Roteiro 1

Ana Beatriz Barbosa Yoshida - RA: 245609
Julio Nunes Avelar - RA: 241163

Agosto de 2025

Sumário

1 Experiência 1

1.1 1.1 Identificação das GPIOs do LED RGB

Pergunta: Identifique as GPIOs que estão conectadas no LED RGB da BitDogLab.

Resposta: As conexões do LED RGB com resistores de proteção estão descritas a seguir:

- Vermelho: GPIO 13 com resistor de 220 Ω
- Verde: GPIO 11 com resistor de 220 Ω
- Azul: GPIO 12 com resistor de 150 Ω

1.2 1.2 Níveis lógicos do RP2040

Pergunta: Qual a tensão de saída de cada nível lógico do RP2040?

Resposta: Os níveis lógicos do microcontrolador RP2040 são:

- Nível lógico 0: 0 V
- Nível lógico 1: 3.3 V

1.3 1.3 Circuito básico e cálculo dos resistores

Pergunta: Desenhe o circuito básico para acender este LED com as GPIOs. Calcule o valor de cada resistor.

Resposta: O circuito básico do LED RGB está ilustrado na Figura 1.

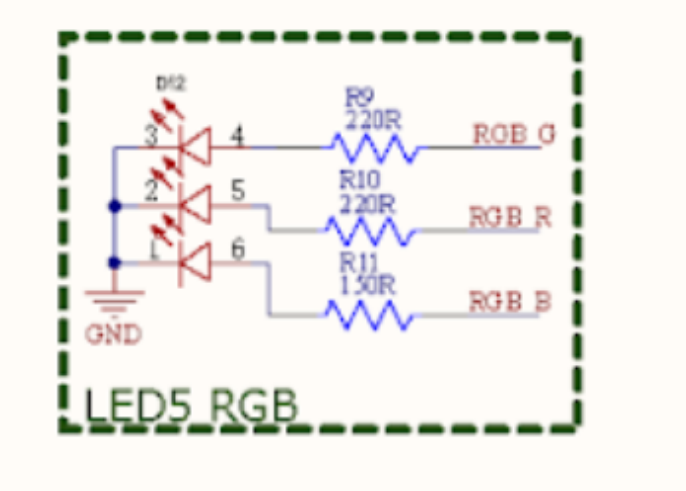

Figura 1: Circuito do LED RGB com resistores.

O cálculo dos resistores é feito pela Lei de Ohm:

$$R = \frac{V_{CC} - V_{LED}}{I_{LED}}$$

Os valores de corrente e tensão utilizados para cada cor do Led RGB podem ser encontrados no datasheet do Led

1.4 Tarefa 1.1 – Comparação entre linguagens

Pergunta: Comparar o tempo de resposta e fluxo de execução entre C e MicroPython. Sugira um método para obter esses valores e crie uma tabela para realizarmos um Benchmark.

Resposta: Para realizar o Benchmark entre as duas implementações (C e MicroPython), podemos analisar o tempo gasto para executar rotinas simples, como configuração de GPIOs, execução de funções básicas (acender um LED, imprimir uma string) e cálculos matemáticos. Os resultados obtidos estão apresentados na Tabela 1.4.

Linguagem	Tempo (μs)	Teste
C	13	Inicialização das GPIOs
MicroPython	452	Inicialização das GPIOs
C	10010890	Loop com blink (10 iterações)
MicroPython	10000689	Loop com blink (10 iterações)
C	246757	Verificação de primos
MicroPython	1226509	Verificação de primos

Tabela 1: Benchmark básico entre C e MicroPython.

1.5 Tarefa 1.2 – Comparativo Imperativo vs OO

Pergunta: Comparar tamanho do código (em bytes), tempo de resposta e fluxo de execução entre os modos imperativo e OO para ambos: MicroPython e C. Sugira um método para obter esses valores e crie uma tabela para realizarmos um Benchmark.

Resposta: Uma comparação direta do tamanho do código em bytes não é apropriada, pois se trata de linguagens e paradigmas diferentes. Técnicas de escrita distintas podem resultar em códigos mais ou menos verbosos. No caso do C, a implementação de OO (via ponteiros de funções, structs com métodos ou C++) é amplamente otimizada pelo compilador, reduzindo overhead em tempo de execução.

Ainda assim, é possível avaliar a diferença de desempenho comparando implementações imperativas e orientadas a objetos quanto ao tempo de resposta e eficiência. Essa análise está resumida na Tabela 1.5.

Paradigma	Linguagem	Tamanho Código (bytes)	Tempo de Resposta (μs)	Observações
Imperativo	C	...	...	...
OO	C	...	...	...
Imperativo	Python	...	...	...
OO	Python	...	...	...

Tabela 2: Benchmark entre Imperativo e OO em C e Python.

2 Experiência 2

2.1 2.1 GPIOs conectados aos botões

Pergunta: Identifique as GPIOs que estão conectadas aos 3 botões.
Resposta: A identificação está apresentada na Tabela 3.

Botão	IO	Função
RESET	RUN	Reset do Pi Pico
Botão A	GP05	Botão genérico
Botão B	GP06	Botão genérico
Botão C	GP10	Botão genérico

Tabela 3: Mapeamento dos botões conectados ao RP2040.

2.2 2.2 Limites de tensão para nível lógico

Pergunta: Quais os limites de tensão considerados como nível baixo e alto no microcontrolador utilizado?

Resposta:

- Nível baixo: $V < V_{IL(max)}$
- Nível alto: $V > V_{IH(min)}$

Com $V_{IL(max)}$ sendo 0.8V e $V_{IH(min)}$ sendo 2.0V.

2.3 2.3 Esquema dos botões

Pergunta: Represente como os botões estão conectados ao microcontrolador.

Resposta: O esquema está mostrado na Figura 2.

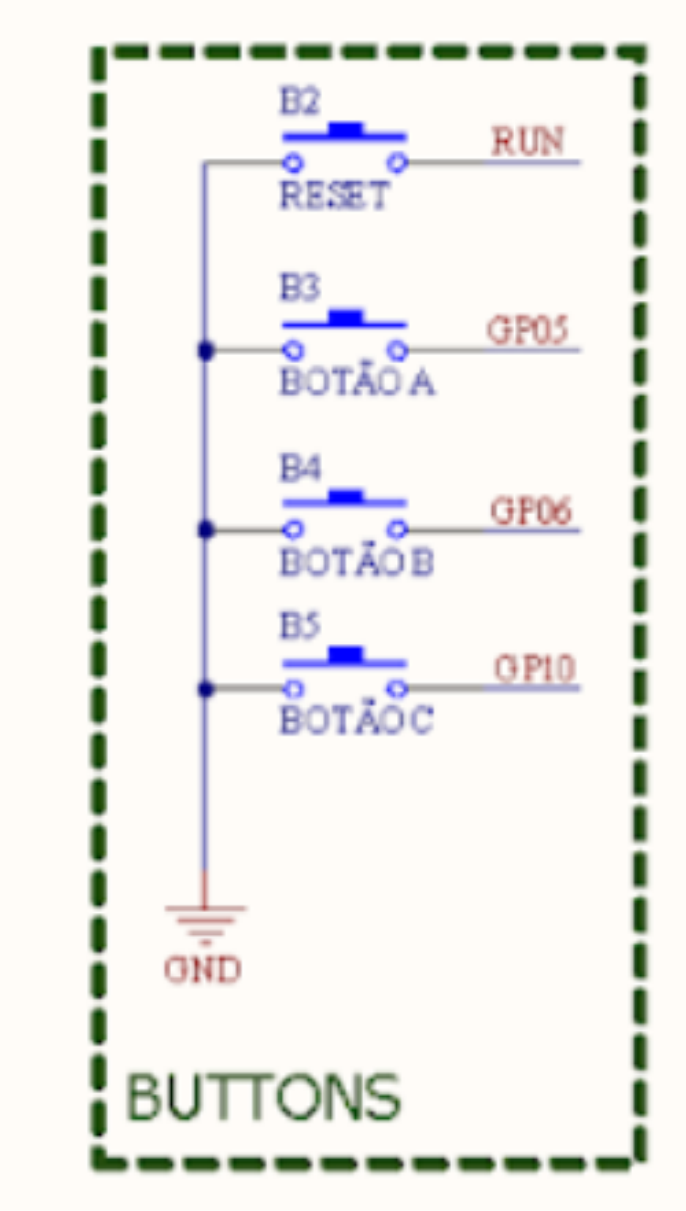

Figura 2: Conexão dos botões ao RP2040.

2.4 Debounce, Polling e IRQ

Funcionalidade: Confirmar a alternância do LED nos dois modos.

Latência: O aumento da carga no modo Polling provoca maior latência, enquanto o uso de IRQ garante resposta mais imediata.

Consumo de energia: O IRQ é mais eficiente, pois evita que a CPU fique em espera constante.

Aplicações: Polling pode ser suficiente em sistemas simples (ex.: leitura periódica de sensores), enquanto IRQ é essencial em sistemas críticos (ex.: teclados, comunicações seriais).

2.5 Tabela Comparativa – Polling × IRQ

A Tabela 4 resume a comparação entre Polling e IRQ.

Critério	Polling	IRQ
Latência percebida	Maior	Menor
Perdas de eventos	Possível	Improvável
Consumo de CPU	Elevado	Reduzido
Complexidade	Baixa	Maior
Situações suficientes	Sistemas simples	—
Situações obrigatórias	—	Sistemas críticos

Tabela 4: Comparativo entre Polling e IRQ.

3 Experiência 3

3.1 Demonstração

Vídeo: Disponível no YouTube em: `https://youtube.com/xxxxxx` #EA701.

3.2 Comparação de código

Comparar a simplicidade entre implementações em Python OO e em C modular, destacando clareza e manutenção do código.

3.3 Reflexão: IRQ + OO

Discutir casos em que a combinação de IRQ e paradigma OO é vantajosa em sistemas embarcados reais (ex.: sistemas de tempo real, IoT e automação).

3.4 Expansão para 10 botões

Discutir a viabilidade de Polling versus IRQ em um sistema com 10 botões. O Polling aumenta consideravelmente o consumo e a latência, enquanto o IRQ se mostra mais adequado.

4 Conclusão

Neste roteiro, foram explorados os conceitos de GPIOs, LEDs RGB, botões, limites de tensão lógica, técnicas de tratamento de eventos (Polling e IRQ) e comparações entre paradigmas de programação (imperativo e OO) em C e Python.

Os experimentos demonstraram diferenças significativas de desempenho entre C e MicroPython, além da importância da escolha correta entre Polling e IRQ de acordo com a aplicação. Como recomendação, projetos futuros devem considerar o uso de IRQ em sistemas mais complexos ou com múltiplos eventos assíncronos, e a escolha de linguagem/paradigma deve levar em conta tanto desempenho quanto legibilidade e manutenção do código.