# DISCIPLINA EA801
# Laboratório de Projetos de Sistemas Embarcados

## Circuitos Auxiliares de Microcontroladores

## Circuitos de Interface, Circuitos de Alimentação, Circuitos de Oscilação e Aterramento

**Profs. Fabiano Fruett, José Raimundo de Oliveira, Antonio Augusto Fasolo Quevedo e Wu, Shin-Ting**

**FEEC / UNICAMP**

**Editado em julho de 2025 com suporte de Chatgpt, Gemini e NotebookLM**

# INTRODUÇÃO

No universo dos sistemas embarcados, um projeto de sucesso vai muito além de escrever código. Tão crucial quanto a lógica de programação, o *software*, é sua base física: o *hardware*. É no *hardware* que os sinais elétricos se manifestam e as interações físicas acontecem. Sem garantir o fluxo adequado desses sinais em termos de tensão, corrente, tempo e forma, os comandos do microcontrolador, por mais perfeitos que sejam no *software*, não terão efeito no mundo real.

Este roteiro serve como um guia prático para projetar circuitos de interface, alimentação e oscilação. Esses são componentes críticos que não vêm integrados no microcontrolador, mas são absolutamente essenciais para o funcionamento de qualquer sistema embarcado. Vamos entender como as decisões de *design* de *hardware* são tomadas para resolver problemas que o *software*, sozinho, não consegue contornar, como ruídos elétricos, a compatibilidade de níveis de tensão e a necessidade de uma alimentação estável. Além de detalhar o funcionamento desses circuitos, veremos como documentar a concepção de um *hardware* através de esquemáticos por um aplicativo que suporta ainda a geração de placas de circuitos impressos (em inglês, *Printed Circuit Board* – PCB). Afinal, um bom projeto de hardware é a base sólida onde todo o potencial de um *software* pode realmente se manifestar.

# DESAFIO

Este projeto tem como objetivo principal o desenvolvimento de um sistema embarcado funcional utilizando o *kit* BitDogLab (BDL) e a placa Raspberry Pi Pico RP2040. A equipe será responsável por identificar um problema (real ou hipotético) e propor uma solução focada em microcontroladores. A equipe deve usar os periféricos *on-board* do *kit* e pelo menos um periférico *off-board* com comunicação serial (UART, I2C, SPI, etc). Monte em um ***protoboard*** o circuito de interface do periférico *off-board* usado no projeto. Implemente o *firmware* em **linguagem C**, utilizando o Visual Studio Code como ambiente de desenvolvimento. Para interagir com o *hardware*, use as funções de alto nível do Pico SDK, que são prefixadas com `pico_` (por exemplo, `pico_time`), evitando o acesso direto aos registradores de *hardware*.

Recursos disponíveis são

- *Kit* de Desenvolvimento BDL (com placa RP2040 e diversos periféricos).
- *Probe* de depuração DAPLink,
- *Protoboard* e conectores.
- Documentação técnica do microcontrolador (*datasheet*, Raspberry Pi Pico-series C/C++ SDK).
- Documentação técnico da placa Raspberry Pi Pico (*datasheet*), do BDL (esquemático) e dos periféricos (*datasheets*).

- Ambientes de desenvolvimento integrados (IDEs) e *toolchains* relevantes: Visual Studio Code.
- Ambiente de edição de esquemáticos: KiCAD.
- Exemplos de programação em C para os *firmwares* do Raspberry Pi Pico disponíveis no Visual Studio C.
- Apoio do professor para dúvidas técnicas e metodológicas.

O projeto será dividido nas seguintes fases:

1. Fase de Conceituação e Requisitos
   - Definição do Problema: A equipe deverá identificar e descrever um problema que possa ser solucionado por um sistema embarcado. O problema deve ser claro e permitir a aplicação dos conceitos estudados.
   - Levantamento de Requisitos: Com base no problema, a equipe definirá os requisitos funcionais (o que o sistema deve fazer) e requisitos não-funcionais (desempenho, custo, tamanho, consumo de energia, etc.).
   - Justificativa Técnica: Elaborar uma breve justificativa da relevância e viabilidade técnica da solução proposta.
2. Fase de Análise Técnica e Seleção de Periféricos:
   - Seleção de Periféricos do *Kit* BDL:
     - Com base nos requisitos definidos, a equipe deverá identificar e selecionar os periféricos necessários do *kit* BDL e no mínimo um novo periférico.
     - Justificativa para cada periférico: Explicar por que cada periférico selecionado é o mais adequado tecnicamente para cumprir sua função no sistema, considerando suas especificações (precisão do sensor, capacidade de corrente, tipo de interface de comunicação, etc.).
3. Fase de Implementação e Testes
   - Desenvolvimento de *Software***:** Implementar o *firmware* do sistema no microcontrolador selecionado, utilizando a linguagem C/C++. O código deve ser bem estruturado, modularizado e utilize a sintaxe Doxygen para a documentação dos programas em C.
   - Desenvolvimento de *Hardware***:** Desenhar o esquemático do circuito de interface entre os periféricos *off-board* com os conectores da BitDogLab e implementá-lo em *protoboard*.
   - Integração *Hardware-Software***:** Conectar o circuito em *protoboard* com a placa de desenvolvimento no *kit* BDL e realizar a integração física e lógica.
   - Testes**:** Realizar testes de unidade, integração e validação do sistema para garantir que todos os requisitos funcionais e não-funcionais sejam atendidos. Isso abrange testar a leitura de sensores, o acionamento de atuadores e a lógica de controle, quando aplicável.
4. Entregas do Projeto
   - Documento de Projeto (Relatório no arquivo README.md do repositório do projeto no GitLab seguindo o modelo, e o *link* do projeto no Moodle).
   - Esquemáticos com as dimensões dos *footprints* corretas:
     - Circuitos de interface dos periféricos *off-board* usados no projeto.
   - Código-Fonte:
     - Código completo do *firmware*, organizado e comentado, no repositório de controle de versão GitLab da Unicamp.
   - Demonstração do Protótipo:
     - Apresentação do sistema funcionando no *kit* BDL, com o circuito adicional montado no *protoboard*, durante a aula de entrega do projeto.

## Avaliação

O projeto será avaliado com base na:

- funcionalidade da solução implementada.
- qualidade do código (estrutura, clareza, documentação).
- qualidade do esquemático (legibilidade, *footprints*).
- capacidade de aplicar os conceitos de sistemas embarcados.
- organização e clareza do relatório final e da apresentação.

# FUNDAMENTOS TEÓRICOS

Esta seção é dedicada aos fundamentos teóricos do projeto de *hardware* essenciais para qualquer sistema embarcado. Exploraremos os circuitos de interface, que atuam como tradutores e adaptadores entre o microcontrolador e os diversos periféricos, garantindo compatibilidade e proteção. Abordaremos também os circuitos de alimentação, responsáveis por fornecer a energia limpa e estável necessária para o funcionamento confiável de todos os componentes. Por fim, discutiremos os circuitos de relógio, com foco no circuito de oscilação, que é o coração pulsante do sistema, fornecendo a base de tempo para todas as operações digitais. A compreensão desses elementos é o pilar para construir sistemas embarcados que não apenas funcionem, mas funcionem de forma precisa e confiável.

## Circuitos de interface

Os microcontroladores modernos integram uma ampla variedade de recursos nos seus pinos de entrada e saída digital e analógica, permitindo que eles interajam com o ambiente ao seu redor de maneira eficiente e versátil. Esses recursos incluem conversores analógico-digitais (ADC), saídas PWM, interfaces de comunicação como I2C, SPI e UART. Além disso, eles já vêm com proteções básicas contra curto-circuitos e controle de corrente nos pinos GPIO. Contudo, na prática, suas capacidades integradas frequentemente não são suficientes para garantir uma interface segura, compatível e eficiente com todos os periféricos externos. É aqui que os circuitos de interface se tornam indispensáveis.

**Circuitos de interface para microcontroladores** são componentes ou conjuntos de componentes eletrônicos externos que atuam como um “tradutor” ou um “protetor” que permitem que o microcontrolador interaja de forma robusta e segura com os diversos dispositivos e o ambiente ao seu redor. Esses circuitos complementam as funcionalidades do microcontrolador, resolvendo incompatibilidades e garantindo a operação adequada de todo o sistema. Eles são essenciais para a adaptação de níveis de tensão e corrente, permitindo que um microcontrolador que opera em baixa tensão (por exemplo, 3.3V) se comunique com sensores industriais ou acione componentes que demandam tensões ou correntes muito mais altas (como relés ou motores), utilizando transistores ou *drivers* dedicados.

Além disso, os circuitos de interface são cruciais para a proteção contra ruídos e interferências eletromagnéticas, filtrando sinais indesejados e utilizando componentes como diodos de proteção ou diodos de roda livre para salvaguardar o microcontrolador de picos de tensão. Eles também garantem o isolamento elétrico, frequentemente através de optoacopladores, que permitem a transmissão de sinais sem uma conexão elétrica direta, prevenindo danos e aumentando a segurança em sistemas com diferentes potenciais de terra ou tensões perigosas. Finalmente, esses circuitos otimizam a aquisição de sinais analógicos de alta precisão e facilitam o controle de barramentos

compartilhados, usando buffers de nível lógico para lidar com diferenças de tensão bidirecionais e garantir a integridade do sinal em comunicações de alta velocidade.

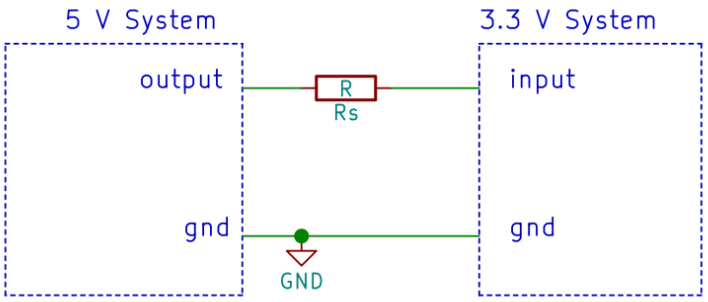
Esta apostila abordará detalhadamente esses tipos de circuitos de interface, explorando as razões técnicas para sua utilização e apresentando as soluções comuns. Serão discutidos circuitos para adaptação de níveis de tensão e corrente, como divisores de tensão, buffers, transistores e drivers dedicados. Também serão abordadas as técnicas para proteção contra ruídos e interferências, incluindo filtros e diodos de proteção, além da importância dos optoacopladores para isolamento elétrico. A apostila cobrirá ainda os desafios na aquisição de sinais analógicos de alta precisão e as soluções para controle de barramentos compartilhados, oferecendo um guia completo sobre como fazer os microcontroladores se comunicarem de forma robusta e segura com o mundo físico.

## Adaptação de níveis de tensão e corrente

A integração de microcontroladores com o mundo físico exige atenção a detalhes críticos, como a adaptação de níveis de tensão e a amplificação de corrente. Estas são duas das funções mais fundamentais dos circuitos de interface, garantindo a compatibilidade elétrica e a segurança operacional entre o microcontrolador e os elementos externos.

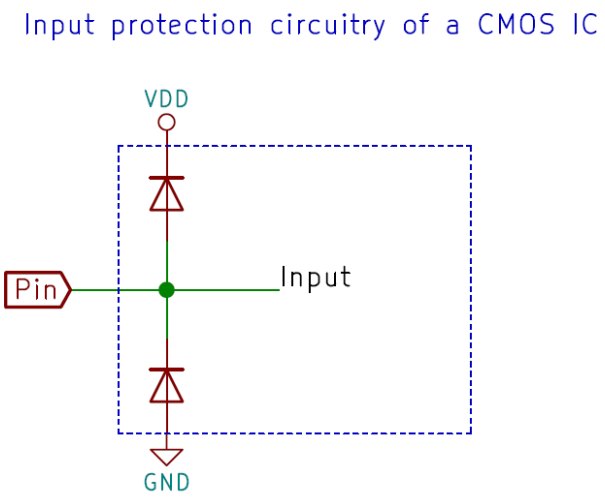
A necessidade de adaptação de tensão surge porque a maioria dos microcontroladores modernos opera com níveis lógicos de 3,3 V ou 5 V. Contudo, muitos sensores industriais, atuadores ou circuitos de comunicação, como módulos RS-232, sensores industriais de 24 V, ou interfaces com lógica de 1,8 V, operam com tensões incompatíveis com as entradas e saídas diretas do microcontrolador. Conectar diretamente uma saída digital com nível alto de tensão superior ao máximo tolerado pela entrada do microcontrolador pode causar um fluxo de corrente excessivo, resultando em danos permanentes tanto à saída quanto à entrada.

Para mitigar esses riscos, são empregadas soluções de adaptação. Quando não há uma tolerância explícita, uma alternativa é o uso de um simples resistor em série, para limitar a corrente, como mostra a seguinte figura.



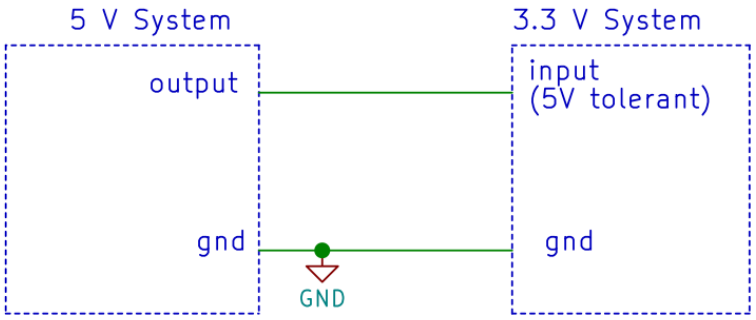

Fonte: Next-hack.

Isto pode ser feito porque a maioria das entradas digitais possui diodos de proteção, denominados **diodos *clamp*** mostrados na seguinte conexão, que normalmente são polarizados de forma a não conduzir, mas que conduzem quando a tensão na entrada digital ultrapassa os limites da tensão de alimentação.



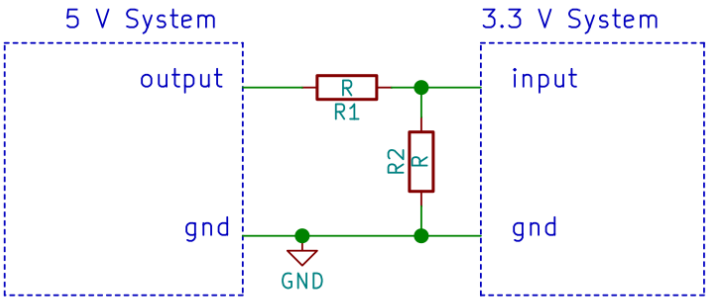

Fonte: Next-hack.

Sem um resistor série, a corrente na saída digital seria muito alta. É importante ressaltar que essa solução com resistor em série só deve ser usada quando há certeza da presença desses diodos de proteção na entrada do componente. Além disso, a combinação da resistência do resistor com as capacitâncias da entrada digital e da linha de conexão forma um circuito RC. Essa constante de tempo RC limita a velocidade de transferência do sinal, o que pode ser um fator limitante para aplicações que exigem alta frequência.Alguns circuitos integrados (CIs) já vêm com circuitos de proteção internos nas entradas, como resistores em série e diodos limitadores de tensão, que os tornam diretamente compatíveis com diferentes níveis. Por exemplo, certos microcontroladores alimentados com 3,3 V podem ter pinos de GPIO específicos marcados como “tolerantes a 5 V”, permitindo sua conexão direta a saídas digitais que operam com 5 V no nível lógico alto.



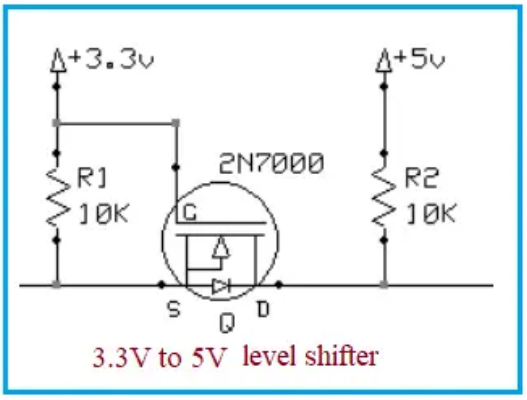

Fonte: Next-hack.

Em outros casos, a adaptação pode ser feita por meio de soluções mais simples com resistores (como divisores de tensão). Um exemplo clássico é a comunicação entre um microcontrolador de 3,3 V e um sensor digital que opera com nível lógico de 5 V. Se o sinal de saída do sensor (5 V) for aplicado diretamente à entrada do microcontrolador de 3,3 V, pode ocorrer sobrecarga e dano ao circuito de entrada. Para resolver isso, pode-se usar um divisor de tensão resistivo, composto por dois resistores em série. Por exemplo, um divisor com R1 = 1,8 kΩ e R2 = 3,3 kΩ converterá 5 V em aproximadamente 3,3 V na entrada do microcontrolador. Essa solução é simples e barata, mas adequada apenas para sinais digitais de baixa frequência e sem requisitos de precisão elevada.




Fonte: Next-hack.

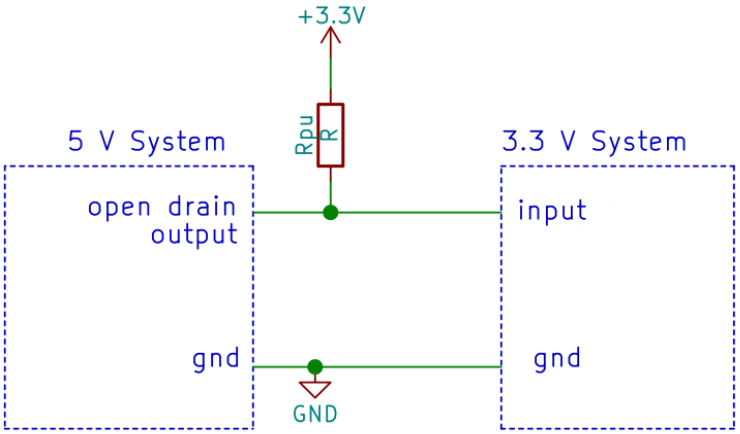
O MOSFET 2N7000, um transistor de efeito de campo de canal N de modo de intensificação, atua essencialmente como uma chave eletrônica controlada por tensão, permitindo que uma pequena tensão em seu terminal de porta (em inglês, *gate*) controle um fluxo de corrente significativamente maior entre seus terminais de dreno (em inglês, *drain*) e fonte (em inglês, *source*). Essa característica o torna uma solução superior ao simples divisor de tensão resistivo para a comunicação entre um microcontrolador de 3,3V e um sensor digital de 5V. Enquanto um divisor de tensão sofre com limitações de frequência e perda de sinal, o 2N7000, conectado com sua fonte ao terra, sua porta à saída de 5V do sensor e seu dreno a um resistor de *pull-up* conectado à alimentação de 3,3V do microcontrolador, transforma eficientemente os sinais. Quando o sensor envia 0V, o MOSFET desliga, puxando a entrada do microcontrolador para 3,3V; ao enviar 5V, o MOSFET liga, aterrando a entrada do microcontrolador para 0V. Embora essa configuração inverta a lógica do sinal (um 5V do sensor se torna 0V para o microcontrolador), o 2N7000 oferece uma comutação rápida, baixo consumo de corrente da fonte do sinal e níveis lógicos de saída bem definidos, características que garantem a integridade do sinal, mesmo em frequências mais elevadas, superando em muito a performance de um divisor resistivo para interfaces digitais.




Fonte: RFWireless World.

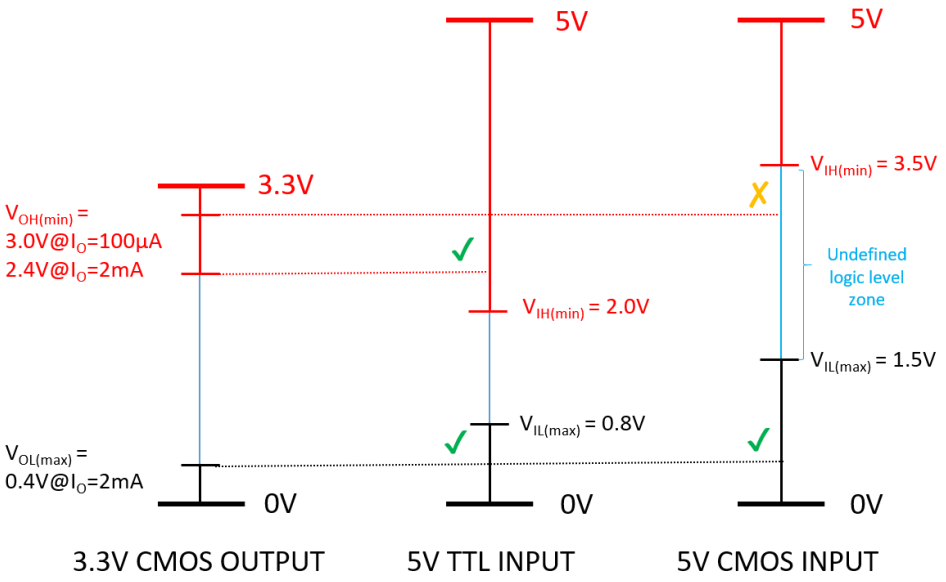
A configuração de saída *Open-Collector* (em transistores bipolares) ou *Open-Drain* (em transistores de efeito de campo, como MOSFETs) é uma forma muito eficaz e simples de adaptar níveis de tensão. Isso é especialmente útil quando se precisa conectar um sensor ou dispositivo com saída digital de 5V a uma entrada de um microcontrolador ou outro CI que opera com 3.3V de forma segura e simples. Isso é possível porque, no estado de “nível alto”, essas saídas não fornecem tensão ativamente; elas apenas “flutuam”. Ao adicionar um resistor de pull-up conectado à tensão de 3.3V

na linha de sinal, ele “puxa” a tensão para 3.3V quando a saída do sensor está em nível alto (desligada), enquanto no nível baixo (ligada), a saída puxa a linha diretamente para 0V. Assim, ambos os níveis lógicos são compatíveis, protegendo o CI de 3.3V contra sobretensão e eliminando a necessidade de circuitos complexos, como ilustra a seguinte figura.



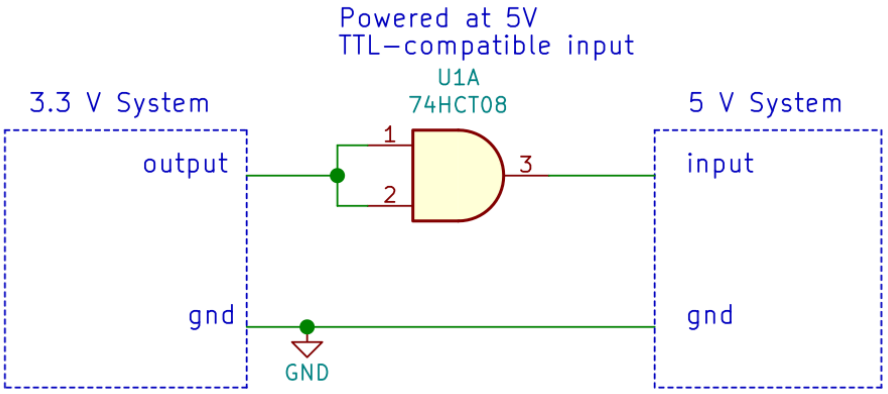

Fonte: Next-hack.

Quando uma saída digital de um Circuito Integrado opera com uma tensão menor do que a entrada de outro CI ao qual está conectada, pode haver incompatibilidade nos níveis lógicos. Isso é especialmente crítico para o nível lógico alto, que pode não ser reconhecido corretamente. A figura a seguir ilustra essa questão, comparando as faixas de tensão dos níveis lógicos de uma saída CMOS de 3.3V com entradas TTL de 5V e CMOS de 5V. Se uma saída CMOS de 3.3V for conectada a uma entrada TTL de 5V, as faixas de tensão dos níveis lógicos da saída são compatíveis com a entrada. Isso significa que, se a entrada do CI for especificada como “TTL 5V”, a conexão pode ser feita diretamente, garantindo a compatibilidade dos sinais. Por outro lado, se a entrada for de um CI CMOS de 5V, a compatibilidade ocorre apenas para o nível lógico baixo. Os valores de tensão do nível lógico alto da saída de 3.3V caem na região indefinida da entrada CMOS de 5V. Nesses casos, é necessário utilizar um circuito de compatibilização (também conhecido como *level shifter*) para ajustar os níveis de tensão e garantir a comunicação correta.




Fonte: Next-hack.

Uma solução bastante direta é usar como intermediária entre a entrada e a saída uma porta lógica de saída não-inversora, alimentada com os 5V e com entrada compatível com TTL. A figura mostra o uso de uma porta AND com suas entradas ligadas juntas, formando um simples *buffer*. O CI 74HCT08, apesar de usar tecnologia CMOS, possui entradas compatíveis com TTL (seu *datasheet* informa que "Inputs are TTL-Voltage compatible"). Como a saída da porta AND segue o padrão para 5V, a entrada do próximo CI receberá os níveis lógicos corretos.

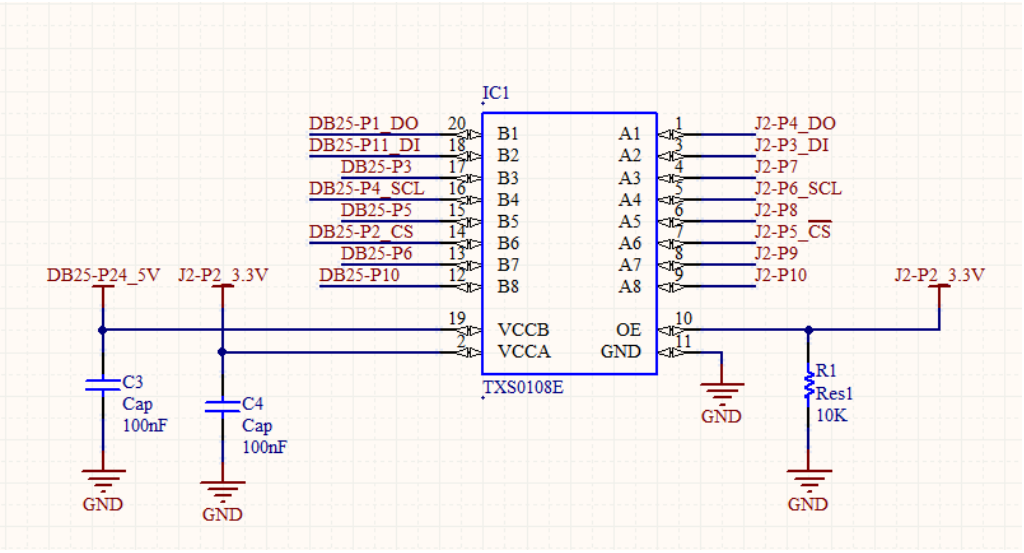



Fonte: Next-hack.

Para aplicações que exigem maior robustez, precisão e suporte a sinais de alta frequência, a solução mais sofisticada é o uso de ***level shifters*** dedicados. Estes são circuitos integrados projetados especificamente para traduzir níveis de tensão entre diferentes lógicas, oferecendo funcionalidades como tradução bidirecional, alta velocidade e imunidade a ruídos. Exemplos incluem o TXS0108E e *buffers* de nível lógico similares, como o 74LVC245, que se tornam indispensáveis para interfaces como I2C, SPI ou UART em cenários mais exigentes.

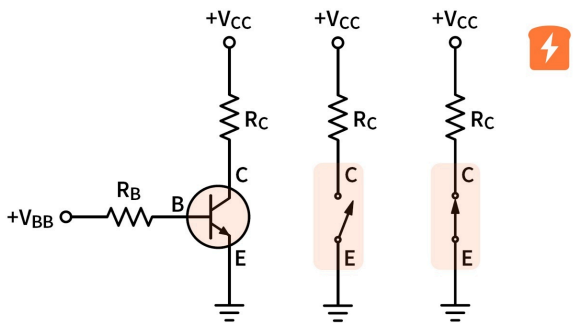
A necessidade desses componentes surge principalmente da incompatibilidade de níveis lógicos bidirecionais, onde microcontroladores e periféricos operam em voltagens distintas. O TXS0108E,

por exemplo, executa essa conversão automaticamente em ambas as direções. Além disso, esses *buffers* dedicados mantêm a integridade do sinal em altas velocidades e longas distâncias, atuando como *drivers* de linha para reforçar o sinal e preservar a forma de onda. Eles também oferecem uma proteção elétrica adicional contra surtos de tensão ou descarga eletrostática (em inglês, *Electrostatic Discharge* – ESD). Em suma, enquanto um MOSFET como o 2N7000 pode resolver a conversão de nível para sinais digitais de forma eficiente, um *level shifter* dedicado como o TXS0108E é fundamental para a robustez e confiabilidade de comunicações seriais mais complexas e bidirecionais.

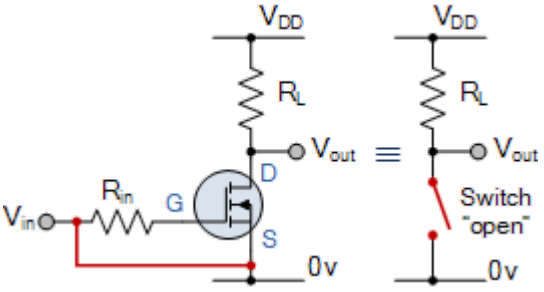



Fonte: Texas Instruments.

Outra situação comum ocorre quando o microcontrolador precisa controlar cargas que exigem mais corrente do que ele pode fornecer. Tipicamente, um pino de saída digital consegue fornecer entre 10 mA e 30 mA, o que é insuficiente para acionar diretamente um motor DC, um relé eletromecânico, ou mesmo um LED de alta potência. Nesses casos, utilizamos transistores bipolares (BJT) ou transistores de efeito de campo (MOSFETs) como **chaves eletrônicas**.



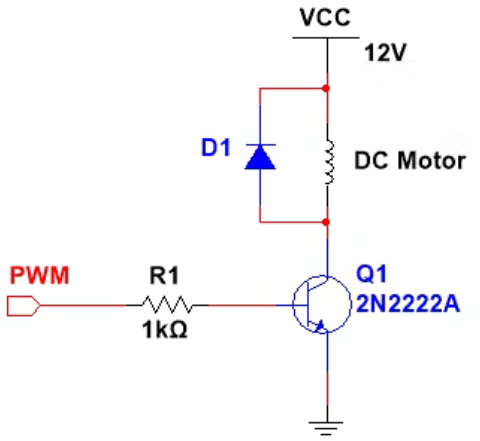

Fonte: CircuitBread.




Fonte: Electronics Tutorials.

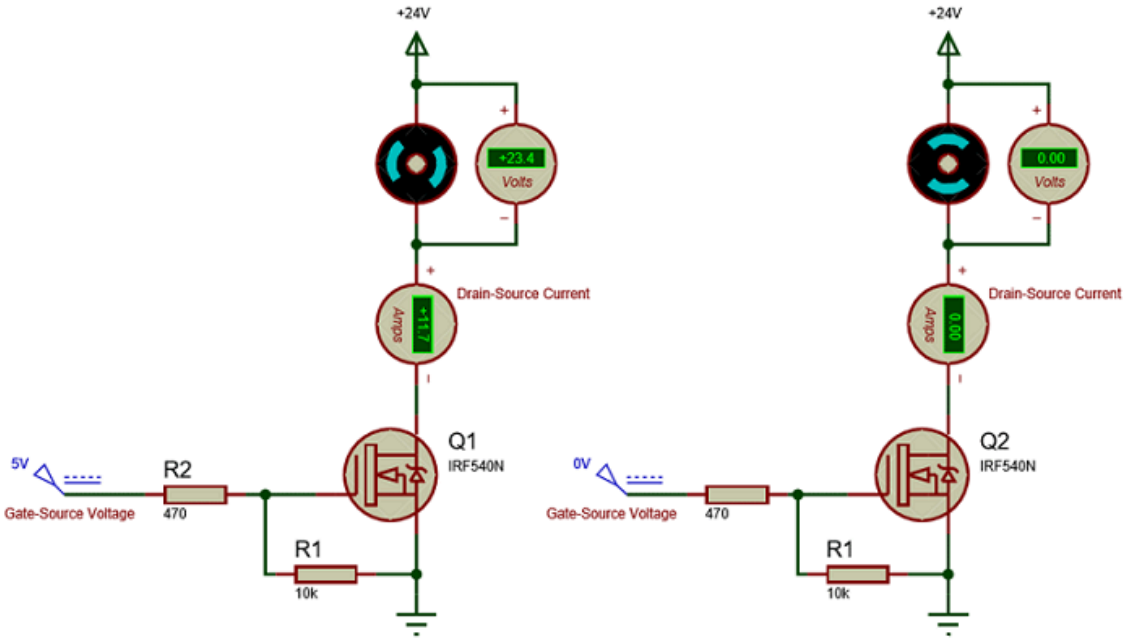
Por exemplo, para acionar um relé de 12 V, pode-se usar um transistor NPN (como o 2N2222), cujo emissor está ligado ao GND, e o coletor ao relé (que por sua vez está conectado à alimentação de 12 V). A base do transistor é conectada ao pino do microcontrolador através de um resistor de polarização (tipicamente 1 kΩ), garantindo o acionamento do relé quando o pino for colocado em

nível alto. Um diodo de roda livre (em inglês, *flyback diode* ou *freewheeling diode*), como o 1N4007, deve ser colocado em paralelo com a bobina do relé, para proteger o transistor contra picos de tensão induzidos pelo campo magnético do relé ao ser desligado.




Fonte: Hackster.io.

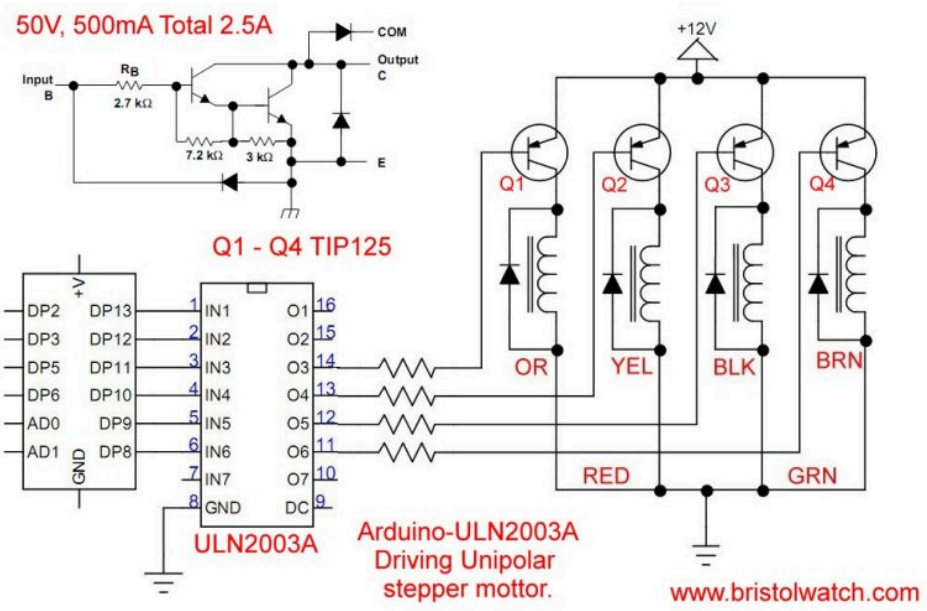
Para aplicações que exigem comutação mais eficiente ou maior velocidade, especialmente em controle de motores ou cargas PWM, os MOSFETs do tipo canal N, como o IRF540N ou o IRLZ44N (este último com lógica compatível a 5 V), são preferíveis. Um circuito típico de acionamento usa o microcontrolador para aplicar tensão na porta do MOSFET, permitindo a condução entre dreno e fonte, e acionando assim a carga conectada.




Fonte: Utmel Electronic.

Em sistemas mais complexos ou com múltiplas cargas, é comum empregar *drivers* de corrente dedicados, como o ULN2003A. O ULN2003A é um circuito integrado para sistemas que necessitam controlar múltiplas cargas de maior corrente, como relés, motores de passo ou *displays*,

que os pinos de um microcontrolador não conseguem acionar diretamente. Este CI abriga sete pares de transistores Darlington, uma configuração que oferece um ganho de corrente extremamente elevado, permitindo que a baixa corrente de saída de um microcontrolador controle correntes de até 500 mA por canal. Uma de suas características mais valiosas são os diodos de proteção internos, ou diodos de roda livre, que cada canal possui. Esses diodos protegem o *driver* e o microcontrolador contra os picos de tensão reversos ("*flyback*") gerados quando cargas indutivas são desativadas, eliminando a necessidade de componentes de proteção externos. Assim, o ULN2003A atua como uma interface robusta e simplificada entre a lógica de baixa potência de um microcontrolador e o mundo das cargas que exigem mais energia, garantindo operação segura e eficiente.



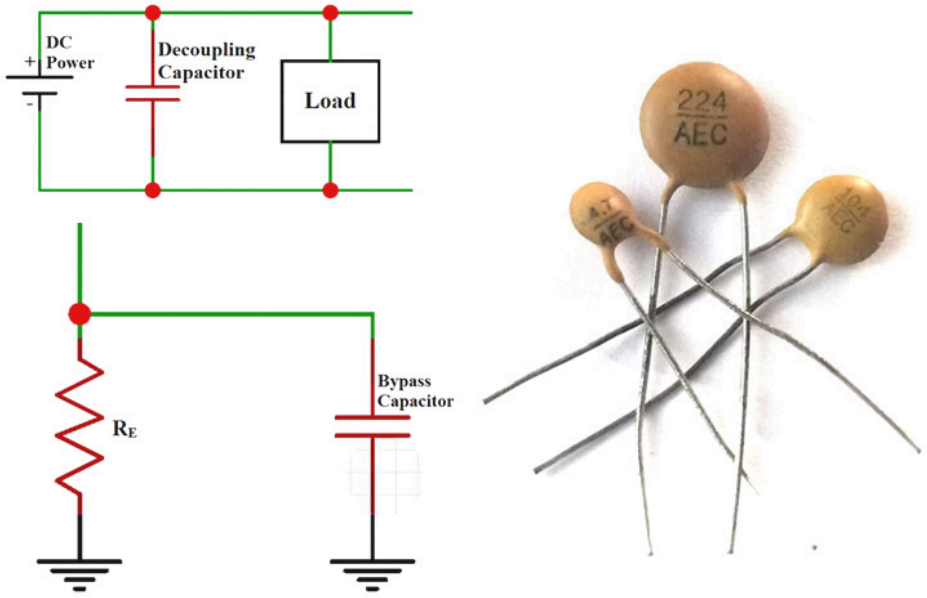

Fonte: Bristolwatch.

## Proteção contra ruídos e transientes

Um aspecto crucial na interface entre microcontroladores e o mundo físico é a proteção contra ruídos elétricos e interferências eletromagnéticas (em inglês, *Eletromagnetic Interference* – EMI), que são especialmente presentes em ambientes industriais, automotivos ou próximos a fontes de comutação de alta corrente, como inversores, motores ou transformadores. Além disso, é importante considerar os transientes decorrentes de chaves mecânicas, que podem gerar picos de tensão e corrente no momento da comutação. Esses distúrbios podem induzir sinais espúrios nas entradas digitais, oscilações indesejadas em leituras analógicas, e em casos mais severos, danificar fisicamente os circuitos do microcontrolador. Para mitigar esses efeitos, emprega-se uma combinação de técnicas e componentes passivos e ativos, organizados em etapas de proteção e filtragem.

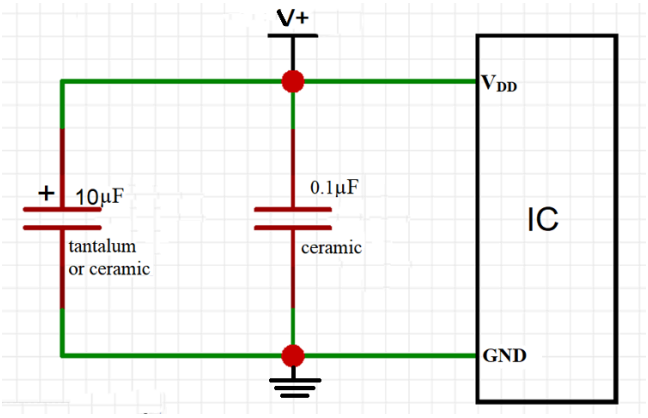
A primeira linha de defesa contra ruídos e flutuações em circuitos eletrônicos é a correta utilização de capacitores na estabilização da alimentação e na filtragem de sinais. É comum que os termos "capacitores de desacoplamento" e "capacitores de *bypass*" sejam usados de forma intercambiável,

mas eles se referem a funções ligeiramente distintas, embora muitas vezes desempenhadas pelo mesmo componente.




Fonte: Components101.

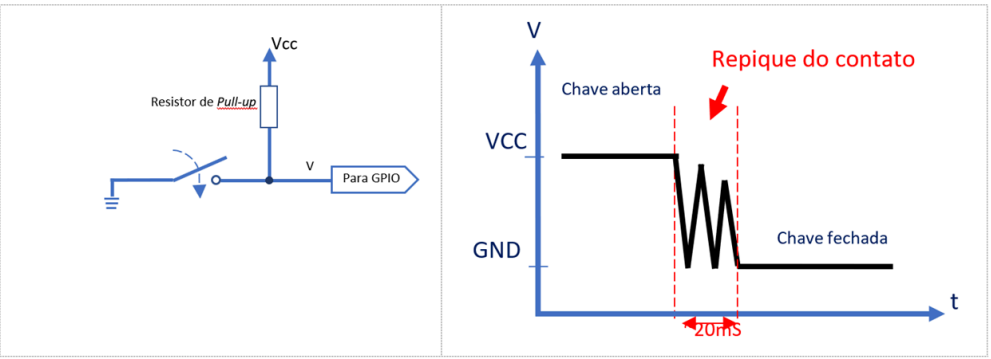
Os **capacitores de desacoplamento** são estrategicamente posicionados o mais próximo possível dos pinos de alimentação (Vcc e GND) de microcontroladores e outros circuitos integrados, geralmente com valores em torno de 100 nF (cerâmicos). Sua principal função é atuar como pequenos reservatórios de energia próximos ao CI. Quando o circuito integrado realiza uma comutação rápida (como um pino digital mudando de estado), há um súbito pico de demanda de corrente. O capacitor de desacoplamento fornece essa energia imediatamente, suavizando as variações rápidas na tensão de alimentação e prevenindo que o ruído gerado por essa comutação se propague para o restante do sistema. Eles "desacoplam" o CI do ruído da linha de alimentação principal. Em fontes de alimentação maiores, capacitores eletrolíticos de maior valor (como 10 µF ou 47 µF) são usados para filtragem de baixa frequência, complementando o trabalho dos capacitores de desacoplamento.




Fonte: Components101.

Já os **capacitores de *bypass*** são utilizados para desviar, ou "bypassar", ruídos de alta frequência para o terra, impedindo que eles atinjam ou saiam de pontos sensíveis do circuito. Eles são aplicados não apenas nas linhas de alimentação (onde a função de desacoplamento é também uma forma de bypassar o ruído da alimentação), mas também nas entradas digitais e sinais de controle expostos a ambientes ruidosos. Nesses casos, formam filtros passa-baixa simples RC, tipicamente com um resistor em série (entre 1 kΩ e 10 kΩ) e um capacitor (entre 10 nF e 100 nF) conectado entre o sinal e o terra. Esse arranjo suaviza variações rápidas e ruídos de alta frequência, garantindo que apenas transições mais lentas e significativas atinjam o pino do microcontrolador. Por exemplo, uma entrada digital conectada a um sensor indutivo em um ambiente ruidoso pode se beneficiar de um resistor de 4,7 kΩ em série com o fio do sensor e um capacitor de 100 nF para o GND, protegendo contra picos e interferências. Embora a função seja similar, a distinção reside no local de aplicação: desacoplamento foca na estabilização da alimentação do CI, enquanto *bypass* é um termo mais amplo para desviar ruídos de qualquer ponto sensível para o terra.

Além da proteção contra ruídos na alimentação, a interface com o mundo físico frequentemente envolve o uso de **chaves mecânicas**, que introduzem um tipo diferente de desafio: o fenômeno do **contato de repique** (em inglês *bounce*). Este fenômeno ocorre porque o fechamento da chave é feito pela colisão entre dois metais. Nessa colisão, durante alguns milissegundos, os metais se separam e se tocam repetidas vezes, gerando diversos eventos de fechamento e abertura da chave.

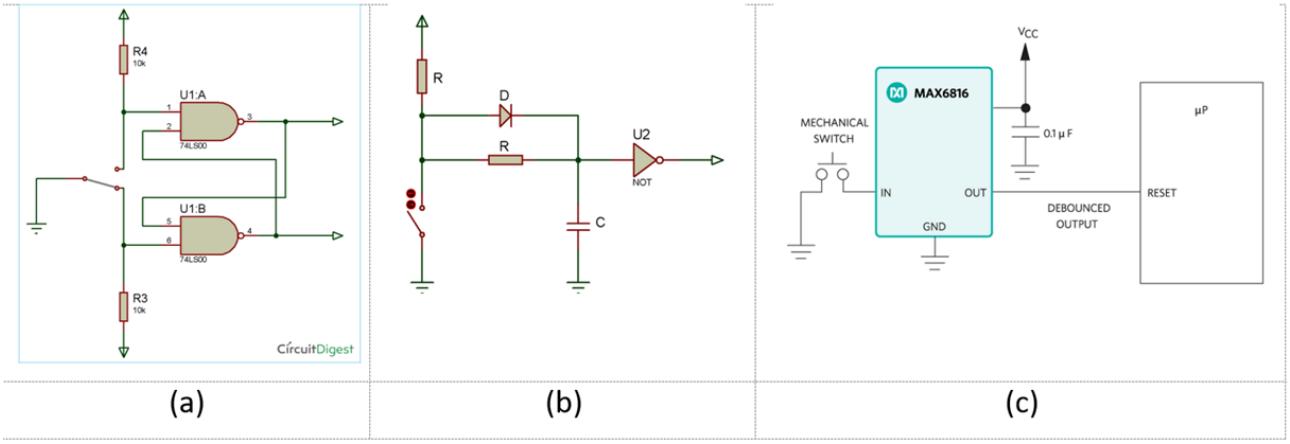



Dependendo da aplicação da chave, a duração deste repique não tem importância. Porém, se esta chave é utilizada para gerar eventos num sistema computacional, a duração deste repique de um único fechamento de circuito pode gerar diversas reações do sistema, como se esta chave estivesse sendo aberta e fechada várias vezes. Este fenômeno também ocorre ao se abrir a chave, havendo o repique também quando se retira o acionamento mecânico. Os **mecanismos de *debouncer*** são projetados especificamente para mitigar esse problema, garantindo que cada acionamento físico da chave seja interpretado como um único evento lógico. Eles podem ser feitos via circuito (*hardware)* ou via programação (*software*). Em *hardware*, há diversas formas de implementação como ilustradas na figura que segue:

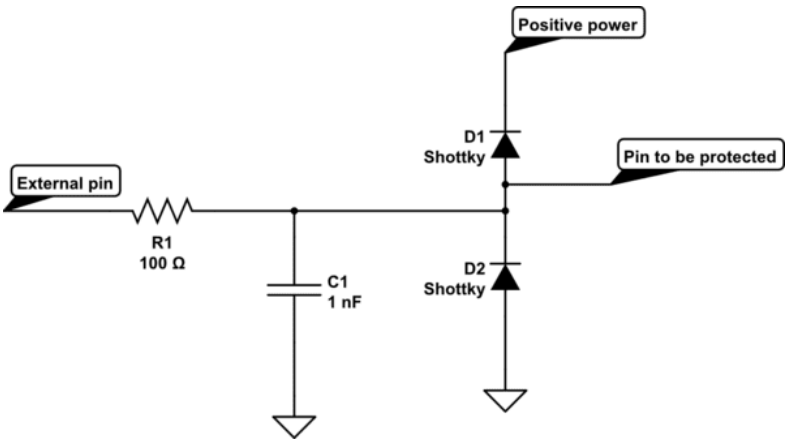
- Flip-flop: Utiliza um circuito biestável (como um SR *flip-flop*) para "travar" o estado da chave no primeiro contato estável, ignorando os repiques subsequentes. Uma vez que o *flip-flop* muda de estado, ele permanece nesse estado até que a chave seja liberada completamente e um novo acionamento estável ocorra.
- Filtro passa-baixa (RC Filter): Emprega uma combinação de resistor (R) e capacitor (C) para criar um atraso na subida ou descida do sinal. O capacitor carrega e descarrega lentamente,

"suavizando" os pulsos rápidos do repique e garantindo que o microcontrolador só detecte uma mudança de estado quando o sinal estiver estável por um período de tempo suficiente.

- *Chips* dedicados, como os da família MAX6816/MAX6817/MAX6818 da *Maxim Integrated*: São circuitos integrados desenvolvidos especificamente para a função de *debouncing*. Eles contêm lógicas internas otimizadas para detectar e filtrar o repique de chaves, fornecendo um sinal de saída limpo e estável para o microcontrolador sem a necessidade de componentes discretos adicionais ou programação complexa.




Para proteger os delicados circuitos internos dos microcontroladores contra picos de tensão que podem facilmente exceder seus limites de operação (normalmente restritos à faixa de −0,3V a VCC+0,3V), são empregados **diodos de proteção**. Embora a maioria dos microcontroladores já incorpore diodos internos de grampeamento (em inglês, *clamp*) entre seus pinos e os trilhos de alimentação (VCC e GND), esses diodos podem ser sobrecarregados e danificados em situações críticas, como grandes transientes de tensão ou descargas eletrostáticas (ESD). É por essa razão que se torna uma prática comum a adição de **diodos Schottky externos de condução rápida**, como o popular 1N5819. Conectados entre a linha de sinal e os trilhos de alimentação (VCC e GND), esses diodos fornecem uma camada extra e robusta de proteção.




Fonte: UltraLibrarian.

A principal vantagem dos diodos Schottky reside em sua baixa queda de tensão direta e em seu tempo de recuperação extremamente rápido quando comparados aos diodos de junção PN convencionais. Essa rapidez é fundamental para a proteção contra picos transitórios, pois permite

que o diodo conduza quase instantaneamente assim que a tensão no pino do microcontrolador tenta exceder VCC (ou cair abaixo de GND), desviando o excesso de corrente para a linha de alimentação ou para o terra. Por exemplo, se um sinal na entrada do microcontrolador subir acima de VCC+0,3V, o diodo Schottky conectado entre o pino e VCC entra em condução, “grampeando” a tensão no pino a um nível seguro, geralmente VCC mais a baixa queda de tensão do Schottky (tipicamente 0,2 V a 0,4 V). De forma similar, se a tensão cair abaixo de GND−0,3V, o diodo Schottky conectado ao GND irá conduzir, desviando o excesso de corrente para o terra. Essa ação rápida e eficaz protege a porta de entrada do microcontrolador contra sobretensões e a energia destrutiva das descargas eletrostáticas, garantindo a longevidade e a confiabilidade do sistema.

Quando os circuitos eletrônicos são submetidos a transientes de alta energia, como aqueles gerados pela comutação de grandes cargas indutivas, descargas atmosféricas (raios) ou interferência industrial pesada, a proteção vai além dos diodos de *clamp* simples. Nesses cenários, empregam-se componentes robustos como os varistores (MOVs), os supressores de tensão transiente (TVS) e os tubos de descarga a gás (GDTs). Cada um oferece uma abordagem única, e frequentemente são usados em conjunto para uma proteção mais abrangente.

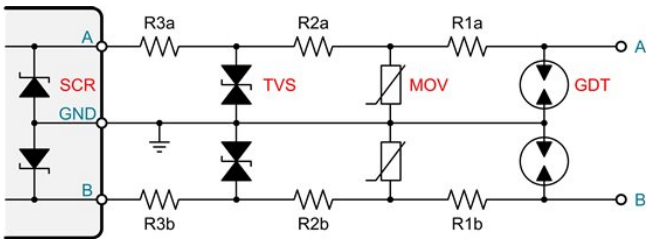
Os **varistores de óxido metálico** (em inglês, *Metal Oxide Varistors* – MOVs), como o MOV 14D471K, são dispositivos baseados em óxido metálico que funcionam como resistores não lineares dependentes de tensão. Em condições normais de operação, sua resistência é extremamente alta. No entanto, quando a tensão em seus terminais excede um valor específico (por exemplo, os 470V nominais do MOV 14D471K), a resistência do MOV cai abruptamente para um valor muito baixo. Isso faz com que ele desvie a grande quantidade de corrente do surto de tensão para o terra, "grampeando" a tensão no circuito a um nível seguro e protegendo os componentes sensíveis. Eles são geralmente conectados em paralelo com a linha de sinal ou alimentação e o terra, e são ideais para proteção de circuitos de potência ou onde surtos de energia significativos são esperados, oferecendo uma boa capacidade de absorção de energia.

Por outro lado, os **diodos TVS** (do inglês *Transient Voltage Suppressor*), como o 1.5KE6.8A, oferecem uma camada de proteção com maior precisão e velocidade de resposta em comparação com os MOVs, sendo particularmente indicados para circuitos mais sensíveis ou de alta velocidade, como interfaces de comunicação serial ou sinais digitais expostos. Um diodo TVS opera de forma similar a um diodo Zener, mas é otimizado para absorver grandes pulsos de energia em tempo recorde (na ordem de picossegundos). Quando a tensão no pino excede o nível de *clamping* do TVS, ele entra rapidamente em avalanche, desviando o excesso de corrente para o terra e mantendo a tensão no circuito dentro de limites seguros, protegendo contra sobretensões e ESD.

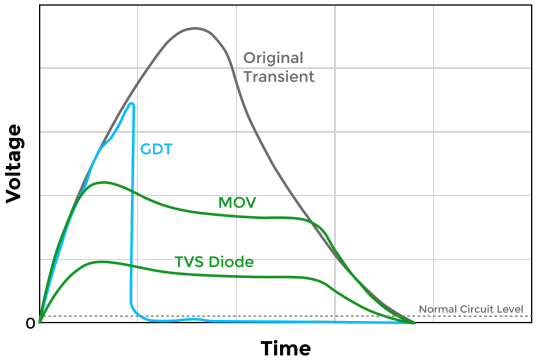
Complementando essa gama de proteções, temos os **tubos de descarga a gás** (em inglês, *Gas Discharge Tube* – GDTs). Um GDT consiste em eletrodos separados por um gás inerte, que se mantém isolante em condições normais. Contudo, sob um surto de tensão que excede sua tensão de disparo, o gás é ionizado, criando um arco voltaico de baixíssima resistência entre os eletrodos. Essa ignição permite que o GDT desvie correntes extremamente elevadas (na ordem de quiloampères) para o terra, protegendo o circuito de danos catastróficos causados por raios ou surtos de potência massivos. Embora sejam mais lentos para disparar que os MOVs e TVS, e possam apresentar uma “corrente de seguimento” após o surto, sua altíssima capacidade de energia e baixíssima capacitância (o que os torna ideais para linhas de comunicação de alta frequência)

fazem deles a primeira linha de defesa em muitas aplicações robustas, frequentemente seguidos por MOVs e TVS em uma estratégia de proteção em cascata.

Por exemplo, em uma interface RS-485 utilizada em ambientes industriais, onde os sinais diferenciais A e B podem estar sujeitos a surtos de até 1 kV, uma estratégia de proteção em cascata é aplicada para garantir a integridade dos sinais diferenciais A e B e a sobrevivência do *transceiver*. A primeira linha de defesa contra os surtos mais severos, especialmente os de alta energia como os causados por raios (que podem atingir picos de até 1 kV), é tipicamente fornecida por GDTs. Eles são posicionados na entrada da linha, aterrando o excesso de corrente assim que a tensão atinge seu ponto de disparo. Sua capacidade de desviar correntes massivas para o terra os torna ideais para absorver o impacto inicial de surtos destrutivos. Uma segunda camada de proteção pode ser implementada com MOVs. Esses componentes complementam os GDTs absorvendo a energia residual de surtos menos intensos, mas ainda prejudiciais, ou os picos que o GDT pode ter deixado passar devido ao seu tempo de resposta relativamente mais lento. Os MOVs, conectados entre as linhas A e B e o terra, reduzem a tensão para um nível mais gerenciável. Finalmente, para a proteção fina e rápida do *transceiver* da RS-485, são utilizados TVS bidirecionais. Posicionados o mais próximo possível dos pinos de entrada do *transceiver*, os diodos TVS reagem em nanossegundos, "grampeando" qualquer transiente remanescente para dentro dos limites seguros de operação do componente. Como ilustra o gráfico que segue, essa ação rápida protege contra transientes muito velozes, como as descargas eletrostáticas (ESD), que poderiam danificar o *transceiver* antes que os GDTs ou MOVs pudessem disparar completamente.




Fonte: Texas Instruments.




Fonte: M.C.C.

Além da proteção elétrica direta, deve-se considerar o isolamento galvânico em sistemas que exigem alta imunidade ou segurança, utilizando isoladores digitais, que separam fisicamente o microcontrolador da fonte de ruído ou tensão elevada, como em sistemas de controle de motores AC, medição de rede elétrica ou controle de aquecedores industriais.

## Isolamento de sinais

Além das questões relacionadas à tensão e corrente, um fator determinante em muitos projetos é assegurar o isolamento elétrico entre o microcontrolador e outras partes do sistema. Esse cuidado se torna ainda mais essencial quando há interação com circuitos de potência ou fontes externas de energia, que podem representar riscos significativos de choque elétrico ou provocar danos irreversíveis a componentes sensíveis. Para mitigar esses riscos, empregam-se isoladores

galvânicos, dispositivos projetados para bloquear a passagem direta de corrente elétrica, permitindo apenas a transmissão segura de sinais. Eles oferecem alto grau de isolamento e excelente imunidade a ruídos de modo comum, características fundamentais em ambientes com interferências eletromagnéticas severas.

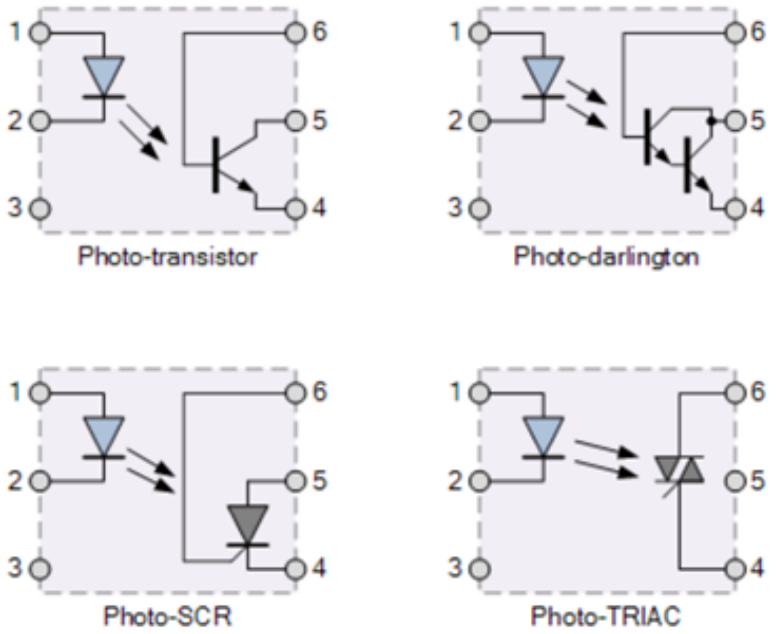
Esse tipo de isolamento é indispensável em diversos contextos. Em sistemas de controle industrial, por exemplo, ele protege os circuitos lógicos diante da proximidade com maquinário pesado e ruidoso. Na automação residencial, assegura a integridade de dispositivos conectados à rede elétrica, promovendo maior segurança ao usuário. Já em aplicações que envolvem redes de comunicação sujeitas a interferências elétricas ou variações abruptas de tensão, ele preserva a integridade da troca de dados. Com isso, o isolamento galvânico atua como uma barreira confiável entre ambientes de alta e baixa tensão, garantindo que o circuito de controle permaneça estável, protegido e funcional, mesmo sob condições adversas do lado de potência.

Um **optoacoplador** funciona utilizando luz para transmitir informações de um bloco a outro do circuito, sem conexão elétrica direta. Dessa forma, ele permite que componentes mais sensíveis — como microcontroladores — possam controlar ou se comunicar com circuitos submetidos a altas tensões ou sujeitos a transientes elétricos, minimizando o risco de danos. Esses dispositivos, também chamados de **acopladores ópticos** ou **opto-isoladores**, são amplamente utilizados para acionar componentes como transistores de potência e TRIACs diretamente a partir de saídas digitais (GPIO). Sua principal vantagem é oferecer um alto grau de isolamento elétrico entre os terminais de entrada e saída, o que possibilita o controle de cargas em corrente alternada (CA) de alta potência com sinais digitais de baixa tensão. Em aplicações de baixa potência, especialmente utilizando foto-transistores, os optoacopladores também permitem a leitura segura de sensores externos instalados em ambientes eletricamente hostis pelas entradas digitais do microcontrolador.

Internamente, a estrutura básica de um acoplador óptico consiste em um LED emissor de luz infravermelha e um dispositivo fotossensível semicondutor, que detecta o feixe de luz emitido. Ambos os componentes são montados em um encapsulamento opaco à luz externa, o que garante que apenas o sinal óptico interno atue no processo de transmissão, como ilustrado na figura.

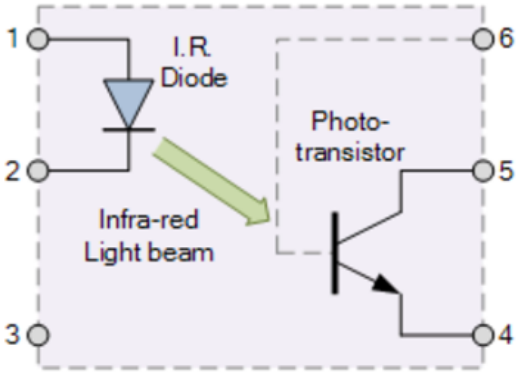

Entre os dispositivos fotossensíveis, destacam-se o foto-resistor, o foto-diodo, o foto-SCR, o foto-TRIAC e o foto-transistor. O foto-resistor, também conhecido como célula fotoelétrica ou LDR (do inglês *Light Dependent Resistor*), é um resistor cuja resistência varia conforme a intensidade da luz incidente. Quanto ao foto-transistor, ele é um transistor que responde à luz, modificando a corrente de base quando a luz incide sobre a junção p-n, o que influencia a corrente entre o coletor e o emissor. Por sua vez, o foto-diodo é um tipo de diodo semicondutor sensível à luz, onde a corrente elétrica gerada na junção p-n é proporcional à intensidade luminosa; ele é ligado em polarização reversa e passa a conduzir quando a luz incide sobre ele. Embora possa ser usado de forma similar

ao foto-transistor, o foto-diodo tem sensibilidade menor (já que o foto-transistor amplifica a corrente gerada na junção pela luz), mas, por outro lado, apresenta uma resposta mais rápida, pois possui apenas uma junção P-N em vez de duas. Complementando essa família, o foto-SCR utiliza a luz para disparar o SCR (do inglês *Silicon Controlled Rectifier*), permitindo o controle da condução de corrente. Da mesma forma, o foto-TRIAC emprega a luz para controlar o fluxo de corrente de potência, onde o terminal de disparo de um TRIAC (do inglês *TRIode for Alternating Current*) convencional é sensível à luz.




Fonte: eletronic-tutorials.

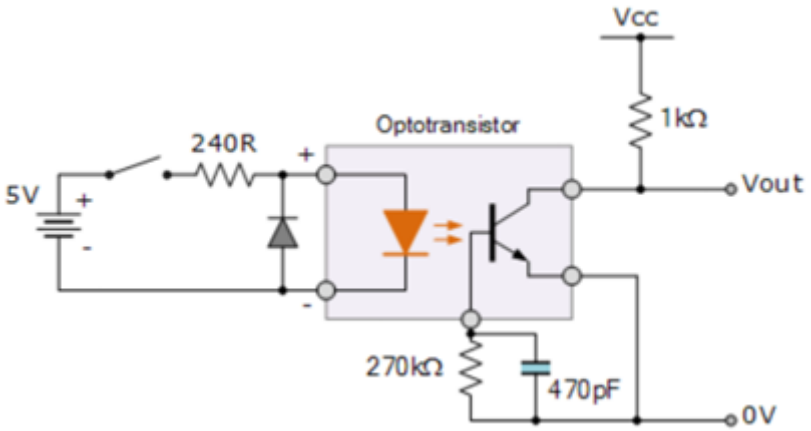
O foto-transistor é o componente central ou a base do princípio de operação de um optoacoplador típico. No foto-transistor, a corrente proveniente de uma fonte de sinal percorre um LED emissor de luz infravermelha. Essa luz, por sua vez, incide sobre a base do foto-transistor, que então passa a conduzir, comportando-se como um transistor bipolar ativado por luz. A base do foto-transistor pode ser deixada desconectada para garantir máxima sensibilidade à luz infravermelha ou, alternativamente, ser ligada ao terra por meio de um resistor externo; esta última opção ajuda a controlar a sensibilidade e a evitar disparos falsos causados por ruído elétrico ou transientes de tensão. Quando a corrente no LED é interrompida, a emissão de luz cessa, fazendo com que o foto-transistor pare de conduzir. Esse mecanismo permite o controle eficaz do estado lógico da saída, com isolamento elétrico total entre os circuitos de entrada e saída, isolamento esse que pode chegar a até 10kV, já que os componentes ópticos são separados por um meio transparente — como vidro, plástico ou ar — sem contato elétrico direto.




Fonte: eletronic-tutorials.

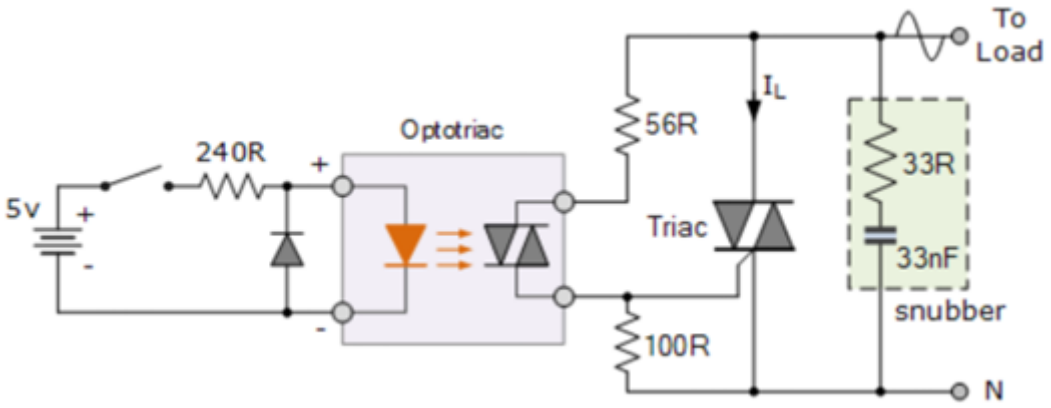
Os dispositivos foto-diodo, foto-transistor e foto-darlington são predominantemente utilizados em circuitos de corrente contínua (CC), enquanto o foto-SCR e o foto-TRIAC são empregados para o controle de circuitos alimentados por corrente alternada (CA). Segue-se o detalhamento do princípio de operação do foto-transistor.

Uma aplicação dos acopladores ópticos é na detecção do acionamento de uma chave ou outro tipo de sinal de entrada digital, mostrada na seguinte figura. Isso é útil quando a chave ou o sinal que está sendo detectado estiver dentro de um ambiente eletricamente ruidoso. A saída pode ser usada para operar um circuito externo ou como entrada digital para um microcontrolador. Na figura, o resistor de 270kΩ conectado externamente é usado para controlar a sensibilidade da região de base do foto-transistor. O valor do resistor pode ser escolhido para se adequar ao dispositivo foto-acoplador selecionado e à sensibilidade necessária. O capacitor impede que quaisquer picos ou transientes indesejados produzam falsos acionamentos na base dos opto-transistores.




Fonte: eletronic-tutorials.

Além de aplicações em corrente contínua, existem optoacopladores com TRIACs integrados, ideais para controle de cargas em corrente alternada. Um exemplo é o MOC3020, um opto-TRIAC capaz de suportar até 400V e 100mA, adequado para ligação direta à rede elétrica. Para controle de cargas maiores, o opto-TRIAC pode fornecer o pulso de gate necessário a um TRIAC externo, usando um resistor limitador de corrente, como mostrado na seguinte figura.

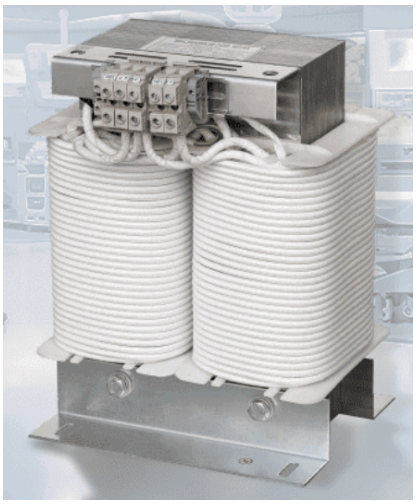



Fonte: eletronic-tutorials.

Esse tipo de configuração é a base de **relés de estado sólido** (em inglês *Solid-State Relays* – SSR), capazes de acionar lâmpadas, motores e outros dispositivos CA com elevada eficiência e confiabilidade. Diferentemente do SCR, o TRIAC pode conduzir durante ambas as metades do ciclo CA. Com o uso de detecção de cruzamento por zero, ou detector de passagem por zero, é possível acionar a carga com mínimo ruído elétrico, evitando picos de corrente durante a comutação de cargas indutivas. Um exemplo de *chip* que incorpora a detecção de zero em um optoacoplador é a série MOC304x (MOC3041, MOC3042, MOC3043) ou MOC306x (MOC3061, MOC3062, MOC3063).

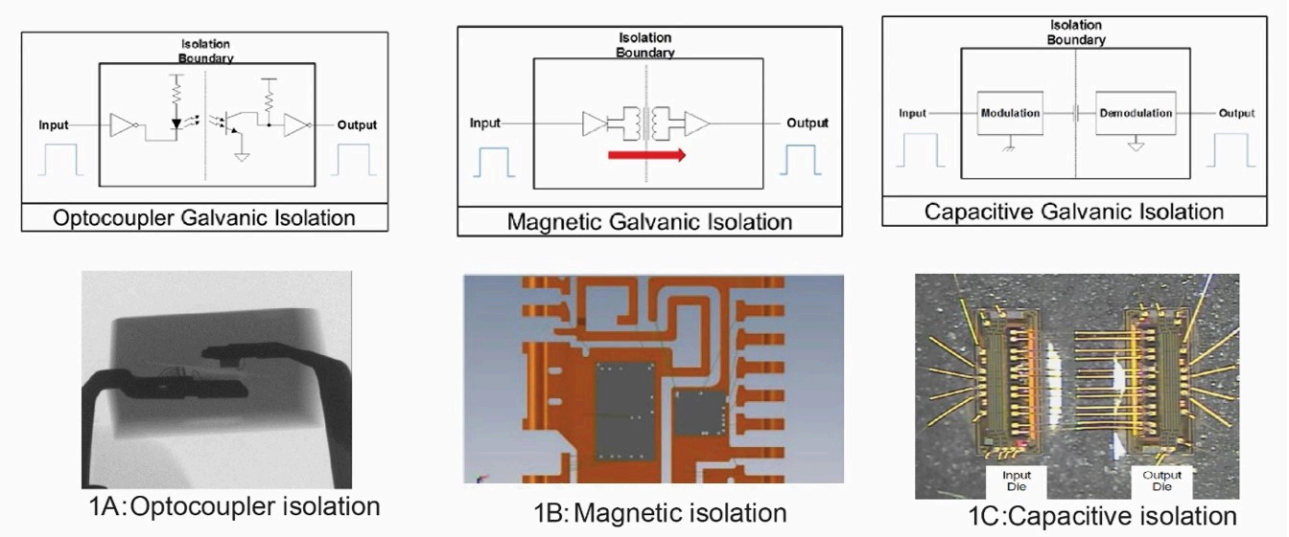
Embora os optoacopladores sejam uma solução robusta e amplamente utilizada para o isolamento de sinais elétricos, eles não são a única tecnologia disponível. Existem outros circuitos e abordagens que oferecem isolamento galvânico, cada um com vantagens específicas que os tornam mais adequados para determinadas aplicações, considerando fatores como velocidade, robustez, imunidade a ruído e custo.

Uma das formas mais tradicionais de isolamento é através de **transformadores de isolamento**. O princípio se baseia na transferência de energia ou sinal por indução magnética entre enrolamentos eletricamente separados. Eles são amplamente utilizados para isolar sinais analógicos, pulsos digitais e até mesmo em fontes de alimentação isoladas (DC-DC). Dentre seus tipos, destacam-se os transformadores de pulso, comuns em redes Ethernet e RS-485, e as fontes chaveadas; os transformadores para áudio, essenciais para isolamento de sinais analógicos em equipamentos de som de alta fidelidade; e os transformadores de ferrite, que garantem o isolamento galvânico entre a entrada e a saída em fontes de alimentação. Suas principais vantagens incluem a alta capacidade de isolamento, a ausência de contato elétrico direto e a capacidade de transmitir potência, embora sejam ineficientes em baixas frequências e não consigam transmitir corrente contínua (DC).



Fonte: Aktif.

Outra tecnologia relevante são os **isoladores capacitivos**, que utilizam o princípio do acoplamento capacitivo. Nesses dispositivos, os sinais são transmitidos por meio de variações de tensão entre placas separadas por um material dielétrico. Essa abordagem é particularmente eficaz para o isolamento de sinais digitais de alta velocidade, com exemplos notáveis em famílias de isoladores digitais da Silicon Labs (como a série Si86xx) e da Texas Instruments (série ISO7xxx). Suas vantagens residem na alta velocidade de operação (podendo chegar a gigahertz), no tamanho compacto e na baixa dissipação de energia. Contudo, em alguns contextos industriais, sua imunidade a ruído pode ser inferior à dos optoacopladores. Enquanto os optoacopladores são uma escolha confiável e versátil para isolamento geral de CA/CC, os isoladores por transformador são a melhor aposta para ambientes que demandam máxima robustez e imunidade a ruídos severos. Já os isoladores capacitivos sobressaem em aplicações que exigem alta velocidade e compactação. A escolha ideal dependerá das exigências específicas de velocidade, tipo de sinal (CA/CC), nível de isolamento e condições de ruído do seu projeto.




Fonte: MPS.

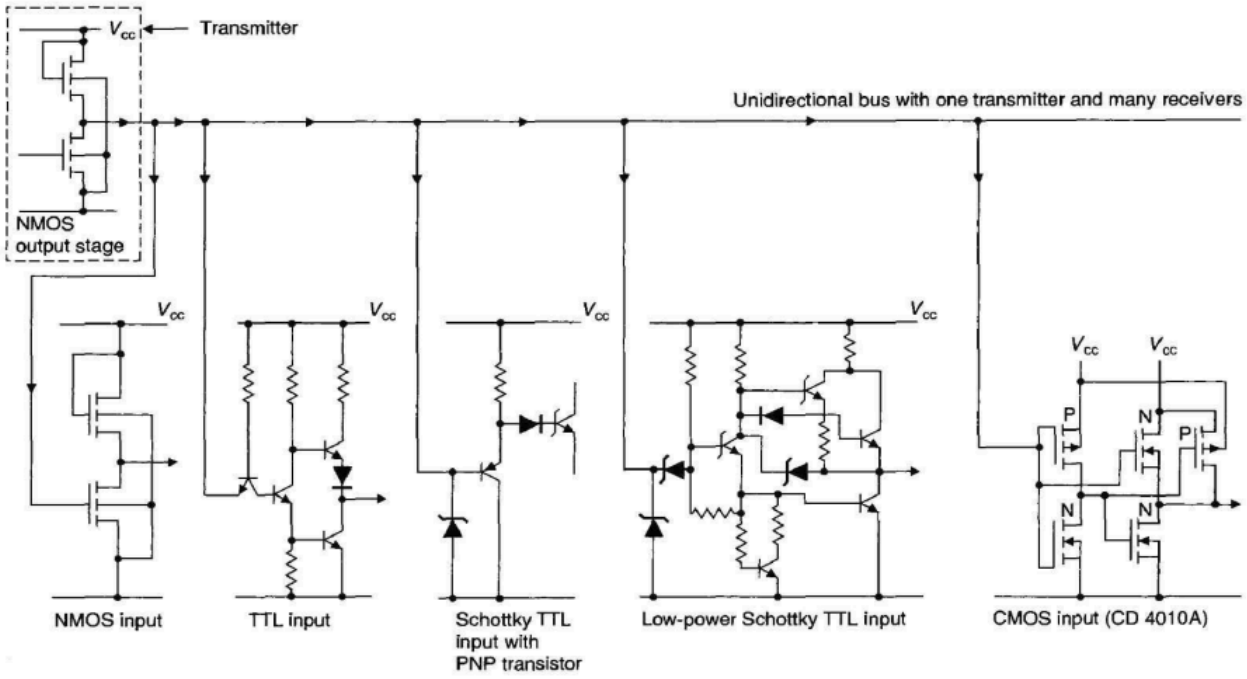
Por fim, os isoladores com acoplamento magnético representam uma solução moderna e altamente eficiente. Apesar de também se basearem no magnetismo, como os isoladores por transformadores,

a transmissão do sinal acontece por meio de campos magnéticos gerados internamente a um *chip*. Isso elimina a necessidade de componentes magnéticos discretos e volumosos, como os transformadores externos usados nos isoladores mais tradicionais. Empresas como a Analog Devices, com suas séries ADuM, são exemplos de fabricantes desses *chips* avançados. As aplicações desses isoladores são vastas, indo desde o isolamento de portas USB e interfaces SPI até o controle de *gates* em componentes de alta tensão como IGBTs e MOSFETs. Suas principais vantagens incluem alta velocidade, baixo consumo de energia, robustez e uma excelente imunidade a ruídos de modo comum (em inglês *Common Mode Rejection* – CMR). Essas características os tornam ideais para ambientes exigentes, como interfaces industriais e médicas, onde a confiabilidade é absolutamente crítica.

É importante ressaltar uma característica comum entre os isoladores capacitivos e os isoladores por transformador (incluindo os de acoplamento magnético): eles não conseguem transmitir corrente contínua (DC) diretamente. Se for necessário isolar um sinal DC usando um desses métodos, ele precisará ser modulado ou codificado em um sinal CA pulsado antes de atravessar a barreira de isolamento.

## Controle de acesso a barramentos

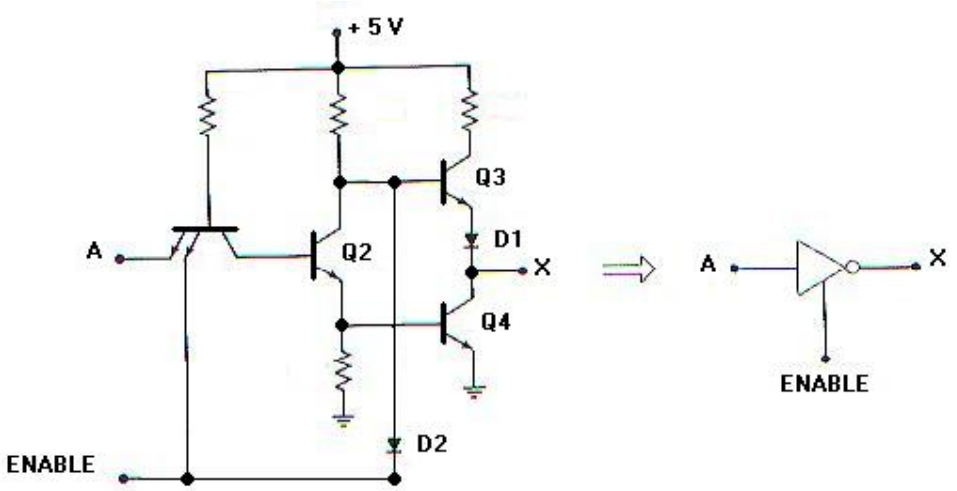
Nos sistemas embarcados modernos, especialmente aqueles baseados em microcontroladores, é comum usar barramentos compartilhados para otimizar a comunicação entre diversos componentes, como memórias, sensores, atuadores e periféricos. Barramentos como I2C, SPI e os tradicionais barramentos paralelos utilizam linhas comuns para dados e endereços, o que exige um cuidado especial para evitar conflitos de sinal e garantir o funcionamento correto do sistema. É nesse ponto que os circuitos de interface para barramentos se tornam essenciais. Eles permitem que múltiplos dispositivos compartilhem fisicamente as mesmas linhas sem que haja interferência elétrica. A importância desses circuitos é tão grande que muitas funções de controle de barramento já vêm integradas diretamente nos pinos de entrada/saída dos microcontroladores mais modernos. Mesmo assim, o uso de circuitos de interface externos ainda é muito relevante em aplicações mais complexas envolvendo a interconexão de componentes de diferentes tecnologias, como ilustra a seguinte figura, ou em situações que exigem maior robustez e flexibilidade no projeto.




| | *Logic Family* | | | | | |
|---|---|---|---|---|---|---|
| **Characteristic** | **LS TTL** | **S TTL** | **ALS TTL** | **NMOS** | **CMOS** | **Units** |
| $V_{OL}$ | 0.5 | 0.5 | 0.4 | 0.4 | 0.01 | V |
| $V_{OH}$ | 2.7 | 2.7 | 2.7 | 2.4 | 4.99 | V |
| $V_{IL}$ | 0.8 | 0.8 | 0.8 | 0.8 | 1.5 | V |
| $V_{IH}$ | 2.0 | 2.0 | 2.0 | 2.0 | 3.5 | V |
| $I_{OL}$ | 8 | 20 | 4 | 1.6 | 0.4 | mA |
| $I_{OH}$ | −400 | −1000 | −400 | −200 | −500 | µA |
| $I_{IL}$ | −0.4 | −2.0 | −0.4 | 2.5 µA | 10 pA | mA |
| $I_{IH}$ | 20 µA | 50 µA | 20 µA | 2.5 µA | 10 pA | mA |
| Propagation delay | 9.5 | 3 | 4 | 2.5 | 3.5 | ns |
| Input capacitance | 3.5 | — | — | 10–160 | 5 | pF |

Fonte: Clements.

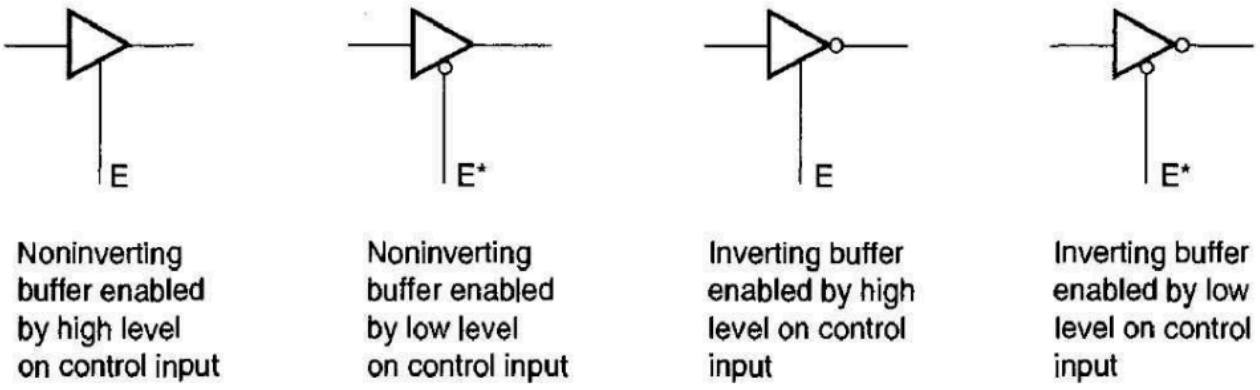
Os ***buffers* tri-state** são componentes fundamentais na construção de barramentos compartilhados. Eles possuem três estados de saída: alto (1 lógico); baixo (0 lógico); e alta impedância (Z). Um circuito típico de uma porta de três-estados com 4 transistores, usando uma saída *push-pull* e operando de forma a ter três estados possíveis na sua saída, é apresentado na seguinte figura.




Fonte: endigital.

A saída *push-pull* de uma porta *tri-state* é formada por dois transistores em série, conectados entre a alimentação e o terra (geralmente um PMOS/PNP e outro NMOS/NPN). São eles os responsáveis por estabelecer o nível lógico HIGH (alto) ou LOW (baixo) na saída. O comportamento *tri-state*, que permite um terceiro estado de alta impedância, é controlado por outros dois transistores que atuam como chaves de habilitação. Esses transistores determinam se a saída *push-pull* está conectada ou isolada do circuito.

Quando o sinal de controle de habilitação (enable) está ativo, ele liga esses transistores de habilitação. Nesse momento, a lógica de entrada da porta pode comandar a saída *push-pull* para definir um estado HIGH ou LOW. Por outro lado, quando o sinal de habilitação desliga esses dois transistores que controlam o *push-pull*, isso efetivamente desconecta a saída da alimentação e do terra, fazendo com que ela se comporte como um circuito aberto. Nesse estado de alta impedância (HIGH-Z), a porta não drena nem fornece corrente, permitindo que outros dispositivos no mesmo barramento de dados controlem o nível lógico da linha sem interferência. Essa capacidade de "flutuar" a saída é essencial para a construção de barramentos, onde múltiplos dispositivos precisam compartilhar as mesmas linhas de comunicação sem criar conflitos quando não estão transmitindo dados.Seguem-se algumas variantes de uma porta tri-state.



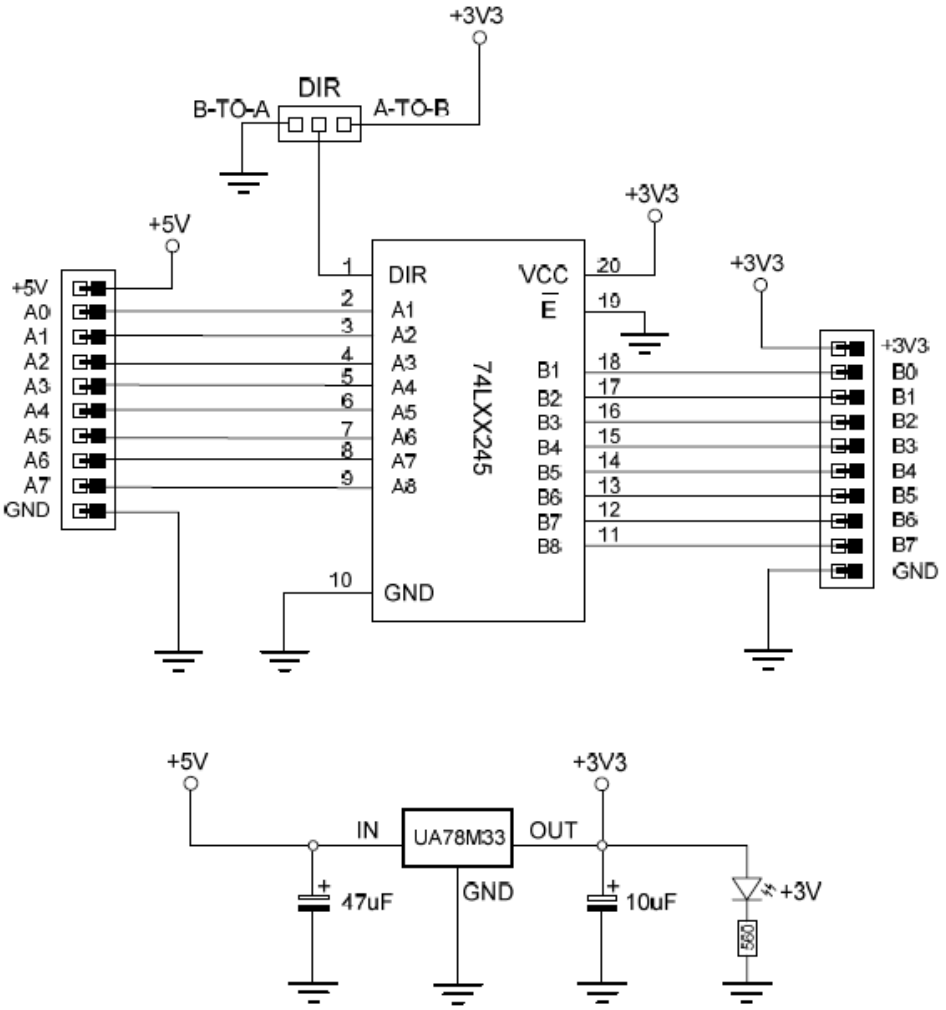

Fonte: Clements.

Um exemplo clássico é o *chip* 74HC244, frequentemente utilizado para isolar barramentos paralelos, permitindo ao microcontrolador habilitar ou desabilitar a sua influência nas linhas de dados através de um sinal de controle (geralmente chamado OE, *Output Enable*). Esses *buffers* são especialmente úteis em sistemas com memórias externas paralelas, onde o microcontrolador precisa liberar o barramento durante operações de leitura, ou em sistemas com vários dispositivos conectados ao mesmo barramento, onde cada um é ativado individualmente.

Os **transceptores** (em inglês ***transceivers***) são componentes fundamentais em sistemas digitais, pois permitem a comunicação bidirecional sobre um mesmo conjunto de fios, controlando ativamente a direção e a habilitação da comunicação através do barramento. Isso é crucial para otimizar o uso de pinos em microcontroladores e simplificar o cabeamento. Um exemplo clássico e amplamente utilizado em sistemas embarcados é o 74HC245. Este *chip* é um *transceiver* de barramento bidirecional de 8 *bits* que permite que os mesmos oito fios sejam usados tanto para entrada quanto para saída de dados. O sentido do fluxo de dados é definido pelo pino DIR (do inglês *Direction*). Quando está em um estado lógico (por exemplo, HIGH), os dados fluem de um lado do *chip* para o outro (digamos, de A para B). Quando está no estado oposto (LOW), os dados fluem na direção contrária (de B para A). O pino de controle OE (do inglês *Output Enable*), por sua vez, habilita ou desabilita a saída do *transceiver.* Quando ativo (geralmente LOW), o *transceiver* opera no modo bidirecional definido por DIR. Quando inativo (HIGH), a saída é colocada em alta impedância (HIGH-Z), desconectando efetivamente o *transceiver* do barramento e permitindo que outros dispositivos utilizem as linhas.

Os *transceivers* como o 74HC245 são essenciais em cenários onde um microcontrolador precisa alternar dinamicamente entre leitura e escrita em um periférico paralelo (como uma memória SRAM paralela), ou quando a comunicação é bidirecional sobre um único conjunto de fios, economizando *hardware* e complexidade. Além de sua função de controle de direção e habilitação, muitos *transceivers*, incluindo variações do 74HC245 ou *chips* dedicados, como vimos antes, possuem uma característica fundamental em circuitos baseados em microcontroladores: a adaptação de níveis de tensão (em inglês *level shifting*).

Em sistemas embarcados modernos, é comum encontrar componentes que operam com diferentes tensões lógicas. Por exemplo, um microcontrolador pode trabalhar a 3.3V, enquanto um periférico (um sensor ou um módulo mais antigo) pode exigir ou fornecer sinais a 5V. Vimos que conectar diretamente pinos que operam em tensões diferentes pode causar danos aos componentes ou leituras errôneas de sinais. *Transceivers* com capacidade de adaptação de níveis resolvem esse problema. Eles são projetados para ter diferentes tensões de alimentação em seus lados A e B. O lado conectado ao microcontrolador pode ser alimentado a 3.3V (VA), enquanto o lado conectado ao periférico é alimentado a 5V (VB). O *transceiver* assume a tarefa de converter automaticamente os níveis lógicos entre essas duas tensões. Por exemplo, um sinal HIGH de 3.3V vindo do microcontrolador é interpretado e retransmitido como um HIGH de 5V para o periférico, e o processo inverso ocorre quando o periférico envia um sinal de volta ao microcontrolador. Isso é ilustrado no esquemático abaixo. O circuito de alimentação será detalhado posteriormente. Assim, o 74HC245 (e seus irmãos *transceivers* com *level shifting*) não é apenas um controlador de fluxo de dados, mas um verdadeiro circuito de interface que garante a compatibilidade elétrica e a comunicação eficaz em sistemas com diferentes domínios de tensão.

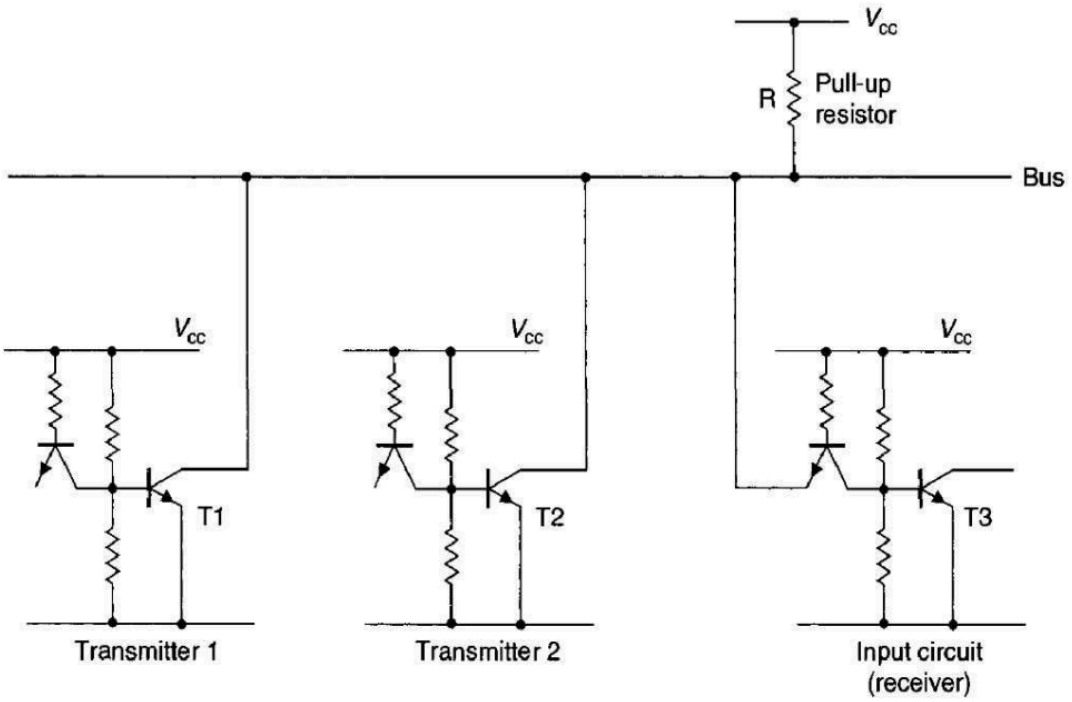






Fonte: Stackexchange.

As **saídas de coletor aberto** (em circuitos bipolares) ou dreno aberto (em circuitos MOSFET) são largamente utilizadas em circuitos de interface, especialmente em barramentos compartilhados. Nessa configuração, a saída do circuito é ligada apenas ao terminal coletor ou dreno de um transistor interno, sem conexão direta ao nível lógico alto. O funcionamento baseia-se no seguinte princípio. Quando o transistor está ligado (saturado), ele conecta a linha diretamente ao terra (GND), forçando o nível lógico baixo (0). E quando o transistor está desligado (em corte), a linha fica desconectada internamente, entrando em estado de alta impedância (flutuante). Para garantir o nível lógico alto (1), é necessário o uso de um resistor de *pull-up* externo, que puxa a linha passivamente para a tensão de alimentação Vcc, como mostra a figura que segue. Assim, o nível lógico 1 depende unicamente do resistor externo, enquanto o nível lógico 0 é obtido diretamente pelo acionamento do transistor. Essa abordagem facilita a integração de dispositivos com diferentes tensões lógicas, permite expansão simples do sistema e garante funcionamento seguro em ambientes com múltiplos transmissores, especialmente em arquiteturas multi-mestre.



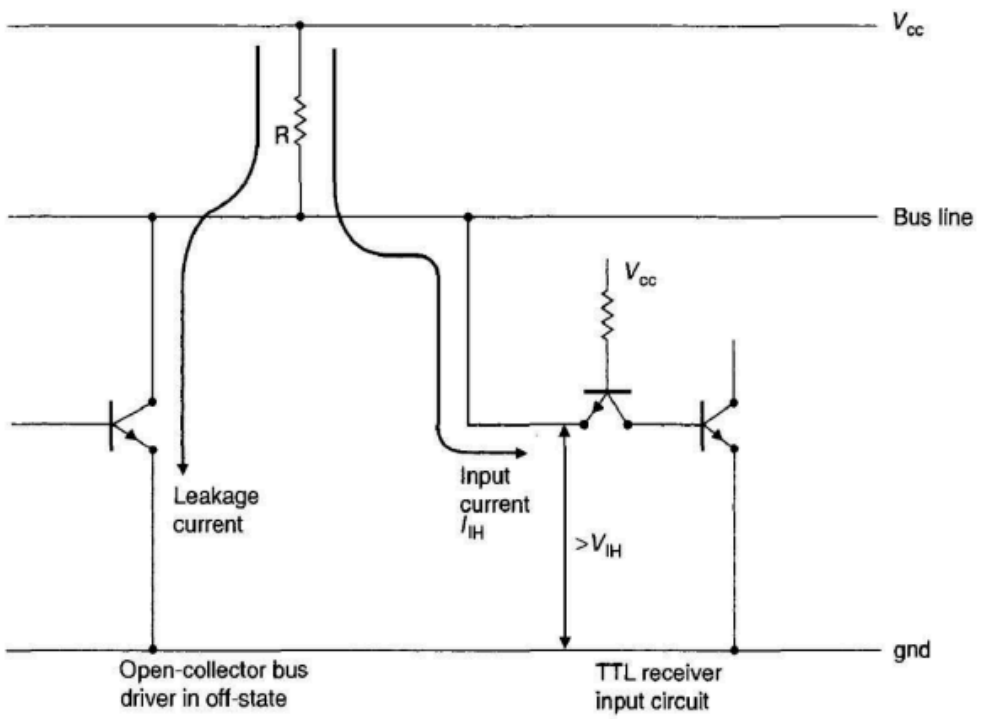

Fonte: Clements.

Esse tipo de saída pode ser entendido como uma versão simplificada de uma saída do tipo *push-pull*, onde o transistor de subida é removido, restando apenas o transistor que conecta ao terra. Isso tem como consequência a dependência do circuito externo para definir o nível lógico alto. Por outro lado, ganhamos a vantagem de permitir que múltiplos dispositivos compartilhem a mesma linha física sem gerar conflitos. Qualquer dispositivo pode forçar o nível lógico baixo ao saturar seu transistor, enquanto o nível lógico alto é obtido de forma passiva via resistor, evitando conflitos elétricos. Esse comportamento é frequentemente descrito como lógica AND por cabeamento (em inglês, *wired-AND logic*), pois a linha fica em nível 0 sempre que qualquer dispositivo ativa sua saída, e apenas retorna a 1 quando nenhum dispositivo força o nível baixo.

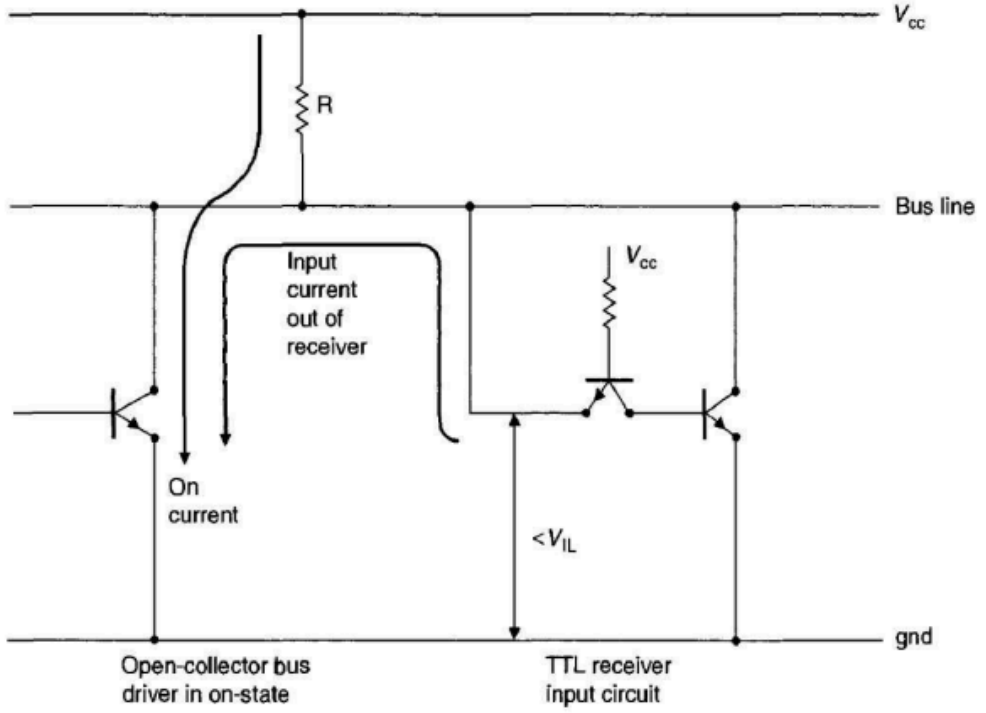
O correto dimensionamento do resistor de pull-up é crucial para o funcionamento seguro e confiável do barramento, considerando dois aspectos principais:

- Quando a linha está em nível alto, o resistor não deve ser muito grande, pois correntes parasitas e de entrada podem causar quedas de tensão indesejadas. O valor máximo do resistor, $R_{max}$, é calculado a partir da soma das correntes de entrada em nível alto ($I_{IH}$) de todos os receptores, das correntes de fuga ($I_{fuga}$) dos transmissores, e da tensão mínima garantida para o nível lógico alto ($V_{OH}$): $R_{max} = (V_{cc} - V_{OH})/(\sum I_{fuga} + \sum I_{IH})$.
- Quando a linha está em nível baixo, o resistor não deve ser muito pequeno, pois isso aumentaria a corrente drenada pelo transistor que força o nível baixo, podendo exceder o limite de corrente de saída ($I_{OL}$). O valor mínimo do resistor, $R_{min}$, depende da corrente máxima que o transistor pode drenar e das correntes de entrada em nível baixo ($I_{IL}$) dos receptores: $R_{min} = (V_{cc} - V_{OL})/(I_{OL} + \sum I_{IL})$.

O intervalo seguro para escolha do resistor é, portanto, entre $R_{min}$ e $R_{max}$, garantindo integridade dos níveis lógicos sem sobrecarregar os dispositivos.




(a) Bus pulled up to $V_{cc}$ passively




(b) Bus pulled down to ground actively

Fonte: Clements.

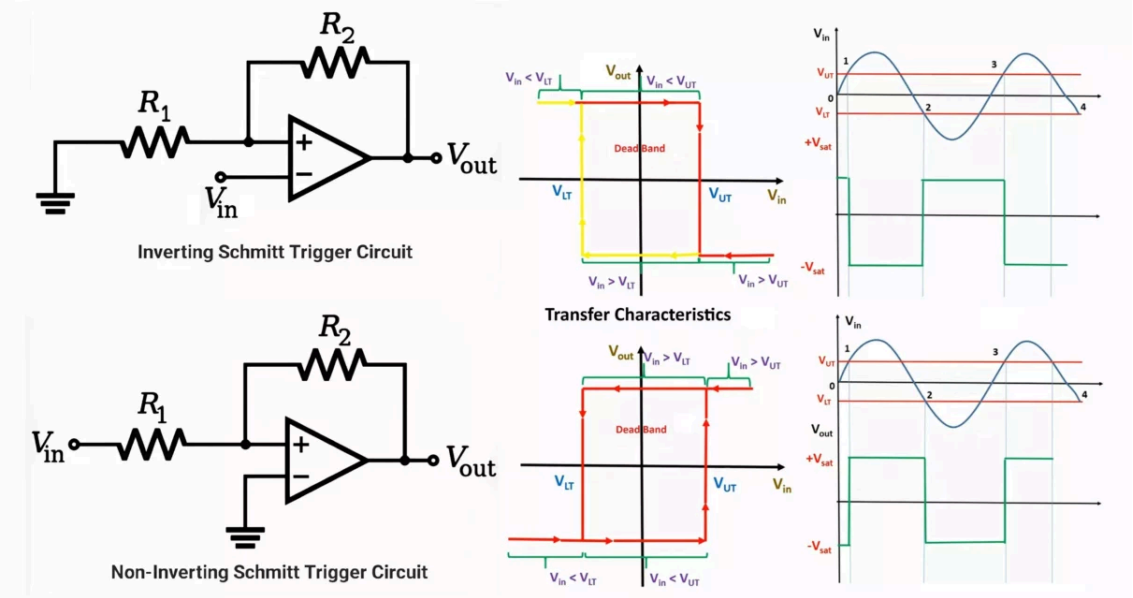
Um exemplo clássico é o barramento I2C, onde as linhas SDA (dados) e SCL (*clock*) utilizam saídas de dreno aberto, permitindo que qualquer dispositivo, mestre ou escravo, possa transmitir sem conflito. O nível 1 é mantido pelos resistores de *pull-up* e qualquer dispositivo pode forçar 0 sem competição. Outro uso comum ocorre em sistemas embarcados que utilizam interrupções

compartilhadas, onde múltiplos dispositivos podem sinalizar uma interrupção em uma mesma linha, ou ainda em sinais de controle simples como “pronto para transmissão” ou “falha detectada”.

## Otimização da aquisição de sinais analógicos

No que diz respeito à aquisição de sinais analógicos, os ADCs internos dos microcontroladores geralmente possuem resolução limitada e podem ser afetados por ruídos. Quando exigimos maior precisão ou estabilidade nas medições, é comum empregar circuitos auxiliares, como amplificadores operacionais para condicionamento de sinal, filtros analógicos para rejeição de ruído e referências de tensão externas que aumentam a confiabilidade do sistema de aquisição.

Embora alguns microcontroladores modernos possam ter recursos internos para **mitigar ruídos** ou até mesmo funções de *Schmitt Trigger* programáveis em seus pinos, a implementação externa com componentes discretos ainda é uma prática comum em aplicações críticas. Um *Schmitt Trigger* é um tipo de comparador de tensão que incorpora histerese, o que significa que ele possui dois limiares de comutação distintos: um para quando a tensão de entrada está subindo e outro para quando está caindo.




Fonte: Hackatronic.

Essa característica única permite que ele transforme sinais de entrada lentos ou ruidosos em saídas digitais claras e bem definidas, sem oscilações indesejadas, conhecidas como *chattering* ou *debouncing*. Ao ignorar pequenas flutuações de ruído que ocorrem entre os dois limiares, o Schmitt Trigger garante uma comutação limpa e robusta, tornando-o ideal para aplicações onde a integridade do sinal é crítica, como na interface com sensores ou em circuitos de detecção de nível. Um exemplo clássico de *chip* que implementa essa funcionalidade é o 74HC14. Este circuito integrado contém seis inversores (portas NOT) independentes, mas com entradas do tipo Schmitt Trigger. Isso significa que cada uma das seis portas do 74HC14 não apenas inverte o sinal lógico, mas também o “limpa” ruídos. Se há um sinal de sensor analógico que varia lentamente ou está sujeito a ruído elétrico antes de ser lido por um microcontrolador, passá-lo por uma das entradas do 74HC14 antes de enviá-lo para um pino GPIO do microcontrolador garante que o microcontrolador

receberá um sinal digital estável, sem transições falsas ou oscilações. Isso é particularmente útil para a leitura de botões mecânicos (evitando *debouncing* de *hardware*) ou sinais analógicos que precisam ser convertidos em digitais.

Os ADCs (do inglês *Analog-Digital Converters*) internos dos microcontroladores frequentemente possuem resolução limitada (por exemplo, 10 ou 12 *bits*) e são susceptíveis a ruídos, o que pode comprometer a precisão das medições. Quando a aplicação exige maior exatidão ou estabilidade na leitura de sinais, como em sensores de alta precisão ou sistemas de controle críticos, é prática comum empregar circuitos auxiliares de condicionamento de sinal, como amplificadores operacionais para condicionamento de sinal, filtros analógicos para rejeição de ruído e referências de tensão externas. Essas soluções aprimoram a qualidade do sinal analógico antes que ele chegue ao ADC do microcontrolador.

Os **amplificadores operacionais** (op-amps) são componentes versáteis, capazes de manipular sinais analógicos. No contexto da aquisição de dados, eles são usados para:

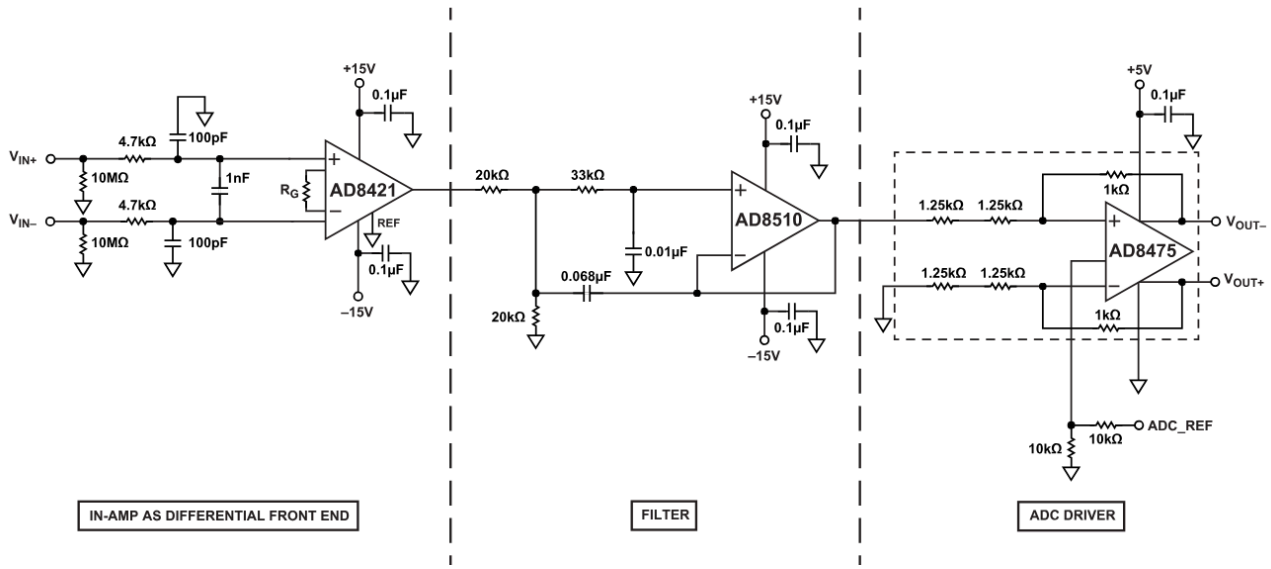
- amplificação: Sinais de sensores que geram tensões muito baixas (como termopares ou células de carga) precisam ser amplificados para usar a faixa completa de entrada do ADC. Um op-amp configurado como amplificador não inversor ou diferencial pode elevar esse sinal para um nível ideal.
- bufferização (seguidor de tensão): Alguns sensores têm alta impedância de saída, o que pode "carregar" a entrada do ADC e distorcer a leitura. Um op-amp configurado como *buffer* (ganho unitário) isola a fonte do sinal da entrada do ADC, garantindo que a impedância da fonte não afete a precisão e o tempo de conversão.
- subtração/soma: Em sistemas com múltiplos sensores ou sinais de referência, op-amps podem subtrair um *offset* ou somar múltiplos sinais.
- conversão de corrente para tensão (I/V): Muitos sensores, como fotodiodos, geram uma corrente proporcional à grandeza física. Um op-amp pode converter essa corrente em uma tensão mensurável pelo ADC.

Essencialmente, o op-amp ajusta o sinal para que ele se encaixe perfeitamente na faixa dinâmica e nas características de impedância do ADC, otimizando a utilização da resolução disponível. Existem diversos tipos de op-amps, cada um com suas particularidades. Para aplicações mais gerais e menos críticas, modelos como o LM358/TL072 são acessíveis e eficientes. O LM358 é popular por operar com uma única fonte de alimentação (*single-supply*), enquanto o TL072 se destaca por seu baixo ruído, característica atribuída à sua entrada de transistores de efeito de campo de junção (*JFET input*). Quando a necessidade é de alta precisão e rejeição de ruído em modo comum (CMR) em sinais diferenciais, como os provenientes de células de carga ou outros sensores de baixa amplitude, amplificadores de instrumentação são recomendáveis. *Chips* como INA122/AD620/AD8421 são exemplos de componentes que já integram a configuração otimizada para amplificação diferencial de alto desempenho. Ainda mais especializados são os amplificadores como o AD8475. Este é um amplificador diferencial de alto modo comum que se especializa em converter grandes tensões de modo comum, que excederiam a faixa de entrada de um ADC ou de um amplificador de instrumentação comum, em um sinal diferencial de saída com faixas de entrada mais limitadas e adequadas para o ADC. Ou seja, ele lida com situações onde o ruído de modo comum é excepcionalmente alto, garantindo que o sinal útil seja isolado e preparado para a digitalização.

Ruídos elétricos são quase inevitáveis em qualquer ambiente e podem corromper seriamente as medições de sinais analógicos. Para combater essas interferências indesejadas, empregamos **filtros analógicos**, circuitos projetados especificamente para remover ou atenuar o ruído antes que ele afete o sinal. Ao aplicar um filtro analógico antes do ADC, garantimos que o conversor receba um sinal mais "limpo", o que resulta em leituras significativamente mais estáveis e precisas, menos susceptíveis a variações aleatórias. Esses filtros podem ser passivos, construídos com componentes básicos como resistores, capacitores e indutores (como já vimos), ou ativos, que incorporam amplificadores operacionais para maior flexibilidade e desempenho.

Entre os tipos mais comuns de filtros para condicionamento de sinal, destacam-se filtros passa-baixa, que suavizam sinais permitindo a passagem de frequências baixas, onde o sinal útil geralmente reside, e atenuam frequências altas, onde o ruído de alta frequência, como o gerado por fontes chaveadas ou interferência RF, costuma se manifestar, e filtros *notch*, ou rejeita-banda, que são projetados para atenuar uma banda muito estreita de frequências, como o zumbido de 60 Hz da rede elétrica brasileira, que pode ser induzido nos cabos de sinal. Exemplos de *chips*, que atuam em conjunto com resistores e capacitores, são op-amps quadrúplos de propósito geral LM324/TL084/AD8510 que podem ser usados para construir filtros ativos de ordem superior (passa-baixa, passa-alta, banda-passa, notch), pois oferecem múltiplos amplificadores em um único encapsulamento, e os controladores com amp-ops integrados, como MF10/LTC1064, que permitem configurar a frequência de corte e o tipo de filtro externamente com resistores ou até mesmo digitalmente, oferecendo grande flexibilidade.

O esquemático a seguir ilustra o projeto de um condicionador de sinais que prepara sinais diferenciais analógicos antes de serem processados por um conversor analógico-digital. Nesse circuito, os sinais são cuidadosamente amplificados pelo AD8421, filtrados usando o AD8510 (configurado como filtro, já que é um op-amp de precisão para essa finalidade) e, por fim, condicionados pelo AD8475 para atingir os valores ideais para a entrada do conversor.

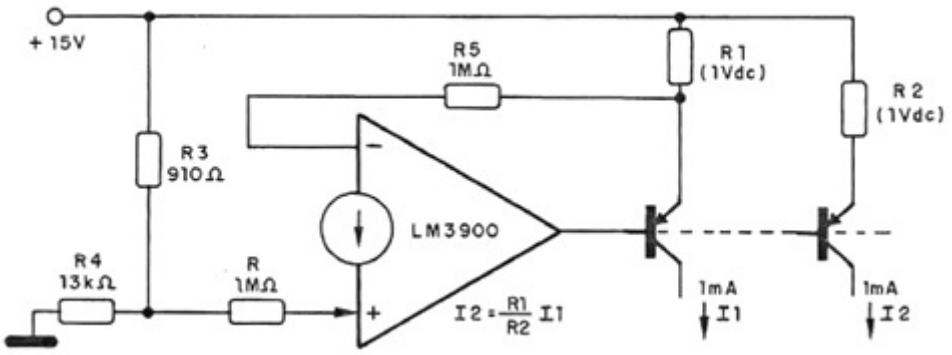



Fonte: Analog Devices.

Os op-amps também desempenham um papel significativo na **interface de corrente** em aplicações analógicas ou mistas. Quando um microcontrolador precisa gerar um sinal analógico com corrente

controlada para atuadores como válvulas proporcionais, resistores de aquecimento variáveis, ou LEDs que exigem controle de brilho preciso, um op-amp pode ser configurado como uma fonte de corrente controlada por tensão. O sinal de tensão analógico gerado pelo microcontrolador (via PWM e filtro, ou por um DAC externo) é aplicado à entrada do op-amp, que então amplifica e regula a corrente através da carga, mantendo-a constante ou proporcional à tensão de entrada, independentemente de variações na resistência da carga. Isso permite um controle muito mais fino e estável de dispositivos que respondem a níveis específicos de corrente.

Considere uma fonte de corrente múltipla baseada no op-amp LM3900 mostrada na seguinte figura. Neste circuito, uma referência de tensão precisa de 1V é estabelecida pelos resistores R3 e R4. Graças à realimentação negativa (um conceito fundamental em op-amps que força a entrada a se igualar a uma referência), essa tensão de 1V de referência é replicada no resistor R1, causando uma queda de tensão de exatamente 1V sobre ele. Essa queda de tensão em R1 é o que, por sua vez, controla o transistor conectado à saída do Op-Amp, resultando na produção da corrente desejada. O resistor R2, nesse arranjo, oferece a flexibilidade de ser ajustado para obter correntes diferentes da corrente base de 1 mA, permitindo a personalização do valor da corrente fixa para diversas aplicações.

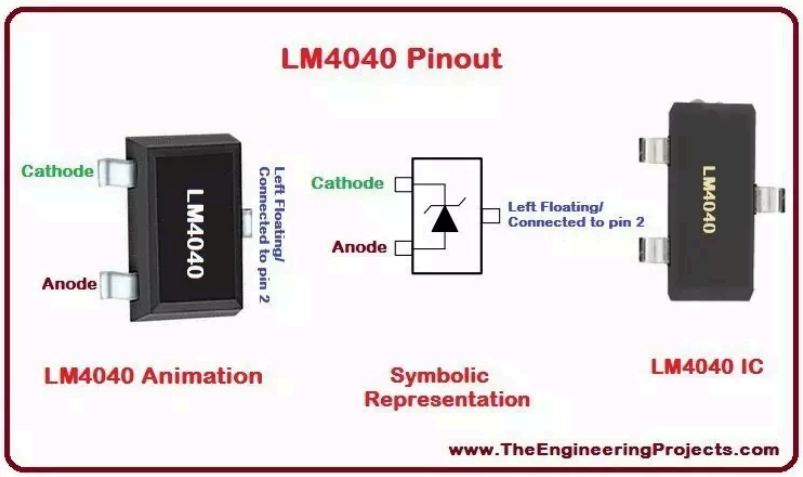


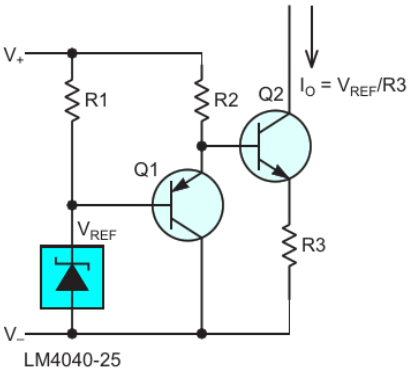

Fonte: newtonbraga.

A **referência de tensão** (Vref) é o ponto de comparação fundamental que um ADC usa para transformar um sinal analógico em seu equivalente digital. A precisão de qualquer medição digital está diretamente ligada à estabilidade e exatidão dessa Vref. Embora a maioria dos microcontroladores venha com referências de tensão internas, elas podem ser afetadas por variações de temperatura, ruídos da própria alimentação do *chip* e limitações inerentes ao processo de fabricação, o que compromete sua precisão. Para superar essas limitações, muitos projetos optam por utilizar uma referência de tensão externa. Essa é uma escolha por um componente dedicado, de alta precisão, projetado para fornecer uma tensão de referência extremamente estável e exata para o ADC.

Essa decisão impacta diretamente na precisão: imagine um ADC de 10 *bits* usando uma referência interna de 5V que varia em 1%; isso pode levar a erros significativos na leitura. Em contraste, uma referência externa de 5V com uma variação de apenas 0,01% reduz drasticamente esse erro, assegurando que cada "passo" do ADC, o LSB (do inglês *Least Significant Bit*), seja o mais preciso possível. Além disso, uma referência externa é projetada para ter alta imunidade a ruídos da alimentação do microcontrolador, garantindo que o ADC sempre tenha um ponto de comparação limpo e estável, independentemente das flutuações na fonte de energia. Para ilustrar, alguns exemplos práticos de *chips* de referência de tensão incluem o LM4040 e o TL431. Ambos são

referências de tensão *shunt*[1], como ilustrado no esquemático a seguir, que oferecem baixo custo e bom desempenho, adequados para uma vasta gama de aplicações. Contudo, para cenários que demandam a máxima exatidão e estabilidade, com deriva de temperatura e ruído extremamente baixos, as séries como a ADR45xx da Analog Devices (por exemplo, ADR4525, ADR4530), que utilizam tecnologia X-FET, são as escolhas ideais.




Fonte: The Engineering Projects. Autor: Stephen Woodward

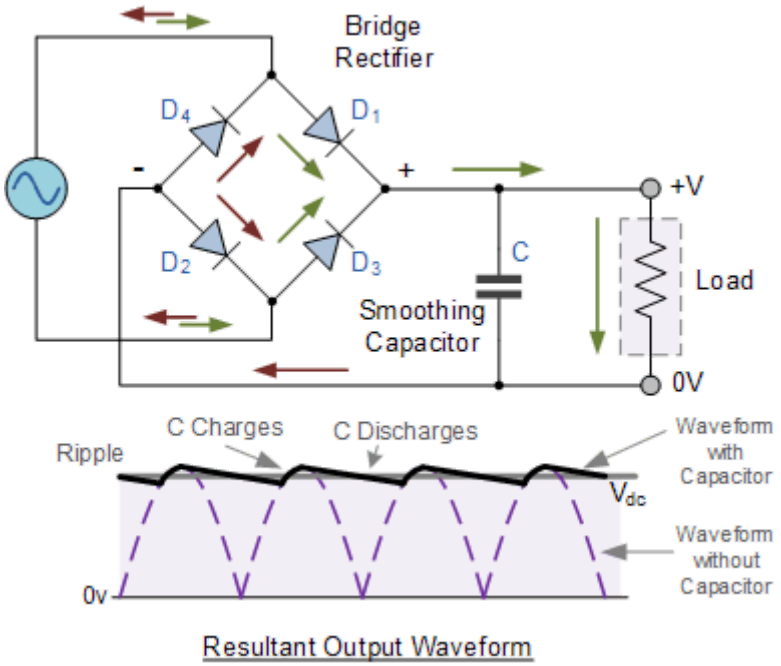
Em conjunto, o uso estratégico de amplificadores operacionais, filtros analógicos e referências de tensão externas permite que os engenheiros extraiam o máximo de precisão e estabilidade de seus sistemas de aquisição de sinais analógicos, superando as limitações dos ADCs internos e garantindo a confiabilidade necessária para aplicações exigentes.

## Interfaces com corrente alternada (CA)

Os circuitos de interface entre sistemas de corrente alternada (CA) e microcontroladores de corrente contínua (CC) são uma parte essencial do campo de interfaces eletrônicas. Sua relevância é inegável em automação industrial, sistemas de medição de energia, controle de potência e eletrodomésticos inteligentes. Essencialmente, eles garantem o isolamento elétrico, a segurança e a compatibilidade entre dois "mundos" fisicamente distintos: o domínio da alta tensão e corrente alternada, e o domínio lógico e de baixa tensão dos microcontroladores. Para que esses dois mundos se comuniquem e interajam de forma eficaz, são necessários circuitos que possam converter energia e sinais entre eles. Por um lado, temos os retificadores, que transformam a corrente alternada da rede elétrica em corrente contínua, uma etapa crucial para alimentar os microcontroladores e a maioria dos componentes eletrônicos. Por outro lado, para controlar cargas CA complexas, são empregados inversores, que convertem a corrente contínua dos microcontroladores (ou de baterias) em corrente alternada com as características desejadas (como frequência e forma de onda controladas). Esses circuitos de conversão, retificadores e inversores, juntamente com os isoladores e condicionadores de sinal, formam a base de sistemas híbridos que combinam inteligência digital com o poder da energia CA.

[1] Tensão *shunt* se refere a um tipo de componente que regula a tensão operando em paralelo (ou em *shunt*) com a carga do circuito. Se a tensão na linha tender a subir, o dispositivo drena mais corrente através de si mesmo para o terra, aumentando a queda de tensão no resistor série e, assim, "puxando" a tensão da linha de volta para baixo, mantendo-a constante. Se a tensão tentar cair, ele drena menos corrente.

O tipo mais comum de circuito retificador para converter CA em CC é a **ponte retificadora de onda completa**. Embora o termo “ponte de Wheatstone” seja associado a medição de resistência, a ponte para retificação é uma ponte de diodos. Ela consiste em quatro diodos organizados em uma configuração de ponte. Durante o semiciclo positivo da tensão CA de entrada, dois diodos da ponte conduzem, permitindo que a corrente flua através da carga em uma direção específica. No semiciclo negativo, os outros dois diodos conduzem, mas a corrente continua a fluir na *mesma direção* através da carga. O resultado é uma tensão pulsante de CC na saída. Para transformar essa tensão pulsante em uma CC mais suave e estável, essencial para alimentar circuitos eletrônicos sensíveis como microcontroladores, geralmente é adicionado um capacitor de filtro em paralelo com a carga. Este capacitor se carrega durante os picos da onda retificada e se descarrega lentamente nos vales, "suavizando" a tensão e reduzindo a ondulação (*ripple*). Para aplicações que exigem CC de alta qualidade, reguladores de tensão lineares ou chaveados podem ser adicionados após o filtro capacitivo.



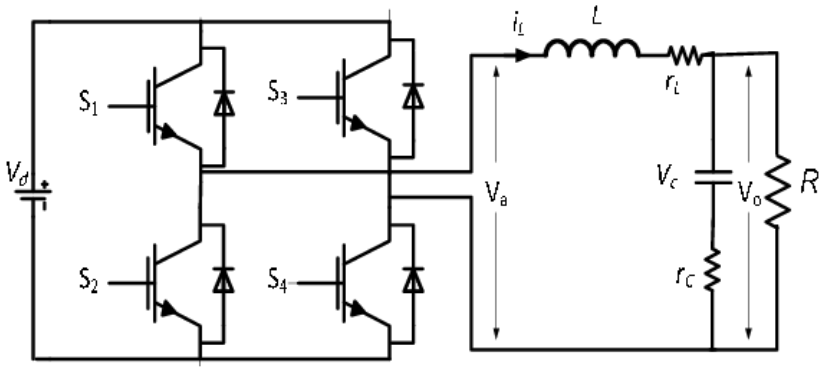

Fonte: Electronic-tutorials.

Um exemplo comum de *chip* comercial que encapsula essa funcionalidade é o DF005M. Este é um encapsulamento compacto que integra os quatro diodos da ponte retificadora, simplificando o design e a montagem em diversas aplicações de baixa a média potência.

Inversores são circuitos que realizam a tarefa oposta aos retificadores: eles transformam corrente contínua (CC) em corrente alternada (CA). Sua importância é fundamental em diversas aplicações, como sistemas de energia solar, onde a CC gerada pelas placas fotovoltaicas precisa ser convertida para alimentar a rede CA ou eletrodomésticos, e no controle preciso de motores CA. Em inversores modernos, a forma mais eficiente e controlável de realizar essa conversão é através da técnica de modulação por largura de pulsos (em inglês, *Pulse Width Modulation* – PWM), daí serem comumente chamados de **inversores PWM**.

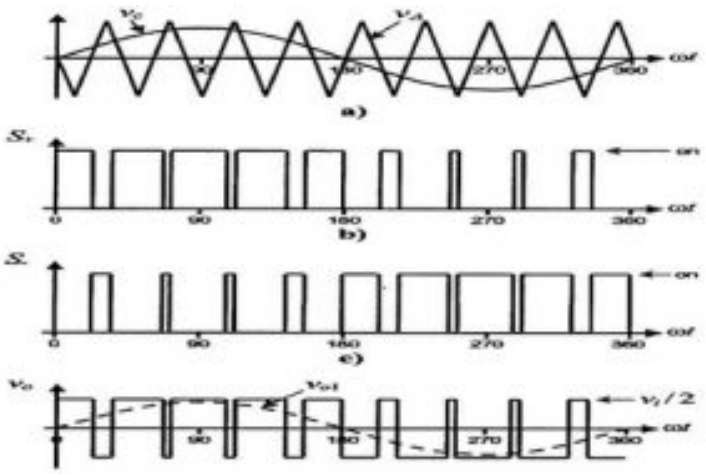
Um inversor típico para uma única fase utiliza uma configuração de ponte H semelhante àquela usada para controlar motores CC, composta por quatro chaves eletrônicas de potência, como MOSFETs ou IGBTs. A corrente contínua (CC) é aplicada a essa ponte, como ilustrado na seguinte

figura. Ao chavear os transistores da ponte em sequências específicas e com larguras de pulso variáveis, o inversor consegue gerar uma onda de tensão de saída que se aproxima de uma senoide CA.




Fonte: Research Gate.

O PWM atua controlando precisamente o tempo de condução (largura do pulso) de cada chave. Para gerar os sinais de controle, uma onda senoidal de referência de baixa frequência é comparada com uma onda triangular (portadora) de alta frequência. Simplesmente, quando a onda senoidal é maior que a triangular, as chaves são ligadas; quando é menor, são desligadas. Esse processo gera trens de pulsos de larguras variáveis na saída, conforme ilustrado na figura a seguir. Ao variar a largura desses pulsos, a tensão média na saída é modulada de tal forma que, ao ser devidamente filtrada, resulta em uma forma de onda senoidal de corrente alternada. Para garantir uma CA limpa e precisa para a carga, filtros de saída (compostos por indutores e capacitores) são então utilizados para suavizar a forma de onda PWM, eliminando as componentes de alta frequência indesejadas. Esse método de controle via PWM permite ajustar tanto a tensão quanto a frequência da CA de saída com alta eficiência e mínimo conteúdo harmônico.



Fonte: EL-PRO-CUS.

## Expansão de memória

Em sistemas embarcados, especialmente com microcontroladores, a **expansão de memória** é um recurso essencial quando a capacidade de armazenamento ou de execução de código interna do

microcontrolador é insuficiente para a aplicação desejada. Ela permite adicionar *chips* de memória externos, aumentando a capacidade total disponível para dados e programas. Essa expansão não é um processo isolado; ela depende intrinsecamente de outros circuitos auxiliares, como os circuitos de interface, que gerenciam a comunicação entre o microcontrolador e a memória externa. Além disso, memórias externas precisam de circuitos de alimentação adequados e um sinal de relógio (*clock*) preciso para suas operações.

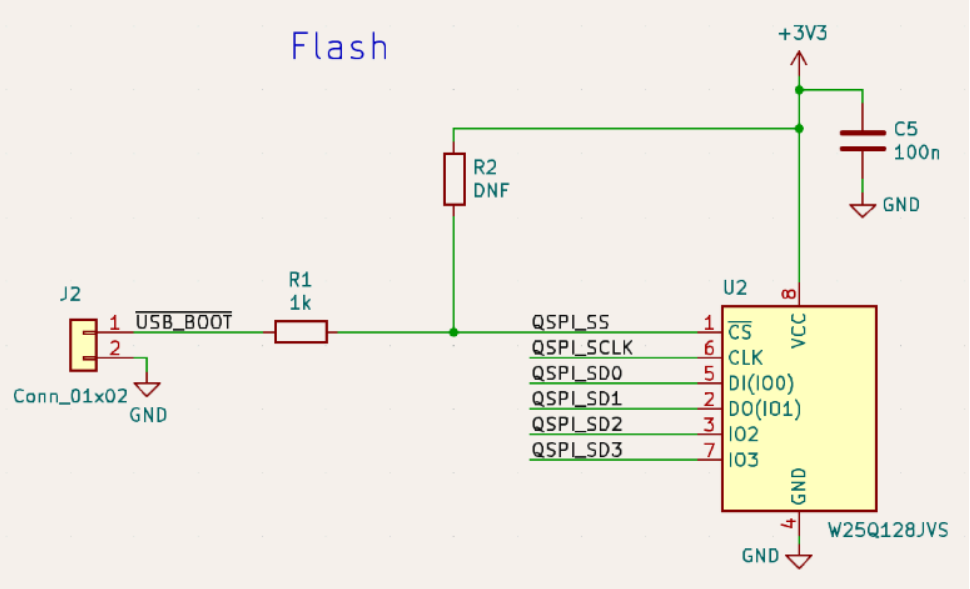
Os *chips* de memória externos se comunicam com o microcontrolador de diversas maneiras, geralmente classificadas pelo tipo de interface serial ou paralela. A **comunicação paralela** em *chips* de memória emprega múltiplos pinos para dados (como 8, 16 ou 32 *bits*), além de pinos de endereço e linhas de controle para leitura, escrita e seleção do *chip*. Historicamente, sua principal vantagem é a alta velocidade de transferência, já que diversos *bits* são transmitidos simultaneamente. No entanto, suas desvantagens incluem a necessidade de um grande número de pinos no microcontrolador, o que eleva o custo e o tamanho da placa, maior complexidade no roteamento da PCB e maior susceptibilidade a problemas de *timing* e *crosstalk* em altas frequências. Memórias SRAM (como CY62256), DRAM (como MT40A) e Flash (como M29W640FB) paralelas são exemplos típicos que utilizam essa abordagem.

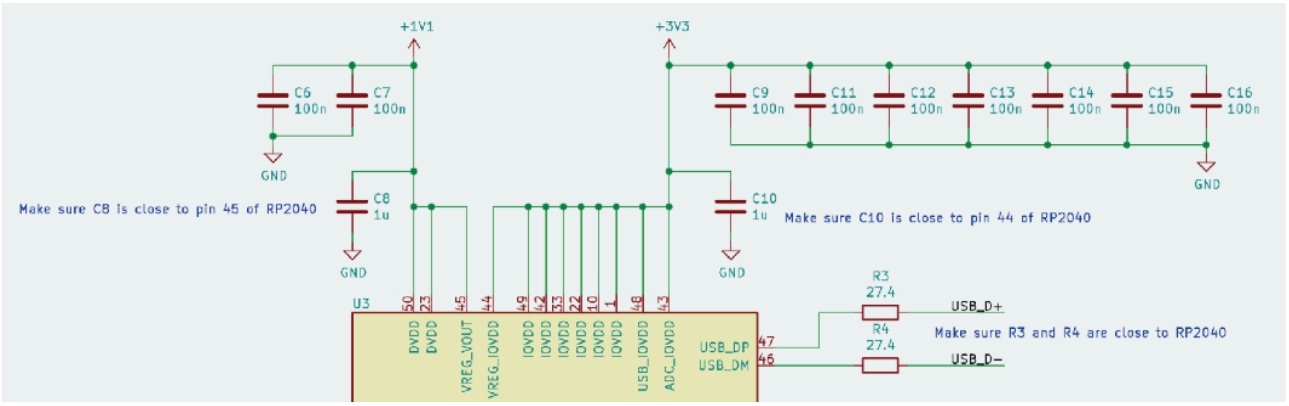
Por outro lado, a comunicação serial se destaca por usar um número reduzido de pinos (geralmente de 1 a 4) para transmitir dados sequencialmente, *bit* a *bit*. Essa característica resulta em economia de pinos no microcontrolador, simplifica o *layout* da placa, reduz custos e a torna menos susceptível a problemas de *timing* e *crosstalk* em distâncias curtas. A desvantagem é que, em termos de *throughput* bruto por *bit*, ela tende a ser mais lenta que a comunicação paralela, embora protocolos modernos de alta velocidade venham diminuindo essa diferença. Exemplos de protocolos seriais incluem o I2C (do inglês *Inter-Integrated Circuit*), como 24LC256, que usa apenas dois fios (dados e *clock*) e é ideal para dispositivos de baixa velocidade como EEPROMs e sensores; o SPI (do inglês *Serial Peripheral Interface*), como W25Q128FV, que emprega tipicamente quatro fios (MOSI, MISO, SCK, CS) e oferece maior velocidade, sendo comum para Flash e EEPROM; e o QSPI (Quad SPI), como MT25QL256ABA, e OctoSPI, como IS25WX512**,** que são versões aprimoradas do SPI que utilizam 4 ou 8 linhas de dados, respectivamente, para transferências de alta velocidade em memórias Flash de alto desempenho.

A escolha do tipo de memória e da interface de comunicação dependerá das necessidades da aplicação em termos de velocidade, capacidade, custo e complexidade do projeto. A integração eficiente desses componentes externos com o microcontrolador é um pilar fundamental no projeto de sistemas embarcados robustos e funcionais.

Por exemplo, para permitir que o microcontrolador RP2040 do Raspberry Pi Pico possa armazenar e executar seu *firmware*, é necessário utilizar uma memória *Flash* externa do tipo Quad SPI (QSPI). Foi utilizado o chip W25Q128JVS no Raspberry Pi Pico, com capacidade de 128 Mbit (16 MB), que representa o limite máximo suportado pelo RP2040. A conexão entre o microcontrolador e a memória Flash deve ser feita com trilhas curtas e diretas, especialmente nas linhas QSPI, a fim de preservar a integridade do sinal e minimizar efeitos de *crosstalk*. A linha de seleção do *chip* (QSPI_SS) requer atenção especial: apesar do RP2040 ativar internamente um *pull-up* durante a inicialização, um resistor externo de *pull-up* (R2) a 3,3 V pode ser incluído como precaução, principalmente se for utilizada uma *Flash* diferente da recomendada. Também é adicionada uma resistência de 1 kΩ (R1) conectada a um *jumper* (USB_BOOT), permitindo forçar o RP2040 a

entrar no modo BOOTSEL ao manter o QSPI_SS em nível lógico baixo durante o *reset*. Capacitores cerâmicos de 1 µF (C8 e C10), próximos às entradas e saídas do regulador interno de 1,1 V (VREG_IN e VREG_OUT), são essenciais para garantir a estabilidade da alimentação do *chip* e atender aos requisitos de baixa ESR (do inglês *Equivalent Series Resistance*) recomendados no *datasheet*. Essas práticas garantem um funcionamento confiável da memória Flash externa e do processo de *boot* do microcontrolador.



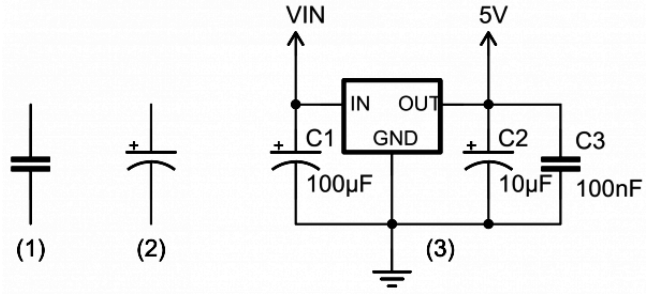

Fonte: Raspberrypi.

## Circuitos de alimentação

Um circuito de alimentação é crucial para um sistema eletrônico ou elétrico. Obviamente, tais sistemas não funcionam sem energia elétrica. Entre as características de uma boa alimentação estão:

- fonte de tensão estável e sem grandes ondulações (do inglês, \textit{ripples}).
- corrente suficiente para a operação do sistema.
- alta eficiência energética.
- desempenho estável para a faixa de temperatura de operação.
- razoável desempenho térmico mesmo sem circulação do ar.
- adequada filtragem de ruídos, compatível com os padrões de interferências eletromagnéticas -- EMI (do inglês *ElectroMagnetic Interference*).

- apropriado desacoplamento de ruídos de alta frequência nos sinais de alimentação, como o uso de capacitores na entrada e saída de um regulador de tensão para desacoplamento de ruídos de alta frequência nos sinais de alimentação.




Seguem-se alguns modelos de fontes de alimentação (do inglês *power supply unit* – PSU), comumente empregados nos sistemas embarcados baseados em microcontroladores.

## Fonte de alimentação de parede

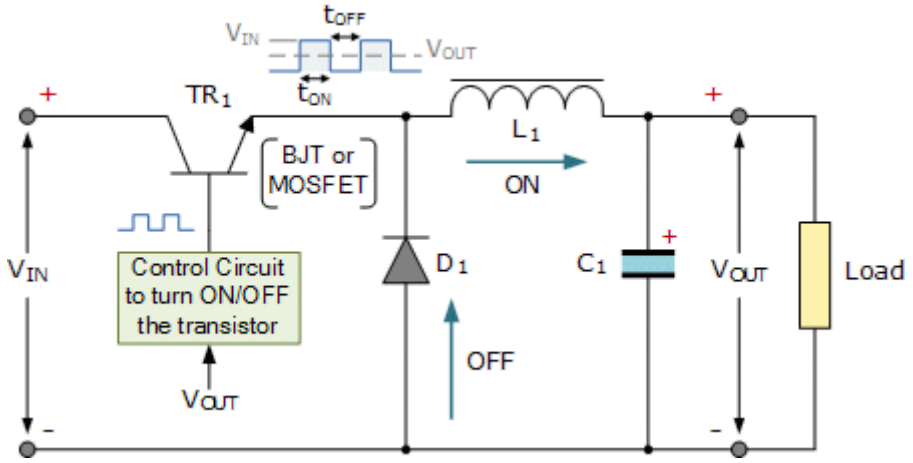
Dispositivos alimentados diretamente pela rede elétrica (tomada) são comuns em aplicações que exigem mais potência do que baterias podem fornecer. Exemplos incluem equipamentos hospitalares e industriais de uso geral (não-críticos), além de sistemas de áudio e vídeo domésticos. Esses dispositivos são tipicamente alimentados por fontes de alimentação de parede, popularmente conhecidas como eliminadores de pilha. Tais eliminadores conseguem converter a tensão alternada (AC) de 127V ou 220V em uma tensão contínua (CC), que geralmente varia de 3.3V a 12V, com corrente de até alguns ampères. Os principais parâmetros para especificar um eliminador de pilha são justamente a tensão e a corrente, que devem atender à demanda da tecnologia a ser alimentada.

Internamente, o circuito de uma fonte de alimentação de parede é tipicamente composto por um transformador abaixador de tensão (em inglês, *step-down transformer*), um circuito retificador de onda completa e um regulador de tensão. O transformador é responsável por converter a energia de alta tensão e baixa corrente da rede para baixa tensão e alta corrente, com o valor da conversão determinado pela relação de espiras entre seu primário e secundário. A retificação, por sua vez, pode ser realizada por uma ponte retificadora, muitas vezes complementada por um filtro capacitivo para atenuar variações na tensão e garantir uma saída mais estável, como vimos anteriormente.

A tensão de saída do circuito retificador precisa ser estabilizada para atender aos requisitos de um sistema computacional. Essa tarefa é realizada por um regulador de tensão, que garante um fornecimento de energia constante e seguro. Existem duas tecnologias principais para esses reguladores:

- reguladores de tensão lineares que utilizam uma tensão de referência estável, geralmente um diodo Zener, para manter a tensão de saída constante. Eles operam com pelo menos um componente ativo, como transistores, e exigem que a tensão de entrada seja maior que a tensão de saída desejada. Embora forneçam uma tensão de saída limpa e estável em uma determinada faixa de correntes, sendo populares por sua qualidade de sinal (baixo ruído), dimensões compactas e custo acessível, os reguladores lineares dissipam o excesso de tensão na forma de calor. Isso resulta em baixa eficiência energética e, para potências mais altas, a necessidade de dissipadores de calor volumosos. Além disso, a faixa de tensões que podem estabilizar é limitada.

- reguladores de tensão chaveados que empregam um modulador de largura de pulso (PWM) para controlar a energia de saída. A grande vantagem é sua capacidade de reduzir, elevar e/ou inverter a polaridade da tensão de entrada, oferecendo maior flexibilidade. Eles transferem energia em "pacotes" discretos, utilizando chaves (tipicamente MOSFETs) e componentes passivos como indutores e capacitores. Ao controlar a duração desses pacotes, é possível ajustar precisamente o nível da tensão de saída. Estes reguladores se destacam pela eficiência energética (podendo chegar a 90%, em contraste com os 70% a 80% dos lineares) e pela capacidade de suportar níveis de energia de entrada mais altos. No entanto, a alta frequência de chaveamento de seu circuito de controle pode gerar mais ruído, e a complexidade inerente de seu *design* os torna mais caros e volumosos que os reguladores lineares. Para mitigar as interferências eletromagnéticas (EMI) e o ruído, são frequentemente integrados circuitos de filtragem e controle de ruído, sendo um circuito passivo LC simples muitas vezes suficiente para essa finalidade.




Fonte: Electronic-tutorials.

As fontes de alimentação que utilizam reguladores de tensão chaveados são comumente chamadas de fontes chaveadas (em inglês *Switched-Mode Power Supply - SMPS*). Atualmente, elas são amplamente empregadas em diversos dispositivos eletrônicos, como telefones portáteis, videogames, câmeras digitais, robôs e computadores, devido à sua alta eficiência. A figura a seguir ilustra a diferença prática entre essas tecnologias: uma fonte de alimentação regulada com o regulador linear CI LM317 e outra com o regulador chaveado CI LM2596. Note, em particular, a presença de um dissipador de calor no entorno do regulador LM317, necessário para evitar o superaquecimento, uma característica da menor eficiência dos reguladores lineares.



Regulador LM317 Regulador LM2596

Ao selecionar uma fonte de alimentação de parede para um projeto de sistema embarcado, alguns pontos são cruciais para garantir o funcionamento adequado e a longevidade do dispositivo. Primeiramente, observe a voltagem e a corrente fornecidas pela fonte. É essencial que a voltagem seja compatível com a do sistema. Quanto à corrente, é uma boa prática escolher uma fonte que entregue uma corrente maior do que a estimada para o sistema. Isso oferece uma margem de segurança para picos de consumo inesperados e evita sobrecargas. Além disso, a qualidade da fonte é vital: fontes de má qualidade podem injetar ruídos elétricos no sistema e não conseguir manter a tensão estável sob diferentes demandas de corrente, comprometendo a operação do dispositivo. A eficiência do regulador de tensão interno da fonte também é um fator a ser considerado, impactando o consumo de energia e a dissipação de calor.

Outro ponto importante é o tipo e a polaridade dos conectores. É muito comum que as fontes utilizem um *plug* bipolar coaxial, conforme ilustrado na figura a seguir. No sistema a ser alimentado, um conector tipo soquete ou “jack” é instalado, como também é mostrado na figura.

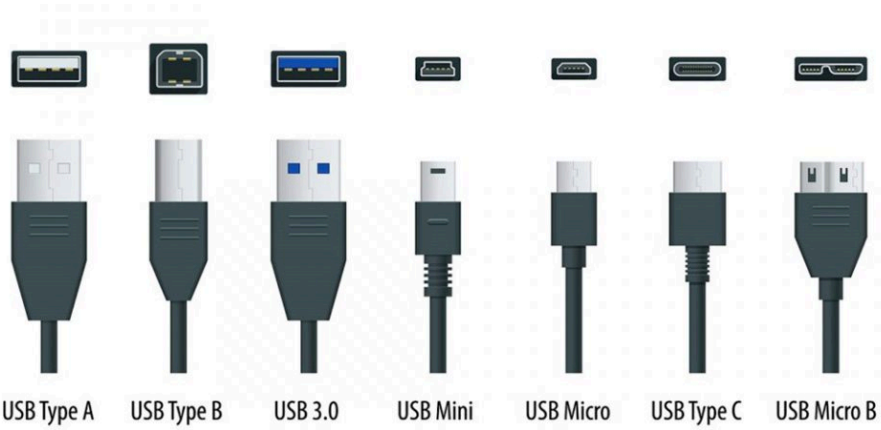
Plug P4 Jack P4

A principal dificuldade com os conectores tradicionais reside na falta de padronização. Existem diversos tamanhos de *plugs* e *jacks* considerados compatíveis, mas que variam quanto à forma e à localização dos polos positivo e negativo (polaridade). Embora a configuração mais comum

coloque o polo positivo no centro do plugue e o negativo no anel externo, a ausência de um padrão universal pode gerar confusão e até danificar dispositivos quando a polaridade é invertida.

Ao longo do tempo, alguns conectores USB ajudaram a mitigar parcialmente esses problemas. O USB tipo A, por exemplo, foi amplamente utilizado em computadores e carregadores, sendo confiável, mas com limitação por exigir inserção em orientação específica. Já o micro-USB se tornou popular em *smartphones* e acessórios portáteis, oferecendo tamanho reduzido, mas ainda assim sujeito a desgaste e também com orientação fixa. Esses padrões trouxeram avanços importantes, mas ainda apresentavam limitações em termos de usabilidade e versatilidade.

Nesse contexto, a adoção do conector USB tipo C representa uma solução mais completa para os desafios de padronização. Ele vem se consolidando como um padrão global amplamente reconhecido, promovendo maior compatibilidade entre diferentes dispositivos e carregadores. Além disso, o USB-C oferece praticidade no uso, já que pode ser conectado em qualquer orientação — o que reduz erros e o risco de danos físicos. Outra vantagem relevante é a possibilidade de incorporar inteligência tanto na fonte quanto no dispositivo por meio da tecnologia USB *Power Delivery* (USB-PD), que permite negociar dinamicamente os níveis de tensão e corrente, além de habilitar funcionalidades avançadas.




Fonte: Altervista.

## Baterias recarregáveis

Uma bateria é um dispositivo eletroquímico que armazena energia elétrica na forma de energia química e a reconverte em corrente elétrica contínua (CC) quando em uso. Internamente, ela contém produtos químicos que reagem, gerando um gradiente de elétrons, em processos denominados reações eletroquímicas. Ao selecionar uma bateria recarregável para um projeto de engenharia, dois parâmetros são primordiais: sua capacidade e sua taxa de carga/descarga.

A capacidade da bateria define a quantidade total de carga elétrica que ela pode fornecer sob uma dada tensão e temperatura, com a unidade padrão sendo o Ampere-hora (Ah). Por exemplo, uma bateria de 9 Volts com capacidade de 1 Ah pode, teoricamente, fornecer 1 Ampere por uma hora. É crucial entender que essa medição é padronizada, e a bateria é considerada esgotada quando sua

tensão cai abaixo de um valor limite (por exemplo, 5.4 Volts para uma bateria de 9V). Geralmente, a medição em Ampere-hora assume um tempo de descarga de 20 horas, o que implica descarregá-la a uma corrente de 1/20 de sua capacidade nominal durante esse período.

A C-rate (do inglês *Charging/Discharging rate*) é o segundo parâmetro, descrevendo a velocidade com que a carga armazenada pode ser utilizada ou reposta, expressa como um múltiplo da capacidade da bateria. Assim, para uma bateria de 1000 mAh, uma taxa de 1C significa uma corrente de 1 Ampere em 1 hora, enquanto 2C indica que a mesma carga é transferida em 30 minutos, e 0.5C (ou C/2) levaria 2 horas, e assim por diante. Essa relação se estende proporcionalmente, como mostra a tabela. Além disso, a taxa de auto-descarga da bateria, que é a perda de carga mesmo sem uso, é um fator importante a ser considerado no projeto, pois está diretamente relacionada à periodicidade com que a bateria precisará ser recarregada.

| Taxa de carregamento | Tempo |
|---|---|
| 5C | 12 min |
| 2C | 30 min |
| 1C | 1h |
| 0.5C or C/2 | 2h |
| 0.2C or C/5 | 5h |
| 0.1C or C/10 | 10h |
| 0.05C or C/20 | 20h |

Fonte: BatteryUniversity.

As baterias comerciais utilizam diferentes composições químicas, cada uma com características únicas que as tornam adequadas para aplicações específicas em engenharia. Conhecer essas particularidades pode otimizar a seleção em projetos.

- Íon-Lítio (Li-ion): As baterias de Íon-Lítio são a escolha primordial quando alta densidade de energia (menor peso e volume para a mesma capacidade) é o requisito mais crítico. Sua tensão nominal por célula é de 3.6V, e a corrente de carga ideal é tipicamente 1C. Apesar de sua performance superior em densidade energética, essa tecnologia é considerada sensível. Picos de tensão durante a carga ou quedas excessivas de tensão na descarga podem comprometer a segurança e a vida útil da bateria. Por essa razão, um circuito de proteção robusto é mandatório para cada célula, garantindo que os limites de operação seguros sejam sempre respeitados. Uma grande vantagem para a manutenção é que descargas periódicas não são necessárias, e sua taxa de auto-descarga é significativamente menor que a das tecnologias Níquel-Cádmio (NiCd) e Níquel-Hidreto Metálico (NiMH). Embora sejam mais caras inicialmente, são amplamente utilizadas em laptops e telefones celulares. Do ponto de vista ambiental, as baterias de Íon-Lítio são geralmente preferidas por não conterem metais pesados altamente tóxicos como chumbo, cádmio ou mercúrio, comuns em outras químicas.

- Níquel-Cádmio (NiCd): As baterias de Níquel-Cádmio (NiCd) se destacam pela longa vida útil, alta taxa de descarga e custo relativamente baixo, tornando-as ideais para aplicações onde esses fatores são mais relevantes que a densidade de energia. A tensão nominal por célula é de 1.25V, com uma corrente de carga recomendada de 1C. A sua capacidade de suportar altos picos de corrente de descarga, chegando a até 20C, a torna adequada para aplicações de alta potência de pulso. No entanto, sua elevada taxa de auto-descarga impõe a necessidade de recargas mais frequentes, mesmo quando não estão em uso. Suas principais aplicações incluem rádios de comunicação (*walkie-talkies*), equipamentos biomédicos, câmeras de vídeo profissionais e ferramentas elétricas. É importante notar que, devido à presença de cádmio, um metal tóxico, as baterias NiCd são consideradas ambientalmente mais hostis e requerem descarte e reciclagem cuidadosos.
- Níquel-Hidreto Metálico (NiMH): As baterias de Níquel-Hidreto Metálico (NiMH) oferecem uma densidade de energia superior à das NiCd, embora geralmente com uma vida útil ligeiramente menor. Elas compartilham a tensão nominal de 1.25V por célula com as NiCd, mas se beneficiam de uma corrente de carga ideal em torno de 0.5C. Uma vantagem significativa das NiMH é que elas não contêm metais tóxicos, tornando-as uma alternativa mais ecológica que as NiCd. Apresentam uma capacidade 30% a 40% superior em relação às NiCd e são aproximadamente 20% mais caras. Historicamente, foram amplamente utilizadas em dispositivos como telefones móveis e *laptops* antes da ampla adoção do Íon-Lítio.
- Chumbo-Ácido: As baterias de Chumbo-Ácido são uma opção econômica para aplicações de alta potência, especialmente quando o peso não é uma restrição crítica. A tensão nominal por célula é geralmente de 2V. A corrente de carga ideal para otimizar sua vida útil é tipicamente baixa, em torno de 0.2C. São a escolha preferencial em cenários que demandam fornecimento de energia robusto e de longa duração, como em equipamentos hospitalares, cadeiras de rodas motorizadas, sistemas de iluminação de emergência e fontes de alimentação ininterruptas (em inglês, *Uninterruptible Power Supply* – UPS). Embora eficientes para essas aplicações, o chumbo presente em sua composição as classifica como ambientalmente desafiadoras, exigindo processos de reciclagem específicos.

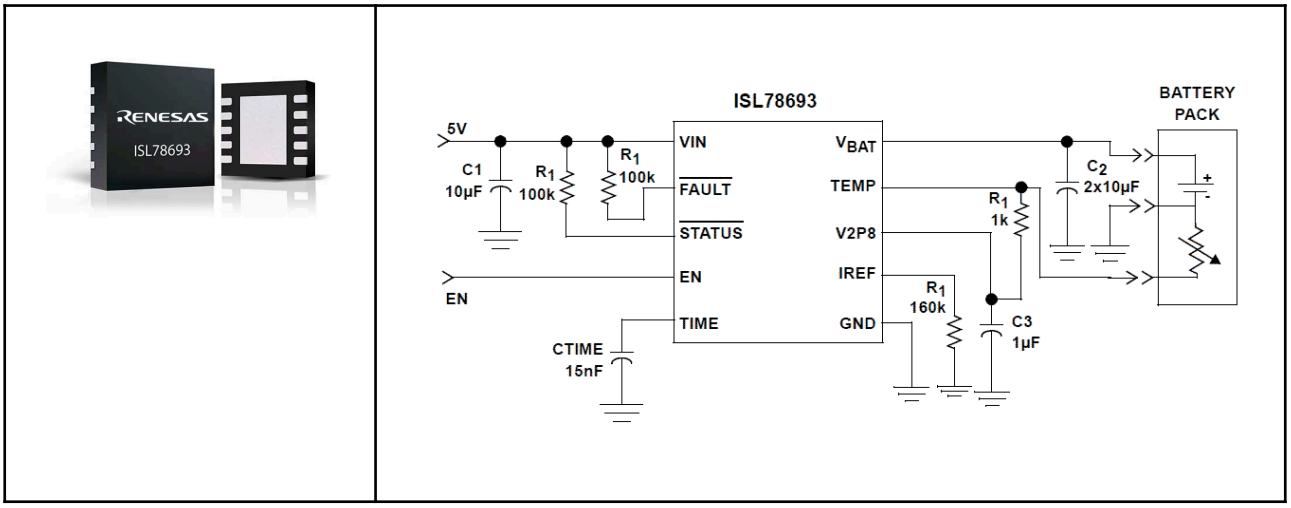
Quando a tensão exigida por um sistema é superior àquela fornecida por uma única célula de bateria, a solução comum é conectar múltiplas células em série. Essa configuração permite somar as tensões das células, atingindo o nível necessário para alimentar o dispositivo.

Os procedimentos de carga variam de acordo com o tipo de bateria. Para garantir uma carga eficiente e segura, muitas baterias requerem um processo em duas etapas: inicialmente em modo de tensão e, posteriormente, em modo de corrente. No modo de tensão, a tensão é mantida constante enquanto a corrente se ajusta da carga zero até o valor desejado. Já no modo de corrente, a corrente é fixa e a tensão é ajustada. A complexidade desses procedimentos, especialmente para tecnologias mais sensíveis como Íon-Lítio, torna inviável a implementação com circuitos de carga simples em projetos básicos. Felizmente, o mercado oferece circuitos dedicados para carga de bateria em diversas aplicações. Estes módulos de gerenciamento de carga, frequentemente encapsulados em uma única pastilha (*chip*) como esses carregadores de TI, são projetados para monitorar a tensão nos terminais da bateria e controlar de forma segura todo o processo de carga. Eles geralmente incluem uma interface digital, como a I2C, que permite a aplicativos externos acessarem dados

importantes sobre a bateria – como capacidade, estado de carga e condições operacionais – além de ajustar parâmetros de carga conforme necessário.

A corrente de carga deve ser especificada com muita atenção. Uma corrente de carga mais alta reduz o tempo total de carregamento, o que é desejável para muitas aplicações. No entanto, ela exige um *design* de circuito de carga mais robusto e complexo, especialmente quando o sistema está em operação simultânea com a carga da bateria. Por outro lado, uma corrente de carga baixa simplifica o circuito, mas pode resultar em um tempo de carga excessivamente longo, tornando-o impraticável para o uso diário.

Atualmente, existem no mercado *chips* programáveis e integrados para gerenciamento de carga. Um exemplo é o ISL78693, projetado especificamente para baterias de Íon-Lítio de 3.6V. A figura que segue ilustra a pastilha 78693 e apresenta um esquema de sua ligação com uma bateria para o gerenciamento de carga, demonstrando a compactação e a inteligência desses componentes.




Fonte: Renesas.

## Baterias não-recarregáveis

As baterias não-recarregáveis, ou primárias, representam uma opção válida de alimentação em sistemas embarcados, especialmente naqueles com baixo consumo de energia e que exigem longa autonomia sem intervenção. Por não permitirem recarga, essas baterias são descartadas ao final de sua vida útil, mas oferecem vantagens em determinadas aplicações. Entre seus principais atrativos estão a alta densidade de energia e a baixa taxa de autodescarga, características que permitem seu uso em dispositivos que permanecem por longos períodos em modo de espera ou operação esporádica. Isso as torna especialmente úteis em aplicações como sensores remotos, dispositivos de Internet das Coisas (IoT), relógios, controles, detectores de fumaça e aparelhos médicos portáteis.

As baterias alcalinas, nos formatos AA ou AAA, são as mais comuns, amplamente disponíveis e de baixo custo, sendo adequadas para prototipagem ou sistemas com consumo moderado. Já as baterias de lítio, como os modelos CR2032 e CR123A, oferecem maior estabilidade e desempenho em diferentes faixas de temperatura, além de maior densidade energética em formatos compactos, sendo ideais para sistemas embarcados miniaturizados. Em aplicações de ultrabaixo consumo, como

em aparelhos auditivos ou sensores ambientais de ciclo lento, baterias de zinco-ar podem ser consideradas, devido à sua elevada capacidade relativa ao tamanho.

Embora apresentem vantagens como simplicidade de integração ao sistema, ausência de circuitos de recarga e estabilidade em longos períodos de armazenamento, as baterias não-recarregáveis possuem limitações importantes. A principal delas é o descarte inevitável após o uso, o que implica em custos recorrentes e impacto ambiental superior quando comparadas às recarregáveis. Além disso, a tensão fornecida por essas baterias tende a diminuir ao longo do tempo, o que pode afetar a operação de sistemas sensíveis à variação de tensão.

No desenvolvimento de sistemas embarcados alimentados por baterias primárias, é fundamental otimizar o consumo energético. Estratégias como o uso de modos de baixo consumo, a redução da frequência de operação e a ativação seletiva de periféricos ajudam a prolongar a autonomia e a minimizar a necessidade de substituição frequente das baterias. Além disso, o dimensionamento adequado da capacidade energética da bateria, em relação ao perfil de consumo do sistema, é essencial para garantir uma operação confiável e duradoura.

## Fonte de alimentação híbrida: energia de rede e baterias recarregáveis

Em sistemas embarcados e equipamentos eletrônicos que exigem alta confiabilidade, é comum a adoção de fontes de alimentação híbridas, que combinam a energia fornecida pela rede elétrica com uma bateria recarregável integrada. Essa abordagem permite que o dispositivo opere normalmente enquanto estiver conectado à tomada, mas mantenha a funcionalidade, total ou parcial, mesmo durante uma interrupção no fornecimento de energia.

Conhecida como alimentação com *battery backup* ou *uninterruptible power supply* (UPS), essa configuração é amplamente empregada em dispositivos cuja operação não pode ser interrompida abruptamente. A bateria atua como reserva e entra em funcionamento automaticamente quando a alimentação principal é interrompida. A capacidade dessa bateria pode variar conforme o propósito do sistema. Em casos simples, ela serve apenas para garantir que o sistema seja desligado de forma segura, salvando configurações e evitando perda de dados. Em aplicações mais críticas, a bateria possui maior capacidade e é capaz de manter o sistema em funcionamento pleno por um período determinado, permitindo continuidade operacional sem interrupções perceptíveis. Esse tipo de fonte híbrida é indispensável em equipamentos médico-hospitalares de uso crítico, como monitores de sinais vitais ou respiradores, que não podem sofrer quedas de energia sob nenhuma circunstância. Também é utilizado em dispositivos que, mesmo podendo parar momentaneamente, precisam retomar sua operação do ponto exato em que foram interrompidos, como sistemas de automação, roteadores e dispositivos de armazenamento de dados.

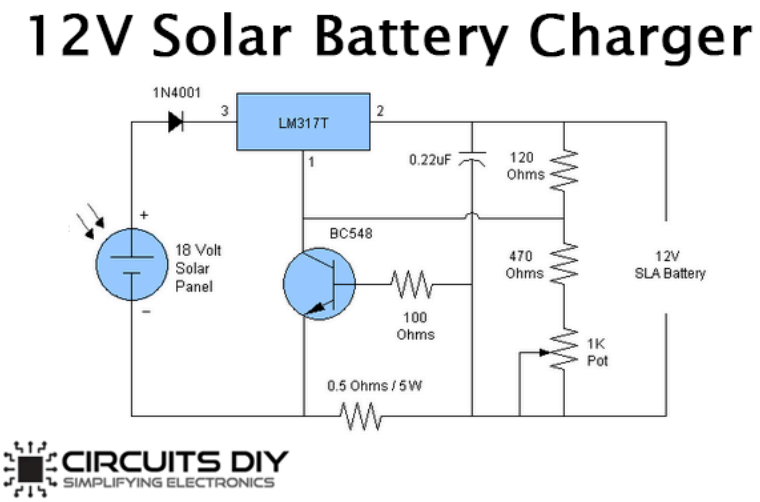
Em outra configuração híbrida, há dispositivos projetados para operar prioritariamente com bateria, como *smartphones* e *laptops*, mas que também incluem circuitos internos de recarga e gerenciamento energético. Nesses casos, a conexão à rede elétrica serve para manter a carga da bateria e garantir operação contínua durante longos períodos, enquanto a bateria garante mobilidade e autonomia.

O uso de fontes de alimentação híbridas traz vantagens como maior robustez, tolerância a falhas no fornecimento elétrico e flexibilidade de uso em contextos móveis ou instáveis. No entanto, requer

cuidados específicos no projeto, incluindo a implementação de circuitos de gerenciamento de carga, proteção contra sobrecarga e monitoramento do estado da bateria, além de lógica de comutação automática entre as fontes de energia. Esses elementos são fundamentais para garantir uma transição suave e segura entre os modos de alimentação, sem prejudicar o funcionamento do sistema embarcado.

## Fontes de alimentação alternativas

Em diversas aplicações de sistemas embarcados, a alimentação via rede elétrica ou baterias convencionais pode não ser viável ou suficiente. Isso é especialmente verdadeiro em cenários de difícil acesso, como áreas rurais, ambientes remotos ou em dispositivos portáteis de uso contínuo, como os dispositivos vestíveis. Nesses casos, o uso de fontes de energia alternativas tem se mostrado uma solução promissora. Fontes como a energia solar, a energia eólica, a piezoelétrica (obtida por vibração ou movimento), a energia térmica (proveniente da diferença de temperatura entre o corpo humano e o ambiente), entre outras, estão sendo exploradas como meios de suprir ou complementar a energia necessária ao funcionamento do sistema embarcado. A seguinte figura ilustra um exemplo de uso de energia solar como fonte principal de alimentação.



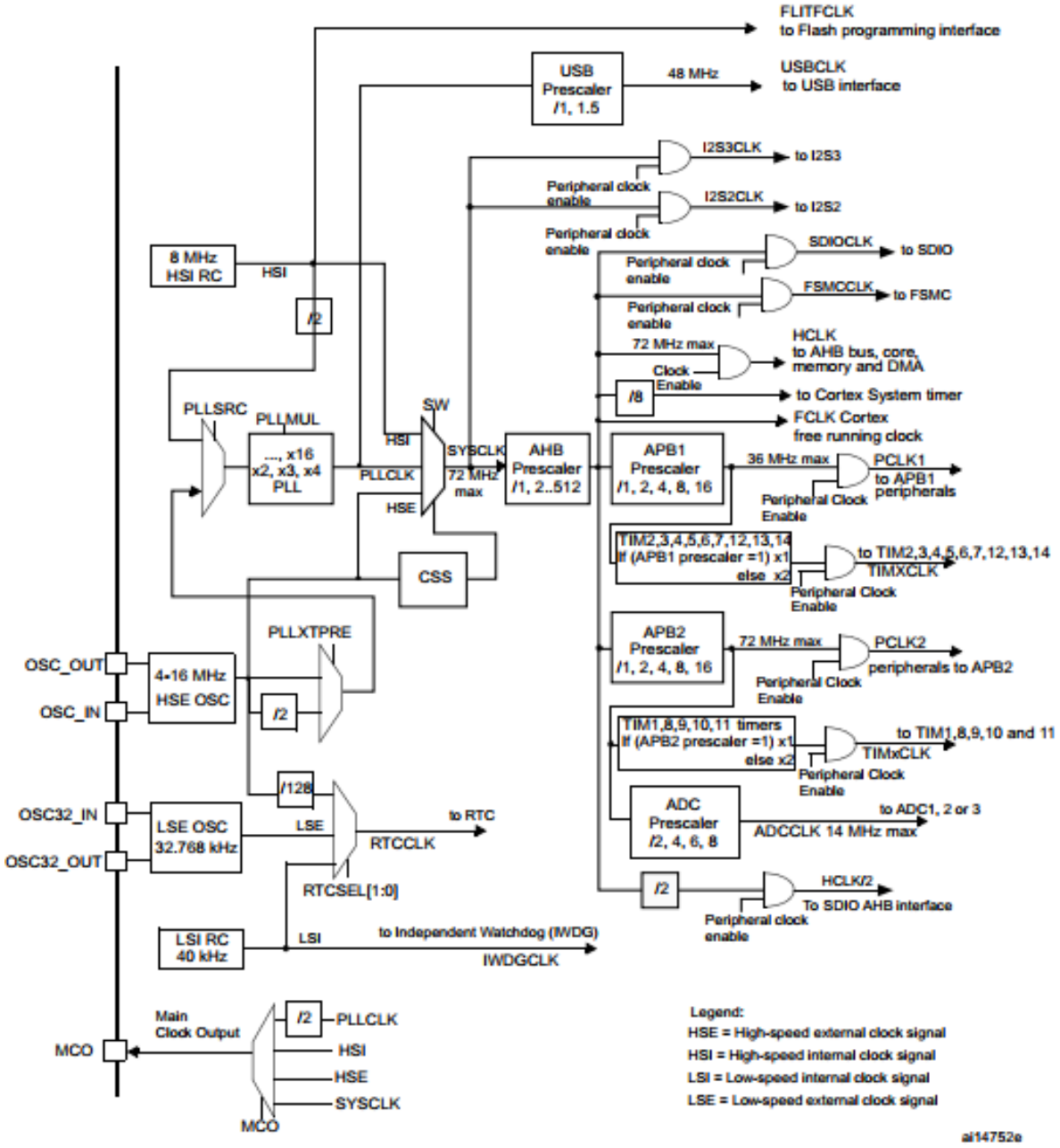

Carregador solar Esquemático do circuito de carregador de bateria

Apesar de ainda estarem em fase de pesquisa e desenvolvimento em muitas aplicações, essas fontes alternativas já vêm sendo utilizadas em sistemas de monitoramento remoto, como estações meteorológicas, sensores ambientais ou dispositivos médicos, que precisam operar por longos períodos sem manutenção e em locais onde a rede elétrica não está disponível. É importante destacar que, na maioria dos casos, essas fontes não são constantes nem previsíveis (por exemplo, variações na intensidade solar ou na presença de vento). Por esse motivo, é comum associá-las a um sistema de armazenamento de energia, como baterias recarregáveis ou supercapacitores. Esse armazenamento permite que a energia captada em momentos de abundância (como durante o dia, no caso da energia solar) seja utilizada em momentos de escassez, garantindo a operação contínua

do sistema. Assim, o dimensionamento e a escolha adequada de uma fonte de alimentação alternativa envolvem considerações tanto sobre a demanda energética do sistema quanto sobre o perfil de disponibilidade da fonte energética no ambiente de aplicação.

## Circuitos de *clock* configuráveis

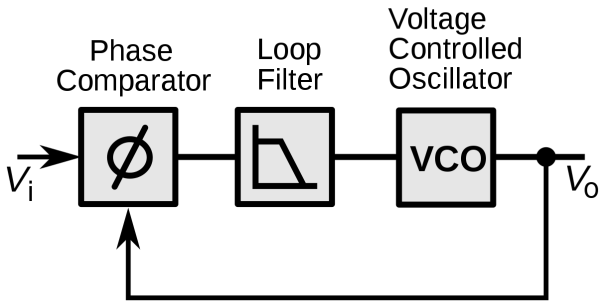
Microcontroladores modernos incorporam sofisticados módulos internos de gerenciamento e distribuição de sinais de relógio (ou sinais de *clock*) de múltiplas frequências para atender aos diversos requisitos de temporização de suas diversas unidades internas. Esse sistema de geração e distribuição de sinais de *clock* é conhecido como árvore de clock (em inglês *clock tree*) como ilustra a seguinte figura. A árvore de *clock* tem a função de produzir sinais derivados, estáveis e sincronizados a partir de uma frequência de referência, atendendo aos requisitos de operação dos diversos blocos funcionais do microcontrolador.




Fonte: Manual de referência de STM32F10.

Além da divisão de frequência, facilmente implementada por contadores digitais, microcontroladores utilizam técnicas mais sofisticadas para multiplicar a frequência do sinal de clock base. Essas técnicas são baseadas em sistemas de controle com realimentação, que ajustam dinamicamente a saída com base em uma referência, gerando sinais estáveis e com frequências superiores à de entrada:

- Malha Travada em Fase (em inglês, *Phase-Locked Loop* – PLL): O PLL é um sistema de controle projetado para sincronizar a fase do seu sinal de saída com a fase de um sinal de entrada de referência. Sua estrutura básica consiste em três blocos principais, como ilustra a figura: um Detector de Fase, que compara a fase dos sinais; um Oscilador Controlado por Tensão (VCO), que gera um sinal cuja frequência depende de uma tensão de controle; e um Filtro de *Loop*, que suaviza essa tensão de controle para evitar oscilações. O funcionamento do PLL baseia-se em um mecanismo de realimentação: o detector de fase monitora continuamente a diferença entre os sinais e ajusta a frequência do VCO até que suas fases coincidam, momento em que o sistema está "travado em fase". Para multiplicar a frequência, um divisor de frequência é inserido no caminho de realimentação (entre a saída do VCO e o detector de fase). Dessa forma, o VCO é forçado a oscilar em uma frequência que, após ser dividida, iguala a fase do sinal de referência. Por exemplo, se o divisor for 4, o VCO ajustará sua frequência para ser 4 vezes a da referência, garantindo que o sinal pós-divisão ainda tenha a mesma fase do sinal de entrada. Esse princípio é amplamente utilizado em microcontroladores para gerar *clocks* de sistema de alta frequência (como 32 MHz) a partir de osciladores de baixa frequência (como um cristal de 8 MHz), com alta estabilidade e baixo custo.




Fonte: Wikipedia.

- Malha Travada em Frequência (em inglês, *Frequency-Locked Loop* – FLL): O FLL é estruturalmente similar ao PLL, mas sua diferença crucial reside no elemento de comparação, que atua sobre a frequência dos sinais, e não sobre a fase. Em um FLL, um comparador de frequência mede a diferença entre a frequência do sinal gerado (que pode ser dividido antes da comparação) e a frequência do sinal de entrada. Essa diferença de frequência é então utilizada para ajustar um oscilador controlado até que as frequências coincidam, alcançando um estado "travado em frequência". Assim como no PLL, é possível adicionar um divisor no caminho de realimentação para que a frequência do sinal de saída seja um múltiplo da frequência de entrada. A distinção fundamental entre FLL e PLL é que o FLL garante apenas o alinhamento de frequência, não a coerência de fase. Essa característica pode ser aceitável ou até mesmo desejável em certas aplicações onde a sincronização de fase não é um requisito crítico.

Segue um quadro comparativo das características de PLL e FLL.

| Características | PLL | FLL |
|---|---|---|
| Alvo de controle | fase | frequência |
| Travamento | Em fase e, consequentemente, em frequência | Somente em frequência |
| Estabilidade dinâmica | Reage a variações de fase | Reage a variações de frequência |
| Complexidade | Geralmente maior (requer detecção e filtro de fase) | Levemente menor (detecção apenas de frequência) |
| Consumo de energia | Maior consumo | Menor consumo |
| Uso típico | Comunicações, geração precisa de *clock* | Aplicações com menos exigência de coerência de fase |

## Circuitos de oscilação

No entanto, os módulos de gerenciamento de sinais de *clock* em microcontroladores não operam isoladamente; eles dependem de uma fonte inicial de sinal oscilante, gerada por circuitos conhecidos como osciladores. Um oscilador é um circuito eletrônico que, a partir de uma fonte de corrente contínua, produz sinais periódicos (geralmente senoidais ou quadrados). Esses sinais são fundamentais e chamados de sinais de *clock*, pois estabelecem o ritmo e a ordem temporal das operações internas de todo o sistema digital.

Os osciladores podem ser internos ou externos ao microcontrolador e se dividem, principalmente, em:

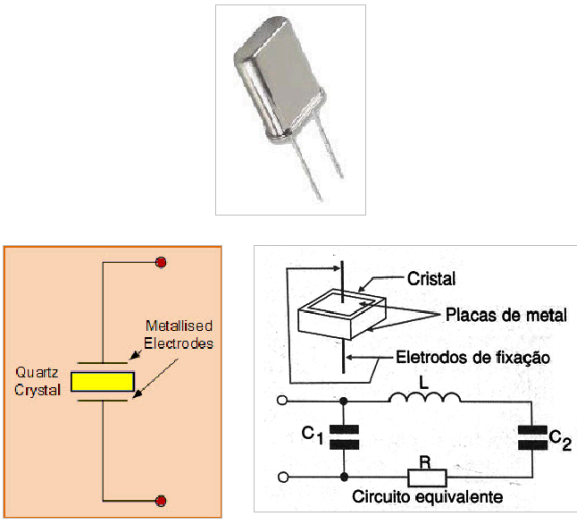
- Osciladores eletrônicos são circuitos que geram sinais periódicos, que operam como sinais de *clock,* a partir de uma fonte de corrente contínua (DC), sendo essenciais para estabelecer o ritmo de operação em sistemas digitais. Esses circuitos podem ser construídos com elementos passivos (como resistores, capacitores e indutores), ativos (como transistores), ou uma combinação de ambos. Alguns microcontroladores integram osciladores totalmente dentro do *chip*, o que traz vantagens como economia de espaço na placa de circuito impresso e liberação de pinos, por dispensarem componentes externos. No entanto, essa integração pode comprometer a estabilidade da frequência, especialmente sob variações de temperatura, devido à menor precisão dos componentes passivos implementados no silício. São comumente classificados em duas categorias principais: lineares e não-lineares. Os osciladores lineares, que produzem sinais senoidais, utilizam circuitos ressonantes para definir a frequência de oscilação. Exemplos incluem o oscilador em ponte de Wien (baseado em filtros RC), os osciladores Hartley, Colpitts e Clapp (baseados em circuitos LC). Por outro lado, os osciladores de relaxação são circuitos não lineares projetados para gerar formas de onda como sinais quadrados, triangulares ou dente-de-serra. Eles operam com base em ciclos de carga e descarga de capacitores e são caracterizados por sua simplicidade

e resposta rápida. Exemplos incluem o multivibrador astável, o oscilador de anel (ring) e o oscilador Royer.

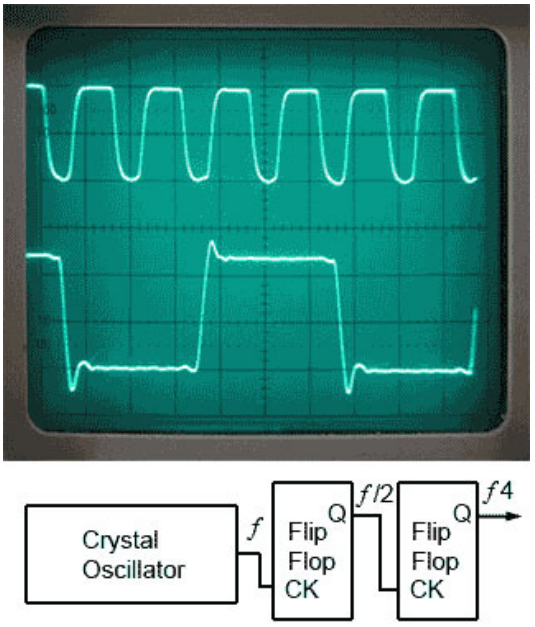
- Osciladores cerâmicos (ou ressonadores cerâmicos) são componentes eletrônicos que geram sinais de frequência suficientemente estáveis e precisos para a maioria dos circuitos digitais. São construídos com materiais piezoelétricos, como o titanato de bário ou o zircônio-titanato de chumbo (PZT), que possuem a propriedade de vibrar mecanicamente em resposta a uma tensão elétrica aplicada. Essas vibrações mecânicas, por sua vez, geram uma tensão oscilante, graças ao efeito piezoelétrico inverso, produzindo um sinal elétrico de frequência bem definida. Embora não alcancem a exatidão extrema dos osciladores a cristal, cuja tolerância pode ser da ordem de ±10 ppm, os osciladores cerâmicos oferecem estabilidade suficiente para muitas aplicações, com tolerâncias típicas em torno de ±0,5%. Isso os torna ideais para uso em microcontroladores, sistemas embarcados, relógios digitais, equipamentos de consumo e sistemas de comunicação que não exigem uma precisão de temporização rigorosa. Representam, assim, uma solução prática, econômica e confiável para aplicações que podem tolerar pequenas variações de frequência. Sua popularidade se deve ao baixo custo, tamanho compacto, robustez mecânica e facilidade de integração. Um exemplo comercial é o Murata CSTNE8M00GH5C000R0, um ressonador cerâmico de 8 MHz, com capacitores embutidos, ideal para uso direto com microcontroladores que incluem um circuito oscilador interno compatível.
- Osciladores de cristal são circuitos eletrônicos essenciais para gerar sinais de frequência altamente precisos e estáveis. Seu funcionamento se baseia no uso de um cristal piezoelétrico, geralmente de quartzo, que atua como um transdutor entre energia mecânica e elétrica. Quando submetido a um campo elétrico, o cristal vibra mecanicamente em sua frequência natural de ressonância. De forma reversa, essas vibrações mecânicas geram cargas elétricas. Esse comportamento bidirecional é consequência do efeito piezoelétrico direto e inverso, característico de certos materiais como o quartzo. Um cristal piezoelétrico é um sólido com estrutura cristalina altamente ordenada, cujas dimensões, forma, módulo de elasticidade e velocidade de propagação do som determinam sua frequência de ressonância. Cristais destinados a frequências mais altas são cortados em lâminas finas (circulares ou retangulares), enquanto frequências mais baixas utilizam cortes em forma de diapasão, como ilustra a seguinte figura.

Entre os materiais disponíveis, o quartzo é amplamente preferido devido à sua estabilidade térmica, baixo custo e alta disponibilidade. Eletricamente, o cristal pode ser modelado como um circuito RLC paralelo com uma frequência de ressonância extremamente bem definida e um fator de qualidade (Q) elevado, o que significa uma filtragem muito seletiva de frequências. Essa característica torna os osciladores de cristal ideais para aplicações em que precisão e estabilidade são críticas, como sistemas de comunicação de dados, relógios digitais, microcontroladores, processadores e temporizadores de alta exatidão. A frequência de operação típica de um cristal de quartzo varia de cerca de 1 kHz a 200 MHz, dependendo

do tipo de corte e da espessura do material. Em virtude dessas propriedades, o termo “oscilador de cristal” refere-se, por extensão, ao conjunto do circuito oscilador que emprega um cristal como elemento ressonante principal, dentro da ampla classe dos circuitos osciladores.



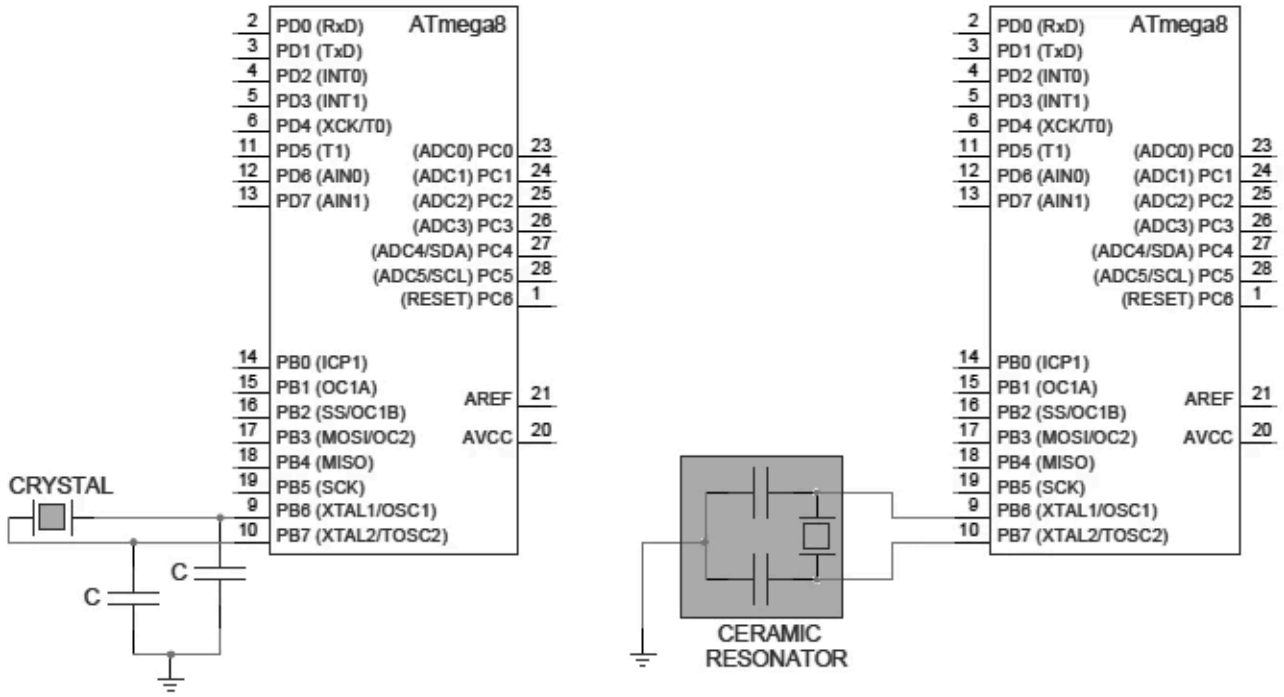

Ressonador de cristal




Ondas oscilantes geradas

## Estratégias de inicialização e configuração

Apesar de osciladores baseados em cristal ou cerâmica terem parte de sua eletrônica integrada ao microcontrolador, o componente oscilante (cristal ou cerâmica) ainda precisa ser conectado externamente, como mostra o seguinte diagrama.




Dada a possibilidade de que esses componentes externos não estejam conectados no momento da energização ou *reset*, muitos microcontroladores são projetados para ativar inicialmente um oscilador interno de relaxação. Nesse cenário, os pinos destinados ao cristal são configurados como GPIOs (General Purpose Input/Output). Cabe então ao *software* da aplicação configurar adequadamente o uso do cristal externo, caso necessário. Isso envolve redefinir a função dos pinos para uso do cristal, ativar o oscilador correspondente, aguardar a estabilização do sinal e, por fim, comutar a fonte do sinal de clock principal do sistema.

Além disso, alguns microcontroladores oferecem flexibilidade adicional, permitindo a utilização de um oscilador externo completo (um gerador de *clock* conectado diretamente) ou até mesmo a exportação do sinal de seu próprio oscilador (seja de relaxação ou cristal) para outros dispositivos, possibilitando que múltiplos microcontroladores compartilhem uma mesma fonte de *clock*.

## Aterramento

O aterramento, ou "terra", em circuitos eletrônicos é uma referência elétrica comum, definida por convenção como o ponto de potencial zero (0V). Ele serve de base para a medição de todas as tensões do sistema. Além disso, o aterramento atua como caminho para correntes, mas esta função varia entre sistemas de corrente contínua e alternada. Em sistemas de corrente contínua (DC), como os usados em circuitos eletrônicos, o terra é o caminho de retorno para as correntes. A corrente flui da fonte, passa pelas cargas e retorna ao polo negativo da fonte através do terra. Para simplificar o projeto, é comum conectar todos os componentes a um plano de terra (uma grande superfície de cobre na placa de circuito). Essa prática cria um caminho de retorno de baixa impedância, o que é ideal para a estabilidade do circuito, como ilustra a Figura (b). Em contraste, nos sistemas de corrente alternada (AC), como os da rede elétrica residencial, o retorno da corrente é feito pelo fio neutro. Nesse caso, o terra funciona como um caminho de segurança para desviar correntes de falha, protegendo contra choques elétricos, como mostrado na Figura (a).

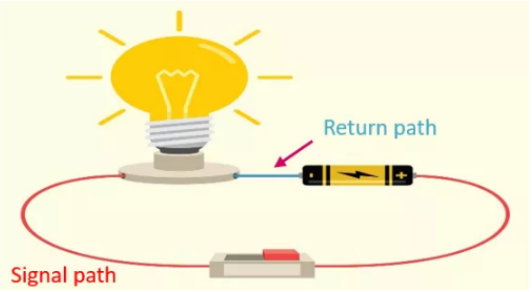
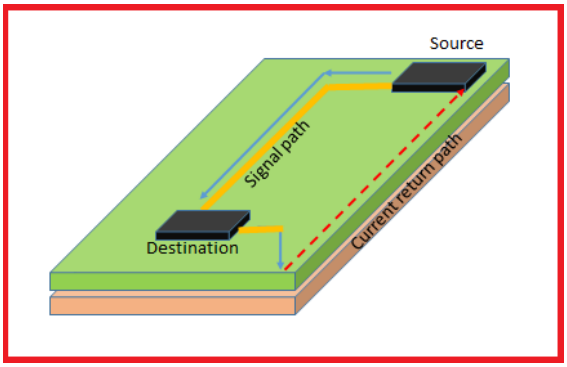



图1




(a) Fonte: Censtry. (b) Fonte: Sierra Circuits

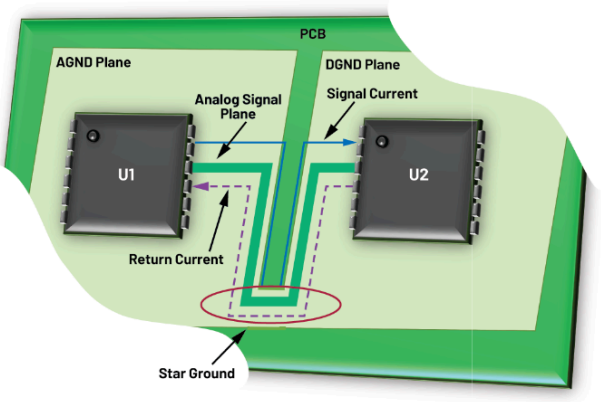
O aterramento é uma das partes mais críticas e frequentemente negligenciadas no projeto de circuitos eletrônicos. Projetar a topologia de aterramento com cuidado é essencial para garantir a confiabilidade e o desempenho de um sistema, especialmente em projetos embarcados que misturam sinais analógicos e digitais. Um sistema de aterramento bem projetado é fundamental para:

- reduzir ruídos e interferências eletromagnéticas (em inglês, *electromagnetic interference* – EMI), que, se não controlados, podem causar leituras imprecisas em sensores e falhas de comunicação;
- garantir a estabilidade e o bom funcionamento dos circuitos, como conversores A/D, prevenindo problemas de oscilação e mau funcionamento de sinais digitais e analógicos;
- proteger componentes sensíveis contra sobretensões e descargas eletrostáticas (em inglês, *electrostatic discharges* – ESD), que podem danificar permanentemente os componentes; e
- evitar laços de terra (ground loops) e correntes indesejadas em sistemas complexos.

Em projetos modernos, especialmente sistemas embarcados, é comum separar o aterramento em dois domínios distintos para evitar que os ruídos gerados pelos circuitos digitais contaminem os sinais analógicos, que normalmente são de baixa amplitude e muito mais sensíveis a interferências. Essa distinção ajuda a preservar a integridade dos sinais e garantir a precisão dos sistemas que misturam componentes digitais e analógicos:

- Terra Digital (em inglês, *Digital Ground* – DGND): Serve como referência para circuitos digitais (microcontroladores, memórias, etc.). É um domínio com correntes de chaveamento rápidas e ruído de alta frequência.
- Terra Analógico (em inglês, Analog Ground – AGND): É a referência para circuitos sensíveis (sensores, conversores A/D e D/A). Este aterramento precisa ser o mais “limpo” possível, livre de ruídos e oscilações para garantir a precisão.

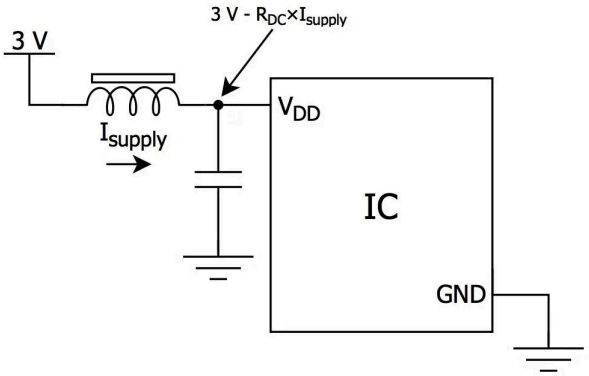
Para facilitar o projeto e garantir a estabilidade do circuito, é comum desenhar dois planos de terra separados, plano DGND, onde todos os componentes digitais são conectados, e o plano AGND usado para conectar todos os componentes analógicos. Essa separação cria caminhos de retorno de baixa impedância para cada tipo de sinal, o que ajuda a evitar que o ruído gerado pelos circuitos digitais interfira nos componentes analógicos mais sensíveis.




Fonte: Chatgpt.

Apesar da separação física ou lógica, os planos de terra digital (DGND) e analógico (AGND) devem ser conectados em um único ponto, conhecido como ponto estrela (em inglês, *star ground* ou *single-point ground*). Essa conexão é geralmente feita perto do conversor A/D ou da fonte de alimentação. O objetivo é evitar laços de terra (*ground loops*) que tendem a se formar e garantir uma referência comum entre os dois domínios, o que é essencial para o correto funcionamento de circuitos mistos.

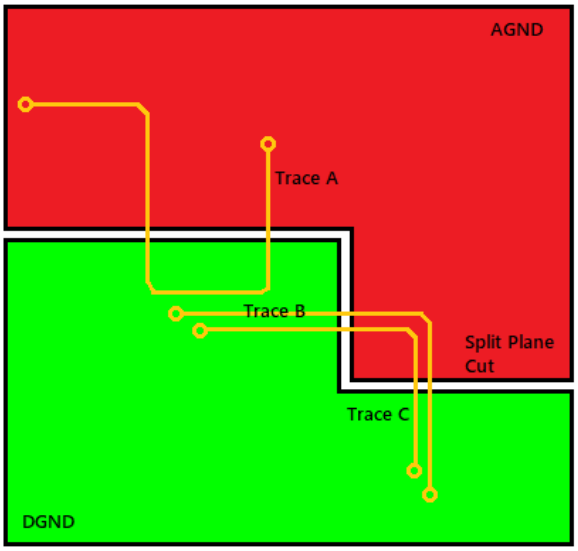
O aterramento isolado com ferrites é uma variação do ponto estrela, onde a conexão entre os terras digital (DGND) e analógico (AGND) é feita através de um componente chamado conta de ferrite (em inglês, *ferrite bead*), geralmente posicionado perto do conversor A/D ou da fonte de alimentação, ao lado de um capacitor. Essa topologia é particularmente útil para atenuar o ruído de alta frequência, uma vez que o ferrite é um componente indutivo que atua como um filtro, bloqueando essas interferências geradas pelos circuitos digitais e impedindo que elas cheguem ao domínio analógico, mais sensível. Ao mesmo tempo, o ferrite permite a passagem de correntes de baixa frequência (DC), garantindo uma referência de potencial comum e estável para todo o sistema, o que é fundamental para o funcionamento preciso de circuitos mistos.




Fonte: All About Circuits.

A topologia de aterramento em malha (*mesh ground*) é outra abordagem comum, especialmente em placas de alta densidade onde o espaço é crítico. Nela, todos os pontos de terra são interconectados por uma rede de trilhas (em inglês, *traces*) que formam uma malha, criando múltiplos caminhos de retorno para a corrente. Embora essa configuração resulte em uma impedância de terra muito baixa,

o que é ótimo para estabilidade e dissipação de calor, mas aumenta o risco de formação de laços de terra (*ground loops*). Por isso, o aterramento em malha é raramente usado em sistemas que combinam sinais analógicos e digitais, pois a malha facilita a circulação de correntes de ruído, comprometendo a integridade dos sinais analógicos.




Fonte: Chatgpt.

# BITDOGLAB

Para compreender como os conceitos de circuitos de interface, alimentação e oscilação se materializam em um projeto real, vamos explorar a placa de desenvolvimento **BitDogLab**. Ela serve como um excelente exemplo prático de como essas considerações são aplicadas para garantir a funcionalidade e robustez de um sistema embarcado.

No desenvolvimento de qualquer placa eletrônica, a compatibilização entre diferentes componentes é um desafio central. Essa compatibilização não é apenas elétrica, mas também mecânica e temporal:

- Compatibilização Elétrica: Refere-se a garantir que as tensões, correntes e impedâncias dos sinais sejam adequadas entre os componentes. Isso impede danos, assegura a correta interpretação dos níveis lógicos (HIGH/LOW) e otimiza a transferência de energia.
- Compatibilização Mecânica: Envolve o encaixe físico e o posicionamento correto dos componentes, o tamanho da placa, a disposição dos conectores e a dissipação de calor.
- Compatibilização Temporal: Diz respeito à sincronização dos sinais e à garantia de que os eventos ocorram na sequência e no tempo corretos para a operação do sistema. Isso inclui tempos de subida e descida de sinais, atrasos de propagação e frequências de operação.

Todos esses fatores impactam diretamente o projeto de *hardware* e *software*. No *hardware*, a escolha de componentes, o *layout* da placa e a implementação dos circuitos de interface são guiados por essas necessidades de compatibilização. No *software*, o código precisa considerar os tempos de resposta dos periféricos, os atrasos de comunicação e as sequências de inicialização para garantir a correta interação.

Na BitDogLab, há diferentes abordagens para conectar periféricos. Alguns periféricos de entrada e saída (I/O), que já são elétrica e mecanicamente compatíveis com o microcontrolador, operando na mesma faixa de tensão e podendo ser conectados diretamente, são ligados diretamente aos seus pinos. Em contraste, outros componentes, seja por operarem em níveis de tensão diferentes, exigirem mais corrente, necessitarem de isolamento ou precisarem de um condicionamento de sinal específico, dependem de circuitos de compatibilização externos. Existe ainda uma terceira categoria de periféricos que se beneficiam dos recursos de interface integrados nos pinos GPIO do próprio microcontrolador. Funções como resistores de *pull-up* ou *pull-down* que garantem que um pino tenha um estado lógico definido quando não está sendo ativamente acionado, são frequentemente configuráveis diretamente via *software* no microcontrolador. Essa integração simplifica o projeto, reduzindo a necessidade de componentes externos em muitos casos.

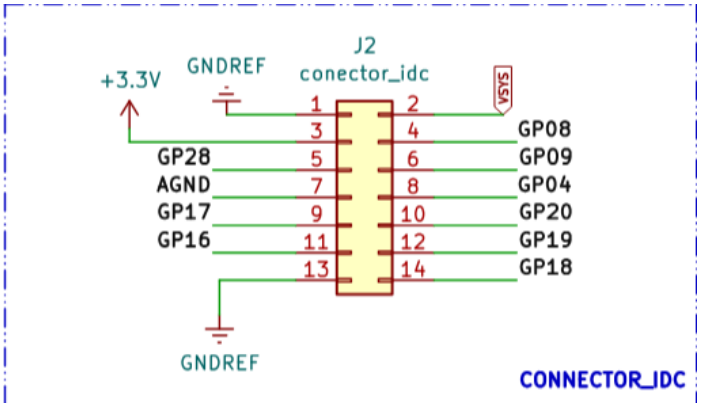
A compatibilidade temporal, embora muitas vezes resolvida com *hardware* especializado, em muitos casos, ocorre também a nível de programação (*software*). Por exemplo, atrasos (*delays*) são inseridos no código para aguardar a estabilização de um sensor ou a resposta de um módulo externo. Ou ainda, rotinas de *polling* ou interrupções são implementadas para gerenciar a temporização da comunicação com periféricos. Um exemplo clássico é o *debouncing* de botões, onde o *software* espera um breve período após a detecção inicial de um clique para confirmar que o botão foi realmente pressionado e não é apenas um ruído mecânico.

Ao longo desta seção, analisaremos o esquemático da versão 6.0 da BitDogLab e o esquemático do Raspberry Pi Pico para entender como essas diferentes camadas de compatibilização são implementadas, transformando a teoria em um dispositivo funcional e robusto.

## Periféricos integralmente compatíveis

Na BitDogLab, alguns componentes são conectados diretamente aos pinos do microcontrolador, sem exigir nenhum *hardware* adicional. São eles:

- **Conector de expansão de 14 pinos[2]:** Este conector permite a ligação direta de módulos ou outros dispositivos que sejam compatíveis com a tensão de operação e os níveis lógicos do microcontrolador.

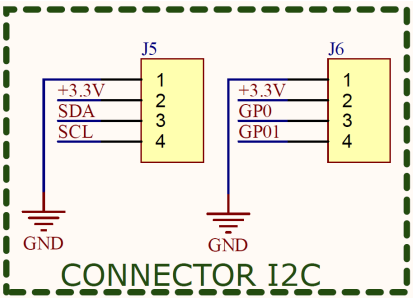



- **Conectores de jacaré:** Oferecem uma forma prática e rápida de conectar fios e realizar experimentos simples, ligando diretamente a sinais digitais ou analógicos compatíveis. Os

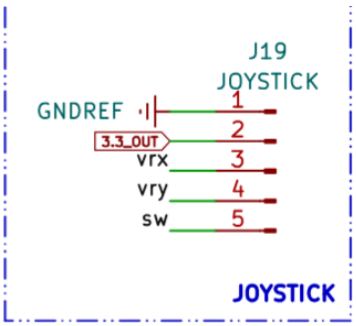
[2] Na versão 7.0, os pinos AGND e GNDREF passaram a ser conectados com GP15 e GP14, respectivamente.

eletrodos DIG 0, 1, 2 e 3 da BitDogLab estão conectados aos GP 0, 1, 2 e 3, respectivamente.


- **Conectores I2C**[3]: Na extremidade superior da placa, há dois conectores de 4 pinos que facilitam a comunicação com periféricos externos. Localizado no lado direito, o conector I2C0 (J6) dá acesso aos pinos GP0 (SDA) e GP1 (SCL), além de 3V3 e GND. Embora seja nomeado como I2C0 para a comunicação I2C, ele também pode ser usado para comunicação UART, dando acesso aos pinos TX ou RX. Isso é útil para conectar módulos como um GPS, por exemplo. Posicionado no lado esquerdo, o conector I2C1 (J5) acessa os pinos GP2 (SDA) e GP3 (SCL), juntamente com 3V3 e GND. Ele é nomeado de I2C1 para a comunicação I2C.



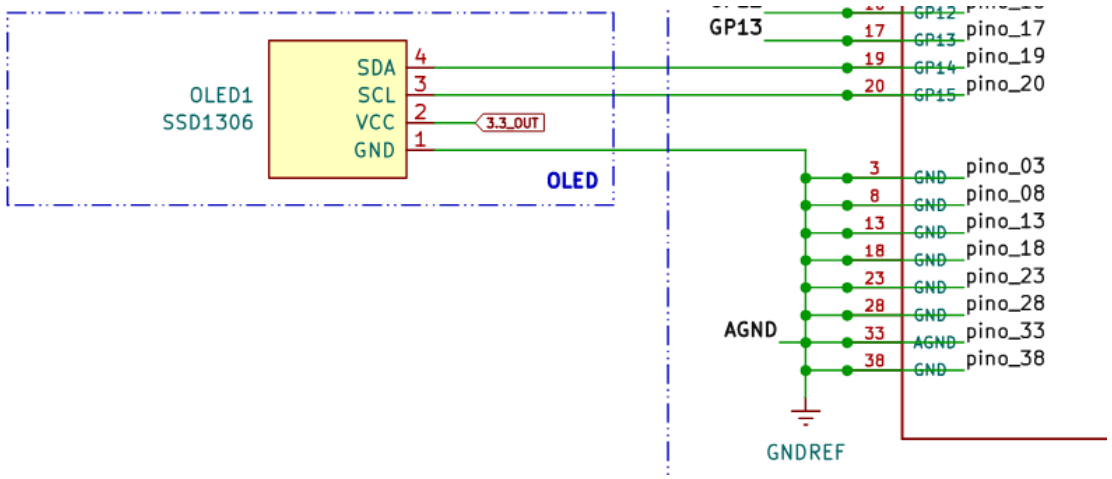

- ***Joystick*:** O *joystick*, sendo um sensor que gera saídas de sinais analógicos, tem seus pinos conectados diretamente. Para ler as entradas do *joystick*, os pinos de sinal analógico são conectados diretamente ao Pico. A saída VRy está ligada ao GPIO26 e a VRx ao GPIO27. Já o botão SW do joystick é conectado ao GPIO22, enquanto o outro terminal do botão é aterrado (GND). Para que o botão funcione corretamente, o GPIO22 deve ser configurado com o resistor de pull-up interno. Para a leitura dos sinais analógicos, presumimos que as saídas do joystick já estão adequadamente condicionadas — ou seja, dentro da faixa de tensão e com a impedância correta para uma leitura precisa pelo conversor ADC do microcontrolador.




[3] Foram utilizadas as referências J5 e J6 do esquema da versão 7.0 da BitDogLab, pois o desenho do circuito desta versão estava disponível para consulta. Na versão 5.0, esses conectores tem uma numeração ou posição diferente.

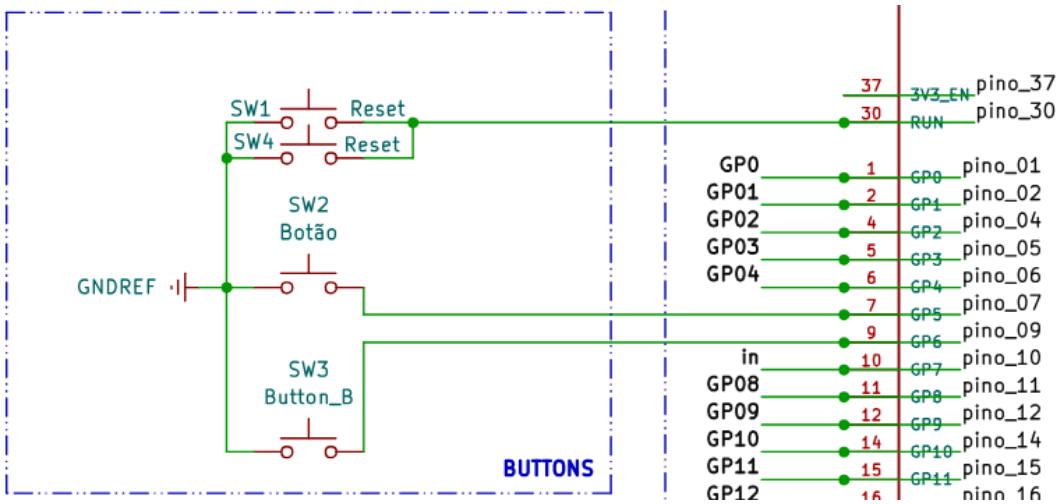
## Periféricos com circuitos de interface internos

Na BitDogLab, há periféricos que, embora não conectados diretamente de forma “bruta” aos pinos do microcontrolador, utilizam os recursos internos do próprio Raspberry Pi Pico para sua compatibilização. É o caso do ***display* OLED**, que se comunica com o microcontrolador via interface I2C. Essa interface, por natureza, requer resistores de *pull-up* (internos) nas suas linhas de dados (SDA) e *clock* (SCL) para garantir que os sinais estejam em um estado lógico definido quando inativos[4]. Apesar da comunicação simplificada pelo I2C, a programação do *display* é um pouco mais complexa: cada texto ou imagem exibido é, na verdade, um aglomerado de *pixels*. Para que uma imagem apareça na tela, cada um desses *pixels* deve ser ativado ou desativado (setado ou resetado) manualmente via código. Essa tarefa, que seria tediosa e susceptível a erros se feita do zero, é facilitada por programas de terceiros, como o código de exemplo disponível no repositório `pico-examples` do Pico, ssd1306_i2c, o que acelera o desenvolvimento e evita a necessidade de programar cada *pixel* individualmente.




Da mesma forma, os ***push-buttons*** (botões)[5] na placa são configurados como ativo-baixo, o que significa que eles enviam um sinal baixo (GND) quando pressionados e, para estarem em um estado alto quando não pressionados, também necessitam de resistores de *pull-up*.

[4] Na versão 7.0, os pinos SCL e SDA são conectados aos GP3 e GP2, respectivamente.
[5] Na versão 7.0, um novo botão foi adicionado. Ele está conectado ao GPIO 10, pino que era usado pelo segundo *buzzer* na versão 6.0.




No entanto, em ambos os casos, não são observados resistores *pull-up* externos nos circuitos de interface. Isso porque a placa BitDogLab aproveita uma característica fundamental dos pinos do Raspberry Pi Pico: a capacidade de configurar resistores de *pull-up* internos diretamente via programação. Essa funcionalidade elimina a necessidade de adicionar componentes físicos externos entre os periféricos e o microcontrolador, simplificando significativamente o *hardware*. Contudo, é importante lembrar que, para o funcionamento correto desses periféricos, a ativação desses resistores de *pull-up* internos deve ser explicitamente realizada no código do projeto.

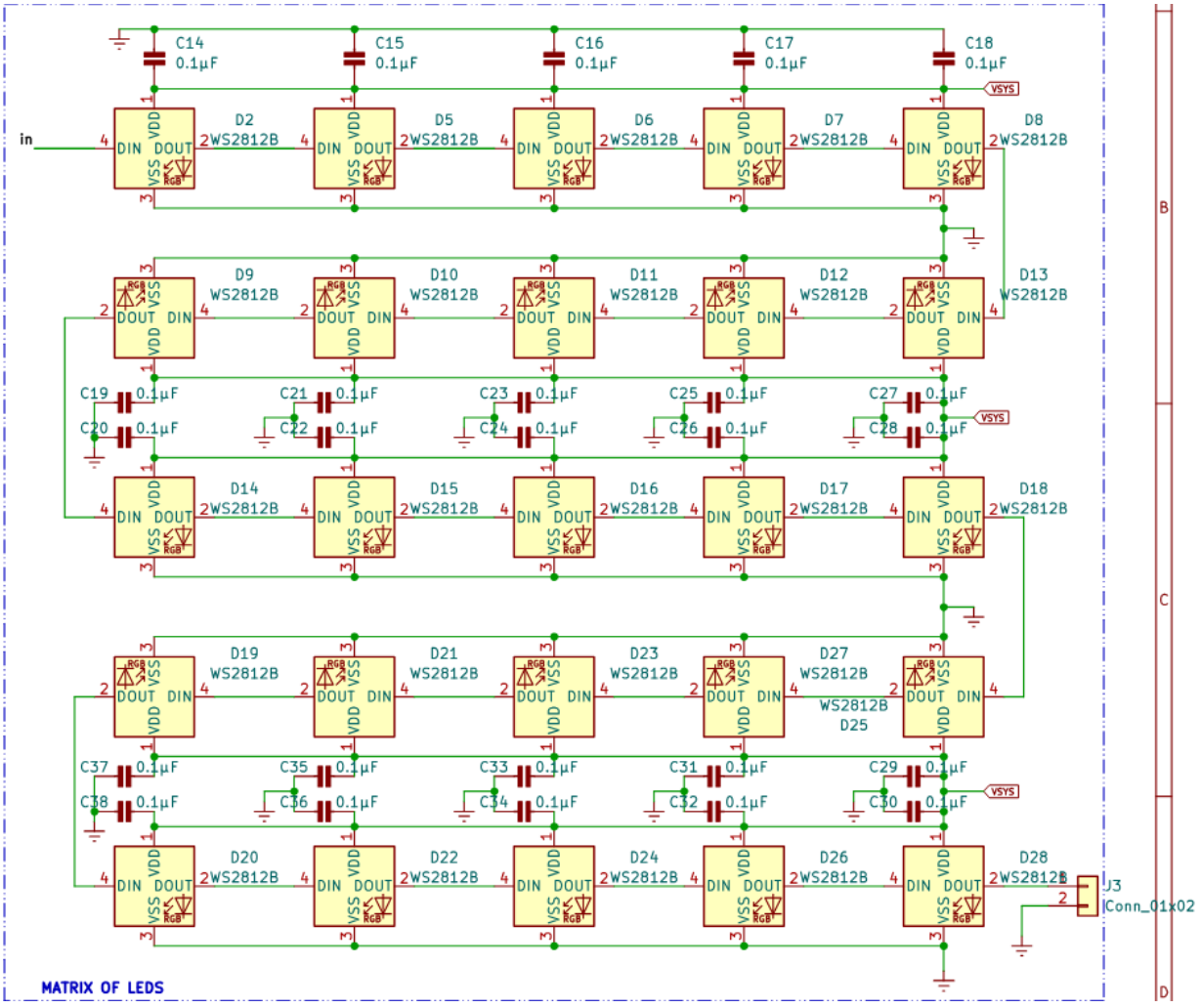
## Periféricos com circuitos de interface externos

Para garantir o funcionamento adequado e a proteção dos componentes em um projeto com microcontroladores como o Raspberry Pi Pico, a utilização de componentes discretos externos é frequentemente indispensável, mesmo quando há recursos internos disponíveis. Esses componentes são para adaptar níveis de tensão, fornecer filtragem de ruído, controlar correntes e isolar seções do circuito, preenchendo lacunas que o microcontrolador por si só não consegue suprir de forma eficiente ou segura.

A seguir, detalharemos a necessidade desses componentes em algumas conexões específicas.

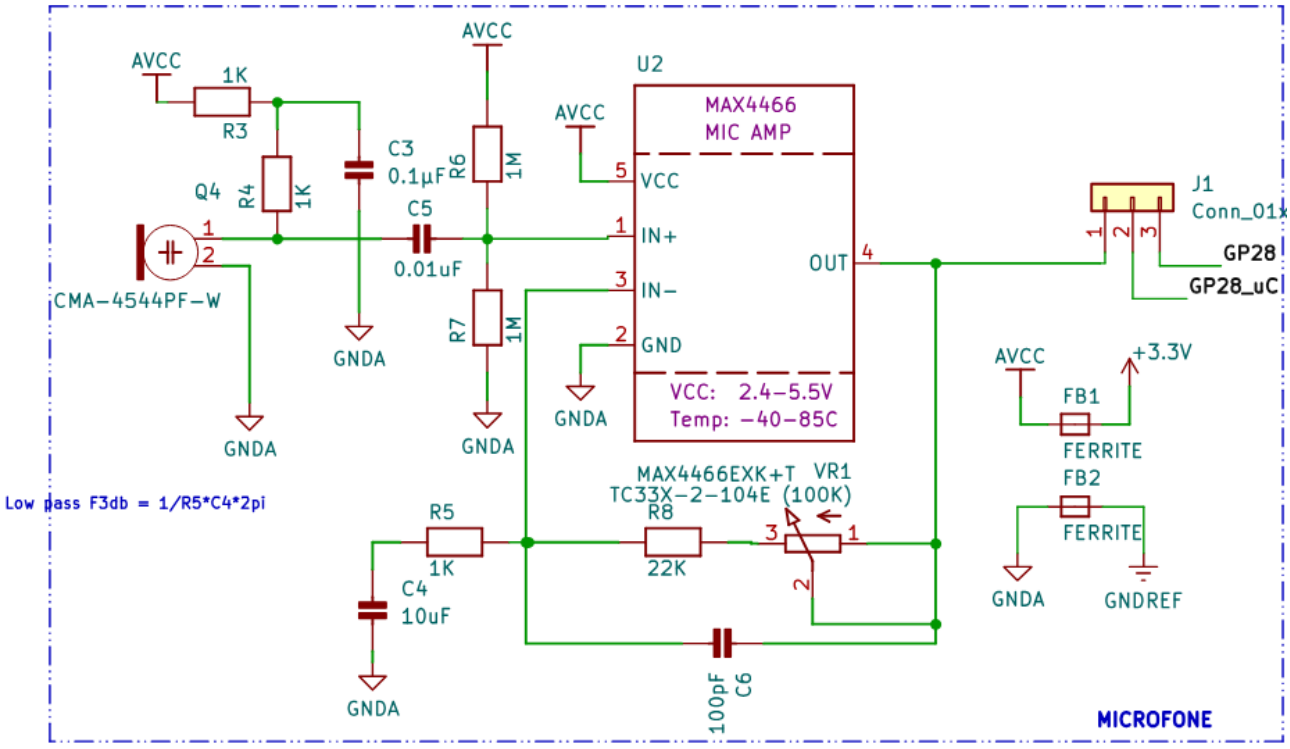
- **Matriz de LEDs WS2812B 5x5**: As matrizes de LEDs WS2812B, conhecidas como *NeoPixels*, são popularmente usadas devido à sua capacidade de controle individual de cada LED. Cada LED nessa matriz possui um *chip* controlador embutido que se comunica digitalmente. No entanto, esses LEDs são sensíveis a flutuações na linha de alimentação (VCC e GND) e podem apresentar comportamento errático ou piscar de forma indesejada se a energia não for “limpa”. O microcontrolador não possui filtragem interna em seus pinos de alimentação capaz de lidar com a demanda dinâmica de corrente desses LEDs. Uma solução é a filtragem de alimentação em cada LED. Um capacitor de 0.1uF conectado entre o VCC e o GND de cada LED WS2812B atua como um capacitor de desacoplamento (ou *bypass*). Ele serve como um pequeno “reservatório” de energia, fornecendo corrente instantânea quando um LED muda de cor ou brilho rapidamente. Mais importante, ele filtra ruídos de alta frequência que podem vir da fonte de alimentação ou ser gerados pelo próprio chaveamento rápido dos LEDs. Sem esse capacitor, o ruído na linha de alimentação pode corromper os dados digitais que chegam aos LEDs, causando falhas visuais. Para a programação, a complexidade é maior, pois o controle dos *NeoPixels* exige a manipulação

de diversos registradores para gerar o *timing* preciso do protocolo de comunicação. Para simplificar o desenvolvimento, a prática comum é utilizar programas de terceiros, que já contêm as funções necessárias para controlar os LEDs de forma fácil e confiável. O pino "in" está conectado ao GPIO07. No final da sequência de 25 leds desta matriz há um conector dando acesso ao pino de dados (Dout, 5V e GND). Se o usuário precisar conectar mais LEDs coloridos é só usar o jumper J3[6].

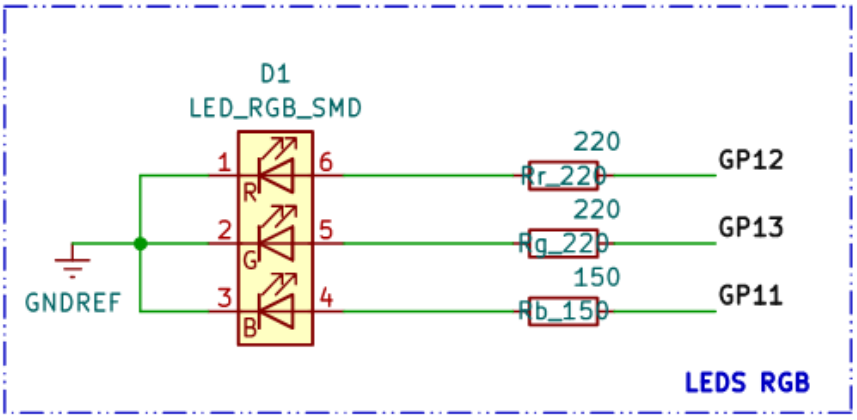



- **Microfone CMA-4544PF-W**: O microfone CMA-4544PF-W é um microfone de eletreto, um tipo de sensor que produz um sinal analógico muito pequeno, geralmente na ordem de milivolts. Esse sinal é fraco demais para ser diretamente lido com precisão pelo ADC do Pico, que opera em uma faixa de Volts. Como o microcontrolador não possui um amplificador analógico de instrumentação ou de áudio interno com ganho programável e baixo ruído capaz de condicionar o sinal do microfone adequadamente, foi usado o amplificador operacional de áudio dedicado MAX4466EXK. Esse op-amp atua como um pré-amplificador, elevando o sinal de áudio de baixo nível do microfone para uma faixa de tensão entre 0V e 3.3V que o ADC do Pico consegue digitalizar. Além de amplificar, o MAX4466EXK é projetado para ter baixo ruído, garantindo que o sinal de áudio amplificado mantenha sua fidelidade.

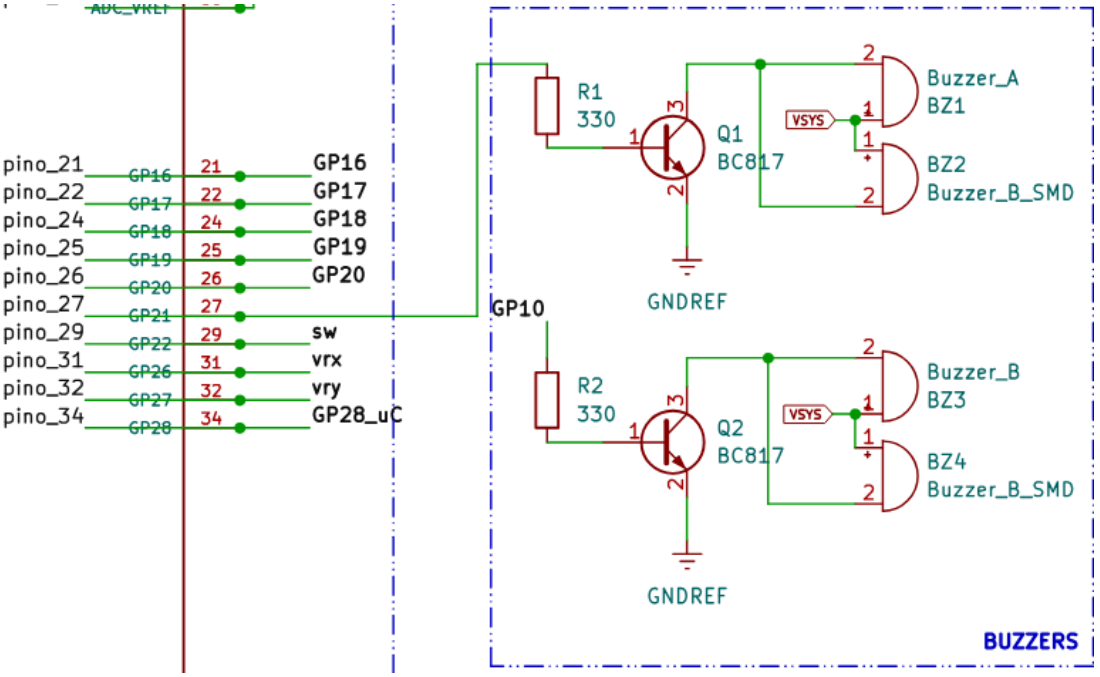
[6] Na versão 7.0, deve-se usar o jumper J8.




- **LED RGB**: Um LED RGB é composto por três pequenos LEDs (vermelho, verde e azul) em um único encapsulamento. Cada cor tem uma tensão direta de operação (Vf) e uma corrente de operação ideal. Conectar um LED diretamente a um pino GPIO do microcontrolador sem um resistor resultaria em uma corrente excessiva fluindo através do LED, o que o danificaria permanentemente e poderia até mesmo danificar o pino do microcontrolador. O microcontrolador não pode limitar a corrente de saída de seus pinos a um valor preciso para um LED; ele apenas fornece um nível de tensão. Os resistores de 220 Ohms conectados em série com cada cor do LED RGB (e, portanto, a cada pino GPIO, GP12, GP13 e GP11) são resistores limitadores de corrente. Eles criam uma queda de tensão que limita a corrente que flui através do LED para um valor seguro e desejável (geralmente entre 10mA e 20mA por cor). Sem esses resistores discretos, a vida útil do LED seria drasticamente reduzida ou ele queimaria instantaneamente.



- ***Buzzers*[7]*:*** Um *buzzer* (especialmente os passivos) é um componente que gera som ao ser energizado com uma forma de onda. Ele possui uma natureza indutiva, o que significa que, quando a corrente que o atravessa é rapidamente desligada, ele gera um pico de tensão reverso (tensão indutiva) que pode ser muito maior do que a tensão de operação normal do microcontrolador. Além disso, muitos *buzzers* requerem correntes maiores do que um pino GPIO típico de um microcontrolador pode fornecer diretamente. Os transistores BC817 (que são transistores NPN) atuam como chaves eletrônicas que ligam e desligam a corrente para os *buzzers*. O resistor de 330Ω conectado à base de cada transistor (e ao pino GPIO do Pico, GP10 e GP21) é um resistor limitador de corrente de base. Ele garante que uma corrente controlada e segura seja aplicada à base do transistor para ligá-lo (*saturá-lo*), protegendo o pino do microcontrolador de sobrecorrente. O transistor, por sua vez, permite que uma corrente muito maior flua do terra para o *buzzer*, proveniente da fonte de alimentação principal VSYS (e não do pino do microcontrolador), superando a limitação de corrente do GPIO. Mais importante, o transistor isola o microcontrolador do pico de tensão indutiva gerado pelo *buzzer* quando ele é desligado, pois a carga indutiva está no coletor do transistor, e não diretamente no pino GPIO.

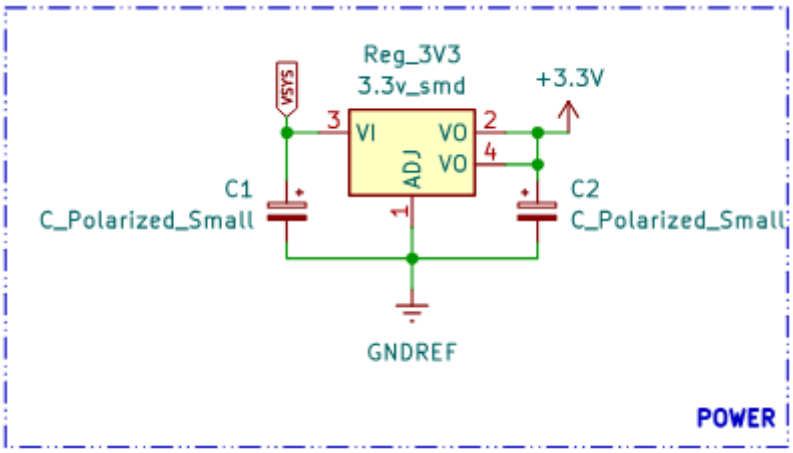



## Circuito de alimentação

A seguir, seguem os detalhes dos circuitos de alimentação na BitDogLab. A alimentação primária da placa são os 5V fornecidos pela própria Raspberry Pi Pico quando seu conector USB-C é conectado em uma fonte com esta tensão. Os 5V ficam disponíveis em 2 pinos: VBUS e VSYS. A diferença entre os dois pinos é que VSYS pode ser usado para receber 5V de outra fonte, e neste caso VBUS não terá os 5V disponíveis. Os periféricos alimentados com 5V (por exemplo, a matriz de LEDs) são ligados em VSYS.

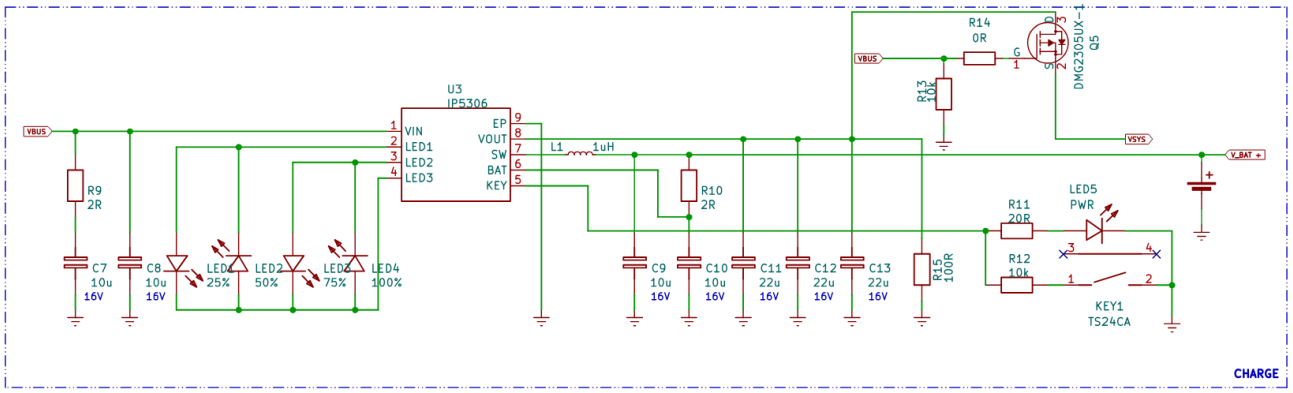
[7] Na versão 7.0, há apenaa um buzzer (Bz1) conectado no GP21.

Os periféricos alimentados com 3.3V (por exemplo, o *display* OLED) recebem esta tensão de um regulador de tensão que recebe a tensão em VSYS e disponibiliza os 3.3V.

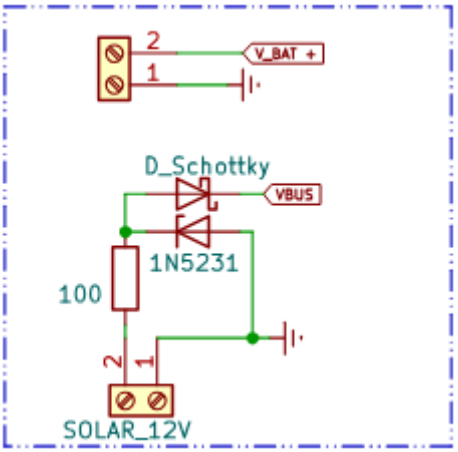



A BitDogLab conta ainda com um  módulo de controle de carga e descarga de bateria Li-Fe do tipo 18650. Este módulo é composto por um circuito integrado IP5306 e componentes auxiliares. Ele recebe a tensão em VBUS como entrada (VIN) para carregar a bateria, enquanto a USB-C está conectada. Nesta situação, a mesma tensão em VBUS mantém o MOSFET Q5 (canal-P) em corte, para que a tensão VOUT do IP5306 não seja ligada em VSYS (pois VSYS recebe a tensão diretamente da USB-C). Quando a USB-C é desconectada, o módulo converte a tensão da bateria Li-Fe para 5V e disponibiliza esta tensão em seu pino VOUT. Sem a tensão VBUS, o MOSFET passa a conduzir, conectando VOUT a VSYS da Pico. Assim, a BitDogLab passa a operar com bateria.

O IP5306 é um *chip* de gerenciamento de energia que atua tanto como carregador de bateria quanto como conversor de tensão. O pino VIN é a entrada de energia para o *chip* quando ele está sendo alimentado pela fonte VBUS. É por aqui que a energia entra para carregar a bateria. O pino BAT é o ponto de conexão direto com a bateria. Ele é o pino de entrada e saída principal da bateria. A energia da fonte externa chega ao *chip* pelo pino VIN e é transferida para a bateria pelo pino BAT durante o carregamento. Da mesma forma, quando a bateria está alimentando o circuito, a energia sai dela pelo pino BAT. O pino SW é um dos pinos de controle do conversor de tensão (também chamado de “chaveador”). Esse circuito interno pega a baixa tensão da bateria (pino BAT) e a converte em uma tensão de 5V. A energia de 5V é então disponibilizada na saída VOUT. Por fim, o pino KEY é a interface de controle para o usuário. Ele funciona como o botão de liga/desliga, ativando ou desativando o conversor de tensão, que fornece energia para o pino VOUT. O filtro de ferrite colocado entre a bateria e o pino SW é uma boa prática. Ele garante que a energia que chega ao *chip* seja o mais limpa possível, protegendo o circuito de ruídos de alta frequência.




Há ainda conectores para fontes alternativas[8]: um conector para baterias Li-Fe externas (V_BAT+) e um conector (SOLAR_12V) para um painel solar. É bom ressaltar que este recurso nunca foi utilizado. A rigor, VBUS precisa receber 5V, então o nome "SOLAR_12V" não é muito adequado.

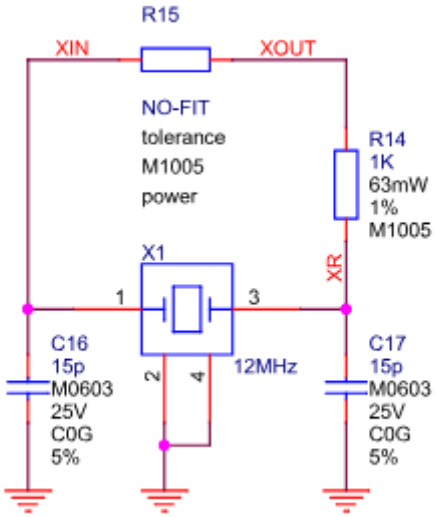



## Circuito de oscilação

O Raspberry Pi Pico utiliza um circuito de oscilação externo para gerar seu sinal de *clock* principal, usado pelo microcontrolador RP2040. A peça-chave do circuito é um cristal de 12 MHz. Este componente passivo, quando estimulado eletricamente, ressoa a uma frequência muito precisa e estável. Para que o cristal oscile corretamente e para sintonizar a frequência exata, são utilizados dois capacitores de 1 pF, conectados do cristal para o terra. Esses capacitores de carga (*load capacitors*) garantem que o cristal oscile na frequência desejada e ajustam a capacitância total do circuito oscilador.

Os resistores R14 (1 kΩ) e R15 (o valor de R15 é omitido, mas geralmente é um resistor de polarização ou parte de um divisor de tensão ou rede de feedback para o amplificador inversor interno do microcontrolador) desempenham papéis de polarização e feedback. O RP2040 possui um amplificador inversor interno que, junto com o cristal e os capacitores, forma um oscilador Pierce. R14 e R15 ajudam a configurar o ponto de operação desse amplificador e a fornecer a realimentação necessária para sustentar as oscilações, garantindo que o circuito comece e mantenha a oscilação de forma estável.

[8] Na versão 7.0, foram removidos os conectores para fontes externaas, SOLAR_12V e V_BAT+.




# PROJETOS DE PCB

O desenvolvimento de um sistema eletrônico envolve diversas etapas que vão desde a concepção teórica até a materialização física do circuito. Entre essas etapas, o projeto da placa de circuito impresso (PCB) ocupa um papel central, pois é ele que transforma um conjunto de conexões elétricas abstratas em um dispositivo concreto, pronto para fabricação e testes. A qualidade do projeto de PCB impacta diretamente o desempenho, a confiabilidade e até o custo final do produto. Para garantir um bom projeto, começa-se pela representação lógica e visual do circuito. É uma etapa que precede o desenho da própria placa.

Em um projeto de *hardware*, a representação visual dos circuitos é tão importante quanto a escrita do código em um projeto de *software*. É nesse ponto que entram os editores de esquemáticos, ferramentas indispensáveis para engenheiros e desenvolvedores de sistemas embarcados. Editores de esquemáticos são *softwares* especializados que permitem criar e manipular diagramas eletrônicos, conhecidos como esquemáticos (ou esquemas elétricos). Esses diagramas são representações visuais abstratas de um circuito eletrônico, utilizando símbolos padronizados para componentes (resistores, capacitores, transistores, microcontroladores, etc.) e linhas para suas conexões elétricas, que servem como a base para a criação de PCBs (do inglês *Printed Circuit Boards*) funcionais e a integração com sistemas de *hardware* programáveis. Em vez de desenhar um circuito à mão, o editor esquemático oferece uma interface gráfica que agiliza o processo, garante a precisão e facilita a colaboração.

A relevância dos editores de esquemáticos em projetos de sistemas embarcados é imensa, abrangendo todas as etapas do desenvolvimento de *hardware*. O editor de esquemáticos é o primeiro lugar onde a ideia de um circuito se transforma em um plano tangível. É onde as conexões lógicas e elétricas entre o microcontrolador e seus periféricos são definidas. Muitos editores de esquemáticos permitem a simulação do circuito em diferentes estágios do projeto, validando seu comportamento antes mesmo da fabricação física. Isso é essencial para identificar erros, otimizar componentes e prever o desempenho, economizando tempo e recursos. Além disso, o esquemático serve como a documentação primária do circuito, sendo essencial para a depuração, manutenção e futuras modificações.

Após a validação do esquemático, a próxima etapa é geralmente o *layout* da placa de circuito impresso (PCB). Editores esquemáticos modernos são parte de automação de *design* eletrônico (em inglês, *Electronic Design Automation* – EDA) que se integra diretamente com a ferramenta de *layout*. O esquemático gera uma *netlist*, que é uma lista de todas as conexões lógicas e elétricas entre os pinos dos componentes. Esta *netlist* é então importada para a ferramenta de *layout*, que usa essas informações para "desenhar" as trilhas físicas na PCB. As ferramentas de EDA usam o esquemático como referência para verificar se o *layout* da PCB está de acordo com as regras elétricas definidas, como ausência de curtos-circuitos, aberturas, e se as conexões da netlist foram seguidas corretamente.

Embora FPGAs (em inglês *Field-Programmable Gate Arrays*) e CPLDs (em inglês *Complex Programmable Logic Devices*) sejam tradicionalmente programados via linguagens de descrição de *hardware* (HDLs), como VHDL ou Verilog, que são o método predominante para *designs* complexos no estado-da-arte, os editores esquemáticos ainda desempenham um papel complementar. Para blocos específicos dentro de um FPGA, ou em *designs* menores, é possível usar a captura esquemática para descrever o *hardware*. O editor traduz esse esquema em uma *netlist* que é então sintetizada para o FPGA. Essa abordagem pode ser útil para projetistas mais familiarizados com a lógica de circuitos tradicionais. No entanto, em projetos mais avançados e complexos, as HDLs oferecem maior abstração, reusabilidade e escalabilidade, sendo as ferramentas primárias. Assim, a captura esquemática e as HDLs atuam como alternativas complementares, onde esquemáticos podem ser usados para diagramas de blocos de alto nível ou interfaces, e HDLs para a implementação detalhada da lógica interna.

O mercado oferece uma variedade de editores de esquemáticos, cada um com suas próprias características, custos e comunidades de usuários. A capacidade de uma ferramenta de suportar um projeto desde a concepção até a fabricação (sua abrangência) geralmente depende se ela faz parte de uma suíte completa de EDA. Para projetos profissionais e de alta complexidade, o `Altium Designer` se destaca como uma das suítes EDA mais completas. Ele oferece integração perfeita entre a captura esquemática, o *layout* de placas de circuito impresso (PCBs) em 2D e 3D, simulações SPICE, gerenciamento robusto de bibliotecas e análises de integridade de sinal e potência, além de recursos essenciais para a fabricação. Embora seja uma solução de custo elevado, é a escolha principal para engenheiros e grandes empresas.

Para quem busca uma alternativa gratuita e de código aberto, o `KiCad` EDA tem ganhado enorme popularidade. Sua abrangência permite que um projeto vá do conceito à fabricação da PCB, incluindo captura esquemática, *layout* de PCB com visualização 3D, e um editor de bibliotecas. É uma ferramenta poderosa e acessível para amadores, estudantes e pequenas e médias empresas. Outro *software* popular, especialmente entre amadores e pequenas empresas, é o `Eagle` (Autodesk EAGLE). Ele oferece funcionalidades de captura esquemática e *layout* de PCB, com ferramentas de simulação e uma vasta biblioteca de componentes, disponível em versões pagas e uma gratuita com limitações.

No segmento profissional, o `OrCAD` (Cadence OrCAD) se posiciona como uma ferramenta robusta para o *design* de PCBs complexas e simulações avançadas, como SPICE e análises de integridade de sinal e potência, sendo ideal para empresas que trabalham com alta complexidade. Assim como o `Altium`, possui um custo elevado. A `Proteus Design Suite` (Labcenter Electronics), por

sua vez, é reconhecida por sua poderosa capacidade de simulação, permitindo que microcontroladores e seus *firmwares* sejam simulados em conjunto com o circuito analógico/digital. Além da captura esquemática e do *layout* de PCB, sua força reside na simulação completa do sistema, sendo amplamente utilizada por estudantes, amadores e profissionais que necessitam validar o comportamento integrado de *hardware* e *software*. É um *software* pago.

Independentemente da suíte de EDA escolhida, o desenvolvimento de uma PCB segue um fluxo de trabalho padrão consistindo as seguintes etapas:

1. Definir o mapa de conexões (*Pinout*)
    - Mapeie as conexões entre a placa que será depurada (DUD – *Device Under Debug* ou *target*) e a placa que atuará como ferramenta de *debug*, que será uma Raspberry Pi Pico.
    - Identifique quais sinais precisam ser conectados: SWDIO, SWCLK, GND, UART-TX, UART-RX, LEDs, GPIOs de status, etc.
2. Escolher os conectores
    - Com base nas conexões desejadas, escolha os conectores adequados (por exemplo, cabos IDC (em inglês, *Insulation Displacement Connector*) de 14 vias, JST (em inglês, *Japanese Solderless Terminal*) de 4 vias, "cabos *multi jumper*" etc.).

IDC de 14 vias

JST de 4 vias

cabos *multi-jumper*

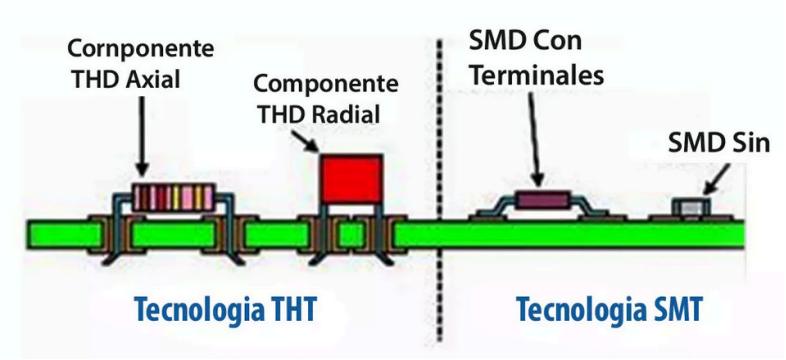
    - Leve em conta a usabilidade, a robustez mecânica e a padronização com a BitDogLab.

**Observação:**
Antes de partir para o esquemático, certifique-se de que as bibliotecas necessárias estão disponíveis.
O KiCad já fornece uma biblioteca padrão bastante completa, e no caso do projeto BitDogProbe ela é suficiente. Mesmo assim, é sempre bom verificar se todos os símbolos e *footprints* que você precisa estão disponíveis.

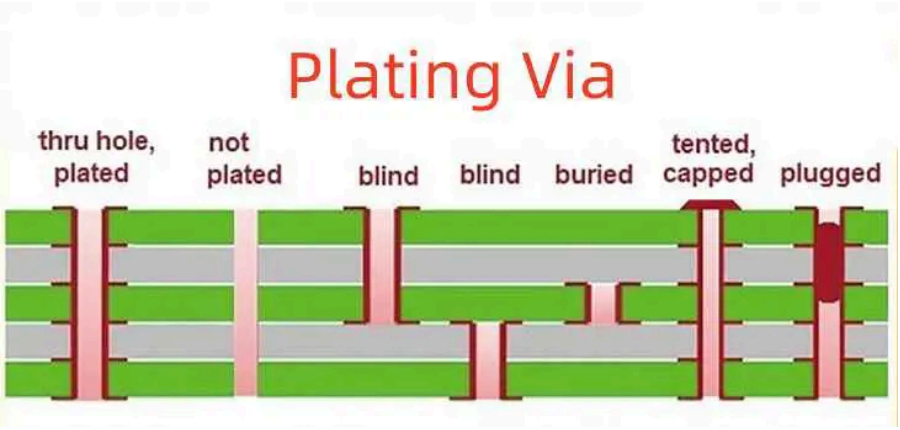
3. Desenhar o Esquemático em uma suíte

- Insira os componentes no esquemático, começando pelas conexões principais e depois incluindo os sinais auxiliares.
- Mantenha uma organização lógica e clara no diagrama para facilitar o entendimento.

4. Atribuir *Footprints* aos Componentes
   - Associe cada símbolo do esquemático ao seu respectivo *footprint*. Essa associação é o elo crucial entre a representação lógica do circuito (o esquemático) e a sua implementação física (o *layout*). O *footprint* atua como uma ponte, definindo a geometria física do componente na PCB, incluindo o tamanho e o espaçamento dos *pads* de solda, que devem corresponder exatamente ao componente real, e o método de montagem. Essa precisão é o que garante que o circuito projetado no esquemático possa ser fabricado e montado corretamente na placa física. Existem dois métodos principais de montagem: THT (do inglês *Through-Hole Technology*) em que os terminais do componente são inseridos em furos passantes na PCB e soldados no lado oposto, e SMD (do inglês *Surface Mount Device*), onde os componentes são soldados diretamente na superfície da PCB, sem a necessidade de furos. Vale ressaltar que os furos metalizados que atravessam as camadas da PCB para criar conexões elétricas são conhecidos como PTH (do inglês *Plated Through Hole*).




Fonte: The Engineering Projects.




Fonte: Rayming PCB & Assembly.

5. Verificar o Esquemático com o ERC

    - Use o *Electrical Rules Checker* (ERC) disponível na suíte para detectar conexões soltas, pinos flutuantes e conflitos lógicos.
    - Corrija os erros antes de prosseguir.
6. Conferir os Nomes dos Nós
    - Verifique se os sinais importantes estão nomeados corretamente, especialmente os que se conectam a mais de um ponto ou vão para conectores. Bons nomes facilitam o roteamento e a leitura do layout.
7. Fazer o Roteamento (PCB *Layout*)
    - Importe o esquemático para o editor de *layout* e posicione os componentes de forma funcional e compacta.
    - Faça o roteamento dos sinais principais. Não é necessário rotear manualmente o GND: usaremos zonas de cobre para isso.
8. Criar Plano de Terra
    - Adicione uma zona de cobre para GND em ambos os lados da placa. Isso melhora o desempenho elétrico, facilita o roteamento e reduz interferências.
9. Verificação Final
    - Execute o DRC para garantir que não há erros de roteamento, violação das regras de design ou conexões faltando.
    - Faça uma revisão visual no visualizador 3D ou no Gerber Viewer do KiCad. Essa inspeção final ajuda a identificar erros que o DRC pode não detectar, como orientação errada de conectores, colisão de serigrafia ou furação mal posicionada.

De forma simplificada o fluxo segue este diagrama:

# FERRAMENTAS DE DESENVOLVIMENTO DE PROJETOS

Nesta seção, vamos explorar as principais ferramentas que permitem ir além do MicroPython, abrangendo tanto a programação de *firmware* de baixo nível quanto o *design* do *hardware*. A API do Pico para C/C++ (Pico SDK) é a escolha ideal para quem busca o máximo de desempenho e controle do microcontrolador. Paralelamente, o KiCad é a ferramenta de *software* livre e de código aberto indispensável para o *design* de *hardware*, possibilitando a criação de esquemáticos, o *layout* de placas de circuito impresso (PCBs) e até mesmo a visualização 3D do projeto.

## Pico SDK

O Raspberry Pi Pico C/C++ SDK (do inglês *Software Development Kit*) oferece uma API (do inglês *Application Programming Interface*) para programar microcontroladores da família RP2040 (e RP2350 via porta) usando C ou C++. Publicado como documentação oficial, este SDK cobre desde infraestrutura básica até suporte a hardware avançado. O Pico SDK é uma ferramenta poderosa e organizada para programar os microcontroladores RP2040/RP2350 em C/C++. Sua arquitetura modular, com abstrações por hardware e utilitários práticos, facilita a construção de aplicações robustas envolvendo interrupções, DMA e mais. Ao explorar a documentação — com atenção ao sumário, prefixos de módulo, e exemplos aplicados — você maximiza sua produtividade e compreensão do SDK.

O SDK do Raspberry Pi Pico é organizado em módulos, cada um representando uma funcionalidade específica dos registradores de cada periférico. Em vez de incluir o SDK inteiro em um projeto, usa-se o CMake para controlar quais módulos serão compilados. O CMake lê o arquivo CMakeLists.txt, onde o desenvolvedor especifica os módulos necessários. Por exemplo, se o projeto usa apenas comunicação I2C e a biblioteca padrão, basta indicar esses módulos no CMakeLists.txt para que o processo de construção de executável (em inglês, *build*) gere um executável incluindo somente o que for necessário. Esse controle direto sobre quais funcionalidades devem estar presentes em um projeto permite que o *firmware* final ocupe menos espaço de memória possível.

As convenções de codificação e *design* da API do Pico em C/C++ seguem as melhores práticas da linguagem, focando em desempenho e legibilidade. Para otimizar o desempenho, o SDK utiliza funções `inline`. Em vez de realizar uma chamada de função, o compilador insere o código diretamente onde a função é usada. Isso elimina a sobrecarga da chamada, tornando o código mais rápido. Para identificar rapidamente a qual periférico de *hardware* a função pertence, melhorando a legibilidade e reduzindo erros, as funções são nomeadas de forma padronizada. O nome do módulo (ex: `adc_`, `gpio_`, `i2c_`) serve como prefixo, seguido de sua função, como em `adc_select_input()`, `gpio_get_functio()` e `i2c_init()`. Além disso, a configuração do *hardware* é projetada de acordo com o padrão `builder`. Em vez de uma única função complexa com muitos argumentos, usa-se uma sequência de chamadas de métodos simples para configurar o *hardware* passo a passo. Isso torna o processo mais claro e flexível, permitindo várias combinações de configurações de forma intuitiva.

O *SDK* agrupa funcionalidades conforme módulos de hardware. Por exemplo, hardware_gpio (`hardware/gpio.h`), hardware_timer (`hardware/timer.h`), hardware_uart (`hardware/uart.h`), hardware_dma (`hardware/dma.h`), hardware_irq (`hardware/irq.h`), hardware_pio (`hardware/pio.h`) entre outros. Cada módulo oferece abstrações de alto nível, mascarando detalhes de registradores, mas permitindo, quando necessário, acesso direto aos mesmos.

Para acesso de baixo nível, os cabeçalhos da "Hardware Structs Library" (por exemplo, `hardware/structs/gpio.h`, `hardware/structs/timer.h`, `hardware/structs/uart.h` etc.) e da "Hardware Registers Library" (por exemplo, `hardware/regs/gpio.h`, `hardware/regs/timer.h`, `hardware/regs/uart.h` etc.) expõem a definição de registradores e *bitfields*, possibilitando controle direto do periférico. Todos esses arquivos de *structs* e *regs* seguem a interface CMSIS (do inglês *Cortex Microcontroller Software Interface Standard*), que padroniza o mapeamento de registradores, nomes de campos e tipos de dados para microcontroladores ARM Cortex. Essa padronização garante que módulos como `hardware_gpio`, `hardware_timer`, `hardware_uart` e demais periféricos sejam descritos de forma consistente, facilitando tanto o acesso direto quanto a portabilidade do código entre microcontroladores da mesma arquitetura.

As funções para cada módulo são detalhadas em seções dedicadas no Capítulo 5 do manual *Raspberry Pi Pico-series C/C++ SDK Libraries and Tools*. A Seção 1 apresenta os módulos de baixo nível, como `hardware_gpio`, `hardware_timer`, `hardware_uart`, `hardware_dma`, entre outros. Cada módulo tem seu próprio grupo de funções e uma subseção dedicada, o que torna mais fácil localizar exatamente o que se precisa. E a Seção 2 descreve as bibliotecas de alto nível, como `pico_stdlib`, `pico_time` etc., que simplificam tarefas comuns e aumentam a produtividade, evitando a necessidade de manipular diretamente registradores de *hardware*.

Ao procurar funções de configuração no SDK do Raspberry Pi Pico, siga a organização da própria documentação oficial. A documentação é extensa (~745 páginas), mas ela é estruturada de forma lógica e facilita a navegação. Seguem-se algumas dicas que ajudam na consulta:

1. Use o índice (Table of Contents) logo nas primeiras páginas para localizar o módulo de interesse rapidamente. Por exemplo, `hardware_dma` aparece direto no sumário.

2. Busque por prefixos específicos: na visualização PDF ou no visor do VSCode, pesquise por `hardware_dma` ou `hardware_irq` para ir direto ao cabeçalho e logo abaixo as funções e comentários relevantes.

3. Acompanhe o *naming convention* (prefixo `hardware_*` vs. `pico_*`), para diferenciar APIs de *hardware* puro (em inglês, *bare-metal*) versus utilitários ou camadas mais altas.

4. Use o padrão *builder* sempre que aplicável, pois muitos módulos usam esse estilo para configurar opções passo a passo.

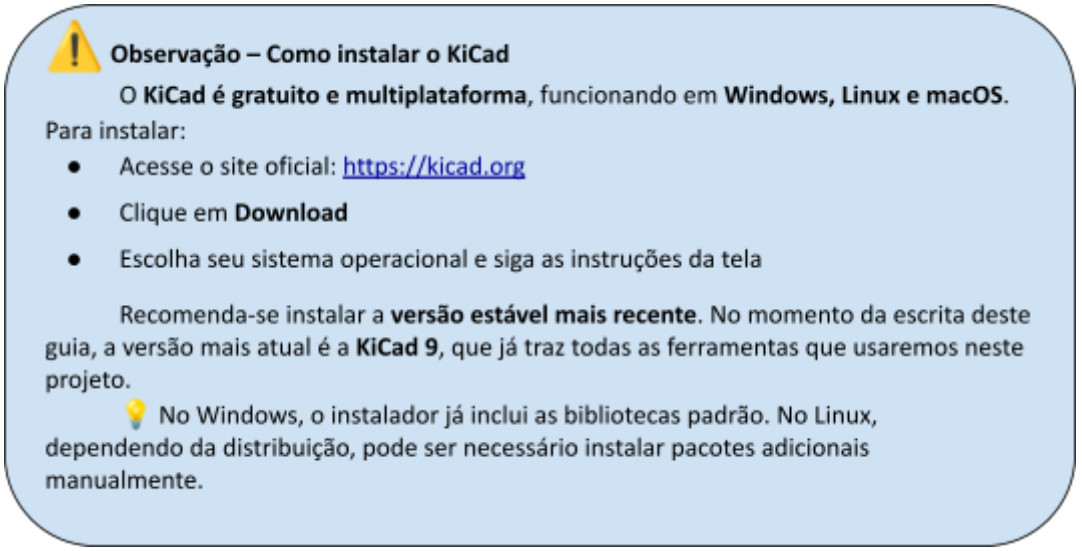
5. Explore exemplos práticos: a documentação inclui exemplos como "A First PIO Application", "WS2812 LEDs", "Attaching an OLED display via I2C" e especialmente "PIO and DMA (A Logic Analyser)". Eles são ótimos para ver uso prático de interrupções e DMA.

Nesta disciplina usaremos as funções pico_* das bibliotecas de alto nível

## KiCAD

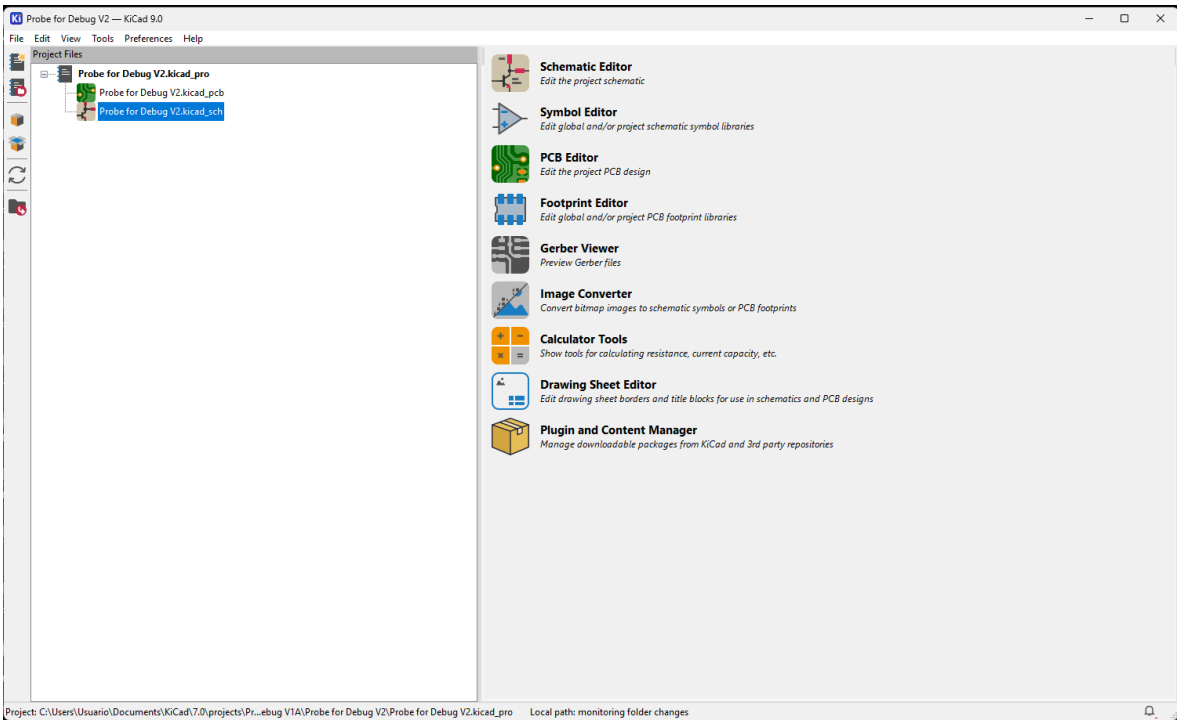
O **KiCad** é uma suíte de *software* de código aberto e gratuita para automação de *design* eletrônico (EDA), amplamente utilizada para projetar esquemáticos e *layouts* de placas de circuito impresso (PCBs). Ele se destaca por sua capacidade de levar um projeto do conceito à fabricação da PCB de

forma completa e acessível. Instalar e começar a usar o **KiCad** é um processo bem direto, seja para um iniciante ou um profissional experiente. Como é uma ferramenta de código aberto, ela está disponível para os principais sistemas operacionais. É possível habilitar o mecanismo de versionamento, git, no kiCAD.




Todos os esquemáticos da placa BitDogLab apresentados neste roteiro foram editados no KiCAD. A interface gráfica do KiCad é modular, composta por vários aplicativos interconectados, cada um com uma função específica no processo de *design*. Destacam-se:

1. KiCad Project Manager (Gerenciador de Projetos): Esta é a janela inicial do KiCad, onde um projeto é organizado como um todo. Aqui pode-se:
    - criar novos projetos;
    - abrir esquemáticos ou *layouts*; e
    - acessar ferramentas como o gerador de netlist, editor de bibliotecas e visualizador 3D.




Pense nessa janela como o “painel de controle” de um projeto.

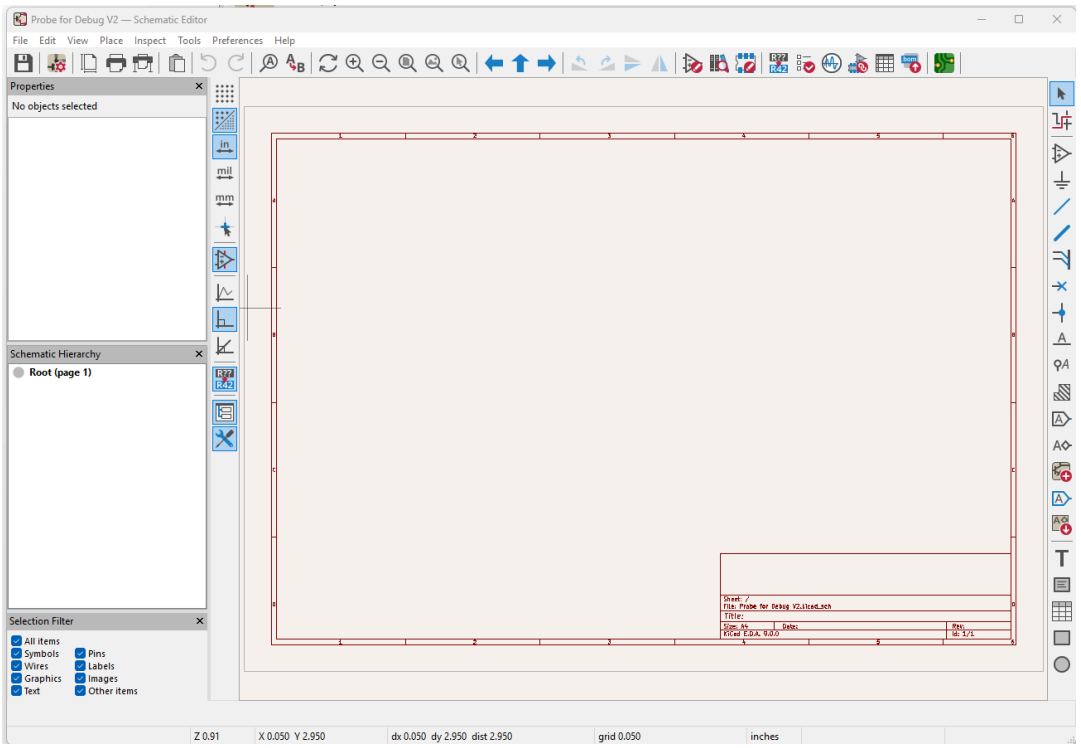
2. Editor Esquemático:

    - Função: É onde se cria o esquemático, ou a representação lógica, de um circuito eletrônico. Adiciona-se símbolos de componentes, os conecta com fios (em inglês, *nets*), e define suas propriedades.

    - Interface: Possui uma área de trabalho central para o desenho, barras de ferramentas com ícones para adicionar componentes, fios, barramentos, rótulos, etc., e painéis laterais para navegação e propriedades.

    - Bibliotecas de Componentes: O KiCad vem com uma vasta coleção de bibliotecas de símbolos esquemáticos padrão (resistores, capacitores, CI's comuns, conectores, etc.). Pode-se pesquisar, adicionar e até criar símbolos personalizados. Cada símbolo é associado a uma *pegada* (em inglês, *footprint*) de PCB, que será usada no *layout*.

    - Verificação de Regras Elétricas (ERC): Uma ferramenta integrada que verifica erros comuns no esquemático, como pinos não conectados ou conflitos de conexão.

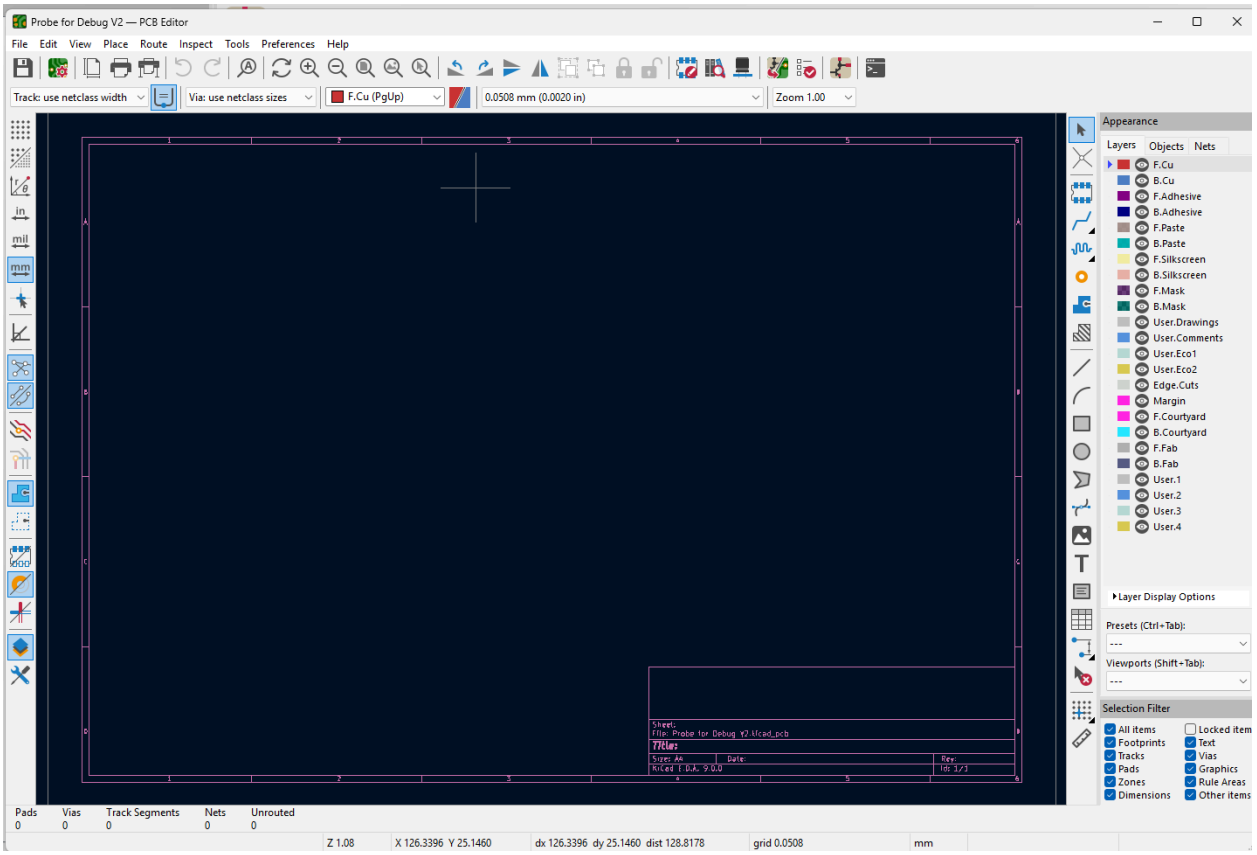



O esquemático é como o "roteiro" do projeto — ele diz o que está conectado com o quê, sem se preocupar ainda com a posição física dos componentes.

3. Editor de PCB:
    - Função: O *layout* da PCB é o coração do *design* da placa. Nesta etapa, a lista de conexões (em inglês, *netlist*) do Eeschema é importada para organizar os componentes fisicamente, rotear as trilhas, definir planos de terra e potência, e configurar as camadas da PCB.
    - Interface: Uma área de trabalho 2D/3D onde se visualiza a placa. As linhas de ar (em inglês, *ratlines*) mostram as conexões que precisam ser roteadas.
    - Roteamento de Trilhas:
        - Manual: Pode-se desenhar as trilhas manualmente, camada por camada, conectando os pinos dos componentes. O KiCad oferece ferramentas para ajustar a largura das trilhas, adicionar vias (furos metalizados que conectam trilhas entre diferentes camadas) e gerenciar planos de cobre.
        - Interativo: O roteador interativo do KiCad auxilia no roteamento ao empurrar outras trilhas e vias para abrir caminho, facilitando o processo.
        - Autorroteador (opcional): Embora o KiCad não tenha um autorroteador interno tão sofisticado quanto algumas suítes comerciais, ele pode ser configurado para usar autorroteadores externos, como o *FreeRouting*, para automatizar parte do processo.

- Visualizador 3D: Permite visualizar a PCB em três dimensões, com os modelos 3D dos componentes, o que é excelente para verificar a estética, o encaixe mecânico e possíveis colisões.

- Verificação de Regras de *Design* (DRC): Essencial para garantir que o *layout* atenda aos requisitos de fabricação (distância mínima entre trilhas, tamanho de furos, etc.) e evitar erros que levariam a uma placa defeituosa.




Essa etapa transforma o projeto lógico em algo concreto e pronto para fabricação.

4. Gerenciador de Bibliotecas: O KiCad possui editores dedicados para criar e gerenciar as próprias bibliotecas de símbolos esquemáticos e pegadas de PCB, permitindo a reutilização de componentes em diferentes projetos.

5. Gerenciamento de Arquivos de Fabricação (Gerber):

    - Função: Após o *layout* da PCB ser finalizado e verificado, o KiCad gera os arquivos necessários para a fabricação da placa. O padrão da indústria para isso é o formato Gerber.

    - Geração: O KiCad exporta um conjunto de arquivos Gerber (um para cada camada de cobre, máscara de solda, serigrafia superior e inferior, etc.) e um arquivo de furação (Excellon). Esses arquivos são enviados para o fabricante da PCB, que os utiliza para produzir a placa física.

Uma introdução ao KiCAD está disponível *online*.

# Projeto da placa BitDogProbe usando KiCad

No Roteiro 1, exploramos a construção de *probes* de depuração utilizando um segundo microcontrolador. Apresentamos duas versões: uma baseada no Raspberry Pi Pico e outra no Bluepill. A placa BitDogProbe, que é a versão baseada no Raspberry Pi Pico, será usada como exemplo prático para ilustrar as etapas do projeto e desenvolvimento de uma placa de circuito impresso (PCB) usando KiCad. Detalharemos o processo de criação da BitDogProbe, desde a concepção inicial até a placa final. Neste roteiro, mostramos as etapas de desenho de esquemáticos apropriados para geração de *layout* de um circuito realizável fisicamente. O desenvolvimento prático desse circuito será explorado em detalhes no Roteiro 3.

## Mapa de conexões

Neste mapa de conexões em forma de tabela, observe que o *Device Under Debug* (DUD) está à direita, que é uma placa BitDogLab, e faz conexões como os dispositivos de *probe* que poderá ser outra BitDogLab ou Raspberry Pico "family" (Pico ou Pico_W ou, Pico2 ou Pico2_w). Apenas um dispositivo será usado como *probe*. Então o usuário deve fazer esta escolha no momento do uso conforme a sua disponibilidade de *hardware*.

| Probe | | | | | DUD - Device Under Debug BitDogLab que será Debugada | | | | |
|---|---|---|---|---|---|---|---|---|---|
| BitDogLab Conector IDC | | Placa Pico "family" | | Função das Conexões | BitDogLab Conector IDC (Opção 1) Recomendada | | BitDogLab Conector JST (Opção 2) Emergencial CUIDADO | | Conexão direta com a placa Pico |
| Pino | Função | Pino | Função | | Pino | Função | Pino | Função | Função |
| 1 | GND | GND | | GND — Alimentação da Placa em debug — OPCIONAL CUIDADO | 1 | GND | 4 | GND | |
| 2 | 5V | 39 | VSYS | | 2 | 5V | | | |
| 4 | GP8 Uart1 Tx | 6 | GP4-Uart1 Tx | Canal entre a: STDIO da Placa em Debug com a Probe — RECOMENDADO | 9 | GP17 Uart0 Rx | 2 | GP01 Uart0 Rx | |
| | | 11 | GP8-Uart1 Tx | | | | | | |
| 6 | GP9 Uart1 Rx | 7 | GP5-Uart1 Rx | | 11 | GP16 Uart0 Tx | 1 | GP00 Uart0 TX | |
| | | 12 | GP9-Uart1 Rx | | | | | | |
| 9 | GP17 | 5 | GP3 | Comunicação para debug — OBRIGATÓRIO | | | | | SWDIO |
| | | 22 | GP17 | | | | | | |
| 11 | GP16 | 4 | GP2 | | | | | | SWCLK |
| | | 21 | GP16 | | | | | | |
| 1 | GND | GND | | | | | | | GND |
| | GP12 | 16 | GP12 | LED - USB — OPCIONAL | | | | | |
| | GP11 | 15 | GP11 | LED - UART TX | | | | | |
| | GP13 | 17 | GP13 | LED - UART RX | | | | | |
| 14 | GP18 | 24 | GP18 | LED - DAP Connected | | | | | |
| 12 | GP19 | 25 | DP19 | LED - DAP Running | | | | | |

**Notas**

- Para as conexões do *Probe* utilizando a placa Pico, os campos em marrom foram adicionados para manter a compatibilidade do *software* entre um *probe* que utiliza a placa Raspberry Pico e a BitDogLab.
- Na DUD há duas opções:

- utilizar as GP16 e 17 para preservar o canal de UART0 utilizado pela stdio, desta forma é necessário apenas realocar, no início do código, os pinos da UART0, sem necessidade de mudar para a UART1. Esta é a opção mais recomendada.
- Ou utilizar as GP01 e GP02, no caso que se esteja em uso o conector IDC, mas não pode ser descartado o risco de conflito com os I2C conectados nesta linha.

No conector IDC da BitDogLab na sua versão mais atual, que também é compatível com a versão 6.3, o mapa de conexões é o seguinte:

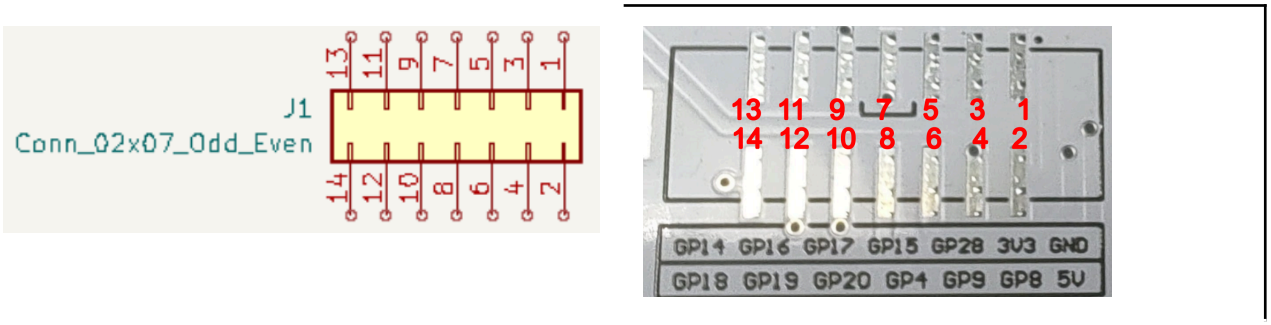



Figura: Correspondência dos pinos do conector IDC 2x7 com as GPIOs da BitDogLab

⚠️ **Observação – Escolhendo os Primeiros Símbolos e *Footprints* no Esquemático**
Ao montar seu primeiro esquemático, é comum surgir dúvida sobre **qual símbolo usar** e **como associar o *footprint*** corretamente. No KiCad, **o símbolo representa a função lógica** (por exemplo, um conector IDC ou uma Raspberry Pi Pico) e o ***footprint* representa a forma física e os pinos reais na placa**. Aqui vão dois exemplos que você pode usar neste projeto:

**Conector IDC 2x7 (14 pinos)**

- **Símbolo (Value):** Conn_02x07_Odd_Even
- **Footprint:** Connector_IDC:IDC-Header_2x07_P2.54mm_Vertical
  Este é o conector usado para cabos *flat*, muito comum em *debug*.

**Módulo Raspberry Pi Pico**

- **Símbolo (Value):** RaspberryPi_Pico
- **Footprint:** Module:RaspberryPi_Pico_SMD_HandSolder
  Esse *footprint* é ideal para montagem manual, e já traz a pinagem correta para o modelo SMD.
  ❗ Sempre confira se o *footprint* corresponde ao **modelo físico real** que você tem em mãos. Verifique número de pinos, espaçamento e posição para evitar problemas na fabricação e montagem.

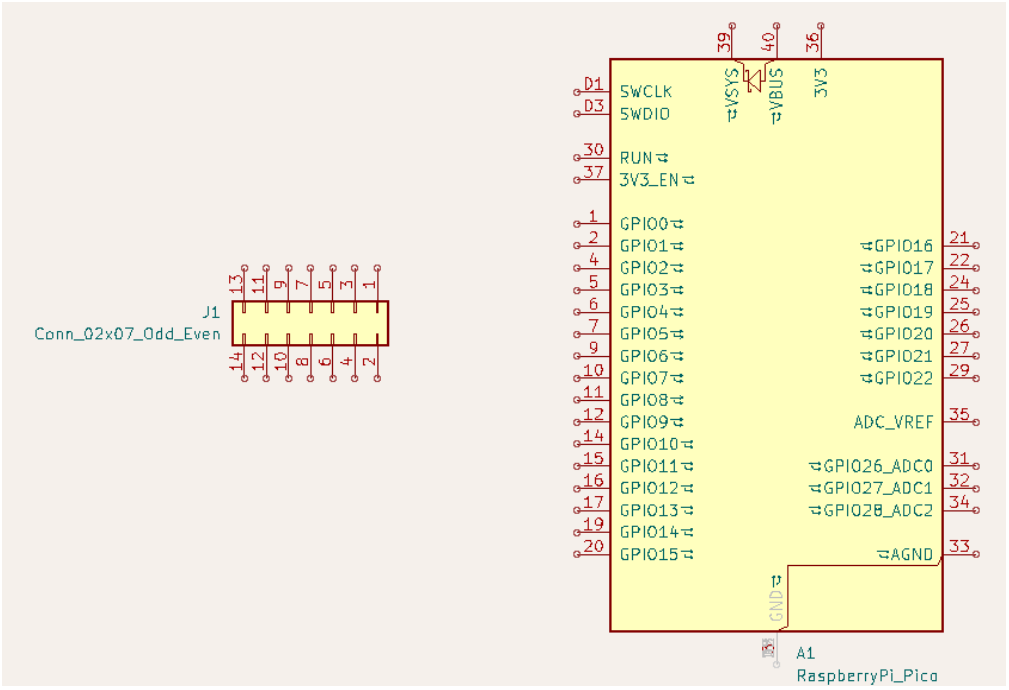
## Construindo o esquemático e preparando o *layout* da placa

Feitas as primeiras seleções de componentes:

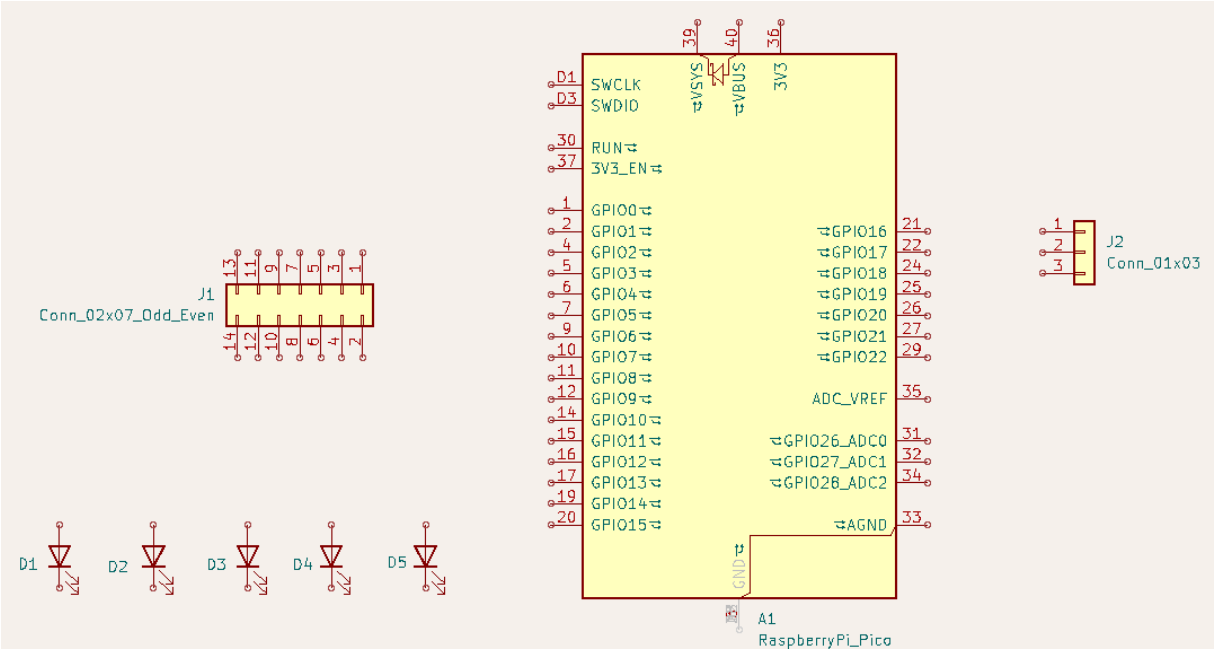
- **Conector IDC 2x7**

    - **Nome do símbolo (*Value*):** Conn_02x07_Odd_Even
    - ***Footprint*:** Connector_IDC:IDC-Header_2x07_P2.54mm_Vertical
- **Módulo Raspberry Pi Pico**
    - **Nome do símbolo (*Value*):** RaspberryPi_Pico
    - ***Footprint*:** Module:RaspberryPi_Pico_SMD_HandSolder

...os símbolos aparecerão automaticamente no editor de esquemático.




Pode-se também incluir alguns LEDs de sinalização conectados aos GPIOs para que uma sinalização visual seja possível durante o processo de Debug.




A tabela a seguir mostra todos os nomes que são usados para referenciar todos os componentes da placa em cada instância de projeto.

| Nome de referência | Nome do componente | Nome do símbolo (Value) na biblioteca KiCad | Nome do Footprint na biblioteca KiCad |
|---|---|---|---|
| Probe1 | Módulo Raspberry pi pico | RaspberryPi_Pico | Module:RaspberryPi_Pico_Common_THT |
| J1_BDL_DUD1<br>J4_BDL_Probe1 | Conector IDC 2x07 - 14 pinos | Conn_02x07_Odd_Even | Connector_IDC:IDC-Header_2x07_P2.54mm_Vertical |
| D1 a D5 | LED THT | LED | LED_THT:LED_D3.0mm_FlatTop |
| R1 a R10 | Resistor THT - 1/4W | R | R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm |
| J2_BDL_DUD1 | Conector JST 3 pinos passo 2,54mm | Conn_01x03 | Connector_JST:JST_EH_B3B-EH-A_1x03_P2.50mm_Vertical |
| J3 | Barra de pinos 01x02 | Conn_01x02 | Connector_JST:JST_EH_B2B-EH-A_1x02_P2.50mm_Vertical |
| J5 | Conector JST 4 pinos passo 2,54mm | Conn_01x04 | Connector_JST:JST_XA_B04B-XASK-1-A_1x04_P2.50mm_Vertical |

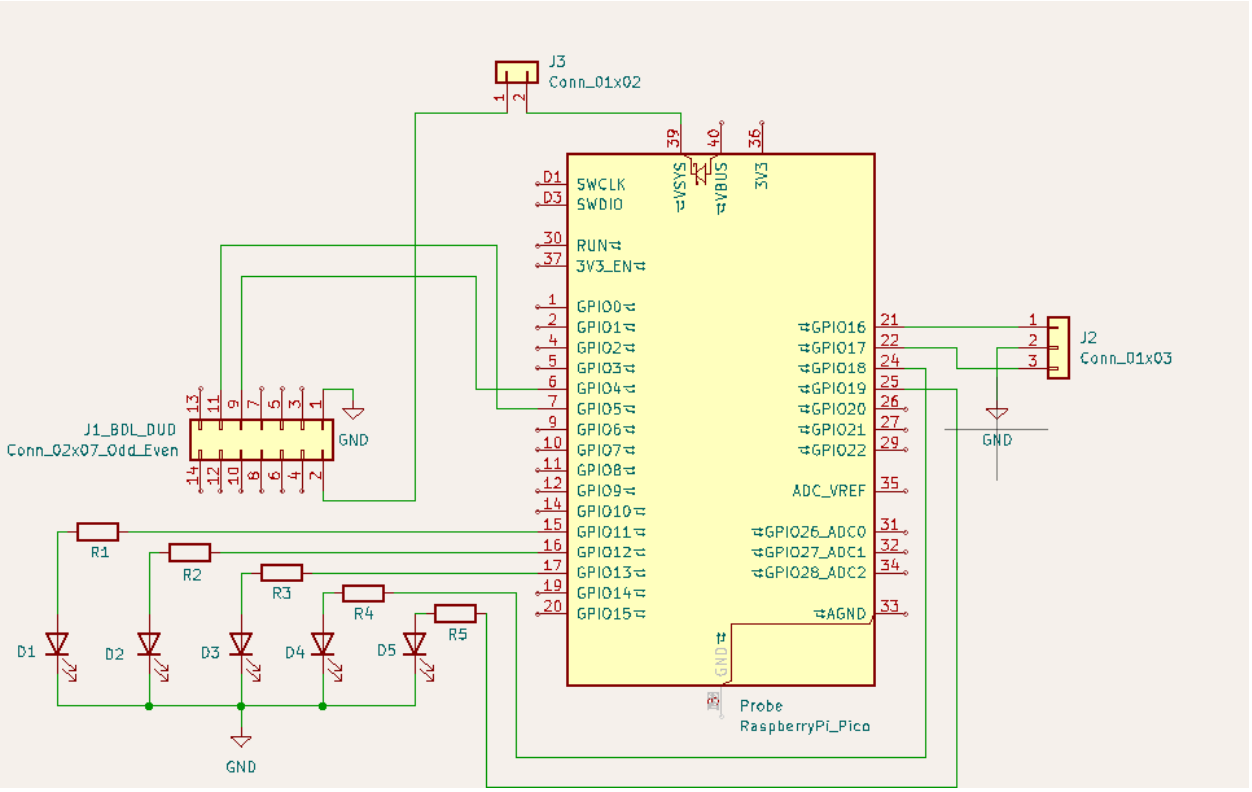
Adicionar valores de resistência para os RESISTORES DE R1 a R10.

Se a Raspberry Pi Pico é usada com placa de Debug, o mapa de conexão destes LEDs é a seguinte:

| LED | GPIO da Pico X | Sinalização |
|---|---|---|
| D1 | 11 | UART Tx |
| D2 | 12 | USB |
| D3 | 13 | UART Rx |
| D4 | 18 | DAP Con |
| D5 | 19 | DAP Run |

A terceira coluna mais a esquerda mostra uma sugestão de sinalização para cada LED. Esta definição deve ser incorporada no software, porém já é desejável que seja identificada em cada LED pelo *silkscreen* correspondente.

Com os símbolos posicionados, o próximo passo é conectar os componentes com base no mapa de conexões planejado. Use os símbolos apropriados de GND e VCC como *ports* para criar referências de alimentação.



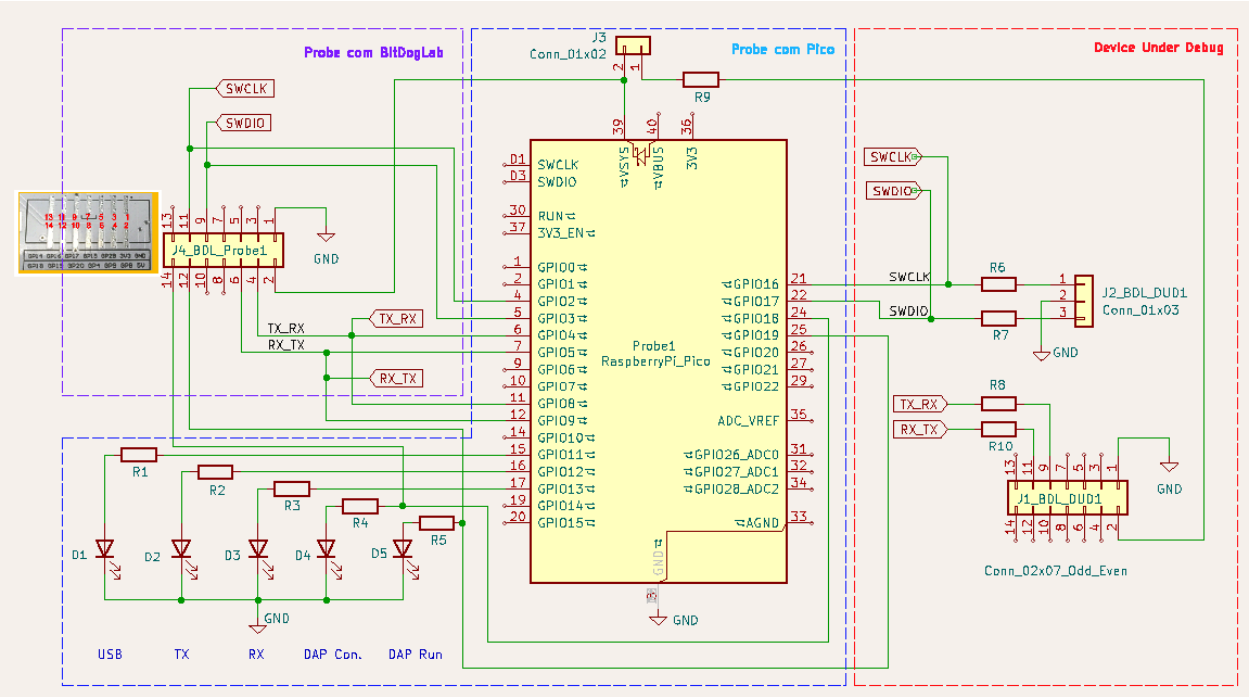

Organize o esquemático pensando na clareza visual e na colaboração futura. Um projeto bem organizado facilita revisões, reuso e melhorias contínuas por outros integrantes da comunidade BitDogLab.

Na aba direita do editor de esquemático, pode-se selecionar linhas e texto para delimitar áreas que organizam o *layout* do circuito.

**Barra lateral direita do editor de esquemáticos (Eeschema) do KiCad 9**.
Abaixo segue um resumo funcional de cada ícone, de cima para baixo:

1. **Selecionar (seta preta)** – Ferramenta para selecionar e mover componentes e fios.
2. **Colocar componente** – Adiciona um símbolo (componente) ao esquemático.
3. **Colocar pino de alimentação (power)** – Adiciona fontes de alimentação como VCC, GND, etc.
4. **Colocar fio (Wire)** – Desenha fios simples para conexão elétrica entre pinos.
5. **Colocar barramento (Bus)** – Insere um barramento (linha grossa) para múltiplos sinais agrupados.
6. **Colocar fio de barramento (Bus Entry)** – Cria um triângulo para entrada/saída de sinais entre fios e barramentos.
7. **Colocar fio global (Label)** – Insere rótulo de sinal local.
8. **Colocar rótulo hierárquico** – Usado para ligar sinais entre folhas hierárquicas.
9. **Colocar pino de hierarquia** – Adiciona um pino na folha de hierarquia.
10. **Colocar ponto de junção** – Adiciona um nó (ponto de conexão entre fios).
11. **Colocar ponto de teste (probe)** – Para depuração, indica pontos de medição.
12. **Colocar zona de preenchimento (Fill Area)** – Define uma área a ser preenchida com polígonos ou máscaras.
13. **Colocar caixa de hierarquia** – Insere uma subfolha (módulo hierárquico).
14. **Entrar em subfolha** – Abre a folha hierárquica selecionada.
15. **Sair da subfolha** – Volta ao nível hierárquico superior.
16. **Importar folha de hierarquia (ícone com +)** – Importa símbolos e conexões da subfolha.
17. **Gerar folha de hierarquia** – Cria um novo arquivo de subfolha.
18. **Remover folha de hierarquia (ícone com x)** – Remove folha da hierarquia.
19. **Inserir texto** – Adiciona texto livre no esquemático.
20. **Inserir comentário** – Comentário explicativo sem função elétrica.
21. **Inserir tabela** – Insere uma tabela para documentação.
22. **Inserir retângulo** – Desenha uma forma retangular para fins gráficos.
23. **Inserir elipse/círculo** – Desenha uma forma oval ou circular.
24. **Inserir arco** – Desenha um arco gráfico.
25. **Inserir curva à mão livre (Bezier)** – Desenha curvas manuais.
26. **Ferramenta de identidade do usuário** – Define dados do usuário/editor.

Após conectar todos os pinos, obtemos o seguinte esquemático.




## Checklist do esquemático

Execute o ERC (do inglês *Electrical Rules Checker*) no editor de esquemático.(MENU > Inspecionar > Verificador de Regras Elétricas (ERC)). Se algum componente não tiver um *footprint* associado, o KiCad mostrará um erro nessa etapa. Corrija todos os avisos e esteja certo de que o esquemático está pronto antes de passar ao *layout*.

# Guia completo de desenvolvimento – projeto P2

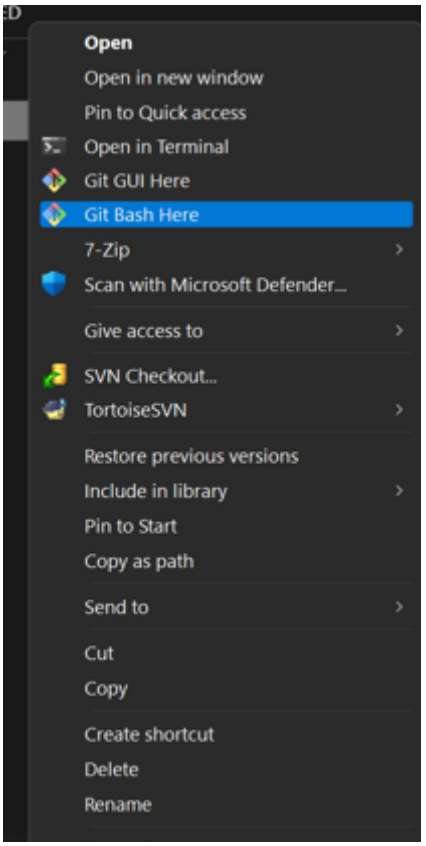
Este guia cobre o processo completo de construção da versão 0.1 (*branch* principal) do projeto P2, que envolve *firmware* em C e *hardware* eletrônico, utilizando

- Git para controle de versões do projeto.
- Visual Studio Code para desenvolvimento de *firmwares*/*softwares*.
    - Extensão GitLab Workflow para serviço de hospedagem de repositórios remotos baseado em Git (https://gitlab.unicamp.br/)
    - Extensão Raspberry Pi Pico para desenvolvimento de *firmwares*, incluindo Pico SDK.
    - Extensão Cortex-Debug para depuração de *firmwares* dos microcontroladores baseados na arquitetura ARM Cortex-M.
    - Extensão Doxygen Documentation Generator para documentação de *firmwares*.
- KiCad para captura de esquemáticos e geração de *layouts* de PCB.

### Clonagem do repositório GitLab

Etapas:

1. Abra um Git Bash.




2. Navegue até uma pasta local de trabalho no terminal (shell) aberto, usando a linha de comando

   cd caminho/da/pasta/do/projeto/local

3. Clone o projeto P2 do grupo,  por exemplo o projeto P2 do grupo G1 da turma A.

```shell
# Clonar o repositório
git clone https://gitlab.unicamp.br/EA801_2025S2/tA/G1/P2.git

# Acessar a pasta do projeto (opcional)
cd P2
```

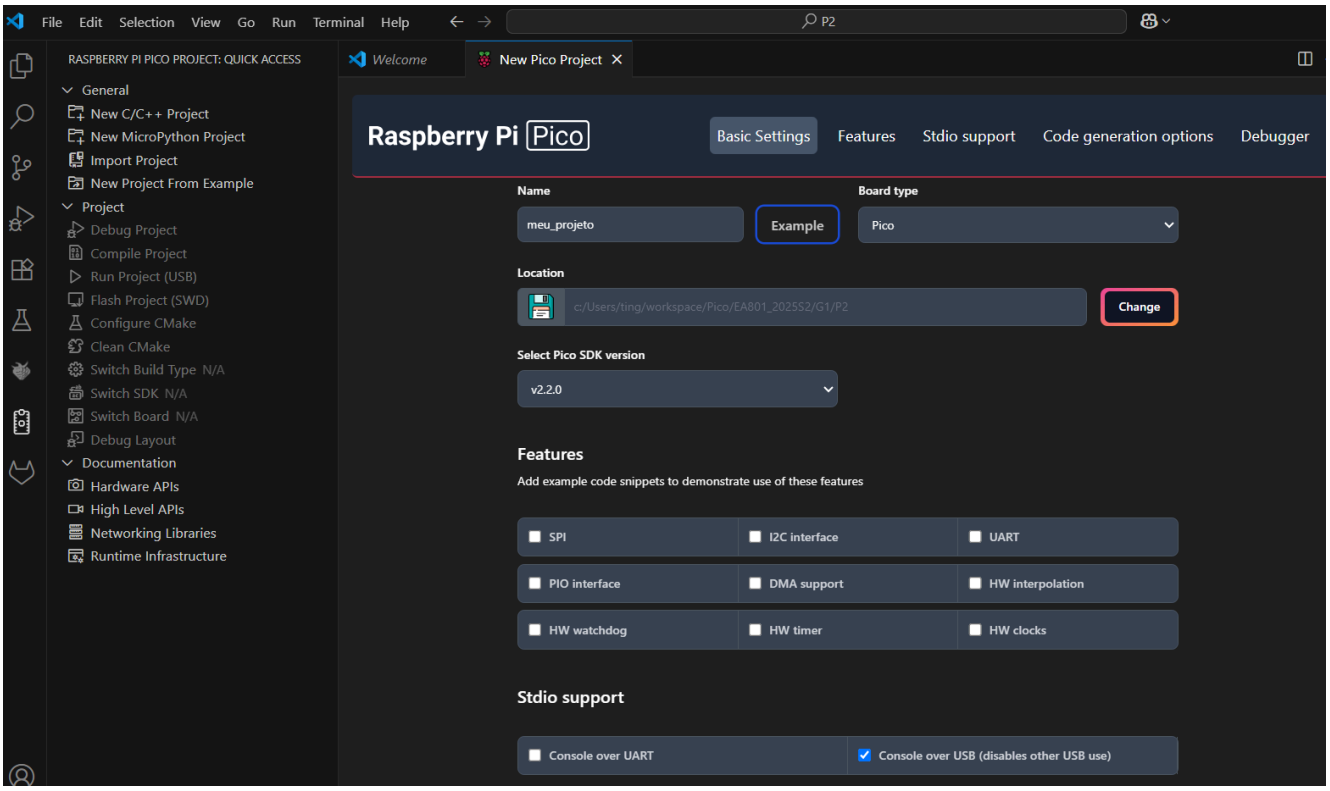
### Configuração do Visual Studio Code no Raspberry Pi Pico

Etapas:

1. Crie um novo projeto Raspberry Pi Pico no VS Code, usando a interface gráfica
   a. Abra o VS Code.
   b. Abra a tela de "Raspberry Pi Pico Project".
   c. Selecione "New C/C++ Project" na aba esquerda.

d. Preencha os campos de dados do novo projeto, incluindo o nome do projeto de *firmware* (por exemplo, meu_projeto), tipo de placa (Pico) e localização (caminho onde o projeto P2 do repositório foi clonado).

   O *console over USB* é a escolha padrão, porque ele aproveita a infraestrutura já existente na placa e no seu *setup* de desenvolvimento, eliminando a necessidade de *hardware* extra e simplificando o processo de depuração e visualização de *logs*. A UART é mais usada em cenários específicos, como quando se precisa comunicar com outro dispositivo serial ou quando a interface USB já está sendo usada para outra coisa no projeto.




e. Clique em “Create” no final da tela e confirme “Yes, I trust the authors”, para gerar um novo projeto de nome P2 na pasta clonada.

2. Abra `.gitignore`

   Por quê usar `.gitignore`? Evita versionar arquivos que não são arquivos-fonte como

   - compilados (binários);
   - gerados automaticamente (documentação, gerbers); e
   - temporários (VS Code, backups).

   Apenas duas regras de exclusão são incluídas no arquivo .gitignore por padrão: build/ e !.vscode/*. A primeira ignora todos os arquivos gerados na pasta build, enquanto a segunda garante que os arquivos de configuração do VS Code dentro do primeiro nível da pasta .vscode sejam incluídos. Para o nosso controle de versão, vamos usar as seguintes regras de exclusão:

```
None
# Compilação e linkagem
# Ignorar o conteúdo de build
build/*
# mas não a pasta
!build/.gitkeep

# Configuração de VS Code
!.vscode/*

# Logs
*.log

# Backups
*~
*.swp

# Doxygen
# Ignorar o conteúdo de docs
docs/*
# mas não a pasta
!docs/.gitkeep

# Outros
.DS_Store
Thumbs.db
```

### Reestruturação das pastas do projeto

Por padrão, a estrutura do projeto gerado pelo Visual Studio Code tem apenas um nível de profundidade:

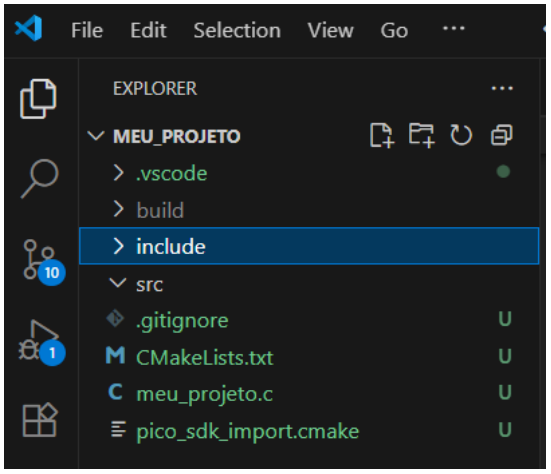
```
None
P2/meu_projeto
├── .vscode
├── build
├── .gitignore
├── CMakeLists.txt
```

```
├── meu_projeto.c
└── pico_sdk_import.cmake
```

No entanto, à medida que os projetos crescem em complexidade, a adoção de uma estrutura mais organizada, como a separação dos arquivos de código-fonte (.c, .cpp) e de cabeçalho (.h) em pastas distintas (src e include, respectivamente), torna-se uma prática recomendada. Essa organização melhora a clareza, a manutenção e a escalabilidade do código. Para um projeto melhor estruturado, a estrutura ideal seria:

```
None
P2/meu_projeto
├── .vscode/
├── build/
├── src/
│   └── meu_projeto.c
├── include/
│   └── meu_projeto.h
├── .gitignore
├── CMakeLists.txt
└── pico_sdk_import.cmake
```

Para criar as duas subpastas, clique no ícone “New Folder …” (pasta com um sinal de mais) que aparece ao lado do nome do MEU_PROJETO na barra lateral do VS Code.

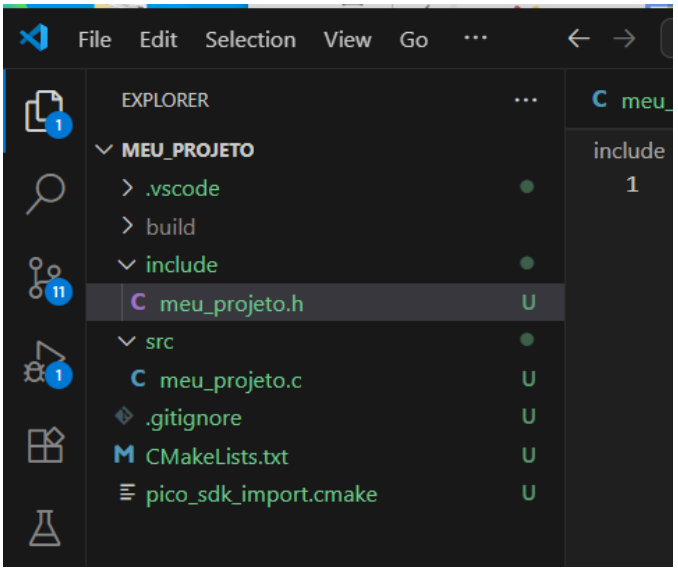



Depois de criar as pastas, arraste o arquivo meu_projeto.c para a pasta src. Em seguida, crie um novo arquivo de cabeçalho. Para isso, selecione a pasta include e clique no ícone “New File …” (arquivo com um sinal de mais), nomeando-o como meu_projeto.h.

Como o Git não rastreia pastas vazias, é necessário adicionar um arquivo de marcador para que a pasta build seja incluída no repositório. A convenção mais comum para isso é criar um arquivo vazio chamado .gitkeep dentro da pasta. Pode-se fazer isso no Terminal (do tipo Git bash ou Command Prompt) com o seguinte comando:

```
None
touch build/.gitkeep
git add build/.gitkeep
git commit build/.gitkeep -m "Adiciona a pasta build"
```

O arquivo .gitkeep serve apenas para sinalizar a intenção de manter a pasta no controle de versão, mesmo que ela esteja vazia.




## Desenvolvimento de *firmware* em C no VS Code

O que é um Makefile? É um arquivo de texto usado principalmente em programação para automatizar o processo de compilação e construção de um projeto de *software*. Ele age como um conjunto de instruções ou "receitas" que dizem ao compilador e a outras ferramentas como transformar o código-fonte em um programa executável.

Para que serve Makefile? A principal função do Makefile é otimizar e simplificar o fluxo de trabalho de compilação, especialmente em projetos grandes e complexos, garantindo que ele seja compilado de forma eficiente, correta e repetível. Ele faz isso através de duas funções principais:

1. Automação: Em vez de digitar manualmente uma longa série de comandos de compilação no terminal, pode-se simplesmente usar o comando make. O Makefile cuida de todo o processo, desde compilar arquivos individuais até ligar todas as partes em um executável.

2. Compilação Incremental: O Makefile verifica quais arquivos de código-fonte foram modificados desde a última compilação. Ele só recompila os arquivos que foram alterados e aqueles que dependem deles, economizando tempo e recursos.

O problema é que os arquivos Makefile tendem a ser complexos, difíceis de manter e, o pior, não são portáveis. Um Makefile escrito para Linux pode não funcionar em Windows, por exemplo. É aí que o CMake entra. O CMake não é um compilador; ele é um gerador de sistemas de compilação. Ele age como uma camada de abstração. Em vez de escrever um Makefile diretamente, cria-se um arquivo chamado `CMakeLists.txt` que descreve seu projeto de uma forma mais simples e genérica.

O CMake lê o `CMakeLists.txt` e, com base no seu sistema operacional e compilador, gera automaticamente os arquivos de *build* apropriados para você. No Linux, ele gera um Makefile. No Windows, ele pode gerar um arquivo de projeto do Visual Studio. Essa abordagem garante que um mesmo projeto possa ser compilado em diferentes plataformas sem que se precise alterar o código ou as regras de compilação. Para o Raspberry Pi Pico SDK, essa é a abordagem padrão. O SDK já foi projetado para usar o CMake, o que simplifica a configuração de projetos e a inclusão das bibliotecas do Pico.

Segue-se a explicação das definições no CMakeLists.txt, geradas automaticamente pelo Visual Studio Code – Raspberry Pi Pico:

- cmake_minimum_required(VERSION 3.13): Define a versão mínima do CMake necessária para o projeto.
- set(CMAKE_C_STANDARD 11): Instrui o compilador a usar a versão C11 da linguagem C.
- set(CMAKE_EXPORT_COMPILE_COMMANDS ON): Instrui o CMake a criar um arquivo que ajuda outras ferramentas a "entender" e analisar seu projeto.
- if(WIN32): Verifica se o sistema operacional em que a compilação está sendo executada é o Windows. A variável WIN32 é uma variável interna do CMake que se torna verdadeira quando o sistema operacional é o Windows.
- set(USERHOME $ENV{USERPROFILE}): Se o sistema for Windows, esta linha define a variável CMake USERHOME com o valor da variável de ambiente USERPROFILE. No Windows, USERPROFILE aponta para o diretório do usuário (por exemplo, C:\Users\NomeDeUsuario).
- else(): Marca o início do bloco de código a ser executado se a condição if(WIN32) for falsa, ou seja, se o sistema operacional não for Windows.
- set(USERHOME $ENV{HOME}): Se o sistema não for Windows (sendo, por exemplo, Linux ou macOS), esta linha define a variável CMake USERHOME com o valor da variável de ambiente HOME. A variável HOME é o padrão nesses sistemas para o diretório do usuário (por exemplo, /home/nomedeusuario).
- set(sdkVersion 2.2.0): Define uma variável chamada sdkVersion com o valor 2.2.0, indicando a versão do SDK do Pico que o projeto espera usar.
- set(toolchainVersion 14_2_Rel1): Define a variável toolchainVersion com o valor 14_2_Rel1, especificando a versão da cadeia de ferramentas (ferramentas de compilação) que o projeto precisa.

- set(picotoolVersion 2.2.0): Define a variável picotoolVersion com o valor 2.2.0, indicando a versão da ferramenta picotool. Essa ferramenta é frequentemente usada para carregar programas para o Pico.
- set(picoVscode ${USERHOME}/.pico-sdk/cmake/pico-vscode.cmake): Constrói o caminho completo para o arquivo pico-vscode.cmake. Ele usa a variável USERHOME que foi definida anteriormente para apontar para o diretório do usuário e, em seguida, adiciona a localização padrão do arquivo de configuração específico do VS Code (Visual Studio Code).
- if (EXISTS ${picoVscode}): Verifica se o arquivo especificado no caminho picoVscode realmente existe no sistema.
- include(${picoVscode}): Se o arquivo existir, esta linha o inclui no processo de compilação do CMake. O arquivo pico-vscode.cmake geralmente contém configurações específicas para integrar o SDK do Pico com o VS Code, facilitando o desenvolvimento e a depuração.
- set(PICO_BOARD pico CACHE STRING "Board type"): Configura a placa padrão do projeto para ser a Raspberry Pi Pico e salva essa configuração para uso futuro, facilitando o processo de compilação.
- pico_sdk_import.cmake: Este comando importa as configurações do Pico SDK, permitindo que o CMake saiba como compilar para o Pico.
- project(meu_projeto ...): Define o nome do projeto (meu_projeto) e as linguagens de programação usadas.
- pico_sdk_init(): Inicializa as configurações padrão do SDK.
- add_executable(meu_projeto main.c): Diz ao CMake para criar um executável chamado P2 a partir do arquivo main.c. Se tiver mais arquivos, basta listá-los aqui.
- pico_set_program_name(meu_projeto "meu_projeto"): define um nome mais descritivo ao arquivo binário final (por exemplo, meu_projeto.uf2) ou para exibir o nome do programa em ferramentas de depuração.
- pico_set_program_version(meu_projeto "0.1"): define a versão do programa.
- target_include_directories(meu_projeto PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}): Diz ao CMake que os arquivos de cabeçalho (.h) para este projeto (P2) estão na pasta ${CMAKE_CURRENT_SOURCE_DIR}/include. O argumento PUBLIC garante que as bibliotecas que dependem deste projeto também possam ver esses cabeçalhos. ${CMAKE_CURRENT_SOURCE_DIR} é o caminho da pasta onde o `CMakeLists.txt` está.
- target_link_libraries(meu_projeto pico_stdlib): Liga a biblioteca pico_stdlib, que contém funções úteis como sleep_ms() e as funções de GPIO.
- pico_add_extra_outputs(meu_projeto): Esta linha é específica para o Pico SDK. Ela instrui o CMake a gerar arquivos extras, como o .uf2 para o *upload* e o .elf para a depuração.

Para se adequar à nova estrutura de pastas (src e include), as seguintes linhas do *script* CMakeLists.txt precisam ser ajustadas:

1. A linha target_include_directories precisa ser alterada para apontar para a nova pasta include.

```
None
target_include_directories(meu_projeto PRIVATE

    ${CMAKE_CURRENT_LIST_DIR}/include )
```

2. A linha add_executable deve ser modificada para refletir que o arquivo-fonte (meu_projeto.c) foi movido para a pasta src.

```
None
add_executable(meu_projeto src/meu_projeto.c )
```

Supondo que o meu_projeto.h contenha apenas o protótipo da função main. Clique no meu_projeto.h para abrí-lo e digite os seguintes comandos:

```
None
#ifndef MEU_PROJETO_H
#define MEU_PROJETO_H

#include "pico/stdlib.h"

int main();

#endif
```

Escreva o código C em meu_projeto.c. Para este exemplo, usaremos um programa simples que pisca um LED.

```
None
#include "meu_projeto.h"

#ifndef LED_DELAY_MS
#define LED_DELAY_MS 250
#endif

int main() {
    gpio_init(PICO_DEFAULT_LED_PIN);
    gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
```

```
    while (true) {
        gpio_put(PICO_DEFAULT_LED_PIN, 1);
        sleep_ms(LED_DELAY_MS);
        gpio_put(PICO_DEFAULT_LED_PIN, 0);
        sleep_ms(LED_DELAY_MS);
    }
}
```

### Iniciando a depuração do projeto

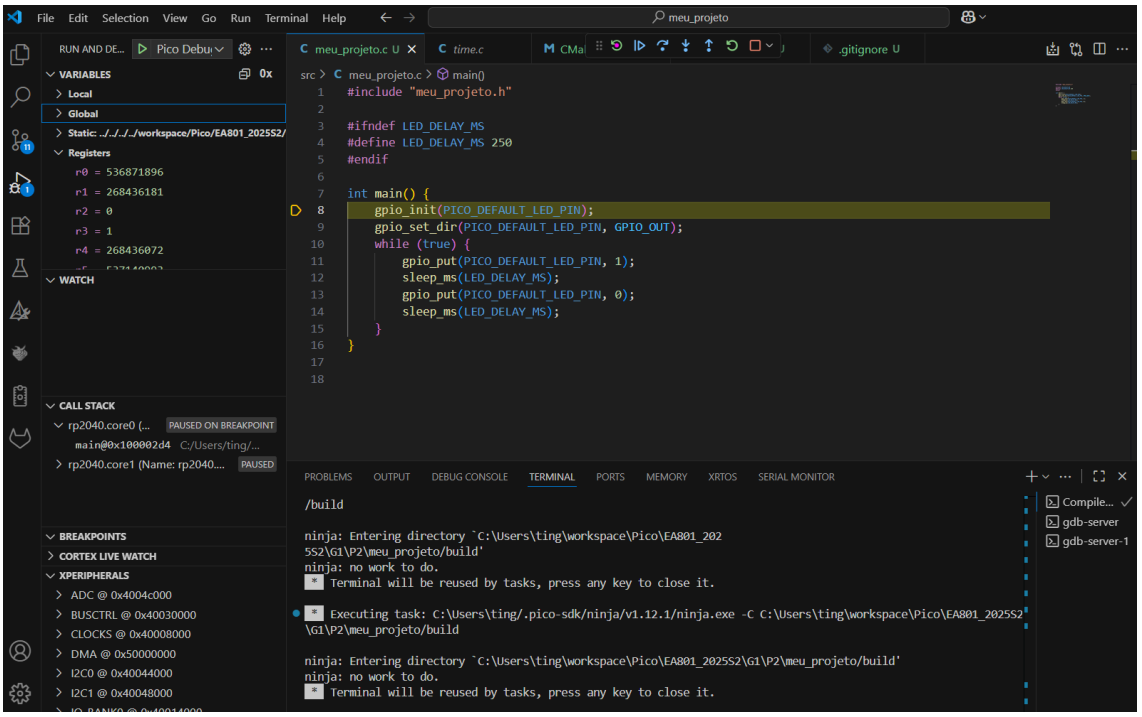
Conecte o *probe* DAP-LINK ao computador via USB e, em seguida, conecte-o ao Raspberry Pi Pico usando os pinos de depuração SWD.

Para o VS Code compilar, transferir o código ao Pico via *probe* e executar o código, tudo em um só passo:

1. Vá em Run (Executar) na barra de ferramentas superior.
2. Clique em Start Debugging.

Após iniciar a execução do código, a interface do Visual Studio Code muda para focar nas ferramentas de depuração. Pode-se acessar as funcionalidades em algumas áreas principais, todas concentradas na barra lateral esquerda, e em uma barra de ferramentas superior (ícones em azul), como:

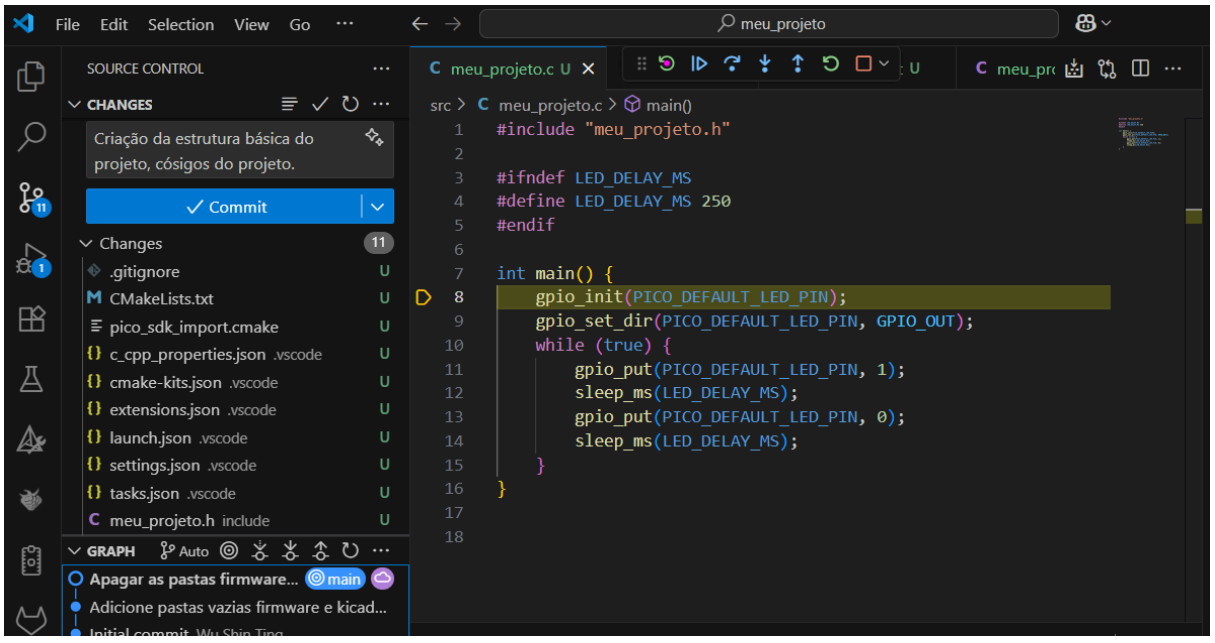
- Definir pontos de interrupção (*breakpoints*).
- Executar o código linha por linha.
- Pausar ou continuar a execução.
- Inspecionar o valor de variáveis em tempo real.
- Visualizar a pilha de chamadas e a memória.



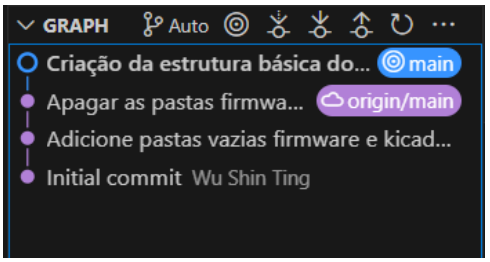

## Confirmação de alterações com o Git

Ao fazer alterações no código, o ícone de “Source Control” (o ícone de bifurcação) na barra lateral do VS Code mostra um número que corresponde à quantidade de arquivos modificados. Para registrar essas alterações:

1. Clique no ícone de “Source Control” para ver a lista de arquivos que foram modificados. Esses arquivos são marcados com a letra U de “*untracked*” (não rastreado) ou M de “modified” (modificado).
2. Na caixa de texto acima da lista de arquivos, insira uma mensagem de *commit* que descreva claramente as alterações feitas.
3. Para confirmar o *commit*, clique no botão de Commit (o ícone de ✅) ou use o atalho Ctrl + Enter.



Isso irá registrar as alterações no repositório local, criando um ponto de salvamento no histórico do projeto, “Criação da estrutura …”. A partir daí, pode-se enviar (push) essas alterações para o repositório remoto.




## Geração de documentação com Doxygen

Para gerar a documentação com Doxygen, adicionamos a pasta docs para armazenar os arquivos de saída. Também criamos e commitamos o arquivo de configuração Doxyfile, que define como a documentação será gerada. Para criar o arquivo `Doxyfile`, execute o seguinte comando no Terminal:

```
doxygen -g Doxyfile
```

Em seguida, personalize os valores dos seguintes parâmetros.

```
None
PROJECT_NAME = "Nome do seu projeto"
INPUT = ./include
OUTPUT_DIRECTORY = ./docs
EXTRACT_ALL = YES
RECURSIVE = YES
MSCGEN_PATH = "caminho/para/o/arquivo/do/mscgen"
HAVE_DOT = YES
DOT_PATH = "caminho/para/o/dot/bin"
CALL_GRAPH = YES
CALLER_GRAPH = YES
GRAPH_METRICS = YES
CLASS_DIAGRAMS = YES
HTML_OUTPUT = html
GENERATE_LATEX = NO
CREATE_SUBDIRS = YES
```

Como o Git não rastreia pastas vazias, a pasta `docs` não seria incluída no controle de versão se estivesse vazia. Para que a estrutura do projeto seja compartilhada corretamente com a pasta `docs` já no lugar, precisamos adicionar um arquivo de marcador dentro dela. A convenção padrão para isso é usar um arquivo vazio chamado `.gitkeep`. Ele sinaliza ao Git para rastrear a pasta, mesmo

que não haja nenhum conteúdo gerado ainda. Pode-se fazer isso no Terminal (do tipo Git bash ou Command Prompt) com o seguinte comando:

```
None
touch docs/.gitkeep
git add docs/.gitkeep
git commit docs/.gitkeep -m "Adiciona a pasta docs"
```

O arquivo .gitkeep serve apenas para sinalizar a intenção de manter a pasta no controle de versão, mesmo que ela esteja vazia.

Vamos documentar o protótipo da função main no arquivo de cabeçalho meu_projeto.h com Doxygen:

```
C/C++
/**
 * @brief Fluxo de controle principal do estado alternante do
LED
 */
int main (void);
```

Para poder digitar o comando doxygen Doxyfile no Terminal, pode-se abrir um novo terminal:

1. Vá até o canto superior direito do Terminal e clique no ícone (^ invertido) e selecione “Split Terminal” seguido do tipo de terminal, como “Git Bash” ou “Command Prompt”.
2. Um novo terminal será criado, e poderá ser usado para executar qualquer comando, incluindo doxygen Doxyfile.

Ou encerrar a tarefa atual, selecionando no canto superior direito da janela do terminal o ícone (+) seguido de “New Terminal”. Digite no Terminal:

```
Shell
doxygen Doxyfile
```

Abra `docs/html/index.html` em um navegador.

## Envio de alterações locais para repositório remoto

Para enviar as alterações para o repositório remoto, sincronize o histórico de *commits*. No canto inferior esquerdo, clique no ícone de sincronização (uma seta para cima ⬆️) ou no botão “Sync Changes”.

Note que o *commit* e o *push* são comandos essenciais no Git, mas eles funcionam em níveis diferentes. Commit salva as alterações no repositório local. Cada commit deve agrupar alterações relacionadas e ter uma mensagem clara que explique o que foi feito. Uma boa prática é fazer *commits* pequenos e frequentes. Push, por sua vez, envia os *commits* do seu repositório local para um repositório remoto (como o GitLab ou GitHub). Isso torna as alterações locais visíveis para a equipe do projeto. Faça o push apenas quando as alterações estiverem estáveis e prontas para serem compartilhadas. É uma boa prática fazer um pull antes de dar o push para garantir que está trabalhando com a versão mais recente do código.

## Desenvolvimento de *hardware* com KiCad e Git

Etapas:

1. Abra o KiCad.
2. Crie um novo projeto eletrônico (Arquivo > Novo Projeto ...) chamado meu_projeto_HW. Ele deve ser salvo na mesma pasta onde está o projeto de firmware (meu_projeto). Isso garantirá que todos os arquivos relacionados fiquem organizados de acordo com a estrutura de pastas a seguir.

```
None
P2/meu_projeto_HW/
├── meu_projeto_HW.kicad_pro       # Projeto KiCad
├── meu_projeto_HW.kicad_sch       # Esquemático
├── meu_projeto.kicad_pcb       # Layout da PCB
├── symbols/       # Componentes personalizados
├── footprints/    # Pegadas dos componentes personalizadas
└── gerbers/           # Arquivos de fabricação
```

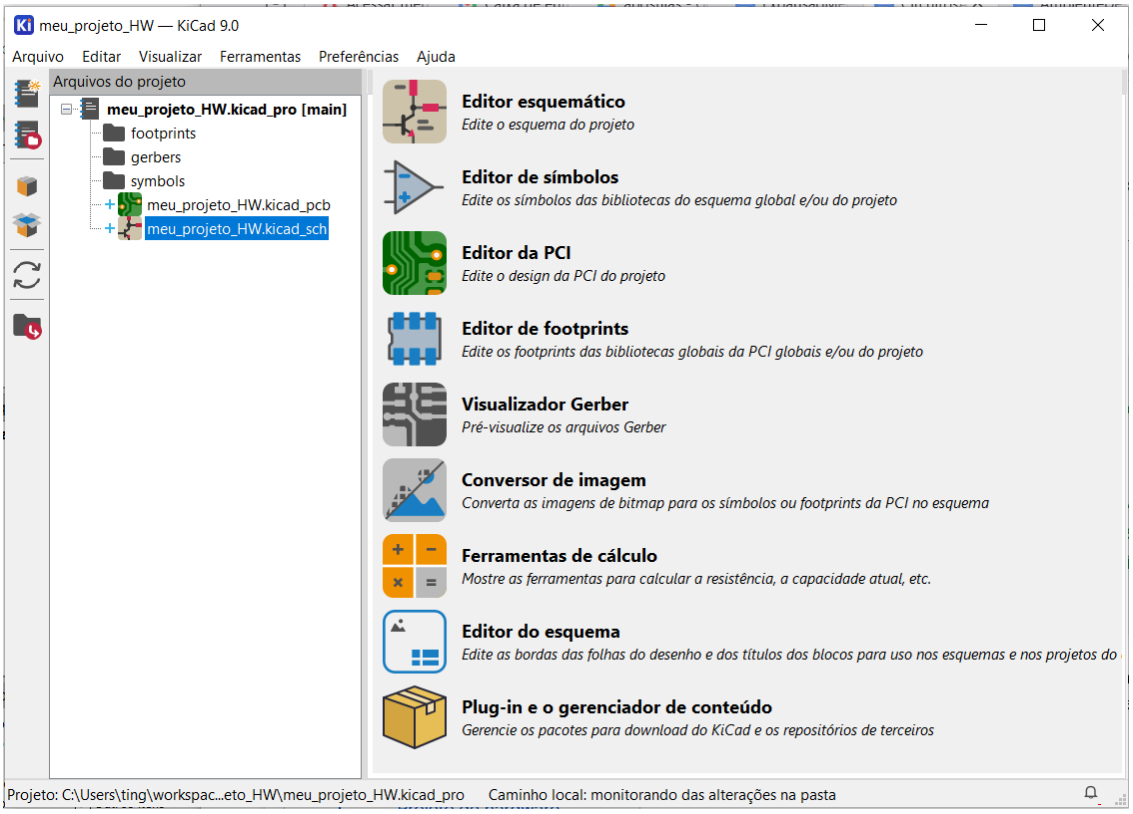
Ao criar um projeto no KiCad, três arquivos principais são gerados automaticamente: .kicad_pro, que gerencia o projeto; .kicad_sch, que contém o esquemático; e .kicad_pcb, que armazena o layout da placa. Para manter uma organização escalável, é uma boa prática adicionar três pastas vazias à raiz do projeto: symbols/, para componentes personalizados; footprints/, para *footprints* customizados; e gerbers/, para armazenar os arquivos de fabricação. Essa estrutura de diretórios permite que separemos os arquivos fonte dos arquivos de saída e dos recursos customizados, facilitando a gestão do projeto e a colaboração.

Para criar a estrutura de pastas vazias em um projeto KiCad usando o Git Bash, pode-se usar os seguintes comandos, garantindo que as pastas sejam adicionadas ao controle de versão, mesmo que estejam vazias.

```
None
    # Abra Git Bash
    # Navegue até a pasta raiz do seu projeto KiCad
    cd /caminho/para/seu/projeto/P2/meu_projeto_HW
```

```
# Crie a pasta 'symbols' e um arquivo .gitkeep para rastreá-la
mkdir symbols
touch symbols/.gitkeep

# Crie a pasta 'footprints' e um arquivo .gitkeep
mkdir footprints
touch footprints/.gitkeep

# Crie a pasta 'gerbers' e um arquivo .gitkeep
mkdir gerbers
touch gerbers/.gitkeep

# Adicione os arquivos criados pelo KiCad, as pastas e os arquivos .gitkeep ao Git
git add .
```

Como resultado, temos a seguinte estrutura de arquivos e pastas reconhecidas pelo KiCad.




3. Ainda na pasta meu_projeto_HW, insira um arquivo .gitignore com as seguintes regras de exclusão recomendadas no *site* de github. A inclusão desse arquivo .gitignore na pasta raiz de um projeto KiCad é uma prática essencial para o controle de versão com Git. O objetivo

principal é instruir o Git a ignorar arquivos temporários, de *cache*, de *backup* e de saída gerados automaticamente pelo KiCad. Esses arquivos são desnecessários no repositório, pois podem ser recriados a partir dos arquivos principais do projeto (.kicad_pcb, .kicad_sch). Ao ignorá-los, o repositório permanece limpo e focado apenas nos arquivos fonte essenciais, evitando o envio de arquivos grandes ou que causem conflitos de merge, facilitando a colaboração entre os membros da equipe. Isso também garante que apenas as mudanças intencionais e significativas no design do hardware sejam rastreadas e versionadas.

None

```
# For PCBs designed using KiCad: https://www.kicad.org/
# Format documentation: https://kicad.org/help/file-formats/

# Temporary files
*.000
*.bak
*.bck
*.kicad_pcb-bak
*.kicad_sch-bak
*-backups
*-cache*
*-bak
*-bak*
*~
~*
_autosave-*
\#auto_saved_files\#
*.tmp
*-save.pro
*-save.kicad_pcb
fp-info-cache
~*.lck
\#auto_saved_files#

# Netlist files (exported from Eeschema)
*.net

# Autorouter files (exported from Pcbnew)
*.dsn
```

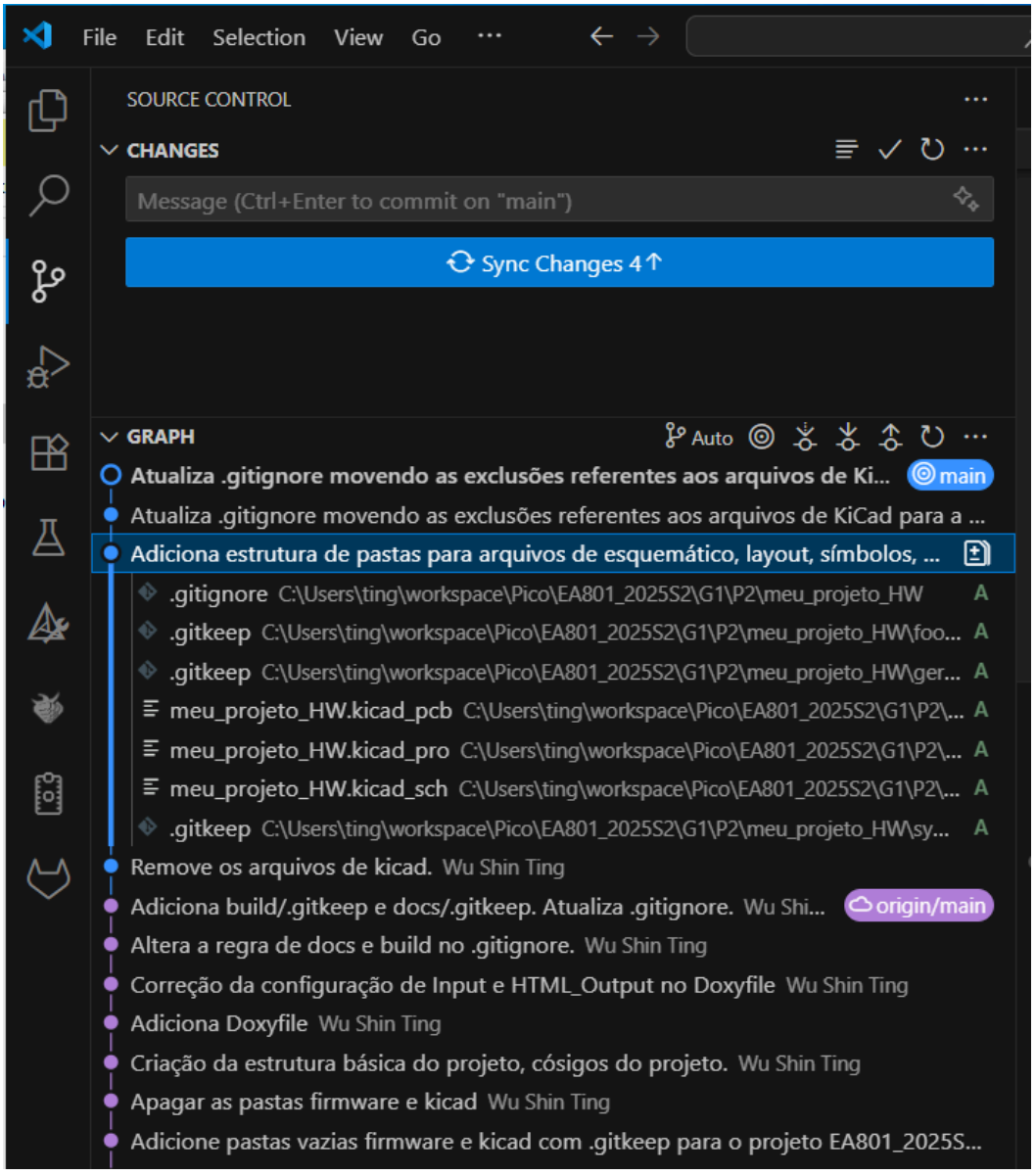
```
*.ses

# Exported BOM files
*.xml
*.csv

# Fabrication files
*.drl
*.dxf
*.gcode
*-BOM.csv
*-gerbers.zip

# Gerbers
*.gbr
*.gtp
*.gts
*.gto
*.gtl
*.gbl
*.gbo
*.gbp
*.gbs

# Archived Backups (KiCad 6.0)
**/*-backups/*.zip
*.kicad_wks-bak

# Local project settings
*.kicad_prl

# Outros
.DS_Store
Thumbs.db
```

Adicione o .gitignore no Git.

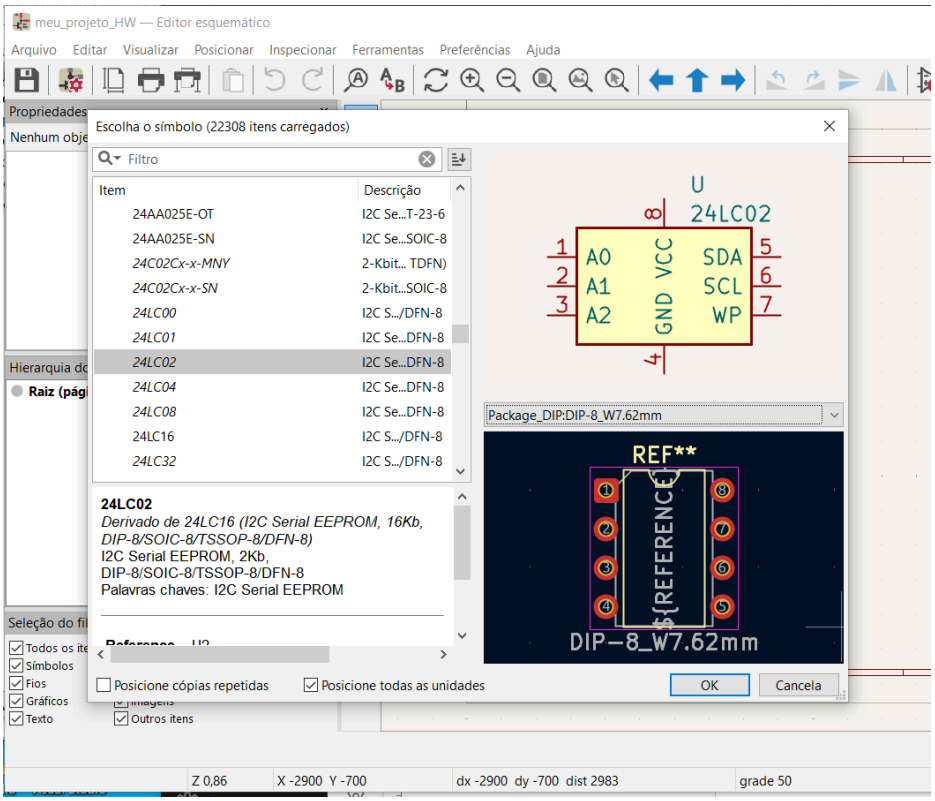
None

```
# Adiciona .gitignore do KiCad no Git no Git Bash
git add .gitignore
```

O KiCad tem um controle de versão nativo limitado, o que dificulta o trabalho em equipe e o rastreamento de mudanças. Para resolver isso, utilize a extensão GitLab Workflow no Visual Studio Code. Depois de configurar a estrutura de pastas e o controle de versão, abra seu projeto de sistema embarcado (por exemplo, o projeto P2) no VS Code. A extensão fornecerá uma interface gráfica para as funcionalidades do Git. Isso permite que gerencie mos commits, pushes e branches de forma visual e intuitiva, versionando todos os arquivos do projeto, incluindo os do KiCad. A imagem abaixo apresenta o `commit` inicial dos arquivos e da estrutura de pastas do projeto KiCad, realizado diretamente no Visual Studio Code.



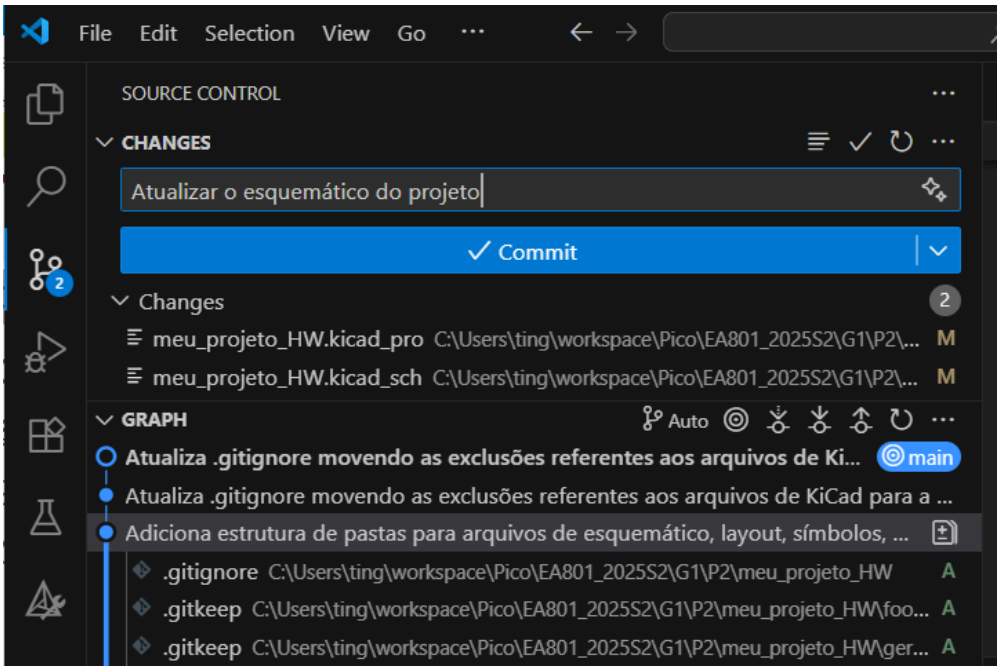
4. Abra o editor de esquemático, clicando em cima do arquivo meu_projeto_HW.kicad.sch. Algumas dicas básicas:
    a. Para inserir um componente, vá em Posicionar > Posicionar Símbolos. Na janela que se abrir, você poderá escolher tanto o símbolo quanto a pegada (em inglês, *footprint*) do componente.




    b. Para que o cursor esteja no modo normal de “Selecione os itens”, pressione a tecla Esc (Escape) para cancelar as ações do modo corrente. Nesse modo, ao clicar com o botão direito do *mouse* em um componente, um menu de contexto será exibido, oferecendo diversas opções, como mover, rotacionar, deletar, e outras operações disponíveis para o item selecionado. Ao clicar o botão esquerdo do *mouse*, páram-se as interações.
    c. Para desenhar linhas, vá em Posicionar > Desenhar os fios. Para outros elementos, selecionar itens correspondentes.
    d. Conectar todas as entradas de energia (símbolos de alimentação e aterramento) ao símbolo PWR_FLAG para sinalizar ao ERC que essas trilhas têm, de fato, uma origem de energia, evitando que o *software* aponte erros falsos.

Uma introdução mais aprofundada ao KiCad está disponível *online*, oferecendo guias detalhados sobre todas as suas funcionalidades.

Todas as modificações foram rastreadas pelo Git. Usando a interface do GitLab Workflow no VS Code, basta fazer o `commit` e, em seguida, o `push` para enviar as alterações ao repositório remoto. É uma boa prática fazer um `pull` antes de enviar as alterações.




5. Executar verificação elétrica (ERC), clicando no ícone “Verificador das Regras Elétricas” na barra de ferramentas superior. Isso irá analisar seu esquemático em busca de erros de conexão.

6. Gerar a *netlist*, clicando no ícone “Gerar a lista de materiais (BOM) …” na barra de ferramentas superior. Isso irá criar a lista de conexões elétricas do seu projeto.

## Boas Práticas

- Não versionar arquivos gerados. Use `.gitignore`.
- Comente funções com Doxygen.
- Use branches para novas features ou correções.
- Commits pequenos e frequentes.
- Documente o hardware no repositório (PDFs, prints).
- Evite caminhos absolutos nos arquivos do KiCad.