

Arquitetura de Software para Sistemas Embarcados

Quarta - feira
25/MAR - 19:30



Jorge Guzman
Desenvolvedor de
Software Embarcado

GRATUITO

Apoio



SOBRE MIM

- Graduado em Engenharia da Computação
- Pós Graduação Automação industrial e Engenharia de Software
- 10 anos de experiência
- Participei e desenvolvi equipamentos para:

Laboratório

Subestações de energia elétrica

Automotivo

Indústria

IoT

Rodovias

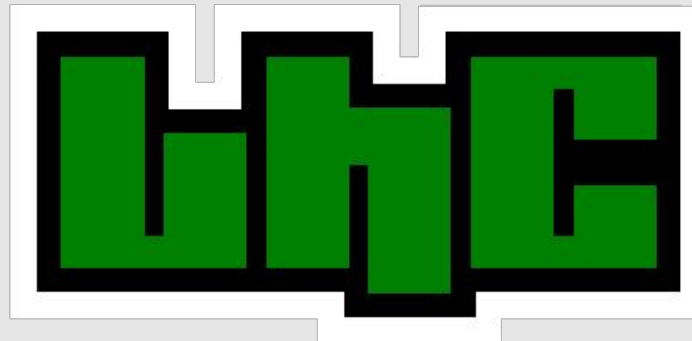


COMUNIDADES


- Articulista no Portal Embarcados
 - <https://www.embarcados.com.br/author/jorge-gzm/>



- Membro do Laboratório Hacker de Campinas
 - <http://lhc.net.br/>



AGENDA

- 
- Montagem de requisitos
 - Fluxo de desenvolvimento de um produto
 - Boas práticas
 - Arquitetura de software
 - Perguntas




Porque falar sobre
**ARQUITETURA DE
FIRMWARE ?**



Toda edificação começa
com a elaboração de seu
projeto arquitetônico



FLUXO DE DESENVOLVIMENTO

- 
- Etapas
 - Planejamento
 - Execução
 - Validação
 - Industrialização

Link fluxograma: <https://bit.ly/3adfnxS>



MONTAGEM DE REQUISITOS

- **Escopo de projeto** (Equipe gestora)
 - Informações sobre o projeto:
 - Descrição
 - Limites
 - Objetivos
 - Entregas
 - Responsáveis
 - Custos
 - Prazos
 - Restrições
 - Premissas



MONTAGEM DE REQUISITOS

- **Requisitos do software** (Equipe técnica)
 - **Casos de Uso**
 - Requisitos funcionais
 - Representa o que o software faz, em termos de tarefas e serviços
 - Não funcionais
 - **Qualidade:** usabilidade, confiabilidade, portabilidade, eficiência
 - **Implementação:** hardware, plataforma que serão usadas, linguagens de programação
 - **Ambiente:** Interoperabilidade, segurança, privacidade, sigilo
 - **Organização:** aderência a padrões

Caso de Uso: UC-09 Salvar telemetrias

Descrição: Toda falha ou retorno de falha de sensores ou do hardware detectadas pelo sistema devem ser registrada na forma de telemetria com data e hora

(a) Ator(es): RTC, sensores e equipamento

(b) Pré-condições:

1. Ocorrência de uma falha no sistema ou nos sensores de temperatura, humidade,

(c) Pós-condições:

1. O registro da falha passa a constar no histórico (log) na EEPROM.
2. O número total de telemetrias deve ser salva na estrutura de parâmetros de status na EEPROM

(d) Requisitos funcionais:

- 1.**RF01.** Criar uma estrutura de telemetria
- 1.**RF02.** O sistema deve conseguir escrever a telemetria na EEPROM
- 2.**RF03.** Falhas de watchdog e variação brusca da tensão de alimentação devem ser registradas
- 3.**RF04.** A ocorrência de queda de energia do equipamento deve ser registrada
- 4.**RF05.** Registrar falhas do sensor de temperatura e humidade
- 5.**RF06.** Salvar telemetria no EEPROM e manter até 5000 registros

(e) Requisitos não-funcionais:

- 1.**RNF01.** Durante a queda de energia ou falha na escrita da EEPROM, o sistema não pode perder os registros e número de telemetrias
- 2.**RNF02.** Evitar saturação da memória EEPROM por múltiplas escritas em um mesmo endereço
- 3.**RNF03.** No caso de falha na EEPROM, reiniciar o driver do periférico, ligar o led de erro, ...

METODOLOGIAS ÁGEIS

Tenta reduzir custos, prazos e aumentar a qualidade (**BOTAR ORDEM NA CASA**)

- **Scrum**

- Product Backlog
- Sprints
- Scrum master

- **Kanban**

- Status do projeto
- Monitoramento das atividades individuais
- Medição de retrabalho por causa de Bugs



KANBAN - GITLAB

STM32 > FMW > STM32-FMW-00020 > Painéis

Development ▾ Pesquisar ou filtrar resultados... Show labels ☒ Editar painel Add list Add issues

► To Do 4 +

- Implementar Driver para leitura de 5 teclas #6
- Implementar telas de menu no Display TFT #5
- Implementar Driver para o Display TFT st7735 #2
- Implementar Driver para o modulo ES8266 #1

► Doing 1 +

- Implementar lib para blink de leds #7

► Validar 1 +

- Salvar configurações no formato de arquivo JSON #4

► Closed 1

- Recuperar configurações de um arquivo JSON #3

Open 1 Closed 0 All 1

Filtrar por nome de marco

Data de vencimen... ▾

Novo marco

v1.0.0

STM32 / FMW / STM32-FMW-00020

7 Issue · 0 Merge Request

14% complete



Close Milestone

ATÉ AQUI CONCLUÍMOS QUE...



- É preciso entender bem o problema e os riscos antes de iniciar um projeto
- É preciso cobrar a documentação com os de requisitos de entrada dos gestores antes de sair codificando
- Os gestores precisa ajudar a equipe a encontrar soluções, mesmo que seja uma ajuda externa
- Incluir no cronograma do projeto o período de tempo que será gasto em **testes e validação**
- Aplicar metodologias ágeis em uma equipe **gera desconforto** no começo
- Avisar a equipe quando estiver tendo dificuldades para finalizar uma atividade



BOAS PRÁTICAS



FIRMWARES DE PRODUÇÃO



Bootloaders

Tipos de sistema de update:

- On-Board bootloader: bootloader interno do microcontrolador
- Dual bank: Memória flash fragmentada em 2 partes iguais
- Seu próprio bootloader: escrita e customização do seu sistema de update
- Atualização via mem. RAM: uso da memória RAM para alocar todo o código de update

FIRMWARES DE PRODUÇÃO - CONT.

- **Seu próprio Bootloader:**

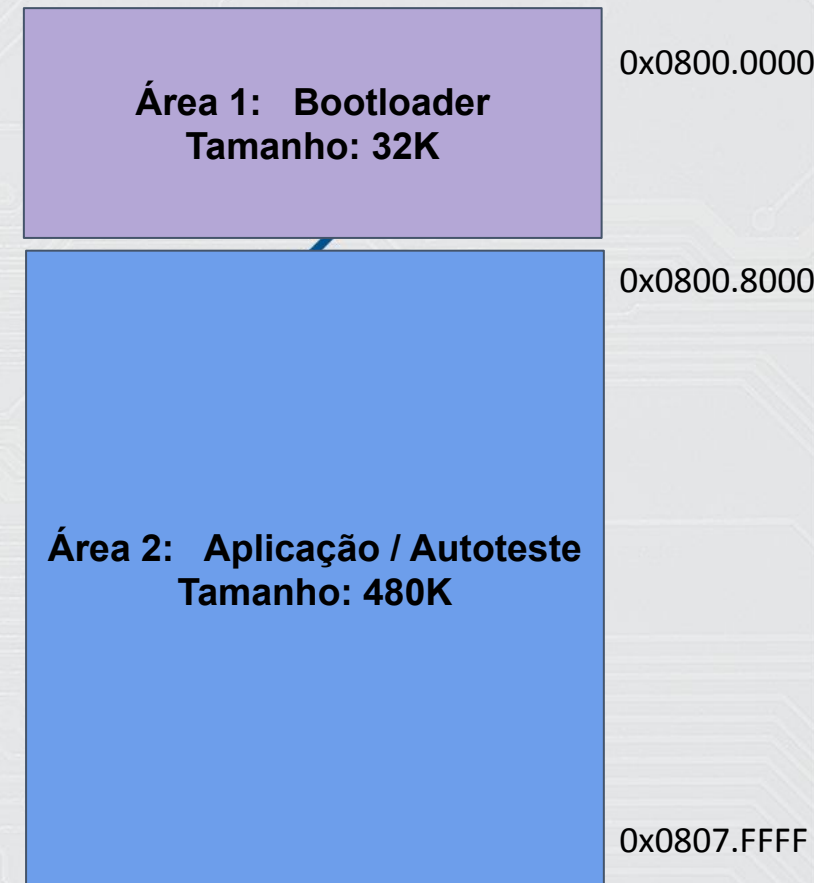
- Protocolo de comunicação (Ex: XModem)
- Criptografia

- **Informações de fábrica**

Versão de hardware (Ex: V1R2E1 --> **V**ersão, **R**evisão, **E**missão)

Versão do firmware de bootloader

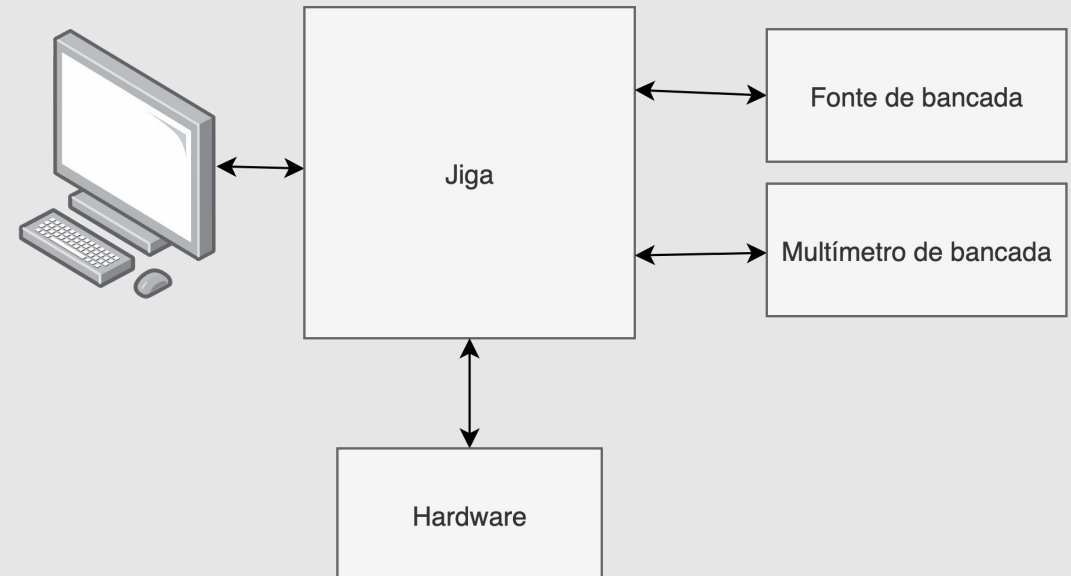
Checksum da aplicação gravada



FIRMWARES DE PRODUÇÃO - CONT.

- **Firmware de Autoteste**

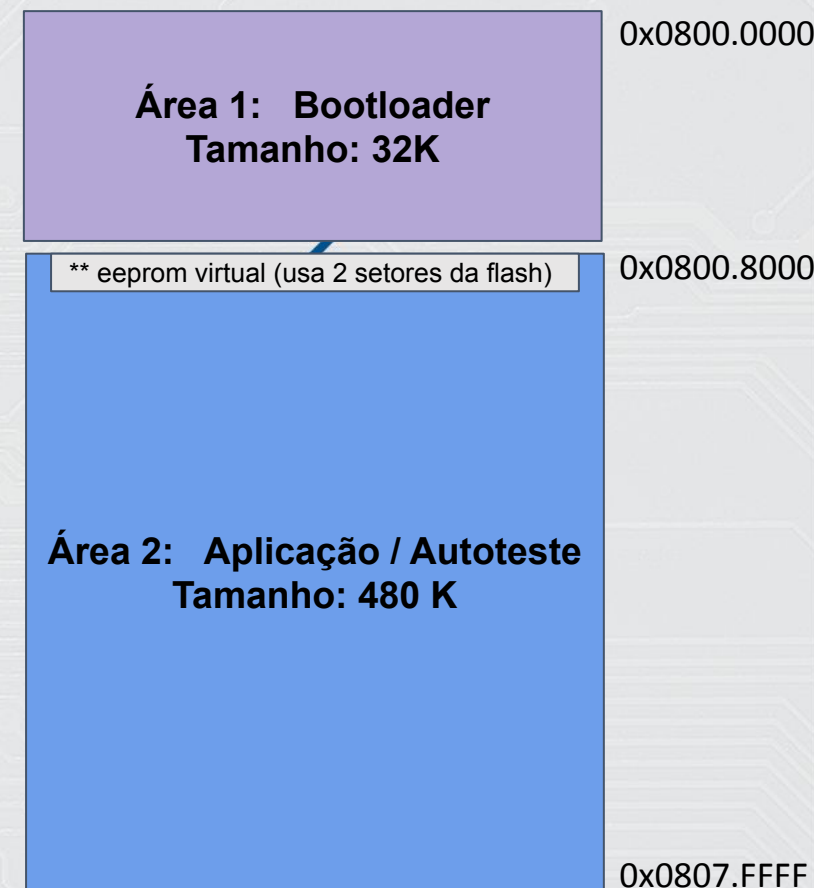
- Firmware para testar todos os periféricos do hardware
- Usado na etapa de **bring up** e teste de **qualidade** na fábrica
- Em alguns casos é usado uma jiga de teste / cama de agulhas para testes de peças, tensão, corrente, sinais on/off, etc



FIRMWARES DE PRODUÇÃO - CONT.

Firmware da Aplicação (RTOS ou Bare Metal)

- Tipos:
 - Bare Metal ou single loop
 - RTOS
- Informações de fábrica
 - Número de série, usado na rastreabilidade do produto
 - Versão de firmware da aplicação



** Uso opcional



CODIFICAÇÃO - BOAS PRÁTICAS



- **Code Style / Code Standard**

- Aparência visual do código e legibilidade
- Impor boas práticas para desenvolver um software seguro
 - Ex: Padrão MISRA-C

- **Clean Code**

- Documentação de código, principalmente funções matemáticas
- Diminuir complexidade de código, quebrando uma rotina em várias funções

CODIFICAÇÃO - BOAS PRÁTICAS - CONTINUAÇÃO

- **Test-Driven Development**

- Testes unitários
- Segurança no desenvolvimento e durante a correção de bugs
- Garante a qualidade das libs implementadas

- **Code-Review / Peer Code Review**

- Aumento na qualidade do código com mais pessoas revisando
- Aumento de senso de equipe
- Compartilhar conhecimento



CODIFICAÇÃO - BOAS PRÁTICAS - CONTINUAÇÃO



- **Manipulando Variáveis**

- A Linguagem C não suporta criação de classes, onde são encapsulados variáveis e funções em um único objeto.
- Evitar usar **variáveis globais**, atrapalha na manutenção, portabilidade e controle da Lib
 - Exemplo: Como aproveitar somente o conteúdo do arquivo task_protocol.c ?

```
/* Arquivo serial.c:*/  
bool pacoteRXPronto;  
  
/* Arquivo task_led.c*/  
#include "serial.h"  
extern bool pacoteRXPronto;  
  
/* Arquivo task_protocol.c: */  
#include "serial.h"  
extern bool pacoteRXPronto;
```

CODIFICAÇÃO - BOAS PRÁTICAS - CONTINUAÇÃO

- Usar **estruturas de dados** para reunir variáveis que manipularam um mesmo objeto

```
/* Arquivo setup_database.h */
typedef struct {
    InfoDevice_t info;          /* Parametros de fabrica          */
    Rede_t rede;                /* Parametros de rede wifi       */
    ConfigMQTT_t mqtt;          /* Parametros de conexao MQTT    */
    Languages_e language;       /* Idioma                         */
    PIDConfigParams_t pid;       /* Parametros de controle PID    */
    uint32_t reservado[10] /* Espaco para gravar futuros parametros (1280 bytes) */
    uint32_t checksum;          /* checksum dos dados acima      */
}DataBase_t;
```

- Usar funções **get** e **set** quando possível

```
/* Arquivo setup_database.c */
DataBase_t xSystemParams;
...
void network_setHost(uint8_t *str) {
    memcpy(xSystemParams.rede.host, str, sizeof(xSystemParams.rede.host));
}

uint8_t* network_getHost(void) {
    return xSystemParams.rede.host;
}
...
```


CODIFICAÇÃO - BOAS PRÁTICAS - CONTINUAÇÃO

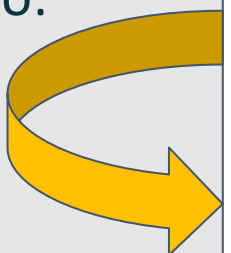
- Não usar números mágicos, usar **defines** ou **enum**

- Exemplo:

```
#define TIMEOUT_5_MS      5
#define TIMEOUT_5_SEG    5000
typedef enum { eESPERANDO = 0, eCOLETANDO, eFINALIZADO} EstadosColeta_e;
```

- Usar **static** em variáveis locais que, ao sair da determinada função, não perdem seu valor

- Exemplo:



```
EstadosColeta_e status = eESPERANDO
void RX_ISR_RX(uint8_t dado)
{
    static EstadosColeta_e status = eESPERANDO;
    ...
    status = eCOLETANDO;
}
```



CODIFICAÇÃO - BOAS PRÁTICAS - CONTINUAÇÃO

- **Command line interface**

- Configurar múltiplos parâmetros
- Monitorar estado de variáveis de controle via UART, USB, Ethernet, J-Link, etc
- Executar rotinas ou funções específicas sem precisar interagir com uma interface ou sensores.
- Injetar valores fake para testar rotinas e funções em tempo de execução

```
Receber ASCII Receber HEX
Init Program...
$ help

Supported commands:
-> rede -w WIFI_NAME -pwd PASSWORD
-> connect -h HOST -p PORT

$ rede -w @Casa -p 123456789

(X) Sintax error
$ rede -w @Casa -pwd 123456789

$ connect -h iot.eclipse.org -p 1883

$ show -cfg

===== CONFIGS =====
wifi name: @Casa
password: 123456789
host: iot.eclipse.org
port: 1883

ASCII enviado HEX enviado
help
rede -w @Casa -p 123456789
rede -w @Casa -pwd 123456789
connect -h iot.eclipse.org -p 1883
show -cfg

Saída
Enviar ASCII CR+LF end
```



CODIFICAÇÃO - BOAS PRÁTICAS - CONTINUAÇÃO



- **Macros para rotinas de debug**

- Porque usar macros ao invés de funções ?
 - E possível habilitar e desabilitar elas durante o build
 - E possível selecionar a função que a macro irá usar
 - Ajuda na portabilidade por não precisar incluir libs específicas

```
/* Arquivo sertup_debug.h */
#include "setup_hw.h"

#if configENABLE_DEBUG > 0

#define DBG_BKPT()          asm("BKPT #0")
#define DBG_PRINTFLN(fmt, ...) eDebug_Printf(fmt"\r\n", ##__VA_ARGS__)
#define DBG_PRINTF(fmt, ...) eDebug_Printf(fmt, ##__VA_ARGS__)
#define DBG_SEND(buff, size) eDebug_SendBuffer(buff, size)
#define DBG_ASSERT_PARAM(x) SYSLOG_ASSERT_PARAM(x)

#define DBG(fmt, ...)        vSyslogMsg(eSYSLOG_DEBUG, fmt, ##__VA_ARGS__)
#define INFO(fmt, ...)       vSyslogMsg(eSYSLOG_INFO,  fmt, ##__VA_ARGS__)
#define ERR(fmt, ...)        vSyslogMsg(eSYSLOG_ERROR, fmt, ##__VA_ARGS__)
#define WARN(fmt, ...)       vSyslogMsg(eSYSLOG_WARN,  fmt, ##__VA_ARGS__)
#define MSG(type, fmt, ...)  vSyslogMsg(type,         fmt, ##__VA_ARGS__)
```

```
#else

#define DBG_BKPT()          __NOP()
#define DBG_PRINTFLN(fmt, ...) __NOP()
#define DBG_PRINTF(fmt, ...)  __NOP()
#define DBG_SEND(buff, size) __NOP()
#define DBG_ASSERT_PARAM(x)  __NOP()

#define DBG(fmt, ...)        __NOP()
#define INFO(fmt, ...)       __NOP()
#define ERR(fmt, ...)        __NOP()
#define WARN(fmt, ...)       __NOP()
#define MSG(type, fmt, ...)  __NOP()

#endif /* configENABLE_DEBUG */
```

```
color01 base08 AB4642 Red
color02 base0B A1B56C Green
color03 base0A F7CA88 Yellow
color04 base0D 7CAFC2 Blue
```


CODIFICAÇÃO - BOAS PRÁTICAS - CONTINUAÇÃO




- **Processamento de dados críticos**

- Usar fila circular para armazenar dados dentro de uma interrupção
- Usar estrutura de ping pong para não perder a captura de dados
 - Enquanto um vetor é processado o outro fica aquisitando os dados
- Usar o recurso de DMA dos periféricos
 - Fazendo isso a CPU não é sobrecarregada
- Não usar rotinas de printf via uart para monitorar dados
 - Uma alternativa é usar a o printf da lib RTT (Real Time Transfer) da Segger com o J-Link



FERRAMENTAS DE TRACE

- 
- Possibilita monitorar a mudança de contexto de tarefas em tempo real
 - Ferramentas
 - Segger SystemView (gratuito, funciona somente com gravador J-Link)
 - Percepio (ferramenta paga)





PADRÕES DE PROJETO



DESIGN PATTERNS

Uma mesma solução para problemas diferentes

- **Padrões de criação:** se preocupam com o processo de criação de um objeto
- **Padrões estruturais:** lidam com a composição de classes ou de objetos
- **Padrões comportamentais:** caracterizam as maneiras que classes ou objetos interagem e distribuem responsabilidade

Padrões de criação

- *Abstract Factory*
- *Object pool*
- *Builder*
- *Factory Method*
- *Prototype*
- *Singleton*

Padrões estruturais

- *Private class data*
- *Adapter*
- *Bridge*
- *Composite*
- *Decorator*
- *Façaade* (ou *Facade*)
- *Business Delegate*
- *Flyweight*
- *Proxy*

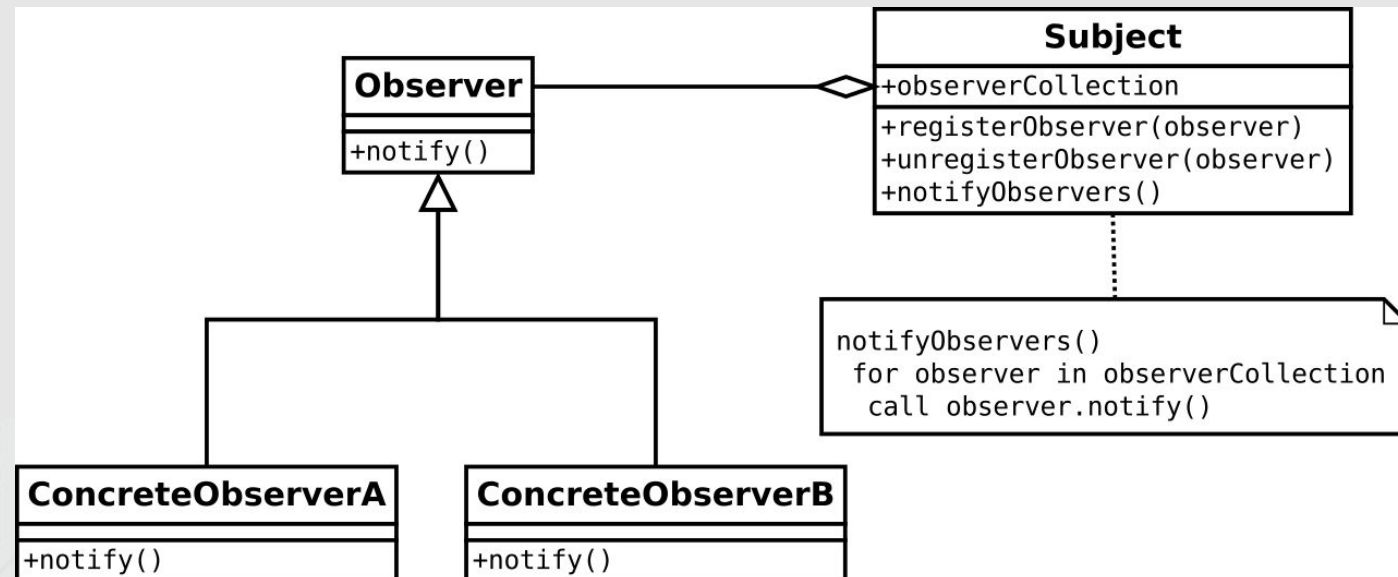
Padrões comportamentais

- *Chain of Responsibility*
- *Command*
- *Interpreter*
- *Iterator*
- *Mediator*
- *Memento*
- *Observer*
- *State*
- *Strategy*
- *Template Method*
- *Visitor*



DESIGN PATTERNS - OBSERVER

- Define uma dependência um-para-muitos entre objetos. Quando um objeto muda o estado, todos seus dependentes interessados são notificados e atualizados automaticamente.
- O padrão Observer é também chamado de:
 - **Publisher-Subscriber**
 - Event Generator
 - Dependents.



DESIGN PATTERNS - OBSERVER CONTINUAÇÃO

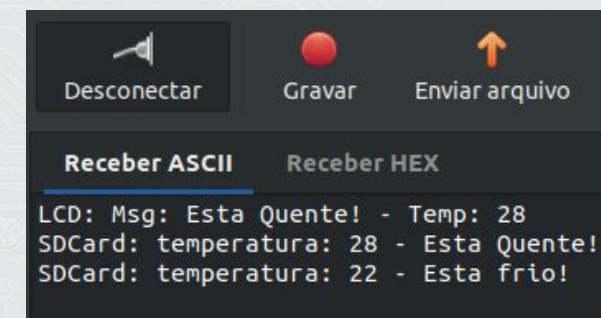


MOUSER
ELECTRONICS

```
71 typedef struct
72 {
73     uint16_t (*notify)(uint8_t *message, int16_t val);
74 }Subject_t;
75
76 Subject_t TempNotify[4] = { 0 };
77
78 void temp_subjectAttach(uint16_t index, uint16_t (*notify)(uint8_t *, int16_t))
79 {
80     if(index < sizeof(TempNotify)/sizeof(Subject_t))
81     {
82         TempNotify[index].notify = notify;
83     }
84 }
85
86 void temp_subjectDetach(uint16_t index)
87 {
88     if(index < sizeof(TempNotify)/sizeof(Subject_t))
89     {
90         TempNotify[index].notify = NULL;
91     }
92 }
93
94 void temp_notify(uint8_t *data, uint16_t temp)
95 {
96     for(uint16_t index =0; index < sizeof(TempNotify)/sizeof(Subject_t); index++)
97     {
98         if(TempNotify[index].notify != NULL)
99         {
100             TempNotify[index].notify(data, temp);
101         }
102     }
103 }
104
105 void temp_calc(int16_t temperature)
106 {
107     if(temperature < 23)
108     {
109         temp_notify((uint8_t*)"Esta frio!", temperature);
110     }
111     else if(temperature >=23 && temperature <=27)
112     {
113         temp_notify((uint8_t*)"Esta agradável!", temperature);
114     }
115     else
116     {
117         temp_notify((uint8_t*)"Esta Quente!", temperature);
118     }
119 }
```

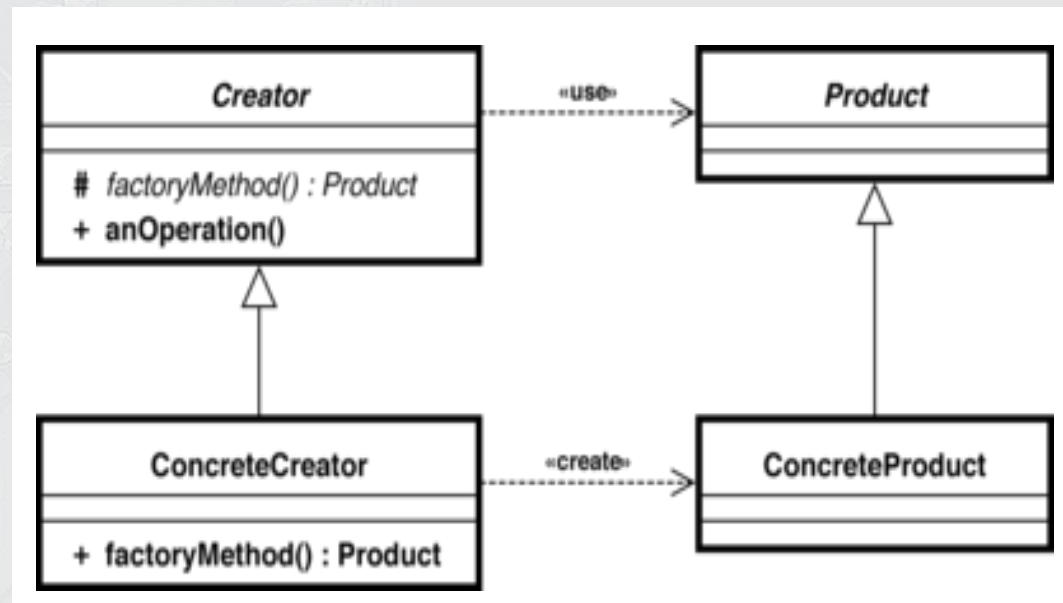
```
121 uint16_t lcd_ObserverNotify(uint8_t *message, int16_t val)
122 {
123     uint8_t lcdMessage[50];
124     uint16_t size;
125     size = snprintf((char*)lcdMessage, sizeof(lcdMessage), "LCD: Msg: %s - Temp: %d\r\n", message, val);
126     return HAL_UART_Transmit(&huart2, (uint8_t*)lcdMessage, size, 1000);
127 }
128
129 uint16_t sdcard_ObserverNotify(uint8_t *message, int16_t val)
130 {
131     uint8_t sdCardMessage[50];
132     uint16_t size;
133     size = snprintf((char*)sdCardMessage, sizeof(sdCardMessage), "SDCard: temperatura: %d - %s\r\n", val, message);
134     return HAL_UART_Transmit(&huart2, (uint8_t*)sdCardMessage, size, 1000);
135 }
```

```
205 void setup_init(void)
206 {
207     temp_subjectAttach(0, lcd_ObserverNotify);
208     temp_subjectAttach(1, sdcard_ObserverNotify);
209
210     temp_calc(28);
211
212     temp_subjectDetach(0);
213
214     temp_calc(22);
215 }
```



PADRÃO FACTORY

- Fornece uma classes base somente com interfaces, cabendo as classes filhas implementarem seu conteúdo.



PADRÃO FACTORY - CONTINUAÇÃO

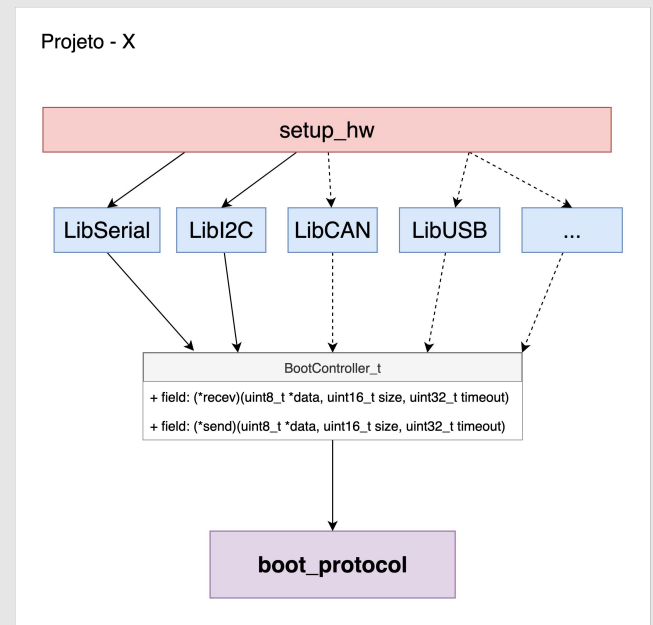
- Interface abstrata para para criar um objeto do tipo BootController_t

```
/* Arquivo boot_protocol.h */
typedef struct {
    HAL_StatusTypeDef (*recev)(uint8_t *data, uint16_t size, uint32_t timeout);
    HAL_StatusTypeDef (*send)(uint8_t *data, uint16_t size, uint32_t timeout);
}BootController_t;
```

- Criando objetos do tipo i2c e UART para usar a lib bootloader

```
/* Arquivo setup_hw.c */
#define BOOT_TIME_OUT_MS 200
BootController_t bootDrvI2C = { i2c_read, i2c_write} ;
BootController_t bootDrvUART = { serial_recev, serial_send};

void main(void) {
    ...
    boot_attachFunc(&bootDrvI2C);
    boot_exec(BOOT_TIME_OUT_MS);
    ...
    boot_attachFunc(&bootDrvUART);
    boot_exec(BOOT_TIME_OUT_MS);
    ...
}
```



PADRÃO FACTORY - EXEMPLO

```
/* Arquivo boot_protocol.c */

BootController_t *bootDriver = { 0 };

void boot_attachFunc(BootController_t *drvController)
{
    bootDriver = drvController;
}

HAL_StatusTypeDef boot_exec(uint32_t timeout)
{
    HAL_StatusTypeDef err = HAL_ERROR;

    uint8_t rxBuffer[512];
    uint8_t txBuffer[512];

    err = bootDriver->recev(rxBuffer, sizeof(rxBuffer), timeout);
    ...
    err = bootDriver->send(txBuffer, sizeof(txBuffer), timeout);
    ...
    return err;
}
```

```
/* Arquivo LibI2C.c */
static I2C_HandleTypeDef *I2Cdrv = NULL;

HAL_StatusTypeDef i2c_write(uint8_t *data, uint16_t size, uint32_t timeout)
{
    HAL_StatusTypeDef resp = HAL_ERROR;

    if( I2Cdrv != NULL )
    {
        resp = HAL_I2C_Mem_Write( I2Cdrv, MCU_ADDRESS, 0x00,
                                   I2C_MEMADD_SIZE_8BIT, data, size, timeout );
    }

    return resp;
}

HAL_StatusTypeDef i2c_read(uint8_t *data, uint16_t size, uint32_t timeout)
{
    HAL_StatusTypeDef resp = HAL_ERROR;

    if( I2Cdrv != NULL )
    {
        resp = HAL_I2C_Mem_Read( I2Cdrv, MCU_ADDRESS | 0x01, 0x00,
                                   I2C_MEMADD_SIZE_8BIT, data, size, timeout );
    }

    return resp;
}
```

```
/* Arquivo LibSerial.c */
static UART_HandleTypeDef *drvUart = NULL;

HAL_StatusTypeDef serial_send(uint8_t *data, uint16_t size, uint32_t timeout)
{
    HAL_StatusTypeDef resp = HAL_ERROR;

    if( drvUart != NULL )
    {
        resp = HAL_UART_Transmit(drvUart, data, size, timeout);
    }

    return resp;
}

HAL_StatusTypeDef serial_recev(uint8_t *data, uint16_t u16Size, uint32_t timeout)
{
    HAL_StatusTypeDef resp = HAL_ERROR;

    if( drvUart != NULL )
    {
        resp = HAL_UART_Receive(drvUart, data, size, timeout);
    }

    return resp;
}
```




ARQUITETURA



PROJETOS - MULTIPLATAFORMA

- **Vantagens:**

- Popularização da cultura em eletrônica e programação
- Popularização de hardwares de baixo custo (Placas, shields, robôs, impressora 3d, etc)
- Ferramentas gratuitas e sem limite de código
- Portabilidade de código para diferentes micros com pouco esforço
- Prototipagem rápida, variedade drivers para CI e periféricos
- Códigos open source
- Padronização de funções usando Wiring language (Ex: digitalWrite) (**Menos o ARM mbed**)

- **Desvantagens:**

- Algumas pessoas não conseguem ter a visão de arquitetura e orientação a objetos
- Uso de printf para debug de código(**isso está mudando**)
- A qualidade de algumas libs nem sempre é garantida
- Ao usar RTOS é preciso alterar algumas libs
- Configurações avançadas algumas vezes é escondida

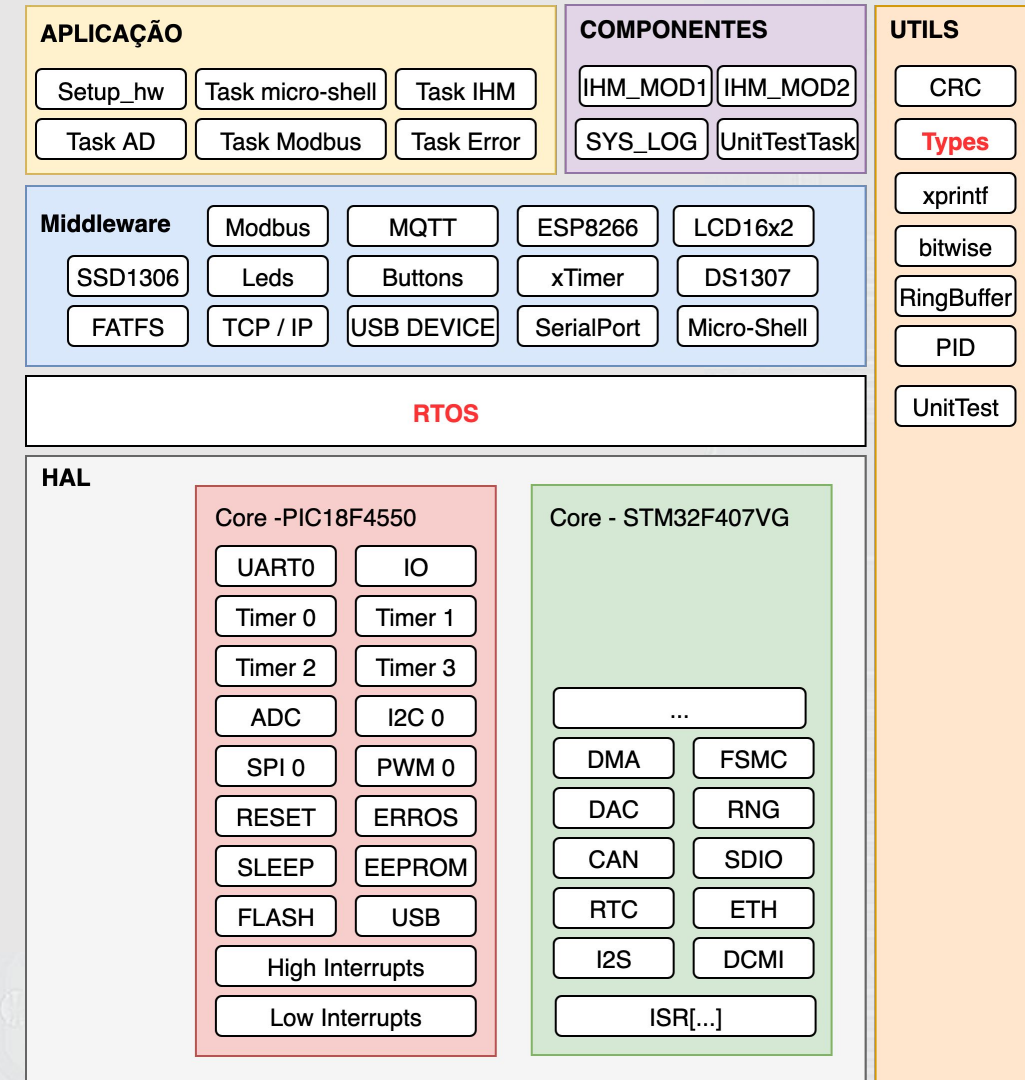


Pinguino



ARQUITETURA - CAMADAS

- **UTILS**
 - Libs genéricas
 - Sem acesso a Hardware
 - **Types**: estruturas globais e compartilhadas por todas as camadas
- **HAL**
 - Padronização para escrever ou ler um periférico (Ex: digitalWrite(), pinMode(), etc)
 - Drivers específicos do microcontrolador
- **RTOS**
 - Uso opcional, porém se usar todas as libs acima devem implantá-la
- **MIDDLEWARE**
 - Drivers de Periféricos e CIs
 - Libs com protocolos de comunicação
- **COMPONENTES**
 - componentes de hardware ou serviços usados em vários projetos
- **APLICAÇÃO**
 - Rotinas de configuração de hardware e libs
 - Tarefas da aplicação



DIFICULDADES AO MONTAR SUA PRÓPRIA ARQUITETURA

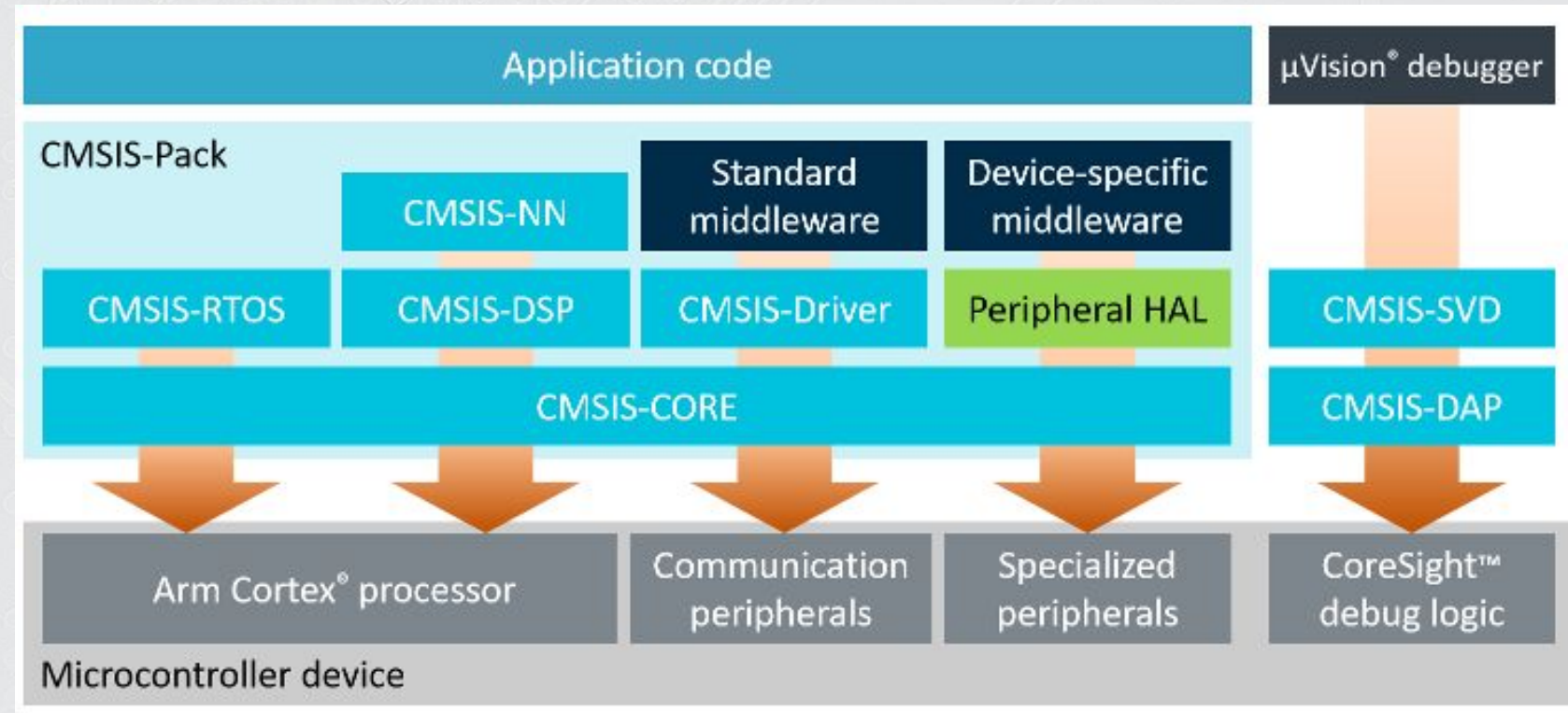


- Portabilidade de código entre fabricantes diferentes
 - Microcontroladores com diferentes arquiteturas 8/16/32 bits
 - Tamanho de Flash e RAM diferentes
 - Padronizar os protótipo de funções para configurar os periféricos
 - Compilador (Ex: comandos pragma, attribute)
 - Excesso de macros de configuração
- Perda de desempenho e performance em determinadas rotinas
- Quebra do funcionamento se não houver controle de versão para cada projeto
- Uso de diferentes RTOS
- Adaptação de Libs ao usar RTOS
- Elevado grau de manutenção e consumo de tempo
- Ao longo do tempo a manutenção de micros com arquitetura 8 e 16 bits **podem** ser deixada de lado

CAMADA DE ABSTRAÇÃO (API)

- **ARM CMSIS (Cortex Microcontroller Software Interface Standard)**

- CMSIS-CORE
- CMSIS-DAP
- CMSIS-RTOS
- CMSIS-DSP
- CMSIS-NN
- CMSIS-Driver
- CMSIS-SVD



Documentação: <http://www.keil.com/pack/doc/CMSIS>



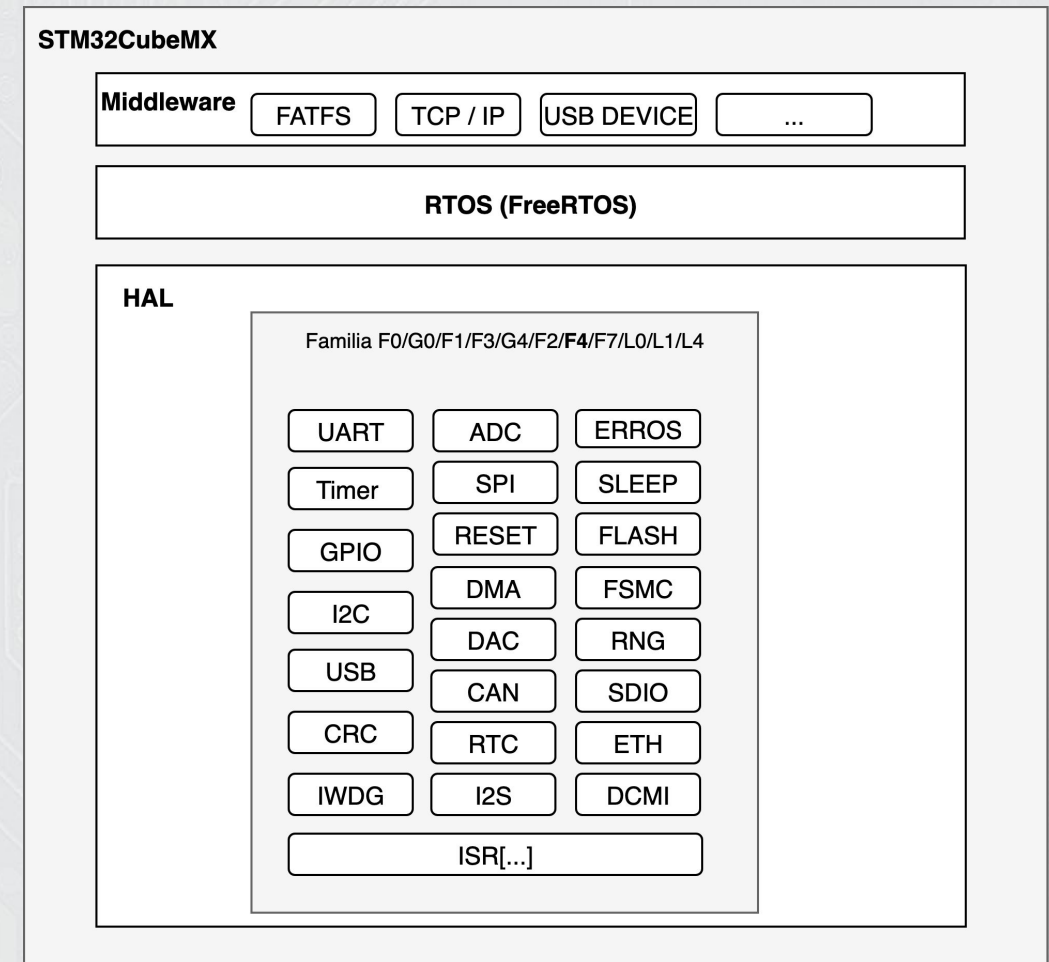
FERRAMENTA DE GERAÇÃO DE CÓDIGO

- **Vantagens:**

- Geração de código de inicialização
- Ferramenta visual para configurar periféricos
- Padronização do protótipo de funções
- Redução de tempo de desenvolvimento e prototipagem
- Suporte do fabricante
- Suporte da comunidade
- Periféricos instanciados na forma de objetos

- **Desvantagens:**

- Atualização da IDE pode gerar quebra no build
- Devido ao nível de abstração algumas libs podem gerar perda de performance
- Aumento no consumo de Flash e RAM
- **As aplicações ficam presas à HAL do fabricante**



FRAMEWORK



- **Vantagens:**

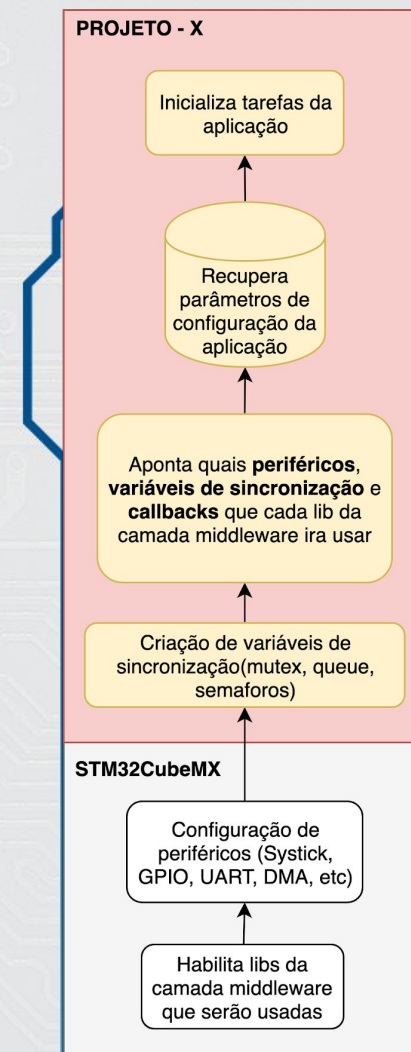
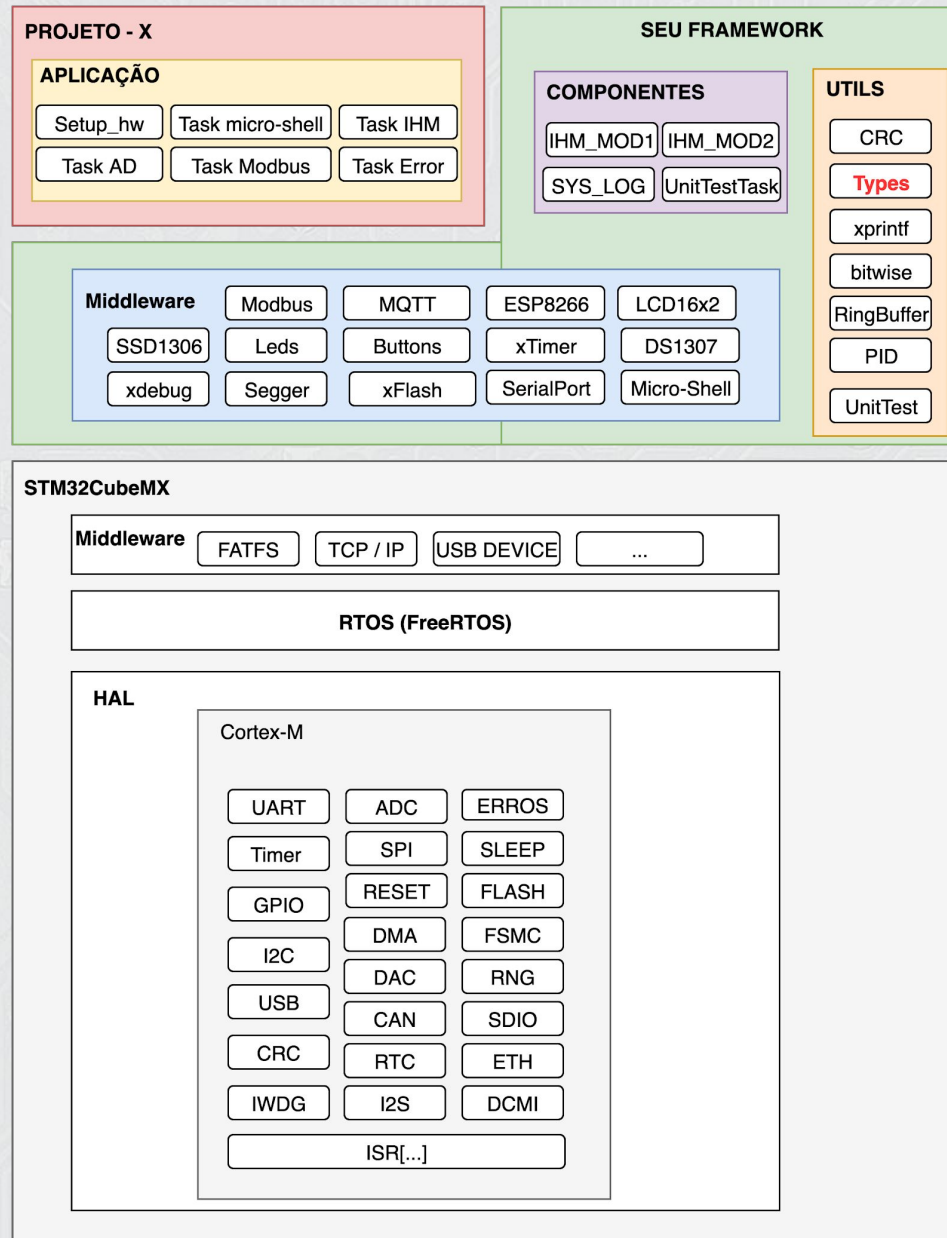
- Garante a portabilidade de código para outros projetos
 - Protótipo de funções padronizado
 - Uso de git submodule e subtree para controle de versionamento do framework

- **Desvantagens:**

- Alguns periféricos tem configurações de uso diferentes
 - Ex: M0 usa página para dividir flash enquanto M4 usa setor
 - **Usar compilação condicional**, para habilitar o arquivo xFlash_M0.c ou xFlash_M4.c, os dois arquivos implementam os protótipo de funções do arquivo xFlash.h

FRAMEWORK - CONTINUAÇÃO

- Projeto X
 - Tarefas específicas de sua aplicação
- Framework
 - Componentes
 - Middleware
 - Utils



CALLBACKS



Usada para notificar o acontecimento de um evento

- Método 1:
 - Usa ponteiro de função para chamar uma determinada função
 - O ponteiro de função aponta a função que será chamada, e ambos devem possuir o mesmo protótipo
 - É possível alterar a função que será chamada em tempo de execução
- Método 2:
 - Não usa ponteiro de função
 - Usa função **`__weak`**
 - Toda implementação é jogada a uma única função que pode ser implementada ou não

CALLBACKS - MÉTODO 1

```
/* Exemplo callback via ponteiro de função */

/* Arquivo calculadora.h */

float soma(int16_t a, int16_t b);
float dividir(int16_t a, int16_t b); float calc_exec(int16_t a, int16_t b,

/* Arquivo calculadora.c */

float calc_exec(int16_t a, int16_t b, float (*func)(int16_t, int16_t));
{
    /* Chama uma funcao qualquer */
    int16_t r = func(a,b);
    return r;
}

float soma(int16_t a, int16_t b)
{
    return (float)(a)/(float)(b);
}

float divide(int16_t a, int16_t b)
{
    return (float)(a)/(float)(b);
}
```

```
/* Arquivo app_calculadora.c */
#include "calculadora.h"

void chamada()
{
    int16_t a = 1;
    int16_t b = 2;
    float result;

    result = calc_exec(a,b,&soma);
    ENVIAR("Soma %d + %d = %f", a, b, result);

    result = calc_exec(a,b,&dividir);
    ENVIAR("Divisao %d / %d = %f", a, b, result);
}
```

CALLBACKS -MÉTODO 2

```
/* Exemplo callback via funcao WEAK */
/* Arquivo calc_temp.h */

float temp_alarmCallback(float temp);
float calc_temp(uint16_t adc);

/* Arquivo calc_temp.c */

float calc_temp(uint16_t adc);
{
    /* Chama uma funcao qualquer */
    float tempGraus = converterADCToGraus(adc);

    if(tempGraus < 10.0 || tempGraus > 30.0F)
    {
        /* Notifica temperatura fora dos Limites */
        temp_alarmCallback(tempGraus);
    }
    return tempGraus;
}
```

```
__weak float temp_alarmCallback(float temp)
{
    UNUSED(temp);
    return 0;
}
```

```
/* Nao implementado a funcao de callback */

/* Arquivo app_temp.c */

#include "calc_temp.h"

void chamada()
{
    float temp;
    uint16_t testADC = 1025;
    temp = calc_temp(testADC)
    ENVIAR("temperatura calculada: %f graus", temp)
}
```

```
/* Implementado a funcao de callback */
/* Arquivo app_temp.c */

#include calc_temp.h
#include "lcd.h"
#include "rele.h"
#include "log.h"

float temp_alarmCallback(float temp)
{
    lcd_print("Alarme de temperatura: %d graus", temp)
    rele_energia_off();
    criar_log(ERR_TEMP);
}
```

```
void chamada()
{
    float temp;
    uint16_t testADC = 1025;
    temp = calc_temp(1025)
    ENVIAR("temperatura calculada: %f graus", temp)
}
```

COMPONENTIZAÇÃO

- Realiza uma função específica
- Composta por uma ou mais libs da camada de middleware
- Precisa ser eficiente, confiável, escalável e reutilizável
- Ter interfaces claras de uso, configuração e execução
- Usar mutex ou semáforos para controle de recursos compartilhados
 - Exemplo: No mesmo barramento I2C (periférico I2C_0) existem outros sensores usados por outras bibliotecas
- Especificar quais periféricos do hardware que serão usados (ex: I2C, GPIO, UART, etc)

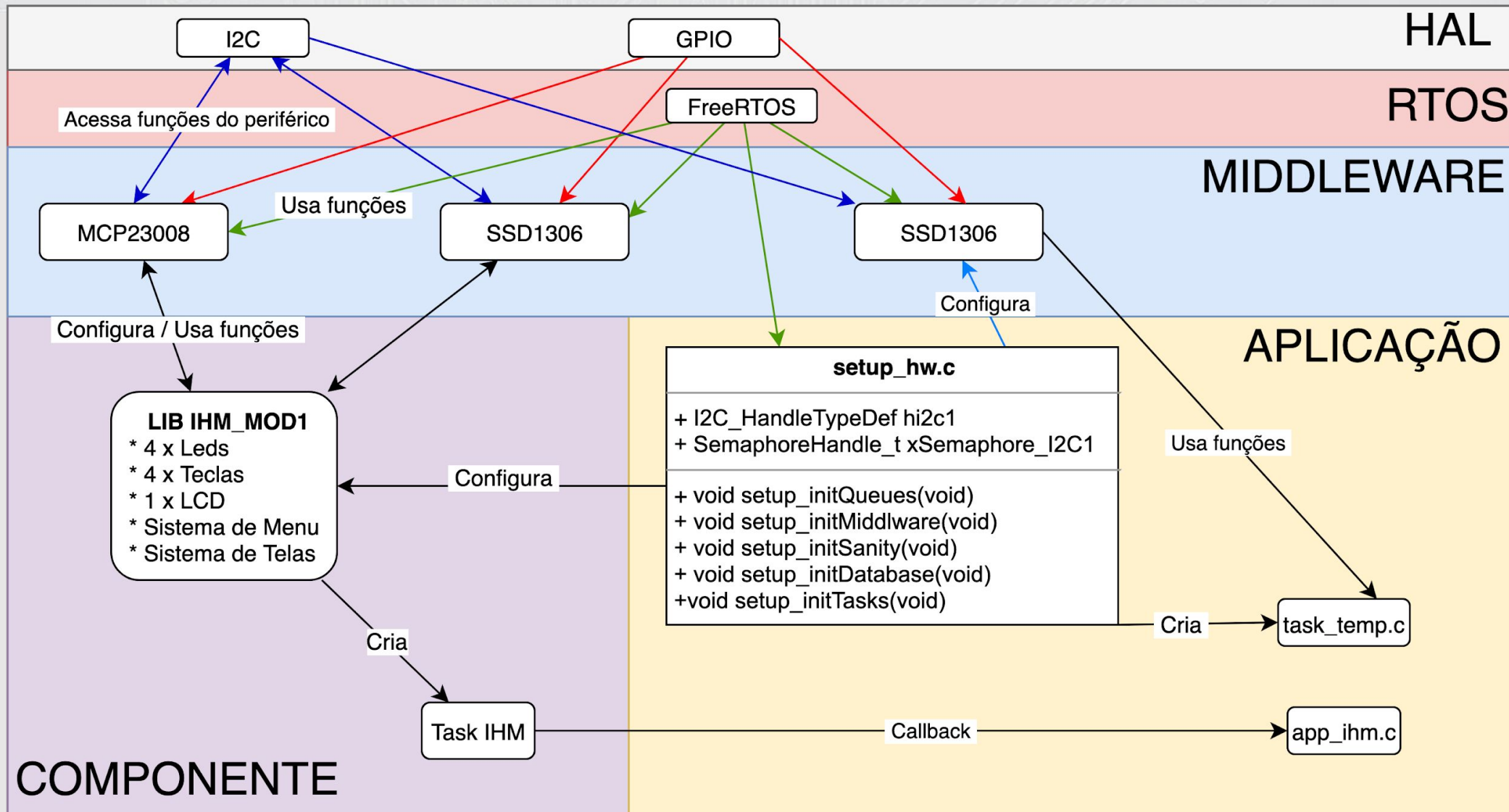


COMPONENTIZAÇÃO CONTINUAÇÃO

- Criar um módulo de IHM com 4 teclas, 4 leds e um display gráfico
- Implementar uma rotina de menu para configurar diversos tipo de parâmetros
- Será possível alterar o idioma dos labes do menu
- Criar uma tela para confirmar toda alteração de um parâmetro



EXEMPLO IHM - CONTINUAÇÃO



EXEMPLO IHM - CONT.

- Arquivo **setup_hw.c** responsável por:
 - Habilitar libs da camada de middleware que serão usadas no projeto
Ex: #define MIDDLEWARE_IHM_MOD1_ENABLED
 - Informar quais periféricos de hardware e variáveis de sincronização serão usados por cada lib da camada de middleware e componentes
 - Inicializar a tarefa da IHM

```
/* Arquivo: setup_hw.c */
#include "main.h"

#define MIDDLEWARE_IHM_MOD1_ENABLED

extern I2C_HandleTypeDef hi2c1;

static void setup_InitMiddleware(void)
{
    ...
    app_IHM_attach(configMINIMAL_STACK_SIZE * 3,
                    &hi2c1,
                    LCD_RESET_GPIO_Port, LCD_RESET_Pin,
                    INT_HMI_GPIO_Port, INT_HMI_Pin, GPIO_PIN_4,
                    xI2CMutex, portMAX_DELAY,
                    TIMEOUT_PRESS_KEY_MS);
    ...
}

static void setup_initTasks(void)
{
    ...
    /* Inicializa task IHM */
    app_IHM_init();
    ...
}

void setup_init(void)
{
    /* Inicializa Rotinas de Queue, Semp, etc */
    setup_initQueues();

    /* Inicializa Drivers usados pelas Libs */
    setup_initMiddleware();

    /* Visualiza causa do Reset */
    setup_initSanity();

    /* Inicializa os parametros de configuracao do equipamento */
    setup_initDatabase();

    /* Inicializa Tasks da Aplicacao */
    setup_initTasks();

    /* Envia mensagem de start do sistema */
    DBG("\r\nInit Program...\r\n");
}
```



EXEMPLO IHM - CONTINUAÇÃO

```
/* Arquivo: task_ihm.c */
void app_IHM_attach(uint16_t size_task,
                    I2C_HandleTypeDef *I2C_InitStruct,
                    GPIO_TypeDef *lcd_reset_gpio, uint16_t lcd_reset_pin,
                    GPIO_TypeDef *io_reset_gpio, uint16_t io_reset_pin, uint16_t io_ISR_pin,
                    SemaphoreHandle_t *i2c_mutex, uint32_t mutex_timeout,
                    uint32_t timeout_pressKey)
{
    if(size_task != 0)
    {
        ihm_size_task = size_task;
    }

    // Initialization display LCD
    SSD1306_attach(SSD1306A_ADDR2, I2C_InitStruct, lcd_reset_gpio, lcd_reset_pin);
    SSD1306_attach_mutex(i2c_mutex, mutex_timeout);

    // Initialization io expensor, Must be called next display lcd
    mcp23008_attach(I2C_InitStruct, io_reset_gpio, io_reset_pin, io_ISR_pin);\
    mcp23008_attach_mutex(i2c_mutex, mutex_timeout);
    mcp23008_init_semaphore(timeout_pressKey);
}

void app_IHM_init(void)
{
    BaseType_t xReturned;
    xReturned = xTaskCreate((TaskFunction_t) vTaskIhmThread, "IHM", ihm_size_task, NULL, 3, &xTaskIhmThread);
    DBG_ASSERT_PARAM(xReturned);
}
```

```
/* Arquivo: task_ihm.c */

#include "setup_hw.h"

#if defined(MIDDLEWARE_IHM_MOD1_ENABLED)

static void task_ihmThread(void const *args)
{
    ihm_init();
    ihm_leds_initVars();
    app_IHM_menu_callback(0);
    vTaskDelete(xHandleIHM);
}

__weak void app_IHM_menu_callback(void *handler)
{
    vTaskDelay(1000);
}

#endif
```

EXEMPLO IHM - CONT.

- Implementado a callback da IHM
- Usando as funções do componente
 - ihm_lcd_setCursor
 - ihm_lcd_BMPFile
 - ihm_leds_set
 - ihm_lcd_on
 - ihm_lcd_off
 - ihm_GetKey
 - + outras funções
- main_menu contém toda a estrutura de menu específica da aplicação

```
/* Arquivo: app_ihm.c */
#include "IHM_MOD1.h"

void app_IHM_menu_callback(void *handler)
{
    uint8_t resp = UI_KEY_ENTER;

    ihm_lcd_setCursor(0, 0);
    ihm_lcd_BMPFile((uint8_t*) LogoImage, 136, 54, IHM_COLOR_BLACK);

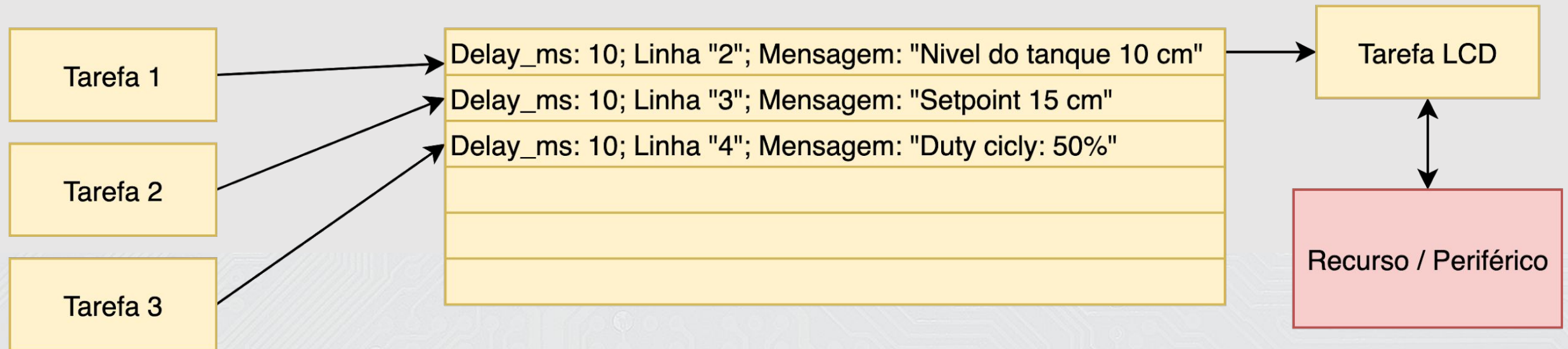
    ihm_lcd_setCursor(0, 50);
    ihm_lcd_print(UI_CNF_LCD_CENTER, 5, Font_7x10, IHM_COLOR_WHITE,
                  0, (uint8_t*)"Mini-CLP %s", version_firmware);

    ihm_leds_set(N_LED1, LED_BLINK_HEARTBEAT);
    ihm_leds_set(N_LED2, LED_OFF);
    ihm_leds_set(N_LED3, LED_OFF);
    ihm_leds_set(N_LED4, LED_OFF);

    for(;;)
    {
        if(resp != UI_KEY_CANCEL && resp != UI_KEY_EXIT_TIMEOUT)
        {
            ihm_lcd_on();
            /* Chama a estrutura de menu feita p/ esta aplicacao */
            main_menu();
            ihm_lcd_off();
        }
        resp = ihm_GetKey(10000);
    }
}
```

GATEKEEPER

- Acesso a um determinado recurso protegido
- Evita inversão de prioridade e deadlocks
- Somente um tarefa tem acesso direto à escrita e leitura de um determinado recurso ou periférico



GATEKEEPER - CONTINUAÇÃO

```
/* app_shell.c */

...
/*COMMAND> lcd test" */
else if (strcmp((const char *)"lcd", (const char *)cmd) == 0)
{
    if(argc == 1)
    {
        if (strcmp((const char *)"-test", (const char *)argv[0])
        {
            /* Linha, mensagem, delay */
            vLCDSendMessage(1, "Linha 1 012345!", 10);
            vLCDSendMessage(2, "Linha 2 abcdef!", 5);
            vLCDSendMessage(3, "Linha 3 ABCDEF!", 3);
        }
    }
}
...

```

```
/* Initialise lcd.c */
static void vLCDTask( void *pvParameters )
{
    xLCDMessage xMessage;
    unsigned portSHORT usRow = 0;

    /* Initialise the hardware. This uses delays so must not be called prior
    to the scheduler being started. */
    prvSetupLCD();

    /* Welcome message. */
    prvLCDPutString( "www.FreeRTOS.org" );

    for( ;; )
    {
        /* Wait for a message to arrive that requires displaying. */
        while( xQueueReceive( xLCDQueue, &xMessage, portMAX_DELAY ) != pdPASS );

        /* Clear the current display value. */
        prvLCDClear();

        /* Switch rows */
        prvLCDGotoRow( xMessage.usRow );
        prvLCDPutString( xMessage.pcMessage );

        /* Delay the requested amount of time to ensure the text just written to the LCD is
        not overwritten. */
        vTaskDelay( xMessage.xMinDisplayTime );
    }
}

```



TEST DRIVEN DEVELOPMENT (TDD)

- **Objetivo**

- Validar e garantir o funcionamento das função ou rotinas implementadas
- Verificar se correções não geraram outros bugs

- **Tipos de testes:**

- Testes unitários
- Testes de integração
- Testes de sistema

- **Libs para teste unitário**

- Linguagem C: Unity, CUnity, **Ceedling**
- Linguagem C++: CppTest, Google Test



TDD - CONTINUAÇÃO



- Executando no Host

- Vantagens:

- Mais recursos de processamento
 - Uso de funções mock para simular o acesso aos periféricos do hardware
 - Exemplo: Lib HAL `gpio_write_io()` substituída por `mock_gpio_write_io`
 - Em alguns casos é possível usar simulador de hardware (Ex: Qemu)
 - É possível usar ferramentas de integração contínua para automatizar testes
 - Exemplo: Jenkins e GitLab CI/CD
 - É possível usar **code coverage** para verificar se os testes estão cobrindo todas as linhas de código

- Desvantagens:

- Problemas de **otimização** de código do compilador não serão detectados
 - Consumo de tempo para escrever os testes ou dar manutenção
 - É difícil dizer que você cobriu todas as rotinas críticas corretamente

TDD - CONTINUAÇÃO



- **Executando no target**

- Vantagens:

- O teste é diretamente no hardware que vai para campo
 - É possível detectar problemas otimização de código do compilador (Ex: gcc ARM)
 - Usando programação orientada a eventos (Ex: gatekeeper) é possível realizar testes de integração

- Desvantagens:

- Você tem muitas restrições (poder de processamento, memória etc.)
 - O teste unitário é mais desafiador

TESTES UNITÁRIOS - EXEMPLO

- Para cada biblioteca desenvolvida, uma rotina de testes precisa ser criada
- É preciso testar tanto o que deve dar certo quanto o que deve dar errado
- Organização da pastas /FRAMEWORK/Middleware/DS1307
 - DS1307.c -> Driver para ler e escrever no RTC DS1307
 - DS1307.h
 - DS1307_cli.c -> Lib com as chamadas de funções via command line interface
 - DS1307_cli.h
 - DS1307_test.c -> Lib com os testes unitários do driver
 - DS1307_test.h

```
/* Arquivo DS1307_test.c */
#include "DS1307_test.h"

#ifdef TEST_MIDDLEWARE_DS1307_ENABLED
#include "unity.h"

static void DS1307_test_RTC_write( void ) {
    DateTime_t dt;
    dt.day = 27;
    dt.month = 7;
    dt.yearInt = 2017;
    dt.yearChar = 17;
    dt.minute = 30;
    dt.hour = 18;
    dt.second = 0;
    dt.weekday = 4;
    TEST_ASSERT_EQUAL_MESSAGE( DS1307_write( dt ), HAL_OK, "Erro ao gravar o horario no RTC" );
}

static void DS1307_test_RTC_read(void) {
    DateTime_t dt;
    TEST_ASSERT_EQUAL_MESSAGE( DS1307_read( &dt ), HAL_OK, "Erro ler o RTC" );
    TEST_ASSERT_EQUAL_MESSAGE( dt.day, 27, "Erro ao lero dia" );
    TEST_ASSERT_EQUAL_MESSAGE( dt.month, 7, "Erro ao ler o mes" );
    TEST_ASSERT_EQUAL_MESSAGE( dt.yearInt, 2017, "Erro ao ler o ano YY" );
    TEST_ASSERT_EQUAL_MESSAGE( dt.yearChar, 17, "Erro ao ler o ano YYYY" );
    TEST_ASSERT_EQUAL_MESSAGE( dt.hour, 18, "Erro ao ler as horas" );
    TEST_ASSERT_EQUAL_MESSAGE( dt.minute, 30, "Erro ao ler os minutos" );
}

void DS1307_UnitTest( void ) {
    UNITY_BEGIN();
    RUN_TEST( DS1307_test_RTC_write );
    RUN_TEST( DS1307_test_RTC_read );
    UNITY_END();
}

#else

__weak void DS1307_UnitTest( void );

#endif
```



CONCLUSÃO

- Conhecer e aplicar design patterns faz toda a diferença
- Criar seu próprio framework possibilita realizar a portabilidade de libs para outros projetos
- Você pode ser um radical que cria os próprios drivers do microcontrolador olhando datasheet e nunca usa a Lib HAL do fabricante
- Testes unitários ajudam no controle de qualidade do código
- Ferramentas de trace podem nos poupar muita dor de cabeça
- Interaja mais com a comunidade de sistemas embarcados
- Crie um projeto open source

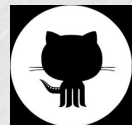




OBRIGADO!



jorge.gzm@gmail.com



github.com/jorgegzm



linkedin.com/in/eng-jorge-guzman