Desenvolvimento de Sistemas Embarcados em Tempo Real

Prof. Hermano Cabral

Departamento de Eletrônica e Sistemas — UFPE

18 de julho de 2024

Plano de Aula

Tema central

- Chibios — Hardware Abstraction Layer

Plano de Aula

Tema central

- Chibios — Hardware Abstraction Layer

Objetivos

- Conhecer as características dos principais módulos do HAL do ChibiOS

ChibiOS - Introdução

Introdução

- O ChibiOS é um RTOS de código aberto: http://www.chibios.org.

ChibiOS - Introdução

Introdução

- O ChibiOS é um RTOS de código aberto: http://www.chibios.org.
- Pronunciamos em português como Quibi-Ó-S.

ChibiOS - Introdução

Introdução

- O ChibiOS é um RTOS de código aberto: http://www.chibios.org.
- Pronunciamos em português como Quibi-Ó-S.
- A grande vantagem do ChibiOS é sua arquitetura elegante e o suporte a diferentes hardwares.

Introdução

- A versão mais nova do ChibiOS (atualmente, a 21.11.3) pode ser baixada no link https://osdn.net/projects/chibios/releases.

Introdução

- A versão mais nova do ChibiOS (atualmente, a 21.11.3) pode ser baixada no link https://osdn.net/projects/chibios/releases.
- O arquivo deve ser descompactado em uma pasta à escolha do usuário.

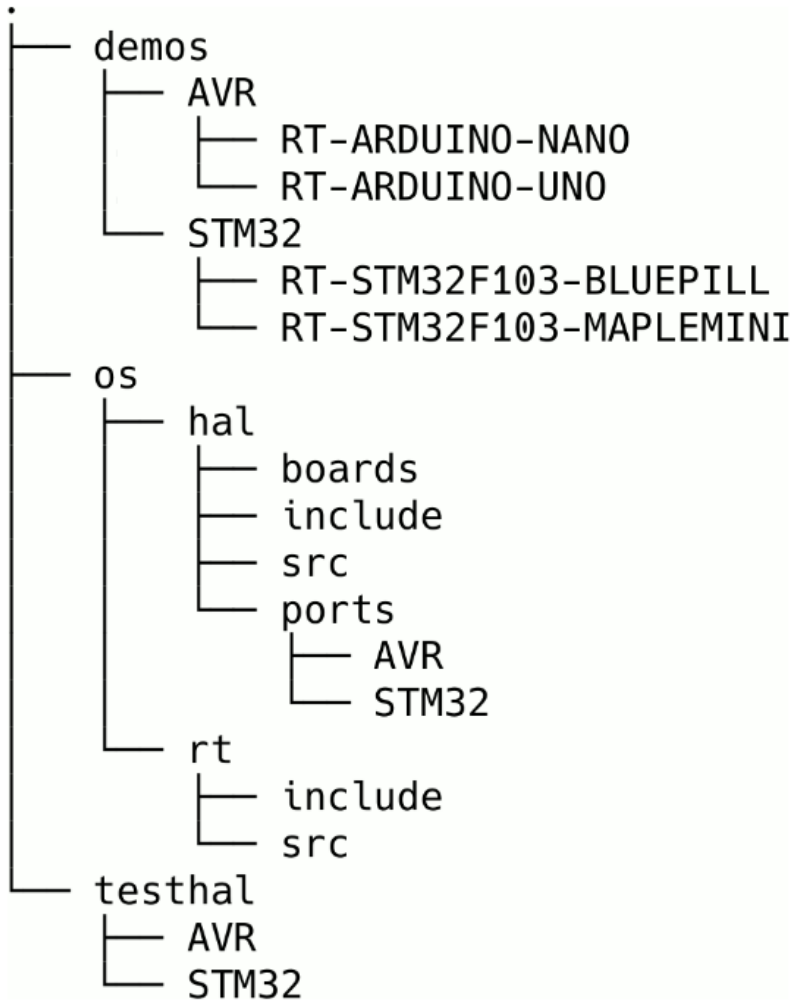
Introdução

- A versão mais nova do ChibiOS (atualmente, a 21.11.3) pode ser baixada no link https://osdn.net/projects/chibios/releases.
- O arquivo deve ser descompactado em uma pasta à escolha do usuário.
- Dentro da pasta do ChibiOS, as principais subpastas são:
  - os/hal: contém o código para a HAL
  - os/rt: contém o código para o kernel
  - demos: contém programas de demonstração

Introdução

- A pasta de demos e a da HAL são divididas em várias arquiteturas de hardware, incluindo o Stm32 e o AVR.

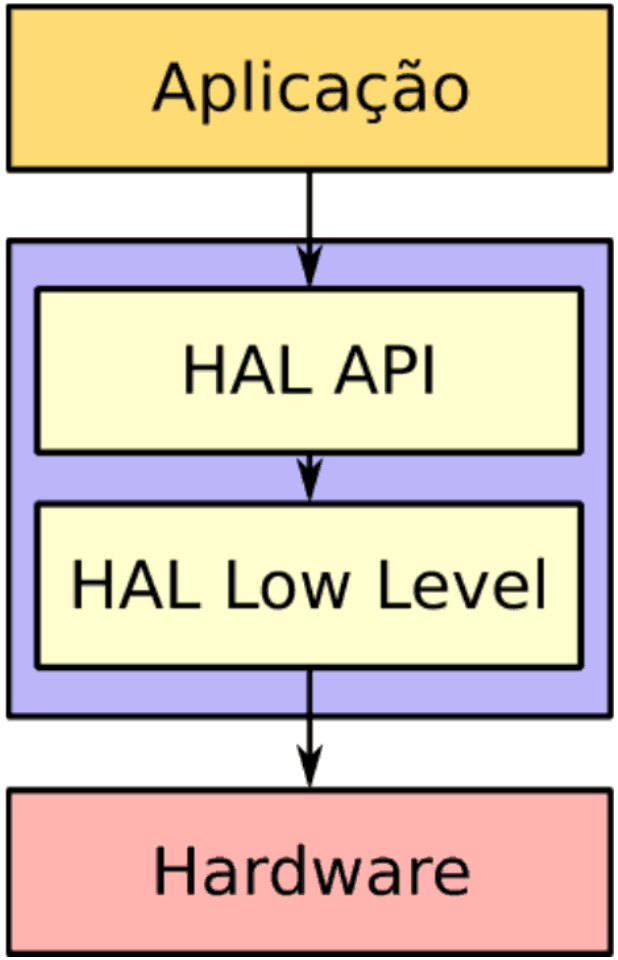
ChibiOS - Estrutura



ChibiOS - Camada de Abstração de Hardware

Introdução

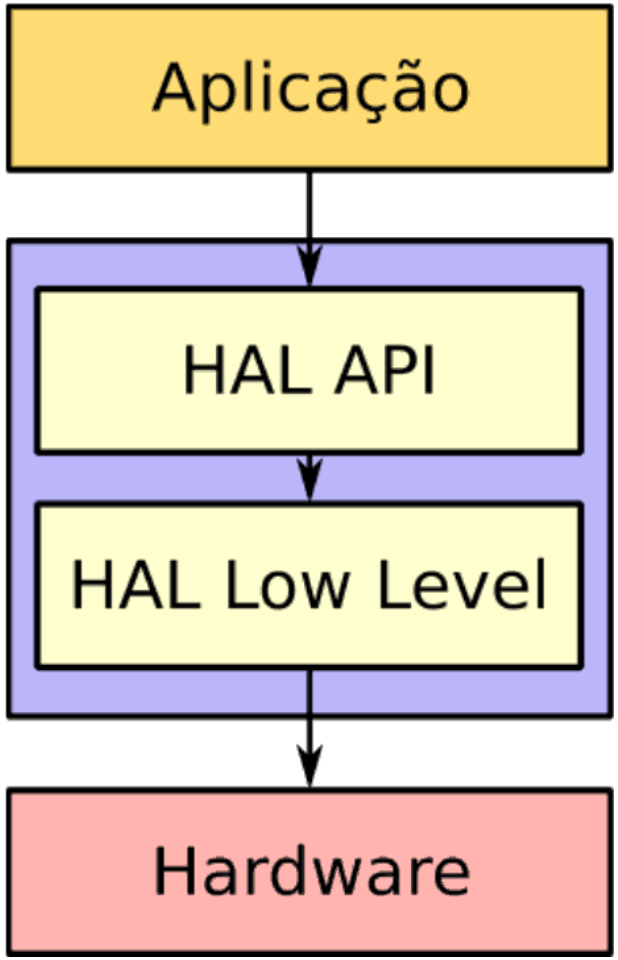
- A Camada de Abstração de Hardware - HAL - permite que uma aplicação programe de uma forma quase independente do hardware.



ChibiOS - Camada de Abstração de Hardware

Introdução

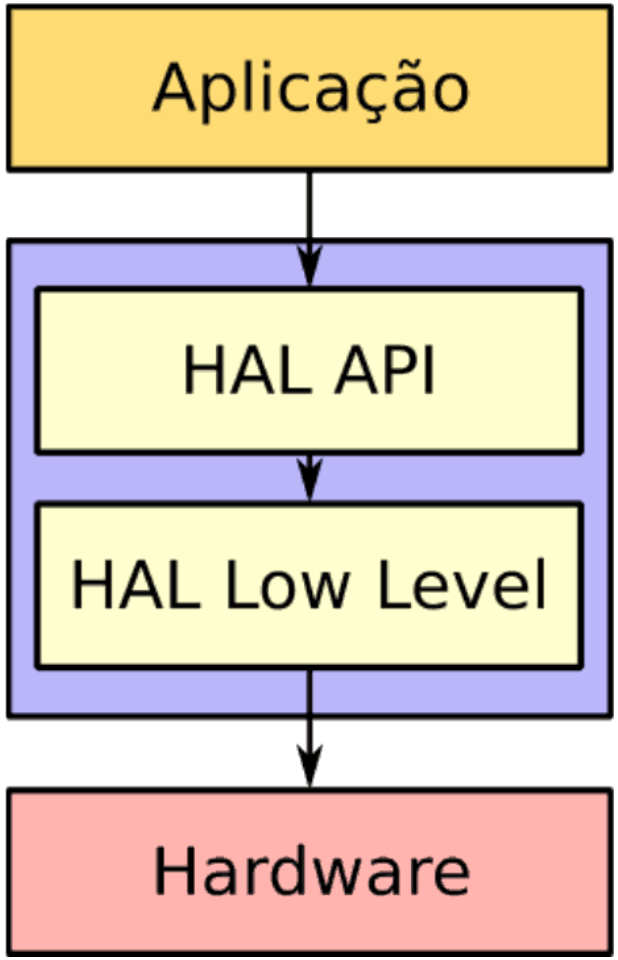
- A Camada de Abstração de Hardware - HAL - permite que uma aplicação programe de uma forma quase independente do hardware.
- Isto é possível pela criação de uma interface de programação - API



ChibiOS - Camada de Abstração de Hardware

Introdução

- A Camada de Abstração de Hardware - HAL - permite que uma aplicação programe de uma forma quase independente do hardware.
- Isto é possível pela criação de uma interface de programação - API
- A HAL lida com o hardware através de uma subcamada de baixo nível, que é a única que precisa ser modificada ao se trocar o hardware



Observações

- A subcamada HAL API é projetada o mais abrangente possível de forma a lidar com a maior quantidade de hardware possível.

Observações

- A subcamada HAL API é projetada o mais abrangente possível de forma a lidar com a maior quantidade de hardware possível.
- A subcamada LLD realiza os passos necessários para a configuração e uso do hardware específico.

Observações

- A subcamada HAL API é projetada o mais abrangente possível de forma a lidar com a maior quantidade de hardware possível.
- A subcamada LLD realiza os passos necessários para a configuração e uso do hardware específico.
- Iremos estudar a subcamada HAL API do RTOS ChibiOS.

Periféricos

- A HAL abrange os periféricos de todos os microcontroladores que ela dá suporte.

Periféricos

- A HAL abrange os periféricos de todos os microcontroladores que ela dá suporte.
- Veremos nesta disciplina apenas os principais:
  - Pinos
  - GPIO
  - Temporizadores
  - Captura de entradas (Input Capture)
  - PWM

Arquivos de configuração

- Uma aplicação desenvolvida para o ChibiOS terá 3 arquivos de configuração:

Arquivos de configuração

- Uma aplicação desenvolvida para o ChibiOS terá 3 arquivos de configuração:
  - mcuconf.h
  - halconf.h
  - chconf.h

Arquivos de configuração

- Uma aplicação desenvolvida para o ChibiOS terá 3 arquivos de configuração:
  - mcuconf.h
  - halconf.h
  - chconf.h
- Nestes arquivos de configuração, diversas funcionalidades podem ser inclusas ou não
  - Caso não precisemos de uma funcionalidade, podemos não incluí-la na geração do programa para economia da memória flash

Arquivos de configuração

- Além desses arquivos de configuração, temos o arquivo Makefile para compilar o programa.

Arquivos de configuração

- Além desses arquivos de configuração, temos o arquivo Makefile para compilar o programa.
- Os arquivos mcuconf.h e Makefile dependem do microcontrolador e da placa usada

Arquivos de configuração

- Além desses arquivos de configuração, temos o arquivo Makefile para compilar o programa.
- Os arquivos mcuconf.h e Makefile dependem do microcontrolador e da placa usada
- Os arquivos halconf.h e chconf.h tem pouca ou nenhuma dependẽncia com o microcontrolador

Uso

- Para desenvolvermos um projeto, o mais simples é copiar a pasta apropriada e modificar apenas o que precisamos.

Uso

- Para desenvolvermos um projeto, o mais simples é copiar a pasta apropriada e modificar apenas o que precisamos.
- Para padronizar, crie uma pasta com o nome “apps” dentro da pasta do ChibiOS, e dentro desta pasta crie uma pasta com o nome AVR ou STM32, a depender do MCU que você utilizar

Uso

- Para desenvolvermos um projeto, o mais simples é copiar a pasta apropriada e modificar apenas o que precisamos.
- Para padronizar, crie uma pasta com o nome "apps" dentro da pasta do ChibiOS, e dentro desta pasta crie uma pasta com o nome AVR ou STM32, a depender do MCU que você utilizar
- Dentro desta pasta, crie uma pasta para cada programa que você fizer usando o ChibiOS, copiando tudo o que houver dentro da pasta apropriada na pasta demos
  - Esta será a pasta a ser usada para gerar o programa

Arquivo mcuconf.h

- A primeira coisa que configuramos é o arquivo mcuconf.h.

Arquivo mcuconf.h

- A primeira coisa que configuramos é o arquivo mcuconf.h.
- Este arquivo define que periféricos do MCU que queremos usar.

Arquivo mcuconf.h

- A primeira coisa que configuramos é o arquivo mcuconf.h.
- Este arquivo define que periféricos do MCU que queremos usar.
- Exemplos de configuração são:
  - `#define AVR_EXT_USE_INT0 FALSE`
  - `#define AVR_PWM_USE_TIM1 FALSE`
  - `#define AVR_SERIAL_USE_USART0 TRUE`
  - `#define AVR_I2C_USE_I2C1 FALSE`

Arquivo mcuconf.h

- A primeira coisa que configuramos é o arquivo mcuconf.h.
- Este arquivo define que periféricos do MCU que queremos usar.
- Exemplos de configuração são:
  - `#define AVR_EXT_USE_INT0 FALSE`
  - `#define AVR_PWM_USE_TIM1 FALSE`
  - `#define AVR_SERIAL_USE_USART0 TRUE`
  - `#define AVR_I2C_USE_I2C1 FALSE`
- Se quisermos usar o periférico, colocamos o valor `TRUE`

Arquivo mcuconf.h

- A primeira coisa que configuramos é o arquivo mcuconf.h.
- Este arquivo define que periféricos do MCU que queremos usar.
- Exemplos de configuração são:
  - `#define AVR_EXT_USE_INT0 FALSE`
  - `#define AVR_PWM_USE_TIM1 FALSE`
  - `#define AVR_SERIAL_USE_USART0 TRUE`
  - `#define AVR_I2C_USE_I2C1 FALSE`
- Se quisermos usar o periférico, colocamos o valor `TRUE`
- Caso contrário, colocamos o valor `FALSE`

Arquivo halconf.h

- Depois do mcuconf.h, configuramos o halconf.h, que também depende do MCU utilizado.

Arquivo halconf.h

- Depois do mcuconf.h, configuramos o halconf.h, que também depende do MCU utilizado.
- Este arquivo define os módulos do HAL e suas características.

Arquivo halconf.h

- Depois do mcuconf.h, configuramos o halconf.h, que também depende do MCU utilizado.
- Este arquivo define os módulos do HAL e suas características.
- Exemplos de configuração são:
  - `<3->#define HAL_USE_PAL TRUE`
  - `<3->#define HAL_USE_PWM FALSE`
  - `<3->#define HAL_USE_SERIAL TRUE`
  - `<3->#define HAL_USE_I2C FALSE`

Arquivo halconf.h

- Depois do mcuconf.h, configuramos o halconf.h, que também depende do MCU utilizado.
- Este arquivo define os módulos do HAL e suas características.
- Exemplos de configuração são:
  - `<3->#define HAL_USE_PAL TRUE`
  - `<3->#define HAL_USE_PWM FALSE`
  - `<3->#define HAL_USE_SERIAL TRUE`
  - `<3->#define HAL_USE_I2C FALSE`
- Note que, para usar um módulo da HAL, precisamos ativar o periférico de hardware correspondente antes

Arquivo chconf.h

- Em seguida, configuramos o arquivo chconf.h.

Arquivo chconf.h

- Em seguida, configuramos o arquivo chconf.h.
- Este arquivo define as funcionalidades do sistema operacional.

Arquivo chconf.h

- Em seguida, configuramos o arquivo chconf.h.
- Este arquivo define as funcionalidades do sistema operacional.
- Exemplos de configuração são:
  - `<3->#define CH_CFG_ST_FREQUENCY 1000`
  - `<3->#define CH_CFG_ST_TIMEDELTA 0`
  - `<3->#define CH_CFG_TIME_QUANTUM 0`
  - `<3->#define CH_CFG_USE_MUTEXES TRUE.`

Arquivo chconf.h

- Em seguida, configuramos o arquivo chconf.h.
- Este arquivo define as funcionalidades do sistema operacional.
- Exemplos de configuração são:
  - `<3->#define CH_CFG_ST_FREQUENCY 1000`
  - `<3->#define CH_CFG_ST_TIMEDELTA 0`
  - `<3->#define CH_CFG_TIME_QUANTUM 0`
  - `<3->#define CH_CFG_USE_MUTEXES TRUE.`
- No caso de chconf.h, a grande maioria das configurações-padrão são adequadas para os nossos programas

Arquivo Makefile

- Por último, alteramos o arquivo Makefile.

Arquivo Makefile

- Por último, alteramos o arquivo Makefile.
- Este arquivo define os arquivos que serão compilados e as características de compilação, e depende do MCU e da placa utilizada.

Arquivo Makefile

- Por último, alteramos o arquivo Makefile.
- Este arquivo define os arquivos que serão compilados e as características de compilação, e depende do MCU e da placa utilizada.
- Em geral só configuraremos as seguintes linhas:
  - `<3->PROJECT = ch`
  - `<3->CSRC = $(ALLCSRC) main.c`

Arquivo Makefile

- Por último, alteramos o arquivo Makefile.
- Este arquivo define os arquivos que serão compilados e as características de compilação, e depende do MCU e da placa utilizada.
- Em geral só configuraremos as seguintes linhas:
  - `<3->PROJECT = ch`
  - `<3->CSRC = $(ALLCSRC) main.c`
- A primeira linha define o nome do arquivo do programa binário final
  - Não é essencial, mas ajuda na organização

Arquivo Makefile

- Por último, alteramos o arquivo Makefile.
- Este arquivo define os arquivos que serão compilados e as características de compilação, e depende do MCU e da placa utilizada.
- Em geral só configuraremos as seguintes linhas:
  - `<3->PROJECT = ch`
  - `<3->CSRC = $(ALLCSRC) main.c`
- A primeira linha define o nome do arquivo do programa binário final
  - Não é essencial, mas ajuda na organização
- A segunda linha é essencial e nela devemos listar todos os arquivos de código C a serem compilados
  - Não apague qualquer trecho do tipo `$(ALLCSRC)`.

Arquivo Makefile

- Entretanto, é possível que seja necessário modificar o arquivo Makefile.

Arquivo Makefile

- Entretanto, é possível que seja necessário modificar o arquivo Makefile.
- Para a arquitetura AVR, Talvez seja necessário mudar a porta serial e / ou sua taxa de transmissão, a depender da placa usada.

Arquivo Makefile

- Entretanto, é possível que seja necessário modificar o arquivo Makefile.
- Para a arquitetura AVR, Talvez seja necessário mudar a porta serial e / ou sua taxa de transmissão, a depender da placa usada.
- Para o Stm32, temos que adicionar algumas linhas de instrução adicionais para podermos gravar o programa na placa com o Makefile.

Arquivo Makefile

- Entretanto, é possível que seja necessário modificar o arquivo Makefile.
- Para a arquitetura AVR, Talvez seja necessário mudar a porta serial e / ou sua taxa de transmissão, a depender da placa usada.
- Para o Stm32, temos que adicionar algumas linhas de instrução adicionais para podermos gravar o programa na placa com o Makefile.
- Arquivos Makefile com essas mudanças estão disponíveis no site junto com esta apresentação.

Introdução

- No ChibiOS, o módulo que lida com os pinos GPIO se chamam Port Abstraction Layer, ou PAL.

Introdução

- No ChibiOS, o módulo que lida com os pinos GPIO se chamam Port Abstraction Layer, ou PAL.
- Podemos acessar um único pino (chamado de pad), um grupo de pinos ou todos os pinos de uma porta.

Introdução

- No ChibiOS, o módulo que lida com os pinos GPIO se chamam Port Abstraction Layer, ou PAL.
- Podemos acessar um único pino (chamado de pad), um grupo de pinos ou todos os pinos de uma porta.
- Para o endereço de um pino, podemos usar a dupla (porta, pino) ou podemos usar o conceito de linha.

Uso

- Para uso do PAL, temos que configurar o modo de operação dos pinos antes de usá-los.

Uso

- Para uso do PAL, temos que configurar o modo de operação dos pinos antes de usá-los.
- Uma vez configurados, podemos fazer as seguintes operações:

Uso

- Para uso do PAL, temos que configurar o modo de operação dos pinos antes de usá-los.
- Uma vez configurados, podemos fazer as seguintes operações:
  - Escrever valores binários.
  - Setar pinos.
  - Resetar pinos.
  - Inverter os valores de pinos.

HAL - GPIO

Características

- A subcamada HAL API é projetada de forma o mais abrangente possível de forma a lidar com a maior quantidade de hardware possível.

HAL - GPIO

Características

- A subcamada HAL API é projetada de forma o mais abrangente possível de forma a lidar com a maior quantidade de hardware possível.
- A subcamada LLD realiza os passos necessários para a configuração e uso do hardware específico

Camada API

- A API define as seguintes funções:
  - `palReadPad(port, pad)`
  - `palWritePad(port, pad, bit)`
  - `palSetPad(port, pad)`
  - `palClearPad(port, pad)`
  - `palTogglePad(port, pad)`
  - `palSetPadMode(port, pad, mode)`.

Camada API

- O modo do pino pode ser um dos seguintes valores:
  - PAL_MODE_INPUT
  - PAL_MODE_INPUT_PULLUP
  - PAL_MODE_INPUT_PULLDOWN
  - PAL_MODE_INPUT_ANALOG
  - PAL_MODE_OUTPUT_PUSHPULL
  - PAL_MODE_OUTPUT_OPENDRAIN.

Camada API

- O modo do pino pode ser um dos seguintes valores:
  - PAL_MODE_INPUT
  - PAL_MODE_INPUT_PULLUP
  - PAL_MODE_INPUT_PULLDOWN
  - PAL_MODE_INPUT_ANALOG
  - PAL_MODE_OUTPUT_PUSHPULL
  - PAL_MODE_OUTPUT_OPENDRAIN.

Camada API

- O modo do pino pode ser um dos seguintes valores:
  - PAL_MODE_INPUT
  - PAL_MODE_INPUT_PULLUP
  - PAL_MODE_INPUT_PULLDOWN
  - PAL_MODE_INPUT_ANALOG
  - PAL_MODE_OUTPUT_PUSHPULL
  - PAL_MODE_OUTPUT_OPENDRAIN.
- Funções semelhantes existem para trabalhar com portas, grupos de bits e linhas
  - Documentação: http://chibiforge.org/doc/21.11/hal/group___p_a_l.html

Exemplo

- Um exemplo de um programa que usa o módulo PAL para fazer o led da placa piscar está mostrado ao lado.

```c
#include "hal.h"
#include "ch.h"

/* Constantes relacionadas ao led no blue pill */
#define LED_PORT IOPORT2
#define LED_PIN 5
#define LED_PERIODO 250

/* Função ineficiente, mas simples, para implementar um delay */
void delay_ms(int16_t ms) {
    int16_t i, j;
    for (i=0; i<ms; i++)
        for (j=0; j<1000; j++)
            asm("nop");
}

int main(void) {
    /* Sempre precisaremos inicializar tanto a HAL como o kernel */
    halInit();
    chSysInit();

    /* Configuramos o pino do led para o modo correto e ligamos o
    led */
    palSetPadMode(LED_PORT, LED_PIN, PAL_MODE_OUTPUT_PUSHPULL);
    palClearPad(LED_PORT, LED_PIN);

    while (1) {
        delay_ms(LED_PERIODO);
        palSetPad(LED_PORT, LED_PIN);
        delay_ms(LED_PERIODO);
        palClearPad(LED_PORT, LED_PIN);
    }
}
```

Exemplo

- Um exemplo de um programa que usa o módulo PAL para fazer o led da placa piscar está mostrado ao lado.
- É importante notar que o mesmo programa, com apenas pequenas alterações, funciona no ATMega328p e no Stm32.

```
#include "hal.h"
#include "ch.h"

/* Constantes relacionadas ao led no blue pill */
#define LED_PORT IOPORT2
#define LED_PIN 5
#define LED_PERIODO 250

/* Função ineficiente, mas simples, para implementar um delay */
void delay_ms(int16_t ms) {
    int16_t i, j;
    for (i=0; i<ms; i++)
        for (j=0; j<1000; j++)
            asm("nop");
}

int main(void) {
    /* Sempre precisaremos inicializar tanto a HAL como o kernel */
    halInit();
    chSysInit();

    /* Configuramos o pino do led para o modo correto e ligamos o
    led */
    palSetPadMode(LED_PORT, LED_PIN, PAL_MODE_OUTPUT_PUSHPULL);
    palClearPad(LED_PORT, LED_PIN);

    while (1) {
        delay_ms(LED_PERIODO);
        palSetPad(LED_PORT, LED_PIN);
        delay_ms(LED_PERIODO);
        palClearPad(LED_PORT, LED_PIN);
    }
}
```