



UNIVERSIDADE
ESTADUAL DE LONDRINA



Processador ARM Cortex M3/M4

Parte II – Introdução ao Desenvolvimento de Software Embarcado

Prof. Francisco Granziera JR

granziera@uel.br

O que há dentro de um típico microcontrolador ARM?

- Há muitas coisas diferentes dentro de um microcontrolador.
- Em muitos microcontroladores o processador ocupa apenas 10% da área do die, o resto é ocupado por:
 - Memória de programa (flash)
 - SRAM (RAM estática)
 - Periféricos (Timers, USART, I2C, SPI, CAN, etc.)
 - Infraestrutura de barramento
 - Gerador de Clock (incluindo a PLL, reset e distribuição)
 - Regulador de tensão e circuitos de controle
 - Pinos de I/O;
 - Analógicos (ADC, DAC, referência de tensão, etc.)

O que há dentro de um típico microcontrolador ARM?

- Geralmente o projetista não precisa conhecer profundamente o hardware para começar a desenhar um circuito e escrever um software, mas não engane: muitos fabricantes ***vendem facilidades para que não haja dificuldades.***
- Para usar um periférico de um microcontrolador, ***não ceda a tentação de utilizar um código pronto de primeira,*** leia tudo sobre o periférico no Reference Manual do fabricante antes, e veja todos os Application Notes também.

O que há dentro de um típico microcontrolador ARM?

- No ARM, bem como vários outros microcontroladores, os registradores que controlam os periféricos estão acessíveis pelo mapa de memória.
- Para facilitar, os fabricantes fornecem header files e bibliotecas de drivers em C.
- Na maioria das vezes estes arquivos são desenvolvidos com o CMSIS (Cortex Microcontroller Software Interface) que padroniza os headers para acesso aos recursos dos periféricos.

O que é preciso para começar?

Developments Suites

- Existem muitos desenvolvedores de compiladores/IDE (ambiente integrado de desenvolvimento).
- As ferramentas vão de gratuitas, passando por baixo custo até as mais elaboradas e custosas.
- Os principais fornecedores de hoje são:
 - Keil® MDK-ARM
 - ARM® DS-5 (Development Studio 5)
 - IAR Systems (Embedded Workbench for ARM Cortex

O que é preciso para começar?

Developments Suites

- E continua...
 - Red Suite from Cod Red Technologies
 - Mentor Graphics Sourcery CodeBench
 - Mbed.org
 - Altium Tasking VX-toolset for ARM Cortex-M
 - Rowley Associates (CrossWorks)
 - Coocox
 - Texas Instruments Code Composer Studio (CCS)
 - Raisonence RIDE
 - Atollic TrueStudio

O que é preciso para começar?

Developments Suites

- E continua...
 - GNU Compiler Collection (GCC)
 - ImageCraft ICCV8
 - Cosmic Software C Cross Compiler for Cortex-M
 - mikroElektronika microC
 - Arduino
- Alguns kits de desenvolvimento proveem outras linguagens para programação:
 - Oracle Java ME Embedded
 - IS2T MicroEJ Java Virtual Machine
 - mikroElektronika microBasic, microPascal




















O que é preciso para começar?

Placas/Kits de Desenvolvimento

- Devido ao grande número de fabricantes e desenvolvedores existem muitos Kits de Desenvolvimento disponíveis.
- Alguns deles chegam a custar menos de U\$ 12 dólares.
- Até desenvolvedores de software fornecem Kits de desenvolvimento.
- Por meio do software é plenamente possível aprender sobre o microcontrolador por meio de simulação.

O que é preciso para começar?

Developments Suites

		497-13895-ND 	STM32F401C-DISCO	STMicroelectronics	KIT DISCOVERY STM32 F4 SERIES	38 - Immediate	15.43000	1 Non-Stock 	STM32F4	Evaluation Platform	MCU 32-Bit	ARM® Cortex®-M4
		497-15990-ND 	STM32F469I-DISCO	STMicroelectronics	DISCOVERY KIT MCU STM32F469NI	65 - Immediate	61.25000	1	STM32F4	Evaluation Platform	MCU 32-Bit	ARM® Cortex®-M4
		497-15211-ND 	STM32F411E-DISCO	STMicroelectronics	DISCOVERY KIT STM32F411VE MCU	50 - Immediate	15.96000	1	STM32F4	Evaluation Platform	MCU 32-Bit	ARM® Cortex®-M4
		497-13898-ND 	STM32F429I-DISCO	STMicroelectronics	KIT DISCOVERY STM32 F4 SERIES	554 - Immediate	25.54000	1	STM32F4	Evaluation Platform	MCU 32-Bit	ARM® Cortex®-M4
		497-11455-ND 	STM32F4DISCOVERY	STMicroelectronics	EVAL KIT STM32F DISCOVERY	1,954 - Immediate	15.83000	1	STM32F4	Evaluation Platform	MCU 32-Bit	ARM® Cortex®-M4
		497-14652-ND 	STM32F3348-DISCO	STMicroelectronics	KIT DISCOVERY STM32 F3	48 - Immediate	18.09000	1	STM32F3	Evaluation Platform	MCU 32-Bit	-

O que é preciso para começar?

Adaptador para Depuração

- Para efetuar o download do programa para o microcontrolador e para realizar operações de depuração é fundamental uma boa ferramenta, chamada: adaptador de depuração (debug adapter).
- O microcontrolador geralmente possui uma interface de comunicação que deve ser ligada, nos dias de hoje, a interface USB dos PCs (antigamente já foi a COM e a LPT).
- Os geralmente os desenvolvedores das IDEs possuem seus próprios adaptadores.

O que é preciso para começar?

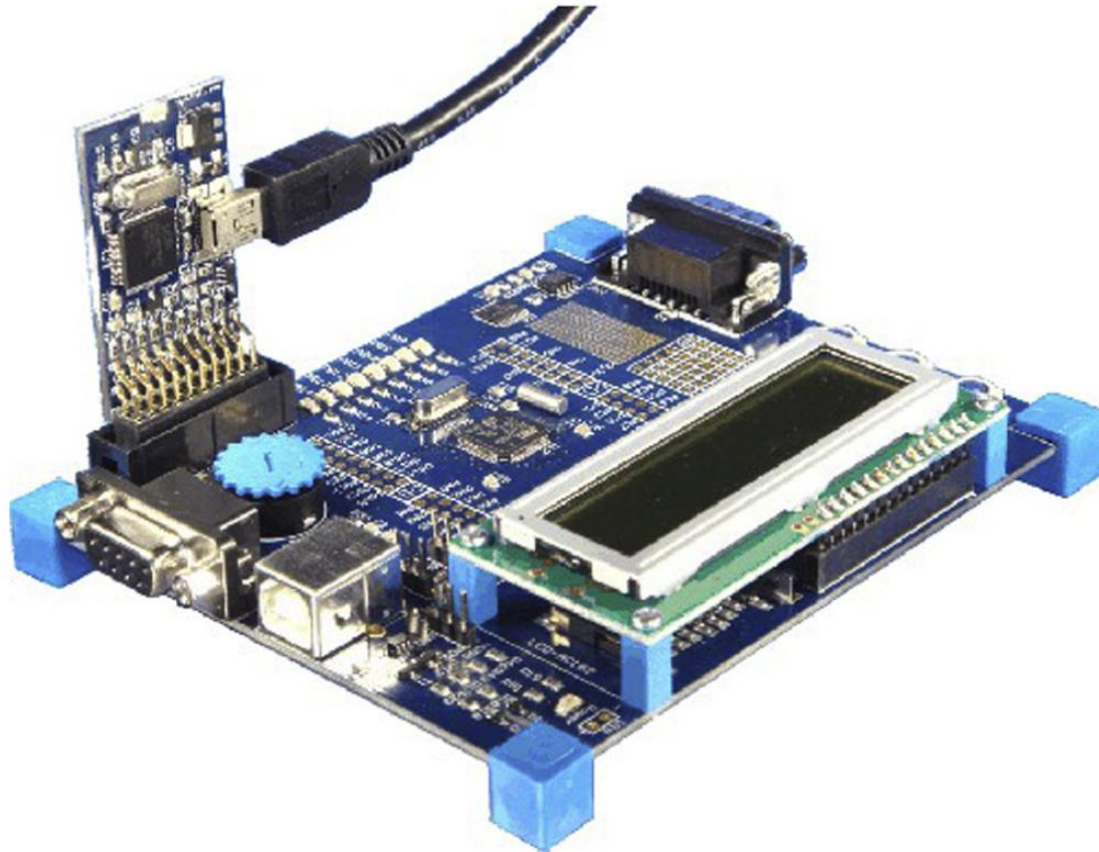
Adaptador para Depuração

- Keil® possui o produto chamado ULINK.
- IAR® fornece o I-Jet.
- Em geral, as Suites também suportam ferramentas de terceiros. Os nomes mudam, mas o propósito da ferramenta é o mesmo.
- USB-JTAG, JTAG/SW Emulator, JTAG In-Circuit Emulator (ICE) são alguns exemplos.
- Alguns DevKits já possuem um adaptador USB na própria placa, como a TI, STM, NXP, e outras.

O que é preciso para começar?

Adaptador para Depuração / DevKits

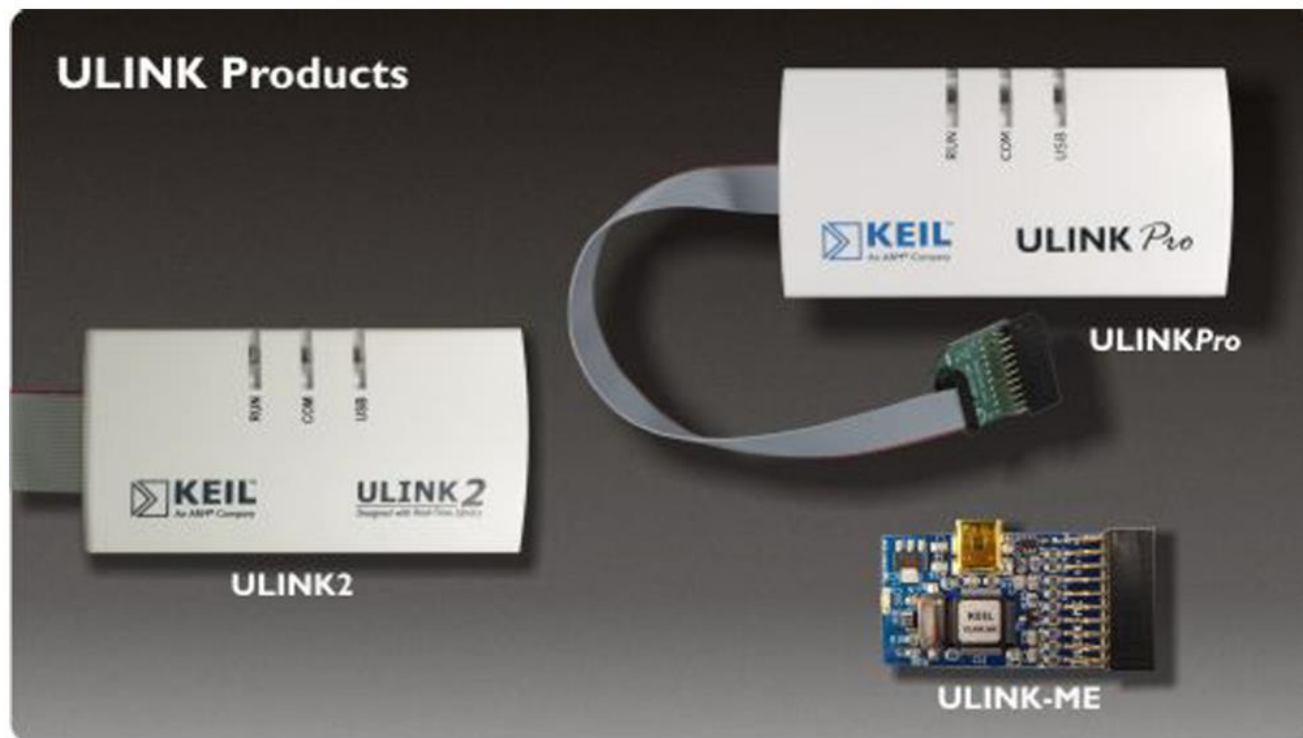
- DevKit Keil:



O que é preciso para começar?

Adaptador para Depuração / DevKits

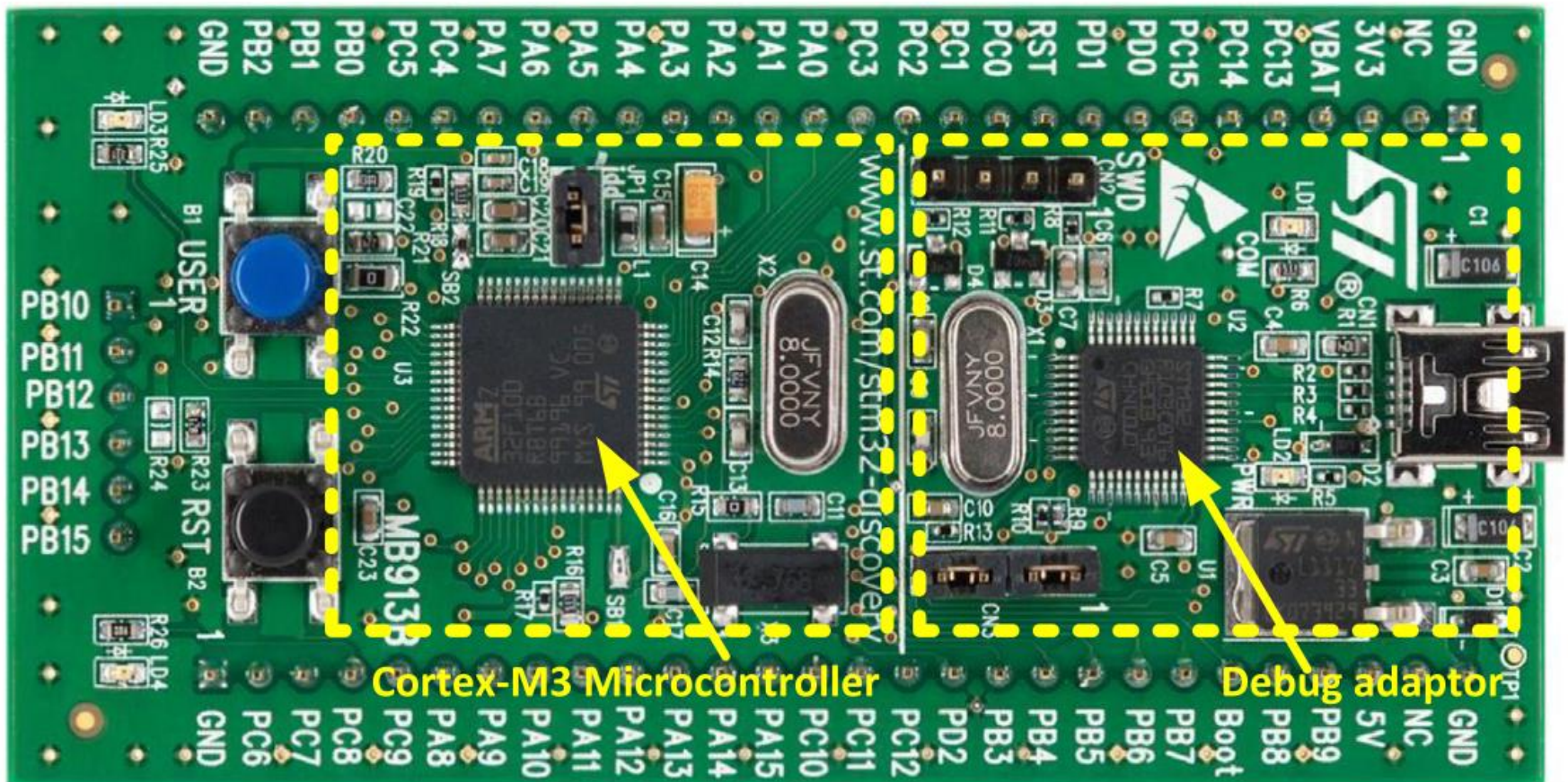
- ULINK Keil:



O que é preciso para começar?

Adaptador para Depuração / DevKits

- STM32Discovery



O que é preciso para começar?

Adaptador para Depuração / DevKits

- Existem versões abertas destes adaptadores: CMSIS-DAP da ARM e o CoLink da Coocox são dois exemplos de adaptadores deste tipo.
- Adaptadores comerciais oferecem muito mais que funções básicas de gravação e depuração.

O que é preciso para começar?

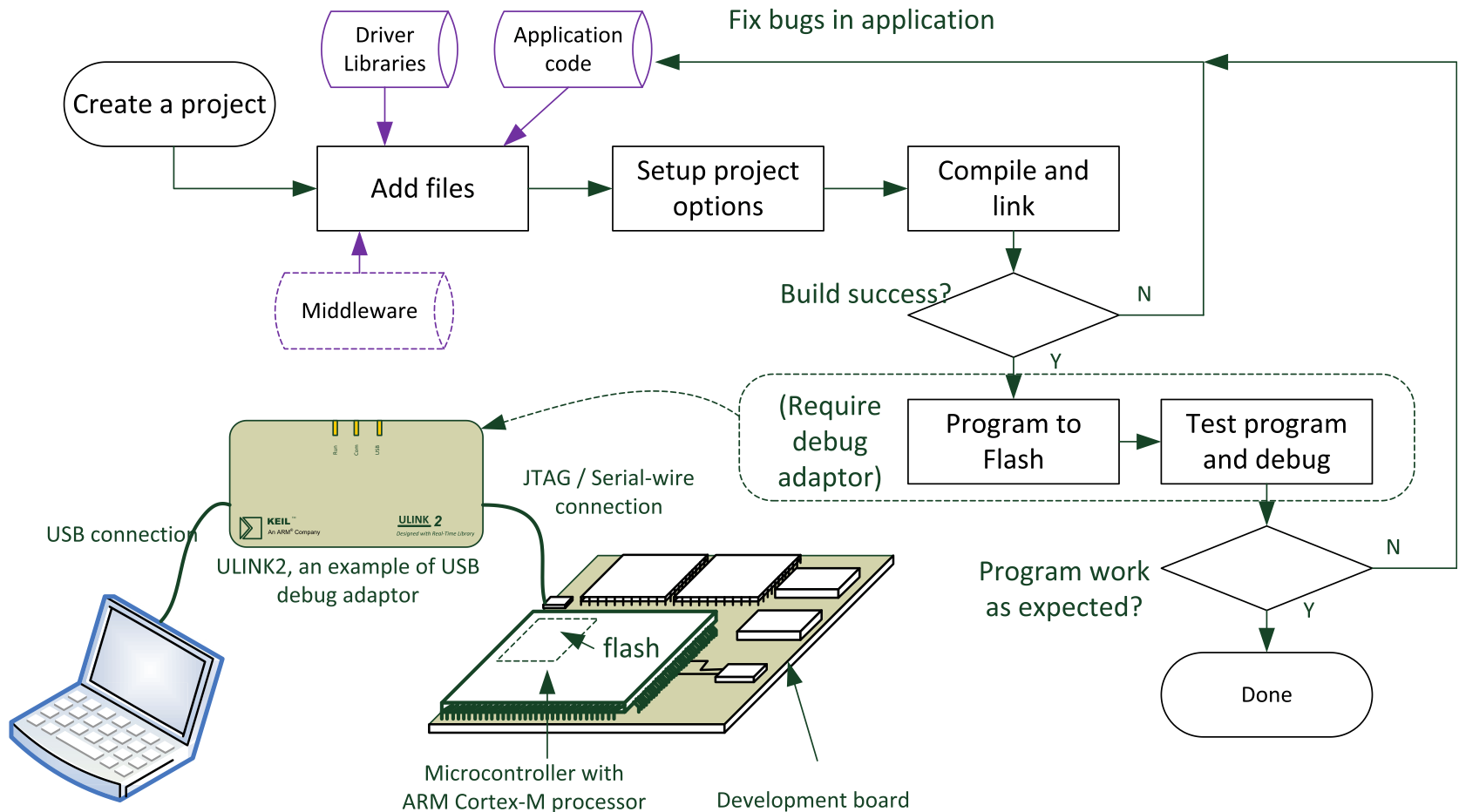
Software Device Drivers

- Device Driver para microcontrolador não tem o mesmo significado de Device Driver para PCs.
- Para os MCs refere-se a Header files e código C com:
 - Definição de registradores dos periféricos;
 - Funções de acesso para configuração e acesso dos periféricos.
- Fabricantes também disponibilizam exemplos de uso dos periféricos para agilizar mais ainda o desenvolvimento das aplicações.

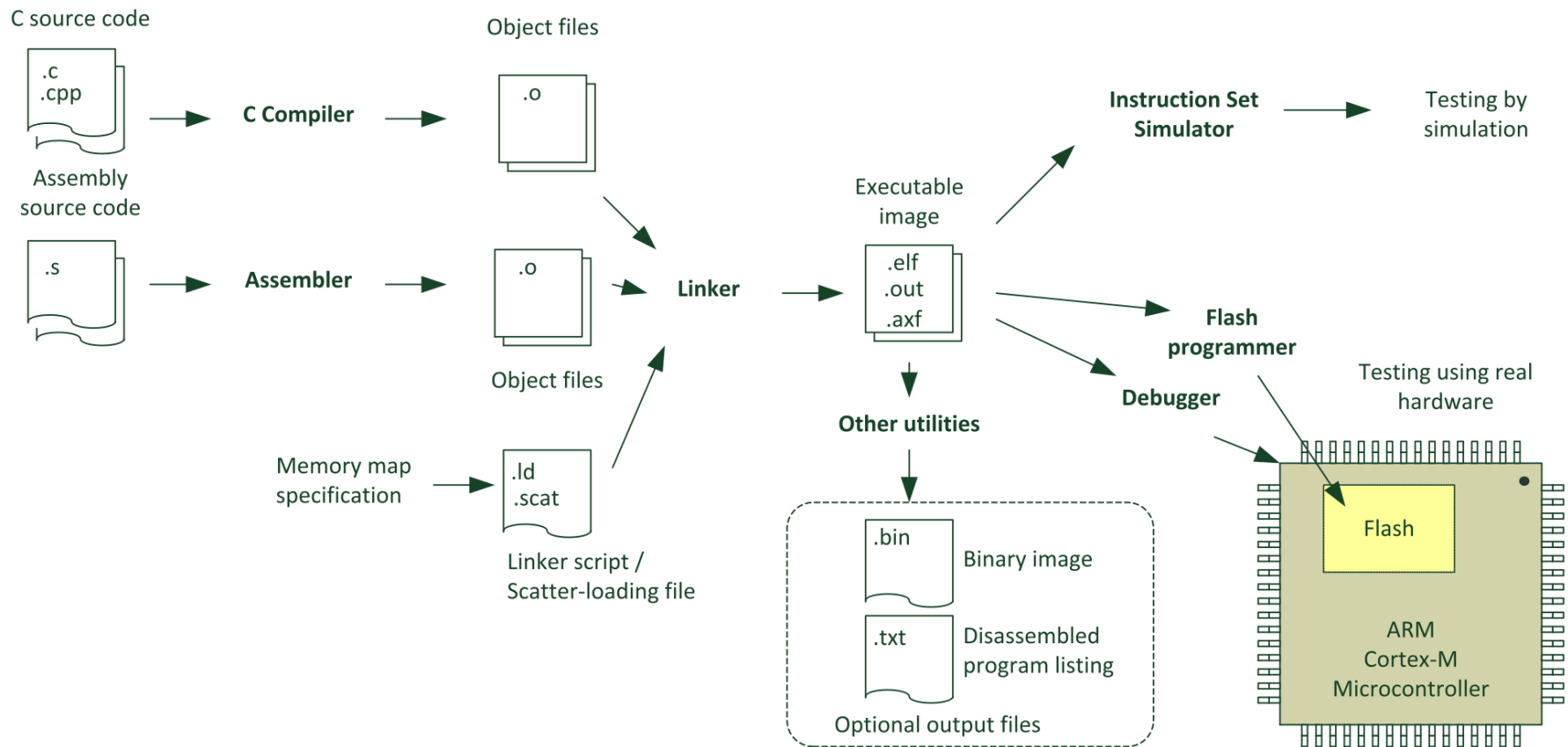
Fluxograma do desenvolvimento de Software

- A maioria das IDEs viabilizam uma sequência de operações necessárias para se desenvolver o programa da aplicação.
- A sequencia básica é resumida como:
 - Criar um projeto
 - Adicionar arquivos ao projeto
 - Configurar o projeto/compilador
 - Compilar/Linkar
 - Programar a Flash
 - Executar o programa e depurar

Fluxograma do desenvolvimento de Software



Fluxograma do desenvolvimento de Software



Ferramentas das Suites de desenvolvimento de Software

Tools	Descriptions
C compiler	To compile C program files into object files
Assembler	To assemble assembly code files into object files
Linker	A tool to join multiple object files together and define memory configuration
Flash programmer	A tool to program the compiled program image to the flash memory of the microcontroller
Debugger	A tool to control the operation of the microcontroller and to access internal operation information so that status of the system can be examined and the program operations can be checked
Simulator	A tool to allow the program execution to be simulated without real hardware
Other utilities	Various tools, for example, file converters to convert the compiled files into various formats

Fluxo de Software

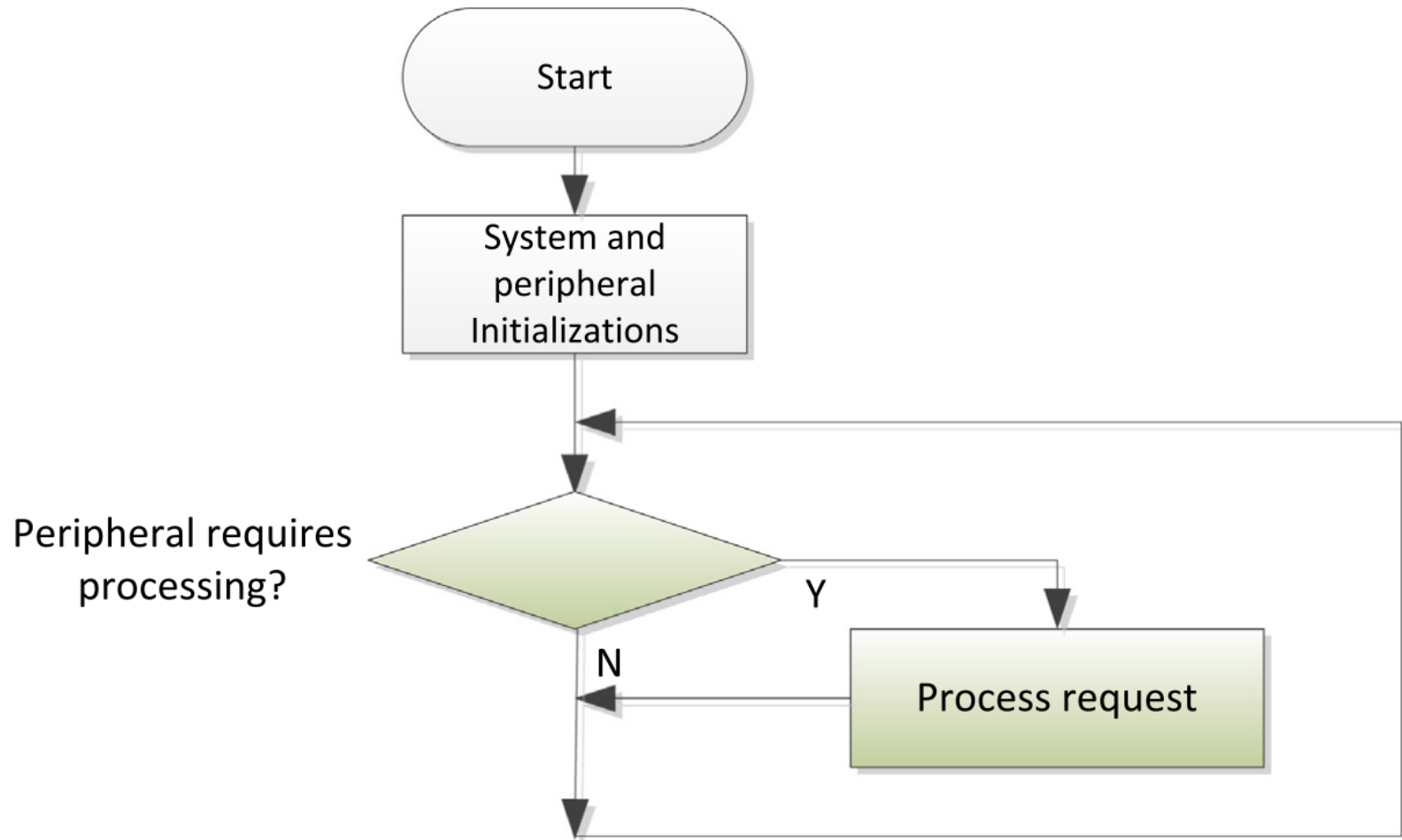
- Na hora de confeccionar um software, quanto mais opções o microcontrolador puder proporcionar, melhor.
- As formas básicas de organizar o software é por polling, interrupção ou multi-tarefa.

Fluxo de Software:

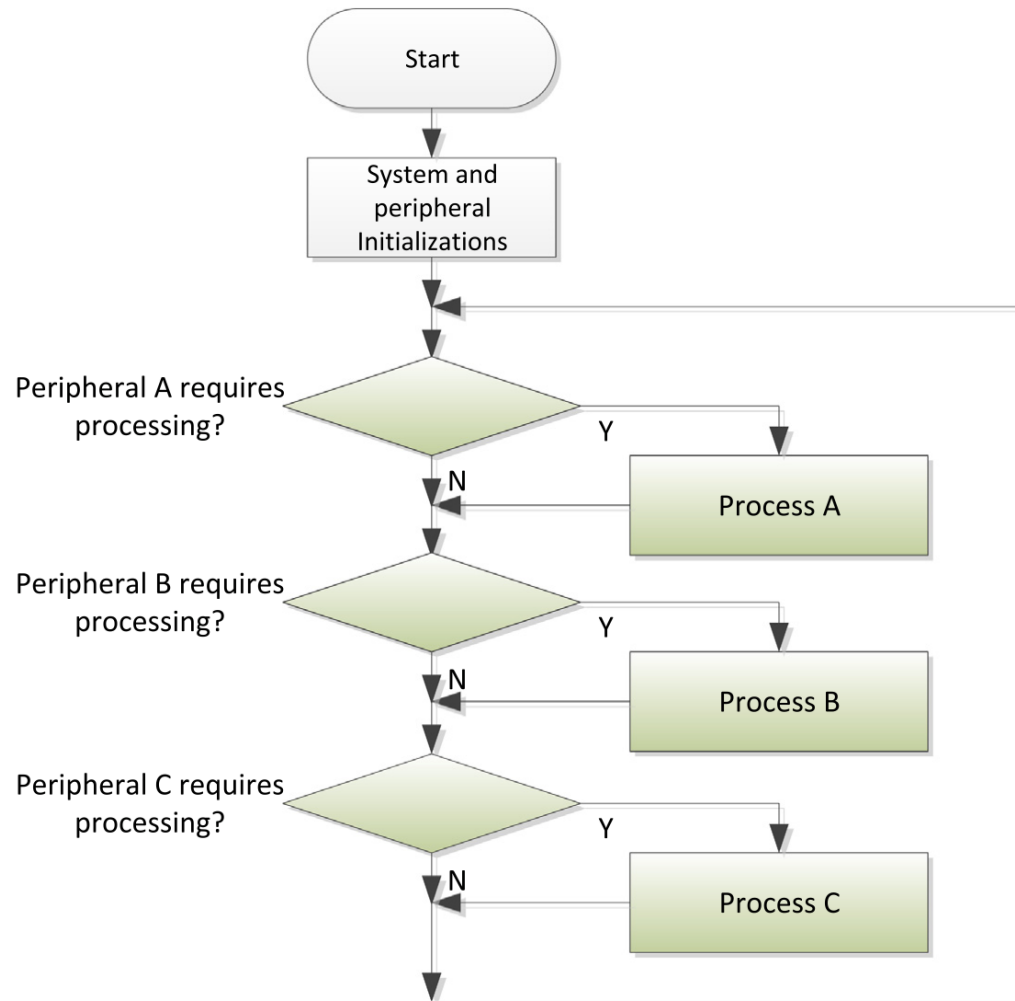
Polling

- Em aplicações muito simples é possível escrever um software onde o processador fique aguardando a informação para realizar a tarefa em questão.
- Por exemplo, um `while(botao_1_press == 1) {}` para segurar o programa até que o botão 1 seja pressionado.
- Note que este método é um tanto ineficiente. Caso o programa tenha 2 ou mais botões para serem monitorados, a solução em polling é um super-loop.

Fluxo de Software: Polling



Fluxo de Software: Polling / Super-loop



Fluxo de Software:

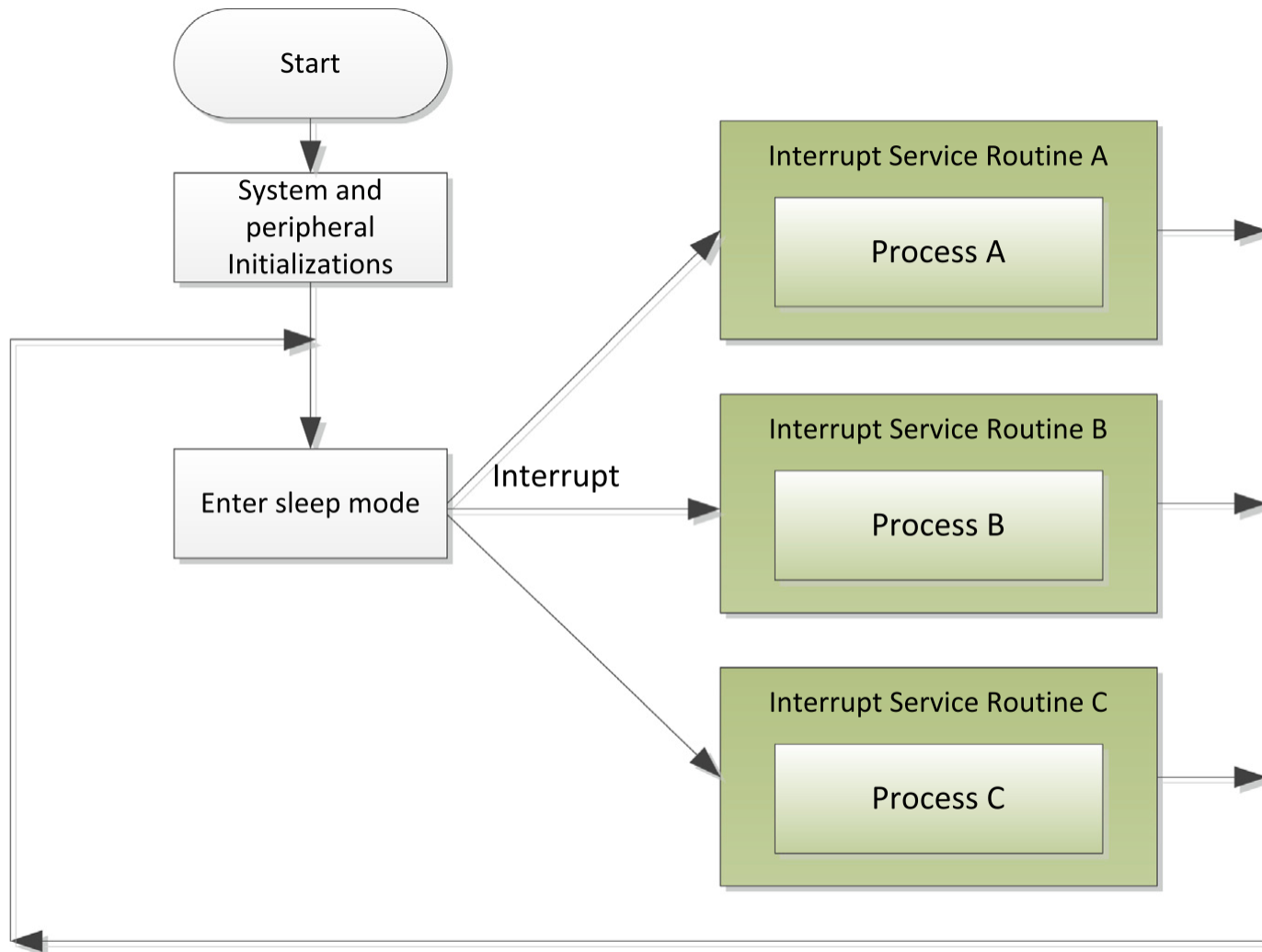
Polling

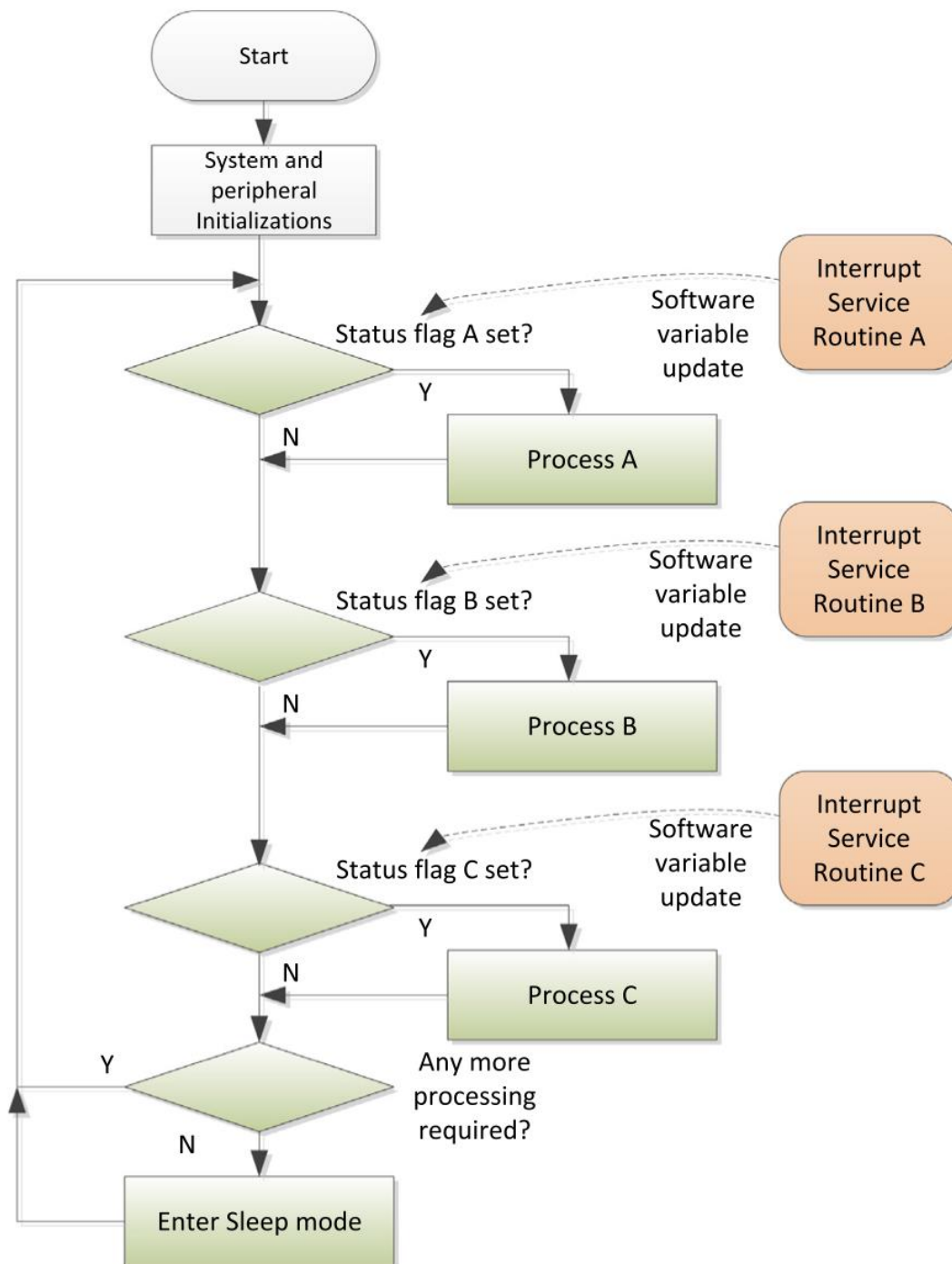
- O método de polling trabalha bem para aplicações simples, sequenciais e periódicas, mas em aplicações de serviços esporádicos (um botão pressionado ou uma tecla que chega pela USART, por exemplo) não é recomendado.
- Aplicações complexas ficam difíceis de manter neste modo.
- Não é possível (difícil) estabelecer prioridades em processos por polling.

Fluxo de Software: por Interrupção

- O método de *polling* também é ineficiente do ponto de vista energético. Manter o processador ligado, em loop, esperando um botão ser pressionado é um tanto desperdicioso.
- Muitos microcontroladores possuem modo *sleep* justamente para hibernar enquanto aguardam algum evento.
- Os periféricos podem produzir interrupções com diferentes prioridades

Fluxo de Software: por Interrupção





Fluxo de Software: multi-tasking system (sistema multi-tarefa)

- Quando o sistema fica mais complexo de forma que polling ou interrupção combinados não conseguem atender, é necessário recorrer à métodos mais sofisticados.
- Um exemplo disso é a execução de tarefas que demoram muito tempo, deixando outras tarefas aguardando. Exemplo: transferir dados da RAM para memória NAND Flash por 500 ms, deixando os botões, interfaces e outros sem checagem.

Fluxo de Software: multi-tasking system (sistema multi-tarefa)

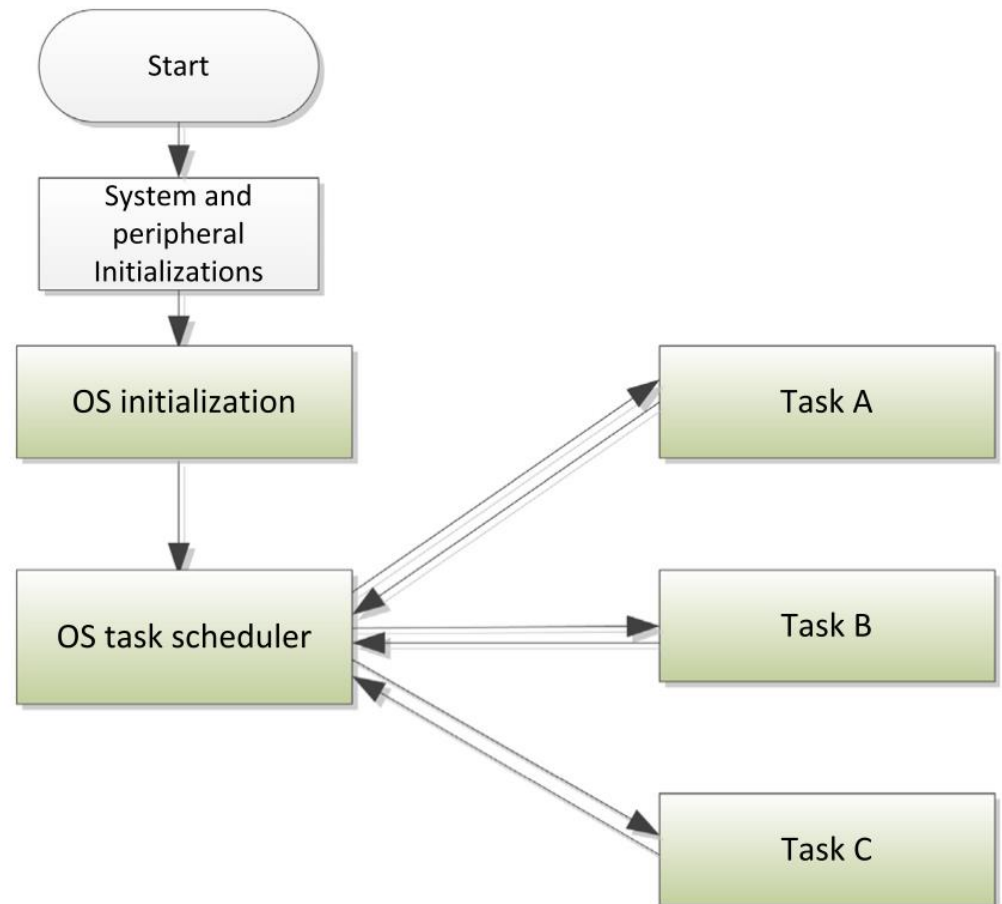
- Para resolver este problema, divide-se o tempo de processamento em slots temporais.
- Cada aplicação roda em um destes slots que podem possuir tamanhos proporcionais a complexidade da tarefa.
- O problema de divisão de tempo de processamento entre as tarefas e escalonamento da mesmas não é tão simples.
- Esta função é deixada para um RTOS.

Fluxo de Software: multi-tasking system (sistema multi-tarefa)

- Um RTOS permite múltiplos processos serem executados concorrentemente, pois divide o tempo de processamento em slots de tempo que são alocados aos processos que requerem os serviços.
- Um timer é necessário para lidar com o tempo dos processos no RTOS.
- O timer gera interrupções que suspende um processo e aloca outro. O escalonador é responsável em decidir se o processo suspenso será trocado por outro e também decide dentre vários processos qual será o próximo.

Fluxo de Software: multi-tasking system (sistema multi-tarefa)

- Além do **Scheduling** o RTOS também é responsável por semáforos, mensagens entre processos, etc.



Tipos de dados em linguagem C

- A linguagem C suporta alguns tipos padrões. Entretanto a forma como será representado no hardware dependerá da arquitetura e do compilador C.
- Exemplo, o tipo **int** pode ser 8 ou 16-bits em microcontrolador de 8-bits, 16-bits em microcontroladores 16-bits, mas será 32-bits em um ARM 32-bits.
- Quanto se porta, ou se aproveita código de um microcontrolador para outro, este ponto é muito importante e deve-se configurar o compilador corretamente.

Tipos de dados em linguagem C

C and C99 (stdint.h) Data Type	Number of Bits	Range (Signed)	Range (Unsigned)
char, int8_t, uint8_t	8	−128 to 127	0 to 255
short int16_t, uint16_t	16	−32768 to 32767	0 to 65535
int, int32_t, uint32_t	32	−2147483648 to 2147483647	0 to 4294967295
Long	32	−2147483648 to 2147483647	0 to 4294967295
long long, int64_t, uint64_t	64	−(2 ⁶³) to (2 ⁶³ - 1)	0 to (2 ⁶⁴ - 1)
Float	32	−3.4028234 × 10 ³⁸ to 3.4028234 × 10 ³⁸	
Double	64	−1.7976931348623157 × 10 ³⁰⁸ to 1.7976931348623157 × 10 ³⁰⁸	
long double	64	−1.7976931348623157 × 10 ³⁰⁸ to 1.7976931348623157 × 10 ³⁰⁸	
Pointers	32	0x0 to 0xFFFFFFFF	
Enum	8 / 16/ 32	Smallest possible data type, except when overridden by compiler option	
bool (C++ only), _Bool (C only)	8	True or false	
wchar_t	16	0 to 65535	

Acesso às entradas/saídas e periféricos

- Microcontroladores possuem várias interfaces tais como GPIO, RTC, SPI, UART, I2C, USB, CAN, Ethernet, ADC, DAC, dentre outras.
- A maioria destas interfaces e periféricos são específicos de cada vendedor, logo é preciso ler extensos manuais para fornecidos pelos vendedores para aprender como usá-los.
- Nestes microcontroladores, os periféricos são mapeados em memória, o que significa que os registradores são acessíveis do sistema do mapa de memória.

Acesso às entradas/saídas e periféricos

- Com intuito de acessar estes registradores de periféricos por meio de programação em C, devemos usar ponteiros.
- Tipicamente um periférico requiere um processo de inicialização antes de ser utilizado.

Configuração dos Periféricos

Sistema de Clock

- Programação do clock para habilitar o clock ao periférico e o clock até os I/O quando necessário. Em microcontroladores modernos o sistema de clock é complexo, pode-se ajustar minuciosamente as frequências e cada periférico ou em um grupo deles e também é possível ligar/desligar os clocks para economizar energia. Por definição, os clocks são desabilitados quando o processador liga, e portanto, é necessário habilitá-los para uso.

Configuração dos Periféricos

Modo de Operação

- Em alguns casos pode ser necessário configurar o modo de operação dos pinos de I/O. Os pinos são multiplexados (várias funções) e pode ser necessário casar a configuração com a operação (entrada, saída, função, etc.). Pode-se também configurar a parte elétrica (tensão, pull up/down, open drain, etc.)

Configuração dos Periféricos

Complexidade versus facilidades

- Para usar um periférico, tal como Timer, I2C ou outro qualquer, é necessário estudá-lo e configurá-lo. Comparando com microcontroladores de 8-bits, os registradores e modos são bem mais complexos e com muitos recursos, por outro lado, os fabricantes sempre fornecem bibliotecas de códigos e exemplos de uso do periférico para ajudar a programação.

Configuração dos Periféricos

Interrupção

- Para usar um periférico por interrupção é necessário habilitar e configurar a interrupção bem como estabelecer uma prioridade para mesma.
- O controlador de interrupção do ARM Cortex M3/M4 é o NVIC, que possui muitos recursos e será visto posteriormente.

Configurações dos Periféricos

Denominação dos Endereços/Nomes

```
/* STM32F 100RBT6B — GPIO A Port Configuration Register Low */  
#define GPIOA_CRL (*((volatile unsigned long *) (0x40010800)))  
/* STM32F 100RBT6B — GPIO A Port Configuration Register High */  
#define GPIOA_CRH (*((volatile unsigned long *) (0x40010804)))  
/* STM32F 100RBT6B — GPIO A Port Input Data Register */  
#define GPIOA_IDR (*((volatile unsigned long *) (0x40010808)))  
/* STM32F 100RBT6B — GPIO A Port Output Data Register */  
#define GPIOA_ODR (*((volatile unsigned long *) (0x4001080C)))  
/* STM32F 100RBT6B — GPIO A Port Bit Set/Reset Register */  
#define GPIOA_BSRR (*((volatile unsigned long *) (0x40010810)))  
/* STM32F 100RBT6B — GPIO A Port Bit Reset Register */  
#define GPIOA_BRR (*((volatile unsigned long *) (0x40010814)))  
/* STM32F 100RBT6B — GPIO A Port Configuration Lock Register */  
#define GPIOA_LCKR (*((volatile unsigned long *) (0x40010818)))
```

Configurações dos Periféricos

Exemplo de uso

Forma comum de acesso aos registradores de um microcontrolador:

```
void GPIOA_reset(void) /* Reset GPIO A */
{
    // Set all pins as analog input mode
    GPIOA_CRL = 0; // Bit 0 to 7, all set as analog input
    GPIOA_CRH = 0; // Bit 8 to 15, all set as analog input
    GPIOA_ODR = 0; // Default output value is 0
    return;
}
```

Configurações dos Periféricos

Denominação dos Endereços/Nomes

- Este método é bom para uso quando há poucos periféricos e poucos registradores. Quando o número de registradores cresce:
 - Para cada definição de registrador o programa precisa armazenar mais uma constante de 32-bits.
 - Quando a múltiplas instâncias de um mesmo periférico, a mesma definição se repete para cada instância. Exemplo: 5 GPIOs com as mesmas configurações.
 - Neste caso não se pode usar a mesma função para configurar todos as 5 GPIOs! É necessário uma pra cada, o que aumenta o código.

Configurações dos Periféricos

Denominação alternativa

- Definir todos os registradores associados a um periférico em uma única **struct** é uma boa solução:

```
typedef struct
{
    __IO uint32_t CRL;
    __IO uint32_t CRH;
    __IO uint32_t IDR;
    __IO uint32_t ODR;
    __IO uint32_t BSRR;
    __IO uint32_t BRR;
    __IO uint32_t LCKR;
} GPIO_TypeDef;
```

Configurações dos Periféricos

Denominação alternativa

- Cada endereço base de um periférico pode então ser definido como ponteiro para uma estrutura de dados:

```
#define PERIPH_BASE ((uint32_t)0x40000000)
    /*!< Peripheral base address in the bit-band region */
...
#define APB2PERIPH_BASE (PERIPH_BASE + 0x10000)
...
```

Configurações dos Periféricos

Denominação alternativa

```
#define GPIOA_BASE  (APB2PERIPH_BASE + 0x0800)
#define GPIOB_BASE  (APB2PERIPH_BASE + 0x0C00)
#define GPIOC_BASE  (APB2PERIPH_BASE + 0x1000)
#define GPIOD_BASE  (APB2PERIPH_BASE + 0x1400)
#define GPIOE_BASE  (APB2PERIPH_BASE + 0x1800)
...
#define GPIOA  ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB  ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC  ((GPIO_TypeDef *) GPIOC_BASE)
#define GPIOD  ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOE  ((GPIO_TypeDef *) GPIOE_BASE)
...
```

Configurações dos Periféricos

Algumas novas definições

__IO

```
#ifndef __cplusplus
#define __I volatile          /*!< defines 'read only' permissions */
#else
#define __I volatile const /*!< defines 'read only' permissions */
#endif
#define __O volatile /*!< defines 'write only' permissions */
#define __IO volatile /*!< defines 'read / write' permissions */
```

Configurações dos Periféricos

Algumas novas definições

Novos tipos:

```
#include <stdint.h> /* Include standard types */  
/* C99 standard data types:  
    uint8_t  : unsigned 8-bit, int8_t   : signed 8-bit,  
    uint16_t : unsigned 16-bit, int16_t  : signed 16-bit,  
    uint32_t : unsigned 32-bit, int32_t  : signed 32-bit,  
    uint64_t : unsigned 64-bit, int64_t  : signed 64-bit  
*/
```


Configurações dos Periféricos

Funções

- Por este método de declarações, pode-se usar uma mesma função para atender várias

```
void GPIO_reset(GPIO_TypeDef* GPIOx)
{
    // Set all pins as analog input mode
    GPIOx->CRL = 0; // Bit 0 to 7, all set as analog input
    GPIOx->CRH = 0; // Bit 8 to 15, all set as analog input
    GPIOx->ODR = 0; // Default output value is 0
    return;
}
```

Configurações dos Periféricos

Funções

- O uso da mesma função para várias instâncias:

```
GPIO_reset(GPIOA); /* Reset GPIO A */  
GPIO_reset(GPIOB); /* Reset GPIO B */  
...
```

Interfaces

- Os aplicativos nos microcontroladores se conectam ao mundo externo por meio das interfaces.
- Estas interfaces são tão simples quanto um simples LED e botão, até mesmo teclados, display 7-seg, display alfanuméricos e gráficos, etc.
- As GUIs em microcontroladores não são tão ricas quanto as de PCs.

Interfaces

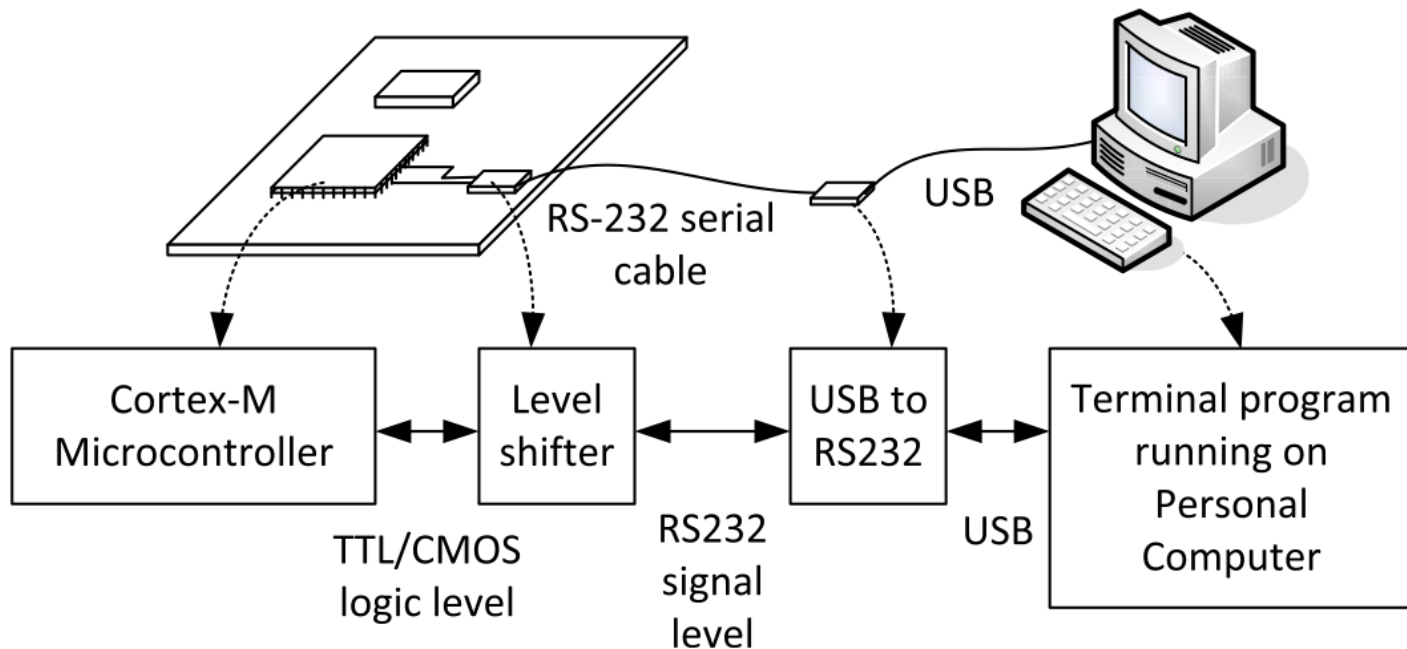
- Ao ler um dado de um ADC em um DevKit, onde mostrá-lo?
 - Caso o Kit possua um display é a opção mais rápida e prática, pois não precisa de conexões externas.
 - Usar a USART para se comunicar com um Terminal no PC.
 - Utilizar o **Instrumentation Trace Macrocell (ITM)**, uma interface padrão de depuração do Cortex-M3/M4, para se comunicar com o software de depuração.

Interfaces

- Embora o LCD seja prático, também é bastante limitado e dados mostrados não ficam armazenados, pois são sobrescritos.
- A interface USART (serial COM) é provavelmente a mais versátil, bem como a interface padrão de Debug.
- Embora não seja padrão, a maioria dos fabricantes incluem esta interface no projeto de seus chips e também são fornecidas bibliotecas em software.
- Os PCs já não possuem mais COMs. Um cabo USB-Serial é necessário para viabilizar a comunicação neste caso. Algumas vezes ainda é necessário um conversor RS-232 para ajustar os níveis de tensão.

Interfaces

- Uso da USART para comunicação com o PC



Interfaces

- Alguns DevKits já incluem um conversor USB-Serial evitando assim o cabo de conversão.
- Alguns microcontrolares possuem interface USB que também pode ser utilizada com uma COM Virtual. É a opção mais econômica, e também a mais complexa.
- Adaptadores de Depuração como ULINK da Keil, possui o ITM que facilita o envio de mensagens.
- O retargeting é um recurso de configurar o `printf()` para imprimir em algum dispositivo de saída que pode ser a USART ou um display.

O CMSIS da ARM

- **Cortex Microcontroller Software Interface Standard (CMSIS)**
- O CMSIS foi desenvolvido pela ARM para permitir a vendedores de software e microcontroladores uma estrutura consistente para desenvolvimento de software.
- Projetos complexos reúnem software desenvolvidos por diferentes programadores, reusos de diferentes projetos anteriores, bibliotecas e drives de vendedores distintos, RTOS livres ou proprietários e protocolos de comunicação também de terceiros.

O CMSIS da ARM

- A padronização criada pelo CMSIS permite:
 - Aumentar a reutilização de software;
 - Melhorar a compatibilidade;
 - Facilitar o aprendizado;
 - Evitar o casamento da ferramenta de debug com o fabricantes de microcontroladores / softwares;
 - Fomentar software livre;

O CMSIS da ARM

- O CMSIS possui 5 categorias:
 - **CMSIS-Core:** um conjunto de APIs para acessar os recursos do Cortex-M independente do microcontrolador, ferramenta ou IDE utilizada.
 - **CMSIS-DSP library:** biblioteca de funções com recursos DSP
 - **CMSIS-SVD:** system view description é um arquivo XML para acessar rapidamente informações dos periféricos do microcontrolador.
 - **CMSIS-RTOS:** uma especificação de API para SO rodando em um Cortex-M
 - **CMSIS-DAP:** debug access port, é um design de referência para construir um Debug Adaptor.

Áreas de padronização do CMSIS-Core

- Padroniza definições para os periféricos do processador.
- Padroniza funções de acesso à recursos do processador.
- Padroniza funções de acesso a instruções especiais.
- Padroniza nomes para uso em funções que tratam exceções.
- Padroniza funções para inicialização do sistema.

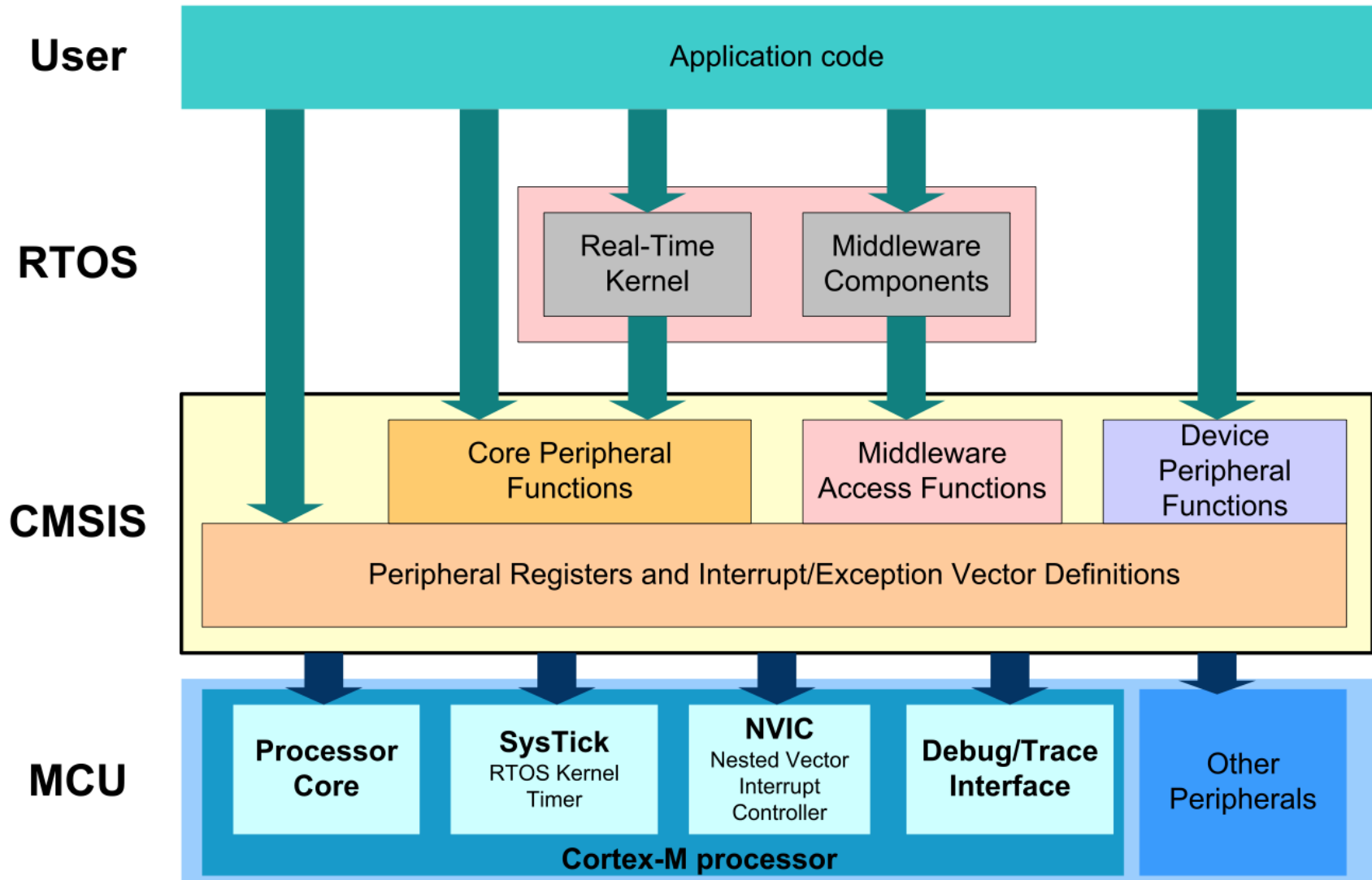
Organização do CMSIS-Core

- Os arquivos CMSIS são incorporados as bibliotecas de device drivers do fabricantes de microcontroladores.
- Alguns destes arquivos são feitos pela ARM e comuns a vários microcontroladores. Também existem arquivos específicos de cada vendedor.
- Em geral, o CMSIS é definido em múltiplos layers.

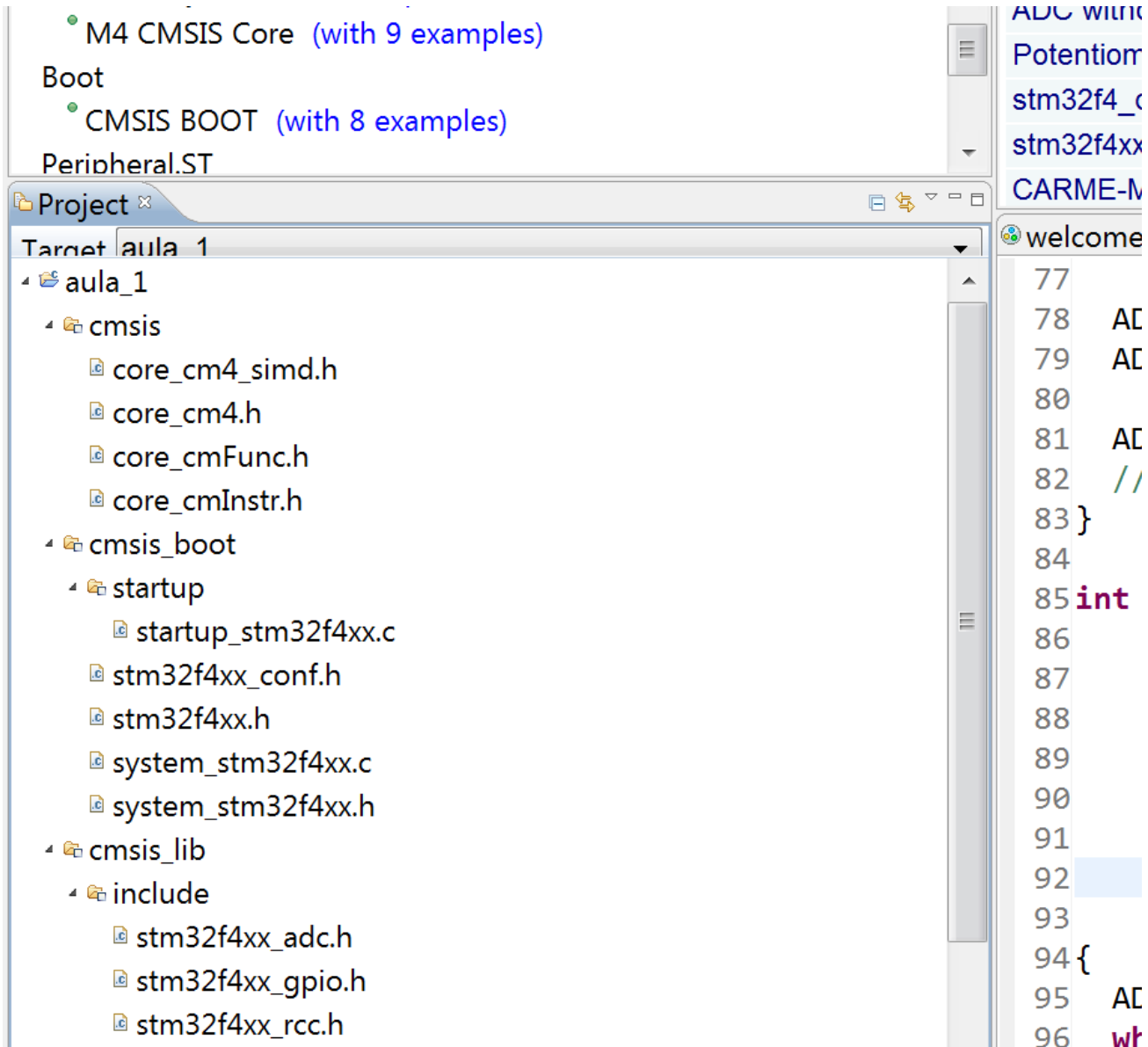
Organização do CMSIS-Core

- Layer de acesso aos periféricos e núcleo.
- Layer de acesso aos dispositivos periféricos.
- Funções de acesso aos periféricos.
- *A padronização diz respeito aos nomes das definições, aos endereços, atribuição de interrupções, etc.*

Estrutura do CMSIS



Uso do CMSIS



The screenshot displays an IDE interface with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'aula_1' with a target 'aula_1'. The project structure is as follows:

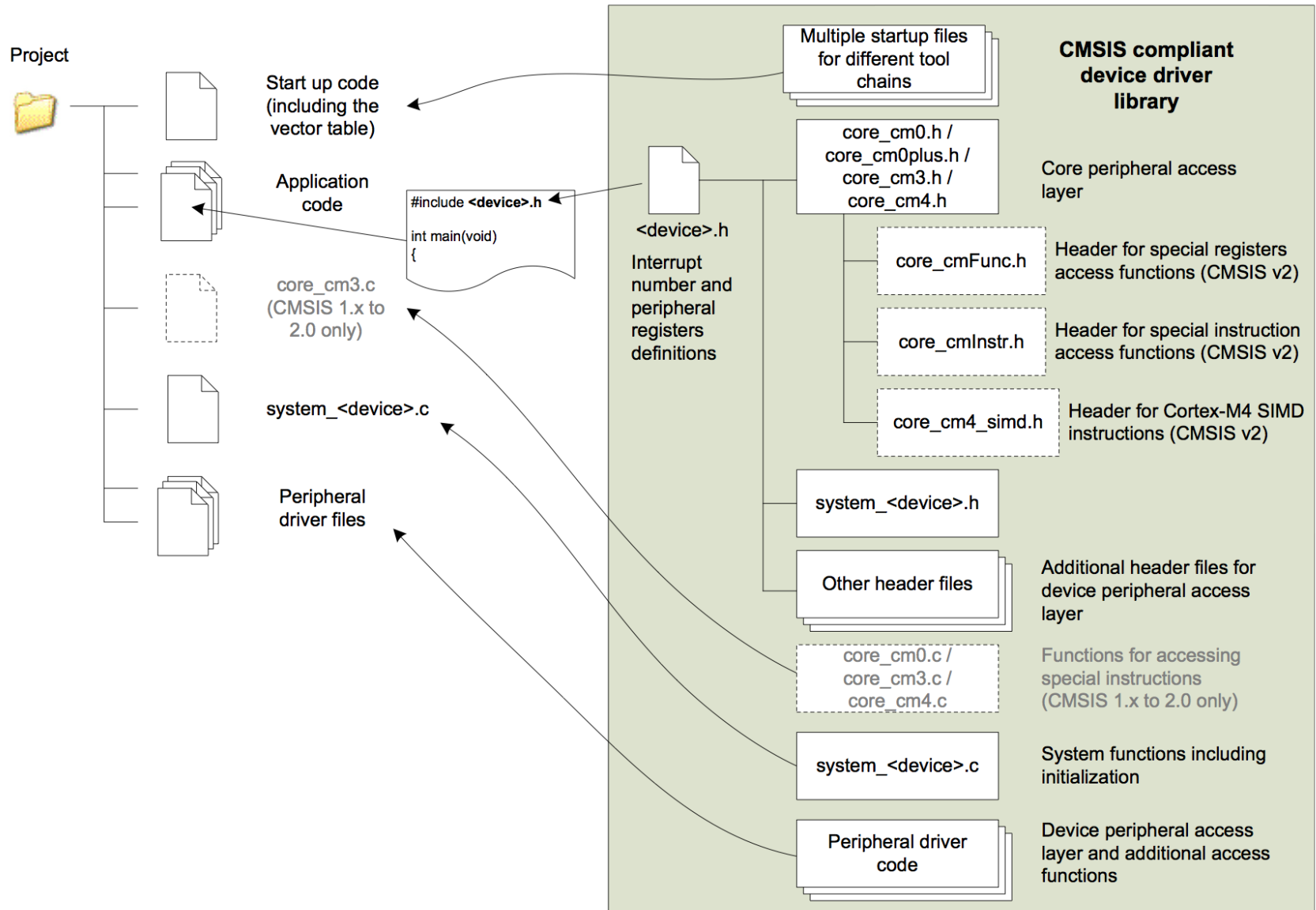
- M4 CMSIS Core (with 9 examples)
 - Boot
 - CMSIS BOOT (with 8 examples)
 - Peripheral.ST

Two red arrows point to the 'cmsis' and 'cmsis_boot' folders in the project explorer.

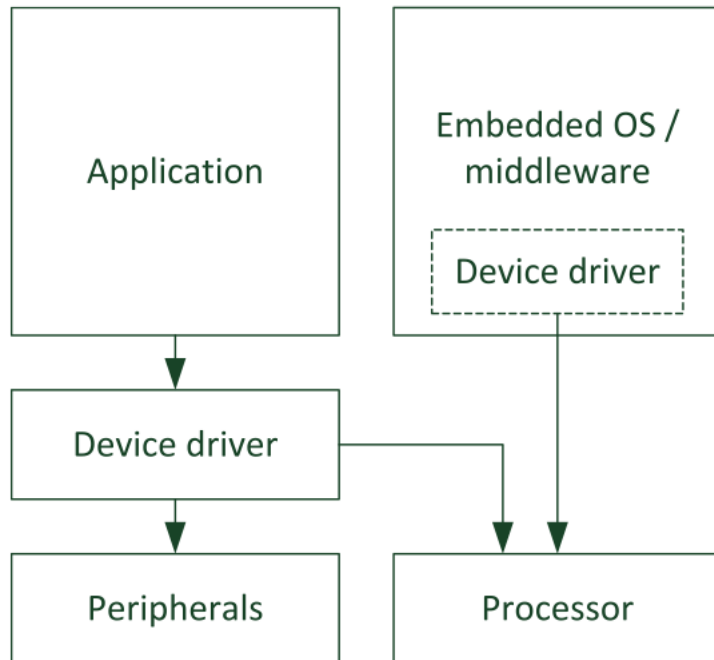
The code editor on the right shows a snippet of C code:

```
welcome
77
78  AC
79  AC
80
81  AC
82  //
83 }
84
85 int
86
87
88
89
90
91
92
93
94 {
95  AC
96  wh
```

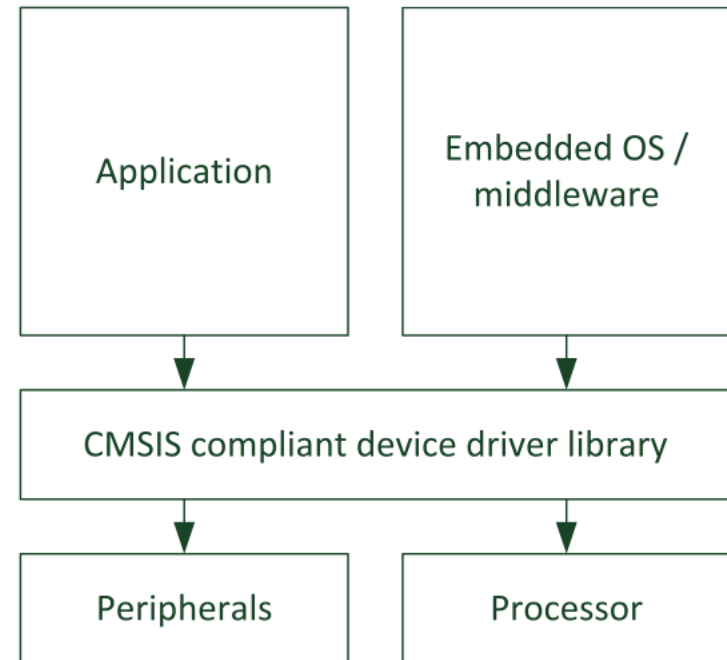
Uso do CMSIS



Com e Sem CMSIS



Without CMSIS, an embedded OS or middleware needs to include access functions to use processor features.



With CMSIS, an embedded OS or middleware can use standardized core access functions from device driver library