Blink:

Aula: Blink - Piscar um LED com a BitDogLab

Introdução:

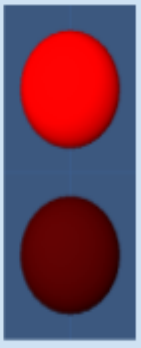
Nesta aula, vamos aprender a usar o exemplo básico do *Blink*, uma prática fundamental no desenvolvimento de sistemas embarcados. Este exemplo consiste em controlar o piscar de um LED presente na BitDogLab, utilizando um GPIO (*General Purpose Input/Output*) configurado como saída digital. Vamos alternar o estado do LED, entre ligado e desligado, em intervalos de tempo definidos pelo nosso programa.

Desenvolvimento:

Para controlar periféricos externos ao microcontrolador, precisamos compreender alguns detalhes de *hardware* que permitem que nossos objetivos sejam atingidos. Quando trabalhamos com sistemas embarcados, é fundamental compreender como os pinos de um microcontrolador são organizados e mapeados em diferentes módulos e placas. No caso do RP2040, microcontrolador utilizado no módulo Raspberry Pi Pico e na BitDogLab, a identificação dos GPIOs segue uma convenção consistente, que facilita a programação e a integração de hardware. O RP2040 possui 30 pinos de GPIOs (*General Purupose Input/Output* ou ou Entrada/Saída de Propósito Geral) numerados de GPIO0 a GPIO29. Esses números correspondem diretamente aos pinos físicos do microcontrolador. No módulo Raspberry Pi Pico, essa numeração é mantida e disponibilizada nas bordas da placa, sem alterações. Assim, um GPIO específico no RP2040 recebe o mesmo número identificador no módulo Pico, e também mantém-se o mesmo identificador na BitDogLab.

No caso do *Blink*, vamos acionar um LED RGB em que o pino do azul está conectado ao GPIO 12 da BitDogLab. Ao configurarmos esse pino em nível lógico alto, o LED acende na cor azul; em nível lógico baixo, ele apaga.

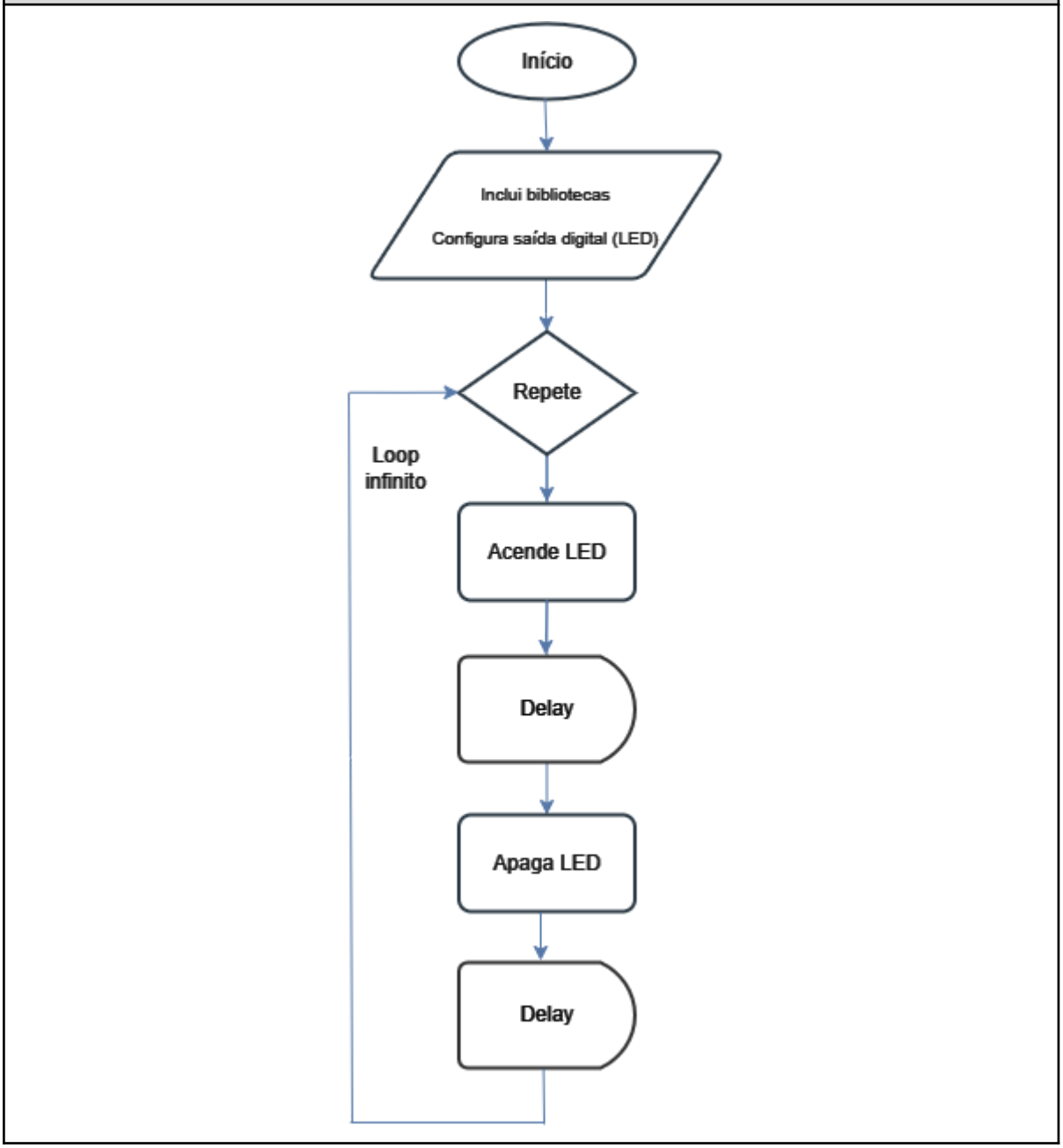
Níveis Lógicos:
Os níveis lógicos (alto e baixo) são tensões que definimos nos pinos de controle. Na Raspberry Pi Pico, o nível lógico alto corresponde a aproximadamente 3,3V, e o nível baixo a 0V. Manipulando esses níveis usando um programa. Fisicamente este nível de tensão estará presente em um pino do microcontrolador, chamado GPIO. Desta forma, conseguimos controlar o estado (ligado ou desligado) do LED comandando os níveis lógicos nas GPIOs correspondentes.



A partir destas informações, podemos criar um fluxograma: um diagrama mostrando passo a passo os eventos realizados pelo sistema embarcado durante a execução do programa *blink*. Perceba que o fluxograma estabelece uma sequência, executando um passo de cada vez, com uma organização lógica. Neste fluxograma, cada figura geométrica segue uma convenção, porém – neste momento – não é o foco do nosso contexto detalhar esta convenção. O fluxograma parte do início, inclui bibliotecas e configura a saída (pino no

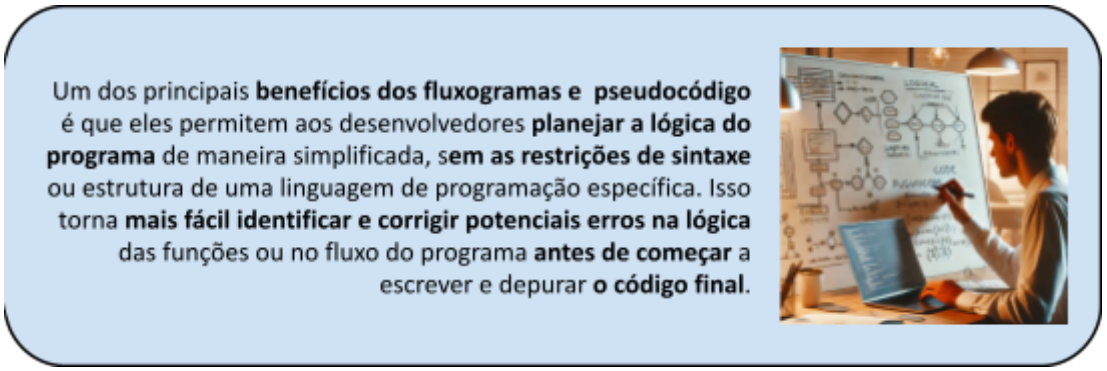
LED). Observe que neste fluxograma não há entrada de dados. Na sequência, o fluxograma entra num *loop* que se repete indefinidamente acendendo e apagando o LED. Os blocos de *Delays* são necessários para que possamos ver o processo de acende e apaga do LED. Caso contrário, este processo seria muito rápido e não conseguiríamos perceber que o LED está piscando.

Fluxograma



A partir deste fluxograma, podemos destacar e detalhar as seguintes etapas principais num novo documento chamado de pseudocódigo:

1. **Início**: Representa o ponto de partida do programa.
2. **Configuração de Inicialização**: O programa começa incluindo bibliotecas e configurando a saída digital que será estruturada para fazer o LED piscar.
3. **Loop Infinito**: Uma estrutura de repetição (*loop*) onde o programa executa repetidas vezes a ação de acender e apagar o LED, com um certo tempo de espera entre as ações.
    a. **Acende LED**: Configura-se o nível lógico de saída do GPIO 12 como alto, fazendo o LED acender na cor azul.
    b. **Delay**: O programa fica inativo por um certo tempo, com o LED permanecendo aceso neste período.
    c. **Apaga LED**: Em seguida, o programa configura o nível lógico de saída do GPIO 12 como baixo, fazendo o LED apagar.
    d. **Delay**: O programa fica inativo novamente, e o LED fica apagado enquanto isso.
4. **Retorno ao Início do Loop**: Ao fim do processo abcd, o programa retorna ao início do *loop*, onde ele acende novamente o LED e prossegue com os outros passos, repetindo o ciclo indefinidamente.



Na etapa seguinte, podemos transcrever o pseudocódigo em um programa, escolhendo alguma linguagem específica, respeitando sua sintaxe e convenções. Esta construção pode ser realizada etapa por etapa, respeitando a sequência do pseudocódigo, ou então pode ser reaproveitada de um repositório. Normalmente é aqui que o desenvolvedor investe um certo tempo, depurando o código e testando, até que o resultado atenda às suas expectativas.

Exemplo de Código em C para o Blink:
Abaixo está um código em C para a Raspberry Pi Pico que alterna o estado do LED a cada segundo. Este exemplo utiliza a biblioteca padrão do SDK do Raspberry Pi Pico para configurar o GPIO e controlar o LED.

Explicação do Pseudocódigo

1. **Definições e Configurações**: Define-se o tempo de atraso e o pino do LED.
2. **Função de Inicialização (`led_init`)**: Configura o GPIO para controlar o LED e verifica se a inicialização foi bem-sucedida.

3. **Função de Controle (`set_led`)**: Liga ou desliga o LED conforme o valor de entrada.
4. **Loop Principal**: Alterna continuamente entre ligar e desligar o LED, com intervalos definidos por `LED_DELAY_MS`.

Pseudocódigo	Linguagem C
Início: Representa o ponto de partida do programa. Para começar, incluímos um comentário como cabeçalho.	`/**` ` * Embarcatech` ` * Exemplo Blink com a BitDogLab` ` */`
Inclusão de Biblioteca: Inclui a biblioteca "pico/stdlib.h" para acessar funções de GPIO e temporização.	`#include "pico/stdlib.h"`
Definições e Configurações: Define o pino do LED onde ele está conectado (exemplo: GPIO 12). Inicializa o pino do LED e configura-o como saída.	`int main() {` `    // Define o pino do LED e o configura como saída` `    const uint LED_PIN = 12;  // Substitua com o número do GPIO desejado` `    gpio_init(LED_PIN);` `    gpio_set_dir(LED_PIN, GPIO_OUT);`
Loop Infinito: Define a condição, que é sempre verdadeira, para repetir o *loop*. Liga o LED.	`    while (true) {` `        // Liga o LED` `        gpio_put(LED_PIN, true);` `        sleep_ms(250);  // Espera 250ms`

Aguarda pelo tempo de atraso e define para 250ms. Desliga o LED. Aguarda pelo tempo de atraso. Fim	``` // Desliga o LED gpio_put(LED_PIN, false); sleep_ms(250); // Espera 250ms } }```

Código completo em C

```c
/**
 * Embarcatech
 * Exemplo Blink com a BitDogLab
 */

#include "pico/stdlib.h"

int main() {
    // Define o pino do LED e o configura como saída
    const uint LED_PIN = 12;  // Substitua com o número do GPIO desejado
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);

    while (true) {
        // Liga o LED
        gpio_put(LED_PIN, true);
        sleep_ms(250);  // Espera 250ms

        // Desliga o LED
        gpio_put(LED_PIN, false);
        sleep_ms(250);  // Espera 250ms
    }
}
```

Antes de validarmos o nosso algoritmo, precisamos organizar algumas configurações essenciais para que o programa possa ser compilado e executado corretamente na BitDogLab. Essas configurações são realizadas em um arquivo especial chamado **CMakeLists.txt**, que é utilizado pelo sistema de build CMake.

O **CMake** é uma ferramenta que automatiza o processo de configuração e geração dos arquivos necessários para compilar o código-fonte em diferentes plataformas. No caso da Raspberry Pi Pico, o arquivo **CMakeLists.txt** permite que integremos o SDK do Pico, vinculando as bibliotecas adequadas e gerando o arquivo final no formato `.uf2`, que será carregado na placa.

A seguir, detalharemos como este arquivo é estruturado e quais são os passos necessários para utilizá-lo no projeto. Isso garantirá que o ambiente de desenvolvimento esteja devidamente configurado e pronto para compilar o programa que criamos.

Arquivo CMake

Além do arquivo em C (extensão .c), é necessário configurar um arquivo CMake para compilar e executar o programa no microprocessador. Este arquivo define as configurações de *build* do projeto, como as bibliotecas que serão usadas, os arquivos de origem e as especificações do sistema.

Aqui está um exemplo básico do CMakeLists.txt para nosso projeto do *blink*:

CMakeLists.txt

```
# Define a versão mínima do CMake que será utilizada
cmake_minimum_required(VERSION 3.13)

# Define o nome do projeto
project(blink C CXX ASM)

# Define a versão do padrão C a ser utilizada
set(CMAKE_C_STANDARD 11)  # Uso do C11
set(CMAKE_CXX_STANDARD 17)  # Caso utilize C++ no projeto futuro

# Inclui o SDK do Raspberry Pi Pico
# Essa variável deve ser configurada previamente no ambiente
include($ENV{PICO_SDK_PATH}/pico_sdk_init.cmake)

# Inicializa o SDK do Raspberry Pi Pico
pico_sdk_init()

# Adiciona o arquivo principal ao projeto
add_executable(blink
    main.c  # Substitua pelo nome do arquivo se diferente
```

```
)

# Alinha o projeto com as bibliotecas padrão da Pico
target_link_libraries(blink pico_stdlib)

# Define o nome da placa (pico ou pico_w, conforme o hardware)
pico_add_extra_outputs(blink)

# Adiciona suporte às bibliotecas necessárias
# A função pico_enable_stdio_* habilita saída pela UART ou USB
pico_enable_stdio_uart(blink 1)  # Habilita saída pela UART (1 = ativo)
pico_enable_stdio_usb(blink 0)   # Desativa saída pela USB (0 = inativo)

# Comentário: Essa configuração faz o link com as bibliotecas básicas
# e configura o suporte de entrada/saída para o projeto.
```

Explicação dos principais comandos do CMAKE:

1. `cmake_minimum_required`: **Garante que o CMake usado seja compatível com os comandos do projeto.**
2. `project`: **Define o nome do projeto.**
3. `set`: **Configura os padrões de C e C++.**
4. `include`: **Inclui os arquivos do SDK do Raspberry Pi Pico, necessários para compilar o projeto.**
5. `pico_sdk_init`: **Inicializa o SDK.**
6. `add_executable`: **Especifica o nome do executável e os arquivos de código-fonte que farão parte do projeto.**
7. `target_link_libraries`: **Vincula as bibliotecas padrão necessárias, como `pico_stdlib`.**
8. `pico_add_extra_outputs`: **Gera os arquivos necessários para gravar o programa na placa (ex.: `.uf2`).**
9. `pico_enable_stdio_*`: **Controla a saída de depuração, permitindo exibir logs pela UART ou USB.**

Em seguida:

1. Salve este conteúdo como CMakeLists.txt no mesmo diretório do arquivo blink.c.
2. Configure o ambiente do SDK do Pico.
3. Compile o projeto com os comandos no terminal:

```
mkdir build

cd build

cmake ..

Make
```

4. O arquivo .uf2 estará disponível no diretório *build*. Copie-o para a Pico conectada ao seu computador.

Um pouco mais sobre o hardware da BitDogLab:

Um **LED RGB,** cátodo comum, tem o eletrodo do vermelho lidado no **GPIO 13** através de um resistor de 220 ohm, o pino de verde está ligado no **GPIO 11** também através de um resistor de 220 ohm e o pino do azul no **GPIO 12** através de um resistor de 150 ohm.

Exercício:

Faça uma adaptação neste código em C de forma que o programa pisque o LED na cor vermelha e também na cor verde. Comece pelo fluxograma, depois modifique o pseudocódigo, e por fim implemente e depure seu código em C. Faça um vídeo, de no máximo 15 segundos, mostrando seu funcionamento e carregue no Moodle.