Chapter 4

Sistemas Embarcados: Uma Abordagem Prática com BitDogLab

Cairon Ferreira Prado, Darlys Ferreira Neris de Aguiar, Fabrício de Carvalho Mota, Jonathas Jivago de Almeida Cruz, Matusalen Costa Alves, Pedro Henrique Valentino

Abstract

This minicourse presents a practical introduction to embedded systems using the BitDogLab development board. Targeted at beginners with basic programming skills, the minicourse introduces essential hardware and software concepts through hands-on activities. Participants will work with digital I/O, analog sensors (ADC), pull-up/down resistors, and develop a simple interactive game. The proposal aims to foster interest in embedded systems through accessible and engaging experimentation.

Resumo

Este minicurso apresenta uma introdução prática aos sistemas embarcados utilizando a placa de desenvolvimento BitDogLab. Destinado a iniciantes com conhecimentos básicos em programação, aborda conceitos fundamentais de hardware e software por meio de atividades interativas. Os participantes trabalharão com entradas e saídas digitais, sensores analógicos (ADC), resistores pull-up/down e desenvolverão um jogo interativo simples. A proposta busca despertar o interesse por sistemas embarcados de forma acessível e didática.

4.1. Introdução

A educação em sistemas embarcados tornou-se um pilar para a formação de profissionais capazes de desenvolver tecnologias que interagem diretamente com o mundo físico. A onipresença desses sistemas — que permeiam veículos autônomos, eletrodomésticos inteligentes, dispositivos médicos e redes de manufatura — projeta um mercado que reforça a urgência de formar engenheiros com um domínio versátil e integrado de hardware e software [Pasricha 2022].

Historicamente, o ensino dessa área enfrenta desafios significativos. Currículos tradicionais de engenharia, muitas vezes compartimentados em domínios como Ciência da Computação e Engenharia Elétrica, dificultam a formação interdisciplinar que é essencial para o desenvolvimento de sistemas embarcados [Sztipanovits et al. 2005]. Como resposta a essa lacuna, os Computadores de Placa Única (SBCs), como Raspberry Pi e Arduino, foram amplamente adotados na educação, com estudos sistemáticos demonstrando um aumento no engajamento e na motivação dos alunos por meio de uma abordagem mais prática [Ariza and Baez 2021].

Nesse contexto, plataformas educacionais focadas na experimentação surgem como ferramentas poderosas de aprendizagem ativa. Frameworks pedagógicos recentes enfatizam a importância de um currículo que aborde desde os fundamentos de hardware e software até a integração de sensores, com foco em projetos práticos para capacitar profissionais para áreas como robótica e automação [Benyeogor et al. 2024].

A placa BitDogLab, objeto central deste minicurso, insere-se nesse movimento. Desenvolvida com base no Raspberry Pi Pico, ela apoia atividades STEAM (Ciência, Tecnologia, Engenharia, Artes e Matemática) e busca democratizar o acesso ao ensino de sistemas embarcados com uma baixa barreira de entrada. Ao combinar hardware aberto e suporte educativo, a plataforma permite que estudantes avancem de conceitos teóricos a projetos práticos de forma integrada [Fruett et al. 2024].

O restante deste capítulo está organizado da seguinte maneira: a Seção 4.2 explora os conceitos teóricos de sistemas embarcados; a Seção 4.3 apresenta em detalhes a placa BitDogLab; a Seção 4.4 detalha os fundamentos de programação e eletrônica necessários para a prática; a Seção 4.5 consolida o aprendizado com a construção de um jogo interativo; e, por fim, a Seção 4.6 conclui o trabalho.

4.2. Fundamentos de Sistemas Embarcados

Para explorar de forma prática o desenvolvimento de projetos com a placa BitDogLab, é essencial, primeiramente, estabelecer uma base conceitual sólida. A Engenharia de Sistemas Embarcados é um campo vasto e interdisciplinar, que combina conhecimentos de eletrônica, ciência da computação e engenharia de controle. Esta seção introduz os conceitos fundamentais da área, começando pela definição formal de um sistema embarcado e suas classificações. Em seguida, serão destacadas as principais diferenças que o distinguem da computação de propósito geral e, por fim, será apresentada a arquitetura típica que caracteriza esses sistemas, detalhando seus componentes de hardware e software.

4.2.1. O que é um Sistema Embarcado?

Um sistema embarcado é, em sua essência, um sistema computacional projetado para ser o "cérebro" oculto dentro de um dispositivo maior, com a missão de executar uma ou poucas funções de forma dedicada. Diferente de um computador de propósito geral (como um notebook, que pode rodar inúmeros programas diferentes), um sistema embarcado é uma combinação otimizada de hardware e software, desenvolvida sob medida para uma aplicação específica. Seus componentes — um processador, memória, interfaces de entrada e saída (I/O) e o software especializado conhecido como firmware — são dimensionados para operar sob restrições rigorosas de tempo de resposta, consumo de energia, custo

e tamanho físico. Essa natureza dedicada significa que o hardware e o software são intrinsecamente acoplados, priorizando a eficiência e a interação com o mundo físico em vez da versatilidade genérica [Wolf 2001, Micco et al. 2018, Kopetz 2022].

Para compreender a diversidade desses sistemas, eles podem ser classificados com base em suas demandas operacionais. A seguir, apresentamos uma taxonomia com as principais categorias [Akdur et al. 2018, Kopetz 2022, Pereira et al. 2017]:

1. **Tempo Real Estrito (Hard Real-Time):** São sistemas em que uma falha no cumprimento de um prazo pode ter consequências catastróficas. A correção do sistema depende criticamente do tempo. Exemplos clássicos incluem o sistema de freios ABS de um carro, um marca-passo cardíaco ou o piloto automático de uma aeronave.
2. **Tempo Real Suave (Soft Real-Time):** Nesses sistemas, a falha em cumprir um prazo resulta em uma degradação da qualidade ou do desempenho, mas não em uma falha crítica do sistema. Um exemplo é a transmissão de vídeo em uma smart TV, onde um pequeno atraso pode causar um travamento momentâneo na imagem.
3. **Missão Crítica (Safety-Critical):** Esta classificação abrange sistemas cuja falha pode resultar em danos significativos, ferimentos ou morte. Eles exigem processos de desenvolvimento e certificações de segurança rigorosos e são comuns em setores como o automotivo, médico e aeroespacial.
4. **IoT e Borda (Edge):** Focados em conectividade, esses sistemas operam na "borda" da rede, coletando e, muitas vezes, pré-processando dados localmente antes de enviá-los para a nuvem. O baixo consumo de energia é um requisito fundamental para esses dispositivos, que frequentemente operam com baterias.
5. **Sistemas Ciber-Físicos (CPS):** Representam uma integração profunda entre computação, rede e processos físicos. Eles operam em um ciclo de realimentação contínuo (malha fechada), onde sensores monitoram o ambiente e atuadores o modificam, como em robôs industriais ou redes elétricas inteligentes.

Uma característica fundamental que define o desenvolvimento de sistemas embarcados é a enorme importância dos requisitos não-funcionais. Estes requisitos descrevem como o sistema deve operar, em vez de o que ele deve fazer. Propriedades como o tempo máximo de resposta a um evento, o consumo de energia por operação, a confiabilidade ao longo de anos de uso contínuo e a tolerância a falhas são, muitas vezes, mais importantes do que a própria lógica da aplicação.

Devido a essa criticidade, o desenvolvimento de sistemas embarcados em setores regulados é guiado por normas de segurança e padrões formais rigorosos, como a ISO 26262 para a indústria automotiva, a IEC 62304 para software de dispositivos médicos e a DO-178C para a aviação. Essas normas impõem processos estritos de gerenciamento de requisitos, rastreabilidade e testes. Mesmo em um contexto educacional, a introdução a noções básicas de verificação e validação (como testes de software-in-the-loop e hardware-in-the-loop) prepara o estudante para as exigências do mercado, que enfrenta desafios constantes na especificação e validação desses requisitos não-funcionais complexos [Kopetz 2022, Pereira et al. 2017, Garousi et al. 2018].

4.2.2. Diferenças para a Computação de Propósito Geral

Sistemas embarcados distinguem-se fundamentalmente da computação de propósito geral — como desktops, laptops e servidores — pelo seu foco em tarefas específicas e pela sua profunda integração com o mundo físico [Elsevier / ScienceDirect 2025]. Enquanto um computador tradicional pode ser visto como uma ferramenta universal, projetada para executar múltiplas e variadas aplicações sob sistemas operacionais complexos, um sistema embarcado é uma ferramenta especialista, otimizada para executar sua função de forma autônoma e eficiente, muitas vezes sem intervenção humana constante [Wolf 2001, Kopetz 2022].

As diferenças se manifestam de forma concreta tanto no hardware quanto no software. Computadores de propósito geral são projetados para alto desempenho, equipados com processadores de múltiplos núcleos, gigabytes de memória RAM e sistemas de armazenamento massivo. Em contraste, o hardware de um sistema embarcado é minimalista e otimizado. Ele geralmente utiliza microcontroladores (MCUs) ou Sistemas em um Chip (SoCs), que integram processador, memória e periféricos (como portas de comunicação) em um único componente. A memória é limitada a kilobytes ou poucos megabytes, e o software, conhecido como firmware, é altamente especializado. Em muitos casos, em vez de um sistema operacional completo como Windows ou Linux, o sistema pode rodar diretamente sobre o hardware ou utilizar um Sistema Operacional de Tempo Real (RTOS), que é um software minimalista projetado para garantir a execução de tarefas dentro de prazos rigorosos [Kopetz 2022, Akesson et al. 2020].

Uma das distinções mais críticas reside na operação em tempo real. A maioria dos sistemas de propósito geral opera com base no desempenho médio; não há problema se um programa levar alguns milissegundos a mais para abrir. Em muitos sistemas embarcados, no entanto, a previsibilidade é essencial. Eles demandam respostas determinísticas a eventos externos, o que significa que uma ação deve ser concluída dentro de um prazo máximo garantido. Para assegurar essa previsibilidade, os engenheiros analisam métricas como o Pior Caso de Tempo de Execução (WCET). O sistema de airbag de um veículo, por exemplo, não pode depender de um "tempo médio" de resposta; ele deve acionar em milissegundos, sempre.

Além disso, a interação com o mundo físico é a principal razão de ser de um sistema embarcado. Ele utiliza sensores para perceber o ambiente (medindo temperatura, pressão, movimento) e atuadores para agir sobre ele (acionando motores, luzes, válvulas). Esse acoplamento direto com o hardware exige um ciclo de desenvolvimento e verificação distinto, que frequentemente inclui técnicas como testes de Hardware-in-the-Loop (HIL), onde o sistema real é testado em um ambiente que simula suas interações físicas. Essa abordagem contrasta com a computação tradicional, que foca em interfaces mais abstratas entre o usuário e o software [Akesson et al. 2020, Pereira et al. 2017, Garousi et al. 2018].

Em suma, a identidade dos sistemas embarcados deriva de seu caráter dedicado, que impõe um forte vínculo entre hardware e software. As restrições de custo, energia e tamanho, a necessidade de determinismo temporal e a integração direta com o ambiente físico orientam todas as escolhas de arquitetura, as práticas de desenvolvimento e as técnicas de validação, distinguindo-os fundamentalmente dos computadores de uso geral.

4.2.3. A Arquitetura Típica: Hardware e Software

A arquitetura de um sistema embarcado é caracterizada por uma integração eficiente entre seus componentes de hardware e software, projetada para otimizar a função específica do dispositivo. Embora as implementações variem enormemente, uma estrutura fundamental pode ser identificada na maioria dos projetos.

O núcleo de processamento do sistema é geralmente um microcontrolador (MCU) ou um System-on-Chip (SoC). O MCU pode ser entendido como o "cérebro" do dispositivo, pois integra em um único circuito a Unidade Central de Processamento (CPU), a memória (tanto a volátil, RAM, quanto a não-volátil, Flash) e uma variedade de periféricos. Esses periféricos incluem timers para o controle de tempo, portas de Entrada/Saída de Propósito Geral (GPIO), conversores analógico-digitais (ADC) e controladores para protocolos de comunicação como I²C e SPI, que permitem ao MCU interagir com outros componentes eletrônicos.

Para perceber o ambiente, o sistema utiliza sensores. Eles funcionam como os "sentidos" do dispositivo, capturando dados do mundo físico e convertendo-os em sinais elétricos que o microcontrolador pode processar. Exemplos incluem acelerômetros que medem movimento, microfones que captam som ou termistores que medem a temperatura. Para agir sobre o ambiente, o sistema emprega atuadores, que podem ser vistos como os "músculos". Componentes como motores, LEDs, telas ou relés recebem comandos elétricos do microcontrolador e os convertem em ações físicas, como movimento, luz ou som. Juntos, sensores e atuadores formam um ciclo de controle que pode ser de malha aberta (apenas executa uma ação) ou de malha fechada (reage continuamente às mudanças percebidas pelos sensores) [Kopetz 2022, Akdur et al. 2018].

No âmbito do software, o firmware é o programa especializado que orquestra todos esses componentes de hardware. Frequentemente escrito em linguagens de baixo nível como C ou C++, que oferecem controle direto sobre o hardware, o firmware é responsável por inicializar os periféricos, ler os dados dos sensores, executar a lógica da aplicação e enviar os comandos apropriados para os atuadores. Para sistemas que precisam gerenciar múltiplas tarefas com garantias de tempo, pode-se utilizar um (RTOS), como o FreeRTOS. Um RTOS é um sistema operacional minimalista cujo principal objetivo é garantir o determinismo temporal, ou seja, a capacidade de executar tarefas dentro de prazos rigorosamente definidos.

Essa arquitetura evidencia um conceito central da área: o co-design de hardware e software. As decisões tomadas em uma área impactam diretamente a outra. Por exemplo, a escolha de um sensor de baixo custo e menor precisão pode exigir algoritmos de filtragem mais complexos no software para compensar o ruído. Inversamente, um hardware mais poderoso pode simplificar o firmware. A análise desses trade-offs entre latência, consumo de energia e complexidade é uma habilidade crítica no desenvolvimento de sistemas embarcados e um aspecto fundamental para ilustrar a otimização integrada em projetos educacionais [Akdur et al. 2018, Micco et al. 2018].

4.3. Apresentando a Placa de Desenvolvimento BitDogLab

Após a exploração dos fundamentos teóricos dos sistemas embarcados, esta seção direciona o foco para a ferramenta de hardware que servirá como nosso laboratório prático: a placa de desenvolvimento BitDogLab. Criada para ser uma porta de entrada acessível ao universo da eletrônica e da programação de baixo nível, a BitDogLab materializa os conceitos de hardware e software em uma plataforma interativa. Iniciaremos com uma visão geral de sua concepção e filosofia educacional e, em seguida, faremos uma análise detalhada dos periféricos integrados que utilizaremos em nosso estudo de caso.

4.3.1. Visão Geral da Placa

A BitDogLab é uma placa educacional versátil, construída sobre o Raspberry Pi Pico, que visa democratizar o ensino de sistemas embarcados por meio de um ecossistema open-source. Seu objetivo principal é fomentar o aprendizado progressivo em programação, eletrônica e sistemas ciber-físicos, ao proporcionar um ambiente sinestésico que integra elementos visuais, auditivos e interativos. A plataforma incentiva a modificação colaborativa, permitindo que usuários copiem, fabriquem e aprimorem o design, o que a torna ideal para projetos educacionais que enfatizam a inovação e a colaboração [Fruett et al. 2024, Community 2025].

A versatilidade para a prototipagem rápida é um dos pilares do projeto. A BitDogLab oferece suporte a versões de montagem manual (through-hole) e de montagem em superfície (SMD), com todos os arquivos de design, incluindo esquemáticos e layouts, disponíveis no formato KiCad. Isso facilita a fabricação personalizada e o estudo aprofundado de seu circuito. A placa é programada principalmente em MicroPython, com firmwares específicos que já incluem bibliotecas para o controle de todos os periféricos integrados. O núcleo da placa, o microcontrolador RP2040, com seus dois núcleos ARM Cortex-M0+, 264 kB de SRAM e o subsistema de I/O Programável (PIO), habilita a experimentação com processamento paralelo e a criação de protocolos de comunicação customizados, permitindo que os projetos evoluam de simples jogos educativos a sistemas de sensoriamento ambiental de forma ágil [Foundation 2021, Community 2025, Industries 2025].

4.3.2. Periféricos Integrados: Visão Geral dos Componentes

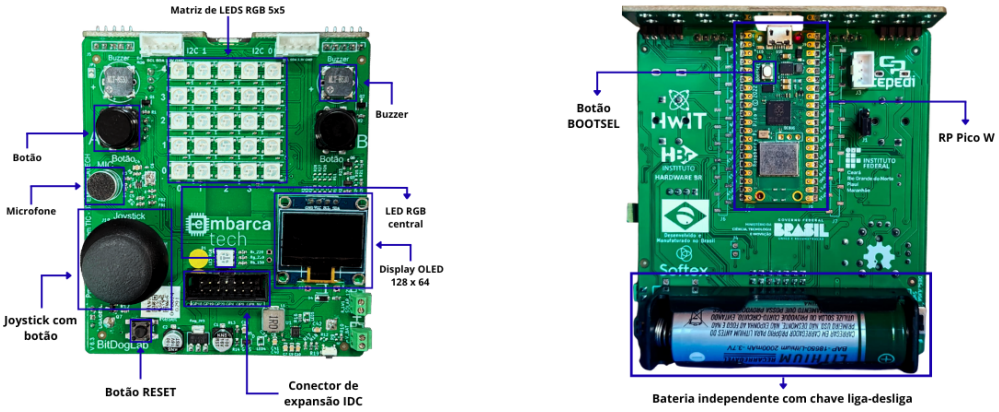
A Figura 4.1 apresenta as duas faces da placa BitDogLab, destacando a riqueza de componentes que a tornam um laboratório portátil completo para experimentação prática. Ao integrar sensores e atuadores diretamente no circuito, a placa elimina a necessidade de montagens complexas em protoboards, permitindo que o estudante foque na lógica de programação e na interação com o hardware desde o início.

A vista frontal (Subfigura 4.1a) concentra os componentes de interação com o usuário:

- **Display OLED:** Uma tela gráfica para exibir informações, menus e animações.
- **Matriz de LEDs:** Um conjunto de LEDs RGB endereçáveis (WS2812B) para feedback visual dinâmico.
- **Joystick Analógico:** Permite a entrada de dados em dois eixos (X e Y), ideal para

controle de movimento.

- **Botões de Usuário:** Entradas digitais para comandos discretos.
- **Microfone MEMS:** Um sensor para capturar som ambiente e criar aplicações reativas a ruídos.
- **Buzzer Piezoelétrico:** Um atuador para gerar feedback sonoro e melodias simples.



(a) Placa BitDogLab (vista frontal)

(b) Placa BitDogLab (vista reversa)

Figure 4.1: Visões da Placa BitDogLab

O verso da placa (Subfigura 4.1b) abriga os circuitos de suporte, como o sistema de alimentação, o módulo de carregamento para bateria de Li-ion e o botão *bootsel*, que facilita a gravação de um novo firmware através de uma simples interface de arrastar e soltar.

4.3.3. Periféricos Integrados: Análise Técnica

A Figura 4.1, apresentada anteriormente, serve como base para esta análise detalhada dos periféricos integrados da BitDogLab. A seguir, descrevem-se os principais componentes com foco em suas características técnicas — como protocolos de comunicação, resoluções e consumos de energia — e sua relevância para projetos educacionais em sistemas embarcados, promovendo a compreensão prática de conceitos como ADC, PWM e I²C.

- **Joystick Analógico:** Este componente serve como uma entrada de controle multidirecional. Ele está conectado a canais do Conversor Analógico-Digital (ADC) do RP2040, que possui uma resolução de 12 bits. Isso significa que a posição do joystick em cada eixo pode ser lida como um valor entre 0 e 4095, permitindo uma detecção de movimento precisa e gradual, ideal para interfaces interativas como jogos ou o controle de robôs [Foundation 2021, Community 2025].

- **Botões de Usuário:** Dois botões físicos oferecem entradas digitais robustas para interações simples, como selecionar menus ou disparar eventos. Eles são conectados a pinos GPIO configurados com resistores de *pull-down*, o que garante um estado lógico estável. O firmware pode ser programado para detectar o acionamento desses botões por meio de checagens contínuas ou, de forma mais eficiente, através de interrupções de hardware, um conceito fundamental em sistemas embarcados [Foundation 2021, Community 2025].

- **Buzzer Piezoelétrico:** Mapeado para um pino GPIO específico, este atuador sonoro permite a geração de tons e melodias simples. Seu funcionamento é controlado por modulação por largura de pulso (PWM), uma técnica que o RP2040 suporta nativamente. Ao variar a frequência e a largura do pulso, é possível controlar a nota e o volume do som, o que exemplifica a aplicação de conceitos de geração de sinais para fornecer feedback auditivo em projetos [Foundation 2021, Community 2025].

- **Microfone MEMS:** A placa integra um microfone digital (MP34DT01-M) que utiliza uma interface PDM (*Pulse Density Modulation*) para se comunicar com o microcontrolador. Com alta sensibilidade e baixo consumo de energia, este sensor é adequado para aplicações de sensoriamento de áudio, como o reconhecimento de comandos de voz simples ou a detecção de eventos sonoros (como uma palma) para acionar uma ação no sistema [STMicroelectronics 2015, Foundation 2021].

- **Tela OLED:** Para a saída de informações visuais, a BitDogLab conta com uma tela de matriz de pontos com resolução de 128x64 pixels, controlada pelo driver SSD1306. A comunicação com o microcontrolador é feita via protocolo I²C, um barramento serial comum em sistemas embarcados. A tela possui um buffer de memória interno, o que permite atualizações eficientes, e é perfeita para exibir menus, status de sensores ou interfaces gráficas simples, tornando os projetos mais interativos e informativos [Solomon Systech / Adafruit (mirror) 2010, Community 2025].

- **LEDs RGB Endereçáveis:** Um conjunto de LEDs RGB (baseados no chip WS2812B) oferece um recurso de feedback visual altamente dinâmico. Cada LED pode ser controlado individualmente para exibir qualquer uma das 16 milhões de cores, através de um protocolo serial de um fio. No RP2040, o subsistema de I/O Programável (PIO) é ideal para controlar esses LEDs, pois pode gerar os sinais de temporização precisos que o protocolo exige sem sobrecarregar a CPU principal [WORLDSEMI 2013, Foundation 2021, Fruett et al. 2024].

4.4. Conceitos Essenciais de Programação e Eletrônica

Com a familiaridade estabelecida com os componentes físicos da placa BitDogLab, esta seção aprofunda-se nos conceitos de programação e eletrônica fundamentais para dar vida a esses periféricos. Para construir aplicações interativas e que percebem o ambiente, é necessário compreender como o microcontrolador lê informações de sensores e como ele comanda os atuadores. Ao integrar noções de interfaces digitais e analógicas e gerenciamento de sinais, estes tópicos preparam o terreno para o desenvolvimento do nosso estudo de caso. Utilizando o microcontrolador RP2040 como referência, exploraremos as técnicas

para leitura e controle de periféricos, facilitando a transição de conceitos teóricos para protótipos funcionais [Foundation 2021, Fruett et al. 2024].

4.4.1. Entradas e Saídas Digitais (GPIO)

Os pinos de Entrada/Saída de Propósito Geral, conhecidos pela sigla GPIO (*General-Purpose Input/Output*), constituem a interface mais fundamental entre o microcontrolador e o mundo físico. Eles são canais de comunicação versáteis que podem ser configurados via software para operar de duas maneiras principais: como uma **entrada**, para ler informações do ambiente, ou como uma **saída**, para controlar dispositivos externos. O microcontrolador RP2040, presente na BitDogLab, oferece 30 desses pinos multifuncionais, que são a base para a interação com sensores e atuadores [Foundation 2021, Community 2025].

No modo digital, um pino GPIO opera com apenas dois estados lógicos: ALTO (*HIGH*) ou BAIXO (*LOW*). O estado ALTO corresponde a uma tensão elétrica específica (3.3V no RP2040), representando o valor binário "1", enquanto o estado BAIXO corresponde a 0V, representando o "0".

- **Configuração como Saída (Output):** Quando um pino é configurado como saída, o firmware pode controlar seu estado, "escrevendo" um valor ALTO ou BAIXO nele. Essa é a base para controlar atuadores digitais. Por exemplo, para acender um LED ou acionar o buzzer na BitDogLab, o firmware instrui o microcontrolador a colocar o pino correspondente em estado ALTO, fornecendo a tensão necessária para a ação. É crucial, no design de circuitos, respeitar a corrente máxima que cada pino pode fornecer (cerca de 12 mA no RP2040) para evitar danos tanto ao microcontrolador quanto ao componente externo.
- **Configuração como Entrada (Input):** Quando configurado como entrada, o pino "escuta" o nível de tensão presente nele, permitindo que o microcontrolador leia o estado de sensores digitais, como os botões da BitDogLab. Para detectar o pressionamento de um botão, o firmware monitora o estado do pino associado. A detecção pode ser feita de duas formas: por *polling*, onde o software verifica o estado do pino repetidamente em um laço, ou por *interrupções*, uma técnica mais eficiente onde o próprio hardware notifica a CPU quando uma mudança de estado (como uma borda de subida ou descida do sinal) ocorre.

O diagrama de pinagem, ou *pinout*, é a ferramenta de referência essencial para o desenvolvedor de sistemas embarcados. A Figura 4.2 exibe o pinout do Raspberry Pi Pico W, a base da nossa BitDogLab. Ele mapeia cada pino físico a sua numeração de GPIO (ex: GP0, GP1) e suas funções alternativas, como os canais de comunicação I²C, SPI e os canais do conversor ADC. A consulta a este diagrama é o primeiro passo para conectar e programar qualquer periférico externo.

A programação de GPIOs em sistemas como o RP2040 é robusta, com suporte a operações atômicas que garantem o acesso seguro aos pinos mesmo em aplicações com múltiplas tarefas concorrentes. Em projetos educacionais com a BitDogLab, a manipulação de GPIOs é o primeiro passo para a criação de aplicações interativas, como o nosso jogo

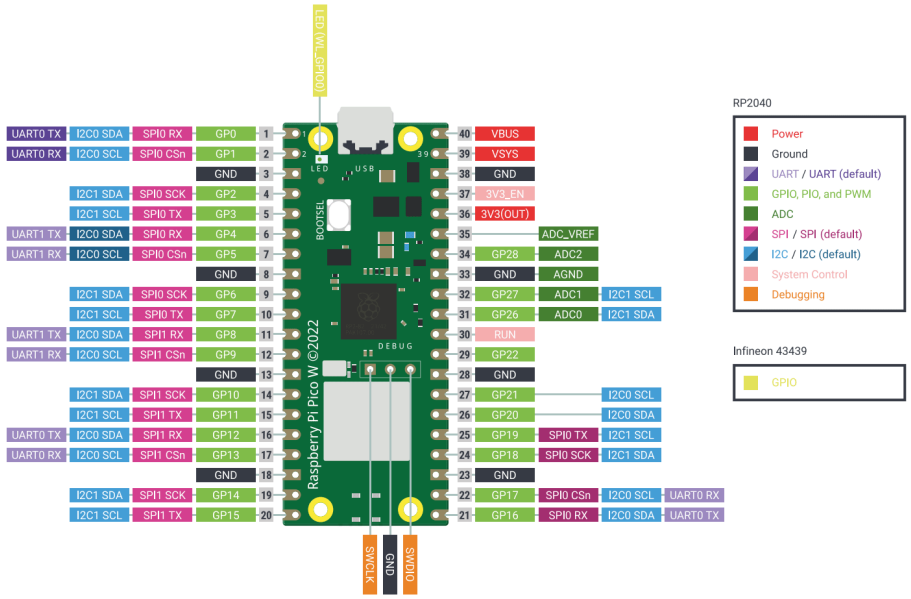


Figure 4.2: Diagrama de pinagem (pinout) do Raspberry Pi Pico W, base da BitDogLab.

de reação, e para o controle de periféricos mais complexos, como os LEDs RGB, que utilizam o subsistema de I/O Programável (PIO) para gerar os protocolos de comunicação necessários [Foundation 2021, WORLDSEMI 2013, Fruett et al. 2024].

4.4.2. Garantindo Sinais Estáveis: Resistores Pull-up e Pull-down

Ao trabalhar com entradas digitais, como botões, um dos desafios mais comuns é garantir que o microcontrolador realize uma leitura de sinal estável e previsível. Um pino GPIO configurado como entrada, quando não está conectado a um nível de tensão definido (nem 3.3V nem 0V), entra em um estado de "flutuação". Neste estado, o pino se comporta como uma pequena antena, suscetível a captar ruídos elétricos do ambiente, o que pode levar o microcontrolador a interpretar valores lógicos aleatórios (0s e 1s), resultando em um comportamento errático da aplicação [Instruments 2021, Foundation 2021].

Para resolver este problema, utilizamos resistores de *pull-up* ou *pull-down*. Esses componentes são essenciais para "ancorar" o pino de entrada a um estado lógico padrão quando o circuito está aberto (por exemplo, quando um botão não está pressionado).

- **Resistor de Pull-up:** Um resistor de *pull-up* conecta o pino GPIO a uma fonte de tensão positiva (VCC, ou 3.3V na BitDogLab). Com essa configuração, quando o botão está solto, o pino lê um estado lógico ALTO por padrão. Ao pressionar o botão, o circuito se fecha para o terra (GND), a corrente flui para o caminho de menor resistência, e o pino passa a ler um estado lógico BAIXO.

- **Resistor de Pull-down:** De forma análoga, um resistor de *pull-down* conecta o pino ao terra (GND). Neste caso, o estado padrão do pino quando o botão está solto é BAIXO. Ao pressionar o botão, o circuito se fecha para a fonte de tensão (VCC), e o

pino passa a ler um estado lógico ALTO.

Para simplificar o design de circuitos, microcontroladores modernos como o RP2040 incluem resistores de *pull-up* e *pull-down* internos, que podem ser habilitados via software para cada pino GPIO. Na BitDogLab, os botões já são projetados com essa estabilização. Compreender este conceito é vital em projetos educacionais, pois ensina sobre a importância da estabilidade de sinais e como prevenir falhas de leitura, uma das fontes mais comuns de bugs em sistemas embarcados [Instruments 2021, Community 2025, Foundation 2021].

4.4.3. Lendo o Mundo Analógico: O Conversor ADC

Enquanto os pinos GPIO em modo digital são perfeitos para ler estados discretos (ligado/desligado, pressionado/solto), o mundo físico é, em sua maior parte, analógico. Grandezas como temperatura, intensidade de luz e a posição de um joystick não variam em saltos, mas sim de forma contínua. Para que um microcontrolador, que opera no domínio digital, possa interpretar esses sinais, ele precisa de um tradutor: o **Conversor Analógico-Digital**, ou ADC (*Analog-to-Digital Converter*).

O ADC é um periférico fundamental que mede uma tensão analógica contínua em um pino e a converte em um valor numérico digital que o software pode processar. A precisão dessa conversão é definida pela **resolução** do ADC, medida em bits. O RP2040, por exemplo, possui um ADC de 12 bits, o que significa que ele pode representar a faixa de tensão de entrada (de 0 a 3.3V) em 2^{12}, ou 4096, níveis distintos. Na prática, isso permite leituras muito precisas de sensores como o joystick da BitDogLab, onde um pequeno movimento pode ser detectado como uma mudança sutil no valor digital lido [Foundation 2021, Kester 2009].

Outro parâmetro crucial é a **taxa de amostragem**, que define quantas vezes por segundo o ADC realiza uma conversão. Para reconstruir um sinal analógico de forma fiel no domínio digital, a teoria da comunicação estabelece que a taxa de amostragem deve ser pelo menos o dobro da frequência máxima do sinal que se deseja medir, um princípio conhecido como Teorema de Nyquist-Shannon [Shannon 1948].

No contexto da BitDogLab, o ADC é a ponte que permite ao firmware ler a posição exata do joystick e a intensidade do som captado pelo microfone. No entanto, a aquisição de dados analógicos no mundo real apresenta desafios, como o ruído elétrico, que pode causar flutuações nas leituras. Para mitigar isso, técnicas de software, como a aplicação de médias móveis (*moving average*) ou *oversampling*, são frequentemente utilizadas para filtrar o ruído e obter um valor mais estável e preciso. A compreensão desses conceitos e técnicas é essencial para o desenvolvimento de sistemas embarcados que interagem de forma confiável com o ambiente [STMicroelectronics 2009, Lee and Levin 2025, Fruett et al. 2024].

4.5. Estudo de Caso: Construindo um Jogo de Velocidade

Este estudo de caso aplica os conceitos teóricos e práticos discutidos anteriormente em um projeto interativo e educativo, utilizando a placa BitDogLab para desenvolver o jogo "Ligeirinho" — uma aplicação de medição de tempo de reação que integra programação embarcada, interfaces digitais e analógicas, e periféricos sensoriais. Baseado no microcon-

trolador RP2040, o projeto exemplifica o co-design hardware-software, com foco em entradas/saídas GPIO, gerenciamento de sinais, temporização e feedback multimodal (visual, auditivo e tátil). Essa implementação prática reforça princípios de sistemas embarcados, como determinismo temporal e eficiência energética, enquanto promove experimentação em contextos STEAM, alinhando-se a abordagens educacionais que incentivam a prototipagem rápida e a depuração iterativa [Fruett et al. 2024, Alves 2025, Foundation 2021].

4.5.1. Definição do Escopo e Lógica do Jogo

O fluxo lógico segue uma estrutura sequencial com estados bem definidos: (i) inicialização e exibição de mensagem de prontidão no display OLED; (ii) detecção de pressão no botão A para iniciar; (iii) fase de preparação com LED verde aceso e atraso randômico (1-5 segundos); (iv) ativação da fase de reação com LED vermelho, buzzer e início de temporizador; (v) captura de tempo ao pressionar B, com cálculo de latência em milissegundos; (vi) exibição do resultado no OLED e reset para nova rodada. Essa lógica incorpora mecanismos de debouncing para estabilidade de sinais e interrupções para respostas determinísticas, destacando restrições de tempo real soft (onde atrasos degradam a experiência, mas não causam falhas críticas). O escopo enfatiza acessibilidade, utilizando firmware em C com SDK do RP2040, e incentiva extensões como multiplayer ou integração com sensores adicionais para explorar trade-offs em complexidade e consumo [Alves 2025, Foundation 2021, Kopetz 2022].

4.5.2. Implementação Prática do Firmware

A implementação do firmware para o "Ligeirinho" é realizada em linguagem C utilizando o SDK 2.1.0 do Raspberry Pi Pico, com configuração via CMakeLists.txt para compilar o executável e vincular bibliotecas essenciais (pico_stdlib, hardware_timer, hardware_pwm, hardware_clocks, hardware_i2c). Esse arquivo gerencia o processo de build de forma cross-platform, definindo padrões de C/C++ (C11 e C++17), importando o SDK via pico_sdk_import.cmake e inicializando-o com pico_sdk_init(). Ele também especifica o board como pico_w (para suporte Wi-Fi, embora não usado aqui, permitindo extensões futuras), ativa saídas de depuração via USB/UART (pico_enable_stdio_usb e pico_enable_stdio_uart) e gera outputs extras como .uf2 para upload bootloader [Pi 2025, Foundation 2021, Alves 2025].

Um trecho chave extraído do CMakeLists.txt ilustra a adição do executável e vinculação de bibliotecas, destacando a integração com periféricos do RP2040 (timers, PWM, clocks e I²C para OLED):

```
# Adiciona o arquivo-fonte correto
add_executable(Ligeirinho Ligeirinho.c inc/ssd1306_i2c.c)

# Define o nome e a versão do programa
pico_set_program_name(Ligeirinho "Ligeirinho")
pico_set_program_version(Ligeirinho "0.1")

# Ativa saída USB para depuração
pico_enable_stdio_uart(Ligeirinho 0)
pico_enable_stdio_usb(Ligeirinho 1)

# Adiciona bibliotecas necessárias
```

```
target_link_libraries(Ligeirinho pico_stdlib hardware_timer
    hardware_pwm hardware_clocks hardware_i2c)

# Inclui diretórios do projeto
target_include_directories(Ligeirinho PRIVATE ${CMAKE_CURRENT_LIST_DIR
    })

# Gera arquivos adicionais necessários para o Pico
pico_add_extra_outputs(Ligeirinho)
```

Listing 4.1: Configuração do ambiente de compilação para o projeto Ligeirinho com o SDK do Raspberry Pi Pico

Aqui, add_executable compila os fontes principais (Ligeirinho.c e a biblioteca OLED ssd1306_i2c.c), enquanto target_link_libraries vincula módulos hardware para suporte a timers (medição de reação), PWM (controle de LEDs e buzzer), clocks (gerenciamento de frequência) e I²C (display). A ativação de stdio USB facilita depuração via serial, crucial para testes educativos, e pico_add_extra_outputs gera binários como .uf2 para upload direto, promovendo iterações rápidas sem ferramentas externas. Essa configuração exemplifica boas práticas em co-design, onde o build reflete restrições embarcadas como memória limitada (via stdlib minimalista) e determinismo temporal (via hardware_timer para WCET baixo) [Pi 2025, Akdur et al. 2018].

O código integra conceitos da Seção 1.4, como GPIOs para entradas/saídas digitais (botões e LEDs), resistores pull-up internos para estabilidade de sinais, e PWM para controle de brilho e áudio, evitando estados de flutuação e garantindo respostas precisas. A seguir, apresentam-se trechos comentados do arquivo Ligeirinho.c, destacando a aplicação prática na BitDogLab [Alves 2025, Foundation 2021, Community 2025].

Primeiro, a inicialização de hardware e periféricos demonstra o uso de GPIOs como entradas com pull-up (para botões, prevenindo flutuação conforme Seção 1.4.2) e saídas PWM para LEDs e buzzer (controlando duty cycle para eficiência energética):

```
// Inicializa a interface I2C para o display OLED
i2c_init(i2c1, ssd1306_i2c_clock * 1000);
gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
gpio_pull_up(I2C_SDA);
gpio_pull_up(I2C_SCL);

// Inicializa o display OLED e exibe mensagem inicial
ssd1306_init();
display_text("PRESSIONE A PARA COMECAR!");

//Configura os botões como entradas com pull-up interno
gpio_init(BUTTON_START);
gpio_init(BUTTON_STOP);
gpio_set_dir(BUTTON_START, GPIO_IN);
gpio_set_dir(BUTTON_STOP, GPIO_IN);
gpio_pull_up(BUTTON_START);
gpio_pull_up(BUTTON_STOP);

// Inicializa os LEDs para PWM
```

```
pwm_init_led(LED_GREEN);
pwm_init_led(LED_RED);
// Inicialmente, ambos os LEDs estão desligados
pwm_set_gpio_level(LED_GREEN, 0);
pwm_set_gpio_level(LED_RED, 0);

// Inicializa o buzzer com PWM
pwm_init_buzzer(BUZZER);
```

Listing 4.2: Inicialização de periféricos da BitDogLab para o jogo Ligeirinho

Aqui, gpio_pull_up aplica resistores internos (aprox. 50-80kΩ) para estabilidade, enquanto pwm_init_led/buzzer configura slices PWM com wrap fixo (1000 para LEDs, ajustável para buzzer), ilustrando controle de saídas digitais moduladas para feedback sensorial [Instruments 2021, Foundation 2021].

Em seguida, a lógica de debouncing e interrupções para botões integra detecção de bordas e temporização, evitando ruídos em entradas digitais:

```
bool debounce_button(uint gpio)
{
    static uint32_t last_time = 0;
    uint32_t current_time = to_ms_since_boot(get_absolute_time());

    if (current_time - last_time < 50)
    {
        return false;
    }

    last_time = current_time;
    return gpio_get(gpio) == 0;
}

// Configura a interrupção para o botão B (BUTTON_STOP)
gpio_set_irq_enabled_with_callback(BUTTON_STOP, GPIO_IRQ_EDGE_FALL,
   true, &gpio_callback);
```

Listing 4.3: Implementação de debouncing e interrupção para botões no jogo Ligeirinho

Essa função debounce_button filtra pressões rápidas (<50ms), complementando pull-ups para sinais estáveis, enquanto interrupções (EDGE_FALL) garantem respostas em tempo real sem polling constante, otimizando consumo [Foundation 2021, Instruments 2021].

A fase de jogo utiliza timers absolutos para medição precisa e rand() para atraso aleatório, aplicando conceitos de tempo real:

```
void start_game()
{
    // ... (preparação com LED verde via PWM)
    pwm_set_gpio_level(LED_GREEN, LED_ON);

    uint delay_ms = 1000 + (rand() % 4000);
    for (uint i = 0; i < delay_ms / 10; i++)
    {
        sleep_ms(10);
```

```
        if (gpio_get(BUTTON_STOP) == 0)
        {
            false_start_detected = true;
            break;
        }
    }

    // ... (ativação de LED vermelho, buzzer e timer)
    pwm_set_gpio_level(LED_GREEN, 0);
    pwm_set_gpio_level(LED_RED, LED_ON);
    buzzer_beep(3000, 300);
    start_timer();
    reaction_phase = true;
}

// Callback de interrupção para botão B
void gpio_callback(uint gpio, uint32_t events)
{
    if (gpio == BUTTON_STOP && game_running && reaction_phase)
    {
        reaction_time = get_absolute_time();
        button_b_pressed = true;
    }
}
```

Listing 4.4: Lógica principal e controle do jogo Ligeirinho

Finalmente, o loop principal e cálculo de tempo unem esses elementos, com exibição via OLED (usando I²C, embora não haja ADC explícito aqui, o framework suporta extensões analógicas como joystick para variações):

```
while (true)
{
    if (debounce_button(BUTTON_START))
    {
        if (!game_running)
        {
            start_game();
        }
        sleep_ms(300);
    }

    if (game_running && reaction_phase && button_b_pressed)
    {
        uint32_t elapsed_time = get_elapsed_time();
        pwm_set_gpio_level(LED_RED, 0);
        stop_buzzer(0, NULL);

        char buffer[20];
        sprintf(buffer, "Tempo: %.1f ms", (float)elapsed_time);
        display_text(buffer);

        sleep_ms(5000);
        // Reset estados
        game_running = false;
```

```
        reaction_phase = false;
        // ...
        display_text("PRESSIONE A PARA COMECAR!");
    }
}
```

Listing 4.5: Loop principal do jogo Ligeirinho para controle de botões e exibição de resultados

Essa estrutura aplica GPIOs, pull-ups e timers da Seção 1.4, com PWM para atuadores, promovendo depuração via hardware-in-the-loop [Alves 2025, Garousi et al. 2018].

4.5.3. Análise e Execução do Projeto

Ao executar o firmware na BitDogLab (compilado e carregado via USB em modo bootloader), o comportamento esperado inicia com a mensagem "PRESSIONE "A" PARA COMECAR!" no OLED. Pressionar o botão A acende o LED verde (brilho 50% via PWM), seguido de atraso aleatório; em seguida, o LED vermelho acende com beep no buzzer (3000Hz por 300ms), iniciando o timer. Pressionar B captura o tempo (ex.: "Tempo: 250.0 ms" exibido), desliga LEDs e reseta após 5s. Queimas de largada piscam o LED vermelho três vezes com mensagem "MUITO CEDO!". Essa execução destaca estabilidade de sinais (sem flutuações graças a pull-ups e debouncing) e determinismo temporal (latências <1ms via interrupções), com consumo eficiente (50mA em operação). Incentiva-se experimentação: ajuste delays para simular hard real-time, integre joystick via ADC para controles analógicos, ou adicione métricas WCET para análise de desempenho, fomentando iterações educativas e depuração de falhas como ruídos em botões [Alves 2025, Foundation 2021, Kopetz 2022, Garousi et al. 2018].

4.6. Conclusão

Este capítulo apresentou uma jornada introdutória ao universo dos sistemas embarcados, partindo dos pilares teóricos até a aplicação prática dos conceitos na placa de desenvolvimento BitDogLab. Iniciamos com a definição de sistemas embarcados como soluções computacionais dedicadas, otimizadas para operar sob restrições de tempo real, energia e custo, em contraste com a flexibilidade da computação de propósito geral [Wolf 2001, Kopetz 2022]. A análise da arquitetura típica destacou o acoplamento intrínseco entre hardware — microcontroladores, sensores e atuadores — e software — o firmware —, enfatizando o co-design como uma estratégia essencial para balancear requisitos funcionais e não-funcionais [Akdur et al. 2018, Micco et al. 2018].

A apresentação da BitDogLab como uma ferramenta educacional open-source demonstrou sua versatilidade para a prototipagem rápida. A exploração de seus periféricos integrados permitiu a aplicação prática de conceitos fundamentais, como o uso de pinos GPIO para o controle de entradas e saídas digitais, a aplicação de resistores de pull-up/down para garantir a estabilidade de sinais e a utilização do conversor ADC para a leitura de sensores analógicos [Foundation 2021, Fruett et al. 2024, Instruments 2021]. O estudo de caso com o jogo "Ligeirinho" consolidou esses aprendizados, ao exigir a implementação de uma lógica com temporização precisa, feedback multimodal (visual e sonoro) e um ciclo de depuração interativo, ilustrando a integração hardware-software em um contexto

STEAM acessível [Alves 2025, Garousi et al. 2018].

A integração entre hardware e software emerge, portanto, como o cerne do desenvolvimento de sistemas embarcados. O projeto "Ligeirinho" exemplifica essa sinergia, onde a escolha de usar PWM para controlar a intensidade dos LEDs e os tons do buzzer, ou o protocolo I²C para se comunicar com a tela OLED, são decisões coordenadas que influenciam diretamente o firmware. Essa experiência prática prepara os aprendizes para os desafios da indústria, que incluem a conformidade com normas de segurança (como a ISO 26262) e a análise de falhas, além de fomentar a criatividade para a prototipagem de novos sistemas ciber-físicos [Pereira et al. 2017, Pasricha 2022]. A BitDogLab, com seu design colaborativo, oferece um ponto de partida ideal para a experimentação, incentivando o aprendizado contínuo em áreas como robótica, automação e dispositivos interativos, alinhando-se às demandas de um campo em constante evolução [Fruett et al. 2024, Industries 2025, Sztipanovits et al. 2005].

References

[Akdur et al. 2018] Akdur, D., Comer, R., and Magedanz, T. (2018). Co-design and co-simulation of heterogeneous embedded systems: A survey. *Simulation Modelling Practice and Theory*, 89:276–292.

[Akesson et al. 2020] Akesson, B., Nasri, M., Nelissen, G., Altmeyer, S., and Davis, R. I. (2020). An empirical survey into industry practice in real-time embedded systems. *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*.

[Alves 2025] Alves, C. M. (2025). Ligeirinho — jogo de reflexo com bitdoglab. GitHub Repository. Jogo de tempo de reação que utiliza LEDs, botões, buzzer e display OLED em C com SDK do Raspberry Pi Pico (RP2040). Acesso em: 6 de setembro de 2025.

[Ariza and Baez 2021] Ariza, J. A. and Baez, H. (2021). Understanding the role of single-board computers in engineering and computer science education: A systematic literature review. *Computer Applications in Engineering Education*.

[Benyeogor et al. 2024] Benyeogor, M. S., Benyeogor, A., Olaiya, K. A., and Agumey, P. (2024). Advancing embedded systems education: A pedagogical programming framework for smart system and control applications. *TechRxiv / arXiv preprint*.

[Community 2025] Community, B. (2025). Bitdoglab — open-hardware educational platform (readme and hcd). `https://github.com/BitDogLab/BitDogLab`. Acesso em: 1 de setembro de 2025.

[Elsevier / ScienceDirect 2025] Elsevier / ScienceDirect (2025). Embedded computer — sciencedirect topics. `https://www.sciencedirect.com/topics/computer-science/embedded-computer`. Acesso em: 1 de setembro de 2025.

[Foundation 2021] Foundation, R. P. (2021). Rp2040 microcontroller datasheet. `https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf`. Acesso em: 1 de setembro de 2025.

[Fruett et al. 2024] Fruett, F., Barbosa, F. P., Fraga, S. C. Z., and ao Guimarães, P. I. A. (2024). Empowering steam activities with artificial intelligence and open hardware: The bitdoglab. *IEEE Transactions on Education*, 67(3):462–471.

[Garousi et al. 2018] Garousi, V., Felderer, M., and Ribeiro, L. (2018). Testing embedded software: A survey of the literature. *Information and Software Technology*, 95:123–147.

[Industries 2025] Industries, A. (2025). Bitdoglab is a raspberry pi pico learning lab which encourages ai-assisted programming. `https://blog.adafruit.com/2025/02/03/bitdoglab-is-a-raspberry-pi-pico-learning-lab-which-encourages-ai-assisted-programming`. Acesso em: 1 de setembro de 2025.

[Instruments 2021] Instruments, T. (2021). Implications of slow or floating cmos inputs (application report scba004e). Texas Instruments Application Report SCBA004E, July 1994 – Rev. E (Reviewed July 2021). `https://www.ti.com/lit/an/scba004e/scba004e.pdf`. Acesso em: 1 de setembro de 2025.

[Kester 2009] Kester, W. (2009). Adc architectures ii: Successive approximation adcs (mt-021 tutorial). Analog Devices Tutorial. `https://www.analog.com/media/en/training-seminars/tutorials/mt-021.pdf`. Acesso em: 1 de setembro de 2025.

[Kopetz 2022] Kopetz, H. (2022). *Real–Time Systems: Design Principles for Distributed Embedded Applications*. Springer. Referência moderna que cobre requisitos temporais em sistemas embarcados.

[Lee and Levin 2025] Lee, M. and Levin, M. (2025). Anti-alias filter design for a 100 khz bandwidth data acquisition signal chain. Texas Instruments Application Note SBAA655A. `https://www.ti.com/lit/pdf/sbaa655a/sbaa655a.pdf`. Acesso em: 1 de setembro de 2025.

[Micco et al. 2018] Micco, L. D., Vargas, F. L., and Fierens, P. I. (2018). A literature review on embedded systems. *IEEE Latin America Transactions*. Survey com definição e panorama de sistemas embarcados.

[Pasricha 2022] Pasricha, S. (2022). Embedded systems education in the 2020s: Challenges, reflections, and future directions. *arXiv preprint*, arXiv:2206.03263.

[Pereira et al. 2017] Pereira, T., Albuquerque, D., Sousa, A., Alencar, F., and Castro, J. (2017). Retrospective and trends in requirements engineering for embedded systems: A systematic literature review. *Proceedings of the 20th Workshop on Requirements Engineering (WER 2017)*. Revisão sistemática sobre engenharia de requisitos em sistemas embarcados.

[Pi 2025] Pi, R. (2025). raspberrypi/pico-sdk. GitHub repository. SDK oficial para desenvolvimento em C/C++ para microcontroladores RP-series (RP2040 / Pico), com APIs para GPIO, PWM, I²C, timers, PIO e mais. Acesso em: 1 de setembro de 2025.

[Shannon 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3–4):379–423, 623–656.

[Solomon Systech / Adafruit (mirror) 2010] Solomon Systech / Adafruit (mirror) (2010). Ssd1306 oled driver – datasheet. `https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf`. Acesso em: 1 de setembro de 2025.

[STMicroelectronics 2009] STMicroelectronics (2009). Understanding and minimising adc conversion errors: Application note an1636. `https://www.st.com/resource/en/application_note/an1636-understanding-and-minimising-adc-conversion-errors-stmicroelectronics.pdf`. Acesso em: 1 de setembro de 2025.

[STMicroelectronics 2015] STMicroelectronics (2015). Mp34dt01-m – mems digital microphone datasheet. `https://www.st.com/resource/en/datasheet/mp34dt01-m.pdf`. Acesso em: 1 de setembro de 2025.

[Sztipanovits et al. 2005] Sztipanovits, J., Biswas, G., Frampton, K., Gokhale, A., et al. (2005). Introducing embedded software and systems education and advanced learning technology in an engineering curriculum. *ACM Transactions on Embedded Computing Systems*, 4(3):549–568.

[Wolf 2001] Wolf, W. (2001). *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufmann. Livro-texto clássico sobre sistemas embarcados.

[WORLDSEMI 2013] WORLDSEMI (2013). Ws2812b intelligent control led – datasheet. `https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf`. Acesso em: 1 de setembro de 2025.