Padrão CMSIS na Programação de Microcontroladores ARM Cortex-M

Common Microcontroller Software Interface Standard (CMSIS)

UFC CAMPUS QUIXADÁ

UNIVERSIDADE FEDERAL DO CEARÁ

PREFÁCIO

- **Disciplina**: Microcontroladores
- **Orientador**: Prof. Dr. Thiago Werlley Bandeira da Silva
- **Equipe**:
  - 427602 - Erick Correia Silva
  - 472012 - Francisco Ítalo de Andrade Moraes
  - 471047 - Pedro Henrique Magalhães Botelho

ÍNDICE

PADRÃO CMSIS

- Common Microcontroller Software Interface Standard.
- Padroniza o acesso ao hardware dos microcontroladores baseados no ARM Cortex-M.
    - Linha **Cortex-M**: Processadores de baixo consumo energético.
- Criado para sanar o problema de portabilidade entre microcontroladores de diferentes fabricantes.
    - Cada Fabricante: Diferentes sistemáticas e bibliotecas.
- Padroniza:
    - A estrutura do código-fonte.
    - A forma de acessar os registradores dos periféricos.
    - O conteúdo dos arquivos de código fonte comum a várias aplicações.
    - O conteúdo dos arquivos de cabeçalho.
    - Funções de acesso aos registradores do núcleo, periféricos e acesso à instruções especiais.

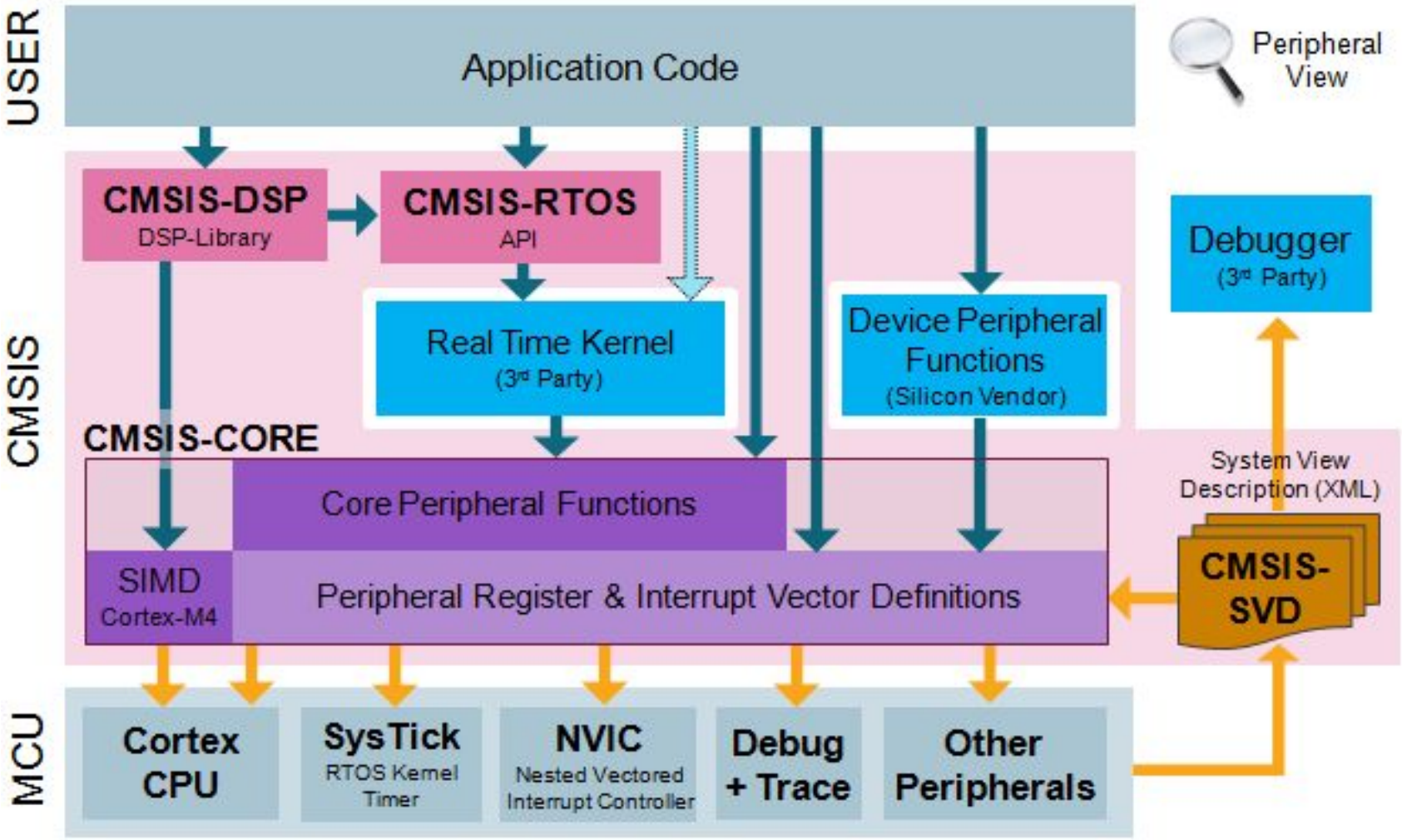
- Em resumo: Uma camada de abstração independente do fabricante para microcontroladores baseados no ARM Cortex-M.
  - Define interfaces genéricas para os dispositivos.
  - Simplifica o reúso de código.
  - Reduz a curva de aprendizado de microcontroladores.
- Provê interfaces para o processador, periféricos, RTOS e componentes de middleware.
- Permite a combinação de componentes de software de múltiplos vendedores.

Middleware: Software que fornece recursos e abstrações para aplicações, como sistemas de arquivo, interfaces gráficas e pilhas de comunicação.

COMPONENTES DO CMSIS

- Os principais componentes do CMSIS são:
  - **CMSIS-CORE**: API para processador e periféricos.
  - **CMSIS-Driver**: Interfaces genéricas de periféricos para *middleware*.
  - **CMSIS-DSP**: Biblioteca com rotinas para processamento de sinais e SIMD usando instruções específicas.
  - **CMSIS-RTOS-API**: API Padrão para RTOS.
  - **CMSIS-Pack**: Uma maneira padrão para gerar arquivos de cabeçalho, etc.
  - **CMSIS-SVD**: Descreve os periféricos de um microcontrolador de uma forma padrão.
  - **CMSIS-DAP**: Firmware padrão para depuração.
- A documentação de cada componente pode ser encontrada no link nas referências.

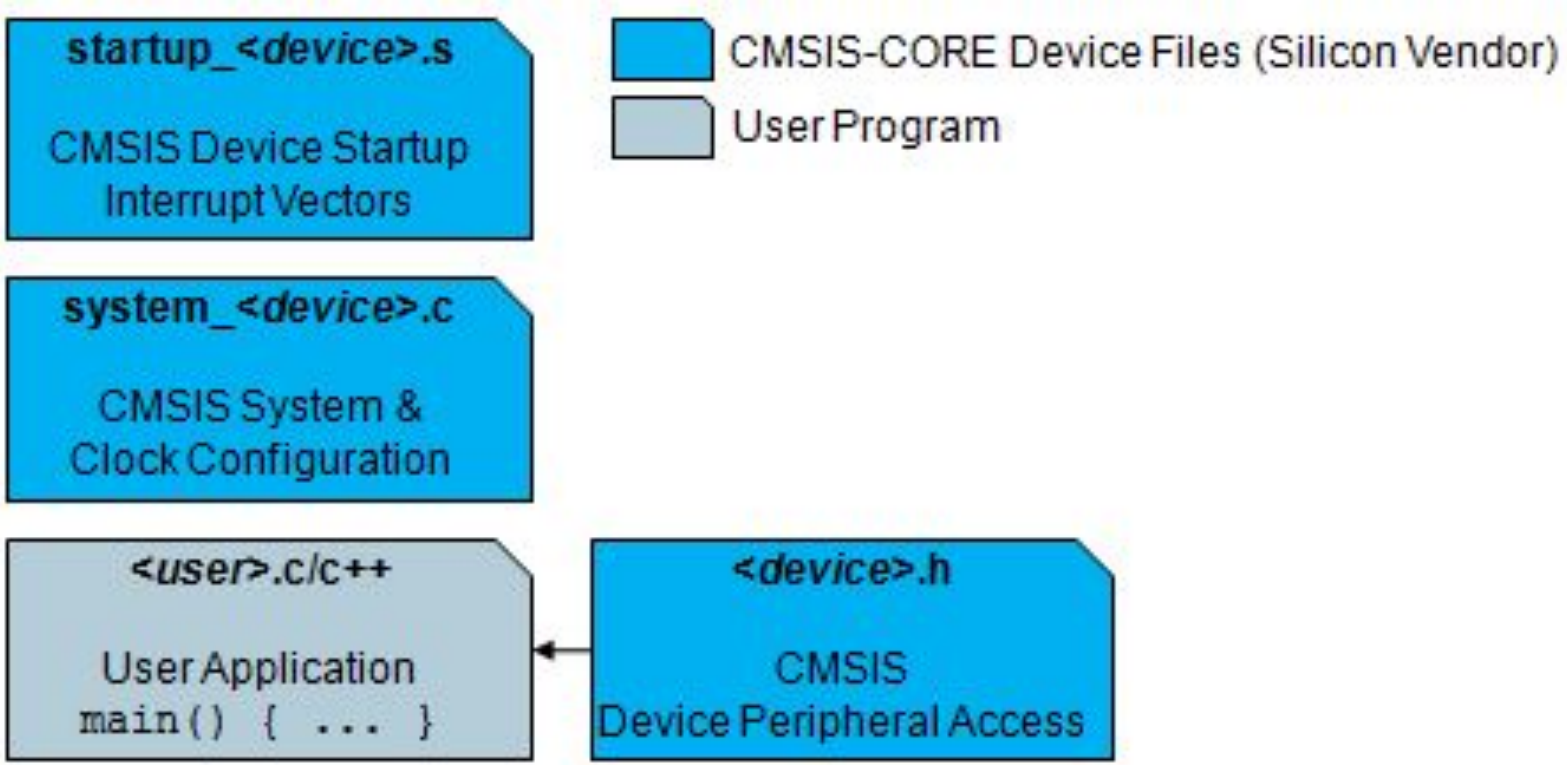
COMPONENTES DO CMSIS



ESTRUTURA DE ARQUIVOS

- Um projeto deve conter os arquivos da aplicação, dentre eles o código principal, no main.c, e os arquivos de configuração, fornecidos pelo fabricante:
  - ***startup_<device>.s***: Código de inicialização do processador:
    - Rotina de Reset
    - Configuração da Tabela de Vetores de Interrupção
    - Configuração da Pilha
  - ***system_<device>*.c** e ***system_<device>*.h:** Rotinas genéricas de configuração do sistema:
    - Inicialização do Sistema, Clock e Barramentos.
  - ***<device>*.h**: Arquivo cabeçalho para o processador.
    - Usado em praticamente todos os arquivos fontes.
    - Define constantes e estruturas para acesso aos registradores da CPU e de periféricos.
    - Inclui bibliotecas padrão fornecidas pela ARM.

- Viemos utilizando a camada CMSIS-CORE na disciplina para interfacear com o núcleo ARM e periféricos do chip.
  - Versão Atual: 5.6.0
- Diagrama dos arquivos principais do padrão:



- Grande ponto é a utilização de estruturas para acessar periféricos.

```
typedef struct {
    uint32_t CTRL;  /* Offset: 0x00 (R/W) */
    uint32_t LOAD;  /* Offset: 0x04 (R/W) */
    uint32_t VAL;   /* Offset: 0x08 (R/W) */
    uint32_t CALIB; /* Offset: 0x0C (R) */
} SysTick_Type;
```

- É realizado então um “cast” do endereço para um ponteiro de struct.

```
/* System Control Space Base Address */
#define SCS_BASE        (0xE000E000UL)

/* SysTick Base Address */
#define SysTick_BASE    (SCS_BASE + 0x0010UL)

/* SysTick struct pointer */
#define SysTick         ((SysTick_Type *) SysTick_BASE)
```

- Portanto, para acessar os registradores do SysTick fazemos:

SysTick->VAL

- Em bibliotecas antigas sem padronização é fornecido uma constante simbólica para cada registrador:

```
#define NVIC_ST_CTRL_R (*((volatile uint32_t *)0xE000E010))
#define NVIC_ST_RELOAD_R (*((volatile uint32_t *)0xE000E014))
#define NVIC_ST_CURRENT_R (*((volatile uint32_t *)0xE000E018))
```

- Várias funções no CMSIS-CORE usam os tipos de dados padronizados.
  - Especificação do C99: Biblioteca **<stdint.h>**

Type	Data
uint32_t	Unsigned 32-bit integer
uint16_t	Unsigned 16-bit integer
uint8_t	Unsigned 8-bit integer

- Define os tipos de interrupção como uma enumeração IRQn:
  - Cada interrupção: Um ISR pré-definido e um número IRQn.
  - Tipos de Exceções:

Exception type	Exception	CMSIS Handler name	CMSIS IRQn enumeration (value)
1	Reset	Reset_Handler	-
2	NMI	NMI_Handler	NonMaskableInt_IRQn (-14)
3	HardFault	HardFault_Handler	HardFault_IRQn (-13)
11	SVC	SVC_Handler	SVCall_IRQn (-5)
14	PendSV	PendSV_Handler	PendSV_IRQn (-2)
15	SysTick	SysTick_Handler	SysTick_IRQn (-1)

- Fornece funções para acessar o NVIC:

Function name	void NVIC_EnableIRQ(IRQn_Type IRQn)
Description	Enable Interrupt in NVIC Interrupt Controller
Parameter	IRQn_Type IRQn specifies the interrupt number (IRQn enum). This function does not support system exceptions.
Return	None

Function name	void NVIC_DisableIRQ(IRQn_Type IRQn)
Description	Disable Interrupt in NVIC Interrupt Controller
Parameter	IRQn_Type IRQn is the positive number of the external interrupt. This function does not support system exceptions.
Return	None

Function name	uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)
Description	Read the interrupt pending bit for a device-specific interrupt source
Parameter	IRQn_Type IRQn is the number of the device-specific interrupt. This function does not support system exceptions.
Return	1 if pending interrupt else 0

Function name	void NVIC_SetPendingIRQ(IRQn_Type IRQn)
Description	Set the pending bit for an external interrupt
Parameter	IRQn_Type IRQn is the number of the interrupt. This function does not support system exceptions.
Return	None

Function name	void NVIC_ClearPendingIRQ(IRQn_Type IRQn)
Description	Clear the pending bit for an external interrupt
Parameter	IRQn_Type IRQn is the number of the interrupt. This function does not support system exceptions.
Return	none

Function name	void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
Description	Set the priority for an interrupt or system exceptions with programmable priority level.
Parameter	IRQn_Type IRQn is the number of the interrupt. unint32_t priority is the priority for the interrupt. This function automatically shifts the input priority value left by 6 bits to put priority value in implemented bits.
Return	None

Function name	uint32_t NVIC_GetPriority(IRQn_Type IRQn)
Description	Read the priority for an interrupt or system exceptions with programmable priority level.
Parameter	IRQn_Type IRQn is the number of the interrupt
Return	uint32_t priority is the priority for the interrupt. This function automatically shifts the input priority value right by 6 bits to remove unimplemented bits in the priority value register.

- Fornece funções para controlar o sistema e configurar o SysTick:

Function name	void NVIC_SystemReset(void)
Description	Initiate a system reset request.
Parameter	None
Return	None

Function name	uint32_t SysTick_Config(uint32_t ticks)
Description	Initialize and start the SysTick counter and its interrupt. This function programs the SysTick to generate SysTick exception for every "ticks" number of core-clock cycles.
Parameter	ticks is the number of clock ticks between two interrupts.
Return	Return 0 if reload value range is valid. Return 1 if reload value is more than 24-bit wide.

Function name	void SystemInit (void)
Description	Initialize the system. Device specific—this function is implemented in system_<device>.c (e.g., system_LPC11xx.c)
Parameter	None
Return	None

Function name	void SystemCoreClockUpdate (void)
Description	Update the SystemCoreClock variable. This function is available from CMSIS version 1.3 and device specific—this function is implemented in system_<device>.c (e.g., system_LPC11xx.c). It should be used every time after the clock settings have been changed.
Parameter	None
Return	None

- Várias outras funções para abstração de hardware:

Function name	Instruction	Descriptions
void __WFI(void)	WFI	Wait-for-Interrupt (sleep)
void __WFE(void)	WFE	Wait-for-Event (conditional sleep)
void __SEV(void)	SEV	Send event
void __enable_irq(void)	CPSIE i	Enable interrupt (clear PRIMASK)
void __disable_irq(void)	CPSID i	Disable interrupt (set PRIMASK)
void __NOP(void)	NOP	No operation
void __ISB(void)	ISB	Instruction synchronization barrier
void __DSB(void)	DSB	Data synchronization barrier
void __DMB(void)	DMB	Data memory barrier

- **Na camada de DSP**: Várias rotinas para aplicações de processamento de sinais, operações com vetores (SIMD), operações de filtros, seno, etc…
- **Na camada de RTOS**: API de abstração para rotinas comuns, como:
  - Gerenciamento de *Tasks*;
  - Semáforos;
  - Queues;

FERRAMENTAS DE SOFTWARE

- Ferramentas Principais:
  - **Compilador**: Responsável por gerar o binário a ser gravado no microcontrolador.
  - **Makefile**: Facilita o processo de compilação, organizando-o.
  - **Gravador**: *Software* responsável por gravar o binário no microcontrolador.
  - **Depurador**: Utilizado para realizar testes no sistema, executando o programa passo-a-passo.
  - **Editor/IDE**: Um ambiente de desenvolvimento completo, podendo facilitar a criação do projeto.
  - **CMSIS**: Caso o chip não forneça uma implementação do CMSIS é possível utilizar os templates da ARM.

- No caso da NXP:
  - O Ambiente Integrado de Desenvolvimento (IDE) fornecido pela NXP, o **MCUXpresso** já fornece todas as ferramentas necessárias, inclusive projetos já padronizados com CMSIS.
  - Portanto todos os detalhes do desenvolvimento são realizados pela IDE, para o programador.
  - O depurador embutido das placas *Freedom* facilita o processo de gravação e depuração, não precisando de um hardware externo.

EXEMPLOS DE CÓDIGO

- *Blink* acessando registradores diretamente:

```c
#include "K32L2B31A.h"

void delay_ms(int ms) {
    for(int i = 0; i < ms; i++) {
        for(int j = 0; j < 7000; j++);
    }
}

int main(void) {
    /* Enable clock for PORTD */
    SIM->SCGC5 |= (1U << 12);

    /* Configure PTD5 as GPIO */
    PORTD->PCR[5] |= (1U << 8);

    /* Configure PTD5 as OUTPUT */
    GPIOD->PDDR |= (1U << 5);

    while(1) {
        /* Toggle the LED level */
        GPIOD->PTOR |= (1U << 5);

        /* Wait 1 second*/
        delay_ms(1000);
    }
}
```

EXEMPLOS DE CÓDIGO

- *Blink* usando interface para GPIO.
- A API SDK, fornecida pela NXP, fornece várias funções para controle de periféricos de maneira padronizada.
- Além de oferecer uma abstração para o periférico, a API deixa o programa modularizado e organizado.

```c
#include "K32L2B31A.h"
#include "fsl_gpio.h"
#include "fsl_port.h"

void delay_ms(int ms) {
    for(int i = 0; i < ms; i++) {
        for(int j = 0; j < 7000; j++);
    }
}

int main(void) {
    /* Enable PORTD module clock */
    CLOCK_EnableClock(kCLOCK_PortD);

    /* Configures the LED pin as GPIO */
    PORT_SetPinMux(PORTD, 5, kPORT_MuxAsGpio);

    /* Configure the LED pin as OUTPUT */
    gpio_pin_config_t led = {kGPIO_DigitalOutput, 0};
    GPIO_PinInit(GPIOD, 5, &led);

    while(1) {
        /* Toggle the LED level */
        GPIO_TogglePinsOutput(GPIOD, 1U << 5);

        /* Wait 1 second*/
        delay_ms(1000);
    }
}
```

EXEMPLOS DE CÓDIGO

- *Blink* usando o temporizador LPTMR:
  - ISR definida em *startup_<device>.h*:

```
WEAK void LPTMR0_IRQHandler(void);
```

UFC CAMPUS QUIXADÁ

UNIVERSIDADE FEDERAL DO CEARÁ

```
#include "K32L2B31A.h"

void LEDS_Init(void);
void LPTMR0_Init(void);

int main(void) {
    LEDS_Init();
    LPTMR0_Init();
    while(1);
}
```

```
void LPTMR0_IRQHandler(void) {
    // CLEAR INTERRUPT FLAG
    LPTMR0->CSR |= (1U << 7);

    // BLINK THE TWO LEDS
    GPIOD->PTOR |= 1U << 5;
}
```

```
void LPTMR0_Init(void) {
    // ENABLE CLOCK FOR LPTMR0
    SIM->SCGC5 |= (1U << 0);

    // BYPASS PRESCALER AND SELECT 1KHz LPO
    LPTMR0->PSR |= (0b01U << 0) | (1U << 2);

    // T = (1/1KHz) = 1ms
    // CMR = 1s/1ms = 1000 COUNTS
    LPTMR0->CMR = 999;

    // ENABLE INTERRUPTS FOR LPTMR0 AND ENABLE IT
    LPTMR0->CSR |= (1U << 0) | (1U << 6);

    // ENABLE INTERRUPTS FOR LPTMR0 IN NVIC
    NVIC->ISER[0] |= (1U << 28);
}
```

```
void LEDS_Init(void) {
    // ENABLE CLOCK FOR PORTD
    SIM->SCGC5 |= 1U << 12;

    // GPIO FUNCTION FOR PTD5
    PORTD->PCR[5] |= 1U << 8;

    // OUTPUT DIRECTION FOR PTD5
    GPIOD->PDDR |= 1U << 5;
}
```

Ao invés de usar o registrador ISER do NVIC podíamos ter usado a função **NVIC_EnableIRQ(LPTMR0_IRQn)**, do CMSIS!

EXEMPLOS DE CÓDIGO

```c
void lcdInit(void) {
    pin_handler_t rs;
    rs.port = pinPORT_E;
    rs.pin = 1;
    gpioPinInit(&rs, gpioOUTPUT);

    pin_handler_t en;
    en.port = pinPORT_E;
    en.pin = 0;
    gpioPinInit(&en, gpioOUTPUT);

    pin_handler_t d4;
    d4.port = pinPORT_E;
    d4.pin = 22;
    gpioPinInit(&d4, gpioOUTPUT);

    pin_handler_t d5;
    d5.port = pinPORT_E;
    d5.pin = 23;
    gpioPinInit(&d5, gpioOUTPUT);

    pin_handler_t d6;
    d6.port = pinPORT_B;
    d6.pin = 20;
    gpioPinInit(&d6, gpioOUTPUT);

    pin_handler_t d7;
    d7.port = pinPORT_E;
    d7.pin = 30;
    gpioPinInit(&d7, gpioOUTPUT);

    lcd.data[0] = d4;
    lcd.data[1] = d5;
    lcd.data[2] = d6;
    lcd.data[3] = d7;
    lcd.rs = rs;
    lcd.en = en;

    lcdInitModule(&lcd);
    NVIC_EnableIRQ(LPTMR0_IRQn);
}
```

```c
#include "MKL46Z4.h"

#include "kl46z/lcd.h"
#include "kl46z/gpio.h"

lcd_handler_t lcd;
```

```c
int main(void) {
    /* Initialize LCD Pins */
    lcdInit();

    /* Setup a message */
    lcdClearDisplay(&lcd);
    lcdSetCursor(&lcd, 0, 0);
    lcdWriteString(&lcd, "CMSIS!");

    while(1);
}
```

UFC CAMPUS QUIXADÁ

UNIVERSIDADE FEDERAL DO CEARÁ

- LCD utilizando API de Abstração.
  - API encontra-se no repositório, em [4].
  - Abstrai o uso de registradores.
  - Cada pino do LCD: Uma *struct* GPIO.
  - *Struct* LCD: Todos os pinos do LCD.

- Um exemplo de fabricante que não segue fielmente o CMSIS é a Atmel.
- Como exemplo temos o microcontrolador **SAM3X8E**, do Arduino Due.
  - A estrutura do código é semelhante, com algumas diferenças.
  - A programação do código pode ser feita no Microchip Studio.
  - A Atmel disponibiliza um conjunto de funções em uma API: **ASF**.
- Exemplo de *Blink*:

```c
#include <asf.h>

#define LED     IOPORT_CREATE_PIN(PIOB, 27)

int main (void) {
    /* INITIALIZE THE SYSTEM CLOCK */
    sysclk_init();
    /* INITIALIZE THE BOARD COMPONENTS */
    board_init();

    /* SELECT IO FUNCTION FOR THE PIN */
    ioport_enable_pin(LED);
    /* SELECT OUTPUT FUNCTION */
    ioport_set_pin_dir(LED, IOPORT_DIR_OUTPUT);

    while(1) {
        /* TOGGLE PIOB_27 LEVEL EVERY 500ms */
        ioport_toggle_pin_level(LED);
        delay_ms(500);
    }
}
```

```c
#include <sam3x8e.h>
#include <pio.h>
#include <delay.h>

#define LED (1U << 27)

int main (void) {
    /* INITIALIZE THE SYSTEM CLOCK */
    sysclk_init();
    /* INITIALIZE THE BOARD COMPONENTS */
    board_init();
    /* SELECT IO FUNCTION FOR THE PIN */
    PIOB->PIO_PER |= LED;
    /* SELECT OUTPUT FUNCTION */
    PIOB->PIO_OER |= LED;
    /* ENABLE WRITING TO PIO_ODSR */
    PIOB->PIO_OWER |= LED;
    while(1) {
        /* TOGGLE PIOB_27 LEVEL EVERY 500ms */
        PIOB->PIO_ODSR ^= LED;
        delay_ms(500);
    }
}
```

REFERÊNCIAS

[1] CMSIS - Arm Developer. Acessado em: 13/07/2022.
[2] O padrão CMSIS - Sergio Prado. Acessado em: 13/07/2022.
[3] Exemplos para o Tiva Launchpad usando CMSIS. Acessado em: 10/07/2022.
[4] Drivers API for K32 and KL43. Acessado em: 13/07/2022.