negativo do gerador. Os geradores CA possuem um terminal com a fase e outro com o neutro. A corrente vai mudando de sentido (positivo e negativo) entre a fase e o neutro - cinquenta vezes por segundo na rede elétrica.

Baterias, pilhas, carregadores e fontes de alimentação fornecem corrente contínua. A corrente contínua é habitualmente usada para alimentação de circuitos de baixa potência, envolvendo tipicamente tensões abaixo dos 24 V. A rede elétrica, e alternadores fornecem corrente alternada. A corrente alternada é usada em máquinas domésticas e industriais, e na produção, transporte e distribuição de eletricidade por ser mais fácil de adaptar a linhas de menor tensão usando transformadores.

Pilhas e Baterias As pilhas e baterias são geradores eletroquímicos de CC usados para alimentar aparelhos eletrónicos de baixa tensão e portáteis, nomeadamente relógios, lanternas, rádios, carros telecomandados. As pilhas

e as baterias são identificadas pelas sua tensão, carga e dimensões (exemplo: CR2032, AA e AAA) e capacidade de fornecer energia (Ah), e podem ser recarregáveis ou não.

As pilhas recarregáveis e as baterias (auto ou *powerbank*) são utilizadas em equipamento cujo consumo esgota a sua capacidade em pouco tempo. Estas podem ser recarregadas no próprio equipamento, ou em carregadores adequados à sua tecnologia (ex. NiCd, NiMH, chumbo/ácido, LiFePO, e Li-Ion).

Uma pilha do tipo AA alcalina quando carregada tem aos seus terminais 1,5 V e é capaz de fornecer uma carga de cerca de 2000 mAh. Ligando três pilhas do tipo AA em série obtém-se uma pilha equivalente de 4,5 V com 2000 mAh - as tensões somam. Se as pilhas fossem ligadas em paralelo, o equivalente seria uma pilha de 1,5 V com 4000 mAh de carga - as correntes somam.

Uma forma de se obter uma estimativa sobre o tempo de funcionamento (h) do sistema embebido alimentado por uma pilha ou bateria passa por medir a corrente consumida, e dividir o valor da carga da bateria (Ah) pelo valor da corrente consumida (A).

Fontes de Alimentação As fontes de alimentação servem para fornecer uma determinada tensão mais baixa em CC a partir da rede elétrica em CA. Exemplos de fontes de alimentação são os carregadores de telemóvel e de computadores portáteis. As fontes de alimentação podem ser do tipo linear ou comutadas. As primeiras são as mais antigas, e são constituídas por um transformador, um circuito retificador, e um circuito regulador de tensão.

O transformador tem dois enrolamentos de fio (primário e secundário) sobre um núcleo de ferro, ou bobinas. O enrolamento primário encontra-se ligado à rede elétrica, produzindo um campo eletromagnético variável, e o enrolamento secundário do transformador é usado para o processo inverso, ou seja, converter o campo magnético numa tensão variável. A tensão à saída do transformador depende da relação do número de espiras entre o primário e o secundário. Por exemplo, um transformador com razão de espiras (primário:secundário) 20:1 com o primário ligado a 220 V CA vai gerar 11 V CA na saída do secundário. A vantagem dos transformadores é permitirem o isolamento galvânico da rede elétrica, isto é, não existe ligação elétrica entre o circuito de entrada e saída. O inconveniente é serem volumosos, pesados, dissiparem calor e terem uma eficiência energética na ordem dos 30-40%.

O circuito rectificador é normalmente construído com uma ponte de díodos para fazer a corrente alternada circular apenas num sentido, e um condensador para armazenar energia de forma a produzir uma tensão DC. De acordo com o circuito da figura 2.6, o transformador converte 220 V AC para uma tensão AC mais baixa, por exemplo 12 V. O andar seguinte é uma ponte

CAPÍTULO 2. CONCEITOS BÁSICOS

retificadora, constituída por quatro díodos, que apenas deixam a corrente passar no sentido positivo-negativo (ânodo-cátodo), de modo que quando a tensão de entrada for:

- positiva (+ no pino de cima da fonte, e a corrente flui da esquerda para a direita): o diodo D1 deixa passar corrente e o D4 não. Circulando pela carga (Load), D2 deixa passar corrente e D3 não;
- negativa (+ no pino de baixo da fonte, e a corrente flui da direita para a esquerda): os diodo D1 e D2 estão ao corte, ao passo que D3 e D4 estão em condução.

A saída da ponte retificadora é pulsada, tal como ilustrada pela onda a tracejado. Para que a tensão fique estável é necessário usar um condensador para armazenar energia nos períodos em que a onda está no valor máximo, e fornecer energia quando a onda está abaixo do valor máximo. O circuito integrado regulador de tensão de 7805 é colocado após o rectificador para que a saída da fonte seja constante 5 V DC. Os condensadores adicionais (C2 e C3) servem para estabilizar a tensão e reduzir o ruído da fonte.

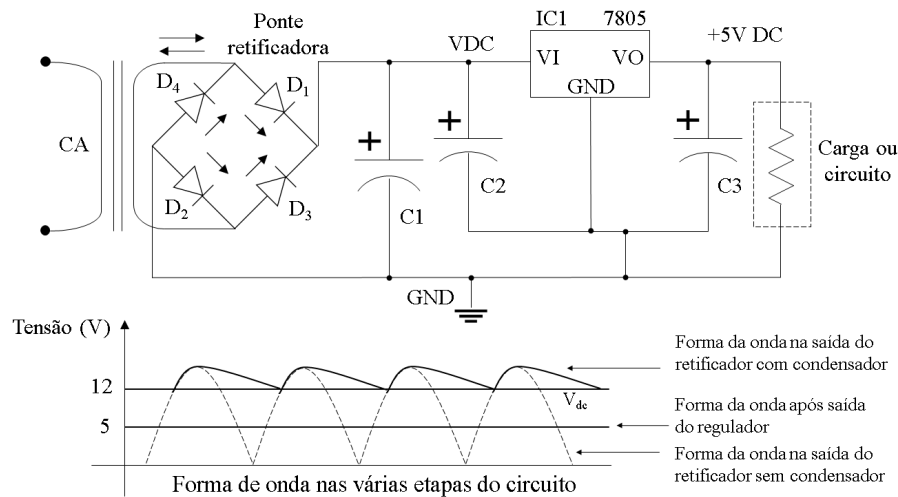


Figura 2.6: Esquema de uma fonte de alimentação com transformador e retificador.

As fontes de alimentação comutadas são mais recentes que os transformadores e comportam-se como se tivessem um interruptor que liga o circuito à rede elétrica durante instantes de tempo muito breves. Um circuito de controlo monitoriza a rede elétrica e trata da ativação para garantir que a saída da fonte é correta e estabilizada. A vantagem deste tipo de fontes é o fato de serem bastante compactas, leves e terem maior eficiência energética que as fontes com os transformadores antigos. A desvantagem deste tipo de fontes é a presença de ruído da rede elétrica, ou flutuações na tensão de saída.



Figura 2.7: Fonte de alimentação de bancada regulável.

Este problema é especialmente relevante nas unidades de baixo custo, ou de marca branca.

As fontes de alimentação de laboratório (bancada) permitem ajustar a tensão aos seus terminais, e a corrente máxima fornecida. Esta versatilidade é útil para testar circuitos com diferentes tensões de funcionamento, e impedir que o circuito consuma corrente em demasia em caso de funcionamento anómalo do circuito. A figura 2.7 mostra o aspecto de uma fonte deste tipo.

Painéis Solares O painel solar ou fotovoltaico é uma fonte de energia renovável uma vez que a sua capacidade de transformar a luz solar (fótons) em energia elétrica não se esgota. Sempre que houver luz solar a incidir no painel, existe energia elétrica a ser produzida. Os painéis solares são usados alimentar desde pequenas calculadoras até cidades inteiras.

O painel solar é usado para carregar baterias de 12 ou 24 V, que vão servir para alimentar circuitos eletrónicos em CC, ou aparelhos elétricos em CA, através do uso de inversores (geram 220 V CA a partir de 12 V ou 24 V CC).

Existem diferentes tecnologias de painel solar, nomeadamente silício policristalino e monocristalino, sendo os que os últimos têm maior rendimento. Um painel solar tem um período de vida útil de cerca de 10 anos.

Vale a pena sobre-dimensionar um painel solar para compensar momentos com menos sol e a degradação do painel ao longo do tempo. Para maximizar a energia transferida do painel solar para a bateria usa-se um controlador de seguimento do ponto máximo de potência (*Maximum Power Point Tracking* (MPPT)), tal como ilustrado na figura 2.8.

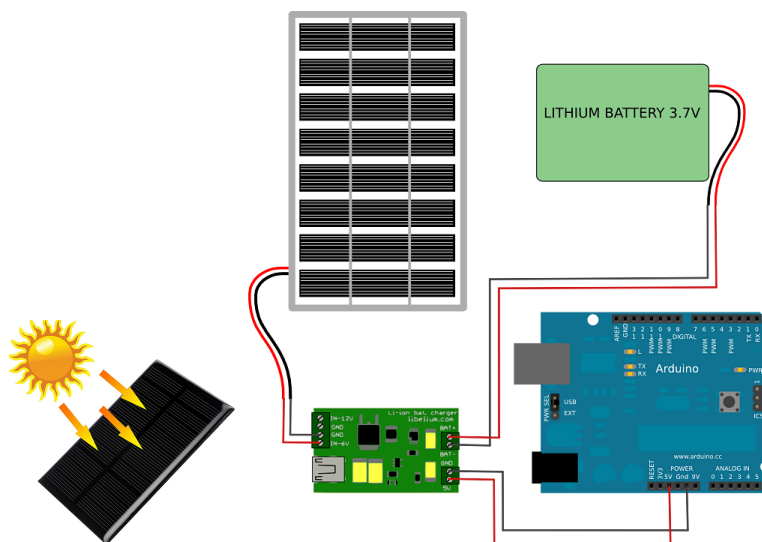


Figura 2.8: Esquema de alimentação de um Arduino por energia solar.

Alimentação Sem Fios A alimentação sem fios é usada para o carregamento de telemóveis sem ter de ligar o cabo de alimentação. Este método de transferência de energia assenta no princípio de funcionamento do transformador, mas sem o núcleo de metal. O circuito de transferência é composto por dois enrolamentos de fio acoplados de tal forma que um dos enrolamentos ligado a uma fonte de tensão alternada cria um campo magnético que varia ao longo do tempo, e o outro enrolamento transforma a variação do campo magnético em variação de tensão. No andar final do recetor existe um circuito rectificador e de filtragem que compensa as flutuações de tensão e gera uma tensão contínua. A figura 2.9 apresenta os elementos emissor e recetor de energia sem fios.

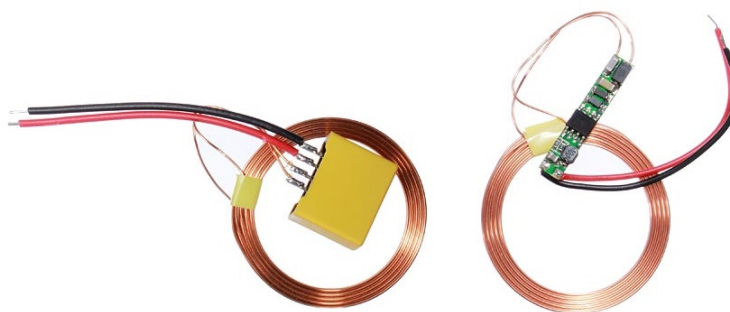


Figura 2.9: Circuitos de transmissão e recepção de energia sem fios.

Cargas

As cargas são todos os elementos consumidores de energia elétrica. Uma carga é caracterizada pela sua tensão de funcionamento (mínima e máxima), pela corrente consumida e, nalguns casos, polaridade. Exemplos de cargas vão desde resistências de aquecimento, LEDs, motores, Arduínos, computadores, e aparelhos eletrodomésticos.

Condutores

Os condutores servem para ligar os vários componentes e devem ser escolhidos de acordo com a sua aplicação. Um circuito que consuma mais corrente é ligado com fios de secção maior, ao passo que para ligar sinais digitais em pequenas distâncias, podem-se usar fios de secção muito reduzida. Para a transmissão de sinais RF são necessários cabos especiais que permitem minimizar as interferências e manter as características do sinal inalteradas o mais possível. Cabos e condutores para diferentes aplicações têm também associadas fichas ou terminais diferentes.



Figura 2.10: Condutores e cabos de diferentes tipos para ligação entre sistemas embebidos e outros periféricos.

2.2.5 Componentes Eletrónicos

Um circuito eletrónico é uma composição de vários componentes ligados de uma determinada forma para realizarem uma função. Existem diferentes tipos de componentes eletrónicos para provocar diferentes variações nos sinais que os percorrem. Os componentes eletrónicos mais usados estão sistematizados na tabela 2.6.

Com a miniaturização da eletrónica, os componentes mais usuais passaram a ser fabricados em tamanhos mais reduzidos e sem os longos terminais. A maioria dos circuitos eletrónicos usa componentes para montagem em superfície, ou Surface Mount Device (SMD) pois permite serem fabricados em linhas de montagem automatizadas.

Componente	Símbolo	Função
Interruptores	SW	ligar/desligar circuitos
Resistências	R	restringir a passagem de corrente, ou dissipar energia elétrica sob a forma de calor
Potenciômetro	P	resistência variável por intermédio de um veio que se desloca ou roda
Condensadores	C	armazenamento de energia, usando para filtrar sinais que variam no tempo
Bobinas	L	armazenar corrente
Diodos	D	deixar passar corrente num sentido (retificar)
Transistores	Q ¹	amplificar um sinal, ou controlar uma carga (ON/OFF)
Circuitos integrados	U	componentes de integram vários componentes simples da mesma tecnologia
Transformador	T	reduzir ou amplificar sinal CA

Tabela 2.6: Lista dos componentes eletrônicos mais usados.

Resistências

A resistência é o componente eletrônico mais elementar, serve para limitar a corrente máxima num circuito ou provocar uma queda de tensão, e é construída por material que se opõe à passagem de corrente elétrica. As resistências são caracterizadas pela sua impedância, medida em Ohm (Ω), e pela sua potência máxima dissipada, medida em Watts (W). As resistências podem ter um valor de impedância fixo ou variável. As resistências fixas têm os seus valores definidos por séries de fabrico (norma IEC 60063:1963), com uma sequência de valores normalizados, por exemplo, a série E12 é conhecida por ter 12 valores base: 10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, e 82 que podem ser obtidos multiplicando o valor anterior por 1,2. Aos valores base são multiplicadas potências de base 10^x , em que x varia entre 0 a 9. A série E12 admite uma tolerância de fabrico de $\pm 10\%$.

As resistências são normalmente fabricadas em séries de 1/4 W, 1/2 W e 1 W, existindo outras séries para valores de potência superiores. Se o limite de potência não for respeitado a resistência queima.

Considerando uma fonte de 5 V e querendo-se uma carga que use apenas 3,3 V, significa que 1,7 V vão “ter de ficar” na resistência. Para se determinar a resistência a introduzir no circuito é necessário saber qual a corrente consumida pela carga de 3,3 V. Assumindo que a carga consome 10 mA, pela Lei de Ohm determina-se que a resistência deve ter o valor de 170 Ω .

Para determinar qual a resistência a usar para limitar a corrente da carga usa-se a Lei de Ohm: $R = V/I$

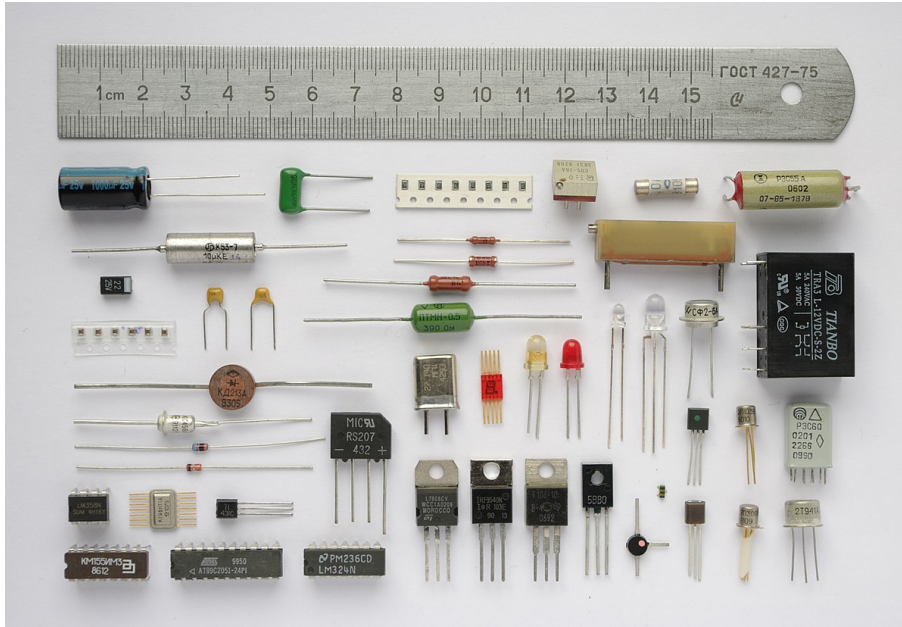


Figura 2.11: Aspecto de diferentes tipos de componentes eletrónicos.

Neste caso procura-se que caiam na resistência 1,7 V quando o circuito consumir 10 mA:

$$R = 1,7 / 10 \times 10^{-3} = 1,7 \times 1000 / 10 \quad R = 170 \, \Omega$$

A potência dissipada nesta resistência é:

$$P = R \times I^2 = 170 \times 10 \times 10^{-3} = 17 \, mW < 1/4 \, W$$

Nalguns circuitos podemos encontrar várias resistências associadas para dar um valor de resistência equivalente fora das séries de fabrico. As resistências podem ser associadas em série ou em paralelo, tal como exemplificado na figura 2.12.

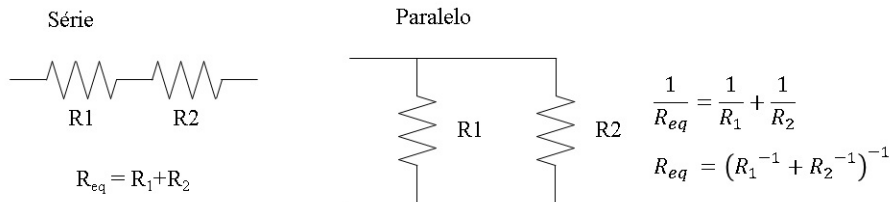


Figura 2.12: Associação de resistências em série (esquerda) e em paralelo (direita).

O valor da resistência equivalente é:

- em série, é a soma do valor das resistências: $R_{eq} = \sum_i R_i = R_1 + R_2 + \dots + R_n$;

CAPÍTULO 2. CONCEITOS BÁSICOS

Cor	Dígito 1	Dígito 2	Dígito 3	Multiplicador	Tolerância
Preto	0	0	0	1	-
Castanho	1	1	1	10	$\pm 1\%$
Vermelho	2	2	2	100	$\pm 2\%$
Laranja	3	3	3	1k	-
Amarelo	4	4	4	10k	-
verde	5	5	5	100k	$\pm 0.5\%$
Azul	6	6	6	1M	-
Violeta	7	7	7	10M	-
Cinzentos	8	8	8	-	-
Branco	9	9	9	-	-
Dourado	-	-	-	-	$\pm 5\%$
Prateado	-	-	-	-	$\pm 10\%$

Tabela 2.7: Tabela com a correspondência entre as cores das riscas das resistências e o seu valor.

- em paralelo, é o recíproco da soma do recíproco do valor de cada resistência: $1/Req = \sum_i R_i = 1/R_1 + 1/R_2 + \dots + 1/R_n$.

O valor da impedância de uma resistência pertencente a uma série de baixa potência é indicado pelo código de 4 ou 5 riscas de cores apresentado no seu corpo, tal como ilustrado na figura 2.13, e na tabela 2.7. As cores codificam os dígitos do valor da impedância, onde nas resistências com 4 riscas as duas primeiras compõem o valor com dois dígitos do valor (dezenas e unidades) e o terceiro o fator multiplicativo. As resistências de 5 riscas usam as três primeiras riscas para codificar o valor, e a quarta risca para o fator multiplicativo. Em ambas, a última risca corresponde à tolerância de fabrico das resistências, ou seja, por quanto o valor real diverge do valor ideal. Por exemplo, uma resistência com as riscas amarelo (4), roxo (7), vermelho ($\times 10^2$) e dourado (5%), corresponde ao valor $47 \times 10^2 = 4,7 \text{ k}\Omega$ com 5% de tolerância.

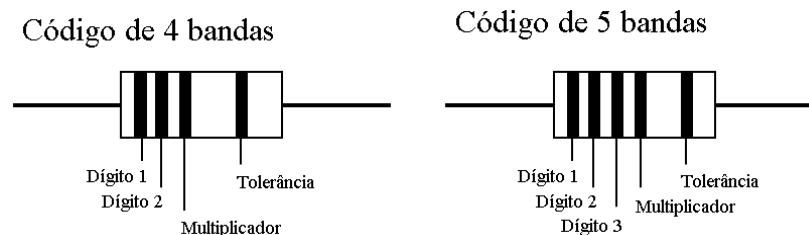


Figura 2.13: Código de cores das resistências.

Resistências variáveis são componentes cuja resistência varia de acordo com uma grandeza física. Esta grandeza física pode corresponder ao mo-

vimento de um veio que se desloca com movimento rectilíneo (reóstato), circular (potenciómetro), temperatura, luminosidade, etc.

A figura 2.14 mostra um caderno com amostras de uma série de resistências SMD. Este tipo de cadernos existe também para condensadores, sendo bastante útil para manter um *stock* de componentes organizados. As fitas com as resistências possuem furos de um dos lados para poderem ser usadas em máquinas de colocação automática de componentes (*pick & place*).



Figura 2.14: Caderno com amostras de todas as resistências de uma série de fabrico (esquerda). Detalhe das resistências de $68\ \Omega$ (direita).

Condensadores

Os condensadores permitem armazenar energia elétrica e são construídos por dois eletrodos ligados a cada um dos terminais, e estão em contacto com um eletrólito não condutor. Um condensador encontra-se inicialmente descarregado e, ao ser aplicada uma tensão aos seus terminais, começa a carregar. Quando a tensão fornecida deixar de estar disponível, o condensador começa a descarregar, fornecendo energia ao circuito. Uma das aplicações principais dos condensadores é a estabilização de fontes de alimentação CA para CC.

Existem diversos tipos de condensadores, dependendo do tipo de eletrólito e da sua tensão de funcionamento, sendo os mais habituais:

- sólido - baixa capacidade (pF - nF) e sem polaridade: papel, plástico, vidro, mica, e cerâmico,
- eletrolítico - elevada capacidade (μ F) e com polaridade: alumínio, e óxido de tântalo;

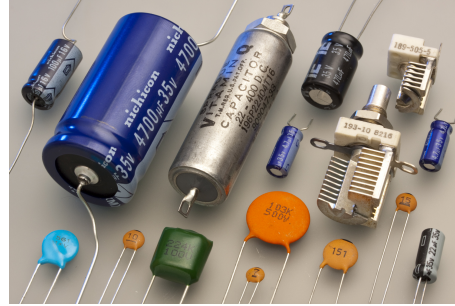
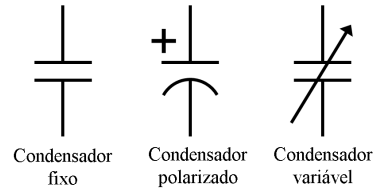


Figura 2.15: Símbolo e aspecto de condensadores fixos, polarizados e variáveis.

- ar - condensadores variáveis de baixa capacidade usados para sintonia, ou afinação, de circuitos RF (rádio/TV).

Ao construir um circuito é importante observar a polaridade dos condensadores eletrolíticos, indicada no seu corpo. Os condensadores eletrolíticos têm indicado o terminal negativo, mas os condensadores de tântalo têm indicado o terminal positivo. A troca da polaridade resulta na destruição, via explosão, do condensador. Os condensadores sólidos não possuem polaridade.

Tal como as resistências, os condensadores podem ser associados em paralelo ou em série para obter outros valores não existentes nas séries de fabrico. A capacidade resultante da associação de dois ou mais condensadores encontra-se ilustrada na figura 2.16 e é a seguinte:

- em paralelo, é a soma das capacidades: $C_{eq} = \sum_i C_i = C_1 + C_2 + \dots + C_n$;
- em série, é o recíproco da soma do recíproco das capacidades: $1/C_{eq} = \sum_i 1/C_i = 1/C_1 + 1/C_2 + \dots + 1/C_n$.

Como os condensadores não são perfeitos, apresentam um circuito equivalente com uma resistência parasita de baixo valor em série com o condensador. As características dos condensadores vêm definidos no *datasheet* do fabricante, sendo as mais relevantes, a capacidade do condensador, a tensão máxima de funcionamento e a resistência série parasita.

O valor da resistência série parasita difere entre tecnologias e capacidades dos condensadores, e é importante observar para circuitos onde o condensador fornece energia elétrica a cargas que consomem um valor elevado de corrente durante breves instantes, ou que necessitam de ter a sua alimentação bastante estável (equipamento áudio). Na realização prática de circuitos, associando vários condensadores em paralelo permite reduzir o efeito da resistência parasita em comparação com usar apenas um condensador de capacidade equivalente.

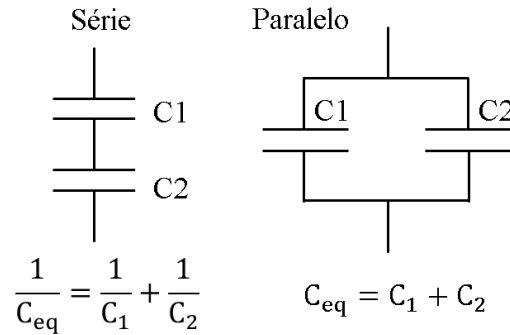


Figura 2.16: Associação de condensadores em série (esquerda) e em paralelo (direita).

Bobinas

A bobina é um componente com dois terminais que serve para armazenar corrente sob a forma de campo magnético, é constituído por um enrolamento de fio em torno de um núcleo, e não tem polaridade. As propriedades das bobinas dependem do uso de fios condutores de diferentes seções e materiais, enroladas sobre um núcleo de ar ou de ferro. Os condutores são isolados com verniz para reduzir o espaço entre os condutores. A figura 2.17 mostra diversos tipos de bobinas. Uma bobina é caracterizada pela sua indutância, medida em Henry (H). A indutância da bobina varia com: o número de espiras, a secção dos fios, e o material do núcleo, por exemplo ferro, plástico ou ar. A corrente máxima suportada pela bobina depende da secção do fio usado.

Ao armazenar corrente, a bobina irá “opor-se” à variação de CA que a atravessa. Devido a esta característica é usual para colocar uma pequena bobine (10 uH) em série com a entrada da alimentação em circuitos que produzem picos de corrente durante breves períodos de tempo, tais como os circuitos integrados digitais. Esta modificação faz com que os picos não se propaguem aos restantes componentes no circuito.

Outro exemplo de aplicação de bobinas é colocar uma bobina em série com a alimentação antes de condensadores de elevada capacidade e tensão para que este carregue mais lentamente. No instante em que um circuito é ligado e os condensadores não têm carga, eles comportam-se como um curto-circuito. Se a tensão de saída da fonte de alimentação for elevada, é como se os seus terminais fossem curto-circuitados até o condensador começar a adquirir carga. Para minimizar este efeito, uma bobina ligada em série com o condensador vai limitar a corrente que circula no circuito e carregar o condensador mais lentamente.

A bobina equivalente da associação de outras bobinas, quer em série ou em paralelo é obtida da mesma forma que para as resistências:

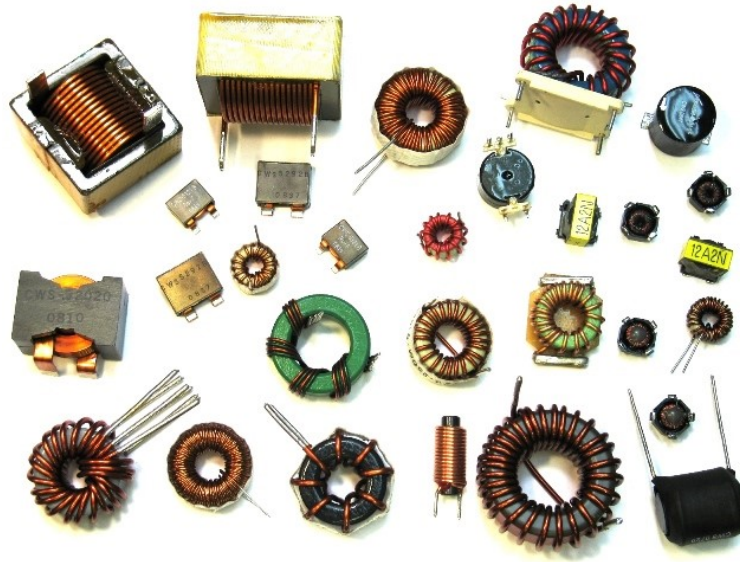


Figura 2.17: Vários tipos de bobinas.

- em paralelo, é a soma do recíproco das indutâncias: $1/L_{eq} = \sum_i 1/L_i = 1/L_1 + 1/L_2 + \dots + 1/L_n$.
- em série, é a soma das indutâncias: $L_{eq} = \sum_i L_i = L_1 + L_2 + \dots + L_n$;

Diodos

O díodo é um componente eletrônico com dois terminais, composta por dois elementos semicondutores dopados², um do tipo P (positivo) e outro do tipo N (negativo), para formar uma junção PN. O díodo é caracterizado por apenas permitir o fluxo de corrente num sentido, do terminal positivo, ou Ânodo (A), para o terminal negativo, ou Cátodo (K). Os diodos são caracterizados pela tensão de junção PN, que é a tensão positiva mínima presente aos terminais do diodo para que ele comece a funcionar (V_d); e não são lineares, ou seja, a corrente que os percorre tem um crescimento exponencial, em função da tensão aos seus terminais (V_d).

O funcionamento de um diodo é exemplificado com os dois circuitos da figura 2.18. À esquerda o diodo está montado no sentido direto e existe corrente a percorrer o circuito. À direita, com o diodo montado no sentido inverso não existe passagem de corrente no circuito, é como se a carga não estivesse ligada à fonte de alimentação. Tipicamente a tensão V_d é 0,7 V,

²A dopagem de material semicondutor passa por inserir impurezas no material, através de processos químicos, para que tenha maior concentração de eletrões (tipo P) ou menor concentração de eletrões (tipo N). Ao criar excesso num lado e falta de eletrões no outro, o material vai favorecer a migração de eletrões do positivo para o negativo

ou seja, um componente ligado com um diodo em série a VCC é alimentado com $VCC - 0,7\text{ V}$.

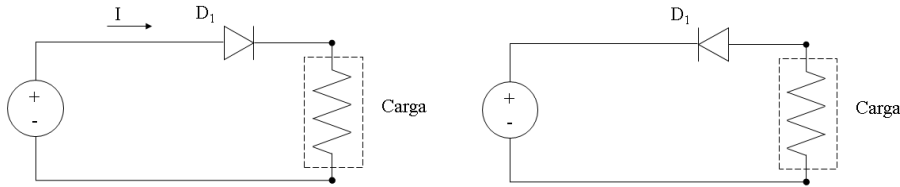


Figura 2.18: Montagem de um diodo num circuito em sentido direto (esquerda) e inverso (direita).

Os diodos ao deixarem passar a corrente apenas num sentido são usados principalmente como retificadores de CA para CC. Existem outros tipos de diodos fabricados com diferentes tipos de materiais semicondutores e processos físico-químicos, como por exemplo:

- retificador - apenas deixam passar corrente num sentido - são usados em conversores de CA para CC;
- Diodos emissores de luz - *Light Emitting Diode* (LED) - servem para emitir radiação luminosa numa frequência do espectro visível. O LED infravermelho foi inventado por James Biard em 1962, ao centro na figura ?? à esquerda;
- Fotodiodo - díodo sensível à luz, conduz na presença de radiação luminosa;
- Zener - produzem uma queda de tensão V_z (própria do díodo) no sentido inverso, e V_d no sentido direto;
- Schottky - têm a tensão V_d muito baixa (apx. $0,2\text{ V}$) e, por isso, são usados em retificadores;
- PIN - construídos com a inserção de material intrínseco (não dopado) entre a junção P e N (materiais dopados). São normalmente usados em RF, sistemas de comunicação por micro-ondas e fototransistores;
- Gunn - operam em muito altas frequências, e são aplicados em sistemas de comunicação por micro-ondas;
- Varicap - comportam-se como um condensador variável controlado por tensão.

A figura 2.19 mostra diferentes tipos de diodos e encapsulamentos.

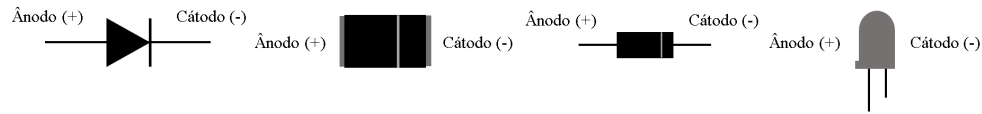


Figura 2.19: Símbolo do díodo retificador, e exemplos de vários tipos de diodos com diversos encapsulamentos.

Transistores

Os transistores servem para amplificar um sinal ou controlar outro circuito que requer uma corrente ou tensão superior à disponível no sinal de controlo. Os transistores são usados habitualmente na construção de amplificadores áudio ou comutadores de cargas tais como ventoinhas e relés. Um transistor é um componente com 3 terminais e composto por duas junções PN, e podem ser do tipo NPN ou PNP.

O primeiro transistor surgiu nos Bell Labs nos anos 40, sendo comercializados por volta dos anos 50-60 na Califórnia, dando origem a um local de inovação em eletrónica que ficou conhecido como “Silicon Valley”.

Existem diferentes tipos de transistores que são fabricados com diferentes materiais semicondutores e têm características e funcionamento diferentes, sendo os mais habituais em sistemas embebidos: Transistores de Junção Bipolar, do inglês Bipolar Junction Transistor (BJT), e Transistor de Efeito de Campo formado por Metal, Óxido e Silício, do inglês *Metal-Oxide-Semiconductor Field-Effect Transistor* (MOSFET). Os MOSFETs funcionam como um interruptor entre o Dreno (D) e a Fonte (S), controlado pela tensão aplicada na Porta (G) e a Fonte (S).

Os diferentes tipos de transistores são identificados através de símbolos, tal como ilustrado na figura 2.20.

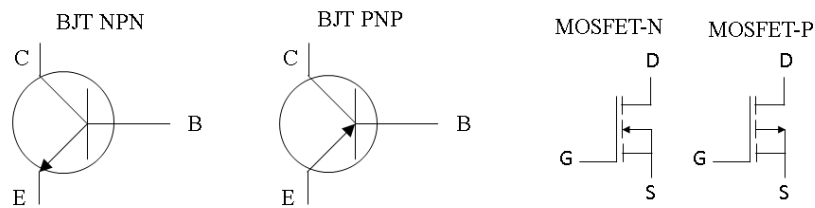


Figura 2.20: Símbolos elétricos dos diferentes tipos de transistores, da esquerda para a direita: BJT tipo-N, BJT tipo-P, MOSFET canal-N e MOSFET canal-P.

Os MOSFETs possuem 3 modos de funcionamento, mas com características diferentes dos transistores BJT:

- Modo de Corte - o transistor não está em funcionamento, ou seja, não existe passagem de corrente entre o Dreno e a Fonte;

- Modo Triodo - o transistor encontra-se em funcionamento e comporta-se como uma resistência controlada por tensão da Porta;
- Modo Ohmico - o transistor está ligado “no máximo”, ou seja, apresenta resistência mínima entre o Dreno e a Fonte (R_{DSon}). Esta zona de funcionamento acontece após a tensão da Porta exceder o valor limite da saturação, conhecido como V_{gs} Threshold.

O limiar das zonas de funcionamento depende das características dos transistores e da montagem do circuito. Em eletrónica analógica importam as diferentes zonas de funcionamento do transistor, especialmente a zona ativa no projeto de amplificadores, no entanto, em eletrónica digital, onde os sinais são GND ou VCC, apenas importam as zonas de funcionamento corte e saturação. No âmbito dos sistemas embebidos é frequente usar os transistores para controlo de componentes ou sistemas usando as zonas de funcionamento de corte e saturação. Por exemplo, para acionar um mostrador luminoso, uma ventoinha de arrefecimento ou um motor DC.

Existem várias montagens típicas com transistores, tais como Emissor-Comum, Base-Comum e Colector-Comum nos BJTs; e Fonte-Comum, Porta-Comum e Dreno-Comum nos MOSFETs. O transistor NPN numa montagem Emissor-Comum, tem o Emissor partilhado com os circuitos de entrada e de saída. A base do transistor encontra-se do lado do circuito de controlo e o colector do lado da carga. Nesta montagem quando a tensão da Base para o Emissor (V_{be}) é superior ao limite para o transistor entrar em condução ($V_{be\ threshold} \approx 0,7\ V$), passa a haver uma corrente que atravessa o transistor do colector para o emissor. Caso a tensão V_{be} seja menor que $0,7\ V$, o transistor encontra-se ao corte e não existe corrente a circular por ele.

Nos transistores do tipo PNP o funcionamento é o mesmo que nos NPN mas considerando a tensão V_{eb} inferior a $-0,7\ V$. Como o PNP é usado para controlar tensões positivas significa que ele fica ativo com uma tensão na base do transistor igual a $VCC - 0,7\ V$; Na montagem equivalente para os MOSFETs, Fonte-Comum, o circuito a controlar encontra-se ligado ao dreno, e o quando a tensão Porta-Fonte (Gate-Source ou V_{gs}) é próxima de $0\ V$ o transistor encontra-se ao corte. Quando a tensão V_{gs} é superior a $V_{gs\ threshold}$ (ver datasheet) o transistor fica em saturação.

Para o controlo de cargas que usam tensões e correntes superiores às suportadas pelos transistores são usados os tirístores ou variantes, como por exemplo os *TRIode for Alternating Currents* (TRIACs) (5 camadas de silício) e os *Silicon Controlled Rectifiers* (SCRs) (4 camadas de silício). A maior diferença entre ambos é que os TRIACs deixam passar CA nos dois sentidos ao passo que os SCR's apenas deixam passar num sentido. Como estes componentes estão ligados diretamente na rede elétrica, devem-se usar isoladores elétricos/galvânicos para proteger o sistema embebido, e os demais componentes nele ligados.

Circuitos Integrados

O primeiro circuito integrado foi inventado em 1949 por Jacob Werner ao colocar vários transistores sobre o mesmo substrato. Em Setembro de 1958, Jack Kilby e Robert Noyce, na figura ??, criaram o primeiro circuito integrado que viria a ser comercializado pela Texas Instruments, o que valeu a atribuição do prémio Nobel em física no ano 2000 a Jack Kilby.

A partir do momento em que foi possível criar dispositivos em silício contendo vários transistores não demorou muito tempo a reduzir o tamanho dos transistores e a integrar cada vez mais transistores na mesma pastilha de silício. O primeiro microprocessador fabricado numa única pastilha de silício foi criado em 1968 pela Viatron Computer Systems. O primeiro processador comercial foi o 4004 da Intel, na figura 2.21 à esquerda, em 1971, baseado em tecnologia desenvolvida na Fairchild Semiconductor. Este processador tinha 2300 transistores, barramento de 4 bits, funcionava a 0,74 MHz (740 kHz) e era destinado a ser usado numa calculadora.

Dada a evolução do fabrico de transistores cada vez mais pequenos, Gordon Moore previu que seria possível fabricar um novo chip a cada 2 anos com o dobro dos transistores. Isto é o equivalente a ter uma impressão de uma folha A4 toda preenchida com texto de tamanho 72, e a cada nova impressão ir reduzindo o tamanho da fonte para metade. A cada nova impressão é a quantidade de texto impressa na mesma folha é duplicada.

Esta tendência na capacidade de fabrico foi observada até cerca de 2012 porque passou a ser mais difícil fabricar transistores cada vez mais pequenos. Uma das tendências para contornar a limitações na produção de circuitos no futuro é o fabrico de circuitos com nanotubos de carbono. Em 2019, o processador da Intel Core i9-9990XE, na figura 2.21 à direita, é fabricado com cerca de 7 milhares de milhões de transistores de 14 nm (14×10^{-9} m), distribuídos por 14 núcleos, e funcionar a 5000 MHz (5 GHz).

Amplificadores Operacionais

O AMPlificador OPeracional (AMPOP ou OPAMP, derivado do inglês *OPerational AMPlifier*) é um circuito integrado composto por dezenas de transistores capaz de amplificar pequenos sinais à sua entrada milhões de vezes. Para além dos terminais de alimentação, positivo (VCC) e negativo (GND), o AMPOP tem duas entradas, terminal positivo e negativo, e uma saída. O AMPOP quando em funcionamento, em malha aberta, ou seja, sem nenhum circuito a ligar a saída à entrada, ele amplifica a diferença entre os sinais de entrada milhares de vezes.

Para que os AMPOPs possam ser usados em circuitos analógicos é necessário controlar o seu ganho, através de um circuito de realimentação que liga a saída à entrada, que faz com que o sinal não sature no valor máximo ou mínimo. O ganho de um amplificador com AMPOP é controlado através

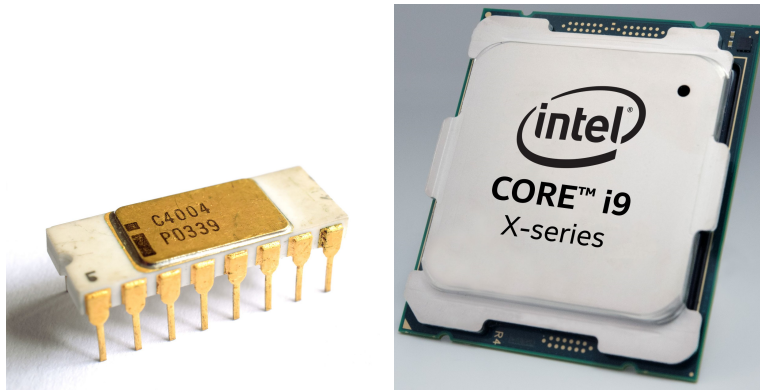


Figura 2.21: Primeiro processador da Intel de 1971, o 4004 funcionava com um barramento de 4 bits (esquerda), e um processador Core-i9 a 64 bits de 2019 (direita).

de um circuito de realimentação.

Para além dos circuitos amplificadores, o AMPOP é usado como detetor de nível de tensão. Numa montagem básica, sem realimentação, quando a tensão na entrada positiva do AMPOP é superior à entrada negativa, o AMPOP aumenta a diferença entre elas milhares de vezes e satura, gerando VCC na sua saída. Quando a tensão de entrada negativa é superior à positiva, o AMPOP satura negativamente e a sua saída passa a GND. Ajustando a tensão de entrada negativa com um circuito divisor de tensão é possível ter um sinal binário que é 1 quando a tensão da entrada positiva é superior, e 0 quando é inferior.

Microcontroladores

A figura 2.22 mostra os elementos que compõe do microcontrolador (MCU) ATmega328 da Atmel/Microchip. Um dos elementos do microcontrolador é o processador (CPU). Os detalhes da arquitetura do CPU existente dentro do MCU ATmega328 encontra-se na figura 2.23.

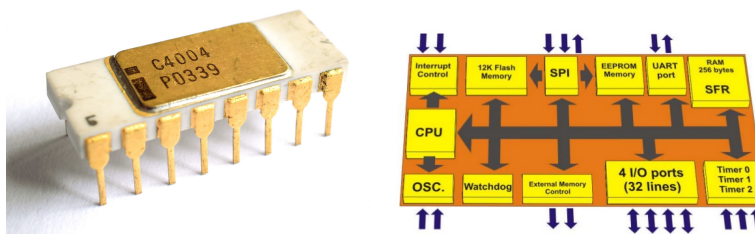


Figura 2.22: Arquitectura do microcontrolador ATmega328 da Atmel/Microchip.

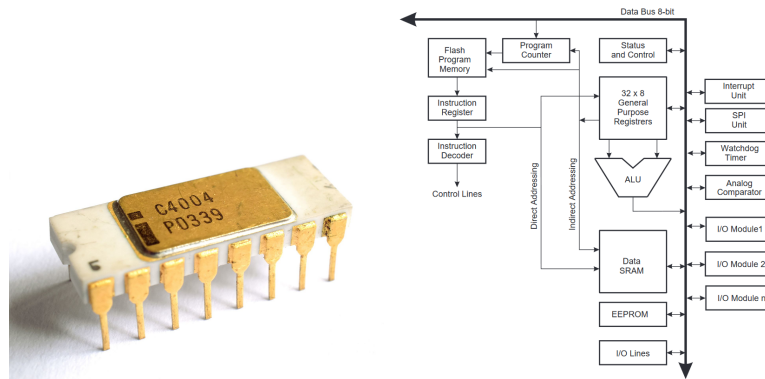


Figura 2.23: Arquitectura do microcontrolador ATmega328 da Atmel/Microchip.

2.2.6 Desenvolvimento de Circuitos

O desenvolvimento de circuitos eletrônicos implica não só conhecer a teoria de circuitos como também saber conhecer o comportamento dos componentes na prática para que se possam escolher os componentes eletrônicos mais adequados e combiná-los de forma a realizar a funcionalidade desejada. Circuitos com eletrônica analógica são mais complexos do que a digital, porque é necessário observar as tolerâncias dos materiais e componentes usados, a tensão, corrente e frequência dos sinais em cada componente do circuito; ao passo que o projeto de um sistema digital, a nível físico, “apenas” tem de respeitar a frequência máxima de funcionamento, passando o “esforço” para as funções lógicas a realizar. Ao nível dos sistemas embebidos, a eletrônica analógica é usada para fazer adaptação de circuitos ou condicionamento de sinais de sensores.

O primeiro passo para a realização de um circuito eletrónico é a sua especificação, começando por fazer um esboço dos blocos incluindo a sua funcionalidade e ligações. O passo seguinte passa por detalhar os componentes necessários, e realizar as ligações entre os seus terminais. Após o circuito “desenhado” em papel é necessário validá-lo através de ferramentas de simulação. Se a simulação estiver correta, pode-se começar a montar e testar o protótipo do circuito. Só depois destes passos se deve partir para uma montagem definitiva em placa de circuito impresso - Placa de Circuito Impresso - *Printed Circuit Board* (PCB).

A realização do protótipo normalmente é feita numa escala maior do que a do sistema final para facilitar o seu desenvolvimento, nomeadamente trocar componentes, corrigir ligações e ter espaço poder observar sinais com equipamento de teste e medida.

Os protótipos não definitivos são montados em placas de ensaio próprias chamadas *breadboard*, que traduzido à letra significa “tábua do pão”. Este nome deve-se ao facto de os primeiros circuitos serem realizados em tábuas

de madeira para cortar o pão, tal como ilustrado na figura 2.24, à esquerda³.

As *breadboards* contêm uma matriz de furos, onde os fios unifilares e os terminais dos componentes ficam presos por molas existentes dentro de cada furo, evitando ter de soldar os fios e os componentes. Internamente existem ligações entre os furos, tal como ilustrado na figura 2.24, à direita, com os riscos a indicar a ligação dos furos. As filas de topo e fundo estão ligadas na horizontal e são usadas para a alimentação do circuito: positivo (VCC) e negativo (GND). Nalgumas placas elas aparecem pintadas de vermelho e azul, sugerindo que a fila vermelha deve ser o positivo, e a fila azul o negativo. As filas do meio estão ligadas na vertical, ou seja, cada coluna vai servir de ligação para um sinal. O espaço no meio da *breadboard* é idêntico ao espaçamento entre as duas filas de pinos dos circuitos integrados com encapsulamento DIP, do inglês *Dual Inline Package*, e o espaçamento entre os furos é o mesmo que entre pinos do circuito integrado (0.1 polegada (inch) ou 100 mil = 2.54 mm). Nos lados das *breadboards* existem encaixes para juntar outras placas para poder acomodar circuitos maiores. As placas Arduino possuem uma base em acrílico com os mesmos encaixes que as *breadboards* para as manter juntas. Nos protótipos, onde normalmente a área não é um problema, podem-se empregar componentes mais volumosos e mais fáceis de manipular.

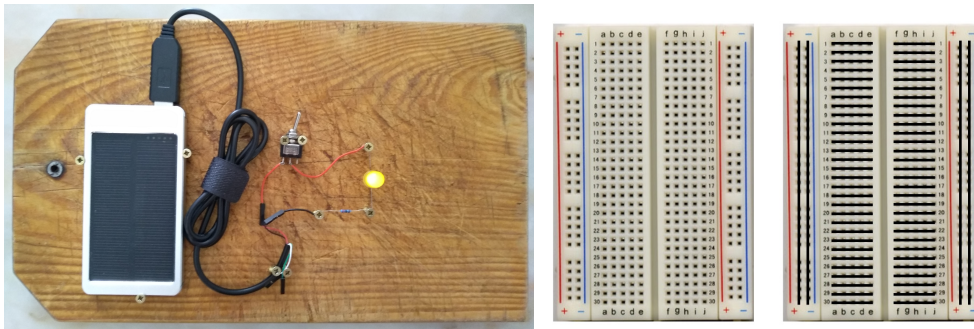


Figura 2.24: Antes das atuais placas de prototipagem os circuitos eram realizados em tábuas de pão (*breadboard*), à esquerda. Ao centro e à direita as atuais *breadboards* para prototipagem de circuitos eletrônicos, com as respectivas ligações internas.

A melhor abordagem para começar o desenvolvimento de qualquer sistema embebido é usar *kits* de desenvolvimento, semelhantes aos do Arduino, pois dão a garantia de um circuito mínimo estar a funcionar, e ter algumas demonstrações para facilitar o desenvolvimento do resto do sistema.

Após a validação do funcionamento correcto do circuito pode-se passar

³Como ainda não existiam caixas próprias para circuitos elétricos, os rádios e amplificadores a válvulas eram construídos em tabuleiros metálicos (alumínio) por estarem facilmente disponíveis e serem fáceis de adaptar para acomodar os circuitos

CAPÍTULO 2. CONCEITOS BÁSICOS

para uma construção definitiva usando uma placa de circuito impresso, do inglês PCB, e componentes mais pequenos para montagem em superfície SMD. Uma PCB é uma sandes de várias fatias de material isolante a separar camadas de material condutor (cobre). As camadas de cobre são recortadas de forma a criar as pistas para ligação entre os pinos dos componentes que ficam colocados na superfície da placa. Na fabricação da PCB, o cobre em excesso, entre as pistas desenhadas, é removido, ficando apenas o cobre das pistas. Depois de furada, envernizada e adicionada a serigrafia (marcações do lugar dos componentes com tinta branca), os componentes podem soldados na PCB nos seus lugares. Como alguns componentes encaixam de várias posições, é necessário ter atenção à sua orientação. A soldadura dos pinos dos componentes é suficiente para os manter seguros à PCB, no entanto se os componentes forem puxados as soldaduras vão fazer arrancar as pistas de cobre da PCB.

Ferramentas de Desenvolvimento

As ferramentas de desenvolvimento de circuitos servem para desenhar os esquemas elétricos, simular os circuitos (ex. SPICE) e desenhar as várias camadas das placas de circuito impresso. Muitas ferramentas são gratuitas, normalmente destinadas ao ensino e uso pessoal, outras para profissionais e produtividade são pagas. Algumas destas ferramentas são: Fritzing, Eagle, gEDA, KiCAD, AutoTRAX, Pads, OrCAD, DesignSpark, CircuitMaker, Cadence, e Altium. Os circuitos apresentados neste livro foram desenhados em Fritzing.

A figura 2.25 ilustra o projecto de um circuito realizado com o Fritzing, incluindo o esquema elétrico, a vista da montagem em *breadboard*, e o desenho da placa de circuito impresso para um *shield* Arduino.

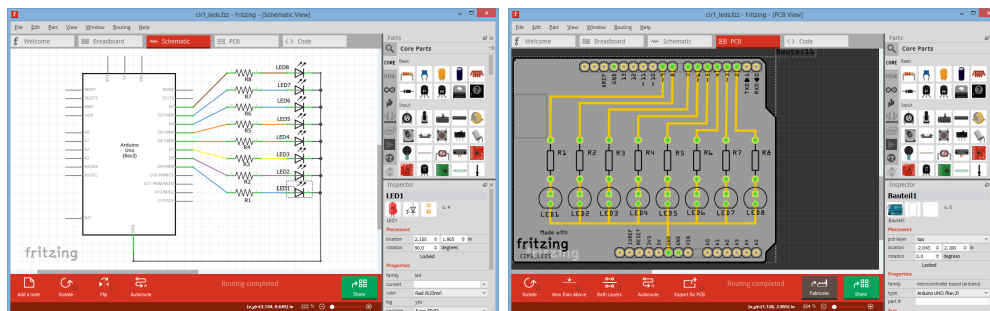


Figura 2.25: Aplicação Fritzing em uso para o desenho de esquemas elétricos (esquerda) e PCBs (direita).

Placas de Circuito Impresso

Para que a montagem do circuito fique segura e robusta é necessário montar os componentes numa PCB, semelhante ao que encontramos nos equipamentos eletrónicos de consumo e industriais. Uma PCB é constituída por pistas de material condutor sobre uma placa de material isolante (fibra de vidro ou baquelite). As pistas são usadas para ligar os pinos dos vários componentes. Os pinos dos componentes que atravessam a placa necessitam de ser encaixados em furos com a sua região envolvente em cobre (ilha) para permitir a soldadura do pino à pista. A PCB pode ter uma ou mais camadas ou planos de condutores, sendo os componentes apenas soldados nas camadas externas (topo e fundo). Os desenhos das várias camadas são produzidos pelas ferramentas de desenho de circuitos referidas anteriormente. Em muitas ferramentas o desenho da PCB pode ser gerado automaticamente, mas é necessário instruir a ferramenta através de regras para garantir que o desenho fica coerente com o funcionamento do circuito, nomeadamente a topologia das derivações, a largura das pistas de alimentação, a criação de planos de GND, e o espaçamento entre pistas.

O processo é iniciado com a introdução do esquema eléctrico e a criação de todas as ligações existentes no circuito no ambiente de desenvolvimento Fritzing, adicionando os símbolos dos componentes e as ligações entre os seus pinos e os de outros componentes.

O desenho da PCB pode ser feito automaticamente com base na lista de ligações (*netlist*) que permite colocar os componentes, a partir de uma biblioteca de componentes, incluindo as ligações aos seus terminais. Neste processo também é feita a substituição dos símbolos dos componentes pelos seus *footprints*, e são feitas as pistas com base nas ligações existentes no esquema eléctrico, exportadas para a *netlist*.

A figura A.2 ilustra os *footprints* mais comuns de circuitos integrados para montagem em superfície nas PCBs.

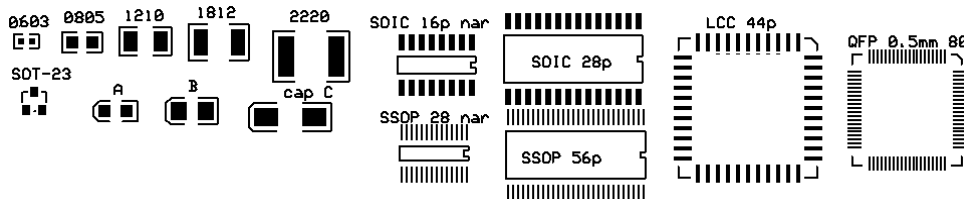


Figura 2.26: Exemplos de *footprints* SMD (não estão à escala).

As PCBs são fornecidas com toda a face coberta por um plano de cobre. As pistas são formadas através de um processo químico para remoção do cobre em excesso. A marcação do cobre a ser removido é feita através de um processo fotossensível, onde o desenho desse plano é gravado pela exposição a radiação ultra-violeta. Para produção em ambiente industrial ou em grande

CAPÍTULO 2. CONCEITOS BÁSICOS

volume, a indústria faz uso de vários tipos de ficheiros, independentes das ferramentas de desenho, para permitir a interoperabilidade dos vários sistemas, nomeadamente os Gerber. Uma alternativa ao processo foto-químico é fresar o excesso de cobre da superfície da PCB.

Nas PCBs com duas ou mais camadas, é possível colocar pequenas vias que estabelecem a ligação entre os vários planos. Em PCBs com três ou mais camadas, é possível criar vias entre dois dos planos (*blind and buried vias*). A figura 2.27 mostra o aspecto de uma PCB de um circuito de controlo de um iluminador de infravermelhos ainda sem componentes soldados. A PCB está coberta por um verniz verde que protege o cobre da oxidação e de contactos elétricos acidentais. A zona destinada a soldar os terminais dos componentes (*pads*) está descoberta de verniz e está pré-estanhada para facilitar a soldadura dos componentes. Na PCB é possível observar a marcação da posição dos componentes (serigrafia) para auxiliar a soldadura manual dos componentes. Os componentes com polaridade têm indicação da sua orientação.

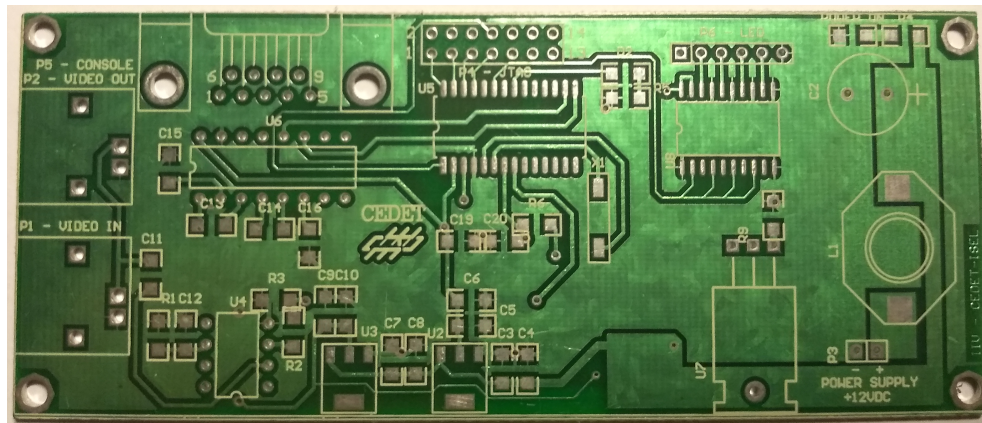


Figura 2.27: Vista da face superior de uma PCB de duas camadas após fabricação.

Algumas estratégias de desenho para placas de circuito impresso:

- Se usar mais do que duas camadas, reservar um par de camadas para a alimentação do sistema: VCC e GND;
- Caso a placa tenha apenas duas camadas, ocupar o espaço livre com plano de GND para reduzir o ruído parasita;
- Distribuir o VCC usando uma topologia em estrela - usar um ponto para derivar pistas independentes para as alimentações dos diferentes componentes;
- Usar condensadores de desacoplamento de 100 nF perto dos pinos de alimentação de todos os circuitos integrados. Nos circuitos digitais

usam-se também bobinas de 10 μH em série com a pista de alimentação para minimizar os picos gerados pelo funcionamento dos componentes;

- Os planos de GND digital e analógico apenas se tocam num ponto, e perto do terminal da alimentação, para reduzir a injeção de ruído dos circuitos digitais nos sinais analógicos;
- Separar/agrupar os componentes de acordo com os seus sinais/barramentos e funcionalidade - tornar as pistas o mais curtas possível;
- Prestar atenção à energia dissipada sob a forma de calor, especialmente nos reguladores de tensão e nos transistores de controlo de cargas que consomem elevada potência;
- Usar resistências de 0R para deixar opções de configuração no circuito, para não ter de cortar pistas ou soldar fios.

A figura 2.28 ilustra o detalhe da vista lateral de uma secção de uma PCB de quatro camadas. Nesta imagem observam-se as camadas de topo e fundo, onde os componentes são soldados, e entre estas duas, encontra-se uma pilha de camadas condutoras alternadas com camadas isolante. Na figura são visíveis também as vias de ligação entre as várias camadas. Algumas das vias estabelecem contacto elétrico nas camadas interiores.

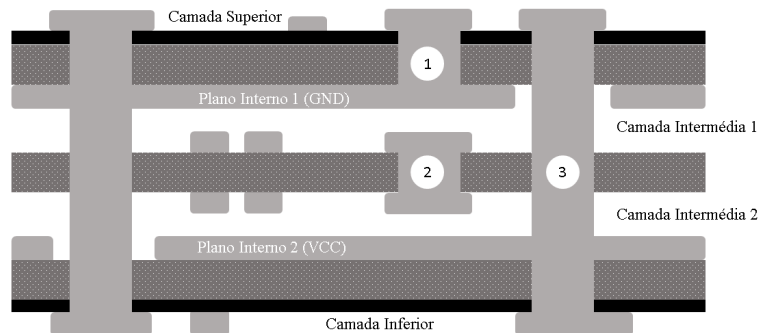


Figura 2.28: Exemplo de uma placa de circuito impresso de 4 camadas.

As ferramentas de desenvolvimento possuem bibliotecas com diferentes *footprints*, correspondentes a componentes com diferentes encapsulamentos. Para dar ao projetista alguma sensibilidade em relação às dimensões reais das PCBs existem demonstradores disponibilizados por fabricantes de PCB ou vendedores de componentes com exemplos de pistas e componentes na PCB. A figura 2.29 mostra os detalhes de uma régua realizada em PCB com exemplos de vários *footprints* típicos de componentes SMD, pistas com várias grossuras e espaçamentos, e furos de várias dimensão.

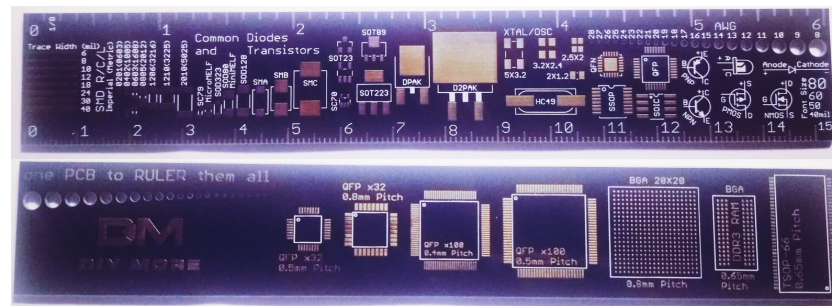


Figura 2.29: Faces superior e inferior de uma placa de circuito impresso com demonstração da capacidade de fabricação, espaçamento e espessura das pistas e tamanho dos furos.

Soldadura

A soldadura é realizada através da aplicação de solda de estanho derretido a alta temperatura como forma de ligação e fixação dos componentes e fios nos *pads* da PCB, de forma a que o circuito fique ligado permanentemente.

Para realizar a soldadura é necessário um ferro de soldar, que aquece a temperaturas entre 250°C e 350°C. Em alternativa pode-se usar uma estação de soldadura, que permite ajustar qual a temperatura da ponta do ferro e costuma ter acessórios tais como soprador de ar quente, para soldadura com pasta de solda. A figura 2.30 mostra uma estação de soldadura com ferro de soldar e soprador de ar quente, à esquerda, e um rolo de solda, à direita.

A solda é constituída por estanho misturado com um composto à base de resina *flux*. Com o calor da ponta do ferro de soldar, o *flux* ajuda o estanho a liquidificar e a aderir à PCB e aos componentes. Uma alternativa à solda de estanho é a solda em pasta, onde pequenas partículas, ou limalhas, de estanho estão imersas numa pasta com *flux*. A solda em pasta é aplicada com uma seringa assistida por um sistema pneumático para empurrar o êmbolo da seringa com precisão e dispensar apenas uma pequena quantidade sobre cada *pad* dos componentes.

Em ambiente industrial, o processo de soldadura dos componentes começa com a colocação da solda em pasta por uma máquina em todos os pads a soldar. De seguida os componentes são colocados automaticamente sobre a placa nos sítios a soldar, por máquinas *pick and place* e são depois encaminhadas por um tapete rolante para a máquina de soldadura. A máquina de soldadura (por onda) possui um tanque de solda quente, em estado líquido, por onde as placas são passando e ficando com os terminais dos componentes soldados na PCB.

A operação de soldadura manual é realizada aquecendo, com a ponta do ferro de soldar, o terminal do componente em simultâneo com a respectiva ilha na PCB. Encosta-se a ponta do ferro nestes dois pontos durante um a dois segundos, para que os materiais possam adquirir alta temperatura,



Figura 2.30: Estação de soldadura (esquerda) e rolo de solda (direita).

e de seguida derrete-se um pouco de solda (apx 5mm do fio) na ponta do ferro. Aguarda-se cerca de um a dois segundos para que a solda liquidifique, e pareça homogénea. Neste momento pode-se mover ligeiramente a ponta do ferro para distribuir a solda uniformemente pelo ponto de soldadura. Por fim, retira-se a ponta do ferro de soldar e espera-se que arrefeça.

É importante que o terminal do componente não seja movido durante esta operação, sob pena da soldadura ficar imperfeita e quebradiça. Deve-se ter em atenção as elevadas temperaturas usadas no processo. Deve-se usar um alicate para segurar os terminais dos componentes a soldar. Isto permite retirar algum calor do componente e não o danificar, como também evitar que se queimem os dedos!

Existem bases de apoio para PCBs que possuem pinças para segurar a PCB nas posições mais convenientes para soldar. A figura 2.31 ilustra o processo de soldadura descrito anteriormente. No caso de ocorrer um erro na soldadura, ou seja necessário a remoção e troca de um componente danificado, usa-se malha de dessoldadura para absorver a solda e assim libertar o terminal. Uma alternativa que requer mais perícia, é “encharcar” em solda as soldaduras do componente e manter a ponta do ferro na solda para esta ficar no estado líquido e poder-se soltar o componente.

2.2.7 Verificação de Circuitos

Na realização de um circuito eletrónico deve ser adotada uma abordagem incremental e modular, em que à medida que se vai construindo as diversas partes do circuito, elas devem ser verificadas individualmente. Os primeiros componentes a serem soldados e testados devem ser os dos circuitos de alimentação para garantir que todos os componentes recebem a alimentação

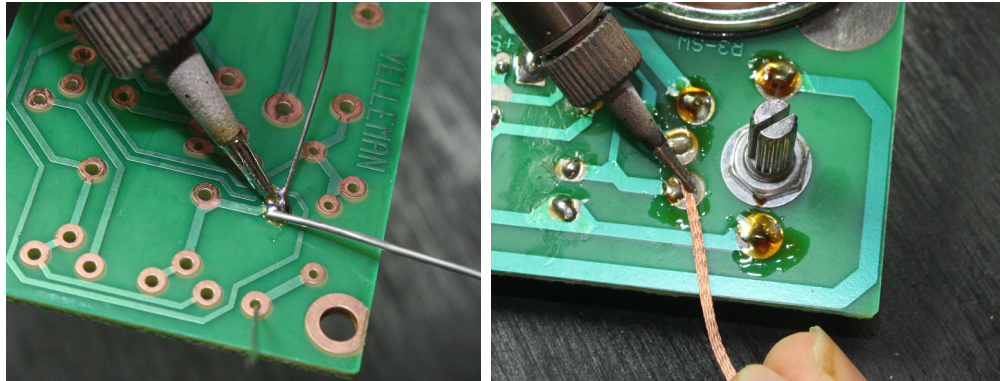


Figura 2.31: Exemplo de soldadura (esquerda) e dessoldadura (direita) de um terminal de um componente numa PCB.

com a tensão e a polaridade corretas.

Inspecção Visual

Ainda antes de se ligar um circuito à fonte de alimentação, deve ser feita uma inspeção visual para procurar curto-circuitos, componentes mal soldados, ou soldados em posições erradas. Deve-se usar um multímetro para verificar a existência de curto-circuitos na alimentação do circuito e noutros pontos de alimentação do circuito (valores de impedância abaixo dos $5\ \Omega$ podem significar problemas). Para componentes com dimensões reduzidas pode ser necessário recorrer a uma lupa.

Multímetro

Um multímetro é um equipamento que permite medir pelo menos as três grandezas fundamentais em eletrónica: tensão, resistência e corrente. Alguns multímetros medem também continuidade, diodos, transistores, e temperatura. O multímetro é composto por:

- Mostrador numérico - para ler o valor da grandeza medida;
- Selector - para escolher a grandeza a escolher, e a escala;
- Bornes/Alveolos - para ligar as pontas de prova ao circuito a testar - Habitualmente existem 3: comum ou negativo (preto), positivo para medir tensão (vermelho), e positivo para medir corrente (vermelho).

Os pontos para ligação das pontas de prova para medir corrente têm um fusível para proteger o multímetro.

Os parâmetros que definem um multímetro, e que devem ser tidos em conta aquando da sua compra são:

- Função: medição de tensão, corrente em AC e DC, resistência, teste de continuidade, diodos, transistores e temperatura;
- Escala (*range*): gama de valores medidos. Alguns aparelhos possuem escala automática evitando a selecção da escala manual (ex. 200 mV, 2 V, 20 V);
- Resolução - número de dígitos lidos (ex. 3 1/2 dígitos mostra valores entre 000 e 1999);
- Precisão/Tolerância - quão certo/errado é o resultado apresentado, quando comparado com equipamento de referência/calibração.

A figura 2.32 apresenta dois exemplos de multímetros, um de uso geral com as respectivas pontas de prova e outro para medir a alimentação e o consumo de periféricos USB nele ligados.



Figura 2.32: Multímetro de uso geral (esquerda) e multímetro para periféricos USB (direita).

Osciloscópio

O osciloscópio é um equipamento que permite visualizar a variação de um sinal eléctrico ao longo do tempo. O osciloscópio tem um ecrã para mostrar o sinal como um gráfico, onde a amplitude é representada no eixo vertical, e o tempo no eixo horizontal, da esquerda para a direita. Os osciloscópios possuem também botões de controlo para ajustar a visualização, nomeadamente escala vertical e horizontal, a origem do sinal apresentado, e a sincronização do sinal no ecrã. A figura 2.33 mostra um osciloscópio digital de 4 canais e uma ponta de prova com os respectivos acessórios para calibração e fixação da ponta nos terminais de componentes e fios. O osciloscópio tem um gerador

CAPÍTULO 2. CONCEITOS BÁSICOS

de 1 kHz de qualidade para ser usado como sinal de calibração da ponta de prova. A calibração passa pela afinação de um cursor com uma chave de fendas de plástico, até que a onda esteja perfeitamente quadrada. A ponta de prova pode ter um seletor de atenuação para reduzir a amplitude do sinal de entrada em 10x ou 100x. A garra da ponta de prova deve ser ligada ao GND do circuito a testar, e a ponta deve tocar num contacto metálico (pista ou terminal de componente) do sinal a visualizar.



Figura 2.33: Osciloscópio digital de 4 canais (esquerda), e ponta de prova de osciloscópio e respetivos acessórios (direita).

Um osciloscópio é um equipamento complexo típico de bancada de laboratório, o seu custo elevado (> 250 €). Os aparelhos mais sofisticados são capazes de realizar medidas, analisar os sinais, e decodificar sinais de canais de dados. No entanto, existem algumas alternativas mais económicas, nomeadamente os osciloscópios USB para visualização no computador, e aplicações para telemóveis.

O osciloscópio baseado em aplicação para telemóvel faz uso da entrada do sinal do microfone para adquirir o sinal. O sinal é depois mostrado na aplicação como se tratasse de visualizar o sinal áudio adquirido pelo telemóvel. Este método é atrativo porque evita a compra de um osciloscópio, no entanto a grande limitação deste método é a reduzida largura de banda que o canal áudio possui ($10\text{ Hz} - 20\text{ kHz}$), poucas funcionalidades em comparação com um osciloscópio tradicional, mostrador pequeno, dificuldade na calibração da ponta de prova e requer cuidado na ligação a sinais com tensões superiores às do sinal do microfone, sob pena de danificar o telemóvel. O circuito do telemóvel não é capaz de medir tensões CC devido a um circuito que rejeita sinais CC e apenas deixar passar sinais CA, como o áudio.

A figura 2.34 mostra as ligações para um *jack* áudio com quatro terminais: saída esquerda (L), saída direita (R), GND e entrada (MIC). Como os microfones habitualmente são componentes passivos - podem ser vistos como uma resistência que varia com o som e por isso, necessitam de uma tensão de polarização (entre 1,8 V a 2,9 V), pelo que é recomendado colocar um par de diodos em paralelo, com orientações opostas, entre o sinal de entrada e

GND (circuito *Clipping*), para não deixar que a tensão ultrapasse o limite máximo. O circuito *clipping* é explicado no capítulo ??.

Para se poder analisar sinais com amplitude superior a 2 V, usou-se uma ponta de osciloscópio com atenuação de 100x. Isto vai fazer com que os sinais de menor amplitude sejam também reduzidos 100x. Esta atenuação pode ser compensada na aplicação (e nos aparelhos de bancada) indicando o fator de escala. A figura 2.35 mostra a demonstração do funcionamento do osciloscópio para telemóvel, ou tablet, usando uma ponta de 100x de atenuação na entrada do microfone, ligada a um pino de saída do Arduino. Foi usada a aplicação “Oscilloscope”, no entanto existem diversas aplicações capaz de visualizar o sinal da entrada de microfone. O potenciômetro de 10 k Ω está ligado como divisor de tensão, para atenuar a tensão de entrada.

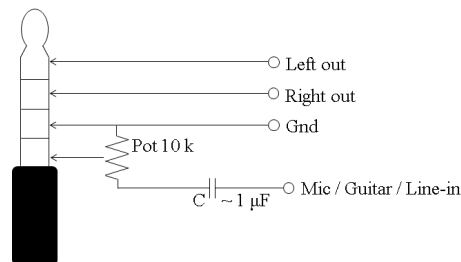


Figura 2.34: Circuito de adaptação do conector áudio/mic do telemóvel para ponta de osciloscópio.

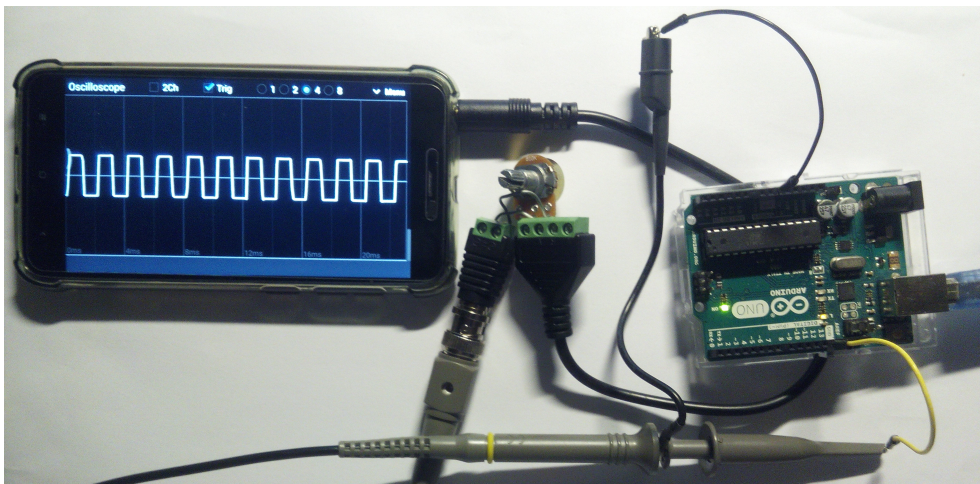


Figura 2.35: Demonstração da aplicação Android para funcionar como um osciloscópio digital usando a entrada de microfone do telemóvel.

Os sinais apresentados neste livro foram obtidos com um osciloscópio USB, que é assistido por uma aplicação que é executada no computador para visualizar os sinais.

Gerador de Sinais

O gerador de sinais é capaz de criar sinais periódicos com diferentes amplitudes, frequências e formas (onda sinusoidal, quadrada, triangular, e rampa), e são usados para substituir outras fontes de sinal durante ensaios a circuitos.

Antes de colocar um circuito completo em funcionamento nas condições finais, deve-se testar cada um dos seus blocos isoladamente, para garantir que todos funcionam corretamente e produzem o resultado esperado. O gerador de sinais vai servir para estimular a entrada de cada bloco a testar.

Existem diversos tipos de geradores de sinal de bancada, sendo caracterizados pelas formas de sinal que são capazes de gerar, e as gamas de frequências. Também aqui surgem alternativas, tal como para a visualização de sinais, com o telemóvel ou via USB, havendo as mesmas limitações que enunciadas anteriormente. Neste caso, a saída do sinal para além de limitada à gama de frequências 10 Hz a 20 kHz e amplitude máxima de 2,2 Vpp.

2.3 Programação

Esta secção aborda alguns dos aspetos mais importantes no desenvolvimento de programas para sistemas embebidos, e as particularidades da programação para o Arduíno.

A limitação de recursos de memória e desempenho de um sistema embebido oferece aos programadores desafios que habitualmente não têm num computador pessoal. Pode até acontecer um programa pode funcionar num computador mas não num sistema embebido por falta de memória ou não conseguir ser executado em tempo útil. Ao escrever um programa para um sistema embebido é necessário ter em mente a arquitectura do CPU usado.

Esta secção procura oferecer uma revisão dos conceitos mais relevantes na programação C/C++ para programação de sistemas embebidos e não introduzir a linguagem de programação. Se o leitor necessitar dos fundamentos, existem diversas publicações sobre o tema, nomeadamente em https://en.wikibooks.org/wiki/C_Programming.

2.3.1 C/C++ e Arduíno

A programação da plataforma Arduíno usa a sintaxe C/C++ na sua programação. Para adquirir conhecimento a nível de programação C/C++ recomenda-se a consulta de bibliografia sobre o assunto. Um programa para Arduíno é chamado *Sketch* (esboço ou rascunho) e é guardado em ficheiro com a extensão INO. A maior diferença entre o código fonte de um programa C/C++ standard para um Sketch Arduíno passa pela rotina principal. Em C/C++ a função principal chama-se `main()`, e tem de existir obrigatoriamente em todos os programas e com esse nome. No Arduíno existem duas

funções que são sempre chamadas: a primeira serve de preparação e iniciação de periféricos e variáveis, chamada `setup()`, e a segunda chamada `loop()` que fica em execução repetindo-se ciclicamente.

Para facilitar o desenvolvimento de aplicações, a plataforma Arduino inclui bibliotecas pré-instaladas que facilitam a interação com componentes externos ao Arduino. As bibliotecas são associadas ao programa através de ficheiros *header*, por exemplo `#include <arduino.h>`

Internamente, no IDE do Arduino o Sketch é adaptado para um projecto C/C++ e são gerados ficheiros C/C++ (.CPP) temporários para auxiliar a compilação de “forma tradicional” com um compilador adequado ao *Instruction Set Architecture* do MCU usado. A listagem 2.1 mostra o conteúdo da função `main()` no ficheiro `Arduino/hardware/arduino/avr/cores/arduino/main.cpp` onde são invocados os métodos `setup()` e `loop()`. Estes detalhes são ocultados do utilizador e trazem a vantagem do Sketch (ou projeto C/C++ Arduino) ficar com menos ficheiros e ocupar menos espaço. Se o leitor tiver curiosidade pode encontrar estes ficheiros temporários na pasta de ficheiros temporários do sistema operativo, por exemplo: `C:/Users/Rui/AppData/Local/Temp/arduino_build_452066`).

Listagem 2.1: Conteúdo do ficheiro `main.cpp` no IDE Arduino

```
int main(void)
{
    init();

    initVariant();

#ifdef USBCON
    USBDevice.attach();
#endif

    setup();

    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }

    return 0;
}
```

Após a compilação do Sketch, o compilador produz um ficheiro binário com extensão ELF (não dá para ler com um editor de texto), que contém o programa pronto a ser descarregado para o MCU do Arduino. O IDE Arduino permite também descarregar o programa no formato Intel HEX (Sketch→Export Compiled Binary) para ser usado com outros programadores compatíveis com o MCU do Arduino. O ficheiro pode ser encontrado na pasta do Sketch (Sketch→Show Sketch Folder).

Para depuração de erros de programação pode ser interessante observar onde é que cada variável ou instrução ficou no programa. Essa inspeção pode ser feita recorrendo a um desassemblador. O leitor pode encontrar um desassemblador capaz de interpretar o *Instruction Set Architecture* da família de processadores AVR em: <https://onlinedisassembler.com/>.

2.3.2 Tipos e Estruturas de Dados

As variáveis são os elementos que guardam os dados da aplicação. Uma aplicação pode registar a informação sob a forma de variáveis e constantes em memória volátil RAM ou constantes em memória de programa ou não-volátil ROM/FLASH. A diferença entre as duas é que o conteúdo da RAM perde-se após desligar a alimentação do circuito, ao passo que o conteúdo da ROM não desaparece.

O espaço de memória ocupado por uma variável ou constante depende do seu tipo, que por sua vez limita a informação nela contida. A tabela 2.8 resume os tipos de dados disponíveis na linguagem C/C++. O número de bytes que uma variável ocupa em memória é dado pela função `sizeof(nome do tipo)`.

Se uma variável do tipo `long` fosse usada para guardar valores que estão contidos entre 0 e 100 não seria uma boa opção pois uma variável do tipo `char` (1 byte) seria suficiente para a mesma funcionalidade, ao passo que uma variável do tipo `long` ocupa 4 bytes. Como os sistemas embebidos são bastante limitados em termos de memória disponível, se não houver cuidado na escolha dos tipos de dados facilmente se esgota a memória disponível.

Para facilitar a programação, o Arduino definiu novos tipos de dados, que na realidade são *Macros* em C, existentes no ficheiro `arduino.h`, e que permitem ter nomes de tipos de variáveis mais compactos, listagem 2.2.

Listagem 2.2: Tipos de variáveis definidos pelo Arduino

```
typedef unsigned int word;
typedef bool boolean;
typedef uint8_t byte;
```

A listagem 2.3 contém o código de um Sketch Arduino para mostrar quantos bytes são ocupados pelas variáveis de cada tipo, nativas do C/C++ e definidas pelo Arduino.

Listagem 2.3: Listagem do Sketch `varTypes` para mostrar no terminal de texto a ocupação em memória de cada tipo de variável

```
void setup() {
  Serial.begin(9600);
  while (!Serial) ; // wait for serial port to connect. }

  Serial.println("Var Type      Num Bytes");
  Serial.println("::integer numbers stdint/C99::");
  Serial.print("int8_t: \t");      Serial.println(sizeof(int8_t));
  Serial.print("int_fast8_t: \t");  Serial.println(sizeof(int_fast8_t));
  Serial.print("int_least8_t: \t"); Serial.println(sizeof(int_least8_t));
  Serial.print("uint8_t: \t");      Serial.println(sizeof(uint8_t));
  Serial.print("uint16_t: \t");     Serial.println(sizeof(uint16_t));
  Serial.print("uint32_t: \t");     Serial.println(sizeof(uint32_t));
  Serial.print("uint64_t: \t");     Serial.println(sizeof(uint64_t));
  Serial.print("intmax_t: \t");     Serial.println(sizeof(intmax_t));

  Serial.println("::integer numbers ANSI C/C89::");
  Serial.print("char: \t");        Serial.println(sizeof(char));
  Serial.print("short: \t");       Serial.println(sizeof(short));
```

```

Serial.print("int: \t");      Serial.println(sizeof(int));
Serial.print("long: \t");     Serial.println(sizeof(long));
Serial.print("long long: \t"); Serial.println(sizeof(long long));

Serial.println(".:real numbers IEEE-754:.");
Serial.print("float: \t");     Serial.println(sizeof(float));
Serial.print("double: \t");    Serial.println(sizeof(double));

Serial.println(".:pointers:.");
Serial.print("uint8_t*: \t");  Serial.println(sizeof(uint8_t *));
Serial.print("uint16_t*: \t"); Serial.println(sizeof(uint16_t *));
Serial.print("uint32_t*: \t"); Serial.println(sizeof(uint32_t *));
Serial.print("uint64_t*: \t"); Serial.println(sizeof(uint64_t *));

Serial.println(".:Arduino:.");
Serial.print("bool: \t\t");    Serial.println(sizeof(bool));
Serial.print("byte: \t\t");    Serial.println(sizeof(byte));
Serial.print("word: \t\t");    Serial.println(sizeof(word));
}

void loop() {
  delay(500); // do nothing for 500ms
}

```

O número de bytes ocupado por variáveis dos diferentes tipos é mostrado na tabela 2.9.

Classes

As classes estão na base da programação orientada a objetos. Uma classe é uma espécie de tipo de dados semelhante a uma **struct** em C++, que agrega atributos e métodos. Quando é criada uma variável com o nome da classe diz-se que é um objecto ou uma instância da classe. Os objetos possuem os atributos e os métodos definidos na classe que lhes deu origem. Por omissão os atributos e métodos herdados ou da classe são privados (apenas acedidos por métodos da classe), ao contrário do que acontece nas estruturas que são públicos (qualquer método do programa acede). As classes podem ter atributos ou não, mas todas devem ter pelo menos um método construtor (tem o mesmo nome da classe), mesmo que não execute nenhuma instrução. O construtor é usado para tratar das iniciações do objeto, no mesmo espírito do método **setup()** no Arduino.

O uso de classes em sistemas embebidos traz a vantagem de poder encapsular sobre uma única estrutura de dados variáveis e métodos necessários para interagir com um periférico de um determinado tipo e não ter a preocupação da organização das suas variáveis e métodos. Esta vantagem estende-se para os casos onde existem vários componentes com a mesma funcionalidade, e é necessário separar o estado de cada um deles.

A listagem 2.4 mostra um exemplo de um Sketch com uma classe criada para controlar um LED. Esta classe contém dois atributos, o pino (pin) e o estado do LED (state), e os métodos **setON()**, **setOFF()** e **getState2**, para além do construtor. Os atributos são privados para evitar que outros métodos que não pertençam à classe **Led** os alterem, especialmente o pino de E/S onde o LED está ligado. Como o atributo do estado do LED só pode

ser afetado por funções do objeto, para se avaliar se o LED está aceso ou apagado foi criado o método `getState()`, que simplesmente devolve o seu valor.

O construtor da classe recebe como parâmetro o número identificador do pino de E/S por uma variável com o mesmo nome do atributo `pin`, e trata de configurar o pino de E/S. Para se distinguir o parâmetro do atributo usa-se o *this pointer*. Em vez do ponto, para aceder aos atributos e métodos, usa-se `this->` atributo/método uma vez que o `this` é um ponteiro par ao objeto. Graças ao uso de classes o programador já não está preocupado em escrever instruções para atuar sobre os portos de E/S, mas sim em descrever o comportamento do objeto, neste exemplo acender e apagar um LED.

Listagem 2.4: Sketch de demonstração da utilização de classes para definição do componente LED.

```
class Led {
private:
    byte pin;
    bool state;
public:
    Led(byte pin = 13) {
        this->pin = pin;
        pinMode(pin, OUTPUT);
    }
    void setON() {
        digitalWrite(pin, HIGH);
        state = HIGH;
    }
    void setOFF() {
        digitalWrite(pin, LOW);
        state = LOW;
    }
    bool getState() {
        return this->state;
    }
};

Led onboardLed = Led(13);
Led laserLed = Led(11);

void setup ()
{
    onboardLed.setOn();
    laserLed.setOff();
}

void loop () {
    laserLed.setOff();
    delay(1000);
    laserLed.setOn();
    delay(1000);
}
```

2.3.3 Funções Especiais Arduíno

Existem funções adicionais na instalação do ambiente de desenvolvimento Arduíno que não existem no C++ standard, que servem para facilitar a interação com os periféricos e realizar cálculos realizados frequentemente no contexto de sistemas embebidos. Os protótipos das funções mais relevantes

encontram-se no ficheiro *header* `arduino.h` e encontra-se sintetizado na tabela 2.10. Existem limitações no uso de algumas funções habitualmente usadas em C/C++ porque ocupam demasiado espaço em memória ou demoram bastante tempo a processar, nomeadamente funções matemáticas e suporte para escrita de caracteres para números reais. Em `file:///C:/Program%20Files%20(x86)/Arduino/reference/www.arduino.cc/en/Reference/HomePage.html` existe documentação de referência que providencia todos os detalhes sobre cada função.

O Arduino implementa a biblioteca matemática `Avr-Libc` via `math.h`, no entanto para muitas aplicações, incluir esta biblioteca introduz uma grande penalização a nível de recursos consumidos. Por este motivo existem algumas funções matemáticas simplificadas fora da biblioteca `math.h`, sumarizadas na tabela 2.11.

2.3.4 Programar para o Mundo Exterior

Uma das diferenças entre programar um computador e um sistema embebido consiste na utilização de periféricos e interfaces. Enquanto um computador é usado para transferir, processar e mostrar informação em periféricos ubíquos tais como teclado, rato, rede ethernet, e monitor, um sistema embebido é favorecido pela flexibilidade e diversidade de componentes que nele podem ser ligados, como por exemplo receber informação de um sensor de movimento ou dar uma indicação com um LED ou um mostrador gráfico. Contudo, a nível do programa, pode ser necessário usar métodos de suporte às operações de leitura e escrita do periférico, através de bibliotecas com métodos específicos, *Board Support Packages* ou *Device Drivers* para sistemas complexos.

Nas plataformas Arduino os portos analógicos são distintos dos portos digitais, e os portos digitais podem ser usados tanto para leitura (entrada) ou escrita (saída), dependendo da sua função configuração. A configuração da direção dos pinos digitais é feita com a função `pinMode(numPino, modo)`. O modo pode ter o valor `INPUT` (entrada), `INPUT_PULLUP` (entrada polarizada a 1) e `OUTPUT` (saída). Nas plataformas Arduino, habitualmente existe um LED ligado ao pino digital 13, conhecido no código pela constante `LED_BUILTIN`. O positivo do LED encontra-se ligado ao pino do Arduino, e o seu negativo ligado a GND por meio de uma resistência (220Ω)⁴. Os portos de E/S são explicados em detalhe no capítulo ??.

A interação com os componentes ligados aos pinos de E/S de um Arduino é feito com recurso às rotinas:

- `analogRead(numPino)` - Leitura de um valor de tensão analógico (0-

⁴É indiferente o pino do LED onde a resistência fica ligada. Pode ser ligada entre o positivo do LED e VCC, com o negativo do LED ligado a GND.

pino de entrada digital (também pode ser um sinal de um periférico interno). Depois da rotina terminada, a execução do programa é retomada a partir do ponto onde ficou suspenso. Exemplos de utilização de interrupções são: contagem precisa do tempo, transferência de dados num porto de comunicação ou processamento de um conjunto de dados produzidos por um sensor. Um sistema embebido pode ter várias entradas para sinais de interrupção e cada entrada associada uma rotina de atendimento de interrupção. De uma forma simplista, a interrupção pode ser vista como a execução de uma rotina para tratar um alarme que não pode esperar pelo programa para avaliar a situação, mas que não demora muito tempo a tomar conta da ocorrência.

No Arduíno as interrupções são programadas usando a função `attachInterrupt(numIntr, ISR, ativação)`, onde `numIntr` corresponde ao número da interrupção a usar para o pino de entrada desejado (atenção que não é o número do pino), `ISR` corresponde ao nome da função a executar.

O Sketch na listagem 2.5 demonstra uma utilização básica do mecanismo de interrupções com um Arduíno Uno. Neste exemplo, o pino 2 do Arduíno Uno é configurado como entrada com um *pull-up*, ficando no estado 1 quando em repouso. O `loop()` copia constantemente o valor da variável de estado (`state`) para o LED na saída 13. Quando pino 2 é ligado a GND (estado 0), a interrupção fica ativa quando o sinal passa de 1 para 0 (FALLING), e a rotina `isr()` é executada. No seu corpo, a rotina atualiza o valor da variável estado para o seu complemento (negação).

Listagem 2.5: interruptDemo.ino

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), isr, FALLING);
}

void loop() {
  digitalWrite(ledPin, state);
}

// Interrupt Service Routine
void isr() {
  state = !state;
}
```

No quarto capítulo são dados mais detalhes relacionados com a implementação do mecanismo de interrupções em Arduíno.

2.3.6 Bibliotecas Arduíno

As bibliotecas são bastante úteis quando existem ficheiros com código que é usado constantemente, como por exemplo funções matemática e rotinas de baixo nível para interação com periféricos como mostradores. Para evitar ter

de replicar este código, e propagar futuras atualizações aos diferentes projetos, as bibliotecas permitem de forma prática reutilizar o código, e reduzir o número de linhas de código no projeto.

O Arduino inclui algumas bibliotecas pré-instaladas cuja funcionalidade está resumida na tabela 2.13.

Bibliotecas Pré-Instaladas

A plataforma Arduino inclui algumas bibliotecas que não fazem parte das do C/C++. Algumas destas bibliotecas servem para interação com periféricos ou para realização de algoritmos complexos, por exemplo: mostradores gráficos, portos SPI e I2C, controlador Ethernet, SD Card e sistema de ficheiros FAT.

Par poder usar a funcionalidade de uma biblioteca pré-instalada basta incluir o ficheiro *header* no Sketch, exemplo: `#include <Ethernet.h>`.

Criação

Para criar uma biblioteca na plataforma Arduino é necessário ter dois ficheiros: `.h` (cabeçalhos - *header*) e `.cpp` (corpo das funções - *source*). O ficheiro `.h` contém todas as definições internas a usar pela biblioteca, ao passo que o ficheiro `.cpp` contém a implementação. Não esquecer de usar as macros `#ifndef MY_LIB`, `#define MY_LIB` e `#endif` para evitar incluir uma biblioteca duas vezes no mesmo projecto.

Instalação

A instalação das bibliotecas Arduino pode ser automática ou manual. Recomenda-se a instalação automática: Após descarregar o ficheiro `.ZIP` com a biblioteca, ir a menu **Sketch/Rascunho** → **include library** → **add .ZIP library**, e indicar o caminho onde a biblioteca ficou guardada e abrir esse ficheiro.

A instalação manual é um método alternativo para quando ficheiro `.ZIP` não existe, e passa por copiar os ficheiros `.h` e `.cpp` para uma nova sub-pasta a criar dentro da pasta **libraries** na instalação do IDE Arduino. Na mesma pasta um ficheiro de texto chamado **keywords.txt** que contém as palavras-chave a realçar pelo IDE Arduino. Neste ficheiro de texto cada palavra a ser realçada deve estar no início de cada linha seguida de um **TAB** e indicação de qual a cor a usar: **KEYWORD1** para laranja e **KEYWORD2** para castanho. Exemplo de um ficheiro de palavras-chave.

Palavra1	KEYWORD1
Palavra2	KEYWORD2
Palavra3	KEYWORD2

Utilização

A utilização das bibliotecas passa pela inclusão dos ficheiros *header* (.h) copiados para a pasta **libraries**. A listagem 2.6 ilustra a inclusão dos ficheiros .h da biblioteca gráfica. A partir deste momento podem-se invocar os métodos e declarar variáveis com os tipos definidos nesta biblioteca.

Listagem 2.6: Exemplo de utilização das biblioteca gráfica Adafruit.

```
##include <Adafruit_GFX.h>
##include <Adafruit_NeoMatrix.h>
##include <Adafruit_NeoPixel.h>

(...)

Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(5, 8, PIN,
  NEO_MATRIX_TOP + NEO_MATRIX_RIGHT +
  NEO_MATRIX_COLUMNS + NEO_MATRIX_PROGRESSIVE,
  NEO_GRB + NEO_KHZ800);

(...)
```

2.3.7 Avaliação de Desempenho e Recursos Usados

A forma para aferir o desempenho, ou a capacidade, de um sistema computacional passa por medir o tempo e energia necessários para a execução de um programa conhecido (*benchmark*). Um programa é caracterizado pelo espaço em memória necessário para o guardar e pelo tempo necessário a executar num sistema computacional. Basicamente, quanto mais rápido for o processador menor será o tempo de processamento. Nos processadores existem estruturas internas, tais como a cache, que permitem acelerar o processamento sem aumentar a frequência do relógio. Nesta secção é mostrado como aferir o desempenho na plataforma Arduino.

Desempenho

No Arduino existem métodos que permitem medir o tempo, nomeadamente `millis()` e `micros()`⁷. Ambas as funções são chamadas sem parâmetros e devolvem um valor do tipo `unsigned long`, correspondente ao tempo decorrido desde o arranque do programa no Arduino, mas com diferentes granularidades. Com estas duas funções pode-se determinar quanto tempo dura a execução de um programa, parte dele, ou chamada de função.

O programa `time_count`, na listagem 2.7, mede o tempo total de execução do programa, com excepção da última instrução que é usada para mostrar o tempo decorrido. Neste programa são medidos os tempos de execução da função `delay(100)`, que suspende a execução durante 100 ms, e de 1000 execuções da função `sin(.)`. A listagem do Sketch 2.7 mostra o código fonte usado.

⁷1 segundo = 1.000 milissegundos (10^3) = 1.000.000 microsegundos (10^6) = 1.000.000.000 nanosegundos (10^9).

Listagem 2.7: timeCount.ino

```
unsigned long t0, t1, t2, t3;

void setup() {
  Serial.begin(9600);
  while (!Serial) ; // wait for serial port to connect.
}

void loop() {
  int i;
  volatile float x = 0.0;

  t0 = millis();

  t1 = micros();
  delay(100);
  t2 = micros();
  t3 = t2 - t1;

  Serial.print("t0: (ms) "); Serial.println(t0);
  Serial.print("t1: (us) "); Serial.println(t1);
  Serial.print("t2: (us) "); Serial.println(t2);
  Serial.print("Elapsed time #1: (us) ");
  Serial.println(t3);

  t1 = micros();
  for(i = 0; i < 1000; i++) {
    x = sin((float)i);
  }
  t2 = micros();
  t3 = t2 - t1;

  Serial.print("Elapsed time #2: (us) ");
  Serial.println(t3);

  Serial.print("Program time: (ms) ");
  Serial.println(millis() - t0);

  while(1) delay(100);
}
```

Neste *Sketch* As variáveis `t0`, `t1`, `t2` e `t3` são do tipo `unsigned long`, e são usadas para guardar os valores do tempo decorrido aquando da execução de diferentes etapas:

- `t0` - tempo de após o arranque do programa (`setup()`);
- `t1` - tempo anterior à chamada da função a medir;
- `t2` - tempo após a execução da função a medir;
- `t3` - diferença de tempo correspondente à duração da execução da função ($t3 = t2 - t1$).

O resultado lido no *Serial Monitor* após a execução num Arduino Uno é o seguinte:

```
t0: (us) 0
t1: (us) 8
t2: (us) 100020
```

```
Elapsed time #1: (us) 100008
Elapsed time #2: (us) 123200
Program time: (ms) 229
```

É relevante observar nestes resultados que o tempo medido para o atraso é superior aos 100ms, devido ao tempo necessário para as chamadas de funções, e que o tempo de execuções de 1000 execuções da função `sin(.)` demoram 140096 μ s. O tempo da execução de apenas uma execução da função `sin(.)` é 4 μ s.

A listagem 2.8 mostra o código-fonte do Sketch para medir o tempo de execução das instruções de E/S. Este programa executa 10.000 leituras/escritas analógicas e digitais e mede o tempo decorrido em microssegundos. A repetição da escrita no pino digital serve para ilustrar que existe variabilidade nos tempos de escrita nos portos. Se o programa for executado várias vezes, é natural serem medidos valores diferentes. Para que exista significância estatística, um programa deve ser executado várias vezes para se obter um valor mais próximo do valor realidade.

Listagem 2.8: Sketch para medição dos tempos das operações de E/S.

```
#define N 10000

void setup() {
  pinMode(LED.BUILTIN, OUTPUT);
  pinMode(12, INPUT_PULLUP);
  Serial.begin(9600);
  while(!Serial);
  Serial.println("*** Basic I/O ***");
}

void loop() {
  uint32_t t1, t2, dt;
  int i;

  t1 = micros();
  for(i = 0; i < N; i++) {
    digitalWrite(LED.BUILTIN, HIGH);
  }
  t2 = micros();
  dt = t2 - t1;
  Serial.print("Elapsed time digitalWrite H\t: "); Serial.println(dt);

  // repetir ciclo anterior

  t1 = micros();
  for(i = 0; i < N; i++) {
    digitalWrite(LED.BUILTIN, LOW);
  }
  t2 = micros();
  dt = t2 - t1;
  Serial.print("Elapsed time digitalWrite L\t: "); Serial.println(dt);

  // repetir ciclo anterior

  t1 = micros();
  for(i = 0; i < N; i++) {
    digitalRead(12);
  }
  t2 = micros();
  dt = t2 - t1;
  Serial.print("Elapsed time digitalRead\t: "); Serial.println(dt);
```

CAPÍTULO 2. CONCEITOS BÁSICOS

```
t1 = micros();
for(i = 0; i < N; i++) {
    analogWrite(0, 0);
}
t2 = micros();
dt = t2 - t1;
Serial.print("Elapsed time analogWrite 0\t: "); Serial.println(dt);

t1 = micros();
for(i = 0; i < N; i++) {
    analogWrite(0, 255);
}
t2 = micros();
dt = t2 - t1;
Serial.print("Elapsed time analogWrite 255\t: "); Serial.println(dt);

t1 = micros();
for(i = 0; i < N; i++) {
    analogRead(0);
}
t2 = micros();
dt = t2 - t1;
Serial.print("Elapsed time analogRead \t: "); Serial.println(dt);

Serial.println(" ");
}
```

A figura 2.37 mostra o resultado da execução da medida do tempo para 10.000 leituras/escritas nos portos de um Arduino Uno.

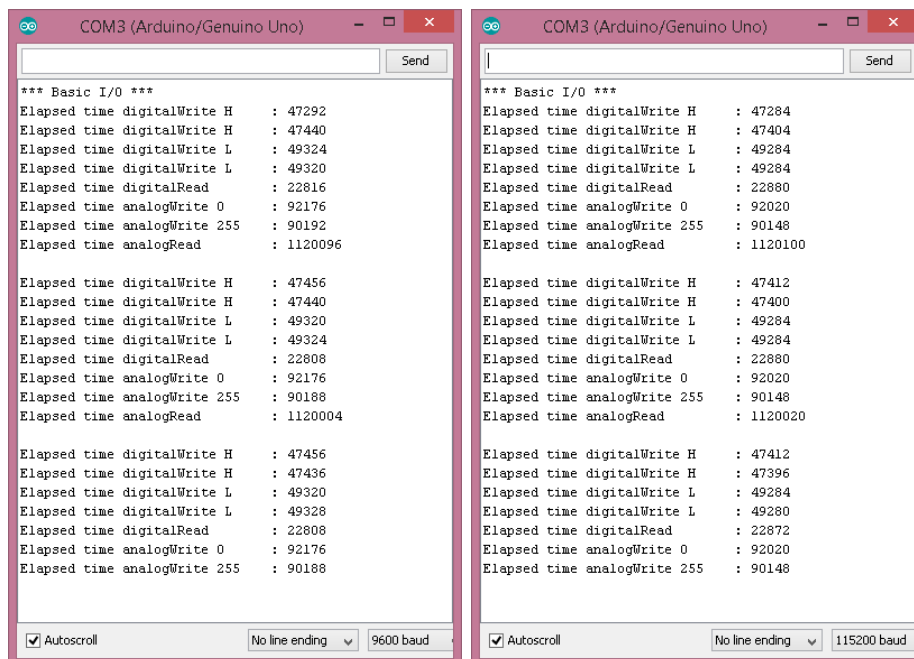


Figura 2.37: Tempo despendido com as instruções de E/S.

Dos resultados obtidos, observa-se que não só os diferentes tipos de leituras e escritas, quer analógicas como digitais, como também o seu valor, e o valor anterior, influenciam o tempo de execução. Esta observação é esperada

uma vez que o CPU tem de interagir com diversos periféricos que têm diferentes tempos de resposta. As rotinas para operações de E/S oferecidas pelo Arduino, se por um lado facilitam, por outro introduzem instruções extra no programa, e que demoram tempo a ser processadas. É importante reter que qualquer instrução executada tem um determinado tempo de execução, e não é nulo! Para aplicações onde o tempo das operações é crítico pode-se optar pela programação direta do porto ou recorrer a programação em linguagem *assembly*.

Recursos

Os recursos que um programa necessita são: memória e tempo de processamento. A nível da programação de um sistema embebido a memória é o recurso mais crítico, pois não só limita a quantidade de instruções que o programa pode executar, como também os dados a processar. Após a compilação de um *Sketch* sem erros o IDE Arduino reporta na janela de mensagens os recursos que um programa ocupa. Se um Sketch necessitar de mais memória do que a disponível devem-se adoptar estratégias para reduzir o consumo de memória, nomeadamente:

- reutilizar variáveis ou converter várias em **unions** - é necessário acompanhar o contexto da variável para manter a coerência do programa;
- passar para constantes as variáveis declaradas com valores inalterados, e guardá-las em memória de programa;
- passar ponteiros de estruturas de dados em vez de copiar valores;
- declarar variáveis globais para passar valores entre funções (*blackboard*);
- codificar várias variáveis em diferentes conjuntos de bits de uma só variável - pode implicar a execução de mais instruções.

Deve-se evitar usar mais do que 90% dos recursos, sob pena da aplicação falhar inesperadamente. Para além dos recursos constantes como as instruções do programa (.TEXT), em ROM/FLASH, e as variáveis fixas declaradas (.DATA para inicializadas e .BSS para não-inicializadas e do tipo **static**), em RAM, existem outras estruturas em memória que consomem recursos que variam durante a execução do programa, nomeadamente:

- *Stack* - Zona reservada para registar valores de argumentos de entrada e saída de funções, variáveis declaradas dentro de funções, e endereços de retorno após a conclusão de funções. Na memória o *Stack* ocupa os endereços mais altos da memória RAM livre. O endereço tende a decrescer com a ocupação de novos dados, e a crescer quando são removidos da pilha;

- *Heap* - Zona reservada para variáveis alocadas dinamicamente, e outros dados criados durante a execução do programa (`malloc()` e `new()`). O *Heap* ocupa os endereços mais baixos da memória RAM livre e o seu endereço tende a crescer com a sua ocupação.

No mapa de memória, o *Stack* e o *Heap* ocupam o resto da memória RAM que não foi ocupada pelas variáveis fixas, mas cada uma destas zonas começa no extremo oposto, tal como ilustrado na figura 2.38. Se uma delas, ou ambas, crescerem demasiado e se sobrepõem, os dados em cada uma destas zonas serão destruídos, levando o programa a ter um comportamento errado. O compilador pode dar um aviso que tem pouco espaço de memória para alocação dinâmica, mas não consegue prever, pois só aquando da execução do programa se consegue saber o uso real.

A listagem 2.11 corresponde a um Sketch que demonstra a corrupção do *Stack* através da chamada de uma função recursiva para calcular um valor da série de Fibonacci (0, 1, 1, 2, 3, 5, 8, 13...). Esta função calcula o valor pretendido à custa de dois valores anteriores da função até chegar a um dos dois valores iniciais (0, 1). Por exemplo, para o cálculo do quarto elemento da série de Fibonacci, é necessário calcular os valores do terceiro e do segundo elementos, que por sua vez vão ser calculados com os valores anteriores, até que chegue a 0 ou 1. Como não é possível guardar os valores intermédios em variáveis da função chamada recursivamente, o *Stack* é usado para os guardar. O *Stack* guarda não só os valores intermédios, como também as posições do programa aquando da chamada das funções. Isto faz com que quando a função `fib()` é invocada com um valor elevado como o 999, o *Stack* vai crescer até se esgotar a memória. Quando isto acontece o programa normalmente bloqueia ou tem uma manifestação errada.

Embora o resultado da compilação, apresentado na listagem 2.10, mostrasse pouca ocupação de memória de programa e dados num Arduino Uno (11%), o problema com o *Stack* não podia ser antecipado. Se a chamada da função `fib(999)` for comentada, o programa corre sem problemas até ao fim. Quanto maior a memória dinâmica disponível, mais pode crescer o *Stack*.

Listagem 2.9: Demonstração da corrupção do *Stack* num Arduino.

```
long f;
char buf[8];
void* h = NULL; // heap

extern unsigned char __data_start;
extern unsigned char __data_end;
extern unsigned char __bss_start;
extern unsigned char __bss_end;
extern unsigned char __heap_start;
extern unsigned char *__brkval;

int FreeRam() {
    unsigned char top;
    return __brkval ? &top - __brkval : &top - &__bss_end;
}
```

```

unsigned int freeBytes;

// recursive function to implement Fibonacci sequence
// fib(n) = fib(n-1) + fib(n-2)
// fib(0) = 0, fib(1) = 1
long fib(long n) {
    long i, j, k;

    // sprintf(buf, "SP: %p", SP);
    // similar effect as above as function variables go to the stack
    void* sp = NULL;
    freeBytes = FreeRam();
    sprintf(buf, "%p", &sp);
    Serial.print(buf);
    sprintf(buf, " %d", &h);
    Serial.print(buf);
    sprintf(buf, " %d ", (unsigned int)&sp - (unsigned int)&h);
    Serial.print(buf);
    Serial.println(freeBytes);

    if(n <= 1) {
        return n;
    } else {
        j = fib(n - 1);
        k = fib(n - 2);
        i = j + k;
        return i;
    }
}

void setup() {
    /* for this example need high baud rate, otherwise may
    not have the time to receive all characters before stack kaboom */
    Serial.begin(115200); while(!Serial);
    Serial.println(FreeRam());
    Serial.println("Stack Heap Diff Free");
    h = malloc(1); // dynamic alloc 1byte to get heap position
    f = fib(3);
    Serial.println(f, DEC);
    f = fib(4);
    Serial.println(f, DEC);
    f = fib(999); // <- stack KABOOM :(
    Serial.println(f, DEC);
    Serial.println("***end***");
}

void loop() {
    // no need to repeat
}

```

Listagem 2.10: Resultado da compilação do Sketch para corrupção do Stack

```

Sketch uses 4256 bytes (13%) of program storage space. Maximum is 32256 bytes.
Global variables use 238 bytes (11%) of dynamic memory, leaving 1810 bytes for local variables.
Maximum is 2048 bytes.

```

A listagem 2.11 mostra o resultado da execução do Sketch no terminal. Cada vez que a função `fib()` é chamada, é apresentado o valor do *Stack Pointer*, e a diferença em relação ao valor do endereço do *Heap*. O valor do endereço do *Stack Pointer* é obtido à custa da declaração de uma variável dentro da função. Variáveis que apenas existem durante a execução da função são colocadas no *Stack*, ex. `i`, `j` e `k`. O valor do último endereço do *Heap* é obtido através da alocação dinâmica de um byte para uma variável antes da chamada da função `fib()`.

Durante a execução do Sketch é possível observar a diferença entre os dois valores a decrescer à medida que a função `fib()` se chama a ela própria

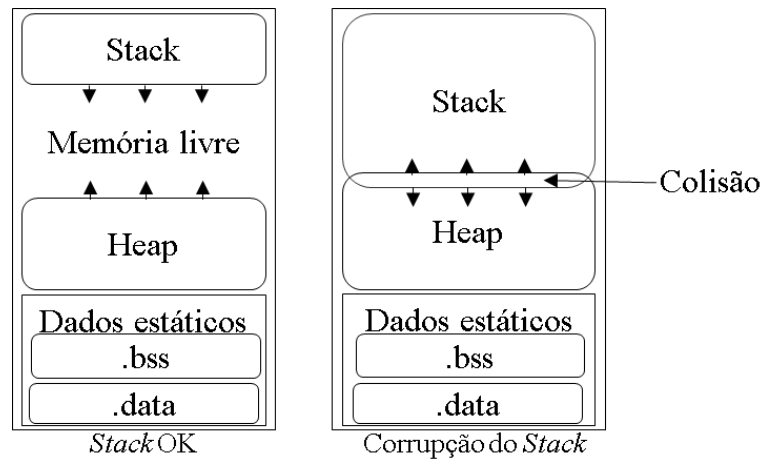


Figura 2.38: Evolução da ocupação das zonas de *Heap* e *Stack*, até ser esgotada a memória dinâmica disponível.

para calcular os valores anteriores da série de Fibonacci. A última execução apresentada é quando o *Stack* está em 0x25E e a diferença para o endereço do *Heap* é 0x8E (142) bytes. O não ter diferença próxima ou igual a zero deve-se ao facto das variáveis e funções executadas dentro da função `fib()` consumirem várias posições do *Stack*, nomeadamente o `sprintf()` e o `println()` que colocam os caracteres no *Stack*, o que explica os caracteres fora do formato hexadecimal no final. Esta experiência realizada num Arduino Uno. Se for usado MCU diferente serão produzidos resultados diferentes.

Listagem 2.11: Resultado da execução do Sketch para corrupção do *Stack*

```
Stack Heap Diff
0x8ee 0x142 0x3d6
0x8de 0x142 0x3ce
0x8ce 0x142 0x3c6
0x8ce 0x142 0x3c6
0x8de 0x142 0x3ce
2
(...)
0x8ee 0x142 0x3d6
0x8de 0x142 0x3ce
(...)
0x26e 0x142 0x96
0x25e 0x142 0x8e
A?BXG424IAB?
```

Variáveis que tenham sempre o mesmo valor durante toda a execução do programa são conhecidas como constantes e podem ficar guardadas em memória de programa não-volátil (ROM) em alternativa à memória volátil (RAM). As variáveis em ROM devem ser globais ou do tipo `static`. Como os acessos são feitos à memória com instruções `pgm_read_`, onde é necessário indicar qual o tamanho das palavras a ler (`byte`, `word`, `dword`, `float`, `pointer`) e o tipo de endereço (`near` = 16 bits ou `far` = 32 bits). Exemplo de funções disponíveis: `pgm_read_byte_near(...)` e `pgm_read_word(...)`.

A listagem 2.12 demonstra a utilização da memória não volátil para guar-

dar constantes da tabela de senos 0-90°. Dadas as relações trigonométricas é possível obter o seno e cosseno de ângulos (θ) de qualquer quadrante à custa dos senos entre 0° e 90°, $\sin(\Theta) = \cos(90 - \Theta)$, tal como sumarizado na tabela 2.14 e ilustrado na figura 2.39.

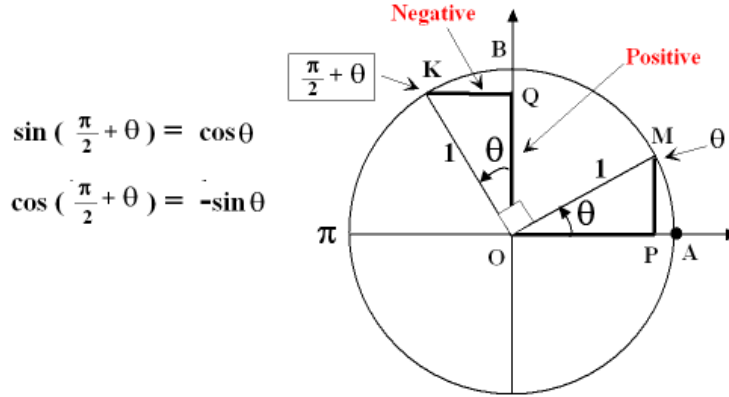


Figura 2.39: Relação entre as várias identidades trigonométricas.

Listagem 2.12: Demonstração da utilização de constantes em memória não volátil.

```
// demo constant variables @ROM/FLASH
// uncomment statements and set the corresponding variable definitions #if 0/1
#if 0
// octave/matlab: sprintf('%d, ', floor(256 * sin([0:90] * 2*pi/360)))
const PROGMEM byte sin_byte[] = {
0, 4, 8, 13, 17, 22, 26, 31, 35, 40, 44, 48, 53, 57, 61, 66, 70, 74, 79, 83,
87, 91, 95, 100, 104, 108, 112, 116, 120, 124, 127, 131, 135, 139, 143, 146,
150, 154, 157, 161, 164, 167, 171, 174, 177, 181, 184, 187, 190, 193, 196, 198,
201, 204, 207, 209, 212, 214, 217, 219, 221, 223, 226, 228, 230, 232, 233, 235,
237, 238, 240, 242, 243, 244, 246, 247, 248, 249, 250, 251, 252, 252, 253, 254,
254, 255, 255, 255, 255, 255, 256 };
#endif
#if 1
// sprintf('%f, ', sin([0:90] * 2*pi/360))
const PROGMEM float sin_float[] = {
0.000000, 0.017452, 0.034899, 0.052336, 0.069756, 0.087156, 0.104528, 0.121869,
0.139173, 0.156434, 0.173648, 0.190809, 0.207912, 0.224951, 0.241922, 0.258819,
0.275637, 0.292372, 0.309017, 0.325568, 0.342020, 0.358368, 0.374607, 0.390731,
0.406737, 0.422618, 0.438371, 0.453990, 0.469472, 0.484810, 0.500000, 0.515038,
0.529919, 0.544639, 0.559193, 0.573576, 0.587785, 0.601815, 0.615661, 0.629320,
0.642788, 0.656059, 0.669131, 0.681998, 0.694658, 0.707107, 0.719340, 0.731354,
0.743145, 0.754710, 0.766044, 0.777146, 0.788011, 0.798636, 0.809017, 0.819152,
0.829038, 0.838671, 0.848048, 0.857167, 0.866025, 0.874620, 0.882948, 0.891007,
0.898794, 0.906308, 0.913545, 0.920505, 0.927184, 0.933580, 0.939693, 0.945519,
0.951057, 0.956305, 0.961262, 0.965926, 0.970296, 0.974370, 0.978148, 0.981627,
0.984808, 0.987688, 0.990268, 0.992546, 0.994522, 0.996195, 0.997564, 0.998630,
0.999391, 0.999848, 1.000000 };
#endif

int angle;
volatile float sinVal;
unsigned long t0;

void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println(F("\n\r*** Const Vars ROM/FLASH Demo ***"));
  t0 = micros();
}
```


CAPÍTULO 2. CONCEITOS BÁSICOS

```
    for(angle = 0; angle < 90; angle++) {  
      // sinVal = pgm_read_byte_near(sin_byte + angle) / 256.0; // store bytes  
      sinVal = pgm_read_float_near(sin_float + angle); // store floats  
      // sinVal = sin(angle); // compute sin(rad) - incorrect angle  
      // sinVal = sin(angle * 2.0 * M_PI / 360.0); // compute sin(deg)  
      // Serial.println(sinVal); // print value @serial plotter - don't do profile w/ this  
    }  
    Serial.println(micros() - t0);  
  }  
  
void loop() { delay(1000); }
```

A avaliação dos recursos necessários para cálculo dos primeiros 90^o recorrendo a implementações diferentes, num Arduino Uno, encontra-se na tabela 2.15. A primeira implementação apenas executa o ciclo da contagem do valor do ângulo entre 0 e 90. A segunda implementação guarda os valores do seno numa tabela com 1 byte por posição. Nesta implementação é necessário um cálculo adicional recuperar o valor em vírgula flutuante. Para o seno dos ângulos 0^o-90^o com representação a 8 bits, o erro máximo é de 0,0039, correspondente ao peso do bit menos significativo (2^{-8}). A terceira implementação guarda os valores do seno numa tabela em vírgula flutuante, não sendo necessária qualquer conversão. A quarta implementação corresponde à execução da função `sin()` sem conversão do valor do ângulo em graus para radianos, originando valores diferentes dos pretendidos. Esta implementação serve para aferir o impacto da conversão do ângulo de graus para radianos usada na implementação seguinte. A quinta implementação converte o valor do ângulo de graus para radianos e produz o mesmo resultados das anteriores.

Das cinco implementações a que demora menos tempo e gasta menos memória é a tabela com os valores em vírgula flutuante. A tendência é assumir que a tabela com inteiros de 1 byte seria a opção mais económica, no entanto a conversão de inteiro para real gasta bastantes recursos e demora mais tempo. As conversões penalizam também a última implementação de tal maneira que quase demora 70x mais e excede o programa em 1 kB do que o necessário para ter o mesmo resultado.

Com este ensaio conclui-se que se houver necessidade de calcular valores de funções que demoram tempo a serem calculadas, a recomendação é sempre que possível usar uma tabela em memória do que calcular a função. De salientar ainda que a biblioteca com funções matemáticas ocupam bastante espaço, pelo que se os valores forem calculados antecipadamente aproveita-se algum desse espaço para guardar a tabela.

A memória de programa pode ser usada para armazenar mensagens pré-definidas a ser mostradas, como por exemplo mensagens de boas vindas e opções de menu, normalmente com a instrução `printf()`. O Arduino facilita essa utilização através da macro `F()` a envolver a sequência de caracteres a ser apresentada.

Tipo	Bytes em memória	Descrição
boolean	1	Valor lógico: True/False
char	1	Valor inteiro entre -128 e 127
unsigned char	1	Valor inteiro positivo entre 0 e 255
byte	1	Idem
int	2	Valor inteiro entre - 32.768 e 32.767
unsigned int	2	Valor inteiro positivo entre 0 e 65535
word	2	Idem
long	4	Valor inteiro entre 2.147.483.648 e 2.147.483.647
unsigned long	4	Valor inteiro positivo entre 0 e 4.294.967.295
float	4	Valor real entre $-3,4028235 \times 10^{38}$ e $3,4028235 \times 10^{38}$
double	4	
string (N elem.)	1 + N	caracteres da string mais o terminador \0
array (N elem.)	N * sizeof(tipo)	vetor com N elementos de um tipo
enum	*	Sequência de nomes aos quais são atribuídos índices de valor inteiro
struct	*	Composição de várias variáveis de qualquer tipo para criar uma nova mais complexa. Exemplo: DateTime ano, mês, dia, hora, minutos, segundos
pointer	2	Ponteiro corresponde ao endereço em memória onde o conteúdo da variável começa. Exemplo: int *p = "xpto"

Tabela 2.8: Tipos básicos de dados.

Tipo	Num Bytes
int8_t, int_fast8_t, int_least8_t, uint8_t	1
uint16_t	2
uint32_t	4
uint64_t, intmax_t	8
char	1
short, int	2
long	4
long long	8
float, double	4
uint8_t*, uint16_t*, uint32_t*, uint64_t*	2
bool, byte	1
word	2

Tabela 2.9: Tipos de variáveis e o espaço que ocupam em memória

2.3. PROGRAMAÇÃO

Tipo	Nome	Descrição
Tempo	<code>delay(unsigned long x)</code> <code>delayMicroseconds(unsigned int x)</code> <code>unsigned long micros()</code> <code>unsigned long millis()</code>	suspende a execução durante x milissegundos suspende a execução durante x microssegundos ($x < 16383$) número de microssegundos decorridos desde que o programa iniciou número de milissegundos decorridos desde que o programa iniciou
Caracteres	<code>bool isAlpha(char c)</code> <code>bool isAlphaNumeric(char c)</code> <code>bool isAscii(char c)</code> <code>bool isControl(char c)</code> <code>bool isDigit(char c)</code> <code>bool isGraph(char c)</code> <code>bool isHexadecimalDigit(char c)</code> <code>bool isUpperCase(char c)</code> <code>bool isLowerCase(char c)</code> <code>bool isPrintable(char c)</code> <code>bool isPunct(char c)</code> <code>bool isWhitespace(char c)</code> <code>bool isSpace(char c)</code>	avalia se o carater c é uma letra [a-z,A-Z] avalia se o carater c é uma letra ou dígito [0-9,a-z,A-Z] avalia se o carater c é um carater ASCII válido avalia se o carater c é um carater ASCII de controlo avalia se o carater c é um dígito [0-9] avalia se o carater c é um carater visível avalia se o carater c é um dígito hexadecimal [0-9,A-F] avalia se o carater c é um carater maiúsculo avalia se o carater c é um carater minúsculo avalia se o carater c é um carater de texto (oposto a controlo) avalia se o carater c é um carater de pontuação [.,;:] avalia se o carater c é um espaço [ex. TAB,ENTER] avalia se o carater c é um carater espaço
Números aleatórios	<code>long random(long a = 0, long b)</code> <code>randomSeed(long)</code>	gera um valor pseudo-aleatório entre a e b ou 0 e b define o valor da semente do gerador pseudo-aleatório
Bits e Bytes	<code>long bit(int b)</code> <code>bitClear(v, b)</code> <code>bitRead(v, b)</code> <code>bitSet(v, b)</code> <code>bitWrite(v, b, bv)</code> <code>highByte(w)</code> <code>lowByte(w)</code>	devolve o valor decimal corrente ao peso do bit b Coloca a zero o valor do bit b no valor v Devolve o valor do bit b, num valor v coloca a um o bit b do valor v recebe um valor v, e atualiza o seu bit b para o valor bv, e devolve o novo valor devolve o byte mais significativo de uma palavra de 16 bits devolve o byte menos significativo de uma palavra de 16 bits
Interrupções	<code>attachInterrupt(uint8_t int, void (*)(void), int mode)</code> <code>detachInterrupt(uint8_t int)</code> <code>interrupts()</code> <code>noInterrupts()</code>	Associa um método (void (*)) à interrupção int, sensível ao comportamento do sinal indicado por mode desassocia a interrupção int do método Ativa o atendimento das interrupções Desativa o atendimento das interrupções
Communication	<code>Serial</code> <code>Stream</code>	Objecto associado ao porto de comunicações série, ligado à UART que interage com o terminal série Classe de objetos para tratamento de sequências de caracteres

Tabela 2.10: Lista de funções disponibilizadas na biblioteca Arduino.

CAPÍTULO 2. CONCEITOS BÁSICOS

Função	Descrição
<code>abs(x)</code>	obtém o valor absoluto (positivo)
<code>constrain(amt,low,high)</code>	satura o valor <code>amt</code> nos valores <code>high</code> ou <code>low</code>
<code>map(x, fromL, fromH, toL, toH)</code>	mapeia um valor entre dois limites inferior e superior noutros limites
<code>max(a,b)</code>	devolve o máximo dos dois valores <code>a</code> e <code>b</code>
<code>min(a,b)</code>	devolve o menor dos dois valores <code>a</code> e <code>b</code>
<code>pow(x,a)</code>	calcula a potência x^a
<code>sq(x)</code>	calcula o quadrado de <code>x</code>
<code>sqrt(x)</code>	calcula a raiz quadrada de <code>x</code>
<code>cos(x)</code>	cosseno de <code>x</code> em radianos
<code>sin(x)</code>	seno de <code>x</code> em radianos
<code>tan(x)</code>	tangente de <code>x</code> em radianos

Tabela 2.11: Funções matemáticas implementadas na biblioteca Arduino.

Tipo	Nome	Descrição
Digital	<code>int digitalRead(uint8_t pin)</code>	Lê o valor da entrada digital <code>pin</code> , 0 ou 1
	<code>void digitalWrite(uint8_t pin, uint8_t val)</code>	Escreve o valor <code>val</code> no pino de E/S <code>pin</code>
	<code>void pinMode(uint8_t pin, uint8_t mode)</code>	Define o modo de funcionamento do pino <code>pin</code> de entrada com ou sem pull-up, ou saída <code>mode</code>
Analogico	<code>int analogRead(uint8_t pin)</code>	Lê o valor de a entrada analógica <code>pin</code>
	<code>analogReference(uint8_t mode)</code>	Configura a tensão de referência (máxima) do conversor analógico-digital do AVR. <code>mode</code> = DEFAULT: 5 V, = INTERNAL1V1: 1,1 V (Mega), = INTERNAL2V56: 2,56 V (Mega), = INTERNAL: 1,1 V (ATmega168, ATmega328P) ou 2,56 V (ATmega32U4 e ATmega8), = EXTERNAL: tensão no pino AREF (0-5 V).
	<code>analogWrite(uint8_t pin, int v)</code>	Faz com que o pino <code>pin</code> produza um sinal cuja proporção do tempo entre estar a 0 e 1 seja $100*v/1024$
Avançado	<code>Tone(uint8_t pin, uint8_t frequency)</code>	Produz um sinal quadrado com a frequência especificada num pino <code>pin</code> de E/S. Para mudar de frequência é necessário usar <code>noTone()</code> primeiro
	<code>noTone(uint8_t pin)</code>	Desativa a geração do sinal quadrado no pino especificado
	<code>unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout = 1000000L)</code>	Determina a duração de um pulso, em microssegundos, até abandonar o estado inicial <code>state</code> , num pino de E/S.
	<code>uint8_t shiftIn(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder)</code>	Recebe um bit de <code>dataPin</code> , e desloca os recebidos anteriormente a cada transição do sinal no pino de E/S (<code>clockPin</code>) e devolve um byte. A ordem do primeiro bit recebido é indicada por <code>bitOrder</code> MSBFIRST ou LSBFIRST.
	<code>void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val)</code>	Serializa um byte (<code>val</code>) num pino de E/S (<code>dataPin</code>) síncrono com um sinal de relógio apresentado noutro pino de E/S (<code>clockPin</code>). O parâmetro <code>bitOrder</code> determina se é enviado primeiro o bit mais significativo (MSBFIRST) ou o bit menos significativo (LSBFIRST).

Tabela 2.12: Lista de funções de E/S disponibilizadas na biblioteca Arduino.

Nome	Descrição
EEPROM	Interface com memórias do tipo EEPROM (<i>Electrically Erasable Programmable Read Only Memory</i>)
Ethernet	Comunicação com um controlador de rede Ethernet
GSM	Suporte para modems GSM/GPRS estabelecerem chamadas de voz, envio e recepção de SMS, e acesso à Internet via GPRS
WiFi	Biblioteca para configuração do módulo WiFi existente nos Shields Arduino
LiquidCrystal	Interface com mostradores LCD alfanuméricos
TFT	Interface com mostradores gráficos do tipo TFT
SD	Suporte para interface SD-card e sistema de ficheiros FAT
Servo	Interface para servo-motor
Stepper	Interface para motor-de-passo
SPI	Funções para utilização do porto série síncrono (hardware)
SoftwareSerial	Biblioteca que emula um porto série assíncrono em software
Wire	Suporte para comunicação com o protocolo 1-wire

Tabela 2.13: Lista de bibliotecas existentes na instalação do IDE Arduino.

Quadrante	$\sin(\Theta)$	$\cos(\Theta)$
Q1 $\Theta \in [0 : 90]$	$\sin(\Theta)$	$1 - \sin(\Theta)$
Q2 $\Theta \in [90 : 180]$	$1 - \sin(\Theta - 90)$	$-\sin(\Theta - 90)$
Q3 $\Theta \in [180 : 270]$	$-\sin(\Theta - 180)$	$-\sin(\Theta - 180)$
Q4 $\Theta \in [270 : 360]$	$-1 + \sin(\Theta - 270)$	$1 - \sin(\Theta - 270)$

Tabela 2.14: Relação entre os ângulos das diferentes identidades trigonométricas em relação aos valores do seno no primeiro quadrante.

Implementação	Tempo [us]	ROM/FLASH	RAM
S/ calculo seno	4	1906	202
Tabela Byte	1244	2558	202
Tabela Float	204	2316	202
Seno (rad)	10868	2974	202
Seno (deg)	13996	3248	202

Tabela 2.15: Tempo e memória de programa necessários para o cálculo do seno dos ângulos entre 0° e 90° .

A

Anexos

A.1 Representações Numéricas

A.1.1 Binário

A representação binária e as suas operações, usadas nos circuitos lógicos, foram publicadas entre 1679-1710 por Gottfried Wilhelm Leibniz. Em que o “BInary digiT”, ou *bit*, é o elemento básico de memória. Cada bit pode conter um de dois valores, 0 ou 1, e daí designar-se por representação em base 2. Para representar uma variável que possa contar mais do que esses 2 valores é necessário adicionar mais bits. Desta forma com 2 bits é possível representar 4 combinações possíveis: 00, 01, 10, e 11. Por cada bit adicionado duplica-se o número de valores possíveis de representar. A cada valor binário corresponde um valor inteiro, em base decimal, sendo obtido à custa da soma do peso de cada bit: $dec = \sum_i b_i \cdot 2^i = b_N \cdot 2^N + \dots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$.

A.1.2 Hexadecimal

Hexadecimal é uma representação alternativa onde cada dígito codifica 16 valores possíveis. Esta representação contém os dígitos 0-9 e as letras A (10) a F (15). Esta é a representação binária mais compacta.

Uma palavra binária é constituída por vários bits. O conjunto de 4 e 8 bits designa por Nibble e Byte (B), respectivamente.

Na representação binária, um determinado número é escrito à custa de 0s e 1s, conforme o exemplo anterior. Para conjuntos tais como os usados em computadores, 32 e 64 bits é difícil identificar o valor correspondente se não

APÊNDICE A. ANEXOS

existir um método. A forma mais prática de obter um valor em base decimal é associar um peso a cada bit. Por exemplo um número com 4 bits, em base 10 é calculado com a seguinte formula: $Num_{10} = b_3.2^3 + b_2.2^2 + b_1.2^1 + b_0.2^0$. Esta formula pode ser generalizada para qualquer número de bits. Resolvendo o valor do peso de cada bit, o calculo pode ser simplificado para:

Número de Bit	7	6	5	4	3	2	1	0
Valor do bit	128	64	32	16	8	4	2	1
Binary Digit Value	0	1	1	0	0	1	0	1

Se forem somados os valores dos pesos cujos bits na representação binária estejam a 1, irá dar o mesmo valor em base 10. No exemplo dado o valor decimal correspondente é igual a: $256 + 64 + 32 + 4 + 1 = 357$.

Embora a representação binária permita obter a informação facilmente, a sua representação é aquela que requer mais dígitos, $\lceil \log_2(valor) \rceil$

Por ser a representação mais compacta, a notação hexadecimal é usada para representar endereços de memória. Por exemplo uma memória de 16kB tem os seus endereços compreendidos entre os valores 0x0000 e 0x3FFF. Se a memória for dividida ao meio em dois blocos, a primeira metade encontra-se entre os endereços 0x000 e 0x1FFF, e a segunda metade entre 0x2000 e 0x3FFF. Se for definida uma variável que ocupa 20 (20d = 0x14) bytes em memória, o próximo endereço disponível para guardar novas variáveis é o 0x2014.

A.2 Tabela ASCII

A tabela ASCII (nome derivado de American Standard Code for Information Interchange) é uma convenção adotada para possibilitar a comunicação ente sistemas informáticos e respetivos periféricos. Quando esta convenção foi inicialmente proposta nos anos 60, o poder computacional dos sistemas era bastante limitado e o mesmo canal de comunicação era usado para passar informação e controlar os terminais ou impressoras. Por este motivo existem dois conjuntos de caracteres: controlo e dados. A tabela seguinte mostra a codificação dos caracteres de controlo, sendo os mais relevantes atualmente a mudança de linha (LF), com o valor 10 e reposição da posição inicial (CR), com o valor 13.

Binário	OCT	DEC	HEX	Abreviatura	ESC	Descrição
00000000	000	0	00	NULL		
0	Nulo					
00001010	012	10	0A	LF		
n	Nova li- nha					
00001101	015	13	0D	CR		
r	Início de Li- nha					

A.3. ARMAZENAMENTO DE INFORMAÇÃO

A tabela seguinte mostra diferentes representações para os caracteres legíveis, a partir de 32 até 126.

Bin	Oct	Dec	Hex	Car	Bin	Oct	Dec	Hex	Car
00100000	040	032	20	~	00100001	041	033	21	!
00100010	042	034	22	”	00100011	043	035	23	#
00100100	044	036	24	\$	00100101	045	037	25	%
00100110	046	038	26	&	00100111	047	039	27	,
00101000	050	040	28	(	00101001	051	041	29	)
00101010	052	042	2a	*	00101011	053	043	2b	+
00101100	054	044	2c	,	00101101	055	045	2d	-
00101110	056	046	2e	.	00101111	057	047	2f	/
00110000	060	048	30	0	00110001	061	049	31	1
00110010	062	050	32	2	00110011	063	051	33	3
00110100	064	052	34	4	00110101	065	053	35	5
00110110	066	054	36	6	00110111	067	055	37	7
00111000	070	056	38	8	00111001	071	057	39	9
00111010	072	058	3a	:	00111011	073	059	3b	;
00111100	074	060	3c	i	00111101	075	061	3d	=
00111110	076	062	3e	¿	00111111	077	063	3f	?
01000000	100	064	40	@	01000001	101	065	41	A
01000010	102	066	42	B	01000011	103	067	43	C
01000100	104	068	44	D	01000101	105	069	45	E
01000110	106	070	46	F	01000111	107	071	47	G
01001000	110	072	48	H	01001001	111	073	49	I
01001010	112	074	4a	J	01001011	113	075	4b	K
01001100	114	076	4c	L	01001101	115	077	4d	M
01001110	116	078	4e	N	01001111	117	079	4f	O
01010000	120	080	50	P	01010001	121	081	51	Q
01010010	122	082	52	R	01010011	123	083	53	S
01010100	124	084	54	T	01010101	125	085	55	U
01010110	126	086	56	V	01010111	127	087	57	W
01011000	130	088	58	X	01011001	131	089	59	Y
01011010	132	090	5a	Z	01011011	133	091	5b	[
01011100	134	092	5c	“	01011101	135	093	5d	]
01011110	136	094	5e	^	01011111	137	095	5f	.
01100000	140	096	60	‘	01100001	141	097	61	a
01100010	142	098	62	b	01100011	143	099	63	c
01100100	144	100	64	d	01100101	145	101	65	e
01100110	146	102	66	f	01100111	147	103	67	g
01101000	150	104	68	h	01101001	151	105	69	i
01101010	152	106	6a	j	01101011	153	107	6b	k
01101100	154	108	6c	l	01101101	155	109	6d	m
01101110	156	110	6e	n	01101111	157	111	6f	o
01110000	160	112	70	p	01110001	161	113	71	q
01110010	162	114	72	r	01110011	163	115	73	s
01110100	164	116	74	t	01110101	165	117	75	u
01110110	166	118	76	v	01110111	167	119	77	w
01111000	170	120	78	x	01111001	171	121	79	y
01111010	172	122	7a	z	01111011	173	123	7b	—
01111100	174	124	7c	—	01111101	175	125	7d	”
01111110	176	126	7e	~	01111111	177	127	7f	

A.3 Armazenamento de Informação

Para uniformizar o armazenamento de informação passou-se a usar o Byte. N armazenamento de vários Bytes em memória usam-se abreviaturas para designar grandes quantidades:

Número de Bytes	Designação
1,024 (2^{10})	kilobyte (kB)
1,048,576 (2^{20})	Megabyte (MB)
1,073,741,824 (2^{30})	Gigabyte (GB)
1,099,511,627,776 (2^{40})	Terabyte (TB)

A.4 Códigos de Terminal ANSI

Os códigos de terminal ANSI, também conhecidos como “sequências de escape” (*escape sequences*) servem para controlar o terminal gráfico em termos de coordenadas do terminal a escrever, alterar cores do texto e do fundo, tipo de letra, limpar partes do ecrã e alterar o cursor. Existem várias sequências de códigos, sendo a mais usada VT100 (herança dos terminais de vídeo da Digital Equipment Corporation). Os códigos seguem todos a sequência de caracteres ESC[Xm, onde ESC corresponde ao carácter com o código (decimal 27 / hexadecimal 0x1B / octal 033), X é uma sequência variável de códigos separados por “;”. Os valores de X determinam a função a executar no terminal. As tabelas seguintes ilustram os diferentes códigos suportados pelos terminais VT100 e os respetivos comandos.

Código	Comando	Código	Comando
0	Reset / Normal	1	Negrito ou mais intenso
2	Menos intenso	3	Itálico
4	Sublinhado	5	Piscar lento
6	Piscar rápido	7	Inverter cores
9	Rasurado	10	Tipo de letra principal
11-19	Tipo de letra alternativo	21	Remove negrito
22	Cor e intensidade normais	23	Remove itálico
24	Remove sublinhado	25	Remove piscar
27	Remove inversão de cores	28	Revelar
29	Remover riscado	30-37	Cor do texto (tabela)
38	Cor do texto RGB	39	Cor por omissão
40-47	Cor de fundo (tabela)	48	Cor de fundo (RGB)
49	Default background color	51	Moldura
52	Círculo	53	Risco superior
54	Remove modura e círculo	55	Remove risco superior
90-97	Cor de texto brilhante	100-107	Cor de fundo brilhante

A.4. CÓDIGOS DE TERMINAL ANSI

Cor	Texto	Fundo
Preto	30	40
Vermelho	31	41
Verde	32	42
Amarelo	33	43
Azul	34	44
Rosa	35	45
Ciano	36	46
Branco	37	47
Preto brilhante	90	100
Vermelho brilhante	91	101
Verde brilhante	92	102
Amarelo brilhante	93	103
Azul brilhante	94	104
Rosa brilhante	95	105
Ciano brilhante	96	106
Branco brilhante	97	107

A utilização destes códigos passa por apresentá-los no terminal da mesma forma que se mostra a informação. Ao identificar a sequência o terminal trata o comando e não mostra essa sequência de caracteres. Este método designa-se por sinalização na banda, uma vez que o canal é partilhado para dados e controlo. Se o emulador de consola, ou terminal de texto, ao qual o sistema embebido é ligado suportar os códigos de terminal ANSI, como por exemplo o programa TeraTerm, é possível mostrar informação de uma forma mais estruturada em vez de escrever novas linhas e do simples rolar do texto. A diferença mais significativa no controlo do terminal passa por poder escrever pouco texto em determinadas coordenadas do terminal, evitando o processo moroso de reescrever todo o terminal. Isto é útil para criar animações tais como barras de progresso e realçar partes do texto com cor. O exemplo seguinte mostra na consola a frase “Ola Arduino” com “Ola” escrito a vermelho e “Arduino” escrito a verde. O último comando `[0m` serve para anular os anteriores antes da próxima escrita. `Serial.println(" 033[31;1;4mOla 033[32;1;4mArduino 033[0m");` As figuras seguintes ilustram o jogo clássico da cobra (snake) e um ambiente de janelas. Neste jogo a cobra não pode tocar nos limites do tabuleiro nem no próprio corpo que cresce de cada vez que apanha uma peça colocada aleatoriamente no tabuleiro. Usando um terminal ANSI o movimento da cobra corresponderá a fazer surgir um novo carácter na cabeça da cobra de acordo com a sua trajetória, e apagar o último carácter, correspondente à sua cauda.

A.5 Convenções de Programação

As convenções de programação (em Inglês *Coding Standards*) são conjuntos de regras adequadas a uma linguagem de programação, e que recomendam um determinado estilo de programação e organização do código fonte, para facilitar a leitura e posterior manutenção. Uma convenção de programação habitualmente cobre a organização da escrita do código fonte, nomeadamente:

- tamanho da indentação,
- formato dos comentários,
- número máximo de linhas de código por ficheiro fonte e por função,
- número máximo de caracteres por linha,
- declaração de variáveis e alocação de memória,
- estruturação da aplicação,
- convenção de nomes.

O Arduino fornece um conjunto de recomendações para a escrita de código em <https://www.arduino.cc/en/Reference/StyleGuide>. Existem outras convenções, nomeadamente: CERT C, MISRA C e High Integrity C++. Depende de quem define o projeto escolher qual a mais conveniente para usar. Sistemas críticos, por exemplo em automóveis e em equipamentos aeroespaciais, possuem convenções bastantes estritas para reduzir o risco de ocorrência de falhas.

Em programação os nomes são constituídos por uma palavra apenas. Para facilitar a criação de nomes compostos, como por exemplo o botão de pressão (push button), foram criadas convenções que permitem uma leitura mais fácil dos nomes, descritas na tabela A.1. Estas convenções evitam também que se gere confusão entre variáveis com nomes semelhantes.

A vantagem de `medialCapital` é ser uma escrita mais compacta ao passo que a `snake_case` permite uma leitura mais fácil. É habitual em programação em C++ usar `medialCapital` ou `snake_case` para variáveis, `ALLCAPS` para constantes e `CamelCase` para nomes de classes. Independentemente da escolha feita o importante é que seja adotada a mesma convenção em todo o projeto.

A.6 Segurança e Boas Práticas

Na realização de um circuito eletrónico, ou elétrico, deve-se ter sempre a preocupação de trabalhar sempre em segurança. Pensando na lei de Ohm, se existir um curto-circuito (uma resistência de valor muito baixo) num par de

A.6. SEGURANÇA E BOAS PRÁTICAS

Notação	Descrição	Exemplos
Medial Capital	Apenas iniciais das palavras, excepto a primeira são maiúsculas	<code>pushButton</code> , <code>isPressed()</code>
Camel Case	Todas as iniciais de palavras são maiúsculas	<code>PushButton</code> , <code>IsPressed()</code>
Snake Case	Todas as palavras são minúsculas separadas pelo caracter <code>_</code> (sublinhado ou underscore)	<code>push_button</code> , <code>is_pressed()</code>
All Caps	Semelhante a snake case mas com todas as letras maiúsculas	<code>PUSH_BUTTON_PIN</code> , <code>PIN_MASK</code>
Notação Húngara	Derivada de CamelCase onde as primeiras letras são minúsculas e representam o tipo ou o propósito da variável	<code>bButtonPressed</code> (binary), <code>iPinNumber</code> (integer), <code>strPushButtonName</code> (string)

Tabela A.1: Convenções de nomes mais comuns.

fios da rede elétrica, a potência dissipada é tremenda e muito provavelmente derreterá material e dará origem a fogo caso o circuito não seja rapidamente interrompido. Por este motivo existem os quadros elétricos à entrada de cada casa, ou escritório.

Desenvolver a curiosidade sobre o funcionamento das coisas, questionar o porquê das ligações, ter uma atitude de ir “espreitar debaixo do *capô*” e não ser só “dar à chave”/ligar a alimentação. É altamente recomendado ler a documentação e os manuais (*datasheet*) dos componentes e dos circuitos, antes de os testar e evitar o “experimentar para ver o que dá”, porque o mais provável é não funcionar ou até arder!

Os equipamentos ligados à rede elétrica devem ser protegidos por um fusível e ter ligação à terra (condutor amarelo e verde). Os fusíveis são o elemento mais básico de protecção. Eles são constituídos por um cilindro de vidro com dois terminais metálicos. Os terminais estão ligados por um fio muito fino, que foi dimensionado para se queimar/destruir quando correntes elétricas superiores às especificadas forem atingidas. Os fusíveis podem ser encontrados em quadros elétricos antigos e automóveis. No Reino Unido, as fichas e tomadas elétricas contêm um fusível e um interruptor para proteger a instalação elétrica de curto-circuitos que possam acontecer dentro dos aparelhos nela ligados. Apesar de serem baratos, os fusíveis têm o inconveniente de não serem reutilizáveis, e necessitam de serem substituídos. Os fusíveis são ligados em série, ou seja, ficam entre a alimentação e o circuito, e devem

ser escolhidos em função do circuito.

Devem-se isolar todas as ligações (soldaduras) para evitar curto-circuitos acidentais e potencial destruição de circuitos e componentes. Pode-se usar manga termo-retrátil ou fita adesiva do tipo electricista. A manga termo-retrátil é um tubo de material plástico que encolhe quando submetido a temperaturas elevadas, e isso permite moldar-se à forma dos fios. Por isso, a manga oferece maior robustez mecânica do que fita adesiva. A manga termo-retrátil existe em diferentes tamanhos e cores, ilustrado na figura A.1 à esquerda. É necessário ter atenção na escolha da largura da manga pois a sua capacidade de contração é limitada e caso os fios a proteger sejam mais finos, a sua aplicação não terá o efeito desejado. A manga começa a contrair por volta dos 70°C e vai até aos 125°C , e consegue contrair para cerca de $1/3$ do seu diâmetro original. Na prática, se o ferro de soldar permitir controlar a temperatura para 100°C , pode-se usar a ponta do ferro. Se não for possível controlar a temperatura, pode-se aproximar a ponta do ferro mas sem tocar na manga para não a derreter. Não se recomenda o uso de fósforos ou isqueiros. A figura A.1 à direita ilustra o aspeto final da aplicação da manga para proteger as soldaduras dos dois condutores. Por vezes, quando os condutores no interior são finos, aplica-se uma manga mais larga para acompanhar as várias ligações e reforçar a junção dos cabos.

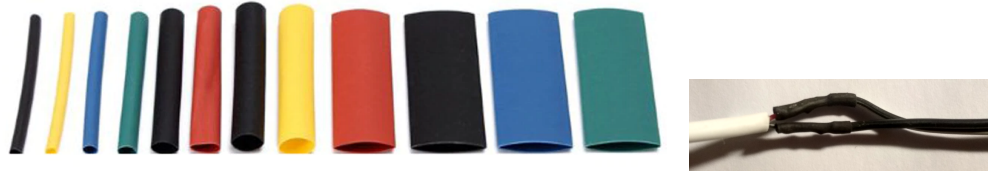


Figura A.1: Vários exemplos de manga termo-retrátil de diferentes cores e tamanhos (esquerda), e exemplo de soldaduras protegidas com dois pedaços de manga termo-retrátil (direita).

Todos os materiais bons condutores apresentam alguma resistência à passagem de corrente, incluindo os cabos e fios de ligação. A sua resistência é muito baixa, mas não é nula, e em diversos tipos de circuitos onde circulam correntes elevadas a sua impedância não deve ser desprezada por questões de segurança (ex: $1\Omega \times 2\text{ A} = 2\text{ V} \rightarrow P = 4\text{ W}$). Quanto mais comprido for o condutor, maior a sua resistência.

Se um condensador for removido de um circuito enquanto em funcionamento ele poderá manter a sua carga durante muito tempo, e se não for manuseado com cuidado pode provocar curto-circuitos. Os componentes semicondutores, mesmo desligados, não devem ser manuseados sem que exista uma ligação à terra para descarga da eletricidade estática. A eletricidade estática é adquirida pelo corpo humano, principalmente através do vestuário e calçado, e por muito breve e fraca que possa parecer é suficiente para da-

nificar alguns dispositivos semicondutores. A solução passa por usar uma pulseira antiestática ligada à terra e manusear os componentes sobre um tapete antiestático.

Deve haver cuidado na escolha do fornecedor de componentes eletrônicos, pois algumas precedências vendem componentes reciclados ou fabricados sob especificações diferentes, escrevendo apenas uma nova marcação no seu corpo. Para um sistema fabricado em larga escala isto pode significar centenas ou milhares de unidades defeituosas e atrasos na produção. A figura A.2 ilustra a medição de um potenciômetro de $10\text{ k}\Omega$, mas que, na realidade, tem apenas $8,75\text{ k}\Omega$, o que corresponde a um desvio de $-12,5\%$ (por defeito). Um lote de cinco potenciômetros comprado *on-line* foi medido com um multímetro, e registaram-se os seguintes valores de resistência: $9,99\text{ k}$, $9,71\text{ k}$, $9,50\text{ k}$, $8,75\text{ k}$ e $8,65\text{ k}\Omega$. É boa prática medir os componentes antes de produzir o circuito, pois no final pode ser difícil identificar qual o componente defeituoso, e avalia-lo depois de soldado no circuito.

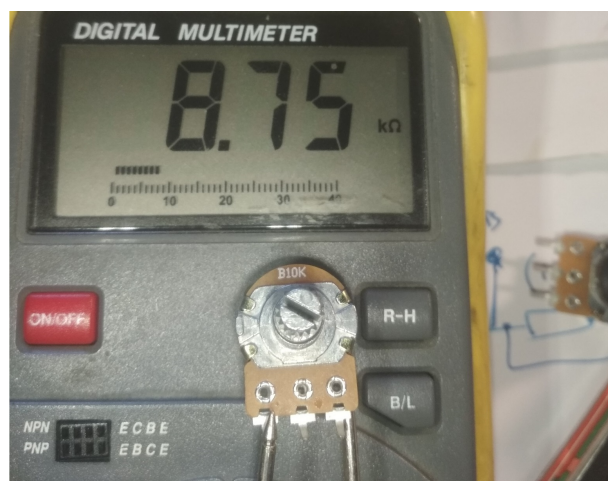


Figura A.2: Medida da resistência máxima de um potenciômetro com valor diferente do especificado ($10\text{ k}\Omega$).

A.7 Referência Rápida de C++

A listagem A.1 corresponde a um Sketch Arduino que demonstra o uso da maioria da sintaxe C/C++. Esta listagem procura ser um resumo breve e não ser exaustiva.

Listagem A.1: Demonstração de programação num Sketch Arduino

```
/* C/C++ programming language demo
   (this is a multi-line/block comment) */
// single-line comment

// Std libs
```


APÊNDICE A. ANEXOS

```
#include <stdio.h>
#include <arduino.h>
// ignore false/0 statements
#if 0
#include "my_lib.h"
#endif

// Macros - replaced with definition in the code
#define XPIO 1
#define N -1
#define bitXor(value, bit) ((value) ^= (1UL << (bit)))

// Enumerates
enum X { X1, X2, X3 } x;

// Constants
const int a = B1010; // 1010 (bin) = 10 (dec) = 0x0A (hex)
const unsigned int b = 10u;
const long l = 123456L;
const float pi = 3.14159;

// Structures
struct Xy {
    int x;
    int y;
} varXY;

// new type definitions
typedef unsigned char byte;
typedef enum enum_name { ZERO, ONE, TWO } enum_type;
typedef struct yx {
    int yy;
} typeYX;

// Classes
class Azz {
private:
    int a;
public:
    int z;
    int get_a() {
        return a;
    }
} az;

// Templates
template <typename T>
T func(T a, T b) {
    return a + b;
}

template <class T>
class Cl {
public:
    T func(T x, T y) {
        return x + y;
    }
};

// Global Variables
bool globalVar = true;
uint16_t zx = 0;
char buf[80];
enum_type bc;
Cl <int> intObj;
Cl <float> floatObj;

// auxiliary functions
void func1()
{
    bool localVar = false;
    uint16_t zx = 1; // different zx variable (local)
```

```

} //do stuff

int func2(int x3)
{
    if (x3 < 0)    {
        return 0;
    }
    return x3 - 1;
}

void logic_operators()
{
    // receive bool, return bool
    bool a = true, b = false, c;
    c = !a; // NOT ⇒ c = false
    c = a && b; // AND ⇒ c = false
    c = a || b; // OR ⇒ c = true
}

void bitwise_operators()
{
    byte a = 0x55, b = 0xAA, c;
    // 0x53 = 01010011, 0xA6 = 10100110
    c = ~a; // NOT ⇒ c = 0xAC
    c = a & b; // AND ⇒ c = 0x02
    c = a | b; // OR ⇒ c = 0xF7
    c = a ^ b; // XOR ⇒ c = 0xF5
    c = a >> 1; // SHIFT-RIGHT ⇒ c = 0x29
    c = a << 1; // SHIFT-LEFT ⇒ c = 0xA6
    c &= a; // AUGMENTED-AND ⇒ c = c & a
    c |= a; // AUGMENTED-OR ⇒ c = c | a
    c ^= a; // AUGMENTED-XOR ⇒ c = c ^ a
    c >>= a; // AUGMENTED-SHIFT-R ⇒ c = c >> a
    c <<= a; // AUGMENTED-SHIFT-L ⇒ c = c << a
}

void boolean_conditions()
{
    byte a = 1, b = 2;
    bool c;
    c = a == b; // equals ⇒ c = false
    c = a != b; // different than ⇒ c = true
    c = a > b; // greater than ⇒ c = false
    c = a >= b; // greater or equal than ⇒ c = false
    c = a < b; // lesser than ⇒ c = true
    c = a <= b; // lesser or equal than ⇒ c = true
}

void arithmetic_operations()
{
    int a = 5, b = 2, c;
    c = a + b; // ADD ⇒ c = 7
    c = a - b; // SUB ⇒ c = 3
    c = a * b; // MULT ⇒ c = 10;
    c = a / b; // DIV ⇒ c = 2;
    c = a % b; // REM ⇒ c = 1 (= a-(a/b)*b)
    c++; // INC ⇒ c = c + 1
    c--; // DEC ⇒ c = c - 1;
    c += b; // AUGMENTED-ADD ⇒ c = c + b
    c -= b; // AUGMENTED-SUB ⇒ c = c - b
    c *= b; // AUGMENTED-MUL ⇒ c = c * b
    c /= b; // AUGMENTED-DIV ⇒ c = c / b
    c %= b; // AUGMENTED-REM ⇒ c = c % b
}

// setup - configure peripherals
void setup()
{
    pinMode(LED.BUILTIN, OUTPUT);
}

```

```
// loop - main application
void loop()
{
    // function variables
    bool boolean_condition = false;
    int variable = 0;
    byte i, x;

    // *** main stuff ***
    if (boolean_condition == true) {
        // instructions to execute if true
    }
    else {
        // instructions to execute if false
    }

    switch (variable) {
        case 0: /* instructions to execute if variable = 1*/ break;
        case 1: /* instructions to execute if variable = 2*/ break;
        //...
        case N: /* instructions to execute if variable = N*/ break;
        default: /* instructions to execute by default*/ break;
    }

    // ternary operator: condition ? true statement : false statement
    i = x > 0 ? 1 : 0;
    x > 0 ? i = 1 : i = 0;

    // for(init values; loop condition; vars update) {
    for (x = 0, i = 1; (x < 10) && (i > -4) ; x += 2) {
        //instruction to execute while loop condition true
    }

    while (boolean_condition == true) {
        //instruction to execute while loop condition true
        if (x > 10) continue; // returns to the while
        //more instructions to execute
    }

    do {
        //instruction to execute at least once
    } while (boolean_condition == true); // if true then repeat
}
}
```

A.8 Erros Frequentes

Devido à quantidade de conceitos envolvidos, a falta de prática leva a que existem alguns erros típicos aconteçam. Esta secção aborda alguns desses erros, para que o leitor fique atento e não os cometa.

A.8.1 Erros em Arduino

Em relação ao Arduino os erros mais frequentes são:

- Esquecer de configurar os sinais de E/S corretamente.
- Não encontrar o Arduino para programar - Modelo de Arduino incorreto, ou porto de USB trocado. Este erro é frequente se existirem

vários periféricos USB com portas COM ligados no computador ou se forem usadas placas compatíveis com Arduino pois elas não informam o ambiente de desenvolvimento de qual a porta COM do Arduino.

- Deixou de conseguir programar o Arduino, depois de o ter programado anteriormente - Se está a programar o porto de comunicação série (Serial) para interagir com um periférico, não esquecer que este porto é partilhado com a interface de programação do Arduino. A solução passa por desligar o periférico durante a programação.
- Símbolos estranhos ou ausência de texto enviado pelo Arduino - significa que existe diferença entre o débito do Arduino e o terminal. A solução passa por ter a mesma configuração em ambos os lados.
- Não aparecem as primeiras mensagens enviadas para o Arduino, mas aparecem as seguintes - Não espera pela configuração da porta série antes de avançar com a execução do programa. Na maioria dos Arduínos este tempo é desprezável, mas nalguns modelos, esta configuração demora tempo e deve ser respeitada. Uma solução passa por colocar uma espera:

```
while(!Serial);
```

- O programa tem uma dinâmica diferente da esperada - demora mais tempo do que o intervalo de espera definido com `delay()`. Muito provavelmente estão a ser enviados caracteres para o terminal de texto. Funções como `printf` demoram bastante tempo e devem ser usadas com muito cuidado em aplicações que são sensíveis a variações de tempo. Se for mesmo imprescindível enviar dados pelo porto série deve-se usar o maior débito possível para reduzir o tempo de transmissão. Outra técnica passa pelo envio da mesma quantidade de caracteres independentemente do valor transmitido. Isto permite que essa instrução demore sempre o mesmo tempo e não haja variação em função do valor.

A.8.2 Erros em Programação

Existem alguns erros de programação que são recorrentes, ou porque a sintaxe é semelhante ou por esquecimento devido a complexidade do programa.

Falta de Organização

Um dos erros frequentes deve-se ao uso de versões de código e documentação erradas, alterações não registadas, vários ficheiros com o mesmo nome, trabalhar na pasta errada, projetos de electrónica e programação misturados.

Vale a pena criar uma pasta para cada projeto, e dentro de cada pasta ter sub-pastas organizadas por assunto: manuais, bibliotecas, código, esquemas

elétricos, circuitos impressos, programas, etc. A figura A.3 sugere um tipo de organização de ficheiros simples.

Para gerir as várias versões de código, as abordagens minimalistas vão desde a simples cópia do ficheiro fonte com a data da alteração até à utilização de repositórios `git` ou `svn`. Recomenda-se vivamente o investimento na aprendizagem da utilização de um desses tipos de repositórios pois não só ajudam na organização, como na realização de cópias de segurança. Por outro lado, desaconselha-se a utilização de um único ficheiro fonte para o desenvolvimento, pois existe a possibilidade de existirem erros em ficheiros, alterações não intencionais, ou código “cortado e colado” incorreto.

Figura A.3: Exemplo de organização das pastas de um projeto.

Erros no Código C/C++

Alguns erros são típicos devido às idiossincrasias das linguagens de programação e às distrações do programador. Em C/C++ as mais frequentes são:

- Usar variáveis não iniciadas. Nunca assumir que as variáveis têm valor 0, pois podem não ter.
- Usar `=` em vez de `==` - A instrução `if (x = 123) ...` não está a testar se `x` tem o valor 123, mas está a afetar. A forma correta seria `if (x == 123) ...`
- Num vetor de valores inteiros `int a[10]`; escrever um valor no último elemento com `a[10] = 99`; em vez de `a[9] = 99`; . Num vetor com `N=10` elementos, o primeiro índice é 0 e o último é `N-1`.
- Usar `=` em `#define`. O igual é apenas usado para definir variáveis e constantes `const int i = 1`; . Macros que são substituídas pelo compilador não usam `=`. Exemplo `#define TEMP_F(x) ((x) - 2)`. Recomenda-se usar os parêntesis para evitar erros devido a questões de precedência de operadores;
- usar `|` em vez de `||`. `1|4` usa o operador binário, ou seja, realiza a operação lógica OR entre 1 (001_b) e 4 (100_b) cujo resultado é 5 (101_b). `1||0` usa o operador lógico OR e considera os valores 0 ou diferente de 0 (normalmente 1). Outro exemplo: `int a = !3; // NOT: a = 0`;
- Esquecer as chavetas a seguir a um `if(..)`, `for(..)` ou `while()`. `if (a == 1) b = 2; c = 3`; faz com que `c` não seja afetado caso `a` seja igual a 3;

- Trocar `,` por `;`. O ciclo `for` é composto por três partes `for` (**condições iniciais**; **condição de paragem**; **atualiza iterador**) e suporta várias condições iniciais e atualizações dos iteradores separadas por vírgula, por exemplo `for (i=0; i < 10; i++, j--)`. Caso um ciclo `for` seja apenas escrito com vírgulas é como se tivesse apenas condições iniciais, e não tivesse condições de paragem nem atualizações dos iteradores;
- Usar `switch-case` sem `break`. Num `switch`, se um `case` não terminar com um `break`, a execução de instruções continua até encontrar um ou o `switch` terminar.
- Trocar ponteiros, referências e valores de variáveis. `int a = 1; int *b; b = &a; *b = 2;`. O valor do ponteiro `b` vai corresponder à posição de memória da variável `a`. Ao afetar `b` com o valor da referência (endereço) de `a`, o valor apontado (`*b`) vai corresponder à variável `a`, e `a` vai passar a ter o valor 2. Enquanto `b` não for afetado com a referência de `a`, não é possível usá-lo. Pode valer a pena enviar para a consola o valor de `a` e `b` antes e depois de cada instrução para perceber a diferença entre eles `printf("a: %d, b: %d, *b %d\n", a, b, *b);`.
- Problemas de herança de atributos e métodos entre classes (derivadas). Vale a pena usar um diagrama para representar as classes e planear a relação entre métodos atributos públicos e privados antes de iniciar a programação da solução.

A.8.3 Erros em Electrónica

Os erros típicos em electrónica são:

- Esquecer de ligar a alimentação ou cabo USB para programar.
- Trocar polaridade - de componentes e alimentações. Todos os componentes têm os seus pinos identificados, e estão explicados no seu *data sheet*. Para evitar trocar polaridade de alimentações deve-se usar a convenção de cores nos condutores: VCC -vermelho e GND - preto, identificar os pontos de alimentação na PCB na serigrafia, com um marcador permanente ou autocolante.
- Ligar circuitos 3,3V e 5V sem circuito de adaptação. O circuito de 3,3V provavelmente vai queimar (existem alguns que toleram 5V), mas se isso não acontecer o circuito a 5V pode não conseguir discriminar o nível 1 do circuito a 3,3V.

- Desenhar, ou fabricar, uma placa de circuito impresso virada do lado contrário;
- Trocar *footprints* de componentes. O *data sheet* identifica o encapsulamento do componente e dá as dimensões do *footprint*. Garantir que a referência do componente encomendado corresponde ao *footprint usado*.
- Esquecer de ligar a terra num circuito ligado a 220V.
- Medir uma resistência com o circuito ligado (A.D.). O multímetro possui uma pilha para provocar uma queda de tensão e aferir a resistência. Ao ligar as pontas de prova num circuito ligado pode promover a destruição do multímetro e ou do circuito testado.

A.9 Referências

Alguns endereços de sites na Internet com conteúdos sobre Arduínos e sistemas embebidos.

1. <http://www.arduino.cc/>
2. <http://www.arduino.cc/reference/en/>
3. <http://www.circuitdigest.com/>
4. <http://www.hackaday.com/>
5. <https://www.circuitmess.com/>
6. <https://www.elektor.com/>
7. <http://www.adafruit.com/>
8. <http://www.sparkfun.com/>
9. <http://www.dfrobot.com/>
10. <http://www.instructables.com/>

A.10 Fabricantes de Placas de Circuito Impresso

A maioria das fábricas aceita as encomendas on-line, onde validam os ficheiros submetidos e produzem o orçamento do fabrico no momento. O custo

A.10. FABRICANTES DE PLACAS DE CIRCUITO IMPRESSO

depende das dimensões da placa, número de tamanhos de furos, número total de furos, a largura das pistas e o espaçamento entre elas (quanto menor, maior será o custo), aplicação de verniz e serigrafia. Algumas lojas de componentes eletrônicos têm acordos com fabricantes locais que podem fabricar pequenos protótipos sem os acabamentos.

Muitos dos fabricantes de placas de circuito impresso produzem placas em grande volume e não fazem protótipos. Alguns fabricantes usados pelo autor do livro que produzem placas em pequena quantidade:

1. Eurocircuits, <https://www.eurocircuits.com>
2. OSHPark, <https://oshpark.com/>
3. PCB Train, <https://www.pcbtrain.co.uk/>